Open Issues in `std::execution::task`

Document Number:	P4007R1
Date:	2026-03-12
Audience:	LEWG

Reply-to:	Vinnie Falco vinnie.falco@gmail.com Mungo Gill mungo.gill@me.com

Table of Contents

Abstract

`std::execution::task` (P3552R3[1]) has open issues identified by national ballot comments, LWG issues, and published papers. This paper classifies each issue by whether it can be resolved after C++26 ships or whether shipping forecloses the fix.

Revision History

R1: March 2026 (prior to Croydon meeting)

- Complete rewrite as an informational classification of open issues.

R0: February 2026 (pre-Croydon mailing)

- Original analysis of structural gaps. See P4007R0[2].

---

1. Disclosure

The authors developed P4007R0[2] ("Senders and Coroutines"). The classification below holds regardless of whether any alternative design exists.

---

2. Fixed After Ship

`task`'s `promise_type` is a class template instantiated in user code. Its `operator new`, allocator selection, stop token storage, environment forwarding, and destruction ordering can change between standard revisions without binary incompatibility. These issues are fixable post-ship: Unusual Allocator Customisation[3], Flexible Allocator Position[3], Shadowing The Environment Allocator[3], Stop Source Always Created[3], Stop Token Default Constructible[3], Task Not Actually Lazily Started[3], Frame Destroyed Too Late[3], No Default Arguments[3], `unhandled_stopped` Not `noexcept`[3], Environment Design Inefficient[3], Non-Sender Awaitables Unsupported[3], Future Language Feature Could Avoid `co_yield`[3], No TLS Capture/Restore Hook[3], `return_value` / `return_void` Have No Specification[3], `co_return { args... }` Unsupported[3], `change_coroutine_scheduler` Requires Assignable Scheduler[3], Sender-Unaware Coroutines Cannot `co_await` a `task`[3], Missing Rvalue Qualification[3], Parameter Lifetime Is Surprising[4], No Protection Against Dangling References[4], `co_yield with_error` Is Clunky[4], `co_await schedule(sch)` Is an Expensive No-Op[4], Coroutine Cancellation Is Ad-Hoc[4].

`affine_on` semantics, rescheduling behavior, and algorithm customisation are specification-level concerns. Tightening requirements or adding default implementations does not change any published interface. These issues are fixable post-ship: `affine_on` Default Implementation Lacks Specification[3], `affine_on` Semantics Not Clear[3], `affine_on` Shape May Not Be Correct[3], `affine_on` Shouldn't Forward Stop Requests[3],

`affine_on` Customisation For Other Senders[3], Starting a `task` Reschedules Unconditionally[3], Resuming After a `task` Reschedules Unnecessarily[3], `bulk` vs. `task_scheduler` [3], No Completion Scheduler[3], `with_awaitable_senders` Unused[3].

---

3. Not Fixable Post-Ship

The issues in this section are items where shipping forecloses the fix.

Issue	References	Fixed
Allocator Timing	P3980R0[5], P3796R1[3], LWG 4356[6], US 254-385[7]	no
Allocator Propagation	P3980R0[5], P3796R1[3]	no
Error Return	P3950R0[8], P3801R0[4], P1713R0[9]	no
Symmetric Transfer	P2583R3[10], US 246-373[11], LWG 4348[12], P3801R0[4], P3796R1[3]	no

- **Allocator Timing.** `task` does not allow the user to specify an allocator at the launch site for coroutine frame allocation. The coroutine frame is allocated by `operator new` at the call site, before any sender `connect` / `start` machinery runs. The receiver's environment - which carries the allocator - is not yet available at that point. The `allocator_arg` mechanism in P3552R3[1] works around this by requiring the allocator in the coroutine's parameter list, but that locks in a caller-specified approach and forecloses environment-based injection.
- **Allocator Propagation.** `task` does not propagate the frame allocator to child tasks. Each child task allocates its frame independently. Shipping without propagation locks in a design where every launch site must specify its allocator explicitly, foreclosing transparent propagation through the coroutine call tree.
- **Error Return.** `task` requires `co_yield with_error(e)` to propagate an error to the caller. `co_return` cannot carry an error value because `return_value` and `return_void` are mutually exclusive in the current coroutine specification. Shipping this interface locks in the `co_yield` mechanism and forecloses `co_return`-based error propagation, which would require a language change.
- **Symmetric Transfer.** The completion functions (`set_value` , `set_error` , `set_stopped`) and `start()` return `void` , providing no channel to propagate a `coroutine_handle<>` . When a sender completes synchronously, the receiver calls `handle.resume()` on the caller's stack, adding a frame per completion

with no upper bound. Shipping this protocol forecloses the `coroutine_handle<>`-returning completion protocol that would enable symmetric transfer.

Acknowledgements

The authors thank Andrzej Krzemieński for feedback that sharpened the scope of this revision. Thanks are also due to Dietmar Kühl, Michael Hava, Mark Hoemmen, Ian Petersen, and Ville Voutilainen for technical discussion that informed the analysis.

References

Papers, issues, and ballot comments referenced in this document.

WG21 Papers

1. P3552R3 - "Add a Coroutine Task Type" (Dietmar Kühl, Maikel Nadolski, 2025). https://wg21.link/p3552r3
2. P4007R0 - "Senders and Coroutines" (Vinnie Falco, Mungo Gill, 2026). https://wg21.link/p4007r0
3. P3796R1 - "Coroutine Task Issues" (Dietmar Kühl, 2025). https://wg21.link/p3796r1
4. P3801R0 - "Concerns about the design of std::execution::task" (Jonathan Müller, 2025). https://wg21.link/p3801r0
5. P3980R0 - "Task's Allocator Use" (Dietmar Kühl, 2026). https://isocpp.org/files/papers/P3980R0.html
6. LWG 4356 - "connect() should use get_allocator(get_env(rcvr))". https://cplusplus.github.io/LWG/issue4356
7. US 254-385 - C++26 NB ballot comment. https://github.com/cplusplus/nbballot/issues/960
8. P3950R0 - "return_value & return_void Are Not Mutually Exclusive" (Robert Leahy, 2025). https://wg21.link/p3950r0
9. P1713R0 - "Allowing both co_return; and co_return value; in the same coroutine" (Lewis Baker, 2019). https://wg21.link/p1713r0
10. P2583R3 - "Symmetric Transfer and Sender Composition" (Mungo Gill, Vinnie Falco, 2026). https://isocpp.org/files/papers/P2583R3.pdf
11. US 246-373 - C++26 NB ballot comment. https://github.com/cplusplus/nbballot/issues/948
12. LWG 4348 - "task doesn't support symmetric transfer". https://cplusplus.github.io/LWG/issue4348
13. P2300R10 - "std::execution" (Michał Dominiak et al., 2024). https://wg21.link/p2300r10

Other

14. C++ Working Draft - (Richard Smith, ed.). https://eel.is/c++draft/