When order does not matter

a different approach to C++ functions' arguments passing

Bartek 'BaSz' Szurgot

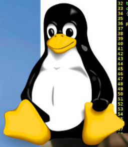
`http://www.baszerr.eu`

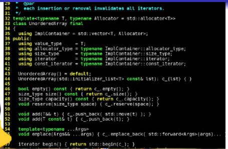
2015-06-17

Part 1

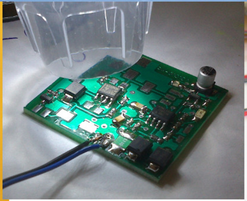
Code for food

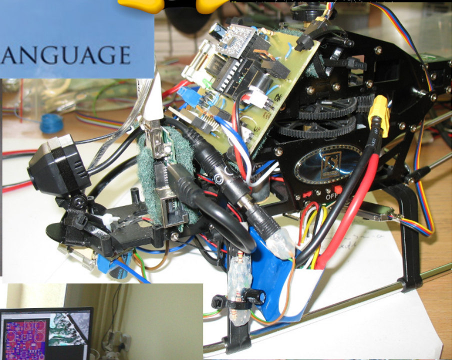
Professional geek

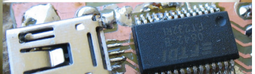




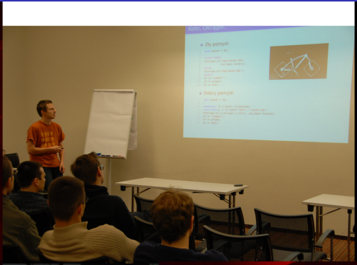
```
 * @par
 *  each insertion or removal invalidates all iterators.
 */
template<typename T, typename Allocator = std::allocator<T>>
class UnorderedArray final
{
  using ImplContainer = std::vector<T, Allocator>;
public:
  using value_type     = T;
  using allocator_type = typename ImplContainer::allocator_type;
  using size_type      = typename ImplContainer::size_type;
  using iterator       = typename ImplContainer::iterator;
  using const_iterator = typename ImplContainer::const_iterator;

  UnorderedArray() = default;
  UnorderedArray(std::initializer_list<T> const& lst): c_{lst} { }

  bool empty() const { return c_.empty(); }
  size_type size() const { return c_.size(); }
  size_type capacity() const { return c_.capacity(); }
  void reserve(size_type space) { c_.reserve(space); }

  void add(T&& t) { c_.push_back( std::move(t) ); }
  void add(T const& t) { c_.push_back(t); }

  template<typename ...Args>
  void emplace(Args&& ...args) { c_.emplace_back( std::forward<Args>(args)...

  iterator begin() { return std::begin(c_); }
```

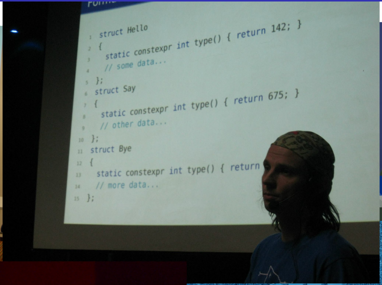




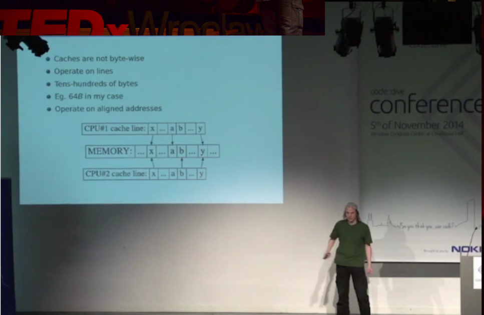


Occasional speaker



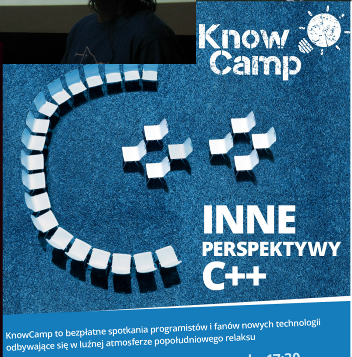


```
struct Hello
{
  static constexpr int type() { return 142; }
  // some data...
};
struct Say
{
  static constexpr int type() { return 675; }
  // other data...
};
struct Bye
{
  static constexpr int type() { return
  // more data...
};
```



- Caches are not byte-wise
- Operate on lines
- Tens-hundreds of bytes
- Eg. 64B in my case
- Operate on aligned addresses

CPU#1 cache line: x ... a b ... y
MEMORY: ... x ... a b ... y ...
CPU#2 cache line: x ... a b ... y
code::dive
conference
5th of November 2014
NOKI


KnowCamp
INNE
PERSPEKTYWY
C++
KnowCamp to bezpłatne spotkania programistów i fanów nowych technologii odbywające się w luźnej atmosferze popołudniowego relaksu

Weekend blogger

http://www.baszerr.eu/doku.php/blog/blog

Part 2

Background

- Beginning of 2014
- Happy programming
- First assignment
- Mission – framework:
  - Messaging (production)
  - Mocking (testing)
- Focus – **testing**
- Unordered arguments. . .

http://upload.wikimedia.org/wikipedia/commons/a/a1/Bundesarchiv_B_145_Bild-F031434-0006%2C_Aachen%2C_Technische_Hochschule%2C_Rechenzentrum.jpg

Regular function call

```cpp
void someFunc(Type1, Type2, Type3);
// ...
const Type1 t1;
const Type2 t2;
const Type3 t3;
// ...
someFunc(t1, t2, t3);        // ok!
//someFunc(t2, t3, t1);      // oops...
```

- Argument order does matter
- Wrong order == compilation failure

Good or bad?

- Pros
  - Types can repeat
  - Strong typing
  - Type safety
  - Early errors detections
    - Earlier == better
    - Earlier == cheaper
    - Compare – scripting languages. . . ;)

Good or bad?

- Pros
  - Types can repeat
  - Strong typing
  - Type safety
  - Early errors detections
    - Earlier == better
    - Earlier == cheaper
    - Compare – scripting languages. . . ;)
- Yes, but. . . (aka: cons)
  - Need to remember **The Order**™
  - Also for unique types
  - Defaults in the middle?

http://upload.wikimedia.org/wikipedia/commons/d/d0/Balance_scale_IMGP9755.jpg

Conclusion

- Generally good

Conclusion

- Generally good
- Unless not

Conclusion

- Generally good
- Unless not

- What to do?

http://upload.wikimedia.org/wikipedia/commons/7/7e/Week_z00.jpg

Right tool for the job!

http://upload.wikimedia.org/wikipedia/commons/8/84/Claw-hammer.jpg

Real-life example

- New messaging framework
- Simple API:

```
struct MyHandler // ...
{
  // ...
  void handle(Context const& ctx, MessageOne const& msg)
  {
    // .. user's code ...
  }
};
```

Real-life example

- New messaging framework
- Simple API:

```
struct MyHandler // ...
{
  // ...
  void handle(Context const& ctx, MessageOne const& msg)
  {
    // .. user's code ...
  }
};
```

- Need testing framework
- Usable mocks wanted!

Context-ual problem...

```
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

Context-ual problem...

```cpp
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

- Many fields
  - Generated. . .
  - Unpredictable. . .
  - Uninteresting. . .
- Often used in tests

Context-ual problem...

```cpp
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

- Many fields
  - Generated. . .
  - Unpredictable. . .
  - Uninteresting. . .
- Often used in tests

- Tricky part!
- EXPECT_CALL on that?!

Ideal solution

```
struct HandlerMock
{
  MOCK_METHOD2(handle, void(Context const&, MessageOne const&));
  // ...
};
// ...
HandlerMock mock;

EXPECT_CALL( mock, handle(
      makeContext( Sender{/*...*/}, Receiver{/*...*/} /* ignore rest */ ),
      MessageOne{/*...*/} ) );
```

Ideal solution

```
struct HandlerMock
{
  MOCK_METHOD2(handle, void(Context const&, MessageOne const&));
  // ...
};
// ...
HandlerMock mock;

EXPECT_CALL( mock, handle(
      makeContext( Sender{/*...*/}, Receiver{/*...*/} /* ignore rest */ ),
      MessageOne{/*...*/} ) );
```

Ideal solution

```
struct HandlerMock
{
  MOCK_METHOD2(handle, void(Context const&, MessageOne const&));
  // ...
};
// ...
HandlerMock mock;

EXPECT_CALL( mock, handle(
      makeContext( Sender{/*...*/}, Receiver{/*...*/} /* ignore rest */ ),
      MessageOne{/*...*/} ) );
```

- makeContext() problems:
  - Arity – $O(N)$ options
  - Different order – $O(N!)$ overloads each
- $O(N!)$? Nah...

Part 3

C++ programmer’s best f(r)iend

C++ programmer’s best f(r)iend

```
template<>
```

Repeating without a loop

```cpp
template<int on, int times>
struct repeat
{
  template<typename F>
  static void call(F f)
  { f(); repeat<on+1, times>::call(f); }
};

template<int end>
struct repeat<end,end>
{
  template<typename F>
  static void call(F) { }
};
```

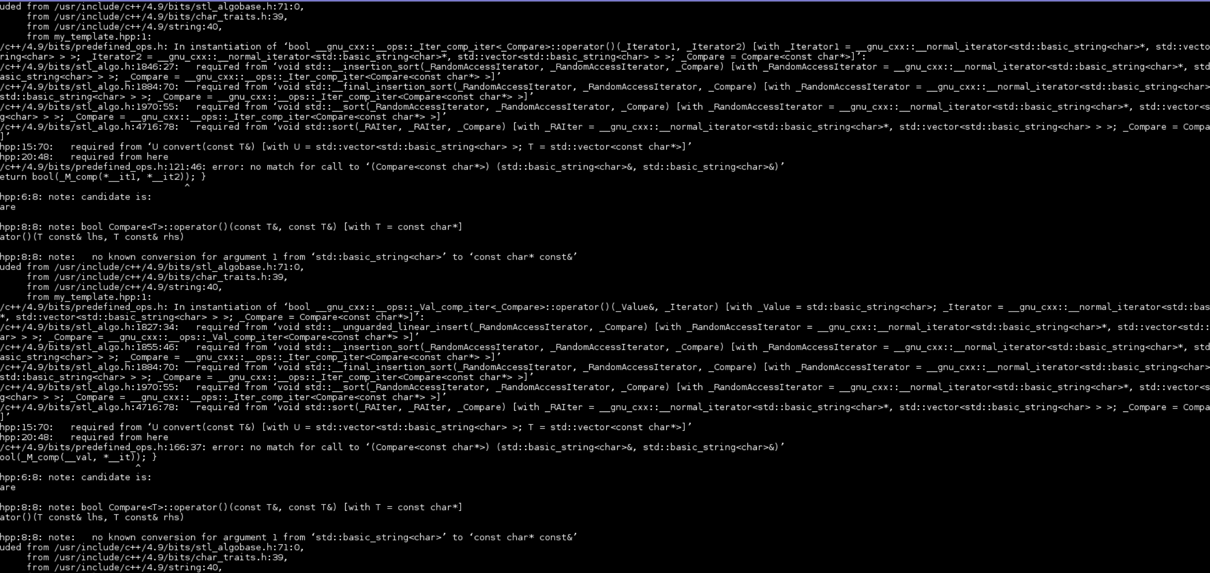
Work saver!

```
#include "repeat.hpp"

void punishment()
{
  cout << "i␣will␣not␣be␣lazy" << endl;
}

int main()
{
  repeat<0,300>::call(punishment);
}
```

Can be helpful

```
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
i will not be lazy
```

Or whimsy...



```
uded from /usr/include/c++/4.9/bits/stl_algobase.h:71:0,
     from /usr/include/c++/4.9/bits/char_traits.h:39,
     from /usr/include/c++/4.9/string:40,
     from my_template.hpp:1:
/c++/4.9/bits/predefined_ops.h: In instantiation of ‘bool __gnu_cxx::__ops::_Iter_comp_iter<_Compare>::operator()(_Iterator1, _Iterator2) [with _Iterator1 = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vecto
ring<char> > >; _Iterator2 = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<std::basic_string<char> > >; _Compare = Compare<const char*>]’:
/c++/4.9/bits/stl_algo.h:1846:27:   required from ‘void std::__insertion_sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std
asic_string<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:1884:70:   required from ‘void std::__final_insertion_sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>
std::basic_string<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:1970:55:   required from ‘void std::__sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<s
g<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:4716:78:   required from ‘void std::sort(_RAIter, _RAIter, _Compare) [with _RAIter = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<std::basic_string<char> > >; _Compare = Compa
]’
hpp:15:70:   required from ‘U convert(const T&) [with U = std::vector<std::basic_string<char> >; T = std::vector<const char*>]’
hpp:20:48:   required from here
/c++/4.9/bits/predefined_ops.h:121:46: error: no match for call to ‘(Compare<const char*>) (std::basic_string<char>&, std::basic_string<char>&)’
eturn bool(_M_comp(*__it1, *__it2)); }
                                 ^
hpp:6:8: note: candidate is:
are

hpp:8:8: note: bool Compare<T>::operator()(const T&, const T&) [with T = const char*]
ator()(T const& lhs, T const& rhs)

hpp:8:8: note:   no known conversion for argument 1 from ‘std::basic_string<char>’ to ‘const char* const&’
uded from /usr/include/c++/4.9/bits/stl_algobase.h:71:0,
     from /usr/include/c++/4.9/bits/char_traits.h:39,
     from /usr/include/c++/4.9/string:40,
     from my_template.hpp:1:
/c++/4.9/bits/predefined_ops.h: In instantiation of ‘bool __gnu_cxx::__ops::_Val_comp_iter<_Compare>::operator()(_Value&, _Iterator) [with _Value = std::basic_string<char>; _Iterator = __gnu_cxx::__normal_iterator<std::bas
*, std::vector<std::basic_string<char> > >; _Compare = Compare<const char*>]’:
/c++/4.9/bits/stl_algo.h:1827:34:   required from ‘void std::__unguarded_linear_insert(_RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<std::
ar> > >; _Compare = __gnu_cxx::__ops::_Val_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:1855:46:   required from ‘void std::__insertion_sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std
asic_string<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:1884:70:   required from ‘void std::__final_insertion_sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>
std::basic_string<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:1970:55:   required from ‘void std::__sort(_RandomAccessIterator, _RandomAccessIterator, _Compare) [with _RandomAccessIterator = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<s
g<char> > >; _Compare = __gnu_cxx::__ops::_Iter_comp_iter<Compare<const char*> >]’
/c++/4.9/bits/stl_algo.h:4716:78:   required from ‘void std::sort(_RAIter, _RAIter, _Compare) [with _RAIter = __gnu_cxx::__normal_iterator<std::basic_string<char>*, std::vector<std::basic_string<char> > >; _Compare = Compa
]’
hpp:15:70:   required from ‘U convert(const T&) [with U = std::vector<std::basic_string<char> >; T = std::vector<const char*>]’
hpp:20:48:   required from here
/c++/4.9/bits/predefined_ops.h:166:37: error: no match for call to ‘(Compare<const char*>) (std::basic_string<char>&, std::basic_string<char>&)’
ool(_M_comp(__val, *__it)); }
                         ^
hpp:6:8: note: candidate is:
are

hpp:8:8: note: bool Compare<T>::operator()(const T&, const T&) [with T = const char*]
ator()(T const& lhs, T const& rhs)

hpp:8:8: note:   no known conversion for argument 1 from ‘std::basic_string<char>’ to ‘const char* const&’
uded from /usr/include/c++/4.9/bits/stl_algobase.h:71:0,
     from /usr/include/c++/4.9/bits/char_traits.h:39,
     from /usr/include/c++/4.9/string:40,
```

By the end of the day...

- Templates are good
  - Powerful tool
  - Metaprogramming-enablers

By the end of the day...

- Templates are good
  - Powerful tool
  - Metaprogramming-enablers
- Language within the language
  - Add new possibilities
  - Extend at will!

By the end of the day...

- Templates are good
  - Powerful tool
  - Metaprogramming-enablers
- Language within the language
  - Add new possibilities
  - Extend at will!
- Use with care
  - Great for backend
  - VooDoo for experts
  - Avoid user-contact

http://upload.wikimedia.org/wikipedia/commons/c/c0/Biohazard_symbol.svg

Some theory

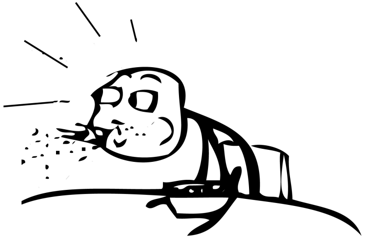
Theorem 1

Any, arbitrary complex problem can be solved using finite number of templates.

Some theory

Theorem 1

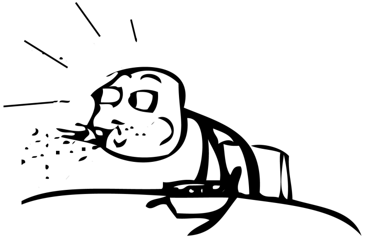
Any, arbitrary complex problem can be solved using finite number of templates.



Some theory

Theorem 1

Any, arbitrary complex problem can be solved using finite number of templates.



Theorem 2

Computers with sufficient amount of random access memory and fast enough processors do not exist yet.

http://i.imgur.com/s6uyF.jpg

Part 4

MPL warning

Bring it on!

http://www.giornalettismo.com/wp-content/uploads/2012/10/Awesome-Behind-The-Scenes-Photos-from-Horror-Movies-31.jpg

Context (reminder)

```
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

- Simple structure
- Brace-initializable
- All types unique

Context (reminder)

```
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

- Simple structure
- Brace-initializable
- All types unique

- How to fill it up?
- Cannot create directly...

Context (reminder)

```
struct Context
{
  Clock::time_point timestamp_;
  MessageId         id_;
  ReplyToId         replyToId_;
  Sender            sender_;
  Receiver          receiver_;
};
```

- Simple structure
- Brace-initializable
- All types unique
- How to fill it up?
- Cannot create directly...
- Layer of indirection
- Use a helper function
- ...

Helper function

```cpp
#include "Context.hpp"

template<typename ...Args>
Context makeContext(Args&&... args)
{
  return Context{
                // extract & assign filed 1
                // extract & assign filed 2
                // ...
                // extract & assign filed N
              };
}
```

Function’s implementation

```
#include "Context.hpp"
#include "extract.hpp"

template<typename ...Args>
Context makeContext(Args&&... args)
{
  return {
             extract<Clock::time_point>::from( std::forward<Args>(args)... ),
             extract<MessageId>::from( std::forward<Args>(args)... ),
             extract<ReplyToId>::from( std::forward<Args>(args)... ),
             extract<Sender>::from( std::forward<Args>(args)... ),
             extract<Receiver>::from( std::forward<Args>(args)... )
           };
}
```

Looks like fun?

`http://www.chud.com/wp-content/uploads/2011/04/critters20.png`

Argument extraction - interface

```
template<typename T>
struct extract
{
  static T from();

  template<typename Head, typename ...Tail>
  static T from(Head&& h, Tail&&... t);

private:
  template<typename Head, typename ...Tail>
  static T fromImpl(std::true_type, Head&& h, Tail&&...);

  template<typename Head, typename ...Tail>
  static T fromImpl(std::false_type, Head&&, Tail&&... t);
};
```

Argument extraction - interface

```cpp
template<typename T>
struct extract
{
  static T from();

  template<typename Head, typename ...Tail>
  static T from(Head&& h, Tail&&... t);

private:
  template<typename Head, typename ...Tail>
  static T fromImpl(std::true_type, Head&& h, Tail&&...);

  template<typename Head, typename ...Tail>
  static T fromImpl(std::false_type, Head&&, Tail&&... t);
};
```

Interface implementation

```cpp
template<typename T>
T extract<T>::from() { return {}; }

template<typename T>
template<typename Head, typename ...Tail>
T extract<T>::from(Head&& h, Tail&&... t)
{
  using DT = typename std::decay<T>::type;
  using DH = typename std::decay<Head>::type;
  constexpr auto same = std::is_same<DT,DH>{};
  return fromImpl( same,
                   std::forward<Head>(h),
                   std::forward<Tail>(t)... );
}
```

Argument extraction - internals

```
template<typename T>
struct extract
{
  static T from();

  template<typename Head, typename ...Tail>
  static T from(Head&& h, Tail&&... t);

private:
  template<typename Head, typename ...Tail>
  static T fromImpl(std::true_type, Head&& h, Tail&&...);

  template<typename Head, typename ...Tail>
  static T fromImpl(std::false_type, Head&&, Tail&&... t);
};
```

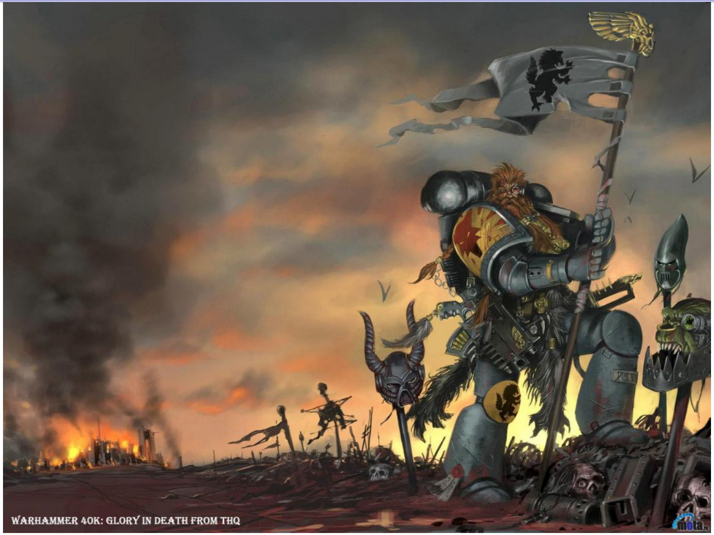
Argument extraction - internals

```cpp
template<typename T>
struct extract
{
  static T from();

  template<typename Head, typename ...Tail>
  static T from(Head&& h, Tail&&... t);

private:
  template<typename Head, typename ...Tail>
  static T fromImpl(std::true_type, Head&& h, Tail&&...);

  template<typename Head, typename ...Tail>
  static T fromImpl(std::false_type, Head&&, Tail&&... t);
};
```

Internal details

```cpp
template<typename T>
template<typename Head, typename ...Tail>
T extract<T>::fromImpl(std::true_type, Head&& h, Tail&&...)
{
  return std::forward<Head>(h);
}

template<typename T>
template<typename Head, typename ...Tail>
T extract<T>::fromImpl(std::false_type, Head&&, Tail&&... t)
{
  return from( std::forward<Tail>(t)... );
}
```

We’ve made it!



http://img.mota.ru/upload/wallpapers/2009/07/18/11/03/19788/warhammer_40000_-_gid_001-1152x864.jpg

Part 5

All carefully planned...

I can see no way in which this
carefully laid plan could ever fail.

http://img2.owned.com/media/images/1/3/6/9/13694/what_could_possibly_go_wrong_here_540.jpg

User strikes back

- *Now what it i just. . .*

```
std::string sender{"Mr.␣Evil"};
// ...
auto ctx = makeContext( sender,      // unsupported type
                        Receiver{/*...*/},
                        ReplyToId{/*...*/} );
```

User strikes back

- *Now what it i just. . .*

```
std::string sender{"Mr.␣Evil"};
// ...
auto ctx = makeContext( sender,      // unsupported type
                        Receiver{/*...*/},
                        ReplyToId{/*...*/} );
```

- Type-typo
- Compiles fine
- Unused at runtime
- Confusing. . .

http://conversationswithdonmachingaandotherbeings.files.wordpress.com/2013/05/the-empire-strikes-back2.gif

Step 1: all types are valid?

```cpp
#include "ValidType.hpp"

template<typename ...Args>
struct CheckAllValid
{
  static constexpr bool value = true;
};

template<typename Head, typename ...Tail>
struct CheckAllValid<Head, Tail...>
{
  using H = typename std::decay<Head>::type;
  static constexpr bool value = ValidType<H>::value &&
                                CheckAllValid<Tail...>::value;
};
```

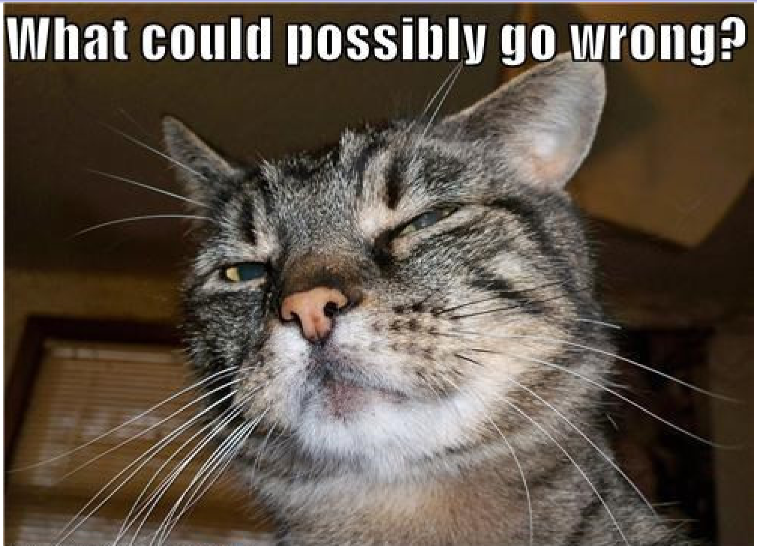
Step 2: given type is valid?

```
template<typename T>
struct ValidType: std::false_type { };

template<>
struct ValidType<Clock::time_point>: std::true_type { };

template<>
struct ValidType<MessageId>: std::true_type { };

template<>
struct ValidType<ReplyToId>: std::true_type { };

template<>
struct ValidType<Sender>: std::true_type { };

template<>
struct ValidType<Receiver>: std::true_type { };
```

Step 3: glue it up!

```cpp
#include "Context.hpp"
#include "extract.hpp"
#include "CheckAllValid.hpp"

template<typename ...Args>
Context makeContext(Args&&... args)
{
  static_assert( CheckAllValid<Args...>::value, "unknown␣type␣detected");
  return {
           extract<Clock::time_point>::from( std::forward<Args>(args)... ),
           extract<MessageId>::from( std::forward<Args>(args)... ),
           extract<ReplyToId>::from( std::forward<Args>(args)... ),
           extract<Sender>::from( std::forward<Args>(args)... ),
           extract<Receiver>::from( std::forward<Args>(args)... )
         };
}
```

Are we done here?



http://www.raymondcamden.com/images/download1.jpg

Not quite there yet

http://confessionsofatrolleydolly.files.wordpress.com/2013/01/commercial-airplane-engine-fire.jpg

Return of the user

- *Now what it i just. . .*

```
auto ctx = makeContext( Sender{/*...*/},
                        Sender{/*...*/},    // Receiver{} ?
                        ReplyToId{/*...*/} );
```

Return of the user

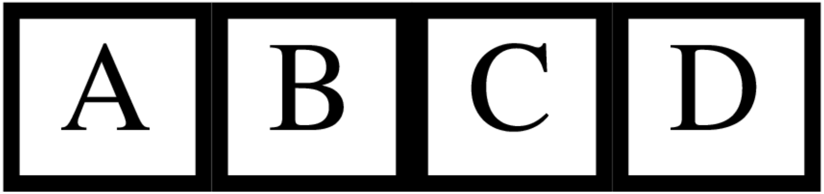
- *Now what it i just. . .*

```
auto ctx = makeContext( Sender{/*...*/},
                        Sender{/*...*/},    // Receiver{} ?
                        ReplyToId{/*...*/} );
```

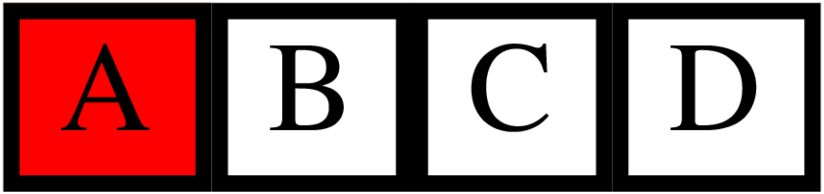
- More typos?
- Which value to take?
- Second value ignored
- Yet it compiles
- Confusing. . .

http://moviehaku.com/img/gallery/large/4-Return-of-the-Jedi.jpg

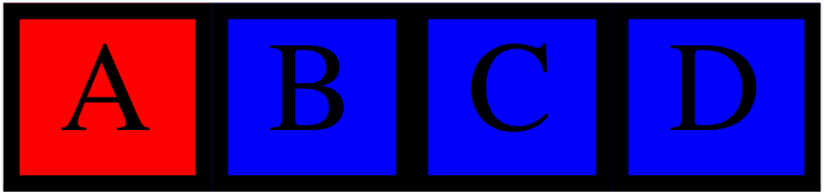
Uniqueness checking concept



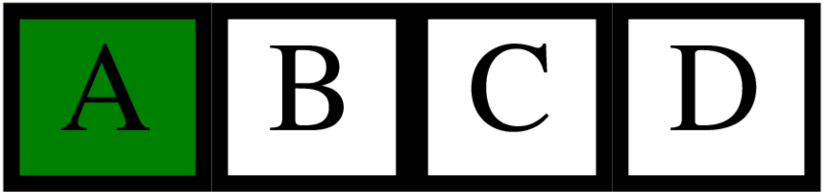
Uniqueness checking concept



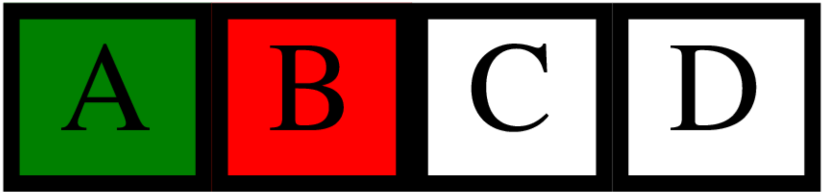
Uniqueness checking concept



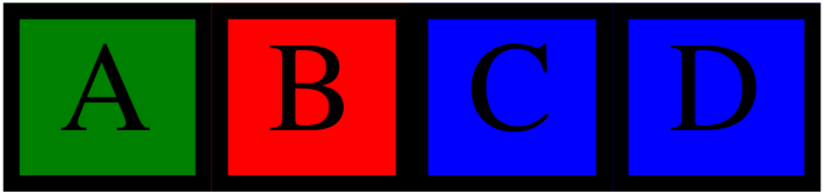
Uniqueness checking concept



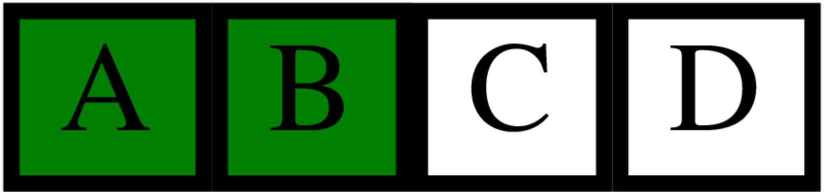
Uniqueness checking concept



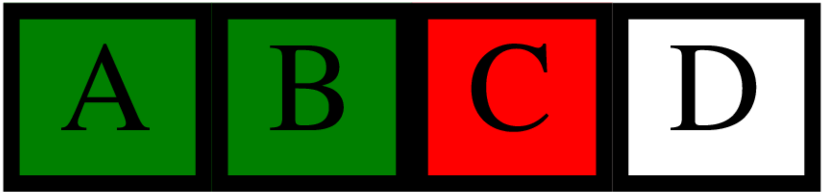
Uniqueness checking concept



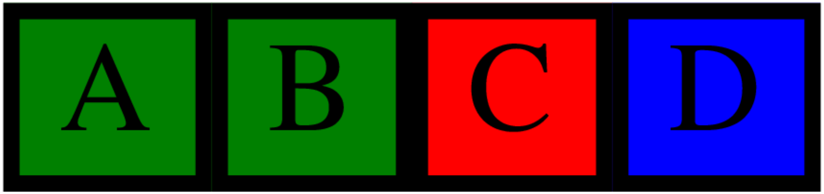
Uniqueness checking concept



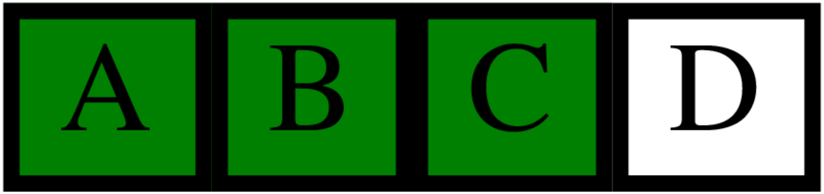
Uniqueness checking concept



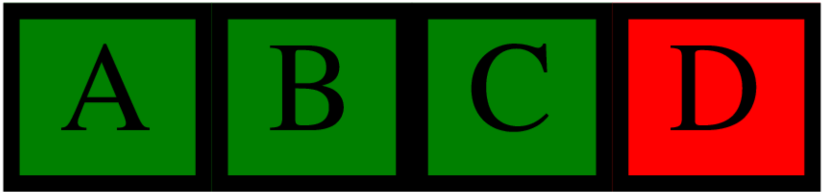
Uniqueness checking concept



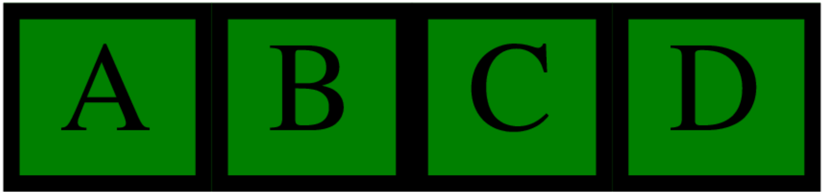
Uniqueness checking concept



Uniqueness checking concept



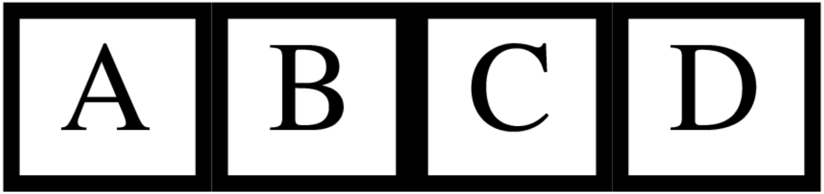
Uniqueness checking concept



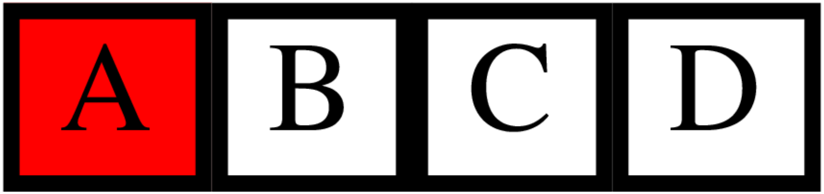
Step 1: all types are unique?

```
#include "Has.hpp"

template<typename ...Args>
struct Unique
{
  static constexpr bool value = true;
};

template<typename Head, typename ...Tail>
struct Unique<Head, Tail...>
{
  static constexpr bool value = not Has<Head, Tail...>::value &&
                                Unique<Tail...>::value;
};
```

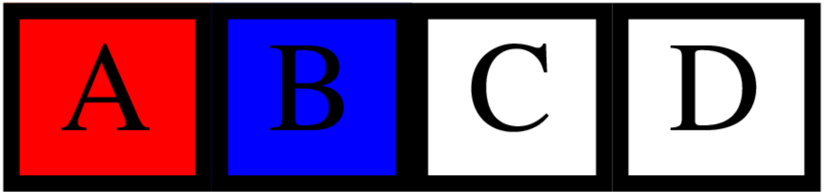
"Has<>" algorithm



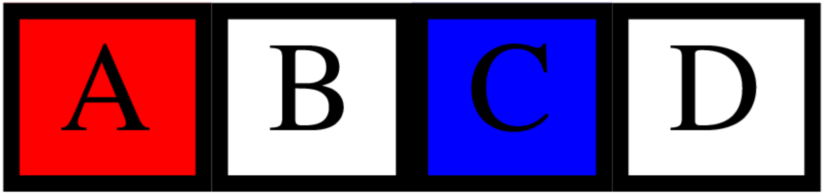
"Has<>" algorithm



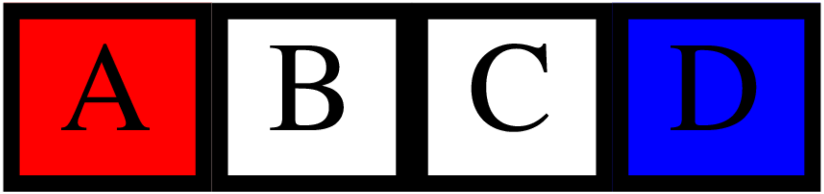
"Has<>" algorithm



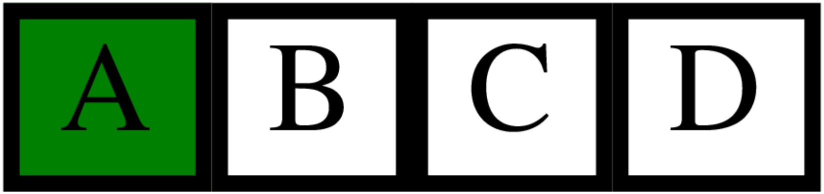
"Has<>" algorithm



"Has<>" algorithm



"Has<>" algorithm



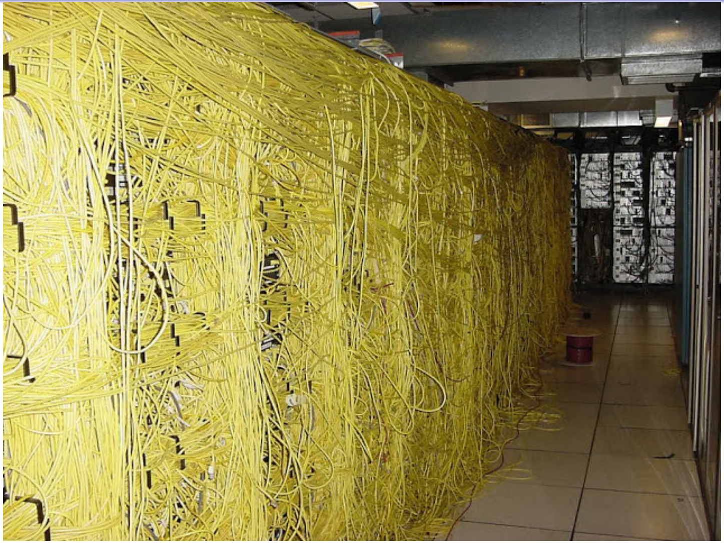
Step 2: given type is (not) unique?

```cpp
#include <type_traits>

template<typename ...Args>
struct Has
{
  static constexpr bool value = false;
};

template<typename T, typename Head, typename ...Tail>
struct Has<T, Head, Tail...>
{
  using DT = typename std::decay<T>::type;
  using DH = typename std::decay<Head>::type;
  static constexpr bool same  = std::is_same<DT,DH>::value;
  static constexpr bool value = same || Has<T, Tail...>::value;
};
```

Step 3: glue it up!

```
#include "Context.hpp"
#include "extract.hpp"
#include "CheckAllValid.hpp"
#include "Unique.hpp"

template<typename ...Args>
Context makeContext(Args&&... args)
{
  static_assert( CheckAllValid<Args...>::value, "unknown␣type␣detected");
  static_assert( Unique<Args...>::value, "non-unique␣type␣detected");
  return {
           extract<Clock::time_point>::from( std::forward<Args>(args)... ),
           extract<MessageId>::from( std::forward<Args>(args)... ),
           extract<ReplyToId>::from( std::forward<Args>(args)... ),
           extract<Sender>::from( std::forward<Args>(args)... ),
           extract<Receiver>::from( std::forward<Args>(args)... )
         };
}
```

Everything in place



http://lh3.ggpht.com/abramsv/R-8V6vXRjSI/AAAAAAAANC4/RLBSbFxaGiA/s640/2036057464_0cc80962b6_o.jpg

Part 6

Tested application ('no-order')

```cpp
template<typename ...Args>
Context makeContext(Args&&... args);

void sink(Context const&); // does nothing

int main()
{
  for(auto i=0; i<10*1000*1000; ++i)
  {
    const auto ctx = makeContext(/*arg 1, arg 2, ..., arg N*/);
    sink(ctx);                  // disable optimizing ctx away
  }
}
```

Reference application (’regular’)

```
// makeContext() delcared in the same translation unit
Context makeContext(/*arg 1, arg 2, ..., arg N*/);

void sink(Context const&); // does nothing

int main()
{
  for(auto i=0; i<10*1000*1000; ++i)
  {
    const auto ctx = makeContext(/*arg 1, arg 2, ..., arg N*/);
    sink(ctx);                     // disable optimizing ctx away
  }
}
```

Test conditions

- Compilers:
  - GCC-5.1 with libstdc++
  - Clang-3.6 with libc++

Test conditions

- Compilers:
  - GCC-5.1 with libstdc++
  - Clang-3.6 with libc++
- Type:
  - Speed: -O3
  - Size: -Os

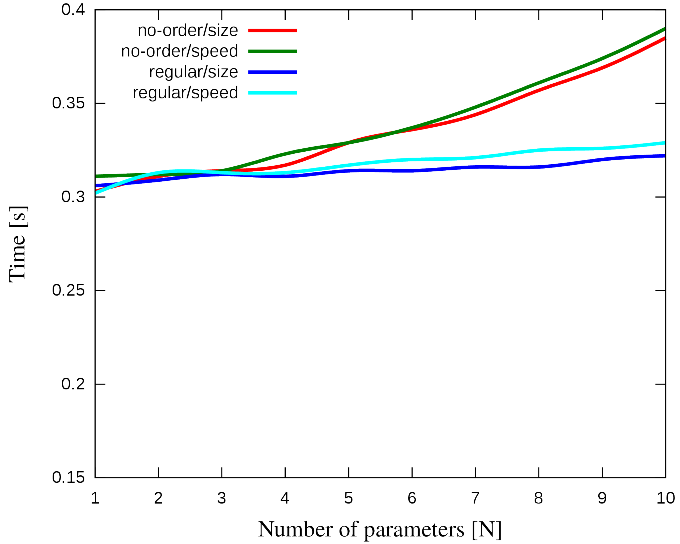
Test conditions

- Compilers:
  - GCC-5.1 with libstdc++
  - Clang-3.6 with libc++
- Type:
  - Speed: `-O3`
  - Size: `-Os`
- Common flags: `-DNDEBUG -s -march=native`
- No LTO (note: `sink()` function)
- Measurements averaged from 10 runs

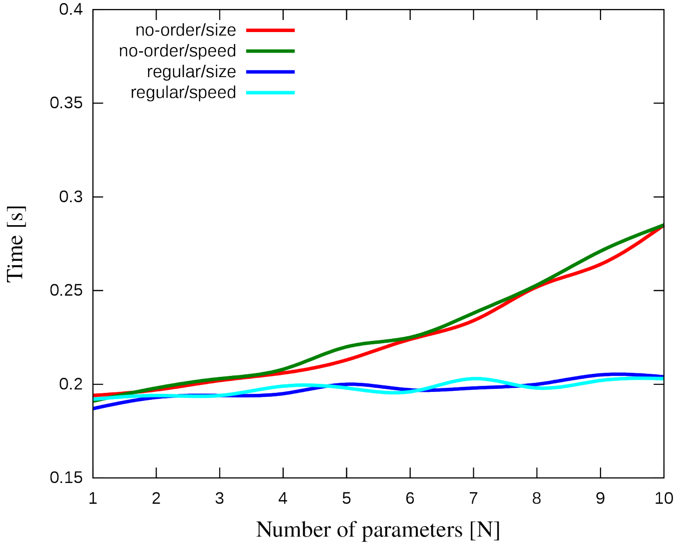
Compilation time

http://3.bp.blogspot.com/-7lT5HlRH92M/UMNoO1_UifI/AAAAAAAAAl4/fiU7UGltqr8/s1600/Black_Hole.jpg

Compilation time - GCC



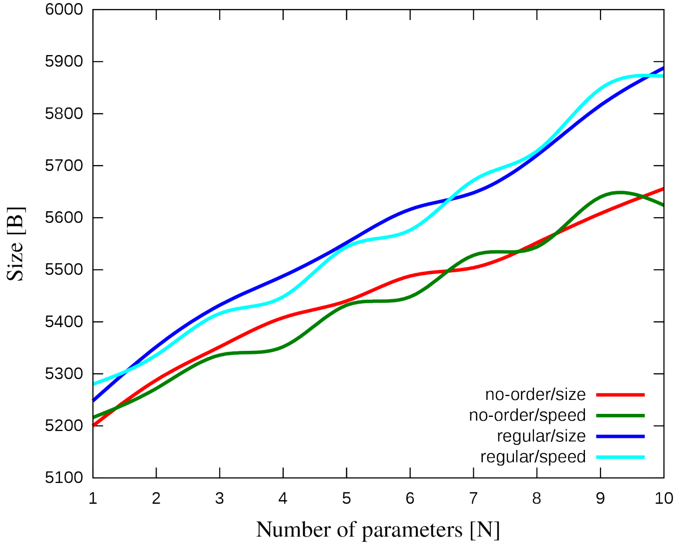
Compilation time - Clang



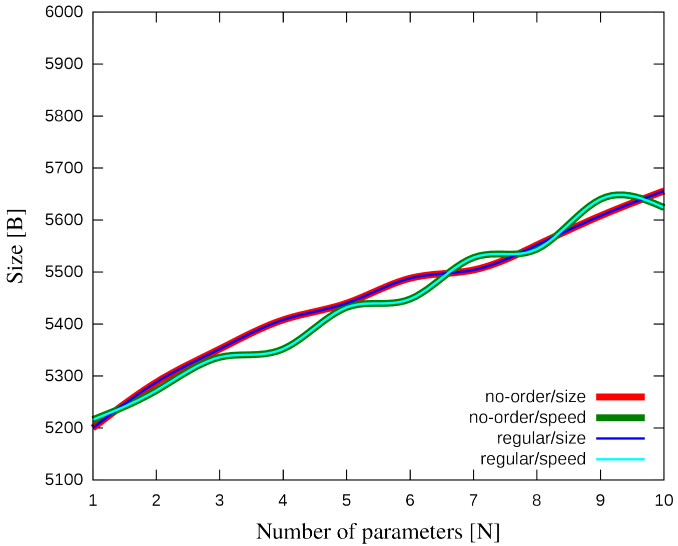
Binary size

http://2.bp.blogspot.com/-VGWIQzQ_-A4/UBR8u3aMnXI/AAAAAAAAAqY/5PqZWFTkDxc/s1600/overloaded-funny-pics-02.jpg

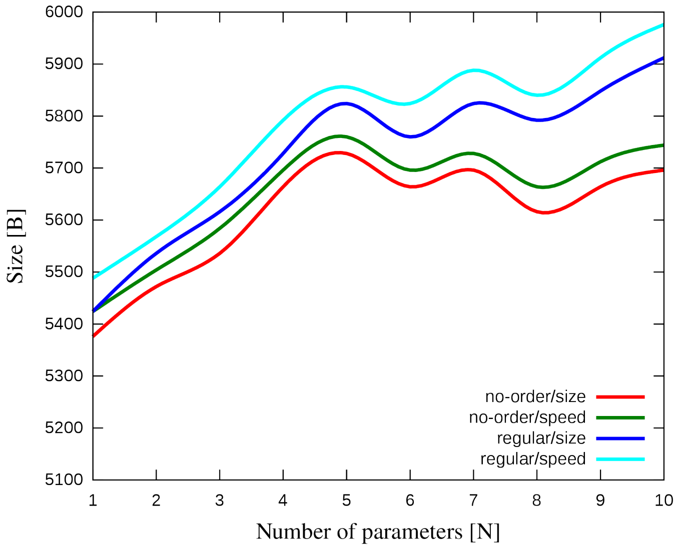
Binary size - GCC



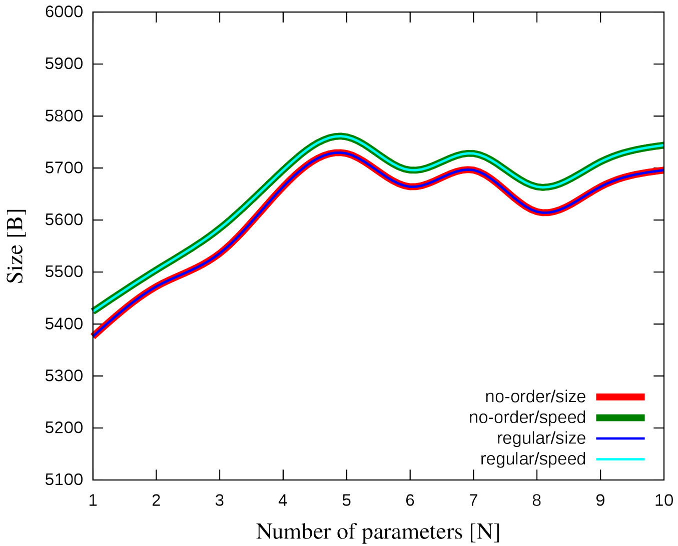
Binary size (inline version) - GCC



Binary size - Clang



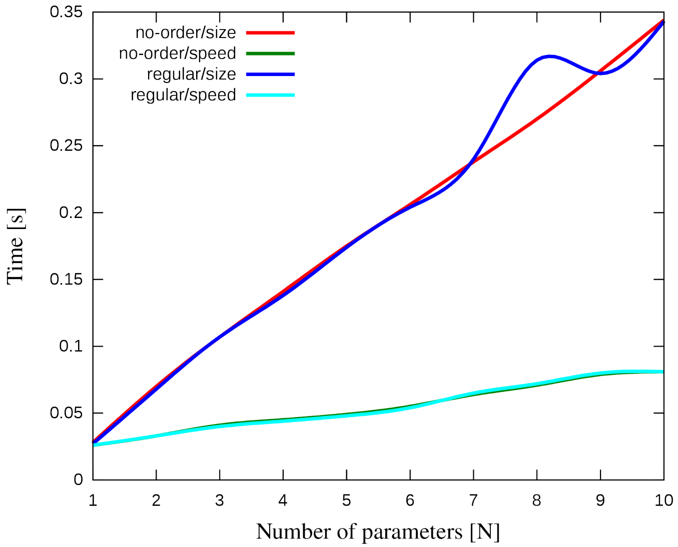
Binary size (inline version) - Clang



Run time

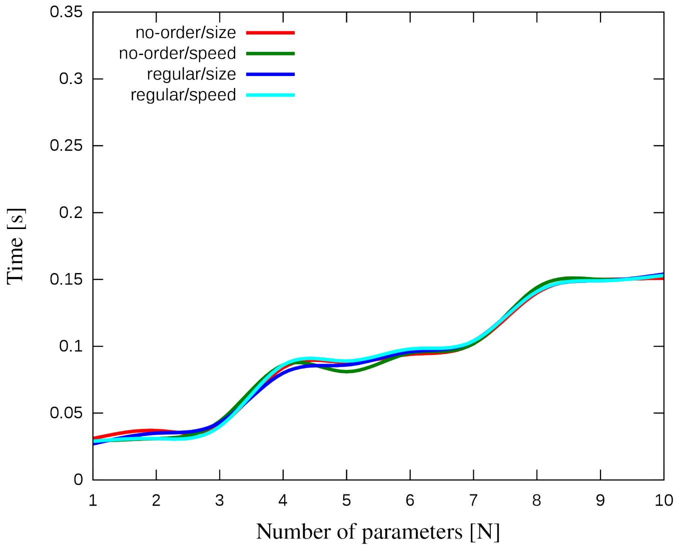
http://pic5.nipic.com/20100114/3507523_174247075692_2.jpg

Run time - GCC



Time [s]
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
1
2
3
4
5
6
7
8
9
10
Number of parameters [N]
no-order/size
no-order/speed
regular/size
regular/speed

Run time - Clang



Part 7

Basic version ready...

https://upload.wikimedia.org/wikipedia/commons/4/45/1973_Fiat_126_IMG_7855.jpg

...let's do some upgrading!

https://upload.wikimedia.org/wikipedia/commons/6/61/Cuba%2C_Havana%2C_FIAT_126p_Polski.jpg

Optional fields

- Current solution:
  require default c-tors!
  - Extra overhead
  - Often limiting
  - Looks bad. . .
- What if. . .

Optional fields

- Current solution: require default c-tors!
  - Extra overhead
  - Often limiting
  - Looks bad. . .
- What if. . .

- Solution: `boost::optional<>`
  - Setting only used fields
  - Default-constructable
  - Explicit intention (unset vs. default)
  - Nicely printable (*any*)
- Changed `Context`

Optional fields

- Current solution: require default c-tors!
  - Extra overhead
  - Often limiting
  - Looks bad. . .
- What if. . .

```
struct ContextMkII
{
  boost::optional<Type1> t1_;
  boost::optional<Type2> t2_;
  // ...
};
```

- Solution: `boost::optional<>`
  - Setting only used fields
  - Default-constructable
  - Explicit intention (unset vs. default)
  - Nicely printable (*any*)
- Changed `Context`

Custom return type

- Current solution:
  return `Context`
  - Must obey `Context`'s rules
  - All fields always set
- What if. . .

Custom return type

- Current solution: return `Context`
  - Must obey `Context`'s rules
  - All fields always set
- What if. . .

- Solution: return custom type
  - Setting only used fields
  - Comparable with `Context`
  - Nicely printable
- Context of optionals

Custom return type

- Current solution: return `Context`
  - Must obey `Context`'s rules
  - All fields always set
- What if...

- Solution: return custom type
  - Setting only used fields
  - Comparable with `Context`
  - Nicely printable
- Context of optionals

```
struct OptionalContext
{
  boost::optional<Type1> t1_;
  boost::optional<Type2> t2_;
  // ...
};
template<typename ...Args>
OptionalContext makeContext(Args&&... args);

bool operator==(Context const&, OptionalContext const&);
bool operator==(OptionalContext const&, Context const&);
```

Exploit language features

- Current solution:
  tones of templates
  - Template error messages
  - Non-trivial implementation
- What if. . .

Exploit language features

- Current solution:
  tones of templates
  - Template error messages
  - Non-trivial implementation
- What if. . .

- Solution: use C++ features
  - Setting fields by name
  - Direct language support
  - Mixes with other extensions

Exploit language features

- Current solution:
  tones of templates
  - Template error messages
  - Non-trivial implementation
- What if...

```cpp
struct OptionalContext
{
  boost::optional<Type1> field1_;
  boost::optional<Type2> field2_;
  boost::optional<Type3> field3_;
  // ...
};

OptionalContext oc{ .field1_ = Type1{/*...*/},
                    .field2_ = Type2{/*...*/}  };
```

- Solution: use C++ features
  - Setting fields by name
  - Direct language support
  - Mixes with other extensions

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:
  - GCC:
    - Only field-by-field
    - Only in-order

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:
  - GCC:
    - Only field-by-field
    - Only in-order

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:
  - GCC:
    - Only field-by-field
    - Only in-order
  - MSVC – not implemented

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:
  - GCC:
    - Only field-by-field
    - Only in-order
  - MSVC – not implemented
  - Clang – works!

Exploit language features - "yes, but" part

- Features:
  - Nice error messages
  - No extra code
  - Types can repeat
- Would be nice. . .
- Uncommon knowledge
- Poorly supported:
  - GCC:
    - Only field-by-field
    - Only in-order
  - MSVC – not implemented
  - Clang – works!
- Other drawbacks:
  - Bit more verbose
  - Gives test type explicitly

http://dailynewsdig.com/wp-content/uploads/2013/06/20-Funny-Shocked-Cat-Memes-6.jpg

Other ideas

- Policies:
  - Concept’s generalization
  - Separation type-specific parts
  - Providing as a library

Other ideas

- Policies:
  - Concept's generalization
  - Separation type-specific parts
  - Providing as a library
- C++14's tuples:
  - Addressed by type: `std::get<Type1>(tup)`
  - Replacement for `extract<T>::from()`
  - Standard-compliant

http://upload.wikimedia.org/wikipedia/commons/9/9b/Licenciado_Gustavo_D%C3%ADaz_Ordaz_International_Airport_%282014%29_-_03.JPG

Part 8

Summary

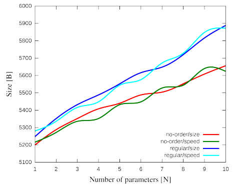
- Problem:
  - Constructing complex type
  - Interesting fields for tests
  - Simple to use

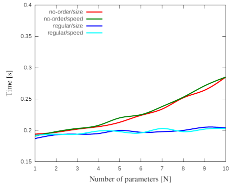
Summary

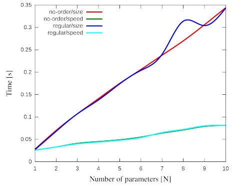
- Problem:
  - Constructing complex type
  - Interesting fields for tests
  - Simple to use
- Proposed::Solution<>:
  - Helper function
  - Extraction by (distinct!) types
  - Arguments order does not matter
  - Elastic arity
  - Sanity checks

Summary

- Problem:
  - Constructing complex type
  - Interesting fields for tests
  - Simple to use
- Proposed::Solution<>:
  - Helper function
  - Extraction by (distinct!) types
  - Arguments order does not matter
  - Elastic arity
  - Sanity checks
- Measurements:
  - No code bloat
  - Slightly longer compile times
  - Identical generated code
  - No runtime overhead







Knowledge - use with care!

+ Powerful tool
+ Arguments can be reordered. . .

Knowledge - use with care!

- + Powerful tool
- + Arguments can be reordered. . .

- . . . NOT a golden hammer!
- Special purpose tool

http://fc01.deviantart.net/fs71/i/2011/327/4/7/deep_thought_by_cryptisc-d4h18ya.png

Questions?

http://upload.wikimedia.org/wikipedia/commons/8/82/Stora_tv%C3%A4rv%C3%A4gen_-_Ystad_11sep2013.jpg