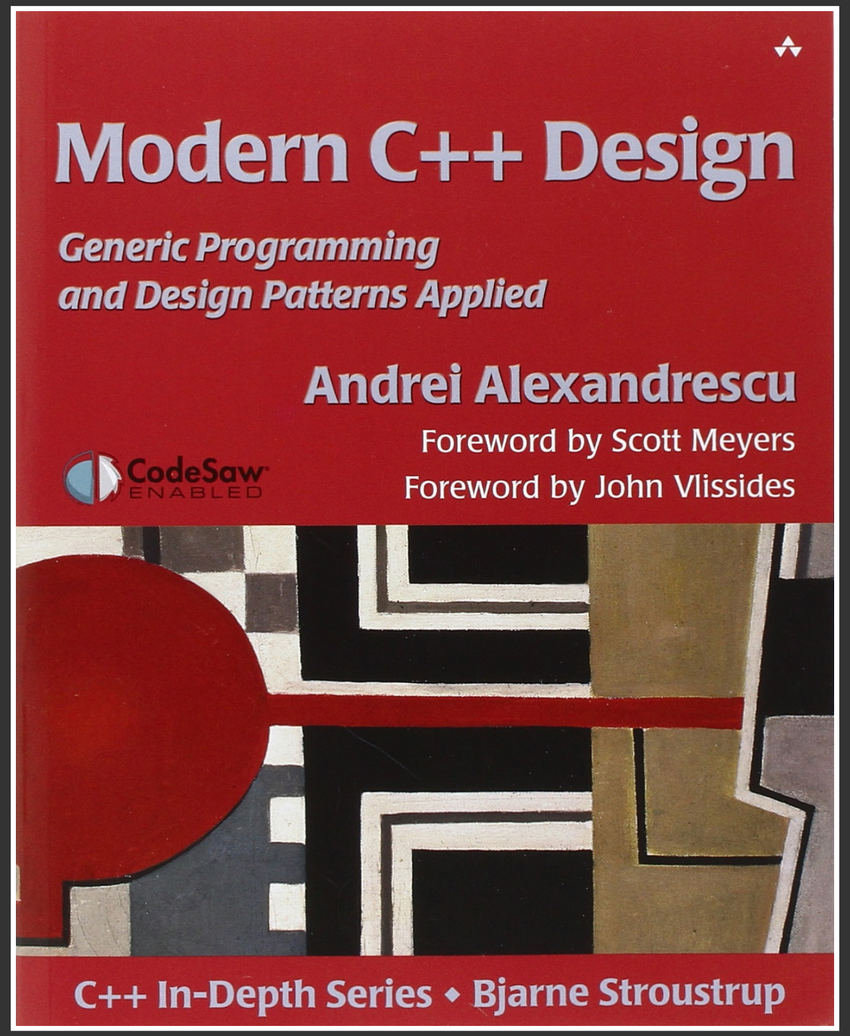

TYPE BASED METAPROGRAMMING
IS NOT
DEAD
BY ODIN HOLMES

A STORY OF RABBIT HOLES


Modern C++ Design
Generic Programming
and Design Patterns Applied
Andrei Alexandrescu
Foreword by Scott Meyers
Foreword by John Vlissides
CodeSaw
ENABLED
C++ In-Depth Series ◆ Bjarne Stroustrup

OTHER LIBS

- BOOST.HANA
- MP11
- METAL
- META
- BRIGAND

A DSL FOR TMP : META

```
namespace l = meta::lazy;
template <class L>
using cart_prod = reverse_fold<
    L, list<list<>>,
    lambda<
        _a, _b,
        l::join<l::transform< _b, lambda<_c,
            l::join<l::transform<_a,
                lambda<_d,
                    list<l::push_front<_d, _c>>>>>>>>>>;
```

A DSL FOR TMP : BRIGAND

```
template <class L>
using cart_prod = reverse_fold<
    L, list<list<>>,
    l::join<l::transform<
            _2, pin<l::join<
                l::transform<super<_1>, pin<
                    list<
                        l::push_front<_1, super<_1>>
                    >>>>>>>>>;
```

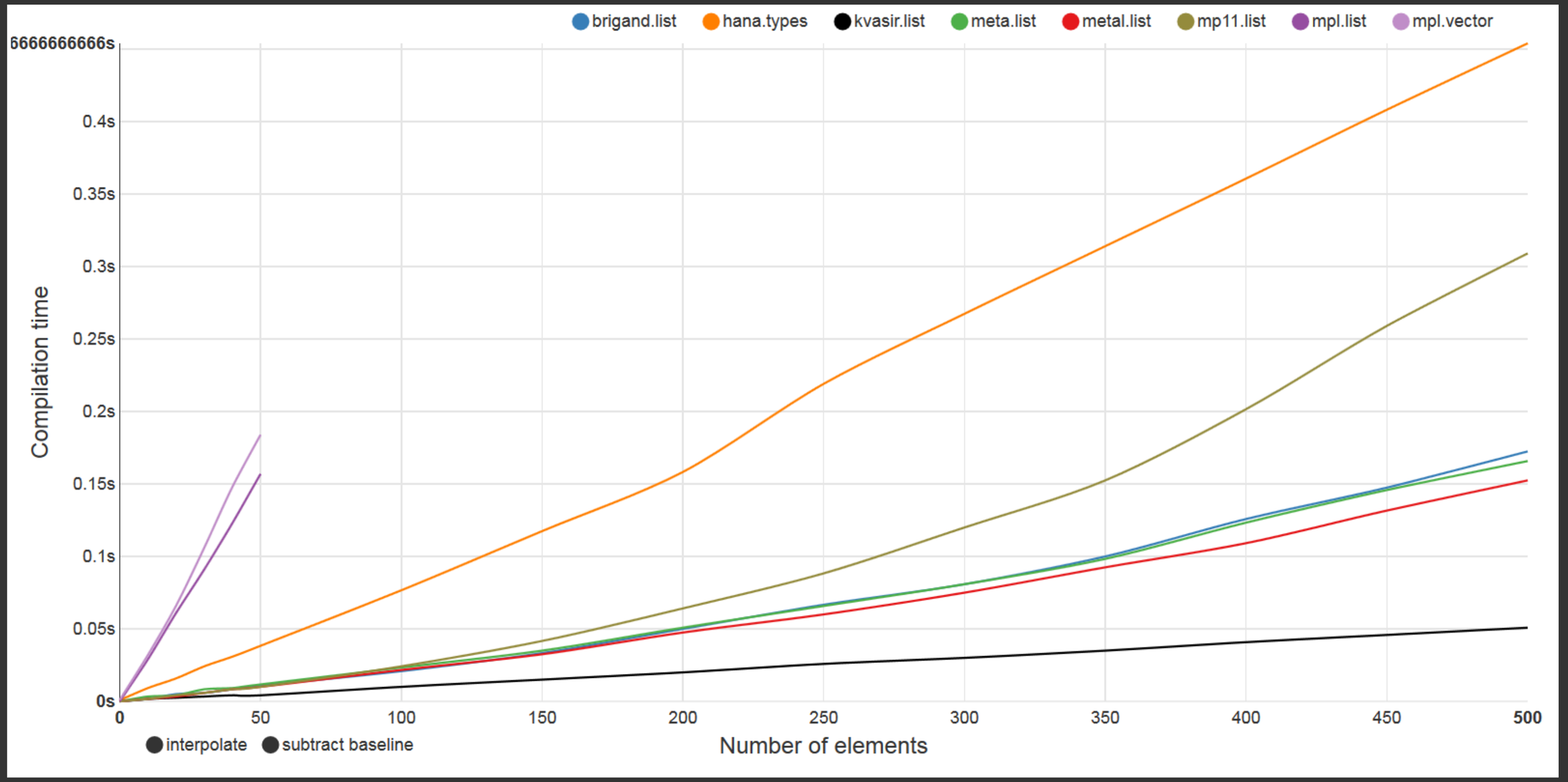
CHIEL DOUWES

COST OF OPERATIONS

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

AKA THE RULE OF CHIEL

REPLACE IF



CLASSIC CONDITIONAL

```
struct A{};
struct B{};
using result1 = typename std::conditional<true,A,B>::type;
using result2 = typename std::conditional<false,A,B>::type;
```

CLASSIC CONDITIONAL COST

```
struct A{};
struct B{};
using result1 = typename std::conditional<true,A,B>::type;
using result2 = typename std::conditional<false,A,B>::type;

using result3 = typename std::conditional<false,B,A>::type;
```

CLASSIC CONDITIONAL COST

```
struct A{};
struct B{};
using result1 = typename std::conditional<true,A,B>::type;
using result2 = typename std::conditional<false,A,B>::type;

using result3 = typename std::conditional<false,B,A>::type;

using result4 = typename std::conditional<false,A,B>::type;
```

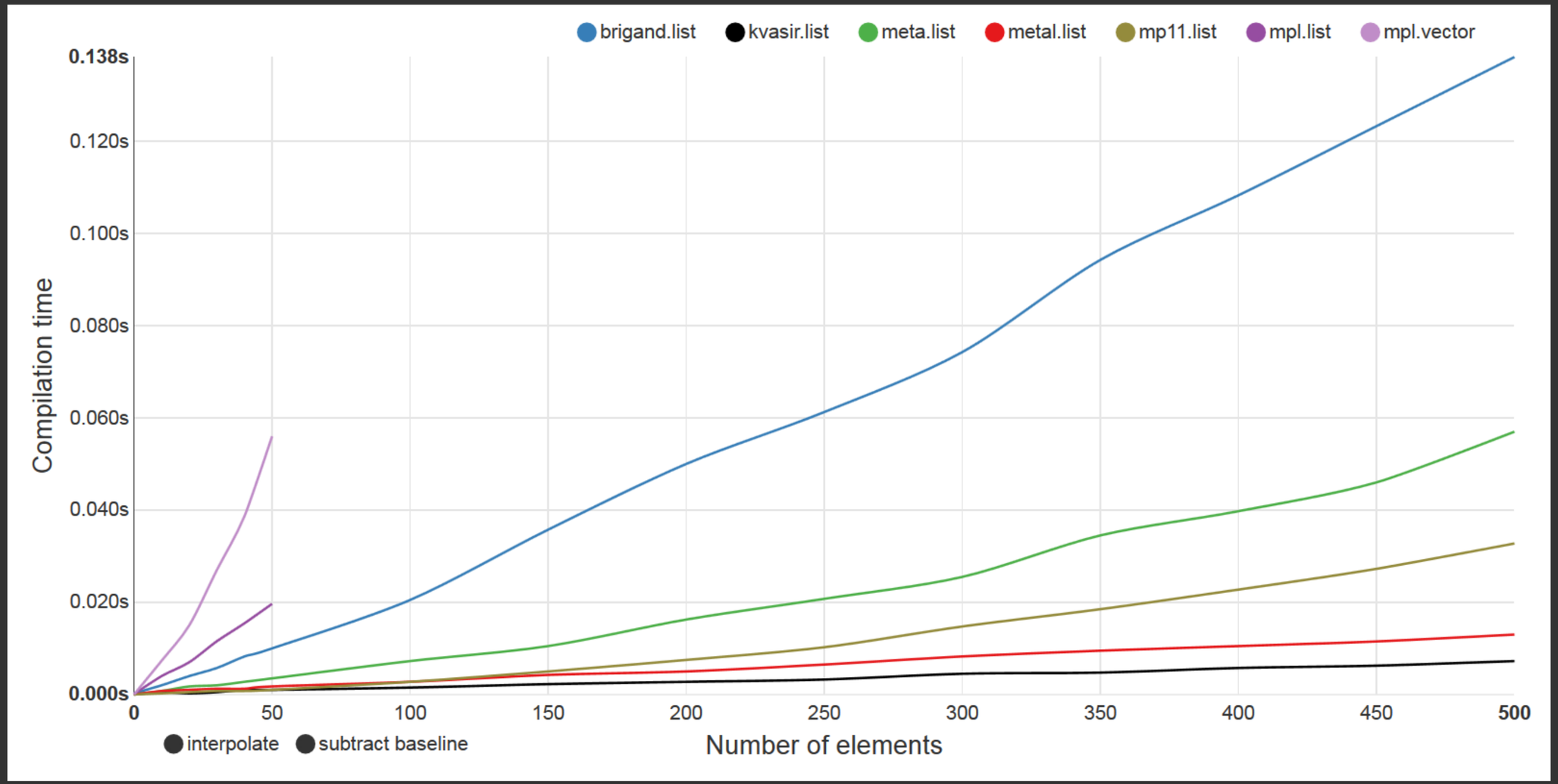
SELECTIVE ALIAS PATTERN

```
template<bool>
struct conditional{
        template<typename T, typename U>
        using f = T;
};

template<>
struct conditional<false>{
        template<typename T, typename U>
        using f = U;
};

struct A{};
struct B{};
using result1 = typename conditional<true>::template f<A,B>;
using result2 = typename conditional<false>::template f<A,B>;
using result3 = typename conditional<false>::template f<B,A>;
```

REPLACE IF

```
template<typename R, template<typename...> class F>
struct pred{
    template<typename T>
    using f = typename conditional<F<T>::value>::template f<R,T>;
};

template<typename R, template<typename...> class F, typename...Ts>
using replace_if = list<typename pred<R,F>::template f<Ts>...>;




template<typename T>
using is_void = std::is_same<T,void>;

using result = replace_if<int,is_void,int,void,char>;
```

FOLD LEFT



RECURSIVE ALIAS PATTERN

```
template<unsigned, template<typename...> class F>
struct fold_left_impl;
template<template<typename...> class F>
struct fold_left_impl<0,F>{
    template<typename T>
    using f = T;
};

template<template<typename...> class F>
struct fold_left_impl<1,F>{
    template<typename T, typename U, typename...Ts>
    using f = typename fold_left_impl<(sizeof...(Ts)>1?1:0),F>::
        template f<F<T,U>,Ts...>;
};


template<template<typename...> class F, typename T, typename...Ts>
using fold_left = typename fold_left_impl<(sizeof...(Ts)>0?1:0),F>::
    template f<T,Ts...>;
```

FASTTRACKING

```
template<unsigned, template<typename...> class F>
struct fold_left_impl;
template<template<typename...> class F>
struct fold_left_impl<0,F>{
    template<typename T>
    using f = T;
};

template<template<typename...> class F>
struct fold_left_impl<1,F>{
    template<typename T, typename U, typename...Ts>
    using f = typename fold_left_impl<(sizeof...(Ts)>0?1:0),F>::
        template f<F<T,U>,Ts...>;
};

template<template<typename...> class F>
struct fold_left_impl<2,F>{
    template<typename T0, typename T1, typename T2, typename T3, typename T4, typename T5,
        typename T6, typename T7, typename T8, typename T9, typename T10, typename...Ts>
    using f = typename fold_left_impl<(sizeof...(Ts)>0?sizeof...(Ts)>=10?2:1:0),F>::
        template f<F<F<F<F<F<F<F<F<F<T0,T1>,T2>,T3>,T4>,T5>,T6>,T7>,T8>,T9>,T10>,Ts...>;
};
```

COMPOSITION

```
template<template<typename...> class F, typename T, typename...Ts>
using fold_left = typename fold_left_impl<
    (sizeof...(Ts)>0?sizeof...(Ts)>=10?2:1:0),F>::template f<T,Ts...>;

template<typename R, template<typename...> class F, typename...Ts>
using replace_if = list<typename replace_if_pred<R,F>::template f<Ts>...>;




using result = replace_if<int_<0>,is_void,int_<1>, void, int_<7>, int_<5>>;
using sum = fold_left<add,int_<1>,int_<3>, int_<7>, int_<5>>;
```

META CLOSURES

```
template <typename F, typename R, typename C = listify>
struct replace_if {
    template <typename... Ts>
    using f = typename C::template
        f<typename replace_if_pred<F,R>::template f<Ts>...>;
};
```

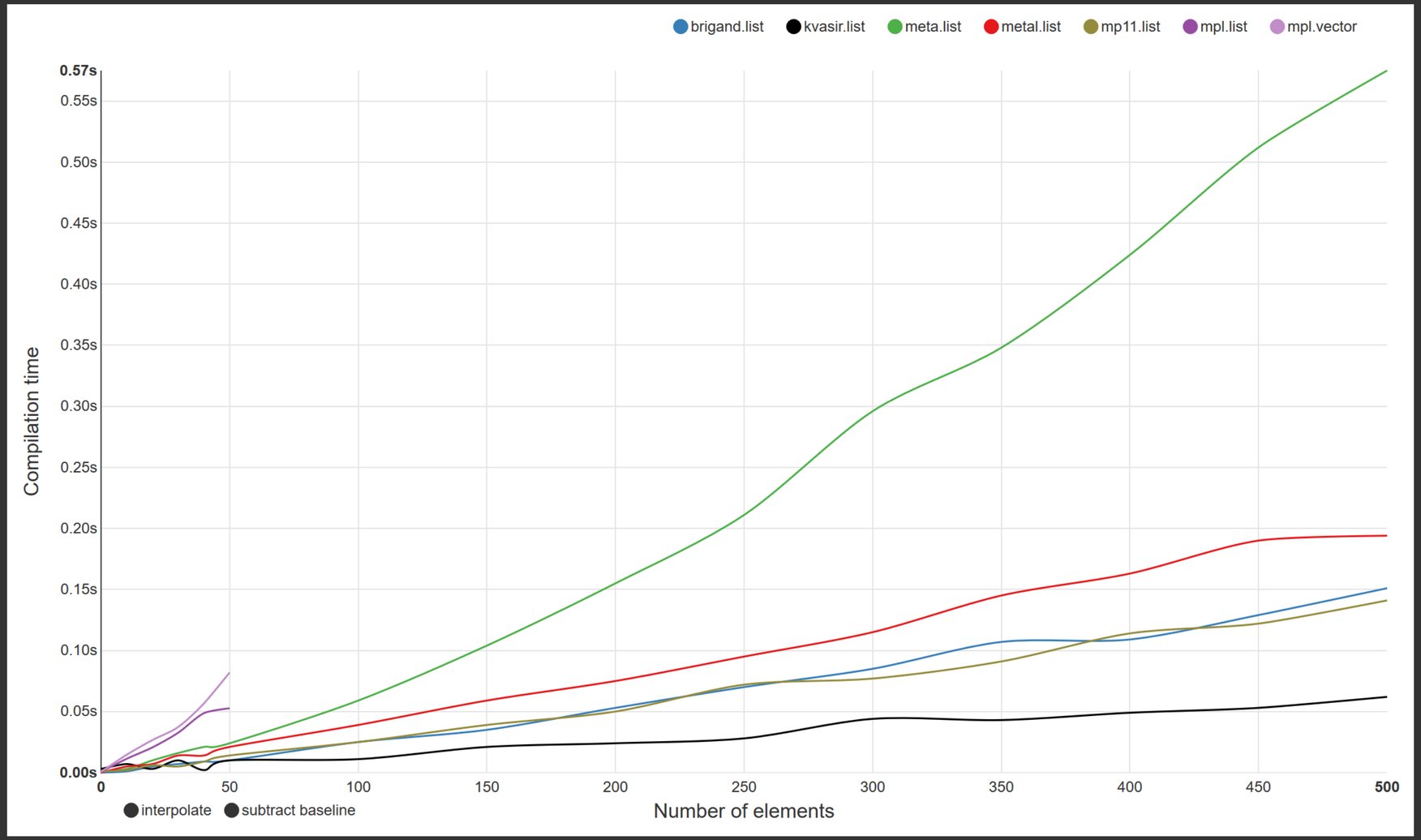
ZERO COST COMPOSITION

```
template <typename F, typename C = listify>
struct transform {
    template <typename... Ts>
    using f = typename C::template f<typename F::template f<Ts>...>;
};

template <typename Input, typename F = identity, typename C = listify>
using replace_if = transform<detail::replace_if_pred<F, Input>, C>;
```

(LESS THAN) ZERO COST COMPOSITION

```
template <typename F = identity, typename C = listify>
using partition = fork<list<remove_if<F>, filter<F>>,C>;
```

PARTITION



HIGHER ORDER METAFUNCTIONS

```
namespace mpl = boost::mpl;
typedef mpl::vector_c<int, 1, 1, 1> vec1;
typedef mpl::vector_c<int, 2, 2, 2> vec2;
typedef mpl::vector_c<int, 3, 3, 3> vec3;
typedef mpl::vector<vec1, vec2, vec3> vvec;

typedef typename mpl::lambda<
        mpl::fold< _1, mpl::int_<0>,
            typename mpl::lambda<
                mpl::plus< _1, _2>
            >::type
        > >::type lam;

typedef typename mpl::fold< vvec, mpl::int_<0>,
        mpl::plus< _1, mpl::protect< lam >::type::apply<_2>
        > >::type result;

static_assert(mpl::equal_to<result, mpl::int_<18>>::value,
    "should be 18");
```

CONTINUATIONS AS HIGHER ORDER METAFUNCTIONS

```
using namespace kvasir::mpl;
using l1 = list<int_<1>, int_<1>, int_<1>>;
using l2 = list<int_<2>, int_<2>, int_<2>>;
using l3 = list<int_<3>, int_<3>, int_<3>>;

using result = call<
    fold_left<
        each<
            list<
                identity,  //pretty much the same as mpl::_
                unpack<    //no need for mpl::_, just works
                    fold_left< plus<> >
                >
            >,
            plus<>
        >
    >, int_<0>, l1, l2, l3>;

static_assert(result::value == 18, "should be 18");
```

KVASIR

AUTO INTERN


TAKE-AWAY

FIND ME ONLINE

@ODINTHENERD

github.com/odinthenerd
twitter.com/odinthenerd
odinthenerd.blogspot.com