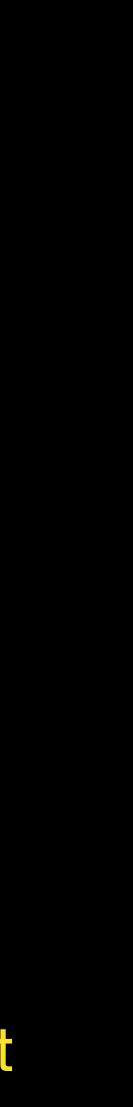
P3517R0 **Trivial Relocation for C++26** Library Design for Wrocław

Alisdair Meredith, Bloomberg, 2024 November 20

What is in the Proposal? How P2786 updates the standard library

- Trivial relocation specified in the core language
 - Bitwise moving of objects
- Library APIs to detect and use trivial relocation
- Replaceability specified in the core language
 - Consistency between construction and assignment
- Library APIs to detect and use replaceability
- Library front matter update to allow Qol use of trivial relocation and replacement



What is no longer in the Proposal? Changes since Monday to bring more consensus

- Core language proposal that passed EWG on Monday is not touched 0
 - Apart from reverting the spelling of the keywords
- Library interface has been cut down to the bare minimum
 - We are committed to providing several options for a consumer API next meeting
- We specifically removed:
 - The "simple" relocate function
 - The swap value representation function
 - All talk about optimizing swap now deferred to Qol

Relocation and Trivial Relocation

How Does P2786 Support Trivial Relocation? Syntax and library APIs

- Define core notion of trivial relocatability
- Deduce whether type is trivially relocatable if is has no user-supplied move constructor, move assignment operator, nor destructor
 - Use a keyword to deduce otherwise
- Provide a type trait to report if a type is trivially relocatable
- Provide a "magic" library function to safely perform trivial relocation
 - This function replaces old memcpy with well defined behavior

Library API for Trivial Relocation API to support core language design using compiler intrinsics

- Type trait is trivially relocatable<T>
 - Reports whether a type is trivially relocatable, per core language definition
- Magic function to copy object representations T* trivially relocate(T* begin, T* end, T* new location);
 - Mandates: is trivially relocatable v<T> && !is const v<T>
 - Postconditions: new location range has a copy of the object representations of the source range; ends lifetime of source range objects
 - Remarks: Overlapping ranges are supported. No constructors or destructors are executed.
- Implemented in Corentin's branch; available on Compiler Explorer

Summary of Relocation APIs New LWG content supporting relocation

- is trivially relocatable<T>
- is trivially relocatable v<T>
- T* trivially relocate(T* begin, T* end, T* new location);

Replaceability

What is Replaceability? **Backwards compatibility for the standard library**

- Several parts of the library expect that move-assignment and destroy-thenmove-construct are interchangeable
 - We name this property replaceability
 - We provide language support to declare a type replaceable
 - We proved a trait to detect replaceable types
- We may want to check for replaceability before applying trivial relocation optimizations in places where assignment has been used as an optimization
 - Otherwise, there may be a change of observable behavior on existing code

How Does P2786 Support Replaceability? Syntax and library APIs

- Define core notion of replaceability
- Deduce whether type is replaceable if is has no user-supplied move constructor, move assignment operator, nor destructor
 - Use a keyword to deduce otherwise
- Provide a type trait to report if a type is replaceable
 - is_replaceable<T>

How Does P2786 Use Replaceability?

- Enables Qol consistency checks in library implementations
 - e.g., to give warnings in std::vector
 - To guard against unsafe optimizations, e.g., in std::swap

The Complete Library Interface

Summary of all new APIs **New LWG content for C++26**

- is trivially relocatable<T>
- is replaceable<T>
- #define _____cpp_lib_trivially relocatable

• T* trivially relocate(T* begin, T* end, T* new location);



Vendor Freedom

Library Adoption of New Features Which library types are trivially relocatable and replaceable?

- Too early to provide a full library review
 - Common cases like array, pair, and tuple should "just work" D
 - Desirable for vector, shared ptr and others, but are we ready to specify?
 - Unlikely to be portable for basic string, list, and others
- Do vendors have freedom to experiment (like with noexcept) or are they bound by the exact specification (like with constexpr)?
 - Library introduction will explicitly bless freedom for vendors to make types trivially relocatable, replaceable, or not — unless otherwise specified



What Comes Next?

Library Adoption of New Features Which library types are trivially relocatable and replaceable?

- A paper providing a full update to the uninitialized_* algorithms to support relocation
 - Jointly authored by Louis Dionne and Alisdair Meredith
- A proposal for an additional simple relocate function
 - Authored by Alisdair Meredith
- A proposal for an addition function to relocate a single object
 - Tentatively authored by Louis Dionnel
- Libraries will experiment with optimizations for std::swap
 - P2786, as seen today, offers everything needed for standard library vendor Qol



Any questions?