

Yandex Taxi

# Шустрый, простой и современный C++

**Полухин Антон**

Antony Polukhin

**Y**andex Taxi

# О чём поговорим

Зачем вообще об этом говорить?

Алгоритмы

Структуры данных

Move семантика

Оптимизации и микрооптимизации

Многопоточность

Почему и Зачем?!



# Задача

# Задача

```
pres ^= (pres >> 31) & 0x7fffffff;
```

# Задача

```
int prec = fi.i;  
prec ^= (prec >> 31) & 0x7fffffff;
```

# Задача

```
union { float f; int i; } fi;  
fi.f = f;  
int prec  = fi.i;  
prec ^= (prec >> 31) & 0x7fffffff;
```



# Задача

```
unsigned apply(float f) {  
    union { float f; int i; } fi;  
    fi.f = f;  
    int prec  = fi.i;  
    prec ^= (prec >> 31) & 0x7fffffff;
```

Что это за дичь!?!?!?!?



# Задача

```
struct mapper {  
    unsigned apply(float f) {  
        union { float f; int i; } fi;  
        fi.f = f;  
        int prec = fi.i;  
        prec ^= (prec >> 31) & 0x7fffffff;  
        for (unsigned i = 0; ; ++i) {  
            if (prec < ranges[i]) return i;  
        }  
    }  
};  
std::vector<int> ranges;
```

# Задача

```
struct mapper {  
    unsigned apply(float f) {  
        union { float f; int i; } fi;  
        fi.f = f;  
        int prec = fi.i;  
        prec ^= (prec >> 31) & 0x7fffffff;  
        for (unsigned i = 0; ; ++i) {  
            if (prec < ranges[i]) return i;  
        }  
    }  
};  
std::vector<int> ranges;
```

Оно того стоило?



# Int vs Float

```
bool test_ints(int lhs, int rhs) {  
    return lhs < rhs;  
}
```

```
bool test_floats(float lhs, float rhs) {  
    return lhs < rhs;  
}
```

# Int vs Float

<https://godbolt.org/#>

# Int vs Float

```
test_ints(int, int): # @test_ints(int, int)
```

```
    cmp edi, esi
```

```
    setl al
```

```
    ret
```

```
test_floats(float, float): # @test_floats(float, float)
```

```
    ucomiss xmm1, xmm0
```

```
    seta al
```

```
    ret
```



# cmp vs ucomiss

## Instruction tables

Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs

By Agner Fog. Technical University of Denmark.

# cmp vs ucomiss

- Latency:

This is the delay that the instruction generates in a dependency chain. The numbers are minimum values. Cache misses, misalignment, and exceptions may increase the clock counts considerably. Where hyperthreading is enabled, the use of the same execution units in the other thread leads to inferior performance. Denormal numbers, NAN's and infinity do not increase the latency. The time unit used is core clock cycles, not the reference clock cycles given by the time stamp counter.
- Reciprocal throughput:

The average number of core clock cycles per instruction for a series of independent instructions of the same kind in the same thread.

## Integer instructions

Instruction	Operands	$\mu$ ops fused domain	$\mu$ ops unfused domain	$\mu$ ops each port	Latency	Reciprocal through put	Comments
Move instructions							
MOV	r,i	1	1	p0156		0.25	
MOV	r8/16,r8/16	1	1	p0156	1	0.25	

# cmp vs ucomiss

## Haswell

ADD SUB	m,r/i	2	4	2p0156 2p237 p4	6	1
ADC SBB	r,r/i	2	2	2p0156	2	1
ADC SBB	r,m	2	3	2p0156 p23		1
ADC SBB	m,r/i	4	6	3p0156 2p237 p4	7	2
CMP	r,r/i	1	1	p0156	1	0.25
CMP	m,r/i	1	2	p0156 p23	1	0.5
INC DEC NEG	r	1	1	p0156	1	0.25
NOT						
INC DEC NOT	m	3	4	p0156 2p237 p4	6	1
NEG						

# cmp vs ucomiss

Instruction	Operands	1	2	3	4	5	6	7
VRCPPS	y,y	3	3	2p0 p15	7	2	AVX	
VRCPPS	y,m256	4	4	2p0 p15 p23		2	AVX	
CMPccSS/D								
CMPccPS/D	x,x / v,v,v	1	1	p1	3	1		
CMPccSS/D								
CMPccPS/D	x,m / v,v,m	2	2	p1 p23		1		
(U)COMISS/D	x,x	1	1	p1		1		
(U)COMISS/D	x,m32/64	2	2	p1 p23		1		
MAXSS/D PS/D								
MINSS/D PS/D	x,x / v,v,v	1	1	p1	3	1		
MAXSS/D PS/D								
MINSS/D PS/D	x,m / v,v,m	1	2	p1 p23		1		

cmp vs ucomiss

Разницы практически нет

cmp vs ucomiss

Разницы практически нет

Нужны инструменты покруче

cmp vs ucomiss

Intel Architecture Code Analyzer

# Идём глубже

```
#include "iacaMarks.h"

template <class T>

unsigned test(const T* range, T prec) {
    for (unsigned i = 0; ; ++i) {
        if (prec < ranges[i]) return i;
    }
}

void testing_cmps(unsigned& a, const int* range, int prec) {
    IACA_START
    a = test(range, prec);
    IACA_END
}
```



# Идём глубже

```
#include "iacaMarks.h"

template <class T>

unsigned test(const T* range, T prec) {
    for (unsigned i = 0; ; ++i) {
        if (prec < ranges[i]) return i;
    }
}

void testing_cmps(unsigned& a, const int* range, int prec) {
    IACA_START
    a = test(range, prec);
    IACA_END
}
```

# cmp vs ucomiss

## Throughput Analysis Report

-----

Block Throughput: 2.00 Cycles

Throughput Bottleneck: Dependency chains

<...>

Total Num Of Uops: 10

# Идём глубже

```
#include "iacaMarks.h"

template <class T>

unsigned test(const T* range, T prec) {
    for (unsigned i = 0; ; ++i) {
        if (prec < ranges[i]) return i;
    }
}

void testing_cmps(unsigned& a, const float* range, float prec) {
    IACA_START
    a = test(range, prec);
    IACA_END
}
```

# cmp vs ucomiss

Throughput Analysis Report

-----

Block Throughput: 2.52 Cycles

Throughput Bottleneck: FrontEnd

<...>

Total Num Of Uops: 12

Давайте воспользуемся крутым  
бенчмарком и определим разницу  
между стр и iscomіs на **практике**



# cmp vs ucomiss

measure	naïve	optim
*** GCC 7		
8	1.00	1.13
64	1.00	1.02
512	1.00	1.21
4096	1.00	0.98
8192	1.00	1.18
*** Clang 5		
8	1.00	1.19
64	1.00	1.02
512	1.00	1.00
4096	1.00	1.18
8192	1.00	0.90
***		

# Стоило ли это так оптимизировать?

# Стоило ли это так оптимизировать?

- Мы потратили кучу времени, пытаясь осознать, что написано в коде
- Эта функция на критическом пути?



# История про 99%, 50% и 1%

- Мы потратили кучу времени, пытаюсь осознать, что написано в коде
- Эта функция на критическом пути?
  - Выиграли 20% производительности в **лучшем** случае

# История про 99%, 50% и 1%

■ Профилирование всего приложения спец инструментами

# История про 99%, 50% и 1%

- Профилирование всего приложения спец инструментами
- Знаете и без профилирования, что тормозит?

# История про 99%, 50% и 1%

- Профилирование всего приложения спец инструментами
- Знаете и без профилирования, что тормозит?
  - Замените тело функции на пустое и перепроверьте!

Так что, можно весь код писать тят-ляп, хорошо оптимизируя только несколько основных функций?



Нельзя!

Кеши, оптимизаторы, рост данных...



# Не пессимизируем

```
struct mapper {  
    unsigned apply(float f) {  
        union { float f; int i; } fi;  
        fi.f = f;  
        int prec = fi.i;  
        prec ^= (prec >> 31) & 0x7fffffff;  
        for (unsigned i = 0; ; ++i) {  
            if (prec < ranges[i]) return i;  
        }  
    }  
};  
std::vector<int> ranges;
```

# Не пессимизируем

```
struct mapper {  
    unsigned apply(float f) {  
        union { float f; int i; } fi;  
        fi.f = f;  
        int prec = fi.i;  
        prec ^= (prec >> 31) & 0x7fffffff;  
        for (unsigned i = 0; ; ++i) {  
            if (prec < ranges[i]) return i;  
        }  
    }  
};  
std::vector<int> ranges;
```



# Правильное решение

```
struct mapper_fixed {  
    unsigned apply(float f) {  
        const auto it = std::lower_bound(ranges.cbegin(), ranges.cend(), f);  
        return it - ranges.cbegin();  
    }  
  
    std::vector<float> ranges;  
};
```

Какие выводы мы сделали? Что  
запомнили?



# Выводы

Перед микрооптимизациями нужно оптимизировать алгоритм

Перед любыми оптимизациями нужна **профилировка**

Надо писать **комментарии** к микрооптимизированному коду

**Не** надо **пессимизировать**

Прежде чем мы перейдём к  
алгоритмам...



# Оставим за бортом

- IO (сеть, диск, устройства со своей памятью и т.д.)

- Архитектура

**А стоит ли делать маленькие и  
понятные микрооптимизации?**



# Давайте распрощаемся с иллюзиями

<https://godbolt.org/#>

```
unsigned foo(unsigned i)
```

```
    return i * 2;
```

// mul или imul – умножение

```
    return i << 1;
```

// shl или + – сдвиг влево

```
    return i / 64;
```

// div – деление

```
    return i / 11;
```

// shr – сдвиг вправо

```
    return i % 2;
```

// add и sub – сложение и вычитание

```
    return i * i * i * i * i * i;
```

```
    ...
```

# Оговорки

■ <https://godbolt.org/z/H-0TA->

■ <https://godbolt.org/z/4MW8LL>



# Алгоритмы



# «O» большое

«O» большое — время работы алгоритма/функции в зависимости от количества входных элементов  $N$

# «O» большое

“O” большое — время работы алгоритма/функции в зависимости от количества входных элементов N

for (size\_t i = 0; i < N; ++i)  $\Rightarrow O(N)$

# «O» большое

“O” большое — время работы алгоритма/функции в зависимости от количества входных элементов  $N$

for (size\_t i = 0; i < N; ++i)  $\Rightarrow O(N)$

for (size\_t i = 0; i < N; ++i)  
    for (size\_t j = 0; j < N; ++j)  $\Rightarrow O(N^2)$

# «O» большое

N	$N \cdot \log(N)$	$N \cdot N$
2	2	4
4	8	16

# «O» большое

N	$N \cdot \log(N)$	$N \cdot N$
2	2	4
4	8	16
8	24	64
16	64	256
32	160	1,024
64	384	4,096
128	896	16,384
256	2,048	65,536
512	4,608	262,144
1,024	10,240	1,048,576

# std::sort(**beg**, **end**)

Сортирует диапазон

4	1	7	3	0	9	5	8	2	6	@
---	---	---	---	---	---	---	---	---	---	---

0	1	2	3	4	5	6	7	8	9	@
---	---	---	---	---	---	---	---	---	---	---

# it std::lower\_bound(beg, end, value)

Бинарный поиск в **сортированном** диапазоне

Позиция элемента value или элемент сразу след за ним

Найти позицию для вставки value в сортированный диапазон

it std::lower\_bound(beg, end, 5)

0	1	2	3	4	5	6	7	8	9	@
0	1	2	3	4	4	6	7	8	9	@



# pair std::equal\_range(beg, end, value)

Бинарный поиск диапазона в **сортированном** диапазоне

pair std::equal\_range(beg, end, 5)

0	1	2	3	4	5	6	7	8	9	@
0	1	2	3	4	4	6	7	8	9	@
0	1	2	5	5	5	7	7	8	9	@

# «O» большое

 std::sort  $\Rightarrow O(N \log_2(N))$

# «O» большое

<code>std::sort</code>	$\Rightarrow O(N \log_2(N))$
<code>std::stable_sort</code>	$\Rightarrow O(N \log_2^2(N))$

# «O» большое

~~std::sort~~  $\Rightarrow O(N \log_2(N))$

~~std::stable\_sort~~  $\Rightarrow O(N \log_2^2(N))$

std::minmax\_element  $\Rightarrow O(N)$

std::partition  $\Rightarrow O(N)$

std::nth\_element  $\Rightarrow \sim O(N)$

std::partial\_sort  $\Rightarrow O(N \log_2(S))$

# result std::partition(beg, end, pred)

Слева от result будет true, справа false:

[beg, result) == true

[result, end) == false

4	0	3	1	2	9	5	8	7	6	@
---	---	---	---	---	---	---	---	---	---	---

# `std::nth_element(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

- Если отсортировать `[beg, end)` то значение `mid` не изменится

- Слева от `mid` — значения *меньшие* или *равные* `mid`

- Справа от `mid` - значения *большие* или *равные* `mid`

4	0	3	1	2	5	9	8	7	6	@
---	---	---	---	---	---	---	---	---	---	---

# `std::nth_element(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

- Если отсортировать `[beg, end)` то значение `mid` не изменится

- Слева от `mid` — значения *меньшие* или *равные* `mid`

- Справа от `mid` - значения *большие* или *равные* `mid`

4	0	3	1	2	5	9	8	7	6	@
0	1	2	3	4	5	6	7	8	9	@

# std::nth\_element

Найти 5 людей с наименьшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end());
```

Найти 5 людей с наибольшим балансом

```
std::nth_element(v.begin(), v.begin() + 4, v.end(), std::greater<>{});
```

Найти 1001 позвонившего

```
std::nth_element(v.begin(), v.begin() + 1000, v.end());
```



# `std::partial_sort(beg, mid, end)`

Выставить значение по итератору `mid` так чтобы:

- `[beg, mid)` не изменятся, если отсортировать `[beg, end)`

- `[beg, mid)` - отсортированы

0	1	2	3	4	9	5	8	7	6	@
---	---	---	---	---	---	---	---	---	---	---

# std::partial\_sort

Распределить 5 призовых мест по наименьшему кол-ву штрафных баллов

```
std::partial_sort(v.begin(), v.begin() + 5, v.end());
```

Покарать 5 школьников, пришедших последними на урок

```
std::partial_sort(v.begin(), v.begin() + 5, v.end(), std::greater<>{});
```

# std::minmax\_element

Найти самого бедного и самого богатого клиента банка

```
auto mm = std::minmax_element(v.begin(), v.end());  
std::cout << *mm.first << ' ' << *mm.second << '\n';
```

К компилятору!



<https://github.com/apolukhin>



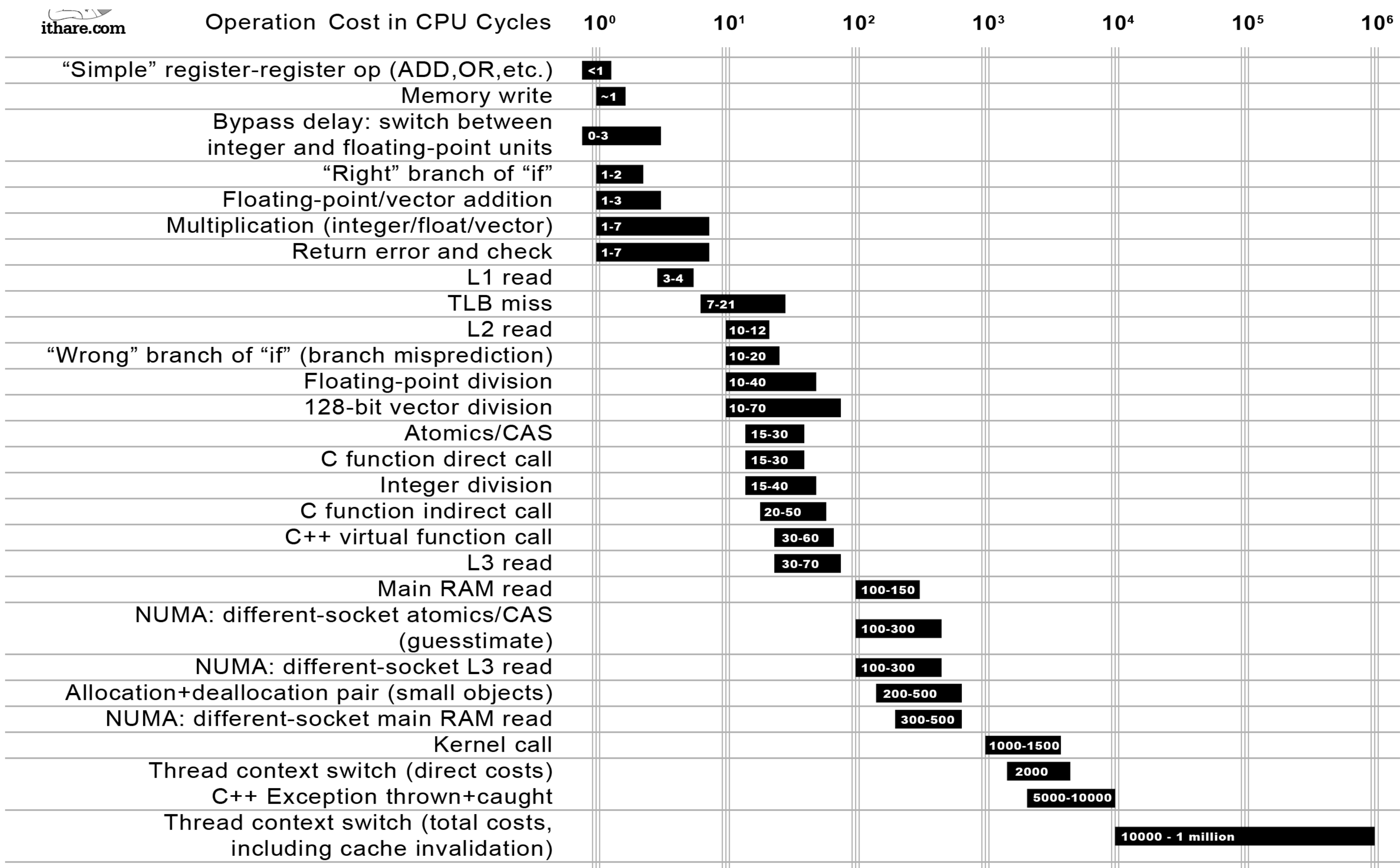
# Контейнеры



# «O» большое

«O» большое — время работы алгоритма/функции в зависимости от количества входных элементов  $N$

# Знай микро-врага в лицо!





# Знай микро-врага в лицо!

Allocation+deallocation pair (small objects)	200-500
NUMA: different-socket main RAM read	300-500
Kernel call	1000-1500
Thread context switch (direct costs)	2000
C++ Exception thrown+caught	5000-10000
Thread context switch (total costs, including cache invalidation)	10000 - 1 million

# Знай нано-врага в лицо!

L2 read	10-12	
“Wrong” branch of “if” (branch misprediction)	10-20	
Floating-point division	10-40	
128-bit vector division	10-70	
Atomics/CAS	15-30	
C function direct call	15-30	
Integer division	15-40	
C function indirect call	20-50	
C++ virtual function call	30-60	
L3 read	30-70	
Main RAM read		100-150

# Знай нано-врага в лицо!

L2 read	✗	10-12	
“Wrong” branch of “if” (branch misprediction)	✗	10-20	
Floating-point division	✗	10-40	
128-bit vector division	✗	10-70	
Atomics/CAS	✗	15-30	
C function direct call	✗	15-30	
Integer division	✗	15-40	
C function indirect call	✗	20-50	
C++ virtual function call	✗	30-60	
L3 read		30-70	
Main RAM read			100-150

# Кеш линия

- В одной кеш линии x86 — 64 Байта

- В одну кеш линию помещается 16 int

- 

# Знай нано-врага в лицо!

L2 read	✗	10-12	
“Wrong” branch of “if” (branch misprediction)	✗	10-20	
Floating-point division	✗	10-40	
128-bit vector division	✗	10-70	
Atomics/CAS	✗	15-30	
C function direct call	✗	15-30	
Integer division	✗	15-40	
C function indirect call	✗	20-50	
C++ virtual function call	✗	30-60	
L3 read		30-70	
Main RAM read			100-150

Все контейнеры, для хранения данных (не ассоциативные)

# Все контейнеры, для хранения данных

 `std::vector`

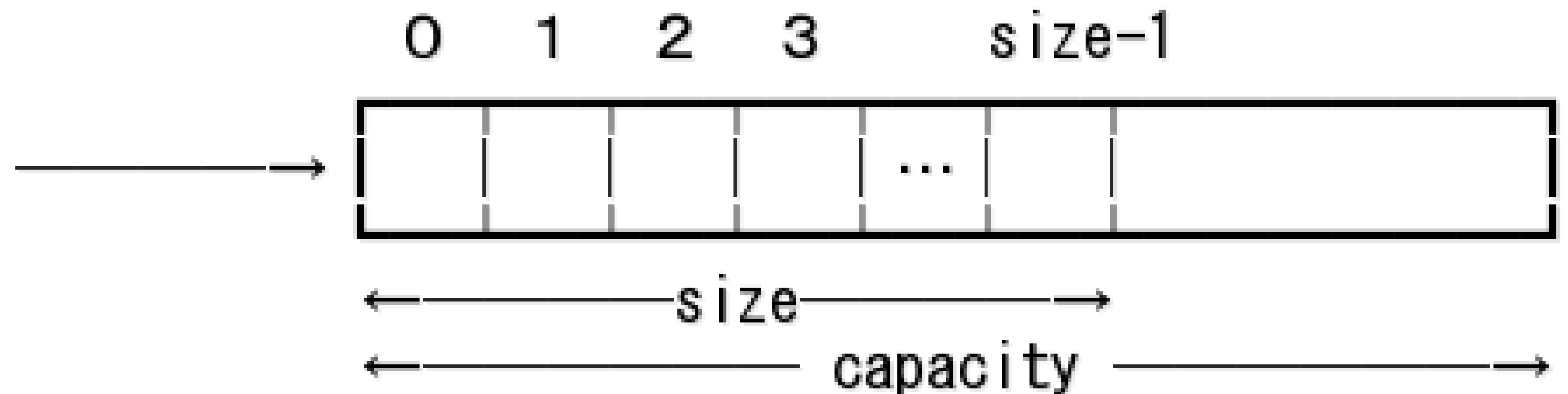
container\_1.hpp





# std::vector

<code>std::vector&lt;int&gt;</code>
<code>m_data : int *</code> <code>m_capacity : size_t</code> <code>m_size : size_t</code>
<code>empty() : bool</code> <code>size() : size_t</code> <code>.....</code>



container\_2.hpp



~~Бағи~~ Фишки `std::vector`

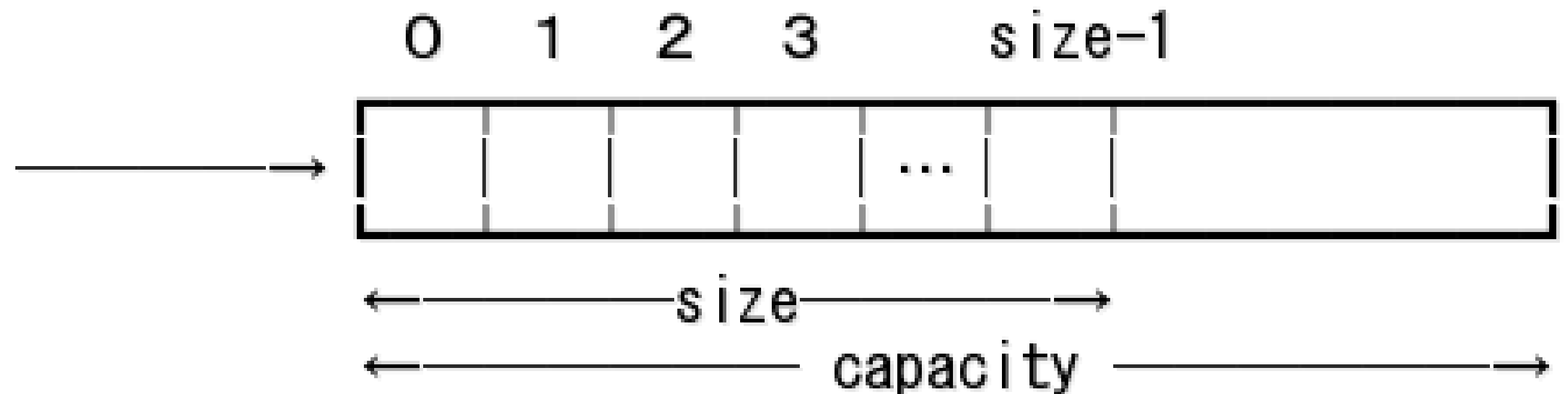


# Фишки

Move конструкторы и noexcept

# Move construction & noexcept

<code>std::vector&lt;int&gt;</code>
<code>m_data : int *</code> <code>m_capacity : size_t</code> <code>m_size : size_t</code>
<code>empty() : bool</code> <code>size() : size_t</code> <code>.....</code>



# Фишки

Move конструкторы и noexcept

TriviallyCopyable

std::vector::reserve

std::vector::at

std::vector<bool>

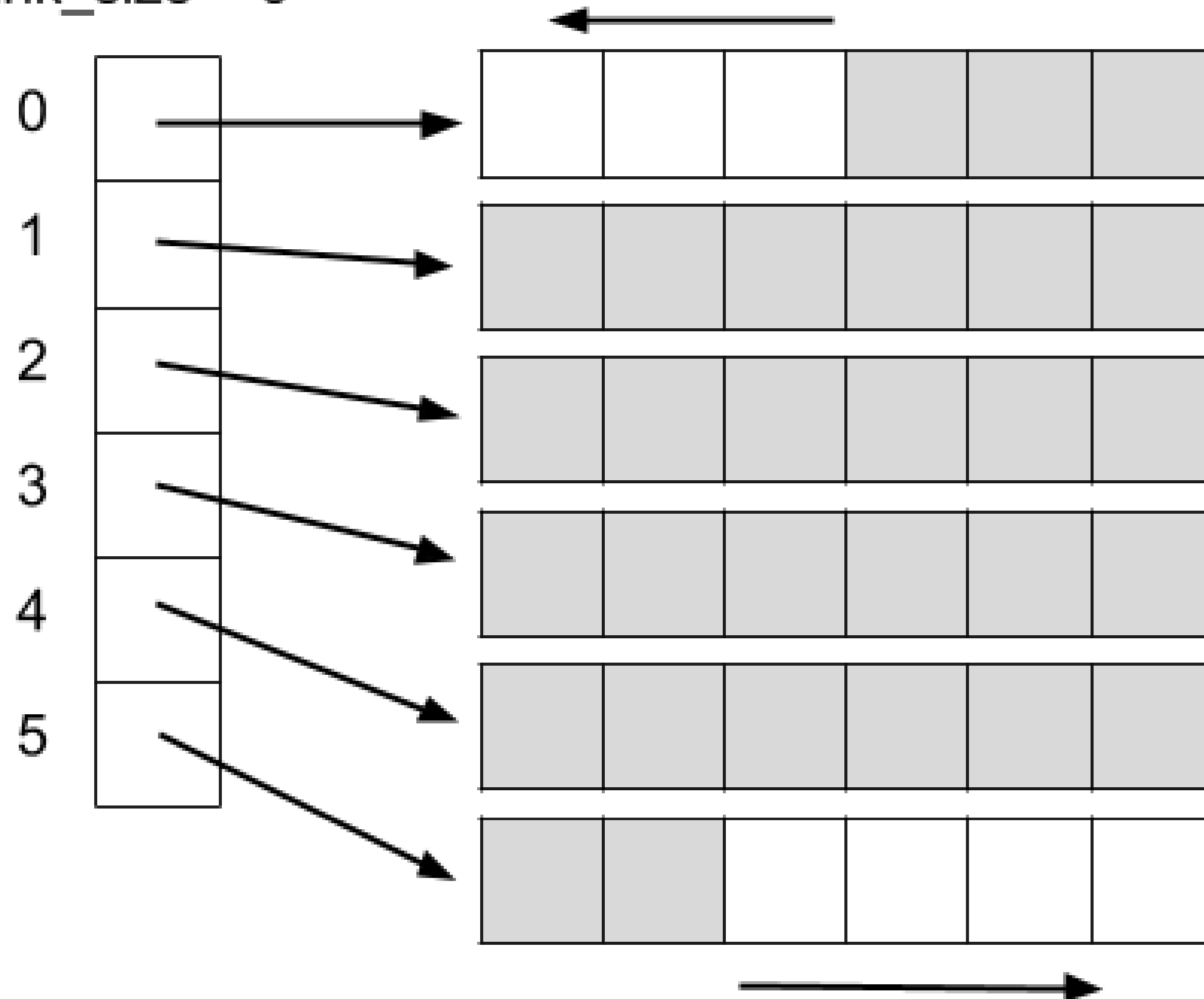
container\_[3-5].hpp



# Немного о `std::deque`

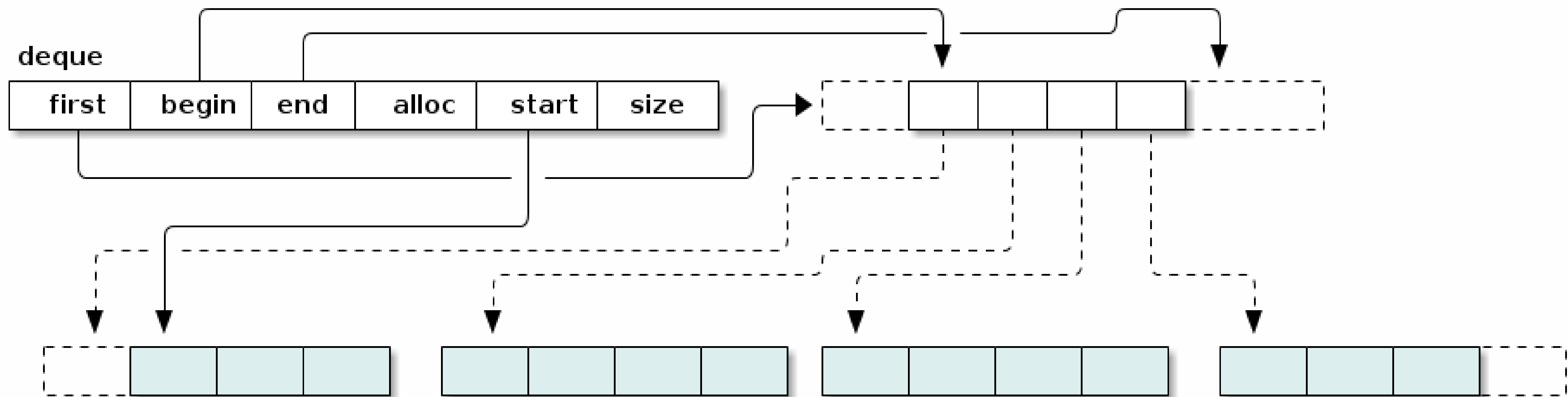
shift = 3

chunk\_size = 6





# Немного о `std::deque`



# Немного о `std::deque`

 <https://godbolt.org/g/4cJ5HF>

# Немного о `std::deque`

<https://godbolt.org/g/4cJ5HF>

[https://gcc.gnu.org/bugzilla/show\\_bug.cgi?id=81461](https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81461)

Мы уже пробовали использовать  
short?



container\_[1-2].hpp



Особые случаи, когда `vector` не очень



# Особые случаи

 Огромные объёмы данных

# Особые случаи

- Огромные объёмы данных
- Многопоточность и особая нагрузка



# Особые случаи

- Огромные объёмы данных
- Многопоточность и особая нагрузка
- Ассоциативные контейнеры (иногда)

# Особые случаи

- Огромные объёмы данных

- Многопоточность и особая нагрузка

- Ассоциативные контейнеры (иногда)

- `std::string`

# Особые случаи

- Огромные объёмы данных

- Многопоточность и особая нагрузка

- Ассоциативные контейнеры (иногда)

- `std::string`

- `std::array`

# Особые случаи

- Огромные объёмы данных

- Многопоточность и особая нагрузка

- Ассоциативные контейнеры (иногда)

- `std::string`

- `std::array`

- Использование сторонних библиотек со стековыми векторами

`boost::container::small_vector`



container\_[1-5].hpp



# Ассоциативные контейнеры



# Все ассоциативные контейнеры



# Все ассоциативные контейнеры

 `std::unordered_set / std::unordered_map`

# std::unordered\_map<int, int\*>

\_Hashtable<int, int\*, ...>

__bucket_type*	_M_buckets
size_type	_M_bucket_count
__before_begin	_M_bbegin
size_type	_M_element_count

\_Hash\_node<int\*, false>

_M_nxt
_M_v pair<int, int*>

node\_base\*

\_\_node\_base\*

\_\_node\_base\*



_M_nxt
_M_v

_M_nxt
_M_v

_M_nxt
_M_v

# Внимание

`std::unordered_*` контейнеры инвалидируют итераторы при вставке! Лучше использовать ссылки/указатели на элементы

Не нужен value? Используйте `unordered_set<key>`

container\_6.hpp



Особые случаи



# Особые случаи

Нужно хранить данные **именно** упорядоченно

# Особые случаи

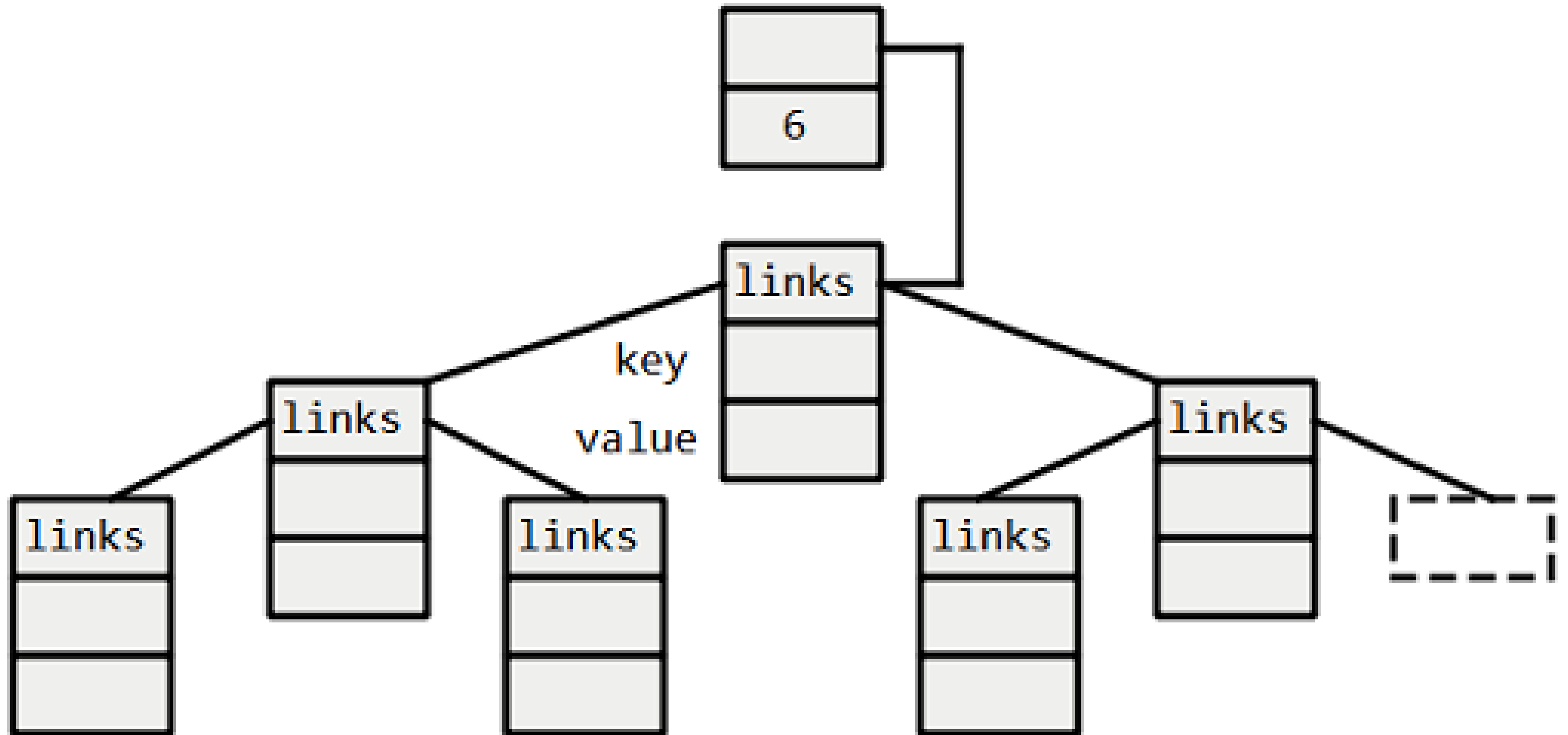
- Нужно хранить данные **именно** упорядоченно
- Малые объёмы данных

# Особые случаи

- Нужно хранить данные **именно** упорядоченно
- Малые объёмы данных
- Использование сторонних библиотек с “плоскими” ассоциативными контейнерами



std::map



flat\_set / flat\_map



container\_6.hpp



# Move семантика



# Rvalue & lvalue

```
void examples() {  
    std::string s0{"Hello word. Nice to be here!"};  
    std::string s1 = s0;           // Конструктор копирования для s1  
    std::string s2 = std::move(s0); // Конструктор перемещения для s2,  
                                   // s0 становится пустым  
  
    s0 = s1;                       // Оператор присваивания для s0  
    s2 = std::move(s0);            // Оператор перемещения для s2  
}
```

# Rvalue & lvalue args

```
void example_rvalue(std::string&& s);    // rvalue reference
```

```
void example_lvalue(std::string& s);     // lvalue reference
```

```
void example_rvalue_exp(std::string&& s) { // rvalue reference
```

```
    std::string s_copy = s;                // Конструктор КОПИРОВАНИЯ! Нужен std::move
```

```
    // для конструктора перемещения!
```

```
}
```

# Forwarding reference

```
template <class T>
void foo(T&& value) { // forwarding reference - любая ссылка (const&, &, && и т. д.)

    std::string s = std::forward<T>(value); // превращает бывшую && в настоящую
                                           // rvalue ТОЛЬКО в случае forwarding reference
}
```

# Move элементов из контейнера

```
void example_appending_data(std::vector<std::string>&& data) {  
    some_vector.insert(  
        some_vector.end(),  
        data.begin(),  
        data.end()  
    );  
}
```



# Move элементов из контейнера

```
#include <iterator>
```

```
void example_move_iterator(std::vector<std::string>&& data) {  
    some_vector.insert(  
        some_vector.end(),  
        std::make_move_iterator(data.begin()),  
        std::make_move_iterator(data.end())  
    );  
}
```

# Для производительности:

**std::move** объект если вы его больше не используете

И если вы его

**не** возвращаете из функции

`return str;`

**не** создали прямо во время вызова функции

`rvalue(std::string{"Hello"})`

**не** возвращаете его из функции

`rvalue(foo())`

# Move семантика

```
v_new.push_back(std::move(v.back())); v.pop_back();
```

```
v_new.front() = std::move(v[10]);
```

```
v_new.insert(  
    v_new.end(),  
    std::make_move_iterator(v.begin()),  
    std::make_move_iterator(v.end())  
);
```

container\_7.hpp



Не пессимизируйте



# Простой код сразу пишете правильно!

 `std::vector`

# Простой код сразу пишете правильно!

 `std::vector`

 `std::vector::reserve()`


# Простой код сразу пишете правильно!



```
std::vector
```



```
std::vector::reserve()
```



```
++it;
```



# Простой код сразу пишете правильно!

`std::vector`

`std::vector::reserve()`

`++it;`

`for (auto& v: container)`

`std::unordered_map`

# Простой код сразу пишете правильно!

`std::vector`

`std::vector::reserve()`

`++it;`

`for (auto& v: container)`

`std::unordered_map`

`<algorithm>`

# Простой код сразу пишете правильно!

`std::vector`

`std::vector::reserve()`

`++it;`

`for (auto& v: container)`

`std::unordered_map`

`<algorithm>`

`??? std::move & std::make_move_iterator ???`

# Простой код сразу пишете правильно!

`std::vector`

`std::vector::reserve()`

`++it;`

`for (auto& v: container)`

`std::unordered_map`

`<algorithm>`

`??? std::move & std::make_move_iterator ???`

`string_view`

# Простой код сразу пишете правильно!

`std::vector`

`std::vector::reserve()`

`++it;`

`for (auto& v: container)`

`std::unordered_map`

`<algorithm>`

`??? std::move & std::make_move_iterator ???`

`string_view`

Пишите выражения просто и используйте целые числа если float вам не нужен

# Многопоточность



# Доступ (неправильный) к общим переменным

```
int shared_i = 0;

void do_inc() {
    for (std::size_t i = 0; i < 500000; ++i) {
        // ...
        const int i_snapshot = ++shared_i;
        // ...
    }
}

void do_dec() {
    for (std::size_t i = 0; i < 500000; ++i) { const int i_snapshot = --shared_i;
}
```

# Доступ (неправильный) к общим переменным

```
int shared_i = 0;
```

```
void do_inc();
```

```
void do_dec();
```

```
void run() {
```

```
    std::thread t1(&do_inc);
```

```
    do_dec();
```

```
    t1.join();
```

```
    // assert(shared_i == 0); // Oops!
```

```
    std::cout << shared_i << std::endl;
```

```
}
```



# Доступ к общим переменным

```
#include <mutex>
```

```
int shared_i = 0;
```

```
std::mutex mutex_i;
```

```
void do_inc() {
```

```
    for (std::size_t i = 0; i < 500000; ++i) {
```

```
        // ...
```

```
    {
```

```
        std::lock_guard<std::mutex> lock(mutex_i);
```

```
        const int i_snapshot = ++shared_i;
```

```
    }
```

```
    // ...
```

# Правильный и быстрый доступ

```
#include <atomic>
```

```
std::atomic<int> shared_i{0};
```

```
void do_inc() {  
    for (std::size_t i = 0; i < 500000; ++i) {  
        // ...  
        const int i_snapshot = ++shared_i;  
        // ...  
    }  
}
```

# Модель памяти



# Memory order

```
std::atomic<int> i{0};
```

```
int a, b, c, d;
```

# Memory order

```
std::atomic<int> i{0};
```

```
int a, b, c, d;
```

```
void dependency() {
```

```
    a = 0;
```

```
    b = a + 1;
```

```
    c = b + a;
```

```
}
```

# Memory order

```
std::atomic<int> i{0};
```

```
int a = 10, b = 11, c = 12;
```

```
void dependency() {
```

```
    a = 0;          // Для других потоков ↑↑ ↓↓
```

```
    b = a + 1;      // Для других потоков ↑↑ ↓↓
```

```
    c = b + a;      // Для других потоков ↑↑ ↓↓
```

```
}
```

# Memory order

```
std::atomic<int> i{0};
```

```
int a, b, c, d;
```

```
void seq_cst() {
```

```
    /* ↑↑ */ a = 0;
```

```
    i.fetch_add(1, std::memory_order_seq_cst);
```

```
    /* ↑ do seq_cst, ↓ do seq_cst */ b = 0;
```

```
    /* ↑ do seq_cst, ↓ do seq_cst */ c = 0;
```

```
    i.fetch_sub(1, std::memory_order_seq_cst);
```

```
    /* ↓↓ */ d = 0;
```

```
}
```

# Memory order

`std::memory_order_acquire` / `std::memory_order_release`

`std::memory_order_relaxed`



Очереди



# conditional\_variable

```
#include <vector>
#include <condition_variable>

class mt_stack {
    std::vector<int>      data_;
    std::mutex           data_mutex_;
    std::condition_variable cond_;
public:
    void push(int data);
    int pop();
};
```

# conditional\_variable

```
void mt_stack::push(int data) {  
    std::unique_lock<std::mutex> lock(data_mutex_);  
    data_.push_back(data);  
    lock.unlock();  
  
    cond_.notify_one();  
}
```

# conditional\_variable

```
int mt_stack::pop() {  
    std::unique_lock<std::mutex> lock(data_mutex_);  
    while (data_.empty()) {  
        cond_.wait(lock);  
    }  
  
    int ret = data_.back();  
    data_.pop_back();  
    return ret;  
}
```

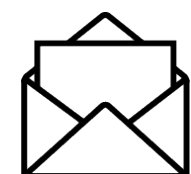
**Спасибо**

# Полухин Антон

Старший разработчик Yandex.Taxi



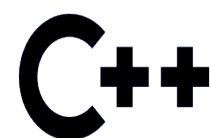
antoshkka@gmail.com



antoshkka@yandex-team.ru

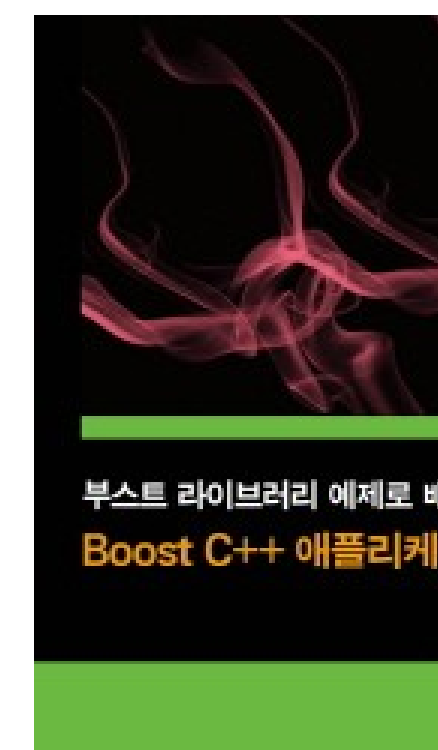


<https://github.com/apolukhin>



РГ21 C++ РОССИЯ

<https://stdcpp.ru/>



# Бонусы



shared\_ptr





# shared\_ptr

```
#include <memory>
```

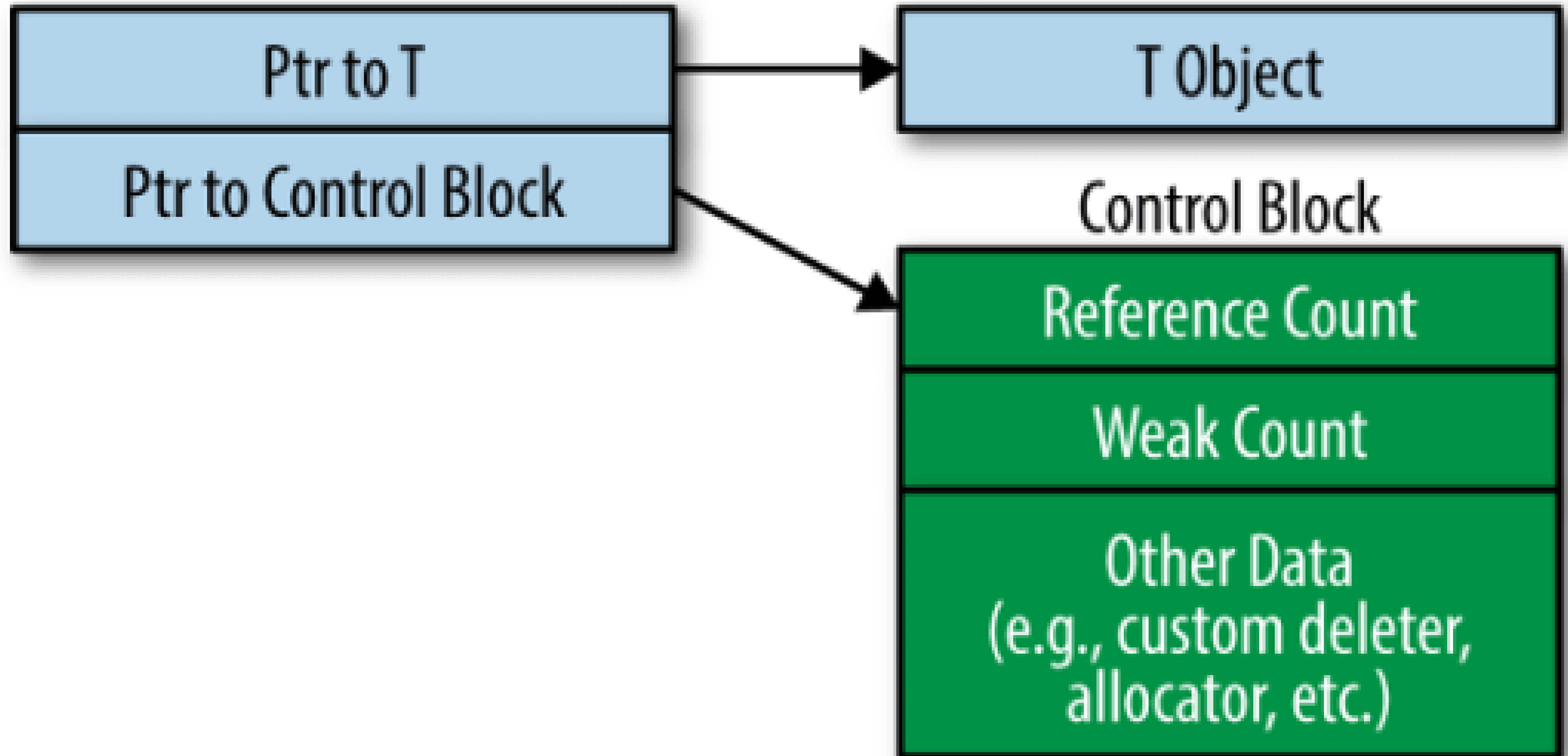
```
std::shared_ptr<int> p = std::make_shared<int>();
```

```
auto p1 = p;
```

```
*p1 = 42;
```

```
assert(*p == 42);
```

```
std::shared_ptr<T>
```



throw



memcpy/memmove

