August 1st 2017

Scaling Blockchain

Bitcoin Austria
Ethereum Vienna

sponsored by bitpanda

bitpanda

Bitpanda – Europas führende Plattform für digitale Währungen

- Vollautomatisierte Handelsplattform für digitale Währungen mit eigenen Wallet-Lösungen für Bitcoin, Ethereum, Litecoin und Dash
- Industry leader im Bereich KYC
- Starkes, stetiges Wachstum seit der Gründung 2014
- Größtes Blockchain-Team in Europa mit 20 Mitarbeitern mit 10 weiteren ausgeschriebenen Positionen (bitpanda.com/career)

300,000+

User

50,000+ monatl. Neuanmeldungen

€ 25 Mio.+

Umsatz

20-30% monatl. Wachstum

bitpanda to go

Ab sofort exklusiv bei der Post

Ethereum| mit Bargeld kaufen. Einfach, sicher & günstig.

Bequem in über 400 Postfilialen und bei rund 1300 Postpartnern in ganz Österreich kaufen und in Sekundenschnelle in Bitcoin, Ethereum, Dash oder Litecoin einlösen.

Code einlösen Filiale finden

Founders & Co-CEOs

Contact Details

Eric Demuth
eric@bitpanda.com

Paul Klanschek
paul@bitpanda.com

Agenda

Scaling Bitcoin

Doge Cakes

Scaling Ethereum

Socialising

bitcoin austria

62. Meetup

von Bitcoin Austria

Bitcoin Scaling

Where are we **now**

What do we want

How can we get it

Lessons learned from the **past**

Fixes

Community matters

Where are we now

A decentralized distributed network with

- Blocks of 1MB
- On average every 10 minutes
- Gives us ~1500 Transactions per Block
- Average 260,000 Transactions per day (3 tps)
- Peak: 369,089 transactions per day (May 14th)
- Cost per Transaction (USD) $29.20
  - (Fees + Coinbase = Block Reward) / transactions

What do we want?

Less Fees

- Normal Banks have about 1$ additional transaction costs
  - Plus maintenance/bank charge
- Bitcoin Transaction fees ~2.40$
  - But no additional costs

More Speed

- More transactions or faster transactions?
  - No one cares as long as his transaction is included.
  - Instant acceptance of Transactions (for payment processors)

How can we get it

No easy solution.
- Every solution has consequences!
- Every solution includes Risks

Raise the limit
- Deploy Hardfork
- Relieve fee pressure
- Increases transaction time of the Block
  - Therefore increases the likelihood of small forks
    - No uncle block mechanism like ethereum

How can we get it

No easy solution.

- Every solution has consequences!
- Every solution includes Risks

Improve the data structure of the protocol (Segwit)

- Fix Bugs
  - Enable things that should have been possible all along
- Enable more specific behaviour
  - Faster verification (Payment channels)
  - More security for SPV clients

Lessons learned from the past

Where did it start?

Satoshi decides to introduce a limit. Because he thinks that the network is too small and possibly could not handle many transactions.

Commit: `a30b56eb`

fix openssl linkage problems,

```
disable minimize to tray
on Linux because it has
too many problems
including a CPU peg bug
```

```
static const unsigned int
MAX_BLOCK_SIZE = 1000000;
```

```
```

Lessons learned from the past

Upgrades can fail

Version 0.7 vs 0.8

Database changed
(and removed a specific Bug)

Users were upgrade lazy
A new Block triggered the Bug
The users "rejected" the 0.8 chain

Miners had to downgrade

...[D]atabases sometimes have errors which cause them to fail to return records, or to return stale data. And if those exist consistency must be maintained; and "fixing" the bug can cause a divergence in consensus state that could open users up to theft.

Greg Maxwell

Lessons learned from the past

New idea:

Signal protocol upgrades
Activate behaviour conditionally
Wait for enough miners to upgrade
How many miners should signal?

What about 95%?

New Problem:

Miners abuse the mechanism to block features

How to fork

Hard-fork

this what Ethereum does

Works great

Soft-fork

this is what Bitcoin does

Has minimal incentive for users to upgrade

Users run old software

Community matters

The Community decides (in a more traditional way) how Bitcoin evolves.

The Community are

- Miners
- Users
- Exchanges
- Payment Processors

Community is complicated

Bitcoin Cash

Bitcoin Unlimited

Bitcoin ABC

Bitcoin The New York Agreement

Bitcoin ...

Danke für Ihre Aufmerksamkeit!


bitcoin
austria


bitcoin
austria

ethereum vienna

Updates

Parity Multisig Contract Hack

On July 19th

- Bug in parity "enhanced" multisig contract
- 30m$ ETH taken from SwarmCity, Aeternity and Edgeless
- WHG drained the other 150m$ of ETH and ERC-20 tokens

Cause: Sensitive function had **no** security measures whatsoever

Standard multisig wallet not affected

As of yesterday all rescued funds have been returned!

Metropolis

Will probably be split into two parts

Part 1: everything but EIP-86

Part 2: EIP-86

Probably in September

Current block time: **~19-20s**

Projected block time: **45s** before Metropolis

ethereum

vienna

Scaling Ethereum

(The Road to 2.0: Scalability)

Ethereum 2.0


Casper
Abstractions
Scalability
April 2016
September 2016
Now

Ethereum Today

Only "scaling" mechanism today: blockGasLimit (more on that later)

Currently ~6.712.392

Standard Transactions ~21.000

=> ~17 tx/s at current block time if every tx is standard

Currently ~2.5 tx/s on average

Ethereum Today

Current blockchain size: ~8.7gb

But now growing at almost 0.05gb / day (~18gb/y)

receipts + recent states appear to be ~8gbs

state snapshots < 6gb

all states ever ~250gb

two types of scaling

Onchain

- Increase transaction throughput
- Independent shards with async communication
- Very hard problem

Offchain

- Do as much as possible outside the chain
- Still maintain similar security properties

Offchain Scaling

General idea:

- Participants use protocols externals to the blockchain
- Use chain only for settlement and disputes

Some approaches:

- Payment Channels (offchain agreements)
- Channel Networks (connecting different channels together, e.g. raiden)
- State Channels (generalisation of payment channels)
- Verifiable Computation (e.g. truebit)

Payment Channels

Arbitrary number of

- ether / ERC-20 transfers
- mostly between 2 parties
- with at most 2-3 onchain transactions

can be unidirectional (is much simpler)

or bidirectional (much more useful)

Unidirectional Payment Channels

Example:

Alice needs to pay server S

- many small payments
- but all instantaneous

No need for money to go the other way => can be unidirectional

Unidirectional Payment Channels

Channel

Channel creation

Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

Channel
5 ETH

Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

Alice signs a new promise

Server: 1

Channel
5 ETH

Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

Alice signs a new promise

Alice sends promise to server

Channel
5 ETH

Server: 1

Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

Alice signs a new promise

Alice sends promise to server

Updated for every payment

Channel
5 ETH

Server: 2

Unidirectional Payment Channels

Channel creation

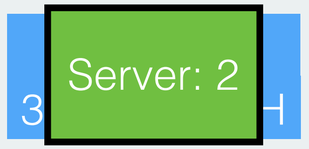
Alice locks up 5 eth

Alice signs a new promise

Alice sends promise to server

Updated for every payment

Server cashes out



Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

Alice signs a new promise

Alice sends promise to server

Updated for every payment

Server cashes out

Channel

3 ETH | 2 ETH

Unidirectional Payment Channels

Channel creation

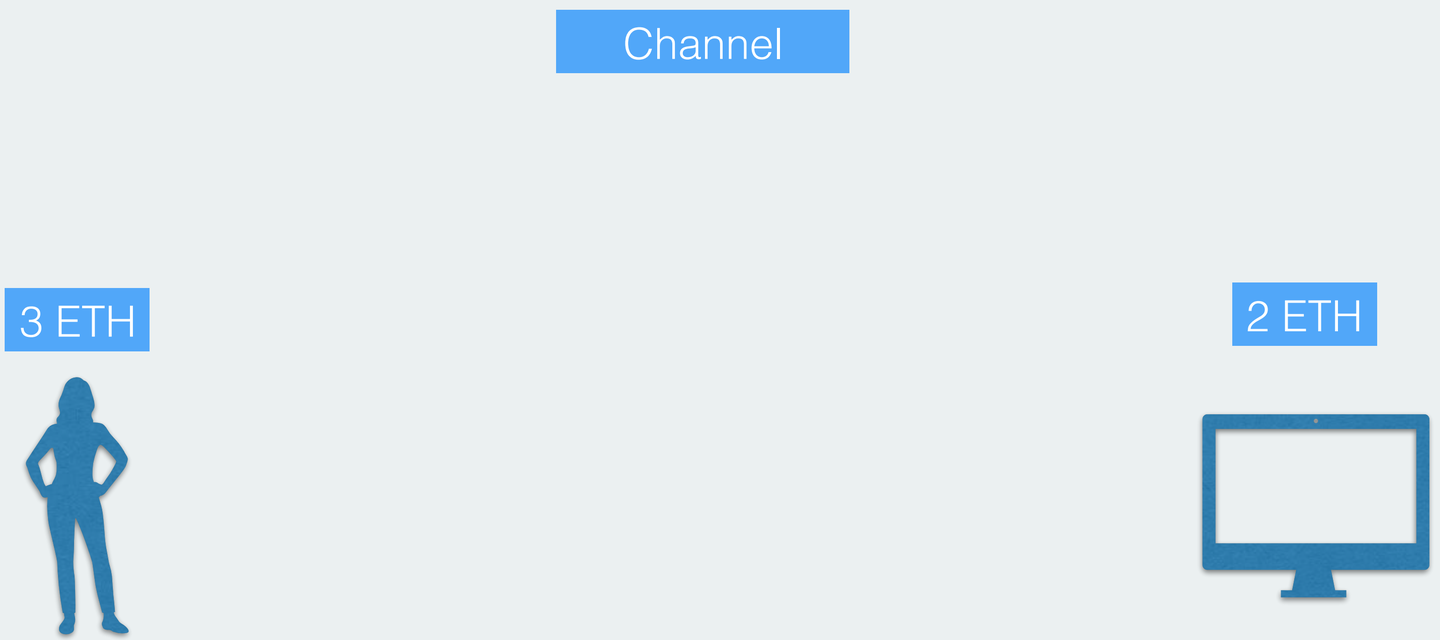
Alice locks up 5 eth

Alice signs a new promise

Alice sends promise to server

Updated for every payment

Server cashes out



Unidirectional Payment Channels

Channel creation

Alice locks up 5 eth

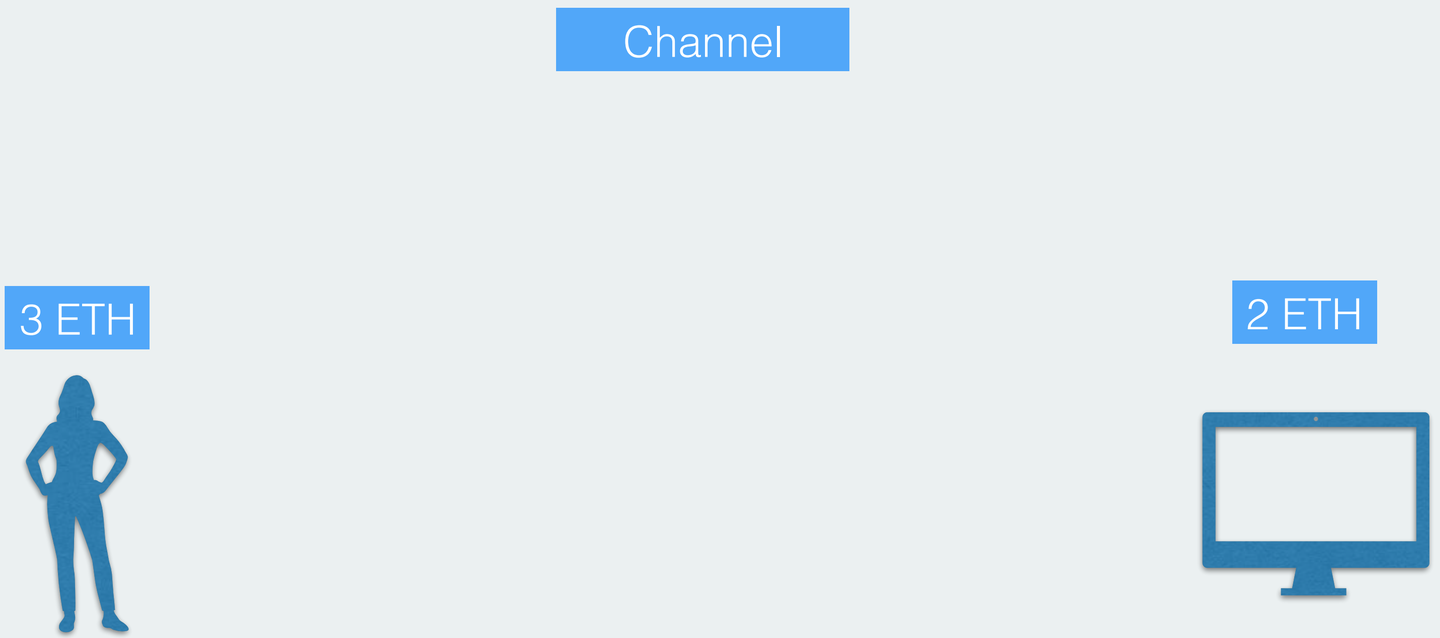
Alice signs a new promise

Alice sends promise to server

Updated for every payment

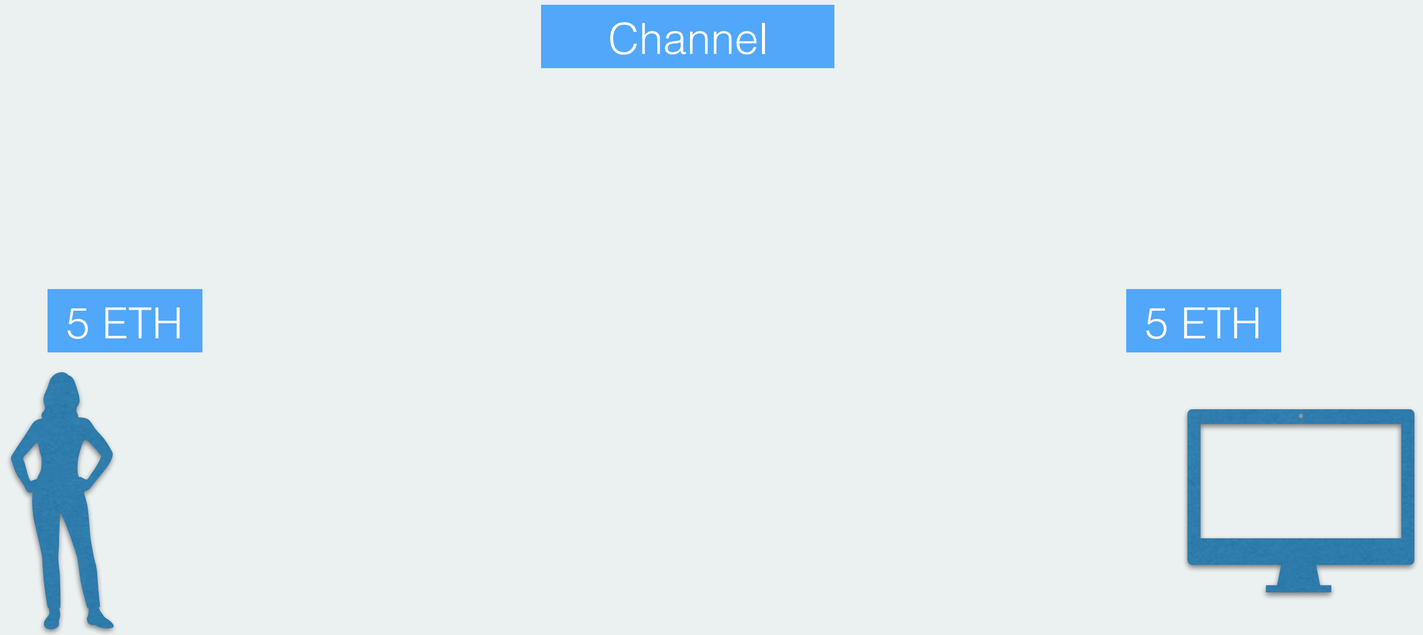
Server cashes out

Or channel timeout



Bidirectional Payment Channels

Channel creation



Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Channel
10 ETH

Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Channel
10 ETH

Server: 2

Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Alice: 1

Channel
10 ETH

Server: 2

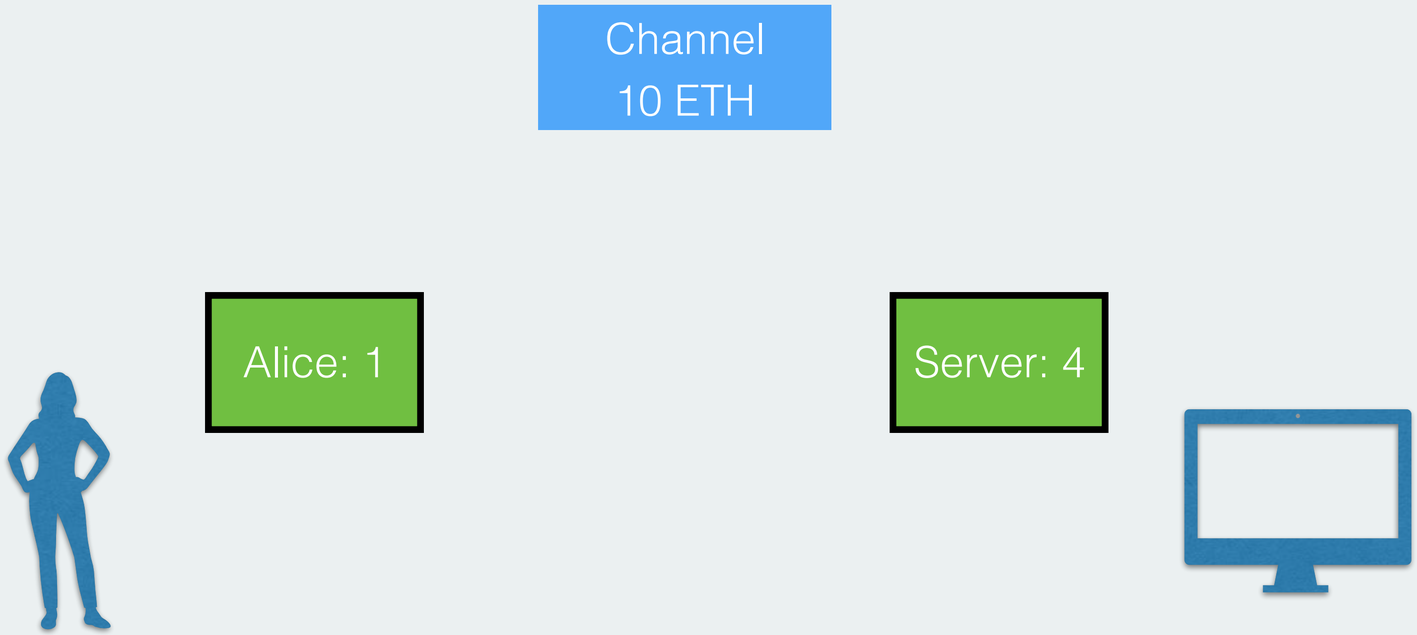
Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Updated for every payment



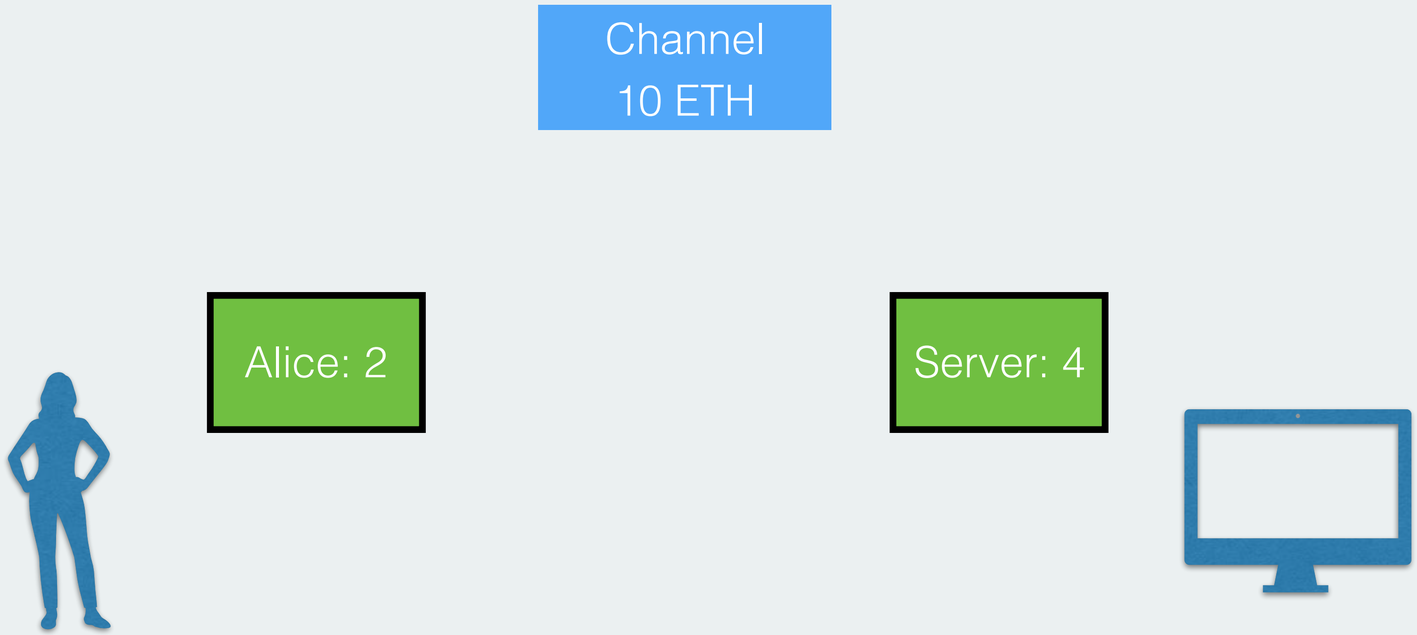
Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Updated for every payment



Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Updated for every payment

Channel closure

Alice: 2

Channel
10 ETH
Server: 9
Alice: 1

Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

Promise is signed on payment

Updated for every payment

Channel closure

Update Period

Channel
10 ETH
Server: 7
Alice: 3

Bidirectional Payment Channels

Channel creation

Both lock up 5 eth

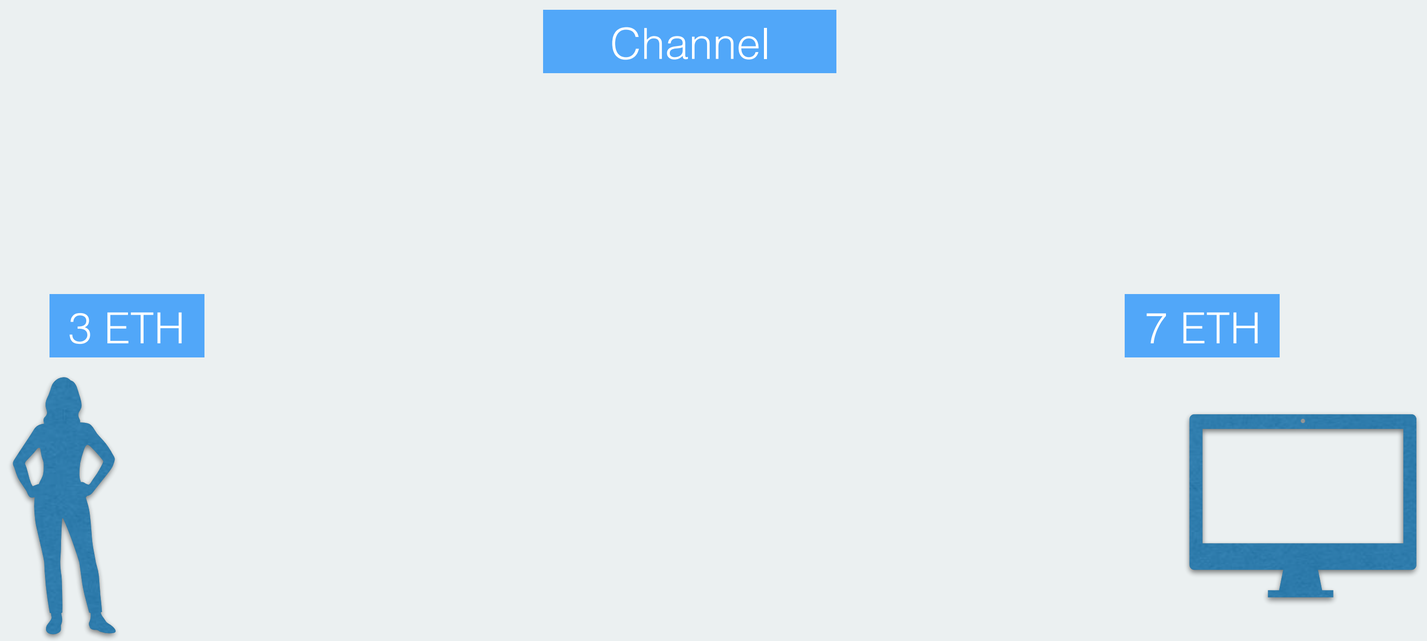
Promise is signed on payment

Updated for every payment

Channel closure

Update Period

Settlement



Bidirectional Payment Channels

Alternative implementations possible:

- Signed message always contains the latest balance
- Cheater looses everything if newer message is presented
- Can be enforced by
  - increasing nonce (classical ethereum payment channel)
  - signing follow-up transactions for old messages (lightning network)

Bidirectional Payment Channels

Problem

- channel required with every party you want to send to
- n^2 channels
- does not make sense if you send a few transaction to each party

Solution

- chain channels together
- requires payment channels with hashlocks

Hashlock

Up until now necessary condition for transfer is a signed message

A hashlock requires also requires revealing some preimage

Enables additional conditional transfers in the channel

Idea: Use the same hashlock for multiple channels

=> if lock revealed for one, it is revealed for all of them

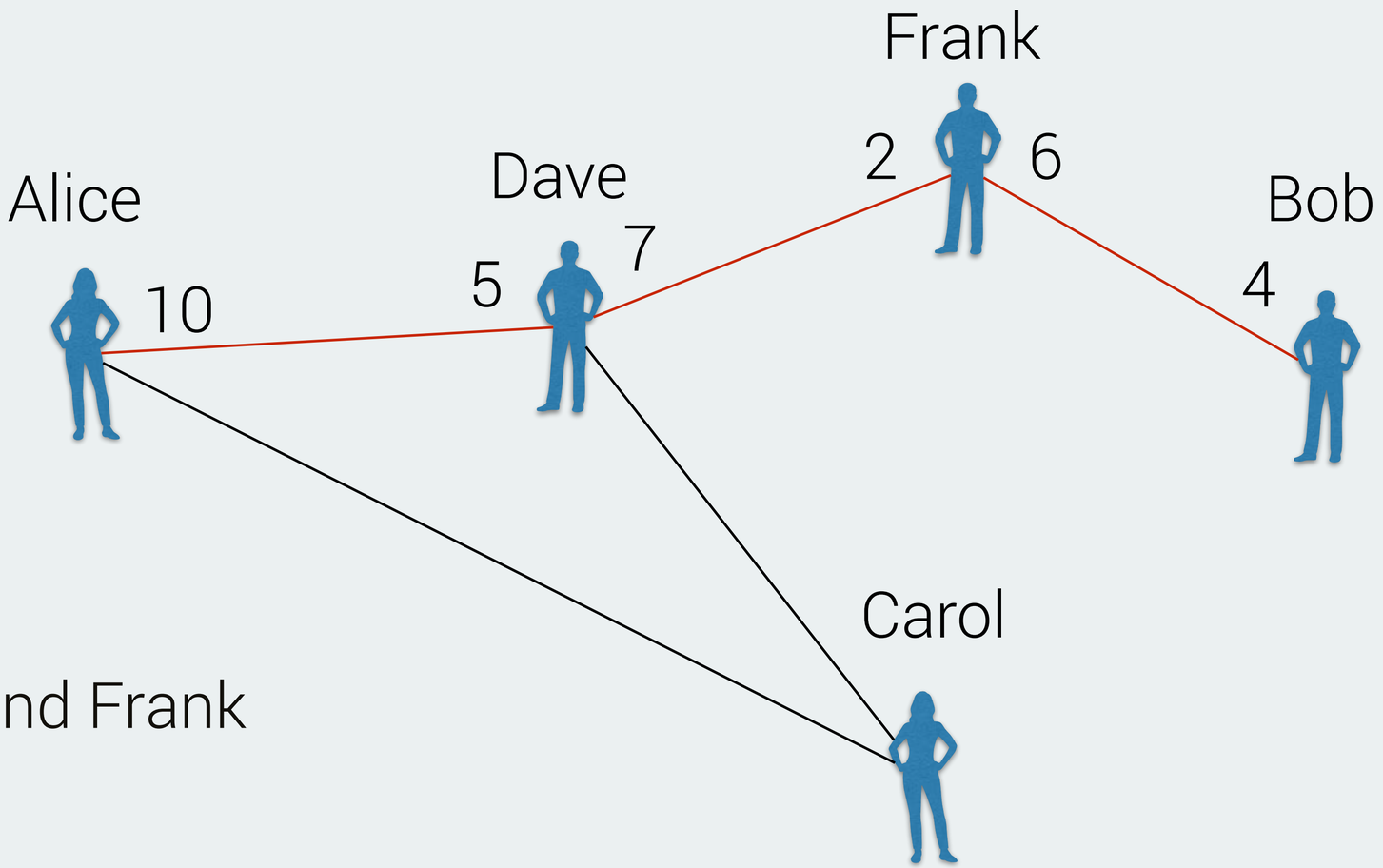
Raiden / Channel Networks

Example:

Alice -> Bob / 5 REP

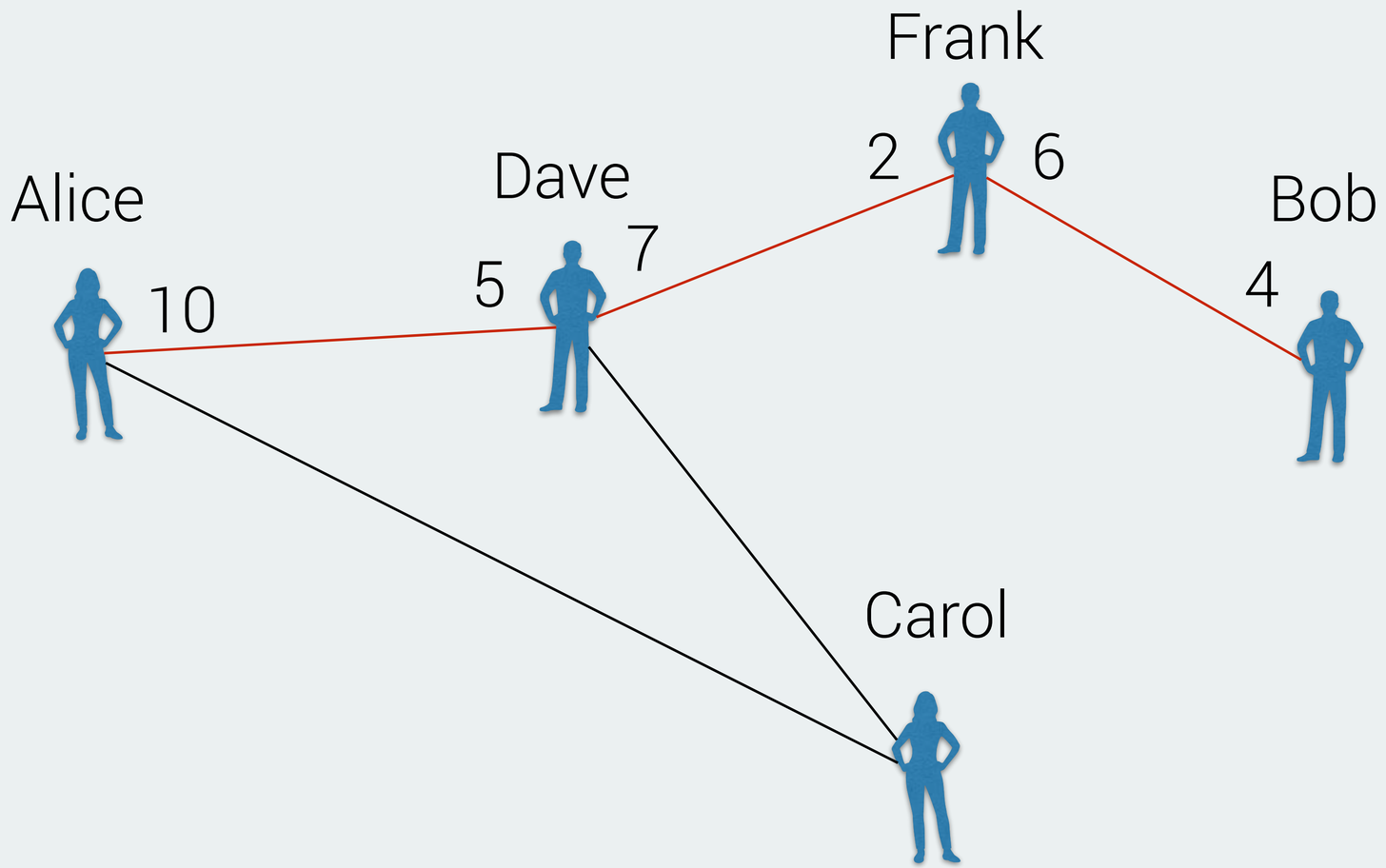
No direct connection.

Can be routed over Dave and Frank



Raiden / Channel Networks

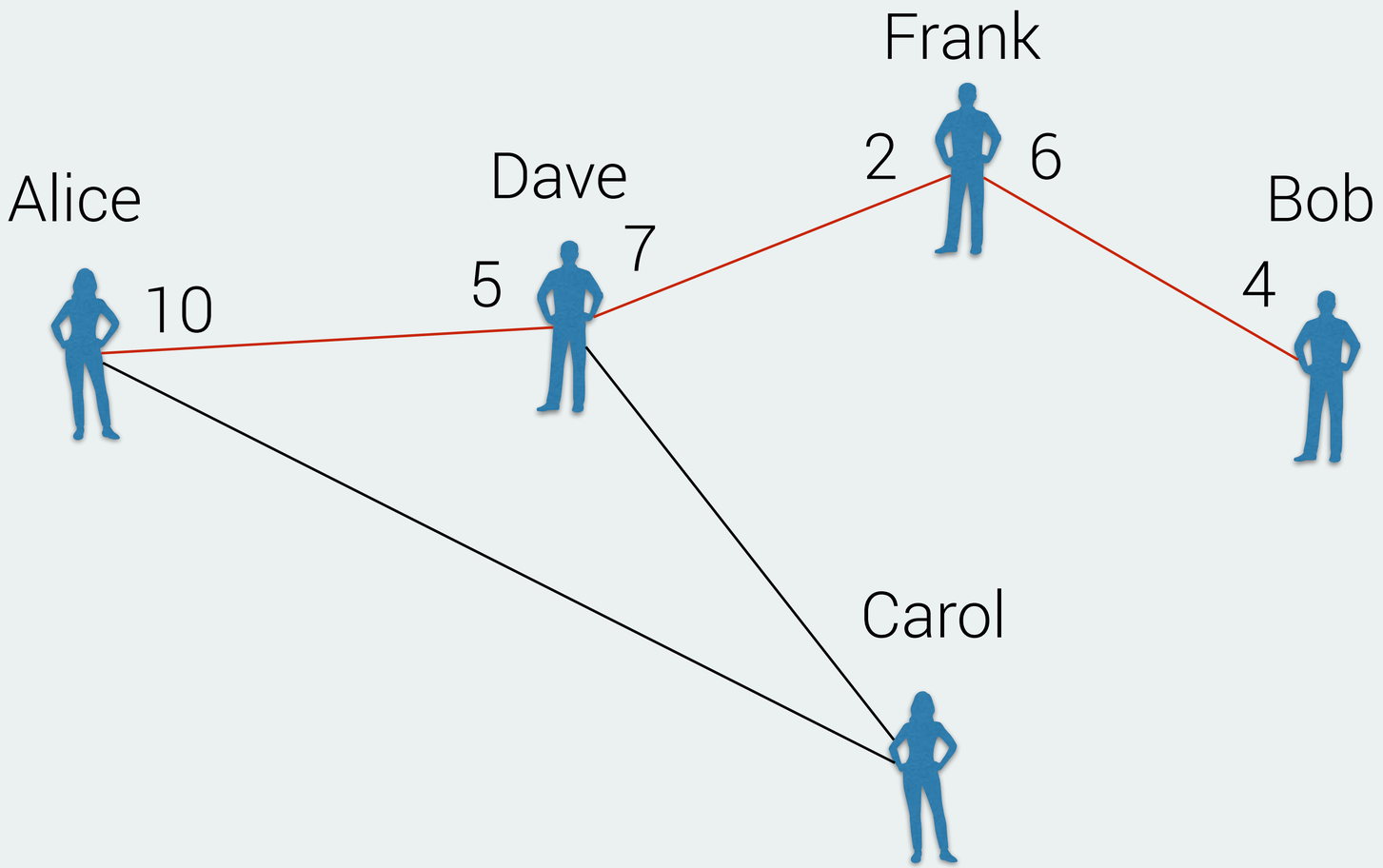
Alice generates secret s



Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

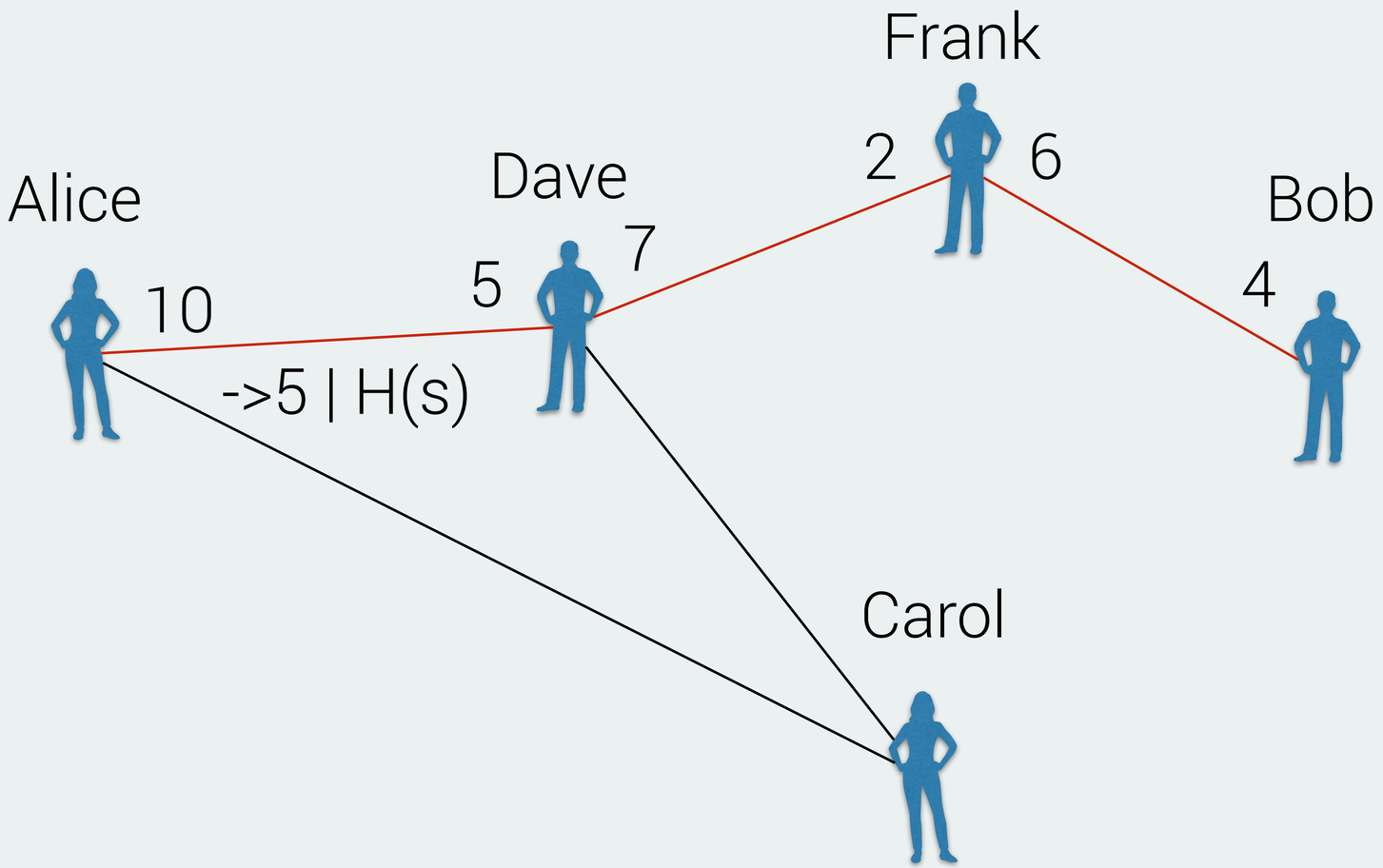


Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

Alice -> Dave / 5 REP | H(s)



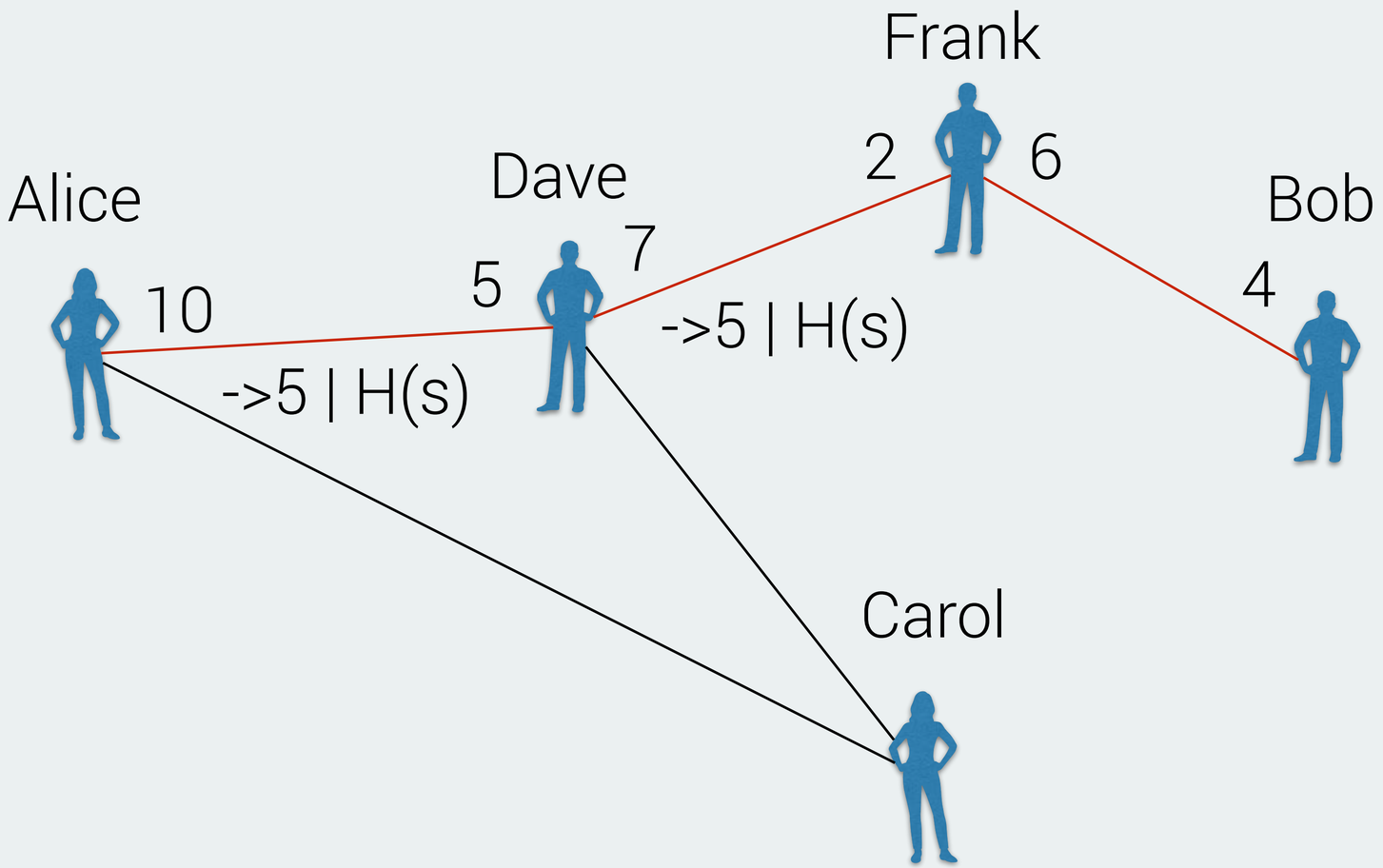
Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)



Raiden / Channel Networks

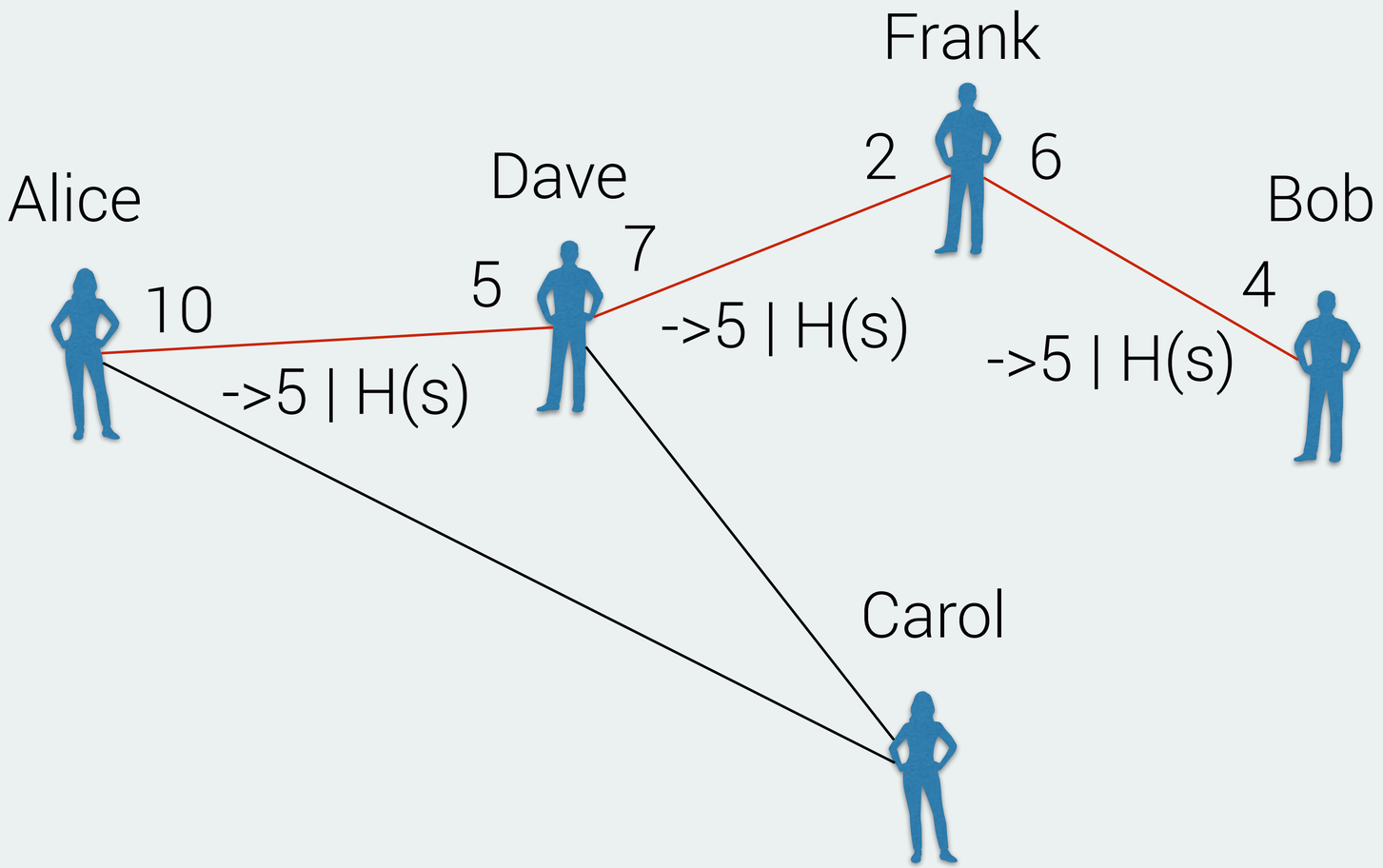
Alice generates secret s

Computes hash H(s)

Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)

Frank -> Bob / 5 REP | H(s)



Raiden / Channel Networks

Alice generates secret s

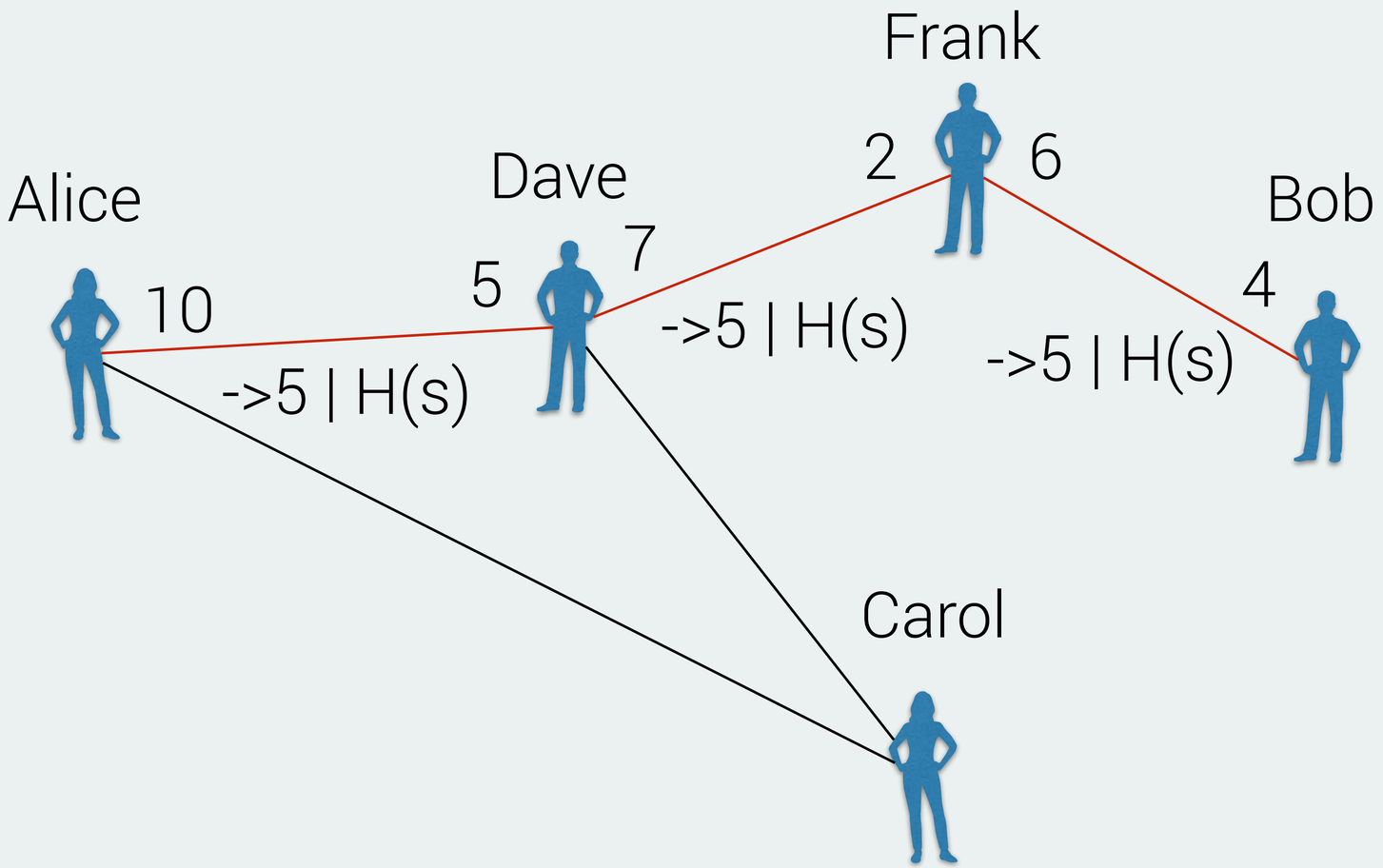
Computes hash H(s)

Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)

Frank -> Bob / 5 REP | H(s)

Alice reveals s to Bob



Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

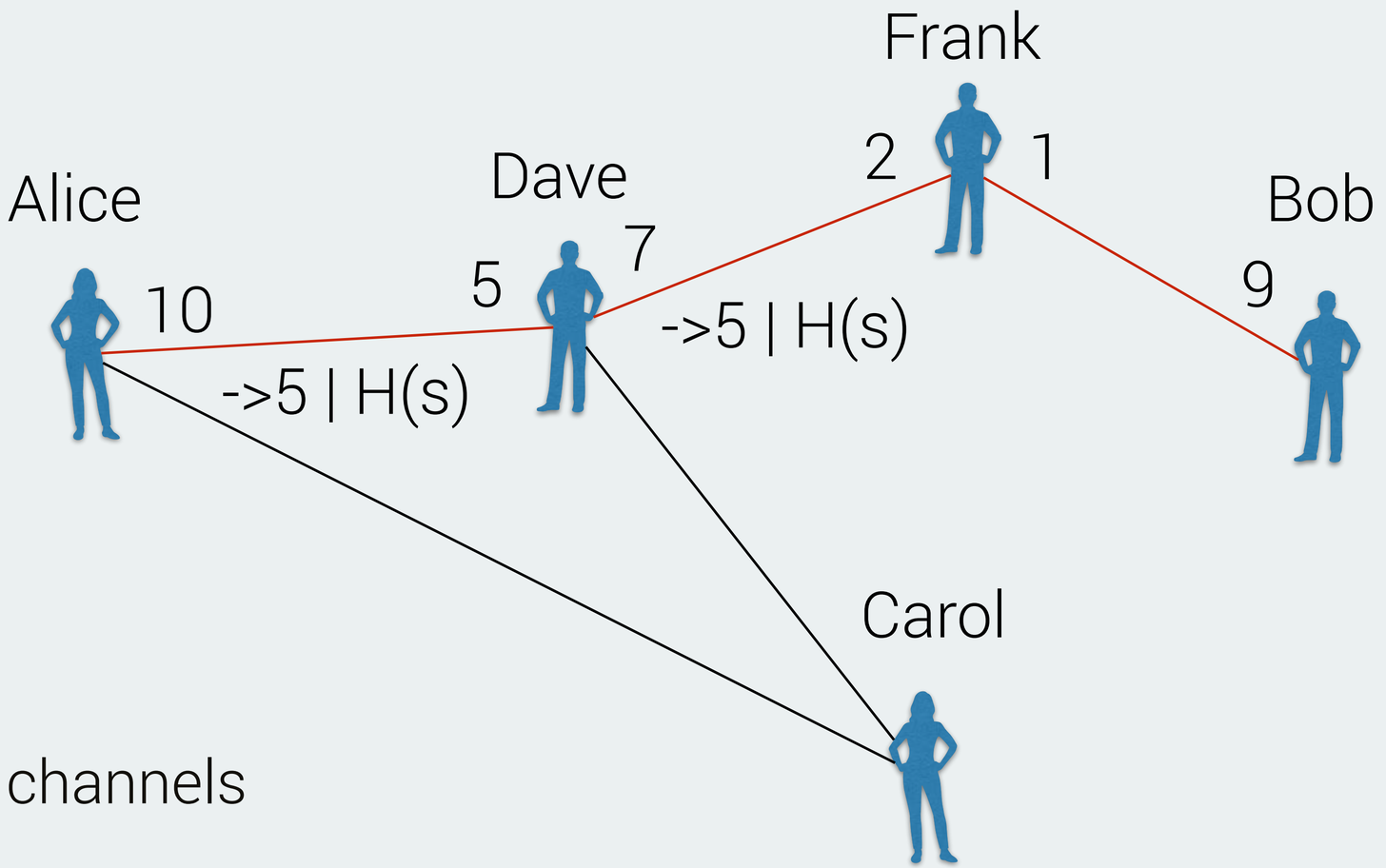
Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)

Frank -> Bob / 5 REP | H(s)

Alice reveals s to Bob

Everybody can update their channels



Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

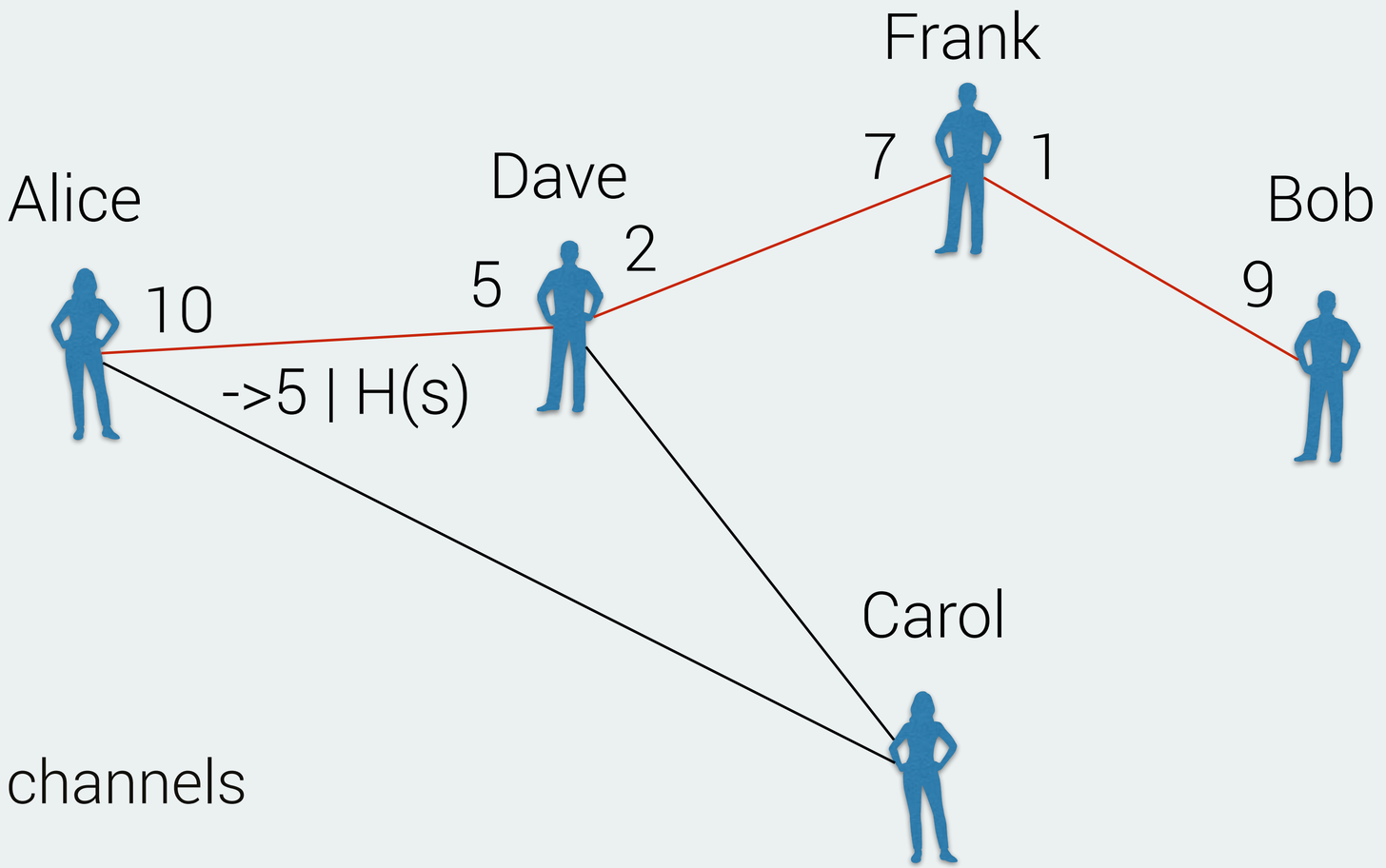
Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)

Frank -> Bob / 5 REP | H(s)

Alice reveals s to Bob

Everybody can update their channels



Raiden / Channel Networks

Alice generates secret s

Computes hash H(s)

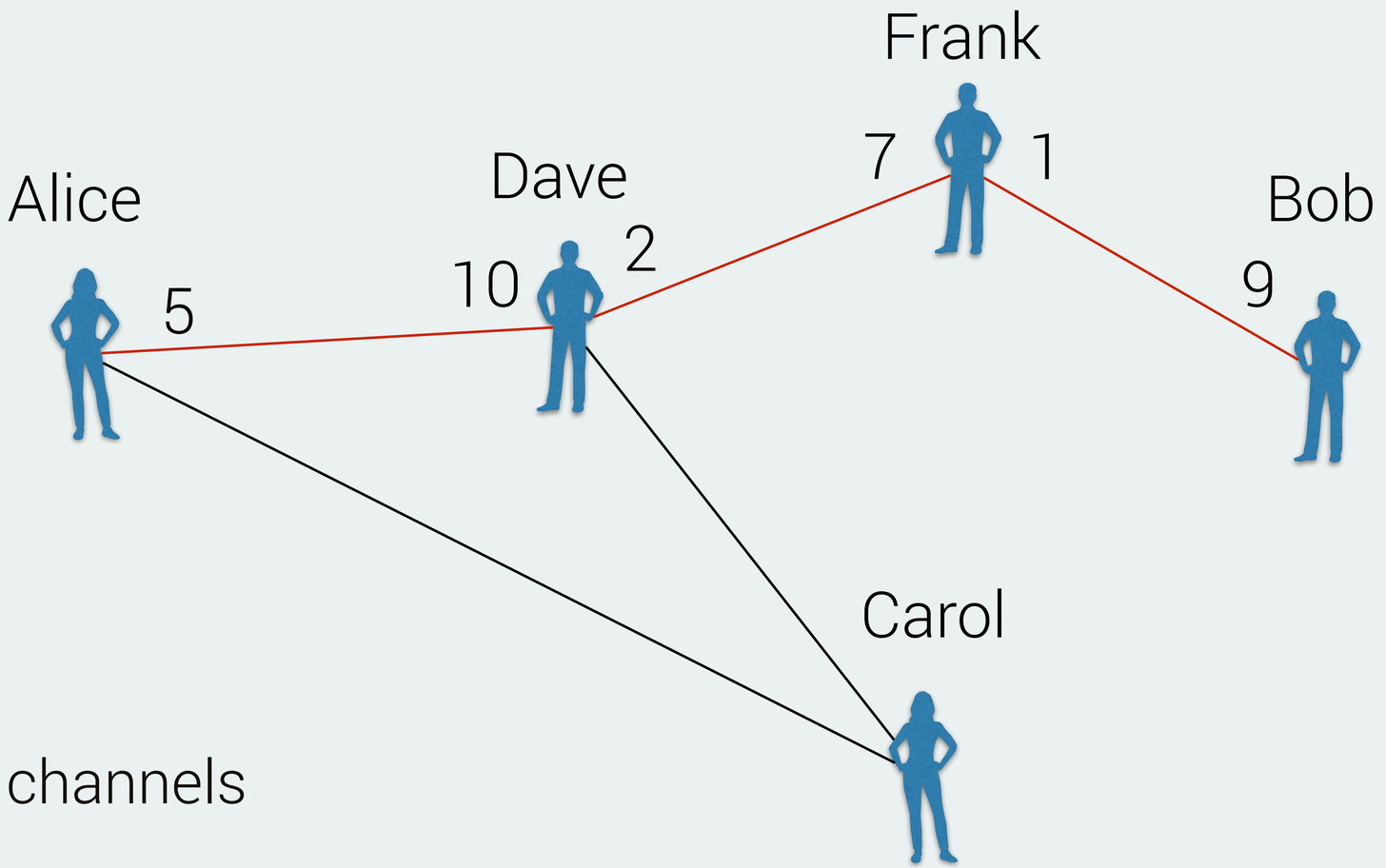
Alice -> Dave / 5 REP | H(s)

Dave -> Frank / 5 REP | H(s)

Frank -> Bob / 5 REP | H(s)

Alice reveals s to Bob

Everybody can update their channels



Potential Problems

Centralisation Risks ("Payment Hubs")

- Capital needs to be locked up for any channel
- Might incentive big hubs where everybody connects to

Channel Depletion

- If a lot of Eth is routed over a channel in only one direction
- Blocks both the sending party and future routing

Potential Problems

Routing Issues

- Routes cannot always be found
- but seems to work rather well in testing

No space on chain

- Channels only secure if counterparty message can be presented on time
- Might be problematic if chain is congested (e.g. big ICO)

State Channels

Generalisation of Payment Channels

Don't just sign balances, but arbitrary state

Idea: Unanimous Consent Objects

- Avoid reimplementations with Subchannels / Composability
- Counterfactual instantiation

Problems

- Requires unanimous consent on every change => Channel might stall
- Still requires complete onchain path for resolution

Truebit: Verifiable Computation

Only small computations feasible on chain

Old Idea:

- run offchain
- post result to blockchain (with some deposit)
- can be contested (within some time limit)
- traditionally requires running computation onchain
- which was infeasible to begin with

Truebit: Verifiable Computation

Truebit

- computation runs in some vm
- state of vm organised as a merkle tree
- merkle root after every instruction stored in another merkle tree
- root of that tree stored along with result

=> contract only needs to check a single instruction in case of dispute

=> still not the cheapest, but cheaper than all the alternatives

Onchain Scaling

Many offchain solutions

- still require much higher onchain throughput
- rely on availability for security

Methods

- Block Gas Limit
- Transaction Groups
- Sharding

Block Gas Limit

Maximum allowed gas usage per block

Closest thing ethereum has to a blocksize limit

A miner can raise or lower it by 1/1024th per block

Could theoretically double in ~3h (with 100% miner support)

Not really a scaling solution!

Nodes still need to process every transaction!

Transaction Ranges / Groups

Right now all transactions need to be processed sequentially

Some could be processed in parallel if they don't affect each other

Idea: Transactions specify the range of accounts allowed to be touched

=> A set can be run in parallel if ranges don't overlap

Not really a scaling solution either!

Only a one-time 8x increase

Sharding

Long established concept from the database world (but more difficult here)

Always requires more trust in validators than conventional blockchains*

Basic idea

- Split the chain and state into multiple parts (=shards)
- Different parts have their own validators
- Different parts can be processed mostly independently
- But still mechanisms for cross shard communication in place

* if you don't verify every shard yourself

Sharding in Ethereum

Current approach requires Casper (or equivalent slashing scheme)

Shard 0 is special: contains "collation headers"

Those are basically like block headers but for shards

Validators from the pool are assigned to a shard at random

All consensus-relevant transactions happen in shard 0

Treat some shards (includes 0) like full nodes, others like light clients

Sharding in Ethereum

Shards can communicate with each other through issuing receipts

These can be checked across shards

Receipts need to be deep within some shard before being accepted

This is to contain reorganisations to their shard

This also is not a full scaling solution!

Can only scale up by a constant factor

Sharding in Ethereum

More complex sharding protocol required for further scaling

Proposals still change frequently

Very little independent analysis of the current ones

For details about sharding see

- Mauve Paper
- github.com/ethereum/sharding
- the Sharding FAQ

Scaling Timeline

onchain

No improvements in the Metropolis forks

Small EVM performance improvements (and later eWASM)

Sharding probably won't be implemented until after Casper deployment

But maybe we will see some kind of transaction groups

Will come in multiple forks

Sharding in 2018 very unlikely

Scaling Timeline

offchain

- Payment channels already exist (though they lack usage)
- Raiden to release developer preview *soon*^{TM}
- No general solution for state channels yet
- Truebit only theory for the most part

All presentations available at:

github.com/ethereum-vienna-meetup