COINMETRICS

Backend Software Engineer

Take-Home Assignment

Version 1.0

Last Revised - June 30, 2021

OBJECTIVES

The test task consists of **four assignments** (A, B, C, D).
A and B are Kotlin/Java tasks and C and D are SQL tasks.

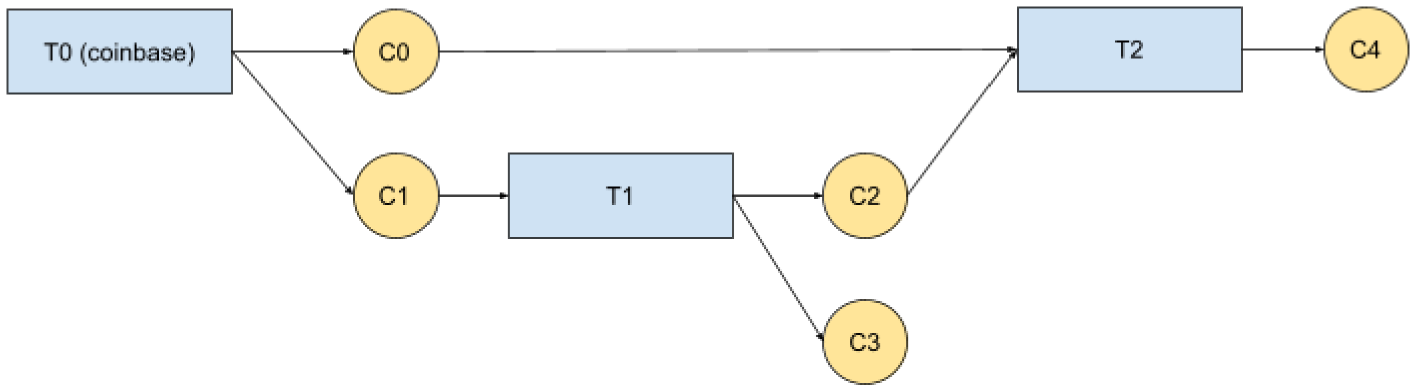
The preliminary reading for candidates not familiar with Bitcoin concepts:

- **How a Bitcoin Transaction Works.pdf** is sent along with this assignment.
- https://en.bitcoin.it/wiki/Coinbase

KOTLIN/JAVA TASKS

DEFINITIONS

1. Coin A is called the *parent* of coin B if B is one of the outputs of transaction T, and A is one of the inputs.
2. Coin A is called an *ancestor* of coin B if there exists a chain of coins C_0 , C_1 , ... , C_n such that A is a parent of C_0 , C_n is a parent of B, and C_i is the parent of C_{i+1} for each i. If A is the parent of B then A is also an ancestor of B.
3. Transaction T is called the *creator* of coin C if C is one of the T's outputs.
4. Transaction T is called the *spender* of coin C if C is one of the T's inputs.
5. Coin A is called the *coinbase ancestor* of coin B if A is an ancestor of B and the creator of A is the coinbase (generation) transaction.
6. Balance of an address A is the sum of all unspent coins that belong to this address.



In the figure above, coin C1 is the parent of coins C2 and C3. Coins C0, C1, and C2 are ancestors of coin C4, but only C0 and C1 are its coinbase ancestors. The transaction T2 is the creator of the coin C4 and spender of coins C0 and C2.

Let us assume that Bitcoin-like blockchain is represented by the following in-memory data structures (C++):

```
struct Blockchain
{
 std::vector<Block*> blocks; // this list is ordered by block_time in
ascending order, i.e. if i > j then blocks[i].block_time >=
blocks[j].block_time
};

struct Block
{
 std::chrono::time_point block_time;
 std::vector<Transaction*> transactions;
};

struct Transaction
{
 Block* block; // never is a nullptr
 bool is_coinbase;
 std::vector<Coin*> inputs; // this list is guaranteed to be empty for
coinbase transactions (is_coinbase=true)
 std::vector<Coin*> outputs;
};

struct Coin
{
 Transaction* creator_transaction; // never is a nullptr
 Transaction* spender_transaction; // is nullptr, if the coin isn't spent
yet
 std::string address;
 uint64_t amount;
};
```

TASKS

A) Write a function that outputs the address that received the maximum total value of coins for a given time range.

```
std::string find_maximum_inbound_volume_address(const Blockchain*
blockchain, std::chrono::time_point interval_start,
std::chrono::time_point interval_end);
```

B) Write a function that returns all coinbase ancestors for a given coin.

```
/* please choose the return type yourself */
find_coinbase_ancestors(const Coin* coin);
```

SQL TASKS

Let's assume that Bitcoin-like blockchain is represented by the following PostgreSQL tables:

```
CREATE TABLE blocks (
 block_hash TEXT PRIMARY KEY,
 block_height INTEGER,
 block_time TIMESTAMPTZ
)
CREATE TABLE transactions (
 tx_hash TEXT PRIMARY KEY,
 tx_block_hash TEXT REFERENCES blocks(block_hash)
)
CREATE TABLE coins {
 coin_address TEXT,
 coin_amount BIGINT,
 coin_creator_tx_hash TEXT REFERENCES transactions(tx_hash),
 coin_spender_tx_hash TEXT REFERENCES transactions(tx_hash) -- is NULL if
the coin isn't spent yet
}
```

C) Write a query that will return the address with maximum even balance.

D) Write a query that will return the top 10 blocks ordered by the total volume of spent coins.

SUBMISSION CRITERIA

- The time frame for the submission is 7 days from receipt.
- Please confirm that you have received the document and that you are able to open it. We count the time from the moment we have received the confirmation.
- If you need an extension, please let us know.
- For Kotlin/Java tasks, the language of the solution should be Kotlin or Java. The C++ code is provided for reference only.
- Kotlin/Java tasks should be completed with near-production ready quality, and with tests.
- You may submit the completed code as either a zip file or a private git repository.
- You may consider any assumptions you make as pre-approved.

How a Bitcoin Transaction Works

By Venzen Khaosan @venzen

Originally published at https://www.ccn.com/bitcoin-transaction-really-works/

This article explains what a Bitcoin transaction is, its purpose and outcome. The explanation made below is suitable for both novice and intermediate Bitcoin users.

As a cryptocurrency user you need to be familiar with transaction rudiments – for the sake of your own confidence with this evolving innovation, and as a foundation for understanding emerging *multi-signature* transactions and *contracts*, both of which will be explored later in the series. This is not a technical article and the explanation will focus on what you need to know about **standard bitcoin transactions** –

the spend transactions we commonly make – and we'll gloss over what you can safely ignore.

An infographic at the bottom of the article provides a comprehensive illustration of the entire Bitcoin transaction process from wallet to blockchain.

Note: Even the Core developers acknowledge that some of the language being used to describe transactions and their components can lead one to a mistaken concept of what is really happening. These misconceptions are avoided in the explanation below. So, while trying to keep things as simple as possible, and with the aid of a few diagrams, let's dive right in.

Definition of Terms and Abbreviations

Bitcoin with a capital "B" refers to the protocol – the code, the nodes, the network and their peer-to-peer interaction.

bitcoin with a lowercase 'b' refers to the currency – the cryptocurrency we send and receive, via the Bitcoin network.

tx – wherever it is used in the text – is an abbreviation for '**Bitcoin transaction**'

txid is an abbreviation for 'transaction id' – this is a hash that is used by both humans and the protocol to reference transactions.

Script is the name of the Bitcoin protocol's scripting system that processes and validates transactions. Script is a clever, stack-based instruction engine, and it makes all transactions from simple payments to complex oracle overseen contracts possible.

UTXO is an abbreviation for Unspent Transaction Output, also referred to as an “output”.

satoshi – 1 BTC = 100,000,000 satoshi

What is a Bitcoin Transaction and Why?

Definition

A Bitcoin transaction is a signed piece of data that is broadcast to the network and, if valid, ends up in a block in the blockchain.

Purpose

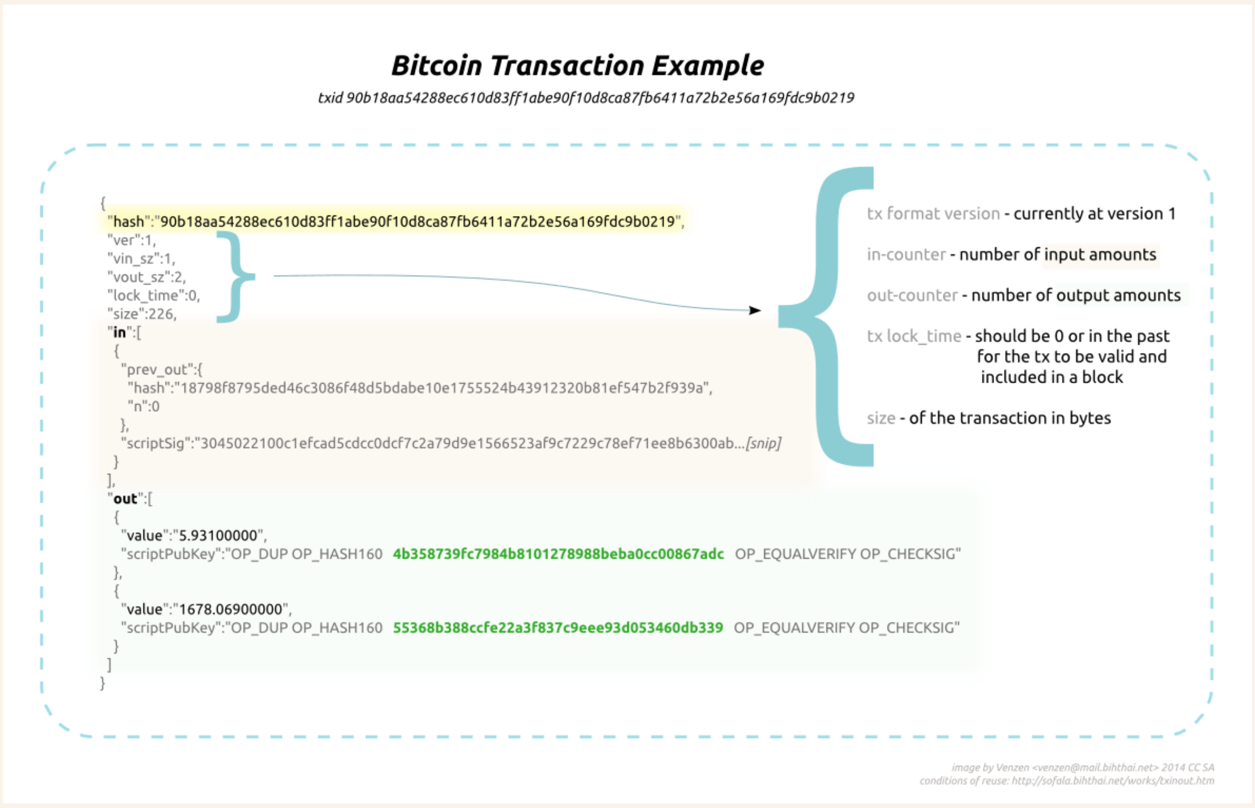
The purpose of a Bitcoin transaction is to *transfer ownership* of an amount of Bitcoin to a Bitcoin address.

Outcome

When you send Bitcoin, a single data structure, namely a Bitcoin transaction, is created by your wallet client and then broadcast to the network. Bitcoin nodes on the network will relay and rebroadcast the transaction, and if the transaction is valid, nodes will include it in the block they are mining. Usually, within 10-20mins, the transaction will be included, along with other transactions, in a block in the blockchain. At this point, the receiver is able to see the transaction amount in their wallet.

Example

Here is an example transaction that was included in the blockchain earlier this year:


Bitcoin Transaction Example
txid 90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219
tx format version - currently at version 1
in-counter - number of input amounts
out-counter - number of output amounts
tx lock_time - should be 0 or in the past for the tx to be valid and included in a block
size - of the transaction in bytes
image by Venzen <venzen@mail.bihthai.net> 2014 CC SA
conditions of reuse: http://sofala.bihthai.net/works/txinout.htm

```
{
 "hash":"90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219",
 "ver":1,
 "vin_sz":1,
 "vout_sz":2,
 "lock_time":0,
 "size":226,
 "in":[
  {
   "prev_out":{
    "hash":"18798f8795ded46c3086f48d5bdabe10e1755524b43912320b81ef547b2f939a",
    "n":0
   },
   "scriptSig":"3045022100c1efcad5cdcc0dcf7c2a79d9e1566523af9c7229c78ef71ee8b6300ab...[snip]
  }
 ],
 "out":[
  {
   "value":"5.93100000",
   "scriptPubKey":"OP_DUP OP_HASH160  4b358739fc7984b8101278988beba0cc00867adc  OP_EQUALVERIFY OP_CHECKSIG"
  },
  {
   "value":"1678.06900000",
   "scriptPubKey":"OP_DUP OP_HASH160  55368b388ccfe22a3f837c9eee93d053460db339  OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
}
```

The main components of this standard transaction are color-coded:

- **Transaction ID** *(highlighted in yellow)*
- **Descriptors and meta-data** *(blue curly brace elaborated upon to the right)*
- **Inputs** *(pink area)*
- **Outputs** *(green area)*

Bitcoin Transaction Inputs and Outputs

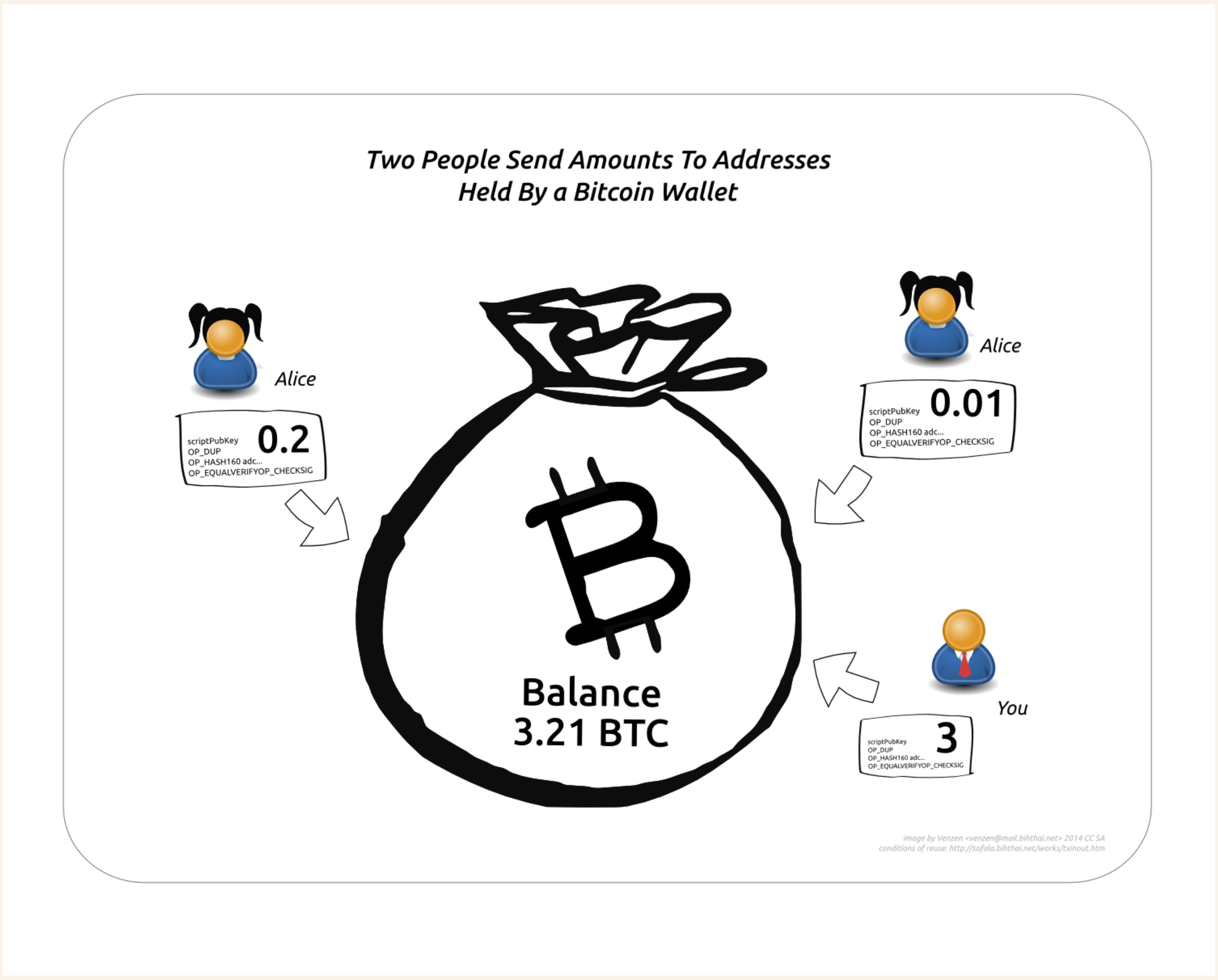
Firstly, four axiomatic truths about transactions:

- *Any Bitcoin amount that we send is always sent to an address.*
- *Any Bitcoin amount we receive is locked to the receiving address – which is (usually) associated with our wallet.*
- *Any time we spend Bitcoin, the amount we spend will always come from funds previously received and currently present in our wallet.*
- *Addresses receive Bitcoin, but they do not send Bitcoin – Bitcoin is sent from a wallet.*

The amounts that go into our wallet are not jumbled like the coins in a physical wallet. The **received amounts don’t mix** but remain separate and distinct as the exact amounts received by the wallet. Here’s an illustration:

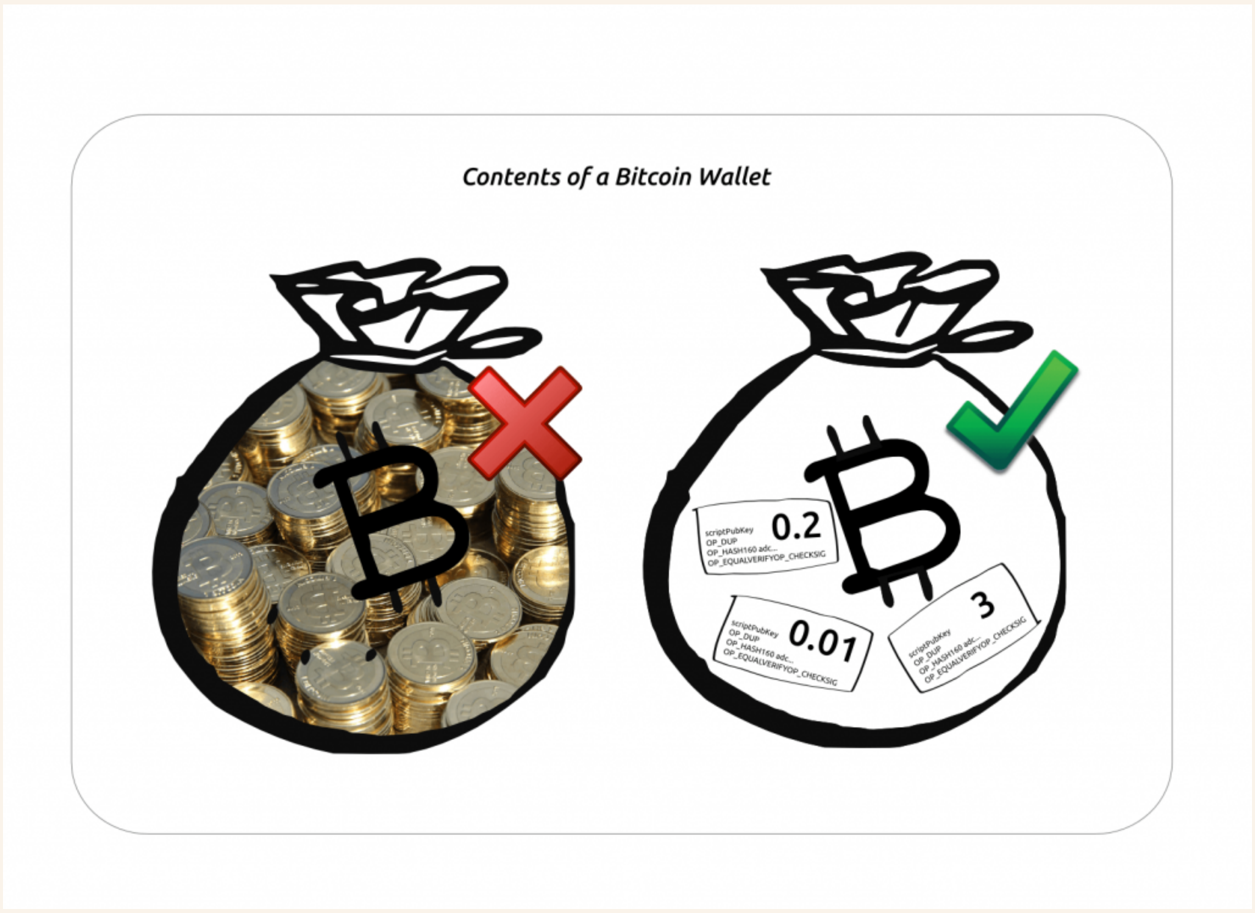
Example

You create a brand new wallet and, in time, it receives three amounts of 0.01, 0.2 and 3 BTC as follows: you send 3 BTC to an address associated with the wallet and two payments are made to another address by Alice.



Two People Send Amounts To Addresses Held By a Bitcoin Wallet
Alice
scriptPubKey OP_DUP OP_HASH160 adc... OP_EQUALVERIFYOP_CHECKSIG 0.2
Alice
scriptPubKey OP_DUP OP_HASH160 adc... OP_EQUALVERIFYOP_CHECKSIG 0.01
Balance 3.21 BTC
You
scriptPubKey OP_DUP OP_HASH160 adc... OP_EQUALVERIFYOP_CHECKSIG 3
image by Venzen <venzen@mail.bihthai.net> 2014 CC SA
conditions of reuse: http://sofala.bihthai.net/works/txinout.htm

The wallet reports a balance of 3.21 BTC, yet if you were to virtually peek inside the wallet, you would see – not 321,000,000 satoshi (321 mil satoshi) – but three distinct amounts still grouped together by their originating transactions: 0.01, 0.2 and 3 BTC.



The received bitcoin amounts don't mix but remain separated as the exact amounts sent to the wallet. The three amounts in the example above are called the **outputs** of their originating transactions.

Bitcoin wallets always keep outputs separate and distinct.

Definition

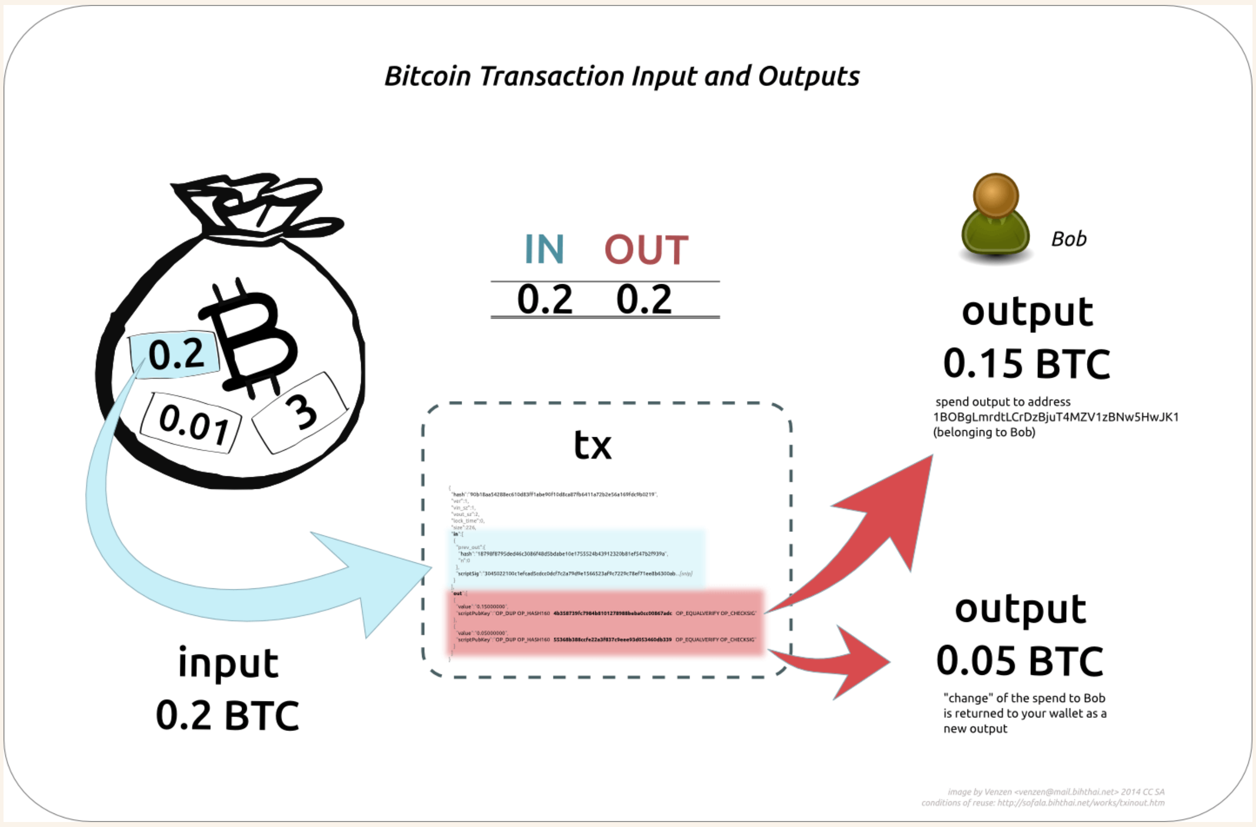
An **output** is an amount that was sent (via a standard transaction) to a Bitcoin address, along with a set of rules to unlock the output amount. In Bitcoin parlance an output is called an *“unspent transaction output”*, or **UTXO**.

A standard transaction output can be unlocked with the private key associated with the receiving address. Addresses and their associated public/private key pairs will be covered later in the series. For now, we are concerned with the output amount only.

Example

Let’s consider an example by following the money in a scenario where you send 0.15 BTC to Bob.

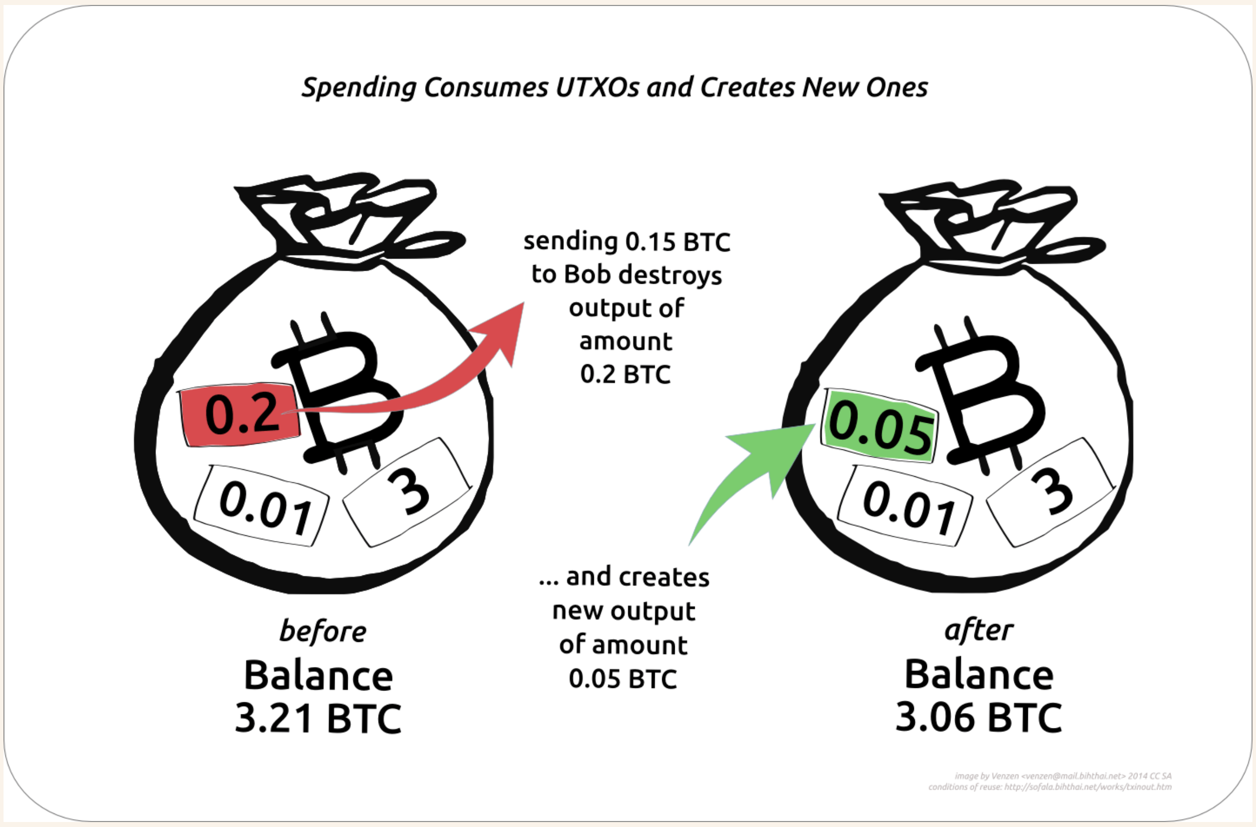
As we have seen, your wallet does not select 15 mil satoshi (0.15 BTC) from an undifferentiated pool of 321 mil satoshi making up the wallet balance. Instead, the wallet selects a spend candidate from amongst the three existing “outputs” contained in the wallet. So, it chooses (for various reasons that are not important now) the 0.2 BTC output. The wallet will unlock the 0.2 BTC output and use the whole amount of 0.2 BTC as an **input** to your new 0.15 BTC transaction. The 0.2 BTC output is “spent” in the process. *–Read this paragraph a second time.*


Bitcoin Transaction Input and Outputs
0.2
0.01
3
input
0.2 BTC
IN OUT
0.2 0.2
tx
Bob
output
0.15 BTC
spend output to address
1BOBgLmrdtLCrDzBjuT4MZV1zBNw5HwJK1
(belonging to Bob)
output
0.05 BTC
"change" of the spend to Bob
is returned to your wallet as a
new output
image by Venzen <venzen@mail.bihthai.net> 2014 CC SA
conditions of reuse: http://sofala.bihthai.net/works/txinout.htm

```
{
 "hash":"90b18aa54288ec610d83ff1abe90f10d8ca87fb6411a72b2e56a169fdc9b0219",
 "ver":1,
 "vin_sz":1,
 "vout_sz":2,
 "lock_time":0,
 "size":226,
 "in":[
  {
   "prev_out":{
    "hash":"18798f8795ded46c3086f48d5bdabe10e1755524b43912320b81ef547b2f939a",
    "n":0
   },
   "scriptSig":"3045022100c1efcad5cdcc0dcf7c2a79d9e1566523af9c7229c78ef71ee8b6300ab...[snip]
  }
 ],
 "out":[
  {
   "value":"0.15000000",
   "scriptPubKey":"OP_DUP OP_HASH160 4b358739fc7984b8101278988beba0cc00867adc OP_EQUALVERIFY OP_CHECKSIG"
  },
  {
   "value":"0.05000000",
   "scriptPubKey":"OP_DUP OP_HASH160 55368b388ccfe22a3f837c9eee93d053460db339 OP_EQUALVERIFY OP_CHECKSIG"
  }
 ]
}
```

The spend transaction your wallet creates will send 0.15 BTC to Bob's address – where it will reside in *his* wallet as an output – waiting eventually to be spent.

The 0.05 BTC difference (0.2 BTC input minus 0.15 BTC output) is called **"change"** and the transaction will send this back to your wallet via a newly created address. The 0.05 BTC change amount will reside in your wallet as a new output – waiting eventually to be spent. So, now, a virtual peek inside your wallet reveals the following:


Spending Consumes UTXOs and Creates New Ones
0.2
0.01
3
sending 0.15 BTC to Bob destroys output of amount 0.2 BTC
... and creates new output of amount 0.05 BTC
0.05
0.01
3
before
Balance 3.21 BTC
after
Balance 3.06 BTC
image by Venzen <venzen@mail.bihthai.net> 2014 CC SA
conditions of reuse: http://sofala.bihthai.net/works/txinout.htm

Each of the three outputs that are “waiting to be spent”, is locked to its receiving addresses until such time as one or more of them are selected as input(s) to a new spend transaction.

Behind the scenes, different wallet clients apply different logic rules when selecting UTXOs as inputs to new transactions. A sane wallet policy is to use older UTXOs first, wherever possible, but implementations differ. The manner in which UTXOs are selected is not of concern to us right now, since the objective has been emphasis of the point that amounts received to our wallets remain separate and distinct.

Summary of How a Bitcoin Transaction Works

Various received amounts don't mix as they do in a physical wallet. Instead, received amounts (UTXOs) are used individually (or in combination) at the moment we spend Bitcoin. When creating the spend transaction our wallet selects UTXOs (of sufficient value to satisfy the amount we want to send) and typically creates two new outputs: one for the receiver and one for the change we receive back to our wallet. The change becomes a brand new UTXO in our wallet, and the amount we send becomes a UTXO locked to the recipient address – which may or may not be associated with a wallet, e.g. cold storage. The original UTXO used as input to the spend transaction is "spent" and destroyed forever.

This has been an introduction to how outputs (UTXOs) are handled by wallet software. Once a UTXO is selected for expenditure, it requires the private key associated with the address that received it. This private key redeems the UTXO and allows it to become an input in a new spend transaction. The mechanism whereby previous transaction outputs are reused as the inputs to new transactions is central to the Bitcoin protocol's function – and exactly as per Satoshi's design.