Syed Samroze Ali
System Design

I designed a

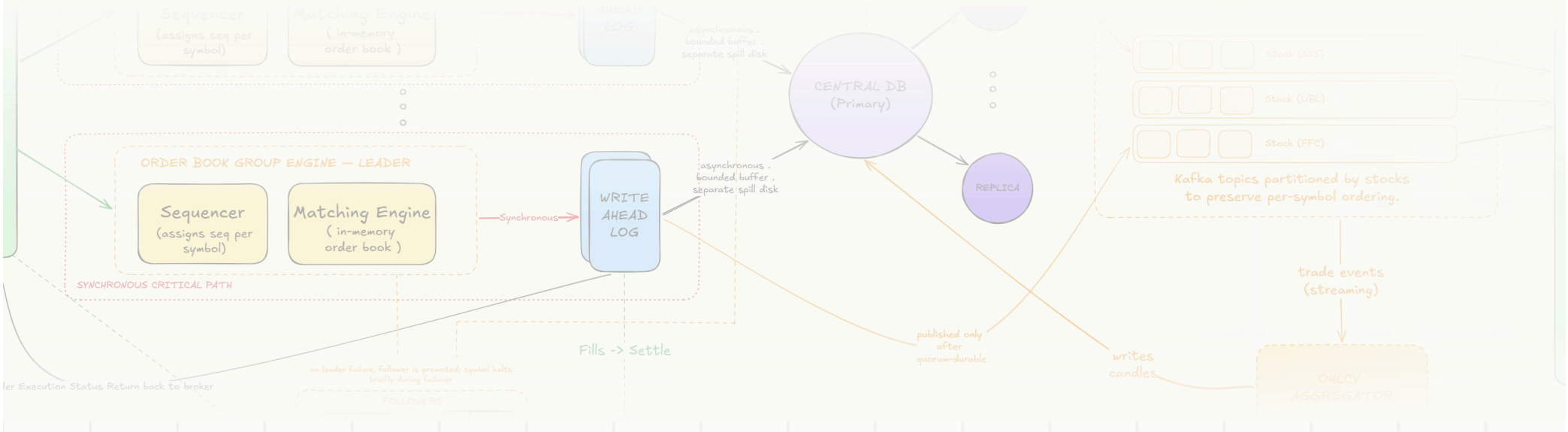
Stock Exchange

From order entry to matching engine to market data — anchored on real PSX numbers.

561 symbols · 400 brokers · 620,763 trades/day

Anchored on Pakistan Stock Exchange (PSX)

↓ EVERY BOX EXPLAINED



linkedin.com/in/syed-samroze-ali

Syed Samroze Ali

First, the **Scope Decision**

This is the exchange core — not a brokerage app.

Retail investors never touch this system. Brokers are the exchange's only clients — 400 member firms, each holding one persistent, authenticated trading session.

Broker
place / cancel / query orders, receive fills

Exchange Operator
halt the market, list & delist symbols

Regulator
pull the append-only audit trail

Data Subscriber
live book stream + historical candles

Cutting retail out drops the concurrency unit from ~220,000 users to **400 connections**. That one scoping call reshapes everything downstream.

linkedin.com/in/syed-samroze-ali

Syed Samroze Ali

Every decision traces to **These Numbers**

No architecture choice here was made in the abstract.

620,763
trades/day — a real PSX session

20 : 1
assumed order-to-trade ratio

12.4 M
orders/day that implies

5,750/s
peak order submissions (10× avg)

12,648/s
peak total write ops

~2.43 TB
5-year raw storage @ 20% YoY

21,600-sec session · 10× peak-to-average from open/close concentration · 50:1 retained as a stress case, not the baseline.
Source: Pakistan Stock Exchange public data & data portal, Feb 2026.

linkedin.com/in/syed-samroze-ali

Syed Samroze Ali

The four
Hard Constraints

These NFRs eliminated most “normal” options before I drew a single box.

Latency

P50 1 ms, P99 10 ms on order processing

→ **kills REST/JSON on the hot path**

Consistency

strict price-time priority, deterministic replay

→ **kills concurrent matching**

Durability

acknowledge only after the write is on disk

→ **WAL before response, always**

Availability

on failure, halting beats mismatching

→ **consistency wins on the critical path**

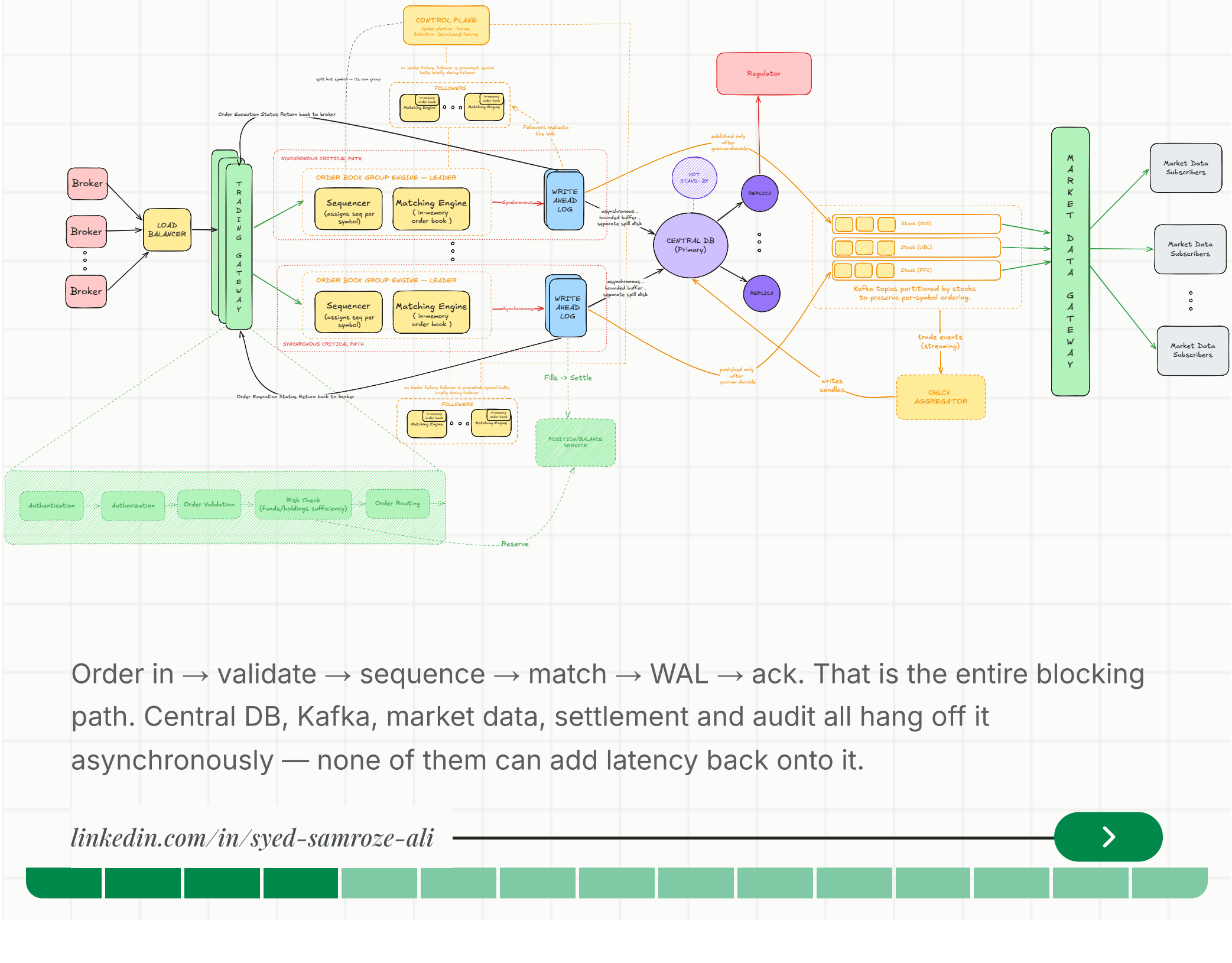
Syed Samroze Ali

The whole **System**

One small synchronous critical path. Everything else runs async behind it.

SYNCHRONOUS CRITICAL PATH

ASYNC



Order in → validate → sequence → match → WAL → ack. That is the entire blocking path. Central DB, Kafka, market data, settlement and audit all hang off it asynchronously — none of them can add latency back onto it.

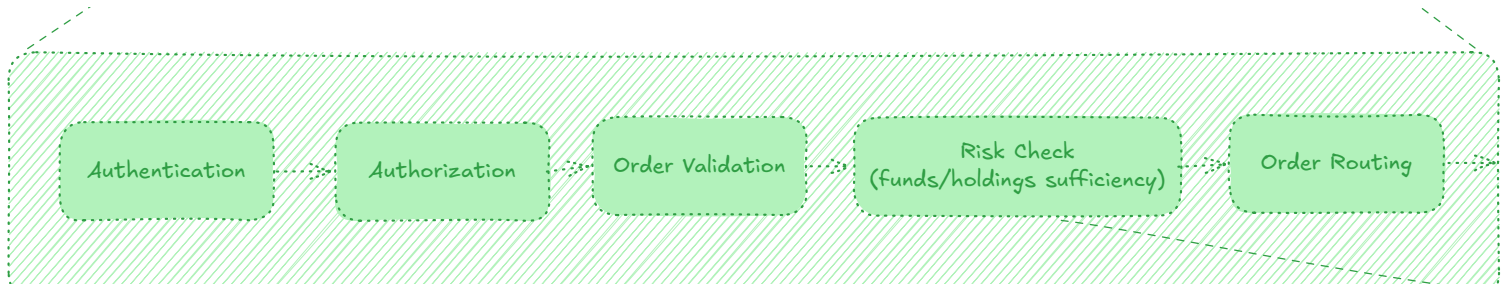
linkedin.com/in/syed-samroze-ali

01

Syed Samroze Ali

Load Balancer & **Trading Gateway**

Stateless by design — that is the entire reason it scales sideways.



1. **Authenticate**
   verify broker identity
2. **Authorize**
   trading permission for this instrument
3. **Validate**
   format, order type, quantity, price
4. **Risk check**
   funds/holdings sufficiency **+ reserve**
5. **Route**
   hand off to the correct book group

> All five are synchronous and reject-on-failure. None touch shared state across symbols — which is exactly why the gateway replicates freely behind the load balancer.

02

Syed Samroze Ali

The partitioning **Insight**

Symbol is the right logical boundary — and the wrong deployment unit.

A dedicated stack per symbol means 561 symbols × (server + sequencer + leader + 2 followers + WAL) ≈ **1,700 processes**, each serving about 4 orders/sec. So symbols are consolidated into **book groups** by consistent hashing — one single-threaded process owning the books for a *set* of symbols.

~~~~1,700 processes~~ → **20–50 book groups**

30–80× smaller footprint · zero loss of determinism

Determinism survives
it is still one thread processing one sequenced stream — just scoped to a group instead of a symbol, so price-time priority holds exactly as before

Rebalancing stays cheap
consistent hashing means moving one symbol between groups touches only that symbol, not the whole assignment scheme

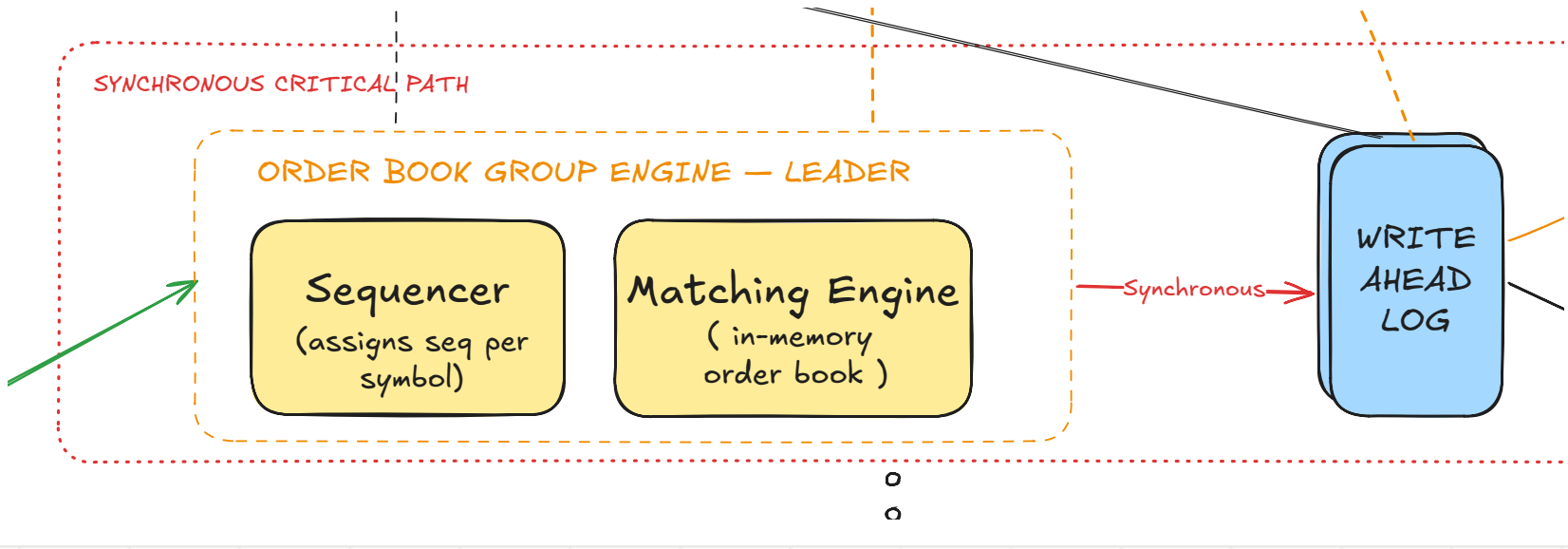
> A hot symbol is not a special case — it is a rebalance. Pin it to its own group of one, with dedicated hardware and core pinning.
~~~~

03

Syed Samroze Ali

Inside a **Book Group**

Sequencer and matching engine are one thread, not two services.



Sequencer step
assigns a monotonic sequence number per symbol — system-assigned, never client timestamps

Matching step
every book in the group lives fully in memory; matched in the same thread, same process

No hop between them
a separate sequencer service would add a network round-trip inside a 1 ms budget

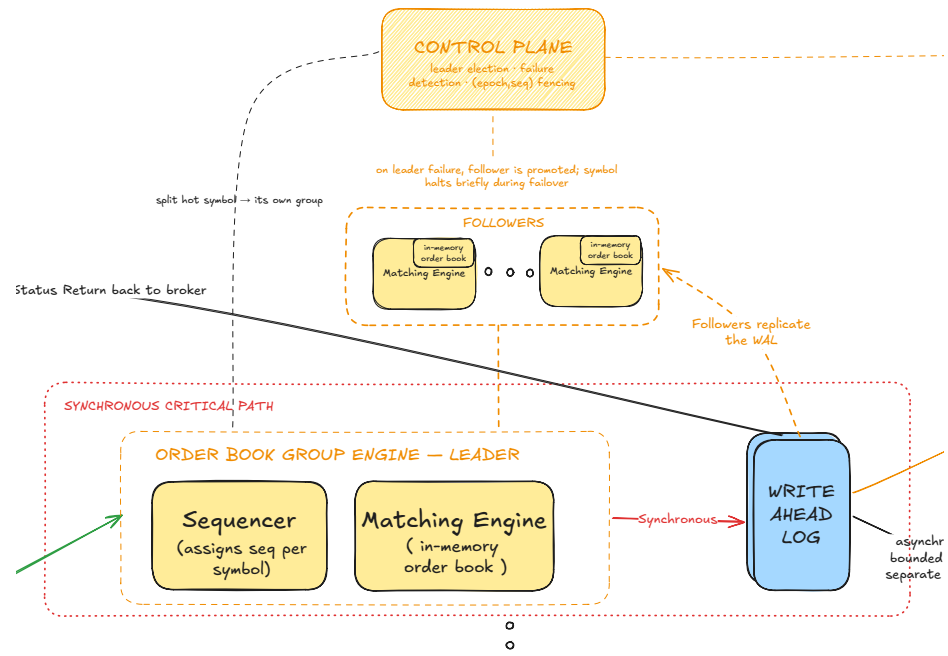
> The only measured hop left on the critical path is the WAL write.

04

Syed Samroze Ali

Failover without **Split-Brain**

Followers replicate the log — they never process orders.



The Control Plane owns leader election, failure detection and **(epoch, seq) fencing**. Every write carries an epoch and stale-epoch writes are rejected, so a demoted or partitioned old leader physically cannot keep writing.

> On leader failure that group's symbols **halt briefly** rather than paying consensus latency on every single order. Blast radius is one group, not the exchange — predictable 1 ms latency bought with a rare, bounded outage.

05

Syed Samroze Ali

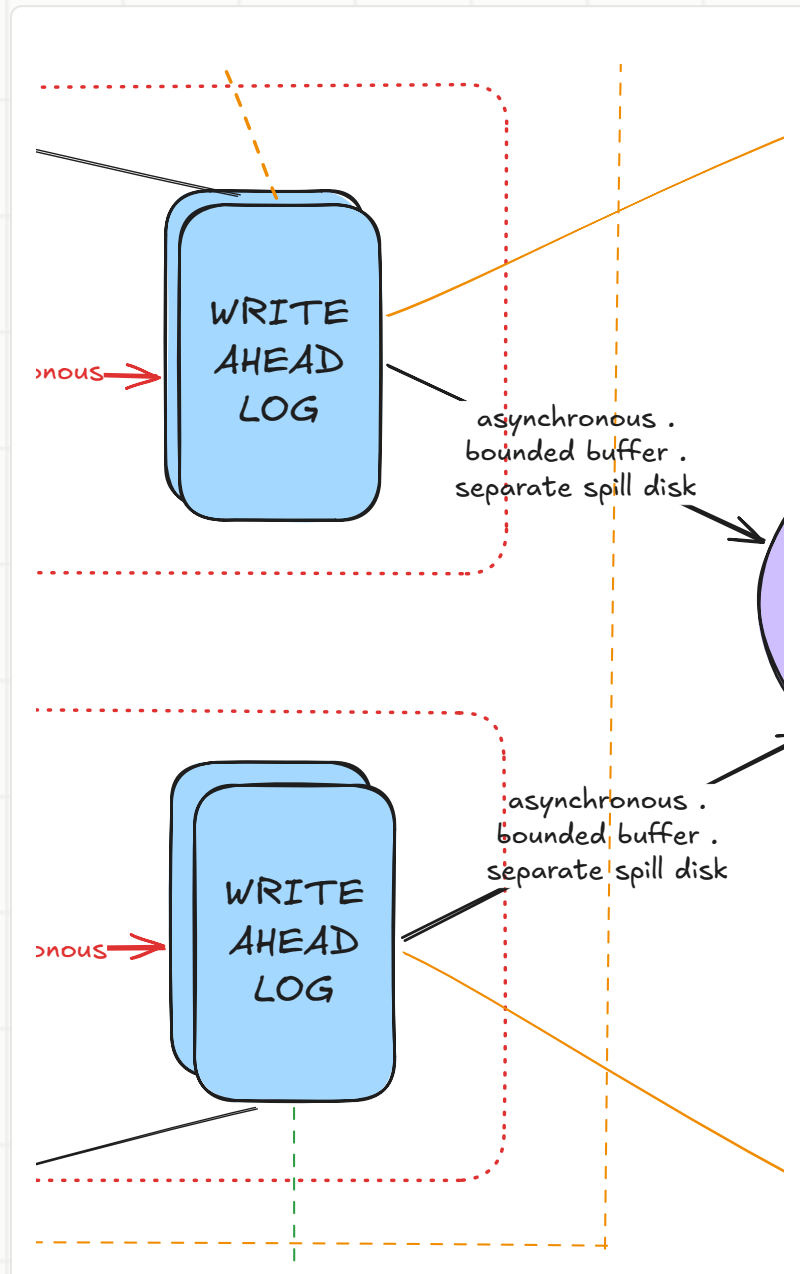
The durability **Boundary**

Nothing is real until it is in the write-ahead log.

Atomic
the order and its resulting trade are written together, or not at all

Ack-after-write
only then does the execution report go back to the broker

Local per group
no shared log, so no group ever waits on another's write volume



> Downstream is fed through a **bounded buffer with its own spill disk**. If the database slows down the buffer spills — matching never stalls.

06

Syed Samroze Ali

Why one node is **Enough**

The unglamorous answer is usually the right one.

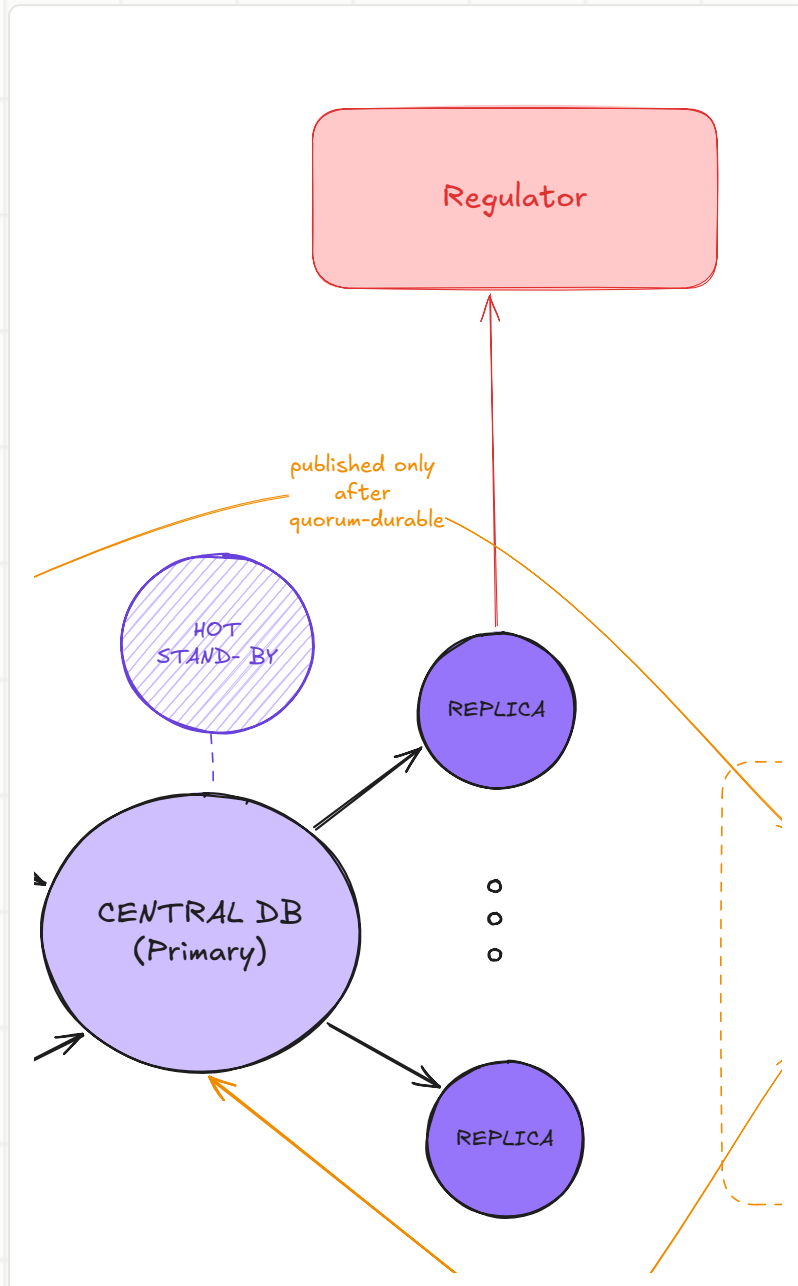
Five-year cumulative raw storage lands at **~2.43 TB** (~7.3 TB replicated). That fits comfortably on a single well-replicated primary. Sharding it would be complexity with no number behind it.

Read Replicas

serve traffic — regulator audit pulls, candle queries. Slight lag is fine.

Hot Standby

serves nothing — stays synced, ready to be promoted. Its only job is being ready.



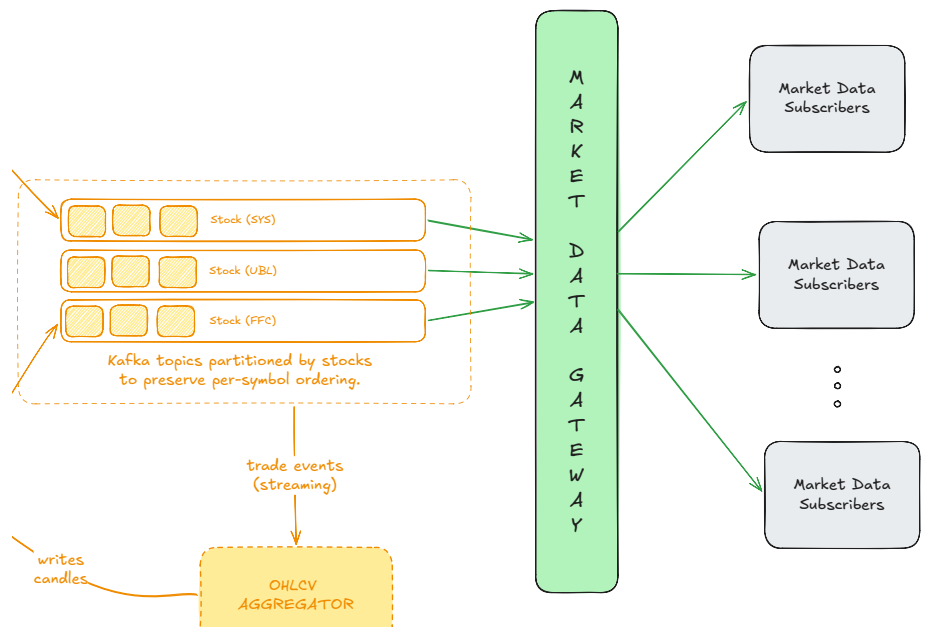
> External readers only ever see **quorum-durable** data — never a write that could still be rolled back.

07

Syed Samroze Ali

Real-time **Fan-Out**

Kafka is partitioned by symbol. Subscribers never need to know that.



Kafka does

per-symbol topics preserving the exact ordering the matching engine enforced · durable and replayable · only ~5,750 events/sec at peak

The Gateway does

one stable WebSocket/REST endpoint · per-connection auth · per-symbol subscription filtering · backpressure on slow subscribers

> The gateway keeps partition layout an implementation detail instead of a public contract — Kafka's topology can change without breaking a single subscriber.

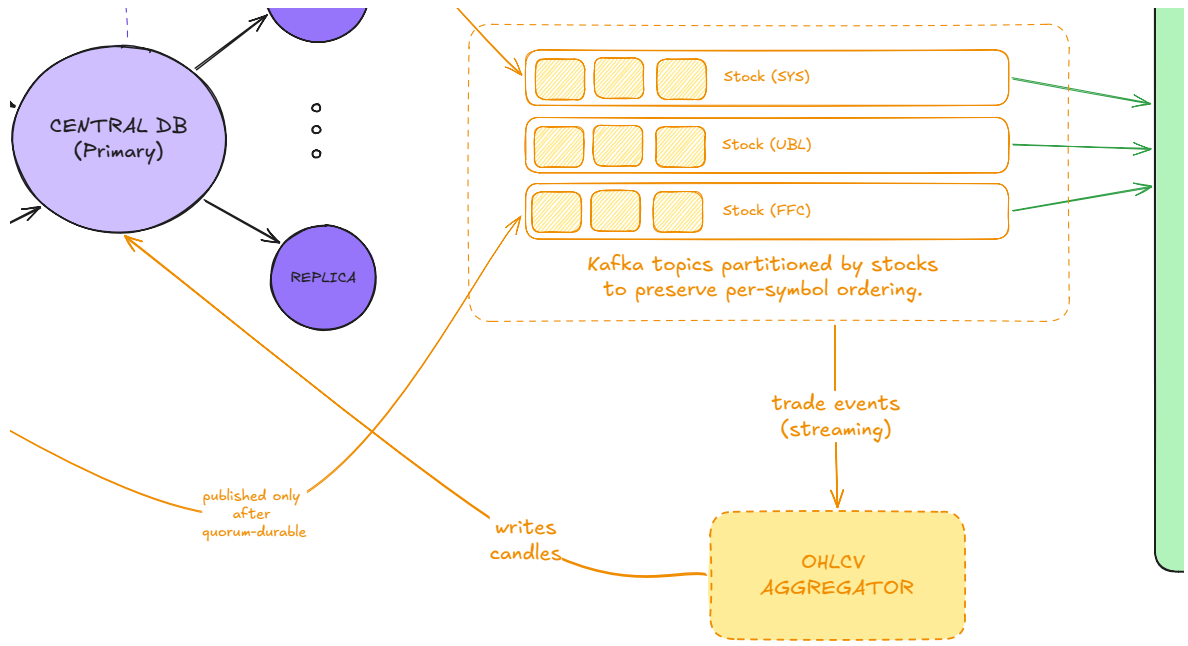
linkedin.com/in/syed-samroze-ali

08

Syed Samroze Ali

Charts without the **Query Cost**

Candles are computed continuously, never on demand.



The aggregator subscribes to the same Kafka trade stream and builds 1-minute candles as trades arrive, then writes them into Central DB on its own dedicated path.

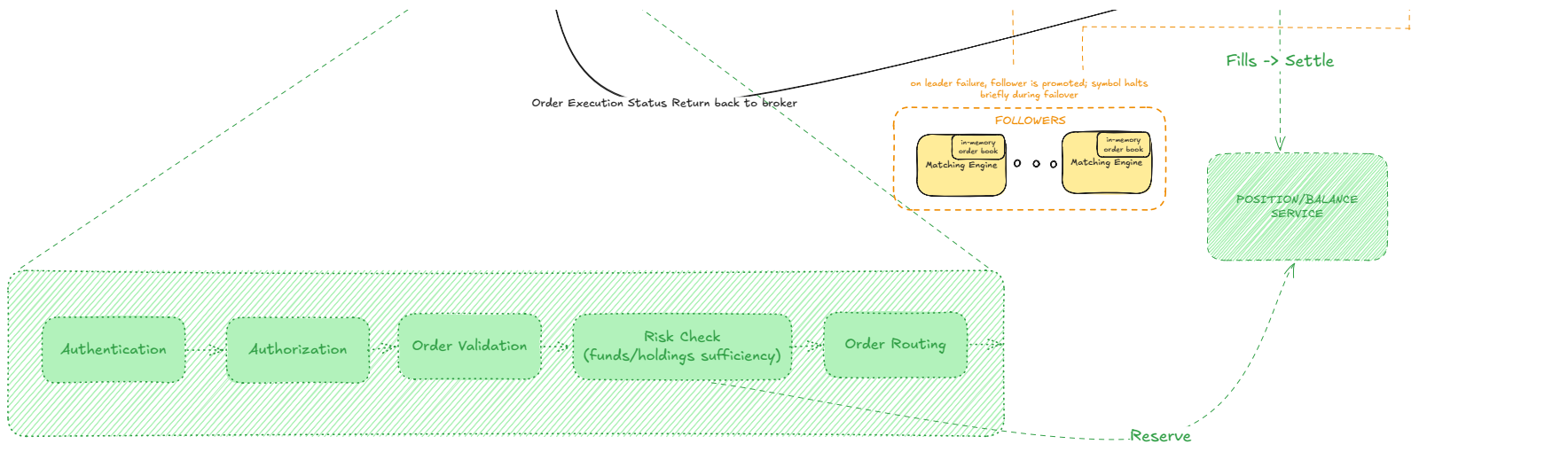
> A subscriber asking for a month of 1-minute candles gets a **fast row-range read** — not an on-the-fly aggregation over millions of raw trades. Candles cost ~15 MB/day against ~1,242 MB/day of orders.

09

Syed Samroze Ali

Reserve, then **Settle**

The same capital can never be committed twice.



Reserve — pre-trade
at Risk Check, required funds/holdings are reserved against the broker's account, so a second order cannot over-commit the same capital

Settle — post-trade
fills flow asynchronously off the critical path; positions finalize and the reservation is consumed or released

> Neither step adds latency to order entry. The reservation is a fast check against state already held; settlement happens entirely after the fact.

Syed Samroze Ali

The uncomfortable **Choices**

Good system design is mostly choosing what to give up.

GAVE UP	GOT
~~Concurrency in matching~~	**deterministic, replayable, provably correct ordering**
~~Availability during failover~~	**1 ms latency without consensus on every order**
~~A sharded database~~	**operational simplicity, justified by a real storage number**
~~REST/JSON on order entry~~	**a latency budget that actually holds under load**

> Every “simpler” choice here is defended by a number from the QPS or storage estimate — not by taste.

linkedin.com/in/syed-samroze-ali

Syed Samroze Ali

Architecture is

Tradeoffs, Priced

Every number behind this — QPS math, storage model, ERD, API design — was worked out before a single box got drawn.

Building something latency-critical? I'd genuinely like to hear which of these calls you would have made differently — drop it in the comments.

linkedin.com/in/syed-samroze-ali