Lecture #13:
Light and Stateless Clients

COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Lecture #13

1. Block headers and light clients.
   – headers = metadata, light clients = know only headers

2. Simplified payment verification (SPV).
   – verifying properties of state from metadata + proofs
   – UTXO-based and accounts-based models

3. Stateless clients and statelessness.
   – checking block validity without knowing the state

Block Headers

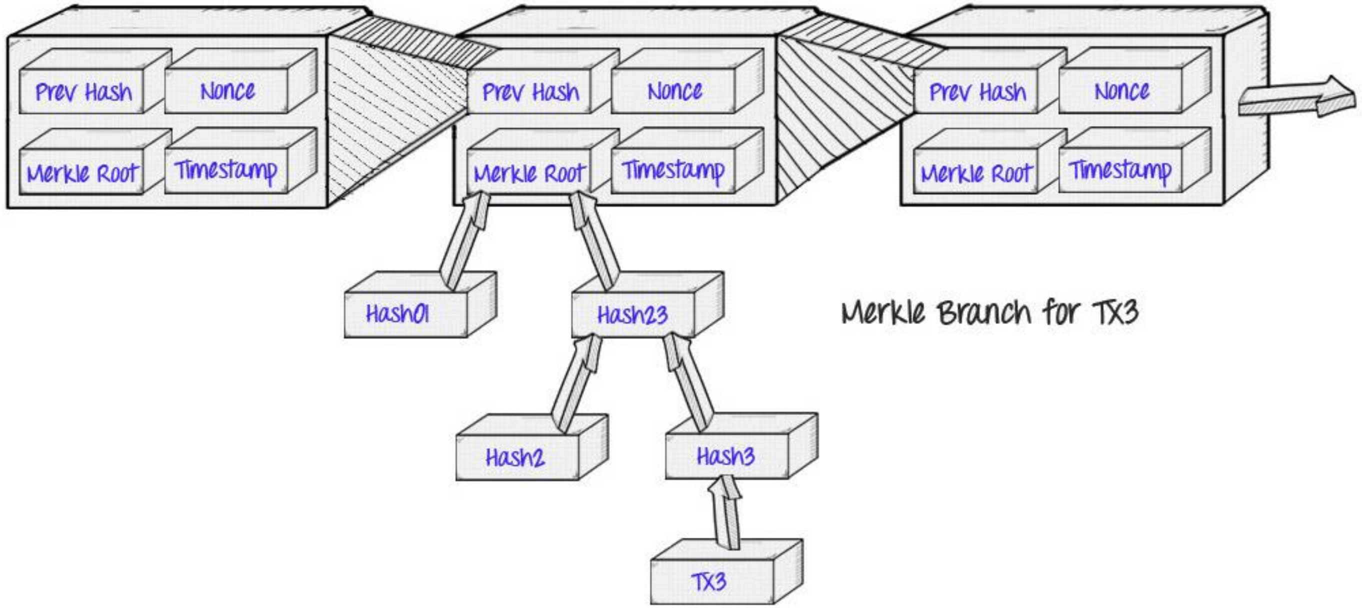
- block = header (metadata) + transactions (payload)

Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper

Block Headers

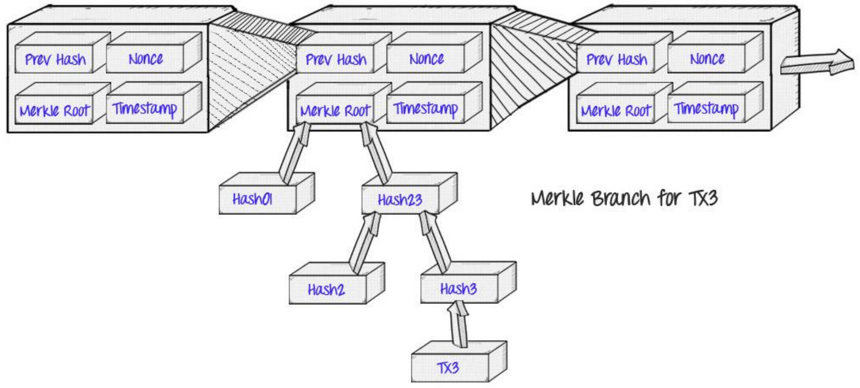
- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper



From "Merkling in Ethereum" by Vitalik Buterin https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

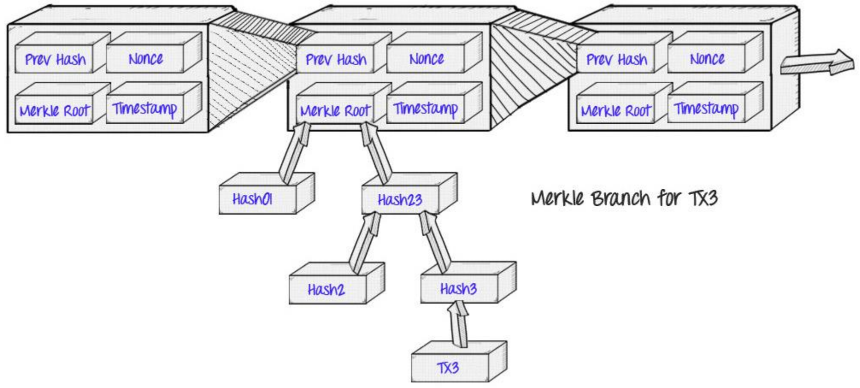
Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper
- “name” of block := hash of its header



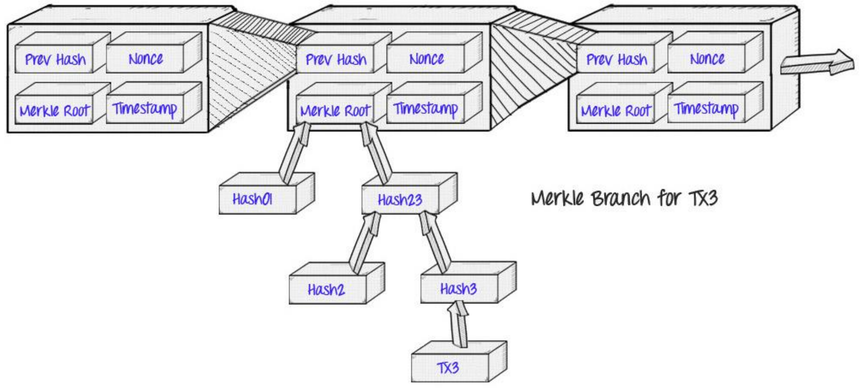
Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper
- “name” of block := hash of its header
- typical ingredients of block header:
  - predecessor (specified by name, as above)



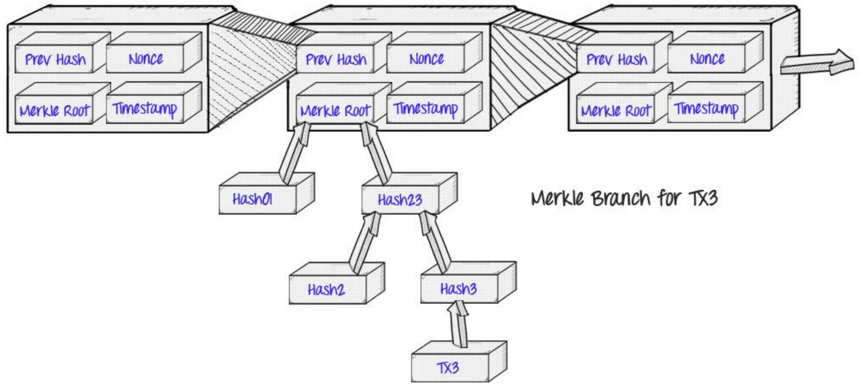
Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper
- “name” of block := hash of its header
- typical ingredients of block header:
  - predecessor (specified by name, as above)
  - transaction root (root of Merkle tree of txs)



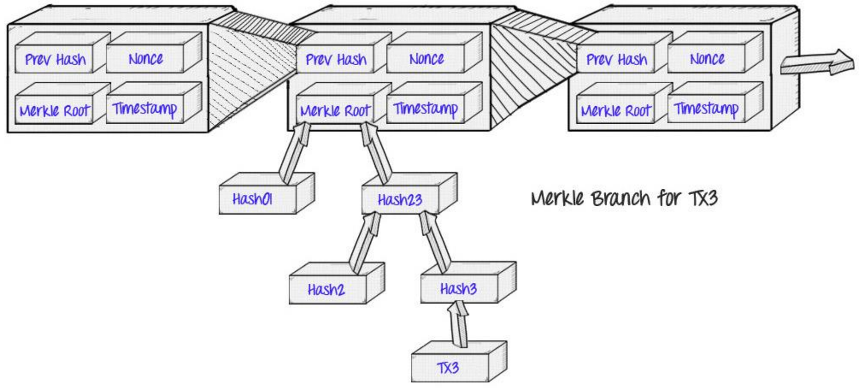
Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper
- “name” of block := hash of its header
- typical ingredients of block header:
  - predecessor (specified by name, as above)
  - transaction root (root of Merkle tree of txs)
  - state root (in account-based blockchains)



Block Headers

- block = header (metadata) + transactions (payload)
  - optional: additional data (cf., witness data in Bitcoin, blobs in Etherem)
  - additional data may be non-archival, cheaper
- “name” of block := hash of its header
- typical ingredients of block header:
  - predecessor (specified by name, as above)
  - transaction root (root of Merkle tree of txs)
  - state root (in account-based blockchains)
  - consensus-related data (e.g., signature of proposer, view #, etc.)
    - details vary for permissioned vs. proof-of-work vs. proof-of-state



Light Clients

Note: block header size << block size (by factor of 100-10000).

Light Clients

Note: block header size << block size (by factor of 100-10000).

Light client: downloads block headers only, not full blocks.

Light Clients

Note: block header size << block size (by factor of 100-10000).

Light client: downloads block headers only, not full blocks.

- example #1: app running on user's phone

Light Clients

Note: block header size << block size (by factor of 100-10000).

Light client: downloads block headers only, not full blocks.

- example #1: app running on user's phone
- example #2: smart contract on a blockchain ("bridge contract")

Light Clients

Note: block header size << block size (by factor of 100-10000).

Light client: downloads block headers only, not full blocks.

- example #1: app running on user's phone
- example #2: smart contract on a blockchain ("bridge contract")
- typically associated with one or small number of public keys/accounts
- generally no PKI, not participating in or following consensus
    - just subscribing to a "block header service"

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

- examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

- examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance?

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion?

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

- examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers?

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]
4. check if corresponding blocks have been finalized?

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]
4. check if corresponding blocks have been finalized? [no]

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

- examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]
4. check if corresponding blocks have been finalized? [no]
5. check validity of predecessor pointers, back to genesis?

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

- examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]
4. check if corresponding blocks have been finalized? [no]
5. check validity of predecessor pointers, back to genesis? [yes!]

Light Clients (con’d)

Light client: downloads block headers only, not full blocks.

– examples: app on user’s phone, smart contract on a blockchain

Question: what can light clients do (without help):

1. check account balance? [answer: no]
2. check tx inclusion? [answer: no]
3. check correctness of tx/state roots in block headers? [no]
4. check if corresponding blocks have been finalized? [no]
5. check validity of predecessor pointers, back to genesis? [yes!]

Goal: enable 1+2 through short + verifiable (Merkle) proofs.

The Cast of Characters

The Cast of Characters

1. Validators: participate in consensus and execution, maintain full state. [increasingly: outsource block-building from other duties]
   - source of “decentralization,” need (super-)majority to be honest/correct

The Cast of Characters

1. Validators: participate in consensus and execution, maintain full state.
   [increasingly: outsource block-building from other duties]
   – source of "decentralization," need (super-)majority to be honest/correct

2. Full node: maintain full state, don't participate in consensus.
   – can serve RPC requests and/or act as watchdog on validators

The Cast of Characters

1. Validators: participate in consensus and execution, maintain full state.
   [increasingly: outsource block-building from other duties]
   – source of "decentralization," need (super-)majority to be honest/correct

2. Full node: maintain full state, don't participate in consensus.
   – can serve RPC requests and/or act as watchdog on validators

3. Archival node: store all historical data.
   – really one need one honest such node

The Cast of Characters

1. Validators: participate in consensus and execution, maintain full state.
   [increasingly: outsource block-building from other duties]
   – source of "decentralization," need (super-)majority to be honest/correct

2. Full node: maintain full state, don't participate in consensus.
   – can serve RPC requests and/or act as watchdog on validators

3. Archival node: store all historical data.
   – really one need one honest such node

4. Light client: store only block headers, trust that state roots are correct.

The Cast of Characters

1. Validators: participate in consensus and execution, maintain full state.
   [increasingly: outsource block-building from other duties]
   – source of "decentralization," need (super-)majority to be honest/correct
2. Full node: maintain full state, don't participate in consensus.
   – can serve RPC requests and/or act as watchdog on validators
3. Archival node: store all historical data.
   – really one need one honest such node
4. Light client: store only block headers, trust that state roots are correct.
5. Stateless client: verify correctness of state roots in block headers.
   – future: validators might just be stateless clients (with block-building outsourced)

Light Clients in the UTXO Model

Simplified payment verification (SPV) in Bitcoin: a light client should be able to verify payments to/from it. (➔ track balance)

Light Clients in the UTXO Model

Simplified payment verification (SPV) in Bitcoin: a light client should be able to verify payments to/from it. (→ track balance)

Solution:

- keep track of all block headers (with tx roots)
- verify all predecessors

Light Clients in the UTXO Model

Simplified payment verification (SPV) in Bitcoin: a light client should be able to verify payments to/from it. (➔ track balance)

Solution:

- keep track of all block headers (with tx roots)
- verify all predecessors
- for inbound payments, insist on receipt (Merkle proof of tx inclusion)
- for outbound payments, request Merkle proof from validator

Light Clients in the UTXO Model

Simplified payment verification (SPV) in Bitcoin: a light client should be able to verify payments to/from it. (➔ track balance)

Solution:

- keep track of all block headers (with tx roots)
- verify all predecessors
- for inbound payments, insist on receipt (Merkle proof of tx inclusion)
- for outbound payments, request Merkle proof from validator

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Solution: lean on consensus + honest (super)majority assumption.

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Solution: lean on consensus + honest (super)majority assumption.

- if using Tendermint or similar: (< n/3 Byzantine validators)

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Solution: lean on consensus + honest (super)majority assumption.

- if using Tendermint or similar: (< n/3 Byzantine validators)
  - have validators broadcast “finalized h(B)” msg after finalizing block B
  - only accept a block header accompanied by > n/3 such messages

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Solution: lean on consensus + honest (super)majority assumption.

- if using Tendermint or similar: ($< n/3$ Byzantine validators)
  - have validators broadcast “finalized h(B)” msg after finalizing block B
  - only accept a block header accompanied by $> n/3$ such messages
- if using longest-chain consensus: ($< n/2$ Byzantine validators)

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

Solution: lean on consensus + honest (super)majority assumption.

- if using Tendermint or similar: (< n/3 Byzantine validators)
  - have validators broadcast “finalized h(B)” msg after finalizing block B
  - only accept a block header accompanied by > n/3 such messages
- if using longest-chain consensus: (< n/2 Byzantine validators)
  - verify proposer signatures on all block headers (from leader of view)

Are the Block Headers Correct?

Issue: how to know that block header corresponds to real + finalized block? [worry: tx root corresponds to bogus txs]

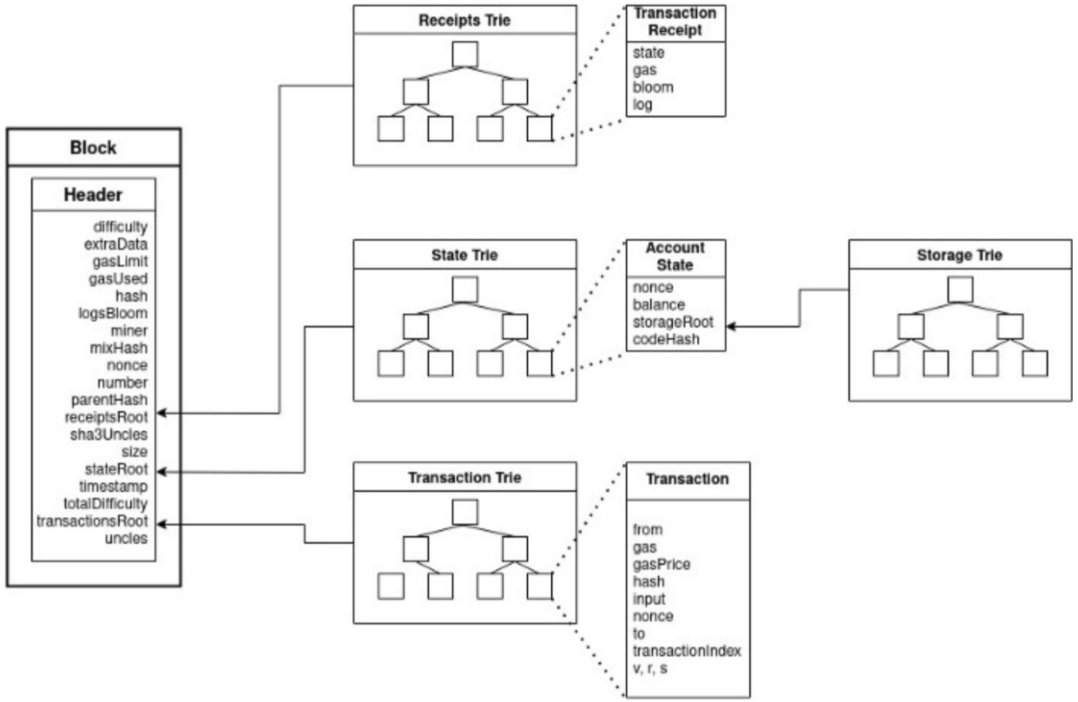
Solution: lean on consensus + honest (super)majority assumption.

- if using Tendermint or similar: (< n/3 Byzantine validators)
  - have validators broadcast “finalized h(B)” msg after finalizing block B
  - only accept a block header accompanied by > n/3 such messages
- if using longest-chain consensus: (< n/2 Byzantine validators)
  - verify proposer signatures on all block headers (from leader of view)
  - trust block header only if ≥ k blocks deep on the longest chain
    - k = a user-specified security parameter

Light Clients in the Accounts-Based Model

Example: Ethereum.

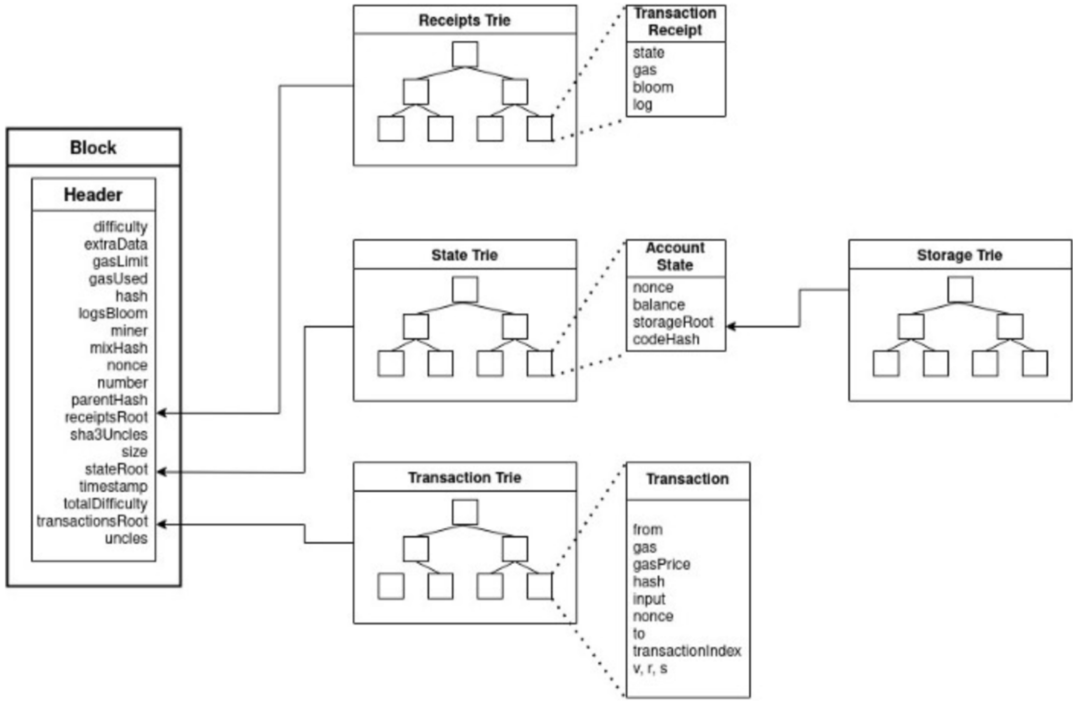
- state = accounts, each account has balance/code/data



Light Clients in the Accounts-Based Model

Example: Ethereum.

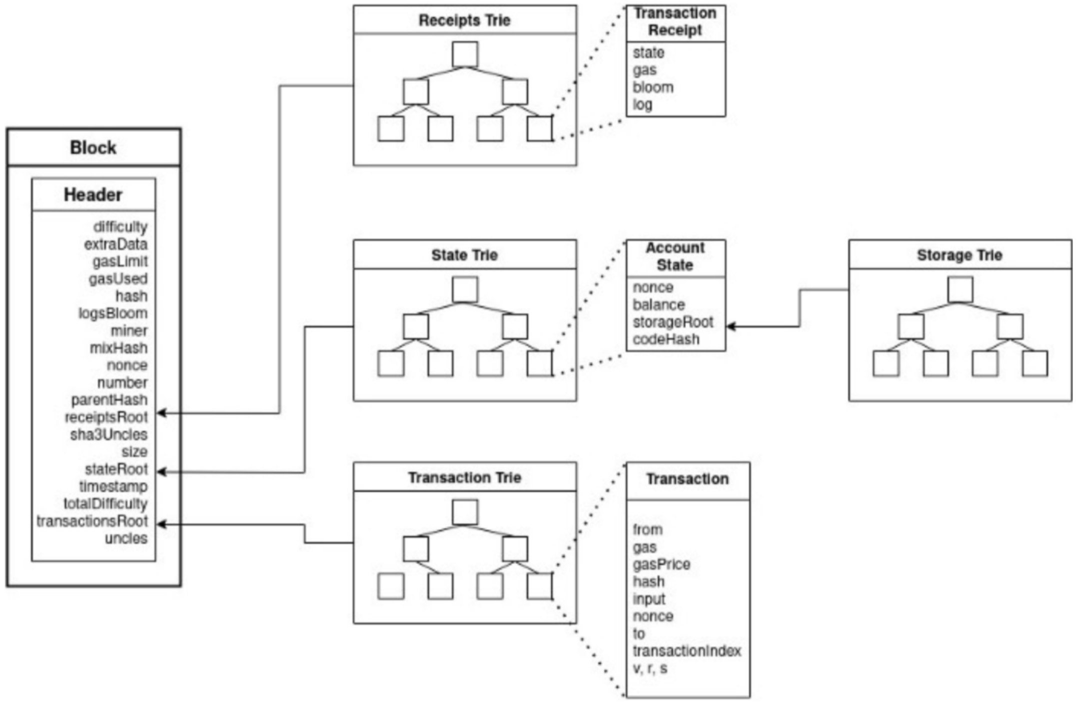
- state = accounts, each account has balance/code/data
- stored in Merkle-Patricia tree
  - leaves = accounts



Light Clients in the Accounts-Based Model

Example: Ethereum.

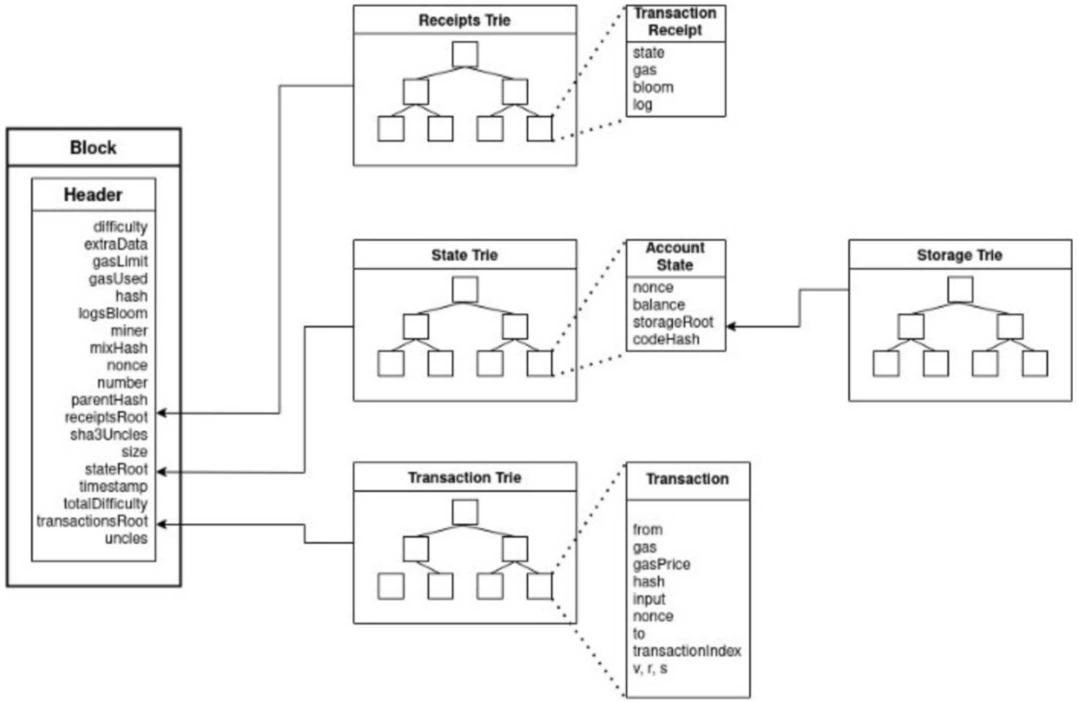
- state = accounts, each account has balance/code/data
- stored in Merkle-Patricia tree
  - leaves = accounts
- block header includes state root (in addition to tx root, receipts root)



Light Clients in the Accounts-Based Model

Example: Ethereum.

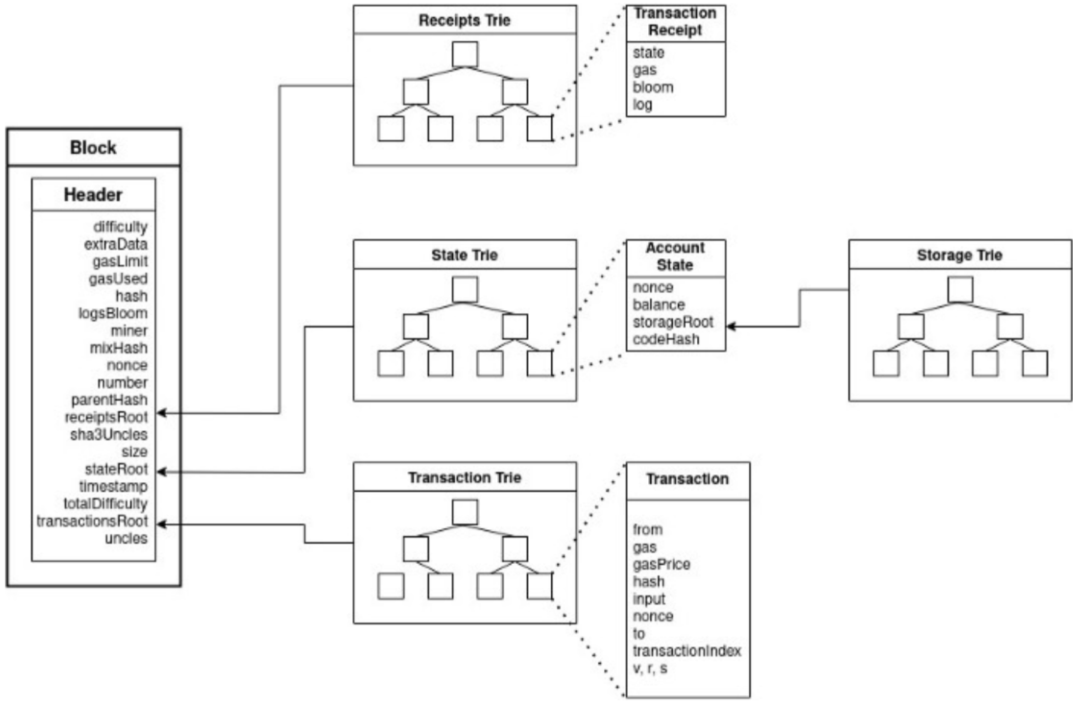
- state = accounts, each account has balance/code/data
- stored in Merkle-Patricia tree
  – leaves = accounts
- block header includes state root (in addition to tx root, receipts root)
- to check tx inclusion: same as UTXO case (use tx root)



Light Clients in the Accounts-Based Model

Example: Ethereum.

- state = accounts, each account has balance/code/data
- stored in Merkle-Patricia tree
  - leaves = accounts
- block header includes state root (in addition to tx root, receipts root)



- to check tx inclusion: same as UTXO case (use tx root)
- to check account balance: Merkle proof (but now use state root)
  - assume state root is correct (otherwise rejected by honest validators)

Stateless Clients/Validation

Goal: verify correctness of state root without knowing full state.

Stateless Clients/Validation

Goal: verify correctness of state root without knowing full state.

Inputs to verification problem:

- initial state root (assume already verified)
- list of txs
- alleged new state root r' (after execution of all the txs)

Stateless Clients/Validation

Goal: verify correctness of state root without knowing full state.

Inputs to verification problem:

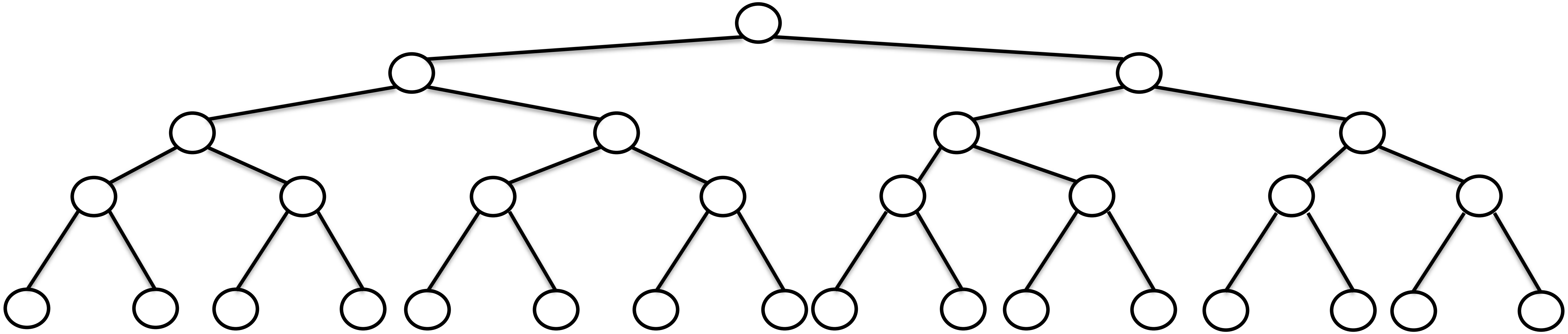
- initial state root (assume already verified)
- list of txs
- alleged new state root r’ (after execution of all the txs)

Question: how much of the actual state is necessary to verify the correctness of r’?

Example: Simple Transfers

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

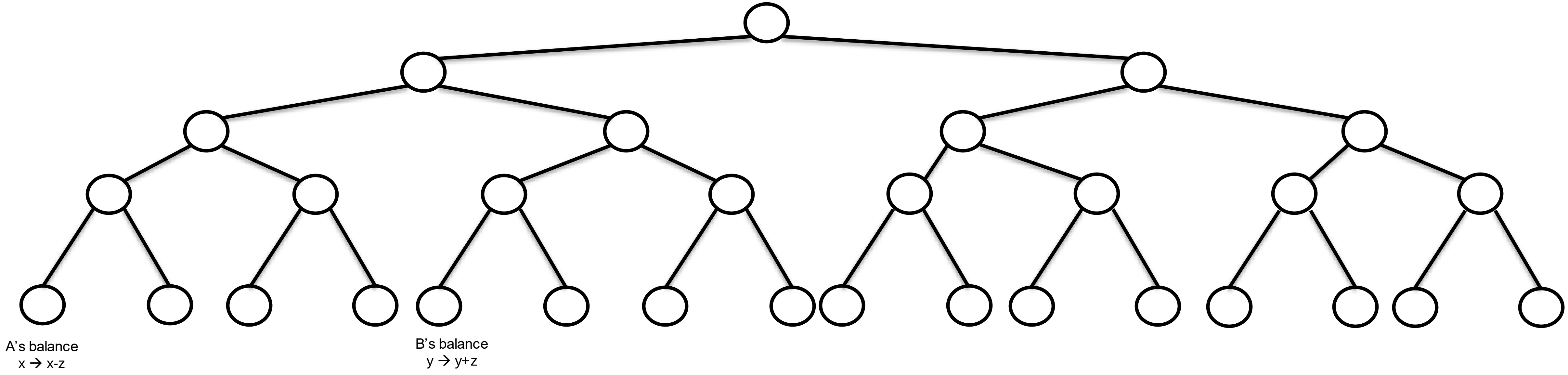
Question: how much of the actual state is necessary to verify the correctness of r'?



Example: Simple Transfers

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

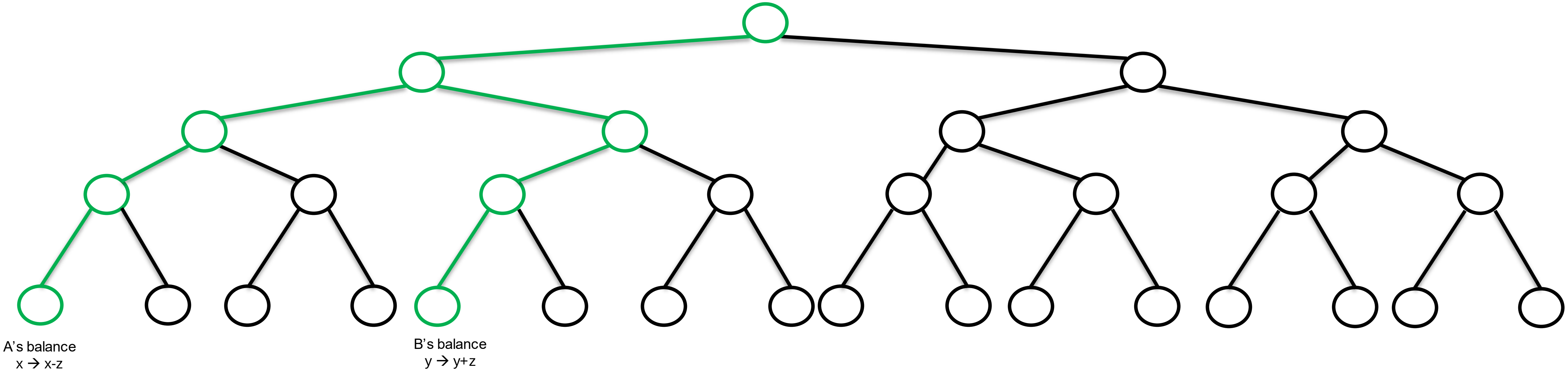
Question: how much of the actual state is necessary to verify the correctness of r'?



Example: Simple Transfers

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

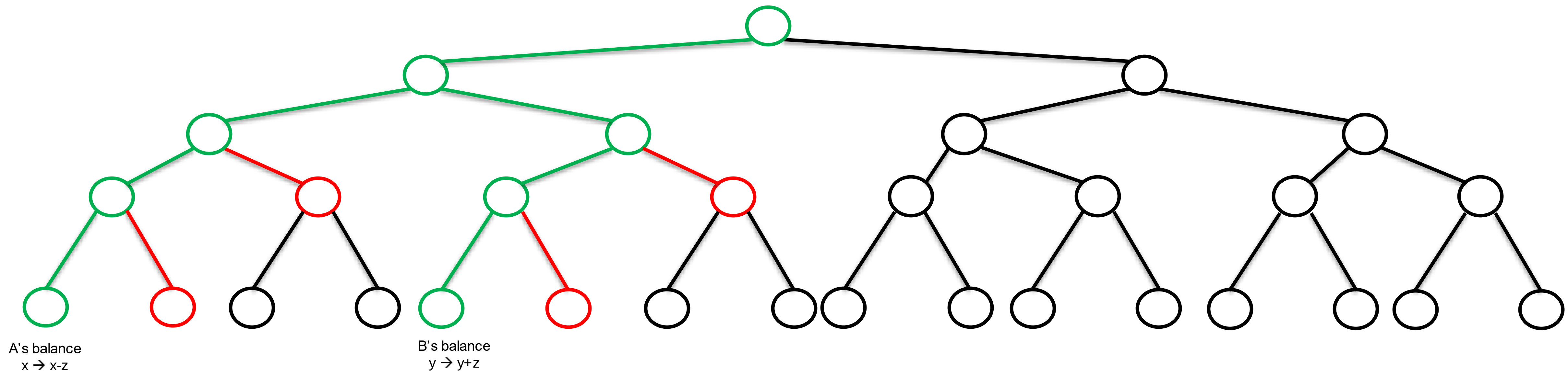
Question: how much of the actual state is necessary to verify the correctness of r'?



Example: Simple Transfers

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

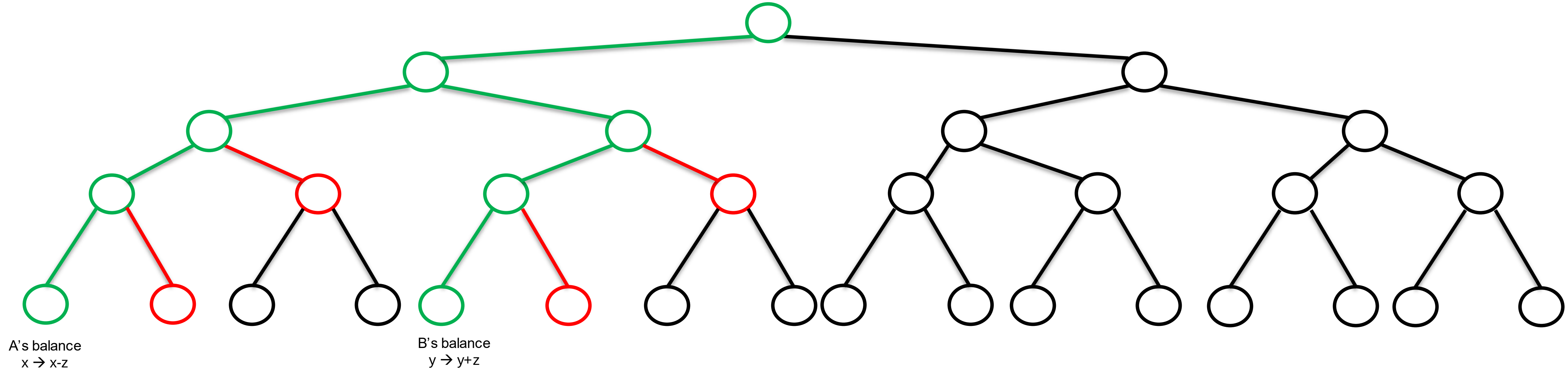
Question: how much of the actual state is necessary to verify the correctness of r'?



Example: Simple Transfers

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

Question: how much of the actual state is necessary to verify the correctness of r'?



- sufficient to supply Merkle proofs for balances of A and B
  - stateless client then has enough info to compute new Merkle root

Verifying General Transactions

Inputs to verification problem: initial state root, list of txs, alleged new state root r’.

Question: how much of the actual state is necessary to verify the correctness of r’?

Verifying General Transactions

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

Question: how much of the actual state is necessary to verify the correctness of r'?

- need Merkle proof for each part of blockchain state accessed by some tx (e.g., supplied by a validator or full node)
  - Merkle proofs supply state info on need-to-know basis

Verifying General Transactions

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

Question: how much of the actual state is necessary to verify the correctness of r'?

- need Merkle proof for each part of blockchain state accessed by some tx (e.g., supplied by a validator or full node)
  – Merkle proofs supply state info on need-to-know basis
- after each update to state, recompute new state root
  – increment nonce, write new value to variable in contract storage, etc.

Verifying General Transactions

Inputs to verification problem: initial state root, list of txs, alleged new state root r'.

Question: how much of the actual state is necessary to verify the correctness of r'?

- need Merkle proof for each part of blockchain state accessed by some tx (e.g., supplied by a validator or full node)
  - Merkle proofs supply state info on need-to-know basis
- after each update to state, recompute new state root
  - increment nonce, write new value to variable in contract storage, etc.
- after processing all txs, can check if new state root = r'

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

– in Bitcoin, need to know current UTXOs to assess block validity

– in Ethereum, need to know state to know outcome of computations

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity
- one possible future: Verkle trees (using KZG commitments)

“Statelessness”

Note: to determine if block is valid, generally need to keep track of transactions processed in previous blocks.

- in Bitcoin, need to know current UTXOs to assess block validity
- in Ethereum, need to know state to know outcome of computations

Long-time Ethereum dream: include enough metadata in block to assess validity purely from block itself (no other state needed).

- in principle, could include the entire state in the block
- slightly less crazy: include all Merkle proofs needed to assess validity
- one possible future: Verkle trees (using KZG commitments)
- another: each block includes SNARK proving its correctness