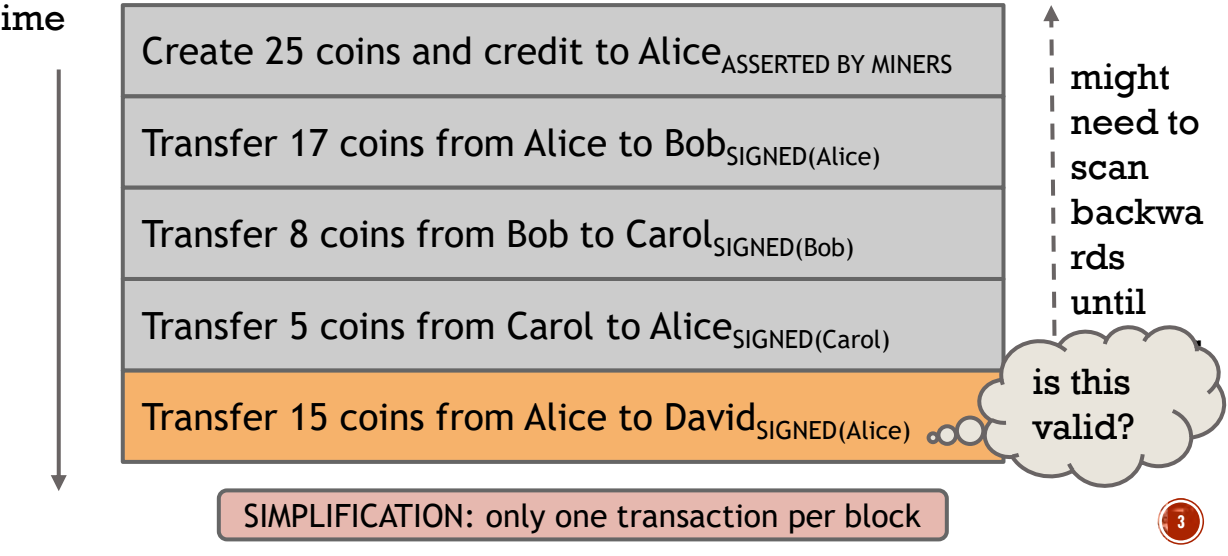
Blockchain Technologies

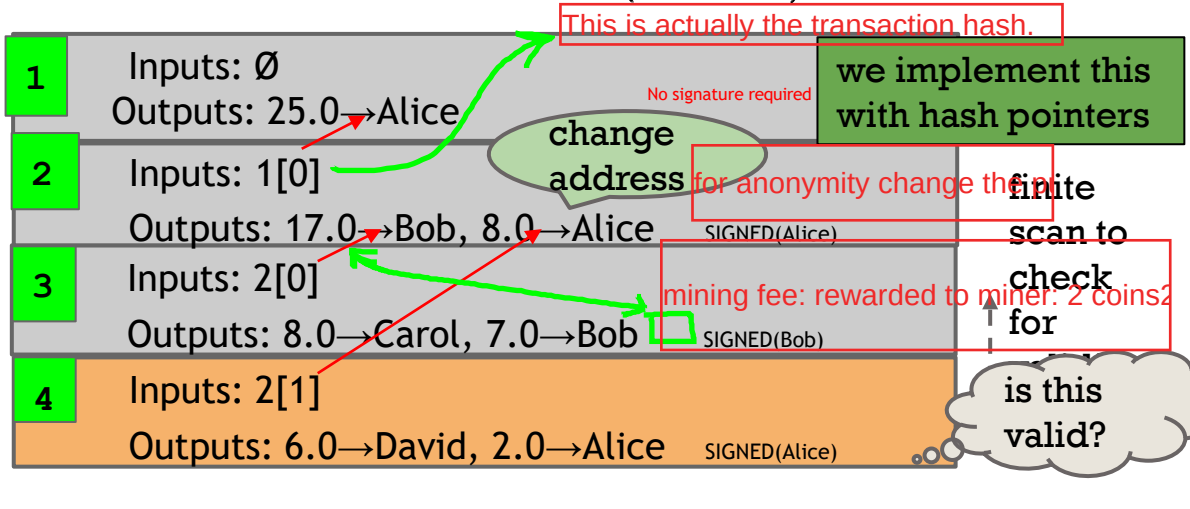
Mechanics of Bitcoin

BITCOIN TRANSACTIONS

AN ACCOUNT-BASED LEDGER (*NOT* BITCOIN)

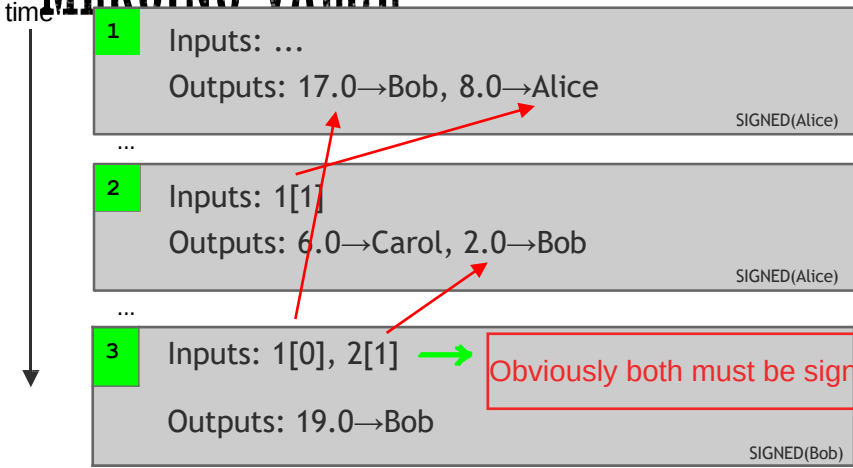


A TRANSACTION-BASED LEDGER (BITCOIN)



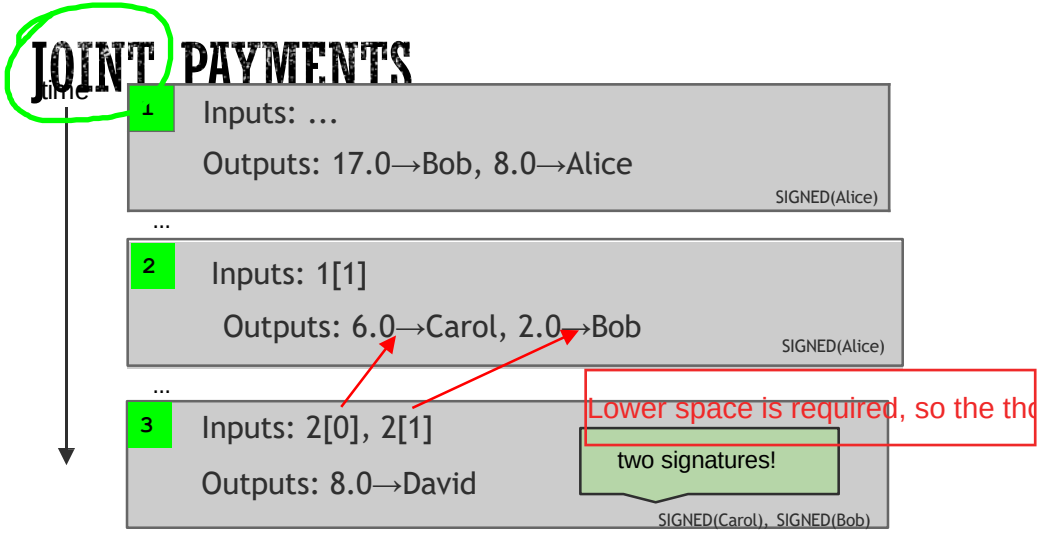
SIMPLIFICATION: only one transaction per block

MERGING VALUE

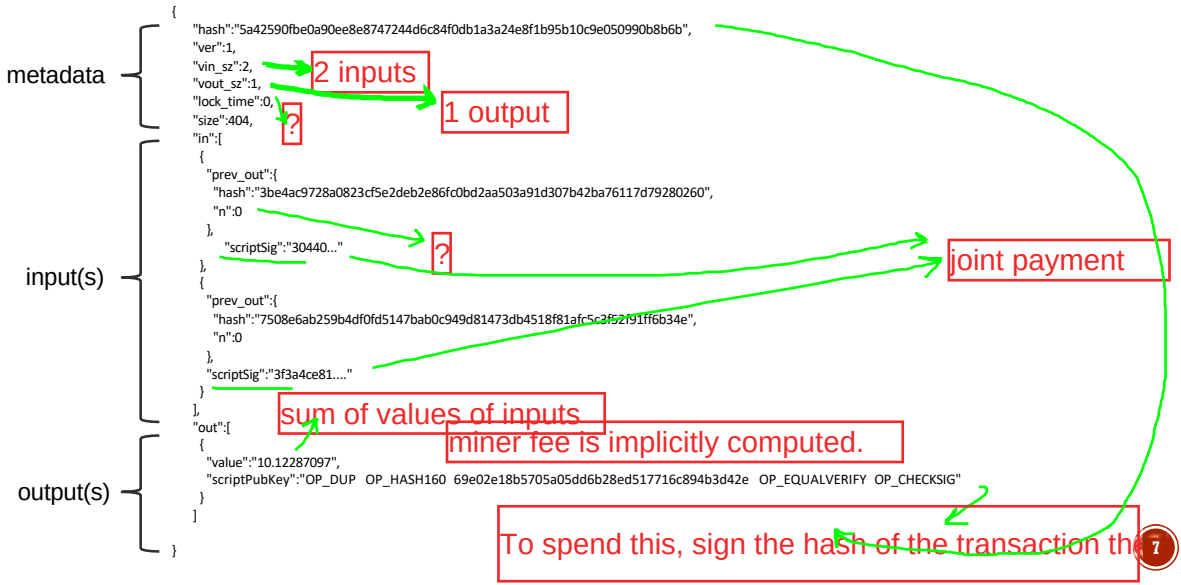


SIMPLIFICATION: only one transaction per block

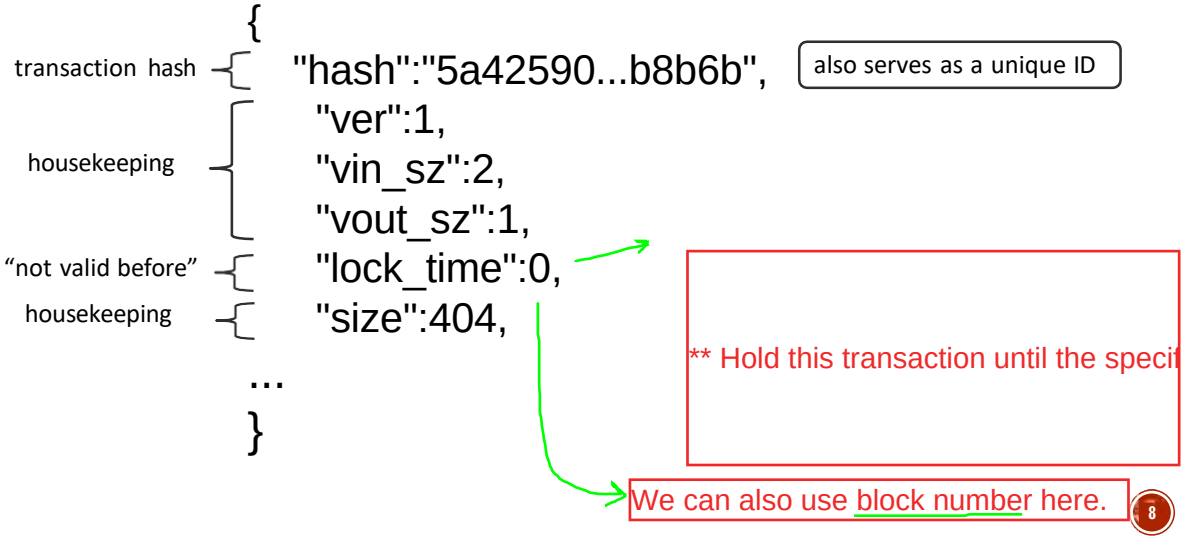
JOINT PAYMENTS



THE REAL DEAL: A BITCOIN TRANSACTION



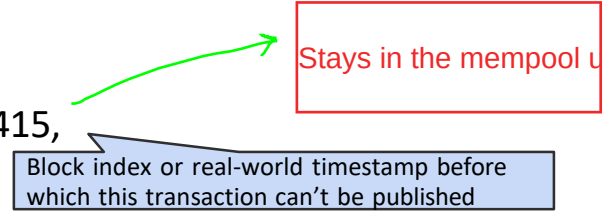
THE REAL DEAL: TRANSACTION METADATA



TIMELOCKS

lock_time

```
{
   "hash":"5a42590...b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":315415,
    "size":404,
...
}
```



THE REAL DEAL: TRANSACTION INPUTS

THE REAL DEAL: TRANSACTION OUTPUTS

output value

recipient address

(more outputs)

- "out":[
  - {
    - "value":"10.12287097",
    - "scriptPubKey":"OP_DUP OP_HASH160 69e...3d42e OP_EQUALVERIFY OP_CHECKSIG"
  - },
  ]

Sum of all output values less than or equal to sum of all input values!

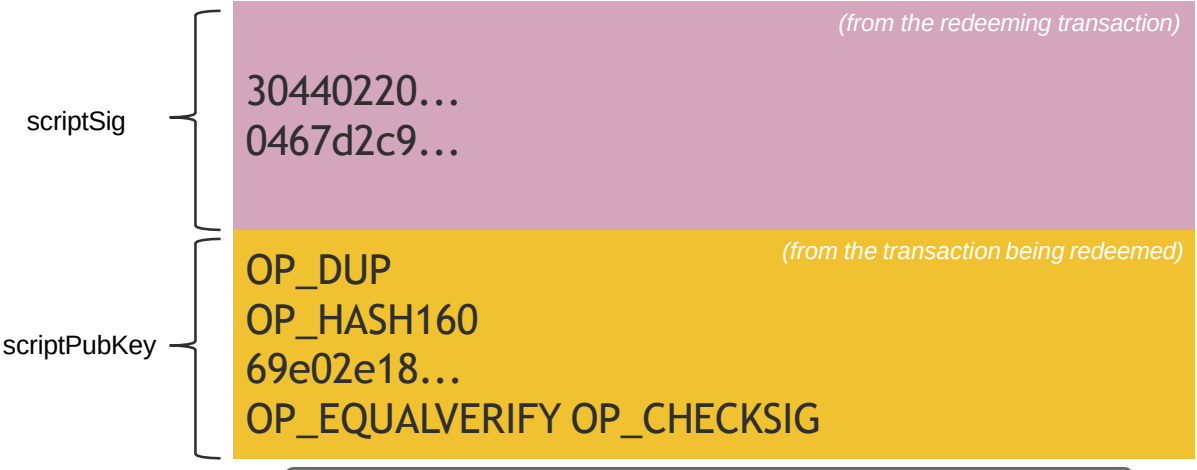
If sum of all output values less than sum of all input values, then difference goes to miner as a transaction fee

BITCOIN SCRIPT

OUTPUT "ADDRESSES" ARE REALLY *SCRIPTS*

```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

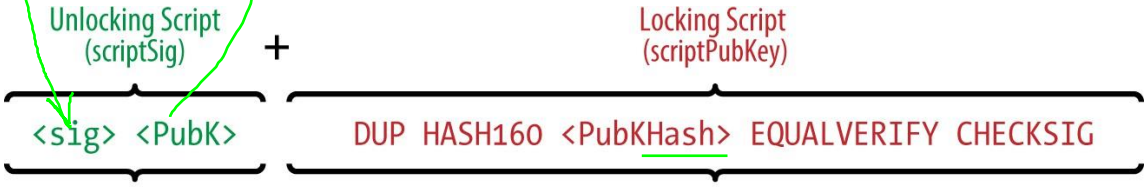
INPUT "ADDRESSES" ARE *ALSO* SCRIPTS



TO VERIFY: Concatenated script must execute completely with no errors

SCRIPT CONSTRUCTION (LOCK + UNLOCK)

The pubkey verifies the signature.



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

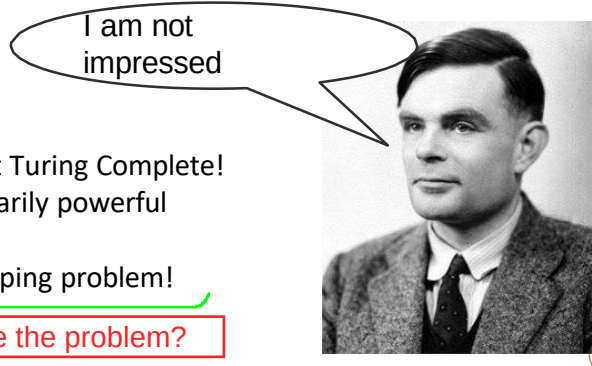
WHY SCRIPTS?

- Redeem previous transaction by signing with correct key
- "This can be redeemed by a signature from the owner of address X"
- Recall: address X is hash of public key
- What is public key associated with X?!
- "This can be redeemed by a public key that hashes to X, along with a signature from the owner of that public key"

BITCOIN SCRIPTING LANGUAGE ("Script")

- Design goals
  - Built for Bitcoin (inspired by Forth)
  - Simple, compact
  - Support for cryptography
  - Stack-based (linear)
  - Limits on time/memory
  - No looping
    - **Result**: Bitcoin script is not Turing Complete! i.e, cannot compute arbitrarily powerful functions
    - **Advantage**: No infinite looping problem!

ETH uses gas fee to mitigate the problem?



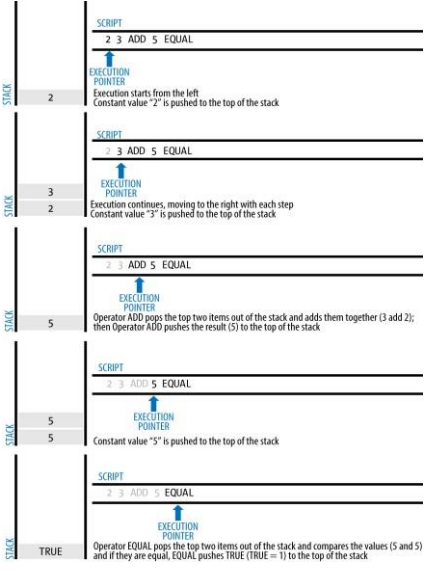
BITCOIN SCRIPTING LANGUAGE ("SCRIPT")

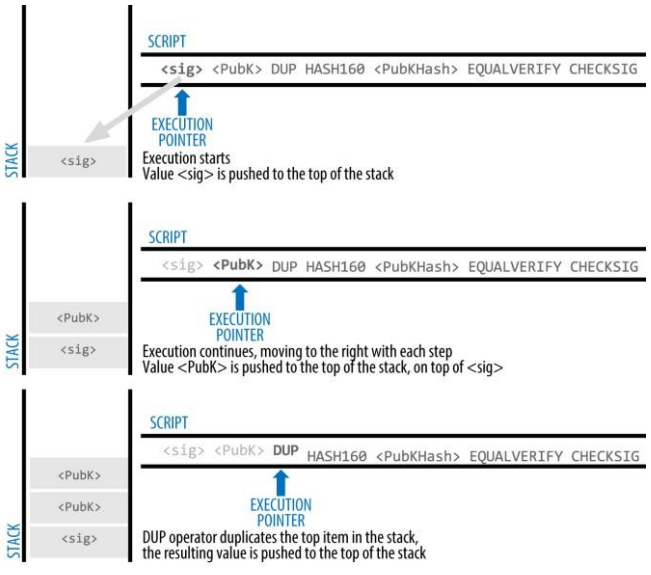
- 256 instructions (each represented by 1 byte)
  - 75 reserved, 15 disabled
  - Basic arithmetic, basic logic ("if" → "then"), throwing errors, returning early, crypto instructions (hash computations, signature verifications), etc.
- Only two possible outcomes of a Bitcoin script
  - Executes successfully with no errors → transaction is valid
  - OR Error while execution → transaction invalid and should not be accepted in the block chain

COMMON SCRIPT INSTRUCTIONS

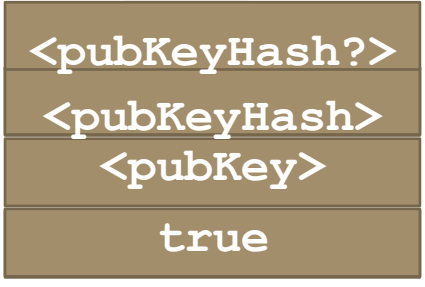
Name	Functions
OP_DUP	Duplicates top item on the stack
OP_HASH160	Hashes twice: first using SHA-256, then using RIPEMD-160
OP_EQUALVERIFY	Returns true if inputs are equal, false (marks transaction invalid) otherwise
OP_CHECKSIG	Checks that the input signature is valid using input public key for the hash of the current transaction
OP_CHECKMULTISIG	Checks that *t* signatures on the transaction are valid from *t (out of n)* of the specified public keys

SCRIPT: A STACK-BASED LANGUAGE





BITCOIN SCRIPT EXECUTION EXAMPLE



OP_CHECKMULTISIG

Implemented as a fork.

- Built-in support for joint signatures
- Specify *n* public keys
- Specify *t*
- Verification requires *t* signatures are valid

Other blockchains have

BUG ALERT: Extra data value popped from the stack and ignored

Fixing the bug required a fork. This is one of the reason the dev

MULTISIGNATURE

- The general form of a locking script setting an M-of-N multisignature condition is:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

- Example:

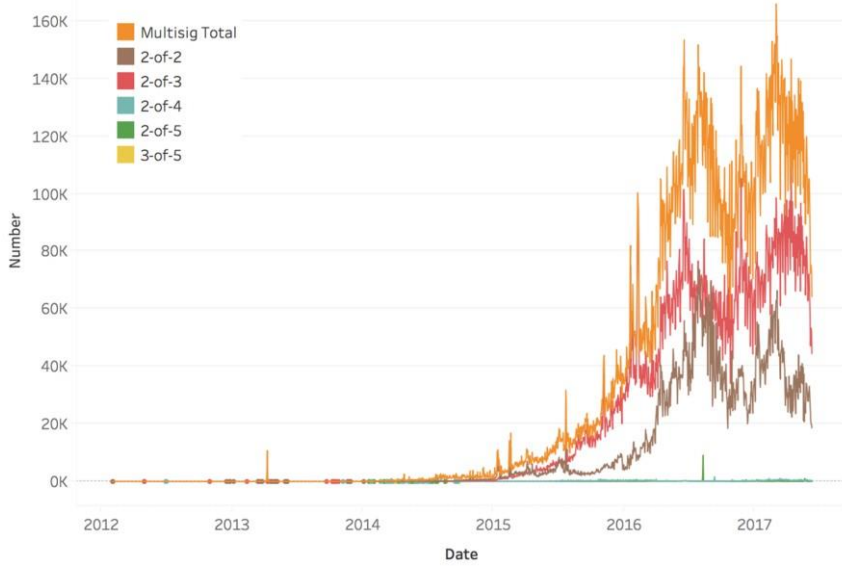
Locking: `2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG`

Unlocking: `<Signature B> <Signature C>`

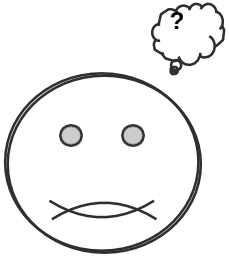
transaction fee is proportial to the size of the transaction. And also other i

Hash of this section is the redeemtion script.

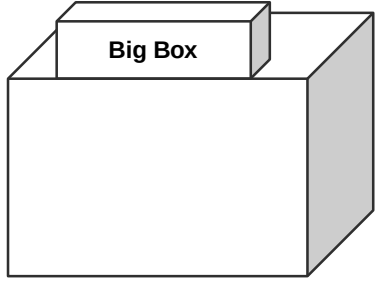
DIFFERENT MULTISIG VARIANTS



SHOULD SENDERS SPECIFY SCRIPTS?



I'm ready to pay for my purchases!



Cool! Well we're using MULTISIG now, so include a script requiring 2 of our 3 account managers to approve. Don't get any of those details wrong. Thanks for shopping at Big Box!

PAY-TO-SCRIPT-HASH (P2SH) WORKFLOW

This method improves t

- Bob
  1. creates a redeem script with whatever script he wants
  2. hashes the redeem script
  3. sends redeem script hash to Alice.
- Alice
  1. creates a P2SH-style output containing Bob's redeem script hash.
- When Bob wants to spend the output
  1. provides his signature along with the redeem script in the signature script.
- P2P network then
  1. ensures the redeem script hashes to the same value as the script hash Alice put in her output;
  2. then processes the redeem script exactly as it would if it were the primary pubkey script, letting Bob spend the output if the redeem script does not return false

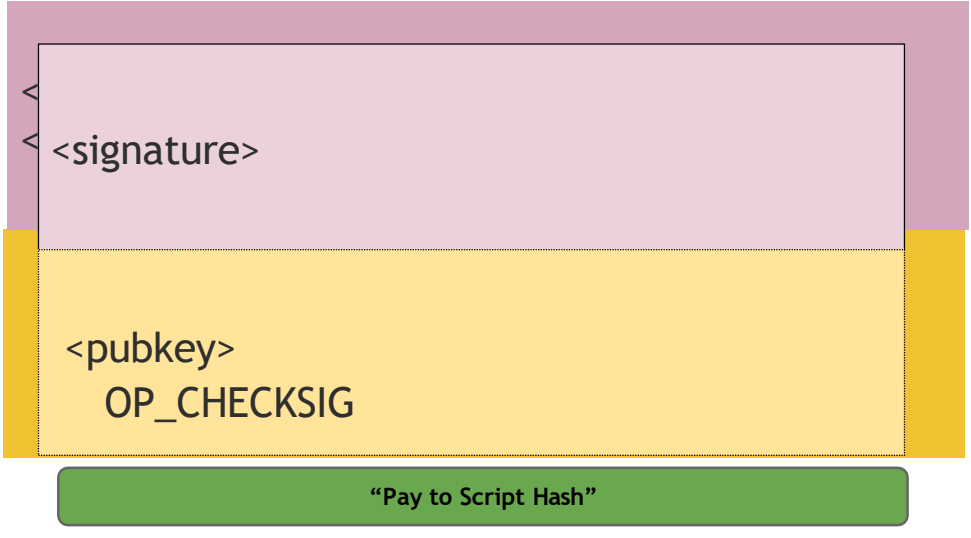
IDEA: USE THE HASH OF REDEMPTION SCRIPT

<signature>
<<pubkey> OP_CHECKSIG>

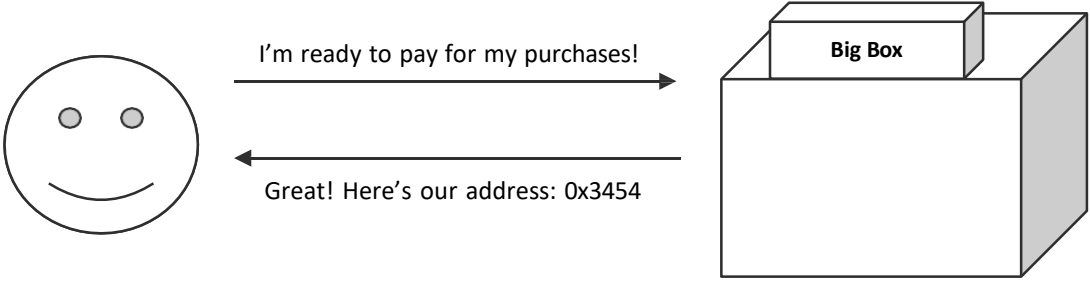
OP_HASH160
hash of <redemption script>
OP_EQUAL

"Pay to Script Hash"

IDEA: USE THE HASH OF REDEMPTION SCRIPT



PAY TO SCRIPT HASH



EXAMPLE

Complex script without P2SH

Locking Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Unlocking Script	Sig1 Sig2

Complex script as P2SH

Redeem Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Locking Script	HASH160 <20-byte hash of redeem script> EQUAL
Unlocking Script	Sig1 Sig2 <redeem script>

EXAMPLE

Complex script without P2SH

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395
D7EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23E3
FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D78D
0E34224858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5737812
F393DA7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD7149B255
382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D087227440645ABE8A
35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF50D56231BE0
AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D137AAB59E0B
000EB7ED238F4D800        5 CHECKMULTISIG
```

Redeem Script 2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

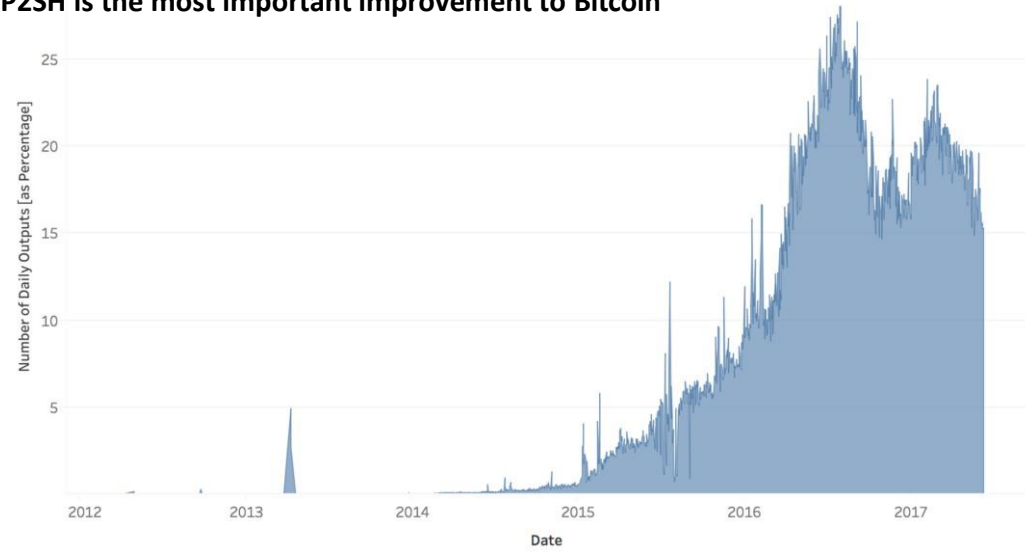
Unlocking Script Sig1 Sig2 <redeem script>

BENEFITS OF P2SH

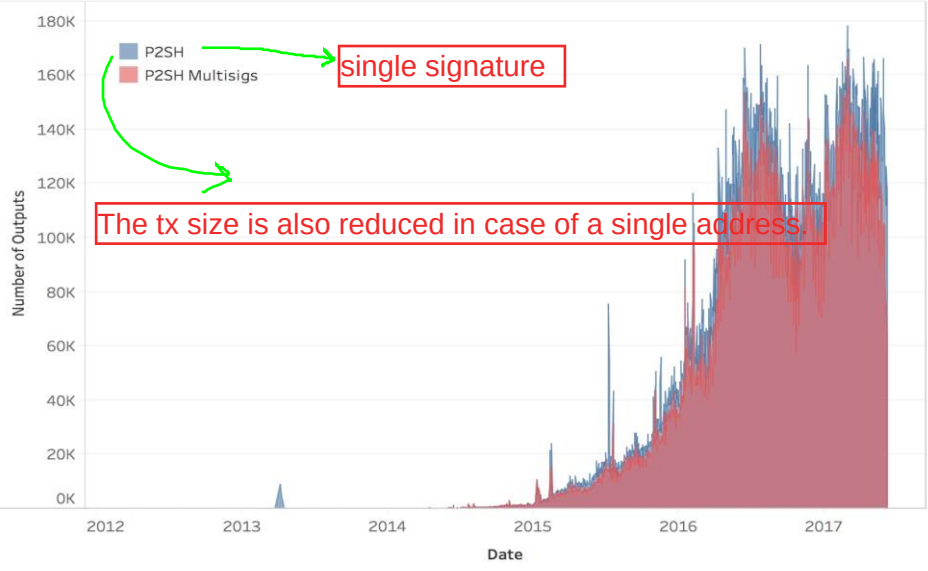
1. Complex scripts are replaced by shorter fingerprints in the transaction output, making the transaction smaller.
2. Scripts can be coded as an address, so the sender and the sender's wallet don't need complex engineering to implement P2SH.
3. P2SH shifts the burden of constructing the script to the recipient, not the sender.
4. P2SH shifts the burden in data storage for the long script from the output (which is in the UTXO set) to the input (stored on the blockchain).
5. P2SH shifts the burden in data storage for the long script from the present time (payment) to a future time (when it is spent).
6. 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.

DEVELOPMENT OF P2SH OUTPUTS

- **P2SH is the most important improvement to Bitcoin**



P2SH ADDRESSES AND P2SH MULTISIGS



RECAP: P2PKH VS. P2SH

- In Bitcoin, senders specify a locking script, recipients provide an unlocking script
- Pay-to-Pub-Key-Hash **(P2PKH):** Vendor (recipient of transaction) says "Send your coins to the hash of this Public Key."
  - Simplest case and by far the most common case
  - Addresses begin with '1': e.g. `15Cytz9sHqeqtKCw2vnpEyNQ8teKtrTPjp`
- **Pay-to-Script-Hash (P2SH):** Vendor says "Send your coins to the hash of this Script; I will provide the script and the data to make the script evaluate to true when I redeem the coins."
  - A vendor cannot say, "To pay me, write a complicated output script that will allow me to spend using multiple signatures."
  - Addresses begin with '3': e.g. `347N1Thc213QqfYCz3PZkjoJpNv5b14kBd`

SEGREGATED WITNESS

hard fork: not backward compatible

- Segregated Witness (segwit) is an upgrade to the bitcoin consensus rules and network protocol, proposed and implemented as a BIP-9 soft-fork that was activated on bitcoin's mainnet on August 1st, 2017.

- In cryptography, the term **witness** is used to describe a solution to a cryptographic puzzle.

- In bitcoin terms, the witness satisfies a cryptographic condition placed on a unspent transaction output (UTXO).
  - A digital signature is *one type of witness*, but a witness is more broadly any solution that can satisfy the conditions imposed on an UTXO and unlock that UTXO for spending.

TRANSACTION ID CALCULATION

- nVersion: `01000000`
- inputs
  - count: `01`
  - 1st input:
    - prevout_hash: `0000000000000000000000000000000000000000000000000000000000000000`
    - prevout_n: `ffffffff`
    - scriptSig:
      `4d` : `04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73`
    - sequence: ffffffff
- outputs
  - count: `01`
  - 1st output:
    - value: `00f2052a01000000` (hex(50*10^8) is 0000012a05f200, and bitcoin puts the bytes in reverse order)
    - scriptPubKey:
      `43` : `4104678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5fac`
- nLockTime: `00000000`

base58

TRANSACTION ID CALCULATION

- nVersion: `01000000`
- inputs
  - count: `01`
  - 1st input:
    - prevout_hash: `0000000000000000000000000000000000000000000000000000000000000000`
    - prevout_n: `ffffffff`

`01000000010000000000000000000000000000000000000000000000000000000000000000ffffffff4d04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73ffffffff0100f2052a01000000434104678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5fac00000000`

- 1st output:
  - value: `00f2052a01000000` (hex(50*10^8) is 0000012a05f200, and bitcoin puts the bytes in reverse order)
  - scriptPubKey: `43` : `4104678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5fac`
- nLockTime: `00000000`

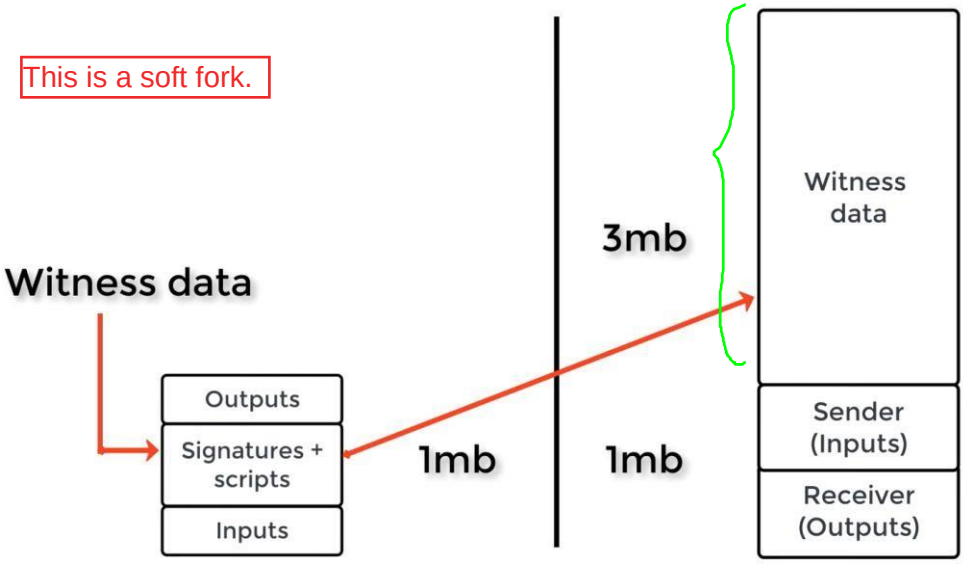
tx hash = hash of this section

concatenation of a

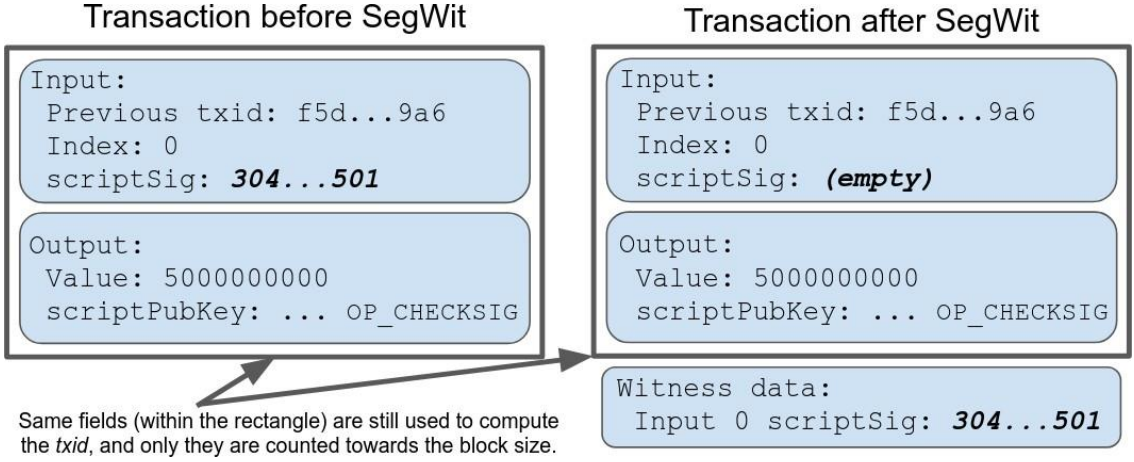
TRANSACTION CHALLENGES ?

- nVersion: `01000000`
- inputs
  - count: `01`
  - 1st input:
    - prevout_hash: `0000000000000000000000000000000000000000000000000000000000000000`
    - prevout_n: `ffffffff`
    - scriptSig:
      `4d` : `04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66207365636f6e64206261696c6f757420666f722062616e6b73`
    - sequence: ffffffff
- outputs
  - count: `01`
  - 1st output:
    - value: `00f2052a01000000` (hex(50*10^8) is 0000012a05f200, and bitcoin puts the bytes in reverse order)
    - scriptPubKey:
      `43` : `4104678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5fac`
- nLockTime: `00000000`

SEGWIT VS. NON-SEGWIT



SEGWIT VS. NON-SEGWIT



P2WPKH OUTPUT EXAMPLE

witness

Example P2PKH output script

```
DUP HASH160 ab68025513c3dbd2f7b92a94e0581f5d50f654e7 EQUALVERIFY CHECKSIG
```

Example P2WPKH output script

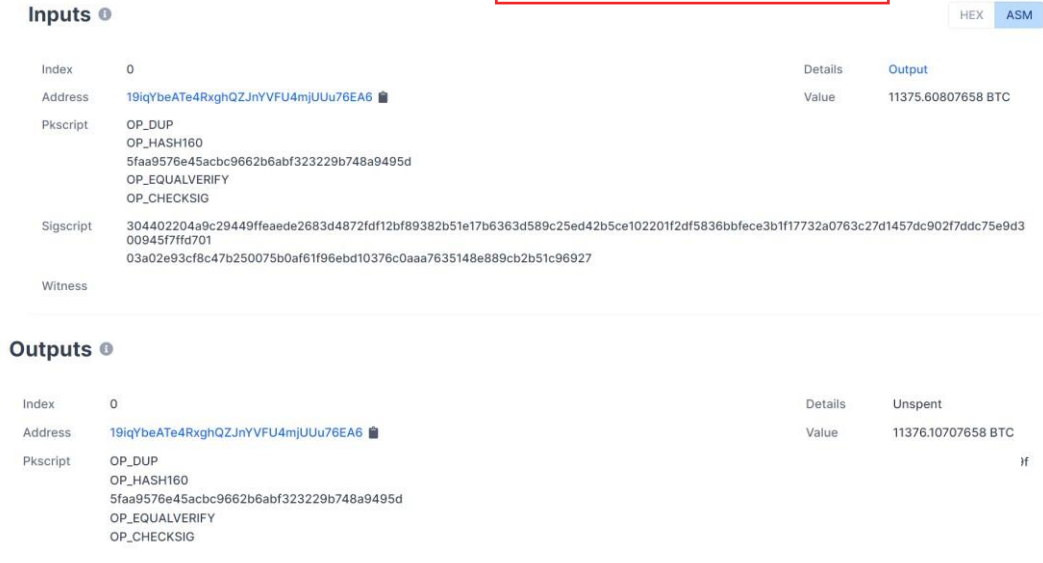
```
0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7
```

nop

witness address

EXAMPLE: P2PKH

*** Refer to the screenshot.



Inputs

HEX ASM

Index	0	Details	Output
Address	19iqYbeATe4RxghQZJnYVFU4mjUUu76EA6	Value	11375.60807658 BTC
Pkscript	OP_DUP OP_HASH160 5faa9576e45acbc9662b6abf323229b748a9495d OP_EQUALVERIFY OP_CHECKSIG		
Sigscript	304402204a9c29449ffeaede2683d4872fdf12bf89382b51e17b6363d589c25ed42b5ce102201f2df5836bbfece3b1f17732a0763c27d1457dc902f7ddc75e9d300945f7ffd701 03a02e93cf8c47b250075b0af61f96ebd10376c0aaa7635148e889cb2b51c96927		
Witness			

Outputs

Index	0	Details	Unspent
Address	19iqYbeATe4RxghQZJnYVFU4mjUUu76EA6	Value	11376.10707658 BTC
Pkscript	OP_DUP OP_HASH160 5faa9576e45acbc9662b6abf323229b748a9495d OP_EQUALVERIFY OP_CHECKSIG		

EXAMPLE: P2WPKH

Inputs

HEX ASM

Index	0	Details	Output
Address	bc1qdyqfggeuy3nzlht0mnx9q3ghsvd584j3xw4gqj	Value	0.01706450 BTC
Pkscript	OP_0 690094233c24662fdd6fdccc504517831b43d651		
Sigscript			
Witness	304402206aa8159019ecc8898e2f4768cfd02e6a9161e970029a5afdf6bdf2f3552ea40902201e329c8c61abc200270d4f2ec48c3f24753a6463a9b47d735eb720f98daf18c901 036bda5d4a931f0026bf1eb2759b4cb47216b98d9f3b932c5269e13feaeae4ddc3		

Outputs

Index	0	Details	Unspent
Address	bc1quugjmxvk8mx7ql8pxa77kc4des09n5u626nwjy	Value	0.01682968 BTC
Pkscript	OP_0 e7112d99963ecde07ce1377deb62adcc1e59d39a		

SEGWIT ADVANTAGES

- ***Transaction Malleability:*** Since the witness data is the only part of the transaction that can be modified by a third party, removing it also removes the opportunity for transaction malleability attacks.
- ***Script Versioning:*** With the introduction of segwit, every locking script is preceded by a *script version* number, which allows the scripting language to be upgraded in a backward-compatible way to introduce new script operands, syntax, or semantics.
- ***Network and Storage Scaling:*** The witness data is a big contributor to the size of a transaction. By moving the witness data outside the transaction, segwit improves bitcoin's scalability
- ...

OUTPUT TYPES BY COUNT