Doc. No.: P4029R1

Date: 2026-02-26
Audience: SG14
Author: Michael Wong
Reply to: fraggamuffin@gmail.com
Contributor: SG14

The SG14 Priority List for C++29/32

Revision History

- **R1 (2026-02-26):**
  - Revised Section 2 to articulate architectural requirements rather than endorse specific proposals, consistent with SG14's role as a constituency voice on constraints and requirements.
  - Corrected Section 4.2: P1112 was rejected by EWG (thanks Hana) and the relevant portions of P1847 have been merged into the C++26 draft. These are no longer listed as open priorities. The underlying SG14 interest in standard memory layout control is preserved as a future direction pending new proposals. General editorial improvements throughout.
- **R0 (2026-02-23):**
  - Initial publication with improper attributions.

Abstract

This document outlines the strategic priorities for SG14 (Low Latency, Games, Finance, and Embedded) for the C++29 and C++32 cycles. While SG14 shares the broader committee's goals regarding safety and asynchronous execution, this constituency operates under strict constraints: zero-overhead abstractions, predictable latency, and deterministic execution. This paper defines how those constraints must shape upcoming standard features.

---

1. The "Elephant in the Room": Safety

SG14 uniformly endorses and requests an Embedded "Safe C++ / Profiles" as the inescapable context for C++29 to augment C++ freestanding.

SG14 Stance: While we acknowledge Safety is the global priority, SG14's specific concern is ensuring safety features do not introduce runtime overhead or latency spikes (e.g., hidden allocations or locks). We view "Deterministic Exception Handling" as our specific contribution to the safety conversation.

---

2. Networking and Asynchronous Execution: Architectural Requirements

SG14 constituencies — game engines, financial trading systems, embedded devices, and real-time systems — impose hard constraints on any networking and asynchronous execution model adopted for C++29. This section describes those constraints. SG14 takes no position on which specific proposal best satisfies them, but asks that any adopted model be evaluated against these requirements.

2.1 Zero Dynamic Allocation on the Hot Path

Low-latency networking code cannot tolerate dynamic memory allocation during steady-state operation. Any networking model that requires heap allocation per operation, per connection, or per completion is unsuitable for SG14 constituencies. Allocation must be controllable by the user: either eliminated entirely through static buffers and fixed-capacity structures, or confined to initialization and teardown phases.

2.2 Predictable Latency and Transparent Cost Model

SG14 constituencies require that the cost of each asynchronous operation be visible and predictable at the call site. Models that introduce hidden queuing, type erasure, or indirect dispatch in the completion path impose costs that are unacceptable in sub-microsecond execution environments. The programmer must be able to reason about the generated code path from initiation to completion without opaque abstraction boundaries.

2.3 Coroutine Compatibility Without Coroutine Dependence

Many SG14 use cases benefit from coroutine ergonomics for expressing asynchronous sequences. However, the networking model must not require coroutine frames for basic operations. Callback and completion-token patterns remain essential for the lowest-latency use cases. The ideal model supports coroutines as an optional integration layer rather than a structural dependency.

2.4 Direct-Style I/O as an Architectural Direction

SG14 observes that the networking patterns with the longest production track record in low-latency C++ — including shipping implementations used in finance, games, and embedded systems — follow a direct-style model: the programmer initiates an operation and receives its result through a coroutine suspension or a completion handler, with the library managing I/O readiness transparently.

SG14 recommends that WG21 evaluate networking proposals for C++29 against this direct-style model. Proposals that layer networking on top of a general-purpose asynchronous execution framework should demonstrate that they can meet the constraints in Sections 2.1 through 2.3 without requiring SG14 constituencies to bypass the framework to achieve acceptable performance.

2.5 Decouple Networking from General-Purpose Async Frameworks

SG14 advises that Networking (SG4) should be evaluable on its own merits as a networking facility, independent of which general-purpose asynchronous execution model WG21 adopts. Tightly coupling networking to any single execution framework creates a dependency that may force SG14 constituencies to choose between using the standard networking facility and meeting their latency requirements. SG14 asks that the committee preserve the option of a networking model that interoperates with, but does not depend upon, the chosen execution framework.

---

3. Prioritize Making C++ Better for Game Developers

The game development industry is a massive consumer of C++, driving hardware utilization to its absolute limits. To ensure C++ remains the undisputed language for AAA game engines, SG14 champions the features outlined in P2966R1 (Prioritize Making C++ Better for Game Developers).

Specifically, we advocate for the immediate prioritization of the following proposals, which provide high value with minimal implementation risk:

3.1 P3442R1 ([[invalidate_dereferencing]] attribute)

Game developers rely heavily on custom allocators and manual memory management. This attribute bridges the gap between manual management and static safety. By allowing developers to explicitly annotate when a pointer is logically invalidated, it supercharges static analysis tools (and the upcoming Safety Profiles) to detect use-after-free bugs at compile-time, without imposing any runtime checking overhead.

3.2 P3707R0 (std::is_always_exhaustive trait)

Game logic is fundamentally state-driven, heavily relying on switch statements and variant visiting over complex enums. Providing a standard library trait to enforce exhaustiveness at compile-time eliminates a massive category of subtle runtime logic bugs (unhandled states). This is a pure compile-time safety feature that aligns perfectly with the zero-overhead philosophy.

---

4. Ultra Low-Latency and High-Frequency Trading (HFT)

Financial engines operate under strict constraints: they require sub-microsecond predictability, zero dynamic memory allocation on the "hot path," perfect cache locality, and exact decimal arithmetic.

4.1 Lock-Free Concurrency & Messaging

In low-latency finance, different threads (e.g., market data ingestion vs. order execution) must communicate without OS-level locks or blocking.

Ring Buffer (ring_span) (P0059): Tracked specifically for "Games, Finance, Embedded." A ring buffer (circular buffer) is the backbone of HFT message passing because it operates on a fixed-capacity contiguous memory block, meaning it never allocates memory dynamically after initialization.

Concurrent Queues (P0260): Standardizing lock-free and concurrent queues is critical for fast thread-to-thread communication without rolling custom, error-prone atomics.

Hazard Pointers & RCU: Slated for C++26 with extensions for C++29, these are essential mechanisms for safely reclaiming memory in lock-free data structures (e.g., order books) without blocking reader threads.

4.2 Cache Locality & Memory Layout

Predictable memory access is often more important than algorithmic complexity in low-latency systems.

Member Layout Control: SG14 has a long-standing interest in giving developers standard control over struct layout to maximize CPU cache line utilization — including automatic packing, field reordering, and AoS-to-SoA transformations. Previous proposals in this space (P1112, P1847) have either been rejected by EWG or had their applicable portions merged into the C++26 working draft. SG14 considers the problem space still underserved: the merged provisions in C++26 address a subset of the use cases, but fine-grained, portable, zero-overhead layout control for performance-critical structures remains an open need. SG14

encourages new proposals in this area for C++29, potentially informed by reflection capabilities (P2996) and emerging practice in data-oriented design.

Object Relocation / Trivial Relocatability (P1144 / P2786): Tracked with interest from multiple constituencies including financial firms. Allows the compiler to use memcpy to move objects in memory rather than calling expensive move constructors, significantly accelerating container reallocations.

---

5. Ongoing Watch List

SG14 maintains a living tracking document of features, issues, defects, and their status across all SG14 domains:

https://docs.google.com/spreadsheets/d/1JnUJBO72QVURttkKr7gn0_WjP--P0vAne8JBfzbRiy0/edit?gid=0#gid=0

This spreadsheet reflects the full scope of SG14's interests and is updated as proposals progress through the committee pipeline. Items include freestanding library work (P1642, P2013, P2338, P2407, P2833, P2937, P2976), linear algebra (P1385, P1673), deterministic exceptions (P0709), hive/colony containers (P0447), fixed-capacity data structures (P0843), numerics and fixed-point arithmetic (P0037, P1889), graph data structures (P1709), physical units (P1935), and game developer priorities (P2966).

SG14 encourages proposal authors working in these domains to coordinate with us to ensure low-latency, embedded, and real-time constraints are considered early in the design process.