Highlayer: A New Architecture for Effective Rollups

Ver. 1
Mykyta Rykov nik@pentachoron.tech
Samuel Poirier sam@pentachoron.tech
Max max@pentachoron.tech
Crimson crimson@pentachoron.tech

Abstract

This paper introduces Highlayer, a novel Layer 2 solution based on bitcoin, employing an improved sovereign rollup architecture. Highlayer's approach differentiates itself by integrating a suite of novel technologies designed to alleviate the inherent constraints of traditional blockchain networks. These improvements include proposals for more effective data availability (DA), enhanced execution of computationally intensive tasks, dApp user experience (UX), censorship-resistance and improved developer accessibility.

1 Problem

When Satoshi Nakamoto was developing the Bitcoin protocol, he aimed to solve one specific issue: creating a decentralized, censorship-resistant mechanism for transaction timestamping [1]. The blockchain model worked well for achieving maximum possible decentralization and censorship-resistance, however it did not assume great scalability in terms of fast finality and transaction throughput.

Blockchain technology was designed with a sole focus on security, decentralization, and immutability. This enabled the blockchain to work well on a small scale, but it faced challenges in meeting global demand due to its limited transaction throughput. A notable early example of these scalability issues is the ‘Ethereum scalability Crisis' (the 2017 CryptoKitties activity surge), during which the network experienced a significant backlog of transactions [2]. This backlog resulted in substantially higher fees and processing delays than usual.

These limitations, however, do not diminish the significance of Satoshi’s original vision for a robust, multipurpose payment system. As the needs of users in the blockchain ecosystem have evolved, advancements such as optimistic rollups, ZK-rollups, and sharding, have marked the beginning of a new era of innovation.

These developments have sought to address the scalability and versatility challenges of earlier blockchain models.

2 Solutions

People have been experimenting with solutions to this problem for a long time, inventing scaling mechanisms such as sharding [3], appchains – application-specific blockchains, and layer 2 networks, including rollups – systems that are built on top of existing blockchains, getting their security and censorship-resistance from the underlying layer 1 network, while getting better throughput due to (non-mandatory) ability of centralized transactions processing.

This article's primary focus is on rollups - layer 2 systems built on top of established, secure and decentralized layer 1 networks, other solutions are out of the scope of this paper.

2.1 Rollups

Rollups have emerged as a notable innovation to the scalability problem, providing a mechanism to increase transaction throughput beyond the inherent limitations of traditional blockchains. This section is dedicated to the two currently predominant types of rollups – zero-knowledge rollups (zk-rollups) and optimistic rollups, examining their operational methodologies, benefits, and inherent challenges.

2.1.1 Optimistic Rollups

Optimistic rollups [4] are a form of layer 2 scaling solution, primarily designed to alleviate the computational burden on the base layer (layer 1) nodes. The core principle of optimistic rollups is to minimize the consensus process regarding transaction execution by these base layer entities. In this model, transactions are executed off-chain, and only the transaction data and outcome state of these transactions is posted to the layer 1 blockchain. This approach reduces the amount of computational work required by the main blockchain nodes, functioning effectively as a gas optimization technique.

In an optimistic rollup, the transaction data is aggregated and submitted to the base layer by a centralized relayer, often referred to as a "sequencer". The sequencer performs the role of collecting, executing, and posting transaction data along with the outcome state to the layer one blockchain. Most importantly,

transactions within an optimistic rollup are assumed to be valid by default — posted by the sequencer, which puts itself at the stake of transaction being valid, with ability to challenge outcome state by anyone (thus, slashing sequencer and preventing attack), allowing it to be an intermediary for efficiency rather than as a fully trusted entity.

Consequently, these transactions are not immediately executed or validated by the base layer nodes. Instead, they are appended to the blockchain in a provisional state, awaiting potential validation or challenge.

The system's integrity is safeguarded by a group of nodes, typically selected by the project's founders. These nodes are empowered to identify and challenge outcome state submissions by the sequencer through fraud proofs, introducing the need for “fraud window”, required for validator nodes to pick up layer 2 transactions and have enough time to be able to submit a “fraud event” to the base blockchain in case of sequencer being compromised.

However, this model introduces notable concerns regarding centralization and effectiveness of its scalability. The reliance on a select group of nodes for transaction validation and fraud detection could pose serious risk to users of such layer 2 networks. In a hypothetical situation, where the sequencer and validator nodes decide to collude against the network, they could withdraw all users’ funds and facilitate any layer 2 transaction on behalf of its users.

Scaling efficiency for optimistic rollups faces inherent limitations too. A critical requirement involves the sequencer uploading transaction data to the base blockchain, enabling the fraud detection nodes to identify and penalize any malicious activity. Consequently, this requirement raises the cost of layer 2 storage with that of layer 1, thereby constraining throughput. This constraint is primarily due to the expenses and capabilities associated with transferring unexecuted raw data to the base blockchain.

2.1.2 Zero-Knowledge Rollups

Zk-rollups [5] represent a different verification system in the rollup ecosystem. The sequencer is required to submit proofs that verify the authenticity of state transitions, alongside the transaction data. These proofs, grounded in cryptographic principles, enable on-chain contracts to independently verify the

legitimacy of state changes without processing the underlying transactions in their entirety.

While zk-rollups offer enhanced security and fast withdrawals to layer 1 when compared to optimistic rollups, there are still some shortcomings. A significant challenge is the computational intensity and associated costs in generating and verifying zero-knowledge proofs on-chain. This complexity often translates into higher operational costs, posing a barrier to the broader adoption of zk-rollups, particularly for applications with limited financial resources.

Requirement for layer 1 storage of all layer 2 transactions also persists, keeping the most expensive and inefficient part in the system.

2.1.3 EIP-4844 Shard Blob Transactions

Despite the advancements introduced by rollups, a fundamental limitation persists: the on-chain storage of transaction data. While rollups effectively increase transaction throughput by compressing or aggregating transactions, they do not entirely eliminate the bottleneck of on-chain data storage. This inherent constraint continues to cap the scalability potential of blockchain networks, as the increased volume of transactions, even in a compressed format, still exerts considerable load on the underlying blockchain infrastructure.

When talking about the requirement to store transaction data on layer 1, it's fair to note the recent development in rollup technology: "EIP4844 Shard Blob Transactions" [13], which allows to optimize storage cost of layer 2 transactions on main blockchain by only checking for seeding (instead of fully storing data).

This method allows for greater layer 2 data availability cost optimization, however, it still necessitates that layer 1 nodes temporarily download all layer 2 transactions. Consequently, a limitation is imposed on the scalability of such a system: an artificial cap represented by the maximum size of blobs per block and a physical cap constrained by the internet bandwidth of layer 1 nodes.

Additionally the chosen layer 1 network where a rollup would operate has to support such blob transactions at the consensus level, stopping it from being used on the most decentralized and censorship resistant network – Bitcoin.

In conclusion, while offering benefits to simple layer 1 storage of layer 2 transactions, EIP-4844 is not a solution to limited layer 1 powered DA, it is however an optimization of it.

3 The Highlayer Solution

Highlayer is a layer 2 blockchain that uses Bitcoin as the settlement layer. Highlayer presents itself as a censorship-resistant, gas-efficient, and high-tps solution inspired by sovereign rollups [6] architecture.

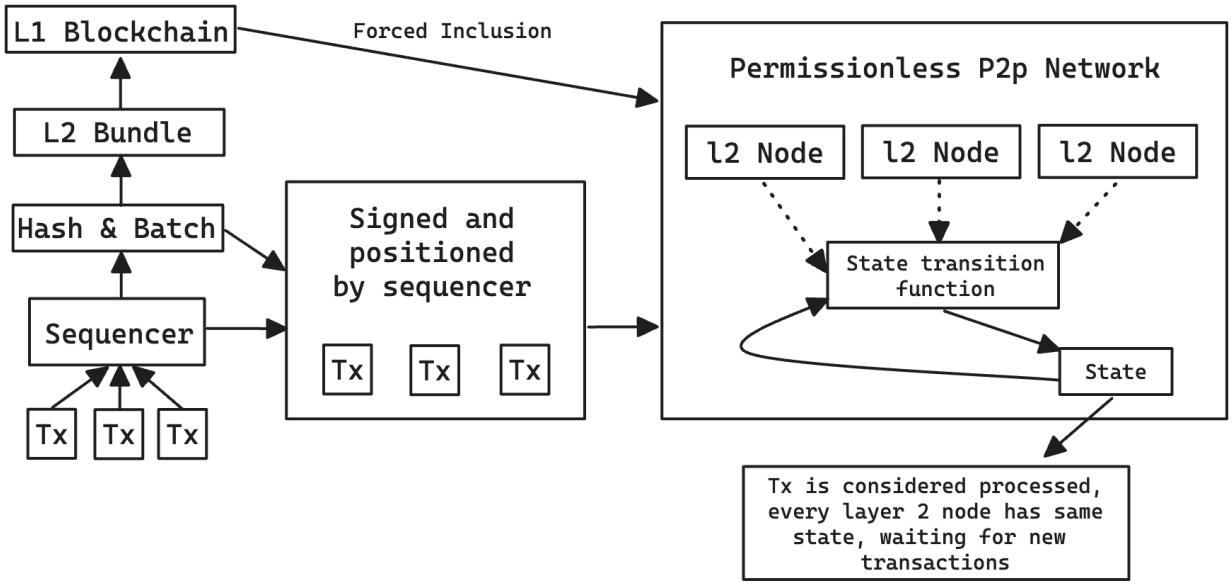

Fig. 1, Overview of Highlayer's core system

In the sections below, a more detailed description of different concepts used to achieve this architecture's objectives is presented.

3.1 Self-Sustainable Data Availability System

The cornerstone of Layer 2 security within the blockchain ecosystem is Data Availability (DA) protocol. This protocol mandates the complete and public dissemination of transaction data, thus ensuring its availability for unrestricted access. This attribute of DA is indispensable for the operational efficacy of fraud proofs within optimistic rollups. It also facilitates Layer 2 nodes in executing transactions autonomously in sovereign rollups and upholds the integrity of zero-knowledge proofs in zk-rollups.

Data Availability is often provided by layer 1 blockchains, enabling smart contracts on layer 1 blockchains to verify the validity of layer 2 blockchain state. Hence, DA unlocks permissionless asset bridging between layer 1 and layer 2 blockchains. However, DA suffers from a serious flaw: it has limited data storage

capacity on layer 1 chains. To solve this issue, Highlayer leverages the advantages of the sovereign rollup model, which inherits a lot of traits from traditional l1 blockchains, allowing it to use the base data availability system that most layer 1 blockchains use.

In most layer 1 blockchains, its network nodes, (importantly, not only miners or validators) ensure availability of block data. In most cases, this system does not need an incentive structure to work. Moreover, self-sustained Data Availability has proven itself to be one of the most bulletproof protocols, serving Bitcoin for 15 years.

In the Self-Sustained Data Availability System (SSDAS), when nodes fail to download data in situations such as data chunk loss, or when seeding of data stops abruptly, the nodes pause execution, effectively preventing the “sudden seeding” issue. Sudden seeding is an edge-case when either the miner, the sequencer, the validator, or the bundler does not share transaction info with other participants of the network at the time of block/bundle/rollup upload. Other participants continue executing transactions, unknowingly tolerating the fact that an unseeded transaction is not included in the block.

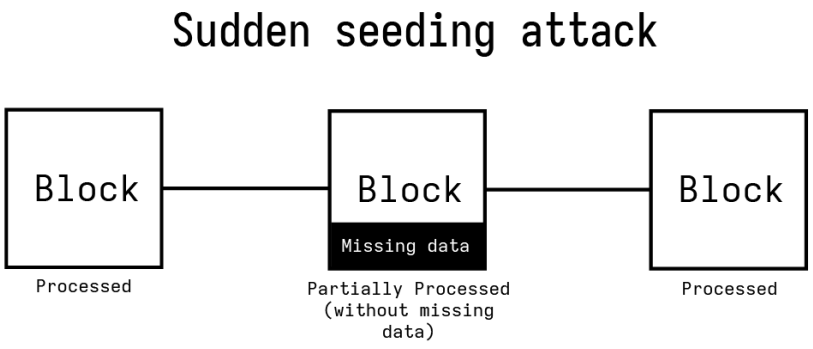

Fig.2 Sudden seeding attack

The block producer (attacker), shares missing data with a significant delay after the block was processed, which causes the outcome state to be vastly different. This enables exploits such as double spending and fake transfers.

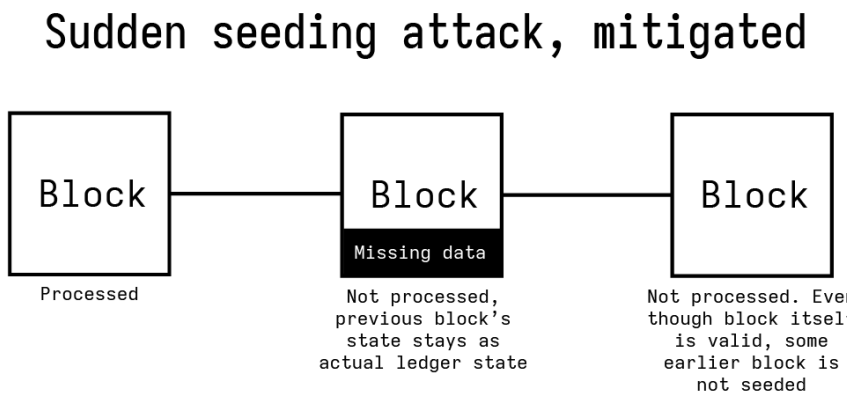

Fig.3 Sudden seeding attack, mitigated

The attacker's block with missing data is prevented from being included by other participants of the network, which causes a fork process to take place starting from the point where data is missing. This mechanism prevents malicious block producers from submitting semi-complete blocks onto the network.

SSDAS makes sure transaction data is stored permanently by using the following mechanism: if data from a block is deleted by every node, new nodes will not have the ability to get the blockchain state beyond this block. In such a situation, a new state fork is created, with all previous existing data being fully valid (invalidating any transactions committed after the faulty block with missing data).

The benefits of using such a system include simplicity and wide usage across layer 1 blockchains. Additionally, mitigating sudden seeding attacks gives an organic incentive for entities that have a stake in the project to run a full layer 2 node to ensure data integrity.

3.2 Promise of inclusion and positioning

To achieve zero-time transaction finality, the sequencer must sign transactions it receives, along with its position relative to other transactions in the ledger. If the sequencer signs 2 or more transactions with the same position – attempting a double sign attack, layer 2 nodes must stop execution at the moment of double sign attack detection, and broadcast the proof of attack to other layer 2 nodes within the same transaction, which would halt the network. Their operators then should come to a social consensus about further actions, investigate the reason for bad behavior from the sequencer, and either manually forgive it or elect a new sequencer.

This would mean that by double signing, the sequencer would not achieve its malicious goals. This would result in loss of trust for the sequencer, and network downtime of several hours. This behavior is not beneficial for the sequencer, and allows to assume, optimistically, that a transaction will be included as long as it is signed by the sequencer and there are no conflicting transactions found (network is not halted).

3.3 Forced transaction inclusion through L1

Highlayer allows for 2 distinct ways of transaction inclusion to layer 2 ledger: a slow pathway that ensures enhanced censorship resistance, and a fast pathway that prioritizes speed of inclusion with a centralized approach. The slow inclusion method leverages the recent advancements in Bitcoin, namely inscriptions[7], allowing users to embed arbitrary data directly onto the bitcoin blockchain via witness. This system provides a mechanism to enforce the inclusion of transactions that might otherwise be subject to censorship, thereby ensuring the highest degree of decentralization, security and censorship-resistance (inheriting these parameters from bitcoin).

The fast inclusion method utilizes a centralized sequencer, elected by the network nodes' operators using a local consensus (each node chooses its sequencer and follows it, similar to choosing a hard fork). This sequencer is responsible for aggregating transactions, appending timestamps, authenticating them through cryptographic signatures, and disseminating this information across the Layer 2 peer-to-peer network of nodes. Nodes within the network then independently execute these transactions, maintaining the blockchain's integrity by updating to the latest served state locally derived from their own computations. This method mirrors the operational principles of traditional blockchains.

The sequencer might not always be able to approve or sign some transactions. It might be located in a region with strict AML requirements or OFAC enforcement. In this case, the sequencer may refuse signing or positioning such types of transactions. For such cases of censorship, it is possible to submit a transaction to layer 2 avoiding the sequencer, directly via layer 1 network.
Transactions submitted through the base layer are times more expensive than identical ones on layer 2 (due to the need to pay layer 1 fees for data stored in the witness field), and they inherit base layer's full finality time (long enough that it's almost impossible to ever revert it), however also inheriting censorship-resistance of the layer 1's settlement layer.

This system, inspired by Arbitrum's Delayed Inbox[8] is presented in Figure 4.

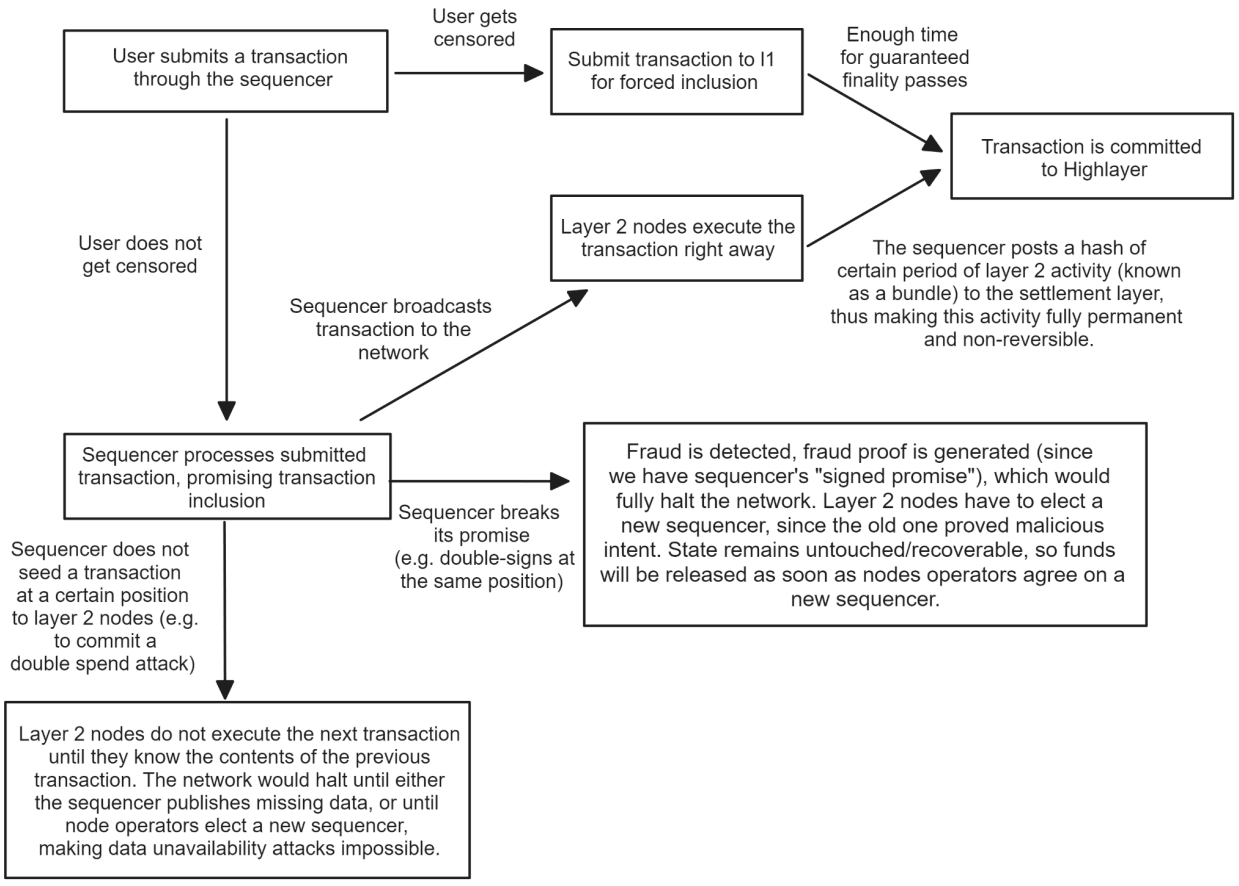

Fig.4 The path of a Highlayer transaction

3.4 Highlayer Virtual Machine

The Highlayer Virtual Machine (HVM) stands as the core smart contract engine of Highlayer. It is designed to deliver a seamless developer experience while maintaining a competitive execution speed.

The conceptualization and development of the HVM were guided by the principle of accessibility for all developers. Adopting JavaScript as a language for smart contracts lowers the entry barriers for the general developer audience, which positions Highlayer as an accessible ecosystem for developers who may or may not be familiar with general cryptocurrency development.

HVM uses a fork of the Duktape JS engine[9], enhanced with features for gas metering and ensuring deterministic execution. This fork is maintained by the Highlayer team.

HVM also includes features that allow for more advanced dApps built on it, such as scheduled actions and hybrid execution model.

3.4.1 Scheduled actions

HVM allows for the scheduling of code execution by contracts for a specific block or VDF clock tick (section 3.5).

It also enables contracts to make payments for the gas consumption of these scheduled computations using gas-suitable coins they own, making on-chain cron systems possible and allowing for use cases that were only possible earlier with use of centralized oracles to trigger an action or with use of own blockchains with actions per block (such as daily incentive redistribution from DEX for providing liquidity, trigger-free flow based simple on-chain games, etc).

The farther the target tick is, the more uncertain gas price at that time is.
To minimize risk for protocol of executing transactions in the future by price that would be inappropriate for that time, each tick adds 0.1% of current gas cost to the total price of scheduling.

The scheduled actions system also allows for the use of Hybrid Execution Model, described in section 3.4.2.

3.4.2 Hybrid Execution Model

HVM allows for the use of the hybrid execution model - a system that allows for cheap execution of heavy payloads without blocking the flow of other transactions, achieving more efficient, parallel execution, allowing to parallelize only what really matters - computation of heavy, computationally-intensive tasks.

Hybrid execution model works by spinning up a thread when heavy execution API is called:

```
// Phase 1
input = 0
allowedGas = 100000
// Each gas unit is x50 cheaper than in main flow, so the gas limit
provided gets multiplied by 50.

result = HVM.executeHeavyCode(`
// Phase 2

// Do heavy computing, like zero-knowledge proof operations or large
payload processing.

input+1; // Just return input + 1

`, input, allowedGas);
// Phase 3

changes.push(KV.set("result", result)) // Do something with result of
heavy computation

return changes // Can return final list of actions here
```

Code example 1. Usage of hybrid execution API in smart contract.

After *executeHeavyCode* is called, HVM suspends execution of main contract's code and starts executing the heavy code, calculating delay for resuming the main code dynamically, by gas limit provided to *executeHeavyCode* and by target pessimistic processing power of cores l2 nodes, in gas units per second. After the target block is calculated, cron job is scheduled onto this block, which resumes execution of the main contract instructions.

If an L2 node is far below average in processing power, or is syncing from genesis, HVM on that node will wait for heavy computation to finish before processing any further to ensure determinism. This will not affect rest of network in any way, but slow/syncing node will have old state due to execution being suspended by awaiting execution result

Transaction with usage of HEM (hybrid execution model) is split into 3 parts:

1. **Preparatory phase**
   The phase where all the needed components for heavy processing are being prepared, state access can happen here, everything that a normal smart contract can do is done here.

2. **Execution phase**

The phase where preparation is finished, all needed state has been fetched, and execution of a compute-heavy algorithm is started.
Gas cost is orders of magnitude cheaper here, no access to the ledger's state can be performed, only one value can be returned.

3. Finalizer phrase

This phase is scheduled to the start of the block which statistically should be produced when execution has already finished.
It has the ability to access results of heavy execution and the ability to do reads and writes to the blockchain state (by returning a list of changes to be made).

Tasks from phase 2, which involve computationally demanding operations, are limited to interacting with immutable data already passed to the executor, exemplified by the "input" in the code example 1. This ensures they cannot modify or access the blockchain's current state in any way beyond this specific interaction.

The reason for that is that heavy code being executed in parallel with other transactions makes race conditions (and non-deterministic behavior) possible when accessing the same state.

This 3-phase flow might be unintuitive to developers that did not face it before due to its asynchronous nature, where the finalizer phase can occur several blocks after the preparatory phase. This time gap means other transactions could modify the blockchain's state in the interim, potentially invalidating assumptions made during the preparatory phase. It's important for developers to recognize that while the preparatory phase allows for reading and setting up initial conditions based on the current state, allowing data transfer from the blockchain to the heavy payload, the actual state upon which the finalizer phase acts may be significantly different due to these intervening transactions.

It's critically important to conduct balance and other state-related checks during the finalizer phase, not just during the preparatory phase. Given the potential changes to the state between these phases, checks made early on might no longer be valid by the time the finalizer phase executes. This means essential validations that rely on state must be repeated to ensure that the conditions for the transaction still hold true. If these checks fail in the finalizer phase, it may be necessary to discard the computation results and revert the transaction altogether to prevent double-spending attacks.

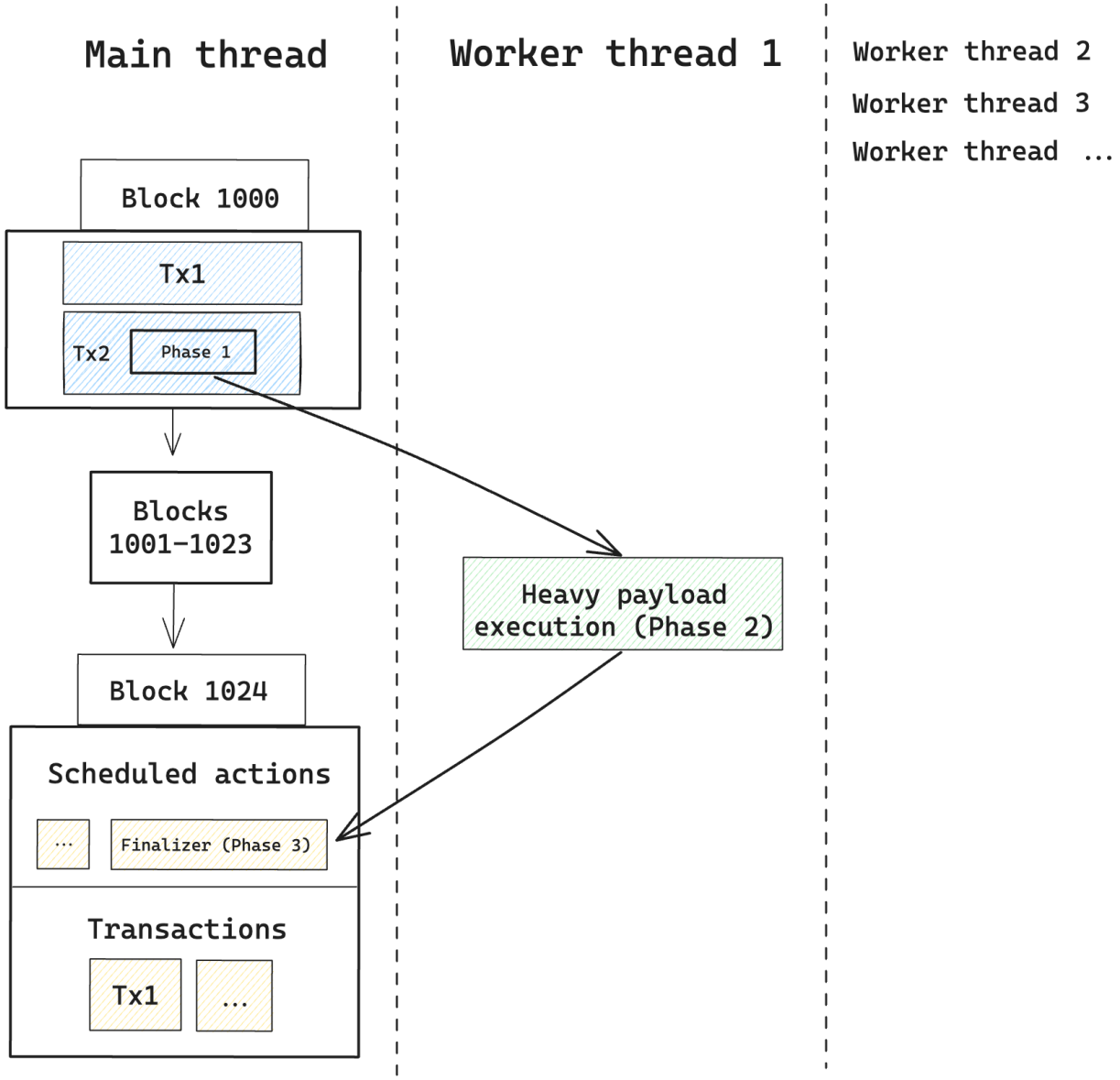

Fig.5 Hybrid execution by phases

3.4.2.1 Comparison with parallel and sequential execution

When comparing sequential execution with hybrid execution, it is important to acknowledge that hybrid execution is an extension of sequential one, allowing the use of parallelism to execute heavy payloads.

It is still possible (and recommended) to build smart contracts in the sequential way (by just avoiding calls to the hybrid execution API), as finality time of transaction executed in a hybrid way is higher than for sequentially executed one, due to the need to wait until scheduled block will be produced.

Where hybrid execution demonstrates its strength over sequential one is execution of actually heavy (and thus, gas-costly) payloads, like zero knowledge math operations, heavy cryptographic algorithms, simulations of scenarios with depth, processing of layer 3 transactions in case of fraud proof, etc.

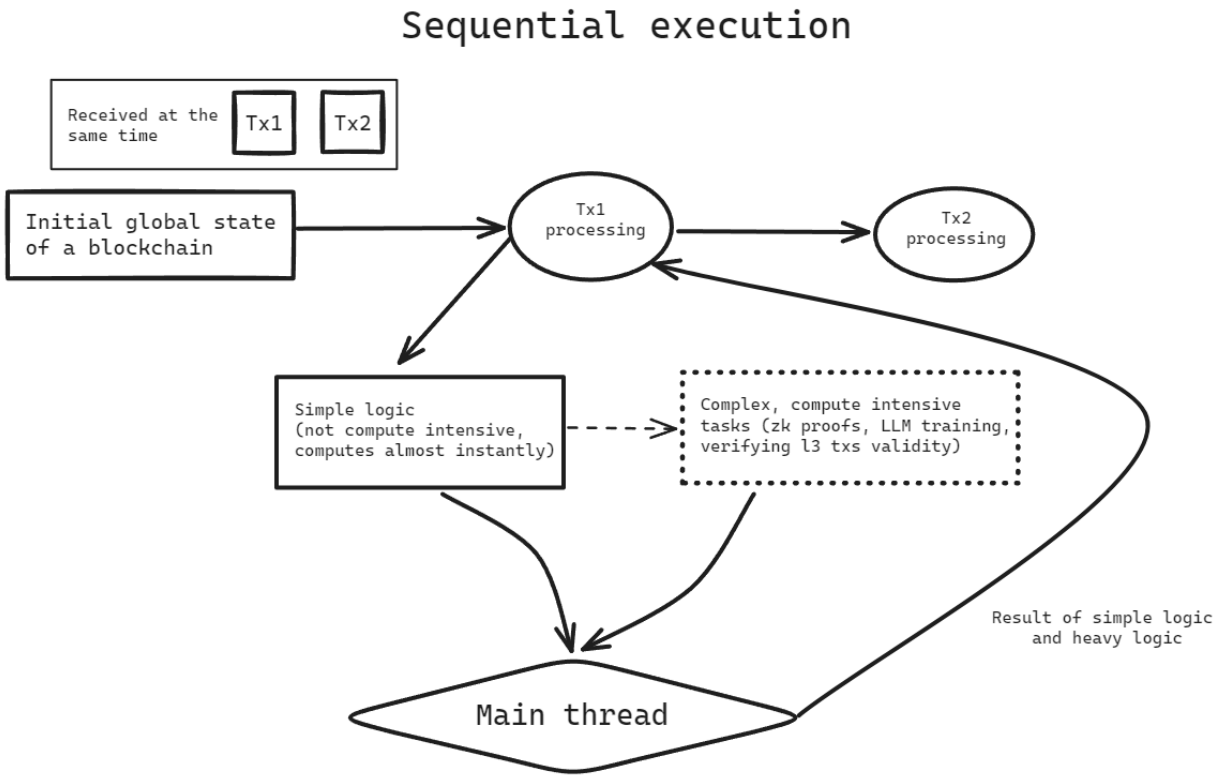

Fig.6 Sequential execution overview

Parallel execution[10], however, is on the other side of the execution parallelization spectrum, attempting to achieve parallel execution of all transactions, not just ones with heavy computation involved.

In theory, the parallelization of simple transactions such as transfers and swaps may appear advantageous. However, in practice, the execution of such transactions is not computationally intensive, with the primary limiting factor being internet bandwidth.

Consequently, parallelizing these transactions would incur the overhead associated with parallelization without providing a significant performance improvement for the blockchain in processing terms.

Parallel Execution

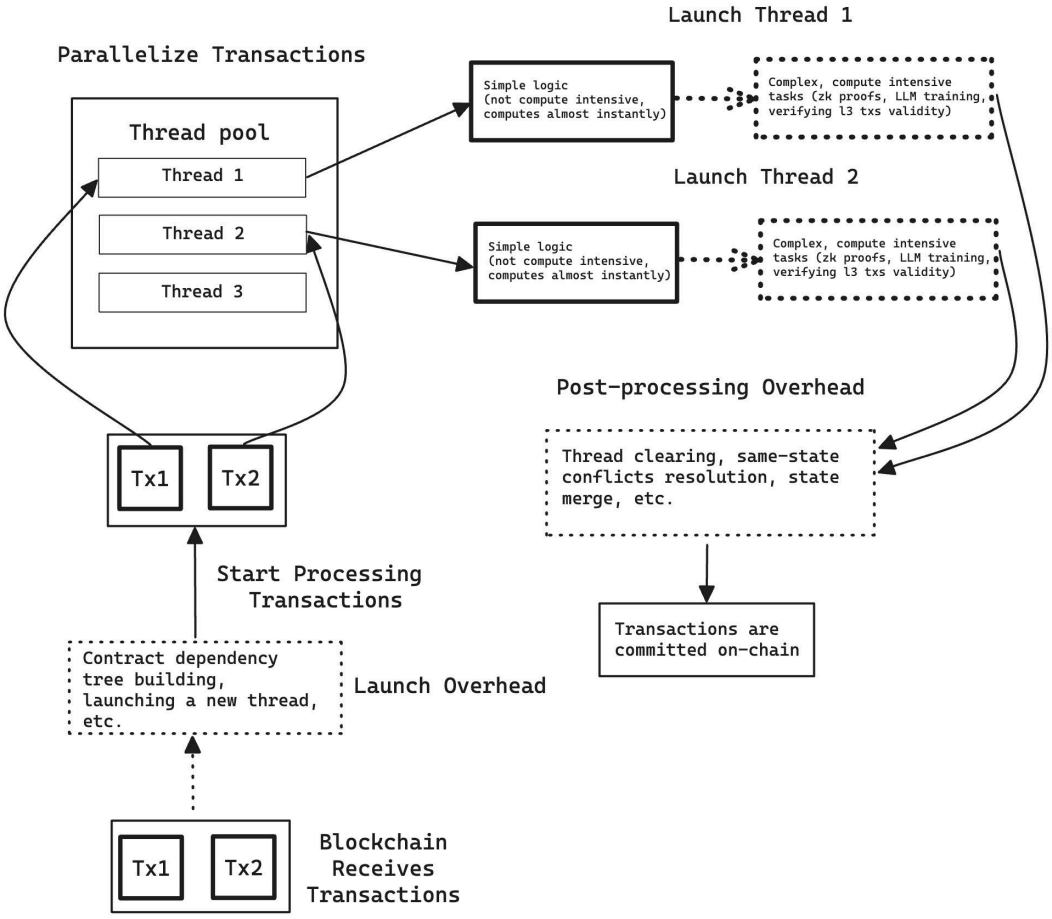

Fig.7 Parallel execution overview

The primary advantage of hybrid execution in comparison to parallel execution is that parallelization occurs on an as-needed basis, and in areas where it has a significant impact, allowing to avoid launch/post processing overhead, not overcomplicating contract flow while still giving smart contract developers powerful framework for effective on-chain execution of heavy operations.

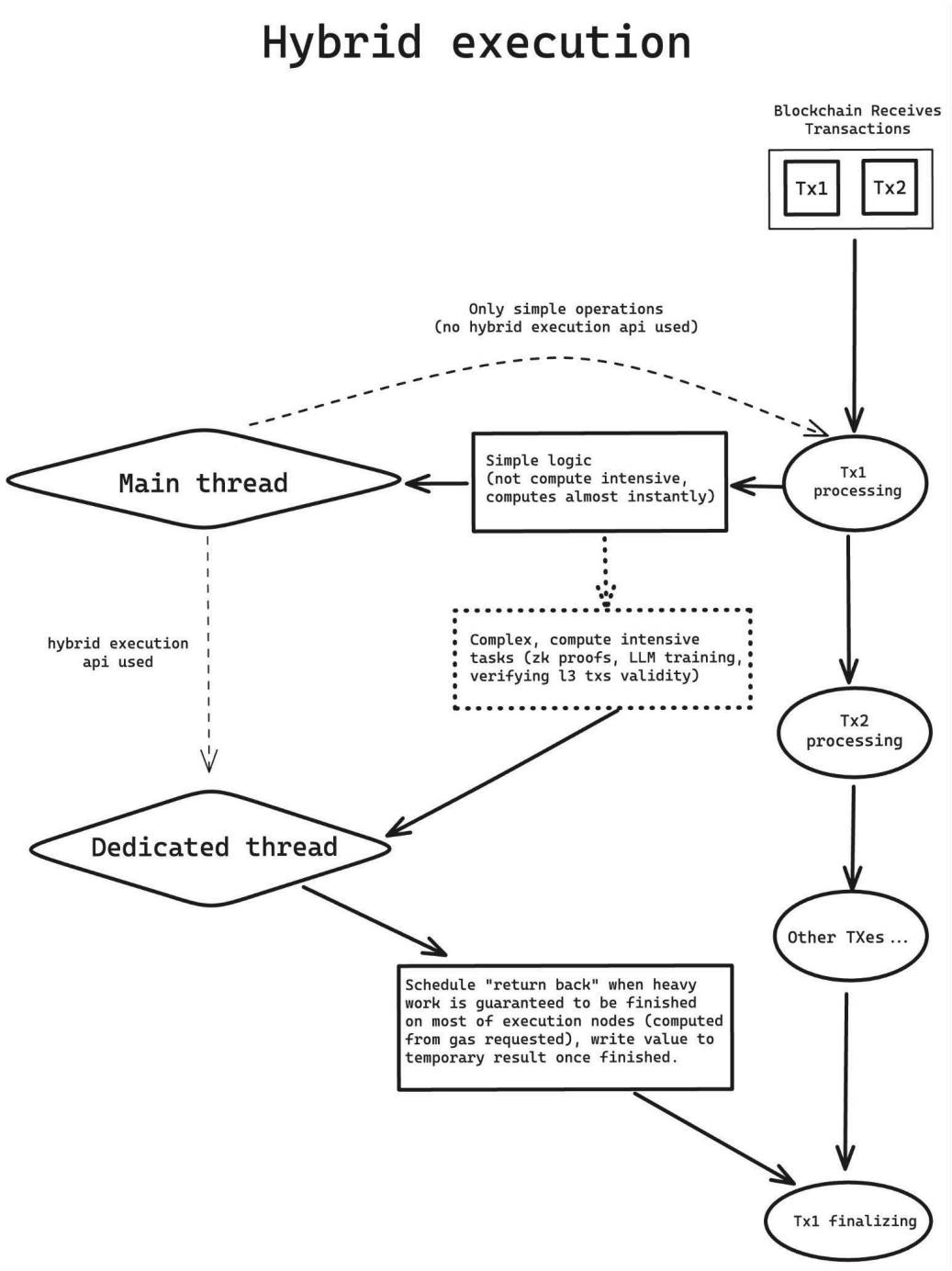

Fig.8 Hybrid execution overview

3.5 VDF-based time measuring transactions

Due to finality being independent of block height in the highlayer system, the speed of block time is not as critical.

Considering the hashes of bundles posted by the sequencer as “l2 blocks”, the substantial interval between new bundles posted to the Bitcoin blockchain renders it impractical as a time measurement scheme for smart contracts.

One potential solution to this issue could involve the sequencer posting timestamps to transactions. However, this solution is centralized and may pose a security threat to time-sensitive contracts (e.g., vesting contracts, liquidity locks, DAO staking) in the event of a compromised sequencer.

To address this challenge, we propose a unique type of transaction: the Clock transaction.

The clock transaction mechanism adapts a Verifiable Delay Function (VDF) with an ***x***-second delay to ensure that this transaction type can only be submitted once every ***x*** seconds.

VDFs are very similar to PoW, however they are not parallelizable or parallelization does not give significant boost to the time it can be computed for. This makes sure that even though multiple nodes might compute VDF at the same time, they will get results at approximately the same time, producing “tick” predictably.

To ensure that the Clock transaction is not significantly slower than ***x***, the submission of such a transaction may be incentivized, making it attractive to run a network node and earn this "clock tick" (VDF computing) reward. This approach significantly reduces the likelihood of a clock tick being delayed for a duration that would matter in the context of smart contracts. Additionally, it strengthens data storage and availability guarantees by indirectly incentivizing the operation of an l2 node.

Number of linked clock transactions (ticks) will be globally available for the rest of transactions, making it possible to use the number of ticks to measure time in contracts, tie scheduled actions and hybrid execution finalizers to tick number, not block number, optimizing it by redistributing computation more consistency and precisely.

Wesolowski VDF[11] will be used in reference implementation.

4 Additional Concepts

This section focuses on Highlayer elements that, while not part of the Highlayer project's core structure (l2 node/sequencer/core software), are central to our broader objectives and understanding. Identified through our exploration and development efforts, these elements are crucial for a comprehensive appreciation of what the Highlayer project seeks to accomplish. Our aim is to highlight the importance of these concepts, demonstrating their essential role in the project's wider context. By integrating an understanding of these elements, we provide a complete, more nuanced picture of the project, ensuring a thorough understanding of its scope and development abilities available on it.

4.1 Subaccounts

Note: This is a short representation of "Cryptographic Subkeys in Decentralized Applications"[12]. Please refer to it for the complete specification. The use of this proposal is optional; it only represents our view and direction on enhancing the user experience in dApps.

The Subaccounts is a UX protocol that allows decentralized applications ("dApps") built on Highlayer to have a smooth user experience by use of delegated keys. The use of subaccounts allows dApps to create transactions without explicit user approval, while keeping wallets of users fully secure.

Subaccounts can be utilized for seamlessly closing trades on decentralized exchanges, posting in decentralized social networks, playing on-chain games or any other high-interactivity use case.

In the Subaccounts protocol, dApp user utilizes different wallet for each dApp, with dApp frontend having full control over the wallet that user delegated to it, allowing to bypass the need for interacting with wallet provider unless monetary deposit is needed (as funds remain in main wallet until they are deposited to dApp's subaccount).

Subaccounts are created by using hash of signature as seed for deterministically generating a unique keypair, then uploading public key to announcement channel (Highlayer smart contract) to create a publicly available link between subaccount and main account. To recover the subaccount when having user wallet (provider) access and an dApp name, the same re-derivation process is needed.

Pseudocode Example:

```
function createSubkey(masterPublicKey):
    seed = createSeedWithMainKey(appName)
    subkeyPair = generateKeyPair(seed)
    announceSubkey(appName, subkeyPair.public)

function recoverSubkey(encryptedSubkey, masterPrivateKey):
    seed = createSeedWithMainKey(appName)
    subkeyPair = generateKeyPair(seed)
```

Code example 2, subaccount management

5 Conclusion

This paper presents Highlayer, a layer 2 blockchain architecture aimed at improving scalability and efficiency while maintaining the principles of security, decentralization, and resistance to censorship. By analyzing the limitations of existing blockchain scalability solutions, we've developed Highlayer with specific mechanisms for enhanced data availability, transaction finality, and a virtual machine that supports accessible development and efficient execution.

Highlayer's design integrates with Bitcoin's settlement layer, offering a layered solution that enhances transaction throughput without compromising the blockchain's censorship resistance. It operates on a peer-to-peer network of nodes that maintain a secure and coherent transaction history, showcasing the system's resilience and adaptability.

In essence, Highlayer contributes to the blockchain field by addressing scalability and efficiency issues, while enhancing developer accessibility and not compromising on censorship-resistance.
It offers a framework for developing fast, secure, user-friendly decentralized applications, crucial for transitioning the web into its next iteration, web 3.0.

References

1. “Bitcoin: A Peer-to-Peer Electronic Cash System”: https://bitcoin.org/bitcoin.pdf
2. “The Inside Story of the CryptoKitties Congestion Crisis”: https://consensys.io/blog/the-inside-story-of-the-cryptokitties-congestion-crisis
3. “Building Blocks of Sharding Blockchain Systems: Concepts, Approaches, and Open Problems”: https://ar5iv.labs.arxiv.org/html/2102.13364
4. “Optimistic rollups”: https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/
5. “zkRollup”: “https://github.com/iden3/rollup/blob/testnet/README.md”
6. “An introduction to sovereign rollups”: https://celestia.org/learn/sovereign-rollups/an-introduction/
7. “Bitcoin Inscriptions & Ordinals”: https://www.galaxy.com/insights/research/bitcoin-ordinals-inscriptions-5-billion-nft-market/
8. “The Sequencer and Censorship Resistance”: https://docs.arbitrum.io/sequencer
9. “Duktape”: https://duktape.org/
10. “Parallel and Asynchronous Smart Contract Execution”: https://arxiv.org/pdf/2306.05007.pdf
11. “Efficient verifiable delay functions”: https://eprint.iacr.org/2018/623.pdf
12. “Cryptographic subkeys in dApps”: https://github.com/angrymouse/papers/blob/main/subaccounts.md
13. “EIP-4844: Shard Blob Transactions”: https://eips.ethereum.org/EIPS/eip-4844