**White Paper: The [System Name] Ledger – A BFT-Based State Machine Replication for Fractional Balance Management**

**Abstract**

This white paper introduces the [System Name] Ledger, a distributed ledger system designed for efficiently managing and updating a shared state representing fractional balances. Unlike traditional blockchain systems that maintain a complete history of all transactions, the [System Name] Ledger focuses on maintaining a current, authoritative state while discarding historical data after consensus is achieved. This approach prioritizes speed and storage efficiency, making it suitable for applications where immutability of history is less critical than real-time state synchronization. The system employs a Byzantine Fault Tolerance (BFT) consensus protocol to ensure data integrity and availability even in the presence of malicious or faulty nodes. A novel mechanism for account creation controlled by existing account holders is also introduced.

**1. Introduction**

The [System Name] Ledger is designed to address the need for a fast, efficient, and decentralized system for managing fractionalized assets or representations of value. Traditional blockchain systems, while providing strong immutability, can suffer from high transaction costs, slow confirmation times, and large storage requirements due to the need to maintain a complete transaction history.

The [System Name] Ledger adopts a different approach by focusing on maintaining an accurate and readily available *current* state, discarding historical data after it has been validated through consensus. This design allows for faster transaction processing, reduced storage overhead, and simplified node operations.

**2. Design Principles**

The [System Name] Ledger is built on the following key design principles:

* **State-Centric:** The primary focus is on maintaining a consistent and up-to-date representation of the current state of the system (i.e., account balances).
* **Byzantine Fault Tolerance:** A robust BFT consensus protocol ensures that the state is accurately replicated across all nodes, even if some nodes are malicious or experience failures.
* **Speed and Efficiency:** Prioritization of transaction processing speed and minimized storage requirements.
* **Permissioned Account Creation:** New accounts can only be created through the endorsement of existing account holders, providing a degree of control over network participation.
* **Fractionalized Representation:** Supports the representation and transfer of assets in highly divisible units.

**3. System Architecture**

The [System Name] Ledger consists of the following key components:

* **Data Object (State):** The core data structure is a key-value store where:
  + *Key:* Public key/address of an account.
  + *Value:* The account balance (a fractional value, adding up to a fixed total supply).
* **Transactions:** Transactions are signed messages containing the following information:
  + sender\_address: The public key of the account sending the funds.
  + recipient\_address: The public key of the account receiving the funds.
  + amount: The amount to transfer.
  + signature: The sender's digital signature of the transaction.
  + new\_account\_address (optional): The public key of a new account being created (must be endorsed by the sender).
* **Blocks:** Blocks serve as containers for transactions to be agreed upon, blocks contain
  + timestamp: Block creation timestamp
  + transactions: A list of transactions to be applied to the state.
  + previous\_hash: Hash of the previous block (links the blocks together, providing some immutability *during the block creation process*).
  + merkle\_root: The Merkle root of the block's transactions.
  + bft\_signatures: List of signatures from BFT validators agreeing on the block.
* **Nodes (Validators):** Nodes are responsible for:
  + Receiving and verifying transactions.
  + Participating in the BFT consensus protocol.
  + Maintaining a copy of the current data object.

**4. Account Creation**

The [System Name] Ledger employs a permissioned account creation mechanism to control network participation. New accounts can only be created if an existing account holder includes the new account's public key in a transaction and signs it. This transaction serves as an endorsement, vouching for the new account's legitimacy. The new account is initialized with a zero balance. This mechanism helps to prevent Sybil attacks and provides a degree of social trust within the network.

**5. Consensus Protocol (BFT)**

The [System Name] Ledger utilizes a BFT consensus protocol to ensure that all nodes agree on the current state of the system. The BFT protocol guarantees data integrity and availability even if some nodes are malicious or experience failures (up to a certain threshold).

[**Detailed Description of Chosen BFT Protocol (e.g., Simplified Tendermint)**]

This section should provide a detailed explanation of the chosen BFT protocol, including:

* **Roles (Proposer, Validator)**
* **Message Types (Propose, Pre-vote, Pre-commit, Commit)**
* **Steps in the Consensus Process (Propose, Vote, Commit)**
* **Fault Tolerance Threshold**
* **Mechanism for Validator Selection/Rotation (if applicable)**

[ *Example: In a simplified Tendermint-like implementation:*

1. **Propose:** One node (the *proposer*) is selected to propose a new block. The proposer can be selected randomly, or based on a round-robin schedule.
2. **Pre-vote:** Validators receive the proposed block and check its validity (transactions, signatures). If the block is valid, they send a "pre-vote" for the block; otherwise, they send a "pre-vote nil".
3. **Pre-commit:** If a validator receives pre-votes for the same block from more than 2/3 of the total voting power (of all validators), they send a "pre-commit" for the block.
4. **Commit:** If a validator receives pre-commits for the same block from more than 2/3 of the total voting power, the block is considered *committed*. Validators then apply the block to their local state. \*]

**6. Transaction Validation and Processing**

Before a transaction is included in a block and processed by the consensus protocol, it must be validated. Transaction validation involves the following steps:

* **Signature Verification:** Verifying that the transaction signature is valid and corresponds to the sender\_address.
* **Sender Existence:** Ensuring that the sender\_address exists in the data object.
* **Sufficient Balance:** Verifying that the sender\_address has enough balance to cover the amount being sent.
* **New Account Creation Check:** If the transaction includes a new\_account\_address, verifying that the account does not already exist.

After a block is finalized through the BFT consensus protocol, the transactions in the block are applied to the data object. Balances are updated, and new accounts are created (if specified in a transaction).

**7. State Management and History Discarding**

After a block is finalized and the transactions are applied to the data object, the historical block data is discarded. This includes the block itself, the transactions it contained, and the previous state of the data object. This design decision prioritizes storage efficiency and speed, but it also means that the system does not maintain a complete audit trail of past transactions. It is possible to create snapshots periodically for debugging or auditing, but the snapshot will not be a immutable record.

**8. Economic and Philosophical Rationale**

[**This section is intentionally left blank for you to fill in.**]

This section is critical for explaining:

* **Why this system is needed (the problem it solves).**
* **The economic model: How the system incentivizes participation and prevents abuse.**
* **The philosophical underpinnings: What values and principles guide the design and operation of the system?**
* **Why discarding history is acceptable in the context of the intended use case.**
* **The long-term vision for the [System Name] Ledger.**
  + How can the ledger be applied to current or real world situations?
  + What advantages does it hold over modern methods of value transfer?
  + What advantages does it hold over competing crypto assets?

**9. Security Considerations**

The [System Name] Ledger employs several security measures to protect against various threats:

* **BFT Consensus:** Protects against malicious or faulty nodes.
* **Digital Signatures:** Ensure the authenticity and integrity of transactions.
* **Permissioned Account Creation:** Helps prevent Sybil attacks.
* [**Add any additional security measures here (e.g., network security, key management, DoS protection).**]

**10. Future Directions**

* **Smart Contract Integration:** Explore the possibility of integrating smart contract functionality into the system.
* **Scalability Improvements:** Research and implement techniques to improve the scalability of the BFT consensus protocol.
* **Improved Network Communication:** Investigate the use of more efficient network protocols for node communication.
* **Data Object Snapshots:** A system to create snapshots to preserve state data

**11. Conclusion**

The [System Name] Ledger offers a unique approach to distributed ledger technology, prioritizing speed and storage efficiency by maintaining a current state while discarding historical data. The BFT consensus protocol and permissioned account creation mechanism ensure data integrity and controlled network participation. The [System Name] Ledger is well-suited for applications where immutability of history is less critical than real-time state synchronization and efficient fractional balance management.

**[End of White Paper Draft]**