

+ 23

Things Happening in SG14...

PATRICE ROY



20
23



October 01 - 06

Things happening in SG14

Making C++ better for game development & other low-latency domains

Patrice Roy, CeFTI, Université de Sherbrooke, Collège Lionel-Groulx

Patrice.Roy@USherbrooke.ca, Patrice.Roy@clg.qc.ca

Who am I?

- Father of five, aged 28 to 10
- I feed and take care of a varying number of animals
 - Take a look at [Paws of Britannia](#) with your favorite search engine
- I used to write software for industrial electrical breakers and military flight simulators
 - CAE Electronics Ltd, IREQ
- Full-time professor since 1998
 - Collège Lionel-Groulx, Université de Sherbrooke
- I work a lot with game programmers
- WG21 and WG23 member (but I missed recent WG23 meetings)
 - Involved quite a bit with SG14, the low-latency study group
 - Occasional WG21 secretary
- etc.

Who am I?

- Father of five, aged 28 to 10
- I feed and take care of a varying number of animals
 - Take a look at [Paws of Britannia](#) with your favorite search engine
- I used to write software for industrial electrical breakers and military flight simulators
 - **CAE Electronics Ltd, IREQ**
- Full-time professor since 1998
 - Collège Lionel-Groulx, Université de Sherbrooke
- **I work a lot with game programmers**
- ~~WG21 and WG23 member (but I missed recent WG23 meetings)~~
 - **Involved quite a bit with SG14, the low-latency study group**
 - Occasional WG21 secretary
- etc.

Things happening in SG14

- The abstract of this talk is as follows:
 - « *The C++ committee is made of a small number of working groups and of a larger number of study groups, including SG14 (low-latency, finances, games and embedded systems).*

Things happening in SG14

- The abstract of this talk is as follows:
 - *« The C++ committee is made of a small number of working groups and of a larger number of study groups, including SG14 (low-latency, finances, games and embedded systems).*
 - *Since pandemic times, SG14 has been brewing a number of proposals meant to make C++ «better for game developers».*

Things happening in SG14

- The abstract of this talk is as follows:
 - *« The C++ committee is made of a small number of working groups and of a larger number of study groups, including SG14 (low-latency, finances, games and embedded systems).*
 - *Since pandemic times, SG14 has been brewing a number of proposals meant to make C++ «better for game developers».*
 - *We will look at the principles behind this effort, the output of this work and what this means for C++. »*

Things happening in SG14

- The abstract of this talk is as follows:
 - *« The C++ committee is made of a small number of working groups and of a larger number of study groups, including SG14 (low-latency, finances, games and embedded systems).*
 - *Since pandemic times, SG14 has been brewing a number of proposals meant to make C++ «better for game developers».*
 - *We will look at the principles behind this effort, the output of this work and what this means for C++. »*

The overall project is described in P2966R1, a paper that is meant as a progress report of this effort, and that will be updated over time. This talk will look at some interesting details you will not find in P2966

What is SG14?

A diverse group of people with some shared technical interests

What is SG14?

- SG14 was « born »... with CppCon, in a sense

What is SG14?

- SG14 was « born »... with CppCon, in a sense
- Discussions between Michael Wong and members of the game development community during the early years of CppCon brought to light that members of that community thought the language was good, but could serve their needs better

What is SG14?

- It was quickly noted that the concerns raised by the game programming community had echoes in other communities
 - In particular, embedded systems programming as well as finance and high-frequency trading

What is SG14?

- It was quickly noted that the concerns raised by the game programming community had echoes in other communities
 - In particular, embedded systems programming as well as finance and high-frequency trading
- SG14 was founded under the title « SG14, Game Development & Low Latency »
 - Accompanying mention on isocpp.org: « Topics of interest to game developers and (other) low-latency programming requirements »
 - SG14 is often simply named the « low-latency study group »

What is SG14?

- SG14 holds regular meetings
 - Wednesdays, every month or so
 - The main topic of interest varies from meeting to meeting (games, finance, embedded, etc.)
 - Sub-chairs per group of interest help run some of the meetings

What is SG14?

- SG14 holds regular meetings
 - Wednesdays, every month or so
 - The main topic of interest varies from meeting to meeting (games, finance, embedded, etc.)
 - Sub-chairs per group of interest help run some of the meetings
- SG14 meetings also serve as an opportunity to poll the opinion of domain experts on ongoing or future proposals
 - There's a lot going on in WG21 and its 23 (to date!) study groups!

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::uninitialized_default_construct()`, `std::uninitialized_value_construct()`, `std::uninitialized_move()`, `std::destroy()`, etc.
 - Used by some who write their own containers, for example

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - Alternatives for `std::map` and `std::set` with different use-cases and different tradeoffs

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - `[[likely]]`, `[[unlikely]]`
 - Optimization hints to use with care

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - `[[likely]]`, `[[unlikely]]`
 - `[[no_unique_address]]`
 - Non-inheritance-based alternative to the empty base optimization in some cases

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - `[[likely]]`, `[[unlikely]]`
 - `[[no_unique_address]]`
 - Freestanding
 - Clarifying what freestanding C++ implementations and their libraries should conform to
 - Important for embedded systems development

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - `[[likely]]`, `[[unlikely]]`
 - `[[no_unique_address]]`
 - Freestanding
 - `std::hive`
 - A new container with different tradeoffs that can be useful for the use-cases of low-latency-oriented developers
 - Might be very close to adoption in the standard

What is SG14?

- There have already been some contributions to C++ due in part to SG14 efforts. For example:
 - Some raw memory algorithms from `<memory>`
 - `std::flat_map` and `std::flat_set`
 - `[[likely]]`, `[[unlikely]]`
 - `[[no_unique_address]]`
 - Freestanding
 - `std::hive`
 - ... and many, many others at various stages of progress

What is P2966?

How that paper came about, and where it seeks to lead us

What is P2966?

- Once upon a time...

What is P2966?

- Once upon a time...
 - E-mail contact from Nicolas Fleury (Ubisoft Montréal, occasional CppCon speaker)
 - This is his fault, he started the process 😊

What is P2966?

- Once upon a time...
 - E-mail contact from Nicolas Fleury (Ubisoft Montréal, occasional CppCon speaker)
 - Meeting in a café, discussing various things that could make C++ better

What is P2966?

- Once upon a time...
 - E-mail contact from Nicolas Fleury (Ubisoft Montréal, occasional CppCon speaker)
 - Meeting in a café, discussing various things that could make C++ better
 - Taking notes, synthesizing ideas, collating material, examples, use cases...

What is P2966?

- Once upon a time...
 - Time passes, covid occurs

What is P2966?

- Once upon a time...
 - Time passes, covid occurs
 - Professors work absurd number of hours...
 - Work progresses, but slowly...

What is P2966?

- Once upon a time...
 - Time passes, covid occurs
 - A proto-paper with no name is put together, then presented to SG14 in a very preliminary state

What is P2966?

- Once upon a time...
 - Time passes, covid occurs
 - A proto-paper with no name is put together, then presented to SG14 in a very preliminary state
 - Reactions mostly positive!
 - Ideas coming from the game industry, but seem fitting for the group

What is P2966?

- Once upon a time...
 - Meeting after meeting, SG14 examines some ideas in that proto-paper
 - Some are met with enthusiasm
 - Some require clarifications
 - Some revive previous SG14-related ideas
 - Some appear to be variations of other ideas progressing through non-SG14 proposals
 - Some are put aside (non-viable, other paths preferred, etc.)
 - Some are worked on, and in some cases workarounds are found in the existing language

What is P2966?

- For each suggestion, SG14 provided guidance
 - Is this feature something SG14 wants?
 - If the suggestion is to be pursued, should it be pursued on its own or as part of a related group?
 - Is this something that can be achieved with existing language facilities? If so, is it worthwhile to pursue the suggestion?
 - Are there alternative approaches that would be preferable?

What is P2966?

- For each suggestion, SG14 provided guidance
 - Is this feature something SG14 wants?
 - If the suggestion is to be pursued, should it be pursued on its own or as part of a related group?
 - Is this something that can be achieved with existing language facilities? If so, is it worthwhile to pursue the suggestion?
 - Are there alternative approaches that would be preferable?
- This led to questions being raised, requests being dropped, requests being modified, workarounds being identified, etc.
 - This discussion effort is still ongoing, but sufficient progress has been made that the production of actual papers can begin

What is P2966?

- The proto-paper still exists
 - It is not destined for publication
 - Instead, it is meant as a starting point for many of the numerous proto-proposals that were discussed
 - Examples, alternative designs, various notes and suggestions
 - A working document for the group, really
 - Hopefully, helpful to those who will write the actual proposals

What is P2966?

- P2966 is a progress report
 - It state the principles on which that overall effort is based
 - It lists the paths explored so far, and tries to identify those that are expected to be pursued, those that might be (uncertain), some that have been discussed but will probably not be...

What is P2966?

- P2966 is a progress report
 - It state the principles on which that overall effort is based
 - It lists the paths explored so far, and tries to identify those that are expected to be pursued, those that might be (uncertain), some that have been discussed but will probably not be...
- The expectation is that authors (mostly but not restricted to SG14) will now write individual proposals to make the various requests gain traction

A word on P2000 and P0592

How P2966 fits in the « overall plan for C++... »

A word on P2000 and P0592

- To help focus C++ evolution efforts, two notable papers were published (and revised a few times since)
 - Initially meant to help organize WG21 efforts towards C++23
 - Currently meant to help do so for C++26
- P2000 stems from the Direction group of WG21
 - Sets high-level priorities for C++
- P0592 aims to prioritize the work done by the working groups
 - Suggests a way to determine what to work on first, and what can wait

A word on P2000 and P0592

- P2966 is a recent paper
 - Groups together a variety of « small things »... that mean a lot to a sizeable part of the C++ programming community
 - Individual items might not fit in the overall directional papers' orientations
 - However, it is the hope of SG14 that, taken as a whole, the progress of the various efforts put forward in P2966 can be seen as beneficial for C++ and its users

A word on P2000 and P0592

- Like P2000 and P0592, the aim of P2966 is to be revised regularly and show the progression of the various efforts
 - Those that are promising
 - Those that have changed focus
 - Those that have made their way into the standard
 - Those that are dropped
 - The new demands from the community that SG14 has been made aware of

Underlying principles

What P2966 is built on

Underlying principles

- Contributors to this effort have committed to the following guiding principles

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good
 - Things that make C++ more teachable are good

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good
 - Things that make C++ more teachable are good
 - Avoid negative performance impacts

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good
 - Things that make C++ more teachable are good
 - Avoid negative performance impacts
 - Debugging matters

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good
 - Things that make C++ more teachable are good
 - Avoid negative performance impacts
 - Debugging matters
- Not all suggestions made throughout this process fall into the purview of one or more of these guiding principles, but they all aim not to contravene these principles

Underlying principles

- Contributors to this effort have committed to the following guiding principles
 - Things that simplify C++ are good
 - Things that make C++ more teachable are good
 - Avoid negative performance impacts
 - Debugging matters
- Not all suggestions made throughout this process fall into the purview of one or more of these guiding principles, but they all aim not to contravene these principles
 - These principles also influence SG14 reaction to proposals from other sources

Principle: things that simplify C++ are good

Principle: things that simplify C++ are good

- Some of the suggestions that stem from this effort aim to reduce the number of “gotchas” and pitfalls faced by C++ programmers and would reduce the number of workarounds and trickery involved in using C++ to write games
- Some specific remarks that have been made include:
 - Things that make generic programming simpler are appreciated
 - One needs to understand how the code will run based on the source code
 - Unexpected side-effects are to be avoided

Principle: things that make C++ more teachable are good

Principle: things that make C++ more teachable are good

- The game programming industry is big and turnover rates make training new colleagues something important
- Things that make C++ more teachable reduce costs, make professional insertion easier, and help reduce debugging efforts

Principle: avoid negative performance impacts

Principle: avoid negative performance impacts

- SG14 developers use C++ for many reasons, but control and performance characteristics are very high on the list
- Things with negative impact on performance are unacceptable to them
- New features that could impact performance negatively need to be accompanied with an opt-out mechanism

Principle: debugging matters

Principle: debugging matters

- For some of the suggestions, availability only in so-called « debug » builds would be acceptable
 - ...due to the costs expected in so-called « release » builds

Principle: debugging matters

- For some of the suggestions, availability only in so-called « debug » builds would be acceptable
 - ...due to the costs expected in so-called « release » builds
 - SG14 contributors know that the standard does not recognize this distinction but hope we can find a way to make some of the more costly features available in a conditional manner

Principle: debugging matters

- For some of the suggestions, availability only in so-called « debug » builds would be acceptable
 - Contributors have discussed the importance of manageable « Debug/ -O0 » builds as today, they sometimes need to debug code built with « Release/ -O2/ -O3 » builds to make their programs fit into memory

Principle: debugging matters

- Many have reported that design styles tend to change (monadic programming, functional programming, lazy execution) making it harder to grasp what's going on from the source code (it's « more magic »)

Principle: debugging matters

- Many have reported that design styles tend to change (monadic programming, functional programming, lazy execution) making it harder to grasp what's going on from the source code (it's « more magic »)
- Call stacks that are too deep make debugging harder

Actions

It's time to turn these ideas into proposals...

Actions

- SG14 contributors plan to write papers that will help these requests lead to adoption of corresponding features in the C++ language or its standard library

Actions

- SG14 contributors plan to write papers that will help these requests lead to adoption of corresponding features in the C++ language or its standard library
 - Some of these efforts are pursued through other papers, not necessarily SG14 related; in such cases, SG14 contributors will try to help bring these other papers to fruition

Actions

- SG14 contributors plan to write papers that will help these requests lead to adoption of corresponding features in the C++ language or its standard library
 - Some of these efforts are pursued through other papers, not necessarily SG14 related; in such cases, SG14 contributors will try to help bring these other papers to fruition
 - In the cases where the request can be served with existing facilities or that other approaches would serve the stated goals better, SG14 will try to make these existing alternatives better known

Actions

- SG14 contributors plan to write papers that will help these requests lead to adoption of corresponding features in the C++ language or its standard library
 - Some of these efforts are pursued through other papers, not necessarily SG14 related; in such cases, SG14 contributors will try to help bring these other papers to fruition
 - In the cases where the request can be served with existing facilities or that other approaches would serve the stated goals better, SG14 will try to make these existing alternatives better known
 - Where there are competing proposals to solve a problem from the set below, SG14 will try to provide guidance with respect to those aspects of these proposals that serve best this community

P2966 requests

What are those things that should become actual proposals?

P2966 requests

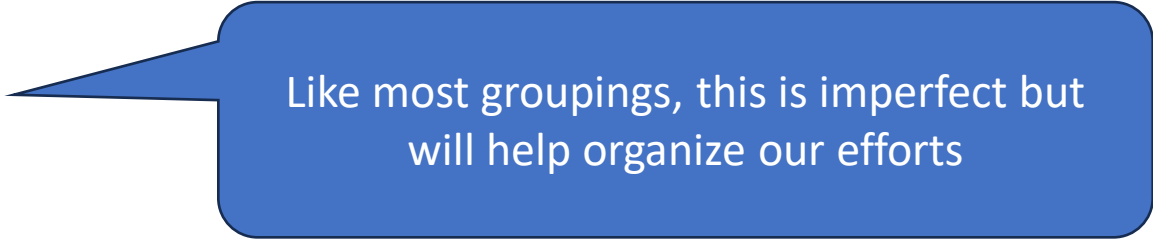
- Requests from P2966 can be grouped in categories

P2966 requests

- Requests from P2966 can be grouped in categories
 - Compile-Time Computing
 - Memory Allocation and Deterministic Behavior
 - Attributes
 - Move Semantics
 - Handling Disappointment
 - Pattern Matching
 - Tooling and Ease-of-Coding
 - Networking
 - Parallel and Concurrent Computing
 - Logging and I/O
 - Numeric Computing
 - Miscellaneous

P2966 requests

- Requests from P2966 can be grouped in categories
 - Compile-Time Computing
 - Memory Allocation and Deterministic Behavior
 - Attributes
 - Move Semantics
 - Handling Disappointment
 - Pattern Matching
 - Tooling and Ease-of-Coding
 - Networking
 - Parallel and Concurrent Computing
 - Logging and I/O
 - Numeric Computing
 - Miscellaneous



Like most groupings, this is imperfect but will help organize our efforts

P2966 requests

- Requests from P2966 can be grouped in categories
 - Neither the categories nor the requests therein form a closed set
 - Revisions of P2966 will aim to inform the community of additions, modifications and removals to either categories of requests

Compile-time computing

P2966 requests: compile-time computing

- Allow overloading based on constexpr arguments
 - Has been discussed in the past (through proposals, e.g.: P1045)
 - There are workarounds in some cases
 - Design space to be explored

P2966 requests: compile-time computing

- Allow overloading based on constexpr arguments

- Motivating example:

```
template <class T>
    string MyFormat(constexpr const char*, T&&); // A
template <class T>
    string MyFormat(const char*, T&&);           // B
// ...
MyFormat("Some format {:d}", someArg); // calls A
MyFormat(RuntimeFmtString(), someArg);  // calls B
```

P2966 requests: compile-time computing

- Allow overloading based on constexpr arguments

- Motivating example:

```
class MyString {  
    // ...  
public:  
    // A (e.g.: stores pointer directly)  
    MyString(constexpr const char*);  
    // B, uses normal code path (SSO, heap, etc.)  
    MyString(const char*);  
};  
MyString s{ "some str" };    // ctor A  
MyString s{ RuntimeStr() }; // ctor B
```

P2966 requests: compile-time computing

- Allow overloading based on constexpr arguments

- Motivating example:

```
class CommandLineArguments {  
    // ...  
public:  
    // first argument only compiles with string literal,  
    // object only stores pointer  
    void add(constexpr const char*, char, void(*)());  
    // ...  
};  
// ...  
CommandLineArguments cmd;  
cmd.add("help", 'h', &someFunc);  
// cmd.add(RuntimeString(), 'x', &otherFunc); // doesn't compile
```

P2966 requests: compile-time computing

- Static reflection
 - General support from SG14 (work underway in SG7)
 - Willingness to contribute to make this better known to the community
 - Willingness to offer guidance when there are competing proposals

P2966 requests: compile-time computing

- Static reflection
 - General support from SG14 (work underway in SG7)
 - Willingness to contribute to make this better known to the community
 - Willingness to offer guidance when there are competing proposals
 - Some features particularly desired (design spaces to be explored)
 - `std::variable_name()`
 - `std::declaration_source_location()`
 - Reflection on enums (how many symbols are there? Are the values consecutive? A checked cast to the underlying type...)
 - Reflection on class and struct types, including iteration over members

P2966 requests: compile-time computing

- Compile-time string interpolation

- Progress is ongoing already, e.g.: P2741 but more is needed
- Example ('\$' is strawman syntax):

```
template <int N>
    struct Facto {
        // Ok: N known at compile-time
        static_assert(N >= 0, $"{N} is negative");
        enum : unsigned long long { value = N * Facto<N-1>::value };
    };
template <>
    struct Facto<0> {
        enum : unsigned long long { value = 1ULL };
    };
```

- Runtime string interpolation seen as useful too

Memory allocation and deterministic behavior

P2966 requests: memory allocation and deterministic behavior

- SOO Thresholds

- Exposing the SOO thresholds at compile-time to allow for designs that avoid dynamic memory allocation
- Example (strawman syntax):

```
template <class F> std::function<void()> make_func(F f) {  
    // only compiles if construction of a function<void()>  
    // from an object of type F would not allocate  
    static_assert(  
        sizeof(F) <= soo_max_size_v<std::function<void()>>  
    );  
    return { f };  
}
```

P2966 requests: memory allocation and deterministic behavior

- SOO Thresholds

- Exposing the SOO thresholds at compile-time to allow for designs that avoid dynamic memory allocation
- Example (strawman syntax):

```
template <class F> auto make_func(F f) {  
    // returns a function<void()> if one can  
    //construct it without allocating; fallback  
    // on a homemade "plan B" otherwise  
    if constexpr  
        (sizeof(F) <= soo_threshold_v<std::function<void()>>)  
        return std::function<void()>{ f };  
    else  
        return my::inplace_function<void()>{ f }; // see below  
}
```

P2966 requests: memory allocation and deterministic behavior

- `std::inplace_function`
 - `std::function` can allocate if constructed from a function object of a size greater than an implementation-specific threshold
 - Some companies reject that type outright and roll out their own homemade version
 - For this reason, a `std::inplace_function` or equivalent, which never allocates, is desired
 - Has been discussed by SG14 in the past
 - There is implementation experience
 - Candidate for freestanding

P2966 requests: memory allocation and deterministic behavior

- SOO-Enabled vector
 - A `std::vector<T>/std::array<T,N>` alternative that has a (potentially compile-time known) capacity and never allocates
 - Might already be solved (verifications to be made)
 - P0843 (`inplace_vector<T>`) approved in Varna

P2966 requests: memory allocation and deterministic behavior

- External Buffer Vector
 - A vector that manages an externally provided buffer and switches to heap-allocated memory should that buffer's capacity not be sufficient
 - The use-case might be solved by PMR vector and `monotonic_buffer_resource` (to verify)

P2966 requests: memory allocation and deterministic behavior

- Intrusive Containers
 - Some SG14 contributors say they use these containers extensively, particularly `intrusive_list<T>`
 - P0406 proposed in the past; see if it is still adequate and should be revived or if a different design would be preferable

P2966 requests: memory allocation and deterministic behavior

- InplaceContainer<Size> Inheriting from Container Pattern
 - A set of containers (e.g.: `inplace_vector<T,Sz>`) that expose the same interface as standard containers but supply a fixed-size buffer to manage by default
 - Might be solved through PMR containers (to verify)

P2966 requests: memory allocation and deterministic behavior

- Heap-Free Functions
 - Add heap-free options to all situations that might lead to dynamic memory allocation (e.g.: passing client-allocated buffers)
 - In some cases, that might simply be a matter of adding a function overload taking an array of `std::byte` as argument
 - Applies to standard library containers as mentioned in other items of P2966
 - Candidates for Freestanding

P2966 requests: memory allocation and deterministic behavior

- “No RTTI” guarantees
 - Many SG14 contributors compile with RTTI turned off but might still want to use PMR allocators
 - Some vendor implementations have been reported to use `dynamic_cast` in their PMR types
 - Offering PMR with a “no-RTTI” guarantee, or at least a compile-time checkable guarantee would be desirable

P2966 requests: memory allocation and deterministic behavior

- “Predictable lambdas”
 - Being able to declare a lambda on the stack, without initializing it right away, and having access to its constructor
 - Some sort of placement new on an uninitialized lambda, kind of like an `optional<lambda>`

P2966 requests: memory allocation and deterministic behavior

- Move-With-Last-Swap / Reorderase
 - The technique of object removal from a container where order isn't important by swapping to the end and erasing there is used extensively by SG14 contributors
 - Empirical data shows a standard library solution would be advantageous

Attributes

P2966 requests: attributes

- Support for User Attributes
 - Allowing users to implement their own attributes to replace macro-based tricks frequently found in game engines with something “in-language”

P2966 requests: attributes

- `[[invalidate_dereferencing]]`
 - Annotate the pointer argument passed to `realloc()` and similar functions with `[[invalidate_dereferencing]]`
 - Intent: the compiler should consider `*ptr` or `p->` to be invalid after the call but using `ptr` without dereferencing would still be valid
 - Would fix what some consider to be a “UB pitfall” with `realloc()`, while providing an attribute usable for user code wanting the same optimization opportunities and semantics
 - The desired result is a compile-time error
 - Note: this is currently QoI
 - Note: even if standard library functions might do this, the intent is to allow user code to convey this meaning too

P2966 requests: attributes

- `[[invalidate]]`
 - Annotate the pointer argument passed to `free()` and similar functions with `[[invalidate]]`
 - Intent: the compiler should consider `ptr`, `*ptr` and `ptr->` to be invalid after the call
 - This would address what some consider to be “UB pitfalls” at compile-time, while providing an attribute usable for user code wanting the same optimization opportunities and semantics
 - There is implementation experience
 - The desired result is a compile-time error
 - Note: this is currently QoI
 - Note: even if standard library functions might do this, the intent is to allow user code to convey this meaning too

P2966 requests: attributes

- `[[simd]]`
 - Not pursued as such
 - Work ongoing for the Parallelism TS
 - P1928 was approved in Varna for C++26

P2966 requests: attributes

- `[[no_copy]]`
 - Annotate types and function arguments with `[[no_copy]]` if only move and RVO are acceptable. Example:

```
[[no_copy]] SomeContainer<SomeType> Foo();  
[[no_copy]] SomeType Bar();  
// SomeType st; // default ctor  
// st = Bar(); // assignment (would be rejected)  
SomeType st = Bar();
```
 - Workarounds exist, but...
 - Type definition and function code can evolve over time, making the guarantee at the function level valuable.
 - Note: part of the intent is to help junior programmers who might not understand the intricacies of C++ value categories

P2966 requests: attributes

- `[[rvo]]`
 - Annotate functions with `[[rvo]]` to ensure calls to these functions only compile if used in a RVO situation
 - Analogous to `[[no_copy]]`
 - There might be a basis in P2025 and in Clang's non-standard `[[musttail]]` attribute
 - Example:

```
[[rvo]] X f();  
// ...  
auto x0 = f(); // Ok  
X x1;  
// x1 = f(); // not Ok
```

P2966 requests: attributes

- `[[side_effect_free]]`
 - Annotate functions with `[[side_effect_free]]` and make this checkable at compile-time
 - Intent: open up optimization opportunities such as automatic memorization
 - Prior work includes `[[pure]]` proposals and the `[[conveyor]]` suggestion for contracts
 - There is implementation experience

P2966 requests: attributes

- `[[trivially_relocatable]]`
 - SG14 contributors have expressed strong interest in a `[[trivially_relocatable]]` attribute such as the one in P1144
 - Note: some companies have their own `is_memcopyable` trait to simulate `[[relocatable]]`.

Move semantics

P2966 requests: move semantics

- Move semantics are perceived as important but too easy to misuse
 - Make it so a function taking `const T&&` as argument fails to compile or is warned about
 - Too easy to write such a signature by copy-pasting from a copy constructor / copy assignment
 - Reported as a pain point by numerous contributors.

P2966 requests: move semantics

- Common mistake is typing:

```
MyClass(const MyClass& other)
    : m_Member(other.m_Member) {
    // ...
}
MyClass(const MyClass&& other)
    : m_Member(std::move(other.m_Member)) {
    // ...
}
```

- instead of:

```
MyClass(const MyClass& other)
    : m_Member(other.m_Member) {
    // ...
}
MyClass(MyClass&& other)
    : m_Member(std::move(other.m_Member)) {
    // ...
}
```

Handling disappointment

P2966 requests: handling disappointment

- So-called « Herbceptions » are looked upon favorably
- Quoting a contributor: « [handling disappointment] is a major and multi-faceted issue that requires a paper on its own »

Pattern matching

P2966 requests: pattern matching

- The switch-case style pattern matching (inspect) is looked upon favorably
 - General support from SG14 (work underway in other groups)
 - Willingness to contribute to make this better known to the community
 - Willingness to offer guidance when there are competing proposals

Tooling and ease-of-coding

P2966 requests: tooling and ease-of-coding

- nameof operator
 - See the nameof operator in C# for inspiration
 - See `#[derive(Debug)]` in Rust for inspiration
 - Also <https://github.com/Neargye/nameof>
 - Note: might be solved by SG7 efforts

P2966 requests: tooling and ease-of-coding

- Better compile-time error detection
 - General wish for things that will help compiler catch more errors at compile-time
 - Hope that concepts will play a role
 - Clarifying what the compiler “sees” and what it does not “see” would help wrote more “debuggable” code
 - Design space to be explored in order to clarify the contours of what is requested

P2966 requests: tooling and ease-of-coding

- Conditional compilation
 - Something that would replace `#ifdef ... #endif` and would allow one's code to be checked for one platform while compiling for another
 - Pain point for individuals writing code for multiple platforms

Networking

P2966 requests: networking

- A small, fast and low-level layer including sockets
 - Networking is something every game engine implements by itself
 - A C++ standard library version would be seen as something useful
 - Boost ASIO seems “heavy” to most SG14 contributors, but a replacement for C sockets would be a huge win
 - Games would probably use the low-level standard library API for networking and use their own mechanisms on top of it, including their own asynchronous utilities

P2966 requests: networking

- A small, fast and low-level layer including sockets
 - Note: there have been discussions in WG21 as to whether it would be reasonable to provide a basic sockets replacement for C++ would be useful given all of the security concerns we have today
 - For games, the answer to this would be “yes”. Not everyone needs security; some people just need fast and low-level
 - For embedded, a small and fast low-level interface would be most important
 - There’s a stack most SG14 contributors use: <https://en.wikipedia.org/wiki/LwIP>

Parallel and concurrent computing

P2966 requests: parallel and concurrent computing

- Compile-time Evaluated Thread-Safety
 - Allow enforcing Rust-inspired resource management in order to help validation non-thread-safe operations at compile-time
 - There seems to be prior art in Clang's thread safety analysis

P2966 requests: parallel and concurrent computing

- Naming, tracing and debugging
 - Adding facilities to portably name mutexes and threads
 - Work is ongoing in P2019

P2966 requests: parallel and concurrent computing

- Support of almost portable facilities
 - Adding facilities to control thread priority and stack size
 - There has been work already, see P0484, P0320, P2019
 - Some game engines use `alloca()` and similar features in somewhat scary ways; investigate how to constrain these facilities

Logging and I/O

P2966 requests: logging and I/O

- Better logging facilities
 - Some languages have optional attributes to know “who called you” which can be useful for logging
 - `std::stacktrace` will help
 - Static reflection will help

Numeric computing

P2966 requests: numeric computing

- Linear algebra
 - SG14 supports the addition of foundational types for linear algebra
 - Efforts are ongoing in that respect
 - Each game engine has its own version of such utilities, and so does each middleware, but there seems to be “holes” in most of them
 - In general, it would be good if what can be done in a language such as HLSL could be done directly in C++

P2966 requests: numeric computing

- Opt-in UB on unsigned overflow
 - There is a need for an integral type (at least the 32 bits flavor) for which overflow would be UB
 - Most game companies use their own aliases for types
 - Intent: to change that alias from a “classic” unsigned integral where overflow is defined to this new type, to see if performance gains could be achieved

Miscellaneous

P2966 requests: miscellaneous

- Forward Class Declarations with Inheritance

- Allow a forward class declaration specifying inheritance relationships when using pointer-to-base / pointer-to-derived conversions

- Example:

```
class X : public Y;  
Y * f();  
X * g();  
X * h(bool b) { return b? f() : g(); }
```

P2966 requests: miscellaneous

- “namespace class”
 - When defining a class’ member functions in a .cpp file, repeating the class name everywhere can get tedious
 - Would be nice to « reduce the noise »

P2966 requests: miscellaneous

- “namespace class”

- Example before

```
class X {  
    static const std::string S;  
public:  
    using type = int;  
    X(type);  
    type f() const;  
};  
// ... in the .cpp file  
const std::string X::S = "...";  
X::X(type) {  
}  
X::type X::f() const { // or auto X::f() const -> type  
    return {};  
}
```

P2966 requests: miscellaneous

- “namespace class”

- Example after (strawman syntax)

```
class X {  
    static const std::string S;  
public:  
    using type = int;  
    X(type);  
    type f() const;  
};  
// ... in the .cpp file  
namespace class X {  
    const std::string S = "...";  
    X(type) { }  
    type f() const { // or auto f() const -> type  
        return {};  
    }  
}
```


P2966 requests: miscellaneous

- Constrained Construction
 - A syntax that would constrain the number of constructors involved at the call site
 - Example:
`construct(1) auto a = f();`
 - Might help protecting against performance losses resulting from unwanted conversions
 - Design space to be explored

P2966 requests: miscellaneous

- Constrained Construction

- A syntax that would constrain the number of constructors involved at the call site

- Example

constr

- Might help with conversion

- Design

“Simply put, we’re looking for ways to protect some important performance properties of our code against accidental upstream modifications. We want compilation to fail at the call site if someone (employee or vendor) accidentally modified a class somewhere that suddenly causes extra copies and allocations to happen. Otherwise small problems like this pile up and become big problems that are only caught much later when profiling, and we lost the context for the problematic change”

nted

P2966 requests: miscellaneous

- Flags-Only enums
 - Enumerations that can only be flags, which could influence “stringification”, particularly if two symbols have the same value
 - Note: workarounds have been proposed in the past (see P2966 for a list)

P2966 requests: miscellaneous

- Member Functions of Enums
 - Of particular interest would be conversion operators

P2966 requests: miscellaneous

- Better Support of Arrays with enum-Based Strong Types
 - Enum-based strong types and arrays mix unpleasantly, which blocks their adoption in some companies
 - Example (<https://wandbox.org/permlink/dZvsd4MTz3WD7282>):

```
int main() {  
    using namespace std;  
    [[maybe_unused]] byte b0{ 0 }; // ok  
    // byte b1[]{ 0, 0 }; // nope  
    [[maybe_unused]] byte b1[2]{ }; // ok  
    [[maybe_unused]] array<byte,1> b2; // ok  
    // array<byte,1> b3{ 0 }; // not ok  
    [[maybe_unused]] array<byte,1> b4{ byte{} }; // ok  
    [[maybe_unused]] array<byte,1> b5{ {} }; // ok  
}
```

P2966 requests: miscellaneous

- Making `std::initializer_list` Movable
 - Would allow such things as initializing a `std::vector<std::unique_ptr<T>>` with a pair of braces containing a sequence of calls to `std::make_unique<T>()`
 - Prior efforts include P0065

P2966 requests: miscellaneous

- Explicit list-initialization
 - `initializer_list` has many rough edges
 - The `vector<int> v(10,-1)` vs `vector<int>v{ 10, -1 }` is a recurring problem
 - SG14 contributors have suggested never using braces unless one really wants list initialization
 - Hard to enforce over a large codebase

P2966 requests: miscellaneous

- Explicit list-initialization
 - To fix performance regressions introduced by `initializer_list`, some contributors use the following instead

```
template<class T>
struct explicit_init_list {
    std::initializer_list<T> lst;
    constexpr explicit_init_list(std::initializer_list<T> lst)
        : lst(lst) {}
    constexpr const T* begin() const { return lst.begin(); }
    constexpr const T* end() const { return lst.end(); }
    constexpr std::size_t size() const { return lst.size(); }
};
```


P2966 requests: miscellaneous

- Efficient Downcasting
 - Need for a way to downcast to the most-derived type at low cost, e.g.: using sorted vtables for statically linked .exe
 - Companies write their own currently but it's nonportable
 - Quoting: *"We can achieve assembly code limited to a load of a vtable, but vtables have to be sorted beforehand so it has to be wired in the compiler to be portable (we write it per-compiler)"*

P2966 requests: miscellaneous

- Covariant Cloning
 - Being able to have covariant return types based on `unique_ptr<T>` as well as on `T*`
 - No need for `shared_ptr<T>` support has been reported yet
 - There has been prior work in that regard

P2966 requests: miscellaneous

- Homogeneous Variadics

- Direct support for the following:

```
template <class T> void f(T...) { /* ... */ }
```

- There are workarounds:

- Using `std::conjunction`
 - Using concepts with a `requires (same_as<T, Ts> && ...)`

P2966 requests: miscellaneous

- Named Arguments
 - A recurring request
 - Prior efforts exist, e.g.: N4172
 - There are workarounds with designated initializers

P2966 requests: miscellaneous

- SoA to AoS
 - Arrays of structs (AoS) make it easier to understand and structure classes but are often less efficient in terms of time and space usage than structs of arrays (SoA)
 - This is well-known to the game development community
 - Lots of empirical data to that effect
 - A way to “transform” something expressed as an AoS into its SoA equivalent, it would be an appreciated feature of the language
 - Alternative: distinguishing « presentation » layout from actual layout
 - Very warm reception by SG14 contributors
 - An example of strawman syntax to simplify discussion would help

P2966 requests: miscellaneous

- Unified Function Call Syntax
 - Tooling and ease of use are motivating factors
 - Current code editors tend to be better at assisting programmers with `x.f(y)` than they are with `f(x, y)`
 - There have also been reports that free functions tend to be coded two or three times separately
 - Programmers don't always find them, and end up rolling their own

P2966 requests: miscellaneous

- Adoption of Modules
 - C++ modules can't come soon enough
 - Make templates a lot more palatable
 - Get us rid of the infamous Unity builds which break the concept of a translation unit but are ubiquitous in large game codebases to get reasonable compilation speeds
 - SG14 seeks to speed up adoption of modules, or at least standardize how hybrid header + modules setups work to facilitate transition

P2966: now what?

Where do we go from here?

P2966: now what?

- The overall picture described in P2966 is wide and no single individual can write all those proposals
 - Current contributors have pledged to participate, of course
- Taken as a group, SG14 thinks that many (maybe most) of these requests could lead to beneficial improvements for the C++ programming community overall

P2966: now what?

- What is needed now is:
 - Interested authors or co-authors for proposals targeting these requests
 - Exploration of design space
 - Willingness to publish these proposals
 - There can be champions if authors cannot make it to WG21 meetings
 - Willingness to address committee feedback and guidance and make these requests evolve

P2966: now what?

- SG14 is also interested in ways to make P2966 evolve
 - Workarounds or solution to some of the problems exposed therein using the existing language or library?
 - Demonstration that some of these requests should be altered or otherwise not pursued?
 - Suggestions for ways to improve or better define these requests
 - etc.
- ...please, in writing, such that these ideas can be tracked

Come and contribute!

Thank you!