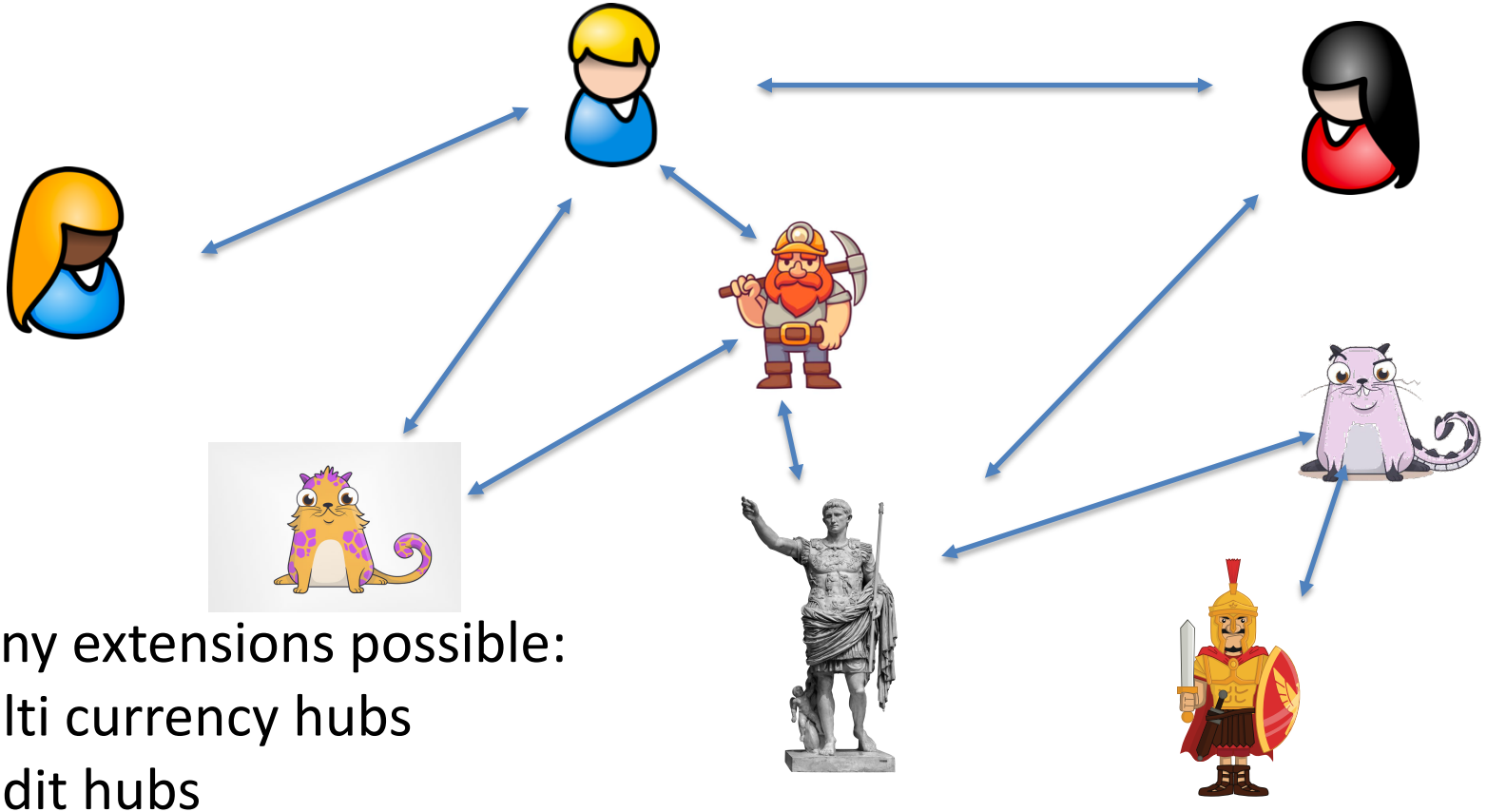
CS251 Fall 2020

(cs251.stanford.edu)

Scaling II: Rollup

Benedikt Bünz

Lightning network



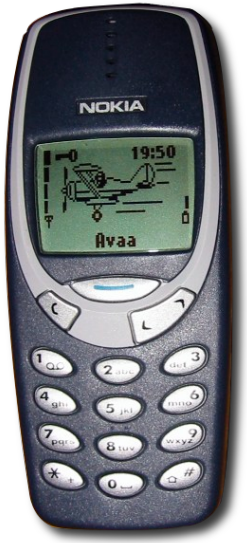
Many extensions possible:
Multi currency hubs
Credit hubs

Watchtowers

Lightning requires nodes to be periodically online to check for claim TX

Watchtowers outsource this task

User gives latest state to watchtower.

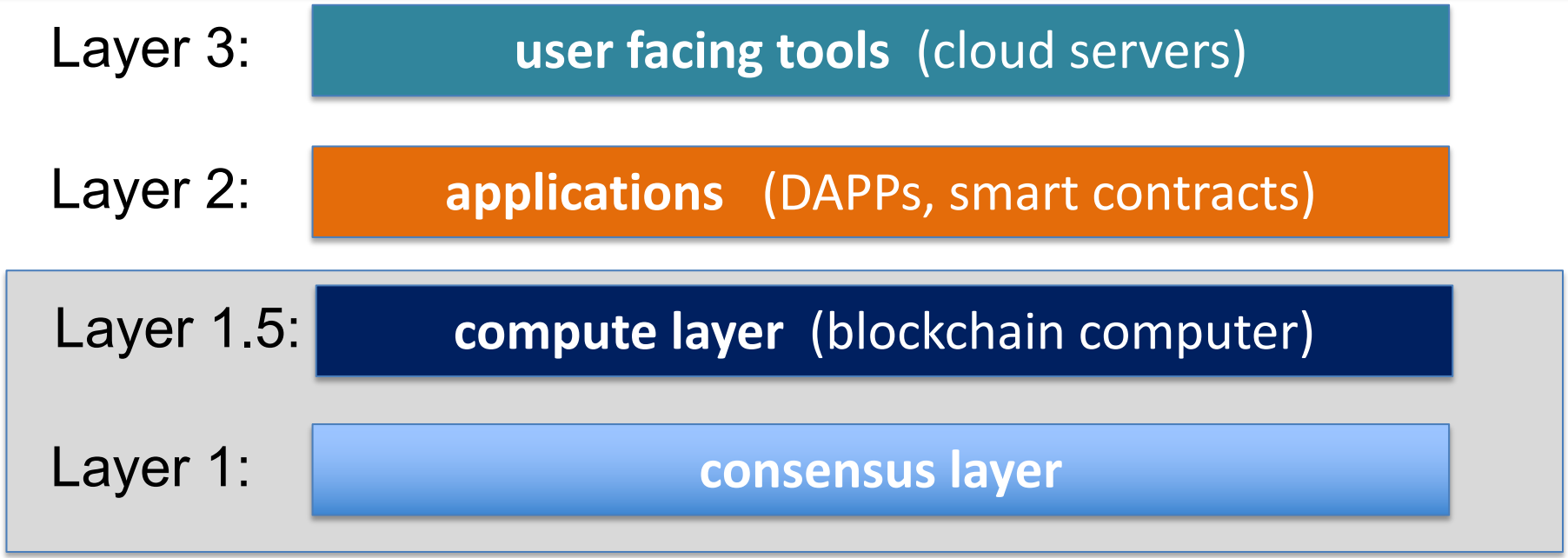


Trusted for availability not custodian of funds
Risk of bribing

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

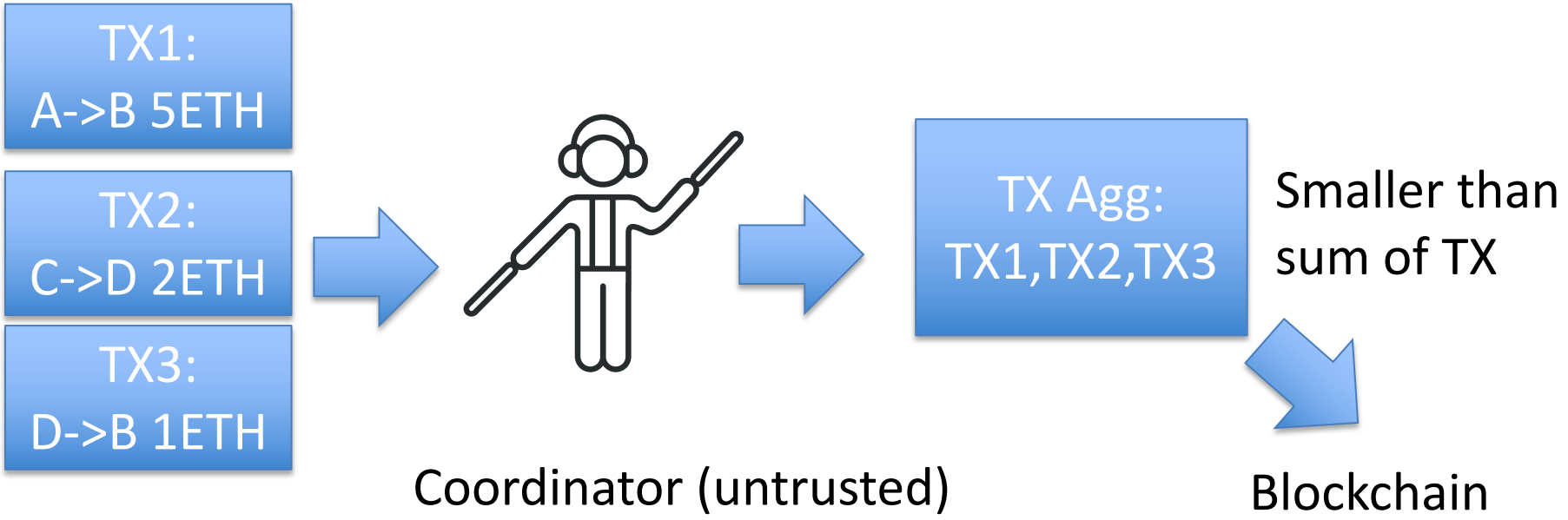
Blockchain Layers



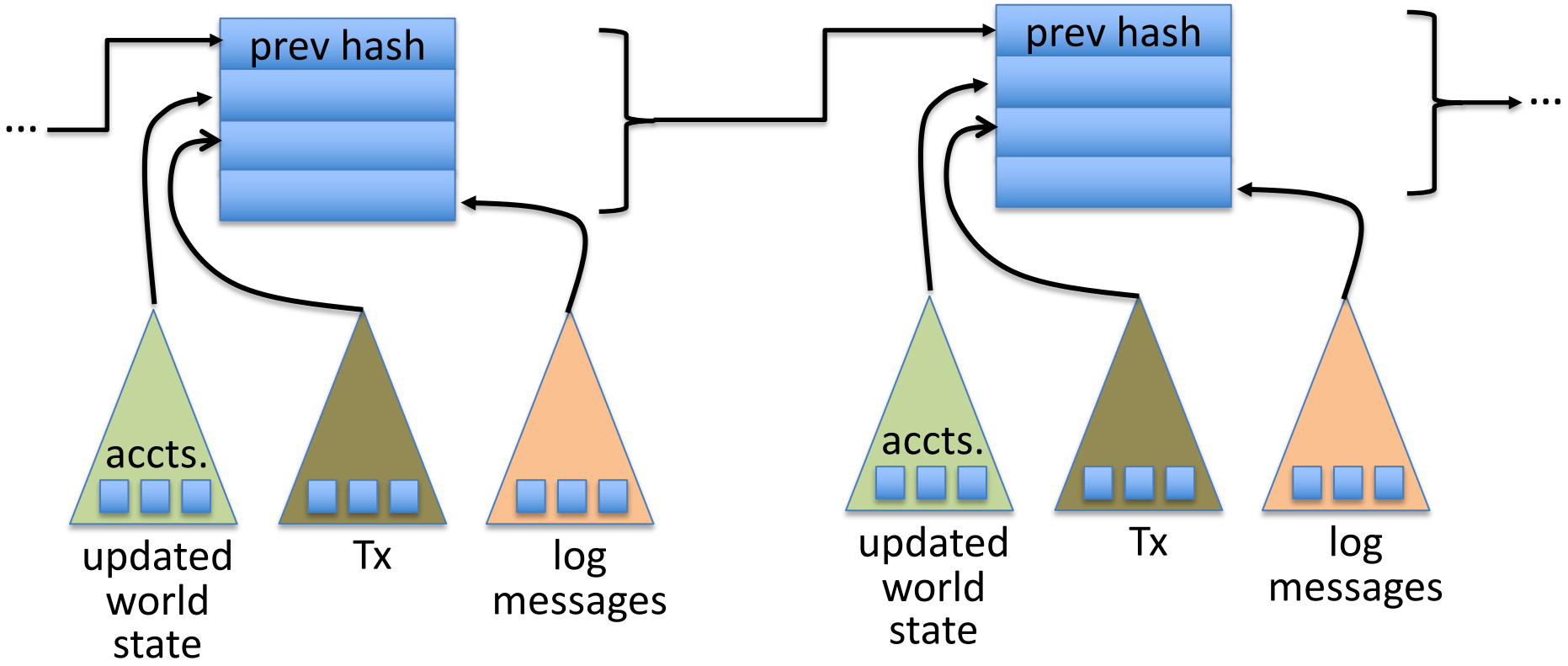
Bitcoin/Ethereum combine ordering (layer 1) and verification (1.5)
What if we can outsource verification? Makes consensus cheaper

Idea: Aggregate Transactions

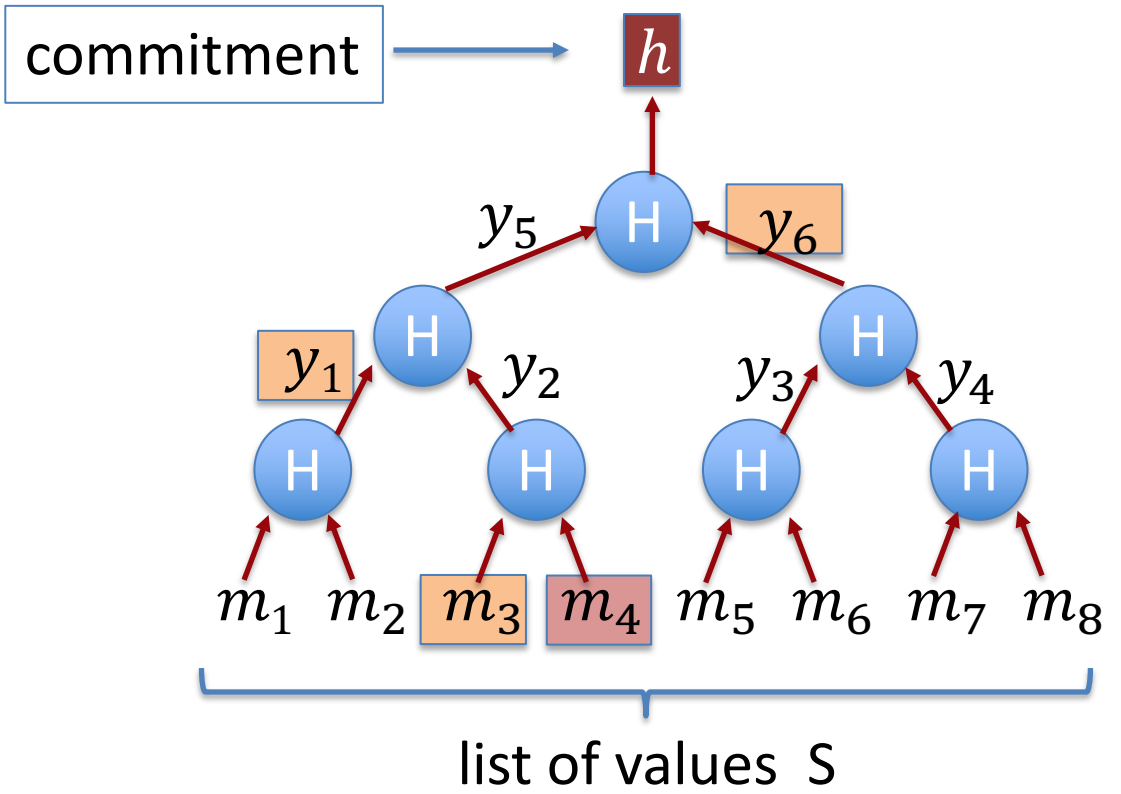
- Payment channels move more transactions offchain
- Idea: Combine Transaction, Coordinator verifies



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

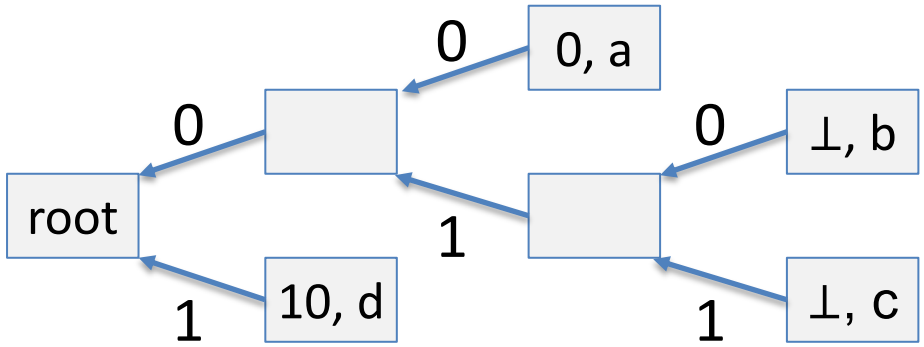
Every contract has an associated **storage array S**[]:

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



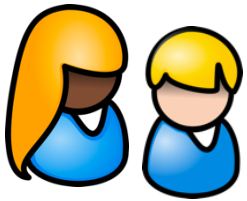
time to compute root hash:
$\leq 2 \times |S|$

$|S|$ = # non-zero cells

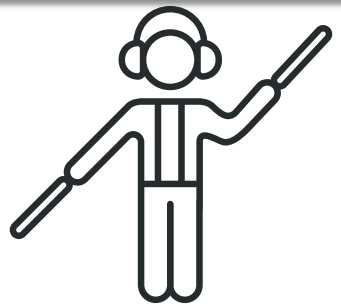
Merke (Patricia) Tree Proofs

- 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



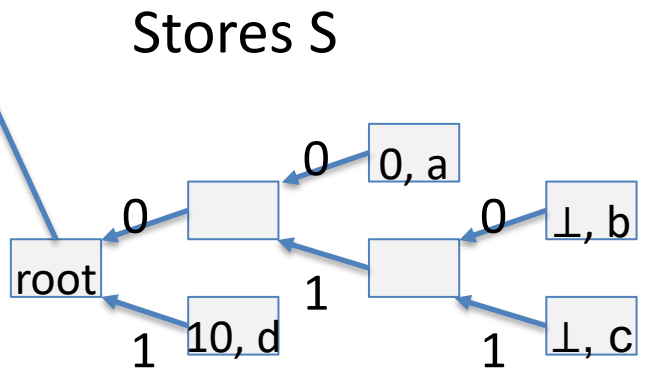
Coordinator

Rollup Smart Contract

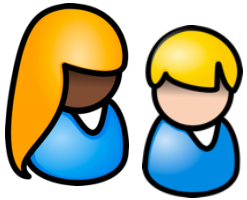
Rollup State S

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

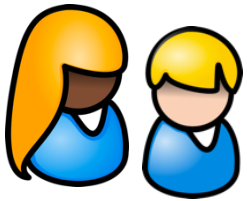
Rollup Smart Contract root

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

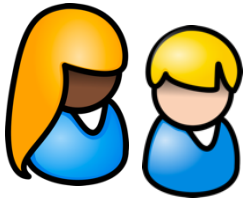
Rollup Smart Contract root

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

TX Transfer:

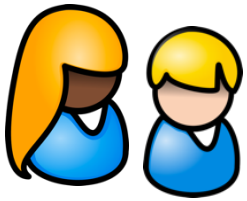
Proof that given root
S[A's PK]={3 ETH, 0}
S[B's PK]={2 ETH, 0}
Transfer amount 2 ETH
Signature by A

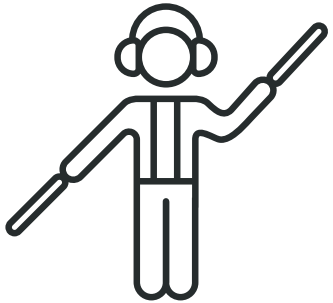
Rollup Smart Contract root

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

Verifiable Computation

```
3     function validatePatriciaProof(
4         bytes32 rootHash,
5         bytes memory key,
6         bytes memory value,
7         bytes[] memory path
8     ) pure returns (bool accept) {
```

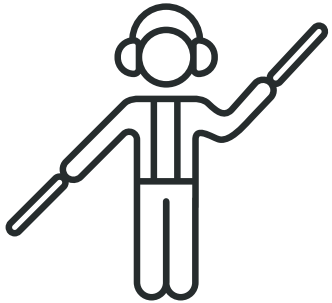




Provides Proof/SNARK that given given public inputs (rootHash, key, value) it knows private inputs (path) such that function outputs true

SNARK is short/easy to check

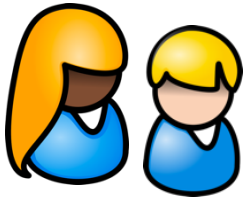
Verifiable Computation



Public

F(x,w)-> o

Private



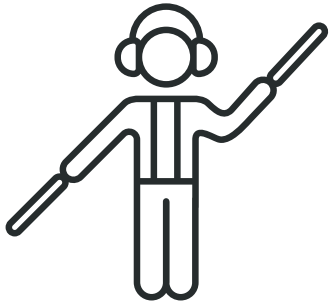
Prover

Verifier

Prove(x,o,w)→ π (SNARK)

Verify(x,o,π)→ $Accept/Reject$

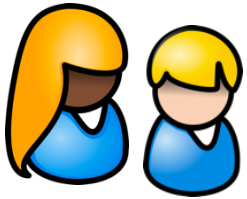
Verifiable Computation



Public

F(x,w)-> o

Private



Completeness:
Honest Prover convinces honest Veifier

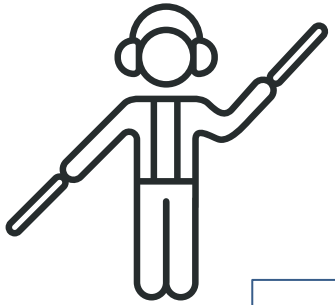
Prover

Verifier

Prove(x,o,w)→ π (SNARK)

Verify(x,o,π)→ $Accept/Reject$

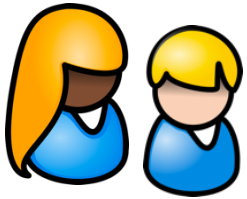
Verifiable Computation



Public

F(x,w)-> o

Private



Knowledge Soundness:
If Verifier accepts then Prover knows w such that F(x,w)= o

Prover

Verifier

Prove(x,o,w)→ π (SNARK)

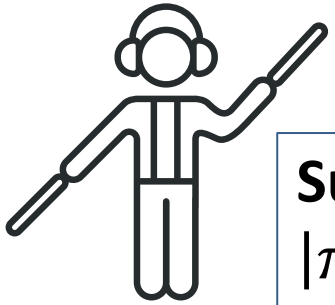
Verify(x,o,π)→ $Accept/Reject$

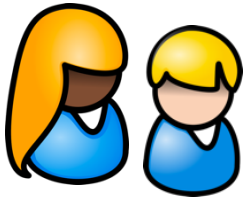
Verifiable Computation

Public

F(x,w)-> o

Private





Succinctness:

$|\pi| \ll w$ Time(Verify) $\ll$ Time(F)

Practice: $|\pi| < 100\ bits$ and

Time(Verify)=10ms

~500k Gas

Prover

Verifier

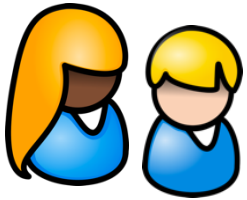
Prove(x,o,w)$\rightarrow \pi$ (SNARK)

Verify(x,o,π)$\rightarrow Accept/Reject$

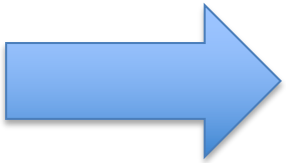
SNARKRollup

- Merkelize Transactions (omit)
- SNARK proves that given transactions I know signatures such that state transition S -> S’ valid
- No Data availability problem

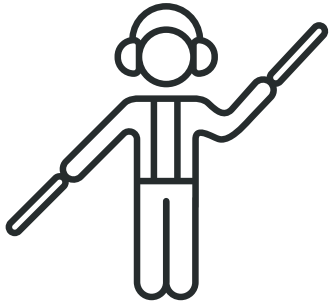
SNARKRollup (ZKRollup)



Users



Transactions



Coordinator
Stores S

root'

π

Rollup Smart Contract

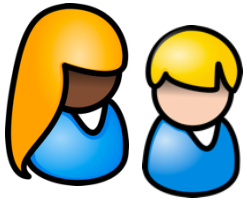
root

Commitment to S

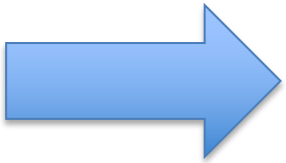
Coordinator does:

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

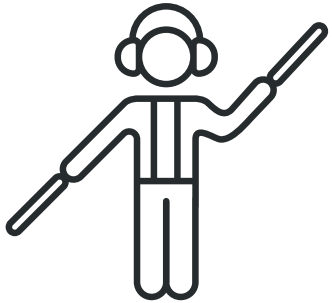
SNARKRollup (ZKRollup)



Users



Transactions



Coordinator
Stores S'

root'

π

Rollup Smart Contract

root'

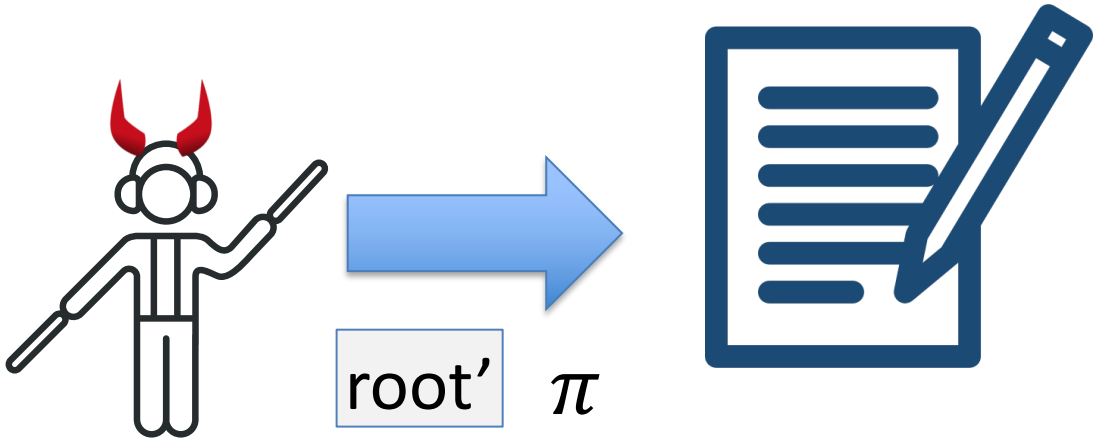
Commitment to S

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

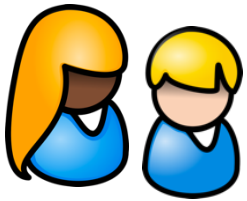


Update must be valid!

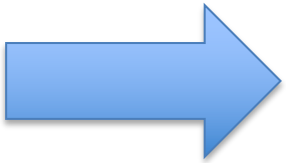
What if Coordinator does not reveal data?

Can’t update Merkle proofs

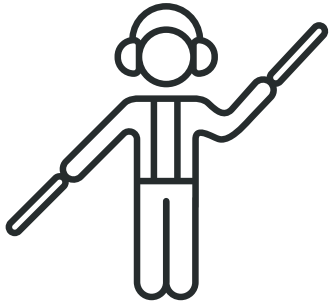
Publish diff on chain



Users



Transactions
with signatures



Coordinator
Stores S'

root'

π

txlist

Rollup Smart Contrac

root'

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)

In practice: 3600 rollup tx vs 570 normal tx per block

Cool things to do with Rollup

- SNARKRollup is cheaper than onchain tx
- Can scale to max ~300tx now, 1000tx soon
- Cost dominated by SNARK verification
- Finality ~ Blockchain finality (no instant transfer)
- Only simple transfers of value

Insurance of Rollup -> Instant Finality

- Rollup is not instant
- But if coordinator is trusted then giving them transaction -> finality
- Idea: Use insurance to achieve finality
- Coordinator signs insurance
- If transaction not included in next (few) blocks insurance can be used to get insurance premium

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

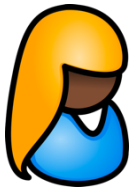
Txlist= [{A-> B 3 ETH and B-> A 2 DAI}, {D-> B 1 BAT}]

Exchanges

Buy 3 ETH for 5 DAI

Exchanges match orders

Classical exchanges also store funds



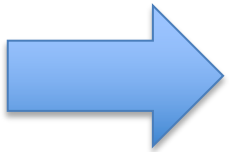
Sell 3 ETH for 5 DAI

Order book

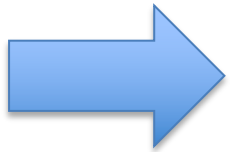
	give		get	
Alice	3	ETH	5	DAI
Bob	5	DAI	3	ETH
Carol	4	BAT	10	DAI

Rolled up Exchange

Submit orders



Has orderbook
Matches orders
Rolls up transactions as atomic swaps



Txs Root, π

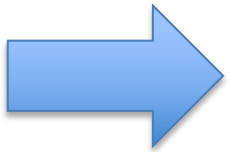
Verifies π

Root of balance tree

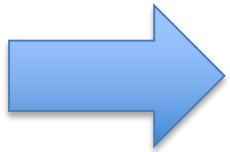
Exchange trusted for honest matching

Rolled up Exchange

Submit orders



Updates orderbook tree
on chain and proves
correct matching



Txs Root, π

Root of balance tree
Root of orderbook tree

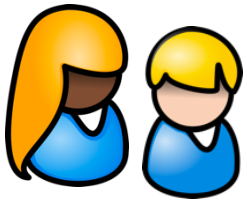
Verifies π

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

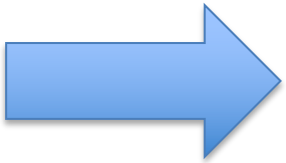
SNARKRollup Problems

- Creating SNARKs is very expensive
  - Only simple TX possible
  - No arbitrary SMART Contracts
  - SNARKs are improving all the time (hot research area)
- SNARK verification is expensive on chain
  - 500k gas -> 1.5k gas/tx
  - Likely to get better soon

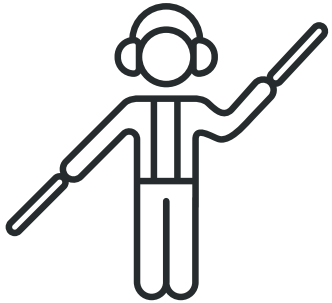
Optimistic Rollup



Users



Transactions
with signatures



Coordinator
Stores S'

root'

π

txlist

Rollup Smart Contrac

root'

What if we remove the SNARK?

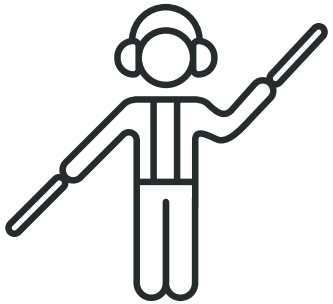
Idea: Instead of proving correctness, prove fraud!

New Role: Validator checks correctness, provides fraud proofs

Optimistic Rollup

- Coordinator updates transaction root
- Coordinator adds high bond
- 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 prove fraud?

Fraud Proofs



Coordinator

root

Commits to state S

txlist

root’

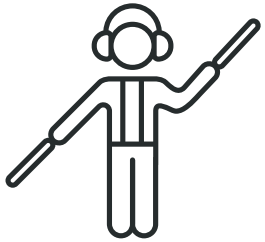
Validator

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

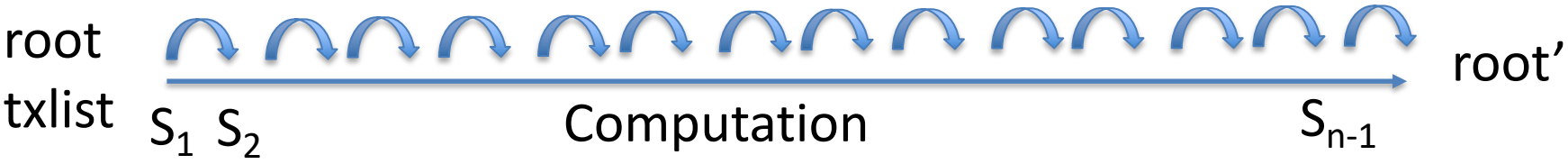
Idea: Coordinator and Validator find first point of disagreement



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

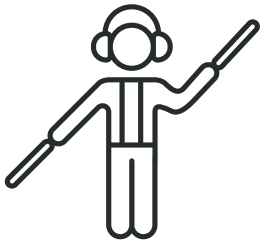
Validator does the same

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



Referee Delegation

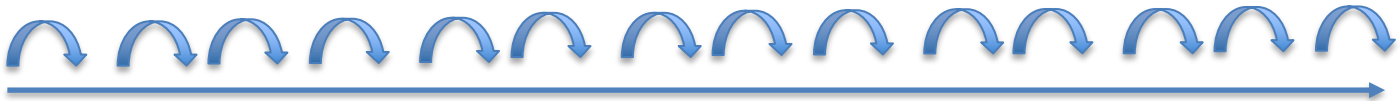
Coordinator and Validator run interactive binary search



$S_{n/2}$

Checks whether
$S_{n/2}=S'_{n/2}$
If yes disagreement in fist half
Otherwise in second

root
txlist

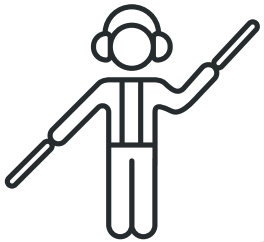


S_1 S_2 Computation S_{n-1} S_n

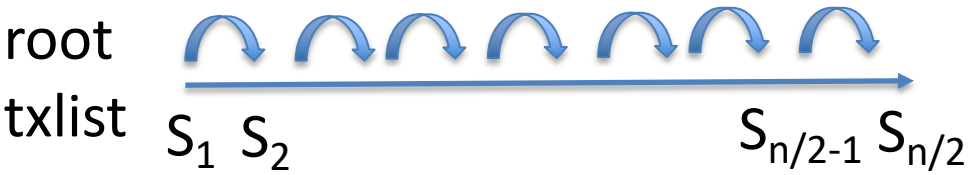
root'

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}



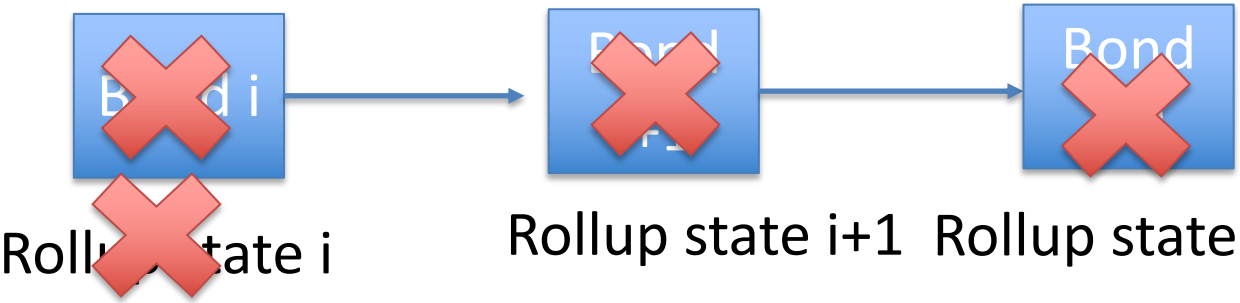
Smart Contract checks transition between S_i and S_{i+1} and declares winner

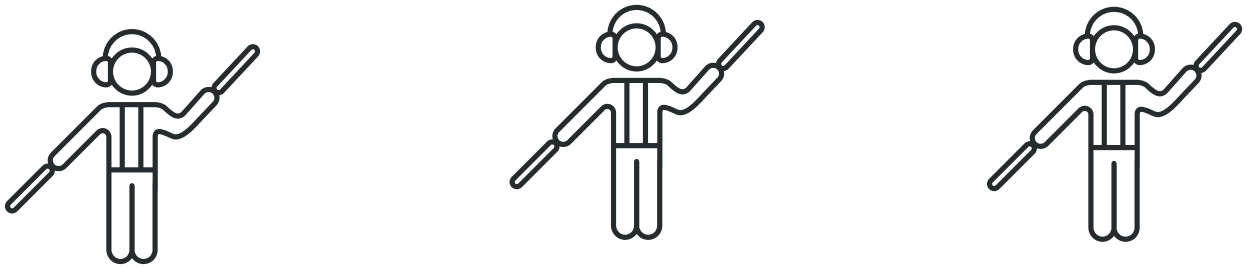


Problem: Checks take a long time

- $\log_2(n)$ 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!

Pipelined Assertions

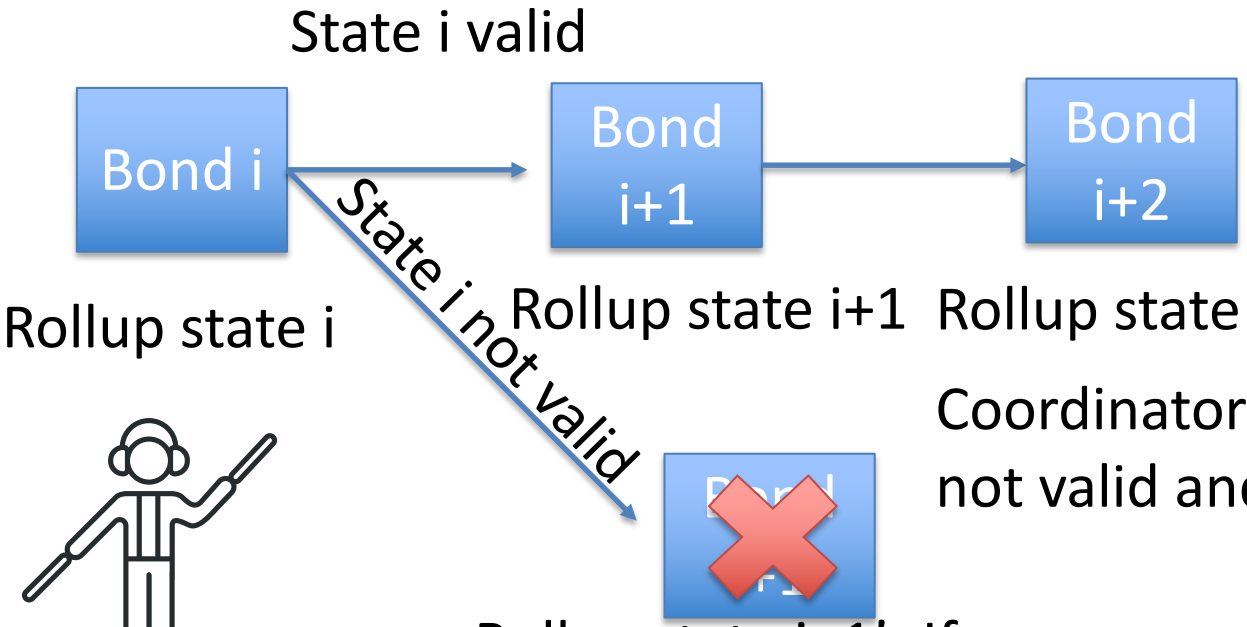




Coordinators can build on states before timeouts

If prior state invalid, all subsequent bonds are slashed

Pipelined Assertions



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

If no successful fraud proof then reward gets slashed

Multiple Rollup Coordinators

- Rollup coordinator (in either scheme) is not trusted for security
- It can reasonably be a single coordinator
- But it is trusted for liveness
  - Censorship resistance
  - Progress of rollup state
- Multiple Coordinators?

Multiple Rollup Coordinators

- Rotating coordinators
- Random coordinator (using Beacon)
- Race to submit new rollup state (usually same party wins)
- One solution is using classical consensus between fixed set of coordinators
  - At least 2/3rd of coordinators sign roll up
  - If trusted instant finality

Multi Coordinator Insurance

- Get insurance signature from 2/3rd of coordinators
- If next block does not include transaction post signature
- Slash reward from intersection of insurer and rollup block signers
  - At least 1/3rd of the coordinators

END OF LECTURE

Next lecture:

Privacy 1: Tracing transactions and Mixers