



CS1699: Blockchain Technology and Cryptocurrency

8. Mechanics of Bitcoin

Bill Laboon

Bitcoin Consensus

- ❖ Append-only ledger (via blockchain)
- ❖ Decentralized consensus (via consensus mechanisms)
- ❖ Validation of transactions (via miners & proof-of-work)

Account-Based Ledger: Fungible Coins - Signatories And Account Balances

BLOCK	ACTION	SIGNATORY
1	CREATE 25 COINS; TRANSFER ALL TO ALICE	MINER
2	TRANSFER 10 COINS TO BOB	ALICE
3	TRANSFER 5 COINS TO CAROL	ALICE
4	TRANSFER 2 COINS TO DAN	BOB

Account-Based Ledger

- ❖ Accounts (could still be delineated by public keys) would have a certain number of bitcoins
- ❖ A transaction would e.g. take five coins from Alice and give five coins to Bob

Account-Based Ledger: Efficiency Issues

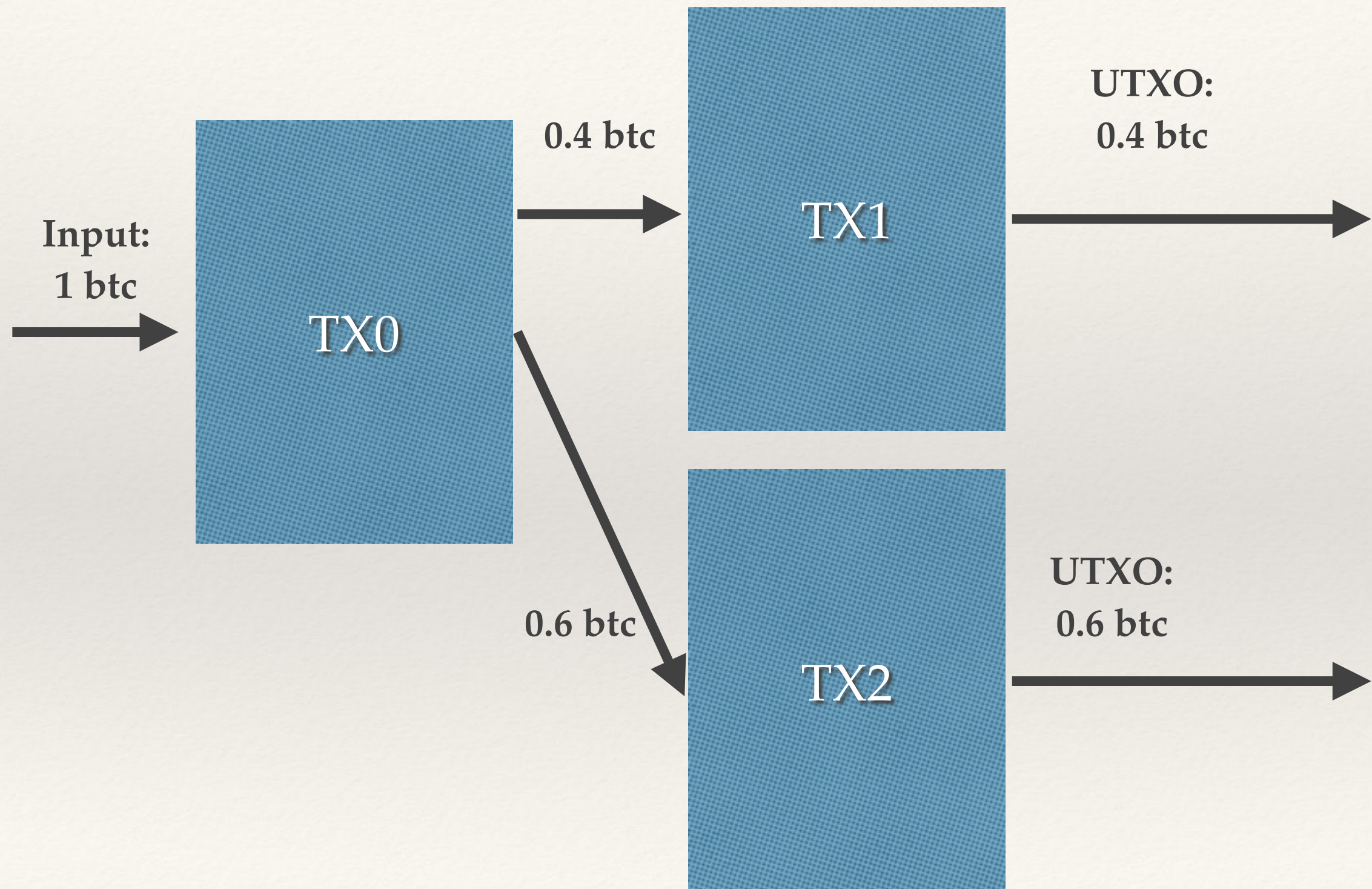
Q: Is the transaction in red valid? A: May have to track back to genesis block!

BLOCK	ACTION	SIGNATORY
1	CREATE 25 COINS; TRANSFER ALL TO ALICE	MINER
2	TRANSFER 10 COINS TO BOB	ALICE
3	TRANSFER 5 COINS TO CAROL	ALICE
4	TRANSFER 2 COINS TO DAN	BOB

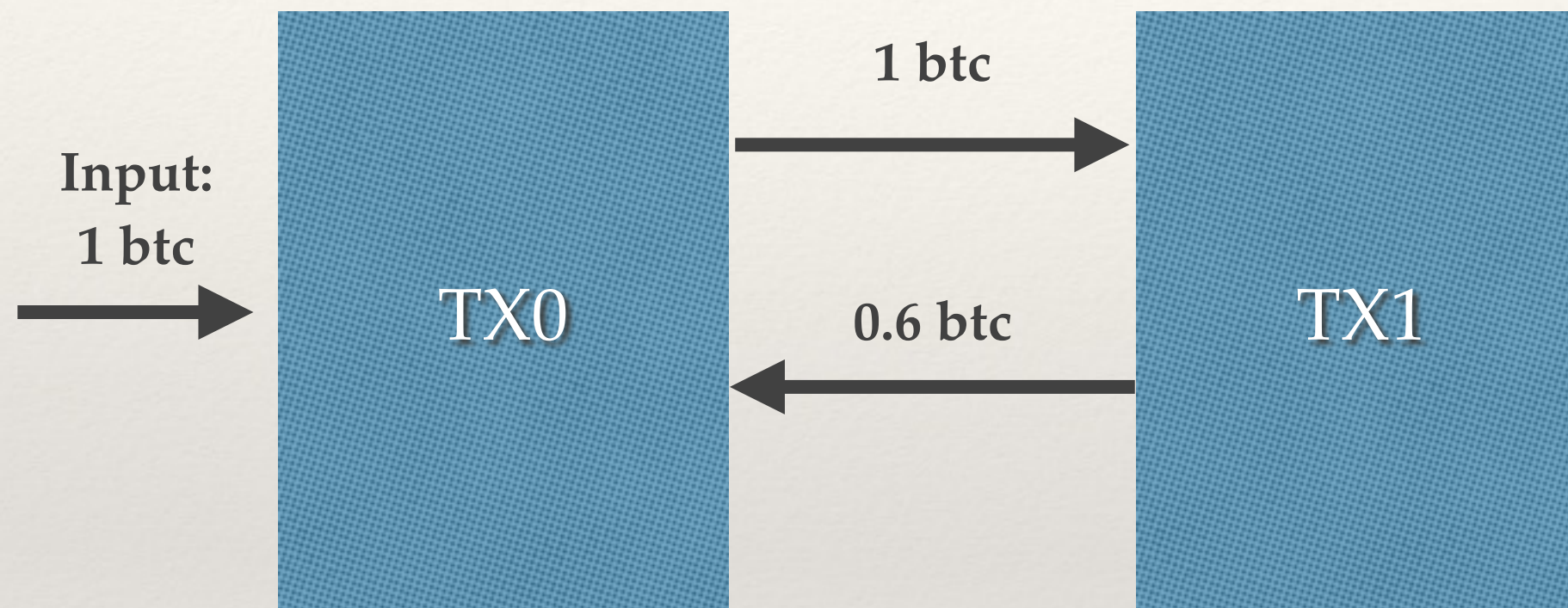
Transaction-Based Model (Bitcoin)

- ❖ Every transaction has *valid* inputs and outputs
 - ❖ Properly signed
 - ❖ Inputs must be consumed entirely
 - ❖ Total value of outputs \leq inputs
- ❖ “Accounts” are just a collection of *UTXOs* (*unspent transaction outputs*)

Transactions Behind the Scenes

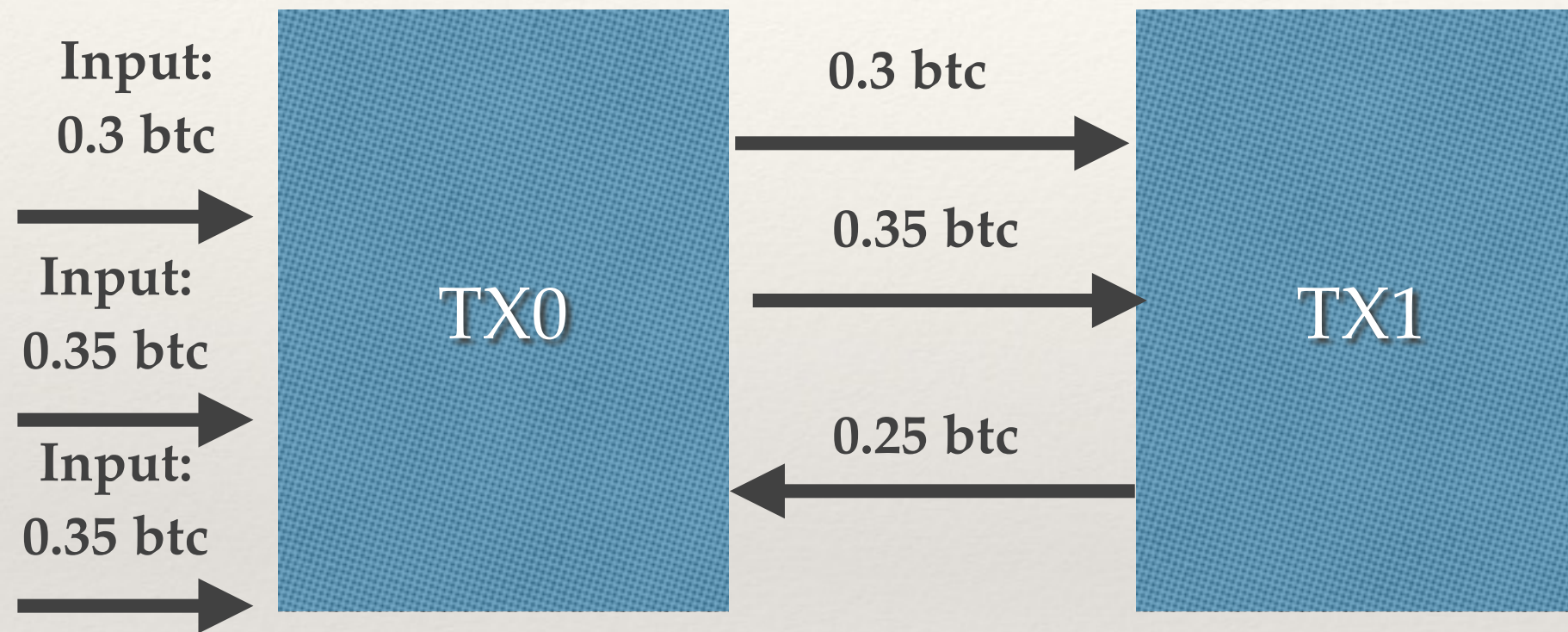


Splitting TXs - Sending 0.4 btc



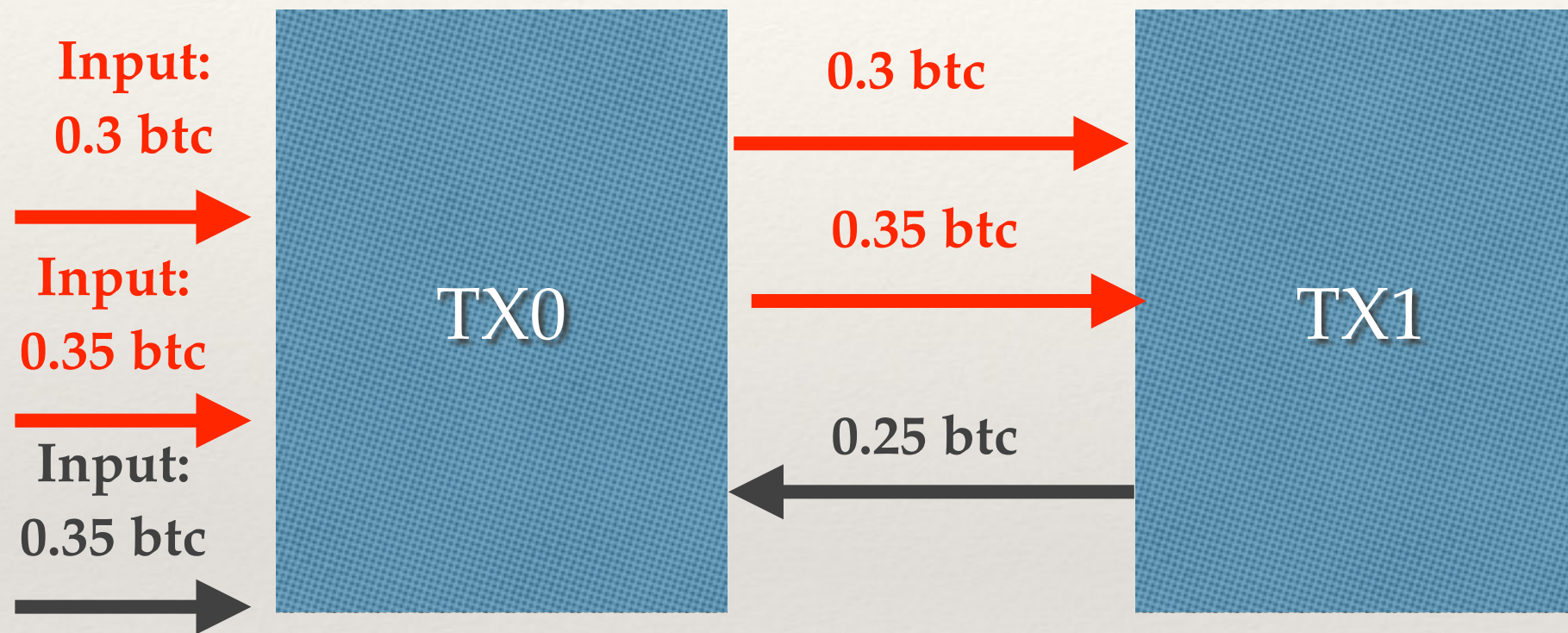
Multiple TX Input/Output

Sending 0.4 btc



Spent vs Unspent Tx's

Sending 0.4 btc



Generator of TX0 now has two UTXOs: 0.35 btc and 0.25 btc

Most bitcoin wallets will just show you $0.35 + 0.25 = 0.6$ btc


$$1 \text{ btc} - 0.4 \text{ btc} = 0.6 \text{ btc}$$

Transaction-Based Ledger

BLOCK	INPUTS/OUTPUTS	SIGNATORY
1	<i>INPUTS:</i> \emptyset <i>OUTPUTS:</i> 25 -> ALICE	N/A
2	<i>INPUTS:</i> 1[0] <i>OUTPUTS:</i> 10 -> BOB; 15 -> ALICE	ALICE
3	<i>INPUTS:</i> 2[1] <i>OUTPUTS:</i> 5 -> CAROL; 10 -> ALICE	ALICE
4	<i>INPUTS:</i> 2[0] <i>OUTPUTS:</i> 2 -> DAN; 8 -> BOB	BOB

Verification Only Back Until Coin Creation

BLOCK	INPUTS/OUTPUTS	SIGNATORY
1	<i>INPUTS: \emptyset</i> <i>OUTPUTS: 25 -> ALICE</i>	N/A
2	<i>INPUTS: 1[0]</i> <i>OUTPUTS: 10 -> BOB; 15 -> ALICE</i>	ALICE
3	<i>INPUTS: 2[1]</i> <i>OUTPUTS: 5 -> CAROL; 10 -> ALICE</i>	ALICE
4	<i>INPUTS: 2[0]</i> <i>OUTPUTS: 2 -> DAN; 8 -> BOB</i>	BOB



Outputs Must Be Entirely Consumed

BLOCK	INPUTS/OUTPUTS	SIGNATORY
1	<i>INPUTS:</i> \emptyset <i>OUTPUTS:</i> 25 -> ALICE	N/A
2	<i>INPUTS:</i> 1[0] <i>OUTPUTS:</i> 10 -> BOB; 15 -> ALICE	ALICE
3	<i>INPUTS:</i> 2[1] <i>OUTPUTS:</i> 5 -> CAROL; 10 -> ALICE	ALICE
4	<i>INPUTS:</i> 2[0] <i>OUTPUTS:</i> 2 -> DAN; 8 -> BOB	BOB

The diagram illustrates the flow of outputs from block 1 to subsequent blocks. A vertical red line originates from the output '25 -> ALICE' of block 1. This line branches into three horizontal red arrows pointing left to the inputs of blocks 2, 3, and 4. The arrows are labeled 'Change addresses' in red text. Specifically, the top arrow points to input '1[0]' of block 2, the middle arrow points to input '2[1]' of block 3, and the bottom arrow points to input '2[0]' of block 4.

Joint Payment - 10 Bitcoin To Carol From Bob & Alice

BLOCK	INPUTS/OUTPUTS	SIGNATORY
1	<i>INPUTS:</i> \emptyset <i>OUTPUTS:</i> 25 -> ALICE	N/A
2	<i>INPUTS:</i> 1[0] <i>OUTPUTS:</i> 10 -> BOB; 15 -> ALICE	ALICE
3	<i>INPUTS:</i> 2[0], 2[1] <i>OUTPUTS:</i> 10 -> CAROL; 10 -> ALICE; 5 -> BOB	ALICE, BOB

A Look at a Raw Block

- ❖ That was conceptual - let's look at an actual block (in JSON - key: value format)
- ❖ Blocks generally indicated by hash or height
- ❖ Latest Block (as of Sunday, 3:38 PM EST)
000000000000000000000000f462731f00e4e600469893990c896dba86fa5fb0c1990
- ❖ <https://blockchain.info/rawblock/000000000000000000000000f462731f00e4e600469893990c896dba86fa5fb0c1990>

Anatomy of a Block

- ❖ NOTE: The textbook's description of blocks is slightly outdated, in at least two ways:
 - ❖ Block metadata is slightly off due to SegWit
 - ❖ Does not take into account BIP-34 changes
- ❖ Information in these slides is correct as of the slides' creation date! Bitcoin block structure is liable to change even more in the future.

Anatomy of a Block

- ❖ **Metadata**

- ❖ *Information about the block (size, block number, Merkle root, nonce, etc.)*

- ❖ **Transactions**

- ❖ *Array of all transactions in this block*
 - ❖ *Number of transaction must be ≥ 1 ! (coinbase transaction always present)*

Block Metadata

```
"hash": "000000000000000000000000f462731f00e4e600469893990c896dba86fa5fb0c1990",  
"ver": 536870912,  
"prev_block": "000000000000000000000005f3c0b802ac8b85488d4f88d7a60a20ee4ea3984c1394",  
"mrkl_root": "a65164b813a723570c7d65fad61987036a566d341a0efce4872836a72cb0b934",  
"time": 1537728446,  
"bits": 388454943,  
"fee": 5310126,  
"nonce": 2278125992,  
"n_tx": 1236,  
"size": 498546,  
"block_index": 1724700,  
"main_chain": true,  
"height": 542728,  
"received_time": 1537728446,  
"relayed_by": "0.0.0.0",
```

Compare to Genesis Block

Missing received_time and relayed_by attributes , added later...

```
"hash": "000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f",  
"ver": 1,  
"prev_block": "0000000000000000000000000000000000000000000000000000000000000000",  
"mrkl_root": "4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b",  
"time": 1231006505,  
"bits": 486604799,  
"fee": 0,  
"nonce": 2083236893,  
"n_tx": 1,  
"size": 285,  
"block_index": 14849,  
"main_chain": true,  
"height": 0,
```

<https://blockchain.info/rawblock/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f>

Transactions

- ❖ **Metadata**

- ❖ *Information about the transaction (hash, size, number of inputs, number of outputs, lock time)*

- ❖ **Inputs**

- ❖ *Previous output (all inputs are previous outputs, except coinbase) to be consumed, signature*

- ❖ **Outputs**

- ❖ *Value of output, a Script script (note: no recipient address directly specified here!)*

Transaction Metadata

```
"weight":900,  
"time":1537728260,  
"tx_index":376130660,  
"vin_sz":1,  
"hash":"1887e38e63ecfcb1490b4adb21b6410b4  
e60454fe09502fb2aa2503b69f3e3c0",  
"vout_sz":2,  
"relayed_by":"0.0.0.0",  
"lock_time":542727,  
"ver":2  
"size":225,
```

Transaction Inputs

Where is the signature?

```
"inputs": [{
  "sequence": 4294967294,
  "witness": "",
  "prev_out": {
    "spent": true,
    "tx_index": 376128954,
    "type": 0,
    "addr": "17pGR12CGLzhiMzUr18RZzVoEoaUbT8LTa",
    "value": 3108223,
    "n": 1,
    "script": "76a9144ac12bf46ceb296f3c98d19186088b12cb21162688ac"
  },
  "script": "473044022063bfe6f169acd12f4329e6e63b09c1c79856ee07fb0d1
29f9eeb10eccb1ec350022037c94f02090b27047bc882360b4a832b0eaef21bd4
ce239f5332615c5889fec0012103e78877a7225046ff0f9de99d91d0632f1411f
0ad8e7d4e70622bfcc39b616573"
}],
```

Transaction Outputs

```
"out": [{
  "spent": true,
  "tx_index": 376130660,
  "type": 0,
  "addr": "1CXpe1tQQKpeUc89kUqQC2FuiZc5u5AnTa",
  "value": 1017919,
  "n": 0,
  "script": "76a9147e7da0fb7c6edce44f0e88b8ea6d0d933d4bc75888ac"
}, {
  "spent": true,
  "tx_index": 376130660,
  "type": 0,
  "addr": "15JLH7qkwKNJMJfCYrye96hNKqx8SjPkys",
  "value": 2044878,
  "n": 1,
  "script": "76a9142f27aab77d5e5ebc1853076c29b318c032d9375788ac"
}]
```

Programmable Money

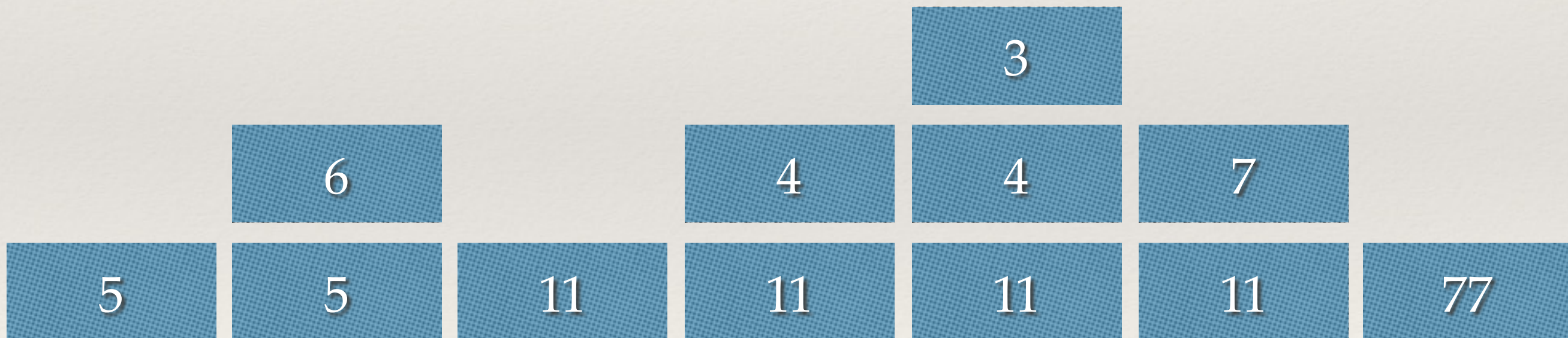
- ❖ Bitcoin is programmable money!
- ❖ A transaction specifies a script to execute
- ❖ The VAST majority of scripts just move previous transaction outputs to a new address, we can (theoretically) do anything that the Script language lets us
- ❖ Although other nodes may have something to say about that...

Script - The Bitcoin Scripting Language

- ❖ Inputs contain scripts (scriptSig attribute) and outputs contain scripts (scriptPubKey) - concatenated and executed
- ❖ Script
 - ❖ Stack-based language
 - ❖ Not Turing-complete (purposely limited)
 - ❖ Native support for complex cryptographic functions
 - ❖ Note: you may see a hex version of this in a blockchain explorer - think of it as the actual bytecode instead of helpful mnemonics

Stack-Based Programming

- ❖ Reverse Polish Notation Calculator
- ❖ $5\ 6\ +\ 4\ 3\ +\ *$



Why Not Turing-Complete?

- ❖ Would never be able to determine if program would end (Halting Problem)
- ❖ Could force arbitrary-length execution on validating nodes (denial-of-service attack)
- ❖ In practice, only a very few kinds of scripts are allowed by most validating nodes (whitelist)
- ❖ See Ethereum (especially concept of *gas*) for one way to have a Turing-complete language running on a blockchain

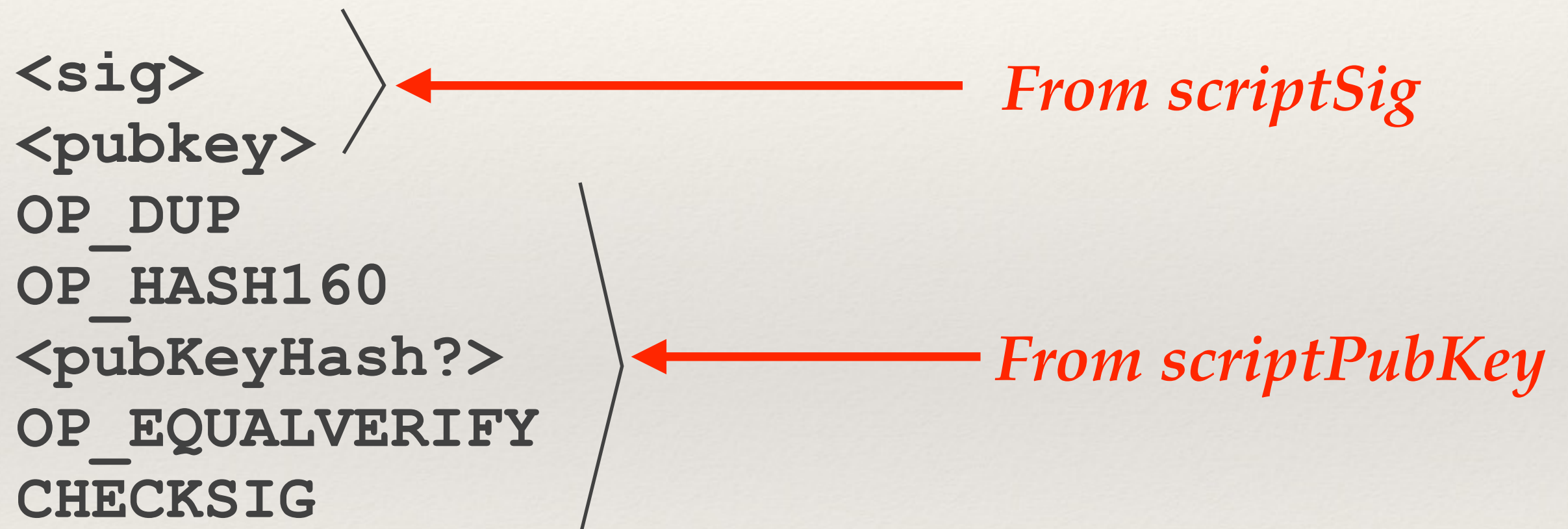
Common Script Commands

- ❖ ***VALUE*** - *Put value on top of stack*
- ❖ **OP_DUP** - *Duplicate top item on stack*
- ❖ **OP_HASH160** - *Hash value on top of stack with SHA-256, then hash again with RIPEMD-160 and put final value back on stack*
- ❖ **OP_EQUALVERIFY** - *Return true if top two values of stack are equal, return false and mark transaction if not*
- ❖ **OP_CHECKSIG** - *Check that the input signature is valid using the input public key for the hash of the current transaction*

Script Execution Walkthrough

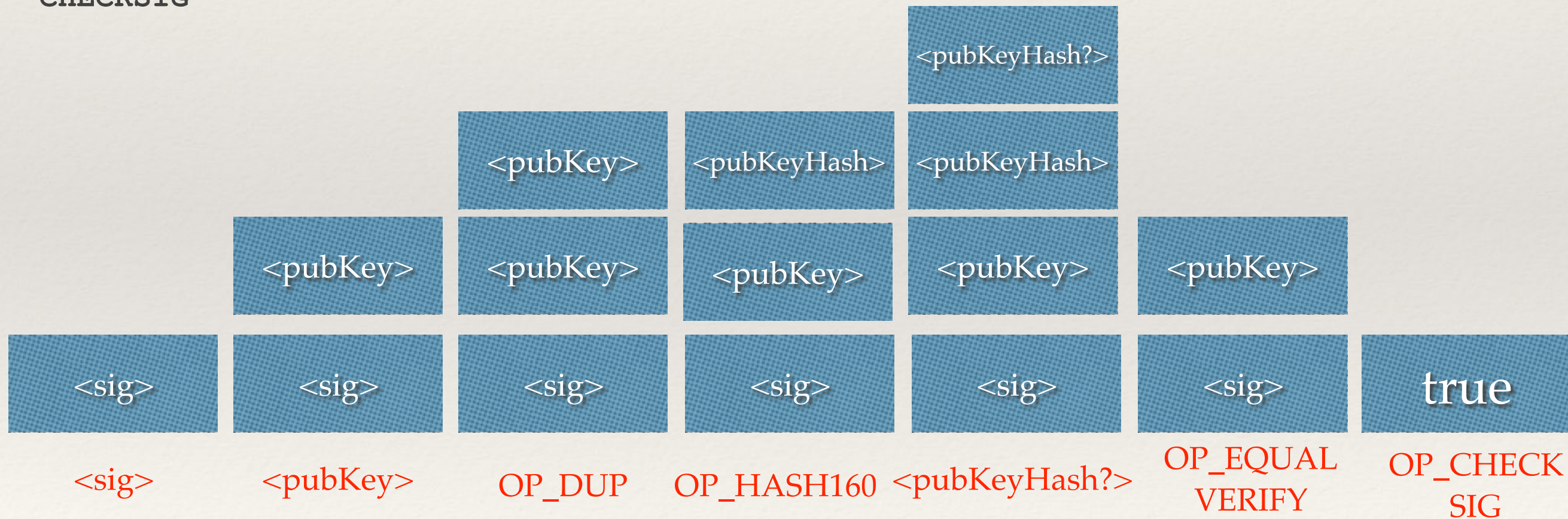
- ❖ Concatenate scriptSig and scriptPubKey
- ❖ Execute script

Script Execution Walkthrough



Basic Bitcoin Script

<sig>
<pubkey>
OP_DUP
OP_HASH160
<pubKeyHash?>
OP_EQUALVERIFY
CHECKSIG



Proof Of Burn

- ❖ Proof of burn establishes that a transaction(s) has been destroyed
- ❖ Why?
 - ❖ Destroy bitcoin to generate some other token
- ❖ How?
 - ❖ Add an OP_RETURN opcode to the scriptPubKey of the transaction - script will always return false

P2PKH vs P2SH

- ❖ P2PKH - *Pay to Public Key Hash* (remember addresses are actually hashes of hashes of a public key, so “pay to address”)
- ❖ P2SH - *Pay to Script Hash* (allow easy multisig or other more complex transactions - not part of original Bitcoin spec, added as BIP-16 in 2012)
- ❖ Additional transaction types:
 - ❖ P2PK - “Pay to Public Key” - generally not used since 0.3, not supported since 0.8
 - ❖ P2WPKH - “Pay To Witness Public Key Hash” - SegWit (Segregated Witness) transaction