@odinthenerd

– not the god

@odinthenerd

brigand.list kvasir.list meta.list metal.list mp11.list mpl.list mpl.vector

Compilation time

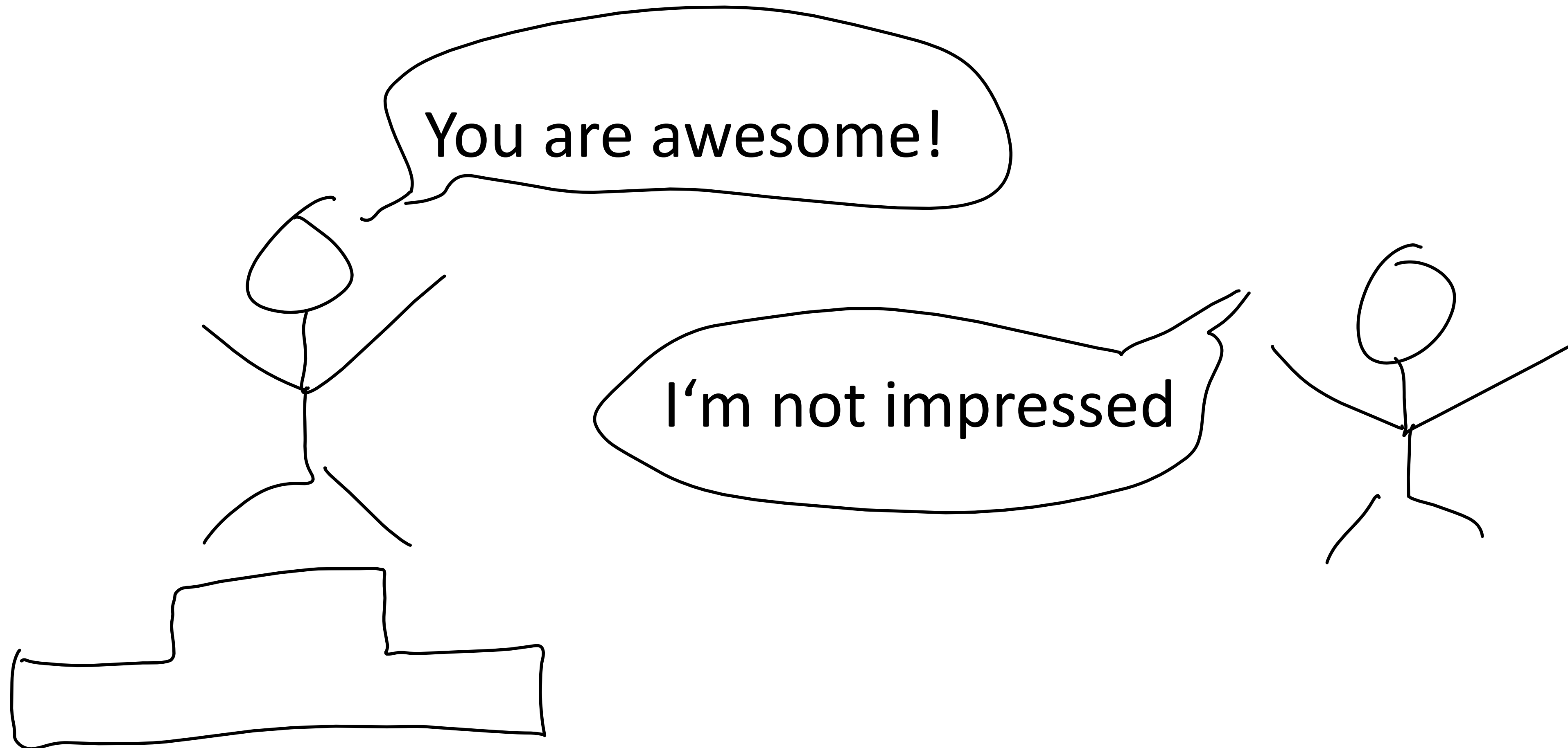
0.18s
0.17s
0.16s
0.15s
0.14s
0.13s
0.12s
0.11s
0.10s
0.09s
0.08s
0.07s
0.06s
0.05s
0.04s
0.03s
0.02s
0.01s
0.00s

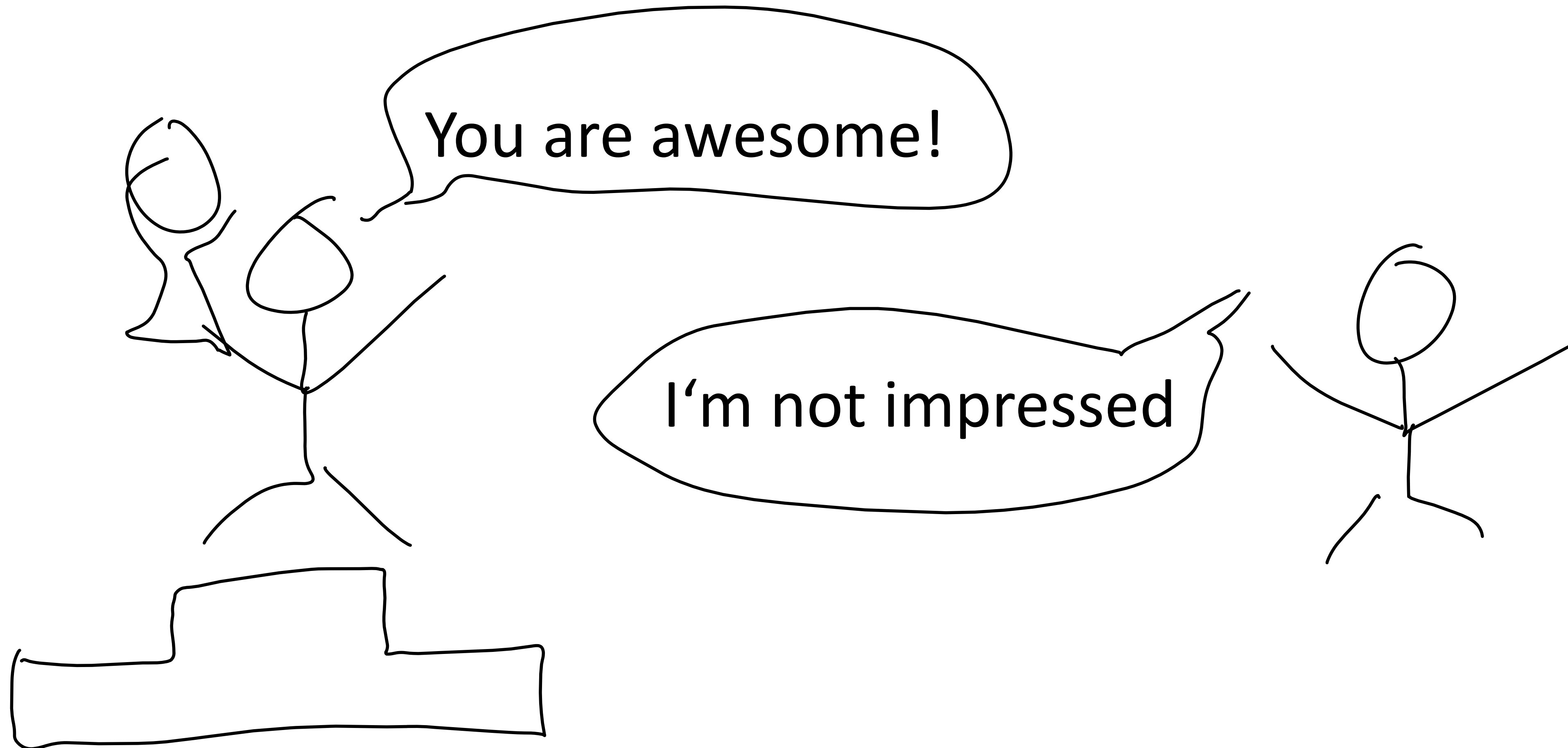
0 50 100 150 200 250 300 350 400 450 500

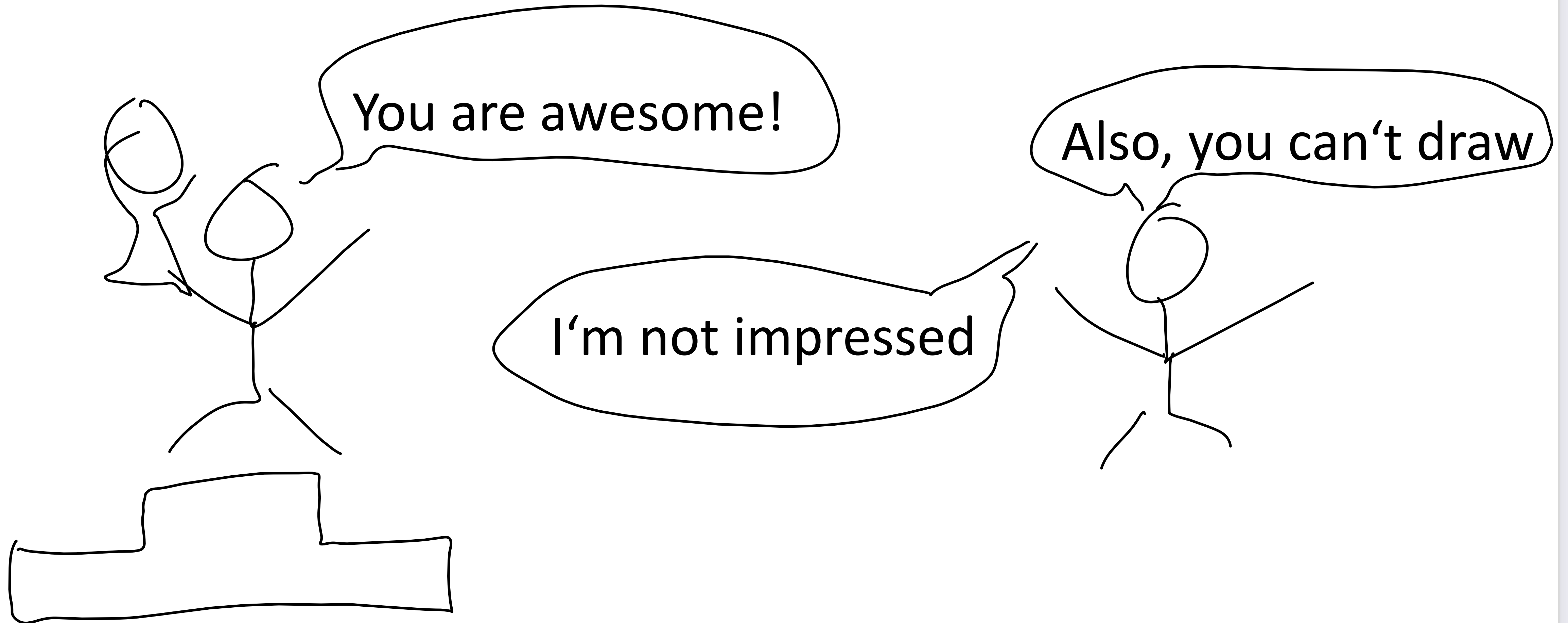
interpolate subtract baseline

Number of elements

You are awesome!
I‘m not impressed


You are awesome!
I‘m not impressed


You are awesome!
I‘m not impressed


You are awesome!
I‘m not impressed
Also, you can‘t draw

@odinthenerd

brigand.list
kvasir.list
meta.list
metal.list
mp11.list
mpl.list
mpl.vector

Compilation time

0.18s
0.17s
0.16s
0.15s
0.14s
0.13s
0.12s
0.11s
0.10s
0.09s
0.08s
0.07s
0.06s
0.05s
0.04s
0.03s
0.02s
0.01s
0.00s

0 50 100 150 200 250 300 350 400 450 500

Number of elements

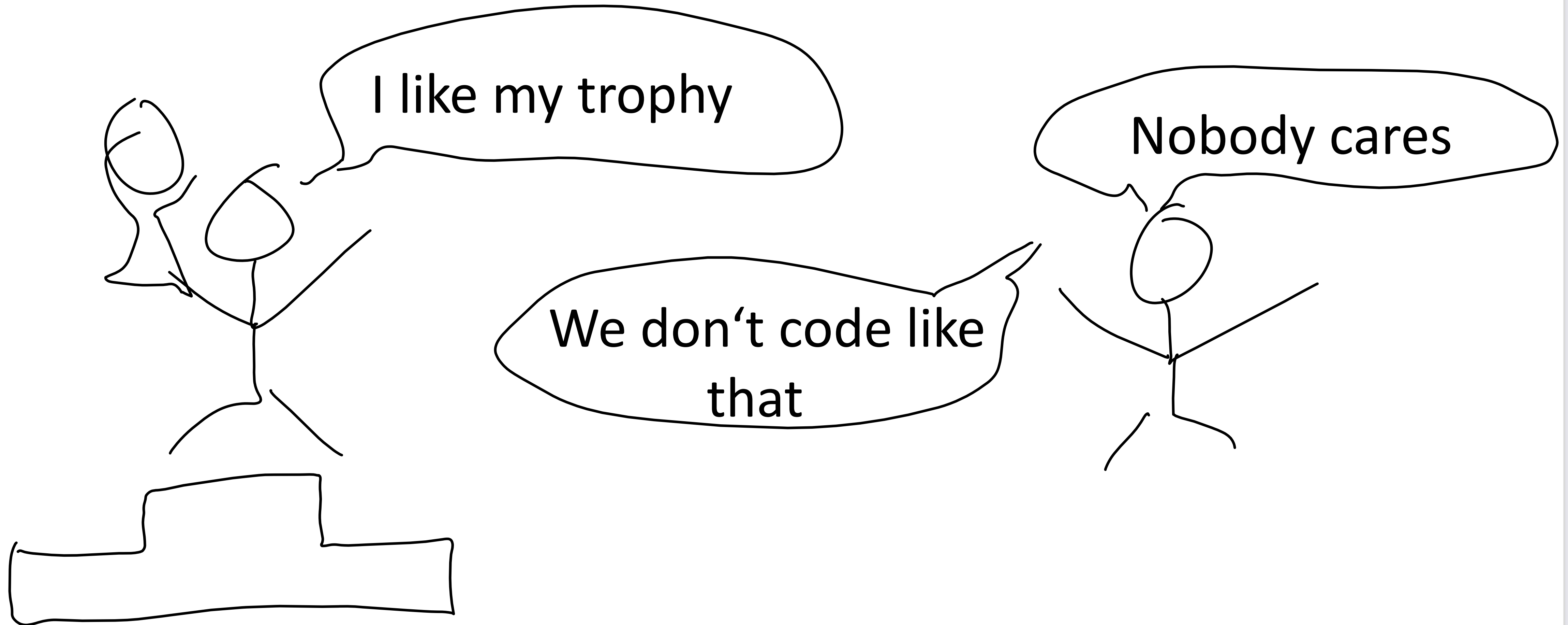
interpolate
subtract baseline

Cost of operations (Rule of Chiel)

- looking up a memoized type
- Adding a parameter to an alias call
- Adding a parameter to a type
- Calling an alias
- Instantiating a type
- Instantiating a function template
- SFINAE

Cost of operations (Rule of Chiel)

- looking up a memoized type
- Adding a parameter to an alias call
- Adding a parameter to a type
- Calling an alias
- Calling a variadic alias
- Instantiating a type
- Instantiating a function template
- SFINAE


I like my trophy
Nobody cares
We don‘t code like that

What is the most common metaprogramming feature?

std::enable_if

Substitution
Failure
Is
Not
An
Error

```
template<typename T>
typename std::enable_if<
    std::is_constructible<T,
        int,
        bool>::value,
    T>::type foo(T t, bool);
```

```
template<typename T>
typename std::enable_if<
    std::is_constructible<T,
        int,
        bool>::value,
    T>::type foo(T t, bool);
```

Do we understand SFINAE?

@odinthenerd

NO TIME TO EXPLAIN

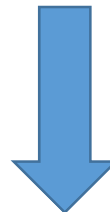
GET IN THE LLAMA

WeKnowMemes

We
Need
To
Understand
This
First

```
void foo(char, int);
void bar(){
    foo(1, false);
}
```

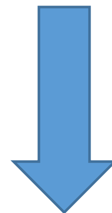
Name lookup
ADL
(Implicit object parameter + implied object argument)
(cv qualifications 😖)
Template argument deduction



Candidate function

```
namespace b{
    void foo(long, int);
    namespace c{
        void foo(char, int);
        void bar(){
            foo(1, false);
        }
    }
}
```

Name lookup
ADL
(Implicit object parameter + implied object argument)
(cv qualifications 😖)
Template argument deduction



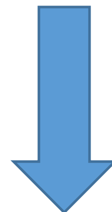
Candidate function

```
namespace a{
    struct s1{
        operator bool(){return false;}
    };

}

namespace b{
    void foo(long, int);
    namespace c{
        void foo(char, int);
        void bar(){
            foo(1, false);
        }
    }
}
```

```
namespace a{
    struct s1{
        operator bool(){return false;}
    };

}

namespace b{
    void foo(long, int);
    namespace c{
        void foo(char, int);
        void bar(){
            foo(a::s1{}, false);
        }
    }
}
```

```
namespace a{
    struct s1{
        operator bool(){return false;}
    };
    void foo(int,bool);
}

namespace b{
    void foo(long, int);
    namespace c{
        void foo(char, int);
        void bar(){
            foo(a::s1{}, false);
        }
    }
}
```

Name lookup
ADL
(Implicit object parameter + implied object argument)
(cv qualifications 😖)
Template argument deduction



Candidate function

```
template<typename T>
typename std::enable_if<
    std::is_constructible<T,
        int,
        bool>::value,
    T>::type foo(T t, bool);
```

Do we understand SFINAE?

```
template<typename T, typename U>
std::common_type_t<T,U> foo(T, U);
```

```
template<typename T, typename U>
some_meta_function<
     std::common_type_t,T,U
     > foo(T, U);
```

SFINAE Friendliness

Friendly:
boost.mp11
boost.mpl
meta
metal

Hard Error:
kvasir::mpl
brigand
boost.hana

```
template<typename T>
auto foo(T t)->decltype(bar(t)) {
    return bar(t);
}
```

```
template<typename T>
auto foo(T t)->decltype(bar(t)) {
    return bar(t);
}
```

```
template<typename T>
auto foo(T t){
    return bar(t);
}
```

```
// Value constructor absl::optional (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<(
        !std::is_same<in_place_t,
            typename std::decay<U>::type>::value &&
        !std::is_same<optional<T>,
            typename std::decay<U>::type>::value &&
        std::is_convertible<U &&, T>::value &&
        std::is_constructible<T, U &&>::value),
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

Do all of optionals constructors need to SFINAE?

```
int foo(int, optional<S>);

int foo(int, char);

decltype(foo(1,2)) bar;
```

How about now?

```
int foo(S, optional<S>);

int foo(int, char);

decltype(foo(1,2)) bar;
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
template <class _Up, bool>
struct __decay {
    typedef typename remove_cv<_Up>::type type;
};

template <class _Up>
struct __decay<_Up, true> {
public:
    typedef typename conditional<
        is_array<_Up>::value,
        typename remove_extent<_Up>::type*,
        typename conditional<
            is_function<_Up>::value,
            typename add_pointer<_Up>::type,
            typename remove_cv<_Up>::type
        >::type
    >::type type;
};

template <class _Tp>
struct decay{
    private:
    typedef typename remove_reference<_Tp>::type _Up;
    public:
    typedef typename __decay<_Up, __is_referenceable<_Up>::value>::type type;
};
```

```
template <class _Up, bool>
struct __decay {
    typedef typename remove_cv<_Up>::type type;
};

template <class _Up>
struct __decay<_Up, true> {
public:
    typedef typename conditional<
        is_array<_Up>::value,
        typename remove_extent<_Up>::type*,
        typename conditional<
            is_function<_Up>::value,
            typename add_pointer<_Up>::type,
            typename remove_cv<_Up>::type
        >::type
    >::type type;
};

template <class _Tp>
struct decay{
    private:
    typedef typename remove_reference<_Tp>::type _Up;
    public:
    typedef typename __decay<_Up, __is_referenceable<_Up>::value>::type type;
};
```

```
struct __is_referenceable_impl {
    template <class _Tp> static _Tp& __test(int);
    template <class _Tp> static __two __test(...);
};

template <class _Tp>
struct __is_referenceable : integral_constant<bool, !is_same<
    decltype(__is_referenceable_impl::__test<_Tp>(0)), __two>::value> {};
```

```
template <class _Up, bool>
struct __decay {
    typedef typename remove_cv<_Up>::type type;
};

template <class _Up>
struct __decay<_Up, true> {
public:
    typedef typename conditional<
        is_array<_Up>::value,
        typename remove_extent<_Up>::type*,
        typename conditional<
            is_function<_Up>::value,
            typename add_pointer<_Up>::type,
            typename remove_cv<_Up>::type
        >::type
    >::type type;
};

template <class _Tp>
struct decay{
    private:
    typedef typename remove_reference<_Tp>::type _Up;
    public:
    typedef typename __decay<_Up, __is_referenceable<_Up>::value>::type type;
};
```

```
std::negation<std::is_same<
    optional<T>,std::decay<U>::type>
>>
```

```
std::negation<std::is_same<
    optional<T>,
    std::remove_reference<std::remove_cv<U>>::type::type
>>
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (explicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        absl::negation<std::is_convertible<U &&, T>>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
explicit constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<(
        !std::is_same<in_place_t,
            typename std::decay<U>::type>::value &&
        !std::is_same<optional<T>,
            typename std::decay<U>::type>::value &&
        std::is_convertible<U &&, T>::value &&
        std::is_constructible<T, U &&>::value),
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
template<typename T>
struct is_a_thing : true_or_false {};
```

```
template<typename T>
struct is_a_thing : true_or_false {};


template<typename T>
using is_a_thing_t = typename is_a_thing<T>::type;
```

```
__is_same(T, U)
```

```
template<typename T, typename U>
using is_same_t = std::bool_constant<__is_same(T, U)>;
```

@odinthenerd

libcpp_is_same epo_is_same libcpp_is_same10 epo_is_same10 libcpp_is_same100 epo_is_same100

Compilation time

12.79s
12.00s
11.00s
10.00s
9.00s
8.00s
7.00s
6.00s
5.00s
4.00s
3.00s
2.00s
1.00s
0.00s

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Number of elements

350000

■	libcpp_is_same	**8.14s**
■	epo_is_same	**1.66s**

interpolate subtract baseline

@odinthenerd

libcpp_is_same epo_is_same libcpp_is_same10 epo_is_same10 libcpp_is_same100 epo_is_same100

Compilation time

12.79s
12.00s
11.00s
10.00s
9.00s
8.00s
7.00s
6.00s
5.00s
4.00s
3.00s
2.00s
1.00s
0.00s

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Number of elements

interpolate subtract baseline

@odinthenerd

libcpp_is_same epo_is_same libcpp_is_same10 epo_is_same10 libcpp_is_same100 epo_is_same100

Compilation time

12.79s
12.00s
11.00s
10.00s
9.00s
8.00s
7.00s
6.00s
5.00s
4.00s
3.00s
2.00s
1.00s
0.00s

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Number of elements

interpolate subtract baseline

500000	
libcpp_is_same	**12.79s**
epo_is_same	**2.84s**
libcpp_is_same10	**4.72s**
epo_is_same10	**2.61s**
libcpp_is_same100	**4.25s**
epo_is_same100	**1.53s**

```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
conjunction<
    is_same<T,int>,
    is_same<T,bool>,
    is_same<T,char>,
    is_same<T,long>>::value
```

@odinthenerd

libcpp_is_same · epo_is_same · libcpp_conjunction · epo_conjunction · if_conjunction

Compilation time: 0.00s, 10.00s, 20.00s, 30.00s, 40.00s, 50.00s, 60.00s, 69.10s

Number of elements: 0, 50000, 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000

350000	
libcpp_is_same	**8.14s**
libcpp_conjunction	**46.89s**

interpolate · subtract baseline

@odinthenerd

libcpp_is_same epo_is_same libcpp_conjunction epo_conjunction if_conjunction

Compilation time

69.10s
60.00s
50.00s
40.00s
30.00s
20.00s
10.00s
0.00s

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Number of elements

500000	
libcpp_is_same	12.79s
epo_is_same	2.71s
libcpp_conjunction	69.10s
epo_conjunction	45.02s

interpolate subtract baseline

@odinthenerd

libcpp_is_same epo_is_same libcpp_conjunction epo_conjunction if_conjunction

Compilation time

69.10s
60.00s
50.00s
40.00s
30.00s
20.00s
10.00s
0.00s

0 50000 100000 150000 200000 250000 300000 350000 400000 450000 500000

Number of elements

interpolate subtract baseline

500000	
libcpp_is_same	12.79s
epo_is_same	2.71s
libcpp_conjunction	69.10s
epo_conjunction	45.02s
if_conjunction	39.83s

boost.TMP stands for Tacit Meta Programming

boost.TMP stands for Tacit Meta Programming

BASH

sort | uniq -c | fold -rn

Ranges

```
std::vector<int> vi{1,2,3,4,5,6,7,8,9,10};
using namespace ranges;
auto rng = vi | view::remove_if([](int i){return i % 2 == 1;})
              | view::transform([](int i){return std::to_string(i);});
// rng == {"2","4","6","8","10"};
```

boost.TMP

```
int bar(int x, int y){
    return x+y;
}

template<typename...Ts>
int foo(Ts...args){
    return args | bar;
}
```

boost.TMP

```
int bar(int x, int y){
    return x+y;
}

template<typename...Ts>
int foo(Ts...args){
    return args | bar;
}

foo(1,2);
```

boost.TMP

```
int bar(int x, int y){
    return x+y;
}

template<typename...Ts>
int foo(Ts...args){
    return pack_(args) | bar;
}

foo(1,2);
```

boost.TMP

```
int bar(int x, int y){
    return x+y;
}

template<typename...Ts>
int foo(Ts...args){
    return pack_(args) | FWD_(bar);
}

foo(1,2);
```

boost.TMP

```
template<typename...Ts>
int foo(Ts...args){
    return pack_(args...) |
        filter<is_integral_>() |
        FWD_(bar);
}


foo("blah","blah","blah",1,baz{},2);
```

boost.TMP

```
template<typename...Ts>
int foo(Ts...args){
    return pack_(args...) |
        at_<uint_<3>>() |
        push_back<>(6) |
        FWD_(bar);
}

foo("blah","blah","blah",1,baz{},2);
```

boost.TMP

```
template<typename...Ts>
int foo(Ts...args){
    return pack_(args...) >>=
        at_<uint_<3>>() |
        push_back<>(6) |
        FWD_(bar);
}

foo("blah","blah","blah",1,baz{},2);
```

boost.TMP

```
using operation = at_<
    uint_<3>>;



using r = call_<
    operation,
    int,bool,char,long,float>;
```

boost.TMP

```
using operation = at_<
    uint_<3>,
    push_back_<long,
        is_same_<>>>;

using r = call_<
    operation,
    int,bool,char,long,float>;
```

```
call_<
    if_<
        is_<int>,
        if_<
            is_<bool>,
            if_<
                is_<char>,
                if_<
                    is_<long>,
                    always<true_>,
                >,
            >
        >
      ,or_else_<false_>>,
    U>
```

```
maybe_<
    if_<
        is_<int>,
        if_<
            is_<bool>,
            if_<
                is_<char>,
                if_<
                    is_<long>,
                    always<true_>,
                >,
            >
        >
    >,
    U>
```

`try_<common_type_t>`

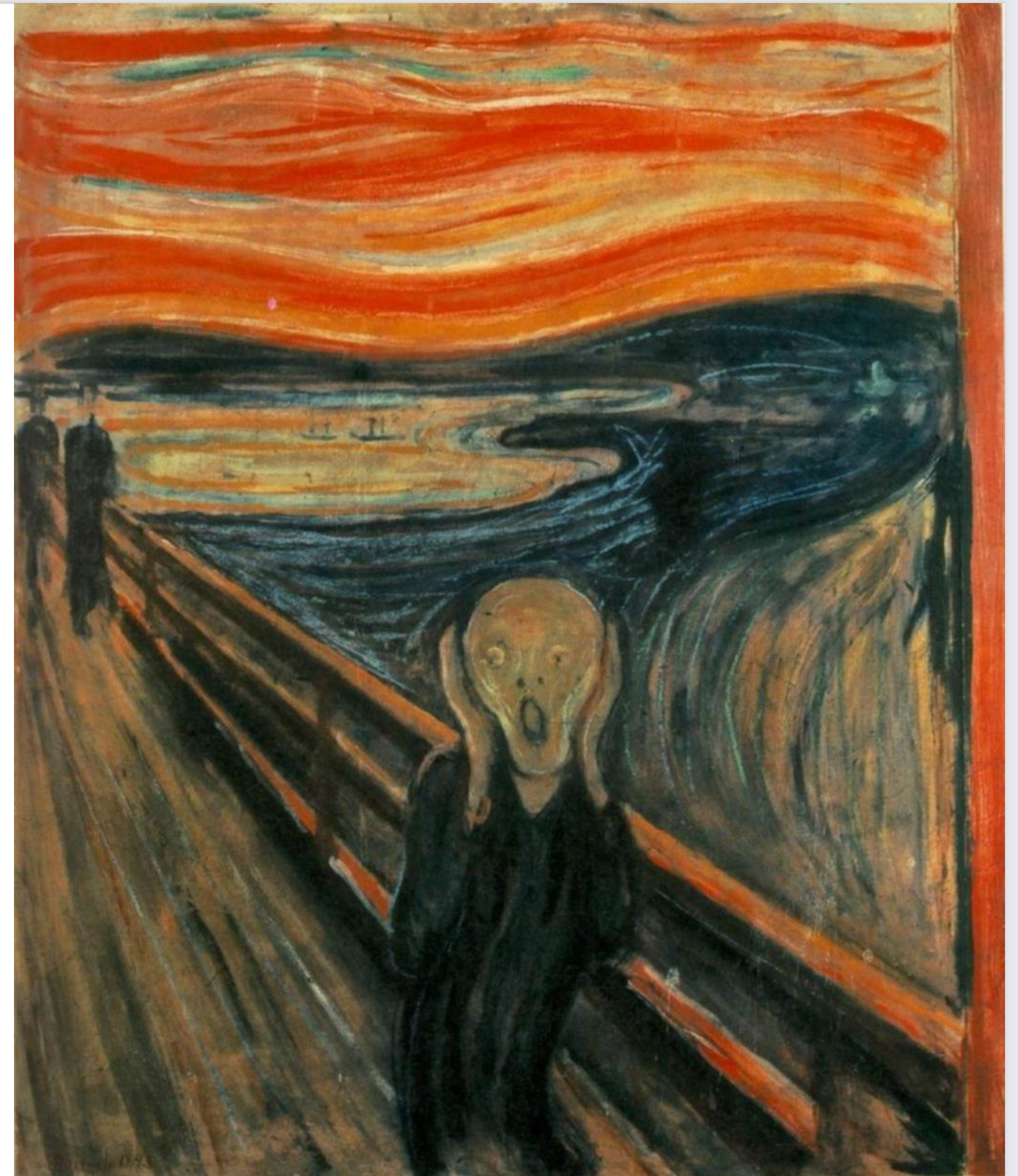
```
// Value constructor (implicit)
template<
    typename U = T,
    typename std::enable_if<
        absl::conjunction < absl::negation < std::is_same<
            in_place_t, typename std::decay<U>::type> >,
        absl::negation <std::is_same<
            optional<T>, typename std::decay<U>::type>>,
        std::is_convertible<U &&, T>,
        std::is_constructible<T, U &&>> ::value,
    bool>::type = false>
constexpr optional(U &&v) :
    data_base(in_place_t(), absl::forward<U>(v)) {}
```

```
template<
    typename U = T,
    typename Tag = maybe_<detail::ctr_tag_, U&&, T>
    >
constexpr optional(U &&v) :
    data_base(Tag(), absl::forward<U>(v)) {}
```

```
template<typename C = listify<>>
using to_sring = if_<
  is_<int_<0>>,
  always_<list_<int_<'0'>>>,
  push_back_<list_<>,
    while_<
      i0_<is_<int_<0>,not_<>>>,
      tee_<
        i0_<push_back_<int_<10>,divide_<>>>,
        each_<
          push_back_<int_<10>,modulo_<
            push_back_<int_<'0'>,plus_<listify_<>>>>>,
          identitiy_,
          join_<>>,
        continue_>
      i1_<unpack_<C>>>>;
```

```
using my_fizzbuzz = call_<
  make_sequence_<
    transform_<
      if_<
        push_back_<int_<3>,modulo_<is_<int_<0>>>>,
        if_<
          push_back_<int_<5>,modulo_<is_<int_<0>>>>,
          always_<fizzbuzz>,
          always_<fizz>>,
        if_<
          push_back_<int_<5>,modulo_<is_<int_<0>>>>,
          always_<fizz>,
          to_string<push_front_<int_<','>>>>>,
      join_<drop_<int_<1>>>>>,int_<1000>>;
```

SAMI DURANTE
LSU
Louisiana State
University
KYLA BRYANT
Stanford
University



@odinthenerd

- Github.com
- Twitter.com
- Gmail.com
- Blogspot.com
- LinkedIn.com
- Embo.io