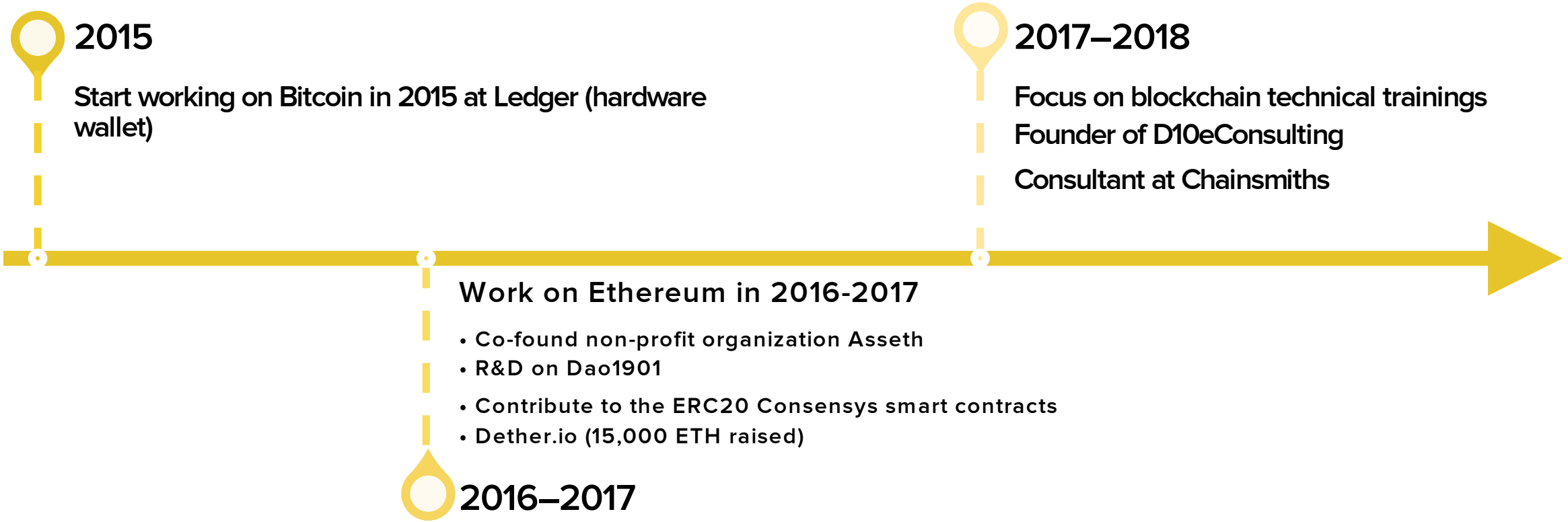
All About Bitcoin Transaction

Stéphane Roche

ABOUT STEPHANE



OUTLINE

1. Structure
2. Signing And Verification
3. Transaction Types
4. Wallet Abstractions

1

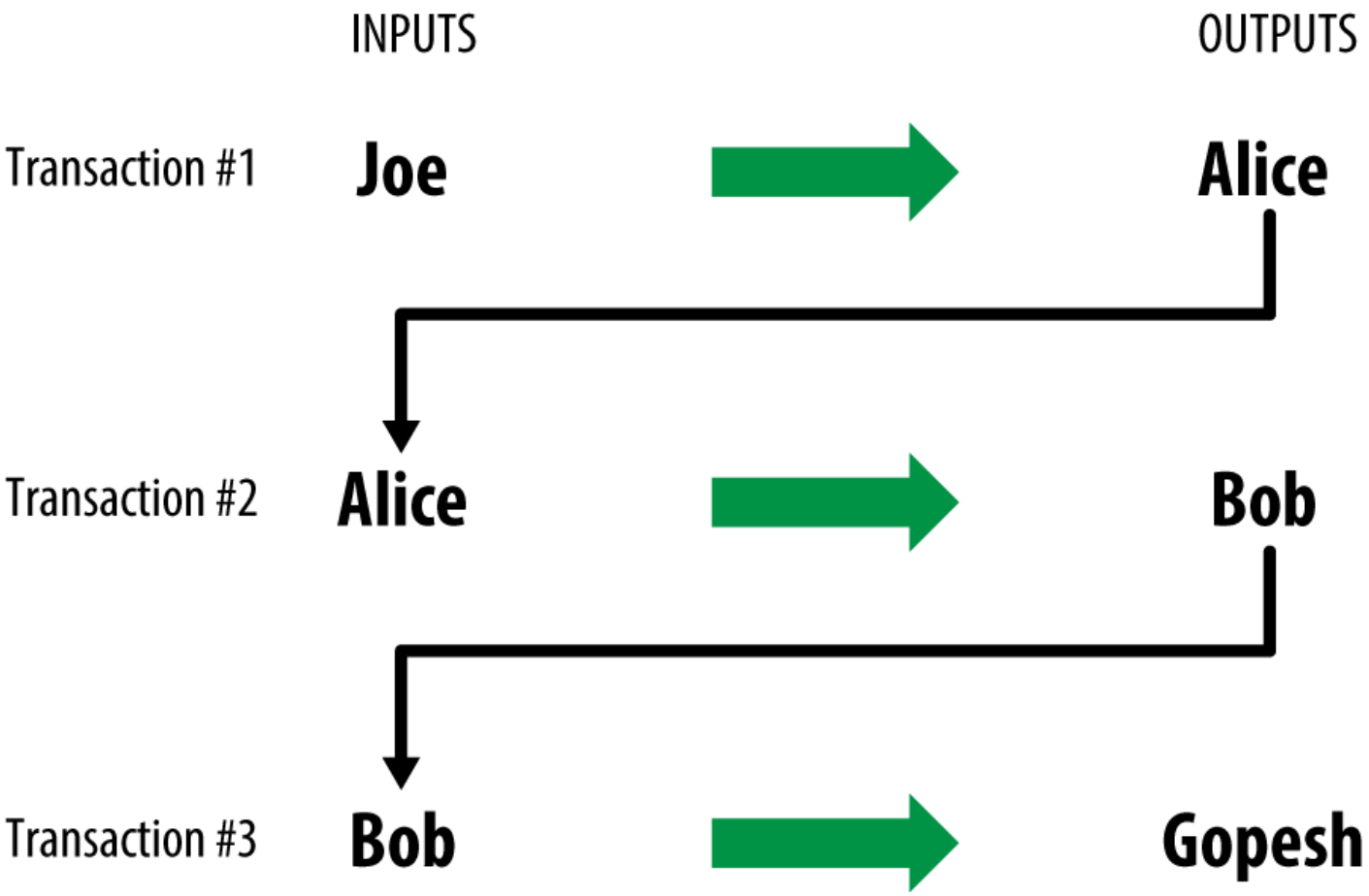
STRUCTURE

Transaction as Double-Entry Bookkeeping

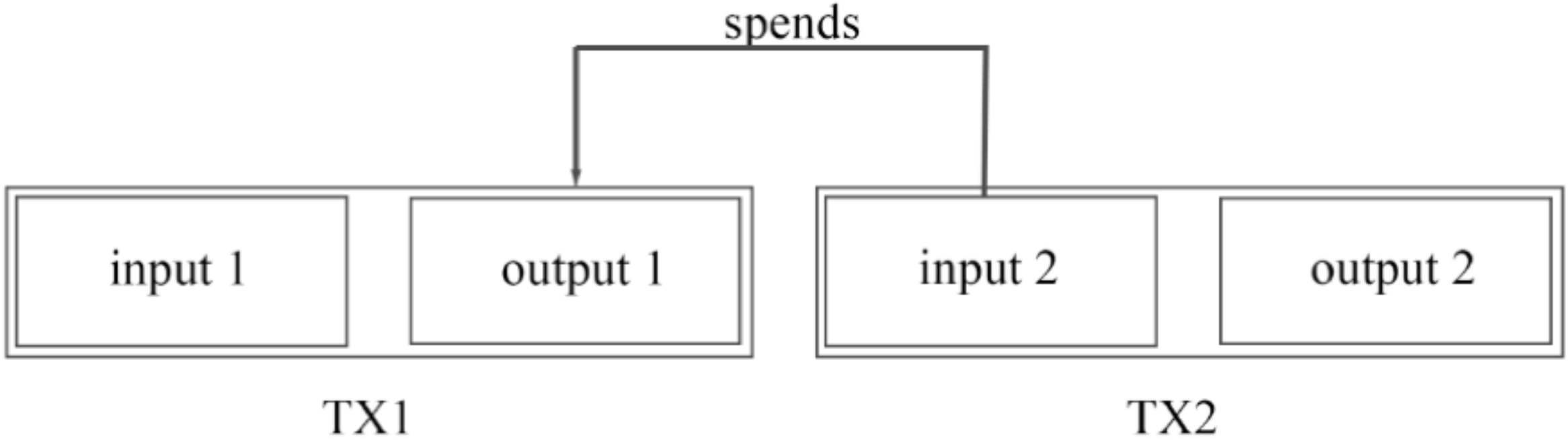
Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC

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

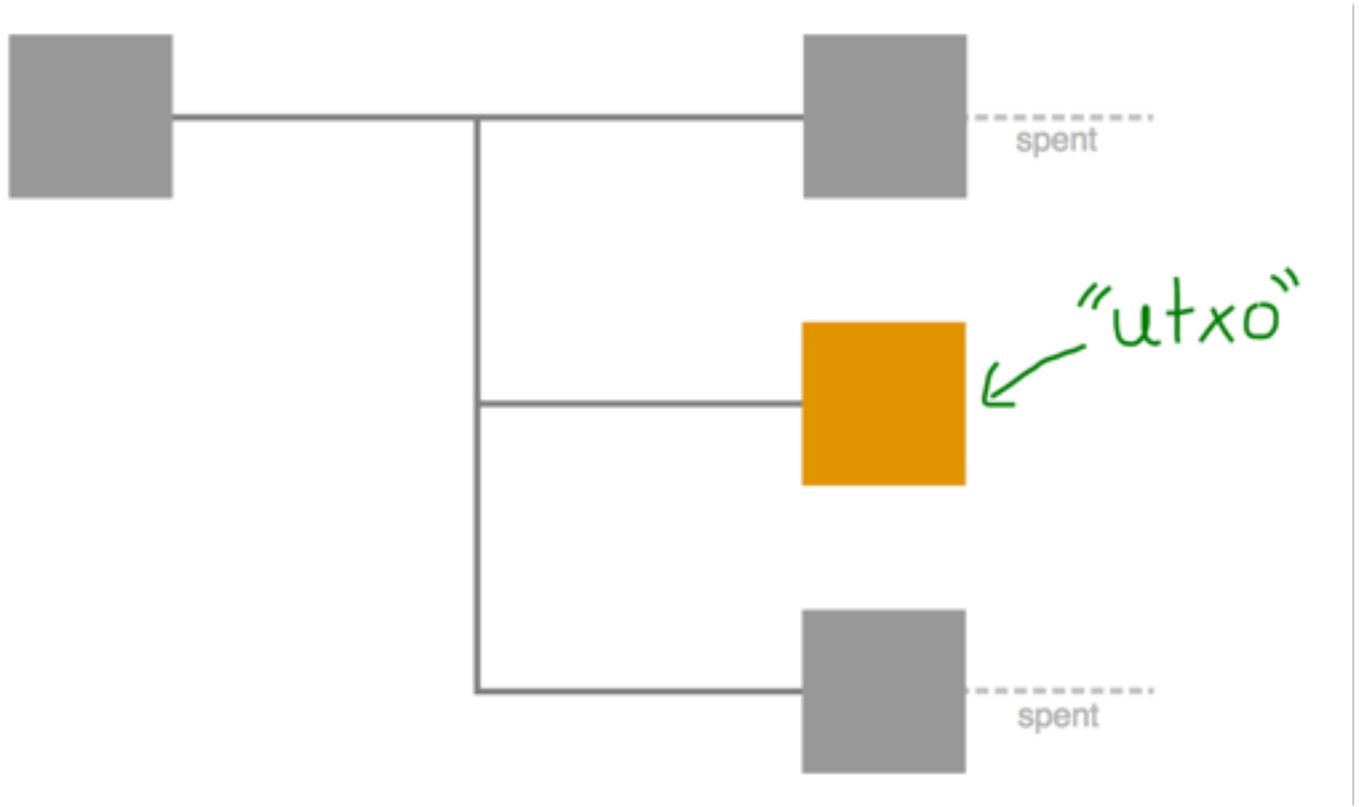
TRANSACTION CHAINS

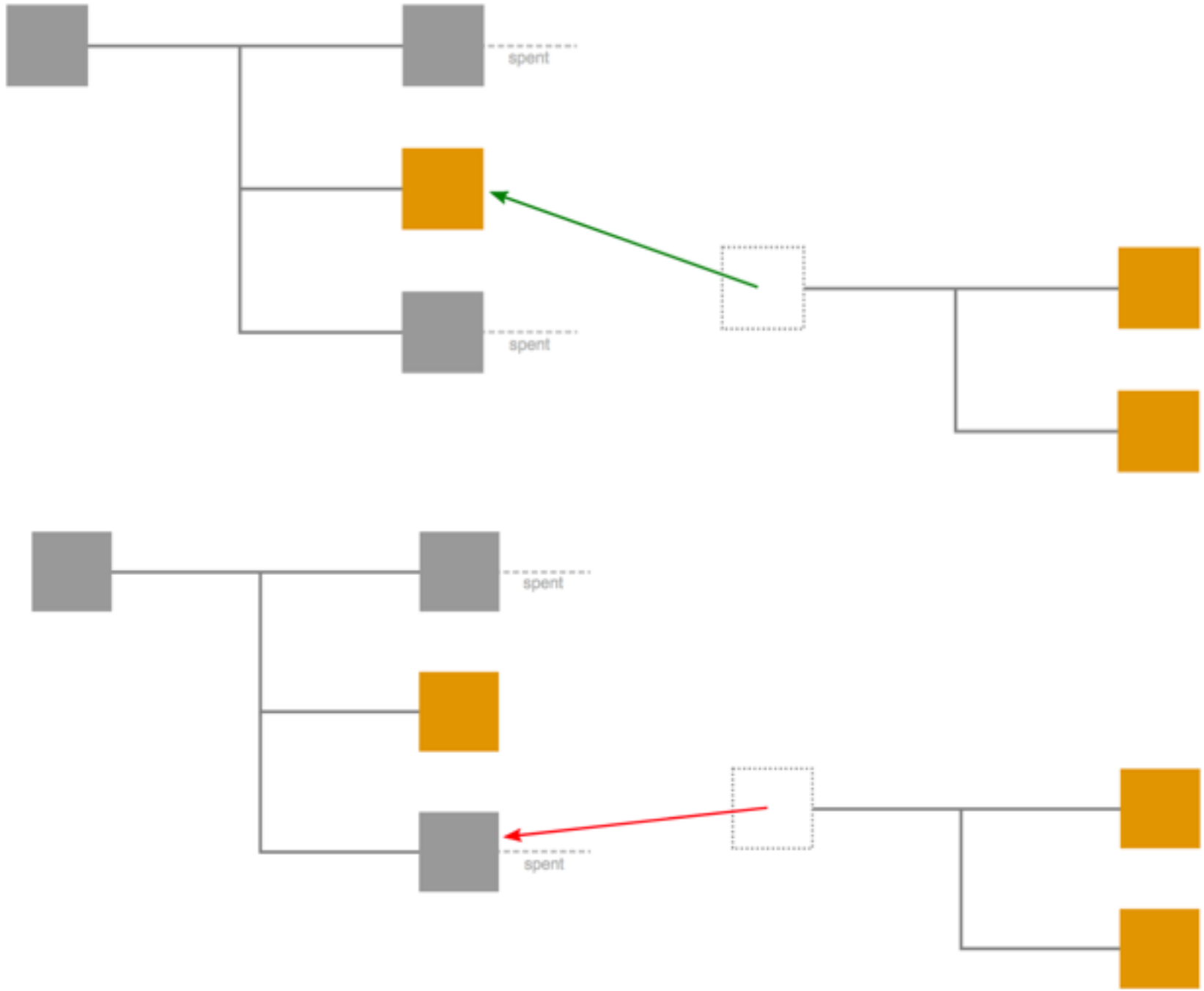


INPUT/OUTPUT CHAIN



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

Commitment Hash

- Sig = privKey.sign(double-SHA256(serialized tx data + SIGHASH))
  - 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 instead is achieved through the execution of a scripting language
- This language allows for a nearly infinite variety of conditions to be expressed
- Bitcoin’s transaction validation engine relies on two types of scripts to validate transactions: locking script and unlocking script
- The input is valid if the unlocking script satisfies the locking script conditions

- Forth-like
- Reverse-polish notation
- 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 the script
- Predictable (execute the same way on any system)

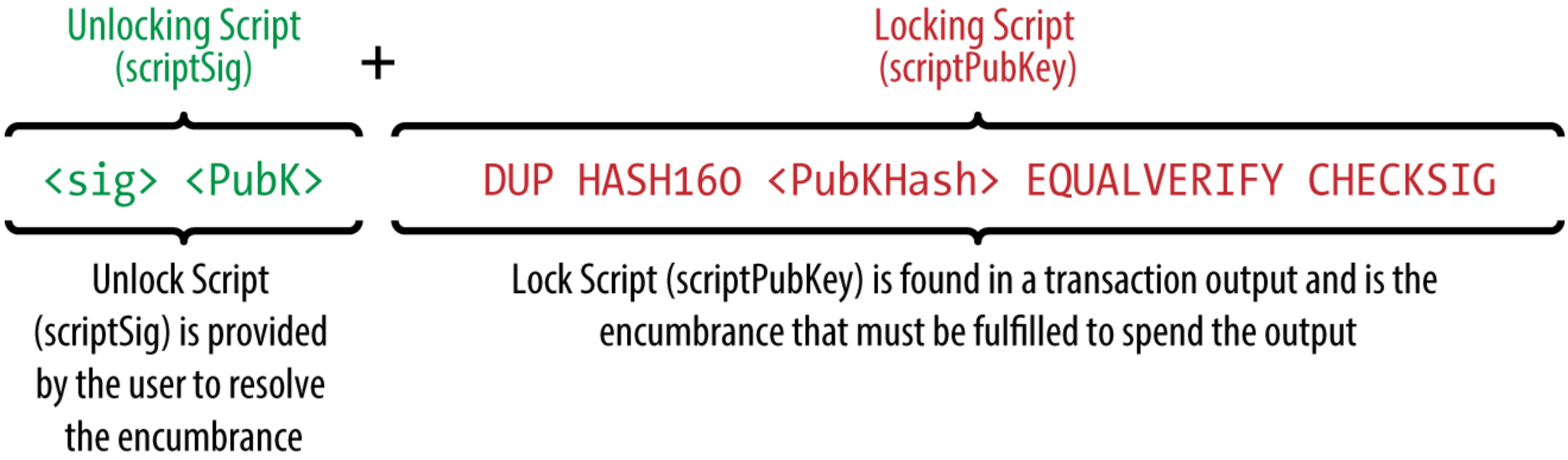
TWO TYPES, DIFFERENT WAYS

- 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
  - HASH160 <20-byte hash of redeem script> EQUAL
- Unlocking script contains the redeem script
  - Sig1 Sig2 <redeem script>
- Base58Check encoded address
  - [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

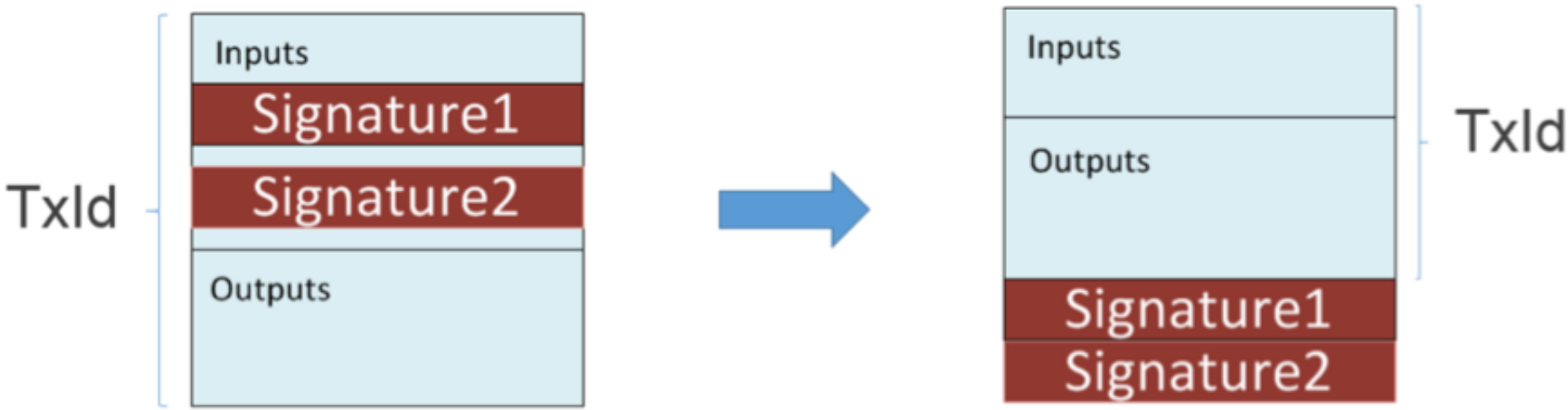
- Scripts can be encoded as a bitcoin address. Locking money with it is simple. But beware of having the corresponding redeem script

- 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


TxId
Inputs
Signature1
Signature2
Outputs
Inputs
Outputs
Signature1
Signature2
TxId

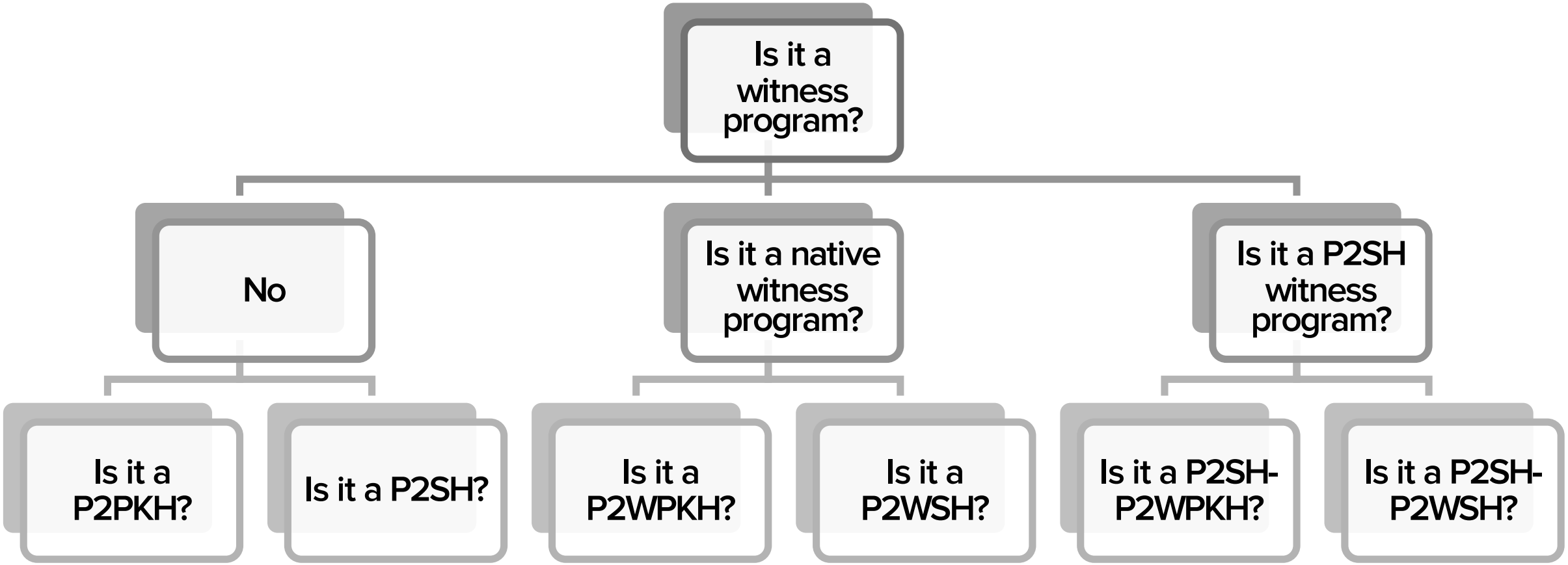
WITNESS VALIDATION LOGIC

- A locking or redeem script that consists of a 1-byte push opcode followed by a data push between 2 and 40 bytes gets a *new special meaning*

- <Version byte + witness program> either in scriptPubKey or scriptSig

- Native witness program
  - scriptPubKey == <version byte> <witness program>
  - scriptSig MUST be *empty* or validation fails

- P2SH witness program
  - scriptSig == <version byte> <witness program>


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 WITNESS PROGRAM (P2WPKH / P2WSH)

- Unlocking script is empty
- Locking script composed of “version + program”
- Witness is either
  - P2WPKH: <sig> <pubKey>
  - P2WSH: <sig> <locking « redeem/witness » script>
- Bech32: address format proposed for native Segwit outputs

TWO TYPES OF NATIVE WITNESS PROGRAM

- If witness program is 20 bytes
  - Interpreted as a P2WPKH program
  - Witness must be <sig> <pubKey>
  - HASH160 of the public key must match the 20-byte witness program
  - The signature is verified against the public key with CHECKSIG operation

- If witness program is 32 bytes
  - Interpreted as a P2WSH program
  - SHA256 of the witnessScript must match the 32-byte witness program
  - The witness stack is executed

NATIVE P2WPKH LOCKING SCRIPT

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

To

```
0 0067c8970e65107ffbb436a49edd8cb8eb6b567f
```

Witness version
/ Version byte

20-bytes witness program

NATIVE P2WPKH V.0 EXAMPLE

- ScriptPubKey: 0 <20-byte-key-hash>
- ScriptSig: empty
- Witness: <signature> <pubKey>

- HASH160 of the pubKey and CHECKSIG are done automatically by the validation engine

NATIVE P2WSH V.0 MULTISIG EXAMPLE

- ScriptPubKey: 0 <32-byte-hash>
- ScriptSig: empty
- Witness: 0 <signature1> <1 <pubkey1> <pubkey2> 2 CHECKMULTISIG>

- The spending script is same as the one for P2SH output but is moved to witness

- Hashing of the witnessScript with SHA256 and comparison is done automatically

P2SH WITNESS PROGRAM

- ScriptPubKey: a regular P2SH script
- ScriptSig: <version byte> <witness program>
  - This special redeem script will trigger the witness

- Witness program nested in a regular P2SH

- Used for smooth transition, wallet compatibility

EXAMPLE OF P2WPKH NESTED IN P2SH (P2SH-P2WPKH)

- ScriptPubKey: HASH160 <20-byte-script-hash> EQUAL
- ScriptSig: <0 <20-byte-key-hash>>
- Witness: <signature> <pubkey>

- ScriptSig is hashed with HASH160 and compared with the 20-byte hash
- The public key and signature are then verified (like a P2WPKH, verified as <signature> <pubkey> CHECKSIG)

EXAMPLE OF P2WSH NESTED IN P2SH (P2SH-P2WSH)

- ScriptPubKey: HASH160 <20-byte-hash> EQUAL
- ScriptSig: <0 <32-byte-hash>>
- Witness: 0 <signature1> <1 <pubkey1> <pubkey2> 2 CHECKMULTISIG>

- ScriptSig is hashed with HASH160, compared against the 20-byte-hash in ScriptPubKey
- ScriptSig is interpreted as a P2WSH, triggers the witnessScript
- The P2WSH witnessScript is then executed

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

1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA - (Unspent) 0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK - (Unspent) 0.0845 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)

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

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

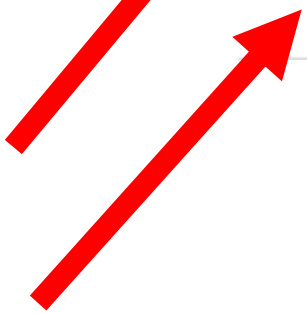
Sum(Inputs) – Sum(Outputs)

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

CONCLUSION

- UTXO-based blockchain
- Locks funds to addresses (person or script) with a script
- Flexible scripting language
- We need more Bitcoin developers, there is plenty of work!

BITCOIN FOR BEGINNERS

WHEN?

MAY, 2018

DURATION?

15h WEEKEND

OR

5 x 3h

WEEKDAY EVENINGS

WHERE?

"THE BLOCK CAFÉ"

RUA LATINO COELHO
63
1050-133 LISBON

PRICE

369€ including VAT

CONTACT

Stéphane Roche
rstephane@protonmail.com
+336 67 29 20 11
&
Armando Neves
hello@chainsmiths.com
+351 925 279 375

MINING

CRYPTOGRAPHY

BLOCKCHAIN

TRANSACTIONS

KEYS

CONSENSUS

P2P NETWORK

FORKS

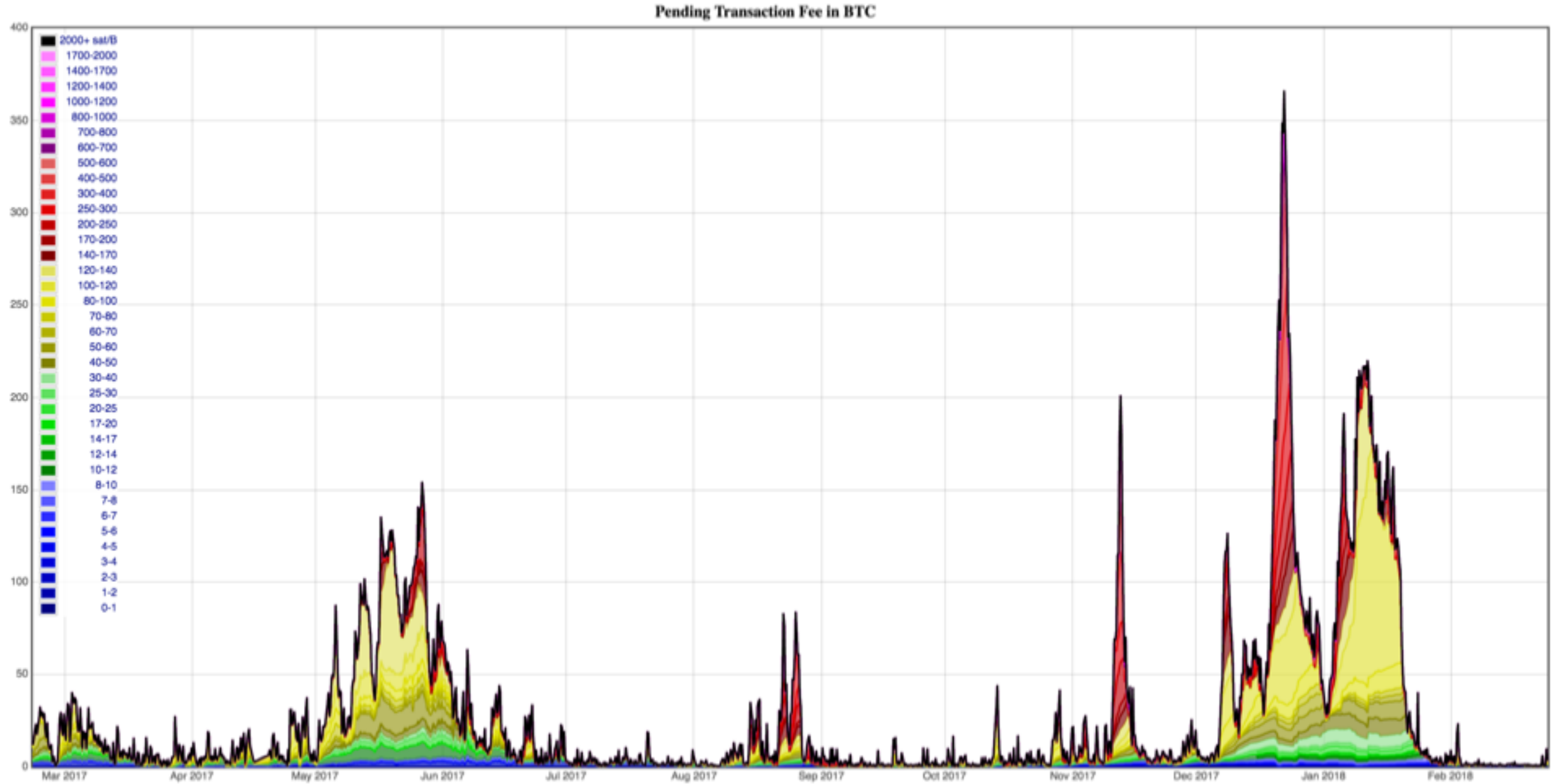
HISTORY

SCRIPTING

BLOCKS

COINBASE TRANSACTION

- First tx in a block
- Placed by the "winning" miner and creates brand-new bitcoin payable to that miner as a reward for mining
- Does not consume UTXO
- Special type of input called the "coinbase"


Pending Transaction Fee in BTC
2000+ sat/B
1700-2000
1400-1700
1200-1400
1000-1200
800-1000
700-800
600-700
500-600
400-500
300-400
250-300
200-250
170-200
140-170
120-140
100-120
80-100
70-80
60-70
50-60
40-50
30-40
25-30
20-25
17-20
14-17
12-14
10-12
8-10
7-8
6-7
5-6
4-5
3-4
2-3
1-2
0-1
400
350
300
250
200
150
100
50
0
Mar 2017
Apr 2017
May 2017
Jun 2017
Jul 2017
Aug 2017
Sep 2017
Oct 2017
Nov 2017
Dec 2017
Jan 2018
Feb 2018

DIGITAL SIGNATURES

Security services:

- Confidentiality: Information is kept secret from all but the authorized parties
- Message authentication: The sender of a message is authentic
- (Message) integrity: The message has not been modified during transmission
- Nonrepudiation: The sender of a message can not deny the creation of the message

Signing a message

```
var bitcoinPrivateKey = new BitcoinSecret("XXXXXXXXXXXXXXXXXXXXXXXXXX");

var message = "I am Craig Wright";

string signature = bitcoinPrivateKey.PrivateKey.SignMessage(message);

Console.WriteLine(signature); //
IN5v9+3HGW1q71OqQ1boSZTm0/DCiMpI8E4JB1nD67TCbIVMRk/e3KrTT9GvOuu3NGN0w8R2lWOV2cxnBp+Of8c=
```

Verifying a signature

```
var message = "I am Craig Wright";

var signature =
"IN5v9+3HGW1q71OqQ1boSZTm0/DCiMpI8E4JB1nD67TCbIVMRk/e3KrTT9GvOuu3NGN0w8R2lWOV2cxnBp+Of8c=";

var address = new BitcoinPubKeyAddress("1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa");

bool isCraigWrightSatoshi = address.VerifyMessage(message, signature);

Console.WriteLine("Is Craig Wright Satoshi? " + isCraigWrightSatoshi);
```

TRANSACTION ID

- Txid is the double SHA256 of the traditional serialization format
  - [nVersion][txins][txouts][nLockTime]

- Wtxid is the double SHA256 of the new serialization with witness data
  - [nVersion][marker][flag][txins][txouts][witness][nLockTime]

TRANSACTION SIZE

- Bytes

- Weight units
  - Proportion of the maximum block size
  - 1 weight unit = 1/4,000,000th of the maximum size of a block

- Virtual size (vsize, or vbytes)
  - Discount witness data by 75% for block size
  - 1 vbyte = 4 weight units
  - The maximum block size measured in vsize is 1 million vbytes
  - vsize of a non-segwit transaction is simply its size
  - Transaction fee should be estimated by comparing the vsize with other transactions, not the size

AVERAGE TRANSACTION SIZE

Inputs	Outputs	legacy	segwit(p2sh)	segwit(bech32)
1	1	192	134	110
1	2	226	166	141
2	1	339	225	178
2	2	372	257	209

GETRAWTRANSACTION


```
{
  "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 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18 true

"vout": [{ "value": 0.10000000,
"scriptPubKey": "OP_DUP OP_HASH160 7f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8 OP_EQUALVERIFY OP_CHECKSIG" }]

BECH32

- Segwit address format specified by BIP 173

- Only encodes witness (P2WPKH and P2WSH) scripts. Not compatible with non-segwit P2PKH or P2SH scripts

- A segwit address is a Bech32 encoding of
  - The human-readable part "bc" for mainnet, and "tb" for testnet
  - The separator « 1 »
  - The data-part values: checksum encoded witness script (which contains *keyhash* or *scripthash*)

- Several advantages over Base58Check
  - QR code is smaller
  - Detection of any error affecting at most 4 characters (BCH codes)
  - Error correction
  - Only consisting of lower cases, should be easier to type and read aloud
  - Excludes characters "1", "b", "i", and "o"
  - Increased security

TERMINOLOGY

- Transaction
- Signature
- Output, UTXO
- Input
- Outpoint (txid +vout index)
- Locking script (scriptPubKey)
- Unlocking script (scriptSig)
- Redeem Script
- ScriptCode
- Encumbrance
- Witness script
- Witness
- Cryptographic puzzle
- Opcode
- Signature hash flag (sighash)
- Vsize
- P2PKH, P2SH, P2WPKH, P2WSH, P2SH-P2W*