

# Цикл обработки событий

Пользователь напрямую не управляет ни одной из условных единиц моделирования (RUN, EVENT, TRACK, STEP), 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

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

## Protected Member Functions

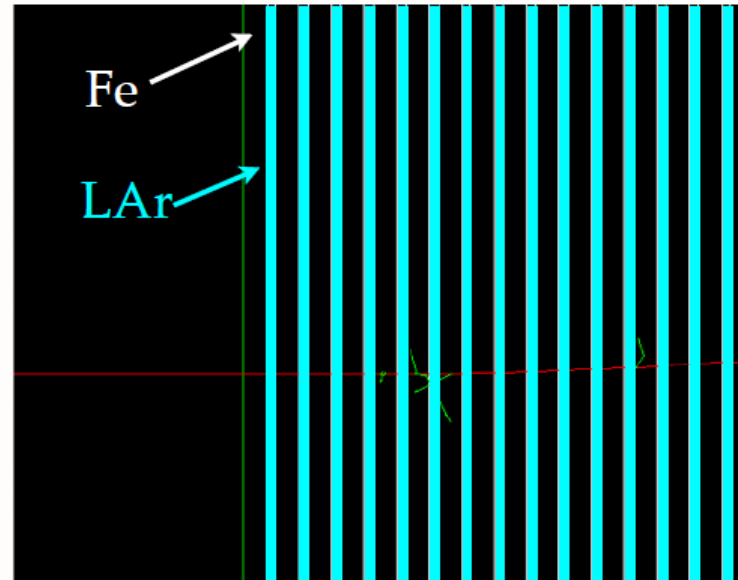
void <b>SetUserAction (G4VUserPrimaryGeneratorAction *)</b> const
void <b>SetUserAction (G4UserRunAction *)</b> const
void <b>SetUserAction (G4UserEventAction *)</b> const
void <b>SetUserAction (G4UserStackingAction *)</b> const
void <b>SetUserAction (G4UserTrackingAction *)</b> const
void <b>SetUserAction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

является чисто техническим классом для определения  
порядка обработки треков

# Детектирующие объёмы

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

мюон, 2 ГэВ



# Детектирующие объёмы

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.

### Protected Member Functions

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

## 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 Attributes

G4CollectionNameVector	collectionName
G4String	SensitiveDetectorName
G4String	thePathName
G4String	fullPathName
G4int	verboseLevel
G4bool	active
G4VReadOutGeometry *	ROGeometry
G4VSDFilter *	filter

# Срабатывания или отклики (hits) детекторов в Geant4

Хит в Geant4 – «отпечаток» физического взаимодействия частицы на данном шаге трека в детектирующем (чувствительном) объёме детектора.

«отпечаток» является информацией, которую можно сохранить на данном временном шаге:

- геометрическое положение и время данного шага;
- импульс или энергия частицы на данном шаге;
- потери энергии частицы на данном шаге;
- информация о геометрии детектора, в котором находится частица.

В Geant4 предусмотрен механизм сбора и сохранения информации о срабатываниях в детекторе.

Хиты можно сохранять только для взаимодействий в чувствительном объёме.

Хиты хранятся в контейнере (коллекции).

Хиты доступны во всех компонентах проекта.

Коллекция Хитов идентифицируется через персональный идентификатор (номер)

# Срабатывания или отклики (hits) детекторов в Geant4

Для трековых детекторов обычно сохраняются хиты для каждого шага трека заряженной частицы:

- пространственное положение
- время
- энергия
- потери энергии
- номер трека

Для калориметра обычно сохраняются хиты для каждой ячейки/элемента :

- суммарное энергосодержание в ячейке в данном событии
- номер ячейки

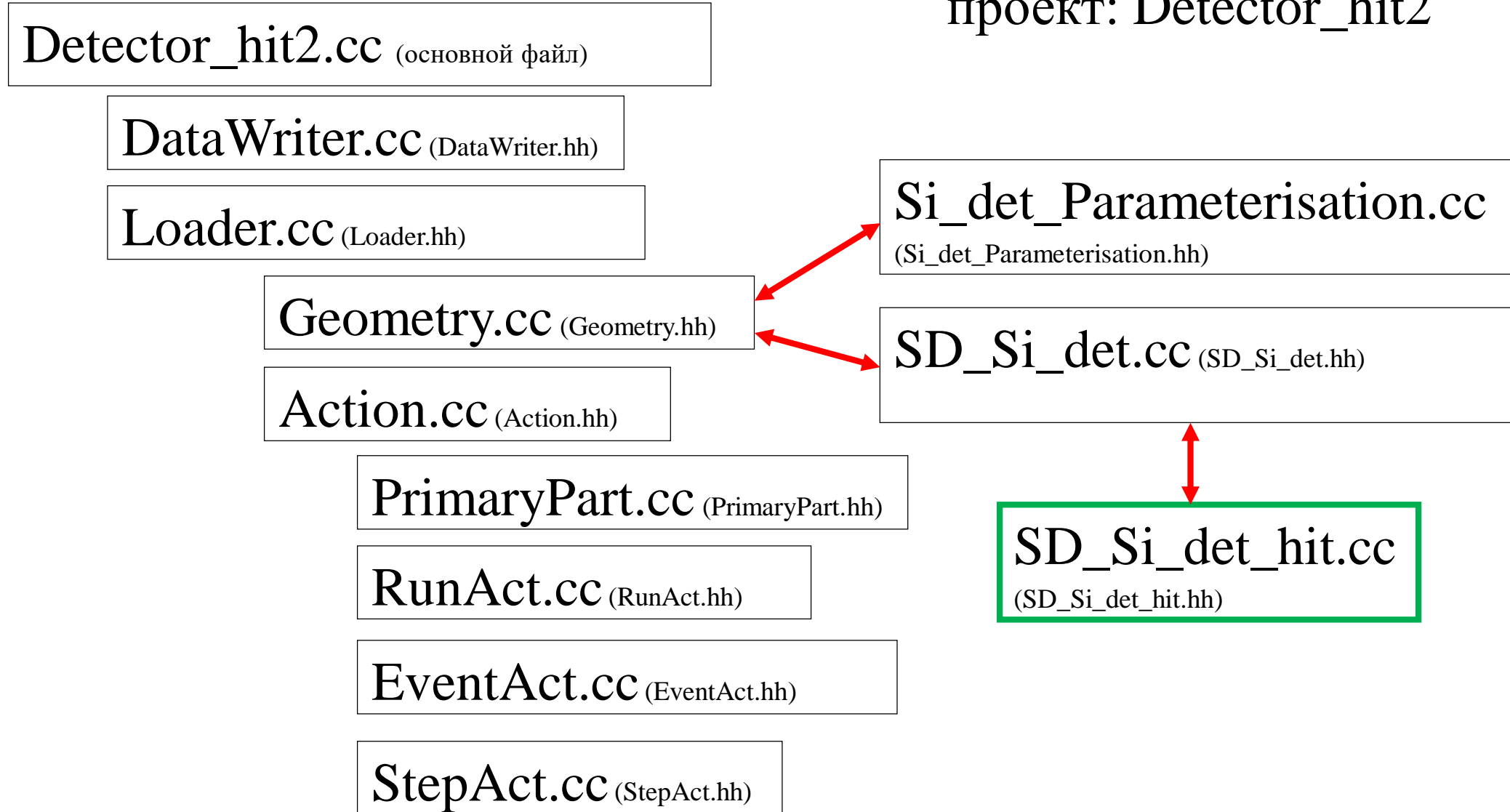
Информация об единичном отклике на текущем шаге описывается в классе –наследнике абстрактного класса **G4VHit**, объекты которого создаются в методе `ProcessHits()` класса чувствительного объёма.

Контейнер (коллекция) для хранения нескольких откликов задаётся с использованием шаблона класса-контейнера **G4VHitsCollection**.

Разные коллекции срабатываний для данного события хранятся в классе-контейнере **G4HCofThisEvent**.

# Срабатывания или отклики (hits) детекторов в Geant4

проект: Detector\_hit2



# Срабатывания или отклики(hits) детекторов в Geant4

Sd\_Si\_det\_hit.hh

class SD\_Si\_det\_hit : public G4VHit ← Наследование класса G4VHit

{

public:

SD\_Si\_det\_hit(); ~SD\_Si\_det\_hit();

SD\_Si\_det\_hit(const SD\_Si\_det\_hit&);

Конструктор копий

Перегрузка оператора =

const SD\_Si\_det\_hit& operator=(const SD\_Si\_det\_hit&);

G4int operator==(const SD\_Si\_det\_hit&) const; ← Перегрузка оператора ==

inline void\* operator new(size\_t); ← Перегрузка оператора new

inline void operator delete(void \*aHit); ← Перегрузка оператора delete

void Draw();

void Print();

← Виртуальные методы класса G4VHit

void SetEdep(const double e) {this->eDep\_hit=e;} ← Значение энергосвечения для данного срабатывания

G4double GetEdep() const {return eDep\_hit;}

void SetLayerNumber(const int c) {this->layerNumber\_hit=c;} ← Номер копии детектора для данного срабатывания

G4int GetLayerNumber() const {return layerNumber\_hit;}

private:

G4int layerNumber\_hit;

G4double eDep\_hit;

← Переменные для данного срабатывания


};

# Срабатывания или отклики (hits) детекторов в Geant4

## Sd\_Si\_det\_hit.hh

```
typedef G4THitsCollection<SD_Si_det_hit> SD_Si_det_hitCollection;
```


Создание коллекции хитов SD\_Si\_det\_hitCollection для записи срабатываний, заданных в классе SD\_Si\_det\_hit



```
extern G4Allocator<SD_Si_det_hit> hitAllocatorSD;
```


```
inline void* SD_Si_det_hit::operator new(size_t) {  
    void* aHit;  
    aHit = (void*) hitAllocatorSD.MallocSingle();  
    return aHit;  
}
```

Размещение объекта срабатываний в памяти



```
inline void SD_Si_det_hit::operator delete(void* aHit) {  
    hitAllocatorSD.FreeSingle((SD_Si_det_hit*) aHit);  
}
```

Удаление объекта срабатываний из памяти





```
#include "SD_Si_det.hh"
#include "SD_Si_det_hit.hh"
```

```
class SD_Si_det : public G4VSensitiveDetector
{
public:
    SD_Si_det(G4String SDname);
    ~SD_Si_det();
```

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



Инициализация коллекции хитов

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



Заполнение хитов и запись в коллекцию

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



Запись информации в конце события

```
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];
```

```
SD_Si_det_hitCollection *hitCollection;
```



Коллекция хитов — член класса  
чувствительных объёмов

```
G4int collectionID;
```


```
};
```

# Срабатывания или отклики (hits) детекторов в Geant4

## Sd\_Si\_det.cc

```
SD_Si_det :: SD_Si_det (G4String SDname) :  
G4VSensitiveDetector(SDname)  
{  
    collectionName.insert("SD_Si_det_hitCollection");  
    G4String filename[10];  
    char str[2];  
    G4int i;  
    for (i=0;i<10;i++)  
    {  
        filename[i] = "hit_Si_det_";  
        sprintf(str,"%d",i+1);  
        filename[i] += str;  
        filename[i] += ".dat";  
        hit_SD_Si_det[i].open(filename[i],std::fstream::out);  
    }  
    for (i=0; i<10; i++) {SumE[i]=0.;}  
}
```

Имя коллекции хитов заносится в вектор имён collectionName в конструкторе класса чувствительных объёмов Sd\_Si\_det.  
collectionName – protected член класса G4VSensitiveDetector



# Срабатывания или отклики (hits) детекторов в Geant4

## Sd\_Si\_det.cc

### Инициализация коллекции хитов

```
void SD_Si_det :: Initialize(G4HCofThisEvent* HCE)
{
    hitCollection = new SD_Si_det_hitCollection(GetName(), collectionName[0]);
    static G4int HCID=-1;
    if (HCID<0) HCID = GetCollectionID(0);
    HCE->AddHitsCollection(HCID, hitCollection);
}
```

Создание коллекции хитов

Первый элемент [0] в векторе имён collectionName

Возвращает имя коллекции хитов: SD\_Si\_det\_hitCollection

Получение ID для коллекции.

Операция выполняется не для каждого события, только один раз.

Регистрация объекта hitCollection коллекции хитов в общей коллекции хитов HCE для данного события

# Срабатывания или отклики (hits) детекторов в Geant4

## Sd\_Si\_det.cc

```
G4bool SD_Si_det :: ProcessHits(G4Step* step, G4TouchableHistory*)
{
    .....
    SD_Si_det_hit *aHit = new SD_Si_det_hit();
    aHit->SetEdep(step->GetTotalEnergyDeposit());
    aHit->SetLayerNumber(step->GetPreStepPoint()->GetTouchableHandle()->GetVolume(0)->GetCopyNo());
    hitCollection->insert(aHit);
    .....
}
```

Создание объекта с хитом

Добавление хита в коллекцию

Сохранение информации в переменных хита

# Срабатывания или отклики (hits) детекторов в Geant4

**EventAct.cc** (Запись результатов)

```
void EventAct::EndOfEventAction(const G4Event *EVE)
```

```
{
```

```
.....
```

```
SD_Si_det_hitCollection *THC = NULL;
```

```
G4HCofThisEvent *HCE = EVE->GetHCofThisEvent();
```

```
if (HCE)
```

```
{
```

```
  THC = (SD_Si_det_hitCollection*)(HCE->GetHC(0));
```

```
}
```

```
if (THC)
```

```
{
```

```
  int n_hit=THC->entries();
```

```
  for (int i=0; i<n_hit; i++)
```

```
  {
```

```
    SD_Si_det_hit *hit = (*THC)[i];
```

```
    G4cout<<"collection "<<hit->GetEdep()<<" "<<hit->GetLayerNumber()<<G4endl;
```

```
  }
```

```
}
```

```
}
```

Доступ к коллекции хитов для данного события

Доступ к коллекции хитов с ID=0

Извлечение хита из коллекции

Вывод на экран

# Срабатывания или отклики (hits) детекторов в Geant4



# Срабатывания (hits) детекторов и оцифровка сигналов

Физические величины, полученные из отклика детектора, могут быть преобразованы в электронный сигнал, аналогичный непосредственно регистрируемому системами сбора данных.

Можно моделировать

- работу АЦП и время-цифровых преобразователей
- схему сбора данных
- триггер (сигнал для записи данных детектором)
- наложение событий (pile-up)
- данные в формате, аналогичном формату электроники считывания

Процедура оцифровки реализуется для информации, записанной в хитах, в специальном модуле оцифровки.

Модуль оцифровки, в отличие от хитов, не связан с объёмом.

Каждому модулю оцифровки соответствует объект-наследник абстрактного класса `G4VDigitizerModule`, определяемого пользователем.

Процедура оцифровки должна быть задана в методе `Digitize()`, объекта-наследника абстрактного класса `G4VDigitizerModule`, определяемого пользователем.

## G4VDigitizerModule Class Reference

### Public Member Functions

	<code>G4VDigitizerModule (G4String modName)</code>
virtual	<code>~G4VDigitizerModule ()</code>
int	<code>operator== (const G4VDigitizerModule &amp;right) const</code>
int	<code>operator!= (const G4VDigitizerModule &amp;right) const</code>
virtual void	<code>Digitize ()=0</code>
G4int	<code>GetNumberOfCollections () const</code>
G4String	<code>GetCollectionName (G4int i) const</code>
G4String	<code>GetName () const</code>
void	<code>SetVerboseLevel (G4int val)</code>

Чисто виртуальный  
метод

### Protected Member Functions

void	<code>StoreDigiCollection (G4VDigiCollection *aDC)</code>
void	<code>StoreDigiCollection (G4int DCID, G4VDigiCollection *aDC)</code>

### Protected Attributes

G4DigiManager *	<code>DigiManager</code>
G4String	<code>moduleName</code>
std::vector< G4String >	<code>collectionName</code>
G4int	<code>verboseLevel</code>

# Срабатывания (hits) детекторов и оцифровка сигналов

Оцифрованный сигнал хранится в объекте-наследнике класса G4VDigi, определяемого пользователем.

Отдельные оцифрованные сигналы могут объединяться в коллекции (G4VDigiCollection).

Управление оцифровкой осуществляется объектом класса G4DigiManager.

## G4DigiManager Class Reference

### Public Member Functions

	~G4DigiManager ()
void	AddNewModule (G4VDigitizerModule *DM)
void	Digitize (G4String mName)
G4VDigitizerModule *	FindDigitizerModule (G4String mName)
const G4VHitsCollection *	GetHitsCollection (G4int HCID, G4int eventID=0)
const G4VDigiCollection *	GetDigiCollection (G4int DCID, G4int eventID=0)
G4int	GetHitsCollectionID (G4String HCname)
G4int	GetDigiCollectionID (G4String DCname)
void	SetDigiCollection (G4int DCID, G4VDigiCollection *aDC)
void	SetVerboseLevel (G4int vl)
void	List () const
G4int	GetVerboseLevel () const
G4int	GetCollectionCapacity () const
G4int	GetModuleCapacity () const
G4DCtable *	GetDCtable () const
void	RestoreDCtable (G4DCtable *dc)

### Static Public Member Functions

static G4DigiManager *	GetDMpointer ()
static G4DigiManager *	GetDMpointerIfExist ()

### Protected Member Functions

G4DigiManager ()
------------------

Доступ к коллекции хитов

Доступ к коллекции оцифрованных сигналов

Доступ к менеджеру управления оцифровкой



# Срабатывания (hits) детекторов и оцифровка сигналов

Digitization.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)

StepAct.cc (StepAct.hh)

Проект: Digitization

Si\_det\_Parameterisation.cc  
(Si\_det\_Parameterisation.hh)

SD\_Si\_det.cc (SD\_Si\_det.hh)

SD\_Si\_det\_hit.cc  
(SD\_Si\_det.hh)

SiDigitizer.cc  
(SiDigitizer.hh)

SiDigi.cc  
(SiDigi.hh)

# Срабатывания (hits) детекторов и оцифровка сигналов

**EventAct.cc** (Инициализация модуля оцифровки)

```
EventAct::EventAct(std::ofstream& ofsa)
{
    this->f_event=&ofsa;
    (*f_event) << "Hi from Event!" << std::endl;
    SiDigitizer* digitizer =new SiDigitizer("digitizer");
    G4DigiManager* digiManager=G4DigiManager::GetDMpointer();
    digiManager->AddNewModule(digitizer);
}
```

Создание объекта класса SiDigitizer (SiDigitizer.cc) с именем digitizer – наследника класса G4VDigitizerModule

Доступ к менеджеру управления объектами оцифровки

Добавления созданного объекта класса SiDigitizer к менеджеру

Оцифровка реализуется в методе Digitize() объекта класса SiDigitizer

# Срабатывания (hits) детекторов и оцифровка сигналов

## SiDigitizer.hh

```
class SiDigitizer : public G4VDigitizerModule
{
public:
    SiDigitizer(G4String SDname);
    ~SiDigitizer();
    void Digitize(); ← Процедура оцифровки
private:
    SiDigi_Collection* digCollection; ← Коллекция оцифрованных сигналов (SiDigi.hh)
};
```

## SiDigitizer.cc

Задание имени модуля оцифровки

```
SiDigitizer :: SiDigitizer (G4String SDname) : G4VDigitizerModule(SDname)
{
    collectionName.push_back("digCollection");
}
```

Имя коллекции оцифрованных сигналов заносится в вектор имён collectionName в конструкторе класса модуля оцифровки SiDigitizer; collectionName — protected член класса G4VDigitizerModule

# Срабатывания (hits) детекторов и оцифровка сигналов

**SiDigitizer.cc**

*Создание коллекции оцифрованных сигналов (SiDigi.hh)*

```
void SiDigitizer::Digitize()
{
    digCollection = new SiDigi_Collection("SiDigitizer", "digCollection");
    G4DigiManager* digitManager = G4DigiManager::GetDMpointer();
    G4int collectionID=digitManager->GetHitsCollectionID("SD_Si_det_hitCollection");
    SD_Si_det_hitCollection* hitCollection{nullptr};
    hitCollection=(SD_Si_det_hitCollection*)(digitManager->GetHitsCollection(collectionID));
    G4int i;
    if (hitCollection!=nullptr)
    {
        G4int numberOfHits=hitCollection->entries();
        for (i=0; i<numberOfHits; i++)
        {
            G4double depositedEnergy=(*hitCollection)[i]->GetEdep();
            depositedEnergy=(int)(depositedEnergy*10.);
            G4int ln=(*hitCollection)[i]->GetLayerNumber();
            SiDigi* digit=new SiDigi();
            digit->SetEdep(depositedEnergy);
            digit->SetLayerNumber(ln);
            digCollection->insert(digit);
        }
    }
    StoreDigiCollection(digCollection);
}
```

*Имя источника оцифрованных сигналов*

*Имя коллекции оцифрованных сигналов*

*Доступ к менеджеру управления объектами оцифровки*

*Доступ к коллекции хитов*

*Цикл по всем хитам в коллекции*

*Оцифровка сигнала, записанного в коллекции хитов*

*Создание объекта-наследника SiDigi класса G4VDigi*

*Запись оцифрованного сигнала в объект-наследник SiDigi класса G4VDigi*

*Добавление оцифрованного сигнала в коллекцию*

*Сохранение коллекции оцифрованных сигналов*

# Срабатывания (hits) детекторов и оцифровка сигналов

**SiDigi.hh** (объект класса оцифрованных сигналов)

```
class SiDigi : public G4VDigi
{
public:
    explicit SiDigi();
    ~SiDigi();
    SiDigi(const SiDigi&);
    const SiDigi& operator=(const SiDigi&);
    G4int operator==(const SiDigi&) const;
    inline void* operator new(size_t);
    inline void operator delete(void *aDigi);
    void Draw();
    void Print();
    void SetEdep(const double e) {this->eDep_digi=e;}
    G4double GetEdep() const {return eDep_digi;}
    void SetLayerNumber(const int c) {this->layerNumber_digi=c;}
    G4int GetLayerNumber() const {return layerNumber_digi;}

private:
    G4int layerNumber_digi;
    G4double eDep_digi;
};
```

*Шаблон для коллекции оцифрованных сигналов* → `typedef G4TDigiCollection<SiDigi> SiDigi_Collection;`  
`extern G4Allocator<SiDigi> DigiAllocator;`

*Размещение в памяти объекта класса оцифрованных сигналов* → `inline void* SiDigi::operator new(size_t)`  
{  
    void\* aDigi;  
    aDigi = (void\*) DigiAllocator.MallocSingle();  
    return aDigi;  
}

*Удаление из памяти объекта класса оцифрованных сигналов* → `inline void SiDigi::operator delete(void* aDigi)`  
{  
    DigiAllocator.FreeSingle((SiDigi\*) aDigi);  
}

*Интерфейс для членов класса оцифрованных сигналов*

*Члены класса оцифрованных сигналов* ←

# Срабатывания (hits) детекторов и оцифровка сигналов

**EventAct.cc** (Оцифровка и запись результатов)

```
void EventAct::EndOfEventAction(const G4Event *EVE)
{
    .....
    G4DigiManager* digiManager=G4DigiManager::GetDMpointer();
    SiDigitizer* digiModule = static_cast<SiDigitizer*>(digiManager->FindDigitizerModule("digitizer"));
    digiModule->Digitize();

    G4int hitsCollID=digiManager->GetHitsCollectionID("SD_Si_det_hitCollection");
    G4int digiCollID=digiManager->GetDigiCollectionID("digCollection");
    const SiDigi_Collection* DHC = static_cast<const SiDigi_Collection*>(digiManager->GetDigiCollection(digiCollID));
    if (DHC)
    { G4cout<<"name DHC="<<DHC->GetName()<<" "<<DHC->GetDMname()<<" "<<DHC->entries()<<G4endl; }

    const SD_Si_det_hitCollection* THC2 = static_cast<const SD_Si_det_hitCollection*>(digiManager->GetHitsCollection(hitsCollID));
    if (THC2)
    { G4cout<<"name THC2="<<THC2->GetName()<<" "<<THC2->GetSDname()<<" "<<THC2->entries()<<G4endl; }
    .....
}
```

*Доступ к менеджеру управления объектами оцифровки*

*Доступ к модулю оцифровки*

*Доступ к коллекции хитов оцифрованных сигналов*

*Доступ к коллекции оцифрованных сигналов*

**Запуск процедуры оцифровки**

*Вывод информации о коллекции оцифрованных сигналов*

*Вывод информации о коллекции хитов*

# Срабатывания (hits) детекторов и оцифровка сигналов

**EventAct.cc** (Оцифровка и запись результатов)

```
void EventAct::EndOfEventAction(const G4Event *EVE)
```

```
{
```

```
.....
```

```
SD_Si_det_hitCollection* THC = NULL;
```

```
G4HCofThisEvent* HCE = EVE->GetHCofThisEvent();
```

```
if (HCE)
```

```
{ THC = (SD_Si_det_hitCollection*)(HCE->GetHC(0)); }
```

```
if (THC)
```

```
{
```

```
G4int n_hit=THC->entries();
```

```
G4cout<<"EventID="<<EVE->GetEventID()<<G4endl;
```

```
for (G4int i=0; i<n_hit; i++)
```

```
{
```

```
SD_Si_det_hit* hit = (*THC)[i];
```

```
G4cout<<"collection ID="<<hitsCollID<<" "<<THC->GetName()<<" "<<THC->GetSDname()<<" "<<hit->GetEdep()<<  
" "<<hit->GetLayerNumber()<<G4endl;
```

```
SiDigi* dig = (*DHC)[i];
```

```
G4cout<<dig->GetEdep()<<" "<<dig->GetLayerNumber()<<G4endl;
```

```
}
```

```
}
```

```
}
```

*Доступ к коллекции хитов для данного события*

*Доступ к коллекции хитов с ID=0*

*Извлечение хита из коллекции*

*Вывод информации о коллекции хитов*

*Извлечение объекта оцифровки из коллекции*

*Вывод оцифрованных сигналов*

# Универсальные детекторы, счётчики, фильтры

Вместо кода пользователя для описания класса чувствительных объёмов можно использовать стандартный объект класса универсальных детекторов (G4MultifunctionalDetector).

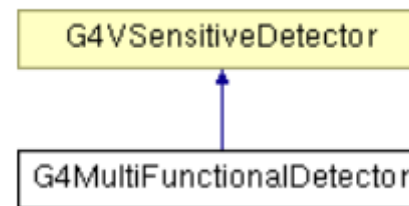
Объект класса универсальных детекторов создаётся в модуле описания геометрии и добавляется к одному или нескольким логическим объёмам.

Отклик детектора записывается с использованием стандартных объектов-счётчиков (G4PrimitiveScorer).

Каждый счетчик сохраняет значения определенной физической величины для каждого срабатывания (шага) в картах хитов (G4THitsMap).

Для каждого события создаётся одна карта, содержащая пары значений: физическая величина и индекс, соответствующий номеру копии объёма.

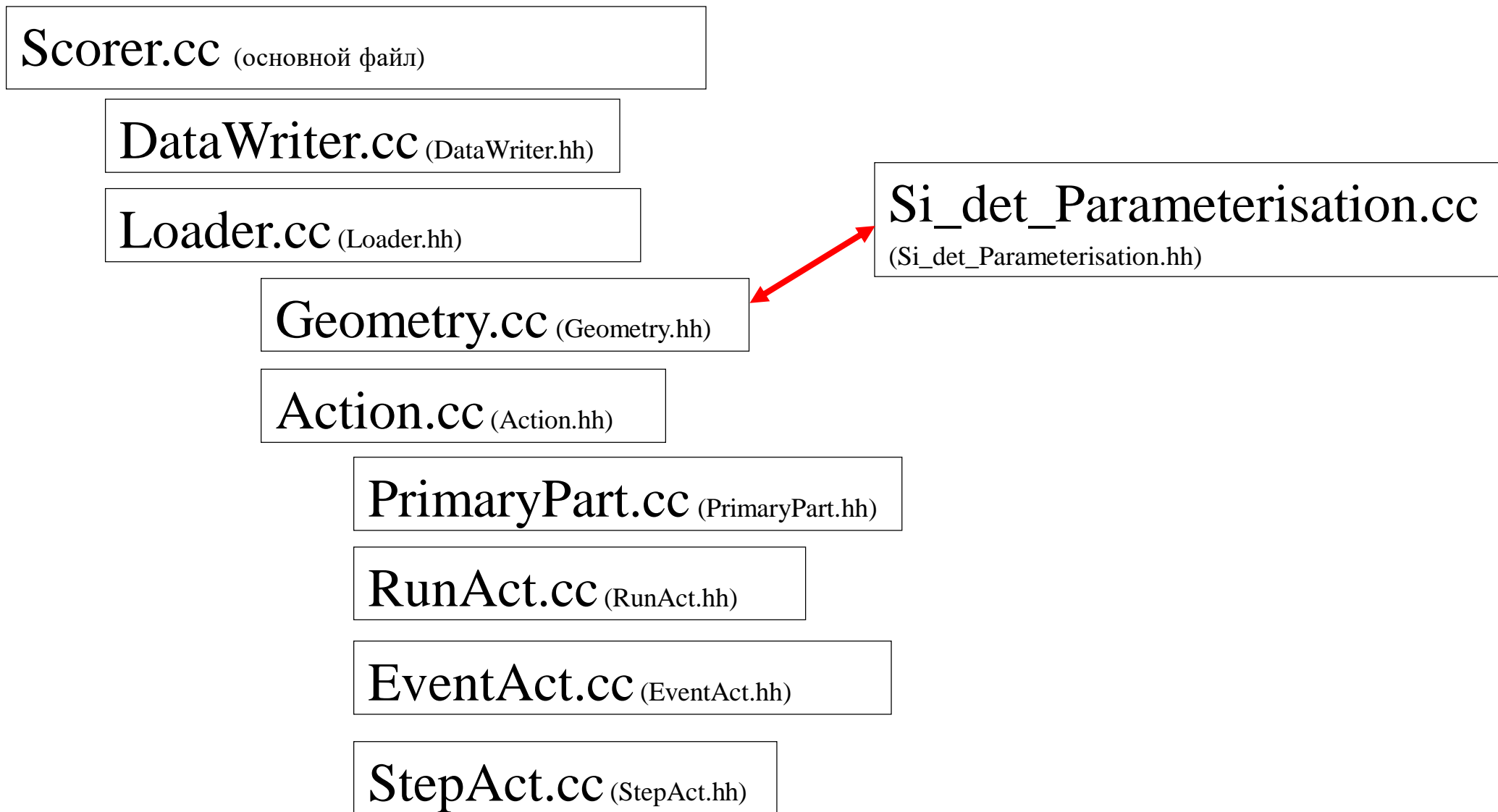
Есть возможность добавлять фильтры (G4SDParticleFilter) для отбора сохраняемых значений по характеристикам частицы





# Универсальные детекторы, счётчики, фильтры

## Проект: Score



G4SDManager\* sdman = G4SDManager::GetSDMpointer(); ← Доступ к менеджеру чувствительных объёмов

G4MultiFunctionalDetector\* detMF= new G4MultiFunctionalDetector("detMF");

G4String filterName, particleName;

G4SDParticleFilter\* electronpositronFilter=new G4SDParticleFilter(filterName="electronpositronFilter");

electronpositronFilter->add(particleName="e-");

electronpositronFilter->add(particleName="e+");

Создание объекта класса универсальных детекторов

Создание фильтра для электронов и позитронов

G4VPrimitiveScorer\* primitive;

primitive=new G4PSEnergyDeposit("edep",0);

primitive->SetFilter(electronpositronFilter);

detMF->RegisterPrimitive(primitive);

primitive=new G4PSEnergyDeposit("edep1",0);

detMF->RegisterPrimitive(primitive);

уровень иерархии объёмов

Создание стандартного счётчика для суммирования выделившейся в объёме энергии

Добавление фильтра к счётчику

Регистрация счётчика в универсальном детекторе

Создание другого стандартного счётчика без добавления фильтра

Регистрация другого счётчика в универсальном детекторе

sdman->AddNewDetector(detMF);

Si\_det\_log->SetSensitiveDetector(detMF);

Добавление сконструированного универсального детектора к логическому объёму

Добавление сконструированного универсального детектора к менеджеру чувствительных объёмов

# Универсальные детекторы, счётчики, фильтры

Стандартные фильтры:

- **G4SDChargedFilter** - все заряженные
- **G4SDNeutralFilter** - все нейтральные
- **G4SDParticleFilter** - по типу частиц
- **G4SDKineticEnergyFilter** - по диапазону энергии
- **G4SDParticleWithEnergyFilter** - по типу частиц и диапазону энергии

Счётчики

- G4PSTrackLength (сумма длин шагов частиц в объёме)
- **G4PSEnergyDeposit** (сумма энерговыделений частиц на каждом шаге в объёме)
- G4PSDoseDeposit (сумма энерговыделений частиц на каждом шаге в объёме, делённая на массу объёма)
- G4PSFlatSurfaceCurrent (для G4Box, число треков, пересекающих площадку  $-Z$  в заданном направлении: In, Out, In/Out)
- G4PSSphereSurfaceCurrent (для G4Sphere, число треков, пересекающих внутреннюю сферу в заданном направлении: In, Out, In/Out)
- G4PSCellFlux (сумма отношений длин треков частиц в объёме к объёму)
- G4PSNofSecondary (число вторичных частиц, образовавшихся в объёме)
- G4PSCellCharge (суммарный заряд частиц, остановившихся в объёме)
- ...

И другие ~50

# Универсальные детекторы, счётчики, фильтры

```

void EventAct::EndOfEventAction(const G4Event *EVE)
{.....
  G4HCofThisEvent* HCE = EVE->GetHCofThisEvent();
  G4SDManager* SDman = G4SDManager::GetSDMpointer();
  G4int mapID = SDman->GetCollectionID("detMF/edep");
  G4THitsMap<G4double>* evtMap = (G4THitsMap<G4double>*)(HCE->GetHC(mapID));
  std::map<G4int,G4double*>::iterator itr = evtMap->GetMap()->begin();
  G4cout<<"scorer edep"<<G4endl;
  for(; itr!=evtMap->GetMap()->end();itr++)
  {
    G4int key=(itr->first);
    G4double val=*(itr->second);
    G4cout<<"key="<<key<<" val="<<val<<G4endl;
  }
  mapID = SDman->GetCollectionID("detMF/edep1");
  G4THitsMap<G4double>* evtMap1 = (G4THitsMap<G4double>*)(HCE->GetHC(mapID));
  itr = evtMap1->GetMap()->begin();
  G4cout<<"scorer edep1"<<G4endl;
  for(; itr!=evtMap1->GetMap()->end();itr++)
  {
    G4int key=(itr->first);
    G4double val=*(itr->second);
    G4cout<<"key="<<key<<" val="<<val<<G4endl;
  }
}

```

Доступ к коллекции хитов для данного события

Доступ к менеджеру чувствительных объёмов

Идентификатор карты хитов для счётчика **edep** универсального детектора **detMF**

Доступ к карте хитов по идентификатору

Инициализация итератора для карты хитов

Вывод значений карты хитов для счётчика **edep** универсального детектора **detMF**

Идентификатор карты хитов для счётчика **edep1** универсального детектора **detMF**

Доступ к карте хитов по идентификатору

Инициализация итератора для карты хитов

Вывод значений карты хитов для счётчика **edep1** универсального детектора **detMF**

# Выделение областей в сложных детекторах

При моделировании сложных установок возникает необходимость более детально изучить отклик отдельных частей (трекера) или, наоборот, нет необходимости подробно моделировать отклик в веществе поглотителя.

Применяя концепцию выделения областей с различной подробностью моделирования можно значительно увеличить скорость моделирования.

Уровень подробности моделирования задаётся через порог рождения “*production cuts*”.

Выделение областей реализуется с использованием класса G4Region.

Объект класса G4Region регистрируется с использованием класса G4RegionStore.

Объект класса G4Region должен быть прикреплён к логическому объёму.

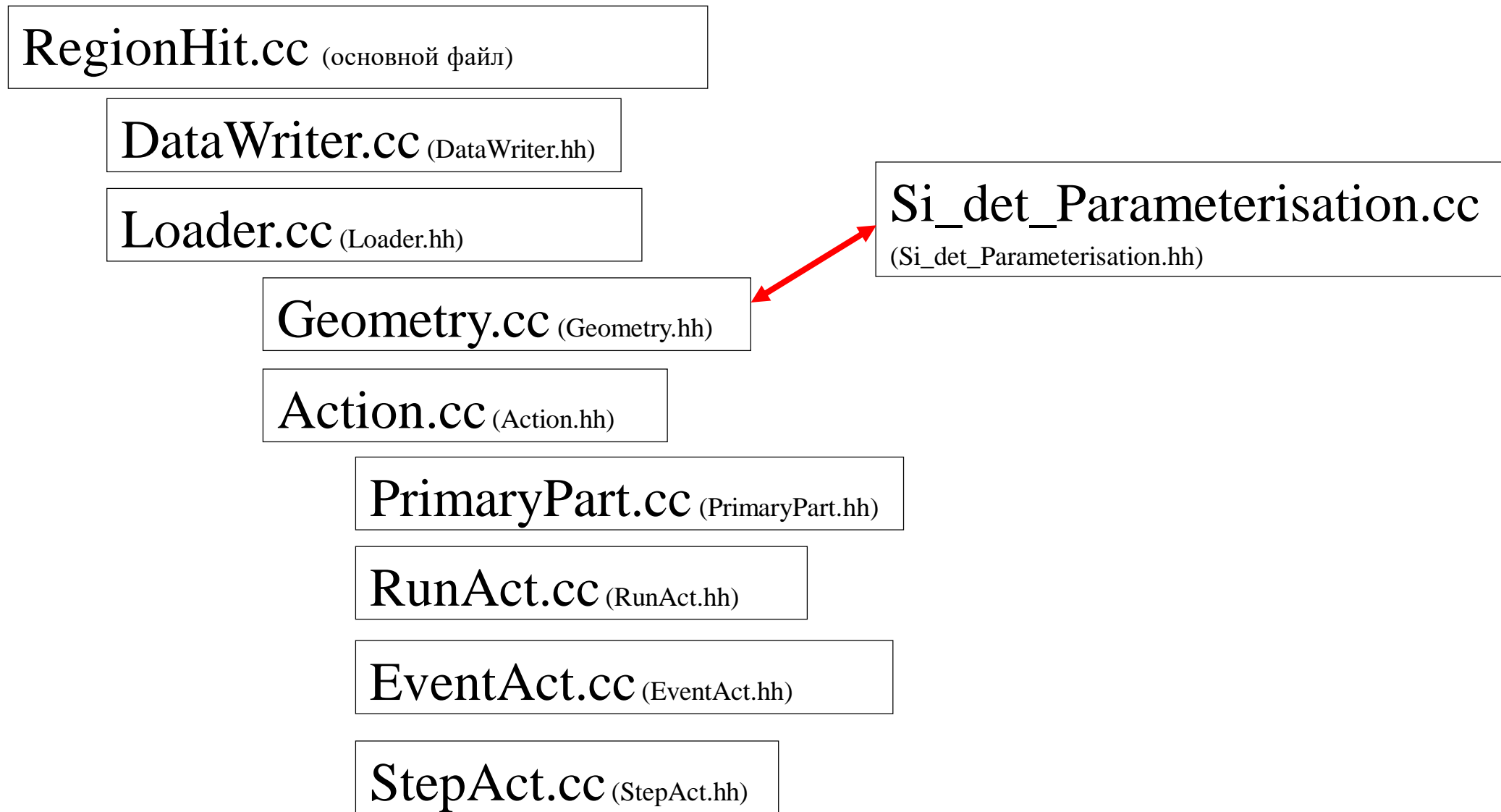
Логический объём с прикреплённым объектом класса G4Region становится корневым.

Все дочерние объёмы обладают установленным порогом рождения для корневого объёма.

Для базового объёма (мира) автоматически создаётся объект класса G4Region с порогами рождения по умолчанию.

# Выделение областей в сложных детекторах

## Проект: RegionHit



# Выделение областей в сложных детекторах

**Geometry.cc**

*Регистрация объекта класса G4Region*

```
G4Region* SILICARegion = G4RegionStore::GetInstance()->GetRegion("SILICA");  
SILICARegion->AddRootLogicalVolume(Si_log);  
G4ProductionCuts* SILICAcuts;  
SILICAcuts=new G4ProductionCuts; SILICAcuts->SetProductionCut(0.01 * mm);  
// SILICAcuts->SetProductionCut(0.001 * mm, G4ProductionCuts::GetIndex("e-"));  
SILICARegion->SetProductionCuts(SILICAcuts);
```

*Присоединение объекта класса G4Region к логическому объёму*

*Инициализация порога рождения вторичных частиц*

*Установка порога рождения для выделенной области*

*Инициализация порога рождения вторичных электронов*

*Инициализация объекта класса G4Region*

**Задание ограничений трекинга в логическом объёме**

**Loader.cc**

```
G4double maxStep = 10.*mm;  
G4double maxTrack = 10.*m;  
G4double maxTime = 1.*s;  
G4double minE = 0.001 * MeV;  
G4double minRange = 0.1 * mm;  
fStepLimit = new 4UserLimits(maxStep,maxTrack,maxTime,minE,minRange);  
Pb_log->SetUserLimits(fStepLimit);
```

```
G4VModularPhysicsList* physicsList = new QBBC;  
physicsList->RegisterPhysics(new G4StepLimiterPhysics());  
G4Region* SILICA = new G4Region("SILICA");  
runManager->SetUserInitialization(physicsList);
```