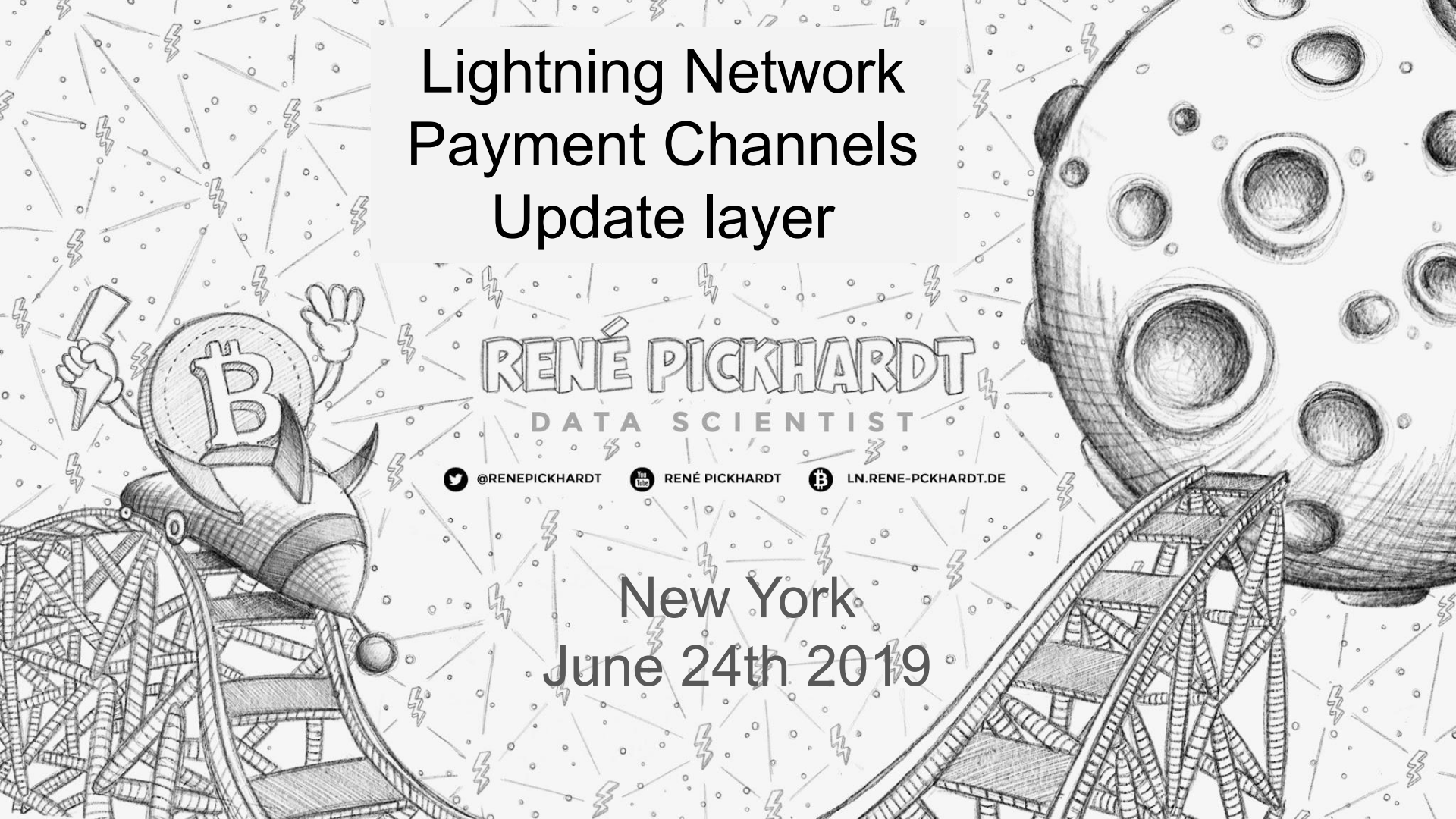
Lightning Network Payment Channels Update layer



RENÉ PICKHARDT

DATA SCIENTIST

@RENEPICKHARDT RENÉ PICKHARDT LN.RENE-PCKHARDT.DE

New York

June 24th 2019

Disclaimer

- Please note the copyright notice at the end of the slide deck
- The pdf version - which you probably read has some weird formatting problems.
  - https://docs.google.com/presentation/d/1-eyceLlSmcLpbPJLzj6_CnVYQdo1AUP3y5XD716U-Lg has the source files without formatting problems and with animations
- This slidedeck is crowd funded (see last slide) if you wish to learn more or contribute
- I do not take any guarantee that the information in this slide deck are 100% correct.
  - Sometimes I made simplifications: which I point out
  - Also I could just have misunderstood something or just made a mistake

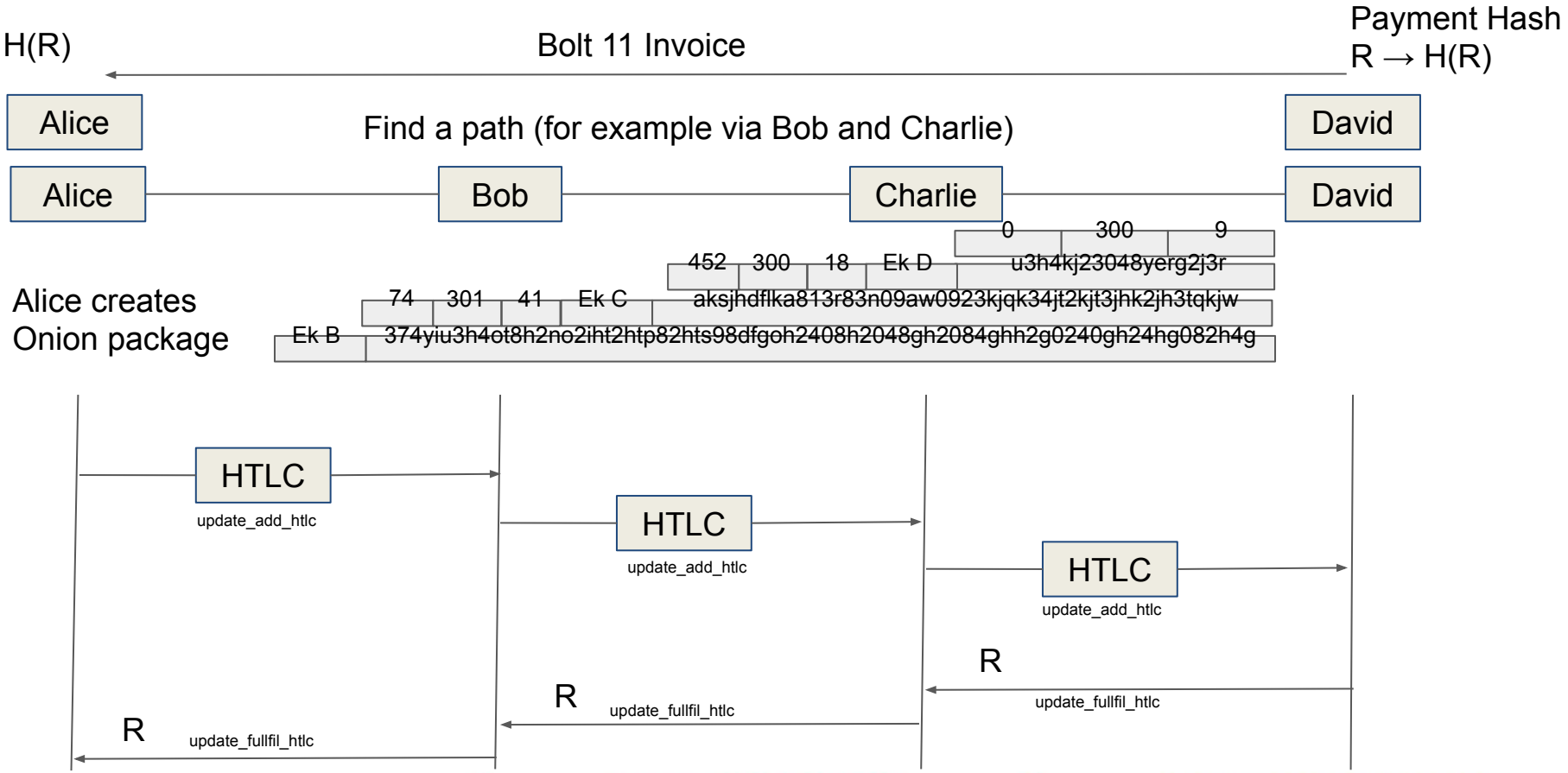
Outline

- Myths about Routing
- The Payment Process on the Lightning Network
- Trustless Payments via HTLCs
- SPHINX Mix-Format and Onion Routing

Myths about routing

- Unsolvable
  - Certainly not true
- NP hard or NP-complete
  - Plain wrong
  - No most pathfinding algorithms have a worst case of cubic runtime
  - However:
    i. Still computationally heavy
    ii. Maybe infeasable for cheap / small hardware and low bandwidth devices
- Relies on trust
  - HTLCs operate trustless
  - Nodes can misbehave to interrupt routing though
- Needs central nodes
  - No research given but SNA suggests quite the opposite
- Has no problems
  - E.g. spamming / delaying htlcs is a principal problem

After this talk you will hopefully understand this slide!



The Payment Process on the Lightning Network

Comparing the payment process of on chain Bitcoin and off chain Lightning Network payments

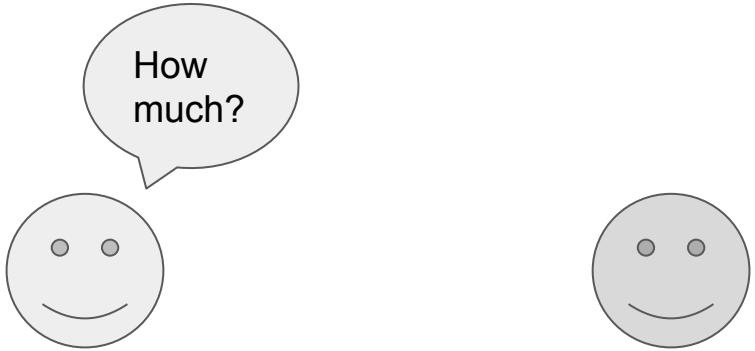
1. Bitcoin address is like a bank account number
2. Bitcoins can be sent to that address
3. Payment requests in Bitcoin exist to have a smoother User experience
4. Lightning cannot send money to another lightning node without invoice
   - unless you use dirty low level routing tricks (https://www.youtube.com/watch?v=Dwl-0cY6KkU)
   - Spontaneous payment extension by lnd in progress
5. In Lightning the recipient or payee first has to issue an invoice
6. The payer in lightning pays the invoice in return for a secret preimage
   - You could probably call it a receipt.

What does it have to do with routing and the fact that lightning is a network?

Also look at: https://www.youtube.com/watch?v=Ol12GrAy8yk

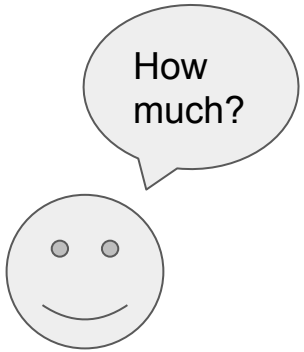
Direct physical Payment with cash

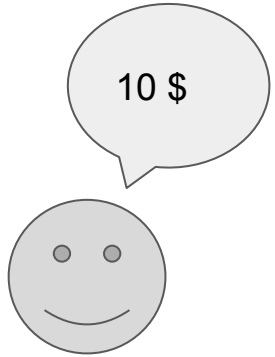
1. Ask for the price



Direct physical Payment with cash

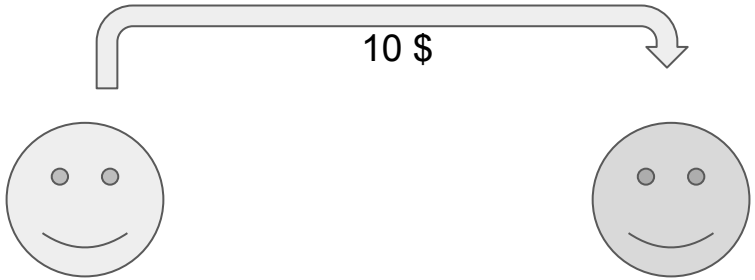
1. Ask for the price





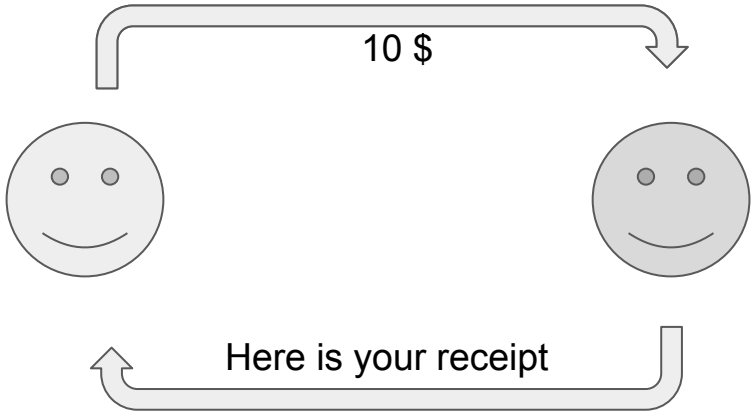
Direct physical Payment with cash

1. Ask for the price
2. Give the money



Direct physical Payment with cash

1. Ask for the price
2. Give the money
3. Get the receipt

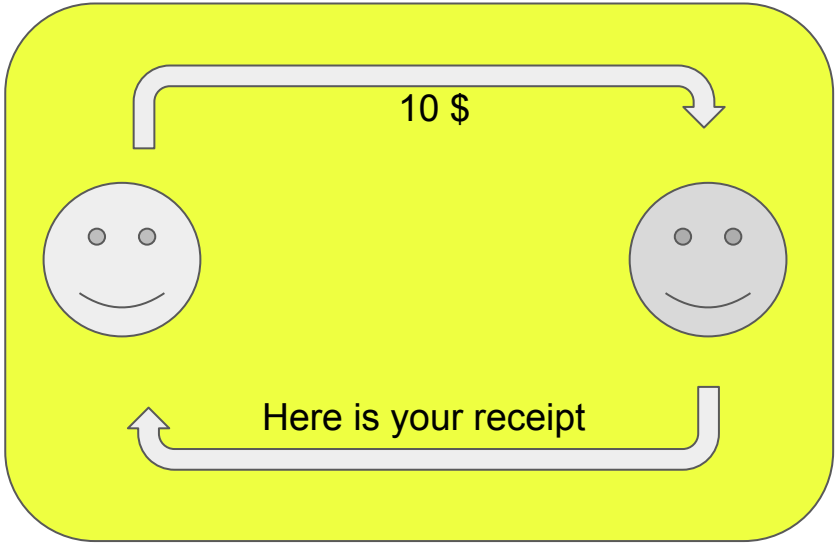


Direct physical Payment with cash

1. Ask for the price
2. Give the money
3. Get the receipt

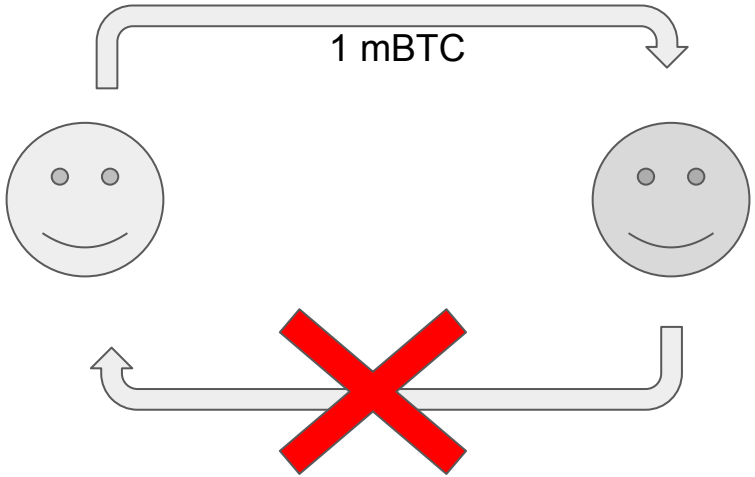
Step two and three are atomic.

(At least ideally they should be atomic)



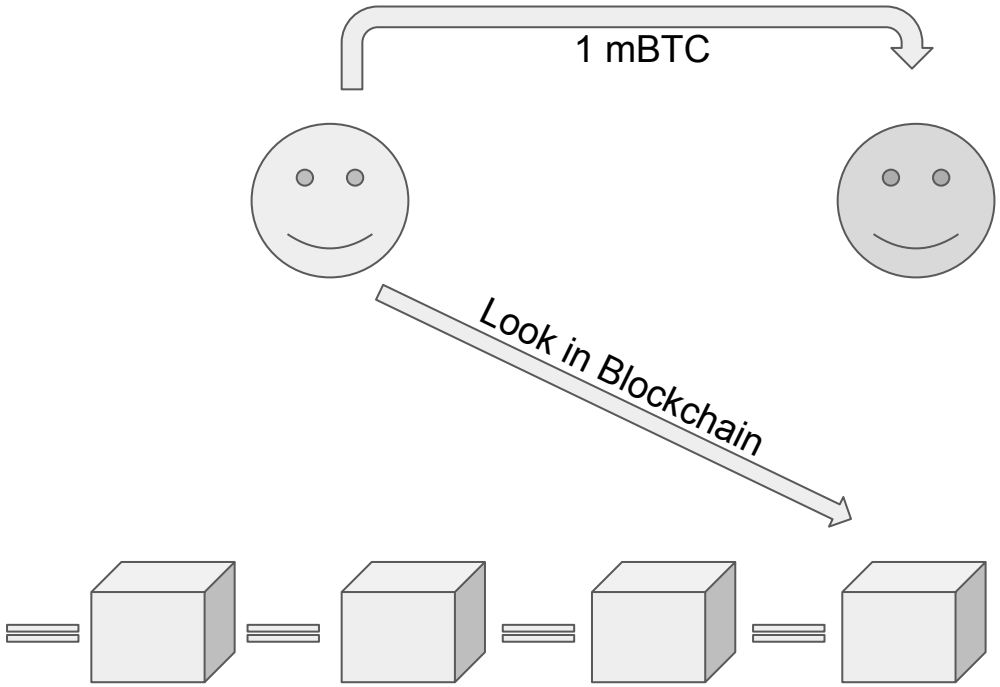
Direct on Chain Payment with Bitcoin

1. Ask for the price
2. Broadcast transaction



Direct on Chain Payment with Bitcoin

1. Ask for the price
2. Broadcast transaction
3. See within blockchain



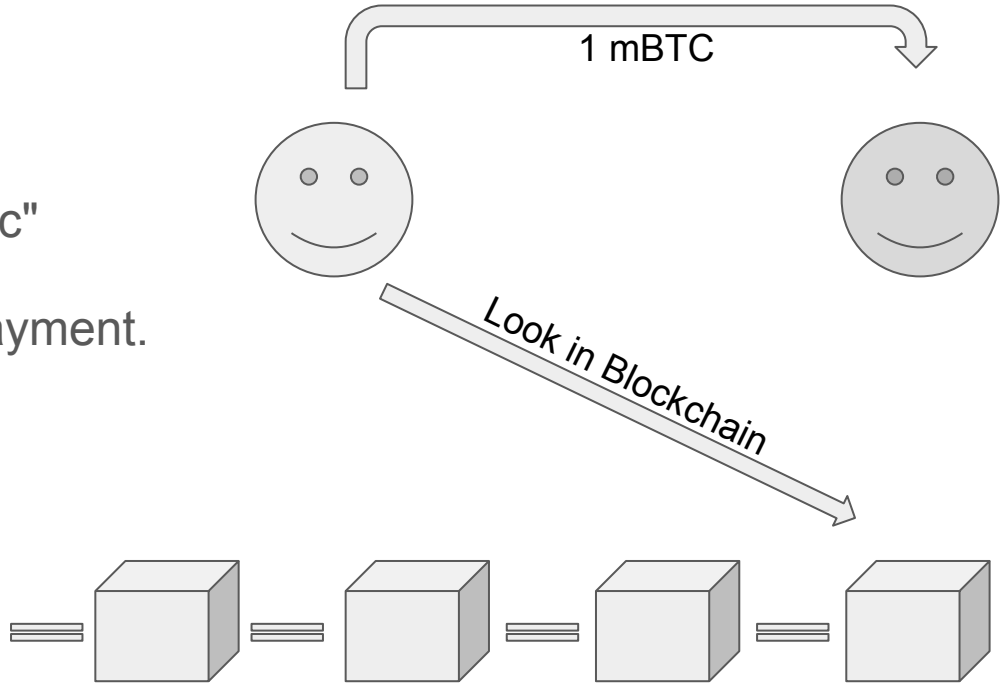
Direct on Chain Payment with Bitcoin

1. Ask for the price
2. Broadcast transaction
3. See within blockchain

Payment is still somewhat "atomic"

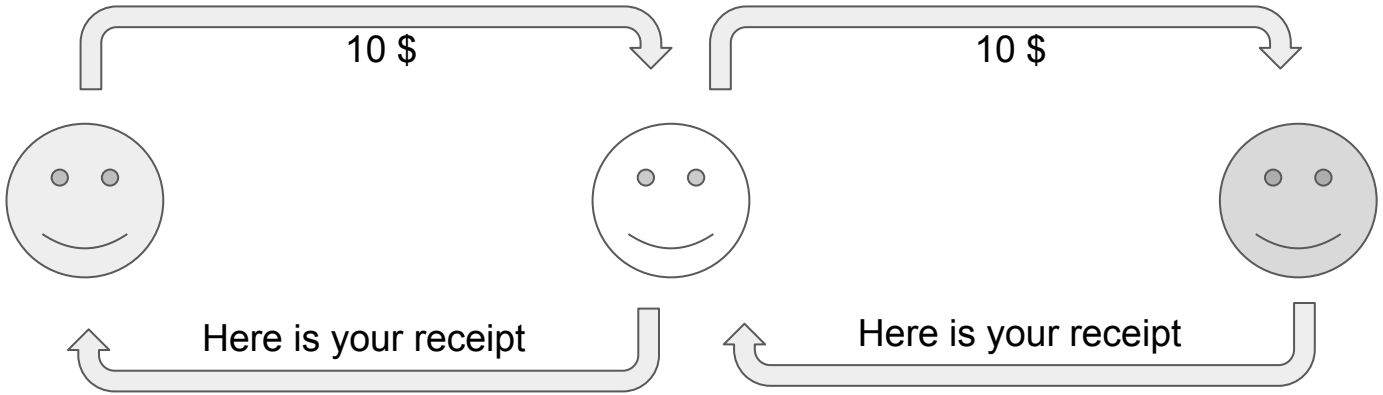
payee must have received the payment.

Payment can't be interrupted



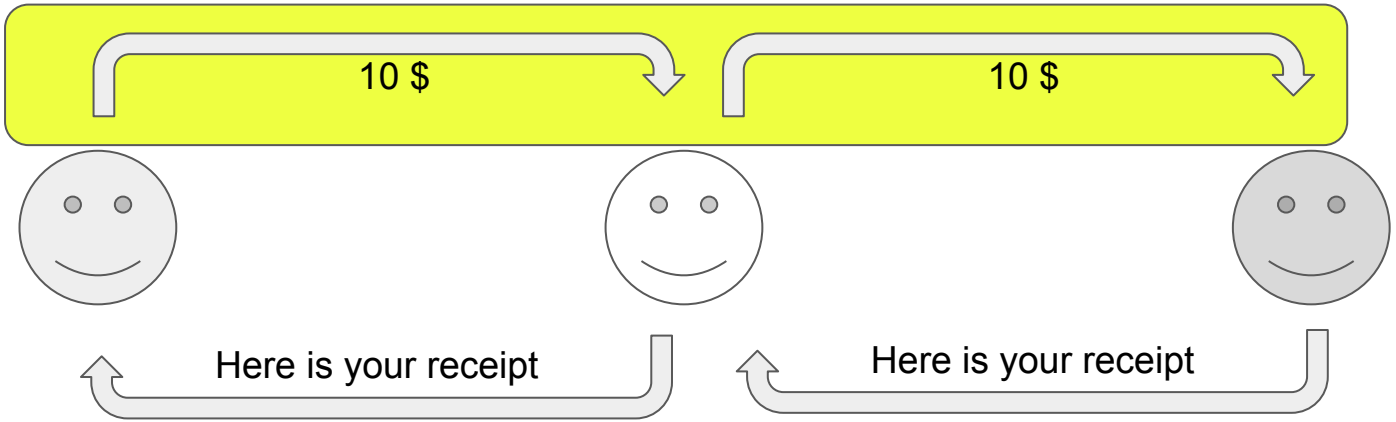
Indirect physical payment with cash.

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment



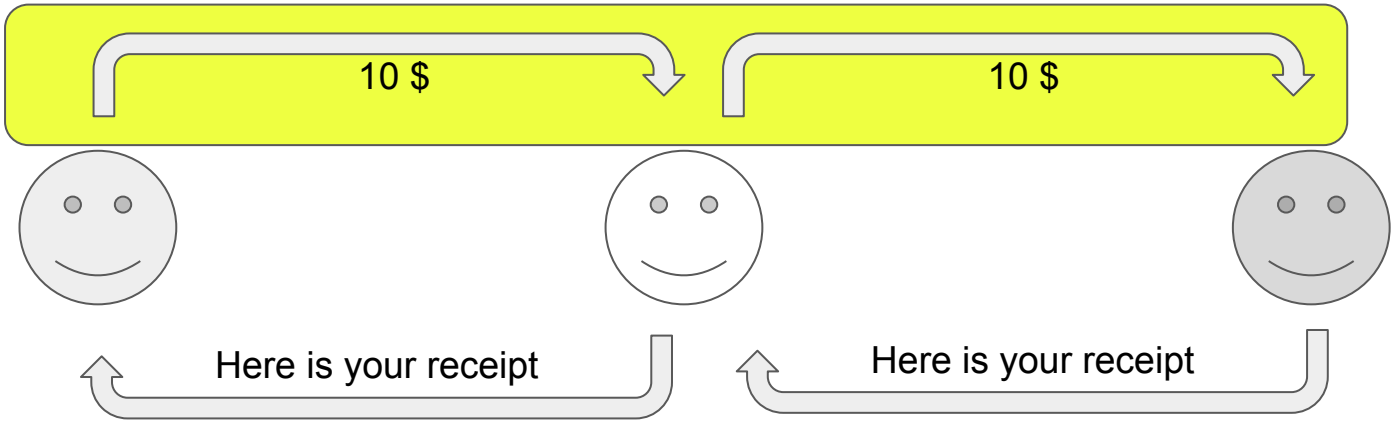
Indirect physical payment with cash.

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic



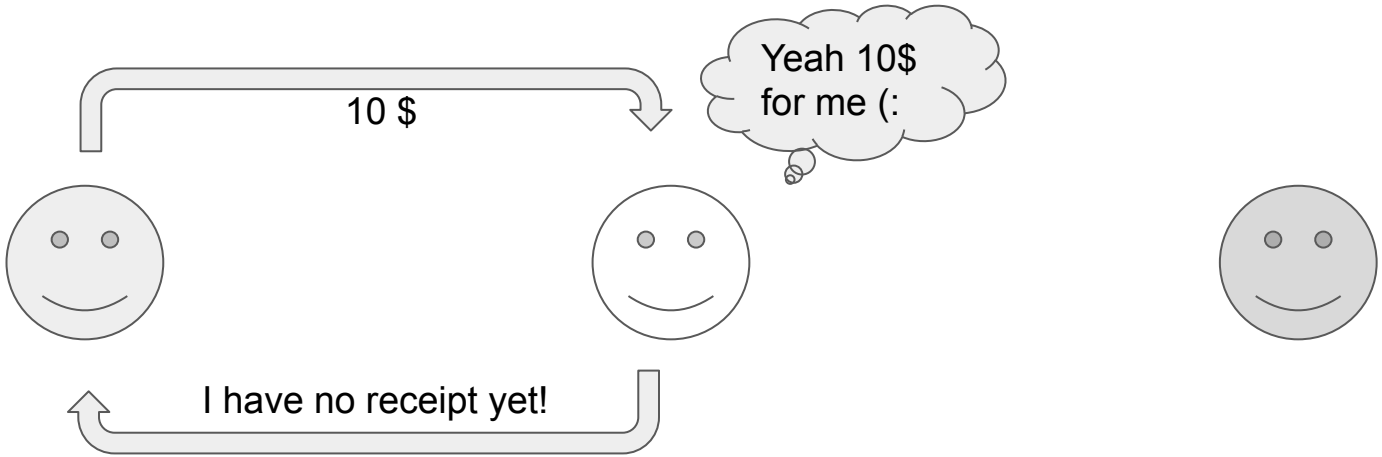
Indirect physical payment with cash.

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
   - Can it though?



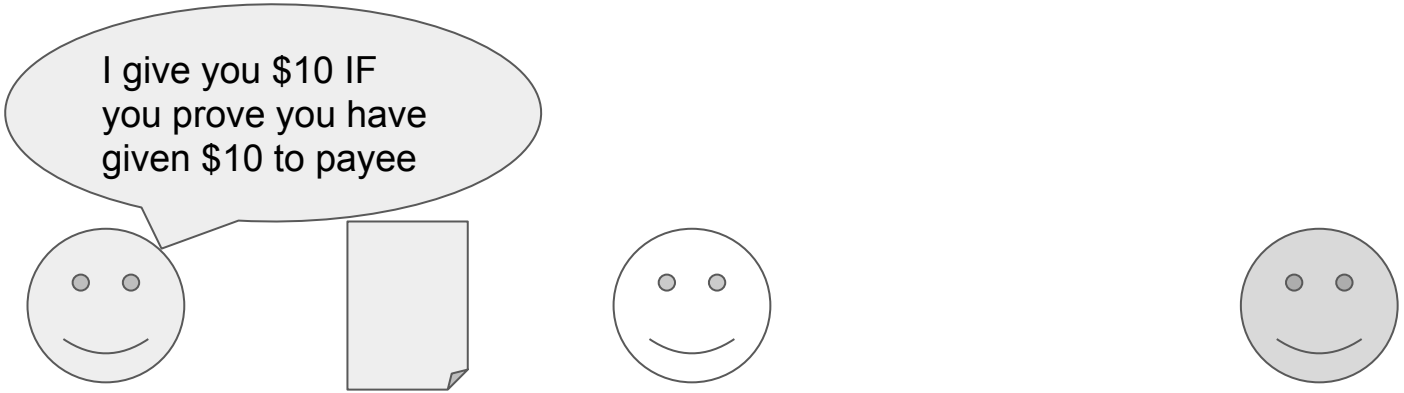
Indirect physical payment with cash.

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
   - otherwise



Indirect physical payment with cash.

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
3. Make a contract!



Indirect physical payment with cash.

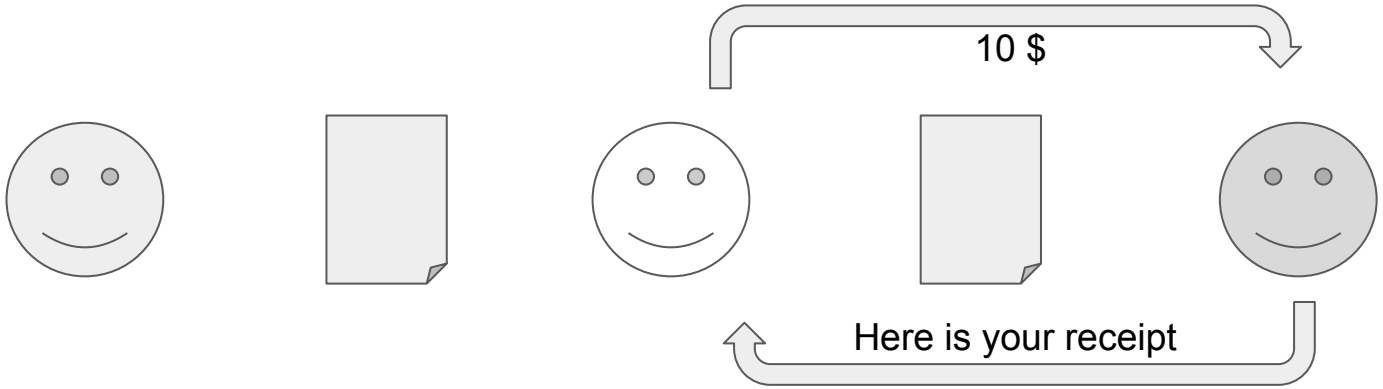
1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
3. Make ~~a~~ contracts!

Indirect physical payment with cash (+ service fee)

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
3. Make ~~a~~ contracts!

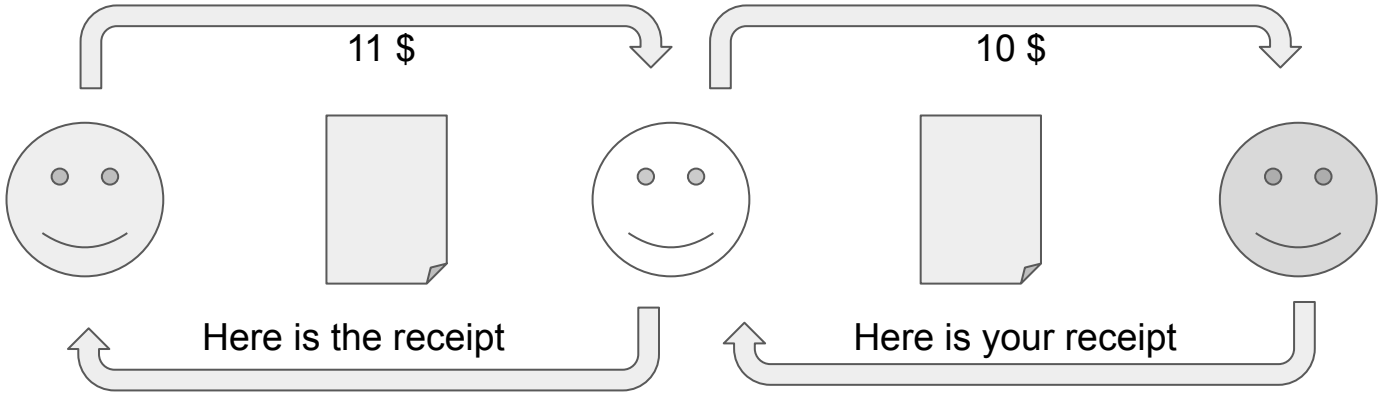
Indirect physical payment with cash (+ service fee)

1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
3. Make ~~a~~ contracts!



Indirect physical payment with cash (+ service fee)

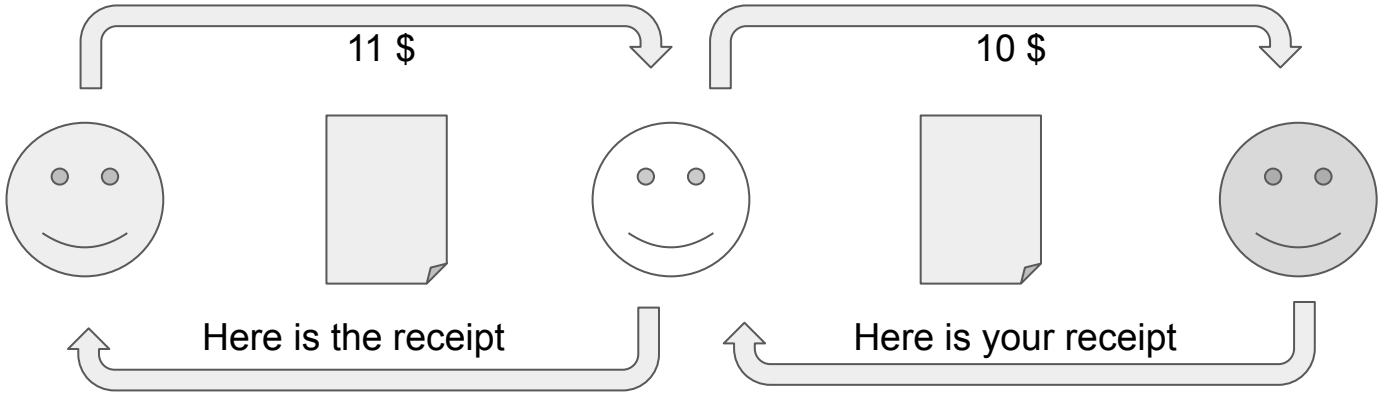
1. Assume there is a third party (e.g. Bank, another person) that should forward your payment
2. Should be atomic
3. Make ~~a~~ contracts!



Huge risks in the physical world

1. Badly written contract
2. Forgery with the receipt
3. Court case more time consuming than losing the money

→ Programmable smart contracts in Lightning (Hashed Time Locked Contracts)



Hashed time locked Contracts

1. It is just a regular bitcoin transaction with a special script inside
2. A conditional Payment
3. The transaction is locked for a certain time
    - Which it takes to forward the payment and get the receipt
4. The receipt is a secret random (in best case unique) value called preimage
5. The Hash of the preimage is called the payment hash
    - Can be seen as an identifier for this particular payment
6. It's like the legal contract from the cash setting but without the disadvantages
    - Cheaply enforceable by publishing the contract (bitcoin transaction) to the bitcoin network
    - Much cheaper than the court system
    - Not possible to have a misunderstanding

Hashed time lock contracts in payment channels

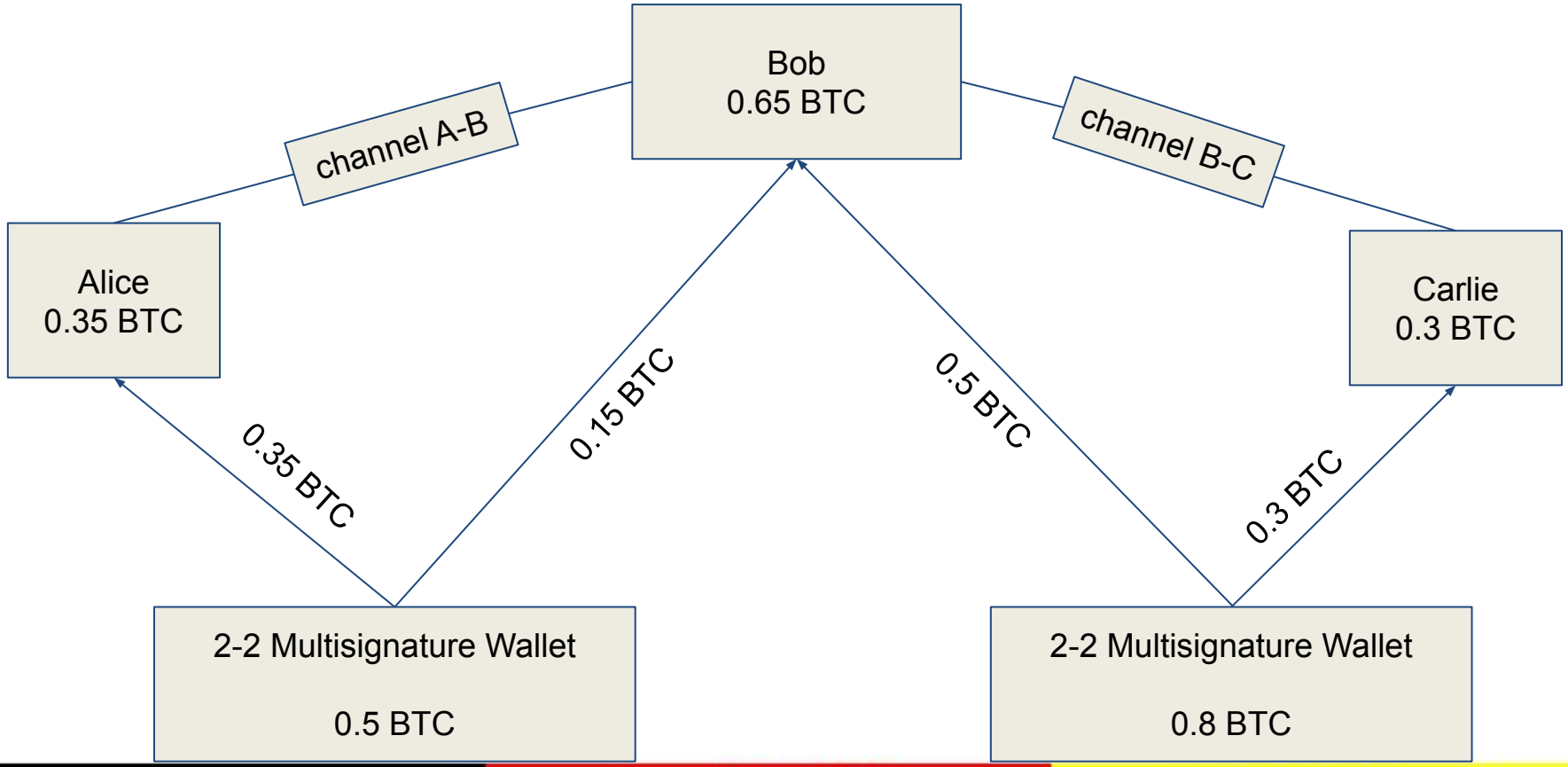
- Payment channels enable the direct atomic payment between two neighbors that share a channel
- Nodes build a network and payment in the network resembles the situation of indirect payments with cash
- Hashed Time Locked Contracts or HTLCs solve this problem
- HTLCs are just another output in the commitment transaction
  - They can be enforced on chain if channel fails
  - They can settle off chain if the preimage is provided

- Before explaining details of routing: How is the payment hash sent over?

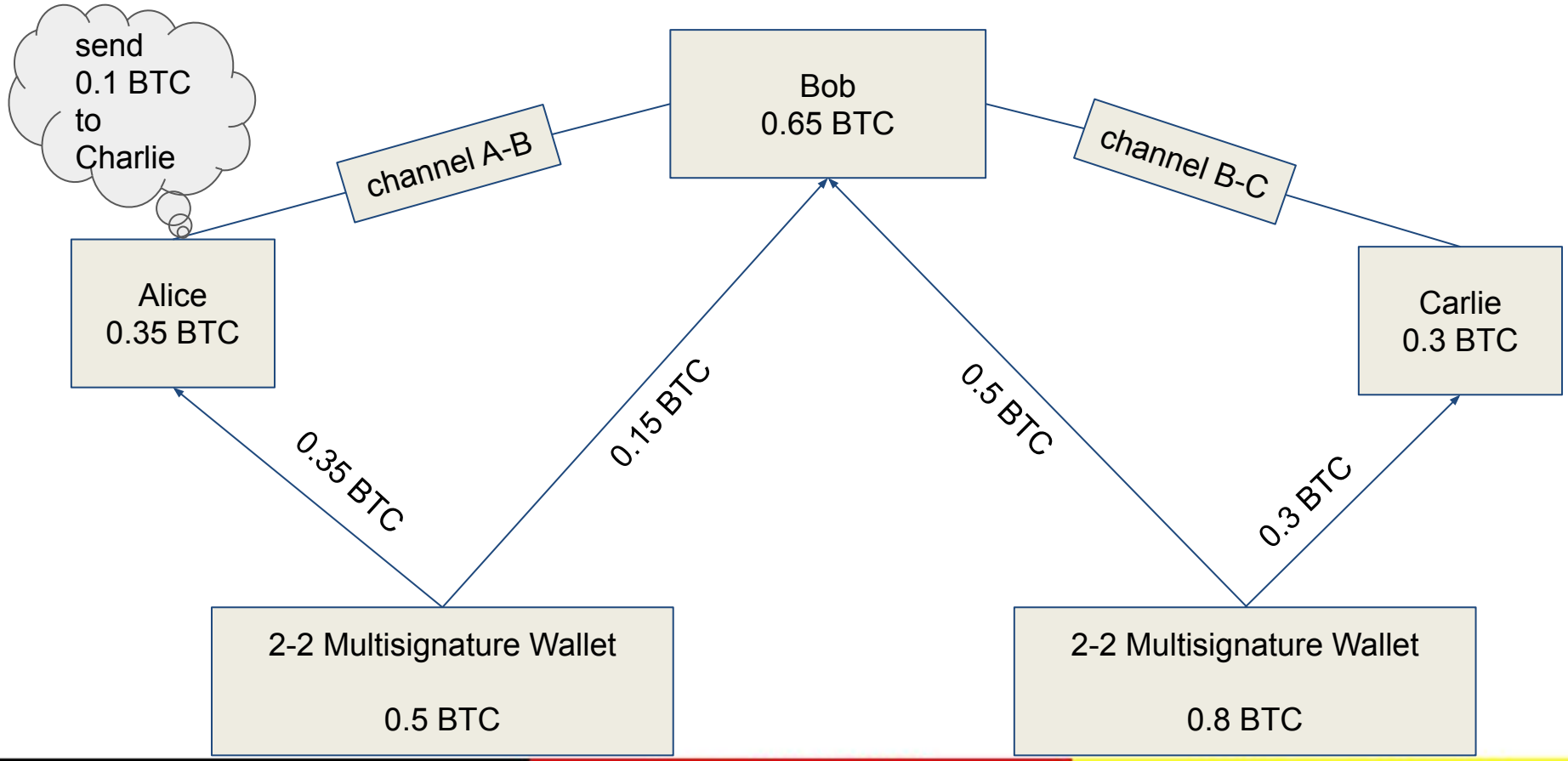
Trustless routing of payments through a network of payment channels

(Details in BOLT 03)

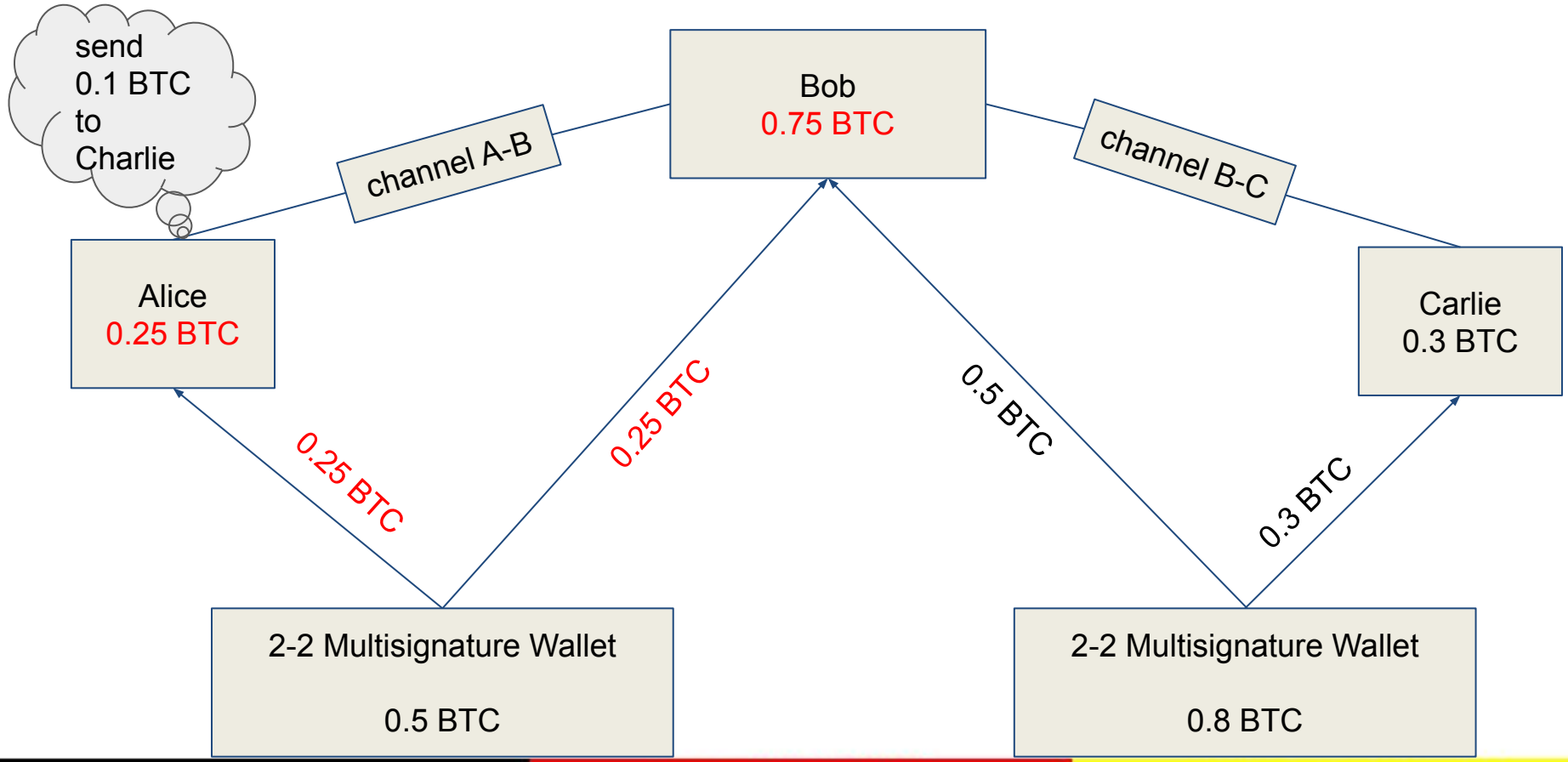
A trusting (Lightning) Network of payment channels



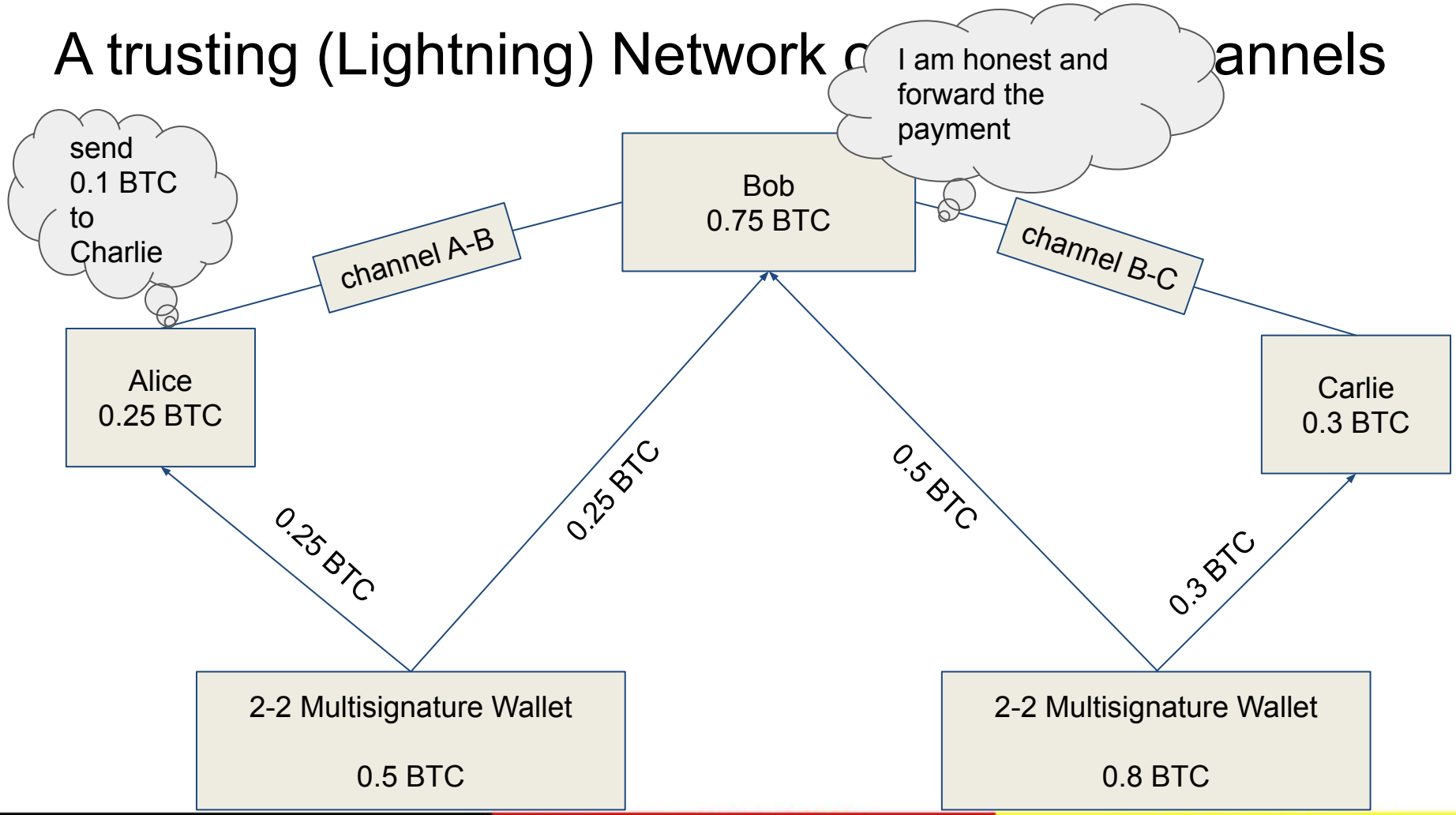
A trusting (Lightning) Network of payment channels



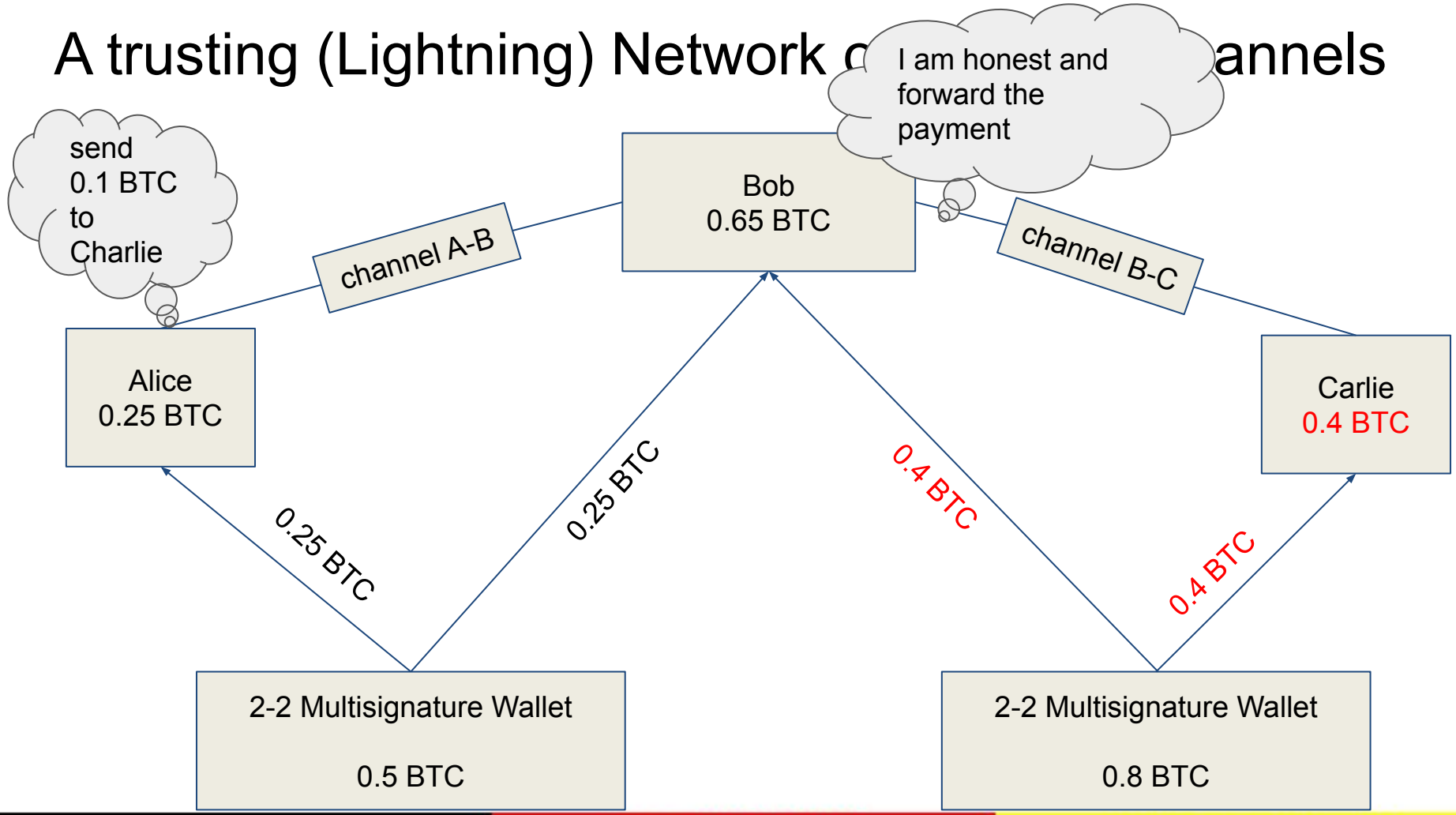
A trusting (Lightning) Network of payment channels



A trusting (Lightning) Network c annels



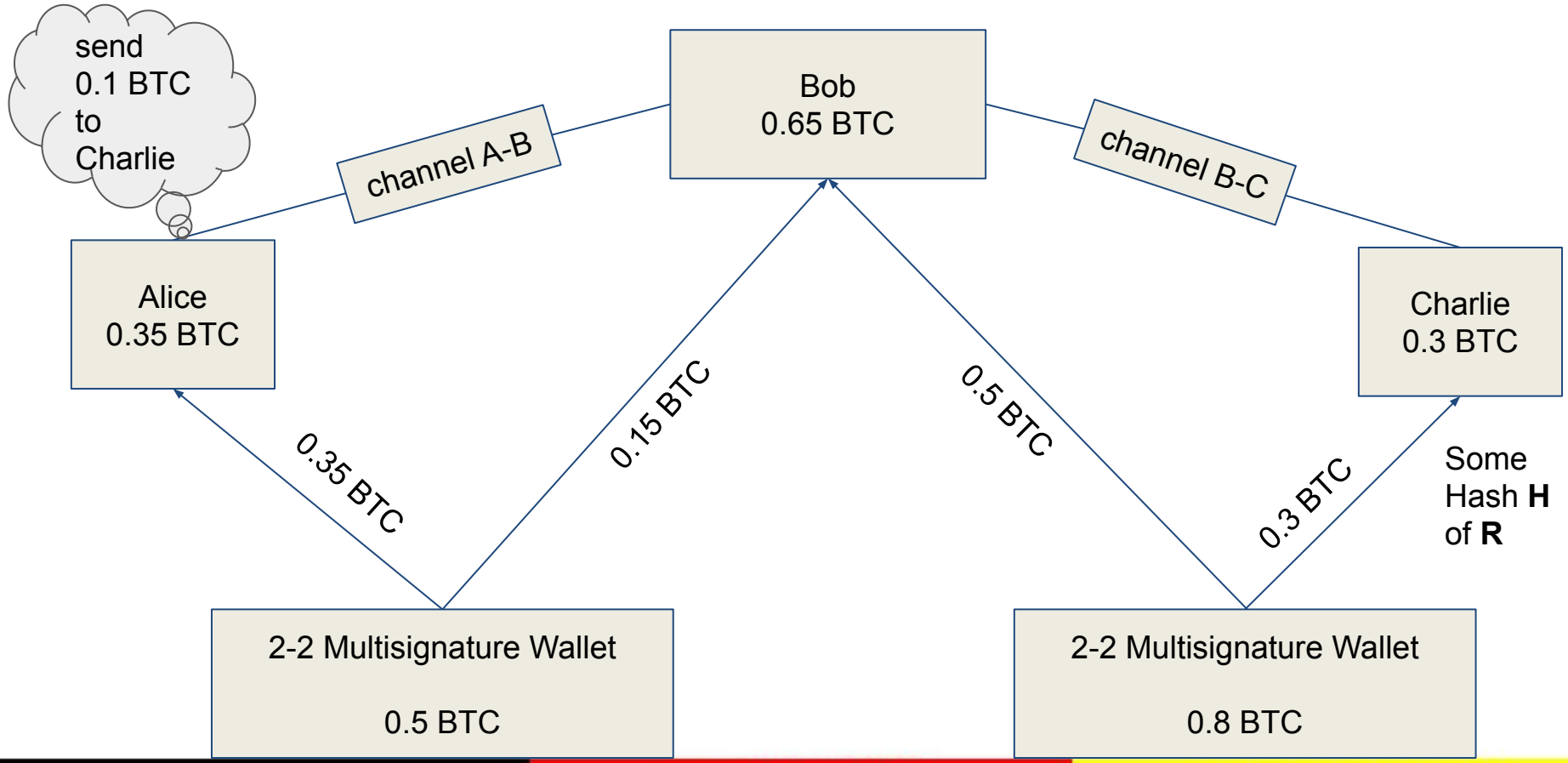
A trusting (Lightning) Network c annels



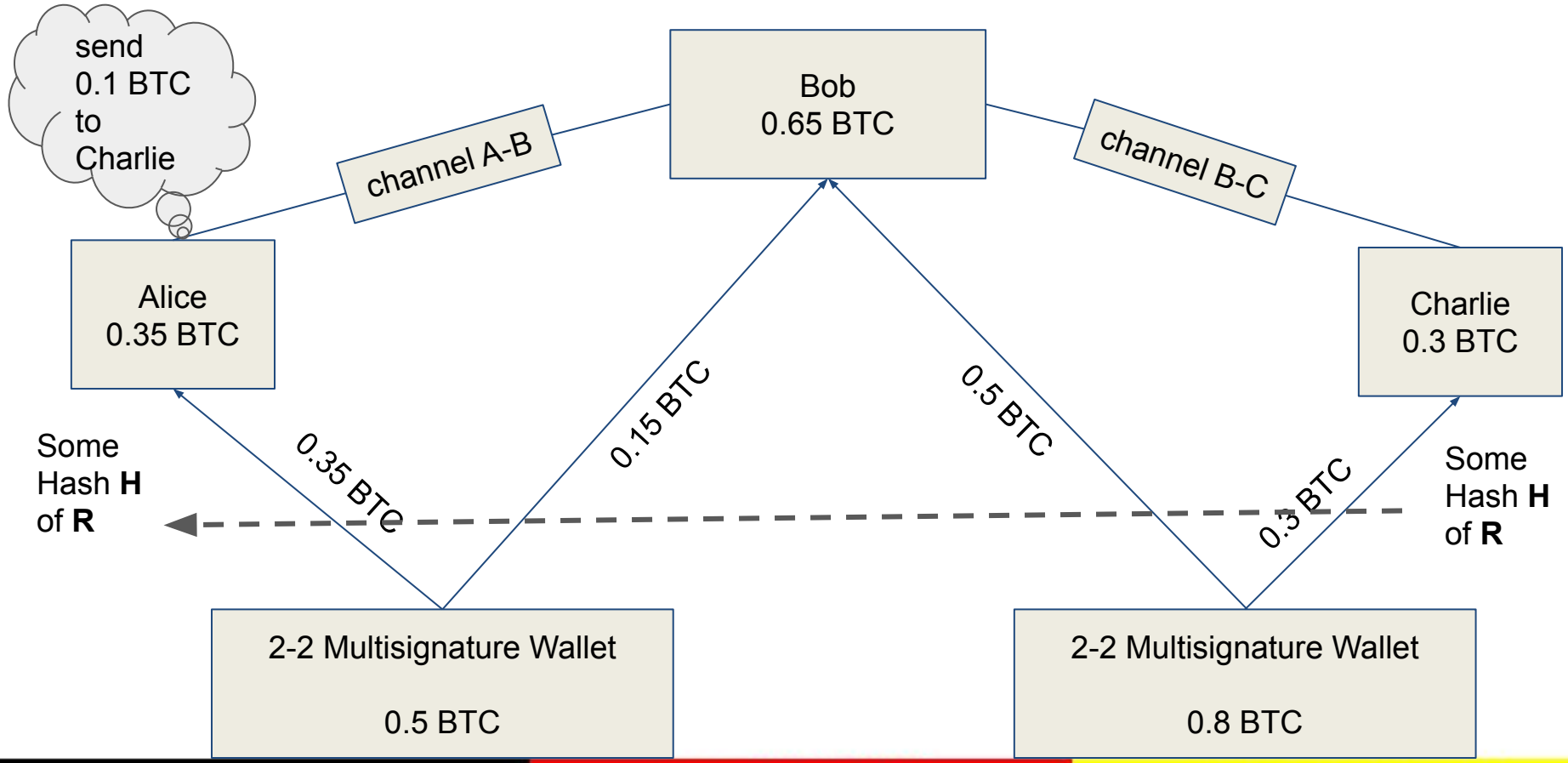
Alice needed to trust Bob to forward the payment

- Both payment channels needed to negotiate new commitment transactions for both sides
- Could we change the output of the CTXs so that routing works trustless?
- Idea (aka Hashed Time Locked Contract - HTLC):
  - Add another conditional output to the Commitment Transactions
  - It can only be spent by the recipient
    - within a certain time frame
    - if the recipient can provide the preimage of some hash
  - After the timeframe the sender can reclaim the funds

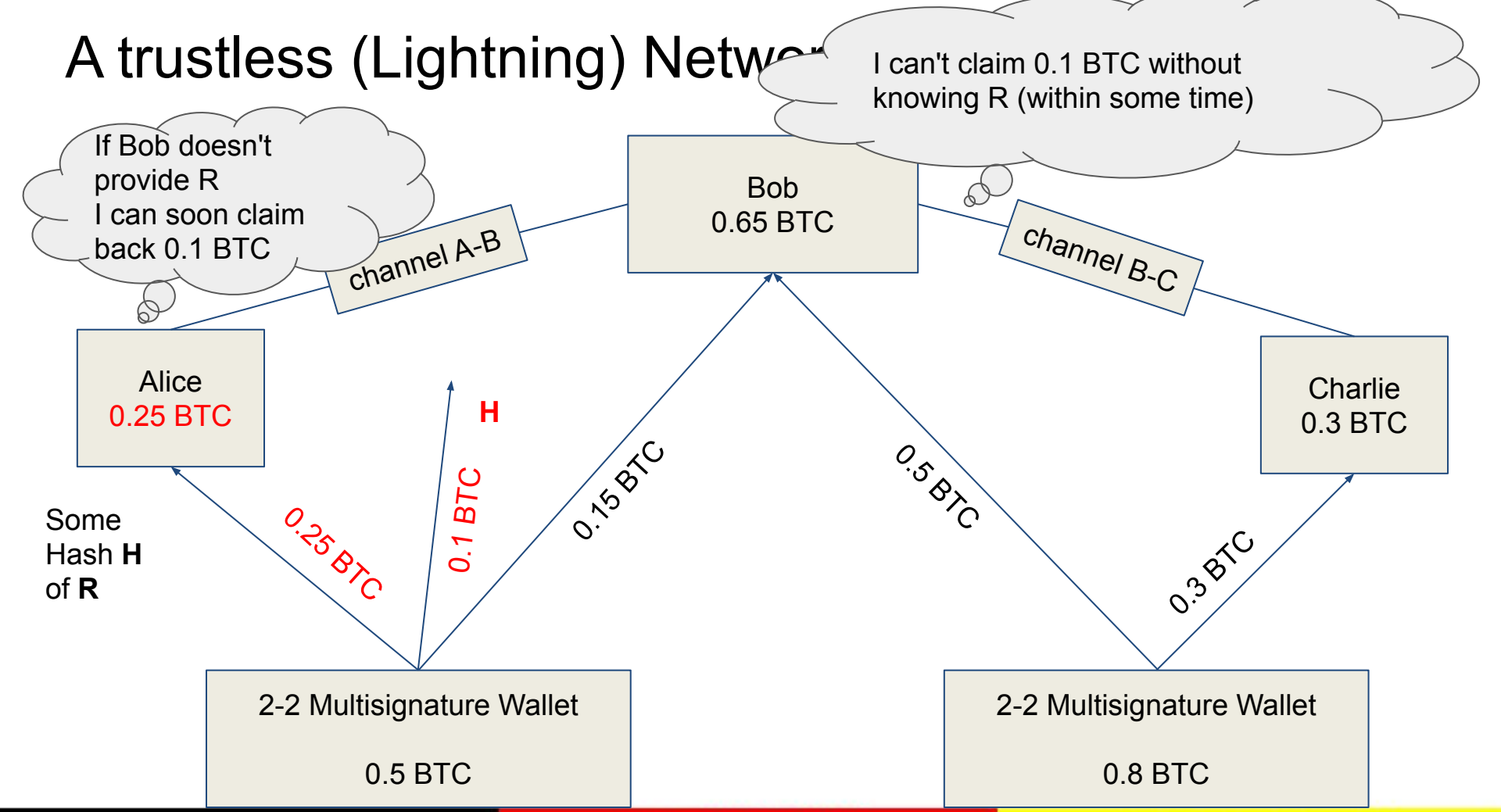
A trustless (Lightning) Network of payment channels



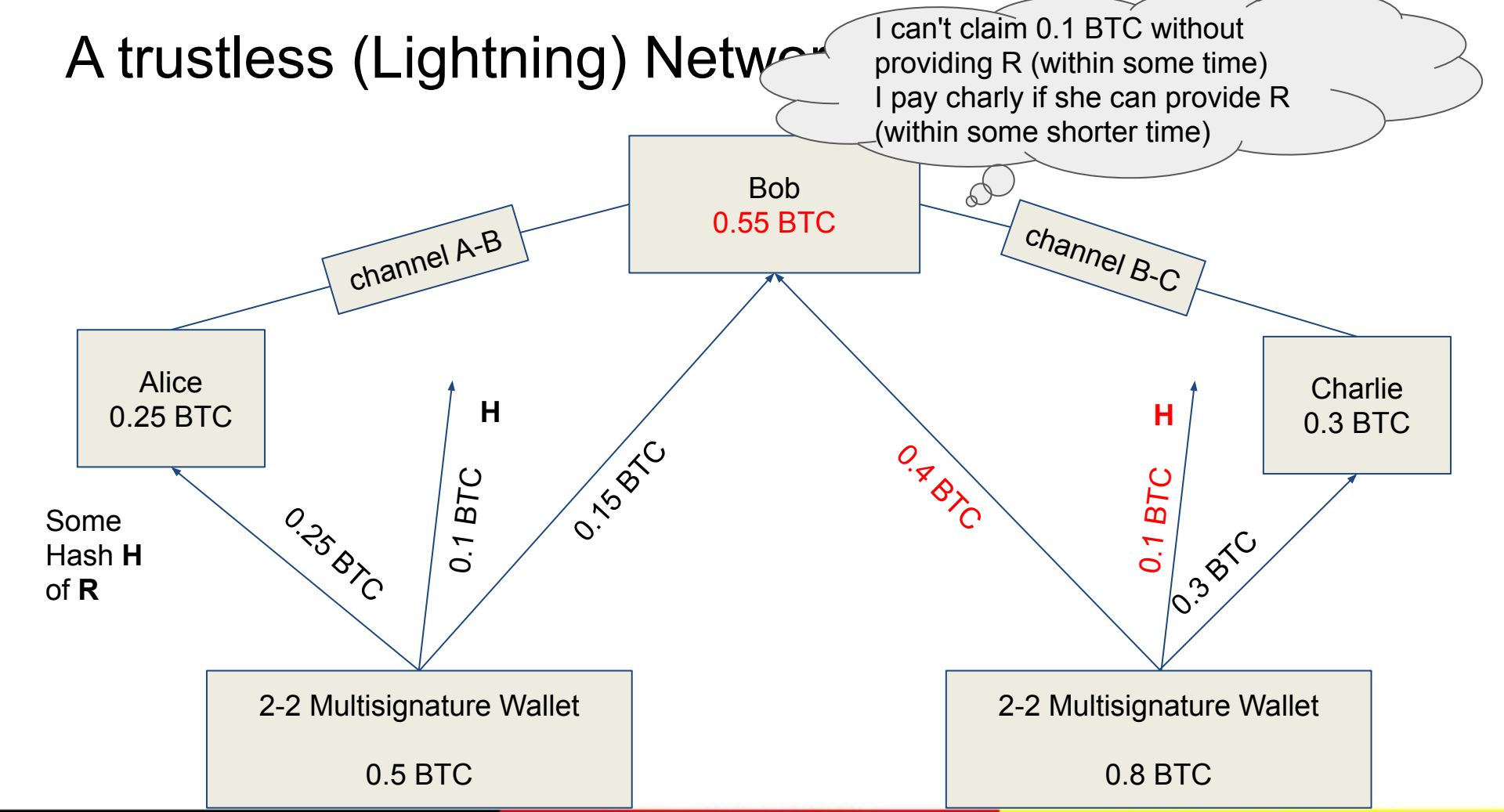
A trustless (Lightning) Network of payment channels



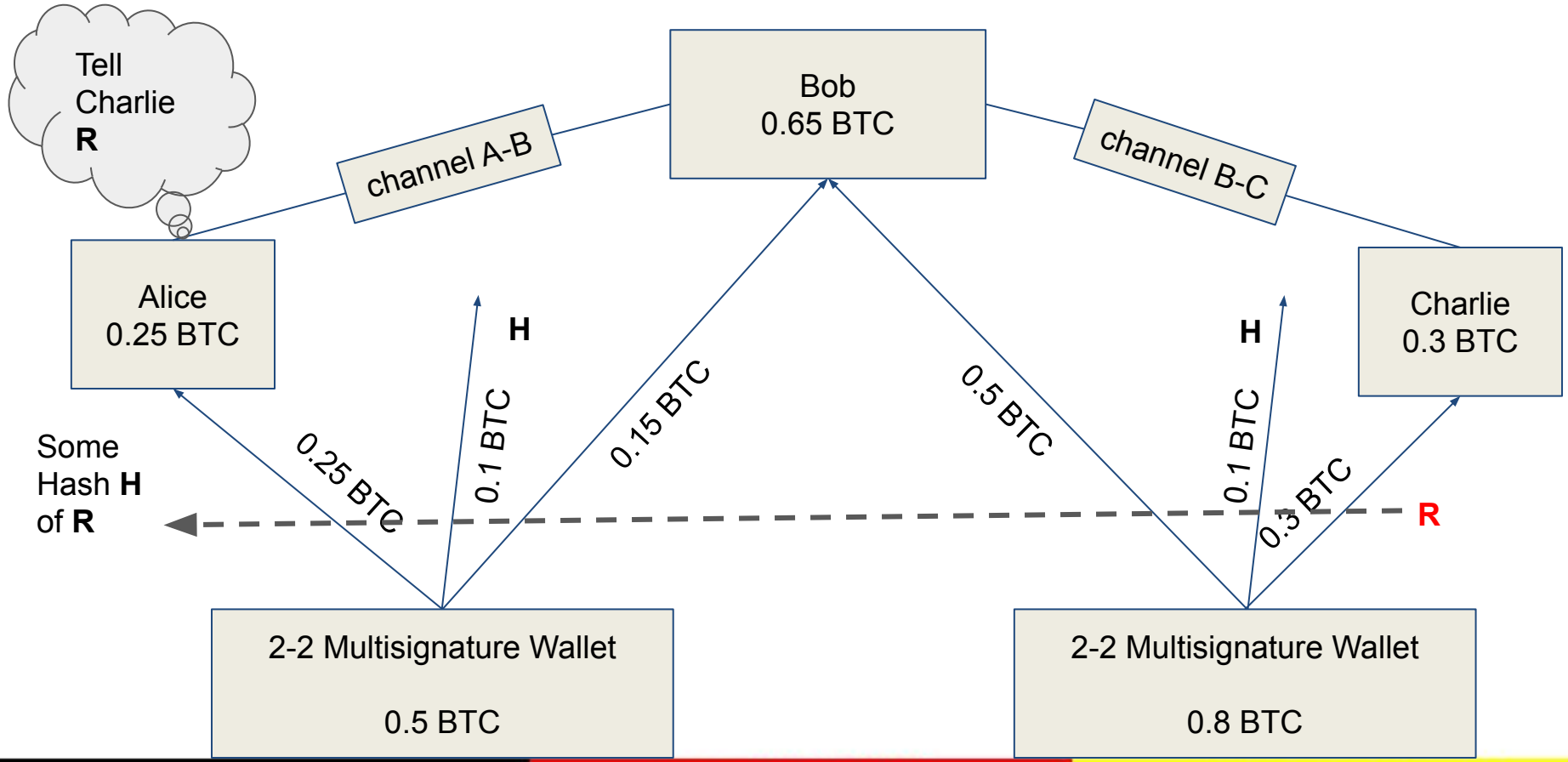
A trustless (Lightning) Netwo



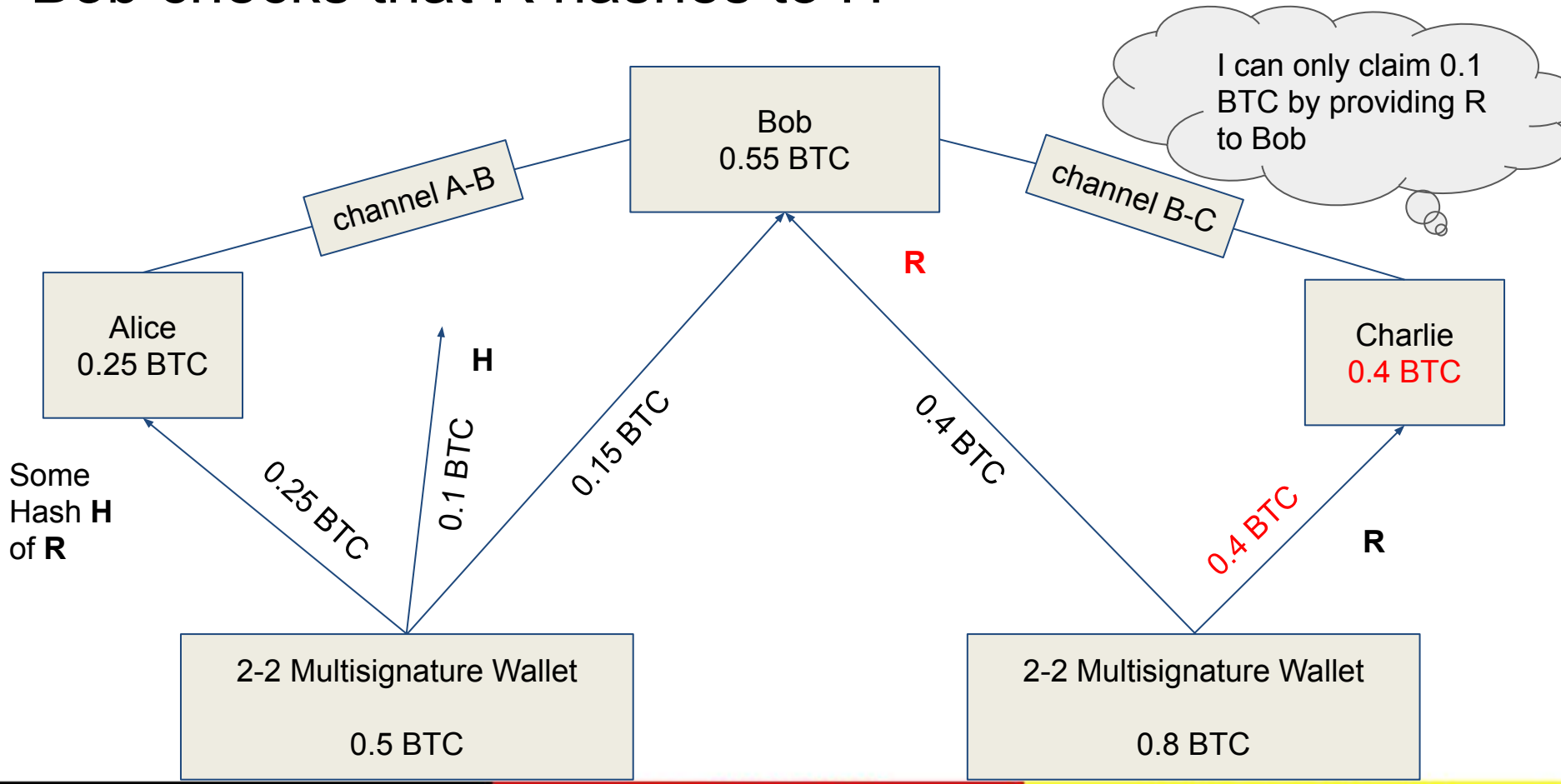
A trustless (Lightning) Netwo



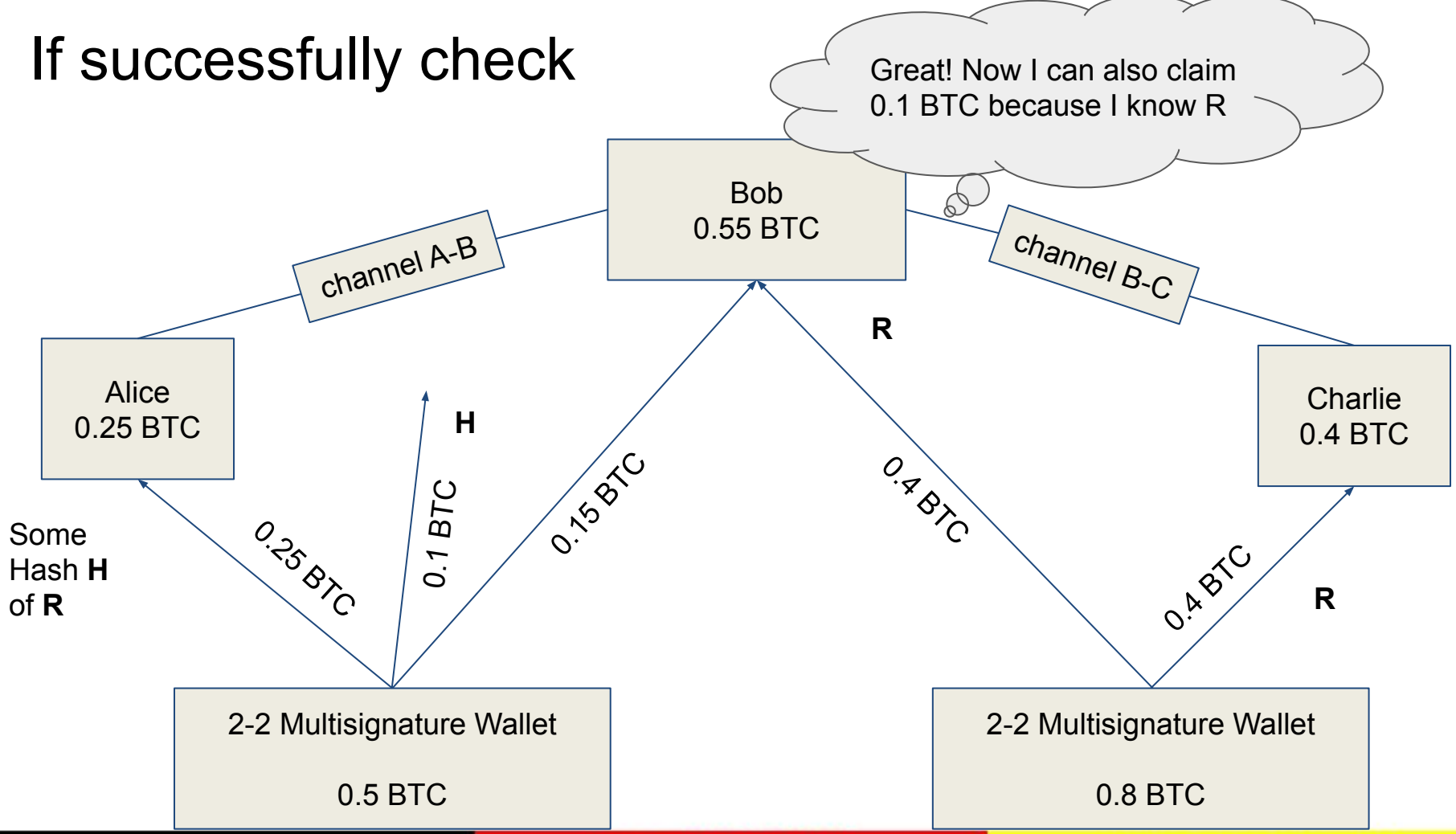
A trustless (Lightning) Network of payment channels



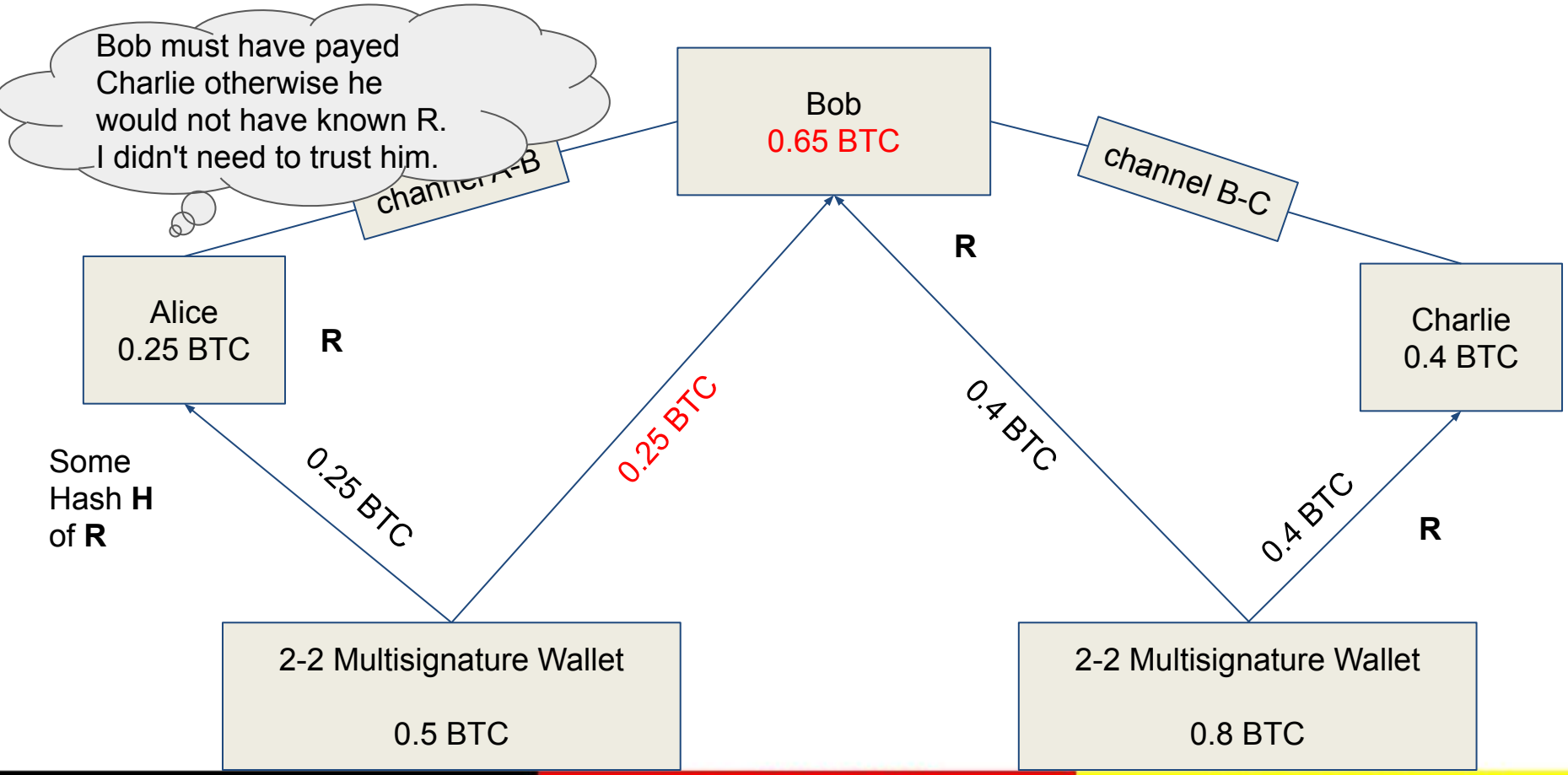
Bob checks that R hashes to H



If successfully check



Bob claims to settle htlc after releasing R



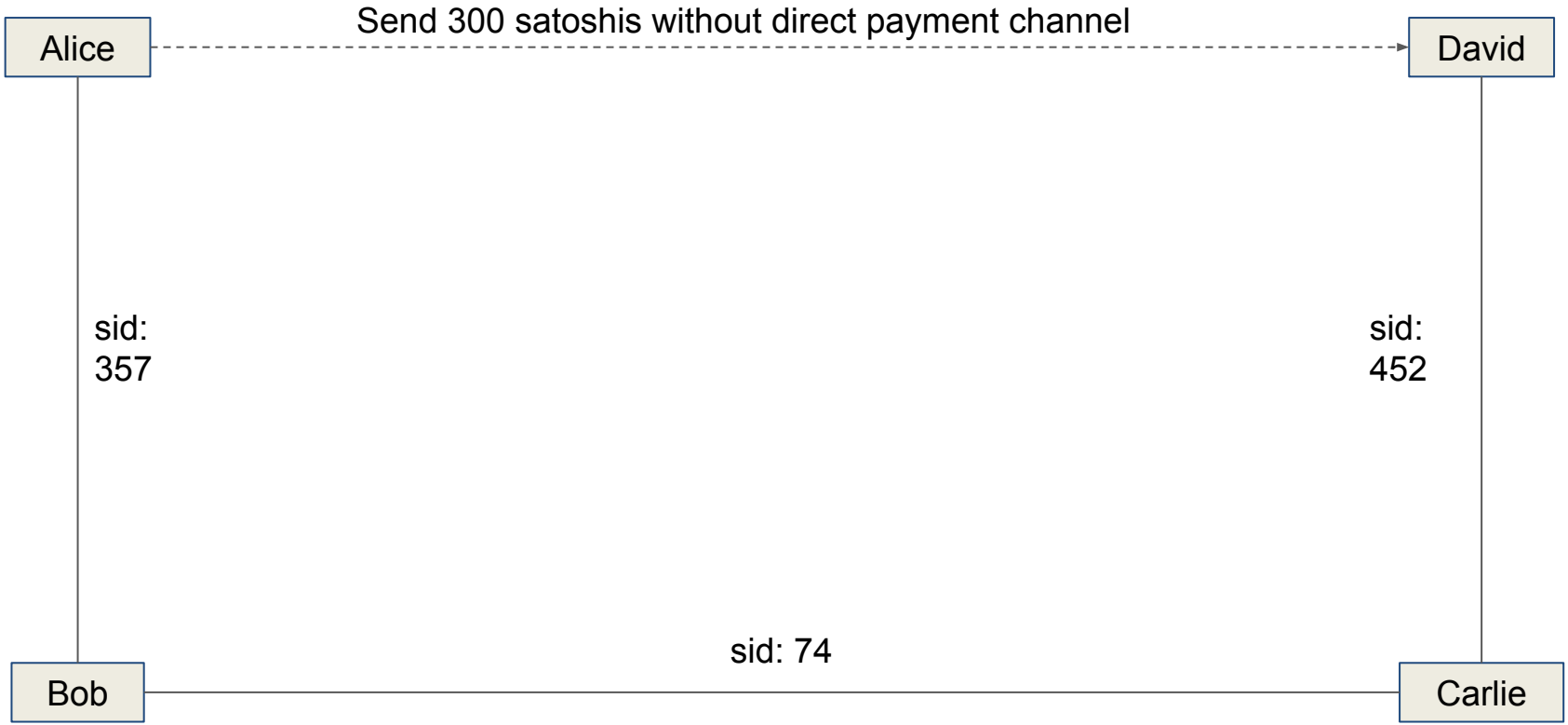
Source based Onion Routing - Sphinx Mix format Setting up htlcs

(Chapter 4 / BOLT 04)

Why using the SPHINX mix Format?

- Payer wants to be able to send money without being exposed
- Payee doesn't want to be exposed
- Routing nodes have to be able to send back an error message
- Routing nodes should only know as little information about the payment as possible
  - Payment hash (will stop with payment decorrelation)
  - An upper bound for the amount (will stop with AMP)
  - Know incoming channel
  - Know outgoing channel
- Routing nodes want to be able to verify the authenticity of the onion
  - HMAC checks at every hop
- Research community will tell you
  - It's very compact - uses only little data
- Better than slides: https://www.youtube.com/watch?v=toarjBSPFqI

Assuming enough capacity on all channels

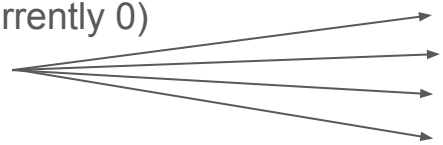


Create onions starting from David (no payment hash)

- Header
  - Version Byte
  - 33 Byte compressed secp256k1 pubkey
    - Not the sender Pubkey (node_id / static key)
    - But an **ephemeral pubkey from the sender for David**!
- Payload
  - 1300 Hops_data
  - 20 x 65 Bytes
    - 1: realm (currently 0)
    - 32 per_hop
    - 32: HMAC
    - … filler
- HMAC
  - 32 Byte to verify the integrity

Per_hop data includes the amount and route info

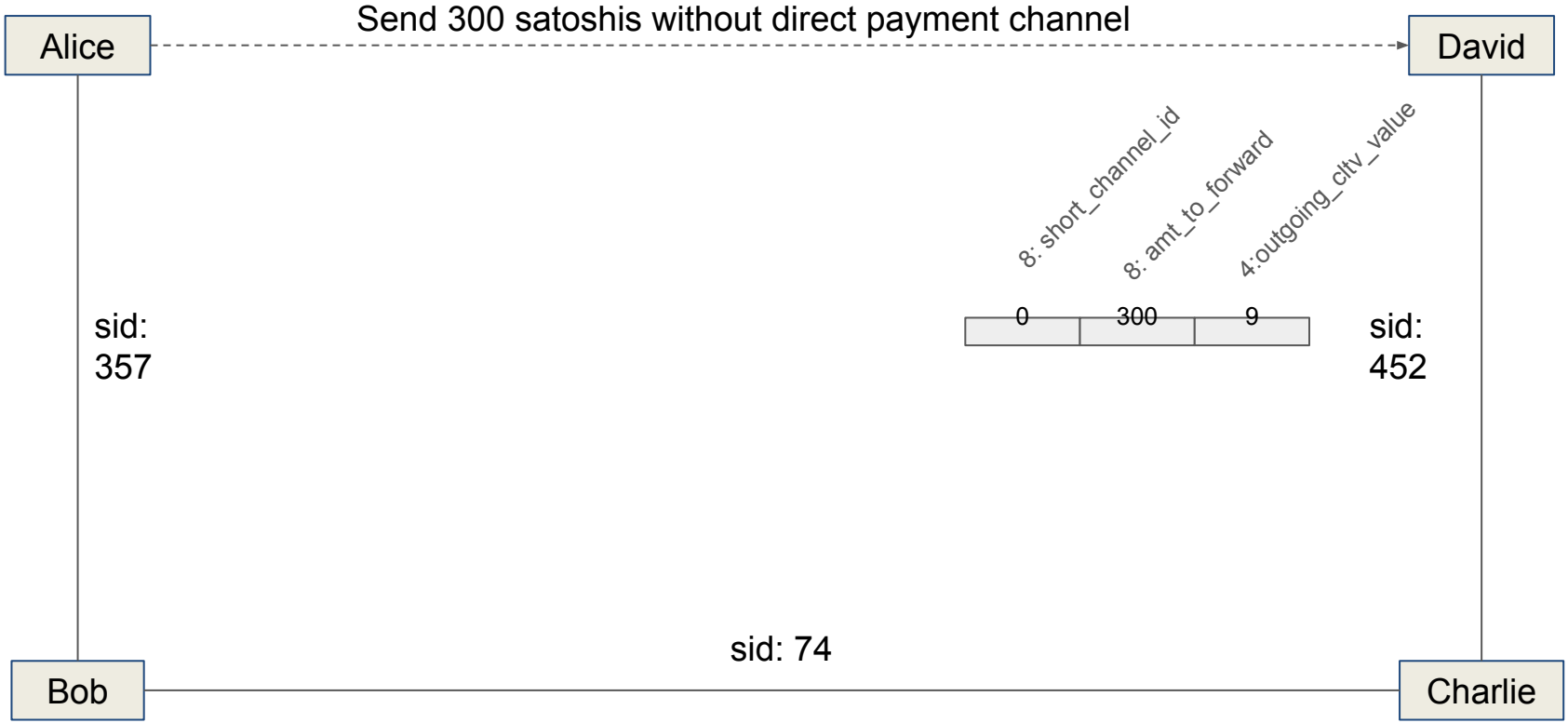
- Header
  - Version Byte
  - 33 Byte compressed secp256k1 pubkey
    - Not the sender Pubkey (node_id / static key)
    - But a pubkey from the sender for David!
- Payload
  - 1300 Hops_data
  - 20 x 65 Bytes
    - 1: realm (currently 0)
    - 32 per_hop
    - 32: HMAC
    - … filler



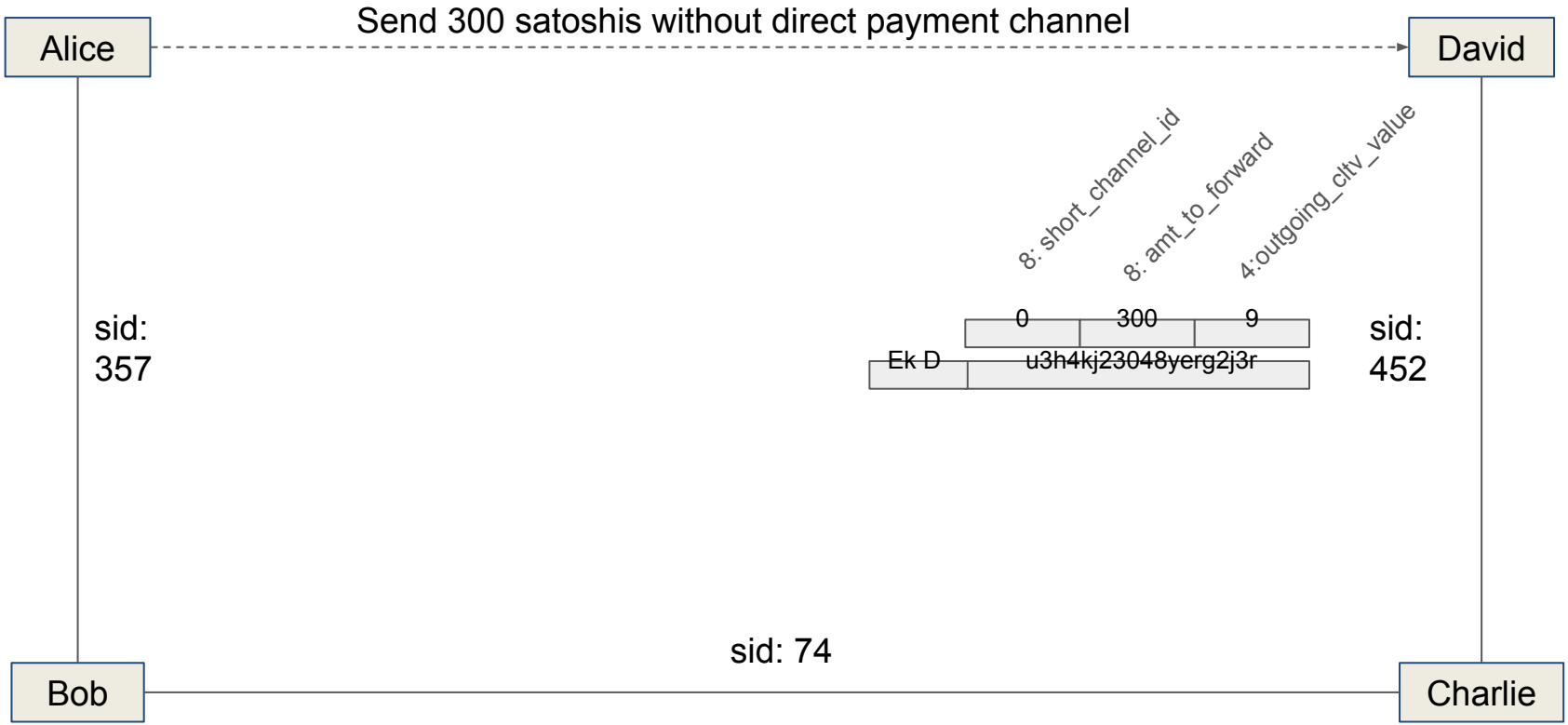
- 8: short_channel_id
- 8: amt_to_forward
- 4: outgoing_cltv_value
- 12: padding (for backwards compatibility)

- HMAC
  - 32 Byte to verify the integrity

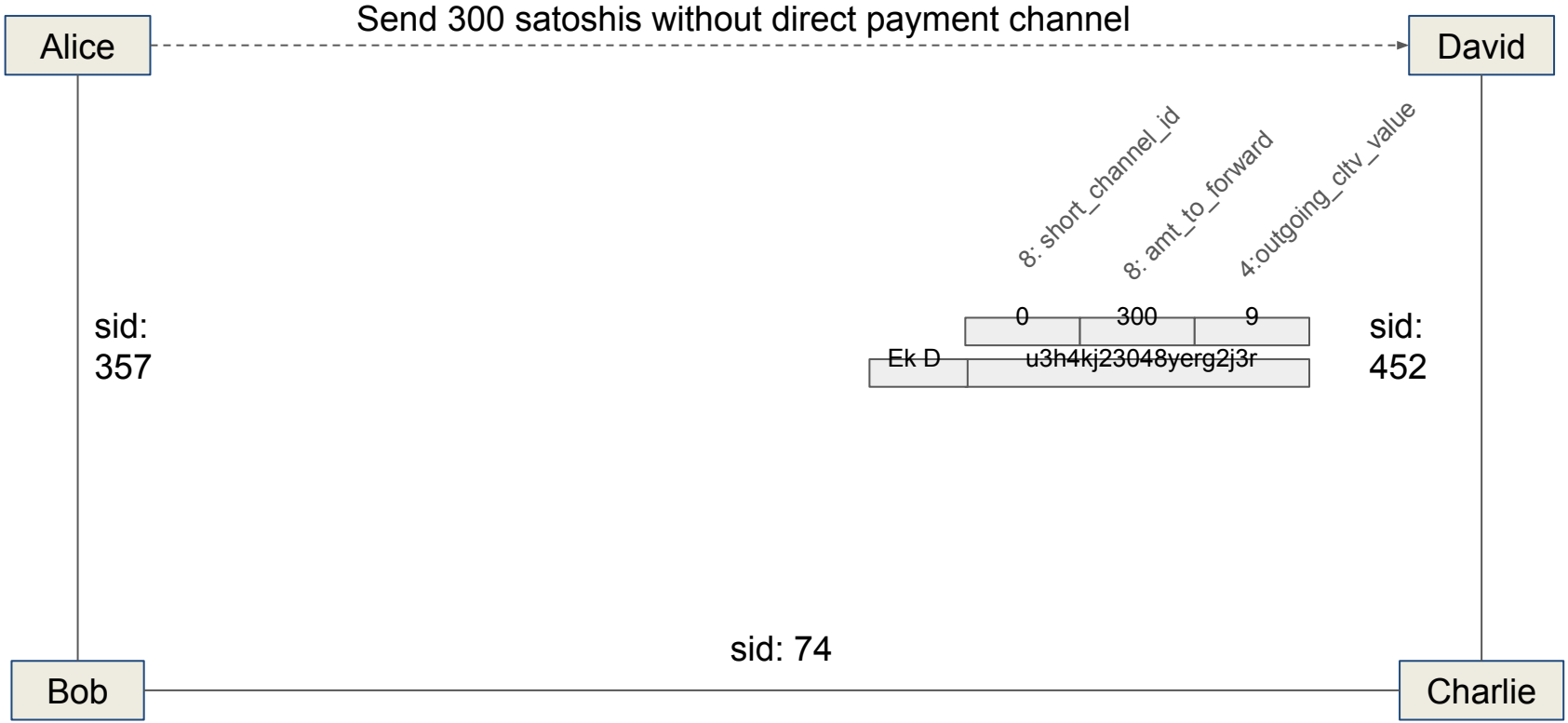
Per_hop payload for david (simplified!)



Onion for David (without HMAC and filler)



Why EK_D an ephemeral public key for David?



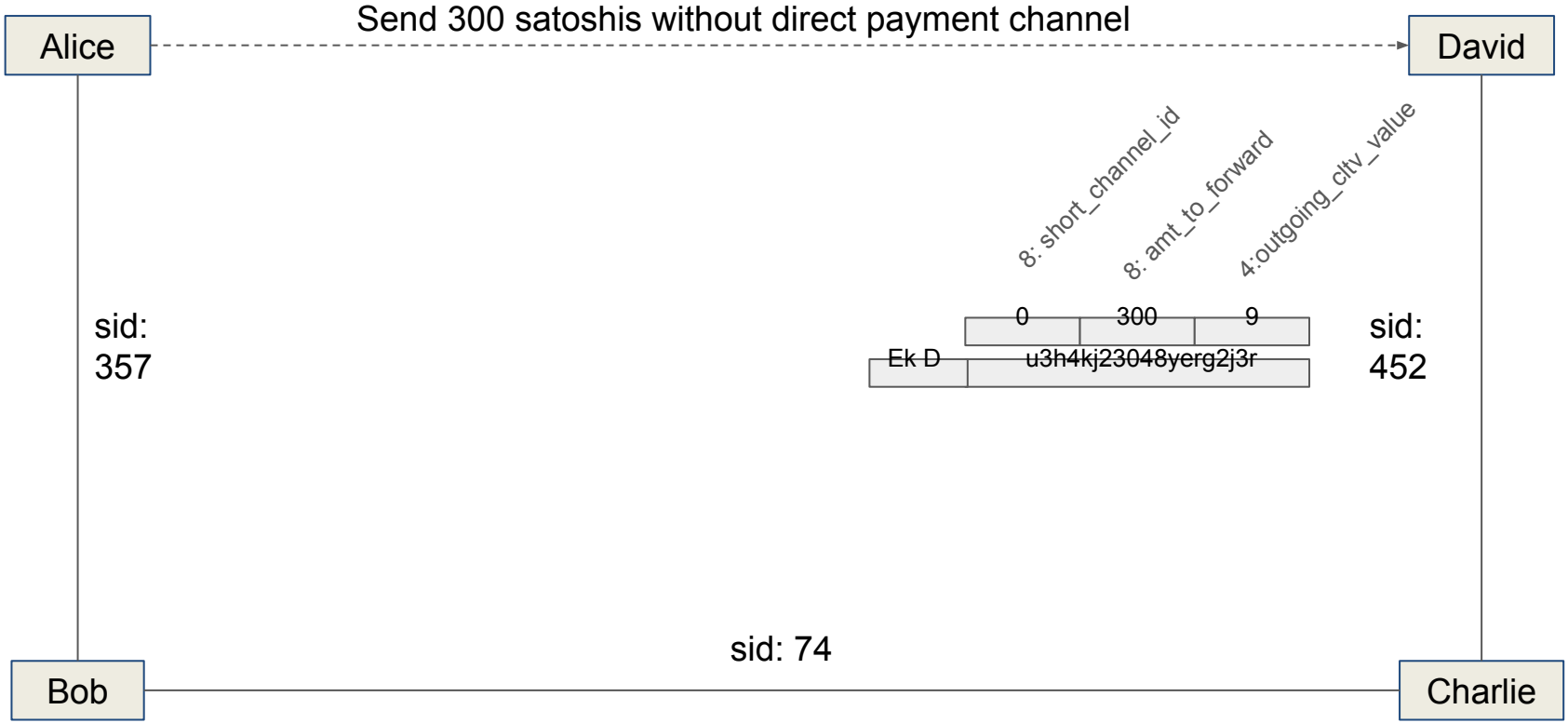
Secret sharing between Alice and David

- Let G be the generator of our elliptic curve of prime order p
- Let a, d < p be private keys for Alice and David
- aG = A, dG = D are the static node_id's for Alice and David
- Elliptic Curve Diffie Helmann Key Exchange: aD = dA = shared secret
  - aD
  - = a(dG) (by definition dG = D)
  - = (ad)G (Associativity)
  - = (da)G (our elliptic curve is an abelian group in particular Z/pZ is abelian)
  - =d(aG) (Associativity)
  - = dA (by definition aG = A)
- Properties
  - Only Alice and David can know their shared secret
    - Shared secrete (curve point dA = aD) can be used for symmetric encryption
  - Once the public keys are known Alice & David can independently compute the shared secret
  -

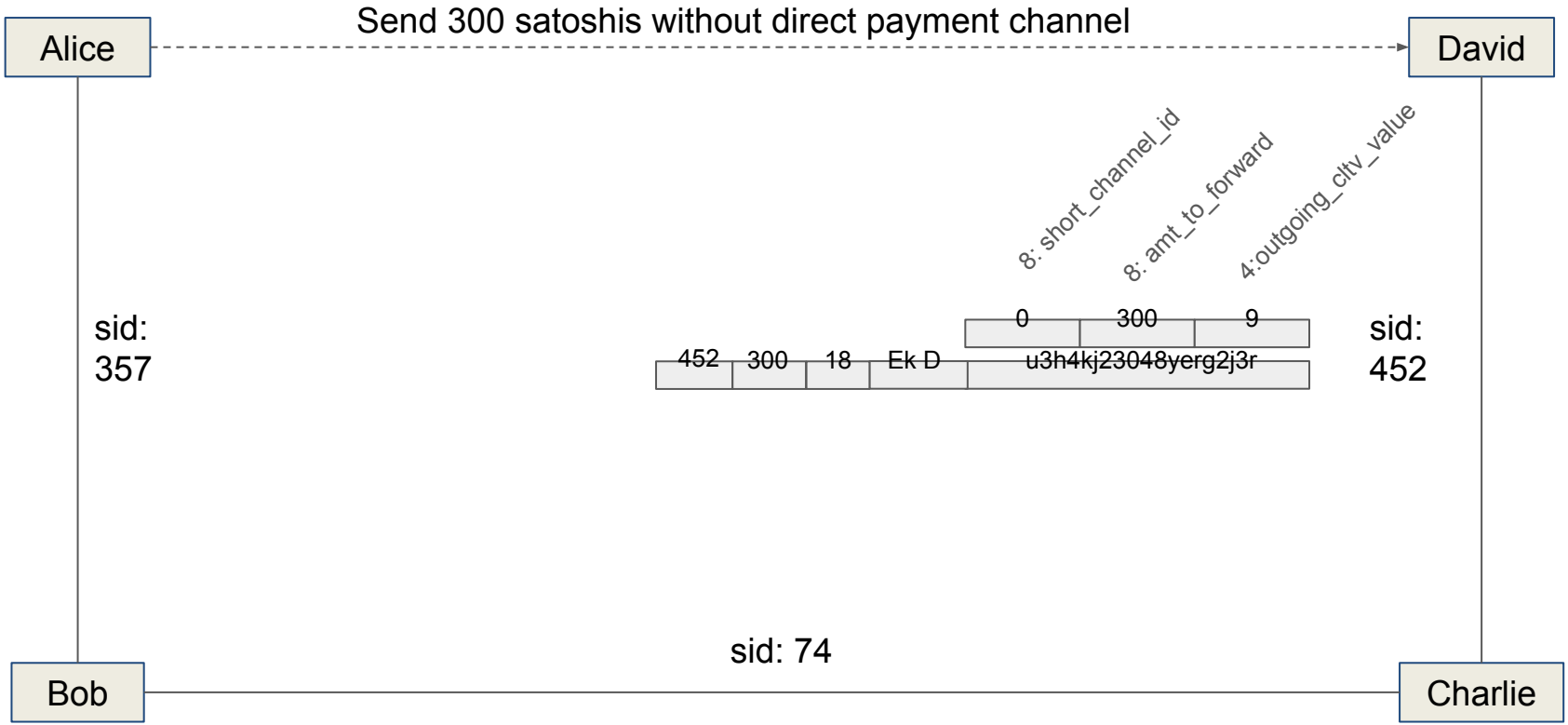
Short Intermezzo for math nerds: Discrete logarithm

- EC is homeomorph to Z/pZ
- ECDSA and DH-Key exchange heavily use the following homeomorphism
- H: Z/pZ ----> EC
  - a |--------> H(a) = aG = A
- Elements in Z/pZ are called private keys
- Elements in EC are called public keys
- By construction of EC (and math theory) we know that H is an isomorphism.
  - This means that H is bijective and an Inverse map H^{-1} does exist
  - H^{-1} is unknown to us unless we compute H for all elements of Z/pZ
  - Explicit construction of H^{-1} for an arbitrary large group computationally infeasible
  - Afaik there is no mathematical theory / reason known that explains why this is not known.
- Knowing H^{-1} would break the security of ECDSA and thus Bitcoin & Lightning
- Giving an analytical, closed formula for H^{-1} is called the discrete logarithm

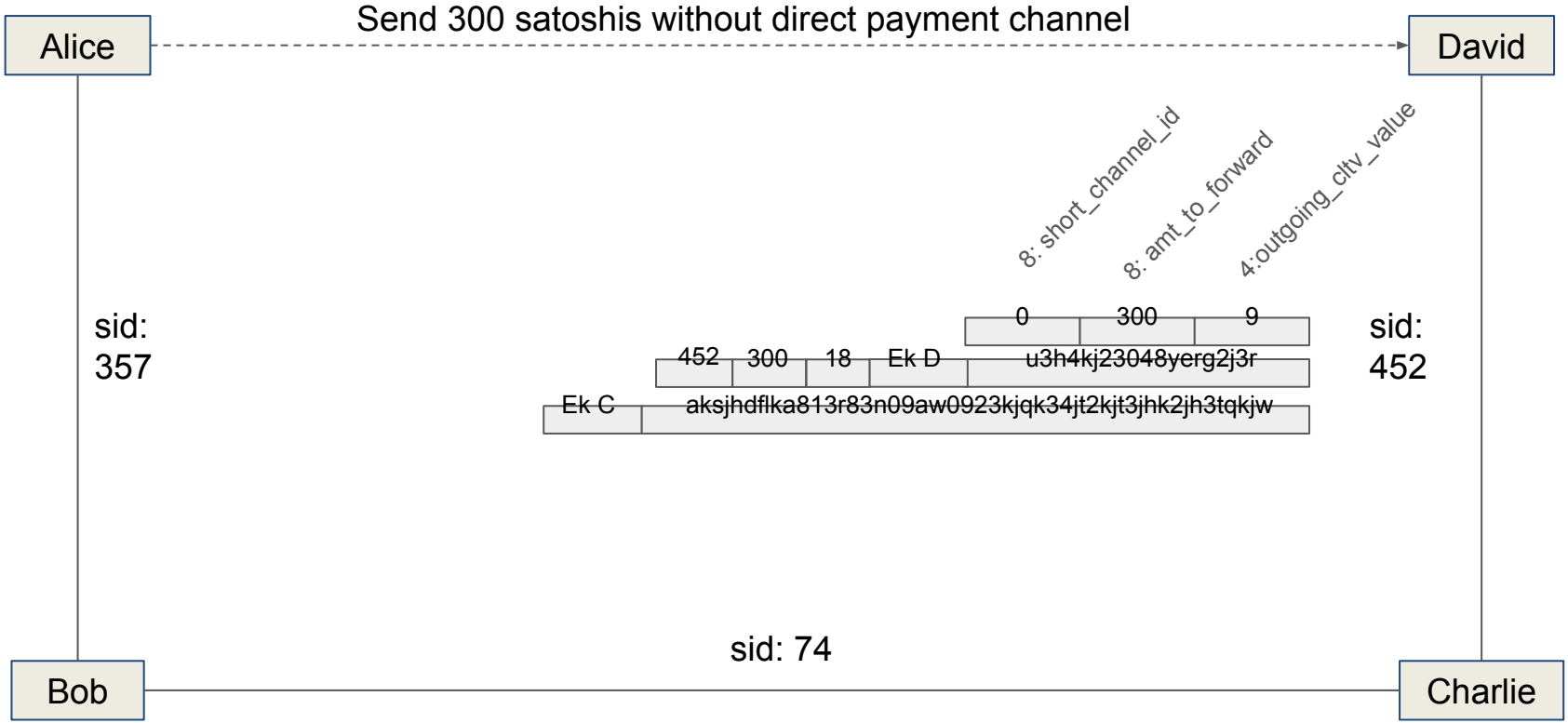
Alice hides her identity with an ephemeral key pair (ek_d, Ek_D) for every hop / participant of the path



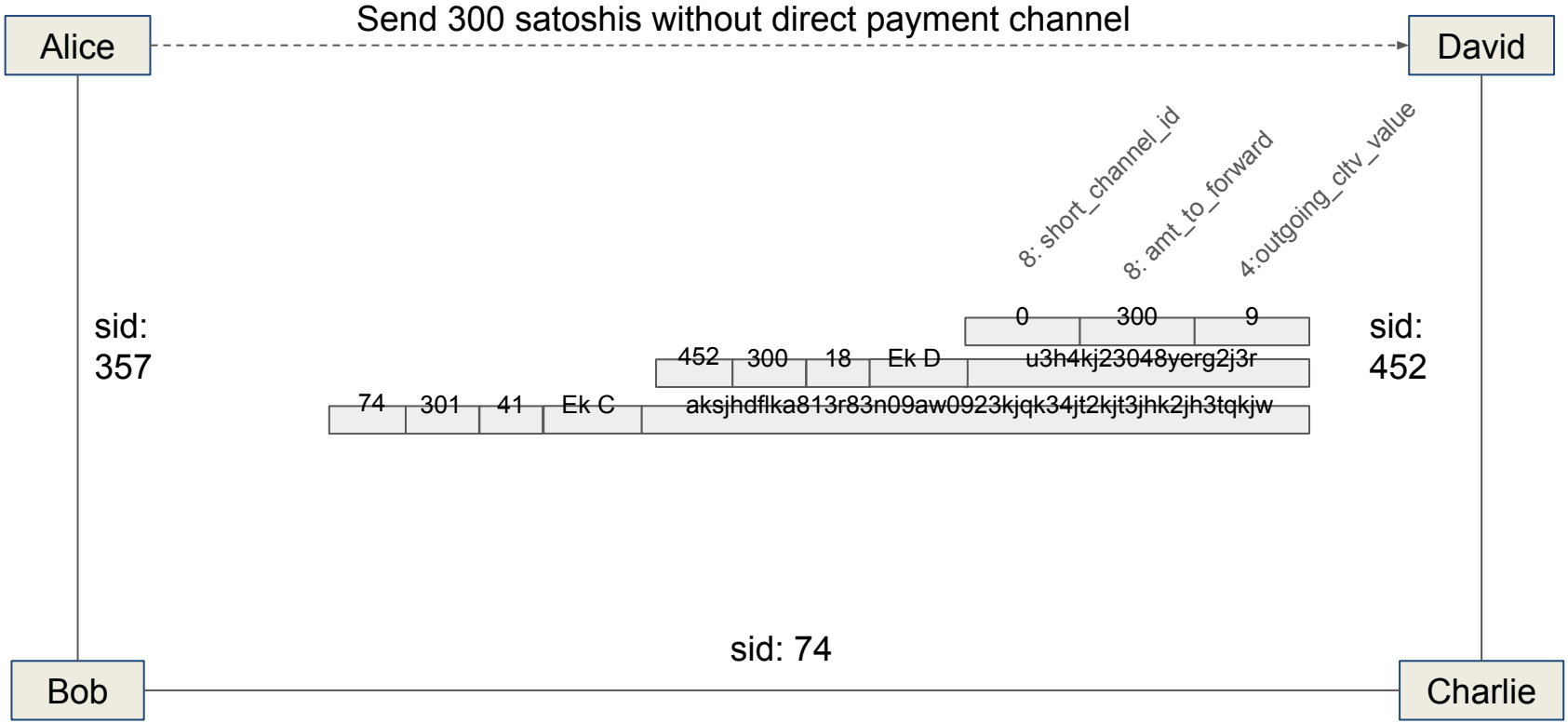
Payload for Charlie (Simplified)



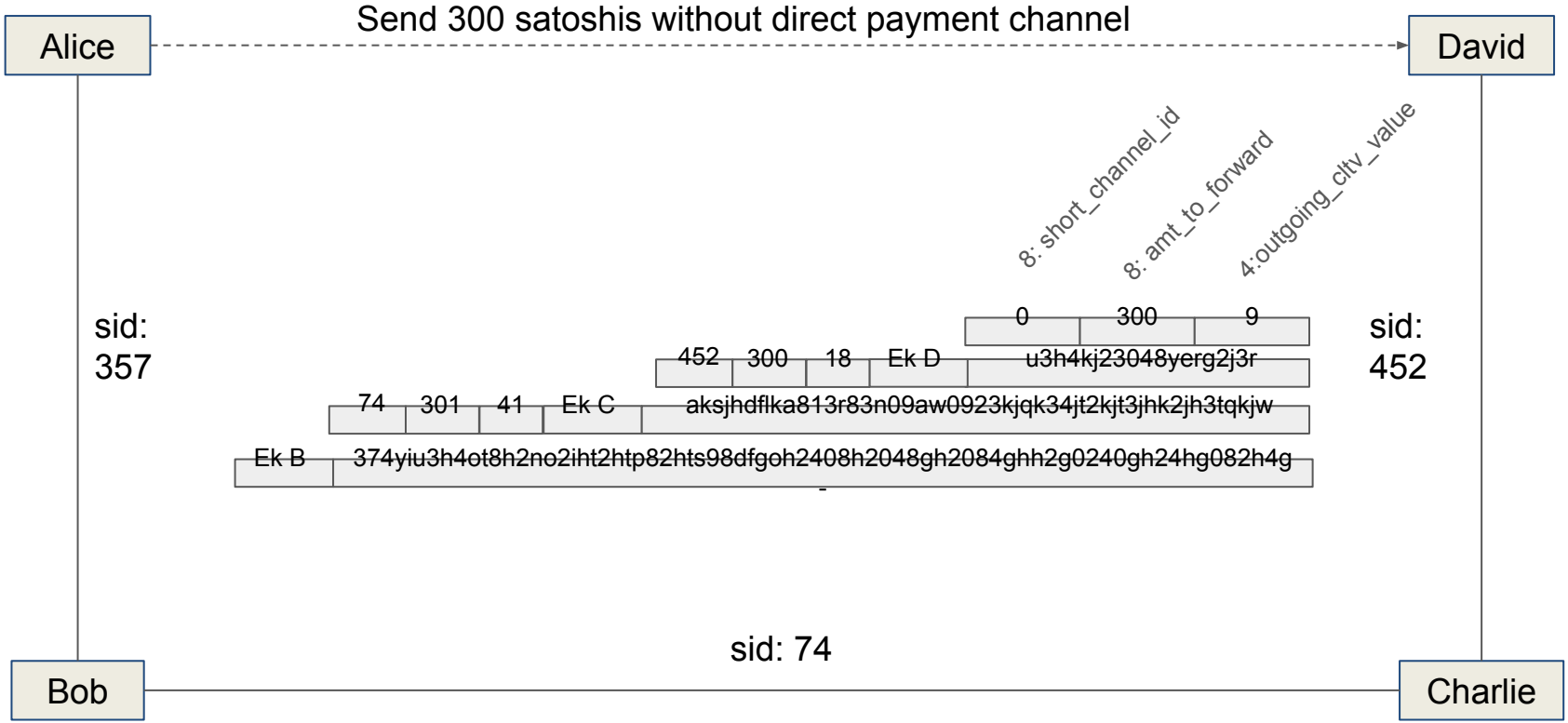
Onion for Charly (without HMAC and filler)



Payload for Bob (simplified)



Onion for Bob without HMAC and filler



Some notes on the presented simplifications

- The onions themselfs are the payload of the update_add_htlc message
  - Described later in the peer protocol
- The message contains the payment hash
- The message offers an htlc with an actual amount
  - Usually nodes offer the amount that they are supposed to forward
- Alice constructs the onion and uses ephemeral keys for every hop
  - Onions are encrypted with a Diffie-Helmann shared secret between
    - Hops ephemeral key (generated by Alice)
    - Hops node_id (static key)
- Onions are always 1366 bytes in length
  - Even if it is the last hop
  - The onions are padded with junk data
    - Padding process left out but described in BOLT 04
  - This prevents a routing node to guess its position in the route by the length of the onion

Normal operation of a channel

(BOLT 02)

3 messages are necessary to operate a channel

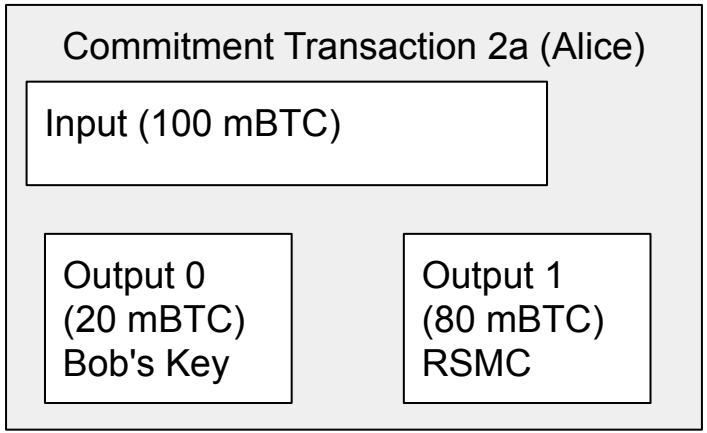
The 5 stages for a htlc to become valid

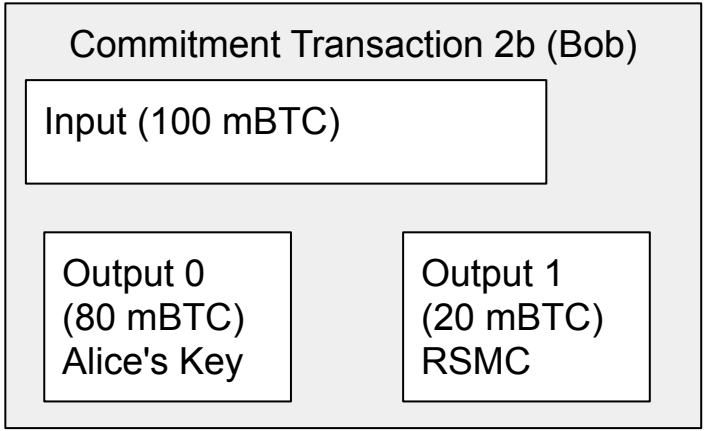
Alice wants to offer a payment to Bob of 15 mBTC

1. Pending on the receiver
2. In the receivers latest commitment tx
3. Receivers old commitment tx is revoked, update is pending at the sender
4. In the senders latest commitment tx
5. Senders old commitment tx is revoked

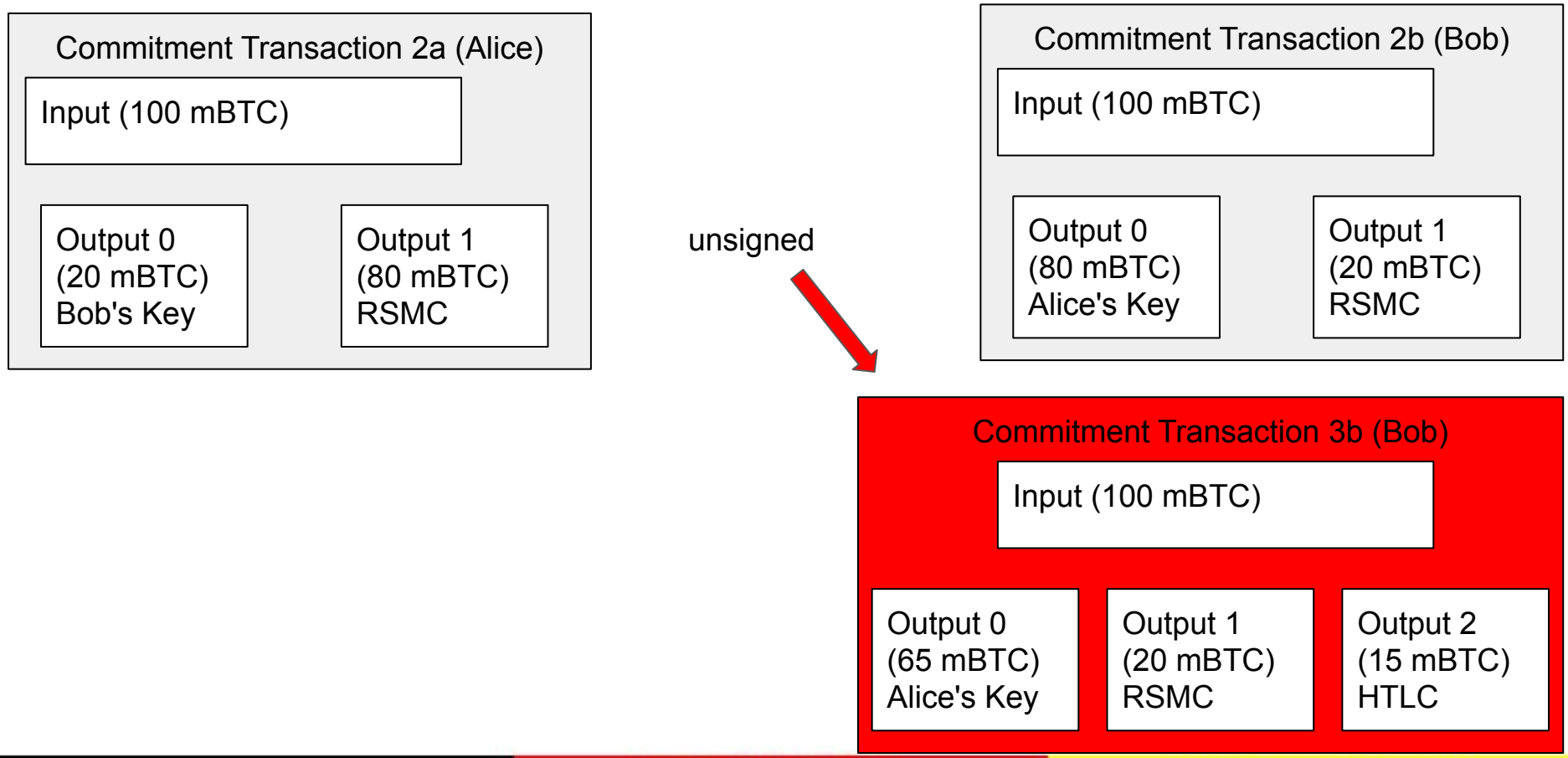
The 5 stages for a htlc to become valid

- Alice wants to offer a payment to Bob of 15 mBTC

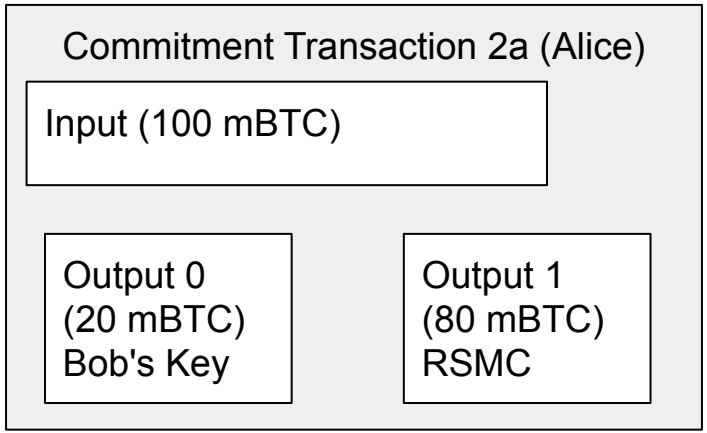


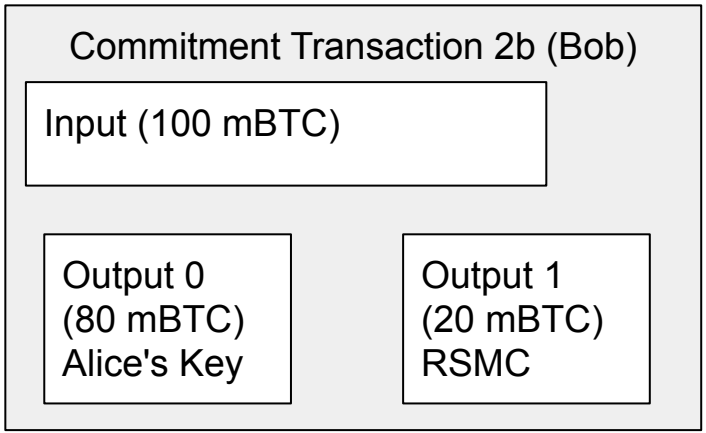


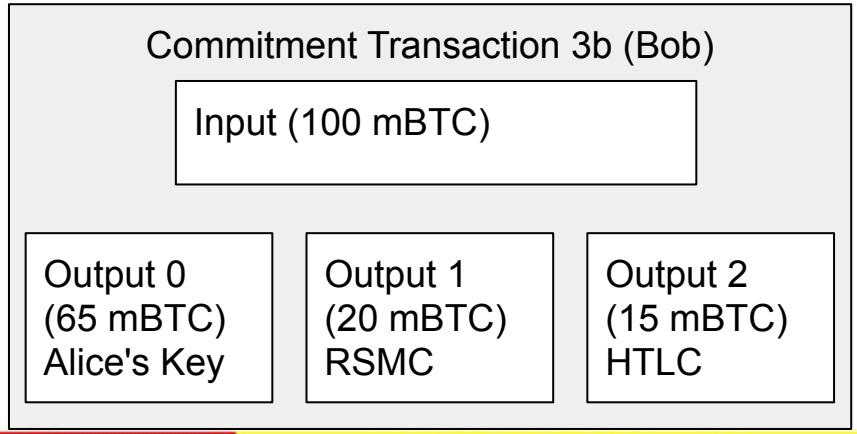
1. Alice sends an update_add_htlc message to Bob



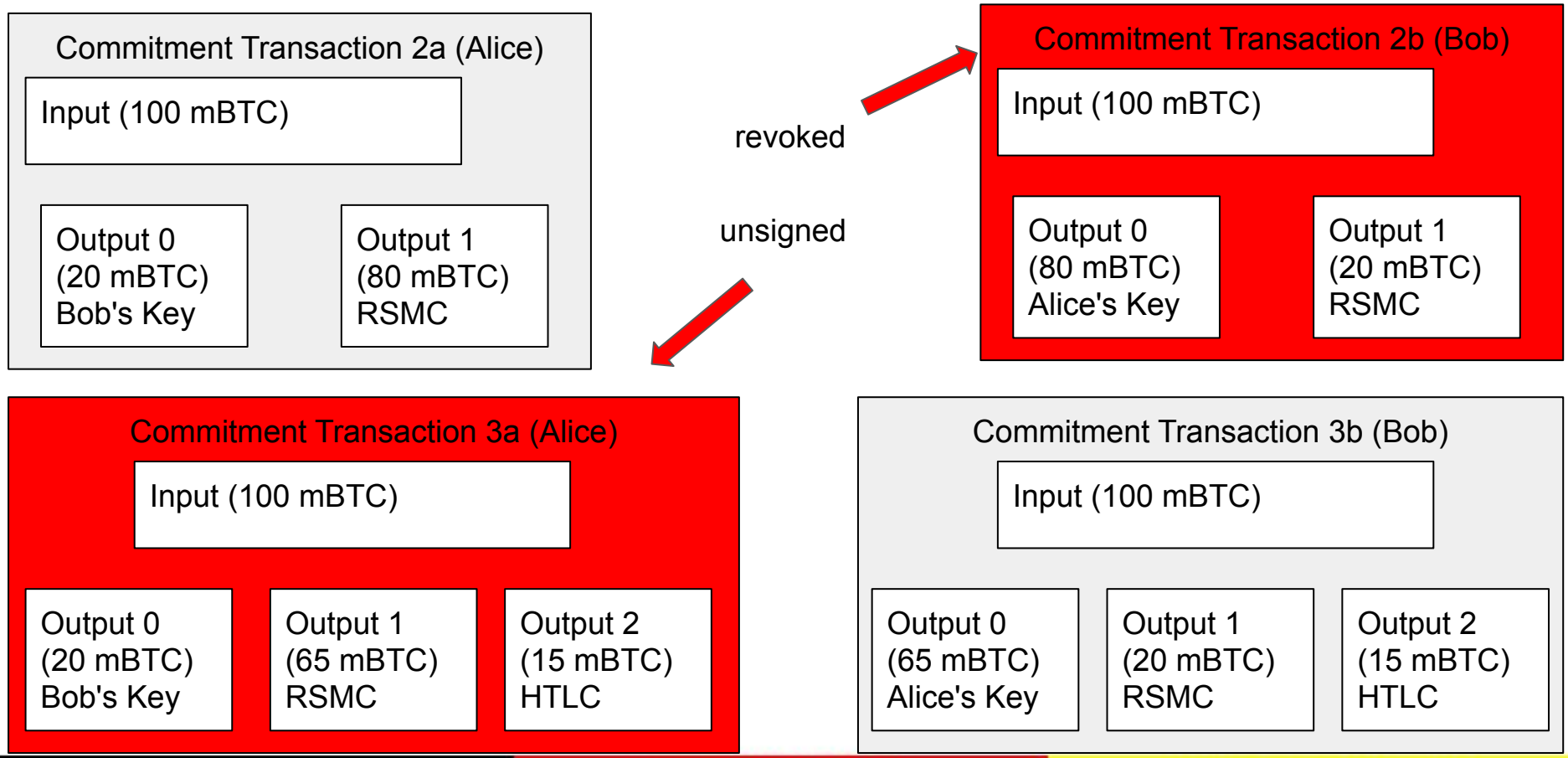
2. Alice sends a commitment_signed message to Bob



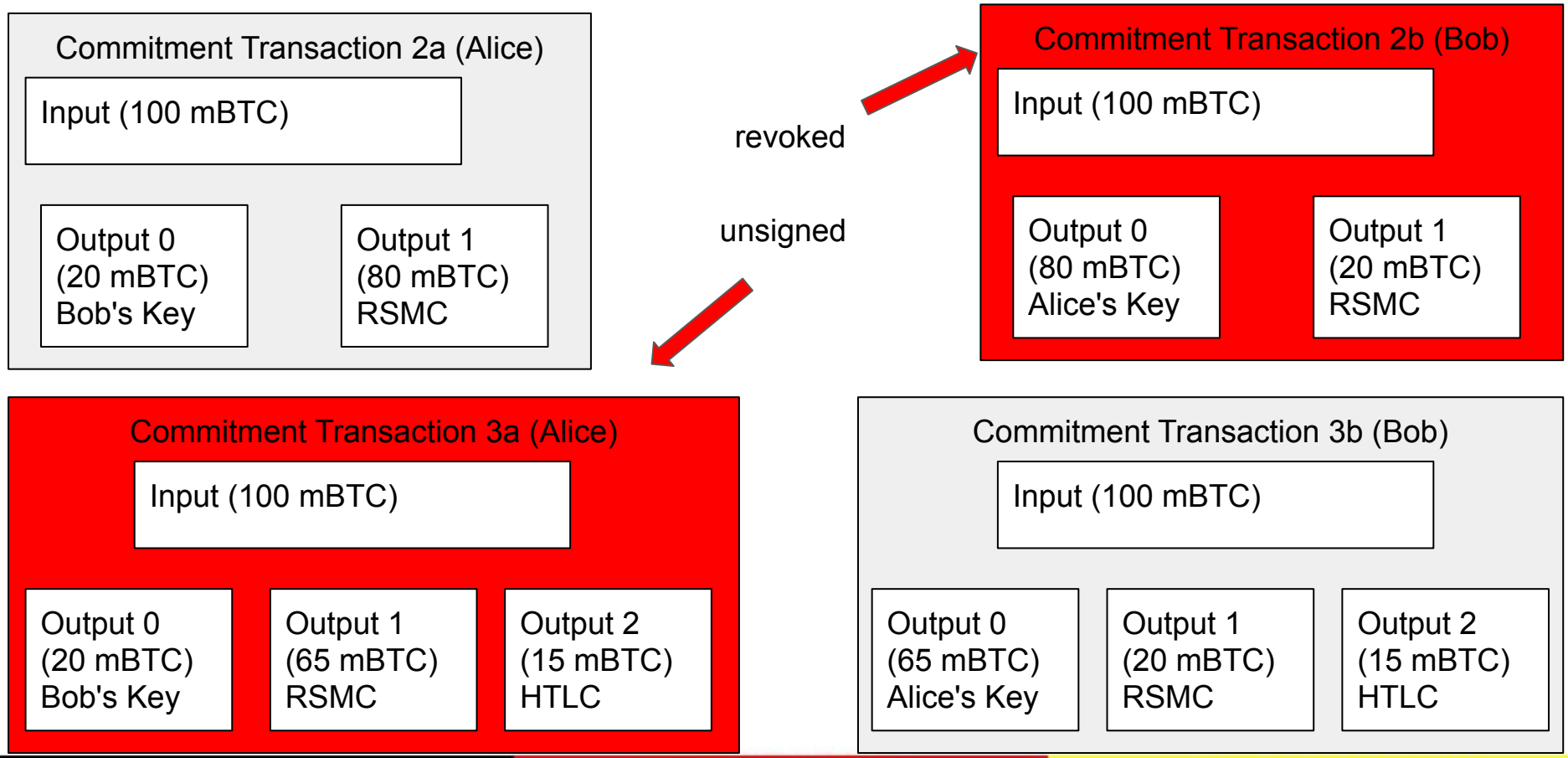




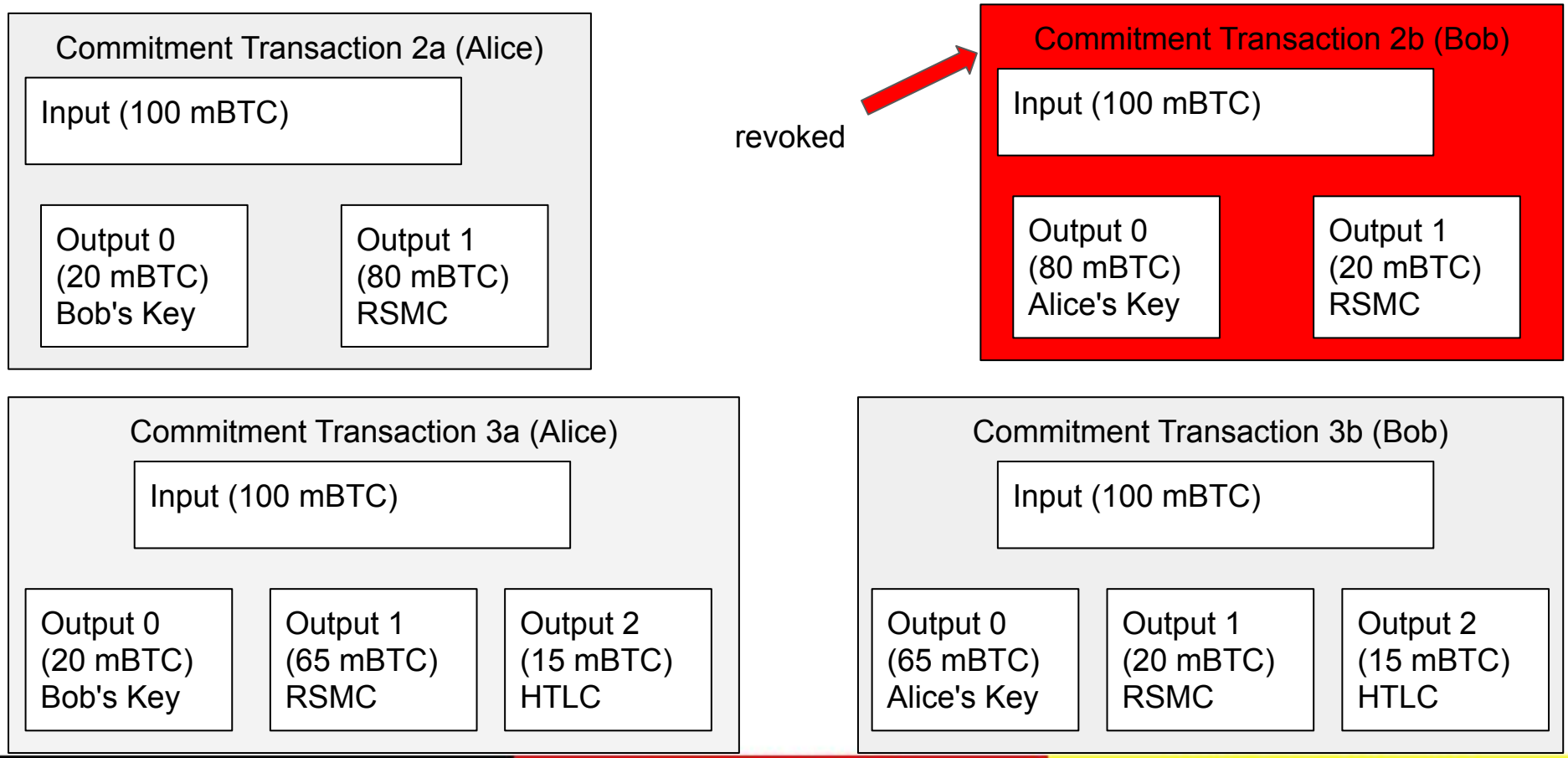
3. Bob sends a revoke_and_ack message to Alice



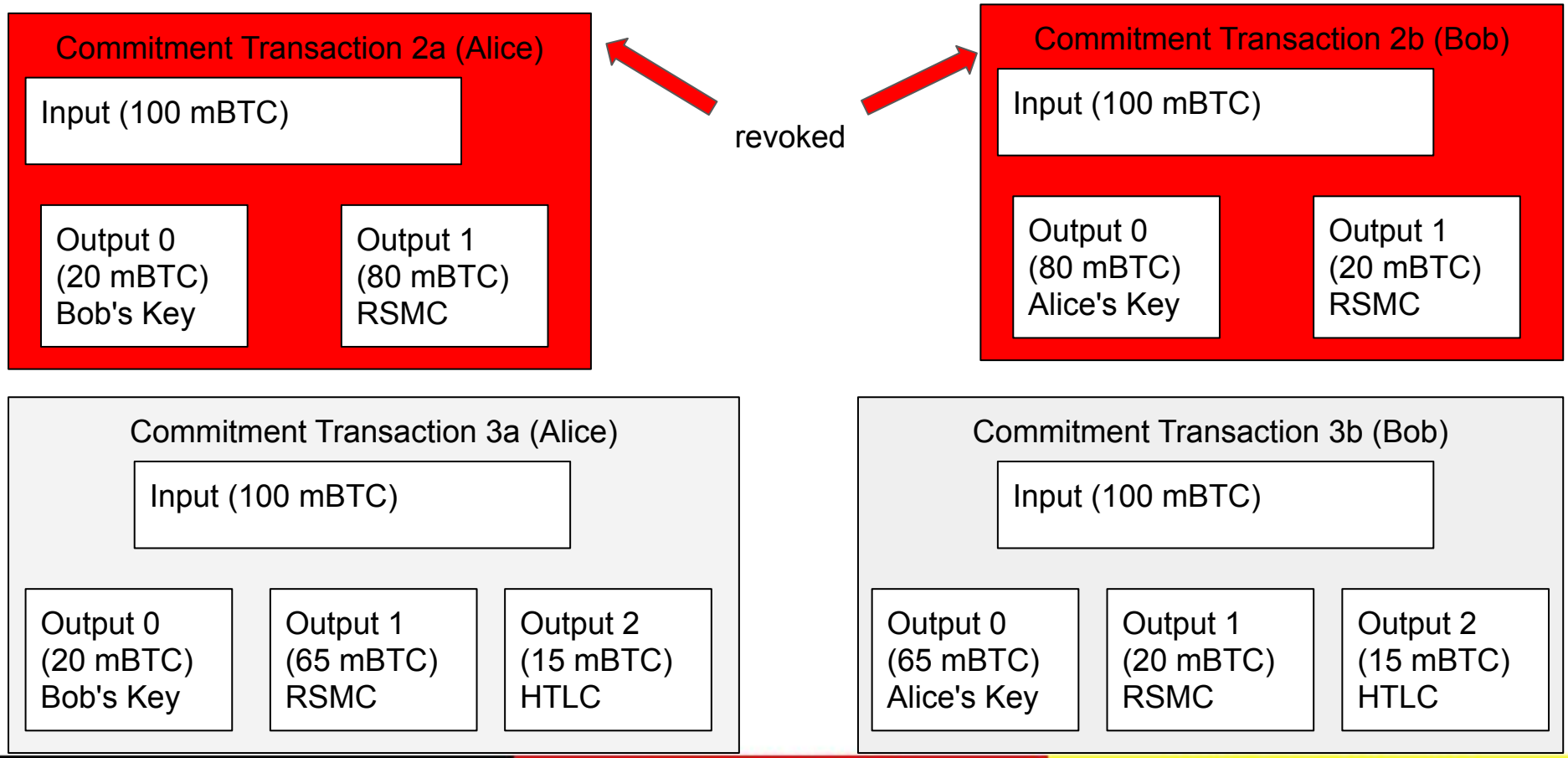
Why doesn't Bob loos funds if Alice publishes ctx2a?



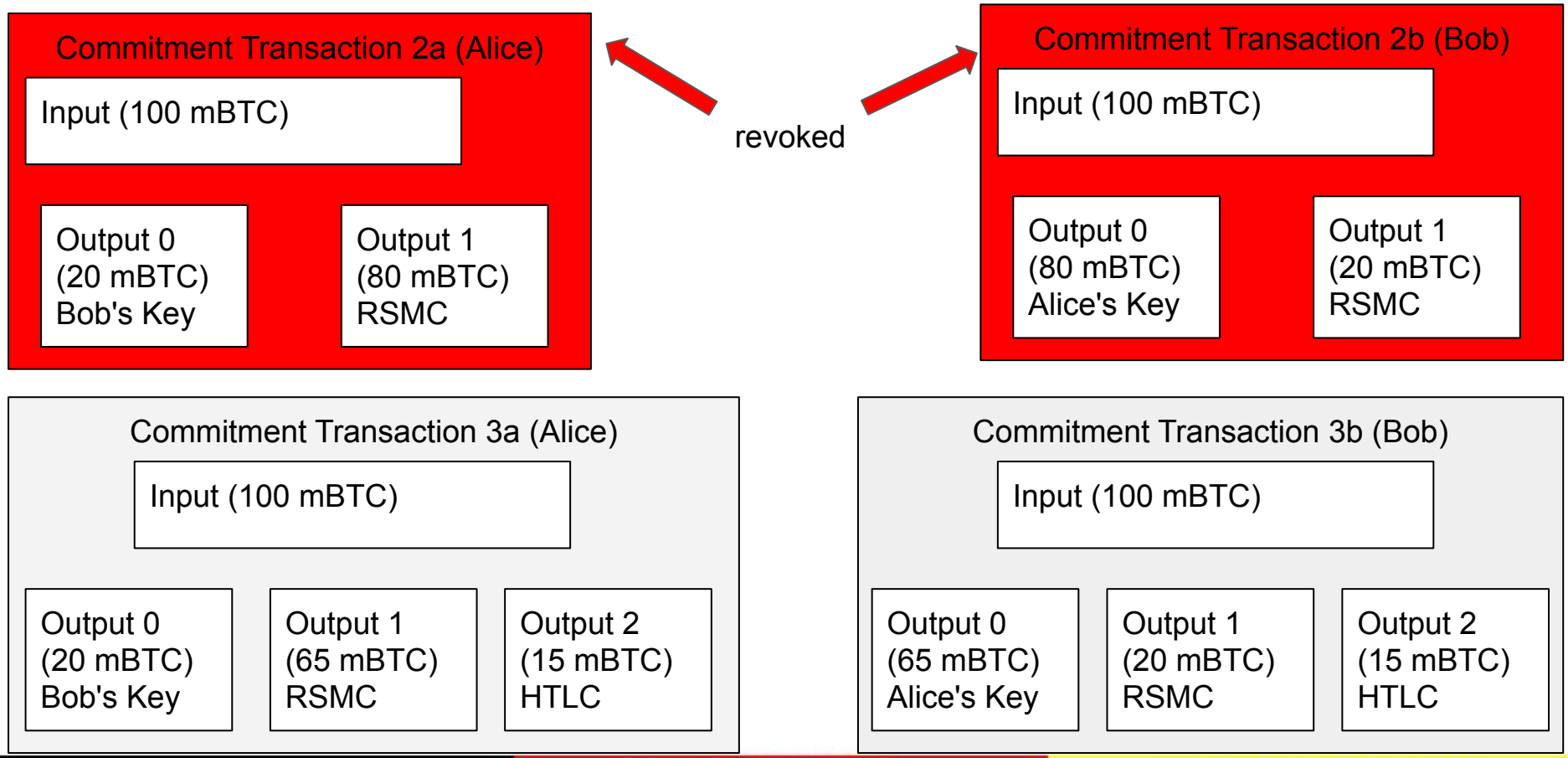
4. Bob sends a commitment_signed message to Alice



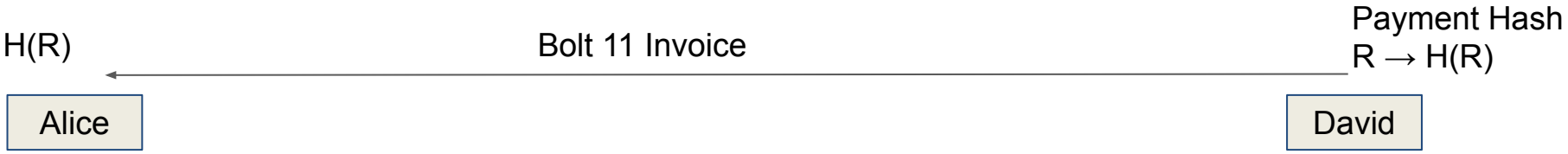
5. Alice sends a revoke_and_ack message to Bob



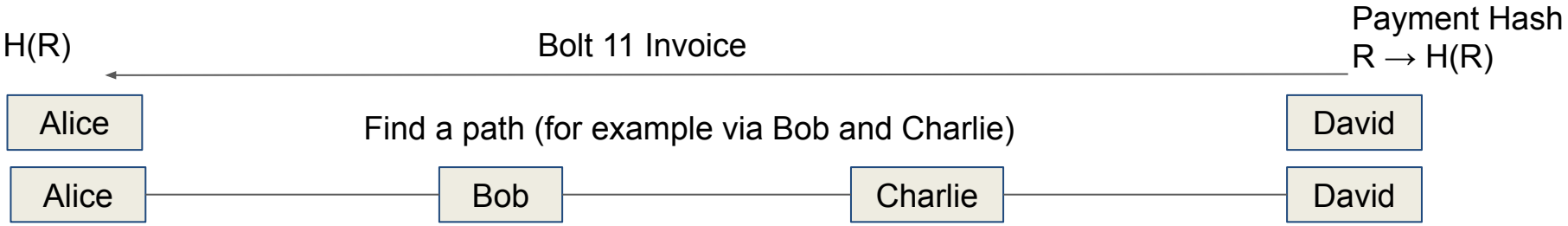
Only now Bob SHOULD forward the htlc!



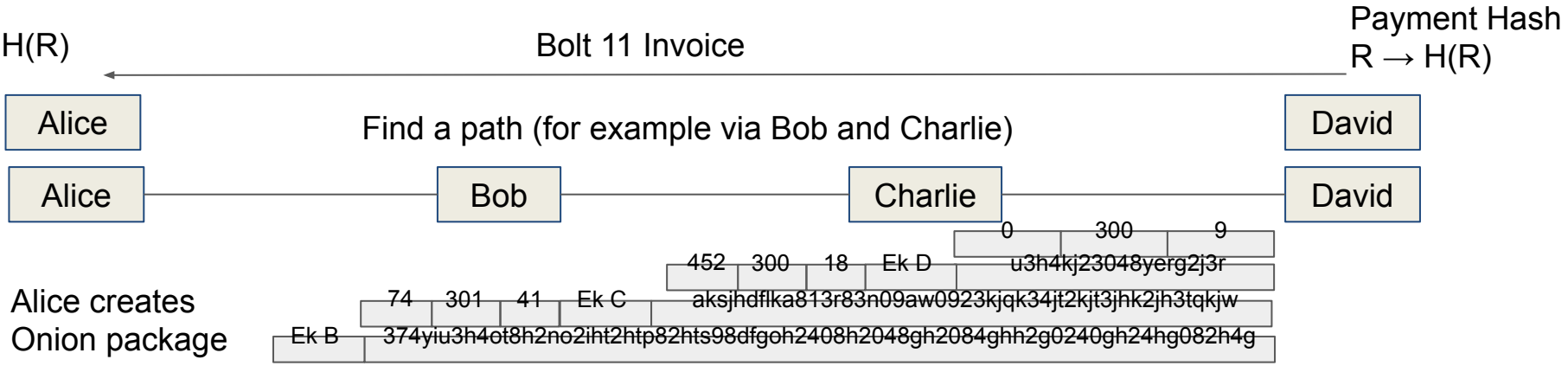
Workflow of a Payment starts with a BOLT 11 invoice



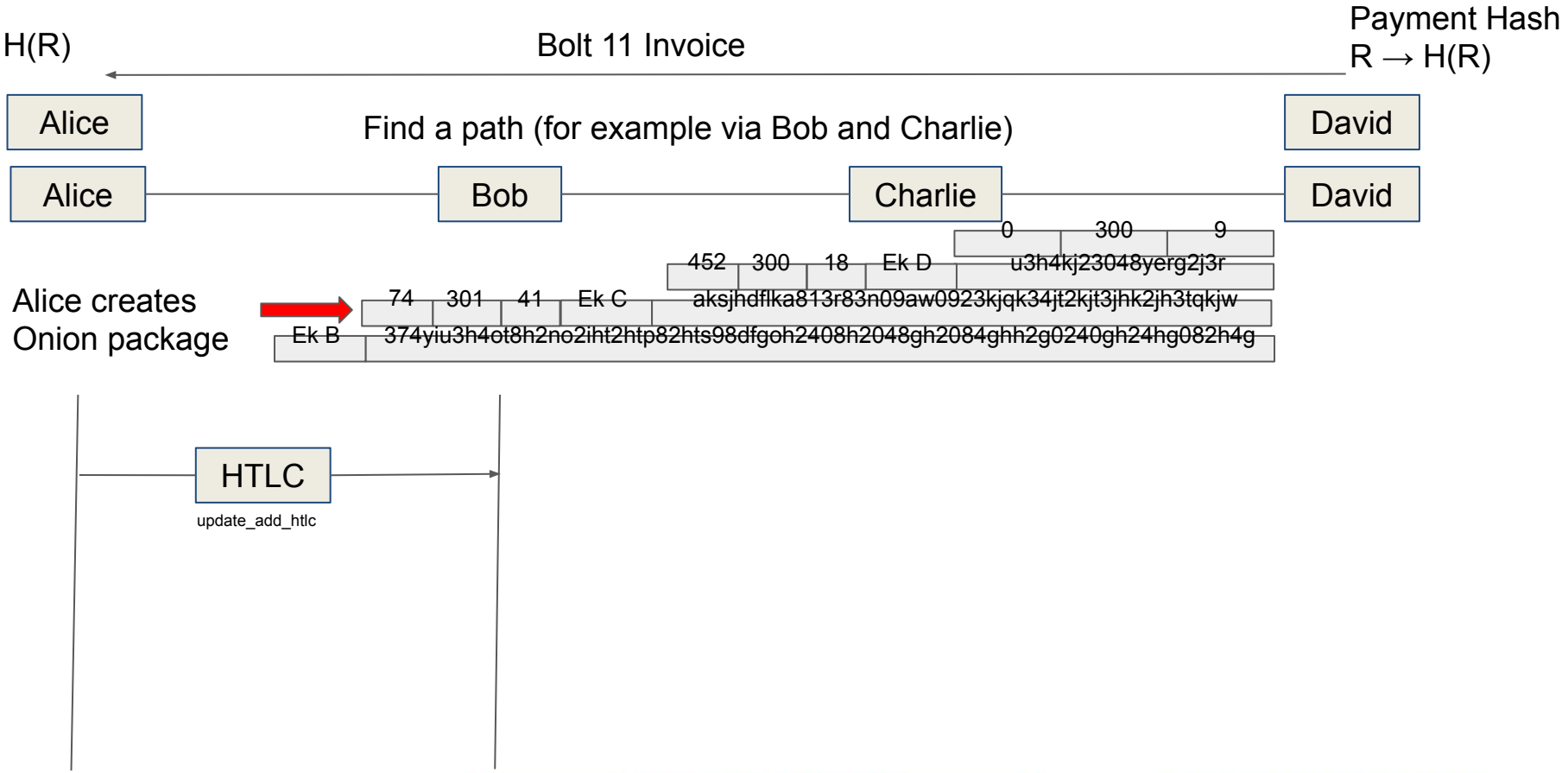
After receiving the invoice Alice selects a path to David



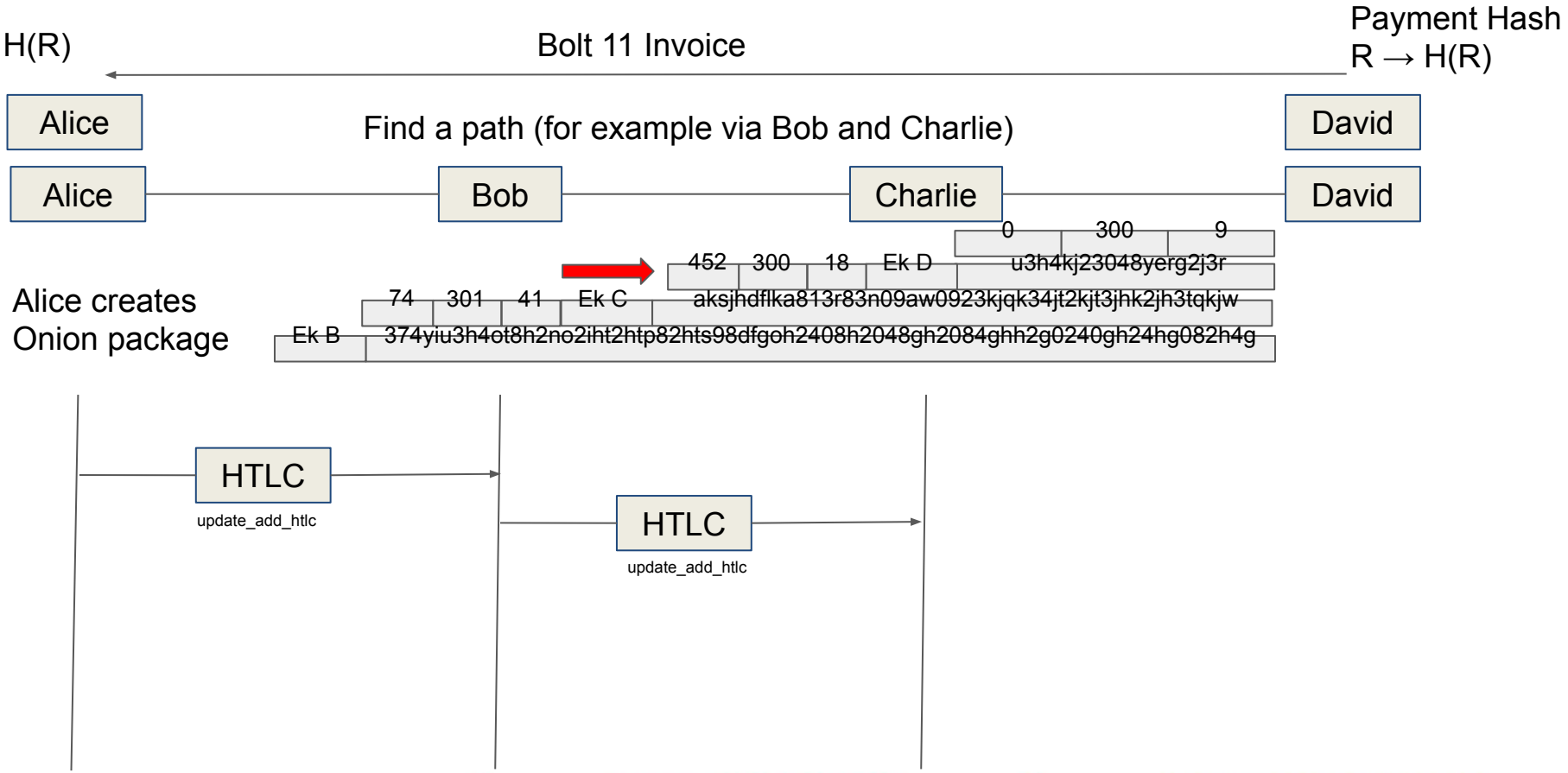
Alice creates the Onion package (starting from David)



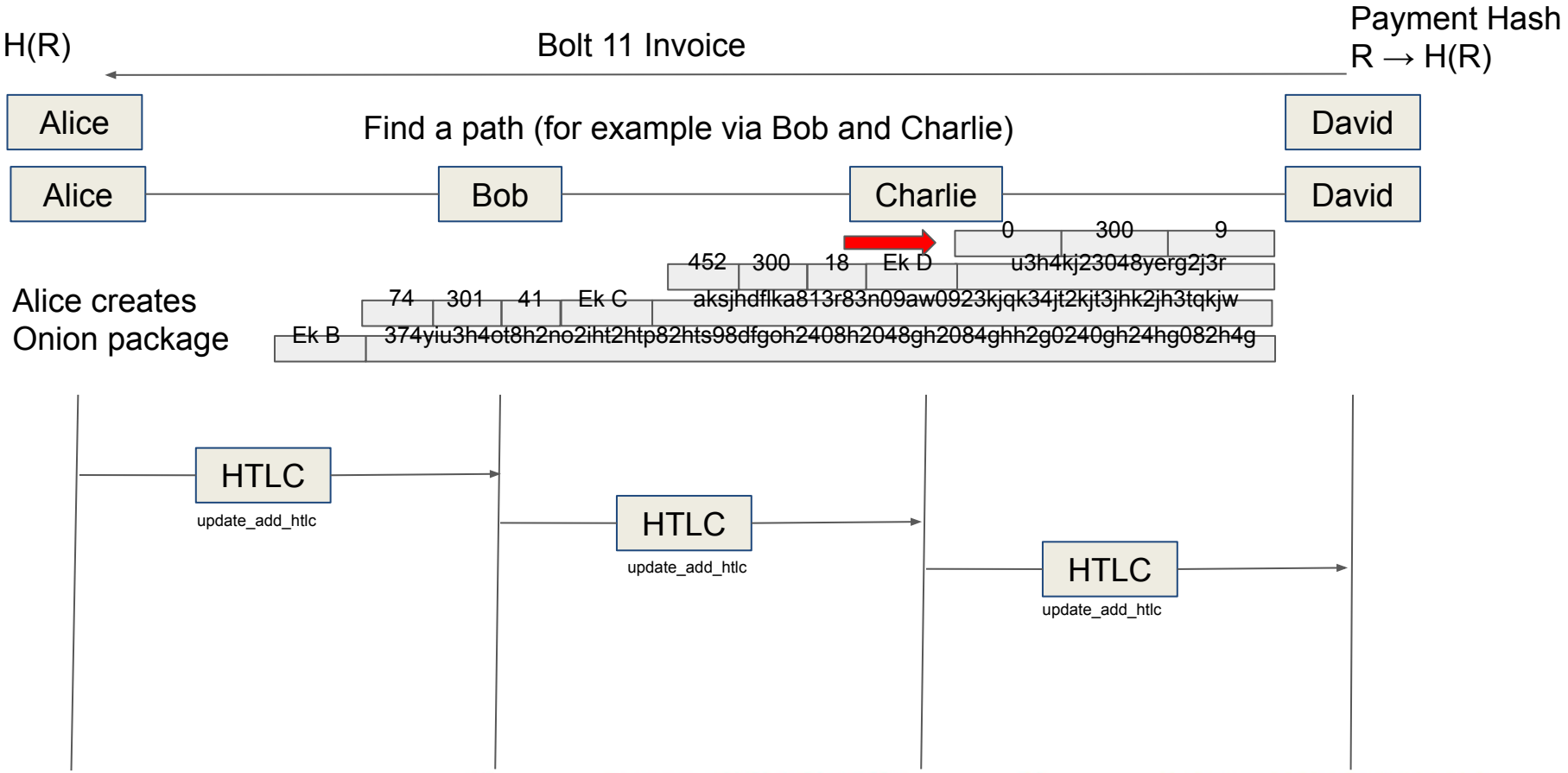
Alice offers the first htlc to Bob



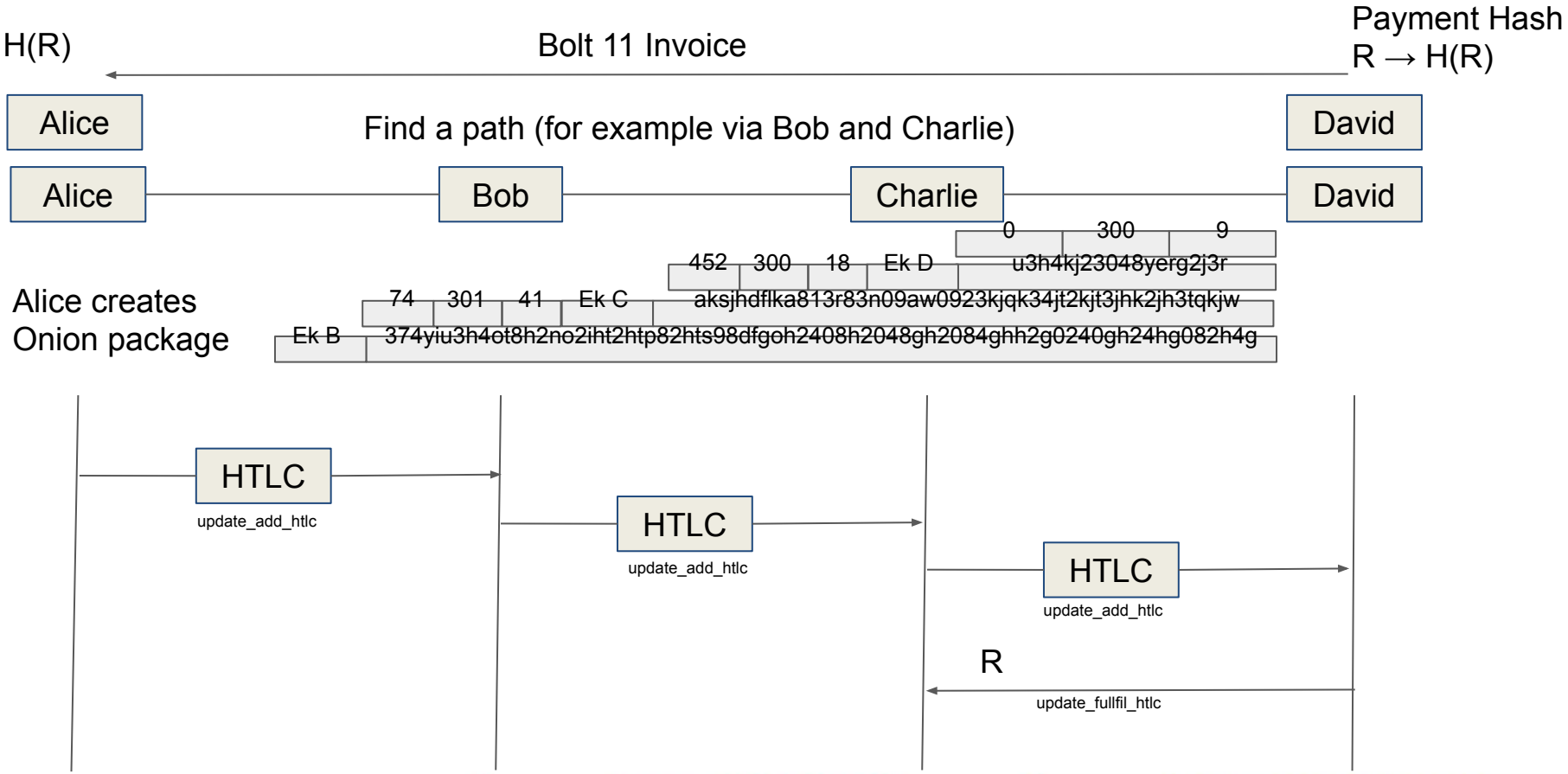
Bob processes the onion and offers an htlc to Charlie



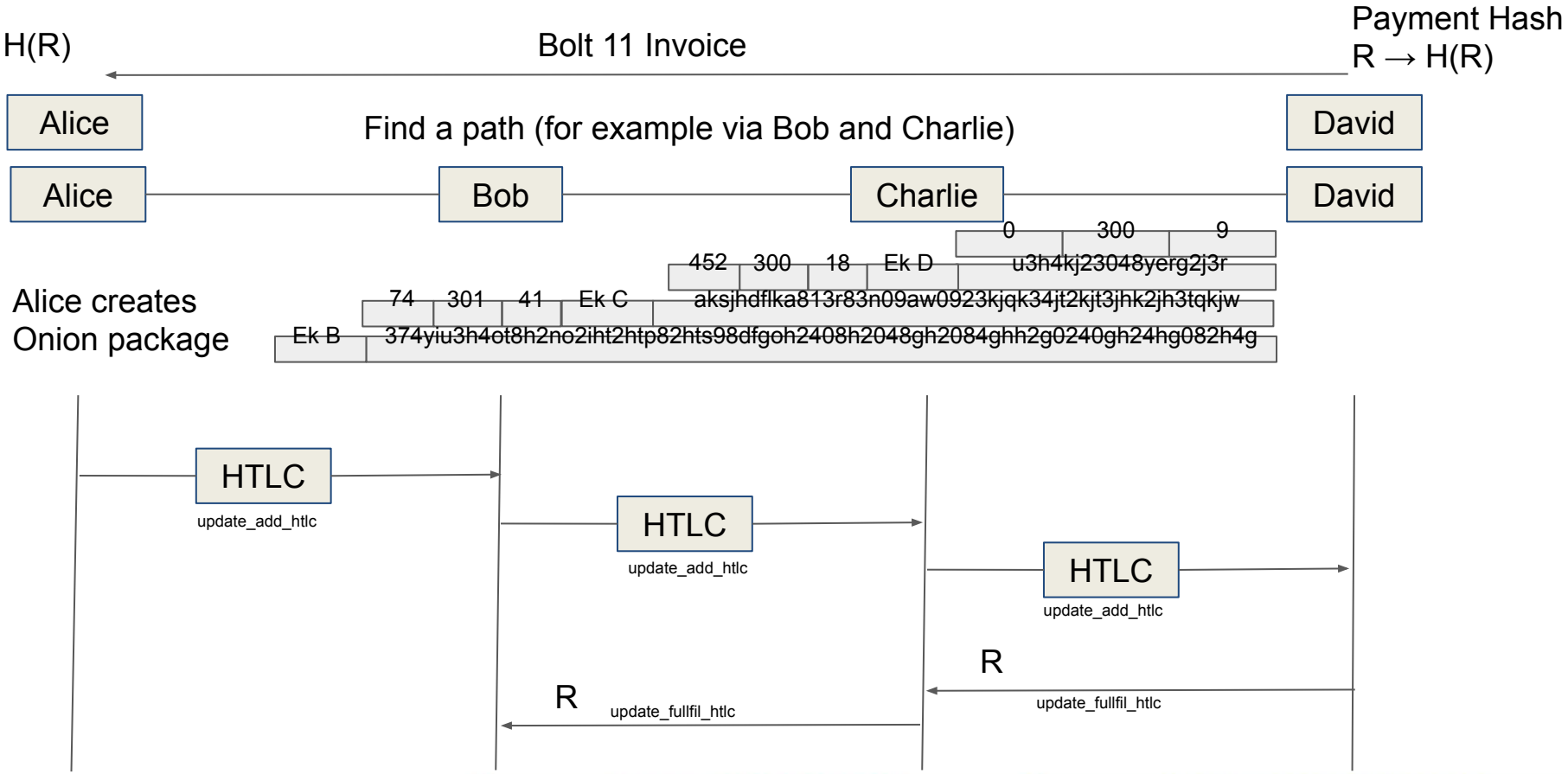
Charlie processes the onion and offers htlc to David



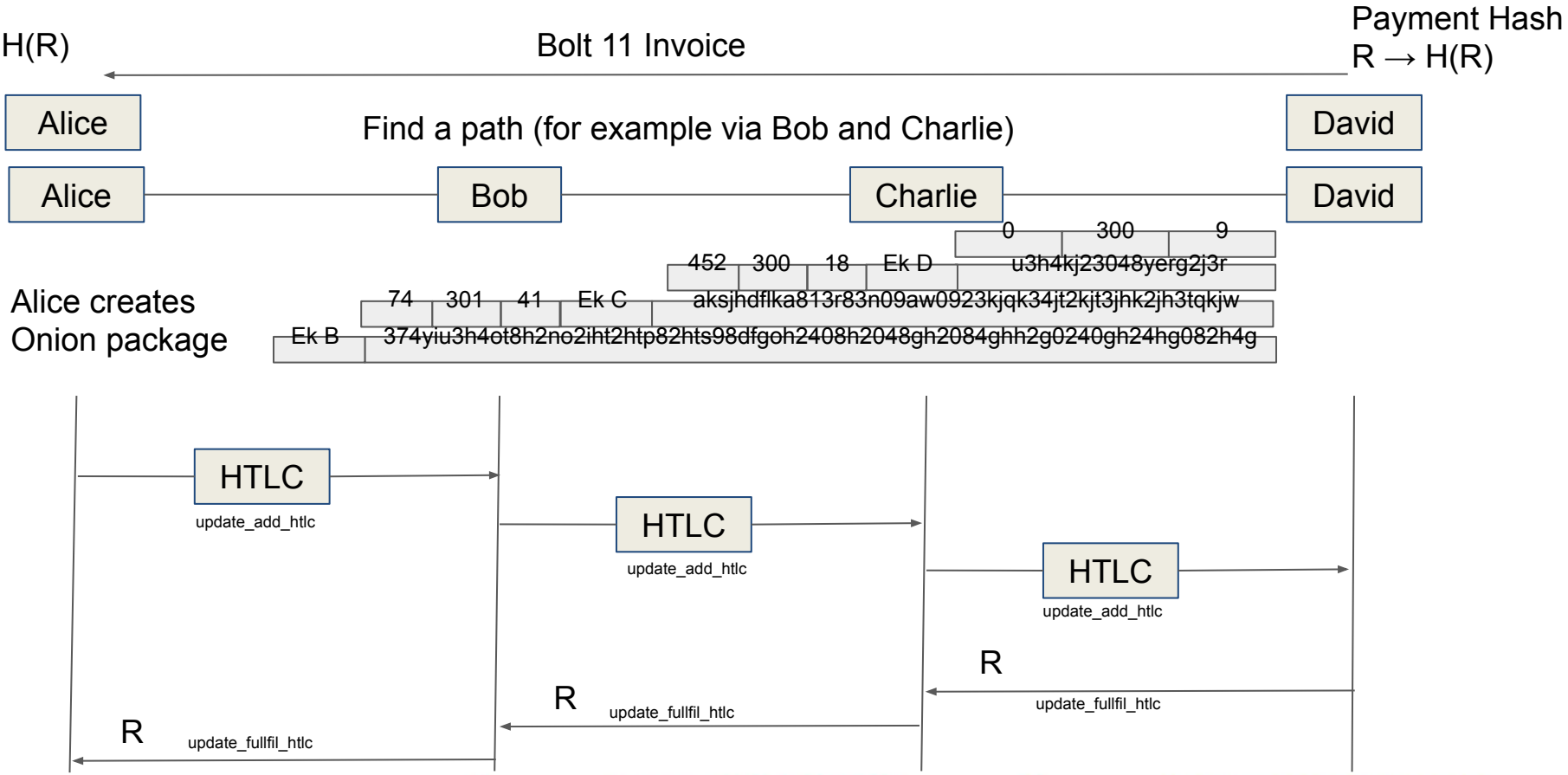
David checks the amount an releases the preimage



Upon receipt of preimage charly claims funds from Bob



Workflow of a Payment on the Lightning Network



References and helpful links

- https://github.com/lightningnetwork/lightning-rfc
- https://cypherpunks.ca/~iang/pubs/Sphinx_Oakland09.pdf (SPHINX Mix Format Paper)
- https://www.youtube.com/watch?v=34TKXELJa2c (SPHINX Mix Format presented)
- https://www.youtube.com/user/RenePickhardt
- https://bitcoin.stackexchange.com/questions/tagged/lightning-network
- https://lightning.network/lightning-network-paper.pdf
- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Discrete_logarithm
- https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
- https://en.wikipedia.org/wiki/Elliptic-curve_Diffie%E2%80%93Hellman
-

Copyright notice

- This slide deck is openly licensed with a creative commons license CC-BY-SA-4.0.
  - The full license text can be found at: https://creativecommons.org/licenses/by-sa/4.0/legalcode
- You are
  - free to
    - Share
    - Remixe
  - As long as you
    - Link to the original work
    - State my Name and Website
    - Mark changes in your derivative work
    - Use the same license for your derivative work
- Screenshots in this slide deck are taken by me but the design of the websites might be protected by copyright
- This slide deck uses parts of the lightning-rfc which is licensed as CC-BY (the lightning developers)
  - The full license text can be found at: https://creativecommons.org/licenses/by/4.0/legalcode
- The graphics from the backup slides are taken from https://en.bitcoin.it/wiki/Transaction and are Public Domain

Thanks to Marietheres Viehler (aka journalspiration) for the design of the title slide.

About this slide deck

The purpose is to help spreading education about the Lightning Network Protocol so that the technology will be adopted more quickly by more people. This shall be my contribution to the Bitcoin / Lightning Network Community.

This slide deck was presented during Chaincodelabs Lightning Residency program in June 2019. It is part of https://commons.wikimedia.org/wiki/File:Introduction_to_the_Lightning_Network_Protocol_and_the_Basics_of_Lightning_Technology_(BOLT_aka_Lightning-rfc).pdf. To the best of my knowledge the original file is the most comprehensive work making an introduction to the BOLT standard.

The slides are part of my effort to create a book about the lightning network. You can follow that effort at: https://github.com/renepickhardt/The-Lightning-Network-Book or you can support the effort at my fundrasing pages at: https://tallyco.in/s/lnbook or at: https://www.patreon.com/renepickhardt or at 1GZx8tWgDd21Rd8b1QdMrzdZGHgyfVkzaD part of this effort also consists of creating video tutorials and teaching materials on my Youtube Channel over at: https://www.youtube.com/user/RenePickhardt

This work was funded (sorted by amount of contribution from top to bottom) by: Me personally, fulmo.org, everyone who contributed to the above mentioned fundraiser and George Danzer.

Thank you to the lightning developers and people in various telegram groups for helpful discussions