

# Designing a feature that doesn't fit

Patrice Roy

[Patrice.Roy@USherbrooke.ca](mailto:Patrice.Roy@USherbrooke.ca)

Université de Sherbrooke / Collège Lionel-Groulx

CppCon 2017

# Who am I?

- Father of five (four girls, one boy), ages 22 to 4
- Feeds and cleans up after a varying number of animals
- Used to write military flight simulator code, among other things
- Full-time teacher since 1998
- Works a lot with game programmers
- Incidentally, WG21 and WG23 member
- Involved in SG14, among other study groups

# Overview

- Working with the standards committee is fun, enlightening, sometimes frustrating of course...
  - ... and a *lot* of (fun) work!
- WG21 members collectively come from diverse backgrounds, and have customers with different needs
  - What we all have in common is that we both care deeply and strongly about our customers, and about the language
  - We aim to make the language better, for our users and for the language itself
- Sometimes, these two aims somewhat conflict

# Overview

- This presentation has a few objectives
  - Showing one possible pathway from the expression of user needs up to the committee work that follows
  - Giving an overview of the process involved in bringing about a feature
  - Providing a kind of look behind the curtain (respectful of ISO rules) for those who have not had the chance to go to a committee meeting
- Be warned, there is no resolution (yet)
  - The work described here is ongoing as of this presentation
  - I hope we will have a satisfactory resolution. I have my opinion as to where it should lead, but the future is full of surprises

# An actual need

- Proposals generally emerge from an actual need
  - ... or from a need to fix something that we see as broken
- Be warned: some might disagree with you as to the « relative brokenness » of a feature
  - These people are not idiots
  - If you believe strongly in something, it's up to you to make a convincing argument
    - ...and that's only part of the job

# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community

# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community

An orange speech bubble with a tail pointing towards the bottom-left.

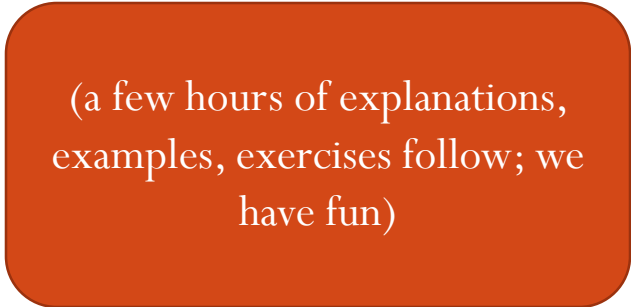
Here's what standard  
threading facilities offer  
starting with C++11

# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community



Here's what standard  
threading facilities offer  
starting with C++11



(a few hours of explanations,  
examples, exercises follow; we  
have fun)

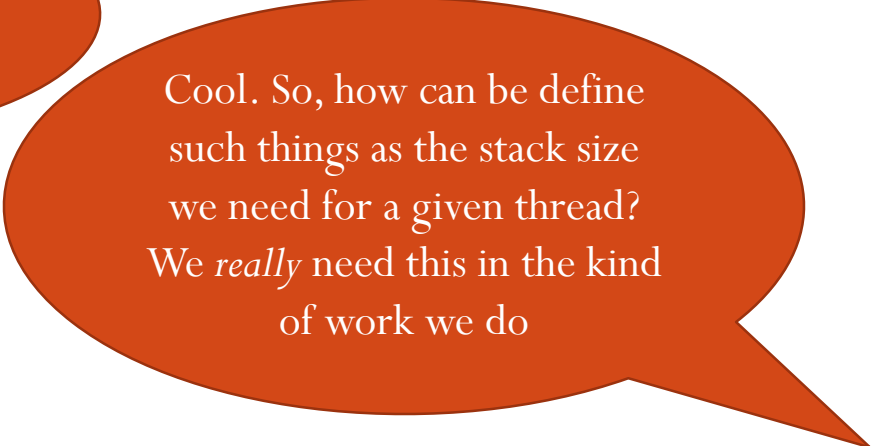


# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community



Here's what standard  
threading facilities offer  
starting with C++11




Cool. So, how can we define  
such things as the stack size  
we need for a given thread?  
We *really* need this in the kind  
of work we do

# An actual need

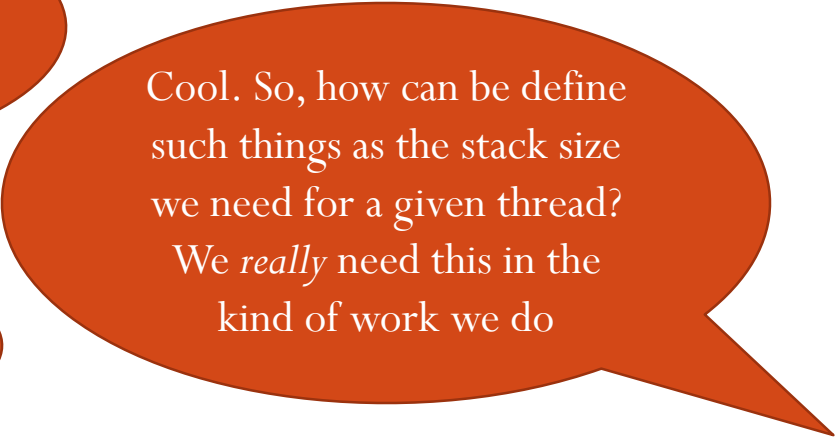
- Except paraphrasing an actual exchange with a prominent member of the game programming community



Here's what standard  
threading facilities offer  
starting with C++11



Well, you can't,  
currently



Cool. So, how can we define  
such things as the stack size  
we need for a given thread?  
We *really* need this in the  
kind of work we do

# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community

Here's what standard  
threading facilities offer  
starting with C++11

Well, you can't,  
currently

Cool. So, how can we define  
such things as the stack size  
we need for a given thread?  
We *really* need this in the  
kind of work we do

...I guess we'll stick to  
platform-specific facilities  
then

# An actual need

- Except paraphrasing an actual exchange with a prominent member of the game programming community



Here's what standard  
threading facilities offer  
starting with C++11

Well, you can't,  
currently

Cool. So, how can we define  
such things as the stack size  
we need for a given thread?  
We *really* need this in the  
kind of work we do

...I guess we'll stick to  
platform-specific facilities  
then

(darn)

# Overview of the problem statement

- We have users with significant multi-threading needs and relevant use-cases that will not use standard threading facilities
  - Some of these non-users would *really* like to use standard threading facilities
  - Others never will, of course, but that's another story
- This leads them to reject a not-insignificant part of standard C++ and creates a kind of ghetto
  - There are precedents in C++, e.g. exceptions
- What seems to be missing?

# Overview of the problem statement

- Most operating systems provide a way to control various characteristics of platform threads
- Some characteristics can be changed at run-time
  - For these, we have `std::thread::native_handle`
  - This lets users stick with `std::thread` for the most part, and go into « platform-specific land » for very specialized needs
- Some characteristics have to be fixed at thread creation time
  - Stack size comes to mind
  - Other aspects (affinity, naming, priority, scheduling, « create detached ») could also be discussed here
  - For these, there is no solution within standard C++ threading facilities today

# Overview of C++'s abstract machine

- What follows is a (brief!) overview of selected parts of C++'s abstract machine
  - Only what's required to grasp the key issues
  - If C++'s abstract machine interests you, I gave another talk on that specific question during CppCon 2017

# Overview of C++'s abstract machine

- We program to an abstract machine, defined by the standard
- This machine describes the contours of what guarantees are provided for your programs
  - You will see there such things as the so-called « as if » rule, which gives optimizers leeway when transforming your code into something hopefully smaller and faster
    - ... as long as the observable behavior stays the same
    - The standard defines what « observable behavior » means
  - This machine also describes such things as implementation-defined behavior and undefined behavior
    - Ideally, avoid UB and avoid relying on implementation-defined behavior



# Overview of C++'s abstract machine


- Why is this important to know?

# Overview of C++'s abstract machine

- Except paraphrasing an actual exchange with a prominent member of WG21

# Overview of C++'s abstract machine


- Except paraphrasing an actual exchange with a prominent member of WG21



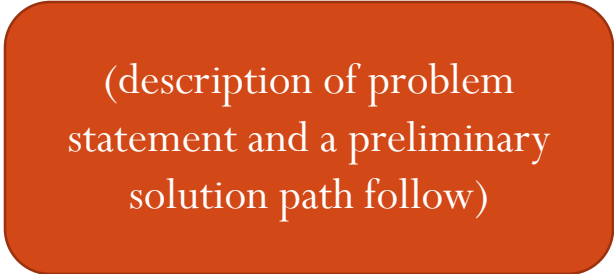
Here's why my users  
want to control the stack  
size of their threads, and  
why I suggest...

# Overview of C++'s abstract machine

- Except paraphrasing an actual exchange with a prominent member of WG21




Here's why my users  
want to control the stack  
size of their threads, and  
why I suggest...



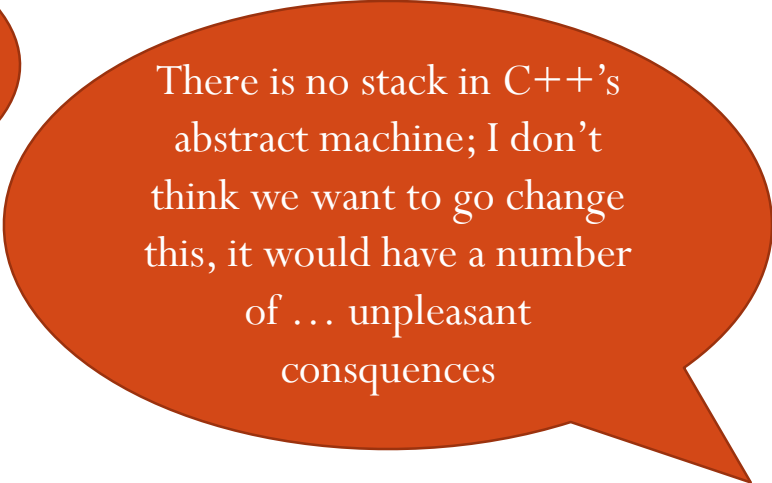
(description of problem  
statement and a preliminary  
solution path follow)

# Overview of C++'s abstract machine

- Except paraphrasing an actual exchange with a prominent member of WG21




Here's why my users  
want to control the stack  
size of their threads, and  
why I suggest...



There is no stack in C++'s  
abstract machine; I don't  
think we want to go change  
this, it would have a number  
of ... unpleasant  
consequences

# Overview of C++'s abstract machine

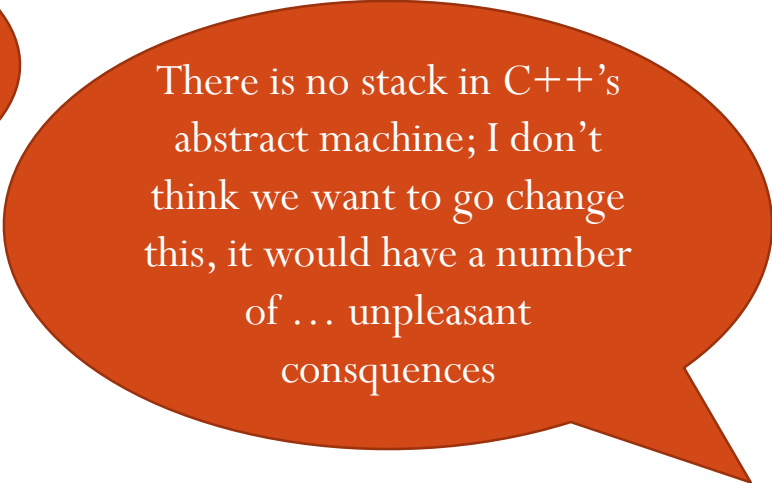
- Except paraphrasing an actual exchange with a prominent member of WG21



Here's why my users  
want to control the stack  
size of their threads, and  
why I suggest...



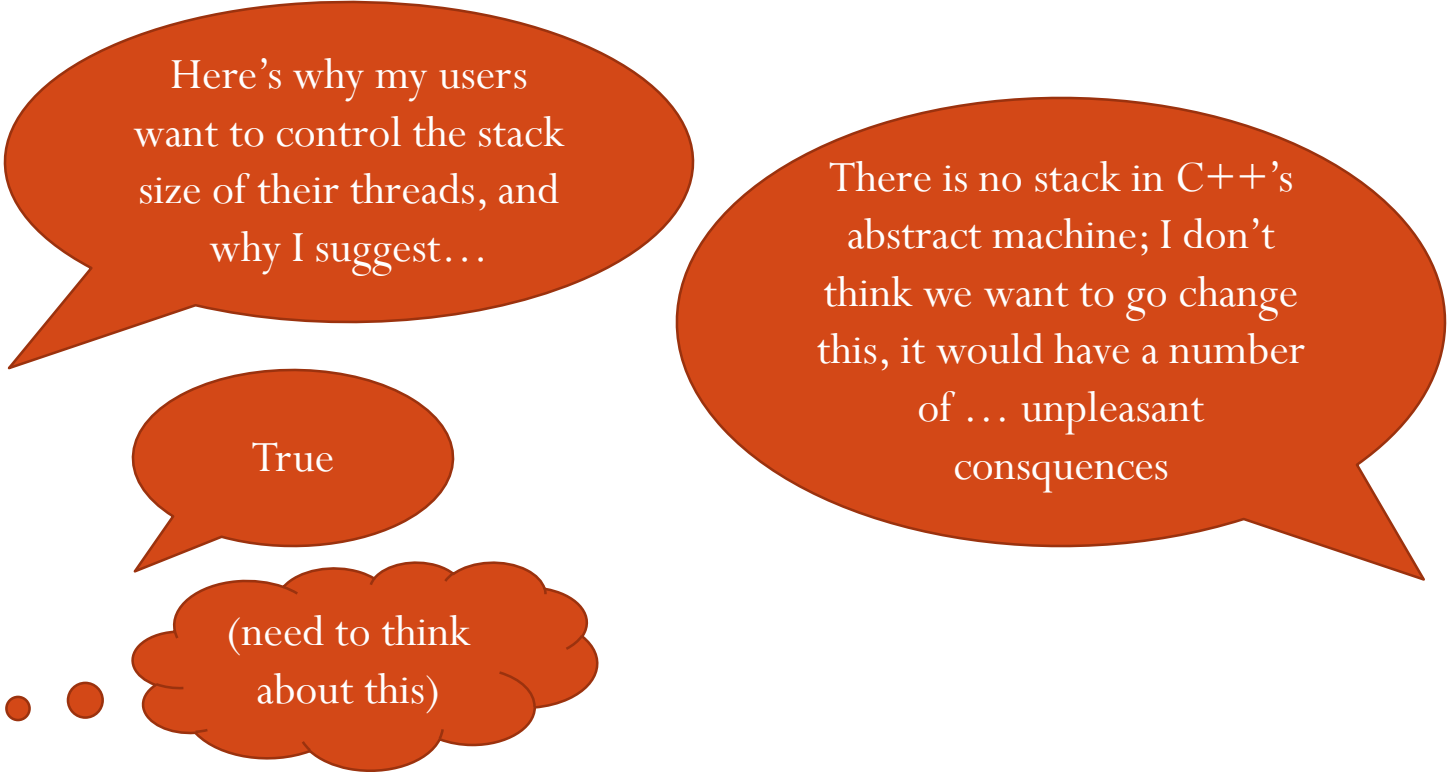
True



There is no stack in C++'s  
abstract machine; I don't  
think we want to go change  
this, it would have a number  
of ... unpleasant  
consequences

# Overview of C++'s abstract machine

- Except paraphrasing an actual exchange with a prominent member of WG21



Here's why my users  
want to control the stack  
size of their threads, and  
why I suggest...

True

...  
(need to think  
about this)

There is no stack in C++'s  
abstract machine; I don't  
think we want to go change  
this, it would have a number  
of ... unpleasant  
consequences

# Revisiting the problem statement

- Some users want control over the stack size of some of their threads
  - This is a special case of the general problem
- Standard wording does not define an execution stack at all
  - Hard to write standardese to use something the standard itself does not really acknowledge
- Since we have an actual problem statement targeting the standard, but expressed in terms the standard does not really acknowledge, we face an interesting problem: designing a feature that, essentially, does not fit



# Revisiting the problem statement

- The base problem involves offering a mechanism that makes standard threading facilities useful for a significant class of users
- The associated problem is wording this mechanism in terms that do not « pollute » the standard with definitions we'd rather avoid

# Exploring the design space

- I took a first stab at this problem during the SG14 meeting held at CppCon 2015
  - It was a last-minute proposal, written in such a way to offer a workable solution but expected to be rejected, hopefully getting the ball rolling along the way and start discussions
  - The proposal was essentially to add constructors to `std::thread` which would take the required stack size as argument
  - This was where the absence of an actual stack in standard wording was brought in the discussion

# Exploring the design space

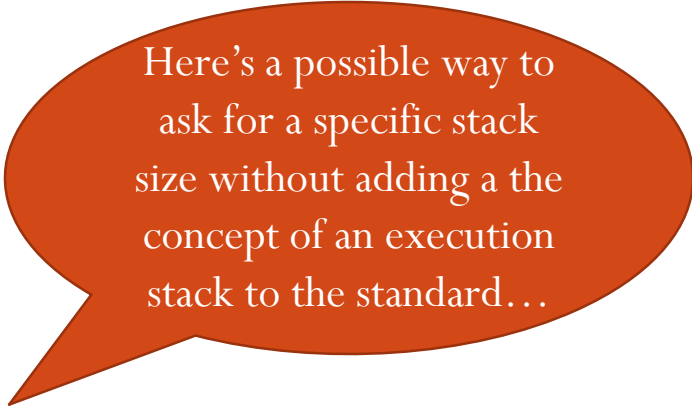
- Discussions continued, and others contributed
  - Special thanks to Vicente J. Botet Escribá, who wrote P0320R0 which was discussed in Oulu
  - I was called in to participate in the discussions there
    - We'll get back to this

# Exploring the design space

- Excerpt paraphrasing an actual exchange with another prominent member of WG21

# Exploring the design space

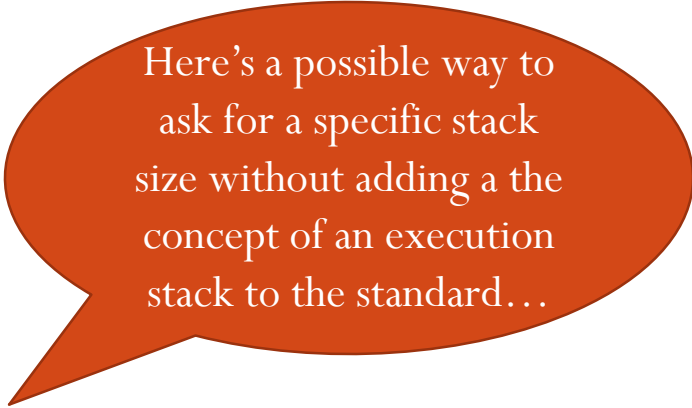
- Except paraphrasing an actual exchange with another prominent member of WG21



Here's a possible way to ask for a specific stack size without adding a the concept of an execution stack to the standard...

# Exploring the design space

- Excerpt paraphrasing an actual exchange with another prominent member of WG21



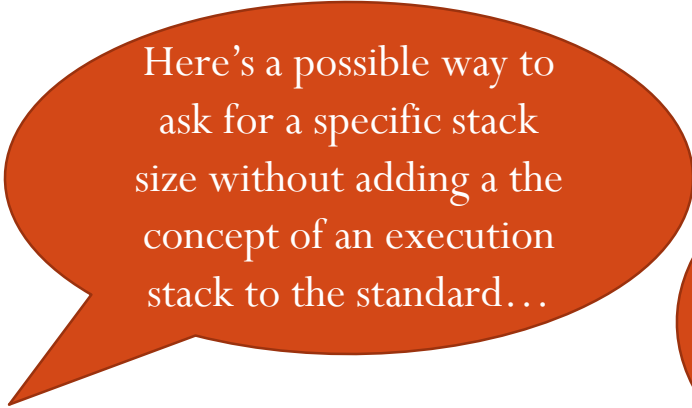
Here's a possible way to ask for a specific stack size without adding a the concept of an execution stack to the standard...



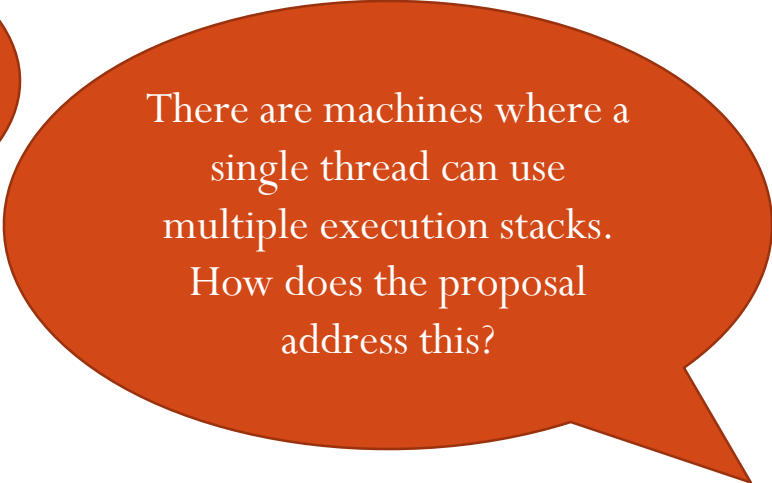
(goes on describing the design under discussion)

# Exploring the design space

- Excerpt paraphrasing an actual exchange with another prominent member of WG21



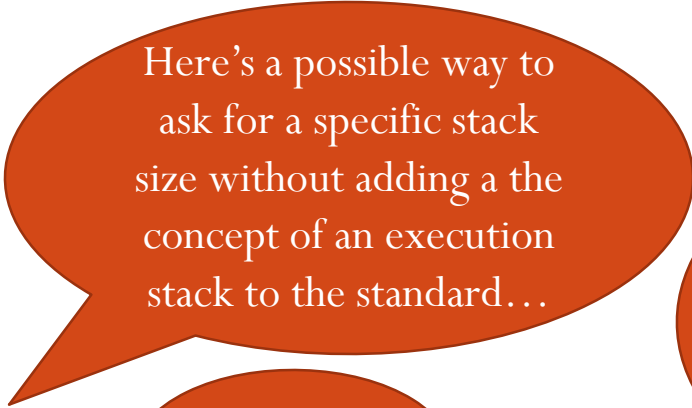
Here's a possible way to ask for a specific stack size without adding a the concept of an execution stack to the standard...



There are machines where a single thread can use multiple execution stacks. How does the proposal address this?

# Exploring the design space

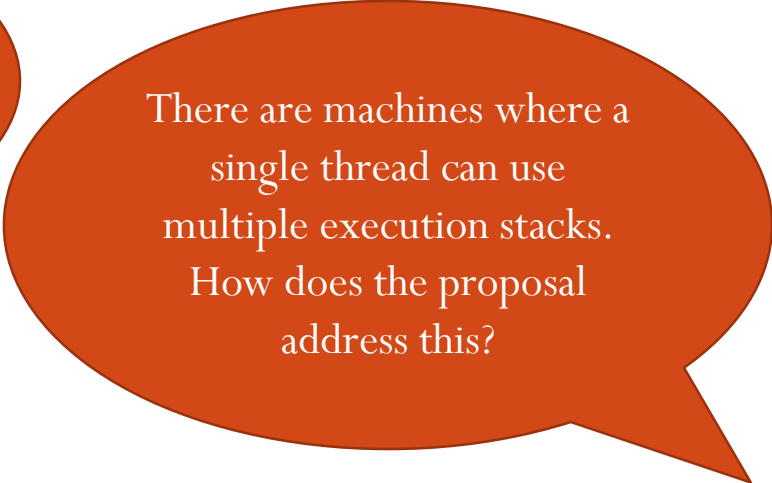
- Excerpt paraphrasing an actual exchange with another prominent member of WG21



Here's a possible way to ask for a specific stack size without adding a the concept of an execution stack to the standard...



True

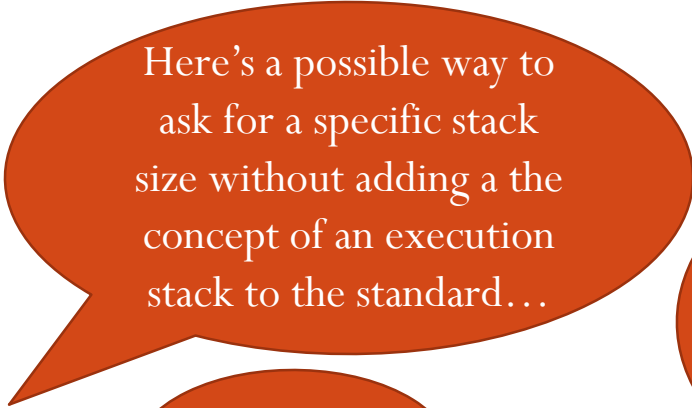


There are machines where a single thread can use multiple execution stacks. How does the proposal address this?



# Exploring the design space

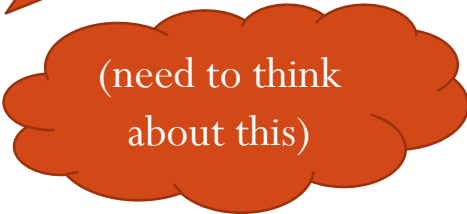
- Excerpt paraphrasing an actual exchange with another prominent member of WG21



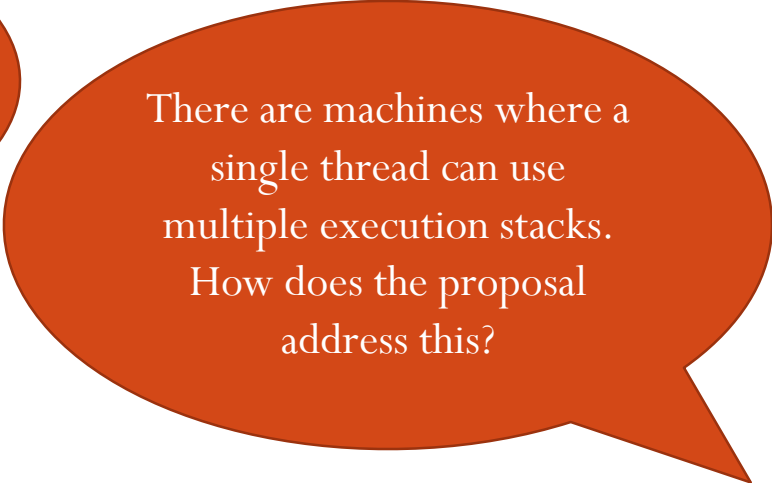
Here's a possible way to ask for a specific stack size without adding a the concept of an execution stack to the standard...



True



...  
(need to think about this)



There are machines where a single thread can use multiple execution stacks. How does the proposal address this?

# Overview of the WG21 process

# Overview of the WG21 process

- (*very* rough overview)

# Overview of the WG21 process

Stating the problem

# Overview of the WG21 process

Stating the problem

If you don't have any real use-case (problem to fix, feature to add or remove, type to add, etc.), there are probably better forums than WG21 for your ideas

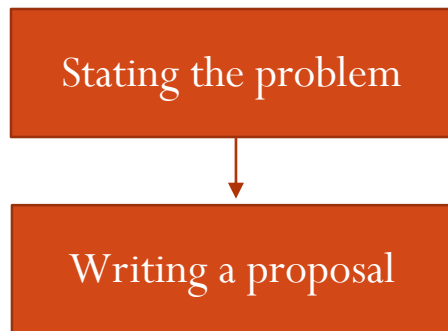
# Overview of the WG21 process

Stating the problem

If you don't have any real use-case (problem to fix, feature to add or remove, type to add, etc.), there are probably better forums than WG21 for your ideas

WG21 is volunteer work. We only take on a problem if there's a proposal, and we have to prioritize

# Overview of the WG21 process

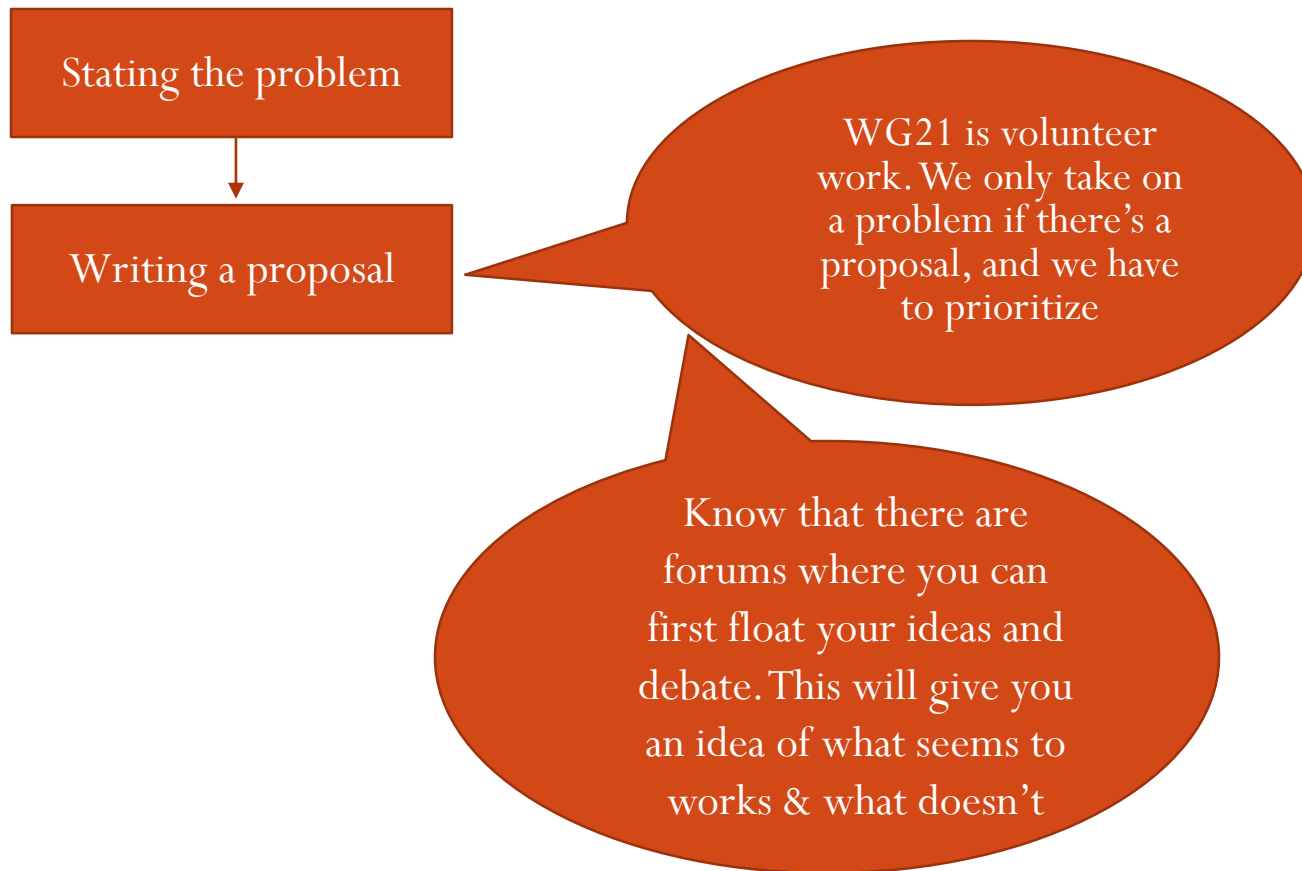


# Overview of the WG21 process

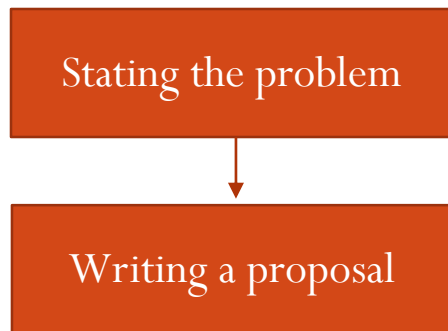




# Overview of the WG21 process



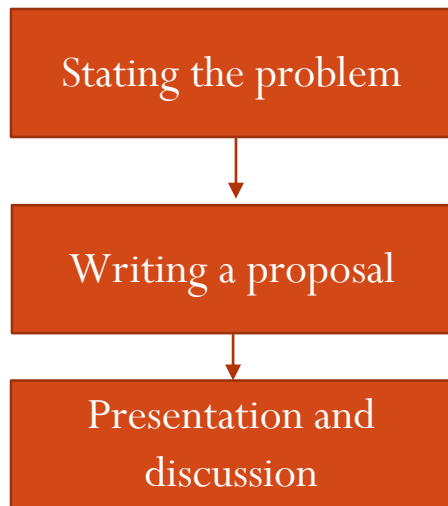
# Overview of the WG21 process



While we're at it: it's cool if you have a reference implementation (shows it can be done, shows the tradeoffs), and it's *really* cool if you have user experience

However, write your proposals expressing what the feature does. Particularly for library features. Let the implementers implement it (they're *very, very* good)

# Overview of the WG21 process



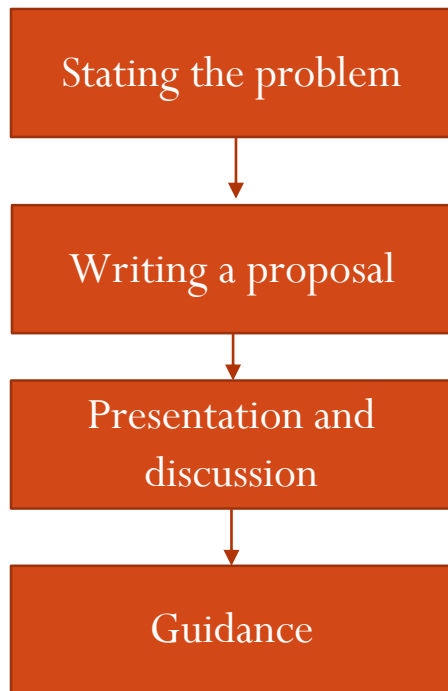
# Overview of the WG21 process



# Overview of the WG21 process



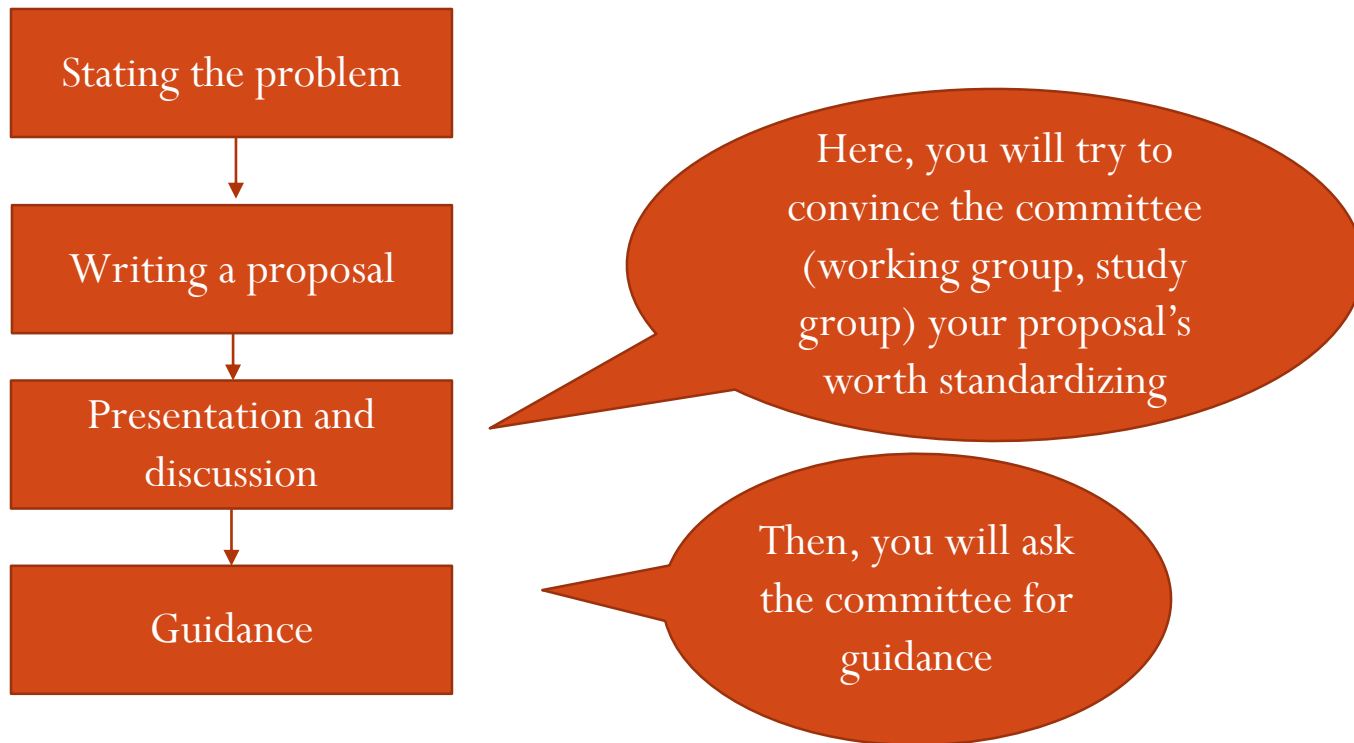
# Overview of the WG21 process



# Overview of the WG21 process

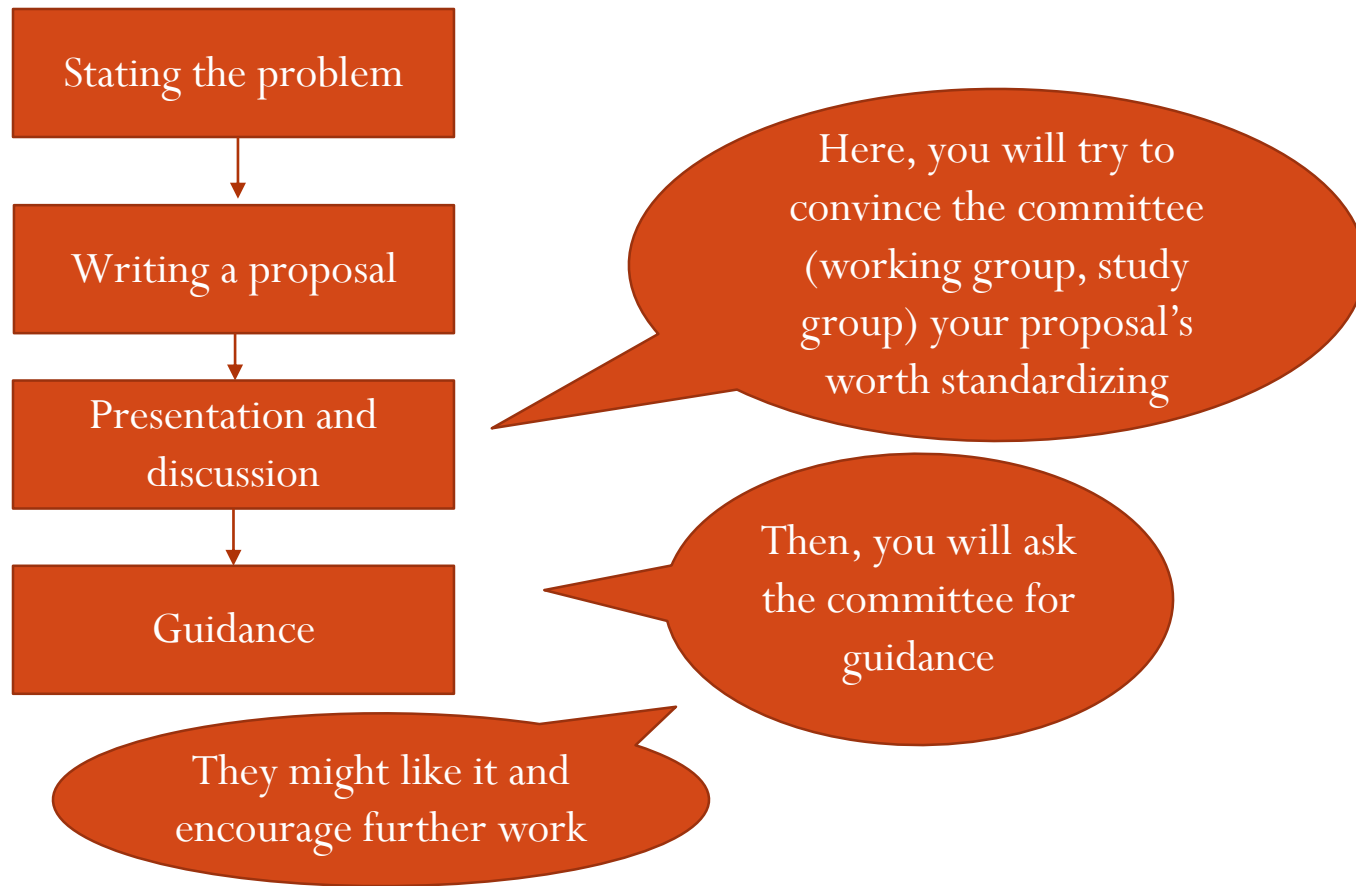


# Overview of the WG21 process





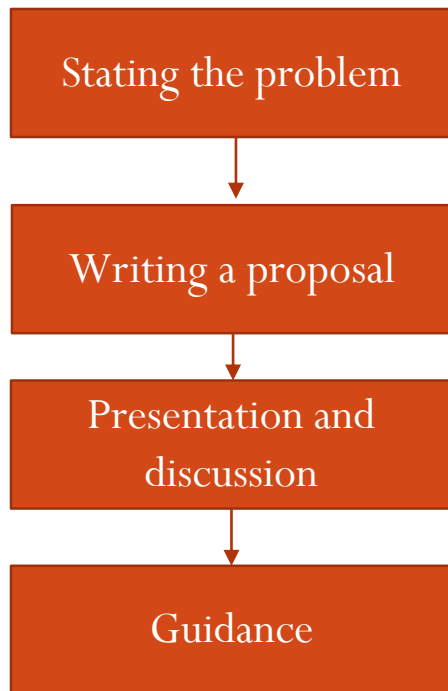
# Overview of the WG21 process



# Overview of the WG21 process



# Overview of the WG21 process

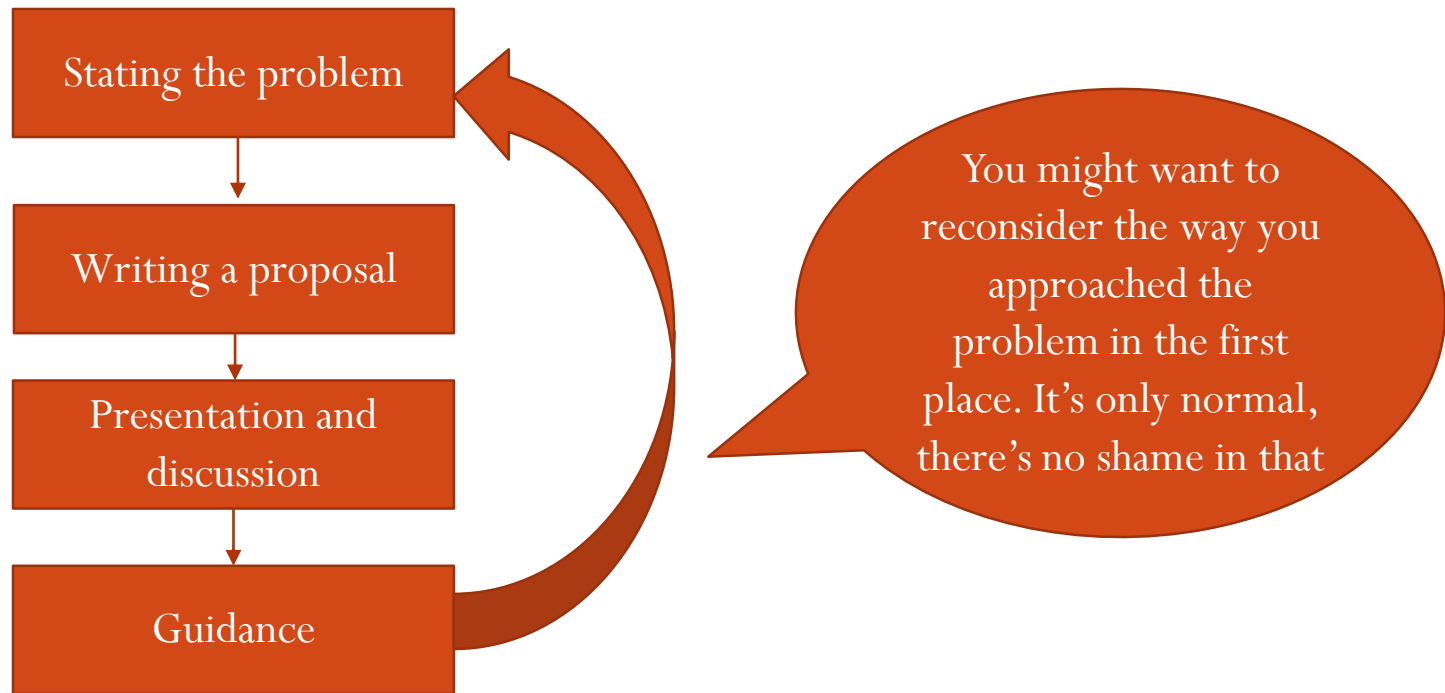


Committee members are not dumb and uncaring. It's your job to convince them. Pay attention to the guidance you will be provided

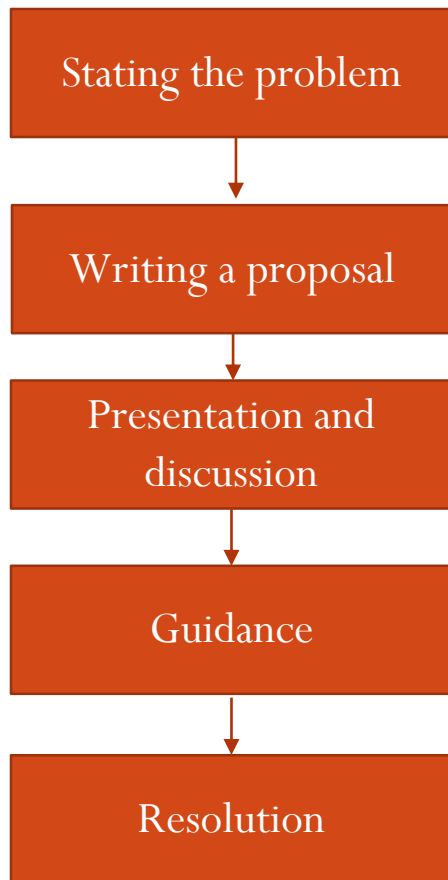
# Overview of the WG21 process



# Overview of the WG21 process



# Overview of the WG21 process



In the end (this might take months, years, decades...), it might get accepted, or you might decide to stop working on it, or there might be another proposal that makes yours obsolete, or...

In this particular case...

# In this particular case...

- The problem was stated at the 2015 SG14 meeting held at CppCon
  - It's an official ISO meeting, with official ISO rules
  - There was useful feedback
    - The C++ community as a whole is just awesome!
    - Many directions to explore were suggested



# In this particular case...

- The problem was stated at the 2015 SG14 meeting held at CppCon
  - It's an official ISO meeting, with official ISO rules
  - There was useful feedback
    - The C++ community as a whole is just awesome!
- As mentioned before Vicente J. Botet Escribá took a stab at this with P0320R0
  - <http://wg21.link/P0320R0>
  - Discussed with SG1 in Oulu 2016

# In this particular case...

- Vicente J. Botet Escribá took a stab at this with P0320R0
  - <http://wg21.link/P0320R0>
  - The proposal suggested `thread::attributes`
    - Included accessory functionalities such as ways to query the actual attributes of individual threads at run-time
    - e.g. `thread::get_attributes()`

# In this particular case...

- Vicente J. Botet Escribá took a stab at this with P0320R0
  - <http://wg21.link/P0320R0>
  - The proposal suggested `thread::attributes`
    - Objects that would be accepted by `std::thread` constructors
    - Their actual implementation would be vendor-specific
    - The trick would be to try to get vendors to agree on a common subset of attributes without actually specifying them in the standard
    - It's a clever approach

# In this particular case...

- Vicente J. Botet Escribá took a stab at this with P0320R0
  - <http://wg21.link/P0320R0>
  - Discussed with SG1 in Oulu 2016
  - The proposal suggested `thread::attributes`
    - Objects that would be accepted by `std::thread` constructors
    - Their actual implementation would be vendor-specific
    - The trick would be to try to get vendors to agree on a common subset of attributes without actually specifying them in the standard
- SG1 guidance was to explore the design space further
  - SG1 provided a number of edge cases that required special consideration

# In this particular case...

- Vicente J. Botet Escribá went back and worked on P0320R1
  - <http://wg21.link/P0320R1>
- Billy Baker, Arthur O'Dwyer and myself worked on a complementary proposal, P0484R0
  - <http://wg21.link/P0484R0>
- I championed both of these before SG1 in Issaquah 2016

# In this particular case...

- Why two proposals? Orthogonal issues
  - Specifying the needs for feature support at thread construction time (the P0320 series)
  - Properly reporting failure from vendors to comply with such a request (the P0484 series)
- P0320 does not depend on P0484
- P0484 requires something such as P0320, but it could be something else than that specific proposal

# In this particular case...

- Reporting failure to comply?
  - Adding `thread::attribute`-aware constructors means that reporting failure to comply strictly based on this mechanism would lead to error reporting through exceptions

# In this particular case...

- Reporting failure to comply?
  - Adding `thread::attribute`-aware constructors means that reporting failure to comply strictly based on this mechanism would lead to error reporting through exceptions
  - It so happens that the users for the initial motivating use-cases, programmers from the games programming community, typically do not use exceptions
    - They compile with `-fnoexcept` and the like



# In this particular case...

- Reporting failure to comply?
  - Adding `thread::attribute`-aware constructors means that reporting failure to comply strictly based on this mechanism would lead to error reporting through exceptions
  - It so happens that the users for the initial motivating use-cases, programmers from the games programming community, typically do not use exceptions
    - They compile with `-fnoexcept` and the like
  - P0484R0 introduced a `make_thread()` factory function

# In this particular case...

- P0484R0 introduced a `make_thread()` factory function
  - Could express failure to comply through other means than exceptions
    - `optional<thread>?`
    - `expected<thread, error_code>?`
    - A composite object providing detailed diagnostics as to which attribute requests could be achieved and which ones could not
    - etc.
- Does not preclude `thread::attributes`-aware constructors for users that want it

# In this particular case...

- Both P0320R1 and P0484R0 were discussed in Issaquah 2016
- In SG1, some really liked it and some expressed concerns
  - The long-term goal is to express abstractions through executors
  - Users express a need today
  - Are these facilities in tune with the direction of concurrent programming in C++?

# In this particular case...

- Both P0320R1 and P0484R0 were discussed in Issaquah 2016
- In SG1, some really liked it and some expressed concerns
  - The long-term goal is to express abstractions through executors
  - Users express a need today
  - Are these facilities in tune with the direction of concurrent programming in C++?
    - It's very interesting to debate design and fine technical considerations with such high-level experts!

# In this particular case...

- SG1 guidance in Issaquah 2016 was to explore two avenues
  - The `make_thread()` factory function using `thread::attributes`
    - Details further the design tradeoffs
    - Provide a sketch of a possible implementation
  - A platform-specific-thread adoption mechanism
    - Create the thread using platform-specific tools
    - Use the « resulting native handle » and pass it to a `std::thread` constructor for adoption
- SG1 stated that whatever approach was picked in the end, a `thread::get_attributes()` querying mechanism would be useful

# In this particular case...

- SG1 guidance in Issaquah 2016 was to explore two avenues
  - The `make_thread()` factory function using `thread::attributes`
  - A platform-specific-thread adoption mechanism
- SG1 stated that whatever approach was picked in the end, a `thread::get_attributes()` querying mechanism would be useful
- SG1 added that they would prefer a low-granularity (succeeded, failed) error reporting mechanism
  - No need to support partial success and provide information as to which attributes requests succeeded and which ones failed

# In this particular case...

- Both P0320 and P0484 stayed put for the Kona 2017 meeting
  - Sometimes, there's just no space left in our schedule
  - Volunteer work is nice, but there's this little thing called life (family, work and such) that gets in the way

# In this particular case...

- Both P0320 and P0484 stayed put for the Kona 2017 meeting
  - Sometimes, there's just no space left in our schedule
  - Volunteer work is nice, but there's this little thing called life (family, work and such) that gets in the way
- For Toronto 2017, we brought P0484R1
  - <http://wg21.link/P0484R1>
  - Billy Baker, Arthur O'Dwyer and myself tried to address the concerns expressed by SG1 and offer a summary exploration of the competing approaches (platform-specific-thread adoption, `thread::attributes`-aware factory function)



# In this particular case...

- Thread adoption is tricky
  - The very concept of a « native handle » is deliberately vague in the standard (it's understandable)
    - In a platform such as Win32, is it the HANDLE or the thread id?
  - In what state is the platform-specific-thread at adoption time?
    - Is it running?
    - It is created but suspended?
    - Has it successfully completed execution? Has it failed?
    - Is the native handle even valid?
  - Also, if we make users go through platform-specific thread creation tools, they might stick to platform-specific tools

# In this particular case...

- The `make_thread()` factory function part of P0484R1 was refined
  - Takes into account SG1 comments
    - Simplified mechanism to report failure to comply
  - Formally supports the `thread::get_attributes()` part of P0320
    - There are a few possible APIs for this
  - We (the authors) expressed a preference for this approach, but showed both approaches could be made to work

# In this particular case...

- Committee work is not just technical (although the technical part is essential)
  - There are political aspects
  - There are human aspects
  - You have to gather experts' interest
  - You have to convince
  - You have to listen and take guidance into consideration

# In this particular case...

- Committee work is not just technical (although the technical part is essential)
  - There are political aspects
  - There are human aspects
  - You have to gather experts' interest
  - You have to convince
  - You have to listen and take guidance into consideration
  - ... and you have to be lucky, sometimes

# In this particular case...

- Committee work is not just technical (although the technical part is essential)
  - ... you have to be lucky, sometimes
- In Toronto 2017, we ended up in SG1 at the very end of the week
  - SG1 has its hands full, so it's understandable

# In this particular case...

- In Toronto 2017, we ended up in SG1 at the very end of the week
  - Presentation of P0484R1 began
  - Discussions were interesting
  - Questions were relevant

# In this particular case...

- In Toronto 2017, we ended up in SG1 at the very end of the week
  - Presentation of P0484R1 began
  - Discussions were interesting
  - Questions were relevant
  - ... but time was running out
  - We all decided it would be better not to rush this and do things right at the next meeting in Albuquerque 2017

# What does the future hold?

- The plan is to continue discussion in Albuquerque 2017
- We still don't know if P0320 will be accepted
- We still don't know if P0484R1 (either approach) will be accepted



# What does the future hold?

- The plan is to continue discussion in Albuquerque 2017
- In the meantime, other work happens
  - The heterogeneous computing efforts have brought D0737r0 : Execution Context of Execution Agents
    - Defines a minimal execution context type
    - One category is a thread execution resource
  - Might subsume the `thread::attributes` proposal (unclear at this time)
    - Upside: fits well with executor proposals (admittedly the preferred direction for the future)

# What does the future hold?

- The plan is to continue discussion in Albuquerque 2017
- In the meantime, other work happens
  - The heterogeneous computing efforts have brought D0737r0 : Execution Context of Execution Agents
  - Might subsume the `thread::attributes` proposal (unclear at this time)
- The P0484 series could conceivably use either the results of the P0320 series or the D0737 series
  - Will probably be part of the design discussions in Albuquerque
  - Hard to tell where this will lead us

# Conclusion?

# Conclusion?

- There is no conclusion yet
- The process itself is interesting

# Conclusion?

- On the design side
  - Understanding the problem
  - Describing actual use-cases
  - Coming up with a solution that fits with the rest of the standard

# Conclusion?

- On the design side
  - Understanding the problem
  - Describing actual use-cases
  - Coming up with a solution that fits with the rest of the standard
    - In this case, it's tricky
    - The feature to represent has a few facets that conflict with the standard
    - Standardizing just the right things, nothing more
    - Error reporting has to take into account the context of target users

# Conclusion?

- On the technical side
  - Describing possible APIs
  - Exploring the costs of individual approaches
  - Writing possible user code
  - Comparing the resulting use-cases
  - Identifying the painful bits
    - Implementation-specific elements
    - Risks of undefined behavior
    - Granularity of diagnostics
    - Even adding constructor arguments can introduce ambiguity

# Conclusion?

- On the technical side
  - Describing possible APIs
  - Exploring the costs of individual approaches
  - Writing possible user code
  - Comparing the resulting use-cases
  - Identifying the painful bits
    - Implementation-specific elements
    - Risks of undefined behavior
    - Granularity of diagnostics
    - Even adding constructor arguments can introduce ambiguity
  - Your design is never in isolation
  - The standard is an actual object that we need to care for



# Conclusion?

- On the human side
  - Lots of work
  - Often teamwork
    - My co-authors are brilliant individuals and pleasant human beings
  - Requires patience, humility
    - Months, years...
  - One reason for committee work is that, paraphrasing Bjarne earlier this week: your good idea is not always as good as you initially thought it was
    - The process makes it better
    - It's Ok. We care

# Conclusion?

- To be continued...

# Tools you should know and use

- Maurice Bos' wg21.link
  - <http://wg21.link/p0484> for example
- Tim Song's CppWp, an online reference to C++'s Draft International Standard
  - <http://eel.is/c++draft/>

# References

- My own site (sorry for the color scheme)
  - <http://h-deb.clg.qc.ca/>
  - It's in French

# Questions? Comments? Threats?