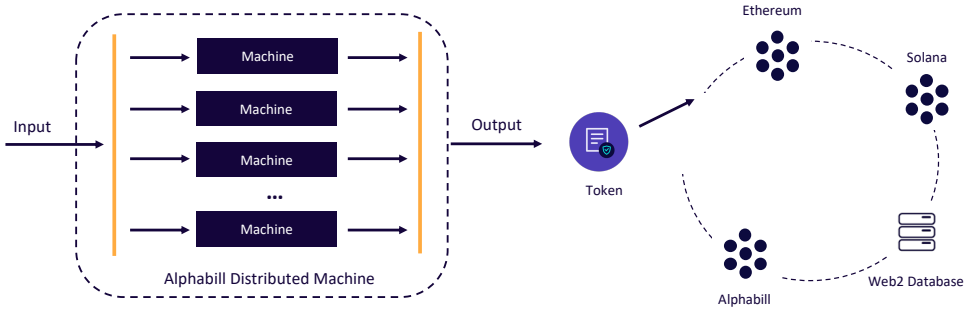
Alphabill

Public Token Infrastructure

TL;DR: Do I really need another blockchain?

Alphabill is brought to you by the team behind **Guardtime**, a team of researchers developing blockchain protocols since before Bitcoin. It is a new public, permissionless blockchain design that elevates tokens to first class citizen status. Tokens are not trapped in smart contracts[1] but free to move across Web2 and Web3 as programmable, autonomous data objects that can be assigned digital property rights.



- Alphabill is designed to have sufficient throughput to tokenize all human and machine generated content on the Internet, with throughput **several orders of magnitude higher** than existing blockchain designs. This is achieved by using **bills** as transaction units (similar to physical cash). Alphabill is the first blockchain to be built using bills i.e. not UTXOs, not accounts.
- Single token programmability is implemented in WebAssembly using "**predicates**", similar to Bitcoin locking scripts but with rich statefulness and robust programmability. An object-oriented programming model is used with inheritance potentially many levels deep, enabling a rich ontology of token types.
- Multi-token programmability (such as Automated Market Makers(AMMs)) can be implemented using a native **EVM,** but any smart contract platform can potentially be used.
- The blockchain can be decomposed into blockchains for individual tokens which can be verified with **zero trust**. This is similar to physical cash – you care about the money in your wallet, not that of anyone else.
- Apart from short term spikes **fees** are low, deterministic, and **independent of throughput** i.e. no congestion.
- Alphabill is a Delegated Proof of Stake Network. Massive decentralization is achieved by **stateless validation**. Anyone with consumer accessible hardware can participate in validating blocks and immediately earn rewards without needing to sync the chain.
- Alphabill enables **offline** transactions for ALPHA, its native currency as well as other crypto assets through cross chain interoperability. In an environment with no network connectivity a payer can irrevocably make a transfer that can be verified with zero trust assumptions
- The major **tradeoff** is that there is no shared global state – flash loans, for example would be impossible on Alphabill.

[1] Tokens are not stored as variables inside smart contracts but instead are allocated space directly on the state tree enabling them to be extracted and verified off-chain.

1 Introduction: A Platform for Digital Property Rights

The Internet, possibly the most important invention of the 20th century, was designed to be an open permissionless network that anyone could access. This freedom led to an explosion of creativity as humans became more connected during the 1990s. Fast forward to 2024 and we see that the original democratic design has been hijacked by corporate gatekeepers who can censor access at a whim, abusing their users and otherwise tax innovation. First the corporate platform owners do everything they can to recruit users; then they abuse those users to make things better for their business customers. Finally, they abuse those business customers to keep all the value for themselves.

The underlying motivation of the Web3 community is to return the Internet to its permissionless roots: a) giving back users control over their data, b) democratizing ownership by providing users governance and economic rights in the platforms and applications to which they contribute and c) preventing corporate gatekeepers maximizing profits at the expense of users and partners.

For blockchain to deliver on this vision a chasm of qualities must be crossed, to provide a performant, secure and user-friendly experience. Alphabill is a new blockchain design that attempts to do this at an industrial scale.

A key contribution is to elevate *tokens* to first class citizen status. Alphabill tokens are programmable and verifiably unique authenticated data structures, i.e. they are not locked into a single blockchain. Instead, they are portable, autonomous data objects that be assigned digital property rights and traverse the Internet, whether decentralized Web3 smart contracts or centralized Web2 applications. At each hop they can be verified and acted upon without a trusted intermediary.

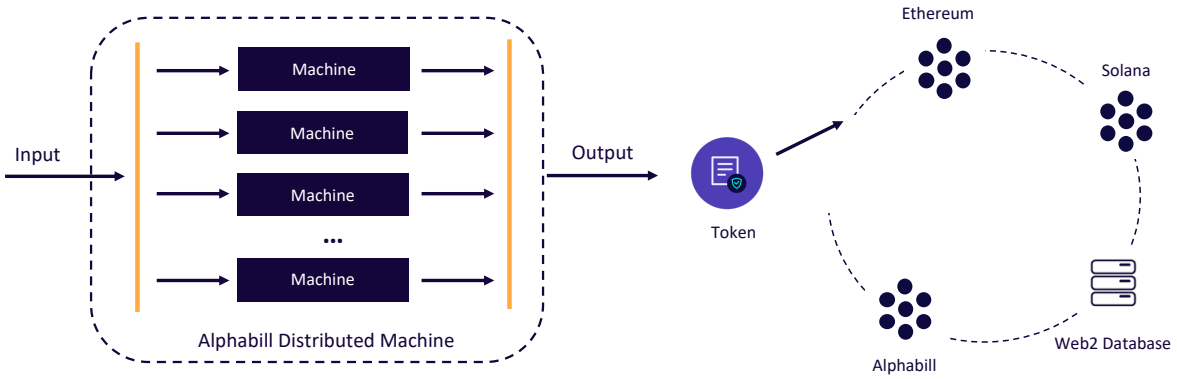


Figure 1. Tokens, created in Alphabill are portable across Web2 and Web3

Tokenizing all human and machine content, i.e. potentially billions of data objects per second, is far beyond the capacity of current blockchain architectures and improvements of several orders of magnitude are needed. In this paper we introduce a series of innovations: bills (not UTXOs, not accounts) as transaction units, state tree recursion, stateless validation, a new consensus protocol and a new computational model, all of which combine to enable an industrial scale design.

This assignment of digital rights at massive scale would satisfy the needs of global financial and property ownership systems. It would also enable the integration of a financial incentive backbone into the logic of next generation applications, and it would enable the zero-trust verification of all data on the Internet.

This introduction is optimized for explanatory clarity. Technically precise descriptions, academic papers, security proofs and specifications are available at www.alphabill.org.

2 Design Goals

2.1 Linear Scale Production

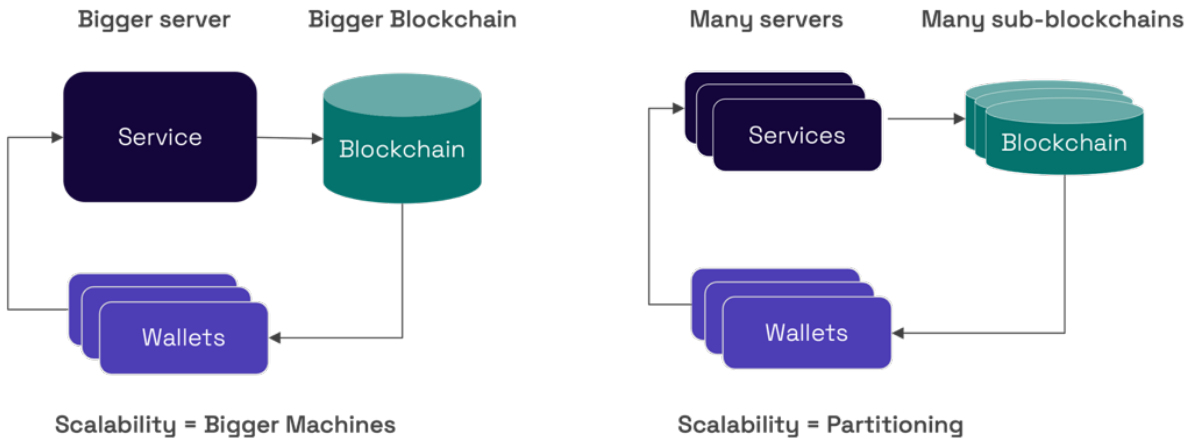


Figure 2. Scaling through more powerful machines or many machines

There are two ways to scale a system; a) “vertical scaling”, using a more powerful machine or b) “horizontal scaling” using parallel decomposition such that the work is split between many machines. Developers are attempting to use both approaches to scale blockchains today. Solana is an example of the former where a single powerful “leader” machine orders and validates transactions and other machines confirm the results. This approach has a hard limit on scalability (the power of a single machine) and limits censorship resistance through decentralization as the computational requirements limit accessibility.

Horizontal scaling is attempted by numerous layer one protocols (Polkadot, Cosmos, ICP, AVAX, Near, etc.) however these resemble a system of federated chains with separate consensus instances and additional chains, called relays, beacons, hubs etc. providing cross-chain settlement.

A design goal of Alphabill is to have practical unlimited throughput, i.e. sufficient to tokenize all human and machine generated data on the Internet, with a single consensus instance that provides linear scalability, without sacrificing security, decentralization, or performance.

2.2 Linear Scale Verification

Scalability of verification is as important as scalability of applications for real-world applications. In all existing protocols, to *independently* verify a single transaction a large set of related transactions must be verified. Due to this requirement, many users choose to sacrifice the benefits of decentralization by using "light clients" which rely on trusted intermediaries.

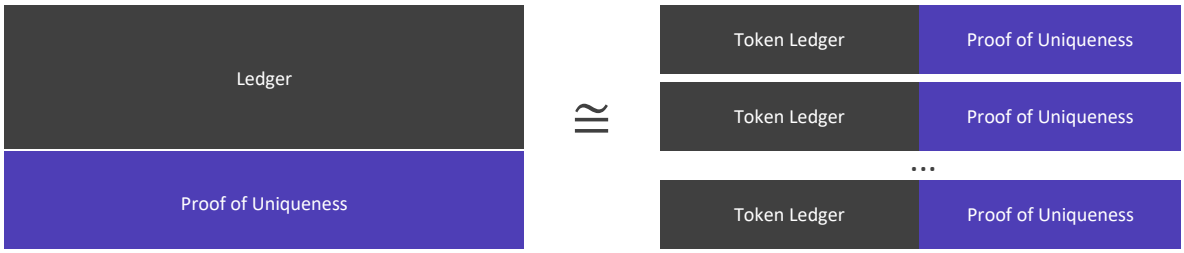


Figure 3. Ledger decomposability

The above figure on the left is the current model of blockchains. There is a single Proof of Uniqueness (whether generated by Proof of Work, Proof of Stake or some other mechanism) for the ledger created once per block.

A design goal of Alphabill is parallel decomposition, i.e., each token in the system (either Alphabill native currency tokens or the equivalent of ERC20 user generated tokens) can be independently updated and verified in parallel.

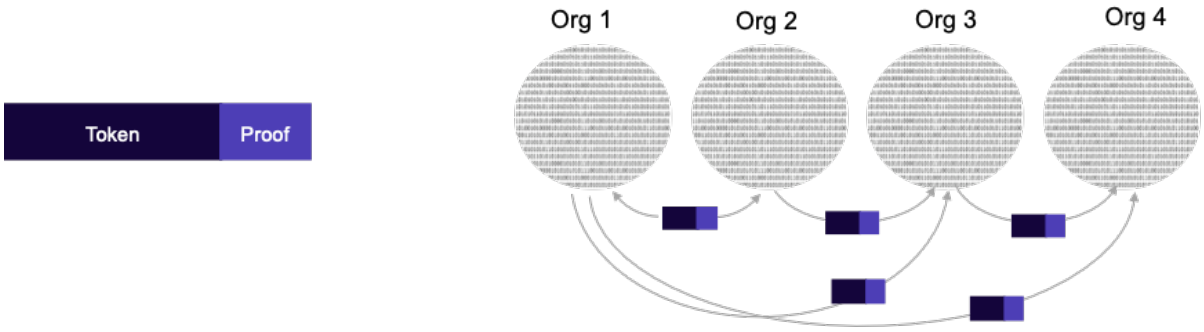


Figure 4. Tokens are portable and verifiable off-chain or off-line.

2.3 Fast Deterministic Settlement Finality

In Alphabill, authenticated data structures known as certificates provide proofs (of uniqueness, unit state, transaction execution). These certificates are created on-chain and potentially used off-chain. Probabilistic finality, such as used in Directed Acyclic Graphs (DAGs), has the potential to reorganize the blockchain which would invalidate a certificate issued during the reorganization. As this would be problematic for off-chain users relying on a certificate, a design requirement for Alphabill is deterministic finality.

2.4 Censorship and Attack Resistance

As has been seen during the Web2 era a single platform gatekeeper can limit access, change pricing at a whim and in general prevent permissionless innovation, where developers are free to develop on the platform with a certain knowledge of the future terms and conditions. A design goal of Alphabill is that there are no gatekeepers. i.e. there is no entity, whether a government, organization, or individual, can exert control over the machine, such as controlling access, set fees or otherwise prevent the machine from continuing to operate.

Alphabill achieves censorship resistance through decentralization. To ensure widespread accessibility as design goal is that the distributed machine should function with consumer accessible hardware. It should also continue to function even under the most adversarial conditions.

2.5 Deterministic Fees and Zero Extractable Value

A design goal of Alphabill is Zero Extractable Value i.e. validators should be economically incentivized to participate in processing transactions but have no agency to decide the price, order, or number of transactions to be processed. As such a uniform gas price should be determined by a decentralized governance process to ensure fees remain as low as possible with the constraint that there are sufficient incentives to ensure an optimal number and diversity of validators to participate in the network.

Linear scale production should guarantee, apart from short term spikes, that long term supply (of computational power) can always match demand. If congestion can be eliminated, then there should be no need to have a marketplace for fees. As demand increases the network can add more computational power, with additional machines operating in parallel to process transactions.

3 Theory of Blockchain Decomposition

Alphabill is based on a theory of blockchain decomposition[2]. This theory can guarantee that there are no bottlenecks and computational resources can be added indefinitely – the system scales linearly (production and verification) without sacrificing security, performance, or decentralization.

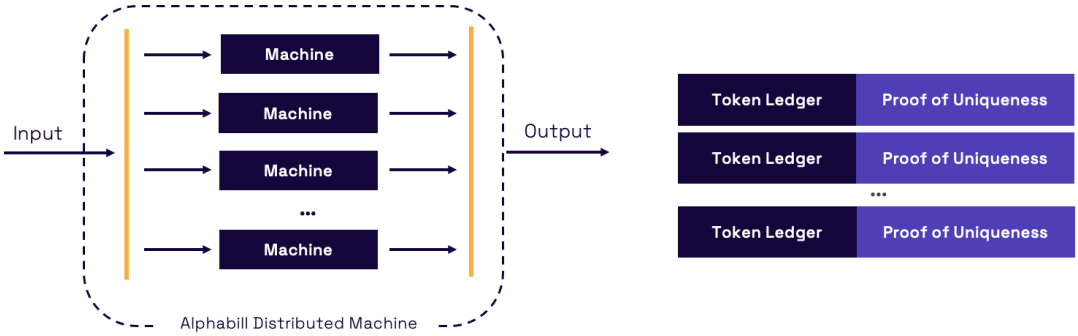


Figure 5. Parallel Decomposition

To implement a decomposable blockchain Alphabill introduces five innovations:

1. **A New Transaction Unit:** UTXOs and accounts do not allow parallel decomposition. The Alphabill blockchain is based on bills (similar to physical cash bills). This allows for the parallel updates and verifications of tokens minted on the state tree. Transaction units are described in section 4.
2. **State Tree Recursion:** The state tree is built recursively allowing for independently verifiable blockchains to be extracted for each token. This allows tokens minted on the blockchain to be verifiable and actionable off-chain in the real world. State tree recursion is described in section 6.4.
3. **A New Consensus Protocol:** Alphabill has a single consensus instance across the network such that deterministic finality is achieved across the network within one block. The consensus protocol is described in section 8.

[2] https://doi.org/10.36227/techrxiv.14994558

4. **A New Computational Model:** The computational model on Alphabill consists of two components, predicates, or unlocking conditions, for computation that requires only single token inputs and smart contracts for computation that requires multiple token inputs. Predicates and smart contracts are described in section 7.
5. **Stateless Validation:** Validators using consumer accessible hardware can instantly start verifying blocks without needing to synchronize the chain. Stateful and stateless validators are described in section 6.4.

4 Transaction Units

Historically, all blockchains have been designed using UTXOs or accounts as transaction units. As every transaction, by definition, will involve at least two transaction units, the ledger is *interconnected* and the history of each asset in the ledger is dependent on other assets. These choices severely limit the achievement of performance goals. Either the chain gets congested or additional layers, such as rollups or federated consensus instances are introduced which result in compromises in security, performance, settlement finality etc.

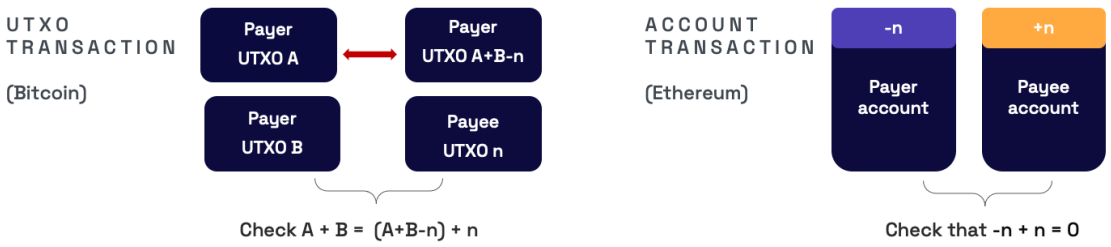


Figure 6. Every transaction with UTXOs and Accounts involves at least two transaction units.

If the accounts or UTXOs are on different machines, then coordination is needed between the machines to atomically execute a transaction. For accounts this is obvious (they are on different machines and the machines need to communicate so that one account balance goes down and the other goes up). The same principle applies to UTXOs. If a user wishes to pay 100 units and has two UTXOs on different machines each worth 50 units then the two UTXOs on different machines need to be marked as spent and a new UTXO needs to be created with a value of 100. This process requires coordination across the different machines.

In Alphabill we use the principle of a bill-based money scheme.

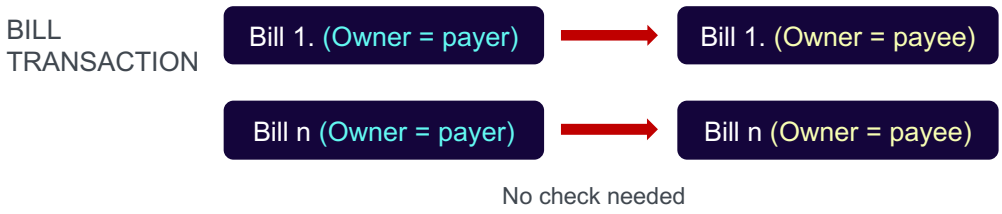


Figure 7. bill-based money schemes can execute transactions in parallel.

In a bill-based money scheme, such as physical cash, the only thing that changes during a transaction is the ownership of the bill. As no checks or coordination is needed, the bills can be on different machines and processed independently in parallel.

Physical cash transactions exhibit perfect parallelism of settlement and verification. Cash transactions can settle independently, and users can independently verify that the cash in their wallet is both available and valid (not-counterfeit). A design goal of Alphabill is to replicate these properties using public blockchain.

Potential limitations of bill schemes are a) atomicity - how to ensure multiple bill transactions are atomic i.e., there is not a situation where only a subset of bills is transferred during a multi-bill payment, and b) precise payments - if a user has a bill worth 100 units how do they make a payment of 50 units?

In the next section we will introduce data structures that allow for atomic precise payments within a single block through state tree splits. This can be achieved in a way that does not break the native parallelism of the bill model.

5 Alphabill State Tree

Historically different blockchain platforms have tried different approaches to represent the state of managed assets. Bitcoin for example stores UTXOs in the *chainstate*, a LevelDB database. Ethereum uses a Merkleized Patricia Trie to represent **accounts** i.e., each leaf of the tree is either an account or a smart contract serialized by its leaf address.

In Alphabill we use a **count-certified authenticated Adelson-Velsky and Landis (AVL)** tree where the nodes of the tree are *units*, which can be bills, tokens or smart contracts. Tokens are first-class citizens in that each token in the system (whether native Alphabill currency tokens or user defined tokens) is serialized by its identifier in the state tree.

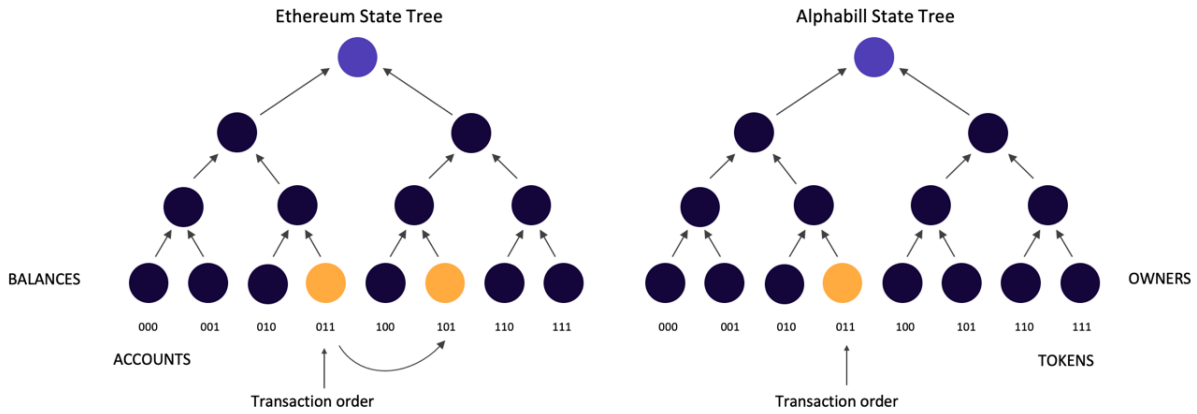


Figure 8. Ethereum uses a state tree of accounts. Alphabill uses a state tree of tokens.

Ethereum uses accounts which are represented as leaves of the state tree. Tokens are created by smart contracts and those tokens exist as variables inside the smart contract. In Alphabill individual tokens are created directly on the state tree and a transaction order in Alphabill will change the ownership of the token.

The main advantage over accounts is sharded parallelism of production and verification i.e. state, transaction, and network sharding become trivial in a bill-based model. In the example above in contrast with the account model where each transaction impacts at least two accounts, a transaction in the bill model updates only a single unit i.e. all tokens are independent – there are no cross dependencies between tokens that require a global ordering of transactions to achieve deterministic execution.

This allows for perfect parallelism i.e., all tokens can be updated and verified independently, and global state can be partitioned and managed by clusters of machines (validators) operating independently in parallel.

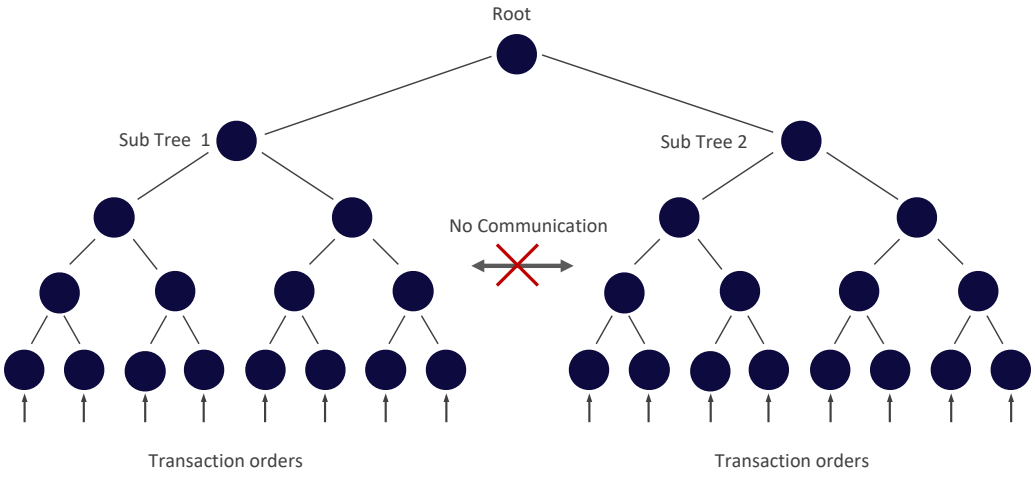


Figure 9. Parallel execution of transaction orders

The above diagram shows that as the throughput increases the state tree can be split into two sub-trees or *shards* with each half of the tree being stored in memory of different machines. The key point is that the machines on different shards do not need to communicate with each other during transaction settlement. A transaction involves only a change in the ownership of a token, validated based on local context: the token's previous state and transaction order only. This implies that the sub-trees can be processed in parallel without any synchronization or coordination with other sub-trees.

To understand scalability of verification and explain how individual ledgers for tokens can be verified in parallel we need to understand how the state tree evolves over time.

5.1 Recursion: State Evolution Over Time

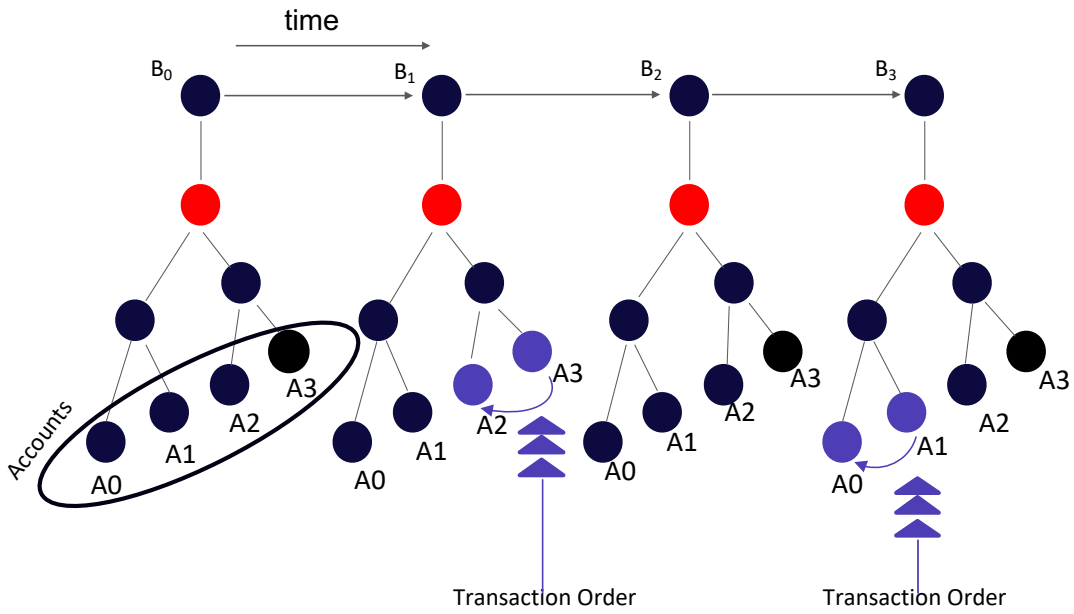


Figure 10. Ethereum state evolution over time.

The above diagram shows the evolution of the state tree for account-based blockchains such as Ethereum. Time moves from left to right and there are 4 blocks with the state tree shown for each block. The state root hash value for each block is shown in red. The leaves of the tree represent four accounts A0 to A3 (the state tree is shown lopsided for reasons which will become clear shortly). In Ethereum the transaction units are accounts, and each transaction will impact at least two accounts.

Every block, transaction orders will be processed which will cause the state tree to be updated. In the figure above we show a transaction order making a transfer from account A3 to A2 in block B_1 and A1 to A0 in block B_3.

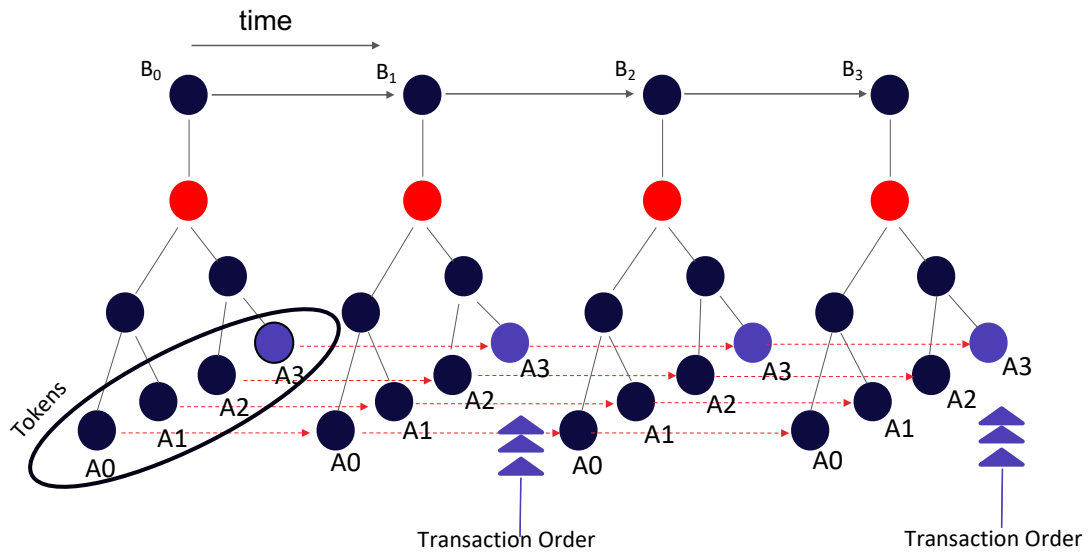


Figure 11. Alphabill state evolution over time.

The figure above shows the evolution over time of the Alphabill state tree. Time moves from left to right and there are 4 blocks with the state tree shown for each block. In the state tree consists of 4 tokens A0 to A3[3]. In Alphabill each transaction will impact a single token.

The key difference from the previous diagram is that when the state tree is built the leaf nodes for each token are cryptographically linked to the leaf nodes of the previous token state (the dotted lines). As all settlement is local and all tokens are independent it is possible to extract the history for an individual token and allow it to be verified without requiring the history of other units. This is shown below for an individual token in the state tree. The blue hash-values are effectively an individual **token blockchain** which can be verified independently with the same security properties as the overall blockchain.

[3] The Alphabill Yellow Paper refers to units. There are three types of units: bill, tokens and smart contracts.

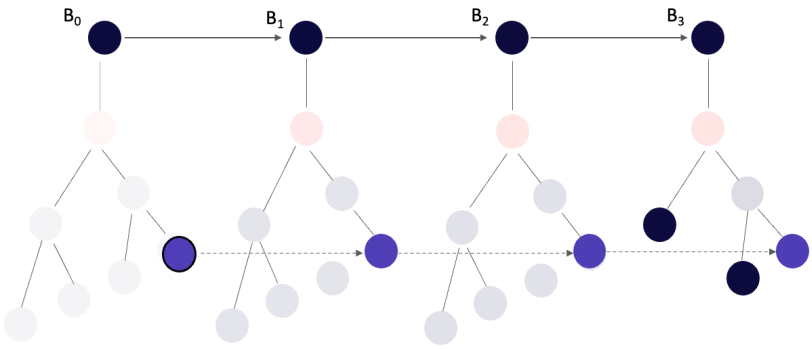


Figure 12. An individual token blockchain, extracted from the main chain.

This ability to decompose the ledger into token sub-ledgers allows a recipient of a transaction to independently verify that they are the new owner of a token without needing the full ledger[4]. This is similar to physical cash. A user cares about the bills in *their* wallet, not those of anyone else. Independent verification can be done on mobile devices, or even offline, without the need to trust third parties as in the case of having to rely on traditional "light clients".

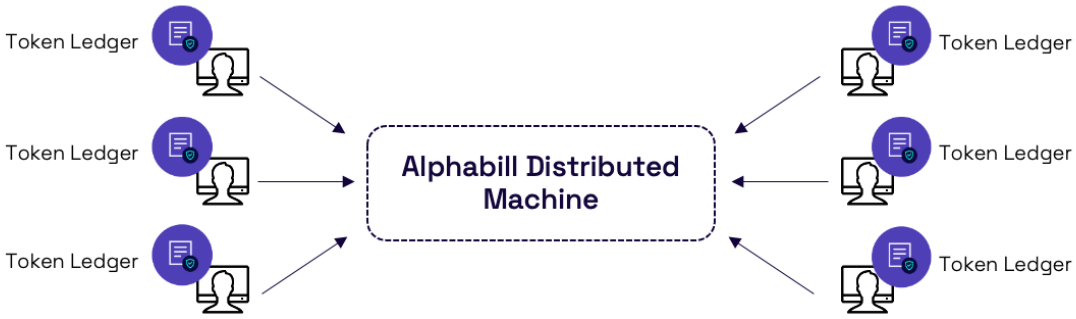


Figure 13. Entire token histories can be stored locally on client-side wallets.

If users can store their tokens off-chain, then a logical conclusion is that no transaction data needs to be stored on-chain at all[5]. A user can store all necessary information (token ledger and their private key) off-chain. To initiate a peer-to-peer transaction they will include a signed transaction order

[4] Technically, what is useful as a proof of payment is a "proof of transaction execution" which applies to a single transaction only, without the full transaction and token history.

[5] There still needs to be a state tree leaf hash created to allow the validators to verify that token ledger received as part of a transaction order was minted and is the **up-to-date** version.

along with the token ledger, both of which will be sent to the Alphabill Distributed Machine, **which can now validate the transaction *statelessly***. This approach gets as close as possible to the properties of physical cash – physical cash bills are tokens which are self-verifiable and be passed from party to party.

Off-line transactions

Under certain conditions the system can be extended to support transfer of tokens with final, irrevocable payments even in the absence of network connectivity. A payer who wishes to transfer tokens offline to a known payee (say a subway operator) can lock[6] the token which can then be unlocked either a) after a specified period of time (say one week) by an arbitrary transaction from the payer, or b) by a transaction order from the payer, where the recipient can only be the known payee. In an environment with no network connectivity the payer can then generate a transaction order for a specified amount for the payee and digitally transfer the transaction order without network connectivity to the payee, who can independently and without trusted intermediaries, mathematically verify that only they can unlock the token and claim ownership prior to the timeout period. The only information the payee requires to verify the validity of the transaction is the genesis block (B_0 in the above figures)

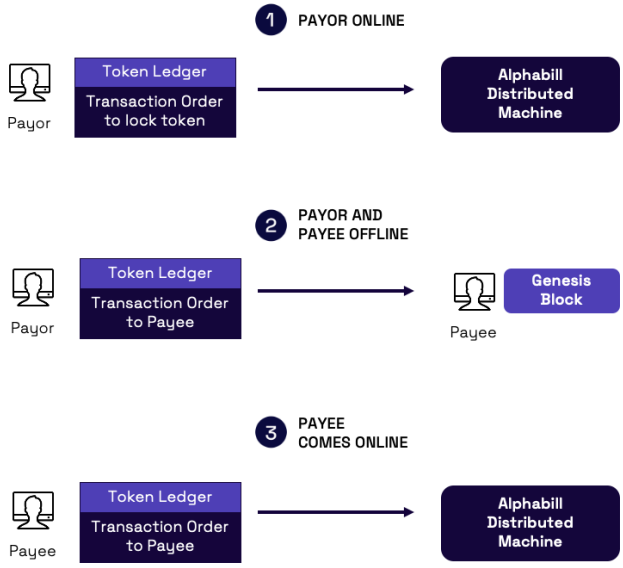


Figure 14. Process for implementing off-line transactions.

[6] Technically they will install a predicate – see section 7.1

Once the payee has connectivity the payee will send the received transaction order to the Alphabill Distributed Machine and claim unconditional ownership of the tokens.

5.2 Certificates

A certificate is an authenticated data structure used within Alphabill that contains elements that enable an interested party to verify proofs such as proof of uniqueness, proof of ownership, proof of transfer etc. The design of the Alphabill state tree allows for different types of certificates to be created.

- Unicity certificates — proof that the ledger, as a whole and its components, are unique and passed validation.
- Transaction execution certificates — a proof that a transaction t is in the block B of the blockchain.
- Unit certificates — a proof that a token has certain attributes, for example ownership, in the state tree.

5.3 State Tree Splits

It would be inefficient to have to pre-allocate space in the state tree for every possible token. For example, the equivalent of an ERC20 smart contract may require the issuance of billions of tokens. Fortunately, creating state space for all individual tokens is not necessary as, upon issuance, there are a limited number of owners. In this section we show how the state tree can be expanded and contracted through transaction orders.

Consider the equivalent of an ERC20 token called AToken, the issuance of which will be 100 billion tokens. Initially, there will be a single owner (the issuer) who will own all 100 billion tokens.

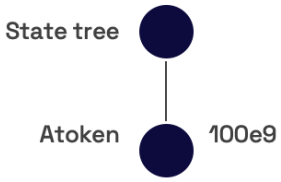


Figure 15. User created token at issuance.

When a transfer of one token occurs to a new owner the tree is split and the original token is replaced with a node with child leaves, which represent the new fractions of the token.

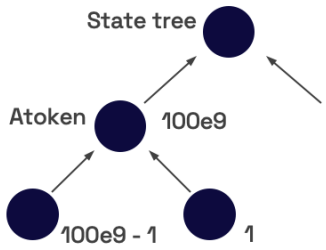


Figure 16. User created token after first split.

The Alphabill AVL tree is count certified so that each parent node maintains the total sum of created parts.

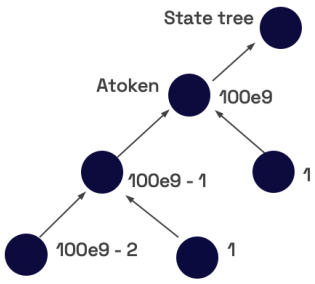


Figure 17. User created token after second split.

After a second transfer of one token the state tree looks like above[7].

The same bill-splitting principle can be applied to any fungible token. For example, the figure below shows a user who wishes to make a payment of 34 cents using a "Alpha-USD" stable coin but only has one single Alpha-USD dollar.

[7] In reality the AVL tree is self-balancing. For a precise description please see Alphabill yellow paper.

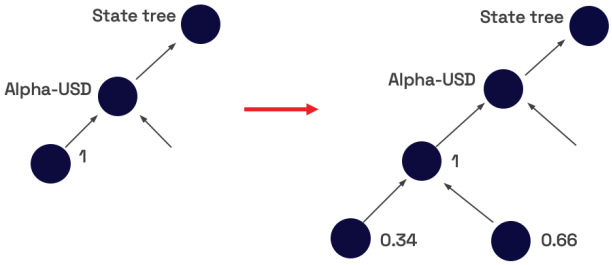


Figure 18. Precise payment of 0.34 Alpha-USD

Allowing for splits in this way also allows for precise atomic payments within a single block. Joins are also possible however they are not available as part of transaction orders. Instead, a user may make a “dust collection” swap request such that small value tokens may be consolidated into a single token. The dust collection procedure is executed independently during the creation of a new block.

Note that the state tree splits and joins do not break the parallelism as all splits and joins can be guaranteed to be local to the machine in which the split or join occurs.

5.4 Double Spending

Double spending is impossible by design. Each unit has a unique address and there can only be one proof of uniqueness per round, assuming the hash function used is collision resistant.

6 Distributed Machine Architecture

The Alphabill Distributed Machine operates one large state tree distributed across a network of redundant validator nodes in a modular fashion. i.e. the state tree is subdivided into partitions which operate sub-trees of the overall state tree. Partitions share a common framework of unicity certification, implemented by the Root Chain, a Delegated Proof of Stake Network. **The system is decentralized and permissionless due to on-chain Delegated Proof of Stake mechanisms controlling the operational aspects of the network.**

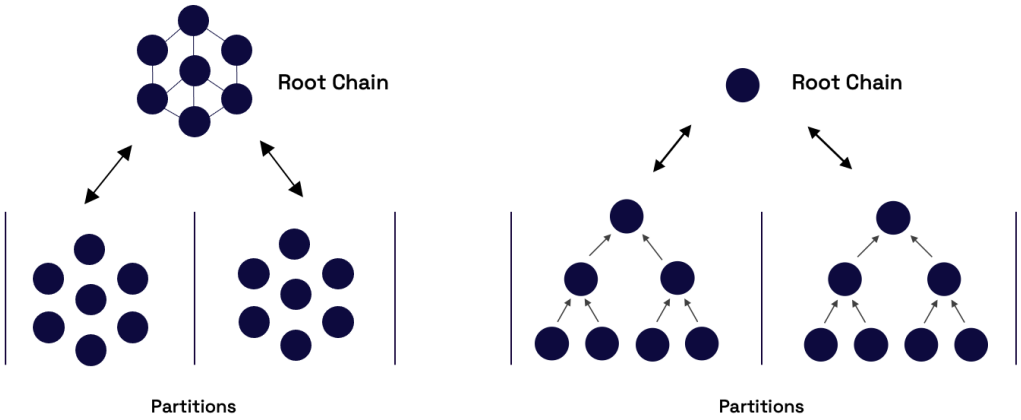


Figure 19. Network (validators) and state tree views of the Alphabill machine.

The left-hand side in the above diagram shows the component machines in the network. The right-hand the state tree view i.e. the data structures managed by respective validators. Each partition is a sub-tree in the overall state tree. The depicted subtree of every partition is replicated across all validators of the respective partition. Note we show the machines in the Root Chain connected (the solid lines) and not in the Partitions to indicate that there is a single consensus instance, maintaining the partition-wide synchrony of each individual partition.

6.1 Root Chain

The Root Chain is a Delegated Proof of Stake network that provides:

- network orchestration providing randomness and timing references (liveness).
- **enforcement of the safety property (no double-spending, no parallel histories).**
- **enforcement of aggregate transaction system rules.**
- **enforcement of partition-level consensus.**
- proofs of uniqueness (unicity certificates).

The Root Chain does not store transaction data or validate transactions. Its primary purpose is to create Unicity Certificates which are passed down from the Root Chain to each partition below and stored in blocks. The Unicity Certificate includes the group signature generated by the validators in the Root Chain after completing all necessary consistency checks. These checks will include whether the validators in a partition are in coherence, whether they extend the previously finalized block **exactly once**, and aggregate checks for the specific transaction system in each partition (for example, in currency partitions the money invariance is also checked).

The Unicity Certificate framework allows a common root of trust across partitions i.e. proofs generated in one partition can be verified in another as they share the same root of trust.

6.2 Partitions

Partitions are sub-trees of the global state tree split according to function. System-defined partitions include the Governance Partition, Alphabill Native Currency Partition, Atomicity Partition, User Token Partition and the EVM Partition. Section 10 onwards describes the functionality of each partition.

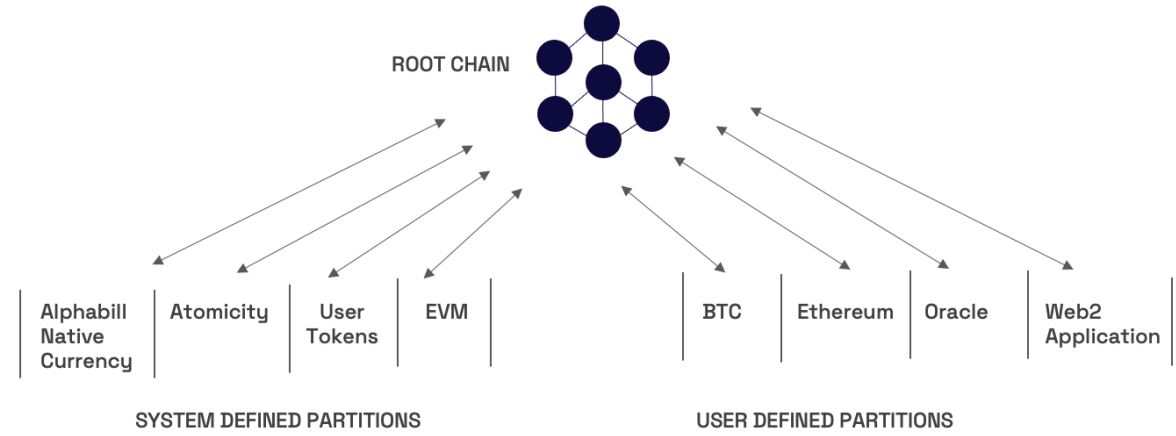


Figure 20. Alphabill Partitions showing system defined and user defined Partitions. User defined Partitions can be added in a permissionless way.

Each Partition consists of one or multiple shards.

6.3 Shards

Partitions start with a single shard **comprised of** a set of validators all of which share the same validation rules. A shard provides bulk transaction validation, state-keeping, ledger handling and smart contract execution.

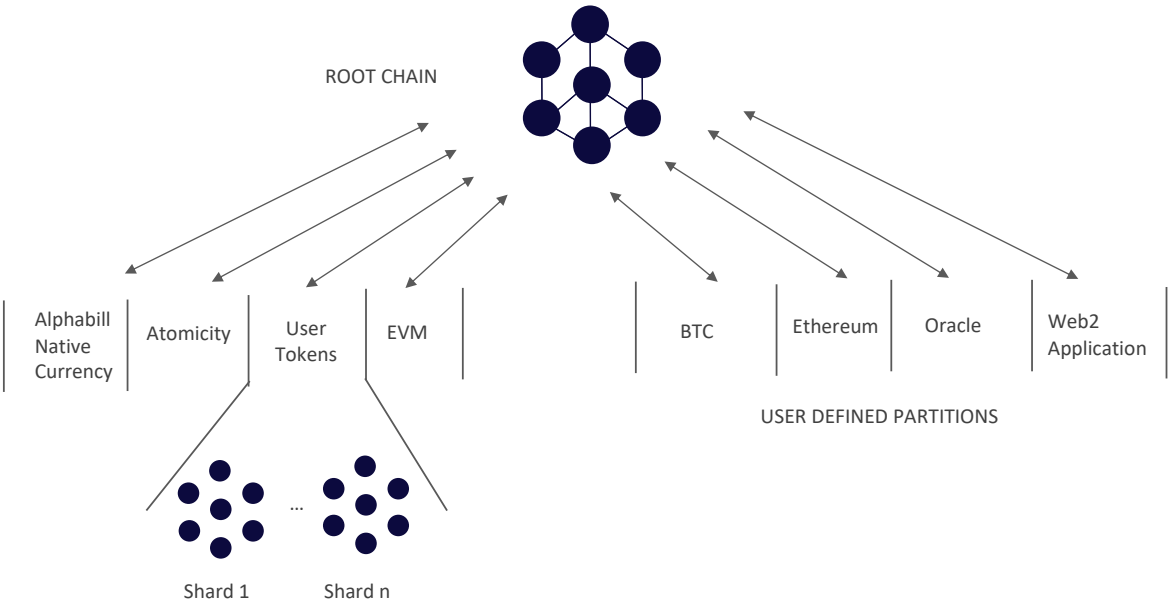


Figure 21. Alphabill Partitions showing a multi-shard User Token Partition

As capacity grows within a partition and the validators approach capacity[8] additional shards can be added by splitting the state tree and the sub-tree splits managed by a different set of validators.

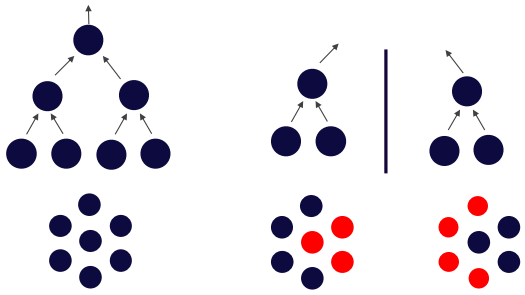


Figure 22. A partition before and after a shard split from 1 to 2 shards, each with 7 validators.

The figure above shows how the shard state tree is split amongst the existing validators (the black dots). New validators (the red dots) join each shard to ensure that the number of validators per shard stays within defined limits.

The state tree can now continue to grow in each shard until it reaches capacity again at which point another split can occur.

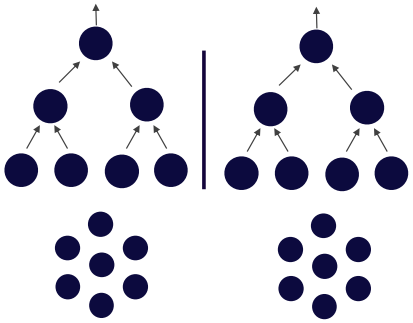


Figure 23. The two shards can now expand their sub trees to meet the increased demand.

Validator machines within a shard participate in a single consensus instance in coordination with validators on the Root Chain – each shard does not have a separate consensus instance.

[8] Constraints on capacity include the memory constraints due to the size of the state tree, computational power to process transaction orders and network bandwidth.

6.4 Stateful and Stateless Validators

Stateful validators are the machines that store the state tree in memory and update the tree in accordance with ledger rules and transaction orders. Stateful validators that join the network need to synchronize to the current state of the blockchain before they can propose and validate blocks. Decomposition theory makes it efficient to have stateless validators also participate in validating blocks. These validators do not need to synchronize the chain i.e. all the information required to verify block proposals can be included within the proposal itself – no additional state information is required.

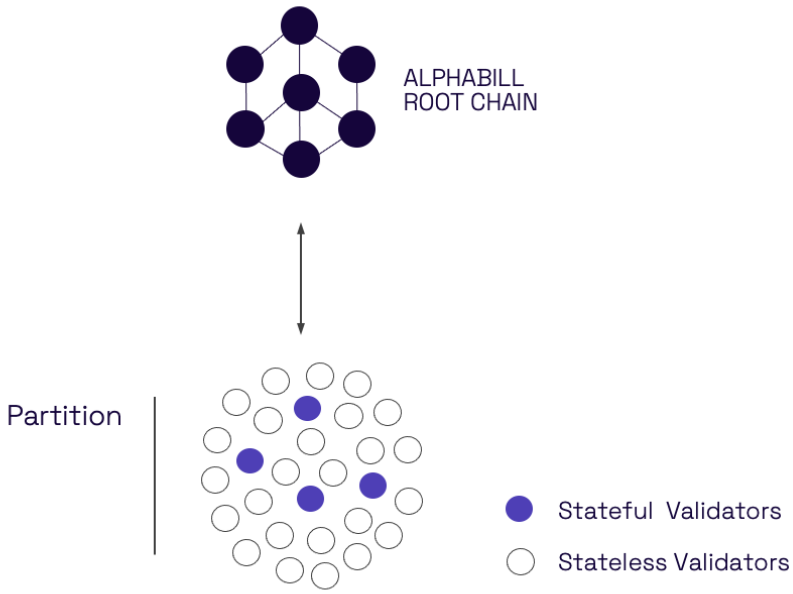


Figure 24. Stateless Validators verify blocks without needing to synchronize the chain.

Stateless validators allow for massive decentralization. Anyone with additional computational power can join the network and instantly start to earn rewards by verifying block proposals sent by stateful validators.

7 Alphabill Computational Model

There are two computational models in Alphabill, one for computations that require only a single unit and one for computations that operate on multiple units.

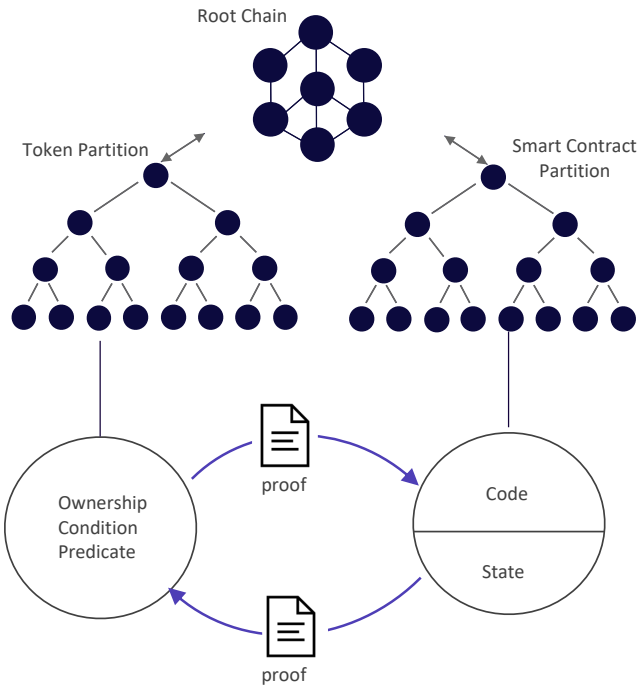


Figure 25. Predicates and Smart Contracts used for single and multi-token computations.

Both models are shown in the above figure. There are two Alphabill partitions shown, a Token Partition and a Smart Contract Partition. Part of the on-chain state of a token is an *owner predicate* or the conditions that need to be satisfied for a transaction order to be accepted and executed. For multi-unit computations smart contracts are used. Alphabill has a system-defined EVM partition for smart contracts but in principle any smart contract platform can be used.

7.1 Predicates for single unit programmability

The most basic form of an owner predicate would be the verification of a digital signature, i.e., in order to transfer ownership, the signature on the transaction order needs to be signed by a private key that matches the public key stored as part of the predicate.

TOKEN FORMAT

Field	Meaning	Type
i	Token ID	256-bit
d	Token data	var
φ	Owner predicate	Script

Figure 26. Simplified data format for a token.

Shared ownership of tokens can be managed through predicates on individual tokens. A predicate might encode a condition saying, "next transaction order must be signed by public key A OR B", giving shared ownership semantics. Another example would be a condition saying "next transaction order must be signed by public keys A AND B" giving multisig semantics to ownership. **Predicate evaluation is part of the transaction validation rules – all performed** locally within a shard, without requiring global ordering **of transactions or the existence of shared state**. Predicates are described in detail in section 13.

7.2 Smart contracts for multi-unit programmability

When a computation such as a smart contract requires multiple tokens as inputs (for example a DEX, (decentralized exchange), then predicates are used to change the ownership condition predicate such that only a specified smart contract is capable of transferring ownership.

To allow a smart contract to verify a proof from another partition we take advantage of the Unicity Certificate framework, to allow a common root of trust. The Unicity Certificate, generated by the Root Chain, provides for the creation of a proof of a token's state, and this proof can then be transported across shard boundaries, interpreted, and acted upon by smart contracts on different shards.

The first step is to transfer conditional ownership of a token to a smart contract. This is done using predicates, where a user sends a transaction order to a token address assigning ownership to a smart contract. Once the new predicate has been registered in the token ledger the data and proof can be sent to the smart contract address. As the smart contract and token share a common unicity framework the smart contract can verify the correctness of the proof of the token transfer and execute its code accordingly. The smart contract then generates proofs which can be used by a new owner to initiate settlement back on the original token shard.

An example of implementing an Automated Market Maker (AMM) is shown in section 16.

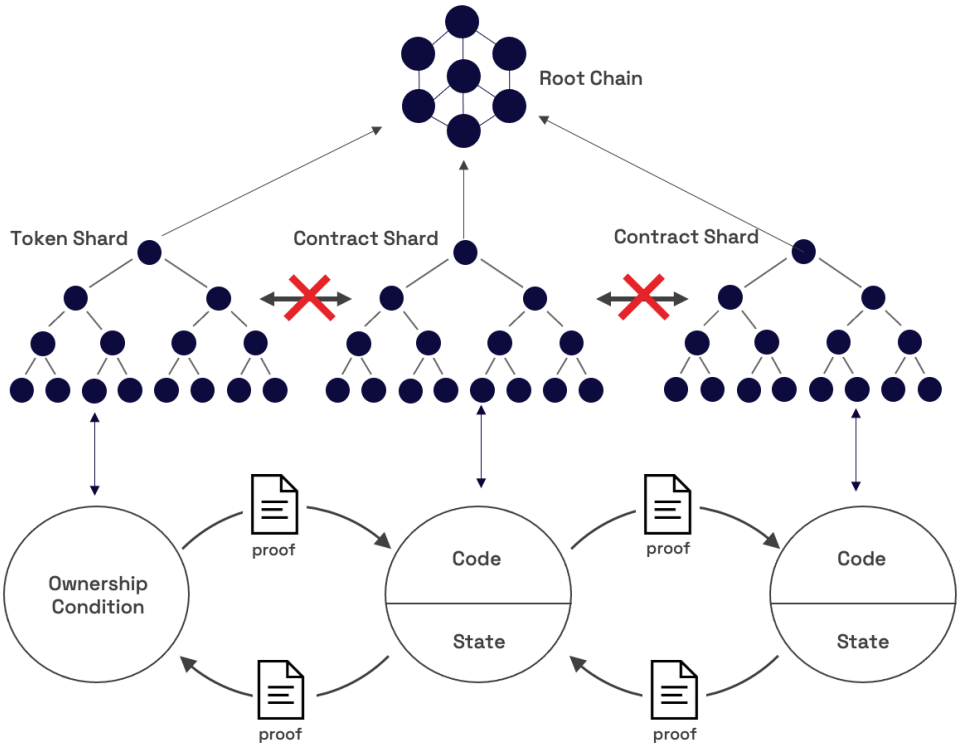


Figure 27. Smart Contract Composability

Smart Contracts are composable, in that multiple smart contracts on different shards can be chained together such that the output of one contract can be used as the input to the next contract.

7.3 Security Advantages of Separating Execution from Settlement

The advantages of this approach to computation are parallelism and protection from “rug-pulls” due to malicious or flawed smart contracts. Settlement (i.e. change in ownership of tokens) does not happen within the smart contract. Only after the smart contract executes can the proofs be generated necessary to initiate settlement on the token partition. This makes it possible to introduce audit functions or checks post smart contract execution but prior to settlement.

This approach of predicates for single unit programmability and smart contracts for multi-unit programmability combines the best of both worlds - there is no shared state amongst contracts and developers have the flexibility to use any development environment. Alphabill has its own EVM based development environment, but any run time can be used including other blockchains, provided the environment has access to the Unicity Certificate generated by the Root Chain in order to verify the proofs it receives as inputs.

8 Alphabill Consensus Protocol

There are several different types of ledger consensus used in blockchain protocols.

Longest Chain Rule (e.g., Bitcoin): blocks are proposed; if the next block chains from this block and then some more valid blocks chain continue the chain, then the block can be considered as final (with probability depending on chain length).

Probabilistic Convergence (e.g., Avalanche/Snowball): an optimization of the previous where the validators gossip and stick to the majority until converging to a likely agreement about the next block's content.

Deterministic Finality (e.g., Tendermint): nodes run a consensus protocol (in the narrower sense of consensus from distributed systems research[9]) to reach agreement and immediately finalize every block.

Alphabill is similar to Tendermint in that it provides deterministic finality. The Root Chain uses a low-level consensus module based on chained Hotstuff[10], **a BFT consensus protocol specifically** optimized for latency not throughput (as the Root Chain does not need to validate client transactions).

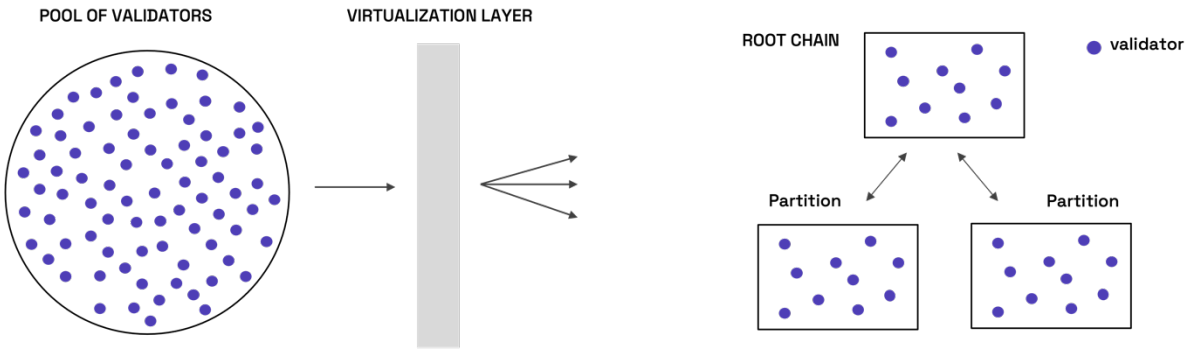


Figure 28. Validator Virtualization

[9] See for example Chapter 5 of "Introduction to Reliable and Secure Distributed Programming" by Christian Cachin, Rachid Guerraoui, Luís Rodrigues. Springer, 2011. DOI: 10.1007/978-3-642-15260-3

[10] https://arxiv.org/abs/1803.05069

A virtualization layer is operated by the Governance Partition validators, such that validators who wish to join the network are allocated at random to either the Root Chain or a specific shard on a partition. Validators build a reputation in a shard and are incentivized based on committed stake, equalization, and stabilization rules, ensuring balance and security across different shards.

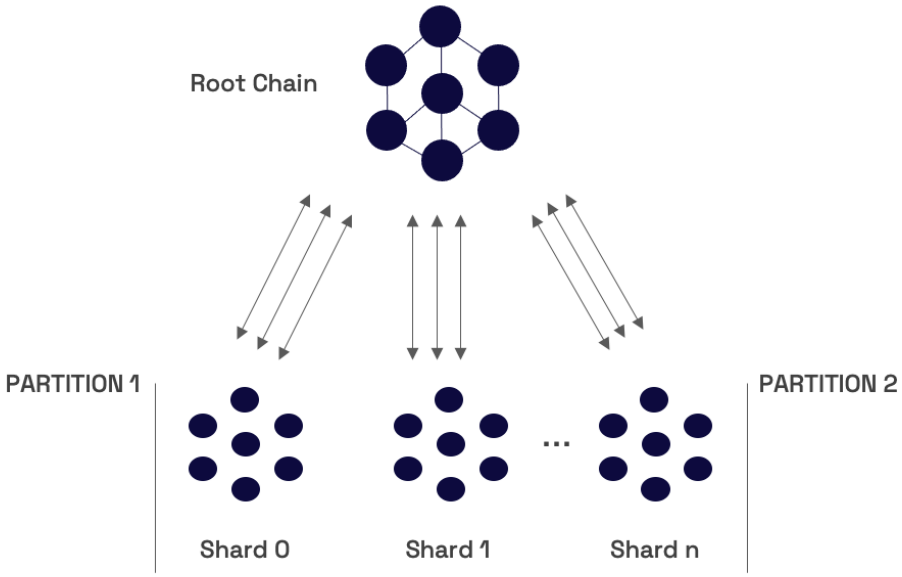


Figure 29. A single consensus instance.

The Root Chain is **isolated from the transaction processing load**. Instead of client transactions, the Root Chain validators only need to validate and **process aggregate summaries of proposed state transfers from shards**. They effectively check the results of the shard validators and confirm they are in coherence.

Importantly the set of validators on a shard **validate and execute transactions but** do not participate in a shard-level consensus protocol. Shard validators cannot create a valid block unless it has received a Unicity Certificate from the Root Chain.

8.1 Steps in the Consensus Protocol

1. Clients send transaction requests to a shard (wallets know which shard as the identifier of the token encodes the relevant information). The same principle applies to smart contracts.
2. Validators within a shard receive transaction orders from clients, check them and forward to parallel block proposers, based on round-specific mapping.
3. Block proposers validate transactions as they arrive, assemble a block proposal, and broadcast it to other validators in the shard. Validators validate the block and update their local state tree.
4. Each shard validator then independently broadcasts an aggregate summary of its proposed state transition to the validators in the Root Chain. Importantly the validators in shards do not participate in independent consensus protocols. They simply demarcate blocks, validate transactions, and send a summary of the result to validators in the Root Chain.
5. Root Chain validators reach consensus confirming that received summaries from shards are in coherence (this is a defined majority agreement depending on the requirements of each partition) and form valid, safe state transitions extending the previous certified states of shards. Unanimous shard-level agreement would require less validators in a shard to ensure decentralization but at the expense of liveness.
6. The Root Chain validators collectively generate a cryptographic proof of uniqueness or Unicity Certificate (UC) as the final step of the ledger consensus protocol.
7. This UC is returned to each shard validator which then adds it to its block to finalize it, and the process repeats.
8. The next round is started, parameterized by the recent input from the Root Chain.

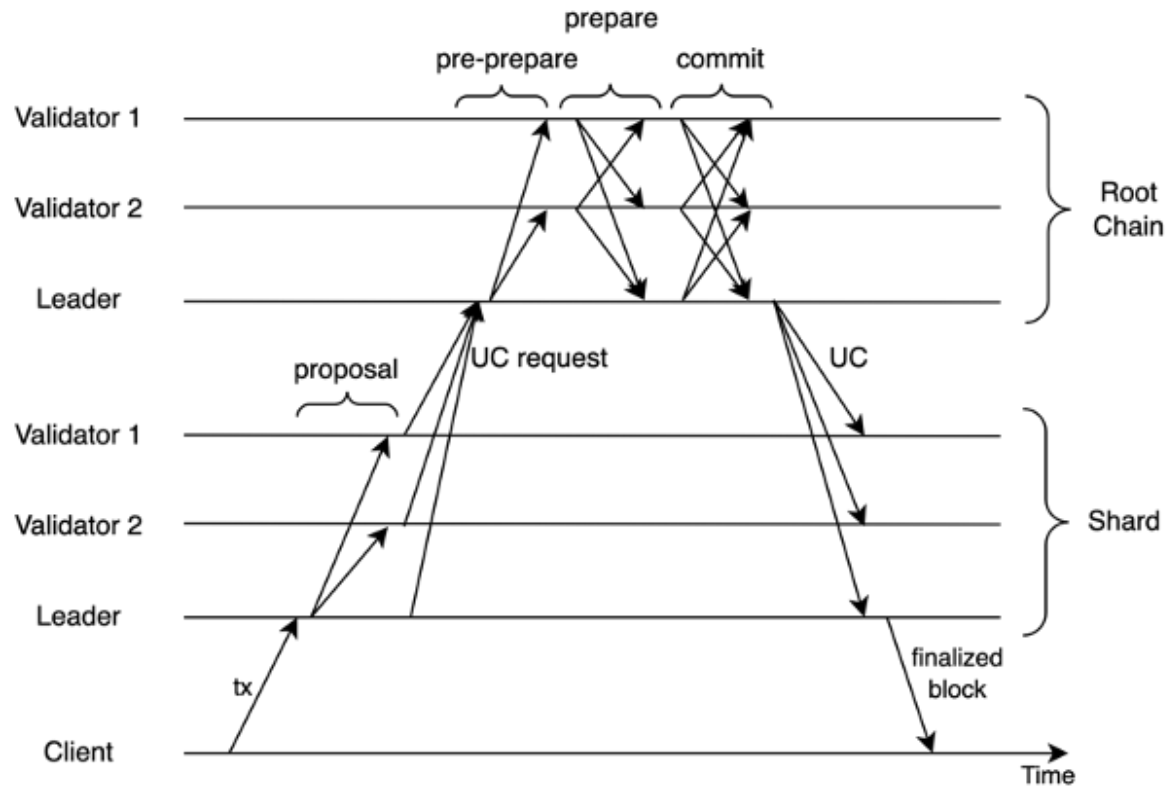


Figure 30. Simplified message flow of a consensus protocol round.

8.2 Delegated Proof of Stake execution

Alphabill is governed by Delegated Proof of Stake (DPoS) rules, as an efficient and environment-friendly alternative to Proof of Work mining. The rules are executed on-chain by the Governance Partition, avoiding the need for centralized parties overseeing the operational aspects of the network. DPoS rules perform permissionless validator assignment into partitions and shards and remunerate the validators and the stakers behind them with block rewards.

DPoS rules prevent “sybil attacks” by requiring that each validator is backed by a possibly delegated but unique stake, which, due to the scarcity of ALPHA tokens, limits the number of candidate validators. Stake is also a tool in the toolbox of crypto-economic security layer. Validators’ block rewards correlate to the amount of stake as well as the stability and the amount of usable work performed.

8.3 A comparison with federated blockchains.

It is informative to compare the Alphabill approach with other approaches to partitioning.

Figure 31. Federated vs Parallelizable Blockchains. Periodic relays vs constant connection.

Previous attempts at horizontal scaling have taken a federated approach, where multiple blockchains operate semi-independently, each with their own consensus protocols run by a distinct set of validators. Federation occurs via “bridges” or “relay chains” that allow communication between the different blockchains. The security model of each blockchain in the federation is distinct and each may only have a small number of validators, and an even smaller number of relay chain or bridge nodes. This approach has uses in *decentralized applications,* where small applications can be run in separate blockchains, each of which has lower security requirements than what would be needed for a global settlement layer.

In Alphabill’s case, the partitions are sub-trees of a large, distributed state tree, thus there is no need for periodic relaying of messages or trust anchors, and no inherent security risks. In other words, it is not a periodic process but a constant connection, greatly enhancing performance as well as security.

8.4 A comparison with layer two scaling solutions

Layer two solutions are proposed as solutions to the scalability limits of layer one protocols however these either sacrifice security (Optimistic Rollups) or performance (ZK Rollups). Due to the state commitments on the layer one, they improve throughput by using centralized components **before hitting a hard limit, improving layer one performance about two orders of magnitude at most**. There is also no direct way to settle transactions across rollups - whether that is a simple payment or linking smart contracts together (composability). Another challenge for rollups is censorship resistance – the sequencer is typically a single monolithic server.

9 The Alphabill Trust and Security Model

The parallelism of Alphabill allows for different trust models to be selected by users. Every user can independently audit the individual histories of tokens that they have ownership of. They can store not just the private key that controls transfer of the token but also the token ledger itself. This independent verification at the token level opens up new methods to ensure data availability and detection of malicious activity.

There are effectively five layers of security to detect and mitigate malicious behavior of validators.

a) The first layer happens during the block production process. Validators on a shard independently send an aggregate summary of their transactions to a set of Root Chain validators. On a partition level basis, the required level of agreement between the shard validators can be selected through a partition management governance process to determine the level of safety vs liveness for the partition in question. This layer guarantees security under the assumption that only a limited fraction of validators is malicious.

b) The second layer also happens during the block production process. A single honest validator that is part of the validator set within a shard can detect malicious activity through the validation of transactions. If malicious activity is detected a fraud proof can be generated and sent to the Root Chain prior to the Unicity Certificate being created and returned to the shard. This layer guarantees security on assumption that there is at least one non-colluding validator observing the invalid block proposal. There is no restriction on the number of validators in a shard and the settlement time is not impacted as the number increases.

c) The third layer is crypto-economic security. Here, the underlying assumption is that validators are malicious, but economically rational actors. The cost of losing rewards and potential slashing outweighs the economic gain of malicious activity. It is still possible that a malicious actor behaves irrationally, for e.g., fame, but it is very unlikely that a majority of validators behave in a self-sacrificing way.

d) The fourth layer of security is applied post block production. Here the underlying assumption is that **all** validators in a shard are potentially malicious and adaptively collude. Stateless validators participate in a crowd-sourced verification of the ledger by sharing the work of auditing every block in every shard in every partition. If an inconsistent block is discovered a fraud resolution process is initiated with potential slashing of offending validators. This function has the additional benefit of confirming data availability of the ledger.

e) The fifth layer of security relies on users to verify the history of each individual token that they take ownership of. In Alphabill, due to decomposability, a user can do this independent verification *at the token level* i.e. users do not need to rely on validator-cluster generated proofs but can instead verify compact token ledgers. In other words, a recipient of a token can verify that a) the token was minted in a valid genesis event and all historic transactions associated with that token have been executed correctly.

d) and e) are unique to Alphabill due to state tree recursion and decomposability.

The analogy with physical cash is relevant. When someone gives you a 20 USD note, you just check the validity of that note. You don't need to run the entire economy through your device and check every other transaction in history to be sure that your 20 USD note is valid.

9.1 Breaking down security into its components

The most important aspect of security is the property of safety. This includes non-forking at the token level (no double-spending) and at the partition level (no alternative hidden histories of partitions). In order to fork, an attacker must control a configurable number between 51% and 100% of partition validators AND at least 2/3 of Root Chain validators, that is, the Root Chain is offering "shared security" to partitions.

- The property of **validity** (rejecting invalid transactions) is guaranteed by a configurable number between 51% and 100% of partition validators and **remains efficiently auditable to token owners (token ledgers) and the world, thanks to unicity guarantee.**

- The property of **liveness** is guaranteed by between 49% to 0% of partition validators (complement to previous) OR at least ⅓ of the Root Chain validators.

- **Data availability** of transactions is provided by a quorum of partition stateful validators.

These honest majority assumptions apply to the first two layers of security encompassing the distributed machine setup (see previous section). All other layers, like the crypto-economic security which works even if all validators are dishonest, are effective in parallel and enforce the overall security.

10 Decentralization and the Blockchain Trilemma

In the original Bitcoin blockchain there is no requirement for users to trust anyone. A user can download the entire blockchain, go through the ledger transaction by transaction to independently verify its consistency. However, as the system scales, this approach inevitably leads to centralization. For example, at 1M transactions per second the Bitcoin blockchain would be in the order of one Exabayte of data, making it beyond reach of individual users to conduct independent verification. As such a major challenge and design goal of Alphabill is to scale without sacrificing decentralization.

To many in the community due to a widely held belief known as the "Blockchain Trilemma" this is considered impossible. The term, as coined by Vitalik Buterin, states that decentralized networks can only provide two out of three benefits at any given time with respect to decentralization, security, and scalability.

The challenge with this and other rules of thumb such as the "Nakamoto Coefficient" is that, while they can be potentially useful generalizations, they are not based on science. To be more scientific the abstract concept of security needs to be decomposed into its sub-components, addressed by a sufficient amount of validation resources. The breakdown of security into its components is described in the previous section.

Decentralization is a compromise. More decentralization adds overhead, as there are more replica machines, synchronizing the state and re-doing the same computations to validate state changes, in order to overwhelmingly outbalance the effect of a malicious entity trying to obstruct or overtake the network.

Alphabill achieves efficiency by providing redundancy in a flexible way. As the network is sliced into task-specific partitions; decentralization is addressed individually by assigning necessary quorum sizes of stateful and stateless validators for partitions. The number of validators per partition does not impact performance allowing the system to have partitions with a very high level of decentralization, e.g., the governance and native currency partitions, or running with just a few validators, e.g. enterprise applications on dedicated partitions.

In summary, with a decomposable blockchain such as Alphabill it is possible to achieve all three elements:

- Scalability is achieved by having many shards validating transactions in parallel. Crucially, cross-shard communication is not required during transaction settlement, ensuring performance is not degraded as the system scales.

- Security is achieved by decomposing it into its sub-components, each addressed by a sufficient amount of validation resources. Users can independently verify the ledger histories of individual tokens that they receive, eliminating the need to trust validator consensus.

- Decentralization is achieved by having a sufficient quorum of stateful and stateless validators per shard, **managed in a permissionless way by on-chain Proof of Stake processes.** Stateless validation allows anyone with consumer accessible hardware to participate in verifying blocks without synchronizing the chain in advance, democratizing access and enabling mass participation.

11 Partitions and Alphabill Transaction Framework

The use of partitioning allows Alphabill to differentiate the transaction validation rules, to optimize for specific use cases across a wide array of possible partition types. Each partition must implement a transaction system, which have:

- units u, each unit having a unique identifier, owner predicate and unit data.
- transactions that delete units, create new units, or change the data of the units.

The Alphabill transaction framework defines a language for describing the functionality of transaction systems: state and transactions (syntax and semantics) as well as provides libraries and toolkits for developing transaction systems. The framework, based on descriptions of transaction systems, registers and assigns identifiers to transaction systems, provides the unicity certificate service for the registered transaction systems: unique root hash h and summary value V for every pair (n;), where n is the block number.

Some of the elective features available to developers are listed below.

11.1 Censorship Resistance and Fair Ordering

Transactions are encrypted at the client side using per-epoch, per partition public key, and then delivered to partition validators, who order transactions in blind and commit to this ordering. Before validation and execution, each partition validator threshold-decrypts transactions and then they exchange their decryption shares. After combining enough shares, it is possible to obtain transactions in clear and continue as usual. Fee payment is handled by the ”double envelope” method.

Alphabill rotates the block proposer role at every round, thus, even without encryption, transactions may be only delayed until accepted by an honest validator.

11.2 Private Transactions

In so-called “dark partitions”, transactions are accompanied with zero knowledge proof of correctness: input unit is valid, transfer authorized by the owner, other transaction rules satisfied. Validators see opaque transactions orders only, without revealing unit data and transaction counterparties.

11.3 ZK Validity Guarantee

The data model of Alphabill allows some partitions to use zero knowledge proofs to prove the validity of transaction execution to the Root Chain. This extends the shared security umbrella of Root Chain from safety property to the validity property. Partitioning allows the use of specific ZK friendly cryptographic algorithms in this partition, and to place limits to the complexity of owner predicates, greatly increasing the efficiency (proving speed) of such partitions.

11.4 KYC and AML

Know Your Customer (KYC) and Anti Money Laundering (AML) rules are implemented at the client interface (wallet, crypto exchange) level, while the successful enforcement is checked by partition’s ledger rules, whenever required by use-case specific compliance environments.

12 ALPHA Native Currency Partition

This is a system-defined partition that manages ALPHA, the native currency of Alphabill. The ALPHA bills have three use cases:

- They can be staked to participate in Delegated Proof of Stake.
- They can be used as means of payment for services on the network.
- They can be used for on-chain governance where the bills are used in a voting process.

For a detailed description of the tokenomics please see www.aphabill.org

13 User Token Partition: Object Oriented Design

Alphabill offers a flexible framework for defining custom token types, creating tokens of these types, and transacting with such tokens.

13.1 Fungible and Non-Fungible Tokens (NFTs)

Each token type in Alphabill is designated as either fungible or non-fungible. **Fungible and non-fungible tokens are represented differently in the state tree: fungible tokens have an amount (64-bit unsigned int) and can be split and joined. NFTs, on the other hand, might contain a URI and arbitrary binary data which is possible to update.**

The most important property of a fungible token is the amount or value that it represents. A fungible token can be split into several smaller tokens of the same type so that the sum of the values of the resulting tokens is equal to the value of the original token. Conversely, several fungible tokens of one type can be joined into one larger token of the same type so that the value of the resulting single token is equal to the sum of the values of the input tokens. In both cases the source tokens are consumed in the process (deleted from the state tree), to ensure that neither of these operations create or destroy value and don't consume memory.

Non-fungible tokens, in contrast, have distinct identities and, in general, even two tokens of the same type are not interchangeable, even though they may share some characteristics. While currently the most well-known use of non-fungible tokens is collectible items of digital artwork, we expect this to be a relatively niche application in the longer term and most future use cases to revolve around representing various permissions, rights, and credentials instead.

13.2 Predicates

Predicates are functions that return a single TRUE or FALSE. They are used to check if an input meets some condition. For example, isDigit(c) might be a predicate function that returns TRUE if its input character c is a digit and FALSE otherwise.

Predicates are used in the User Token Partition to enable the customization and programmability of tokens, similar to Bitcoin locking and unlocking scripts. There are many types of predicates used in Alphabill and their use together with object-oriented inheritance enables a rich ecosystem of tokens to be developed.

13.3 Owner predicate:

In general, ownership of each token in Alphabill is controlled by an *owner predicate*. The predicate is a function that receives as inputs a transaction order and the current state of the unit that the transaction targets. The function returns a decision whether the transaction should be accepted or rejected. In most cases, the condition is that the transaction order must have a signature that verifies with a public key embedded in the predicate. If a transaction order is deemed valid by the predicate, it is allowed to execute. Executing a ”transfer” transaction means just replacing the current owner predicate with a new one. Often, the new predicate is a similar function but contains a different public key – with the effect that a different private key must be used to sign the next transaction order affecting the token; since the old owner can no longer produce valid transaction orders, this indeed means that the control of the token has been handed over to the new owner.

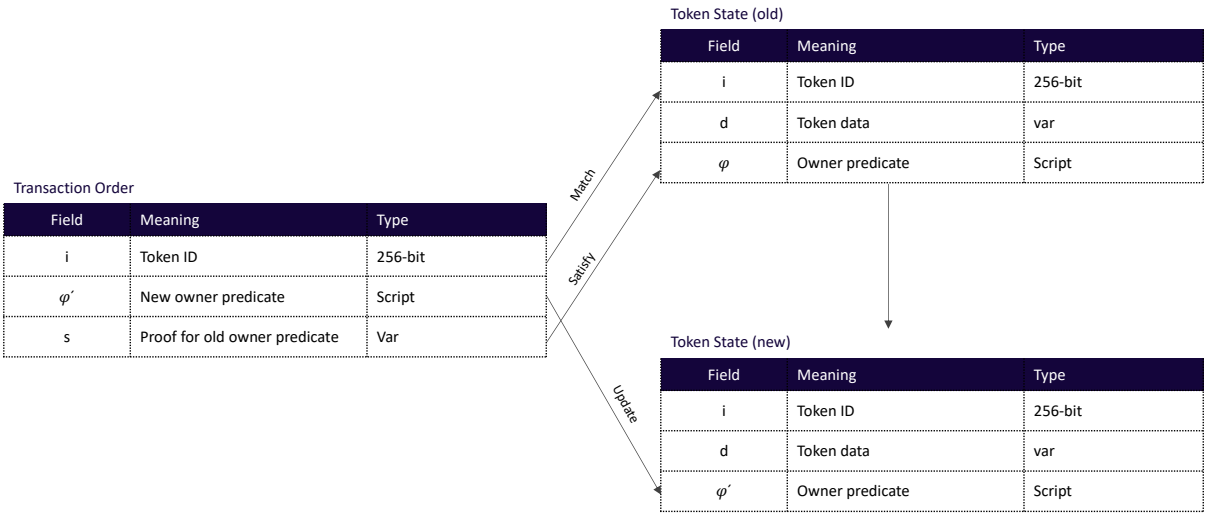


Figure 32. A simplified example of an owner predicate being updated through a transaction order.

The above figure shows a transaction order (on the left) and the state of a token before and after the transaction order is executed. The validator does the necessary checks (such as making sure the tokenID corresponds to an existing token of the correct type for the transaction), then the "proof for old predicate" is computed to make sure it satisfies the current owner predicate. If so then the owner predicate is updated.

13.4 Token Types

Each user-defined token belongs to a token type. The type determines several properties common to all tokens of this type. Token types are also represented as first-class entities in the User Token Partition, i.e. each token type is allocated space in the state tree. Anyone can define a token type but unlike regular tokens, they cannot be transferred. Instead, a token type defines several predicates that affect the tokens of this type.

NON-FUNGIBLE TOKEN TYPE FORMAT

Field	Meaning	Type
sym	Symbol (short name)	UTF-8 text
nam	Name	UTF-8 text
ico	Icon (image)	UTF-8 text
i	Parent type	256-bit
φ_m	Mint predicate	Script
φ_o	Inherited owner predicate	Script
φ_s	Inherited subtype predicate	Script
φ_d	Data update predicate	Script

Figure 33. A simplified illustration of a non-fungible token type

13.5 Mint Predicate: Enforcing Restrictions on Minting

The *mint predicate* must be satisfied to create new tokens of this type. A typical example is that the minting order may be required to be signed using a specific private key – which would limit the issuance of new tokens of this type to the owner of the key. This condition is freely programmable and can enforce many other kinds of restrictions. For example, the creator of the token type may require royalties for the use of the type; this can be enforced by having the minting condition check for proof of payment. Another example is limiting minting to a certain set of authorized parties; this can be enforced by having the mint predicate require proof of membership in the corresponding list, which can itself be implemented and managed as a special user-defined token.

13.6 Inherited Owner Predicate: Enforcing Restrictions on Transfers

A second predicate in a token type is the *inherited owner predicate* of all tokens of the type. With this predicate defined, any transfer order with any token of the type will have to satisfy both the owner predicate on the token itself and the inherited owner predicate from the token type.

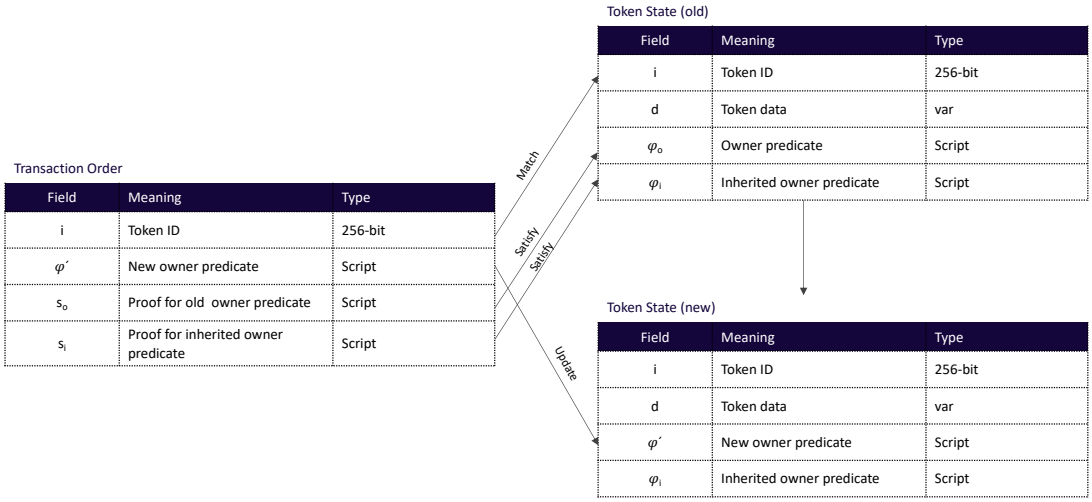


Figure 34. Inherited owner predicates are immutable and not replaced by transaction orders.

The crucial difference between the two is that the transfer transaction replaces the owner predicate on the token itself, but the inherited owner predicate is immutable and can therefore be used to define restrictions that the owners of the token cannot remove. A very simple example is making tokens non-transferable by setting the inherited owner predicate to a function that always returns false; a possible use for such tokens is representing any non-transferable credentials such as driving licenses or academic degrees. Another possible use case for the inherited owner predicate is implementing mandatory royalty payments to the original author of a digital artwork whenever the work is sold from one owner to the next.

13.7 Inherited Subtype Predicate: Enforcing Restrictions through Inheritance

Another predicate in a token type is the *inherited subtype predicate*. When a subtype is defined, the tokens of the subtype inherit the properties of their parent types along the inheritance chain. As an example, many jurisdictions limit certain financial products to accredited investors only in order to prevent less experienced parties from taking undue risks. Some bonds fall into this category and (as a rather simplified example) this can be modeled by having a ”generic bond” token type from which a ”restricted bond” type is derived where the ”generic bond” type defines inherited owner conditions that apply to investors buying any bonds (such as the buyer having passed the KYC checks) and the ”restricted bond” type adds the condition that the buyer has to be an accredited investor. Transfers of tokens of the ”generic bond” type need to satisfy in addition to the tokens’ owner predicate the predicates for transfer of the ”generic bond” type. Transfers of the “restricted bond” type need to satisfy the same predicates as the “generic bond” type and in addition the predicates for the “restricted bond” type.

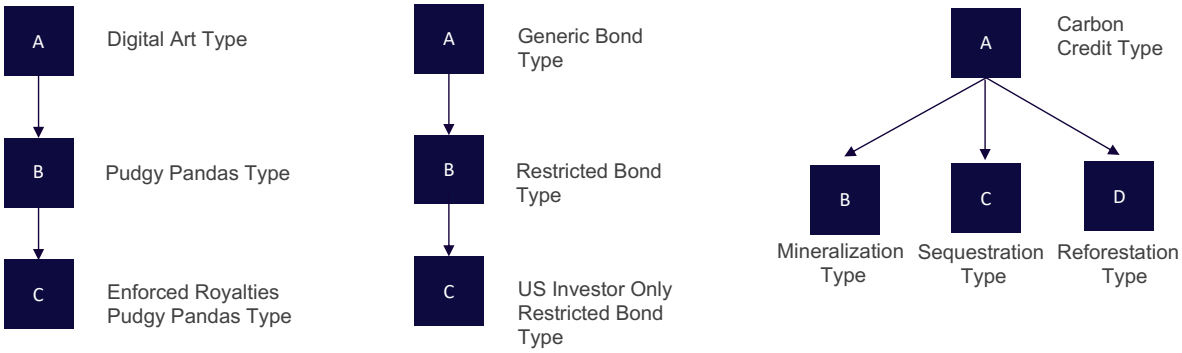


Figure 35. Examples of token type inheritance

In the spirit of object-oriented modeling and design in general software engineering it is possible to inherit many levels deep and define a rich ontology of token types.

13.8 Data Update Predicate: Implementing State Machines

Each non-fungible token has a ”data” field. The contents and interpretation of this field are entirely user-defined. The data is mutable, subject to satisfying an additional predicate called the *data update predicate*. This works like the other predicates: a user sends a transaction order which supplies the new data that should replace the old data. Both the old and the new data are supplied as arguments to the predicate and if the predicate returns true, then the token’s data is changed by the transaction.

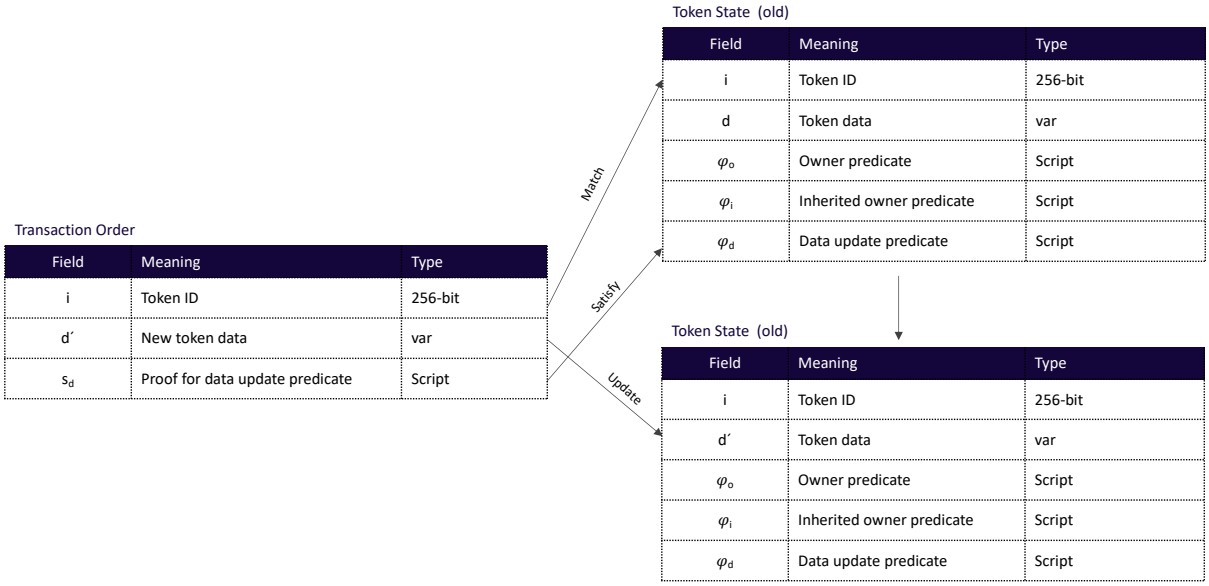


Figure 36. Simplified example of an NFT owner updating the NFT data.

While the data is mutable, the data update predicate is not. It is defined when the token is created and cannot be modified after that. Additionally, non-fungible token types can define clauses that are inherited into the data update predicates of all tokens of their types. This allows application developers to implement various kinds of state machines where the data update predicates specify which state transitions are allowed and which are not, and those rules cannot be overridden by the token owners.

Putting together the features of inheritance and controlled mutability allows developers to create a rich ecosystem of useful tokens that can be used to represent real world processes and use cases. Tokens can be directly used for the exchange of digital rights, goods, and services, and they can be programmed to follow business processes in an automated decentralized manner.

14 Atomicity Partition: Decentralized Exchange

In a sharded blockchain system it is useful to provide native support for atomic transactions. These can be in the form of multi-token atomic transfers or atomic swaps between different token types (e.g., for Delivery versus Payment where assets are exchanged for currency). In a single machine model this is straightforward; smart contracts such as Uniswap have access to the entire global state and can ensure atomicity – either the contract succeeds, and multiple tokens are swapped, or the contract fails, and no ownership is changed. In Uniswap, the smart contract provides both exchange and settlement services - these aspects of trading are typically separated in traditional financial applications for reasons of scalability and specialization.

In Alphabill a dedicated atomicity partition provides this functionality, allowing decentralized exchanges to be separate from the settlement operations.

15 Governance Partition

The governance partition handles voting on governance proposals and manages validators and partitions. The role of this partition is sixfold:

a) On-chain governance: voting on governance proposals.
b) Partition Management: adding and removing new partitions.
c) Validator Assignment: managing validator life cycles and reputations,
d) Network Capacity Management: approving dynamic sharding proposals
e) Validator Reward Handling: unlocking “common good” rewards, staking rewards, Root Validator and Transaction Validator rewards.
f) Software Certification: approving updates to transaction system specific validation software and coordinating software updates.

Latency in this partition is not critical as the operations performed here happen over a longer timescale than actual transactions. As such the block time can be much longer than other partitions, enabling mass participation.

As a decentralized public blockchain, Alphabill has an open validator set, free for public participation. Governance operations are automated as much as possible: for example, validator assignment into shards happens automatically, using a secure on-chain randomness beacon. Similarly, validator rewards and pay outs are calculated automatically using cryptographically authenticated data from other partitions.

15.1 On-chain Voting

Perhaps the most challenging part of governance is on-chain voting for governance proposals: a good solution needs to balance the needs of a diverse set of stakeholders while enabling timely and efficient decision making.

Voting in Alphabill is implemented in a series of steps to ensure a smooth transition to a well-oiled governance process.

In the first release, **all decisions regarding protocol changes** will be implemented fully off-chain, allowing for more experimentation in developing community-wide decision processes. **The Alphabill Foundation coordinates discussions and helps make sure to include opinions from as many stakeholders as possible.**

In a later release, the Governance Partition will implement fully on-chain voting, using a mix of delegated token voting and a bicameral setup allowing for more diverse stakeholder participation. Eventually it can be used to vote on all matters at hand, including software updates for the Alphabill platform.

16 Alphabill EVM Partition

Solidity and the Ethereum Virtual Machine (EVM) were hugely significant contributions to the community. It is possible to celebrate these inventions while recognizing their limitations. Ethereum's implementation of smart contracts has one major limitation – it is based on shared memory i.e., it assumes a shared global state. This enables composability of smart contracts but comes at the cost of scalability. Since the state is global, it must be able to be stored, and manipulated, in its entirety, on every validator. This implies that the overall global state can never grow larger than can be processed by a single machine, and that computation cannot take place in parallel.

Alphabill implements an EVM partition as a system-defined partition as shown below. This partition enables developer to deploy Solidity programs however this partition is **not shardable**, due to the limitations described above.

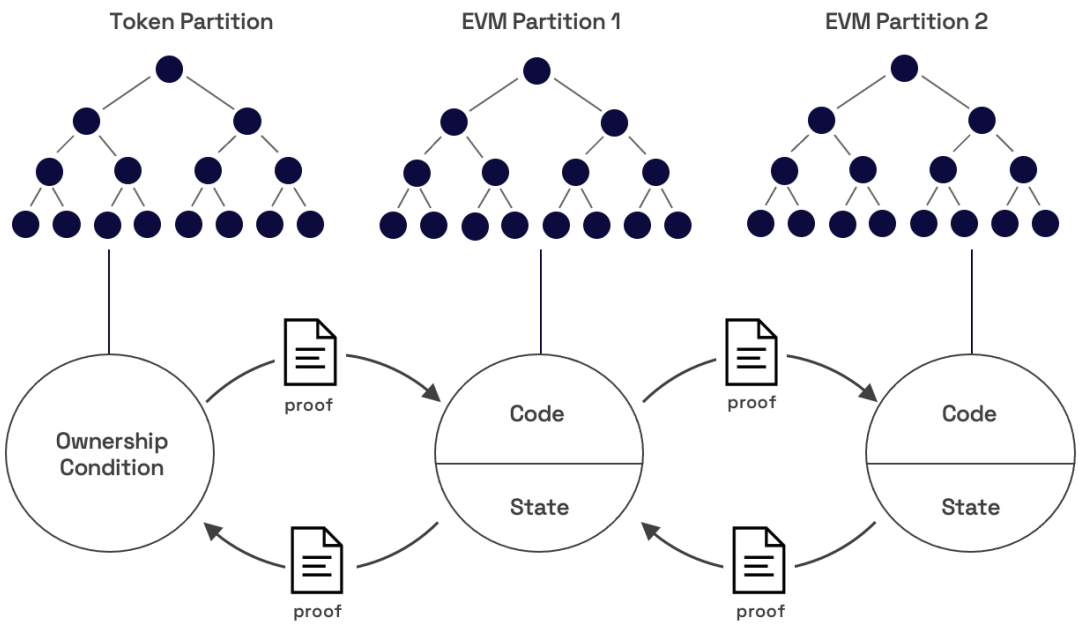


Figure 37. Multiple EVM Partitions

For scalability, the Alphabill architecture allows multiple EVM partition instances to operate in parallel. Smart contracts deployed in different EVM partitions do not share memory and cannot call each other directly. Interoperability is enabled by exchanging proofs which can be verified due to the common root of trust.

16.1 Implementing an AMM Smart Contract

In the above figure tokens live in the User Token Partition and an AMM contract lives on the EVM partition. The mechanism to call a smart contract works without actually moving the token, only the predicate on tokens is changed to allow the smart contract to verify that it is the new owner.

The proof of a token being in the required state is sent to the contract with a subsequent transaction order i.e. the first transaction order is sent by the user to the token shard to lock the token (locking here means that only the specified smart contract can unlock it). A second transaction order is sent by the user to the smart contract address which includes the transaction request to the smart contract as well as the proof that the token has been locked.

A simplified constant product AMM contract would have two token pools in contract memory, with identifiers (the state tree address) of the tokens locked and sent by liquidity providers.

A user who wishes to swap tokens will send a transaction order to the contract's Swap() function, with locked token proofs as an argument. The contract will then calculate the amount of returned tokens from the pair using the constant product formula, and create unlocking proofs in its memory and terminate. Post block creation the user can use the unlocking proof to claim ownership of the returned tokens.

Here is simplified pseudo-code:

```
function SwapToB(tokensA_in):
     invariant = poolA * poolB          // the product of pool sums is kept constant
     new_poolA = poolA + tokensA_in
     new_poolB = invariant / new_poolA
     outB = poolB - new_poolB            // sum of returned tokens B
     transferTokenBTo(sender, outB)      // creates unlocking proof(s)
```

The liquidity providers add new tokens to the contract pair-wise in order to maintain the exchange rate. When users see arbitrage opportunities they can make token swaps, which will bring the exchange rate close to the real-world exchange rate.

17 User Defined Partitions

Alphabill is designed such that partitions can be added by users in a permissionless way. These partitions can be anything from the Bitcoin and Ethereum blockchains to a Web2 database.

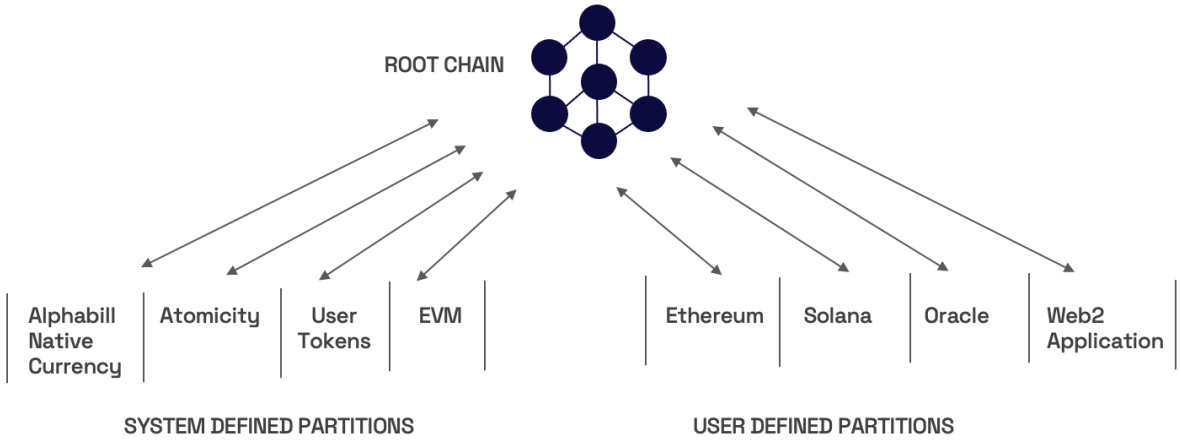


Figure 38. User defined partitions

17.1 External Smart Contract Platforms

To make chains interoperable a subset of validators on different chains may join the Alphabill framework as a partition i.e. they will request a partition ID. connect to the Root Chain and use a unicity certificate, generated by the Root Chain to certify its internal state. That state can then be transported across partition boundaries, verified, and acted upon in other partitions.

For example, to transfer a user token in Alphabill into an Ethereum smart contract a user will first transfer the Alphabill token into a state which gives specific control to an Ethereum smart contract. The proof that control has been given will be generated by the user and sent to the Ethereum smart contract address. As the Ethereum validators are connected to the Root Chain, they share a common root of trust, and the smart contract can verify the proof. The smart contract will then credit its internal account structure, execute, and redistribute value amongst its internal accounts. If a user wishes to withdraw tokens from the smart contract they can request a withdrawal, the smart contract will debit the user's account and then generate a proof that can be used by the user to reclaim ownership of the token on the Alphabill User Token partition.

17.2 Alphabill as a Cross Chain Interoperability layer

It is often claimed within the crypto community [11] that cross-chain communication is impossible without trusted third parties. However, they assume that an atomic swap requires solving the following problem:

> There are two chains X and Y. One wants to add a (swap) transaction *t* to both chains such that *t* is either added as a valid transaction to both chains, or none of the chains.

This task is indeed known to be impossible without a third referencing party as proved in 1980 by Even and Yacobi[12]. However, the atomic swap solution in Alphabill does not require a transaction *t* be simultaneously and atomically added to both chains. Instead, all four possibilities are considered:

a) *t* is included in both X and Y
b) *t* is included only in X
c) *t* is included only in Y
d) *t* is included in neither of the chains

The predicate of the transaction *t* guarantees that only in the first case, *t* changes the ownership of a unit. This is achieved by using a special predicate in *t* that guarantees the next properties:

- the (claimed) new owner can make the next transaction only by presenting evidence that t was accepted in the other chain.
- the previous owner can make the next transaction only by presenting evidence that *t* was not (and will not be) accepted in the other chain.

The Alphabill Atomicity Partition, together with User Defined Partitions in which a subset of other chain validators are connected to the Root Chain makes it possible to implement decentralized cross-chain swaps and other inter-blockchain operations in an atomic trustless manner.

[11] For example, "SoK: Communication Across Distributed Ledgers" https://eprint.iacr.org/2019/1128.pdf

[12] S. Even and Y. Yacobi. Relations among public key signature systems. Technical Report 175, Computer Science Dept., Technion, Haifa, Israel, March. 1980).

17.3 Centralized Web2 Applications

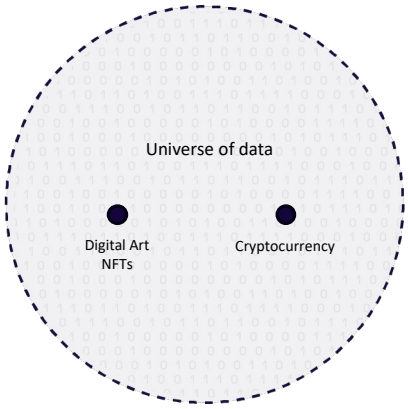
The User-Defined Partitions do not need to be decentralized. For example, it could be an existing enterprise application. The application can request a partition ID, receive, and verify tokens and then generate proofs to reallocate those tokens based on the application logic. This is similar to Ethereum layer 2 logic but enables any type of application, including existing enterprise and Web services to participate in the framework.

17.4 Oracles

A similar approach allows for external data sources to provide certified data to be used within the Alphabill platform. This can be consensus Oracles such as Chainlink or Gnosis, centralized external market data providers such as exchanges or hardware Oracles such as IOT devices. In the case of exchanges there is a single source of truth for market data – the exchange publishes the data. To make this data available in Alphabill an exchange will request a partition ID and use the Unicity Certificate service from the Root Chain to certify their data and allow users to transfer that data to be used in a relevant smart contract where it can be verified as authentic.

18 Public Token Infrastructure

In a *tokenized* Internet every data object created is or has an associated token[13], an authenticated and programmable data structure which can be assigned property rights that can be transferred without a trusted authority.



The cryptocurrency tokens and digital art NFTs we have seen in the first iterations of blockchain networks are just examples in the universe of data objects that can be tokenized. In our view the next version of the Internet all human and machine generated data will be tokenized.

Public Token Infrastructure, the evolution of Public Key Infrastructure (PKI), replaces digital signatures and timestamps with programmable tokens that have an additional proof of *uniqueness*.

Public Key Infrastructure	Public Token Infrastructure
Digital signatures, timestamps	Tokens (fungible, non-fungible, transferable…)
Trust an Authority	Trust no one (public verifiability)
Static, not programmable	Dynamic, programmable

Figure 39. Public Token Infrastructure

PKI, invented in the 1970s, has proved extremely successful for its original use case, i.e. sharing a secret across an insecure channel. However, the complexity and cost of key management means that it is almost universally not used to authenticate data. Today with some very few exceptions data is just 1s and 0s without any mechanism beyond trust to verify where it came from, when it was created etc.

[13] The data is the token for cryptocurrency. For other types of data, the token can encapsulate or be linked to data.

One major innovation of Bitcoin was to prove the *uniqueness* of data (the Bitcoin blockchain), a proof based on the assumption that the amount of energy required to reverse the Proof of Work algorithm would be practically impossible. Proof of uniqueness is necessary if data, such as currency tokens, has ownership and that ownership must be uniquely determined, and "double spending" prevented.

Public Token Infrastructure is a set of tools and technologies enabled through the Alphabill blockchain which extend PKI to support this proof of uniqueness property at the individual data object level. In this view cryptocurrency tokens or NFTs are just a tiny fraction of the universe of data, all of which can be tokenized.

19 Academic References

A. Buldas, et al., "A unifying theory of electronic money and payment systems,'" IEEE TechRxiv, May 2022 https://doi.org/10.36227/techrxiv.14994558

A. Buldas, D. Draheim, M. Saarepera, "A Theory of Secure and Efficient Implementation of Electronic Money," SN COMPUT. SCI. 4, 861 (2023)

A. Buldas et al., ``An Ultra-Scalable Blockchain Platform for Universal Asset Tokenization: Design and Implementation" IEEE Access, July 2022

A. Buldas, D. Draheim, M. Gault, Towards a foundation of Web3. FDSE 2022. Communications in Computer and Information Science (CCIS, vol. 1688)

A. Buldas et al., ``Secure and Efficient Implementation of Electronic Money," FDSE 2022. Communications in Computer and Information Science, vol 1688. Springer, Singapore.