Sensitive Detector

Geant4 CERN Tutorial

Wednesday, 17 February 2010

In This Lecture

- What is a Sensitive Detector?
- What is a G4Step?
- How to create a Sensitive Detector and us it

Introduction

- In task1 we have created a detector
- In task2 we have created primaries
- In task3 we have started to look at the physics simulation

- In these lessons (task4) you will learn how to extract information useful to you. Examples: energy released, number of particles, etc. Different methods:
  - Sensitive Detector (this lecture)
  - User Actions (next lecture)
  - Scoring (tomorrow)

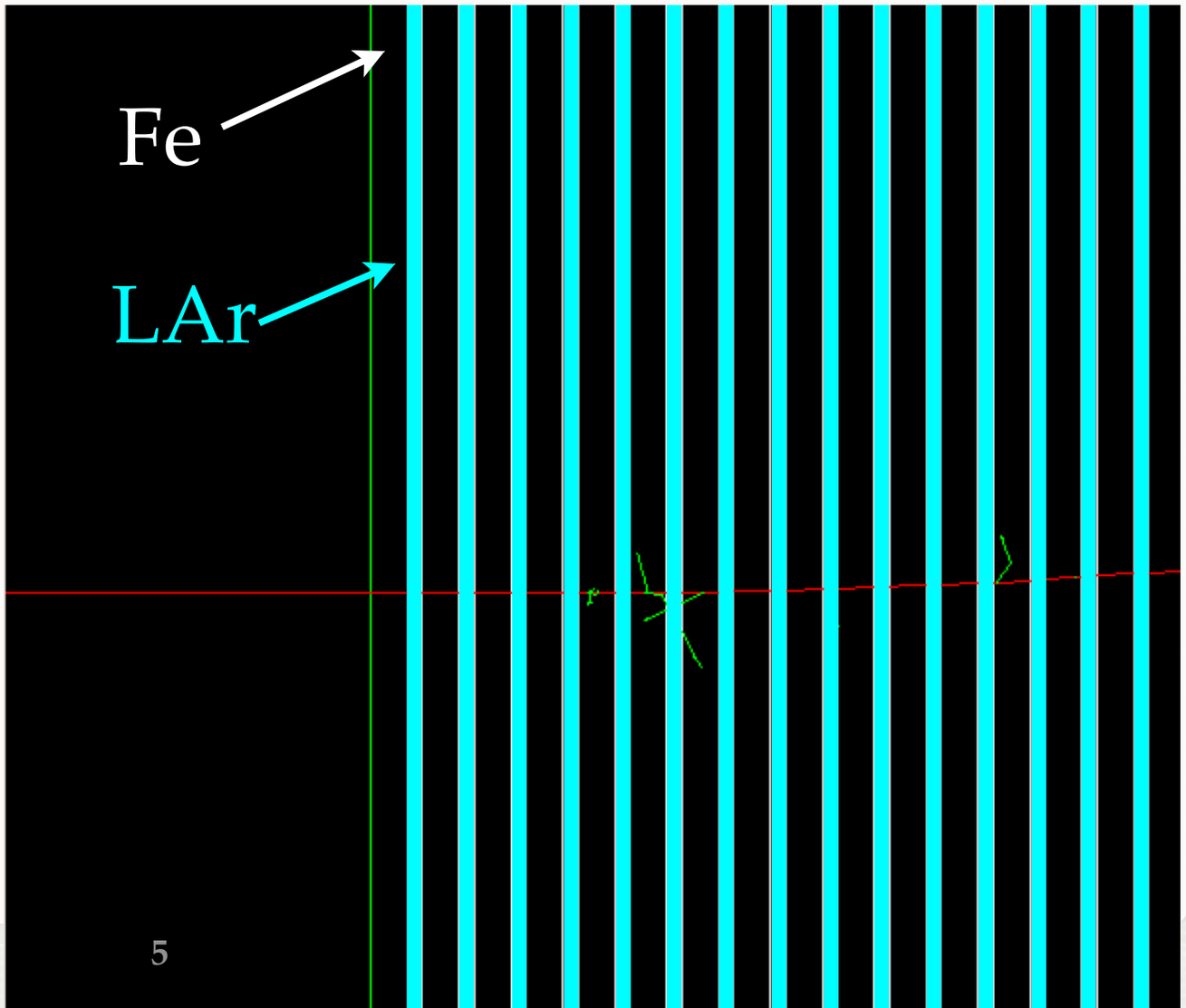
Sensitive Detector

- A SD can be used to simulate the "read-out" of your detector:
  - It is a way to declare a geometric element "sensitive" to the passage of particles
  - It gives the user a handle to collect quantities from these elements. For example: energy deposited, position, time information

Example: HadCalo

- Hadronic Calorimeter consists of layers of absorber (Fe) and layers of active material (LAr)
  - We want to collect energy released in LAr layers
  - G4 tracks particles in the detector, when a particle passes through a detector declared sensitive the user's SD code is called

mu- @ 2 GeV



SensitiveDetector Class

- To create a SD you need to:

  1. Write your SensitiveDetector class

  2. Attach it to a logical volume

- Example: HadCaloSensitiveDetctor class, inherits from G4VSensitiveDetector

  - G4VSensitiveDetector declares interface

Adding A SD

- Basic strategy (in src/DetectorConstruction.cc):

```
G4LogicalVolume* hadLayerLogic = new G4LogicalVolume(hadLayerSolid,lar,"HadLayerLogic");

HadCaloSensitiveDetector* sensitive = new HadCaloSensitiveDetector("/HadClo");
G4SDManager* sdman = G4SDManager::GetSDMpointer();
sdman->AddNewDetector( sensitive );
hadLayerLogic->SetSensitiveDetector(sensitive);
```

- Each SD object must have a unique name.
  - Different logical volumes can share one SD object.
  - More than one SD object can be made from the same SD class with different detector name.

SensitiveDetector Class

```
#include "G4VSensitiveDetector.hh"
```

Add include

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

SensitiveDetector Class

Base class

```
#include "G4VSensitiveDetector.hh"

class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

SensitiveDetector Class

```
#include "G4VSensitiveDetector.hh"
```

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

Constructor: SD are named!

SensitiveDetector Class

```
#include "G4VSensitiveDetector.hh"
```

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

Initialization: called at beginning of event

Note: G4HCofThisEvent will be discussed later today!

SensitiveDetector Class

```
#include "G4VSensitiveDetector.hh"
```

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}

private:
};
```

Finalize: called at end of event

Note: G4HCofThisEvent will be discussed later today!

SensitiveDetector Class

```
#include "G4VSensitiveDetector.hh"
```

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

Called for each G4Step in sensitive volume

SensitiveDetector Class

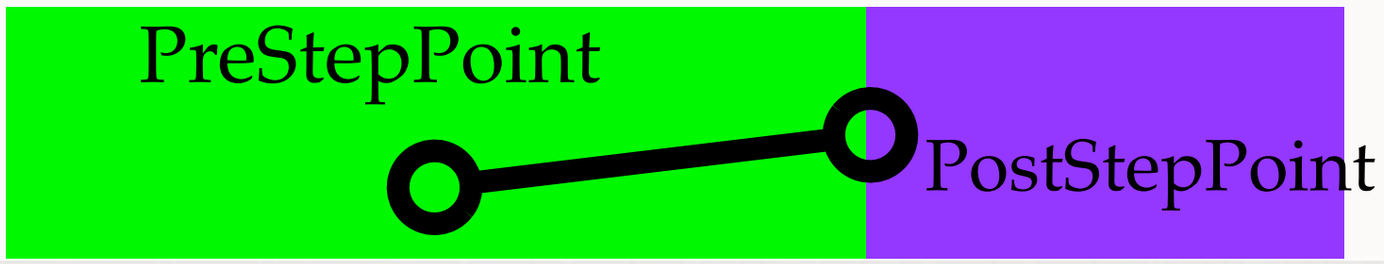
```
#include "G4VSensitiveDetector.hh"
```

```
class HadCaloSensitiveDetector : public G4VSensitiveDetector
{
public:
  /// Constructor
  HadCaloSensitiveDetector(G4String SDname);
  /// Destructor
  ~HadCaloSensitiveDetector();

public:
  /// @name methods from base class G4VSensitiveDetector
  //@{
  /// Mandatory base class method : it must to be overloaded:
   G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);

  /// (optional) method of base class G4VSensitiveDetector
  void Initialize(G4HCofThisEvent* HCE);
  /// (optional) method of base class G4VSensitiveDetector
  void EndOfEvent(G4HCofThisEvent* HCE);
  //@}


private:
};
```

G4Step: What is it?

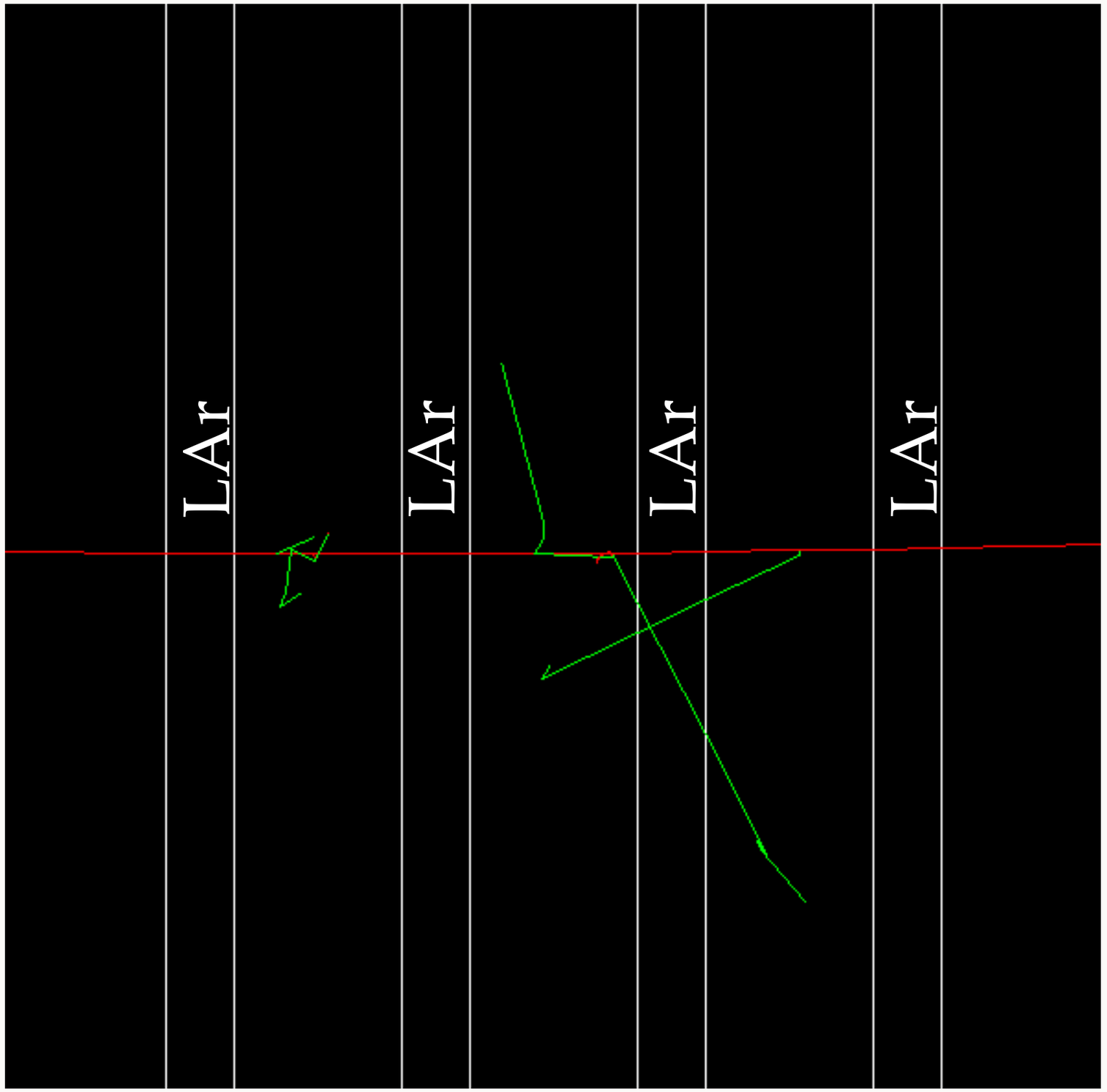
G4Step

- Snapshot of the interaction of a G4Track (particle) with a volume
- A G4Step can be seen as a "segment" delimited by two points
- It contains "delta" information (energy loss along the step, time-of-flight, etc)
- Each point knows the volume (and material) associated to it
- A step never spans across boundaries: geometry or physics define the end points
  - If the step is limited by a boundary, the post-step point stands on the boundary and it logically belongs to the next volume
  - Get the volume information from the PreStepPoint



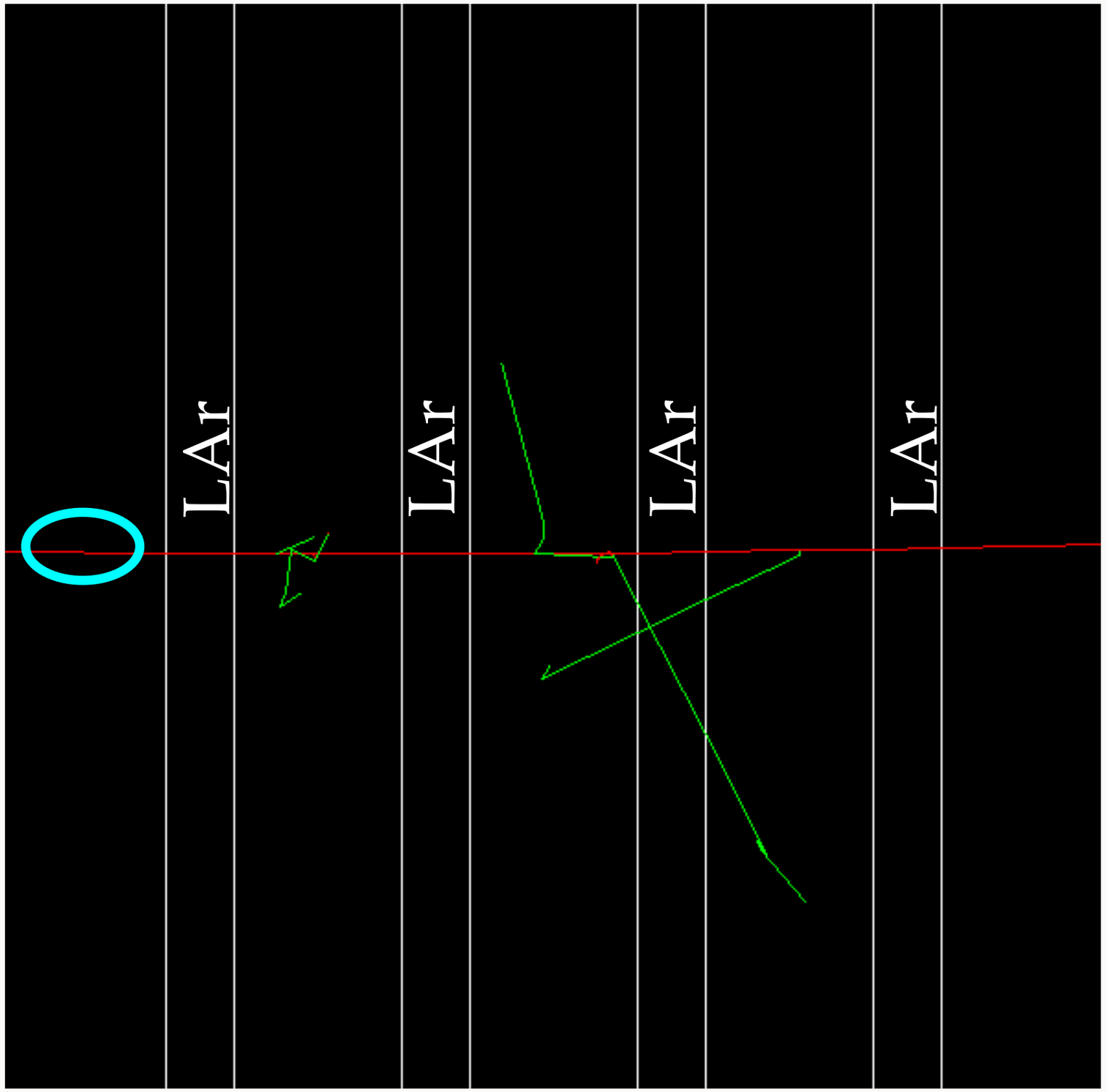
G4Step

The muon track passes through the calorimeter



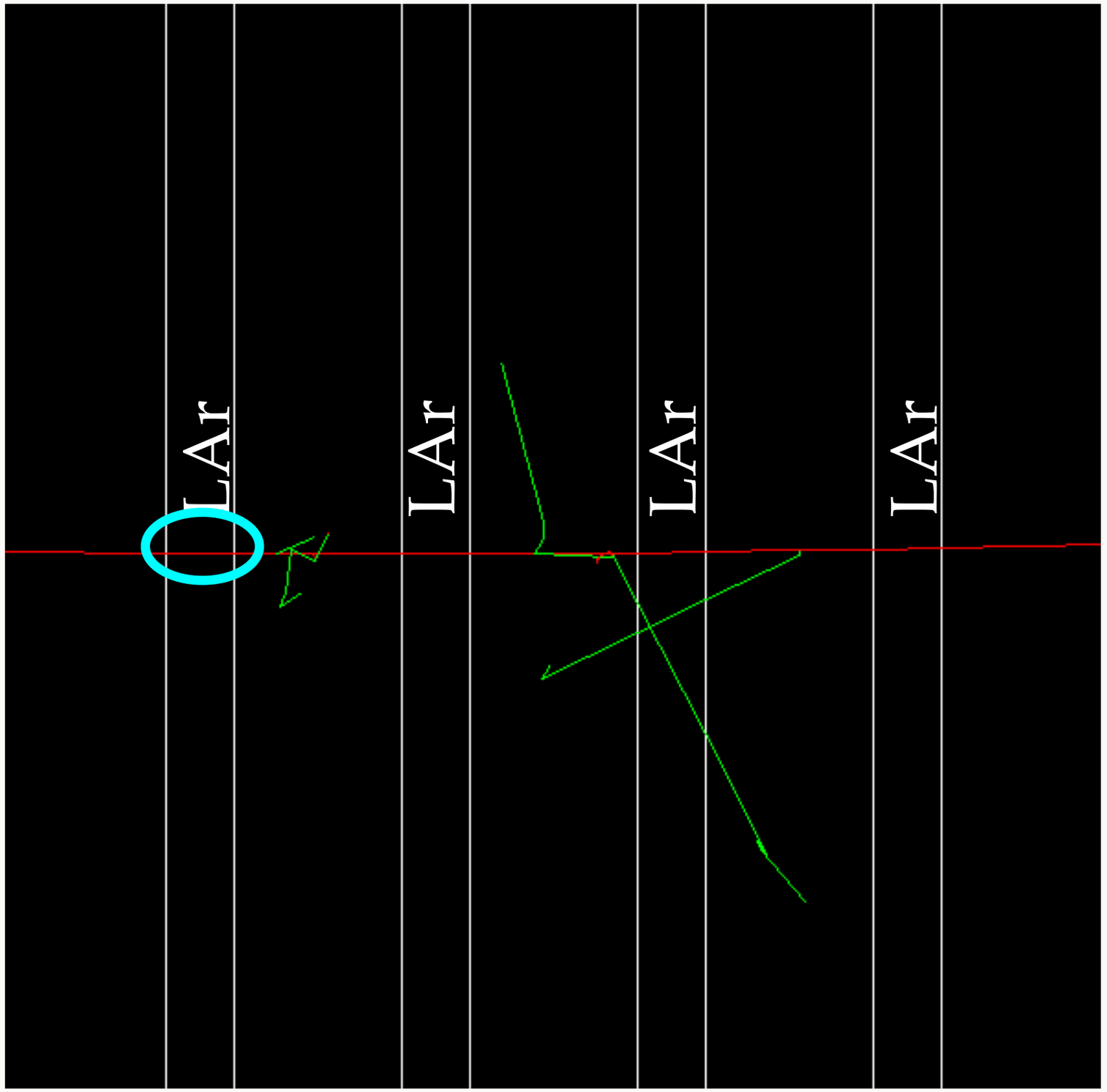
G4Step

A Step in Fe:
SD is ignored



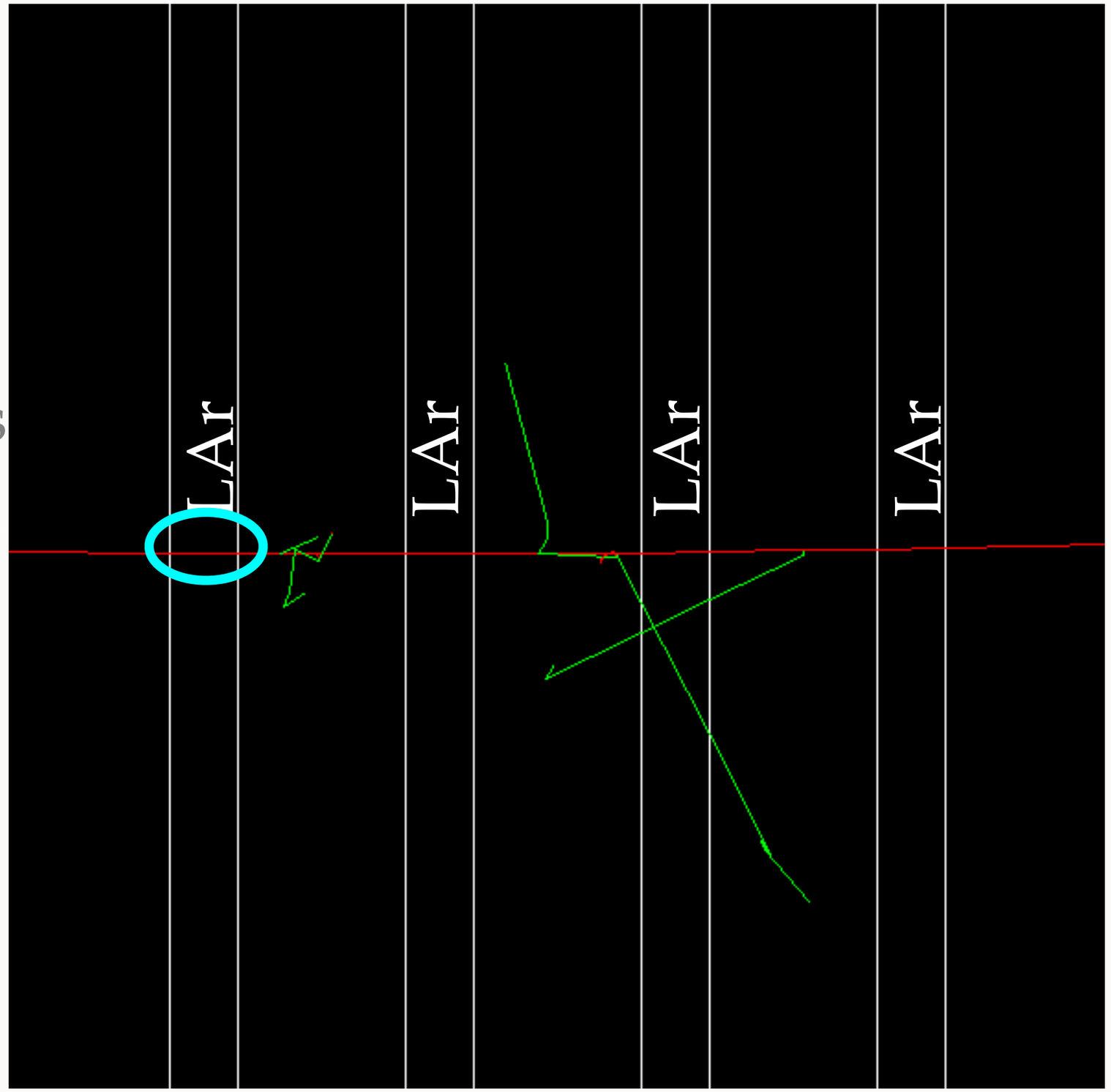
G4Step

A Step in LAr:
It's sensitive thus
::ProcessHits(...) will
be called



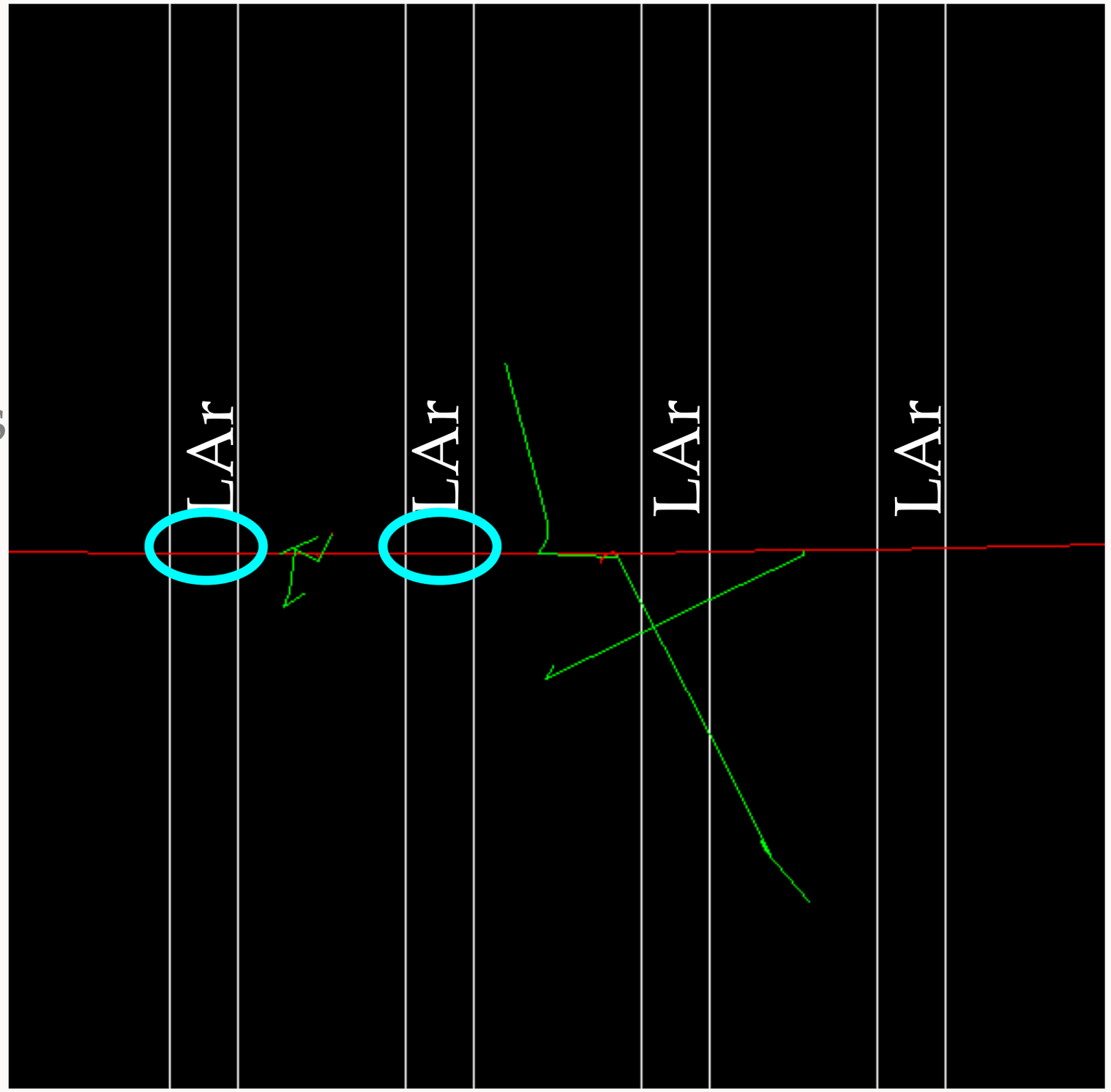
G4Step

For all these G4Steps ::ProcessHits(...) will be called



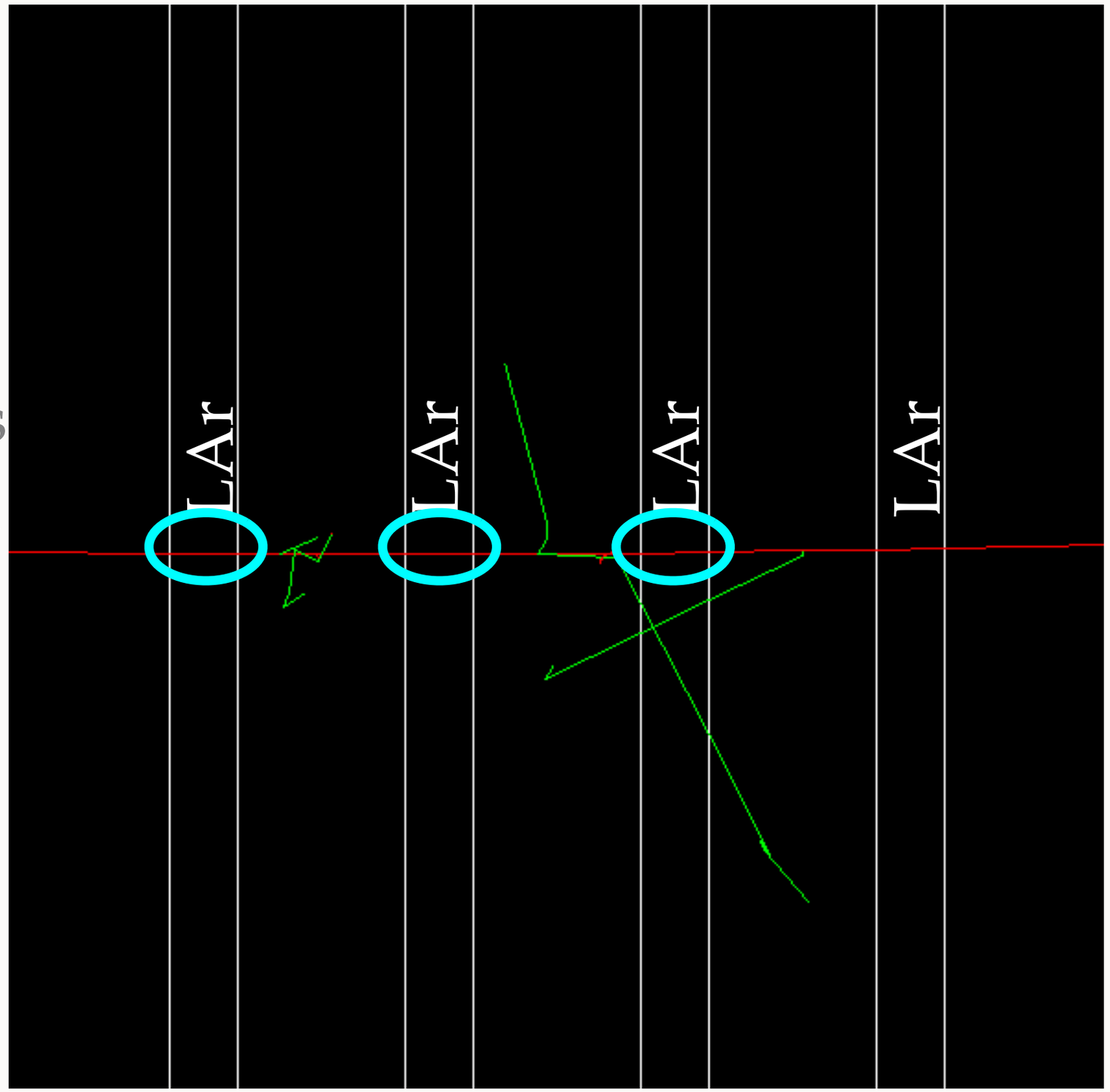
G4Step

For all these G4Steps ::ProcessHits(...) will be called



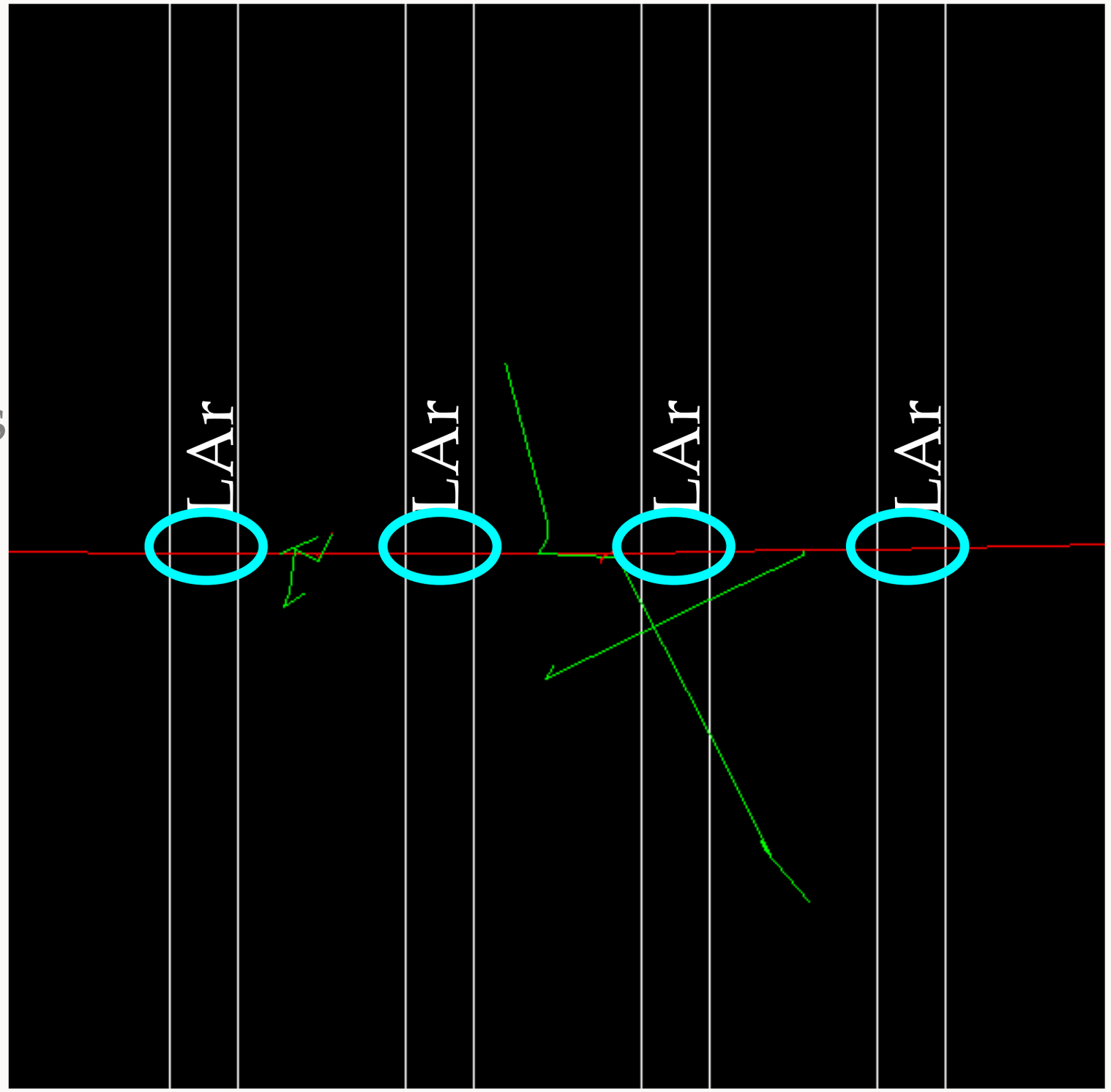
G4Step

For all these G4Steps ::ProcessHits(...) will be called



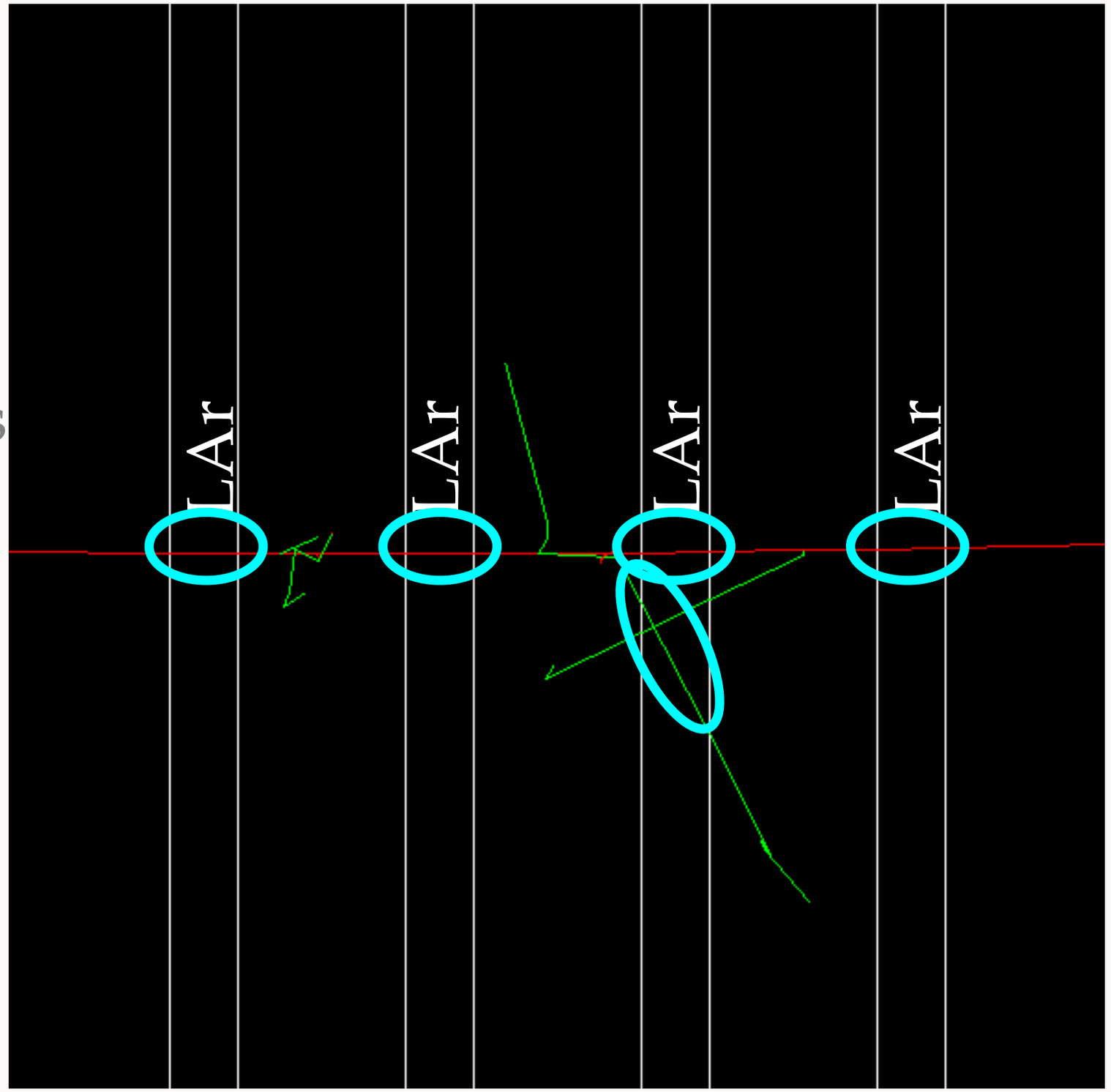
G4Step

For all these G4Steps ::ProcessHits(...) will be called



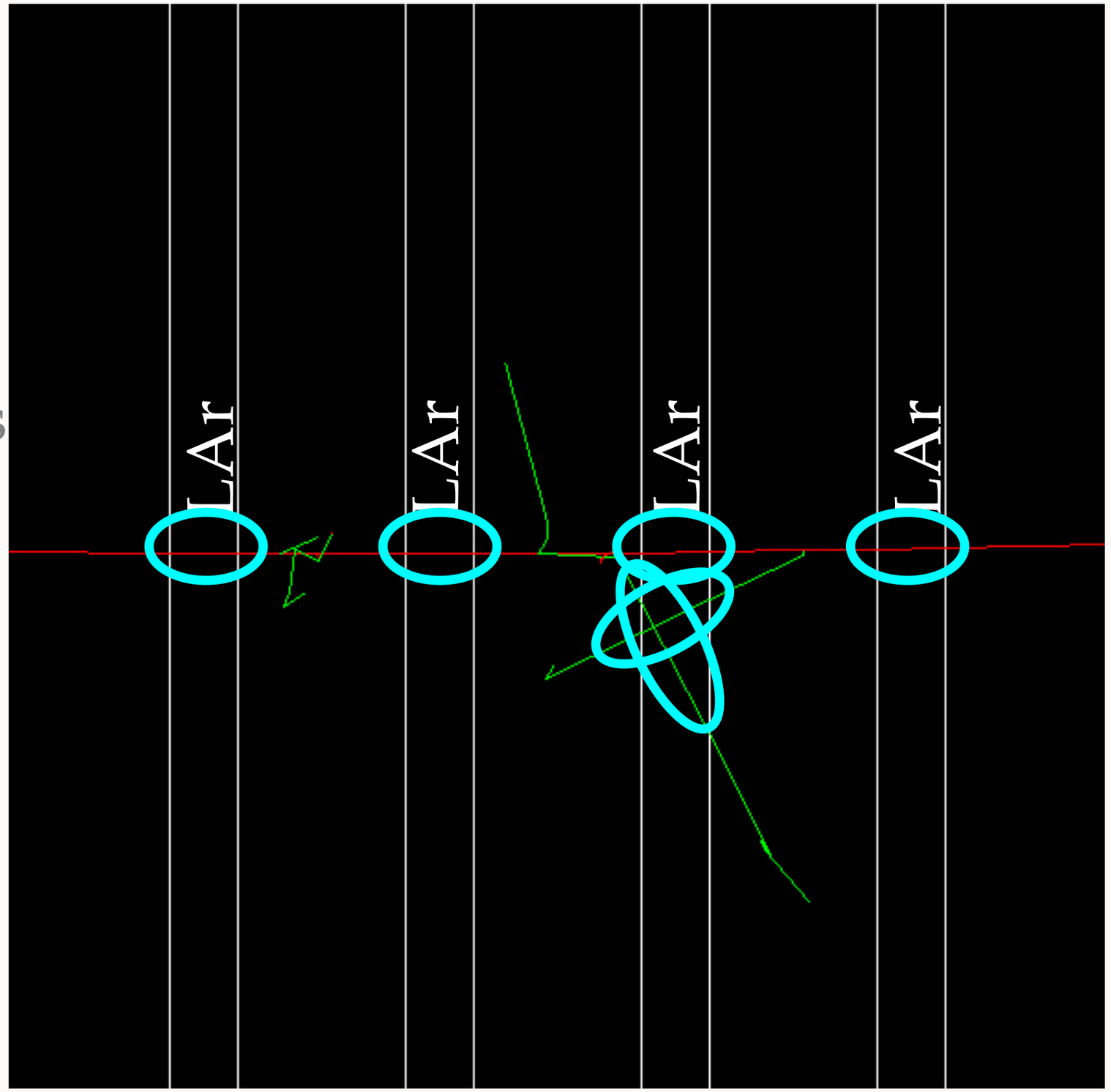
G4Step

For all these G4Steps ::ProcessHits(...) will be called



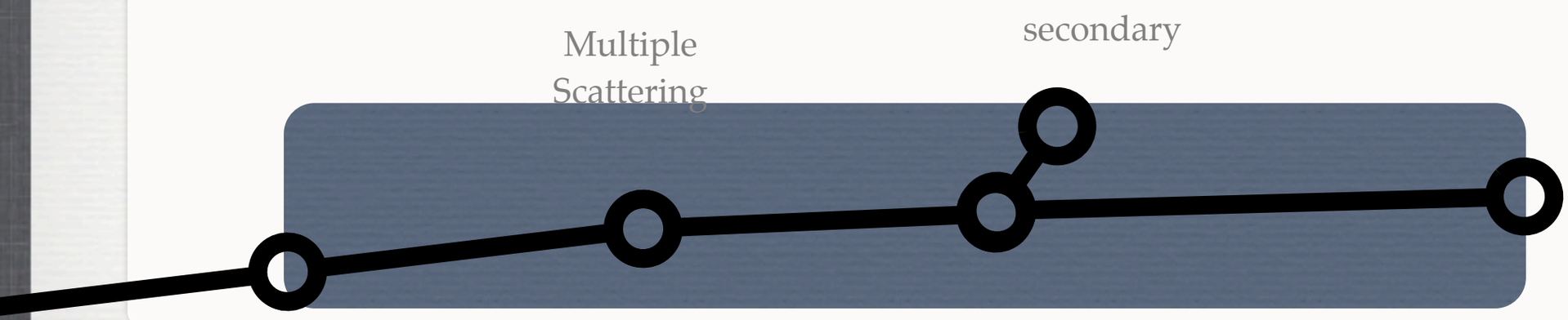
G4Step

For all these G4Steps ::ProcessHits(...) will be called



A Note On G4Step

- A G4Step is delimited by:
  - Geometry boundaries
  - A physics process (non continuous)
- G4Track is constant during step, G4 guarantees step is never too long (i.e. Edep does not change too much Ekin G4Track)



Getting Information From G4Steps

- G4Step can be interrogated to get information about physics process and volumes:

```
G4bool HadCaloSensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory *)
{
```

```
  G4TouchableHandle touchable = step->GetPreStepPoint()->GetTouchableHandle();
  G4int copyNo = touchable->GetVolume(0)->GetCopyNo();
```

Reminder: in task1 you created LAr layers with a unique ID (1001+layernum), this is the copy number: it uniquely identifies the volume in which the step is

Get volume where G4Step is remember:
Use PreStepPoint! PostStep "belongs" to next volume

```
  G4double edep = step->GetTotalEnergyDeposit();
```

Get energy deposited along G4Step (i.e. ionization)

Exercise

- task4a:
  - Instantiate a SD
  - Attach it to LAr layers
  - Retrieve energy deposited in LAr and print on screen
- cd g4course2010/task4/task4a