P3745R0
Rebuttal to P1144R13

Presentation for EWG and LEWG, Sofia, 2025-06-17

Pablo Halpern <phalpern@halpernwightsoftware.com>

EWG Presentation History

- Arthur O'Dwyer presented P1144R13 to EWG in Sophia on 2025-06-17.
- Immediately after Arthur's presentation, Pablo Halpern presented the following slides — except for the last three, which were added later.
- Since the design cutoff for C++26 had passed, the only way forward for P1144 would have been to delay the entire TR feature to C++29.
- After discussion there was strong consensus against reverting P2786.

P1144R13: EWG considers P1144R13 highly desirable and, due it being a breaking change, we instruct CWG to revert P2786 from C++26.

SF	F	N	A	SA
8	6	4	14	20

LEWG Presentation History

- Arthur O'Dwyer also presented P1144R13 in LEWG later the same day, but the chair limited his presentation to the library aspects of the paper.
- Pablo Halpern presented just the last three of the slides in this deck, refuting specific assertions in Arthur's presentation.
- After discussion, there was strong consensus against pursuing P1144.

We encourage more exploration on the library modifications described in P1144R13 (proposing rationale, examples for usage and code differences, etc.).

SF	F	N	A	SA
1	2	2	7	11

Introduction

Status Quo

- P2786R13, *Trivial Relocatability For C++26*, was voted into the WP at the February 2025 meeting in Hagenberg and provides
  - Context-sensitive keywords `trivially_relocatable_if_eligible` and `replaceable_if_eligible`
  - Traits `is_trivially_relocatable<T>`, `is_replaceable<T>`, and `is_nothrow_relocatable<T>`
  - Low-level algorithm `trivially_relocate`
  - High-level algorithm `relocate`
- P3516, *Uninitialized algorithms for relocation*, would add a set of `uninitialized_relocate_*` algorithms. It is currently in LWG and might make it to C++26.
- In Sofia (Mon 2025-06-16), LEWG voted against P3631R0, thus keeping *both* `relocate` and `uninitialized_relocate_*`.

No New Information

- Previous versions of P1144 have failed to reach consensus to move forward.
- A new section in P1144R13 lists complaints about the current WP despite those concerns having already been addressed by this Committee.
- The subsequent slides refute each of the supposedly unaddressed assertions made in P1144R13.
    - The OED defines "address" as "to apply (oneself) or direct (one's energies) to something."
    - "I don't like it." ≠ "It was not addressed."
- P1144R13 does not address the issues that prevented earlier versions from reaching consensus to move forward, nor does it present any new information. The Committee, therefore, would not normally spend time on it.

Point-By-Point Rebuttal

This section was presented to EWG but not LEWG

Complaint: A Trivially Copyable Type Might Not Be Trivially Relocatable

P1144R13: "The programmer is allowed to create a type that is trivially copyable but not trivially relocatable."

- **BY DESIGN**:
  - `is_trivially_relocatable` is deliberately more similar to `is_trivially_move_constructible` than `is_trivially_copyable`.
  - A deleted move constructor turns off automatic TR, but it can be restored with the `trivially_relocatable_if_eligible` keyword.
- **ADDRESSED:** LEWG Telecon 2024-04-09 (and discussed often elsewhere)
  - Present in P2786 since R0
  - Is a deliberate feature (not a bug) to honor the programmer's intent

Complaint: Cannot Make a Class Having an `offset_ptr` Member Into a TR Class

P1144R13: "It's ill-formed to create a trivially relocatable type with an `offset_ptr` member, or even a Boost `static_vector` member."

- **PARTIALLY TRUE (was true pre-St. Louis):** Ineligible types decorated with `trivially_relocatable_if_eligible` will be **well-formed** but **not TR**.
- **BY DESIGN:** To maintain encapsulation and prevent UB, we do not allow trivially relocatable objects having non-trivially relocatable subobjects.
- **ADDRESSED:** In P2786R7
  - Well-formed semantics added after St Louis, June 2024 EWG review of P3236R1. Trivial relocatability is determined automatically by the compiler.
  - Preventing trivial relocation of non-TR subobjects has been a safety feature of P2786 since R0.

Complaint: Syntax Is Not Backward Compatible

P1144R13: “No backward-compatible syntax. P2786 proposes a new keyword, and goes out of its way to make the keyword unusable in C++23-and-earlier.”

- **BY DESIGN:**
  - C++26 language features are not available in C++23. Emulation in earlier versions of the language has never been a major consideration for new proposals.
  - P1144's ignorable attribute syntax doesn't meet EWG criteria for an attribute (see P2553R3).
- **ADDRESSED:** Attribute syntax was polled by EWGI on 2023-02-10 in Issaquah.

EWGI believes that the relocatable annotation in P1144R6 is acceptable as an attribute.

SF	F	N	A	SA
0	6	4	6	2

Assertion: P1144 Provides More Opportunities for Optimization

P1144R13: "P1144, by providing stronger guarantees of value semantics, permits more optimizations."

- **FALSE:**
  - The `is_replaceable` trait was introduced to expand optimization opportunities (but still disallows it for cases where P1144 introduces UB).
  - Many types that are not trivially relocatable in P1144 are (or can be) trivially relocatable in the WP, including polymorphic types and many `pmr` types.
- **ADDRESSED:** The `is_replaceable` trait was added in P2786R7

Assertion: Annotations Will Lead to Bugs

P1144R13: “P2786’s warrant marking is ‘viral downward.’ As container authors, we think P2786’s normalization of a large number of explicit markings will cause programmer fatigue and lead to bugs.”

- **FALSE:**
  - P2786 was designed to minimize UB, whereas P1144 makes adding UB easy.
  - Any rule-of-zero type will have the correct trivial relocatability, and any other TR type will need annotations *in both P1144 and the status quo*.
  - P1144R13 provides no examples of larger numbers of annotations or potential bugs.
- **BY DESIGN:** In contrast to P1144's "sharp knife" (i.e., foot gun), P2786 deliberately disallows trivially relocating non-TR subobjects.
- **ADDRESSED:** EWGI on 2023-02-10 in Issaquah (and repeatedly since)
  - The “viral” nature of P2786’s warrant has been part of the design since R0.

Complaint: Trivially Relocatable Polymorphic Types

P1144R13: "The ARM64e platform uses 'vptr signing.' Its vptrs are never safe to `memcpy`. Yet the Working Draft permits `is_trivially_relocatable` to return `true` for types with vptrs, types recursively containing members with vptrs, etc. This means that it is never safe to use e.g. `realloc` on ARM64e even when we know that the objects being reallocated are all `is_trivially_relocatable`."

- **BY DESIGN:**
  - Anything that is not trivially copyable yields UB when used with those C library functions (and P1144 does not change this).
  - The `trivially_relocate` function was added (in both the WP and P1144) to address this need.
  - Note that P1144's "sharp knife" annotations allow polymorphic objects to be trivially relocated, even if relocation will break on some platforms (e.g., those that use vtbl signing).
- **ADDRESSED**: EWG on 2025-02-14 in Hagenberg – the issue of vtbl signing was discussed at length and confirmed in a unanimous poll:

Confirm changes for unions containing polymorphic types as presented and return to CWG for C++26: It should be implementation-defined whether unions containing polymorphic types are trivially relocatable.

SF	F	N	A	SA
20	20	2	0	0

Complaint: The Keywords Are Ugly

P1144R13: "P2786's approved syntax is shockingly ugly: `class C trivially_relocatable_if_eligible replaceable_if_eligible { }`. P1144's attribute syntax is unobtrusive by comparison: `class [[trivially_relocatable]] C { }`."

- **FALSE:** Beauty is subjective and is not an engineering criterion.
- **BY DESIGN:** The keywords selected are precise, clear, and unambiguous. As engineers, we can agree that these qualities are objectively beautiful. 🙂
- **ADDRESSED:** EWG in Hagenberg , February 2025 – the keywords were discussed and achieved strong consensus when polled.

P2786r11: change `memberwise_trivially_relocatable` to `trivially_relocatable_if_eligible` and `memberwise_replaceable` to `replaceable_if_eligible`, and proceed as previously approved to CWG for C++26.

SF	F	N	A	SA
19	26	7	4	1

Conclusion

- P1144R13's list of complaints incorrectly states that some issues were unaddressed. All the issues listed were discussed and voted on, and P2786 progressed through our regular process.
- The version of P2786 that was adopted in Hagenberg contained wording that was thoroughly reviewed by EWG, CWG, LEWG, and EWG over the course of several meetings.
- The wording in P1144R13 discards the changes adopted in Hagenberg and replaces them with wording that has never been reviewed by any incubator, evolution, or wording group.
- Moving forward with P1144 invites NB comments to excise trivial relocation from the WP. Even the authors of P1144 agree that trivial relocation is important and desired for C++26.
- P1144R13 offers no new information that justifies Committee time.

Q&A

This section was presented in LEWG but not EWG.
It provides more detail on some of the previous material.

UB in P1144

Q: How does P1144's "sharp knife" make adding UB easy?

A: The following class can be trivially relocated safely in MSVC but yields UB with libc++ due to the differences in `std::list` implementations:

```
class X [[trivially_relocatable]] {
  std::list<int> mem;
…
};
```

This issue applies to any type whose implementation differs between libraries or types authored by other people that may change from one version to the next.

Existing Library Experience

Q: Is the definition of `is_trivially_relocatable` in existing public libraries really closer to P1144 than P2786?

A : No: BSL, Folly, and Parlay (*at least*) do not require a type to be replaceable to be trivially relocated.

- Assignment is not part of the definition of the trait in BSL, Folly, and Parlay.
- Absail's definition does forbid user-defined move assignment, but it does not use the trait in such a way that replaceability is needed.
- Thus, it can be argued that existing libraries are closer to P2786 than P1144.
- Nevertheless, a P1144-like trait is easily composed from existing traits.

The vtbl Pointer Signing Red Herring

Q: Can a P1144 trivially relocatable type be used with `realloc`?

A: No. Despite the non-normative note, the wording of P1144 does *nothing* to allow any new types to be used with C-library byte-copying functions.

- Being trivially copyable and having implicit lifetime remain the requirements for `memcpy`, `memmove`, and `realloc`. P1144 adds no new types to that set.
- Although it might work in practice (and will probably continue to do so), a polymorphic type cannot be used with these C functions, regardless of vtbl pointer signing.
- The P1144 polymorphic type constraint can be easily emulated using `is_trivially_relocatable_v<T> && not is_polymorphic_v<T>`