



# Geant4 @ NCBJ, 2022

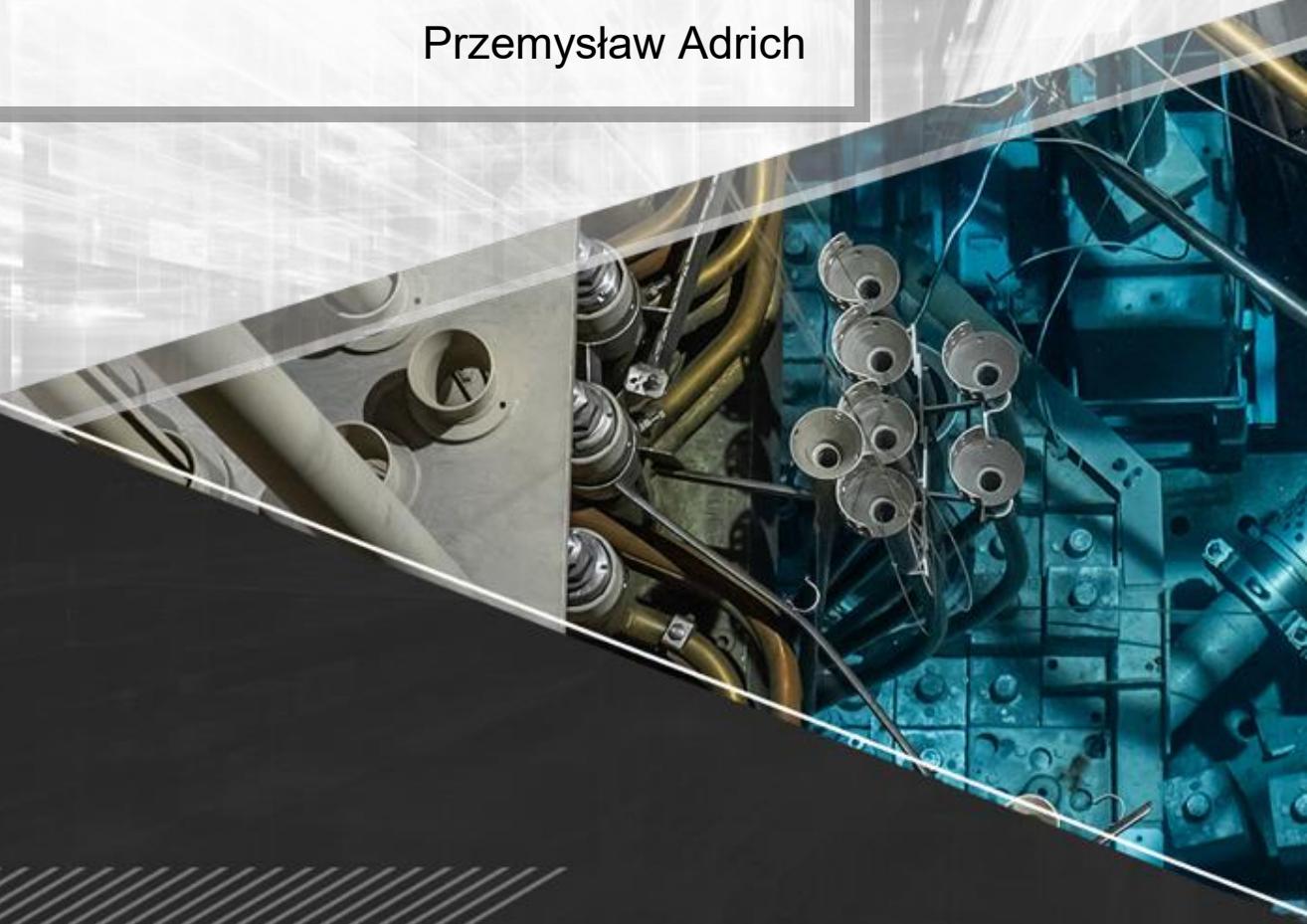
## 6. Wyniki symulacji

### obiekty typu UserAction

Przemysław Adrich



NARODOWE  
CENTRUM  
BADAŃ  
JĄDROWYCH  
ŚWIERK



# Plan kursu

## Wykłady

1. Wstęp do Geant4. Rys historyczny. Zastosowania. Przegląd możliwości. Instalacja. Dokumentacja.
2. Podstawowa struktura kodu. Hierarchie klas. Klasy użytkownika (obowiązkowe, opcjonalne).  
~~Interfejsy. System jednostek. Liczby losowe. Śledzenie przebiegu symulacji („verbosity”).~~
3. Geometria i materiały.
4. Detektory typu „primitive scorer”, „probe”.
5. Detektory użytkownika. Histogramy i n-tuple.
6. **Obiekty typu „UserAction” jako detektory. (Phasespace).**
7. Źródło. Fizyka. Wizualizacja.
8. Niepewność statystyczna w obliczeniach Monte Carlo. Geant4 na klastrze CiŚ.

Przegląd zagadnień pozostawionych na przyszłość:

- wielowątkowość („multithreading”),
- własne interfejsy („messengers”),
- interfejs Roota (histogramy, n-tuple), interfejs python
- redukcja wariancji, „physics biasing”, „event biasing”, „geometrical biasing”,
- fotony optyczne, fizyka hadronowa, procesy i cząstki użytkownika,
- obcięcia energetyczne zależne od cząstki, regionu geometrii,
- zmiany geometrii i detektorów w trakcie wykonania programu,
- pole EM,
- światy równoległe,
- trackInformation, eventInformation, runInformation
- „stacking”,
- fast simulation,
- import geometrii z CAD,
- periodic boundary conditions,
- specjalistyczne kody bazujące na Geant4 (G4Beamline, GAMOS, GATE ...),
- ...

\* „Monte Carlo First Run”  
(wykład bonusowy)

# Geant4 – wprowadzenie. Co jest potrzebne by zbudować aplikację?

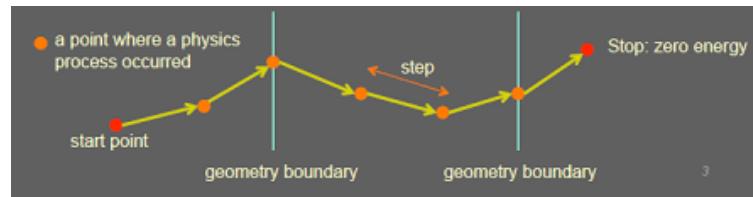
- Geant4 jest zestawem bibliotek i interfejsów. Użytkownik musi zbudować własną aplikację.
- W tym celu trzeba:
  - Zdefiniować układ:
    - geometria, materiały
  - Zdefiniować fizykę, która ma być stosowana w symulacji:
    - Cząstki, procesy/modele procesów fizycznych
    - Progi produkcji
  - Określić od czego mają zaczynać się zdarzenia:
    - Generator cząstek pierwotnych (źródło)
  - **Zadbać o wydobycie użytecznych informacji**
- Opcjonalnie można również:
  - Wizualizować geometrię, trajektorie, wyniki fizyczne
  - Dodać interfejs użytkownika (np. graficzny)
  - Zdefiniować własne komendy, itd., itd.

# Geant4. Dostęp i gromadzenie wyników symulacji

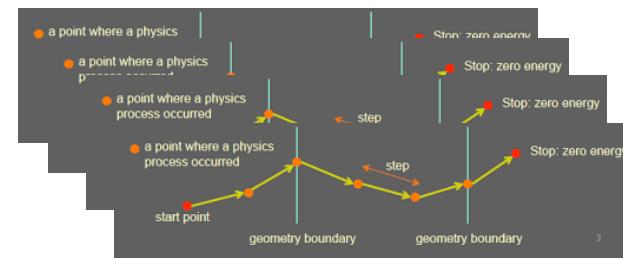
- Standardowo symulacja przebiega bez gromadzenia jakichkolwiek wyników.
- Użytkownik musi sam określić jakie dane i w jaki sposób chce gromadzić.
- Znam 3½ zalecanych sposobów zbierania wyników symulacji:
  1. Proste liczniki (primitive scorer) obsługiwane z poziomu poleceń interfejsu użytkownika.  
Działają w geometrii równoległej, niezależnej od rzeczywistej geometrii modelu zdefiniowanego w G4UserDetectorConstruction.
  2. Zdefiniowanie objętości logicznej (bryły poziomu G4LogicalVolume) jako elementu aktywnego i przypisanie jej detektora:
    - 2a. Implementując własne klasy pochodne od G4VSensitiveDetector i G4VHit.
    - 2b. Stosując gotową klasę G4MultiFunctionalDetector, która jest konkretną implementacją klasy G4VSensitiveDetector opartą o proste liczniki (obiekty typu G4VPrimitiveScore).
- 3. Korzystając z własnych implementacji klas **G4UserTrackingAction**, **G4UserSteppingAction** i **G4UserStackingAction**. W ten sposób również mamy dostęp do pełnej informacji ale sami musimy zadbać o jej przetwarzanie w trakcie procesowania przypadku.

# Geant4 – wprowadzenie. Nomenklatura

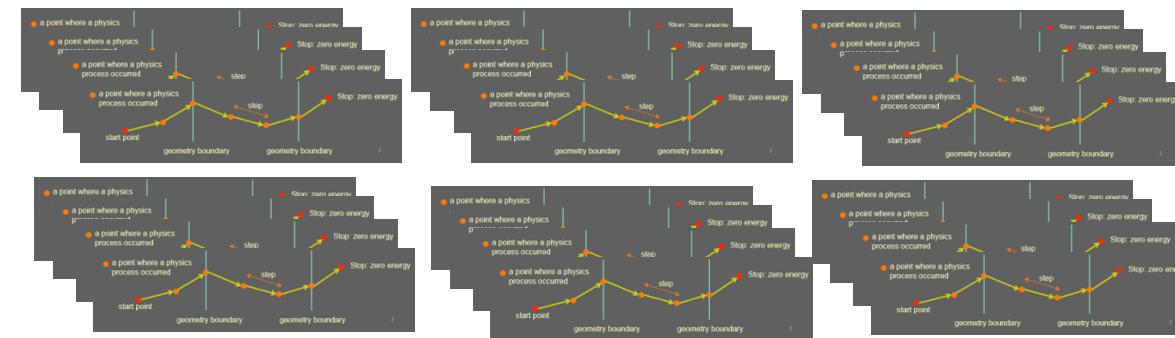
step, track



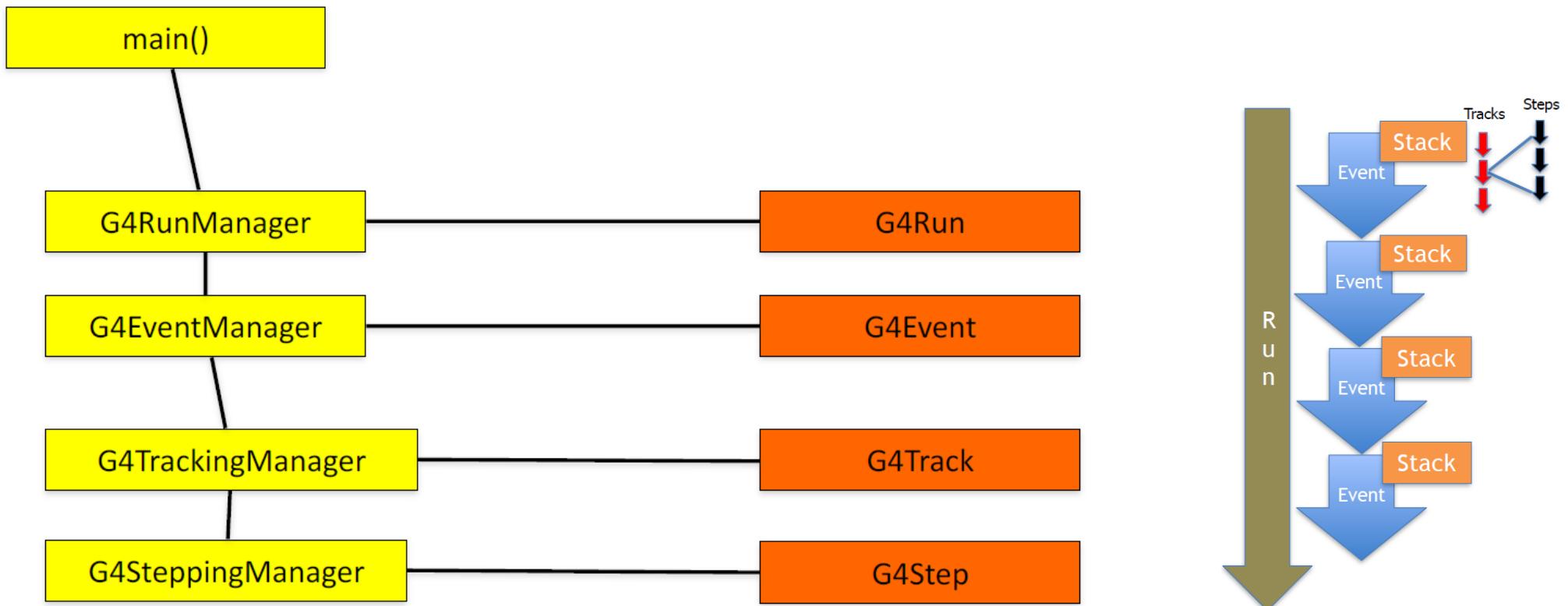
event



run

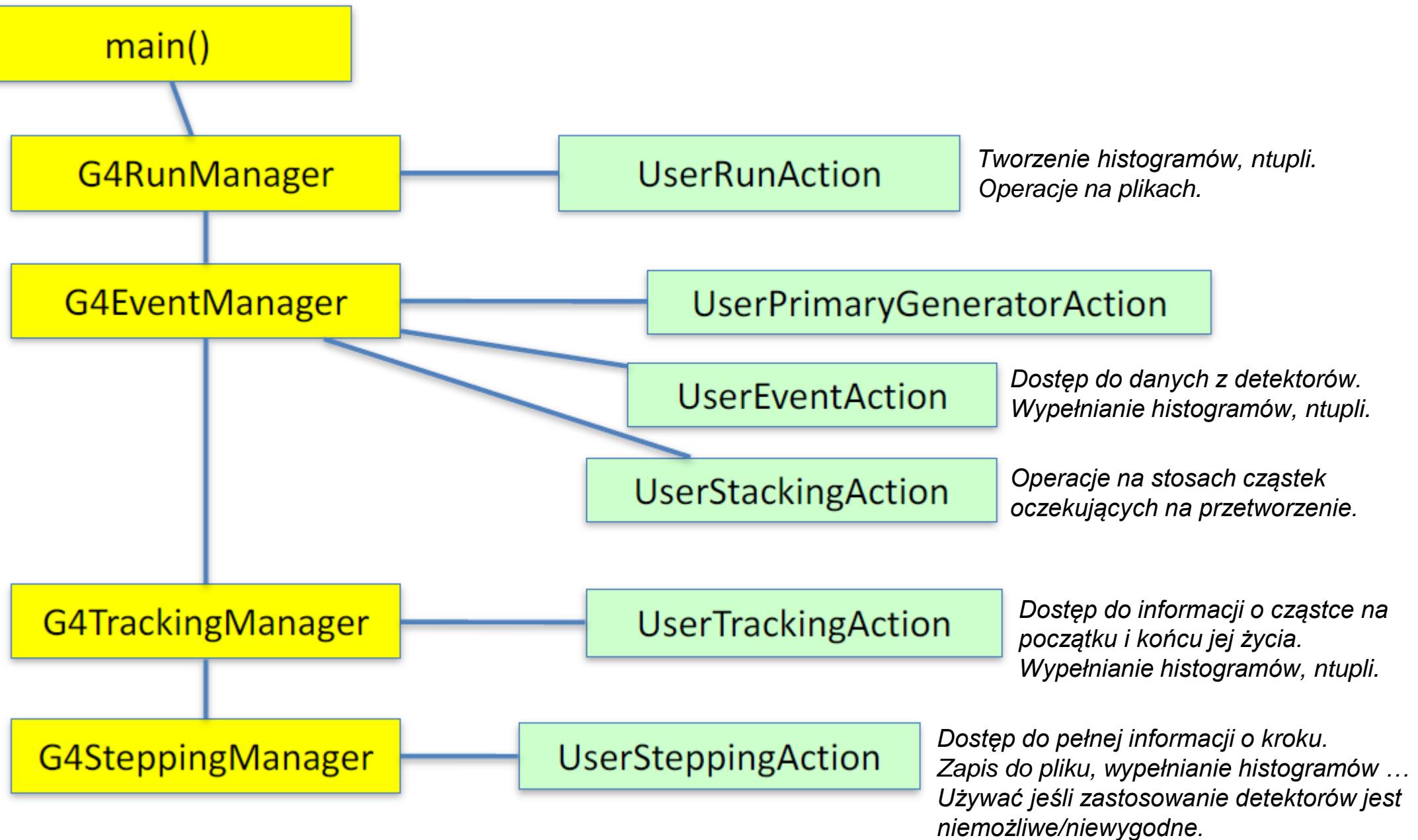


# Geant4 – podstawowa struktura



Zapożyczone od M. Asai za pośrednictwem J.Apostolakis

# Geant4 – podstawowa struktura. Podsumowanie. Klasy użytkownika



## G4UserRunAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserRunAction.html>

## G4VUserPrimaryGeneratorAction Class Reference abstract

<https://geant4.kek.jp/Reference/v11.0.1/classG4VUserPrimaryGeneratorAction.html>

## G4UserEventAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserEventAction.html>

## G4UserStackingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserStackingAction.html>

## G4UserTrackingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserTrackingAction.html>

## G4UserSteppingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserSteppingAction.html>

## G4UserRunAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserRunAction.html>

### Public Member Functions

```
virtual G4Run * GenerateRun ()  
virtual void BeginOfRunAction (const G4Run *aRun)  
virtual void EndOfRunAction (const G4Run *aRun)  
virtual void SetMaster (G4bool val=true)  
G4bool IsMaster () const
```

### Protected Attributes

```
G4bool isMaster = true
```

# G4UserPrimaryGeneratorAction

## G4UserPrimaryGeneratorAction Class Reference abstract

<https://geant4.kek.jp/Reference/v11.0.1/classG4VUserPrimaryGeneratorAction.html>

### Public Member Functions

virtual void **GeneratePrimaries** (G4Event \*anEvent)=0

## G4UserEventAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserEventAction.html>

### Public Member Functions

virtual void **SetEventManager** (G4EventManager \*value)

virtual void **BeginOfEventAction** (const G4Event \*anEvent)

virtual void **EndOfEventAction** (const G4Event \*anEvent)

### Protected Attributes

G4EventManager \* fpEventManager = nullptr

## G4UserStackingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserStackingAction.html>

### Public Member Functions

```
void SetStackManager (G4StackManager *value)
virtual G4ClassificationOfNewTrack ClassifyNewTrack (const G4Track *aTrack)
virtual void NewStage ()
virtual void PrepareNewEvent ()
```

### Protected Attributes

```
G4StackManager * stackManager = nullptr
```

```
enum G4ClassificationOfNewTrack {
    fUrgent=0,      // put into the urgent stack
    fWaiting=1,     // put into the waiting stack
    fPostpone=-1,   // postpone to the next event
    fKill=-9,       // kill
}
```

## G4UserTrackingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserTrackingAction.html>

### Public Member Functions

virtual void **SetTrackingManagerPointer** (G4TrackingManager \*pValue)

virtual void **PreUserTrackingAction** (const G4Track \*)

virtual void **PostUserTrackingAction** (const G4Track \*)

### Protected Attributes

G4TrackingManager \* **fpTrackingManager** = nullptr

## G4UserSteppingAction Class Reference

<https://geant4.kek.jp/Reference/v11.0.1/classG4UserSteppingAction.html>

### Public Member Functions

virtual void **SetSteppingManagerPointer** (G4SteppingManager \*pValue)

virtual void **UserSteppingAction** (const G4Step \*)

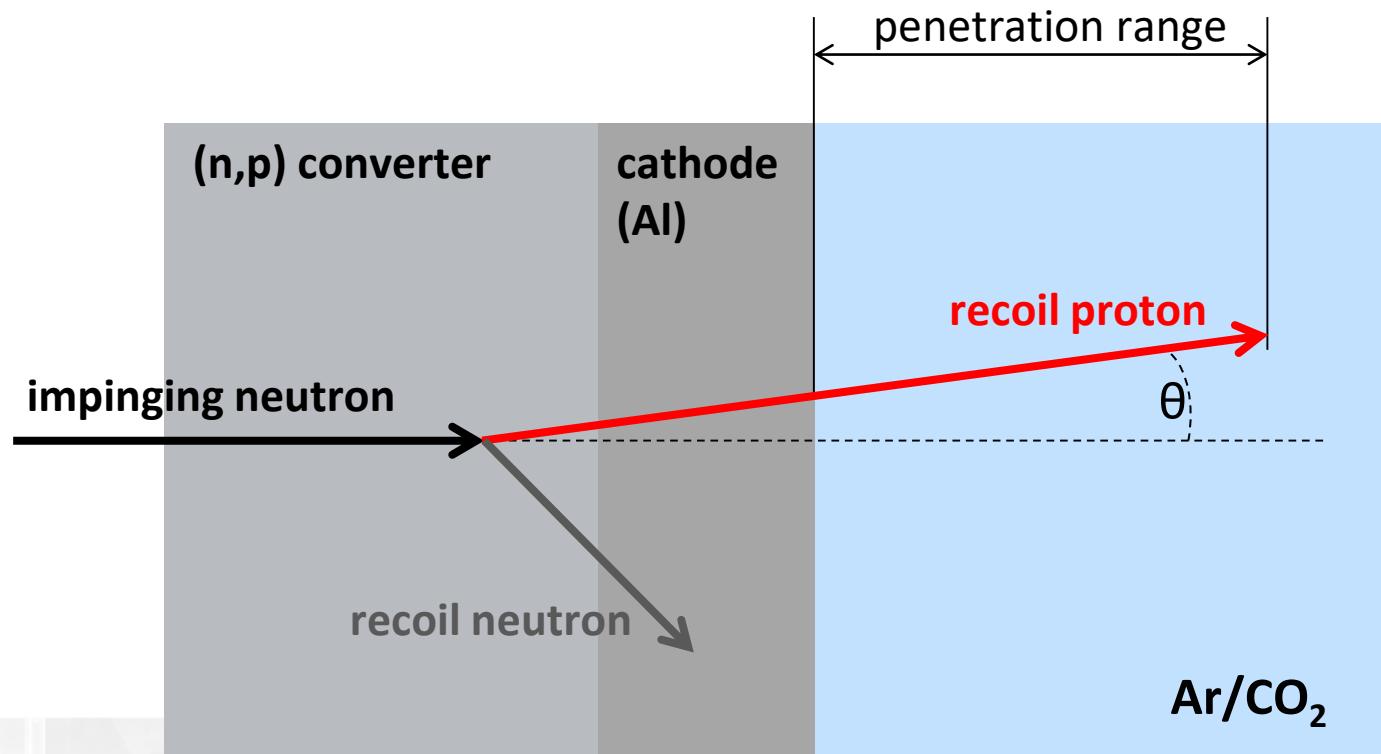
### Protected Attributes

G4SteppingManager \* **fpSteppingManager** = nullptr

# Przykład implementacji detektora w oparciu o UserSteppingAction

Optymalizacja konwertera (*n,p*)

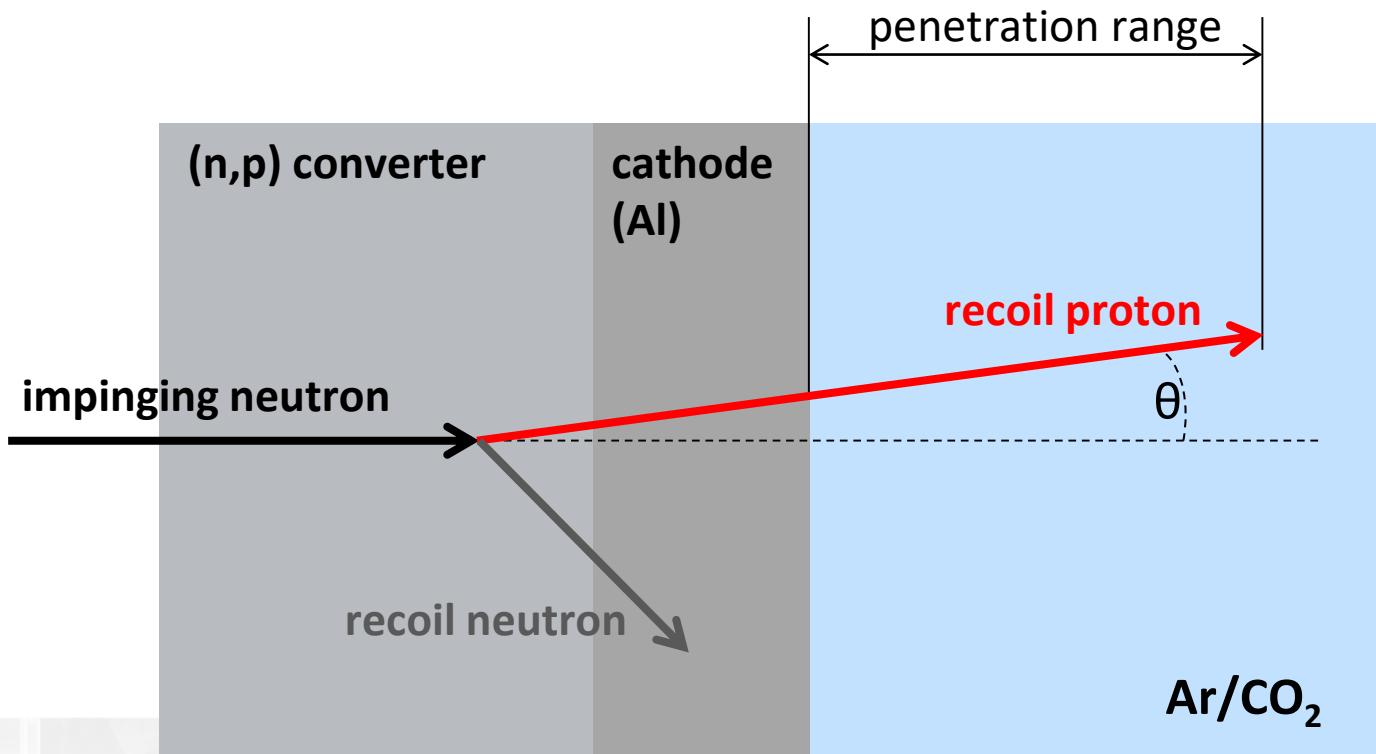
- Znaleźć grubość konwertera, przy której wydajność konwersji jest najwyższa
  - Wystarczy zliczać protony, które wyszły z konwertera w obszar gazu
- Dodatkowo, wyznaczyć rozkład początkowej energii i zasięgu protonów opuszczających konwerter
  - Zapisać energię protonu w momencie wyjścia z katody do gazu
  - Zapisać współrzędną *Z* protonu na końcu jego „życia” (w chwili, w której energia kinetyczna = 0)



# Przykład implementacji detektora w oparciu o UserSteppingAction

Optymalizacja konwertera ( $n,p$ )

- W `UserRunAction::BeginOfRunAction` – otworzyć plik do zapisu danych
- W `G4UserSteppingAction::UserSteppingAction`
  - Czy krok dotyczy protonu? Jeśli tak:
    - Czy proton wychodzi z konwertera w obszar gazu? Tak: zapisać ID i energię do pliku.
    - Czy proton zatrzymał się w obszarze gazu? Tak: zapisać ID i zasięg do pliku.
- W `UserRunAction::EndOfRunAction` – zamknąć plik.



# Przykład implementacji detektora w oparciu o UserSteppingAction

```
#include "G4RunManager.hh"
#include "DetectorConstruction.hh"
#include "ActionInitialization.hh"

int main(int argc,char** argv)
{
    G4RunManager* runManager = new G4RunManager;
    runManager->SetUserInitialization(new DetectorConstruction());
    runManager->SetUserInitialization(PysicsList); ←
// User action initialization
    runManager->SetUserInitialization(new ActionInitialization());
    ...
}
```

Uwaga! Fizyka musi być zainicjalizowana i zarejestrowana do RunManagera przed inicjalizowaniem UserSteppingAction! Fizyką zajmiemy się niebawem.

```
#include "ActionInitialization.hh"
#include "PrimaryGeneratorAction.hh"
#include "RunAction.hh"
#include "SteppingAction.hh"

ActionInitialization::ActionInitialization()
    : G4VUserActionInitialization()
{}

void ActionInitialization::Build() const
{
    SetUserAction(new PrimaryGeneratorAction);
    SetUserAction(new RunAction() );
    SetUserAction(new SteppingAction() );
}
```

# Przykład implementacji detektora w oparciu o UserSteppingAction

## RunAction.hh

```
#ifndef RunAction_h
#define RunAction_h 1
#include "G4UserRunAction.hh"
#include <fstream>
class G4Run;

class RunAction : public G4UserRunAction
{
public:
    RunAction();
    ~RunAction();
    virtual void BeginOfRunAction(const G4Run* );
    virtual void EndOfRunAction(const G4Run* );
    std::ofstream& get_file();
private:
    std::ofstream fPlik;
};

#endif
```

Strumień wyjścia do pliku

## RunAction.cc

```
#include "RunAction.hh"
#include "G4Run.hh"
#include <fstream>

RunAction::RunAction() {}
RunAction::~RunAction() {}

void RunAction::BeginOfRunAction(const G4Run* aRun)
{
    // Otwieramy plik do, którego będzie pisał stepper
    G4int RunID = aRun->GetRunID();
    G4String NazwaPliku = "wyniki-run_" + RunID + ".txt";
    fPlik.open(NazwaPliku);
}

void RunAction::EndOfRunAction(const G4Run* aRun)
{
    // Zamykamy plik, do którego pisał stepper:
    fPlik.close();
}

std::ofstream& RunAction::get_file()
{
    return fPlik;
}
```

# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.hh

```
#ifndef SteppingAction_h
#define SteppingAction_h 1
#include "G4UserSteppingAction.hh"

class SteppingAction : public G4UserSteppingAction
{
public:
    SteppingAction();
    ~SteppingAction();
    virtual void UserSteppingAction(const G4Step*);
};

#endif
```

# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.cc

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
// Sprawdzamy czy mamy do czynienia z protonem:
G4Track* fTrack = aStep->GetTrack();
// G4String ParticleName = fTrack->GetParticleDefinition()->GetParticleName();
// if (ParticleName == "proton") { // Tego NIE ROBIĆ w UserSteppingAction!
//     // Porównywanie stringów jest czasochłonne
```

*W UserSteppingAction robić  
tylko i wyłącznie  
to co jest absolutnie niezbędne*

# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.cc

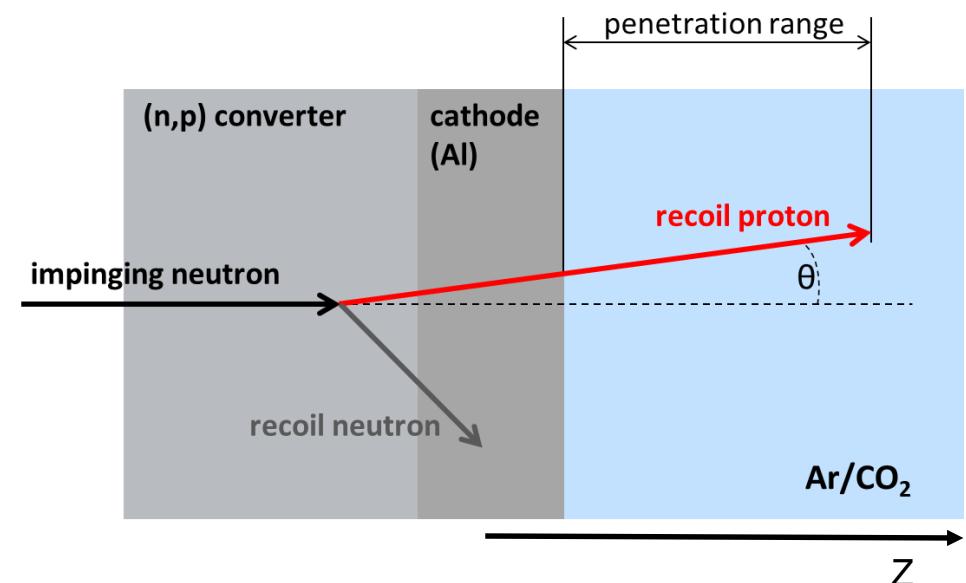
```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
// Sprawdzamy czy mamy do czynienia z protonem:
G4Track* Track = aStep->GetTrack();
G4int pID = Track->GetParticleDefinition()->GetPDGEncoding();
if ( pID == 2212 ) { // proton
```

*Znacznie szybszy sposób sprawdzenia typu cząstki*

# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.cc

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
// Sprawdzamy czy mamy do czynienia z protonem:
G4Track* Track = aStep->GetTrack();
G4int pID = Track->GetParticleDefinition()->GetPDGEncoding();
if ( pID == 2212 ) { // proton // sprawdzamy jego pozycje na początku kroku:
    G4StepPoint* PreStepPoint = aStep->GetPreStepPoint();
    G4double PreStepPositionZ = (PreStepPoint->GetPosition()).z();
    G4double PreStepMomentumZ = (PreStepPoint->GetMomentumDirection()).z();
    DetectorConstruction* DetConst =
        (DetectorConstruction*)(G4RunManager::GetRunManager()->GetUserDetectorConstruction());
    G4double TargetBackPlaneZ = DetConst->GetSlabThickness();
    // czy jest na granicy tarczy i wychodzi z tarczy do obszaru "drift gap"?
    if (PreStepPositionZ == TargetBackPlaneZ && PreStepMomentumZ > 0.0) {
```



# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.cc

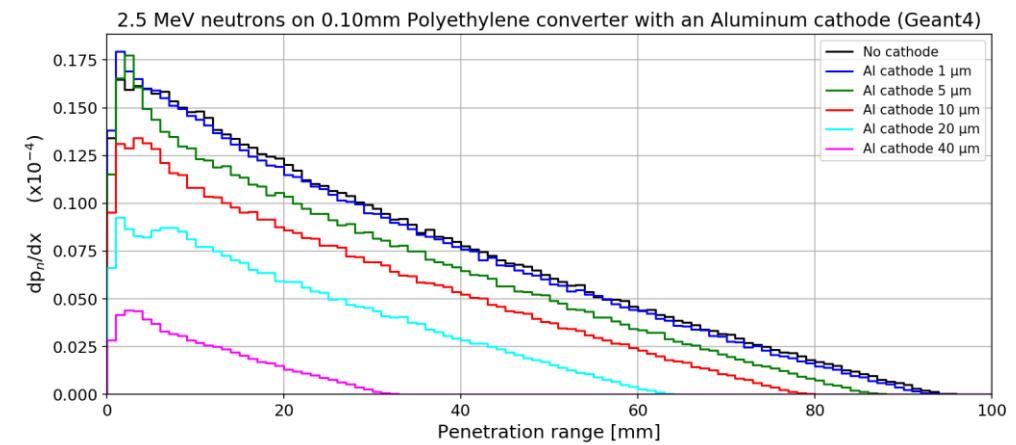
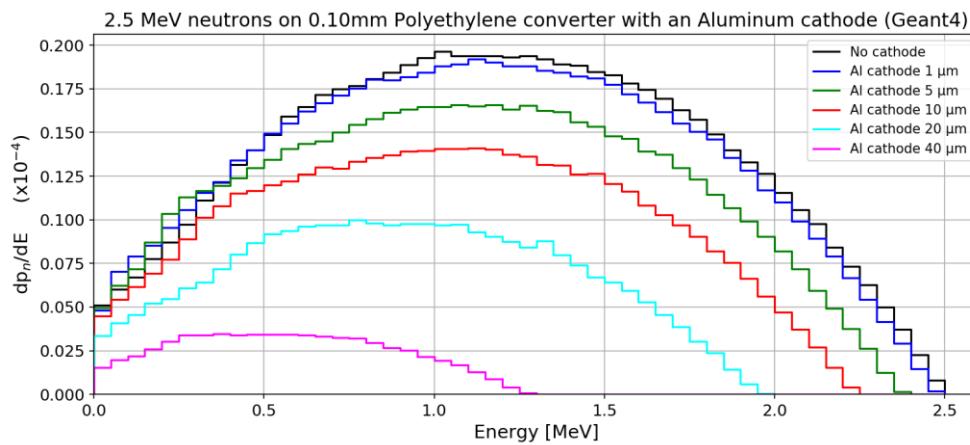
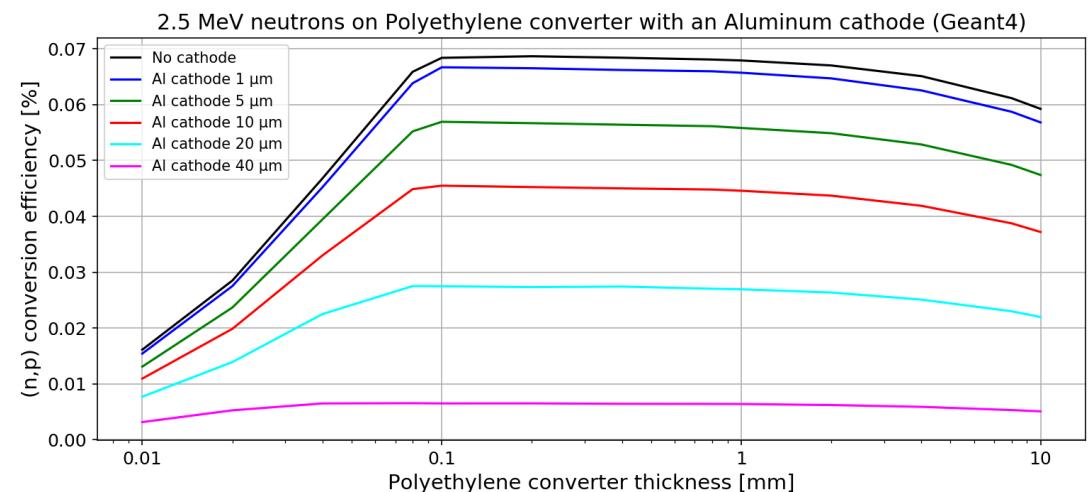
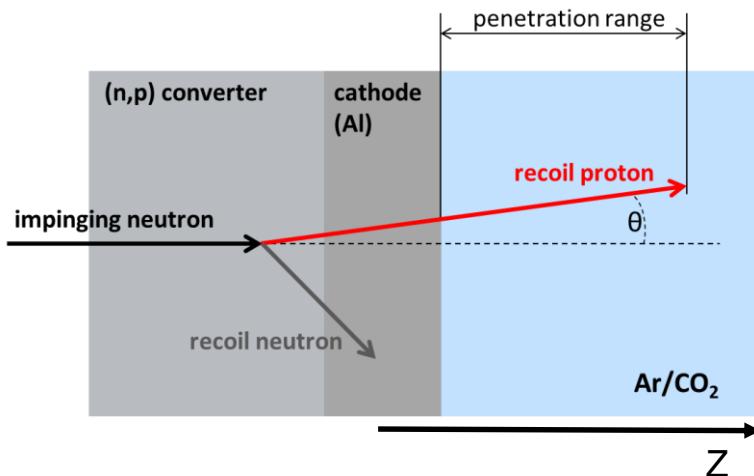
```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
// Sprawdzamy czy mamy do czynienia z protonem:
G4Track* Track = aStep->GetTrack();
G4int pID = Track->GetParticleDefinition()->GetPDGEncoding();
if ( pID == 2212 ) { // proton // sprawdzamy jego pozycje na początku kroku:
    G4StepPoint* PreStepPoint = aStep->GetPreStepPoint();
    G4double PreStepPositionZ = (PreStepPoint->GetPosition()).z();
    G4double PreStepMomentumZ = (PreStepPoint->GetMomentumDirection()).z();
    DetectorConstruction* DetConst =
        (DetectorConstruction*)(G4RunManager::GetRunManager()->GetUserDetectorConstruction());
    G4double TargetBackPlaneZ = DetConst->GetSlabThickness();
    // czy jest na granicy tarczy i wychodzi z tarczy do obszaru "drift gap"?
    if (PreStepPositionZ == TargetBackPlaneZ && PreStepMomentumZ > 0.0) {
        // mamy proton wychodzący z tarczy. Zapisujemy do pliku jego energię.
        const G4Event* evt = G4EventManager::GetEventManager()->GetConstCurrentEvent();
        G4int EventID = evt->GetEventID();
        G4int TrackID = Track->GetTrackID();
        G4double PreStepEnergy = PreStepPoint->GetKineticEnergy();
        RunAction* UserRunAction = (RunAction*)(G4RunManager::GetRunManager()->GetUserRunAction());
        std::ofstream& plik = UserRunAction->get_file();
        plik << EventID << " " << TrackID << " 0 " << PreStepEnergy << G4endl;
    } else {
        // Sprawdzamy czy może jest już w "drift gap" i wyhamował:
```

# Przykład implementacji detektora w oparciu o UserSteppingAction

## SteppingAction.cc

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{ // Sprawdzamy czy mamy do czynienia z protonem:
  G4Track* Track = aStep->GetTrack();
  G4int pID = Track->GetParticleDefinition()->GetPDGEncoding();
  if ( pID == 2212 ) { // proton // sprawdzamy jego pozycje na początku kroku:
    G4StepPoint* PreStepPoint = aStep->GetPreStepPoint();
    G4double PreStepPositionZ = (PreStepPoint->GetPosition()).z();
    G4double PreStepMomentumZ = (PreStepPoint->GetMomentumDirection()).z();
    DetectorConstruction* DetConst =
      (DetectorConstruction*)(G4RunManager::GetRunManager()->GetUserDetectorConstruction());
    G4double TargetBackPlaneZ = DetConst->GetSlabThickness();
    // czy jest na granicy tarczy i wychodzi z tarczy do obszaru "drift gap"?
    if (PreStepPositionZ == TargetBackPlaneZ && PreStepMomentumZ > 0.0) {
      // mamy proton wychodzący z tarczy. Zapisujemy do pliku jego energię.
      const G4Event* evt = G4EventManager::GetEventManager()->GetConstCurrentEvent();
      G4int EventID = evt->GetEventID();
      G4int TrackID = Track->GetTrackID();
      G4double PreStepEnergy = PreStepPoint->GetKineticEnergy();
      RunAction* UserRunAction = (RunAction*)(G4RunManager::GetRunManager()->GetUserRunAction());
      std::ofstream& plik = UserRunAction->get_file();
      plik << EventID << " " << TrackID << " 0 " << PreStepEnergy << G4endl;
    } else {
      // Sprawdzamy czy może jest już w "drift gap" i wyhamował:
      G4StepPoint* PostStepPoint = aStep->GetPostStepPoint();
      G4double PostStepPositionZ = (PostStepPoint->GetPosition()).z();
      G4double PostStepEnergy = PostStepPoint->GetKineticEnergy();
      if (PostStepPositionZ > TargetBackPlaneZ && PostStepEnergy == 0) {
        // mamy proton zatrzymany w obszarze drift gap. Zapisujemy do pliku końcowe położenie w osi Z.
        const G4Event* evt = G4EventManager::GetEventManager()->GetConstCurrentEvent();
        G4int EventID = evt->GetEventID();
        G4int TrackID = Track->GetTrackID();
        RunAction* UserRunAction = (RunAction*)(G4RunManager::GetRunManager()->GetUserRunAction());
        std::ofstream& plik = UserRunAction->get_file();
        plik << EventID << " " << TrackID << " 1 " << PostStepPositionZ-TargetBackPlaneZ << G4endl;
      }
    }
  }
}
```

# Przykład implementacji detektora w oparciu o UserSteppingAction



# Przykład 2 – Phasespace file

W fizyce medycznej (szczególnie w radioterapii) stosuje się tzw. phasespace file (phsp).

Phsp jest to zbiór danych o parametrach N cząstek. Często chodzi o cząstki z głowicy akceleratora na płaszczyźnie pacjenta. Dane często pochodzą z symulacji i są stosowane np. w planowaniu leczenia.



Przykładowy format pliku phsp wg. standardu IAEA.  
<https://www-nds.iaea.org/phsp/phsp.htmlx>

Jak zrobić phsp w Geant4?

Variable	Meaning	Type of variable
X	First coordinate (usually X position in cm)	Real*4
Y	Second coordinate (usually Y position in cm)	Real*4
Z	Third coordinate (usually Z position in cm)	Real*4
U	First direction cosine	Real*4
V	Second direction cosine	Real*4
E	Kinetic energy in MeV	Real*4
Statistical_Weight	Particle statistical weight	Real*4
Particle_type	Type of the particle <i>Current list:</i> photons, electrons, positrons, protons and neutrons	Integer*2
Sign_of_W	Sign of W (Third direction cosine)	Logical*1
Is_new_history	Signifies if particle belongs to new history	Logical*1
Integer_extra	Extra storage space for integer variables <i>Currently defined variables:</i> Incremental history number EGS LATCH PENELOPE ILB	n*(Integer*4) (n ≥ 0)
Float_extra	Extra storage space for real variables <i>Currently defined variables</i> XLAST YLAST ZLAST	m*(Real*4) (m ≥ 0)

# Przykład 2 – Phasespace file

## Jak zrobić phsp w Geant4?

- Zwykle chcemy zapisywać pozycję, pęd, energię cząstek przecinających konkretną płaszczyznę  $z = z_0$
- UserSteppingAction nadaje się do tego idealnie.

## Przykład 2 – Phasespace file

```
void SteppingAction::UserSteppingAction(const G4Step* aStep)
{
    G4Track* Track = aStep->GetTrack();
    G4int pID = Track->GetParticleDefinition()->GetPDGEncoding();
    if ( pID == 11 || pID == -11 ) { // 11 - elektron; -11 pozyton
        G4StepPoint* PreStepPoint = aStep->GetPreStepPoint();
        G4StepPoint* PostStepPoint = aStep->GetPostStepPoint();
        G4ThreeVector preR = PreStepPoint->GetPosition();
        G4ThreeVector postR = PostStepPoint->GetPosition();
        G4double preZ = preR.z();
        G4double postZ = postR.z();
        if (preZ >= zStop && postZ <= zStop ) { // mamy cząstkę, która przecina płaszczyznę  $z = z_{\text{Stop}}$ 
            G4double preE = PreStepPoint->GetKineticEnergy();
            G4double postE = PostStepPoint->GetKineticEnergy();
            G4double kinEnergyMeV = (preE+(postE-preE)*(zStop-preZ)/(postZ-preZ))/MeV;
            // Position
            G4double preX = preR.x();
            G4double postX = postR.x();
            G4double preY = preR.y();
            G4double postY = postR.y();
            G4double x = (preX+(postX-preX)*(zStop-preZ)/(postZ-preZ))/cm;
            G4double y = (preY+(postY-preY)*(zStop-preZ)/(postZ-preZ))/cm;
            G4double z = zStop/cm;
            // Momentum direction
            G4ThreeVector momDir = PreStepPoint->GetMomentumDirection();
            G4double u = momDir.x();
            G4double v = momDir.y();
            G4double w = momDir.z();
            // Track statistical weight
            G4double wt = aTrack->GetWeight();
            // Store particle:
            write_particle(pID, kinEnergyMeV, x, y, z, u, v, w, wt);
        }
    }
}
```

Uproszczony przykład implementacji  
Kompletny: <http://www-nds.iaea.org/phsp>

# Przykład 3 – G4 Basic Example 4a, 4b

[https://geant4-userdoc.web.cern.ch/Doxygen/examples\\_doc/html/ExampleB4.html](https://geant4-userdoc.web.cern.ch/Doxygen/examples_doc/html/ExampleB4.html)

## Variant a: User Actions

These 4 quantities are data members of the [B4a::EventAction](#) class. They are collected step by step in [B4a::SteppingAction::UserSteppingAction\(\)](#), and passed to the event action via two methods: [B4a::EventAction::AddAbs\(\)](#) and [B4a::EventAction::AddGap\(\)](#). In [B4a::EventAction::EndOfEventAction\(\)](#), these quantities are printed and filled in H1D histograms and ntuple to accumulate statistic and compute dispersion.

## Variant b: User data object

In order to avoid dependencies between action classes, a user object [B4b::RunData](#), derived from [G4Run](#), is defined with data members needed for the accounted information. In order to reduce the number of data members a 2-dimensions array is introduced for each quantity. Then the quantities are collected step by step in user action classes: [B4b::SteppingAction::UserSteppingAction\(\)](#) and [B4b::EventAction::EndOfEventAction\(\)](#) in a similar way as in variant a.

## Variant c: Hits and Sensitive detectors

In this option, the physics quantities are accounted using the hits and sensitive detectors framework defined in the Geant4 kernel. The physics quantities are stored in [B4c::CalorHit](#) via two [B4c::CalorimeterSD](#) objects, one associated with the Absorber volume and another one with Gap in [B4c::DetectorConstruction::ConstructSDandField\(\)](#).

In contrary to the [B2](#) example (Tracker) where a new hit is created with each track passing the sensitive volume (in the calorimeter), only one hit is created for each calorimeter layer and one more hit to account for the total quantities in all layers. In addition to the variants a and b, the quantities per each layer are also available in addition to the total quantities.

## Variant d: Scorer

In this option, the Geant4 scorers which are defined on the top of hits and sensitive detectors Geant4 framework are used. In practice this means that the user does not need to define hits and sensitive detector classes but rather uses the classes already defined in Geant4. In this example, the [G4MultiFunctionalDetector](#) with [G4PSEnergyDeposit](#) and [G4PSTrackLength](#) primitive scores are used (see [B4d::DetectorConstruction::ConstructSDandField\(\)](#)).

The scorers hits are saved in form of ntuples in a Root file using Geant4 analysis tools. This feature is activated in the main () function with instantiating [G4TScoreNtupleWriter](#).

Also with this approach, the quantities per each layer are available in addition to the total quantities.

[https://geant4-userdoc.web.cern.ch/Doxygen/examples\\_doc/html/group\\_\\_B4a.html](https://geant4-userdoc.web.cern.ch/Doxygen/examples_doc/html/group__B4a.html)

[https://geant4-userdoc.web.cern.ch/Doxygen/examples\\_doc/html/group\\_\\_B4b.html](https://geant4-userdoc.web.cern.ch/Doxygen/examples_doc/html/group__B4b.html)

<https://en.cppreference.com/w/cpp/language/range-for>

*Dziękuję za uwagę*



NARODOWE  
CENTRUM  
BADAŃ  
JĄDROWYCH  
ŚWIERK

[www.ncbj.gov.pl](http://www.ncbj.gov.pl)

