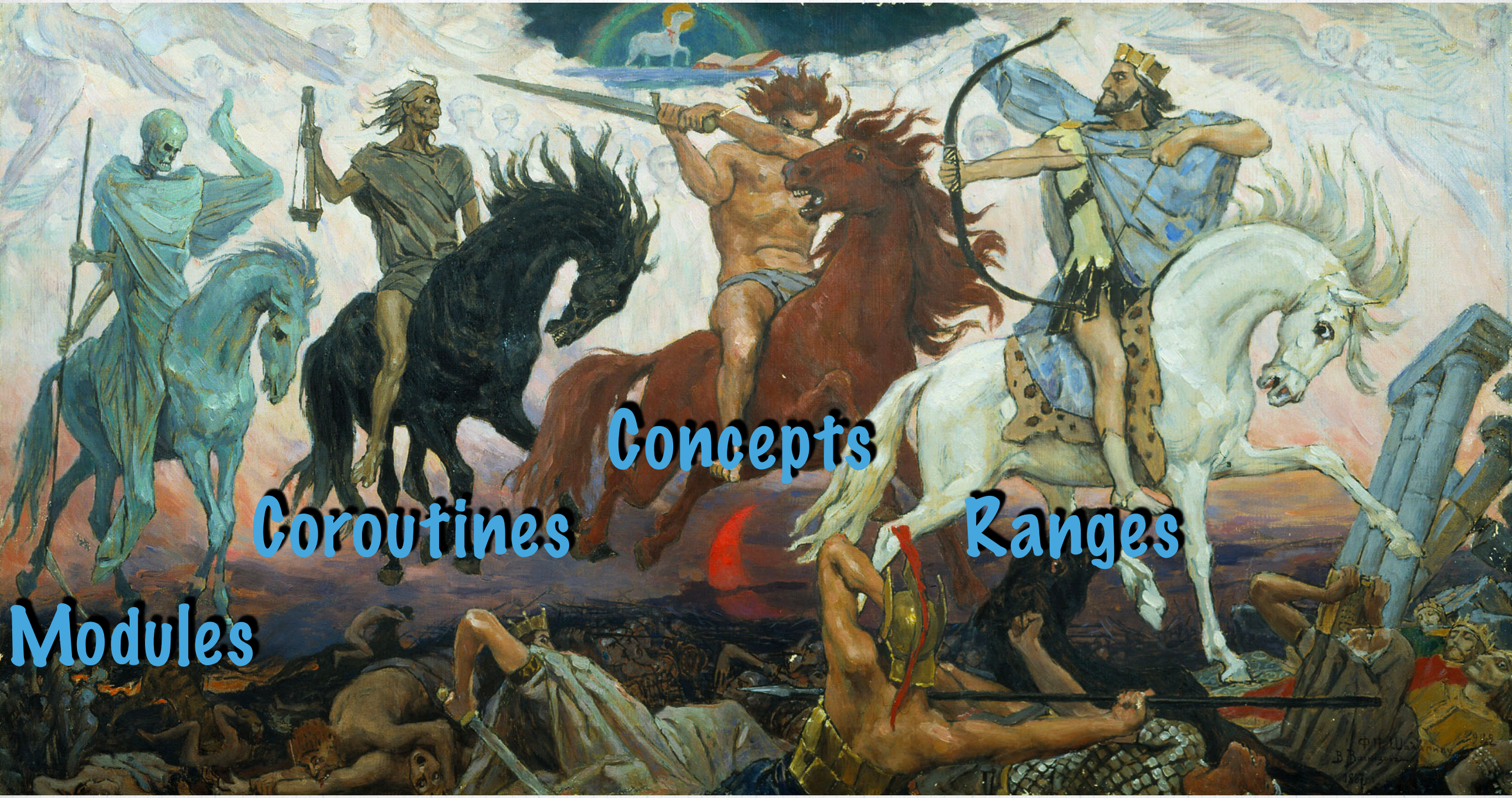
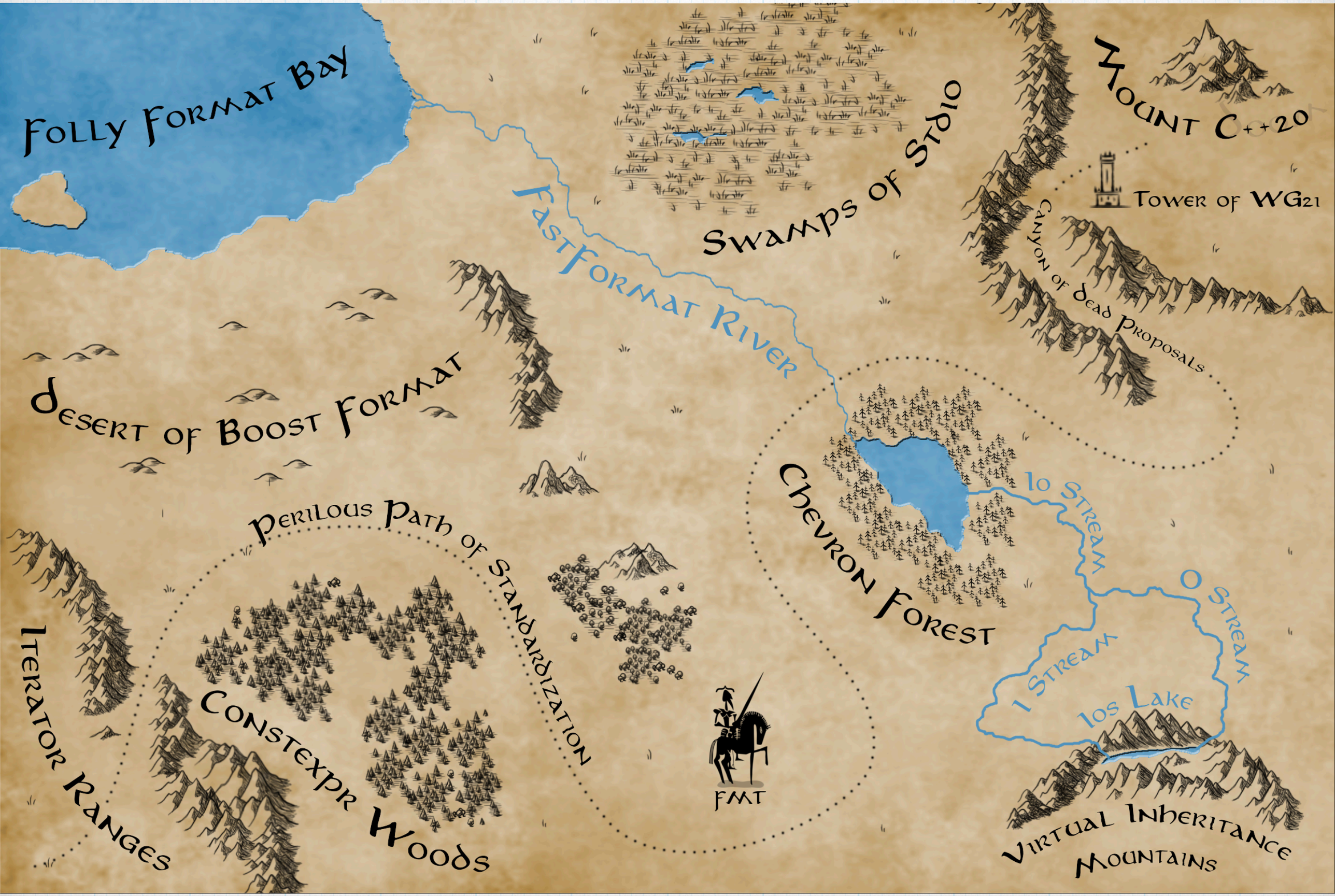
For your in<format>ion

Victor Zverovich

2 July 2020



Four big features of C++20, Vasnetsov (1887)


Folly Format Bay
Fast Format River
Swamps of Std10
Mount C++20
Tower of WG21
Canyon of Dead Proposals
Desert of Boost Format
Chevron Forest
Io Stream
O Stream
I Stream
Ios Lake
Perilous Path of Standardization
Iterator Ranges
Constexpr Woods
FMT
Virtual Inheritance Mountains

20.20 Formatting [format]

20.20.1 Header <format> synopsis [format.syn]

```
namespace std {
  // 20.20.3, formatting functions
  template<class... Args>
    string format(string_view fmt, const Args&... args);
  template<class... Args>
    wstring format(wstring_view fmt, const Args&... args);
  template<class... Args>
    string format(const locale& loc, string_view fmt, const Args&... args);
  template<class... Args>
    wstring format(const locale& loc, wstring_view fmt, const Args&... args);

  string vformat(string_view fmt, format_args args);
  wstring vformat(wstring_view fmt, wformat_args args);
  string vformat(const locale& loc, string_view fmt, format_args args);
  wstring vformat(const locale& loc, wstring_view fmt, wformat_args args);

  template<class Out, class... Args>
    Out format_to(Out out, string_view fmt, const Args&... args);
  template<class Out, class... Args>
    Out format_to(Out out, wstring_view fmt, const Args&... args);
  template<class Out, class... Args>
    Out format_to(Out out, const locale& loc, string_view fmt, const Args&... args);
  template<class Out, class... Args>
    Out format_to(Out out, const locale& loc, wstring_view fmt, const Args&... args);

  template<class Out>
    Out vformat_to(Out out, string_view fmt, format_args_t<Out, char> args);
```

Hello, <format>!

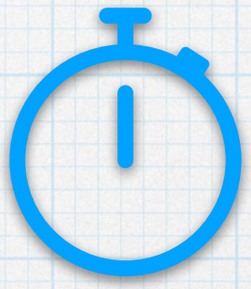
```
#include <format>

std::string message =
   std::format("The answer is {}.", 42);

// message == "The answer is 42."
```

Is this printf?

What if printf was designed in the 21st century?

Safe

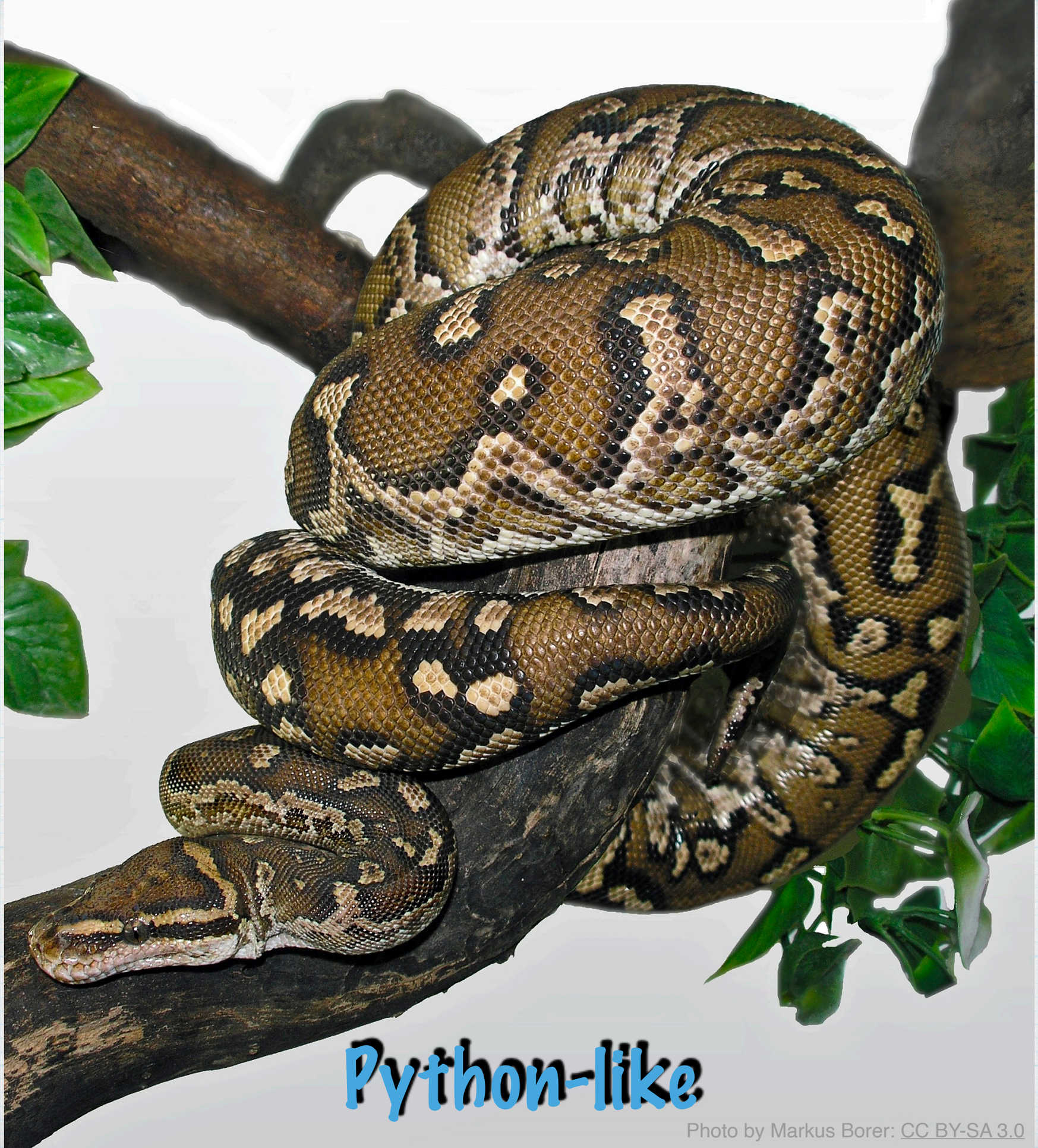
Extensible



Fast

std::format

* Type safe & extensible
* Expressive format syntax based on Python's format
* Designed for good perf, fast compilation and compact binary code
* Positional arguments
* Iterator support
* Locale-independent by default
* Date and time formatting



Syntax

* Format strings - general structure

```
format("Hello, {0:10s}", name)
```

* Format specifiers - presentation

```
format("Hello, {0:10s}", name)
```

* Conveys presentation not type

Syntax

Legacy-free:

```
printf("%d", my_int);
printf("%lld", my_long_long);
printf("%" PRIu64, my_int64);
```

```
format("{}", my_int);
format("{}", my_long_long);
format("{}", my_int64);
```

Semantical: conveys formatting, not type info, e.g. "d" means "decimal formatting" not "decimal int".

BYOG: bring your own grammar

Format strings

```
std::format("The answer is {}.", 42);
```

```
std::format("The answer is {}.", 42);
```

```
std::format("The answer is {}.", 42);
```

```
"The answer is {}."   42
```

```
std::format("The answer is {}.", 42);
```

```
"The answer is {}." "42"
```

```
std::format("The answer is {}.", 42);
```

```
"The answer is 42."
```

Positional arguments

```
std::format("I'd rather be {1} than {0}.", "right", "happy");
```

```
std::format("I'd rather be {1} than {0}.", "right", "happy");
```

```
std::format("I'd rather be {1} than {0}.", "right", "happy");
```

```
"I'd rather be {1} than {0}."  "right"  "happy"
```

```
std::format("I'd rather be {1} than {0}.", "right", "happy");
```

```
"I'd rather be happy than right."
```

Format specifiers

```
std::format("{:*^10.2f}", 1.2345);
```

fill

```
std::format("{:*^10.2f}", 1.2345);
```

fill

```
std::format("{:*^10.2f}", 1.2345);
```

alignment

fill

width

```
std::format("{:*^10.2f}", 1.2345);
```

alignment

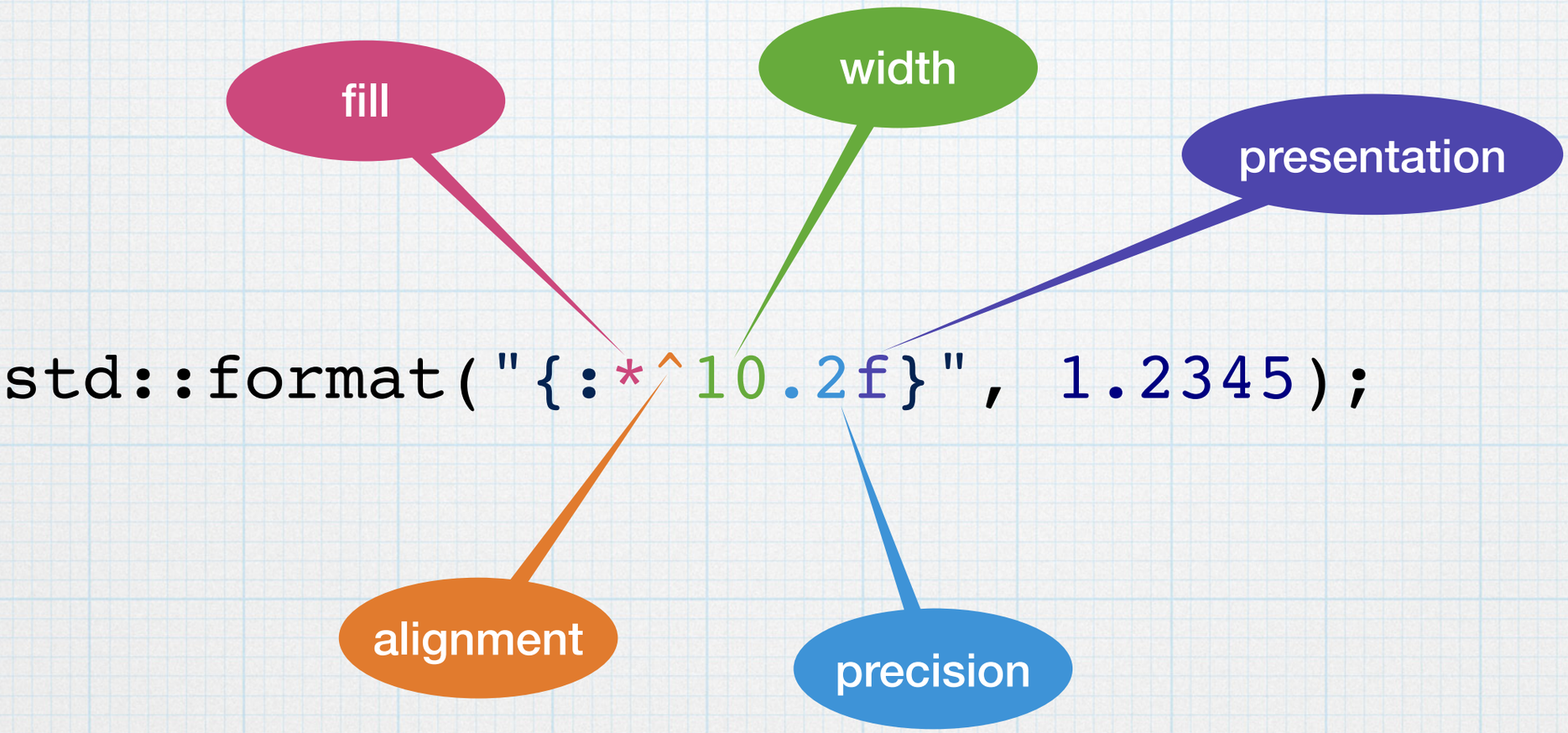
fill

width

```
std::format("{:*^10.2f}", 1.2345);
```

alignment

precision



Format 1.2345 in the fixed form rounded to 2 digits after the decimal point and pad with * to 10 characters aligned to the center:
1.23

```
std::format("{:*^10.2f}", 1.2345);
```

```
std::format("{:*^10.2f}", 1.2345);
```

```
"{:*^10}"       1.23
```

```
std::format("{:*^10.2f}", 1.2345);
```

```
"{:*^10}"      1.23
"{:*}"     "   1.23    "
```

```
std::format("{:*^10.2f}", 1.2345);
```

```
"{:*^10}"      1.23

"{:*}"     "    1.23    "

      "***1.23***"
```

Width & precision

Width:

```
format("{0:5}", 42);                    // "   42"
```

Dynamic width:

```
format("{0:{1}}", 42, 5);               // "   42"
```

Width & precision

Width:

```
format("{0:5}", 42);                    // "   42"
```

Dynamic width:

```
format("{0:{1}}", 42, 5);               // "   42"
```

Precision:

```
format("{0:.2}", 1.234);                // "1.2"
```

Dynamic precision:

```
format("{0:.{1}}", 1.234, 2);           // "1.2"
```

Fill & alignment

Alignment:

```
format("{:<20}", "left");      // "left                "
format("{:>20}", "right");     // "               right"
format("{:^20}", "centered");  // "      centered      "
```

Fill & alignment

Alignment:

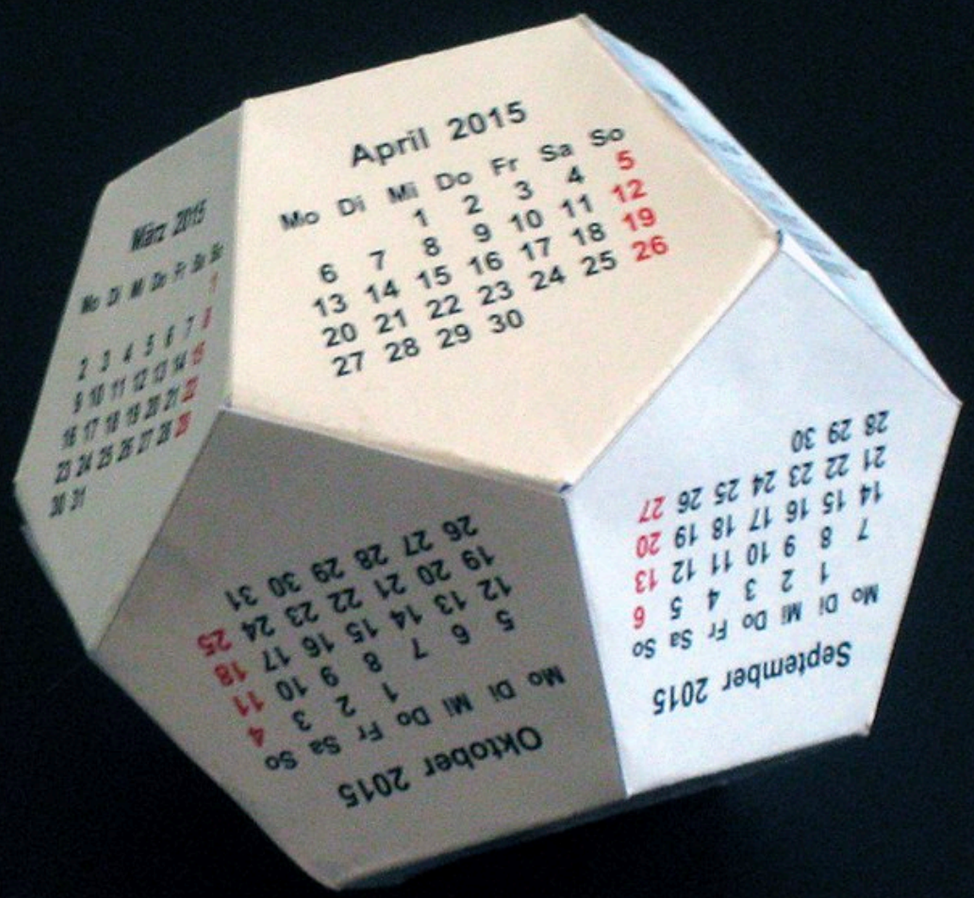
```
format("{:<20}", "left");     // "left                "
format("{:>20}", "right");    // "               right"
format("{:^20}", "centered"); // "      centered      "
```

Fill & alignment:

```
format("{:*^20}", "centered"); // "******centered******"
```

API

```
template <class... Args>
string format(string_view fmt, const Args&... args);

template <class Out, class... Args>
Out format_to(Out out, string_view fmt, const Args&... args);

template <class Out, class... Args>
format_to_n_result<Out> format_to_n(Out out,
                                    iter_difference_t<Out> n,
                                    string_view fmt,
                                    const Args&... args);

template <class... Args>
size_t formatted_size(string_view fmt, const Args&... args);
```



Calendar example

```
      January               February              March
         1  2  3  4  5                  1  2                  1
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   2  3  4  5  6  7  8
13 14 15 16 17 18 19  10 11 12 13 14 15 16   9 10 11 12 13 14 15
20 21 22 23 24 25 26  17 18 19 20 21 22 23  16 17 18 19 20 21 22
27 28 29 30 31        24 25 26 27 28 29     23 24 25 26 27 28 29
                                            30 31

      April                 May                   June
         1  2  3  4  5                     1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

      July                  August                September
         1  2  3  4  5                  1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31

      October               November              December
            1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
                      30
```

```
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
```

```
      January
            1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
  // Print the month name centered.
  auto field_width = 3;
  cout << format("{0:^{1}}", month_name(ym.month()), field_width * 7);
  co_yield false;
```

```
          January
              1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
  // Print the month name centered.
  auto field_width = 3;
  cout << format("{0:^{1}}", month_name(ym.month()), field_width * 7);
  co_yield false;

  // Print spaces until the first weekday.
  auto wd_index = year_month_weekday(ym/1).weekday().iso_encoding();
  cout << format("{0:{1}}", "", (wd_index - 1) * field_width);
```

```
          January
                1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```cpp
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
  // Print the month name centered.
  auto field_width = 3;
  cout << format("{0:^{1}}", month_name(ym.month()), field_width * 7);
  co_yield false;

  // Print spaces until the first weekday.
  auto wd_index = year_month_weekday(ym/1).weekday().iso_encoding();
  cout << format("{0:{1}}", "", (wd_index - 1) * field_width);

  // Print days.
  for (auto day = 1u, end = unsigned((ym/last).day()); day <= end; ++day) {
    cout << format("{:2} ", day);
    if (year_month_weekday(ym/day).weekday() == Sunday)
      co_yield day == end;
  }
```

```
        January
           1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
  // Print the month name centered.
  auto field_width = 3;
  cout << format("{0:^{1}}", month_name(ym.month()), field_width * 7);
  co_yield false;

  // Print spaces unti
  auto wd_index = year
  cout << format("{0:
```

```
January
          1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

```
  // Print days.
  for (auto day = 1u,
    cout << format("{
    if (year_month_we
      co_yield day == end;
  }

  // Print spaces after the last weekday.
  wd_index = year_month_weekday(ym/last).weekday().iso_encoding();
  for (int i = 0; i < 2; ++i, wd_index = 0) {
    cout << format("{0:{1}}", "", (7 - wd_index) * field_width);
    co_yield true;
  }
}
```

```
// Prints a month suspending at every end of line.
auto print_month(year_month ym) -> generator {
  // Print the month name centered.
  auto field_width = 3;
  cout << format("{0:^{1}}", month_name(ym.month()), field_width * 7);
  co_yield false;

  // Print spaces until the first weekday.
  auto wd_index = year_month_weekday(ym/1).weekday().iso_encoding();
  cout << format("{0:{1}}", "", (wd_index - 1) * field_width);

  // Print days.
  for (auto day = 1u, end = unsigned((ym/last).day()); day <= end; ++day) {
    cout << format("{:2} ", day);
    if (year_month_weekday(ym/day).weekday() == Sunday)
      co_yield day == end;
  }

  // Print spaces after the last weekday.
  wd_index = year_month_weekday(ym/last).weekday().iso_encoding();
  for (int i = 0; i < 2; ++i, wd_index = 0) {
    cout << format("{0:{1}}", "", (7 - wd_index) * field_width);
    co_yield true;
  }
}
```

```
void print_calendar_row(year_month start,
                        int num_months) {
```

```
      January                February                 March
         1  2  3  4  5                     1  2                     1
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   2  3  4  5  6  7  8
13 14 15 16 17 18 19  10 11 12 13 14 15 16   9 10 11 12 13 14 15
20 21 22 23 24 25 26  17 18 19 20 21 22 23  16 17 18 19 20 21 22
27 28 29 30 31        24 25 26 27 28 29     23 24 25 26 27 28 29
                                            30 31
```

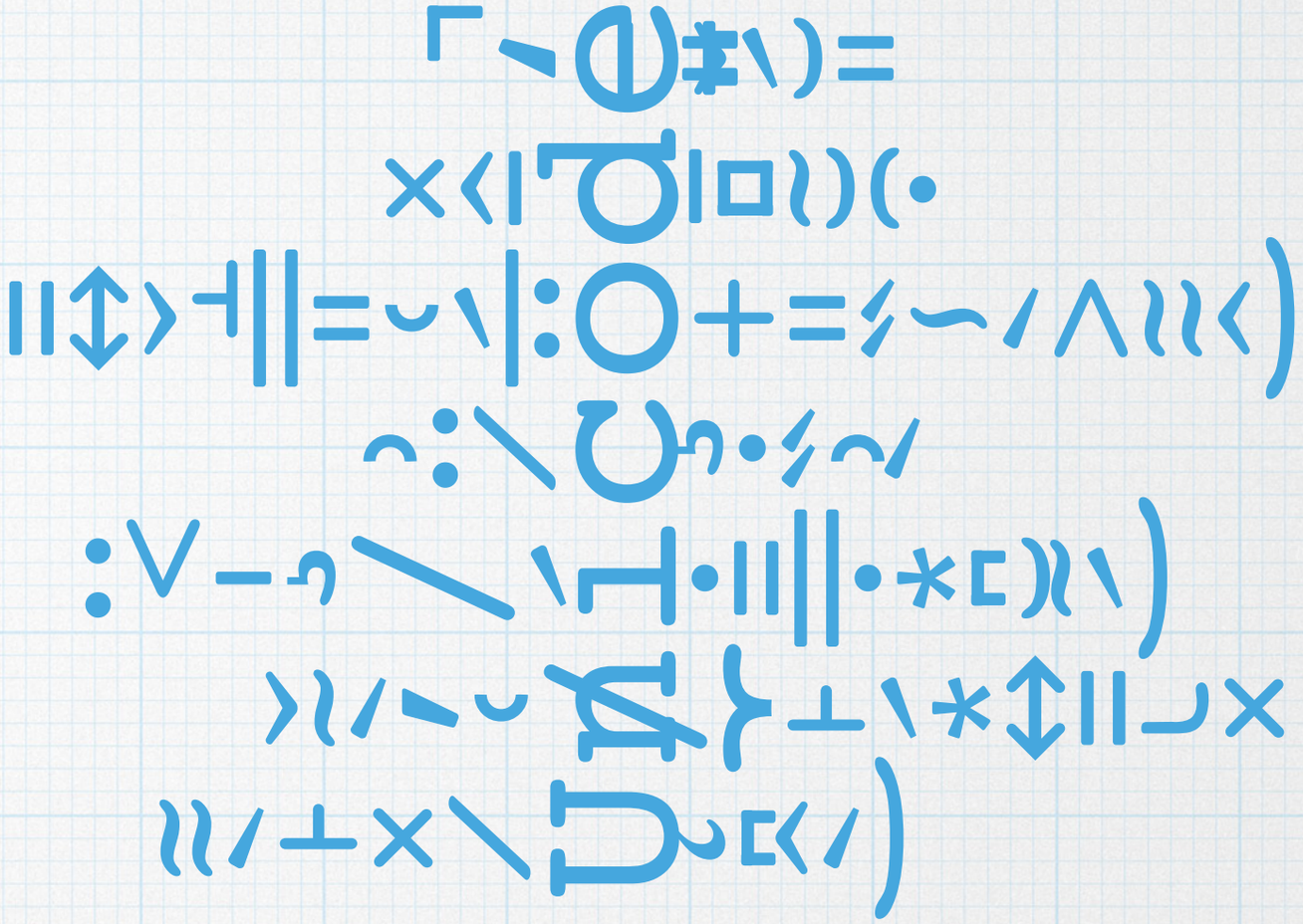
```
void print_calendar_row(year_month start,
                        int num_months) {
  std::vector<generator> gens;
  for (int i = 0; i < num_months; ++i)
    gens.push_back(
      print_month(start + months(i)));
  for (;;) {
    bool done = true;
    for (auto& g: gens) {
      cout << ' ';
      done &= g.next();
    }
    cout << `\n';
    if (done) break;
  }
  cout << '\n';
}
```

```
int main() {
  int months_per_row = 3;
  for (auto start = 2020_y/jan;
       start <= 2020_y/dec;
       start += months(months_per_row)) {
    print_calendar_row(start, months_per_row);
  }
}
```

```
      January               February              March
         1  2  3  4  5                  1  2                  1
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   2  3  4  5  6  7  8
13 14 15 16 17 18 19  10 11 12 13 14 15 16   9 10 11 12 13 14 15
20 21 22 23 24 25 26  17 18 19 20 21 22 23  16 17 18 19 20 21 22
27 28 29 30 31        24 25 26 27 28 29     23 24 25 26 27 28 29
                                            30 31

      April                 May                   June
         1  2  3  4  5                     1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

      July                  August                September
         1  2  3  4  5                  1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31

      October               November              December
            1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
                      30
```

```
const char* const month_names[] = {
   "January", "February", "March",
   "April",   "May",      "June",
   "July",    "August",   "September",
   "October", "November", "December"
};

auto month_name(date::month mon) {
   return month_names[unsigned(mon) - 1];
}
```

```
class generator {
 public:
  struct promise_type;

 private:
  using handle = std::experimental::coroutine_handle<promise_type>;
  handle handle_;

 public:
  struct promise_type {
    bool value;

    auto get_return_object() { return handle::from_promise(*this); }
    auto initial_suspend() { return std::experimental::suspend_always(); }
    auto final_suspend() { return std::experimental::suspend_always(); }
    void unhandled_exception() { std::terminate(); }
    void return_void() {}

    auto yield_value(bool val) {
      value = val;
      return std::experimental::suspend_always();
    }
  };

  generator(handle h) : handle_(h) {}

  bool next() {
    handle_.resume();
    return handle_.promise().value;
  }
};
```



Январь

		🎁	2	3	4	5
6	7	8	9	10	11	12
13	🎄	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Февраль

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	🔫
24	25	26	27	28	29	

Март

						1
2	3	4	5	6	7	💃
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Апрель

		1	2	3	4	5
6	7	8	9	10	11	🚀
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Май

				👷	2	3
4	5	6	7	8	🎖️	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Июнь

1	2	3	4	5	6	7
8	9	10	11	🇷🇺	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Июль

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Август

						1 🛸
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Сентябрь

		📚	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Октябрь

		1	2	3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Ноябрь

						1
2	3	🤝	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Декабрь

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	🎅			

```
const char* const month_names[] = {
  "Январь",  "Февраль", "Март",
  "Апрель",  "Май",     "Июнь",
  "Июль",    "Август",  "Сентябрь",
  "Октябрь", "Ноябрь",  "Декабрь"
};

auto month_name(date::month mon) {
  return month_names[unsigned(mon) - 1];
}
```

```cpp
std::map<month_day, std::string> special_days{
    {jan/ 1, "🎁"},  // New Year's Day
    {jan/14, "🎄"},  // Old New Year
    {feb/23, "🔫"},  // Defender of the Fatherland Day
    {mar/ 8, "💃"},  // International Women's Day
    {apr/12, "🚀"},  // Cosmonautics Day
    {may/ 1, "👷"},  // International Workers' Day
    {may/ 9, "🎖️ "}, // Victory Day
    {jun/12, "🇷🇺 "}, // Russia Day
    {aug/ 2, "🛟"},  // Airborne Forces Day
    {sep/ 1, "📚"},  // Knowledge Day
    {nov/ 4, "🤝"},  // Unity Day
    {dec/31, "🎅"}   // New Year's Eve
};
```

```
// Print days.
for (auto day = 1u, end = unsigned((ym/last).day());
     day <= end; ++day) {

  auto special = special_days.find(ym.month()/day);
  if (special == special_days.end())
    cout << format("{:2} ", day);
  else
    cout << format("{} ", special->second);

  if (year_month_weekday(ym/day).weekday() == Sunday)
    co_yield day == end;
}
```

Январь

		🎁	2	3	4	5
6	7	8	9	10	11	12
13	🎄	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Февраль

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	🔫
24	25	26	27	28	29	

Март

						1
2	3	4	5	6	7	💃
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Апрель

		1	2	3	4	5
6	7	8	9	10	11	🚀
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Май

				👷	2	3
4	5	6	7	8	🎖️	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Июнь

1	2	3	4	5	6	7
8	9	10	11	🇷🇺	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Июль

		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Август

						1 🛸
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Сентябрь

		📚	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Октябрь

		1	2	3	4	
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Ноябрь

						1
2	3	🤝	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Декабрь

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	🎅			

Dates and times

Photo by Alexandre Mirgorodski: source

Dates and times

`std::format` seamlessly integrates with chrono - you'll be able to format durations and dates using `strftime`-like format syntax:

Dates and times

`std::format` seamlessly integrates with chrono - you'll be able to format durations and dates using `strftime`-like format syntax:

```
using namespace std::chrono_literals;
auto s = std::format("{}", 42ms);
// s == "42ms"
```

Dates and times

`std::format` seamlessly integrates with chrono - you'll be able to format durations and dates using `strftime`-like format syntax:

```
using namespace std::chrono_literals;
auto s = std::format("{}", 42ms);
// s == "42ms"

s = std::format("{:%H:%M:%S}", 10'000s);
// s == "02:46:40"
```

Dates and times

`std::format` seamlessly integrates with chrono - you'll be able to format durations and dates using `strftime`-like format syntax:

```
using namespace std::chrono_literals;
auto s = std::format("{}", 42ms);
// s == "42ms"

s = std::format("{:%H:%M:%S}", 10'000s);
// s == "02:46:40"

s = std::format("{}", 2020y/July/2);
// s == "2020-07-02"
```

Extensibility

Unlike `(s)printf`, `std::format` supports user-defined types:

```
enum class color {red, green, blue};

template <>
struct std::formatter<color>: formatter<string_view> {
  // parse is inherited from formatter<string_view>.
  template <typename FormatContext>
  auto format(color c, FormatContext& ctx) {
    string_view name = "unknown";
    switch (c) {
    case color::red:   name = "red"; break;
    case color::green: name = "green"; break;
    case color::blue:  name = "blue"; break;
    }
    return formatter<string_view>::format(name, ctx);
  }
};

auto s = std::format("{:5}", color::red); // s == "red  "
```

Extensibility

`std::format` makes it easier to handle multi-component objects:

```
std::cout << std::left << std::setw(8) << Sunday[2]
          << "game\n";
// prints "Sun     [2]game" with usual implementation of <<.
// Note misaligned index and width applying only to
// Sunday
```

vs

```
std::cout << std::format("{0:<8}{1}\n", Sunday[2], "game");
// prints "Sun[2]  game"
```

This was prevented by defining `operator<<` in terms of `std::format` for chrono types in C++20.

No strings attached

We can have zero dynamic memory allocations using `format_to` & `format_to_n`:

```
char buf[10];
auto result = format_to_n(
    buf, sizeof(buf), "{}", 42);
// result.out - end iterator
// result.size - total output size
```

Replaces `sprintf` & `snprintf`.

Can get output size with `formatted_size`.

Numeric formatting

Defined in terms of `to_chars`.

```
auto s = std::format("{}", 1.0 / 3.0);
// s == 0.33333333333333333
```

Unlike (s)printf and iostreams:

* doesn't use locale by default
* provides round-trip guarantee by default

😱Locales😱

* All formatting is locale-independent by default giving deterministic output and better performance

* Locales are supported as an opt-in via separate format specifiers such as `L` for numbers

```
auto loc = std::locale("ru_RU.UTF-8");
auto s = std::format(loc, "{:L}", 1000000);
// s == "1 000 000"
```

Code bloat

```
std::string to_string(int n) {
    std::ostringstream oss;
    oss << n;
    return oss.str();
}
```

```cpp
std::string to_string(int n) {
  std::ostringstream oss;
  oss << n;
  return oss.str();
}
```

https://godbolt.org/z/JRgeZA

```
.LC0:
    .string "basic_string::replace"
.LC1:
    .string "%s: __pos (which is %zu) > this->size() (which is %zu)"
.LC2:
    .string "basic_string::_M_replace"
.LC3:
    .string "basic_string::_M_create"
to_string[abi:cxx11](int):
    push    r15
    push    r14
    push    r13
    mov     r13d, esi
    push    r12
    mov     r12, rdi
    push    rbp
    push    rbx
    sub     rsp, 392
    lea     rdi, [rsp+112]
    call    std::ios_base::ios_base() [base object constructor]
    mov     rbx, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+8]
    xor     eax, eax
    xor     esi, esi
    pxor    xmm0, xmm0
    mov     r14, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+16]
    mov     WORD PTR [rsp+336], ax
    movups  XMMWORD PTR [rsp+344], xmm0
    movups  XMMWORD PTR [rsp+360], xmm0
    mov     rax, QWORD PTR [rbx-24]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVSt9basic_iosIcSt11char_traitsIcEE+16
    lea     rdi, [rsp+rax]
    mov     QWORD PTR [rsp], rbx
    mov     QWORD PTR [rsp+328], 0
    mov     QWORD PTR [rdi], r14
    call    std::basic_ios<char, std::char_traits<char>
>::init(std::basic_streambuf<char, std::char_traits<char> >*)
    mov     ebp, OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
24
    lea     rdi, [rsp+64]
    lea     r15, [rsp+96]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
64
    movq    xmm0, rbp
```

```
std::string to_string(int n) {
  std::ostringstream oss;
  oss << n;
  return oss.str();
}
```

https://godbolt.org/z/JRgeZA

```
.LC0:
    .string "basic_string::replace"
.LC1:
    .string "%s: __pos (which is %zu) > this->size() (which is %zu)"
.LC2:
    .string "basic_string::_M_replace"
.LC3:
    .string "basic_string::_M_create"
to_string[abi:cxx11](int):
    push    r15
    push    r14
    push    r13
    mov     r13d, esi
    push    r12
    mov     r12, rdi
    push    rbp
    push    rbx
    sub     rsp, 392
    lea     rdi, [rsp+112]
    call    std::ios_base::ios_base() [base object constructor]
    mov     rbx, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+8]
    xor     eax, eax
    xor     esi, esi
    pxor    xmm0, xmm0
    mov     r14, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+16]
    mov     WORD PTR [rsp+336], ax
    movups  XMMWORD PTR [rsp+344], xmm0
    movups  XMMWORD PTR [rsp+360], xmm0
    mov     rax, QWORD PTR [rbx-24]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVSt9basic_iosIcSt11char_traitsIcEE+16
    lea     rdi, [rsp+rax]
    mov     QWORD PTR [rsp], rbx
    mov     QWORD PTR [rsp+328], 0
    mov     QWORD PTR [rdi], r14
    call    std::basic_ios<char, std::char_traits<char>
>::init(std::basic_streambuf<char, std::char_traits<char> >*)
    mov     ebp, OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
24
    lea     rdi, [rsp+64]
    lea     r15, [rsp+96]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
64
    movq    xmm0, rbp
```

```cpp
std::string to_string(int n) {
  std::ostringstream oss;
  oss << n;
  return oss.str();
}
```

https://godbolt.org/z/JRgeZA

```
.LC0:
    .string "basic_string::replace"
.LC1:
    .string "%s: __pos (which is %zu) > this->size() (which is %zu)"
.LC2:
    .string "basic_string::_M_replace"
.LC3:
    .string "basic_string::_M_create"
to_string[abi:cxx11](int):
    push    r15
    push    r14
    push    r13
    mov     r13d, esi
    push    r12
    mov     r12, rdi
    push    rbp
    push    rbx
    sub     rsp, 392
    lea     rdi, [rsp+112]
    call    std::ios_base::ios_base() [base object constructor]
    mov     rbx, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+8]
    xor     eax, eax
    xor     esi, esi
    pxor    xmm0, xmm0
    mov     r14, QWORD PTR VTT for
std::__cxx11::basic_ostringstream<char, std::char_traits<char>,
std::allocator<char> >[rip+16]
    mov     WORD PTR [rsp+336], ax
    movups  XMMWORD PTR [rsp+344], xmm0
    movups  XMMWORD PTR [rsp+360], xmm0
    mov     rax, QWORD PTR [rbx-24]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVSt9basic_iosIcSt11char_traitsIcEE+16
    lea     rdi, [rsp+rax]
    mov     QWORD PTR [rsp], rbx
    mov     QWORD PTR [rsp+328], 0
    mov     QWORD PTR [rdi], r14
    call    std::basic_ios<char, std::char_traits<char>
>::init(std::basic_streambuf<char, std::char_traits<char> >*)
    mov     ebp, OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
24
    lea     rdi, [rsp+64]
    lea     r15, [rsp+96]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESaIcEEE+
64
    movq    xmm0, rbp
    movhps  xmm0, QWORD PTR .LC4[rip]
    movaps  XMMWORD PTR [rsp], xmm0
    pxor    xmm0, xmm0
    movaps  XMMWORD PTR [rsp+16], xmm0
    movaps  XMMWORD PTR [rsp+32], xmm0
    movaps  XMMWORD PTR [rsp+48], xmm0
    call    std::locale::locale() [complete object constructor]
    lea     rsi, [rsp+8]
    lea     rdi, [rsp+112]
    mov     QWORD PTR [rsp+8], OFFSET FLAT:vtable for
std::__cxx11::basic_stringbuf<char, std::char_traits<char>,
std::allocator<char> >+16
    mov     DWORD PTR [rsp+72], 16
    mov     QWORD PTR [rsp+80], r15
    mov     QWORD PTR [rsp+88], 0
    mov     BYTE PTR [rsp+96], 0
    call    std::basic_ios<char, std::char_traits<char>
>::init(std::basic_streambuf<char, std::char_traits<char> >*)
    mov     esi, r13d
    mov     rdi, rsp
    call    std::basic_ostream<char, std::char_traits<char>
>::operator<<(int)
    mov     rax, QWORD PTR [rsp+48]
    lea     r13, [r12+16]
    mov     QWORD PTR [r12+8], 0
    mov     QWORD PTR [r12], r13
    mov     BYTE PTR [r12+16], 0
    test    rax, rax
    je      .L120
    mov     r8, QWORD PTR [rsp+32]
    mov     rcx, QWORD PTR [rsp+40]
    cmp     rax, r8
    ja      .L135
    sub     r8, rcx
    xor     edx, edx
    xor     esi, esi
    mov     rdi, r12
    call    std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char>
>::replace(unsigned long, unsigned long, char const*, unsigned
long) [clone .isra.0]
.L122:
    movq    xmm0, rbp
    mov     rdi, QWORD PTR [rsp+80]
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVNSt7__cxx1119basic_ostringstreamIcSt11char_traitsIcESa
IcEEE+64
    movhps  xmm0, QWORD PTR .LC5[rip]
    movaps  XMMWORD PTR [rsp], xmm0
    cmp     rdi, r15
    je      .L124
    mov     rax, QWORD PTR [rsp+96]
    lea     rsi, [rax+1]
    call    operator delete(void*, unsigned long)
.L124:
    mov     QWORD PTR [rsp+8], OFFSET FLAT:vtable for
std::basic_streambuf<char, std::char_traits<char> >+16
    lea     rdi, [rsp+64]
    call    std::locale::~locale() [complete object destructor]
    mov     rax, QWORD PTR [rbx-24]
    lea     rdi, [rsp+112]
    mov     QWORD PTR [rsp], rbx
    mov     QWORD PTR [rsp+rax], r14
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVSt9basic_iosIcSt11char_traitsIcEE+16
    call    std::ios_base::~ios_base() [base object destructor]
    add     rsp, 392
    mov     rax, r12
    pop     rbx
    pop     rbp
    pop     r12
    pop     r13
    pop     r14
    pop     r15
    ret
.L135:
    sub     rax, rcx
    xor     edx, edx
    xor     esi, esi
    mov     rdi, r12
    mov     r8, rax
    call    std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char>
>::replace(unsigned long, unsigned long, char const*, unsigned
long) [clone .isra.0]
    jmp     .L122
.L120:
    lea     rsi, [rsp+80]
    mov     rdi, r12
    call    std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char>
>::_M_assign(std::__cxx11::basic_string<char,
std::char_traits<char>, std::allocator<char> > const&)
    jmp     .L122
    mov     rbp, rax
    jmp     .L125
    mov     rbp, rax
    jmp     .L127
    mov     rbp, rax
    jmp     .L117
    mov     rbp, rax
    jmp     .L119
to_string[abi:cxx11](int) [clone .cold]:
    mov     rdi, rsp
    call    std::__cxx11::basic_ostringstream<char,
std::char_traits<char>, std::allocator<char>
>::~basic_ostringstream() [complete object destructor]
    mov     rdi, rbp
    call    _Unwind_Resume
.L117:
    mov     QWORD PTR [rsp+8], OFFSET FLAT:vtable for
std::__cxx11::basic_stringbuf<char, std::char_traits<char>,
std::allocator<char> >+16
    mov     rdi, QWORD PTR [rsp+80]
    cmp     rdi, r15
    je      .L118
    mov     rax, QWORD PTR [rsp+96]
    lea     rsi, [rax+1]
    call    operator delete(void*, unsigned long)
.L118:
    mov     QWORD PTR [rsp+8], OFFSET FLAT:vtable for
std::basic_streambuf<char, std::char_traits<char> >+16
    lea     rdi, [rsp+64]
    call    std::locale::~locale() [complete object destructor]
    mov     rax, QWORD PTR [rbx-24]
    mov     QWORD PTR [rsp], rbx
    mov     QWORD PTR [rsp+rax], r14
.L119:
    mov     QWORD PTR [rsp+112], OFFSET
FLAT:_ZTVSt9basic_iosIcSt11char_traitsIcEE+16
    lea     rdi, [rsp+112]
    call    std::ios_base::~ios_base() [base object destructor]
    mov     rdi, rbp
    call    _Unwind_Resume
.LC4:
    .quad   vtable for std::basic_streambuf<char,
std::char_traits<char> >+16
.LC5:
    .quad   vtable for std::__cxx11::basic_stringbuf<char,
std::char_traits<char>, std::allocator<char> >+16
```

```
std::string to_string(int n) {
  return fmt::format("{}", n);
}
```

Type erasure goes brrr

```cpp
std::string to_string(int n) {
  return fmt::format("{}", n);
}
```

https://godbolt.org/z/gDHXsv

```
.LC0:
        .string "{}"
to_string[abi:cxx11](int):
        push    r12
        mov     ecx, 1
        mov     edx, 2
        mov     r12, rdi
        sub     rsp, 16
        mov     DWORD PTR [rsp], esi
        mov     r8, rsp
        mov     esi, OFFSET FLAT:.LC0
        call    fmt::...::vformat[abi:cxx11](
          fmt::v6::basic_string_view<char>,
          fmt::v6::format_args)
        add     rsp, 16
        mov     rax, r12
        pop     r12
        ret
```

Performance

Performance

Much faster than iostreams by design due to

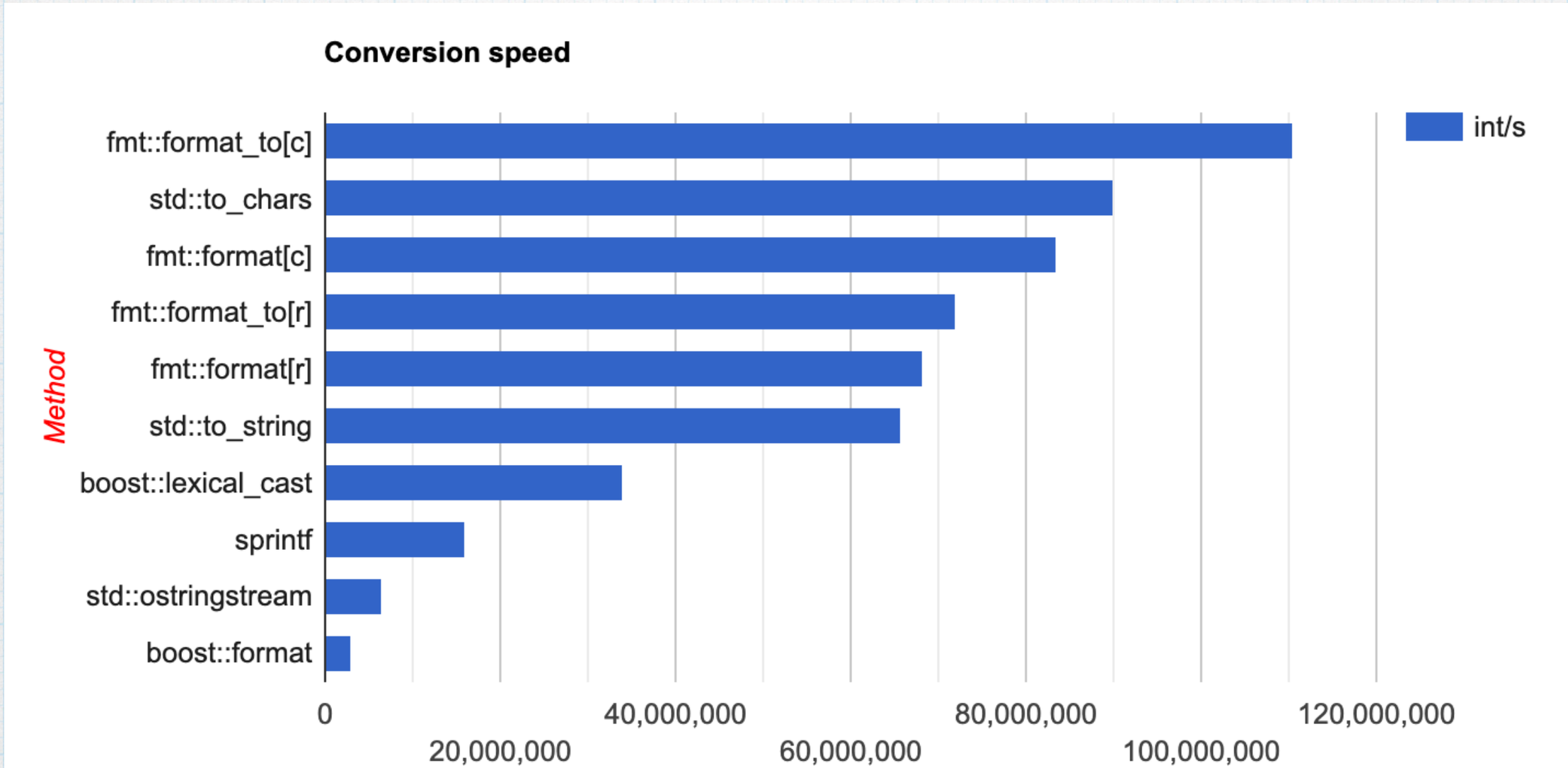
* Stateless API
* Locale-independence by default
* Fewer function calls
* Zero allocations (`format_to[_n]`)

Performance

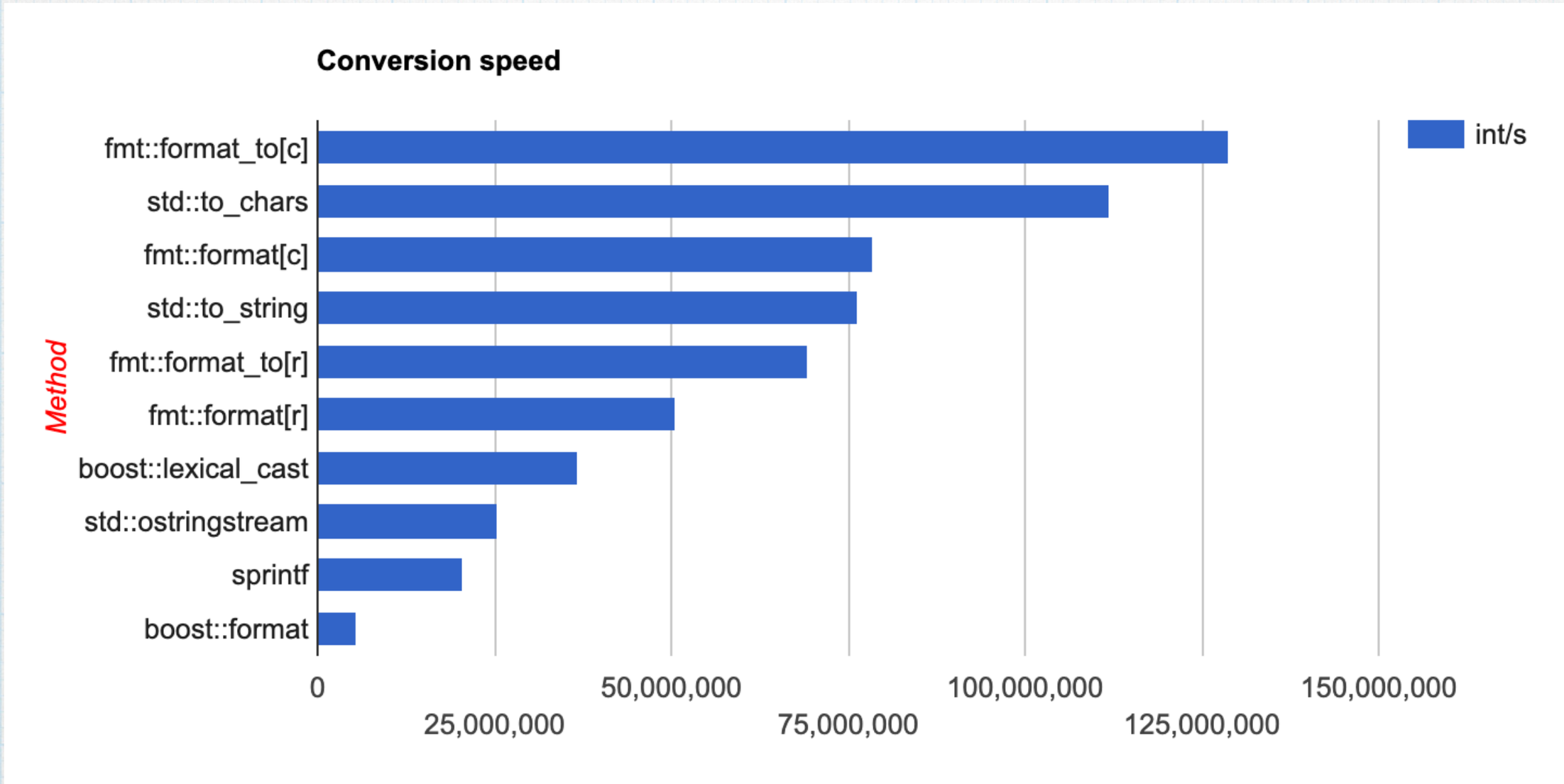
Faster than stdio by design due to

* Simpler format syntax (no legacy specs)
* Locale-independence by default
* More efficient argument handling

macOS/clang/libc++



Linux/gcc/libstdc++



Performance

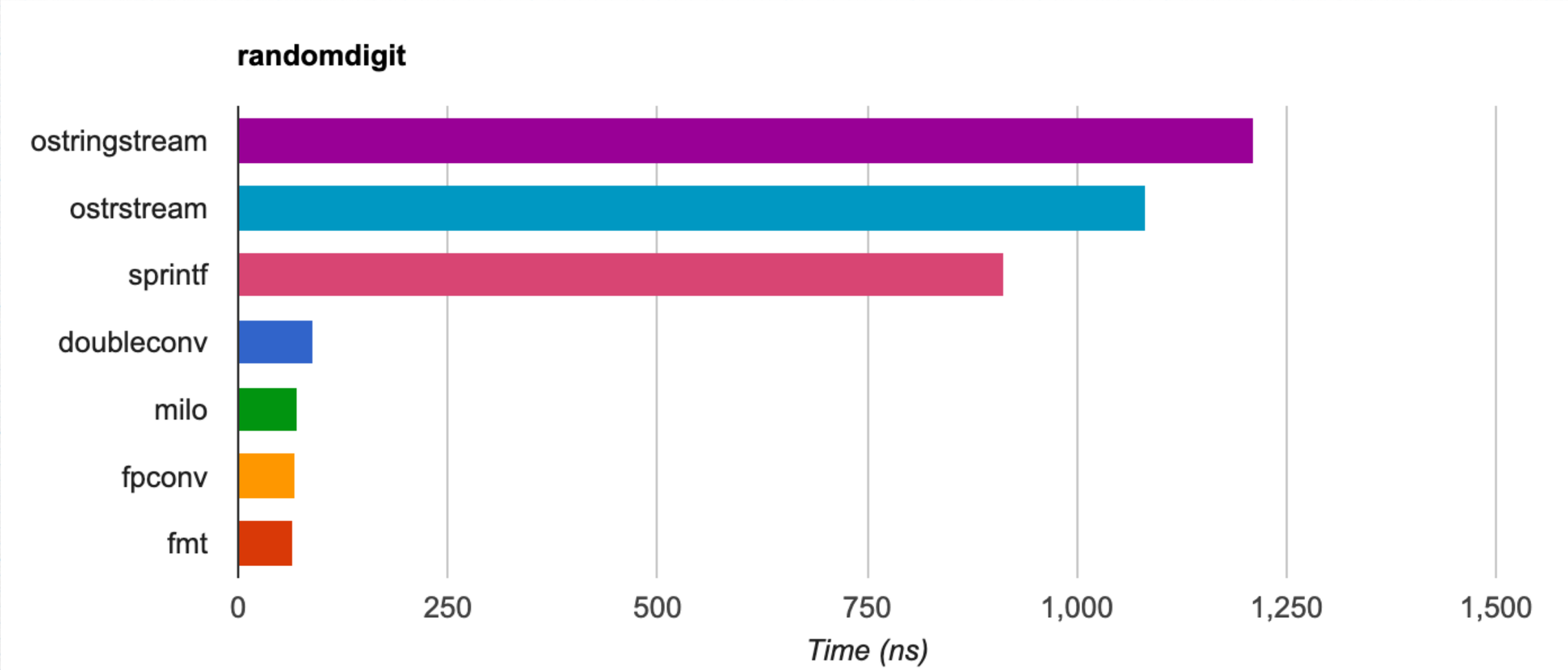
Method	int/s	Speed ratio
fmt::format_to[c]	110,567,284	1
std::to_chars	89,883,340	1.23
fmt::format[c]	83,470,376	1.325
fmt::format_to[r]	72,069,890	1.534
fmt::format[r]	68,167,216	1.622
std::to_string	65,638,239	1.684
boost::lexical_cast	34,091,974	3.243
sprintf	16,006,962	6.907
std::ostringstream	6,442,074	17.163
boost::format	2,986,724	37.02

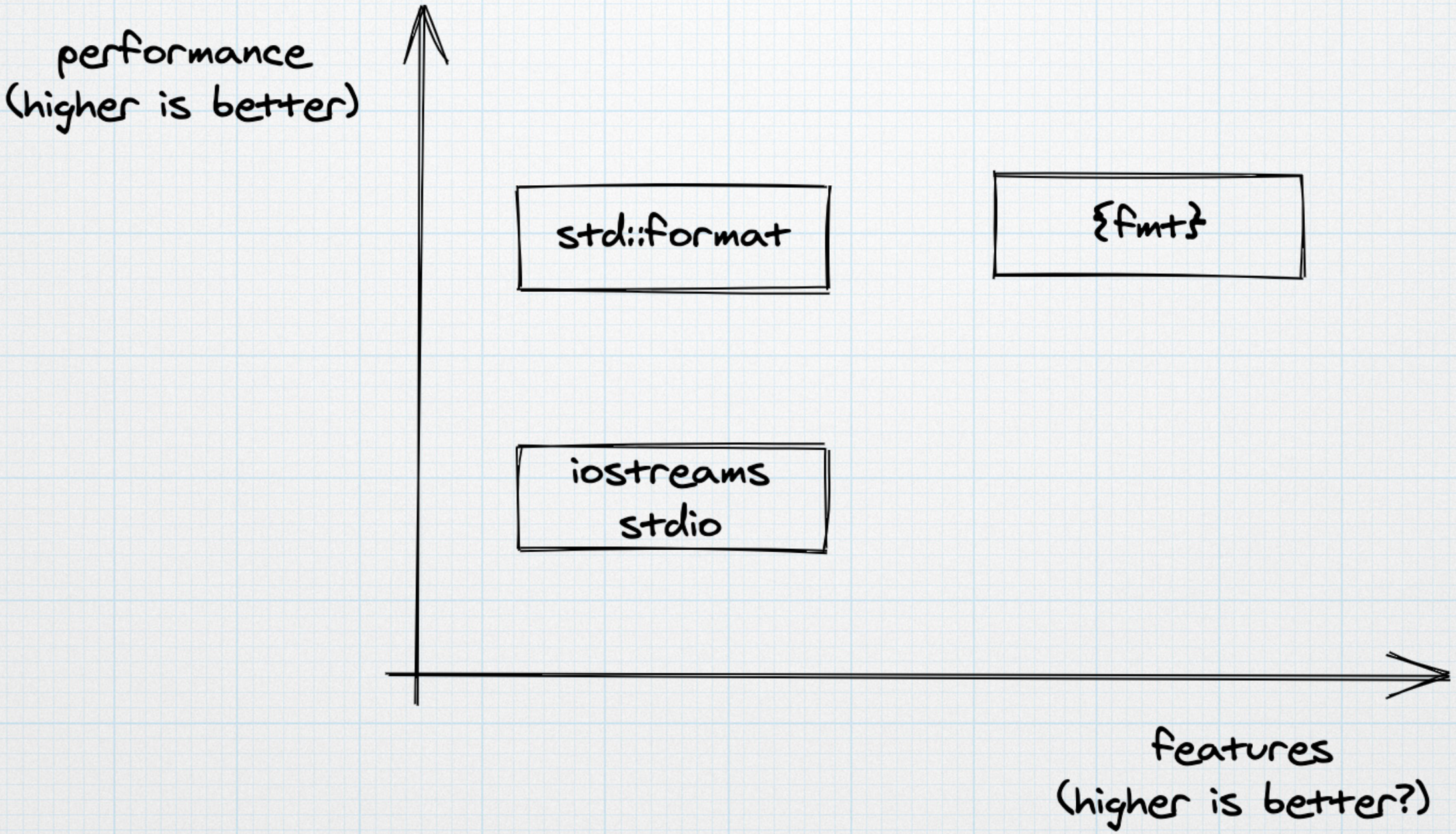
macOS/clang/libc++

Method	int/s	Speed ratio
fmt::format_to[c]	128,882,303	1
std::to_chars	111,938,297	1.151
fmt::format[c]	78,571,346	1.64
std::to_string	76,346,350	1.688
fmt::format_to[r]	69,235,556	1.862
fmt::format[r]	50,597,568	2.547
boost::lexical_cast	36,815,489	3.501
std::ostringstream	25,556,296	5.043
sprintf	20,702,142	6.226
boost::format	5,623,971	22.917

Linux/gcc/libstdc++

Floating point

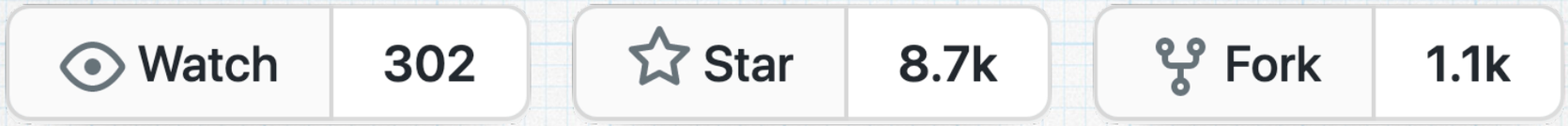



performance
(higher is better)
std::format
{fmt}
iostreams
stdio
features
(higher is better?)

Thanks

* The date library: https://github.com/HowardHinnant/date
* Calendar example is inspired by D's Component programming with ranges: https://wiki.dlang.org/Component_programming_with_ranges#Case_Study:_Formatting_a_Calendar
* Eric Niebler's Ranges for the Standard Library talk: https://www.youtube.com/watch?v=mFUXNMfaciE

The {fmt} library



* GitHub: https://github.com/fmtlib/fmt
* Docs: https://fmt.dev

* 200+ contributors
* Available in package managers of major Linux distributions, HomeBrew, conan, vcpkg, godbolt
* Used in many open-source and proprietary projects e.g. MongoDB, PyTorch, SeaStar, Windows Terminal, spdlog

"Suffering is optional."

- Haruki Murakami

Q&A