

Geant 4

User Application

<http://cern.ch/geant4>

Toolkit + User application

- Geant4 is a **toolkit**
 - i.e. you cannot “run” it out of the box
 - You must write an application, which uses Geant4 tools
- Consequences
 - There are no such concepts as “Geant4 defaults”
 - You must provide the necessary information to configure your simulation
 - You must deliberately choose which Geant4 tools to use
- Guidance: we provide many **examples**
 - **Novice Examples:** overview of Geant4 tools
 - **Advanced Examples:** Geant4 tools in real-life applications

Basic concepts

- What you **MUST** do:
 - Describe your **experimental set-up**
 - Provide the **primary particles** input to your simulation
 - Decide which **particles** and **physics models** you want to use out of those available in Geant4 and the precision of your simulation (cuts to produce and track secondary particles)
- You may also want
 - To interact with Geant4 kernel to **control** your simulation
 - To **visualise** your simulation configuration or results
 - To produce **histograms, tuples** etc. to be further analysed

Interaction with Geant4 kernel

- Geant4 design provides **tools** for a user application
 - To tell the kernel about your simulation configuration
 - To interact with Geant4 kernel itself
- Geant4 tools for user interaction are **base classes**
 - You create **your own concrete class** derived from the base classes
 - Geant4 kernel handles your own derived classes transparently through their base class interface (**polymorphism**)
- **Abstract base classes** for user interaction
 - User derived concrete classes are **mandatory**
- **Concrete base classes** (with **virtual** dummy methods) for user interaction
 - User derived classes are **optional**

User classes

Initialisation classes

- *G4VUserDetectorConstruction*
- *G4VUserPhysicsList*

Action classes

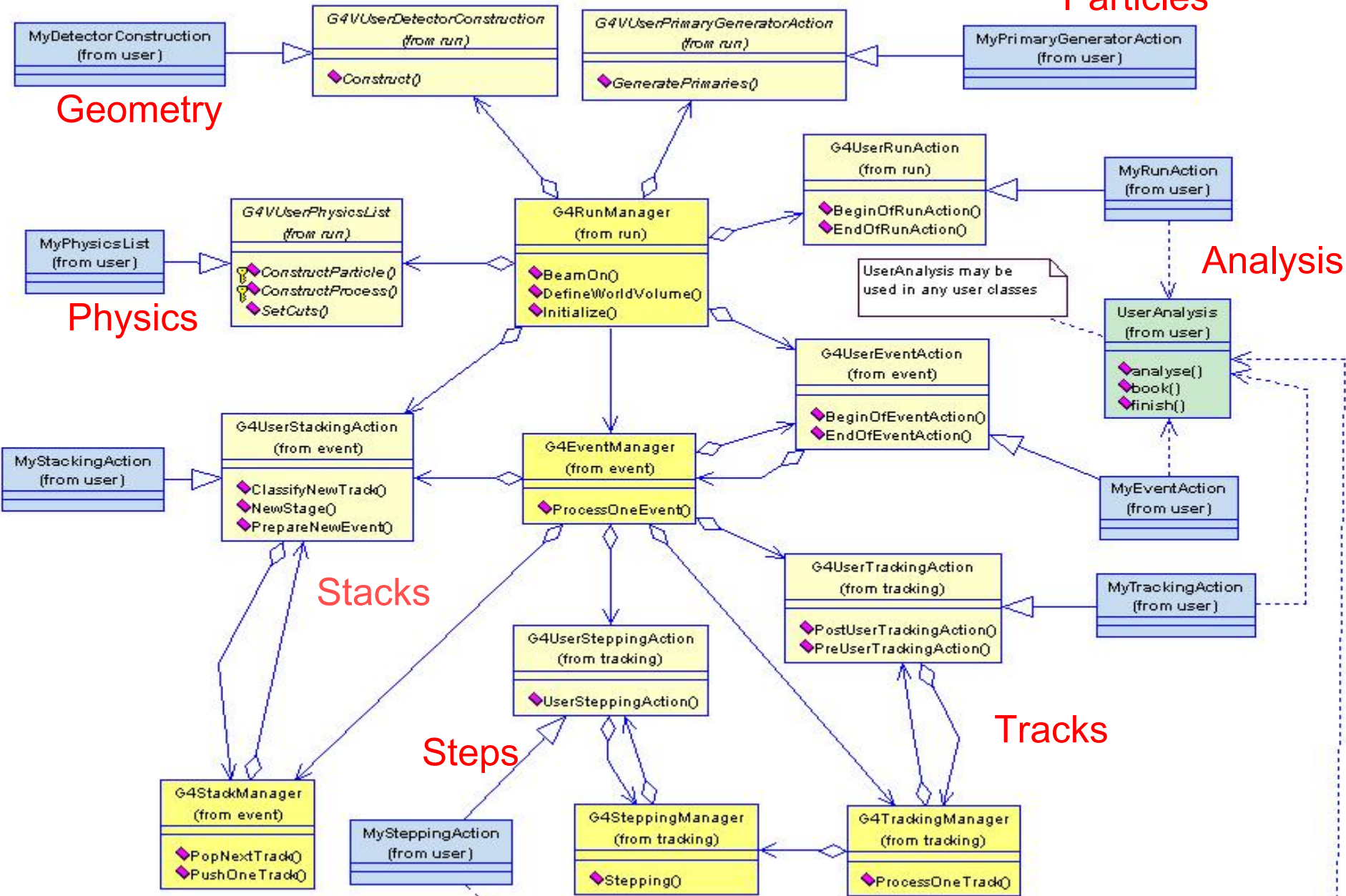
- *G4VUserPrimaryGeneratorAction*
- *G4UserRunAction*
- *G4UserEventAction*
- *G4UserTrackingAction*
- *G4UserStackingAction*
- *G4UserSteppingAction*

Mandatory classes:

- *G4VUserDetectorConstruction*
describe the experimental set-up
- *G4VUserPhysicsList*
select the physics you want to activate
- *G4VUserPrimaryGeneratorAction*
generate primary events

Overview of Geant4 advanced examples

Particles



Development of a Geant4 application

- The following slides provide an overview of the basic concepts of a Geant4 user application
- Your application development will be greatly facilitated, if you adopt a sound **software process**
 - Vision of your simulation, clear user requirements
 - Documented architecture and detailed software design
 - Test process at various levels (unit, integration, system...)
 - Well defined, documented procedures
 - An iterative and incremental process to achieve your goals
 - etc.
- *We will not teach you software process in this course*
 - *(but we could in another course, if you are interested)*

The main function

- Geant4 does not provide the **main()**
 - Geant4 is a toolkit!
 - The main() is part of the user application
- In his/her main(), the user **must**
 - construct **G4RunManager** (or his/her own derived class)
 - notify the G4RunManager mandatory user classes derived from
 - *G4VUserDetectorConstruction*
 - *G4VUserPhysicsList*
 - *G4VUserPrimaryGeneratorAction*
- The user **may** define in his/her main()
 - optional user action classes
 - VisManager, (G)UI session

main()

```
{  
  ...  
  // Construct the default run manager  
  G4RunManager* runManager = new G4RunManager;  
  
  // Set mandatory user initialization classes  
  MyDetectorConstruction* detector = new MyDetectorConstruction;  
  runManager->SetUserInitialization(detector);  
  MyPhysicsList* physicsList = new MyPhysicsList;  
  runManager->SetUserInitialization(myPhysicsList);  
  
  // Set mandatory user action classes  
  runManager->SetUserAction(new MyPrimaryGeneratorAction);  
  
  // Set optional user action classes  
  MyEventAction* eventAction = new MyEventAction();  
  runManager->SetUserAction(eventAction);  
  MyRunAction* runAction = new MyRunAction();  
  runManager->SetUserAction(runAction);  
  ...  
}
```

Describe the experimental set-up

- Derive your own **concrete class** from the ***G4VUserDetectorConstruction*** abstract base class
- Implement the **Construct()** method
 - construct all necessary **materials**
 - define **shapes/solids** required to describe the geometry
 - **construct** and **place volumes** of your detector geometry
 - define **sensitive detectors** and identify detector volumes to associate them to
 - associate **magnetic field** to detector regions
 - define **visualisation** attributes for the detector elements

How to define materials

Different kinds of materials can be defined

Isotopes
Elements
Molecules
Compounds and mixtures

```
PVPhysicalVolume* MyDetectorConstruction::Construct()
```

```
{
```

```
...
```

```
a = 207.19*g/mole;
```

```
density = 11.35*g/cm3;
```

```
G4Material* lead = new G4Material(name="Pb", z=82., a, density);
```

```
density = 5.458*mg/cm3;
```

```
pressure = 1*atmosphere;
```

```
temperature = 293.15*kelvin;
```

```
G4Material* xenon = new G4Material(name="XenonGas", z=54.,  
a=131.29*g/mole, density,
```

```
kStateGas, temperature, pressure);
```

```
...
```

```
}
```

Lead

Xenon
gas

How to define a compound material

For example, a **scintillator** consisting of Hydrogen and Carbon:

```
G4double a = 1.01*g/mole;
```

```
G4Element* H = new G4Element(name="Hydrogen", symbol="H", z=1., a);
```

```
a = 12.01*g/mole;
```

```
G4Element* C = new G4Element(name="Carbon", symbol="C", z=6., a);
```

```
G4double density = 1.032*g/cm3;
```

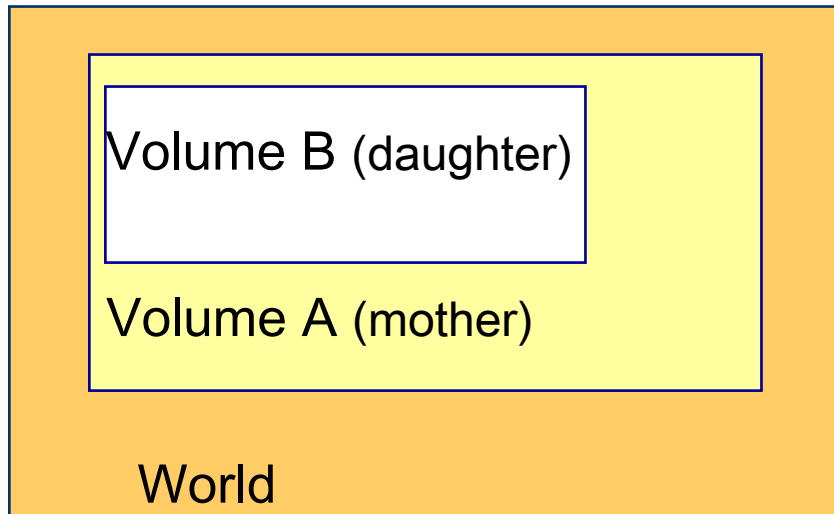
```
G4Material* scintillator = new G4Material(name = "Scintillator", density,  
numberOfComponents = 2);
```

```
scintillator -> AddElement(C, numberOfAtoms = 9);
```

```
scintillator -> AddElement(H, numberOfAtoms = 10);
```

Define detector geometry

- Three conceptual layers
 - **G4VSolid** shape, size
 - **G4LogicalVolume** material, sensitivity, magnetic field, etc.
 - **G4VPhysicalVolume** position, rotation
- A unique physical volume (the **world** volume), which represents the experimental area, must exist and fully contain all other components



e.g.: Volume A is **mother** of Volume B

The mother must contain the daughter volume entirely

Select physics processes

- Geant4 does not have any default particles or processes
- Derive your own **concrete class** from the ***G4VUserPhysicsList*** abstract base class
 - define all necessary particles
 - define all necessary processes and assign them to proper particles
 - define production thresholds (in terms of range)
- Pure virtual methods of G4VUserPhysicsList

ConstructParticles()
ConstructProcesses()
SetCuts()



to be implemented by the user in his/her concrete derived class

PhysicsList: particles and cuts

```
MyPhysicsList :: MyPhysicsList(): G4VUserPhysicsList()
```

```
{  
    defaultCutValue = 1.0*cm;  
}
```

← Define **production thresholds**
(the same for all particles)

```
void MyPhysicsList :: ConstructParticles()
```

```
{  
    G4Electron::ElectronDefinition();  
    G4Positron::PositronDefinition();  
    G4Gamma::GammaDefinition();  
}
```

← Define the **particles**
involved in the simulation

```
void MyPhysicsList :: SetCuts()
```

```
{  
    SetCutsWithDefault();  
}
```

← Set the **production threshold**

PhysicsList: more about cuts

```
MyPhysicsList :: MyPhysicsList(): G4VUserPhysicsList()
```

```
{  
    // Define production thresholds  
    cutForGamma = 1.0*cm;  
    cutForElectron = 1.*mm;  
    cutForPositron = 0.1*mm;  
};
```

```
void MyPhysicsList :: SetCuts()
```

```
{  
    // Assign production thresholds  
    SetCutValue(cutForGamma, "gamma");  
    SetCutValue(cutForElectron, "e-");  
    SetCutValue(cutForPositron, "e+");  
}
```

The user can define
different cuts for
different particles
or
different regions

Physics List: processes

```
void MyPhysicsList :: ConstructParticles()
```

```
{  
  if (particleName == "gamma")  
  {  
    pManager->AddDiscreteProcess(new G4PhotoElectricEffect());  
    pManager->AddDiscreteProcess(new G4ComptonScattering());  
    pManager->AddDiscreteProcess(new G4GammaConversion());  
  }  
}
```

Select physics processes to be activated for each particle type

```
else if (particleName == "e-")
```

```
{  
  pManager->AddProcess(new G4MultipleScattering(), -1, 1,1);  
  pManager->AddProcess(new G4eIonisation(), -1, 2,2);  
  pManager->AddProcess(new G4eBremsstrahlung(), -1,-1,3);  
}
```

The Geant4 *Standard* electromagnetic processes are selected in this example

```
else if (particleName == "e+")
```

```
{  
  pManager->AddProcess(new G4MultipleScattering(), -1, 1,1);  
  pManager->AddProcess(new G4eIonisation(), -1, 2,2);  
  pManager->AddProcess(new G4eBremsstrahlung(), -1,-1,3);  
  pManager->AddProcess(new G4eplusAnnihilation(), 0,-1,4);  
}
```

```
}
```

Primary events

- Derive your own **concrete class** from the *G4VUserPrimaryGeneratorAction* **abstract base class**
- Define primary particles providing:
 - Particle type
 - Initial position
 - Initial direction
 - Initial energy
- Implement the virtual member function **GeneratePrimaries()**

Generate primary particles

```
MyPrimaryGeneratorAction:: My PrimaryGeneratorAction()
```

```
{
```

```
    G4int numberOfParticles = 1;
```

```
    particleGun = new G4ParticleGun (numberOfParticles);
```

```
    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
```

```
    G4ParticleDefinition* particle = particleTable->FindParticle("e-");
```

```
    particleGun->SetParticleDefinition(particle);
```

```
    particleGun->SetParticlePosition(G4ThreeVector(x,y,z));
```

```
    particleGun->SetParticleMomentumDirection(G4ThreeVector(x,y,z));
```

```
    particleGun->SetParticleEnergy(energy);
```

```
}
```

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
```

```
{
```

```
    particleGun->GeneratePrimaryVertex(anEvent);
```

```
}
```

Optional User Action classes

- Five **concrete base classes** whose **virtual member functions** the user may override to gain control of the simulation at various stages
 - G4User**R**unAction
 - G4User**E**ventAction
 - G4User**T**rackingAction
 - G4User**S**tackingAction
 - G4User**S**teppingAction
- Each member function of the base classes has a dummy implementation
 - Empty implementation: does nothing
- The user may implement the member functions he desires in his/her derived classes
- Objects of user action classes must be registered with G4RunManager

Optional User Action classes

G4UserRunAction

- `BeginOfRunAction(const G4Run*)`
 - For example: book histograms
- `EndOfRunAction(const G4Run*)`
 - For example: store histograms

G4UserEventAction

- `BeginOfEventAction(const G4Event*)`
 - For example: perform and event selection
- `EndOfEventAction(const G4Event*)`
 - For example: analyse the event

G4UserTrackingAction

- `PreUserTrackingAction(const G4Track*)`
 - For example: decide whether a trajectory should be stored or not
- `PostUserTrackingAction(const G4Track*)`

Optional User Action classes

G4UserSteppingAction

- UserSteppingAction(const G4Step*)
 - For example: kill, suspend, postpone the track
 - For example: draw the step

G4UserStackingAction

- PrepareNewEvent()
 - For example: reset priority control
- ClassifyNewTrack(const G4Track*)
 - Invoked every time a new track is pushed
 - For example: classify a new track (priority control)
 - Urgent, Waiting, PostponeToNextEvent, Kill
- NewStage()
 - Invoked when the Urgent stack becomes empty
 - For example: change the classification criteria
 - For example: event filtering (event abortion)

Select (G)UI and visualisation

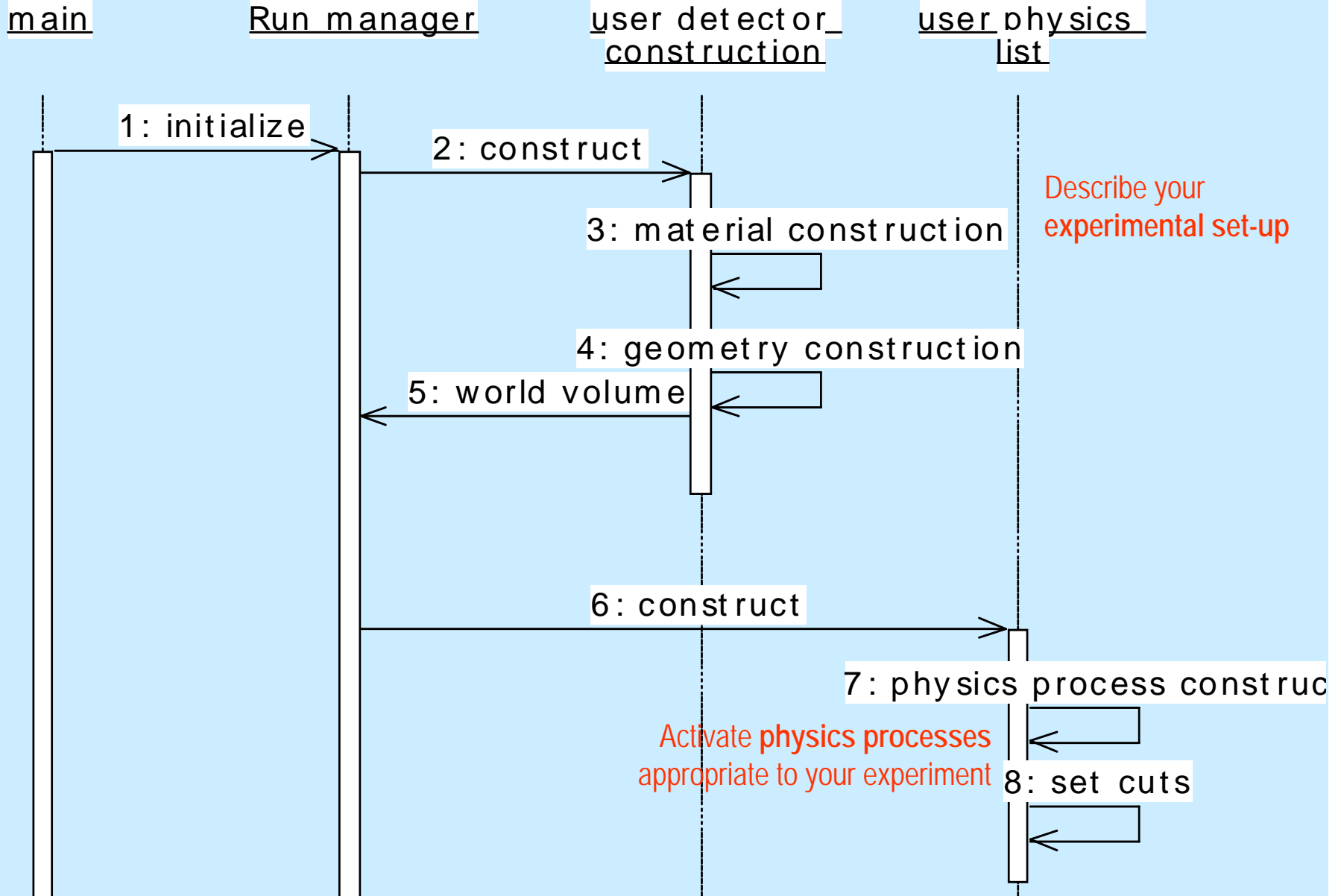
- In your **main()**, taking into account your computer environment, **instantiate a G4UIsession concrete class** provided by Geant4 and invoke its **sessionStart()** method
- Geant4 provides:
 - G4UItterminal
 - csh or tcsh like character terminal
 - G4GAG
 - tcl/tk or Java PVM based GUI
 - G4Wo
 - Opacs
 - G4UIBatch
 - batch job with macro file
 - ...
- In your **main()**, taking into account your computer environment, **instantiate a G4VisExecutive** and invoke its **initialize()** method
- Geant4 provides interfaces to various graphics drivers:
 - DAWN (*Fukui renderer*)
 - WIRED
 - RayTracer (*ray tracing by Geant4 tracking*)
 - OPACS
 - OpenGL
 - OpenInventor
 - VRML
 - ...

Recipe for novice users

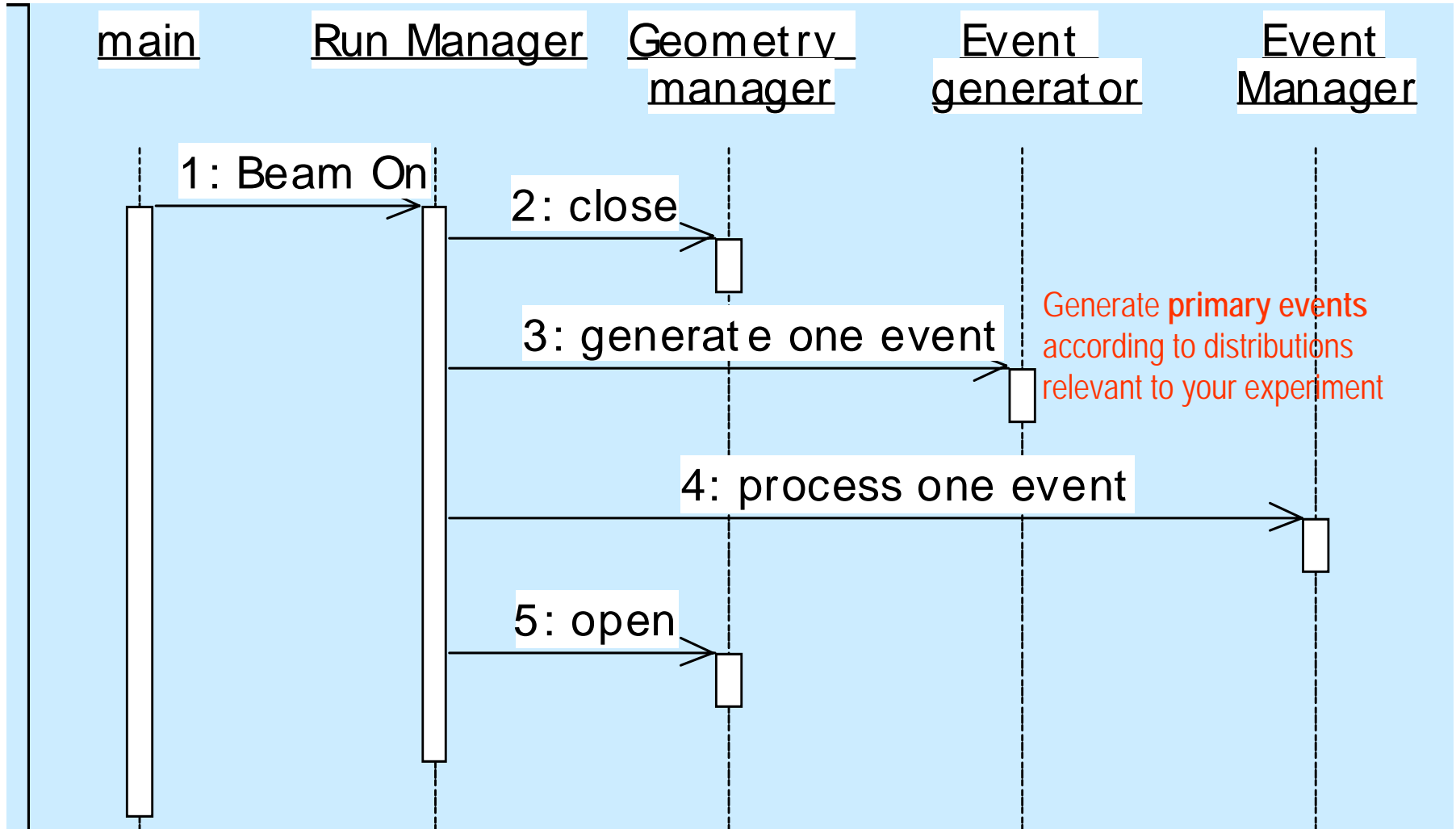
Experienced users may do much more, but the conceptual process is still the same...

- Design diagram as in generic Geant4 Advanced Example
- Create your derived mandatory user classes
 - **MyDetectorConstruction**
 - **MyPhysicsList**
 - **MyPrimaryGeneratorAction**
- Optionally create your derived user action classes
 - **MyUserRunAction**
 - **MyUserEventAction**
 - **MyUserTrackingAction**
 - **MyUserStackingAction**
 - **MyUserSteppingAction**
- Create your main()
 - Instantiate G4RunManager or your own derived MyRunManager
 - Notify the RunManager of your mandatory and optional user classes
 - Optionally initialize your favourite User Interface and Visualization
- **That's all!**

Initialisation



Beam On



Event processing

