SlimChain: Scaling Blockchain Transactions through Off-Chain Storage and Parallel Processing

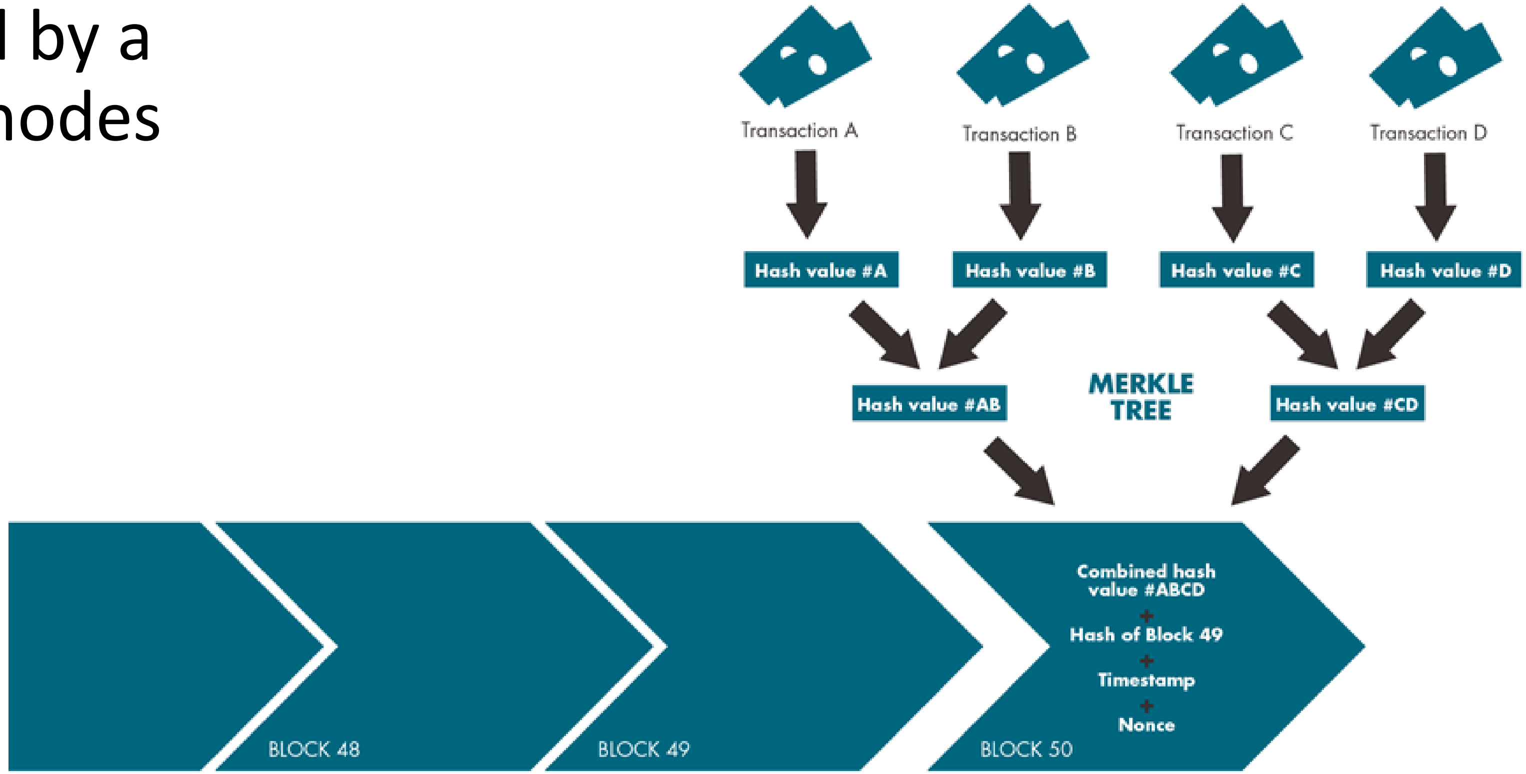
Cheng Xu[1, 2], Ce Zhang[2], Jianliang Xu[2], and Jian Pei[1]

[1]Simon Fraser University, Canada

[2]Hong Kong Baptist University, Hong Kong

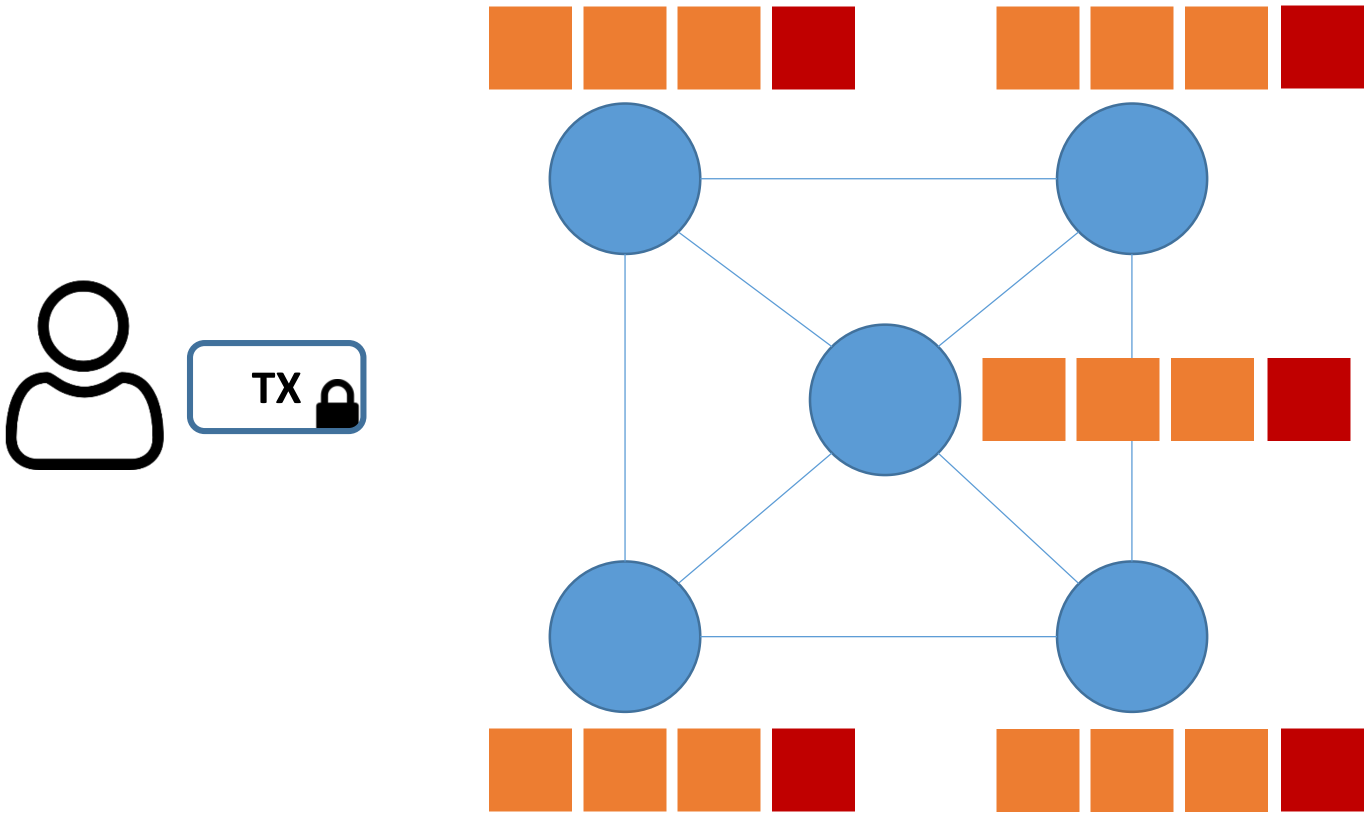
Blockchain Overview

- Append-only data structure collectively maintained by a network of untrusted nodes
  - Hash chain
  - Consensus
  - Immutability
  - Decentralization



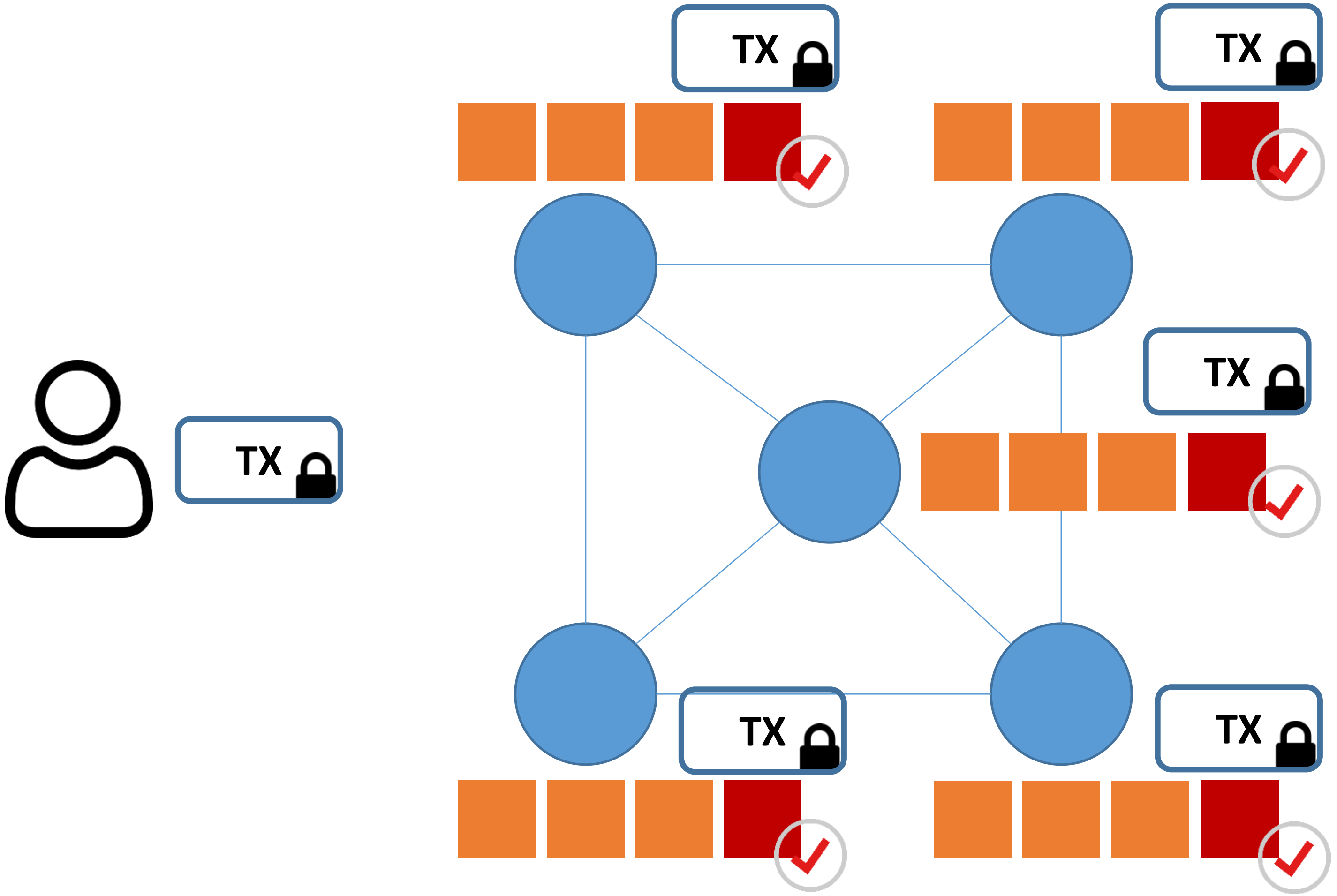
Current Blockchain System

- Features
  - Every node keeps a full replication of transaction history and ledger states
  - Every node validates all transactions in blocks
  - Easy to maintain the same order of transactions
  - Easy to ensure execution integrity
  - Bad for high storage and execution overhead

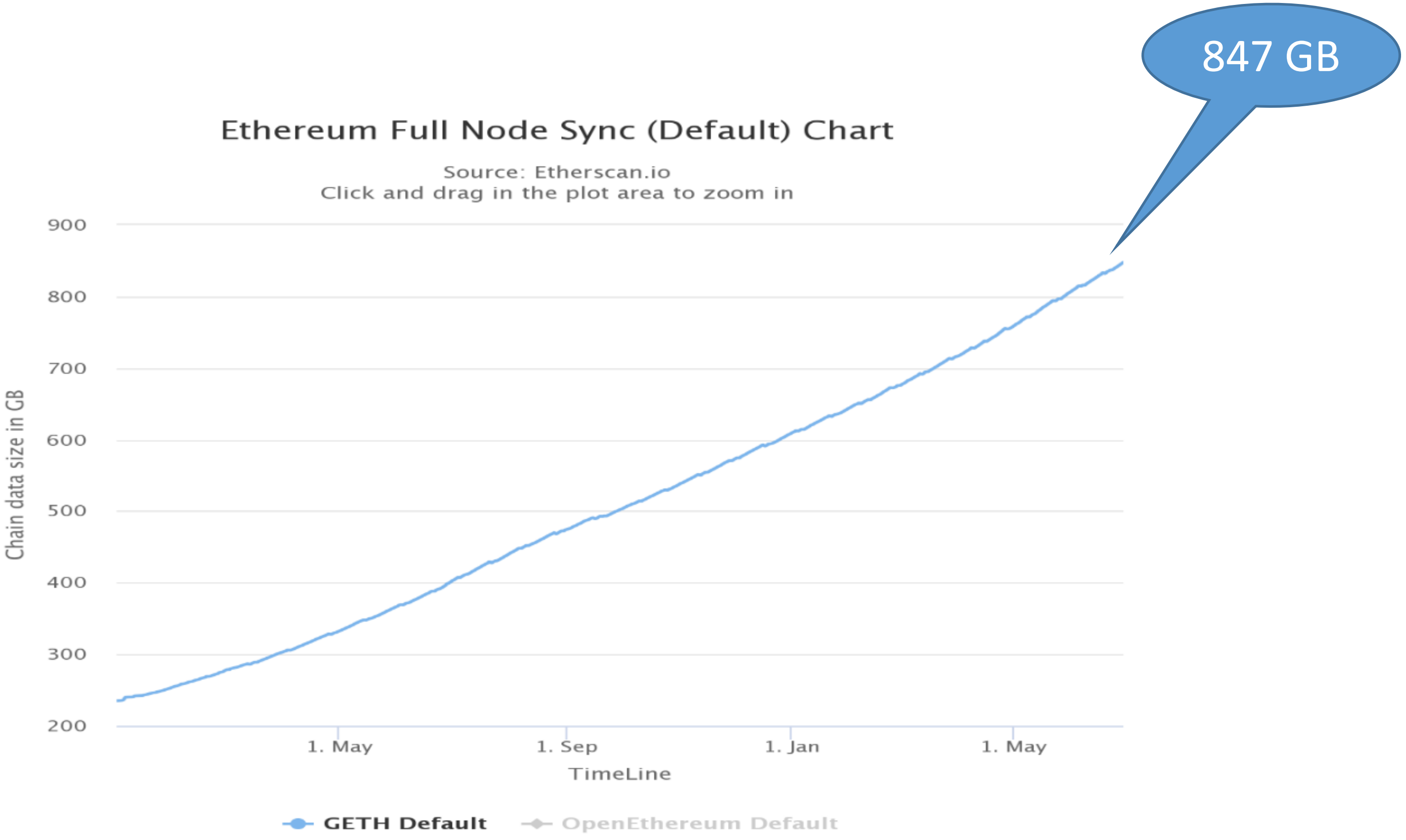


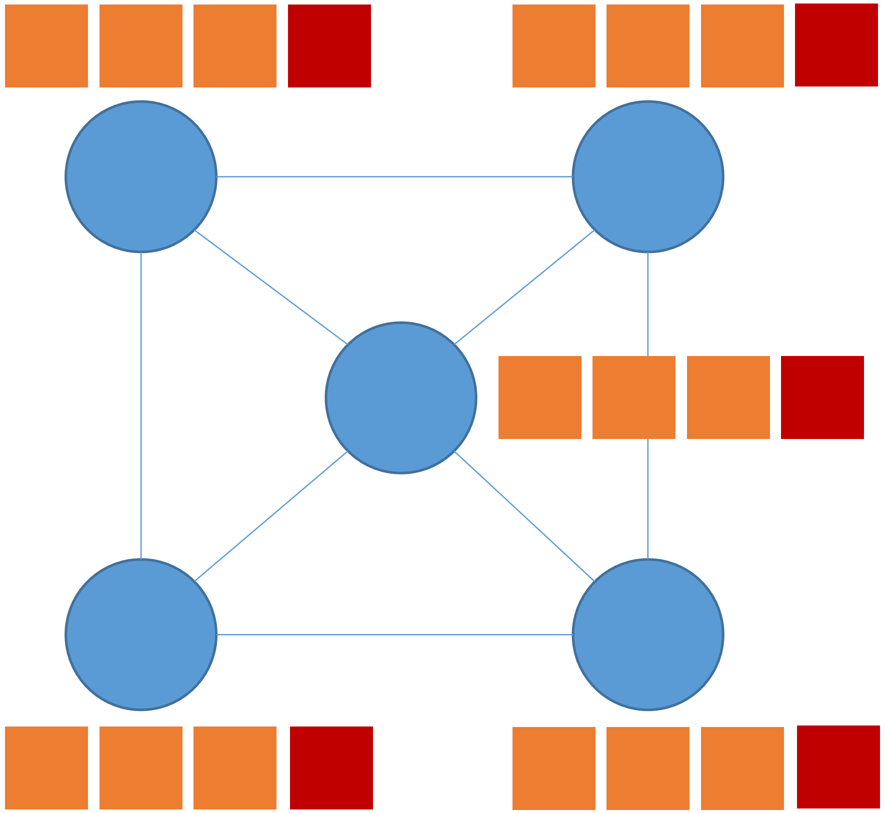
Current Blockchain System

- Features
  - Every node keeps a full replication of transaction history and ledger states
  - Every node validates all transactions in blocks
  - Easy to maintain the same order of transactions
  - Easy to ensure execution integrity
  - Bad for high storage and execution overhead

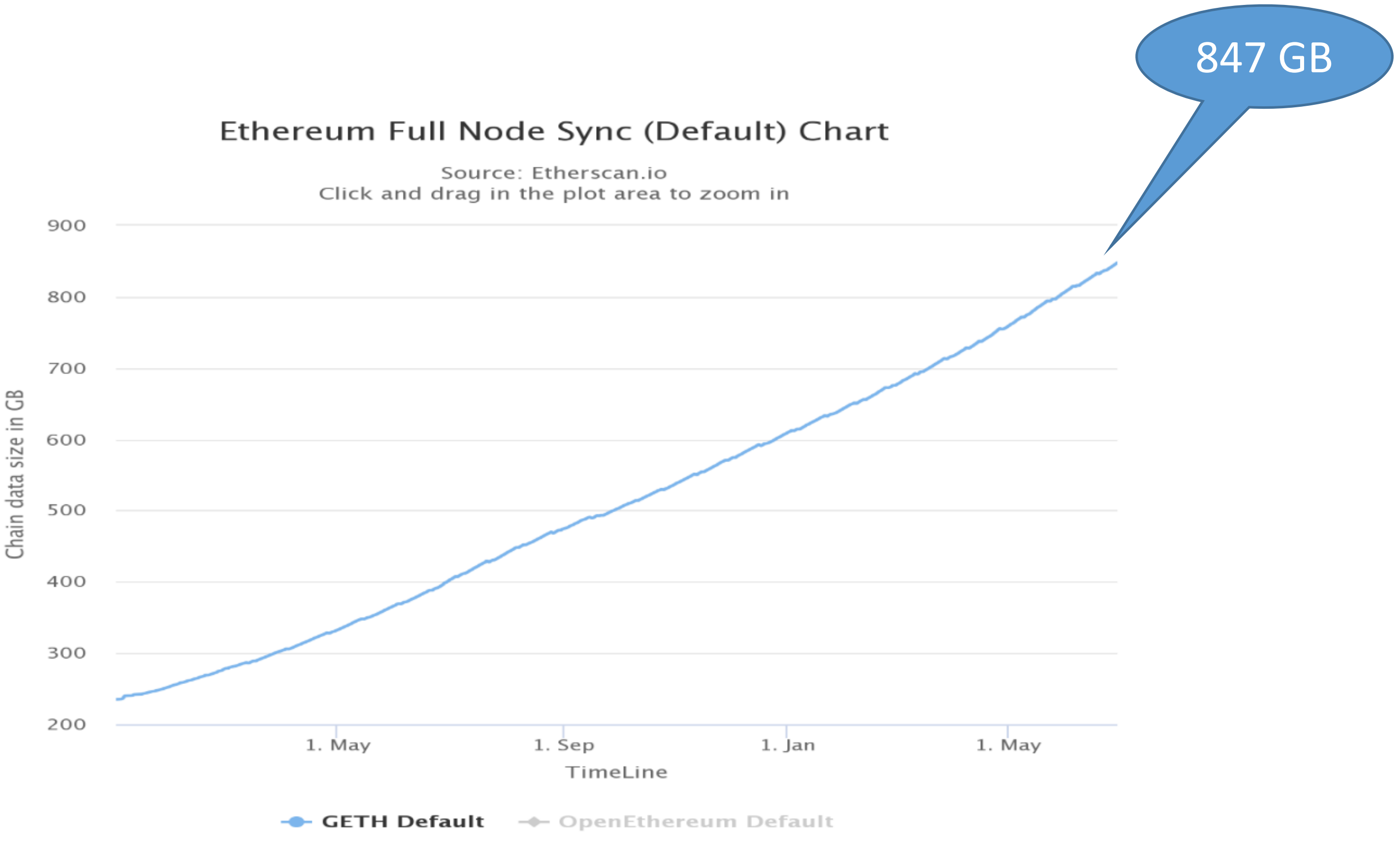


Current Blockchain System



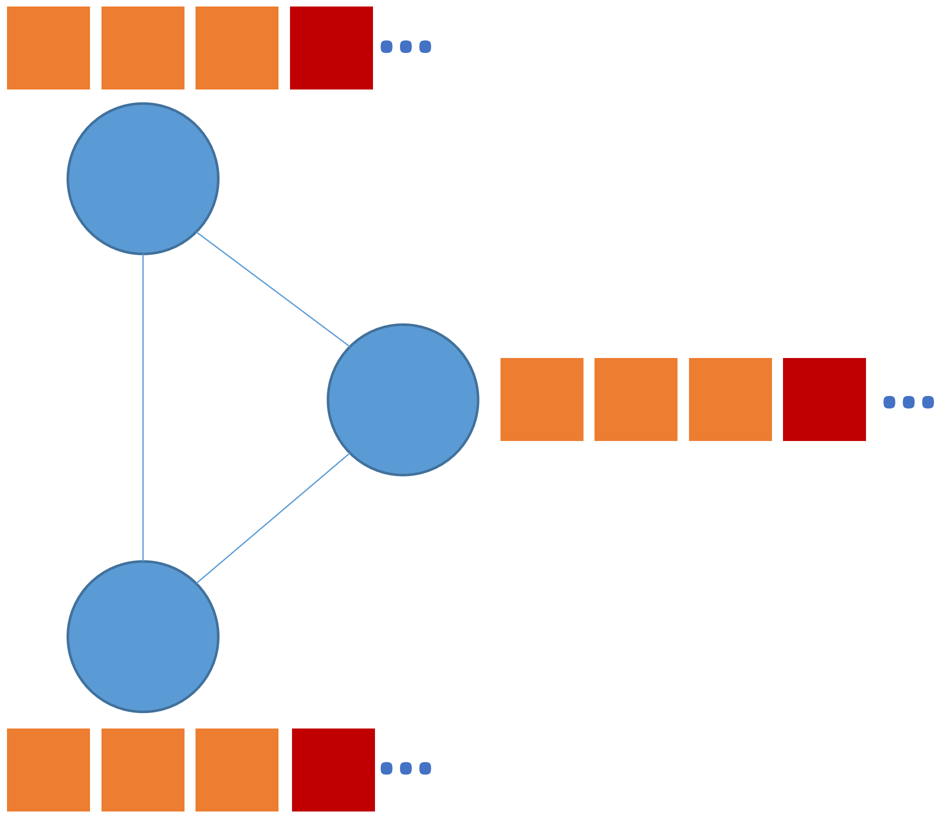


Current Blockchain System


Ethereum Full Node Sync (Default) Chart

Source: Etherscan.io
Click and drag in the plot area to zoom in

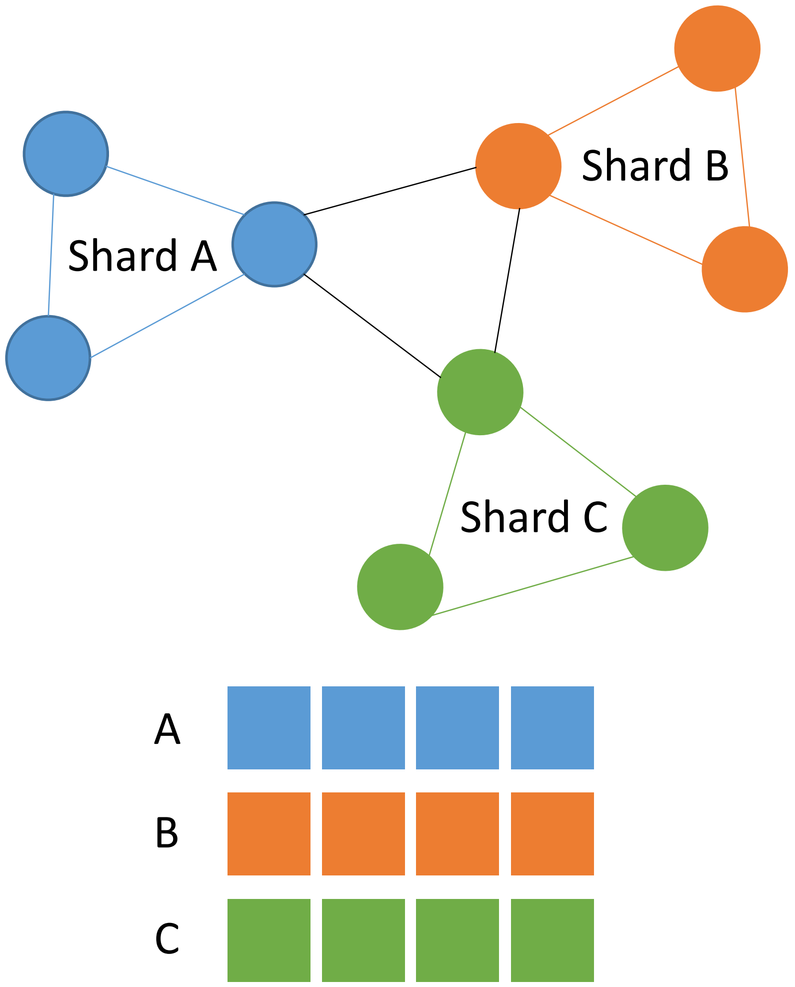
847 GB



Undermine system security and robustness by making the network more centralized!

Possible Solution: Sharding

- General idea [1, 2]
  - Horizontally partition the blockchain into multiple parallel chains
  - Reduce storage and computation duplications among shards
- Drawback
  - Only alleviate the problem by a constant factor (# shards)
  - Introduce new problems (e.g., cross-shard tx)



[1] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. ACM CCS, 2018

[2] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy. BlockchainDB: A shared database on blockchains. VLDB 2019

New Concept: Stateless Blockchain

- General idea [3, 4]
  - Move ledger states and transaction executions off-chain to a subset of nodes
  - Reduce the on-chain overhead
- Drawback
  - Designed particularly for cryptocurrencies
  - Cannot work for general-purpose blockchain that supports smart contracts

[3] A. Chepurnoy, C. Papamanthou, and Y. Zhang. EDRAX: A cryptocurrency with stateless transaction validation. Cryptology ePrint Archive, 2018
[4] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In Annual International Cryptology Conference, 2019

Challenges

Stateless blockchain with smart contracts

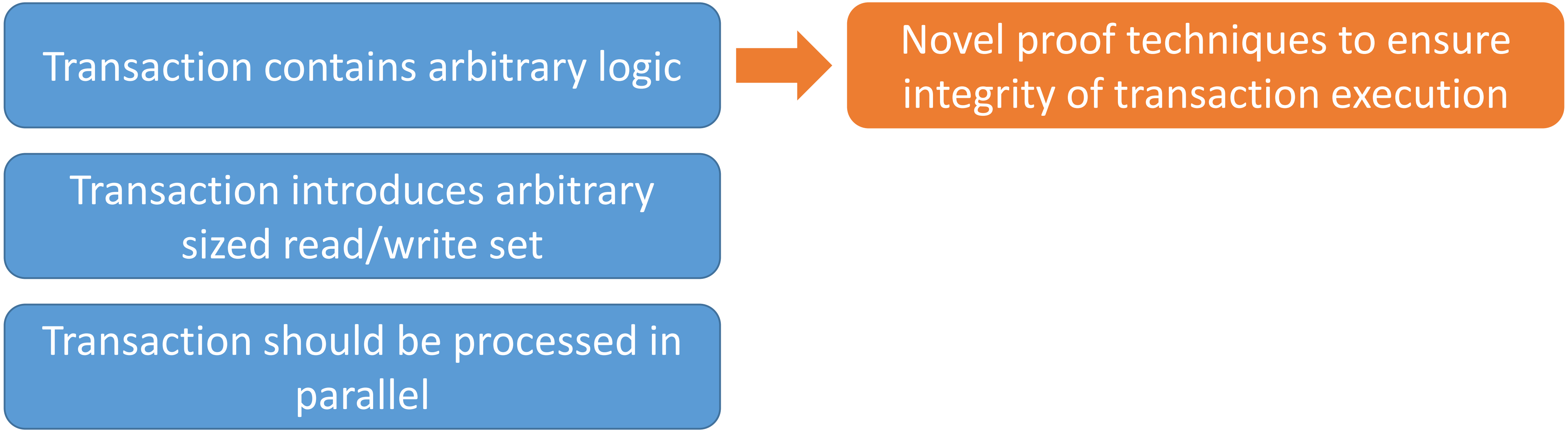
Transaction contains arbitrary logic

Transaction introduces arbitrary sized read/write set

Transaction should be processed in parallel

Challenges

Stateless blockchain with smart contracts



Transaction contains arbitrary logic

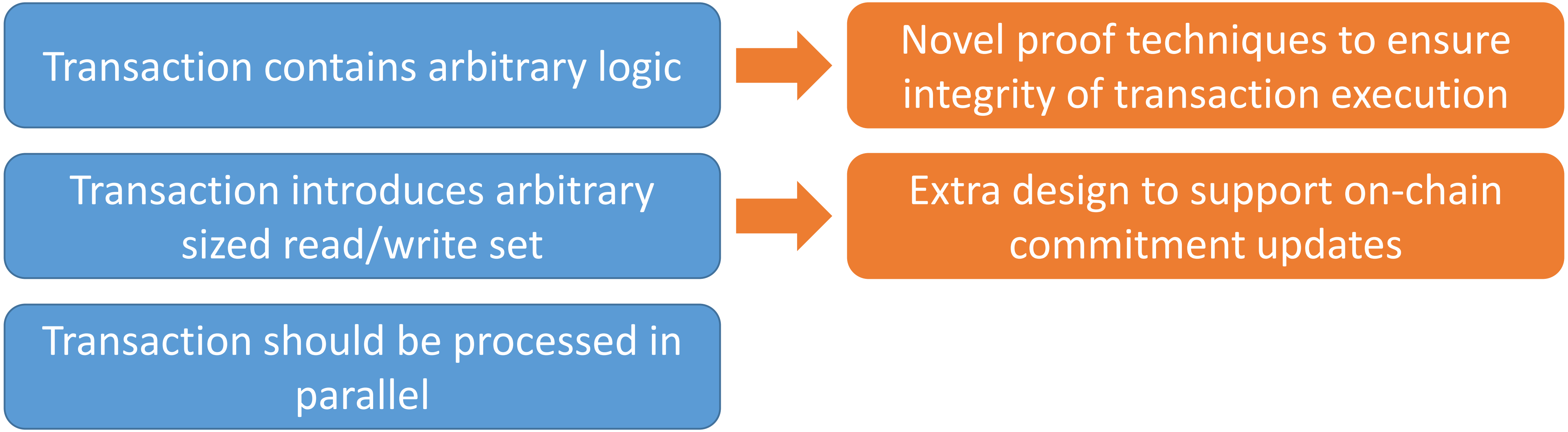
Novel proof techniques to ensure integrity of transaction execution

Transaction introduces arbitrary sized read/write set

Transaction should be processed in parallel

Challenges

Stateless blockchain with smart contracts



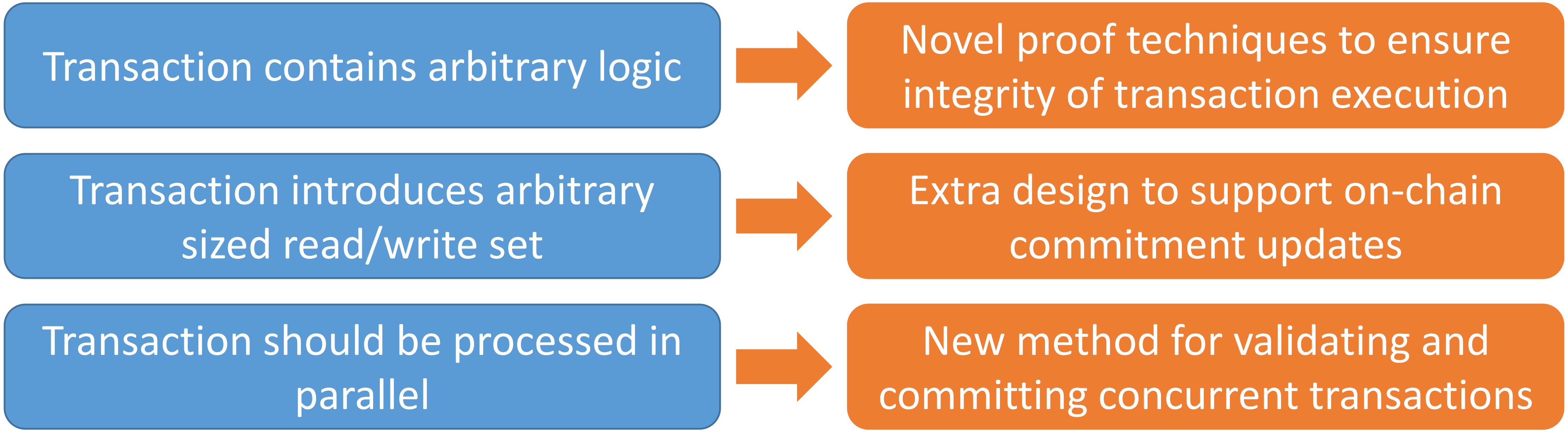
Transaction contains arbitrary logic → Novel proof techniques to ensure integrity of transaction execution

Transaction introduces arbitrary sized read/write set → Extra design to support on-chain commitment updates

Transaction should be processed in parallel

Challenges

Stateless blockchain with smart contracts



Transaction contains arbitrary logic → Novel proof techniques to ensure integrity of transaction execution

Transaction introduces arbitrary sized read/write set → Extra design to support on-chain commitment updates

Transaction should be processed in parallel → New method for validating and committing concurrent transactions

Our Solution: SlimChain

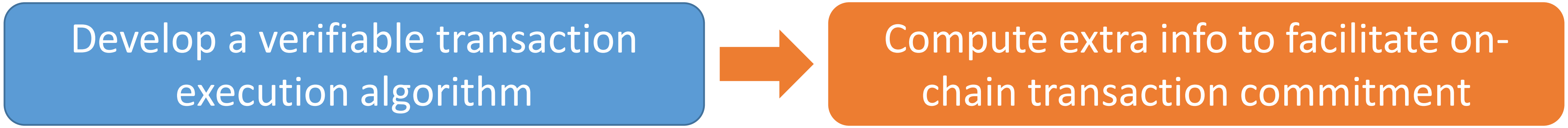
- SlimChain: a stateless blockchain system that scales transactions through off-chain storage and parallel processing

  - Off-chain storage nodes store ledger states and simulate smart contract execution
  - On-chain consensus nodes maintain only the short commitments of ledger states

Our Solution: SlimChain

- SlimChain: a stateless blockchain system that scales transactions through off-chain storage and parallel processing

- Off-chain storage nodes store ledger states and simulate smart contract execution
- On-chain consensus nodes maintain only the short commitments of ledger states



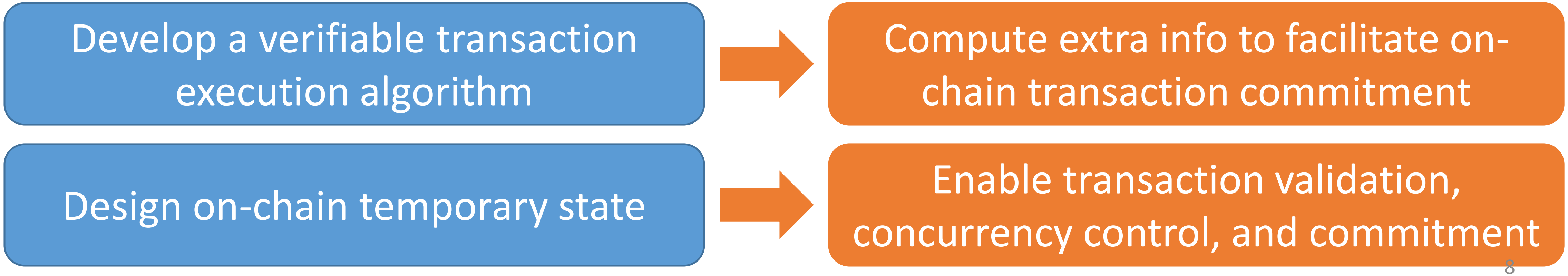
Develop a verifiable transaction execution algorithm

Compute extra info to facilitate on-chain transaction commitment

Our Solution: SlimChain

- SlimChain: a stateless blockchain system that scales transactions through off-chain storage and parallel processing

- Off-chain storage nodes store ledger states and simulate smart contract execution
- On-chain consensus nodes maintain only the short commitments of ledger states



Develop a verifiable transaction execution algorithm

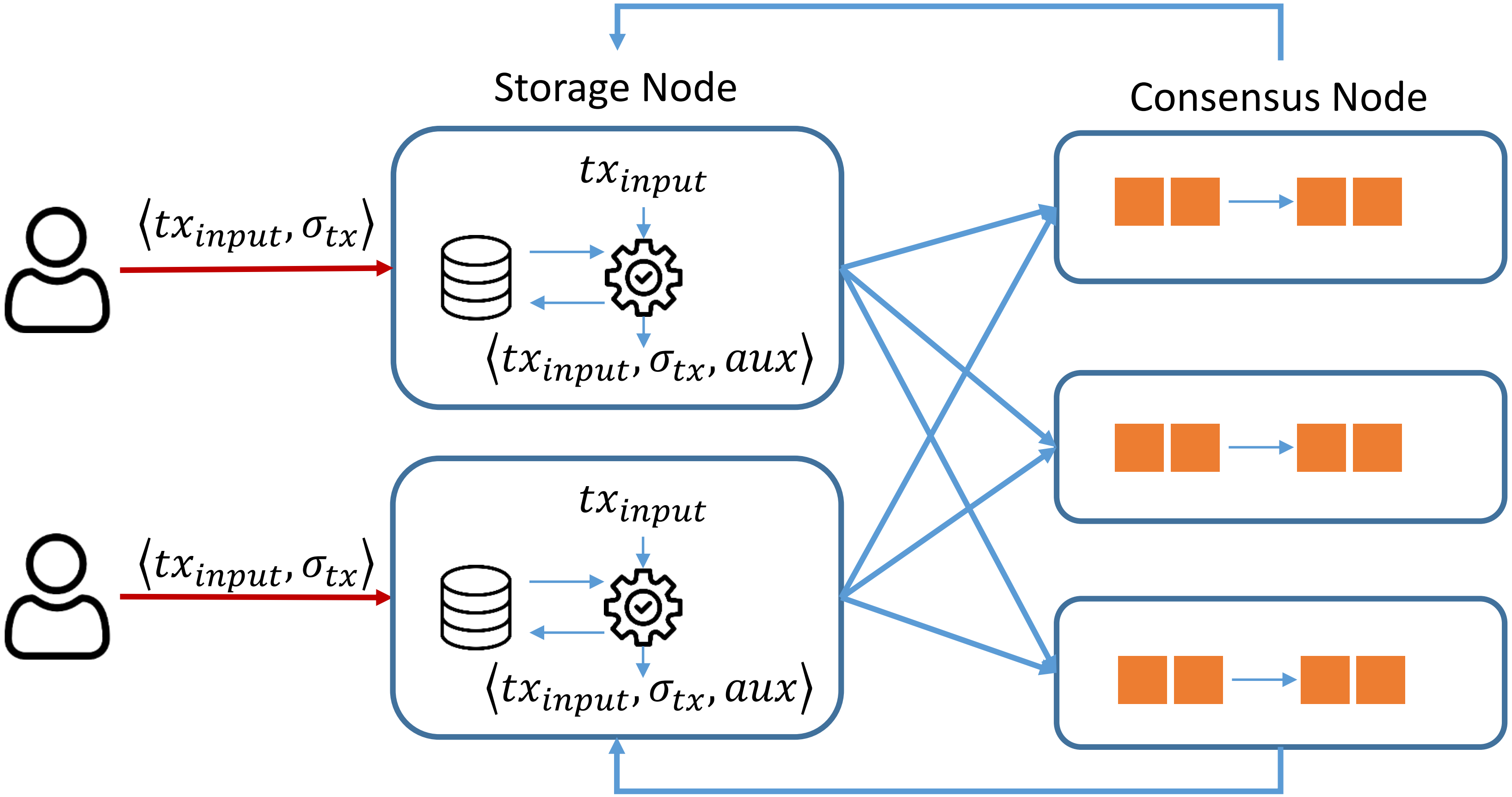
Compute extra info to facilitate on-chain transaction commitment

Design on-chain temporary state

Enable transaction validation, concurrency control, and commitment

SlimChain System Overview

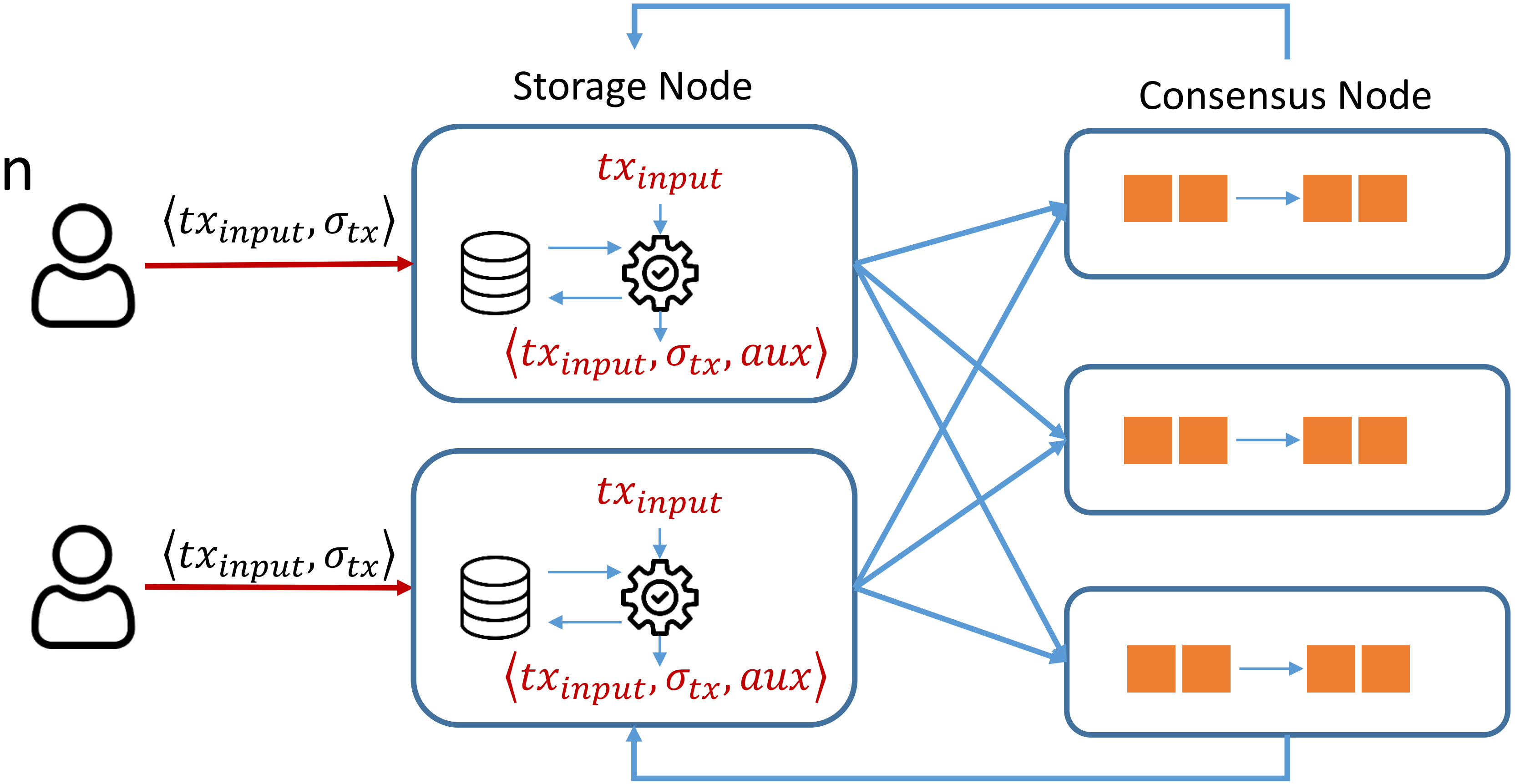
- Send TX



Storage Node
Consensus Node
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$

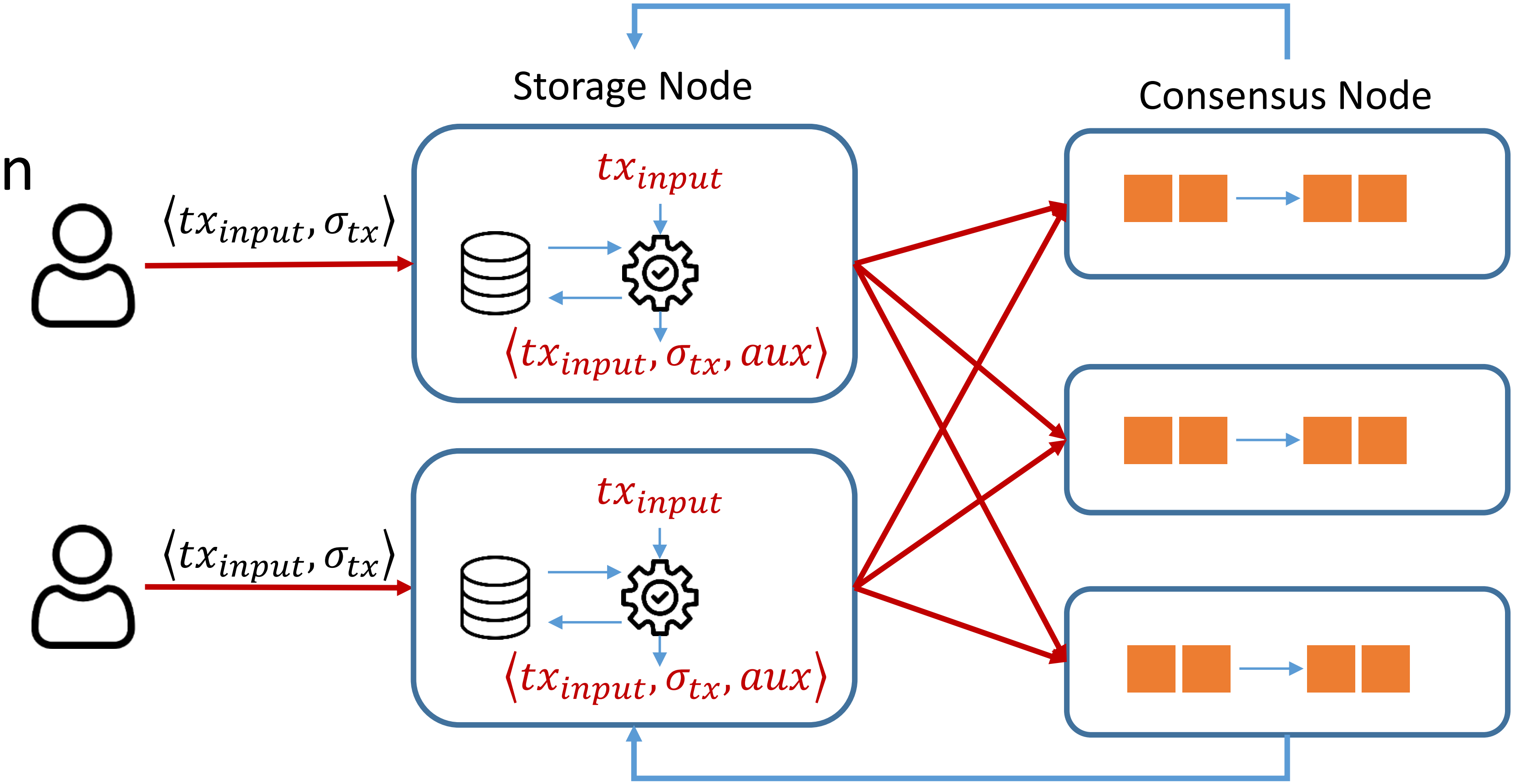
SlimChain System Overview

- Send TX
- Verifiable tx execution


Storage Node
Consensus Node
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$

SlimChain System Overview

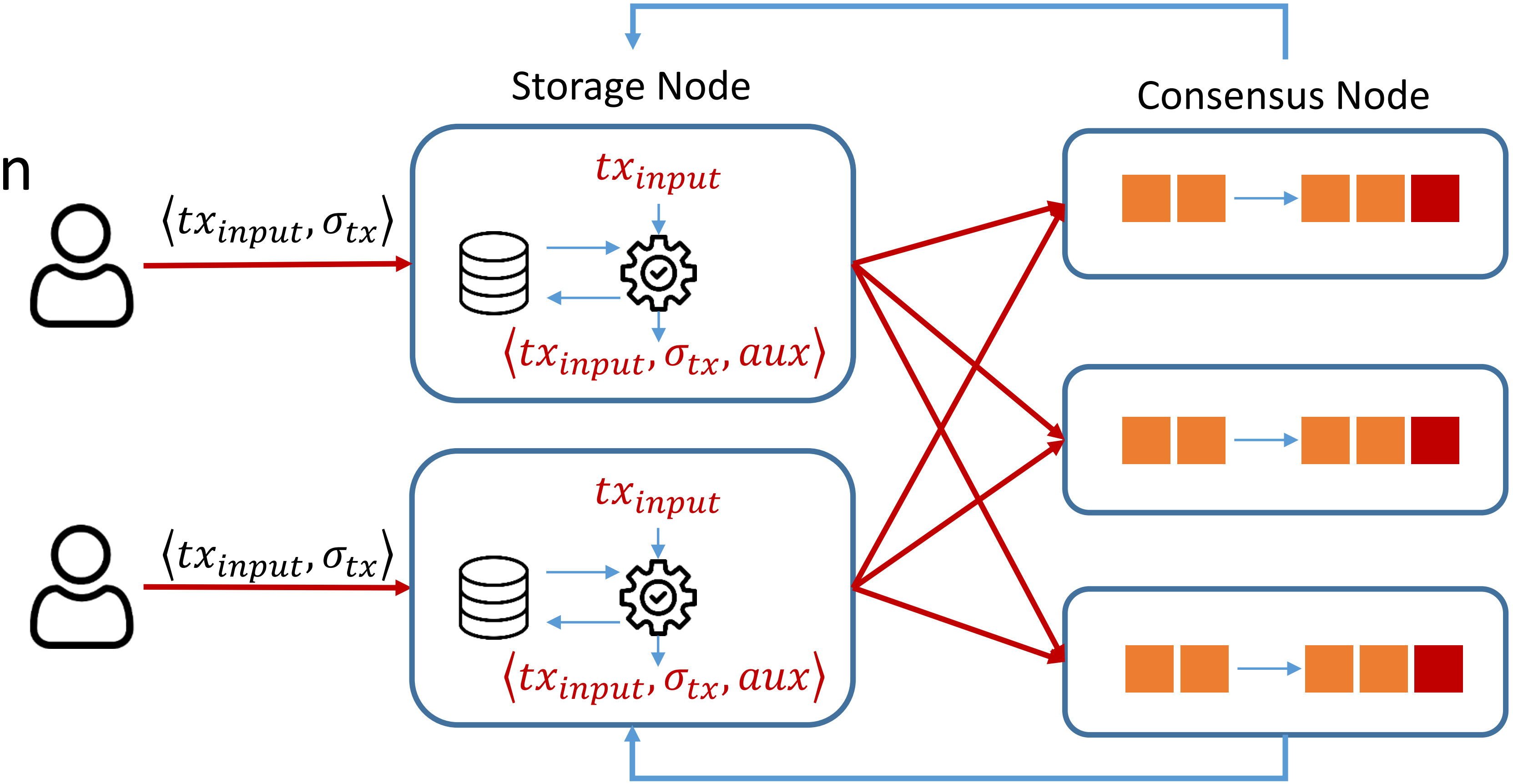
- Send TX
- Verifiable tx execution
- Broadcast



$\langle tx_{input}, \sigma_{tx} \rangle$
$\langle tx_{input}, \sigma_{tx} \rangle$
Storage Node
Consensus Node
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$

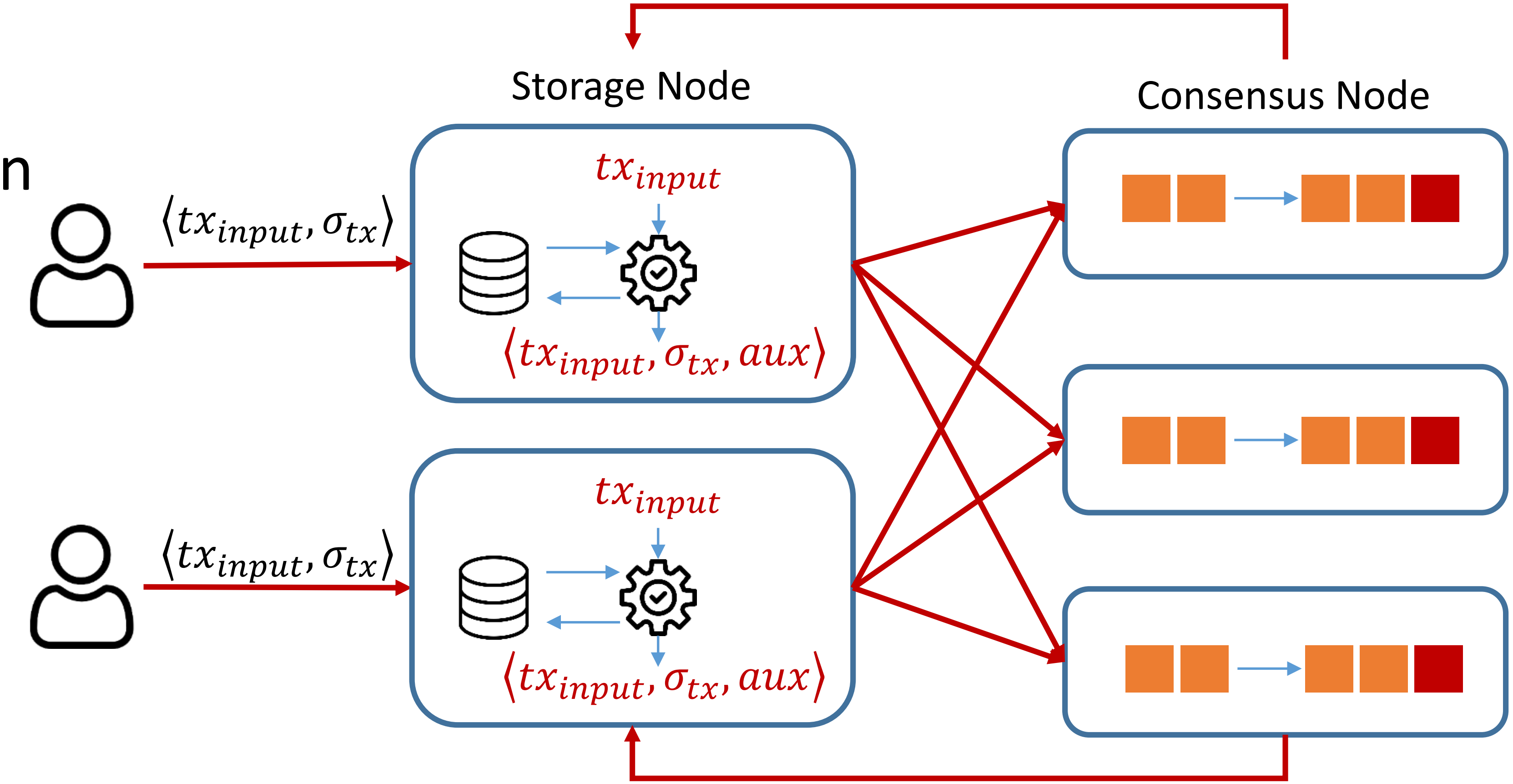
SlimChain System Overview

- Send TX
- Verifiable tx execution
- Broadcast
- Validate & append to ledger


Storage Node
Consensus Node
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$

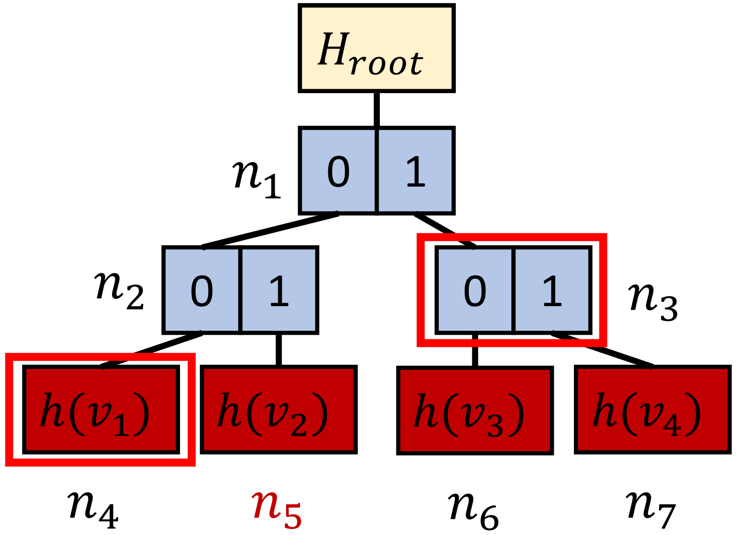
SlimChain System Overview

- Send TX
- Verifiable tx execution
- Broadcast
- Validate & append to ledger
- Synchronize

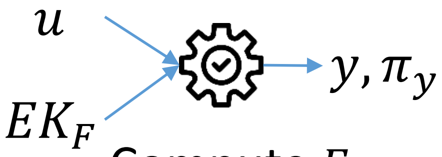

Storage Node
Consensus Node
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$
$\langle tx_{input}, \sigma_{tx} \rangle$
tx_{input}
$\langle tx_{input}, \sigma_{tx}, aux \rangle$

Preliminaries

- Merkle Hash Tree
  - Support verifiable membership testing with logarithmic complexity
  - Hash function combining the child nodes
  - Proof: sibling hashes along the search path
  - Verify: reconstructing the root hash



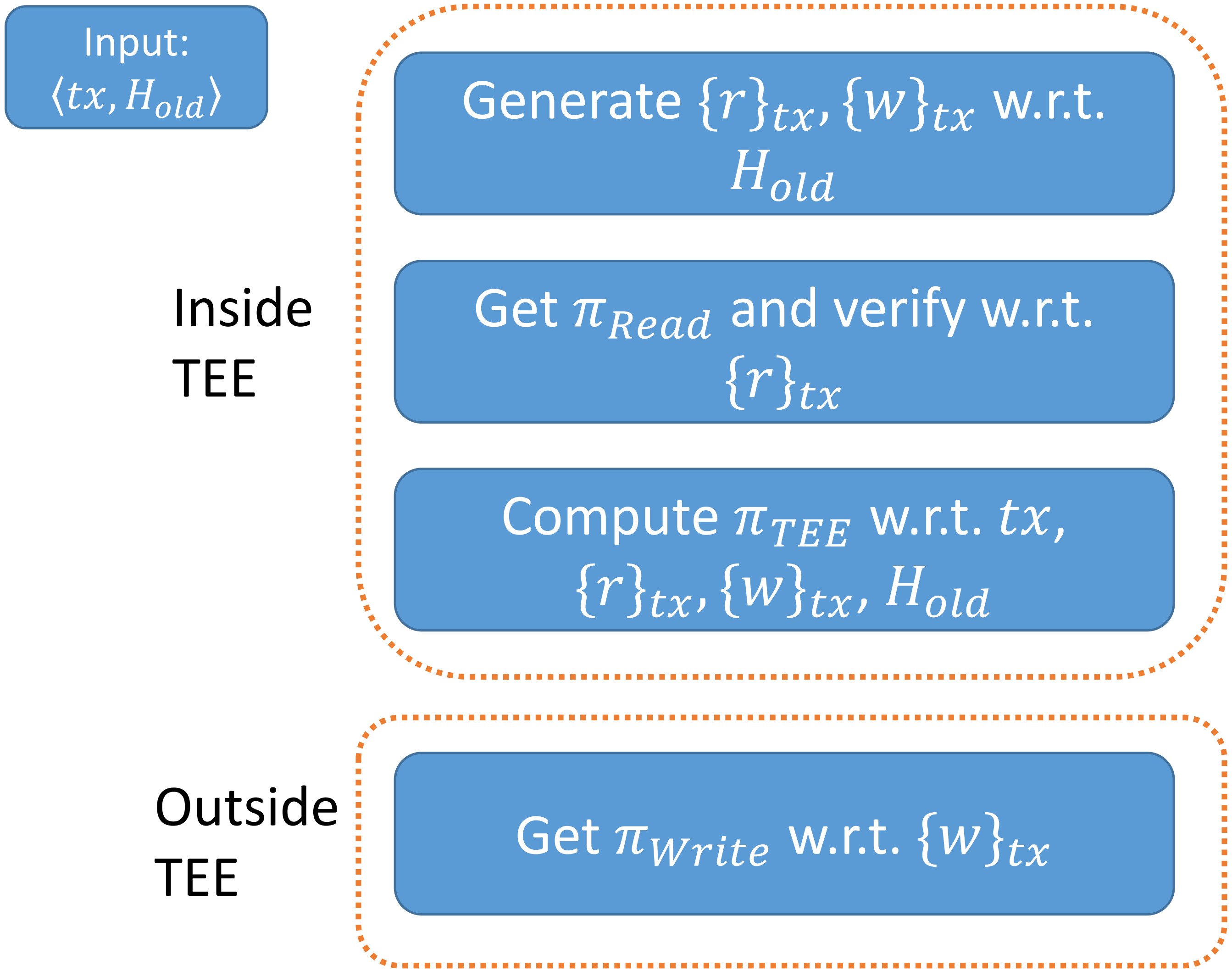
- Verifiable Computing
  - Ensure the integrity of computations performed by untrusted parties
  - One possible implementation: TEE (Intel SGX)



Compute F

$$Verify(VK_F, u, y, \pi_y) = 1$$
$$iff\ F(u) = y$$

Off-chain Transaction Execution



Input: $\langle tx, H_{old} \rangle$

Inside TEE

Generate $\{r\}_{tx}, \{w\}_{tx}$ w.r.t. H_{old}

Get π_{Read} and verify w.r.t. $\{r\}_{tx}$

Compute π_{TEE} w.r.t. $tx, \{r\}_{tx}, \{w\}_{tx}, H_{old}$

Outside TEE

Get π_{Write} w.r.t. $\{w\}_{tx}$

- π_{TEE} ensures execution integrity and read integrity
- $\{r\}_{tx}, \{w\}_{tx}, H_{old}, \pi_{Write}$ provide enough information for on-chain validation and commitment

$$tx_{submit} = \langle tx_{input}, \{r\}_{tx}, \{w\}_{tx}, H_{old}, \pi_{TEE}, \pi_{Write} \rangle$$

On-chain Transaction Commitment

Validate π_{TEE}, π_{Write}

On-chain Transaction Commitment

Validate π_{TEE}, π_{Write}

Check conflict of $\{r\}_{tx}, \{w\}_{tx}$

On-chain Transaction Commitment

Validate π_{TEE}, π_{Write}

Check conflict of $\{r\}_{tx}, \{w\}_{tx}$

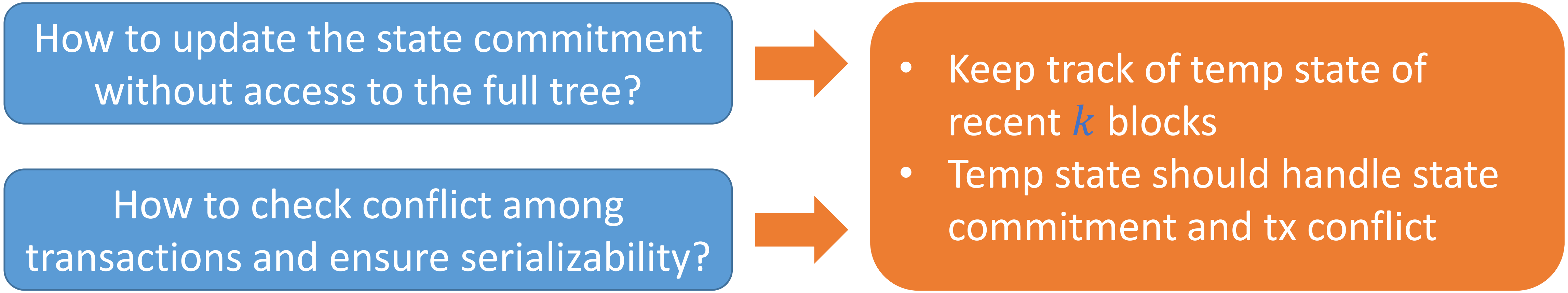
Update ledger state commitment and generate new block

On-chain Transaction Commitment

How to update the state commitment without access to the full tree?

How to check conflict among transactions and ensure serializability?

On-chain Transaction Commitment

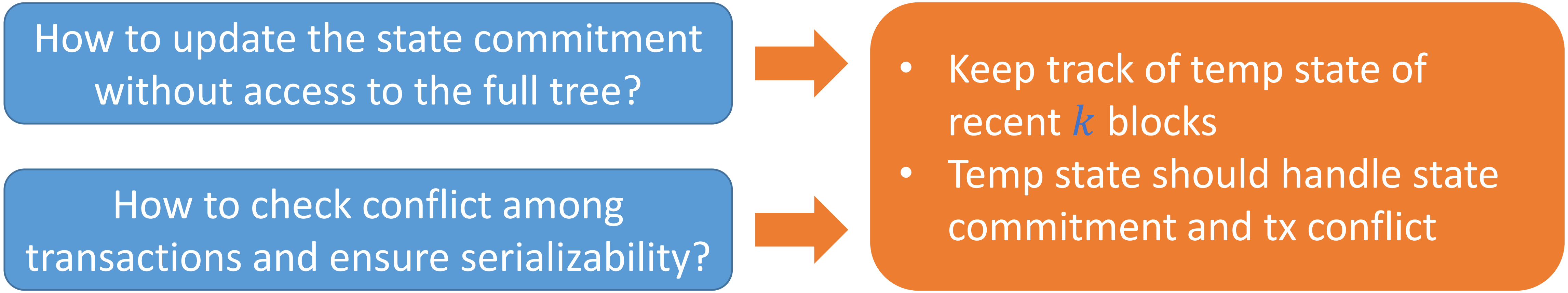


How to update the state commitment without access to the full tree?

How to check conflict among transactions and ensure serializability?

- Keep track of temp state of recent k blocks
- Temp state should handle state commitment and tx conflict

On-chain Transaction Commitment



How to update the state commitment without access to the full tree?

How to check conflict among transactions and ensure serializability?

- Keep track of temp state of recent k blocks
- Temp state should handle state commitment and tx conflict

- Temporary states
  - $\mathcal{T}_w$: a partial Merkle tree w.r.t. the write set in the past k blocks
  - $M_{i \mapsto r}, M_{i \mapsto w}$: map between block height to read, write addresses
  - $M_{r \mapsto i}, M_{w \mapsto i}$: map between read, write addresses to an ordered list of block heights

Conflict Check

- Optimistic Concurrency Control (OCC)
    - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_3

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100:\{10\}$	$100:\{10\}, 101:\{10\}$
$M_{i\mapsto w}$	$100:\{01\}$	$100:\{01\}, 101:\{00\}$
$M_{r\mapsto i}$	$10:\{100\}$	$10:\{100,101\}$
$M_{w\mapsto i}$	$01:\{100\}$	$00:\{101\}, 01:\{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_3

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100: \{10\}$	$100: \{10\}, 101: \{10\}$
$M_{i\mapsto w}$	$100: \{01\}$	$100: \{01\}, 101: \{00\}$
$M_{r\mapsto i}$	$10: \{100\}$	$10: \{100{,}101\}$
$M_{w\mapsto i}$	$01: \{100\}$	$00: \{101\}, 01: \{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}

Check r_{tx} and $M_{w\mapsto i}$ → $10 \notin M_{w\mapsto i}$

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_3

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100{:}\{10\}$	$100{:}\{10\}, 101{:}\{10\}$
$M_{i\mapsto w}$	$100{:}\{01\}$	$100{:}\{01\}, 101{:}\{00\}$
$M_{r\mapsto i}$	$10{:}\{100\}$	$10{:}\{100{,}101\}$
$M_{w\mapsto i}$	$01{:}\{100\}$	$00{:}\{101\}, 01{:}\{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01{:}v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00{:}v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10{:}v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11{:}v_7\}$	H_{100}

Check r_{tx} and $M_{w\mapsto i}$ → $10 \notin M_{w\mapsto i}$

Check w_{tx} and $M_{w\mapsto i}$ → $10 \notin M_{w\mapsto i}$

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_3

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100{:}\{10\}$	$100{:}\{10\}, 101{:}\{10\}$
$M_{i\mapsto w}$	$100{:}\{01\}$	$100{:}\{01\}, 101{:}\{00\}$
$M_{r\mapsto i}$	$10{:}\{100\}$	$10{:}\{100{,}101\}$
$M_{w\mapsto i}$	$01{:}\{100\}$	$00{:}\{101\}, 01{:}\{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01{:}\,v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00{:}\,v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10{:}\,v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11{:}\,v_7\}$	H_{100}

Check r_{tx} and $M_{w\mapsto i}$ → $10 \notin M_{w\mapsto i}$

Check w_{tx} and $M_{w\mapsto i}$ → $10 \notin M_{w\mapsto i}$

tx_3 is valid!

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_4

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i \mapsto r}$	$100: \{10\}$	$100: \{10\}, 101: \{10\}$
$M_{i \mapsto w}$	$100: \{01\}$	$100: \{01\}, 101: \{00\}$
$M_{r \mapsto i}$	$10: \{100\}$	$10: \{100, 101\}$
$M_{w \mapsto i}$	$01: \{100\}$	$00: \{101\}, 01: \{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_4

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100\colon\{10\}$	$100\colon\{10\}, 101\colon\{10\}$
$M_{i\mapsto w}$	$100\colon\{01\}$	$100\colon\{01\}, 101\colon\{00\}$
$M_{r\mapsto i}$	$10\colon\{100\}$	$10\colon\{100{,}101\}$
$M_{w\mapsto i}$	$01\colon\{100\}$	$00\colon\{101\}, 01\colon\{100\}$

TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01\colon v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00\colon v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10\colon v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11\colon v_7\}$	H_{100}

Check r_{tx} and $M_{w\mapsto i}$ → $101 > 100$

Conflict Check

- Optimistic Concurrency Control (OCC)
  - Check whether other committed transactions have modified the data that the current transaction accessed (read or wrote)

Height 101, Check tx_4

Block id	100	101
TX List	$\{tx_1\}$	$\{tx_2\}$
$M_{i\mapsto r}$	$100\!:\{10\}$	$100\!:\{10\}, 101\!:\{10\}$
$M_{i\mapsto w}$	$100\!:\{01\}$	$100\!:\{01\}, 101\!:\{00\}$
$M_{r\mapsto i}$	$10\!:\{100\}$	$10\!:\{100{,}101\}$
$M_{w\mapsto i}$	$01\!:\{100\}$	$00\!:\{101\}, 01\!:\{100\}$

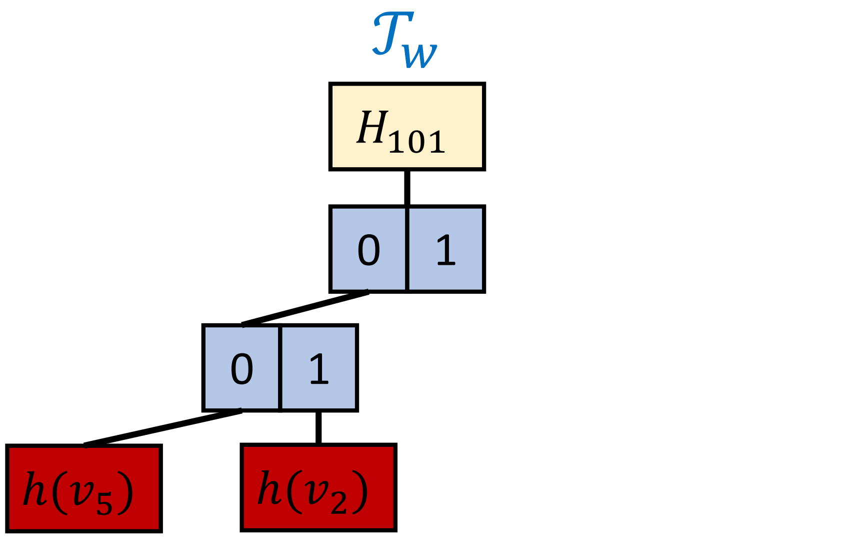
TX	r_{tx}	w_{tx}	H_{old}
tx_1	$\{10\}$	$\{01\!:v_2\}$	H_{99}
tx_2	$\{10\}$	$\{00\!:v_5\}$	H_{99}
tx_3	$\{10\}$	$\{10\!:v_6\}$	H_{100}
tx_4	$\{00\}$	$\{11\!:v_7\}$	H_{100}

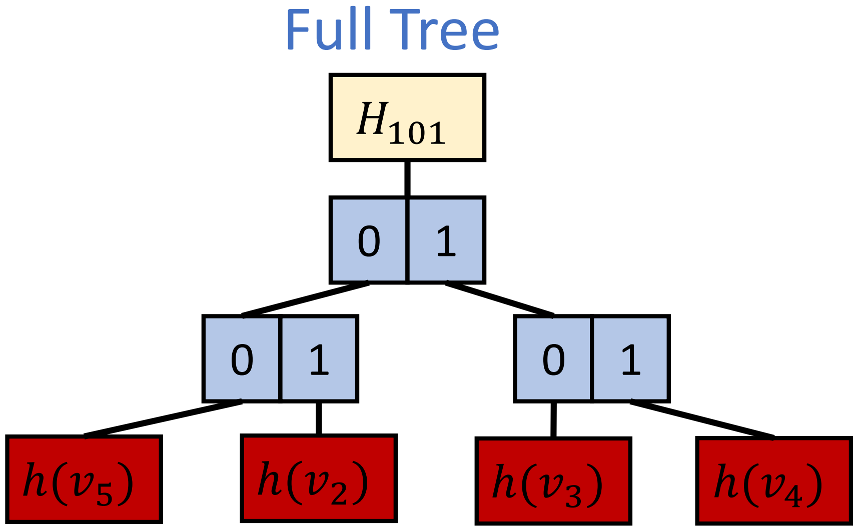
Check r_{tx} and $M_{w\mapsto i}$ → $101 > 100$

tx_4 reads 00 during $block_{100}$
00 is written by tx_2 committed in $block_{101}$
tx_4 is invalid under OCC!

Partial Merkle Tree $\mathcal{T}_w$

- Features of $\mathcal{T}_w$
  - Enable the consensus node to update the state root digest without accessing the full Merkle tree
  - Only the tree nodes corresponding to the written values happening in the past k blocks as well as their Merkle paths are materialized
- Maintenance of $\mathcal{T}_w$
  - Update operation: take the Merkle proof π_{write} and write set $\{w\}_{tx}$ to apply the writes from the transaction
  - Tidy operation: remove the write addresses whose age is more than k blocks





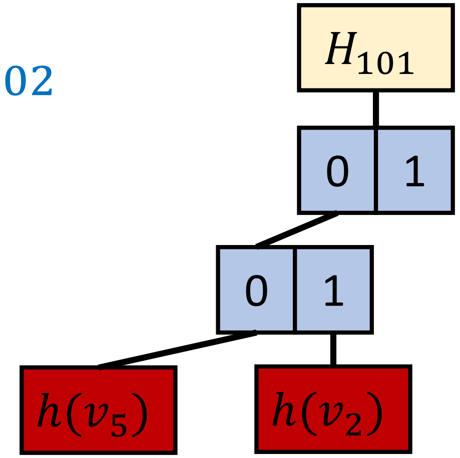
Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

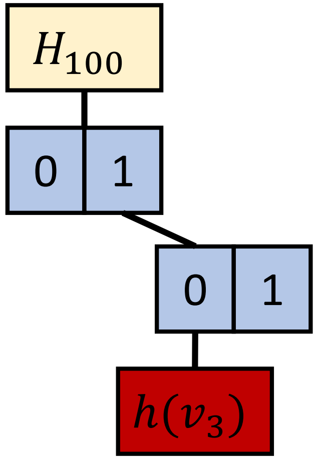
Height: 101

Insert tx_3 to $Block_{102}$

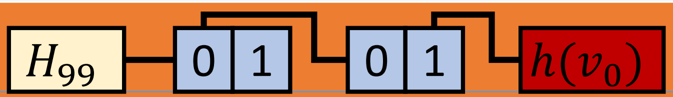
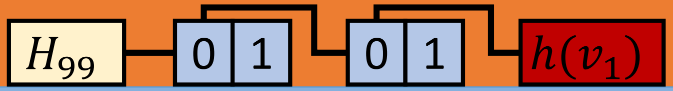
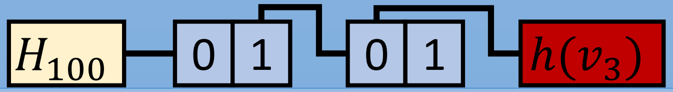
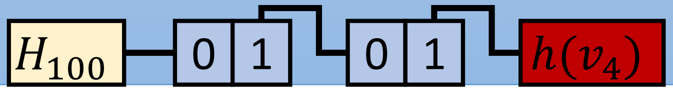
$\mathcal{T}_w$



π_{write} of tx_3



$\boldsymbol{w_{tx}}$ $\{10: v_6\}$

TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	

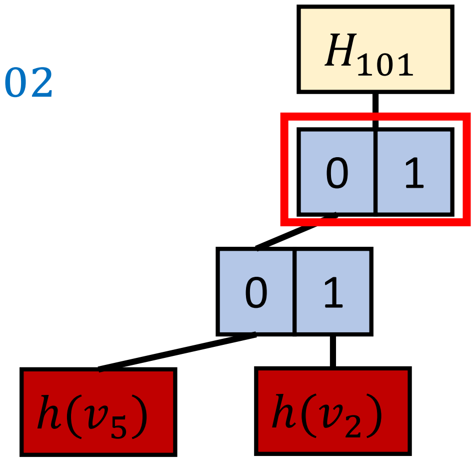
Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

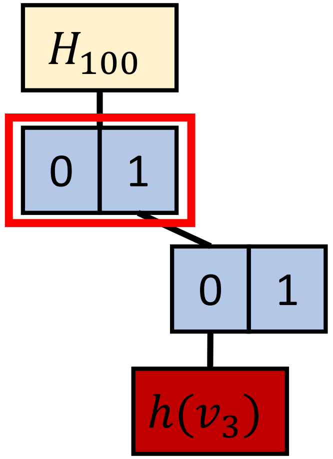
Height: 101

Insert tx_3 to $Block_{102}$

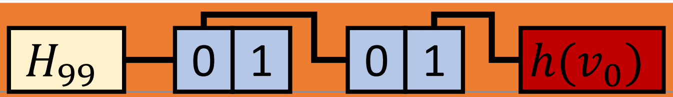
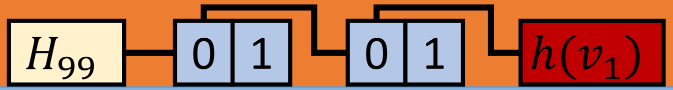
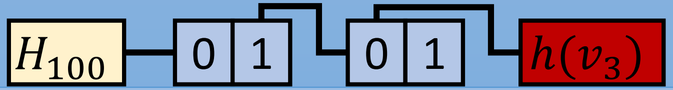
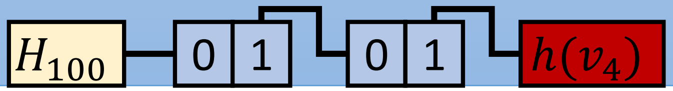
$\mathcal{T}_w$



π_{write} of tx_3



$\boldsymbol{w_{tx}}$ $\{10: v_6\}$

TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_0)$
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_1)$
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_3)$
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_4)$

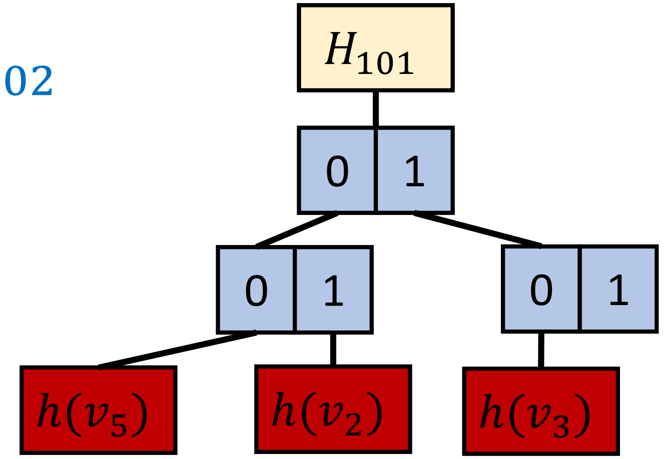
Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

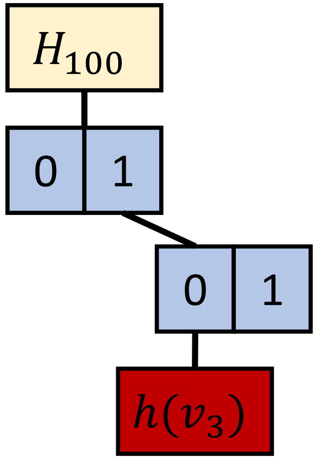
Height: 101

Insert tx_3 to $Block_{102}$

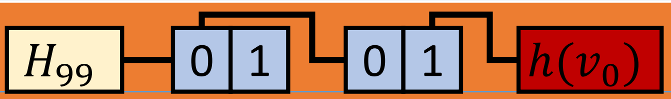
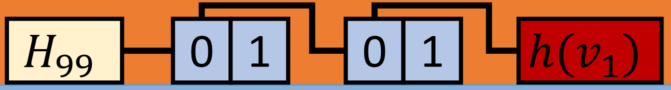
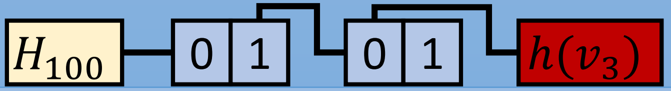
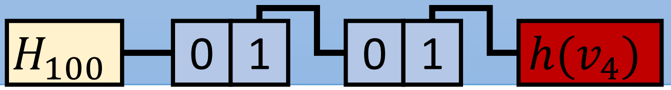
$\mathcal{T}_w$



π_{write} of tx_3



$\boldsymbol{w_{tx}}$ $\{10\!: v_6\}$

TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01\!: v_2\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_0)$
tx_2	$\{10\}$	$\{00\!: v_5\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_1)$
tx_3	$\{10\}$	$\{10\!: v_6\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_3)$
tx_4	$\{00\}$	$\{11\!: v_7\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_4)$

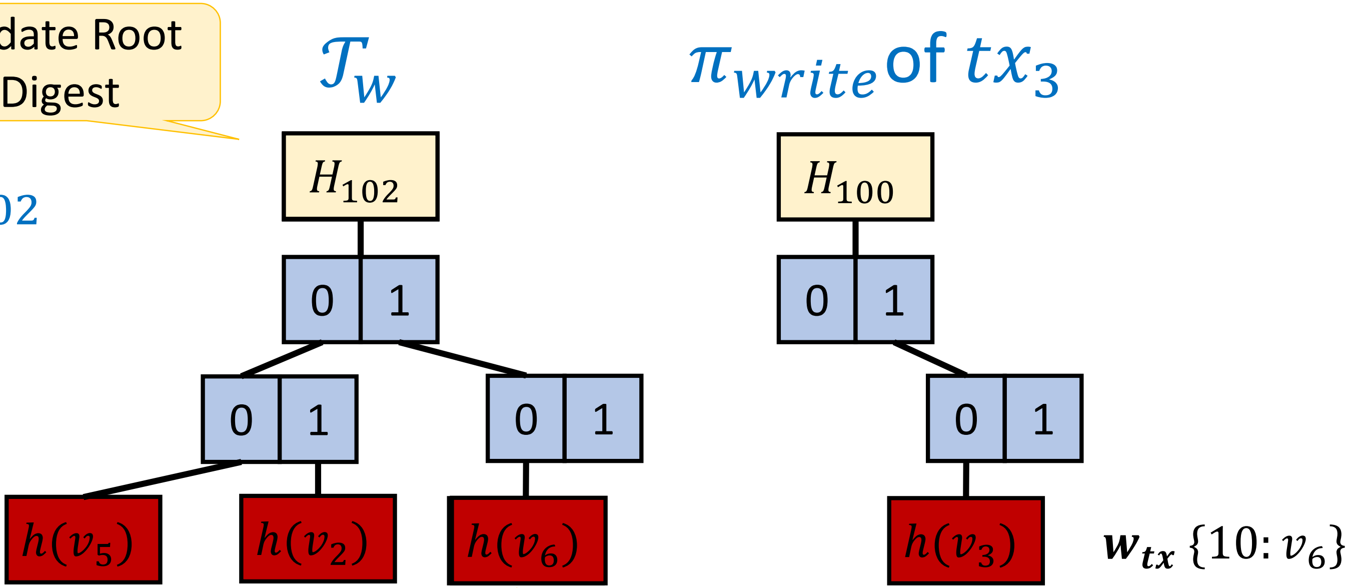
Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

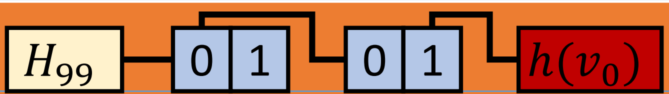
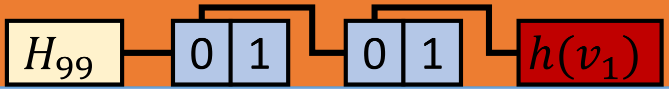
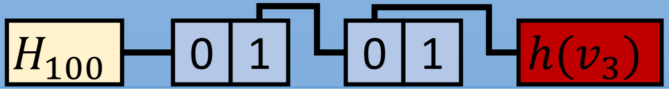
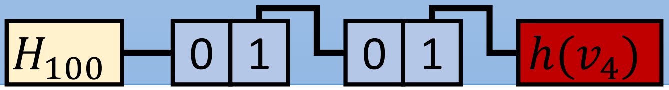
Height: 101

Insert tx_3 to $Block_{102}$

Update Root Digest



Apply write value

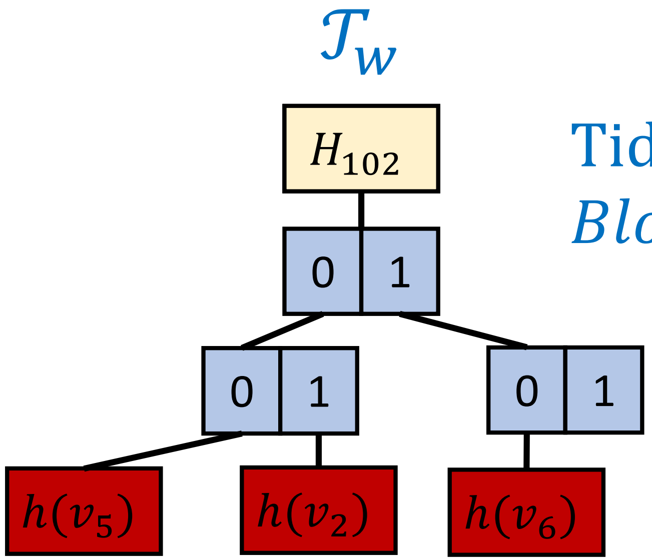
TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	

Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

Height: 102

Remove write addr



Tidy:

$Block_{100}$ contains write set: $\{01: v_2\}$

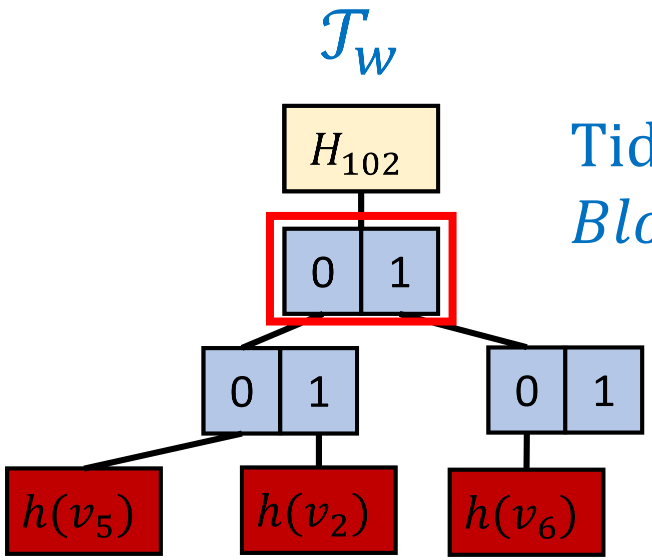
TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	H_{99} 0 1 0 1 $h(v_0)$
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	H_{99} 0 1 0 1 $h(v_1)$
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	H_{100} 0 1 0 1 $h(v_3)$
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	H_{100} 0 1 0 1 $h(v_4)$

Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

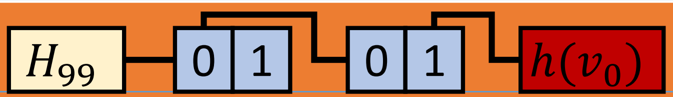
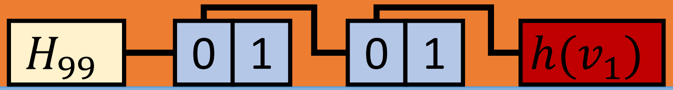
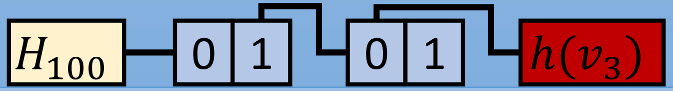
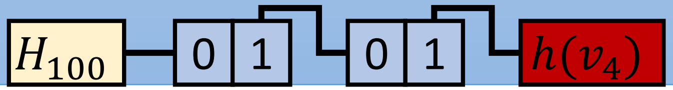
Height: 102

Remove write addr



Tidy:

$Block_{100}$ contains write set: $\{01: v_2\}$

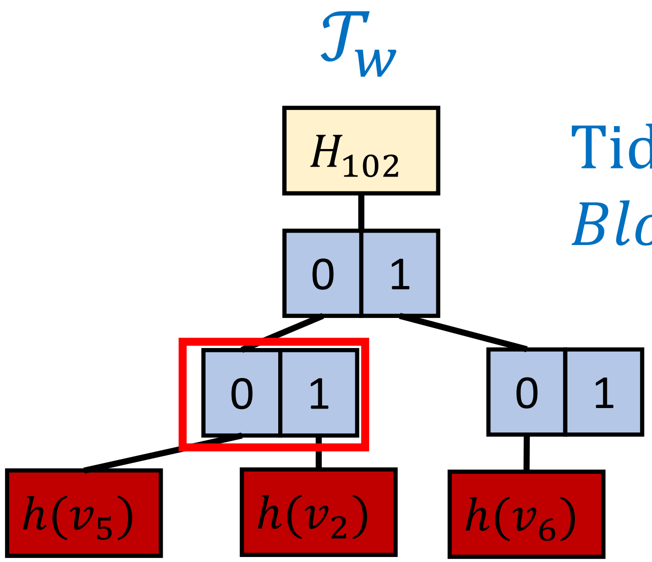
TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_0)$
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	 H_{99} 0 1 0 1 $h(v_1)$
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_3)$
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	 H_{100} 0 1 0 1 $h(v_4)$

Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

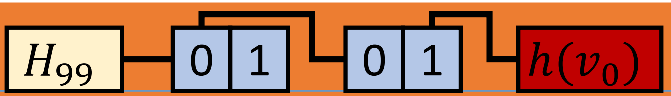
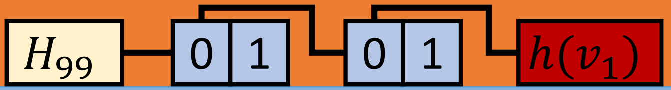
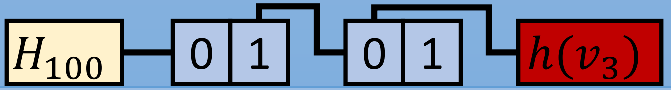
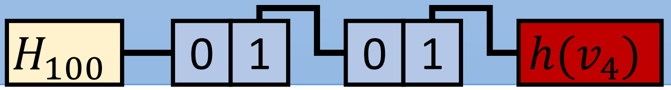
Height: 102

Remove write addr



Tidy:

$Block_{100}$ contains write set: $\{01: v_2\}$

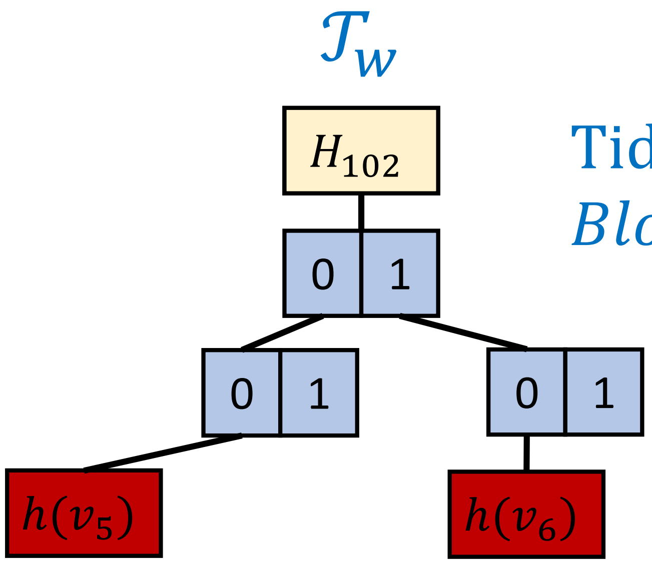
TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	

Partial Merkle Tree $\mathcal{T}_w$

$k = 2$

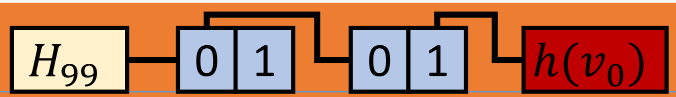
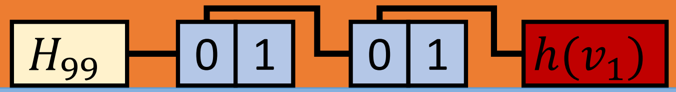
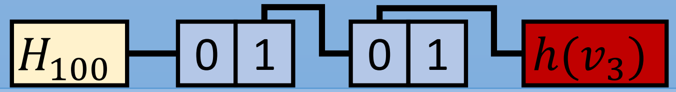
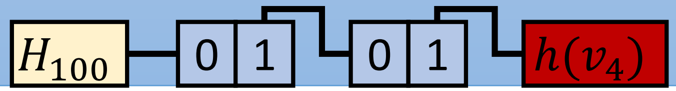
Height: 102

Remove write addr



Tidy:

$Block_{100}$ contains write set: $\{01: v_2\}$

TX	r_{tx}	w_{tx}	H_{old}	π_{write}
tx_1	$\{10\}$	$\{01: v_2\}$	H_{99}	
tx_2	$\{10\}$	$\{00: v_5\}$	H_{99}	
tx_3	$\{10\}$	$\{10: v_6\}$	H_{100}	
tx_4	$\{00\}$	$\{11: v_7\}$	H_{100}	

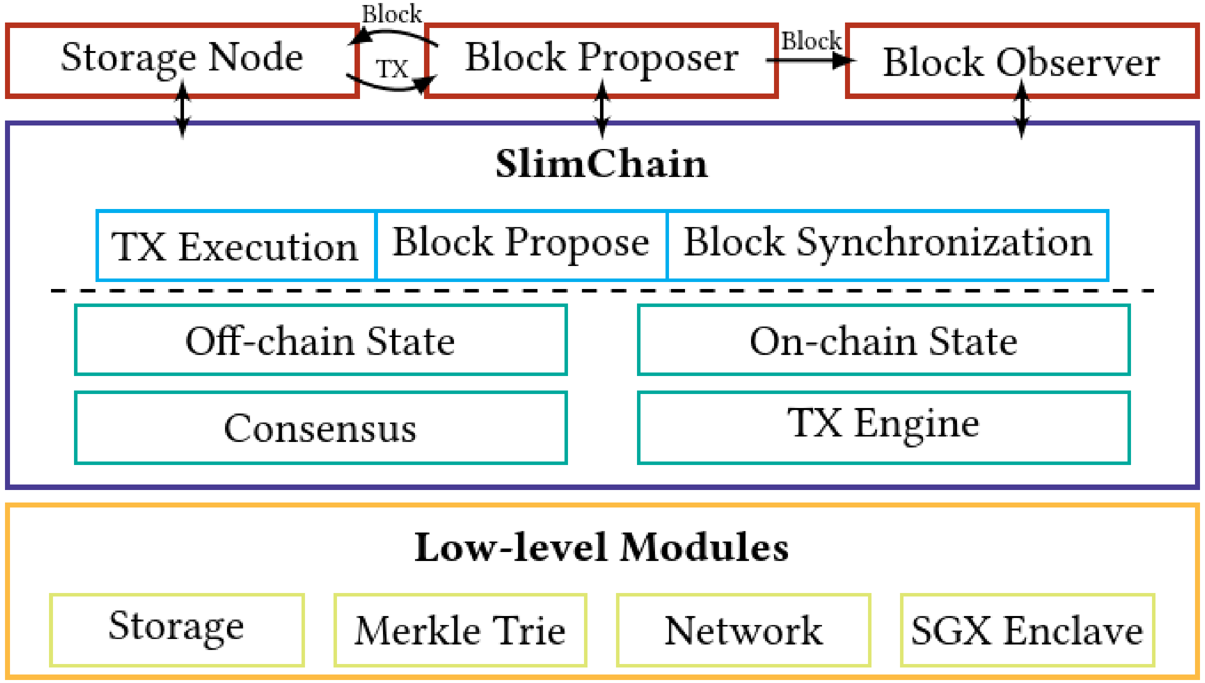
Node Synchronization

- Block Observer
  - Validate and log blocks created by the block proposers

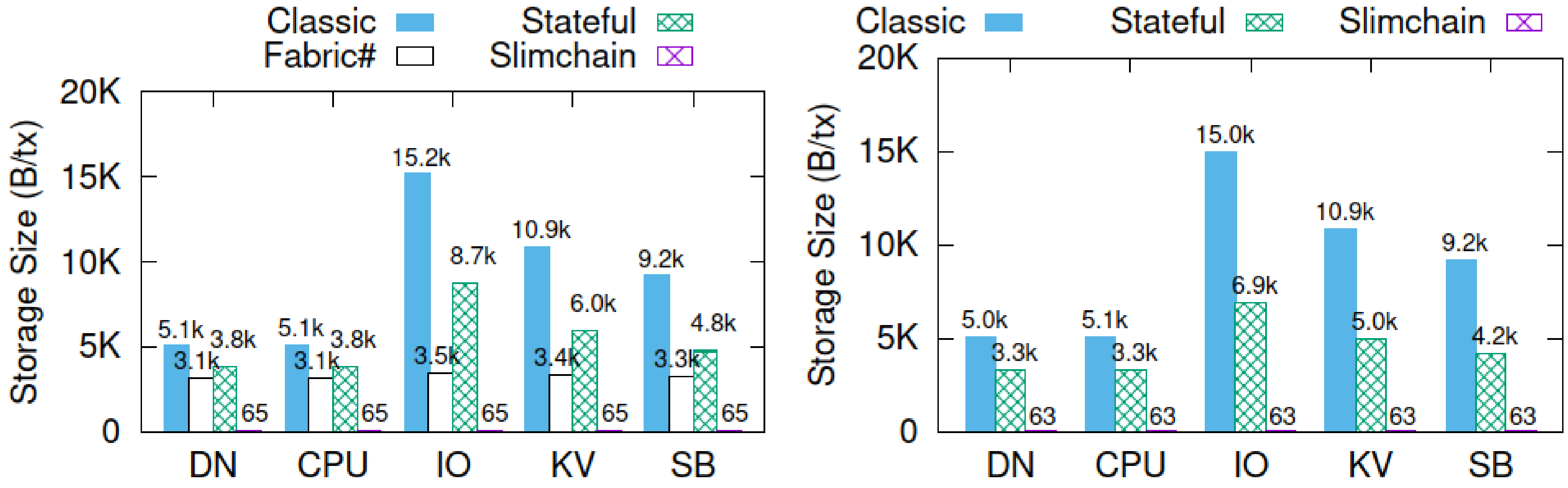
- Storage Node
  - Execute a similar procedure as on-chain transaction commitment
  - Keep transaction data and state data
  - Maintain full Merkle tree instead of partial tree $\mathcal{T}_w$

Implementation

- Implement in Rust program language (LOC: 26,000)
- Two consensus protocols are implemented: PoW, Raft
- Source code available at
  - http://git.io/slimchain



Consensus Node Storage Size

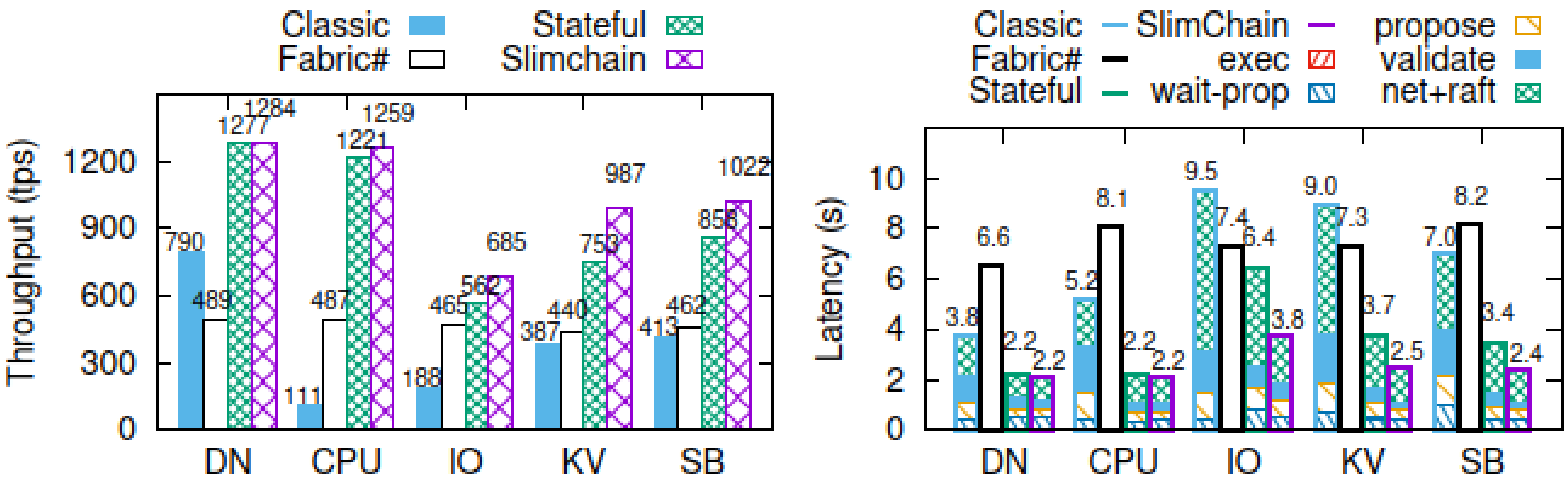


(a) Smart Contract (Permissioned)

(b) Smart Contract (Permissionless)

- SlimChain reduces on-chain storage requirements for consensus nodes by 97%-99%
- The on-chain storage size of SlimChain remains constant regardless of smart contracts

System Throughput and Latency



(a) Smart Contract

(b) Smart Contract

- SlimChain achieves the highest throughput
  - 1.6X-11.3X against Classic
  - 1.4X-2.6X against FabricSharp
- SlimChain has the lowest latency

Thanks

Q&A