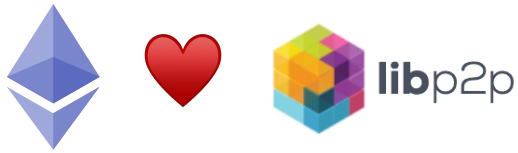
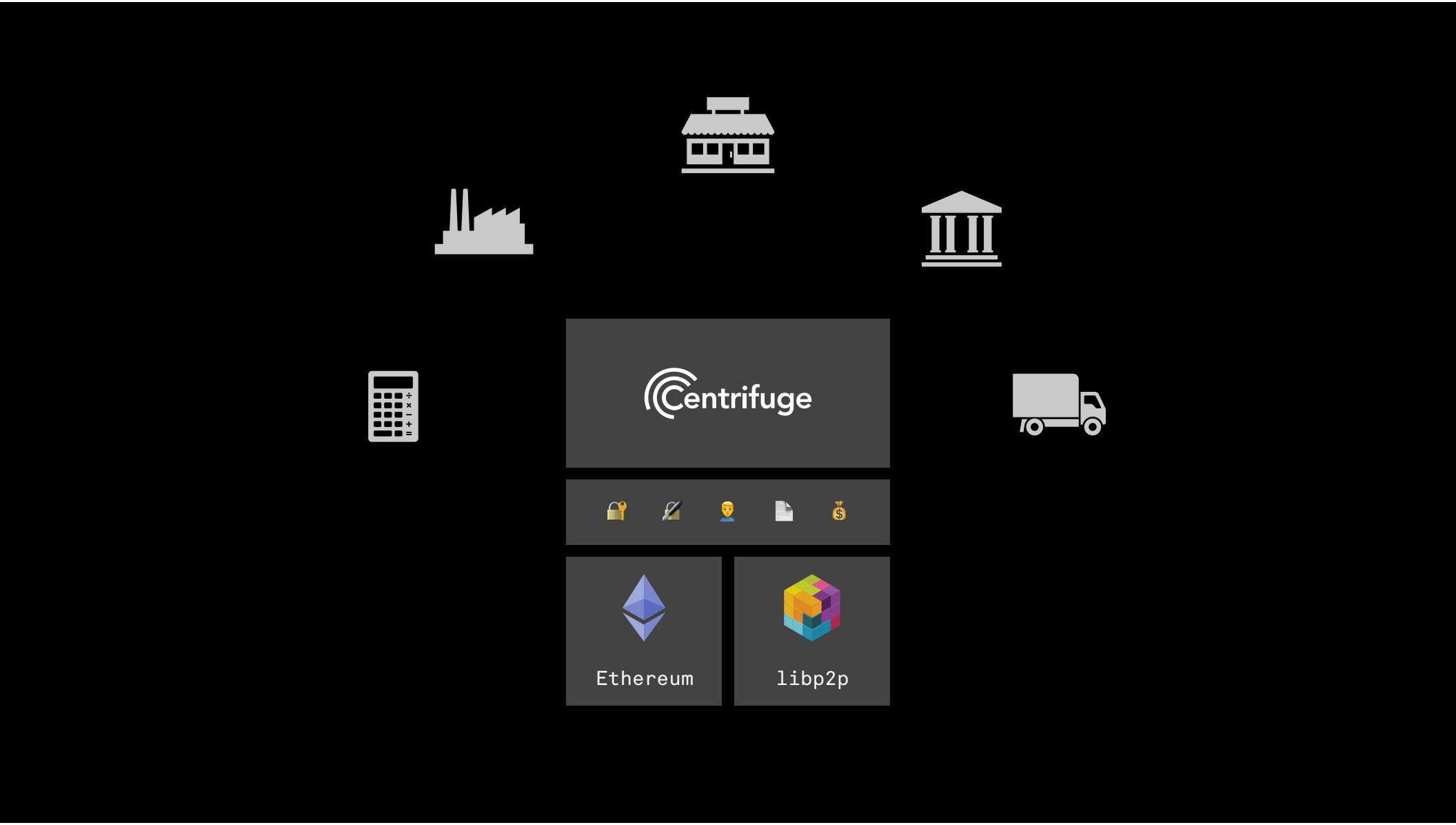


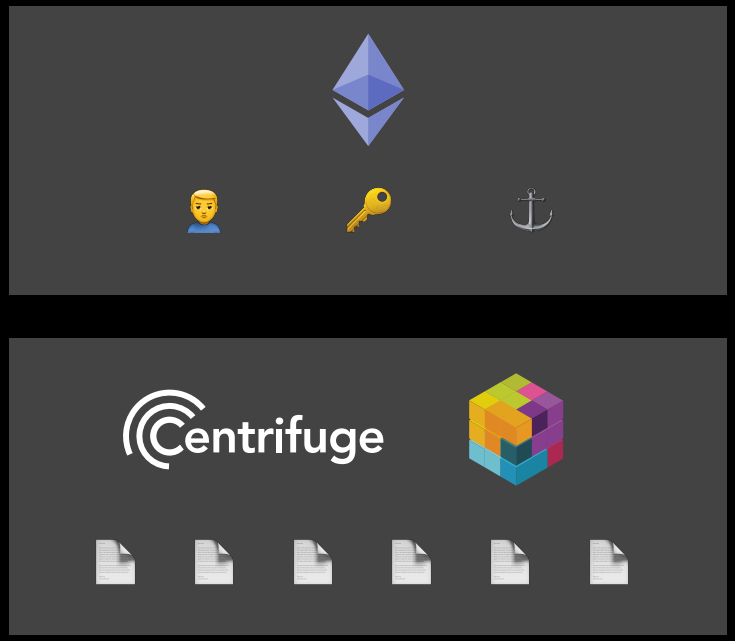
Layer 2 made easy

Implement a payment channel with libp2p

Centrifuge lucas vogelsang - @lucasvo


Centrifuge
Ethereum
libp2p

Centrifuge's P2P Privacy Layer



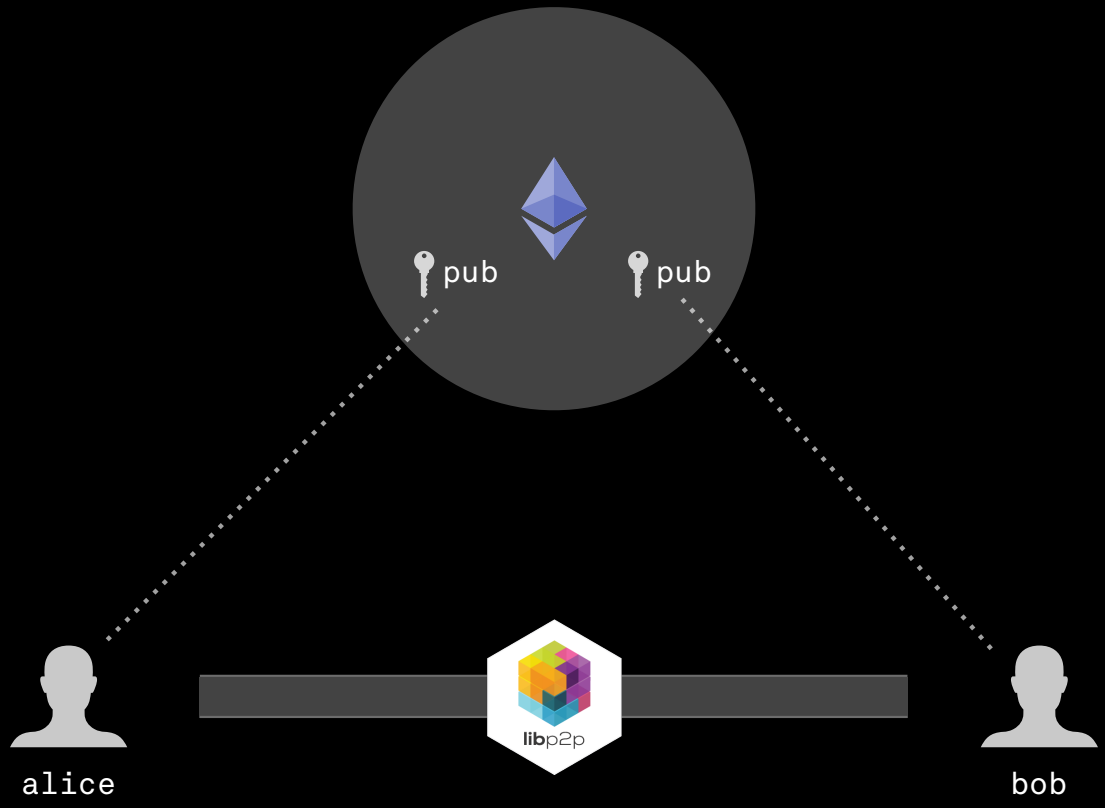
identities & anchors

public · global consensus · on-chain standards

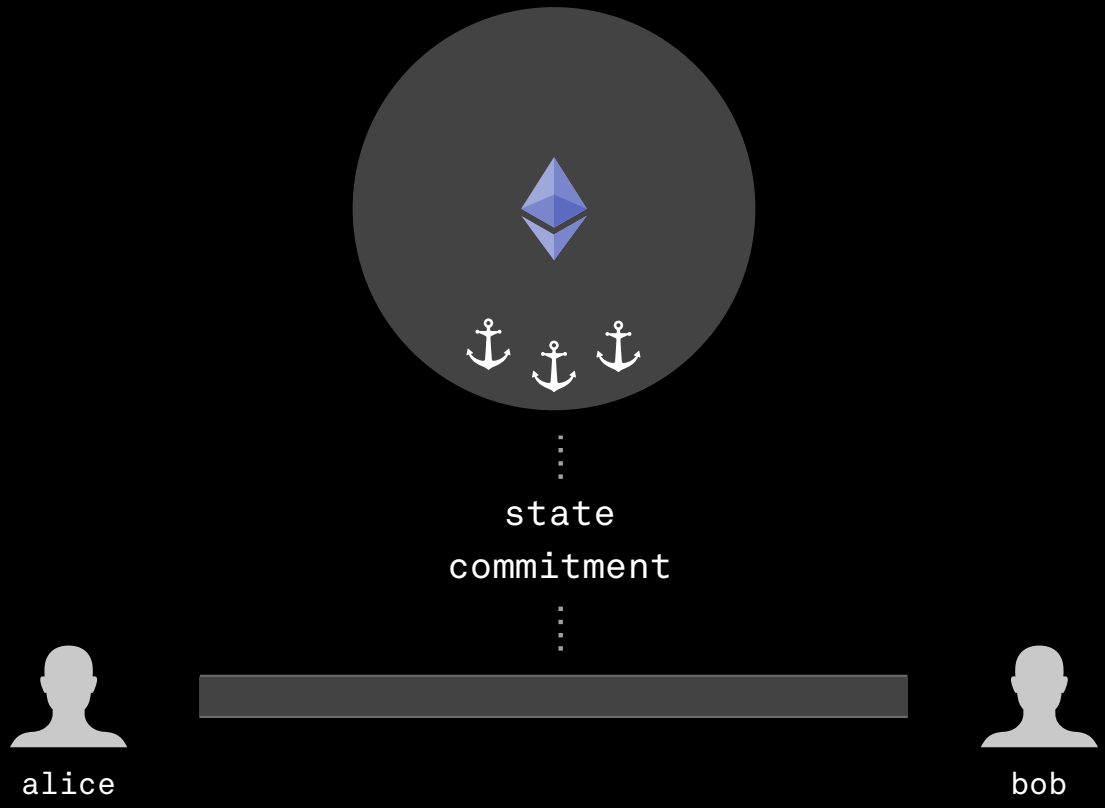
channels/documents

private · building business graph

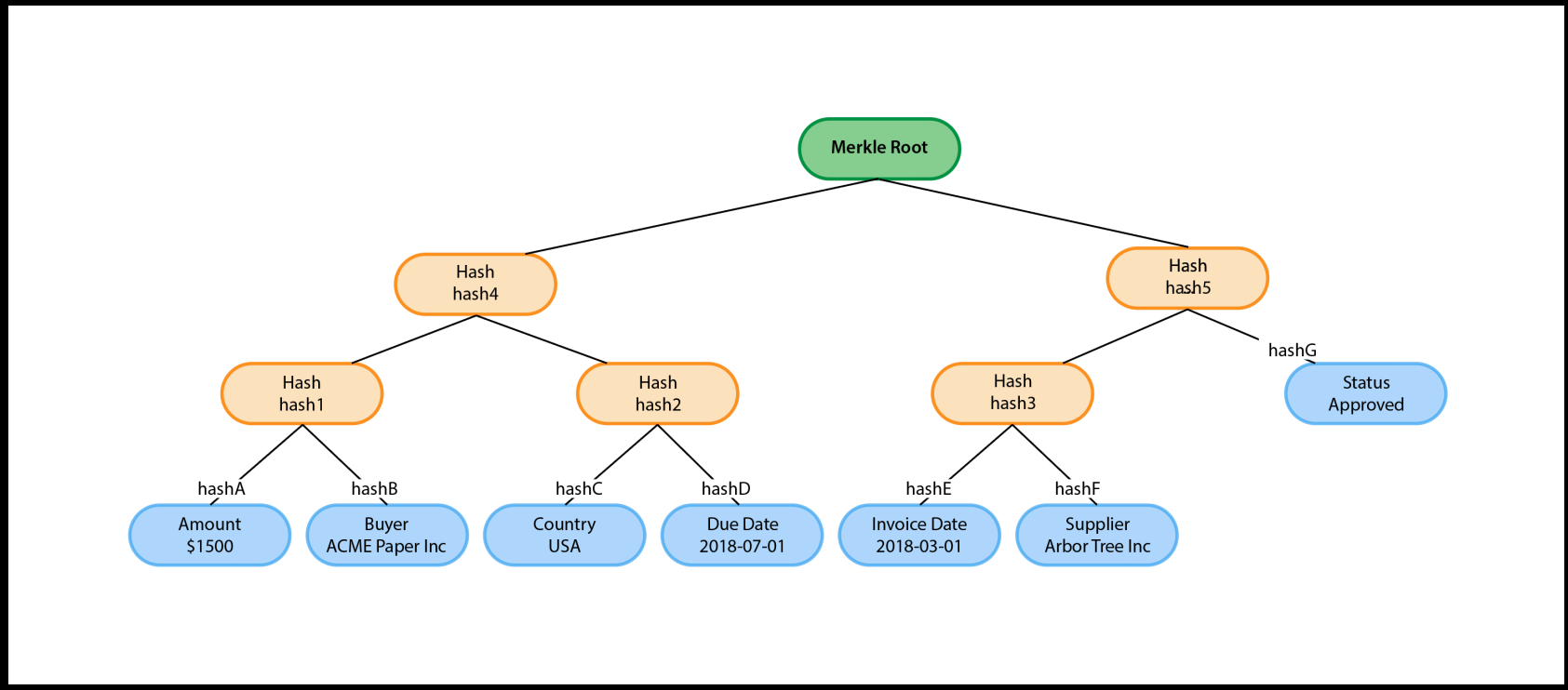
Centrifuge's P2P Privacy Layer



Centrifuge's P2P Privacy Layer



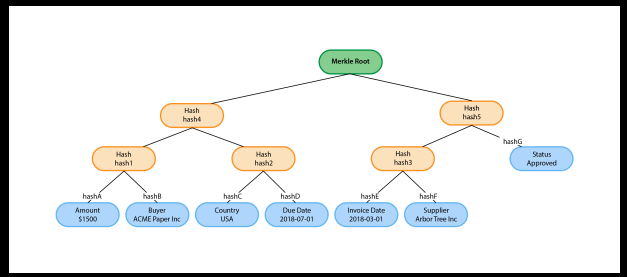
A merkle tree!



precise-proofs

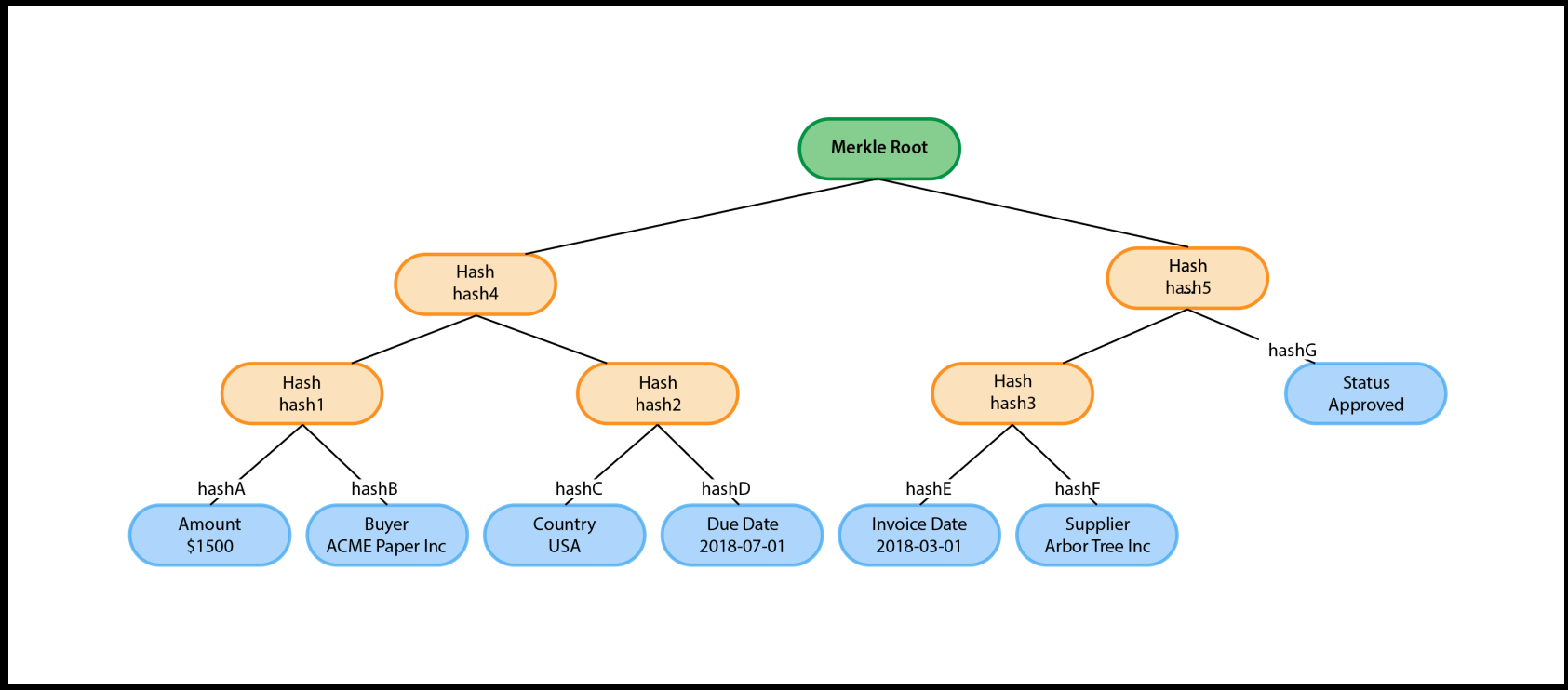
```
invoice

amount: $1500
buyer: acme paper
supplier: arbor tree
due date: 2018-…
…
```



merkle root

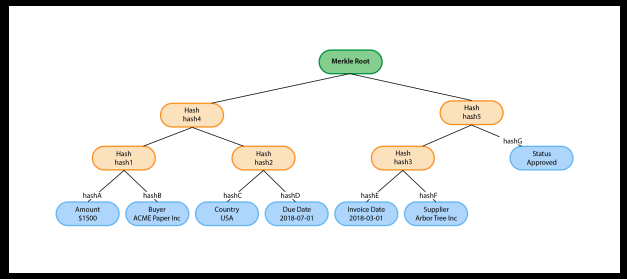
precise-proofs



precise-proofs

```
invoice

amount: $1500
buyer: acme paper
supplier: arbor tree
due date: 2018-…
…
```



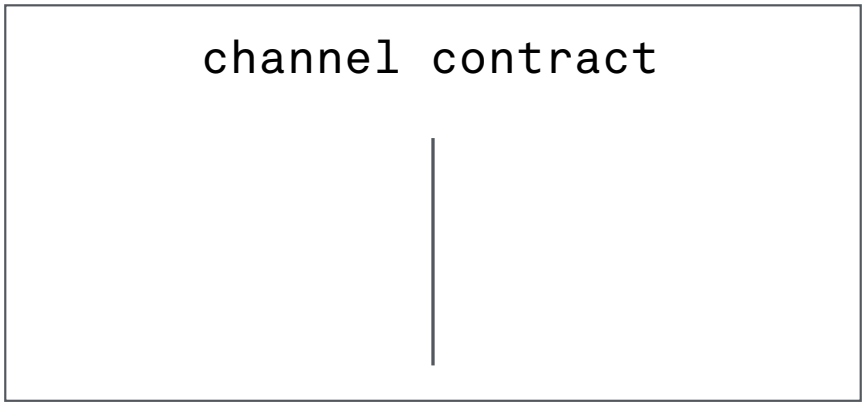
merkle root

precise-proofs: validate tx details instead of tx signature in a block

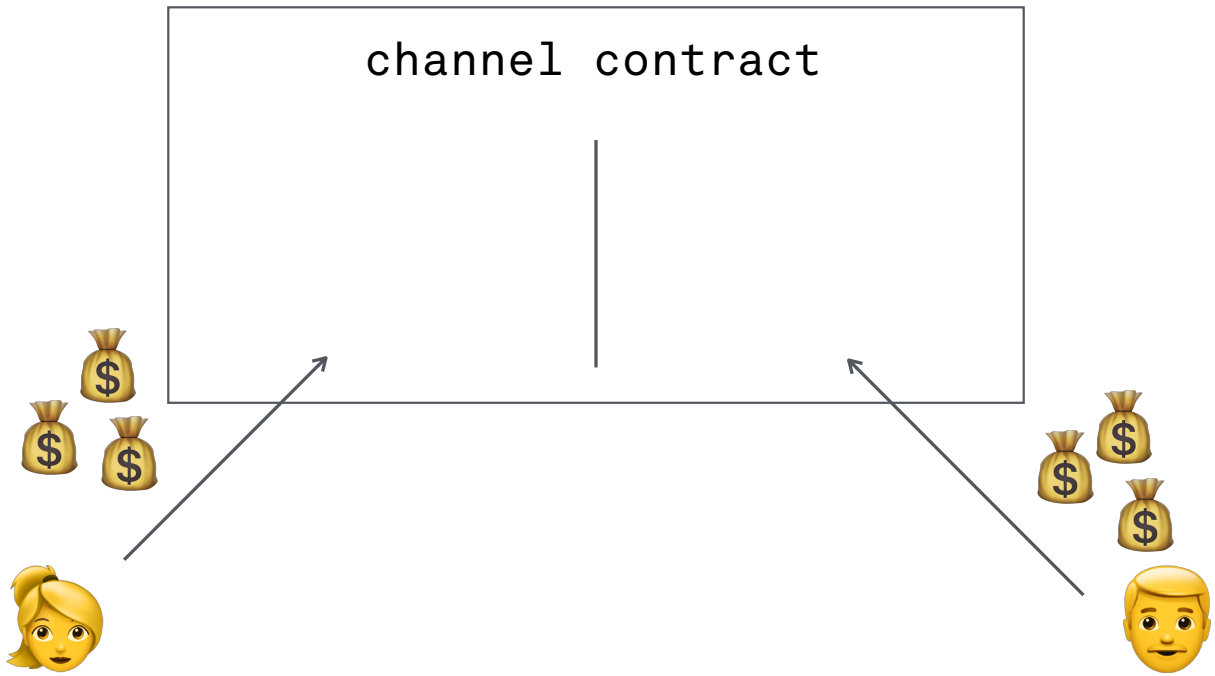
https://github.com/centrifuge/precise-proofs

contents

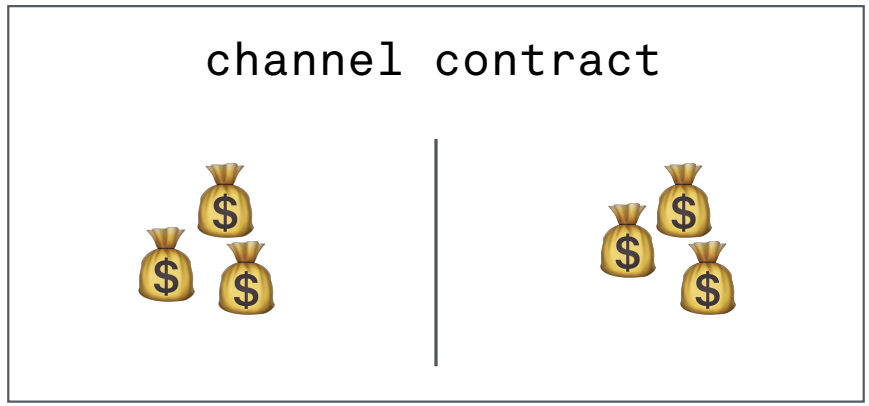
the story of alice & bob



fund contract

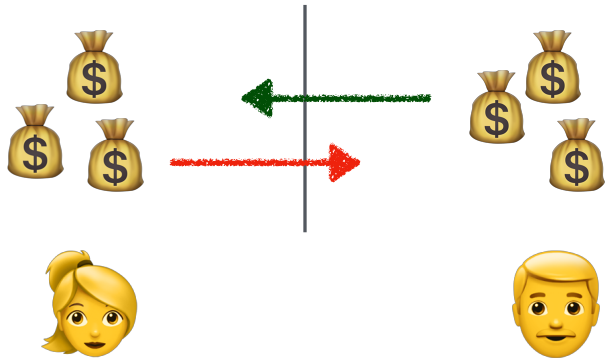


manipulate balance



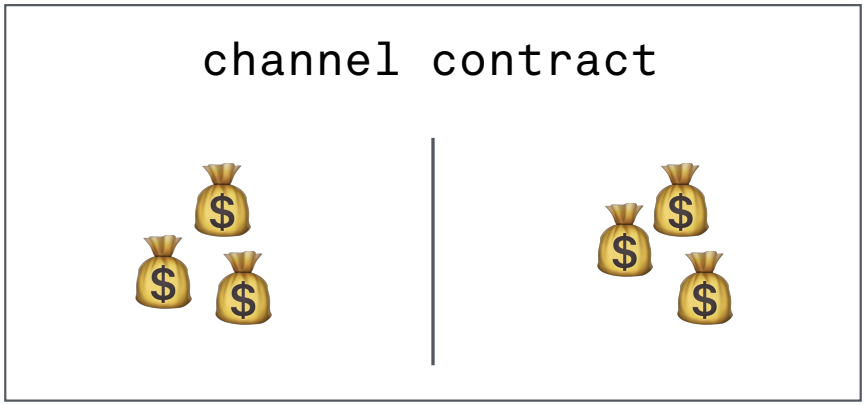
+1 

manipulate balance



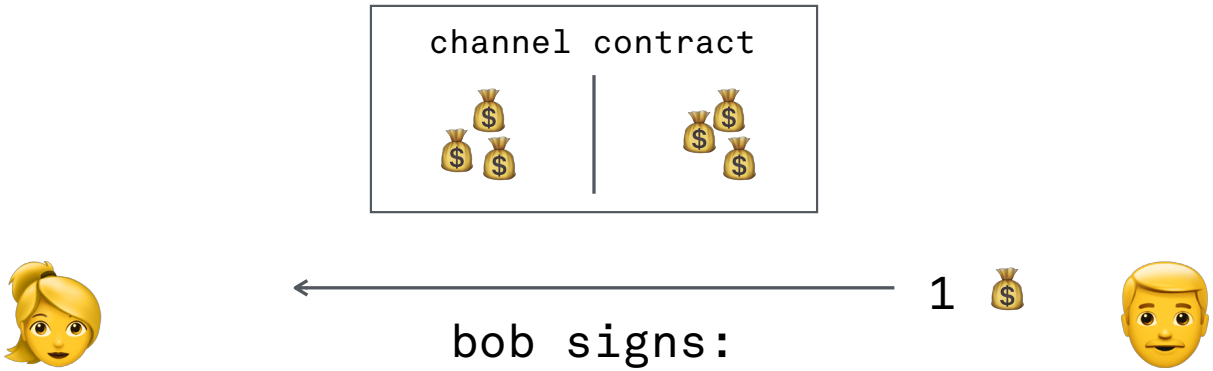
bob can only reduce his balance
alice can only reduce her balance

manipulate balance

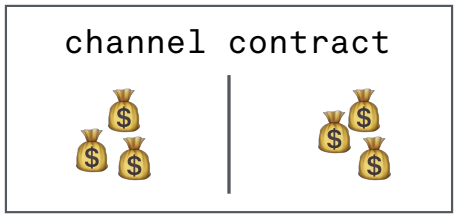


bob signs:
balance now is 4 for alice

manipulate balance



manipulate balance



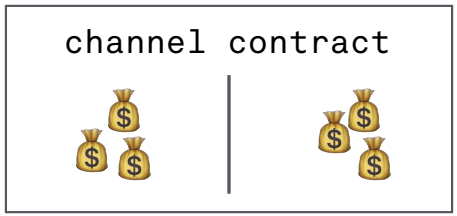
1

bob signs:
balance now is 4 for alice

1

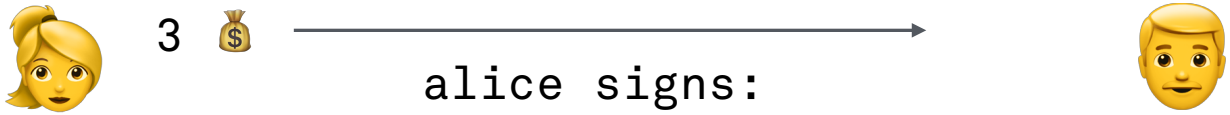
bob signs:
balance now is 5 for alice

manipulate balance



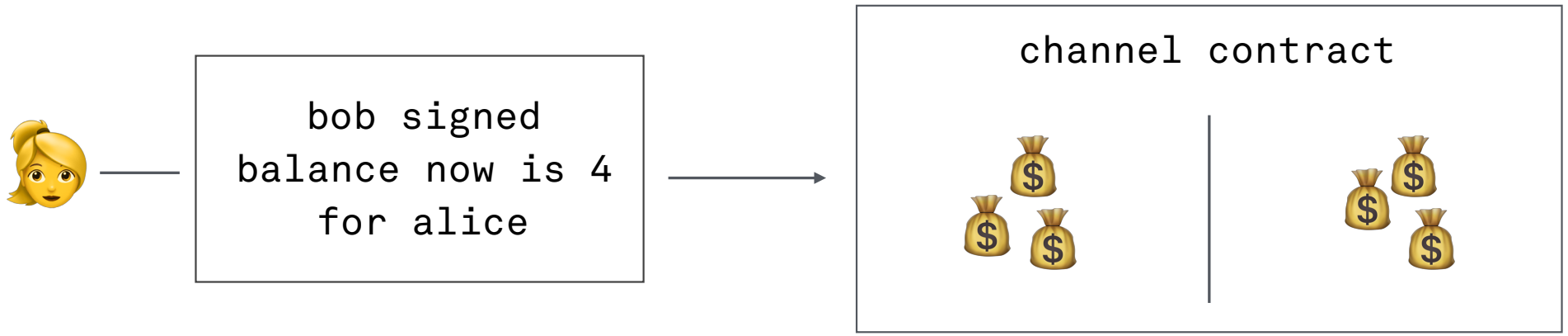
bob signs:
balance now is 4 for alice

bob signs:
balance now is 5 for alice

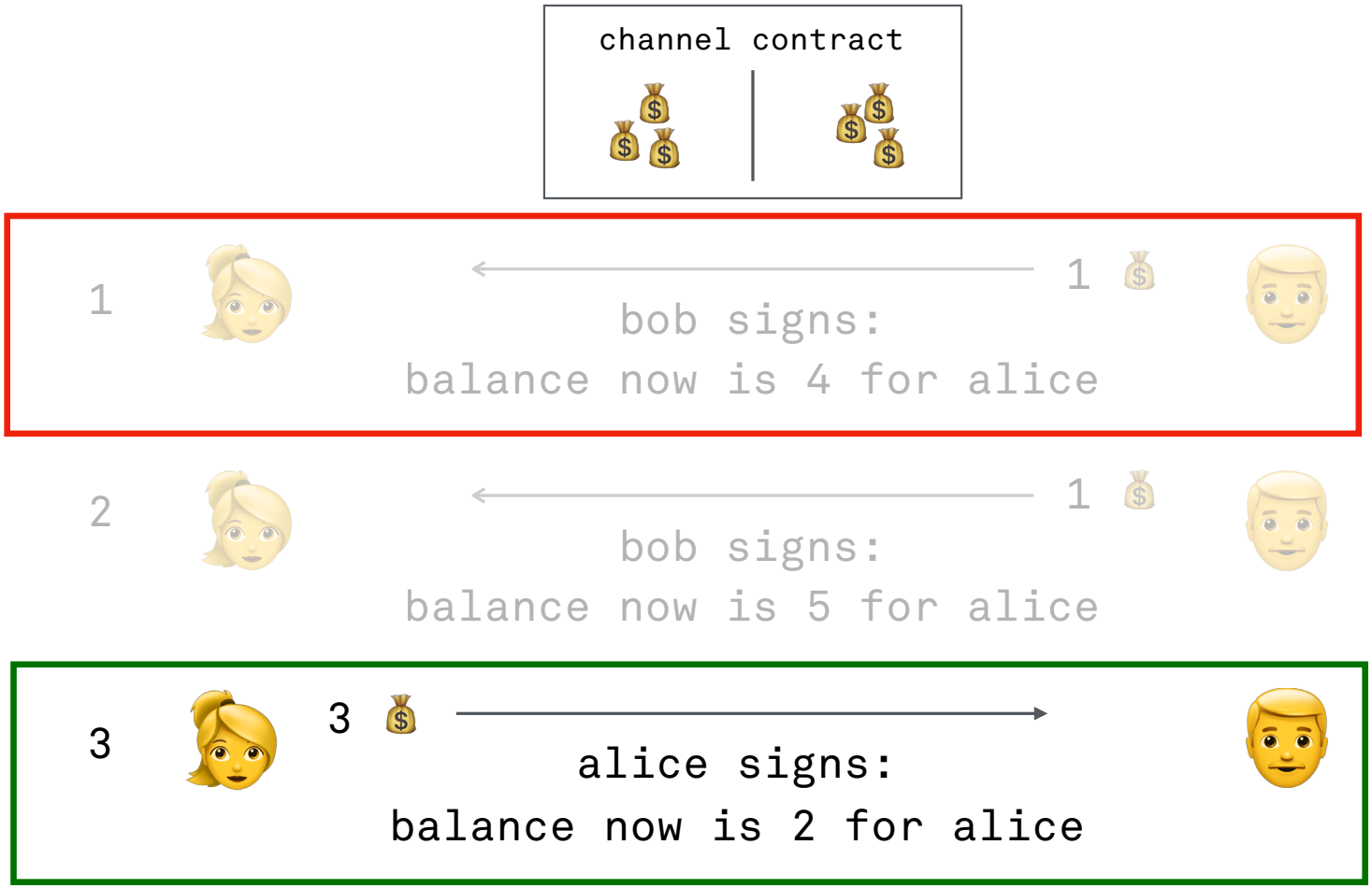


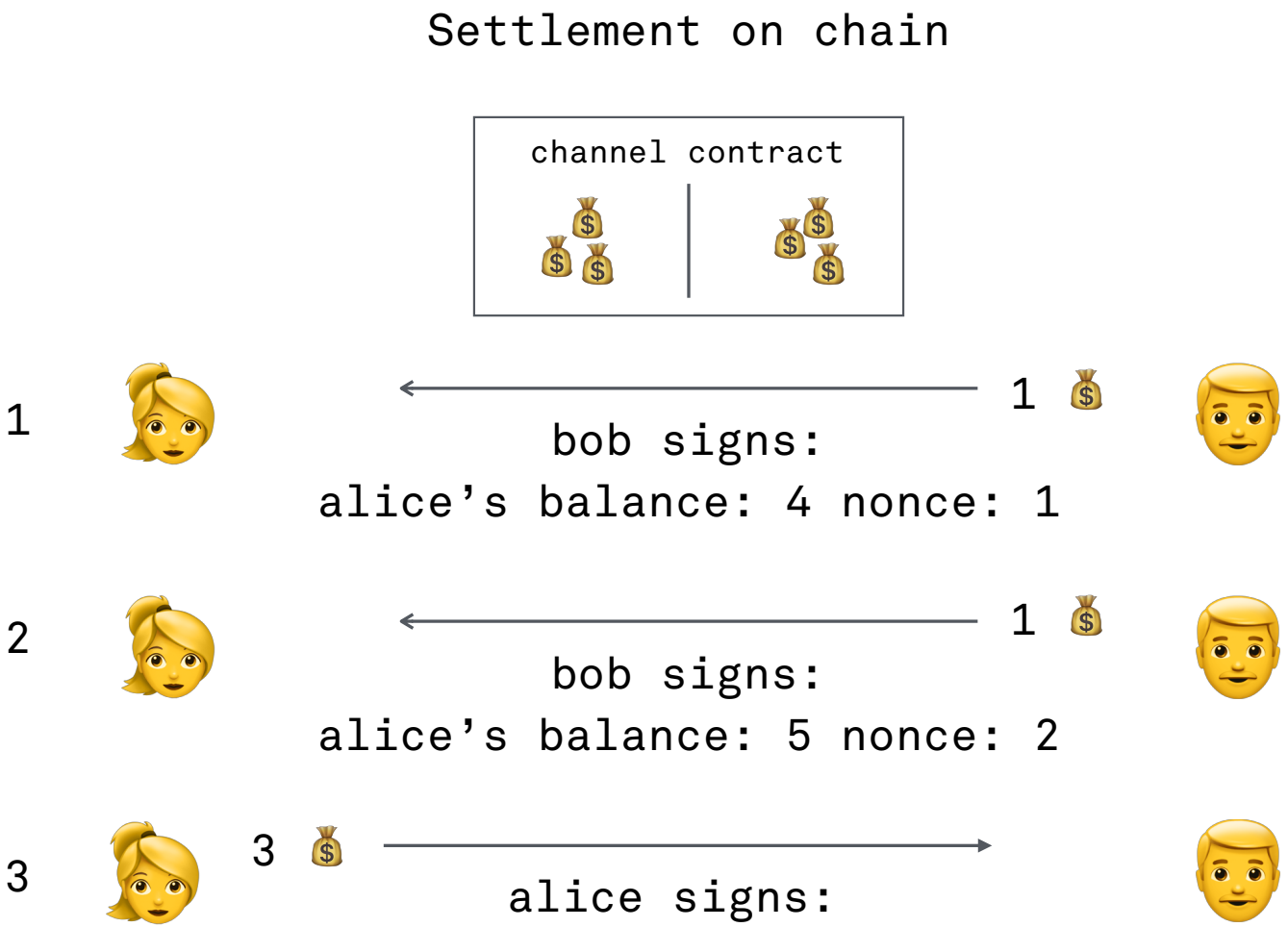
alice signs:
balance now is 2 for alice

settlement tx

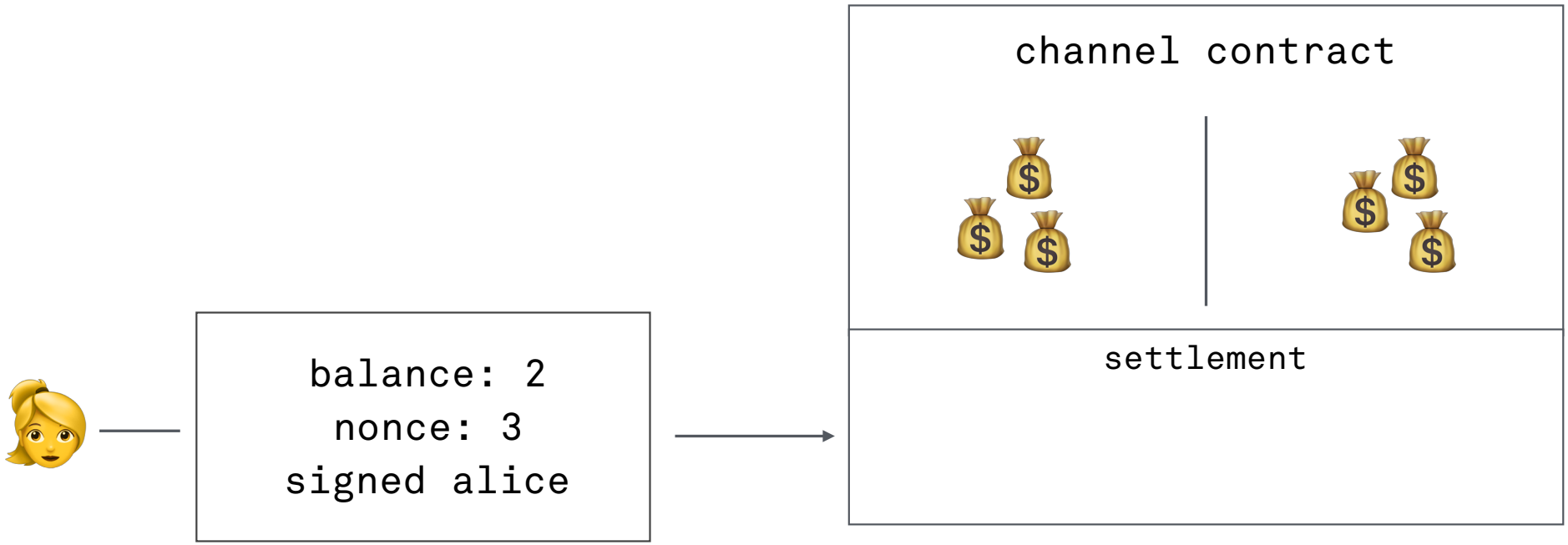


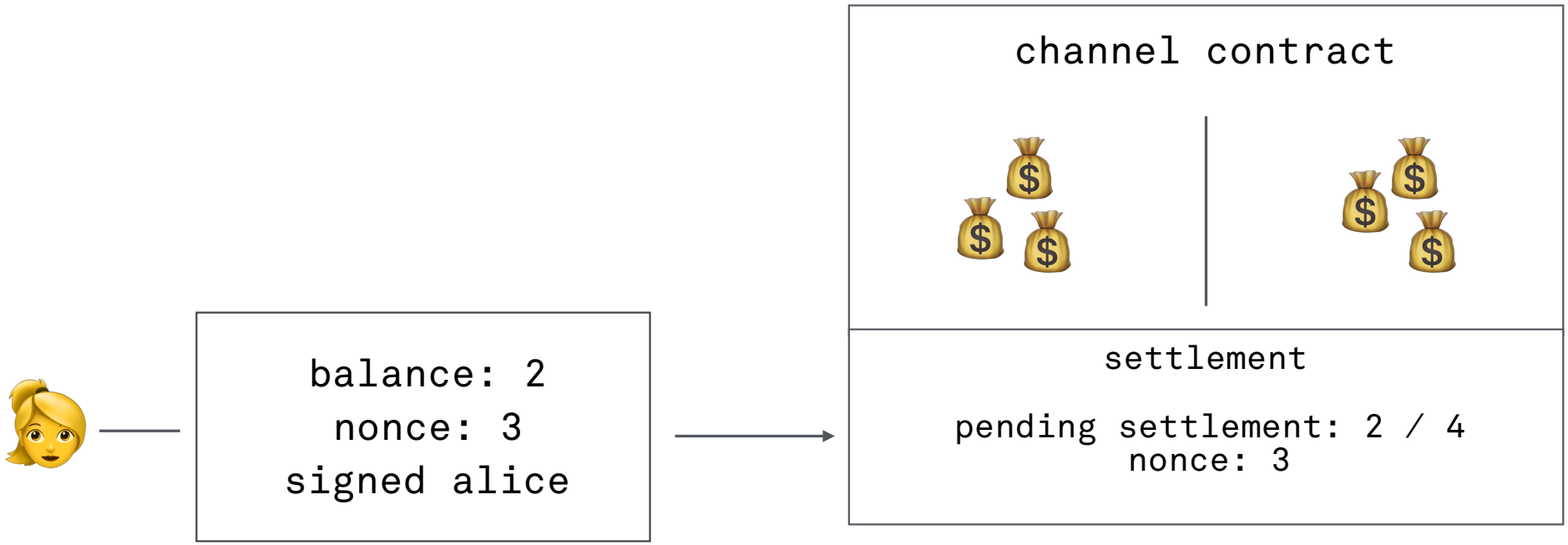
settling with an outdated state

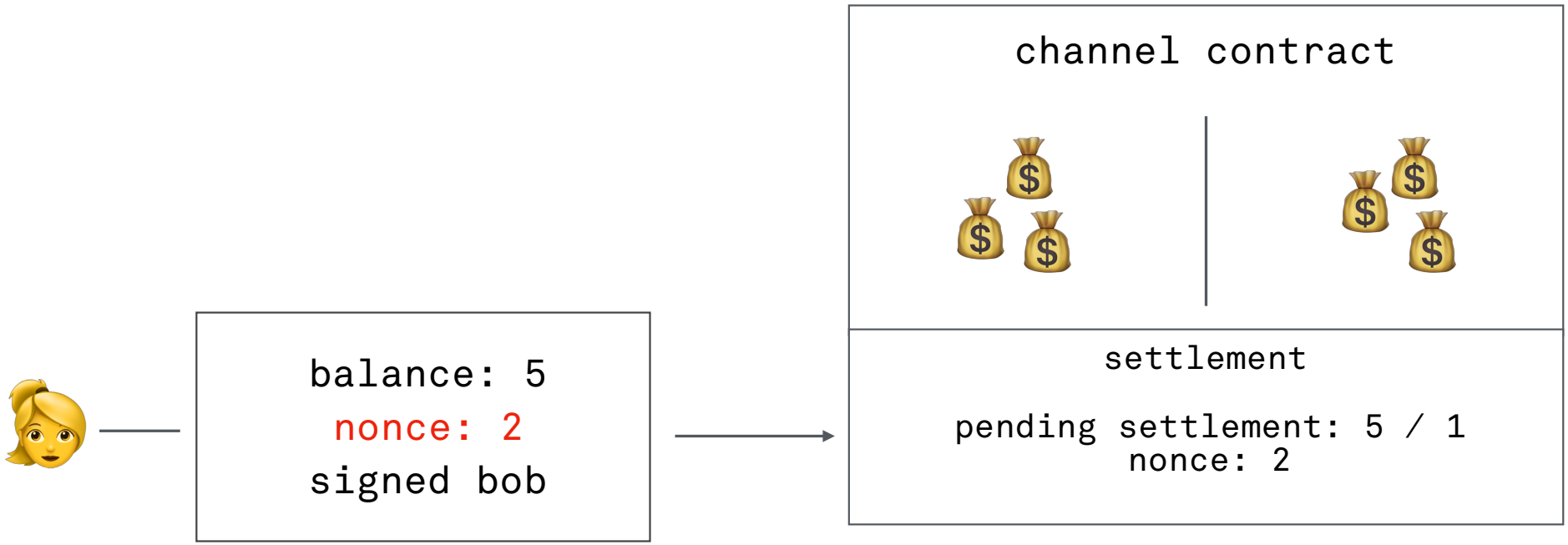


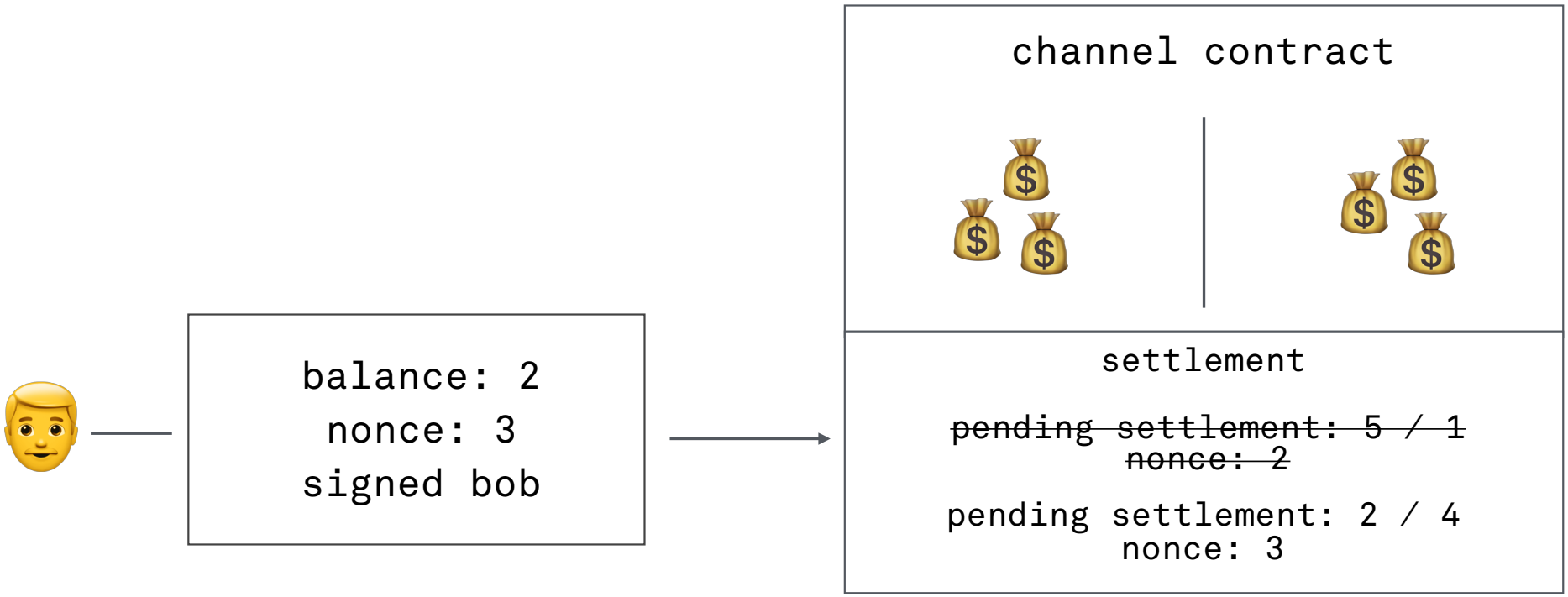

Settlement on chain
channel contract
1
1
bob signs:
alice's balance: 4 nonce: 1
2
1
bob signs:
alice's balance: 5 nonce: 2
3
3
alice signs:
alice's balance: 2 nonce: 3

Settlement

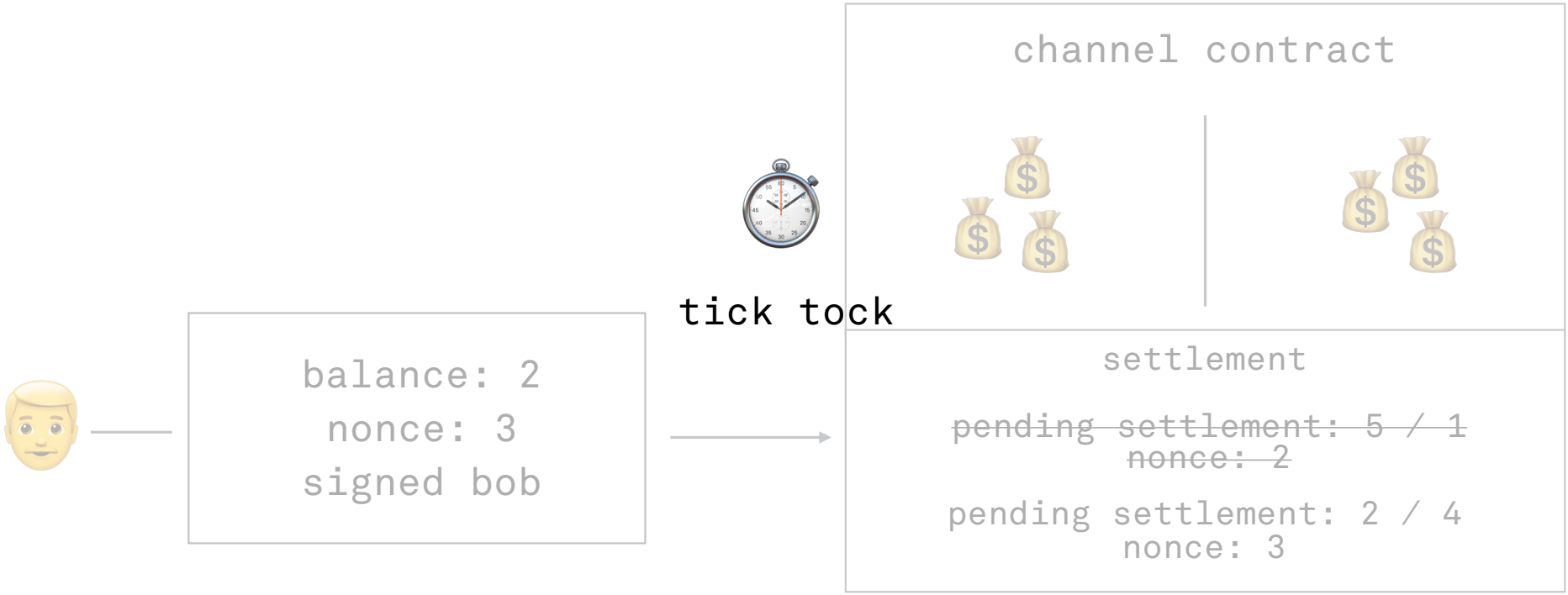

channel contract
settlement
balance: 2
nonce: 3
signed alice


settlement
channel contract
balance: 2
nonce: 3
signed alice
settlement
pending settlement: 2 / 4
nonce: 3

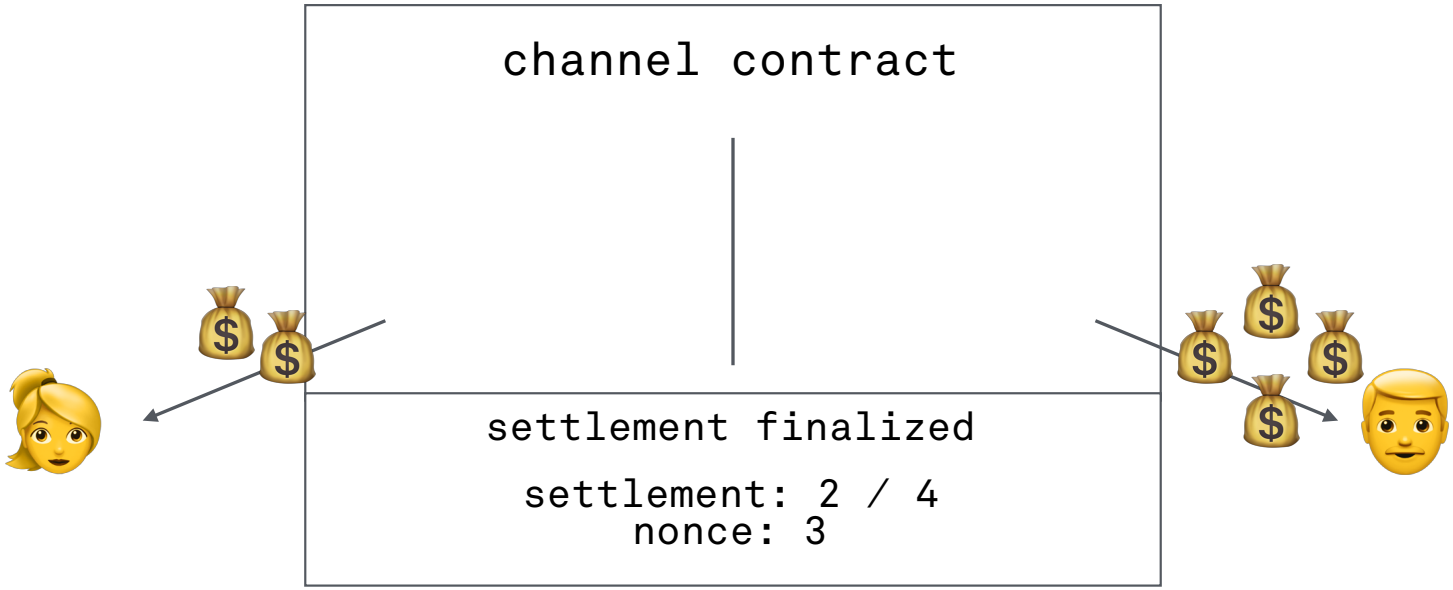

settlement
channel contract
balance: 5
nonce: 2
signed bob
settlement
pending settlement: 5 / 1
nonce: 2


settlement
channel contract
balance: 2
nonce: 3
signed bob
settlement
~~pending settlement: 5 / 1~~
~~nonce: 2~~
pending settlement: 2 / 4
nonce: 3

settlement


channel contract
tick tock
balance: 2
nonce: 3
signed bob
settlement
~~pending settlement: 5 / 1~~
~~nonce: 2~~
pending settlement: 2 / 4
nonce: 3

final settlement



message

when updating a channel:

- only make an update that reduces your balance
- include an increased nonce
- sign it with your key

settlement

when settling a channel:

- proof that your state update is based on an update by the counterparty
- you should submit the most recent known state
- wait for the settlement period to pass
- request final settlement

libp2p

- go/rust/js (some of it works in browser)
- secio: DH key exchange and encryption (the SSL of p2p)
- dht: various ways to discover nodes
- nat hole punching
- routing/protocol translation: translate between two different transports

on to the code

https://github.com/lucasvo/learning-paymentchannels

Requirements

- you will need nodejs/npm & curl
- config.js has all the details for both nodes (alice & bob)
- run with NODE_ROLE=(alice|bob) to trigger either config
- calls.sh has all the curl commands to interact with the node
- ports needed:
  - 3000 & 3001 for node rest api
  - 10333 & 10334 for libp2p
  - 8545 for ganache rpc

repo overview

contracts/ - truffle contracts

node/ - nodejs app with skeleton for rest api & libp2p

```
$ cd contracts/
$ npm install
...
$ cd node/
$ npm install
...
```

start ganache (test Ethereum chain)

```
$ cd contracts/
$ npm run ganache
```

```
# Shortcut for:
ganache-cli --gasLimit 4712388 --networkId 99999 -p 8545 -m \"tumble
gas embody bright agree pony smoke laptop index sight shallow hungry\"

Starts an Ethereum test chain with preconfigured wallets
```

channel manager contract methods

create channel

fund channel

settle channel

release funds

```
contract ChannelManager {
    enum ChannelState {
        INITIALIZED, // 0
        PARTYA_FUNDED, // 1
        PARTYB_FUNDED, // 2
        ACTIVE, // 3
        PENDING_SETTLEMENT, // 4
        SETTLED // 5
    }
    struct Channel {
        address partyA;
        address partyB;
        ERC20 currency;
        uint32 balance;
        ChannelState state;
    }

    mapping(bytes32 => Channel) _channels;

    function createChannel(bytes32 channelId, address partyA, address partyB, ERC20 currency, uint32 balance) public {
        // Abort if channelId is taken
        require(_channels[channelId].partyA == address(0), "channelId is already taken");


        // Validate provided Channel options
        require(partyA != address(0) && partyB != address(0), "parties must be non zero addresses");

        _channels[channelId] = Channel(partyA, partyB, currency, balance, ChannelState.INITIALIZED);
    }
```

fund

```
// fund will attempt to fund the channel by calling transferFrom on
// the currency by withdrawing from the specified from address
function fund(bytes32 _channelId) public {
    Channel storage channel = channels[_channelId];

    // Verify the sender is one of the two party and can fund the channel
    // - An initialized channel should only allow either A or B to fund
    // - A channel funded by A should only be fundable by B
    // - A channel funded by B should only be fundable by A
    require(
        (
            channel.state == ChannelState.INITIALIZED && (
                channel.partyA == msg.sender ||
                channel.partyB == msg.sender)
        ) ||
        (channel.state == ChannelState.PARTYA_FUNDED && channel.partyB == msg.sender)||
        (channel.state == ChannelState.PARTYB_FUNDED && channel.partyA == msg.sender),
        "Can't fund channel"
    );

    channel.currency.transferFrom(msg.sender, this, channel.balance);

    if (channel.state == ChannelState.INITIALIZED) {
        if (msg.sender == channel.partyA) {
            channel.state = ChannelState.PARTYA_FUNDED;
        } else {
            channel.state = ChannelState.PARTYB_FUNDED;
        }
    } else {
        channel.state = ChannelState.ACTIVE;
    }
}
```

withdraw

```
// The withdraw method transfers the final amount to both parties whenever it is
// called and enough blocks have been mined since the last settlement transaction
function withdraw(bytes32 _channelId)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");

    require(channel.state == ChannelState.PENDING_SETTLEMENT &&
            channel.settlementBlock + blockTimeout < block.number,
            "channel must be in pending settlement state and more than `blockTimeout` must have passed.");

    channel.currency.transfer(channel.partyA, channel.balanceA);
    channel.currency.transfer(channel.partyB, channel.balance*2-channel.balanceA);
    channel.state = ChannelState.WITHDRAWN;
}
```

channel manager contract methods

create channel

fund channel

settle channel 👈

release funds

settle transaction

bob proves that alice sent him a state update with nonce X and balance Y

settlement conditions:

- only allow settlement of non-settled channels
- nonce must be increased
- balance can’t be more than total channel amount
- signature must be valid

for simplicity: we don’t implement cooperative settlement & settlement of self signed transaction

settle transaction

bob proves that alice sent him a state update with nonce X and balance Y

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
// for simplicity, we only allow settlement by submitting a signed state update
// from your counterparty. Settling with your own update is not implement.
function settle(bytes32 _channelId, uint32 _nonce, uint32 _balanceA, bytes _signature)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
// for simplicity, we only allow settlement by submitting a signed state update
// from your counterparty. Settling with your own update is not implement.
function settle(bytes32 _channelId, uint32 _nonce, uint32 _balanceA, bytes _signature)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");

    require(channel.state == ChannelState.ACTIVE || channel.state ==  ChannelState.PENDING_SETTLEMENT,
            "can't settle channel. invalid state");
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
// for simplicity, we only allow settlement by submitting a signed state update
// from your counterparty. Settling with your own update is not implement.
function settle(bytes32 _channelId, uint32 _nonce, uint32 _balanceA, bytes _signature)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");

    require(channel.state == ChannelState.ACTIVE || channel.state ==  ChannelState.PENDING_SETTLEMENT,
            "can't settle channel. invalid state");
    require(channel.balance*2 >= _balanceA,
            "channel balance for partyA can't be larger than total amount in escrow");
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
// for simplicity, we only allow settlement by submitting a signed state update
// from your counterparty. Settling with your own update is not implement.
function settle(bytes32 _channelId, uint32 _nonce, uint32 _balanceA, bytes _signature)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");

    require(channel.state == ChannelState.ACTIVE || channel.state ==  ChannelState.PENDING_SETTLEMENT,
            "can't settle channel. invalid state");
    require(channel.balance*2 >= _balanceA,
            "channel balance for partyA can't be larger than total amount in escrow");
    require(channel.nonce < _nonce, "nonce must be increased");
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
// for simplicity, we only allow settlement by submitting a signed state update
// from your counterparty. Settling with your own update is not implement.
function settle(bytes32 _channelId, uint32 _nonce, uint32 _balanceA, bytes _signature)  public {
    Channel storage channel = channels[_channelId];
    require(channel.partyA == msg.sender || channel.partyB == msg.sender,
            "Can only be called by one of the two members");

    require(channel.state == ChannelState.ACTIVE || channel.state ==  ChannelState.PENDING_SETTLEMENT,
            "can't settle channel. invalid state");
    require(channel.balance*2 >= _balanceA,
            "channel balance for partyA can't be larger than total amount in escrow");
    require(channel.nonce < _nonce, "nonce must be increased");

    address counterparty = channel.partyB;
    if (channel.partyB == msg.sender) {
        counterparty = channel.partyA;
    }
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

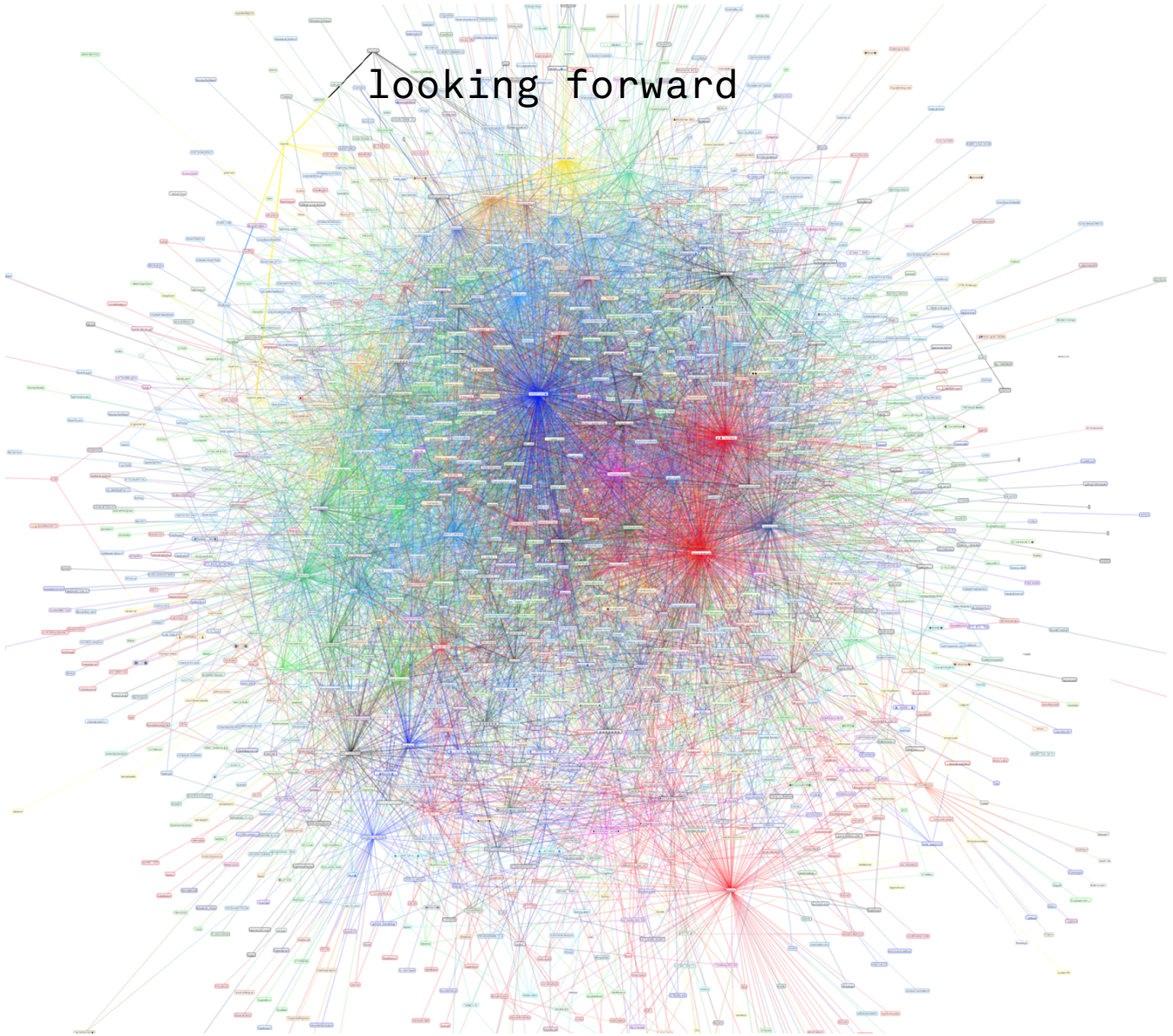
```
bytes32 message = keccak256(abi.encodePacked(_channelId, _nonce, _balanceA, counterparty));
address signer = message.toEthSignedMessageHash().recover(_signature);
require(signer == counterparty, "message needs to be signed by counterparty");
```

settle transaction

```
ChannelManager.settle(_channelId, _nonce, _balanceA, _signature)
```

```
    bytes32 message = keccak256(abi.encodePacked(_channelId, _nonce, _balanceA, counterparty));
    address signer = message.toEthSignedMessageHash().recover(_signature);
    require(signer == counterparty, "message needs to be signed by counterparty");

    channel.nonce = _nonce;
    channel.balanceA = _balanceA;
    channel.settlementBlock = block.number;
    channel.state = ChannelState.PENDING_SETTLEMENT;
}
```


looking forward

looking forward

multi hop payments
atomic multi path payments

onion routing
watchtowers
generalized state channels

resources

LibP2P with Go: https://github.com/mikiquantum/rock-paper-scissors-demo/
LibP2P Specs: https://github.com/libp2p/specs
Lightning RFC: https://github.com/lightningnetwork/lightning-rfc

lucas vogelsang
@lucasvo
lucas@centrifuge.io
http://lucasvo.com