Sei Giga

v0.1

Benjamin Marsh[1], Steven Landers[1], and Jayendra Jog[1]

[1]Sei Labs

May 2025

1 Introduction

In this work, we present Sei Giga, the first multi-proposer EVM layer-1 blockchain. Sei Giga uses Autobahn [Gir+25] as a consensus protocol to allow for 5 gigagas throughput [Mar25] and sub 400ms finality under standard BFT-style security assumptions. Additionally, Sei Giga uses a custom EVM execution client that has been built from scratch, a new storage layer, and asynchronous accumulator-based state commitments. As a result, Sei Giga is able to scale to allow for web2 style usage while ensuring quick finality in settings where it is necessary such as trading, on a verifiable and secure public ledger. Sei Giga is a decentralized, permissionless Proof-of-Stake EVM chain capable of running standard EVM smart contracts written in typical EVM languages such as Solidity and Vyper. Sei Giga's EVM support is equivalent to mainchain Ethereum with the exception of EIP-4844, PREVRANDAO, the state root, the block gas limit, and the transaction fee mechanism, and Sei Giga will continue to support EVM upgrades to maintain near parity. As is standard in Proof-of-Stake blockchains, Sei Giga is maintained by a set of staked nodes which act as validators and executors in the network responsible for ensuring consensus is met and blocks are run; the nodes are staked and eligible for block rewards to ensure an economic incentive to keep the chain running, and subject to slashing of the staked coins in the case of malicious behavior.

In order to ensure speedy consensus, Sei Giga reaches initial consensus over the ordering of transactions in a block and not the state of the chain given that block. Given the deterministic nature of EVM execution, this allows the chain to asynchronously execute blocks post-finality in order to reach an agreed state in a later block. The split block processing model allows Sei Giga to avoid execution bottlenecks by slowing consensus. Gas and other associated fees and rewards on the chain are handled by the underlying native coin of the chain, SEI. Sei Giga does not utilize a mempool, and transactions are immediately included by the node. Sei Giga supports multiple types of clients, including

validator nodes, which participate in both consensus and execution; a full node, which consists of the RPC layer and the execution layer for the reading path; light nodes, which consist of the RPC layer alone, and data nodes, which exists purely to serve recent data. Sei Giga utilizes direct node-to-node networking preventing propagation across nodes for performance purposes. Transactions when submitted to the RPC the transaction will be randomly allocated to a validator reducing the impact of validator targeted spam attacks, transactions may be submitted to multiple validators for censorship resistance leading to a fairer transaction inclusion process. A partial refund of the transaction tip will be refunded if multiple copies of the transaction exist and only a single copy is executed.

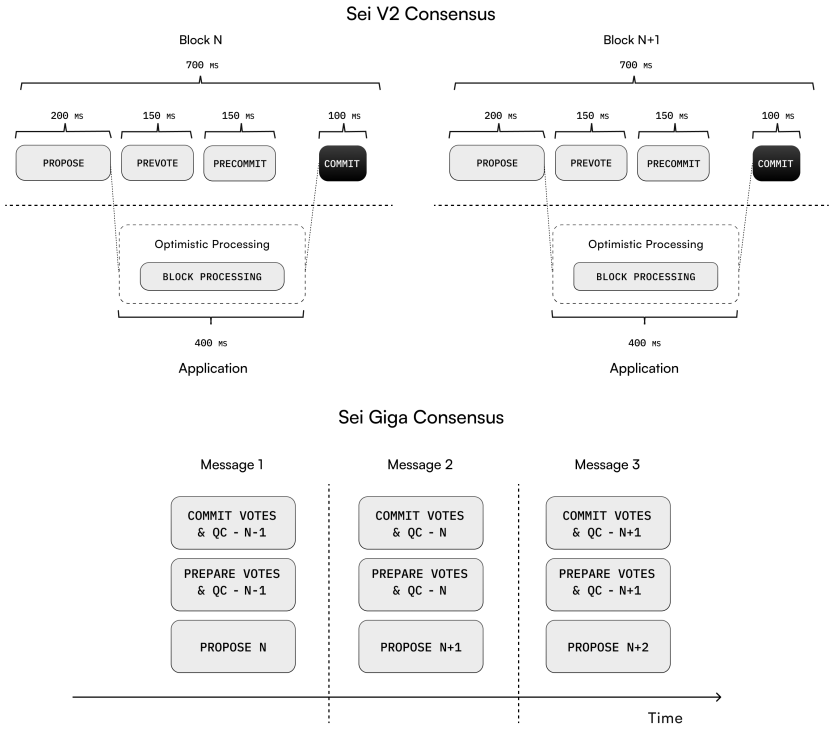


Figure 1: v2 vs Giga

1.1 Comparison to Sei v2

Autobahn achieves over 50× higher throughput than Tendermint by fundamentally rethinking how consensus and data availability are managed in a dis-

tributed system. In Tendermint, each block is processed sequentially: A single leader proposes a block, validators must download and fully execute the block transactions, and then consensus is reached in multiple rounds before the next block is proposed. This tight coupling between data dissemination, execution, and consensus means that every block decision incurs significant overhead and waiting time, resulting in a throughput limited by the speed of one proposer and the time taken to complete all three rounds of communication.

Autobahn, on the other hand, decouples these processes to dramatically increase performance. Instead of relying on one leader at a time, every node continuously disseminates its own stream of data proposals in independent lanes. This multi-proposer architecture eliminates the bottleneck of a single proposer by allowing parallel data proposals. In this system, each node operates as its own proposer, continuously creating a chain of proposals. The consensus layer periodically commits a "tip cut," which is a compact snapshot that aggregates the latest proposals (tips) from every lane. Since each tip implicitly references the entire history of its lane, this mechanism allows multiple blocks to be committed in one consensus instance without having to process each block individually. A critical innovation in Autobahn is its decoupling of data availability from consensus. In Tendermint, validators must download and verify the entire block before they can cast their votes, which ties the consensus process directly to the full dissemination of data. Autobahn, however, uses proofs of availability (PoA) to certify that a block's data is accessible without requiring every validator to download it immediately. Validators can vote based on these compact proofs while data synchronization occurs asynchronously in the background. This separation means that the ordering of transactions is not slowed down by the need to fetch full block data during the consensus round.

Autobahn further reduces latency by lowering the number of communication rounds. While Tendermint typically requires three full rounds of message exchanges to finalize a block, Autobahn's optimized protocol reaches consensus in 1.5 roundtrips in Sei Giga. Fewer rounds of communication directly translate into lower per-block latency and, consequently, higher overall throughput. Additionally, Autobahn supports parallel slot execution where new consensus slots can begin before previous ones are fully committed. This pipelining ensures that the system continuously processes proposals without waiting for each individual block's final commitment. Combined with an efficient view-change mechanism that quickly recovers from network blips or faulty leaders, Autobahn maintains high performance even under suboptimal conditions. Alongside pipelined block production, Sei Giga's Autobahn allows for asynchronous block execution, unlike Tendermint, meaning we reach consensus over the ordering of transactions in a block and agree on the state later; this ensures that execution never becomes a bottleneck in the block production process.

The execution process is fully pipelined with parsing, address recovery, and signature verification happening in parallel to avoid bottlenecks in the execu-

tion process in a new client rewritten since v2. Post-processing of the block, such as receipt generation, is handled out of the hot path meaning the next block can start execution while the previous block is undergoing post-processing. By default, the frequently accessed state of Sei Giga is stored in RAM for optimum performance, with the full state stored on disk. The system is designed to minimize disk reads by ensuring most reads for a transaction (such as balance checks) are in memory, while disk writes are performed asynchronously solely for recovery purposes.

Overall, by decoupling execution from consensus, enabling multi-proposer parallelism, committing multiple blocks in a single tip cut, reducing the number of communication rounds, and separating data availability from the critical consensus path, the Sei Giga upgrade brings significant improvements over Sei v2. The improved execution client achieves a 40x performance boost with a wholly rebuilt client from scratch, enabling 200k TPS compared to the previous 5k TPS. The faster consensus algorithm reduces voting rounds from 3 to 1.5, resulting in a 2x improvement in consensus efficiency. The novel data availability layer with its multi-producer design delivers a 70x improvement in block production with 180 blocks versus 2.5 previously while still ensuring strong Byzantine Fault Tolerance.

2 Async Execution

Sei Giga operates on the basis that once ordering is agreed upon in a block that the deterministic execution of that block will always result in the same output state, that is to say that consensus can be reached over just the contents and ordering of a block without execution.

Claim 1 (Deterministic Execution)**.** *Let S_{init} be an initial EVM state and let $\{\,\text{tx}_1, \text{tx}_2, \ldots, \text{tx}_n\}$ be an ordered list of transactions. If all honest nodes apply these transactions in the same order to the same initial state, then they will arrive at an identical final state S_{final} assuming a majority honest network.*

Sketch. Consider each transaction tx_i as a deterministic function

$$\mathsf{Exec}\big(S, \text{tx}_i\big) \;=\; S',$$

where S is the global state before applying the transaction and S' is the updated state after execution. Since EVM execution has no hidden sources of randomness or nondeterminism, the result of Exec is uniquely determined by S and tx_i. For a block containing n transactions, the final state is computed by sequential, or parallel where viable, application:

$$S_{\text{final}} \;=\; \mathsf{Exec}\big(\mathsf{Exec}(\ldots \mathsf{Exec}(S_{\text{init}}, \text{tx}_1)\ldots), \text{tx}_n\big).$$

All honest nodes starting with the same S_{init} and applying the same transactions in the same order must thus produce the same S_{final}. Consequently, consensus

need only finalize the ordering of blocks, and each node can locally compute the same state in a consistent fashion.

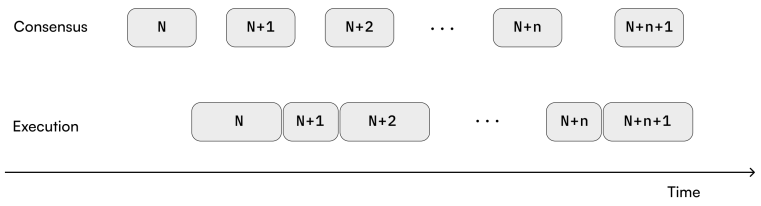


Figure 2: Async block execution

2.1 State consensus

Given the lack of need for execution to reach consensus Sei Giga is able to remove execution as a bottleneck and asynchronously execute finalized blocks in parallel to the block production process. Once a block has been finalized the executor node will execute the block and commit to the state of block n along with others as a batch in a later block $n + x$ for some $x \in \mathbb{Z}, x < 1000$ in a future tip cut once it has a $\frac{2}{3}$ quorum, after this the block is not only finalized but verified through the state consensus having had the state for a single or multiple executed blocks gossiped around the network. During this time the chain continues to produce the blocks $n + 1, n + 2, \ldots$.

The delayed asynchronous state of a block n being included in later block proposals allows for validators to ensure they have not deviated from the agreed chain state, in the case of a small scale deviation where $\phi < \frac{1}{3}$ of the network diverges on the state of the chain the chain will continue as normal, if $\phi > \frac{1}{3}$ then the chain will pause as it would in any standard BFT protocol. A software or hardware bug could allow for the state to diverge despite honest agents hence the need for consensus. The state root is attested by a signature. A transaction that fails to execute does not invalidate the execution of the block but that transaction remains failed. The block production process can be seen as a separate process to execution, with the final state of the execution of a block being committed in a later block.

2.2 Execution client

The execution client is based on evmone [Eth25], a c++ EVM implementation, as opposed to geth. The client only handles tx processing and does not handle

any other functionality such as tracing or log searching. Blocks are received and processed in parallel meaning only the block execution step happens sequentially, that is to say that all pre-processing and checks such as signature recovery happen in parallel, though only a single block will be executed at once. Post-processing of the block, such as receipt generation, is handled out of the hot path meaning the next block can start execution while the previous block is undergoing post-processing. The client is optimised to reduce disk reads by ensuring *most* reads for a transaction (check balance) are in memory. Disk writes are all asynchronous.

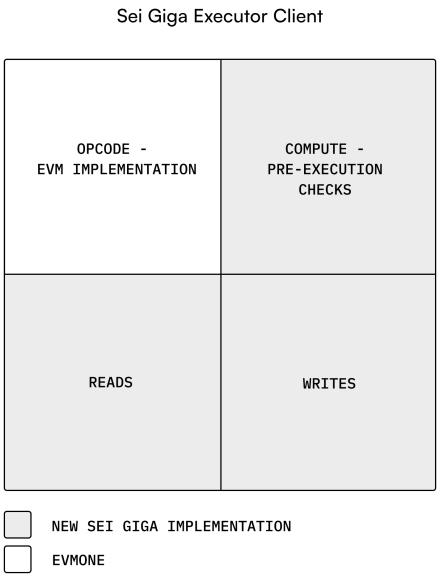


Figure 3: Giga Client

2.3 Pipelining and encoding

The execution process is fully pipelined with parsing, address recovery, and signature verification happening in parallel to avoid bottlenecks in the execution process by ensuring these otherwise potentially blocking processes.

A flat encoding format is also utilized to ensure cheaper transaction decoding. The transaction encoding is a flat, length-prefixed layout. Each field—such as nonce, gas limit, and signature—is written out in a known order. Variable-length fields are prefixed with a small byte indicating their length. This design

eliminates nested structures, enabling fast, single-pass decoding, straightforward zero-copy parsing and overall lower overhead. Once all fields are consumed, any remaining bytes are assumed to be the transaction input. This approach ensures minimal serialization overhead.

Algorithm 1 Parsing a Transaction Payload

Require: *payload* (`[]byte`)
Ensure: *tx* is populated
1: $ptr \leftarrow 0$
2: Read and store *Type*, *ChainID*, *Sender*, *To*, *Value*, and other fields by consuming the appropriate number of bytes from *payload*.
3: **if** next byte indicates a contract creation **then**
4: $\quad$ *To* $\leftarrow$ nil
5: **end if**
6: Read an integer n for the number of access-list entries
7: **for** $i = 1$ to n **do**
8: $\quad$ Read an integer m for the number of storage keys
9: $\quad$ **for** $j = 1$ to m **do**
10: $\quad\quad$ Read each storage key from *payload*
11: $\quad$ **end for**
12: $\quad$ Add the address and its keys to *tx.AccessList*
13: **end for**
14: Remaining bytes become *Input*

3 Consensus and Data Dissemination

Autobahn [Gir+25] is a BFT-style consensus protocol, used as a Proof-of-Stake mechanism in Giga, designed to be performant in the partial synchrony model like HotStuff [Yin+19]. Where other highly performant protocols offer low-latency in fault-free synchronous periods or robust recovery from interrupts. To bridge the gap between traditional view-based protocols which suffer during blips and DAG-based protocols which offer non-optimal latency during good intervals, Autobahn offers a hybrid approach with a parallel asynchronous data dissemination layer with a low-latency, partially synchronous consensus mechanism. Thus Autobahn is able to avoid the hangovers of traditional protocols while matching BFT-throughput with half the latency. A stake weighted leader selection process akin to Tendermint is used to select leaders. Formal discussions of the security of Autobahn can be found in [Gir+25].

3.1 Overview and High-Level Architecture

Autobahn separates data availability from ordering. At its core is a data dissemination layer that organizes proposals into independent lanes, each enhanced with a Proof of Availability (PoA). This ensures that data is available with minimal latency overhead even before being ordered into the global chain. The

consensus layer then periodically commits a snapshot (a *cut*) by ordering the latest certified tips from all lanes.

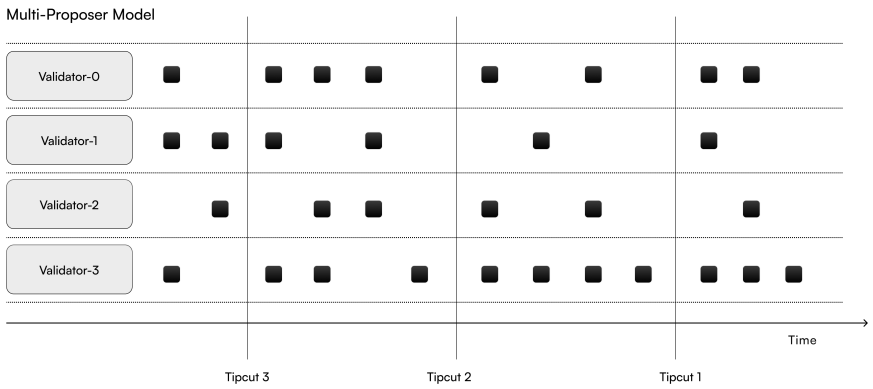


Figure 4: Multi-producer

3.2 Data Dissemination Layer (Lanes) and PoA

The DA layer's sole job is to guarantee that every batch committed in a cut can later be downloaded by any honest node. We achieve this with an f+1 Proof-of-Availability certificate (PoA).

Lane Structure and Proposal Process. Each replica r maintains its own lane ℓ_r, in which it sequentially proposes new batches (or *cars*) of transactions:

$$\ell_r^0, \ell_r^1, \ell_r^2, \ldots$$

When replica r has a new batch of transactions B, it constructs a data proposal:

$$\text{Prop} \;=\; \langle \text{pos}, B, \text{parentRef} \rangle_r,$$

where pos denotes the sequence number in the lane and parentRef references the hash of the previous proposal in ℓ_r.

Voting and PoA Certification. Each replica that receives and validates Prop (i.e., by checking that parentRef matches the previously voted proposal) returns a vote:

$$\mathit{Vote} = \langle \text{digest(Prop)} \rangle.$$

Once r collects $f + 1$ matching votes, it assembles a PoA:

$$\text{PoA} \;=\; \big(\text{digest(Prop)}, \{\sigma_i\}_{i \in Q}\big) \quad \text{with} \quad |Q| = f + 1.$$

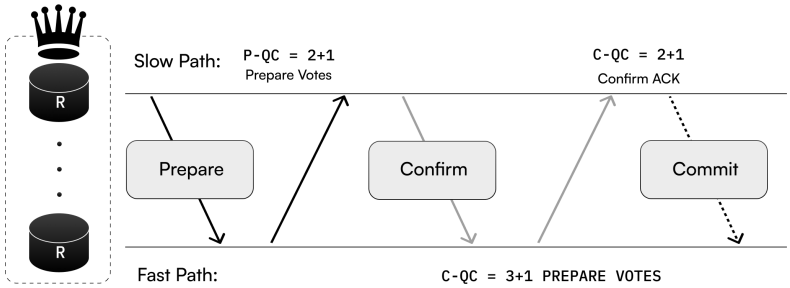
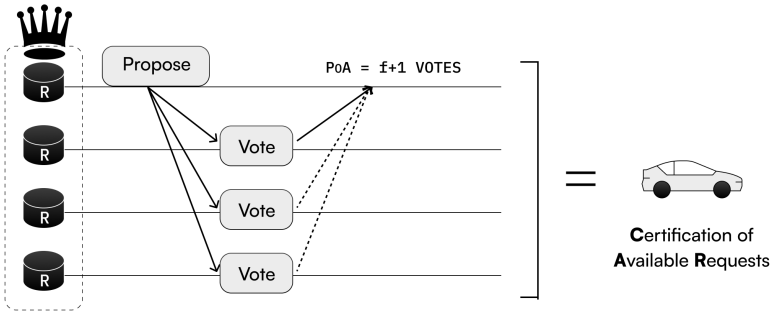


Figure 5: Prepare

This certifies that at least one correct replica can serve the proposal data on request.



]

Figure 6: Proposal

Instant Referencing. Because proposals in lane ℓ_r are chained, referencing the tip (i.e., the most recent proposal with a PoA) implicitly attests that all previous proposals in that lane are available. This transitive guarantee reduces synchronization overhead when lanes are later merged in the consensus process.

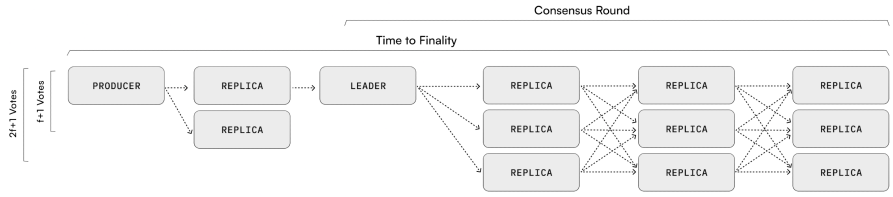


Figure 7: DA and consensus stages with vote reqs

3.3 Data Availability and Final Execution

Data Synchronization (DA) and Asynchronous Retrieval. Since each lane's tip carries a PoA, any replica missing a batch can retrieve it asynchronously from a replica within the PoA set. When a global Cut is committed, any missing batches must be fetched off the critical path. If the leader is correct and the network is synchronous, Autobahn commits a new proposal in at most two communication rounds per slot. When a leader fails to make progress, replicas revert to a standard view change mechanism and elect a new leader via a timeout certificate.

Final Global Ordering. Once the global order is fixed via the committed Cut, all nodes execute transactions in the predetermined order, which will involve a second reordering of transactions by tip. Consistent local execution (as per Prop. 1) ensures all correct replicas reach the same final state.

3.4 Consensus Layer (Cut of Tips)

Global Ordering via Cuts. The consensus layer periodically commits a snapshot of the system by collecting the latest certified tips from all lanes:

$$\text{Cut} \;=\; \{\ell_1[\text{tip}],\, \ell_2[\text{tip}], \ldots, \ell_n[\text{tip}]\}.$$

A designated leader (selected via a stake-weighted process similar to Tendermint) initiates the ordering process.

Protocol Phases and Pipelining. Autobahn implements a two-phase BFT agreement protocol:

1. **Prepare:** The leader collects the most recent certified tips from all lanes and bundles them into a proposal. Honest replicas then broadcast their prepare votes.

2. **Commit:** Once enough votes are collected, a CommitQC is formed, finalizing the blocks.

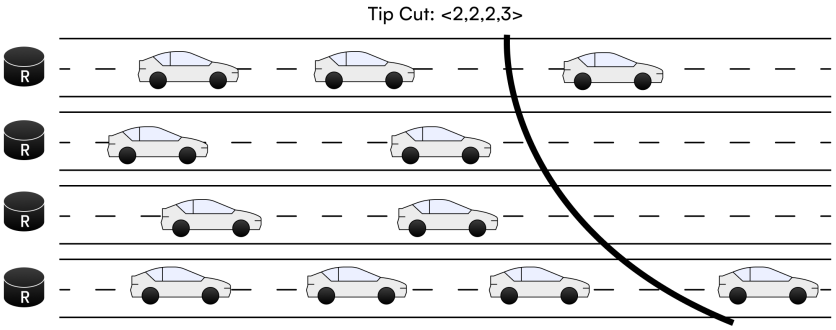


Figure 8: Tip cut

3. **Confirm:** If the leader gathers only $(n - f)$ votes, it enters a confirm phase and waits for additional acknowledgments until a commit certificate is achieved with $2f + 1$ confirm messages.

Pipelining Benefit:
Through the use of quadratic communication and pipelining, Sei Giga achieves a slow path round trip of 1.5 rather than 2.5. That is to say that the leader sends a proposal for tip cut N, triggering nodes to send their prepare votes. Once enough votes are collected, they form a PrepareQC. After that, nodes send their commit votes, and once enough are collected, a CommitQC is formed, finalizing the blocks in N. While N is in its commit phase, the leader for tip cut $N + 1$ may start sending its proposal. In other words, even though each block goes through two distinct rounds, pipelining allows the next block's proposal to start during the commit phase of the previous block, effectively reducing the overall latency to 1.5 rounds.

Parallel Slots. To minimize delay between successive blocks, consensus slots are pipelined. Once a replica sees the Prepare message for slot s, it can begin slot $s + 1$ without waiting for slot s to fully commit, ensuring that proposals arriving just too late for one slot do not cause extra round delays.

3.5 System Model, Security, and Reliable Inclusion

System Model and Assumptions.

- **Replicas:** There are $n = 3f + 1$ replicas; up to f may be faulty.
- **Authentication:** All messages are transmitted over authenticated, point-to-point channels and cryptographic primitives are unforgeable.

- **Partial Synchrony:** Safety is unconditional; liveness requires eventual network stabilization such that message delays respect known upper bounds.
- **Clients:** There is no upper bound on potentially faulty clients; clients submit transactions that are batched into proposals.

Security and Reliable Inclusion. Autobahn's design ensures that no two conflicting proposals can both obtain a valid commit certificate in the same slot. Key points include:

1. Once a proposal receives $f+1$ votes, at least one correct replica stores the data.
2. A correct leader must include such proposals in a future Cut.
3. Faulty proposers cannot indefinitely disseminate data without eventual inclusion.

This mechanism prevents Byzantine replicas from causing wasted dissemination and protects against censorship beyond a bounded delay.

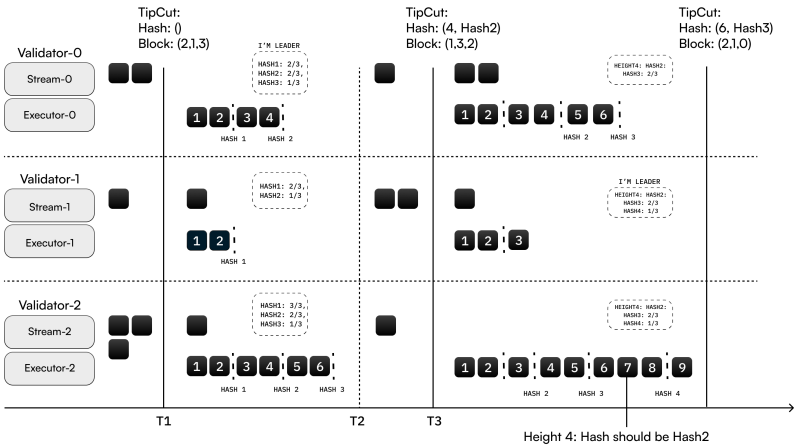


Figure 9: Async block execution

4 Block-STM-Style Parallel Execution

Let $\{t_1, t_2, \ldots, t_n\}$ be the transactions in a finalized block B, arranged in the total order determined by that block. Each transaction t_i reads some set of addresses, storage slots, or global variables, collectively denoted by R_i, and writes

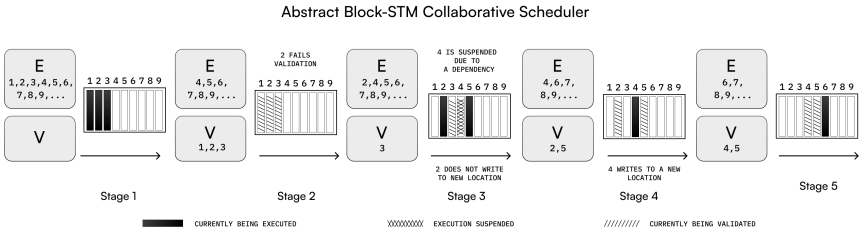


Figure 10: STM

a set of addresses or slots denoted by W_i. We say that t_j depends on t_i, denoted $t_j \to t_i$, precisely when $(j > i)$ and there is at least one element in W_i that intersects with $(R_j \cup W_j)$. Intuitively, this captures all cases in which t_j must see the updated value produced by t_i. Because the block imposes a total order on transactions, the resulting dependency relation is acyclic.

Sei Giga uses optimistic concurrency control (OCC) to leverage the fact that most transactions are unlikely to conflict. In OCC, each node begins by executing all transactions in parallel, distributing them across multiple worker threads and assuming no conflicts will occur [And24]. While executing, each transaction t_i uses a private buffer to store changes to its write set W_i, rather than updating the globally visible EVM state. Once t_i finishes, it enters a validation phase to detect conflicts with any committed transaction t_k where $k < i$. A conflict arises if t_k has written to an address that is in either R_i or W_i after t_i began. If such a conflict is found, t_i is rolled back and re-executed (potentially in a more conservative manner). If no conflict is detected, the changes in W_i are committed to the global state, and t_i is marked complete.

By enforcing that t_j cannot commit before all transactions t_i with $t_j \to t_i$ have committed, the system preserves an execution order consistent with the block's logical sequence [Gel+22]. Consequently, all honest nodes converge to the same final post-execution state, just as if they had run the transactions in a purely sequential fashion from t_1 through t_n. The key advantage is that when conflicts are rare, most transactions do not require re-execution, and the parallel execution phase greatly reduces total latency on modern multi-core hardware. Thus, while OCC may necessitate rolling back and retrying the occasional conflicting transaction, the performance benefit of parallelizing non-conflicting transactions outweighs the re-execution overhead. The process falls back to sequential processing if the parallel process continually fails.
By allowing for parallel, rather than sequential transaction processing, as shown in figure 11 and figure 12 it is trivial to see how execution times can be reduced.

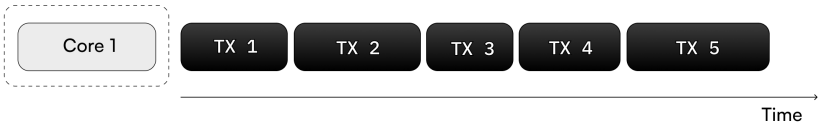


Figure 11: Sequential tx execution

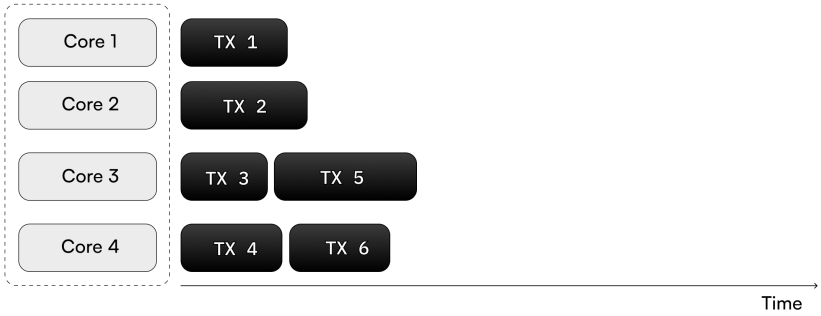


Figure 12: Parallel tx execution

5 Storage

Sei Giga adopts a storage strategy that departs from conventional Merkle-tree-based designs in order to reduce overhead and improve parallelism under extreme throughput. By default the state of Sei Giga is stored in RAM. Instead of maintaining a Merkle tree for every update to global state, the chain uses a flat key-value model, where each account, contract storage slot, or globally accessible variable is mapped directly to a corresponding entry in a log-structured merge (LSM) tree. Storing $\langle k, v \rangle$ pairs in this manner is particularly effective for write-intensive environments. Sequential updates are efficiently batched and flushed to underlying media.

By eliminating the overhead of traversing or updating multiple tree nodes for each write, a flat key-value design inherently reduces structural churn, streamlines concurrency, and yields more predictable I/O behavior. Because each update targets a single $\langle k, v \rangle$ pair rather than an entire branch, locking and write contention are simplified, facilitating parallel access across distinct keys without intricate coordination. In addition, storing data in a single-level structure reduces pointer-chasing and deep lookups, enabling more efficient batch

operations within the underlying LSM engine. Together, these benefits allow Sei Giga to sustain high throughput without the frequent re-hashing and node-level bookkeeping overhead that typify Merkle-tree-based approaches.

Sei Giga further reduces storage burdens through a tiered approach. Recent blocks and frequently accessed data reside on high-performance local SSDs, ensuring fast lookups and updates. Historical or rarely queried information, by contrast, is migrated to a cold layer that leverages a distributed, columnar database. This cold layer architecture alleviates the need for every validator node to store the complete historical ledger on expensive primary media; it also provides a scalable interface for analytical queries, audit trails, and forensics. As the network processes petabytes of new data each year under a 5 gigagas load, offloading stale state from SSDs to cost-effective archival storage becomes central to sustaining practical hardware requirements for validator operators.

Internally, Sei Giga's implementation leverages an append-only write-ahead log (WAL) to guard against crashes and data corruption. Updates are written sequentially to the WAL and then applied to RocksDB, which manages the LSM tree and coordinates compaction routines to maintain balanced input-output performance. This mechanism ensures durability, enabling a node to recover from abrupt failures by replaying the WAL and returning the database to a consistent state. Meanwhile, the cryptographic accumulator is updated according to finalized blocks, preserving a verifiable reference to each version of the global state without demanding an immediate, full-branch hashing procedure.

Through this hybrid of a flat LSM-based key-value store, asynchronous accumulator-based proofs, tiered storage, and an append-only WAL, Sei Giga provides a storage subsystem optimized for continuous high throughput. The approach avoids the overhead typically introduced by Merkle-tree re-computations, while sustaining strong security guarantees and furnishing a clear growth path for accommodating multiple terabytes of new data annually. This design also ensures that user-facing nodes can serve recent data efficiently and that light clients can validate state transitions without bearing the storage cost of all historical versions, thereby preserving decentralization and verifiability under increasing load.

5.1 Accumulator

To ensure verifiability without incurring the overhead of recomputing Merkle paths on every write, Sei Giga replaces the traditional tree structure with a pairing-based cryptographic accumulator [VB20] allowing for constant time operations. This accumulator is updated asynchronously and can compactly aggregate membership and non-membership proofs across many keys in a single structure. By allowing proofs to be generated or refreshed in batches, the system avoids the tight coupling between each individual state update and an expensive re-hashing procedure. Validators and light nodes can still obtain strong cryp-

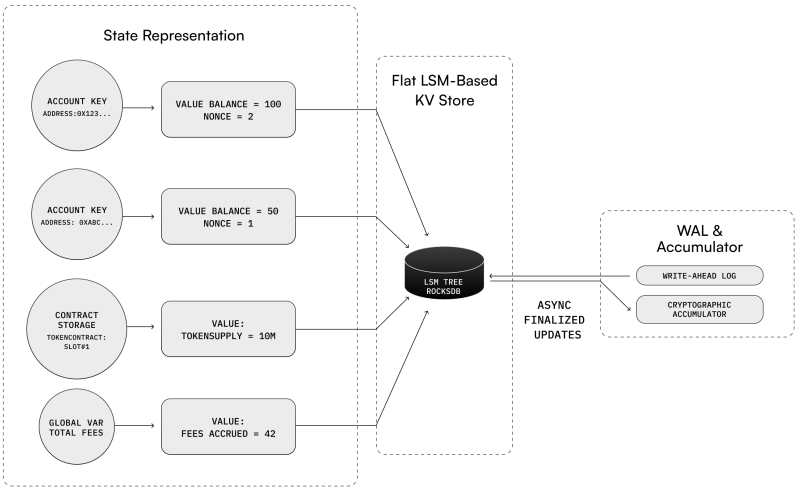


Figure 13: Storage

tographic guarantees of correctness and data availability, since the accumulator attests to an agreed-upon state that encompasses recent updates without requiring each node to store or compute a full Merkle path. The absence of rigid tree-based branching also affords flexible partitioning of the key space, facilitating parallelism across different shards of state. A light node receives a compact, aggregated state commitment from the accumulator alongside batched proofs for the keys or transactions it cares about. It then employs a verification algorithm to check each proof against this commitment, ensuring the correctness of membership or non-membership without needing to reconstruct full Merkle paths.

6 Economics

Sei Giga has a native coin called SEI with a total supply of 10 billion, 10_000_000_000. The Sei coin is used to pay for gas and other associated transaction and usage costs, as well as for staking and validator rewards. Each block receives a block reward in Sei which is not paid to the block producer but the sum of all rewards in an epoch is paid equally to allowed validators and their delegators, $r = \frac{\sum_{b \in e} b_r}{|V|}$. Each validator must stake Sei to run a validator node and the stake is subject to a 21 day bonding period, the 21 day period also applies to delegated stake, chosen to allow for slashing of malicious behaviour by nodes. Slashing penalties apply to liveness, and double signing of blocks at the same

height. Slashing penalties vary by the severity of the action, such that double signing incurs a greater penalty than liveness, and increase in severity for future penalties.

6.1 Governance

Governance on Sei is a community-driven process, that mirrors the approach taken in v2, that lets stakers propose and vote on network changes. After a proposal is submitted and the required deposit is met to discourage spam, it moves to a voting phase where stakers weigh in. If quorum and majority requirements are satisfied, the proposal passes; otherwise, it fails and deposits may be burned. Passed proposals that change the chain's parameters or software are automatically enacted, while more general proposals require manual execution. All governance data is publicly accessible on Seiscan or via direct blockchain queries.

References

[Yin+19] Maofan Yin et al. *HotStuff: BFT Consensus in the Lens of Blockchain*. 2019. arXiv: 1803.05069 [cs.DC]. URL: https://arxiv.org/abs/1803.05069.

[VB20] Giuseppe Vitto and Alex Biryukov. *Dynamic Universal Accumulator with Batch Update over Bilinear Groups*. Cryptology ePrint Archive, Paper 2020/777. 2020. URL: https://eprint.iacr.org/2020/777.

[Gel+22] Rati Gelashvili et al. *Block-STM*. 2022. eprint: 2203.06871v3. URL: https://arxiv.org/pdf/2203.06871.

[And24] Vangelis Andrikopoulos. *64.85 of Ethereum Transactions Can Be Parallelized*. 2024. URL: https://blog.sei.io/research-64-85-of-ethereum-transactions-can-be-parallelized/.

[Eth25] Ethereum. *evmone*. 2025. URL: https://github.com/ethereum/evmone.

[Gir+25] Neil Giridharan et al. *Autobahn: Seamless high speed BFT*. 2025. arXiv: 2401.10369 [cs.DC]. URL: https://arxiv.org/abs/2401.10369.

[Mar25] Ben Marsh. *Sei Giga: Achieving 5 Gigagas with Autobahn Consensus*. 2025. URL: https://blog.sei.io/sei-giga-achieving-5-gigagas-with-autobahn-consensus/.