T2T-Microstack: Tick-to-Trade in Microseconds

A Minimal, Deterministic, and Observable HFT Replay Stack

Quant Systems Report

August 14, 2025

Abstract

This report documents the design, implementation, and performance of *T2T-Microstack*, a miniature high-frequency trading (HFT) replay system engineered for low latency, determinism, and correctness. The pipeline mirrors a production tick-to-trade path: *parse* $\rightarrow$ *limit-order-book update* $\rightarrow$ *signal/quote* $\rightarrow$ *risk gates* $\rightarrow$ *encode*. We implement a price–time LOB in a structure-of-arrays layout, a queue-reactive market-making signal with an optional Avellaneda–Stoikov (AvS) quoting mode, risk throttles, a no-allocation guard on the hot path, and per-stage timing with histograms.

On synthetic ITCH-like feeds the system meets replay SLOs on modern desktop CPUs: end-to-end p50 $\leq 20\,\mu s$, p99 $\leq 80\,\mu s$, throughput ≥ 1 Mmsg/s, and byte-wise determinism across repeated runs. We include latency plots, derivations for OU calibration and AvS quoting, and a reproducibility protocol.

1 Introduction

Low-latency trading systems are less about novelty than discipline: memory locality, predictable scheduling, observability, and hard guardrails. **T2T-Microstack** adopts this mindset in a small codebase (C++20, `-O3 -march=native`, exceptions disabled) with:

- A fixed-pool, price–time limit order book (LOB) with idempotent cancels.
- A queue-reactive quoting signal and an optional stochastic layer (OU $\rightarrow$ AvS).
- Risk gates (inventory cap, per-ms throttle, notional scaffold).
- Scoped timers, per-stage histograms, and a no-malloc guard activated after warm-up.
- Determinism: byte-identical outputs under identical replays.

The goal is not a flashy strategy but a faithful skeleton for HFT engineering tasks.

2 Non-Functional Objectives (SLOs)

Replay mode, single core, thread affinity pinned; warm-ups discarded.

- **Latency (end-to-end):** p50 $\leq 20\,\mu s$, p99 $\leq 80\,\mu s$.
- **Throughput:** ≥ 1 Mmsg/s on synthetic ITCH stream.
- **No allocations on hot path:** a guard aborts on any `new/malloc` post warm-up.
- **Determinism:** three identical replays $\Rightarrow$ byte-identical outputs.
- **Observability:** per-stage timing to CSV; histogram PNGs; quantile summaries.

3 Data Model and Formats

Prices and sides. Prices are integer ticks, $p \in \mathbb{Z}$, carried in `int32_t`. Sides are booleans: buy=1, sell=0.

Input (ITCH-like CSV). Rows are events:

$$(\texttt{ts\_ns}, \texttt{type}, \texttt{order\_id}, \texttt{side}, \texttt{px}, \texttt{qty}),$$

with `type` $\in \{A, C, E\}$ (add, cancel, execute). Executes reference existing `order_id`.

Output (normalized). We log quotes/executions:

$$(\texttt{ts\_ns}, \texttt{event}, \texttt{order\_id}, \texttt{side}, \texttt{px}, \texttt{qty}, \texttt{inv\_after}, \texttt{notional\_after}).$$

4 System Architecture

The pipeline:

$$\text{parse} \rightarrow \text{LOB update} \rightarrow \text{signal} \rightarrow \text{risk} \rightarrow \text{encode}.$$

Parsing. We replay a CSV into a pre-parsed vector; the stage is still timed for accounting.

Limit Order Book. A price–time priority book with:

- Structure-of-arrays: arrays of price levels; per-level FIFO linked lists of orders.
- Fixed node pool to avoid heap on the hot path; open-addressed helper maps.
- Idempotent `cancel(id)`; invariants verified in unit tests.

Signals. Two modes:

1. **Heuristic (queue-reactive).** Spreads widen with cancel/execute intensity and inventory; skew biases to flatten exposure.
2. **Avellaneda–Stoikov (AvS).** Closed-form reservation price and half-spread fed by OU volatility.

Risk. Inventory cap, per-ms throttle, a notional cap scaffold, and a kill-switch. The gate is synchronous and predictable.

Observability and Discipline. `steady_clock` nanosecond timers around stages; lock-free histograms; and a no-malloc guard enabled after warm-up (any allocation aborts with a precise message).

5 Algorithms and Derivations

5.1 Ornstein–Uhlenbeck (OU) Calibration

Let X_t be (log-)mid. The OU dynamics are

$$\mathrm{d}X_t = \kappa(\theta - X_t)\ \mathrm{d}t + \sigma\ \mathrm{d}W_t, \tag{1}$$

with mean reversion $\kappa > 0$, long-run mean θ, and volatility $\sigma > 0$. Discretized at interval Δ:

$$X_{t+\Delta} = aX_t + b + \varepsilon_t, \qquad a = e^{-\kappa\Delta}, \qquad b = \theta(1-a), \tag{2}$$

$$\varepsilon_t \sim \mathcal{N}\left(0, \sigma_\Delta^2\right), \qquad \sigma_\Delta^2 = \sigma^2 \frac{1 - e^{-2\kappa\Delta}}{2\kappa}. \tag{3}$$

We estimate (a, b) by OLS on consecutive pairs $(X_t, X_{t+\Delta})$ and invert:

$$\widehat{\kappa} = -\frac{\ln \hat{a}}{\Delta}, \quad \widehat{\theta} = \frac{\hat{b}}{1-\hat{a}}, \quad \widehat{\sigma} = \sqrt{\hat{\sigma}_\Delta^2 \cdot \frac{2\widehat{\kappa}}{1-e^{-2\widehat{\kappa}\Delta}}}. \tag{4}$$

This is the Gaussian MLE under regularity; residual diagnostics (QQ) are straightforward extensions.

5.2 Avellaneda–Stoikov (AvS) Quoting

Assuming exponential utility and Poisson arrivals $\lambda_\pm(\delta) = Ae^{-k\delta}$, the HJB admits the closed-form:

$$r_t = s_t - q_t\,\gamma\,\sigma^2(T-t), \quad \text{(reservation price)} \tag{5}$$

$$\delta_t^* = \frac{1}{k}\ln\Big(1+\frac{\gamma}{k}\Big) + \frac{1}{2}\gamma\,\sigma^2(T-t), \quad \text{(half-spread)} \tag{6}$$

where s_t is the mid, q_t the inventory, γ risk aversion, k arrival curvature, and $T-t$ the residual horizon. We quote $r_t \pm \delta_t^*$ and round to ticks. Intuitively: higher σ or γ widens spreads; inventory skews quotes to reduce exposure; shrinking horizon tightens them.

6 Implementation Notes

Language and flags. C++20 with `-O3 -march=native -Wall -Wextra -Werror -fno-exceptions -fno-rtti`. Hot-path code avoids exceptions and dynamic allocation.

LOB details. Fixed-capacity pools per side; $O(1)$ insertion/removal at price levels; best levels maintained; invariants unit-tested (non-negative sizes, monotone timestamps, sorted price levels).

SPSC ring. Header-only, padded against false sharing, power-of-two capacity. Not used in replay mode but ready to decouple live ingress from strategy.

Timing and histograms. Each stage stores an array of ns samples. Histograms are built post-run over bucket edges (in µs): {1, 2, 5, 10, 20, 50, 80, 100, 200, 500, 1000}.

No-malloc guard. We enable the guard after N warm-up messages; any `new`/`malloc` thereafter aborts with a backstop message. Output buffers and vectors are pre-sized before enabling the guard and freed after disabling it.

7 Experimental Methodology

Hardware/OS: fill with your specifics (CPU model, OS version, Linux governor).
Build: Release (`-O3 -march=native`).
Affinity: pinned to a fixed core.
Warm-up: first N messages discarded (`--warmup`).
Dataset: synthetic ITCH-like CSV generated by `tools/gen_synth_feed.py`.
Metrics: per-stage nanosecond samples; post-run histograms and quantiles.

8 Results

Figure 1 shows per-stage latency histograms from a representative run on the synthetic feed. Most of the mass is well below 1 µs at the stage level; end-to-end is sharply concentrated near 0 µs to 1 µs with light tails—cleanly within the SLO envelope for replay mode.

Figure 1: Per-stage latency histograms (synthetic feed).

Quantile Table

To extract exact quantiles (in µs) from `latency.csv`:

```
python - << 'PY'
import pandas as pd
df = pd.read_csv('latency.csv')
for s in ['parse','lob','sig','risk','e2e']:
    d = (df[df['stage']==s]['ns']/1000.0)
    if d.empty: continue
    print(f"{s:>4}:␣p50={d.quantile(0.50):.3f}␣␣p90={d.quantile(0.90):.3f}␣␣"
          f"p99={d.quantile(0.99):.3f}␣␣p99.9={d.quantile(0.999):.3f}")
PY
```

Populate Table 1 with your numbers:

Table 1: Latency targets and measured quantiles (fill with script output).

Stage	**Target p50/p99 (µs)**	**p50**	**p99**	**p99.9**
Parse	≤ 3	–	–	–
LOB	≤ 5	–	–	–
Sig	≤ 4	–	–	–
Risk	≤ 3	–	–	–
End-to-end	$\leq 20/80$	–	–	–

9 Determinism and Reproducibility

Determinism. We require that repeated replays produce byte-identical output logs. A unit test replays a micro-feed thrice and asserts $o_1 = o_2 = o_3$ byte-for-byte. The `tools/diff_runs.py` utility compares full runs.

Reproducibility Protocol. Pin the main thread to a fixed core, record CPU/OS/governor, discard a warm-up window, then enable the no-malloc guard. Build with the documented flags; avoid non-deterministic containers and time-dependent randomness in the hot path.

10 Risk Gates

Inventory and notional exposure require stable risk budgets even in replay. We implement:

- **Inventory cap:** blocks the side that would worsen an overage.
- **Per-ms throttle:** limits orders per millisecond; simple and effective.
- **Notional cap (scaffold):** connect to MTM/P&L as needed.
- **Kill-switch:** hard stop.

These checks are synchronous and minimize unpredictable branching.

11 Limitations and Next Steps

- Replay only; networking disabled. Next: AF_XDP/DPDK feed handler, or a shared-memory gateway.
- Strategy minimalism is intentional; plug in your production logic behind the same interfaces.
- OU/AvS can be extended with robust SEs, rolling windows, and microstructure-aware volatility.
- Formalize p99.9 SLO gates and wire auto-regression plots into CI artifacts.

12 Conclusion

T2T-Microstack demonstrates that careful systems work—fixed pools, predictable scheduling, and explicit observability—produces strong latency characteristics without heroics. The OU/AvS layer adds a principled stochastic backdrop for quoting, while determinism and the no-malloc guard provide engineering assurances that survive maintenance and iteration.

References

- Avellaneda, M., & Stoikov, S. (2008). High-frequency trading in a limit order book. *Quantitative Finance*, 8(3), 217–224.
- Uhlenbeck, G. E., & Ornstein, L. S. (1930). On the theory of the Brownian motion. *Physical Review*, 36(5), 823.

A Appendix: Pseudocode of the Replay Loop

```
preallocate_buffers_and_pools();
load_replay_csv(events);
pin_to_core(core_id);
warmup = W;

for i in [0..N):
  if i == warmup: enable_no_malloc_guard();

  // parse stage (already structured in memory)
  t_parse.start();   /* no-op in replay */  t_parse.stop();

  // LOB stage
  t_lob.start();
  switch events[i].type:
    case 'A': book.add(order); break;
    case 'C': book.cancel(id); mm.on_cancel(); break;
    case 'E': mm.on_exec(); pnl.on_exec(...); book.cancel(id); break;
  t_lob.stop();

  // signal stage
  t_sig.start();
    if (mode == 'avs' and enough_mids): q = avs_quote();
    else:                               q = heuristic_quote();
  t_sig.stop();

  // risk stage
  t_risk.start();
```

```
    allow = risk.allow(q, inv, caps, ts_ns);
  t_risk.stop();

  // e2e stage includes encode when allowed
  t_e2e.start();
    if (allow): encode_csv_line(q, pnl, ...);
  t_e2e.stop();

disable_no_malloc_guard();
write_latency_csv();
build_histograms_and_dump();
```