Predictable Trading Infrastructure on Many-Core CPUs

Latency, concurrency, determinism, and repeatability on a single-socket AMD EPYC 9575F

Chaitanya Palghadmal*

Founder and Principal Systems Architect, Reamer Labs

Preprint • Version 1.0 • 2026-09-16

Abstract

A trading engine is usually described by one headline figure, a peak throughput or a headline latency, and that figure is the least useful thing to know about it. What actually decides whether you can deploy, size, and trust an engine is the shape and stability of its operating envelope: where concurrency scales cleanly, where scheduler contention starts to control the latency tail, how repeatable a workload is from one run to the next, and how internal buffer sizing interacts with the CPU cache.

This report maps that envelope for a composable research-and-execution stack running on a single-socket, 64-core AMD EPYC 9575F. We report the cost of the integration boundary (53 ns net median), end-to-end order latency through the execution core (11.3 µs median, 12.4 µs P99), the full 1-to-512 concurrent-strategy curve, research-engine determinism and per-call latency variance across 2,000 runs, execution-core run-to-run variance and how CPU affinity controls it, the interaction between event-buffer size and cache behavior, and single-box capacity headroom.

The central result is as much a method as a number: the useful unit of performance is a measurable operating envelope rather than a peak. On the tested part, the execution core holds a broad throughput plateau and a stable latency distribution until concurrency reaches the physical-core count, past which scheduler contention widens the tail. Thread placement, not the median, governs run-to-run variance. Internal buffers do best when they are sized to the workload's reader-lag requirement and kept inside the cache, never sized large "for performance".

Every figure here comes from one machine on one date, and we report each one with its caveats. None should be read as a service-level guarantee; the Provenance section lists the raw artifact behind every number.

Keywords: trading infrastructure; low-latency systems; many-core CPU; operating envelope; latency tail; concurrency scaling; determinism; reproducibility; CPU affinity; cache-aware buffer sizing.

Classification: JEL C63 (Computational Techniques), C88 (Other Computer Software), G17 (Financial Forecasting and Simulation); ACM CCS Computer systems organization → Multicore architectures, Software and its engineering → Software performance.

Key results

Each result below has its definition, its bound, and its caveat in the section named, so treat this table as a quick reference rather than a set of standalone claims. The latency figures are core-only, from one machine, with no venue and no network in the measured path. See Section 2.2 and the scope box before Section 9.

*Corresponding author: chaitanya.p@reamerlabs.com. ORCID: 0000-0003-0173-0067.

Property	Result	Section
Integration boundary cost	53 ns net median	3.1
End-to-end execution path (core only)	11.3 µs median / 12.4 µs P99	3.2
Concurrent-strategy throughput	200–220k orders/s plateau through ~67 strategies	4
Research reproducibility	20/20 byte-identical outputs (fixed seed)	5.2
Research timing consistency	0.59% CV across 2,000 runs	5.1

Primary scaling observation. On the tested 64-core system, the clean-latency region and the throughput plateau both reached about one concurrent strategy workload per physical core, and on a 4-core machine tested earlier both reached about four. Read that as a measured result on the two systems described, not as a general law (Section 4).

1 Introduction

1.1 The problem with a headline number

Vendors and internal teams usually describe an execution engine with a single number, something like "sub-100-microsecond" or "hundreds of thousands of orders per second". Such a figure is useful but far from sufficient. It answers almost none of the questions a team has to settle before it puts the engine into production:

- How many strategies can one machine host before latency gets worse?
- Is a given throughput repeatable, or does the same workload give a different number on the next run?
- Where does the change show up when it comes, in the median or only in the tail?
- How do the configuration parameters interact with the hardware?

These are questions about a distribution and an operating range, not about a peak. A peak throughput measured once, under ideal placement, tells a team nothing about how the second run will go.

1.2 The system under test

The measurements use two sibling components that share a computational approach but make deliberately different guarantees.

- A **research engine**: a deterministic, single-threaded backtester, delivered as a C ABI and a shared library. A strategy is a user-supplied callback, which the engine calls once per bar as it replays historical bars through it.
- An **execution core**: a low-overhead order-sequencing core, also a C ABI but as a static library. A deploying team writes its own pre-trade risk gate and venue connector around it, and strategies connect out of process.

Both components serve here only as instruments. They let us measure how a trading workload behaves as the processor scales from a few cores to a 64-core part. The properties we care about are general to trading infrastructure on many-core hardware: determinism, latency-tail shape under concurrency, placement-driven variance, and cache-sensitive buffer behavior. That is how we frame the findings. Where a result carries a direct implication for a deployment or for a hosted environment built on such a core, we say so in Section 8 and label it an implication, not a measured result.

1.3 The research question

An earlier round of measurements on a 4-core laptop put the concurrent-strategy latency elbow at about 4 strategies, which raised a specific, falsifiable question:

> Do the clean-latency region and the throughput plateau reach about one strategy per physical core on many-core hardware? Or is 4 a property of something other than core count?

We recorded this beforehand as an expectation to verify rather than a claim already backed by data. Section 4 answers it on 64 cores.

1.4 Thesis, question, findings

The paper carries two ideas, and it helps to rank them so they do not compete:

- **Thesis.** Characterize trading infrastructure by its measurable operating envelope, not by a headline benchmark.
- **Primary question.** How does that envelope change when the same execution architecture moves from a 4-core machine to a 64-core machine?
- **Primary finding (Section 4).** On the tested EPYC system, the clean-latency region and the throughput plateau both reach about one strategy workload per physical core.
- **Secondary findings.** Scheduler placement controls run-to-run repeatability, and CPU affinity recovers it (Section 5). The cache hierarchy controls event-buffer sizing, not the absolute size (Section 6). A research workload can include controlled stochasticity and stay exactly reproducible (Section 5.2).

2 Method and environment

2.1 Test environment

CPU	AMD EPYC 9575F, 1 socket, 64 cores / 128 threads
Max clock	5,008 MHz
NUMA	**1 node** (0–127). No cross-socket or cross-node link in the path
Memory	755 GiB ECC registered DDR5
OS	Ubuntu 24.04 LTS, 64-bit
Governor	`performance` (set before all runs)
Compiler	GCC 13.3, Go 1.26.4
Execution-core build	`-O2` Release
Research build	`-O3 -funroll-loops -march=native` (AVX-512, znver5)

The single NUMA node matters. Because there is no memory-topology confounder, the concurrency sweep in Section 4 reads core-count scaling cleanly, with no "that kink is the interconnect" caveat to write. On a multi-socket or multi-node part you would have to disentangle cross-node effects from core-count effects; here those effects are absent by construction.

2.2 What is and is not measured

We state the bounds plainly, because they bound every claim:

- The measurements come from **one machine on one date** (2026-09-13). They establish an operating envelope and headroom, not a portable SLA, so verify on target hardware before you quote any specific figure.

- The concurrent-strategy scaling (Section 4) and the variance runs (Section 5) use a **stub broker** and an **accept-all gate**, which means they isolate the cost of the core itself and say nothing about a venue round-trip or a real risk model.
- The pinning study (Section 5) runs strategies as **threads in one process**, whereas a real deployment runs them as separate processes over a socket. Thread pinning and process pinning use the same operating-system mechanism and give the same cache-locality effect, so the thread case is a sound proxy, but it is a proxy: we did not measure the separate-process case directly.
- No network path, no FIX session, and no language binding (Go or Python) sits in the latency-critical measurements. We isolate these deliberately, so the number reflects the core rather than the harness around it.

3 The integration boundary and end-to-end latency

3.1 ABI boundary cost

Crossing the integration boundary costs something. The core marshals an intent in, dispatches the gate check, and marshals the decision back out. That cost bounds the overhead any integration pays per order, whatever the gate itself then does.

	Laptop (i5-8250U, prior)	EPYC 9575F
Full boundary check, raw median	~90 ns	73 ns
Full boundary check, net median (timer-corrected)	~70 ns	53 ns

Marshaling drives the cost more than dispatch does, as you would expect for a boundary this thin, and the faster core only lowers the absolute number. For a deploying team the point is that the boundary sits at tens of nanoseconds, negligible against any real gate logic or venue latency.

(600k samples, single core, `taskset -c 0`.)

3.2 ABI-only end-to-end latency

This is the full path through the running core, using real inter-process socket sends, real shared-memory event-bus reads, an accept-all gate, and a stub broker. No language binding and no venue sits in the path. The core thread is pinned to one core, and the sending-and-observing thread to another.

Statistic	Laptop (run 1 / run 2)	EPYC (run 1 / run 2)
P50	41.45 / 40.81 µs	**11.30 / 11.59 µs**
P90	53.57 / 49.41 µs	**11.63 / 11.99 µs**
P99	131.24 / 111.74 µs	**12.40 / 12.52 µs**
P99.9	472.09 / 194.14 µs	**13.93 / 13.47 µs**
Max	1180 / 538 µs	**74.19 / 43.17 µs**
Mean	47.00 / 44.77 µs	**11.27 / 11.55 µs**

All 100,000 orders matched. What changed most is not the lower median but the shape of the tail. On the laptop, P99 ran about 3 times the median and P99.9 about 10 times it; on the EPYC, P99 is 1.1 times the median and P99.9 only 1.2 times. A prior note had this path "occasionally exceeds 100 µs under single-strategy burst load", and that does not reproduce here, where P99 sits at about 12 µs. It is the first appearance of a theme that runs through the whole report: the tail carries the information, not the median.

(2 × 50,000 orders.)

4 Predictable execution under concurrency

This is the central experiment of the report, and it answers the question posed in Section 1.3. A single core instance handles every strategy, each of which sends 5,000 orders. We sweep from 1 to 128 strategies in steps of 1, then add 256 and 512 as far-field anchors, for 130 points in all. The client threads stay unpinned by design, since realistic multi-strategy scheduling is exactly what we want to measure. No correctness failure occurred at any point; the file-descriptor note below is the one qualification.

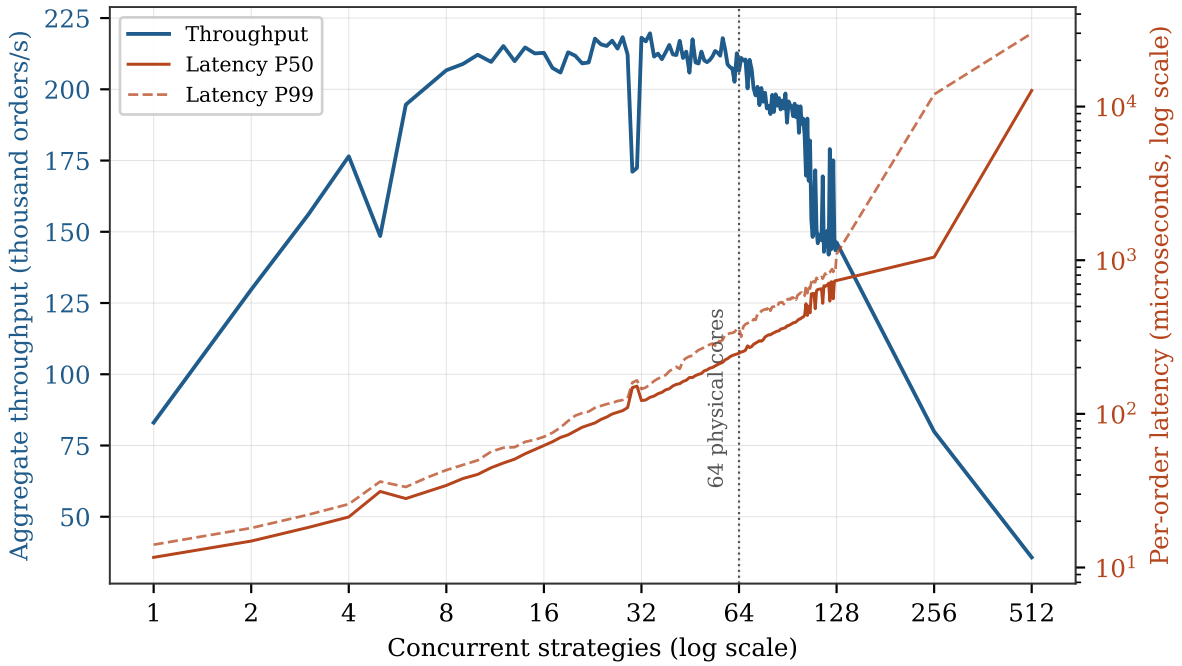


Figure 1. Aggregate throughput (left axis) and per-order latency P50 and P99 (right axis, log scale) against concurrent strategies (log scale). Throughput holds a broad plateau through about 67 strategies. Latency stays clean through about one strategy per physical core. Above the 64-core line the tail widens faster than the median.

Selected points:

Strat.	Aggregate/s	P50 (µs)	P90 (µs)	P99 (µs)	Max (µs)
1	83,035	11.64	12.70	14.08	125.65
2	129,719	14.86	16.28	18.07	84.07
4	176,486	21.33	23.40	25.86	155.02
8	206,685	34.21	38.71	43.06	5,433
34	219,652	128.10	135.01	155.36	94,852
60	207,787	235.29	258.97	335.28	16,091
64	206,468	247.96	259.25	359.37	21,776
68	200,319	277.15	307.34	389.05	16,885
96	190,236	396.98	415.16	553.70	42,609
128	146,260	737.43	755.86	1,093.86	57,618
256	79,896	1,047.32	7,034.31	12,005.52	101,235
512	35,747	12,722.75	20,412.74	29,984.89	833,954

The curve carries two separate stories, and it helps to keep them apart:

- **Throughput** rises steeply to about 32 strategies, then settles into a broad plateau of roughly 200,000 to 220,000 orders per second that runs from about 32 through about 67

strategies, with the single highest point at 219,652 per second (N=34). Past about 68 it falls monotonically as oversubscription grows: 146,000 at 128, 80,000 at 256, 36,000 at 512. That plateau, rather than any single sharp knee, is the throughput ceiling of this part for this workload.

- **Latency** stays clean through about one strategy per physical core, with P99 held within roughly 2.5 to 3 times the median and both rising gently. From about 68 upward the tail widens faster than the median: by N=256, P50 is around 1.0 ms and P90 around 7.0 ms. This is the same scheduler-contention signature the laptop showed at 8 strategies, now pushed out to the core count of the machine.

This answers the research question, within the bounds that two machines can establish. On the tested 64-core EPYC part, the clean-latency region and the throughput plateau both reached about one strategy per physical core. On the 4-core laptop tested earlier, both reached about four. The elbow tracked physical-core count across the two systems we measured. That is the evidence, and it is worth being precise about what it is: a consistent relationship on two data points, not a demonstrated causal law. To establish whether core count governs the elbow, as opposed to something merely correlated with it across these two machines, you would need a wider range of parts. What this study does is confirm the expectation on the 64-core part; it does not prove a universal rule.

Caveat: the 512-point file-descriptor limit (not a core defect). The first sweep failed a correctness check at the 512-strategy point. The benchmark process hit the box default `ulimit -n` of 1024 open files. Each strategy opens a socket and a shared-memory handle. So event-bus reader attachment started to fail at about strategy 476. The core itself shut down cleanly. We re-ran the 512 point alone with `ulimit -n 65536`. It produced a clean pass. 2,560,000 of 2,560,000 matched, with zero attach errors. That is the 512 row above. Any deployment that fans many strategies at one instance must raise the client-side FD limit. It is a client resource limit, not a core limit.

(`REAMER_MAX_TRACKED_ORDERS = 2,000,000`. Full 130-point series in `sweep_full.csv`.)

5 Repeatability is a separate axis from latency

A median is one number, but a deployment lives on the whole distribution behind it. Two experiments here separate how fast a workload is from how repeatable it is.

5.1 The research engine: deterministic-tight by construction

The research engine is single-threaded, so nothing schedules against it. We measured per-callback latency across **2,000 independent backtests** of 884,130 bars each:

Per-call latency	µs
Min	0.57
P50	0.58
P99.9	0.60
Max	0.60
StdDev	**0.00**

The tail is almost flat: across 2,000 runs, P99.9 (0.60 µs) sits just 3% above the median (0.58 µs). Per-run throughput is equally tight, with a mean of **1,721,199 bars per second**, a CV of **0.59%**, and a full min-to-max spread of only **1.04 times**. This is the research-side consistency baseline, and it is a deliberate contrast to the server below.

5.2 Actual determinism, not merely low variance

Low variance is not the same as reproducibility. To test reproducibility, we ran the full path from CSV to backtest to result **20 times**. We hashed each output with SHA-256. The configuration uses a non-zero `price_volatility`. So the run exercises the stochastic slippage and spread path. Determinism here is a property of a fixed RNG seed. It is not a property of an absence of randomness.

Result: 20 runs produced 1 distinct hash. The output was byte-identical every time, including the stochastic path. A research workload can include stochastic market effects and stay exactly reproducible when its randomness has an explicit seed.

5.3 The execution core: variance is governed by placement

The core is concurrent, so its repeatability has to be measured on its own terms. We fixed N=32 (one strategy per core on half the die), 160,000 orders per run, and repeated the run **36 times** unpinned:

Aggregate throughput (orders/s)	value
Mean	233,327
StdDev	24,812
CV	**10.6%**
Spread (min to max)	**1.33 times**

Every run passed correctness, but the throughput is **bimodal** rather than normally scattered. 22 of the 36 runs cluster in a fast band around 252,000 per second, 14 in a slow band around 204,000, and the gap between the two is empty. That is the fingerprint of the operating-system scheduler: a run either places its threads well or it does not, with nothing in between.

The distinction that matters is between the batch and the single run. The **mean** (233,000 per second) is stable, and rerunning the batch lands it near 233,000 again. Any **single run**, though, is a coin-flip between the two bands. So "the core does about 233,000 per second at N=32" is true on average and misleading per run, and both facts have to be stated together.

Client pinning removes the slow band. Pin each strategy to its own physical core and the slow band disappears. Three raw runs at N=62 make it visible:

Config	run 1	run 2	run 3	spread
Unpinned	200,076	198,883	255,329	**1.28**× (two runs in the slow band)
Pinned	250,238	253,195	248,176	**1.02**× (all in the fast band)

The unpinned median is 200,076 per second and the pinned median 250,238, a gain of about 25%, while the worst-case run moved from 198,883 to 248,176 and the spread collapsed from 1.28 times to 1.02 times. `perf stat` confirms the mechanism: pinning roughly halves thread migrations (118 down to 63) with instructions per cycle essentially flat (1.53 to 1.51), because keeping each strategy's working set on one core's caches is what removes the bad placements.

The principle generalizes well beyond this engine. **Pin for repeatability, not for speed.** A median throughput on its own hides the spread, and the spread is exactly what a repeatability-sensitive deployment cares about most. Performance engineering for trading infrastructure should report distributions, not peaks.

(The thread-proxy and stub-broker caveats from Section 2.2 apply. The transferable claim is the variance collapse. It holds across all measured loads. The median gain is not consistent. It was about 25% at N=62, about 5% at N=32, and near zero at N=48. It tracks whether the unpinned batch happened to land in the slow band. So the reliable reason to pin is repeatability, not a fixed speed gain. The exact percentages are not the claim.)

5.4 Two engines, two sources of variance

The research engine's CV is 0.59%; the unpinned core's is 10.6%. Those two numbers are large and small for different reasons, and that difference is the most transferable idea in the report about determinism:

> Determinism and repeatability are distinct properties. The research engine gives exact output reproducibility under a fixed seed (Section 5.2), and its single-threaded execution separately gives very low timing variance (Section 5.1). The concurrent execution core gets neither for free: scheduler placement drives its run-to-run performance, and only explicit CPU affinity makes it reliably repeatable (Section 5.3). Good infrastructure makes each source of variance measurable and separately controllable, instead of treating "determinism" as a single binary property of the whole system.

6 Hardware-aware buffer sizing

The execution core carries a shared-memory event ring. Intuition says a larger ring is both safer and faster, but past a point the measurement says the opposite, and the reason is the cache hierarchy.

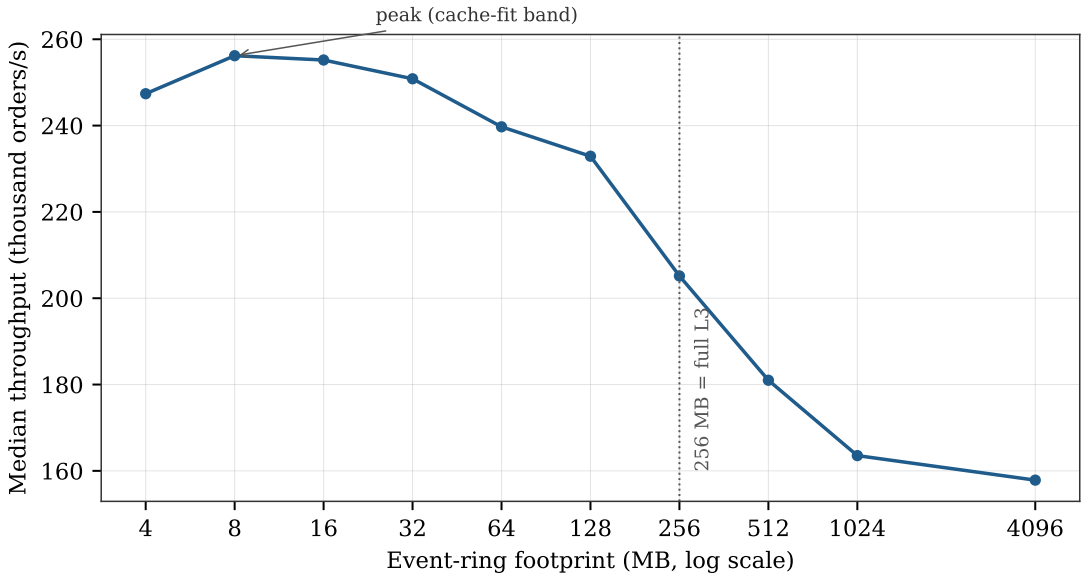


Figure 2. Median throughput against event-ring footprint (log scale) at fixed N=40. Throughput is an inverted-U. It peaks in a broad band around 4 to 32 MB. It falls on both sides. The 256 MB point sits at the full L3 size of this machine, the last point before the DRAM cliff.

Throughput traces an inverted-U in ring size, peaking where the ring still fits comfortably in cache. At fixed N=40, median of 3:

Ring footprint	Median orders/s	vs. 256 MB
4 MB	247,383	+21%
8 MB	256,190	+25%
16 MB (product default)	255,195	**+24% (peak band)**
32 MB	250,843	+22%
64 MB	239,716	+17%
256 MB	205,161	baseline
1 GB	163,536	−20%
4 GB	157,852	−23%

The optimum is a broad flat band around 4 to 32 MB, with throughput falling off on both sides. `perf stat` isolates the mechanism at the same instruction count each run:

Counter	16 MB ring	256 MB ring	4 GB ring
IPC	**1.53**	1.40	**1.19**
Cache-misses	57.7 M	98.1 M	190.7 M
dTLB-load-misses	3.9 M (18%)	12.1 M (44%)	21.4 M (43%)

Going from 16 MB to 4 GB at the same instruction count, cache misses rise about 3.3 times and dTLB misses about 5.5 times, and IPC falls from 1.53 to 1.19 as the pipeline begins stalling on memory. The dTLB miss rate alone jumps from 18% to 44% once the ring grows past what stays TLB-resident. This machine's L3 is 256 MB, so the 16 MB default sits well inside it while the 256 MB point sits at exactly full L3, the last step before the DRAM cliff. That is "cache-fit, not absolute size", read straight off the hardware counters.

The tuning principle. Treat the event buffer as a catch-up-depth knob, not a throughput knob. Size it to the smallest ring that still covers the worst-case reader lag, whether that lag comes from a slow gate, a broker reconnect, or an audit consumer catching up, and then confirm the ring still fits in cache. Sizing it up "for performance" is counterproductive, since all it buys is extra cache and TLB pressure. The optimum in absolute bytes moves with the cache hierarchy of the target processor, so verify it on the hardware you intend to run.

7 Capacity headroom

The following is the headroom of the research engine under adversarial data volumes, on the same machine:

Case	Work	Wall	Rate
CSV parse	about 4.7M rows to 884,130 bars	0.398 s	**2,223,465 bars/s**
Monte Carlo streaming	10,000 trades × 100,000 sims	0.094 s	**10.6 B ops/s**
Extreme capacity	10,000 tickers × 20 yr = **50M bars**	16.6 s	completed, +937 MB resident
Full US-equity depth	10k × 20 yr	29.1 s	completed

The two extreme cases matter most. Both the 50-million-bar universe and the full-depth equity-universe run completed without failing or breaching any threshold, and peak resident cost stayed under 1 GB. The engine's stress suite returned its own verdict, *"All metrics above threshold. No bottlenecks detected."* Read this as a single-box measurement of headroom, not as a published capacity SLA.

8 Implications for deployment

This section is interpretation rather than measurement. Each item is an implication a deploying team or an infrastructure integrator might reasonably draw from the results in Sections 3 to 7, all of which were taken on a single box with the stub-broker and thread-proxy caveats of Section 2.2. The numbers below are starting points to verify on target hardware, not commitments.

- **Concurrent workload density as a first-order estimate.** Because the clean-latency region reaches about one concurrent strategy workload per physical core (Section 4), a team sizing a box can use its core count as a first-order estimate of how many workloads it will host before the latency tail starts to widen. The qualifier matters, though: Section 4 measures 5,000 orders per strategy through an accept-all gate and a stub broker, so it establishes *computational* workload density only. It says nothing about whether a full production strategy, carrying its own market-data processing, risk logic, and venue connectivity, can be hosted one per core. That number depends on all of those things, and this study excludes them.
- **Shared-core deployment becomes a measurable design option.** The throughput plateau and latency behavior together show that multiple concurrent strategy workloads can share one execution-core instance on the tested hardware through roughly the core count. Whether to consolidate onto a shared instance or to isolate one per strategy still depends on the surrounding strategy, market-data, risk, and connectivity workloads. The measurement puts consolidation on the table as a design option to evaluate; it does not prescribe an architecture.
- **Repeatability is a placement decision, not a speed purchase.** A deployment that needs run-to-run reproducible numbers, such as reproducible research or a run-to-run SLA, should pin strategies to cores; one that only needs a healthy average does not have to. We do not sell pinning as "faster" (Section 5.3).
- **Buffer sizing is a workload decision, not a bigger-is-better one.** Size the event ring to reader-lag tolerance and cache fit on the target processor (Section 6), rather than copying a slot count from elsewhere.
- **The composable-core model.** Both components are C ABIs rather than turnkey applications: the research engine calls a user strategy callback, and the execution core is wrapped by a user-written gate and venue connector. The measurements therefore describe a component you can evaluate on its own and drop into a larger trading environment, with the surrounding infrastructure, from market-data distribution and strategy orchestration to risk policy, venue connectivity, persistence, monitoring, and tenancy, remaining the responsibility of the deploying organization or integrator. An integrator does not have to adopt the worldview of the core; it can place the core inside its own. What this report characterizes is the behavior of that core, not a multi-tenant hosting substrate, and any such deployment has to have its own envelope measured in situ.

Scope of the findings

This study does **not** characterize venue latency, network latency, market-data processing, production risk models, real broker behavior, multi-tenant interference, or the performance of a complete trading platform. It establishes no capacity SLA, and it implies nothing about whether the measured figures transfer unchanged to other processors or deployment configurations. The concurrency and variance runs use a stub broker and an accept-all gate, the pinning study uses a thread proxy for separate processes, and every figure comes from one machine on one date.

What it *does* characterize is the measured behavior of the research and execution cores under the workloads and configurations described here, together with the infrastructure relationships you can re-test elsewhere: core-count scaling, placement-driven variance, and cache-driven buffer behavior. Read the single-machine figures as starting points to verify on target hardware, not as commitments.

9 Conclusion

Above all the individual numbers, the measurements support one claim about method:

> The useful unit of performance for a trading engine is a measurable operating envelope, not a peak throughput or a single latency figure. That envelope is where concurrency scales, where scheduler contention begins, how the tail moves relative to the median, how repeatability is controlled, and how internal buffers interact with the cache. An engine is deployable to the extent that you understand and control it.

On a single-socket 64-core AMD EPYC 9575F, the stack studied here gave a consistent picture across every part of that envelope. The integration boundary was thin, at 53 ns. End-to-end latency was low and its tail was tight, with a 11.3 µs median and a P99 only 1.1 times that. The throughput plateau and the clean-latency region shared an elbow that tracked physical-core count on both systems we measured. A stochastic research workload reproduced exactly under a fixed seed. Run-to-run variance on the execution core was bimodal when unpinned and collapsed to roughly 1% spread once pinned. Event-buffer throughput peaked at cache-fit rather than at maximum size. And the research engine carried tens of millions of bars in under 1 GB resident.

These are separate claims about separate parts of the envelope, and we have kept them separate on purpose. None is a service-level guarantee, and all of them come from a single machine on a single date. The contribution is the framing as much as the figures: evaluate, size, and report a trading engine by its operating envelope, which is something you can measure directly on the hardware you intend to run.

Provenance and reproducibility

We captured all measurements on 2026-09-13 on the environment in Section 2.1 (AMD EPYC 9575F, single socket, single NUMA node, `performance` governor). The raw artifacts, per section:

Section	Artifact(s)
3.1 ABI boundary	timer-corrected boundary benchmark, 600k samples
3.2 End-to-end	ABI-only benchmark, 2 × 50,000 orders
4 Concurrency sweep	`sweep_full.csv` (130-point series), `sweep_stdout.txt`, `sweep_512_stdout.txt` (FD-corrected 512 point)
5.1 Research variance	`res_native_2000.txt` (2,000 runs)
5.2 Determinism	20× hashed quickstart runs
5.3 Server variance	`srv_variance.txt` (N=32 × 36 runs), `pinning_A_vs_C.txt` (N=8/32/48/62, ×3, plus `perf stat`)
6 Buffer sizing	`buffer_sweep_N40.csv`, `buffer_window_24-48_64k.txt` / `_1M.txt`, `perf_stat_N40.txt`
7 Capacity	`res_stress.txt` (stress/capacity suite)
Environment	`provenance.txt`

Two benchmark-harness fixes were needed to run on current builds. Neither is a defect in the measured software. One is a retired license-bypass form in one harness target. The other is the client file-descriptor limit at 512 strategies (Section 4).

Availability. The research engine and the execution core are available from Reamer Labs as evaluation kits. Each kit is a C ABI plus a shared or static library. Each kit includes an on-hardware benchmark binary. That binary reproduces these figures on the machine of the evaluator. Contact: hello@reamerlabs.com.

Declarations

Data availability. The raw measurement artifacts listed in the Provenance table are openly available at `https://reamerlabs.com/papers/data/epyc-9575f-2026-09-raw-artifacts.zip`. That archive holds the unedited console and CSV outputs behind every figure, with a README that maps each file to the section it supports.

Reproducibility. The evaluation kits include an on-hardware benchmark binary. That binary reproduces the boundary, end-to-end, concurrency, and buffer results on the hardware of the evaluator, subject to the caveats in Section 2.2.

Competing interests. The author is the founder of Reamer Labs and has a financial interest in the software measured. The measurement method, the caveats, and the raw artifacts are stated so that a reader can verify the findings independently.

Funding. This work received no external funding.

Author contributions. C. Palghadmal designed the study, built the benchmark harness, ran the measurements, and wrote the paper.

References

[1] ASD-STE100. *Simplified Technical English*, Specification ASD-STE100. AeroSpace and Defence Industries Association of Europe, Brussels.

[2] Advanced Micro Devices. *AMD EPYC 9575F Processor* (Zen 5, 64 cores / 128 threads, 5th Generation EPYC). Product specification.

[3] Gregg, B. *Systems Performance: Enterprise and the Cloud*, 2nd ed. Pearson, 2020.

[4] Weaver, V. M. Linux `perf_event` interface and the `perf` tool. In *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software.*

[5] Drepper, U. What Every Programmer Should Know About Memory. Red Hat, Inc., 2007.

[6] Dean, J. and Barroso, L. A. The Tail at Scale. *Communications of the ACM*, 56(2), 2013, pp. 74–80.

[7] Lameter, C. NUMA (Non-Uniform Memory Access): An Overview. *ACM Queue*, 11(7), 2013.

Preprint. The figures come from a single test machine on a single date. The author reports them with their caveats. They characterize an operating envelope, not a guaranteed service level. Reproduce on target hardware before you rely on any specific number. *© 2026 Chaitanya Palghadmal / Reamer Labs. All rights reserved.*