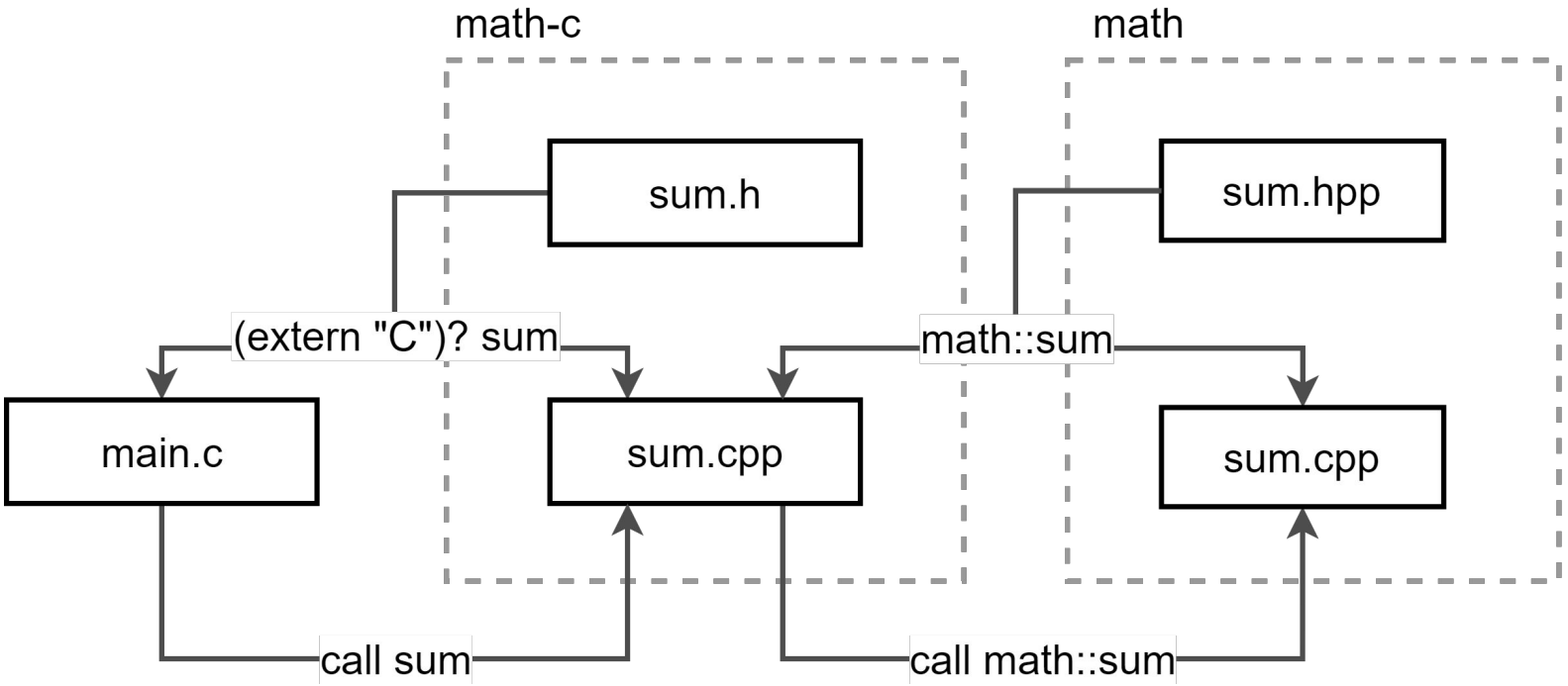
Кроссязыковые библиотеки

В предыдущей лекции

Основные идеи

- Для корректного вызова функции нужно соблюдать ABI
- На ABI языка C есть стандарт


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

Класс Shape

```
namespace geometry {

class Shape {
 public:
  virtual ~Shape() = default;
  virtual double area() const = 0;
};

}
```

Задача: сделать тип доступным из C

```
namespace geometry {
class Circle : public Shape {
 public:
  Circle(Point center, double radius);

  double area() const override;
  double radius() const;
  ...
};
}
```

libgeometry-c/Point.h

```
// Определяем Point за пределами namespace geometry

#pragma once

typedef struct {
  double x_;
  double y_;
} Point;
```

libgeometry-c/Shape.h

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

typedef void Shape;

...

#ifdef __cplusplus
}
#endif
```

libgeometry-c/Circle.h

```
// Оборачиваем конструктор Circle
#ifdef __cplusplus
extern "C" {
#endif

Shape* geometry_circle_new(
       Point center, double radius);

#ifdef __cplusplus
}
#endif
```

libgeometry-c/Circle.h

```
Shape* geometry_circle_new(Point center, double r) {
  const geometry::Point pt{center.x_, center.y_};

  return static_cast<geometry::Shape*>(
    new geometry::Circle(pt, r));
}
```

Реализация — код на C++

От языка C здесь только линковка

libgeometry-c/Shape.h

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

typedef void Shape;

double geometry_shape_area(const Shape* self);
void geometry_shape_delete(const Shape* self);

#ifdef __cplusplus
}
#endif
```

libgeometry-c/Shape.cpp

```
double geometry_shape_area(const Shape* self) {
  const auto* shape
      = static_cast<const geometry::Shape*>(self);
  return shape->area();
}

void geometry_shape_delete(const Shape* self) {
  const auto* shape
      = static_cast<const geometry::Shape*>(self);
  delete shape;
}
```

testapp/main.c

```
const Point pt = {0, 0};
const Shape* circle = geometry_circle_new(pt, 2);
const double area = geometry_shape_area(circle);
printf("circle area = %.6lf\n", area);
geometry_shape_delete(circle);
```

Задача: сделать тип доступным из C

```
namespace geometry {
class Circle : public Shape {
 public:
  Circle(Point center, double radius);

  double area() const override; // DONE
  double radius() const;        // ???
  ...
};
}
```

libgeometry-c/Circle.h

```
// extern "C"


Shape* geometry_circle_new(
       Point center, double radius);

double geometry_circle_radius(const Shape* self);
```

libgeometry-c/Circle.cpp

```
double geometry_circle_radius(const Shape* self) {
  const auto* shape
    = static_cast<const geometry::Shape*>(self);

  const auto* circle
    = dynamic_cast<const geometry::Circle*>(shape);

  return circle->radius();
}
```

Down Cast

```
Shape* s = new Circle();

auto* c = static_cast<Circle*>(s);   // OK

auto* t = static_cast<Triangle*>(s); // No Error, UB
```

Down Cast

```
Shape* s = new Circle();

auto* c = dynamic_cast<Circle*>(s);   // OK

auto* t = dynamic_cast<Triangle*>(s); // nullptr
```

dynamic_cast выполняет приведение с учетом динамического типа обьекта

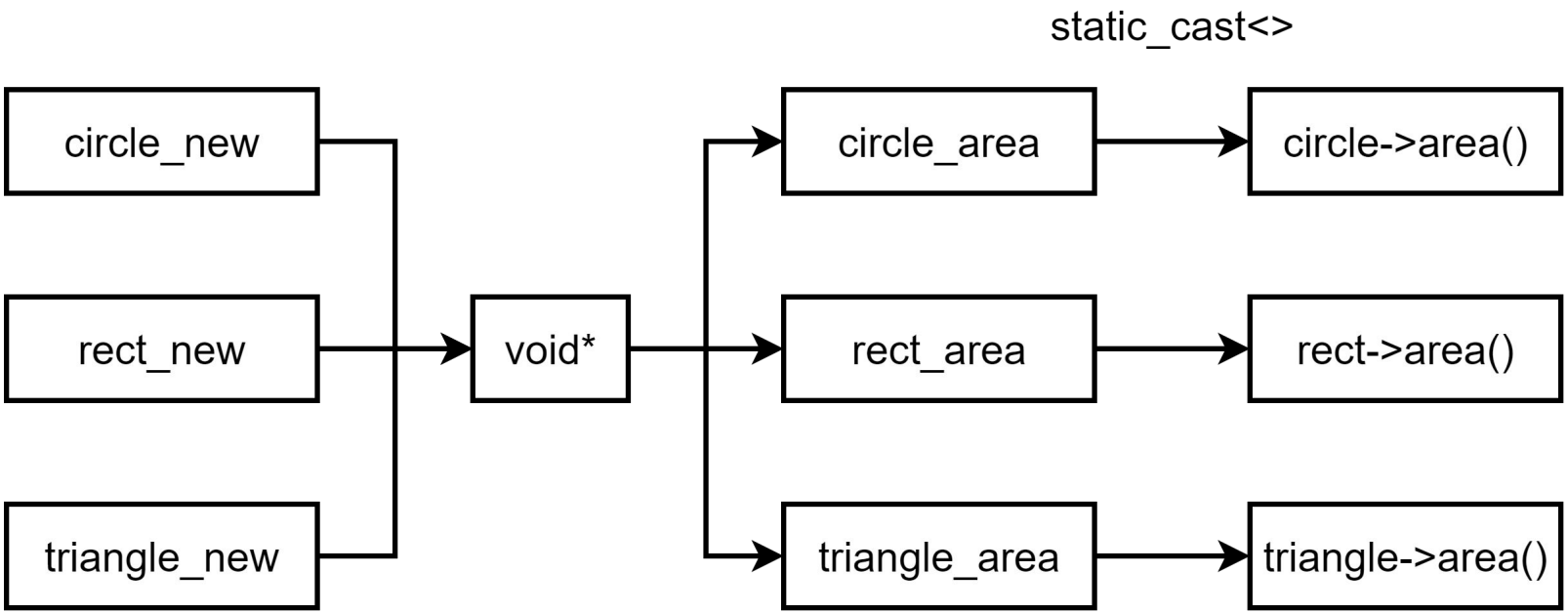
Резюме

В C-Api могут быть только:

- Standard-layout типы
- Функции с C-linkage

Не может быть:

- Пространств имен
- Исключений
- Всего остального, что мы так любим


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

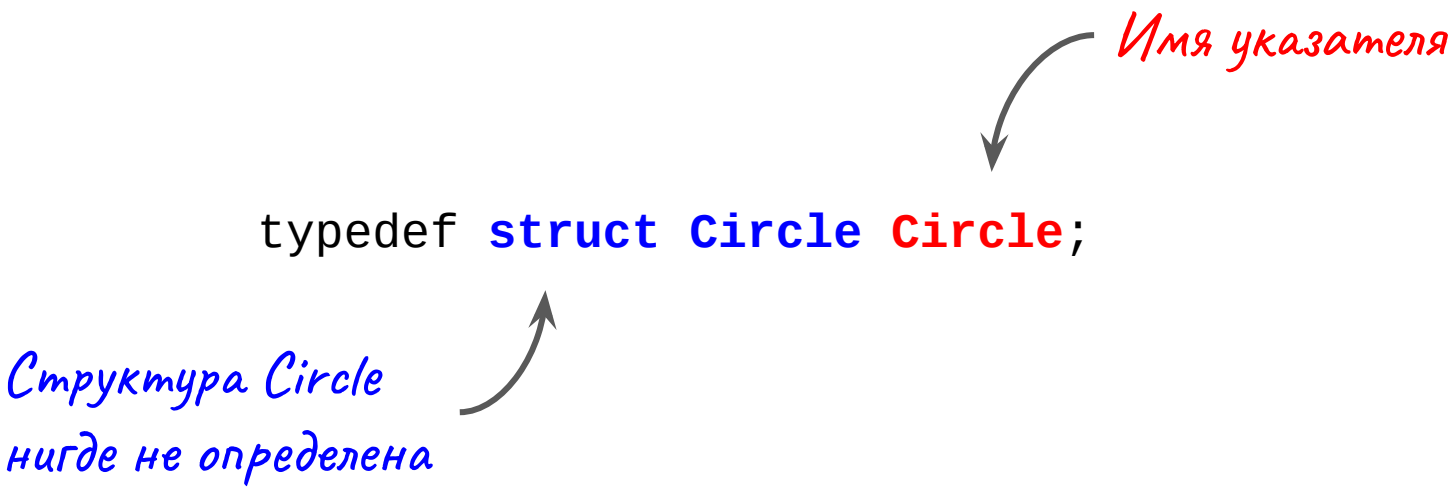
```
Circle* c = circle_new(...);
double area = rect_area(c); // Compiles, UB
```

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

Opaque Ptrs

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

geom-c/Circle.h

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

Circle* geometry_circle_new(Point center,
                            double radius);

double circle_radius(const Circle* self);
```

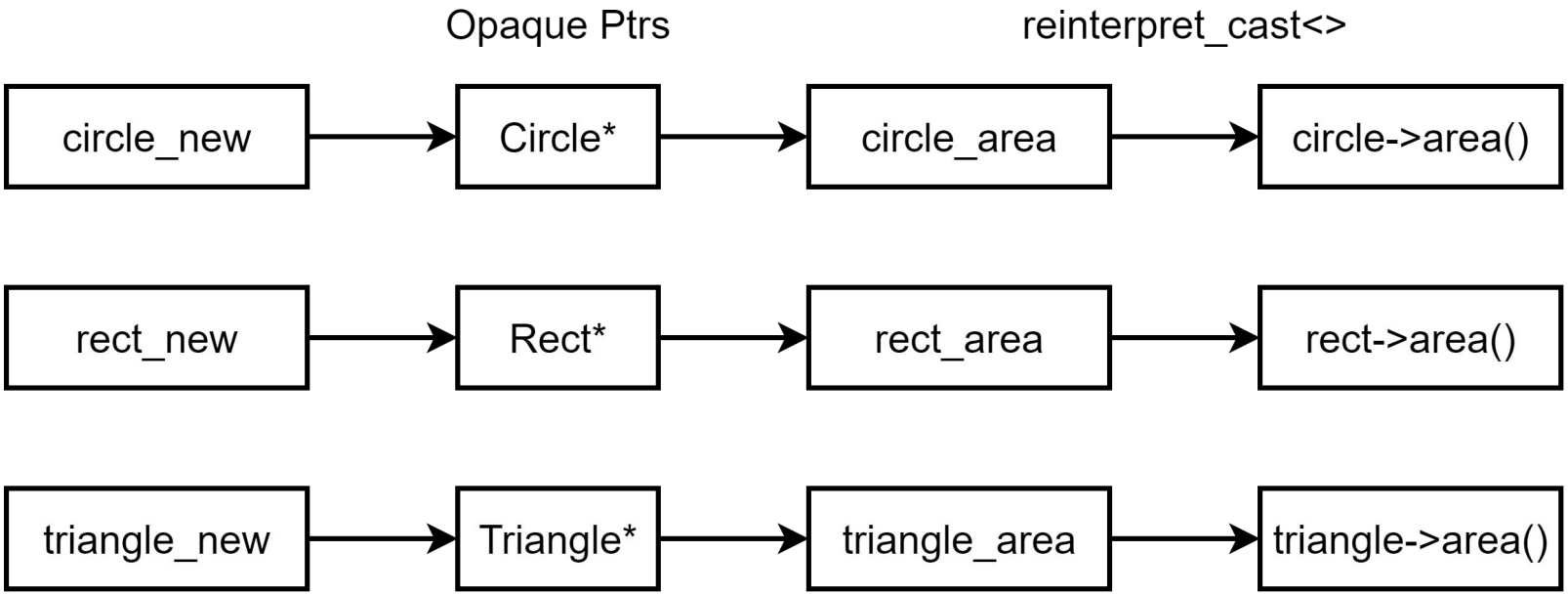
geom-c/Circle.cpp

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

double geometry_circle_radius(const Shape* self) {
  const auto* shape
      = reinterpret_cast<const geometry::Shape*>(self);

  const auto* circle
      = dynamic_cast<const geometry::Circle*>(shape);

  return circle->radius();
}
```

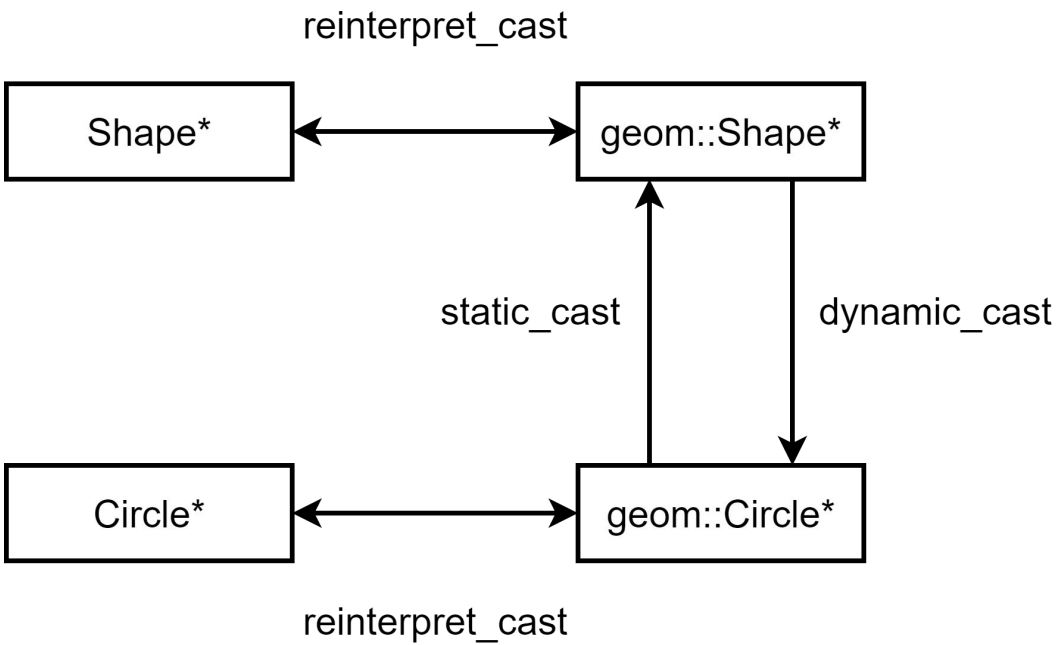

Opaque Ptrs
reinterpret_cast<>
circle_new
Circle*
circle_area
circle->area()
rect_new
Rect*
rect_area
rect->area()
triangle_new
Triangle*
triangle_area
triangle->area()

Как перейти от Circle* к Shape*?

```
double shape_area(const Shape* c);

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);
}
```

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);
}
```

Python -> C -> C++

Вспомним, что у нас есть

- Библиотека geometry на C++
- Библиотека geometry-c на C++ и интерфейсом с C-linkage

Хотим использовать типы из geometry в Python.

Foreign Function Interface (FFI)

Механизм вызова функций из других языков.

В Python представлен модулем ctypes.

client/geometry.py: определение структуры

```
import ctypes

geometry_lib = \
    ctypes.cdll.LoadLibrary('libgeometry-c.so')

class Point(ctypes.Structure):
    _fields_ = [
        ('x', ctypes.c_double),
        ('y', ctypes.c_double)
    ]
```

client/geometry.py: определение функций

```
...
circle_new = geometry_lib.geometry_circle_new
circle_new.argtypes = [Point, ctypes.c_double]
circle_new.restype = ctypes.c_void_p


shape_area = geometry_lib.geometry_shape_area
shape_area.argtypes = [ctypes.c_void_p]
shape_area.restype = ctypes.c_double
```

client/main.py

```
import geometry


circle = geometry.circle_new(geometry.Point(0, 0), 2)
print(geometry.shape_area(circle))
geometry.shape_delete(circle)
```

Нам хочется более нативного интерфейса

client/geometry.py

```
class Shape:
    def __init__(self, obj):
        self._obj = obj

    def __del__(self):
        shape_delete(self._obj)

    def area(self):
        return shape_area(self._obj)
```

client/geometry.py

```
class Circle(Shape):
    def __init__(self, center: Point, radius: float):
        obj = circle_new(center, radius)
        super(Circle, self).__init__(obj)

    @property
    def radius(self):
        return circle_radius(self._obj)
```

client/main.py

```
c = geometry.Circle(geometry.Point(0, 0), 2)

print(c.area())

print(c.radius)
```

Резюме

- FFI — универсальный механизм.
  - Удобство C++ внутри библиотеки
  - Доступность для всех языков снаружи
- Требует много ручной работы:
  - Обертка над C++-кодом
  - Обертка над C-API
  - Нативная обертка для целевого языка

Java -> C++

Java class

```
public class Geometry {
    static { System.loadLibrary("libgeometry-c"); }

    public class Point { ... }

    public native Object geometry_circle_new(
            Point center, double radius);
}
```

By hand

```
$ javac -h . Geometry.java
$ ls
'Geometry$Point.class'   Geometry.class   Geometry.h
```

Geometry.h

```
// extern "C"

JNIEXPORT jobject JNICALL
Java_Geometry_geometry_1circle_1new(
    JNIEnv *, jobject, jobject, jdouble);
```

CMake + Java

https://cmake.org/cmake/help/latest/module/UseJava.html

Q&A