Настоящее и будущее copy elision

Антон Полухин
Yandex Taxi
expert developer
antoshkka@gmail.com

Роман Русяев
Samsung R&D
compiler developer
rusyaev.rm@gmail.com

План доклада

- настоящее: **copy elision**
  - описание
  - ограничения
  - реализация
- будущее: **ultimate copy elision**
  - описание
  - улучшения
  - реализация, результаты

Цель доклада

Рассказать о новом пропозале – ultimate copy elision, показав, каким образом он улучшит жизнь разработчиков C++

Copy Elision

Что такое copy elision?

Что такое copy elision

Компилятор не использует конструктор копирования или перемещения, и производит in-place конструирование объекта.

[class.copy.elision]

Немного истории

Немного истории (90-ые)

Немного истории (90-ые)

“Если объект копируется и либо копия, либо оригинал больше не используются — ...

Немного истории (90-ые)

“Если объект копируется и либо копия, либо оригинал больше не используются — разрешим компиляторам схлопывать их до одного объекта.”

Возникшие проблемы

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
```

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
std::string_view sview = s1;
```

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
std::string_view sview = s1;
std::string s2 = s1;
```

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
std::string_view sview = s1;
std::string s2 = s1;
s2 = "Beware of 2020!";
```

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
std::string_view sview = s1;
std::string s2 = s1;
s2 = "Beware of 2020!";

std::cout << sview;
```

Возникшие проблемы

```
std::string s1 = "Hello world from 90s";
std::string_view sview = s1;
std::string s2 = s1;
s2 = "Beware of 2020!";

std::cout << sview; // BOOM!
```

Возникшие проблемы

*"Давайте **пока** сделаем copy elision только для возврата из функций..."*

Возникшие проблемы

*"Давайте **пока** сделаем copy elision только для возврата из функций.*

*А **потом** кто-нибудь поправит!"*

20 лет спустя...

NRVO

NRVO

```
std::string NRVO() {
    std::string s = "Hello world from 90s";
    return s; // NRVO в деле
}
```

NRVO

```
std::string NRVO() {
    std::string s = "Hello world from 90s";
    return s; // NRVO в деле
}

void usage() {
    std::string x = NRVO();
    // ...
}
```

NRVO (псевдокод)

```
std::string NRVO(std::string* x) {
    new (x) std::string("Hello world from 90s");

}

void usage() {
    std::string x = NRVO(&x);
    // ...
}
```

Guaranteed copy elision C++17

Guaranteed copy elision C++17

```
std::string Materialization1() {
    return std::string{"Hello world from 2017"};
}
```

Guaranteed copy elision C++17

```
std::string Materialization1() {
    return std::string{"Hello world from 2017"};
}

std::string Materialization2() {
    return std::string{ std::string{ std::string{
                std::string{"Hello world from 2017"}
    }}};
}
```

Copy elision не работает:

Copy elision не работает:

- **Для параметров функций**

Copy elision не работает:

```
Widget foo(Widget w) {
    return w; // NRVO пасует
}
```

Copy elision не работает:

- Для параметров функций
- **Для возвращаемого значения в операторе return, отличного от типа возвращаемого значения функции**

Copy elision не работает:

```
Widget foo() {
    Widget w;
    Widget& rw = w;
    return rw; // NRVO пасует
}
```

Copy elision не работает:

- Для параметров функций
- Для возвращаемого значения в операторе return, отличного от типа возвращаемого значения функции
- **Если в операторе return что-то отличное от имени переменной**

Copy elision не работает:

```
Widget foo() {
    Widget w;
    return std::move(w); // NRVO пасует
}
```

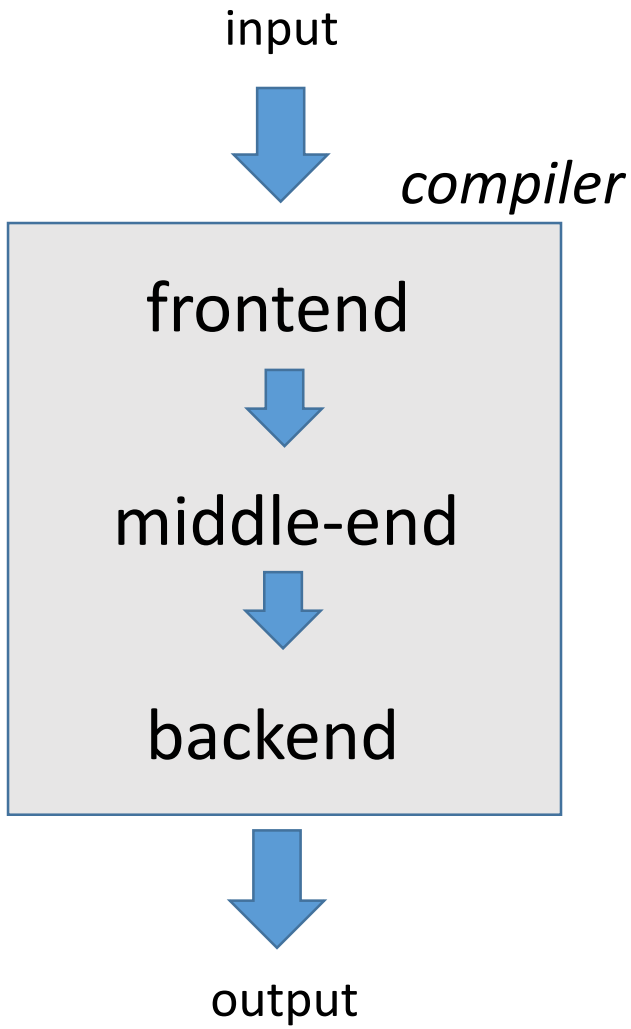
Почему такие ограничения?

Почему такие ограничения?

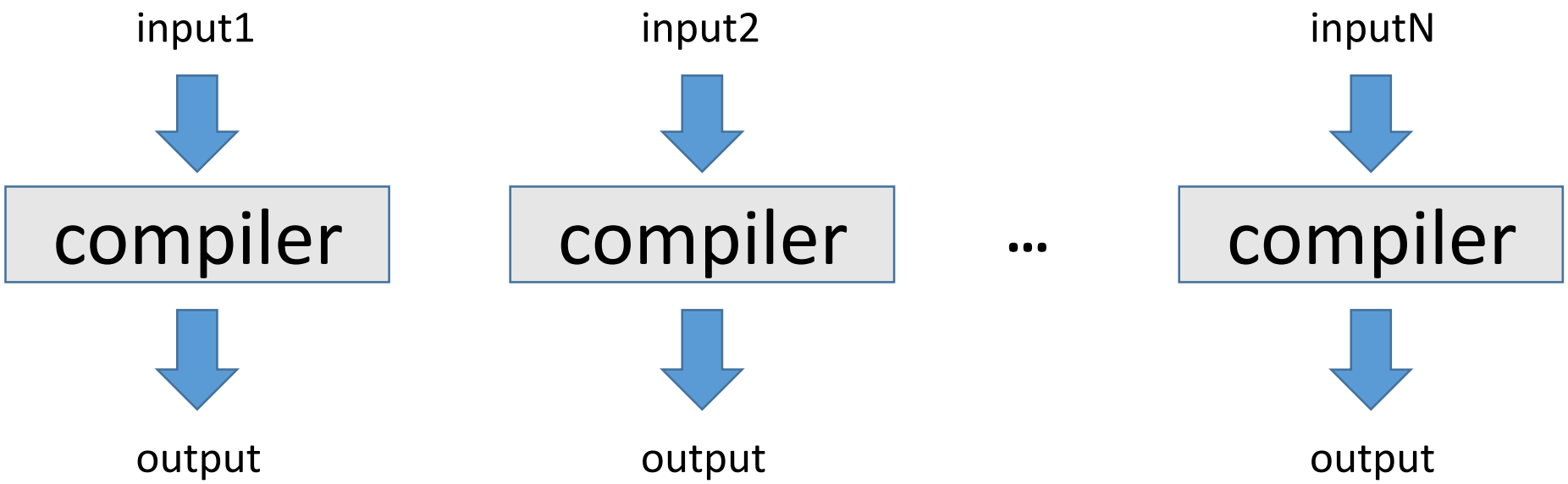
- **Потому что copy elision делается на frontend компилятора**

Copy Elision Implementation

Основы работы компилятора



Концептуальное разделение компилятора



Концептуальное разделение компилятора

N входных языков, M целевых платформ → N*M реализаций

N входных языков, M целевых платформ → N + M реализаций

NRVO

```
Widget bar() {
    Widget w;
    return w;
}
```

NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

```
void bar(Widget *ret) {

        Widget::Widget(ret);
}
```

NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

```
void bar(Widget *ret) {
        // new (ret) Widget();
        Widget::Widget(ret);
}
```

NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();
}
```

```
void bar(Widget *ret) {
        // new (ret) Widget();
        Widget::Widget(ret);
}

void foo() {
        Widget* w = alloca(sizeof(Widget));
        bar(w);
}
```

NRVO

```
define void @bar
```

NRVO

```
define void @bar(%struct.Widget* sret)
```

NRVO

```
define void @bar(%struct.Widget* sret) {
    call void @Widget::Widget(%0)
}
```

NRVO

```
define void @bar(%struct.Widget* sret) {
    call void @Widget::Widget(%0)
}

define void @foo() {
    %w = alloca %struct.Widget
    call void @bar(sret %w)
}
```

without NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

without NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

```
// Widget bar()
void bar(Widget *ret) {
        Widget* w = alloca(sizeof(Widget));

}
```

without NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

```
// Widget bar()
void bar(Widget *ret) {
        Widget* w = alloca(sizeof(Widget));
        Widget::Widget(w);

}
```

without NRVO

```
Widget bar() {
    Widget w;
    return w;
}

void foo() {
    Widget w = bar();

}
```

```
// Widget bar()
void bar(Widget *ret) {
        Widget* w = alloca(sizeof(Widget));
        Widget::Widget(w);
        Widget::Widget(ret, w);
}
```

without NRVO

```
define void @bar(%struct.Widget* sret) {
  %w = alloca %struct.Widget


}
```

without NRVO

```
define void @bar(%struct.Widget* sret) {
  %w = alloca %struct.Widget
  call void @Widget::Widget(%w)

}
```

without NRVO

```
define void @bar(%struct.Widget* sret) {
  %w = alloca %struct.Widget
  call void @Widget::Widget(%w)
  call void @Widget::Widget(%0, %w)
}
```

NRVO

```
define void @bar(%struct.Widget* sret) {
    call void @Widget::Widget(%0)
}
```

```
define void @foo() {
    %w = alloca %struct.Widget
    call void @bar(sret %w)
}
```

without NRVO

```
define void @bar(%struct.Widget* sret) {
    %w = alloca %struct.Widget
    call void @Widget::Widget(%w)
    call void @Widget::Widget(%0, %w)
}
```

```
define void @foo() {
    %w = alloca %struct.Widget
    call void @bar(sret %w)
}
```

NRVO and function parameters

```
Widget bar(Widget w) {
    return w;
}
```

NRVO and function parameters

```
Widget bar(Widget w) {
    return w;
}

void foo() {
    Widget w1;
    Widget w2 = bar(w1);

}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget




}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget
  %w2 = alloca %struct.Widget




}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget
  %w2 = alloca %struct.Widget
  %tmp_arg = alloca %struct.Widget




}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget
  %w2 = alloca %struct.Widget
  %tmp_arg = alloca %struct.Widget
  call void @Widget::Widget(%w1)



}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget
  %w2 = alloca %struct.Widget
  %tmp_arg = alloca %struct.Widget
  call void @Widget::Widget(%w1)
  call void @Widget::Widget(%tmp_arg, %w1)

}
```

NRVO and function parameters

```
define void @bar(%struct.Widget* sret, %struct.Widget*) {
  call void @Widget::Widget(%0, %1)
}
define void @foo() {
  %w1 = alloca %struct.Widget
  %w2 = alloca %struct.Widget
  %tmp_arg = alloca %struct.Widget
  call void @Widget::Widget(%w1)
  call void @Widget::Widget(%tmp_arg, %w1)
  call void @bar(sret %w2, %tmp_arg)
}
```

Ultimate Copy Elision

Цель

Цель

- Применять copy elision в любых возможных контекстах, минуя ограничения, налагаемые текущими требованиями

Зачем?

Зачем?

- Увеличить производительность за счет уменьшения копирований/перемещений

Зачем?

- Увеличить производительность за счет уменьшения копирований/перемещений
- Уменьшить размер исполняемых файлов за счет удаления лишних инструкций

Зачем?

- Увеличить производительность за счет уменьшения копирований/перемещений
- Уменьшить размер исполняемых файлов за счет удаления лишних инструкций
- Упростить разработку, сделав правила возврата объектов более простыми

Что сломаем:

Что сломаем:

- Пользовательский код, полагающийся на побочные эффекты копирующих/перемещающих конструкторов

Что сломаем

- Пользовательский код, полагающийся на побочные эффекты копирующих/перемещающих конструкторов:
    - Является не портируемым

Что сломаем

- Пользовательский код, полагающийся на побочные эффекты копирующих/перемещающих конструкторов:
    - Является не портируемым:
    - в C++ всё опирается на то, что после выполнения “T u = v” объекты “u” и “v” должны быть эквивалентны

Предложение

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором
  - и

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором
  - и
    - либо время жизни Source заканчивается после copy/move конструктора для Destination;

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором
  - и
    - либо время жизни Source заканчивается после copy/move конструктора для Destination;
    - либо **конструктор Source не использует глобальные переменные** или не имеет параметров, через которые возвращает значение; при этом Source не используется между конструктором и copy/move конструктором

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором
  - и
    - либо время жизни Source заканчивается после copy/move конструктора для Destination;
    - либо конструктор Source не использует глобальные переменные или не имеет параметров, через которые возвращает значение; при этом **Source не используется между конструктором и copy/move конструктором**

Предложение

- Для автоматических объектов source и destination можно применить copy elision, если:
  - Source не используется между copy/move конструктором и деструктором
  - и
    - либо время жизни Source заканчивается после copy/move конструктора для Destination;
    - либо конструктор Source не использует глобальные переменные или не имеет параметров, через которые возвращает значение; при этом Source не используется между конструктором и copy/move конструктором
- После copy elision, время жизни Source должно быть продлено до времени жизни Destination

Пример

Пример

```
std::string example() {
    std::string v{"some string that is too big for SSO"};
    return std::move(v); // elide redundant copy
}
```

Ещё пример

```
void takes_by_copy(std::string v) noexcept { /* ... */ }

void example() {
    std::string v{"some string that is too big for SSO"};
    takes_by_copy(v); // elide redundant copy
}
```

Ещё один пример

```
void takes_by_creference(const std::string& v) noexcept {
    std::string copy{v}; // elide redundant copy
    /* ... */
}

void example() {
    return takes_by_creference(
        "some string that is too big for SSO");
}
```

Ещё один пример

```
void takes_by_creference(const std::string& v) noexcept {
    std::string copy{v}; // elide redundant copy
    /* ... */
}

void example() {
    return takes_by_creference(
        "some string that is too big for SSO");
}
```

Ещё один пример

```
void example() {
    std::string copy{
        "some string that is too big for SSO"};
    /* ... */
}
```

Как это сделать, если сигнатуру менять нельзя?

Как это сделать, если сигнатуру менять нельзя?

- **После оптимизации inline** создается большой контекст для применения других оптимизаций

Ultimate Copy Elision Implementation

Концептуальное разделение компилятора

- Оптимизация не должна зависеть от входного языка

Преодоление разделения

- реализовать проход Copy Elision, который:
  - должен быть универсален и не зависеть от нюансов входного языка

Преодоление разделения

- реализовать проход Copy Elision, который:
  - должен быть универсален и не зависеть от нюансов входного языка
  - не должен знать о специфике входного языка, должен работать только с сущностями IR

Redundant Copy Elision Pass (RCE)

- Общая идея – замена source на destination, если они могут быть взаимозаменяемы, т.е. один является алиасом для другого

RCE

- Операция инициализации
- Операция копирования (source – откуда, destination – куда)
- Операция деинициализации

Алгоритм RCE

- Для каждой операции копирования:
  - **с момента начала времени жизни source до операции копирования не утекает адрес source**

Алгоритм RCE

```
struct S {/*...*/};

void bar(S &s);

void foo() {
        S s1;
        bar(s1);
        S s2(s1);
        // ...
}
```

Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - **после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination**

Алгоритм RCE

```
struct S {/*...*/};

void foo() {
        S s1;
        S s2(s1);
        s1.field = /*...*/;
        /*...*/ = s2.field;
        // ...
}
```

Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - **если время жизни destination больше времени жизни source, продлеваем время жизни source до destination**

Алгоритм RCE

```
struct S {/*...*/};

S bar(S s) { return s; }

void foo() {
        S s1;
        S s2 = bar(s1);
        // ...
}
```

Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - если время жизни destination больше времени жизни source, продлеваем время жизни source до destination
  - **если время жизни destination меньше времени жизни source, то оставляем время жизни как у source, если нет операций чтения/записи между окончаниями времен жизни destination и source**

Алгоритм RCE

```
struct locked_mutex {
   mutex m{};
   lock_guard<mutex> lock(m);
};

void foo() {
        unique_ptr<locked_mutex> l1 = make_unique<locked_mutex>();
        {
            unique_ptr<locked_mutex> l2(move(l1));
        }
        bar();
        // ...
}
```

Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - если время жизни destination больше времени жизни source, продлеваем время жизни source до destination
  - если время жизни destination меньше времени жизни source, то оставляем время жизни как у source, если нет операций чтения/записи между окончаниями времен жизни destination и source
  - **удаляем операцию копирования**

Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - если время жизни destination больше времени жизни source, продлеваем время жизни source до destination
  - если время жизни destination меньше времени жизни source, то оставляем время жизни как у source, если нет операций чтения/записи между окончаниями времен жизни destination и source
  - удаляем операцию копирования
  - **заменяем source на destination, удаляя сответствующие операции деинициализации**

Алгоритм RCE

```
struct S {/*...*/};

S bar(S s) { return s; }
void baz(S &s);

void foo() {
        S s = bar(S());
        baz(s);
}
```

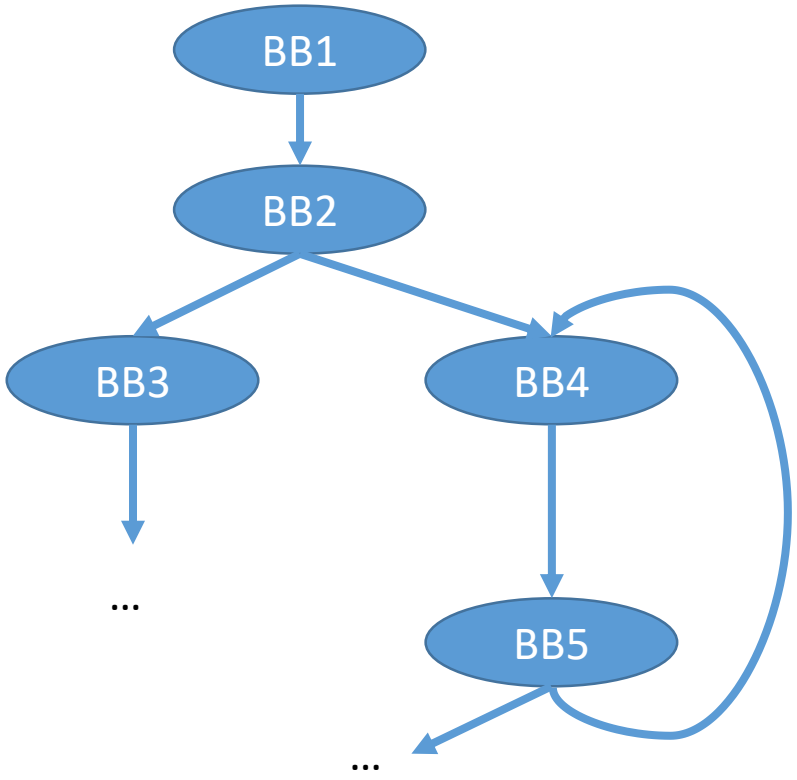
Алгоритм RCE

```
struct S {/*...*/};

void baz(S &s);

void foo() {
        S s;
        baz(s);
}
```

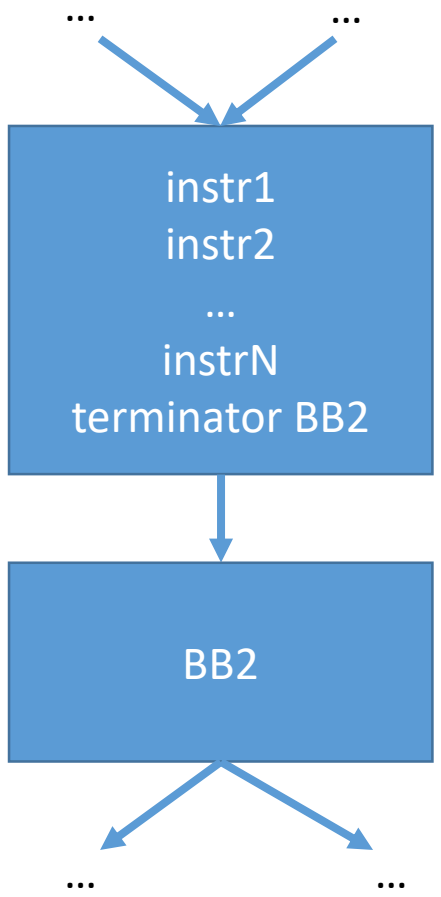
Введение в LLVM IR: основные концепции

- Модуль
  - Функции
    - Базовые блоки
      - Инструкции/Интринсики
        - Метаданные

Граф потока управления (Control Flow Graph – CFG)



Базовый блок (Basic Block)



Интринсики

- Специальные функции, расширяющие LLVM IR:
  - Обладают четко определенной семантикой, о которой известно компилятору

```
declare void @llvm.memcpy.p0i8.p0i8.i32(i8* <dest>, i8* <src>,
                                        i32 <len>, i1 <isvolatile>)

call void @llvm.memcpy.p0i8.p0i8.i32(...)
```

Метаданные

- Специальные данные, которые могут быть привязаны к инструкциям/интринсикам
- Могут быть необходимы оптимизациям (middle-end) и кодогенератору (backend)

```
header0:
; ...
br i1 %cmp, label %t1, label %t2, !irr_loop !0

; ...
!0 = !{"loop_header_weight", i64 100}
```

Операции копирования, инициализации, деинициализации

- На фронтенде поместить метаданные:
  - **copy_init** на copy/move конструкторы
  - **init** на все остальные конструкторы
  - **cleanup** на деструкторы

Inline: пропагация метаданных

```
llvm::InlineFunction(CallBase &CB, /*params*/) {
    // ...
    // Propagate init, copy_init and cleanup metadata.
    PropagateGenMetadata(CB, /*new ins*/, LLVMContext::MD_init);
    PropagateGenMetadata(CB, /*new ins*/, LLVMContext::MD_copy_init);
    PropagateGenMetadata(CB, /*new ins*/, LLVMContext::MD_cleanup);
    // ...
}
```

Операция копирования/деинициализации

```
foo() {
    // ...
    Widget w;
    // ...
    ~Widget(&w); !cleanup
}
```

```
~Widget() {
    std::cout << “dtor\n”;
}
```

Операция копирования/деинициализации

```
foo() {
    // ...
    Widget w;
    // ...
    std::cout << “dtor\n”; !cleanup
}
```

Операция копирования/деинициализации

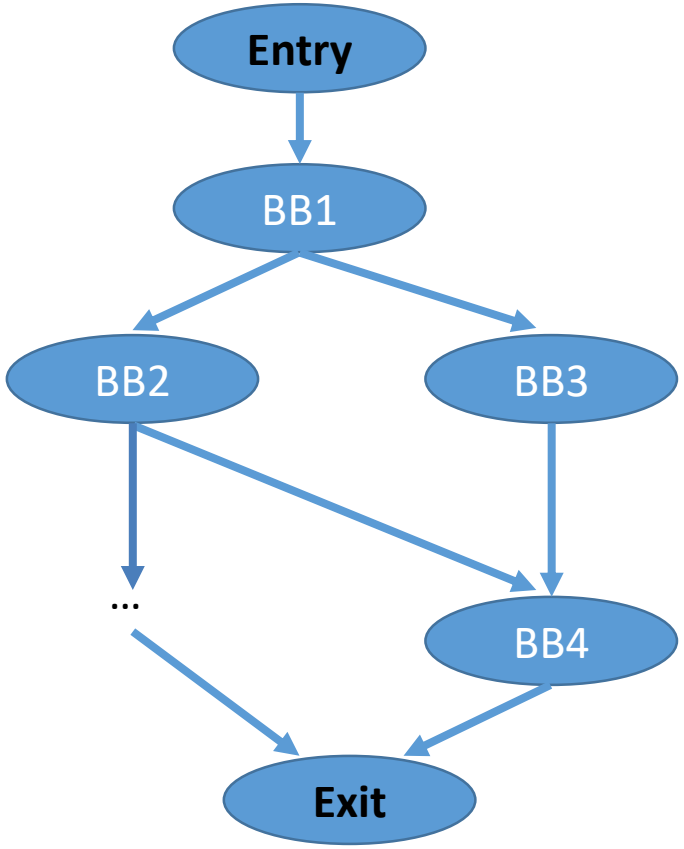
- **declare void @llvm**.copy.start(**type*** <dest>, **type*** <src>)
- **declare void @llvm**.copy.end(**type*** <dest>, **type*** <src>)

- **declare void @llvm**.cleanup.start(**type*** <obj>)
- **declare void @llvm**.cleanup.end(**type*** <obj>)

Операция копирования/деинициализации

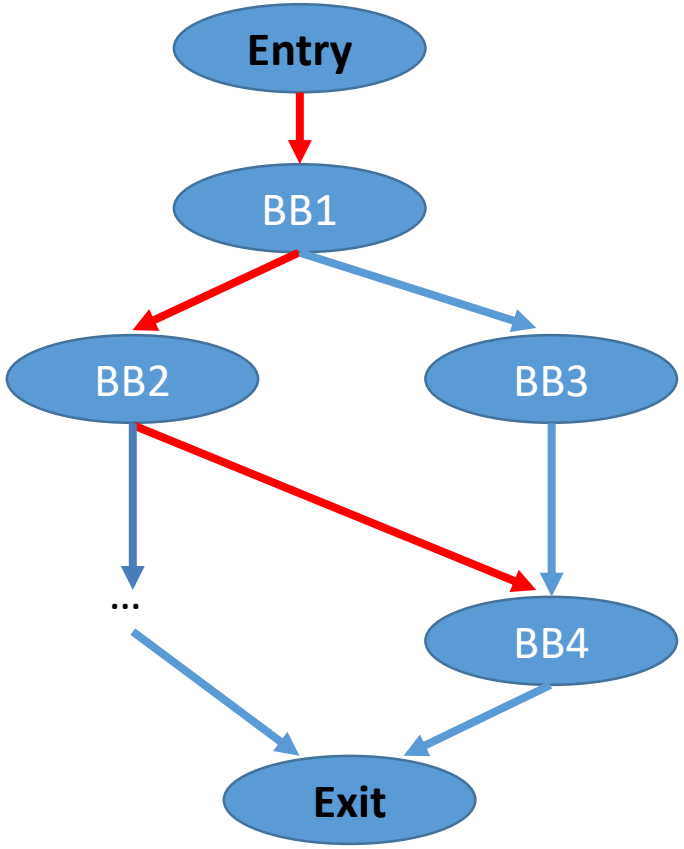
```
declare void @llvm.copy.start(type* <dest>, type* <src>)
; instructions after Inline
declare void @llvm.copy.end(type* <dest>, type* <src>)

declare void @llvm.cleanup.start(type* <obj>)
; instructions after Inline
declare void @llvm.cleanup.end(type* <obj>)
```

Dominators and Post-dominators



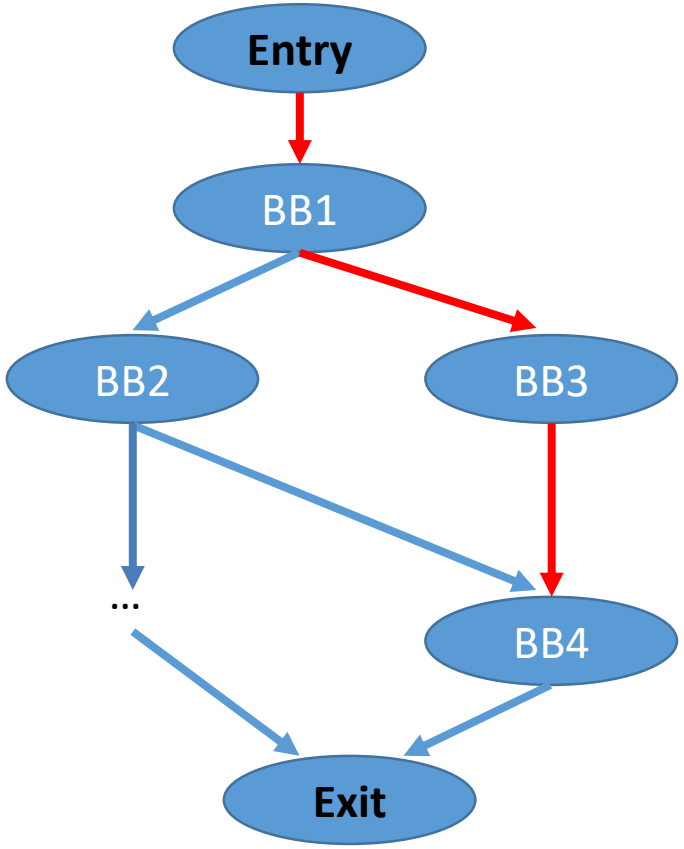
- **BB1** dominates **BB4**

Dominators and Post-dominators



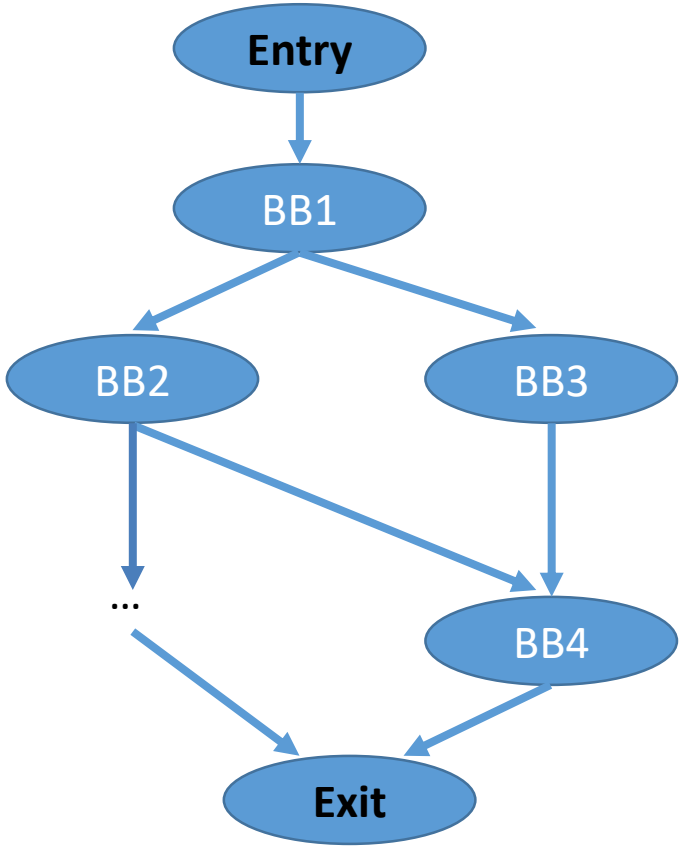
- **BB1** dominates **BB4**

Dominators and Post-dominators



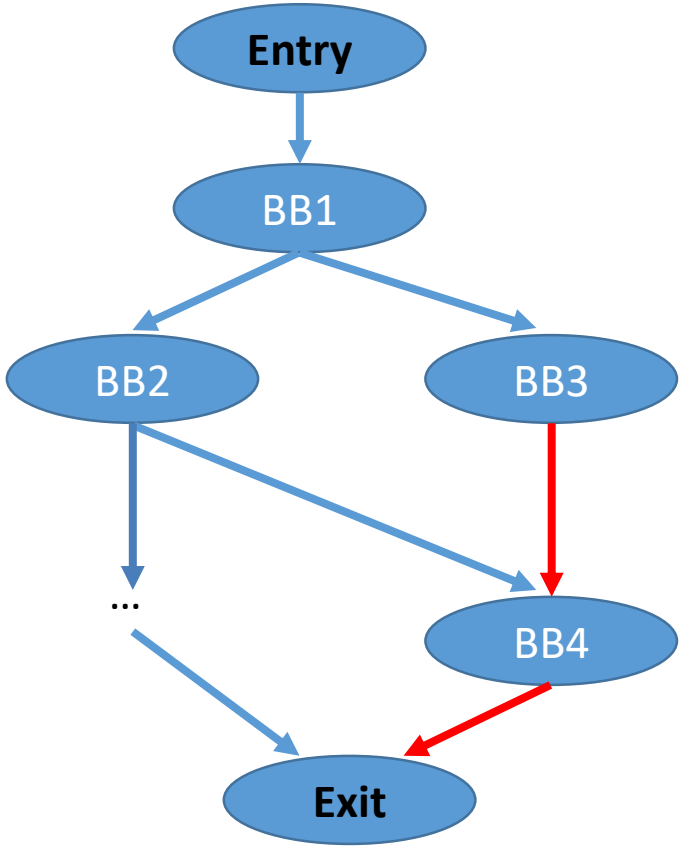
- **BB1** dominates **BB4**

Dominators and Post-dominators



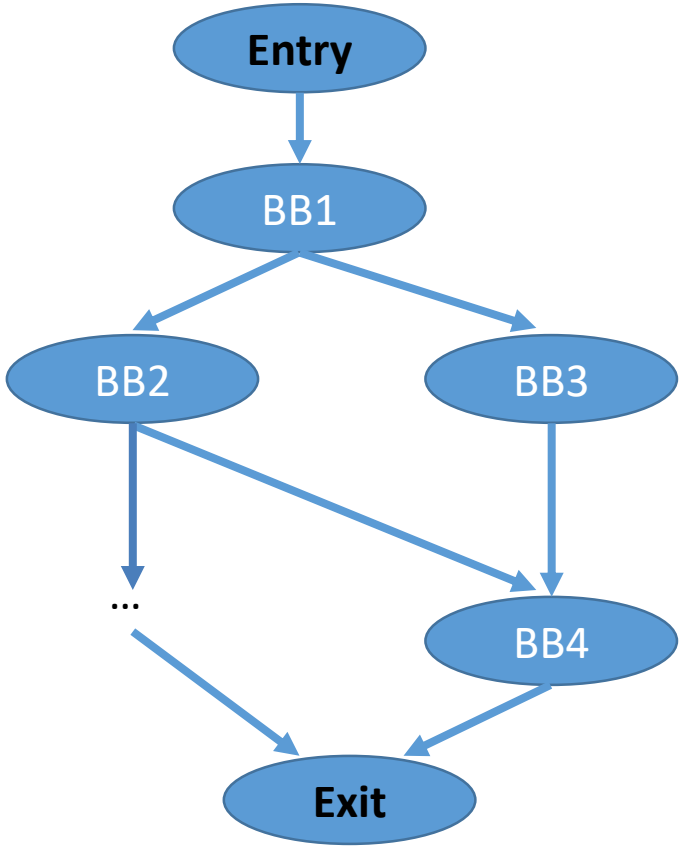
- **BB1** dominates **BB4**
- **BB4** post-dominates **BB3**

Dominators and Post-dominators



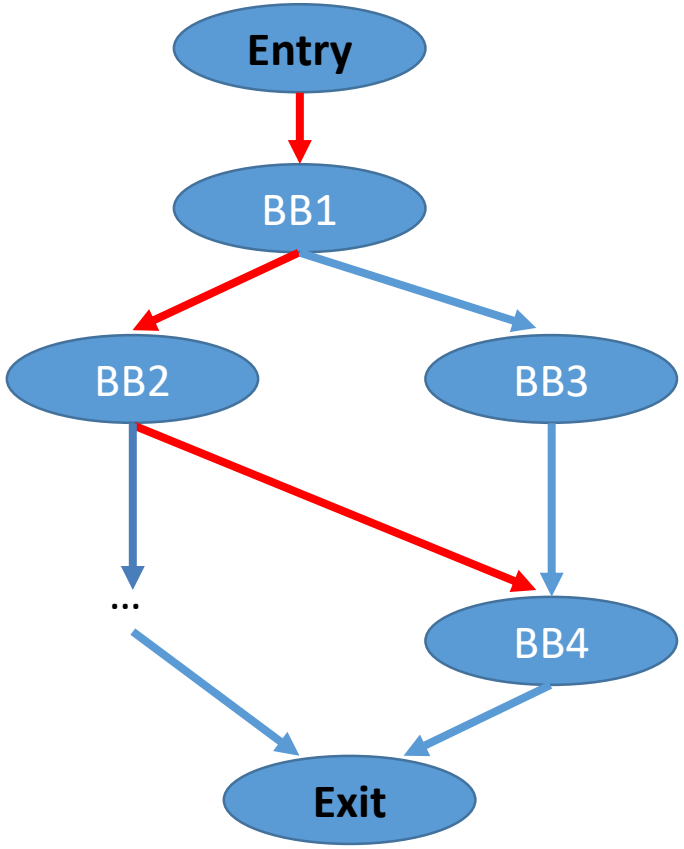
- **BB1** dominates **BB4**
- **BB4** post-dominates **BB3**

Dominators and Post-dominators



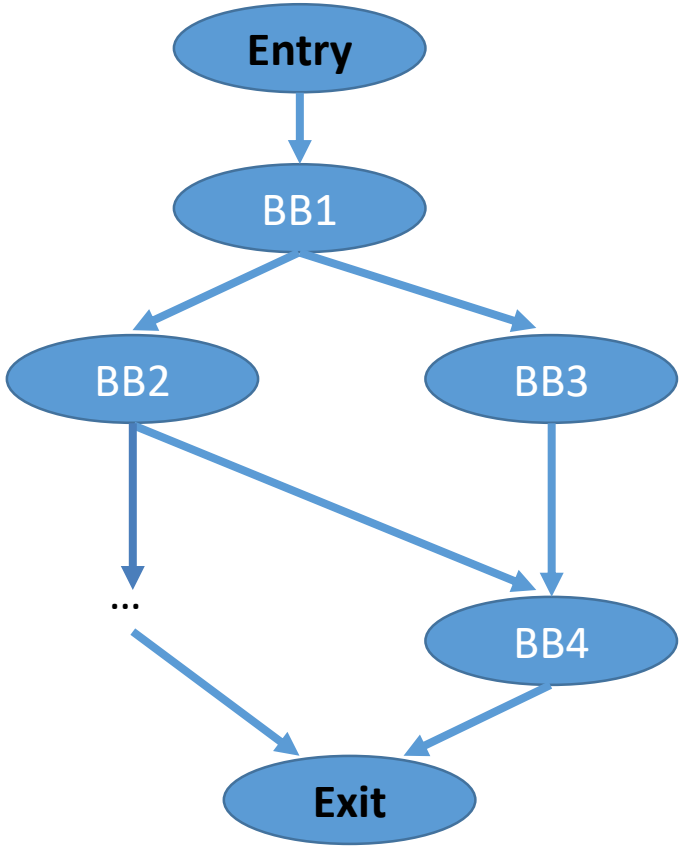
- **BB1** dominates **BB4**
- **BB4** post-dominates **BB3**
  - but **BB3** doesn't dominate **BB4**

Dominators and Post-dominators



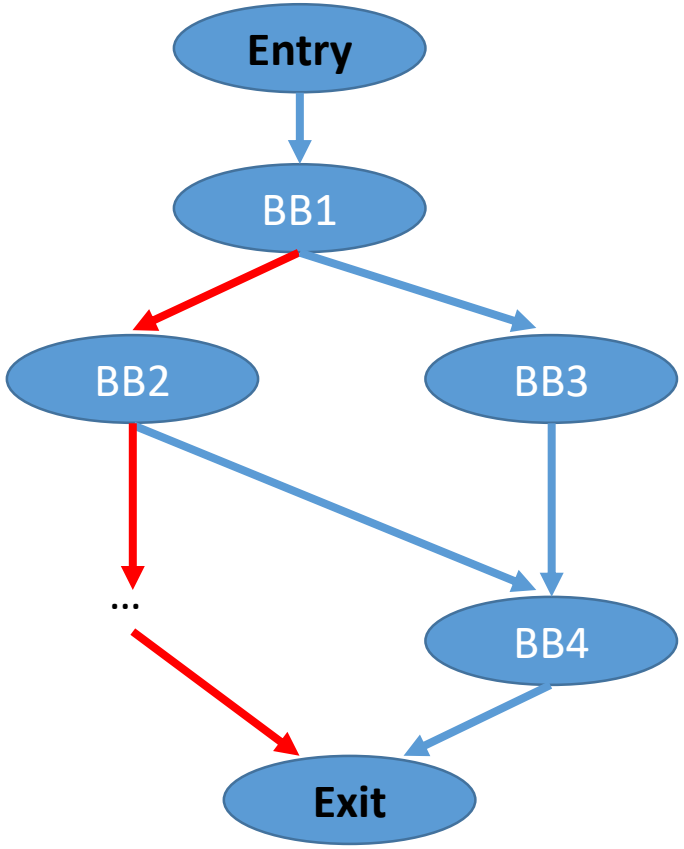
- **BB1** dominates **BB4**
- **BB4** post-dominates **BB3**
  - but **BB3** doesn't dominate **BB4**

Dominators and Post-dominators



- **BB1** dominates **BB4**
  - but **BB4** doesn't post-dominate **BB1**
- **BB4** post-dominates **BB3**
  - but **BB3** doesn't dominate **BB4**

Dominators and Post-dominators



- **BB1** dominates **BB4**
  - but **BB4** doesn't post-dominate **BB1**
- **BB4** post-dominates **BB3**
  - but **BB3** doesn't dominate **BB4**

Принадлежность инструкции к операции копирования

```
if (DT->dominates(CopyStart, Ins) &&
  (DT->dominates(Ins, CopyEnd) || PDT->dominates(CopyEnd, Ins)))
    if (IsCopyInit(*Ins))
        return true;
```

Frontend: кодогенерация

```
if (Context.getLangOpts().UltimateCopyElision &&
    isa<CXXDestructorDecl>(MD))
      type = CXXCallType::Dtor;
```

Frontend: кодогенерация

```
if (Context.getLangOpts().UltimateCopyElision &&
    CtorKind == Ctor_Complete) {
  unsigned IsVolatile = 0;
  bool IsCMCtor = D->isCopyOrMoveConstructor(IsVolatile);
  IsVolatile &= Qualifiers::Volatile;

  type = !IsCMCtor ? CXXCallType::Ctor
                   : IsVolatile ? CXXCallType::None : CXXCallType::CM_Ctor;
}
```

Frontend: кодогенерация

```
CGFunctionInfo *CGFunctionInfo::create(/*params*/) {
   CGFunctionInfo *FI = new(buffer) CGFunctionInfo();
   // ...
   FI->CxxCtor = (type == CXXCallType::Ctor);
   FI->CxxCMCtor = (type == CXXCallType::CM_Ctor);
   FI->CxxDtor = (type == CXXCallType::Dtor);
   // ...
   return FI;
}
```

Frontend: кодогенерация

```
CodeGenFunction::EmitCall(const CGFunctionInfo &CallInfo, /*params*/) {
  // ...
  if (CallInfo.isCxxCMCtor())
    EmitCopyStartOrEnd(IRCallArgs[0], IRCallArgs[1], /*EmitStart*/ true);
  else if (CallInfo.isCxxDtor())
    EmitCleanupStartOrEnd(IRCallArgs[0], /*EmitStart*/ true);
  // ...
```

Frontend: кодогенерация

```
if (!InvokeDest)
    CI = Builder.CreateCall(/*arguments*/);
  else
    CI = Builder.CreateInvoke(/*arguments*/);
// …
```

Frontend: кодогенерация

```
if (CallInfo.isCxxCtor()) {
    AddCxxFuncCallMetadata(CI, llvm::LLVMContext::MD_init);
 } else if (CallInfo.isCxxCMCtor()) {
    AddCxxFuncCallMetadata(CI, llvm::LLVMContext::MD_copy_init);
    EmitCopyStartOrEnd(CI->getOperand(0), CI->getOperand(1), /*EmitStart*/false);
 } else if (CallInfo.isCxxDtor()) {
    AddCxxFuncCallMetadata(CI, llvm::LLVMContext::MD_cleanup);
    EmitCleanupStartOrEnd(CI->getOperand(0), /*EmitStart*/ false);
 }
 // ...
}
```

Подводные камни

- Inline

```
D::~D() {
  S s1;
  S s2(s1); !copy.init
  foo(&s1);
  bar(&s1);
  // ...
  ~S(&s1); !cleanup
}
```

И source и destination могут иметь признак copy_init/cleanup после Inline

Подводные камни

- Inline

```
// ...
S s1; !cleanup
S s2(s1); !copy.init, !cleanup
foo(&s1); !cleanup
bar(&s1); !cleanup
// ...
~S(&s1); !cleanup
```

И source и destination могут иметь признак copy_init/cleanup после Inline

Inline: поколения метаданных

- **Все метаданные имеют первое поколение**

Inline: поколения метаданных

- Все метаданные имеют первое поколение
- **После инлайна поколение инкрементируется и пропагируется на проинлайненные инструкции**

Inline: поколения метаданных

```
void PropagateGenMetadata(CallBase &CB, /*new ins*/, unsigned MType) {
  // ...
  auto NextCBGeneration = /*CB->getMetadataGeneration() + 1*/
  if (auto *NIM = NI->getMetadata(MType)) {
      auto NIMGen = /*NIM->getMetadataGeneration()*/;
      auto NextNIMGen = /*NIMGen + NextGenerationVal*/;

      NI->setMetadata(MType, MDNode::get(NextNIMGen));
  } else {
      NI->setMetadata(MType, MDNode::get(NextCBGeneration));
  }
  // ...
}
```

Inline: поколения метаданных

- Все метаданные имеют первое поколение
- После инлайна поколение инкрементируется и пропагируется на проинлайненные инструкции
- **Если поколение source или destination меньше поколения проверяемой инструкции ➔ эта инструкция является init/copy_init/cleanup для данного source/destination**

Подводные камни

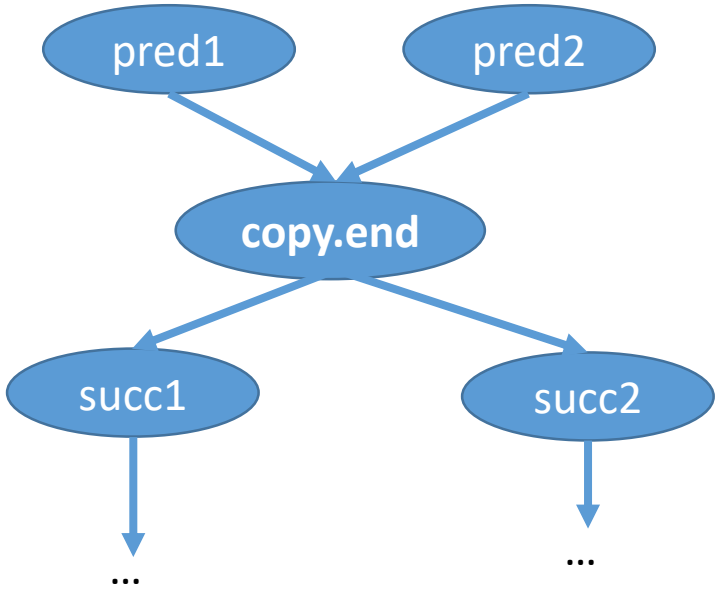
- Inline

```
// ...
S s1; !cleanup:2
S s2(s1); !copy.init:1, !cleanup:2
foo(&s1); !cleanup:2
bar(&s1); !cleanup:2
// ...
~S(&s1); !cleanup:3
```

И source и destination могут иметь признак copy_init/cleanup после Inline

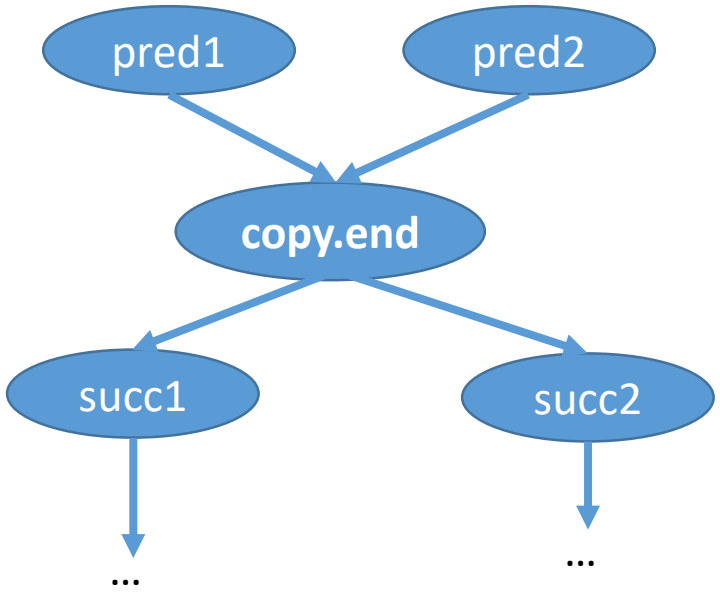
Подводные камни

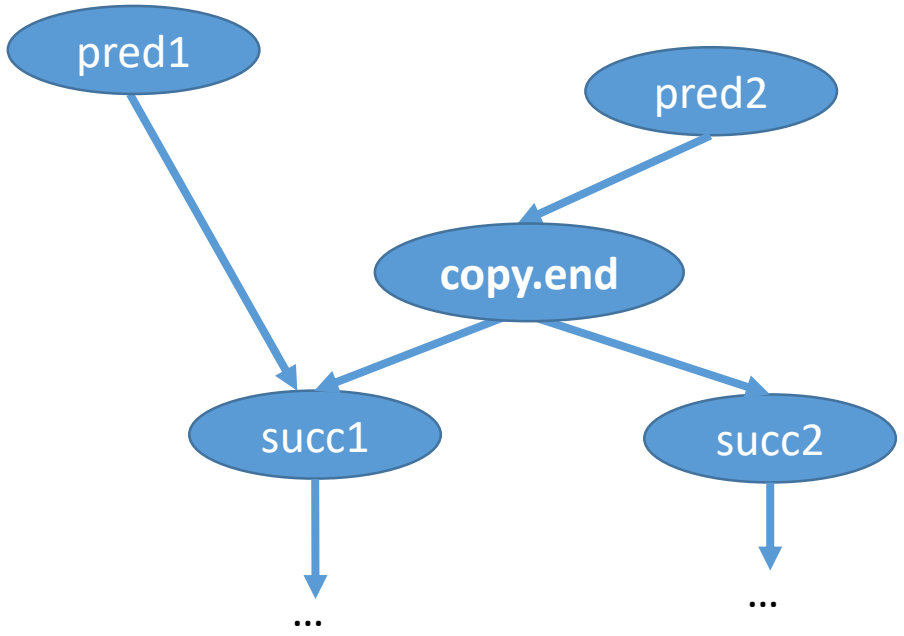
- Jump Threading



Подводные камни

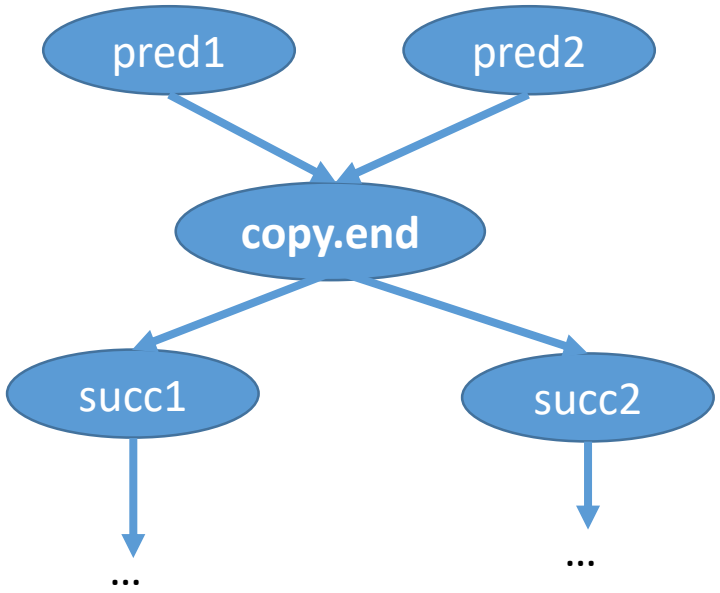
- Jump Threading

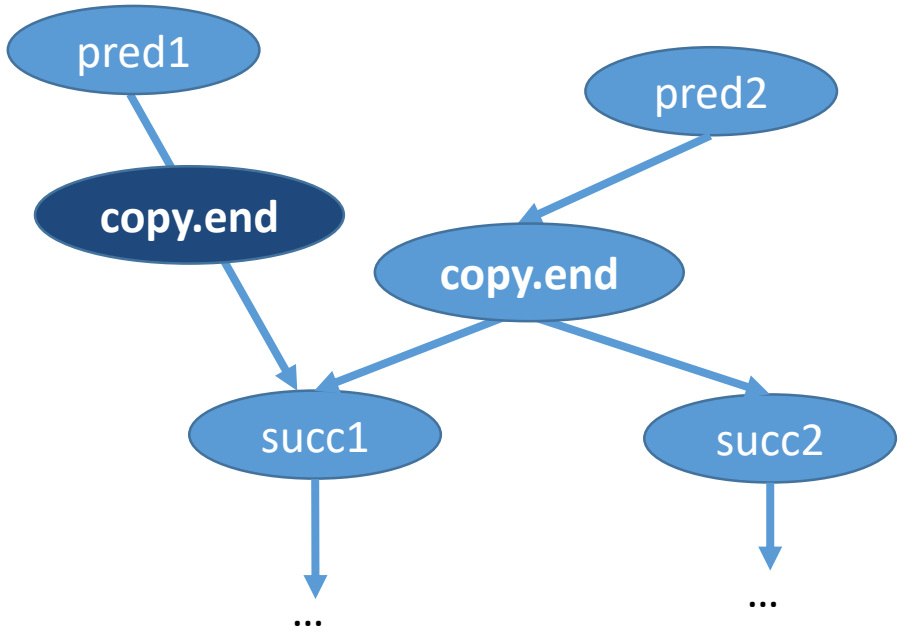




Подводные камни

- Jump Threading





Подводные камни

- CSE (Common Sub-expression Elimination)

```
void foo() {
        // copy.start
        // …
        auto val = a + b;
        // …
        // copy.end

}
```

Подводные камни

- CSE (Common Sub-expression Elimination)

```
void foo() {
        // copy.start
        // …
        auto val = a + b;
        // …
        // copy.end
        if (/*cond*/) {
                /*...*/ = a + b;
        }
}
```

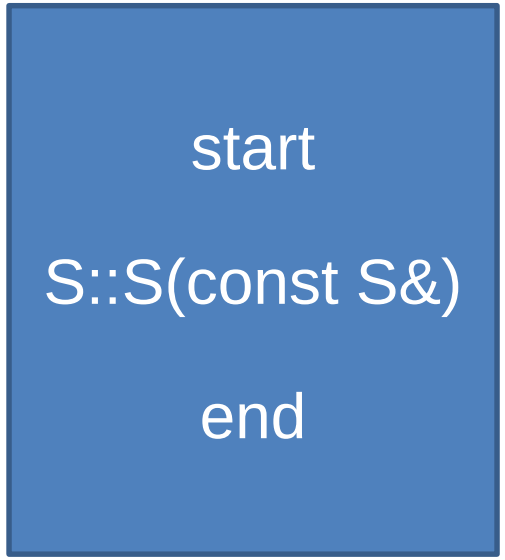
Подводные камни

- CSE (Common Sub-expression Elimination)

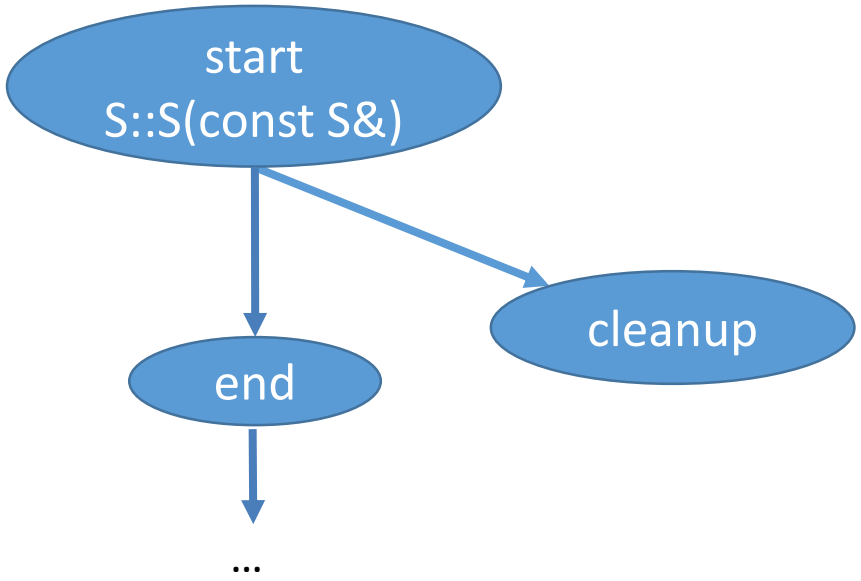
```
void foo() {
        // copy.start
        // …
        auto val = a + b;
        // …
        // copy.end
        if (/*cond*/) {
                /*...*/ = a + b;
        }
}
```

```
void foo() {
        // copy.start
        // …
        auto val = a + b;
        // …
        // copy.end
        if (/*cond*/) {
                /*...*/ = val;
        }
}
```

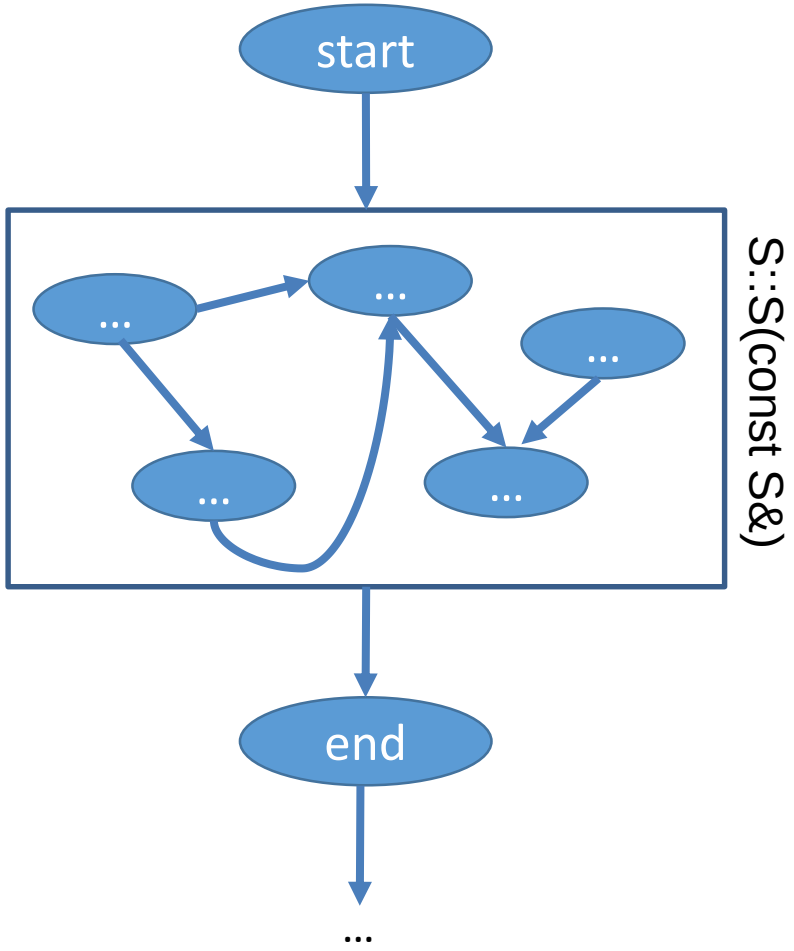
Подводные камни



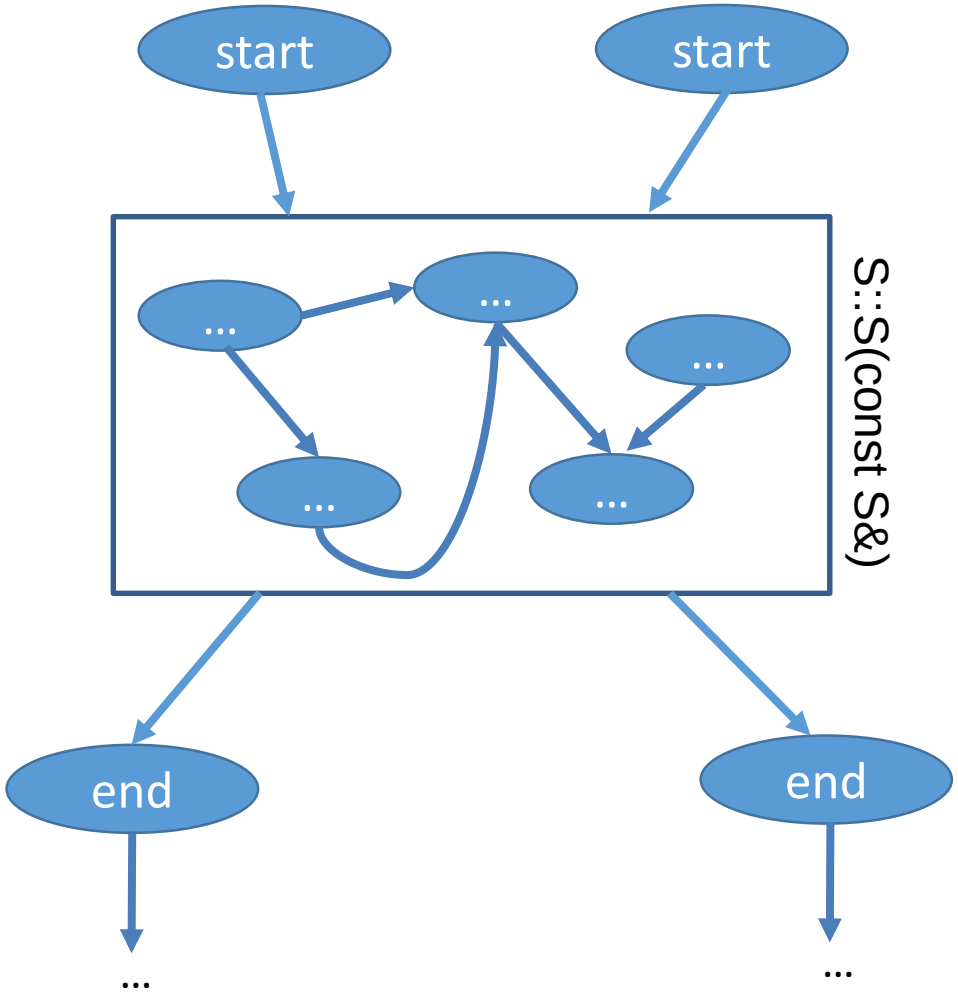
Подводные камни: исключения



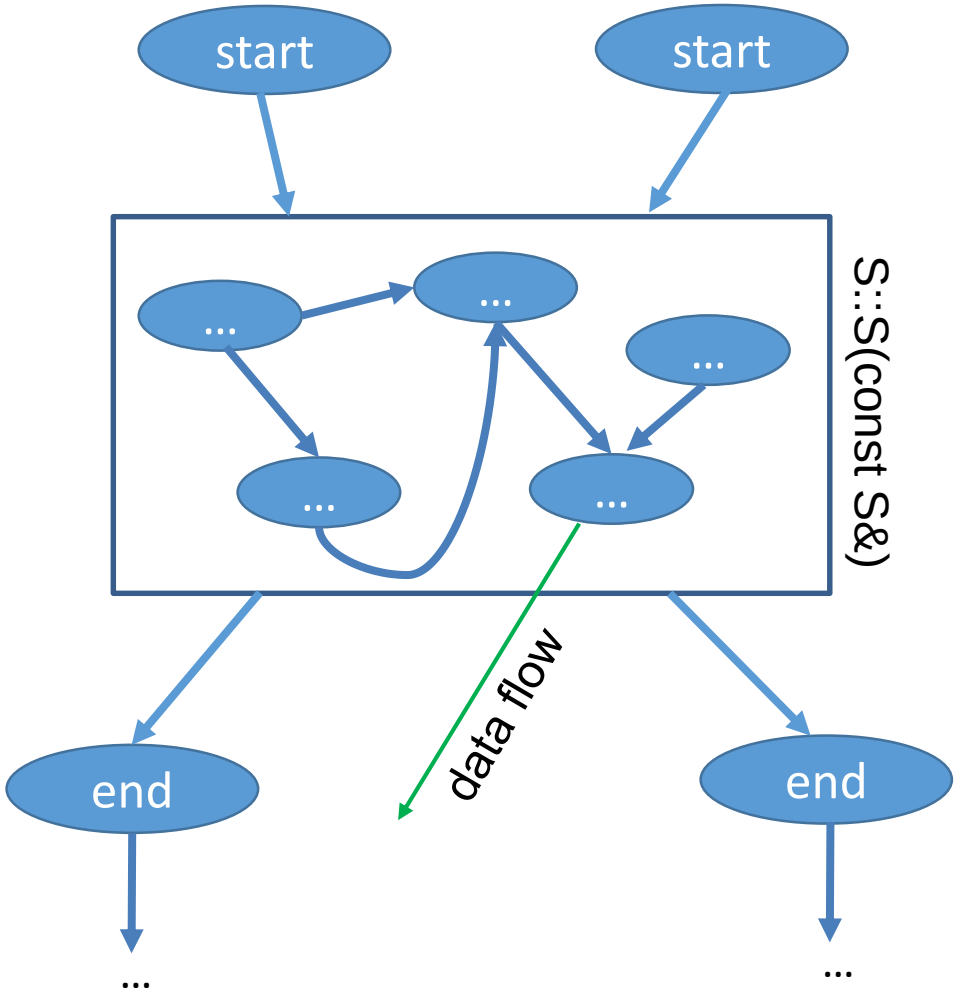
Подводные камни: inline



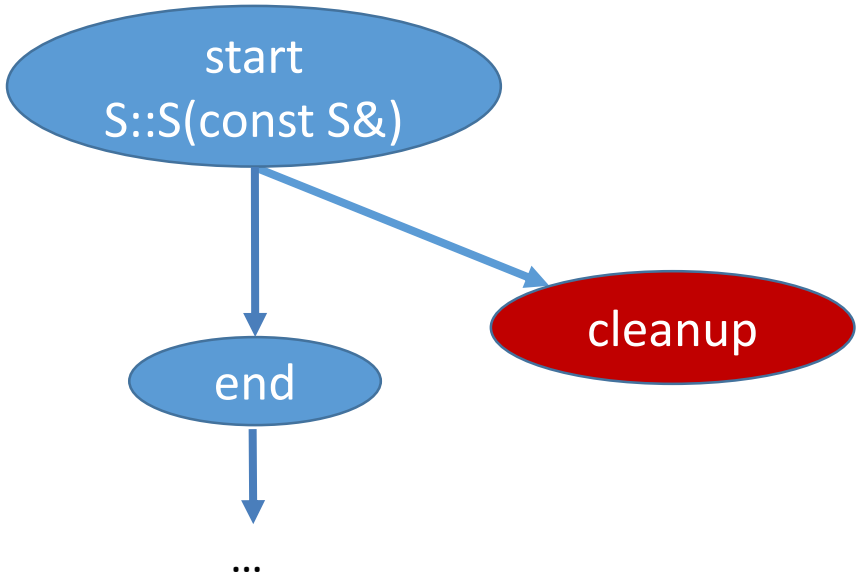
Подводные камни: jump threading



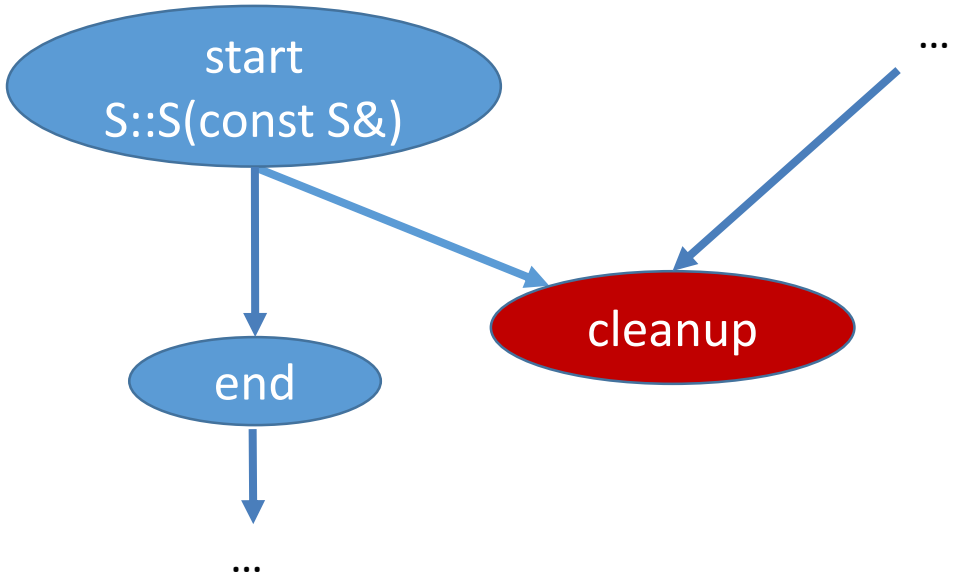
Подводные камни: CSE



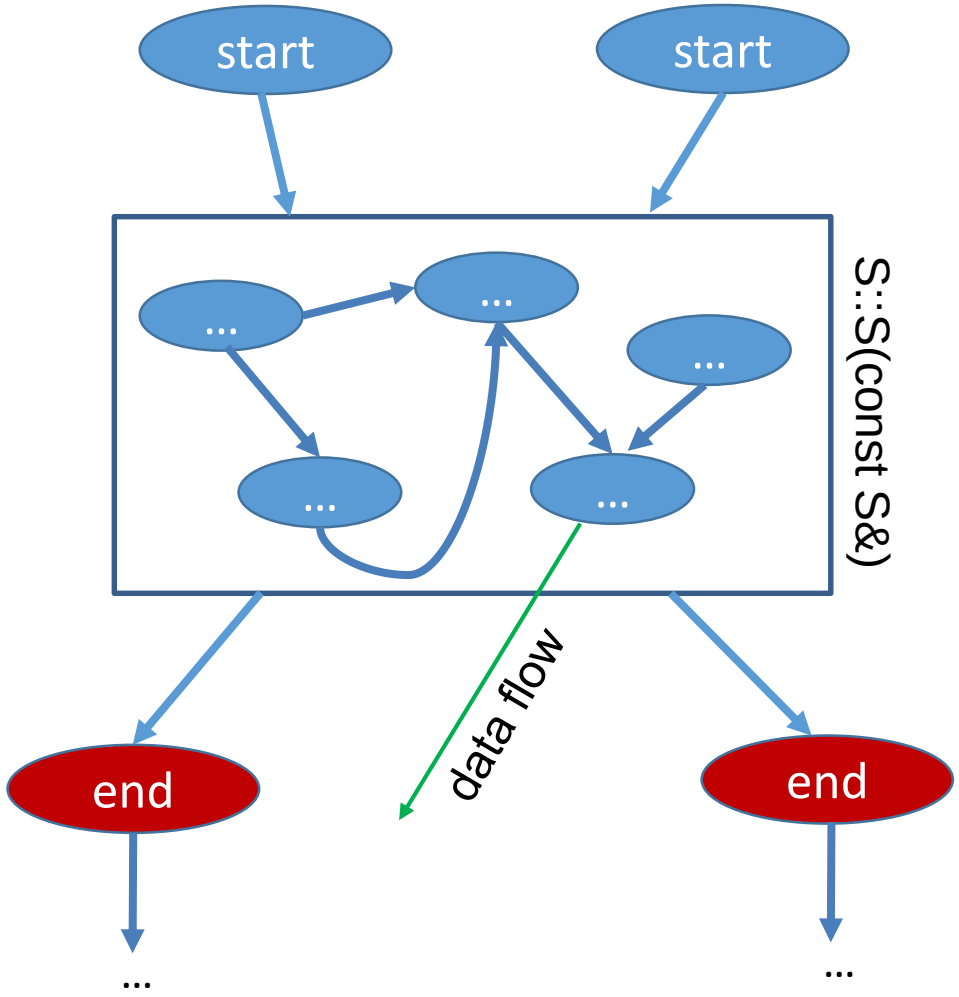
RCE: удаление инструкций



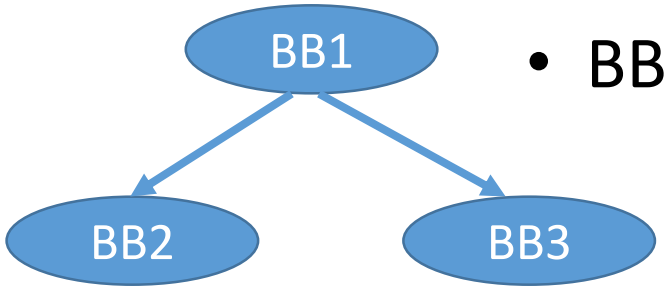
RCE: удаление инструкций



RCE: удаление инструкций

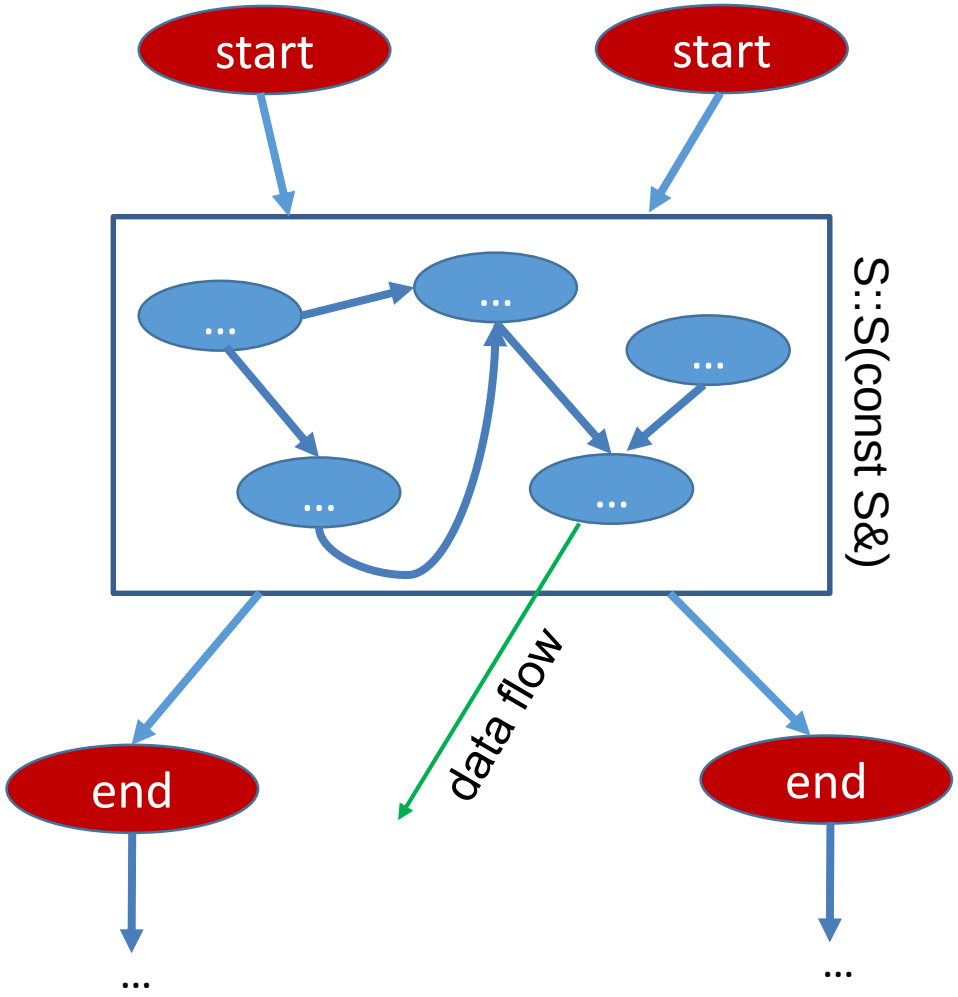


Post order

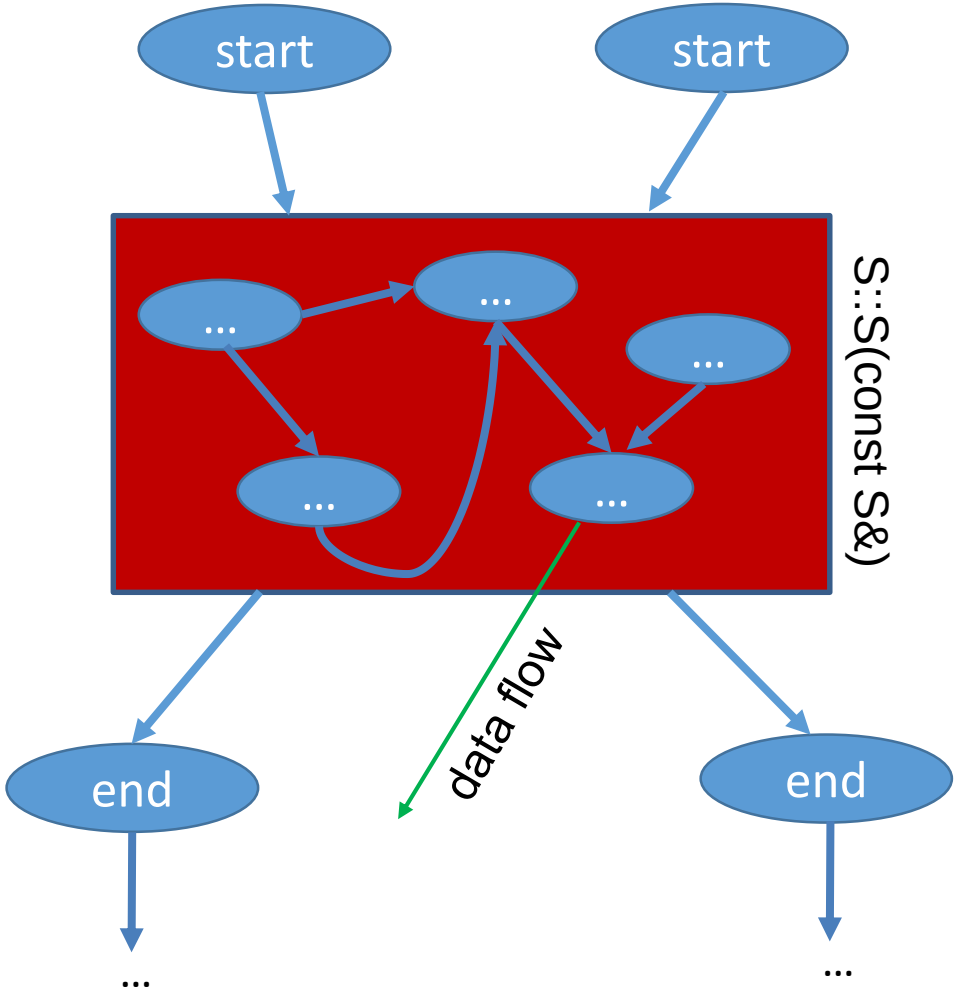


- BB1 обрабатывается только после BB2 и BB3

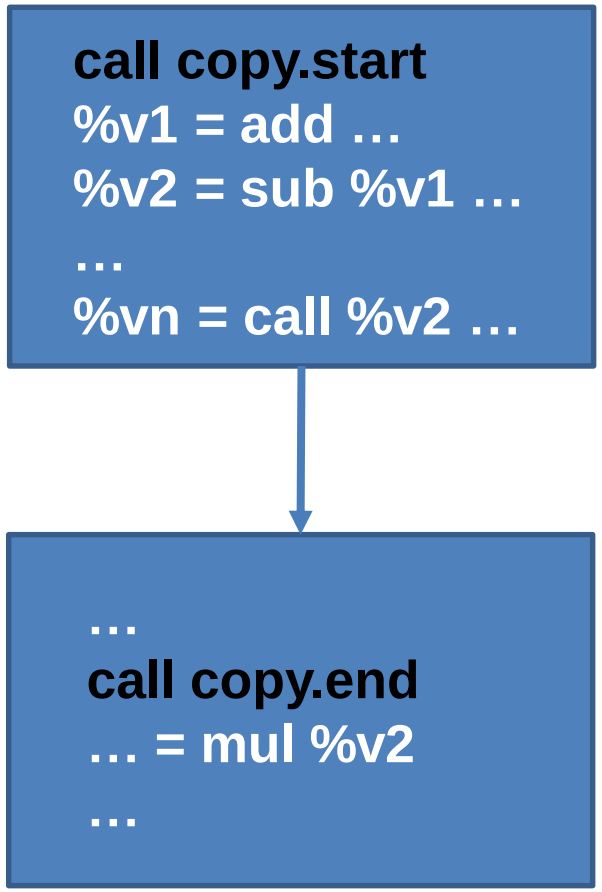
RCE: удаление инструкций



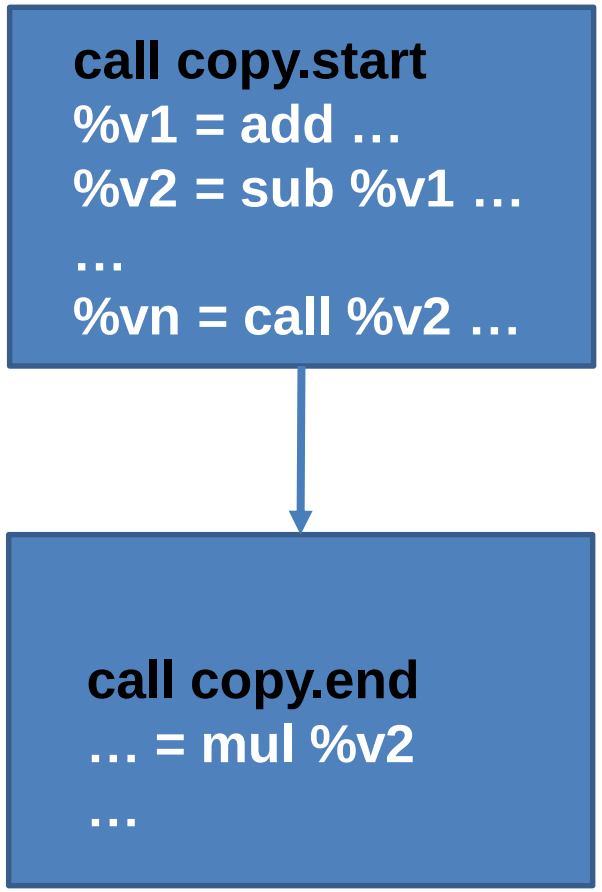
RCE: удаление инструкций



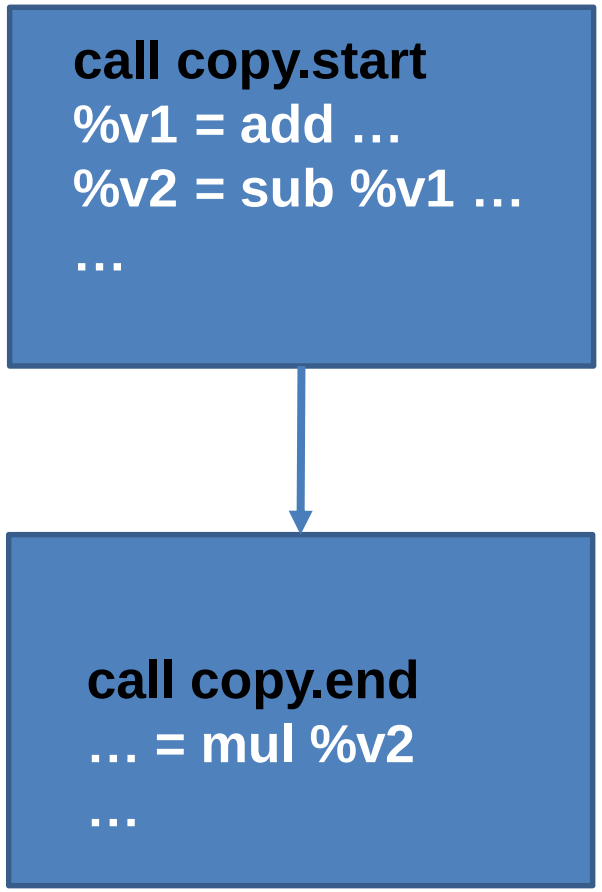
RCE: удаление инструкций



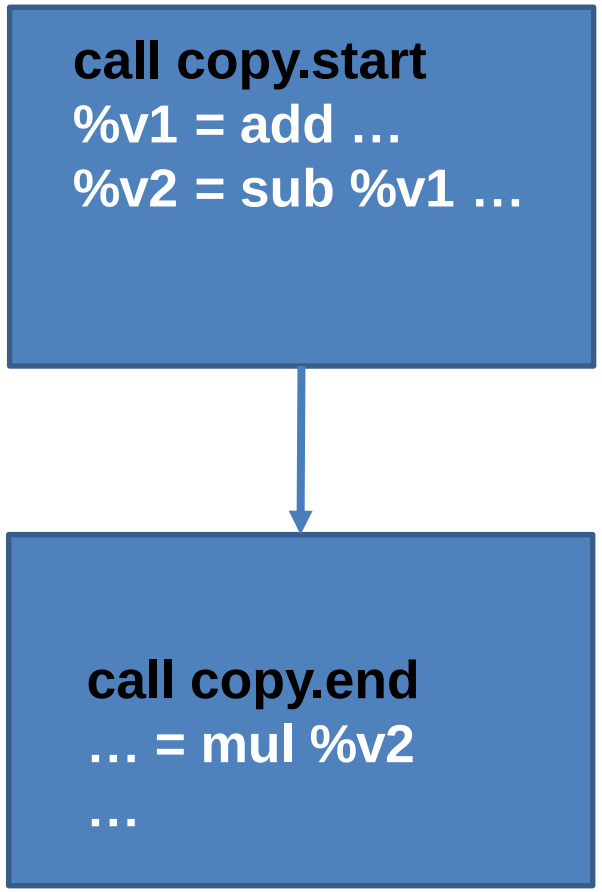
RCE: удаление инструкций



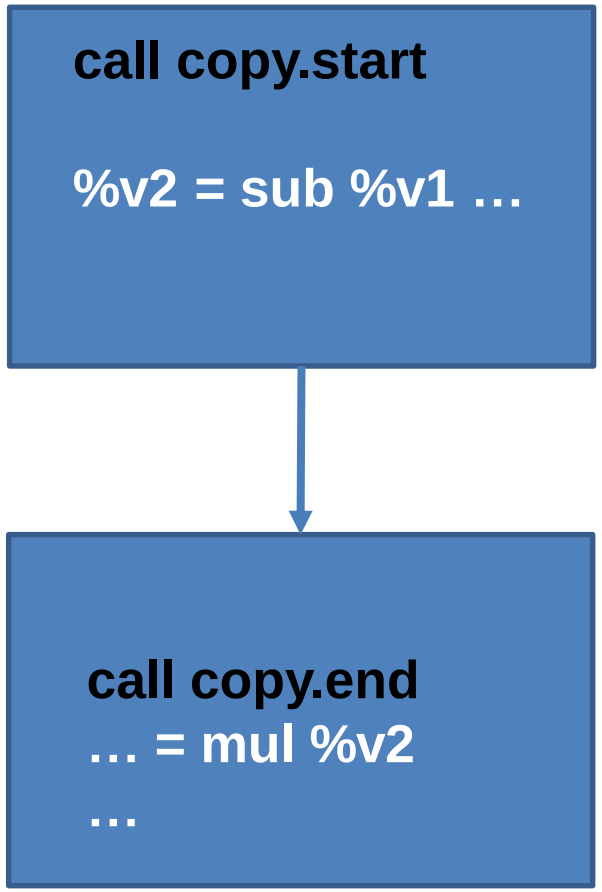
RCE: удаление инструкций



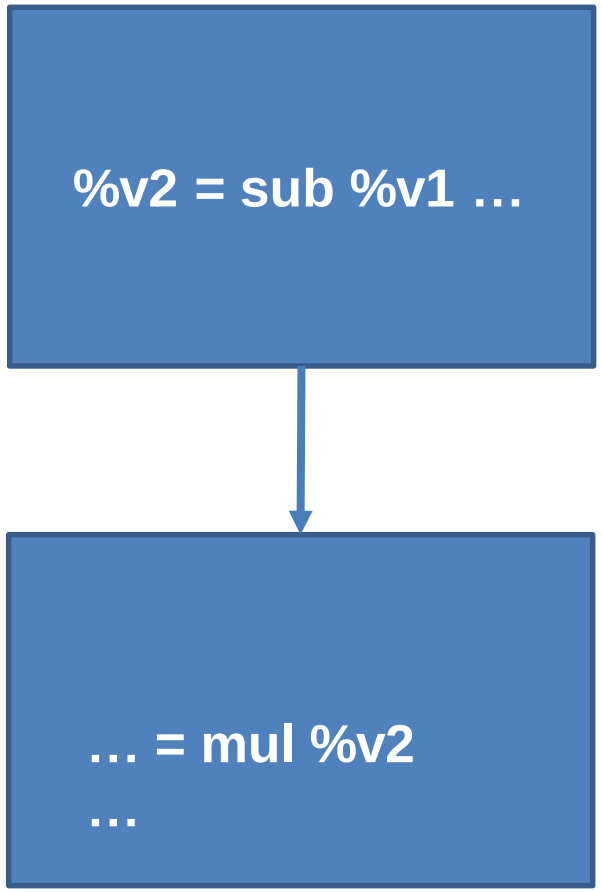
RCE: удаление инструкций



RCE: удаление инструкций



RCE: удаление инструкций



Алгоритм RCE

- Для каждой операции копирования:
  - с момента начала времени жизни source до операции копирования не утекает адрес source
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - если время жизни destination больше времени жизни source, продлеваем время жизни source до destination
  - если время жизни destination меньше времени жизни source, то оставляем время жизни как у source, если нет операций чтения/записи между окончаниями времен жизни destination и source
  - удаляем операцию копирования
  - заменяем source на destination, удаляя сответствующие операции деинициализации

Алгоритм RCE

- Для каждой операции копирования:
  - **с момента начала времени жизни source до операции копирования не утекает адрес source**
  - после операции копирования и до конца времени жизни source или destination не должно быть чтения/записи source или destination
  - если время жизни destination больше времени жизни source, продлеваем время жизни source до destination
  - если время жизни destination меньше времени жизни source, то оставляем время жизни как у source, если нет операций чтения/записи между окончаниями времен жизни destination и source
  - удаляем операцию копирования
  - заменяем source на destination, удаляя сответствующие операции деинициализации

Атрибут nocapture

- Стандартизировать атрибут nocapture

```
struct S {
   S [[nocapture]] (/*params*/);
   // ...
};

void bar [[nocapture]] (S &s);

void foo() {
   S s1;
   bar(s1);
   S s2(s1);
   // ...
}
```

Атрибут nocapture

- Атрибут нужен не только для RCE
- Существует атрибут nocapture для параметров функции

Атрибут nocapture

- Атрибут нужен не только для RCE
- Существует атрибут nocapture для параметров функции
- doesNotCapture:
  - Alias Analysis
  - Instruction Combining
  - Tail Recursion Elimination
  - …

Опция nocapture-ctors

```
auto NoCaptureCtors = [this](const Value &V) {
   return ClNocaptureCtors &&
         (IsInit(V) || IsCopyInit(V));
};

// If NoCaptureCtors == true  optimization can be applied
```

Новая формулировка

- Для автоматических не volatile объектов source и destination можно применить copy elision, если:
  - Source не “утекает” перед копированием/перемещением или передается функциям с *nocapture* атрибутом
  - После операции копирования и до конца времени жизни source или destination не должно быть “наблюдаемых” использований source или destination
  - если время жизни destination меньше времени жизни source, то не должно быть операций чтения/записи между окончаниями времен жизни destination и source
- После copy elision: время жизни объекта – max(lifetime.end(source), lifetime.end(destination))

Статистика Ultimate Copy Elision

Статистика (на clang, lld, compiler-rt)

- Количество кандидатов: ~**27500**

Статистика (на clang, lld, compiler-rt)

- Количество кандидатов: ~**27500**
- Количество удаленных конструкторов: **1370** (~**5**%)

Статистика (на clang, lld, compiler-rt)

- Количество кандидатов: ~**27500**
- Количество удаленных конструкторов: **1370** (~**5**%)
- Причины несрабатывания:
  - ~**20900** – захват перед копированием
  - ~**3900** – использования после копирования

Статистика с опцией nocapture-ctors

- Количество кандидатов: ~**22500**

Статистика с опцией nocapture-ctors

- Количество кандидатов: ~**22500**
- Количество удаленных конструкторов: **2200** (~**10**%)

Статистика с опцией nocapture-ctors

- Количество кандидатов: ~**22500**
- Количество удаленных конструкторов: **2200** (~**10**%)
- Причины несрабатывания:
  - ~**12500** – захват перед копированием
  - ~**6850** – использования после копирования
  - ~**1800** – инструкции между dst.life.end и src.life.end

Планы

- Дать разработчикам опробовать оптимизацию

Планы

- Дать разработчикам опробовать оптимизацию
  - https://github.com/rusyaev-roman/llvm-project/tree/ultimate_copy_elision_v3

Планы

- ~~Дать разработчикам опробовать оптимизацию~~ (**сделано**)
  - https://github.com/rusyaev-roman/llvm-project/tree/ultimate_copy_elision_v3

Планы

- ~~Дать разработчикам опробовать оптимизацию~~ (**сделано**)
  - https://github.com/rusyaev-roman/llvm-project/tree/ultimate_copy_elision_v3
- Заапстримить правки в llvm

Планы

- ~~Дать разработчикам опробовать оптимизацию~~ (**сделано**)
  - https://github.com/rusyaev-roman/llvm-project/tree/ultimate_copy_elision_v3
- Заапстримить правки в llvm
- Обновить wording и презентовать комитету

Планы

- ~~Дать разработчикам опробовать оптимизацию~~ (**сделано**)
  - https://github.com/rusyaev-roman/llvm-project/tree/ultimate_copy_elision_v3
- Заапстримить правки в llvm
- Обновить wording и презентовать комитету
- Включить в C++23/26

Q & A

Антон Полухин
Yandex Taxi
expert developer
antoshkka@gmail.com

Роман Русяев
Samsung R&D
compiler developer
rusyaev.rm@gmail.com

APPENDIX

RCE: удаление инструкций

```
auto AddInsToDeadList =
    [this, &IsInBetween](BasicBlock::reverse_iterator StartPoint,
                         BasicBlock::reverse_iterator StopPoint) {
  for (auto &I : make_range(StartPoint, StopPoint)) {
    DeadInstList.push_back(&I);

  }
};
```

RCE: удаление инструкций

```
auto AddInsToDeadList =
    [this, &IsInBetween](BasicBlock::reverse_iterator StartPoint,
                         BasicBlock::reverse_iterator StopPoint) {
  for (auto &I : make_range(StartPoint, StopPoint)) {
    DeadInstList.push_back(&I);
    if (!IsInBetween(I))
      return false;
  }
  return true;
};
```

RCE: удаление инструкций

```
for (auto *EndI : Ends) {
  auto *EndBB = EndI->getParent();
  auto It = find_if(Starts, [EndBB](const Instruction *I) {
    return I->getParent() == EndBB;
  });



}
```

RCE: удаление инструкций

```
for (auto *EndI : Ends) {
  auto *EndBB = EndI->getParent();
  auto It = find_if(Starts, [EndBB](const Instruction *I) {
    return I->getParent() == EndBB;
  });
  auto StopPoint =
     (It == Starts.end()) ? EndBB->rend() : (*It)->getReverseIterator();


}
```

RCE: удаление инструкций

```
for (auto *EndI : Ends) {
  auto *EndBB = EndI->getParent();
  auto It = find_if(Starts, [EndBB](const Instruction *I) {
    return I->getParent() == EndBB;
  });
  auto StopPoint =
      (It == Starts.end()) ? EndBB->rend() : (*It)->getReverseIterator();

  if (!AddInsToDeadList(std::next(EndI->getReverseIterator()), StopPoint))
    return false;
}
```

RCE: удаление инструкций

```
for (auto *EndI : Ends)
  VisitedBB.insert(EndI->getParent());
```

RCE: удаление инструкций

```
for (auto *EndI : Ends)
  VisitedBB.insert(EndI->getParent());

for (auto *StartI : Starts)
  for (auto *SuccBB : successors(StartI->getParent()))
    if (!IsInBetween(SuccBB->front()))
      VisitedBB.insert(SuccBB);
```

RCE: удаление инструкций

```
for (auto *StartI : Starts) {
    auto *StartBB = StartI->getParent();
    for (auto *BB : post_order_ext(StartBB, VisitedBB)) {



    }
}
```

RCE: удаление инструкций

```
for (auto *StartI : Starts) {
    auto *StartBB = StartI->getParent();
    for (auto *BB : post_order_ext(StartBB, VisitedBB)) {
        if (!AddInsToDeadList(BB->rbegin(), (BB == StartBB)
                                          ? StartI->getReverseIterator()
                                          : BB->rend()))
            return false;
    }
}
```

Доработки по RCE

- Поддержать несколько copy.start/copy.end
- Адаптировать все оптимизации к copy.start/cleanup.start, copy.end/cleanup.end
- Найти бенчмарк
- Обучить оптимизации не трогать метаданные copy_init/init/cleanup