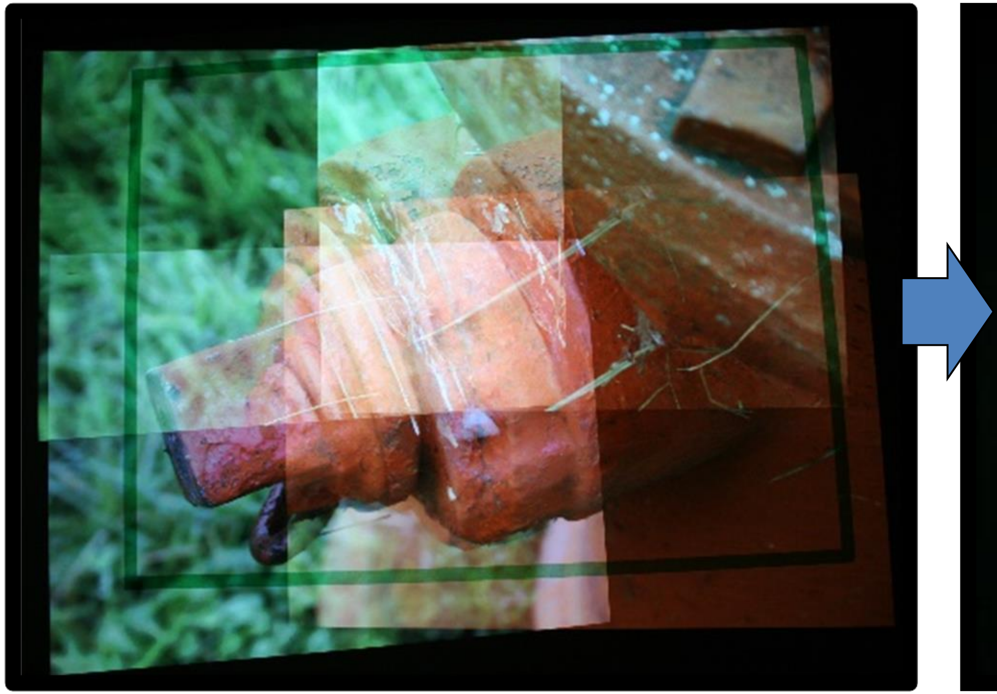
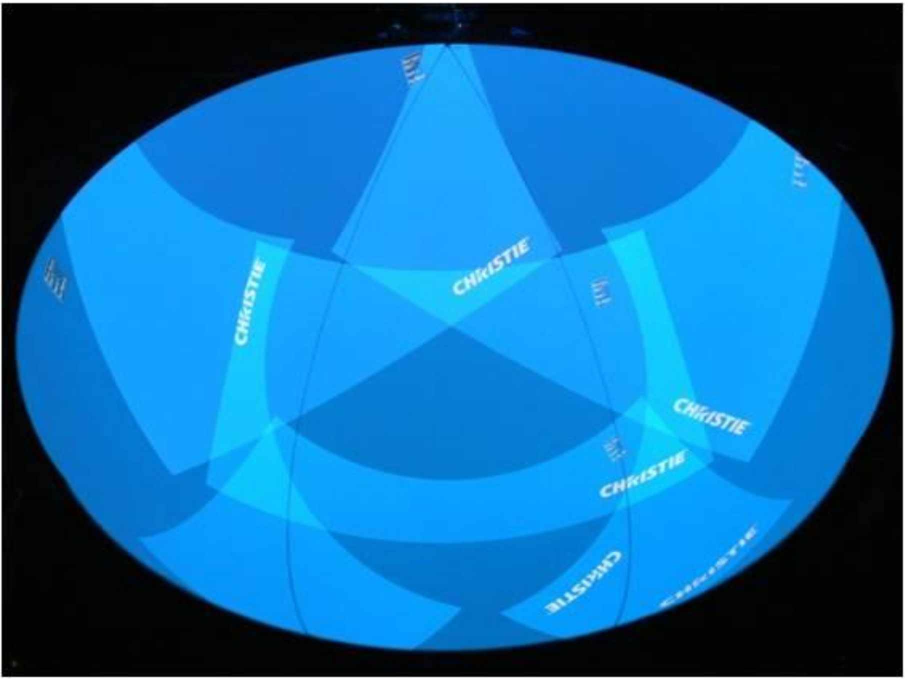
Words of Wisdom

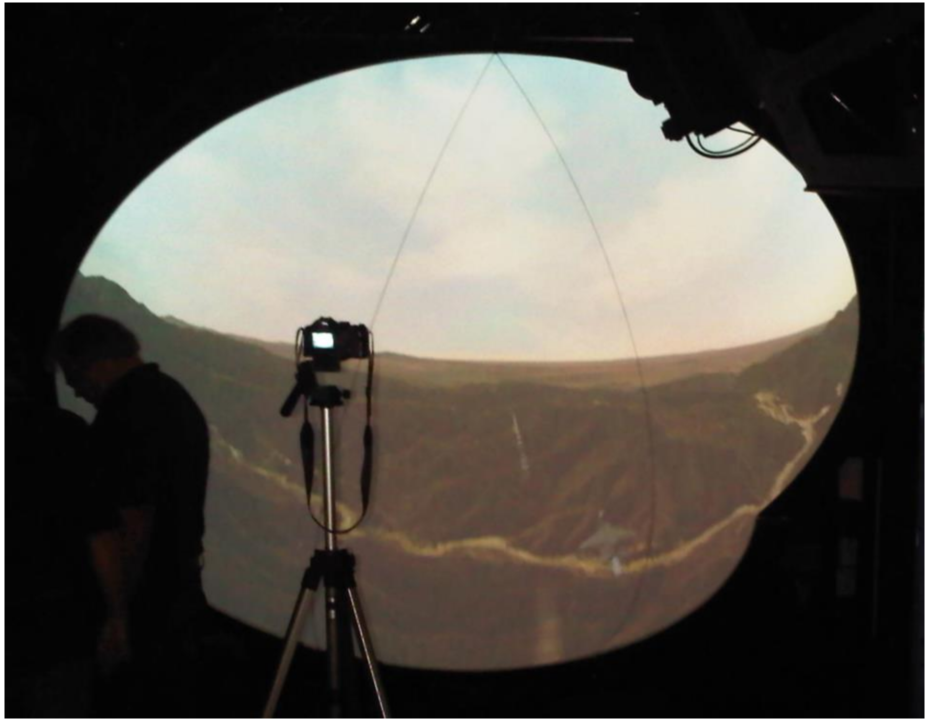
Tony Van Eerd

C++Now May 2018

CHRISTIE®



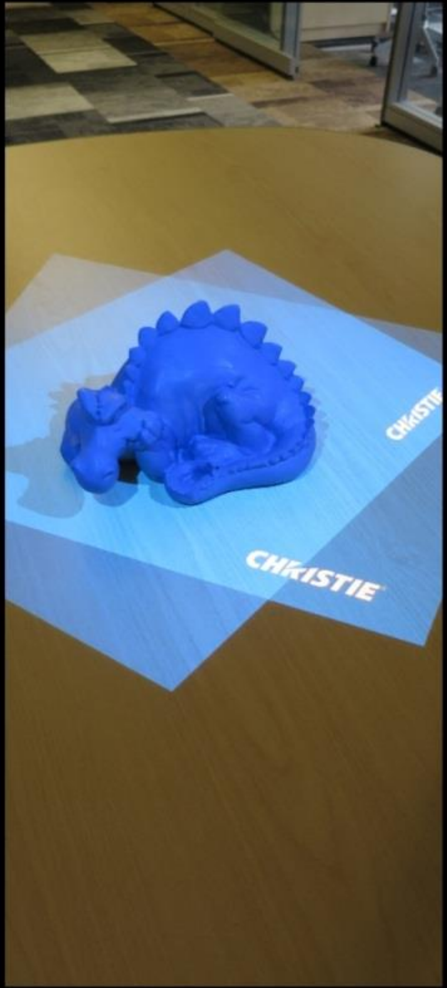

CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE
CHRISTIE

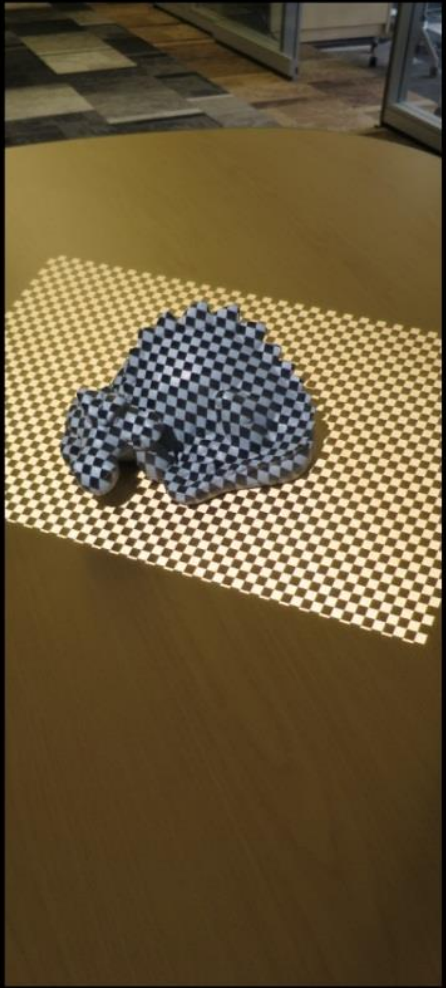


KISTIE

15 CENTER
CHRISTIE
CHRISTIE CENTER
CHRISTIE
Christie Mystique
Spectacular experiences. Simple solutions.
Design
Install
Operate
CHRISTIE
BOEHME
KLASSEN

youtu.be/sql1kD6_Uok


CHRISTIE
CHRISTIE

easy


Words of Wisdom

Tony Van Eerd

C++Now May 2018

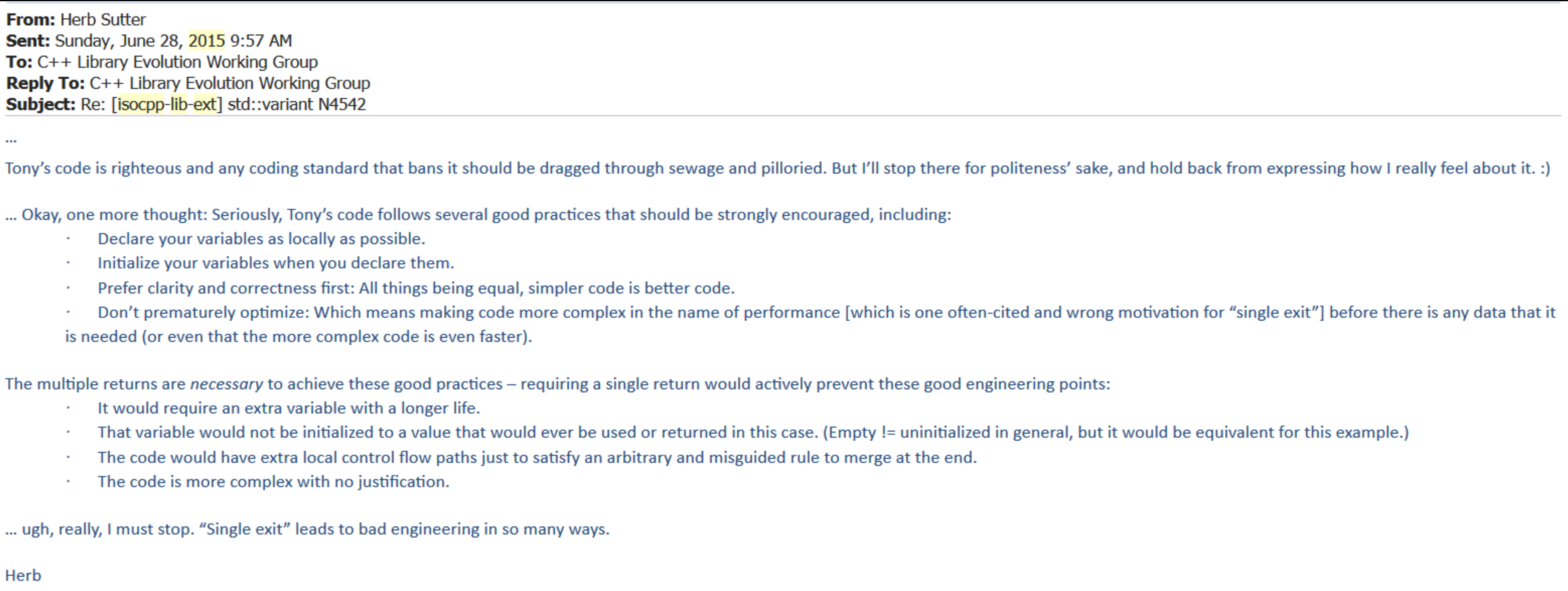
Credentials?

Credentials

"Tony's code is righteous"

- Herb Sutter

Credentials?



From: Herb Sutter
Sent: Sunday, June 28, 2015 9:57 AM
To: C++ Library Evolution Working Group
Reply To: C++ Library Evolution Working Group
Subject: Re: [isocpp-lib-ext] std::variant N4542

…

Tony's code is righteous and any coding standard that bans it should be dragged through sewage and pilloried. But I'll stop there for politeness' sake, and hold back from expressing how I really feel about it. :)

… Okay, one more thought: Seriously, Tony's code follows several good practices that should be strongly encouraged, including:

- Declare your variables as locally as possible.
- Initialize your variables when you declare them.
- Prefer clarity and correctness first: All things being equal, simpler code is better code.
- Don't prematurely optimize: Which means making code more complex in the name of performance [which is one often-cited and wrong motivation for "single exit"] before there is any data that it is needed (or even that the more complex code is even faster).

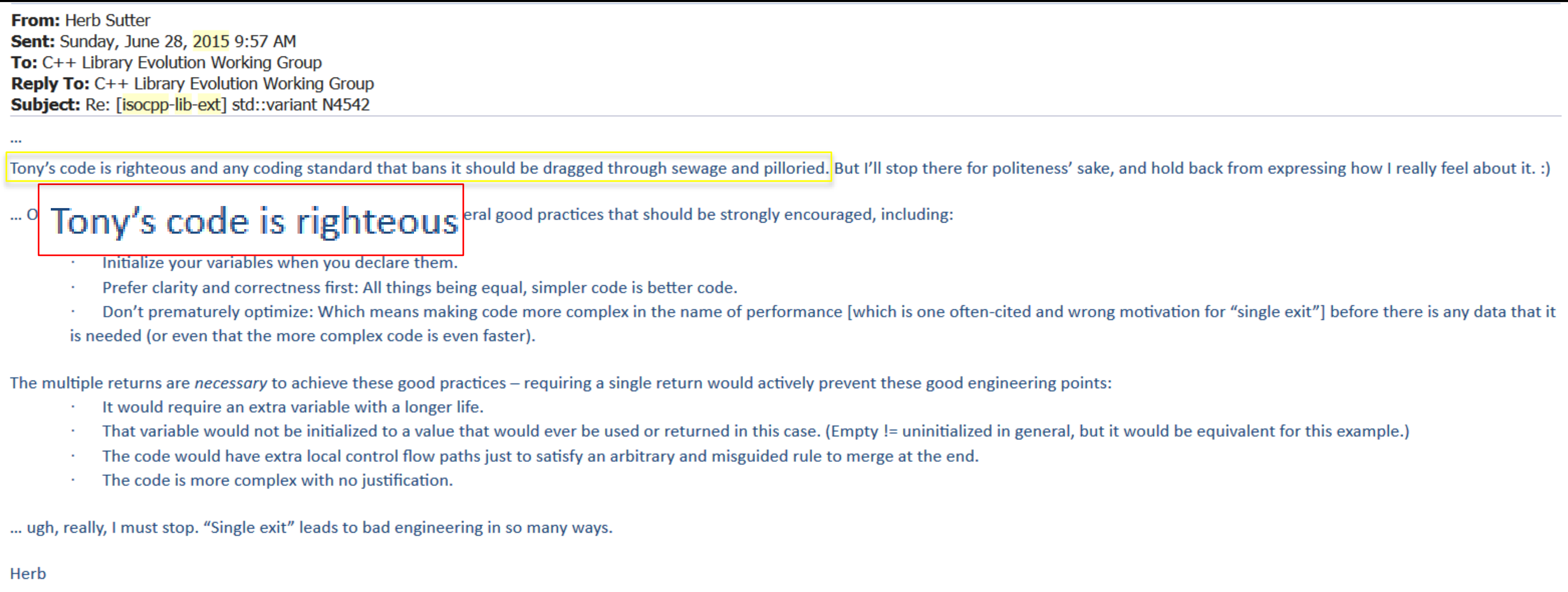
The multiple returns are *necessary* to achieve these good practices – requiring a single return would actively prevent these good engineering points:

- It would require an extra variable with a longer life.
- That variable would not be initialized to a value that would ever be used or returned in this case. (Empty != uninitialized in general, but it would be equivalent for this example.)
- The code would have extra local control flow paths just to satisfy an arbitrary and misguided rule to merge at the end.
- The code is more complex with no justification.

… ugh, really, I must stop. "Single exit" leads to bad engineering in so many ways.

Herb

Credentials?



From: Herb Sutter
Sent: Sunday, June 28, 2015 9:57 AM
To: C++ Library Evolution Working Group
Reply To: C++ Library Evolution Working Group
Subject: Re: [isocpp-lib-ext] std::variant N4542

…

Tony's code is righteous and any coding standard that bans it should be dragged through sewage and pilloried. But I'll stop there for politeness' sake, and hold back from expressing how I really feel about it. :)

Tony's code is righteous

… O eral good practices that should be strongly encouraged, including:

- Initialize your variables when you declare them.
- Prefer clarity and correctness first: All things being equal, simpler code is better code.
- Don't prematurely optimize: Which means making code more complex in the name of performance [which is one often-cited and wrong motivation for "single exit"] before there is any data that it is needed (or even that the more complex code is even faster).

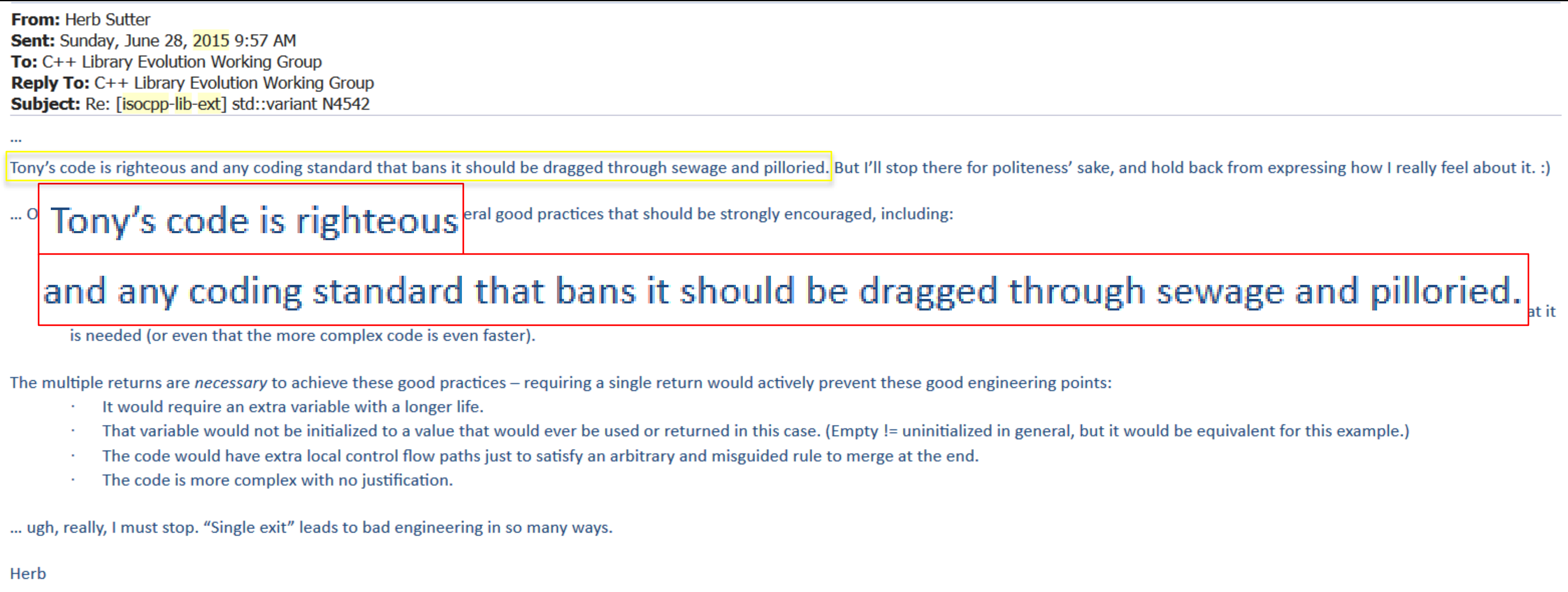
The multiple returns are *necessary* to achieve these good practices – requiring a single return would actively prevent these good engineering points:

- It would require an extra variable with a longer life.
- That variable would not be initialized to a value that would ever be used or returned in this case. (Empty != uninitialized in general, but it would be equivalent for this example.)
- The code would have extra local control flow paths just to satisfy an arbitrary and misguided rule to merge at the end.
- The code is more complex with no justification.

… ugh, really, I must stop. "Single exit" leads to bad engineering in so many ways.

Herb

Credentials?



From: Herb Sutter
Sent: Sunday, June 28, 2015 9:57 AM
To: C++ Library Evolution Working Group
Reply To: C++ Library Evolution Working Group
Subject: Re: [isocpp-lib-ext] std::variant N4542

…

Tony's code is righteous and any coding standard that bans it should be dragged through sewage and pilloried. But I'll stop there for politeness' sake, and hold back from expressing how I really feel about it. :)

Tony's code is righteous

and any coding standard that bans it should be dragged through sewage and pilloried.

… O eral good practices that should be strongly encouraged, including:

at it is needed (or even that the more complex code is even faster).

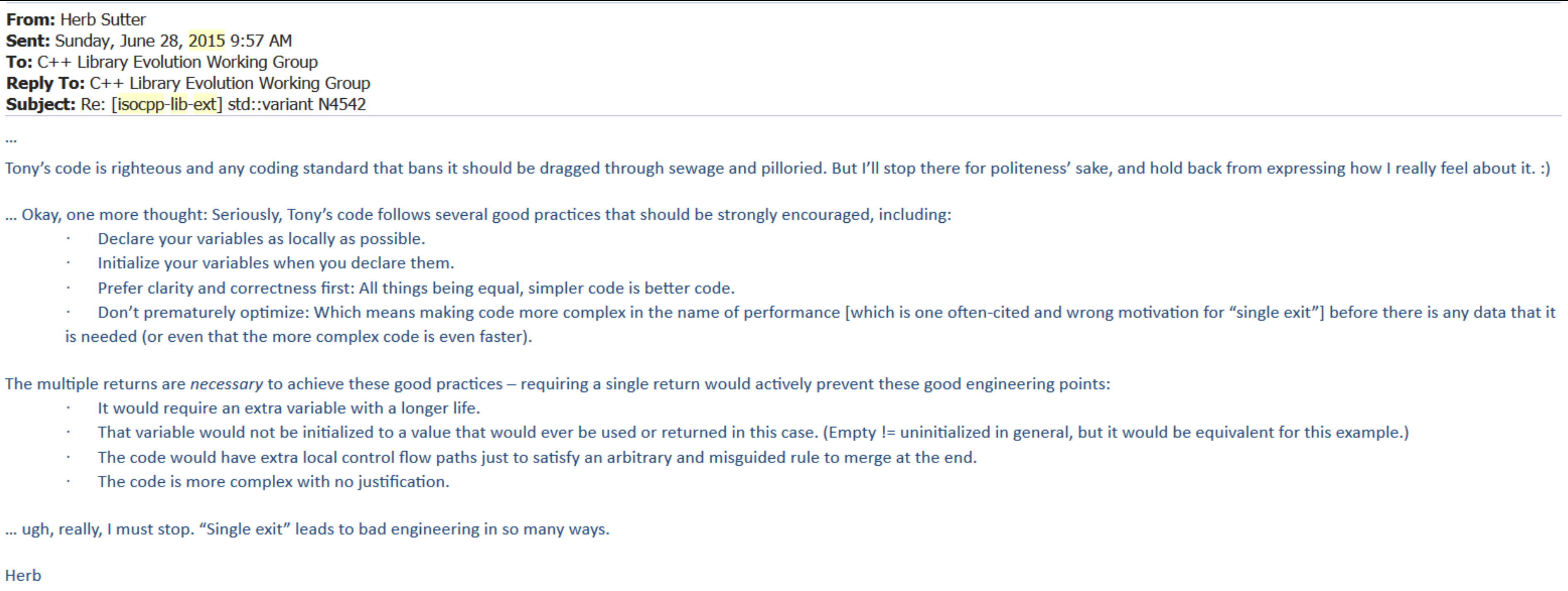
The multiple returns are *necessary* to achieve these good practices – requiring a single return would actively prevent these good engineering points:

- It would require an extra variable with a longer life.
- That variable would not be initialized to a value that would ever be used or returned in this case. (Empty != uninitialized in general, but it would be equivalent for this example.)
- The code would have extra local control flow paths just to satisfy an arbitrary and misguided rule to merge at the end.
- The code is more complex with no justification.

… ugh, really, I must stop. "Single exit" leads to bad engineering in so many ways.

Herb

Credentials?



From: Herb Sutter
Sent: Sunday, June 28, 2015 9:57 AM
To: C++ Library Evolution Working Group
Reply To: C++ Library Evolution Working Group
Subject: Re: [isocpp-lib-ext] std::variant N4542

…

Tony's code is righteous and any coding standard that bans it should be dragged through sewage and pilloried. But I'll stop there for politeness' sake, and hold back from expressing how I really feel about it. :)

… Okay, one more thought: Seriously, Tony's code follows several good practices that should be strongly encouraged, including:

- Declare your variables as locally as possible.
- Initialize your variables when you declare them.
- Prefer clarity and correctness first: All things being equal, simpler code is better code.
- Don't prematurely optimize: Which means making code more complex in the name of performance [which is one often-cited and wrong motivation for "single exit"] before there is any data that it is needed (or even that the more complex code is even faster).

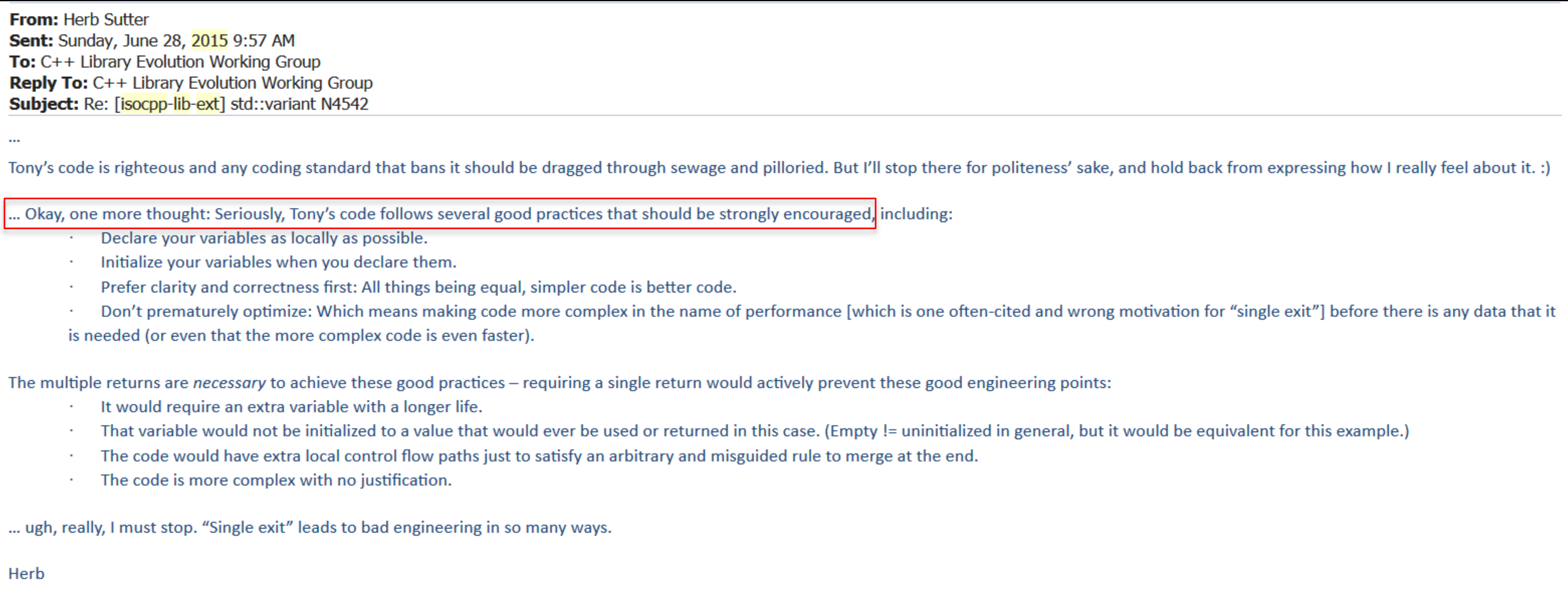
The multiple returns are *necessary* to achieve these good practices – requiring a single return would actively prevent these good engineering points:

- It would require an extra variable with a longer life.
- That variable would not be initialized to a value that would ever be used or returned in this case. (Empty != uninitialized in general, but it would be equivalent for this example.)
- The code would have extra local control flow paths just to satisfy an arbitrary and misguided rule to merge at the end.
- The code is more complex with no justification.

… ugh, really, I must stop. "Single exit" leads to bad engineering in so many ways.

Herb

Credentials?



From: Herb Sutter
Sent: Sunday, June 28, 2015 9:57 AM
To: C++ Library Evolution Working Group
Reply To: C++ Library Evolution Working Group
Subject: Re: [isocpp-lib-ext] std::variant N4542

…

Tony's code is righteous and any coding standard that bans it should be dragged through sewage and pilloried. But I'll stop there for politeness' sake, and hold back from expressing how I really feel about it. :)

… Okay, one more thought: Seriously, Tony's code follows several good practices that should be strongly encouraged, including:

- Declare your variables as locally as possible.
- Initialize your variables when you declare them.
- Prefer clarity and correctness first: All things being equal, simpler code is better code.
- Don't prematurely optimize: Which means making code more complex in the name of performance [which is one often-cited and wrong motivation for "single exit"] before there is any data that it is needed (or even that the more complex code is even faster).

The multiple returns are *necessary* to achieve these good practices – requiring a single return would actively prevent these good engineering points:

- It would require an extra variable with a longer life.
- That variable would not be initialized to a value that would ever be used or returned in this case. (Empty != uninitialized in general, but it would be equivalent for this example.)
- The code would have extra local control flow paths just to satisfy an arbitrary and misguided rule to merge at the end.
- The code is more complex with no justification.

… ugh, really, I must stop. "Single exit" leads to bad engineering in so many ways.

Herb

CHRISTIE®

Credentials

"Tony's code is righteous"

- Herb Sutter

CHRISTIE®

Credentials

“Tony’s code is righteous”

- Herb Sutter

Credentials

“Tony’s code is righteous”

- Herb Sutter

Credentials

"+1"

- Bjarne Stroustrup

Credentials

“Nice rule of thumb”

- Bjarne Stroustrup

CHRISTIE®

#ClashOfTheClichés

“Better safe than Sorry”

“A ship in harbour is safe, but that’s not what ships are built for”

Copy, or copy not;
there is no shallow.

- Master Yoda

Copy, or copy not; there is no shallow.

- Master Yoda

```
T a = b;   assert(a == b);

T a;  a = b;  // same as above

T a = c;
T b = c;
change(a);
assert(b == c && a != b);
```

- Master Stepanov

(http://stepanovpapers.com/DeSt98.pdf)

State leads to objects
Objects lead to references
References lead to sharing
Sharing leads to entanglement
Entanglement leads to suffering

- Master Yoda

Object

Value

Object

- Button
- java / OOP
- *non-copyable*
- *changeable, observable, lifetime*
- Relationships

- OH NO!!! Pointers!!!

Value

- int
- string
- copyable
- *immutable, ephemerable, timeles*
- Math

- Oh, no pointers.

“If you postulate a universe composed of objects, truth is eliminated.”

- Jean-Paul Sartre

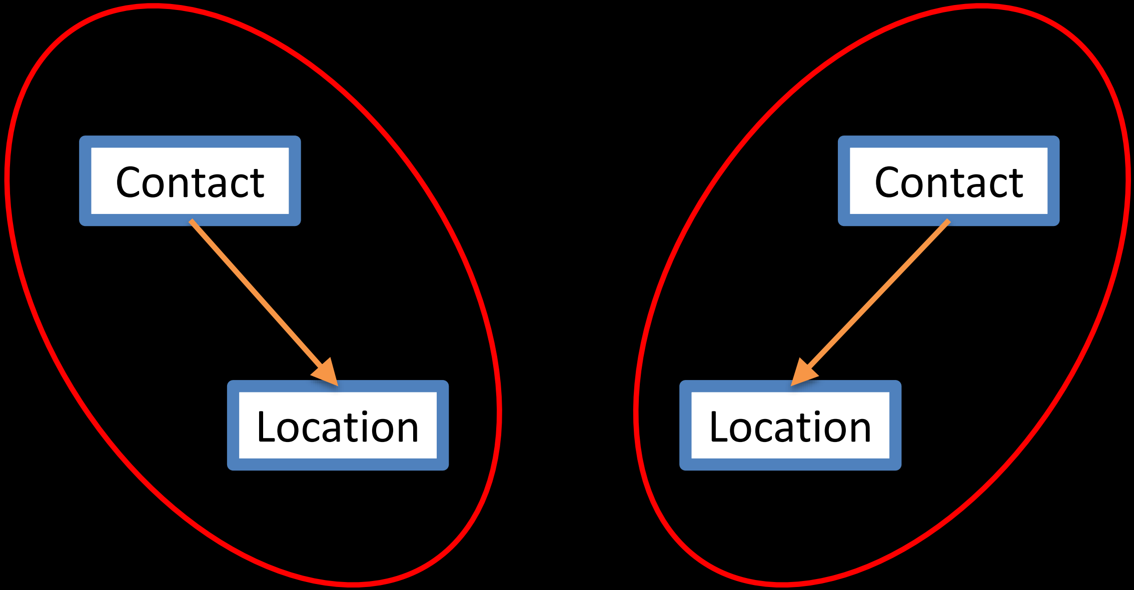
(courtesy of Ben Deane)

“Incidental Data Structures”

- Sean Parent

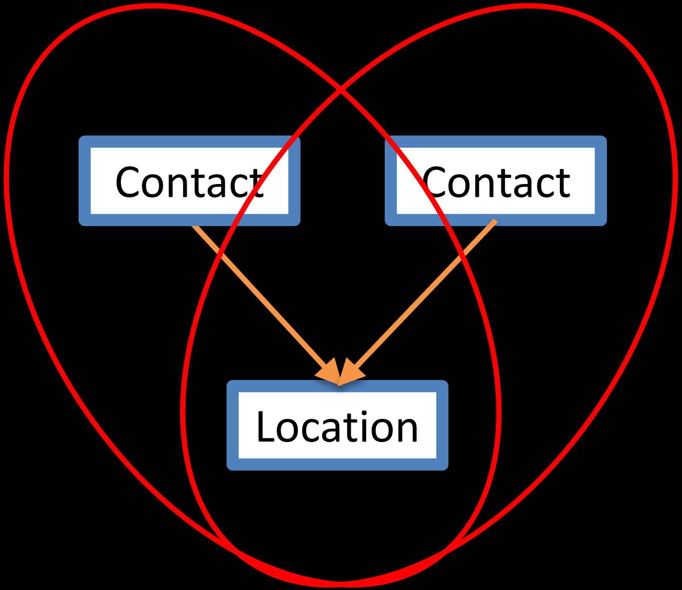
“Incidental Data Structures”

- Sean Parent



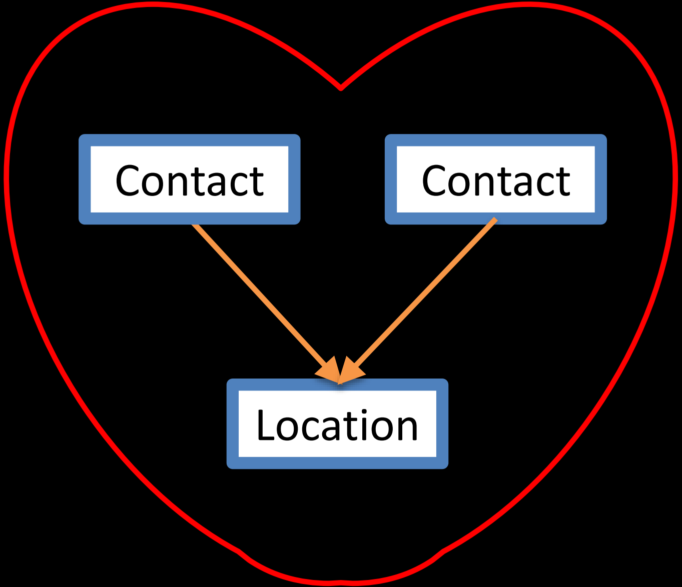
“Incidental Data Structures”

- Sean Parent



"Incidental Data Structures"

- Sean Parent



Object

- Button
- java / OOP
- *non-copyable*
- *changeable, observable, lifetime*
- Relationships

- OH NO!!! Pointers!!!

Value

- int
- string
- copyable
- *immutable, ephemerable, timeles*
- Math

- Oh, no pointers.

“A shared pointer is as good as a global variable.”

- Sean Parent

"A shared pointer is as good as a global variable."

- Sean Parent

"Neither a borrower nor a lender be"

- Shakespeare

momo

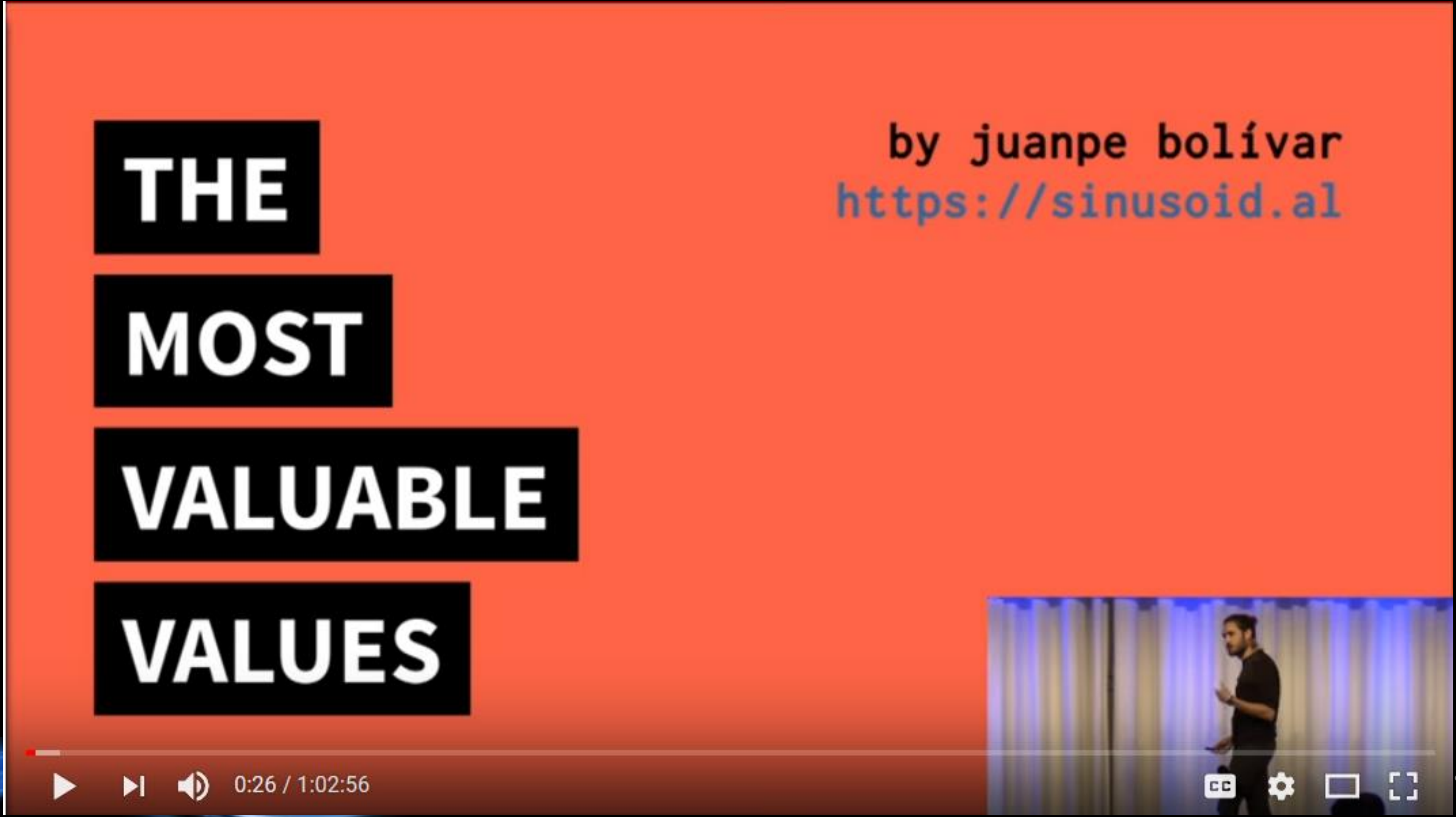
cppcon | 2017
THE C++ CONFERENCE • BELLEVUE, WASHINGTON

// juanpe bolívar
// https://sinusoid.al

POSTMODERN
IMMUTABLE
DATA STRUCTURES

JUAN PEDRO BOLIVAR PUENTE

Postmodern immutable data structures



THE

MOST

VALUABLE

VALUES

by juanpe bolívar

https://sinusoid.al

#ClashOfTheClichés

“If at first you don't succeed, try try again.”

“Don’t throw good money after bad.”

“Only a Sith deals in Absolutes”

- Master Obi-Wan Kenobi on signed vs unsigned

CHRISTIE®

```
int sqrt(unsigned int)
{
    return 17;
}
```

```
int sqrt(unsigned int)
{
    return 17;
}

void test()  // TDD
{
   assert(sqrt(289) == 17);
}
```

```
int sqrt(unsigned int)
{
    return 17;
}
```

```
int sqrt(unsigned int)
{
    return 17;
}

int x = sqrt(-23);
int sqrt(UnsignedInt)
{
    return 17;
}

int x = sqrt(-23); // compiler error
```

```
template<typename T>
bool is_sorted(std::vector<T> const & v)
{
   for (std::size_t i = 0; i < v.size() -1; i++)
       if (v[i+1] < v[i])
           return false;
   return true;
}
```

Going Native 2013
Ask us Anything
https://www.youtube.com/watch?v=Puio5dly9N8

9:50
41:08
1:02:50

“Sorry”

- Stephan T. Lavavej, Andrei Alexandrescu, Bjarne Stroustrup, Scott Meyers, Michael Wong, Sean Parent, Chandler Carruth, and Herb Sutter, on unsigned in the STL

CHRISTIE®

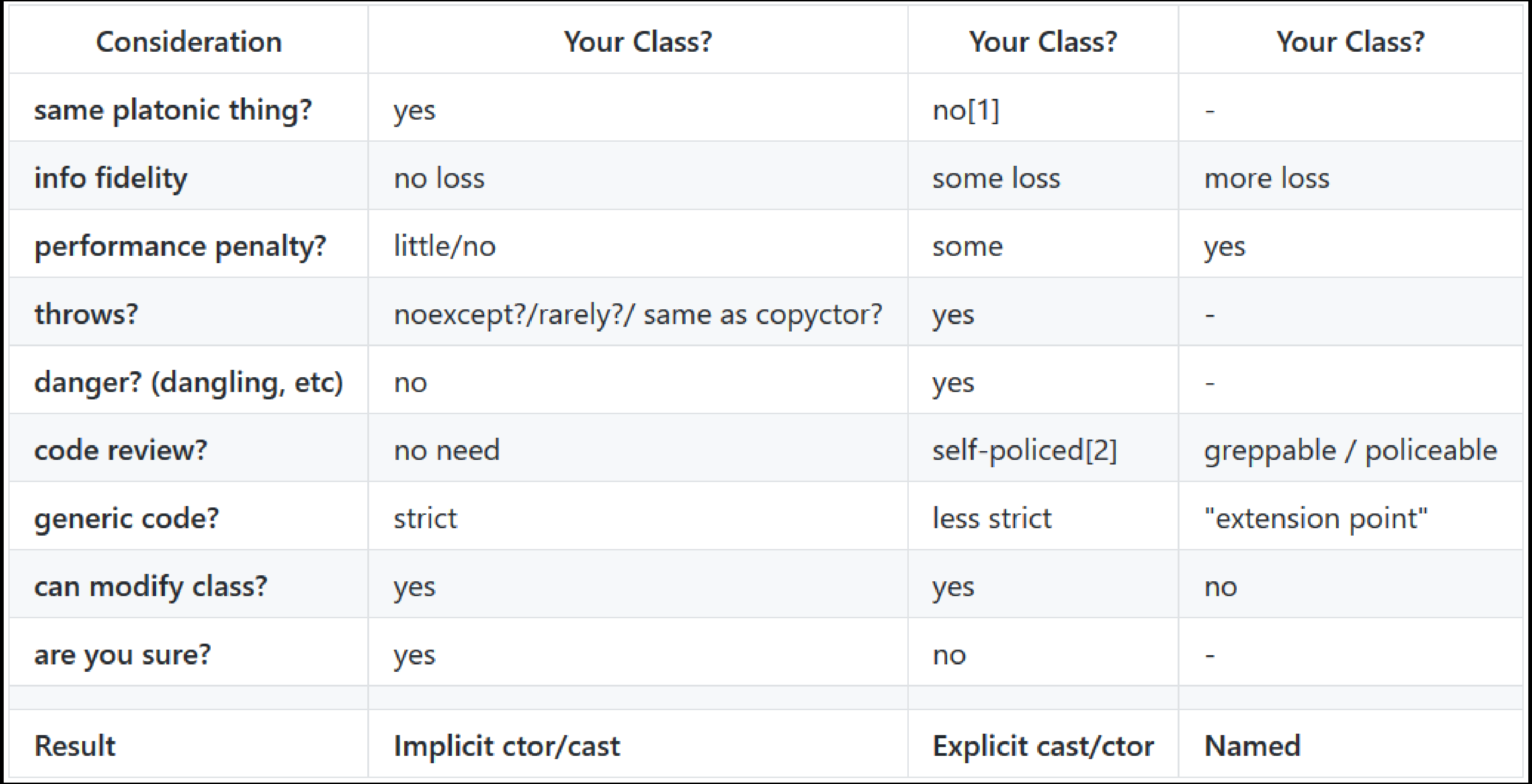
```
Projector * getProjector(string projectorId);
Projector * getProjector(string projectorId);

Camera * getCamera(string cameraId);
```

CHRISTIE®

```
using ProjectorId = StrongId<string, ProjectorIdTag>;
Projector * getProjector(ProjectorId id);

using CameraId = StrongId<string, CameraIdTag>;
Camera * getCamera(CameraId id);
```

```
using ProjectorId = StrongId<string, ProjectorIdTag>;
Projector * getProjector(ProjectorId id);

using CameraId = StrongId<string, CameraIdTag>;
Camera * getCamera(CameraId id);

auto cam = getCamera(projId);        // compiler error
auto proj = getProjector(“123-45”s); // compiler error
```

- explicit



Consideration	Your Class?	Your Class?	Your Class?
same platonic thing?	yes	no[1]	-
info fidelity	no loss	some loss	more loss
performance penalty?	little/no	some	yes
throws?	noexcept?/rarely?/ same as copyctor?	yes	-
danger? (dangling, etc)	no	yes	-
code review?	no need	self-policed[2]	greppable / policeable
generic code?	strict	less strict	"extension point"
can modify class?	yes	yes	no
are you sure?	yes	no	-
Result	**Implicit ctor/cast**	**Explicit cast/ctor**	**Named**

https://github.com/tvaneerd/isocpp/blob/master/conversions.md

```
struct Image {
   explicit Image(char const * filename) { }
};

int main()
{
   Image x("asdf");
   x = (Image)"other";
}
```

```
using ProjectorId = StrongId<string, ProjectorIdTag>;
Projector * getProjector(ProjectorId id);

using CameraId = StrongId<string, CameraIdTag>;
Camera * getCamera(CameraId id);
```

https://github.com/tvaneerd/code/blob/master/StrongId.h

https://foonathan.net/doc/type_safe/
https://github.com/rollbear/strong_type
https://www.fluentcpp.com/2016/12/08/strong-types-for-strong-interfaces/

CHRISTIE®

#ClashOfTheClichés

“There’s no such thing as a free lunch.”

“Don’t look a gift horse in the mouth .”

CHRISTIE®

Student: “If you pass a value it copies, but if you pass a pointer it wraps; it is smart & adapts...” Master hammered a screw into the desk.

- Ancient C++ Koan

CHRISTIE®

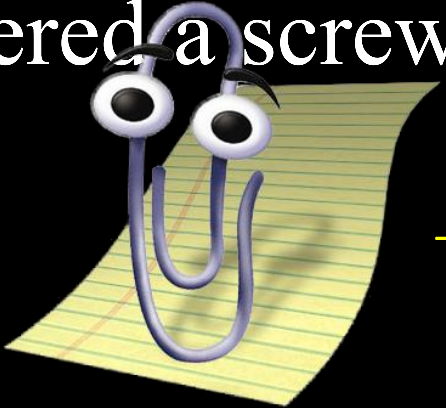
World’s best API…

Student: “If you pass a value it copies, but if you pass a pointer it wraps; it is smart & adapts...” Master hammered a screw into the desk.

- Ancient C++ Koan

CHRISTIE®

Student: “If you pass a value it copies, but if you pass a pointer it wraps; it is smart & adapts...” Master hammered a screw into the desk.



- Ancient C++ Koan

momo

← Feature Creep

#ClashOfTheClichés

"Absence makes the heart grow fonder."

"Out of sight, out of mind."

(Ben Deane)

CHRISTIE®

“What's in a name? That which we call a rose By any other word would smell as sweet;”

- Shakespeare

CHRISTIE®

“What's in a name? That which we call a rose By any other word would smell as sweet;”

- Shakespeare

Well, actually…

- Jelena Djordjevic, Johan N. Lundstrom, Francis Clément, Julie A. Boyle, Sandra Pouliot, and Marilyn Jones-Gotman
  Journal of Neurophysiology, Vol 99, No. 1, January 2008

CHRISTIE®

"Essence is the essence of naming"

- Tony Van Eerd

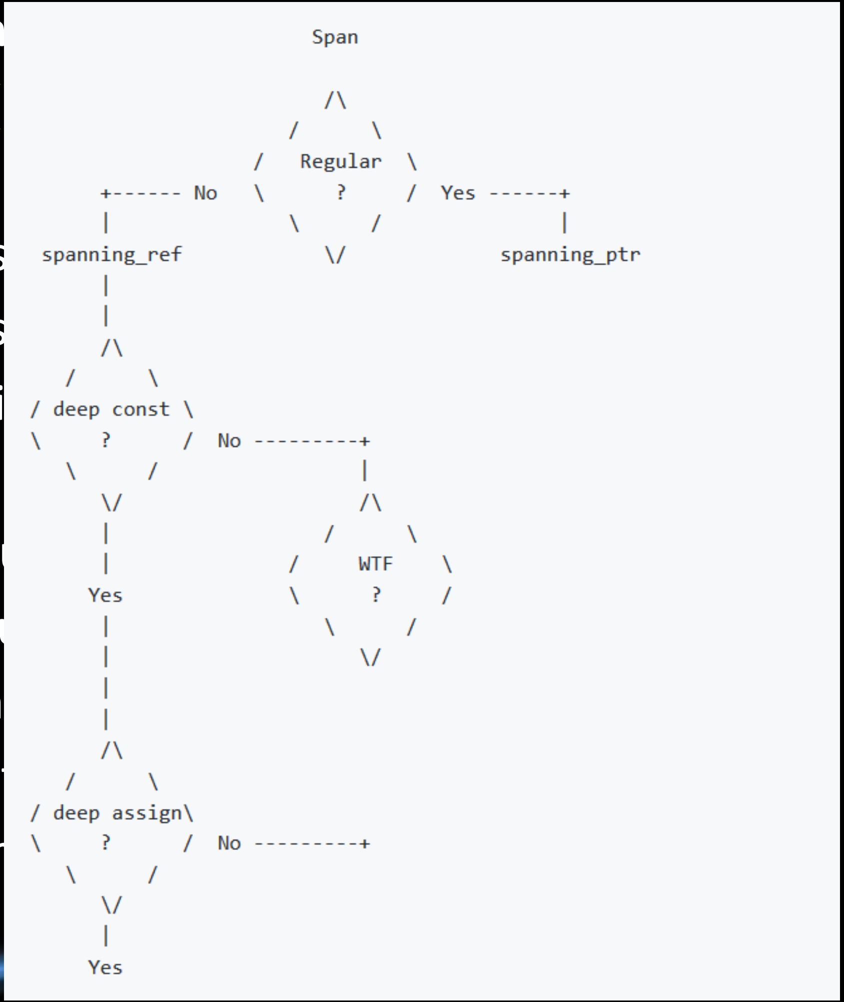
CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

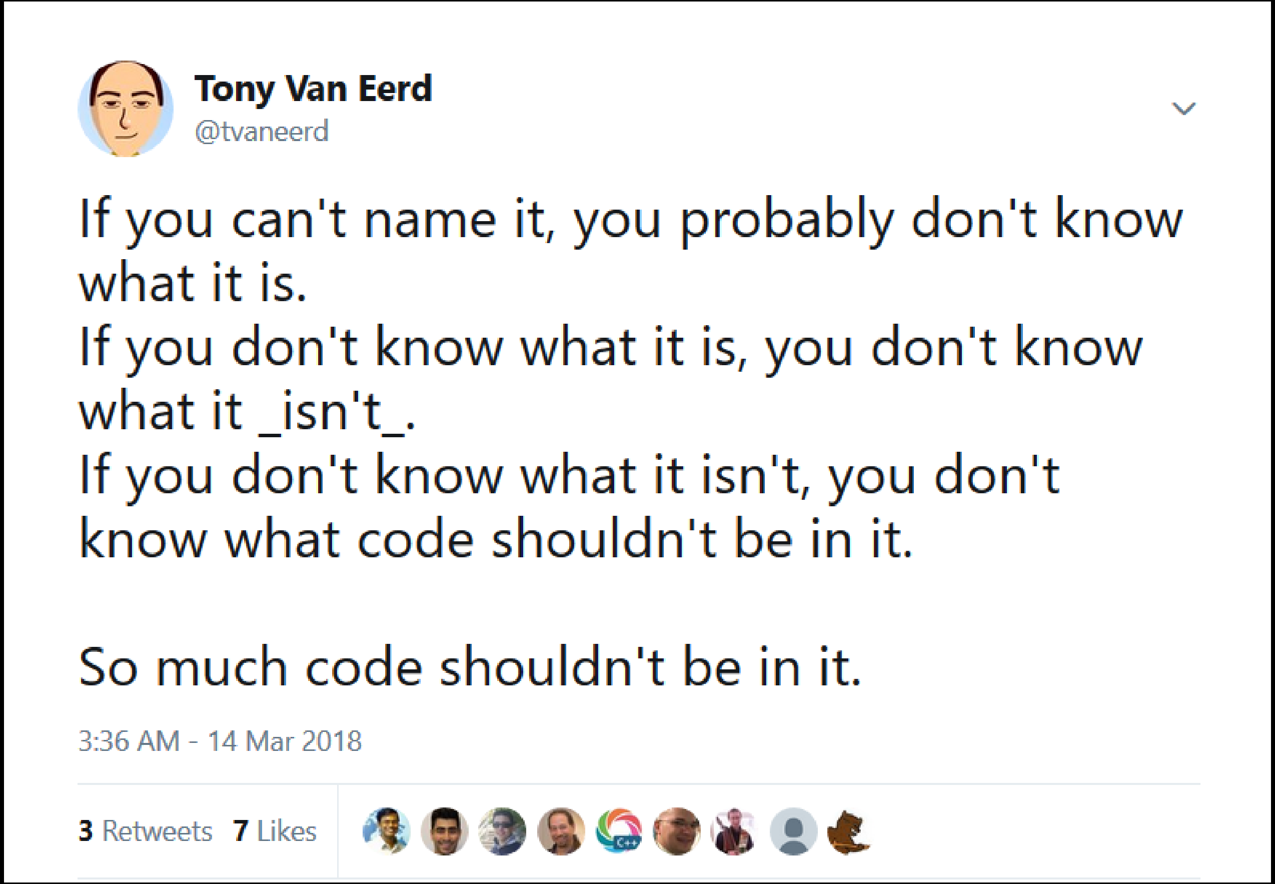
- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

- Describe the thing in
- Determine Essence (
- Be Consistent
- Consistent warning s
- Be *Glaringly* Inconsis
- NOT understanding i
- Co-opt a term?
- Avoid negatives – th
- Avoid spoken ambigu etc)
- Avoid verb/noun am
- Be Concise – concep
- By use or by function



```
                        Span

                         /\
                      /      \
                   /  Regular  \
      +------ No   \     ?     /  Yes ------+
      |               \      /              |
 spanning_ref            \/            spanning_ptr
      |
      |
      /\
   /      \
/ deep const \
\     ?      /  No ---------+
   \      /                 |
      \/                   /\
      |                 /      \
      |              /    WTF    \
     Yes             \     ?     /
      |                 \      /
      |                    \/
      |
      |
      /\
   /      \
/ deep assign\
\     ?      /  No ---------+
   \      /
      \/
      |
     Yes
```



Tony Van Eerd
@tvaneerd

If you can't name it, you probably don't know what it is.
If you don't know what it is, you don't know what it _isn't_.
If you don't know what it isn't, you don't know what code shouldn't be in it.

So much code shouldn't be in it.

3:36 AM - 14 Mar 2018

3 Retweets 7 Likes

CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

- Describe the thing in detail – what words did you use?
- Determi
- Be Cons
- Consiste
- Be *Glari*
- NOT un
- Co-opt a
- Avoid n
- Avoid sp

```
using ProjectorId = StrongId<string, ProjectorIdTag>;

ProjectorId projId{“123”};

takesString(projId); // compile error - good
takesString(static_cast<string>(projId));  //  explicit
```

- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

- Describe the thing in detail – what words did you use?
- Determi
- Be Cons
- Consiste
- Be *Glari*
- NOT un
- Co-opt a
- Avoid n
- Avoid sp
- Avoid ve
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

```
using ProjectorId = StrongId<string, ProjectorIdTag>;

ProjectorId projId{“123”};

takesString(projId); // compile error - good
takesString(static_cast<string>(projId));  //  explicit

takesString(proj.
```

- Describe the thing in detail – what words did you use?
- Determi
- Be Cons
- Consiste
- Be *Glari*
- NOT un
- Co-opt a
- Avoid n
- Avoid sp
- Avoid ve
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

```
using ProjectorId = StrongId<string, ProjectorIdTag>;

ProjectorId projId{"123"};

takesString(projId); // compile error - good
takesString(static_cast<string>(projId));  //  explicit

takesString(proj.operator string());
```

- Describe the thing in detail – what words did you use?
- Determi
- Be Cons
- Consiste
- Be *Glari*
- NOT un
- Co-opt
- Avoid n
- Avoid s
- Avoid v
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

```
using ProjectorId = StrongId<string, ProjectorIdTag>;

ProjectorId projId{“123”};

takesString(projId); // compile error - good
takesString(static_cast<string>(projId));  //  explicit

takesString(proj.functionName());
```

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

- Describe the thing in detail – what words did you use?
- Determi
- Be Cons
- Consiste
- Be *Glari*
- NOT un
- Co-opt a
- Avoid n
- Avoid s
- Avoid v
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

```
using ProjectorId = StrongId<string, ProjectorIdTag>;

ProjectorId projId{“123”};

takesString(projId); // compile error - good
takesString(static_cast<string>(projId));  //  explicit

takesString(proj.get());
```

CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than MISunderstanding
- Co-opt a term?
- Avoid negatives – thus avoiding double negatives
- Avoid spoken ambiguity (or learn to pronounce _, Capitals, etc)
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than M
- Co-opt a term?
- Avoid negatives – thus avoiding dou
- Avoid spoken ambiguity (or learn to
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid su
- By use or by functionality

```
optional<float> op;
expected<float> ex;
any             an;



      op.has_value()
      ex.has_value()
      an.has_value()
```

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than M
- Co-opt a term?
- Avoid negatives – thus avoiding dou
- Avoid spoken ambiguity (or learn to
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid su
- By use or by functionality

```
optional<float> op;
expected<float> ex;
any             an;
variant<float,int>  vr;

      op.has_value()
      ex.has_value()
      an.has_value()
```

CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding is better than M
- Co-opt a term?
- Avoid negatives – thus avoiding dou
- Avoid spoken ambiguity (or learn to
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid su
- By use or by functionality

```
optional<float> op;
expected<float> ex;
any             an;
variant<float,int>  vr;

      op.has_value()
      ex.has_value()
      an.has_value()
vr.valueless_by_exception()
```

CHRISTIE®

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consistent
- Consistent warning signs
- Be *Glaringly* Inconsistent
- NOT understanding
- Co-opt a term?
- Avoid negatives – t
- Avoid spoken ambi
- Avoid verb/noun ambiguity
- Be Concise – conceptually. Avoid sub-concepts.
- By use or by functionality

"Nice rule of thumb"

- Bjarne Stroustrup

CHRISTIE®

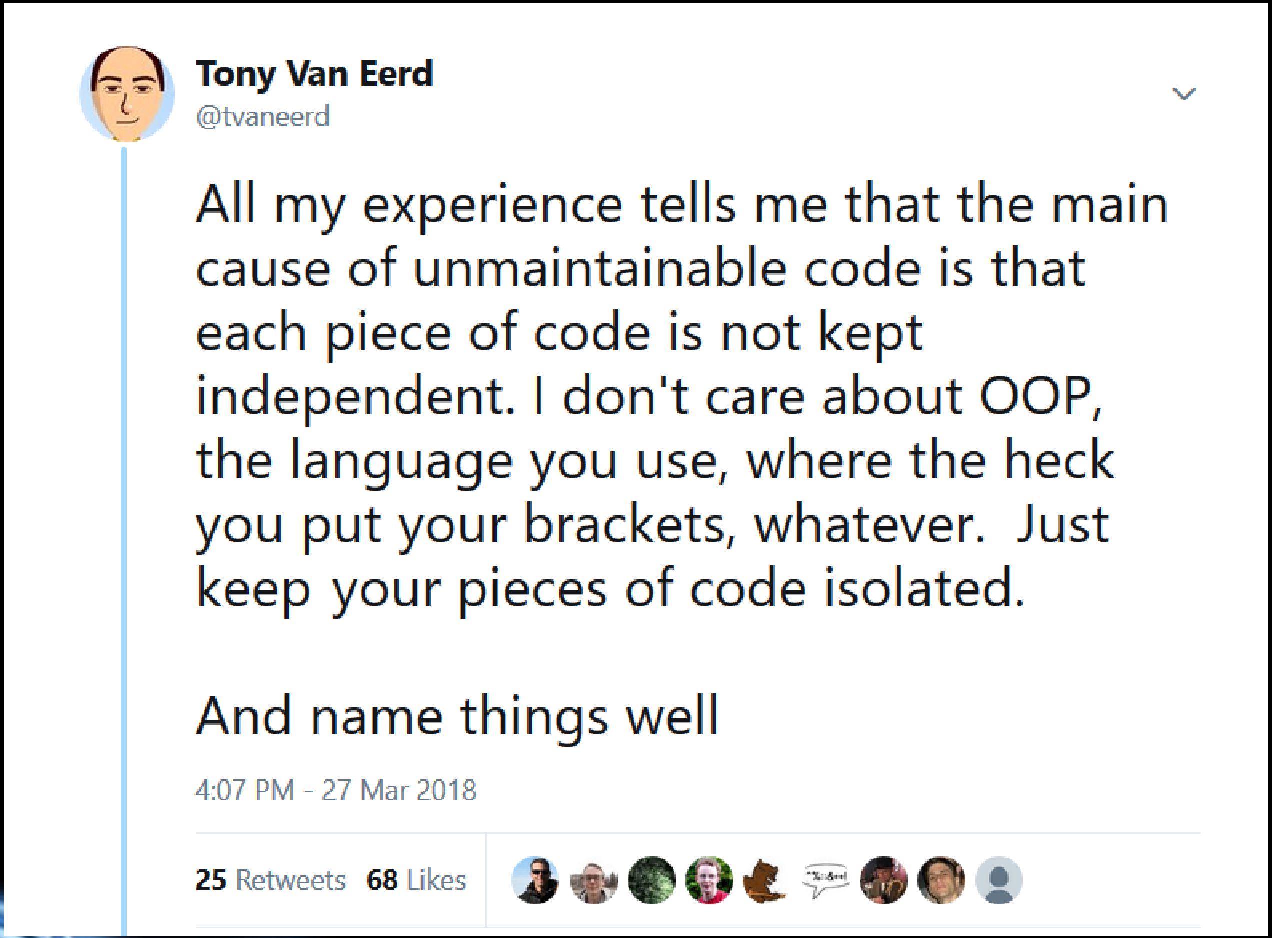
#ClashOfTheClichés

"The early bird gets the worm."
"Good things come to those who wait."
"Patience is a virtue."
"Strike while the iron is hot."

“In sooth, I know not why all code's so bad:
It wearies me; you say it wearies you;”

- Shakespeare, opening lines of *The Coder of Venice*

CHRISTIE®



Tony Van Eerd
@tvaneerd

All my experience tells me that the main cause of unmaintainable code is that each piece of code is not kept independent. I don't care about OOP, the language you use, where the heck you put your brackets, whatever. Just keep your pieces of code isolated.

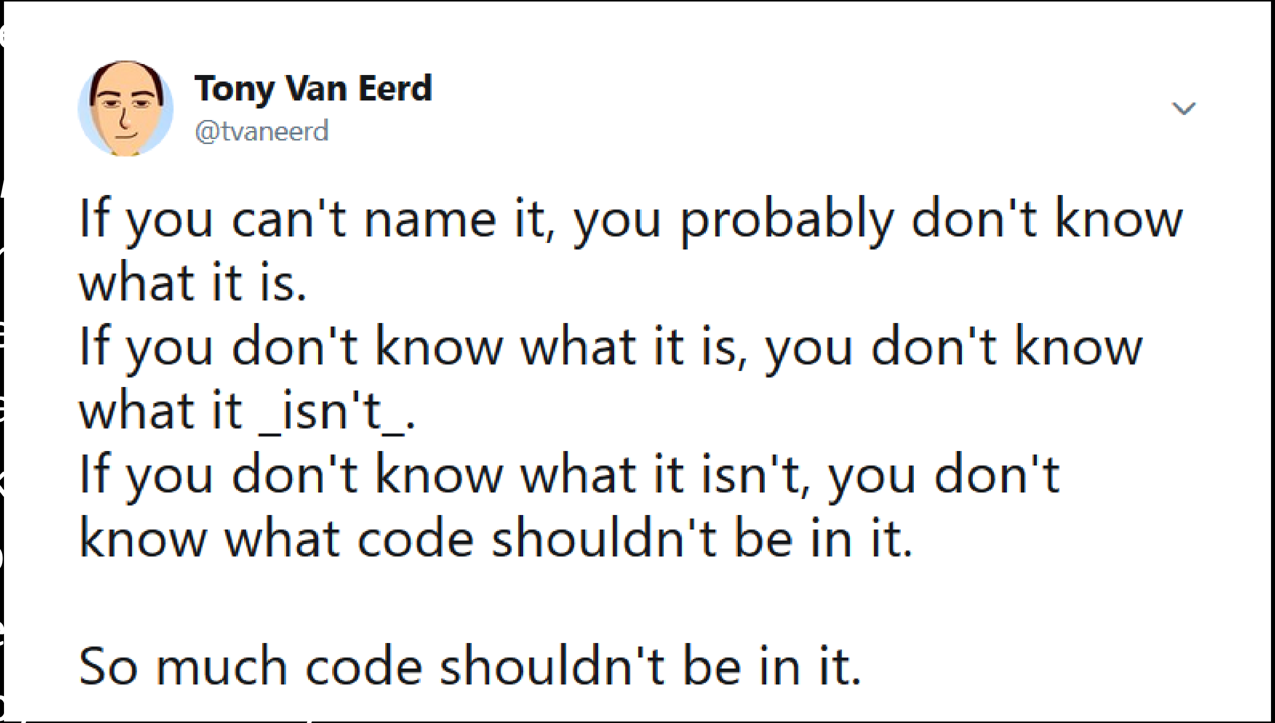
And name things well

4:07 PM - 27 Mar 2018

25 Retweets 68 Likes

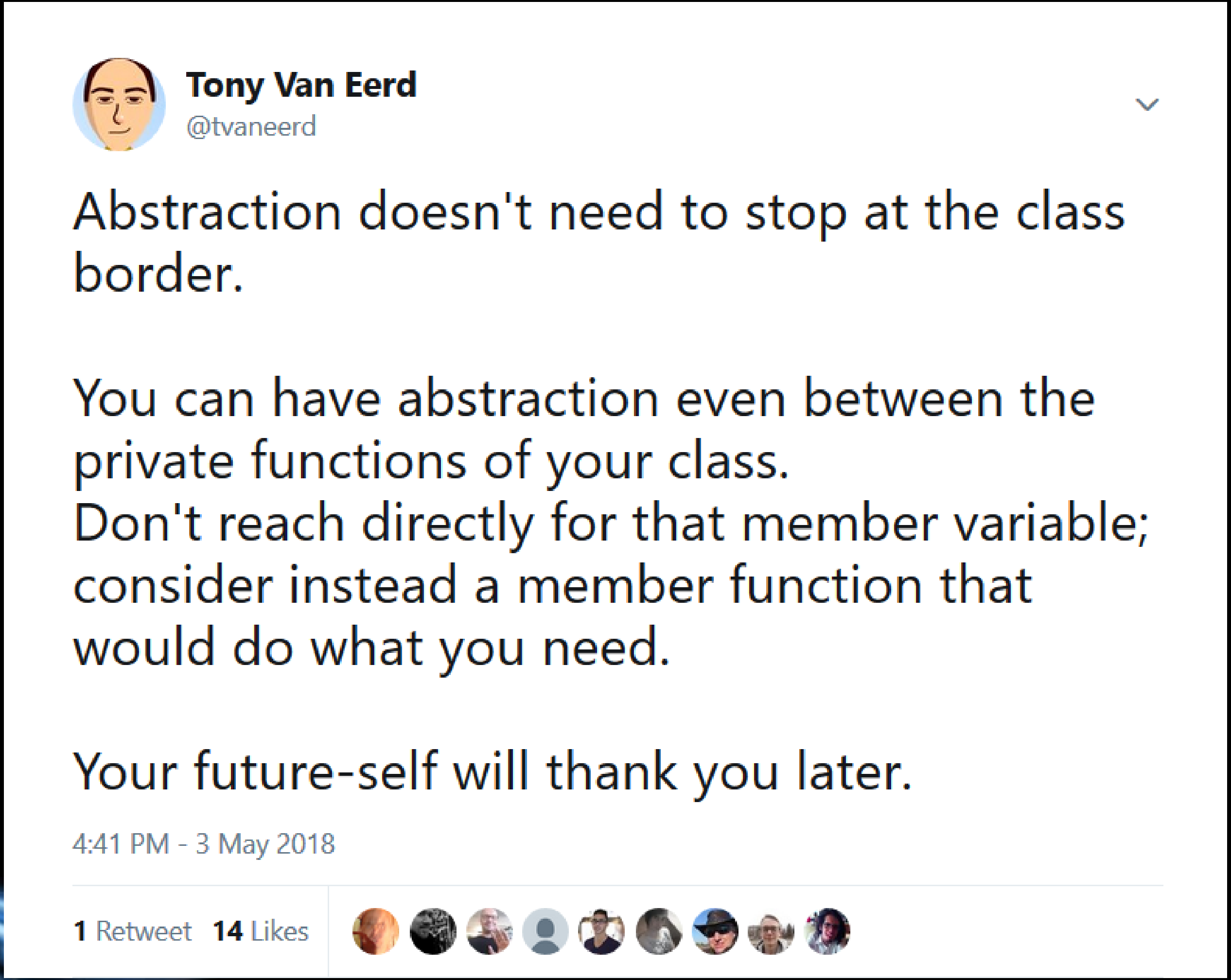
CHRISTIE

- Describe the thing in detail – what words did you use?
- Determine Essence (and does your API agree)
- Be Consiste
- Consistent
- Be *Glaring*
- NOT under
- Co-opt a te
- Avoid nega
- Avoid spok
- Avoid verb
- Be Concise
- By use or b


Tony Van Eerd
@tvaneerd

If you can't name it, you probably don't know what it is.
If you don't know what it is, you don't know what it _isn't_.
If you don't know what it isn't, you don't know what code shouldn't be in it.

So much code shouldn't be in it.



Tony Van Eerd
@tvaneerd

Abstraction doesn't need to stop at the class border.

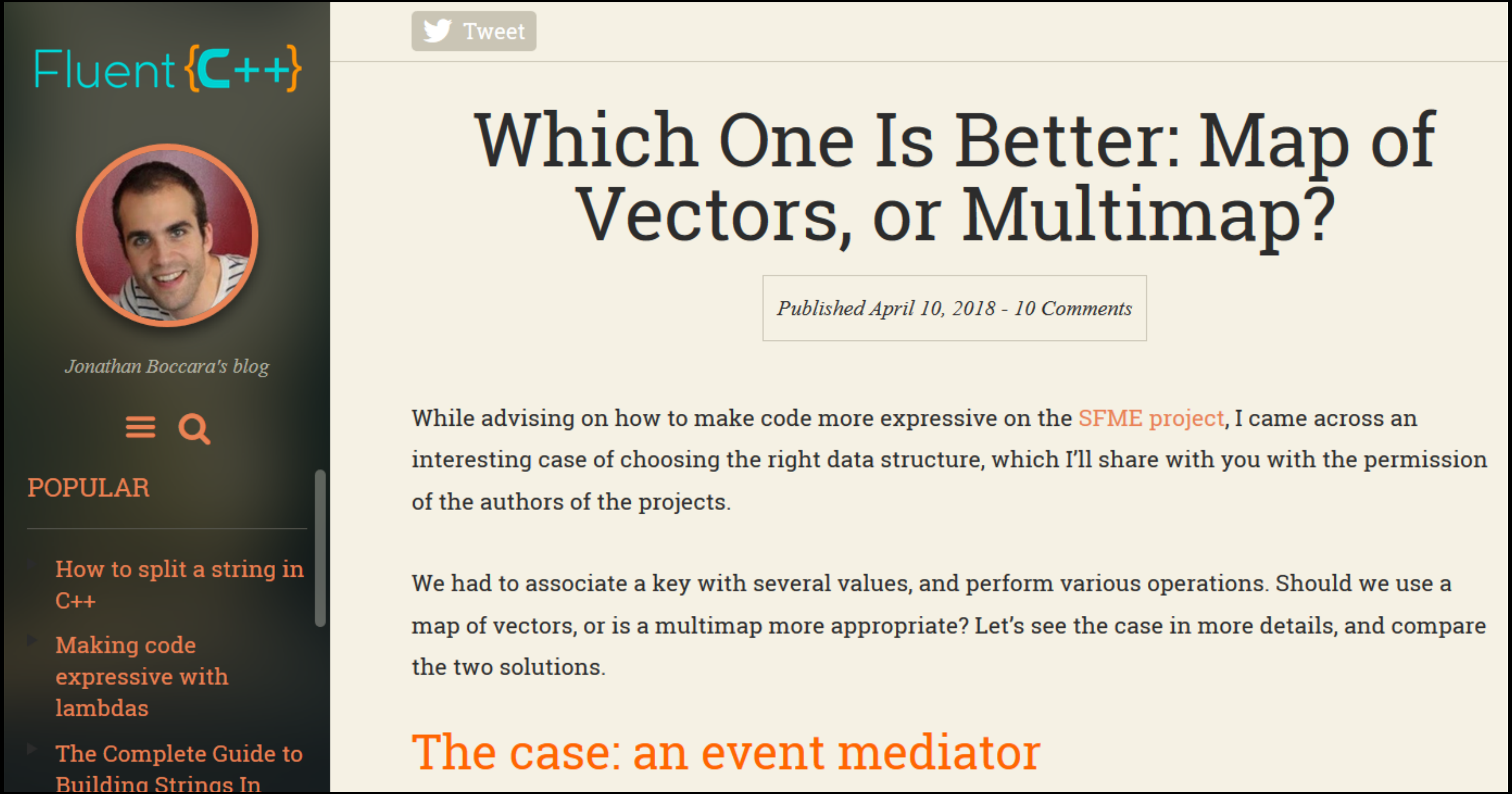
You can have abstraction even between the private functions of your class.
Don't reach directly for that member variable; consider instead a member function that would do what you need.

Your future-self will thank you later.

4:41 PM - 3 May 2018

1 Retweet **14** Likes

CHRISTIE®


Tweet

Fluent {C++}

Jonathan Boccara's blog

POPULAR

- How to split a string in C++
- Making code expressive with lambdas
- The Complete Guide to Building Strings In

Which One Is Better: Map of Vectors, or Multimap?

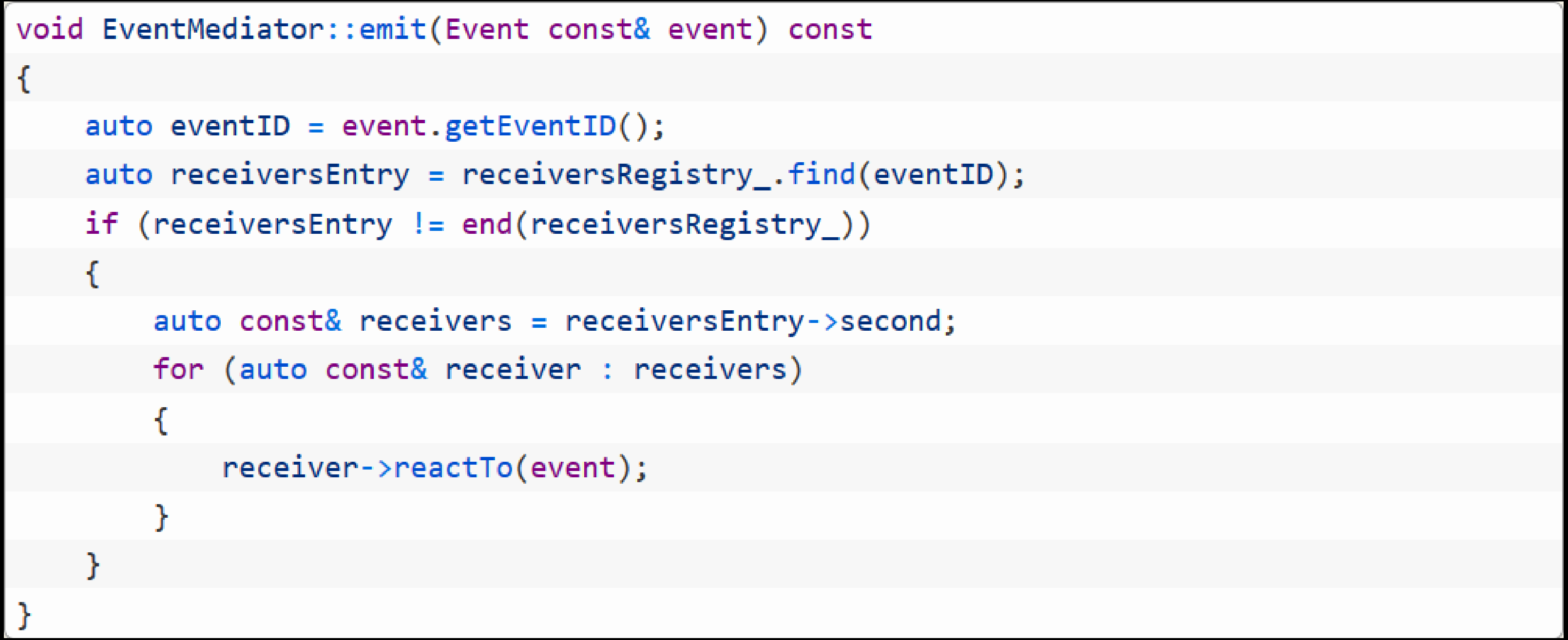
Published April 10, 2018 - 10 Comments

While advising on how to make code more expressive on the SFME project, I came across an interesting case of choosing the right data structure, which I'll share with you with the permission of the authors of the projects.

We had to associate a key with several values, and perform various operations. Should we use a map of vectors, or is a multimap more appropriate? Let's see the case in more details, and compare the two solutions.

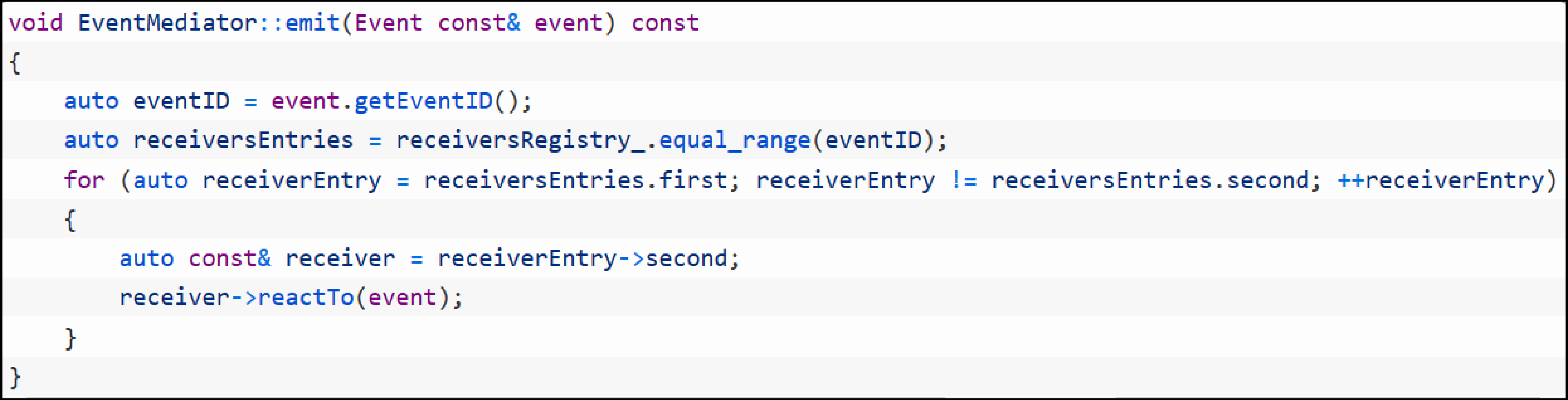
The case: an event mediator

- Jonathan Boccara



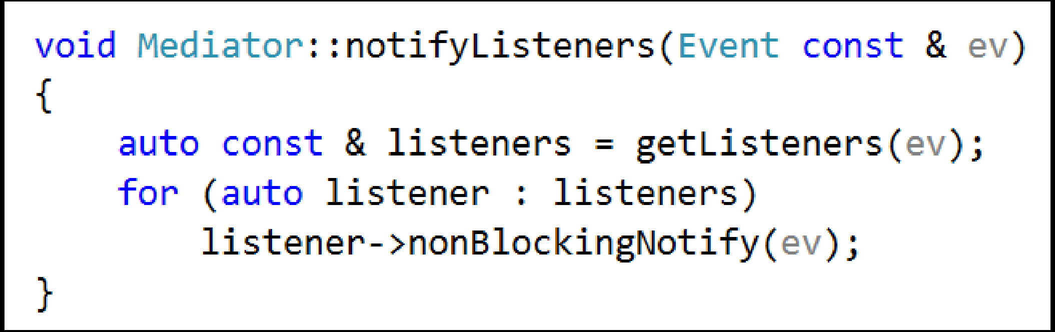
```cpp
void EventMediator::emit(Event const& event) const
{
    auto eventID = event.getEventID();
    auto receiversEntry = receiversRegistry_.find(eventID);
    if (receiversEntry != end(receiversRegistry_))
    {
        auto const& receivers = receiversEntry->second;
        for (auto const& receiver : receivers)
        {
            receiver->reactTo(event);
        }
    }
}
```

- Jonathan Boccara



```cpp
void EventMediator::emit(Event const& event) const
{
    auto eventID = event.getEventID();
    auto receiversEntries = receiversRegistry_.equal_range(eventID);
    for (auto receiverEntry = receiversEntries.first; receiverEntry != receiversEntries.second; ++receiverEntry)
    {
        auto const& receiver = receiverEntry->second;
        receiver->reactTo(event);
    }
}
```

- Jonathan Boccara



```cpp
void Mediator::notifyListeners(Event const & ev)
{
    auto const & listeners = getListeners(ev);
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}
```

- Tony Van Eerd

```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
    ent = buffer[tmp].load(relaxed);
    while( ! is_zero(ent, tmp.gen) ) {
        if (ent.gen < prev) {
           while(!tail.CAS(old,tmp) && old < tmp) { }
           return false; // full
        }  else tmp.incr();
        if (ent.data) prev = ent.gen;
    }
    geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

```
void push(int val) {
   geni pos = tailish; // relaxed load
   do {
      pos = find_tail(pos);
   } while (!try_write_value(pos, val));
   tailish = pos+1;  // thanks Sebastian
}

geni find_tail(geni pos) { // precond: pos <= tail
  while(!maybe_tail(buff[pos.val].load(relaxed), pos.gen))
      pos++;
  return pos;
}

bool maybe_tail(entry e, int gen) {
   return e.data == 0 && e.gen == gen
          || e.data  != 0 && e.gen < gen;
}

bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].c_e_weak(old, nu, release, relaxed);
}
```

#ClashOfTheClichés

“Never give up on your dreams.”

“Be careful what you wish for."

(Phil Nash)

CHRISTIE®

We're smarter than this

We're smarter than this

Apparently not

Apparently not

- Time crunch
- Instant gratification
- 80 / 20 – *"it's not a sprint, it's a marathon"*
- Uncertainty? (know this is wrong, don't know what is right)

We're smarter than this

- Pair programming
- It's not a sprint, it's a marathon - *Make it a relay*
- Positive feedback (from self, others) ie in code reviews
- Write tests – gratification
- Praise publically
- Gamify? Prizes for lines removed? Bugs fixed?
- Visualization – like atheletes; imagine the codebase you want

CHRISTIE®

"A goal is not always meant to be reached, it often serves simply as something to aim at."

- Bruce Lee

CHRISTIE®

#ClashOfTheClichés

“If at first you don't succeed, try, try again.”

“Don’t throw good money after bad.”

“Better safe than Sorry”

“A ship in harbour is safe, but that’s not what ships are built for”

“There’s no such thing as a free lunch.”

“Don’t look a gift horse in the mouth .”

“Never give up on your dreams.”

“Be careful what you wish for."

“The early bird gets the worm.”
“Good things come to those who wait.”
“Patience is a virtue.”
“Strike while the iron is hot.”

“Absence makes the heart grow fonder.”

“Out of sight, out of mind.”

The End