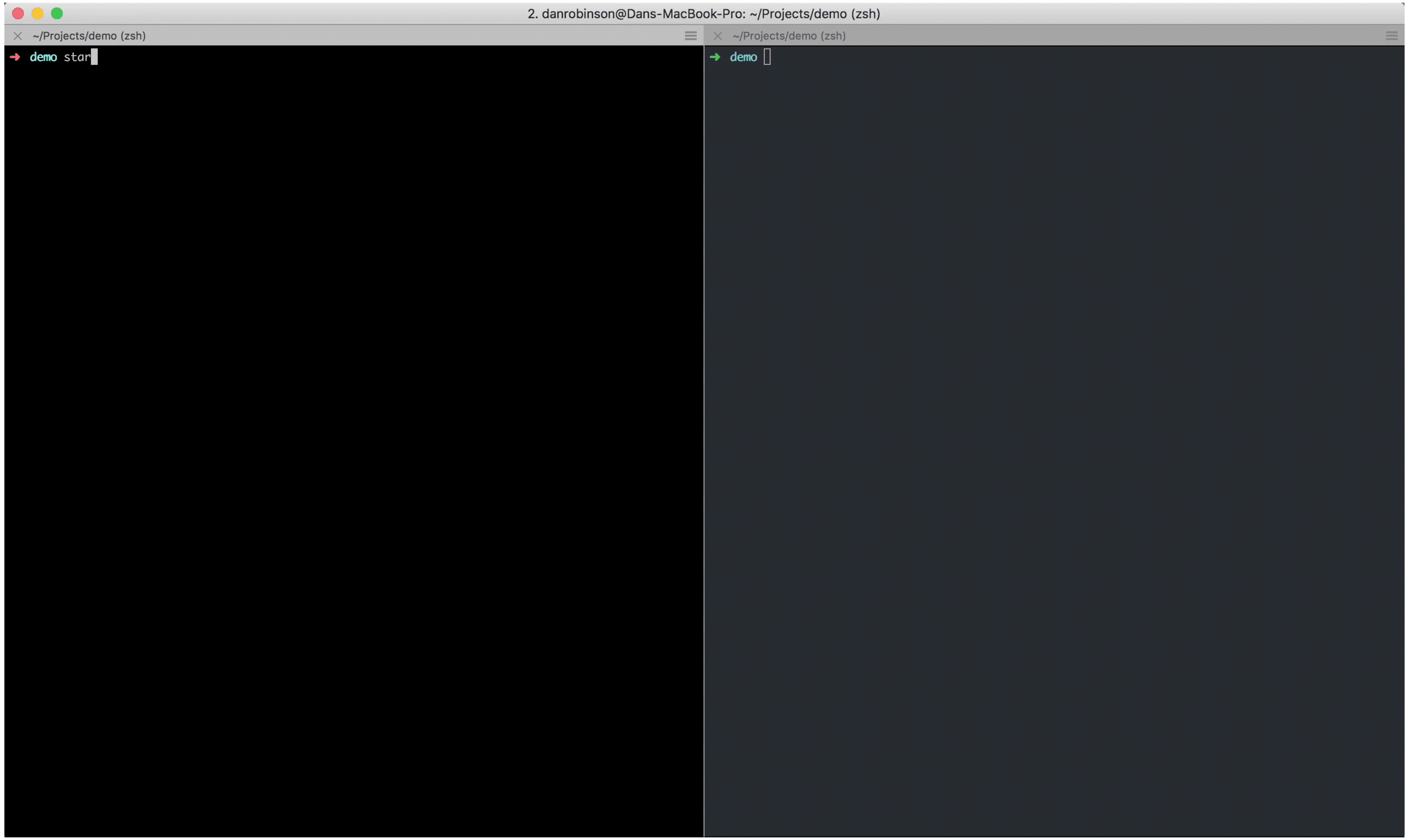
Bitcoin is Smart

Akash Khosla

Engineering @ Anchorage

Stellar + Chain = Inter/stellar



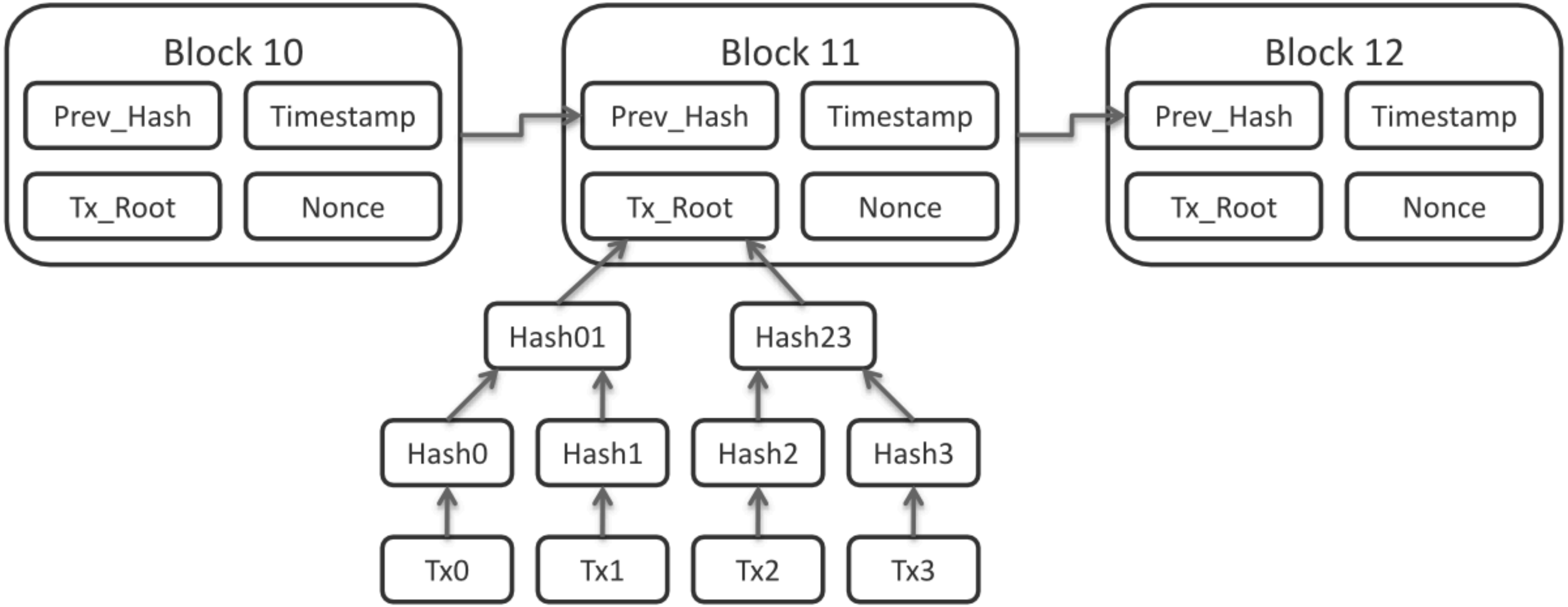
```
➜ demo star
```

```
➜ demo
```

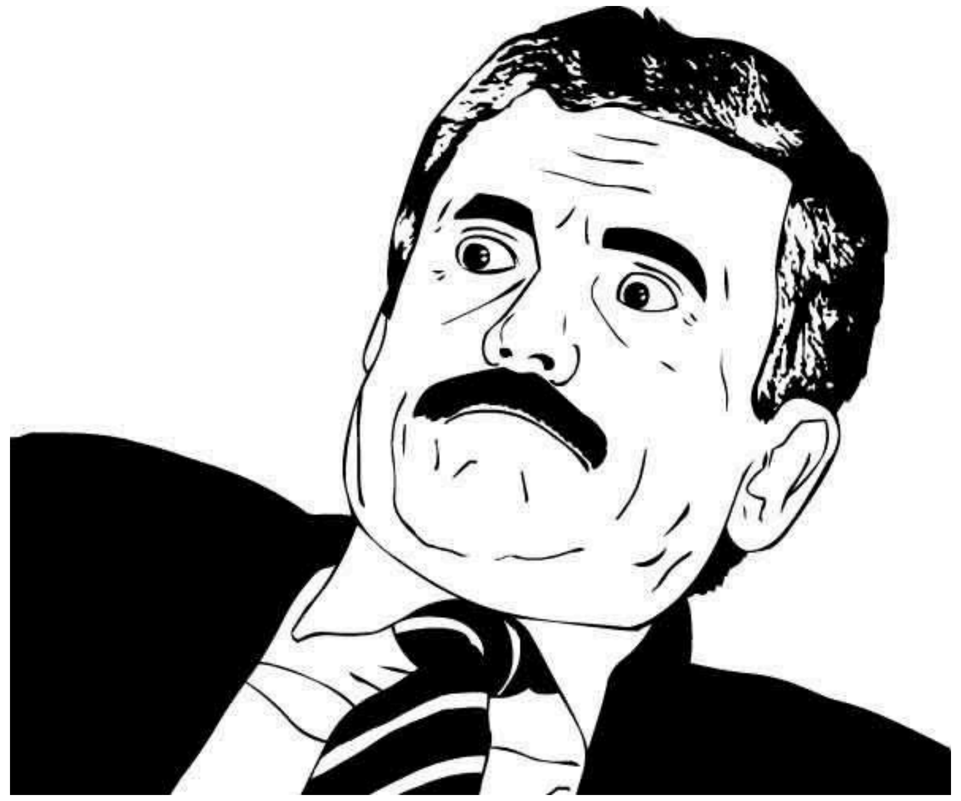
The amount of theory out there is infinite, the number of practical things you can do is finite.

Ownership

- *CB: Confused Bitcoiner, A: Andreas Antonopoulos*
- CB: How do I own coins?
- A: It’s under your address!
- CB: But isn’t that an account system?
- A: The ledger says you have a specific set of UTXOs.
- CB: Well where does it say that?



A: A UTXO is basically an unopened lockbox that your keys can access.
CB: Where does that lockbox sit?



```
{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a8
63ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa3
36a8d752adf",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

Transactions

- Generally, when we think about transactions, we think of updating account balances
- In Bitcoin: no accounts, no balances, no coins, no senders, no recipients, no addresses involved
- All of this = higher level abstraction
- **Inputs** and **Outputs**

```
{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a8
63ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa3
36a8d752adf",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

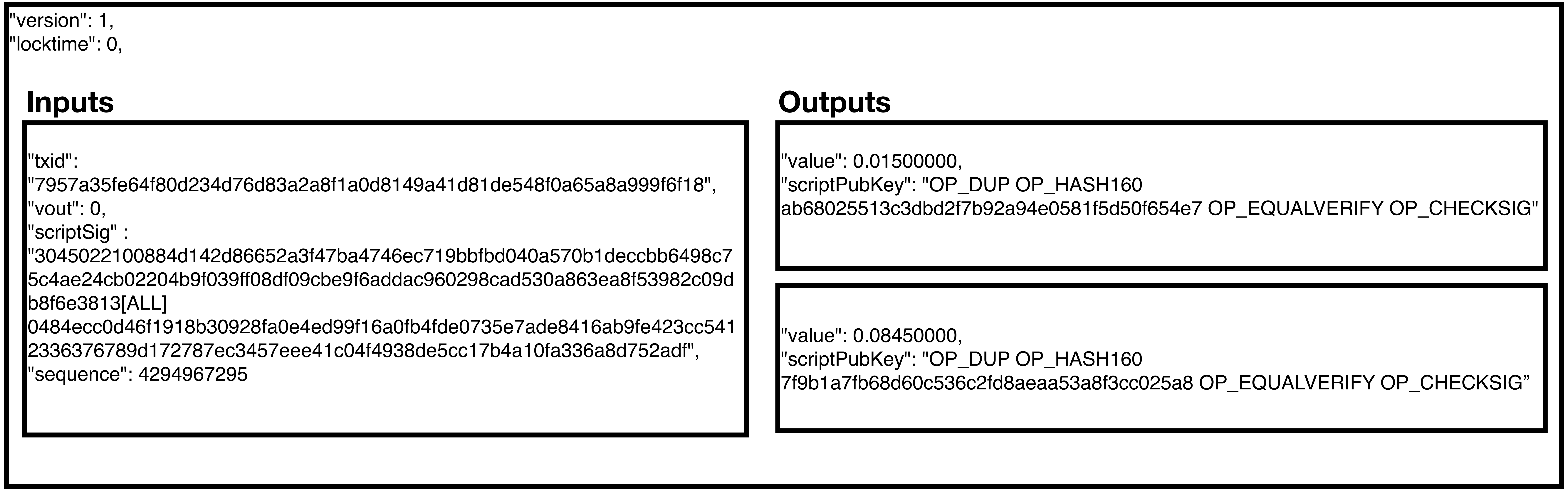
Transactions

```
"version": 1,
"locktime": 0,
```

```
"vin": [
   {
     "txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
     "vout": 0,
     "scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c7
5c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09d
b8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc541
2336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
     "sequence": 4294967295
   }
 ],
```

```
"vout": [
   {
     "value": 0.01500000,
     "scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
   },
   {
     "value": 0.08450000,
     "scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
   }
 ]
```

Transactions



```
"version": 1,
"locktime": 0,
```

Inputs

```
"txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c7
5c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09d
b8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc541
2336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295
```

Outputs

```
"value": 0.01500000,
"scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
```

```
"value": 0.08450000,
"scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG”
```

Outputs

- Indivisible chunks of Bitcoin
- Are spendable (they become inputs to a transaction when spending)
- Spendable outputs known as UTXOs (Unspent Transaction Outputs)
- Every transaction represents a change in the UTXO set
- When you want to spend, use an **output** by referencing it in an **input, via a tx id and index** (called vout)

Outputs

```
"value": 0.01500000,
"scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY
OP_CHECKSIG"
```

```
"value": 0.08450000,
"scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY
OP_CHECKSIG"
```

Transactions

```
"version": 1,
"locktime": 0,
```

Inputs

```
"txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c7
5c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09d
b8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc541
2336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295
```

Outputs

```
"value": 0.01500000,
"scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
```

```
"value": 0.08450000,
"scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG”
```

Inputs

```
"txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f03
9ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787e
c3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295
```

Inputs

```
"txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f03
9ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787e
c3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295
```

Looking at the outputs for 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

```
"vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8
OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
```

Transactions

```
"version": 1,
"locktime": 0,
```

Inputs

```
"txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c7
5c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09d
b8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc541
2336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295
```

Outputs

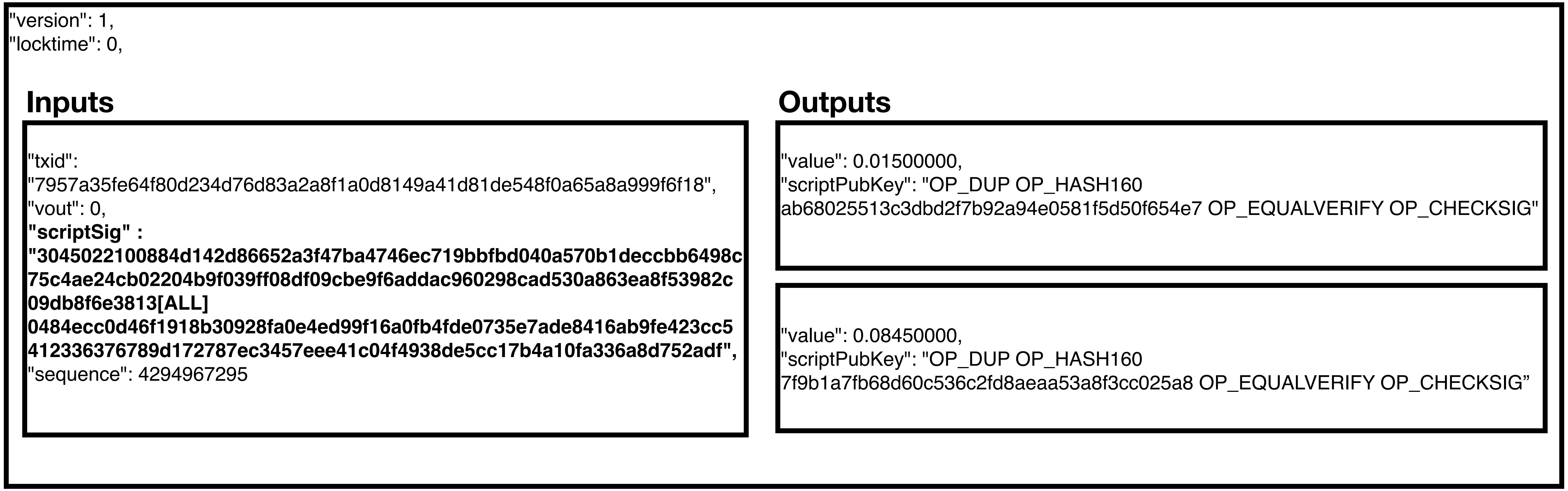
```
"value": 0.01500000,
"scriptPubKey": "OP_DUP OP_HASH160
ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
```

```
"value": 0.08450000,
"scriptPubKey": "OP_DUP OP_HASH160
7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG”
```

```
“vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
```

Looking at the outputs for 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

Transactions



"version": 1,
"locktime": 0,

Inputs

"txid":
"7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
"vout": 0,
"scriptSig" :
"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813[ALL]
0484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adf",
"sequence": 4294967295

Outputs

"value": 0.01500000,
"scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"

"value": 0.08450000,
"scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG”

```
“vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
```

Looking at the outputs for 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

Inputs

- They **spend outputs** (UTXOs)
- **Tx ID** and **Vout** index refer to which UTXO is being consumed
- **scriptSig** is an unlocking script (usually a signature input), which is used in tandem with the **scriptPubKey**
  - Psst. Segwit changed this, but it's mostly an implementation detail that you should augment after you understand this!
- <u>**sequence number**</u> is used to allow for updates to the inputs

Inputs

- In some transactions, we have Bitcoin that is input but not redeemed in the output
  - These are fees, collected by the miner
  - It gets added to the Coinbase transaction, which a transaction without inputs. The Coinbase transaction contains an output with a block reward + the fees to the miner's address
- You can use inputs from as many addresses as you like in a transaction, as long as you provide the signatures
- What happens if I have a 1 BTC UTXO, but only want to give 0.5 BTC?
  - I can give change to myself as one of the outputs in the transaction

Example coinbase

https://www.blockchain.com/btc/tx/c895aa4ae65fe0bb302968e8e46ab82cbfe005fac2bd2f3892b578a35f1579ef

Note: current block reward is 12.5 BTC, so this miner collected 0.732 BTC in fees!

Example with multiple inputs

https://www.blockchain.com/btc/tx/1e70886631e6f8aaa779c5477cea0f3ae9f953c1c679eee19f6f749d1c295947

Example transaction with change

https://www.blockchain.com/btc/tx/fb514ef140734b9143488fe624c25932604a81b003805937eb4be32504dd18d9

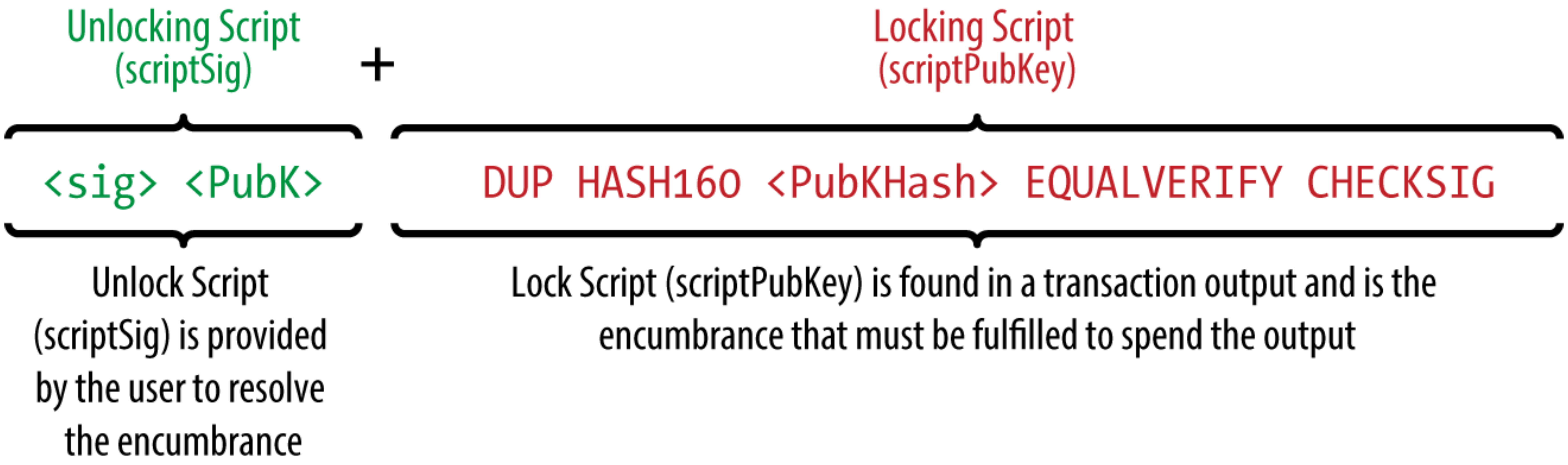
Script

- Most transactions are based on a pay to public key hash script
- When a tx is validated, **unlocking script** in each input is executed alongside the corresponding UTXO's **locking script** to see if it satisfies the locking condition
- Script is a stack based language designed to run on a range of hardware
- Limited in scope, no loops or recursion as a security feature
- Stateless execution

Script

- Sequence of op codes
- Executed in order by stack, after provided clause arguments placed on stack (we’ll see what this means)
- Can’t inspect other inputs or outputs in the same transaction
- Can’t directly control value, like smart contracts

Script



Unlocking Script
(scriptSig)

+

Locking Script
(scriptPubKey)

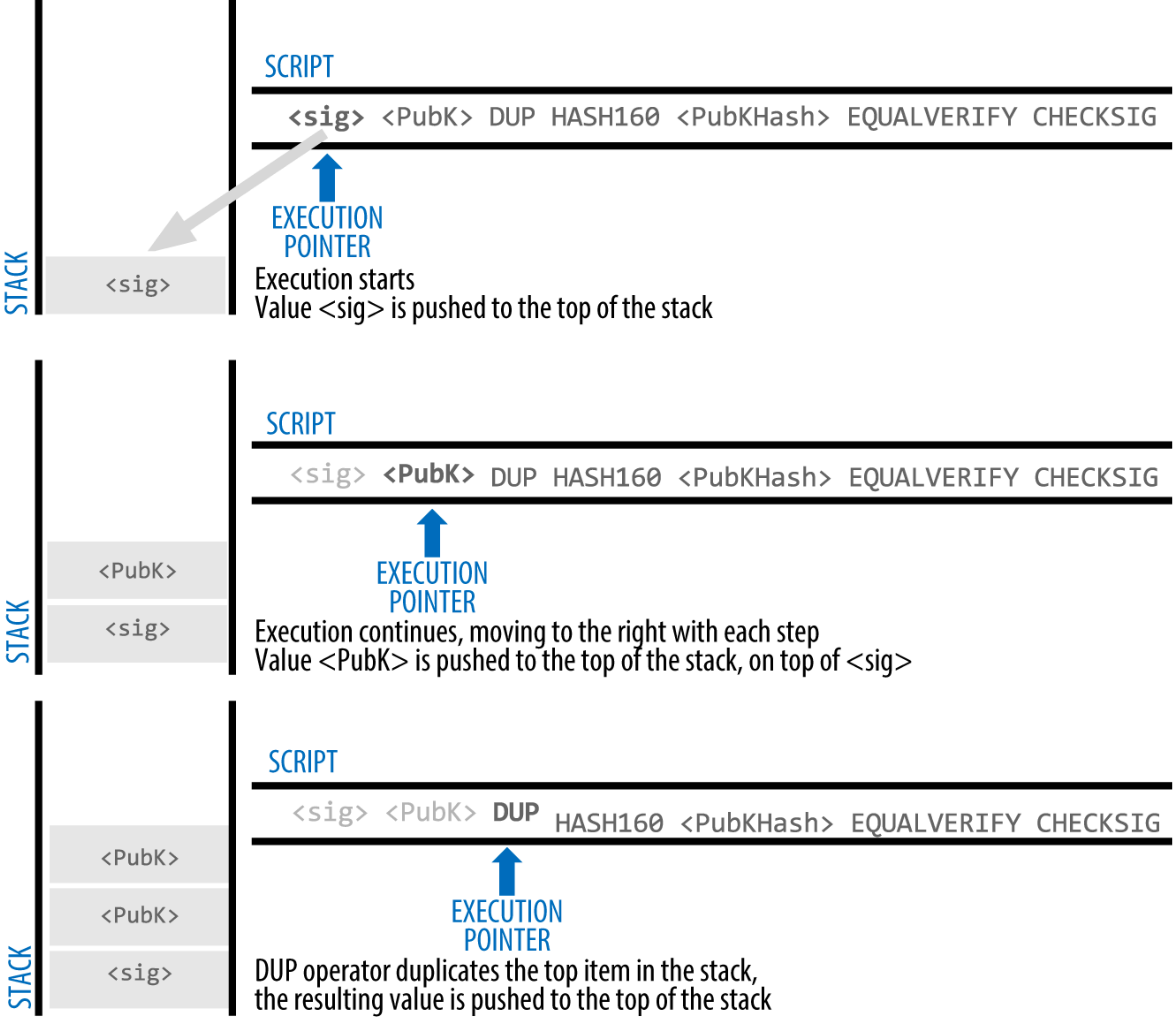
```
<sig> <PubK>
```

```
DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
```

Unlock Script (scriptSig) is provided by the user to resolve the encumbrance

Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

Note: Examples are from Mastering Bitcoin!



SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<sig>

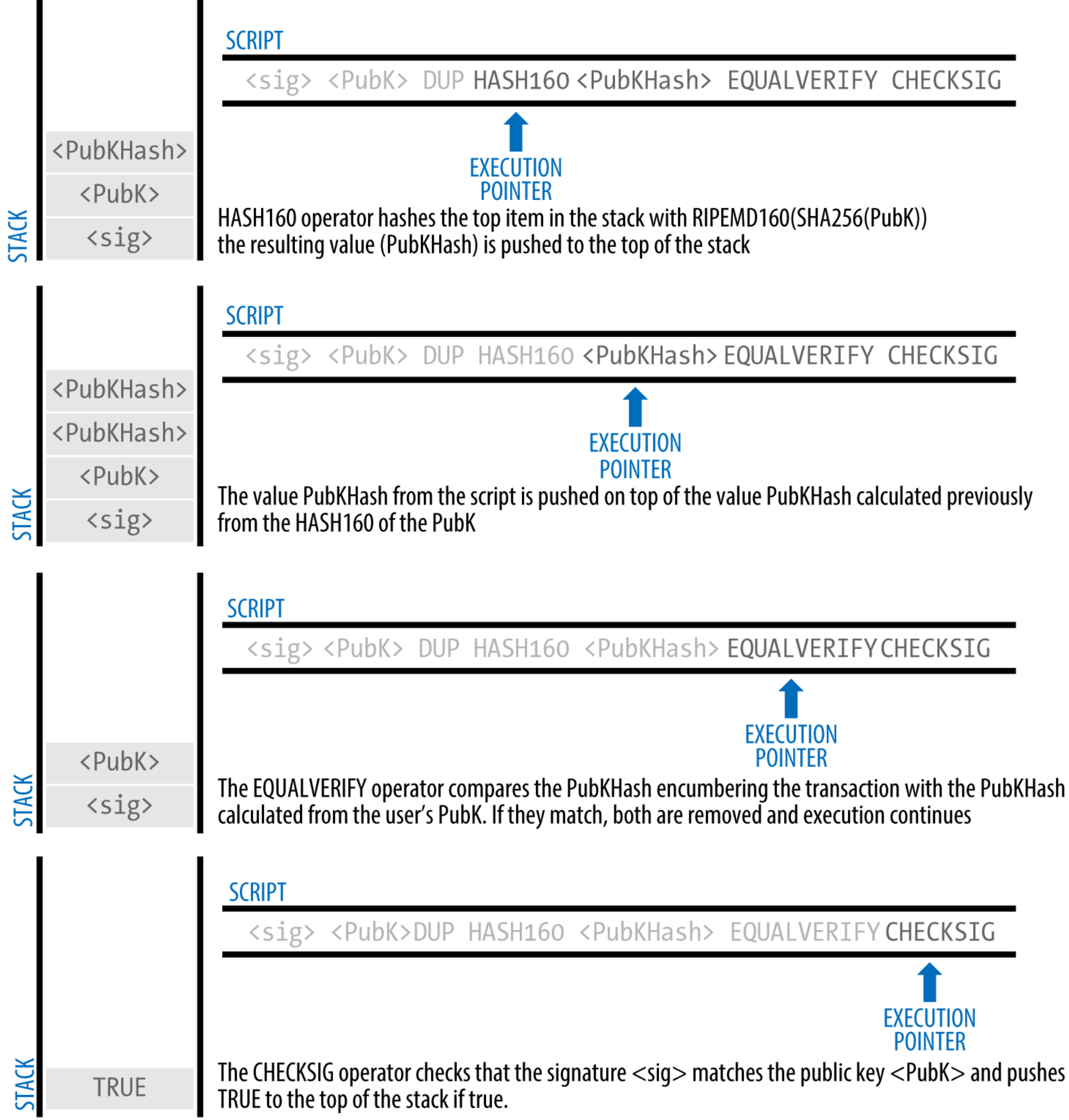
Execution starts
Value <sig> is pushed to the top of the stack

SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>

Execution continues, moving to the right with each step
Value <PubK> is pushed to the top of the stack, on top of <sig>

SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<PubK>
<sig>

DUP operator duplicates the top item in the stack,
the resulting value is pushed to the top of the stack



STACK
<PubKHash>
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER

HASH160 operator hashes the top item in the stack with RIPEMD160(SHA256(PubK))
the resulting value (PubKHash) is pushed to the top of the stack

STACK
<PubKHash>
<PubKHash>
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER

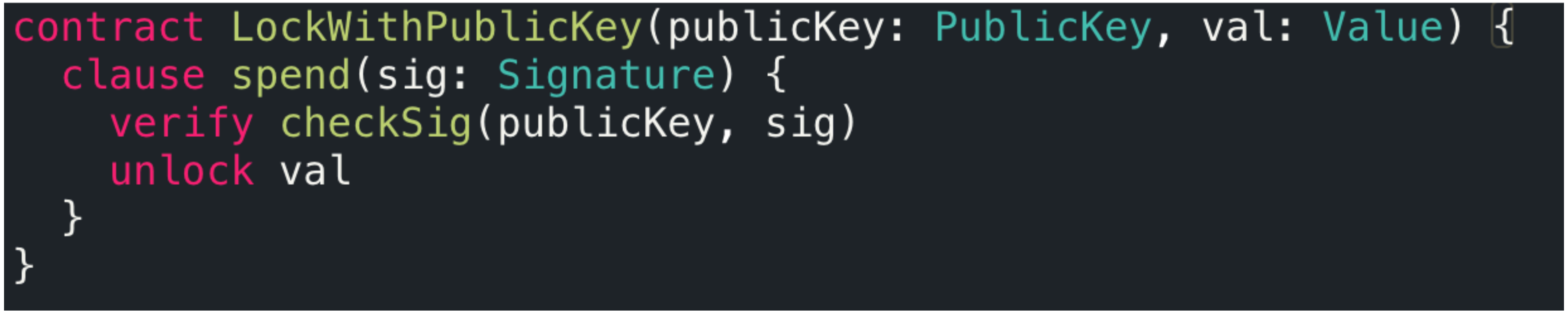
The value PubKHash from the script is pushed on top of the value PubKHash calculated previously from the HASH160 of the PubK

STACK
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER

The EQUALVERIFY operator compares the PubKHash encumbering the transaction with the PubKHash calculated from the user's PubK. If they match, both are removed and execution continues

The CHECKSIG operator checks that the signature <sig> matches the public key <PubK> and pushes TRUE to the top of the stack if true.

Ivy Lang

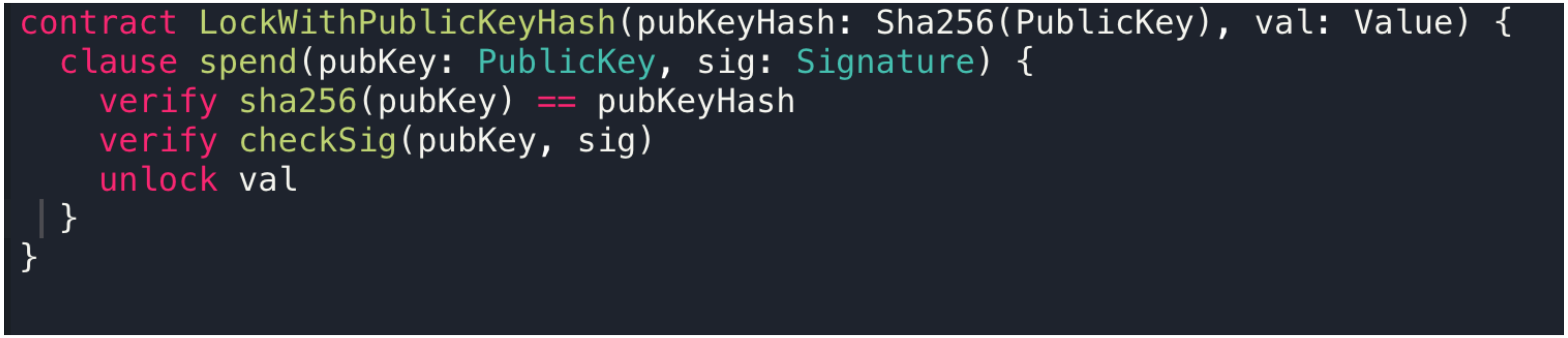


```
contract LockWithPublicKey(publicKey: PublicKey, val: Value) {
  clause spend(sig: Signature) {
    verify checkSig(publicKey, sig)
    unlock val
  }
}
```

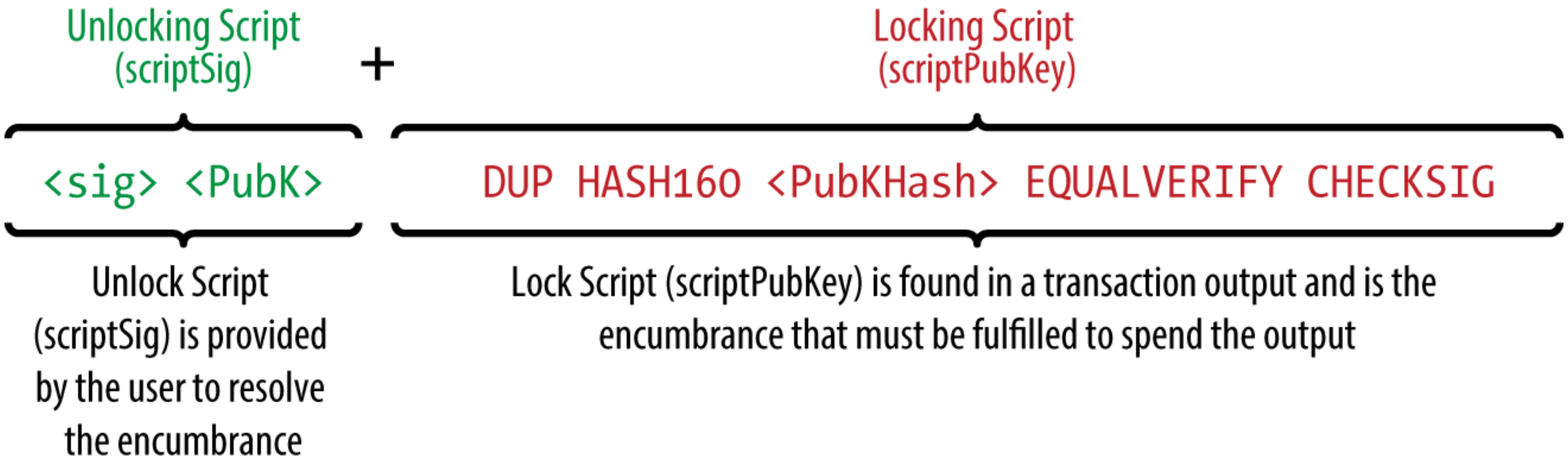
Named “clauses”

Thanks to Dan Robinson for the slide content and creating Ivy Lang!

Lock with Public Key Hash



```
contract LockWithPublicKeyHash(pubKeyHash: Sha256(PublicKey), val: Value) {
  clause spend(pubKey: PublicKey, sig: Signature) {
    verify sha256(pubKey) == pubKeyHash
    verify checkSig(pubKey, sig)
    unlock val
  }
}
```



Unlocking Script (scriptSig) + Locking Script (scriptPubKey)

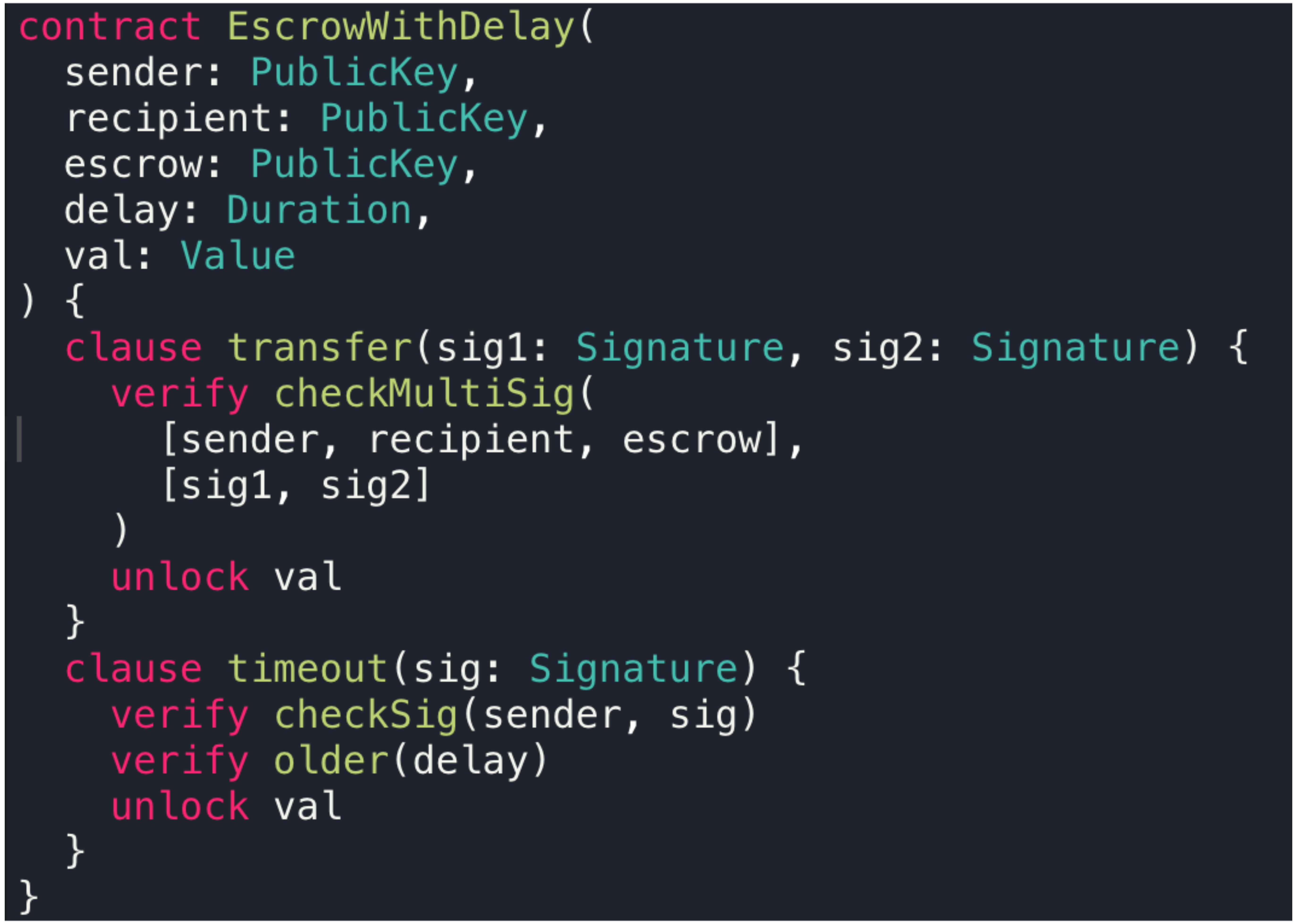
<sig> <PubK>

DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG

Unlock Script (scriptSig) is provided by the user to resolve the encumbrance

Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

Escrow with Timeout



```
contract EscrowWithDelay(
  sender: PublicKey,
  recipient: PublicKey,
  escrow: PublicKey,
  delay: Duration,
  val: Value
) {
  clause transfer(sig1: Signature, sig2: Signature) {
    verify checkMultiSig(
      [sender, recipient, escrow],
      [sig1, sig2]
    )
    unlock val
  }
  clause timeout(sig: Signature) {
    verify checkSig(sender, sig)
    verify older(delay)
    unlock val
  }
}
```

Break time

Payment Channels

- It’s an off-chain ledger, between two parties
- Use case: scalable, recurring payments
- 2-party consensus
- Private - only shows the net result on chain when closing
- Fast - all transactions are just done through message passing and signature sharing
- Cheap - no fees, except for opening and closing one

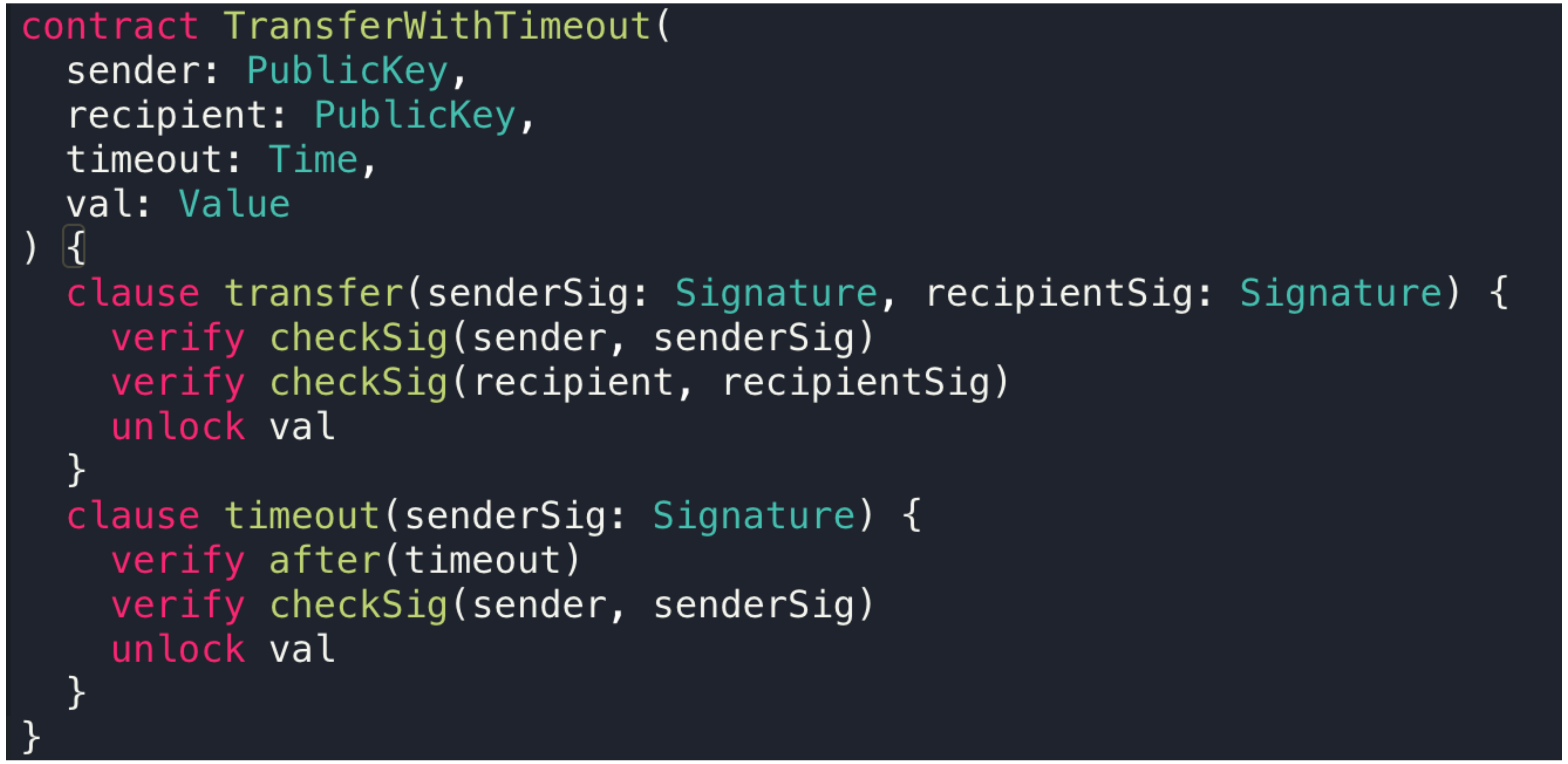
What allows for payments channels to exist

- Quorums of control (multisig)
- Timelocks - in script, **CheckLockTimeVerify** (CLTV) for actual time or specific block height or **CheckSequenceVerify** (CSV) for setting a counter from the time the transaction gets published
- No double spends
- Non-expiration
- Censorship resistance
- Authentication

Unilateral Payment Channel

- Step 1: Create multisig, 2 of 2. It needs to be timed, to handle the case where someone might go offline.
- Step 2: Figure out how to send signable offline transactions
- Step 3: Hope script works

TransferWithTimeOut



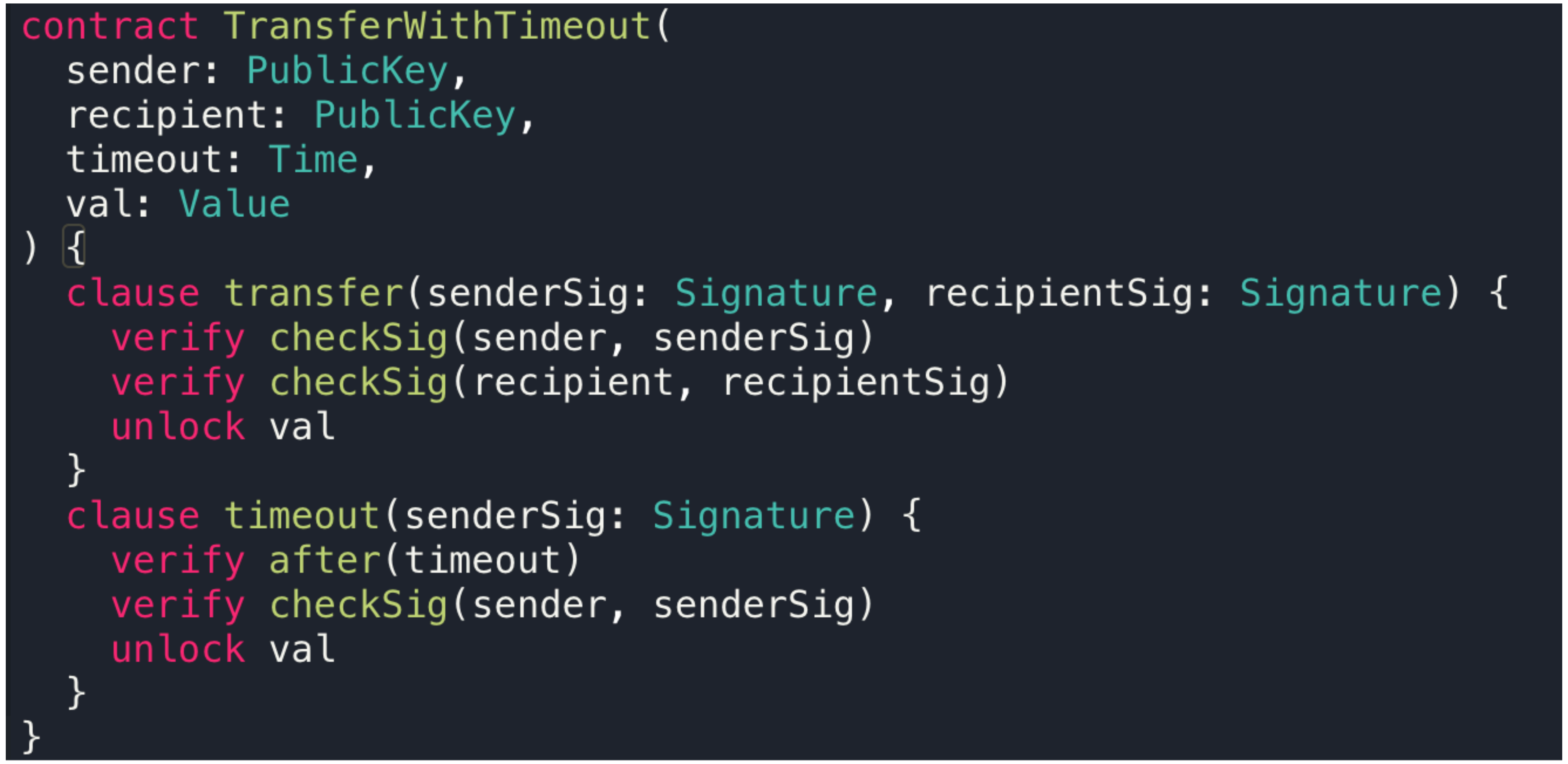
```
contract TransferWithTimeout(
  sender: PublicKey,
  recipient: PublicKey,
  timeout: Time,
  val: Value
) {
  clause transfer(senderSig: Signature, recipientSig: Signature) {
    verify checkSig(sender, senderSig)
    verify checkSig(recipient, recipientSig)
    unlock val
  }
  clause timeout(senderSig: Signature) {
    verify after(timeout)
    verify checkSig(sender, senderSig)
    unlock val
  }
}
```

- Alice wants to pay micropayments to Bob

- Alice can pre-fund this TWT with 10 BTC

- Then she can create transactions that **transfer** 0.0001 to Bob and return the rest to her

TransferWithTimeOut

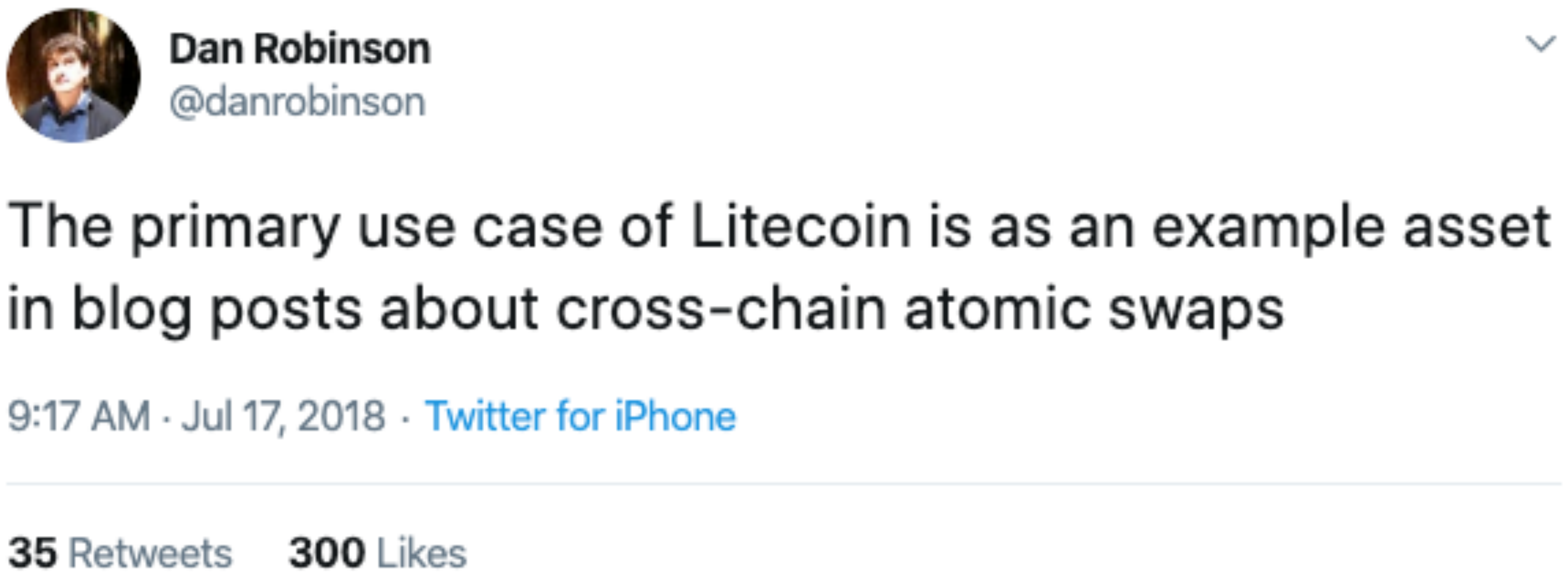


```
contract TransferWithTimeout(
  sender: PublicKey,
  recipient: PublicKey,
  timeout: Time,
  val: Value
) {
  clause transfer(senderSig: Signature, recipientSig: Signature) {
    verify checkSig(sender, senderSig)
    verify checkSig(recipient, recipientSig)
    unlock val
  }
  clause timeout(senderSig: Signature) {
    verify after(timeout)
    verify checkSig(sender, senderSig)
    unlock val
  }
}
```

- What happens if Bob disappears? Timeout!

Bilateral Payment Channels

- Intentionally left blank.



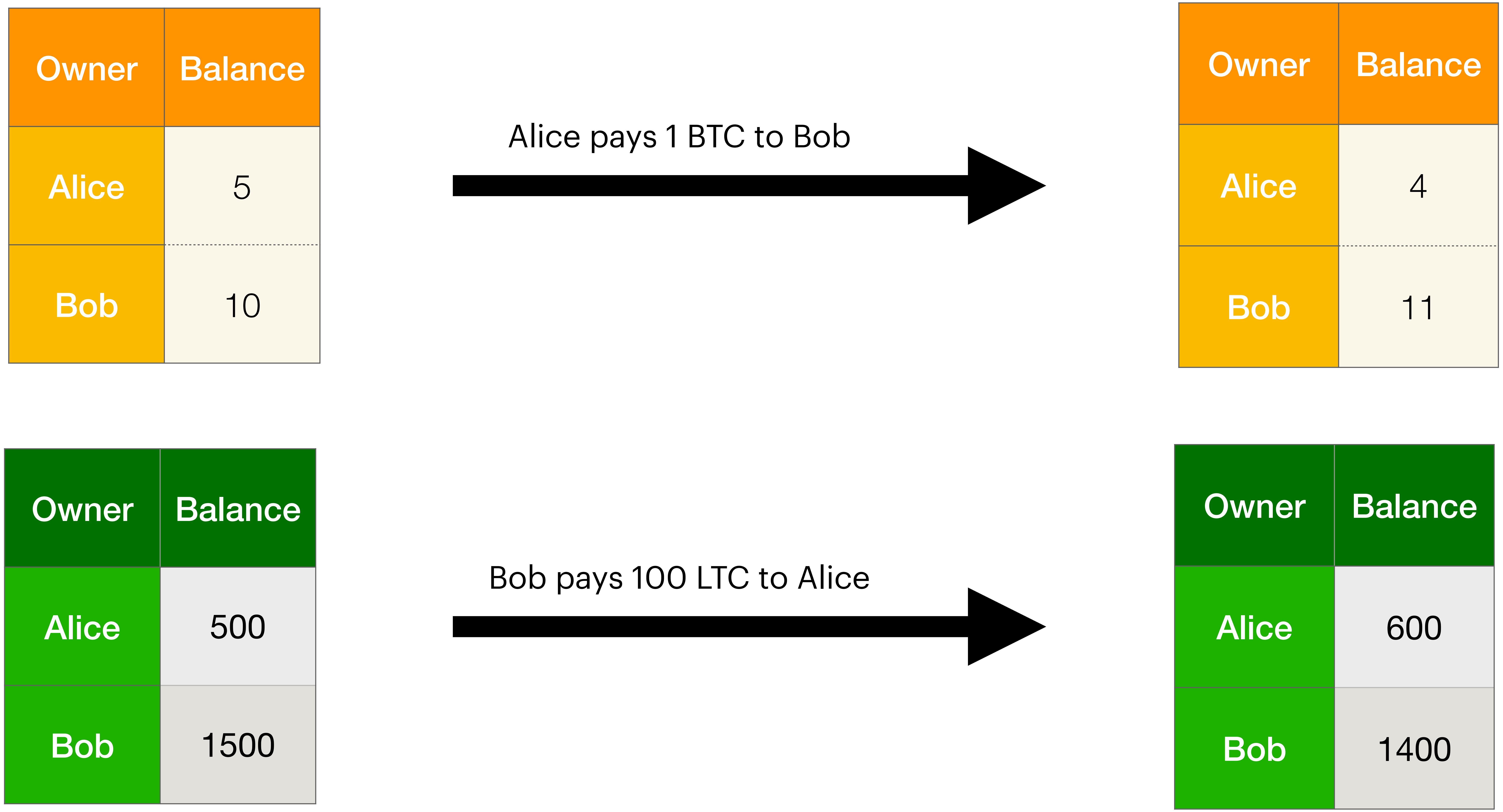
Dan Robinson
@danrobinson

The primary use case of Litecoin is as an example asset in blog posts about cross-chain atomic swaps

9:17 AM · Jul 17, 2018 · Twitter for iPhone

35 Retweets **300** Likes

Atomic Swap



Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob

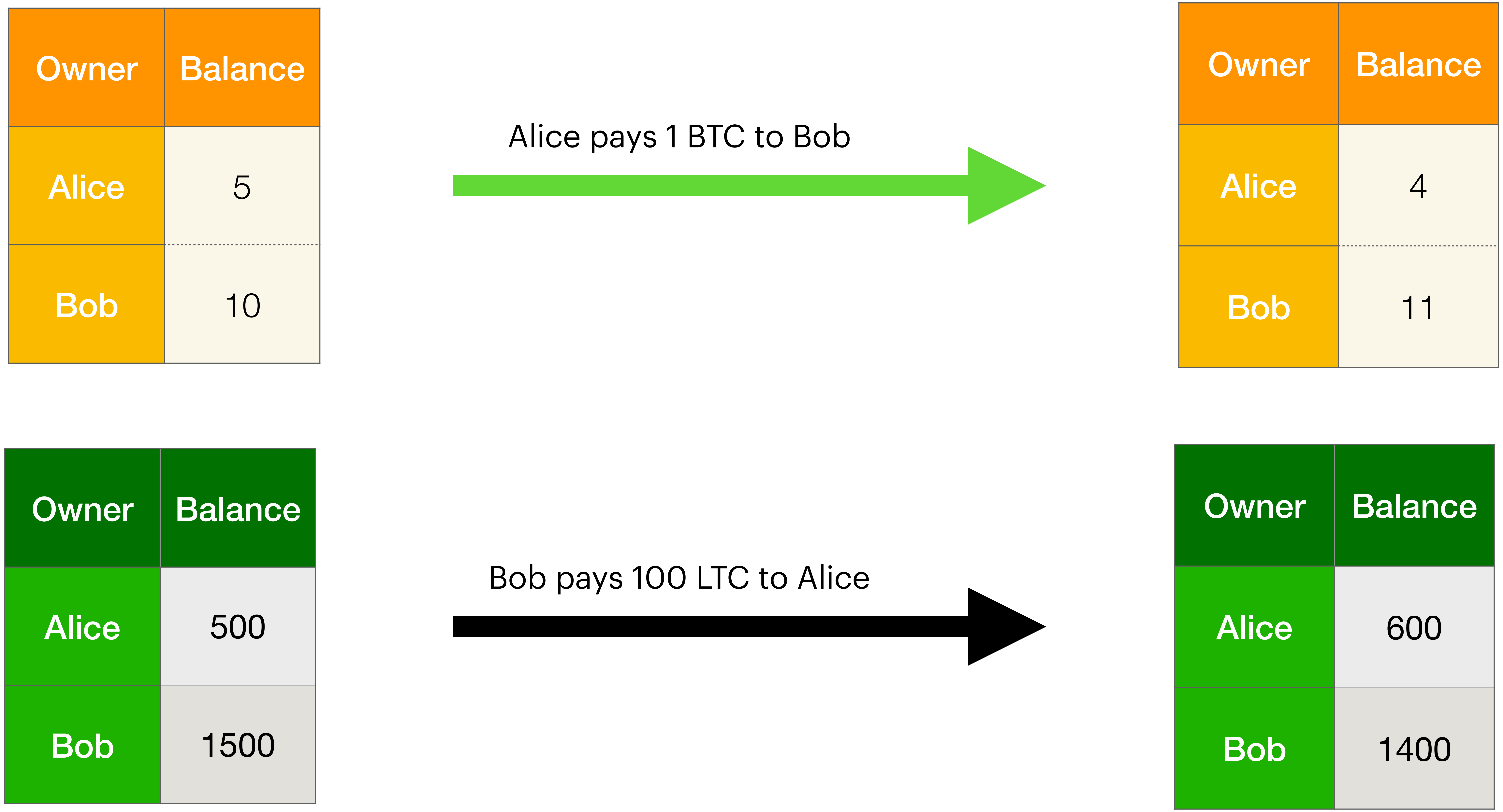
Owner	Balance
Alice	4
Bob	11

Owner	Balance
Alice	500
Bob	1500

Bob pays 100 LTC to Alice

Owner	Balance
Alice	600
Bob	1400

Atomic Swap



Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob

Owner	Balance
Alice	4
Bob	11

Owner	Balance
Alice	500
Bob	1500

Bob pays 100 LTC to Alice

Owner	Balance
Alice	600
Bob	1400

Atomic Swap

Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob

Owner	Balance
Alice	4
Bob	11

Owner	Balance
Alice	500
Bob	1500

Bob pays 100 LTC to Alice

Owner	Balance
Alice	600
Bob	1400

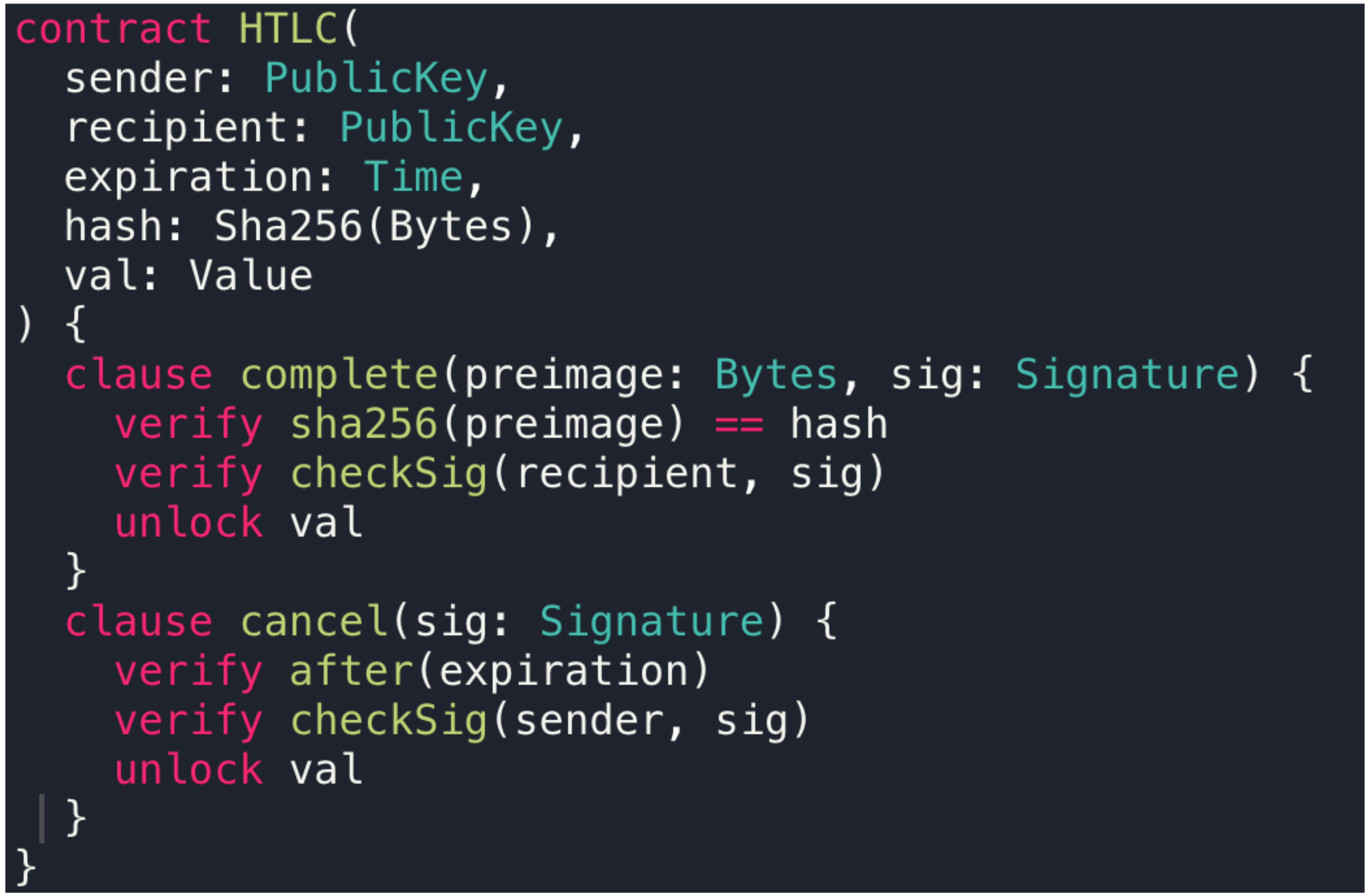
HTLCs

- Hashed Timelock Contracts
- They are a means of providing cross ledger atomic transactions. Either both transaction complete or them don't.

HTLCs

- **Hashlocks** - to restrict spending of funds locked in a contract
- **Timelocks** - act as fail safe, timeout
- **Protocol**
  - Agree to a hashed pre-image, where the pre-image is known by either sender or recipient
  - Agree to exchange rate and time lock period
  - Set up HTLCs on both chains/ledgers
  - Reveal pre-image when ready to transfer

HTLCs



```
contract HTLC(
  sender: PublicKey,
  recipient: PublicKey,
  expiration: Time,
  hash: Sha256(Bytes),
  val: Value
) {
  clause complete(preimage: Bytes, sig: Signature) {
    verify sha256(preimage) == hash
    verify checkSig(recipient, sig)
    unlock val
  }
  clause cancel(sig: Signature) {
    verify after(expiration)
    verify checkSig(sender, sig)
    unlock val
  }
}
```

- A single HTLC is not useful by itself—it is simply a construction that promises to reward a particular recipient for revealing a preimage before a particular time

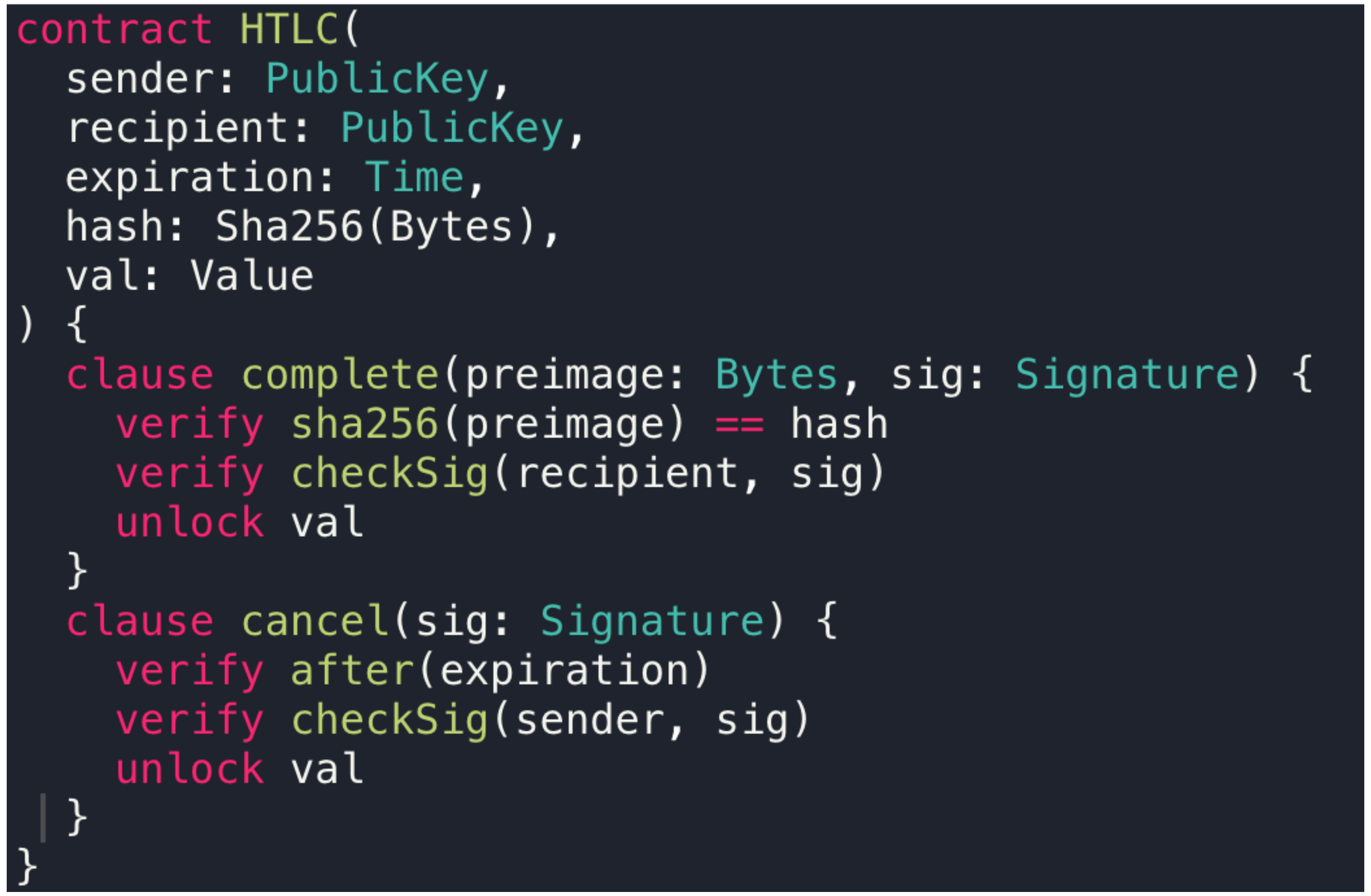
HTLCs

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10
HTLC (Alice’s preimage)	1

Owner	Balance
Alice	500
Bob	1500

Owner	Balance
Alice	500
Bob	1500
HTLC (Alice’s preimage)	100



```
contract HTLC(
  sender: PublicKey,
  recipient: PublicKey,
  expiration: Time,
  hash: Sha256(Bytes),
  val: Value
) {
  clause complete(preimage: Bytes, sig: Signature) {
    verify sha256(preimage) == hash
    verify checkSig(recipient, sig)
    unlock val
  }
  clause cancel(sig: Signature) {
    verify after(expiration)
    verify checkSig(sender, sig)
    unlock val
  }
}
```

HTLCs (happy case)

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10
HTLC (Alice’s preimage)	1

Owner	Balance
Alice	500
Bob	1500

Owner	Balance
Alice	500
Bob	1500
HTLC (Alice’s preimage)	100

- Example: Alice wants to swap 1 BTC for Bob’s 100 LTC
- Alice locks 1 BTC into 48-hour HTLC, using hash of Alice’s pre-image
- Bob locks 100 LTC into 24 hour HTC with same hash
- Alice reveals her pre-image to complete Litecoin HTLC

HTLCs (happy case)

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10
HTLC (Alice’s preimage)	1

Owner	Balance
Alice (LWPKH)	4
Bob (LWPKH)	11

- Bob uses the secret to complete the Bitcoin HTLC

Owner	Balance
Alice	500
Bob	1500

Owner	Balance
Alice	500
Bob	1500
HTLC (Alice’s preimage)	100

Owner	Balance
Alice	600
Bob	1400

HTLCs (unhappy case)

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10

→

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10
HTLC (Alice’s preimage)	1

→

Owner	Balance
Alice (LWPKH)	5
Bob (LWPKH)	10

- Alice doesn’t reveal pre-image and doesn’t claim LTC
- Bob cancels the LTC HTLC after 24 hours
- Alice cancels the BTC HTLC after 48 hours

Owner	Balance
Alice	500
Bob	1500

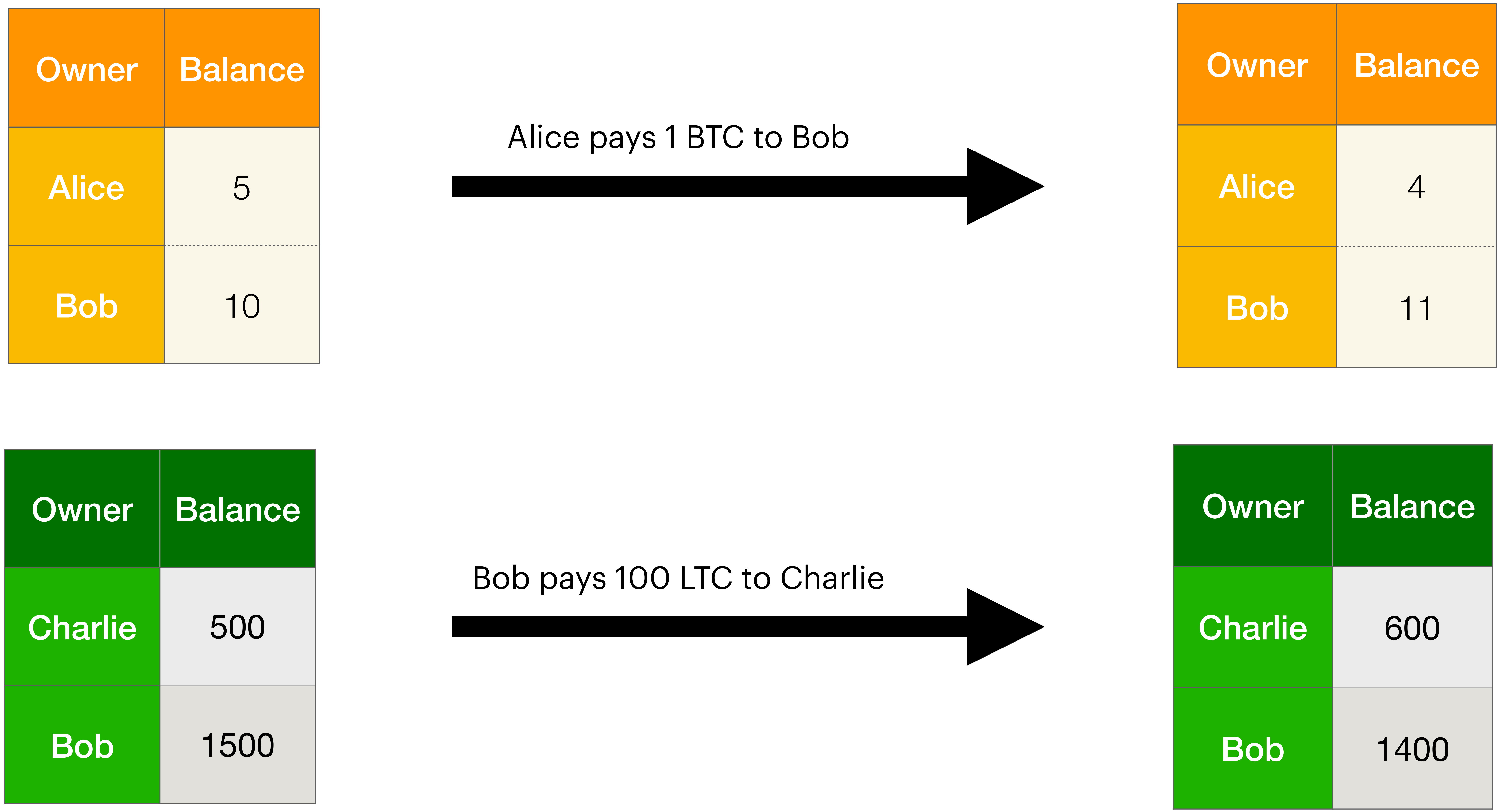
→

Owner	Balance
Alice	500
Bob	1500
HTLC (Alice’s preimage)	100

→

Owner	Balance
Alice	500
Bob	1500

Cross-chain atomic payment



Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob

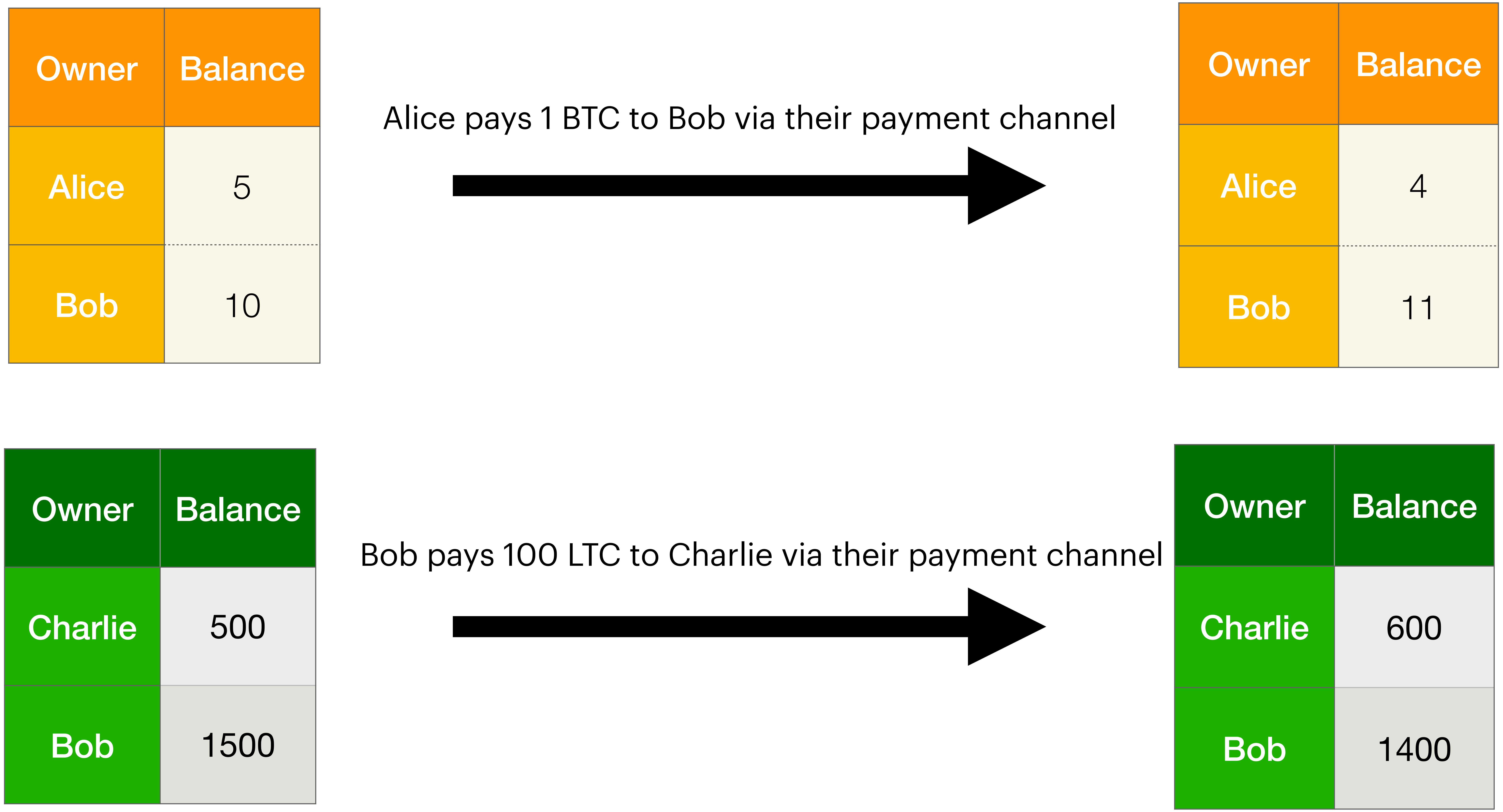
Owner	Balance
Alice	4
Bob	11

Owner	Balance
Charlie	500
Bob	1500

Bob pays 100 LTC to Charlie

Owner	Balance
Charlie	600
Bob	1400

Cross-Payment Channel atomic payment



Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob via their payment channel

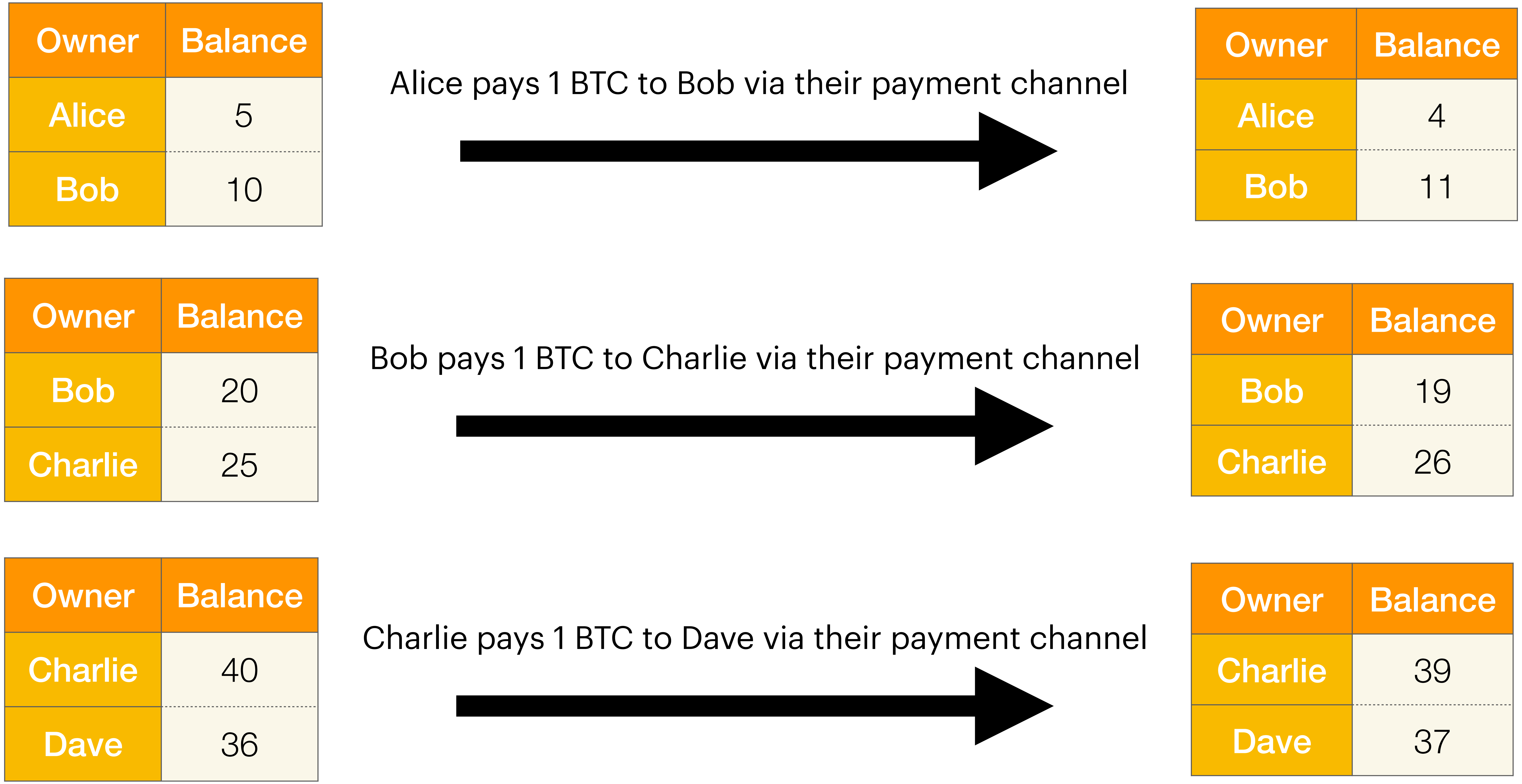
Owner	Balance
Alice	4
Bob	11

Owner	Balance
Charlie	500
Bob	1500

Bob pays 100 LTC to Charlie via their payment channel

Owner	Balance
Charlie	600
Bob	1400

Multi-Hop Cross-Payment Channel atomic payment



Owner	Balance
Alice	5
Bob	10

Alice pays 1 BTC to Bob via their payment channel

Owner	Balance
Alice	4
Bob	11

Owner	Balance
Bob	20
Charlie	25

Bob pays 1 BTC to Charlie via their payment channel

Owner	Balance
Bob	19
Charlie	26

Owner	Balance
Charlie	40
Dave	36

Charlie pays 1 BTC to Dave via their payment channel

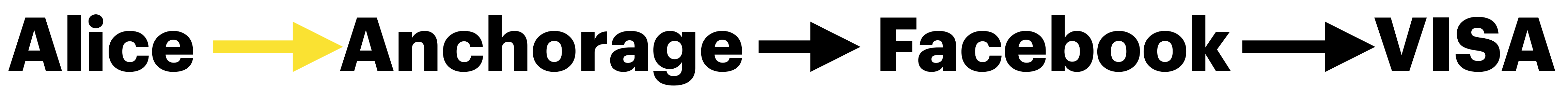
Owner	Balance
Charlie	39
Dave	37

Multi-Hop Payments

→ Payment channel, using TransferWithTimeout as the contract/UTXO as ledger

Alice → Anchorage → Facebook → VISA

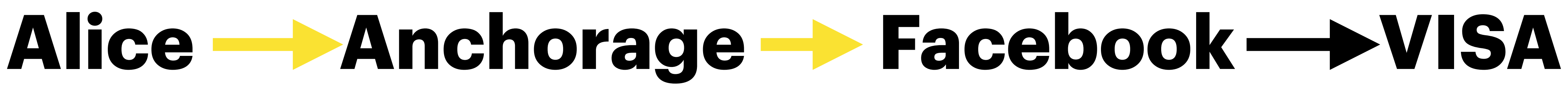
Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

Alice locks 1 BTC into HTLC

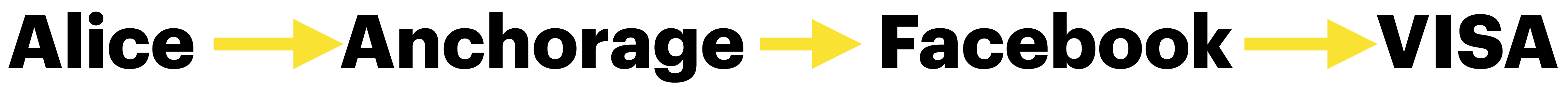
Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

Anchorage locks 1 BTC into HTLC

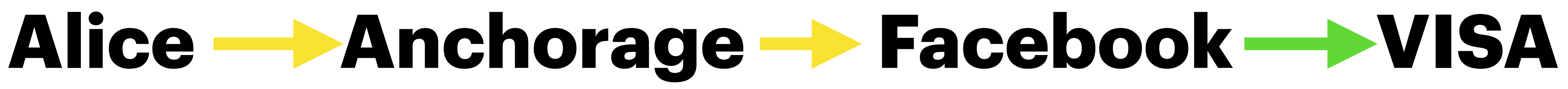
Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

Facebook locks 1 BTC into HTLC

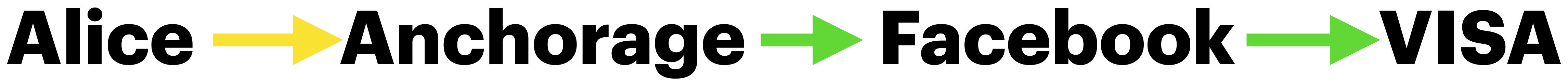
Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

VISA completes HTLC

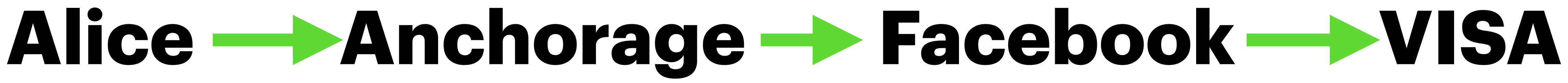
Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

Facebook completes HTLC

Multi-Hop Payments



Alice → Anchorage → Facebook → VISA

Alice completes HTLC

It’s comfortable to be mostly practical or theoretical. Get really good at connecting the two, and you’re well on your way to wealth.

Feel free to reach out

- DM on Twitter: @akash_khosla
- Send me an email: hello@akashkhosla.com
- Signal/iMessage

Where to go next

- Try out Zap Wallet, play with LND, check resources on assignment
- Other interesting Bitcoin stuff
  - tBTC
  - Taproot
  - Schnorr