C++ Stability Diagnostic Report

Project: CCM CODE — Legacy C++ Diagnostics
Target: sensor_pipeline
Date: 2026-02-17
Toolchain: Valgrind 3.22.0 (Memcheck + Helgrind)
Build: g++ -std=c++11 -g

RISK SCORE
8.4
/ 10.0
ACTION REQUIRED

Executive Summary

The sensor_pipeline module exhibits **confirmed memory leaks on error paths** and **multiple unsynchronised accesses** to shared state across three threads.

One finding—an unsignalled running flag write during shutdown—creates a window for use-after-free behaviour and is classified as **CRITICAL**. Two pthread thread-stack entries have been reviewed and classified as expected runtime noise; they do not represent application defects.

Severity Breakdown

1	4	0	2
CRITICAL	HIGH	MEDIUM	LOW / INFO

Findings Summary

	Finding	Location	Category	Severity
F-01	Shutdown race on running flag	destroy_pipeline():123	Data Race	**CRITICAL**
F-02	Memory leak on out-of-range discard — struct	create_reading():38	Memory Leak	**HIGH**
F-03	Memory leak on out-of-range discard — label	create_reading():42	Memory Leak	**HIGH**
F-04	Unsynchronised read of count in flush thread	flush_thread():83	Data Race	**HIGH**

F-05	Unsynchronised read of count in alert thread	alert_thread():98	Data Race	**HIGH**
F-06	Unsynchronised std::cout across threads	flush_thread():85, main():142	Data Race	**LOW**
F-07	pthread thread-stack allocation (expected)	main():132, 133	Noise	**INFO**

Detailed Findings & Fixes

F-01 — Shutdown Race on running Flag

CRITICAL

Root Cause: destroy_pipeline() writes running = false without holding any lock and without calling pthread_join(). Background threads reading this flag may attempt to access g_pipeline.readings after it has been deleted.

Valgrind Trace:

```
Possible data race during write of size 1 at 0x10C28C by thread #1
  Locks held: none
   at 0x10973F: destroy_pipeline() (sensor_pipeline.cpp:123)
   by 0x10998C: main (sensor_pipeline.cpp:146)

This conflicts with a previous read of size 1 by thread #2
  Locks held: none
   at 0x1095DA: flush_thread(void*) (sensor_pipeline.cpp:81)
```

Recommended Fix — destroy_pipeline():

C++

```cpp
void destroy_pipeline() {
    // Signal threads to stop — atomically
    pthread_mutex_lock(&g_pipeline.lock);
    g_pipeline.running = false;
    pthread_mutex_unlock(&g_pipeline.lock);

    // Wait for threads to exit before freeing shared memory
    pthread_join(g_flusher, nullptr);
    pthread_join(g_alerter, nullptr);

    delete[] g_pipeline.readings;
    pthread_mutex_destroy(&g_pipeline.lock);
}
```

F-02 / F-03 — Memory Leak on Validation Early Return

HIGH

Root Cause: When temperature range checks fail in process_reading(), the function returns false without freeing the SensorReading struct or the heap-allocated label string.

Valgrind Trace:

```
30 bytes in 4 blocks are definitely lost
   at create_reading() (sensor_pipeline.cpp:42)  ← label char[]

32 bytes in 1 block are definitely lost
   at create_reading() (sensor_pipeline.cpp:38)  ← SensorReading struct

  Total definitely lost: 54 bytes in 5 blocks
```

Recommended Fix — process_reading():

C++

```
bool process_reading(SensorReading* reading) {
    if (reading == nullptr) return false;

    if (reading->temperature > 150.0f || reading->temperature < -40.0f) {
        std::cerr << "[WARN] Out-of-range reading discarded.\n";
        delete[] reading->label;   // free label string
        delete reading;            // free struct
        return false;
    }
    // ...
}
```

F-04 / F-05 — Unsynchronised Reads of count

HIGH

Root Cause: flush_thread and alert_thread read g_pipeline.count without acquiring a mutex, while process_reading() writes to it under lock. This can cause out-of-bounds array access if the value is updated mid-read.

Valgrind Trace:

```
Possible data race during read of size 4 at 0x10C288 by thread #2

  Locks held: none

   at flush_thread(void*) (sensor_pipeline.cpp:83)



This conflicts with a previous write of size 4 by thread #1

  Locks held: 1, at address 0x10C290  ← mutex IS held on write

   at process_reading(SensorReading*) (sensor_pipeline.cpp:60)
```

Recommendation: Acquire the same mutex on the read side, or upgrade the counter to std::atomic<int>.

F-06 & F-07 — Output Race & Runtime Noise Low / Info

- **F-06:** Concurrency on std::cout produces garbled logs but is not system-critical.
- **F-07:** pthread_create stack allocations are flagged as "possibly lost" by Valgrind; this is standard GLIBC behavior and will resolve once pthread_join is implemented in F-01.

Prioritised Remediation Roadmap

1. **Join Threads:** Add pthread_join to destroy_pipeline() to stop the use-after-free and clear stack noise.
2. **Clean Early Returns:** Add delete calls to error paths in process_reading() to stop the memory leaks.
3. **Modernize Atomics:** Replace bool running and int count with std::atomic types to eliminate lock contention on reads.
4. **RAII Patterns:** Replace raw char* with std::string and use std::unique_ptr for sensor readings to prevent future leaks.
5. **Verification:** Re-run Memcheck and Helgrind. Target: **0 errors from 0 contexts**.

CCM CODE | Valgrind 3.22.0 · Memcheck + Helgrind | 11 errors identified (2 noise)