

Яндекс Такси

# Полезный constexpr

**Полухин Антон**

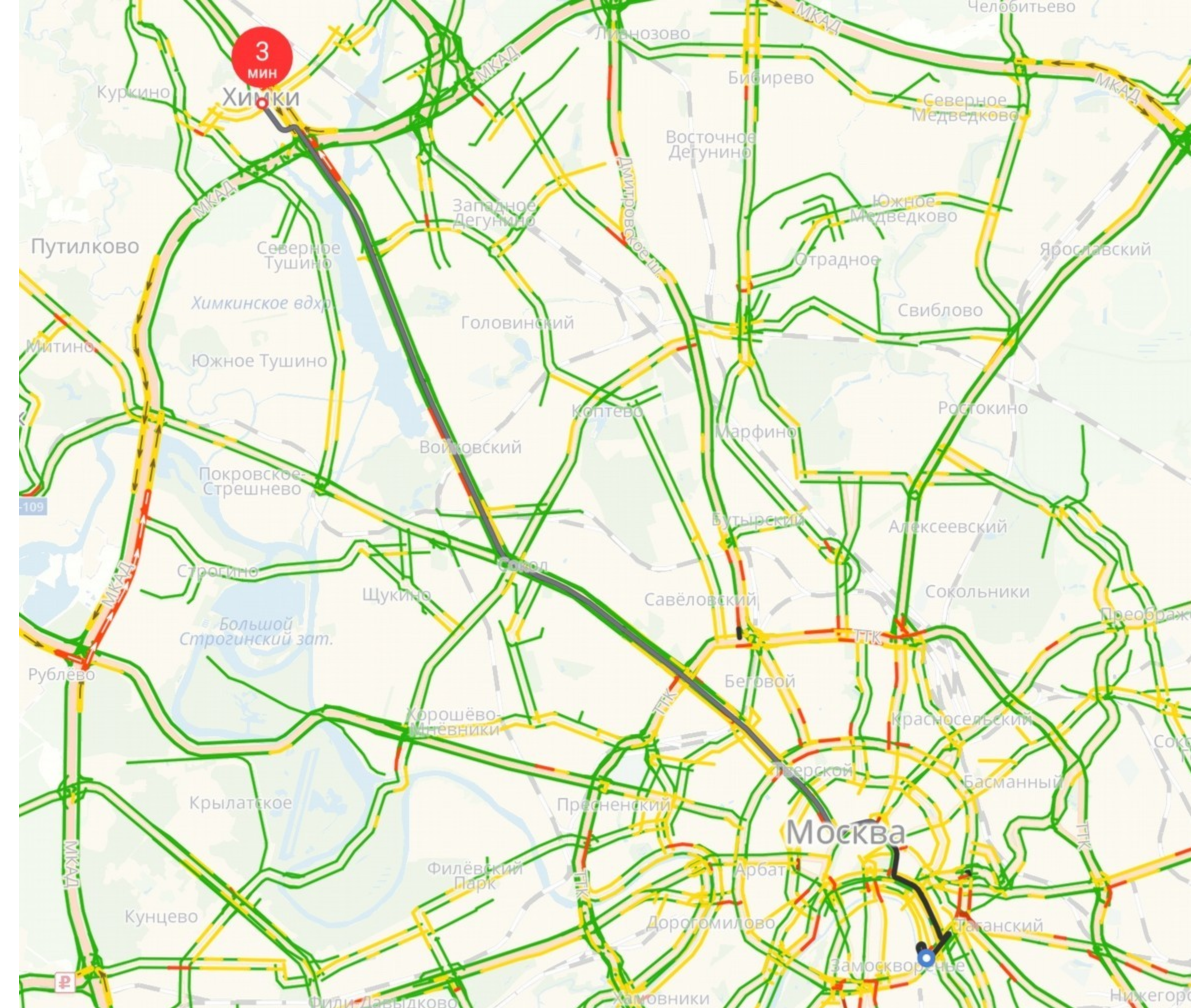
Antony Polukhin

Яндекс Такси



# Содержание

- Старое и известное
- Старое и незнакомое
- Ближайшее будущее
- Очень далёкое будущее



- C++14

- C++26

• 45 мин


Подъезд

ЭКОНОМ  
4₽



КОМФОРТ  
8₽

  
**КОМФОРТ+**  
**9₽**



БИЗНЕС  
34₽



МИНИВЭН  
15₽

ДЕТСКИЙ  
2Р

Комментарий, пожелания

Способ оплаты  
Команда Яндекс.Такси



# Старое и известное

# Старое и известное

```
#include <boost/array.hpp>

inline unsigned factorial(unsigned n) {
    return n ? n * factorial(n - 1) : 1;
}
```

# Старое и известное

```
#include <boost/array.hpp>
```

```
inline unsigned factorial(unsigned n) {  
    return n ? n * factorial(n - 1) : 1;  
}
```

```
boost::array<int, factorial(3)> a;
```

# Старое и известное

**error:** call to non-constexpr function ‘unsigned int factorial(unsigned int)’

```
boost::array<int, factorial(3)> a;
```

~~~~~^~~

**error:** call to non-constexpr function ‘unsigned int factorial(unsigned int)’

note: in template argument for type ‘long unsigned int’

```
boost::array<int, factorial(3)> a;
```

# Старое и известное

```
#include <array>
```

```
constexpr unsigned factorial(unsigned n) {  
    return n ? n * factorial(n - 1) : 1;  
}
```

```
std::array<int, factorial(3)> a;
```



# Старое и известное (2)

```
#include <stdexcept>
```

```
int do_something(unsigned n) {  
    constexpr auto i = 7 + 20;  
    // ...  
    return i;  
}
```

# Старое и известное (2)

```
#include <stdexcept>
```

```
int do_something(unsigned n) {  
    constexpr auto i = 7 + 20;  
    // ...  
    return i;  
}
```

# Старое и известное (2)

```
#include <stdexcept>

constexpr unsigned factorial(unsigned n) {
    if (n > 10)
        throw std::out_of_range("n is greater than 10");
    return n ? n * factorial(n - 1) : 1;
}

int do_something(unsigned n) {
    auto i = factorial(17);
    // ...
    return i;
}
```

# Старое и известное (2)

```
#include <stdexcept>

constexpr unsigned factorial(unsigned n) {
    if (n > 10)
        throw std::out_of_range("n is greater than 10");
    return n ? n * factorial(n - 1) : 1;
}

int do_something(unsigned n) {
    throw std::out_of_range("n is greater than 10");
}
```



# Старое и известное (2)

```
#include <stdexcept>

constexpr unsigned factorial(unsigned n) {
    if (n > 10)
        throw std::out_of_range("n is greater than 10");
    return n ? n * factorial(n - 1) : 1;
}

int do_something(unsigned n) {
    constexpr auto i = factorial(17);
    // ...
    return i;
}
```

# Старое и известное (2)

In function `‘int do_something(unsigned int)’`:

in `constexpr` expansion of `‘factorial(17)’`

**error:** expression `‘<throw-expression>’` is not a constant expression

`throw std::out_of_range("n is greater than 10");`

# Старое и незнакомое

# Статическая инициализация

All static initialization strongly happens before any dynamic initialization.

```
int do_something(unsigned n) {  
    static const int i = 17;  
}
```



# Статическая инициализация

All static initialization strongly happens before any dynamic initialization.

```
int do_something(unsigned n) {  
    static const int i = n;
```

# Статическая инициализация

All static initialization strongly happens before any dynamic initialization.

```
int do_something(unsigned n) {  
    static const int i = factorial(7);  
}
```

# Старое и незнакомое

```
namespace std {  
    class mutex {  
    public:  
        constexpr mutex() noexcept;  
        ~mutex();  
  
        mutex(const mutex&) = delete;  
        mutex& operator=(const mutex&) = delete;  
        /*...*/  
    };  
}
```

# Старое и новое

```
namespace std {  
    class mutex {  
    public:  
        constexpr mutex() noexcept;  
        ~mutex();  
  
        mutex(const mutex&) = delete;  
        mutex& operator=(const mutex&) = delete;  
        /* ... */  
    };  
}
```



# Статическая инициализация

A *constant initializer* for a variable or temporary object `o` is an initializer whose full-expression is a constant expression, except that if `o` is an object, **such an initializer may also invoke constexpr constructors** for `o` and its subobjects even if those objects are of non-literal class types.

[ *Note*: Such a class may have a non-trivial destructor. — *end note*]

All static initialization strongly happens before any dynamic initialization.

# Старое и незнакомое

```
// 1.cpp
```

```
namespace {
```

```
    std::mutex m;
```

```
}
```

```
std::mutex& global_mutex() { return m; }
```

```
// 2.cpp
```

```
std::vector<int> data = []() {
```

```
    std::lock_guard<std::mutex> l(global_mutex()); // Oops?
```

```
    return some_creepy_api();
```

```
}();
```

# Старое и незнакомое (2)

# Старое и незнакомое (2)

```
// Bubble-like sort. Anything complex enough will work
```

```
template <class It>
```

```
void sort(It first, It last) { /*...*/ }
```

```
inline int generate(int i) {
```

```
    int a[7] = {3, 7, 4, i, 8, 0, 1};
```

```
    sort(a + 0, a + 7);
```

```
    return a[0] + a[6];
```

```
}
```

```
int no_constexpr() { return generate(1); }
```



# Старое и незнакомое (2)

```
no_constexpr():
```

```
movabs rax, 30064771075
```

```
mov DWORD PTR [rsp-20], 0
```

```
lea rdx, [rsp-40]
```

```
mov QWORD PTR [rsp-40], rax
```

```
lea rsi, [rdx+28]
```

```
movabs rax, 4294967300
```

```
mov QWORD PTR [rsp-32], rax
```

```
lea rax, [rsp-40]
```

```
mov DWORD PTR [rsp-16], 1
```

```
add rax, 4
```

```
mov DWORD PTR [rsp-24], 8
```

```
mov rcx, rax
```

```
.L2:
```

```
add rax, 4
```

```
cmp rax, rsi
```

```
je .L8
```

```
.L6:
```

```
mov edi, DWORD PTR [rax]
```

```
mov r8d, DWORD PTR [rdx]
```

```
cmp edi, r8d
```

```
jge .L2
```

```
mov DWORD PTR [rax], r8d
```

```
add rax, 4
```

```
mov DWORD PTR [rdx], edi
```

```
cmp rax, rsi
```

```
jne .L6
```

```
.L8:
```

```
lea rax, [rcx+4]
```

```
cmp rax, rsi
```

```
je .L3
```

```
mov rdx, rcx
```

```
mov rcx, rax
```

```
jmp .L6
```

```
.L3:
```

```
mov eax, DWORD PTR [rsp-16]
```

```
add eax, DWORD PTR [rsp-40]
```

```
ret
```

# Старое и незнакомое (2)

```
// Bubble-like sort. Anything complex enough will work
```

```
template <class It>
```

```
constexpr void sort(It first, It last) { /*...*/ }
```

```
constexpr int generate(int i) {
```

```
    int a[7] = {3, 7, 4, i, 8, 0, 1};
```

```
    sort(a + 0, a + 7);
```

```
    return a[0] + a[6];
```

```
}
```

```
int no_constexpr() { return generate(1); }
```

# Старое и незнакомое (2)

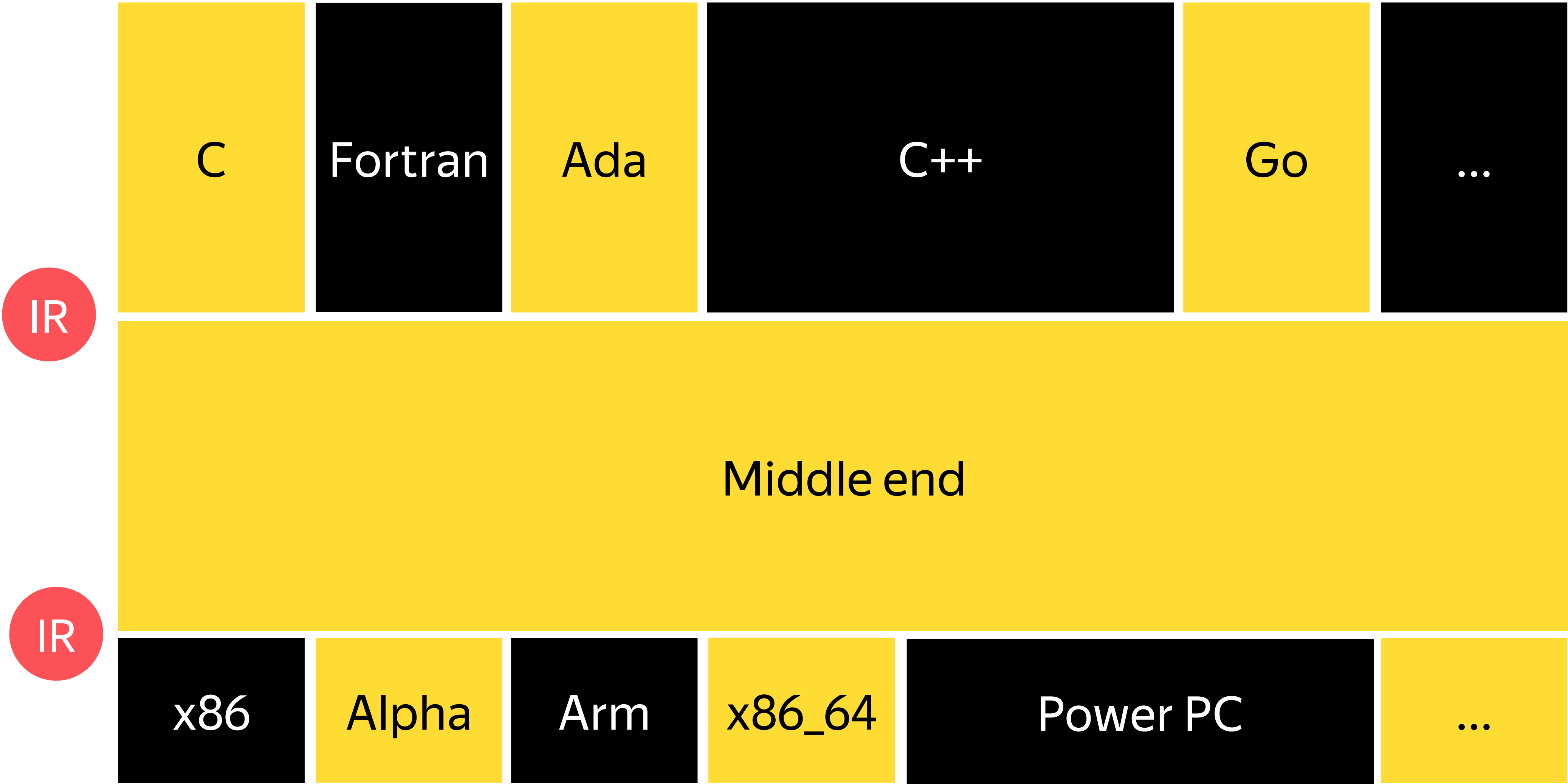
```
no_constexpr():
```

```
    mov eax, 8
```

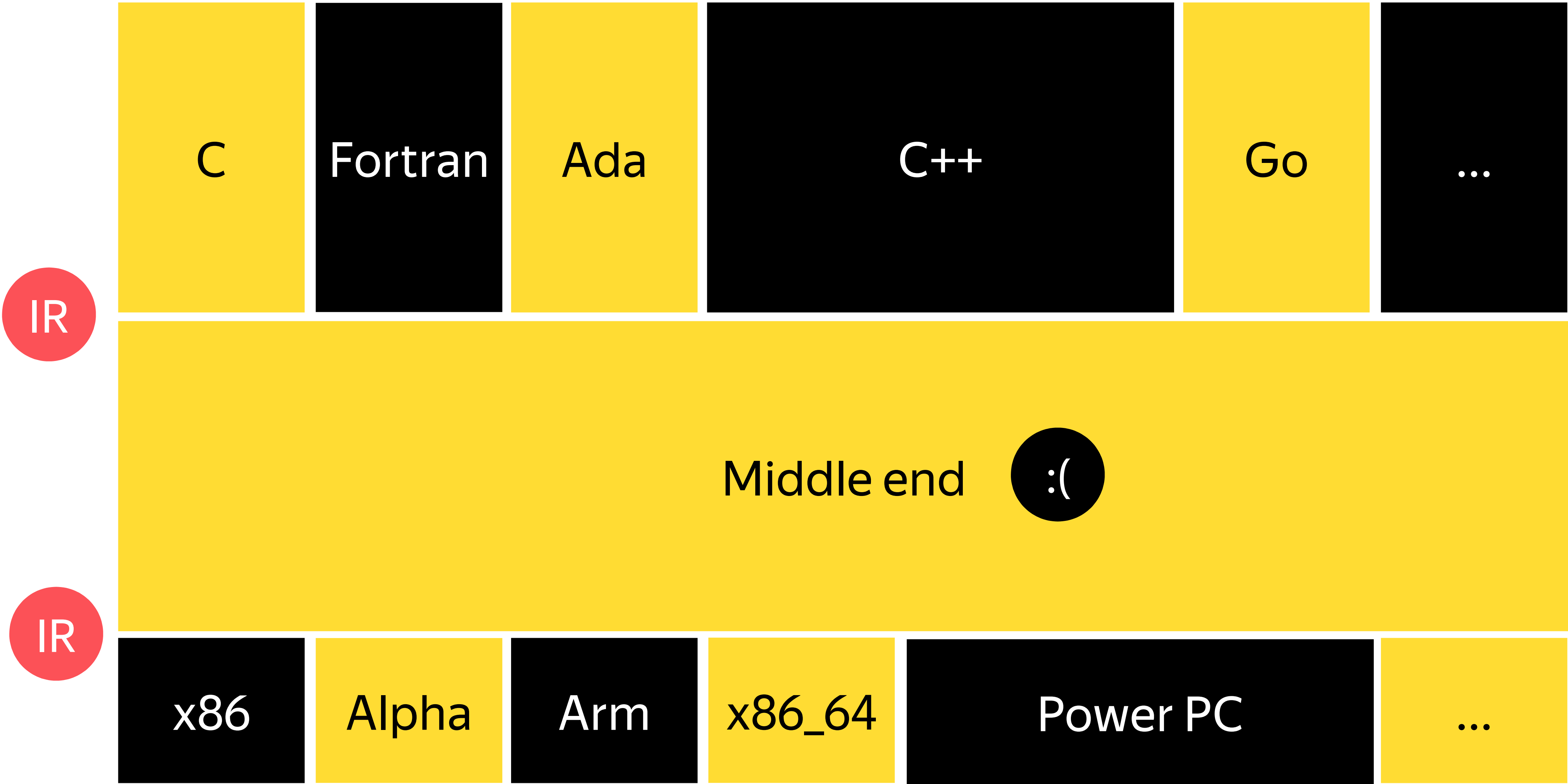
```
    ret
```

# А почему так?

# Анатомия компилятора (упрощённо)



# Анатомия компилятора (без constexpr)





# Старое и незнакомое (2)

```
no_constexpr():
```

```
movabs rax, 30064771075
```

```
mov DWORD PTR [rsp-20], 0
```

```
lea rdx, [rsp-40]
```

```
mov QWORD PTR [rsp-40], rax
```

```
lea rsi, [rdx+28]
```

```
movabs rax, 4294967300
```

```
mov QWORD PTR [rsp-32], rax
```

```
lea rax, [rsp-40]
```

```
mov DWORD PTR [rsp-16], 1
```

```
add rax, 4
```

```
mov DWORD PTR [rsp-24], 8
```

```
mov rcx, rax
```

```
.L2:
```

```
add rax, 4
```

```
cmp rax, rsi
```

```
je .L8
```

```
.L6:
```

```
mov edi, DWORD PTR [rax]
```

```
mov r8d, DWORD PTR [rdx]
```

```
cmp edi, r8d
```

```
jge .L2
```

```
mov DWORD PTR [rax], r8d
```

```
add rax, 4
```

```
mov DWORD PTR [rdx], edi
```

```
cmp rax, rsi
```

```
jne .L6
```

```
.L8:
```

```
lea rax, [rcx+4]
```

```
cmp rax, rsi
```

```
je .L3
```

```
mov rdx, rcx
```

```
mov rcx, rax
```

```
jmp .L6
```

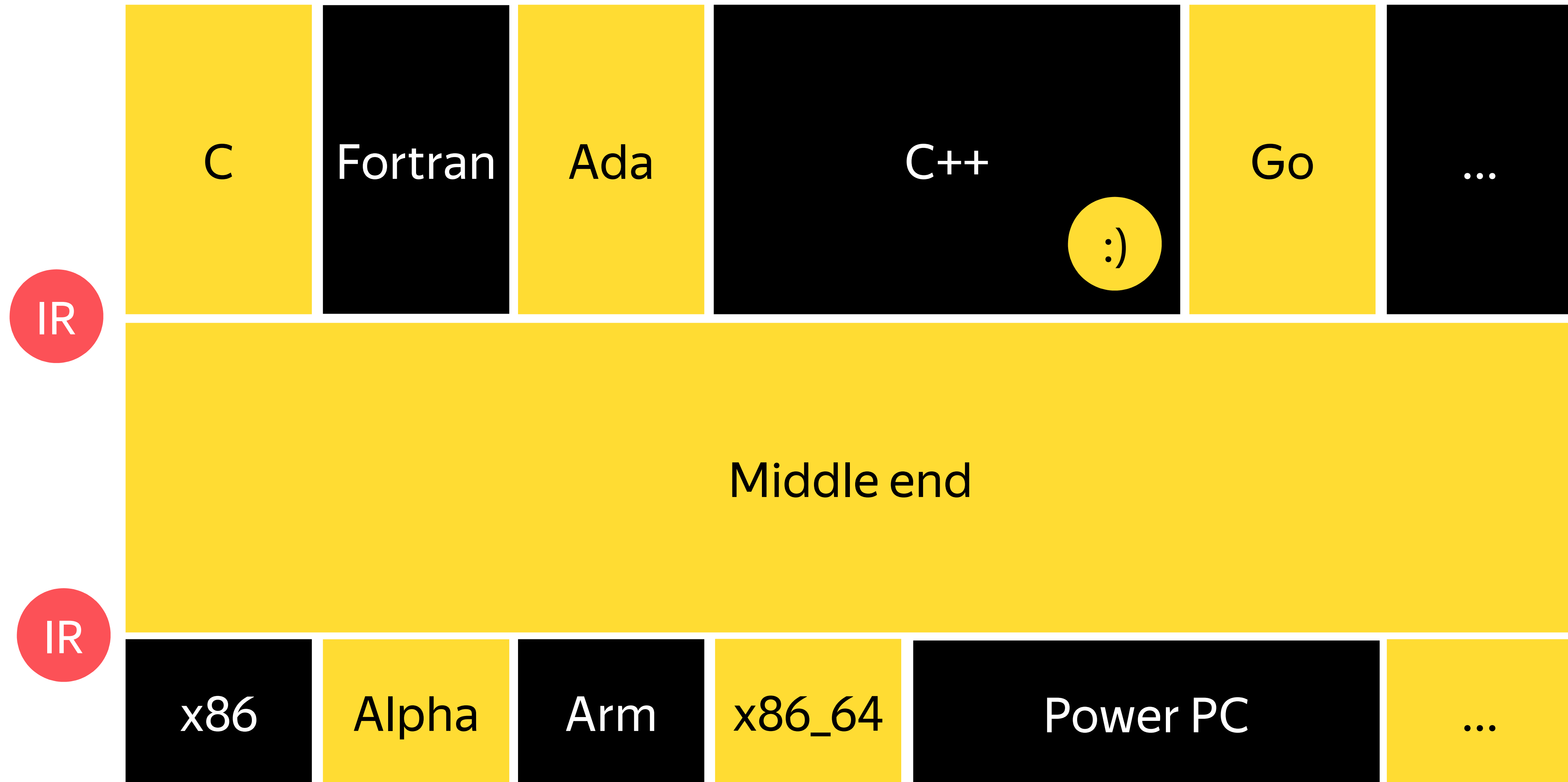
```
.L3:
```

```
mov eax, DWORD PTR [rsp-16]
```

```
add eax, DWORD PTR [rsp-40]
```

```
ret
```

# Анатомия компилятора (constexpr)



# Старое и незнакомое (2)

```
no_constexpr():
```

```
    mov eax, 8
```

```
    ret
```

# Старое и незнакомое (3)

# Старое и незнакомое (3)

```
#include <stdexcept>

constexpr int factorial(int n) {
    return n ? n * factorial(n - 1) : 1;
}

int do_something(unsigned n) {
    constexpr auto i = factorial(27);
    // ...
    return i;
}
```

# Старое и незнакомое (3)

```
main.cpp: In function 'int do_something(unsigned int)':  
main.cpp:200:33:   in constexpr expansion of 'factorial(27)'  
main.cpp:197:29:   in constexpr expansion of 'factorial((n + -1))'  
main.cpp:197:29:   in constexpr expansion of 'factorial((n + -1))'  
<...>  
main.cpp:197:29:   in constexpr expansion of 'factorial((n + -1))'  
main.cpp:197:29:   in constexpr expansion of 'factorial((n + -1))'  
main.cpp:197:29:   in constexpr expansion of 'factorial((n + -1))'  
main.cpp:200:36: error: overflow in constant expression [-fpermissive]  
    constexpr auto i = factorial(27);  
                                ^
```

# Итого:

# Constexpr:

- \* Позволяет использовать функции для вычисления констант
- \* «Заставляет» вычислять константы
- \* Позволяет статически инициализировать объекты
- \* Оптимизирует код
- \* Позволяет проверять на UB



# Ближайшее будущее

# constexpr new

А давайте разрешим вызывать new в constexpr функциях, если в этой же constexpr функции память освобождается. Ну и деструкторы разрешим делать constexpr:

```
constexpr auto precompute_series(int n, int i) {  
    std::vector<int> v(n);  
  
    /*...*/  
  
    return v[i];  
}
```

# constexpr new

А давайте будем память выделенную через new в constexpr функциях записывать прям в бинарник и не вызывать деструктор для объекта [\*] (прям как при константной инициализации):

```
constexpr auto precompute_whole_series(int n) {  
    std::vector<int> v(n);  
  
    /*...*/  
  
    return v;  
}
```

[\*] - есть тонкости

# Constexpr new

А давайте будем память выделенную через new в constexpr функциях записывать прям в бинарник и не вызывать деструктор для объекта [\*] (прям как при константной инициализации):

```
constexpr std::string we_need_this = "Something usefull";
```

```
constexpr std::string time_format = "YYYY-MM-DDTHH:MM:SS";
```

# Ближайшее будущее (2)

# Регулярки

Регулярки медленно инициализируются:

```
bool is_valid_mail(std::string_view mail) {  
    // ~260µs  
    static const std::regex mail_regex(R"((?:(?:[<>()\\[\].,;:\s@\""]+  
    (?:\\. [^<>()\\[\].,;:\s@\""]+)*|\\. +\\. )@(?:(?:[<>()\\[\].,;:\s@\""]+\\. )+  
    [^<>()\\[\].,;:\s@\""]{2,})))");  
  
    return std::regex_match(  
        std::cbegin(mail),  
        std::cend(mail),  
        mail_regex  
    );  
}
```

# Constexpr regex

Ой, смотрите, мы же теперь можем сделать constexpr std::regex:

```
bool is_valid_mail(std::string_view mail) {  
    // Проинициализируется на этапе компиляции  
    static constexpr std::regex mail_regex{/*...*/);  
  
    return std::regex_match(  
        std::cbegin(mail),  
        std::cend(mail),  
        mail_regex  
    );  
}
```

# constexpr boyer\_moore\_horspool\_searcher

std::boyer\_moore\_horspool\_searcher — иногда ищет быстро, а вот конструируется медленно.

| -----                    |      |    |        |            |
|--------------------------|------|----|--------|------------|
| Benchmark                | Time |    | CPU    | Iterations |
| -----                    |      |    |        |            |
| BM_bmh_construction      | 118  | ns | 118 ns | 4644219    |
| BM_bmh_construction_huge | 184  | ns | 184 ns | 3779484    |
| BM_bmh_apply             | 34   | ns | 34 ns  | 20578297   |
| BM_bmh_apply_huge        | 73   | ns | 73 ns  | 9311461    |

Можно будет сделать constexpr конструктор, и наслаждаться конструированием на этапе компиляции, а не на рантайме.



# Очень далёкое будущее

# Рефлексия

```
struct User {  
    std::string name;  
    unsigned age;  
  
    [[orm "id"]]  
    unsigned uuid;  
};
```

# Рефлексия

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, Data& d) {
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it);
```

```
    for (auto field: $d.fields()) {
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

```
        db[name] >> field;
```

```
    }
```

```
}
```

Полезный constexpr

# Рефлексия

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, Data& d) { // User
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it);
```

```
    for (auto field: $d.fields()) {
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

```
        db[name] >> field;
```

```
    }
```

```
}
```

Полезный constexpr

# Рефлексия

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, Data& d) { // User
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it); // d.uuid
```

```
    for (auto field: $d.fields()) {
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

```
        db[name] >> field;
```

```
    }
```

```
}
```

Полезный constexpr

# Рефлексия

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, Data& d) { // User
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it); // d.uuid
```

```
    for (auto field: $d.fields()) {
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

```
        db[name] >> field; // «age», «name» => d.age, d.name
```

```
    }
```

```
}
```

Полезный constexpr

# Рефлексия

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, Data& d) {
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it);
```

```
    for (auto field: $d.fields()) { // constexpr std::vector<field>
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

```
        db[name] >> field;
```

```
    }
```

```
}
```

Полезный constexpr

# Рефлексия

```
template <class Db>
void fill_from_db(Db& db, User& d) {
    db.start(d.uuid);
    db["age"] >> d.age;
    db["name"] >> d.name;
}
```



# Рефлексия

```
int main() {  
    Mongo m;  
    User u{.uuid=42};  
    fill_from_db(m, u)  
    std::cout << "User with id = 42 is " << u.name;  
}
```

# Плюшки для рефлексии

# Рефлексия и проблемы

// Синтаксис изменится!

```
template <class Db, class Data>
```

```
void fill_from_db(Db& db, class Data& d) {
```

```
    auto it = $d.fields.name("id") || $d.fields.attributes("orm \"id\").fields()[0];
```

```
    db.start(*it);
```

```
    for (auto field: $d.fields()) { // constexpr std::vector<field>
```

```
        constexpr std::string name {
```

```
            field.attribute("orm") ? field.attribute("orm") : field.name()
```

```
        };
```

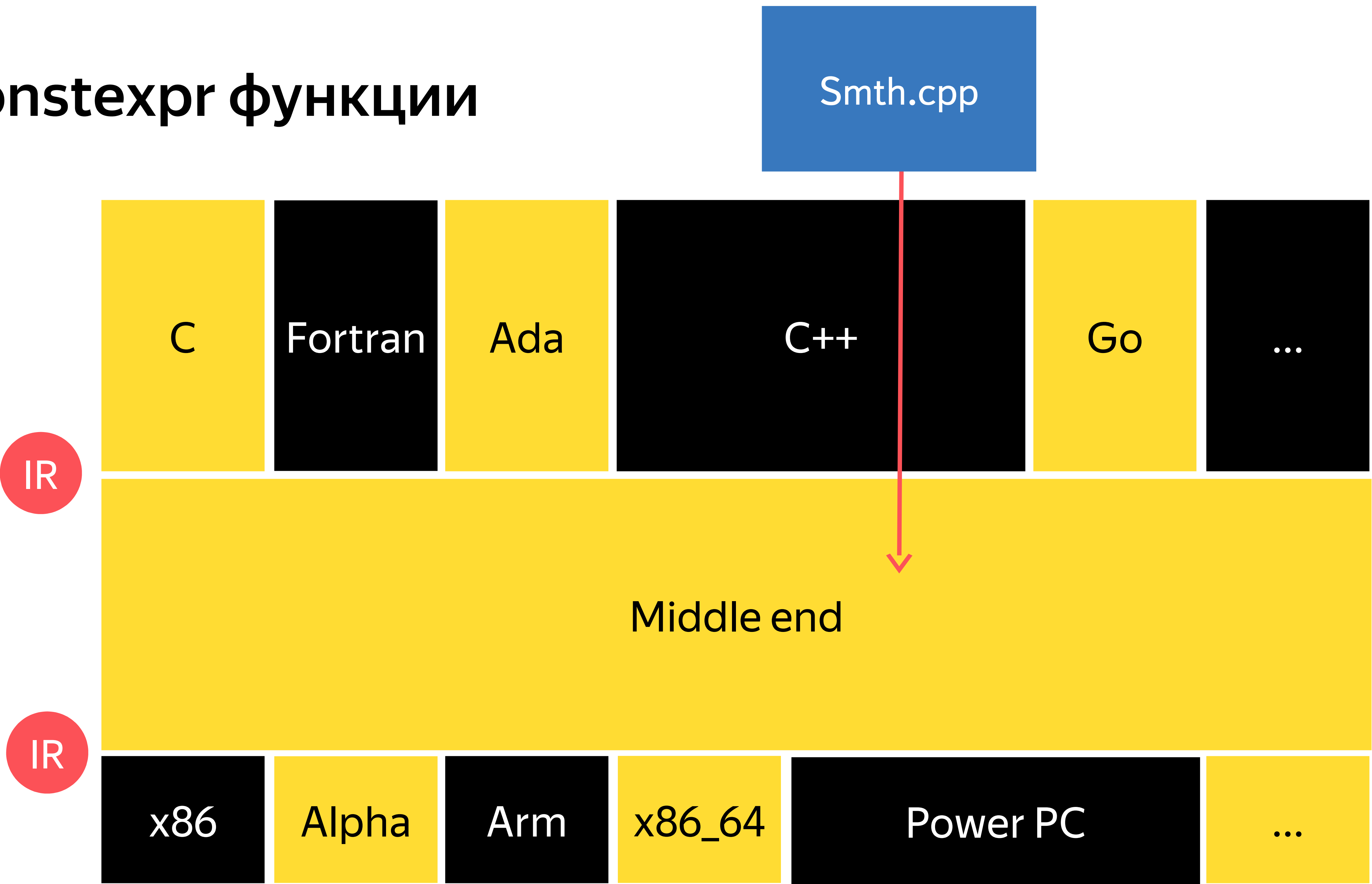
```
        db[name] >> field;
```

```
    }
```

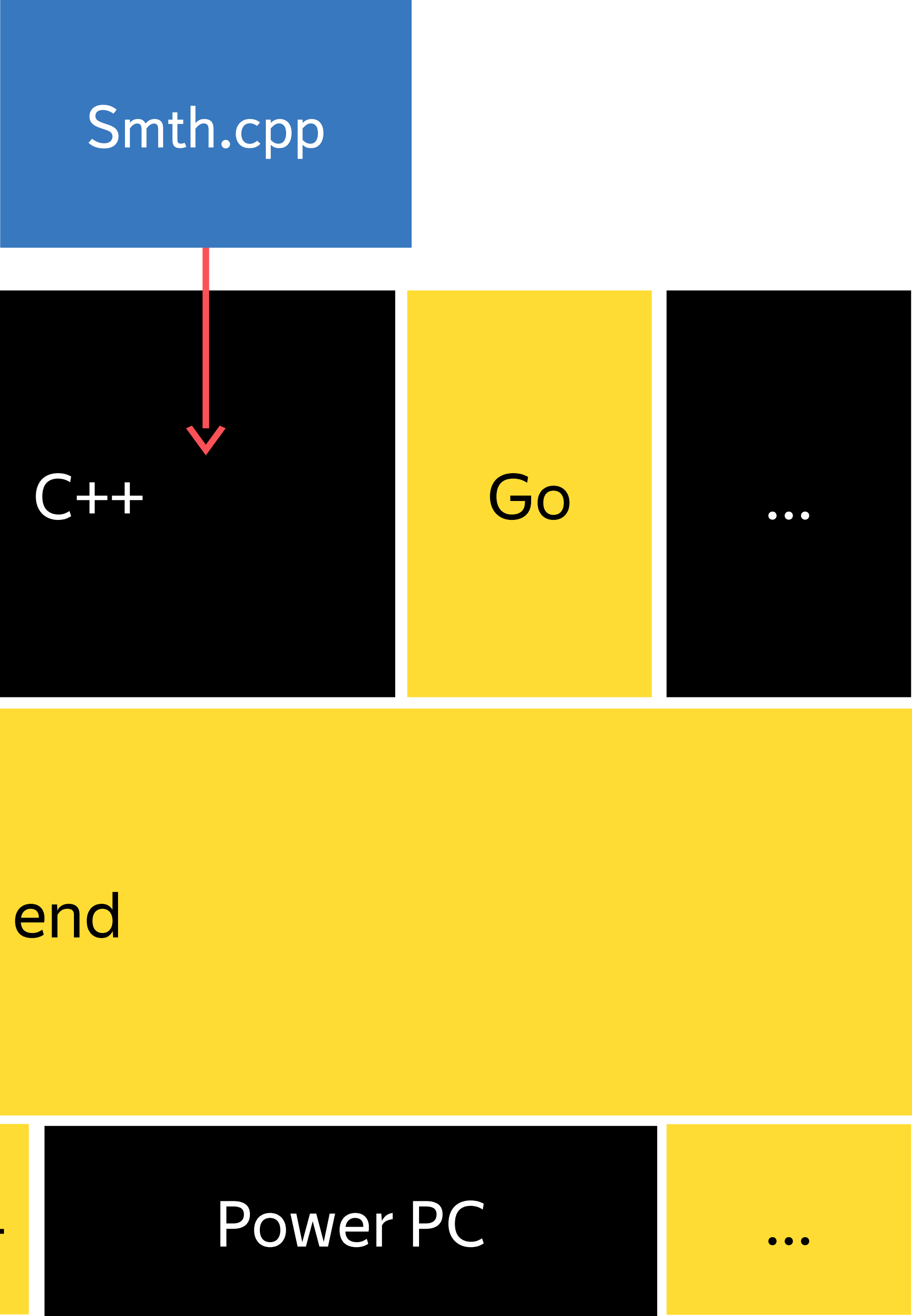
```
}
```

Полезный constexpr

# constexpr функции



# constexpr! функции



# Рефлексирующие классы

# Метаклассы

```
$orm User {  
    std::string name;  
    unsigned age;  
  
    [[orm "id"]]  
    unsigned uuid;  
};
```

# Метаклассы

```
$orm User {  
    std::string name;  
    unsigned age;  
  
    [[orm "id"]]  
    unsigned uuid;  
};
```



# Метаклассы

```
int main() {  
    Mongo m;  
    User u(42, m);  
    std::cout << "User with id = 42 is " << u.name();  
}
```

# Метаклассы

```
$class orm {  
    for (auto field: $this->fields()) {  
        $"auto " + field.name() + "() const { return m_" + field.name() + "; }";  
        $"void set_" + field.name() + "(auto v) { m_" + field.name() + "=std::move(v); }";  
        field.make_private().set_name("m_" + field.name());  
    }  
  
    $"template <class Db>" +  
    $this->name() + $"(unsigned id, Db& db) : m_uuid(id) { " +  
        "fill_from_db(db, *this);" +  
        "}";  
};
```

# Метаклассы

```
class User {  
    std::string m_name;  
    unsigned m_age;  
    unsigned m_uuid;  
public:  
    auto name() const { return m_name; }  
    auto age() const { return m_age; }  
    auto uuid() const { return m_uuid; }  
    // ...  
    template <class Db>  
    User(unsigned id, Db& db) : m_uuid(id) { fill_from_db(db, *this); }  
};
```

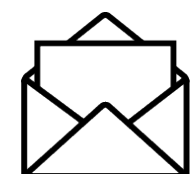
**Спасибо**

# Полухин Антон

Старший разработчик Yandex.Taxi



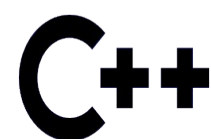
[antoshkka@gmail.com](mailto:antoshkka@gmail.com)



[antoshkka@yandex-team.ru](mailto:antoshkka@yandex-team.ru)



<https://github.com/apolukhin>



РГ21 C++ РОССИЯ

<https://stdcpp.ru/>

