

Consistent by Design

How Sequencer Architectures
Benefit Exchange Participants

SEPTEMBER 2026

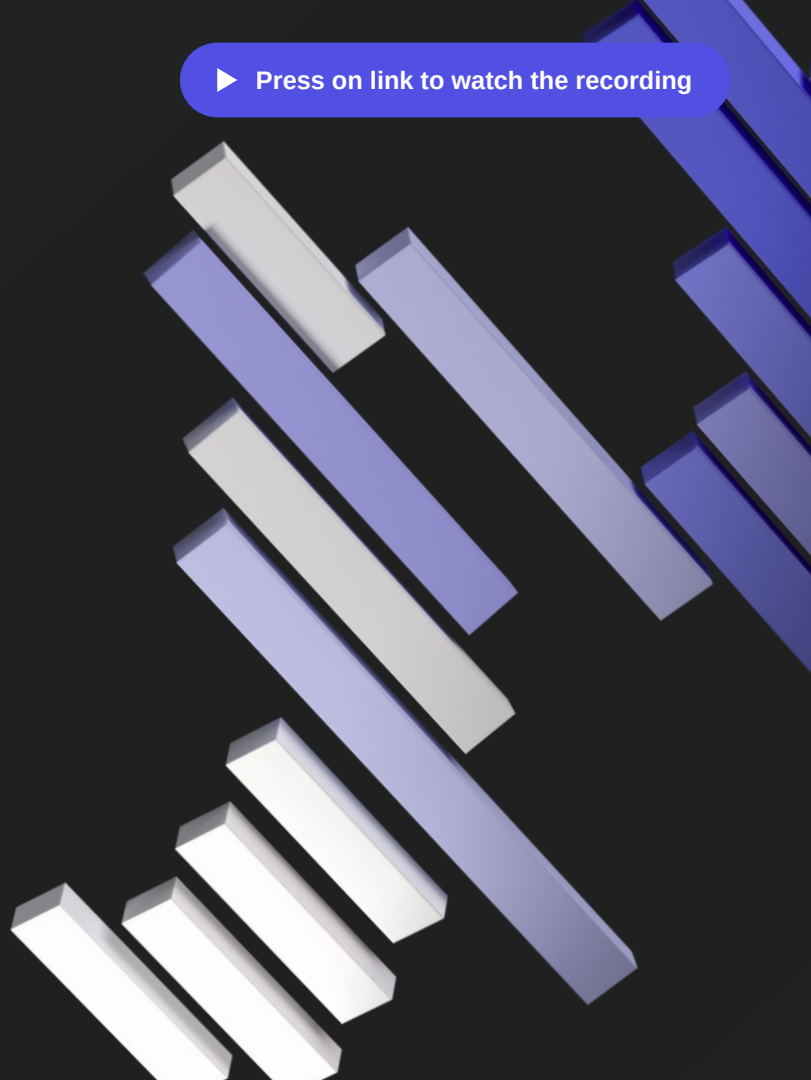


David Smith

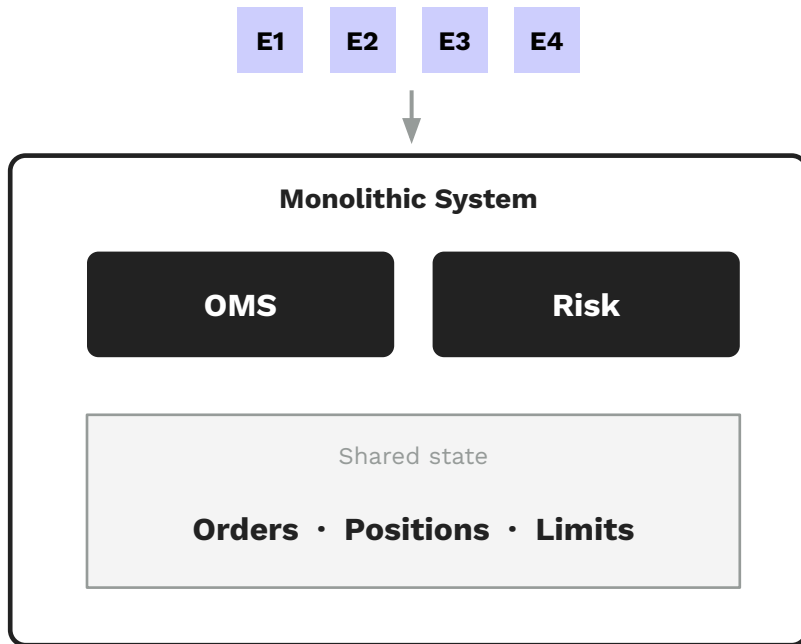
Global Head of Systems Engineering, Adaptive
Deutsche Börse Group Open Day

 **Adaptive**

▶ [Press on link to watch the recording](#)



Monoliths as consistent systems



- One process owns every decision
- Risk checks and reservations happen atomically
- Failure, deployment and scaling remain coupled

Duplicating the system duplicates authority



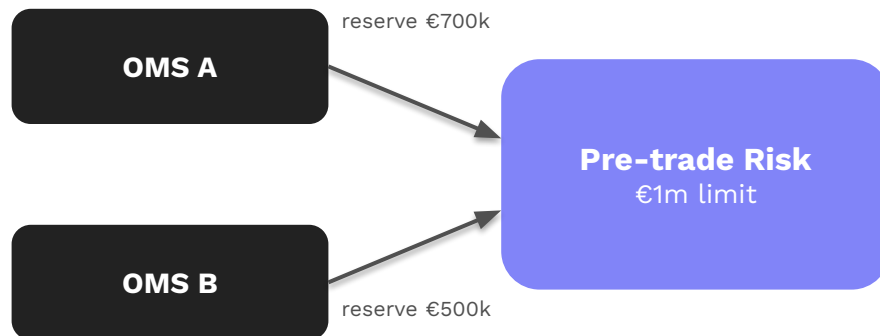
Both copies start from identical state

Both make a locally correct decision

The combined outcome is invalid

Risk serialises reservations

One authoritative limit, coordinated on every order.



Risk decides the order, then the answer

- 1 Reserve €700k**
accepted, €300k remains
- 2 Reserve €500k**
rejected, only €300k remains

The €1m limit after order 1



One authoritative source

Single process owns positions and limits

Every order asks Risk first

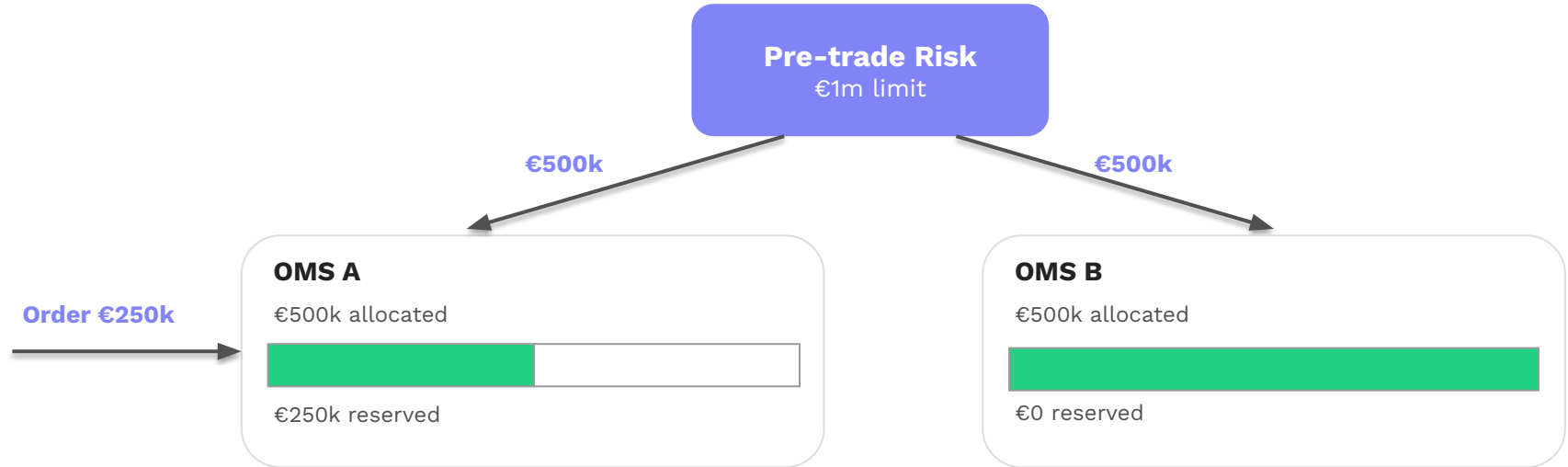
A round trip before anything is sent. Risk is the serialisation point

Risk is on the critical path

Its latency is your latency. Its availability is your availability

Risk serialises allocations

Partition the limit, allowing an OMS to make independent decisions.



Local decisions

No Risk round trip on the order path

Risk still serialises

For allocations and rebalances

Capital efficiency

Rebalancing effectively is difficult

Both designs implicitly sequence events

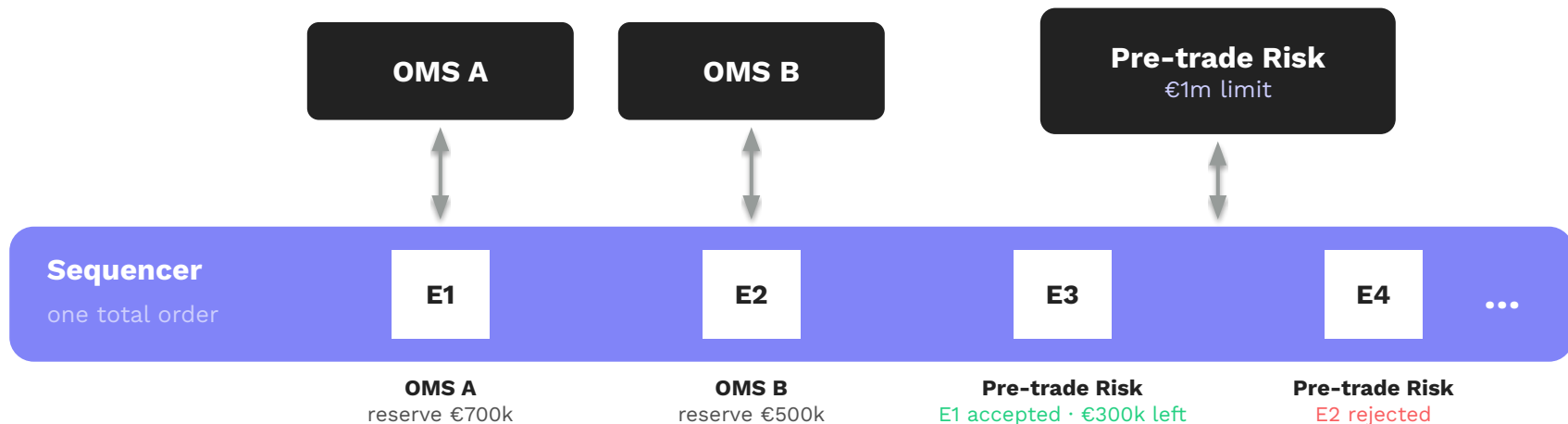
Per-order reservations

- Sequence every order
- Efficient use of capital

Pre-allocated limits

- Sequence each allocation
- Efficient order path
- Coordination during rebalancing

Explicit sequencing



One place decides the order

The sequencer assigns each event a position

Interaction is via the sequence

Components are both producers and consumers of the sequence

Same order, consistent outcome

Components make consistent decisions by observing events in the same order

Replay, observability, test and upgrade

Replay

Components can recover their state by replaying from the sequence.

Deterministic state machines can recover through execution of the exact same code as the live path.

Observability

The sequence is an immutable record of what every component did and in what order.

Audit and regulatory evidence can be observed from the sequence.

Issues can be reasoned about and reproduced in other environments.

Backtesting

Changes to the system can be tested against past or synthesised data.

Upgrades

Components are coupled only through the sequence, so each one is upgraded on its own cadence.

Bring the new version up beside the old, let it replay to the current position, then switch.

Versioned messages let old and new consumers read the same sequence during the change.

Consensus replicates the sequence

Explicit sequences can be replicated.



Durable

Events are stored in multiple locations, so they remain available in the event of process or host failure

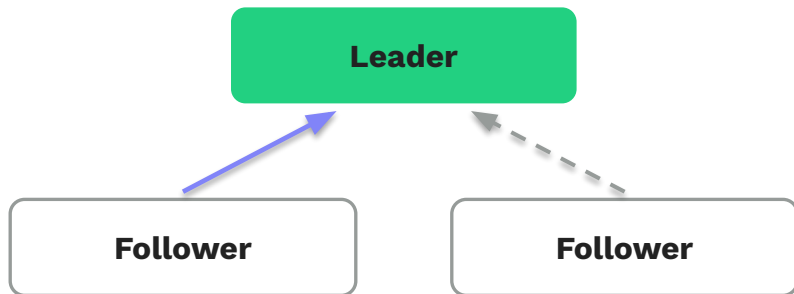
Highly available

In the event of leader failure, remaining replicas elect a new leader

Consistent

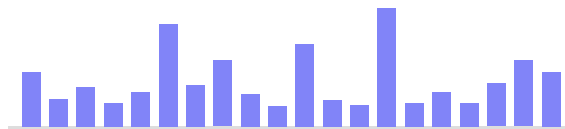
Components see one sequence and order. Events are only seen when a majority of replicas acknowledge them

Multiple active nodes

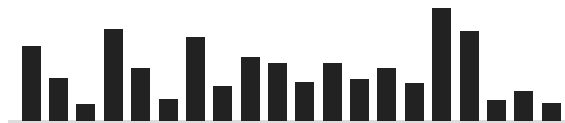


- Consensus-based systems require state to be replicated to greater than half of the nodes.
- If one node is slow to respond the leader still advances state.
- This can produce more consistent latencies than traditional replicated designs.

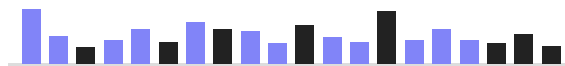
Follower 1 latency



Follower 2 latency

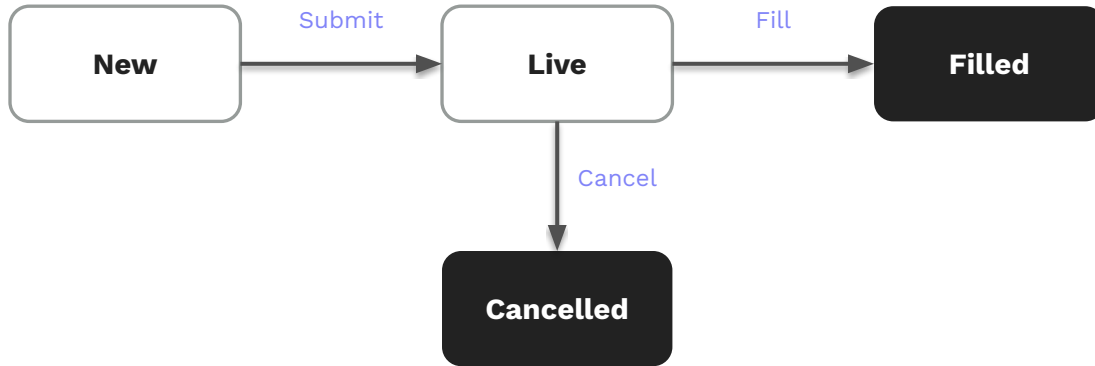


Acknowledgement latency



Deterministic state machines

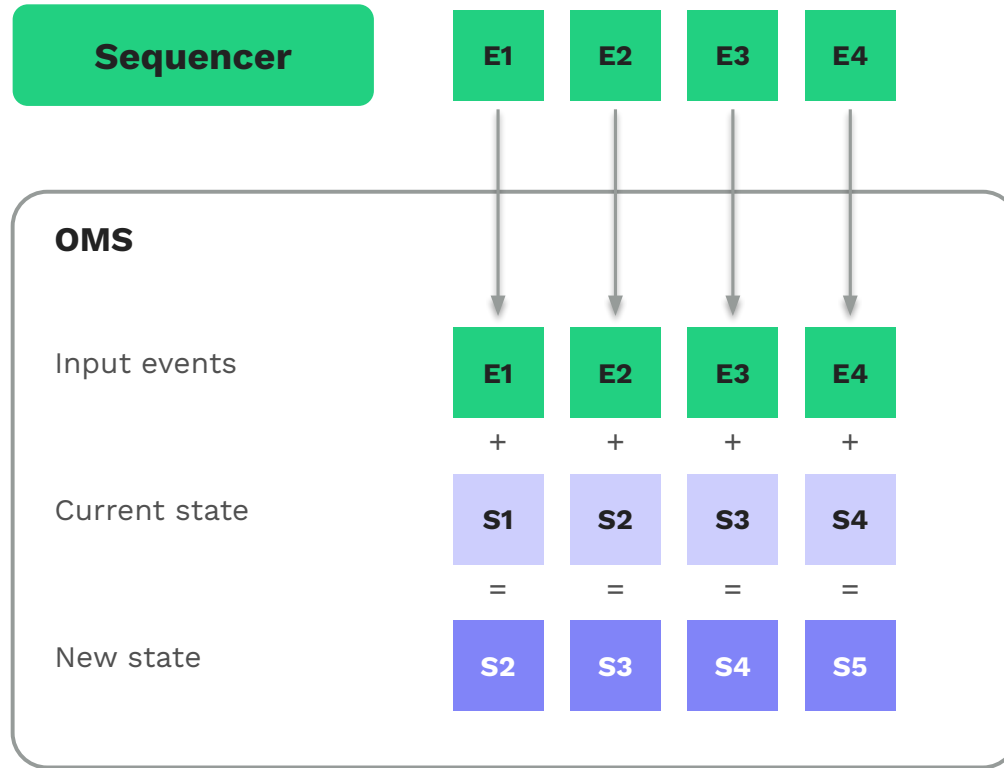
Current state + event = new state.



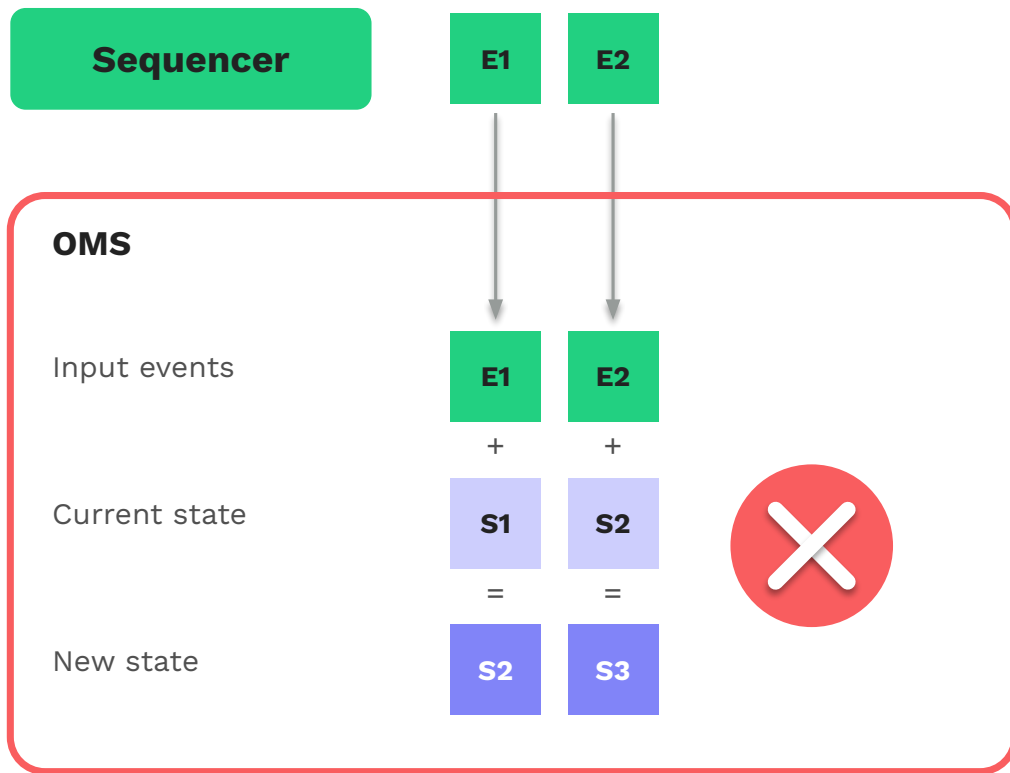
Key properties

- System begins from a defined starting state
- Inputs come only from the log
- State machine transitions to a new state, emitting zero or more events
- The new state and emitted events are always the same for a given starting state and input

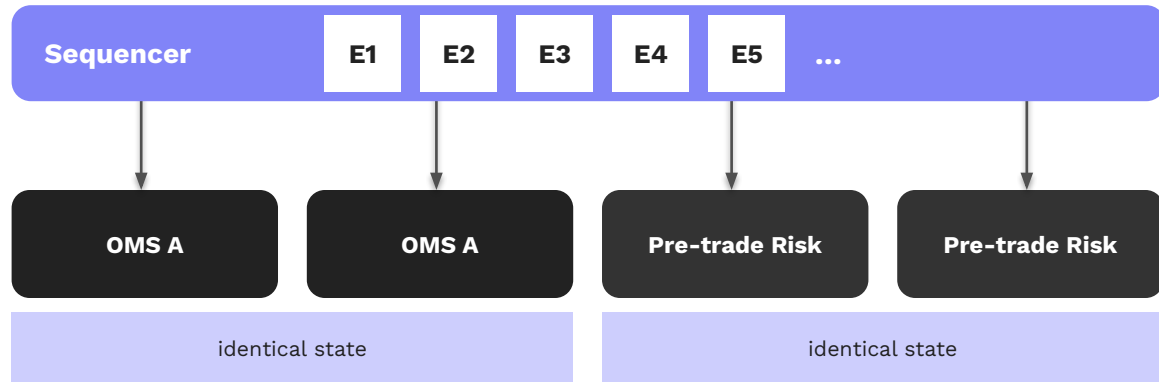
State machines + sequencer in action



State recovery



State machine replication makes the consumers available



Active / active

Both replicas publish. The sequencer keeps the first and discards duplicates

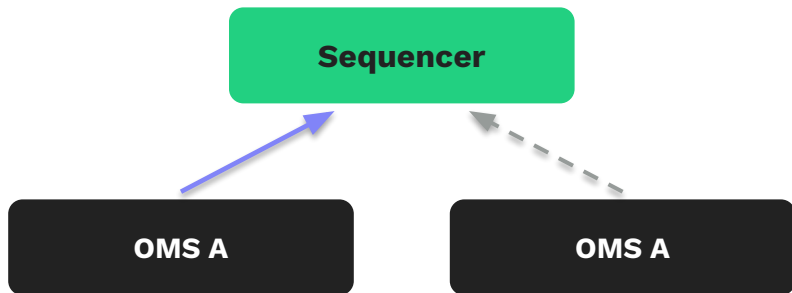
Active / warm

Sequencing components are an ideal place to hold distributed locks

Losing replicas are transparent

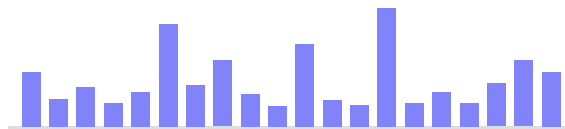
Catastrophic process failures become delays

Multiple active state machines

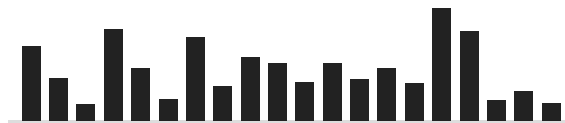


- The same concept applies to replicated state machines.
- A sequencer processes the response from the fastest application, discarding slower responses.

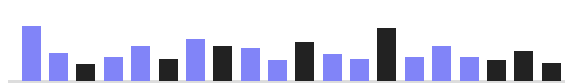
App 1 latency



App 2 latency

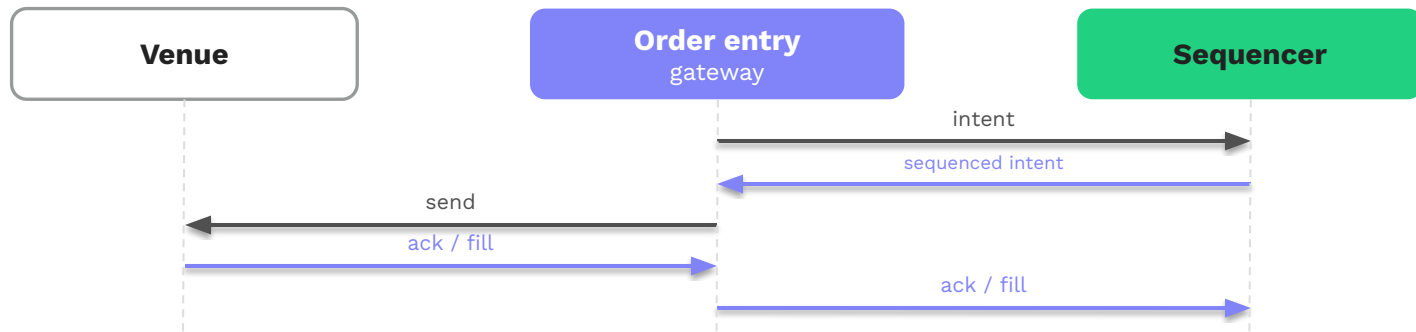


App response latency



Resilient gateway components

Storing intent simplifies reconciliation with external systems



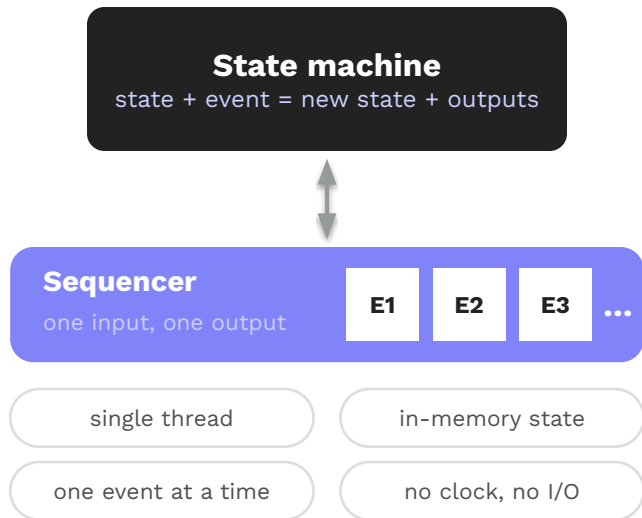
Make external communication durable

- Sequence the intent before you send. "I am about to send order X."
- Listen to the sequencer for the intent and send the message to the venue from that.
- Sequence what you hear, when you hear it. Acks, fills, rejects, session events.

Recovery is the difference between the venue and sequenced events.

Programming for sequencer architectures

Sequencer architectures create guardrails for code; for humans or models.



Bounded programming model

Single threaded, event driven programming model. State is in memory and mutation is atomic

Verification as a first-class concern

Components can be tested in isolation or together. Historical data can be used to verify code changes early in the development lifecycle

Sequencers in industry

Systematic hedge fund

Algo container and execution engine

Algo runtimes, signal engine and FIX order placement as separate processes on one machine, sequenced over shared memory.

Deterministic replay framework for backtesting.

 **Local**
single machine
on-prem

Global investment bank

Internalisation engine

Internal venue: matching engine, FIX, market and reference data gateways on a replicated sequence.

 **Replicated**
on-prem

Institutional equity brokerage

Broker execution platform

Smart order router, algos and OMS integrated on a sequencer architecture that runs on-premises or in AWS.

 **Replicated**
on-prem and cloud

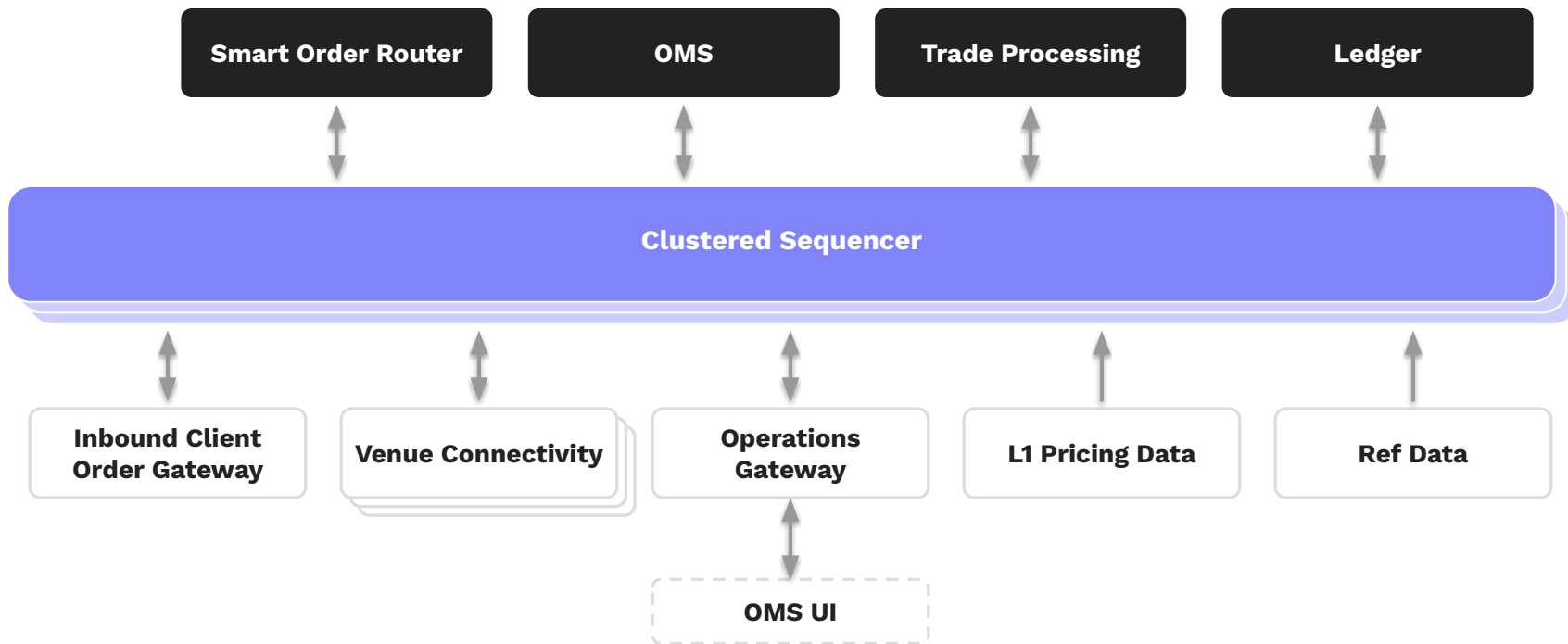
Global retail brokerage

Brokerage platform ledger

Positions, balances and reservations as a deterministic state machine on a replicated sequence.

 **Replicated**
cloud

An example architecture



Thanks

Connect with us



/@weareadaptive



/weareadaptive



/AdaptiveConsulting



/adaptive-ltd

 Adaptive

**Interested in catching the
replay?**
Access the recording here.

