Async Claims and Evidence

Document Number:	P4098R1
Date:	2026-05-01
Intent:	Inform
Audience:	WG21

Reply-to:	Vinnie Falco vinnie.falco@gmail.com C++ Alliance Proposal Team

Table of Contents

Abstract

Published claims about executors, networking, and unification shaped a decade of committee decisions. The published evidence behind those claims is documented here.

This paper surveys the published claims that shaped the trajectory of executors, networking, and asynchronous programming in C++. Each claim is presented verbatim with its source. For each claim, the published record was searched for supporting evidence - user surveys, deployment data, prototypes, experiments, measurements. Where evidence was found, it is documented. Where the published record contains no evidence, the cell is empty.

Revision History

R1: May 2026 (pre-Brno mailing)

- Corrected P4096R0 cross-reference (Section 4.3, not Section 6).
- Formatting corrections.

R0: April 2026 (post-Croydon mailing)

- Initial version.

---

1. Disclosure

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

This paper is part of the Network Endeavor (P4100R0), a project to bring coroutine-native I/O to C++.

The author developed and maintains Capy [1] and Corosio [2] 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.

Selection criteria. Claims were selected from the papers in the causal chain documented in the companion papers - P4094R0 [3], P4095R0 [4], P4096R0 [5], and P4097R0 [6] - plus additional claims from P2453R0 [7] and P2470R0 [8] that shaped the committee's direction. Claims from both sides of the debate were included: the unification proponents (P0443R0, P0761R2, P1525R0), the P2464R0 diagnosis, the P2469R0 response, the P2300R2 deployment claims, and the pro-Networking-TS claims (Section 2.6). The published record was searched using all WG21 papers available to the author through the March 2026 mailing.

Evidence bar. The bar for evidence is deliberately low. A code snippet from a real codebase counts. A prototype counts. A survey of even a handful of users counts. A hypothetical code example constructed by the author counts - the table notes it. Where the Evidence column is empty, the search found nothing.

Limitations. The author acknowledges that absence of evidence is not evidence of absence. Committee discussions occur in rooms, hallways, dinners, and private channels that leave no public trace. Committee members bring domain expertise to polls and design reviews that is not captured in published papers; the absence of a published document does not mean the absence of informed judgment. If a reader is aware of a document, analysis, or discussion that this paper's research did not reach, the author welcomes the correction and will update the record in a future revision.

This paper asks for nothing.

---

2. The Claims

2.1 Unification (2014-2020)

Claim	Source	Year	Evidence
"we want to be able to have a single thread pool object that can be used for all of the above use cases. In real world applications, the use cases do not always exist in isolation."	P0285R0 [9]	2016	No survey of applications that mix networking and parallel algorithm executors on the same pool. No deployment data showing friction from separate pools.
"This unified executor proposal serves the use cases of those independent proposals with a single consistent programming model."	P0443R0 [10]	2016	No prototype demonstrating that the unified model serves all three use cases. No experiment comparing unified and domain-specific approaches.
"each facility's unique interface necessitates an entirely different implementation... the problem worsens as the library introduces new functions."	P0761R2 [11]	2018	One hypothetical code snippet. P0761R2 Section 2 constructs a `parallel_for` with an `if/else` chain over OpenMP, GPU, and thread pool. Authored by the proposal authors, not drawn from a deployed codebase. No measurement from a real standard library implementation.
"P0443 represents a significant body of compromise and consensus seeking."	P1791R0 [12]	2019	The breadth of participation is documented (Google, NVIDIA, Sandia, Codeplay, Facebook, etc.). More than 100 papers. This is evidence of effort. It is not evidence that the unification preserved domain semantics.
"while they may be essential in a particular domain, they were not universal"	P1791R0 [12]	2019	The statement is an acknowledgment that domain-specific requirements were removed during unification. No analysis of what each domain lost.
"some reduction in usability, due to the increased complexity of the unified interface"	P2469R0 [13]	2021	The authors of the unified proposal acknowledge the cost. No measurement of the magnitude.

2.2 Basis Operation (2019)

Claim	Source	Year	Evidence
"Any errors that happen... are handled in an implementation-defined manner... no generic code can respond to asynchronous errors in a portable way."	P1525R0 [14]	2019	Four error-handling strategies documented. Appendix B demonstrates a `fallback_executor` using the Sender/Receiver error channel. Analysis is under the work framing only (P4095R0 [4] Section 4).
"It is not possible to build this kind of non-allocating executor-schedule operation if one-way execute() is the basis operation."	P1525R0 [14]	2019	Appendix C demonstrates a `thread_dispatcher` with zero-allocation scheduling via `co_await ex.schedule()`. The demonstration is valid under the work framing.
"we cannot implement schedule generally in terms of one-way execute."	P1525R0 [14]	2019	Appendix A demonstrates the reverse: `execute` implemented in terms of `schedule` and `submit`. The asymmetry is demonstrated under the work framing.
"Eliminate OneWayExecutor, BulkOneWayExecutor, and interface-changing properties."	P1658R0 [15]	2019	No analysis of what networking loses when the continuation property is removed. No discussion of the continuation framing.

2.3 Networking Dependency (2018-2021)

Claim	Source	Year	Evidence
"SG1 has decided that the Networking TS should not be merged into the C++ working paper before executors go in."	P1256R0 [16]	2018	Published decision. The coupling is a fact.
"How blocked is networking on the executors wording process?"	P2130R0 [17]	2020	Pablo Halpern's question in the Prague minutes. No published answer quantifying the degree of blockage.
"we should not standardize the Networking TS as it's currently designed. The problem is the use of a P0443 executor."	P2464R0 [18]	2021	The analysis is under the work framing. No analysis under the continuation framing (P4096R0 [5] Section 2).
"Stop spending energy on standardizing the Networking TS for C++23."	P2464R0 [18]	2021	Published recommendation. The committee acted on it.

2.4 P2464R0 Diagnosis (2021)

Claim	Source	Year	Evidence
"A P0443 executor is not an executor. It's a work-submitter."	P2464R0 [18]	2021	P2464R0 provides analysis under the work framing. No analysis under the continuation framing at the time. P4096R0 [5] Section 4 provides the continuation-framing analysis retroactively.
"it just plain can't handle errors that occur after submission. That's an irrecoverable data loss"	P2464R0 [18]	2021	P2464R0 provides analysis of the `execute(F&&)` signature. No timeline analysis of when error information exists. P4096R0 [5] Section 4.3 provides the timeline analysis retroactively.
"Composing Networking TS completion handlers or asynchronous operations... seems like an ad-hoc and non-generic exercise. Whether they were designed for or ever deployed in larger-scale compositions... I don't know."	P2464R0 [18]	2021	Boost.Beast (2017) deployed three layers of composed async operations using hand-written state machines. P2464R0 was right that the Networking TS had no generic algorithmic composition mechanism. P4096R0 [5] Section 4.3 concedes this point.
"C++ programmers will write *thousands* of these executors."	P2464R0 [18]	2021	No survey of executor implementations. No measurement of how many executors exist in deployed codebases.

2.5 P2300 Scope and Deployment (2021-2024)

Claim	Source	Year	Evidence
"The sender/receiver model (P2300) is a good basis for most asynchronous use cases, including networking, parallelism, and GPUs." (SF:24 / WF:16 / N:3 / WA:6 / SA:3)	P2453R0 [7]	2021	Consensus in favor. No published prototype of sender-based networking at the time of the vote. No published deployment of sender-based networking as of 2026.
"We believe we need one grand unified model for asynchronous execution in the C++ Standard Library" (SF:4 / WF:9 / N:5 / WA:5 / SA:1)	P2453R0 [7]	2021	No consensus (leaning in favor). The premise was tested by poll and did not achieve consensus.
"the number of monthly users of sender/receiver-based async APIs number in the billions"	P2470R0 [8]	2021	Facebook deployment documented. The deployment is for sender/receiver composition and infrastructure, not for networking.
"NVIDIA is fully invested in P2300 senders/receivers... we plan to ship in production"	P2470R0 [8]	2021	Published statement. The deployment is for GPU dispatch and heterogeneous execution, not for networking.
"Networking in the C++ Standard Library should be based on the sender/receiver model"	P2452R0 [19]	2021	Poll text. No published prototype of sender-based networking at the time. P2762R2 [20] (2023) documents five routing options for compound I/O results, noting the single-argument error channel is "somewhat limiting."

2.6 Networking TS Readiness (2018-2021)

Claim	Source	Year	Evidence
"The Networking TS is baked, P2300 Sender/Receiver is not"	P2469R0 [13]	2021	The Networking TS was published as an ISO TS in 2018. Boost.Asio has been deployed for over fifteen years. The TS had no TLS, no HTTP, and an unresolved executor dependency (P1256R0 [16]).
"ASIO has been shipping this for years, and you can find a similar executor in java.util.concurrent."	P2464R0 [18] (citing proponents)	2021	Boost.Asio deployment is documented and real. No published survey of how many Asio users rely on custom executors. No published measurement of Asio executor usage beyond networking.
"some reduction in usability, due to the increased complexity of the unified interface, but the Networking TS can be updated to work with the new executors"	P2469R0 [13]	2021	The update was never published. No prototype demonstrating the Networking TS working with P0443R14 executors. No prototype demonstrating the Networking TS working with P2300 schedulers.

---

3. Observations

This paper does not draw conclusions from the table. Four observations are offered for the reader's consideration.

The deployment evidence for GPU dispatch and infrastructure is real. P2470R0 [8] documents sender/receiver at scale at Facebook, NVIDIA, and Bloomberg. The committee's decision to move beyond P0443R14 [21] addressed genuine deficiencies in the unified executor model. `std::execution` provides a composition algebra that neither the Networking TS nor the coroutine model provides. These are real achievements supported by published evidence.

The evidence is concentrated in one domain. The deployment evidence (P2470R0) documents sender/receiver at scale for GPU dispatch, thread pools, and infrastructure. No published paper documents production-scale networking using the sender model. The poll that included "networking" in its text (P2453R0 Poll 2) achieved consensus without published networking evidence.

The continuation framing was not examined. No claim in the table was analyzed under both the work framing and the continuation framing at the time it was made. The companion papers in this series (P4095R0 [4], P4096R0 [5]) provide that analysis retroactively.

The evidence bar was not applied uniformly. Some claims in the table are supported by deployment data, prototypes, or code examples. Others are supported by nothing in the published record. Both categories shaped committee decisions. This is not unusual in standards work - committee members bring domain expertise to polls and design reviews that is not

captured in published papers. The absence of a published document does not mean the absence of informed judgment. It does mean the judgment is not available for independent review.

---

4. Anticipated Objections

Q: Absence of evidence is not evidence of absence.

A: Section 1 says so. The table documents what the published record contains. If evidence exists in unpublished form, the author welcomes the correction.

Q: This cherry-picks claims.

A: The table includes claims from both sides - P2464R0's diagnosis, P2469R0's response, P1791R0's defense of P0443, P2470R0's deployment data, and the pro-Networking-TS claims in Section 2.6. If a reader is aware of a published claim that this paper omitted, the author will add it in a future revision.

Q: The committee had private discussions that informed these decisions.

A: Section 1 acknowledges this. The published record is the only record this paper can examine. Decisions that affect the entire C++ ecosystem deserve a published evidence base.

Q: You are arguing for your own library.

A: Section 1 discloses this. The claims and evidence in the table are verbatim quotes from published papers. They stand or fall on the published record, not on who assembled them.

Acknowledgments

The author thanks Christopher Kohlhoff for the executor model and the candid retrospective in P1791R0 [12]; Eric Niebler, Kirk Shoop, Lewis Baker, and Lee Howes for P1525R0 [14]; Ville Voutilainen for P2464R0 [18]; Bryce Adelstein Lelbach for the published poll outcomes in P2453R0 [7]; Detlef Vollmann for P1256R0 [16]; and Steve Gerbino for feedback on this paper.

References

[1] cppalliance/capy - Coroutine I/O primitives library.

[2] cppalliance/corosio - Coroutine-native networking library.

[3] P4094R0 - "Retrospective: The Unification of Executors and P0443" (Vinnie Falco, 2026).

[4] P4095R0 - "Retrospective: The Basis Operation and P1525" (Vinnie Falco, 2026).

[5] P4096R0 - "Retrospective: Coroutine Executors and P2464R0" (Vinnie Falco, 2026).

[6] P4097R0 - "Retrospective: The Networking Claim and P2453R0" (Vinnie Falco, 2026).

[7] P2453R0 - "2021 October Library Evolution Poll Outcomes" (Bryce Adelstein Lelbach, Fabio Fracassi, Ben Craig, 2022).

[8] P2470R0 - "Slides for presentation of P2300R2: std::execution (sender/receiver)" (Eric Niebler, 2021).

[9] P0285R0 - "Using customization points to unify executors" (Christopher Kohlhoff, 2016).

[10] P0443R0 - "A Unified Executors Proposal for C++" (Jared Hoberock, Michael Garland, Chris Kohlhoff, Chris Mysen, Carter Edwards, 2016).

[11] P0761R2 - "Executors Design Document" (Jared Hoberock, Michael Garland, Chris Kohlhoff, Chris Mysen, Carter Edwards, Gordon Brown, Michael Wong, 2018).

[12] P1791R0 - "Evolution of the P0443 Unified Executors Proposal to accommodate new requirements" (Christopher Kohlhoff, Jamie Allsop, 2019).

[13] P2469R0 - "Response to P2464: The Networking TS is baked, P2300 Sender/Receiver is not" (Christopher Kohlhoff, Jamie Allsop, Vinnie Falco, Richard Hodges, Klemens Morgenstern, 2021).

[14] P1525R0 - "One-Way execute is a Poor Basis Operation" (Eric Niebler, Kirk Shoop, Lewis Baker, Lee Howes, 2019).

[15] P1658R0 - "Suggestions for Consensus on Executors" (Jared Hoberock, Bryce Adelstein Lelbach, 2019).

[16] P1256R0 - "Executors Should Go To A TS" (Detlef Vollmann, 2018).

[17] P2130R0 - "WG21 2020-02 Prague Record of Discussion" (Nina Ranns, 2020).

[18] P2464R0 - "Ruminations on networking and executors" (Ville Voutilainen, 2021).

[19] P2452R0 - "2021 October Library Evolution and Concurrency Polls on Networking and Executors" (Bryce Adelstein Lelbach, Fabio Fracassi, Ben Craig, 2021).

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

[21] P0443R14 - "A Unified Executors Proposal for C++" (Jared Hoberock, et al., 2020).