COMP6451 - Cryptocurrency and Distributed Ledger Technologies

Lecture - Bitcoin Block and Transaction Structure

Ron van der Meyden

UNSW School of Computer Science and Engineering

Summary

Overall structure of components of the Bitcoin Blockchain:

- Bitcoin Denominations
- Blocks
- Transactions
- Bitcoin Script

Bitcoin Denominations

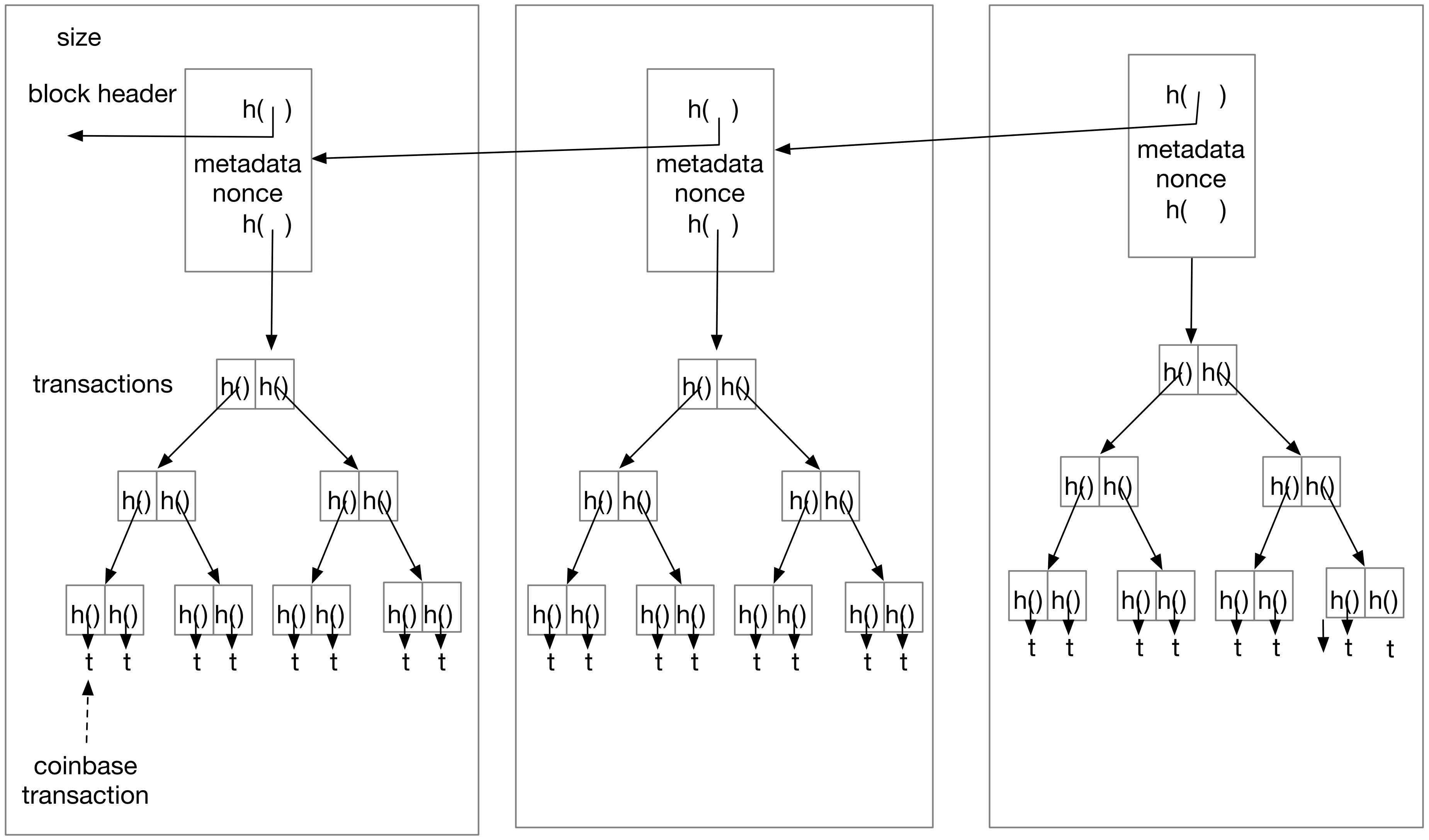
1 *Satoshi* = The smallest allowable unit of Bitcoin

1 Bitcoin = 100,000,000 Satoshis

Currency symbols used for Bitcoin: BTC, XBT (depends on the exchange/service)

Current exchange rate 1 BTC ≈ $A 5,000

Overall Structure of the Blockchain



Block Structure

A block consists of the following fields:

- Block size
- Block Header = {
    - Software Version number
    - Previous Block Hash
    - Hash of Root of the Merkle tree of this block's transactions
    - TimeStamp (approximate creation time)
    - Difficulty Target (for proof of work)
    - Nonce (proof of work puzzle solution) }
- Transaction Counter (no of transactions in this block)
- Transactions in this block (a list, the Merkle tree is computed from this with *h*=SHA256(SHA256(.)))

- A block is identified by the hash SHA256(SHA256(block header)) or
  (ambiguously) by its height in the blockchain. (Neither is included in the block.)

Bitcoin block explorers

http://blockexplorer.com

http://blockchain.info

http://insight.bitpay.com

Genesis Block

The blockchain starts with the *Genesis Block.*

This block is hard-coded into the Bitcoin client software, in file chainparams.cpp

It has hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

(note the leading zero's for proof of work puzzle solution)

Account-based Systems

In an *account-based* system (e.g. your current bank account) each user has an account.

An account holds a *balance.*

Transfers into or out of the account change the balance.

Example:

Alice: \$0, Bob: \$600, Carol: \$100

Bob transfers \$100 to Alice

Carol transfers \$100 to Alice

Alice: \$200, Bob: \$500, Carol: \$0

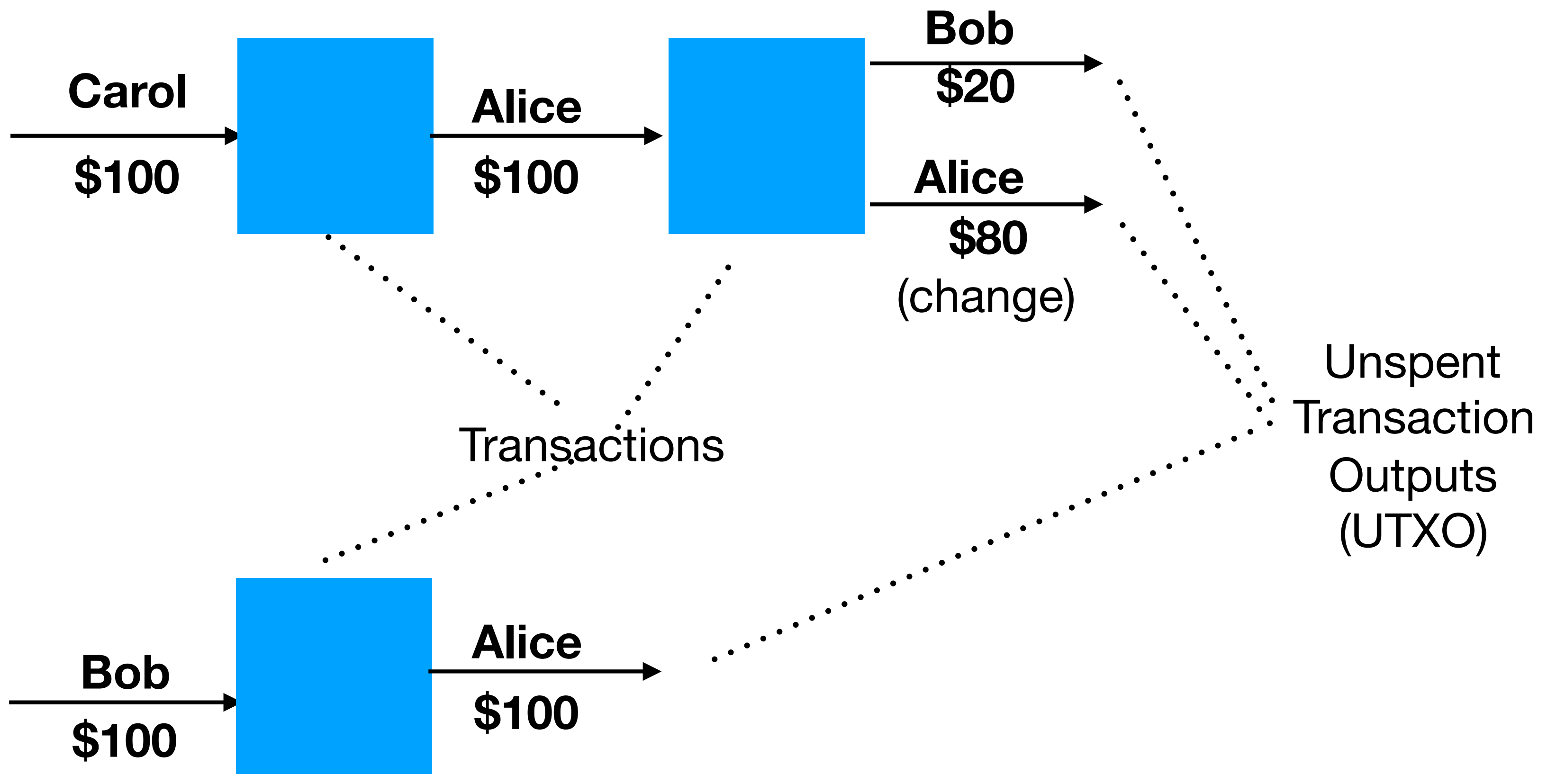
Alice transfers \$20 to Bob

Alice: \$180, Bob: \$520, Carol: \$0

Bitcoin uses a different representation: to send \$20, Alice needs to indicate if she is spending from the \$100 she got from Bob or the \$100 she got from Carol

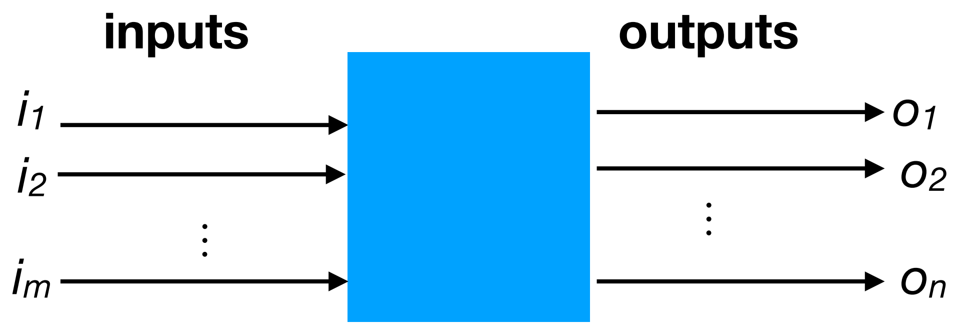
Bitcoin Transaction Abstract Structure

When Alice spends $20 of the $100 she received from Carol, we view the situation like this:



Transaction Abstract Structure

In general, a Bitcoin transaction looks like



where the total amount of inputs $I = \Sigma_{j=1..m}\, i_j$ and outputs $O = \Sigma_{j=1..n}\, o_j$ satisfy $I \geq O$

If $I > O$ then the difference I-O is the *transaction fee,* paid to the miner who includes the transaction in a block.

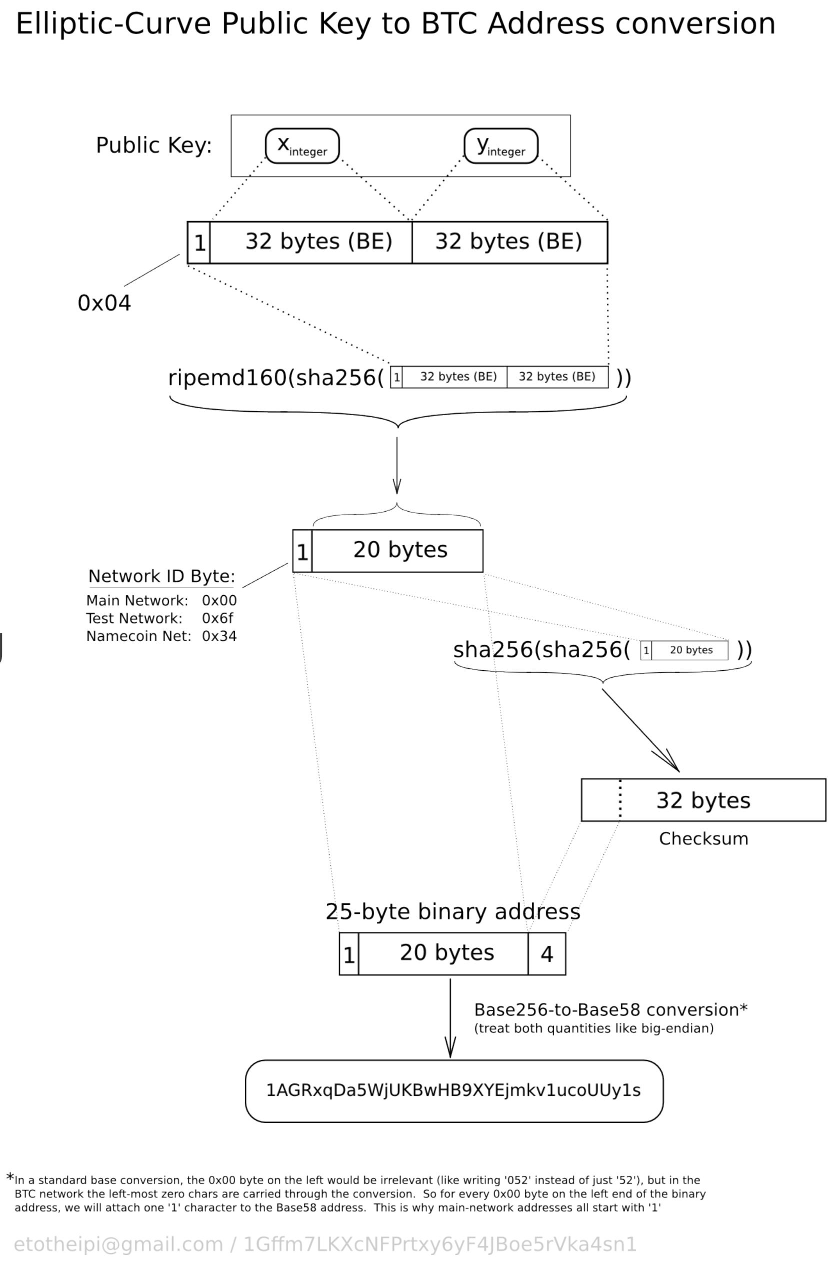
Bitcoin Addresses

An *address* is

- a destination where Bitcoin can be sent/held.
- constructed from a user's EDCSA public key,
- a 160-bit hash + checksum
- encoded using the Satoshi's Base58Check encoding scheme (Upper+Lower case letters + digits, but, for usability, omits ambiguous characters like 0 and O, I (upper i) and l (lower ell)) Includes a checksum.

Why hash the public key?

- shorter, so more convenient
- hiding the public key protects against attempts to compute the private key from public key



Elliptic-Curve Public Key to BTC Address conversion
Public Key:
$x_{integer}$
$y_{integer}$
1
32 bytes (BE)
32 bytes (BE)
0x04
ripemd160(sha256(1 32 bytes (BE) 32 bytes (BE)))
1
20 bytes
Network ID Byte:
Main Network: 0x00
Test Network: 0x6f
Namecoin Net: 0x34
sha256(sha256(1 20 bytes))
32 bytes
Checksum
25-byte binary address
1
20 bytes
4
Base256-to-Base58 conversion*
(treat both quantities like big-endian)
1AGRxqDa5WjUKBwHB9XYEjmkv1ucoUUy1s
etotheipi@gmail.com / 1Gffm7LKXcNFPrtxy6yF4JBoe5rVka4sn1

*In a standard base conversion, the 0x00 byte on the left would be irrelevant (like writing '052' instead of just '52'), but in the BTC network the left-most zero chars are carried through the conversion. So for every 0x00 byte on the left end of the binary address, we will attach one '1' character to the Base58 address. This is why main-network addresses all start with '1'

Image: https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses

Transactions in more detail: structure

A transaction consists of the following parts:

- metadata = {
  - hash of the transaction
    (used as an identifier of the transaction)
  - version number
  - number of inputs
  - number of outputs
  - lock time (earliest time the transaction can be included in a block)
  - size of the transaction }
- inputs (an array)
- outputs (an array)

Transactions in more detail: Transaction outputs

Each output has the following structure:

- The *value* of the output
  (an amount of Bitcoin, in Satoshis)
- A *locking script* that specifies a condition that needs to be met for the value to be released, and the output *spent*

The simplest type of locking script describes a “recipient” of the output (the holder of a private key), by saying “release this output into a transaction signed using (the corresponding private key)”.

However, more complicated types of locking scripts can be specified:

Locking scripts are programs written in *Bitcoin Script,* a small programming language.

Transactions in more detail: Transaction inputs

Each input specifies the following:

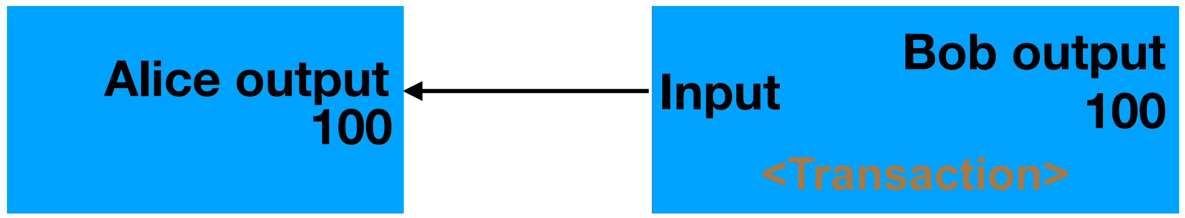
- *hash* of a previous transaction
- *index* of one of the outputs of the previous transaction
- *unlocking script size*
- *unlocking script*
  (proof that the unlocking condition of the previous output has been satisfied)
- *sequence number* (presently unused, for a future extension under debate)

In order for the transaction to be *valid:*

- the previous transaction output should be *unspent,*
- the unlocking script should satisfy the condition in the output's locking script.

Output and Input Structure for a Simple Transfer

Suppose **<Transaction>** includes a simple transfer of 100 Satoshi from Alice to Bob.



The corresponding inputs and outputs use Bitcoin script structures that say the following:

Alice output:

value: 100

locking script: "Check that the input consists of two values **<sig>** **<PublicKey>**, where

1. hash(**<PublicKey>)** is **<AliceAddress>**, and
2. **<PublicKey>** verifies that **<sig>** is a signature of hash(**<Transaction>**)"

Input:

unlocking script : <hash(**<Transaction>**) signed **AlicePrivateKey**> **<AlicePublicKey>**

Recall that **<AliceAddress>** = hash(**<AlicePublicKey>**)

Security Argument for the Transfer

Note that anyone who knows **<AliceAddress>** is able to construct the output locking script.

This means that anyone could have sent the money to Alice by constructing a transaction with this output.

For the unlocking condition to evaluate to True, we need:

1. hash(**<PublicKey>**) = **<AliceAddress>**

   The only value likely to satisfy this is **<PublicKey>** = **<AlicePublicKey>**

2. **<PublicKey>**, i.e., **<AlicePublicKey>** verifies that **<sig>** is a signature of hash(**<Transaction>**)

   The only person likely to have been able to construct such a **<sig>** is Alice, using **<AlicePrivateKey>**

   (We assume Alice has kept **<AlicePrivateKey>** private!)

Summary

That is, with this input/output pattern:

- To *send* someone money, you only need to know their address
- To *spend* money sent to an address, you need to know the associated private key

Summary

That is, with this input/output pattern:

- To *send* someone money, you only need to know their address
- To *spend* money sent to an address, you need to know the associated private key

Bitcoin Script in More Detail

Bitcoin Script is based on a *stack*-based memory model.

There are *conditional statements*, but *no loops* - this guarantees termination.

A program is a *linear* sequence of *Opcodes* (*instructions)* and *data*.

- A data value is simply pushed onto the stack
- Each Opcode *may* do any of
  - consume some values from the top of the stack,
  - calculate a value
  - push a result value onto the top of the stack.

An input *I* in a transaction *T* validly consumes an unspent output *O*, when executing

unlocking-script(I) ; locking-script(O)

(L to R) leaves the stack containing just value TRUE on termination

Opcode Examples

OP_ADD (pop the two top values on the stack, add them, and push the result)

OP_SUB (pop the two top values on the stack, subtract first from second, and push the result)

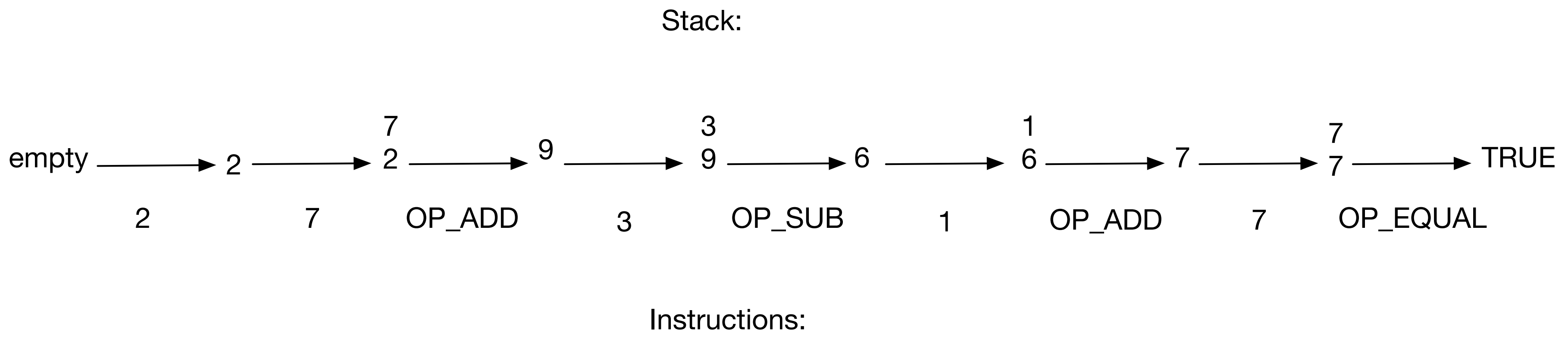
OP_EQUAL (test the two top values for equality and push the Boolean result)

OP_DUP (duplicate the top value on the stack, i.e. push a copy onto the stack)

Example script:

2 7 OP_ADD 3 OP_SUB 1 OP_ADD 7 OP_EQUAL

Executes as:



Exercise:

What does this return?

```
2  OP_DUP  OP_ADD  OP_DUP  OP_ADD  4 OP_EQUAL
```

Some More (Cryptographic) Opcodes:

OP_HASH160 (pop x, and push (RIPEMD(SHA256(x)))

OP_HASH256 (pop x, and push (SHA256(SHA256(x)))

OP_CHECKSIG (pop K and S and push the result of checking if
K is a public key that verifies S as a signature
(by the corresponding private key) of
the hash of the current transaction)

OP_EQUALVERIFY (first check equality of the top two values,
If TRUE then run OP_CHECKSIG on the next two values)

Using these, we can write the “Simple Transfer” example in Bitcoin Script ….

Pay to Public Key Hash Transaction Script

The Simple Transfer has

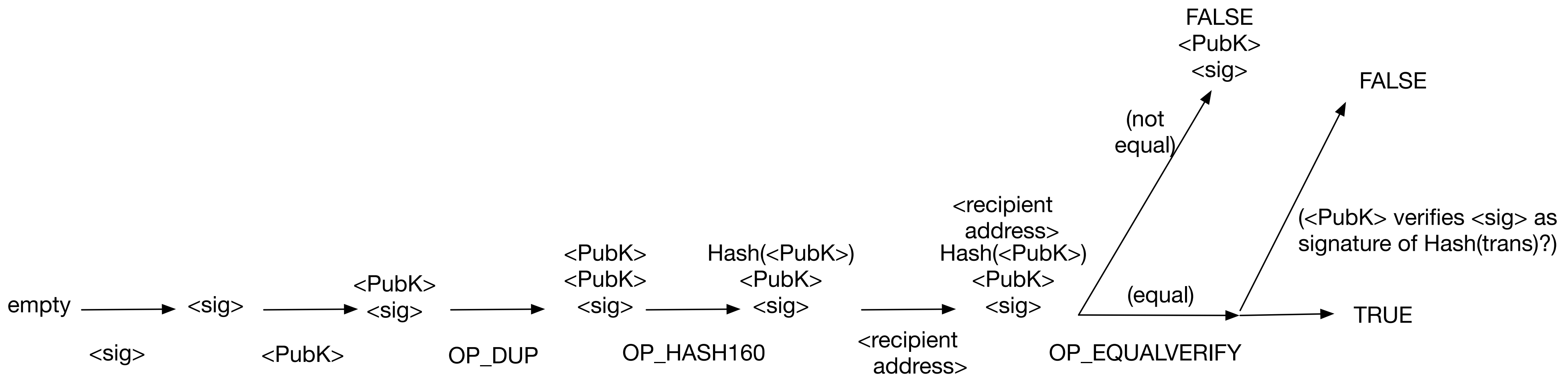
Output locking script:

OP_DUP OP_HASH160 <recipient address> OP_EQUALVERIFY

Input unlocking script:

<sig> <publicKey>

These combine to give the following computation:



Coinbase Transactions

The first transaction in a block is a *coinbase*, or *generation transaction,* constructed by the miner of the block, to pay themselves the block mining reward and transaction fees.

The output of this transaction needs to be equal to

block reward + the sum of the fees from transactions in the block

It has the same structure as a normal transaction, except that there is no input being spent. This leaves a field that can be used to encode a message.

E.g., Satoshi put the following in the input of the coinbase transaction of the first block:

“The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”

Once block difficulty became high, miners also started using this for extra nonce space.