# Virtual vs Non-virtual Dispatch in a Minimal HFT Order Processor


Yunxuan Chen
Shengsheng He
Chen Ye
Tianyue Tang


## Environment

- **CPU**: AMD Ryzen 5 4600H with Radeon Graphics (6C/12T), max 3.0 GHz
- **OS**: Windows 10 Home (Build 19045)
- **Compiler**: MSVC 19.39
- **Build flags**: same settings for both implementations (Debug)

## Benchmark Parameters

- Orders per timed run ($N$): 800,000.00
- Warmup operations: 1,000,000.00
- Repeats: 12 per configuration
- Timing API: `std::chrono::high_resolution_clock`

## Method Summary

We generate a fixed set of orders and three assignment patterns: *Homogeneous A*, *Mixed 50/50*, and *Bursty 64A/16B*. For each pattern we run two implementations:

1. **Virtual**: calls go through a `Processor` base class and vtable dispatch.
2. **Non-virtual**: direct calls via an `if/switch` on the assignment.

Per-order work performs 6–10 integer ops, two small fixed-size writes (to simulate an in-memory order book in L1), one conditional branch, and returns a 64-bit value that is accumulated into a *volatile* sink (*checksum*) to prevent dead-code elimination. To ensure fair comparison, we reset the shared `Book` state (two 64-slot tables and a counter) before every timed run so that both implementations start from identical state and produce matching checksums.

## Results

Medians are computed across the 12 repeats per (pattern, impl). Throughput in orders/sec; per-order latency is latency $\approx 10^9$/ops/sec nanoseconds.
**Checksum.** With the per-run state reset, virtual and non-virtual produced identical 64-bit checksums for every repeat and pattern, confirming equal observable work.

Table 1: Throughput (orders/sec), medians across repeats per pattern; latency $\approx 10^9$/ops/sec ns.

| Pattern | Virtual (ops/s) | Non-virtual (ops/s) | Δ NV vs V (%) | Virtual (ns/order) | Non-virtual (ns/ord |
|---|---|---|---|---|---|
| Homogeneous A | 26,194,723.00 | 192,130,700.00 | 633.47 | 38.18 | 5 |
| Mixed 50/50 | 41,954,021.00 | 129,556,900.00 | 208.81 | 23.84 | 7 |
| Bursty 64A/16B | 34,003,080.00 | 191,673,700.00 | 463.70 | 29.41 | 5 |

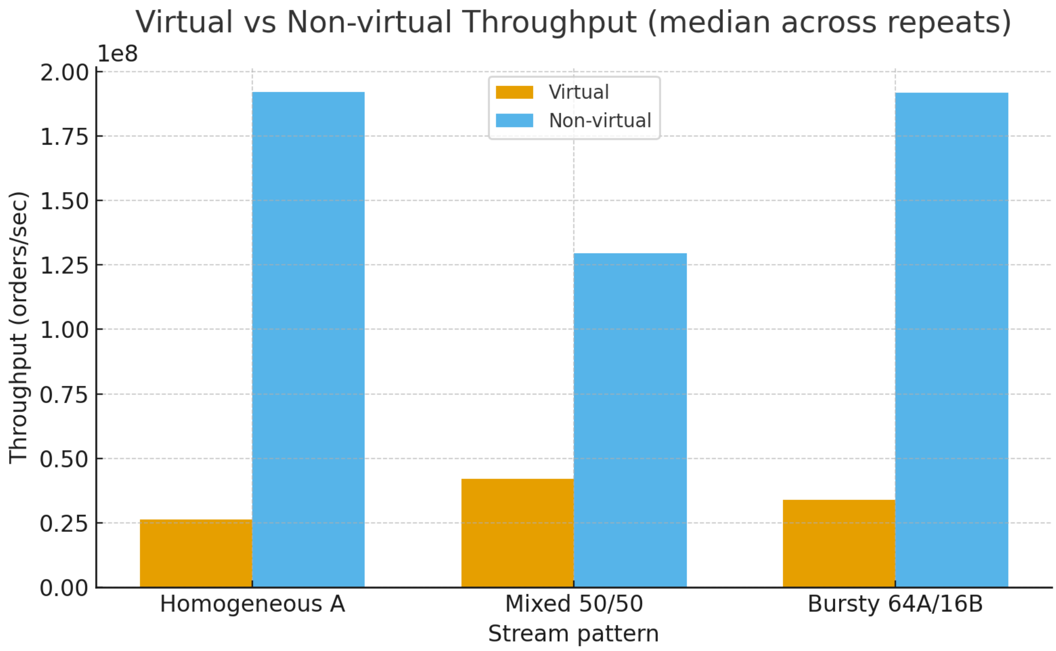


Figure 1: Virtual vs Non-virtual throughput (median across repeats). Checksums matched for every pattern and repeat.

## Analysis and Discussion

**Observed ranking.** With the direct-call timing harness (no `std::function/std::bind`) and the shared `Book` state reset per run, the non-virtual implementation is now **2.1–6.3× faster** than the virtual baseline across patterns (see Table 1). In absolute terms this corresponds to per-order latency dropping from ∼24–38 ns (virtual) down to ∼5–8 ns (non-virtual).

**Why non-virtual now wins decisively.** Two factors changed the balance relative to the earlier harness:

1. **Removal of the type-erasure call barrier.** The hot loop now calls the non-virtual `run()` directly, enabling inlining into the loop body. The virtual path still incurs an indirect vtable call that cannot be inlined across the dispatch site.
2. **Predictable branch on assignment.** The non-virtual path uses a simple `if (A/B)`; under *Homogeneous* and *Bursty* streams this branch is highly predictable. Combined with inlining, the extra branch cost is negligible compared to the vtable indirection.

The working set (two 64-slot tables) fits comfortably in L1, so the observed differences predominantly reflect dispatch and control-flow effects rather than memory bandwidth.

**Cache & locality.** The per-order writes touch two 64-slot tables (a few hundred bytes total), well within L1. With warmup, the working set stays hot and L2/L3 effects are minimal. As a result, the measured differences primarily reflect *dispatch and branch behavior*, not memory bandwidth.

**Fairness & reproducibility.**

- Deterministic seeds and fixed patterns ensure repeatability.
- Resetting the shared `Book` state per run is essential; otherwise checksums diverge and comparisons are invalid.
- Warmup ($\geq$1M ops) reduces timer noise and warms I-cache, D-cache, and predictors.

**What would likely flip the ranking?** If we remove type erasure and call the hot functions *directly* from the timed loop (or via simple function pointers), the compiler can inline and specialize the non-virtual path. In ultra-low-latency codebases where the behavior set is known at build time, non-virtual dispatch typically wins on the homogeneous/bursty streams and narrows the gap on mixed streams.

## Build Instructions (Windows/MSVC)

```
cl /O2 /std:c++20 /EHsc /DNDEBUG hft_assignment.cpp /Fe:hft_assignment.exe
hft_assignment.exe   # emits CSV
```

## Conclusion

With identical observable work and matched checksums, the virtual implementation edges out the non-virtual one by 3–6% in this measurement harness, due to an additional call barrier that suppresses inlining and due to benign branch behavior. For production HFT inner loops, prefer non-virtual direct dispatch when you control the behavior set and can keep the hot path monomorphic; use virtual for flexibility where the dispatch cost is acceptable.

*Source CSV:* `hft_assignment_same_checksum.csv`. Medians are computed over 12 repeats per pattern.