

# Многопоточность

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

## Определения

- Процесс – отдельная инстанция исполняемого программного кода, имеющая собственные параметры исполнения, выделенную память и другие системные ресурсы.
- Поток – параллельные «независимые» исполнения внутри процесса. Потоки имеют общую память и системные ресурсы, выделенные на весь процесс в целом.

## Пособытийный параллелизм

- Каждое событие может быть рассчитано независимо от других
  - требуется «не слишком много» памяти
  - «не слишком много» вычислений
- Много переменных «статического» типа
  - геометрия
  - физические листы (таблицы сечений, характеристики частиц и т. д.)
  - электромагнитные поля (если есть)

CMake: -DGEANT4\_BUILD\_MULTITHREADED=ON

Эффективное распределение памяти может значительно оптимизировать скорость вычислений

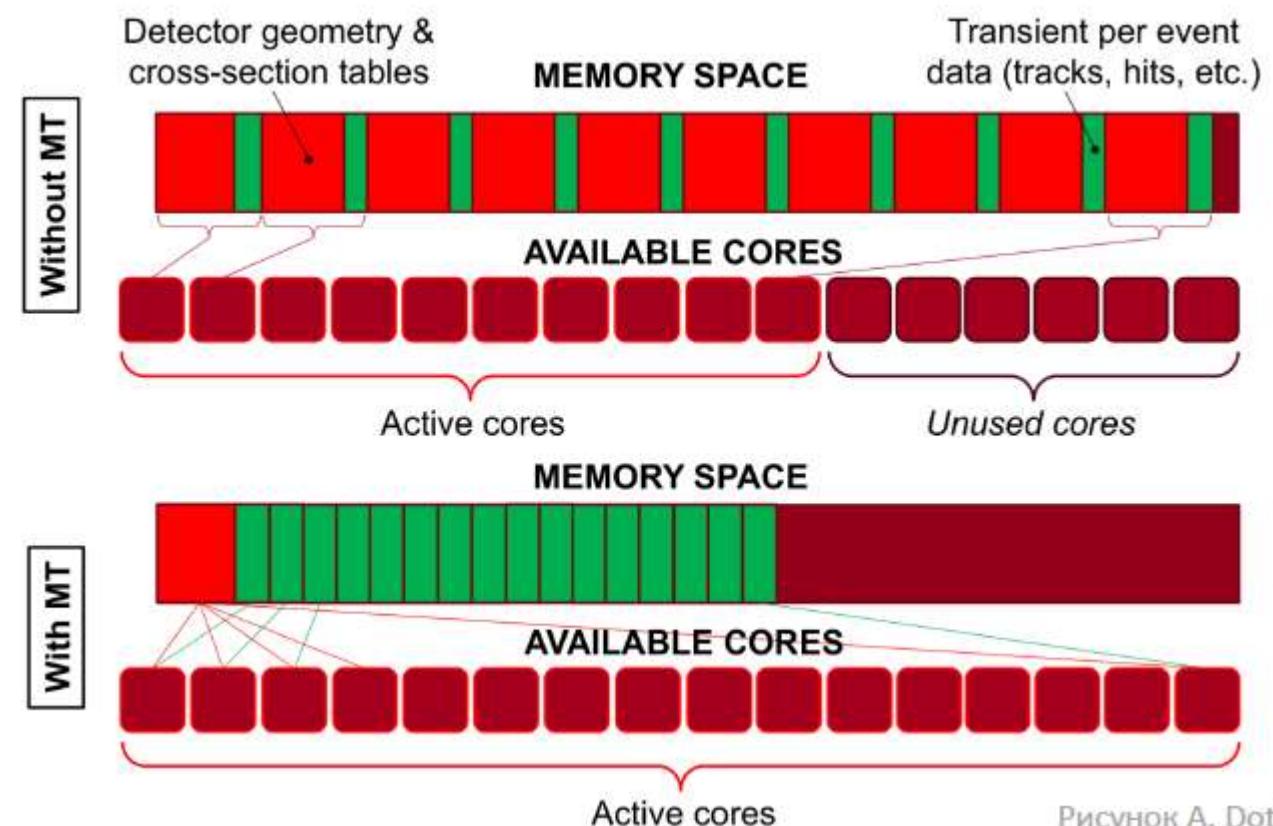


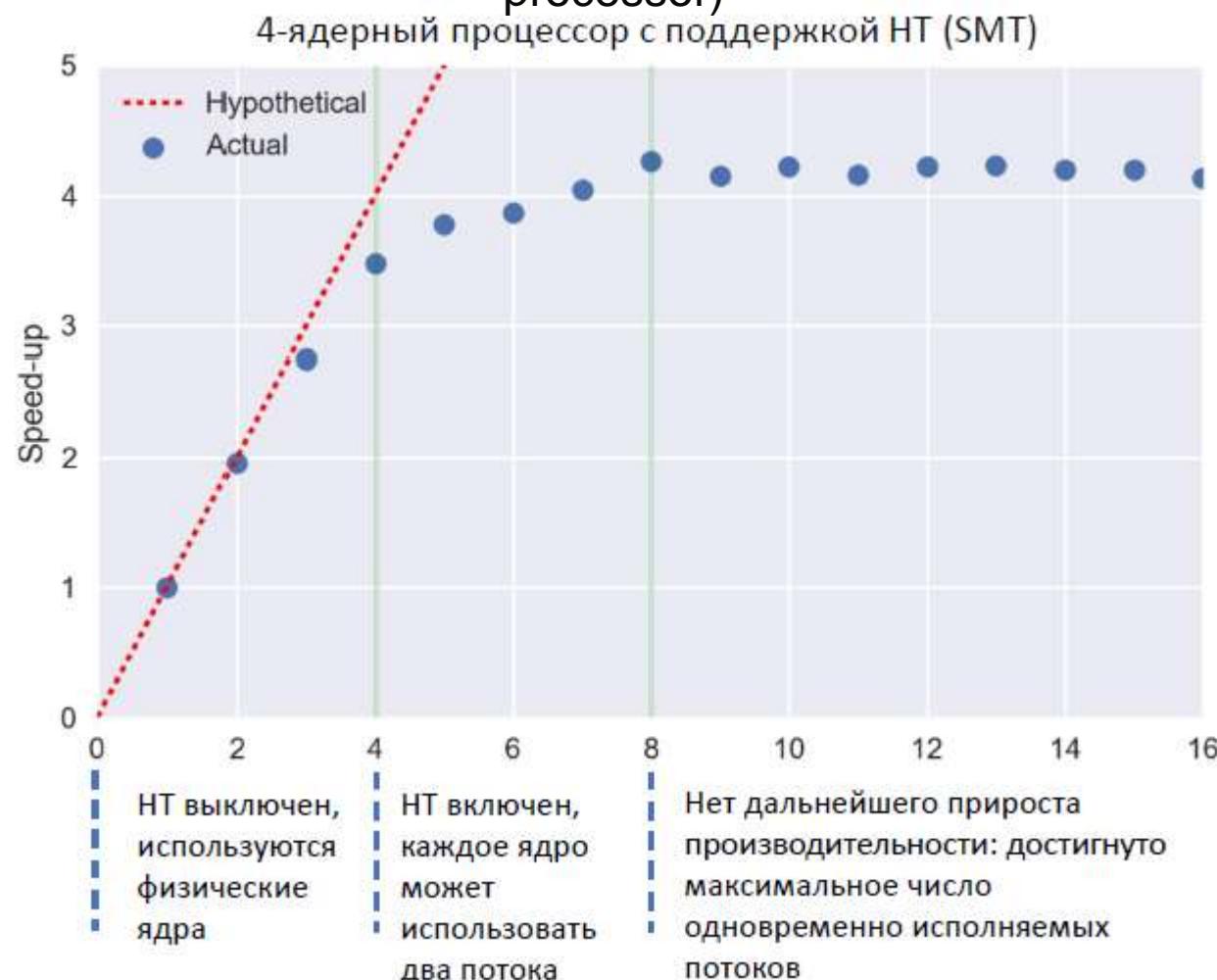
Рисунок А. Доти

# Многопоточность

## Производительность в многопоточном режиме

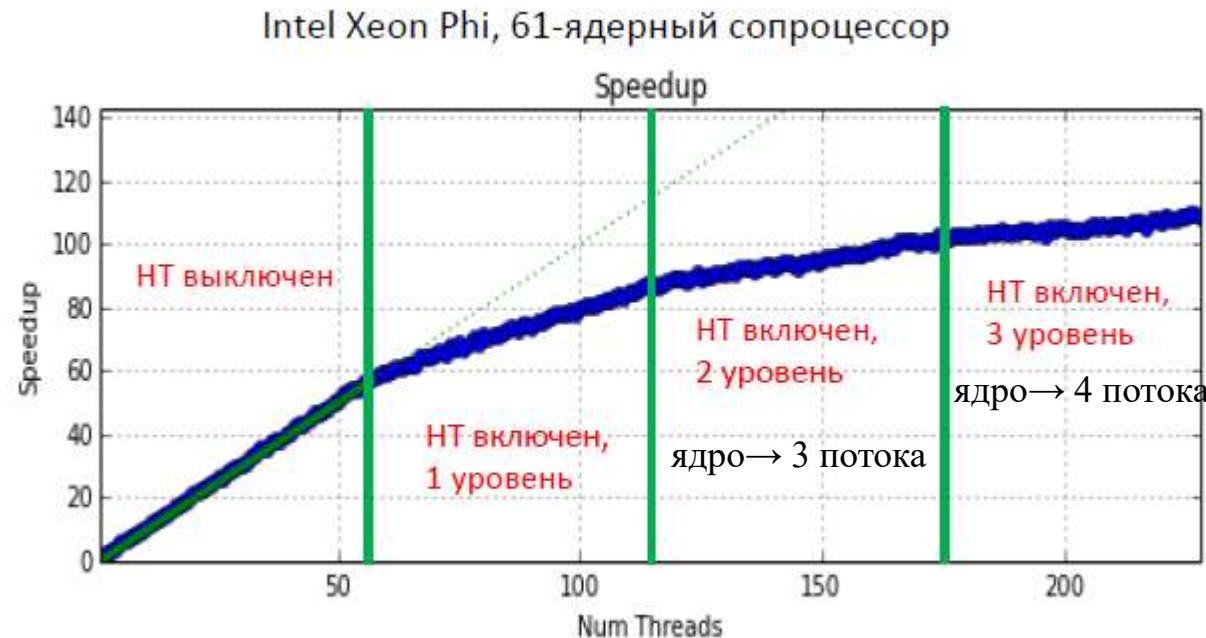
Каждое ядро использует два потока

**Hyper-Threading Technology (Intel processor)**  
**Simultaneous Threading Technology (AMD processor)**



# Многопоточность

## Производительность в многопоточном режиме



Преимущества использования многопоточного режима:

- Эффективное использование ресурсов
- Синхронизация внутри процесса

Возможные проблемы при использования многопоточного режима:

- Сложнее (правильно) писать код
- Сложнее осуществлять отладку кода
- Опасность: dead-lock
- Опасность: неправильное обращение к адресу памяти
- Затраты на синхронизацию потоков

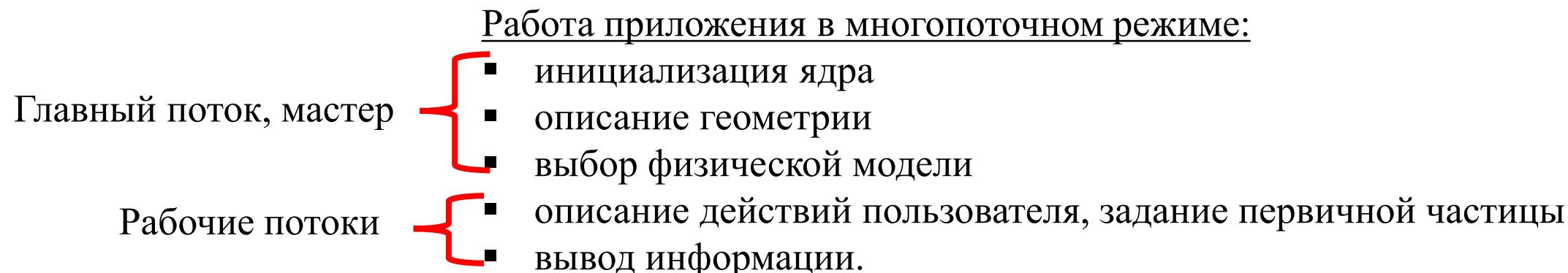
- Параллелизм в Geant4 основан на стандартах POSIX, а именно на библиотеке pthreads. Весь функционал был обернут в специальные имена Geant4.
- Заданная геометрия и физические модели являются общими в памяти объектами, с точки зрения потоков являются данными типа *read-only*
- G4VUserDetectorConstruction, G4VUserPhysicsList и G4UserActionInitialization являются общими классами, остальные классы пользовательских действий – локальные

# Многопоточность

## 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.



Рабочие потоки запускаются по команде мастер-потока.

На время существования рабочих потоков мастер-поток переводится в режим ожидания.

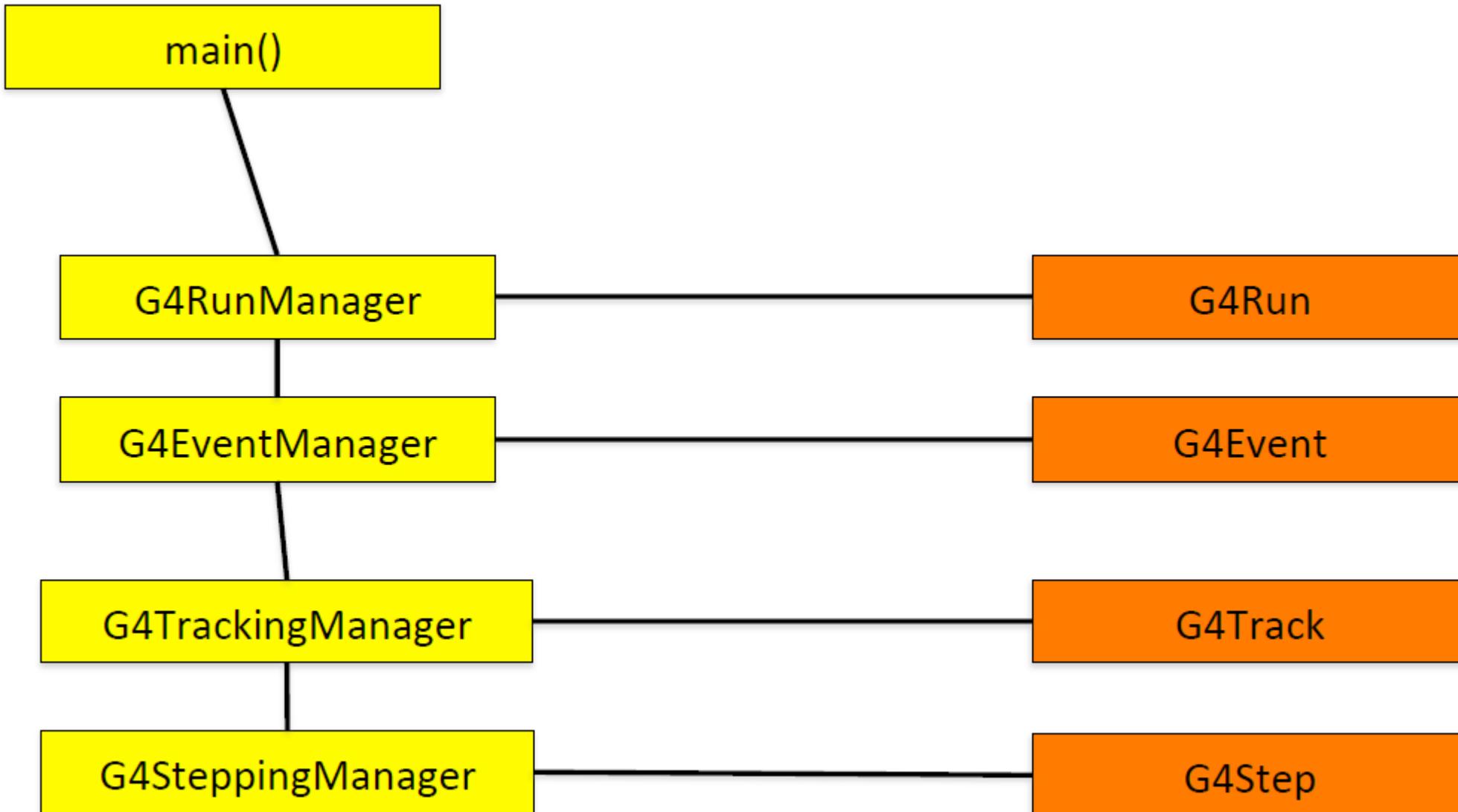
Рабочие потоки выполняются независимо.

По завершении выполнения последнего рабочего потока управление возвращается мастер-потоку.

События моделируются несколькими потоками.

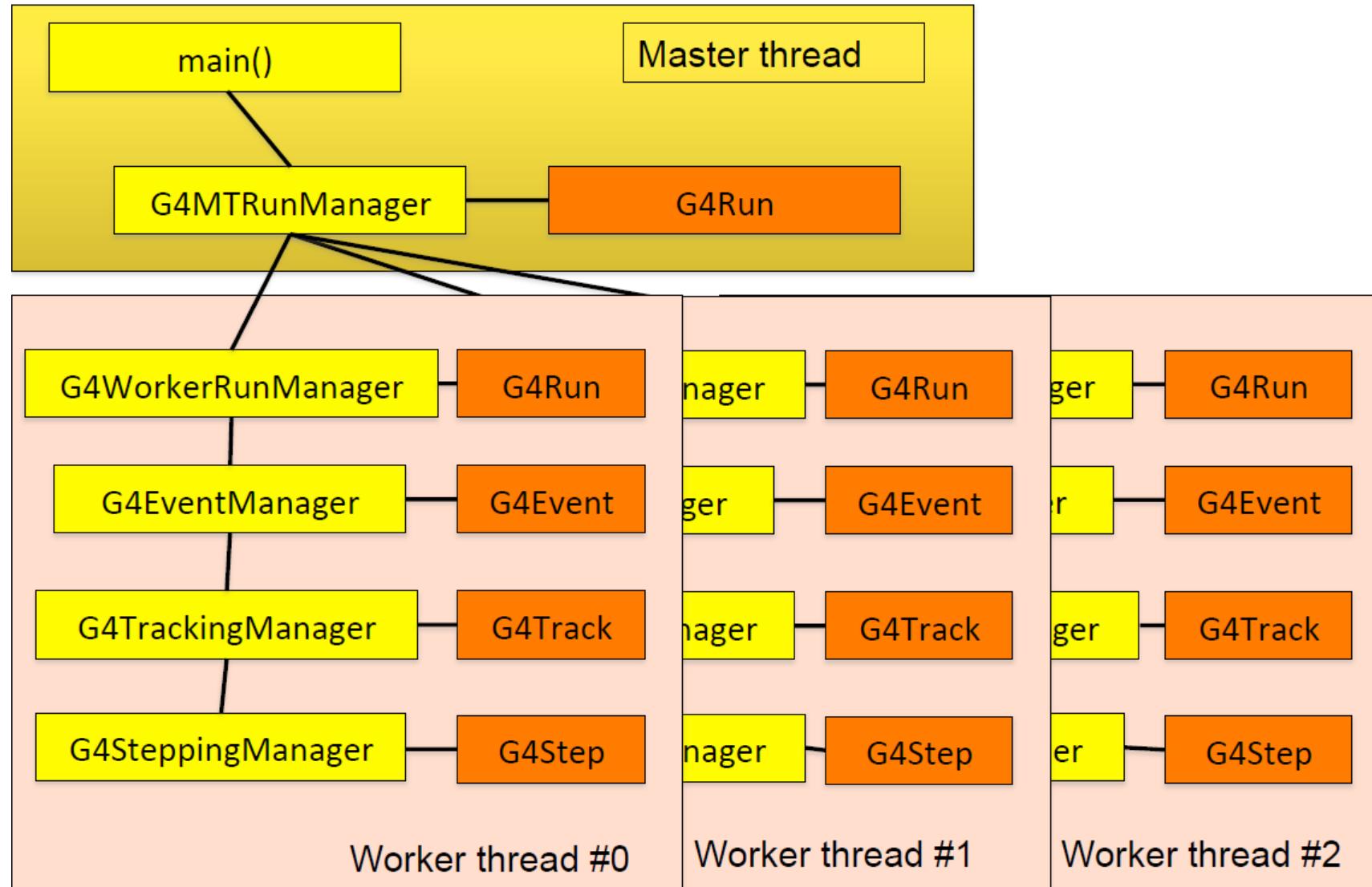
# Многопоточность

## Однопоточный режим



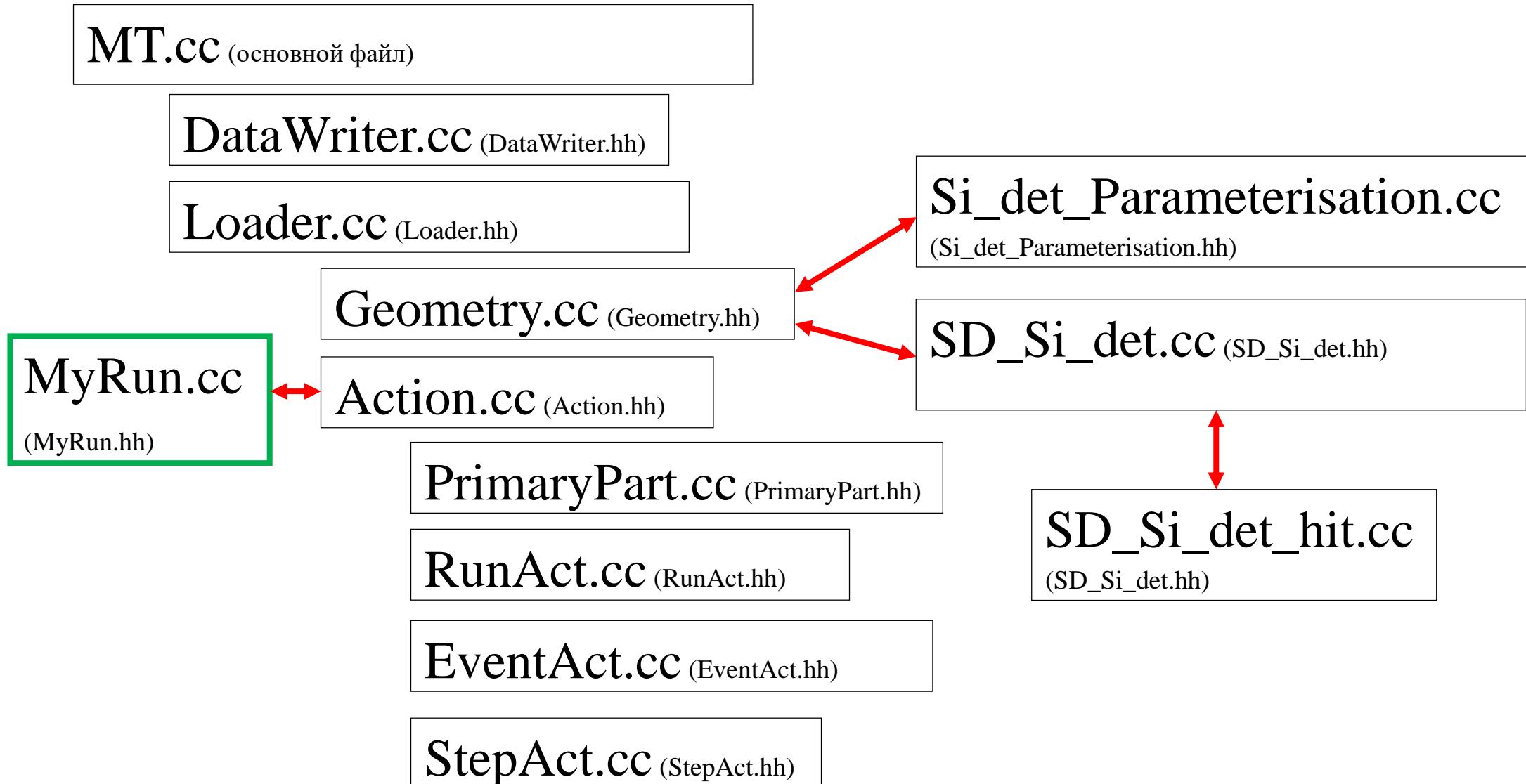
# Многопоточность

## Многопоточный режим



## проект МТ

# Многопоточность



# 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)

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

## *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);
}
```

# МНОГОПОТОЧНОСТЬ

## *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));
}
```

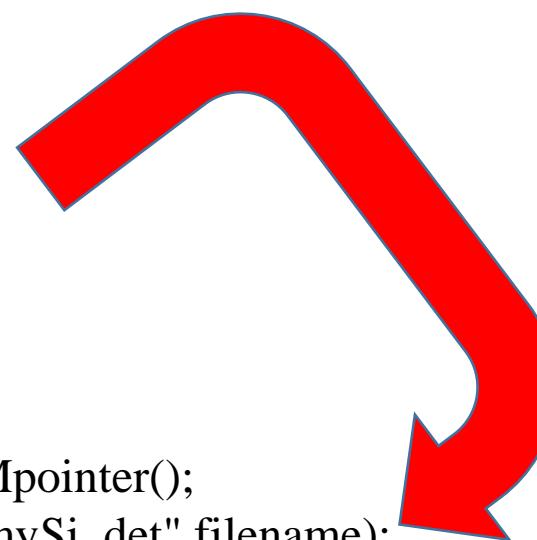
## *Action.hh*

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

### Public Member Functions

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

*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);
};
```

Статический метод суммирования энергии каждого события в общую переменную класса RunAct

*RunAct.cc*

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

*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

- Суммирование в общую переменную класса RunAct TotalEsum.*
- 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); ← Метод суммирования энергии каждого события в рабочем потоке
    G4double Get_totalE();
    void Merge(const G4Run*) override; ← Метод объединения результатов выполнения рабочих потоков.
private:
    G4double Total_dE;
};
```

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

*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);
}
```

# МНОГОПОТОЧНОСТЬ

```

namespace { G4Mutex EventMutex = G4MUTEX_INITIALIZER; } ← Описание мьютекса
G4double Esum=0.;
.....
void EventAct::AddE(G4double edep)      Суммирование энергии на каждом шаге для текущего события в
{Esum+=edep;}                         рабочем потоке.

.....
void EventAct::EndOfEventAction(const G4Event *EVE)      Статический метод вызывается на каждом шаге из класса StepAct.
{
    G4AutoLock lock(EventMutex);           Мьютекс - семафор
    MyRun* currentRun = static_cast<MyRun*>(G4RunManager::GetRunManager()->GetNonConstCurrentRun());
    currentRun->Add_totalE(Esum);          Доступ к менеджеру текущего рабочего потока
    int thr=G4Threading::G4GetThreadId();   Суммирование энергии каждого события в рабочем потоке
    RunAct::AddE_total(Esum);              Идентификатор текущего рабочего потока
    .....                                Суммирование энергии каждого события в общую переменную класса RunAct
}

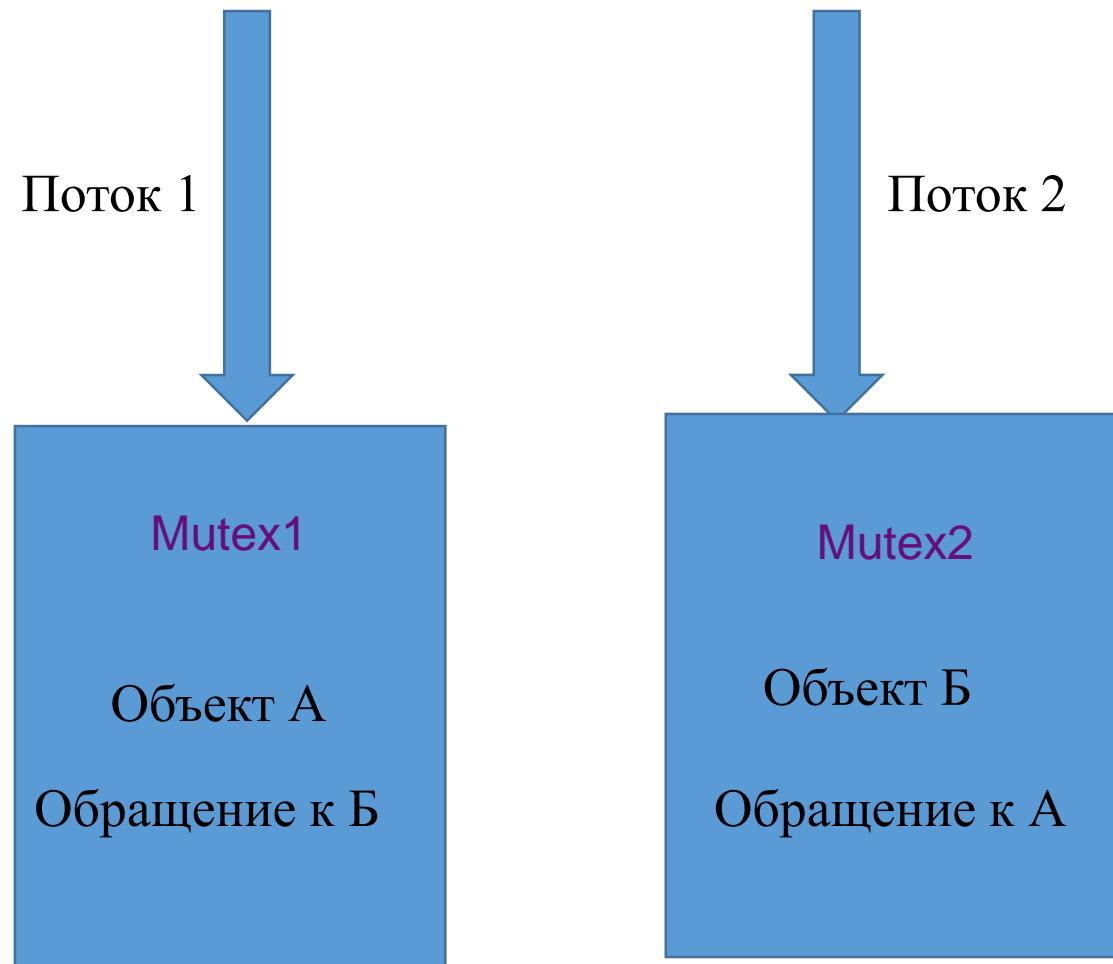
```

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.

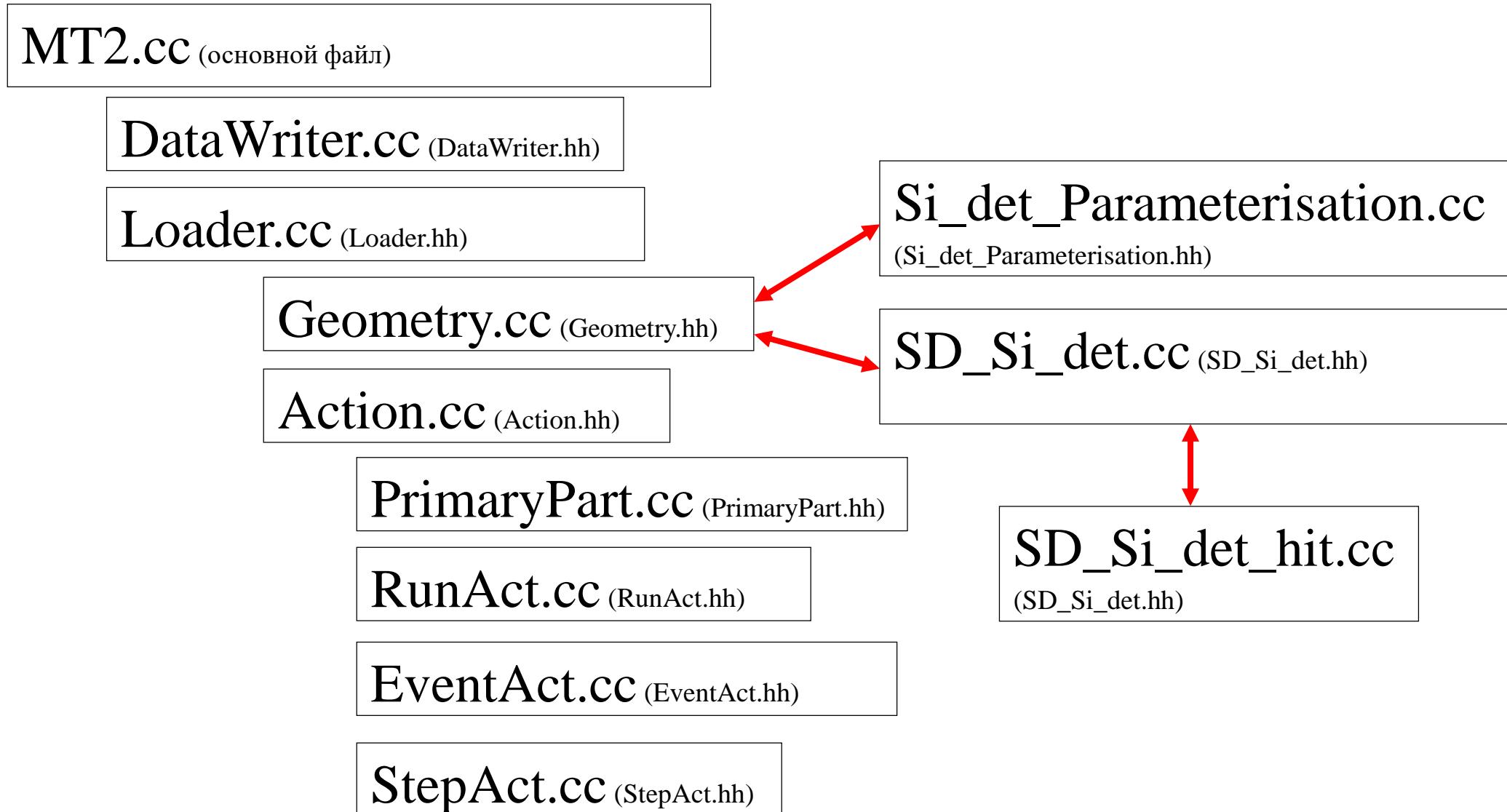
# Многопоточность



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

# Многопоточность

## проект MT2



# Многопоточность

```
class RunAct : public G4UserRunAction
{
public:
    RunAct(std::ofstream& ofsa);
    ~RunAct();
    std::ofstream *f_act;
    void Add_totalE(G4double e); ← Метод суммирования энергии каждого события в рабочем потоке
    G4double Get_totalE();
    void BeginOfRunAction(const G4Run* );
    void EndOfRunAction(const G4Run* );
    static void AddE_total(G4double dE); ← Статический метод суммирования энергии каждого события в общую переменную класса RunAct
private:
    G4Accumable<G4double> total_dE=0.;
```

↑  
Переменная для объединение результатов выполнения рабочих потоков

**G4Accumable** (T initialValue, **G4MergeMode** mergeMode=**G4MergeMode::kAddition**)

Вид объединения по умолчанию: сложение

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

# МНОГОПОТОЧНОСТЬ

*RunAct.cc*

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

{

.....

G4AccumableManager\* accumableManager = G4AccumableManager::Instance();  
accumableManager->RegisterAccumable(total\_dE);

}

.....

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

{

.....

G4AccumableManager\* accumableManager = G4AccumableManager::Instance();  
accumableManager->Merge();

}

.....

*Объединение результатов выполнения  
рабочих потоков для переменной total\_dE*

*Доступ к менеджеру объединения результатов*

*Регистрация  
переменной для  
объединения  
результатов  
выполнения рабочих  
потоков в менеджере*

*Доступ к менеджеру объединения результатов*