Lecture 14: Layer 2 Scaling: Rollups

https://web3.princeton.edu/principles-of-blockchains/

Professor Pramod Viswanath
Princeton University

This lecture:
Layer 2 Scaling -- no need to change the consensus;
Rollups – optimistic rollups and zk rollups

Scaling: Recap

- Ethereum's limited resources
  - Computation
  - Storage

- Leading to extremely high fees

- How to increase these resources?

Scaling solutions

- Option 1: increase gas limit (Lecture 8)
  - Equivalent to increase block size in Bitcoin
  - It takes longer to transmit the block → reduced security
  - Would raise hardware requirements for full nodes
- Option 2: sharding (Lecture 10)
  - Each node stores/executes part of the transactions
  - Requires L1 change
- Option3: sidechains (Lecture 12)
  - Moves both computation and storage off-chain
  - Data availability issue

Desired properties

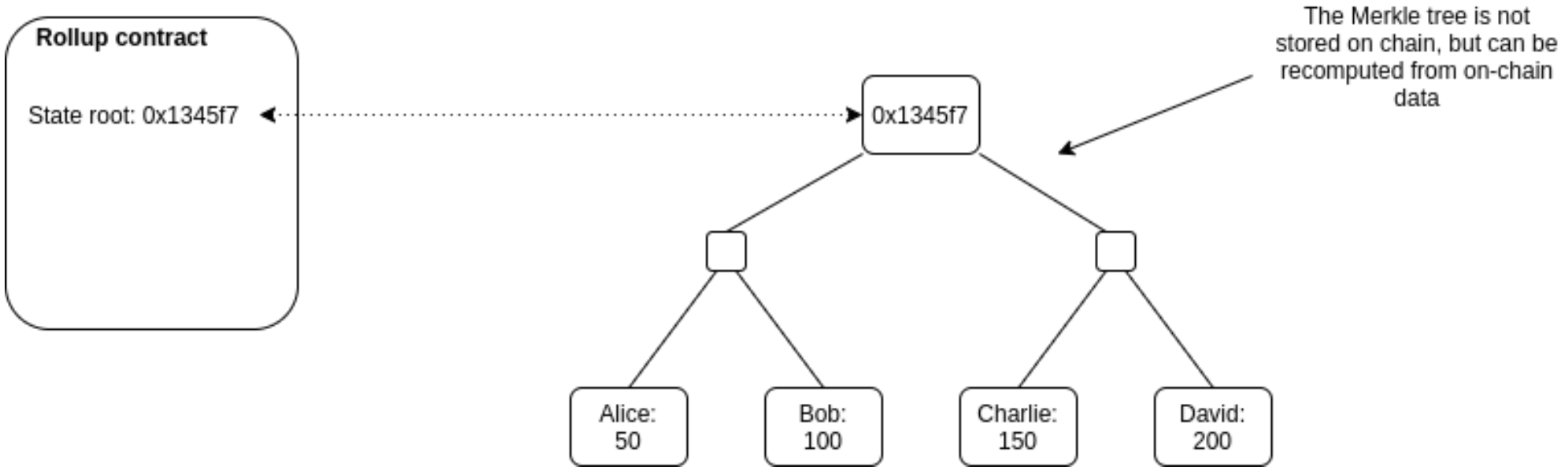
- L2 solution
  - Lightweight protocol via smart contracts
  - No change in L1

- Security and data availability
  - Reuse Ethereum security
  - Avoid introducing additional trust

- EVM compatibility
  - Easy to migrate a Dapp from L1 to L2
  - Reduced Tx fees and increased Tx throughput compared to L1

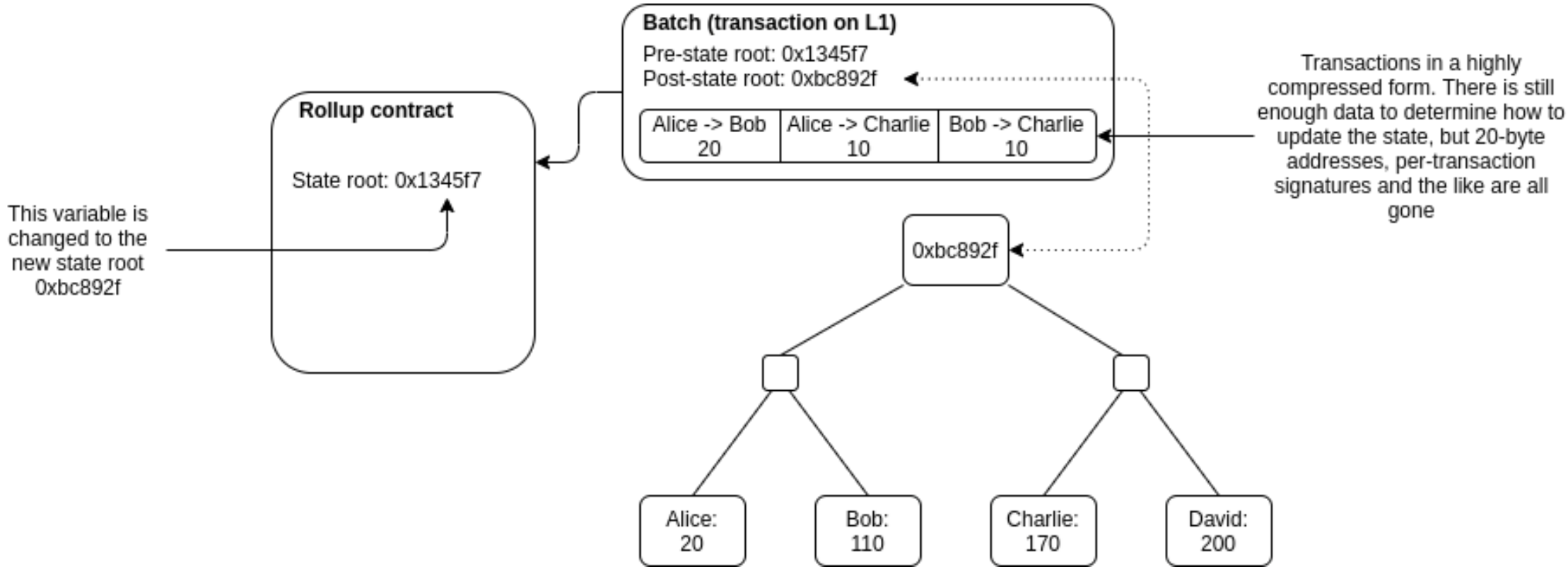
Rollups

- Execute transactions off-chain (outsource computation – just like sidechains)
- Report data on-chain in a compressed way (but enough to verify execution is correct) -- so no data availability problem



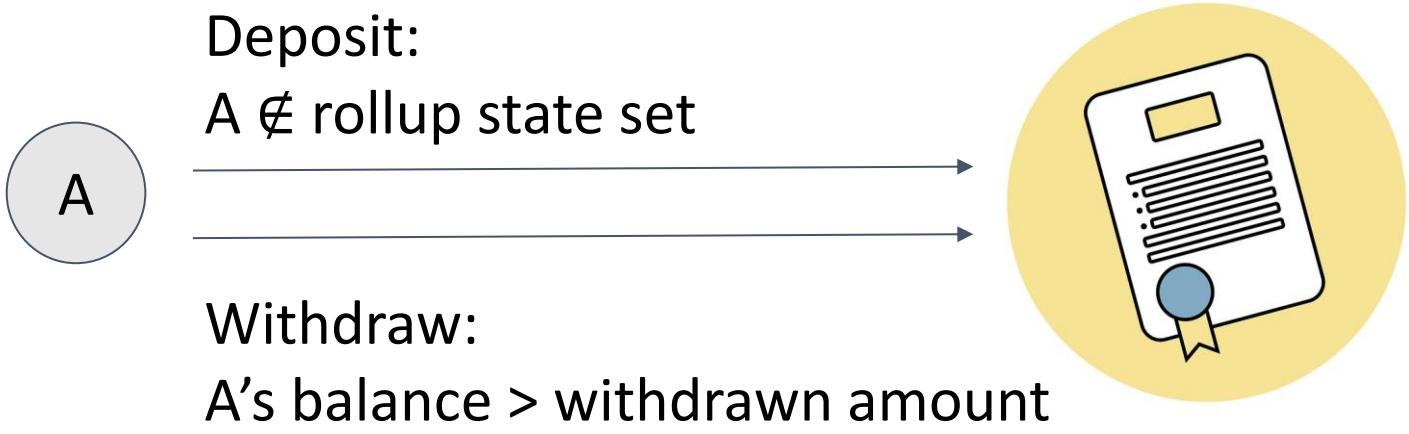
Rollups transfer

- Anyone can publish compressed data on-chain (called *batch)*
  - Batch contains: Pre-state, post-state, compressed data



Rollup state can be composed with L1 transactions

- Input/Output from the outside
- Deposit: transfer assets to rollup contract
- Withdraw: transfer assets to outside account



Simple

- Two key questions:
  - How to know that the post-state roots in the batches are correct?
  - Data on-chain is the key to the data availability issue. But how much?
- Optimistic Rollups vs. ZK Rollups

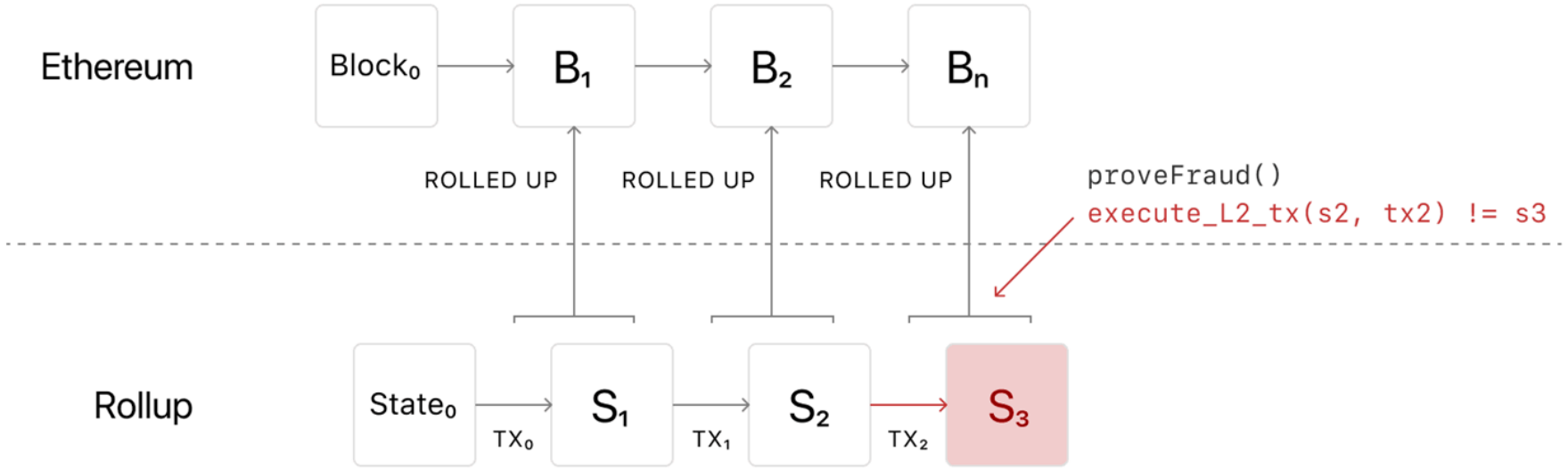
Two different approaches to Rollups

- Optimistic rollup
  - Go ahead and commit the transactions assuming honest assertion
  - Depend on feedback to detect and address fraud

- ZK rollup
  - Compress state with an upfront proof of correctness
  - Verification is easier than the compression stage

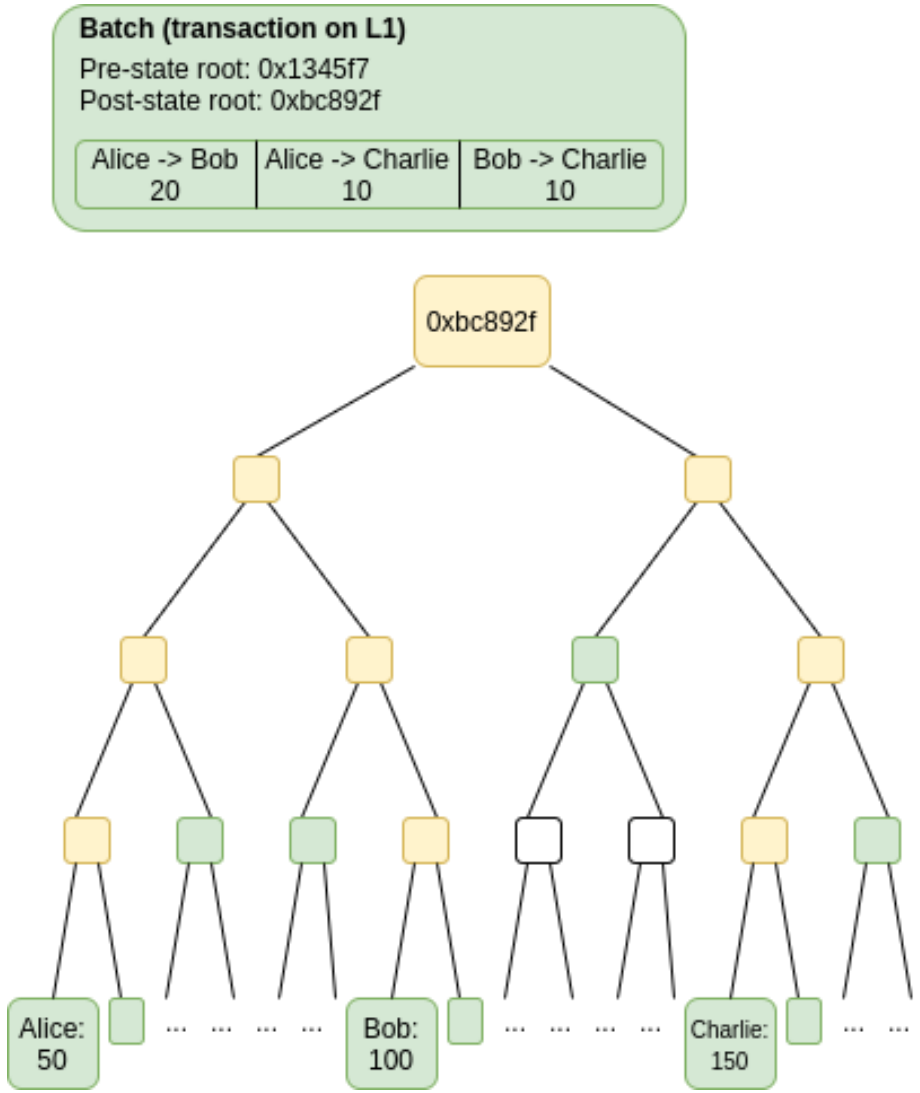
Optimistic Rollups involve fraud proofs

- Any L1 validator that sees an invalid state root can
  - Challenge it by posting the valid state root
  - Slash the L2 assertor that committed such fraud
  - Claim rewards
- L2 chain is rolled back after a fraud proof is finalized



Fraud proof generation and verification

- Fraud proof
  - The batch
  - Parts of the Merkle tree to prove specific accounts (the green nodes in the figure)
- The fraud proof is sufficient to execute the batch and compute the post-state root
- Computed post-state root ≠ the provided post-state root



Complexity of fraud proofs

- Verify the fraud proof using a verifier contract
  - Re-executing the disputed transactions on L1
  - This can incur huge gas costs (see example), an overhead worth avoiding if possible

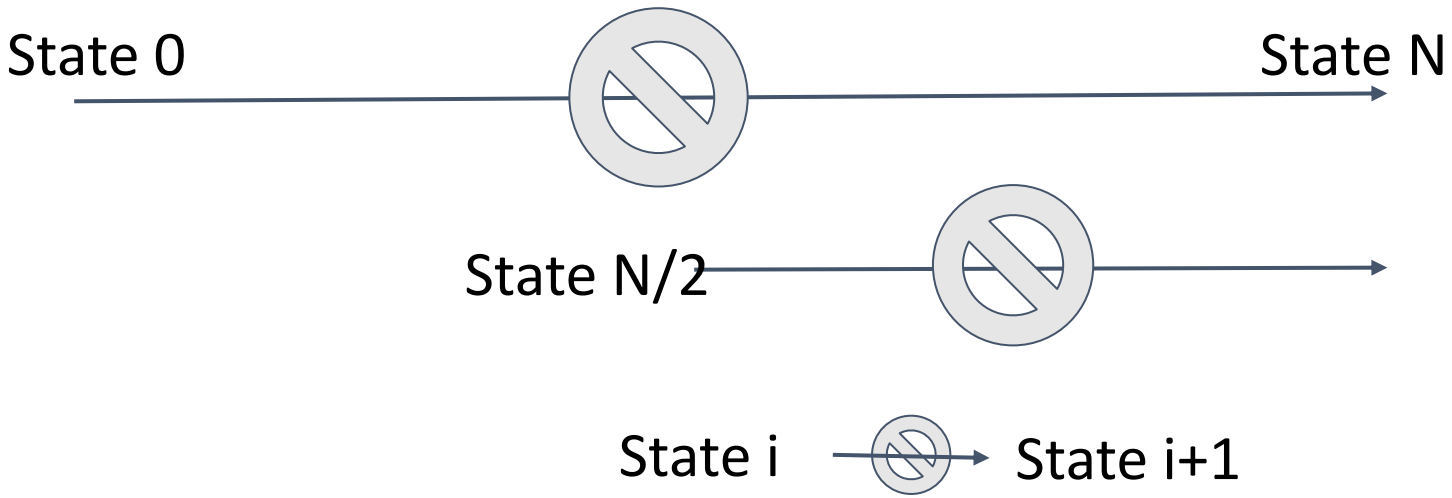
- Two different methods to avoid this overhead
  - Optimism and Arbitrum

Method 1: Publish a bit more data on-chain

- Publish state commitments for all individual transactions
- An instant, single-round fraud-proof
  - Step 1: Declare which state transition you're disputing
  - Step 2: Upload all the transaction pre-state
  - Step 3: Once all pre-state has been provided, run the transaction
  - Step 4: Provide the post-state
  - Step 5: Complete the state transition & finalize the fraud proof
- Optimism blockchain

Method 2: Interactive fraud proof

- No upfront change to on-chain data
- An **interactive**, multi-round fraud-proof
  - Binary search
  - A singular point of transaction disagreement (see figure)
- Similar idea to cross-shard transaction management in sharding
- Arbitrum



ARBITRUM

Addressing data availability

- All the compressed transaction data is on chain
  - Minimum amount of data to address data availability
- A few clever compression tricks
  - Use a fixed fee level in each batch
  - Replace the 20-byte address with an index
  - Use BLS aggregate signatures
  - Write transactions to Ethereum as *calldata*

Data compression of an individual transaction

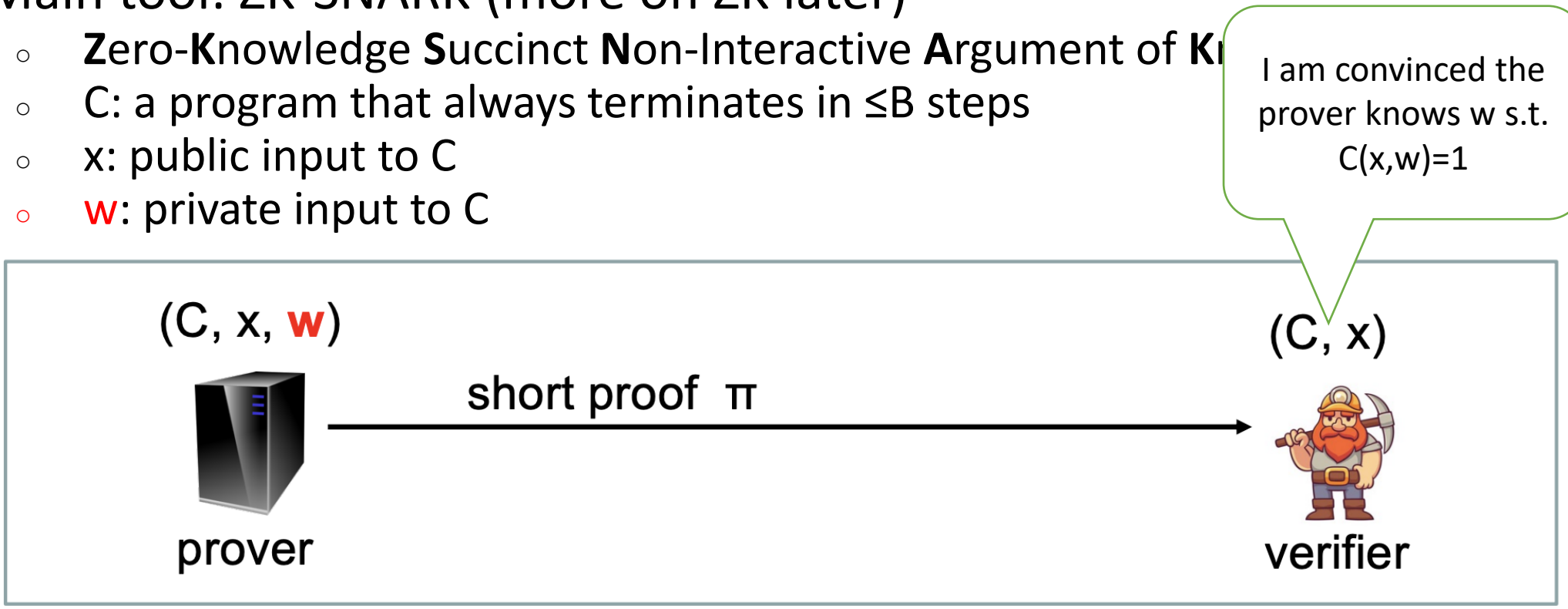
Parameter	Ethereum	Rollup
Nonce	~3	0
Gasprice	~8	0-0.5
Gas	3	0-0.5
To	21	4
Value	~9	~3
Signature	~68 (2 + 33 + 33)	~0.5
From	0 (recovered from sig)	4
Total	~112 bytes	~12 bytes

Optimistic Rollups: pros and cons

Pros	Cons
Offers massive improvements in scalability without sacrificing security	Security model relies on at least one honest node executing rollup transactions and submitting fraud proofs
Permissionless (anyone can force the chain to advance by executing transactions and posting assertions)	**Users must wait for the one-week challenge period to expire before withdrawing funds back to Ethereum**
Compatibility with EVM and Solidity allows developers to port Ethereum-native smart contracts to rollups	Rollups must post all transaction data on-chain, which can increase costs.

Method 2: ZK Rollups

- Main tool: ZK-SNARK (more on ZK later)
  - **Z**ero-**K**nowledge **S**uccinct **N**on-Interactive **A**rgument of **K**
  - C: a program that always terminates in ≤B steps
  - x: public input to C
  - w: private input to C



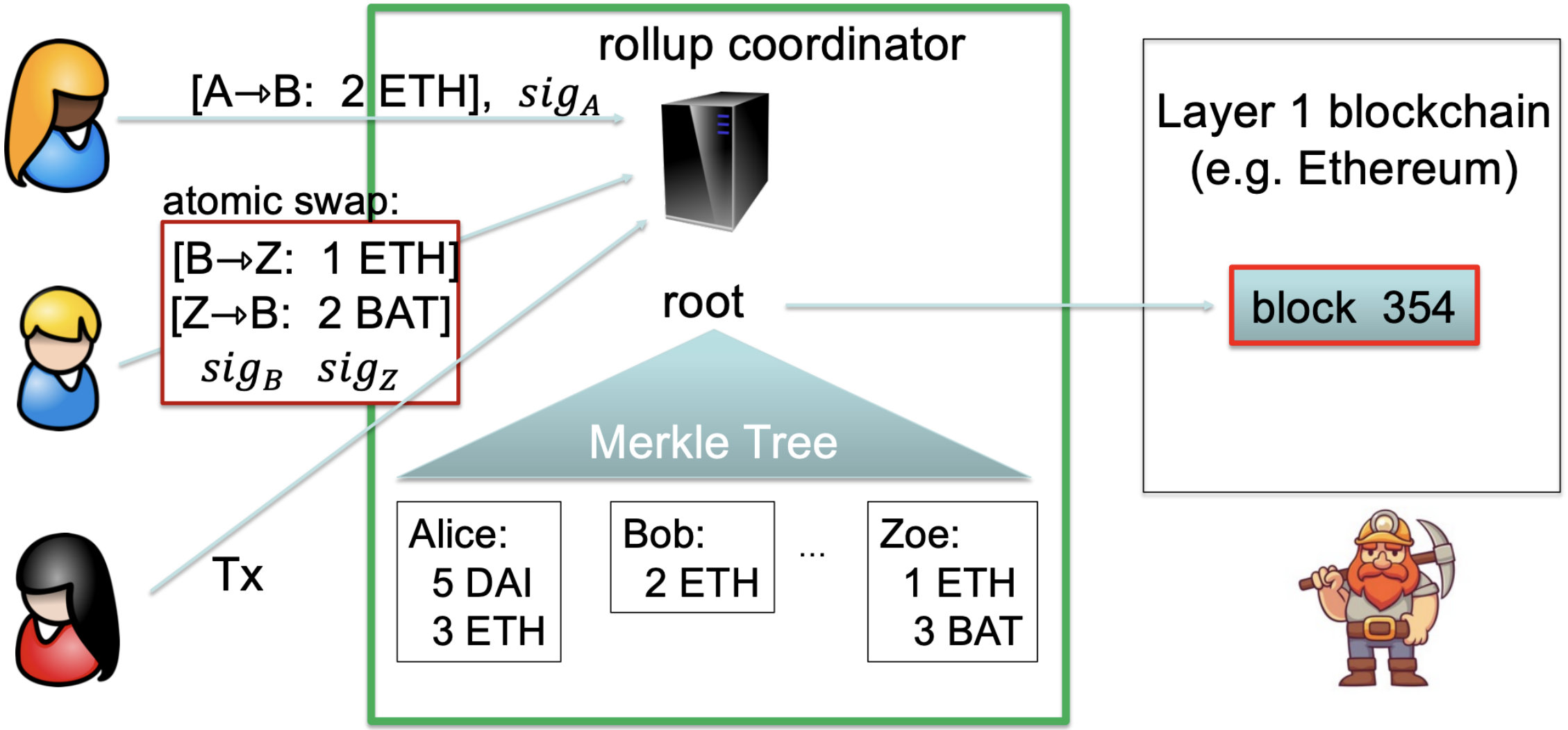
Key point: Verifier's run time is much less than running C

ZK Rollups use validity proof

- ZK-SNARK
  - C: the state transition program
  - x: pre-state, post-state
  - w: all transactions

- A ZK rollup coordinator generates a SNARK proof π that proves it knows the private transactions such that the post-state is correctly updated from the pre-state

Validity proof example

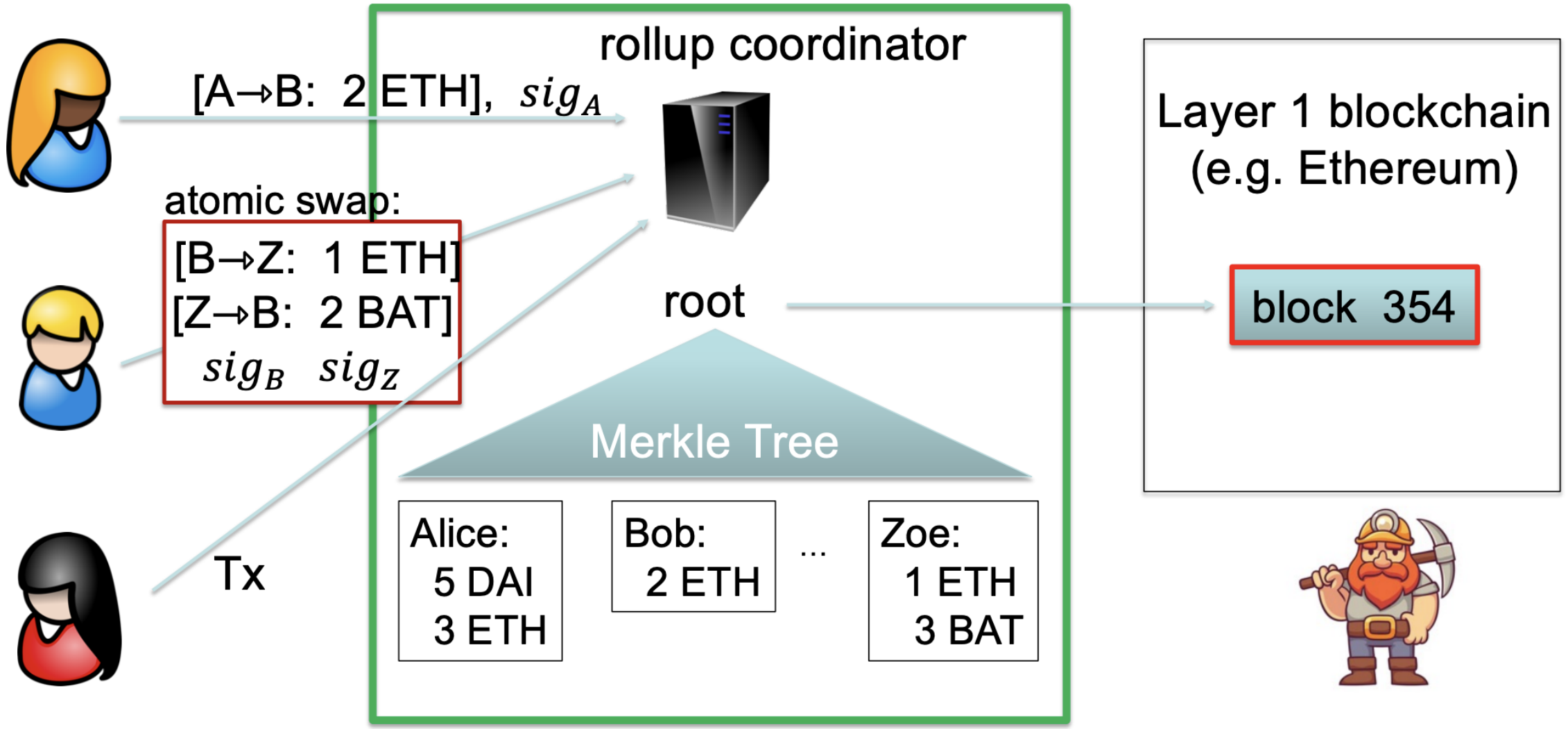


Validity proof example

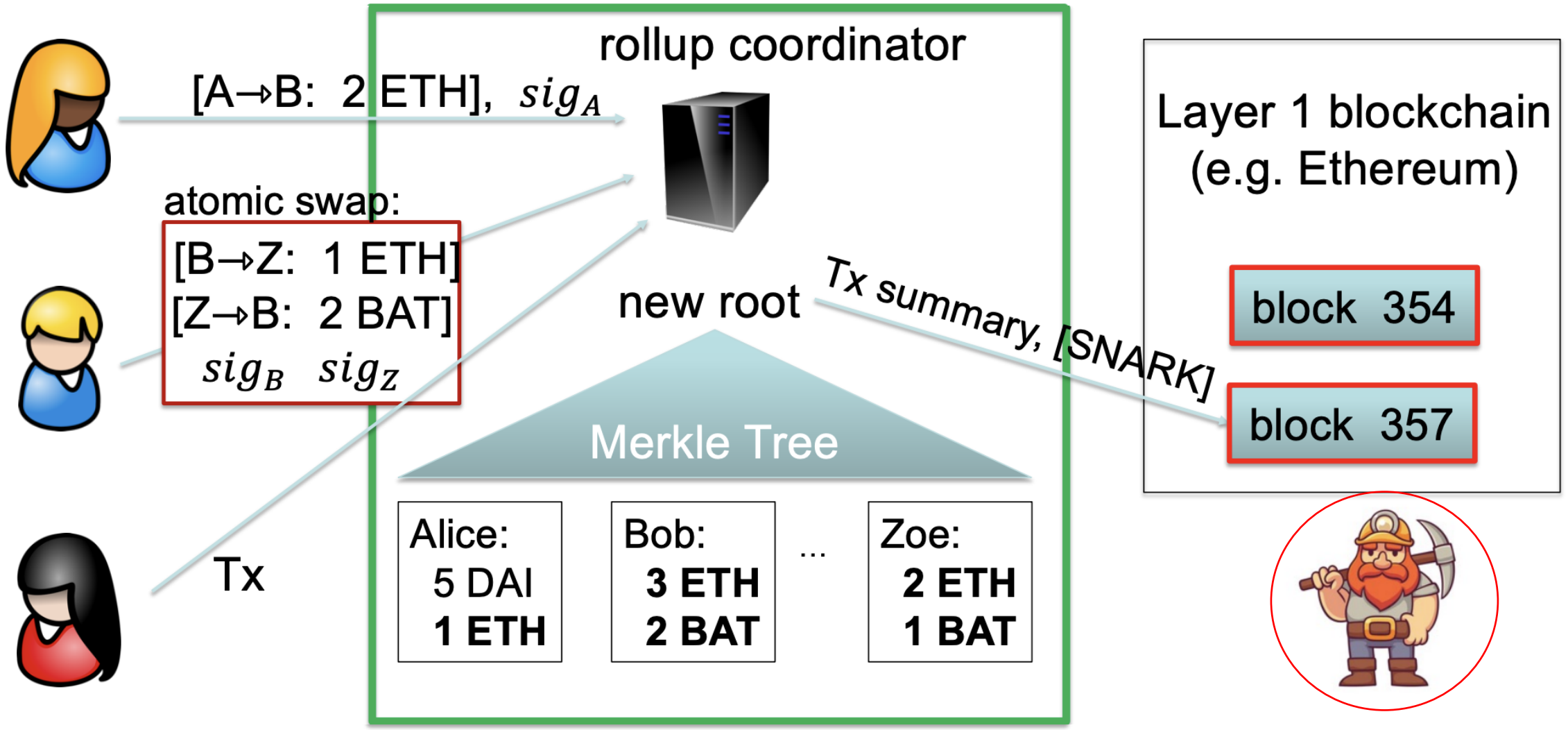
Take one TX **[A -> B: 2ETH]**, sig_A as an example, what is the validity proof?

- Public input x: pre-state root and post-state root (before and after executing TX)
- Private input w:
  - TX itself
  - A's and B's balances in the pre-state and the Merkel proofs
- The state transition program C does the followings:
  - Check TX is correctly signed by A
  - Check A's and B's accounts are part of the pre-state tree using the Merkel proofs
  - Check A's ETH balance >= 2
  - Reduce A's balance by 2 and increase B's balance by 2
  - Hash the updated account data and combines them with the Merkle proofs to generate a new Merkle root
  - Check the new Merkle root is the same as the post-state root
- The SNARK provides a validity proof π for C

Validity proof example



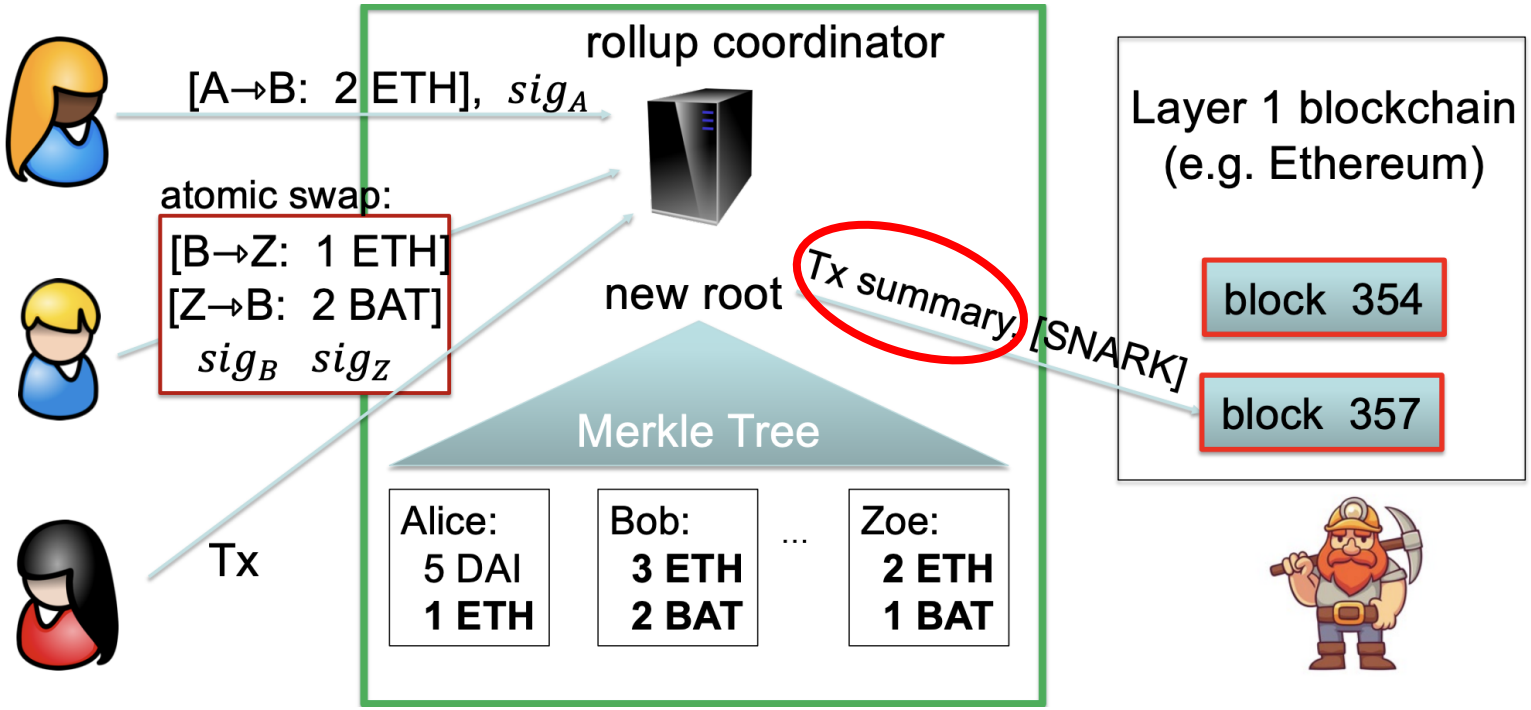
Validity proof example



The verifier is actually a smart contract.

Data availability

- Validity proofs already verify the knowledge of transactions and the authenticity of state transitions.
- Why do we need the **TX summary**?
- And what is it?



Why TX summary?

- Validity proofs already verify the knowledge of transactions and the authenticity of state transitions. Why do we need the **TX summary**?

  - It allows permissionless, independent verification of the L2 chain's state
  - Anyone can submit batches of transactions, preventing malicious operators from censoring or freezing the chain.
  - Without access to state data users cannot query their account balance or initiate transactions that rely on state information.

What is TX summary?

- The TX summary can be:
  - Either all transactions without signatures
    - As signatures have already verified by the validity proof
    - Even less data than Optimistic rollups
  - Or all the changes that should be made to the state

Two approaches: zkSync vs. zkPorter

- zkSync: store all Tx summaries on Ethereum
  - Ethereum accepts Tx batch only if it includes summary of all Tx
  - Other coordinators can reconstruct L2 state from L1 blockchain
  - Downside: higher Ethereum Tx fees. Good for high value assets

- zkPorter: store Tx data on a new blockchain
  - maintained by a set of staked coordinators
  - Cheap off-chain storage, but lower guarantee than zkSync
  - Customer can choose how coordinator will store its account

EVM compatibility

- General-purpose EVM computation
  - Harder to prove in circuits than simple computations (like the token transfer described previously)
  - Resource-intensive

- zkEVM: many ongoing projects
  - Polygon
  - Scroll
  - zkSync
  - StarkWare

zkEVM

- zkEVM
  - Recreates existing EVM opcodes for proving/verification in circuits, allowing to execute smart contracts.
  - Like the EVM, a zkEVM transition between states after computation is performed on some inputs.
  - The difference is that the zkEVM also creates SNARK proofs to verify the correctness of every step in the program’s execution.

ZK Rollups: pros and cons

Pros	Cons
Short withdrawal delays	Building EVM-compatible ZK-rollups is difficult due to complexity of ZK technology
Relies only on trustless cryptographic mechanisms for security	High cost on computing and verifying validity proofs
Only TX summaries on chain	Some proving systems (e.g., ZK-SNARK) require a trusted setup

Comparison

Property	**Optimistic rollups**	**ZK rollups**
Fixed gas cost per batch	**~40,000**	~500,000
Withdrawal period	~1 week	**Very fast**
Complexity of technology	**Low**	High
Generalizability	**Easier**	Harder
Per-transaction on-chain gas costs	Higher	**Lower**
Off-chain computation costs	**Lower**	Higher