Scaling the Blockchain

Ronghui Gu

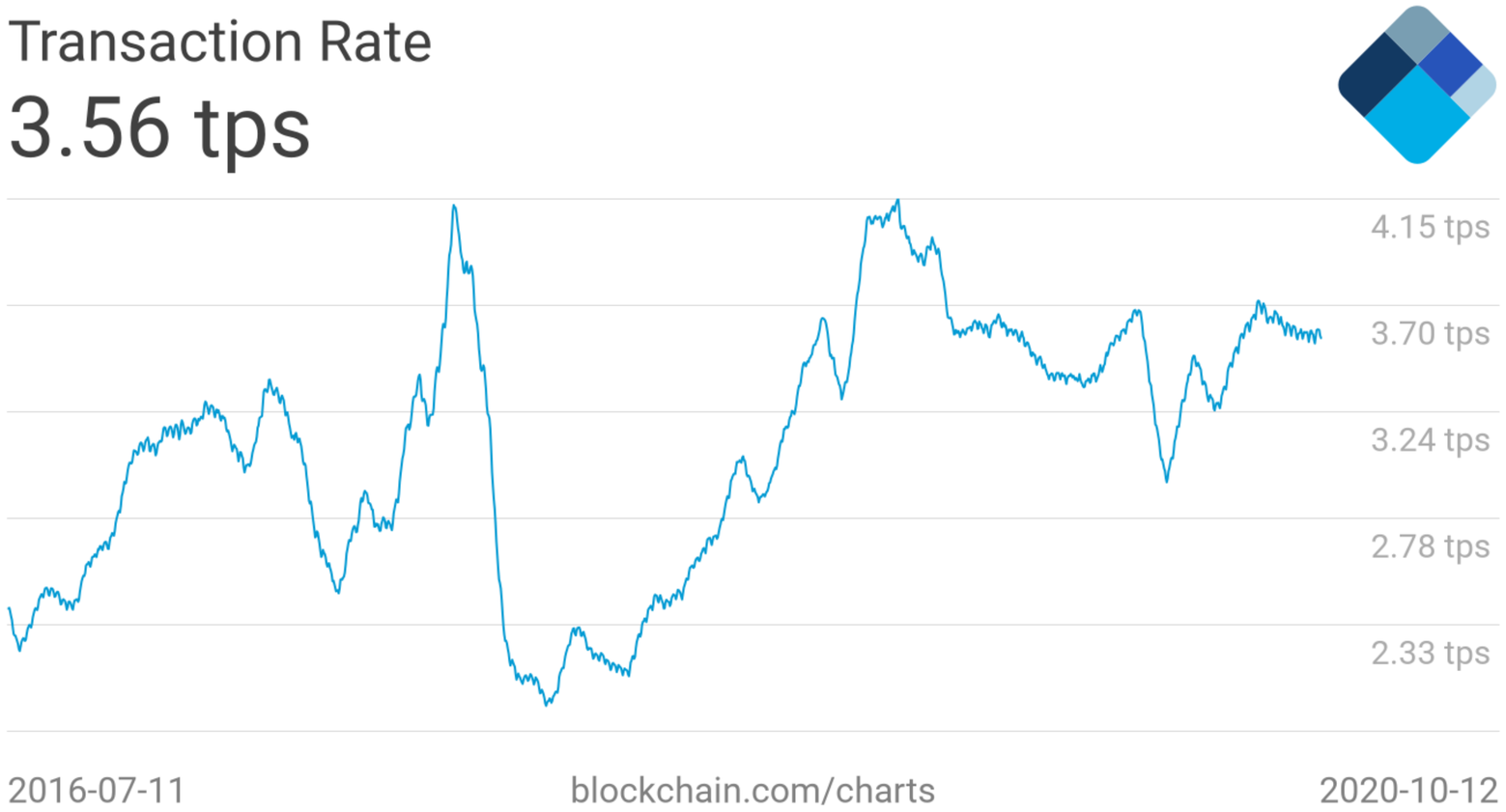
Columbia University

Course website: https://verigu.github.io/6998Fall2024/

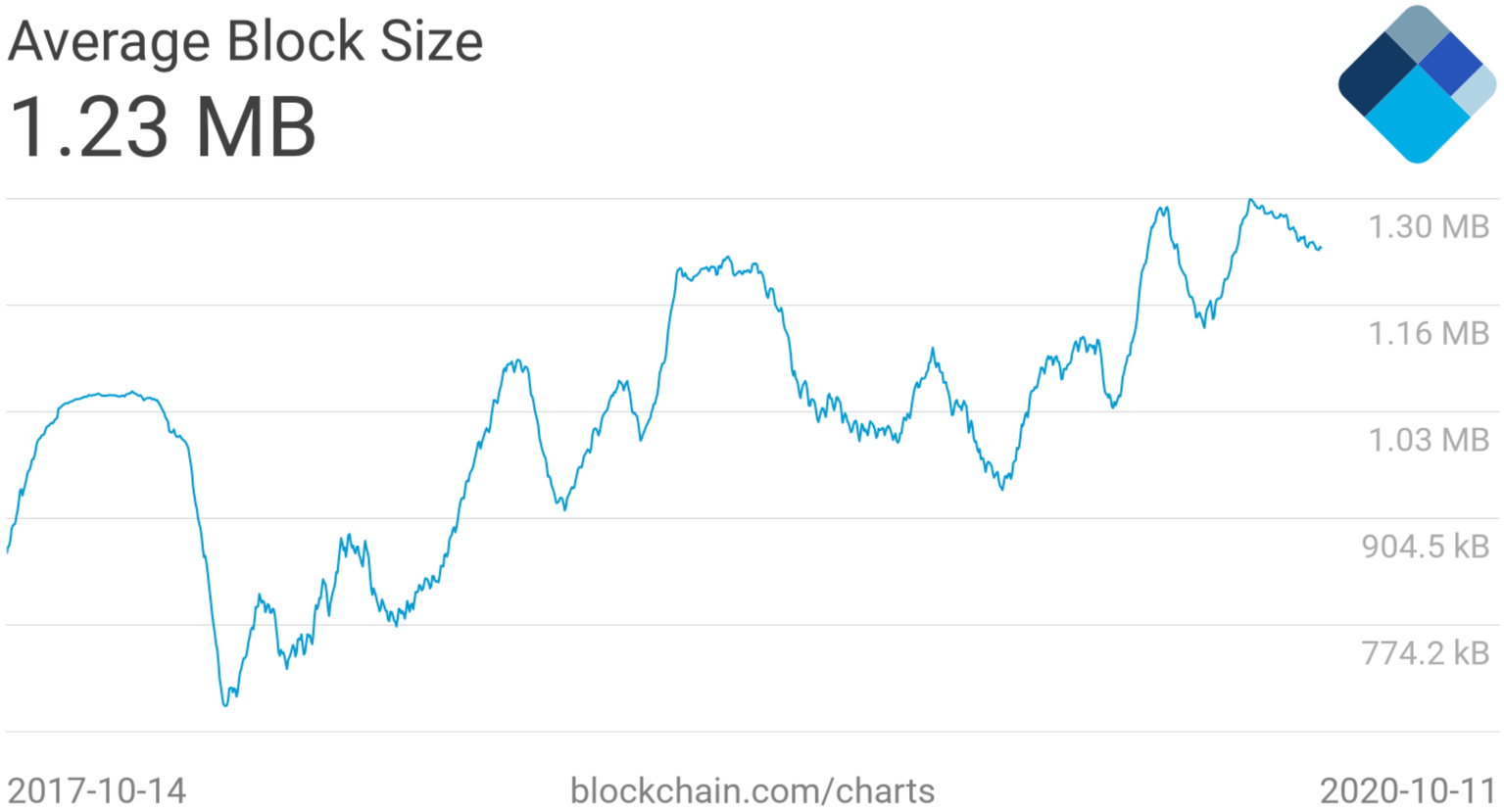
Problem: low throughput of blockchain

- Decentralization leads to lower throughput and higher latency than centralized solutions
  - Consensus protocols require multiple nodes to exchange messages
  - PoW requires time
  - May need to wait for confirmation

Bitcoin throughput



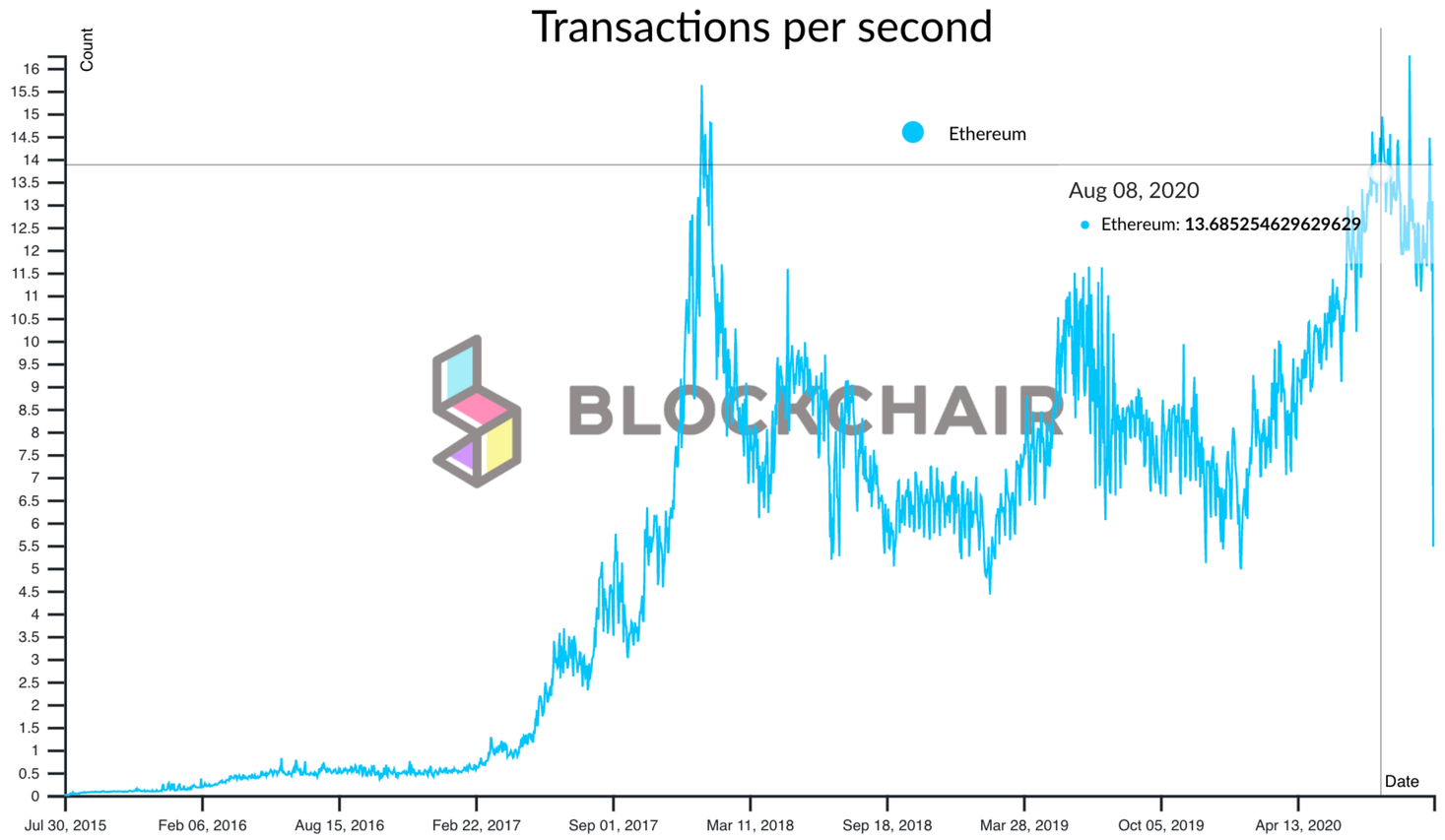
Bitcoin throughput limited by block size



1 MB/block
~250 B/tx
➔ 4000 tx/block

10 min/block
➔ Max: 6.7 tx/s

Ethereum throughput limited by gas



~21K gas/tx
12.5M gas/block
➔ 600 tx/block

15 s/block
➔ Max 40tx/s

Credit card tx throughput

Example: Visa ~2000tx/s, max 65000 tx/s

(Christmas shopping season)

Raising block size or gas limit

Throughput directly depends on block size or gas limit.

Why not simply raise them?

Network delay and consensus security depends on them

Additional issue: Latency (delay till tx confirmation)

Idea: record only settlement on blockchain

- Save fewer tx on chain if everything goes well ➔ higher throughput, lower tx fee
- Use blockchain to resolve any dispute

Ways to scale blockchain

- **Payment channel or state channel**
  - Peer-to-peer channel for payment or contract tx
  - Settlement = net transfers or final state changes
- **Rollups**
  - Rollup server aggregates tx list
  - Settlement = commitment of tx list

1 Payment Channels And State Channels

Motivating application: micropayments

Example: Alice hires Bob for 100 min service at 0.01 BTC/min

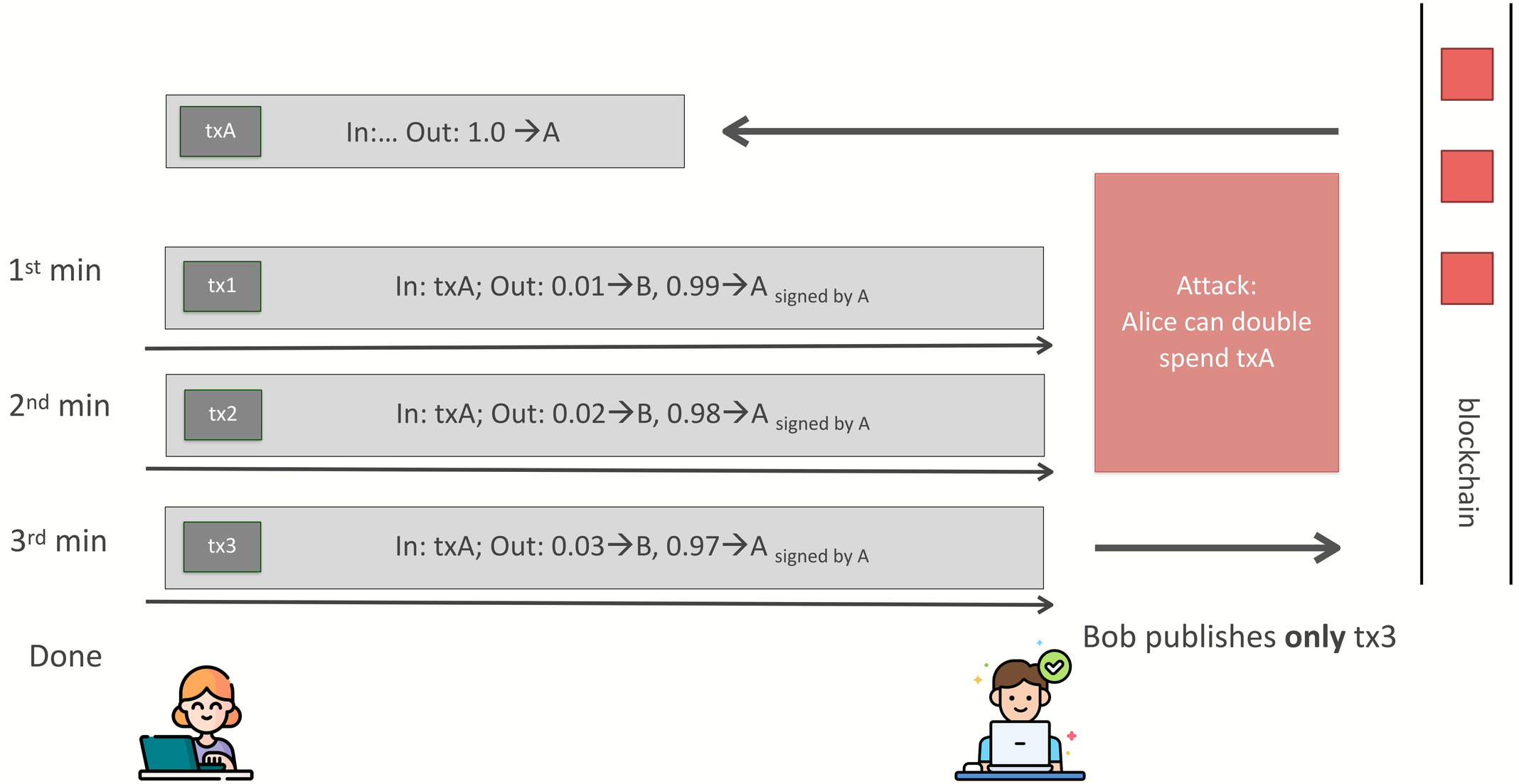
- Upfront payment? Bob may not provide full service
- Pay after service? Alice may not pay

A solution

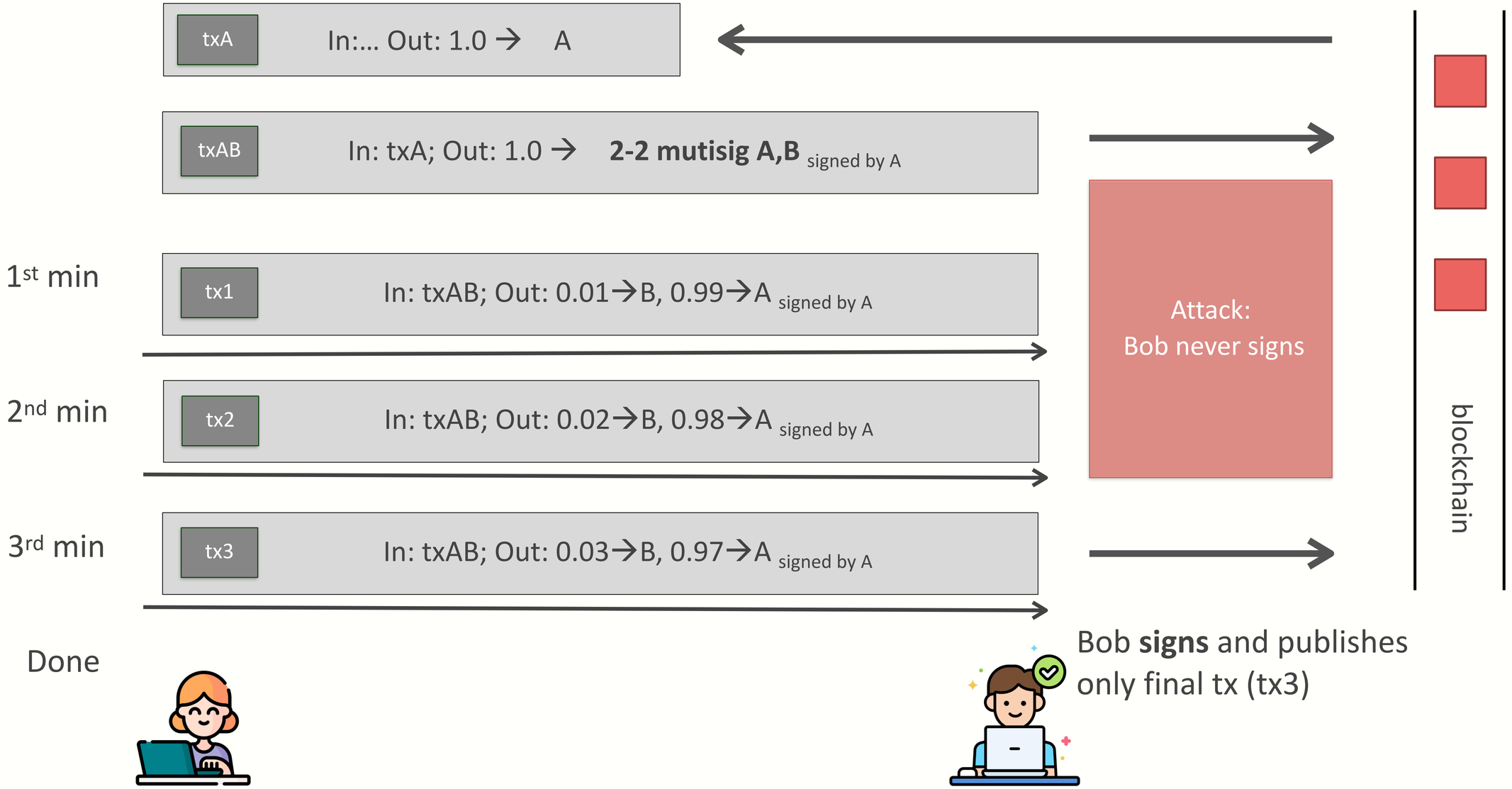
- Alice pays 0.01 BTC after every min of Bob service

Works only if tx fee is low (<< 0.01 BTC)!

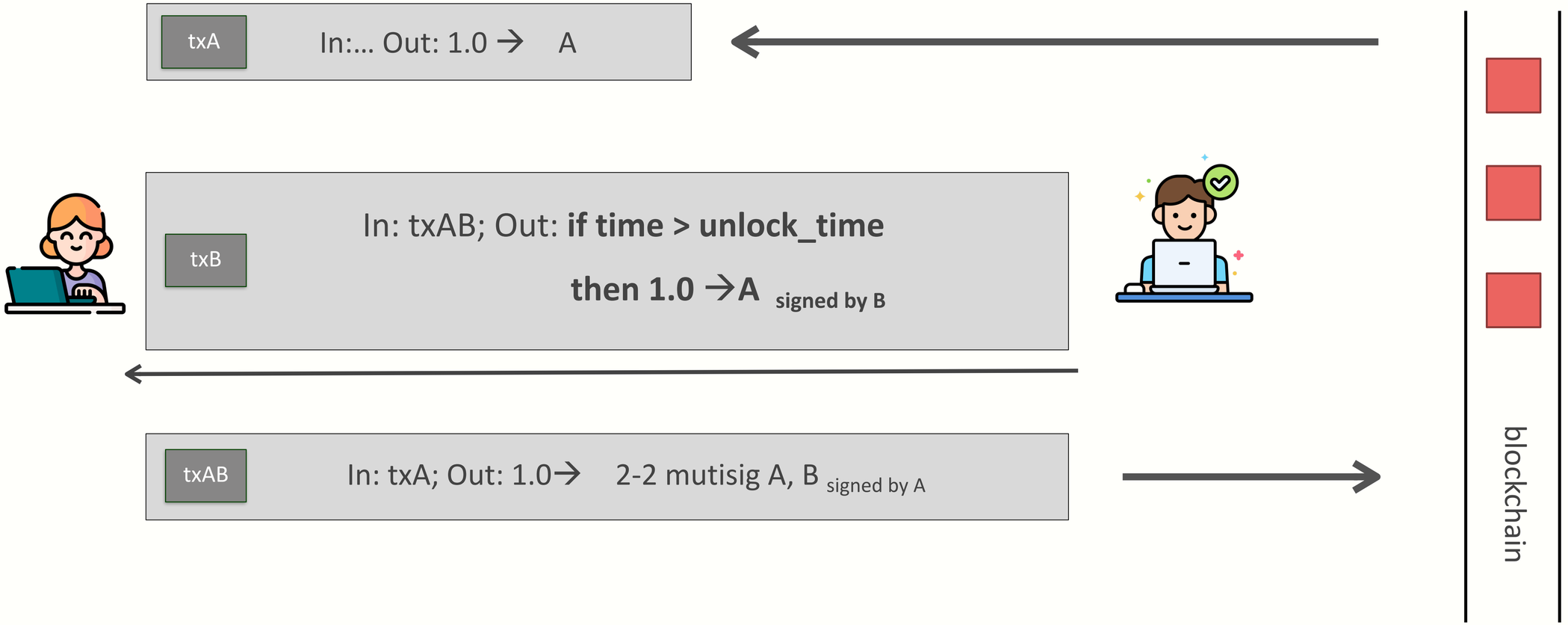
Unidirectional payment channel in bitcoin (broken)



Fixing double spend



Fixing locked fund using timeout



Bob sends Alice a refund tx before Alice publishes the multisig tx

Alice publishes Bob’s refund tx if Bob never publishes final tx

Uni. payment channel in Ethereum

Implemented as smart contract

close(): recipient calls to close out the channel

claimTimeout(): sender calls to reclaim remaining fund

Easier than Bitcoin because accounts have states

Example implementation in Solidity

Home UniChannel.sol

```
pragma solidity >=0.4.24 <0.6.0;

contract SimplePaymentChannel {
    address payable public sender;      // The account sending payments.
    address payable public recipient;   // The account receiving the payments.
    uint256 public expiration;  // Timeout in case the recipient never closes.

    constructor (address payable _recipient, uint256 duration)
        public
        payable
    {
        sender = msg.sender;
        recipient = _recipient;
        expiration = now + duration;
    }


    /// the recipient can close the channel at any time by presenting a
    /// signed amount from the sender. the recipient will be sent that amount,
    /// and the remainder will go back to the sender
    function close(uint256 amount, bytes memory signature) public {
        require(msg.sender == recipient);
        require(isValidSignature(amount, signature));

        recipient.transfer(amount);
        selfdestruct(sender);
    }

    /// if the timeout is reached without the recipient closing the channel,
    /// then the Ether is released back to the sender.
    function claimTimeout() public {
        require(now >= expiration);
        selfdestruct(sender);
    }
}
```

Bidirectional payment channel

Alice and Bob want to move funds back and forth

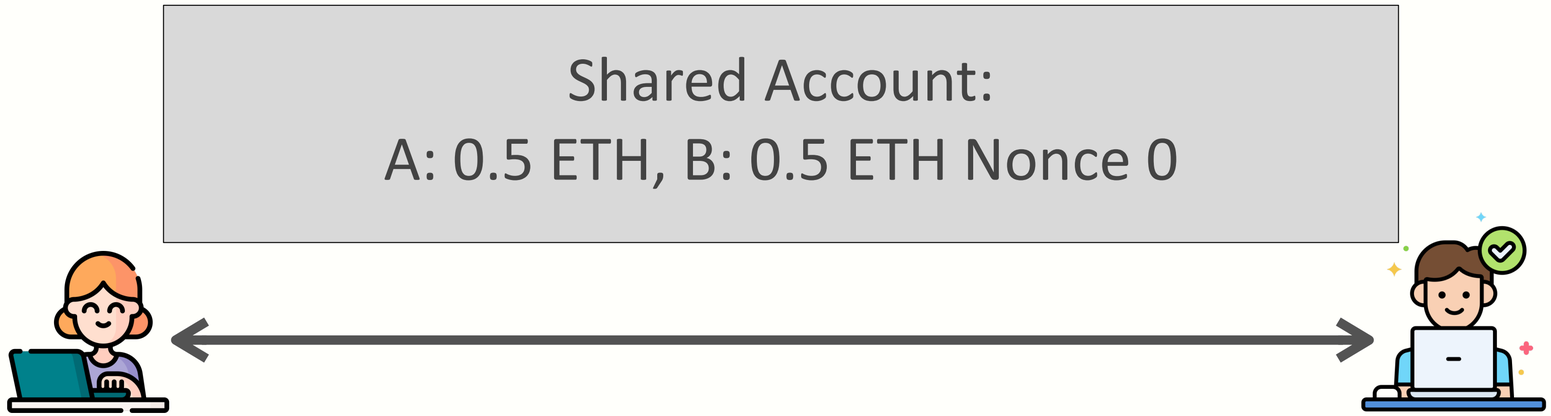
Implement using two unidirectional channels?

Bidirectional payment channel contract

- Contract state tracks balances of both users
- Users agree on new account balances off chain
- Both users sign the state update, and send to contract
- Contract verifies the signatures before updating state

- Security: use nonce to prevent premature channel closures

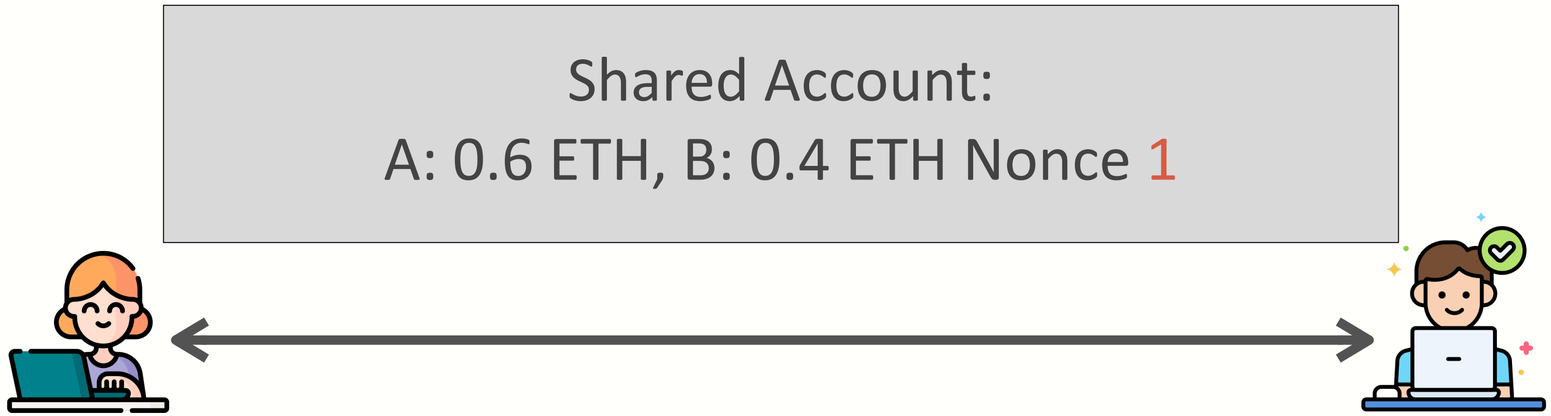
Bidirectional payment channel example



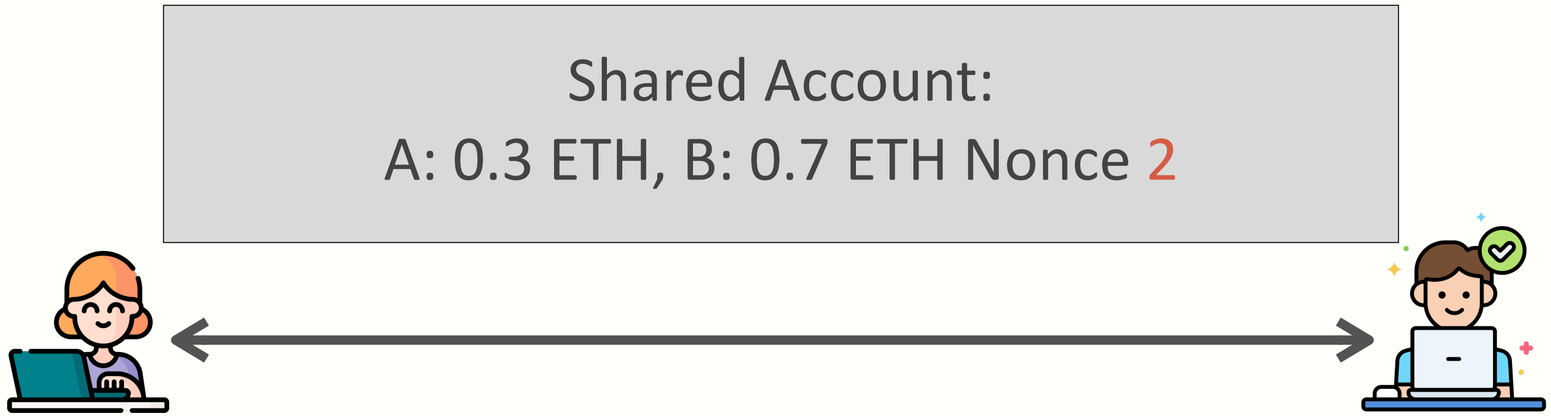
A: 0.6, Bob: 0.4 Nonce 1

Alice Bob

Bidirectional payment channel example



Closing bidirectional payment channel



Before funding Alice and Bob sign initial state

Alice submits balances and signatures to contract.

-> Starts challenge period

If Bob can submit tx with greater nonce: New state is valid

State channels

- Smart contracts support rich tx than just payments
- State channels generalize payment channels to arbitrary two-party smart contracts

Shared Contract:
State: board state Nonce i

Bitcoin bidirectional payment channels

Problem:

UTXOs have no global state -> Can’t store nonce

Solution:

When updating the channel to Alices benefit,

Alice gets TX that invalidates Bob’s old state

UTXO payment channel concepts

- **Relative time-lock**: output can be claimed t timesteps (i.e., blocks) from the time the TX is accepted to the blockchain
- **Hash lock**: Claiming output is pre-conditioned on providing the preimage of a cryptographic hash

Intuition: Both A and B hold TXs they can submit to settle the current split balance. Balance is updated by exchanging new TXs and “invalidating” old. Unilateral settlement is time-locked for one party, allows the other to challenge by providing hash-lock preimage. TXs invalidated by exchanging hash-lock preimages.

UTXO payment channel

2-of-2 Multisig Address C:

UTXO payment channel

2-of-2 Multisig Address C:

X=H(x)

Y=H(y)

Random x

Random y

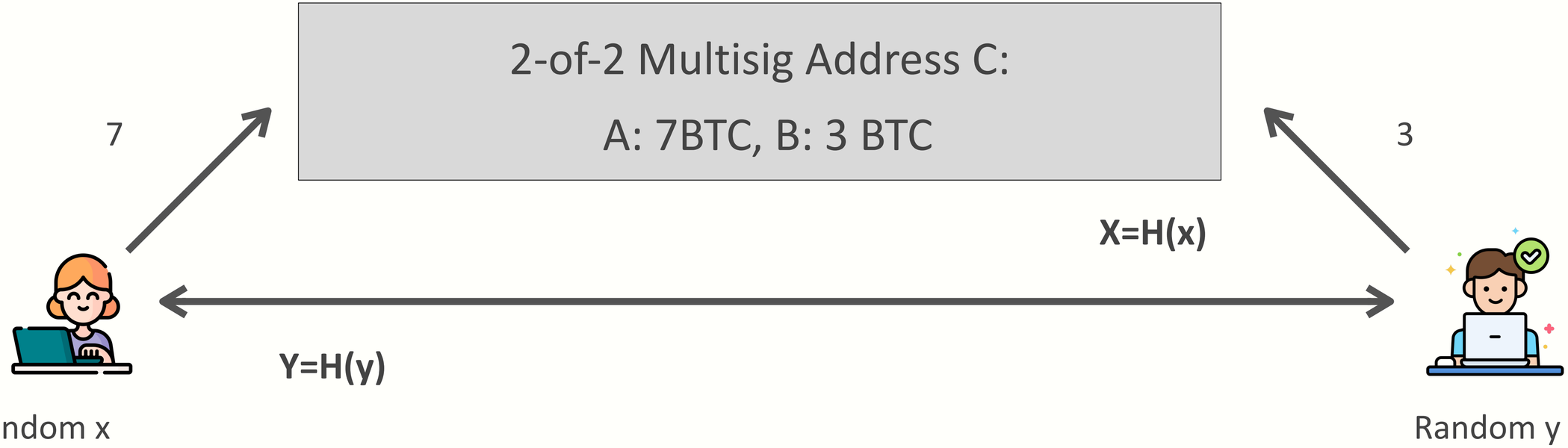
TX1 from C:
Out1: Pay 7 -> A
Out2: Either 3 -> B (7 Day timelock)
Or 3 -> A given y s.t. H(y)=Y

Alice

TX2 from C:
Pay 3 -> B
Either 7 -> A (7 Day timelock)
Or 7 -> B given x s.t. H(x)=X

Bob

UTXO payment channel



TX2 from C:
Pay 3 -> B
Either 7 -> A (7 Day timelock)
Or 7 -> B given x s.t. H(x)=X
Bob

TX1 from C:
Out1: Pay 7 -> A
Out2: Either 3 -> B (7 Day timelock)
Or 3 -> A given y s.t. H(y)=Y
Alice

UTXO payment channel update

2-of-2 Multisig Address C:
A: 6 BTC, B: 4 BTC

X

Random x’

X’=H(x’)

UTXO payment channel update

2-of-2 Multisig Address C:
A: 6 BTC, B: 4 BTC

X

Random x’

X’=H(x’)

TX3 from C:
Out1: Pay 6 -> A
Out2: Either 4 -> B (7 Day timelock)
Or 4 -> A given y s.t. H(y)=Y
Alice

TX4 from C:
Pay 4 -> B
Either 6 -> A (7 Day timelock)
Or 6 -> B given x’ s.t. H(x’)=X’
Bob

UTXO payment channel update

2-of-2 Multisig Address C:
A: 6 BTC, B: 4 BTC

X

Random x’

X’=H(x’)

TX4 from C:
Pay 4 -> B
Either 6 -> A (7 Day timelock)
Or 6 -> B given x’ s.t. H(x’)=X’

Bob

TX3 from C:
Out1: Pay 6 -> A
Out2: Either 4 -> B (7 Day timelock)
Or 4 -> A given y s.t. H(y)=Y

Alice

UTXO payment channel update

2-of-2 Multisig Address C:
A: 6 BTC, B: 4 BTC

X'=H(x') X

Random x'

TX4 from C:
Pay 4 -> B
Either 6 -> A (7 Day timelock)
Or 6 -> B given x s.t. H(x')=X'
Bob

TX3 from C:
Out1: Pay 6 -> A
Out2: Either 4 -> B (7 Day timelock)
Or 4 -> A given y s.t. H(y)=Y
Alice

Security

Alice has TX2,TX4

TX2 from C:
Pay 3 -> B
Either 7 -> A (7 Day timelock)
Or 7 -> B given x s.t. H(x)=X
Bob

TX4 from C:
Pay 4 -> B
Either 6 -> A (7 Day timelock)
Or 6 -> B given x' s.t. H(x')=X'
Bob

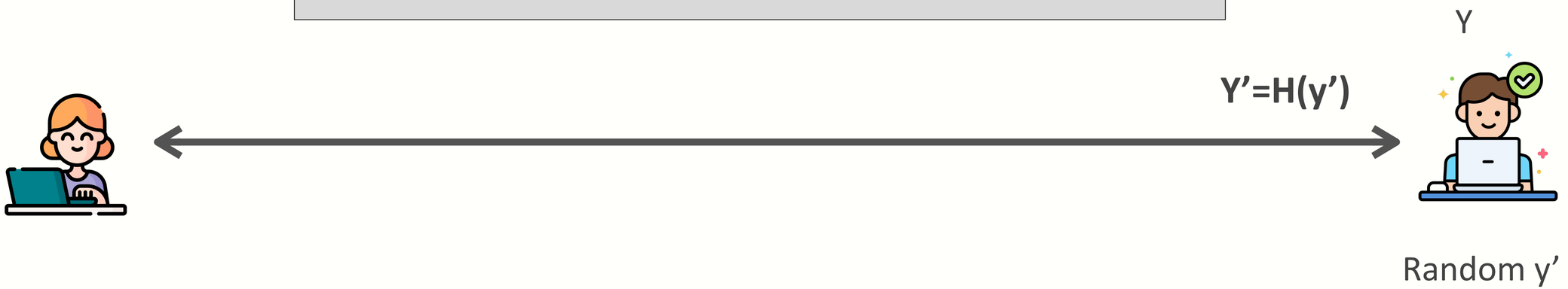
Bob has TX1,TX3, x

TX1 from C:
Pay 7 -> A
Either 3 -> B (7 Day timelock)
Or 3 -> A given y s.t. H(y)=Y
Alice

TX3 from C:
Pay 6 -> A
Either 4 -> B (7 Day timelock)
Or 4 -> A given y s.t. H(y)=Y
Alice

UTXO payment channel update

2-of-2 Multisig Address C:
A: 8 BTC, B: 2 BTC



UTXO payment channel update

2-of-2 Multisig Address C:
A: 8 BTC, B: 2 BTC

Y'=H(y')

Y

Random y'

TX5 from C:
Pay 8 -> A
Either 2 -> B (7 Day timelock)
Or 2 -> A given y s.t. H(y')=Y'
Alice

TX6 from C:
Pay 2 -> B
Either 8 -> A (7 Day timelock)
Or 8 -> B given x s.t. H(x')=X'
Bob

UTXO payment channel update

2-of-2 Multisig Address C:
A: 8 BTC, B: 2 BTC

Y'=H(y')

Y

Random y'

TX6 from C:
Pay 2 -> B
Either 8 -> A (7 Day timelock)
Or 8 -> B given x s.t. H(x')=X'
Bob

TX5 from C:
Pay 8 -> A
Either 2 -> B (7 Day timelock)
Or 2 -> A given y s.t. H(y')=Y'
Alice

UTXO payment channel update

2-of-2 Multisig Address C:
A: 8 BTC, B: 2 BTC

Y'=H(y')

Y

Random y'

TX6 from C:
Pay 2 -> B
Either 8 -> A (7 Day timelock)
Or 8 -> B given x s.t. H(x')=X'

Bob

TX5 from C:
Pay 8 -> A
Either 2 -> B (7 Day timelock)
Or 2 -> A given y s.t. H(y')=Y'

Alice

Security

Alice has TX2,TX6, y

Bob has TX3,TX5, x

TX2 from C:
Pay 3 -> B
Either 7 -> A (7 Day timelock)
Or 7 -> B given x s.t. H(x)=X
Bob

TX3 from C:
Pay 6 -> A
Either 4 -> B (7 Day timelock)
Or 4 -> A given y s.t. H(y)=Y
Alice

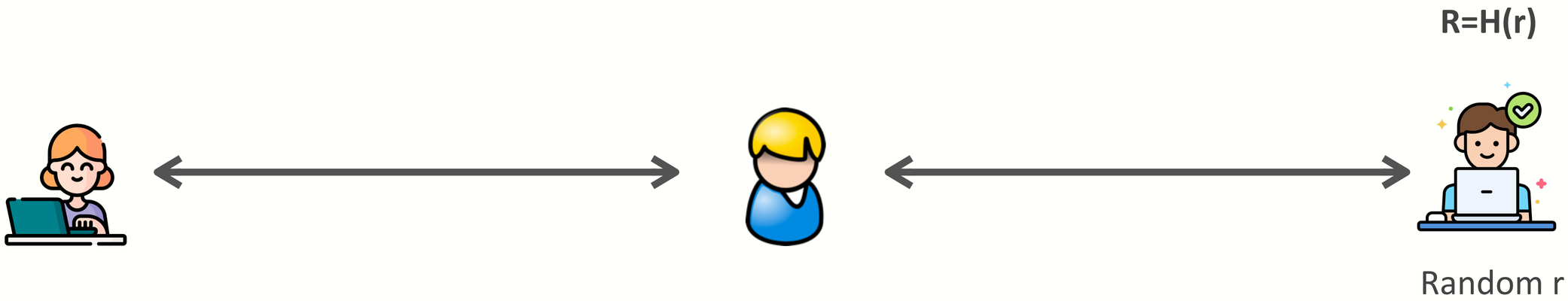
TX6 from C:
Pay 2 -> B
Either 8 -> A (7 Day timelock)
Or 8 -> B given x s.t. H(x')=X'
Bob

TX5 from C:
Pay 8 -> A
Either 2 -> B (7 Day timelock)
Or 2 -> A given y s.t. H(y')=Y'
Alice

Multi-hop payments

Pay through untrusted intermediary

Lightning network



Lightning network

R=H(r)

Random r

Pay 1.01 BTC to B
Hashlocked with R
Timelock to refund

Pay 1 BTC to C
Hashlocked with R
Timelock to refund

B claims 1.01 BTC with r

C claims 1 BTC with r

Watchtowers

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

Watchtowers outsource this task

User gives latest state to watchtower.

2 Rollups

Rollups

A rollup is a Layer 2 scaling solution for blockchains, designed to improve throughput and reduce transaction costs. Rollups work by **processing most transactions off-chain** (on Layer 2) while still leveraging the Layer 1 blockchain (like Ethereum) for security and data availability.

Rollups

1. Batching Transactions:

- Transactions are processed in bulk on the rollup layer (Layer 2).
- These transactions are compressed and aggregated into a single "batch" to be submitted to the main chain.

Rollups

2. Data Submission:

- Essential data or proofs about the transactions are posted on Layer 1 to ensure they can be verified independently.

Rollups

3. Validation:

- The rollup mechanism uses cryptographic proofs and economic incentives to validate transactions and prevent fraud.

Types of Rollups

Rollups are classified based on how they verify transactions:

1. Optimistic Rollups

2. ZK Rollups

Optimistic Rollups

- **How It Works**: Transactions are assumed to be valid by default. Disputes are resolved using fraud proofs.
- **Fraud Proofs**: If an invalid transaction is suspected, anyone can challenge it by submitting a fraud proof to Layer 1.
- **Advantages**: Lower computational costs for verifying transactions. Supports complex smart contracts.
- **Disadvantages**: Requires a dispute period (typically several days), delaying withdrawals.
- **Examples**: Arbitrum, Optimism.

ZK Rollups

- **How It Works**: Every batch of transactions generates a cryptographic proof (e.g., ZK-SNARK or ZK-STARK) that is submitted to Layer 1. The proof instantly verifies the validity of the transactions.
- **Validation Proofs**: These proofs ensure transactions are correct without revealing sensitive details.
- **Advantages**: Immediate finality (no dispute period). Smaller data footprint on Layer 1, improving efficiency.
- **Disadvantages**: Generating zero-knowledge proofs is computationally intensive.
- **Examples**: zkSync, Polygon zkEVM, ZKWasm.