Showcase

Showcase

Why did the C++ template class refuse to go to the party?

Showcase

Why did the C++ template class refuse to go to the party?

Because it didn't want to be **instantiated** with the wrong crowd.

Why? / Use case

Why? / Use case

```
struct not_aligned {
  char c{};        // 1 byte(s)
  int i{};         // 4 byte(s)
  std::byte b{};   // 1 byte(s)
};
```

Why? / Use case

```cpp
struct not_aligned {
  char c{};       // 1 byte(s)
  int i{};        // 4 byte(s)
  std::byte b{};  // 1 byte(s)
};
```

> *"[intro.memory]/1: Object types have alignment requirements ([basic.align])... Objects of a given type shall be allocated at addresses that are multiples of its alignment."*

Why? / Use case

```cpp
struct not_aligned {
  char c{};        // 1 byte(s)
  int i{};         // 4 byte(s)
  std::byte b{};   // 1 byte(s)
};
```

> *"[intro.memory]/1: Object types have alignment requirements ([basic.align])... Objects of a given type shall be allocated at addresses that are multiples of its alignment."*

```cpp
static_assert(sizeof(not_aligned) == 12u);
```

Why? / Use case

Why? / Use case

```
struct not_aligned {
  char c{};       // 1 byte(s)
  int i{};        // 4 byte(s)
  std::byte b{}; // 1 byte(s)
};
```

Why? / Use case

```
struct not_aligned {
  char c{};      // 1 byte(s)
  int i{};       // 4 byte(s)
  std::byte b{}; // 1 byte(s)
};
```

```
static_assert(sizeof(align(not_aligned{})) == 8u);
```

Why? / Use case

```
struct not_aligned {
  char c{};      // 1 byte(s)
  int i{};       // 4 byte(s)
  std::byte b{}; // 1 byte(s)
};
```

```
auto align = [](auto&& t) {
  return to_tuple(t) | sort; // operator|(...) = magic;
};
```

```
static_assert(sizeof(align(not_aligned{})) == 8u);
```

to_tuple (almost standard compliant)

to_tuple (almost standard compliant)

```
struct foo { int i{}; bool b{}; };
```

to_tuple (almost standard compliant)

```
struct foo { int i{}; bool b{}; };
```

```
static_assert(std::tuple{42, true}
           == to_tuple(foo{.i = 42, .b = true}));
```

to_tuple (almost standard compliant)

```
template <class T>
[[nodiscard]] constexpr auto to_tuple(T&& obj) {
```

```
};
```

```
struct foo { int i{}; bool b{}; };
```

```
static_assert(std::tuple{42, true}
           == to_tuple(foo{.i = 42, .b = true}));
```

to_tuple (almost standard compliant)

```
template <class T>
[[nodiscard]] constexpr auto to_tuple(T&& obj) {
```

```
  if constexpr (requires { [&obj] { auto&& [p1, p2] = obj; }; }) {
    auto&& [p1, p2] = std::forward<T>(obj);
    return std::tuple{p1, p2};
  } else {
```

```
};
```

```
struct foo { int i{}; bool b{}; };
```

```
static_assert(std::tuple{42, true}
           == to_tuple(foo{.i = 42, .b = true}));
```

to_tuple (almost standard compliant)

```
template <class T>
[[nodiscard]] constexpr auto to_tuple(T&& obj) {
```

```
  if constexpr (requires { [&obj] { auto&& [p1, p2] = obj; }; }) {
    auto&& [p1, p2] = std::forward<T>(obj);
    return std::tuple{p1, p2};
  } else {
```

```
    ... // for 0..N
  }
```

```
};
```

```
struct foo { int i{}; bool b{}; };
```

```
static_assert(std::tuple{42, true}
           == to_tuple(foo{.i = 42, .b = true}));
```

Template ~~Meta-Programming~~ with std.ranges (C++20)

Template ~~Meta-Programming~~ with std.ranges (C++20)

Powered by https://github.com/boost-ext/mp

Motivation example - C++20 std::ranges (run-time)

Motivation example - C++20 std::ranges (run-time)

```
struct cmd {
  std::string_view name{}; // start, add, mod, del
  std::size_t id{};        // 1..N, 0=special
  constexpr auto operator<=>(const cmd&) const = default;
};
```

Motivation example - C++20 std::ranges (run-time)

Motivation example - C++20 std::ranges (run-time)

```
"std.ranges"_test = [] {
```

```
};
```

Motivation example - C++20 std::ranges (run-time)

```
"std.ranges"_test = [] {
```

```
    const auto v = std::vector{
        cmd{.name="start"},
        cmd{.name="add", .id=1},
        cmd{.name="add", .id=2},
        cmd{.name="add", .id=3},
        cmd{.name="mod", .id=2},
        cmd{.name="del", .id=1},
    };
```

```
};
```

Motivation example - C++20 std::ranges (run-time)

```
"std.ranges"_test = [] {
```

```
    const auto v = std::vector{
        cmd{.name="start"},
        cmd{.name="add", .id=1},
        cmd{.name="add", .id=2},
        cmd{.name="add", .id=3},
        cmd{.name="mod", .id=2},
        cmd{.name="del", .id=1},
    };
```

```
    expect(
      std::vector{
        cmd{.name="mod"},
        cmd{.name="add"},
      }
      ==
      execute(v, pred, op));
    );
```

```
};
```

Motivation example - C++20 std::ranges (run-time)

Motivation example - C++20 std::ranges (run-time)

```
#include <ranges>
```

Motivation example - C++20 std::ranges (run-time)

```
#include <ranges>
```

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1)         // remove start
   | std::views::reverse         // lifo
   | std::views::filter(pred)    // id == 2
   | std::views::transform(op)   // id = 0
   ;
};
```

Motivation example - C++20 std::ranges (run-time)

```
#include <ranges>
```

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1)         // remove start
   | std::views::reverse         // lifo
   | std::views::filter(pred)    // id == 2
   | std::views::transform(op)   // id = 0
   ;
};
```

```
auto pred  = [](const auto& c) { return c.id == 2;};
auto op    = [](const auto& c) { return cmd{.name=c.name}; };
```

Let's do it at compile-time...

Motivation example - boost.mp11 (compile-time)

Motivation example - boost.mp11 (compile-time)

```
struct cmd {
  std::string_view name{};
  std::size_t id{};
  constexpr auto operator<=>(const cmd&) const = default;
};
```

Motivation example - boost.mp11 (compile-time)

```
struct cmd {
  std::string_view name{};
  std::size_t id{};
  constexpr auto operator<=>(const cmd&) const = default;
};
```

```
template<fixed_string Name, std::size_t Id = 0u>
struct cmd {
  static constexpr auto name = Name;
  static constexpr auto id = Id;
};
```

Motivation example - boost.mp11 (compile-time)

Motivation example - boost.mp11 (compile-time)

```
"mp11"_test = [] {
```

```
};
```

Motivation example - boost.mp11 (compile-time)

```
"mp11"_test = [] {
```

```
    using v = boost::mp11::mp_list<
        cmd<"start">,
        cmd<"add", 1>,
        cmd<"add", 2>,
        cmd<"add", 3>,
        cmd<"mod", 2>,
        cmd<"del", 1>
    >;
```

```
};
```

Motivation example - boost.mp11 (compile-time)

```
"mp11"_test = [] {
```

```
    using v = boost::mp11::mp_list<
        cmd<"start">,
        cmd<"add", 1>,
        cmd<"add", 2>,
        cmd<"add", 3>,
        cmd<"mod", 2>,
        cmd<"del", 1>
    >;
```

```
    using expected = boost::mp11::mp_list<
        cmd<"mod">,
        cmd<"add">
    >;
```

```
};
```

Motivation example - boost.mp11 (compile-time)

```
"mp11"_test = [] {
```

```
    using v = boost::mp11::mp_list<
        cmd<"start">,
        cmd<"add", 1>,
        cmd<"add", 2>,
        cmd<"add", 3>,
        cmd<"mod", 2>,
        cmd<"del", 1>
    >;
```

```
    using expected = boost::mp11::mp_list<
        cmd<"mod">,
        cmd<"add">
    >;
```

```
    static_assert(
        typeid(expected) == typeid(execute_t<v>)
    );
```

```
};
```

C++20 typeid operator== is constexpr

---

typeid

C++20 typeid operator== is constexpr

```cpp
// C++17
static_assert(std::is_same_v<int, int>);
```

---

typeid

C++20 typeid operator== is constexpr

```
// C++17
static_assert(std::is_same_v<int, int>);
```

```
// C++20 - concepts
static_assert(std::same_as<int, int>);
```

---

typeid

C++20 typeid operator== is constexpr

```
// C++17
static_assert(std::is_same_v<int, int>);
```

```
// C++20 - concepts
static_assert(std::same_as<int, int>);
```

---

typeid

```
// types
static_assert(typeid(int) == typeid(int));
```

C++20 typeid operator== is constexpr

```cpp
// C++17
static_assert(std::is_same_v<int, int>);
```

```cpp
// C++20 - concepts
static_assert(std::same_as<int, int>);
```

---

typeid

```cpp
// types
static_assert(typeid(int) == typeid(int));
```

```cpp
// values
static_assert(typeid(42) == typeid(int));
```

Motivation example - boost.mp11 (compile-time)

Motivation example - boost.mp11 (compile-time)

```
template<class T> using execute_t =
  boost::mp11::mp_transform<
    op,
    boost::mp11::mp_filter_q<
      pred<2>,
      boost::mp11::mp_reverse<
        boost::mp11::mp_drop_c<T, 1>
      >
    >
  >;
```

Motivation example - boost.mp11 (compile-time)

```
template<auto Id> struct pred {
  template<class T>
  using fn = std::bool_constant<T::id == Id>;
};
```

```
template<class T> using execute_t =
  boost::mp11::mp_transform<
    op,
    boost::mp11::mp_filter_q<
      pred<2>,
      boost::mp11::mp_reverse<
        boost::mp11::mp_drop_c<T, 1>
      >
    >
  >;
```

Motivation example - boost.mp11 (compile-time)

```
template<auto Id> struct pred {
  template<class T>
  using fn = std::bool_constant<T::id == Id>;
};
```

```
template<class T> using op = cmd<T::name>;
```

```
template<class T> using execute_t =
  boost::mp11::mp_transform<
    op,
    boost::mp11::mp_filter_q<
      pred<2>,
      boost::mp11::mp_reverse<
        boost::mp11::mp_drop_c<T, 1>
      >
    >
  >;
```

run-time vs compile-time?

std::ranges - run-time

```
auto pred = [](const auto& c) { return c.id == 2;};
auto op   = [](const auto& c) { return cmd{.name=c.name}; };
```

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1)
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

boost.mp11: compile-time

```
template<auto Id> struct pred {
  template<class T> using fn = std::bool_constant<T::id == Id>;
};
```

```
template<class T> using op = cmd<T::name>;
```

```
template<class T> using execute_t =
  boost::mp11::mp_transform<
    op,
    boost::mp11::mp_filter_q<
      pred<2>,
      boost::mp11::mp_reverse<
        boost::mp11::mp_drop_c<T, 1>
      >
    >
  >;
```

Can we do better?

Motivation example - mp+std::ranges (compile-time)

Motivation example - mp+std::ranges (compile-time)

```
"mp+std.ranges"_test = [] {
```

```
};
```

Motivation example - mp+std::ranges (compile-time)

```
"mp+std.ranges"_test = [] {

    auto v = boost::mp::list<
        cmd<"start">,
        cmd<"add", 1>,
        cmd<"add", 2>,
        cmd<"add", 3>,
        cmd<"mod", 2>,
        cmd<"del", 1>
    >;

};
```

Motivation example - mp+std::ranges (compile-time)

```
"mp+std.ranges"_test = [] {
```

```
    auto v = boost::mp::list<
        cmd<"start">,
        cmd<"add", 1>,
        cmd<"add", 2>,
        cmd<"add", 3>,
        cmd<"mod", 2>,
        cmd<"del", 1>
    >;
```

```
    static_asssert(
      boost::mp::list<
        cmd<"mod">,
        cmd<"add">
      >
      ==
      execute(v, pred, op))
    );
```

```
};
```

Motivation example - mp+std::ranges (compile-time)

Motivation example - mp+std::ranges (compile-time)

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1_c) // constexpr parameter
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

Motivation example - mp+std::ranges (compile-time)

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1_c) // constexpr parameter
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

```
auto pred = []<class T> { return T::id == 2; };
auto op   = []<class T> { return cmd<T::name>{};; });
```

'constexpr' parameters

'constexpr' parameters

```
constexpr auto foo = [](auto v) {
  return std::array<int, v>{};
};
```

'constexpr' parameters

```
constexpr auto foo = [](auto v) {
  return std::array<int, v>{};
};
```

```
constexpr auto f = foo(42); // error: v is not constant expression
```

'constexpr' parameters

```
constexpr auto foo = [](auto v) {
  return std::array<int, v>{};
};
```

```
constexpr auto f = foo(42); // error: v is not constant expression
```

```
constexpr auto f = foo(std::integral_constant<int, 42>{}); // okay
                    // ^
                    // |
                    // 42_c
```

run-time vs compile-time?

std.ranges - run-time

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1)
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

```
auto pred  = [](const auto& c) { return c.id == 2;};
auto op    = [](const auto& c) { return cmd{.name=c.name}; };
```

mp+std.ranges - compile-time

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1_c)
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

```
auto pred  = []<class T> { return T::id == 2; };
auto op    = []<class T> { return cmd<T::name>{};; });
```

std.ranges - run-time (constexpr prameters)

```
auto execute = [](auto&& v, auto&& pred, auto&& op){
  return v
   | std::views::drop(1_c)
   | std::views::reverse
   | std::views::filter(pred)
   | std::views::transform(op)
   ;
};
```

```
auto pred  = [](const auto& c) { return c.id == 2;};
auto op    = [](const auto& c) { return cmd{.name=c.name}; };
```

How? (https://godbolt.org/z/j4W1WE1x4)

How? (https://godbolt.org/z/j4W1WE1x4)

```
struct foo { int f{}; };
struct bar { int b{}; };
struct baz { int b{}; };
```

How? (https://godbolt.org/z/j4W1WE1x4)

```
struct foo { int f{}; };
struct bar { int b{}; };
struct baz { int b{}; };
```

```
static_assert(
    std::tuple{foo{}, bar{}, baz{}}
      | std::views::filter(
          []<class T> { return requires(T t) { t.b; }; }
        )
    ==
    std::tuple{bar{}, baz{}}
);
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
template <template <class...> class T, class... Ts,
          template <class...> class TRng, class A, class Fn>
```

```
[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {
```

```
}
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
template <template <class...> class T, class... Ts,
          template <class...> class TRng, class A, class Fn>
```

```
  requires is_mp_range<T<Ts...>, TRng<A, T<Fn>>>
```

```
[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {
```

```
}
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
template <template <class...> class T, class... Ts,
          template <class...> class TRng, class A, class Fn>

  requires is_mp_range<T<Ts...>, TRng<A, T<Fn>>>

[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {

    constexpr auto fns = std::array{Fn{}.template operator()<Ts>()...};









}
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
template <template <class...> class T, class... Ts,
          template <class...> class TRng, class A, class Fn>
```

```
  requires is_mp_range<T<Ts...>, TRng<A, T<Fn>>>
```

```
[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {
```

```
    constexpr auto fns = std::array{Fn{}.template operator()<Ts>()...};
```

```
    constexpr auto indices = std::views::iota(0u, sizeof...(Ts))
        | A{}([&fns](auto i) { return fns[i]; })
        | std::ranges::to<static_vector<std::size_t, sizeof...(Ts)>>();
```

```
}
```

How? (https://godbolt.org/z/j4W1WE1x4) / Simplified

```
template <template <class...> class T, class... Ts,
          template <class...> class TRng, class A, class Fn>

  requires is_mp_range<T<Ts...>, TRng<A, T<Fn>>>

[[nodiscard]] constexpr auto operator|(T<Ts...>&& t, TRng<A, T<Fn>>&&) {

    constexpr auto fns = std::array{Fn{}.template operator()<Ts>()...};

    constexpr auto indices = std::views::iota(0u, sizeof...(Ts))
        | A{}([&fns](auto i) { return fns[i]; })
        | std::ranges::to<static_vector<std::size_t, sizeof...(Ts)>>();

    return [&]<auto... Is>(std::index_sequence<Is...>) {
        return T{std::get<indices[Is]>(t)...};
    }(std::make_index_sequence<indices.size()>());

}
```

Don't Repeat Yourself (DRY)

DRY

DRY

```
auto slice = [](auto&& v, auto&& begin, auto&& end) {
  return v
    | std::views::drop(begin)
    | std::views::take(end - 1_c);
};
```

DRY

```
auto slice = [](auto&& v, auto&& begin, auto&& end) {
  return v
    | std::views::drop(begin)
    | std::views::take(end - 1_c);
};
```

```
// type_list
static_assert(slice(mp::list<int, double, float, short>, 1_c, 3_c)
                  == mp::list<double, float>);
```

DRY

```
auto slice = [](auto&& v, auto&& begin, auto&& end) {
  return v
    | std::views::drop(begin)
    | std::views::take(end - 1_c);
};
```

```
// type_list
static_assert(slice(mp::list<int, double, float, short>, 1_c, 3_c)
                == mp::list<double, float>);
```

```
// variant
static_assert(slice(std::variant<int, double, float, short>{}, 1_c, 3_c)
                == std::variant<double, float>{});
```

DRY

DRY

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c)
                  == mp::list<2, 3>);
```

DRY

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c)
                  == mp::list<2, 3>);
```

```
// fixed_string - C++20
static_assert(slice(mp::list<"foobar">, 1_c, 3_c)
                  == mp::list<"oo">);
```

DRY

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c)
                  == mp::list<2, 3>);
```

```
// fixed_string - C++20
static_assert(slice(mp::list<"foobar">, 1_c, 3_c)
                  == mp::list<"oo">);
```

```
// tuple of values
static_assert(slice(std::tuple{1, 2, 3, 4}, 1_c, 3_c)
                  == std::tuple{2, 3});
```

DRY

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c)
                  == mp::list<2, 3>);
```

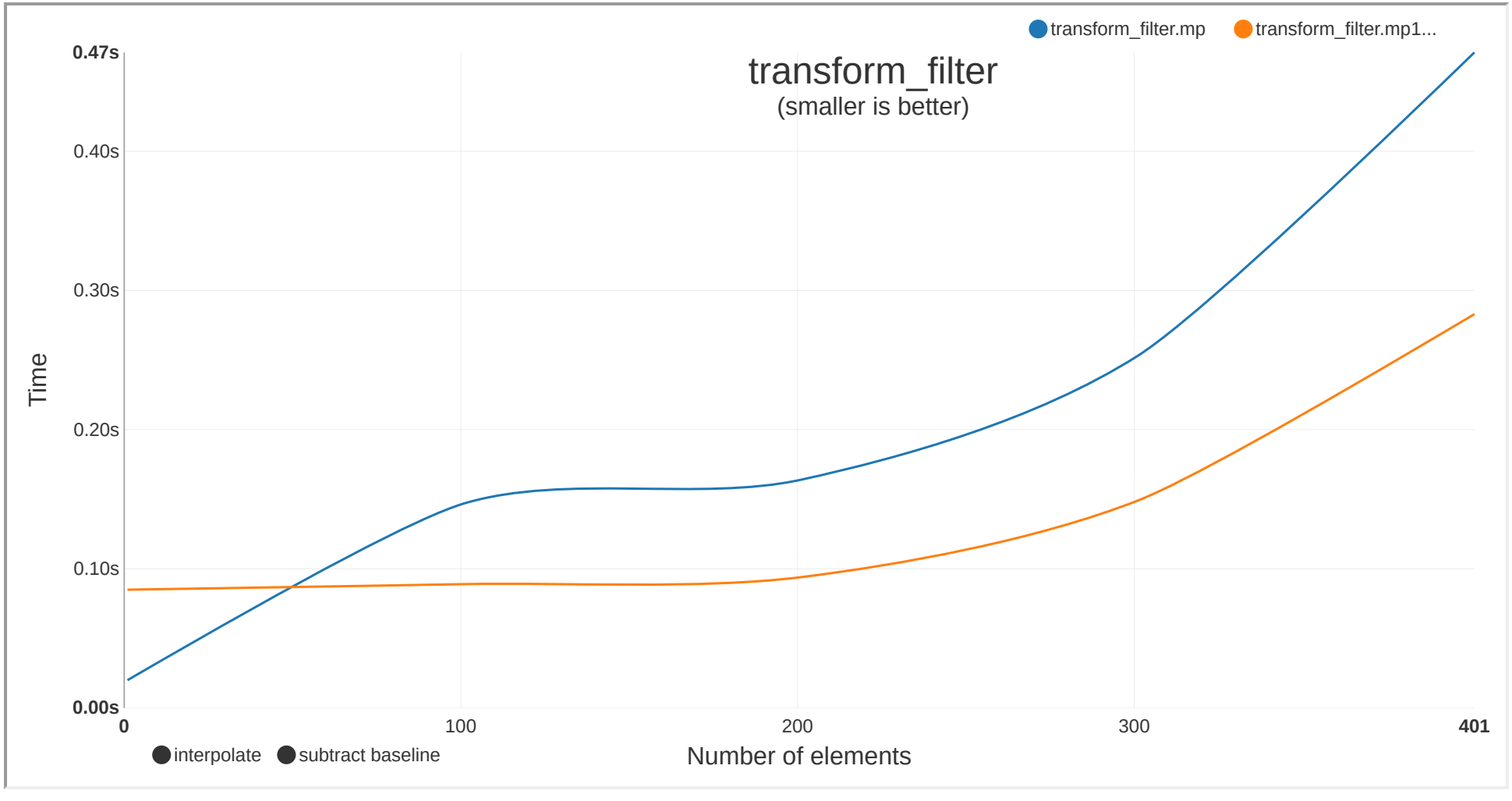
```
// fixed_string - C++20
static_assert(slice(mp::list<"foobar">, 1_c, 3_c)
                  == mp::list<"oo">);
```

```
// tuple of values
static_assert(slice(std::tuple{1, 2, 3, 4}, 1_c, 3_c)
                  == std::tuple{2, 3});
```

```
int main(int argc, const char**) {
  // run-time tuple of values
  assert((slice(std::tuple{1, argc, 3, 4}, 1_c, 3_c) ==
                 std::tuple{argc, 3}));
}
```

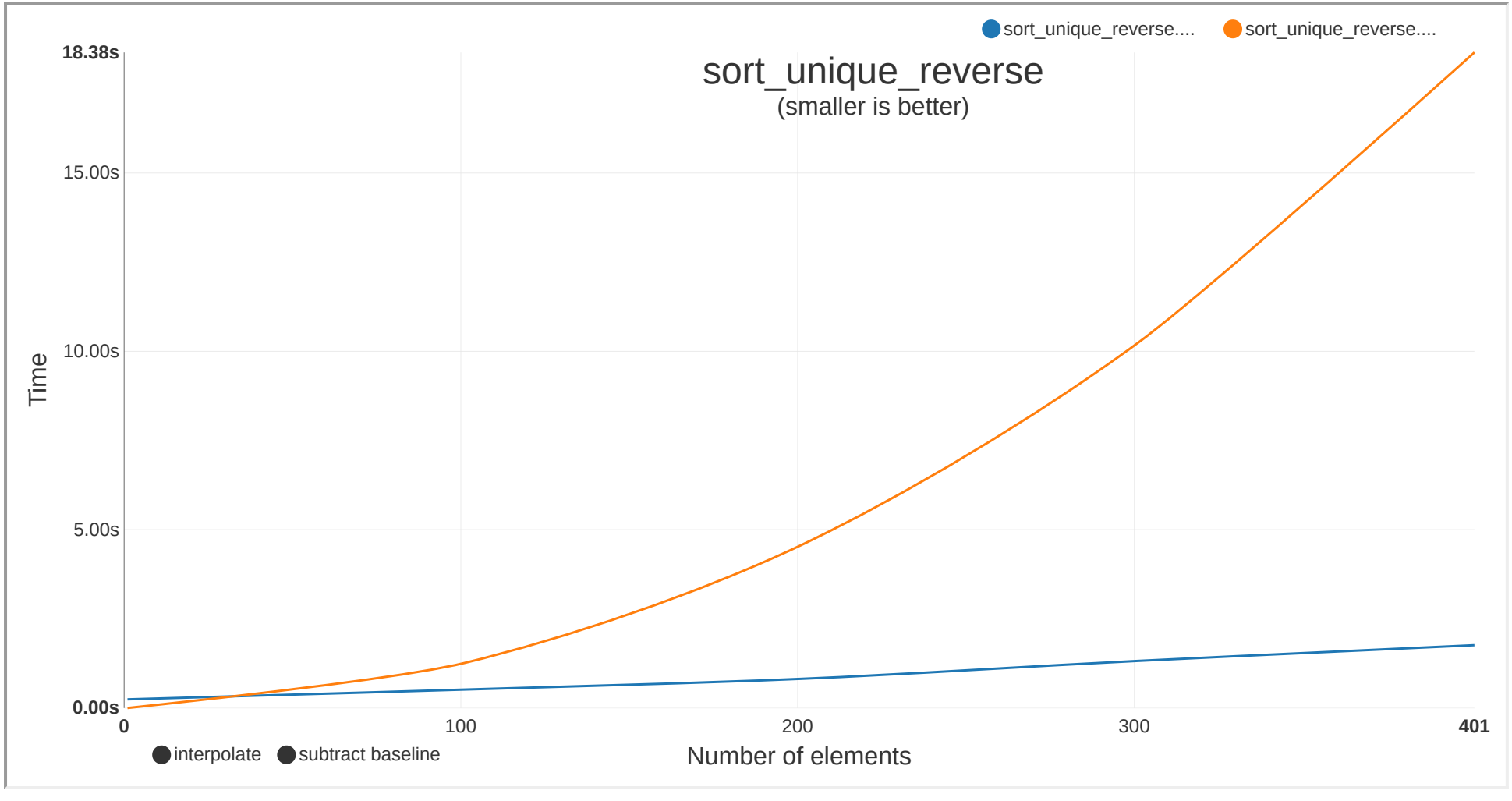
Compile-time benchmarks

[Benchmark - compilation time] - https://github.com/boost-ext/mp#benchmarks



POWERED BY HTTP://METABEN.CH

[Benchmark - compilation time] - https://github.com/boost-ext/mp#benchmarks



POWERED BY HTTP://METABEN.CH

Let's embrace C++20 ~~Template~~ Meta-Programming!