Bitcoin Transaction

EXCHANGING VALUE FREELY

Stéphane Roche

Last update: 12-03-2019

ABOUT STEPHANE

2015

Start working on Bitcoin in 2015 at Ledger (hardware wallet)

2016–2017

Work on Ethereum in 2016-2017

- Co-found non-profit organization Asseth
- R&D on Dao1901
- Contribute to the ERC20 Consensys smart contracts
- Dether.io (15,000 ETH raised)

2017–2018

Focus on blockchain technical trainings

Founder of D10eConsulting

Consultant at Chainsmiths

OUTLINE

1 Structure

2 Signing And Verification

3 Transaction Types

4 Wallet Abstractions

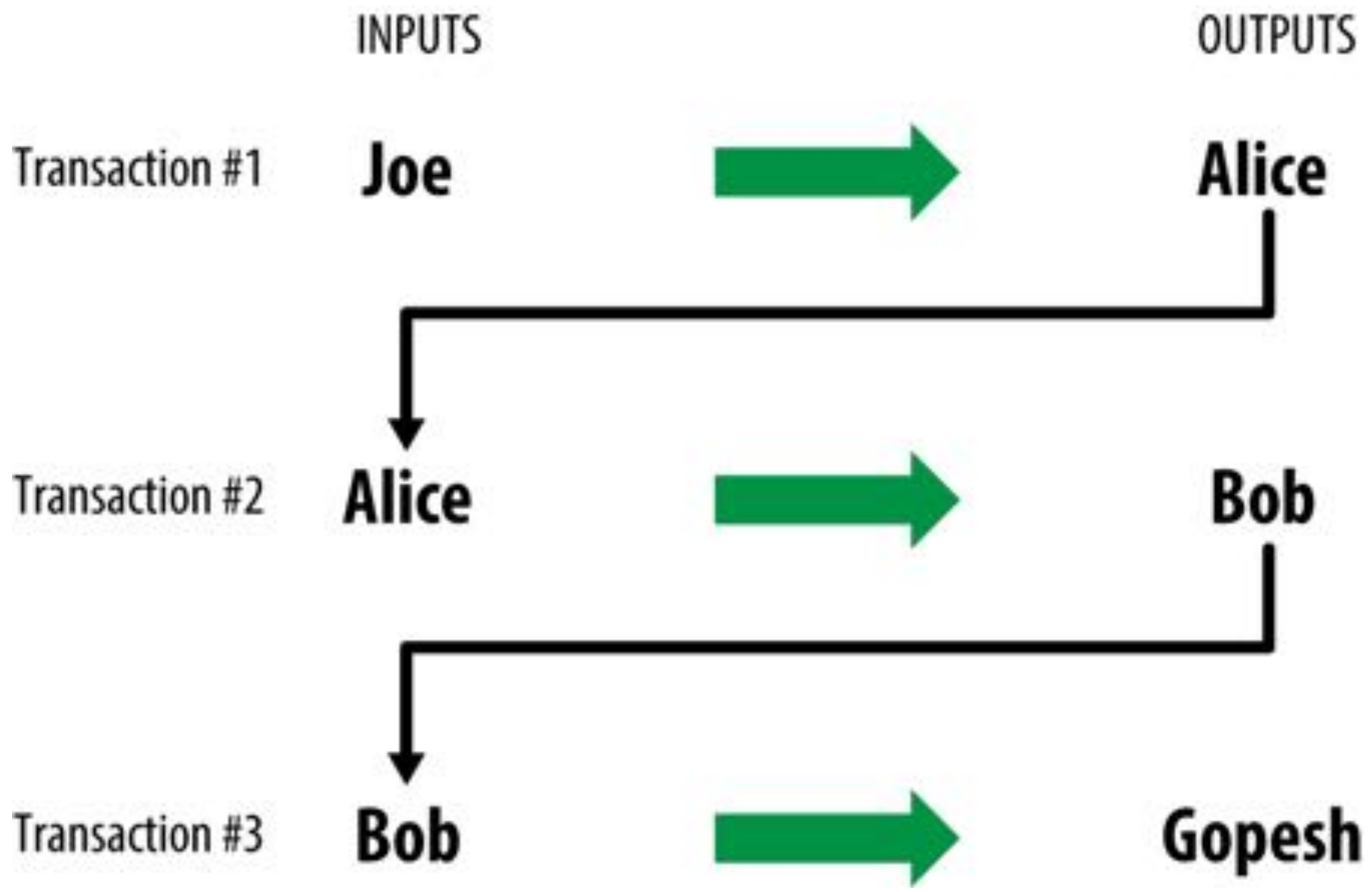
1 STRUCTURE

Transaction as Double-Entry Bookkeeping

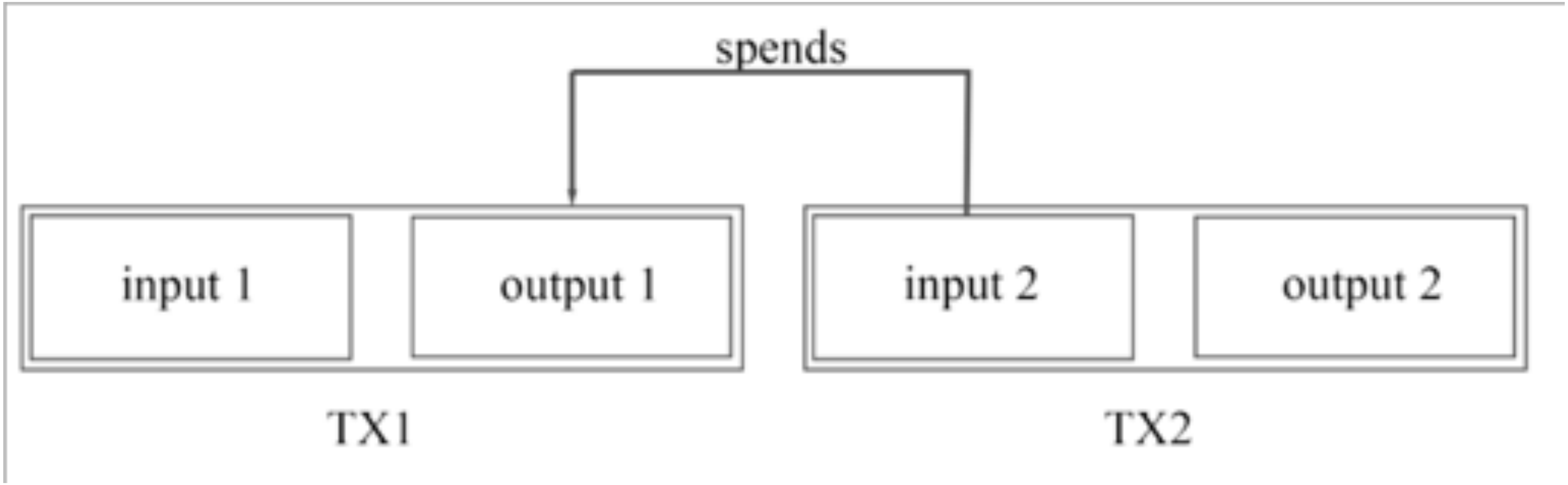
Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC

	Inputs	*0.55 BTC*
-	*Outputs*	*0.50 BTC*
	Difference	*0.05 BTC (implied transaction fee)*

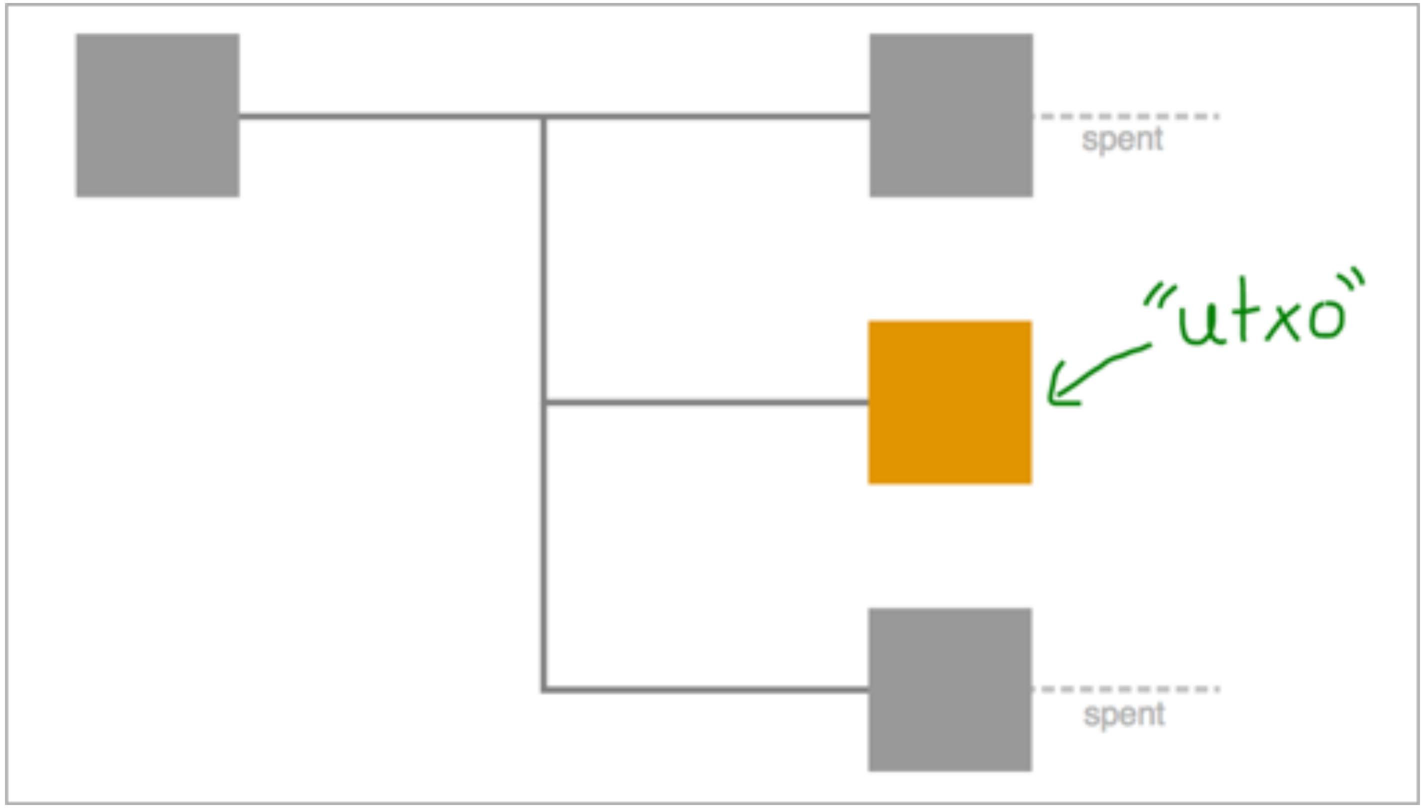
TRANSACTION CHAINS

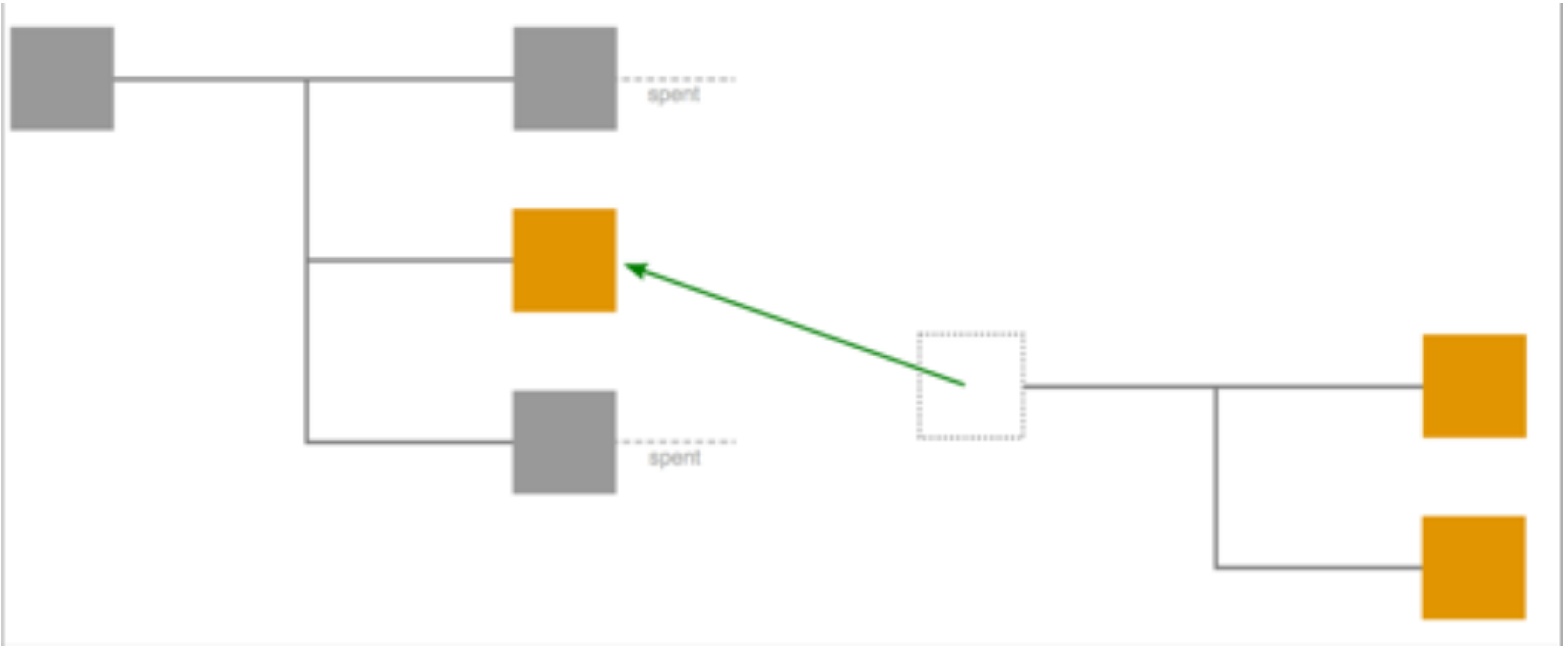


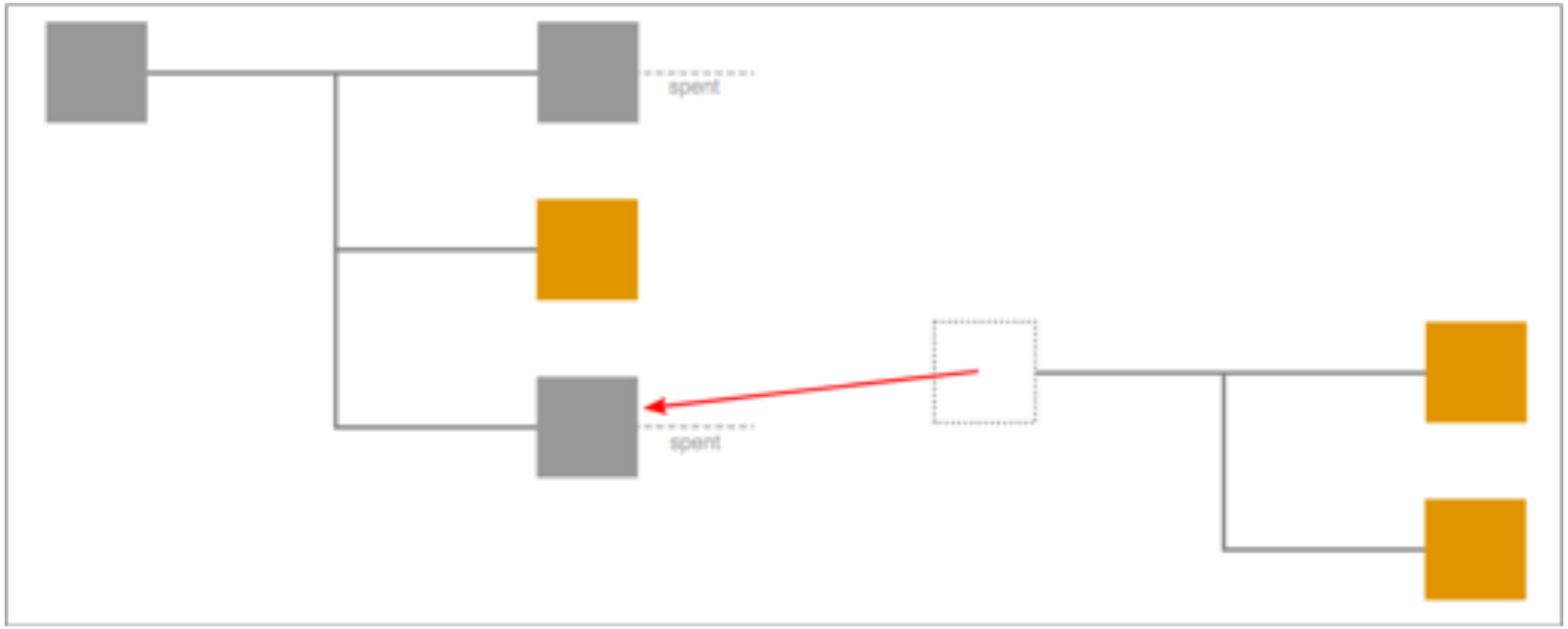
INPUT/OUTPUT CHAIN



SPENT VS UNPSPENT OUTPUTS




spent
spent


spent
spent

TRANSACTION OUTPUTS

- Indivisible chunks of bitcoin currency
- Amount locked to an address, by a locking script
- Recorded on the blockchain and recognized as valid by the entire network
- Full nodes track all available and spendable outputs (UTXO set)
  - Chainstate LevelDB database

TRANSACTION OUTPUT STRUCTURE

- An indivisible amount denominated satoshis
  - An unspent output can only be consumed in its entirety
  - If UTXO > desired tx value, 2 outputs created: paying tx + change tx
- A cryptographic puzzle that determines the conditions required to spend the output (locking script, witness script, scriptPubKey)

TRANSACTION OUTPUT SERIALIZATION

Size	Field	Description
8 bytes (little-endian)	Amount	Bitcoin value in satoshis (10^{-8} bitcoin)
1–9 bytes (VarInt)	Locking-Script Size	Locking-Script length in bytes, to follow
Variable	Locking-Script	A script defining the conditions needed to spend the output

TRANSACTION INPUT

- Transaction inputs identify *by reference* which UTXO to consume
- Provide a proof of ownership through an *unlocking script*
- Each transaction input is signed independently
  - Multiple parties can collaborate to construct transactions and sign only one input each
  - Allows mixing schemes (CoinJoin, etc)

TRANSACTION INPUT SERIALIZATION

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent; first one is 0
1–9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script
4 bytes	Sequence Number	Used for locktime or disabled (0xFFFFFFFF)

NEW SEGWIT TX SERIALIZATION

Field Size	Name	Type	Description
4	version	int32_t	Transaction data format version
1	marker	char	Must be zero
1	flag	char	Must be nonzero
1+	txin_count	var_int	Number of transaction inputs
41+	txins	txin[]	A list of one or more transaction inputs
1+	txout_count	var_int	Number of transaction outputs
9+	txouts	txouts[]	A list of one or more transaction outputs
1+	script_witnesses	script_witnesses[]	The witness structure as a serialized byte array
4	lock_time	uint32_t	The block number or timestamp until which the transaction is locked

GETRAWTRANSACTION

```
$ getrawtransaction txid true

{
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": "3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1decc6498c75c4ae24cbb9f039ff08df[…]",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "scriptPubKey": "OP_DUP OP_HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value": 0.08450000,
      "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG",
    }
  ]
}
```

BUILDING THE CONTEXT

```
$ getrawtransaction referencedTxID true

"vout": [
  {
    "value": 0.10000000,
    "scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8a[…] OP_EQUALVERIFY  OP_CHECKSIG"
  }
]
```

RECAP

- Transactions create a chain of ownership (no fungibility, anonymity)

- Inputs
  - Unlock the bitcoins of a previous *unspent* output
  - Provide a proof of coin ownership (signature)
  - Need to build the context - additional call to get the prevout script and amount
  - Stored in the blockchain
  - Segwit moves the signatures outside of inputs

- Outputs
  - Lock bitcoins to a new owner
  - The owner is a hash/address, can be a person or a script
  - Stored in the blockchain + UTXO also in the UTXO-set database

- Fees are implied (inputs - outputs)

2

SIGNING AND VERIFICATION

SIGNATURE

- Value produced from a private key and a message
- Someone can prove ownership of a private key by comparing the signature against a related public key
- Used in Bitcoin to authorize spending satoshis previously sent to a public key hash

SIGNING A TRANSACTION

- To spend funds we need to prove *private key ownership* without revealing it

- Bitcoin uses ECDSA digital signature to authorize transactions

- The signature proves that
  - The owner of the privKey has authorized the spending (*authentication*)
  - The proof of authorization is undeniable (*non-repudation*)
  - The transaction have not and cannot be modified in transit (*integrity*)

COMMITMENT HASH AND SIGHASH FLAGS

- Every signature has a SIGHASH flag and it can be different from input to input

- Sig = privKey.sign($\overbrace{\text{double-SHA256(serialized tx data + SIGHASH)}}^{\text{Commitment Hash}}$)
  - The signature commits to specific transaction data
  - The signature commits the SIGHASH type as well, so it can’t be changed (e.g. by a miner)
  - Signature scripts are not signed, anyone can modify them (tx malleability)

- The SIGHASH flag is appended to the signature
  - [<DER signature> <1 byte hash-type>]
  - Indicates which part of a transaction’s data is included in the commitment hash

- Many of the SIGHASH flags only make sense if you think of multiple participants *collaborating* outside the bitcoin network and updating a *partially signed* transaction

- Since the signature protects those parts of the transaction from modification, this lets signers *selectively choose to let other people modify their transactions*

- A single-input transaction signed with NONE could have its output changed by the miner who adds it to the blockchain
- A two-input transaction has one input signed with NONE and one input signed with ALL, the ALL signer can choose where to spend the satoshis without consulting the NONE signer — but nobody else can modify the transaction

SIGHASH flag	Value	Description
ALL	0x01	Signature applies to all inputs and outputs, protecting everything except the signature scripts against modification
NONE	0x02	Signature applies to all inputs, none of the outputs
SINGLE	0x03	Signature applies to all inputs but only the one output with the same index number as the signed input
ALL|ANYONECANPAY	0x81	Signature applies to one input and all outputs
NONE|ANYONECANPAY	0x82	Signature applies to one input, none of the outputs
SINGLE|ANYONECANPAY	0x83	Signature applies to one input and the output with the same index number

SIGNATURE VERIFICATION

- The signature verification algorithm takes
  - the signature
  - the commitment hash
  - the signer’s public key
- There are 4 ECDSA signature verification codes in the original Bitcoin script system
  - CHECKSIG
  - CHECKSIGVERIFY
  - CHECKMULTISIG
  - CHECKMULTISIGVERIFY

NEW COMMITMENT HASH ALGORITHM

- New transaction digest algorithm for signature verification for version 0 witness program (BIP-143)
- Quadratic hashing problem: now minimize redundant data hashing in verification
- Now the signature covers the amount of BTC being spent by the input

RECAP

- ECDSA signatures are used to authorized a transaction
- The signature is verified against the commitment hash with the signer’s public key
- Signers can selectively choose which part of the tx to sign, using sighash flags
- Segwit fixes long standing issues on the signature verification algorithm

3

TRANSACTION TYPES

BITCOIN SCRIPT

- Bitcoin transaction validation is not based on a static pattern, but achieved through the execution of a scripting language

- Scripts and script data are spread out over
  - Unlocking script (scriptSig)
  - Locking script (scriptPubkey)
  - Witness (txinwitness)

- Unlocking script and witness have to satisfy the locking script conditions

- Reverse-polish notation
  - 3 2 +
- Stack-based execution language
- Turing Incompleteness
- Stateless
  - no state prior to execution of the script, or saved after execution of the script
  - all the information needed to execute a script is contained within script fields (scriptSig, scriptPubkey, witness)
- Predictable (execute the same way on any system)

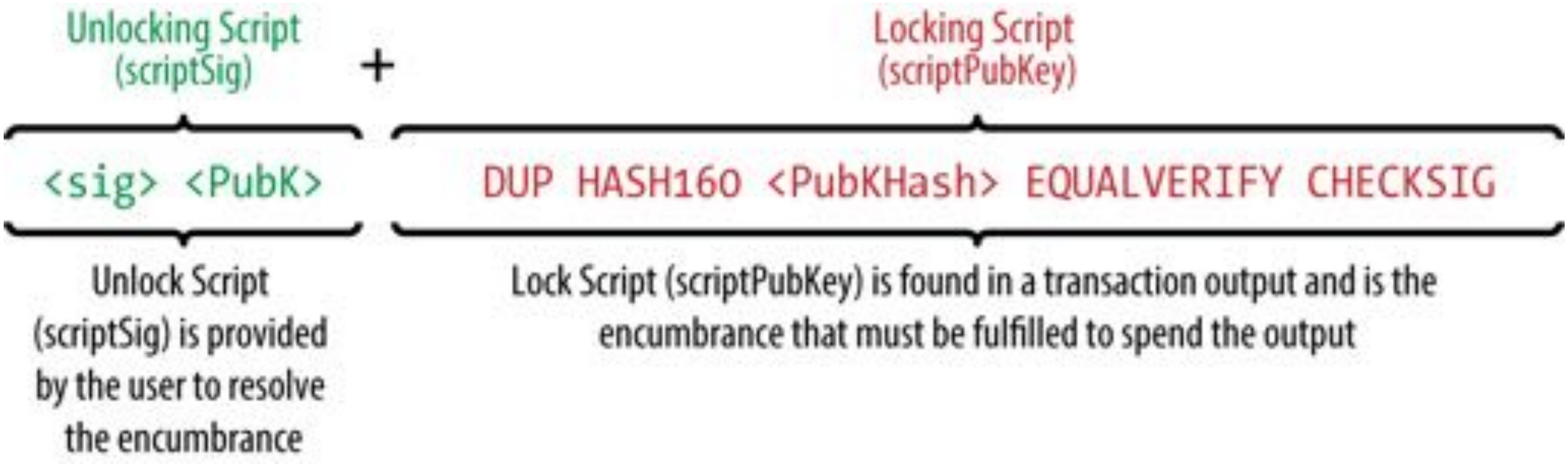
TWO TX CATEGORIES

- Locking the funds with a public key hash
  - P2PKH
  - P2SH-P2WPKH
  - P2WPKH

- Locking the funds with a redeem script hash
  - P2SH
  - P2SH-P2WSH
  - P2WSH

P2PKH TRANSACTION

- Most common type of bitcoin transaction until Segwit
- The output contains a locking script that locks the coins to a public key hash
- A public key and a digital signature are necessary to unlock a P2PKH script
- Now deprecated in favor of Segwit transactions


Unlocking Script
(scriptSig)
+
Locking Script
(scriptPubKey)
<sig> <PubK>
DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
Unlock Script (scriptSig) is provided by the user to resolve the encumbrance
Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

P2SH TRANSACTION

- Move the logic to a new *redeem script* field
  - 2 pubKey1 pubKey2 pubKey3 3 CHECKMULTISIG
- Locking script locks the output to the hash of the redeem script
  - RIPEMD160(SHA256(redeemScript))
  - HASH160 <20-byte hash of redeem script> EQUAL
- Unlocking script contains the redeem script
  - Sig1 Sig2 <redeem script>
- P2SH address is the Base58Check encoding of the script hash
  - [one-byte version][20-byte hash][4-byte checksum]

BENEFITS OF P2SH

- Complex scripts are replaced by shorter fingerprints in the transaction output. Smaller tx, less fees

- P2SH shifts the transaction fee cost of a long script from the sender to the recipient, who has to include the long redeem script to spend it

- P2SH shifts the burden of constructing the script to the recipient, not the sender

- Scripts can be encoded as a bitcoin address. Locking money with it is simple. But beware of having the corresponding redeem script

- P2SH shifts the burden in data storage for the long redeem script from the output to the input, so not in UTXO-set

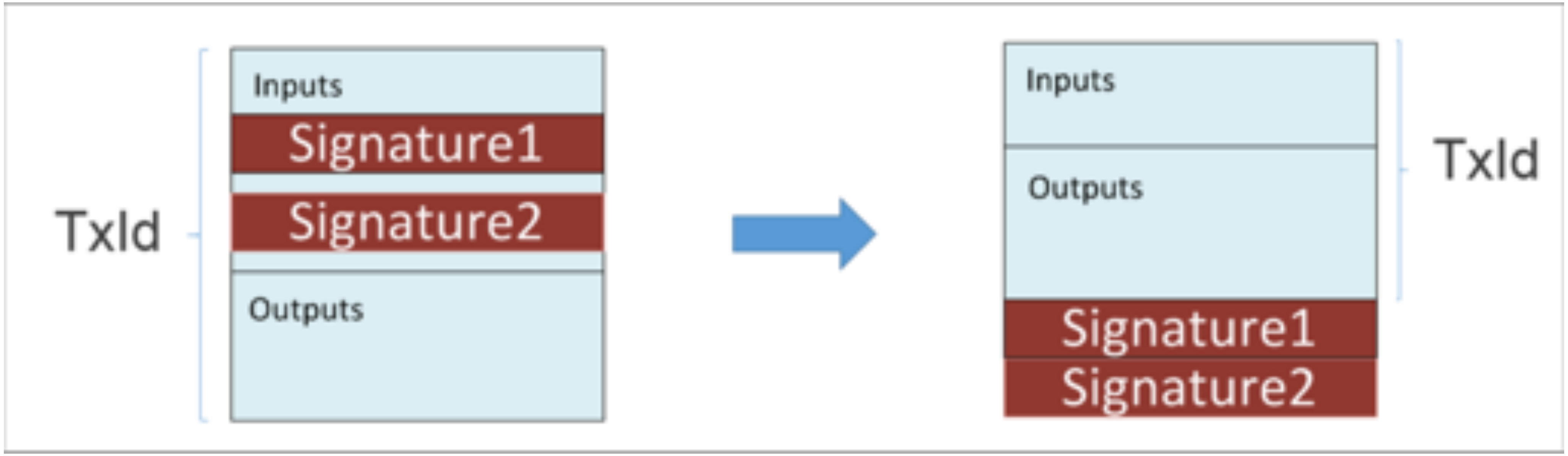
- P2SH shifts the burden in data storage for the long script from the present time (locking of funds) to a future time (spending of funds)

SEGREGATED WITNESS

- Signatures are only required by fully validating nodes, only at validation time
- Signatures accounts for around 60% of the blockchain
- A new structure called a "witness" is committed to blocks separately from the transaction merkle tree
- This structure contains data required to check transaction validity, in particular scripts and signatures

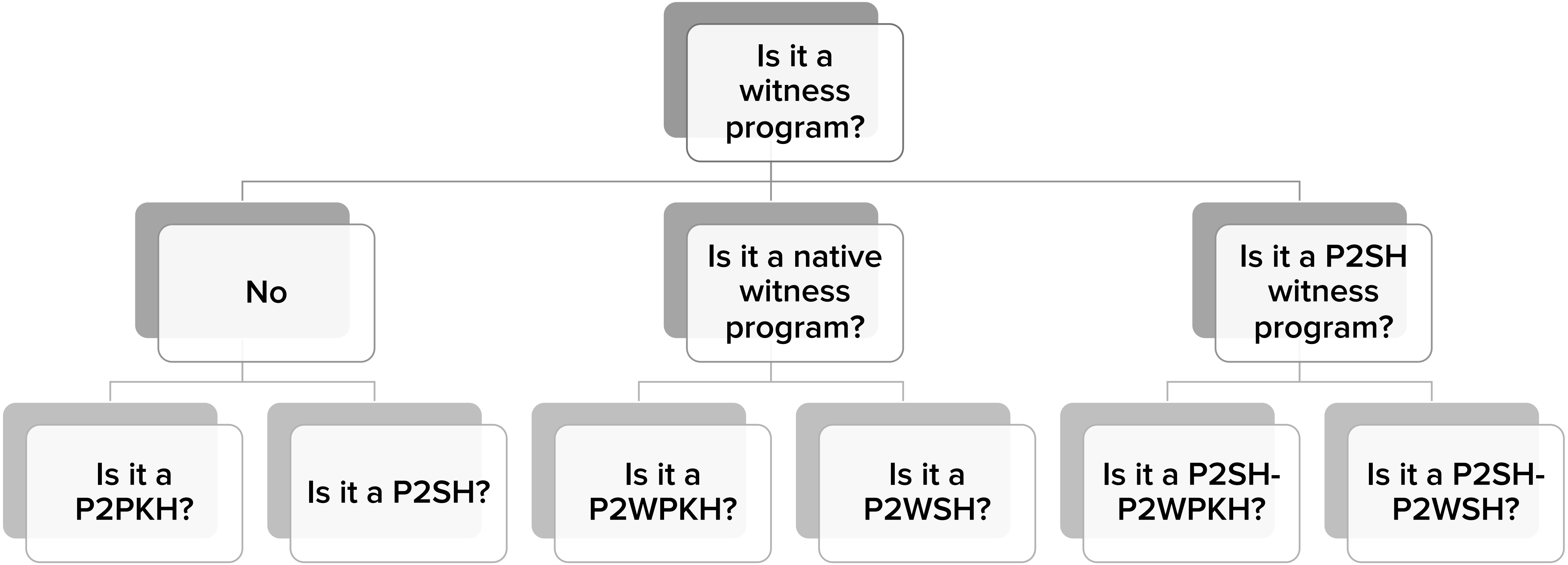
TRANSACTION MALLEABILITY

- Transactions which only spend segwit outputs are not vulnerable to malleability



WITNESS VALIDATION LOGIC

- Versioned witness program triggers witness validation logic
  - <version byte> <witness program>
- Located in scriptPubkey in native witness programs
- Located in scriptSig, as a unique stack item, in P2SH witness programs


Is it a witness program?
No
Is it a native witness program?
Is it a P2SH witness program?
Is it a P2PKH?
Is it a P2SH?
Is it a P2WPKH?
Is it a P2WSH?
Is it a P2SH-P2WPKH?
Is it a P2SH-P2WSH?

NATIVE V.0 WITNESS PROGRAMS

- scriptSig is empty
- scriptPubKey is a versioned witness program
  - Version byte 0 + witness program
- Witness
  - <signature> <pubkey> (P2WPKH)
  - data + witnessScript (P2WSH)

- P2WPKH program
  - 20-byte witness program must match pubKey’s HASH160
  - pubKey’s HASH160 and CHECKSIG are done automatically

- P2WSH program
  - 32-byte witness program must match witnessScript’s SHA256
  - witnessScript’s SHA256 and comparison is done automatically
  - The redeem script moved to witness and called witnessScript

NATIVE P2WPKH LOCKING SCRIPT

P2PKH

```
OP_DUP OP_HASH160 0067c8970e65107ffbb436a49edd8cb8eb6b567f OP_EQUALVERIFY OP_CHECKSIG
```

P2WPKH

```
0 0067c8970e65107ffbb436a49edd8cb8eb6b567f
```

Witness version
/ Version byte

20-bytes witness program

P2SH V.0 WITNESS PROGRAMS

- scriptPubkey is a standard P2SH script
  - Hash160 of the scriptSig (asm)

- scriptSig is a versioned witness program
  - VWP pushed onto the stack as a single stack item
  - P2SH-P2WPKH: 20-byte witness program must match pubKey's HASH160
  - P2SH-P2WSH: 32-byte witness program must match witnessScript's SHA256

- Witness
  - <signature> <pubkey> (P2SH-P2WPKH)
  - data + witnessScript (P2SH-P2WSH)

RECAP

- P2PKH
  - <unlocking script> <locking script>

- P2SH
  - <unlocking script <redeem script>> <locking script>

- P2WPKH
  - <empty unlocking script> <locking script (version + program)>
  - Trigger <witness>

- P2WSH
  - <empty unlocking script> <locking script (version + program)>
  - Trigger <witness>

- P2SH-P2WPKH
  - <unlocking script (version + program)> <locking script>
  - Trigger <witness>
- P2SH-P2WSH
  - <unlocking script (version + program)> <locking script>
  - Trigger <witness>

4 WALLET ABSTRACTIONS

WHAT WALLETS DO FOR YOU?

- Monitor incoming tx
  - detects that a UTXO can be spent with one of the keys it controls

- Construct outgoing tx
  - selects UTXOs
  - calculates and include tx fees
  - creates one input per UTXO
    - retrieves the referenced UTXO, examines its locking script, then uses it to build the necessary unlocking script to satisfy it

- Create balance
  - sums all UTXO that user’s wallet can spend

Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

PKH in locking scripts of tx outputs, encoded in Base58Check

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

PKH in locking script of the first output of the previous tx referenced in input, encoded in Base58Check

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

97 Confirmations 0.0995 BTC

Summary	
Size	258 (bytes)
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 +9 minutes)

Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Estimated BTC Transacted	0.015 BTC

Sum(Inputs) – Sum(Outputs)

Bitcoin Address

Addresses are identifiers which you use to send bitcoins to another person.

Summary	
Address	1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA
Hash 160	ab68025513c3dbd2f7b92a94e0581f5d50f654e7
Tools	Taint Analysis - Related Tags - Unspent Outputs

Transactions	
No. Transactions	25
Total Received	0.17579525 BTC
Final Balance	0.17579525 BTC

Sum of all outputs with this PKH in the blockchain

Sum of all UTXO referencing this PKH
(current spending balance)

CONCLUSION

- UTXO-based model
- Inputs contain a reference to an output we are spending and the data necessary to unlock the coins
  - Signatures
  - Secrets, etc.
- Outputs contain a smart contract that lock coins
  - To a public key hash
  - To a smart contract (redeem script)