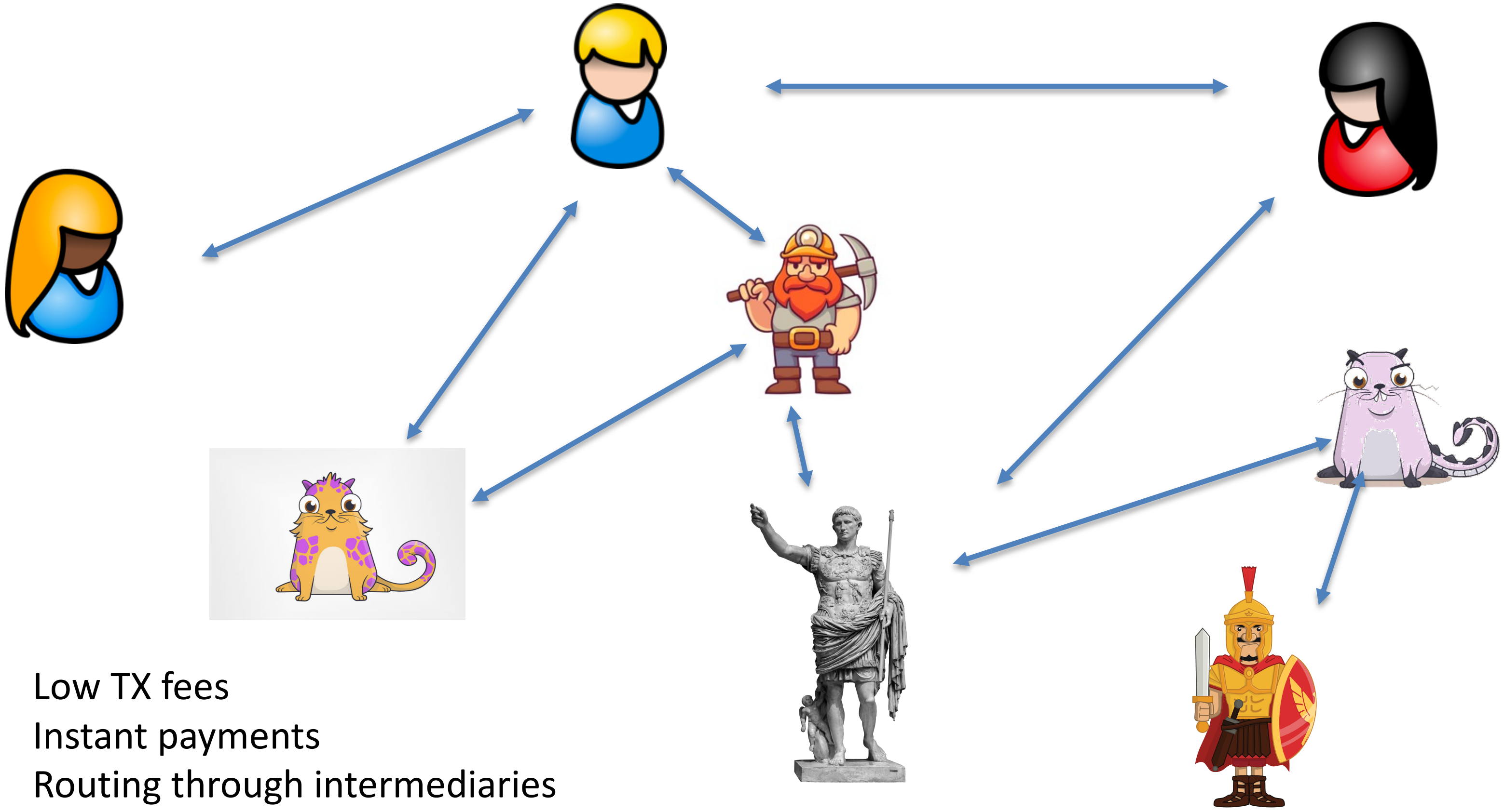
CS251 Fall 2021

(cs251.stanford.edu)

Scaling II: Rollup

Benedikt Bünz

Lightning network



- Low TX fees
- Instant payments
- Routing through intermediaries

Downsides of Payment/State Channels

- Everyone needs to be online
  - Mitigated by watchtowers
  - Hubs need to be online
- Capital is locked up
  - Funds in one channel can’t be used in different channel
  - If network is separated transactions are not possible
- Only Peer to Peer payments
  - No multi party contracts channels
- TX to fund/close

Blockchain Layers

Layer 3: **user facing tools** (cloud servers)

Layer 2: **applications** (DAPPs, smart contracts)

Layer 1.5: **compute layer** (blockchain computer)

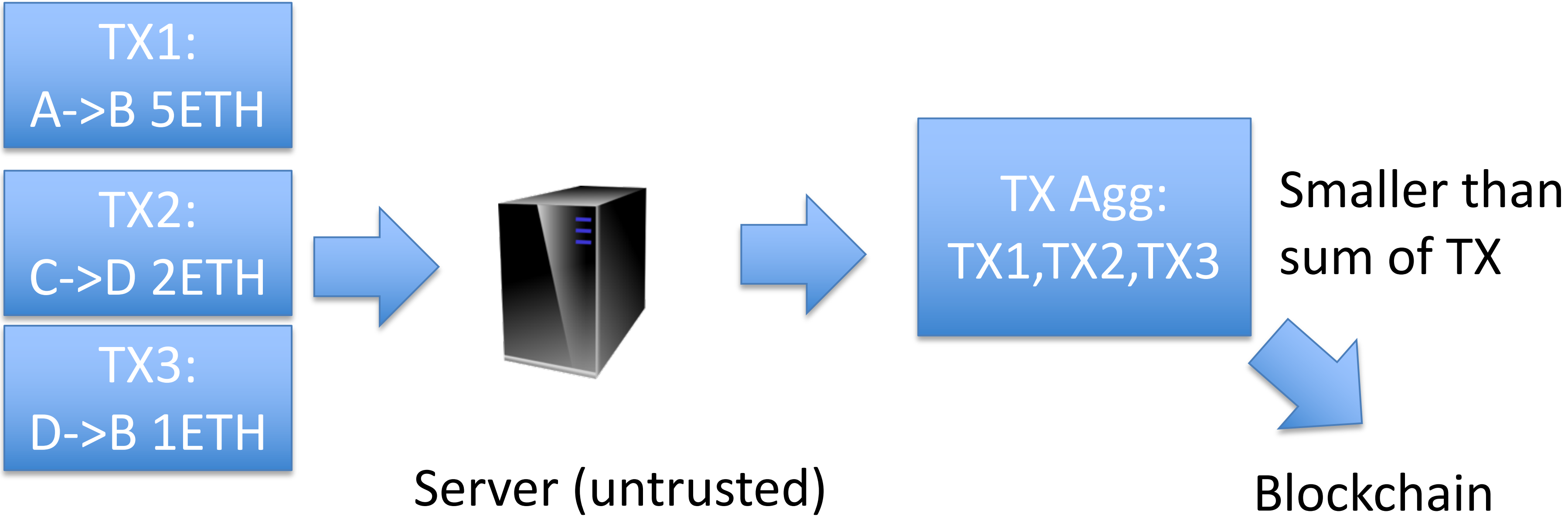
Layer 1: **consensus layer**

Bitcoin/Ethereum combine ordering (layer 1) and verification (1.5)

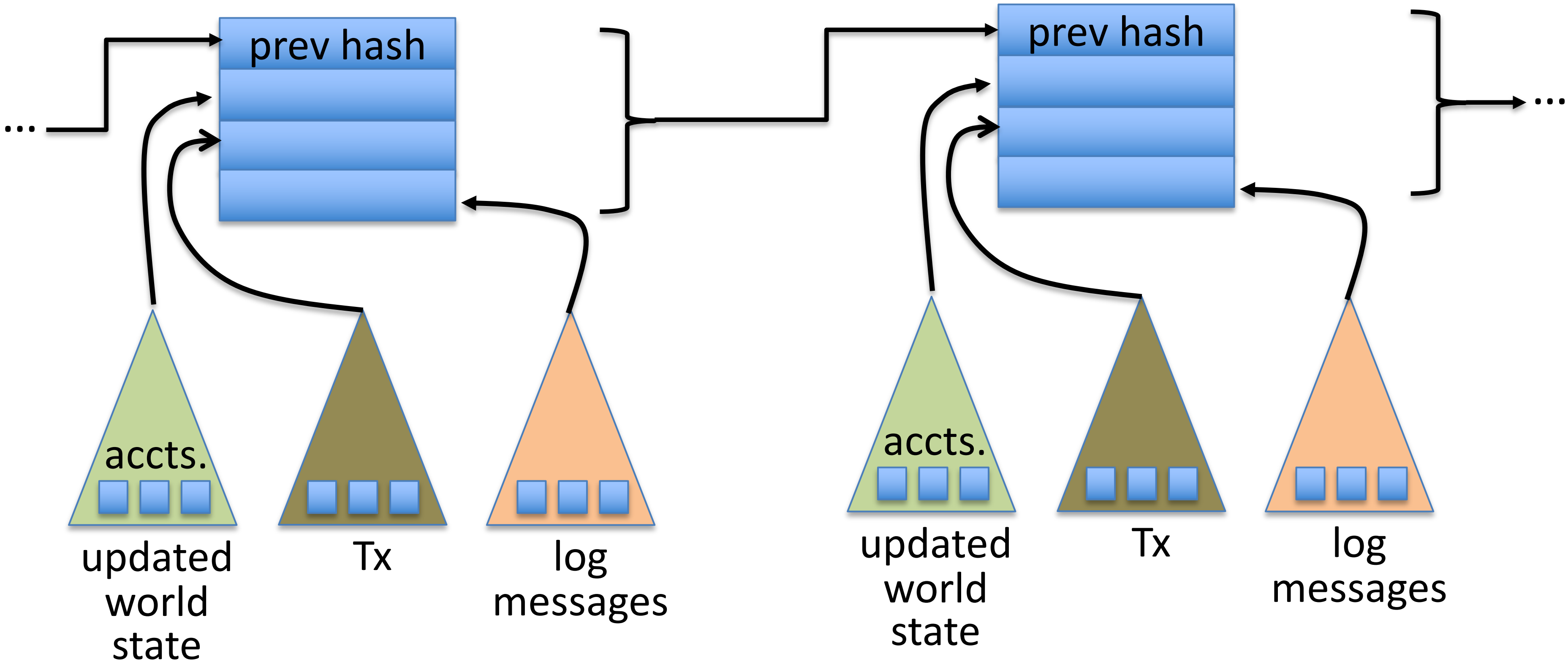
What if we can outsource verification? Makes consensus cheaper

Idea: Aggregate Transactions

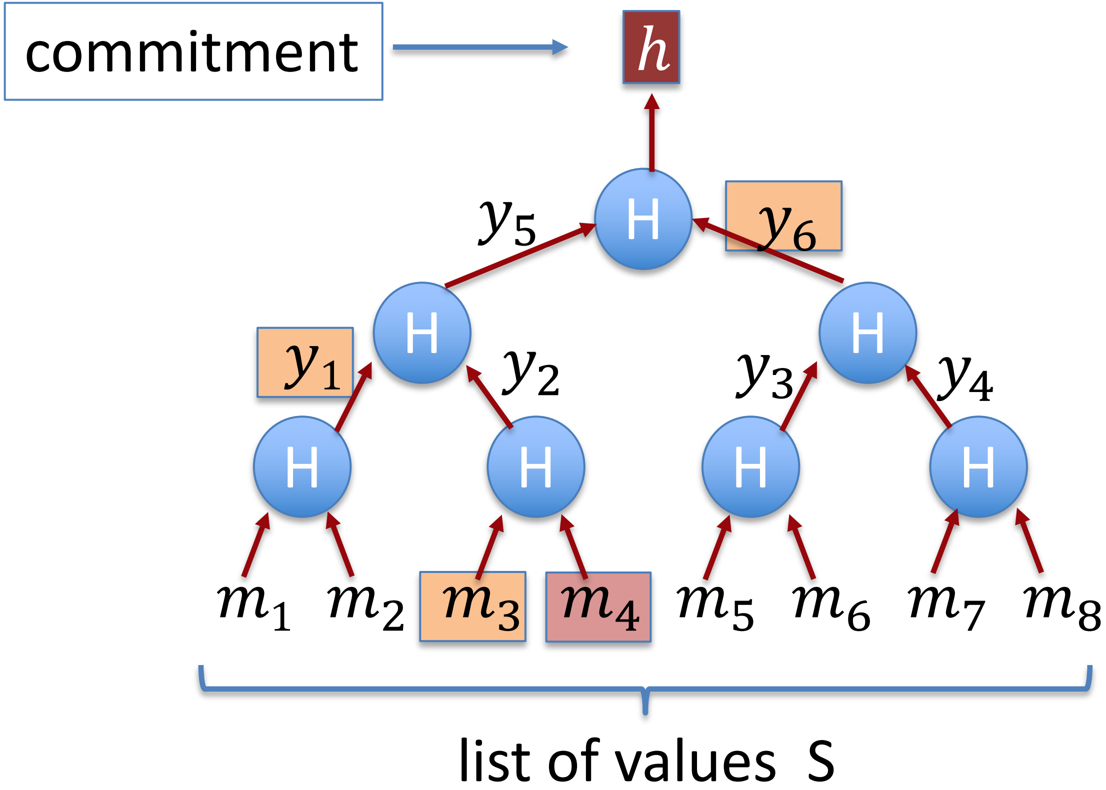
- Payment channels move more transactions offchain
- Idea: Combine Transaction, Rollup Server verifies



Recap: The Ethereum blockchain



Recap: Merkle tree (Merkle 1989)



Goal:

- commit to list S
- Later prove $S[i] = mi$

To prove $S[4] = m_4$,

proof $\pi = (m_3, y_1, y_6)$

length of π: $\log_2 |S|$

Recap State Commitment

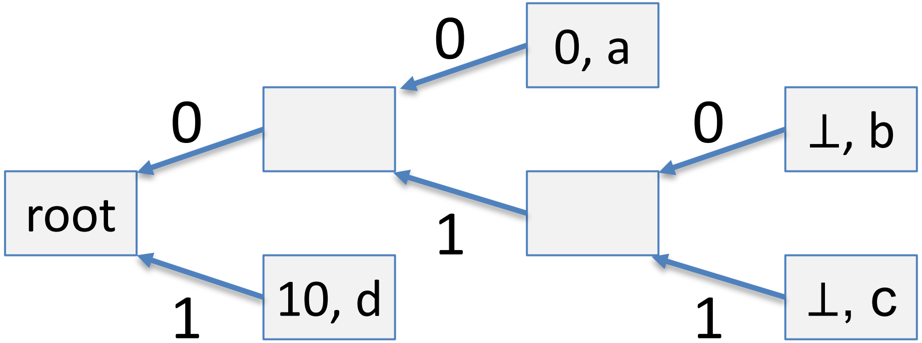
Every contract has an associated **storage array S**[]:

S[0], S[1], … , S[2^{256}-1]: each cell holds 32 bytes, init to 0.

Account storage root: **Merkle Patricia Tree hash** of S[]

- Cannot compute full Merkle tree hash: 2^{256} leaves

S[000] = a
S[010] = b
S[011] = c
S[110] = d



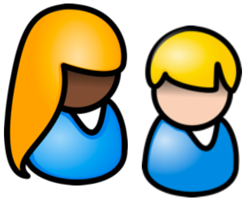
time to compute root hash: $\leq 2 \times |S|$

|S| = # non-zero cells

Merke (Patricia) Tree Proofs

- Logarithmic in tree height
- Given proof for i -> Possible to update S[i] and recompute root
- Given proof for i, proof for j and update of S[j] it's possible to update proof for S[i]
- Exclusion proofs possible in Patricia Trees

Rollup



Users

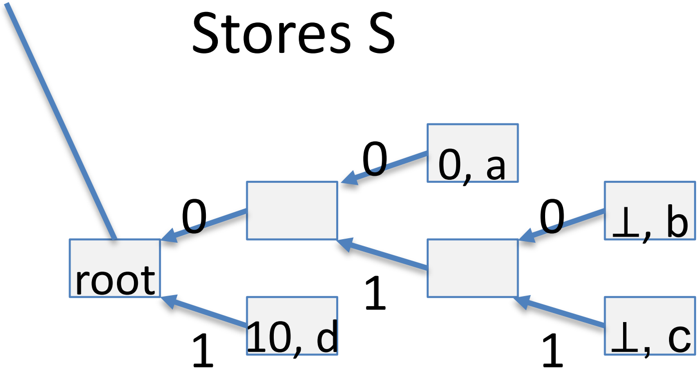
Server

Rollup Smart Contract

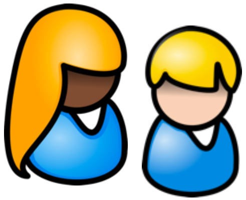
Rollup State S

S[A's PK] = {3 ETH, nonce}
S[B's PK] = {2 ETH, nonce}
S[C's PK] = {10 ETH, nonce}
S[D's PK] = {1 ETH, nonce}

Stores S



Rollup Deposit



TX Deposit

Users

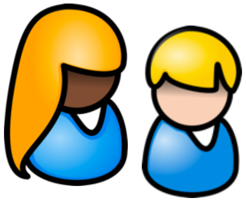
Rollup Smart Contract root

TX Deposit:

Proof that A's PK ∉ S given root
3 ETH transfer

1. Checks Proof
2. Updates root such that S[A's PK]={3 ETH, 0}

Rollup Withdraw



TX Withdraw

Users

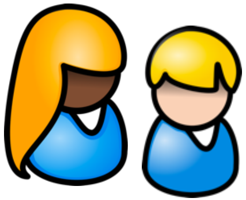
Rollup Smart Contract root

TX Withdraw:

Proof that S[A's PK]={3 ETH, nonce}
given root
Destination Address NewA
Signature by A

1. Checks Proof
2. Checks Signature
3. Sends 3 ETH to NewA

Rollup Transfer



TX Transfer

Space saved but no computation

Users

Rollup Smart Contract root

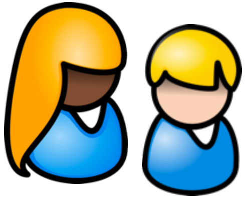
TX Transfer:

Proof that given root
S[A's PK]={3 ETH, 0}
S[B's PK]={2 ETH, 0}
Transfer amount 2 ETH
Signature by A

1. Checks Proofs
2. Checks Signature
3. Set
   1. S[A's PK]={1 ETH, 1}
   2. S[B's PK]={4 ETH, 1}

SNARK

```
function validatePatriciaProof(
    bytes32 rootHash,
    bytes memory key,
    bytes memory value,
    bytes[] memory path
) pure returns (bool accept) {
```



Provides Proof/SNARK that given given public inputs (rootHash, key, value) it knows private inputs (path) such that function outputs true

SNARK is short/easy to check

SNARK: a Succinct ARgument of Knowledge

A **succinct preprocessing argument system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v)

$|\pi|$ =500 bytes
time(V)=500k Gas

```
function validatePatriciaProof(
    bytes32 rootHash,
    bytes memory key,
    bytes memory value,
    bytes[] memory path
) pure returns (bool accept) {
```

roof π ; $|\pi| = O(\log(|C|), \lambda)$

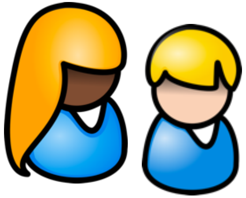
rify ; time(V) = $O(|x|, \log(|C|), \lambda)$

If (S, P, V) is **succinct** and **zero-knowledge** then we say that it is a **zk-SNARK**

ZKRollup

- Merkelize Transactions
- SNARK proves that given transactions I know signatures such that state transition S -> S’ valid
- Publish transaction diff on chain.
- No signatures per transaction.
- 500k gas + data cost for on chain diff

ZKRollup (Validity Rollup)



Users

Transactions

Server
Stores S

root’

π

Rollup Smart Contract

root

Commitment to S

Coordinator does:

1. Applies TXs to S resulting in S’
2. Produces root’= Commit(S’)
3. Produces SNARK π that ∃txs such that root’ is correct update to state S commited in root

ZKRollup

Users

Transactions

Server
Stores S’

root’

π

Rollup Smart Contract

root’

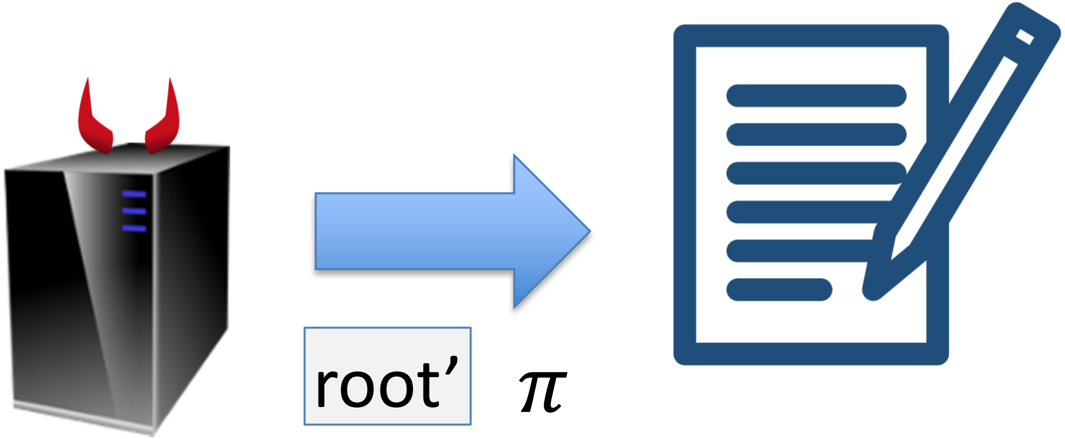
Commitment to S

Smart Contract does:

1. Verify π given root and root’
2. If accept then set root <- root’

Smart contract still allows “manual” withdrawals

Data Availability Problem

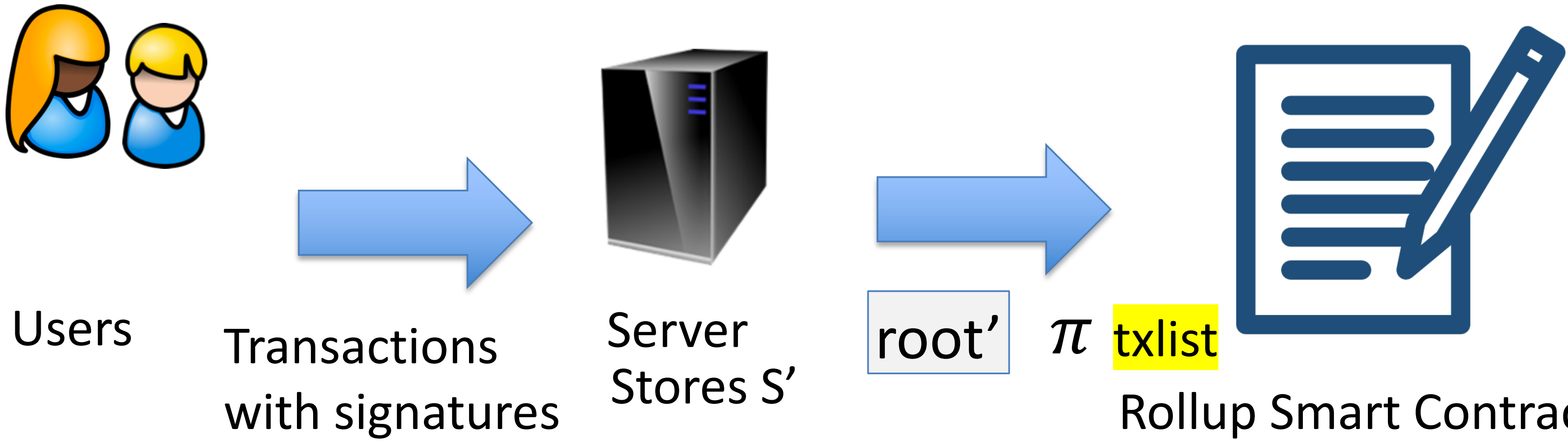


Update must be valid!

What if Server does not reveal data?

Can't update Merkle proofs
Can't withdraw!

Publish diff on chain



Txlist= [{A-> B 3}, {C-> D 2}, {D-> B 1}]

No signatures, Sender, Receiver, Amount only in Calldata (not stored)

<100bytes per tx ~400 gas/tx, SNARK verification ~1500 gas/tx (if full)

Full Block 3600 rollup tx vs 570 normal tx (6x speedup)

zkRollup stats

- ZKRollup is cheaper than onchain tx
- Can scale to max ~300tx/s now, 1000tx/s soon
- Vs. max 40-50tx/s on mainnet
- Cost dominated by SNARK verification
  - Will get cheaper precompiles
- Finality ~ Blockchain finality (no instant transfer)

Multiple Assets

Very easy to support many assets
Simply add asset field to TX
Hardly increases SNARK complexity

Txlist= [{A-> B 3 ETH}, {C-> D 2 DAI}, {D-> B 1 BAT}]

1 byte → 256 assets
2 bytes→ 65k assets

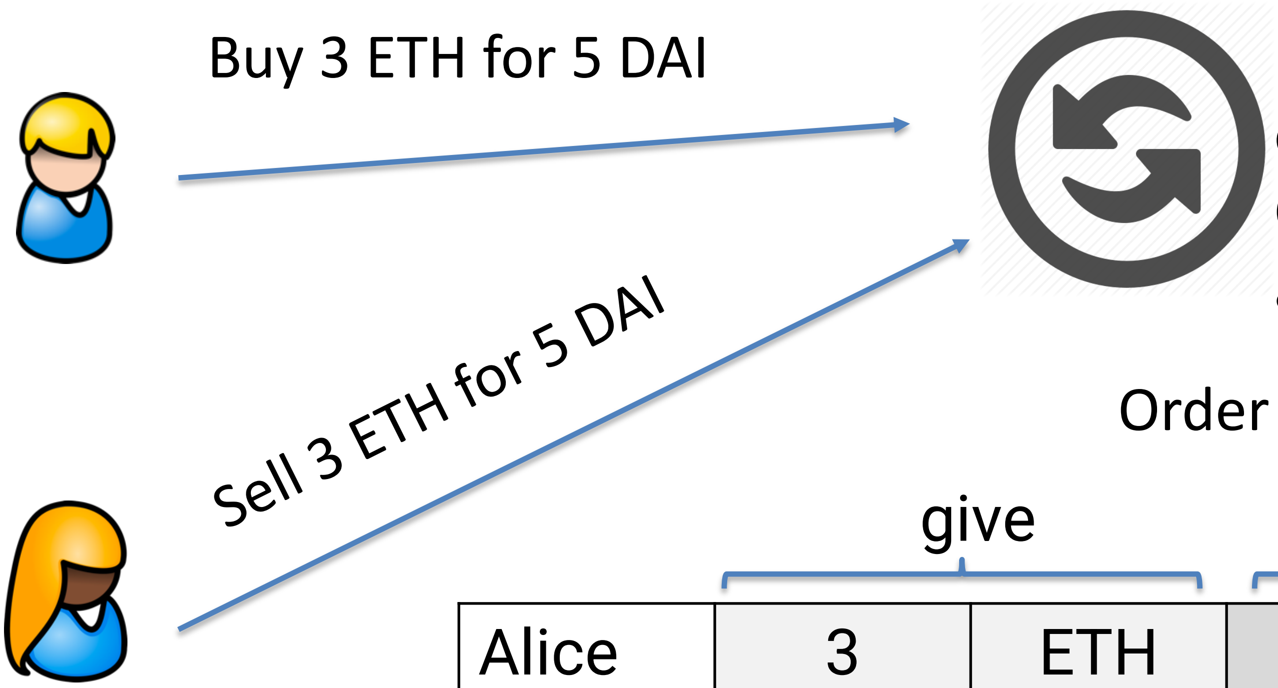
Transaction List/Atomic Swaps

Support transaction list that are executed together
Transactions need to be signed by all senders
Can’t execute part of transaction only all together!

Enables atomic swaps: Alice swaps with Bob 3 ETH for 2 DAI

Txlist= [{A-> B 3 ETH and B-> A 2 DAI}, {D-> B 1 BAT}]

Exchanges



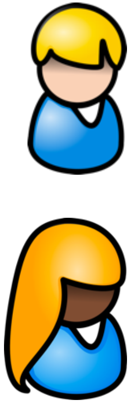
Exchanges match orders

Classical exchanges also store funds

Order book

	give		get	
Alice	3	ETH	5	DAI
Bob	5	DAI	3	ETH
Carol	4	BAT	10	DAI

Rolled up Exchange



Submit orders

Has orderbook
Matches orders
Rolls up transactions as atomic swaps

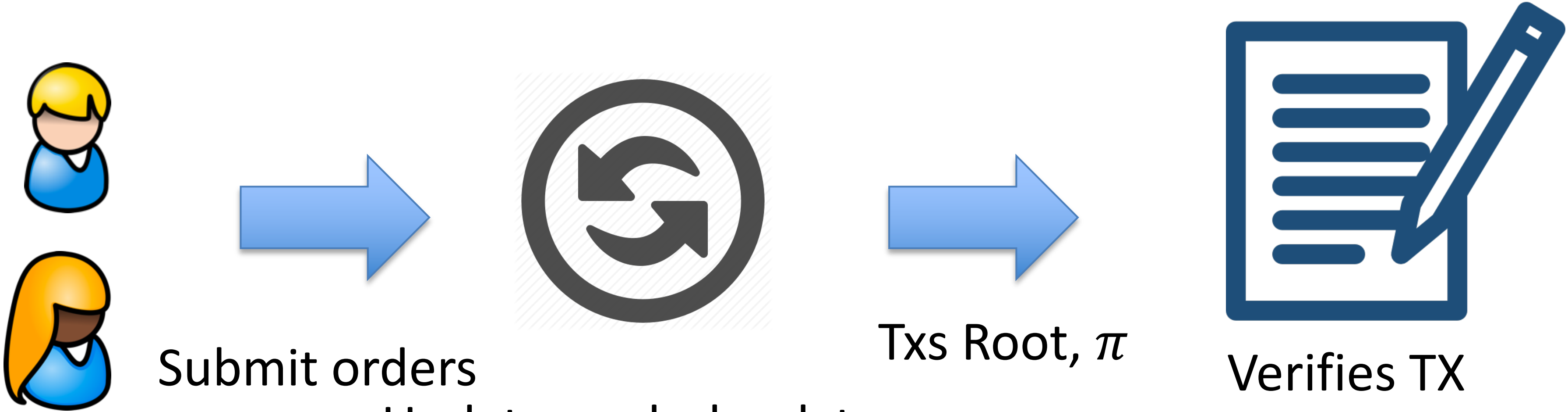
Txs Root, π

Verifies TX

Root of balance tree

Exchange trusted for honest matching

Rolled up Exchange v2



Updates orderbook tree on chain and proves correct matching

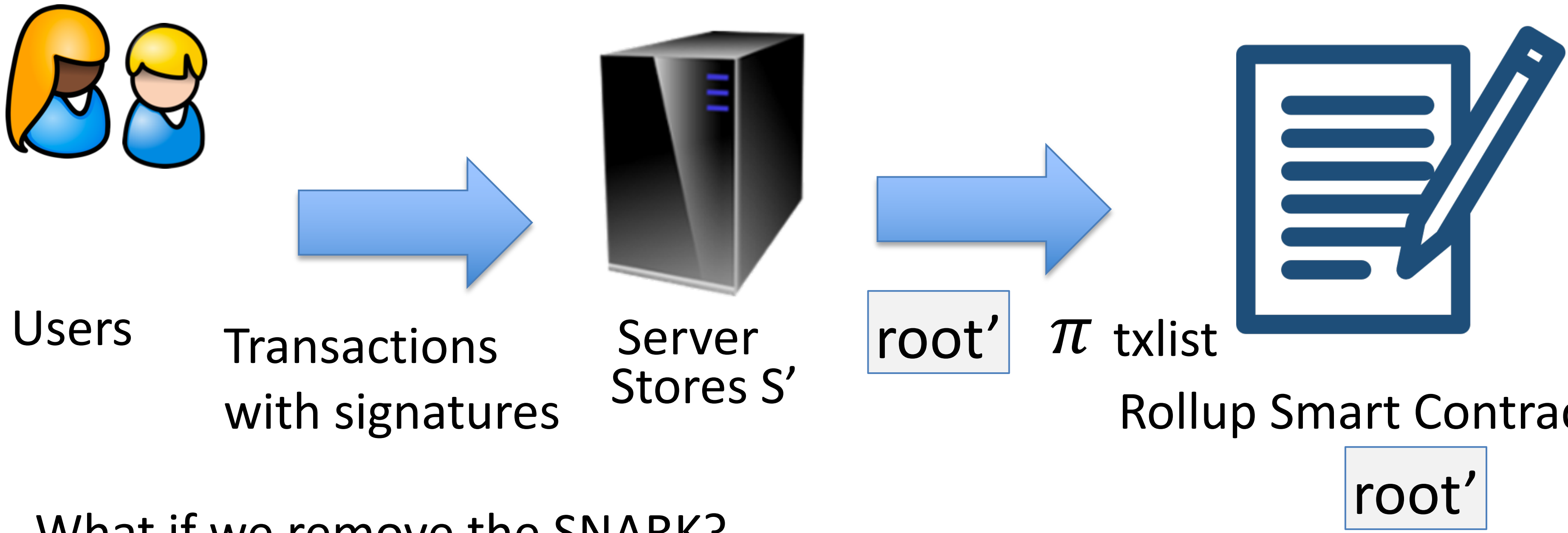
Root of balance tree
Root of orderbook tree

Benefit: No trust
Downside: Every order creates rollup TX, No instant matching

Rolling up Smart Contracts

- zkRollup works best for simple transfers
- zkRollup for the EVM?
  - Roll up generic smart contract transactions?
  - Create SNARK where the circuit implements the EVM
  - More expensive on the server
  - Soon to be a reality for a subset of the EVM (zkEVM)
- Can we support smart contract rollups today?

Optimistic Rollup



What if we remove the SNARK?

Idea: Instead of proving correctness, prove fraud!

New Role: Validator checks correctness, provides fraud proofs

Optimistic Rollup

- Server updates transaction root
- Server puts a large bond into escrow
- If transaction update is invalid users/validators provide *fraud proof*
- Successful fraud proof means bond gets *slashed*
  - Part to validator providing proof part gets burned
- Unsuccessful fraud proof costs validator money
- How to proof fraud?

Fraud Proofs

Server

root

Commits to state S

txlist

root’

Validator

1. Stores S agrees on root
2. Applies txlist to S to compute S’
3. Computes root’’ from S’
4. If root’≠root’’ call “Fraud”

Problem: Validator doesn’t know what’s in root’

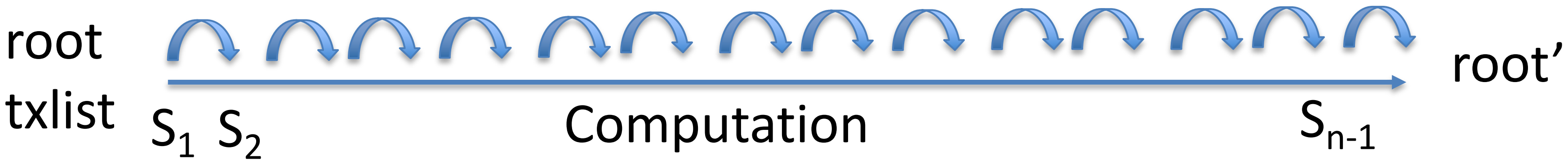
Referee Delegation

Idea: Server and Validator find first point of disagreement

Break down computation of S' into small steps, e.g. cycles on a VM

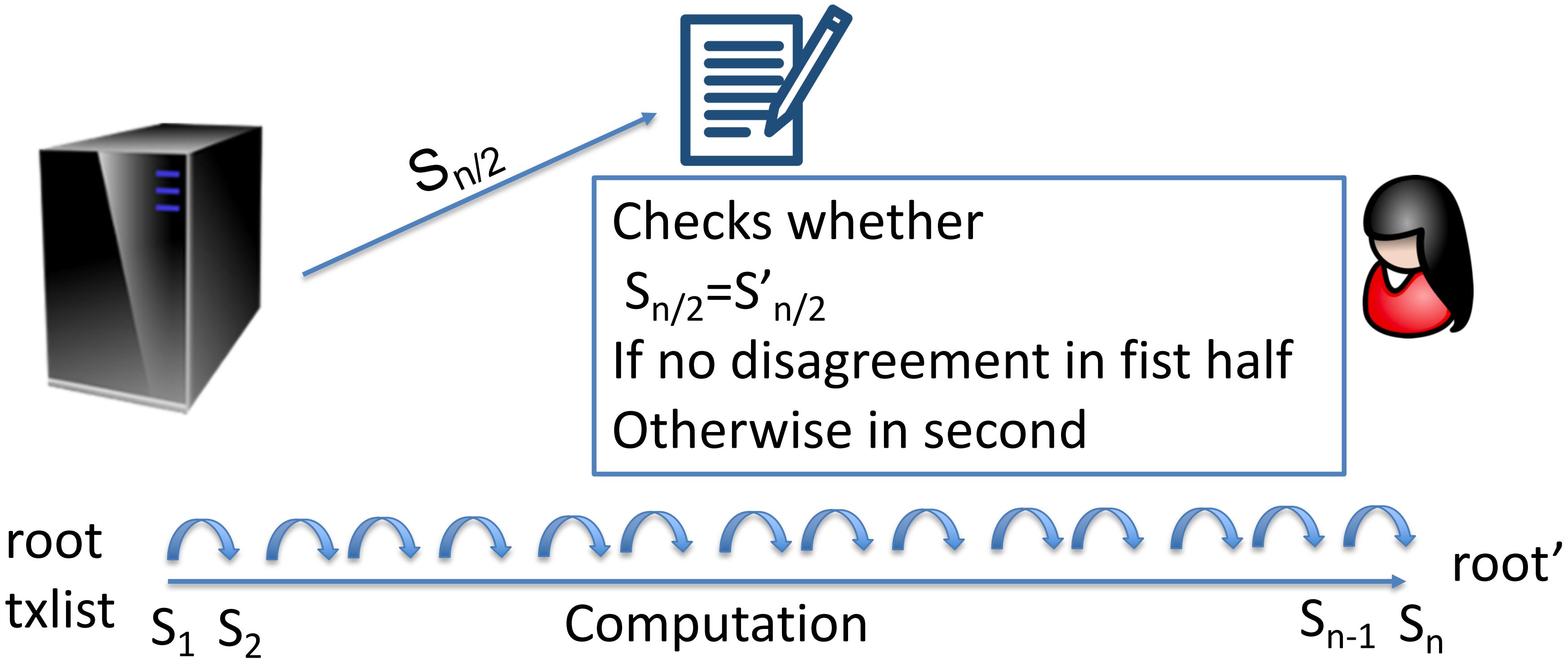
Validator does the same

Let S_i be Server's intermediate states and S'_i the validator's



Referee Delegation

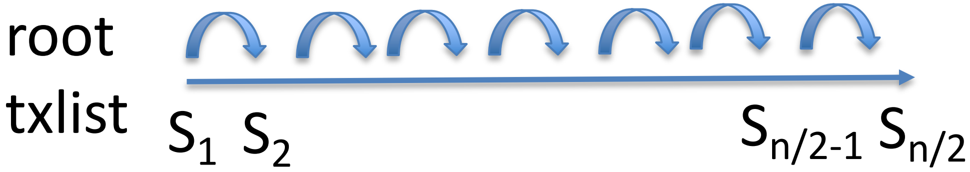
Server and Validator run interactive binary search



Referee Delegation

Repeat protocol for $\log_2(n)$ steps
End with agreement on S_i and disagreement on S_{i+1} and S'_{i+1}

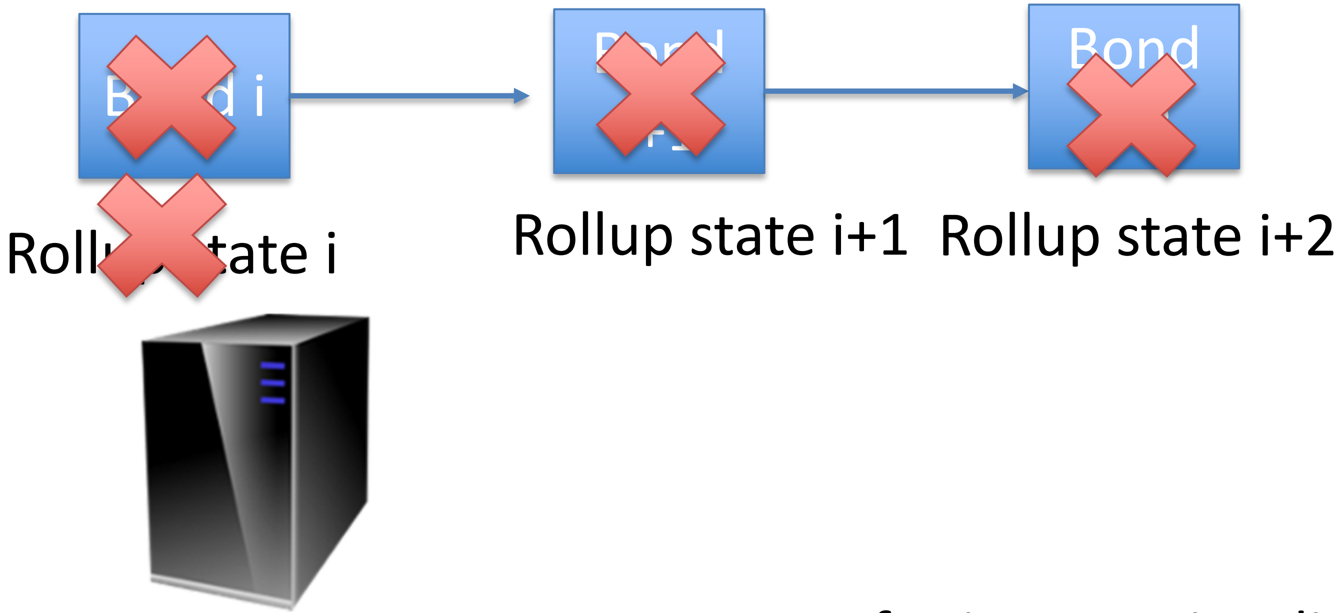
Smart Contract checks transition between S_i and S_{i+1} and declares winner



Problem: Checks take a long time

- $\log_2(n)$ messages (1 hash per message)
- 1 Verification step on smart contract
- If either party timeouts declares winner
- Looser gets *slashed,* Winner rewarded
- Problem: $\log_2(n)$*timeout
- No incentive to cheat
- But: Long wait till finalization! (7 days)

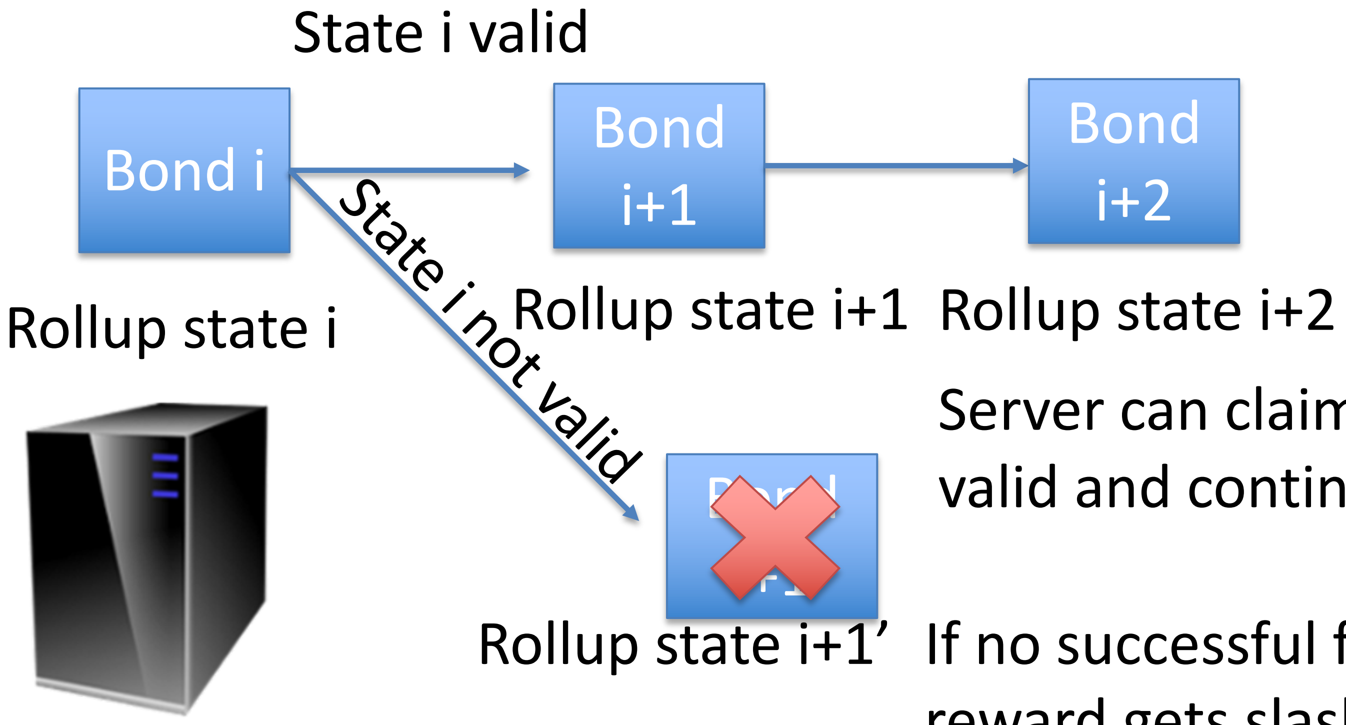
Pipelined Assertions



Server can build on states before timeouts

If prior state invalid, all subsequent bonds are slashed

Pipelined Assertions



Server can claim prior state not valid and continue given this.

If no successful fraud proof then reward gets slashed

Insurance of Rollup -> Instant Finality

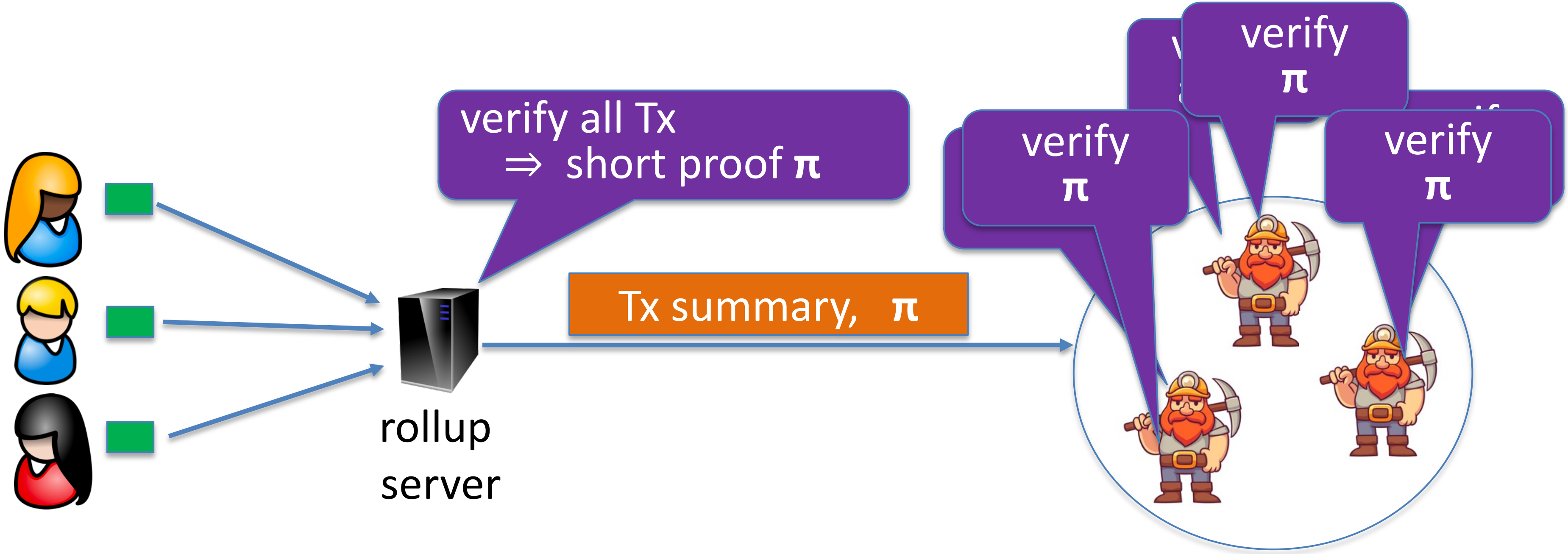
- Rollup is not instant (unlike lightning)
- But if server is trusted then giving them transaction -> finality
- Idea: Use insurance to achieve finality
- Server signs insurance
- If transaction not included in next (few) blocks insurance can be used to get insurance premium
- Works for zk and optimistic rollup
- Does not work for NFTs (directly)

Optimistic Rollup

- Live and implemented (Optimism and Arbitrum)
- You can port arbitrary smart contracts (OVM)
- Works well if honest rollup server
  - Fraud proofs protection if malicious server
- Up to ~4000 tx/s on ETH 1.0
- Important that one independent validator exists
- 7 day finality wait

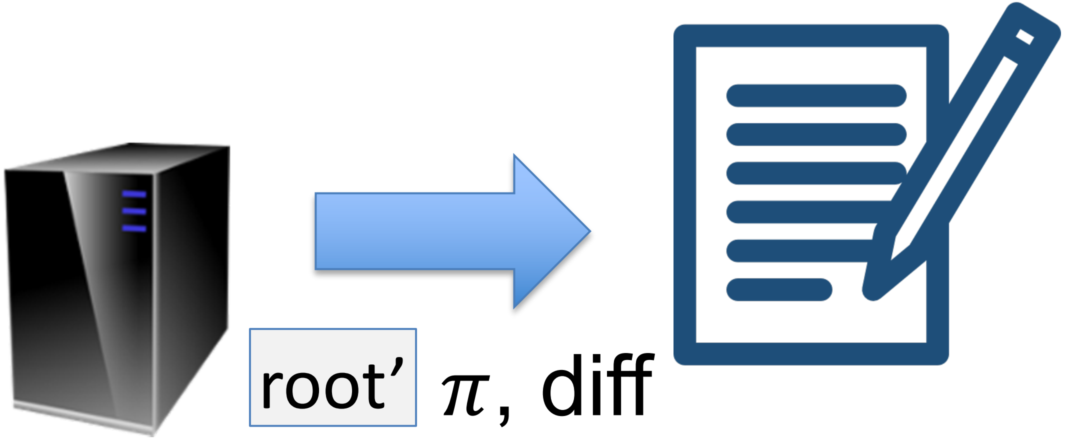
A combined view of rollups

Standard L1 chains: every miner must verify every posted Tx



Rollup server: compresses a thousand Tx into one on-chain proof (SNARK or Fraud)

Data Availability



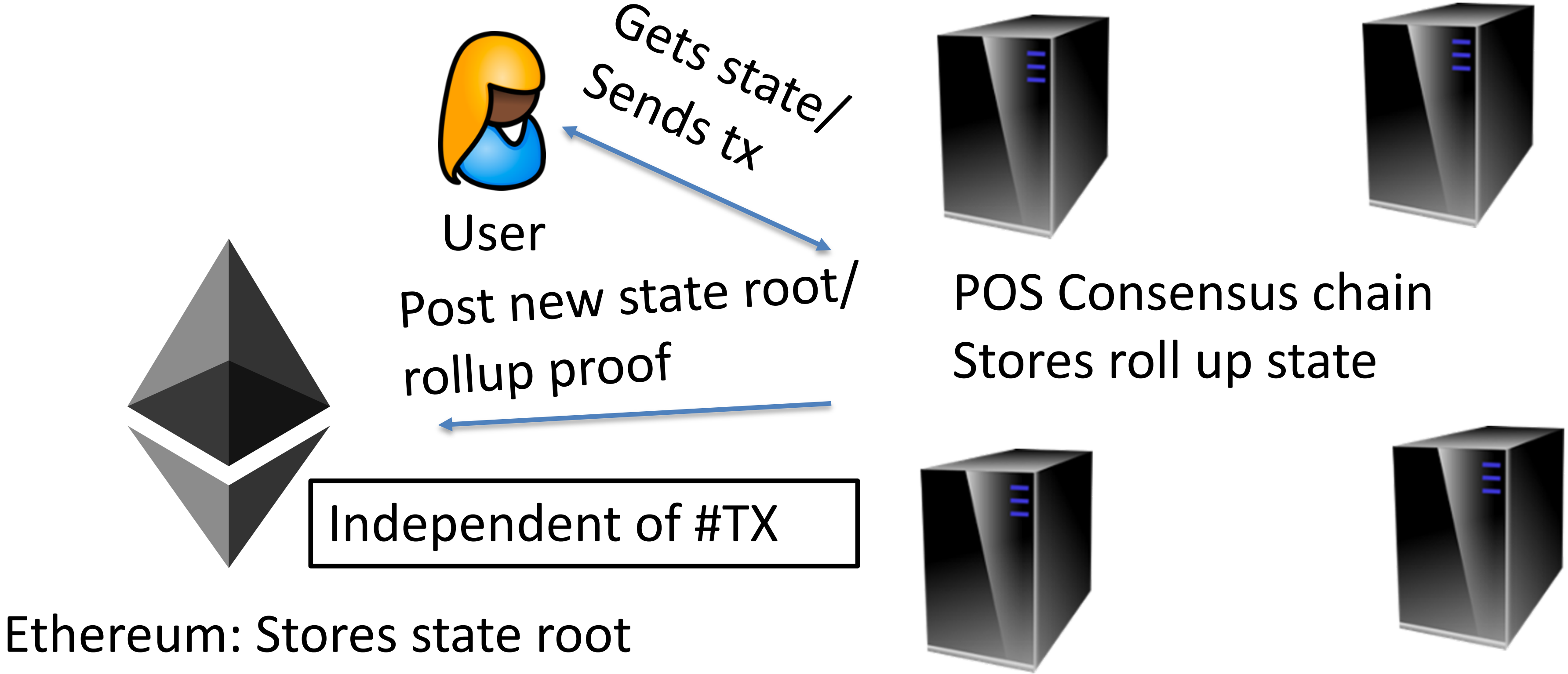
Update must be valid!

User must know state transition
Posting state diff on chain limits rollup benefits

Can we keep the data off chain?

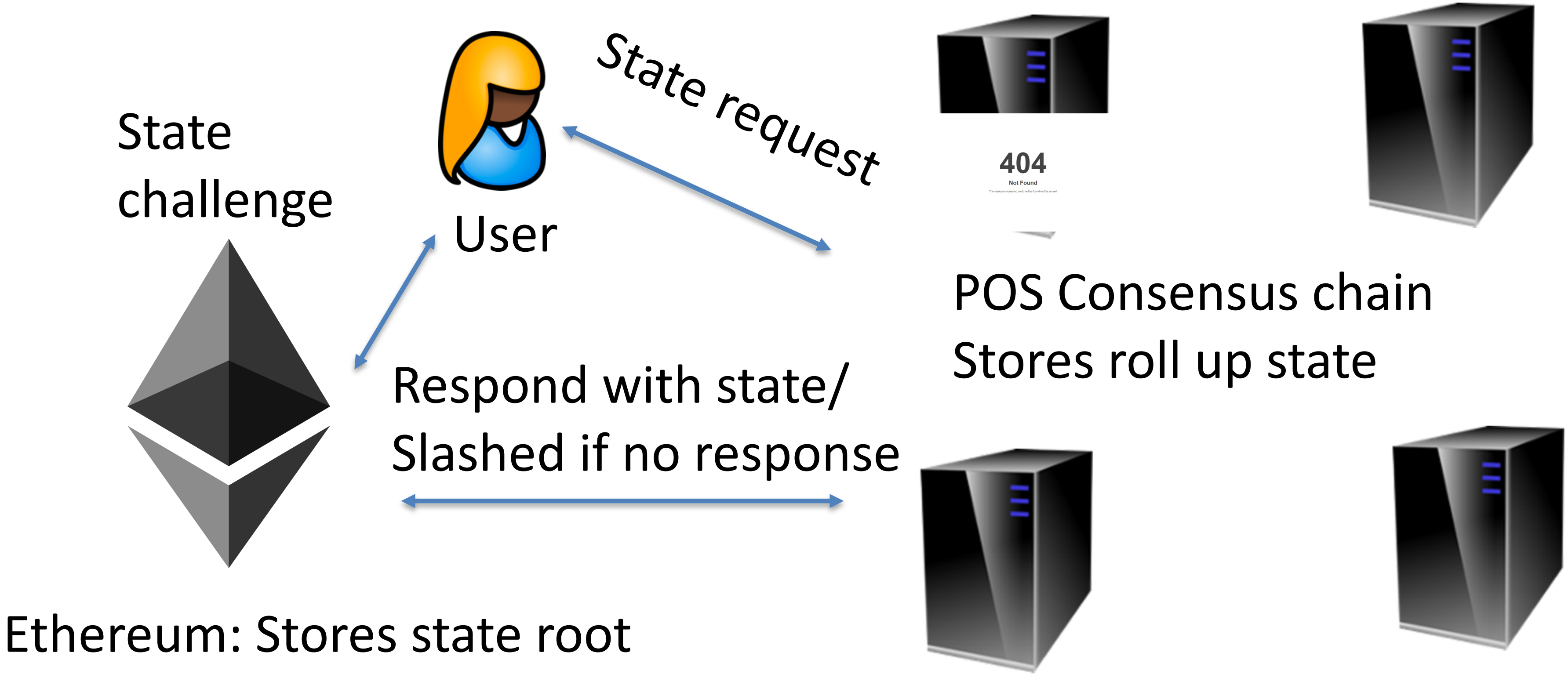
Off Chain Rollups (Validum and Plasma)

- Idea: Create a separate “cheap” chain for the data



Conflict Resolution: On Chain requests

- Idea: Create a separate "cheap" chain for the data



On Chain vs Off Chain Data availability

- Off Chain is much cheaper and independent of #tx
  - Only limitation is data consensus
- Data consensus not trusted for security
  - Can’t steal your money
- Data consensus is trusted for availability
  - Can lock up your money (bribery attack)
  - Can increase fees
- Economic incentives can mitigate issues
- For high value transfer use on chain rollup for low value use off chain rollup

2 by 2 rollup

Scaling the blockchain: Payment channels and Rollups (L2 scaling)

security →

availability {

	SNARK validity proofs	**Fraud proofs**
Tx summary on L1 chain	**zkRollup** blockchain finality, only simple transfers (now)	optimistic Rollup 7 day finality Instant transfers
Tx summary off chain	Vallidium large #tx but vulnerable to lock up attacks	”Plasma” Largest #tx but lock up attacks and long finality

END OF LECTURE

Next lecture: Recursive SNARKs