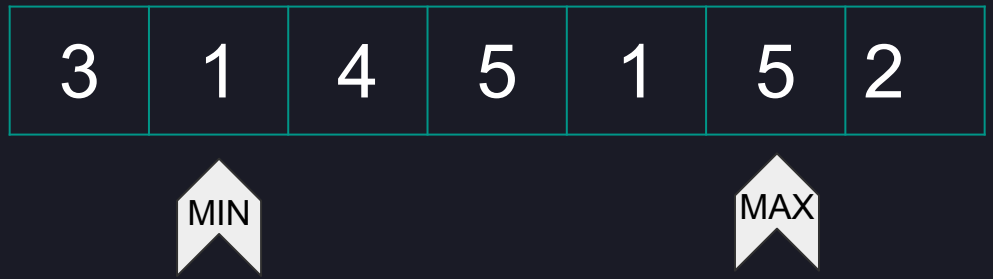
Why do we need `std::minmax_element` anyway?

Mini agenda

1. Problem statement
2. Inefficient solution
3. Efficient solution
4. Implementation
5. Microbenchmark

Problem statement

Find first smallest and last largest element in range [first, last)



```c
int min_element(int a[], int N) {
    int min = a[0];
    for (int i = 1; i < N; i++) {
        if (a[i] < min) {
            min = a[i];
        }
    }
    return min;
}
```

Number of comparisons: **N - 1**

Inefficient STL solution


```
std::vector xs{3, 1, 7, 6, 5, 1, 7, 4};
std::make_pair(std::min_element(xs.begin(), xs.end()),
               std::max_element(xs.rbegin(), xs.rend()));
```

Number of comparisons: **2 * (N - 1)**

Efficient STL solution


```
template <typename ForwardIt>
std::pair<ForwardIt, ForwardIt> std::minmax_element(ForwardIt first, ForwardIt last)
```

- 72’ Ira Pohl - UC Santa Cruz
- Described in CLRS as well

Inductive approach

Let **min** and **max** be pointing to the running **minimum** and **maximum**.

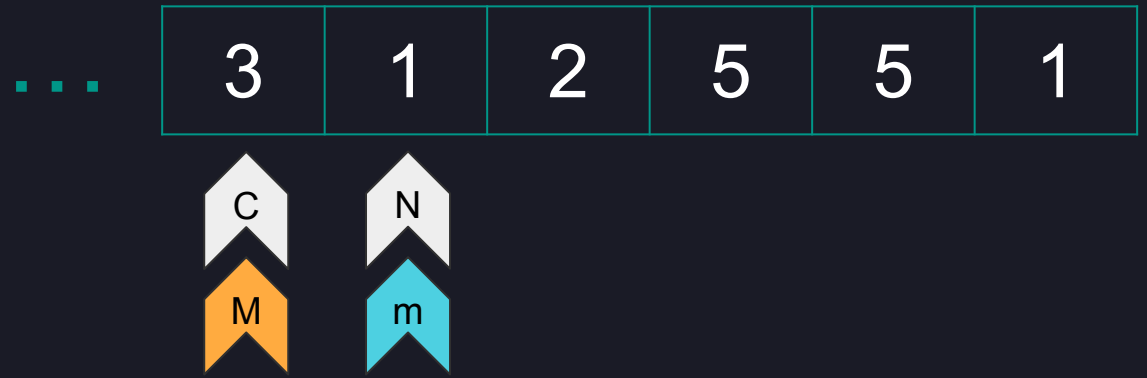
```
*min = 2
*max = 4
```


```
C - current
N - next
m - min candidate
M - max candidate
```

Inductive approach

Let **min** and **max** be pointing to the running **minimum** and **maximum**.

```
*min = ~~2~~ 1
*max = 4
```




```
C - current
N - next
m - min candidate
M - max candidate
```

Inductive approach

Let **min** and **max** be pointing to the running **minimum** and **maximum**.

```
*min = 1
*max = 4 5
```

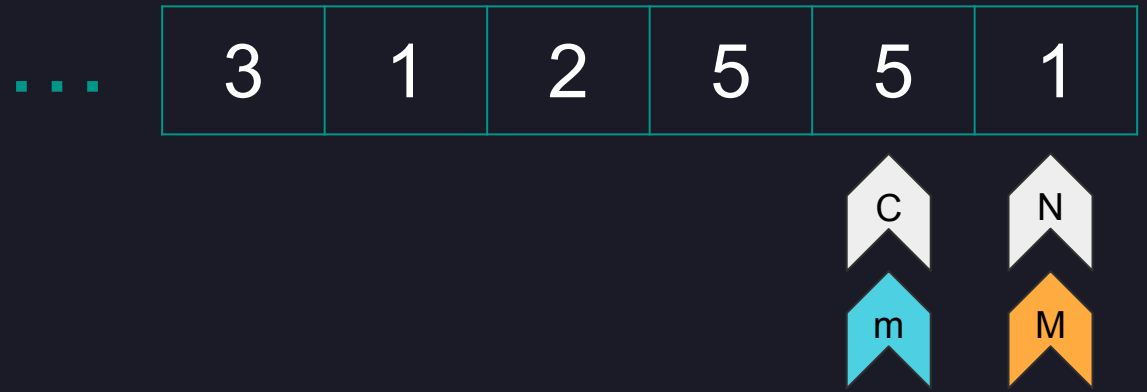
...


```
C - current
N - next
m - min candidate
M - max candidate
```

Inductive approach

Let **min** and **max** be pointing to the running **minimum** and **maximum**.

```
*min = 1
*max = 5
```

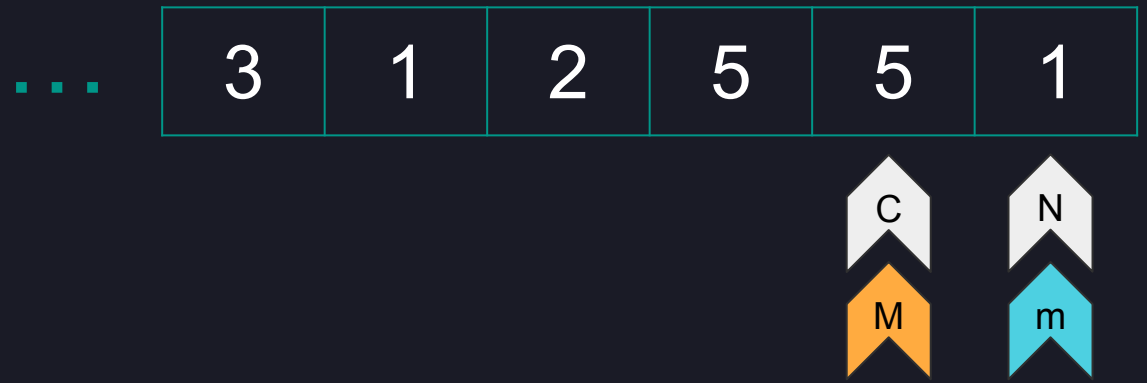



```
C - current
N - next
m - min candidate
M - max candidate
```

Inductive approach

Let **min** and **max** be pointing to the running **minimum** and **maximum**.

```
*min = 1
*max = 5 5
```




```
C - current
N - next
m - min candidate
M - max candidate
```

Number of comparisons: **3 / 2 * N - 2**

```
template <typename It>
// requires It is a ForwardIterator
std::pair<It, It> std::minmax_element(It first, It last)
```

```cpp
template <typename It, typename Compare>
// requires It is a ForwardIterator
//          Compare is a StrictWeakOrder(ValueType(It))
std::pair<It, It> std::minmax_element(It first, It last, Compare cmp)
```

20

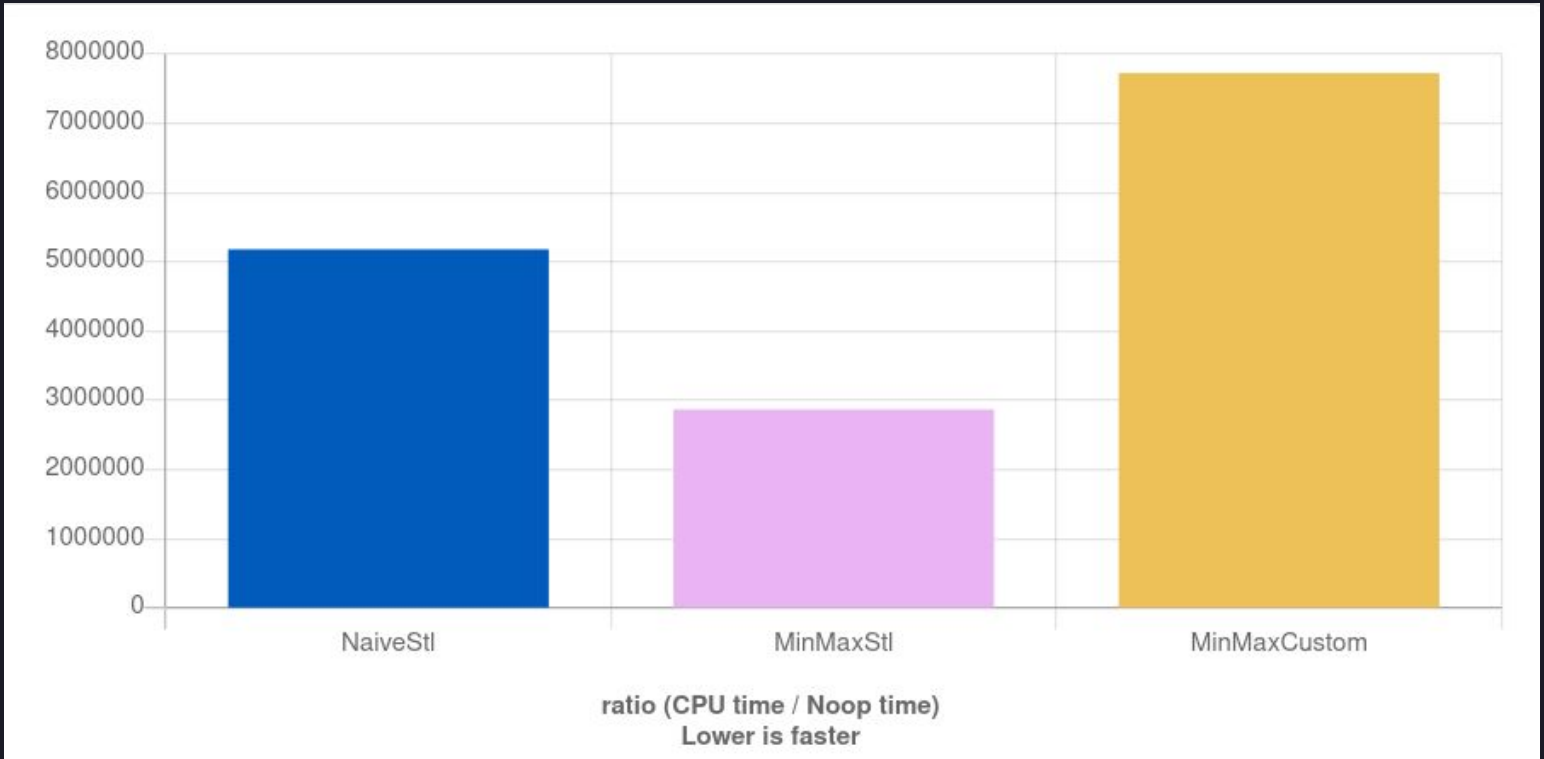
```
template <typename It, typename Compare>
  requires std::forward_iterator<It> &&
           std::indirect_strict_weak_order<Compare, It>
auto minmax_element(It first, It last, Compare cmp) -> std::pair<It, It>
```

CODE

Why do we need `std::minmax_element` anyway?

EFFICIENCY**

- `std::vector<int>` uniformly distributed 1 000 000 elements
- quickbench - clang-14 with -O3 + libstdc++



Thanks!

Aleksandar Šmigić

smiga287@gmail.com
linkedin.com/in/smiga287