ethereum
vienna

The state of state channels

September 3rd 2018

Starts at 18:15

Agenda

1. Constantinople Fork Changes
2. The state of state channels
3. Socialising

Constantinople Fork Changes

Constantinople

The EIPs for the upcoming Hardfork were decided last week:

EIP 145: Bitwise shifting instructions in EVM

Can bring some minor optimisations to contracts

EIP 1014: Skinny CREATE2

New variant of the CREATE opcode, doesn’t use nonce in address generation

Constantinople

EIP 1052: EXTCODEHASH Opcode

Makes checking wether a contract has the expected code cheaper

EIP 1283: Net gas metering for SSTORE without dirty maps

Makes storage writes cheaper if already written earlier in the transaction

Could drastically change the gas cost of some contracts

Constantinople

EIP 1234: Delay bomb and reduce block reward to 2 ETH

Difficulty bomb is delayed by another 12 months

Uses the same fake_block_number mechanism as Byzantium

Block rewards drops from 3ETH to 2ETH

EIP 1218: Blockhash refactoring was delayed again

The state of state channels

Outline

1. Payment Channels
2. State Channels
3. Counterfactual Instantiation
4. Alternate implementations

Payment Channels

Problems

- Blockchains don’t scale very well
- Fees are incurred on every state update
- No instant “finality”
- Privacy: Everybody can see every individual payment

“Solution”

Most payments can be moved off-chain if

- they happen between a closed set of participants
- we assume transactions can be included within a certain time frame

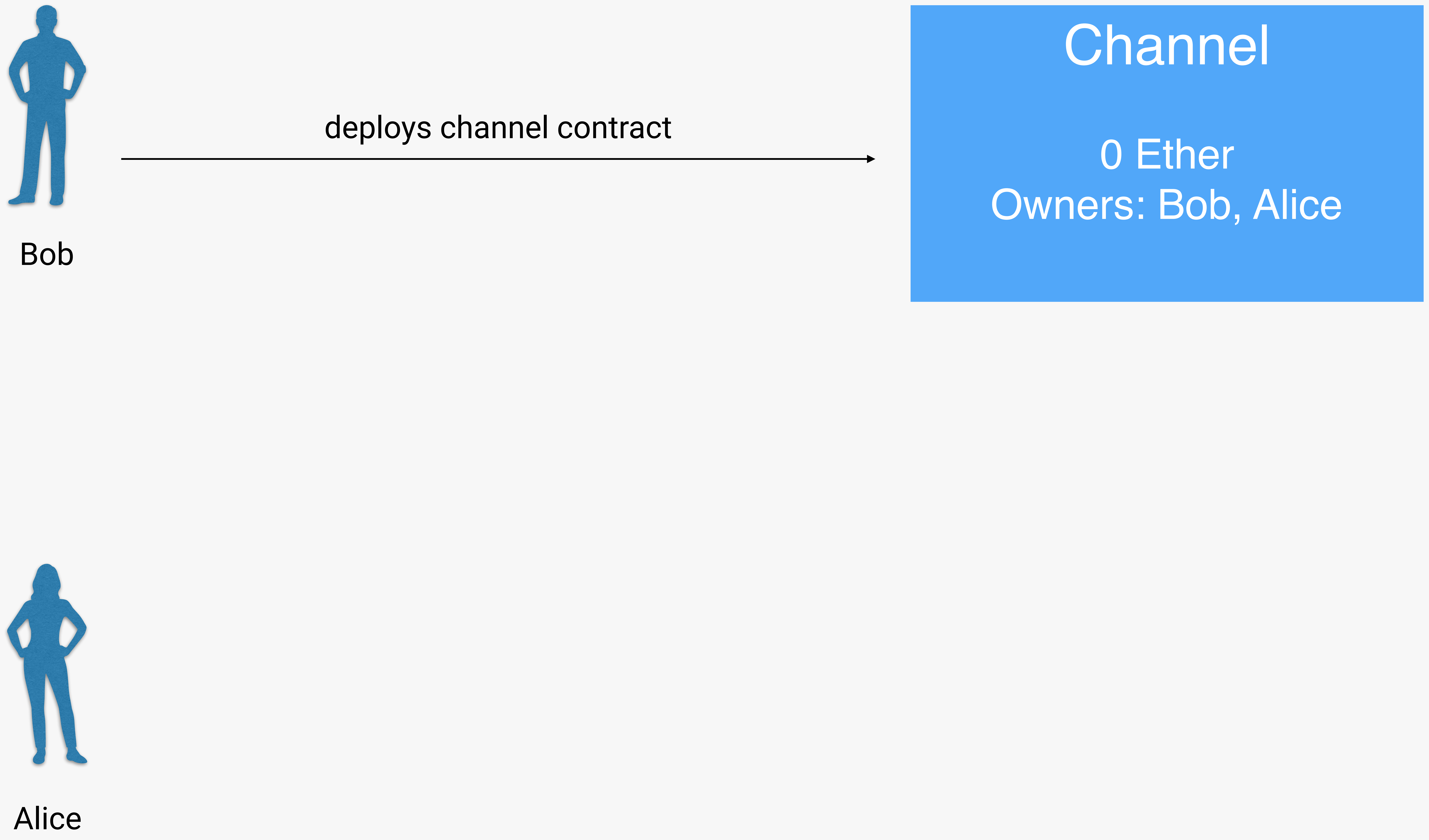
=> Use blockchain only for settlement and dispute resolution

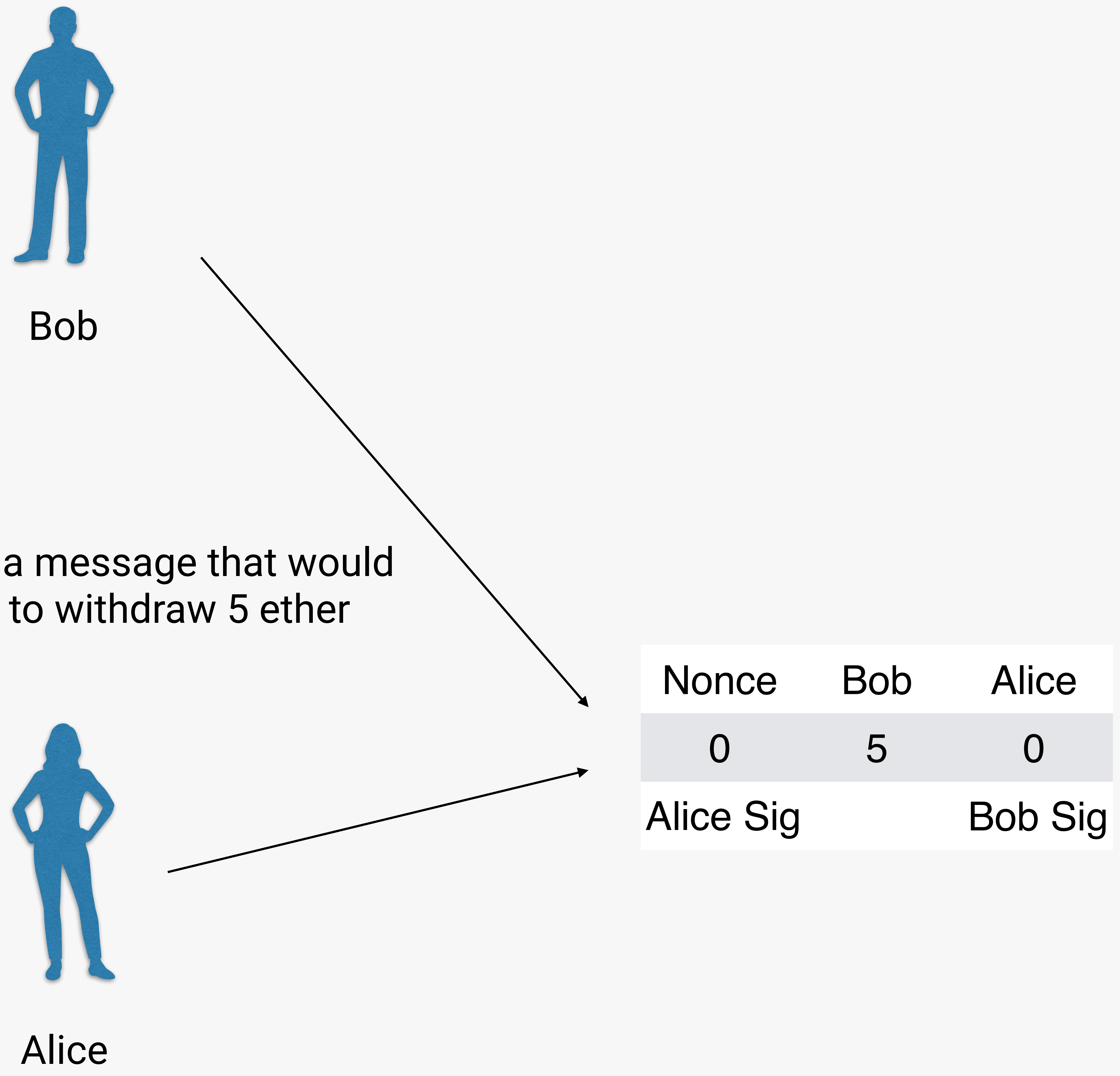
Core Idea

1. All funds are held in a multi-sig-like contract
2. All parties together sign and share updated states off-chain
    - This implies that every update requires **unanimous consent**
    - The state in this case is the distribution of the assets in the channel
    - The state also includes an increasing nonce (to detect stale states)

Core Idea

3. When the channel needs to be closed **any** party can submit the latest state
4. The channel cannot know wether this really is the latest state
   - The funds cannot be released immediately => Timeout needed
   - During this time other parties can submit later versions of the state
   - Later versions can be detected due to the higher nonce


Bob
deploys channel contract
Channel
0 Ether
Owners: Bob, Alice
Alice

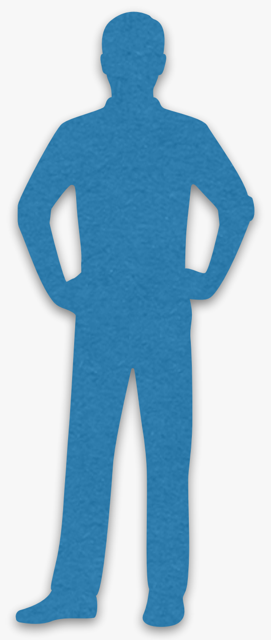


Both sign a message that would allow Bob to withdraw 5 ether

Nonce	Bob	Alice
0	5	0
Alice Sig		Bob Sig

Channel

0 Ether
Owners: Bob, Alice

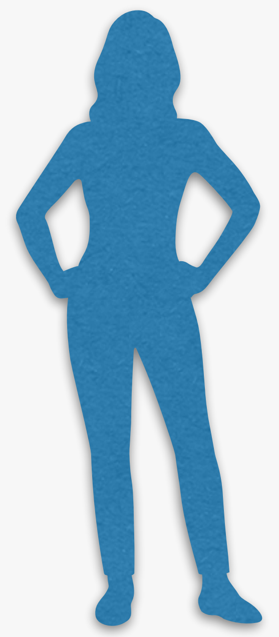


Bob

Bob can now deposit 5 ETH
knowing they can be retrieved again

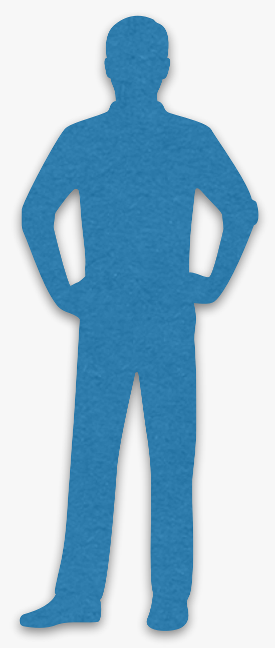
Channel

5 Ether
Owners: Bob, Alice



Alice

Nonce	Bob	Alice
0	5	0
Alice Sig		Bob Sig


Bob

Channel

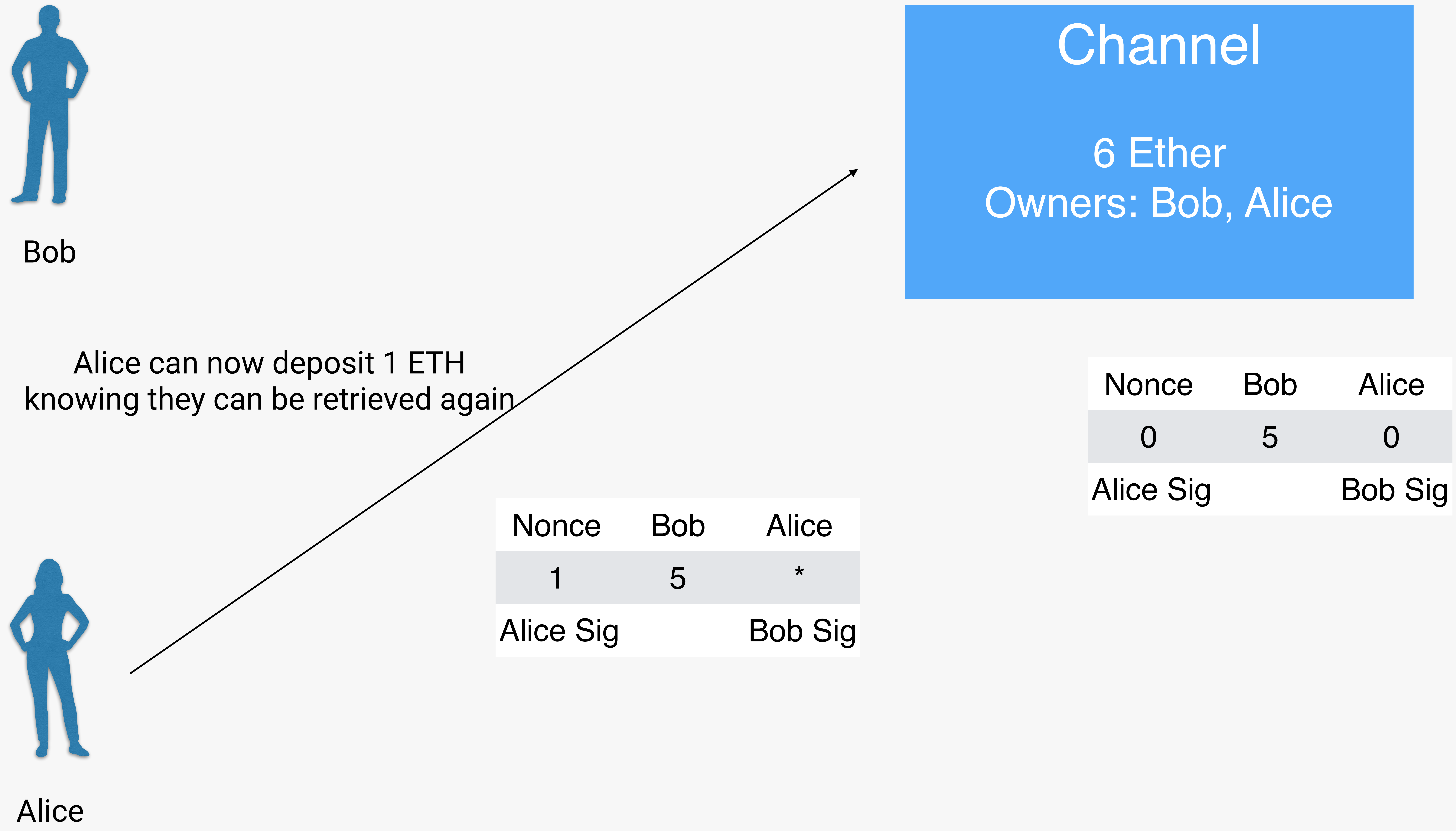
5 Ether
Owners: Bob, Alice

Nonce	Bob	Alice
0	5	0
Alice Sig		Bob Sig

Both sign a message that would allow
- Bob to withdraw 5 ether
- Alice to withdraw the rest

Nonce	Bob	Alice
1	5	*
Alice Sig		Bob Sig

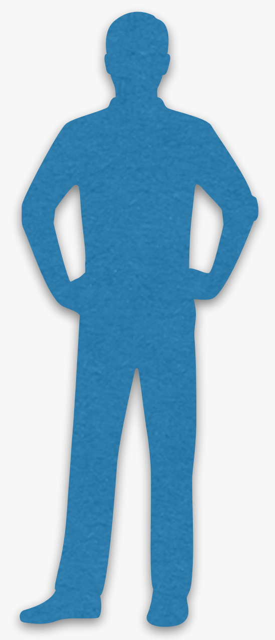
Alice



Alice can now deposit 1 ETH knowing they can be retrieved again

Nonce	Bob	Alice
1	5	*
Alice Sig		Bob Sig

Nonce	Bob	Alice
0	5	0
Alice Sig		Bob Sig

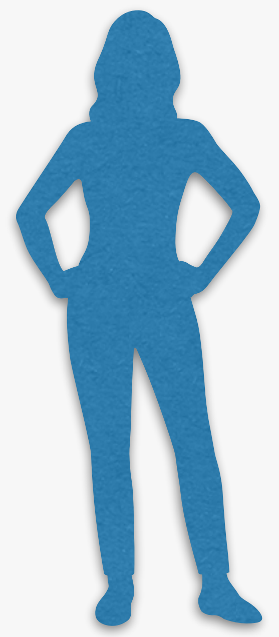


Channel

6 Ether
Owners: Bob, Alice

Both sign a message that would allow

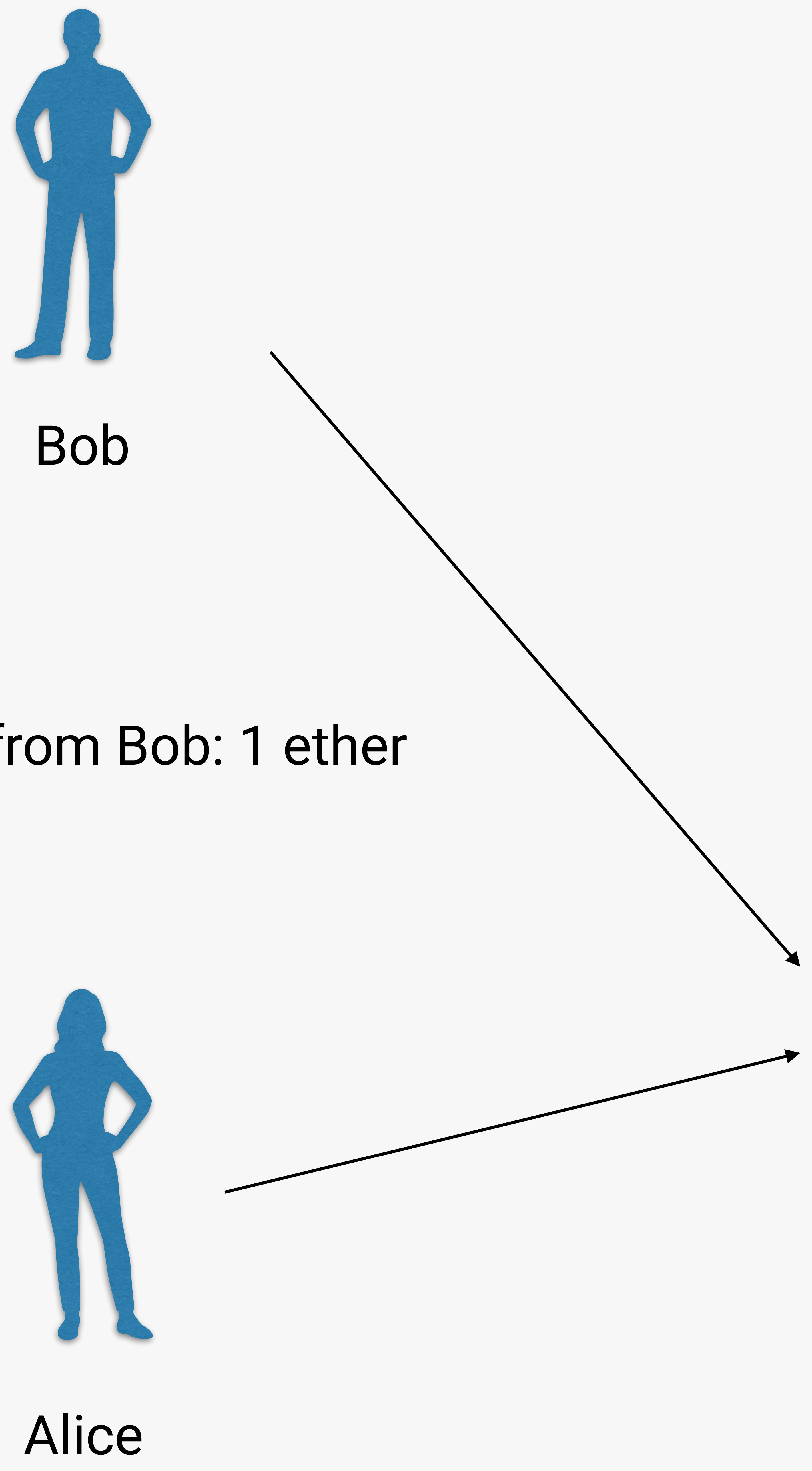
- Bob to withdraw 5 ether
- Alice to withdraw 1 ether



Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
0	5	0
Alice Sig		Bob Sig

Nonce	Bob	Alice
1	5	*
Alice Sig		Bob Sig



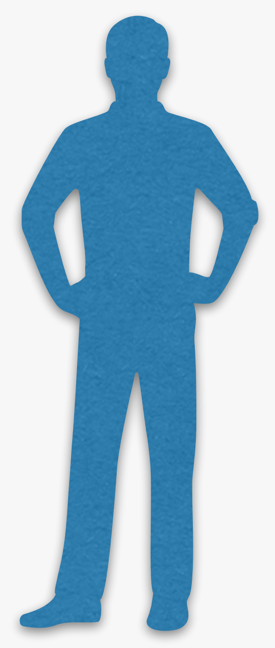
Channel

6 Ether
Owners: Bob, Alice

Payment from Bob: 1 ether

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig


Bob

Channel

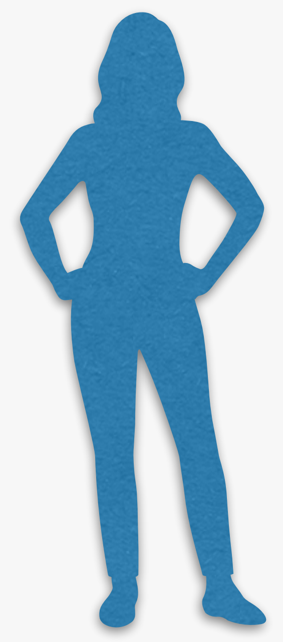
6 Ether
Owners: Bob, Alice

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Another payment from Bob: 1 ether

Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

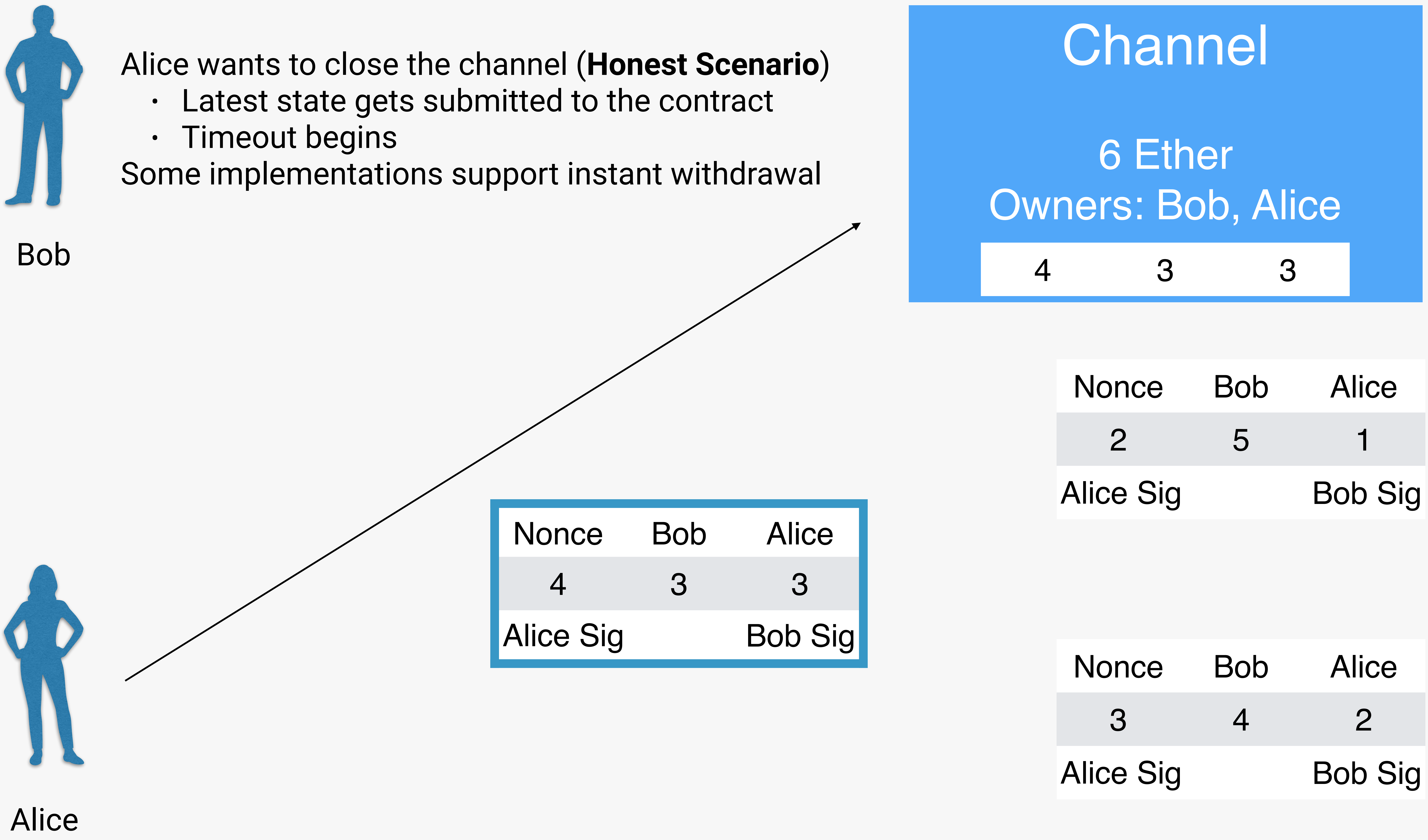
Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig


Alice

Alice wants to close the channel (**Honest Scenario**)

- Latest state gets submitted to the contract
- Timeout begins

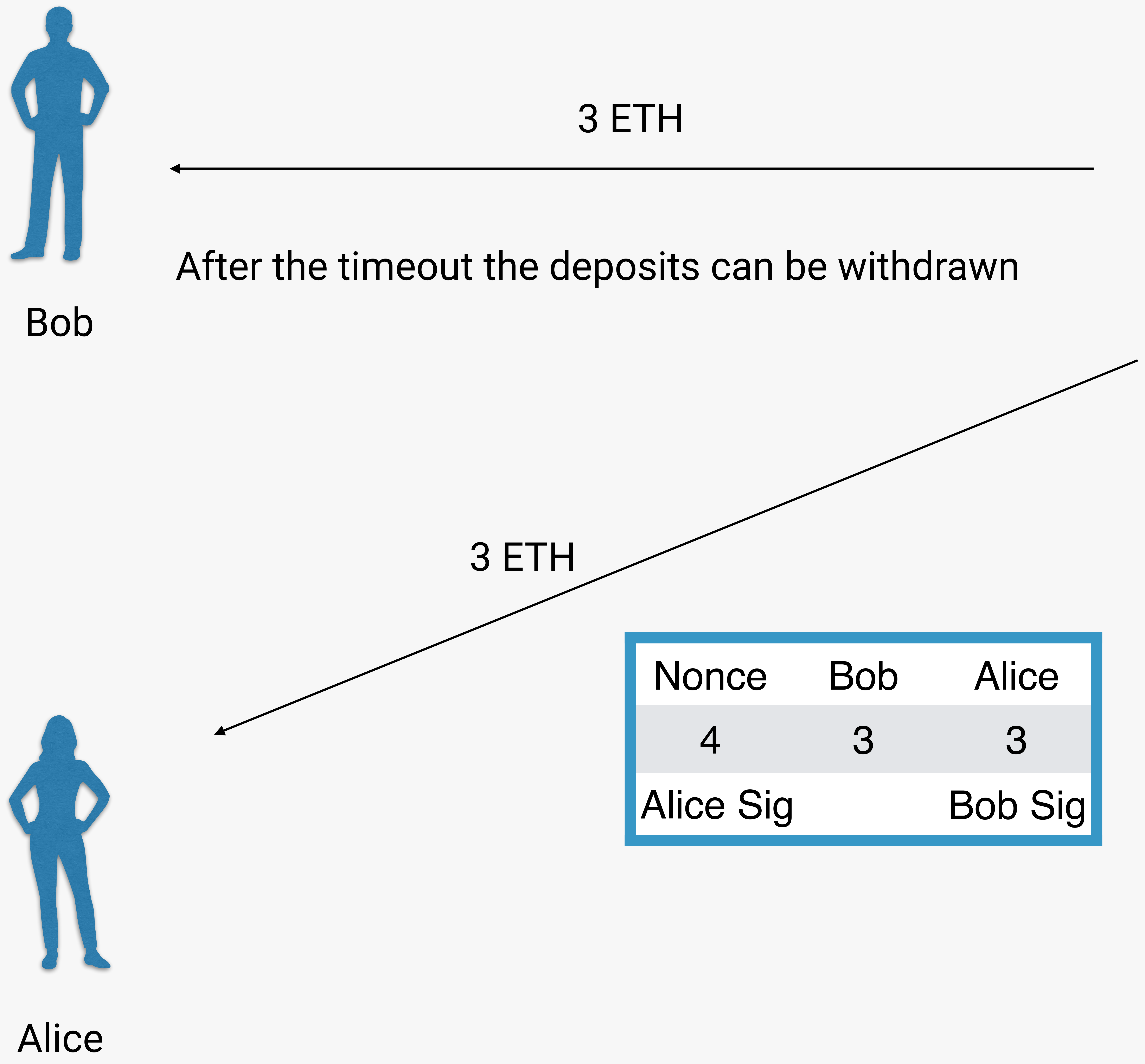
Some implementations support instant withdrawal



Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig



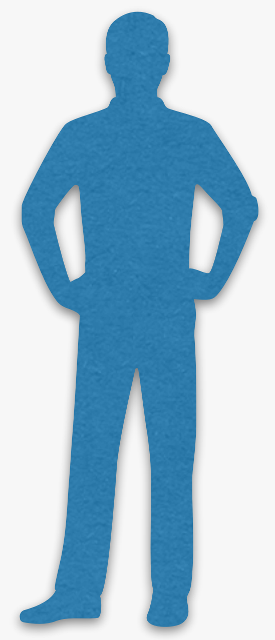
Channel

0 Ether
Owners: Bob, Alice
Closed

Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

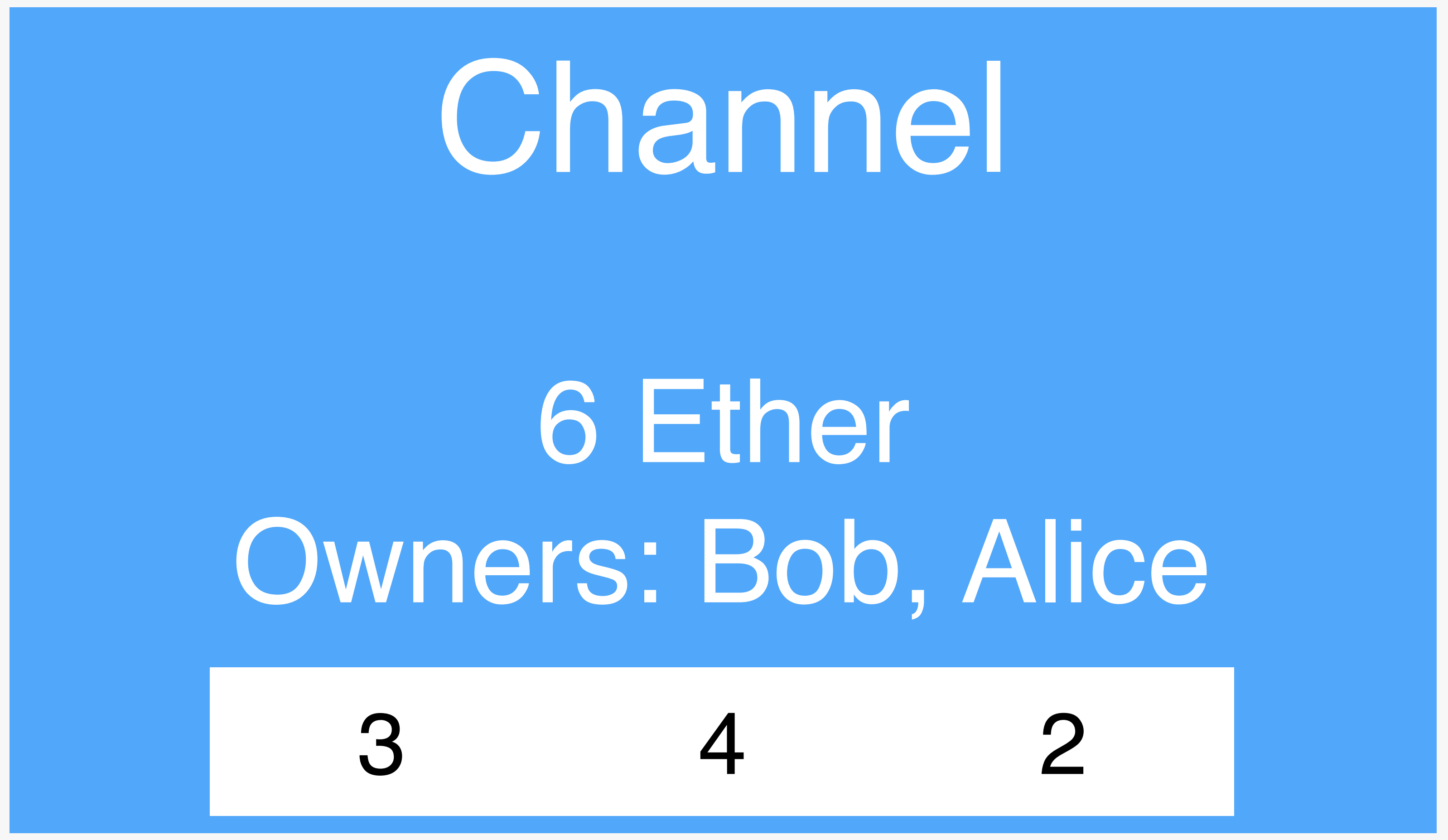
Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig



Bob wants to close the channel (**Cheating Scenario**)

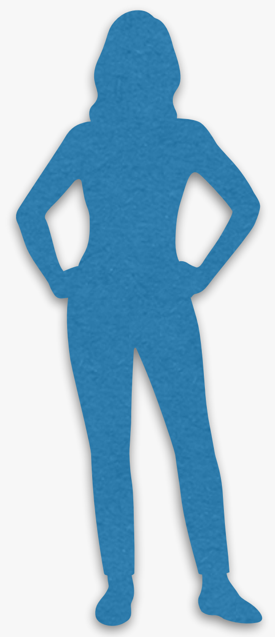
- State with nonce 3 gets submitted
- Timeout begins

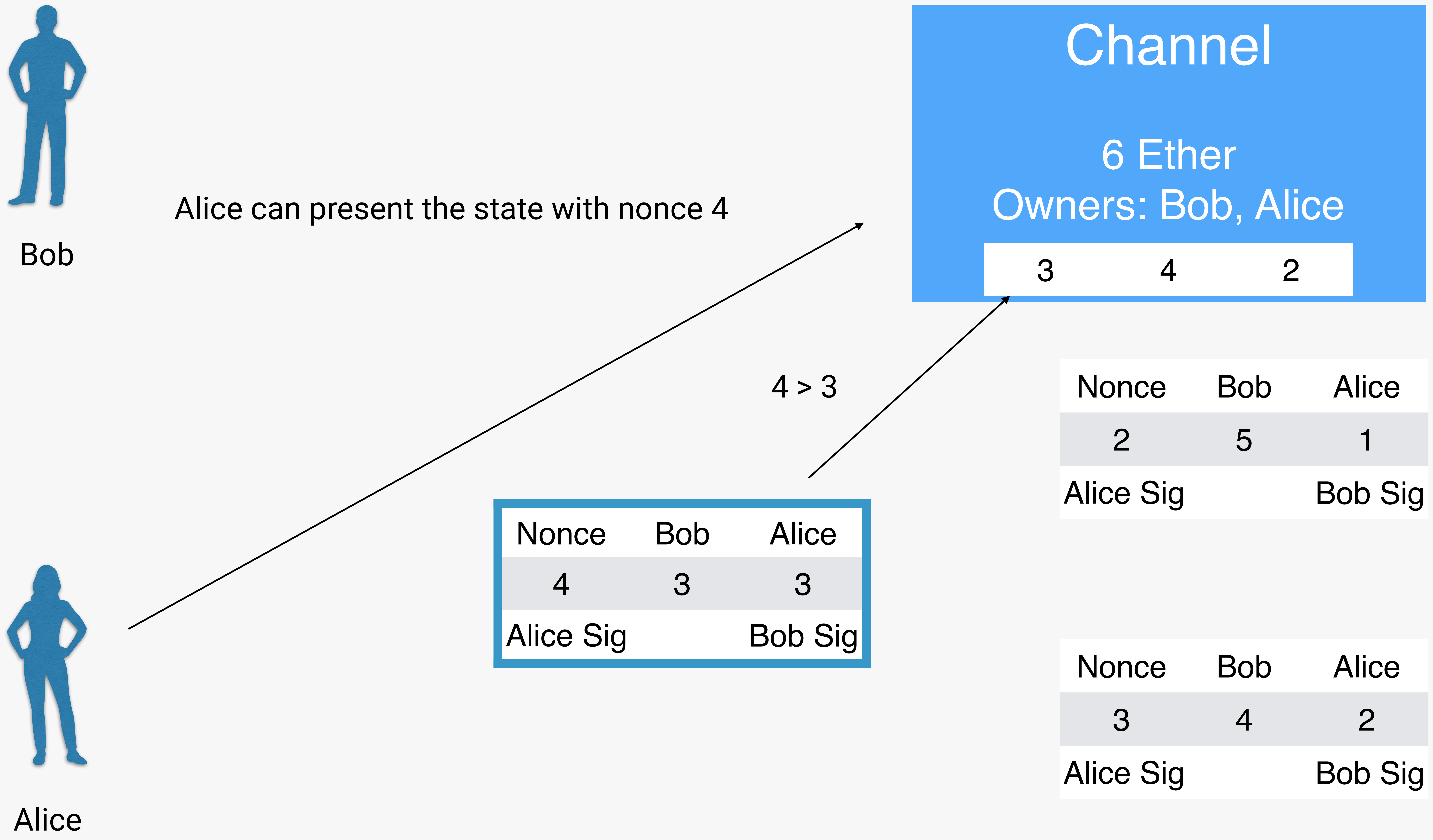


Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig

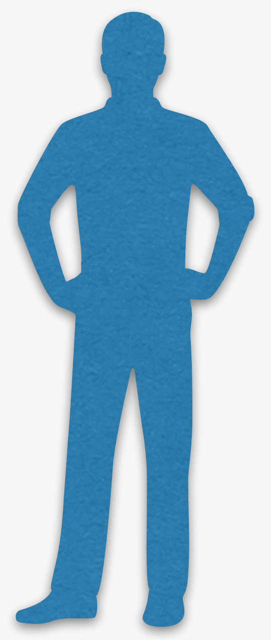




Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig


Bob

Channel

6 Ether
Owners: Bob, Alice

4	3	3

In many implementations Bob would be penalised

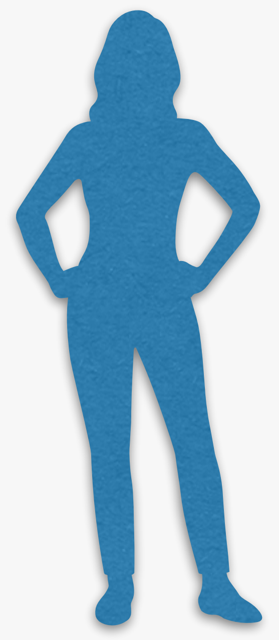
Here we cannot differentiate between:

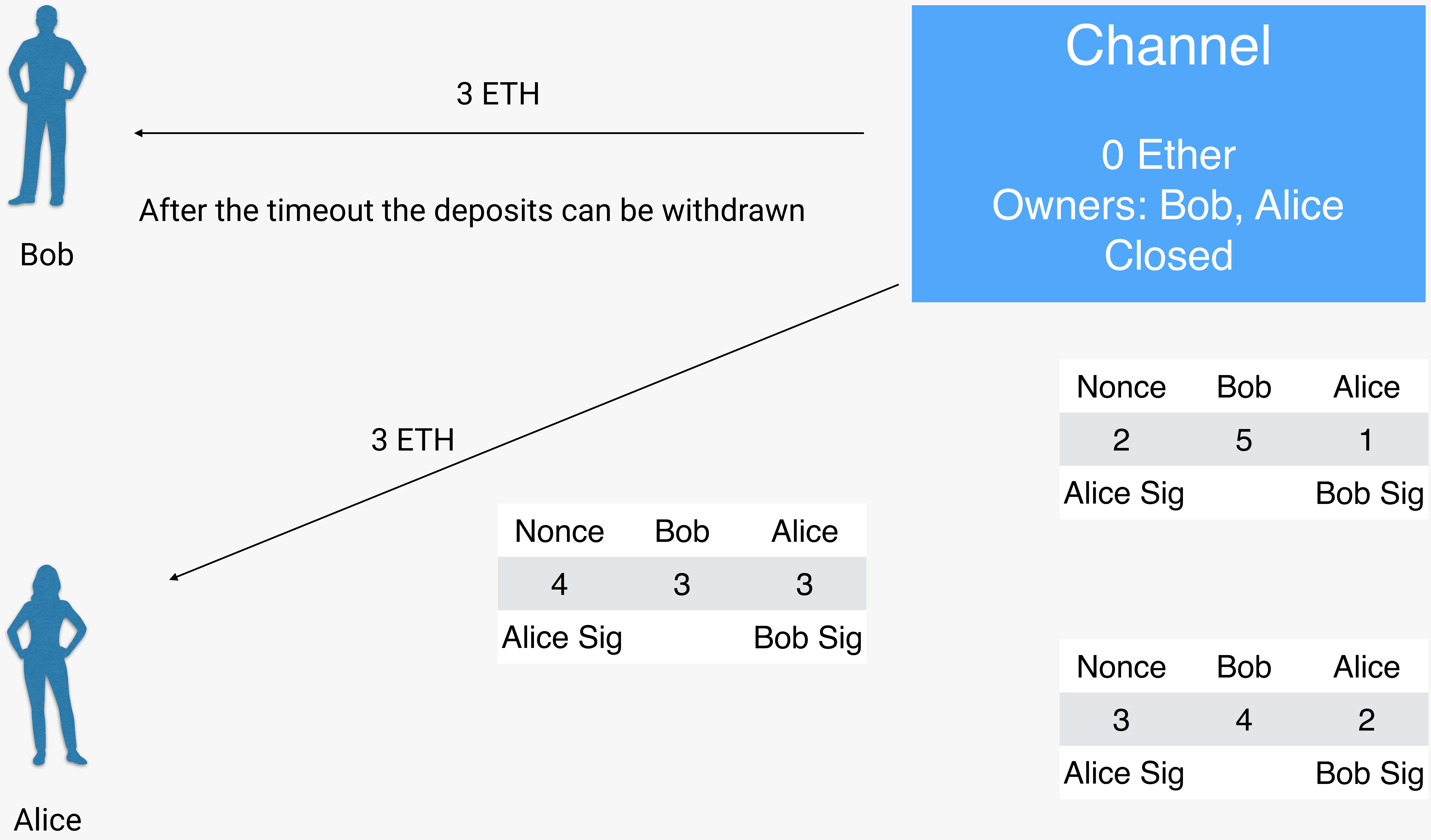
- Bob attempted to cheat
- Alice refused to sign state #4 after Bob already signed it

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig


Alice



After the timeout the deposits can be withdrawn

Nonce	Bob	Alice
2	5	1
Alice Sig		Bob Sig

Nonce	Bob	Alice
4	3	3
Alice Sig		Bob Sig

Nonce	Bob	Alice
3	4	2
Alice Sig		Bob Sig

Implementations

Many different implementations of payment channels exist

Most famously **Lightning** and **Raiden**

These projects also attempt to link different channels to a network

=> Similar ideas exist for state channel networks

State Channels

State Channels

State channels operate very similar to payment channels

The basic ideas are

- Various assets (the **state deposit**) are held in a multisig (the **state deposit holder**)
- Always sign the latest state off-chain
- Escalate to chain in case of dispute
- Use cooperative withdrawal to close the channel (or simply withdraw if possible)

FunFair

Uses Fate Channels for gameplay and random number generation

Channel between player and game server

Game logic on chain in advance

In case of a dispute the result of an action can be verified on-chain

Short-lived channels end after game session

Requirements for State Channels

- Closed set of participants
- It is possible to have a transaction included within the timeouts
- It has to make economic sense to actually do disputes
- All participants need to be online for every action (to avoid stalls)
- Every participant needs to be online often enough to check for stale submits
- Participants must not lose information about (latest) signed states

Important Terminology

Given some action X, **counterfactual X** means

- X could happen on the chain (but doesn't)
- Any participant can unilaterally make X happen (on-chain)
- All participants can act as if X happened

The purpose of state channels is to do all actions counterfactually

Counterfactual Instantiation

Counterfactual Instantiation

Counterfactual instantiation of a contract means

- Instead of actually deploying the contract, a commitment is signed
- Contract only ever gets deployed in case of a dispute

Once deployed the state deposit holder can delegate call into the contract

This allows changing the state channel code arbitrarily after its already opened

Counterfactual Addressing

Problem: Contract addresses depend on the deployer and its nonce

Presently state channel systems with CI use a global **registry contract**

- Deploys all counterfactually instantiated contracts
- Assigns to them a **counterfactual address** not depending on the nonce
- Other components look up the real address during dispute
- **Skinny CREATE2** coming in Constantinople will solve this problem

Counterfactual Instantiation

The dispute resolution mechanism can also be counterfactually instantiated

=> a simple multisig is the only thing that needs to appear on chain

The basic requirements are

- unanimous consent
- support for DELEGATECALL

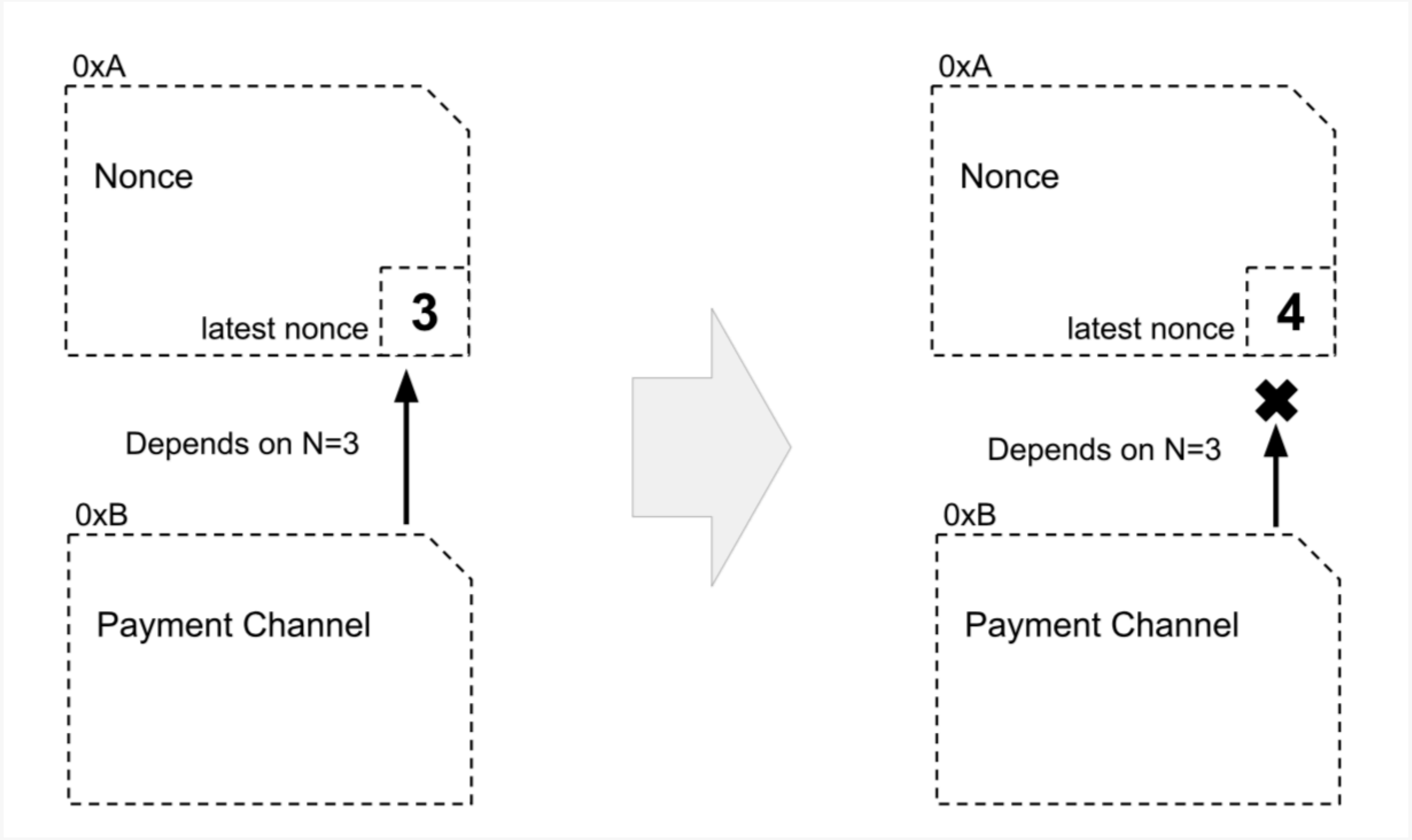
This is the approach Counterfactual is taking

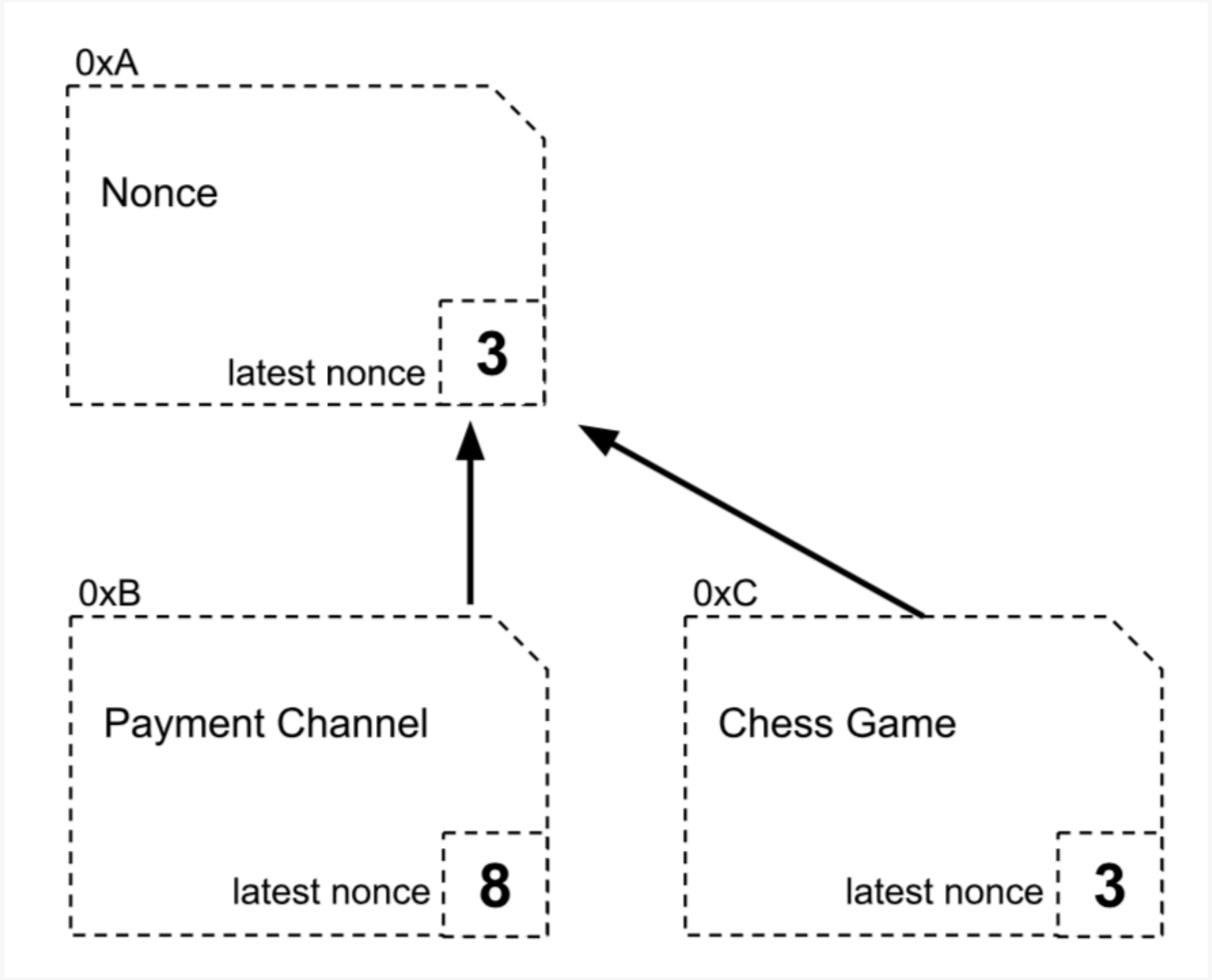
Counterfactual

A (L4) state channel consists of many counterfactual objects

Those are counterfactually instantiated contracts which

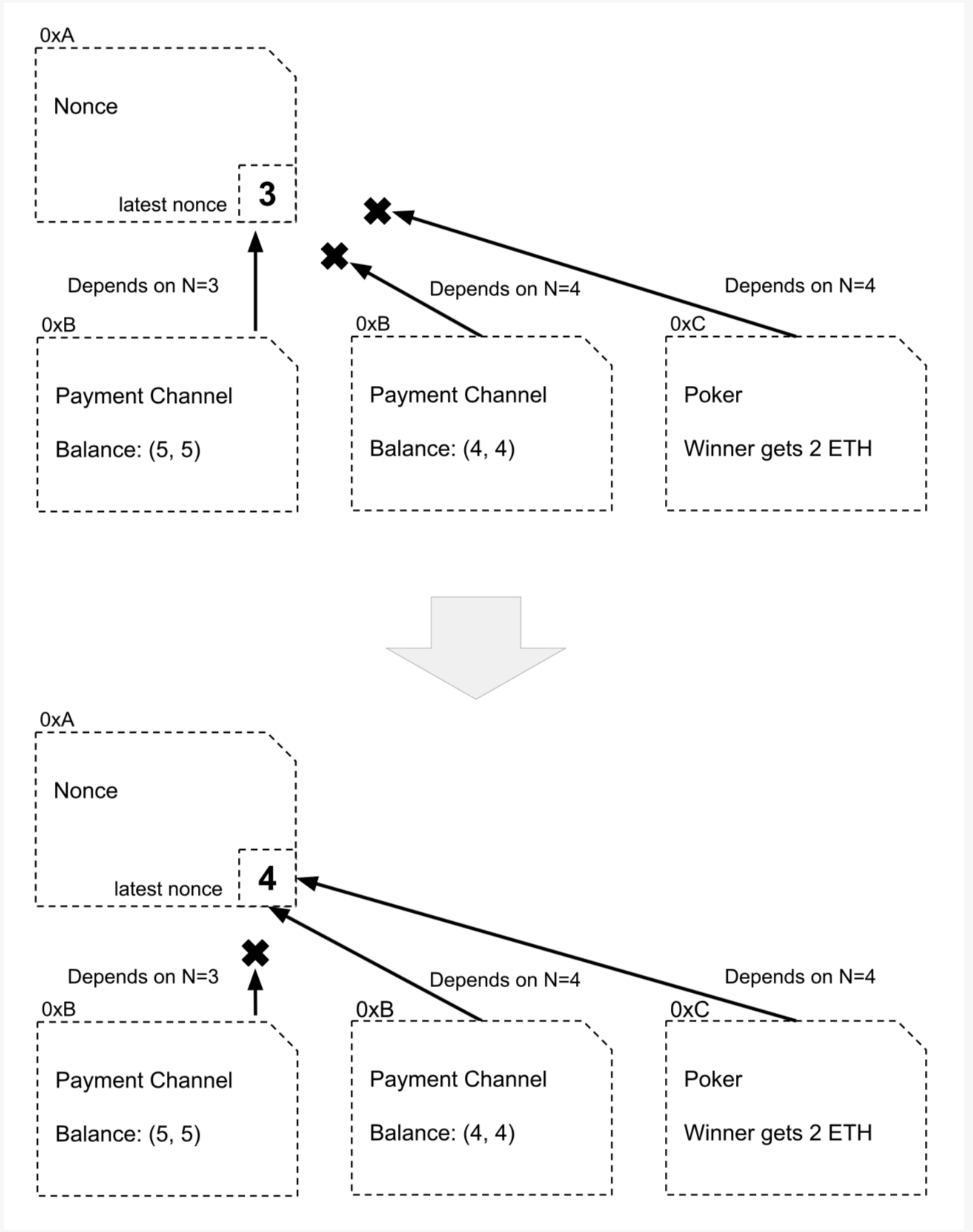
- have their own nonces
- can finalise depending on other counterfactual objects

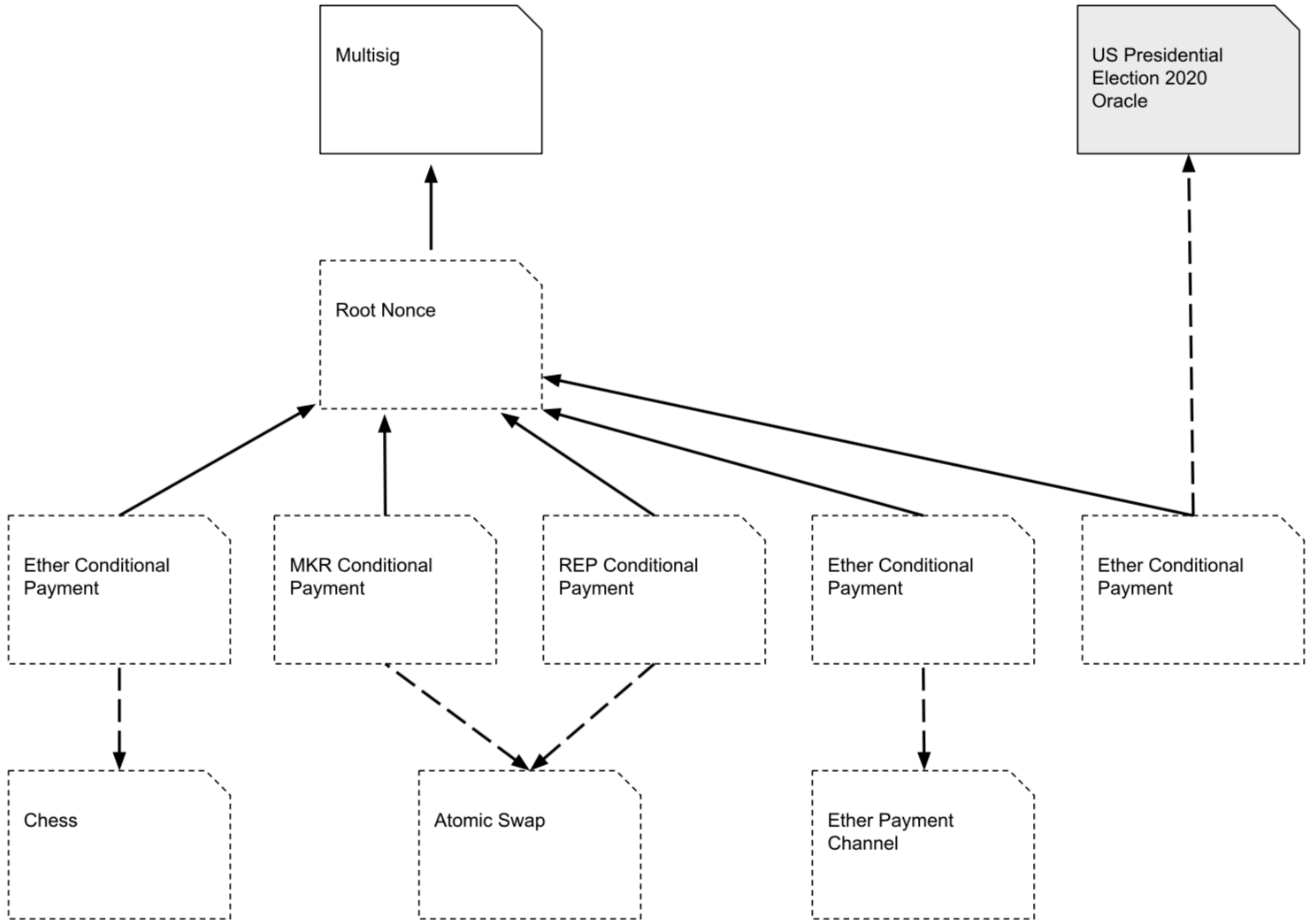

0xA
Nonce
latest nonce
3
Depends on N=3
0xB
Payment Channel
0xA
Nonce
latest nonce
4
Depends on N=3
0xB
Payment Channel

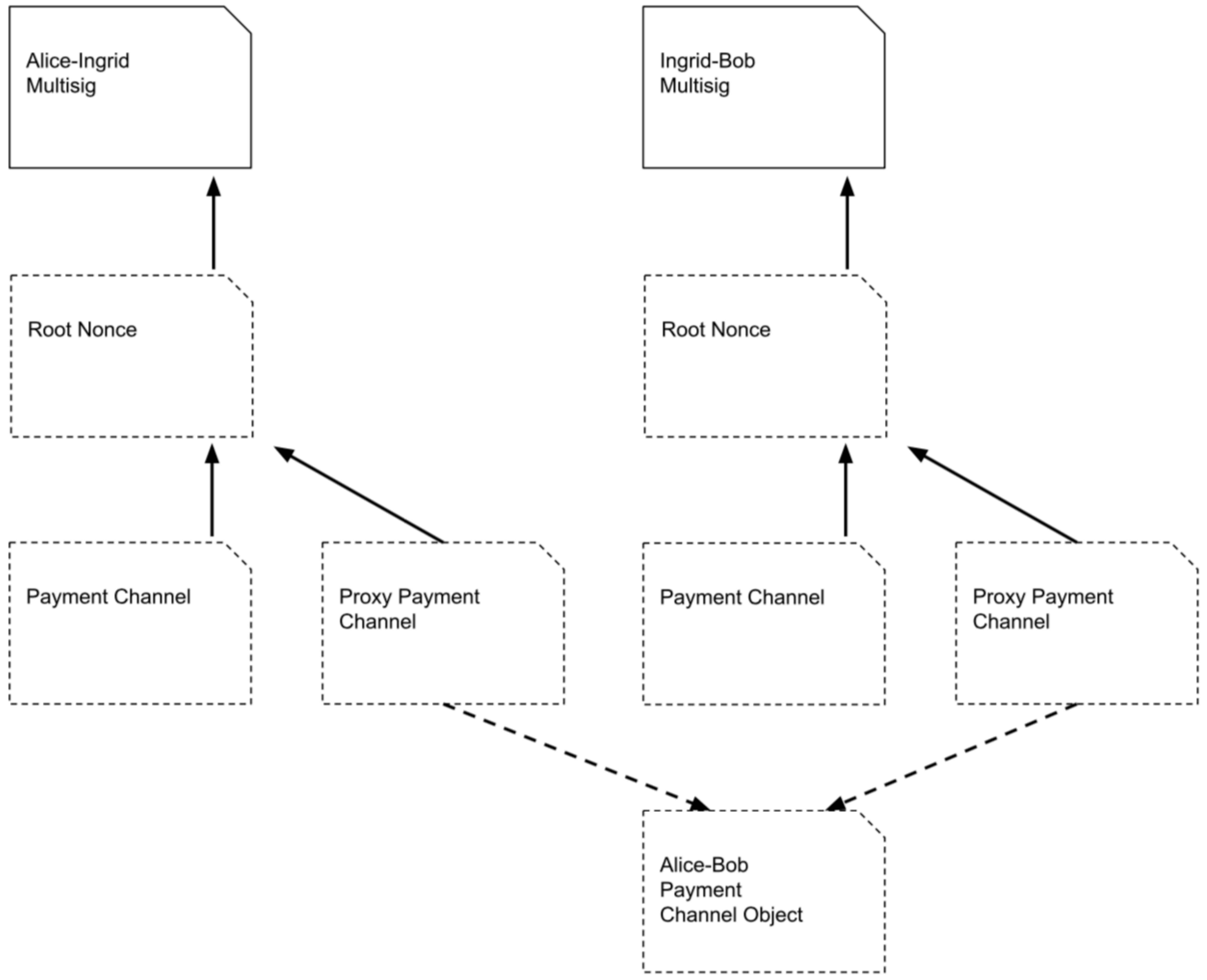

0xA
Nonce
latest nonce
3
0xB
Payment Channel
latest nonce
8
0xC
Chess Game
latest nonce
3

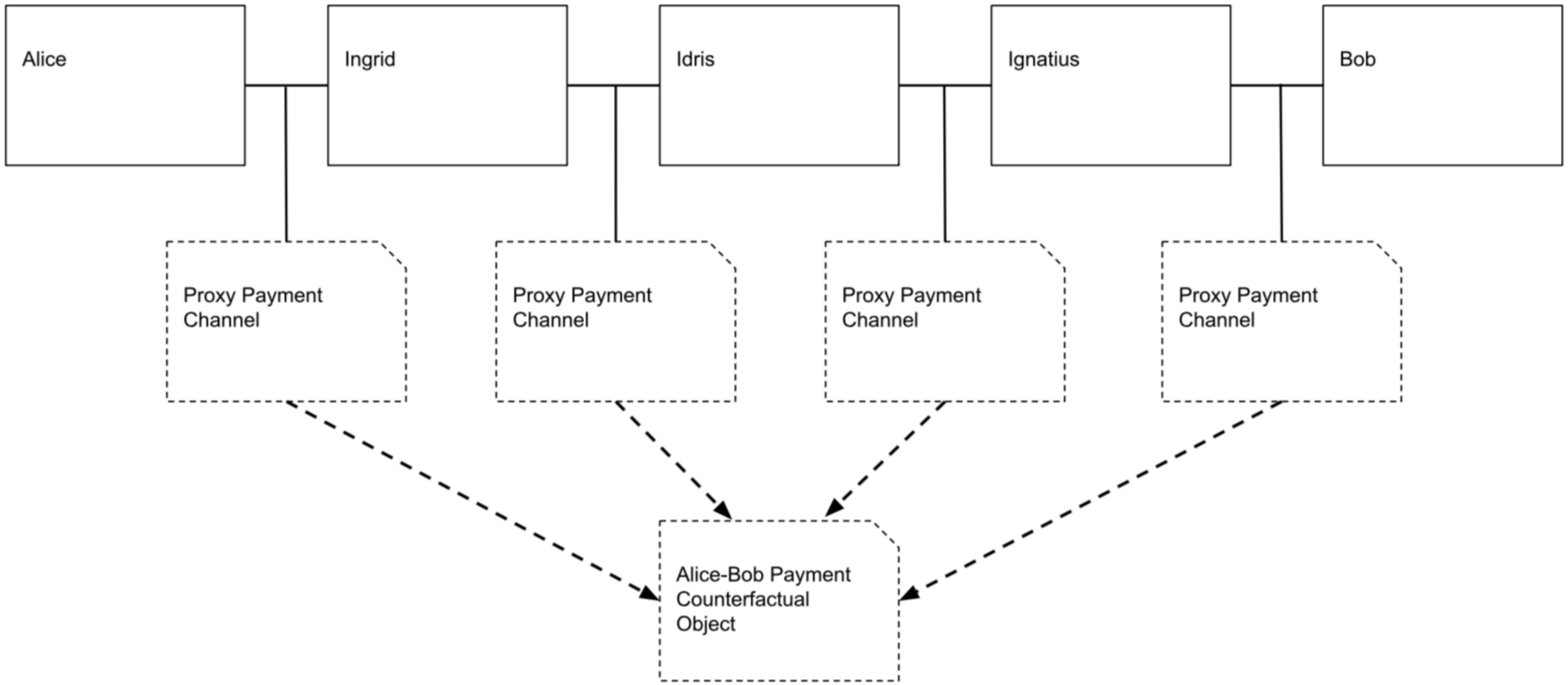
This dependency mechanism allows for atomic state transitions:

- In the beginning only 0xB(5,5) is valid
- Then 0xB(4,4) and 0xC are signed (but cannot be finalised)
- Once 0xA updates those can finalise but 0xB can not anymore



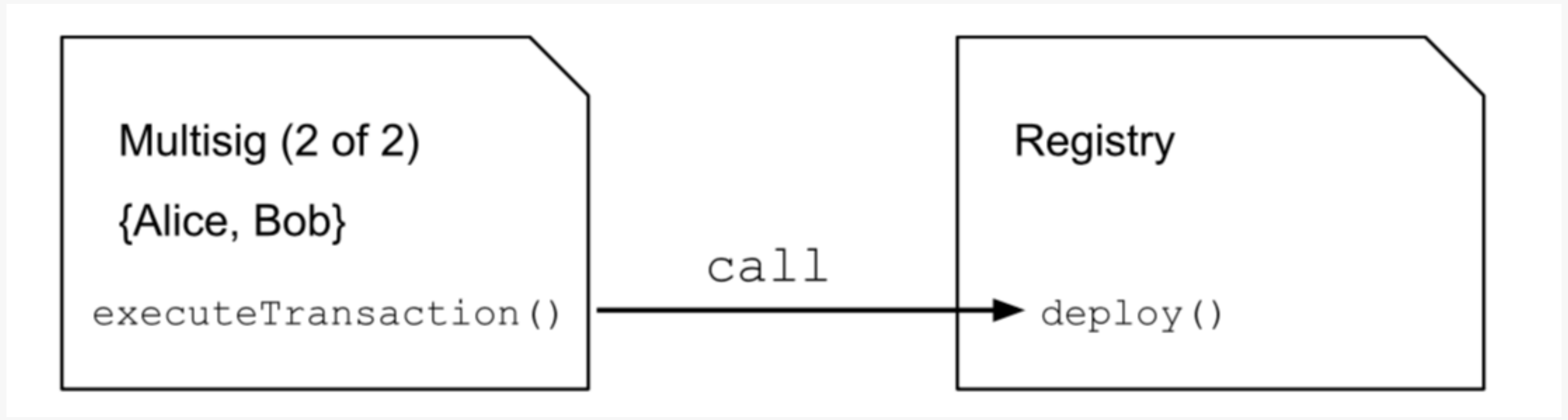

Multisig
US Presidential Election 2020 Oracle
Root Nonce
Ether Conditional Payment
MKR Conditional Payment
REP Conditional Payment
Ether Conditional Payment
Ether Conditional Payment
Chess
Atomic Swap
Ether Payment Channel


Alice-Ingrid Multisig
Ingrid-Bob Multisig
Root Nonce
Root Nonce
Payment Channel
Proxy Payment Channel
Payment Channel
Proxy Payment Channel
Alice-Bob Payment Channel Object


Alice
Ingrid
Idris
Ignatius
Bob
Proxy Payment Channel
Proxy Payment Channel
Proxy Payment Channel
Proxy Payment Channel
Alice-Bob Payment Counterfactual Object

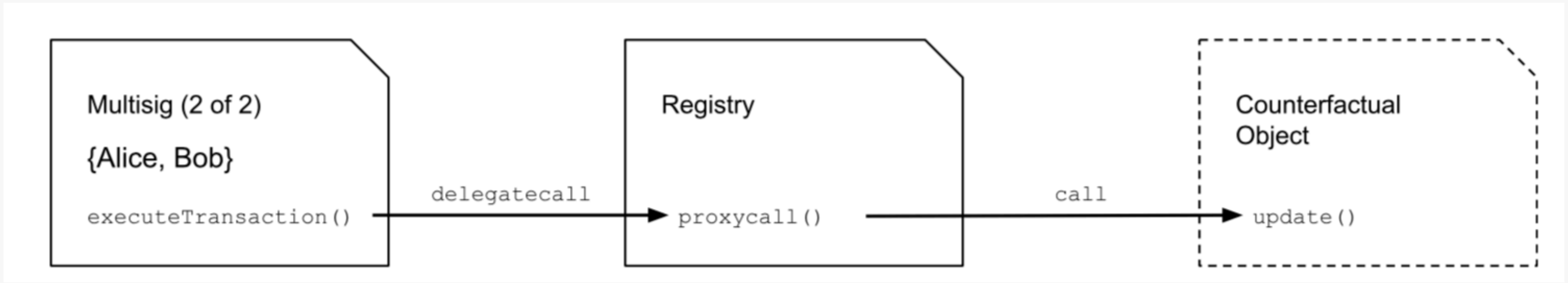
```
Instantiate(multisig, C)

calldata := ABIEncode("deploy(bytes)", [C.dbytecode])
cfaddress := keccak256(C.dbytecode, [multisig.address])
commitment := (CALL, registry, calldata, 0)
SignCommitment(multisig, commitment)
return cfaddress
```



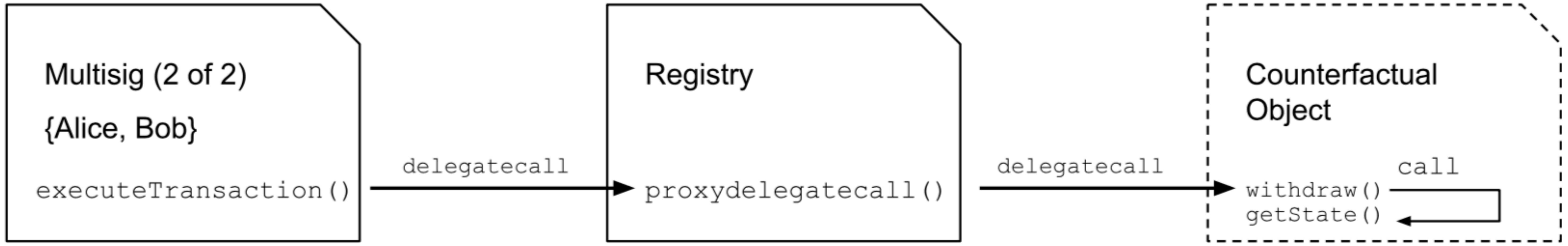
```
Update(multisig, cfAddr, newstate)

C := lookup(cfAddr)
calldata1 := ABIEncode(C.fnSignatures.update, C.id + newstate)
calldata2 := ABIEncode("proxycall(bytes)", calldata1)
commitment := (DELEGATECALL, registry, calldata2)
SignCommitment(multisig, commitment)
```



```
CommitWithdrawal(multisig, cfAddr)

calldata1 := ABIEncode("withdraw(address, bytes32)", [registry, cfAddr])
calldata2 := ABIEncode("proxydelegatecall(address, bytes32, bytes)", [
    registry, cfAddr, calldata1])
SignCommitment(multisig, (DELEGATECALL, registry, calldata2))
```



Implementation

The current implementation is based on the paper but differs in some way

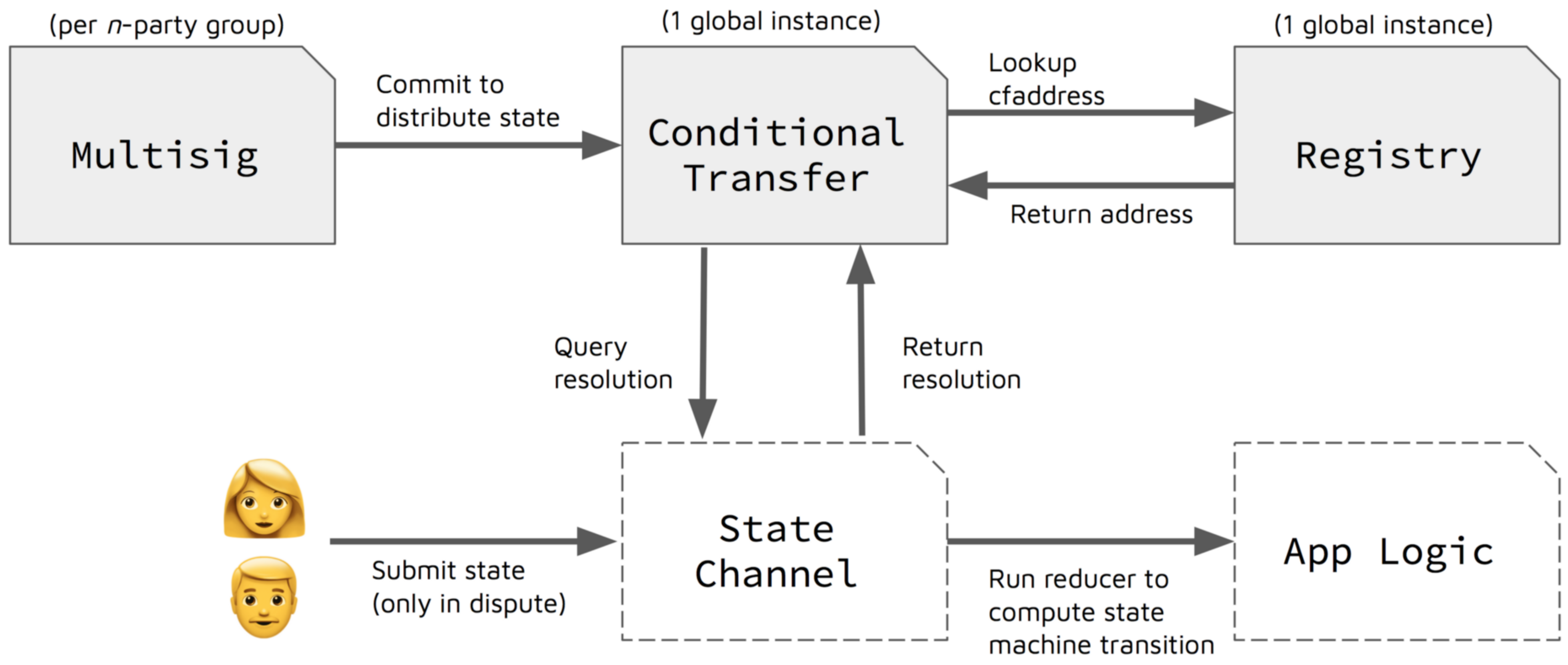
It features

- a global registry for counterfactual addresses
- (also) a global registry for the nonces and their finalisation status
- clean separation from the channel and the application logic

Implementation

Applications are modelled as

- stateless state transition functions (state, action) => state
- turn based interaction (action) => player
  - the turn taker can continue the state transition on-chain in case of a dispute
- apps return a transfer command to the app channel
  - can be about transferring ETH, Tokens, or (not) arbitrary calls


(per *n*-party group)
Multisig
Commit to distribute state
(1 global instance)
Conditional Transfer
Lookup cfaddress
Return address
(1 global instance)
Registry
Query resolution
Return resolution
Submit state (only in dispute)
State Channel
Run reducer to compute state machine transition
App Logic

Alternative Implementations

Perun

Perun is another project working on generalised state channels

Separates states into substates which can be updated independently

Supports **virtual channels** for non-interactive payment routing

(Supposedly) does not support counterfactual instantiation

Much more detailed paper available

Spankchain

Spankchain were (one of?) the first to actually show real code

Supports counterfactual instantiation

Not as “abstracted” as the Counterfactual implementation

Code used to be quite messy

No activity in 3 months

But they use their simpler payment channels in production

Trinity

“Universal Off-chain Scaling Solution”

Pushed into ETH main net a few days ago

Seems to focus on payments and privacy

Seems to be primarily intended for EOS

Has its own token TNC

What is ready?

For payment channels

- µRaiden
- Various other systems

For generalised state channels

- Nothing that’s not in prototype status
- Perun and Counterfactual have at least a lot of the concepts worked out

Reading List

State channels for dummies: https://medium.com/@eolszewski

L4 Counterfactual Paper

Perun Paper(s)

All informations about our events at

https://www.meetup.com/Ethereum-Vienna

All available slides and materials at

https://github.com/ethereum-vienna-meetup