Continuum: A blockchain based on Proof of Sequence

Fermi Labs

February 1, 2026

Abstract

The dominant mechanism for price discovery in global electronic markets is the continuous limit order book with price-time priority. These are based on the notion of First-in, First-Out (FIFO) transaction ordering. By contrast, most blockchains batch transactions into blocks and entrust intra-block ordering to a temporary leader, allowing extractable value. We present Continuum, an opinionated layer-1 blockchain architecture designed around a protocol-enforced ordering at sub-millisecond resolution. Proof of Sequence (POSq) is our sequencing logic, which replaces discretionary sub-block ordering with a verifiable, continuously advancing timechain derived from an RSA-group verifiable delay function. The POSq leader produces a signed, linearly ordered stream of transaction commits at a fixed tick rate, with optional time-lock encryption to prevent content-based manipulation prior to ordering. For execution, Continuum implements an application specific virtual machine optimized for trading use cases, such as perpetual futures. Continuum has a separate consensus layer, ContBFT, a HotStuff-derived finality gadget that checkpoints executed state roots, rotates and slashes the sequencer on misbehavior, and provides economic finality under partial synchrony. We formalize the design, specify the interaction of sequencing, execution, and consensus, and analyze the resulting ordering, finality, security, and performance properties.

1 Introduction

In the most liquid venues in modern finance, the continuous limit order book with price–time priority is the default mechanism for allocating trades at a given price level, because it supports continuous price discovery and provides a well-understood notion of queue position and execution priority [13, 14, 15]. These systems are engineered so that time priority is defined at fine granularity, and the resulting latency competition is a first-order design variable in market structure [17]. In decentralized finance, however, most systems inherit a coarse and leader-discretionary notion of time from block production. Even when a chain achieves high throughput, a block-based abstraction concentrates ordering power into a proposer for each block interval, and the transaction timeline within that interval is not a protocol-level invariant. This structurally allows for frontrunning and transaction reordering. The associated spam dynamics have also been documented as being a significant limit to the scalability of blockchains [5, 6].

The premise behind Continuum is that transaction ordering is the most critical feature of a blockchain for trading workloads. A CLOB that cannot provide an externally auditable, fine-grained, protocol-enforced arrival order will either centralize into a trusted sequencer or degrade into alternative mechanisms that compensate for uncertainty. While beyond the scope of the present paper, we believe based on empirical evidence that FIFO CLOBs are the optimal market microstructure for price discovery, and empirically, in the presence of a FIFO venue, liquidity concentrates there rather than on exchanges with alternative mechanisms.[16]

Proof of Sequence (POSq) is our approach for giving blockchains an internal notion of time, and a way to order transactions temporally. It utilizes a continuous verifiable delay function to produce a canonical temporally ordered stream of transactions, at a fine-grained time resolution.

In this paper, we specify the layer-1 design, including execution and consensus, and we focus the intended use case on exchange market microstructure while remaining compatible with future general-purpose virtual machines.

2 System model and design goals

2.1 Network participants and roles

Continuum consists of clients and several categories of nodes, whose roles are separated to reflect the separation of concerns among sequencing, execution, and consensus.

A *client* constructs transactions and submits them to the current POSq sequencer. Clients may submit plaintext transactions or may submit encrypted commitments that are revealed after a short delay, depending on the fairness policy in effect.

A *sequencer* is the unique leader responsible for producing the canonical transaction timeline during its tenure. The sequencer runs the VDFtimechain, assigns transactions to ticks, emits signed receipts that commit to ordering, and streams the resulting ordered log to the network.

An *executor* is a full node that deterministically applies the execution STF to the ordered log, producing an evolving state and state commitments. Executors verify the POSq log, validate signatures and VDFproofs, and compute state roots. In Continuum, validators are executors, because consensus depends on executed state commitments, and because slashing conditions include invalid execution claims.

A *consensus validator* participates in ContBFT, voting on periodic checkpoints of executed state. Consensus validators collectively finalize checkpoints, arbitrate sequencer rotation, and execute slashing for provable misbehavior. In the baseline deployment, the validator set is shared across execution and consensus, and a validator is expected to track the POSq stream closely enough to validate checkpoints in real time.

Finally, *watchers* and *light clients* may verify subsets of the protocol. A watcher can verify equivocation or invalid receipts without executing the full STF. A light client can track finalized checkpoint headers and request inclusion proofs and execution proofs, relying on full nodes for data availability but retaining cryptographic verification of ordering and finality.

2.2 Goals

The primary goal of Continuum is protocol-enforced canonical ordering at sub-millisecond resolution, together with deterministic execution suitable for a high-performance on-chain CLOB. The design emphasizes millisecond-scale effective finality for co-located participants, achieved by streaming sequencing and deterministic execution outside of consensus, and it emphasizes economically enforced safety against equivocation and invalid execution through a separate, slower finality layer.

3 Cryptographic primitives

A VDF is a function that is efficiently verifiable but inherently sequential to evaluate: computing an output for a given delay requires performing a prescribed number of sequential steps that cannot be asymptotically parallelized, while verification can be substantially cheaper [1]. Constructions based on groups of unknown order implement evaluation as repeated squaring, where the inability to exploit parallelism is grounded in the structure of exponentiation in such groups [2]. In Wesolowski-style VDF s, a prover computes $y = x^{2^D} \bmod N$ for delay parameter D, and provides a succinct proof that can be verified without repeating all D squarings [2]. The requirement that the group order is unknown is typically satisfied either by RSA groups modulo $N = pq$ where the factorization is unknown, or by class groups of imaginary quadratic

fields [2]. Continuous and iteratively verifiable variants of VDF s refine this into a model where a long-running computation emits intermediate states that are efficiently checkable in aggregate [3].

Time-lock encryption predates VDF s as a notion. Rivest, Shamir, and Wagner introduced time-lock puzzles that require a prescribed amount of sequential work to decrypt, providing a primitive for timed-release cryptography [4]. In Continuum, time-locking is not used to create long secrecy windows. Instead, it is used to ensure that, during the critical ordering window, the sequencer commits to the relative order of transactions before learning their content, thereby reducing content-based manipulation.

4 Proof of Sequence (POSq)

4.1 High-level definition

POSq is the sequencing protocol of Continuum. Its purpose is to produce a single canonical ordered stream of transactions with a well-defined temporal structure. The stream is indexed by *ticks*. Each tick corresponds to one iteration of a continuously running RSA-group VDF, tuned so that the evaluation of one tick requires approximately a fixed wall-clock duration Δ, with Δ chosen to be sub-millisecond in the target deployment. For concreteness, we will target a Δ of 100 microseconds, corresponding to 10^4 ticks per second.

Let N be an RSA modulus of unknown factorization, and let $\mathbb{G}$ denote the quadratic residues modulo N. Let $H : \{0,1\}^* \to \mathbb{G}$ be a hash-to-group function. Let $x_0 \in \mathbb{G}$ be a genesis seed. For tick $t \geq 0$, the sequencer maintains a state (x_t, C_t), where x_t is the current VDF output and C_t is a commitment to the transaction log up to tick t. The core VDF transition for one tick is repeated squaring for D steps, denoted abstractly as

$$x_{t+1} = F(x_t) = x_t^{2^D} \bmod N, \tag{1}$$

together with a succinct proof π_{t+1} of correct evaluation as in Wesolowski [2]. In practice, proofs can be batched over ranges of ticks to reduce overhead, without changing the semantics of the tick chain.

4.2 Transaction Commitments and Hash-Linked Chain

A purely sequential VDF chain provides a notion of time, but POSq requires that the transaction sequence is cryptographically braided into the timechain so that rewriting transaction order entails rewriting the VDF history. The sequencer therefore binds each committed transaction into the input of the next tick in a way that is deterministic and publicly verifiable.

Let tx_t denote the transaction committed at tick t, where tx_t may be empty if no transaction is committed in that tick. Define a tick payload hash

$$m_t = H_{\mathrm{hash}}(\mathrm{tx}_t \,\|\, t \,\|\, \mathrm{pk}_{\mathrm{seq}}), \tag{2}$$

where H_{hash} is a collision-resistant hash and $\mathrm{pk}_{\mathrm{seq}}$ is the current sequencer public key. The braided transition updates the next tick's seed as

$$\tilde{x}_t = H(x_t \,\|\, m_t), \qquad x_{t+1} = F(\tilde{x}_t). \tag{3}$$

This ensures that the tick outputs are a deterministic function of both prior time and the committed payload, and it prevents a sequencer from first running the VDF far into the future and then retroactively "filling in" transactions without redoing work.

The log commitment is updated in parallel:

$$C_{t+1} = H_{\mathrm{hash}}(C_t \,\|\, \mathrm{tx}_t \,\|\, x_{t+1} \,\|\, t{+}1). \tag{4}$$

This creates an append-only commitment chain that is indexed by ticks and anchored in VDF time. A verifier who receives the stream can verify that the pair (x_t, C_t) evolves correctly, and that the tick index is strictly increasing.

4.3 Receipts, non-equivocation, and the canonical stream

A critical design choice in Continuum is that a client should receive a cryptographic receipt quickly, and that receipt should either be honored by inclusion at the specified tick or should constitute evidence of slashable misbehavior. Let id(tx) be a transaction identifier, which is the hash of the ciphertext in commit–reveal mode or the hash of the plaintext transaction in plaintext mode. When the sequencer accepts a transaction for inclusion, it assigns it to the next available tick t^* consistent with FIFO admission, and immediately returns a signed receipt

$$\sigma = \mathrm{Sign}_{\mathrm{sk}_{\mathrm{seq}}}(\mathrm{id}(\mathrm{tx}) \,\|\, t^* \,\|\, x_{t^*} \,\|\, C_{t^*})\,. \tag{5}$$

The receipt binds the transaction identifier to a specific position in the timechain, as reflected by the tick output and the running commitment. The sequencer broadcasts the same information to the network as part of the tick stream.

Non-equivocation follows from two complementary mechanisms. First, the chain of commitments C_t is linear and deterministic, so two distinct histories that share a prefix will diverge in C_t and can be detected by any observer. Second, the receipt signature provides a compact equivocation witness: if a sequencer issues two distinct valid receipts for the same tick or for inconsistent chain commitments, then any party can present both signatures as a fraud proof. In Continuum, such evidence is slashable at the consensus layer, and it triggers rotation of the sequencer.

4.4 FIFO transaction processing

The goal here is to narrow the discretionary headroom of the sequencer to reorder/delay/censor transactions, thus achieving effective FIFO for the purpose of market microstructure. POSq defines FIFO with respect to the sequencer's admission queue, while making the admission queue itself externally observable through receipts and streaming. We also make the canonical order relate to actual passage of time, by embedding transactions in a continous VDF stream.

The sequencer maintains an ingress queue of pending transactions. For each tick, it dequeues all received transactions (a transaction batch), produces the receipt, and commits them into the tick payload. To prevent discretionary tie-breaking, Continuum specifies that within any set of transactions whose enqueue timestamps fall within the same tick interval, the sequencer must order them by a deterministic function of their identifiers, such as transaction hashes. This produces a total order even under micro-ties, without giving the leader an arbitrary choice.

4.5 Timelock Encryption for Blind Ordering

Typically, blockchains leak transaction content before ordering is finalized, enabling both content-based reordering by proposers and reactive insertion by searchers. Empirical analysis of on-chain markets has shown that this leakage is central to frontrunning and sandwich strategies [5]. Even if POSq enforces a tick-based FIFO discipline, a sequencer who sees plaintext orders could still attempt to exploit them by selectively censoring, delaying receipts, or inserting its own transactions at adjacent ticks.

Continuum therefore treats commit-reveal as a first-class mechanism. In commit mode, a client submits a ciphertext $c = \mathrm{Enc}_k(\mathrm{tx})$ together with a time-lock structure that ensures k becomes derivable only after a short delay measured in ticks. This can be implemented using time-lock puzzles in an RSA group, or by deriving k from a future VDF output that is not available until the VDF progresses [4, 1]. The sequencer orders and receipts the ciphertext

identifier $\mathrm{id}(c)$ without the ability to decrypt it during the critical ordering window. After a fixed delay of r ticks, either the client reveals k or the time-lock condition renders k computable, and executors then interpret the revealed plaintext as the transaction payload corresponding to the already-committed position.

The commit-reveal mechanism is intended to provide blind ordering guarantees. The delay r is chosen to be small enough that the user experience remains low-latency, but large enough that the sequencer cannot learn content before it has committed to an order that will be audited and checkpointed. In exchange workloads, r is naturally aligned with the definition of effective finality: once an order is sufficiently deep in the tick stream that any reordering is easily detected, revealing the payload does not reintroduce the opportunity to reorder it.

4.6 RSA-group VDF in v1: performance and trust assumptions

Continuum v1 uses an RSA-group repeated-squaring VDF with Wesolowski-style proofs [2]. The main engineering motivation is that modular squaring can be aggressively optimized on CPUs and FPGAs using Montgomery arithmetic, fixed-precision big-integer pipelines, and instruction-level parallelism across independent lanes, while preserving the sequential dependence across squaring steps. Continuous operation is straightforward: the sequencer performs an endless stream of squarings, emits intermediate outputs as ticks, and periodically emits proofs that allow verifiers to validate tick correctness without repeating all work.

The principal trust assumption is the requirement that the RSA modulus N be of unknown factorization. If any party knows $\varphi(N)$, then they can in principle accelerate exponentiation and break the intended delay property. In an RSA-group setting, this necessitates either a multi-party ceremony to generate N such that no participant learns the factorization, or reliance on a widely trusted modulus whose factorization is assumed unknown. This is a structured setup assumption, and Continuum treats it explicitly as such. The design choice is justified by the near-term performance target and by the fact that sequencing hardware is itself a specialized operational requirement. Nevertheless, Continuum anticipates future migration paths, including class-group VDF s that remove the RSA setup assumption at the cost of different performance tradeoffs [2], and continuous-verification designs that compress proof overhead and enable broader verification participation [3]. Migration is facilitated by the layered architecture: changing the VDF instantiation changes the definition of ticks and proofs, but it does not change the execution STF or the consensus checkpointing format, provided the tick stream remains a canonical total order.

5 ContCore: Deterministic execution for on-chain markets

5.1 Execution as a deterministic function of the POSq stream

Continuum defines execution as a deterministic STF over the ordered transaction stream produced by POSq. In contrast to designs where execution is entangled with consensus rounds, Continuum assumes that executors can apply the STF as soon as they observe tick outputs and receipts, producing "soft" state updates with millisecond latency. Consensus later finalizes checkpoints of those states. This separation is aligned with modern modular blockchain architecture, and it is supported by the Sovereign SDK's explicit separation between deterministic state transitions and node-side logic [11, 12].

Let S_t denote the canonical execution state after applying all revealed transactions committed through tick t. Let $\mathrm{Apply}(S, \mathrm{tx})$ be the deterministic function that applies one revealed transaction to state. Then

$$S_{t+1} = \begin{cases} \mathrm{Apply}(S_t, \mathrm{tx}_t) & \text{if tick } t \text{ contains a revealed transaction,} \\ S_t & \text{if tick } t \text{ is empty or contains an unrevealed commitment.} \end{cases} \tag{6}$$

The state commitment R_t is a cryptographic digest of S_t, typically a Merkle root. Executors compute R_t deterministically, and checkpoint proposals in consensus commit to R_t.

The commit–reveal mechanism introduces a notion of an unrevealed tick. The ordering of the tick is fixed by receipt and log commitment, but the execution payload is delayed. Continuum resolves this by treating the committed ciphertext as occupying the tick, while the executor maintains a small buffer of pending reveals keyed by tick index. Once the plaintext is revealed and validated against the ciphertext commitment, the executor applies it in its original tick order. This produces a deterministic mapping from the committed tick stream to the executed transaction stream.

5.2 Optimized VM for exchange workloads

The execution layer in Continuum v1 is intentionally not a general-purpose virtual machine. It is a custom, optimized VM oriented around exchange primitives: order placement, cancellation, matching, margin accounting, funding, liquidation, and risk checks. The design objective is to make the STF cheap enough that executors can keep up with high-frequency order flow without requiring expensive general-purpose smart contract execution for every micro-operation.

The architecture is implemented using the Sovereign SDK stack because it provides a modular STF model, explicit separation of deterministic state and node-side services, and a performance profile suitable for low-latency financial applications [11, 12]. Conceptually, the execution state is organized into modules that together define the exchange system. The order book module maintains per-market price levels and per-order queue position. The margin module tracks collateral, positions, and liquidation eligibility. The risk engine computes deterministic constraints used by the matching engine. The settlement module applies fills and updates balances. The key constraint is that every operation must be deterministic and must have predictable computational complexity, because a leader cannot be allowed to exploit variable execution cost to induce covert reordering.

A CLOB semantics requires protocol-enforced price–time priority at the matching layer. Continuum provides time priority at the sequencing layer, and the execution layer must preserve that priority without introducing discretionary matching. The matching engine therefore treats the POSq order stream as the unique source of time ordering. Within a price level, queue position is derived from the tick index at which the order was committed, and matching consumes resting liquidity in increasing tick order. This mirrors the price–time algorithms specified by major electronic venues [15] while making "time" cryptographically grounded in the POSq tick chain rather than in a local wall clock.

Because the use case is perpetual futures and other leveraged instruments, the execution layer must support a deterministic risk and liquidation regime. The design choice in Continuum is to keep liquidation ordering and execution within the same POSq timeline. This eliminates the typical liquidation race dynamics that arise when participants compete to be first in a block. Under POSq, liquidation attempts are ordered by tick, and if commit–reveal is used, liquidation payloads are not visible until their ordering is committed, reducing the attack surface for reactive liquidation frontrunning.

5.3 Extensibility to general-purpose VMs

Although Continuum v1 is optimized for an exchange VM, the architecture anticipates introducing a general-purpose VM, such as an EVM- or SVM-compatible environment, as an additional execution domain. This extension does not require changing the sequencing and consensus layers. It requires specifying transaction formats that dispatch into the relevant VM and specifying deterministic state roots that aggregate the multiple domains. The ordering guarantee remains the same: all VM domains consume the same canonical POSq stream.

6 ContBFT: consensus for economic finality

6.1 Separation of concerns in the happy path

Traditional blockchains use consensus to determine both order and finality. Continuum deliberately decomposes these concerns. POSq determines order continuously at tick cadence. Deterministic execution applies the STF to that order as soon as data is observed. Consensus is then responsible for three tasks that remain essential in an adversarial setting: it finalizes executed checkpoints under partial synchrony, it enforces slashing and rotation when the sequencer equivocates or produces invalid outputs, and it governs validator membership and stake-weighted security.

This separation yields two different notions of finality. *Effective finality* is the time at which co-located participants, observing the POSq stream and executing deterministically, can treat the outcome as operationally settled because any deviation would be slashable and because rewriting the tick history would require outpacing the VDF. *Economic finality* is the time at which the checkpoint containing that outcome is finalized by ContBFT, making the state root part of the chain's formally committed history.

6.2 Checkpointed consensus

ContBFT finalizes the chain at a coarser granularity than ticks. Define a checkpoint interval of B ticks. At checkpoint height h, corresponding to tick range $[hB, (h{+}1)B)$, the sequencer and executors compute an executed state root $R_{(h+1)B}$ and a header

$$\mathcal{H}_{h+1} = \big(h{+}1,\, x_{(h+1)B},\, C_{(h+1)B},\, R_{(h+1)B}\big)\,. \tag{7}$$

The header binds the timechain, the transaction log commitment, and the executed state. Consensus votes on $\mathcal{H}_{h+1}$ rather than on individual transactions. This removes high-frequency ordering from consensus while still ensuring that finalized history corresponds to a specific prefix of the POSq stream and its deterministic execution.

6.3 HotStuff-derived finality and responsiveness

ContBFT is based on the HotStuff framework [10]. HotStuff is a leader-based BFT protocol in the partially synchronous model that achieves linear communication with pipelining and provides responsiveness once the network becomes synchronous. Continuum adopts HotStuff's core notion of a quorum certificate (QC), typically an aggregated signature from a supermajority of validators, and uses it to finalize checkpoint headers $\mathcal{H}_h$.

Because ordering is not being decided by consensus, ContBFT can be simplified in the following way. Proposals are deterministic functions of the observed POSq stream and execution STF. Therefore, under honest behavior and sufficient data availability, all validators can compute the same candidate header $\mathcal{H}_{h+1}$. The consensus leader for that round proposes $\mathcal{H}_{h+1}$, validators vote if it matches their locally computed header, and the QC commits it. If a sequencer proposes an invalid header, validators refuse to vote, and the protocol triggers view change and potentially sequencer rotation. This design turns consensus into an "execution attestation" layer and a slashing arbiter.

6.4 Slashing and sequencer rotation

The sequencer posts a bond and is slashable for objective misbehavior. Continuum treats the following as primary slashable conditions. If the sequencer issues conflicting receipts or produces two tick streams with a common prefix but diverging commitments, it has equivocating evidence. If it produces invalid VDF proofs or breaks the braided transition, it produces objective invalidity. If it proposes a checkpoint header whose executed state root does not match

the deterministic execution of the tick range, it proposes invalid execution. All such behaviors are detectable by executors and provable to consensus via compact witnesses, typically pairs of signed messages or mismatching hashes.

Rotation is triggered either periodically, at epoch boundaries defined in checkpoint heights, or reactively upon proven misbehavior. In an epoch-based design, the validator set selects the sequencer for the next epoch using stake-weighted selection. In a reactive design, once misbehavior is proven and slashed, the next sequencer is selected immediately. The critical property is continuity of the timechain: the new sequencer must continue from the latest finalized checkpoint header, inheriting $x_{(h+1)B}$ and $C_{(h+1)B}$ as the starting point for subsequent ticks. Because the POSq braid binds transaction payloads into the VDF progression, continuing from the checkpoint header preserves the canonical history.

7 End-to-end protocol flow

A client submits a transaction to the sequencer through a low-latency ingress path. In commit-reveal mode, the client submits a ciphertext commitment, otherwise it submits plaintext. The sequencer enqueues the transaction, assigns it to the next available tick consistent with FIFO admission, and returns a signed receipt binding the transaction identifier to that tick and to the current timechain state. The sequencer then advances the VDF, computes the tick output, braids in the tick payload, updates C_t, and broadcasts the tick record to the network.

Executors receive tick records, verify receipt signatures and the VDF proof schedule, validate the braided transitions, and update their local view of the canonical ordering stream. When the transaction payload is revealed, executors apply it to their deterministic STF and update the state root. Executors may expose low-latency APIs and market data as node-side services, but those services do not influence the deterministic state transition.

At each checkpoint boundary, the consensus leader proposes $\mathcal{H}_{h+1}$ containing the timechain outputs and the executed state root. Validators vote if and only if their local execution yields the same header, and a QC finalizes it. A finalized header provides economic finality for all ticks in the interval. If the sequencer misbehaves, validators refuse to vote and instead vote on a slashing and rotation transition.

From the perspective of a co-located trading participant, the sequence of receipts and tick broadcasts provides near-instant ordering confirmation, and deterministic execution provides near-instant state evolution. From the perspective of an external observer, finalized checkpoints provide canonical, economically final state roots.

For simplicity, the sequencer, executor, and consensus leader roles are allocated to a single leader in v1. However, as previously defined, the leader is required to produce a continuous transaction commitment stream under its role as "sequencer" in the POSq stream, which allows all observers to permissionlessly compute the updated chain state in near real-time.

8 Security analysis

8.1 Ordering integrity and fairness

At a structural level, POSq restricts the leader's degrees of freedom by making time an explicit, verifiable resource. The leader cannot cheaply rewrite the past without redoing sequential work and producing conflicting receipts that can be slashed. This addresses the core mechanism that enables ordering-based extraction in blockchains, where proposers are temporarily centralized and can permute transactions within a block interval.

Order fairness is subtle in distributed systems, and there is a literature that formalizes different fairness properties and shows impossibility results for overly strong definitions [7, 8]. Continuum does not claim to solve all forms of fairness under arbitrary network scheduling.

Instead, it targets a specific property aligned with exchange microstructure: given a single admitted sequence at the sequencer, and given protocol-enforced deterministic tie-breaking, the sequencer cannot profitably reorder transactions relative to that admitted order without creating slashable evidence. Commit–reveal strengthens this by denying the leader advance knowledge of transaction content during the ordering decision, reducing content-based discrimination. In practice, this combination is designed to reduce the economically relevant portion of MEV described in empirical work [5] and the scaling externalities described by Flashbots [6].

8.2 Non-equivocation and consistency

Consistency requires that there be a single canonical history. Continuum enforces this through receipt-based equivocation detection and through checkpoint finality. A sequencer who attempts to present different orderings to different parties must either fork the commitment chain C_t or issue inconsistent receipts. Either behavior yields a compact fraud proof. Consensus finality prevents long-range equivocation by binding the chain to finalized checkpoint headers; after a checkpoint is finalized, any alternative history that disagrees with it is rejected by honest validators.

8.3 Liveness, censorship, and recovery

A single leader can always attempt to censor by refusing to accept or receipt transactions. Continuum mitigates this through rotation: the sequencer is a role, not an identity. If the sequencer fails to provide service, consensus can rotate it. In practice, a deployment must define an operational policy for detecting censorship, because absence of receipts is not itself a cryptographic proof. Continuum therefore treats liveness as an economic and governance property: validators and stakeholders have incentives to maintain a functioning exchange, and a sequencer who persistently refuses service can be rotated out. For transactions that *have* receipts, failure to include them at the specified tick is objective misbehavior.

8.4 Trusted setup and hardware asymmetry

The RSA-group VDF introduces a structured setup assumption. If the modulus factorization were known, the intended delay could be violated. Continuum makes this assumption explicit and treats it as an engineering tradeoff for v1 performance. Hardware asymmetry is also inherent: a party with significantly faster sequential squaring could potentially shorten effective delay. In the near term, the intent is that sequencer eligibility and slashing conditions are coupled to performance requirements, and that the validator set can update parameters as hardware evolves. Longer-term, Continuum anticipates migration to constructions with different trust tradeoffs [2, 3].

9 Performance and finality characteristics

The FIFO ordering throughput ceiling of the sequencing layer is constrained by the tick rate. With $\Delta = 100$ μs, the sequencing layer supports on the order of 10^4 individually sequenced transaction commitments per second, though more can be included with batching or a finer temporal resolution (eg. 10 us). The latency to obtain a receipt is dominated by network round-trip time to the sequencer plus minimal signing overhead. The latency to observe the transaction in the canonical stream is dominated by the next tick boundary and broadcast propagation.

Effective finality for co-located participants arises because, once a receipt is issued and the tick stream progresses, any attempt to retract or reorder that receipt becomes slashable and would require producing an alternative timechain. This yields operational finality on millisecond

scales for participants who are physically close to sequencer and validators. Economic finality is determined by the consensus checkpoint cadence, which is a policy choice. A deployment may choose B so that checkpoints finalize, for example, every 50 to 200 milliseconds, balancing finality speed against network overhead and validator signature aggregation.

The execution layer performance is governed by the optimized exchange VM and by the deterministic STF cost per transaction. By implementing exchange primitives directly rather than via general-purpose smart contracts, Continuum aims to keep the per-transaction execution cost bounded and predictable. The Sovereign SDK's architectural separation between deterministic STF logic and node-side services supports this design, allowing low-latency APIs and schedulers without contaminating determinism. Since there is no real-time proving/zkVM in Continuum v1 design, and since we utilize an optimized low level rust matching engine, performance limits approach the maximal permitted by the hardware and network limits of the validators. [11, 12].

10 Related work

POSq is related to prior uses of sequential functions for ordering. Solana's Proof of History uses a continuously running hash chain as a source of timestamps, but it does not provide a verifiable delay guarantee in the VDF sense and it remains embedded in a block-producing architecture [9]. Continuum instead uses a VDF construction designed for publicly verifiable delay [1, 2] and treats the tick stream as the canonical ordering substrate.

Fair ordering has been formalized in both blockchain and distributed systems research. Kelkar et al. study order fairness for Byzantine consensus and provide protocols and definitions that clarify the limits of fairness under adversarial scheduling [7]. Kursawe shows impossibility results for intuitive fairness definitions and proposes "fairness widgets" that can be layered onto existing chains [8]. Continuum takes a different architectural position by enforcing a leader's ordering discipline through a timechain, receipts, and slashing, rather than by requiring multi-party agreement on each ordering decision.

MEV and transaction reordering have been studied empirically and economically. Daian et al. document frontrunning and priority gas auctions as a mechanism of ordering competition [5], and Flashbots has argued that MEV-linked dynamics are a fundamental scaling pressure in high-throughput systems [6]. Continuum's focus on cryptographically enforced ordering and optional commit–reveal is motivated directly by these observations.

On the consensus front, ContBFT is derived from HotStuff, a leader-based BFT protocol with responsiveness and linear communication [10]. Continuum uses consensus to finalize executed checkpoints and enforce accountability, rather than to drive fast-path ordering.

Finally, the execution model builds on modern modular STF frameworks and on the Sovereign SDK's separation between deterministic state transitions and node-side services [11, 12]. Continuum specializes this into an exchange-oriented VM while retaining a path to general-purpose VMs in future upgrades.

11 Conclusion

Continuum is an opinionated layer-1 architecture for on-chain markets that enforces protocol enshrined ordering. The sequencing layer, POSq, produces a canonical tick-indexed transaction stream at sub-millisecond granularity using an RSA-group VDF commitment structure. The execution layer then deterministically applies a high-performance exchange-oriented STF built on the Sovereign SDK stack, enabling market microstructure comparable in form to the dominant centralized model while providing cryptographic guarantees. The consensus layer, ContBFT, finalizes executed checkpoints, rotates and slashes the sequencer, and provides economic finality without imposing consensus latency on the fast path. The resulting system is designed to offer millisecond-scale effective finality for co-located participants and robust accountability under

adversarial behavior. Continuum is designed to be the first high-throughput trading chain with protocol enforced FIFO ordering.

References

[1] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In *Advances in Cryptology – CRYPTO 2018*, pages 757–788, 2018.

[2] Benjamin Wesolowski. Efficient Verifiable Delay Functions. *Journal of Cryptology*, 33(4):2113–2147, 2020. (Conference version in *EUROCRYPT 2019*.)

[3] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous Verifiable Delay Functions. *IACR Cryptology ePrint Archive*, Report 2019/619, 2019.

[4] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, MIT Laboratory for Computer Science, 1996.

[5] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. In *IEEE Symposium on Security and Privacy*, 2020. (Preprint: arXiv:1904.05234.)

[6] Robert Miller. MEV and the Limits of Scaling. Flashbots Writings, June 16, 2025. https://writings.flashbots.net/mev-and-the-limits-of-scaling.

[7] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-Fairness for Byzantine Consensus. In *Advances in Cryptology – CRYPTO 2020*, pages 451–480, 2020.

[8] Klaus Kursawe. Wendy, the Good Little Fairness Widget. In *ACM Conference on Advances in Financial Technologies (AFT)*, 2020.

[9] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain. Whitepaper v0.8.13, 2018. https://solana.com/solana-whitepaper.pdf.

[10] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT Consensus in the Lens of Blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, 2019. (Preprint: arXiv:1803.05069.)

[11] Sovereign Labs. The Sovereign SDK Book. https://docs.sovereign.xyz/.

[12] Sovereign Labs. Sovereign SDK repository. https://github.com/Sovereign-Labs/sovereign-sdk.

[13] Larry Harris. *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2003.

[14] Maureen O'Hara. *Market Microstructure Theory*. Blackwell, 1995.

[15] Nasdaq. Nasdaq Equity Rules, Rule 4757 (Book Processing), price/display/time execution algorithm. https://listingcenter.nasdaq.com/rulebook/nasdaq/rules/Nasdaq%20Equity%204.

[16] European Securities and Markets Authority (ESMA). Final Report: Call for Evidence on Periodic Auctions (ESMA70-156-1035), June 11 2019. https://www.esma.europa.eu/sites/default/files/library/esma70-156-1035_final_report_call_for_evidence_periodic_auctions.pdf.

[17] Eric Budish, Peter Cramton, and John Shim. The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. *Quarterly Journal of Economics*, 130(4):1547–1621, 2015.