Workshop on monte-carol simulations-applications in science and technology
May 15-17, 2017

Source in Geant4
Primary Generator Action

S.Aogaki

Before presentation

- My name is Soichiro Aogaki
- I am working at ELI-NP (Extreme Light Infrastructure - Nuclear Physics), Magurele (near Bucharest), Romania
- Dr. Zafar Yasin also worked till 2016
- I am working with Geant4
- My presentation is standing user side, NOT developer side

Before presentation

- Geant4 is the toolkit to make simulator between particles and matters
- Installation and overview of it (and also some basics about C++ and ROOT) will be presented by Dr. Tariq and Dr. Yasir
- I will talk about some important parts of Geant4 application
- I wish its will help you to write the Geant4 application
- For not C++ user, my talk is not so friendly

The structure of Geant4 application

- First, Constructing Run manager
- Multithreading is strongly recommended by me
- We make many events
- Each event are totally independent
- Roughly say, we can reduce computing time (1 / Nr. threads)

```
int main(int argc, char **argv)
{

   // Construct the default run manager
#ifdef G4MULTITHREADED
   G4MTRunManager *runManager = new G4MTRunManager();
   runManager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
#else
   G4RunManager *runManager = new G4RunManager();
#endif

   // Detector construction
   runManager->SetUserInitialization(new MyDetectorConstruction());

   // Physics list
   G4VModularPhysicsList *physicsList = new MyPhysicsList();
   physicsList->SetVerboseLevel(0);
   runManager->SetUserInitialization(physicsList);

   // Primary generator action and User action initialization
   runManager->SetUserInitialization(new MyActionInitialization());

   // Initialize G4 kernel
   //
   runManager->Initialize();

   …

   return 0;
}
```

The structure of Geant4 application

- Second, Constructing geometries
- I will explain very basics of constructing the geometry tomorrow

```
int main(int argc, char **argv)
{
   …
   // Construct the default run manager
#ifdef G4MULTITHREADED
   G4MTRunManager *runManager = new G4MTRunManager();
   runManager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
#else
   G4RunManager *runManager = new G4RunManager();
#endif

   // Detector construction
   runManager->SetUserInitialization(new MyDetectorConstruction());

   // Physics list
   G4VModularPhysicsList *physicsList = new MyPhysicsList();
   physicsList->SetVerboseLevel(0);
   runManager->SetUserInitialization(physicsList);

   // Primary generator action and User action initialization
   runManager->SetUserInitialization(new MyActionInitialization());

   // Initialize G4 kernel
   //
   runManager->Initialize();

   …

   return 0;
}
```

The structure of Geant4 application

- Third, Defining physics model used (Physics list)
- Uses can choose physics model used
- This is not explained
- Users can use Physics list implemented in Geant4
- For starting point, using predefined Physics list is better

```
int main(int argc, char **argv)
{
   …
   // Construct the default run manager
#ifdef G4MULTITHREADED
   G4MTRunManager *runManager = new G4MTRunManager();
   runManager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
#else
   G4RunManager *runManager = new G4RunManager();
#endif

   // Detector construction
   runManager->SetUserInitialization(new MyDetectorConstruction());

   // Physics list
   G4VModularPhysicsList *physicsList = new MyPhysicsList();
   physicsList->SetVerboseLevel(0);
   runManager->SetUserInitialization(physicsList);

   // Primary generator action and User action initialization
   runManager->SetUserInitialization(new MyActionInitialization());

   // Initialize G4 kernel
   //
   runManager->Initialize();

   …

   return 0;
}
```

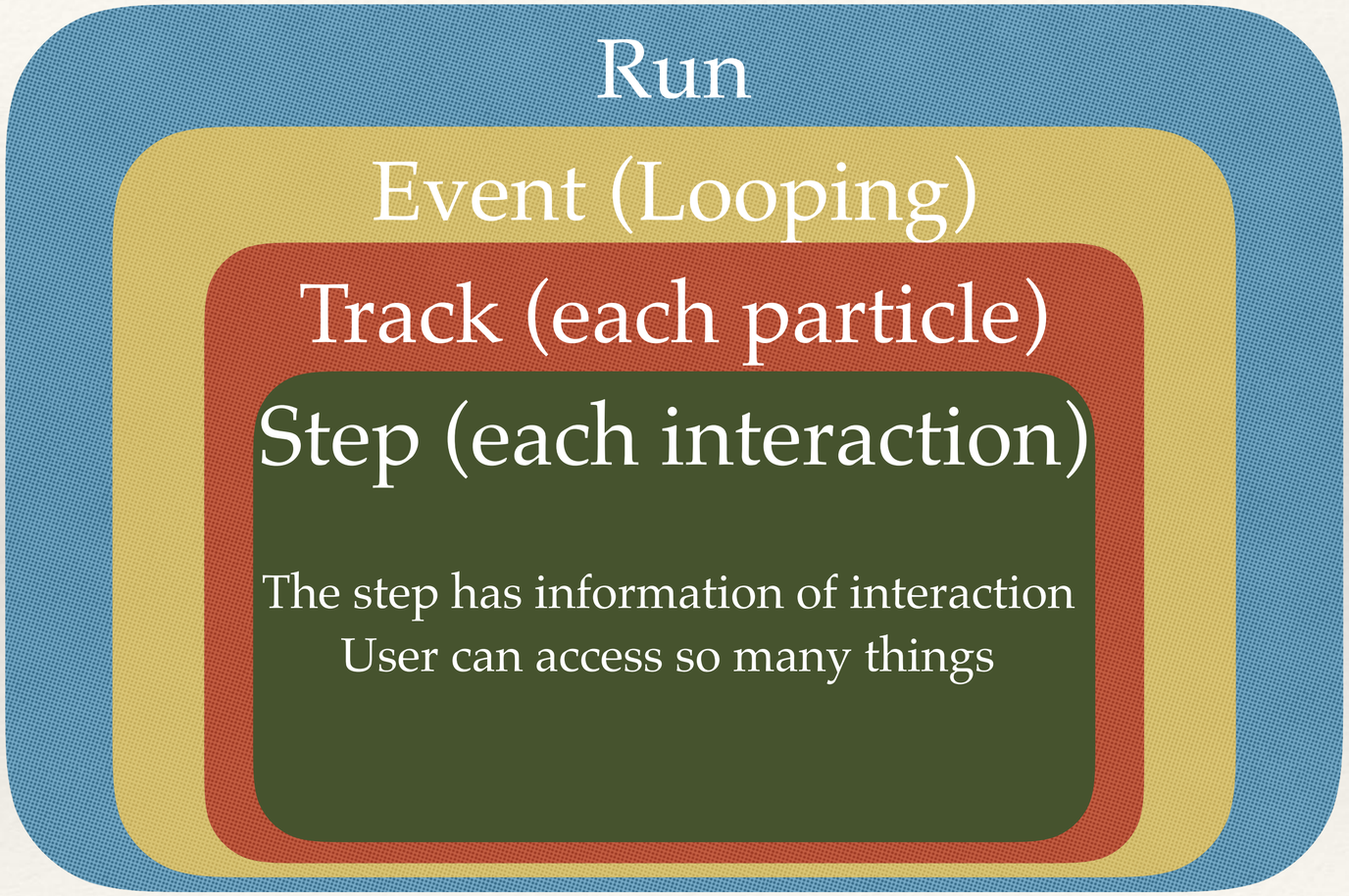
The structure of Geant4 application

- Last, user define some action in the simulation

```
int main(int argc, char **argv)
{
   …
   // Construct the default run manager
#ifdef G4MULTITHREADED
   G4MTRunManager *runManager = new G4MTRunManager();
   runManager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
#else
   G4RunManager *runManager = new G4RunManager();
#endif

   // Detector construction
   runManager->SetUserInitialization(new MyDetectorConstruction());

   // Physics list
   G4VModularPhysicsList *physicsList = new MyPhysicsList();
   physicsList->SetVerboseLevel(0);
   runManager->SetUserInitialization(physicsList);

   // Primary generator action and User action initialization
   runManager->SetUserInitialization(new MyActionInitialization());

   // Initialize G4 kernel
   //
   runManager->Initialize();

   …

   return 0;
}
```

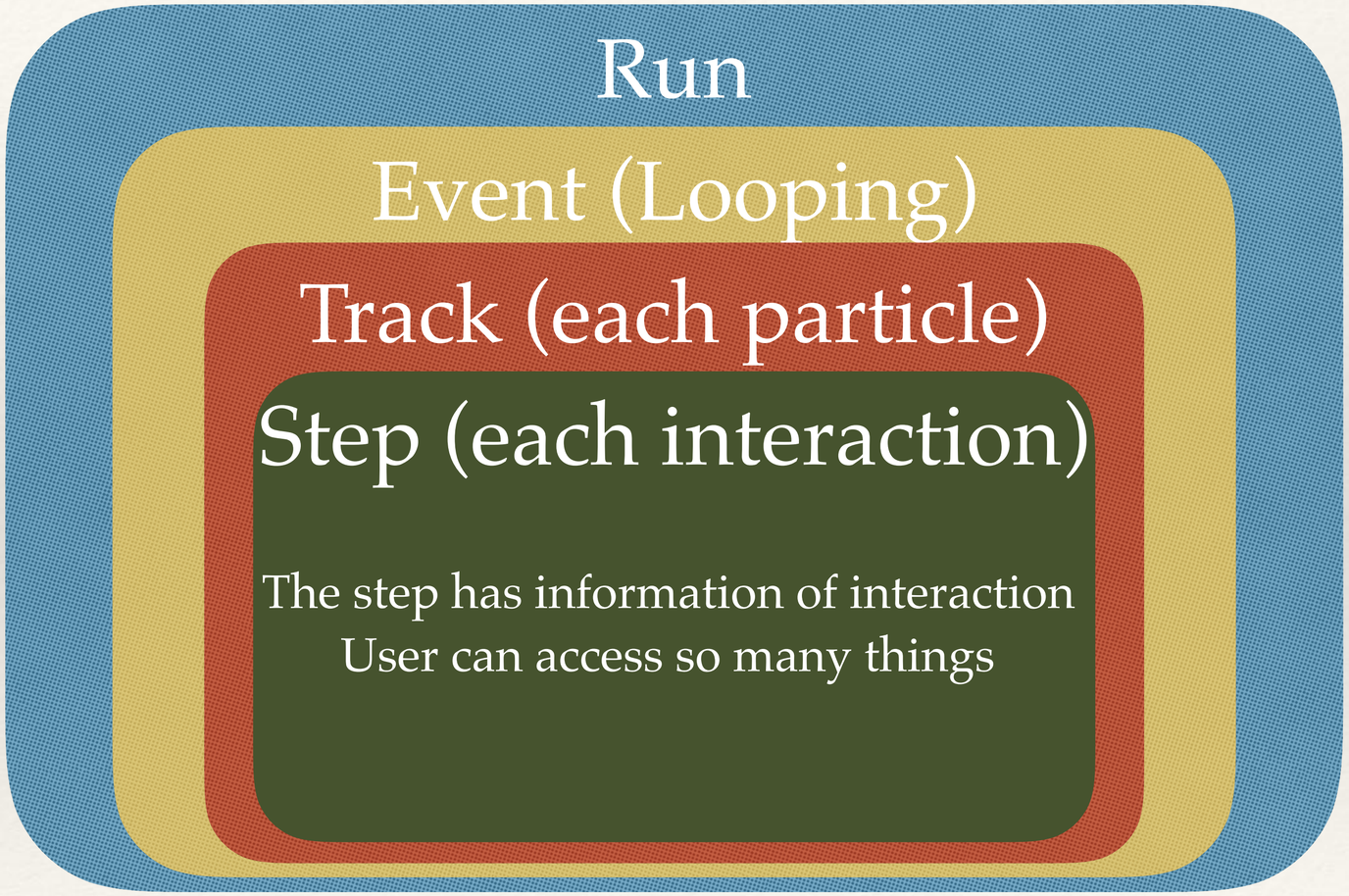
The structure of Geant4 application

- The number of event is set by user
- The number of tracks is same as number of particles generated in one event
- And Step is made at each interaction



The structure of Geant4 application

- User can make Action class for Run, Event, Track, and Step
- In my talk, I don't use Track and Step
- I use another way to obtain the interaction information



The structure of Geant4 application
MyActionInitialization class

- This is my ActionInitialization class
- As you see, this is very simple
- In the Build()
- Some *Action() classes are set
- Usually, these define some actions before and end of one iteration

```
#include "MyActionInitialization.hpp"
#include "MyPrimaryGeneratorAction.hpp"
#include "MyRunAction.hpp"
#include "MyEventAction.hpp"


MyActionInitialization::MyActionInitialization()
   : G4VUserActionInitialization()
{}

MyActionInitialization::~MyActionInitialization()
{}

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

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

The structure of Geant4 application
RunAction class

- My RunAction do make ntuple and save it

```
#include <g4root.hh>

#include "MyRunAction.hpp"


MyRunAction::MyRunAction()
   : G4UserRunAction()
{}

MyRunAction::~MyRunAction()
{}

void MyRunAction::BeginOfRunAction(const G4Run *)
{
   G4AnalysisManager *anaMan = G4AnalysisManager::Instance();
   anaMan->SetVerboseLevel(1);
   G4String fileName = "result";
   anaMan->OpenFile(fileName);
   anaMan->CreateNtuple("hits", "test");
   anaMan->CreateNtupleIColumn(0, "EventID");
   anaMan->CreateNtupleIColumn(0, "TrackID");

   anaMan->CreateNtupleDColumn(0, "x");
   anaMan->CreateNtupleDColumn(0, "y");
   anaMan->CreateNtupleDColumn(0, "z");

   anaMan->CreateNtupleDColumn(0, "DepositEnergy");

   anaMan->FinishNtuple();
}

void MyRunAction::EndOfRunAction(const G4Run *)
{
   G4AnalysisManager *anaMan = G4AnalysisManager::Instance();
   anaMan->Write();
   anaMan->CloseFile();
}
```

The structure of Geant4 application
EventAction class

- My EventAction add (fill) ntuple at end of event
- HitsCollection will be explained at scoring session

```
#include <G4Event.hh>
#include <G4SDManager.hh>
#include <g4root.hh>

#include "MyEventAction.hpp"


MyEventAction::MyEventAction()
    : G4UserEventAction(),
      fHitsCollectionID(-1)
{}

MyEventAction::~MyEventAction()
{}

MyHitsCollection *MyEventAction::GetHitsCollection(G4int hcID, const G4Event *event)
const
{
    …
}

void MyEventAction::BeginOfEventAction(const G4Event *)
{}

void MyEventAction::EndOfEventAction(const G4Event *event)
{
    …
    MyHitsCollection *hc = GetHitsCollection(fHitsCollectionID, event);
    …
    G4AnalysisManager *anaMan = G4AnalysisManager::Instance();

    const G4int kHit = hc->entries();
    for (G4int iHit = 0; iHit < kHit; iHit++) {
        …
        anaMan->AddNtupleRow(0);
    }
}
```

The structure of Geant4 application
MyActionInitialization class

- In this session, I will talk about user defined PrimaryGeneratorAction
- It makes the primary particle for one event

```
#include "MyActionInitialization.hpp"
#include "MyPrimaryGeneratorAction.hpp"
#include "MyRunAction.hpp"
#include "MyEventAction.hpp"


MyActionInitialization::MyActionInitialization()
   : G4VUserActionInitialization()
{}

MyActionInitialization::~MyActionInitialization()
{}

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

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

Before presentation

- I made one example including my talks
- Please download from
- https://github.com/aogaki/Workshop2017
- e-mail: sohichiroh.aogaki@nipne.ro

Contents

- Introduction
- How to define the primal particle and its properties
- Random number generator
- Using ROOT classes
- How to define an ion as the primary particle

Introduction

Introduction

- G4VUserPrimaryGeneratorAction class
- User defines own PrimaryGeneratorActionClass
  - User defines ParticleGun (G4ParticleGun class)

How to define the particle and its properties

ParticleGun

- Setting the Particle Gun
- Some variables starting with “f”, it is member variable
- Its are defined in other file “header file”

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
    : G4VUserPrimaryGeneratorAction(),
      fParticleGun(nullptr),
      fMessenger(nullptr)
{
    fFirstFlag = true;
    fZPosition = -300.*mm;

    fZ = 1;
    fA = 1;
    fIonCharge   = 1.*eplus;
    fExcitEnergy = 0.*keV;

    G4int nPar = 1;
    fParticleGun = new G4ParticleGun(nPar);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle
       = particleTable->FindParticle("geantino");
    fParticleGun->SetParticleDefinition(particle);
    fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., fZPosition));
    fParticleGun->SetParticleEnergy(1.*MeV);
    fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));

    DefineCommands();
}
```

Parti

- Setting the Particle Gun
- Some variables starting with “f”, it is member variable
- Its are defined in other file “header file”

```
#ifndef MyPrimaryGeneratorAction_h
#define MyPrimaryGeneratorAction_h 1

…

class MyPrimaryGeneratorAction: public G4VUserPrimaryGeneratorAction
{
public:
   MyPrimaryGeneratorAction();
   virtual ~MyPrimaryGeneratorAction();

   virtual void GeneratePrimaries(G4Event *);

private:
   G4ParticleGun *fParticleGun;
   G4double fZPosition;
   G4bool fFirstFlag;

   // Ion parameters
   void SetIon();
   G4int fZ;
   G4int fA;
   G4double fIonCharge;
   G4double fExcitEnergy;

   // Commands
   void DefineCommands();
   G4GenericMessenger *fMessenger;
   // Changing Ion
   void SetIonCmd(G4String name);

   TH2D *fHisSource2D;
   TH1D *fHisSource1D;
};

#endif
```

ParticleGun

- Setting the Particle Gun
- Using new G4ParticleGun
- nPar means how many particles are generated in one event

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
    : G4VUserPrimaryGeneratorAction(),
      fParticleGun(nullptr),
      fMessenger(nullptr)
{
    fFirstFlag = true;
    fZPosition = -300.*mm;

    fZ = 1;
    fA = 1;
    fIonCharge   = 1.*eplus;
    fExcitEnergy = 0.*keV;

    G4int nPar = 1;
    fParticleGun = new G4ParticleGun(nPar);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle
       = particleTable->FindParticle("geantino");
    fParticleGun->SetParticleDefinition(particle);
    fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., fZPosition));
    fParticleGun->SetParticleEnergy(1.*MeV);
    fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));

    DefineCommands();
}
```

ParticleGun

- Setting the Particle Gun
- Set the particle to the gun
- geantino is the particle for testing
- You can set any particles in the particle table of Geant4

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
    : G4VUserPrimaryGeneratorAction(),
      fParticleGun(nullptr),
      fMessenger(nullptr)
{
    fFirstFlag = true;
    fZPosition = -300.*mm;

    fZ = 1;
    fA = 1;
    fIonCharge   = 1.*eplus;
    fExcitEnergy = 0.*keV;

    G4int nPar = 1;
    fParticleGun = new G4ParticleGun(nPar);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle
       = particleTable->FindParticle("geantino");
    fParticleGun->SetParticleDefinition(particle);
    fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., fZPosition));
    fParticleGun->SetParticleEnergy(1.*MeV);
    fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));

    DefineCommands();
}
```

ParticleGun

- Setting the Particle Gun
- Set the source position kinetic energy and momentum
- Please don't forget to use units for variables
- It convert the unit of value
- In the Geant4 1.*MeV means 1., 1.*GeV means 1000.

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
    : G4VUserPrimaryGeneratorAction(),
      fParticleGun(nullptr),
      fMessenger(nullptr)
{
    fFirstFlag = true;
    fZPosition = -300.*mm;

    fZ = 1;
    fA = 1;
    fIonCharge   = 1.*eplus;
    fExcitEnergy = 0.*keV;

    G4int nPar = 1;
    fParticleGun = new G4ParticleGun(nPar);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle
       = particleTable->FindParticle("geantino");
    fParticleGun->SetParticleDefinition(particle);
    fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., fZPosition));
    fParticleGun->SetParticleEnergy(1.*MeV);
    fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));

    DefineCommands();
}
```

ParticleGun

- Using the Particle Gun

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
   G4double ene = G4RandExponential::shoot(100)*MeV;
   fParticleGun->SetParticleEnergy(ene);
   fParticleGun->GeneratePrimaryVertex(event);

   G4AutoLock lock(&mutexInPGA);
   if (nEveInPGA++ % 10000 == 0)
      G4cout << nEveInPGA - 1 << " events done" << G4endl;
}
```

ParticleGun

- Using the Particle Gun
- Set the kinetic energy generated with exp(x / -100) MeV
- Using random number generator in Geant4

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
   G4double ene = G4RandExponential::shoot(100)*MeV;
   fParticleGun->SetParticleEnergy(ene);
   fParticleGun->GeneratePrimaryVertex(event);

   G4AutoLock lock(&mutexInPGA);
   if (nEveInPGA++ % 10000 == 0)
      G4cout << nEveInPGA - 1 << " events done" << G4endl;
}
```

ParticleGun

- Using the Particle Gun
- Generating for an event
- All parameters can be changed (i.e. source position, particle direction, and etc.)

```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
   G4double ene = G4RandExponential::shoot(100)*MeV;
   fParticleGun->SetParticleEnergy(ene);
   fParticleGun->GeneratePrimaryVertex(event);

   G4AutoLock lock(&mutexInPGA);
   if (nEveInPGA++ % 10000 == 0)
      G4cout << nEveInPGA - 1 << " events done" << G4endl;
}
```

Random number generator

Random number generator

- This is about C++, not Geant4
- But, Useful for Monte-Carlo simulation
- This is using C++11 features
- When you use g++, Don't forget -std=c++11 (for some version of gcc)

Random number generator

- Since C++11 standard, C++ has some better random number generator
- std::rand() is not good random generator
- std::random_device is Non-deterministic random numbers
- Very good randomness. But, very slow

```
#include <iostream>
#include <random>

int main(void)
{
  std::random_device ranDev;
  for(int i = 0; i < 10; i++){
    std::cout << ranDev() << std::endl;
  }
}
```

Random number generator

- For good randomness and speed
- Mersenne Twister engine is good (Predefined random number)
- Using seed as random_device
- time() is also good seed
- If you have possibility of starting some jobs at same time, use random_device!
- Also for Geant4, generating seed is same procedure

```
#include <iostream>
#include <random>

int main(void)
{
  std::random_device ranDev;
  std::mt19937 mt(ranDev());

  for(int i = 0; i < 10; i++){
    std::cout << mt() << std::endl;
  }
}
```

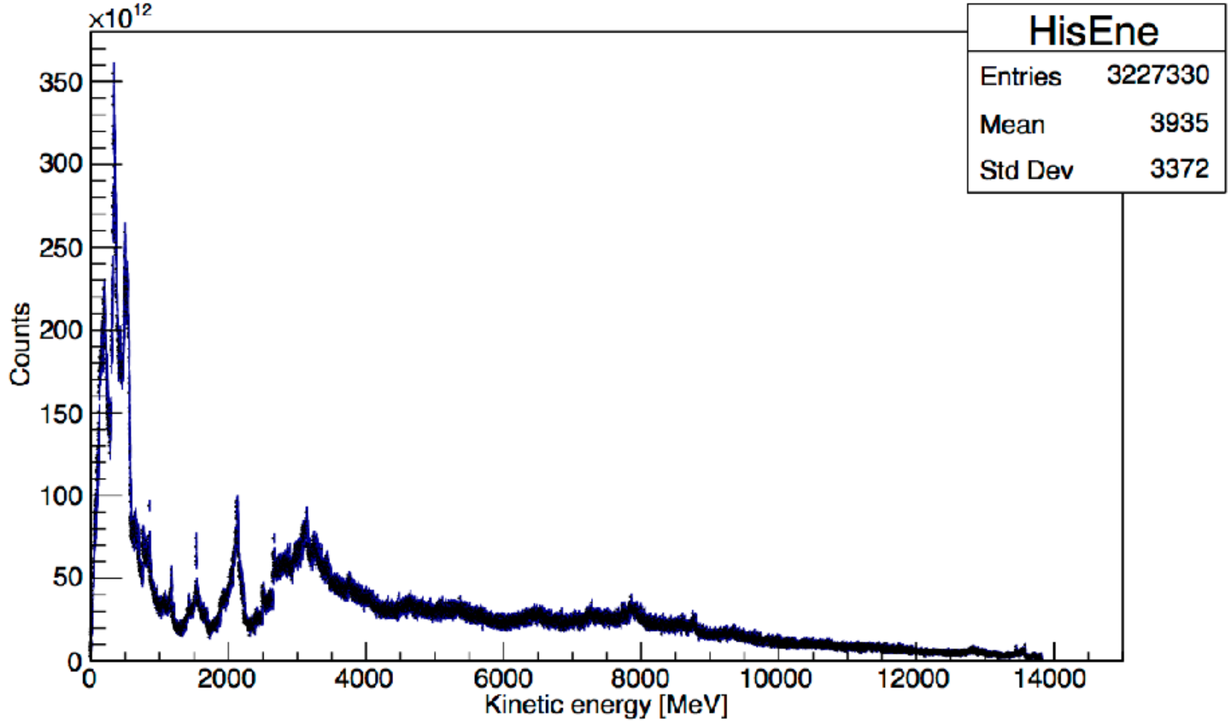
Random number generator

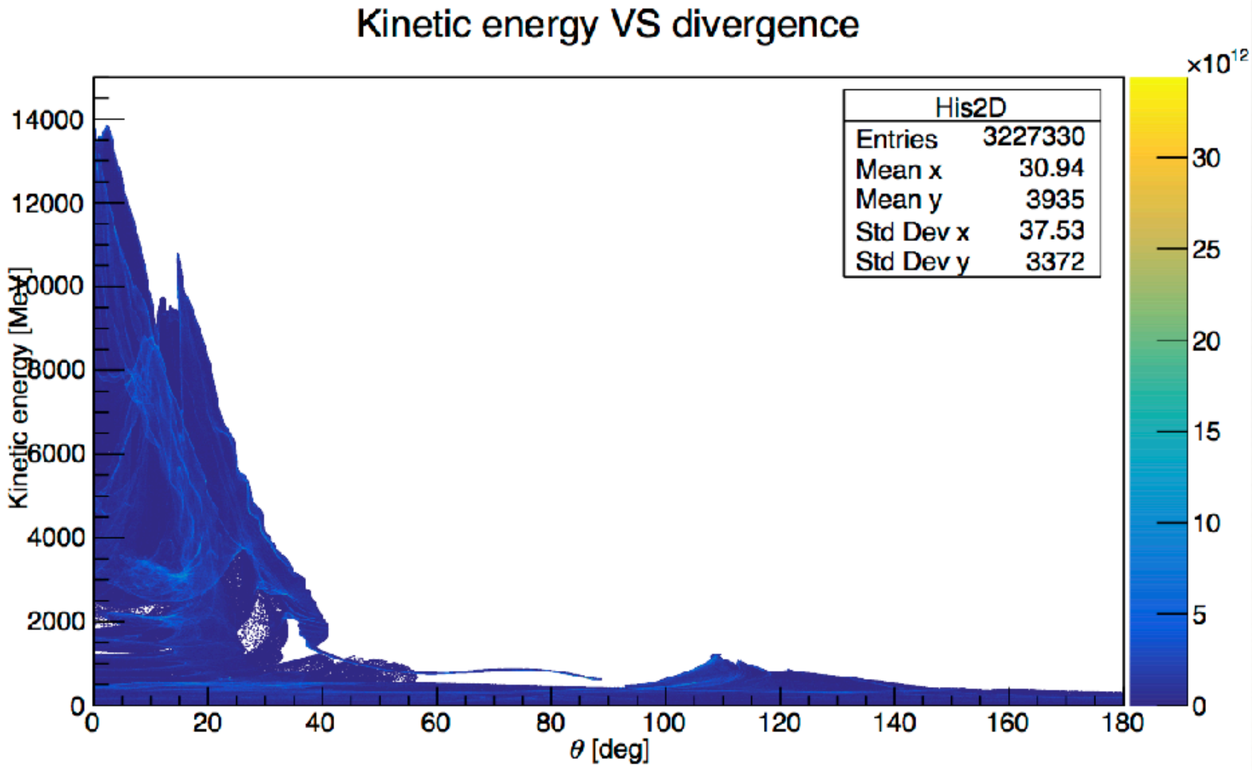
- C++ can make Poisson, Normal distributions and etc.
- Please check the reference of C++
- http://en.cppreference.com/w/cpp/numeric/random
- Of course Geant4 (CLHEP) has more good distribution generator
- Please check the manual of Geant4
- https://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch03s02.html

Using ROOT classes

Using ROOT classes

- Ok, But why?
- Can you make such a distribution by functions in C++ or Geant4?





Using ROOT classes

- ROOT has very nice random number generator by histograms
- TH1::GetRandom(), TH2::GetRandom2(), and TH3::GetRandom3()
- Please check the reference manual of ROOT
- I know, this is not good for before the session about ROOT
- But, It will help you

Using ROOT classes

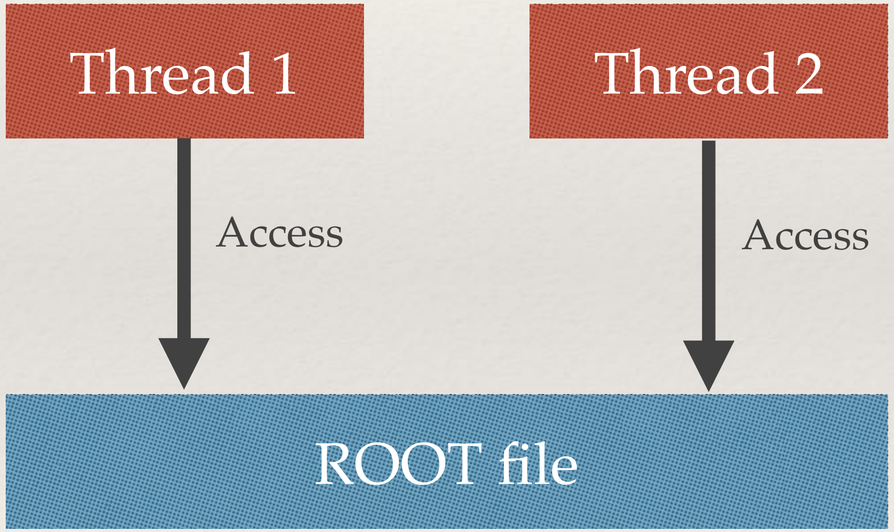
- Open the ROOT file and get the histograms (TH1D and TH2D)
- Attention! When you open the file, be careful!

```
#ifndef MyPrimaryGeneratorAction_h
#define MyPrimaryGeneratorAction_h 1

#include <TH2.h>

class MyPrimaryGeneratorAction: public G4VUserPrimaryGeneratorAction
{
   …
   TH2D *fHisSource2D;
   TH1D *fHisSource1D;
};

#endif
```

```
#include <TFile.h>
#include <G4AutoLock.hh>
#include "MyPrimaryGeneratorAction.hpp"

G4Mutex mutexInPGA = G4MUTEX_INITIALIZER;

MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
   : G4VUserPrimaryGeneratorAction(),
     fParticleGun(nullptr)
{
   …

   G4AutoLock lock(&mutexInPGA);
   TFile *file = new TFile("Data/energy.root", "OPEN");
   fHisSource2D = (TH2D*)file->Get("His2D");
   fHisSource2D->SetDirectory(0);
   fHisSource1D = (TH1D*)file->Get("HisEne");
   fHisSource1D->SetDirectory(0);
   file->Close();
   delete file;

   …

}
```

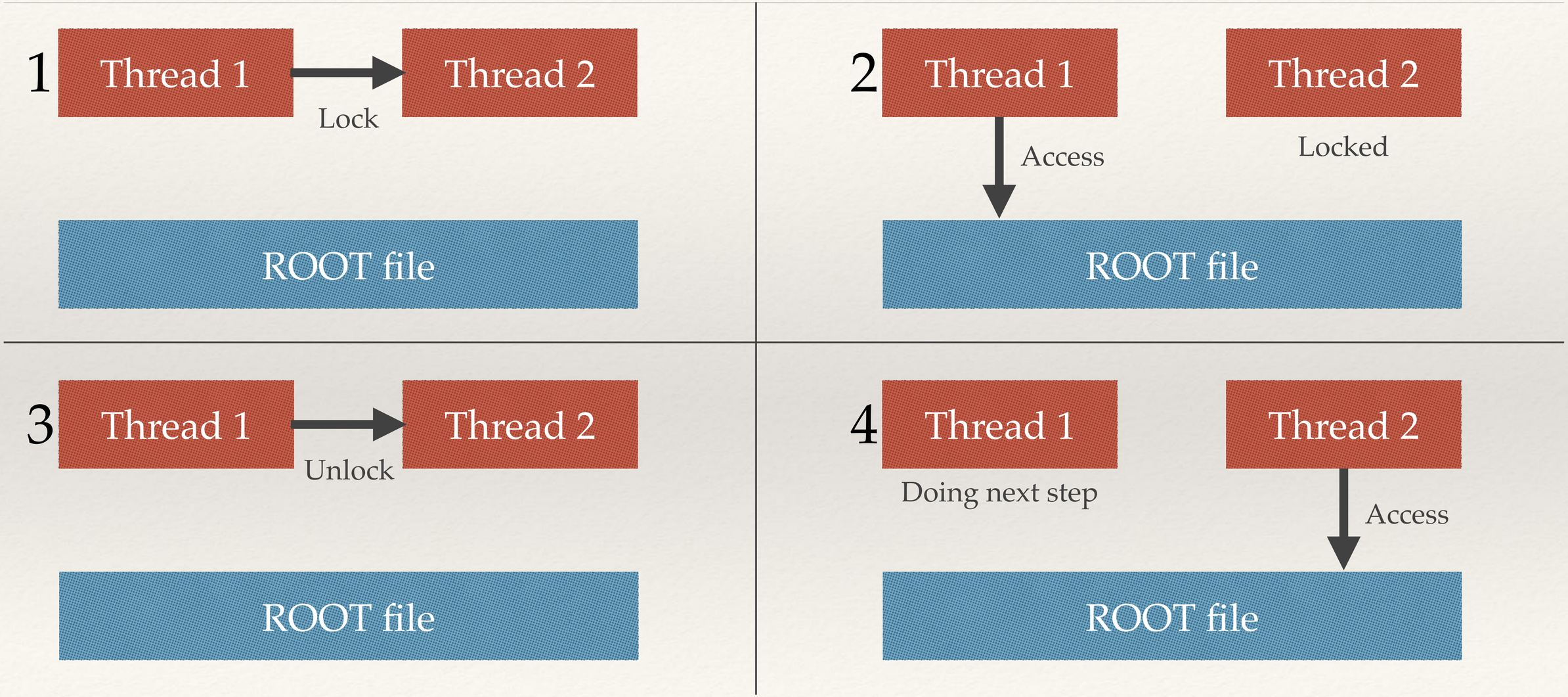
Using ROOT classes

- Usually, some threads accessing to one resource at same time, it is dangerous. If it is not change the file, you should not to do it for safety



- Mutual exclusion is needed

Using ROOT classes



Using ROOT classes

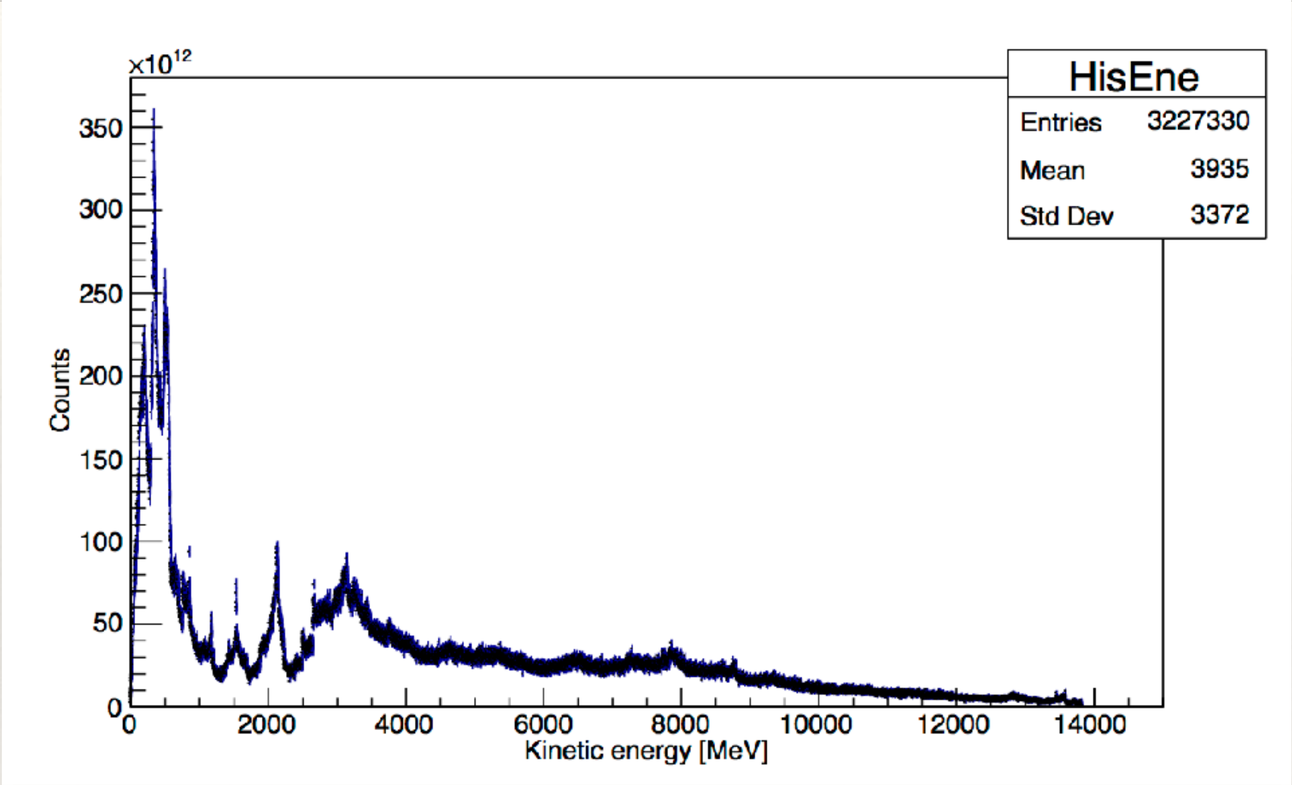
- Use G4AutoLock
  - Lock other threads
  - You can access the resource safely
  - Automatically unlock

```
#include <TFile.h>
#include <G4AutoLock.hh>
#include "MyPrimaryGeneratorAction.hpp"

G4Mutex mutexInPGA = G4MUTEX_INITIALIZER;

MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
   : G4VUserPrimaryGeneratorAction(),
     fParticleGun(nullptr)
{
   …

   G4AutoLock lock(&mutexInPGA);
   TFile *file = new TFile("Data/energy.root", "OPEN");
   fHisSource2D = (TH2D*)file->Get("His2D");
   fHisSource2D->SetDirectory(0);
   fHisSource1D = (TH1D*)file->Get("HisEne");
   fHisSource1D->SetDirectory(0);
   file->Close();
   delete file;

   …

}
```

After "}" unlocked

Using ROOT classes

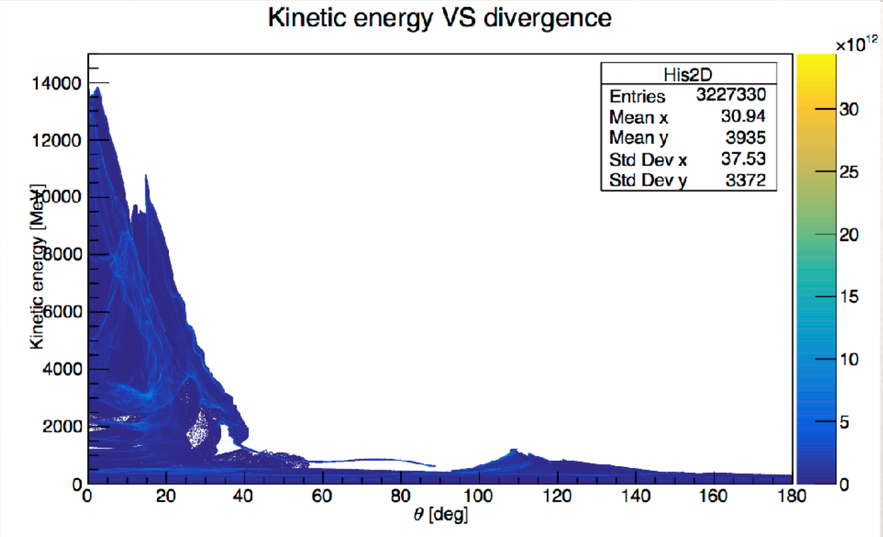
- How to generate the random number by TH1D
- TH1::GetRandom() returns a random number distributed by histogram




```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
    …
    G4double ene;
    ene = fHisSource1D->GetRandom()*MeV;
    fParticleGun->SetParticleEnergy(ene);
    fParticleGun->GeneratePrimaryVertex(event);
    …
}
```

Using ROOT classes

- How to generate the random number by TH2D
- TH2::GetRandom2 takes 2 arguments
- Each argument has the random number distributed by histogram




```
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
   …
   G4double ene;
   G4double theta;
   fHisSource2D->GetRandom2(theta, ene);
   G4double phi = G4UniformRand() * 2.*CLHEP::pi;
   G4double vx = sin(theta) * cos(phi);
   G4double vy = sin(theta) * sin(phi);
   G4double vz = cos(theta);

   fParticleGun->SetParticleEnergy(ene);
   fParticleGun->SetParticleMomentumDirection(G4ThreeVector(vx, vy, vz));
   fParticleGun->GeneratePrimaryVertex(event);
   …
}
```

Using ROOT classes

- Histograms (TH1D and TH2D) are thread safe
- But, a some classes are NOT thread safe (i.e. TF1)
- Incase of NOT thread safe, Don’t forget to use G4AutoLock

Using ROOT classes

- You should write CMakeList.txt to include ROOT headers and libraries
- https://root.cern.ch/how/integrate-root-my-project-cmake
- There are some information
- And also please check my CMakeList.txt
- Don't forget this line

```
ROOT built with classic configure/make use the provided $ROOTSYS/etc/cmake/FindROOT.cmake
```

- Adding include($ROOTSYS/etc/cmake/FindROOT.cmake) in CMakeList.tt

How to define an ion as the primal particle

How to define an ion as the primal particle

- Geant4 has some ions in the particle table
- GenericIon, He3, alpha, deuteron, triton
- But, how can we use carbon ion?

How to define an ion as the primal particle

- You need a trick
- In the constructor, we can not define the NOT listed ion beam
  - If you define it, you will get an error message
- We can define in GeneratePrimaries function

How to define an ion as the primal particle

- At the first shot, ParticleGun assign the C^{6+}

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
   : G4VUserPrimaryGeneratorAction(),
     fParticleGun(nullptr)
{
   fFirstFlag = true;
   fZ = 6;
   fA = 12;
   fIonCharge   = 6.*eplus;
   fExcitEnergy = 0.*keV;
   …
}


void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event *event)
{
   if(fFirstFlag){
      fFirstFlag = false;
      SetIon();
   }
   …
}

void MyPrimaryGeneratorAction::SetIon()
{
   G4ParticleDefinition *ion
      = G4IonTable::GetIonTable()->GetIon(fZ, fA, fExcitEnergy);
   fParticleGun->SetParticleDefinition(ion);
   fParticleGun->SetParticleCharge(fIonCharge);
   G4cout << "Beam changed\tZ = " << fZ
          << ", A = " << fA
          << ", Charge = " << fIonCharge << G4endl;
}
```

Conclusion

- The source term is set at user PrimaryGeneratorAction class
- We can use many kinds of particles
- Now, to use ion beam, we need the trick
- Using ROOT classes helps you to make more complicate source term
- Some ROOT classes are NOT thread safe
- Don’t forget the G4AutoLock

Appendix: How to make a ROOT file including histogram

```
void MakeROOTfile()
{
   TH1D *his1D = new TH1D("his1D", "1D histogram", 1000, 0., 100.);
   TH2D *his2D = new TH2D("his2D", "2D histogram", 180, -90., 90., 1000, 0., 100.);
   for(Int_t i = 0; i < 1000000; i++){
      his1D->Fill(gRandom->Exp(5));
      his2D->Fill(gRandom->Gaus(0., 15.), gRandom->Exp(5));
   }

   TString fileName = "test.root";
   TFile *file = new TFile(fileName, "RECREATE");
   his1D->Write();
   his2D->Write();
   file->Close();
}
```