Яндекс Такси

C++14 + C++17

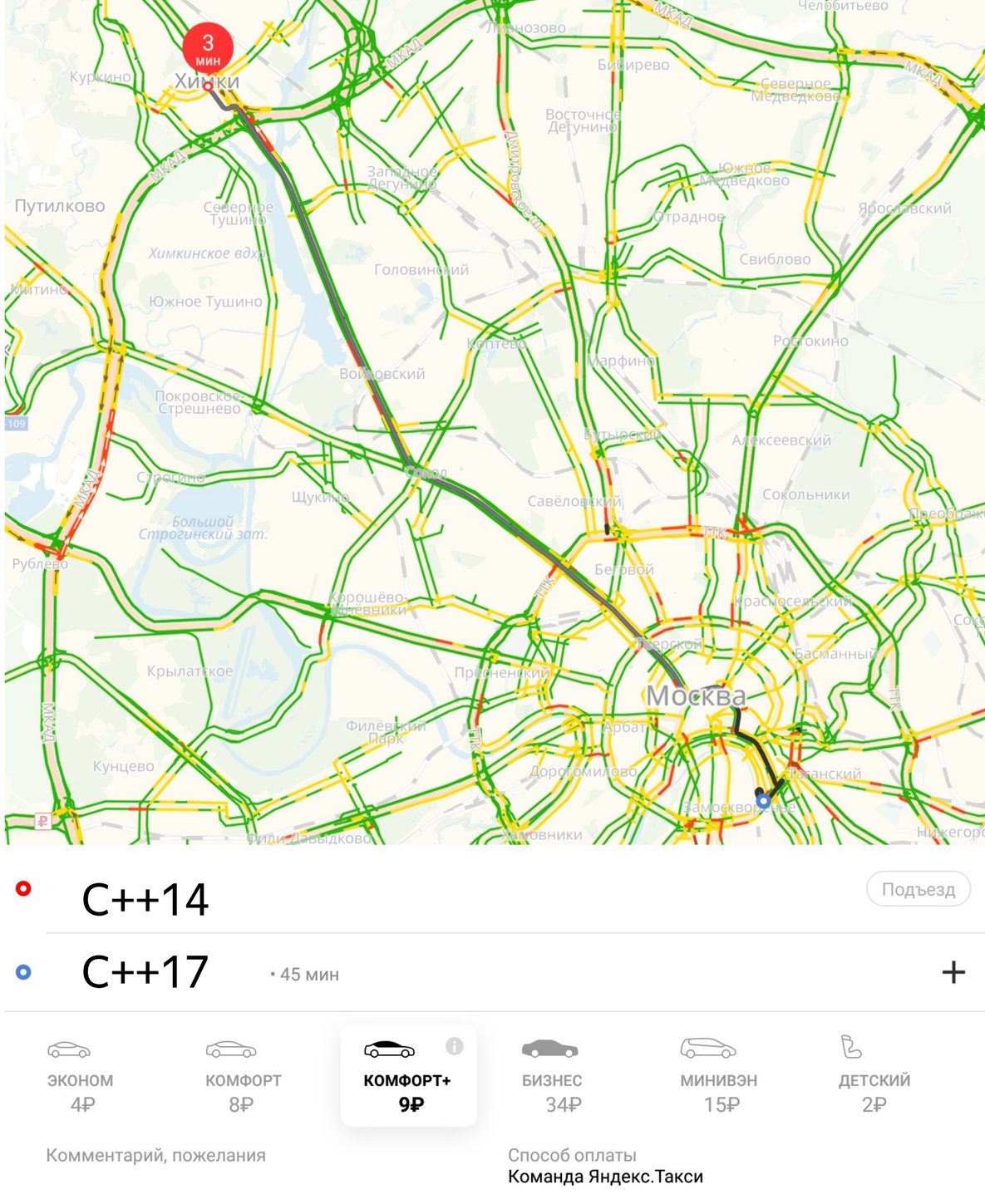
Полухин Антон

Antony Polukhin

Яндекс Такси

Содержание

- Немного про компиляторы
- C++14
  - lambdas
  - Многопоточность
  - Гетерогенный поиск
  - misc
- C++17
  - Core
  - FS
  - Library



Про компиляторы

(или зачем переходить на новые стандарты)

Про компиляторы

```cpp
#include <string>


std::string f() {
  std::string s;
  s = 'd';
  return s;
}
```

g++-8.2 -Wall -O2 -std=c++11

```
f[abi:cxx11]():
push r12
mov r8d, 100
mov ecx, 1
xor edx, edx
push rbp
lea rbp, [rdi+16]
xor esi, esi
push rbx
mov rbx, rdi
mov QWORD PTR [rdi], rbp
mov BYTE PTR [rdi+16], 0
call _M_replace_aux
mov rax, rbx
pop rbx
pop rbp
pop r12
ret
mov r12, rax
jmp .L2
```

```
f[abi:cxx11]() [clone
.cold.11]:
.L2:
mov rdi, QWORD PTR [rbx]
cmp rbp, rdi
je .L3
call operator
delete(void*)
.L3:
mov rdi, r12
call Unwind_Resume
```

g++-8.2 -Wall -O2 -std=c++17

```
f[abi:cxx11]():
lea rdx, [rdi+16]
mov rax, rdi
mov QWORD PTR [rdi+8], 1
mov QWORD PTR [rdi], rdx
mov edx, 100
mov WORD PTR [rdi+16], dx
ret
```

Оптимизации

Новые оптимизации, которые могут поломать очень плохой код, включаются по умолчанию только на новых версиях стандарта.

Оптимизации

Новые оптимизации, которые могут поломать очень плохой код, включаются по умолчанию только на новых версиях стандарта.

Каждая новая версия стандарта языка C++ привносит небольшие правки в ядро языка, позволяющие компиляторам оптимизировать больше, при этом не ломая пользовательский код.

Оптимизации

Новые оптимизации, которые могут поломать очень плохой код, включаются по умолчанию только на новых версиях стандарта.

Каждая новая версия стандарта языка C++ привносит небольшие правки в ядро языка, позволяющие компиляторам оптимизировать больше, при этом не ломая пользовательский код.

В стандартную библиотеку вносят правки для новых стандартов. В том числе и для производительности.

Оптимизации

Новые оптимизации, которые могут поломать очень плохой код, включаются по умолчанию только на новых версиях стандарта.

Каждая новая версия стандарта языка C++ привносит небольшие правки в ядро языка, позволяющие компиляторам оптимизировать больше, при этом не ломая пользовательский код.

В стандартную библиотеку вносят правки для новых стандартов. В том числе и для производительности.

Хотите большую производительность — переходите на новые версии C++.

Про C++14

(опять оптимизации)

Оптимизации C++14

Компилятору разрешили убирать вызовы new/delete

Оптимизации C++14

Компилятору разрешили убирать вызовы new/delete:

```
#include <memory>
int main() {
    auto m = std::make_unique<int>(0);
    return *m;
}
```

Оптимизации C++14

Компилятору разрешили убирать вызовы new/delete:

```
#include <memory>
int main() {
    auto m = std::make_unique<int>(0);
    return *m;
}

main: # @main on Clang
  xor eax, eax
  ret
```

Оптимизации C++14

Компилятору разрешили сложные функции помечать как **constexpr**.

Про constexpr

(и что он делает ещё с C++11)

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 незнакомое

```
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.

constexpr

```
// 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); }
```

constexpr

```
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
```

constexpr

```
// 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); }
```

constexpr

```
no_constexpr():
  mov eax, 8
  ret
```

Про C++14

(стандартная библиотека и не только)

C++14

```
#include <memory>

template <class F> auto call(F callback) { /* ... */ }

struct Data { void do_something(int i); /* ... */};


int main() {
    auto data = std::make_unique<Data>();
    call(
        [d = std::move(data)](auto v) {
            d->do_something(v);
        }
    );
}
```

C++14

```
#include <memory>
template <class F> auto call(F callback) { /* ... */ }
struct Data { void do_something(int i); /* ... */};


int main() {
    auto data = std::make_unique<Data>();
    call(
        [d = std::move(data)](auto v) {
            d->do_something(v);
        }
    );
}
```

C++14

```
#include <memory>

template <class F> auto call(F callback) { /* ... */ }

struct Data { void do_something(int i); /* ... */};


int main() {
    auto data = std::make_unique<Data>();
    call(
        [d = std::move(data)](auto v) {
            d->do_something(v);
        }
    );
}
```

C++14

```
#include <memory>

template <class F> auto call(F callback) { /* ... */ }

struct Data { void do_something(int i); /* ... */};


int main() {
    auto data = std::make_unique<Data>();
    call(
        [d = std::move(data)](auto v) {
            d->do_something(v);
        }
    );
}
```

C++14

```
#include <memory>

template <class F> auto call(F callback) { /* ... */ }

struct Data { void do_something(int i); /* ... */};


int main() {
    auto data = std::make_unique<Data>();
    call(
        [d = std::move(data)](auto v) { // unique_ptr<Data> d = std::move(data);
            d->do_something(v);
        }
    );
}
```

Про C++14
(многопоточность)

C++14 и многопоточность

```
#include <mutex>

std::mutex data_mutex;

std::string data;


std::string read() {
    std::unique_lock<std::mutex> r{data_mutex};
    return data;
}
void set(std::string new_data) {
    std::unique_lock<std::mutex> w{data_mutex};
    data = std::move(new_data);
}
```

C++14 и многопоточность

```
#include <mutex>

std::mutex data_mutex;

std::string data;


std::string read() { // Вызывается крайне часто!
    std::unique_lock<std::mutex> r{data_mutex};
    return data;
}
void set(std::string new_data) { // Вызывается крайне редко!
    std::unique_lock<std::mutex> w{data_mutex};
    data = std::move(new_data);
}
```

C++14 и многопоточность

```
#include <shared_mutex>

std::shared_timed_mutex data_mutex;

std::string data;


std::string read() {
    std::shared_lock<std::shared_timed_mutex> r{data_mutex};
    return data;
}
void set(std::string new_data) {
    std::unique_lock<std::shared_timed_mutex> w{data_mutex};
    data = std::move(new_data);
}
```

C++14 и многопоточность

```
#include <shared_mutex>

std::shared_timed_mutex data_mutex;

std::string data;


std::string read() {
    std::shared_lock<std::shared_timed_mutex> r{data_mutex};
    return data;
}
void set(std::string new_data) {
    std::unique_lock<std::shared_timed_mutex> w{data_mutex};
    data = std::move(new_data);
}
```

C++14 и многопоточность

```
#include <shared_mutex>

std::shared_timed_mutex data_mutex;

std::string data;


std::string read() {
    std::shared_lock<std::shared_timed_mutex> r{data_mutex};
    return data; // Многие потоки вызывают одновременно
}
void set(std::string new_data) {
    std::unique_lock<std::shared_timed_mutex> w{data_mutex};
    data = std::move(new_data);
}
```

C++14 и многопоточность

```
#include <shared_mutex>

std::shared_timed_mutex data_mutex;

std::string data;


std::string read() {
    std::shared_lock<std::shared_timed_mutex> r{data_mutex};
    return data;
}
void set(std::string new_data) {
    std::unique_lock<std::shared_timed_mutex> w{data_mutex};
    data = std::move(new_data); // Все читатели ждут, 1 поток меняет данные
}
```

Shared mutex

Лучше не использовать **без бенчмарков**!

Про C++14
(контейнеры)

Гетерогенный поиск

```
template <class Key, class Value, class Comparator>
class map {
    // ...
public:
    using iterator = ...;
    using const_iterator = ...;

    iterator find(const Key& key);
    const_iterator find(const Key& key) const;
};
```

Гетерогенный поиск

```
std::map<std::string, int> key_value;
key_value["Word"] = 42;
// ...
if (key_value.count("Word")) {          //  :(
    // do something
}
auto it = key_value.find("Word");       //  :(
if (it != key_value.end()) {
    auto& v = *it;
    // do something
}
```

Гетерогенный поиск

```
template <class Key, class Value, class Comparator>
class map {
    // ...
public:
    using iterator = ...;
    using const_iterator = ...;

    iterator find(const Key& key);
    const_iterator find(const Key& key) const;
};
```

Гетерогенный поиск

```
std::map<std::string, int, std::less<>> key_value_het;
```

Гетерогенный поиск

```
std::map<std::string, int, std::less<>> key_value_het;


template <class Key, class Value>
using het_map = std::map<Key, Value, std::less<>>;
```

Гетерогенный поиск

```
het_map<std::string, int> key_value;
key_value["Word"] = 42;
// ...
if (key_value.count("Word")) {
    // do something
}
auto it = key_value.find("Word");
if (it != key_value.end()) {
    auto& v = *it;
    // do something
}
```

Про C++14

(мелочи)

Мелочи C++14

- std::cbegin/std::cend
- std::rbegin/std::rend и std::crbegin/std::crend

Мелочи C++14

- std::cbegin/std::cend
- std::rbegin/std::rend и std::crbegin/std::crend

```
int d[1024] = { /* … */ };

auto it = std::find(std::crbegin(d), std::crend(d), 42);
```

Мелочи C++14

- std::cbegin/std::cend
- std::rbegin/std::rend и std::crbegin/std::crend
- [[deprecated]] и [[deprecated("сообщение")]]
- 10s 24h
- std::quoted

Мелочи C++14

- std::cbegin/std::cend
- std::rbegin/std::rend и std::crbegin/std::crend
- [[deprecated]] и [[deprecated("сообщение")]]
- 10s 24h
- std::quoted

```
ss << std::quoted(in);

ss >> std::quoted(out);
```

Мелочи C++14

- std::cbegin/std::cend
- std::rbegin/std::rend и std::crbegin/std::crend
- [[deprecated]] и [[deprecated("сообщение")]]
- 10s 24h
- std::quoted
- template variables
- std::get<type>(tuple)

Про C++17

(снова оптимизации)

Оптимизация

Guaranteed copy elision!

C++17

```
#include <mutex>

template <class F> void call(F callback) { /* ... */ }

int main() {
    call(
        [m = std::mutex{}]() mutable {
            std::unique_lock<std::mutex> l{m};
            /* ... */
        }
    );
}
```

C++17

```
#include <mutex>


template <class F> void call(F callback) { /* ... */ }


int main() {
    call(
        [m = std::mutex{}]() mutable {
            std::unique_lock<std::mutex> l{m};
            /* ... */
        }
    );
}
```

Про C++17
(filesystem)

Filesystem

```cpp
#include <filesystem>
#include <iostream>

int main() {
    std::filesystem::directory_iterator it("./");
    std::filesystem::directory_iterator end;

    for (; it != end; ++it) {
        std::filesystem::file_status fs = it->status();
        // ...
```

Filesystem

```
std::filesystem::file_status fs = it->status();
switch (fs.type()) {
case std::filesystem::file_type::regular:
    std::cout << "FILE       ";
    break;
case std::filesystem::file_type::symlink:
    std::cout << "SYMLINK    ";
    break;
case std::filesystem::file_type::directory:
    std::cout << "DIRECTORY  ";
    break;
```

Filesystem

```
        if (fs.permissions() & std::filesystem::owner_write) {
            std::cout << "W ";
        } else {
            std::cout << "  ";
        }

        std::cout << it->path() << '\n';
    } /*for*/
} /*main*/
```

Filesystem

```
using namespace std::filesystem;


path read_symlink(const path& p);

path read_symlink(const path& p, std::error_code& ec);
```

Filesystem

```
using namespace std::filesystem;


path read_symlink(const path& p);
path read_symlink(const path& p, std::error_code& ec);
```

Filesystem

```
using namespace std::filesystem;


path read_symlink(const path& p);
path read_symlink(const path& p, std::error_code& ec);
```

Про C++17
(string_view)

string_view

```
// C++14
void foo(const std::string& value);
```

string_view

```
// C++14
void foo(const std::string& value);
void foo(const char* value);
```

string_view

```cpp
// C++14
void foo(const std::string& value);
void foo(const char* value);
void foo(const char* value, std::size_t length);
```

string_view

```cpp
// C++14
void foo(const std::string& value);
void foo(const char* value);
void foo(const char* value, std::size_t length);

// C++17
void foo(std::string_view value);
```

string_view

```
// C++14
template <class CharT>
void foo(const std::basic_string<CharT>& value);

template <class CharT>
void foo(const CharT* value);

template <class CharT>
void foo(const CharT* value, std::size_t length);
```

string_view

```cpp
// C++17
template <class CharT>
void foo(std::basic_string_view<CharT> value);
```

Про C++17

(to_chars / from_chars)

std::to_chars и std::from_chars

```cpp
#include <sstream>                  //   :-(


template <class T>
T to_number_14(const std::string& s) {
    T res{};
    std::ostringstream oss(s); //   :-(
    oss >> res;
    return res;
}
```

std::to_chars и std::from_chars

```
template<typename _Facet>
locale::locale(const locale& __other, _Facet* __f) {
    _M_impl = new _Impl(*__other._M_impl, 1);
    __try { _M_impl->_M_install_facet(&_Facet::id, __f); }
    __catch(...) {
        _M_impl->_M_remove_reference();
        __throw_exception_again;
      }
    delete [] _M_impl->_M_names[0];
    _M_impl->_M_names[0] = 0;    // Unnamed.
}
```

std::to_chars и std::from_chars

```
#include <charconv>


template <class T>
T to_number_17(const std::string& s) {
    T res{};
    std::from_chars(s.data(), s.data() + s.size(), res); //   :-)
    return res;
}
```

Про C++17

(Склеивание ассоциативных контейнеров)

Склеивание ассоциативных контейнеров

```
struct user {
    std::string                     bio;
    std::string                     address;
    std::vector<unsigned char>      photo;
    std::array<unsigned char, 128>  key;

    // ...
};
```

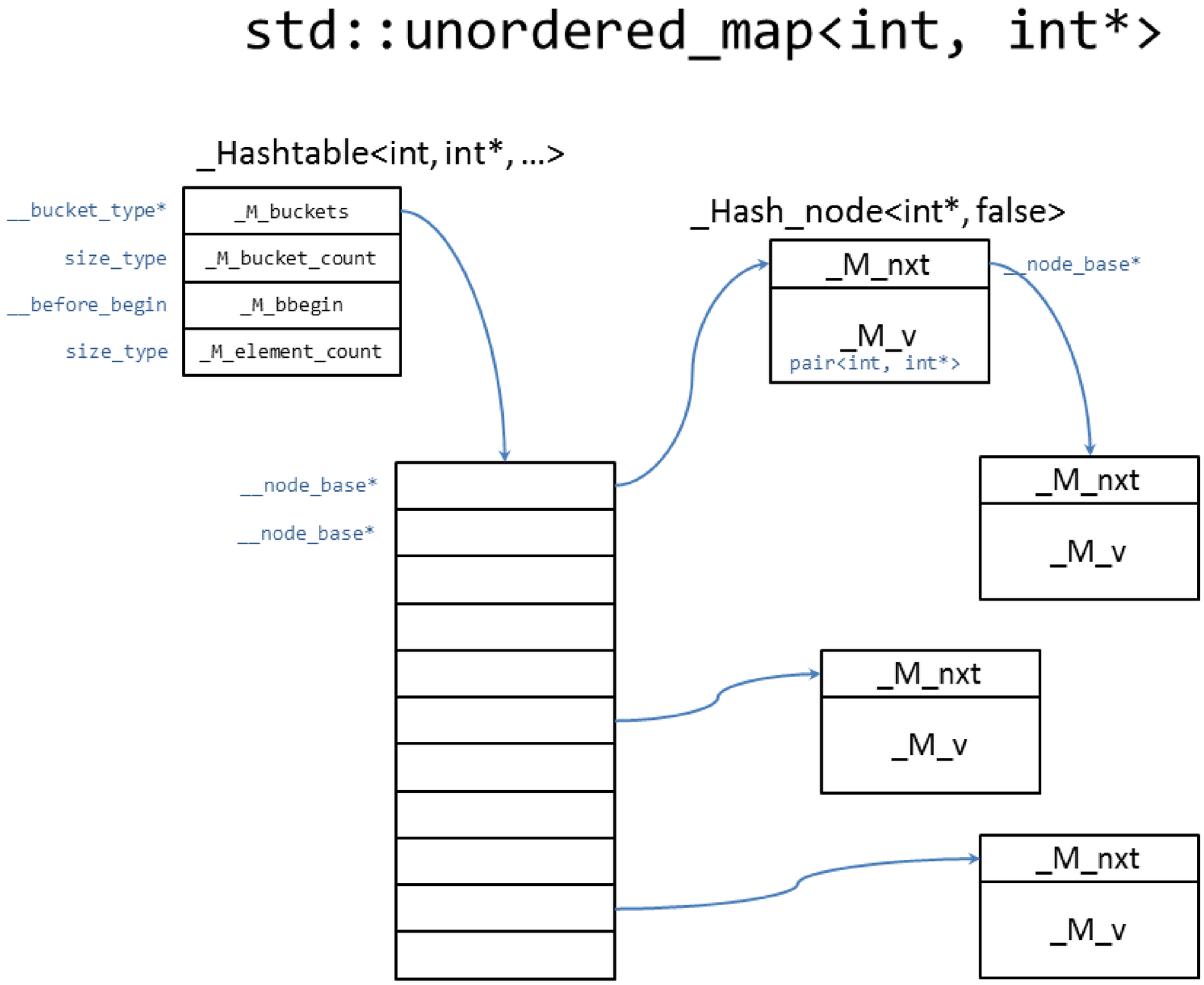
Склеивание ассоциативных контейнеров

```
class user_registry {
    std::unordered_map<std::string, user> data_;

public:
    void update(const std::string& old_name, std::string new_name);
};
```

Склеивание ассоциативных контейнеров

```
// C++14
void user_registry::update(const std::string& old_name, std::string new_name) {
    auto it = data_.find(old_name);
    if (it == data_.cend())
        return;

    user user_copy = std::move(it->second); // :(
    data_.erase(it);
    data_.emplace(std::move(new_name), std::move(user_copy));
}
```

Unordered



C++14 + C++17

Склеивание ассоциативных контейнеров

```
// C++17
void user_registry::update(const std::string& old_name, std::string new_name) {
    auto node = data_.extract(old_name);
    if (!node)
        return;

    node.key() = std::move(new_name);
    data_.insert(std::move(node));
}
```

Про C++17
(мелочи)

Мелочи C++17

- memory_order_consume
- std::function's allocators
- iterator/get_temporary_buffer/is_literal_type
- template <auto V> struct ...
- std::any
- std::variant
- std::optional
- [*this](){ /* ... */ }
- Math special functions
- Inline variables
- namespace foo::bar::example { /* ... */ }
- Template variables
- if constexpr
- constexpr lambda

Мелочи C++17

- constexpr lambda
- if/switch с инициализацией
- Structured bindings
- MT алгоритмы

Спасибо

Полухин Антон

Старший разработчик Yandex.Taxi

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

https://stdcpp.ru/

РГ21 C++ РОССИЯ