LET'S TALK ABOUT STRING OPERATIONS IN C++17

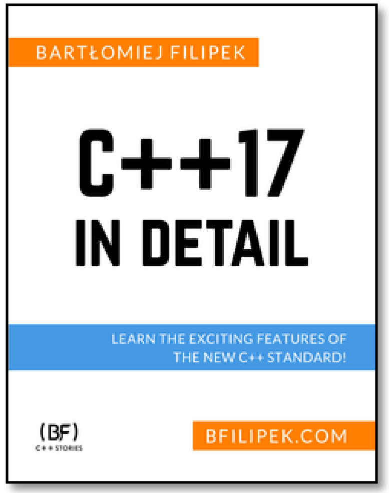
`string_view`, searchers and conversion routines

18th September 2018

Bartłomiej Filipek, bfilipek.com

About me

- See my coding blog at: www.bfilipek.com
- 11y+ experience
- Currently @Xara.com
  - Text related features for advanced document editors
- Somehow addicted to C++ ☺


BARTŁOMIEJ FILIPEK

C++17 IN DETAIL

LEARN THE EXCITING FEATURES OF THE NEW C++ STANDARD!

(BF) C++ STORIES

BFILIPEK.COM

C++17 In Detail

xara cloud

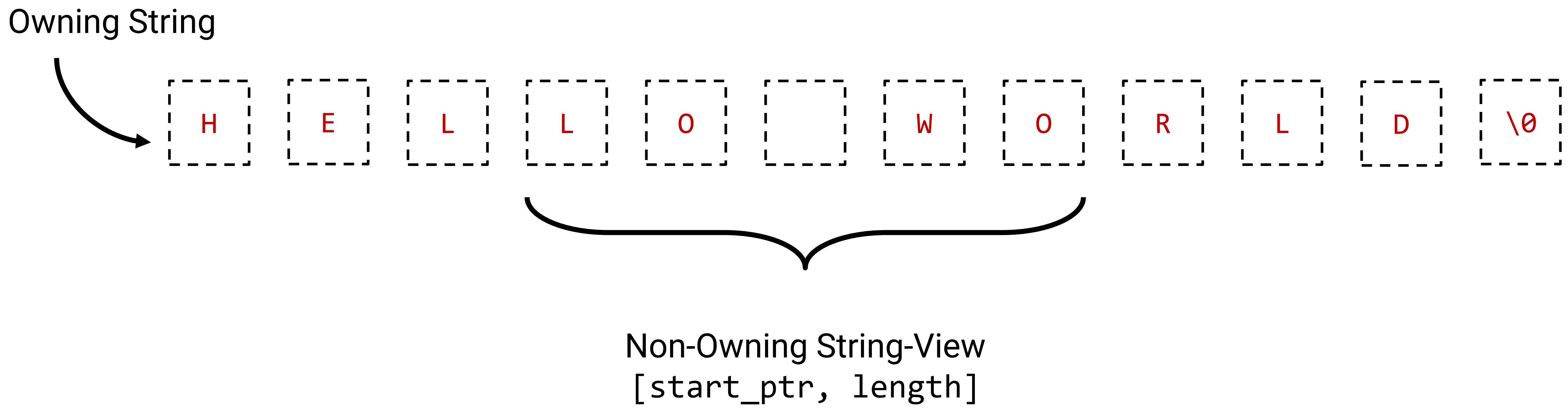
Getting Started

Xara Cloud Demo

The plan

- string_view
- Elementary conversion routines
- Searchers
- Summary

string_view


Owning String
H
E
L
L
O
W
O
R
L
D
\0
Non-Owning String-View
[start_ptr, length]

string_view

```
template<class CharT, class Traits = std::char_traits<CharT>>
class basic_string_view;


std::string_view      std::basic_string_view<char>
std::wstring_view     std::basic_string_view<wchar_t>
std::u16string_view   std::basic_string_view<char16_t>
std::u32string_view   std::basic_string_view<char32_t>
```

string_view creation

```
constexpr basic_string_view() noexcept;
constexpr basic_string_view(const basic_string_view& other) noexcept = default;
constexpr basic_string_view(const CharT* s, size_type count);
constexpr basic_string_view(const CharT* s);

// from string:
operator std::basic_string_view<CharT, Traits>() const noexcept;
```

```
const char* cstr = "Hello World";

std::string_view sv1 { cstr };
std::cout << sv1 << ", len: " << sv1.size() << '\n';
```

```
std::string str = "Hello String";
std::string_view sv3 = str;
std::cout << sv3 << ", len: " << sv3.size() << '\n';
```

Plus some more code…

string_view operations

operator[]
at
front
back
data
size/length
max_size
empty
remove_prefix
remove_suffix
swap

Bonus, C++20:
starts_with
ends_with

copy (not constexpr)
substr - complexity O(1) and not O(n) as in std::string
compare
find
rfind
find_first_of
find_last_of
find_first_not_of
find_last_not_of
operators for lexicography compare: ==, !=, <=, >=, <, >
operator <<

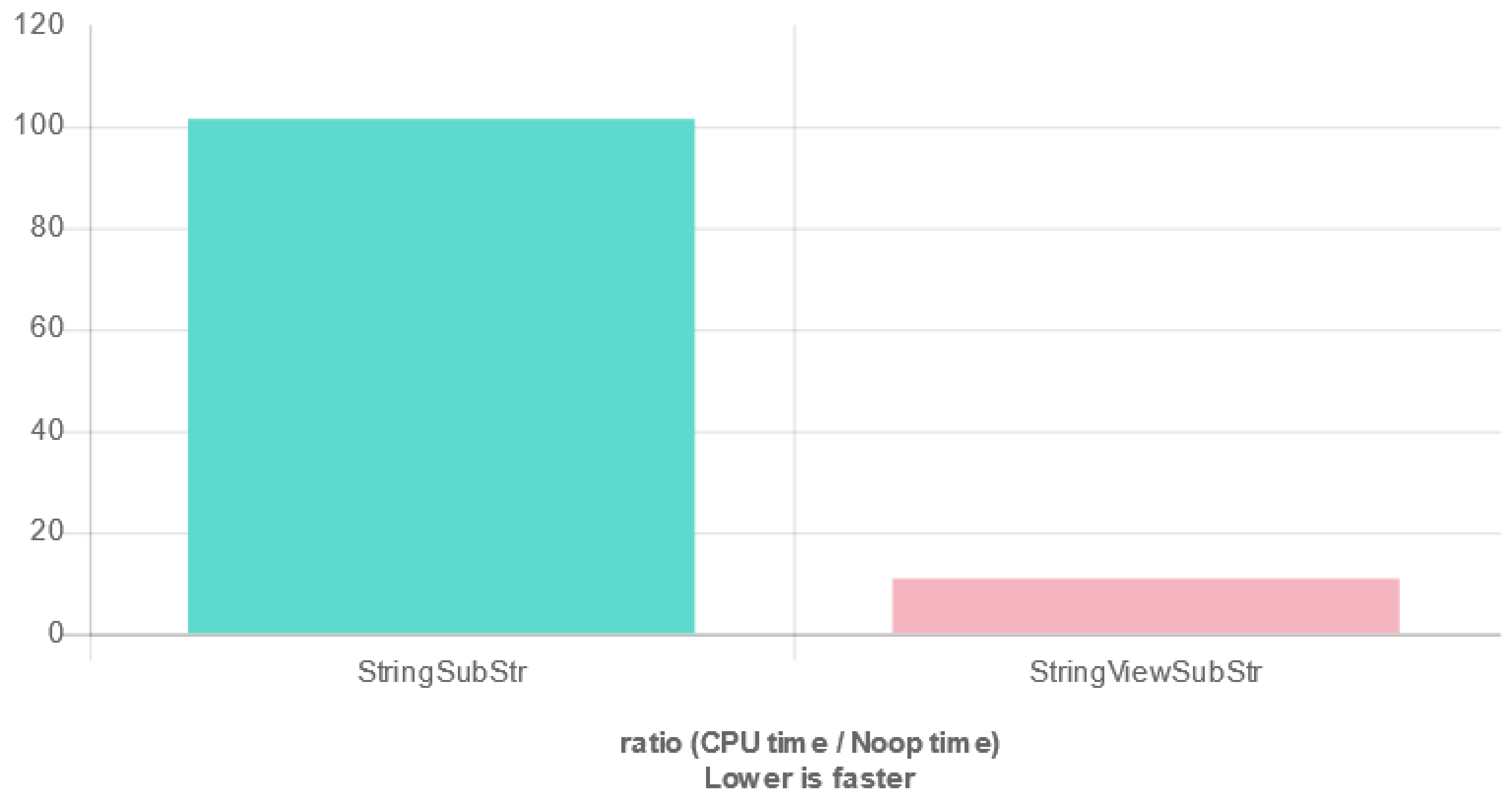
How many string copies?

```
std::string StartFromWordStr(const std::string& strArg, const std::string& word)
{
    return strArg.substr(strArg.find(word));
}

// call:
std::string str {"Hello Amazing Programming Environment" };
auto subStr = StartFromWordStr(str, "Programming Environment");
std::cout << subStr << "\n";
```

How many copies?

```
std::string_view StartFromWord(std::string_view str, std::string_view word)
{
    return str.substr(str.find(word));
}

// call:
std::string str {"Hello Amazing Programming Environment"};
auto subView = StartFromWord(str, "Programming Environment");
std::cout << subView << '\n';
```

Substr performance!



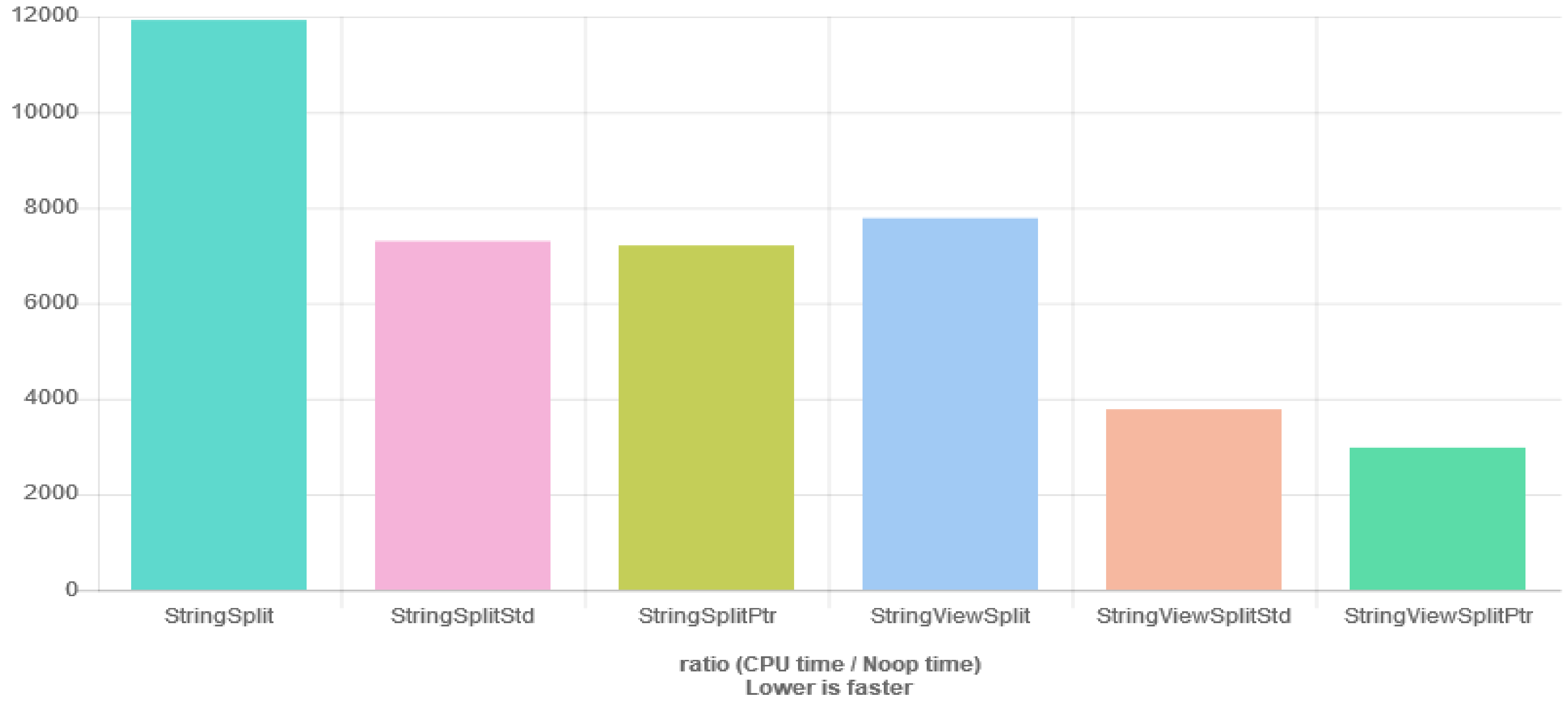
http://quick-bench.com/F1NGrjNtcNimqG2q6QzvHKPDpQY

More advanced example, string split

```
std::vector<std::string_view> splitSVStd(std::string_view strv, std::string_view delims = " ")
{
    std::vector<std::string_view> output;
    //output.reserve(strv.length() / 4);
    auto first = strv.begin();

    while (first != strv.end())
   {
        const auto second = std::find_first_of(first, std::cend(strv), std::cbegin(delims), std::cend(delims));

        if (first != second)
        {
            output.emplace_back(strv.substr(std::distance(strv.begin(), first), std::distance(first, second)));
        }

        if (second == strv.end())
            break;

        first = std::next(second);
    }

    return output;
}
```

https://www.bfilipek.com/2018/07/string-view-perf.html

Split Performance



http://quick-bench.com/mhyUI8Swxu3As-RafVUSVfEZd64

SSO

- Currently, it’s 15 characters in MSVC (VS 2017)/GCC (8.1) or 22 characters in Clang (6.0).

Risks!

- Non null terminated strings
- Temporary objects

Risks – non null terminated strings

```
std::string s = "Hello World";
std::cout << s.size() << '\n';
std::string_view sv = s;
std::cout << sv.size() << '\n';
// 11 & 11
```

```
std::string s = "Hello World";
std::cout << s.size() << '\n';
std::string_view sv = s;
auto sv2 = sv.substr(0, 5);
std::cout << sv2.data() << '\n';
// again "Hello World" !
```

```
std::string number = "123.456";
std::string_view svNum { number.data(), 3 };
auto f = atof(svNum.data()); // should be 123, but is 123.456!
std::cout << f << '\n';
```

```
std::string s { sv.data(), sv.size() };
```

Risks – temporary objects!

```
std::string_view StartFromWord(std::string_view str, std::string_view word)
{
    return str.substr(str.find(word)); // substr creates only a new view
}

auto str = "My Super"s;
auto sv = StartFromWord(str + " String", "Super");
```

Risks – temporary objects!

```
std::vector<int> GenerateVec()
{
    return std::vector<int>(5, 1);
}

const std::vector<int>& refv = GenerateVec();


for (auto &elem : GenerateVec())
{
    // ...
}
```

```
std::string func()
{
    std::string s;
    // build s...
    return s;
}

std::string_view sv = func();
// no temp lifetime extension!
```

```
std::vector<int> CreateVector() { ... }
std::string GetString() { return "Hello"; }
auto &x = CreateVector()[10]; // arbitrary element!
auto pStr = GetString().c_str();
```

https://wg21.link/p0936

string_view - summary

- What do you think?

Elementary Conversion Routines

facility	shortcomings
sprintf	format string, locale, buffer overrun
snprintf	format string, locale
sscanf	format string, locale
atol	locale, does not signal errors
strtol	locale, ignores whitespace and 0x prefix
strstream	locale, ignores whitespace
stringstream	locale, ignores whitespace, memory allocation
num_put / num_get facets	locale, virtual function
to_string	locale, memory allocation
stoi etc.	locale, memory allocation, ignores whitespace and 0x prefix, exception on error

https://wg21.link/p0067r5

Elementary Conversion Routines

	10,000,000 (coliru)	10,000,000 (Laptop1)	50,000,000 (Laptop1)	50,000,000 (Lenovo)	50,000,000 (Laptop1 x64)	50,000,000 (Laptop2)
atol()	616	546	2,994	4,202	3,311	4,068
strtoul()	459	454	2,421	2,560	2,660	2,852
from_chars()	244	136	745	884	1,027	972
>>	1,484	7,299	37,590	47,072	31,351	48,116
stoul()	1,029	798	4,115	4,636	6,328	5,210

https://www.fluentcpp.com/2018/07/24/how-to-convert-a-string-to-an-int-in-c/
https://www.fluentcpp.com/2018/07/27/how-to-efficiently-convert-a-string-to-an-int-in-c/

Elementary Conversion Routines

- `from_chars`, `to_chars`
- No locale
- No memory allocation
- C-style?

Elementary Conversion Routines

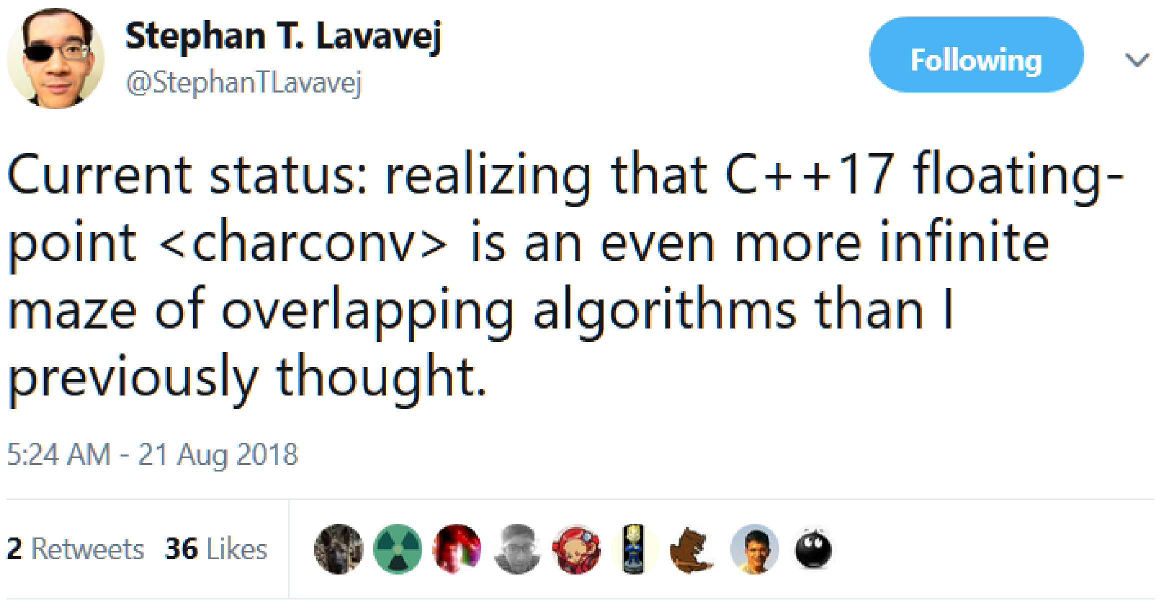
```
std::from_chars_result from_chars(const char* first, const char* last, int &value, int base = 10);

std::from_chars_result from_chars(const char* first, const char* last, float& value,
                                  std::chars_format fmt = std::chars_format::general);

struct from_chars_result {
    const char* ptr;
    std::errc ec;
};
```

```
std::to_chars_result to_chars(char* first, char* last, int value, int base = 10);

std::to_chars_result to_chars(char* first, char* last, float value,
std::chars_format fmt, int precision);

struct to_chars_result {
    char* ptr;
    std::errc ec;
};
```

Result

```
int value;
const auto res = std::from_chars(str.data(), str.data() + str.size(), value);
if (res.ec == std::errc::invalid_argument)
{
    std::cout << "invalid argument!, res.p distance: " << '\n';
}
else if (res.ec == std::errc::result_out_of_range)
{
    std::cout << "out of range! res.p distance: " << '\n';
}
else
{
    std::cout << "value: " << value << '\n';
}
```

Twitter



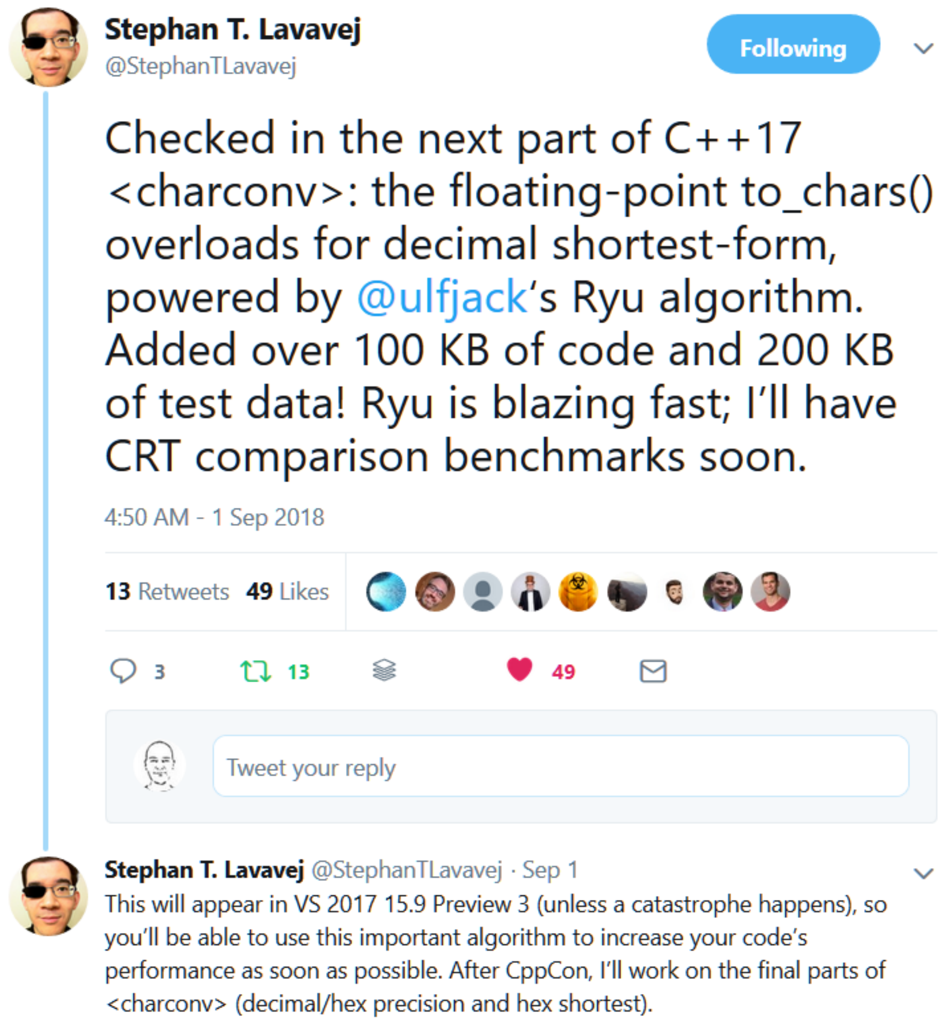
Stephan T. Lavavej
@StephanTLavavej

Following

Current status: realizing that C++17 floating-point <charconv> is an even more infinite maze of overlapping algorithms than I previously thought.

5:24 AM - 21 Aug 2018

2 Retweets **36** Likes



Stephan T. Lavavej
@StephanTLavavej

Following

Checked in the next part of C++17 <charconv>: the floating-point to_chars() overloads for decimal shortest-form, powered by @ulfjack's Ryu algorithm. Added over 100 KB of code and 200 KB of test data! Ryu is blazing fast; I'll have CRT comparison benchmarks soon.

4:50 AM - 1 Sep 2018

13 Retweets **49** Likes

3 13 49

Tweet your reply

Stephan T. Lavavej @StephanTLavavej · Sep 1

This will appear in VS 2017 15.9 Preview 3 (unless a catastrophe happens), so you'll be able to use this important algorithm to increase your code's performance as soon as possible. After CppCon, I'll work on the final parts of <charconv> (decimal/hex precision and hex shortest).

https://twitter.com/StephanTLavavej

Searchers

```
template< class ForwardIt1, class ForwardIt2 >
ForwardIt1 search(ForwardIt1 first, ForwardIt1 last, ForwardIt2 s_first, ForwardIt2 s_last);

template<class ForwardIterator, class Searcher>
ForwardIterator search(ForwardIterator first, ForwardIterator last, const Searcher& searcher);
```

- default_searcher
- boyer_moore_searcher
- boyer_moore_horspool_searcher

Searchers

```
P: word
T: There would have been a time for such a word
                ^
                |
            word
```

Good Suffix Rule

Bad character rule

http://www.cs.jhu.edu/~langmea/resources/lecture_notes/boyer_moore.pdf

https://www.youtube.com/watch?v=4Xyhb72LCX4

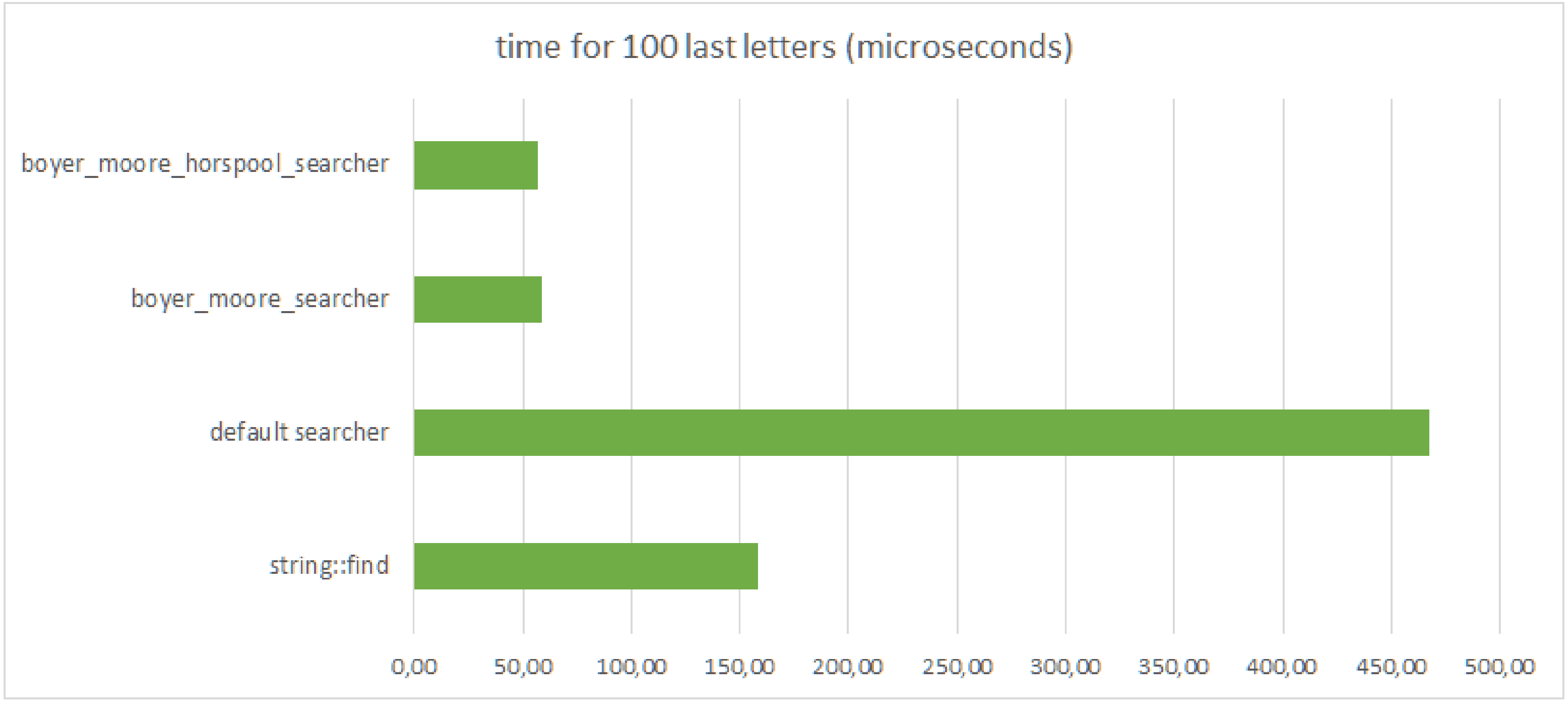
http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm

http://www-igm.univ-mlv.fr/%7Elecroq/string/node18.html

Searchers – some code

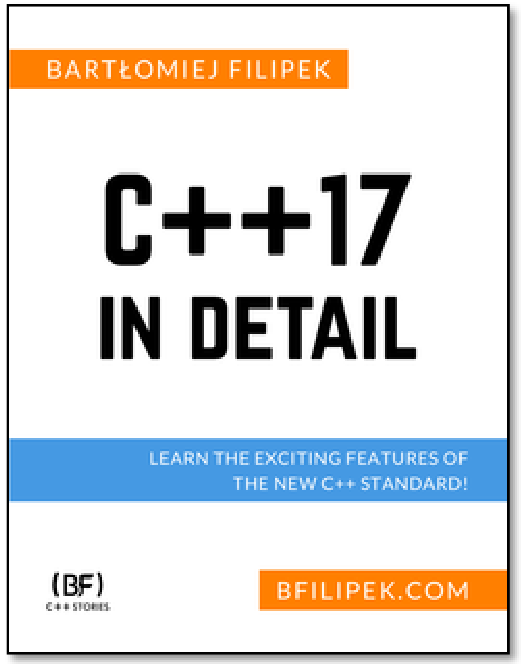
```
std::string testString = "Hello Super World";
std::string needle = "Super";
auto it = search(testString.begin(), testString.end(),
                 boyer_moore_searcher(needle.begin(), needle.end()));

if (it == testString.end())
    cout << "The string " << needle << " not found\n";
```

Searchers - performance



Summary

- string_view – good potential, with some risks!
- Conversion routines – finally!
- Searchers – nice addition!



50% off
Limited offer!

http://leanpub.com/cpp17indetail/c/cppcracow