Яндекс Такси

# C++ в 2018

## Успехи года

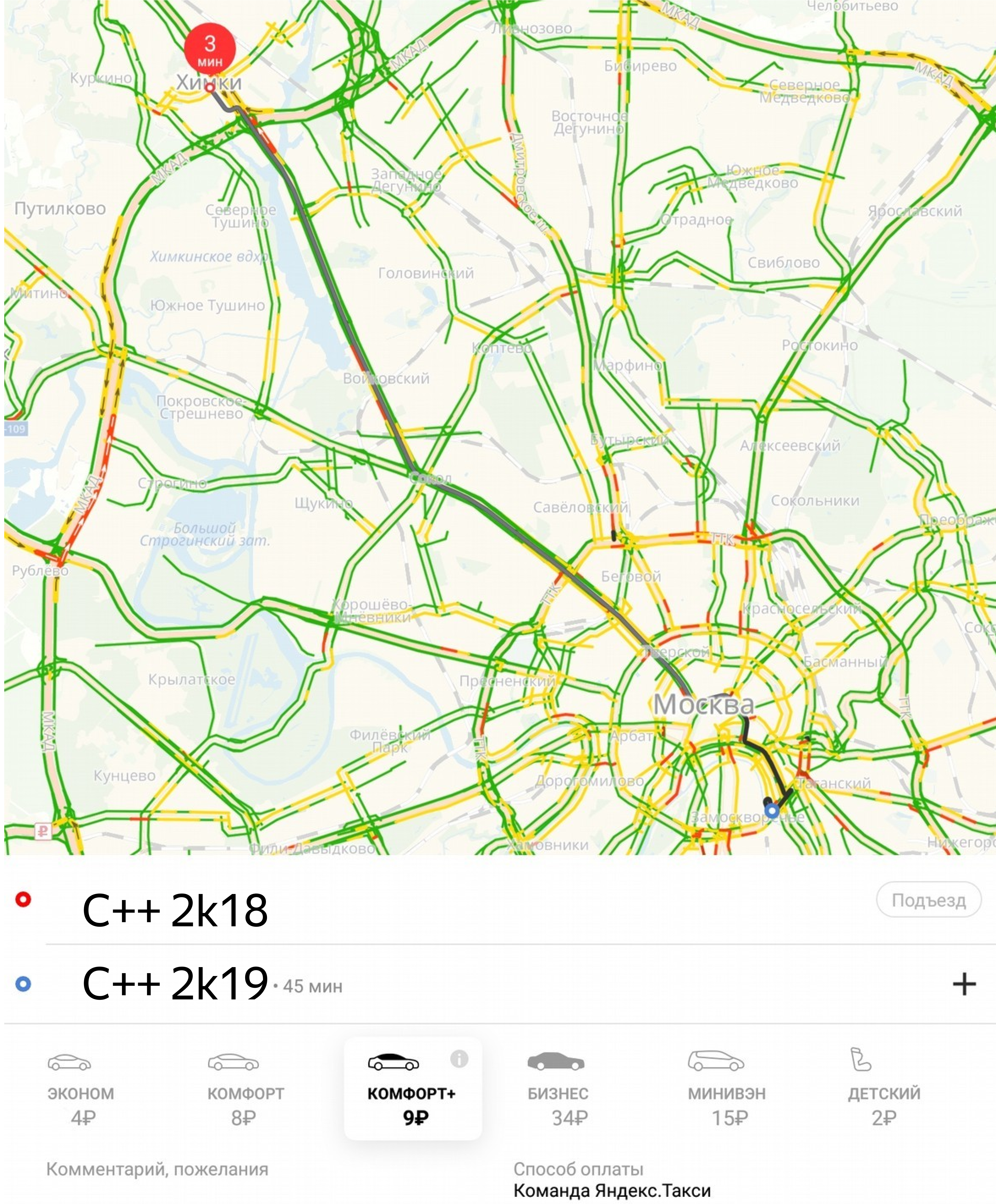
**Полухин Антон**

Antony Polukhin

Яндекс Такси

# Содержание

- Concepts
- Contracts
- Ranges
- ~~Modules~~
- РГ 21



C++ 2k18

C++ 2k19 · 45 мин

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

Комментарий, пожелания

Способ оплаты
Команда Яндекс.Такси

# Concepts

# Concepts

```
template <class T>
void insert_100_elements(T& container) {
  // container.reserve(container.size() + 100);
  assert(!container.empty());

  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Concepts

```cpp
template <Container T>
void insert_100_elements(T& container) {
  // container.reserve(container.size() + 100);
  assert(!container.empty());

  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Concepts

```
void insert_100_elements(Container auto& container) {
  // container.reserve(container.size() + 100);
  assert(!container.empty());

  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Concepts

```
template <class T>
void insert_100_elements(T& container) {
  // container.reserve(container.size() + 100);
  assert(!container.empty());

  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Concepts

```
template <class T>
void insert_100_elements(T& container) {
  // container.reserve(container.size() + 100);
  assert(!container.empty());

  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Concepts

```
template <class T>
void insert_100_elements(T& container) {
  if constexpr (requires{ container.reserve(container.size() + 100); }) {
    container.reserve(container.size() + 100);
  }

  assert(!container.empty());
  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Contracts

# Contracts

```
template <class T>
void insert_100_elements(T& container) {
  if constexpr (requires{ container.reserve(container.size() + 100); }) {
    container.reserve(container.size() + 100);
  }

  assert(!container.empty());
  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Contracts

```
template <class T>
void insert_100_elements(T& container) {
  if constexpr (requires{ container.reserve(container.size() + 100); }) {
    container.reserve(container.size() + 100);
  }

  assert(!container.empty());
  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# Contracts

```
template <class T> void insert_100_elements(T& container)
  [[expects: !container.empty()]]
  [[ensures axiom: container.size() > 100]]
{
  if constexpr (requires{ container.reserve(container.size() + 100); }) {
    container.reserve(container.size() + 100);
  }
  auto v = container.back();
  for (unsigned i = 0; i < 100; ++i) {
    container.insert(container.end(), v + i);
  }
}
```

# 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

```cpp
// <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

```cpp
#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

```
#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"));
```

# Modules

# ~~Modules~~

# РГ21

# РГ21:

```
* Stacktrace

    std::stacktrace s;

    std::cout << s;
```

# РГ21:

* Stacktrace

* Constexpr everything

# РГ21:

```
* Stacktrace

* Constexpr everything (P0639R0)

  * It is simple to add constexpr all around the container
  declaration

  * ...allowing non trivial destructors in constant
  expressions

  * Exceptions could not be thrown in constant expression
  so it seems OK to allow try and catch in constant
  expressions that just do nothing

  * Instead of having a magic constexpr_vector that
  allocates memory, please change it to magic
  constexpr_allocator that allocates memory in constant
  expressions.

  * Instead of placement new use
  constructor+move_assignment
```

# РГ21:

```
* Stacktrace

* Constexpr everything

   * algorithms

   * iterators

   * utility
```

# РГ21:

```
* Stacktrace

* Constexpr everything

* Realloc
```

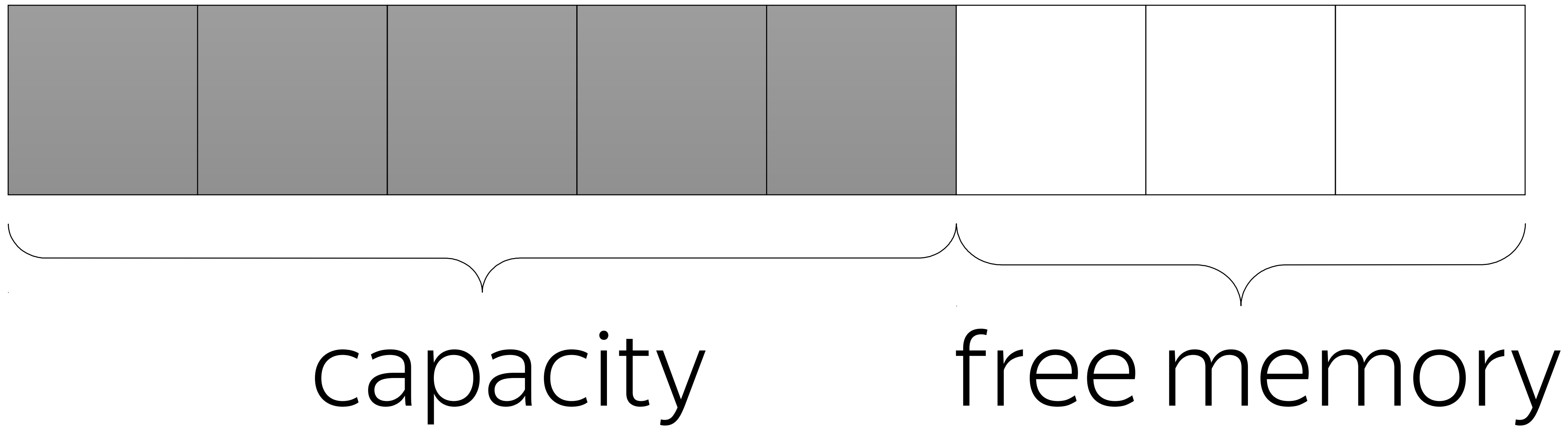
# РГ21:

```
* Stacktrace

* Constexpr everything

* Realloc (презентовали)

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

# РГ21:

```
* Stacktrace

* Constexpr everything
```

* Realloc *(презентовали)*



# РГ21:

* Stacktrace

* Constexpr everything

* Realloc *(презентовали)*

```
template <class T>

using pool_vector

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

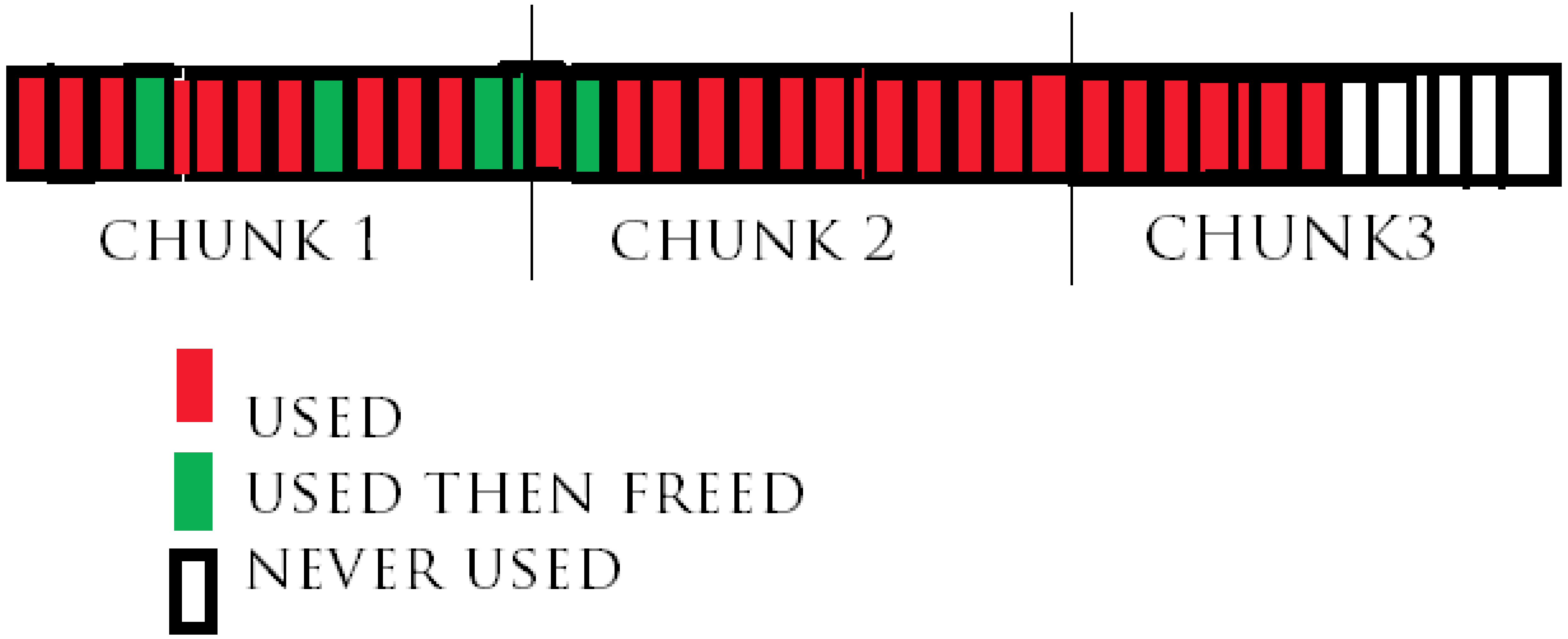
# РГ21:

* Stacktrace

* Constexpr everything

* Realloc *(презентовали)*

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



# РГ21:

- Stacktrace
- Constexpr everything
- Realloc *(презентовали)*
- Concurrent unordered hash map

# РГ21:

* Stacktrace

* Constexpr everything

* Realloc *(презентовали)*

* Concurrent unordered hash map

* Numbers

# РГ21:

```
* Stacktrace

* Constexpr everything

* Realloc (презентовали)

* Concurrent unordered hash map

* Numbers

* [[shared]] (сопереживали)
```

# РГ21:

* Stacktrace
* Constexpr everything
* Realloc *(презентовали)*
* Concurrent unordered hash map
* Numbers
* [[shared]] *(сопереживали)*
* Ultimate copy elisions

# РГ21:

* Stacktrace

* Constexpr everything

* Realloc *(презентовали)*

* Concurrent unordered hash map

* Numbers

* [[shared]] *(сопереживали)*

* 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

https://stdcpp.ru/

深入实践
Boost
Boost程序库开发的94个秘笈
blems
ation
kbook
부스트 라이브러리 예제로 배우
Boost C++ 애플리케이
Antony Polukhin
Boost C++ Ap
Development
Cookb
Second Edition
Recipies to simplify your Application
Анна Нимус
Цветы для случайной прохожей