The problem with “cutting edge C++”

Jens Weller

Meeting C++

C++ Now 2018

Introduction

*** '81**

C++ since '98

was a C++ Freelancer

Meeting C++

C++ Evangelist

- Supporting C++ Community
- Social Media
- global Network for C++

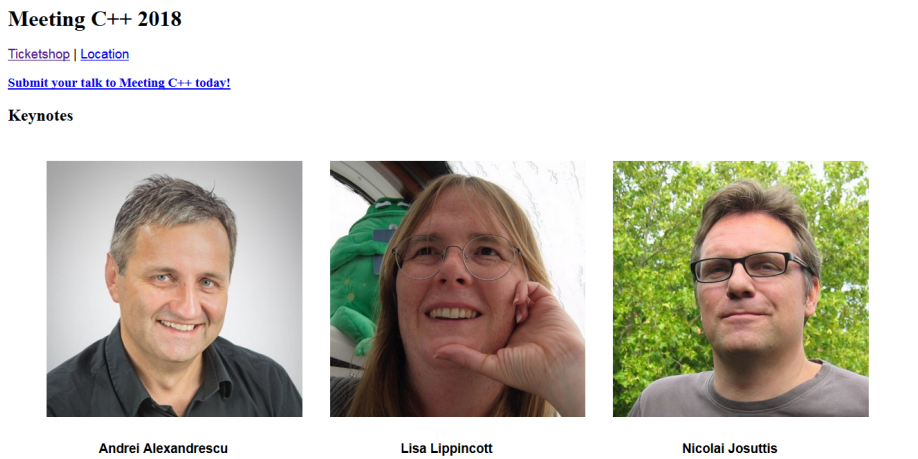
Meeting C++

Meeting C++ 2018

- Submit your talks
- Tickets available

New Features at Meeting C++

- Jobsection
  - Upload your own jobs
  - Employer listing
  - CV Submission form
- Soon
  - Contact C++ Trainers
  - more coming!
- Patreon
  - https://www.patreon.com/meetingcpp



A short example – boost::factory

MinGW

boost factory code works fine

I have a working and compiling application

MSVC

boost factory suddenly stops working

Error: boost\bind\bind.hpp:249: error: C2664: 'Panel *boost::factory::operator () (void) const': cannot convert argument 1 from 'Widget *' to 'Widget *&'

So this boils down to move semantics...

boost::bind<Widget*>(boost::factory<P*>(boost::factory**<P*>())**,_1,_2));

boost::bind<Widget*>(boost::**forward_adapter<**boost::**factory<P*>>(**boost::**factory<P*>())**,_1,_2));

Fix: type_factory<Widget,Panel>{}

boost, boost::move & move semantics

Is boost today 100% C++11 move aware?
Lets discuss this on another day...

“cutting edge C++”

“cutting edge C++”

When today's Hack becomes tomorrows Headache…

When Tools become old and rusted, and can't be trusted…

“cutting edge C++”

Its part of the culture in boost
… and C++Now

Its good and has brought us forward.
But lets also reflect on the hacks of the past…
… and what we'd like to have today.

We don't do “cutting edge C++”

Lets look at parts of the C++ world which don't do it first…

Frameworks which are older then boost usually still have an imprint of that time. OO heavy code with lots of pointers.

Their communities for a long time were quite conservative towards generic C++ and STL usage in general.

Not being “cutting edge C++”

Programming Style Guide

by Vadim Zeitlin

Warning: Parts of this document (especially those dealing with not using some C++ language features) are out of date for the latest wxWidgets development branch. They still apply to 2.8 version but this document needs to be revised for the upcoming 3.0 release.

General C++ Rules

- New or not widely supported C++ features
  1. Don't use C++ templates
  2. Don't use C++ exceptions
  3. Don't use RTTI
  4. Don't use namespaces
  5. Don't use STL
  6. Don't declare variables inside `for()`
  7. Don't use nested classes
  8. Don't use new logical operators keywords

Lets lo do it first…

Framew still have an imprin ode with lots of pointers.

Their communi uite conservative to TL usage in general.

Qt: We don't do “cutting edge C++”

```
int methodIndex = pushButton->metaObject()->indexOfMethod("animateClick()");
QMetaMethod method = metaObject->method(methodIndex);
method.invoke(pushButton, Qt::QueuedConnection);
```

Qt focus is on...

- … using mature C++ features
- Strong focus on OOP
- Signal & Slot for Events
- Templates are just an **obscure** C++ feature.

Qt Innovation: moc

- Reflection based on a QMetaObject
  - Macro based registration
    - QMetaMethod (Qt4)
    - QMetaEnum (Qt 2)
    - QMetaProperty (Qt2)
    - QMetaType (Qt4)
- QObject as base class
  - Q_OBJECT registration in your class
  - Can't be a template
  - Moc creates a helper class with Meta information

Qt: We don't do “cutting edge C++”

Qt focus is on...

- … using mature C++ features
- Strong focus on OOP
- Signal & Slot for Events
- Templates are just an **obscure** C++ feature.

```
int methodIndex = pushButton->metaObject()->indexOfMethod("animateClick()");
QMetaMethod method = metaObject->method(methodIndex);
method.invoke(pushButton, Qt::QueuedConnection);
```

```
bool QMetaMethod::invoke(QObject *object, Qt::ConnectionType connectionType,
QGenericReturnArgument returnValue, QGenericArgument val0 = QGenericArgument( Q_NULLPTR
), QGenericArgument val1 = QGenericArgument(), QGenericArgument val2 = QGenericArgument(),
QGenericArgument val3 = QGenericArgument(), QGenericArgument val4 = QGenericArgument(),
QGenericArgument val5 = QGenericArgument(), QGenericArgument val6 = QGenericArgument(),
QGenericArgument val7 = QGenericArgument(), QGenericArgument val8 = QGenericArgument(),
QGenericArgument val9 = QGenericArgument()) const
```

- Q_OBJECT registration in your class
- Can't be a template
- Moc creates a helper class with Meta information

Boost history

Boost

is a poster child of the C++ committee

Using the full power of the C++ language

Embraces the generic features of C++

Incubator for the C++ Standard

Try & Error

Boost Graph

Unique in its style and approach

2000 – 2001, boost 1.18

Boost MPL

Foundation of boost TMP

2002 – 2004, boost 1.30

Boost is meant to be a service to the C++ Community

Embracing Modern C++

Being cutting edge

Boost 1.9.0

first Boost release *visible in today's documentation*

Config, Integer, Operator, Timer, Value Initialized

(1999 - 2002)

Boost in 2018…

And from github we can observe the creation of issues like these ones:

Boost → C++11 migration for the bitcoin project.

Using C++11 facilities to clean up code and Boost dependencies

Deprecating boost and implementing C++11 support for yaml-cpp project

And here's from the yaml-cpp project repository the motivation behind this decision:

```
Requiring the adoption of the boost libraries so as to use yaml-cpp is problematic for many developers.

Given that the C++11 standard library now includes all the features yaml-cpp currently uses from boost, it woul
d be good to deprecate the use of boost in favour of C++11.
```

It's worth it to migrate from Boost to the C+11/C++14/C++17 standards?

After 20 years of active development, Boost provides many interesting and well tested features and currently the new standards does not cover all the Boost features, which means that If Boost is heavily used be sure that you can't remove the boost dependency. However if only some popular features are used like the Foreach one we can migrate easily to the new standards and remove the Boost dependency.

Boost in 2018...

"Oversimplified C++ Project FAQ 2018"

6. Should you use Boost?

No

Rationale

Boost is a huge and clunky dependency that will explode your build times as soon as you even touch it. And it's 'viral' enough that you can distinguish a Boost project from a non-Boost project. Boost.Optional, Boost.Variant and Boost.Filesystem prepare you for a smooth transition to C++17, but there are other more lightweight alternatives available.

Boost in 2018...

Boost ASIO synchronous UDP client/server

If you know how to write an app that uses an ASIO TCP connection, you are close to know also how to do it on UDP.

Go to the full post

Published: Wednesday, March 21, 2018 0 comment(s) Label: ASIO, C++, socket

Boost ASIO TCP/IP asynchronous server

Having seen how simple is creating a synchronous ASIO TCP/IP server, let's see now how to create an asynchronous one.

Go to the full post

Published: Monday, March 19, 2018 2 comment(s) Label: ASIO, C++, socket

Boost ASIO synchronous exchange on TCP/IP

Let's build a simple synchronous client-server application based on the TCP/IP protocol using the Boost ASIO ip tcp socket. The server waits a connection on a port, as it comes, it writes a message and then terminate. The client connects to the server, reads its message from the socket, outputs it, and then it terminates too.

Boost in 2018...

More TMP with boost::mp11

published at 20.03.2018 23:05

A short blog post on 3 little functions I've written with mp11, to show a bit more how one can work with mp11. The first two are related to working with tags, the last is an easy way to get the member names of a fusion adpated struct into an std::array.

A short TMP experiment with boosts mp11 & fusion

published at 17.03.2018 20:09

Yesterday and today I did some fun coding to play around with boosts new, C++11 TMP library: mp11. I have an idea, and one of the fundamental building blocks would be boost::fusion and its adapt struct infrastructure. I wanted to know if I could combine fusion and mp11 in a good way, to create a certain interface. I'll likely go into the exact ideas and reasoning for this interface later, for now lets focus on how to create it.

Boost in 2018...

Introduction to Boost Phoenix

Published January 26, 2018 - 7 Comments

Example No. 1

variadic templates

Variadic Templates

Today

```
template<class ...Args>
void variadic(Args&&… args)
{
   ...
}
```

In the Past

Boost had “variadic” versions

```
template<class Arg1, class Arg2, class Arg3>
void BoostVariadic(Arg1& a1, Arg2& a2, Arg3& a3)
{
    …
};
```

Variadic Templates

What and how boost go variadic?

```
#if i_ > 0
template<
BOOST_PP_ENUM_PARAMS(i_, typename T)
    >
struct BOOST_PP_CAT(vector,i_)
    : v_item<
          BOOST_PP_CAT(T,BOOST_PP_DEC(i_))
        , AUX778076_VECTOR_TAIL(vector,BOOST_PP_DEC(i_),T)
        >
{
    typedef BOOST_PP_CAT(vector,i_) type;
};
#endif
```

Variadic Templates

What and how boost go variadic?

```
template<
      BOOST_PP_ENUM_PARAMS(i_, typename T)
    >
struct BOOST_PP_CAT(vector,i_)
```

```
#if !defined(BOOST_MPL_LIMIT_VECTOR_SIZE)
#   define BOOST_MPL_LIMIT_VECTOR_SIZE 20
#endif
```

Variadic Templates

What and how boost go variadic?

MPL: Preprocessed, generated headers

%boostroot%/libs/mpl/preprocessed

Python scripts that generate the files for MPL

```
#if !defined(BOOST_MPL_LIMIT_VECTOR_SIZE)
#   define BOOST_MPL_LIMIT_VECTOR_SIZE 20
#endif
```

Variadic Templates

no_ctps:

vector20.hpp

1802 lines of C++

```
namespace boost { namespace mpl {

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10
    >
struct vector11
    : v_item<
          T10
        , vector10< T0,T1,T2,T3,T4,T5,T6,T7,T8,T9 >
        >
{
    typedef vector11 type;
};

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10, typename T11
    >
struct vector12
    : v_item<
          T11
        , vector11< T0,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10 >
        >
{
    typedef vector12 type;
};
```

Variadic Templates

plain

vector20.hpp

1145 lines of C++

```
namespace boost { namespace mpl {

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10
    >
struct vector11
{
    typedef aux::vector_tag<11> tag;
    typedef vector11 type;
    typedef T0 item0;
    typedef T1 item1;
    typedef T2 item2;
    typedef T3 item3;
    typedef T4 item4;
    typedef T5 item5;
    typedef T6 item6;
    typedef T7 item7;
    typedef T8 item8;
    typedef T9 item9;
    typedef T10 item10;


    typedef void_ item11;
    typedef T10 back;
    typedef v_iter< type,0 > begin;
    typedef v_iter< type,11 > end;
};
```

Variadic Templates

typeof_based:

vector20.hpp

160 lines of C++

```
namespace boost { namespace mpl {

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10
    >
struct vector11
    : v_item<
          T10
        , vector10< T0,T1,T2,T3,T4,T5,T6,T7,T8,T9 >
        >
{
    typedef vector11 type;
};

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10, typename T11
    >
struct vector12
    : v_item<
          T11
        , vector11< T0,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10 >
        >
{
    typedef vector12 type;
};

template<
      typename T0, typename T1, typename T2, typename T3, typename T4
    , typename T5, typename T6, typename T7, typename T8, typename T9
    , typename T10, typename T11, typename T12
    >
```

Variadic Templates

What and how boost go variadic?

Fusion does support variadic templates!

Fusion: Preprocessed, generated headers

%boostroot%/libs/fusion/preprocess

using boost wave & boost.build

Variadic Templates

```
namespace boost { namespace fusion
{
    struct vector_tag;
    struct fusion_sequence_tag;
    struct random_access_traversal_tag;
    template <typename T0 , typename T1 , typename T2 , typename T3 , typename T4 , typename T5 , typename T6 , typename T7 , typename T8 , typename T9 , typename T10>
    struct vector_data11
    {
        BOOST_CONSTEXPR BOOST_FUSION_GPU_ENABLED
        vector_data11()
            : m0() , m1() , m2() , m3() , m4() , m5() , m6() , m7() , m8() , m9() , m10() {}
# if !defined(BOOST_NO_CXX11_RVALUE_REFERENCES)
        template <typename U0 , typename U1 , typename U2 , typename U3 , typename U4 , typename U5 , typename U6 , typename U7 , typename U8 , typename U9 , typename U10>
# if !defined(BOOST_CLANG)
        BOOST_CXX14_CONSTEXPR
# endif
        BOOST_FUSION_GPU_ENABLED
        vector_data11(U0 && arg0 , U1 && arg1 , U2 && arg2 , U3 && arg3 , U4 && arg4 , U5 && arg5 , U6 && arg6 , U7 && arg7 , U8 && arg8 , U9 && arg9 , U10 && arg10
          , typename boost::enable_if<is_convertible<U0, T0> >::type* = 0
        )
            : m0(std::forward<U0>( arg0)) , m1(std::forward<U1>( arg1)) , m2(std::forward<U2>( arg2)) , m3(std::forward<U3>( arg3)) , m4(std::forward<U4>( arg4)) , m5(std::forward<U5>( arg5))
        BOOST_CXX14_CONSTEXPR BOOST_FUSION_GPU_ENABLED
        vector_data11(
            vector_data11&& other)
            : m0(std::forward<T0>( other.m0)) , m1(std::forward<T1>( other.m1)) , m2(std::forward<T2>( other.m2)) , m3(std::forward<T3>( other.m3)) , m4(std::forward<T4>( other.m4)) , m5(std::
# endif
# if !defined(BOOST_CLANG)
    BOOST_CONSTEXPR
# endif
        BOOST_FUSION_GPU_ENABLED
        vector_data11(
            typename detail::call_param<T0 >::type arg0 , typename detail::call_param<T1 >::type arg1 , typename detail::call_param<T2 >::type arg2 , typename detail::call_param<T3 >::type arg
            : m0(arg0) , m1(arg1) , m2(arg2) , m3(arg3) , m4(arg4) , m5(arg5) , m6(arg6) , m7(arg7) , m8(arg8) , m9(arg9) , m10(arg10) {}
        BOOST_CONSTEXPR BOOST_FUSION_GPU_ENABLED
        vector_data11(
            vector_data11 const& other)
            : m0(other.m0) , m1(other.m1) , m2(other.m2) , m3(other.m3) , m4(other.m4) , m5(other.m5) , m6(other.m6) , m7(other.m7) , m8(other.m8) , m9(other.m9) , m10(other.m10) {}
        BOOST_CXX14_CONSTEXPR BOOST_FUSION_GPU_ENABLED
        vector_data11&
        operator=(vector_data11 const& vec)
        {
            this->m0 = vec.m0; this->m1 = vec.m1; this->m2 = vec.m2; this->m3 = vec.m3; this->m4 = vec.m4; this->m5 = vec.m5; this->m6 = vec.m6; this->m7 = vec.m7; this->m8 = vec.m8; this->m9
            return *this;
        }
        template <typename Sequence>
```

fusion/preprocessed/vector20.hpp

1825 lines of C++ for the vector_data11 – vector_data20 templates

Variadic Templates

variant, tuple and a few others also have their ways to support variadic like templates pre C++11.

But there is no central solution which solved this problem in boost.

```
template<BOOST_VARIADIC_TEMPLATE(?)>
```

Overview on variadic templates & boost

Boost pioneered in providing interfaces

This is today partly technical dept in boost though.

Boost MPL & Fusion

Simulated variadic template like interfaces for C++03

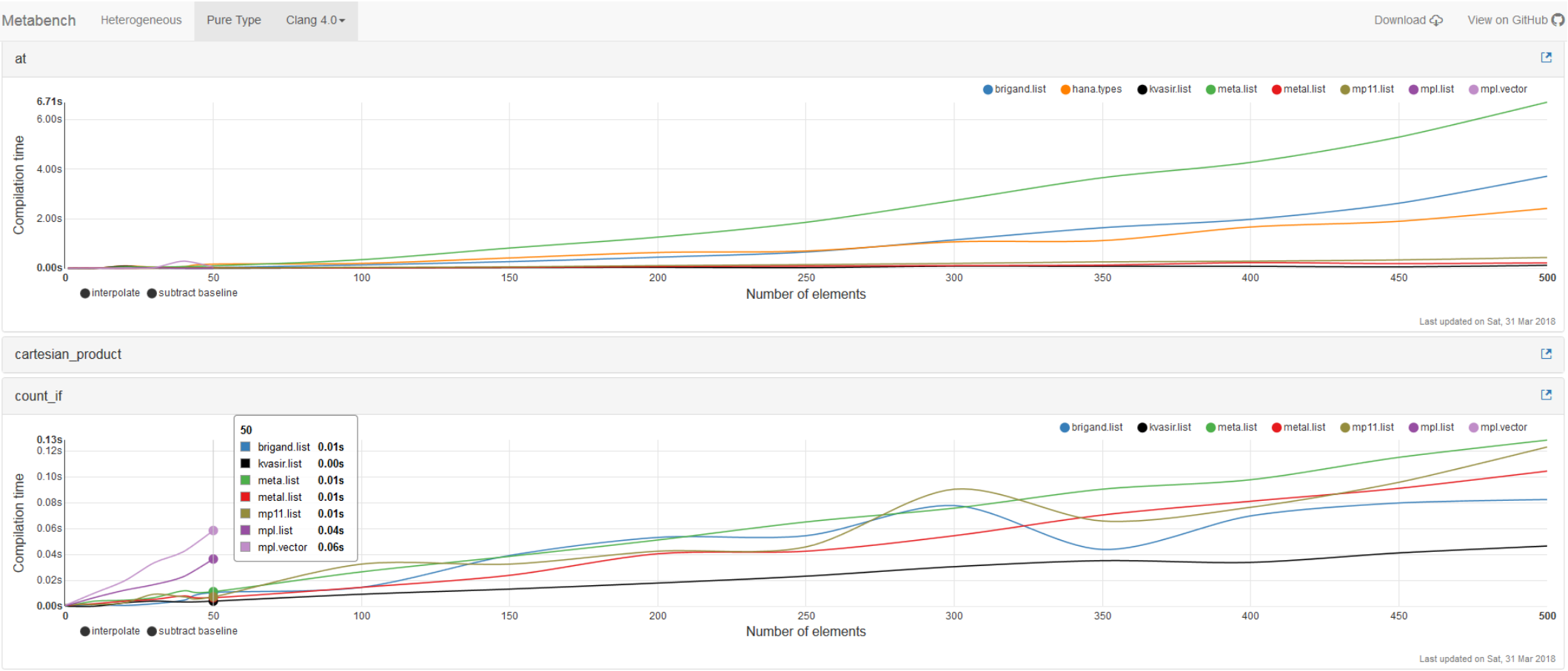
Fusion today also supports variadic templates

Other boost libraries chose their own ways of doing this or that.

Example No. 2

TMP

Template Meta Programming



TMP - Wikipedia

Components of template metaprogramming [edit]

The use of templates as a metaprogramming technique requires two distinct operations: a template must be defined, and a defined template must be instantiated. The template definition describes the generic form of the generated source code, and the instantiation causes a specific set of source code to be generated from the generic form in the template.

Template metaprogramming is Turing-complete, meaning that any computation expressible by a computer program can be computed, in some form, by a template metaprogram.[3]

Templates are different from *macros*. A macro, which is also a compile-time language feature, generates code in-line using text manipulation and substitution. Macro systems often have limited compile-time process flow abilities and usually lack awareness of the semantics and type system of their companion language (a notable exception being Lisp macros, which, as a result of Lisp's homoiconicity, are written in Lisp itself and work by directly elaborating the abstract syntax tree of the to-be generated code, rather than by producing and substituting text to be parsed and compiled in later steps).

Template metaprograms have no mutable variables— that is, no variable can change value once it has been initialized, therefore template metaprogramming can be seen as a form of functional programming. In fact many template implementations implement flow control only through recursion, as seen in the example below.

Using template metaprogramming [edit]

Though the syntax of template metaprogramming is usually very different from the programming language it is used with, it has practical uses. Some common reasons to use templates are to implement generic programming (avoiding sections of code which are similar except for some minor variations) or to perform automatic compile-time optimization such as doing something once at compile time rather than every time the program is run — for instance, by having the compiler unroll loops to eliminate jumps and loop count decrements whenever the program is executed.

```
template <unsigned int n>
struct factorial {
    enum { value = n * factorial<n - 1>::value };
};

template <>
struct factorial<0> {
    enum { value = 1 };
};
```

Introduction

TMP and ...

boost

TMP and ...

mpl

TMP and ...

hana

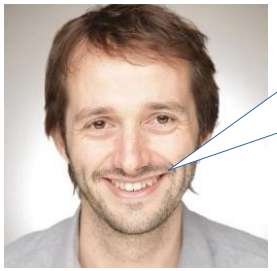
TMP and ...

new ideas

TMP and ...

What about aliases?

Have you tried type based TMP with aliases?



TMP and ...

Brigand

Kvasir

metal

TMP and ...

mp11

TMP and ...

boost

hana ***mp11***

TMP and ...

peak TMP

TMP and ...

constexpr

TMP and ...

BEN DEANE / bdeane@blizzard.com / @ben_deane

JASON TURNER / jason@emptycrate.com / @lefticus

TMP and ...

if constexpr

Cutting Edge

Solutions

Cutting Edge - Solutions

boost

Cutting Edge - Solutions

boost

boost libraries vs standard libraries

Cutting Edge - Solutions

C++ Standard

_Cpp17{
...
}
cpp20{
...
}

Macros & Aliases
Customization Points

C++ is now a dynamic language

too Cutting Edge…

C++Now

Questions?

Thanks!

Meeting C++ 2018

- Submit your talks
- Tickets available

New Features at Meeting C++

- Jobsection
  - Upload your own jobs
  - Employer listing
  - CV Submission form
- Soon
  - Contact C++ Trainers
  - more coming!
- Patreon
  - https://www.patreon.com/meetingcpp

Meeting C++ 2018

Ticketshop | Location

Submit your talk to Meeting C++ today!

Keynotes

Andrei Alexandrescu

Lisa Lippincott

Nicolai Josuttis