



Optional User Actions

Geant4 PHENIICS & IN2P3 Tutorial,

16 – 20 May 2022,

Orsay

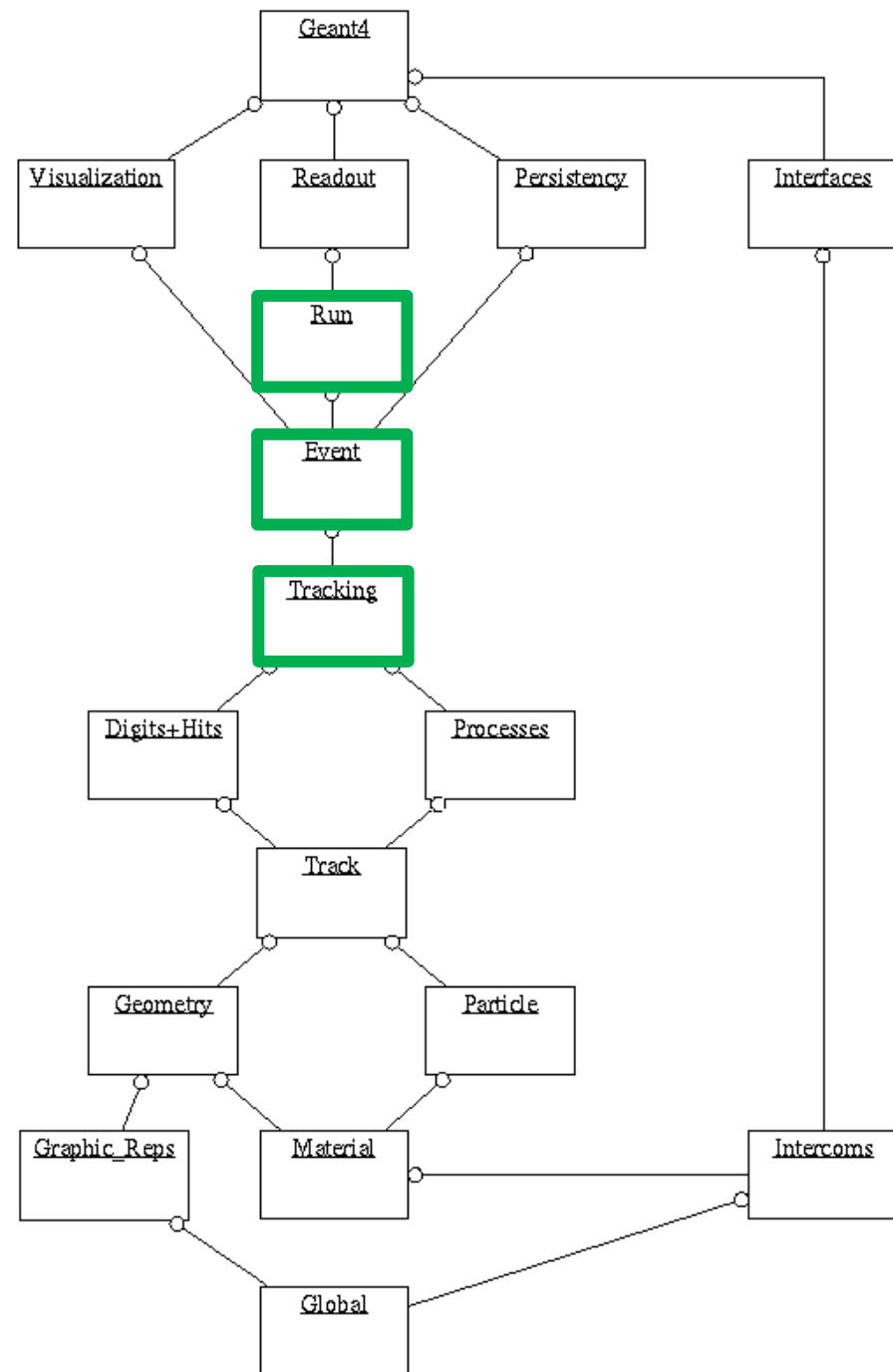
Marc Verderi

LLR, Ecole polytechnique

Where will we look in the toolkit ?

Main categories and directories involved:

- Run
 - `geant4/source/run`
- Event
 - `geant4/source/event`
- Tracking
 - `geant4/source/tracking`



Introduction

- Geant4 works as a set of nested loops:
 - **A job =**
 - Run manager construction and initialization;
 - Then one or several runs or launched;
 - **A run =**
 - Physics and detector construction;
 - Then loop on events;
 - **An event =**
 - » Generation of primary particles;
 - » Then loop for tracking of these particles and all subsequent secondary particles;
 - » **A particle tracking:**
 - Loop on steps, propagating a G4Track object, up to the point this object “dies”;
 - **A step =**
 - Loop on physics processes that apply to the current track to apply physics interactions, generate secondary particles, compute energy deposit in the step, etc.;

Introduction

- Geant4 works as a set of nested loops:
 - **A job =**
 - Run manager construction and initialization;
 - Then one or several runs or launched;
 - **A run =**
 - Physics and detector construction;
 - Then loop on events;
 - **An event =**
 - » Generation of primary particles;
 - » Then loop for tracking of these particles and all subsequent secondary particles;
 - » **A particle tracking:**
 - Loop on steps, propagating a G4Track object, up to the point this object “dies”;
 - **A step =**
 - Loop on physics processes that apply to the current track to apply physics interactions, generate secondary particles, compute energy deposit in the step, etc.;
- You can follow the progression of the simulation and take actions through a set of optional user’s actions:
 - **G4UserRunAction**
 - **G4UserEventAction**
 - **G4UserTrackingAction**
 - **G4UserSteppingAction**
- Explained in this presentation

Introduction

- Geant4 works as a set of nested loops:
 - **A job =**
 - Run manager construction and initialization;
 - Then one or several runs or launched;
 - **A run =**
 - Physics and detector construction;
 - Then loop on events;
 - **An event =**
 - » Generation of primary particles;
 - » Then loop for tracking of these particles and all subsequent secondary particles;
 - » **A particle tracking:**
 - Loop on steps, propagating a G4Track object, up to the point this object “dies”;
 - **A step =**
 - Loop on physics processes that apply to the current track to apply physics interactions, generate secondary particles, compute energy deposit in the step, etc.;
- You can follow the progression of the simulation and take actions through a set of optional user’s actions:
 - **G4UserRunAction**
 - **G4UserEventAction**
 - **G4UserTrackingAction**
 - **G4UserSteppingAction**
 - Explained in this presentation
- You are also given some handles to control the simulation flow:
 - **G4UserStackingAction** : allows to control in what ordering tracks are processed
 - **This will be shown in an other presentation**
- All these actions are optional... but are almost always used in sizeable applications.

User actions classes: virtual methods and invocation sequence

G4UserRunAction:

- **G4Run* GenerateRun()**
Allows user to generate a G4Run object of his/her type if void G4Run::RecordEvent(const G4Event*) method is overridden.
void BeginOfRunAction(const G4Run* currentRun)

Object default lifecycles

Run ✓
Event ✗
Track ✗
Step ✗

- **void EndOfRunAction(const G4Run* currentRun)**

✓ ✗ ✗ ✗

User actions classes: virtual methods and invocation sequence

G4UserRunAction:

- **G4Run* GenerateRun()**
Allows user to generate a G4Run object of his/her type if void G4Run::RecordEvent(const G4Event*) method is overridden.
void BeginOfRunAction(const G4Run* currentRun)



Loop on events



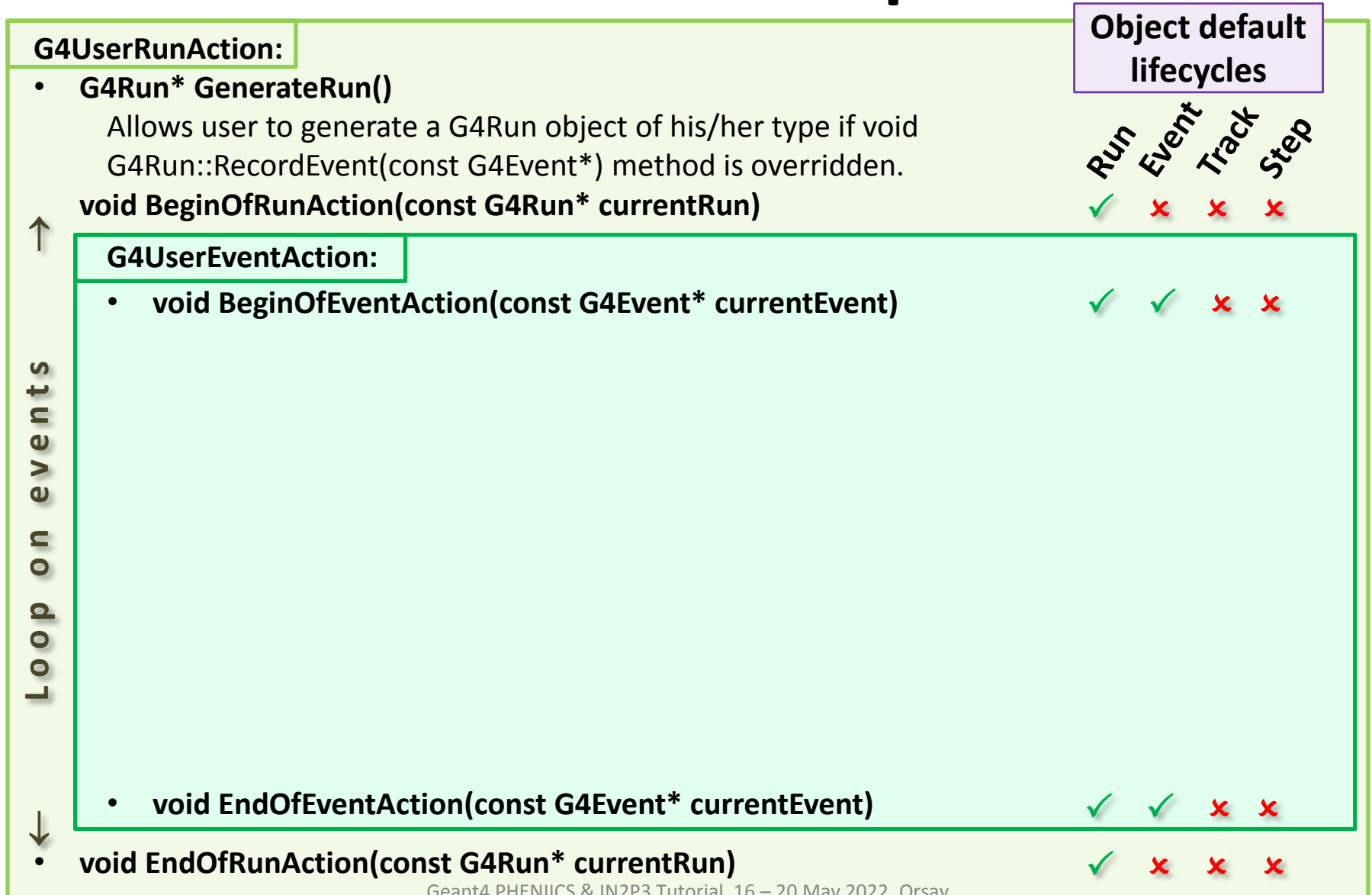
- **void EndOfRunAction(const G4Run* currentRun)**

Object default lifecycles

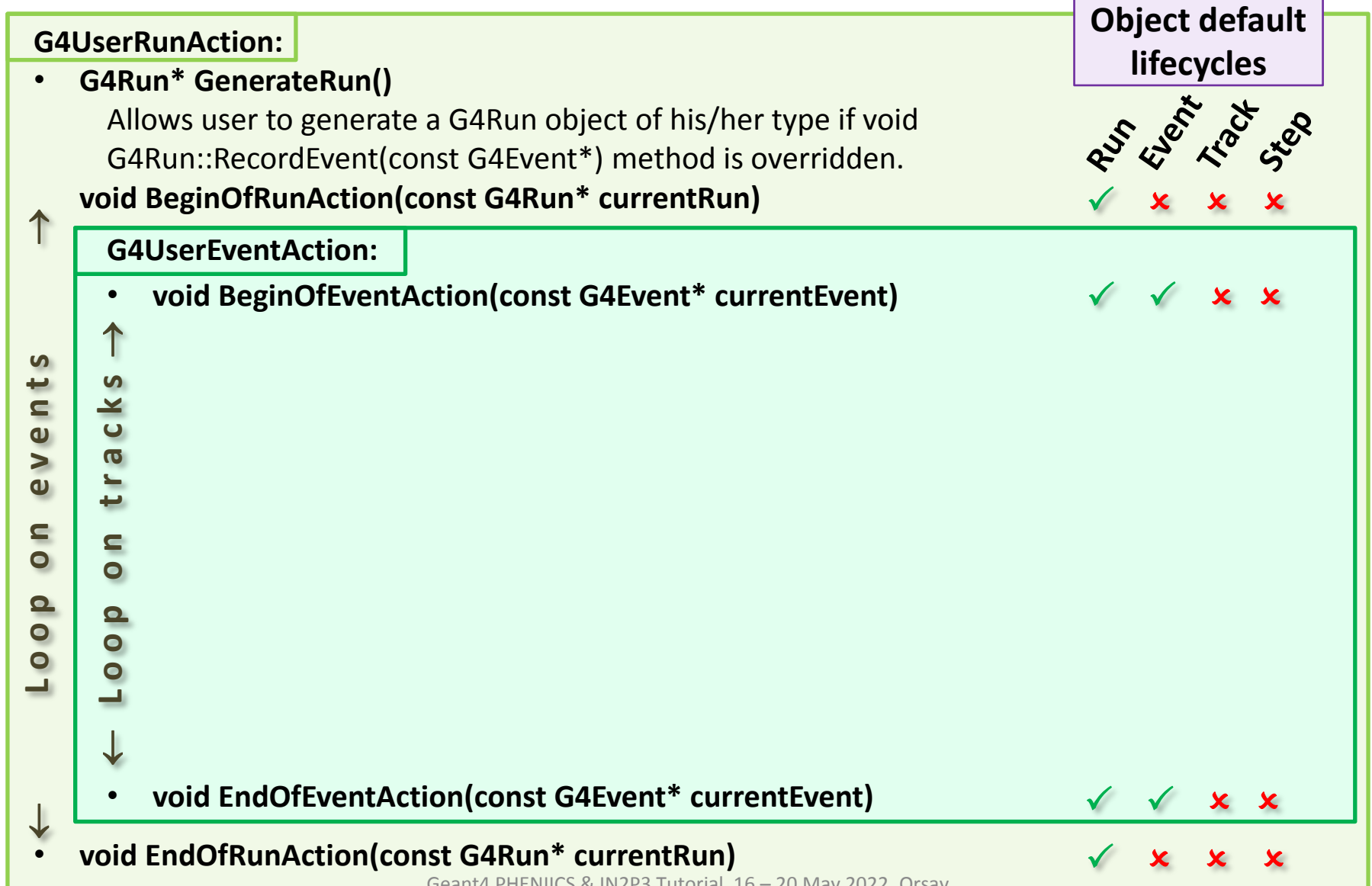
Run ✓
Event ✗
Track ✗
Step ✗

✓ ✗ ✗ ✗

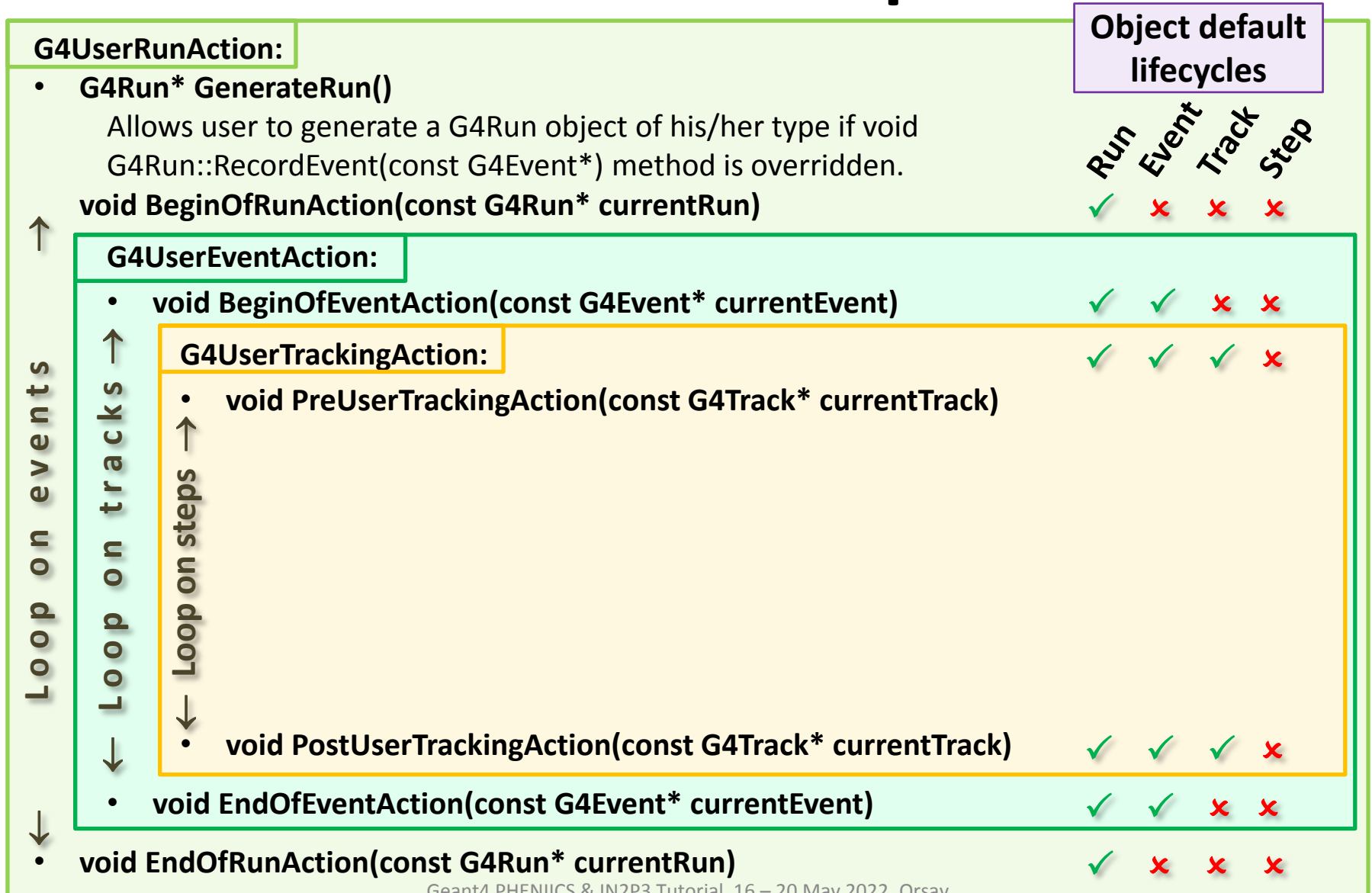
User actions classes: virtual methods and invocation sequence



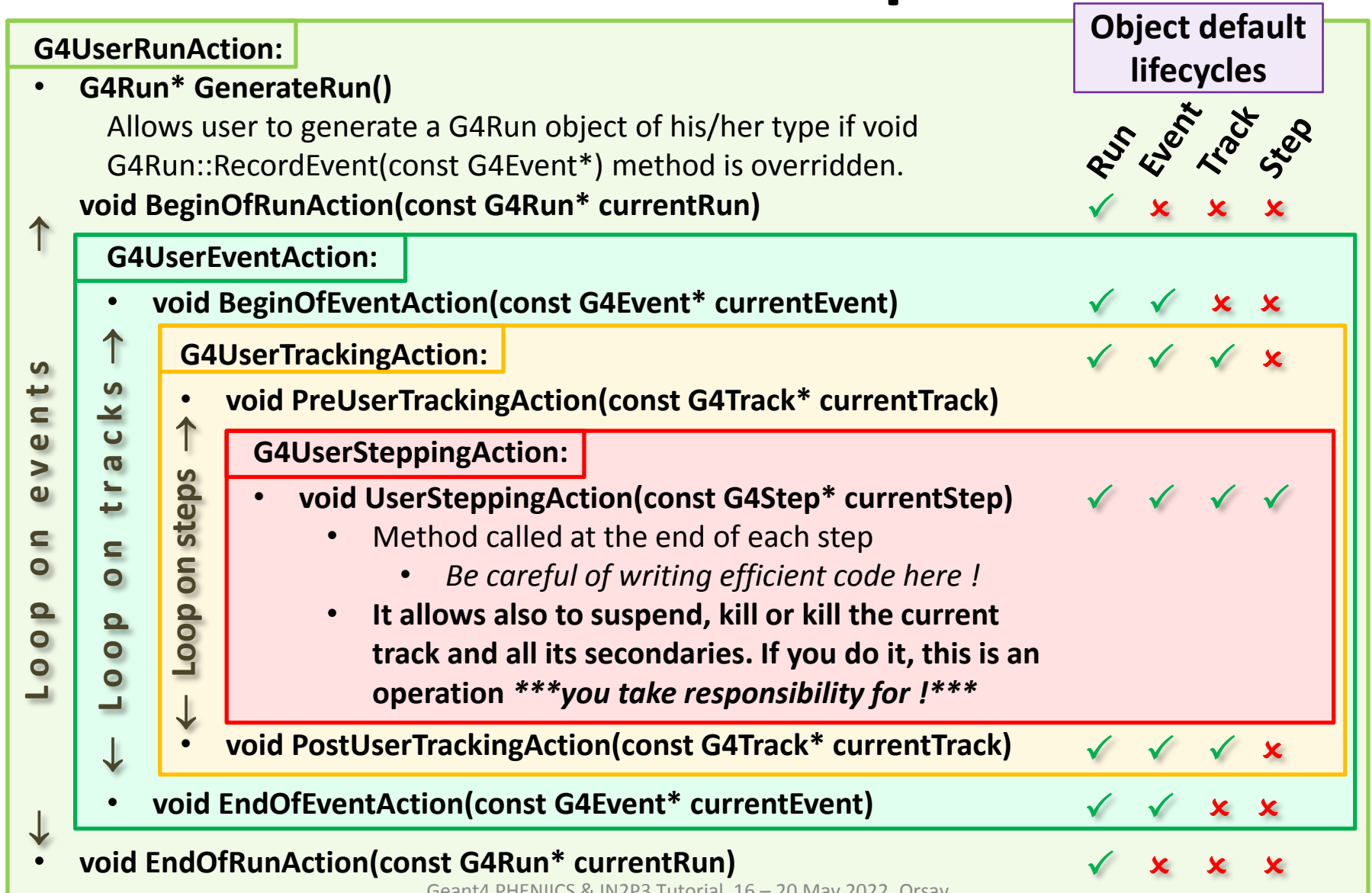
User actions classes: virtual methods and invocation sequence



User actions classes: virtual methods and invocation sequence



User actions classes: virtual methods and invocation sequence



User Action Classes :

virtual methods and usage examples

- G4UserRunAction**

	<u>Usage examples</u>
G4Run* GenerateRun()	Instantiate user-customized run object
void BeginOfRunAction(const G4Run*)	Define histograms
void EndOfRunAction(const G4Run*)	Analyze the run, store histograms
- G4UserEventAction**

void BeginOfEventAction(const G4Event*)	Event selection
void EndOfEventAction(const G4Event*)	Output event information
- G4TrackingAction**

void PreUserTrackingAction(const G4Track*)	Decide to store or not a trajectory; Create user-defined trajectory
void PostUserTrackingAction(const G4Track*)	Delete unnecessary trajectory
- G4UserSteppingAction**

void UserSteppingAction(const G4Step*)	Kill / suspend / postpone the track; Draw the step (for a track not to be stored as a trajectory)
--	--

Declaring Users Actions

- You define you user actions inheriting from the proper base classes:
 - class MyRunAction : public G4UserRunAction {...};
 - class MyEventAction : public G4UserEventAction {...};
 - class MyTrackingAction : public G4UserTrackingAction {...};
 - class MySteppingAction : public G4UserSteppingAction {...};
 - Overwriting the proper virtual methods.
- To take effect, these actions objects must be passed to the runManager:

- **In your action initialization class:**

```
void MyActionInitialization::Build() const
{
    SetUserAction(new MyRunAction);
    SetUserAction(new MyEventAction);
    SetUserAction(new MyTrackingAction);
    SetUserAction(new MySteppingAction);
}

void MyActionInitialization::BuildForMaster() const
{
    SetUserAction(new MyRunAction);
}
```

Mandatory method, must be provided.

Make your “new MyXXXAction” here !
(And *not* in constructor for example)

Only used in Multithreading mode (presented in session 7)

```
G4RunManager* runManager = new G4RunManager;
or
G4MTRunManager* runManager = new G4MTRunManager;
```

- **And in your main program:**

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

– if Geant4 version < Geant4 v10.0 (obsoleting):

- **In your main program:**

```
runManager->SetUserAction(new MyRunAction);
runManager->SetUserAction(new MyEventAction);
runManager->SetUserAction(new MyTrackingAction);
runManager->SetUserAction(new MySteppingAction);
```

Kept in v10.0 for backward compatibility

Summary

- Geant4 provides user action classes that allow you to take actions at the various stages of the simulation:
 - Start and end of run : **G4UserRunAction**
 - Start and end of event : **G4UserEventAction**
 - Start and end of tracking of one track : **G4UserTrackingAction**
 - End of each step : **G4UserSteppingAction**
 - This is the only of these classes with which **you can modify the simulation behavior**
 - eg: killing a track
 - But you take responsibility for this !
- You inherit from these base classes to implement you own actions
- You declare them to the run manager
 - Which is either a G4RunManager object, or your own run manager object, or a G4MTRunManager one, that will be presented later (session 7).
 - Using your concrete G4VUserActionInitialization class
 - Or invoking the runManager->SetUserAction(new MyXXXAction), if < v10.0 (obsoleting)
- An other user action class, **G4UserStackingAction**, that allows to control the simulation flow will be presented later.