Doc. No. P4029R0
Date: 2026-02-23
Audience: SG14
Author: Michael Wong
Reply to: fraggamuffin@gmail.com
Contributor: SG14

The SG14 Priority List for C++29/32

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 asks for an Embedded **"Safe C++ / Profiles"** as the inescapable context for C++29 to augment C++ freestanding:

- **SG14 Stance:** While they 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). They view "Deterministic Exception Handling" as their specific contribution to the safety conversation.

2. Executors/coroutine backward flow + lazy execution vs direct forward flow model for CPU-bound I/O and Networking.

1. Decouple Networking from `std::execution` 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.

2. Standardize "Direct Style" I/O Prioritize **P4003** (or similar Direct Style concepts) as the C++29 Networking model. It offers the performance of Asio/Beast with the ergonomics of Coroutines, maintaining the "Zero-Overhead" principle.

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:

- **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.
- **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.

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 (like order books) without blocking reader threads.

2. Cache Locality & Memory Layout

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

- **Member Layout Control (P1112 / P1847):** Tracked for "Games, Finance, Embedded". This feature aims to give developers standard control over struct layout (e.g., automatically packing data or converting Arrays-of-Structs to Structs-of-Arrays) to maximize CPU cache line utilization.

- **Object Relocation / Trivial Relocatability (P1144 / P2786):** Tracked with interest from financial firms like Bloomberg. It allows the compiler to use `memcpy` to move objects in memory rather than calling expensive move constructors, massively speeding up vector reallocations.

5. Any features on our ongoing watch list:

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