# The Twenty-One Year Networking Arc


| | |
|---|---|
| Document Number: | P4099R1 |
| Date: | 2026-05-01 |
| Intent: | Inform |
| Audience: | WG21 |


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


Table of Contents



## Abstract


Four decisions, each locally reasonable, each under-evidenced, produced a decade without networking in the C++ standard.

This paper assembles the findings of five companion papers into a single causal chain: the unification of executors (2014, P4094R0 [1]), the basis-operation pivot (2019, P4095R0 [2]), the P2464R0 diagnosis (2021, P4096R0 [3]), the networking claim in the poll (2021, P4097R0 [4]), and the evidence base for the trajectory (P4098R0 [5]). Each paper documents one decision point. This paper places them in sequence, documents what is now available that was not available when those decisions were made, and credits the work that produced the tools the committee now has.

## Revision History

### R1: May 2026 (pre-Brno mailing)

- Corrected claim about coroutines in 2014 ("no standard coroutine facility existed", not "did not exist in any form").
- Corrected P0443 footnote mismatch in Acknowledgments.
- 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 [6] and Corosio [7] 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.

This paper asks for nothing.

---

## 2. The Causal Chain

| Year | Decision | Rationale | Published evidence | Companion paper |
|---|---|---|---|---|
| 2014 | Three independent executor models unified into one | Shared execution contexts; N x M explosion; synchronization coherence | One hypothetical code snippet. No prototype. No survey. No deployment data showing friction from separate models. | P4094R0 [1] |
| 2019 | `execute(F&&)` diagnosed as deficient; Cologne pivot to Sender/Receiver | No error channel, no cancellation signal, no zero-allocation scheduling | Diagnosis under the work framing only. Continuation framing not examined. Networking use case not analyzed. | P4095R0 [2] |
| 2021 | P2464R0 diagnosis applied to Networking TS; networking set aside | Executor is a "work-submitter"; no error channel; no generic composition | Analysis under the work framing only. Continuation framing not examined. | P4096R0 [3] |
| 2021 | Poll: sender/receiver "a good basis... including networking" - consensus in favor | P2300 deployments at Facebook, NVIDIA, Bloomberg | Deployments are for GPU dispatch, thread pools, and infrastructure. No sender-based networking deployment. No prototype. One hypothetical example. | P4097R0 [4] |
| 2014-2026 | Twenty claims shaped the trajectory; evidence documented where it exists | Various | Evidence concentrated in GPU/infrastructure domains. Networking evidence column empty for most claims. | P4098R0 [5] |

Each decision was locally reasonable, made by experienced practitioners under real constraints. The published record surveyed in this series is now assembled.

Jonathan Müller reported [28] from St. Louis: *"P2300 was adopted in the plenary vote and is now a part of the working draft which will become the C++26 standard, it was a very narrow vote with 1/3 voting against adoption."*

## 3. What the Committee Got Right

P2300R10 [8], "std::execution," is a genuine achievement. The sender/receiver model provides structured concurrency, sender composition, completion signatures as type-level contracts, and a customization point model that enables heterogeneous dispatch. P2470R0 [9] documented deployments at Facebook ("monthly users number in the billions"), NVIDIA ("fully invested in P2300... we plan to ship in production"), and Bloomberg (experimentation). GPU dispatch, infrastructure, HPC - the domains where compile-time work graphs, zero-allocation pipelines, and heterogeneous composition deliver their full value.

The people who built this - Eric Niebler, Kirk Shoop, Lewis Baker, Lee Howes, Michał Dominiak, and their collaborators - perceived structural problems in the executor concept and proposed a design that solved them. P1525R0 [10] documented gaps in `execute(F&&)` under the work framing. The sender/receiver model adopted by the committee addressed all four. It shipped.

The unification effort that preceded P2300R0 - P0443R14 [11], more than 100 papers, organizations spanning Google, NVIDIA, Sandia, Codeplay, Facebook, Microsoft, and RedHat - was a sustained effort to find common ground. The breadth of participation was extraordinary. The compromise was real.

These are facts. This series documents them alongside what was not examined.

---

## 4. What Is Now Available

None of the following existed when the decisions in Section 2 were made.

**C++20 coroutines** were ratified in 2020. The Coroutines TS was available from 2017 and compiler implementations existed by 2018, but the coroutine executor concept - which resolves the four deficiencies P1525R0 [10] identified - was not formulated until 2026. In 2014, when the unification decision was made, no standard coroutine facility existed. In 2019, when P1525R0 [10] diagnosed the basis operation, the specific two-framing analysis was not available. The coroutine executor concept constrains the handle type to `coroutine_handle<>`, restoring the type constraint that the rename from `dispatch`/`post`/`defer` to `execute(F&&)` removed.

**The coroutine executor concept** (P4003R3 [12], 2026) provides `dispatch` and `post` - continuation-scheduling primitives with a typed handle. The four deficiencies P1525R0 [10] identified do not arise under this concept (P4095R0 [2] Section 4).

**The two-framing distinction** - work framing vs. continuation framing - was documented in P4094R0 [1] Section 6. The continuation framing was the original framing (P0113R0 [13], 2015). The work framing replaced it through two stages of simplification. No published paper in the causal chain discussed the shift. The distinction is now available for the committee to apply.

**The rationale-loss mechanism** (P4094R0 [1] Section 6.3): API surfaces transfer between papers; design rationale does not, unless someone actively carries it forward. The continuation framing was carried by institutional knowledge rather than by the type system. When the property hint was removed by authors who did not carry that knowledge forward, the framing dropped out. This is a structural property of multi-author standardization.

**Interop bridges** (P4092R0 [14], P4093R0 [15]): the two models coexist and interoperate.

`std::execution::task` (P3552R3 [16]) is simultaneously a coroutine and a sender. The committee has already voted to ship a type that fuses two async models into one. The price of two models has been paid.

---

## 5. The Design Fork

| Awaitable | Sender |
|---|---|
| `struct read_awaitable`<br>`{`<br>`  bool await_ready();`<br>`  void await_suspend(`<br>`  std::coroutine_handle<> h);`<br>`  // caller erased`<br>`  io_result<size_t>`<br>`  await_resume();`<br>`};` | `template<class Receiver>`<br>`struct read_operation`<br>`{`<br>`  Receiver rcvr_;`<br>`  // caller stamped in`<br>`  void start() noexcept;`<br>`};` |

---

`coroutine_handle<>` erases the caller. `connect(sender, receiver)` stamps the caller into the operation state. The first choice produces type-erased streams, separate compilation, and ABI stability - properties that have been difficult to achieve in twenty years of networking attempts. The second produces full pipeline visibility, zero-allocation composition, and compile-time work graphs - the properties that make `std::execution` valuable for GPU dispatch, heterogeneous execution, and infrastructure. Neither model can acquire the other's properties without surrendering its own. The technical analysis is in P4088R0 [17].

`std::execution::task` (P3552R3 [16]) already fuses both models. The question is whether I/O facilities should also exploit each model's strengths independently.

One anonymous commenter during the 2021 electronic ballot wrote [29]: *"I don't think it's fair to consider standardizing S&R until there are at least a thousand codebases that use S&R. The probability of missing an important use-case, or an important gotcha is very very high if the actual quantity of 'Junior engineer + intern' experience in the field is low."*

The only published production I/O field report as of 2026 (P4125R1 [30]) describes a derivatives exchange that evaluated and rejected sender/receivers before choosing coroutine-native I/O. The partner's rationale: sender/receivers' structured concurrency trade-offs are at odds with their message-passing architecture, adoption would force a rewrite rather than an incremental port, and the pipeline expression syntax does not scale to their domain (Section 7.2).

---

## 6. Anticipated Objections

**Q: This is hindsight bias.**

A: The companion papers document what evidence was available at the time each decision was made. The questions in Sections 4 and 5 of P4094R0 [1] could have been asked in 2014. They were not. The two-framing analysis in P4095R0 [2] could not have been performed in 2019 because C++20 coroutines did not yet exist. This paper distinguishes between the two cases.

The evidence gap was also noticed contemporaneously. Müller wrote [28] at the time of adoption: *"Concerns were raised that maybe it wasn't reviewed properly, as committee members were not able to fully understand the intricate design details, and instead just trusted the authors that they did a good enough job."*

**Q: P2300 will eventually cover networking.**

A: It might. Prototype implementations exist - Kühl's P2762R2 [18] work and experimental library, Voutilainen's Qt + P2300R0 networking integration - but no production networking deployment on senders has been published as of 2026. The one published production evaluation (P4125R1 [30]) rejected sender/receivers as architecturally incompatible with their domain. The committee now has two models to evaluate. Time will tell which serves networking better, or whether both do.

**Q: Two models fragment the ecosystem.**

A: The committee has shipped multiple design approaches for the same domain before: `iostream` and `std::format`/`std::print`; C++17 execution policies and P2300R0 senders with `bulk`. `std::execution::task` (P3552R3 [16]) is simultaneously a coroutine and a sender. The committee has already chosen coexistence.

**Q: You are arguing for your own library.**

A: Section 1 discloses this. The causal chain in Section 2 is assembled from the published record. The evidence stands or falls on the sources, not on who assembled them.

---

## Acknowledgments

The author thanks Chris Kohlhoff for the executor model that started the journey, for twenty years of Boost.Asio deployment, for the candid retrospective in P1791R0 [19], and for N3747 [20]'s universal async model; Eric Niebler, Kirk Shoop, Lewis Baker, and Lee Howes for the sender/receiver model, for P1525R0 [10]'s structural insights, and for P2300R10 [8]; Michał Dominiak for the editorial work that brought P2300R10 [8] to adoption; Jared Hoberock, Michael Garland, and Chris Mysen for P0443R14 [11], P0761R2 [21], and years of compromise in pursuit of a unified model; Ville Voutilainen for P2464R0 [22]'s analytical framework, which provided the evaluation criteria this series applies; Bryce Adelstein Lelbach for the published poll outcomes in P2453R0 [23] and P2400R2 [24] that made this analysis possible; Dietmar Kühl for P2762R2 [18] and `beman::execution`; Detlef Vollmann for P1256R0 [25]; Jonathan Müller for P3801R0 [26]; Peter Dimov for technical corrections on earlier drafts; Steve Gerbino and Mungo Gill for Capy [6] and Corosio [7] implementation work; Klemens Morgenstern for Boost.Cobalt and the cross-library bridge examples; Jamie Allsop and Richard Hodges for co-authoring P2469R0 [27]; and the national body members who raised concerns at St. Louis.

The effort to bring async programming to C++ has been genuine, sustained, and conducted by people the author respects. This series documents the record. The committee will decide what to do with it.

---

## References


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

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

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

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

[5] P4098R0 - "Retrospective: Claims and Evidence" (Vinnie Falco, 2026).

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

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

[8] P2300R10 - "std::execution" (Michał Dominiak et al., 2024).

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

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

[11] P0443R14 - "A Unified Executors Proposal for C++" (Jared Hoberock, Michael Garland, Chris Kohlhoff, Chris Mysen, Carter Edwards, 2020).

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

[13] P0113R0 - "Executors and Asynchronous Operations, Revision 2" (Christopher Kohlhoff, 2015).

[14] P4092R0 - "Consuming Senders from Coroutine-Native Code" (Vinnie Falco, Steve Gerbino, 2026).

[15] P4093R0 - "Producing Senders from Coroutine-Native Code" (Vinnie Falco, Steve Gerbino, 2026).

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

[17] P4088R0 - "What C++20 Coroutines Already Buy The Standard" (Vinnie Falco, 2026).

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

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

[20] N3747 - "A Universal Model for Asynchronous Operations" (Christopher Kohlhoff, 2013).

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

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

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

[24] P2400R2 - "Library Evolution Report: 2021-06-01 to 2021-09-20" (Bryce Adelstein Lelbach, 2021).

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

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

[27] 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).

[28] Trip Report: Summer ISO C++ Meeting in St. Louis, USA - Jonathan Müller, July 2024.

[29] r/cpp: C++ committee polling results for asynchronous programming - Oct 2021.

[30] P4125R1 - "Coroutine-Native I/O at a Derivatives Exchange" (Mungo Gill, 2026).