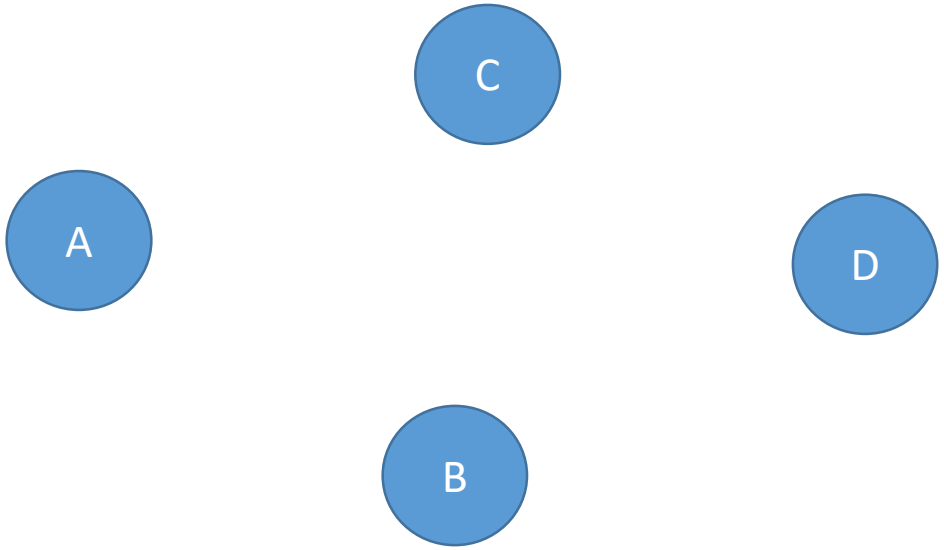
Block Chain

A protocol view

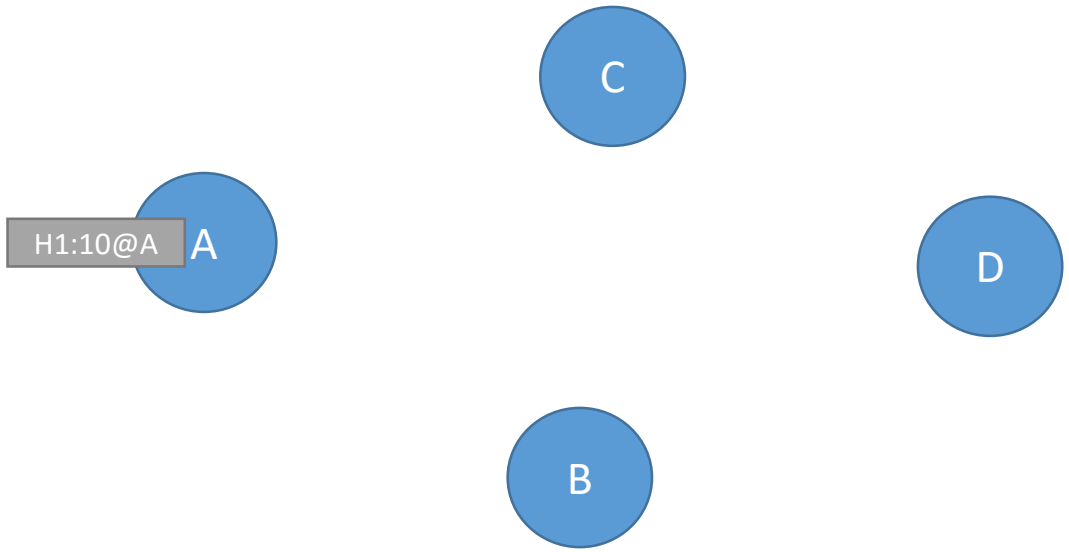
fanfeilong@outlook.com

Block Chain Protocol



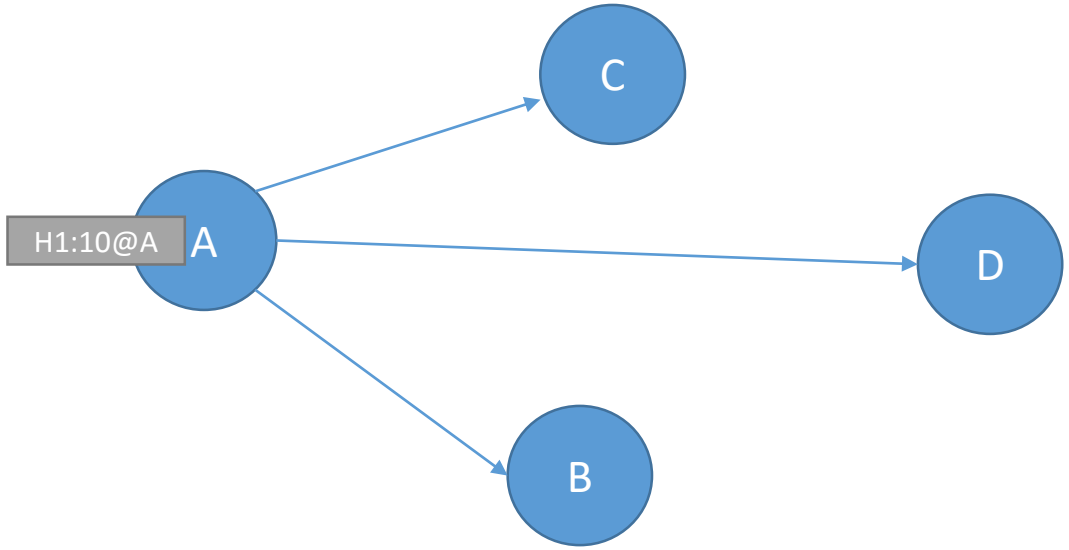
A P2P network, the detail of P2P network will be ignored in this document, since it is another topic.

Block Chain Protocol



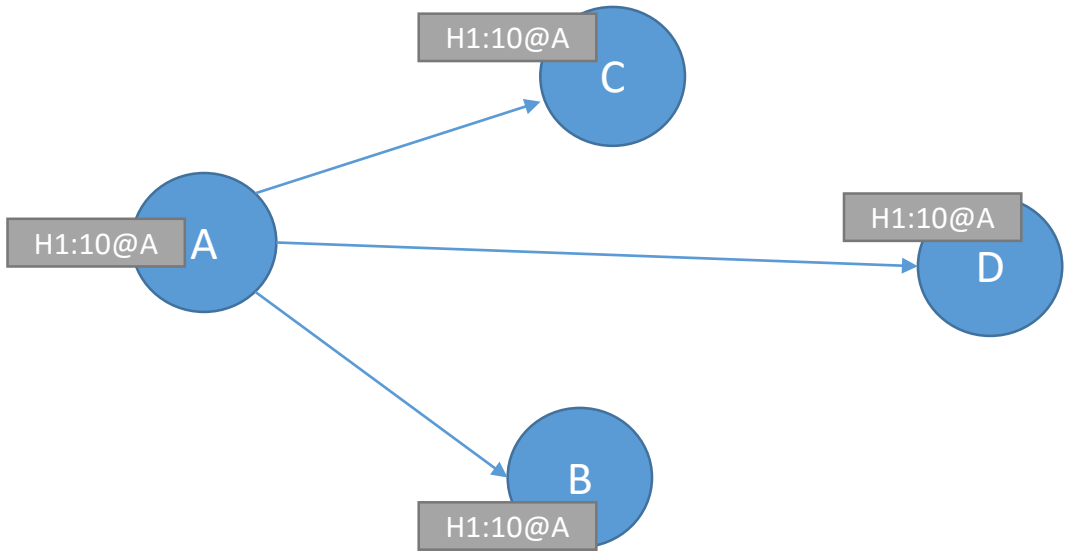
A creates a block with 10 virtual currency (or anything you can imagine)

Block Chain Protocol



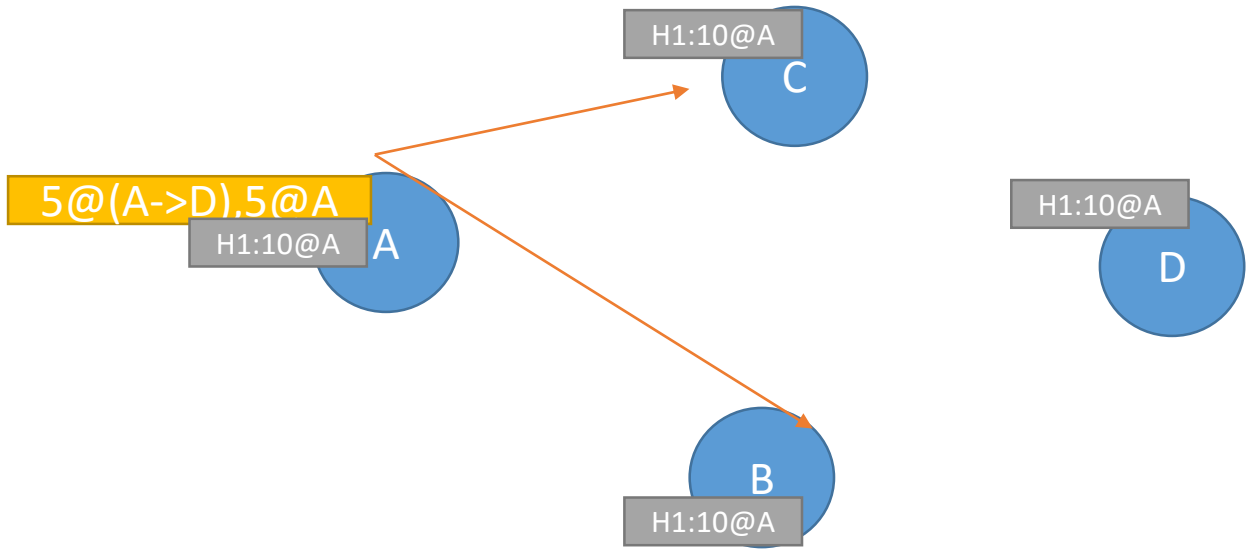
A broadcasts this block to the P2P network

Block Chain Protocol



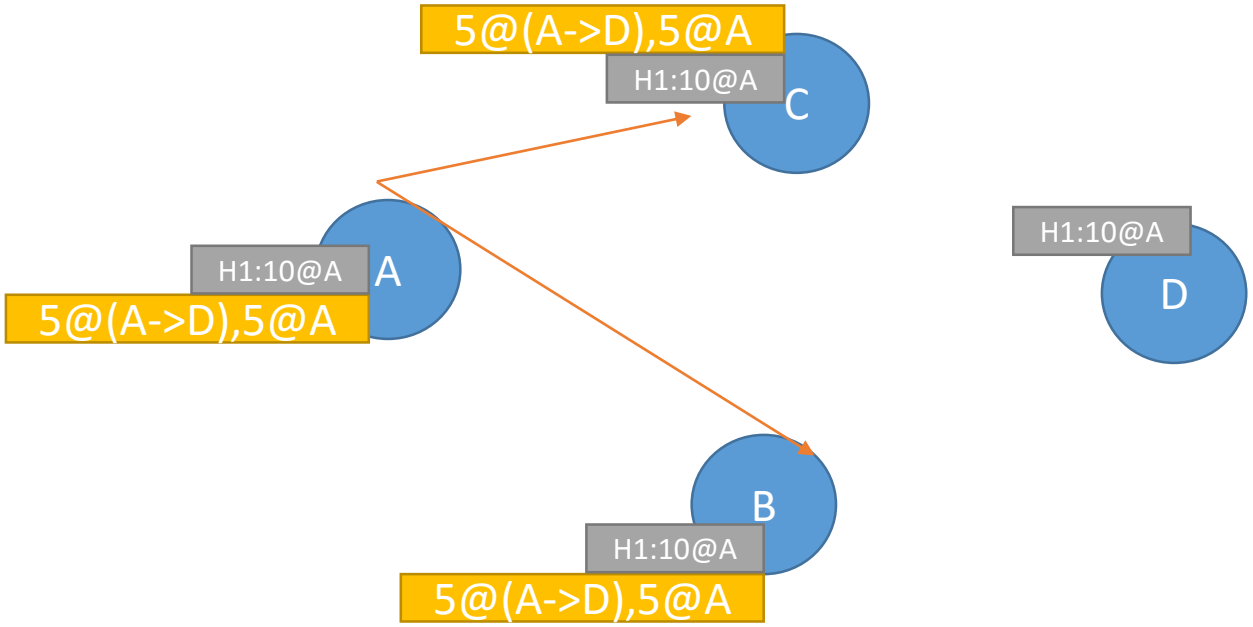
A,B,C in the P2P network will all have the block chain

Block Chain Protocol



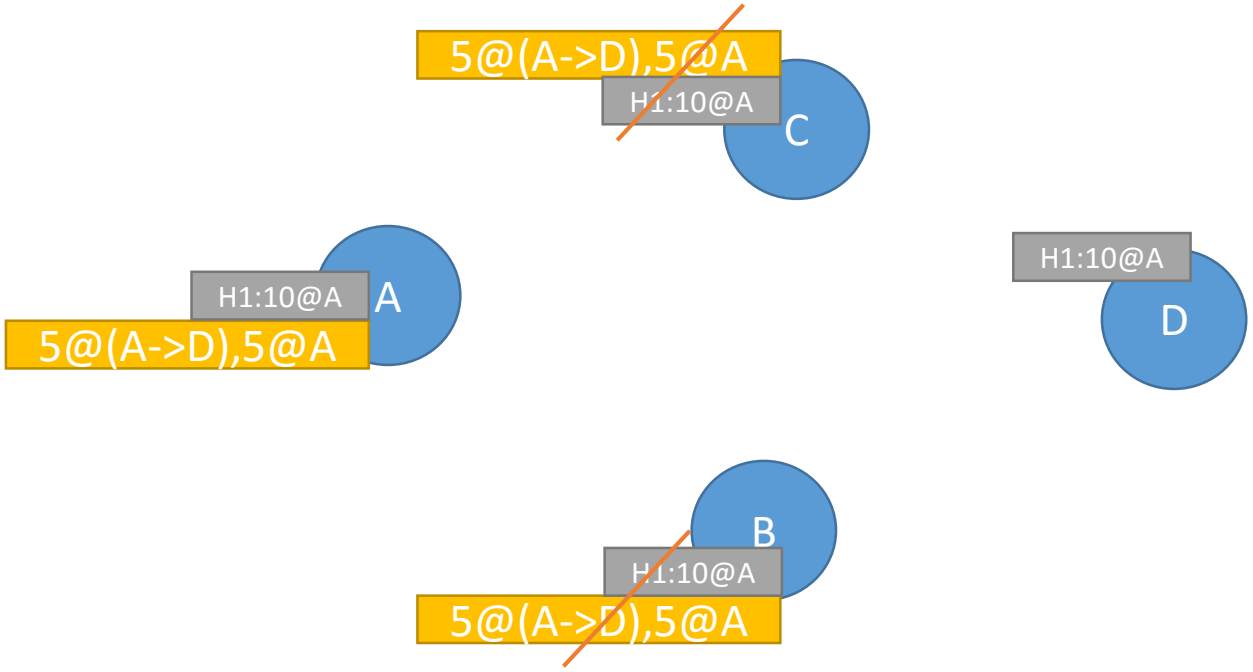
A try to transfer 5 to D, A broadcasts a transaction message to the P2P network

Block Chain Protocol



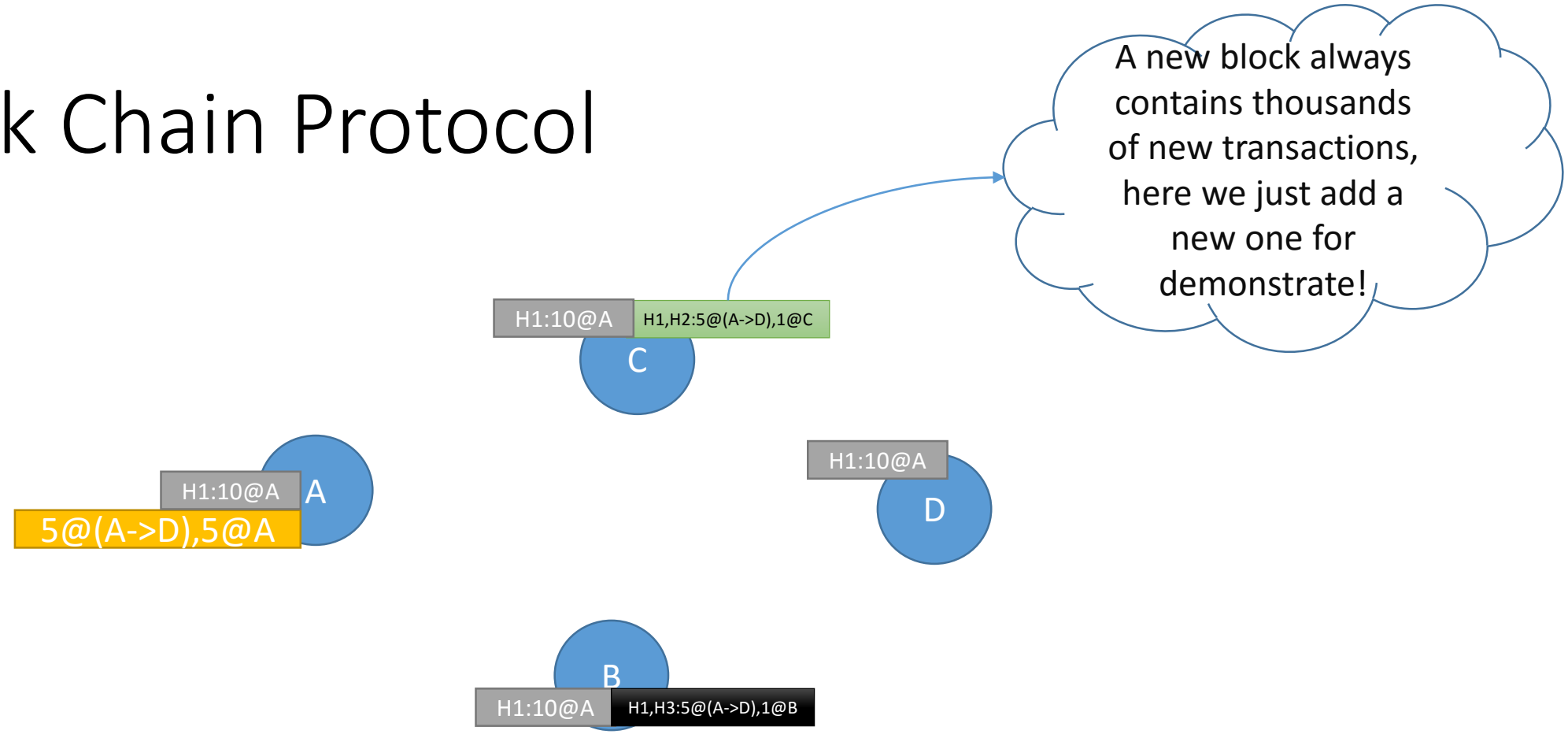
B and C Receive the transaction message

Block Chain Protocol



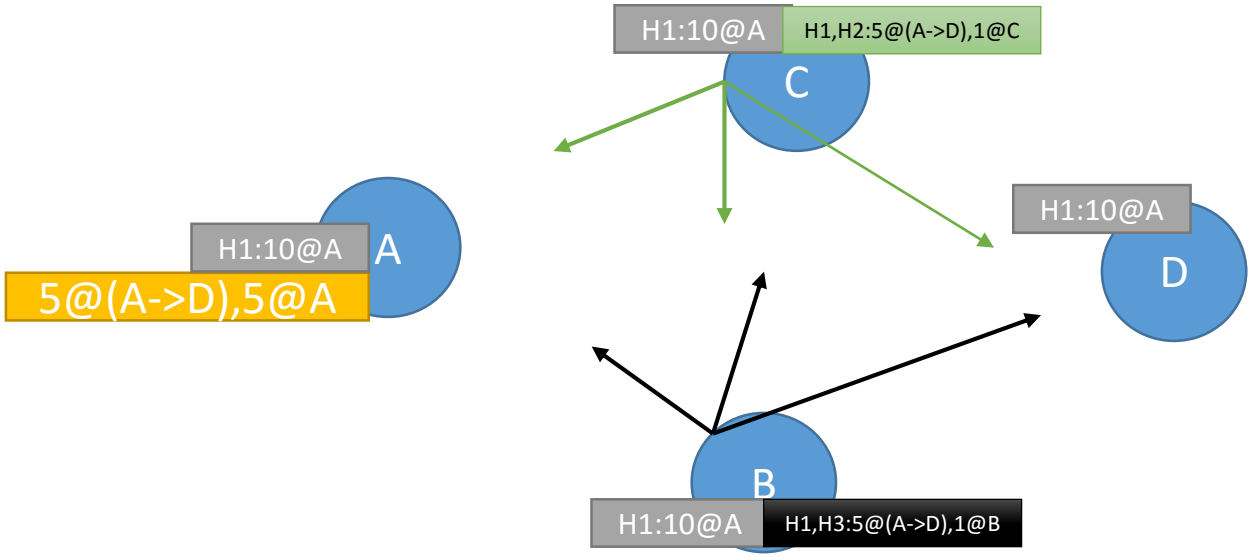
B and C validate the message by both block chain and the transaction message.
In fact, the verify process use the **RSA** algorithm and **P2PKH scripts** or **P2SH scripts** which is so called **smart-contract**.

Block Chain Protocol



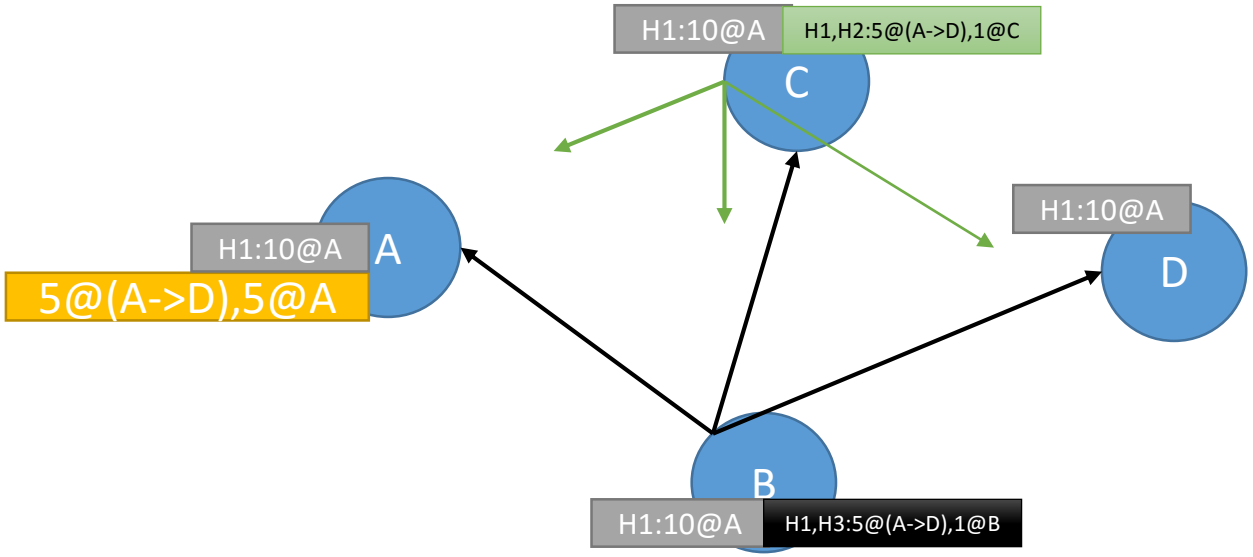
B and C combine the block chain and transaction to create a new block
Both B and C are doing a **Proof Of Work(POW)** to calculate a new hash value which **SHOULD** match some condition, it's not easy and **time consuming**.
B and C are both try to reward himself by 1.

Block Chain Protocol



B and C are both broadcasts the new block to the P2P network,
the Proof Of Work will ensure that the P2P network can product
a new block every ten minutes on average . So, B may be faster than C.

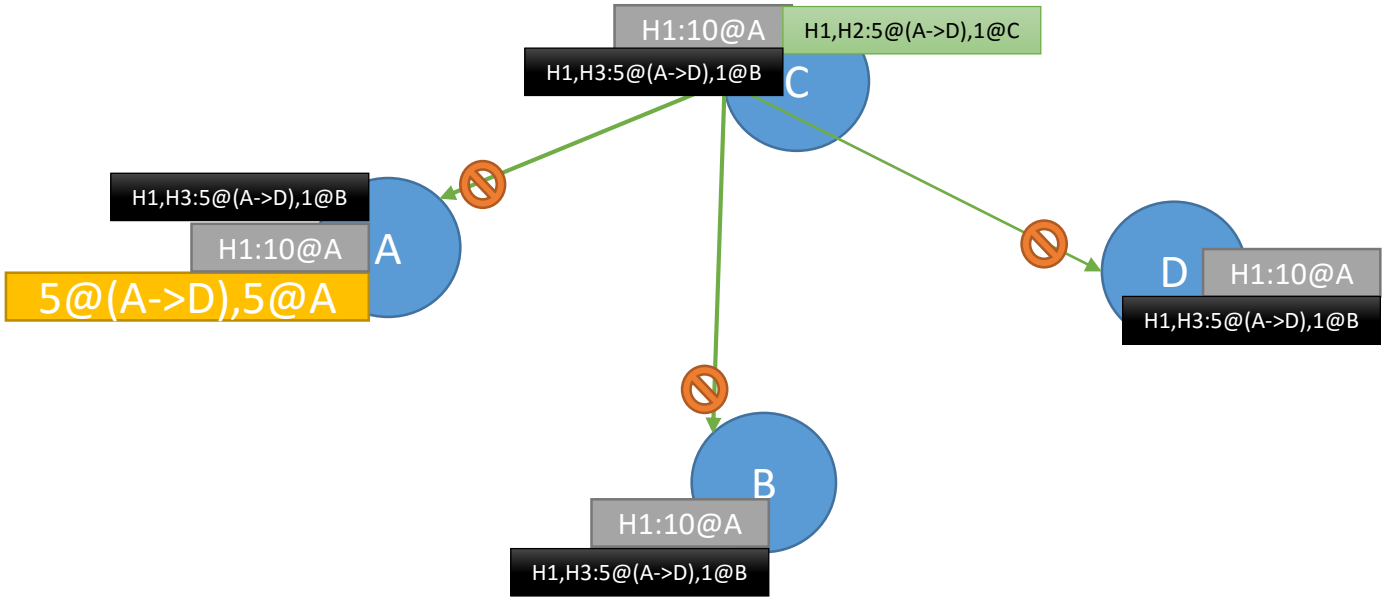
Block Chain Protocol



Assuming that B's block is first accepted by more than half of nodes in the P2P network, and the node D receive 5 from A.

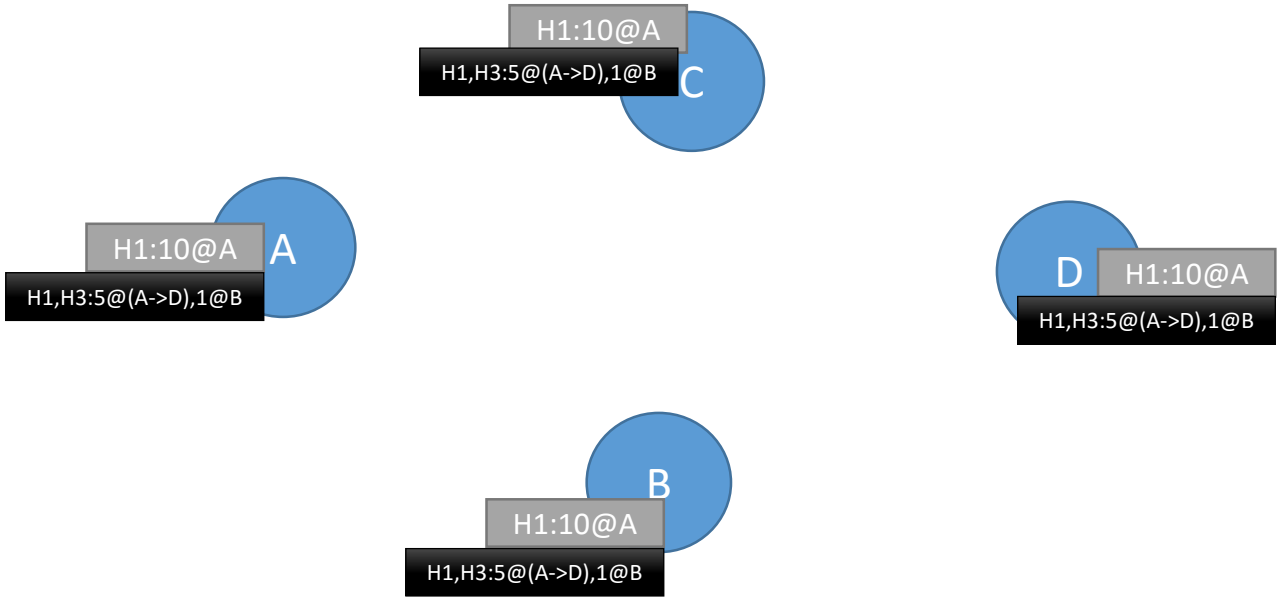
NOTE: Proof Of Work(**POW**) is important, which ensures the eventual consistency of the block chain.

Block Chain Protocol



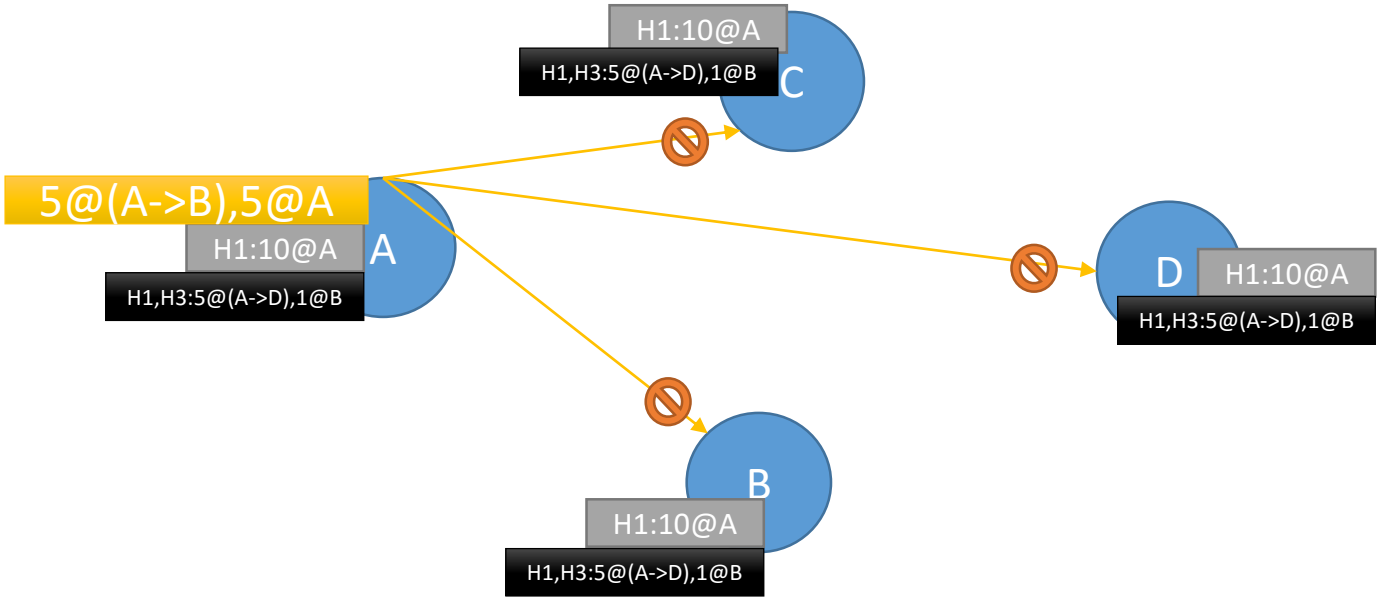
And then C's block will be discarded

Block Chain Protocol



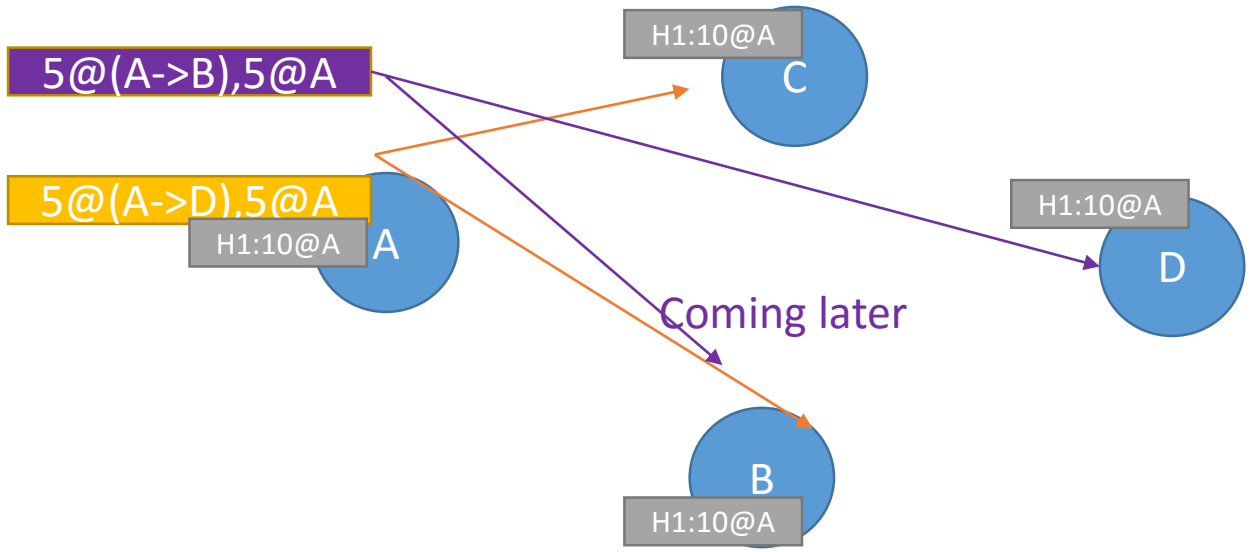
B win the 1

Block Chain Protocol

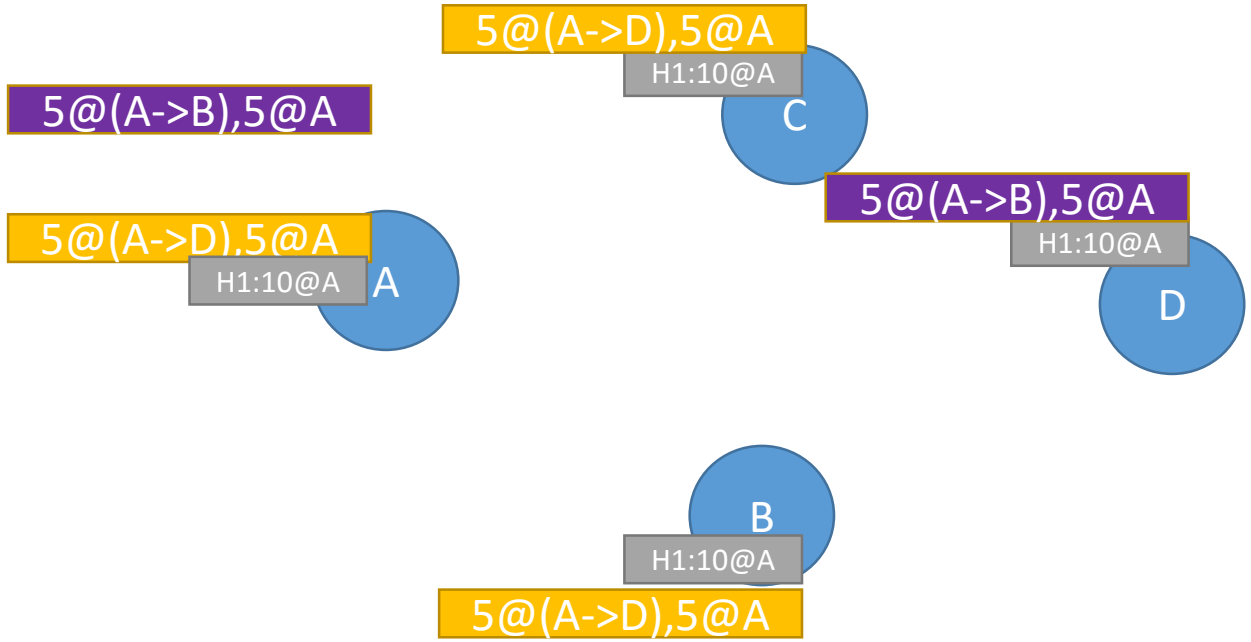


If A try to spend the old 10 again, it will be rejected by most nodes
Since all nodes can use the block chain to verify A's assets.
And the block chain contains all transaction records.

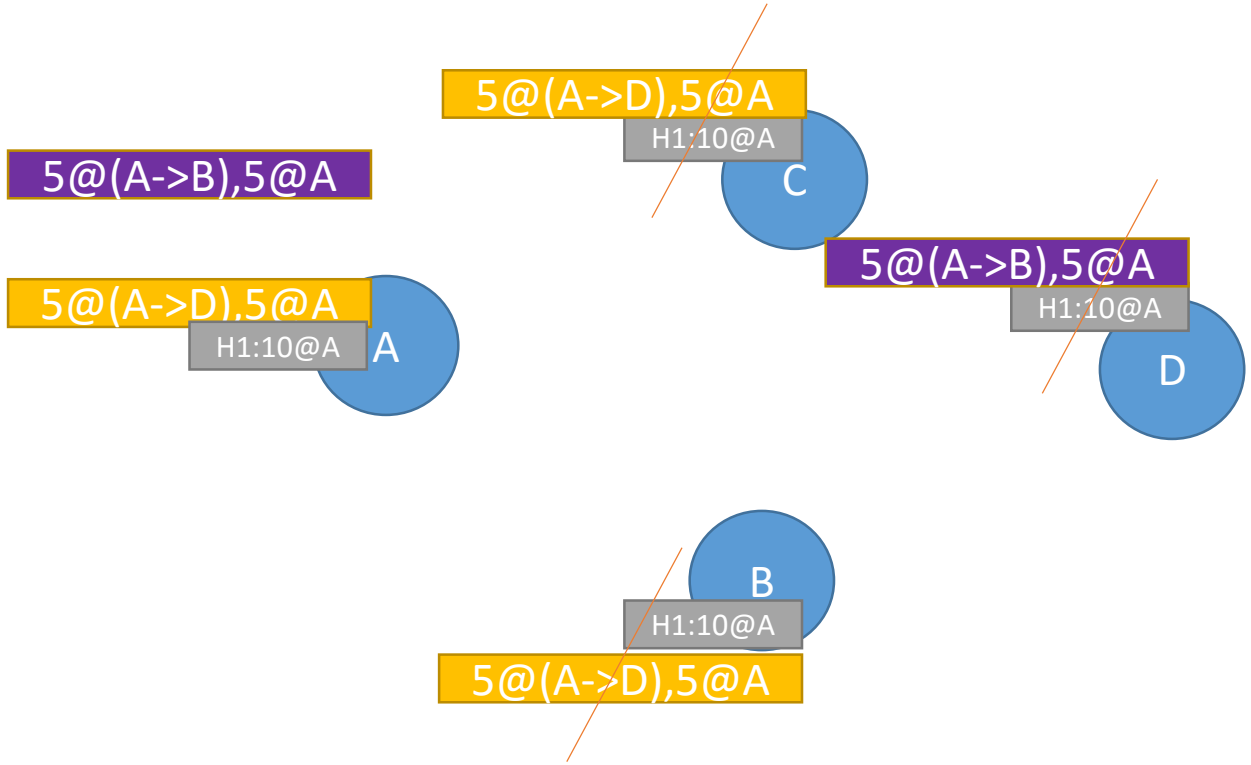
Double spend



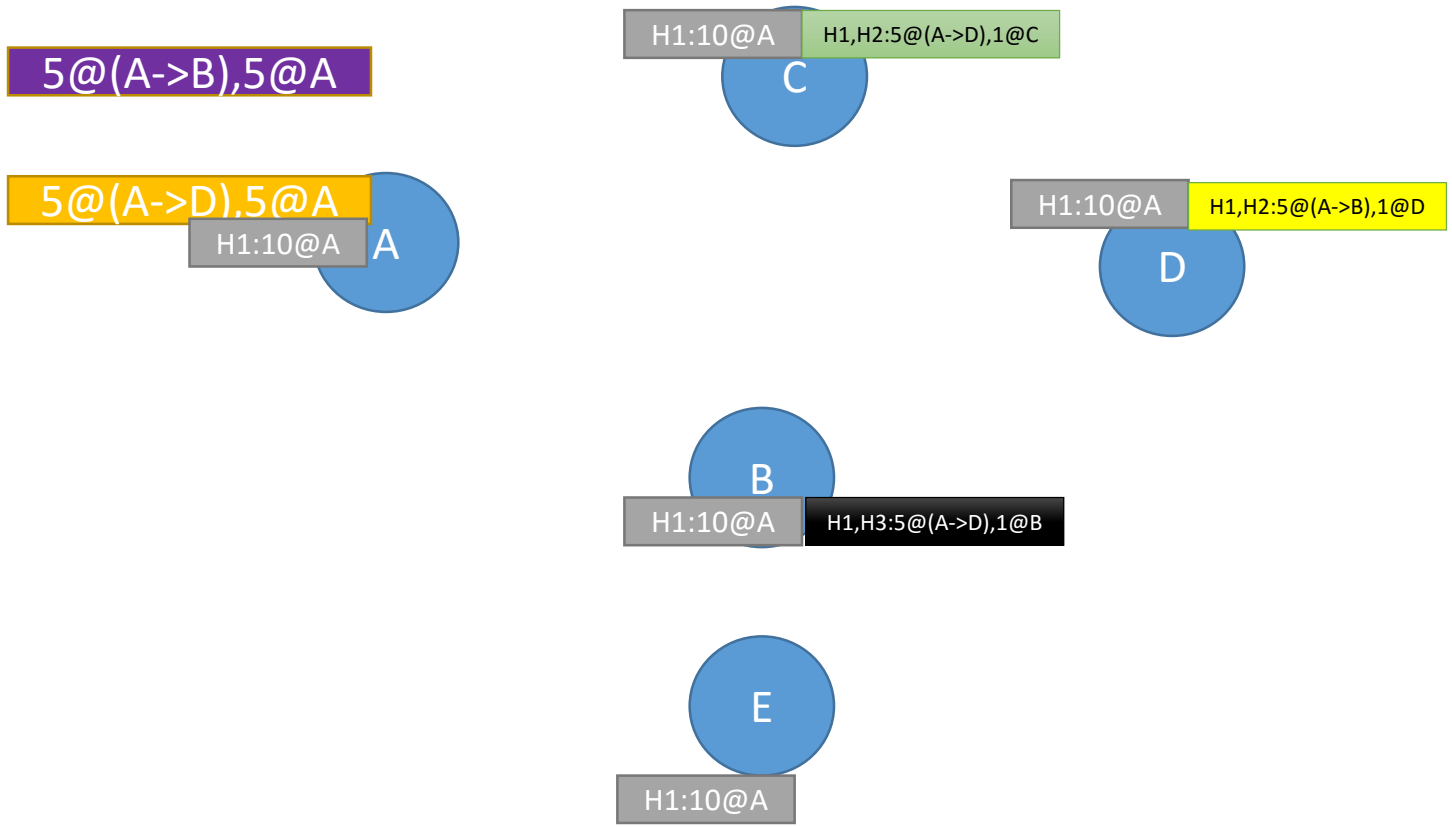
Double spend



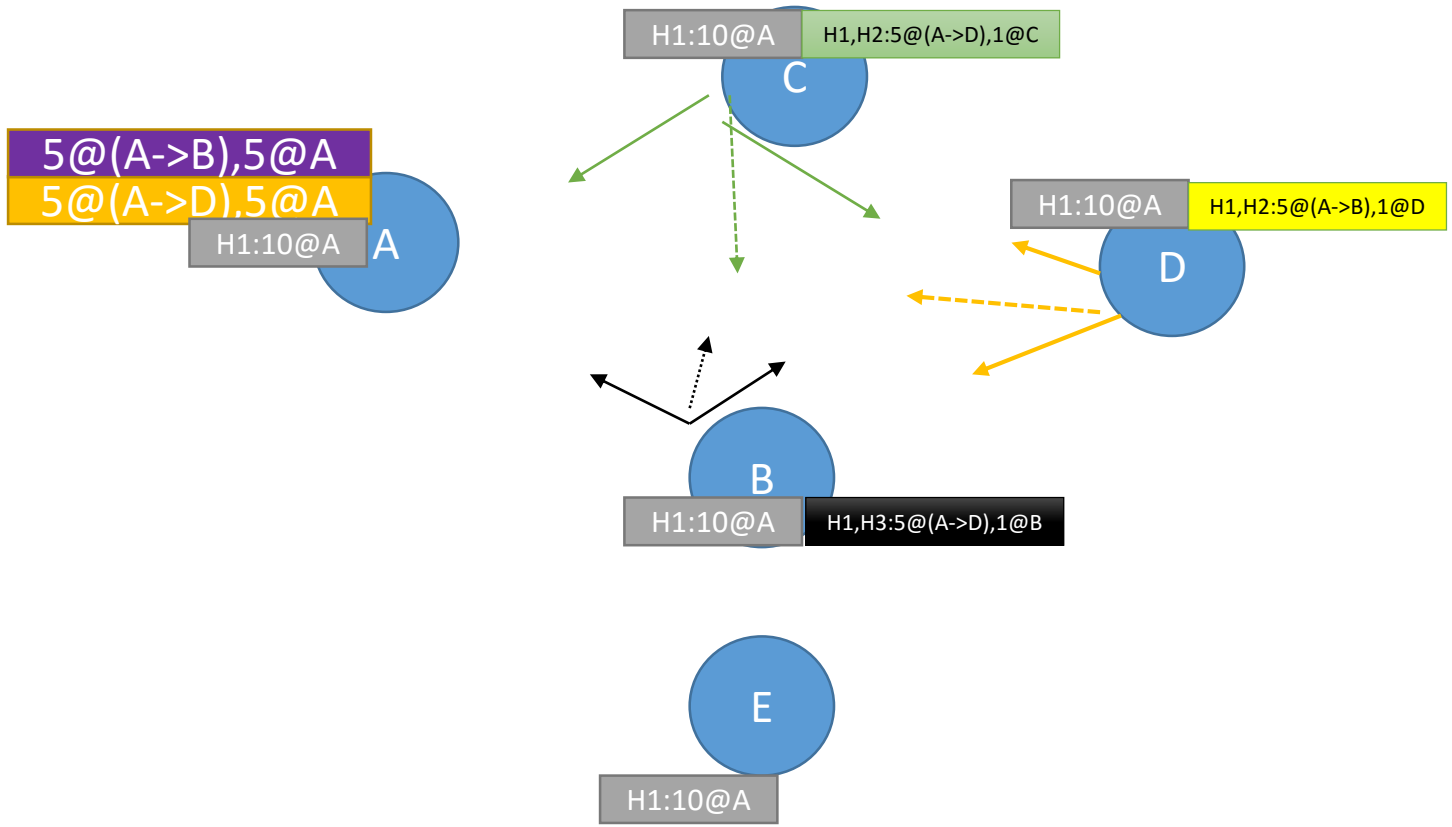
Double spend



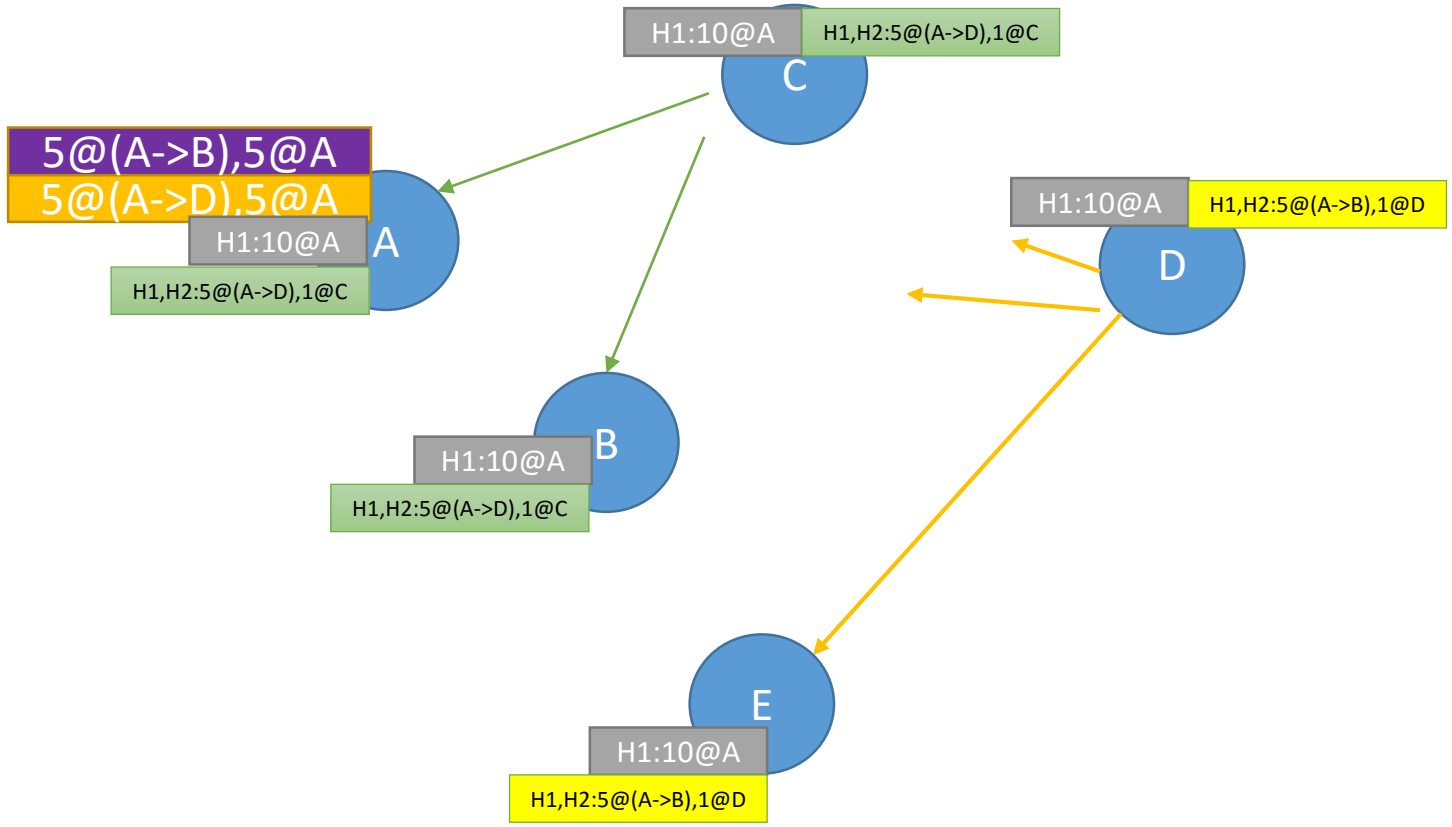
Double spend



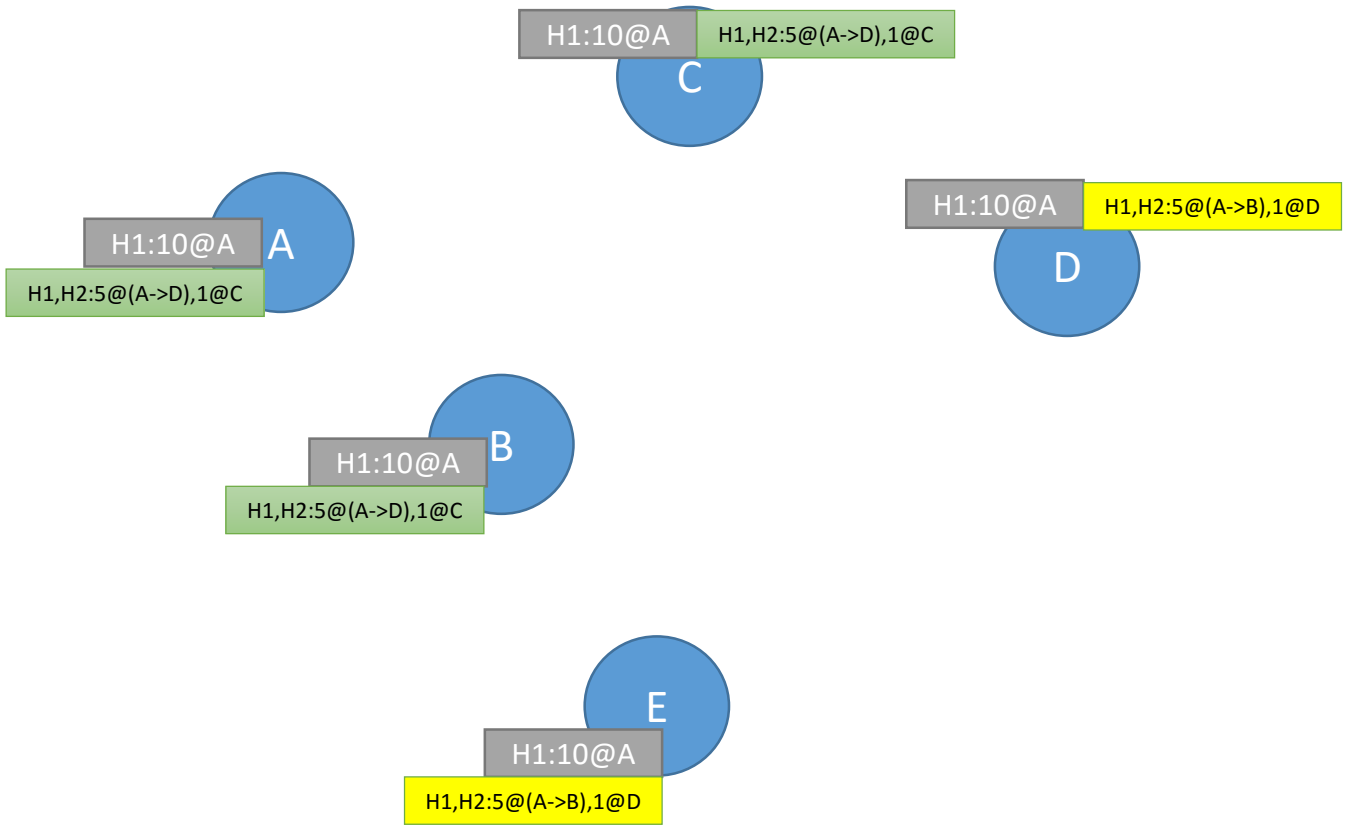
Double spend



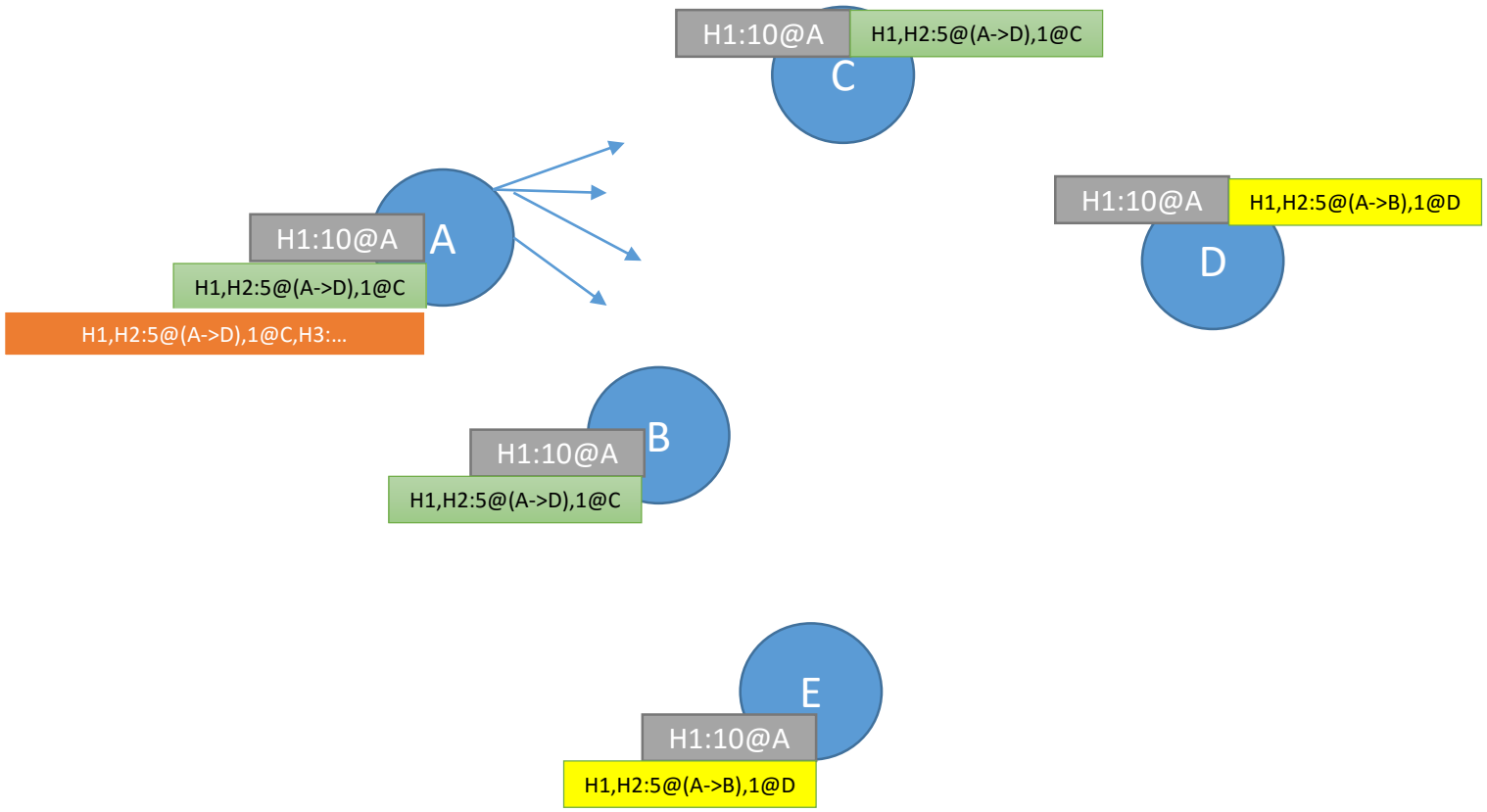
Double spend



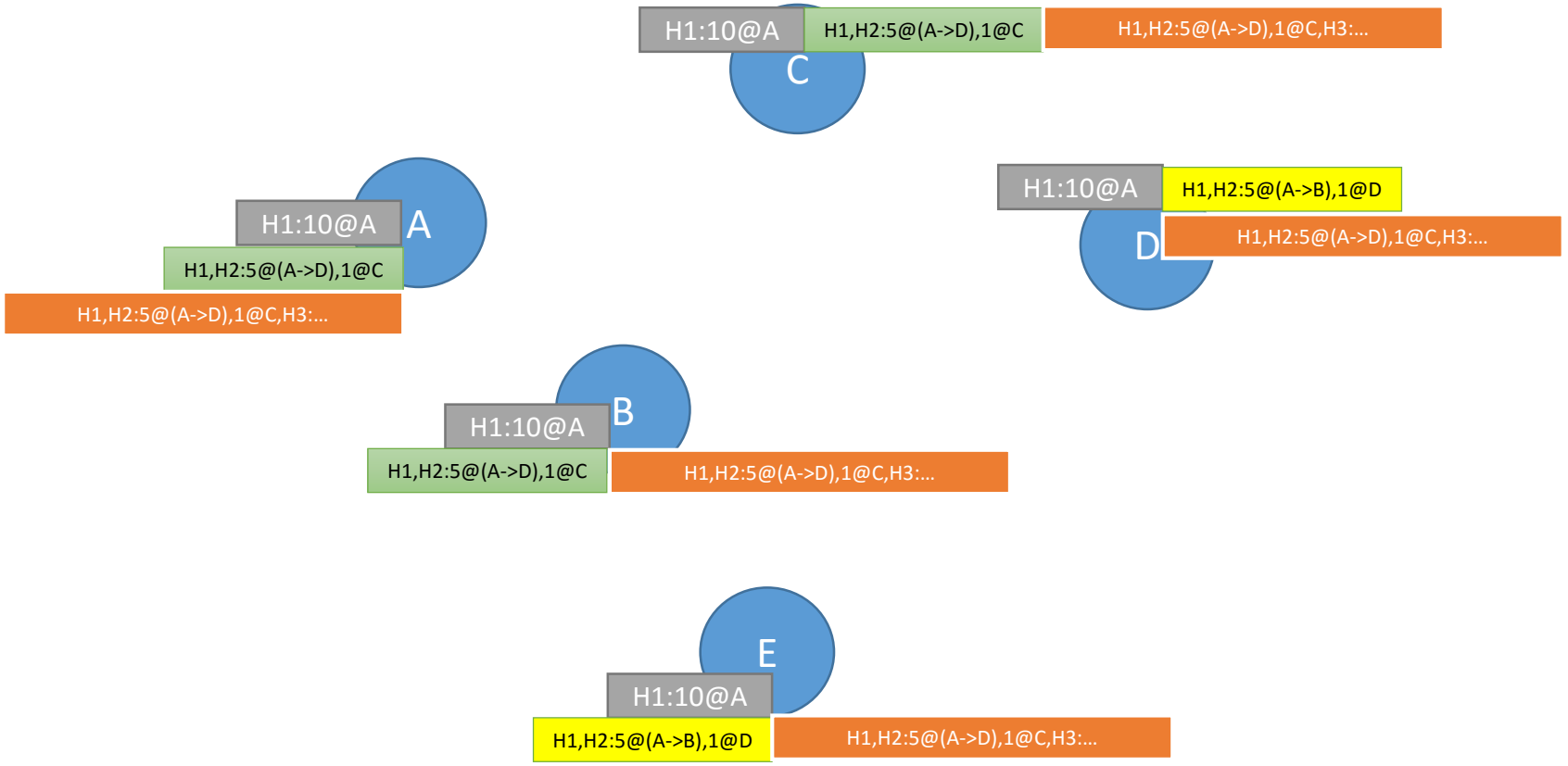
Double spend



Double spend

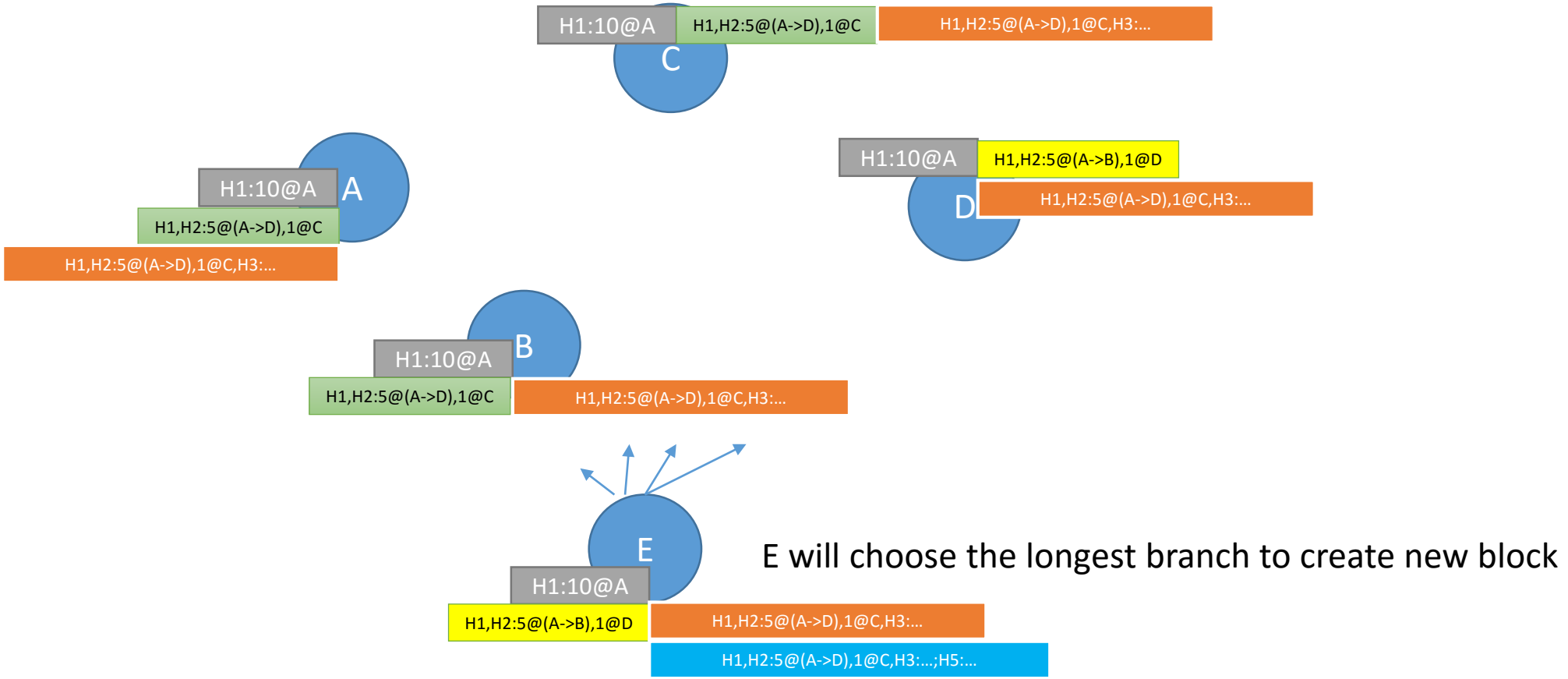


Double spend



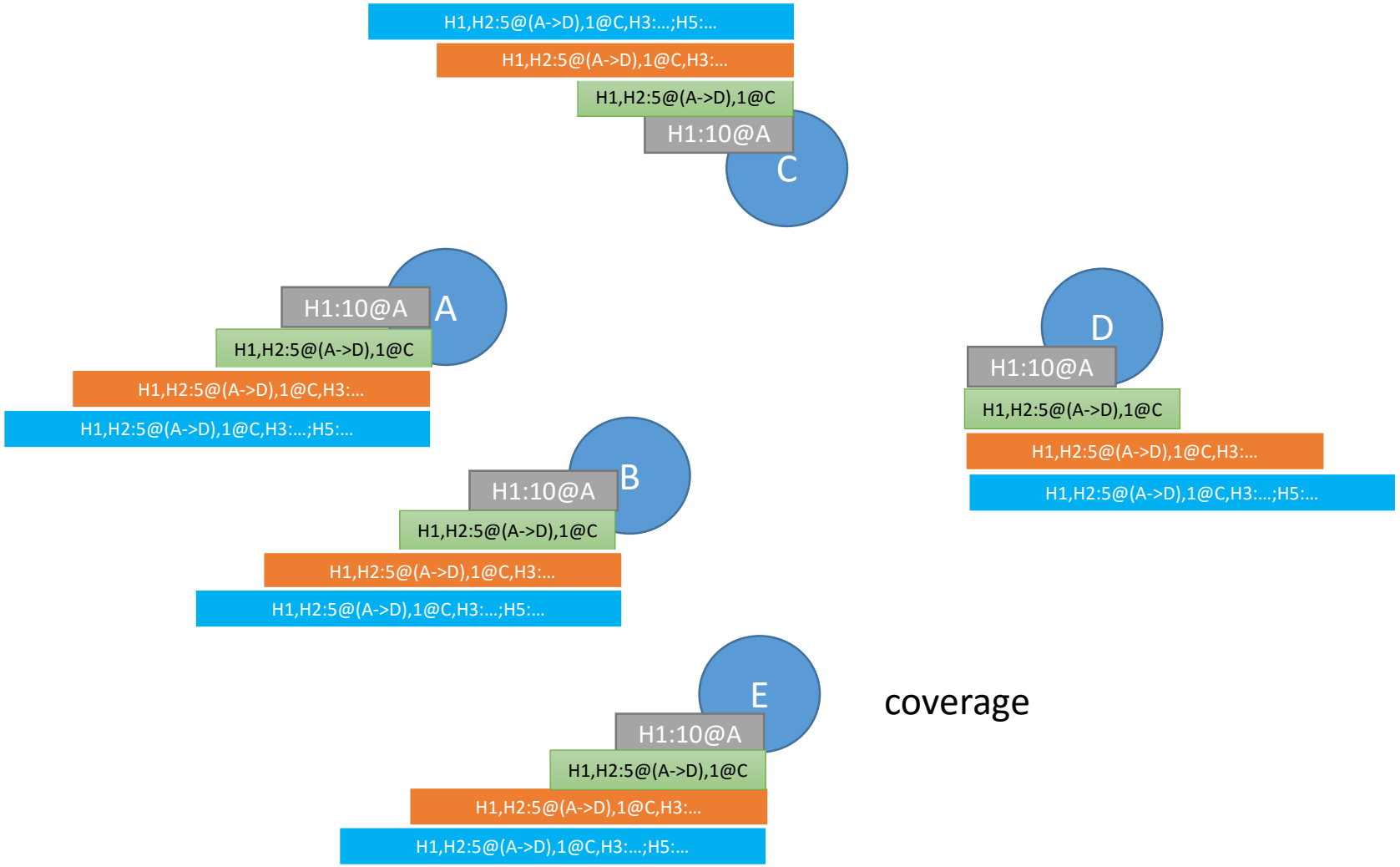
Now, E and D have branches

Double spend



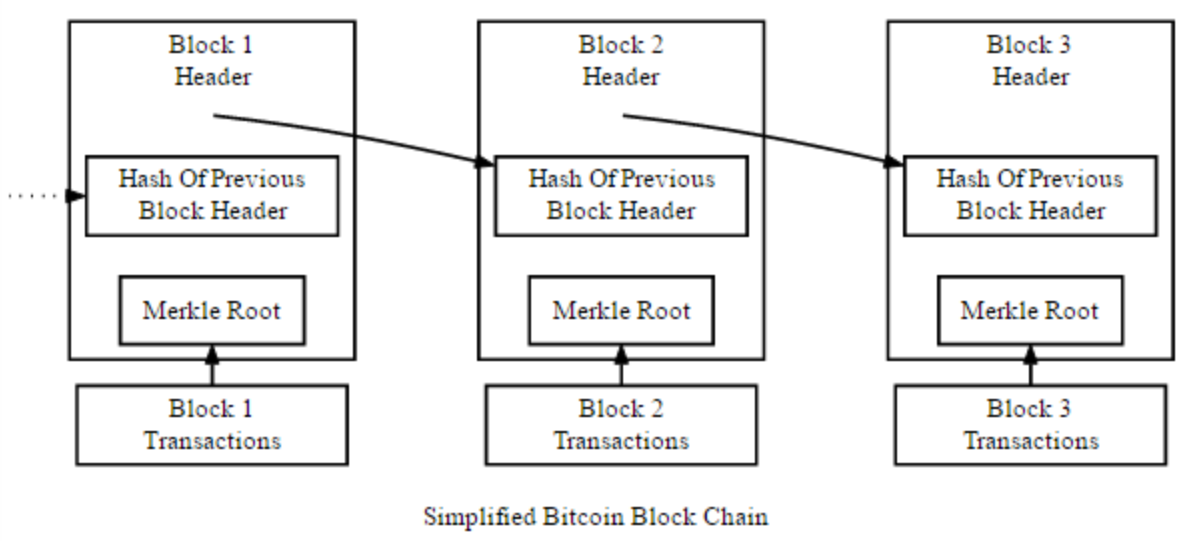
E will choose the longest branch to create new block

Double spend



coverage

The Real Block Chain



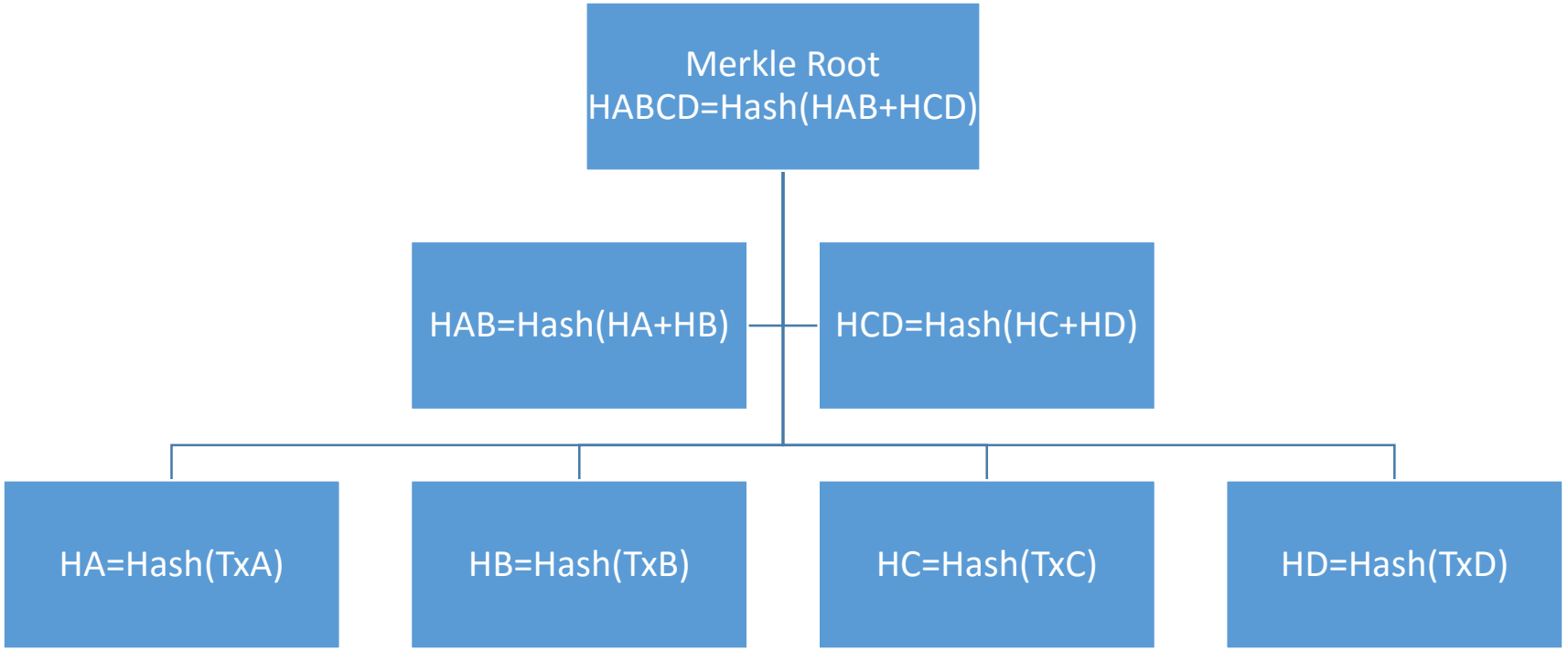
Simplified Bitcoin Block Chain

Proof Of Work(POW)

Version	02000000
Previous block hash	1312af178c253f84028d480a6adc1e25e81caa44c749ec81976192e2ec934c64
Merkle root	ae37343a357a8297591625e7134cbea22f5928be8ca2a32aa475cf05fd4266b7
timestamp	358b0553
Bits(difficulty value)	535f0119
nonce	48750833
Transaction count	63
coninbase transaction	
trasaction	
…	

Create a new block

Proof Of Work(POW)



Proof Of Work(POW)

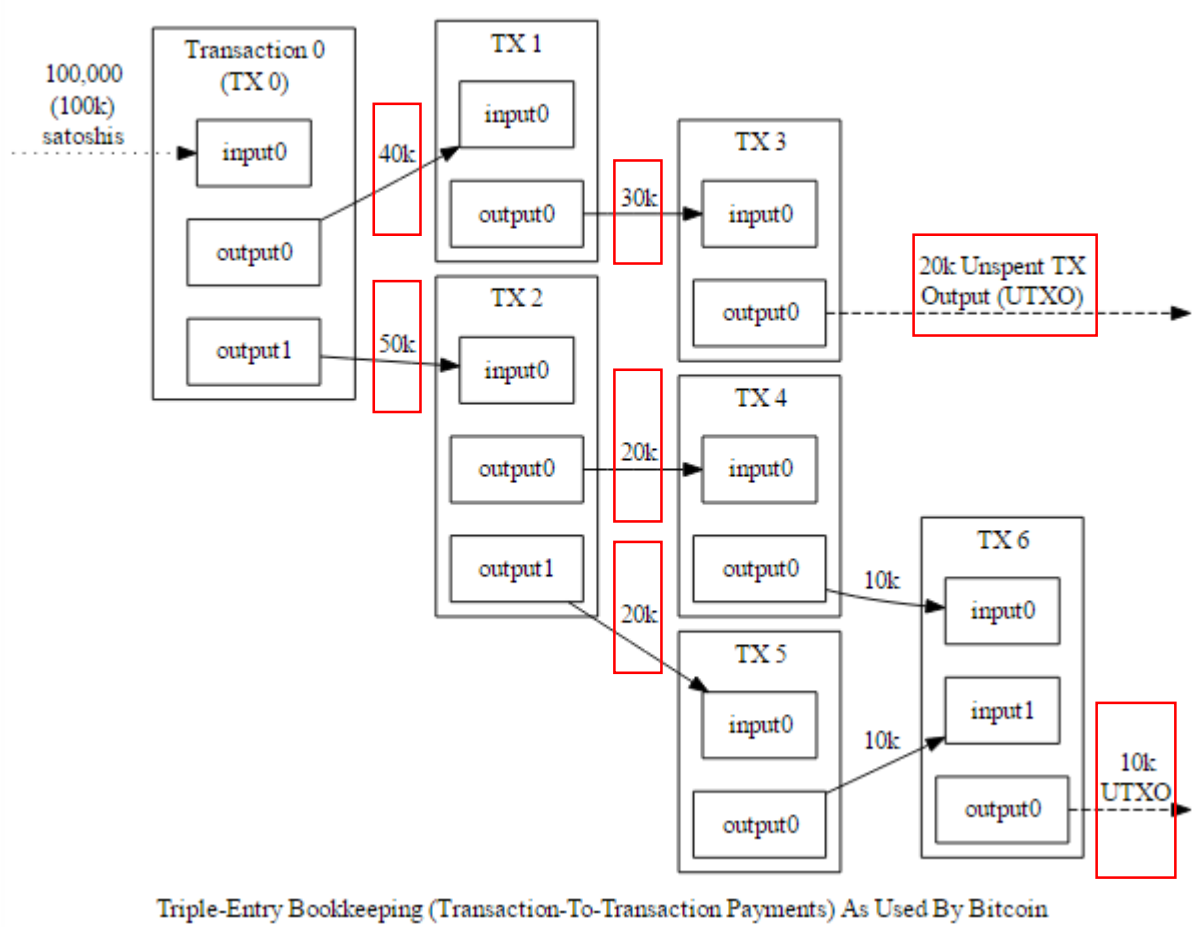
- 1. Calculate Merkle Tree of Transactions, Fill in the Block Header.
- 2. Calculate the current Target:

  Target = 0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF/[difficulty value*(time of create latest 2016 block)/20160 seconds)

- Try to find a nonce so that:

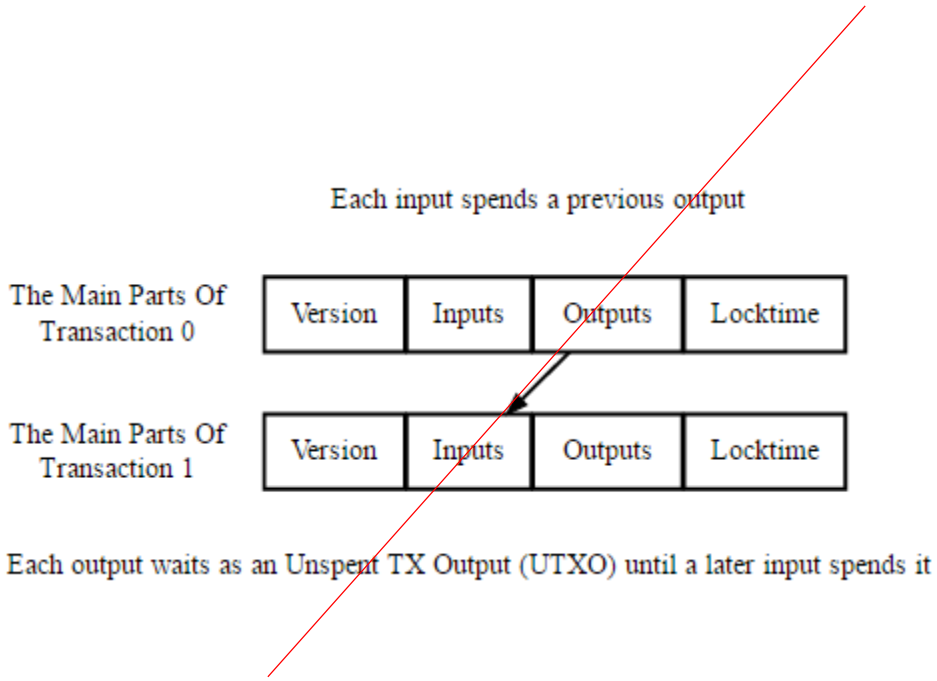
  SHA256(SHA256(Block_Header))<Target

The Real Transaction



Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

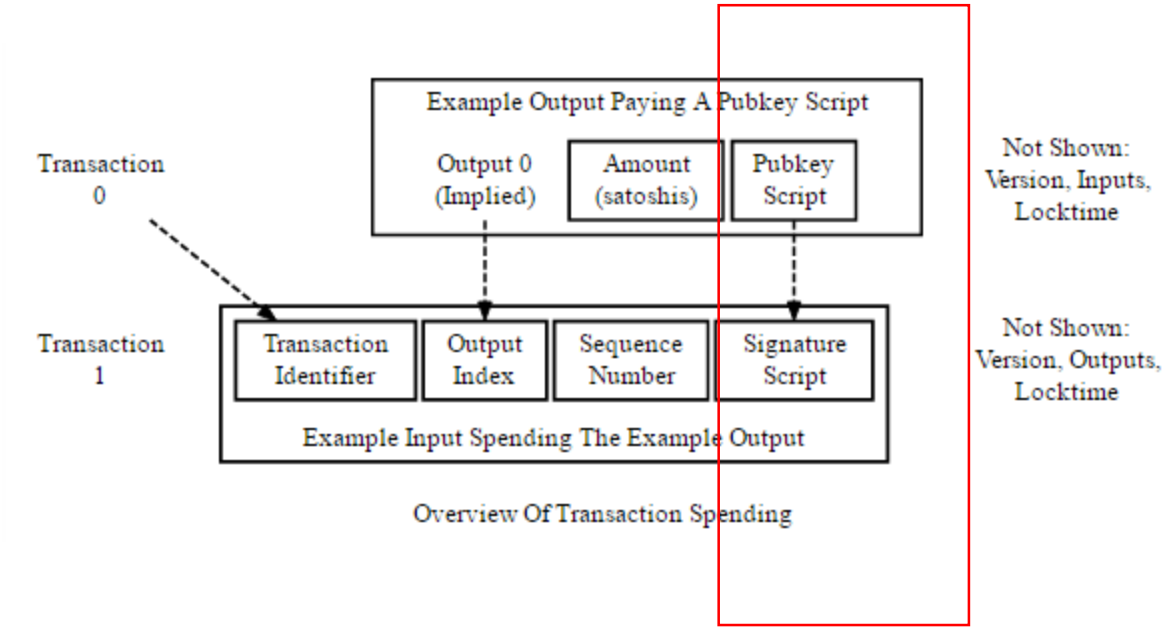
The Real Transaction



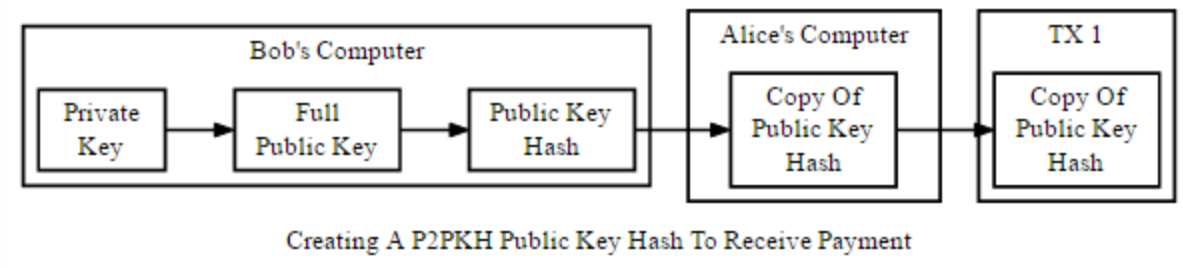
P2PKH

- P2PKH is the most common form of pubkey script used to send a transaction to one or multiple Bitcoin addresses.

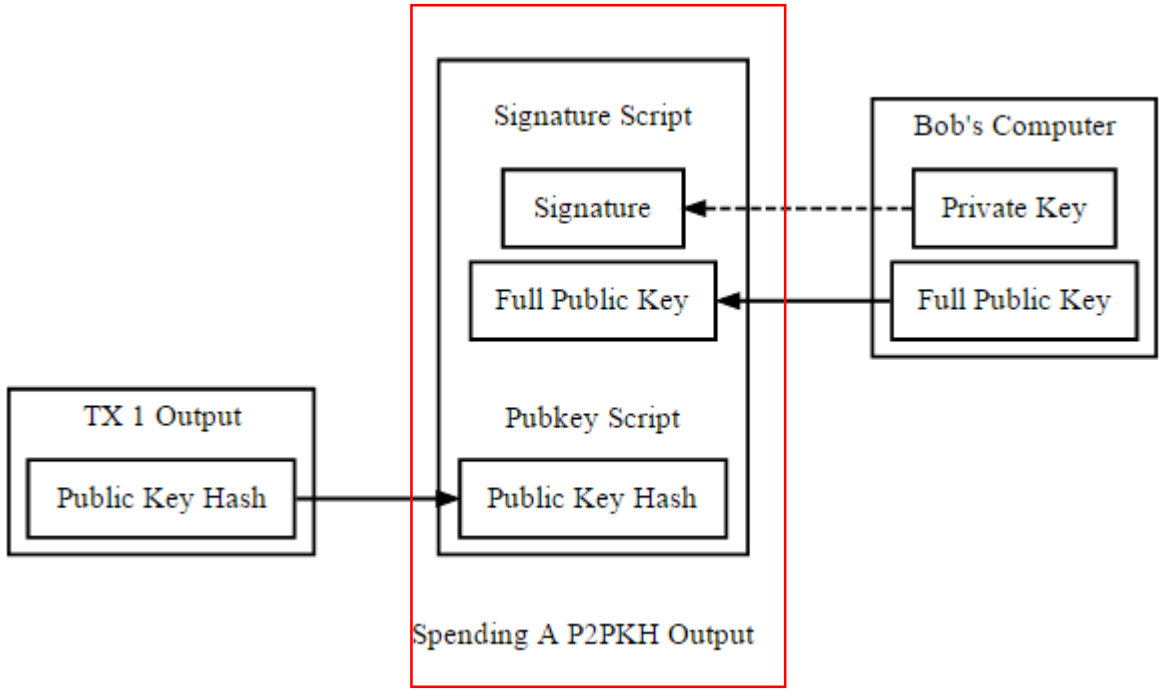
P2PKH Validation Scripts



P2PKH

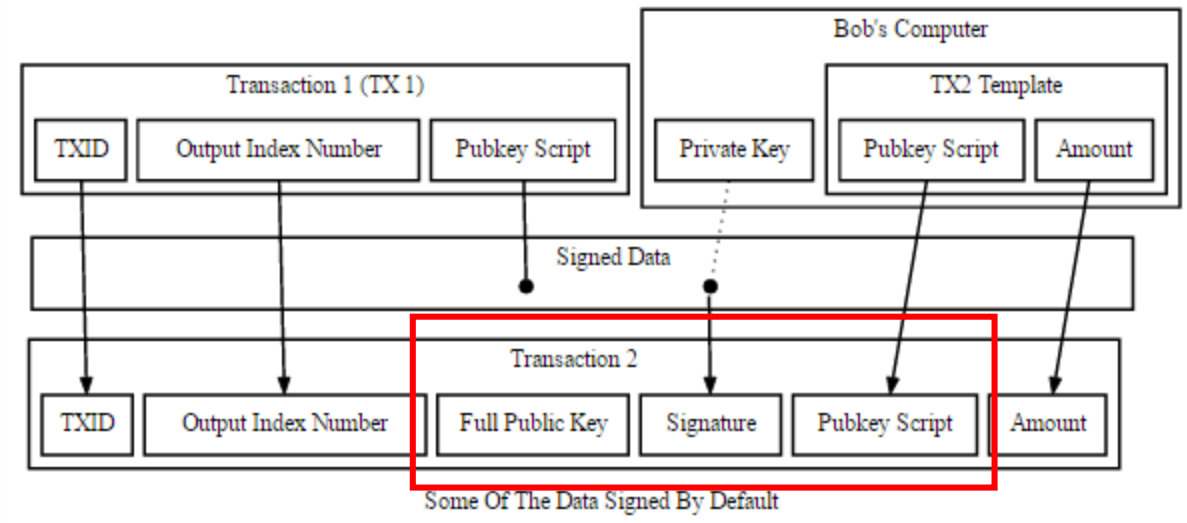


Creating A P2PKH Public Key Hash To Receive Payment



Spending A P2PKH Output

P2PKH


Bob's Computer
Transaction 1 (TX 1)
TX2 Template
TXID
Output Index Number
Pubkey Script
Private Key
Pubkey Script
Amount
Signed Data
Transaction 2
TXID
Output Index Number
Full Public Key
Signature
Pubkey Script
Amount
Some Of The Data Signed By Default

P2PKH Output

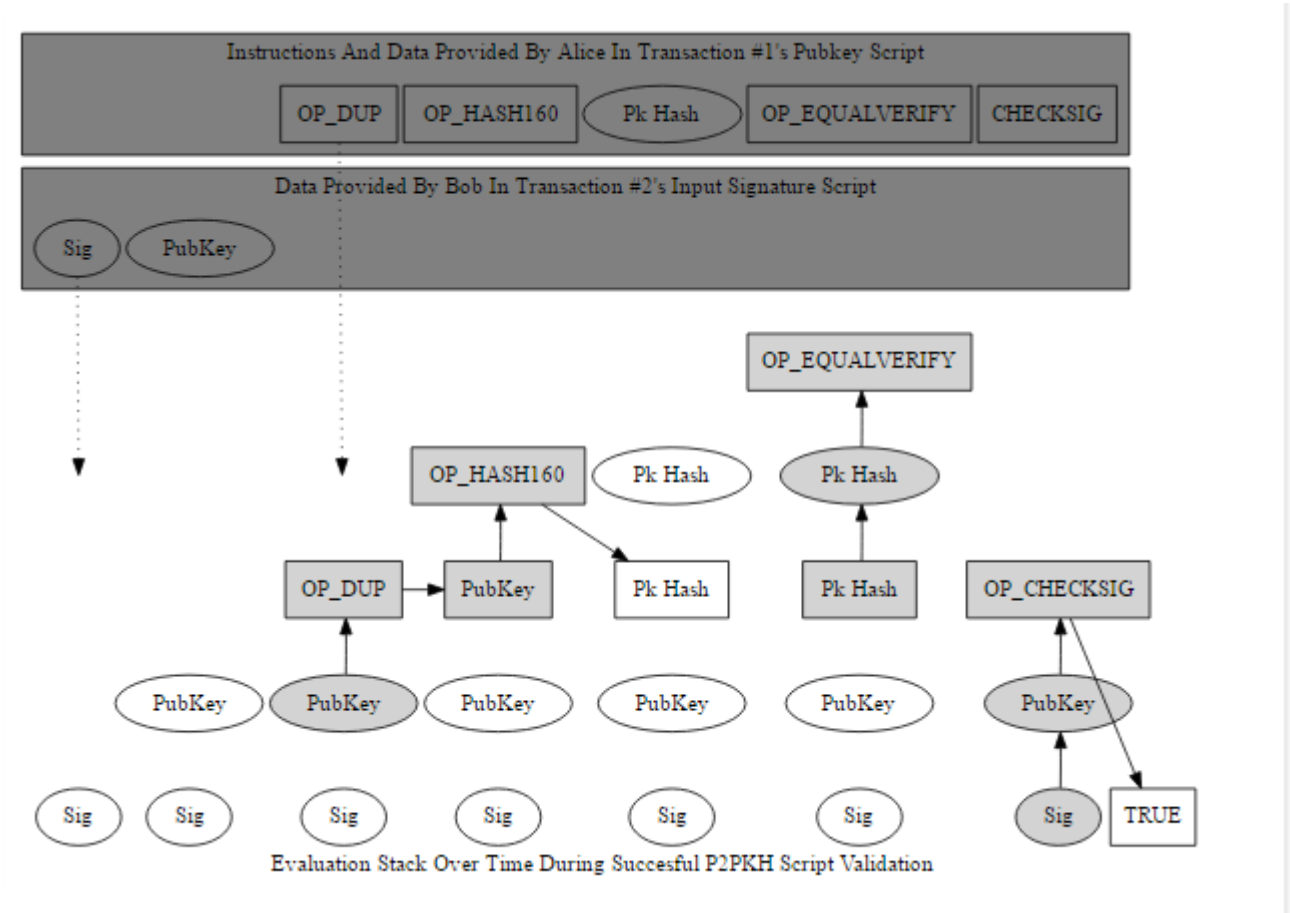
- PublicKey Script :

```
OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

- Signature Script:

```
<Sig> <PubKey> OP_DUP OP_HASH160 <PubkeyHash> OP_EQUALVERIFY OP_CHECKSIG
```

P2PKH validation



P2PKH validation

1. The signature (from Bob's signature script) is added (pushed) to an empty stack. Because it's just data, nothing is done except adding it to the stack. The public key (also from the signature script) is pushed on top of the signature.
2. From Alice's pubkey script, the **OP_DUP** operation is executed. **OP_DUP** pushes onto the stack a copy of the data currently at the top of it—in this case creating a copy of the public key Bob provided.
3. The operation executed next, **OP_HASH160**, pushes onto the stack a hash of the data currently on top of it—in this case, Bob's public key. This creates a hash of Bob's public key.
4. Alice's pubkey script then pushes the pubkey hash that Bob gave her for the first transaction. At this point, there should be two copies of Bob's pubkey hash at the top of the stack.
5. Now it gets interesting: Alice's pubkey script executes **OP_EQUALVERIFY**. **OP_EQUALVERIFY** is equivalent to executing **OP_EQUAL** followed by **OP_VERIFY** (not shown).
6. **OP_EQUAL** (not shown) checks the two values at the top of the stack; in this case, it checks whether the pubkey hash generated from the full public key Bob provided equals the pubkey hash Alice provided when she created transaction #1. **OP_EQUAL** pops (removes from the top of the stack) the two values it compared, and replaces them with the result of that comparison: zero (false) or one (true).
7. **OP_VERIFY** (not shown) checks the value at the top of the stack. If the value is false it immediately terminates evaluation and the transaction validation fails. Otherwise it pops the true value off the stack.
8. Finally, Alice's pubkey script executes **OP_CHECKSIG**, which checks the signature Bob provided against the now-authenticated public key he also provided. If the signature matches the public key and was generated using all of the data required to be signed, **OP_CHECKSIG** pushes the value true onto the top of the stack.

P2SH

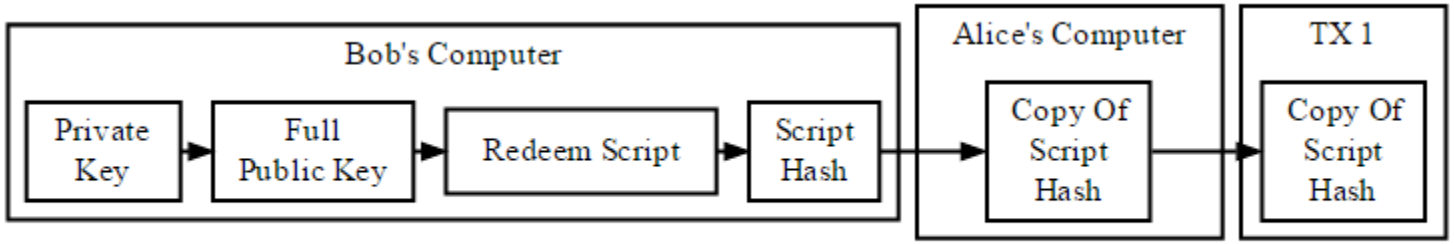
- Pubkey scripts are created by spenders who have little interest what that script does. Receivers do care about the script conditions and, if they want, they can ask spenders to use a particular pubkey script. Unfortunately, custom pubkey scripts are less convenient than short Bitcoin addresses and there was no standard way to communicate them between programs prior to widespread implementation of the BIP70 Payment Protocol discussed later.

- To solve these problems, pay-to-script-hash (P2SH) transactions were created in 2012 to let a spender create a pubkey script containing a hash of a second script, the redeem script.

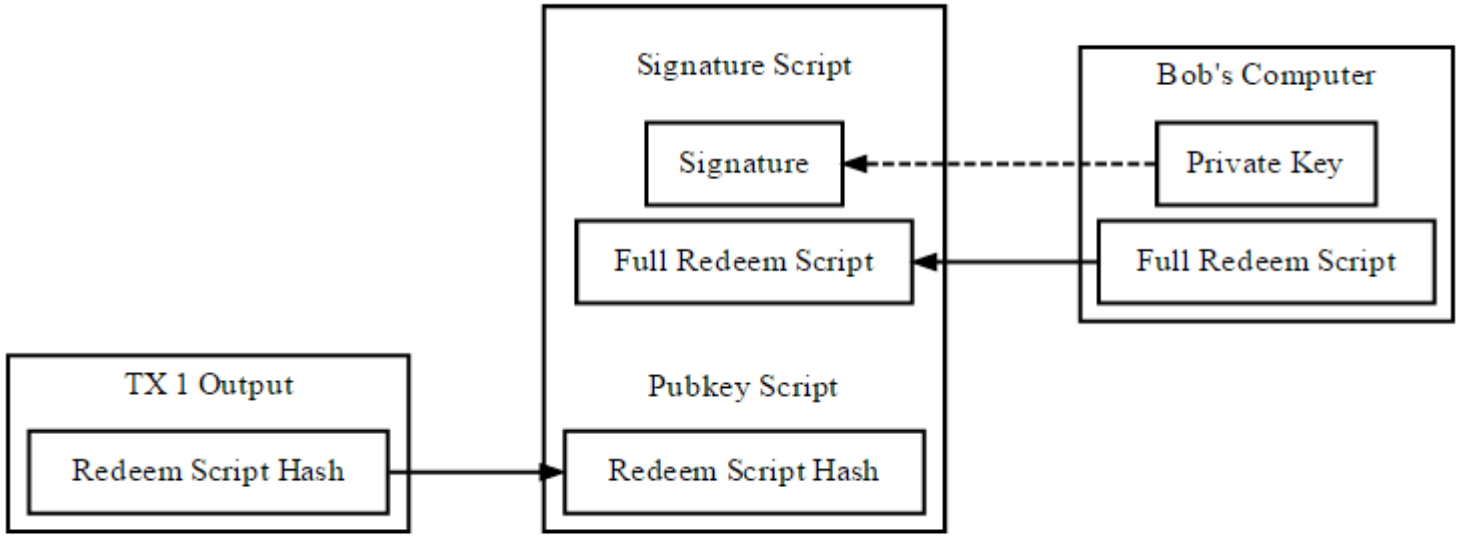
P2SH

- Bob creates a redeem script with whatever script he wants, hashes the redeem script, and provides the redeem script hash to Alice. Alice creates a P2SH-style output containing Bob's redeem script hash.
- When Bob wants to spend the output, he provides his signature along with the full (serialized) redeem script in the signature script. The peer-to-peer network ensures the full redeem script hashes to the same value as the script hash Alice put in her output; it then processes the redeem script exactly as it would if it were the primary pubkey script, letting Bob spend the output if the redeem script does not return false.

P2SH



Creating A P2SH Redeem Script Hash To Receive Payment



Spending A P2SH Output

P2SH Scripts

Pubkey script: OP_HASH160 <Hash160(redeemScript)> OP_EQUAL Signature script: <sig> [sig] [sig...] <redeemScript>

MultiSig

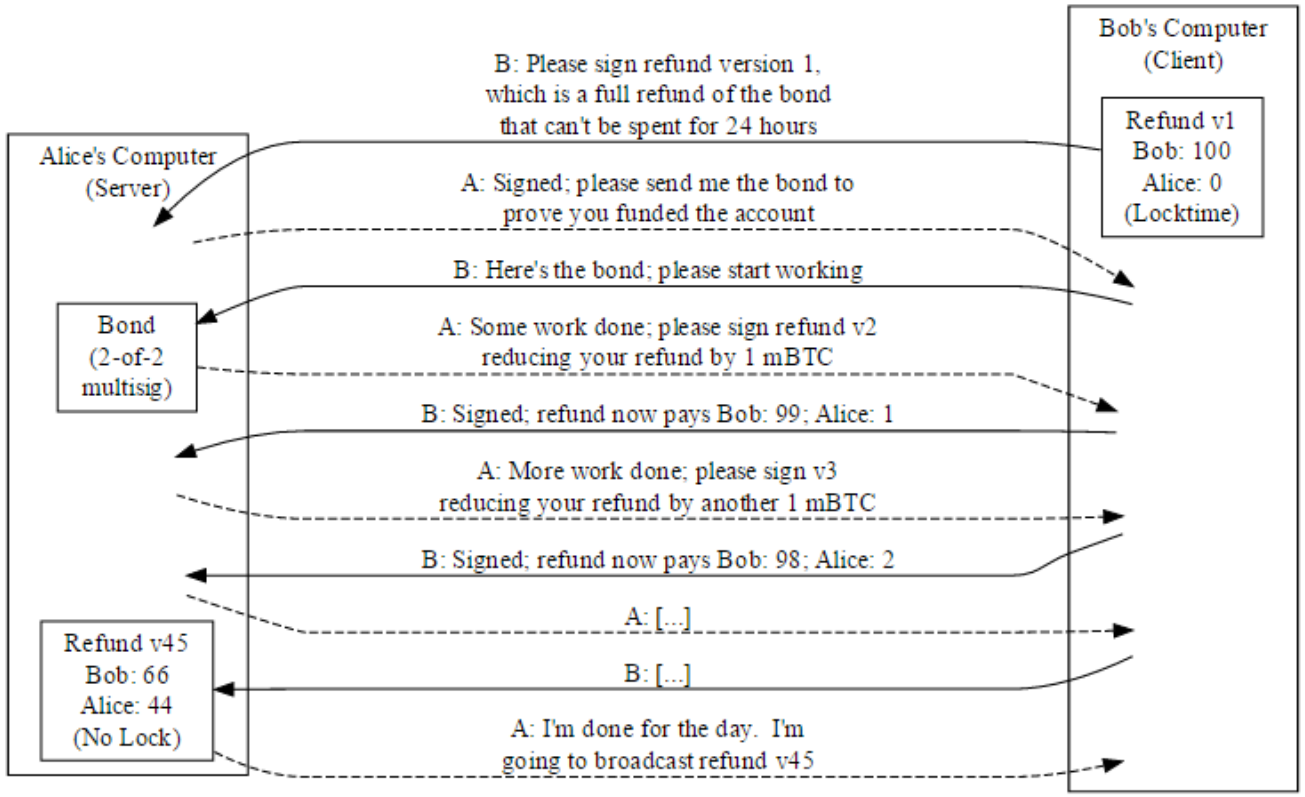
- P2PKH:

Pubkey script: <m> <A pubkey> [B pubkey] [C pubkey...] <n> OP_CHECKMULTISIG Signature script: OP_0 <A sig> [B sig] [C sig...]

- P2SH:

Pubkey script: OP_HASH160 <Hash160(redeemScript)> OP_EQUAL Redeem script: <OP_2> <A pubkey> <B pubkey> <C pubkey> <OP_3> OP_CHECKMULTISIG Signature script: OP_0 <A sig> <C sig> <redeemScript>

A Smart-Contract case



Alice broadcasts the bond to the Bitcoin network immediately.
She broadcasts the final version of the refund when she finishes work
or before the locktime. If she fails to broadcast before refund v1's time lock
expires, Bob can broadcast refund v1 to get a full refund.

Bitcoin Micropayment Channels (As Implemented In Bitcoinj)

References

1. https://en.bitcoin.it/wiki/Network
2. https://en.bitcoin.it/wiki/Protocol_specification
3. https://en.bitcoin.it/wiki/Protocol_rules
4. https://bitcoin.org/en/developer-documentation
5. https://arxiv.org/pdf/1405.7418v1.pdf
6. http://ethfans.org/posts/r3-corda-announcement