Многопоточность

В Geant4, начиная с десятой версии, была реализована поддержка многопоточных приложений в ОС UNIX.

Работа приложения в однопоточном режиме:

- инициализация ядра;
- описание геометрии;
- выбор физической модели;
- описание действий пользователя, задание первичной частицы, вывод информации.

При запуске каждое событие моделируется последовательно.

Работа приложения в многопоточном режиме:

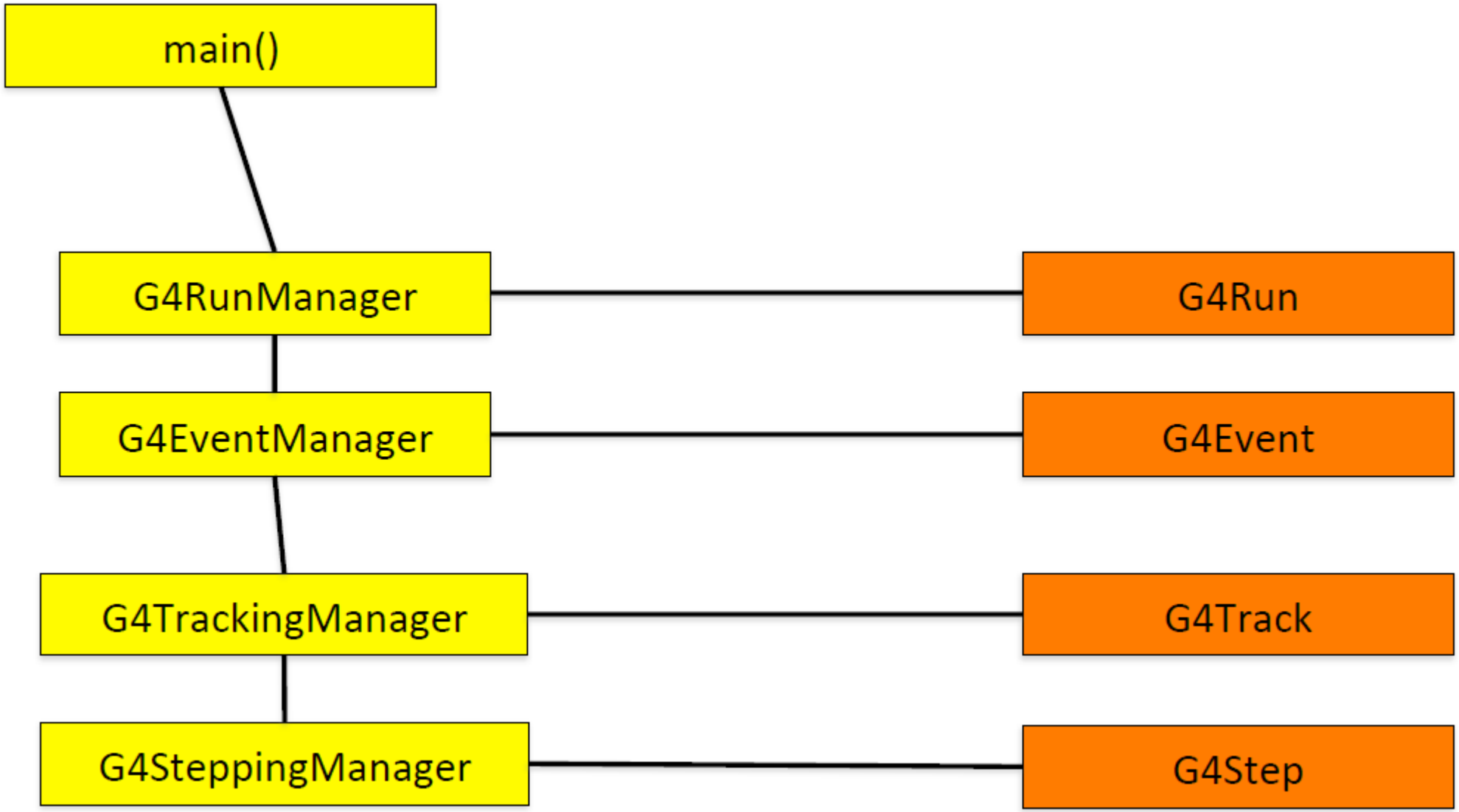
Главный поток, мастер
- инициализация ядра
- описание геометрии
- выбор физической модели

Рабочие потоки
- описание действий пользователя, задание первичной частицы
- вывод информации.

Рабочие потоки запускаются по команде мастер-потока.
На время существования рабочих потоков мастер-поток переводится в режим ожидания.
Рабочие потоки выполняются независимо.
По завершении выполнения последнего рабочего потока управление возвращается мастер-потоку.
События моделируются несколькими потоками.

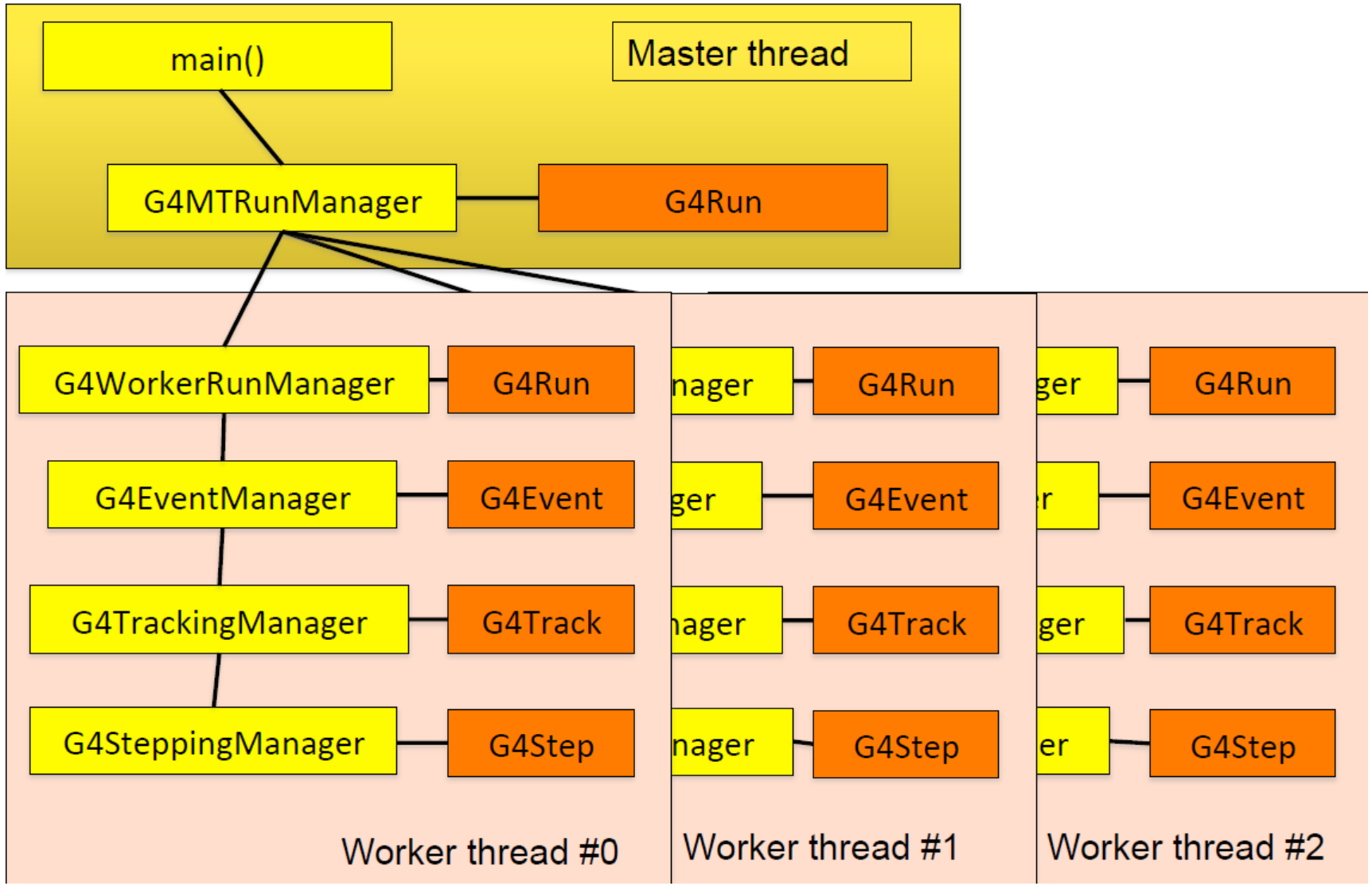
Многопоточность

Однопоточный режим



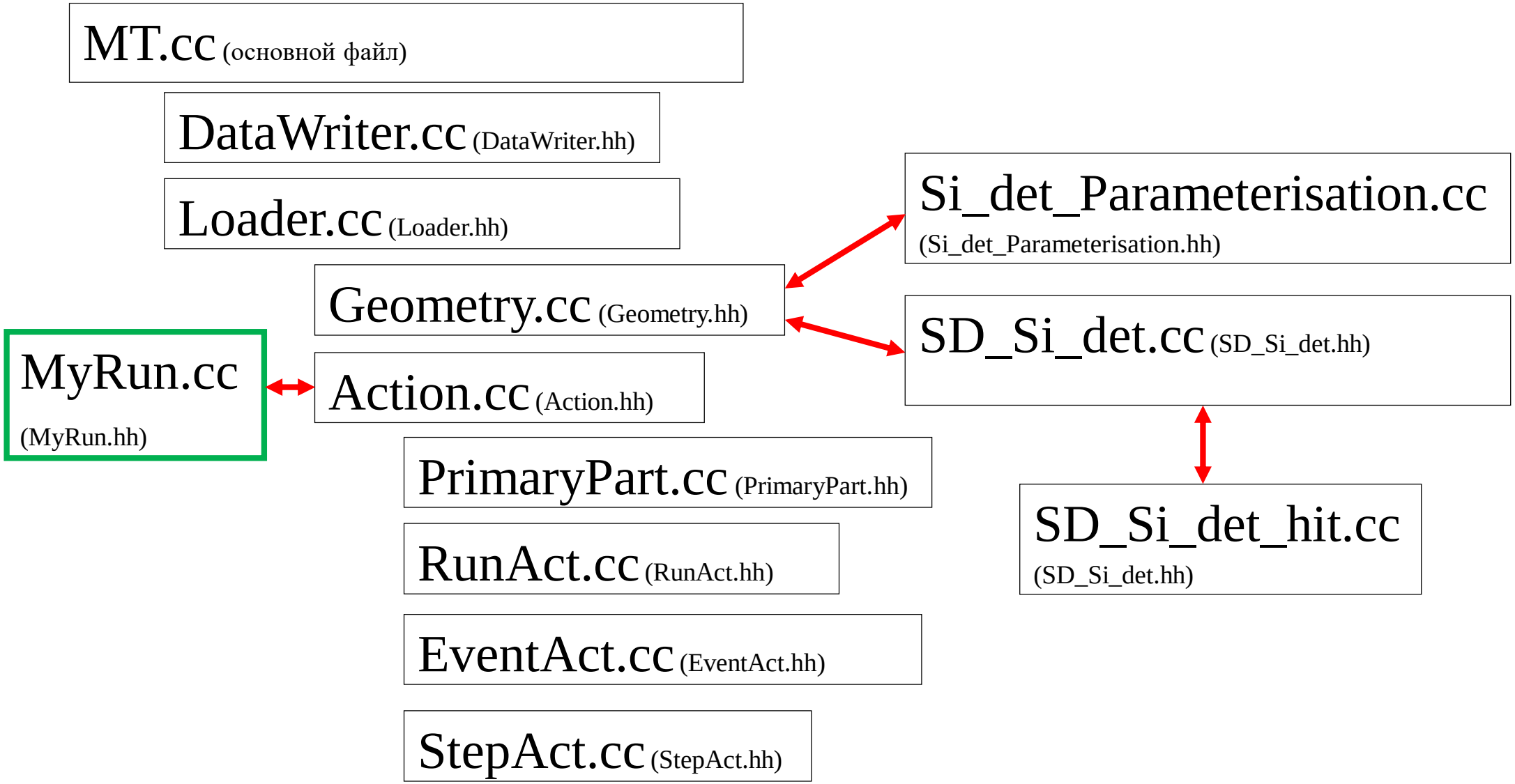
Многопоточность

Многопоточный режим



Многопоточность

проект MT



Многопоточность

Loader.cc

```
……….
#ifdef G4MULTITHREADED
  runManager = new G4MTRunManager;
// runManager->SetNumberOfThreads(G4Threading::G4GetNumberOfCores());
   runManager->SetNumberOfThreads(5);
#else
  runManager = new G4RunManager;
#endif
……….
```

Проверка поддержки многопоточности

Инициализация менеджера управления

Число доступных ядер системы

Задание числа процессов (по умолчанию равно 2)

Геометрия, чисто виртуальный метод

Если в геометрии присутствуют чувствительные объёмы, то для каждого рабочего потока необходимо создать свой объект класса чувствительных объёмов в методе ConstructSDandField() класса G4VUserDetectorConstruction

Public Member Functions

	G4VUserDetectorConstruction ()
virtual	**~G4VUserDetectorConstruction** ()
virtual **G4VPhysicalVolume** *	**Construct** ()=0
virtual void	**ConstructSDandField** ()
virtual void	**CloneSD** ()
virtual void	**CloneF** ()
void	**RegisterParallelWorld** (**G4VUserParallelWorld** *)
G4int	**ConstructParallelGeometries** ()
void	**ConstructParallelSD** ()
G4int	**GetNumberOfParallelWorld** () const
G4VUserParallelWorld *	**GetParallelWorld** (**G4int** i) const

Многопоточность

Geometry.cc

```
G4VPhysicalVolume* Geometry::Construct()
{
  ……
}

void Geometry::ConstructSDandField()
{
  G4SDManager* sdman = G4SDManager::GetSDMpointer();
  SD_Si_det* sensitive_Si_det = new SD_Si_det("/mySi_det",filename);
  sdman->AddNewDetector(sensitive_Si_det);
  SetSensitiveDetector(this->Si_det_log,sensitive_Si_det);
}
```

Action.cc

```
void Action::Build() const

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

Action.hh

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

Public Member Functions

	G4VUserActionInitialization ()
virtual	**~G4VUserActionInitialization** ()
virtual void	**Build** () const =0
virtual void	**BuildForMaster** () const
virtual **G4VSteppingVerbose** *	**InitializeSteppingVerbose** () const

Класс действий запуска (Run) для мастер-потока

Многопоточность

Суммируется энерговыделение всех событий

RunAct.hh

```
class RunAct : public G4UserRunAction
{
 public:
        RunAct(std::ofstream& ofsa);
        ~RunAct();
        std::ofstream *f_act;
        G4Run* GenerateRun() override;
        void Add_totalE(G4double e);
        void BeginOfRunAction(const G4Run*);
        void EndOfRunAction(const G4Run*);
        MyRun* fMyRun;
        static void AddE_total(G4double dE);
};
```

RunAct.cc

```
……….
G4double TotalEsum=0.;
G4Run* RunAct::GenerateRun()
{
 fMyRun = new MyRun();
 return fMyRun;
}
……..
```

Инициализация объекта класса запуска для локального потока – наследника G4Run

Метод суммирования энергии каждого события в рабочем потоке

RunAct.cc

```
void RunAct::Add_totalE(G4double e)
{ fMyRun->Add_totalE(e);}
```

Объявление объекта класса запуска для локального потока-наследника G4Run как члена класса пользователя RunAct

Статический метод суммирования энергии каждого события в общую переменную класса RunAct

RunAct.cc

```
void RunAct::AddE_total(G4double edep)
{TotalEsum+=edep;}
```

1. Суммирование в общую переменную класса RunAct TotalEsum.

2. Суммирование в локальные переменные Total_dE каждого рабочего потока.

Многопоточность

MyRun.hh

```
class MyRun : public G4Run
 {
  public:  MyRun();
           ~MyRun() override;
           void Add_totalE(G4double e);
           G4double Get_totalE();
           void Merge(const G4Run*) override;
  private:
          G4double Total_dE;
 };
```

Метод суммирования энергии каждого события в рабочем потоке

Метод объединения результатов выполнения рабочих потоков. Вызывается после завершения последнего рабочего потока.

MyRun.cc

```
void MyRun::Add_totalE(G4double e)
{ Total_dE+=e;}
G4double MyRun::Get_totalE()
{return Total_dE;}
void MyRun::Merge(const G4Run* aRun)
{
 const MyRun* localRun = static_cast<const MyRun*>(aRun);
 Total_dE += localRun->Total_dE;
 G4Run::Merge(aRun);
}
```

Метод суммирования энергии каждого события в рабочем потоке

Метод объединения результатов выполнения рабочих потоков. Вызывается после завершения последнего рабочего потока.

Доступ к объекту класса запуска G4Run для текущего рабочего потока

Объединение результатов выполнения рабочих потоков для переменной Total_dE

Многопоточность

EventAct.cc

```
namespace {G4Mutex EventMutex = G4MUTEX_INITIALIZER;}
G4double Esum=0.;
……..
void EventAct::AddE(G4double edep)
{Esum+=edep;}
……..
void EventAct::EndOfEventAction(const G4Event *EVE)
{
 G4AutoLock lock(EventMutex);
  MyRun* currentRun = static_cast<MyRun*>(G4RunManager::GetRunManager()->GetNonConstCurrentRun());
 currentRun->Add_totalE(Esum);
 int thr=G4Threading::G4GetThreadId();
 RunAct::AddE_total(Esum);
……..
}
```

Описание мьютекса

Суммирование энергии на каждом шаге для текущего события в рабочем потоке.

Статический метод вызывается на каждом шаге из класса StepAct.

Мьютекс - семафор

Доступ менеджеру текущего рабочего потока

Суммирование энергии каждого события в рабочем потоке

Идентификатор текущего рабочего потока

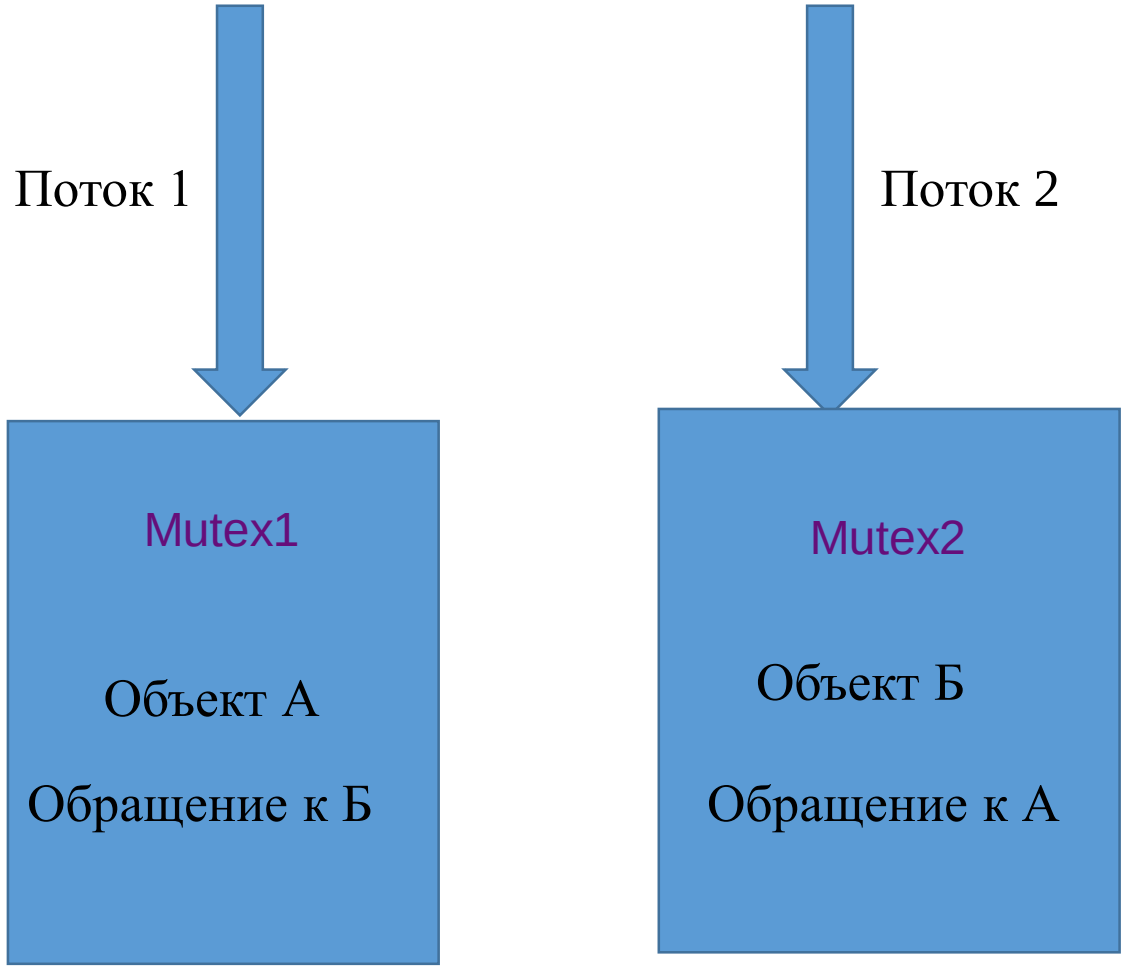
Суммирование энергии каждого события в общую переменную класса RunAct

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом.

В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом.

Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток блокируется до тех пор, пока мьютекс не будет освобождён.

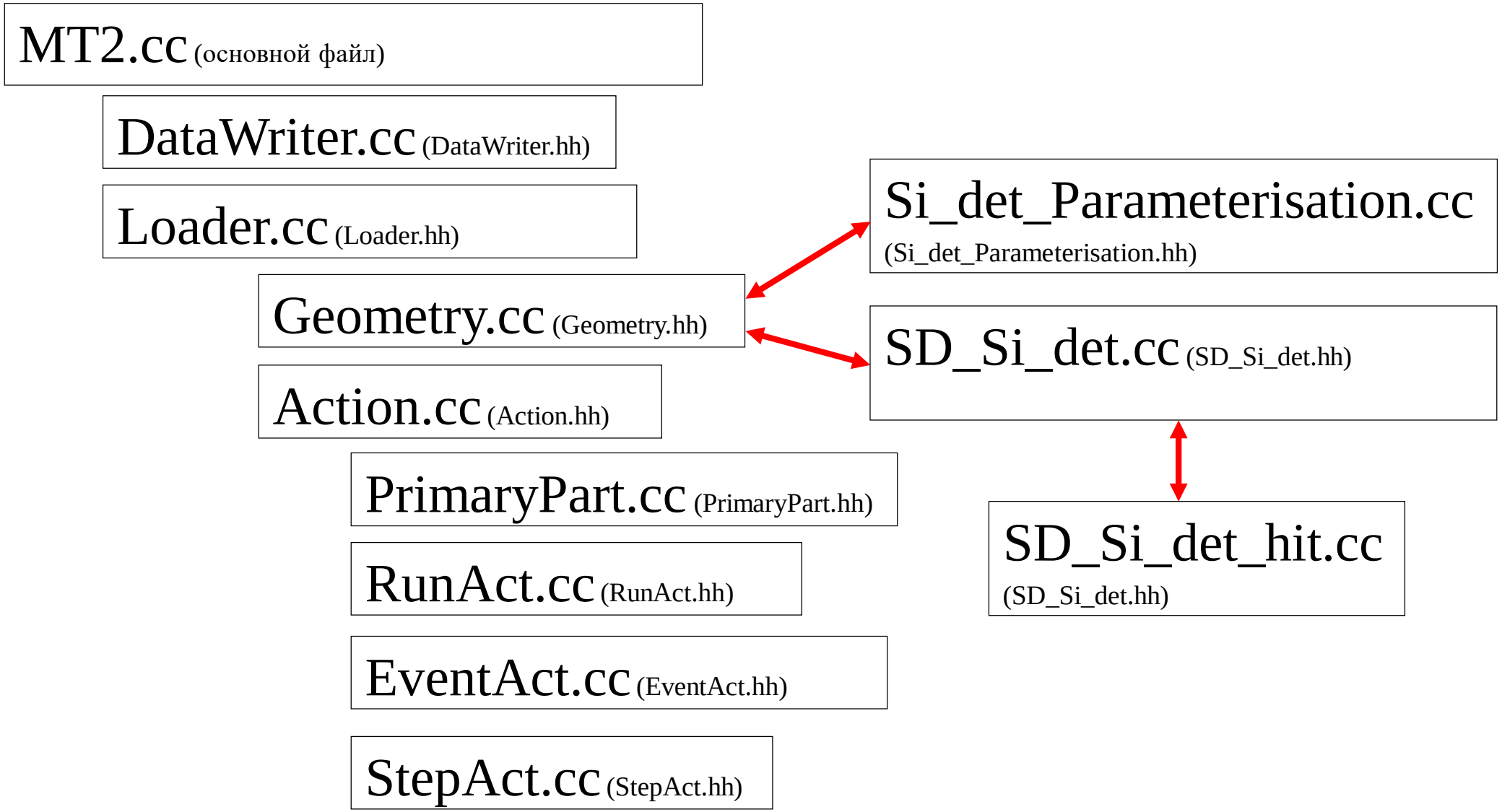
Многопоточность



Множественные мьютексы могут привести к **dead-lock**

Многопоточность

проект MT2



Многопоточность

RunAct.hh

```
class RunAct :  public G4UserRunAction
{
 public:
        RunAct(std::ofstream& ofsa);
        ~RunAct();
        std::ofstream *f_act;
        void Add_totalE(G4double e);
        G4double Get_totalE();
        void BeginOfRunAction(const G4Run*);
        void EndOfRunAction(const G4Run*);
        static void AddE_total(G4double dE);
 private:
        G4Accumulable<G4double> total_dE=0.;
};
```

Метод суммирования энергии каждого события в рабочем потоке

Статический метод суммирования энергии каждого события в общую переменную класса RunAct

Переменная для объединение результатов выполнения рабочих потоков

```
G4Accumulable (T initValue, G4MergeMode mergeMode=G4MergeMode::kAddition)
```

Вид объединения по умолчанию: сложение

```
enum class G4MergeMode
{kAddition,
 kMultiplication,
 kMaximum,
 kMinimum};
```

Многопоточность

RunAct.cc

```
RunAct::RunAct(std::ofstream& ofsa)
 {
  ……..
  G4AccumulableManager* accumulableManager = G4AccumulableManager::Instance();
  accumulableManager->RegisterAccumulable(total_dE);
 }
 ……..
 void RunAct::EndOfRunAction(const G4Run* aRun)
 {
  ……..
  G4AccumulableManager* accumulableManager = G4AccumulableManager::Instance();
  accumulableManager->Merge();
  ……..
 }
```

Доступ к менеджеру объединения результатов

Регистрация переменной для объединение результатов выполнения рабочих потоков в менеджере

Объединение результатов выполнения рабочих потоков для переменной total_dE

Доступ к менеджеру объединения результатов

Многопоточность, загрузка и сохранение состояния генератора случайных чисел

проект RandMT

RunAct.cc

```
void RunAct::BeginOfRunAction(const G4Run* aRun)
 {
 G4cout << "\n---Start------ Run # "<<RunNum<<" --------\n" <<"RunId="<<aRun->GetRunID()<< G4endl;
 time(&Start);
 G4RunManager::GetRunManager()->SetRandomNumberStoreDir("./random");
 G4RunManager::GetRunManager()->SetRandomNumberStore(true);
}
```

Директория для сохранения состояния генератора случайных чисел

Сохранение состояние генератора случайных чисел для всего запуска и для текущего события

EventAct.cc

```
void EventAct::BeginOfEventAction(const G4Event * EVE)
{
 ……..
G4RunManager::GetRunManager()->rndmSaveThisEvent();
}
```

Сохранение состояние генератора случайных чисел для всех событий всех запусков

Многопоточность, загрузка и сохранение состояния генератора случайных чисел

PrimaryPart.cc

```
void PrimaryPart::GeneratePrimaries(G4Event* anEvent)
 {
   ……..
   std::string fileName="./random/runevt.rndm";
   std::string fileName1="./random/myfile.rndm";
   CLHEP::HepRandom::getTheEngine()->restoreStatus(fileName.c_str());
   CLHEP::HepRandom::getTheEngine()->saveStatus(fileName1.c_str());
   ……..
}
```

Имя файла для загрузки заданного состояния генератора случайных чисел

Имя файла для сохранения текущего состояния генератора случайных чисел

Загрузка заданного состояния генератора случайных чисел

Сохранение текущего состояния генератора случайных чисел