CS251 Fall 2021

(cs251.stanford.edu)

Recursive SNARKs

Benedikt Bünz

Recap: Non-interactive Proof Systems

A **non-interactive proof system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier

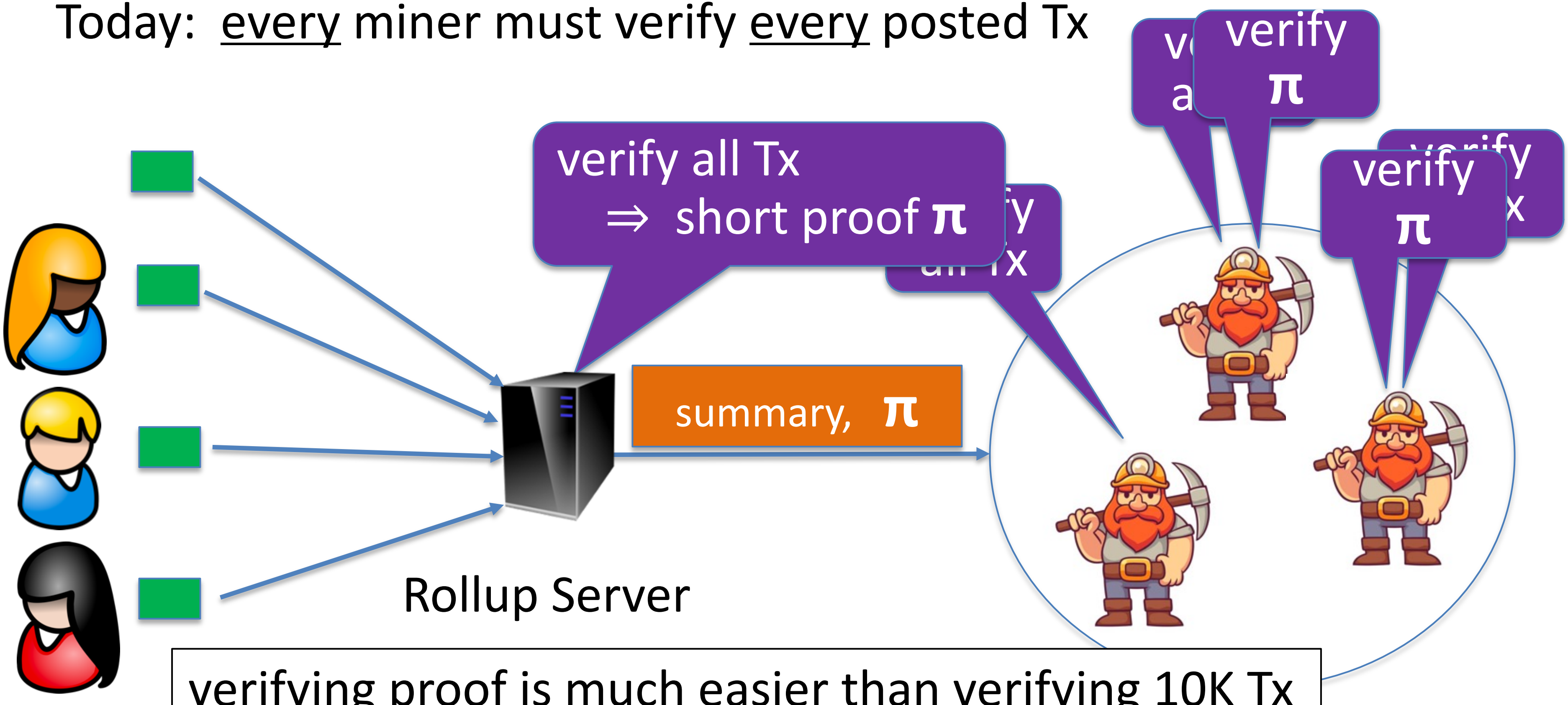
  (S_p, S_v) is called a *reference string*

- **P**(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) → proof π

- **V**(S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$) → accept or reject

Recap: zkRollup

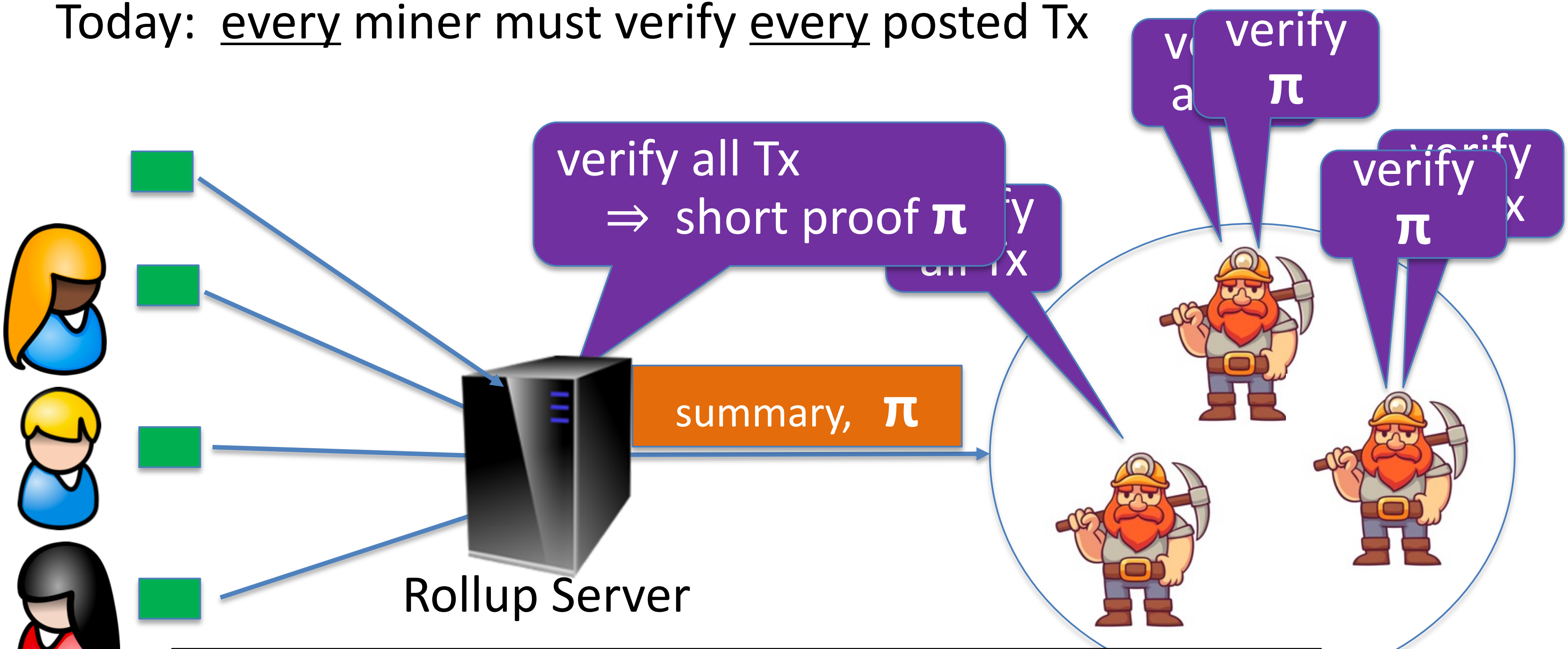
Today: every miner must verify every posted Tx



verifying proof is much easier than verifying 10K Tx

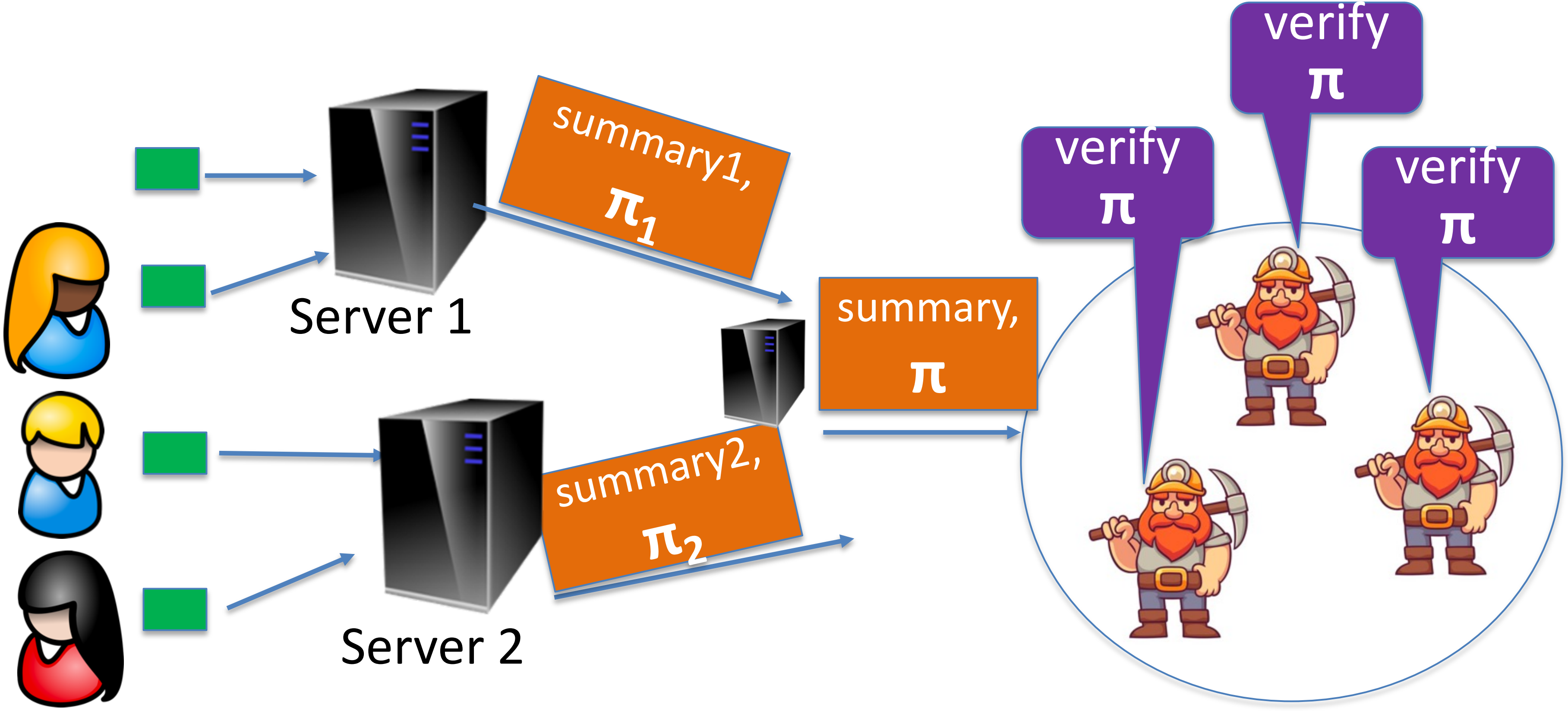
Recap: zkRollup

Today: <u>every</u> miner must verify <u>every</u> posted Tx



verifying proof is much easier than verifying 10K Tx

Rollup with many coordinators



Zk-zk-Rollup

- Multiple servers
- Each responsible for subset of users (no overlaps)
- Rollup aggregator (can be one of the servers)
- Rollup aggregator combines summaries (balance table) and creates one proof that
- How can we combine proofs?
- Trivial solution:
  - All servers forward all Tx
  - Rollup aggregator creates one SNARK
  - Does not save work

Recap: Non-interactive Proof Systems

A **non-interactive proof system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier

  (S_p, S_v) is called a *reference string*

- **P**(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) → proof π

- **V**(S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$) → accept or reject

SNARK of a SNARK Proof

A **non-interactive proof system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier

  (S_p, S_v) is called a *reference string*

- **P**(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) → proof π

- **V**(S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$) → accept or reject

SNARK of SNARK

How can we aggregate proofs?

$$S(C) \rightarrow S_P, S_V$$
$$\pi \leftarrow P(S_P, x, w)$$

Now write a circuit C' that verifies π:

- Input x' is x
- Witness w' is π
- $C'(x', \text{w}') = 0$ iff V(S_V, π, x)=Accept

Finally:

$$S(C') \rightarrow S'_P, S'_V$$
$$\pi' \leftarrow P(S'_P, x', w')$$

SNARK of SNARKs

How can we aggregate proofs?

$$S(C) \rightarrow S_P, S_V$$

$$\pi_1 \leftarrow P(S_P, x_1, w_1)\ \pi_2 \leftarrow P(S_P, x_2, w_2)$$

Now write a circuit C' that verifies π:

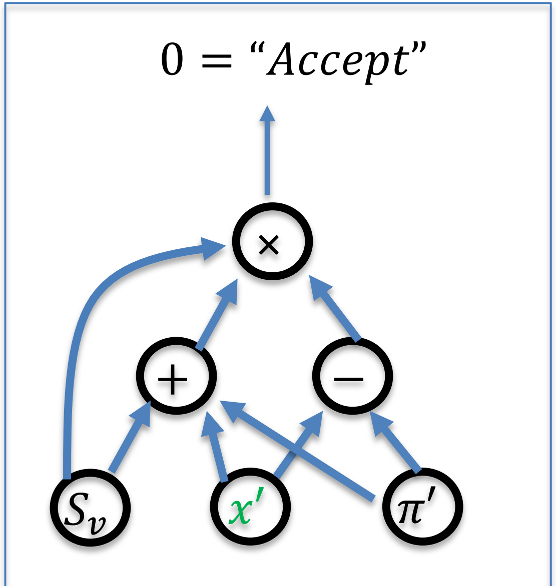
- Input x' is $x_1||x_2$
- Witness w' is $\pi_1||\pi_2$
- $C'(x', w')$ = 0 iff V(S_V, x_1, π_1)=Accept and V(S_V, x_2, π_2)=Accept

Finally:

$$S(C') \rightarrow S'_P, S'_V$$
$$\pi' \leftarrow P(S'_P, x', w')$$

SNARK of SNARKs

- Note that C' depends only on **V** and S_v (not on C_1,C_2)
- We can express **V** as a circuit:



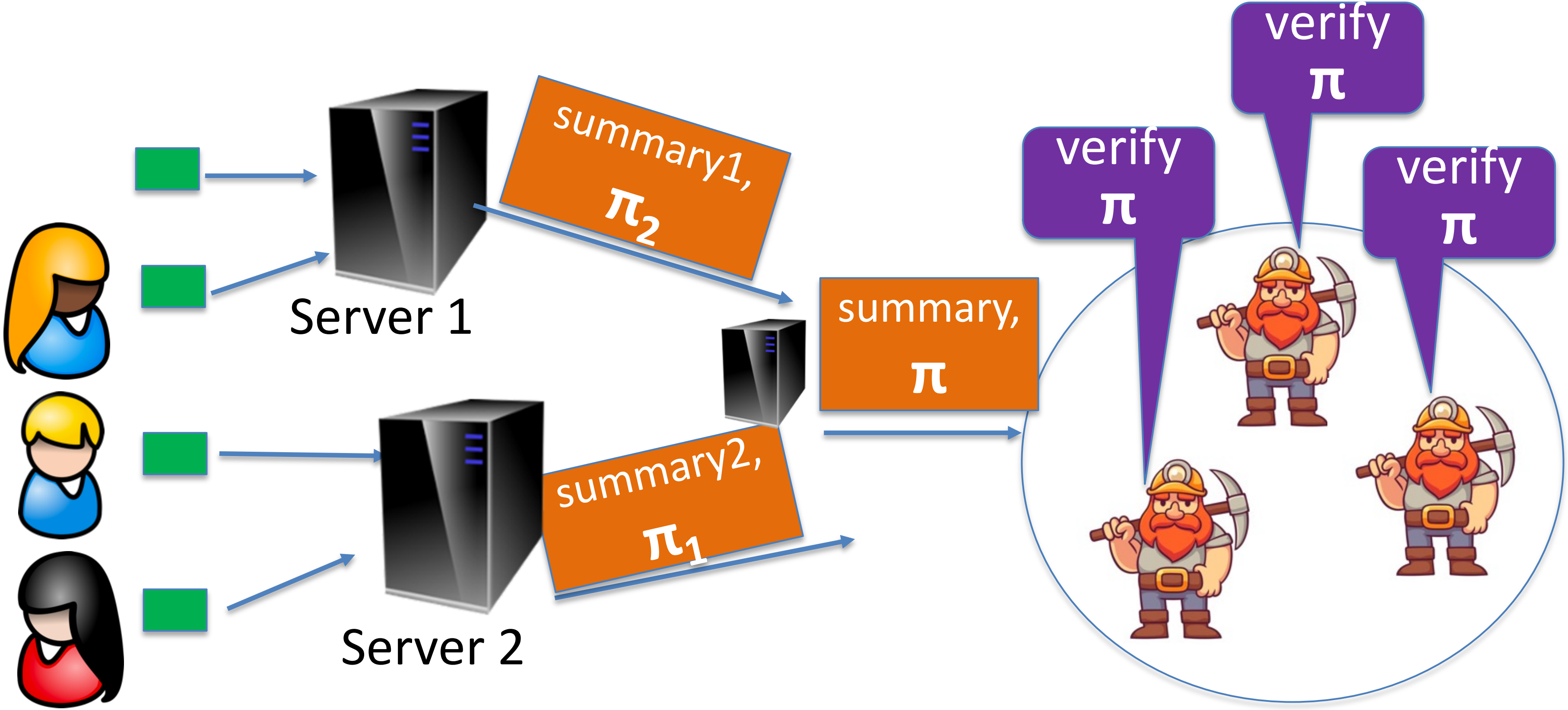
$w' = \pi'$ is independent of w_1, w_2

|C'|=2*|V| < |2* C|

Building SNARK of SNARKs

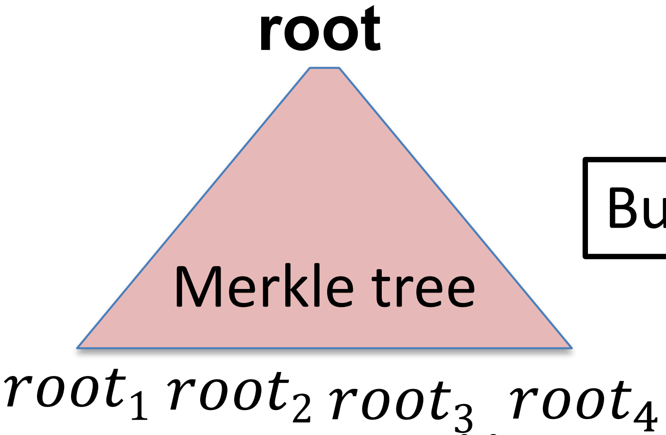
- How big is C'?
- Comparison |SHA256 circuit| = 20k gates
- First SNARK of SNARK ~1 million gates with trusted setup (BCTV14)
- Today less than 50k gates (Halo, BCLMS20, Nova)
    - no trusted setup
- Independent of inner SNARK circuits!

Rollup with many coordinators



Zk-zk-Rollup

- Let **root**$_i$ be the Merkle Tree Root of summary i



Build one root from summaries

- $S_V, S_P \leftarrow \boldsymbol{S}(C_R)$ // C_R rollup circuit

$C_{zk^2}(\mathrm{x} = S_V, \mathbf{root};\ \mathrm{w} = root_1, root_2 \ldots, \pi_1, \pi_2, \ldots)$:
V$(S_V, x = root_i, \pi_i)$ for all i and **root**=MT($root_i$s)

Tornado cash

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

nf, **proof π, A** →
(over Tor)

← **100 DAI**
to address **A**

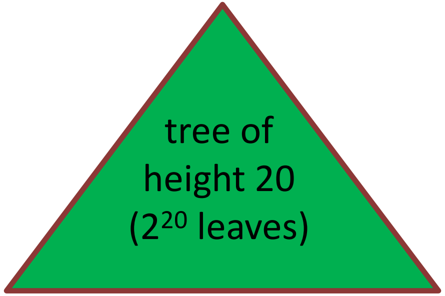
Treasury: **300** DAI

coins Merkle root	nullifiers
(32 bytes)	nf_1
next = 5	nf_2
	nf

contract state

H_1, H_2: R → $\{0,1\}^{256}$

Merkle root



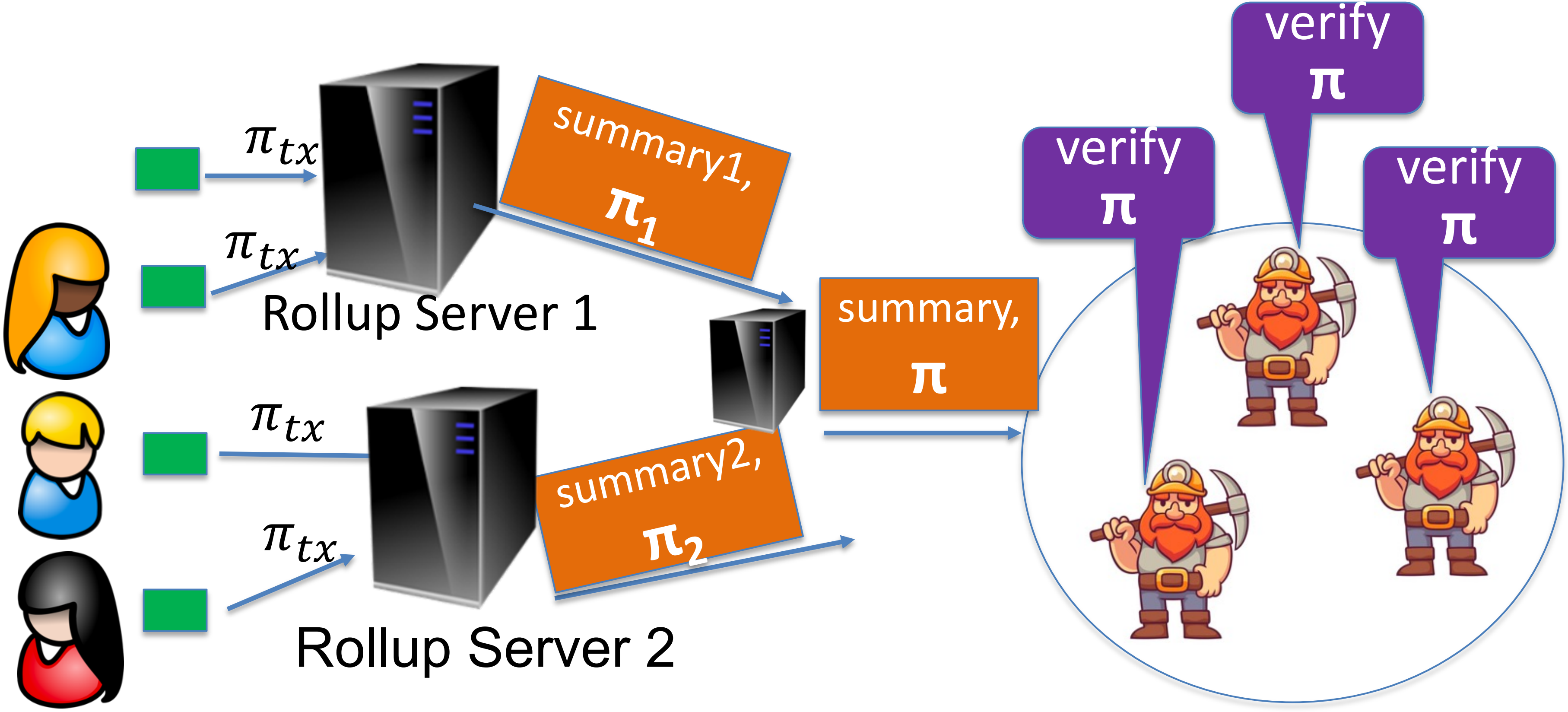
C_1 C_2 C_3 C_4 0 … 0

public list of coins
… but observer does not know which are spent

nf and π reveal nothing about which coin was spent.

But, coin #3 cannot be spent again, because **nf = H_2(k')** is now nullified.

zk³-Rollup (tornado cash rollup)



zk³-Rollup

- Users create SNARK for TC Circuit C_{TC}
  - $\mathrm{S_V}, S_P \leftarrow \boldsymbol{S}(C_{TC})$
  - $\pi_{TC} \leftarrow P(S_P, tx, w)$
- Rollups create SNARKs for $C_R = \forall_i\ V(S_V, tx_i, \pi_i) = \text{“}accept\text{”}$
  - $tx\ root = MT(tx_1, \dots, tx_n)$
  - $\pi' = \pi_{TC,1} || \dots || \pi_{TC,n}$
  - $\mathrm{S_V}', S_P' \leftarrow \boldsymbol{S}(C_R)$
  - $\pi_R = P(S_P', tx\ root, \pi')$
- Rollup Aggregator creates SNARK for $C_A = \forall_i\ V(S_V', root_i, \pi_{R,i})$
  - $\mathrm{S_V''}, S_P'' \leftarrow \boldsymbol{S}(C_A)$
  - $root = MT(root_1, \dots, root_k)$
  - $\pi_R' = \pi_{R,1} || \dots || \pi_{R,k}$
  - $\pi_A = P(S_P'', root, \pi_R')$

Enhancing transactions with SNARKs

- We've seen that private transactions require zero-knowledge proofs
- Add ZK-SNARKs to every transaction
- Level 1 coordinators verify transaction by verifying transaction ZK-SNARKs
- Additionally, we can have more complicated transactions (Smart Contracts)
  - Transaction verification is constant time regardless of proof complexity
- *Can we also hide the smart contract?*

ZEXE private execution

- ZEXE is a model of computation (like UTXOs/Scripts or Accounts/EVM)
- The basic unit is a record (similar to a UTXO)
- Every transaction consumes records and creates records
- Universal predicate: Prevents double spends
- Birth predicate: Says how a record can be created
- Death predicate: Says how a record can be consumed

ZEXE private execution

Record 1:
Birth predicate 1
Death predicate 1
Payload 1

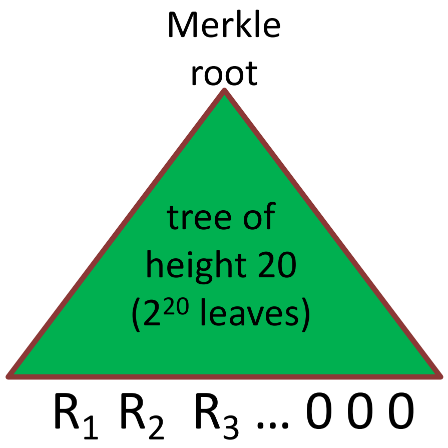
Record 2:
Birth predicate 1
Death predicate 1
Payload 1

Record 3:
Birth predicate 3
Death predicate 3
Payload 3

TX checks that Record 1 and Record 2 have not been spent
Birth3(R1, R2,R3) and Death1(R1, R2,R3) and Death2(R1,R2,R3)

ZEXE private execution

- Universal predicate (similar to tornado cash)
  - Uses nullifiers
  - Checks that nullifier=H(sk,records) is properly created
  - Checks that nullifier only appears once
  - Prevents double spends



Implementing assets with ZEXE

- Record payload has a value v and an asset id
- Birth predicate
  - Defines the token
  - New record id needs to match consumed predicate ids
  - New record value is sum of inputs
- Death predicate
  - Defines the SCRIPT
  - E.g. spendable by signature
  - E.g. Spendable by multisigature + preimage of hash

Implementing smart contracts with ZEXE

- Record payload is state of smart contract, smart contract instance id
- Birth predicate
  - Either creates smart contract or
  - One of the inputs needs to be the old smart contract record
- Death predicate
  - Defines the smart contract logic

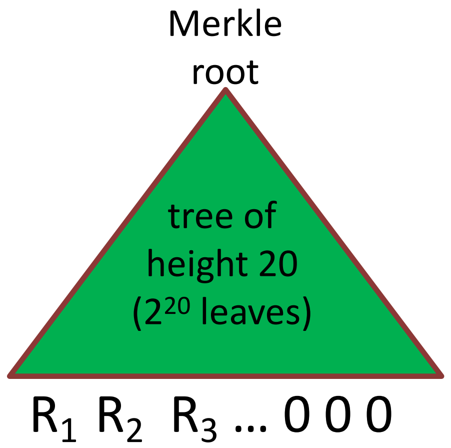
ZEXE game of Chess

- Record payload is state of smart contract, smart contract instance id
- Birth predicate
  - Starts new game (and assigns pks to black/white) or
  - One of the inputs needs to be the old chess game
- Death predicate
  - If game finished then pay money to the winner
  - Otherwise input records must be game record + one move record
  - Move record must be signed by the right player
  - Move record must contain a valid move

Making ZEXE private

- $S_{P_U}, S_{V_U} \leftarrow S(C_U)$ (Universal predicate)
- $S_{P_B}, S_{V_B} \leftarrow S(C_B)$ (Birth predicate)
- $S_{P_D}, S_{V_D} \leftarrow S(C_D)$ (Death predicate)
- $S_{P_{TX}}, S_{V_{TX}} \leftarrow S(C_{TX})$ (TX circuit)
- $C_{TX} = V\left(S_{V_U}, ., .\right)$ =0 and $V\left(S_{V_B}, ., .\right)$ =0 and $V\left(S_{V_D}, ., .\right)$ =0

And Record=$H(payload, S_{V_B}, S_{V_D}, r)$ // r random



MT of all records

- TX: Input records || Output records
- Compute nullifiers $nf_1, \dots, nf_n$ from input records
- To create a TX, create three ZK-SNARKS (now ZK is important)
  - x=TX, $w = payloads, S_{V_B}, S_{V_D}$
  - $\pi_U \leftarrow P\left(S_{P_U}, \mathrm{x} \,||\, nf_1, \dots, nf_n, w \,||\, MT\ proofs\right)$
  - $\pi_B \leftarrow P\left(S_{P_B}, \mathrm{x}, w\right)$
  - $\pi_D \leftarrow P\left(S_{P_D}, \mathrm{x}, w\right)$
- Create $\pi_{TX} \leftarrow P\left(S_{P_{TX}}, \mathrm{x}, w \,||\, \pi_U, \pi_B, \pi_D\right)$

Birth and death predicate as well as records are private!

Hitchhikers guide to the galaxy

What if we want to verify that computation?

Input → Output (42)

Long Computation

SNARKs for long computations

Issues:

-P takes very long

-Starts after proving ***after*** computation finished

-Can't hand off computation

-S also runs at least linear in |C|

(ok if many proofs)

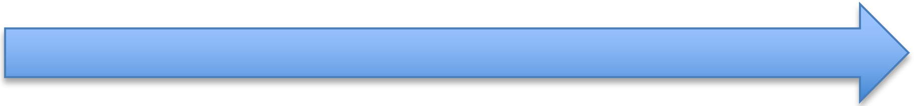
C – Circuit for long computation

S(C) → (S_p, S_v)

$\boldsymbol{x = (input, output)}$

$\boldsymbol{w = transcript}$

Input



Long Computation, Transcript

Output (42)

P(S_p, $\boldsymbol{x, w}$) → π

V(S_v, $\boldsymbol{x}, \pi$) → accept

Handing off computation

C_I– Circuit for ~~long~~ intermediate computation

S(C_I) → (S_p, S_v)

$x_1 = (input, int_1), w_1 = transcript_1$

$x_2 = (int_1, int_2), w_2 = transcript_2$

$x_3 = (int_2, output), w_3 = transcript_3$

P(S_p, x_i, w_i) → π_i

V(S_v, x_1, π_1)

V(S_v, x_2, π_2)

V(S_v, x_3, π_3)

$|\pi|$/ **V** linear in #handoffs

Input → Int$_1$,π_1 → Int$_2$,π_2 → Output (42), π_3

transcript$_1$ transcript$_2$ transcript$_3$

Incremental Proofs

- We need updatable/incremental proofs

C_I– Circuit per computation step, t number of steps/handoffs

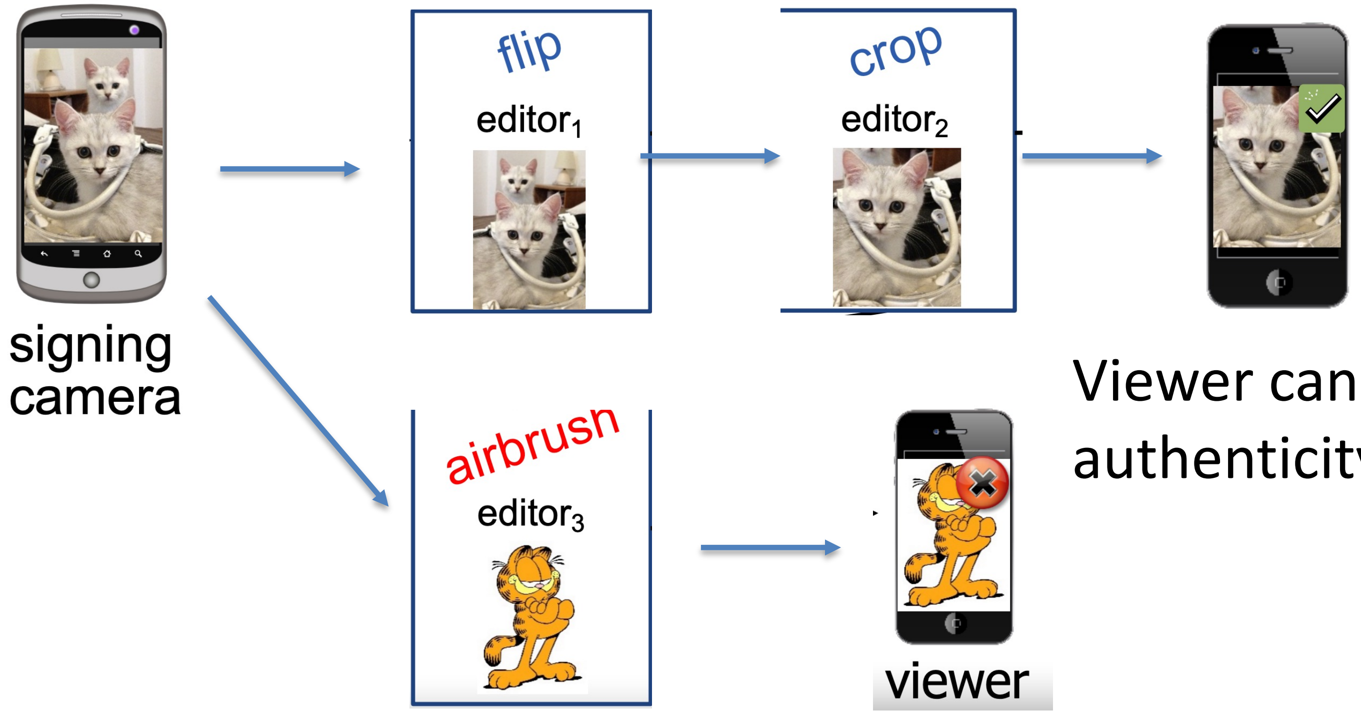
S(C_I) → (S_p, S_v)

P(S_p, x_i, w_i, π_{i-1}) → updated proof π_i // $\pi_0 = \perp$

V(S_v, x_0, x_t, π_t, t) → accept/reject

$|\pi_i|$ = $|\pi_{i-1}|$ // proofs don't grow

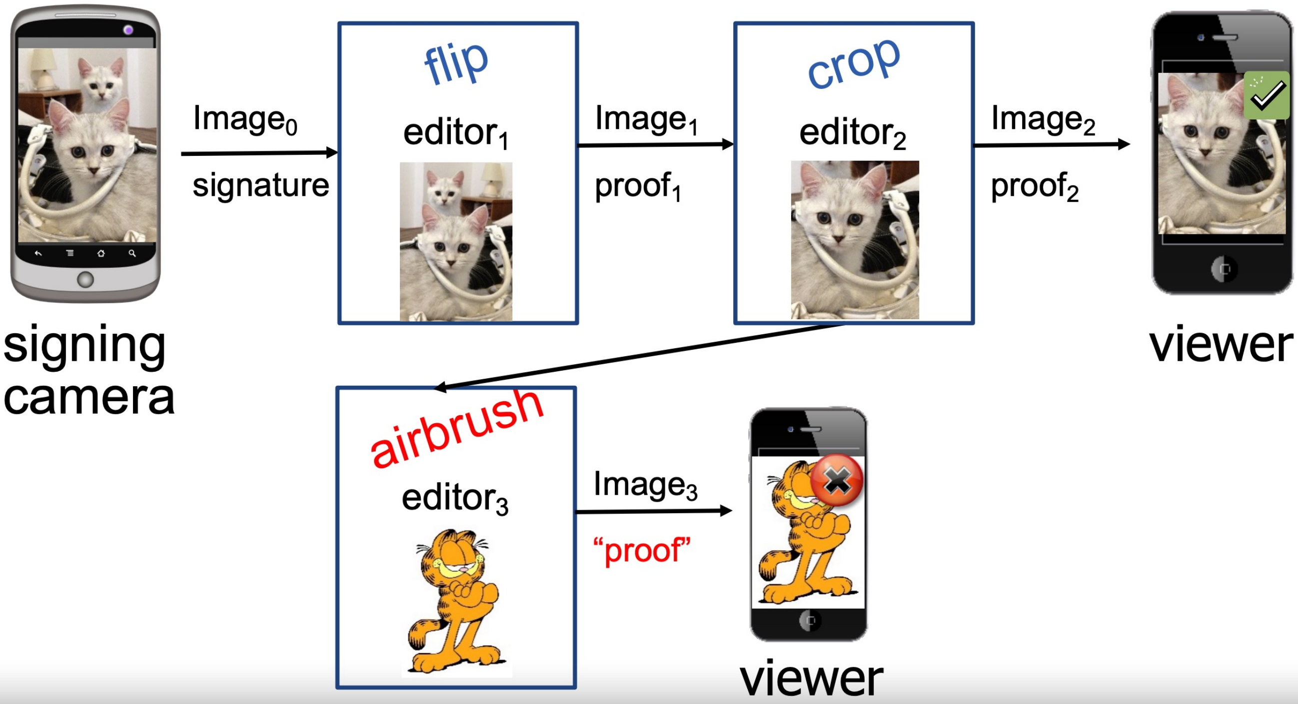
PhotoProof



Viewer can still verify authenticity

Allow valid updates of photo and provide proof

PhotoProof

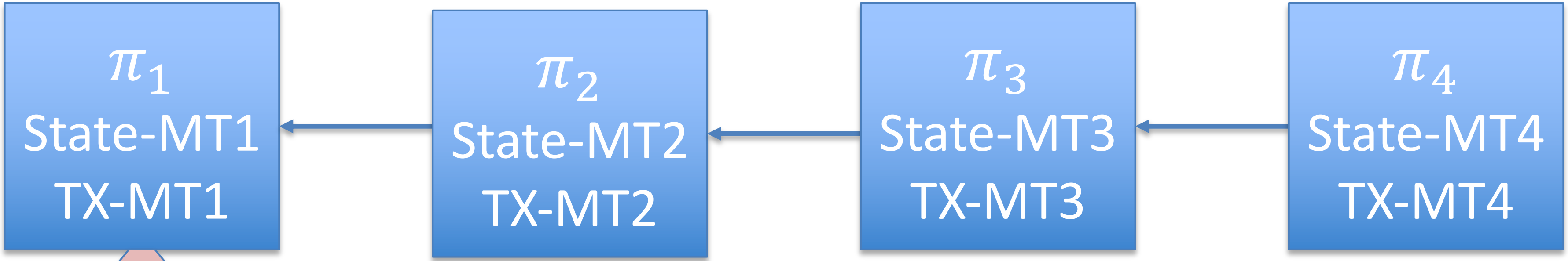


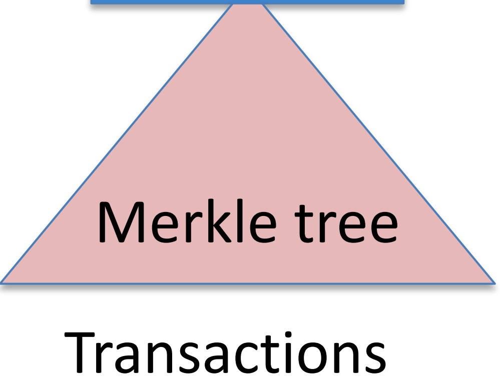
Proof allows valid edits only, Incrementally updated

Constant size blockchains

- Rollup reduces the verification cost
- Still linear in the number of state updates
- When a node joins the network they need to verify one rollup proof per block!
- In general starting a full node requires verification of all blocks
    - Can take days!

Constant size Blockchain



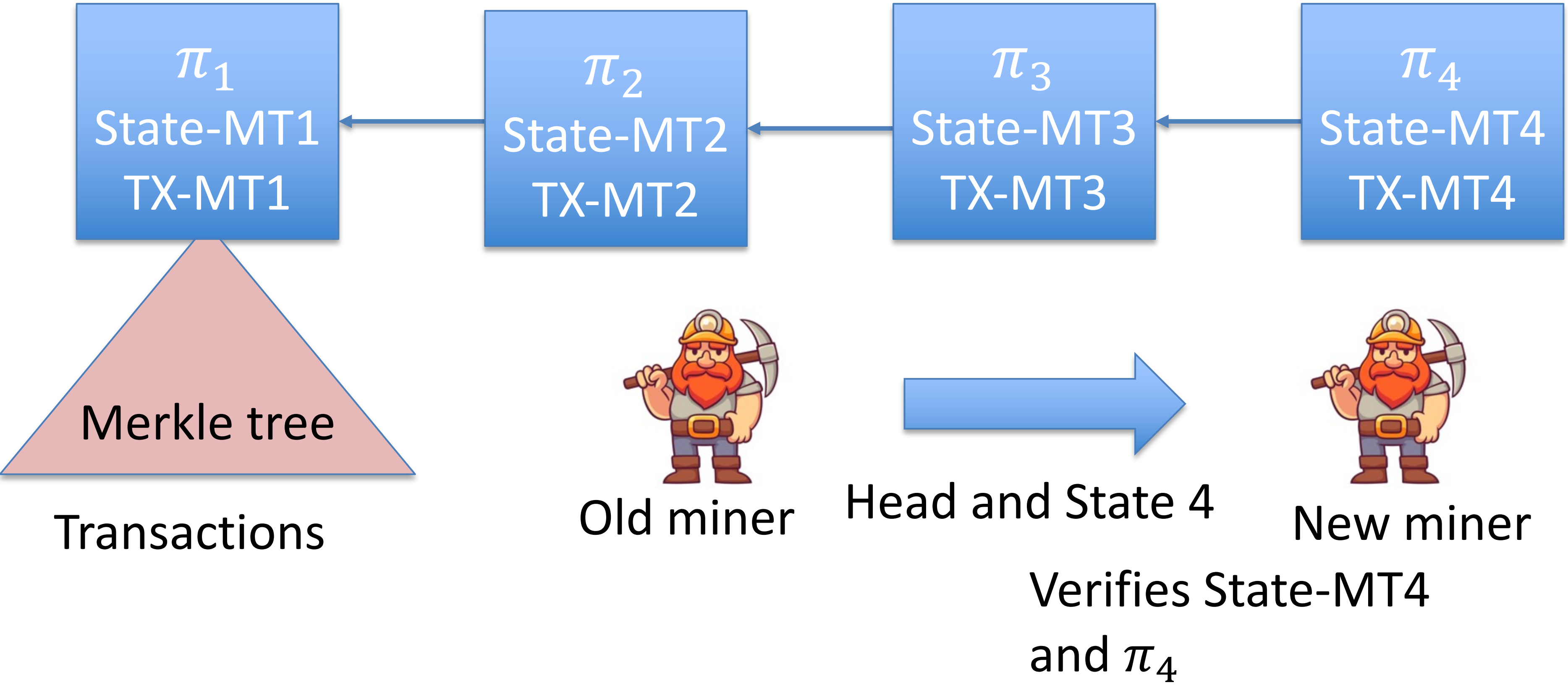


π_i prooves that transactions are valid with respect to the state

AND

π_{i-1} was valid for the previous block

Constant size Blockchain



Constant size Blockchain

- Light clients can verify every block!
  - Low memory, low computation
  - Independent of length of chain or #transactions
- Relies on data serving nodes for synching

- Practical today!

END OF LECTURE

Next lecture: Crypto tricks and open discussion
Please attend last two lectures if you can