

C++ Club UK

Gleb Dolgich

2019-10-10

CppCon 2019 Trip Reports

- Matt Godbolt

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

- Video: <https://youtu.be/ARYP83yNAWk>
 - Reddit: https://www.reddit.com/r/cpp/comments/d87plg/cppcon_2019_herb_sutter_defragmenting_c_making/

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

The slide features two large triangles side-by-side. The left triangle, labeled 'Historical', is blue and oriented downwards, with the words 'add things' at the top, 'fix things' in the middle, and 'simplify' at the bottom. The right triangle, labeled 'A future worth considering?', is yellow and oriented upwards, with the words 'add things', 'fix things', and 'simplify' all pointing towards the apex. A small number '5' is located in the bottom right corner of the slide area.

C++'s evolution priorities

Historical

add things

fix things

simplify

A future worth considering?

5

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:
ansatz

Cppcon | 2019
The C++ Conference
cppcon.org

Herb Sutter

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

A tale of two -compiler /switches

We ❤️ exceptions

```
-fno-exceptions  
#define _HAS_EXCEPTIONS 0
```

We ❤️ RTTI

```
-fno-rtti  
/GR-
```

The logo for CppCon 2019, featuring a stylized orange circle with a white plus sign inside, followed by the text "Cppcon | 2019" and "The C++ Conference" below it.

A photograph of Herb Sutter, a man with glasses and a patterned shirt, speaking at a podium. The background is a repeating CppCon logo. A black banner in front of him displays his name, "Herb Sutter".

7
De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Program-recoverable errors

error: "an act that ... fails to achieve what should be done."

— [Merriam-Webster]

- ▶ P0709: “recoverable error” \equiv “a function couldn’t do what it advertised.”
 - ▶ Its preconditions were met.
 - ▶ It could not achieve its successful-return postconditions.
 - ▶ The calling code can be told and can programmatically recover.
- ▶ Run-time errors (and only those) should be reported to the calling **code**.
 - ▶ Regardless of mechanism: “Prefer exceptions” but applies to any reporting style.



A photograph of Herb Sutter, a man with glasses and a patterned shirt, speaking on stage at CppCon 2019. He is gesturing with his right hand. The background features a repeating logo for CppCon 2019. A black banner at the bottom right of the photo area contains the text: "Herb Sutter", "De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable", and "(“Simplifying C++” #6 of N)".

25

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Abstract machine corruption ≠ recoverable error

- ▶ Abstract machine corruption causes a corrupted state that cannot be recovered from programmatically.
- ▶ So it should never be reported to calling code as an error (e.g., via exception).
- ▶ Example: **Stack exhaustion** is always an abstract machine corruption.
 - ▶ It can happen to any function.
 - ⇒ If we tried to report it using an exception then every function could throw.
 - ▶ We cannot continue running normal code. (**NB:** Destructors are “normal code.”)
 - ⇒ The callee can’t run code to report it to the caller...
 - ... and the caller couldn’t run code to recover anyway.
 - ▶ Conclusion: Reporting this as a runtime error would be a category error.



26

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Programming bug ≠ recoverable error

- ▶ A programming bug (e.g., out-of-bounds access, null dereference) causes a corrupted state that cannot be recovered from programmatically.
 - ▶ Therefore it should never be reported to the calling code as an error (e.g., it should not be reported via an exception).
- ▶ Examples:
 - ▶ A **precondition** (e.g., [[pre...]]) violation is always a bug in the caller (it shouldn't make the call).
 - ▶ Corollary: std::logic_error and its derivatives should never be thrown (§4.2), its existence is itself a “logic error”; use assertions/contracts/... instead.
 - ▶ A **postcondition** (e.g., [[post...]]) violation on “success” return is always a bug in the callee (it shouldn't return success).
 - ▶ Violating a noexcept declaration is also a form of postcondition violation.
 - ▶ An **assertion** (e.g., [[assert...]]) failure is always a bug in the function.



27

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Taxonomy

	What to use	Report-to handler	Handler species
A. Corruption of the abstract machine (e.g., stack exhaustion)	Terminate	User	Human
B. Programming bug (e.g., precondition violation)	Asserts, log checks, programmer contracts, ...	Programmer	Human
C. Recoverable error (e.g., host not found)	Throw exception, error code, etc.	Calling code	Code



28

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Part 1: Exception Handling (EH)

- ▶ Establishing the problem: Today's EH violates the zero-overhead principle
 - “I can't afford to enable exception handling” ⇒ paying for what you don't use
 - “I can't afford to throw an exception” ⇒ can write it more efficiently by hand
 - Bonus “I can't throw through this code” ⇒ lack of control, invisible vs. automatic propagation
- ▶ Key definition: What is a “recoverable error”?
 - Recoverable error != programming bug != abstract machine corruption
 - Exceptions/codes != pre/post contracts != stack and heap overflow
- ▶ Four coordinated proposals
 1. Enable zero-overhead exception handling
 - 2&3. Throw fewer exceptions (~90% of all exceptions should not be)
 4. Support explicit “try” for visible propagation



29

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Core issues: Zero-overhead + **determinism**

- ▶ Exceptions are great: Distinct “error” paths, can’t ignore, auto propagation.
- ▶ But: Inherently not zero-overhead, not deterministic.
- ▶ **“Throwing objects of dynamic types... ... and catching using RTTI.”** ⇒ dynamic allocation + type erasure
⇒ dynamic casting (special)

- ▶ **Proposal:**
 - ▶ **“Throwing values of static types... ... and catching by value.”** ⇒ stack allocation, share return channel
⇒ no dynamic casting, just value comparison
 - ▶ Isomorphic to error codes, identical space/time overhead and **predictability**.
 - ▶ Share return channel ⇒ potential for negative overhead abstraction.
 - ▶ If a function agrees (**opts in**) that any exceptions it emits are **values of one statically known type**, we can implement it with zero dynamic/non-local overheads.



30

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

1. Throw values, not types

- ▶ As-if returning `union{ Success; Error; } + bool`, using the same return channel (incl. registers + CPU flag for discriminant).
- ▶ Best of exceptions and error codes (and fully prior-art):
 - Exactly exceptions' programming model (`throw, try, catch`).
 - Exactly error codes' return-value implementation (w/o monopolizing channel).
- ▶ Doubles down on **value semantics**. (Cf: C++11 move semantics.)
- ▶ If you love:
 - ▶ Exceptions: Can use them more widely, removing perf reasons to avoid/ban.
 - ▶ Expected/Outcome: Gets language support, propagates automatically.
 - ▶ Error codes: Doesn't monopolize return channel, propagates automatically, and the caller can't forget to check it and gets distinct success/error paths.
 - ▶ Termination (fail-fast): Hook the propagation notification (see §4.1.4).



31

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Core proposal summary

- A *static-exception-specification* `throws` ⇒ function can throw `std::error`, an evolution of `std::error_code` + SG14-driven improvements already underway.

```
string f() throws {  
    if (flip_a_coin()) throw arithmetic_error::something;  
    return "xyzzy"s + "plover";           // bad_alloc → std::errc::ENOMEM  
}  
  
string g() throws { return f() + "plugh"; }      // bad_alloc → std::errc::ENOMEM  
  
int main() {  
    try {  
        auto result = g();  
        cout << "success, result is: " << result;  
    } catch(error err) {  
        cout << "failed, error is: " << err.error();  
    }  
}
```



32

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:



CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

Core proposal summary

- A *static-exception-specification* `throws` ⇒ function can throw `std::error`, an evolution of `std::error_code` + SG14-driven improvements already underway.

```
string f() throw
    if (flip_a_coin)
        return "xyzzy"
}
string g() throw
int main() {
    try {
        auto result
        cout << "st"
    } catch(error)
        cout << "fa"
    }
}
```

Default and recommended std::error usage == purely local return values:

- Always allocated as an ordinary stack value
- Share (not waste) the return channel
- Statically known type, so never need RTTI

Zero-overhead: No extra static overhead in the binary image.
No dynamic allocation. No need for RTTI.

Determinism: Identical space and time cost as if returning an error code by hand.

Note: For compatibility, std::error can also wrap an exception_ptr, but this is a compatibility mode where the overheads come from using today's model, which are just passed through



33

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”



1. Simplifications

- ▶ What are the benefits?
 - ▶ Unification: All projects can turn on exception handling.
 - ▶ Zero overhead principle, part 1: “Don’t pay for what you don’t use.”
 - ▶ Unification: All code can report errors using exceptions.
 - ▶ Zero overhead principle, part 2: “When you do use it you can’t reasonably write it better by hand” including by using alternatives.
 - ▶ Even space- and time-constrained code that need statically boundable costs.
- ▶ **Simplification:** Can teach “every function should be declared with exactly one of noexcept or throws.”
 - ▶ Just like we now can teach “every virtual function should be declared with exactly one of virtual, override, or final.”

35

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

3. Allocation failure: Let the allocator decide

- ▶ Cologne (2019-07) Library Evolution subgroup (LEWG) direction:
 - ▶ Add a noexcept-queryable allocator property for “reports vs. fails-fast” on allocation failure. ([Unanimous](#))
 - ▶ Recommend conditional noexcept based on the relevant allocator) pervasively on standard library functions that could only throw bad_alloc. ([Unanimous](#))
 - ▶ Fail-fast for default std::allocator and default operator new. ([19](#) [11](#) [4](#) [1](#) [0](#))
 - ▶ **NB: Even though a breaking change.**
- ▶ Cologne (2019-07) Language Evolution subgroup (EWG) direction:
 - ▶ Fail-fast for default std::allocator and default operator new. ([0](#) [17](#) [6](#) [10](#) [5](#))
 - ▶ **NB: Even though a breaking change.**
 - ▶ Updated proposal: Let the programmer decide for global new.



Herb Sutter, a man with glasses and a patterned shirt, is speaking on stage at CppCon 2019. The stage backdrop features the CppCon logo and the year 2019. A large screen to his right displays the title of his talk: "De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable ("Simplifying C++" #6 of N)". Below the title, it says "Video Sponsorship Provided By: ansatz".

42

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:
ansatz

CppCon 2019: Herb Sutter “De-fragmenting C++: Making Exceptions and RTTI More Affordable and Usable”

4. Proposed extension: *try* expressions

- ▶ Good news: Exceptional control flow is **automatic**.
- ▶ Bad news: Exception control flow is **invisible**.
 - ▶ Hard to reason about exceptions, especially in legacy code.
- ▶ Proposal: **try** before an expression/statement where a subexpression can throw.
 - ▶ Makes exceptional paths visible.
 - ▶ If we required it in new code: **Compile-time guarantees** (e.g., no “try/throw” \Rightarrow noexcept).

```
string f() throws {
    if (flip_a_coin()) throw arithmetic_error::something;
    return try "xyzzy's + "plover";           // greppable
              try string s("xyzzy");           // same, just showing statement form too
              try return s + "plover";
}
string g() throws { return try f() + "plugh"; }
```



46

De-fragmenting C++:
Making Exceptions and RTTI
More Affordable and Usable
("Simplifying C++" #6 of N)

Video Sponsorship Provided By:

ansatz

Arthur O'Dwyer "Thoughts on Herb Sutter's CppCon keynote"

<https://quuxplusone.github.io/blog/2019/09/24/thoughts-on-eh/>

Reddit: https://www.reddit.com/r/cpp/comments/d8p08f/thoughts_on_herb_sutters_cppcon_keynote/



Tony Van Eerd @tvaneerd

```
try {
    return vec.at(index);
}
catch (std::out_of_range const &) {
    return -1;
}
```

I just saw this in a code review and was going to claim it inefficient, but darn [@CompileExplore](#) proved me wrong again. (IIRAC*) All compilers turn it into an if check on the bounds.

*Read ASM

11h • 11/07/2019 • 22:56



Cologne Trip Report

https://www.silexica.com/news/iso_cpp_meeting_2019/

- P1413, Deprecate std::aligned_storage and std::aligned_union
- P1609, C++ Should Support Just-in-Time Compilation
- P1727, Issues with current flat_map proposal
- P0443r10, A Unified Executors Proposal for C++
- P1660r0, A Compromise Executor Design Sketch
- “C++ Tooling Ecosystem” Technical Report
- P1679r0, String Contains function

Quote

Fred Brooks:

*Much of the essence of building a program is in fact
the debugging of the specification.*