FFI Tutorial

В прошлой лекции

- ASM -> ASM
- ASM -> C
- C -> Pascal
- C -> Rust
- C++ -> C
- C -> C++
- Python -> C++
- Raw FFI

Сегодня

- C -> C++

Но помним про Asm

Вызов функции

Чтобы вызвать функцию, нам нужно знать:

- Имя функции
- Способ передачи параметров и возврата результата

Мы можем влиять на все это

Calling Convention

GCC x86-64:

- ms_abi
- sysv_abi

https://godbolt.org/z/Wej41dcje

https://gcc.gnu.org/onlinedocs/gcc/x86-Function-Attributes.html

Имя функции

```
int sum(int a, int b);
```

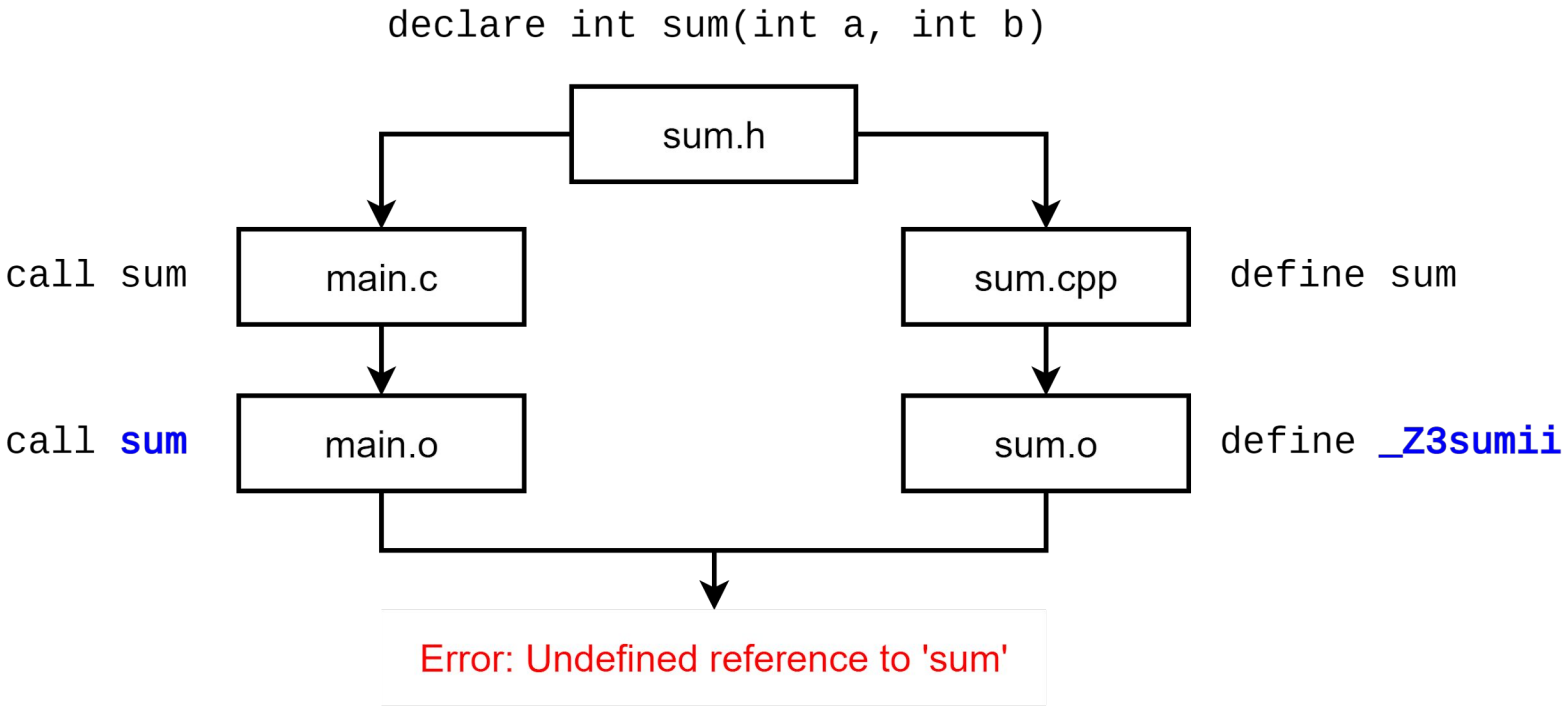
Каким будет имя функции в ASM?

Имя функции

```
int sum(int a, int b);
```

Каким будет имя функции в ASM?

Зависит от компилятора


declare int sum(int a, int b)
sum.h
call sum
main.c
sum.cpp
define sum
call sum
main.o
sum.o
define _Z3sumii
Error: Undefined reference to 'sum'

Воспроизведение

```
$ gcc -c main.c && g++ -c sum.cpp

$ objdump -t main.o | grep sum
... sum

$ objdump -t sum.o | grep sum
... _Z3sumii

$ gcc main.o sum.o
main.c:(.text+0x17): undefined reference to `sum'
```

Манглинг

Искажение имен функций.

- В C++ разрешено манглировать имена типами.
- Благодаря этому существует множество возможностей языка.
- …и отсутствует стандарт на ABI.

```
$ c++filt _Z3sumii

sum(int, int)
```

Применение манглирования

Методы классов:

- `Point Point::move(Vec2d) -> _ZN5Point4moveE5Vec2d`

Перегрузка:

- `int sum(int, int) -> _Z3sumii`
- `double sum(double, double) -> _Z3sumdd`

Пространства имен, шаблоны, перегрузка операторов…

Манглирование

Это полезный механизм, который мешает нашей задаче.

Нам нужен способ отключить его.

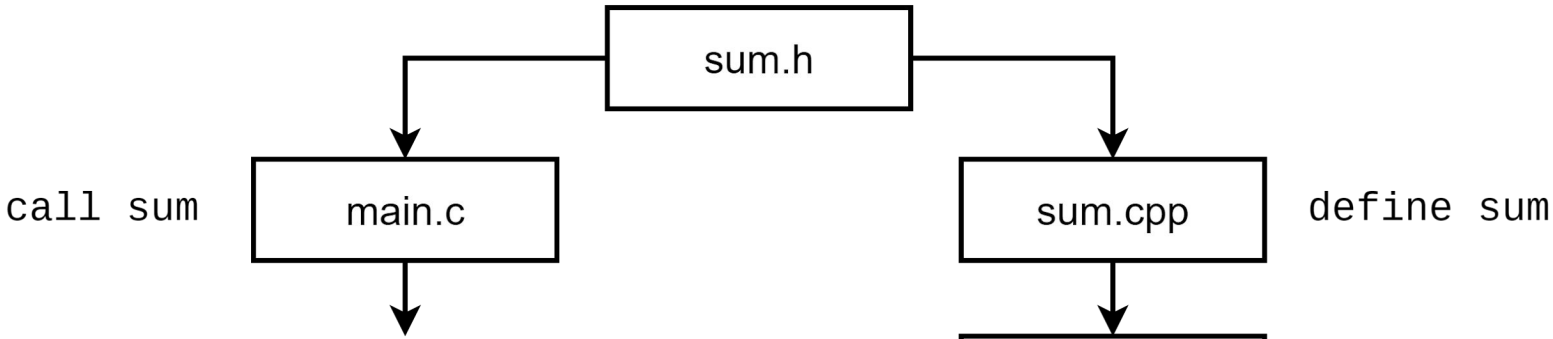
extern “C” — language linkage

Language Linkage

Синтаксис Linkage specification

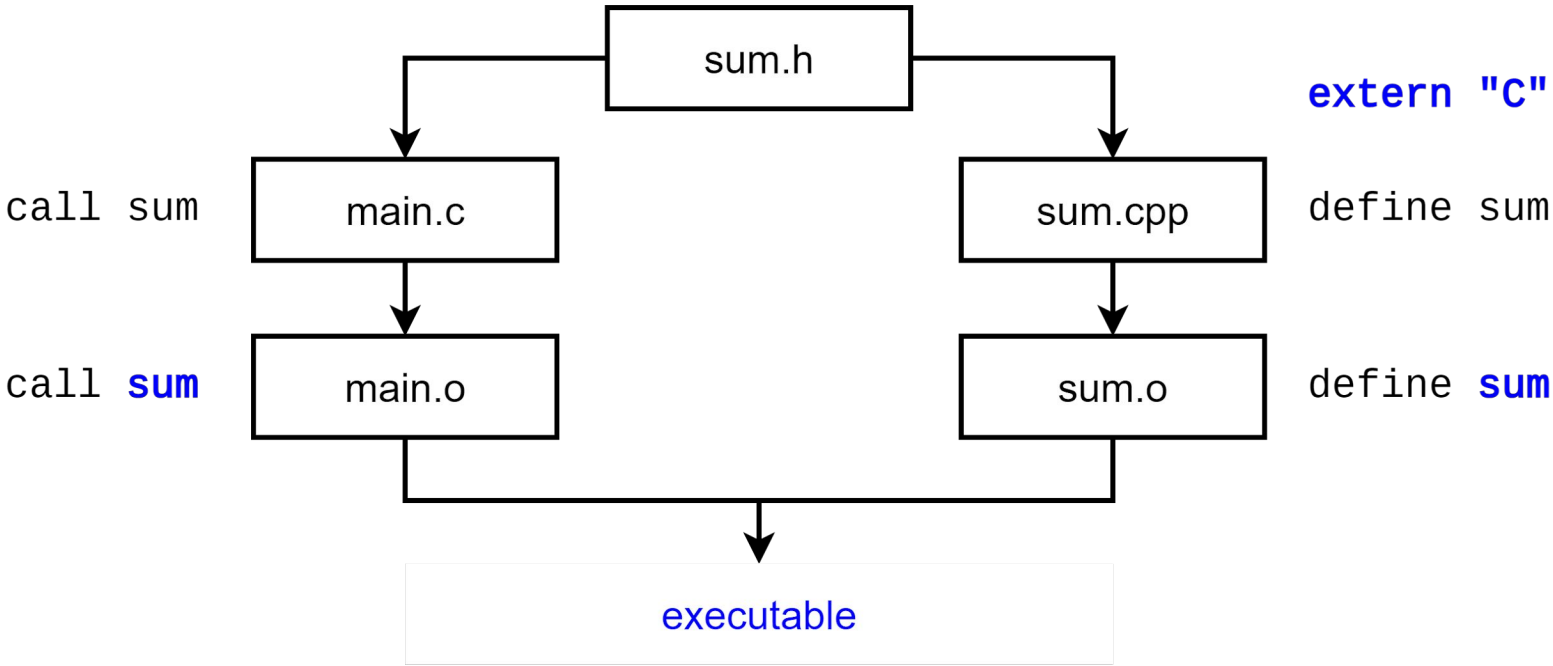
```
extern “C” int sum(int a, int b);

// или

extern “C” {

  int sum(int a, int b);
  // Еще объявления

}
```

```
declare extern "C" int sum(int a, int b)
```




```
error: expected identifier or '('
extern "C" int sum(int a, int b);
       ^~~
```

declare (**extern "C"**)? int sum(int a, int b)



Условное применение `extern "C"`

```
// sum.h
#ifdef __cplusplus
extern "C" {
#endif

int sum(int a, int b);

#ifdef __cplusplus
}
#endif
```


https://godbolt.org/z/sxEbfEda5

Промежуточный итог

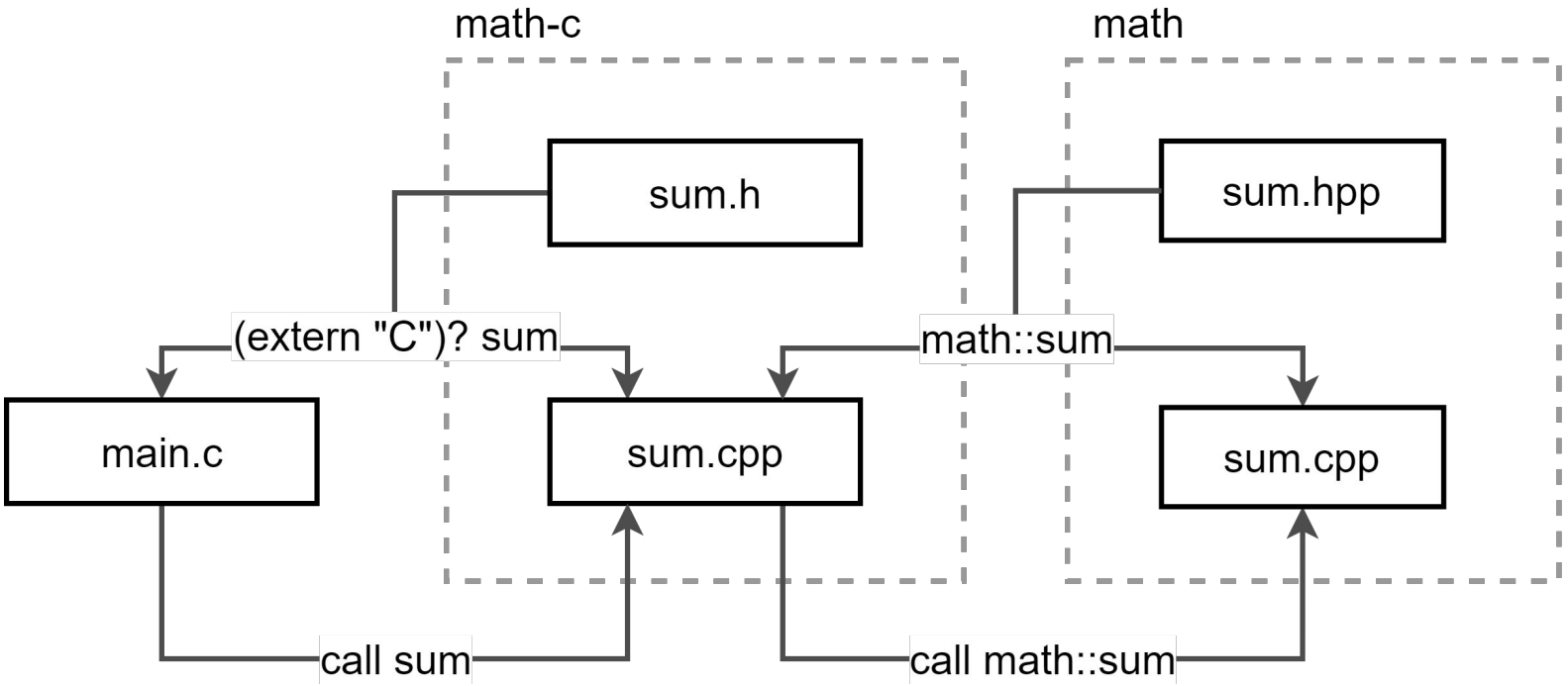
- Пока мы обходились аннотированием существующего кода.
- Часто этого недостаточно:
  - Пространства имен.
  - Методы классов.
  - Перегрузка (не будем рассматривать).

Пространства имен

```
#ifdef __cplusplus
extern "C" {
namespace math {
#endif

int sum(int a, int b);

#ifdef __cplusplus
} // namespace math
} // extern “C”
#endif
```

Это скомпилируется,

но кажется не очень удобным


math-c
math
sum.h
sum.hpp
(extern "C")? sum
math::sum
main.c
sum.cpp
sum.cpp
call sum
call math::sum

main.c

```
#include <math-c/sum.h>

#include <stdio.h>

int main() {
  int x = sum(40, 2);
  printf("x = %d\n", x);
}
```

math-c/sum.h

```
#pragma once

#ifdef __cplusplus
extern "C" {
#endif

int sum(int a, int b);

#ifdef __cplusplus
}
#endif
```

math-c/sum.cpp

```
#include <math-c/sum.h>

#include <math/sum.hpp>

int sum(int a, int b) {
  return math::sum(a, b);
}
```

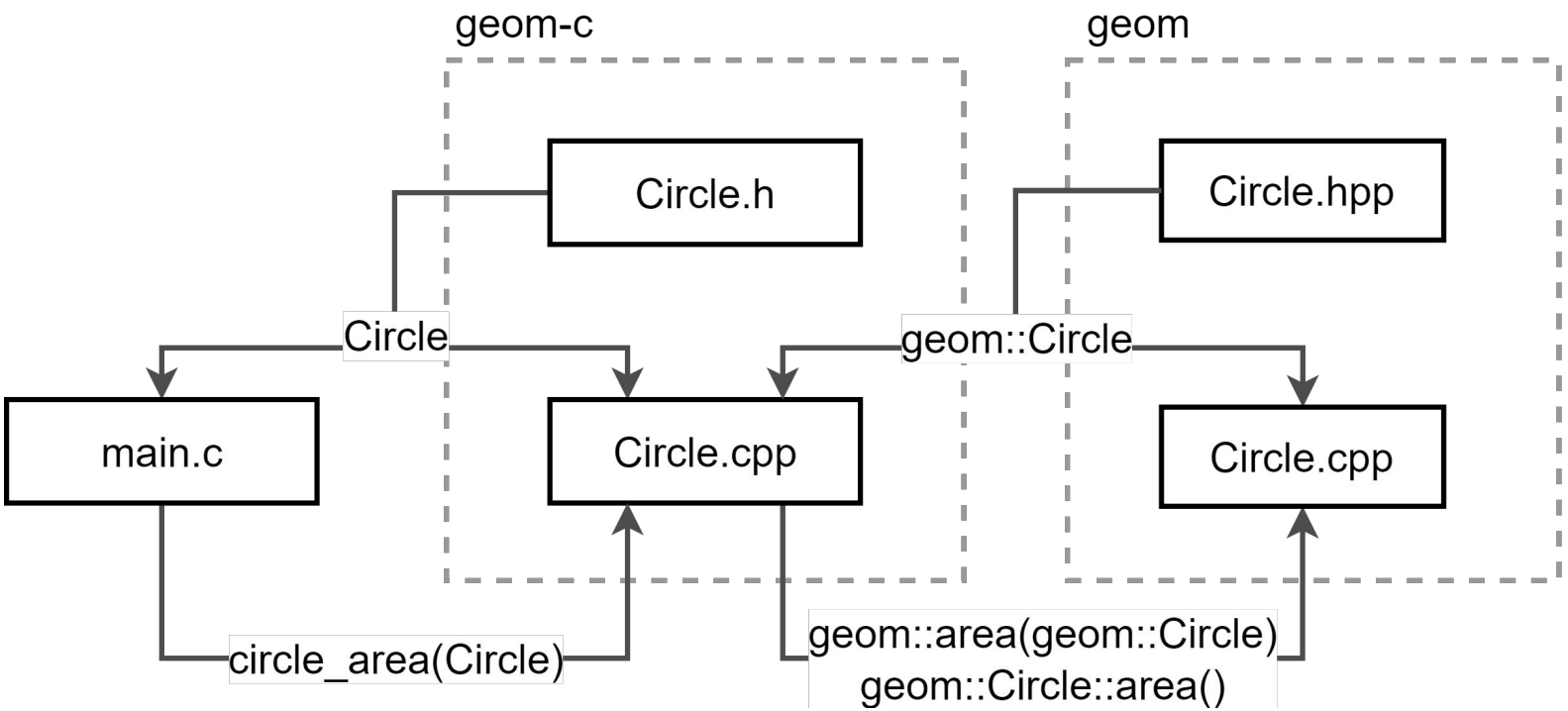
math/sum.hpp, math/sum.cpp

```
#pragma once

namespace math {

int sum(int a, int b);

}
```

```
#include <math/sum.hpp>

namespace math {

int sum(int a, int b) {
  return a + b;
}

}
```

Передача экземпляров структур

- Промежуточная структура
- Указатель
  - void*
  - Opaque Ptr


geom-c
geom
Circle.h
Circle.hpp
Circle
geom::Circle
main.c
Circle.cpp
Circle.cpp
circle_area(Circle)
geom::area(geom::Circle)
geom::Circle::area()

NB

На стыке geom-c/geom нам все равно, что вызывать:
метод geom::Circle::area
или свободную функцию geom::Circle::area(geom::Circle).

Обертка реализуется почти одинаково.

geom-c/Circle.h

```
#include <geom-c/Point.h>

extern "C" { // TODO: ifdef/endif

struct Circle {
  struct Point center_;
  double radius_;
};

double circle_area(struct Circle c);

}
```

geom-c/Circle.cpp

```
#include <geom-c/Circle.h>
#include <geom/Circle.hpp>

double circle_area(Circle c) {
  geom::Circle circle{
      geom::Point{c.center_.x_, c.center_.y_},
      c.radius_};

  return circle.area(); // или geom::area(circle)
}
```

Проблема решения

На каждый вызов circle_area создается временный geom::Circle.

- Накладно для больших объектов.
- Каждый тип приходится дублировать структурой в c-api

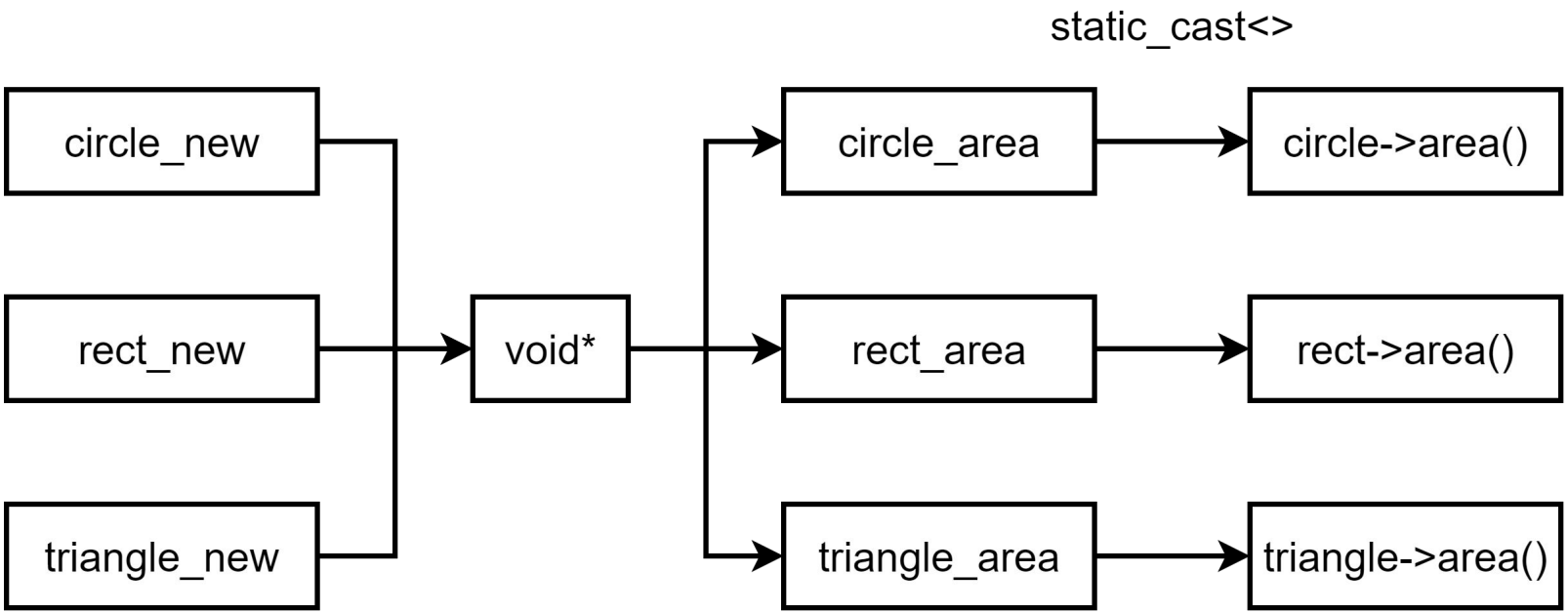
Решение — стирание типа. Вместо зеркалирующего типа возвращать указатель.

geom-c/Circle.h

```
typedef void Circle;

Circle* circle_new(Point center, double radius);
void circle_free(Circle* circle);

double circle_area(Circle* c);
```

geom-c/Circle.cpp

```
Circle* circle_new(Point center, double radius) {
  return new geom::Circle{
      geom::Point{center.x_, center.y_}, radius};
}

void circle_free(Circle* circle) {
  delete static_cast<geom::Circle*>(circle);
}

double circle_area(Circle* c) {
  const auto* circle = static_cast<geom::Circle*>(c);
  return circle->area();
}
```


static_cast<>
circle_new
rect_new
triangle_new
void*
circle_area
rect_area
triangle_area
circle->area()
rect->area()
triangle->area()

Проблемы void*

- void* полностью стирает тип

```
Circle* c = circle_new(...);
double area = rect_area(c); // Compiles, UB
```

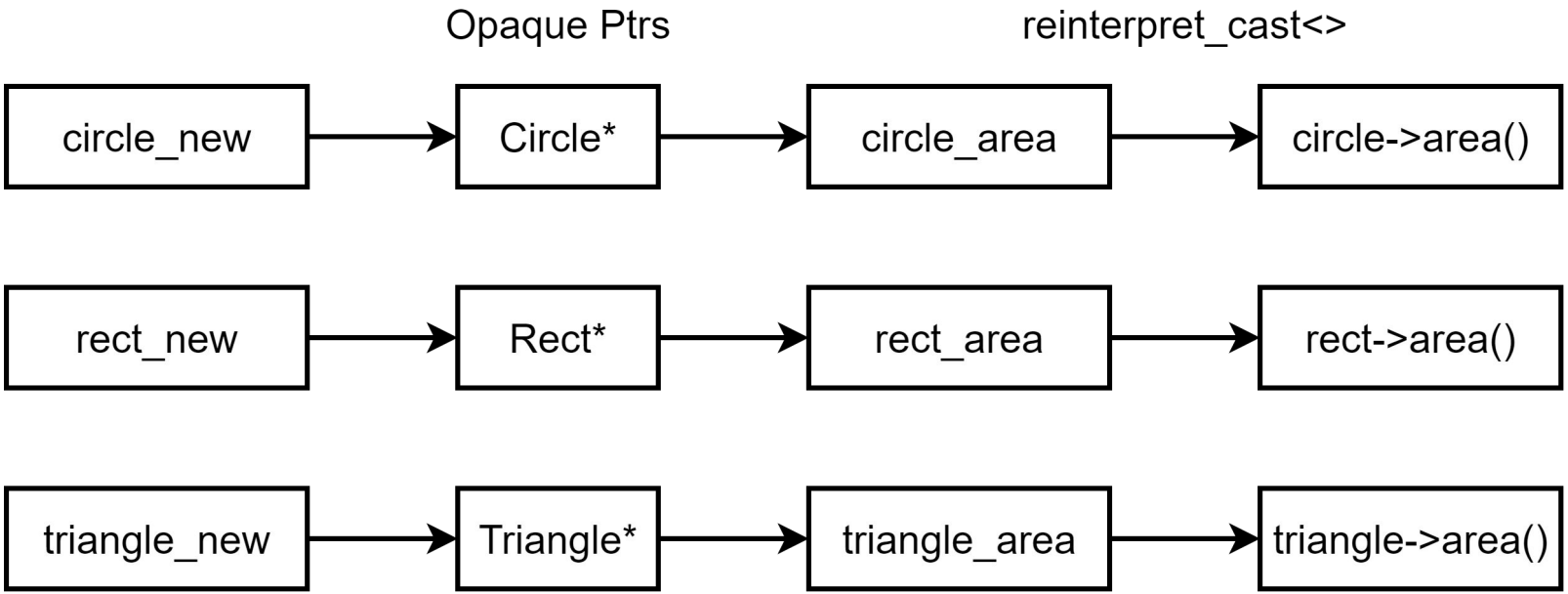
- Вынужденные аллокации в конструкторах
  (с этим мы можем только смириться)

geom-c/Circle.h

```
// Здесь был ~~Вася~~ void
typedef struct Circle Circle;

Circle* circle_new(Point center, double radius);
void circle_free(Circle* circle);

double circle_area(Circle* c);
```

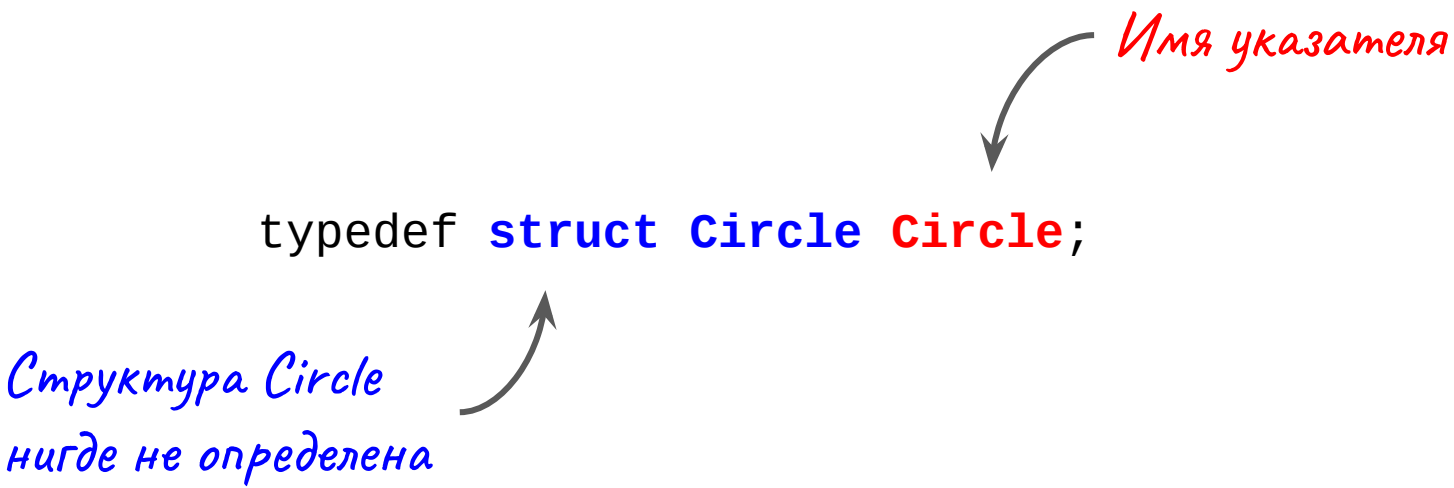
geom-c/Circle.cpp

```
Circle* circle_new(Point center, double radius) {
  return reinterpret_cast<Circle*>(new geom::Circle{...});
}

void circle_free(Circle* circle) {
  delete reinterpret_cast<geom::Circle*>(circle);
}

double circle_area(Circle* c) {
  const auto* circle = reinterpret_cast<geom::Circle*>(c);
  return circle->area();
}
```


Opaque Ptrs
reinterpret_cast<>
circle_new
Circle*
circle_area
circle->area()
rect_new
Rect*
rect_area
rect->area()
triangle_new
Triangle*
triangle_area
triangle->area()

Opaque Ptr

Opaque Ptr («непрозрачный» указатель) — это указатель на тип без определения.



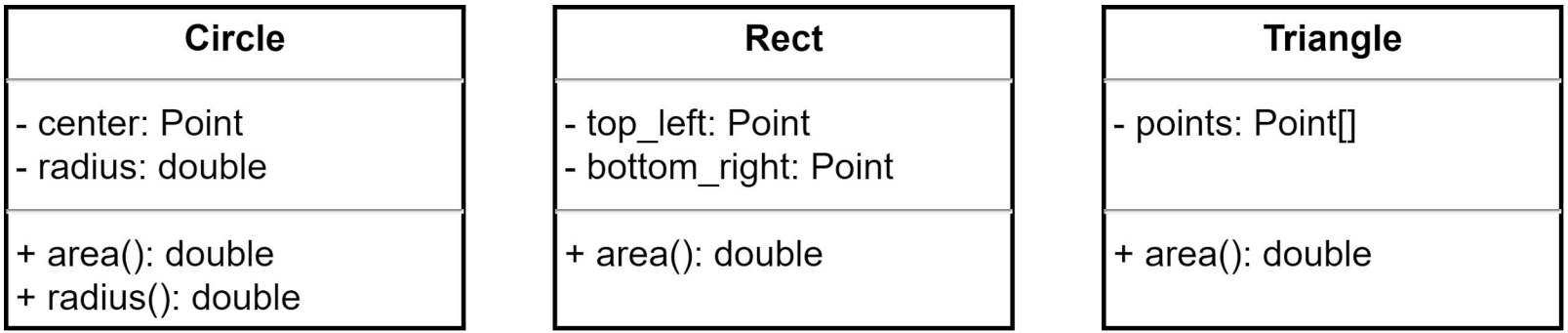
Opaque Ptr

```
Rect* r = rect_new(...);
circle_area(r);
```

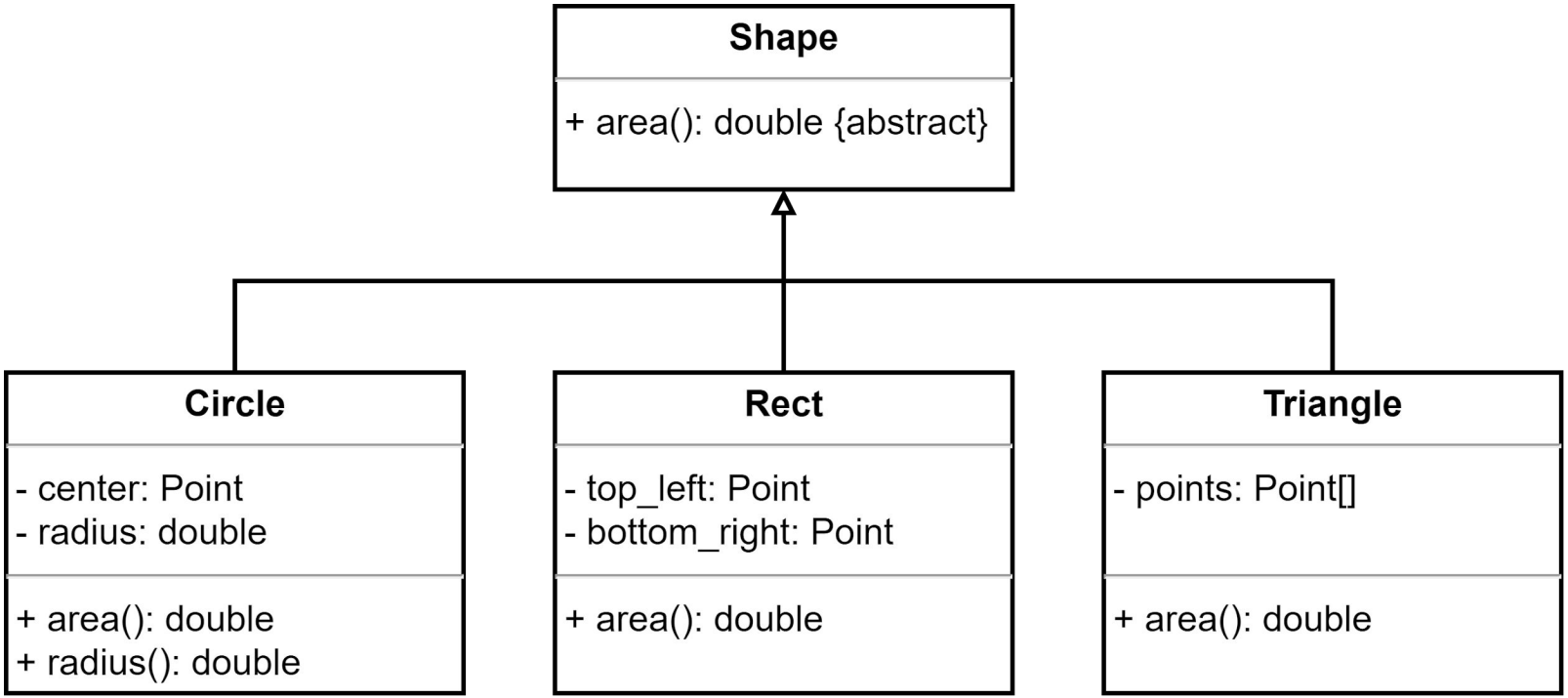
error: passing argument 1 of ‘circle_area’ from incompatible pointer type

`-Werror`

Добавляем наследование



Добавляем наследование

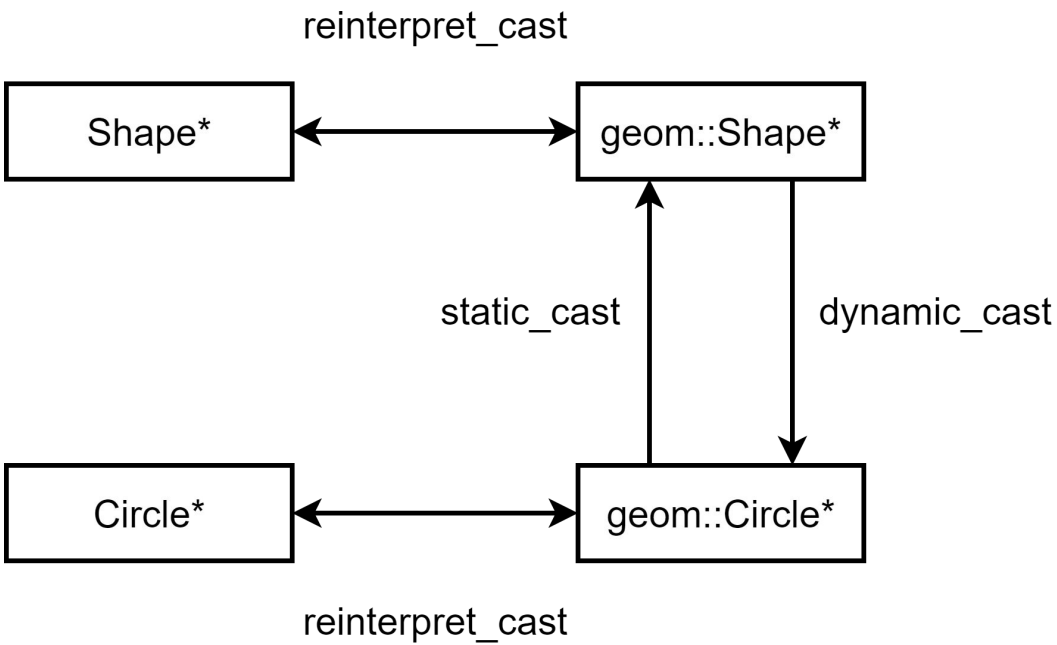


Методы

Shape::area() — абстрактный (pure virtual)

Circle::radius() — конкретный

Допустимые преобразования



c-api

```
typedef struct Shape Shape;
typedef struct Circle Circle;

Circle* circle_new(Point center, double radius);

double circle_radius(Circle* circle)

double shape_area(Shape* c);
```

c-api

```
double circle_radius(Circle* circle) {
  auto* c = reinterpret_cast<geom::Circle*>(circle);
  return c->radius_;
}
```

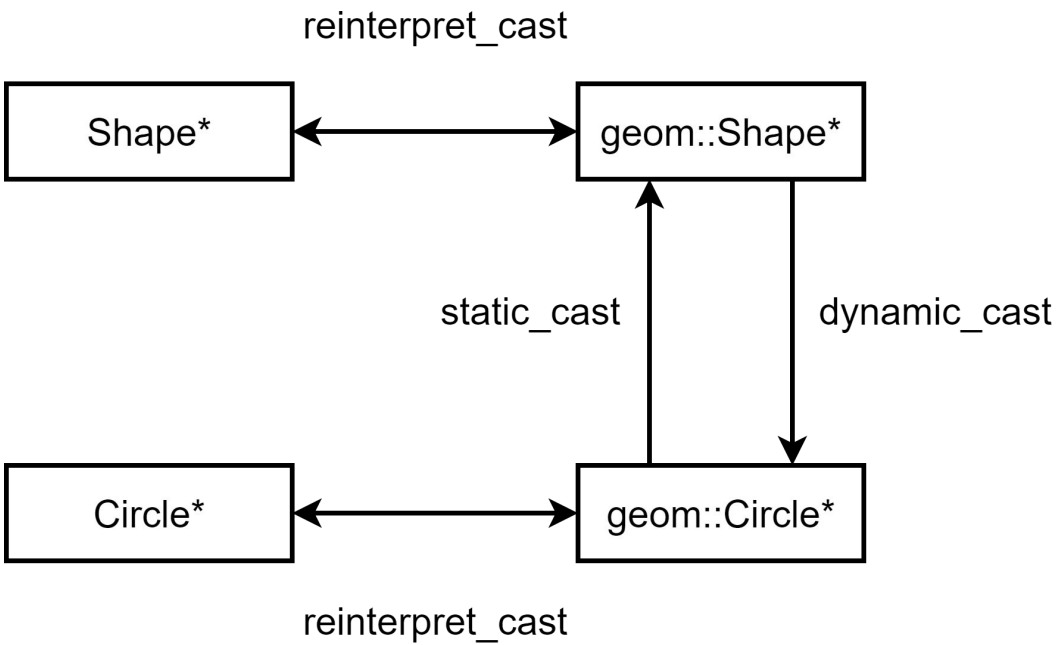
c-api

```
double shape_area(Shape* c) {
  const auto* s = reinterpret_cast<geom::Shape*>(c);
  return s->area();
}
```

Как перейти от Circle* к Shape*?

```
Circle* c = circle_new(center, 1);
double area = shape_area(c); // Oops
```

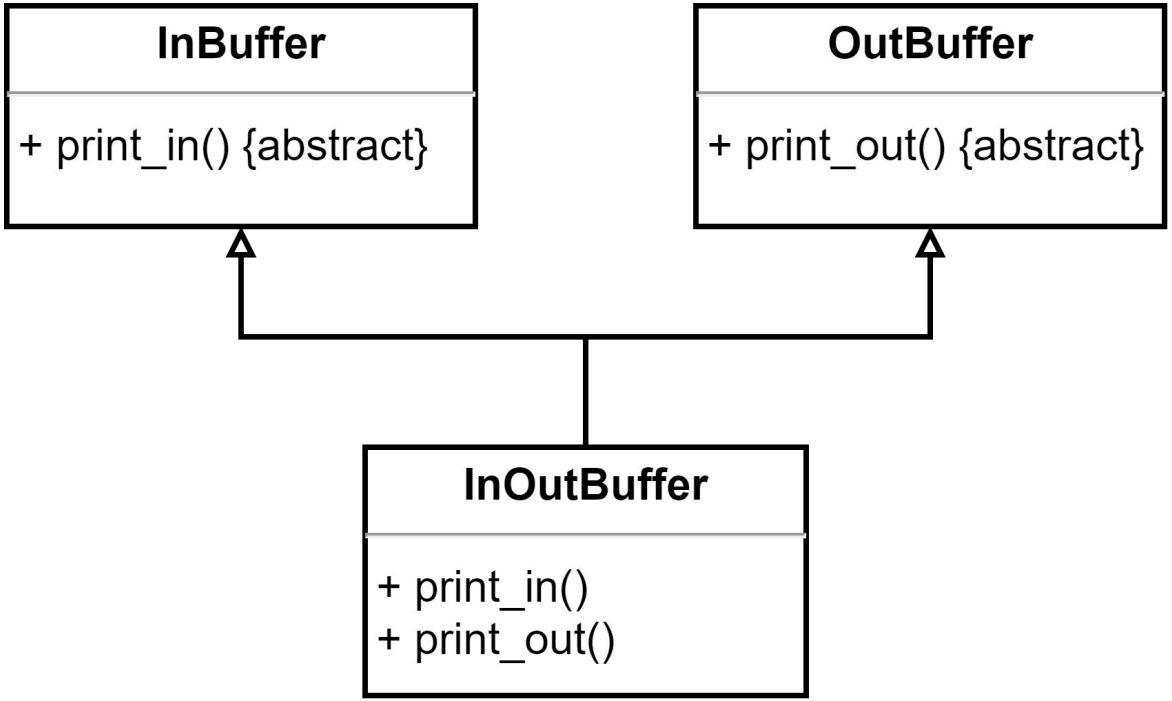
Как перейти от Circle* к Shape*?



Circle* -> Shape*

```
Circle* c = circle_new(center, 1);
double area = shape_area(circle_to_shape(c));

Shape* circle_to_shape(Circle* c) {
  auto* circle = reinterpret_cast<geom::Circle*>(c);
  auto* shape = static_cast<geom::Shape*>(circle);
  return reinterpret_cast<Shape*>(shape);
}
```

Пример, как все сломать



https://godbolt.org/z/7aGYMrfTr

STL

```
std::vector<Circle> generate_circles();
```

STL

```
std::vector<Circle> generate_circles();


// C-api
typedef struct Circles Circles;

Circles* generate_circles();

Circle* circles_get(Circles* circles, size_t idx);
```

STL

```
Circles* generate_circles() {
  auto geom_circles = geom::generate_circles();

  auto* circles
     = new std::vector<geom::Circle>(
         std::move(geom_circles));

  return reinterpret_cast<Circles*>(circles);
}
```

STL

```
Circle* circles_get(Circles* circles, size_t idx) {
  auto* geom_circles
     = reinterpret_cast<
         std::vector<geom::Circle>*>(circles);

  auto* geom_circle = &geom_circles->at(idx);

  return reinterpret_cast<Circle*>(geom_circle);
}
```

Вопросы?