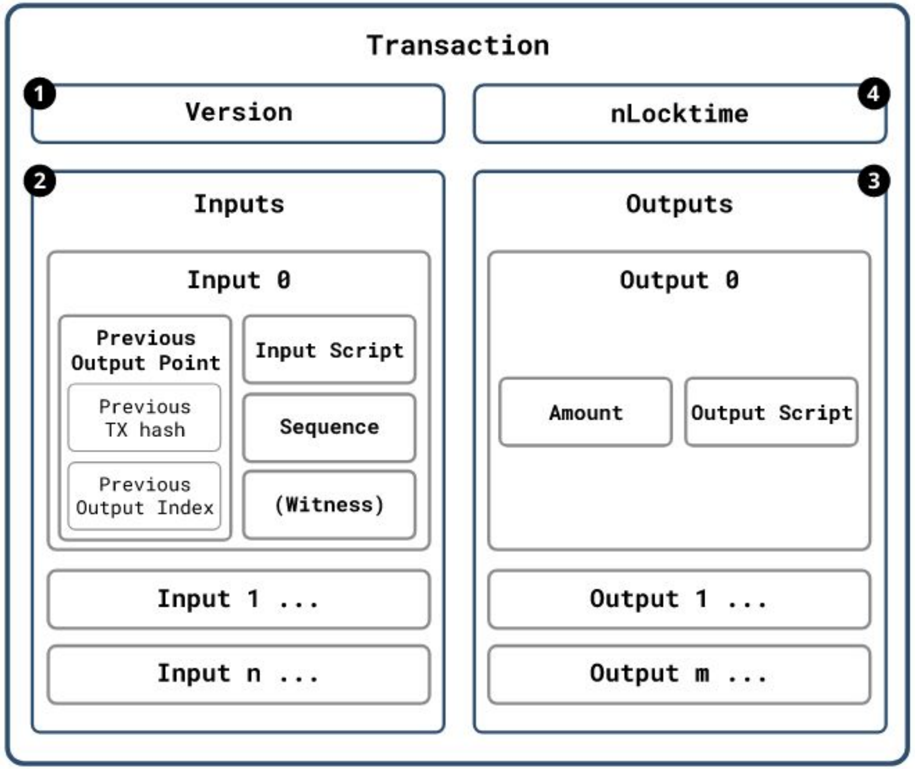
# Wallets

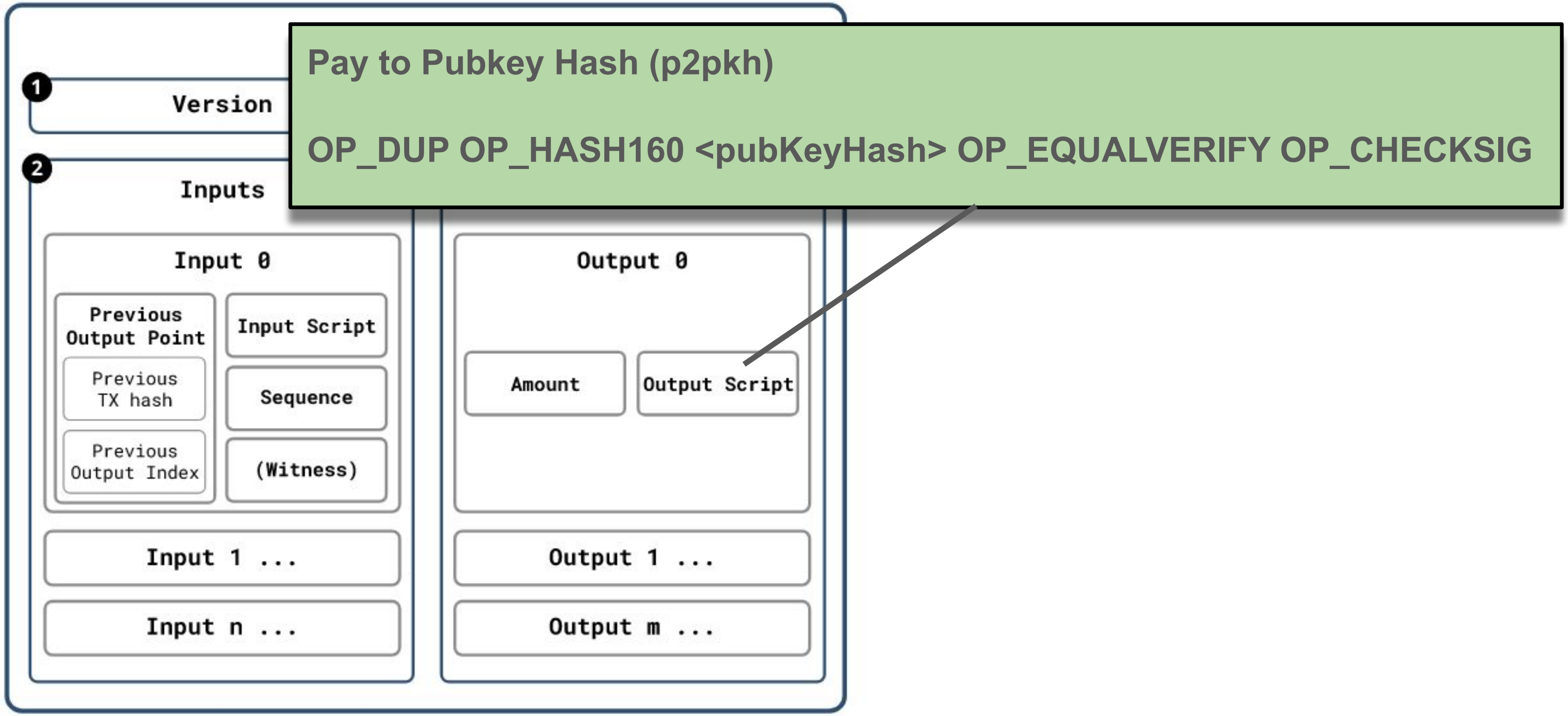
## What is in a wallet?

- Key management?
- Different account types - p2pkh, p2wpkh, multisig
- A lot of BIPs!
- Receive addresses and change addresses
    - Gap limits
- Accessing relevant data from the blockchain
    - Transactions that touch any of your receive or change addresses
- Managing your UTXOs
- Building, signing, and broadcasting transactions
- Privacy considerations

# Refresher: what is an output?



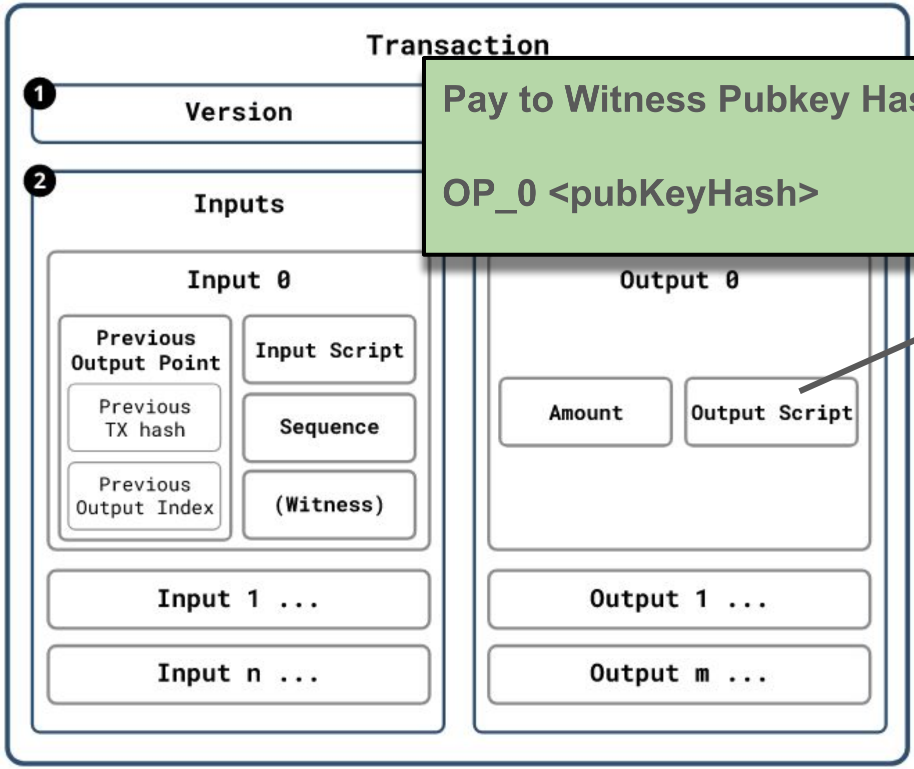
## Refresher: what is an output?



Pay to Pubkey Hash (p2pkh)

OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG

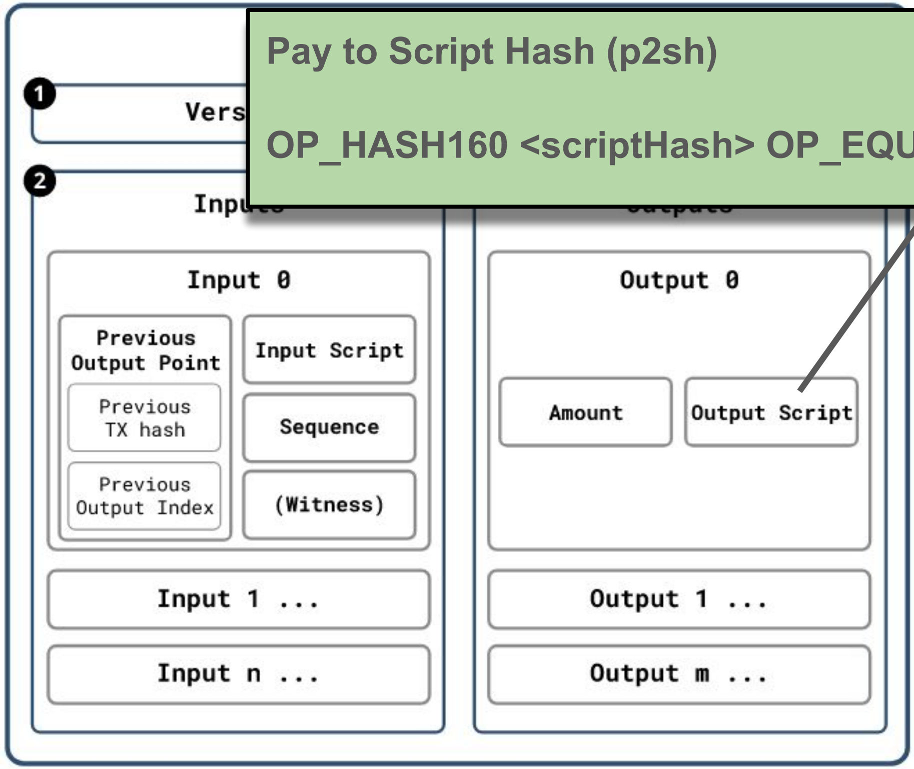
## Refresher: what is an output?



**Pay to Witness Pubkey Hash (p2wpkh)**

**OP_0 <pubKeyHash>**

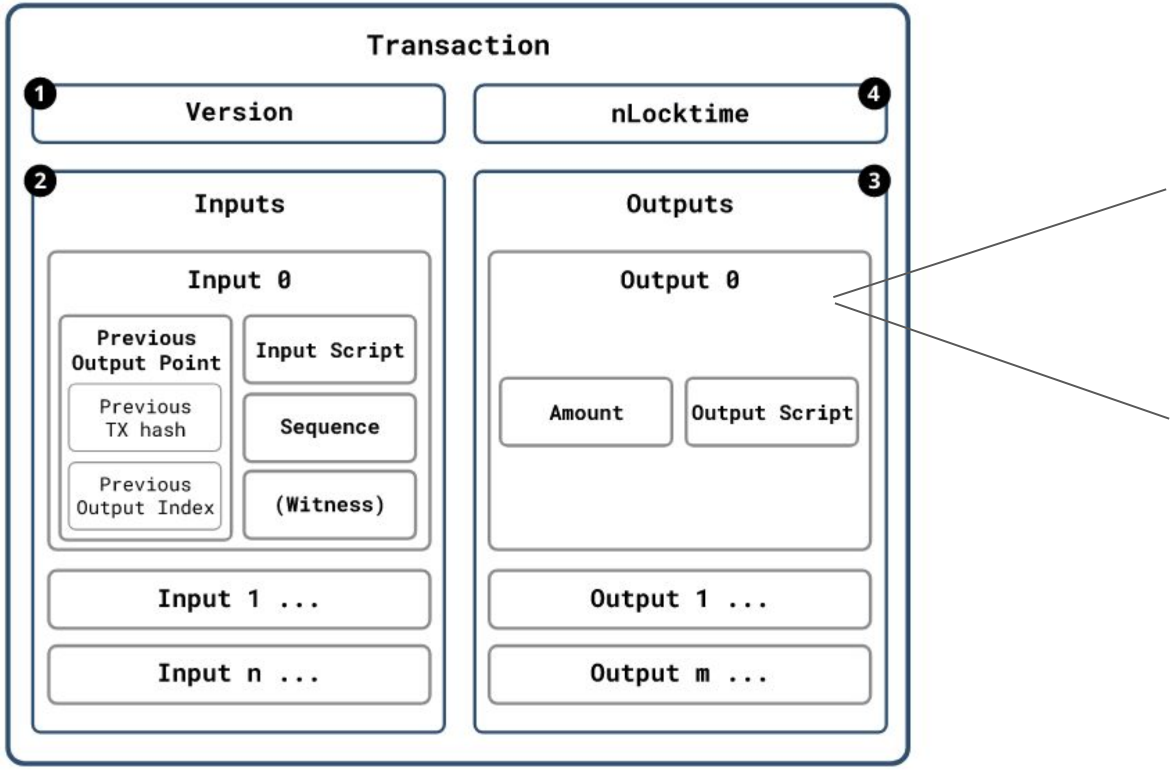
# Refresher: what is an output?



**Pay to Script Hash (p2sh)**

**OP_HASH160 <scriptHash> OP_EQUAL**

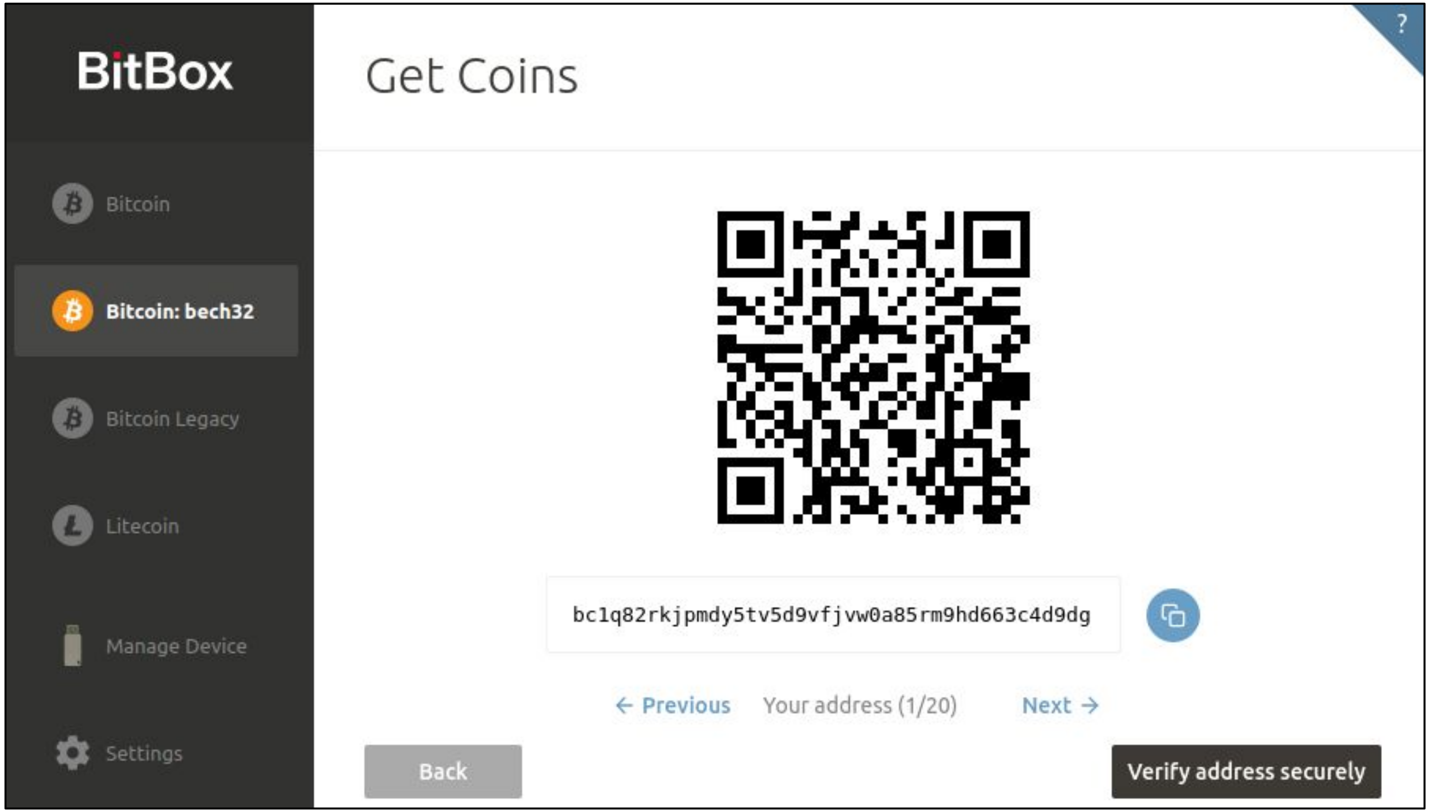
# Refresher: what is an output?



If unspent, called an **UTXO**: Unspent Transaction Output

You have control of the output, if you are able to provide the correct input script

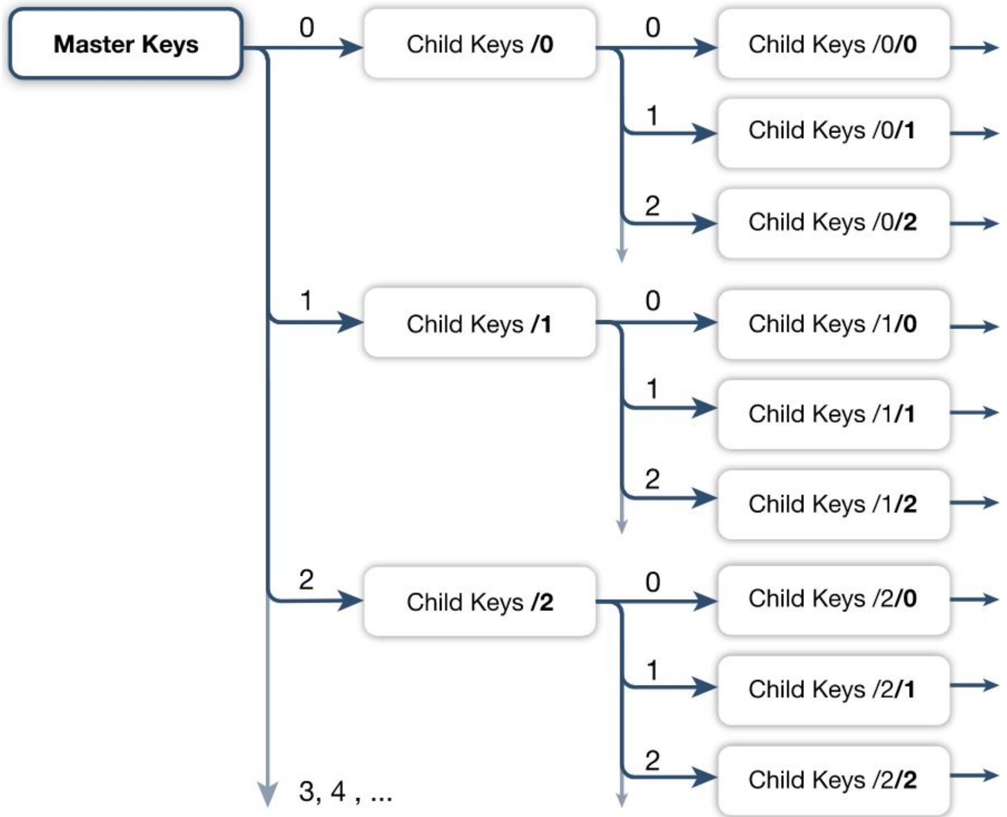
# How to organize keys, accounts, scripts/addresses?



bc1q82rkjpmdy5tv5d9vfjvw0a85rm9hd663c4d9dg

Your address (1/20)

# BIP32 - Hierarchical Deterministic Key derivation



HD wallets (BIP32) can deterministically derive an indefinite number of fresh addresses from a single wallet secret.

**HD Tree**

- Fresh addresses to improve privacy.
- HD Tree is derived from Master Keys.
- HD Tree can be reconstructed from master Keys (given tree structure).

**Master keys**

- Derived from HD root secret.

**Subtrees**

- Allow separation of keys for accounts/usages.
- Selective key sharing.

## BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

## BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH (BIP84):

**Root XPRV** / 84' / 0' / 2' / 0 / 9

points to the 10th key controlling p2wpkh outputs of your 3rd account.

## BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH (BIP84):

**Account XPUB** / 0 / 9

to reconstruct an output script, generate a receive address

## BIP44 - Derivation Paths

Root Key / purpose' / coin' / account' / change / address_index

And a set of BIPs to define the purpose (account type).

Example:

Bitcoin P2WPKH (BIP84):

**Account XPRV** / 0 / 9

to spend outputs matching the output script

## Output Types vs Addresses vs Input Types

How are output types and input types different?

- What info is needed to create an output?
- What info is needed to create an address?

Input type implies output type. Output type implies address (if any)

Example: an account for 2-of-3 multisig wrapped in P2SH:
Input type: 2-of-3 multisig wrapped in P2SH. Determines the output script to receive funds.
Output Type: P2SH. Determines the address.
Address: 3...

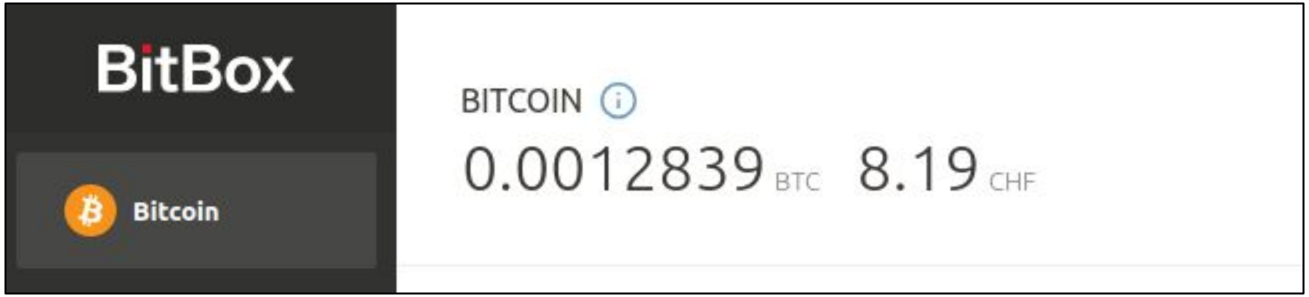
## Output Types vs Addresses vs Input Types

There is an encoding for most standard output scripts.

Output Types: Legacy P2PKH (1…), Legacy P2SH (3…),
Native Segwit equivalents: P2WPKH or P2WSH (bc1…)
Multisig (**no address encoding**)

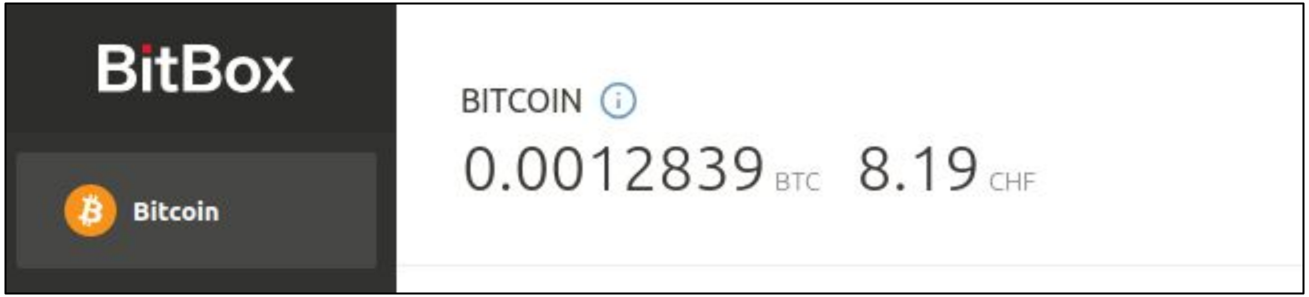
Script/Account Types:
Legacy Pay-to-PubkeyHash: **P2PKH**
Segwit Pay-to-PubkeyHash wrapped in P2SH: **P2WPKH-P2SH**
Native Segwit Pay-to-PubkeyHash **P2WPKH**
Multisig wrapped in P2(W)SH
Timelock scripts wrapped in P2(W)SH

## What is an account balance?



0.0012839 BTC 8.19 CHF

## What is an account balance?


BitBox

Bitcoin

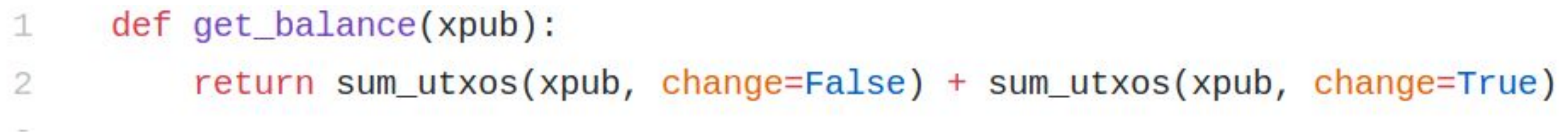
BITCOIN

0.0012839 BTC 8.19 CHF

Sum of all UTXOs found under

**XPRV** / 84' / 0' / 0' / <change> / <index>

## What is an account balance? - Pseudocode



```
def get_balance(xpub):
    return sum_utxos(xpub, change=False) + sum_utxos(xpub, change=True)
```

```python
def sum_utxos(xpub, change):
    xpub = xpub.Child(1 if change else 0)
    gap_limit = 20
    result_sum = 0
    gap, index = 0, 0
    for gap < gap_limit:
        current = xpub.child(index)
        # construct a p2wpkh output script: OP_0 OP_PUSH20 pubkey_hash
        pubkey_hash = hash160(current.publicKey.serializeCompressed())
        output_script = bytearray([0, 20]) + pubkey_hash

        utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
        result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))

        if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
            gap = 0
        else:
            gap += 1
        index += 1
    return result_sum
```

```python
def sum_utxos(xpub, change):
    xpub = xpub.Child(1 if change else 0)
    gap_limit = 20
    result_sum = 0
    gap, index = 0, 0
    for gap < gap_limit:
        current = xpub.chi
        # construct a p2wpk
        pubkey_hash = hash1
        output_script = bytearray([0, 20]) + pubkey_hash

        utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
        result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))

        if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
            gap = 0
        else:
            gap += 1
        index += 1
    return result_sum
```

**Validate!**

- **SPV proofs: were the tx really mined in blocks?**
- **Blocks had sufficient proof of work?**

```python
def sum_utxos(xpub, change):
    xpub = xpub.Child(1 if change else 0)
    gap_limit = 20
    result_sum = 0
    gap, index = 0, 0
    for gap < gap_limit:
        current = xpub.child(index)
        # construct a p2wpkh output script: OP_0 OP_PUSH20 pubkey_hash
        pubkey_hash = hash160(current.publicKey.serializeCompressed())
        output_script = bytearray([0, 20]) + pubkey_hash

        utxos = blockchain.scripthash.listunspent(bitcoin.hash(output_script))
        result_sum += sum(utxo['value'] for utxo in utxos if utxo['confirmed'] or ours(utxo['tx']))

        if bockchain.scripthash.has_history(bitcoin.hash(output_script)):
            gap = 0
        else:
            gap += 1
        index += 1
    return result_sum
```

## Blockchain Indexing Techniques

Problem:

How do we keep track of our coins (UTXOs)?

Given an output script, how do we find all relevant transactions?

- Transactions sending money to it (creating UTXOs)
- Transactions spending it (destroying UTXOs)

We need **`blockchain.scripthash.get_history(output_script_hash)`**`, i.e. something that indexes relevant transactions.`

## Blockchain Indexing Techniques

Three broad areas:

- Personal Index using a full node
- Personal Index using SPV
- Full Index (personal or shared)

Personal: tracks only the utxos and transactions related to your own addresses.

## Personal Index - Bitcoin Core

1. Add an output script (based on a public key) to the watchlist.

2. When a new block arrives, scan all transactions in it:
    a. If a transaction output script matches, add the output to our UTXO set, and the transaction to the transaction list.

    b. If a transaction spends a matching output, remove it from our UTXO set, and add the transaction to the transaction list.

Our balance is the sum of the UTXOs.

## What about Recovery?

Tracking outputs is lightweight, but rescanning the past is not.

Need to rescan the whole chain, which can take a long time.

If willing to lose the transaction history, one can scan only the full UTXO set.

→ Recently released **scantxoutset** RPC call in Bitcoin Core. Still experimental, not yet in use.

## SPV - Bloom Filters and Neutrino

Same indexing as before, but process only a small subset of the blockchain.

Bloom Filters (BIP37): download trimmed blocks with only relevant transactions

Neutrino (BIP157,158): download full blocks, but only relevant ones

(Good for relatively low traffic clients, like Lightning)

## Full Index

A full index maintains the {address: transactions} index for all outputs.

## Full Index

A full index maintains the {address: transactions} index for all outputs.

Often what SPV wallets connect to.

## Full Index

A full index maintains the {address: transactions} index for all outputs.

Often what SPV wallets connect to.

Slow to build index, adds currently ~40GB more data.

## Full Index

A full index maintains the {address: transactions} index for all outputs.

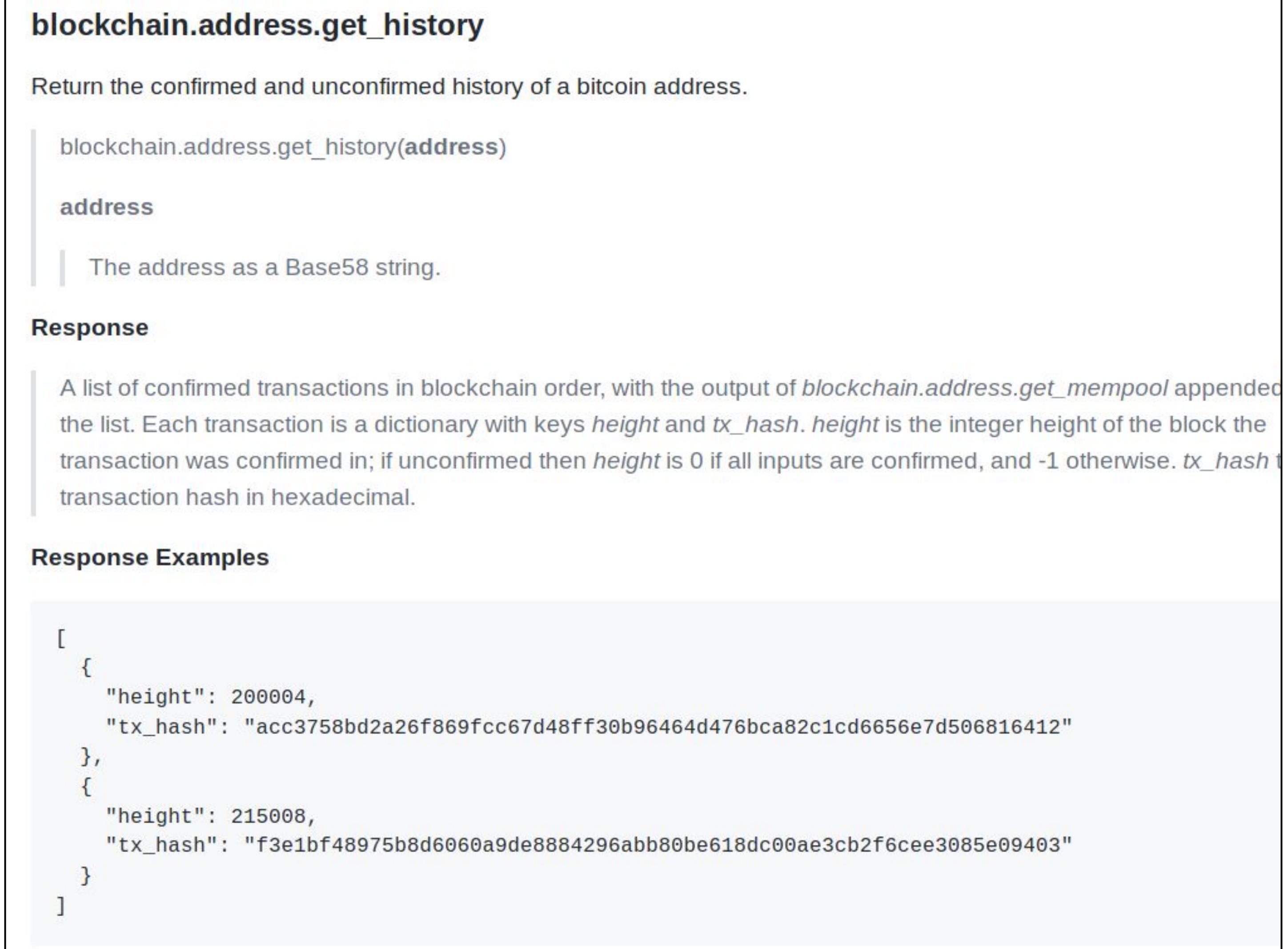
Often what SPV wallets connect to.

Slow to build index, adds currently ~40GB more data.

Fast recovery.

## ElectrumX

de-facto standard protocol



### blockchain.address.get_history

Return the confirmed and unconfirmed history of a bitcoin address.

> blockchain.address.get_history(**address**)
>
> **address**
>
> > The address as a Base58 string.

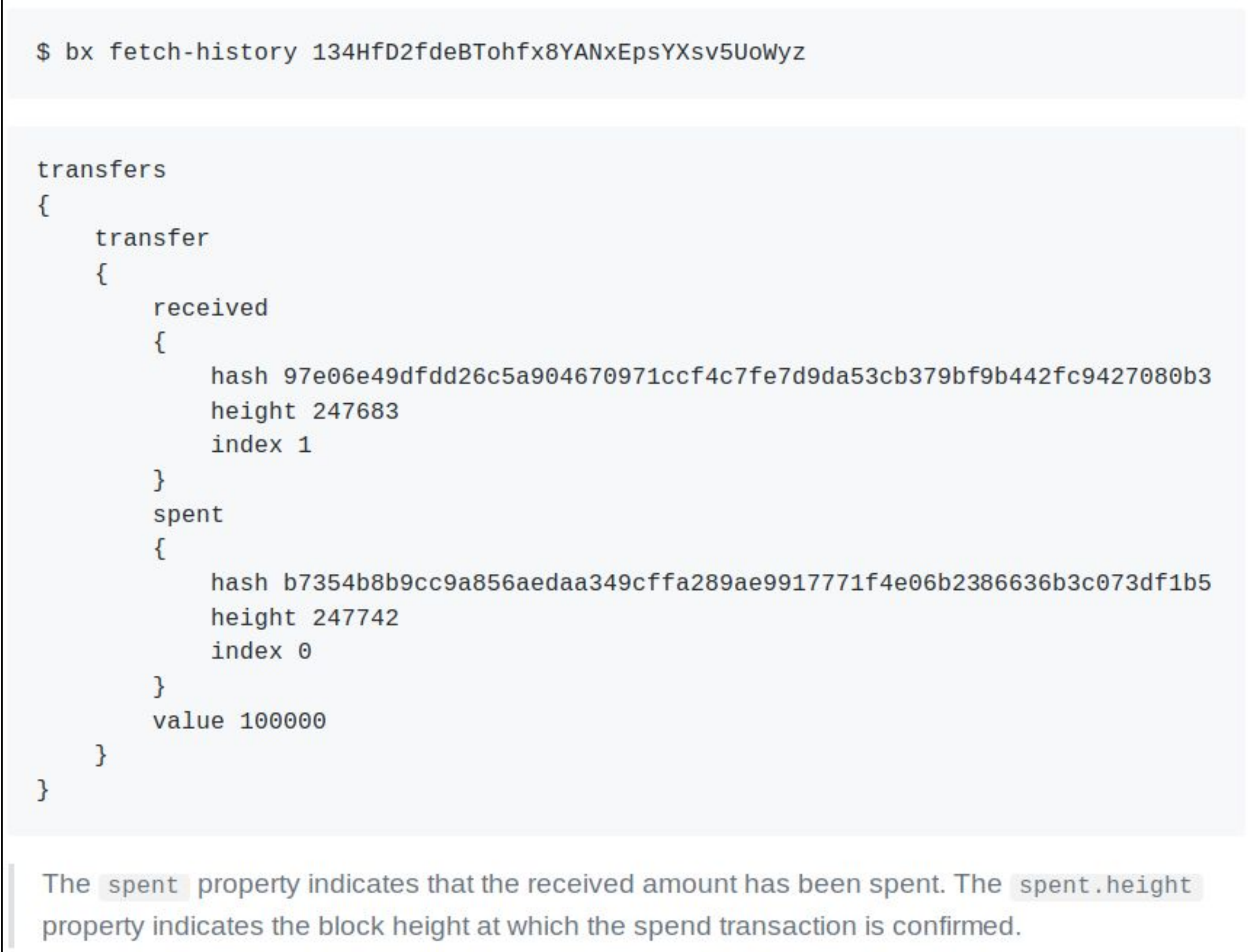
**Response**

> A list of confirmed transactions in blockchain order, with the output of *blockchain.address.get_mempool* appendec the list. Each transaction is a dictionary with keys *height* and *tx_hash*. *height* is the integer height of the block the transaction was confirmed in; if unconfirmed then *height* is 0 if all inputs are confirmed, and -1 otherwise. *tx_hash* t transaction hash in hexadecimal.

**Response Examples**

```
[
  {
    "height": 200004,
    "tx_hash": "acc3758bd2a26f869fcc67d48ff30b96464d476bca82c1cd6656e7d506816412"
  },
  {
    "height": 215008,
    "tx_hash": "f3e1bf48975b8d6060a9de8884296abb80be618dc00ae3cb2f6cee3085e09403"
  }
]
```

## libbitcoin

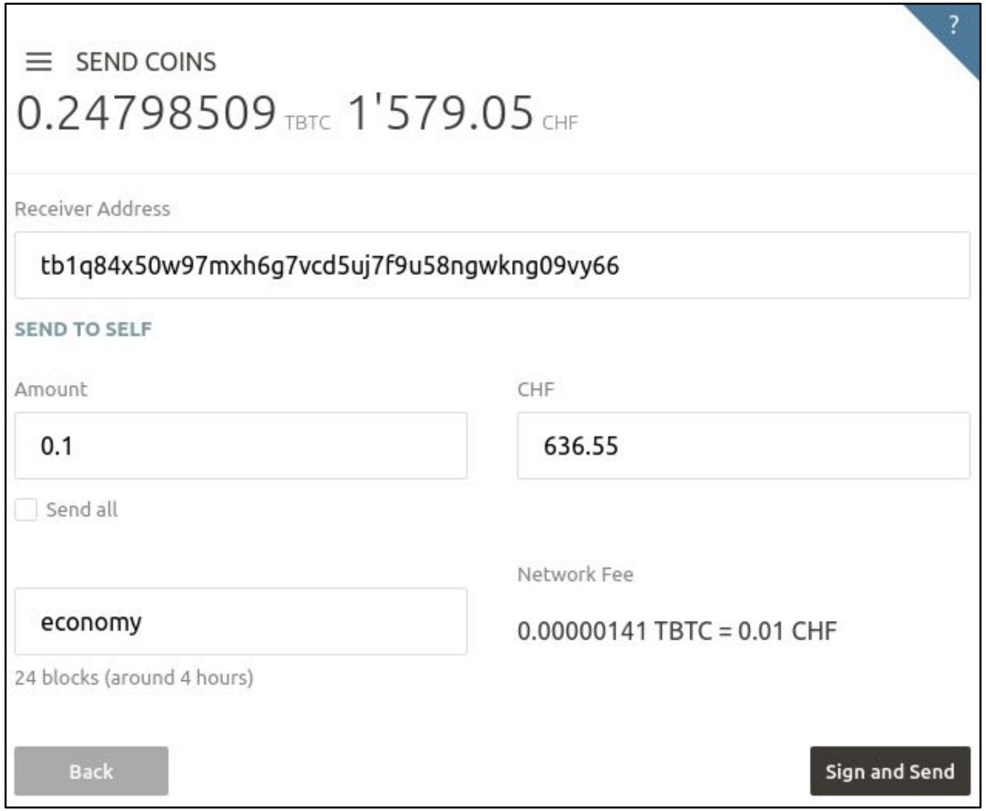


```
$ bx fetch-history 134HfD2fdeBTohfx8YANxEpsYXsv5UoWyz
```

```
transfers
{
    transfer
    {
        received
        {
            hash 97e06e49dfdd26c5a904670971ccf4c7fe7d9da53cb379bf9b442fc9427080b3
            height 247683
            index 1
        }
        spent
        {
            hash b7354b8b9cc9a856aedaa349cffa289ae9917771f4e06b2386636b3c073df1b5
            height 247742
            index 0
        }
        value 100000
    }
}
```

> The `spent` property indicates that the received amount has been spent. The `spent.height` property indicates the block height at which the spend transaction is confirmed.

# Transaction Creation



SEND COINS

0.24798509 TBTC 1'579.05 CHF

Receiver Address: tb1q84x50w97mxh6g7vcd5uj7f9u58ngwkng09vy66

SEND TO SELF

Amount: 0.1

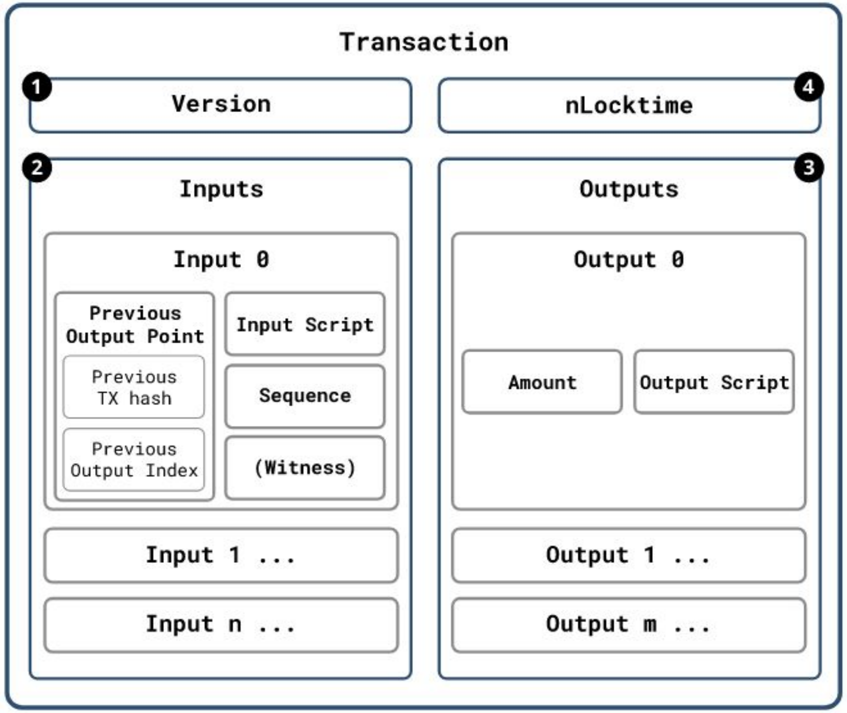
CHF: 636.55

Send all

economy

24 blocks (around 4 hours)

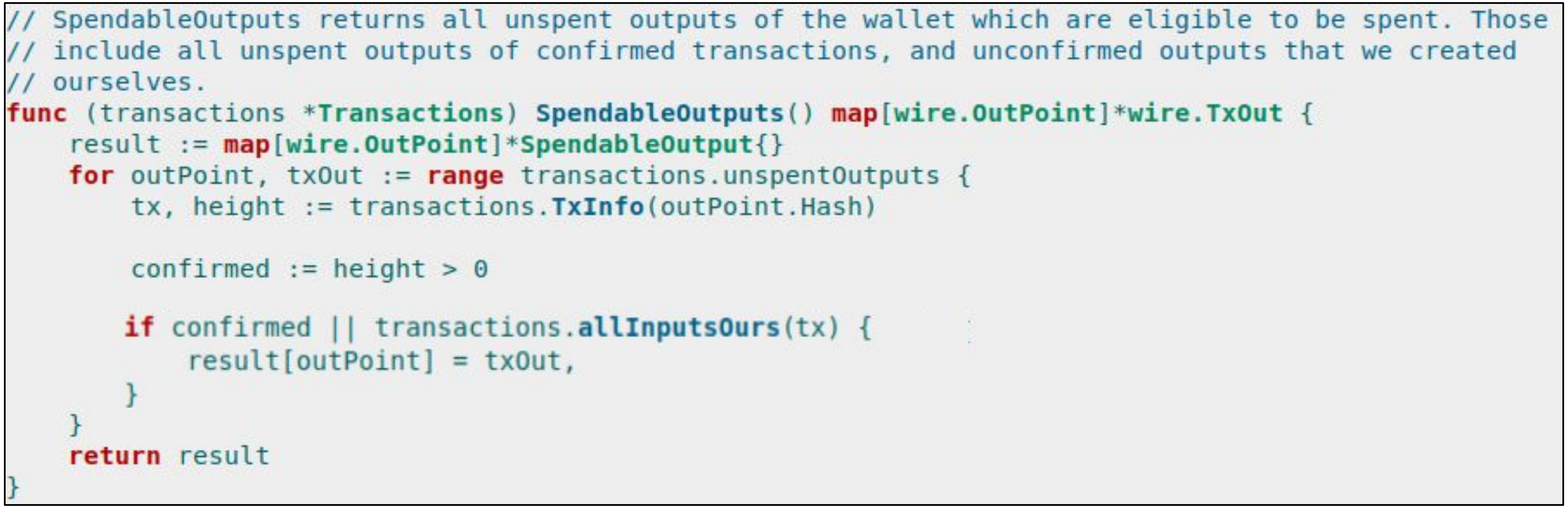
Network Fee: 0.00000141 TBTC = 0.01 CHF



## Transaction Creation - Spendable Outputs

From our UTXOs, filter the ones we can safely spend

## Transaction Creation - Spendable Outputs



```go
// SpendableOutputs returns all unspent outputs of the wallet which are eligible to be spent. Those
// include all unspent outputs of confirmed transactions, and unconfirmed outputs that we created
// ourselves.
func (transactions *Transactions) SpendableOutputs() map[wire.OutPoint]*wire.TxOut {
    result := map[wire.OutPoint]*SpendableOutput{}
    for outPoint, txOut := range transactions.unspentOutputs {
        tx, height := transactions.TxInfo(outPoint.Hash)

        confirmed := height > 0

       if confirmed || transactions.allInputsOurs(tx) {
           result[outPoint] = txOut,
       }
    }
    return result
}
```

## Transaction Creation - Coin Selection

Requirements:

- Input values must cover the output values + mining fee
    - Annoying dependency: the fee also depends on the number of inputs and outputs
- There can be no dust output

Nice to have:

- Reduce UTXO bloat, avoid small change outputs
- Reduce fees: small number of inputs and outputs
- Privacy: make it hard to identify the change output

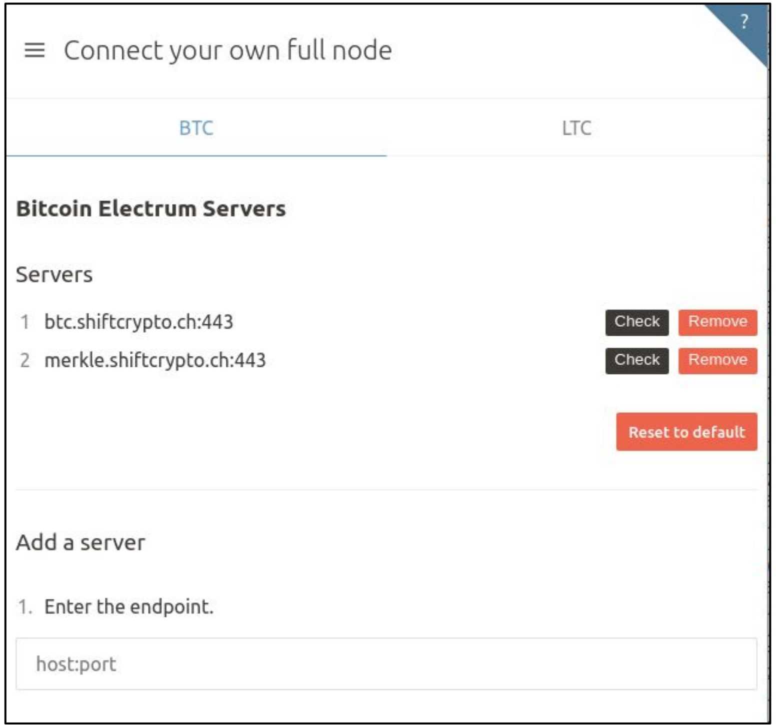
No selection policy works for all.

## Transaction Creation - Coin Selection

- Bitcoin Core's algorithm until recently was very involved
- Turned out that a simple random draw performed better in many of the metrics
- Since last release, using Branch&Bound + Random Draw
  - Optimizing for exact matches to cut costs
  - http://murch.one/wp-content/uploads/2016/11/erhardt2016coinselection.pdf

# Privacy Considerations

Support powering the wallet with your own full node!



**Bitcoin Electrum Servers**

Servers

1 btc.shiftcrypto.ch:443

2 merkle.shiftcrypto.ch:443

Add a server

1. Enter the endpoint.

host:port

## Privacy Considerations

A lot about how a wallet behaves leaks information on the chain

- Types of scripts in use (if there are only a few segwit users, they stand out), and their mix
- Transaction composition can give you away
  (coin selection, input/output ordering)
- All wallets should ideally behave the same way.
- Lightning will help

Others:

- Minimize use of third party servers. Decorrelate with onchain actions.

Fin

## Coding Exercise

Task:

Based on a bip44 account xpub, index the UTXOs and compute the account balance.

- Clone https://github.com/benma/21lectures
- `vagrant ssh` → `pip3 install pywallet`
- Upload the files to your Jupyter VM

Don’t hesitate to ask questions or peek at the solution if you are stuck.

# Coding Exercise

Refresher:

- m/<change>/<address index> derives receive addresses (<change>=0) or change addresses (<change>=1)
- All addresses are scanned (<address index> = 0, 1, 2, …) until there is a long gap of unused addresses
- Account balance is the sum of unspent outputs with those addresses

What's an unspent output? → An output for which there is no transaction input which spends it.

**Happy Hacking!**