Sentry

A New Blockchain Architecture for Institutional Capital Markets

Sentry Team

November 2025

Abstract

Institutional capital markets cannot operate on blockchains because existing designs lack essential capabilities: operational cash cannot earn yield, participants cannot transact in their preferred assets, credit relationships are impossible, and trading venues cannot protect order flow. We present Sentry, a blockchain architecture that provides these capabilities while maintaining permissionless access.

Sentry implements four mechanisms. First, a yield-bearing asset backed by U.S. Treasury securities allows operational capital to earn risk-free returns. Second, market makers stream price quotes that validators cache locally, enabling users to pay fees and post collateral in any asset. Third, qualified participants access epoch-based settlement with multilateral netting and bilateral credit, while permissionless users retain atomic settlement. Fourth, application-specific validator subnets with zero-knowledge proofs provide privacy without compromising correctness. A multidimensional fee market prices execution capacity, netting risk, and collateral costs independently with provably optimal price discovery.

Applications run consensus independently, producing cryptographic commitments that a coordinator orders globally. This separation enables fast cross-application transactions through speculative finality while deferring expensive proof verification.

1 Introduction

Institutional capital markets remain conspicuously absent from blockchain despite a decade of experimentation. The reason is not regulatory uncertainty or cultural resistance—it is architectural: blockchains lack the fundamental capabilities that financial institutions require. A treasury desk managing $100 million in operational cash loses $5 million annually in forgone yield by holding non-interest-bearing stablecoins. A derivatives market maker cannot quote prices in the client's preferred asset without introducing latency that destroys competitive positioning. A clearinghouse cannot provide leverage through multilateral netting because blockchains enforce atomic settlement. An equity trading venue cannot isolate order flow from competitors observing the same shared state. Institutions will not migrate to systems that impose these costs—they require an architecture designed from first principles to eliminate them.

Four Missing Capabilities. Institutional participants maintain substantial operational balances to support trading activity, collateral requirements, and settlement obligations. In traditional finance, these balances earn the overnight rate automatically—money market funds, repo agreements, and Treasury bills convert idle cash into yield-bearing positions. On blockchain, stablecoins provide price stability but no yield. A treasury desk with $100 million in USDC or DAI positioned for trading opportunities forgoes approximately $5 million annually at current risk-free rates. This opportunity cost scales linearly with capital: larger participants face larger

losses. Institutions require that operational cash earn the risk-free rate by default, not through active redeployment into external protocols.

Market participants operate across diverse asset bases and prefer to transact in their native denomination. A Japanese pension fund holds yen, a European bank holds euros, a U.S. institutional investor holds Treasury bills. Requiring conversion to a single payment token introduces friction: participants must acquire the token, monitor exchange rates, and manage foreign exchange risk. The natural user experience is simpler—when User A sends 1000 TSLA shares to User B, a small fraction should disappear as gas, not require maintaining a separate wallet balance in an unrelated token. This creates a pricing problem: validators must determine how much TSLA equals the required gas fee in real time. Traditional request-for-quote (RFQ) systems, where users query market makers for prices, require 100–500 milliseconds—longer than typical block times. Validators cannot wait for external price discovery during block construction. Both institutions and end users require the ability to pay for services in arbitrary assets with pricing resolved at validator timescales, not user timescales.

Blockchains enforce atomic settlement to prevent double-spending: every transaction settles immediately and irrevocably. This eliminates credit risk but destroys capital efficiency. Consider 100 participants who each trade with 10 counterparties during a day, executing 10 transactions per pair—5000 total transactions. Under gross settlement, all 5000 transfers execute independently, requiring participants to prefund the maximum gross exposure. Traditional finance solves this with central clearinghouses: participants submit obligations to a trusted intermediary, which computes net positions and settles once per epoch. By the Central Limit Theorem, offsetting flows reduce settlement volume from $O(n^2)$ gross obligations to $O(n^{3/2})$ net transfers—for 100 participants, a 90% reduction. The Depository Trust & Clearing Corporation processes $2.15 quadrillion annually while requiring only $103 billion in participant deposits, a 21,000:1 capital efficiency ratio [6]. We replicate this capital efficiency in a decentralized setting by embedding epoch-based settlement and net position computation directly into the validator protocol, eliminating the need for a trusted central intermediary.

Public blockchains expose all transaction data to all validators, making them unsuitable for competitive financial markets. An equity trading venue cannot operate on Ethereum: every market maker would observe competitor order flow, every participant would leak trading strategies, and regulatory requirements for confidential execution would be violated. Existing privacy solutions fail to meet institutional requirements. Trusted execution environments (TEEs) centralize trust in hardware enclaves, replacing decentralized consensus with reliance on manufacturer attestation. Fully homomorphic encryption imposes orders-of-magnitude performance overhead that makes real-time trading infeasible. Application-specific sidechains achieve privacy through isolation but sacrifice composability—assets cannot move freely between domains, and shared state like token balances cannot be coordinated. Institutions require privacy that isolates application data from competitors while maintaining verifiable correctness and composability across applications.

Why Existing Chains Cannot Fix This. These capabilities cannot be added to existing blockchains as applications or smart contracts—they require changes to the consensus protocol, validator behavior, and economic model. A yield-bearing native asset requires the base layer to support tokenized real-world assets as first-class citizens in the economic model—bUSD backed by BUIDL must function as the universal numéraire for collateral, settlement, and gas payments, not merely as an ERC-20 token among thousands. Arbitrary-asset gas payments require validators to cache market maker price quotes and perform asset swaps during block construction, before transaction execution. Smart contracts execute after blocks are formed

and cannot influence validator operations. Multilateral netting requires separating transaction execution from settlement and introducing epoch-based finality, fundamentally altering how the state machine processes transactions. Application-level privacy with composability requires partitioning the validator set while coordinating shared state, a consensus-level architectural change. Each capability demands protocol-layer modifications that cannot be implemented on top of existing chains.

Our Contributions. Yield-Bearing Native Asset. We introduce bUSD, a stablecoin backed 1:1 by BlackRock's BUIDL fund [3], which holds tokenized U.S. Treasury securities. Through BUIDL's native yield distribution, bUSD holders earn the risk-free rate directly on-chain, eliminating the opportunity cost of holding operational balances. bUSD serves as the universal numéraire across the protocol: all markets quote prices in bUSD, participants post collateral in bUSD, and settlement occurs in bUSD. This creates structural demand for BUIDL—protocol utility (trading volume, collateral requirements, settlement activity) mechanically drives demand for the underlying Treasury fund. Unlike fee-based tokenomics where protocol revenue accumulates to a treasury, our model channels economic activity directly into demand for a regulated, yield-bearing real-world asset.

Arbitrary-Asset Gas Payments. We enable users to pay transaction fees in any asset without introducing latency. Market makers broadcast streaming price quotes at high frequency (100–500 Hz), which validators cache locally. When a user submits a transaction paying gas in TSLA, the validator performs a constant-time lookup (sub-microsecond) to determine the TSLA/$SEN exchange rate, executes an atomic swap with the market maker, and delivers $SEN to the protocol. This eliminates request-for-quote latency: pricing occurs at validator timescales using pre-committed quotes rather than at user timescales via synchronous requests. Market makers earn the spread between their quoted price and the actual execution price, creating a competitive market for providing liquidity. The result is a seamless user experience—pay gas in any asset—while preserving the economic model where validators are compensated in $SEN.

Dual Settlement with Qualified Participant Infrastructure. We introduce a dual settlement model that partitions the system into permissionless and qualified tiers, enabling different regulatory and operational capabilities. Permissionless transactions execute atomically with immediate settlement, requiring no identity verification or credit relationships. Qualified participants meet application-specific requirements—KYC verification, accredited investor status, geographic restrictions, or other compliance criteria—and gain access to qualified resources: epoch-based settlement with multilateral netting, credit extension during trading epochs, and access to regulated asset classes that require qualified counterparties. The protocol maintains dual state—finalized state S_F and provisional state S_P—allowing qualified participants to trade on credit during an epoch while the system tracks obligations. Multilateral netting reduces settlement volume from $O(n^2)$ gross transfers to $O(n^{3/2})$ net transfers, replicating traditional clearinghouse capital efficiency. This architecture supports both permissionless blockchain access and qualified financial infrastructure within a single protocol.

Application-Level Privacy with Zero-Knowledge Verification. We achieve transaction confidentiality while preserving composability through a two-layer consensus architecture. Applications run private consensus within dedicated validator subnetworks, processing encrypted transactions that remain hidden from other applications. To prevent double-spending across applications, a global coordinator runs separate consensus to order application commitments and verify zero-knowledge proofs of state transitions. This architecture isolates competitive data (order flow, trading strategies, positions) while enabling cross-application composability for shared state like token balances. The coordinator accepts application commitments optimistically

before proofs are ready—since zero-knowledge proof generation has significantly higher latency than consensus—and handles rollbacks gracefully if proofs fail. The result is privacy without trusted hardware, composability without global visibility, and low-latency execution despite expensive cryptographic verification.

The BUIDL Economic Loop. The architecture creates a reflexive economic loop where protocol utility drives demand for BUIDL. Because bUSD serves as the universal numéraire, every participant interaction increases BUIDL exposure. Market makers posting liquidity hold collateral in bUSD. Traders settling positions receive bUSD. Applications quoting prices denominate in bUSD. Qualified participants posting margin deposit bUSD. This creates structural demand distinct from speculative token appreciation: as trading volume increases, collateral requirements increase, settlement activity increases, and margin requirements increase—all mechanically driving demand for the underlying Treasury fund. The protocol does not accumulate bUSD from transaction fees; instead, market makers receive user payments in various assets and deliver $SEN to validators. The economic feedback loop operates through bUSD's role as the operational currency: protocol growth requires participants to hold more bUSD, which requires purchasing more BUIDL shares. This channels blockchain economic activity directly into a regulated, yield-bearing real-world asset.

Paper Organization. The remainder of this paper is organized as follows. Section 2 presents the system architecture, deriving design choices from institutional requirements and showing how the two-layer consensus protocol enables the required capabilities. Section 3 describes the yield-bearing asset model and bUSD's role as universal numéraire. Section 4 specifies the arbitrary-asset gas payment mechanism with market maker quote streaming. Section 5 details the dual settlement model with multilateral netting and qualified participant infrastructure. Section 6 presents application-level privacy through validator partitioning and zero-knowledge verification. Section 7 describes the bilateral credit system enabling qualified participants to trade on margin. Section 8 analyzes security properties including settlement safety, consensus guarantees, and privacy with integrity. Section 9 presents the multidimensional fee market that prices execution, netting risk, and collateral costs independently. Section 10 discusses related work in blockchain privacy, settlement systems, and tokenized assets. Section 11 concludes with future directions.

2 System Architecture

We present Sentry's architecture by showing how the institutional requirements from Section 1 determine the system design. Each capability imposes specific constraints on consensus protocols, validator behavior, and state management.

The system consists of four primary components. First, a two-layer consensus architecture: applications run private consensus (Simplex) within dedicated validator subnetworks, while a global coordinator runs separate consensus (MonadBFT) to order commitments and verify zero-knowledge proofs. Second, a dual settlement model: permissionless transactions settle atomically, while qualified participants access epoch-based settlement with multilateral netting. Third, validator-level price discovery: market makers broadcast streaming quotes that validators cache for sub-microsecond asset swaps during block construction. Fourth, a yield-bearing numéraire: bUSD backed by BUIDL serves as the universal asset for collateral, settlement, and pricing.

The following subsections specify the two-layer consensus protocol in detail, then describe how dual settlement and other features leverage this architecture.

2.1 Two-Layer Consensus Protocol

The system partitions validators into application-specific subnetworks and a global coordinator. Each application A_i maintains its own validator set V_i that processes encrypted transactions visible only to authorized validators. Applications run Simplex consensus [9] to achieve fast deterministic finality: validators decrypt transactions, validate state transitions deterministically, and finalize within 3Δ network delay. The coordinator maintains validator set V_C that orders application commitments globally and verifies zero-knowledge proofs asynchronously. The coordinator runs MonadBFT consensus [8] to handle optimistic execution with deferred proof verification. MonadBFT is necessary because zero-knowledge proof generation has significantly higher latency than consensus rounds—the coordinator must accept commitments before proofs are ready and handle rollbacks gracefully if proofs fail.

Definition 2.1 (Application)**.** An application $A = (V_A, S_A, K_{\text{enc}})$ consists of a validator set $V_A \subset V$, application-specific state S_A, and encryption key K_{enc} known only to authorized validators in V_A. Transactions submitted to A are encrypted under K_{enc}, preventing validators outside V_A from observing transaction contents.

Definition 2.2 (Application Commitment)**.** When application A finalizes block B at height h with state S_A, it produces a commitment $c = (h, H(B), \text{root}(S_A), \pi)$ where $H(B)$ is the block hash, $\text{root}(S_A)$ is the state root, and π is a zero-knowledge proof that B correctly transitions state from S_A^{h-1} to S_A^h. The commitment is submitted to the coordinator chain for global ordering.

Definition 2.3 (Commitment Status)**.** A commitment c can have two finality states on the coordinator:

- **Speculative:** The coordinator has ordered c in the global sequence but has not yet verified the zero-knowledge proof π. The commitment is provisionally accepted.
- **Cryptographic:** The coordinator has verified π and confirmed the state transition is correct. The commitment cannot be reversed.

Applications achieve speculative finality within seconds (via Simplex), while cryptographic finality requires proof verification which may take minutes. This separation enables low-latency user experience while maintaining cryptographic security.

2.1.1 Cross-Application Dependencies and Rollbacks

Cross-application transactions can reference commitments with speculative finality at the coordinator level, enabling low-latency composability. When application A_i executes a transaction that depends on state from application A_j (e.g., spending bUSD tokens minted in A_j), validators in A_i query the coordinator to check whether commitment c_j has been ordered. If c_j has achieved speculative finality at the coordinator, then A_i can proceed with the transaction immediately, without waiting for cryptographic finality. This allows cross-application composability within seconds rather than minutes.

The two consensus protocols have fundamentally different finality properties. Simplex provides deterministic finality: once validators achieve consensus on block B at height h, that decision is irreversible. All honest validators agree on the exact same block at height h, and no rollback can occur. Application subnets use Simplex because deterministic finality is essential for internal state transitions—when an application finalizes a state change, that change must be permanent within the application's consensus domain. In contrast, MonadBFT supports optimistic responsiveness with graceful rollback handling. The coordinator can speculatively

order commitments before achieving cryptographic finality, and if the ordering is incorrect (due to leader equivocation or proof verification failure), the coordinator can roll back to a previous state and re-order commitments. MonadBFT is designed specifically for this: it maintains execution snapshots at block boundaries and provides efficient rollback mechanisms when speculative execution fails.

Rollbacks occur when the coordinator leader equivocates. The coordinator leader proposes block B_C at height h that orders commitments from various applications. If the leader equivocates by sending different proposals $B_C^{(1)}$ and $B_C^{(2)}$ to different validators, some validators may vote for $B_C^{(1)}$ (which includes commitment c_j at position k) while others vote for $B_C^{(2)}$ (which includes c_j at position $k' \neq k$ or excludes it entirely). Neither proposal receives $2f + 1$ votes to form a quorum certificate. In the next view, a new leader proposes $B_C^{(3)}$ with a different ordering. Validators who observed $B_C^{(1)}$ speculatively must roll back their view of the global ordering and accept the new canonical ordering from $B_C^{(3)}$. Validators maintain state through MonadDB, a database of merkle trie diffs where speculative execution produces pointers to new state tries. When rollbacks occur due to equivocation, validators simply discard the pointer to the speculative state, undoing the execution without complex checkpoint restoration. The protocol detects equivocation through cryptographic proofs—validators who observe conflicting signed proposals from the same leader at the same height and view can prove misbehavior. The equivocating leader's stake is subject to slashing and they face punishment, providing economic deterrence. However, rollbacks are rare primarily due to MonadBFT's protocol design: the reproposal mechanism and No-Endorsement Certificates ensure that any block receiving a supermajority of votes will eventually finalize, meaning normal failures like offline leaders or network delays do not trigger rollbacks. Only leader equivocation—proposing two different blocks at the same height—can revert speculative execution, making rollbacks both structurally and economically exceptional events.

When application A_i executes a cross-application transaction based on speculative coordinator state, it observes that commitment c_j has been ordered at position k. Validators in A_i execute the transaction, finalize it via Simplex, and produce commitment c_i with a dependency on c_j. The commitment c_i includes zero-knowledge proof π_i that proves the state transition is valid given that c_j precedes it in the global ordering. When c_i is submitted to the coordinator, the coordinator validates the dependency: it checks whether c_j was actually finalized at a position preceding c_i in the canonical ordering. If the coordinator's view rolled back and c_j is now at position $k' \neq k$ or was excluded, the dependency is invalid. The coordinator rejects c_i and does not include it in the global sequence. The application's internal state remains finalized via Simplex—that finality cannot be reversed. However, this state branch does not enter the global history. Validators in A_i observe the rejection, produce a new commitment c_i' based on the correct coordinator state, and re-submit. This mechanism separates application-level finality (deterministic via Simplex) from global ordering finality (optimistic via MonadBFT), enabling fast composability while maintaining security.

2.2 Dual Settlement Architecture

Dual settlement operates through dual state management. The protocol maintains finalized state S_F and provisional state S_P for each account. Permissionless transactions update both states atomically—execution and settlement occur simultaneously. Qualified participant transactions update provisional state immediately but defer finalized state updates to epoch boundaries. During an epoch, participants trade on credit: transactions execute against S_P, accumulating obligations. At epoch close, the protocol computes multilateral net positions and updates S_F with net transfers only. This requires validators to track both state versions and enforce different

validation rules based on participant qualification status.

3 Yield-Bearing Asset Model

We introduce bUSD, a yield-bearing token backed 1:1 by BlackRock's BUIDL fund, which holds tokenized U.S. Treasury securities. bUSD serves as the protocol's universal numéraire: all markets quote prices in bUSD, participants post collateral in bUSD, and settlement occurs in bUSD. This section specifies the backing mechanism, yield distribution, and the economic feedback loop that creates structural demand for BUIDL.

Backing Mechanism. Each bUSD token represents a claim on one share of the BUIDL fund. BUIDL is an on-chain money market fund that holds U.S. Treasury bills, reverse repurchase agreements, and cash. The fund distributes yield through daily dividend payments, which accrue to BUIDL shareholders. bUSD holders automatically receive this yield: the protocol tracks BUIDL's net asset value (NAV) updates and credits proportional yield to all bUSD balances. Minting bUSD requires depositing BUIDL shares into a protocol-controlled vault; redeeming bUSD returns the underlying BUIDL shares plus accumulated yield. This maintains the 1:1 backing invariant: the vault always holds exactly as many BUIDL shares as outstanding bUSD tokens.

Universal Numéraire. bUSD's role as the universal numéraire creates structural demand for BUIDL. All trading pairs quote prices in bUSD (e.g., TSLA/bUSD, EUR/bUSD), making bUSD the base asset for price discovery. Collateral requirements for qualified participants are denominated in bUSD, requiring participants to hold bUSD balances proportional to their trading activity. Settlement occurs in bUSD—even when trades execute in other assets, net positions settle in bUSD. This creates a reflexive demand loop: as trading volume increases, participants require more bUSD for collateral and settlement. As more applications launch, they adopt bUSD as their pricing denomination. As liquidity concentrates in bUSD pairs, market makers hold bUSD inventory. Each additional use case increases demand for bUSD, which mechanically increases demand for BUIDL shares, channeling protocol growth into Treasury fund holdings.

4 Making Arbitrary Assets First-Class Citizens

Traditional blockchains privilege a single native token—users must acquire and hold this token to interact with the protocol. This creates artificial barriers: a participant holding TSLA shares, EUR stablecoins, or tokenized bonds must first convert to the chain's native token before transacting. We eliminate this barrier by making arbitrary assets first-class citizens for all protocol interactions. Users pay transaction fees, post collateral, and settle obligations in their preferred asset—whether tokenized treasuries, equities, or foreign currencies—and the protocol handles conversion transparently. A user paying gas in TSLA sees a small fraction deducted; a user posting collateral with a portfolio of stocks deposits those shares directly. The protocol converts to $SEN for validator compensation or bUSD for settlement as needed. This requires validators to resolve prices for hundreds of assets in real time without introducing latency. We achieve this through market maker quote streaming: market makers broadcast continuous price feeds, validators cache quotes locally, and asset swaps execute during transaction validation via constant-time lookups.

4.1 Quote Structure and Validity

A quote is a cryptographically signed commitment

$$q = (m, a, p_{\text{bid}}, p_{\text{ask}}, s_{\max}, t, \tau, \sigma_q),$$

where $m \in \mathcal{M}$ identifies the market maker, $a \in \mathcal{A}$ specifies the asset, $p_{\text{bid}}, p_{\text{ask}} \in \mathbb{R}^+$ give bid and ask prices in \$SEN per unit of a, $s_{\max} > 0$ bounds maximum size, t records the issuance timestamp, $\tau > 0$ sets time-to-live (constrained by $\tau \leq \tau_{\max} = 1$ second), and σ_q provides a digital signature. This represents a binding commitment: market maker m will buy or sell asset a at these prices for the next τ seconds, up to quantity $s_{\max}$.

Definition 4.1 (Quote Validity)**.** Quote q is valid at time t' if:

1. Signature verifies: $\text{Verify}(\sigma_q, (m, a, p_{\text{bid}}, p_{\text{ask}}, s_{\max}, t, \tau)) = \mathsf{true}$
2. Not expired: $t' \leq t + \tau$
3. Respects maximum freshness: $\tau \leq \tau_{\max}$
4. Price is reasonable: $|p_{\text{ask}} - P^a_{\text{ref}}(t')| \leq \epsilon \cdot P^a_{\text{ref}}(t')$ for tolerance $\epsilon = 0.05$
5. Sufficient collateral: $O_m(t') + p_{\text{ask}} \cdot s_{\max} \leq C_m$

where $P^a_{\text{ref}}(t')$ denotes the external reference price, $O_m(t')$ denotes outstanding obligations, and C_m denotes posted collateral.

Validators verify that quoted prices are reasonable relative to external market prices P^a_{ref} observed from exchanges. Market makers cannot profitably misreport prices: validators compute reference prices as the median over multiple market maker attestations, so individual misreporting is rejected by the median aggregation, while providing false attestations creates slashing risk without benefit. The protocol mandates $\tau_{\max} = 1$ second to establish a floor on service quality—market makers can update more frequently for competitive advantage but cannot commit for longer periods.

4.2 Block Construction

Market makers broadcast quotes at high frequency (100–500 Hz) via a dedicated gossip network. Each validator maintains a quote cache $\mathcal{Q}_v : \mathcal{A} \times \mathcal{M} \to \text{Quote} \cup \{\perp\}$ in memory, returning the most recent valid quote from market maker m for asset a. A background thread continuously updates this cache off the critical path of block construction.

When validator v constructs block B_n at time t_n, it processes each transaction tx in the mempool:

1. If tx pays in native asset \$SEN, execute normally.
2. Otherwise, for transaction paying in asset a:
   (a) Find valid quote with minimum ask: $q^* = \arg\min_{q \in \mathcal{Q}_v(a,\cdot)} p_{\text{ask}}$ subject to validity.
   (b) If no valid quote exists, reject transaction.
   (c) Calculate required amount: $s_a = \text{gas cost}/p_{\text{ask}}$.
   (d) Record obligation: $o = (tx\_id, m, a, s_a, \text{gas cost}, p_{\text{ask}}, t_n)$.
   (e) Transfer s_a units of asset a from sender to protocol escrow.
   (f) Execute transaction.

The per-transaction overhead is $O(|\mathcal{M}|)$ for quote lookup, with constant factor approximately 100 nanoseconds per market maker. With 10 market makers and 5,000 transactions per block, total overhead is roughly 5 milliseconds.

4.3 Deferred Settlement and Capital Efficiency

Settlement occurs asynchronously over a window of K blocks (typically $K = 50$, giving 20 seconds). When a user pays gas in asset a, the user's tokens are held in escrow, but the market maker has K blocks to deliver the equivalent \$SEN to validators. This deferral enables capital efficiency through netting.

Consider a market maker handling many transactions over 20 seconds. If they receive 100 TSLA from users paying gas (owing 100k \$SEN to validators) while simultaneously receiving 50k \$SEN from other users, the net obligation is only 50k \$SEN. Without deferred settlement, required capital is $C_{\text{atomic}} = N \cdot T_{\text{settle}} \cdot \bar{c}$, where N is transactions per second and $\bar{c}$ is average capital per transaction. With deferred settlement over window W and flow that is fraction θ in one direction, required capital is $C_{\text{deferred}} = N \cdot W \cdot |2\theta - 1| \cdot \bar{c}$. For balanced flow ($\theta = 0.5$), obligations net to nearly zero. For realistic flow ($\theta = 0.6$) with $W = 20$ seconds, this yields 2.5× capital efficiency improvement.

Market makers must settle by block B_{n+K}, providing proof of transfer to the protocol's validator reward pool. Validators prioritize market maker settlement transactions during block construction to ensure market makers can settle promptly and continue providing competitive quotes. A market maker who cannot settle on time must stop broadcasting quotes, reducing competition and widening spreads for users. By prioritizing settlement, validators ensure market makers maintain adequate capital to quote aggressively. In block B_{n+K+1}, the protocol automatically enforces settlement: if no valid settlement exists, the protocol slashes the market maker's collateral, bans them permanently, and invalidates all cached quotes. Settlement is deterministic and requires no governance.

4.4 Eliminating Adverse Selection Through Discrete Time

A key economic contribution of this design is the elimination of adverse selection that plagues continuous markets. As documented by Aquilina, Budish, and O'Neill [2], continuous markets create an "adverse selection tax" where market makers lose money to arbitrageurs in microsecond races. When external prices move, market makers' standing quotes become stale. Both parties race to either cancel or execute against stale quotes within 50–100 microseconds, with the winner capturing the spread and the loser bearing the cost. Market makers respond by widening spreads by 17–33% to compensate for inevitable losses.

Our protocol eliminates this through discrete-time batch processing. All transactions in block B_n execute at time t_n seeing identical cached quotes that were committed before the block began. Market makers cannot selectively withdraw quotes, adjust prices, or refuse trades mid-block based on price movements. The quotes are cryptographically signed commitments that bind for the entire block window. There is no way for arbitrageurs to selectively pick off stale prices because everyone uses the same quotes simultaneously. If external prices move during the block window, either everyone benefits or no one does.

This discrete-time matching at 400ms intervals eliminates the timing game entirely. Market makers can commit to prices for short windows without fear of adverse selection, allowing them to quote tighter spreads—yielding expected spread reductions of 17–33% relative to continuous markets.

5 Dual Settlement with Qualified Participant Infrastructure

Blockchains traditionally treat all participants identically: everyone accesses the same assets, executes under the same rules, and settles transactions atomically. This uniformity conflicts

with financial regulation, which distinguishes between qualified and unqualified participants based on identity, jurisdiction, accreditation status, and compliance requirements. A blockchain supporting institutional finance must accommodate both regulatory tiers without sacrificing permissionless access. We achieve this through a dual identity and resource system. Permissionless participants transact without identity verification, accessing public resources with atomic settlement and no credit extension. Qualified participants meet application-specific requirements—KYC verification, accredited investor status, geographic eligibility, or other compliance criteria—and gain access to qualified resources: epoch-based settlement with multilateral netting, credit extension during trading epochs, leverage through margin, and regulated asset classes requiring qualified counterparties.

The protocol maintains dual state for each account—finalized state S_F and provisional state S_P—enabling qualified participants to trade on credit while tracking obligations. This section outlines how this qualification mechanism operates, how dual state management works, how multilateral netting reduces capital requirements, and the resulting capital efficiency gains.

5.1 Qualification via Composable Hooks

Participants become qualified by obtaining credential attestations from approved verifiers and registering them on-chain. The protocol enforces qualification through composable hooks that validators execute during transaction validation. A hook is a deterministic function that accepts or rejects a transaction based on sender credentials, returning true or false. Applications register required hooks when deploying: an equity trading venue might require [`geographic_restriction`, `kyc_verification`, `non_blacklist`] while a derivatives platform requires [`kyc_verification`, `accredited_investor`, `position_limit`]. During transaction validation, validators execute registered hooks sequentially, rejecting the transaction if any hook returns false. This occurs before transaction execution, making bypass impossible at the application layer.

Credential verification relies on a competitive market of off-chain verifiers—entities like Coinbase, Circle, or traditional KYC providers that verify real-world identity. A user completes verification with an approved verifier, who signs an attestation $\sigma = \text{Sign}_v(\text{address}, \text{credential\_type}, \text{metadata}, \text{expiratio}$ committing to the user's credentials. The user submits this attestation on-chain to a protocol-level registry, which stores only the commitment hash for privacy. When validators execute a hook, they verify the attestation signature and check validity. Multiple verifiers compete to provide verification services—applications can require attestations from N-of-M verifiers for critical checks, and verifiers face slashing if caught providing false attestations. This creates a trust-minimized identity system: no single verifier controls access, credentials are reusable across applications, and new credential types can be added without protocol changes.

Definition 5.1 (Qualified Participant)**.** A participant p is qualified for application A if p possesses valid credential attestations satisfying all hooks $H_1, H_2, \ldots, H_k$ registered by A. Qualified participants have access to protocol-level qualified resources enforced by validators: epoch-based settlement with multilateral netting, dual state management (S_F and S_P), and credit extension during epochs. Additionally, applications may gate access to application-level resources (regulated asset classes, specific trading venues) based on qualification status. Permissionless participants cannot access protocol-level qualified resources regardless of application-level permissions.

Define the access control map $R : \mathcal{P} \times \text{Resource} \to \{\text{Allow}, \text{Deny}\}$ where $R(p, r) = \text{Allow}$ if and only if $p \in P_1$ for protocol-level qualified resources $r \in \{\text{EpochSettlement}, \text{DualState}, \text{Credit}\}$.

Let P_0 denote permissionless participants and P_1 denote qualified participants. These sets partition the participant space: $P_0 \cap P_1 = \emptyset$ for a given application. A participant may be qualified for one application but not another, so qualification status is application-specific.

Validators enforce resource access control at the protocol level: transactions from P_0 attempting epoch-based settlement, accessing provisional state S_P, or requesting credit are rejected during validation before application logic executes.

The protocol guarantees that permissionless participants cannot access qualified resources. During transaction validation, validators check the sender's qualification status before executing state transitions. Any transaction from $p \in P_0$ that attempts to update provisional state S_P, request credit, or invoke epoch-based settlement is rejected before reaching application logic. This enforcement occurs deterministically: all validators verify the same credential attestations and execute the same hooks, ensuring consistent access control across the network.

5.2 Dual State Management

The protocol maintains two state versions for each account: finalized state S_F and provisional state S_P. Finalized state reflects settled balances—funds that have completed settlement and cannot be reversed. Provisional state reflects pending obligations—balances including intra-epoch trades that have executed but not yet settled. For permissionless participants, these states are always identical: $S_F(p) = S_P(p)$ for all $p \in P_0$, as their transactions settle atomically. For qualified participants, provisional state may exceed finalized state during an epoch: $S_P(p) \geq S_F(p)$ for $p \in P_1$, representing credit extended until epoch settlement.

When a qualified participant executes a transaction, validators update provisional state immediately but defer finalized state updates to the epoch boundary. Consider participant p with $S_F(p) = 1000$ bUSD who purchases 500 bUSD of TSLA shares during an epoch. The transaction updates $S_P(p) \leftarrow S_P(p) - 500 = 500$ bUSD immediately, allowing p to continue trading. However, $S_F(p)$ remains 1000 bUSD until epoch close. The difference $S_P(p) - S_F(p) = -500$ represents p's net obligation to the clearinghouse. At epoch close, the protocol computes multilateral net positions across all qualified participants and updates S_F to match S_P after netting.

5.3 Multilateral Netting

At the end of each epoch, the protocol computes net settlement obligations across all qualified participants, dramatically reducing the number of transfers required compared to gross settlement. During an epoch of duration T_{epoch} (typically 24 hours), each qualified participant $p \in P_1$ executes multiple transactions with various counterparties. Under gross settlement, every individual transaction would require an immediate transfer of funds. Under multilateral netting, the protocol accumulates all obligations during the epoch and settles only net positions at epoch close.

The netting computation proceeds as follows. For each participant $p \in P_1$, the protocol computes the net position $\Delta_p = S_P(p) - S_F(p)$, representing the cumulative change in bUSD balance from all intra-epoch transactions with other qualified participants. A positive net position $\Delta_p > 0$ indicates that p received more than they paid—they are owed funds by the clearinghouse. A negative net position $\Delta_p < 0$ indicates that p paid more than they received—they owe funds to the clearinghouse. By construction, net positions sum to zero across all participants: $\sum_{p \in P_1} \Delta_p = 0$, as every transfer within the epoch netting system has both a sender and receiver in the qualified participant set. Trades between qualified and permissionless participants settle atomically and are excluded from epoch netting.

Settlement executes in a single atomic operation. Validators compute the set of participants with positive net positions $P^+ = \{p \in P_1 : \Delta_p > 0\}$ and negative net positions $P^- = \{p \in P_1 : \Delta_p < 0\}$. Participants in P^- transfer their net obligations to a protocol-controlled settlement pool: for each $p \in P^-$, the protocol executes transfer$(p, \text{pool}, |\Delta_p|)$. The settlement

pool then distributes funds to participants in P^+: for each $p \in P^+$, the protocol executes transfer$(\text{pool}, p, \Delta_p)$. After settlement completes, finalized state matches provisional state for all participants: $S_F(p) = S_P(p)$ for all $p \in P_1$. The next epoch begins with all participants starting from their settled balances.

Capital Efficiency Gains. The reduction in settlement volume follows from statistical properties of bilateral flows. Consider n qualified participants who trade uniformly with each other during an epoch. Each participant executes k transactions with random counterparties, transferring value v per transaction. Under gross settlement, total settlement volume is $V_{\text{gross}} = n \cdot k \cdot v$, as every transaction settles individually. Under multilateral netting, each participant's net position is the sum of k random bilateral flows. By the Central Limit Theorem, net positions concentrate around zero with standard deviation $\sigma = v\sqrt{k}$. Total settlement volume is approximately $V_{\text{net}} = n \cdot \sigma = n \cdot v\sqrt{k}$. The ratio of net to gross volume is:

$$\frac{V_{\text{net}}}{V_{\text{gross}}} = \frac{n \cdot v\sqrt{k}}{n \cdot k \cdot v} = \frac{1}{\sqrt{k}}. \tag{1}$$

For participants executing $k = 100$ transactions per epoch, multilateral netting reduces settlement volume by 90%. For $k = 1000$ transactions, the reduction reaches 97%. This capital efficiency gain is fundamental: as trading activity increases (larger k), the benefit of netting increases proportionally.

Settlement and Credit. Settlement proceeds deterministically at epoch close. Participants with negative net positions transfer their obligations to the settlement pool, which distributes to participants with positive net positions. If a participant cannot meet their settlement obligation, their broker—who extended credit enabling the trades—covers the shortfall. This aligns with traditional prime brokerage: brokers extend credit, manage risk, and bear responsibility when clients default. The protocol's role is computing net positions and executing transfers; credit risk management remains with brokers as described in Section 7.

6 Privacy and Isolation

Financial applications require confidentiality: validators processing equity trades should not observe derivatives transactions, and market makers on one venue should not see competitor order flow. The two-layer consensus architecture described in Section 2 provides application-level privacy through validator partitioning and zero-knowledge proof verification. This section demonstrates how the architecture achieves privacy guarantees while maintaining cryptographic security.

6.1 Application Isolation

Each application operates within its own validator subnet with access to an encryption key K_{enc} known only to authorized validators. When a user submits transaction tx to application A, the client encrypts it under K_{enc} before broadcast. Validators in V_A decrypt the transaction, validate it against application state S_A, and execute state transitions. Validators outside V_A observe only encrypted ciphertext—they cannot decrypt transaction contents or infer state transitions. This provides confidentiality at the network layer: application data never leaves the validator subnet in plaintext.

The coordinator receives only cryptographic commitments from applications: a block hash $H(B)$, state root root(S_A), and zero-knowledge proof π. The coordinator cannot decrypt application transactions or inspect application state. The zero-knowledge proof establishes that the state transition is valid without revealing transaction contents. The coordinator verifies π using the application's public verification key, confirming correctness while preserving confidentiality. This separation ensures that even coordinator validators—who order commitments globally—learn nothing about application-specific activity beyond timing and state roots.

6.2 Security Properties

The architecture provides two privacy guarantees. First, *transaction confidentiality*: validators outside V_A cannot learn transaction contents for application A. This follows from semantic security of the encryption scheme—ciphertexts are computationally indistinguishable from random. Second, *state integrity with privacy*: the coordinator can verify that state transitions are correct without learning transaction details. This follows from zero-knowledge proof soundness—if the proof verifies, then the state transition is valid with overwhelming probability, but the proof reveals nothing beyond this fact.

Byzantine applications cannot commit to invalid state without detection. If a byzantine majority in V_A attempts to finalize an invalid block, they can produce a commitment with $2f + 1$ signatures, achieving speculative finality at the coordinator. However, they cannot generate a valid zero-knowledge proof π for invalid state transitions—proof soundness prevents this. When the coordinator attempts to verify π, verification fails. The coordinator marks the commitment as rejected and slashes the validators in V_A who signed invalid state. This mechanism ensures that privacy does not compromise security: applications maintain confidentiality while the coordinator enforces correctness cryptographically.

7 Bilateral Credit System

Traditional finance operates on credit: prime brokers extend margin to institutional clients, allowing traders to take positions larger than their capital. Blockchains traditionally require full collateralization for every position, eliminating the capital efficiency that credit provides. We enable bilateral credit relationships between brokers and qualified participants while maintaining on-chain transparency of obligations. Only qualified participants $p \in P_1$ can access credit facilities—permissionless participants $p \in P_0$ cannot borrow or lend through the protocol. This restriction ensures that credit relationships involve identified counterparties who can be held accountable off-chain if obligations are not met. Brokers assess creditworthiness off-chain using traditional risk management—client track record, portfolio composition, external guarantees—and extend credit through on-chain loan transactions. The protocol tracks these loans as liabilities, enabling qualified participants to trade with borrowed capital while maintaining a transparent record of all obligations.

7.1 Credit Mechanics

A broker $b \in P_1$ extends credit to participant $p \in P_1$ by executing a loan transaction specifying the principal amount $L \in \mathbb{R}^+$, interest rate $r \in \mathbb{R}^+$, and maturity timestamp t_{maturity}. The transaction updates the participant's provisional state: $S_P(p) \leftarrow S_P(p) + L$ in bUSD, while finalized state $S_F(p)$ remains unchanged until the loan is repaid and settled. The protocol records the loan obligation in state, tracking the relationship between broker and participant. The participant can now trade using this borrowed capital—purchasing assets, posting collateral,

or paying transaction fees—within the epoch. The protocol tracks the outstanding principal and accrued interest, computing $\text{owed}(p, b, t) = L \cdot (1 + r)^{(t-t_0)/T}$ where t_0 is the loan origination time and T is the compounding period.

Repayment occurs through a repayment transaction where participant p transfers funds to broker b. The protocol verifies that the repayment amount covers at minimum the accrued interest, updating the outstanding principal accordingly. Partial repayments reduce the principal: $L \leftarrow L - \text{payment}$, while full repayment closes the loan. If the participant fails to repay by maturity, the loan enters default. The protocol does not automatically liquidate positions or seize collateral—default resolution is handled off-chain between the broker and participant according to their bilateral agreement. The broker may pursue legal remedies, liquidate collateral held outside the protocol, or negotiate restructured terms. The on-chain record remains transparent: all participants observe which accounts are in default, allowing brokers to make informed credit decisions.

7.2 Credit Assessment and Risk

Credit decisions occur off-chain. A broker evaluates a participant's creditworthiness using traditional methods: reviewing trading history, assessing portfolio risk, requiring external guarantees or letters of credit, or imposing covenant restrictions on how borrowed capital can be used. The protocol provides brokers with on-chain transparency into participant activity—current positions, outstanding loans with other brokers, historical default events—but does not enforce risk limits or margin requirements. Each broker sets their own credit terms, creating a competitive market for credit provision. Participants with strong track records access credit at favorable rates from multiple brokers, while participants with poor credit history face higher rates or require additional off-chain collateral.

This design mirrors traditional prime brokerage: credit relationships are bilateral, risk management is the broker's responsibility, and defaults are resolved privately. The protocol's role is limited to providing transparent record-keeping and enabling on-chain settlement of credit-based trades. This approach avoids the complexity and brittleness of on-chain risk engines while preserving the capital efficiency that institutional traders require. Brokers compete on credit terms, participants choose brokers based on rates and service quality, and the protocol facilitates these relationships without imposing rigid collateralization requirements.

7.3 Arbitrary Asset Collateral and Portfolio Margining

Because the protocol treats arbitrary assets as first-class citizens (Section 4), participants can post any tokenized asset as collateral for credit lines. A participant's collateral portfolio might include U.S. Treasury tokens, equity shares, EUR stablecoins, tokenized corporate bonds, or baskets of derivatives positions. The broker evaluates collateral value using real-time market prices from the quote streaming system: for each asset a in the portfolio, the broker queries current bid prices p^a_{bid} from market makers and computes total portfolio value $V = \sum_{a \in \text{portfolio}} q_a \cdot p^a_{\text{bid}}$ where q_a denotes quantity held. This enables credit decisions based on total portfolio value rather than requiring collateral in a single designated asset.

Portfolio margining emerges naturally from this design. Traditional DeFi protocols like Aave require isolated collateral per position: borrowing against TSLA requires posting TSLA-specific collateral, borrowing against EUR requires separate EUR collateral, and correlations between positions are ignored. This multiplies capital requirements—a portfolio with offsetting positions still requires full collateral for each leg. In contrast, brokers extending credit on Sentry observe the participant's entire portfolio on-chain: long TSLA and short SPY, long EUR and short GBP,

long 10-year treasuries and short 2-year treasuries. The broker computes portfolio-level risk accounting for correlations, hedges, and offsetting positions. A participant with a market-neutral equity strategy requires less collateral than one with directional exposure, even if the notional position sizes are identical. This portfolio margining significantly reduces capital requirements compared to isolated position collateral.

The protocol provides the infrastructure for portfolio margining without enforcing specific margin rules. Brokers access the participant's complete on-chain position data: current holdings across all assets, active trades in provisional state S_P, outstanding obligations to other brokers, and historical profit-and-loss. The broker applies their own risk models—computing Value-at-Risk, stress testing against historical scenarios, or using proprietary correlation matrices—to determine appropriate credit limits. When market prices move, brokers can request additional collateral off-chain or reduce credit lines by adjusting loan terms for future extensions. If a participant's portfolio value drops below the broker's internal risk threshold, the broker may issue a margin call through external communication channels. The protocol does not automate these decisions: brokers manage risk using traditional methods, with the advantage of complete transparency into on-chain positions.

This combination—arbitrary asset collateral plus broker-managed risk assessment—replicates the flexibility of traditional prime brokerage while maintaining blockchain transparency. Participants gain capital efficiency through portfolio margining without sacrificing the permissionless access that blockchains provide. Brokers differentiate on their ability to model risk accurately: sophisticated risk management allows tighter credit terms, attracting clients who value capital efficiency. The protocol's role is enabling this market by making all assets equally accessible as collateral and providing brokers with complete visibility into participant positions.

8 Security Analysis

This section establishes the security properties of the protocol under adversarial conditions. We specify the threat model, prove settlement safety guarantees for both permissionless and qualified participants, analyze consensus security under the two-layer architecture, and demonstrate that privacy mechanisms do not compromise integrity.

8.1 Threat Model

We assume a Byzantine adversary model for validators: up to $f < n/3$ validators may behave arbitrarily—equivocating, withholding messages, proposing invalid blocks, or colluding to disrupt consensus. This bound is necessary and sufficient for BFT consensus [5]: fewer than $n/3$ byzantine validators cannot prevent honest validators from reaching agreement. We assume partial synchrony for network conditions [7]: messages between honest validators are delivered within bounded delay Δ after some unknown Global Stabilization Time (GST). Before GST, the network may experience arbitrary delays, and after GST, it behaves synchronously with known bound Δ. Both Simplex and MonadBFT maintain safety and liveness under these assumptions.

Market makers are assumed rational: they maximize profit and may collude to manipulate prices or extract value from users. We do not assume market makers are honest—they will deviate from protocol if profitable. The quote streaming mechanism must remain secure even when market makers coordinate on price manipulation or attempt to selectively execute against informed order flow. Brokers extending credit are similarly rational: they assess risk to maximize returns while minimizing defaults. The credit system must not create systemic vulnerabilities when brokers fail or collude.

8.2 Settlement Safety

The protocol provides different safety guarantees for permissionless and qualified participants based on their settlement model. Permissionless participants receive standard blockchain safety: once a transaction achieves finality, the state transition is irreversible and no double-spend is possible. This follows from the safety properties of Simplex and MonadBFT: under $f < n/3$ byzantine validators, consensus ensures all honest validators agree on the same committed blocks, preventing conflicting transactions from finalizing. For permissionless participants, finalized state S_F is updated atomically with each transaction, so settlement is immediate and irreversible.

Qualified participants using epoch-based settlement receive provisional integrity during epochs and cryptographic finality at epoch boundaries. During an epoch, provisional state S_P updates immediately as transactions execute. As long as an honest majority of validators agree on provisional state transitions, S_P remains consistent across the network. However, provisional state is not cryptographically final—it can be rolled back if the coordinator experiences equivocation or if zero-knowledge proofs fail verification. At epoch close, the protocol computes net positions and updates finalized state S_F. Once the coordinator verifies the zero-knowledge proof for the netting computation and the commitment achieves cryptographic finality, the settlement is irreversible. This two-phase approach enables capital-efficient trading during epochs while maintaining the same ultimate finality guarantee as permissionless settlement.

The protocol enforces isolation between settlement classes: attacks on qualified participant settlement cannot affect permissionless participants. Permissionless state S_F for $p \in P_0$ is independent of provisional state S_P—even if byzantine validators manipulate provisional state during an epoch, permissionless balances remain unaffected. Validators enforce this isolation deterministically: transactions from P_0 cannot reference or depend on S_P, and rollbacks of provisional state do not trigger rollbacks of permissionless finalized state. This ensures that users who require immediate finality can opt out of epoch-based settlement without exposure to qualified participant risks.

8.3 Consensus Security

The two-layer consensus architecture provides safety and liveness under partial synchrony with $f < n/3$ byzantine validators. At the application layer, Simplex guarantees that honest validators finalize the same blocks at each height: if honest validator v_1 finalizes block B at height h, then every other honest validator finalizes B at height h. This safety property holds unconditionally—even if the network is asynchronous forever, no two honest validators finalize conflicting blocks. Liveness requires partial synchrony: after GST, honest validators finalize new blocks within 3Δ rounds. Applications maintain deterministic finality, making rollbacks impossible within the application consensus domain.

At the coordinator layer, MonadBFT provides safety for commitment ordering: once a commitment c achieves cryptographic finality, its position in the global sequence is irreversible. Before cryptographic finality, commitments may experience speculative rollbacks due to leader equivocation. However, these rollbacks do not violate safety because speculative finality is explicitly provisional. The coordinator detects equivocation through cryptographic proofs and slashes byzantine leaders, making rational leaders avoid equivocation. After $2f + 1$ validators confirm a commitment and its zero-knowledge proof verifies successfully, the commitment achieves cryptographic finality and cannot be reversed. Cross-application dependencies (Section 2.1.1) maintain safety by rejecting commitments with invalid dependencies: if application A_i references a rolled-back commitment from A_j, the coordinator rejects A_i's commitment, preventing inconsistencies in the global ordering.

8.4 Privacy and Integrity

The privacy architecture maintains confidentiality while enabling cryptographic verification of correctness. Application transactions are encrypted under key K_{enc} known only to validators in V_A. Semantic security of the encryption scheme ensures that ciphertexts leak no information about plaintext contents to validators outside V_A. The coordinator receives only commitments $(h, H(B), \text{root}(S_A), \pi)$ without access to transaction details. Zero-knowledge proof soundness guarantees that if verification succeeds, the state transition is valid with overwhelming probability, but the proof reveals nothing beyond this fact. This separation provides application-level privacy: different applications cannot observe each other's transaction contents, and even the coordinator learns only state roots and timing.

Byzantine applications cannot commit to invalid state without detection. Suppose a byzantine majority in V_A attempts to finalize an invalid block B with incorrect state transition $S^{h-1} \not\to S^h$. They can produce commitment $c = (h, H(B), \text{root}(S^h), \pi)$ with $2f + 1$ signatures, causing the coordinator to order c with speculative finality. However, they cannot generate a valid zero-knowledge proof π for the invalid transition—proof soundness prevents this. When the coordinator verifies π, verification fails. The coordinator marks c as rejected, excludes it from cryptographic finality, and slashes the validators who signed invalid state. Slashing penalties exceed potential gains from committing invalid state, ensuring rational validators in V_A maintain correctness even when privacy hides transaction details from external observers.

9 Multidimensional Fee Market

Transaction fees in traditional blockchains price a single resource—block space or gas—treating all transactions as homogeneous consumers of this resource. This conflates fundamentally different costs: computational execution, state storage, systemic settlement risk, and collateral utilization. When these distinct resources are priced together, the relative prices are fixed by the protocol designer rather than discovered through market mechanisms. This creates inefficiency: a transaction consuming primarily execution capacity pays the same marginal rate per unit as one consuming primarily collateral capacity, even though the costs imposed on the network differ. We implement a multidimensional fee market where different resources are priced independently, allowing market forces to discover the correct relative prices.

9.1 Resource Dimensions

Transactions consume three distinct resources, each with different scarcity characteristics and costs to the network. First, *execution capacity* measures computational work and state access required to validate and execute a transaction. This includes cryptographic operations, state reads and writes, and application-specific logic. Execution capacity is bounded per application by validator computational limits and per block by coordination overhead. We measure execution capacity consumption as $g_E \in \mathbb{R}^+$, denominated in gas units analogous to Ethereum's execution gas.

Second, *netting capacity* measures a qualified participant's contribution to systemic settlement risk during an epoch. When participant $p \in P_1$ executes a transaction that increases their net obligation—for example, purchasing assets with borrowed capital—they utilize the epoch settlement system's capacity to defer settlement. The netting capacity consumed by a transaction is the absolute change in net position: $g_N = |v|$ where v is the value transferred (positive for purchases, negative for sales). This resource is only consumed by qualified participants using

epoch-based settlement; permissionless participants with atomic settlement impose no netting risk.

Third, *collateral capacity* measures the opportunity cost of locking bUSD as collateral or provisional balances during trading. Qualified participants maintain collateral to cover potential obligations, and this collateral cannot be redeemed for the underlying BUIDL shares until settlement completes. The opportunity cost is the foregone yield during the lock period. Collateral capacity consumption is $g_C = v \cdot \tau$ where v is the value locked and τ is the expected lock duration in days. Like netting capacity, this resource applies only to qualified participants; permissionless transactions settle atomically and incur no collateral lock cost.

9.2 Fee Structure

The total fee for any transaction is the sum of a protocol-determined base fee and a user-determined priority fee. This separation, inspired by EIP-1559 [4], aligns incentives among users, validators, and the protocol. The base fee is computed algorithmically as the cost of consuming network resources and imposing systemic risk. Its revenue is not paid to validators but is instead burned or recycled through protocol mechanisms, preventing validators from manipulating base fee levels through artificial demand. The priority fee is a voluntary tip paid directly to the validator who includes the transaction. It functions as a first-price auction for block inclusion priority, allowing users to signal urgency while creating direct financial incentives for validators to include high-value transactions.

For a transaction consuming resources $\mathbf{g} = (g_E, g_N, g_C)$ submitted by sender s with qualification class $C(s) \in \{0, 1\}$, the base fee at block t is:

$$\text{BaseFee}(\mathbf{g}, s) = (1 + \rho^t)\left[\lambda_E^t \, g_E + \pi_{C(s)}^t \, g_N + \mu_{C(s)}^t \, g_C\right], \tag{2}$$

where λ_E^t is the execution price, $\pi_{C(s)}^t$ is the netting risk price, $\mu_{C(s)}^t$ is the collateral capacity price, and ρ^t is the gas opportunity cost multiplier. For permissionless participants ($C(s) = 0$), the protocol enforces $\pi_0^t \equiv 0$ and $\mu_0^t \equiv 0$ since they operate on a fully atomic settlement basis and impose no netting or collateral risks. Their base fee reduces to pure execution cost: $\text{BaseFee}(\mathbf{g}, s) = (1 + \rho^t)\lambda_E^t \, g_E$. Qualified participants ($C(s) = 1$) pay for all three resources, internalizing the externalities their credit-based trading imposes on the system.

The gas opportunity cost multiplier $(1 + \rho^t)$ accounts for the fact that bUSD is a yield-bearing asset commonly used to pay transaction fees. While \$SEN is the actual gas token paid to validators, users frequently pay fees in bUSD which market makers swap to \$SEN. When users spend bUSD for fees, they forgo continuous treasury yield $r \approx 0.05$ (5% annually from BUIDL) on that capital. This opportunity cost should be reflected in gas pricing. We model this as $\rho^t = r \cdot \tau_{\text{eff}}$ where τ_{eff} is the effective duration of yield loss. The current implementation sets $\rho^t = 0.05$, adding 5% to all base fees to compensate for foregone annual yield when paying fees in yield-bearing assets like bUSD.

9.3 Dynamic Price Adjustment

The vector of base prices $\mathbf{p}^t = (\lambda_E^t, \pi_1^t, \mu_1^t)$ adjusts dynamically to balance supply and demand for each resource. The update mechanism, based on online gradient descent applied to the dual of a welfare maximization problem [1], ensures that prices converge to market-clearing levels even under adversarial transaction submission patterns. This class of algorithms achieves average regret bounded by $O(1/\sqrt{T})$ over time horizon T, meaning the welfare achieved by dynamic pricing approaches the welfare achieved by an oracle with perfect knowledge of all future transactions.

At the end of each block t, the protocol observes utilization u_r^t for each resource $r \in \{E, N, C\}$, defined as the ratio of consumed capacity to target capacity. The target capacity b_r^* for each resource represents the protocol's desired utilization level, typically set to 50%–75% of maximum capacity to provide headroom for demand spikes. The raw utilization is smoothed using an exponentially moving average with decay parameter $\alpha \in (0, 1)$:

$$\hat{u}_r^t = \alpha \cdot u_r^t + (1 - \alpha) \cdot \hat{u}_r^{t-1}. \tag{3}$$

This prevents short-term manipulation by any single validator and provides predictable price trajectories for users. The smoothing parameter α is typically set to 0.125, giving half-weight to utilization over approximately the last 8 blocks.

The price for resource r updates multiplicatively based on the smoothed utilization:

$$p_r^{t+1} = p_r^t \cdot \exp[\eta_r \cdot (\hat{u}_r^t - u_r^*)], \tag{4}$$

where $\eta_r > 0$ is the learning rate and u_r^* is the target utilization. If smoothed utilization exceeds the target ($\hat{u}_r^t > u_r^*$), the price increases; if utilization is below target, the price decreases. The multiplicative update ensures prices remain positive and provides percentage-based adjustments that scale appropriately across price levels. To prevent extreme volatility from adversarial behavior or sudden demand shocks, the protocol clamps per-block price changes to a maximum factor $M \in (0, 1)$:

$$p_r^{t+1} = \min\big(\max\big(p_r^{t+1},\ p_r^t(1 - M)\big),\ p_r^t(1 + M)\big). \tag{5}$$

Typical values are $M = 0.125$ (12.5% maximum change per block), which limits price manipulation while allowing responsive adjustment to demand shifts. This update rule is mathematically equivalent to EIP-1559's price mechanism when applied to a single resource, extended here to multiple dimensions with independent price discovery for each.

9.4 Incentive Properties

The two-part fee structure ensures strategy-proofness for users and incentive compatibility for validators. Users face a dominant strategy of submitting transactions with priority fees equal to their true valuation for urgency: the base fee is exogenous to any individual transaction, so users cannot manipulate it, and the priority fee is a standard first-price auction where truthful bidding is optimal given sufficient competition. Validators maximize revenue by including transactions with the highest priority fees subject to resource constraints: fabricating fake transactions requires paying the base fee (which validators do not receive) while forgoing legitimate priority fees from real users. This makes honest inclusion optimal.

The mechanism is also resistant to off-chain agreements between users and validators. Since validators do not receive the base fee, they cannot credibly offer rebates on it—any such agreement would require the validator to pay the user from their own priority fee revenue, making it unprofitable. Priority fee rebates are equivalent to the user simply posting a lower on-chain priority fee, providing no advantage. Price manipulation through artificial demand is costly: an adversary must pay base fees on all fabricated transactions, and the EMA smoothing dampens the effect of any single block's utilization. The multidimensional structure further increases manipulation costs: successfully inflating the price of one resource requires saturating demand for that specific resource while others remain available, creating arbitrage opportunities for users who substitute toward cheaper resources.

10 Related Work

To be completed in a future revision.

11 Conclusion

To be completed in a future revision.

References

[1] Guillermo Angeris, Theo Diamandis, Kshitij Zhang, and Tarun Chitra. Multidimensional gas pricing. *arXiv preprint arXiv:2402.14172*, 2024.

[2] Matteo Aquilina, Eric Budish, and Peter O'Neill. Quantifying the high-frequency trading "arms race". *The Quarterly Journal of Economics*, 137(1):493–564, 2022.

[3] BlackRock. BlackRock USD Institutional Digital Liquidity Fund. https://www.blackrock.com/us/financial-professionals/investment-strategies/buidl, 2024. Accessed: 2024.

[4] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change for ETH 1.0 chain. *Ethereum Improvement Proposals*, 2019. https://eips.ethereum.org/EIPS/eip-1559.

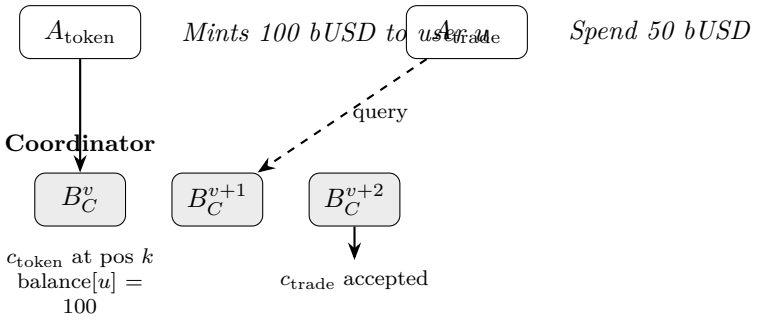
[5] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[6] Depository Trust & Clearing Corporation. DTCC Annual Report. https://www.dtcc.com/annuals/2023/financial-performance, 2023. Accessed: 2024.

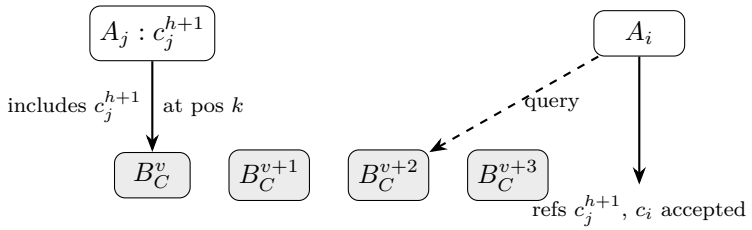
[7] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[8] Monad Labs. MonadBFT: Optimistic BFT with Deferred Execution. Technical Documentation, 2024. Optimistic consensus protocol with rollback support.

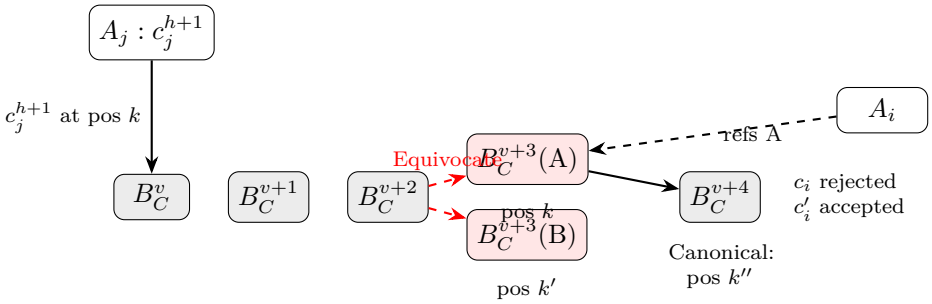
[9] Monad Labs. Simplex Consensus Protocol. Technical Documentation, 2024. Deterministic BFT consensus protocol.



(a) Shared state dependency: Token application A_{token} mints 100 bUSD to user u, creating commitment c_{token} that updates the global balance. Trading application A_{trade} queries the coordinator for u's balance from c_{token}, verifies sufficient funds, and creates a transaction spending 50 bUSD.



(b) Normal cross-app ordering: Application A_j produces commitment c_j^{h+1}. The coordinator orders it at position k. Application A_i queries the coordinator, observes the ordering, and creates commitment c_i that is accepted.



(c) Equivocation causes rollback: The coordinator leader equivocates with proposals A and B ordering c_j^{h+1} at different positions. Application A_i observes proposal A and creates c_i based on it. When validators reach consensus on the canonical ordering (position k''), the dependency is invalid and c_i is rejected. Application A_i re-submits corrected commitment c_i' which is accepted.

Figure 1: Cross-application dependencies and rollback handling.

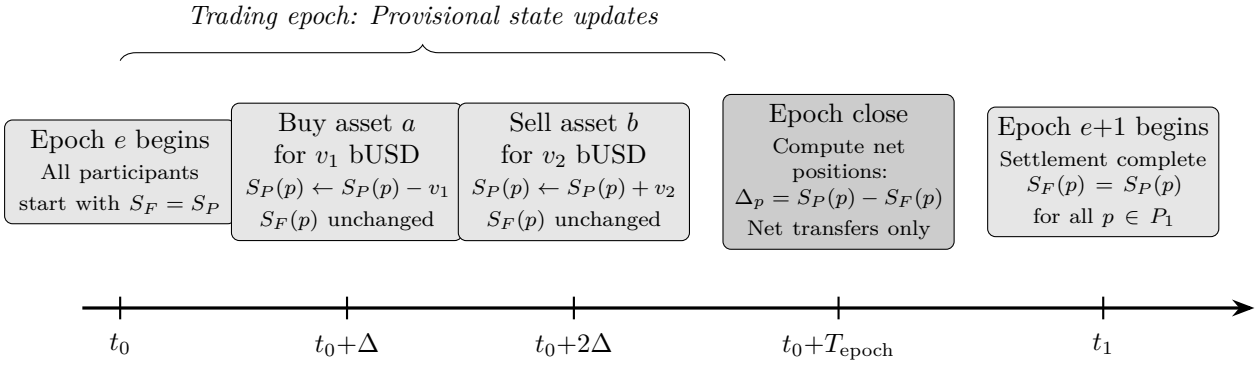


Figure 2: Epoch structure for qualified participants. During the trading epoch (light gray), transactions execute against provisional state S_P while finalized state S_F remains frozen. At epoch close (dark gray), validators compute net positions and execute settlement transfers. The next epoch begins with all participants starting from their settled balances.