Blockchains & Cryptocurrencies

Scaling

www.securitities.io

Instructor: Abhishek Jain
Johns Hopkins University - Spring 2021

The Problem

- Bitcoin transaction rate: 5-7 tx/sec
    - Bounded by block size, TX size
    - All transactions must be globally verified, stored
- Ethereum: 15 transactions per second if they're small
- Visa: 24,000/sec peak (150M/day globally)
- WeChat 256,000/sec peak

Faster computers?

- Why not just build faster computers?

Faster computers?

- Why not just build faster computers?
  - Loss of decentralization
  - Eventually we saturate links, due to broadcast network
  - Replicated global state falls apart
  - Scaling is possible (see Visa, WePay etc.) but it requires dedicated, centralized servers

Can we do better?

Can we do better?

- **Current ideas**:
  - "Off-chain" transactions
  - New consensus algorithms
  - "Sharding"

Off-chain transactions (i.e., **channels**)

- In current Bitcoin-style networks, every transaction appears on the blockchain
    - This allows the whole network to verify financial integrity
    - I.e., we can't go off and do transactions elsewhere, accidentally/deliberately inflate the money supply
- But why does the network need to see every transaction?

Off-chain transactions (channels)

- Overarching idea:
    - If a transaction doesn't affect anyone else (except for the parties willing to risk money), chain doesn't need to see it
    - Simplest example (but centralized):
        - Multiple parties deposit money into an exchange
        - Exchange is just a centralized bank, so everyone can quickly transmit money by adjusting balances
        - Only withdrawals need on-chain transactions

Off-chain transactions (channels)

- Off-chain exchange example still risks loss of funds
  - If the exchange disappears, your money goes with it
  - See e.g., QuadrigaCX
  - The only benefit here is that the rest of the network can't lose money, e.g., due to inflation

Today

- Lightning Network (Poon, Dryja)
- Basic design
- Shortcomings/attacks and defenses

A warmup before Lighting Network

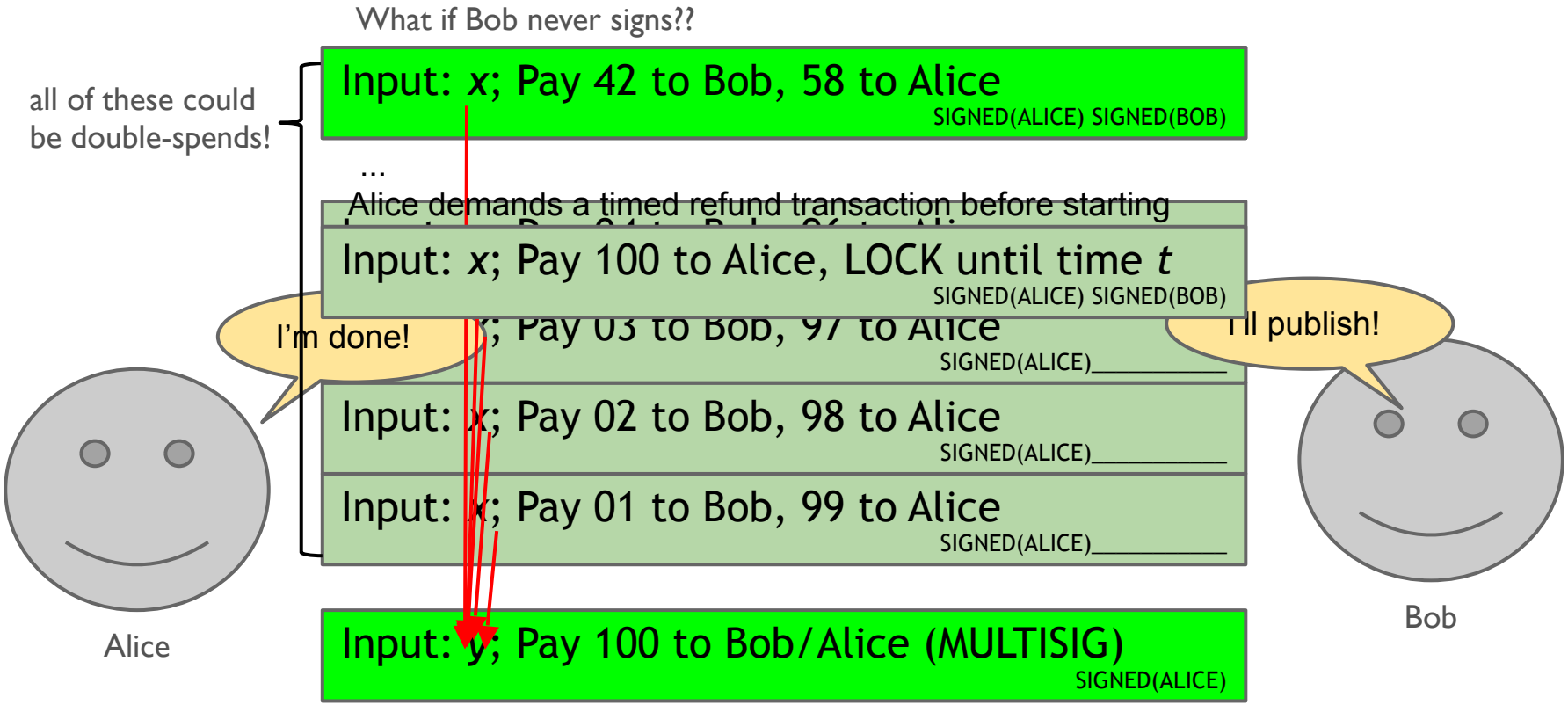
Recap: Micro-payments with Bitcoin

- Pay-as-you-go WIFI: Alice wants to pay WIFI provider (Bob) for each minute of WIFI service. But she doesn't want to incur a transaction fee for every minute
- Similarly, pay-as-you-go online subscriptions
- Ad-free websites

Recap: Micro-payments with Bitcoin

- Main Idea: Instead of doing several transactions, do a single transaction for total payment (and thus incur only a single transaction fee)
- *How to implement it?*

Example 3: Micro-payments with Bitcoin



lock_time


```
{
    "hash":"5a42590...b8b6b",
      "ver":1,
      "vin_sz":2,
      "vout_sz":1,
      "lock_time":315415,
      "size":404,
...
}
```

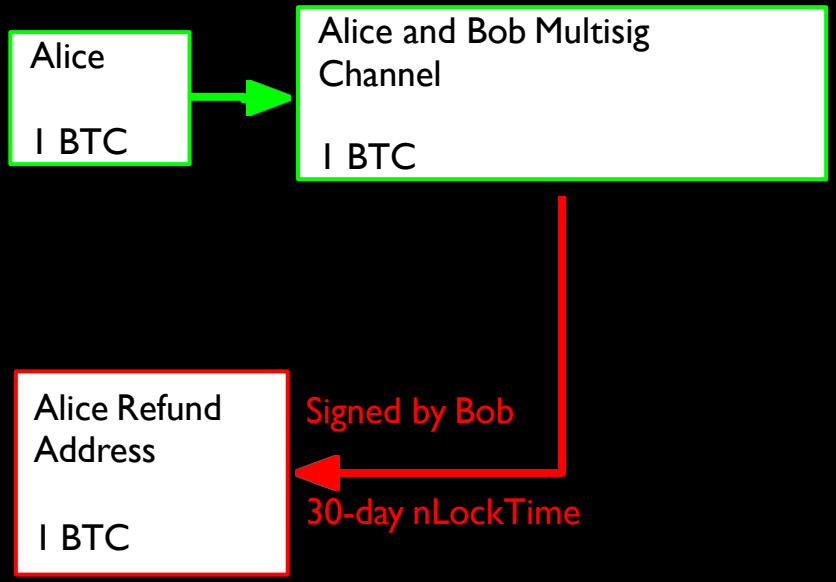

Block index or real-world timestamp before which this transaction can't be published

Lightning Network

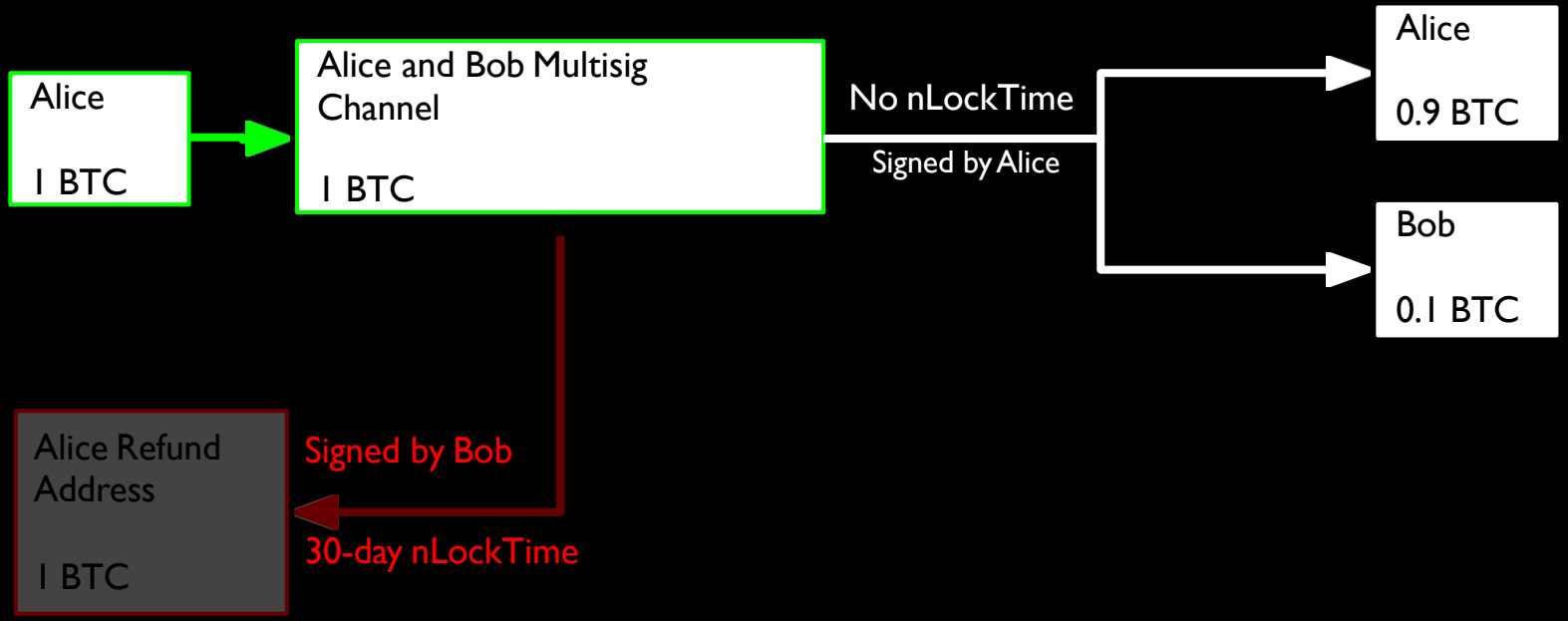
Joseph Poon Thaddeus Dryja

Slides credit: Lightning Network

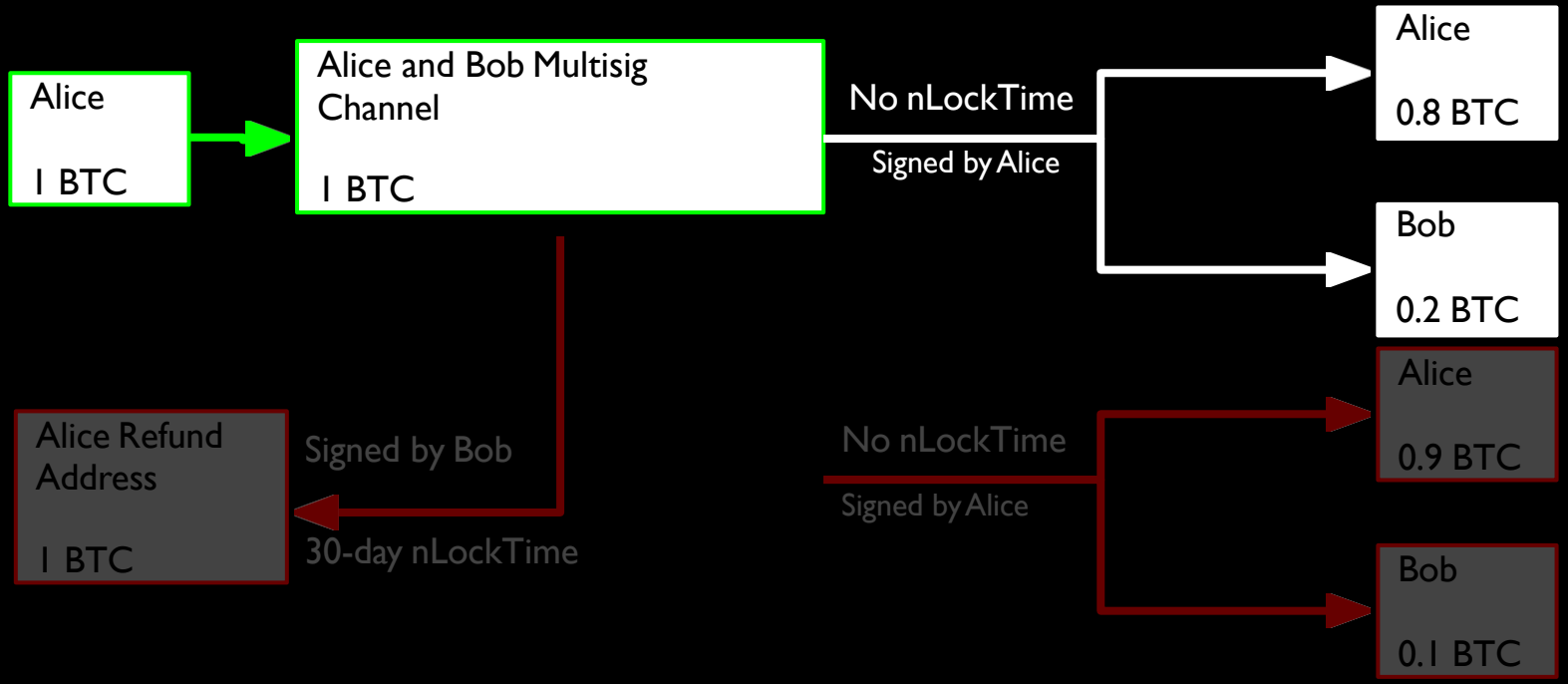
Unidirectional Channel



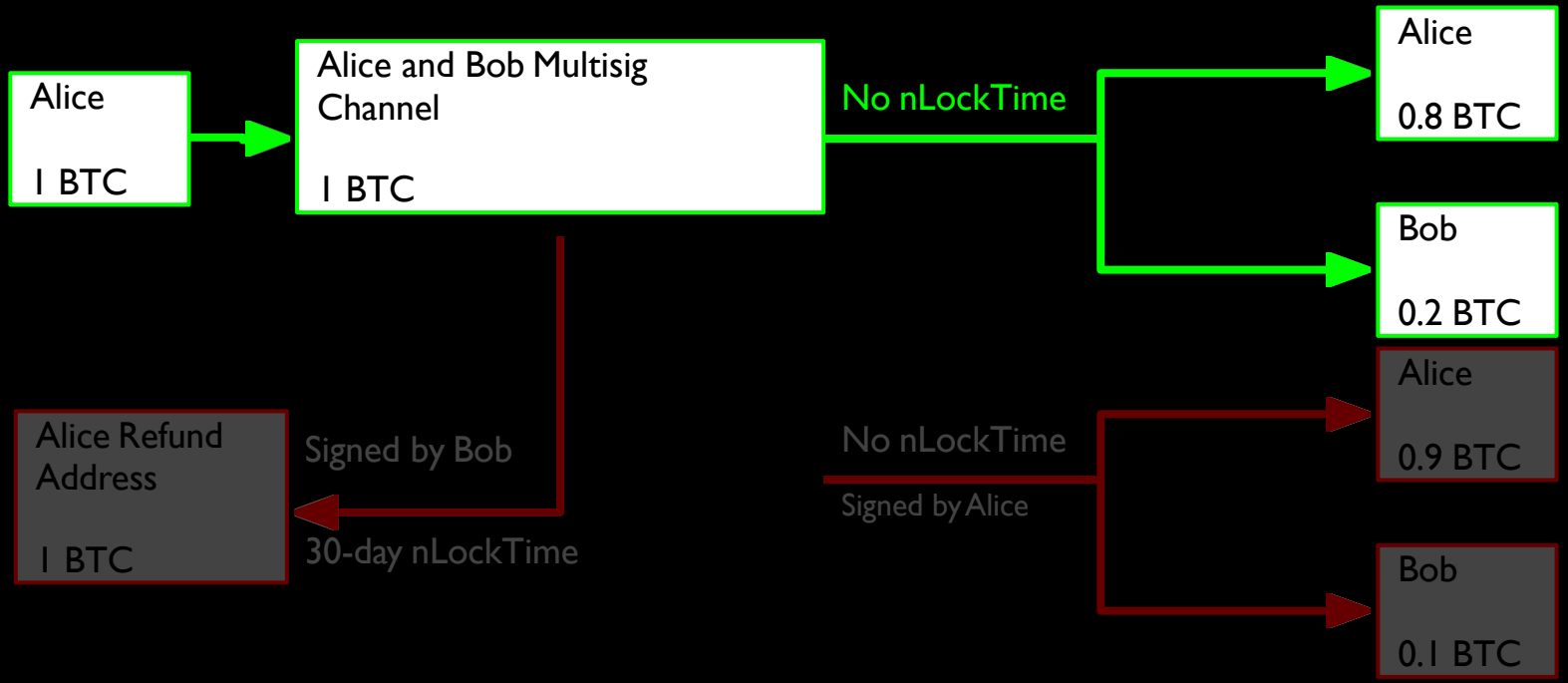
Unidirectional Channel



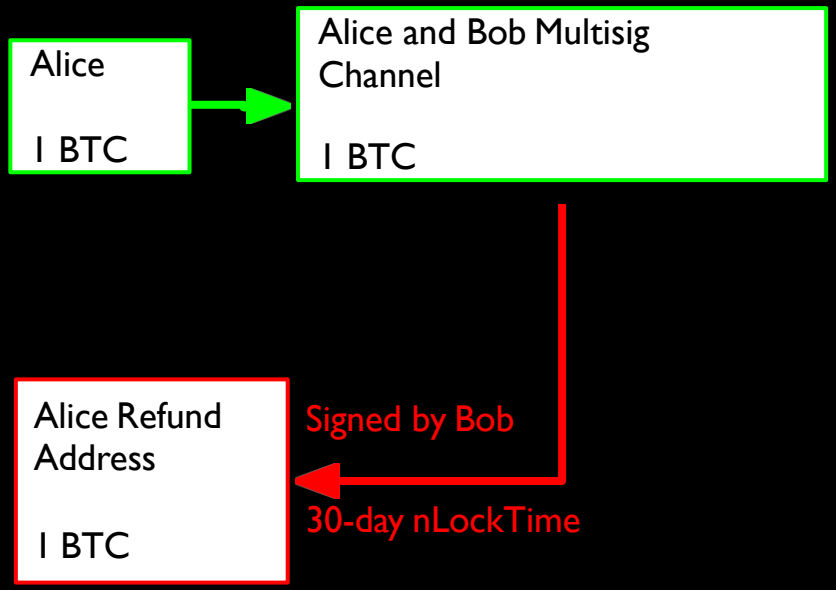
Unidirectional Channel



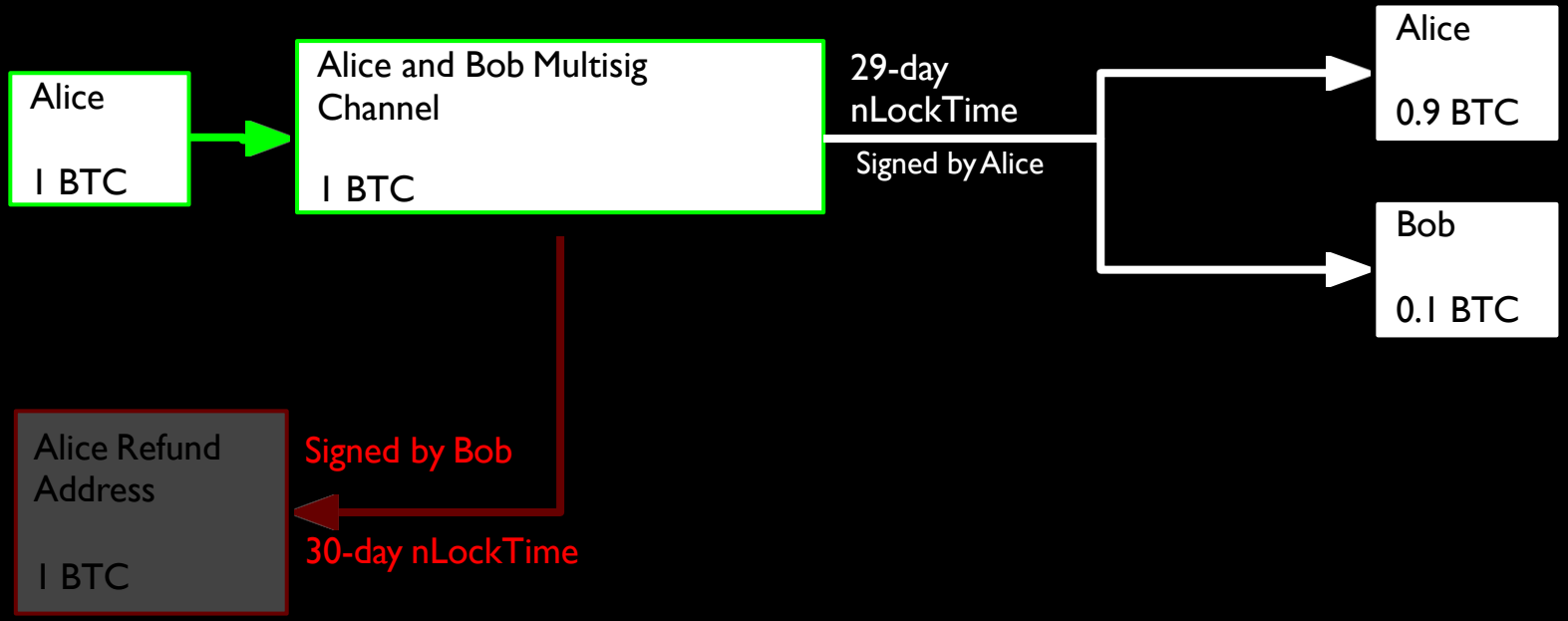
Unidirectional Channel



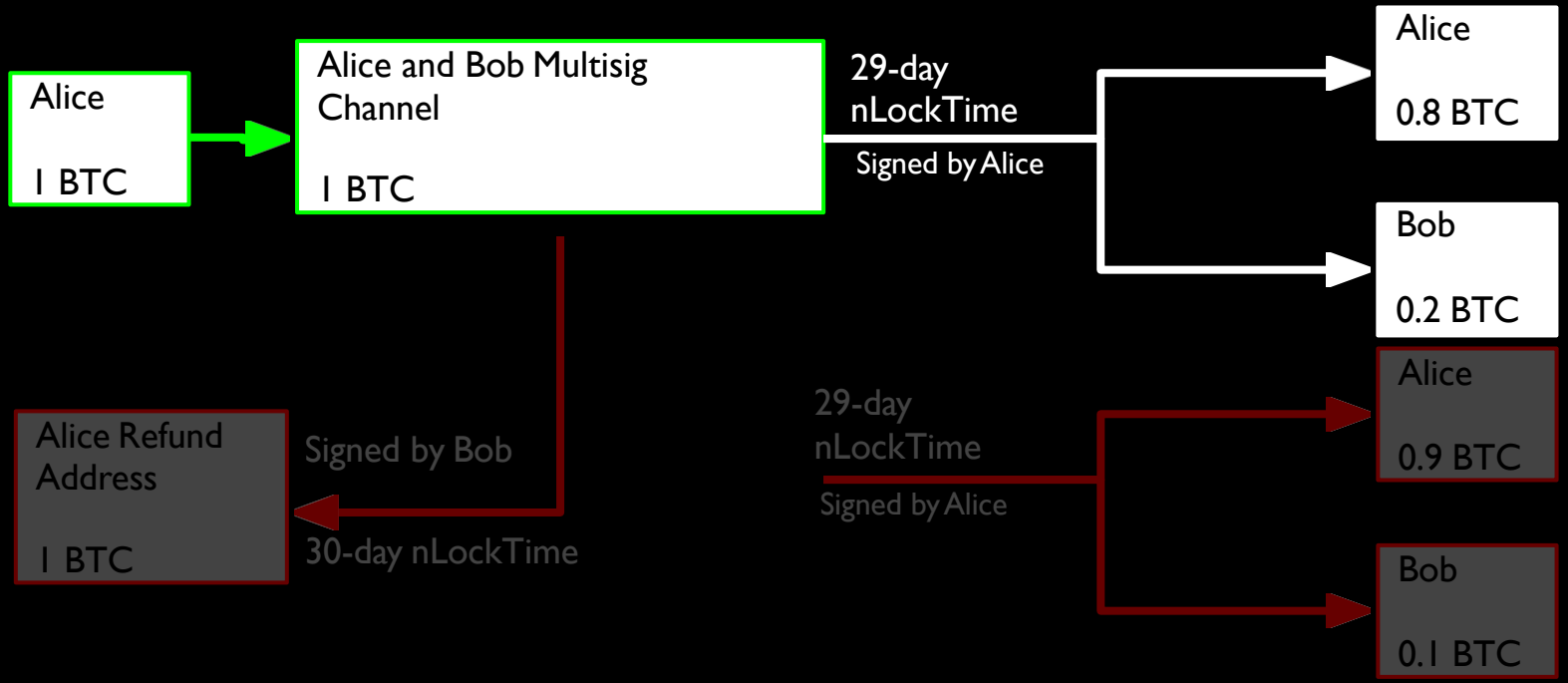
Bidirectional Channel



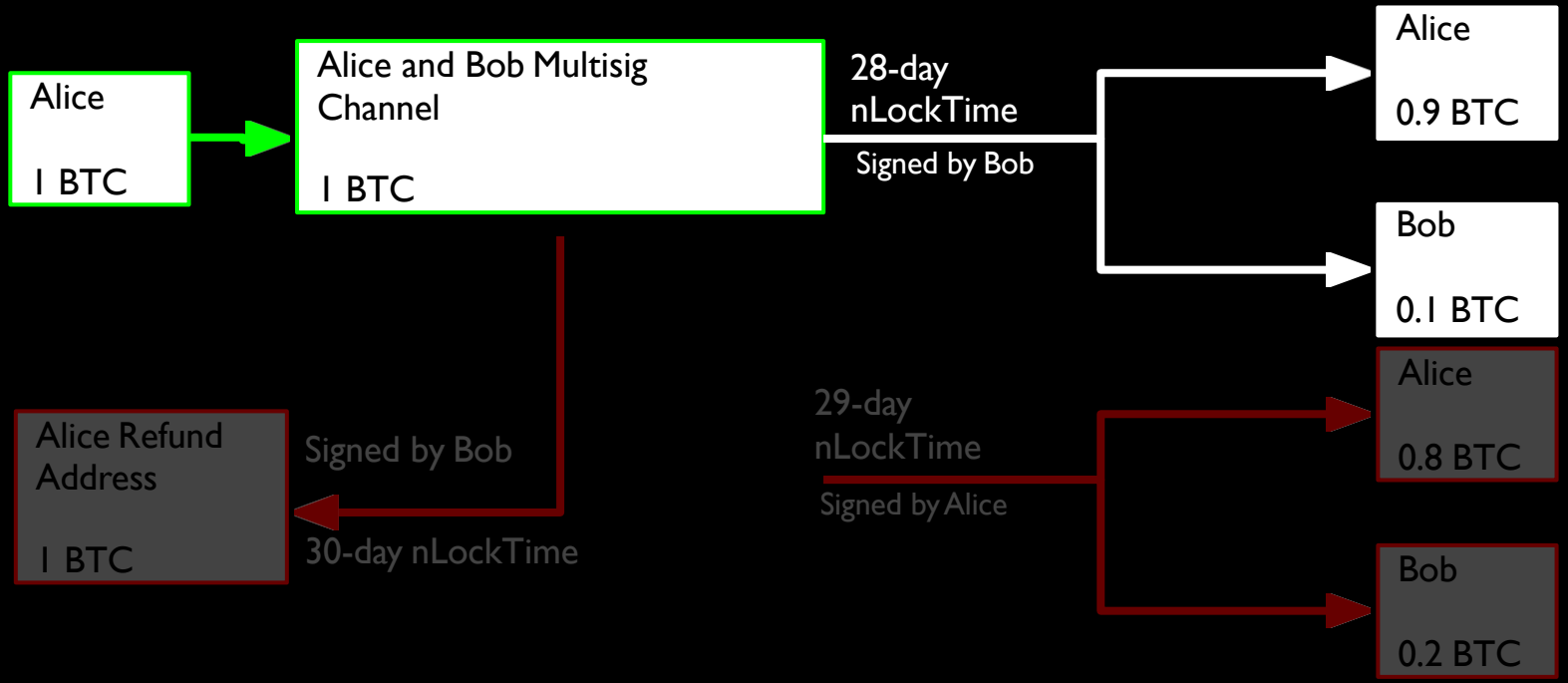
Bidirectional Channel - Payment



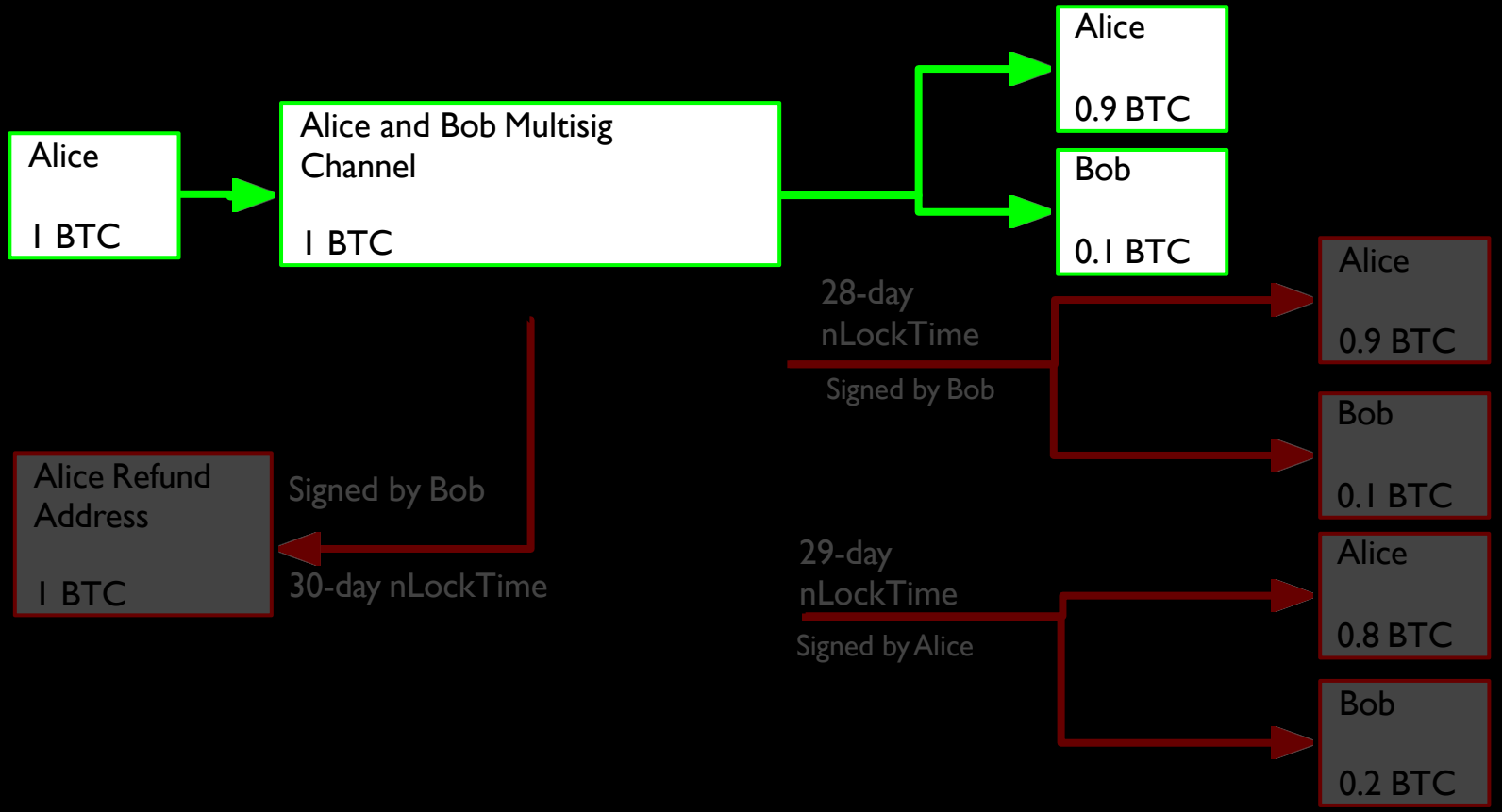
Bidirectional Channel - Payment



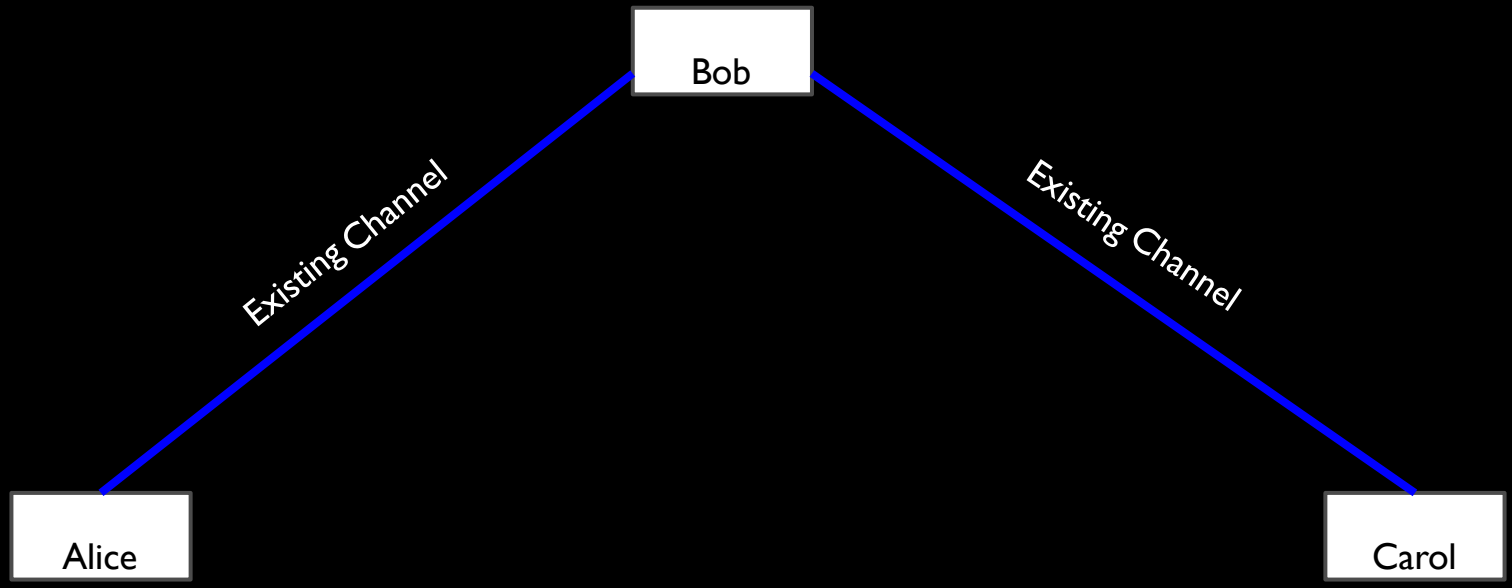
Reversing Direction



Closing Bidirectional Channel

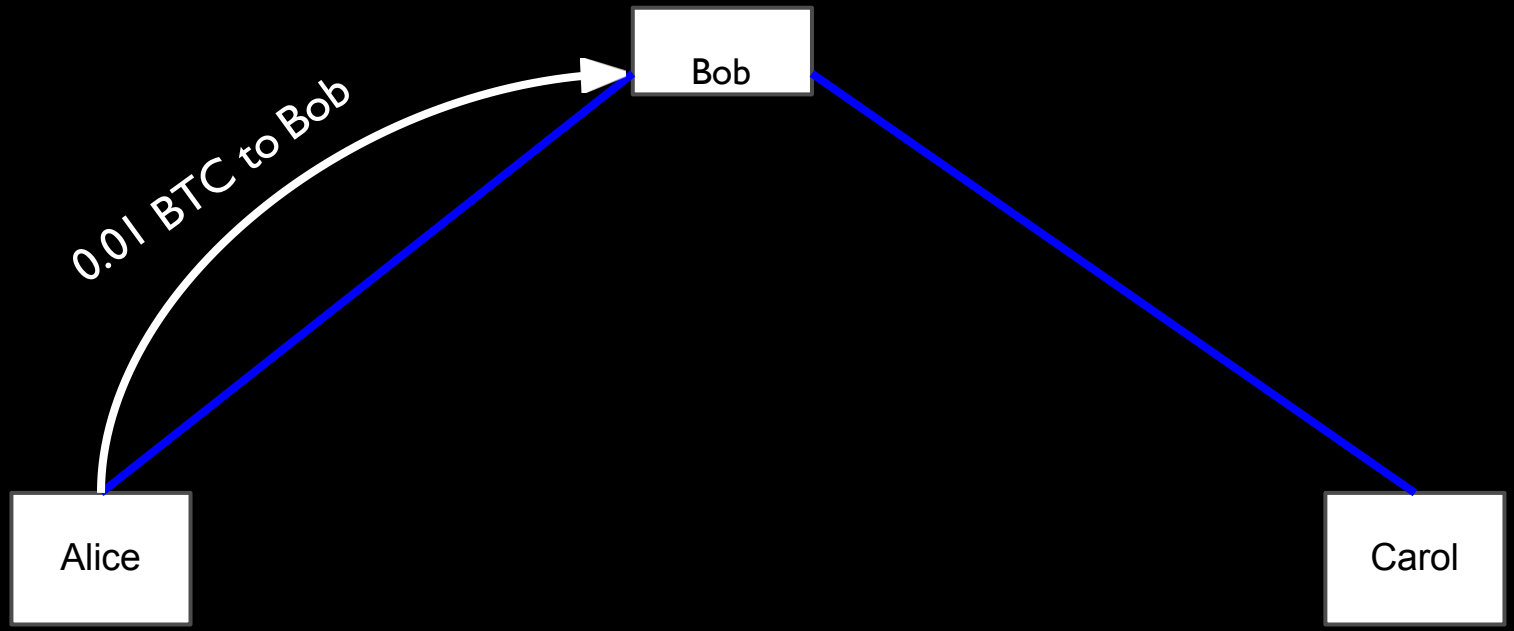


3 Party Payments

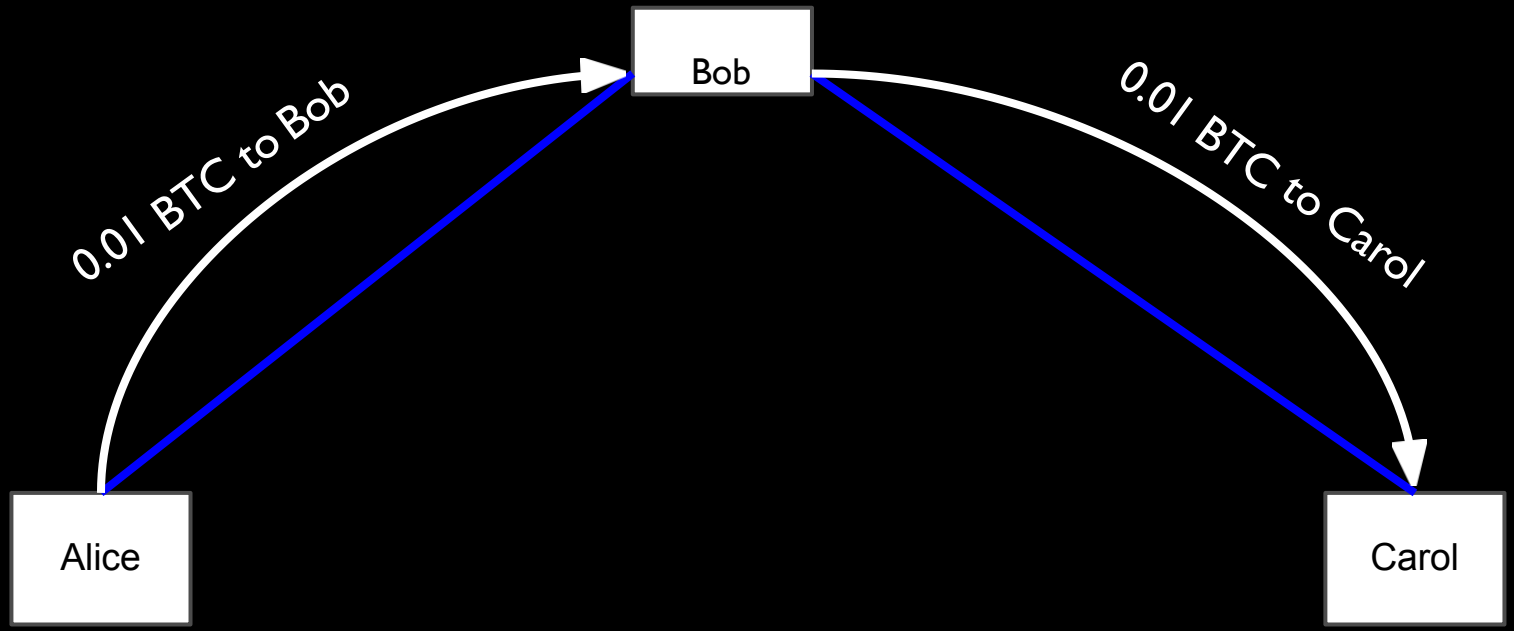


Alice wants to pay Carol, they both have a channel open with Bob

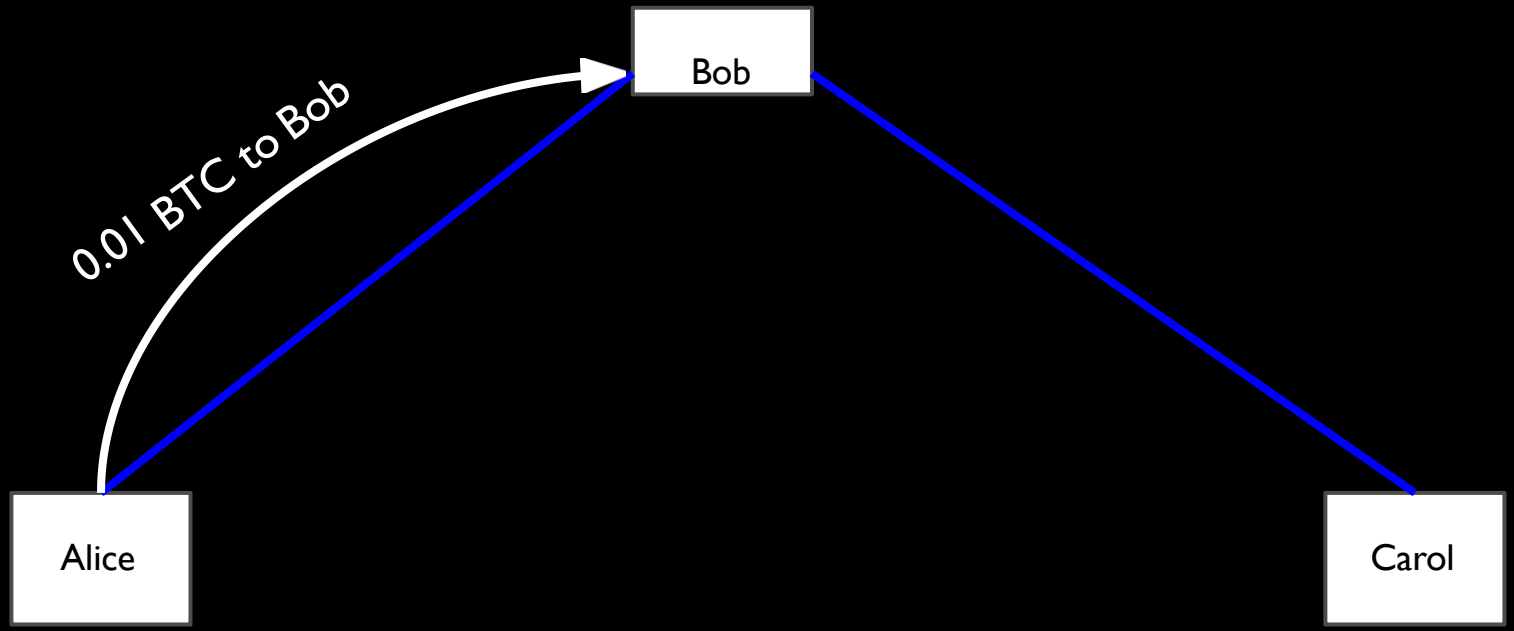
3 Party Payments



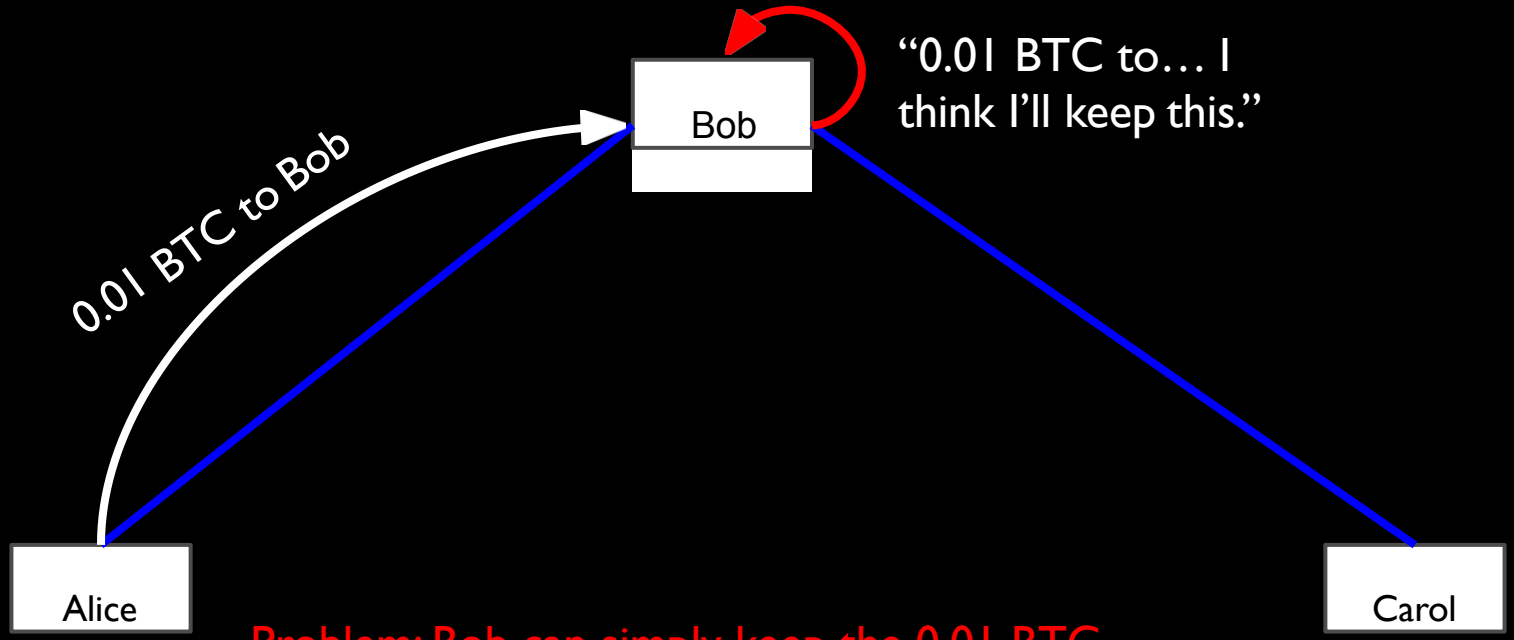
3 Party Payments



3 Party Payments - Trust Issues



3 Party Payments - Trust Issues

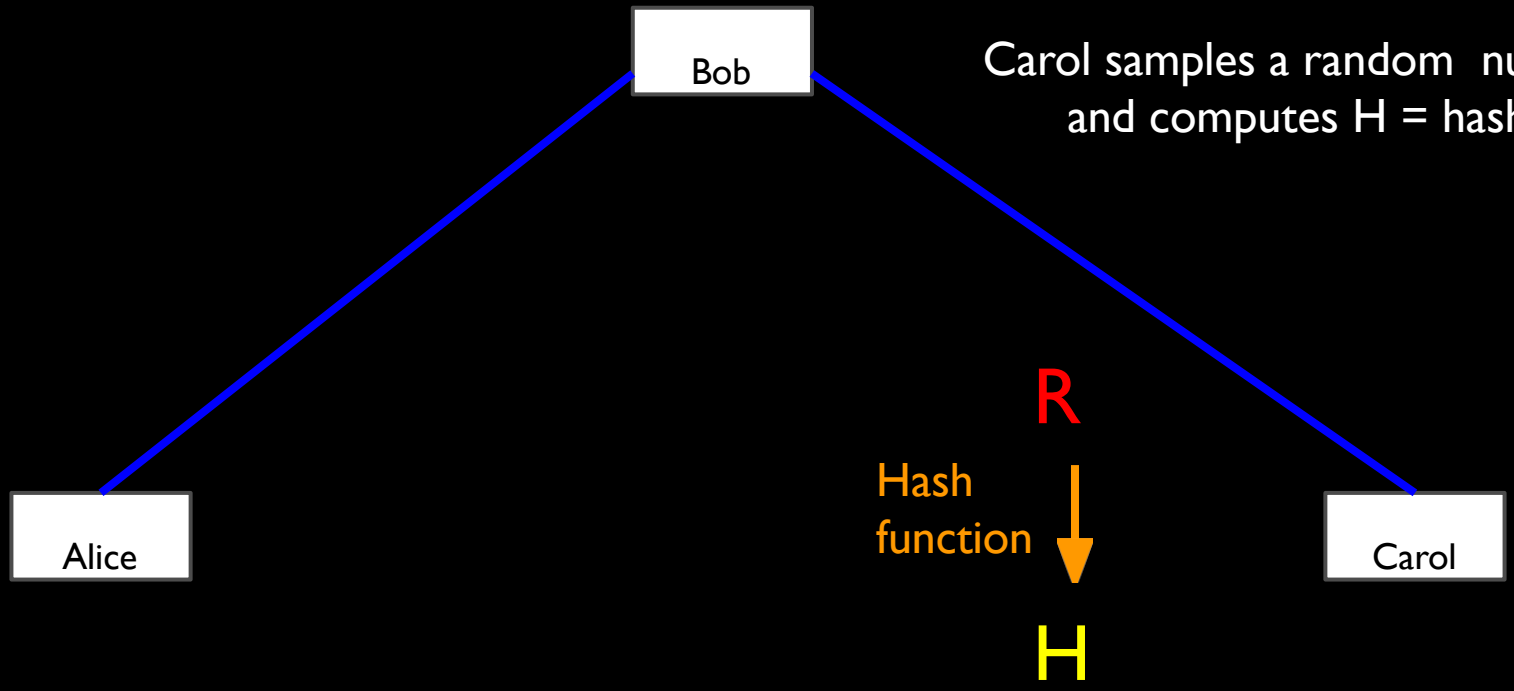


Problem: Bob can simply keep the 0.01 BTC
Problem: Carol can claim she never got the coins!

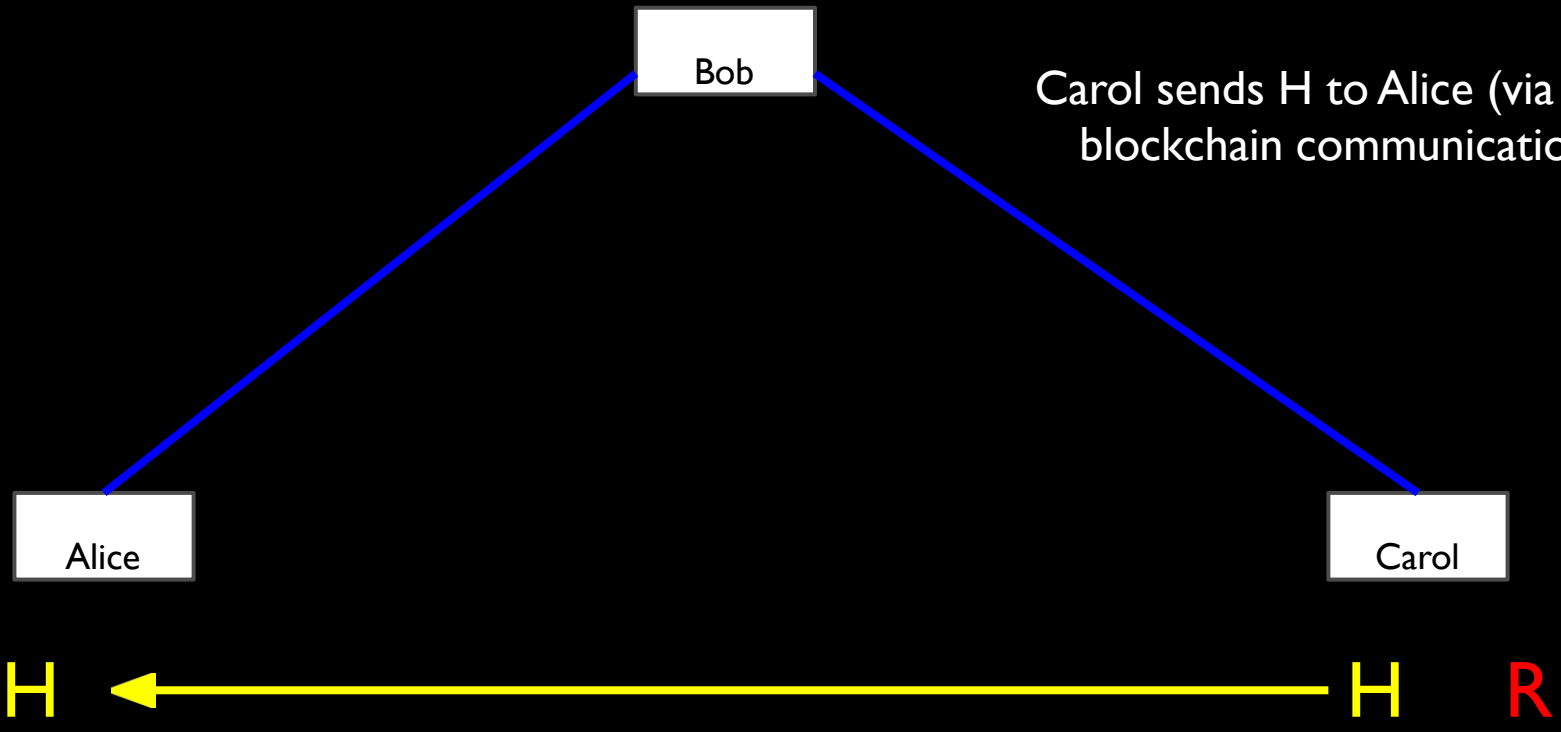
Hash-Locked Contracts

- Using hash functions, Alice can prove she sent funds to Carol off-chain
- Pay to Contract
  - Knowledge of R hashed into H proves receipt
  - Receiver signs a contract stating if R is disclosed funds have been received

Hash-Locked Contracts

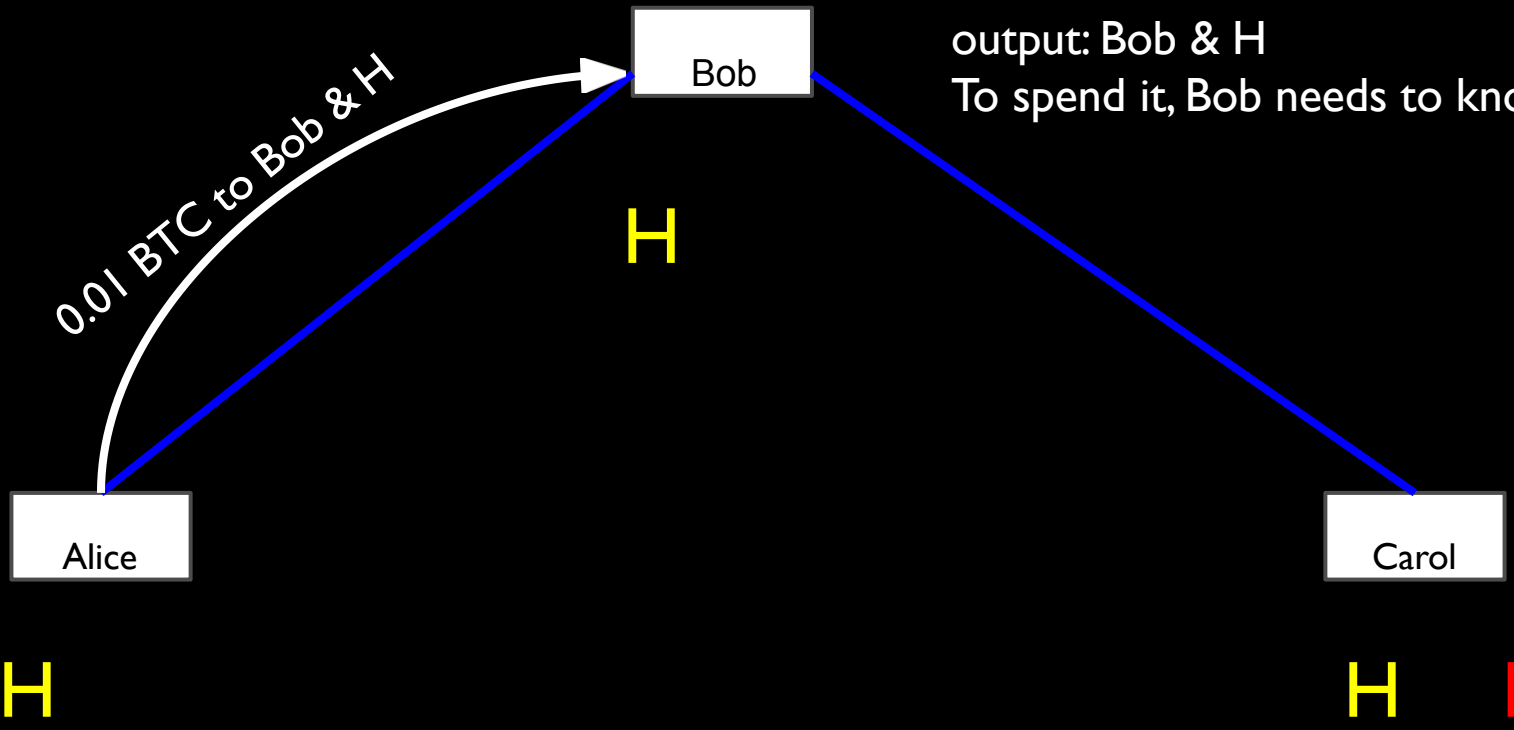


Hash-Locked Contracts



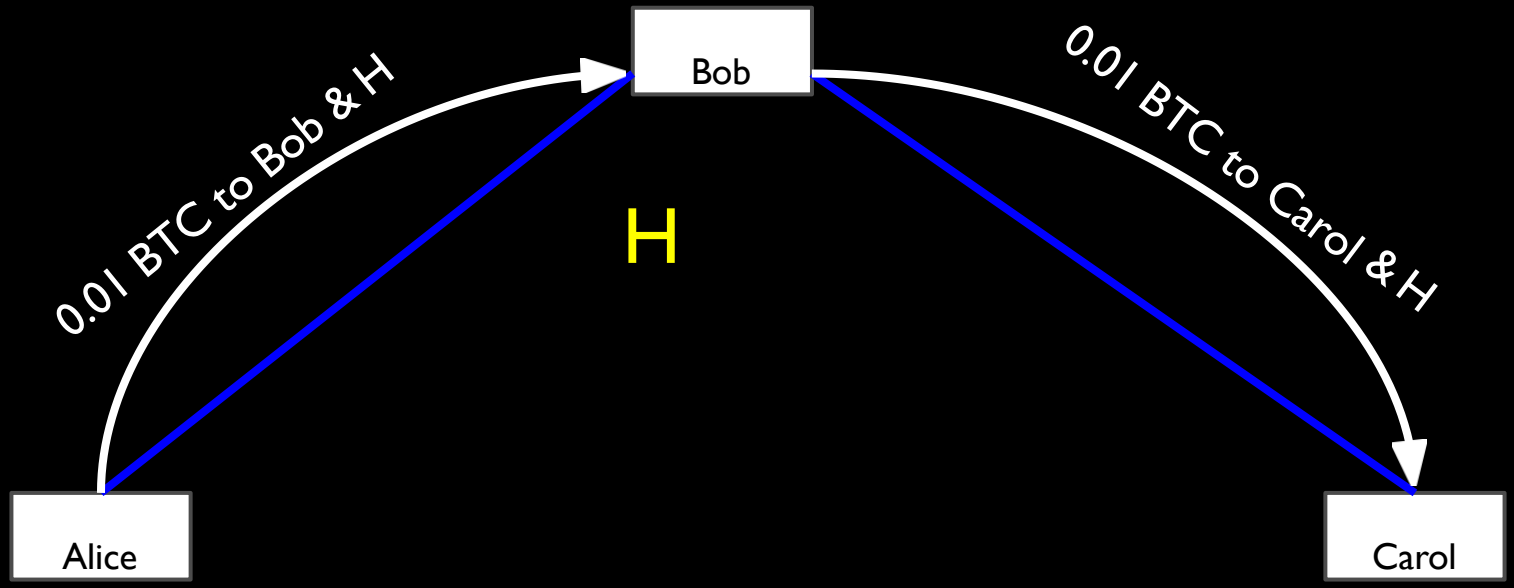
Carol sends H to Alice (via non-blockchain communication)

Hash-Locked Contracts

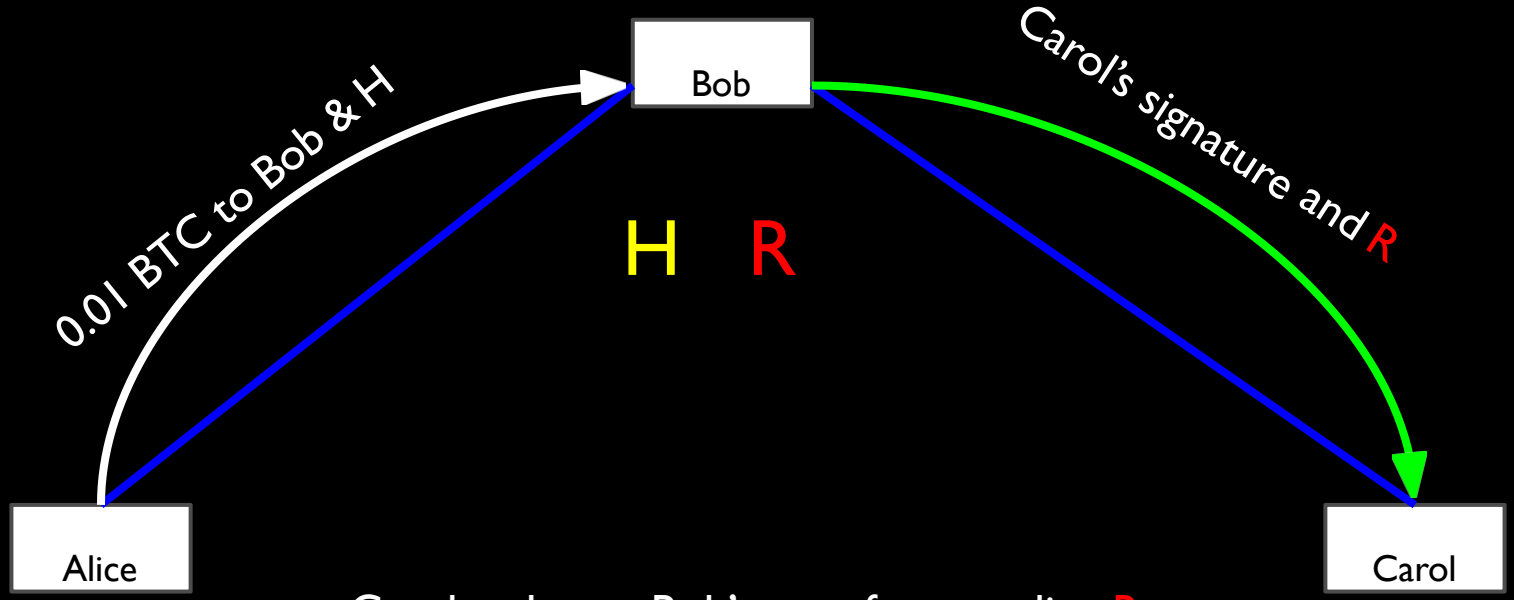


Alice sends 0.01 BTC to a new output: Bob & H
To spend it, Bob needs to know R

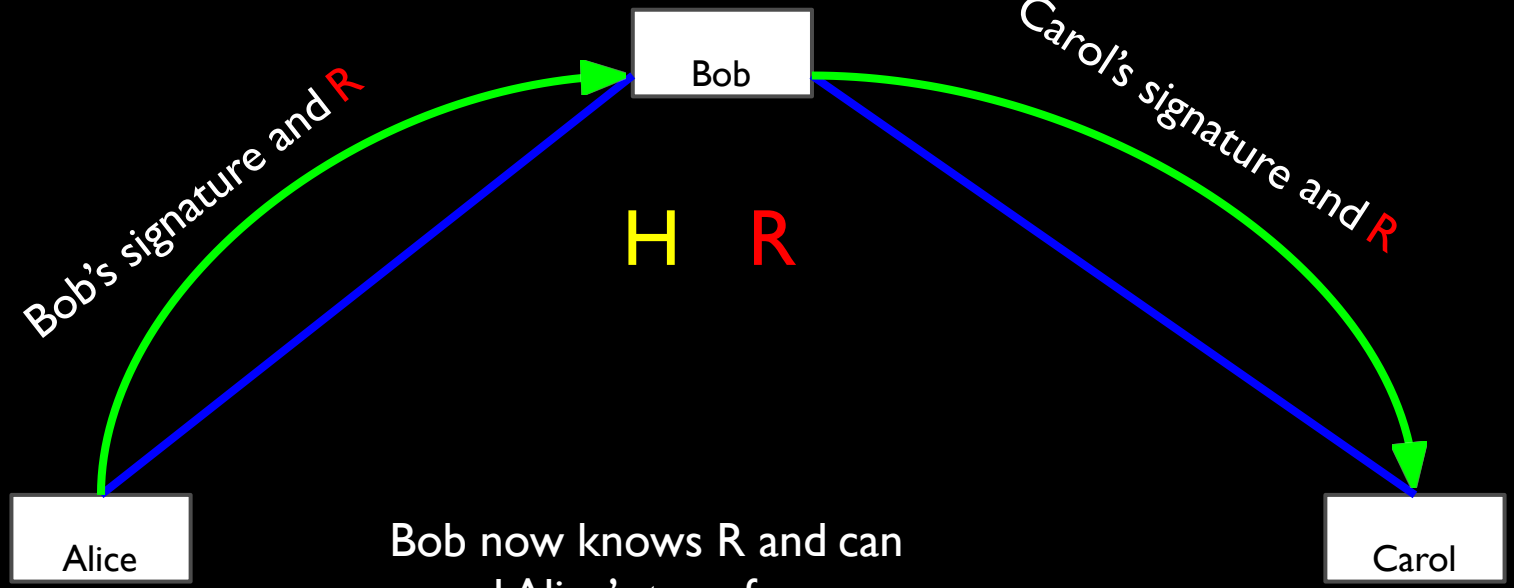
Hash-Locked Contracts



Hash-Locked Contracts



Hash-Locked Contracts

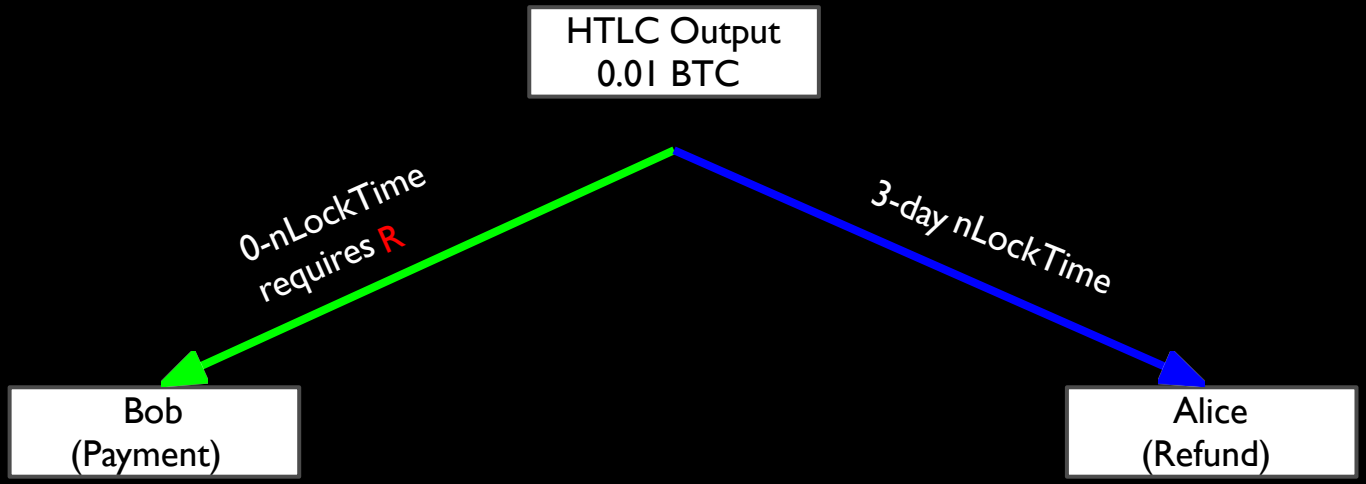


Bob now knows R and can spend Alice's transfer.
The coins have gone from Alice to Carol via Bob

Problem

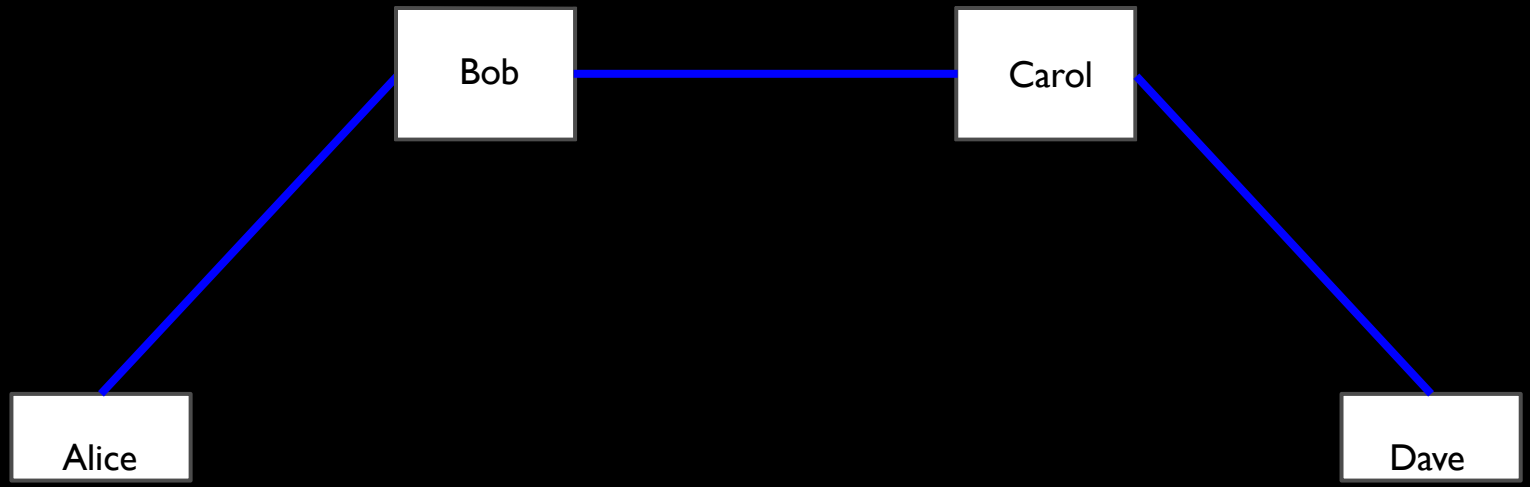
If Carol refuses to disclose R, she will hold up the channel between Alice and Bob

Hashed Time-Lock Contract



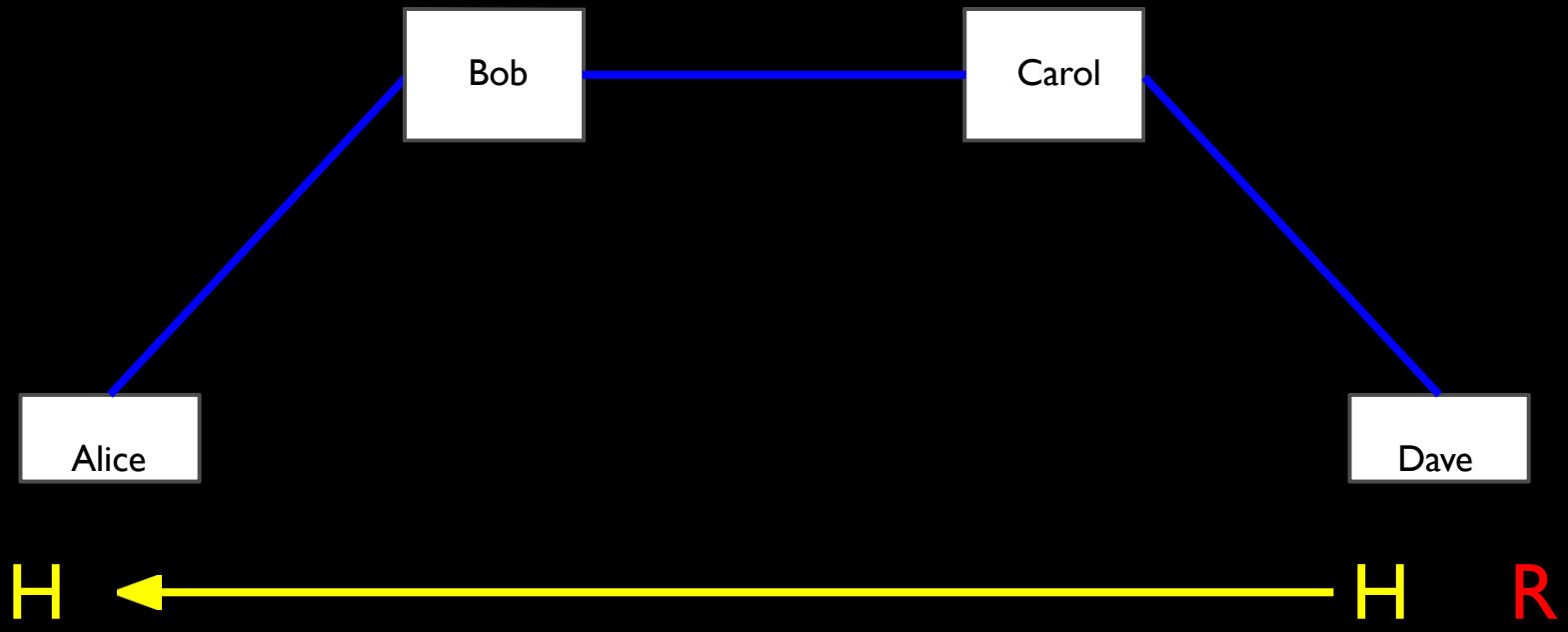
Going beyond three parties

Alice wants to send funds to Dave via Bob and Carol

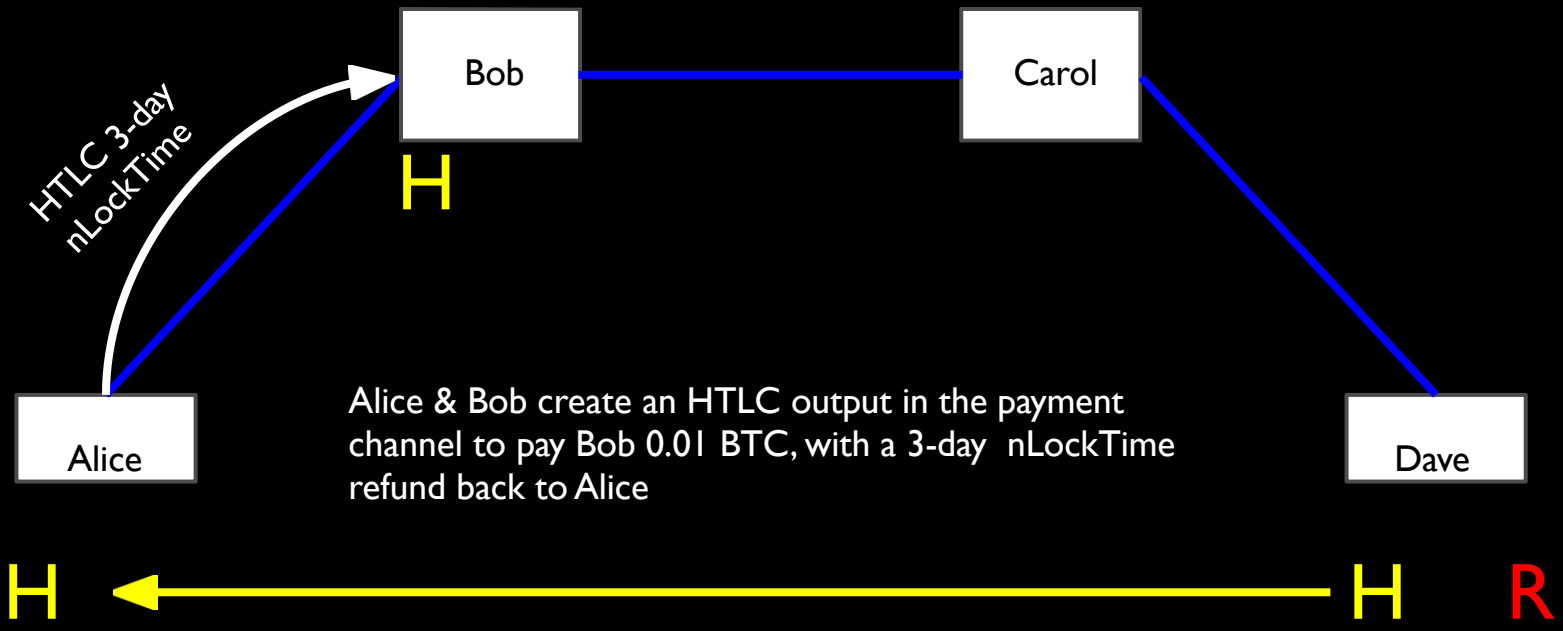


Going beyond three parties

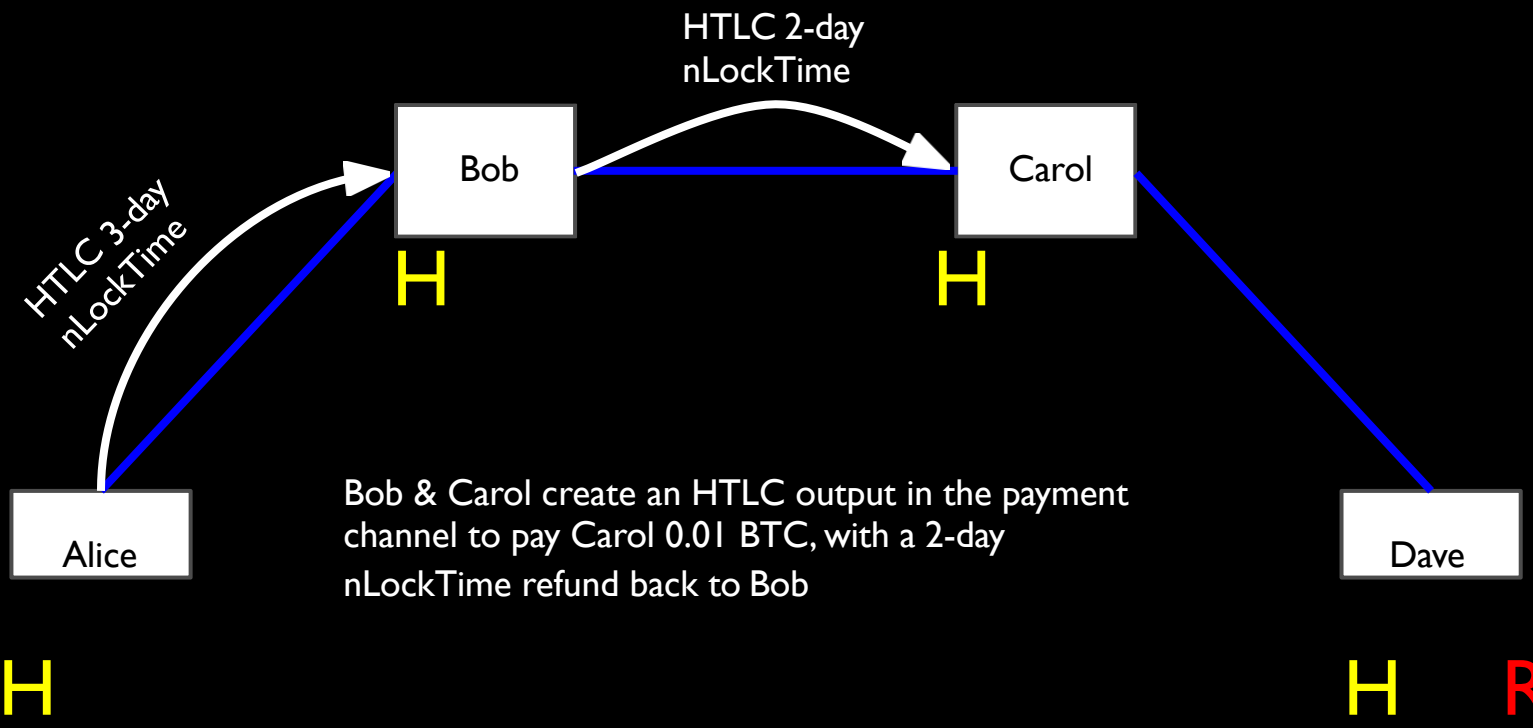
Dave sends Alice hash H produced from random data R



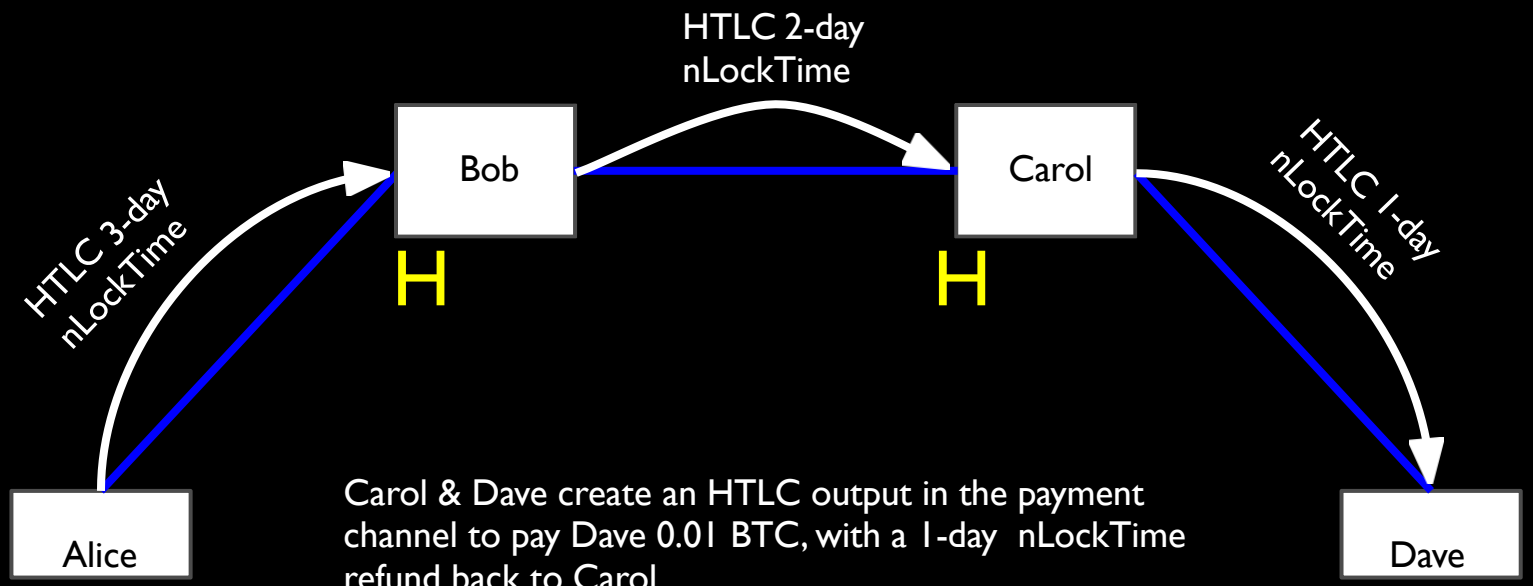
Going beyond three parties



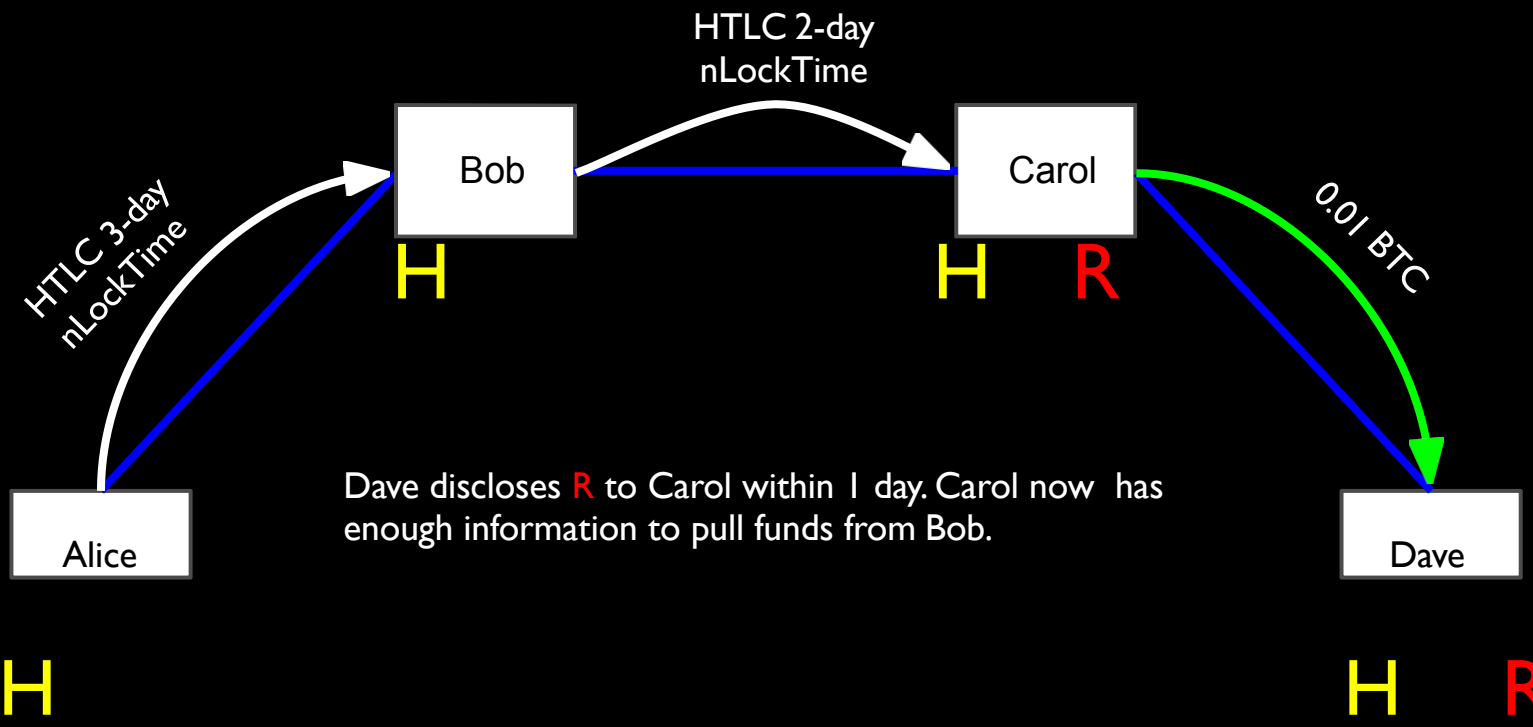
Going beyond three parties



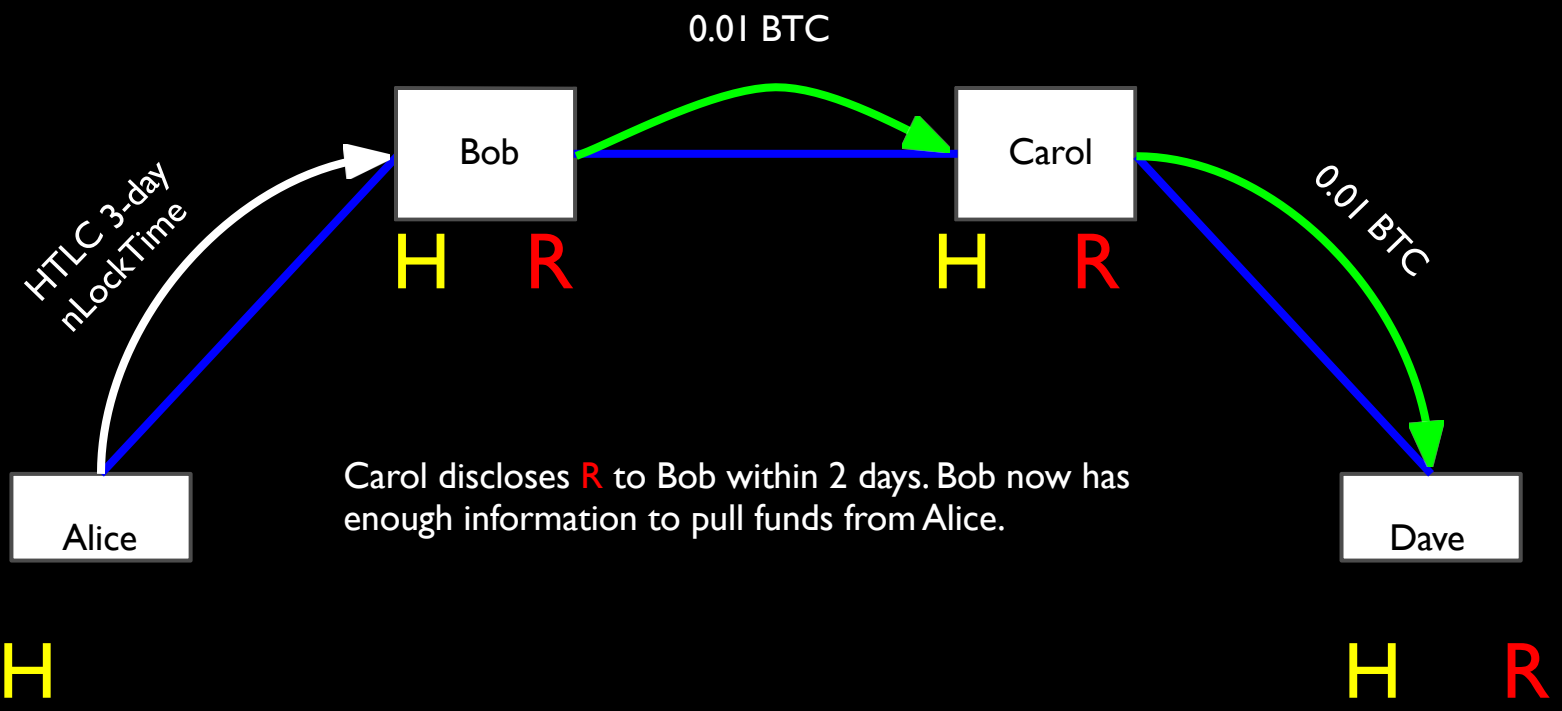
Going beyond three parties



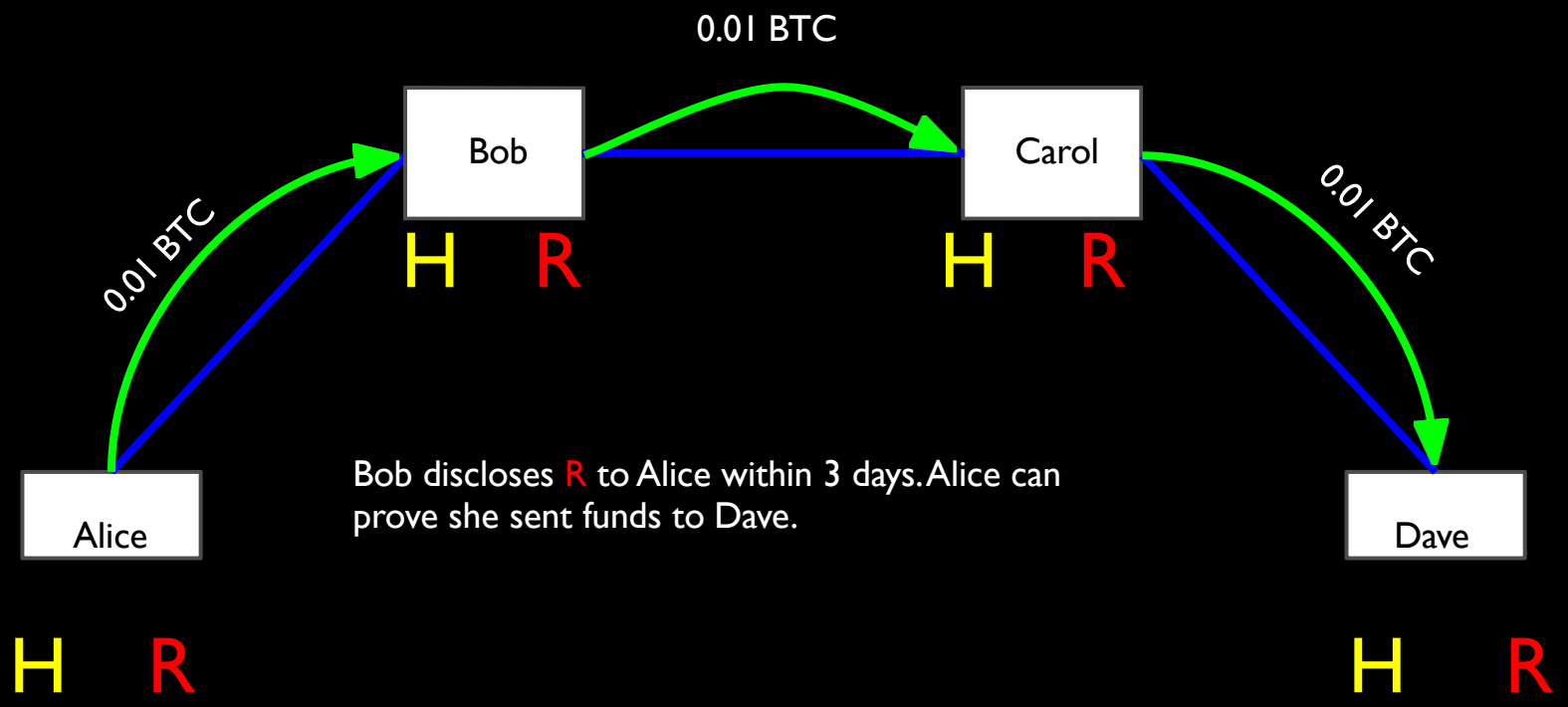
Going beyond three parties



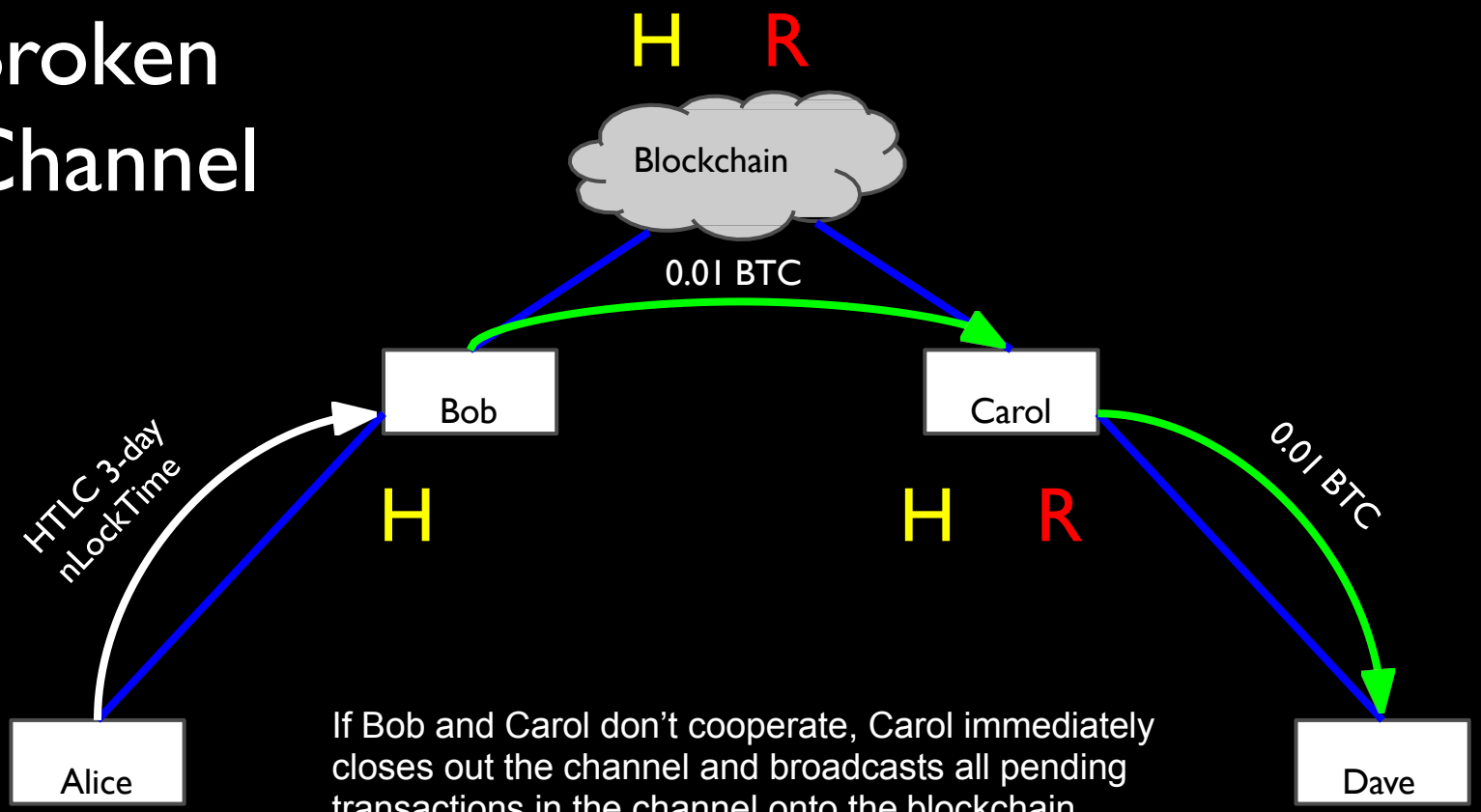
Going beyond three parties



Going beyond three parties



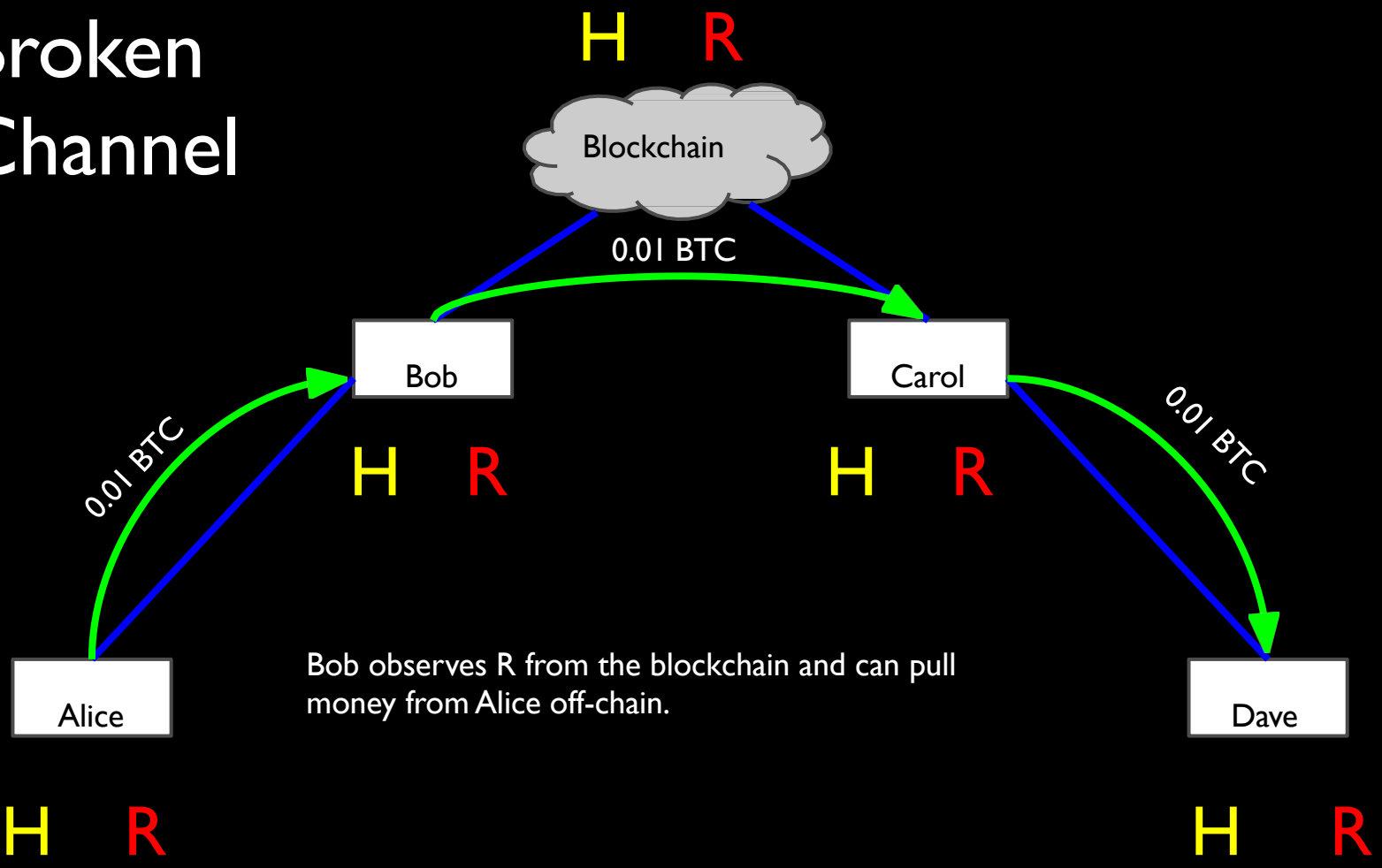
Broken Channel



If Bob and Carol don't cooperate, Carol immediately closes out the channel and broadcasts all pending transactions in the channel onto the blockchain.

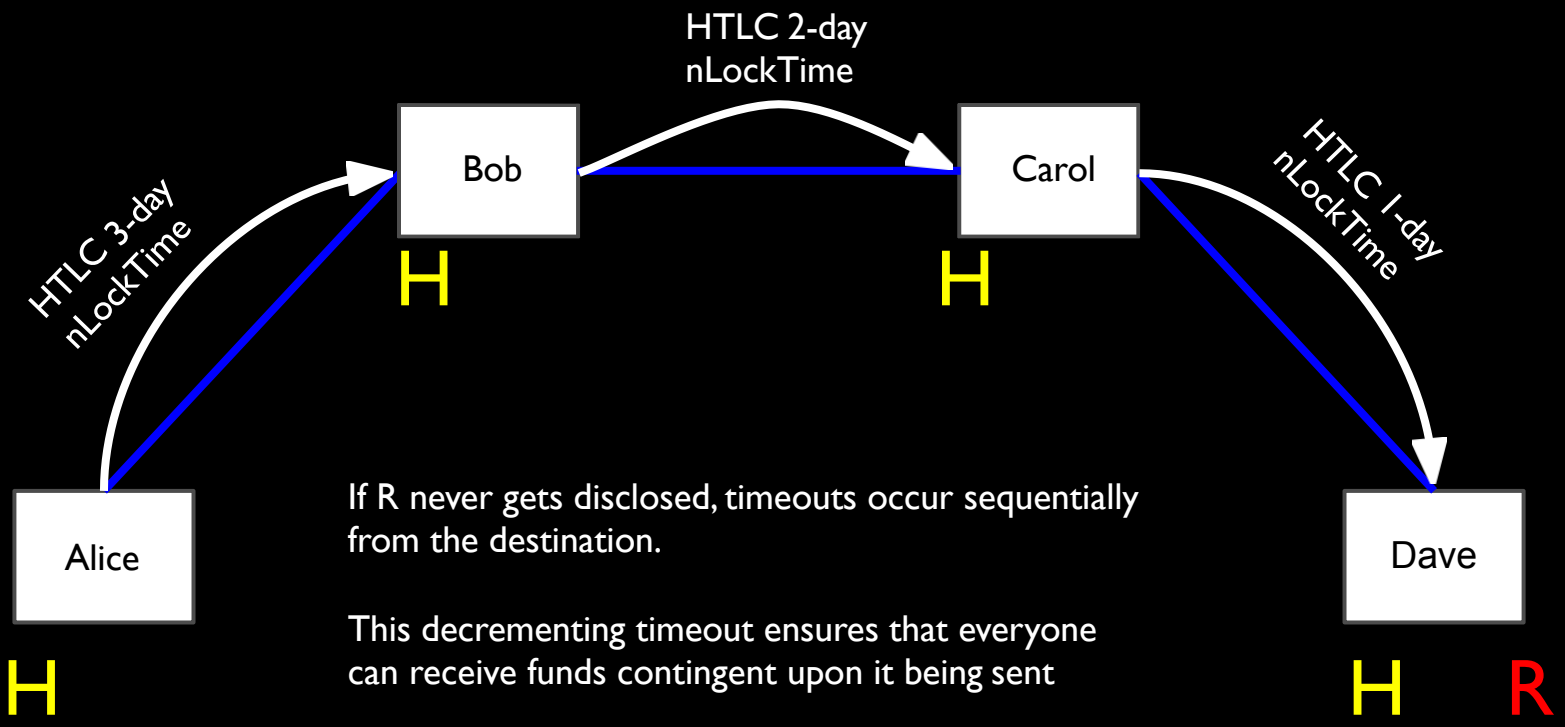
Carol redeems the 0.01 by disclosing the HTLC output and R on the Bitcoin blockchain.

Broken Channel



Bob observes R from the blockchain and can pull money from Alice off-chain.

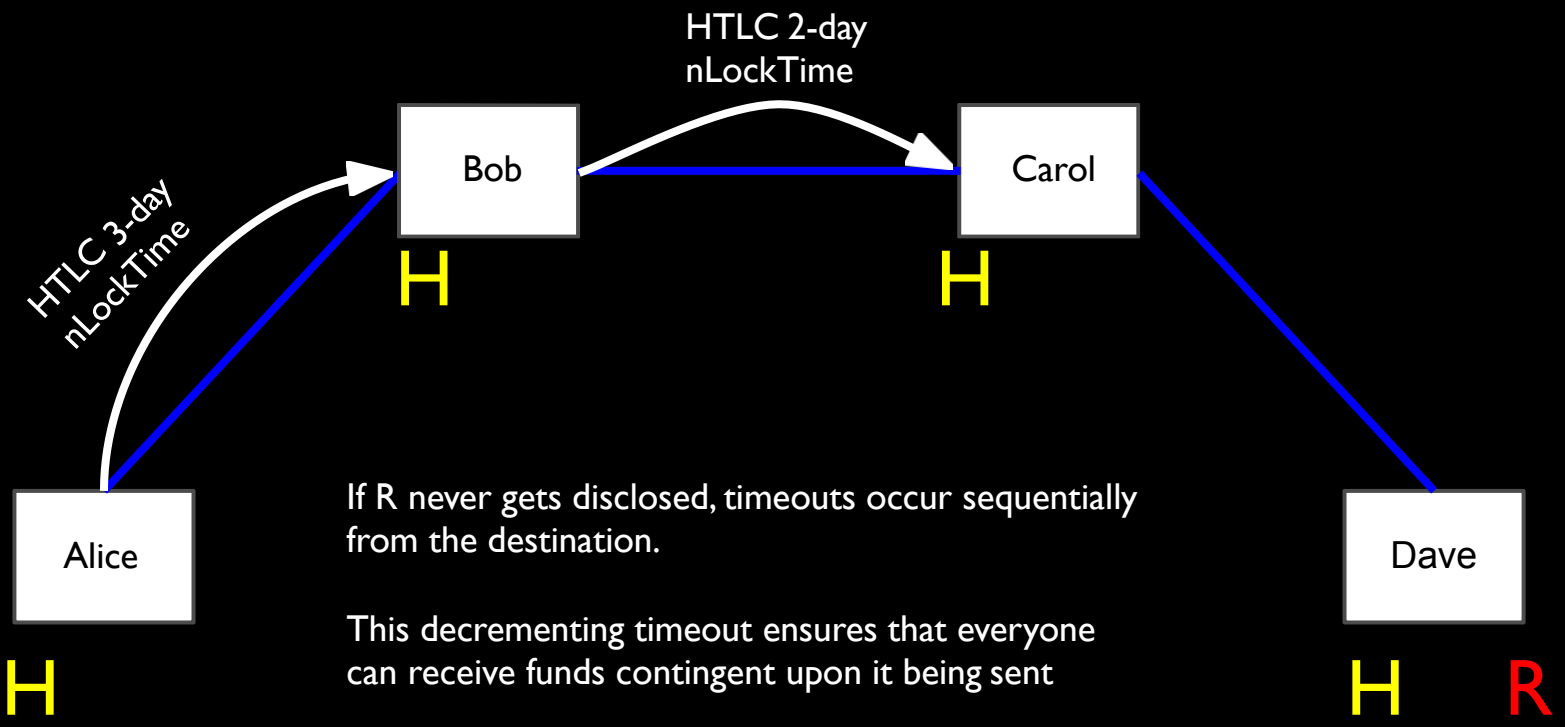
Timeout



If R never gets disclosed, timeouts occur sequentially from the destination.

This decrementing timeout ensures that everyone can receive funds contingent upon it being sent

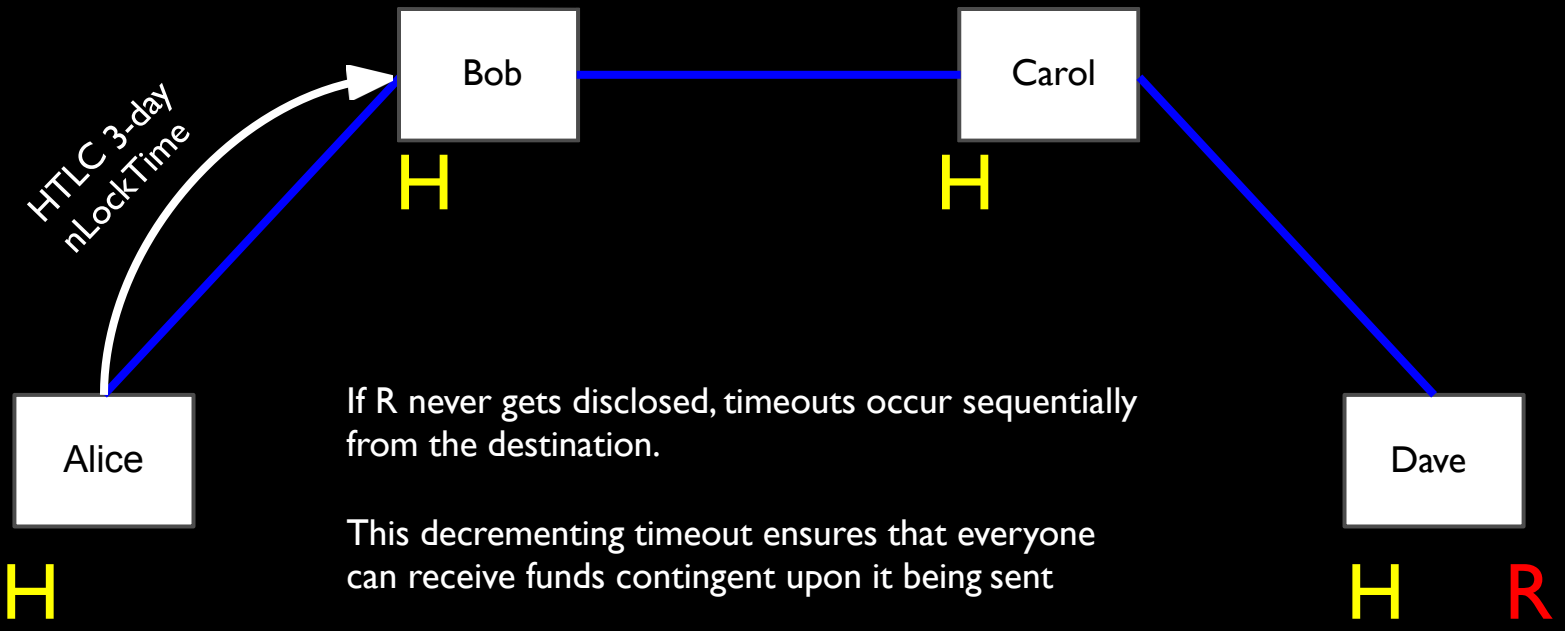
Timeout



If R never gets disclosed, timeouts occur sequentially from the destination.

This decrementing timeout ensures that everyone can receive funds contingent upon it being sent

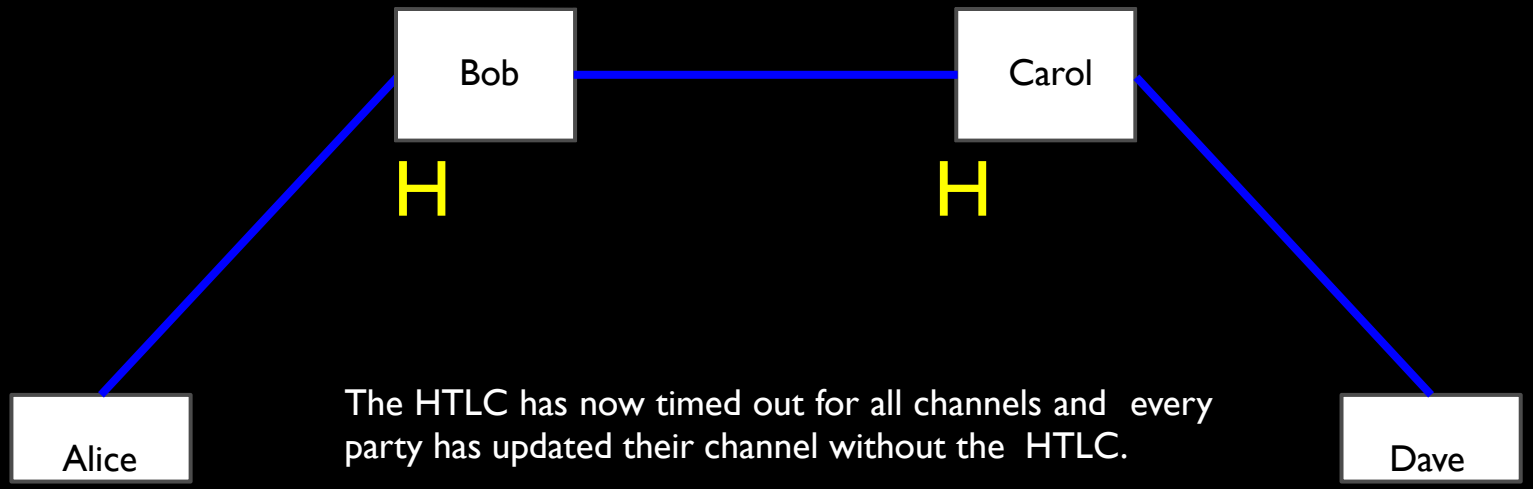
Timeout



If R never gets disclosed, timeouts occur sequentially from the destination.

This decrementing timeout ensures that everyone can receive funds contingent upon it being sent

Timeout



The HTLC has now timed out for all channels and every party has updated their channel without the HTLC.

If one party is uncooperative, that channel is broadcast on the blockchain.

Limitations

- Not secure against collusion attacks
  - Wormhole attack [Malavolta et al., NDSS'19]
- Anonymity limitations: Sender/receiver can be easily linked
  - See, e.g., [Malavolta et al., CCS'17, Green-Miers, CCS'17]
- Reliant on scripting (time-locks)
  - Solution without scripting [Malavolta et al., NDSS'19]

Wormhole Attack

- **So far**: Lightning network described using egalitarian nodes
- In reality, intermediate nodes require **"fees"** for their service
- **Wormhole attack:**
  - Two intermediate nodes collude to steal the fees of the nodes "between them".
  - The longer the chain, the better the payoff.

Wormhole Attack (contd.)

- **Main Idea**: Nodes **X** and **Y** withhold the release value **"R"** from the nodes between them.
    - Say **X** is the right-neighbor of *sender* and **Y** is left-neighbor of the *receiver*
    - Upon receiving "**R**", node **Y** will withhold it, i.e., not send it to its left neighbor and instead directly send it to **X**. Time-outs ensue, sending channels to their prior states, one-by-one, all the way back to node **X**.
    - Now, before next-time out ensues, **X** will use **"R"** to redeem contract with *sender*.
- **Payoff:** Total fees charged by nodes between X and Y

Wormhole Attack (example)

- **Assume**: 1 BTC fees charged by every intermediate node

- **HTLC(X,Y, hash, amount, timeout):** A hash time-lock contract between X and Y for hash value "hash", amount = "amount", with expiration = "timeout" days

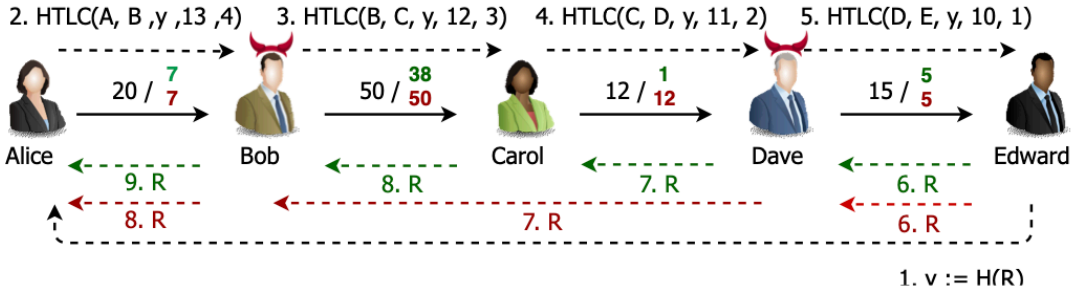


Fig. 1: Payment (with and without wormhole attack) from Alice to Edward for value 10 using HTLC contract. The honest (attacked) releasing phase is depicted in green (red). Non-bold (bold) numbers show the capacity of payment channels before (after) the payment. We assume a common fee of 1 coin.

Understanding the attack

- **Main reason for attack:** All links use the same release value **"R"**
- Wait, why doesn't receiver simply broadcast **"R"** to everyone?
- **Problem:**
  - Implementing broadcast may itself require blockchain
  - Moreover, *receiver* itself may be colluding!
- **Solution:** Use different (but related) "locks" for each link
  - Enforce **sequential unlocking**. No "shortcuts" possible

(Simplified) Solution using HTLCs

- Sender sets lock between nodes **(i, i+1)** as $L_i = H(R_{i+1}+\ldots+R_n)$
- Sender sends "partial" release R_i to node **i**
- Sender sends "final" release R_n to receiver.
- <u>Sequential unlocking:</u>
  - Node **i** receives $y_{i+1} = (R_{i+1}+\ldots+R_n)$ from node **i+1** for contract redemption
  - It computes (R_i+y_{i+1}) and uses it to redeem contract with node **i-1**

More Solutions

- The previous solution [Malavolta et al, CCS'17] uses HTLCs
- **Avoiding HTLCs** [Malavolta et al, NDSS'19]:
  - Above idea can be generalized using "homomorphic" one-way functions (based on, e.g., DLOG) to avoid using HTLCs
  - Idea of "partial computation" can be used to also build solution using EC-DSA signatures (i.e., can also work with Bitcoin without scripts!)
- **Anonymity:** Using anonymoyous communication channels, the above solutions can also be used to achieve **"anonymous" multi-hop payment channels**