

Цикл обработки событий

Моделирование в Geant4 начинается с запуска (**Run**).

Свойства геометрии и физических процессов фиксируются ядром Geant4 и становятся неизменными.

Пользователь указывает, сколько событий (**Event**) он хочет моделировать в данном запуске.

Запуск начинается с момента старта первого события, а заканчивается моделированием всех вторичных частиц последнего.

События состоят из треков (**Track**), моделирование события заканчивается с моделированием последнего трека вторичной частицы в данном событии.

Треки моделируются по одному, независимо и последовательно.

Последовательность моделирования треков задается с помощью реализации стека треков.

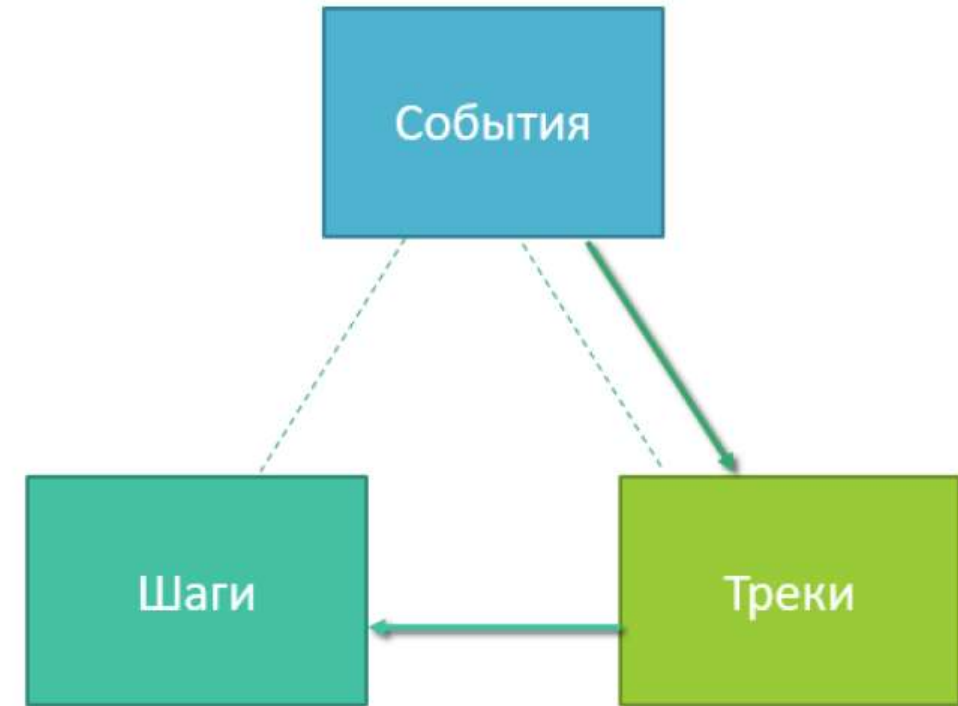
Трек содержит все свойства частицы в процессе моделирования прохождения сквозь вещество.

Каждый трек «знает» имя частицы, хранит информацию о её времени жизни и изменении кинетической энергии.

Все треки состоят из шагов (**Step**).

Шаг — это мельчайшая единица моделирования.

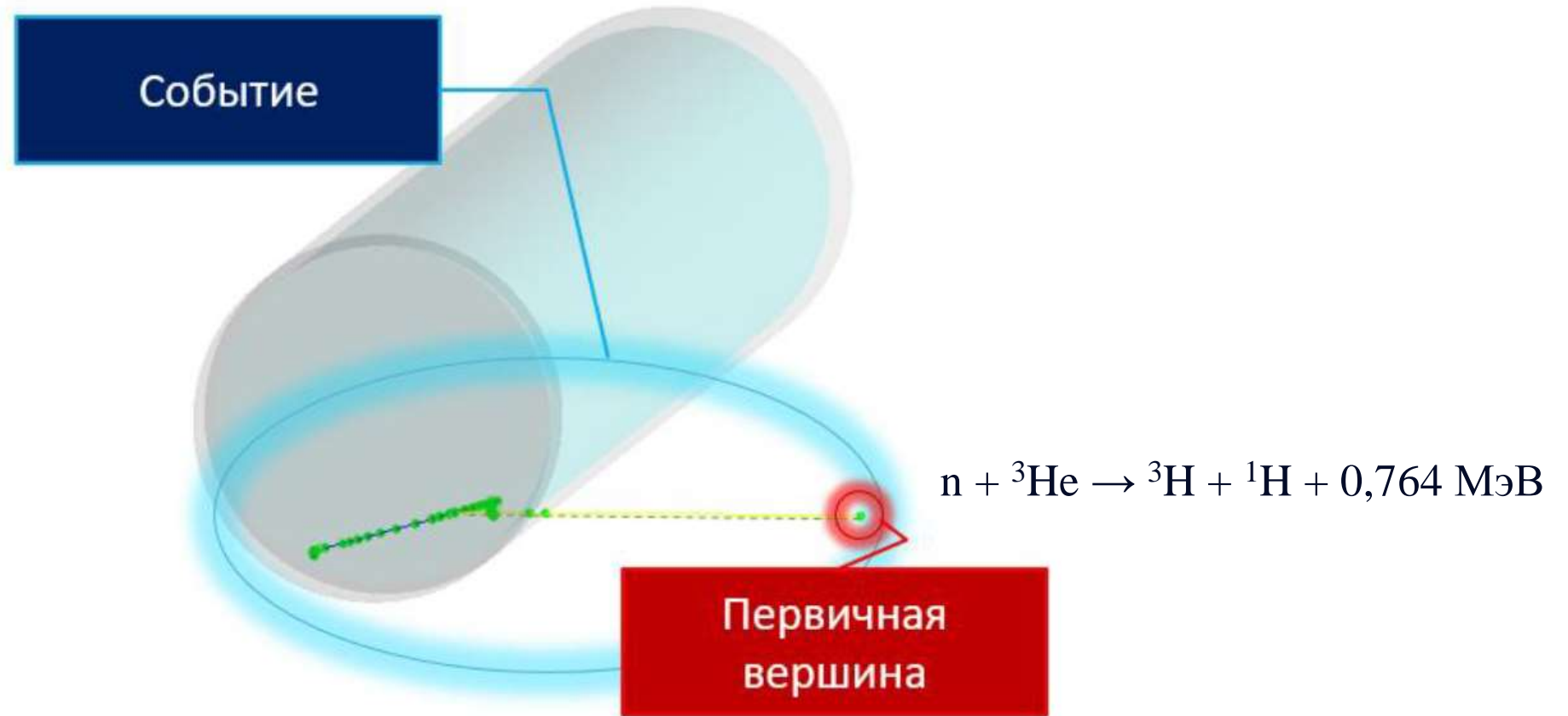
Шаг характеризует расстояние (время), за которое произошло изменение состояния частицы (переход из одного объема в другой, потери энергии на ионизацию, вылет за исследуемую область моделирования).



Цикл обработки событий

Событие — представляет собой единичный цикл от зарождения первичной частицы, до окончания отслеживания последней вторичной частицы.

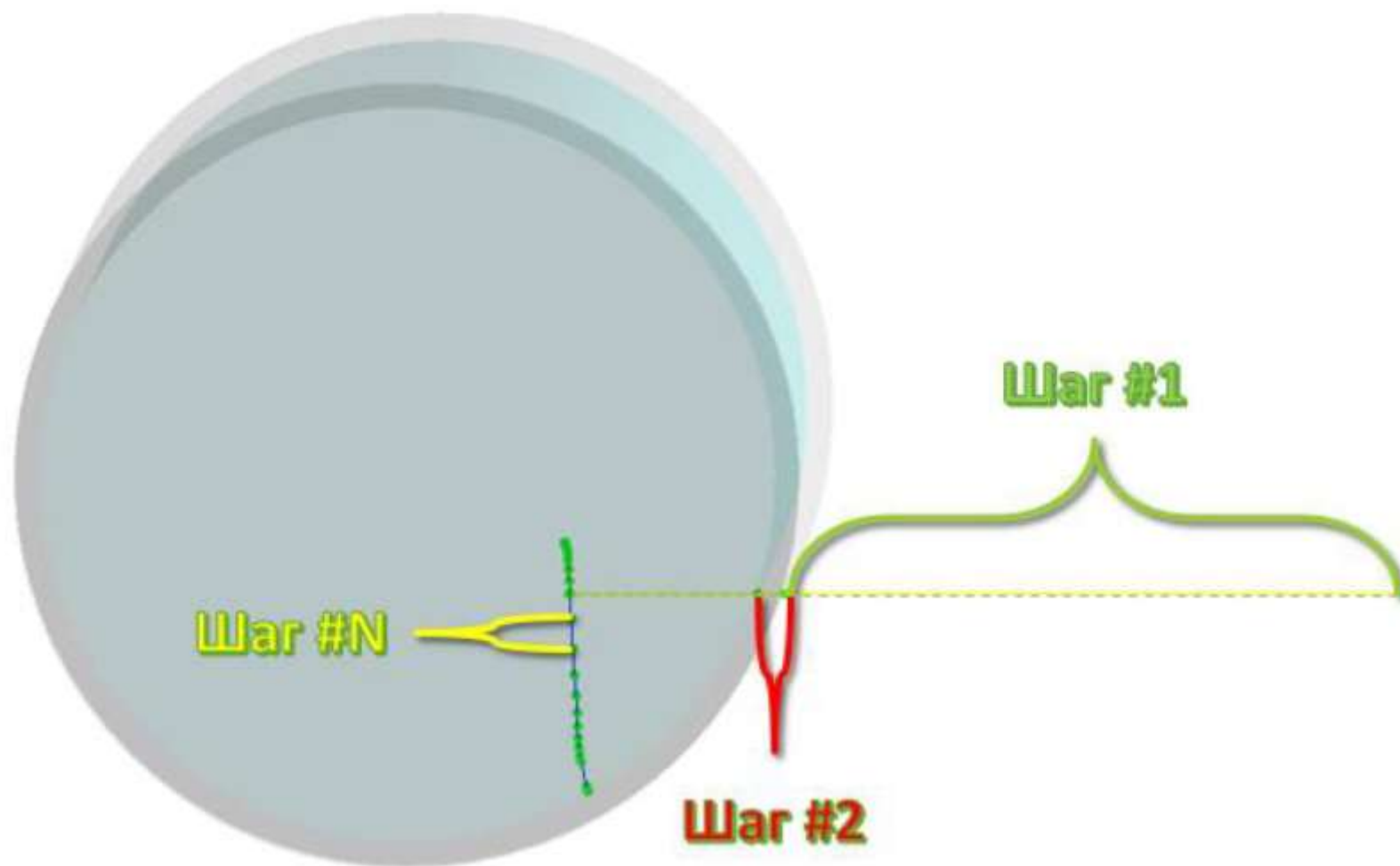
В Geant4 все события рассматриваются атомарно, т.е. независимо от остальных событий.



Нейтрон взаимодействующий в пропорциональном счетчике нейтронов с радиатором из изотопа гелия 3.
В «событие» входят: весь трек нейтрона, треки протона и трития, а так же треки всех образованных ими вторичных частиц

Цикл обработки событий

Шаг — это минимальный элемент цикла обработки событий.



Цикл обработки событий

Пользователь напрямую не управляет ни одной из условных единиц моделирования, однако ему предоставлены классы действий, связанные со своей соответствующей единицей.

В рамках данных классов пользователь может анализировать и аккумулировать информацию в процессе моделирования, вносить некоторые корректировки в процесс моделирования.

Класс `G4VUserActionInitialization`: создание объектов классов действий.

- `G4VUserPrimaryGeneratorAction`
 - `G4UserRunAction`
 - `G4UserEventAction`
 - `G4UserStackingAction`
 - `G4UserTrackingAction`
 - `G4UserSteppingAction`
- Обязательная реализация (генерация первичной вершины)
- Чисто виртуальный метод

Public Member Functions

<code>G4VUserActionInitialization ()</code>
<code>virtual ~G4VUserActionInitialization ()</code>
<code>virtual void Build () const =0</code>
<code>virtual void BuildForMaster () const</code>
<code>virtual G4VSteppingVerbose * InitializeSteppingVerbose () const</code>

Protected Member Functions

<code>void SetUserAction (G4VUserPrimaryGeneratorAction *) const</code>
<code>void SetUserAction (G4UserRunAction *) const</code>
<code>void SetUserAction (G4UserEventAction *) const</code>
<code>void SetUserAction (G4UserStackingAction *) const</code>
<code>void SetUserAction (G4UserTrackingAction *) const</code>
<code>void SetUserAction (G4UserSteppingAction *) const</code>

является чисто техническим классом для определения
порядка обработки треков

TrackStack.cc (основной файл)

DataWriter.cc (DataWriter.hh)

Loader.cc (Loader.hh)

Geometry.cc (Geometry.hh)

Action.cc (Action.hh)

PrimaryPart.cc (PrimaryPart.hh)

RunAct.cc (RunAct.hh)

EventAct.cc (EventAct.hh)

TrackAct.cc (TrackAct.hh)

StackAct.cc (StackAct.hh)

StepAct.cc (StepAct.hh)

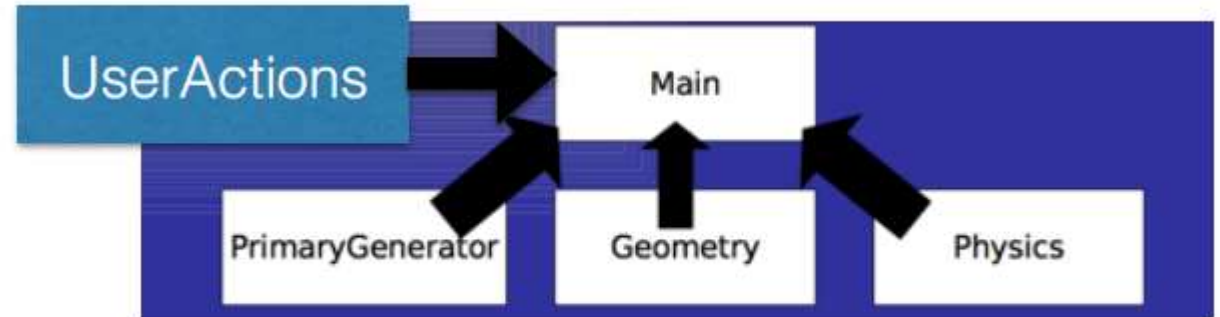
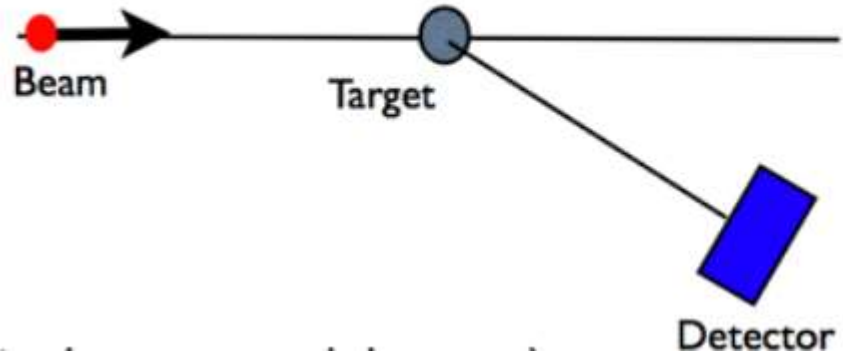
Action.hh

```
class Action: public G4VUserActionInitialization
{
    public: std::ofstream *f_act;
    public: Action(std::ofstream&);
    ~Action();
    virtual void Build() const;
};
```

Action.cc

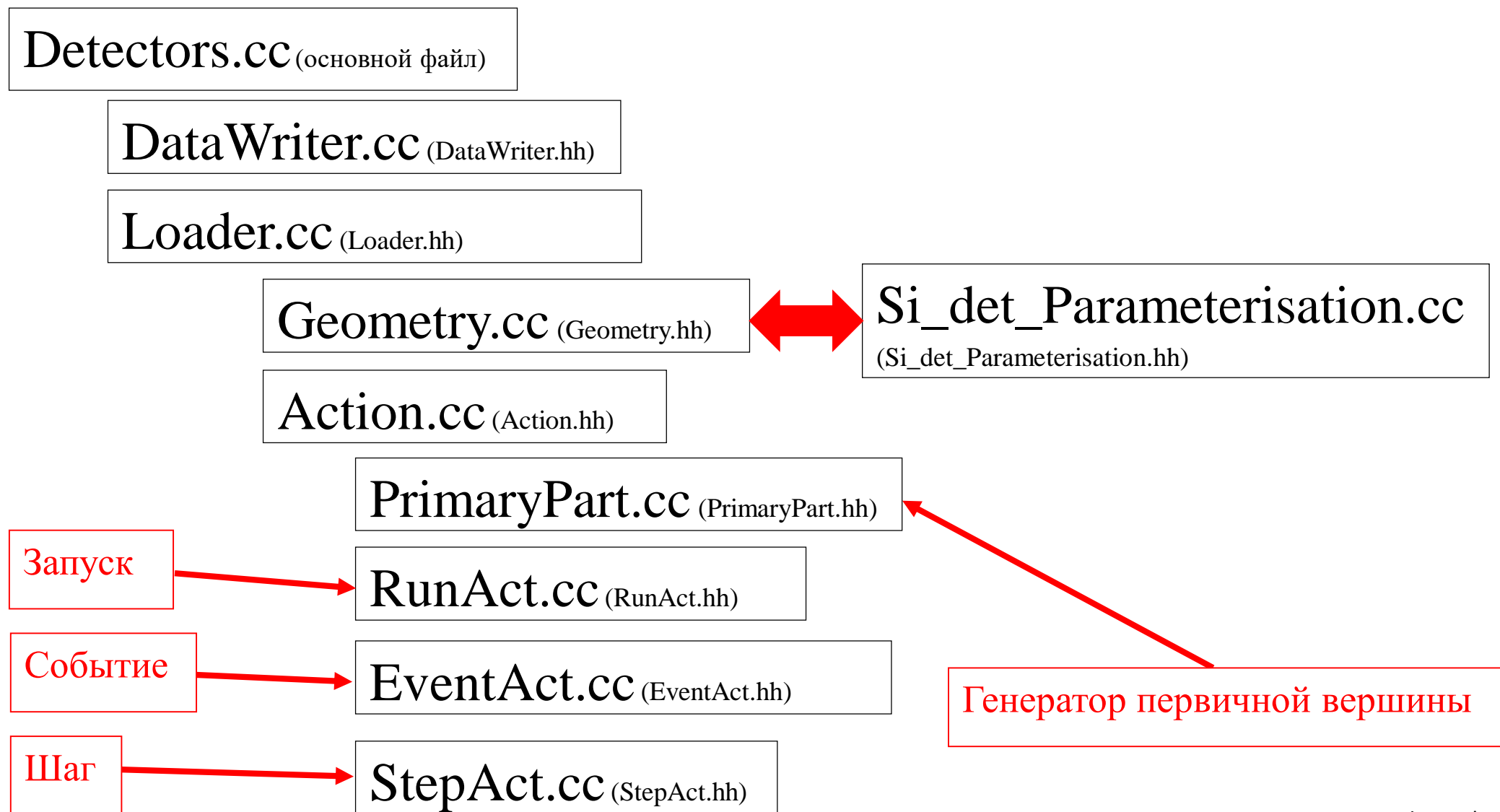
```
void Action::Build() const
{
    SetUserAction(new PrimaryPart(*this->f_act));
    SetUserAction(new RunAct(*this->f_act));
    SetUserAction(new EventAct(*this->f_act));
    SetUserAction(new TrackAct(*this->f_act));
    SetUserAction(new StackAct(*this->f_act));
    SetUserAction(new StepAct(*this->f_act));
}
```

Цикл обработки событий



Цикл обработки событий

проект: Detectors



Цикл обработки событий

```
RunAct::RunAct(std::ofstream& ofsa)
```

```
{  
    this->f_act=&ofsa;  
    (*f_act) << std::setw(12) << "Hi from RunAct!" << std::endl;  
}
```

```
RunAct::~RunAct()
```

```
{  
    (*f_act) << std::setw(12) << "Bye from RunAct!" << std::endl;  
}
```

```
time_t Start, End;
```

```
int RunNum = 0;
```

```
void RunAct::BeginOfRunAction(const G4Run* aRun)
```

```
{  
    G4cout << "\n---Start----- Run # " << RunNum << " ----- \n" << "RunId=" << aRun->GetRunID() << G4endl;  
    time(&Start);  
}
```

```
void RunAct::EndOfRunAction(const G4Run* aRun)
```

```
{  
    time(&End);  
    G4cout << " Time spent on this Run = " << difftime(End, Start) << " seconds" << G4endl;  
    G4cout << "\n---Stop----- Run # " << RunNum << " ----- \n" << "RunId=" << aRun->GetRunID() << G4endl;  
    RunNum++;  
}
```

Самой крупной единицей моделирования является запуск.

С началом запуска начинается цикл моделирования, а конец запуска совпадает с завершением цикла.

Класс **G4UserRunAction** является опциональным пользовательским классом «действий», связанным с запусками.

Цикл обработки событий

G4UserEventAction является опциональным базовым классом, для класса пользовательских «действий» на каждом событии.

Данный класс **не осуществляет** моделирование событий (*его осуществляет экземпляр класса **G4Event***), а лишь позволяет менять параметры событий, или сохранять информацию о моделировании во время событий.

Два виртуальных метода:

```
virtual void BeginOfEventAction(const G4Event* anEvent);  
virtual void EndOfEventAction(const G4Event* anEvent);
```

Вызываются в начале (после создания первичной вершины) и конце каждого события, что можно использовать для сохранения данных.

Цикл обработки событий

EventAct.hh

EventAct.cc

```
class EventAct : public G4UserEventAction
```

```
{
```

```
public:
```

```
    std::ofstream *f_event;
```

```
    EventAct(std::ofstream&);
```

```
    ~EventAct();
```

```
    static void Coordinates(G4ThreeVector V1, G4ThreeVector V2);
```

```
    static void AddE(G4double dE);
```

```
    static void StepLengthCounter(G4double SL);
```

```
    void BeginOfEventAction(const G4Event*);
```

```
    void EndOfEventAction(const G4Event*);
```

```
};
```

Статические методы

```
EventAct::EventAct(std::ofstream& ofsa)
```

```
{
```

```
    this->f_event=&ofsa;
```

```
    (*f_event) << "Hi from Event!" << std::endl;
```

```
}
```

```
EventAct::~~EventAct()
```

```
{
```

```
    (*f_event) << "Bye from Event!" << std::endl;
```

```
}
```

```
int SLcounter=0;
```

```
G4double Esum=0.;
```

```
void EventAct::Coordinates(G4ThreeVector V1, G4ThreeVector V2)
```

```
{
```

```
    G4cout << "X1=" << V1[0] * mm << " Y1=" << V1[1] * mm << " Z1=" << V1[2] * mm << G4endl;
```

```
    G4cout << "X2=" << V2[0] * mm << " Y2=" << V2[1] * mm << " Z2=" << V2[2] * mm << G4endl;
```

```
}
```

```
void EventAct::StepLengthCounter(G4double count1)
{
    SLcounter ++;
    G4cout << "Step N= " << SLcounter << "\t, Length=" << count1 * mm << G4endl;
}

void EventAct::AddE(G4double edep)
{
    Esum+=edep;
}

void EventAct::BeginOfEventAction(const G4Event * EVE)
{
    G4cout << "BeginWorks\t" << EVE->GetEventID() << G4endl;
    SLcounter = 0;
}

void EventAct::EndOfEventAction(const G4Event *EVE)
{
    (*f_event) <<"Esum=" << std::setw(12) << Esum << std::endl;
    G4cout << "EndWorks\t" << EVE->GetEventID() << G4endl;
    SLcounter=0;
    Esum=0.;
}
```

Цикл обработки событий

G4UserSteppingAction является опциональным базовым классом, для класса пользовательских «действий» в конце каждого шага. Данный класс не осуществляет моделирование шагов.

stepAction.cc

```
void StepAct::UserSteppingAction(const G4Step* step)
```

```
{
```

```
  G4StepPoint* point1 = step->GetPreStepPoint();
```

```
  G4StepPoint* point2 = step->GetPostStepPoint();
```

```
  G4ThreeVector vect1, vect2;
```

```
  G4double Ekin1, Ekin2;
```

```
  vect1=point1->GetPosition();
```

```
  vect2=point2->GetPosition();
```

```
  EventAct::Coordinates(vect1,vect2);
```

```
  Ekin1=point1->GetKineticEnergy();
```

```
  Ekin2=point2->GetKineticEnergy();
```

```
  G4double StepLength = step->GetStepLength();
```

```
  EventAct::StepLengthCounter(StepLength);
```

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

```
  EventAct::AddE(edep);
```

```
  G4cout<<step->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName()<<" "<< Ekin1 * MeV
```

```
    <<" "<< Ekin2 * MeV<<" "<< StepLength * mm <<G4endl;
```

```
}
```

stepAction.hh

```
class StepAct :public G4UserSteppingAction
```

```
{
```

```
  public:
```

```
    std::ofstream *f_step;
```

```
    StepAct(std::ofstream& ofsa)
```

```
{
```

```
  this->f_step=&ofsa;
```

```
  (*f_step) << "Hi from Step!" << std::endl;
```

```
};
```

```
~StepAct()
```

```
{
```

```
  (*f_step) << "Bye from Step!" << std::endl;
```

```
};
```

```
void UserSteppingAction(const G4Step*);
```

```
};
```

Виртуальный метод
вызывается в конце
каждого шага.

Цикл обработки событий

G4StepPoint Class Reference

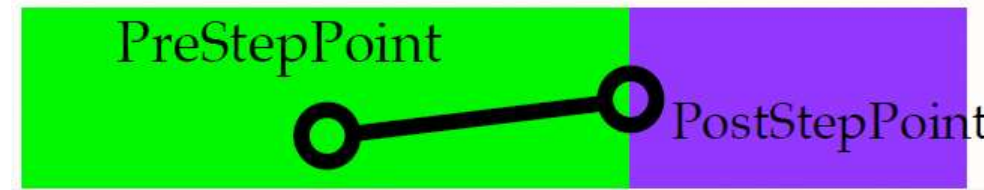
Public Member Functions

	G4StepPoint ()
	~G4StepPoint ()
	G4StepPoint (const G4StepPoint &)
G4StepPoint &	operator= (const G4StepPoint &)
const G4ThreeVector &	GetPosition () const
void	SetPosition (const G4ThreeVector &aValue)
void	AddPosition (const G4ThreeVector &aValue)
G4double	GetLocalTime () const
void	SetLocalTime (const G4double aValue)
void	AddLocalTime (const G4double aValue)
G4double	GetGlobalTime () const
void	SetGlobalTime (const G4double aValue)
void	AddGlobalTime (const G4double aValue)
G4double	GetProperTime () const
void	SetProperTime (const G4double aValue)
void	AddProperTime (const G4double aValue)
const G4ThreeVector &	GetMomentumDirection () const
void	SetMomentumDirection (const G4ThreeVector &aValue)
void	AddMomentumDirection (const G4ThreeVector &aValue)
G4ThreeVector	GetMomentum () const
G4double	GetTotalEnergy () const
G4double	GetKineticEnergy () const
void	SetKineticEnergy (const G4double aValue)
void	AddKineticEnergy (const G4double aValue)
G4double	GetVelocity () const
void	SetVelocity (G4double v)
G4double	GetBeta () const
G4double	GetGamma () const
G4VPhysicalVolume *	GetPhysicalVolume () const
const G4VTouchable *	GetTouchable () const
const G4TouchableHandle &	GetTouchableHandle () const
void	SetTouchableHandle (const G4TouchableHandle &apValue)
G4Material *	GetMaterial () const
void	SetMaterial (G4Material *)
const G4MaterialCutsCouple *	GetMaterialCutsCouple () const
void	SetMaterialCutsCouple (const G4MaterialCutsCouple *)
G4VSensitiveDetector *	GetSensitiveDetector () const

Для извлечения информации на каждом шаге используются методы класса G4Step:

G4StepPoint* GetPreStepPoint() const;
G4StepPoint* GetPostStepPoint() const;

содержащие информацию о состоянии частицы в начале и конце шага, соответственно.



void	SetSensitiveDetector (G4VSensitiveDetector *)
G4double	GetSafety () const
void	SetSafety (const G4double aValue)
const G4ThreeVector &	GetPolarization () const
void	SetPolarization (const G4ThreeVector &aValue)
void	AddPolarization (const G4ThreeVector &aValue)
G4StepStatus	GetStepStatus () const
void	SetStepStatus (const G4StepStatus aValue)
const G4VProcess *	GetProcessDefinedStep () const
void	SetProcessDefinedStep (const G4VProcess *aValue)
G4double	GetMass () const
void	SetMass (G4double value)
G4double	GetCharge () const
void	SetCharge (G4double value)
G4double	GetMagneticMoment () const
void	SetMagneticMoment (G4double value)
void	SetWeight (G4double aValue)
G4double	GetWeight () const

Цикл обработки событий

Методы класса G4Step

Public Member Functions

```
G4Step ()
~G4Step ()
G4Step (const G4Step &)
G4Step & operator= (const G4Step &)
G4Track * GetTrack () const
void SetTrack (G4Track *value)
G4StepPoint * GetPreStepPoint () const
void SetPreStepPoint (G4StepPoint *value)
G4StepPoint * GetPostStepPoint () const
void SetPostStepPoint (G4StepPoint *value)
G4double GetStepLength () const
void SetStepLength (G4double value)
G4double GetTotalEnergyDeposit () const
void SetTotalEnergyDeposit (G4double value)
G4double GetNonIonizingEnergyDeposit () const
void SetNonIonizingEnergyDeposit (G4double value)
G4SteppingControl GetControlFlag () const
void SetControlFlag (G4SteppingControl StepControlFlag)
void AddTotalEnergyDeposit (G4double value)
void ResetTotalEnergyDeposit ()
void AddNonIonizingEnergyDeposit (G4double value)
void ResetNonIonizingEnergyDeposit ()
G4bool IsFirstStepInVolume () const
G4bool IsLastStepInVolume () const
void SetFirstStepFlag ()
void ClearFirstStepFlag ()
void SetLastStepFlag ()
void ClearLastStepFlag ()
G4ThreeVector GetDeltaPosition () const
G4double GetDeltaTime () const
```

Методы, позволяющие определить изменения во времени, позиции и энергии:

```
G4double GetDeltaTime() const;
G4ThreeVector GetDeltaPosition() const;
G4double GetTotalEnergyDeposit() const;
```

доступ к треку частицы:

```
G4Track* GetTrack() const;
```

```
G4ThreeVector GetDeltaMomentum () const
G4double GetDeltaEnergy () const
void InitializeStep (G4Track *aValue)
void UpdateTrack ()
void CopyPostToPreStepPoint ()
G4Polyline * CreatePolyline () const
const std::vector< const G4Track * > * GetSecondaryInCurrentStep () const
const G4TrackVector * GetSecondary () const
G4TrackVector * GetfSecondary ()
G4TrackVector * NewSecondaryVector ()
void DeleteSecondaryVector ()
void SetSecondary (G4TrackVector *value)
void SetPointerToVectorOfAuxiliaryPoints (std::vector< G4ThreeVector > *theNewVectorPointer)
std::vector< G4ThreeVector > * GetPointerToVectorOfAuxiliaryPoints () const
```

доступ к трекам вторичных частиц:

```
const G4TrackVector* GetSecondary() const ;
```

Цикл обработки событий

Сбор информации с шагов в запуске

Универсальной цепочки связи между наследуемыми классами действий в базовых классах нет.

Все базовые классы действий содержат лишь конструкторы по умолчанию.

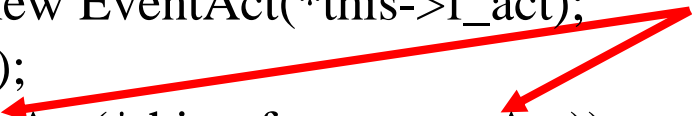
Простейшая передача информации между классами (например, от G4UserSteppingAction к G4UserEventAction) может быть реализована с использованием статических методов.

За счет особенностей наследования пользователь может сам определить, какой класс и как будет связан с другими.

проект Interact (Action.cc)

```
void Action::Build() const
{
    SetUserAction(new RunAct(*this->f_act));
    SetUserAction(new PrimaryPart(*this->f_act));
    EventAct* eventAct = new EventAct(*this->f_act);
    SetUserAction(eventAct);
    SetUserAction(new StepAct(*this->f_act,eventAct));
}
```

Изменив конструкторы классов действий таким образом, что один из них будет принимать указатель на другой, можно получить необходимую связь между ними.



Цикл обработки событий

проект Interact (StepAction.hh)

```
class EventAct;
```

```
class StepAct :public G4UserSteppingAction
```

```
{
```

```
public:
```

```
    std::ofstream *f_step;
```

```
StepAct(std::ofstream& ofsa, EventAct* eventAct):event(eventAct)
```

```
{
```

```
    this->f_step=&ofsa;
```

```
    (*f_step) << "Hi from Step!" << std::endl;
```

```
};
```

```
~StepAct()
```

```
{
```

```
    (*f_step) << "Bye from Step!" << std::endl;
```

```
};
```

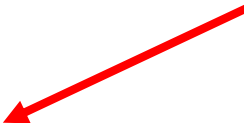
```
void UserSteppingAction(const G4Step*);
```

```
private:
```

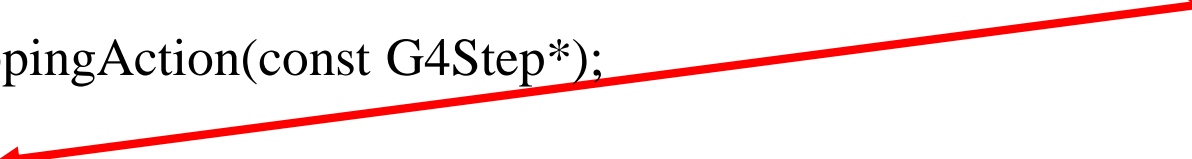

```
EventAct* event;
```

```
};
```

инициализация члена
event класса StepAct



обратная связь класса действий для
шагов с экземпляром созданного
класса действий для событий



Цикл обработки событий

проект Interact (EventAct.cc)

проект Interact (StepAct.cc)

```
class EventAct : public G4UserEventAction
{
public:
    std::ofstream *f_event;
    EventAct(std::ofstream&);
    ~EventAct();
    void Coordinates(G4ThreeVector V1, G4ThreeVector V2);
    void AddE(G4double dE);
    void StepLengthCounter(G4double SL);
    void BeginOfEventAction(const G4Event*);
    void EndOfEventAction(const G4Event*);
};
```

Не статические методы

**Вызов методов созданного
объекта event класса EventAct**

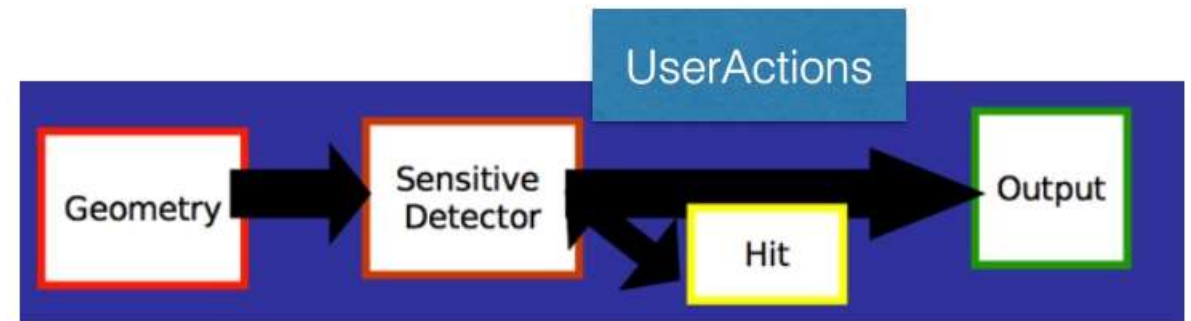
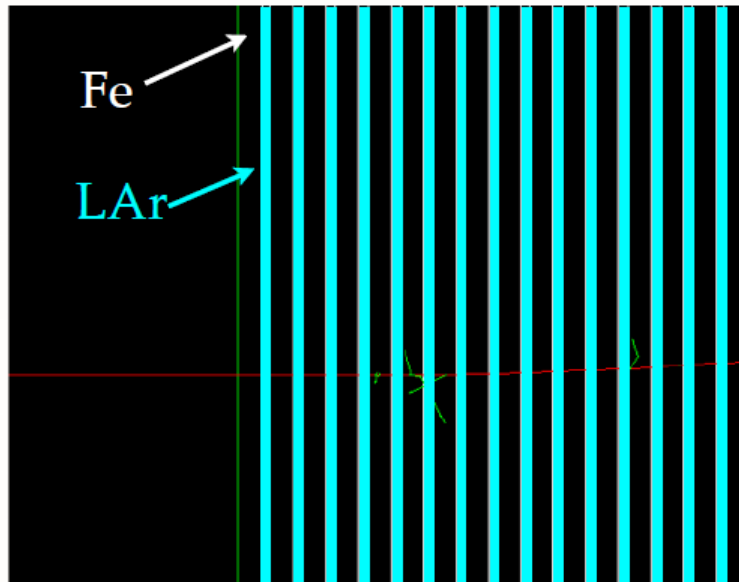
```
void StepAct::UserSteppingAction(const G4Step*
step)
{
    G4StepPoint* point1 = step->GetPreStepPoint();
    G4StepPoint* point2 = step->GetPostStepPoint();
    G4ThreeVector vect1, vect2;
    G4double Ekin1, Ekin2;
    vect1=point1->GetPosition();
    vect2=point2->GetPosition();
    event->Coordinates(vect1,vect2);
    Ekin1=point1->GetKineticEnergy();
    Ekin2=point2->GetKineticEnergy();
    G4double StepLength = step->GetStepLength();
    event->StepLengthCounter(StepLength);
    G4double edep = step->GetTotalEnergyDeposit();
    event->AddE(edep);
```

```
G4cout<<step->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName()<<" "<<Ekin1 * MeV<<" "<<Ekin2 *
MeV<<" "<< StepLength * mm <<G4endl;
}
```

Создание детектирующих объёмов

- ❑ Любой логический объем в модели можно объявить детектирующим, или «чувствительным».
- ❑ При прохождении частицы через данный объем моделируется срабатывание детектора.
- ❑ Детектирующих объемов одновременно может несколько.
- ❑ Обработка срабатываний при этом может происходить по разному.
- ❑ Дополнительно можно смоделировать оцифровку сигнала и электронный отклик детектора (**ПОЗЖЕ**).

мюон, 2 ГэВ



Создание детектирующих объёмов

G4VSensitiveDetector Class Reference

Public Member Functions

	G4VSensitiveDetector (G4String name)
	G4VSensitiveDetector (const G4VSensitiveDetector &right)
virtual	~G4VSensitiveDetector ()
const G4VSensitiveDetector &	operator= (const G4VSensitiveDetector &right)
G4int	operator== (const G4VSensitiveDetector &right) const
G4int	operator!= (const G4VSensitiveDetector &right) const
virtual void	Initialize (G4HCofThisEvent *)
virtual void	EndOfEvent (G4HCofThisEvent *)
virtual void	clear ()
virtual void	DrawAll ()
virtual void	PrintAll ()
G4bool	Hit (G4Step *aStep)
void	SetROGeometry (G4VReadOutGeometry *value)
void	SetFilter (G4VSDFilter *value)
G4int	GetNumberOfCollections () const
G4String	GetCollectionName (G4int id) const
void	SetVerboseLevel (G4int vl)
void	Activate (G4bool activeFlag)
G4bool	isActive () const
G4String	GetName () const
G4String	GetPathName () const
G4String	GetFullPathName () const
G4VReadOutGeometry *	GetROGeometry () const
G4VSDFilter *	GetFilter () const

Protected Member Functions

virtual G4bool	ProcessHits (G4Step *aStep, G4TouchableHistory *ROhist)=0
virtual G4int	GetCollectionID (G4int i)

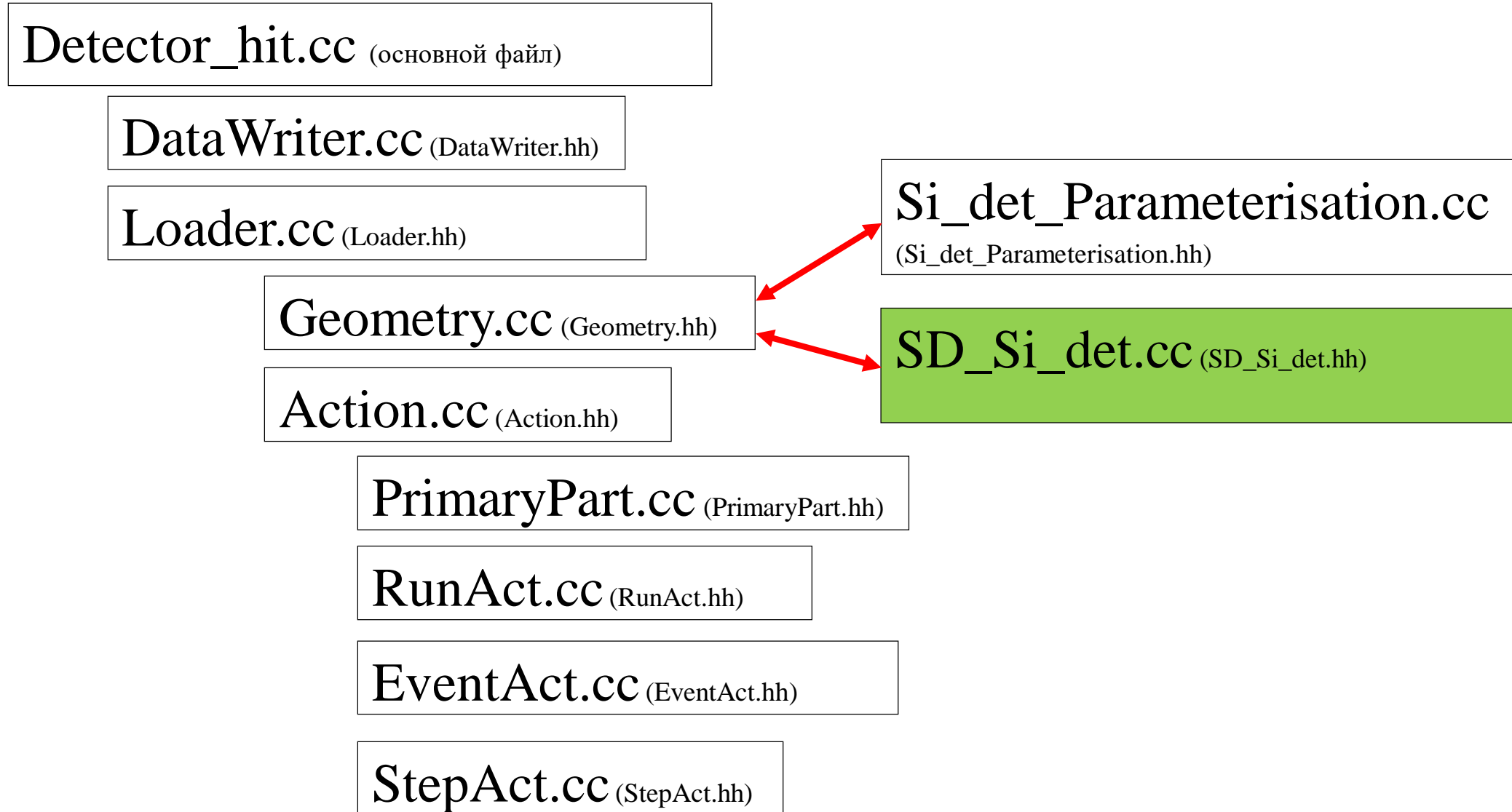
Создается класс-наследник класса G4VSensitiveDetector

Описываются обязательные методы:

- **Initialize()** вызывается в начале каждого события
- **ProcessHits()** вызывается на каждом шаге в детектирующем объеме.
Позволяет получить информацию о характеристиках частицы в данной точке, о взаимодействии с веществом, и смоделировать срабатывание детектора.
- **EndOfEvent()** вызывается в конце события.
Позволяет провести отбор срабатываний, и сохранить результаты.

Создание детектирующих объёмов

проект **Detector_hit**



Создание детектирующих объёмов

```
G4SDManager* sdman = G4SDManager::GetSDMpointer();
SD_Si_det* sensitive_Si_det = new SD_Si_det("/mySi_det");
sdman->AddNewDetector(sensitive_Si_det);
Si_det_log->SetSensitiveDetector(sensitive_Si_det);
```

Инициализируется объект класса G4SDManager

Создание экземпляра класса для детектирующего объёма, нужно задать имя

Объект, описывающий детектирующий объем, регистрируется в менеджере

Детектирующий объем ассоциируется с логическим объемом

Sd_Si_det.hh

```
class SD_Si_det : public G4VSensitiveDetector
```

```
{
```

```
public:
```

```
SD_Si_det(G4String SDname);
```

```
~SD_Si_det();
```

```
G4bool ProcessHits(G4Step* astep, G4TouchableHistory*);
```

```
void EndOfEvent(G4HCofThisEvent* HCE);
```

```
G4double GetSumE(G4int i) const {return SumE[i];}
```

```
void AddSumE(double e, G4int i) {SumE[i]+=e;}
```

```
private:
```

```
std::ofstream hit_SD_Si_det[10];
```

```
G4double SumE[10];
```

```
};
```

информация о размещении физических объёмов

коллекция откликов или срабатываний для данного события
(**ПОКА** не используем)

Создание детектирующих объёмов

```
G4bool SD_Si_det :: ProcessHits(G4Step* step, G4TouchableHistory*)
{
  if (step->GetPostStepPoint()->GetMaterial()==G4NistManager::Instance()->FindOrBuildMaterial("G4_Si"))
  {
    G4TouchableHandle touchable = step->GetPreStepPoint()->GetTouchableHandle();
    G4int copyNo  = touchable->GetVolume(0)->GetCopyNo();
    G4double edep = 0.;
    edep          = step->GetTotalEnergyDeposit();
    this->AddSumE(edep,copyNo);
  }
  return true;
}
```

```
void SD_Si_det :: EndOfEvent(G4HCofThisEvent*)
{
  for (G4int i=0; i<10; i++)
  { hit_SD_Si_det[i] << std::setw(10) << SumE[i] << G4endl;}
  for (G4int i=0; i<10; i++) { SumE[i]=0.;}
}
```

Создание детектирующих объёмов

