Is `std::execution` a Universal Async Model?

Document Number:	P4041R0
Date:	2026-05-01
Intent:	Inform
Audience:	LEWG

Reply-to: Vinnie Falco vinnie.falco@gmail.com

Table of Contents

Abstract

C++20 standardized coroutines. C++26 adds `std::execution`. Both are asynchronous models. The question is whether the C++26 model subsumes the C++20 model, or whether they serve different domains. This paper collects publicly available evidence.

Revision History

R0: May 2026 (pre-Brno mailing)

- Initial version.

---

1. Disclosure

The author provides information and serves at the pleasure of the committee.

The author developed and maintains Capy and Corosio and believes coroutine-native I/O is a practical foundation for networking in C++.

Coroutine-native I/O and `std::execution` are complementary. Each serves the domain where its design choices pay off.

This paper examines the published record. That effort requires re-examining consequential papers, including papers written by people the author respects.

The author developed P4003R3 [1] ("A Minimal Coroutine Execution Model"), P4007R3 [2] ("Open Issues in `std::execution::task`"), P4014R2 [3] ("The Sender Sub-Language For Beginners"), and P2583R4 [4] ("Symmetric Transfer and Sender Composition"). A coroutine-only design cannot express compile-time work graphs, does not support heterogeneous dispatch, and assumes cooperative scheduling. This paper does not cite those libraries or P4003R3 as evidence for any claim. They are disclosed as context for the author's perspective.

This paper asks for nothing.

---

2. Two Standard Async Models

C++20 standardized coroutines (P0912R5 [5], P0913R1 [6]). They are ISO C++.

C++26 adds `std::execution` (P2300R10 [7], "std::execution"). Schedulers, senders, receivers, and composable algorithms for asynchronous computation.

Both are standard facilities. Both are async models. The question is the relationship between them.

Eric Niebler wrote in "Structured Concurrency" [45] (2020):

> *"I think that 90% of all async code in the future should be coroutines simply for maintainability."*

Niebler wrote in "What Are Senders Good For, Anyway?" [46] (2024):

> *"If your library exposes asynchrony, then returning a sender is a great choice: your users can await the sender in a coroutine if they like."*

The framework's architect placed 90% of async code in the coroutine column.

---

3. What `std::execution` Does Well

`std::execution` provides zero-allocation sender pipelines for compile-time work graphs. The `connect`/`start` protocol collapses a sender chain into a single operation state with no heap allocation and no virtual dispatch. Domain customization allows a CUDA scheduler to customize `bulk` to emit a kernel launch instead of a loop. Type-level completion signatures enable compile-time routing without runtime inspection. NVIDIA's CUDA compiler does not support C++20 coroutines in device code [57]; stdexec [48] - the reference implementation - was built at NVIDIA.

Herb Sutter reported [47] that Citadel Securities uses `std::execution` in production: *"We already use C++26's `std::execution` in production for an entire asset class, and as the foundation of our new messaging infrastructure."*

---

4. The Post-Adoption Record

P2300R10 [7] was adopted into the C++26 working draft at St. Louis in July 2024. The following is our attempt at a complete enumeration of post-adoption changes, compiled from published WG21 mailings [58], the LWG issues list [59], and the C++26 NB ballot repository [60].

4.1 Changes by Category

Category	Papers / Issues	Count
Removals	P3682R0 [8], P3149R11 [9] (replacing `ensure_started`)	2
Rewrites	P3481R5 [10] (`bulk`)	1
Architectural fixes	P3887R1 [11], P3557R3 [12]	2
Wording omnibus	P3396R1 [13]	1
Post-adoption adds	P3433R1 [14], P3284R4 [15], P3570R2 [16], P3552R3 [17], P2079R10 [18], P3149R11 [9], P3927R0 [19]	7
LWG defects	4190 [36], 4206 [37], 4215 [38], 4356 [39], 4368 [40]	5
NB ballot	US 255-384 [41], US 253-386 [42], US 254-385 [43], US 261-391 [44]	4

4.2 Direction of Change

Every post-adoption item falls into one of three categories.

Origin	Items	Count
Sender Sub-Language	P2855R1, P2999R3, P3175R3, P3187R1, P3303R1, P3373R2, P3557R3, P3570R2, P3682R0, P3718R0, P3826R3, P3941R1, LWG 4190, 4206, 4215, 4368	16
Sender Integration	P3927R0, P3950R0, D3980R0 [20], LWG 4356, US 255-384, US 253-386, US 254-385, US 261-391	8
Coroutine-Intrinsic	-	0

Twenty-four post-adoption items modified the sender sub-language or its integration. Zero modified coroutines.

---

5. Published Findings

Dietmar Kühl catalogued sixteen open concerns with `task` in P3796R1 [21] ("Coroutine Task Issues," 2025). On symmetric transfer:

> *"The specification doesn't mention any use of symmetric transfer."*

Kühl reworked the allocator model in D3980R0 [20] (2026-01-25) - six months after P3552R3 [17]'s adoption at Sofia.

Jonathan Müller identified a stack overflow vulnerability in P3801R0 [22] ("Concerns about the design of std::execution::task," 2025):

> *"Having iterative code that is actually recursive is a potential security vulnerability."*

Müller confirmed the cause:

> *"The reason `co_yield` is used, is that a coroutine promise can only specify `return_void` or `return_value`, but not both. If we want to allow `co_return;`, we cannot have `co_return with_error(error_code);`. This is unfortunate, but could be fixed by changing the language to drop that restriction."*

Robert Leahy proposed a core language change in P3950R0 [23] (2025):

> *"Disallowing it either disadvantages coroutines vis-a-vis `std::execution` or necessitates library workarounds."*

Chris Kohlhoff identified the partial success tension in P2430R0 [24] ("Partial success scenarios with P2300," 2021):

> *"Due to the limitations of the `set_error` channel (which has a single 'error' argument) and `set_done` channel (which takes no arguments), partial results must be communicated down the `set_value` channel."*

---

6. Structural Observations

These are not design defects. They are tradeoffs the sender model makes for compile-time work graphs. The question is whether those tradeoffs should also bind I/O.

Boundary	Sender Model Requires	Coroutine/I/O Cost
Error channels	Compile-time channel routing	`(error_code, size_t)` cannot route through three channels
Error returns	Separate `set_error` channel	`co_yield with_error(ec);` reverses established `co_yield` semantics
Frame allocator	Deferred execution via `connect`/`start`	`promise_type::operator new` fires before sender machinery
Symmetric transfer	Struct composition with void completions	`coroutine_handle<>`-returning `await_suspend` unreachable

Both P2300R10 [7] bridges - `sender-awaitable` and `connect-awaitable` - use `void await_suspend`.

Senders get the allocator they do not need. Coroutines need the frame allocator they do not get.

6.1 The Error Channel Timeline

Date	Event
February 2020 (Prague)	Partial success raised during P1678R2 [25] review. LEWG polls SF:7/F:14/N:9/A:3/SA:0. No resolution.
February 2021 (SG4 telecon)	Participant states sender/receivers have a loss: no success/partial-success.
July-October 2021 (LEWG)	Debated across five telecons. LEWG outcome document: *"Better explain how partial success works with senders/receivers."*
August 2021	Kohlhoff publishes P2430R0 [24].
2022-2024	Six revisions through P2300R10 [7]. Three-channel model unchanged.
2023	Kühl's P2762R2 [26] preserves Asio's `(error_code, size_t)` convention.
November 2023 (Kona)	SG4 polls that networking must use the sender model (SF:5/F:5/N:1/A:0/SA:1).
November 2024 (Wroclaw)	Channel question resurfaces during P0260 (Concurrent Queues). Debated across two face-to-face meetings.
February 2025 (Hagenberg)	Concurrent queue channel question remained open. Poll to reopen withdrawn.
2025-2026	P3570R2 [16] documents `optional<T>` / channel model mismatch.

7. Established Practice

Library	Symmetric Transfer	Error Delivery	Async Model
cppcoro [49] (Baker)	`coroutine_handle<>`	`co_return` / exceptions	Coroutine-native
folly::coro [50] (Meta)	`coroutine_handle<>`	Exceptions	Coroutine-native
Boost.Cobalt [51] (Morgenstern)	`coroutine_handle<>`	`co_return` / exceptions	Coroutine-native
libcoro [52] (Baldwin)	`coroutine_handle<>`	`co_return` / exceptions	Coroutine-native
Boost.Asio (Kohlhoff)	N/A	Error codes via `as_tuple`	Completion-token
asyncpp [53] (Kardos)	Event-based	`co_return` / exceptions	Coroutine-native

SG14 (February 2026 mailing): *"SG14 advise that Networking (SG4) should not be built on top of P2300. The allocation patterns required by P2300 are incompatible with low-latency networking requirements."*

Six independent libraries converged on `coroutine_handle<>`-returning `await_suspend`. The standard uses `void`.

---

8. Ecosystem Adoption

The sender/receiver model has been public since P2300R0 [61] (2021). stdexec [48] has existed as a reference implementation throughout. These results may not be exhaustive; additions are welcome.

Domain	Project	Status
TCP networking	senders-io [54]	19 stars. "Still very experimental." Requires non-main stdexec branch.
HTTP	fuchsia [55]	2 stars, 0 forks. Last commit August 2023.
TLS	(none found)	
File I/O	senders-io [54]	Experimental. Documentation sections empty.
Database clients	(none found)	
DNS resolution	(none found)	
Signal handling	(none found)	
Bare metal / embedded	Intel cpp-baremetal-senders-and-receivers [56]	288 stars. P2300 subset. "Coroutines are not considered at all."
GPU / heterogeneous	stdexec [48]	1,300+ stars. Reference implementation. Active.
HFT / low-latency	(reported, proprietary)	Cannot be independently verified.

Rubén Pérez Hidalgo, author of Boost.MySQL [62], offered this assessment:

> *"This whole S/R thing is as if someone had seen Asio and said 'too simple'."*

The reference implementation has 1,300 stars. The I/O ecosystem built on it has 21.

Ben FrantzDale, who called [63] P2300 "amazing promise," also wrote:

> *"I've been keeping an eye on the P2300 'Senders' proposal for generic asynchrony for many years, but felt like I never quite 'got' it. I know I'm not the only one who has found it challenging to grok... if you step into implementations, you quickly find yourself in a sea of underscores, namespaces, and customization-point objects."*

Sean Baxter, author of the Circle compiler, reported [64] that swapping the order of a single constraint in the `sender` concept caused Clang to produce an error message so long (5,500 lines) that it triggered an internal compiler error.

Rainer Grimm, author of *Modernes C++*, abandoned [65] his planned C++26 library coverage: *"The implementation status of the library is not good enough. Therefore, I decided to continue with concurrency and* `std::execution`*. I will present the remaining C++26 features if a compiler implements them."*

9. Open Question

The domains where sender-based I/O could be publicly verified show no production implementations. The domains where deployment is reported are proprietary.

Acknowledgments

The author thanks Dietmar Kühl, Jonathan Müller, Chris Kohlhoff, Eric Niebler, and Lewis Baker for their published observations cited in this paper.

References

WG21 Papers

[1] P4003R3 - "A Minimal Coroutine Execution Model" (Vinnie Falco, Steve Gerbino, Mungo Gill, 2026).

[2] P4007R3 - "Open Issues in `std::execution::task`" (Vinnie Falco, Mungo Gill, 2026).

[3] P4014R2 - "The Sender Sub-Language For Beginners" (Vinnie Falco, Mungo Gill, 2026).

[4] P2583R4 - "Symmetric Transfer and Sender Composition" (Mungo Gill, Vinnie Falco, 2026).

[5] P0912R5 - "Merge Coroutines TS into C++20 working draft" (Gor Nishanov, 2018).

[6] P0913R1 - "Add symmetric coroutine control transfer" (Gor Nishanov, 2018).

[7] P2300R10 - "std::execution" (Michał Dominiak, Georgy Evtushenko, Lewis Baker, Lucian Radu Teodorescu, Lee Howes, Kirk Shoop, Michael Garland, Eric Niebler, Bryce Adelstein Lelbach, 2024).

[8] P3682R0 - "Remove std::execution::split" (Robert Leahy, 2025).

[9] P3149R11 - "async_scope — Creating scopes for non-sequential concurrency" (Ian Petersen, Jessica Wong, Kirk Shoop, et al., 2025).

[10] P3481R5 - "std::execution::bulk() issues" (Eric Niebler, 2025).

[11] P3887R1 - "Make when_all a Ronseal Algorithm" (Lewis Baker, 2025).

[12] P3557R3 - "High-Quality Sender Diagnostics with Constexpr Exceptions" (Eric Niebler, 2025).

[13] P3396R1 - "std::execution wording fixes" (Eric Niebler, 2025).

[14] P3433R1 - "Allocator Support for Operation States" (Eric Niebler, 2025).

[15] P3284R4 - "`write_env` and `unstoppable` Sender Adaptors" (Eric Niebler, 2025).

[16] P3570R2 - "Optional variants in sender/receiver" (Fabio Fracassi, 2025).

[17] P3552R3 - "Add a Coroutine Task Type" (Dietmar Kühl, Maikel Nadolski, 2025).

[18] P2079R10 - "System execution context" (Lee Howes, 2025).

[19] P3927R0 - "task_scheduler Support for Parallel Bulk Execution" (Lee Howes, 2026).

[20] D3980R0 - "Task's Allocator Use" (Dietmar Kühl, 2026).

[21] P3796R1 - "Coroutine Task Issues" (Dietmar Kühl, 2025).

[22] P3801R0 - "Concerns about the design of std::execution::task" (Jonathan Müller, 2025).

[23] P3950R0 - "return_value & return_void Are Not Mutually Exclusive" (Robert Leahy, 2025).

[24] P2430R0 - "Partial success scenarios with P2300" (Chris Kohlhoff, 2021).

[25] P1678R2 - "Callbacks and Composition" (Kirk Shoop, 2020).

[26] P2762R2 - "Sender/Receiver Interface For Networking" (Dietmar Kühl, 2023).

[27] P2855R1 - "Member customization points for Senders and Receivers" (Ville Voutilainen, 2024).

[28] P2999R3 - "Sender Algorithm Customization" (Eric Niebler, 2024).

[29] P3175R3 - "Reconsidering the std::execution::on algorithm" (Eric Niebler, 2024).

[30] P3187R1 - "Remove ensure_started and start_detached from P2300" (Kirk Shoop, Lewis Baker, 2024).

[31] P3303R1 - "Fixing Lazy Sender Algorithm Customization" (Eric Niebler, 2024).

[32] P3373R2 - "Of Operation States and Their Lifetimes" (Robert Leahy, 2025).

[33] P3718R0 - "Fixing Lazy Sender Algorithm Customization, Again" (Eric Niebler, 2025).

[34] P3826R3 - "Fix Sender Algorithm Customization" (Eric Niebler, 2026).

[35] P3941R1 - "Scheduler Affinity" (Dietmar Kühl, 2026).

LWG Issues

[36] LWG 4190 - "completion-signatures-for specification is recursive."

[37] LWG 4206 - "connect_result_t should be constrained with sender_to."

[38] LWG 4215 - "run_loop::finish should be noexcept."

[39] LWG 4356 - "connect() should use get_allocator(get_env(rcvr))."

[40] LWG 4368 - "Potential dangling reference from transform_sender."

NB Ballot Comments

[41] US 255-384.

[42] US 253-386.

[43] US 254-385.

[44] US 261-391.

Blog Posts

[45] Eric Niebler, "Structured Concurrency" (November 2020).

[46] Eric Niebler, "What Are Senders Good For, Anyway?" (February 2024).

[47] Herb Sutter, "Living in the future: Using C++26 at work" (April 2025).

Libraries

[48] stdexec - NVIDIA reference implementation of std::execution (1,300+ stars).

[49] cppcoro - C++ coroutine abstractions (Lewis Baker).

[50] folly::coro - Facebook coroutine library.

[51] Boost.Cobalt - Coroutine task types for Boost (Klemens Morgenstern).

[52] libcoro - C++20 coroutine library (Josh Baldwin).

[53] asyncpp - Async coroutine library (Péter Kardos).

[54] senders-io - Sender/receiver adaptation for async I/O (19 stars, "still very experimental").

[55] fuchsia - Experimental async networking on stdexec (2 stars, last commit August 2023).

[56] Intel cpp-baremetal-senders-and-receivers - P2300 subset for embedded (288 stars, "coroutines are not considered at all").

Platform Documentation

[57] NVIDIA CUDA Programming Guide, Section 5.3 - "C++ Language Support." Coroutines not listed for device code compilation.

Other

[58] WG21 paper mailings - ISO C++ committee papers.

[59] LWG issues list - Library Working Group active issues.

[60] C++26 NB ballot comments - National body comments repository.

[61] P2300R0 - "std::execution" (Eric Niebler, Kirk Shoop, Lewis Baker, Lee Howes, 2021).

[62] Boost.MySQL - Async MySQL client library for Boost (Rubén Pérez).

[63] Sender Intuition: Senders Don't Send - Ben FrantzDale, Oct 2024.

[64] NVIDIA/stdexec issue #856: sender concept has side effects, is order-dependent - Sean Baxter, March 2023.

[65] std::execution - MC++ BLOG - Rainer Grimm, Nov 2024.