Яндекс Такси

WG21 San Diego

Обзор встречи

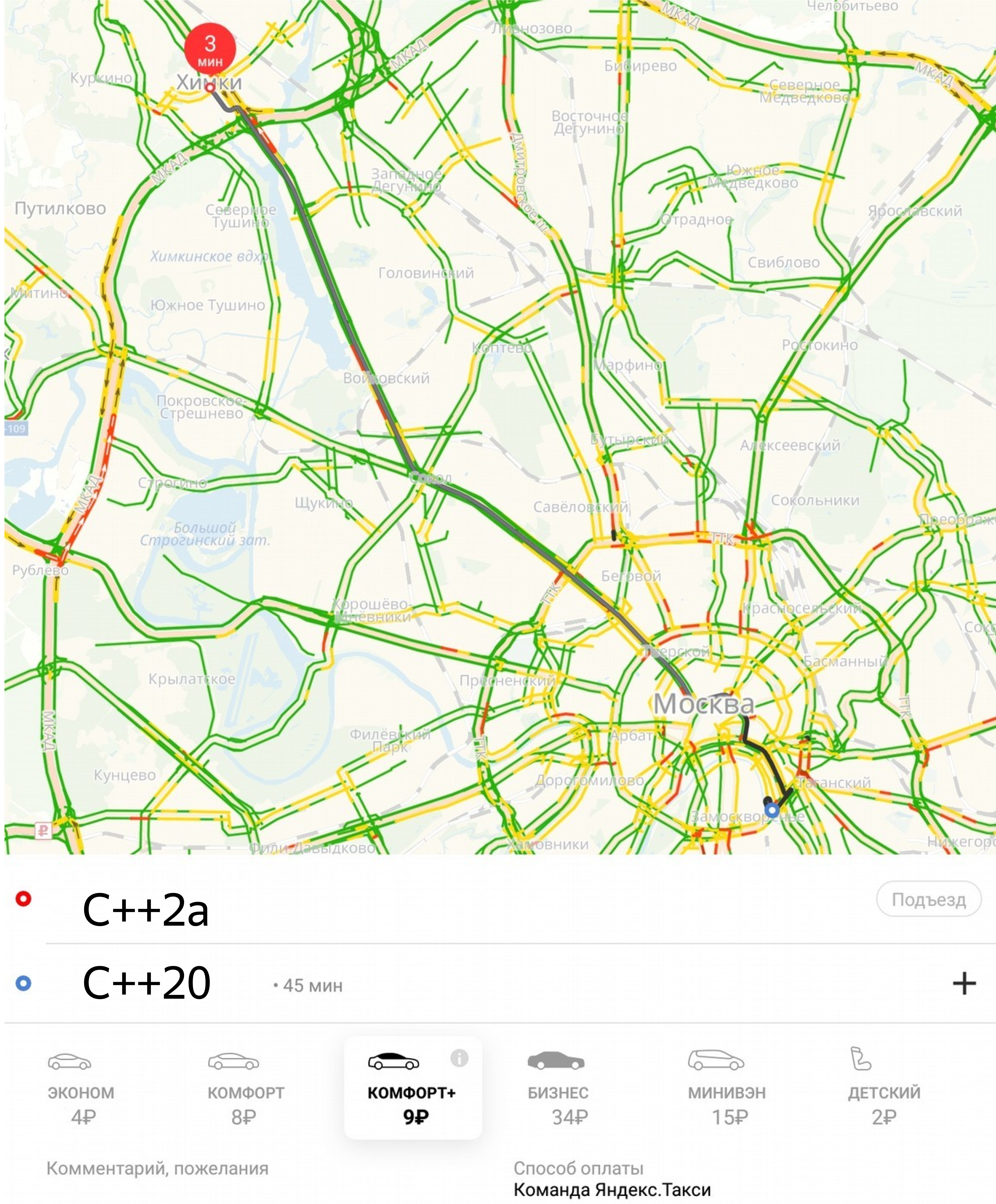
Полухин Антон

Antony Polukhin

Яндекс Такси

Содержание

- Ranges
- ~~Modules~~
- char8_t
- РГ 21



C++2a — Подъезд

C++20 • 45 мин +

ЭКОНОМ	КОМФОРТ	КОМФОРТ+	БИЗНЕС	МИНИВЭН	ДЕТСКИЙ
4₽	8₽	9₽	34₽	15₽	2₽

Комментарий, пожелания

Способ оплаты
Команда Яндекс.Такси

Ranges

Введение в Ranges

```
// <algorithm>
namespace std {

template <class InputIterator, class T>
constexpr InputIterator find(InputIterator first, InputIterator last,
                             const T& value);

}  // namespace std
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges

const char* char_ptr = "....";
auto it = std::ranges::find(char_ptr, std::unreachable_sentinel, '.');
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges

const char* char_ptr = "....";
auto it = std::ranges::find(char_ptr, value_sentinel{'\0'}, '.');
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

}  // namespace std::ranges

std::unordered_map<int, std::string> map = {....};
auto it = std::ranges::find(map.cbegin(), map.cend(), "Hello"sv,
                            [](const auto& v) -> std::string_view { return v.second; });
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputIterator I, Sentinel<I> S, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<I, Proj>, const T*>
constexpr I find(I first, S last, const T& value, Proj proj = {});

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
const char data[] = "....";
auto it = std::ranges::find(data, '.');
```

Введение в Ranges

```
// <algorithm>
namespace std::ranges {

template <InputRange R, class T, class Proj = identity>
  requires IndirectRelation<ranges::equal_to<>, projected<iterator_t<R>, Proj>, const T*>
constexpr safe_iterator_t<R> find(R&& r, const T& value, Proj proj = {});

}  // namespace std::ranges
std::unordered_map<int, std::string> map = {....};
auto it = std::ranges::find(map, "Hello"sv,
                            [](const auto& v) -> std::string_view { return v.second; });
```

Ranges
Часть 2

Views

```
// <ranges>
namespace std::view {

inline constexpr unspecified transform = unspecified;
inline constexpr unspecified filter = unspecified;
inline constexpr unspecified join = unspecified;
inline constexpr unspecified split = unspecified;
inline constexpr unspecified iota = unspecified;
inline constexpr unspecified reverse = unspecified;
inline constexpr unspecified counted = unspecified;

} // namespace std::view
```

Views

```
#include <ranges>


std::string str = "abcd";
```

Views

```
#include <ranges>

std::string str = "abcd";
for (auto c : std::view::reverse(str)) {
  std::cout << c;
}
```

Views

```
#include <ranges>

std::string str = "abcd";
for (auto c : std::view::reverse(str)) {
  std::cout << c;
}

std::ranges::copy(std::view::reverse(str), std::ostream_iterator<char>(std::cout));
```

Views

```
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";

for (auto word : std::view::split(str, ' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '\n';
}
```

Views

```
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";

for (auto word : std::view::split(str, ' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '\n';
}
// "Ranges!\nAre!\nAwesome!\n"
```

Views

```
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";

for (auto word : str | std::view::split(' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '\n';
}
```

Views

```
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";

for (auto word : str | std::view::split(' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '\n';
}
// "Ranges!\nAre!\nAwesome!\n"
```

Views

```
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";

constexpr auto f = [](char c) { return c != '!'; };

for (auto word : str | std::view::filter(f) | std::view::split(' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '\n';
}
// "Ranges\nAre\nAwesome\n"
```

Views

```cpp
#include <ranges>

std::string_view str = "Ranges! Are! Awesome!";
constexpr auto f = [](char c) { return c != '!'; };
constexpr auto t = [](char c) { return std::tolower(c); };
using namespace v = std::view;

for (auto word : str | v::filter(f) | v::transform(t) | v::split(' ')) {
  std::ranges::copy(word, std::ostream_iterator<char>(std::cout));
  std::cout << '|';
}
// "ranges|are|awesome|"
```

Views

```
#include <ranges>
#include <algorithm>
#include <cctype>
template <class T> bool is_palindrome(const T& str) {
  using namespace v = std::view;
  auto f = str | v::filter([](int x) { return std::isalpha(x); })
      | v::transform([](auto x) { return std::tolower(x); });

  return std::ranges::equal(f, v::reverse(f));
}

assert(is_palindrome("Madam, I'm Adam"));
```

Terse syntax

Views

```
#include <ranges>
#include <algorithm>
#include <cctype>

template <class T>
    requires BidirectionalRange<T>
bool is_palindrome(const T& str);
```

Views

```
#include <ranges>
#include <algorithm>
#include <cctype>

bool is_palindrome(const BidirectionalRange auto& str);
```

Modules

~~Modules~~

char8_t

char8_t

```
char8_t hello0[] = u8"Из РГ21 с лбовью!";
```

char8_t

```
char8_t hello0[] = u8"Из РГ21 с лбовью!";

std::u8string hello = u8"Из РГ21 с лбовью!";
```

char8_t

```
char8_t hello0[] = u8"Из РГ21 с лбовью!";
std::u8string hello = u8"Из РГ21 с лбовью!";


// std::cout << hello; // fail
```

char8_t

```
void do_something(unsigned char* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}

void do_something(char8_t* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}
```

char8_t

```
void do_something(unsigned char* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}

void do_something(char8_t* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}
```

char8_t

A Left: x86-64 clang (trunk) -O3 -s... Right: x86-64 clang (trunk) -O3 -s... #2)

```
- do_something(unsigned char*, int
  movzx eax, byte ptr [rdi]
- mov ecx, dword ptr [rsi]
- lea edx, [rax + rcx]
- lea eax, [rax + rcx - 48]
- mov dword ptr [rsi], eax
- movzx eax, byte ptr [rdi + 1]
- lea eax, [rax + rdx]
  add eax, -96
  mov dword ptr [rsi], eax
  ret
```

```
+ _Z12do_somethingPDuRi: # @_Z12do
  movzx eax, byte ptr [rdi]
+ add eax, dword ptr [rsi]
+ movzx ecx, byte ptr [rdi + 1]
+ lea eax, [rcx + rax]
  add eax, -96
  mov dword ptr [rsi], eax
  ret
```

char8_t

```
void do_something(unsigned char* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}

void do_something(char8_t* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}
```

char8_t

```
void do_something(unsigned char* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}

void do_something(char8_t* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}
```

char8_t

```
void do_something(unsigned char* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}

void do_something(char8_t* data, int& result) {
    result += data[0] - u8'0';
    result += data[1] - u8'0';
}
```

char8_t

A
Left:
x86-64 clang (trunk) -O3 -s...
Right:
x86-64 clang (trunk) -O3 -s...
#2)

```
- do_something(unsigned char*, int
    movzx eax, byte ptr [rdi]
-   mov ecx, dword ptr [rsi]
-   lea edx, [rax + rcx]
-   lea eax, [rax + rcx - 48]
-   mov dword ptr [rsi], eax
-   movzx eax, byte ptr [rdi + 1]
-   lea eax, [rax + rdx]
    add eax, -96
    mov dword ptr [rsi], eax
    ret
```

```
+ _Z12do_somethingPDuRi: # @_Z12dc
    movzx eax, byte ptr [rdi]
+   add eax, dword ptr [rsi]
+   movzx ecx, byte ptr [rdi + 1]
+   lea eax, [rcx + rax]
    add eax, -96
    mov dword ptr [rsi], eax
    ret
```

РГ21

РГ21:

```
* Stacktrace

    std::stacktrace s;

    std::cout << s;
```

РГ21:

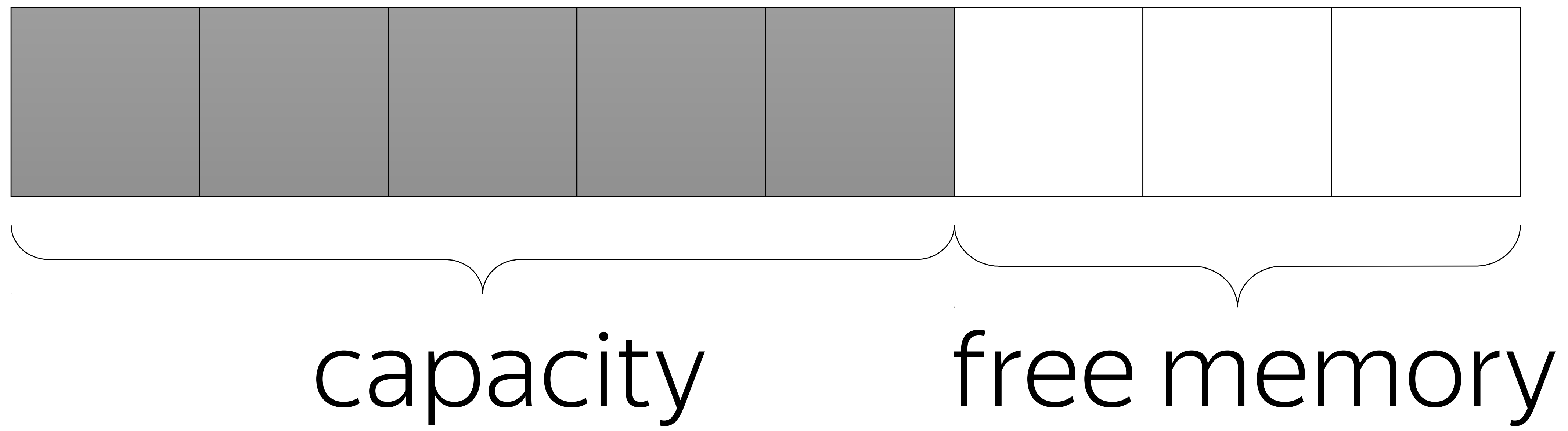
```
* Stacktrace

* Variant comparisons

    std::variant<int, std::string> v;


    v = 42;

    assert(v == 42); // Compile time error!
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

    bool std::allocator_traits<A>::realloc(...)
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)
```



РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)


    template <class T>

    using pool_vector

        = std::vector<T, pool_allocator<T, N>>;
```

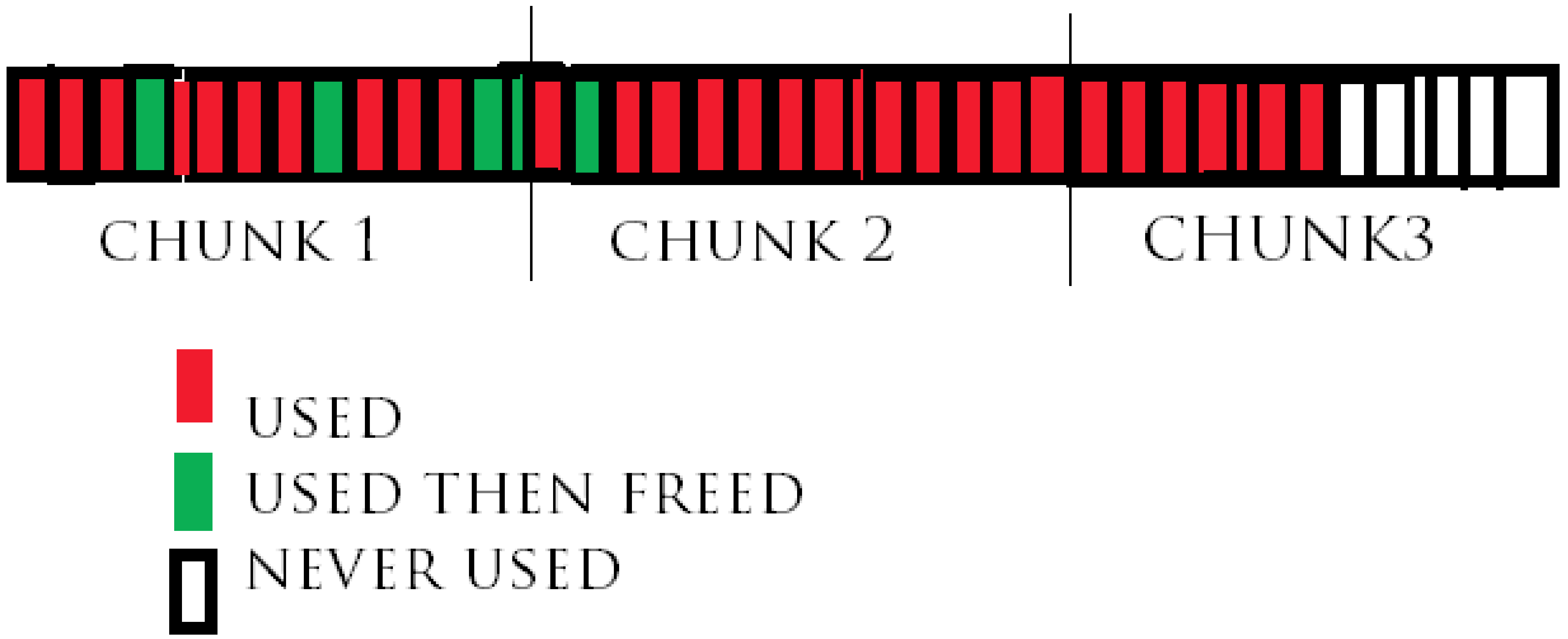
РГ21:

* Stacktrace

* Variant comparisons

* Realloc *(презентовали)*

```
template <class T>
using pool_vector
    = std::vector<T, pool_allocator<T, N>>;
```



РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)
```

РГ21:

* Stacktrace
* Variant comparisons
* Realloc *(презентовали)*
* Concurrent unordered hash map
* Numbers
* [[shared]] *(сопереживали)*
* Plugins

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)

* Plugins

* Constexpr misc
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)

* Plugins

* Constexpr misc

* Ultimate copy elisions
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)

* Plugins

* Constexpr misc

* Ultimate copy elisions

    T produce(); T update(T b); T shrink(T c);

    T d = shrink(update(produce()));
```

РГ21:

```
* Stacktrace

* Variant comparisons

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)

* Plugins

* Constexpr misc

* Ultimate copy elisions

    T produce(); T update(T b); T shrink(T c);

    T d = shrink(update(produce()));
```

Спасибо

Полухин Антон

Старший разработчик Yandex.Taxi

antoshkka@gmail.com

antoshkka@yandex-team.ru

https://github.com/apolukhin

РГ21 C++ РОССИЯ https://stdcpp.ru/