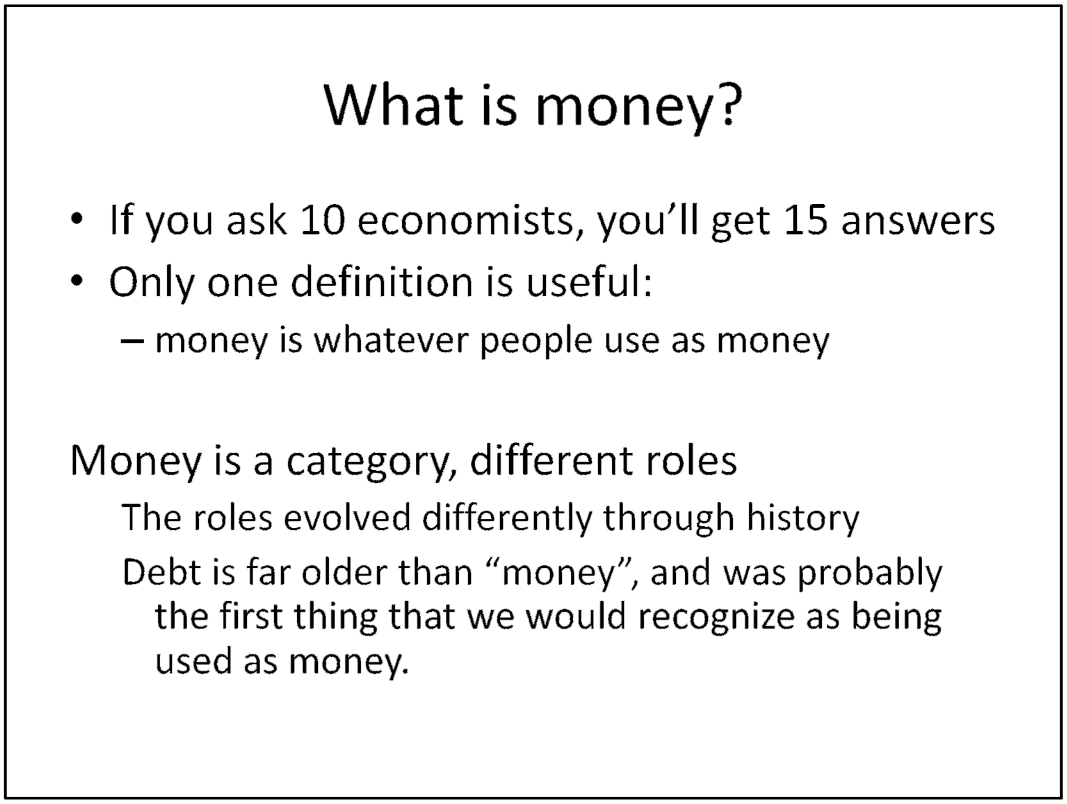
Bitcoin

2014-01-09

Kyle Jerviss
bitcoin-klug@jerviss.org



What is money?

- If you ask 10 economists, you'll get 15 answers
- Only one definition is useful:
  - money is whatever people use as money

Money is a category, different roles

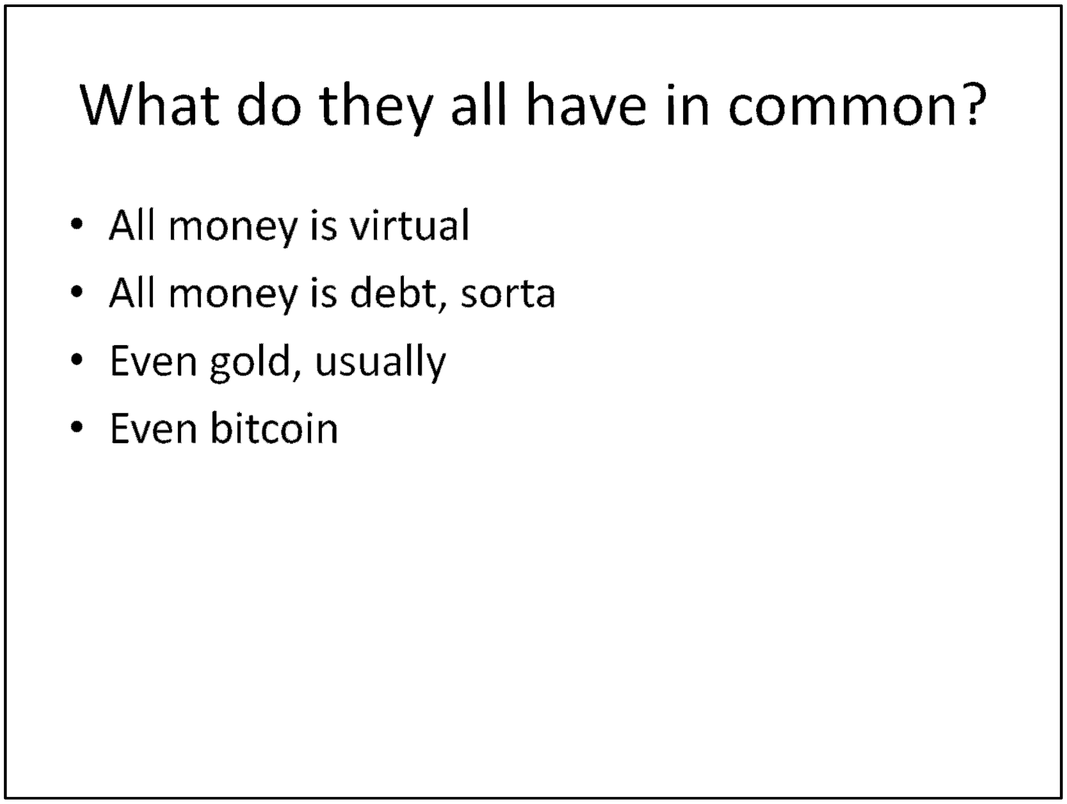
The roles evolved differently through history

Debt is far older than "money", and was probably the first thing that we would recognize as being used as money.

Probably more like 30 definitions.
Debt: The First 5000 Years by David Graeber
Money allowed settlement of debt in a universal way, made monetary fines possible
The evolution of legal codes in Sumer and Babylon are illuminative

What about us?

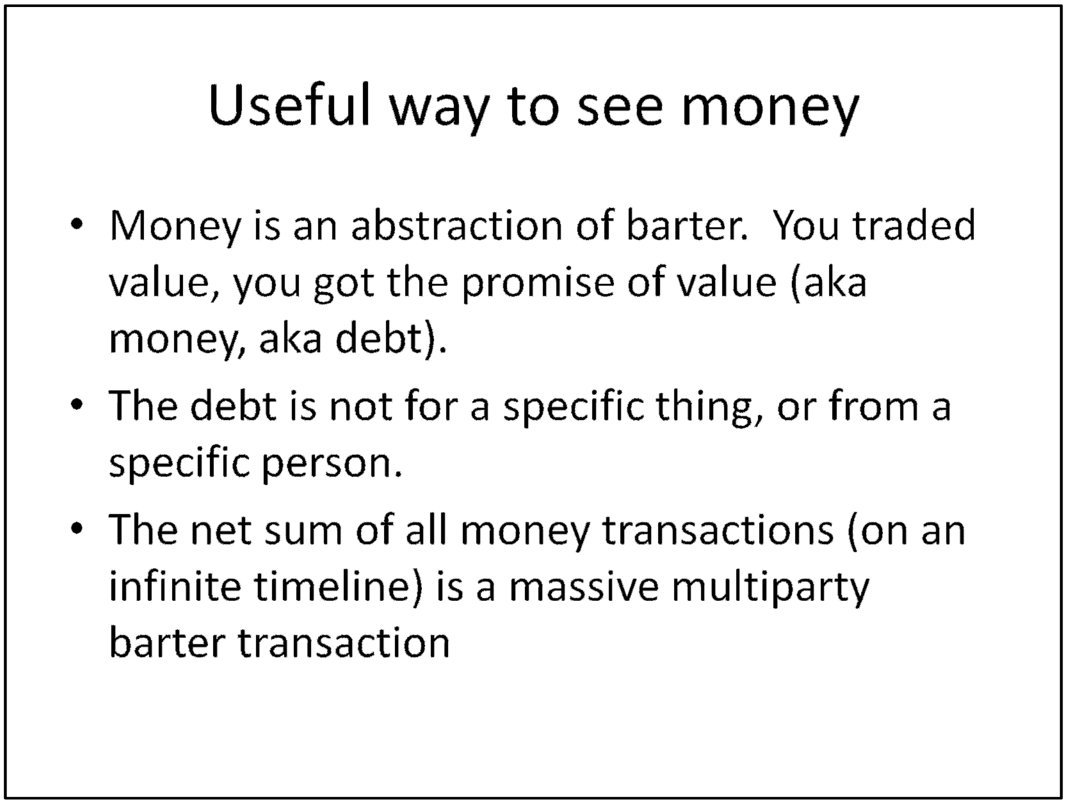
- For most of us, our checking accounts are the real money, and physical cash is used as bearer bonds to transfer between accounts.
- For poorer people, it is the reverse, with cash being real money, and checks as a way to move cash.
- Credit cards fit in there somewhere. I use mine for 100% of ordinary daily transactions.



What do they all have in common?

- All money is virtual
- All money is debt, sorta
- Even gold, usually
- Even bitcoin

Unless you have some use for physical gold, you are accepting it in trade with the expectation of trading it again. You are using it as a bearer bond, just like cash.



Useful way to see money

- Money is an abstraction of barter. You traded value, you got the promise of value (aka money, aka debt).
- The debt is not for a specific thing, or from a specific person.
- The net sum of all money transactions (on an infinite timeline) is a massive multiparty barter transaction

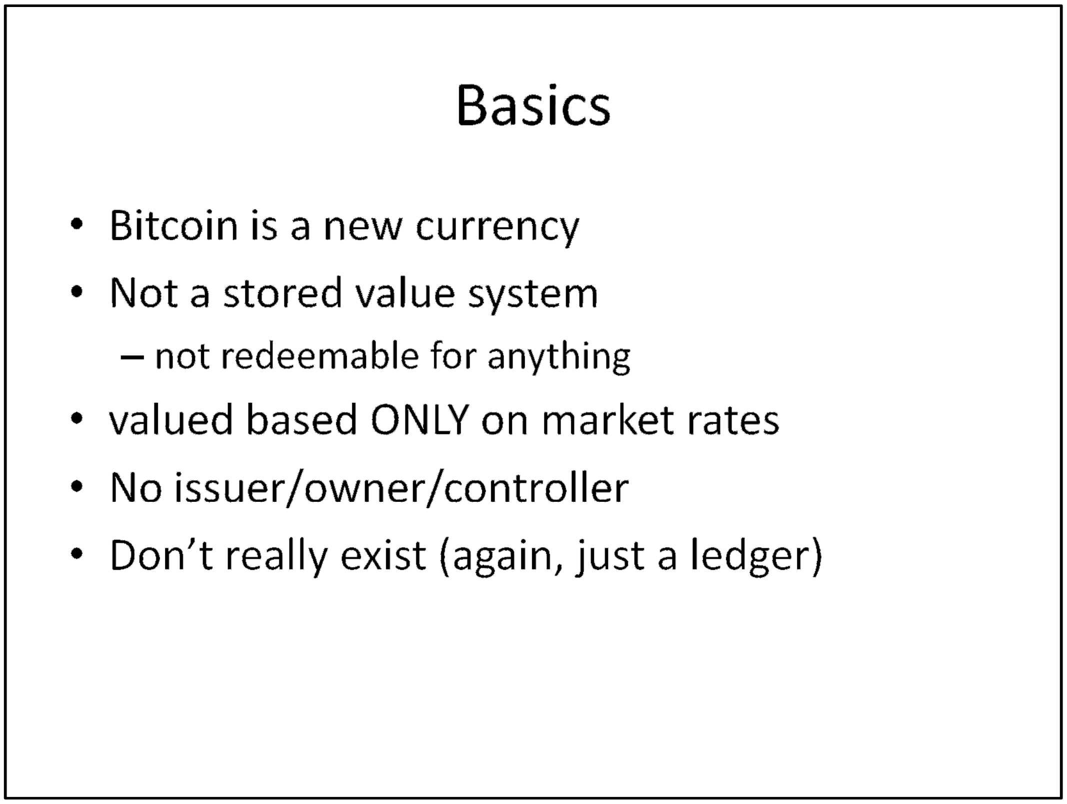
The guy you are selling to got his money by trading something to someone else. You'll get rid of it later by passing it on yet again.
Not specific = virtual
The net sum for you is also a barter, and you didn't have to find the one guy that wanted exactly what you had and had exactly what you wanted

The law

- Interestingly enough, most of our laws are vague about money.
- Words like “money” and “currency” are rarely defined in a useful way.
- Most laws re: money are unconstitutional.
  - See *Pieces of Eight* by Edwin Vieira Jr. if you want 1722 pages explaining exactly how and why.
    - only $200 on Amazon
  - But no one cares.

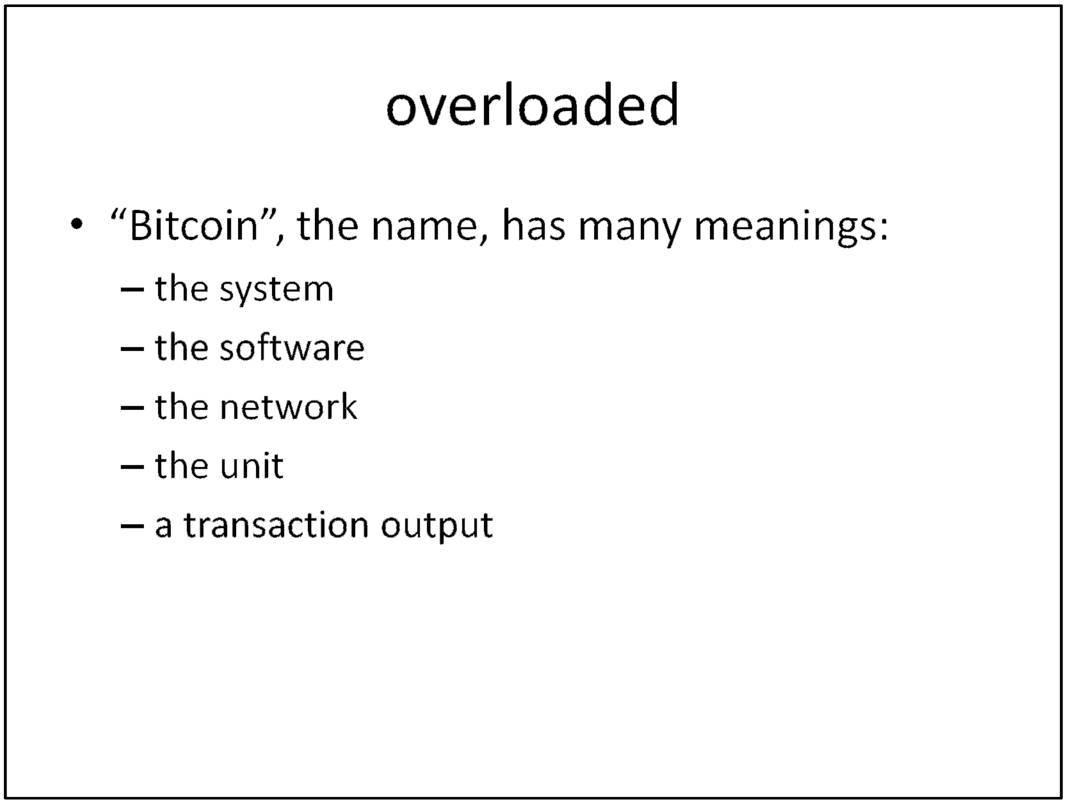
Bitcoin

- Just a ledger
  - a *public* ledger
- You spend it by announcing to the world that you are giving it to someone else
- Not as novel as you might think.
  - Rai stones in the Yap culture
  - Way too big to move easily, you owned them because everyone knew that you owned them, and you spent them by telling everyone that you were spending them.


Basics

- Bitcoin is a new currency
- Not a stored value system
  - not redeemable for anything
- valued based ONLY on market rates
- No issuer/owner/controller
- Don't really exist (again, just a ledger)

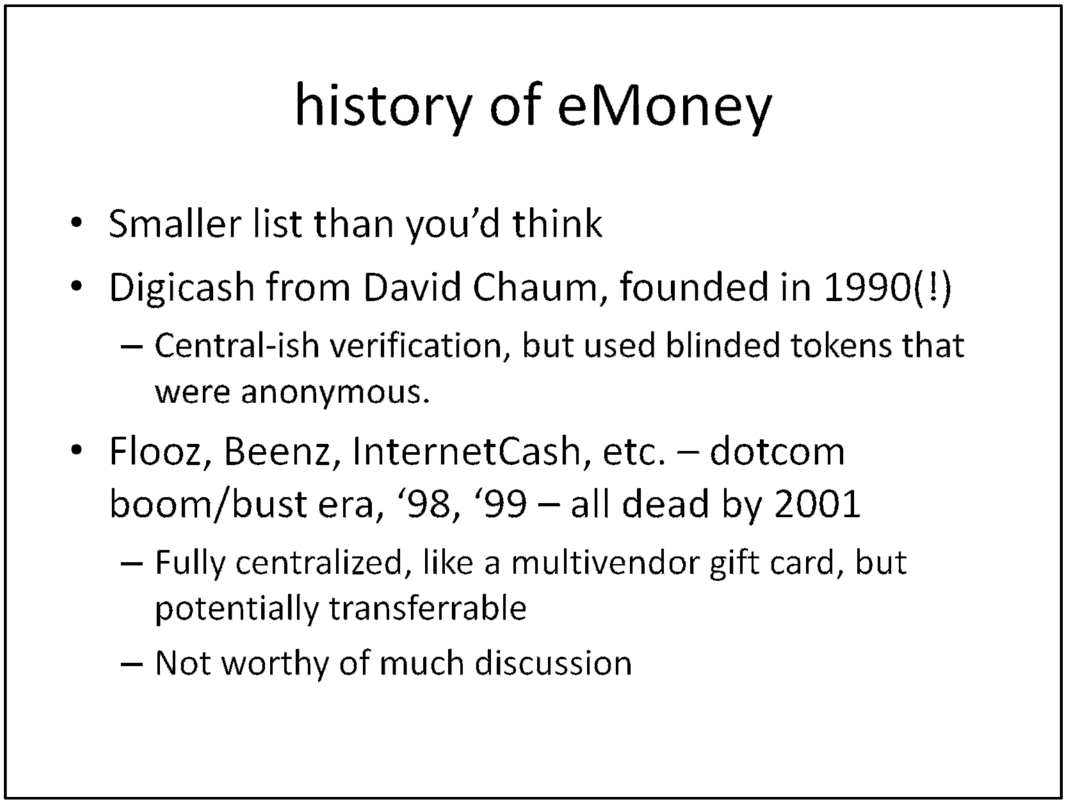
This slide is new. I did not have it during the presentation, but should have.



overloaded

- “Bitcoin”, the name, has many meanings:
  - the system
  - the software
  - the network
  - the unit
  - a transaction output

Another slide of basic info that I should have included, but didn’t.



history of eMoney

- Smaller list than you'd think
- Digicash from David Chaum, founded in 1990(!)
  - Central-ish verification, but used blinded tokens that were anonymous.
- Flooz, Beenz, InternetCash, etc. – dotcom boom/bust era, '98, '99 – all dead by 2001
  - Fully centralized, like a multivendor gift card, but potentially transferrable
  - Not worthy of much discussion

I had intended to start my presentation here, with a discussion of past attempts at electronic money. Turns out that my memory was faulty and there were only a few of note, so this is very quick.

Digicash

- Banks signed tokens
- Spend by giving token to recipient
- Recipient asks bank to transfer value of token to their account, invaliding the token
- Signature is blind – bank can verify that the token has their signature, but can’t tell which account it came from

Digicash, continued

- This really happened. Actual real live banks did this.
- Notably, Mark Twain Bank
  - You've never heard of it because it was acquired by MBNA in 1997 for reasons unrelated to Digicash

aside: Blind Signatures

- Many forms, most common example is RSA
- In RSA, a signature, s = m^d(mod n)
  - m is the message (hash), d is the privkey, n is the order (expressed as key length)
- Blinded, the message, m’ = m*r^e (mod n)
  - m is the hash, r is random, e is the pubkey
- s’ = m’^d(mod n)
- r can be removed from s’, leaving s as a valid signature for m, despite the signer having never seen m

What do we want/need?

- Secure (obvious)
  - no double spends
  - no unauthorized spends
  - no overspends/underspends
- Decentralized (lesson from losing DigiCash)
- Private-ish (lesson from all human history)

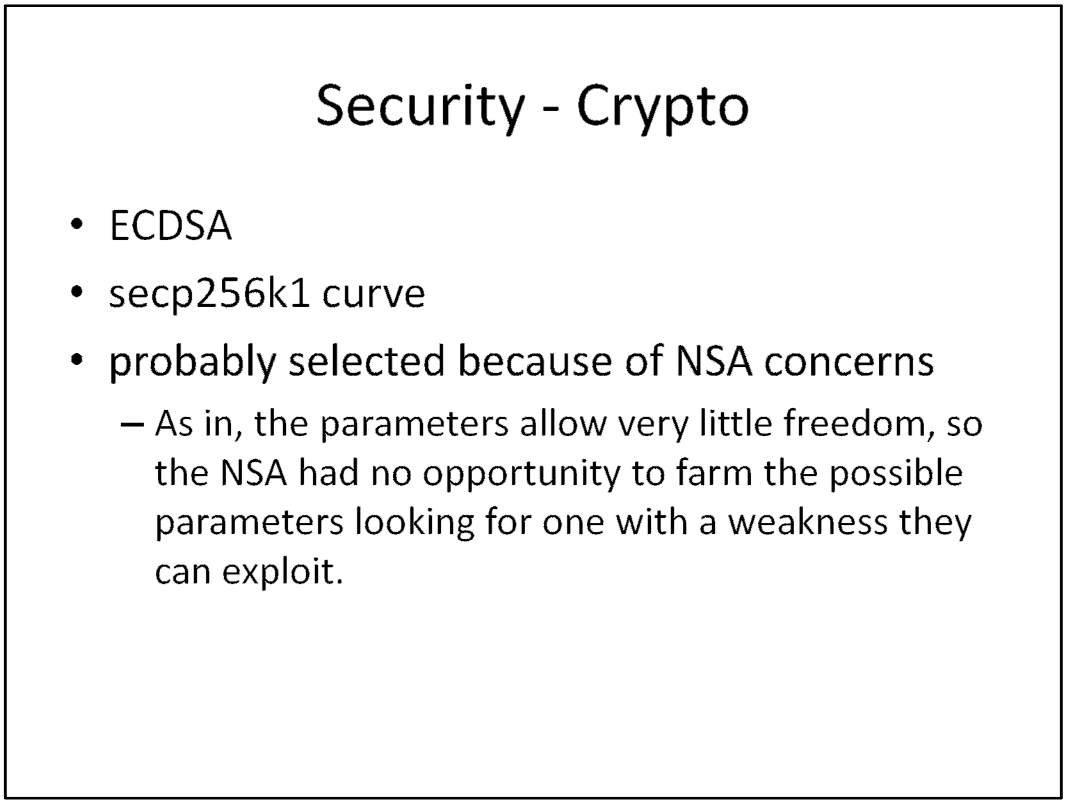
How do we get those?

- Transactions are just simple data structures with cryptographic signatures (security)
- All transactions are verified by the entire world (security)
- To resolve conflicts, transactions are put into a specific order by miners (decentralized)
- New coins are generated over time, by participants (decentralized)

Security - hashes

- Most of the time, hashing in bitcoin means SHA256(SHA256(x))
- In one place, it means RIPEMD160(SHA256(x))
- Hashing is safe, even from quantum attacks, for many decades
- In most places, even totally broken hashes (MD5) would be perfectly fine, other constraints preclude ALL known preimage attacks
  - Output size would be an issue

Grover’s algorithm can “search” SHA-256, or even double SHA-256, but it works on circuits. Quantum circuits. I’m not entirely sure that a classic SHA-256 circuit (no loops, no registers) is possible with our current IC production technology. State of the art in quantum circuits is a handful of gates, and coherence, so far, seems like it is going to be a problem every time we try to add one more.
Preimage attacks rely on shitty inputs, AKA “your message + specially prepared garbage”. Bitcoin does not allow arbitrary garbage insertion in most places.



Security - Crypto

- ECDSA
- secp256k1 curve
- probably selected because of NSA concerns
  - As in, the parameters allow very little freedom, so the NSA had no opportunity to farm the possible parameters looking for one with a weakness they can exploit.

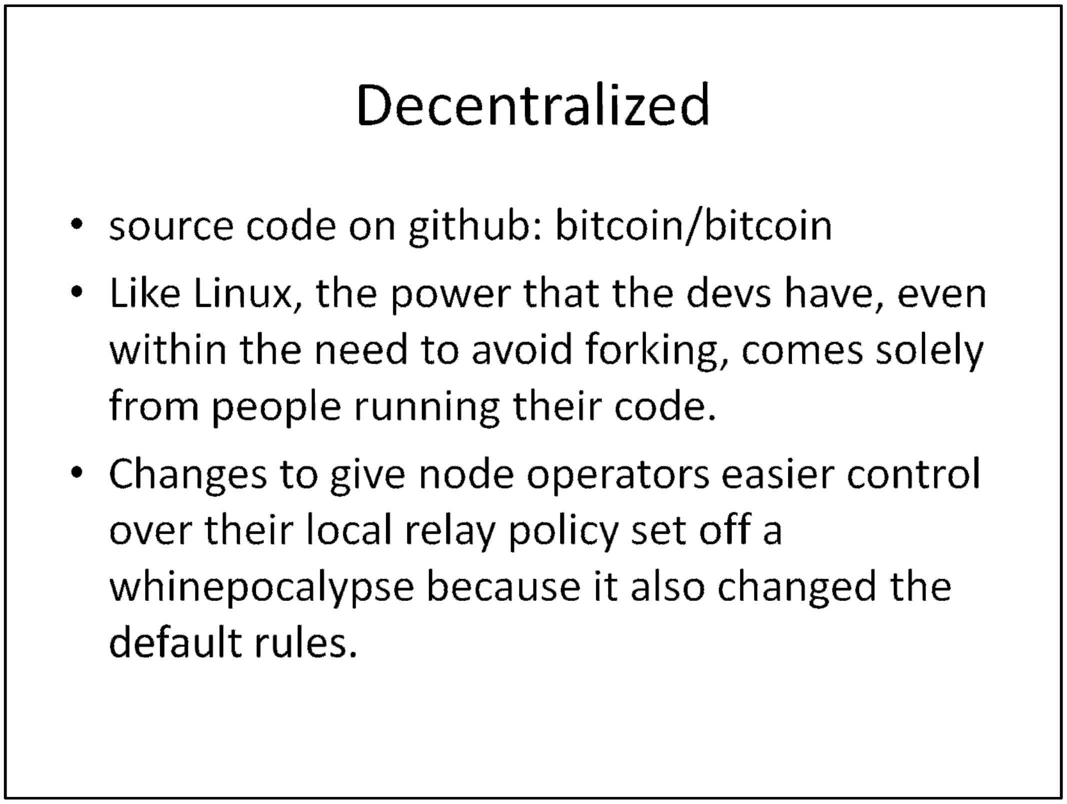
Greg Maxwell has looked into this quite a bit.

Security - system

- Since all transactions are verified by everyone, no one can break the rules.
- Can’t create extra money
- Can’t make you spend more than you intended
- Can’t short the recipient

Decentralized

- No one owns the system.
- No one started with all of the money.
- Even the devs have very limited ability to make changes to the system.
- Incompatible changes define an entirely new system, not a change to the current one.



Decentralized

- source code on github: bitcoin/bitcoin
- Like Linux, the power that the devs have, even within the need to avoid forking, comes solely from people running their code.
- Changes to give node operators easier control over their local relay policy set off a whinepocalypse because it also changed the default rules.

Byzantine Generals

- What stops me from paying two people with the same coins?
- Obviously the first one is good, and the second one is invalid.
- But which was first?
  - Relativity says “no”.
  - Network latency says “Hell no.”

A or B?

- The traditional way (digicash) is to have an issuer approve the first one they see, and deny the rest.
- Incompatible with “decentralized”

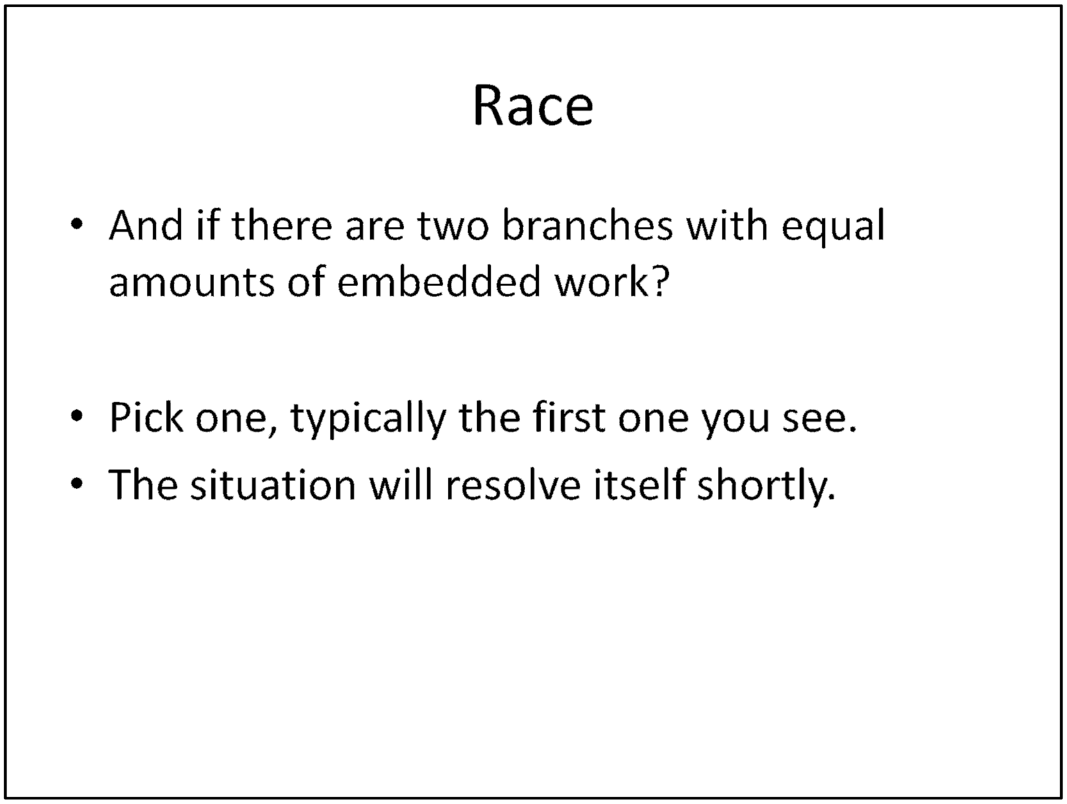
Order

- We define an ordering. Transactions happen in the order they appear in the blockchain.
- Not in the chain? Order undefined.
  - beware undefined

Blocks

- Ok, but what order are blocks in?

- Each has a link to the previous block. Not valid anywhere else in the tree.
- Each takes work to create. Work is tied to that block, can't be saved or stored.
- The longest chain is the right chain.
  - longest defined by work needed to replace

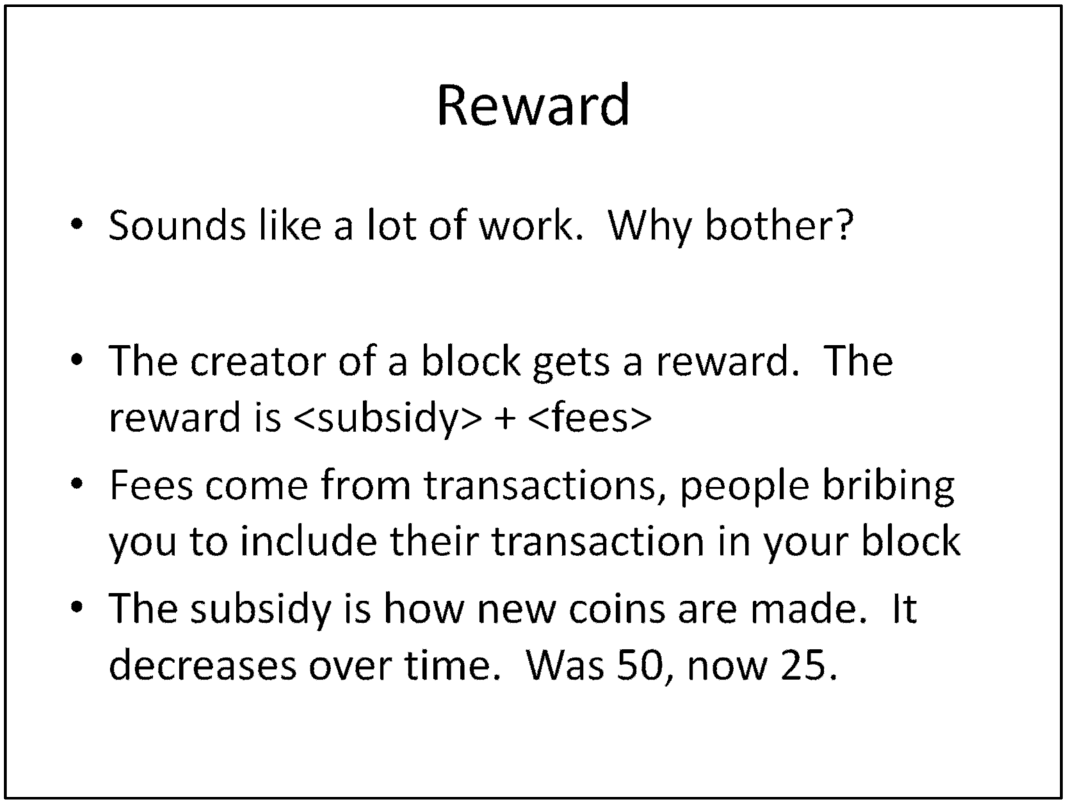


Race

- And if there are two branches with equal amounts of embedded work?

- Pick one, typically the first one you see.
- The situation will resolve itself shortly.

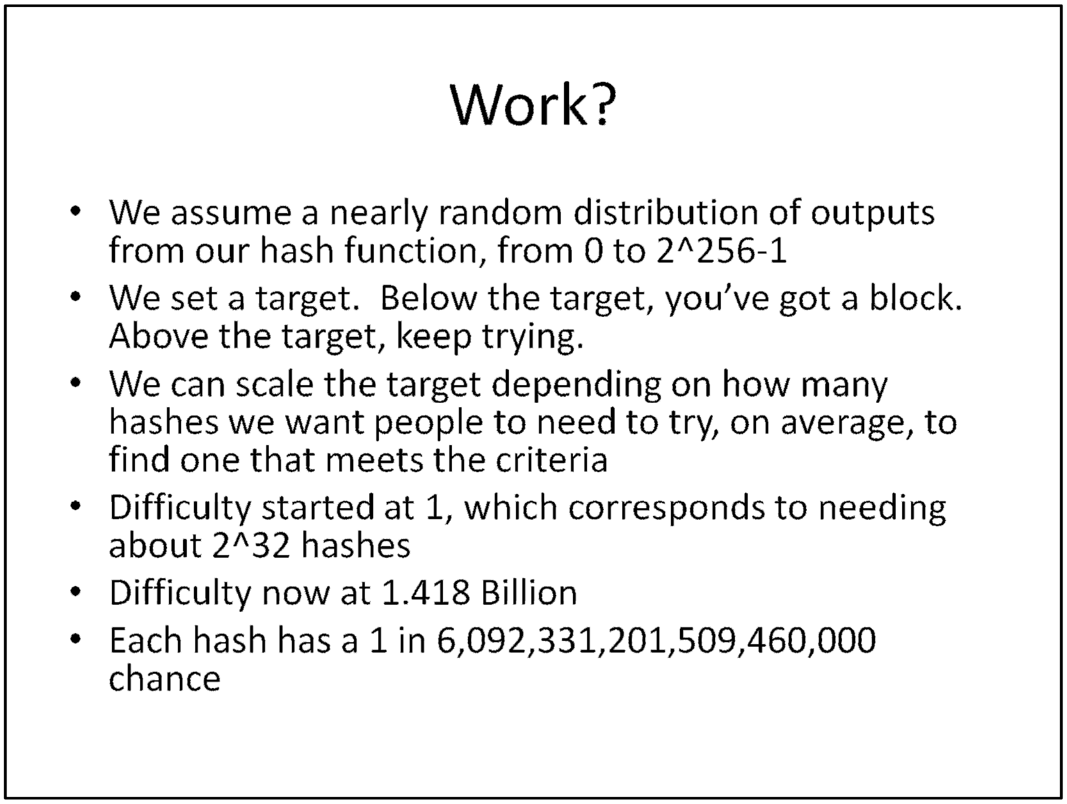
It is extremely unlikely that both blocks will be seen by exactly half of the mining power.
One side or the other will have an advantage, and will probably win.
Even if they were even, mining is random, so two MORE blocks within the latency period are even less likely.
And so on.



Reward

- Sounds like a lot of work. Why bother?
- The creator of a block gets a reward. The reward is <subsidy> + <fees>
- Fees come from transactions, people bribing you to include their transaction in your block
- The subsidy is how new coins are made. It decreases over time. Was 50, now 25.

Fees are dysfunctional right now. For one thing, the subsidy distorts the hell out of the market.
As a result, miners don't compete.
As a result, senders don't compete.
Someday, a decent market will rise up.



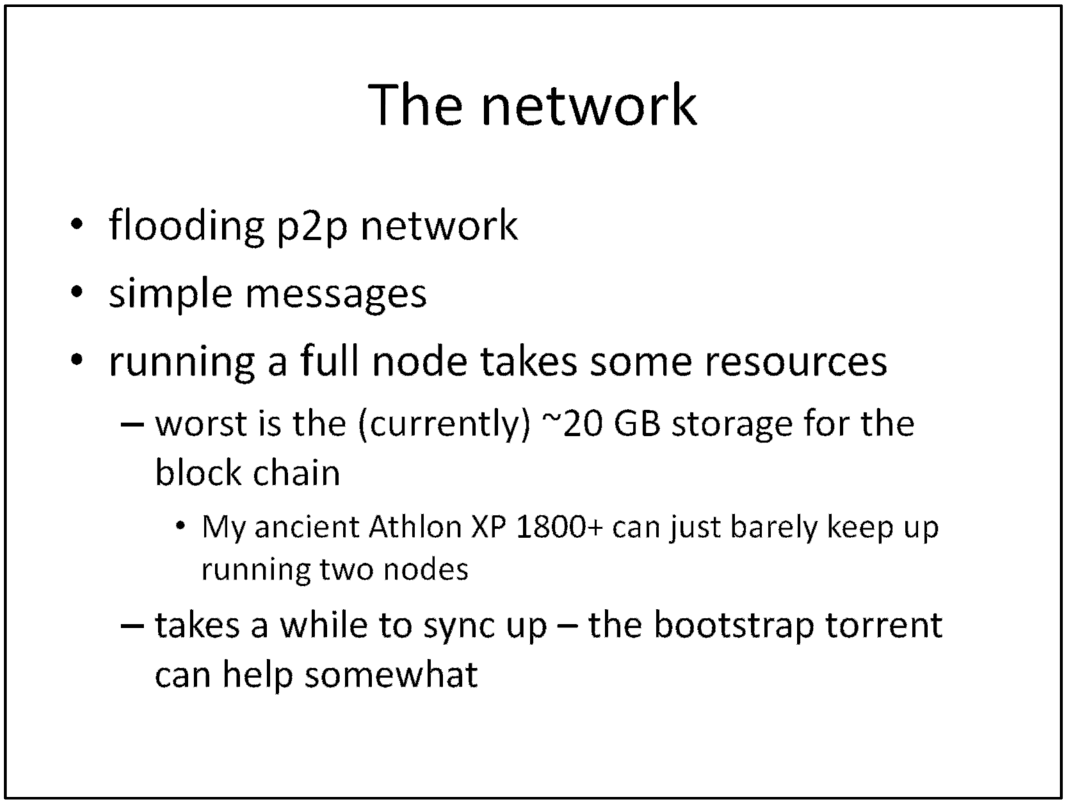
Work?

- We assume a nearly random distribution of outputs from our hash function, from 0 to 2^256-1
- We set a target. Below the target, you've got a block. Above the target, keep trying.
- We can scale the target depending on how many hashes we want people to need to try, on average, to find one that meets the criteria
- Difficulty started at 1, which corresponds to needing about 2^32 hashes
- Difficulty now at 1.418 Billion
- Each hash has a 1 in 6,092,331,201,509,460,000 chance

The random output assumption seems to be pretty good, so far.

Why scale?

- Difficulty scaling decouples the block rate from the work rate.
- Work now happening 1.5 billion times faster than in 2009. Blocks still coming about once per 10 minutes.
- Also the subsidy rate (aka money creation rate).
- related to orphan rate



The network

- flooding p2p network
- simple messages
- running a full node takes some resources
  - worst is the (currently) ~20 GB storage for the block chain
    - My ancient Athlon XP 1800+ can just barely keep up running two nodes
  - takes a while to sync up – the bootstrap torrent can help somewhat

my Athlon XP 1800+ with 896 MB RAM can just barely keep up with running 2 nodes

Clients

- Only a few (full) clients exist
- Writing a new client is damn hard
- human-readable specs are (necessarily) incomplete
- you have to get the bugs right too.
  - In a conflict between the written specs and the way the network actually operates, the specs always lose.

Wallets

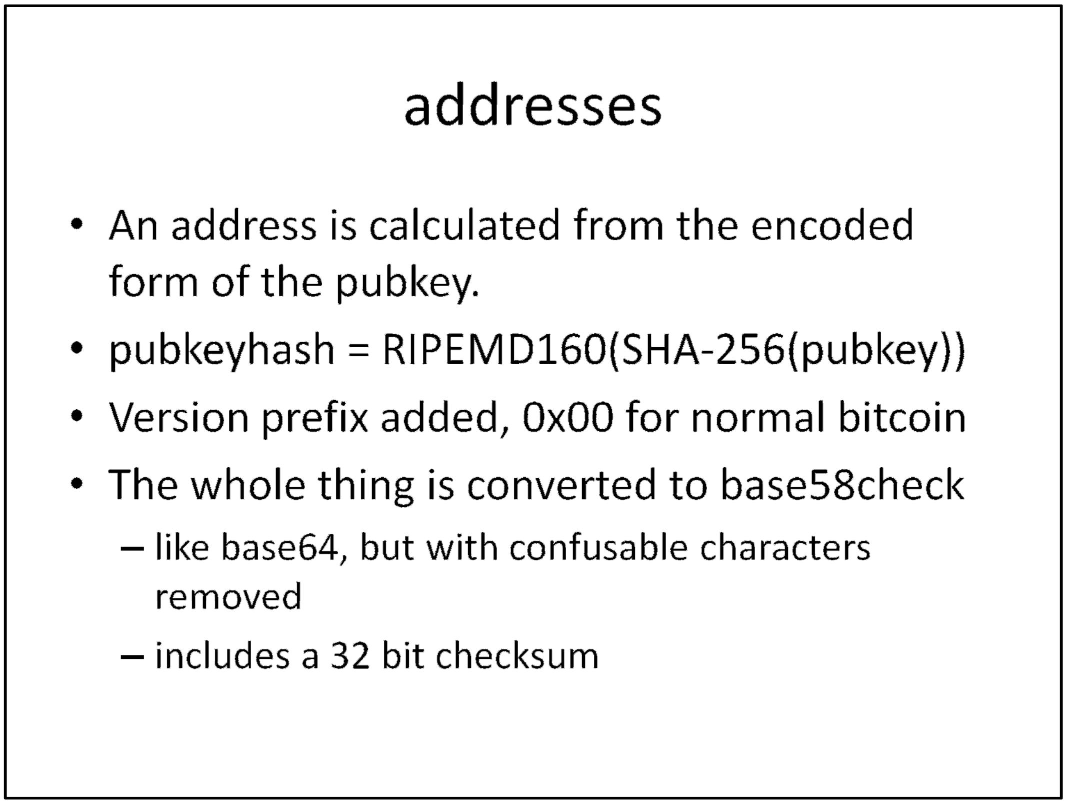
- non-verifying clients are MUCH easier to write, and a few exist
- Armory has tons of security features, but requires that you also run the reference client
- MultiBit and Electrum are fast and aim to be user friendly
- Android clients exist, as do web wallets
  - think carefully before using web wallets

Offline

- A transaction that goes to a key you own is yours, even if you don’t know about it
- Meaning that you can print a key on paper, then send money to that address
- If you do it right: very, very secure

keys

- A private key is 256 bits. ANY 256 bits is a private key.
- The public key is G*privkey. * is EC point multiplication on the curve secp256k1
- The result of point multiplication is (x,y) of a point on the curve, x and y are 256 bits each
- y can be calculated from x, but you get two possible answers
- The pubkey is encoded:
  - 0x04<x><y>
  - 0x03<x> or 0x02<x>.
    - y can be recovered from the short (compressed) forms. 2 means you take the answer with even parity, 3 means you take the odd one.



addresses

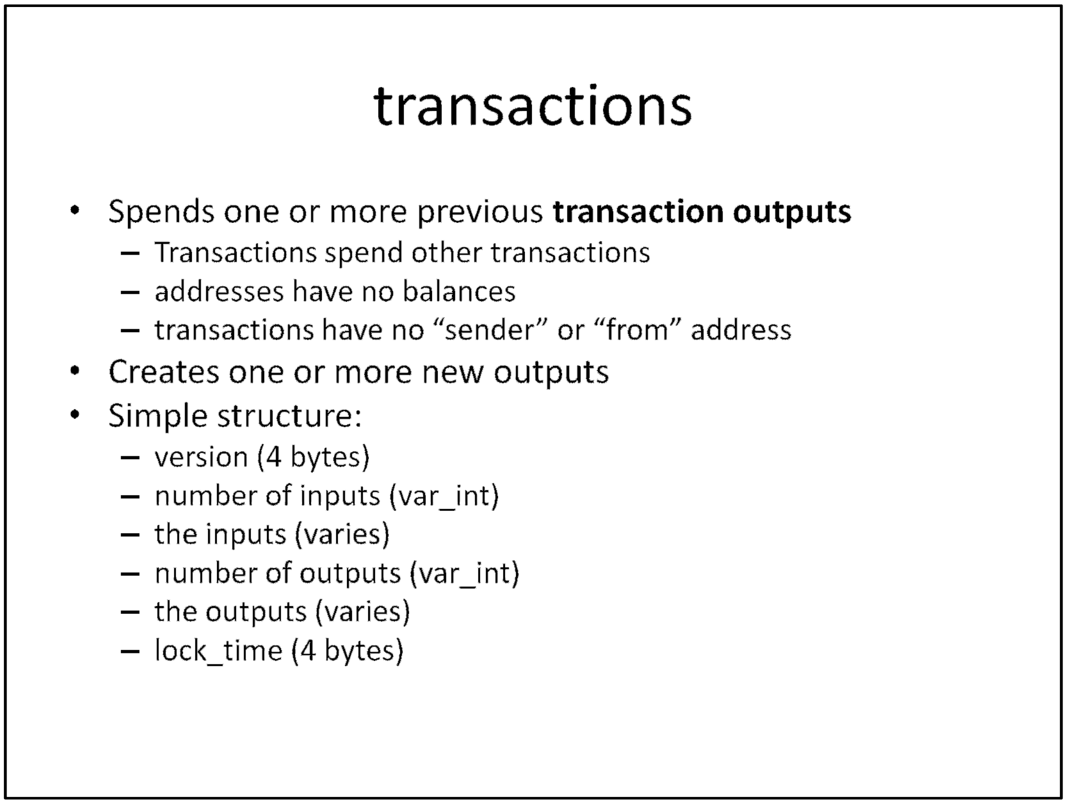
- An address is calculated from the encoded form of the pubkey.
- pubkeyhash = RIPEMD160(SHA-256(pubkey))
- Version prefix added, 0x00 for normal bitcoin
- The whole thing is converted to base58check
  - like base64, but with confusable characters removed
  - includes a 32 bit checksum

This means that the same (x,y) point can have two different addresses. This is usually handled by pretending the other one doesn’t exist.

pubkey hash

- Why not just use the pubkey?

- the hash is shorter (160 bits vs. 256)
- if EC is broken some day, the pubkeys remain hidden until use



transactions

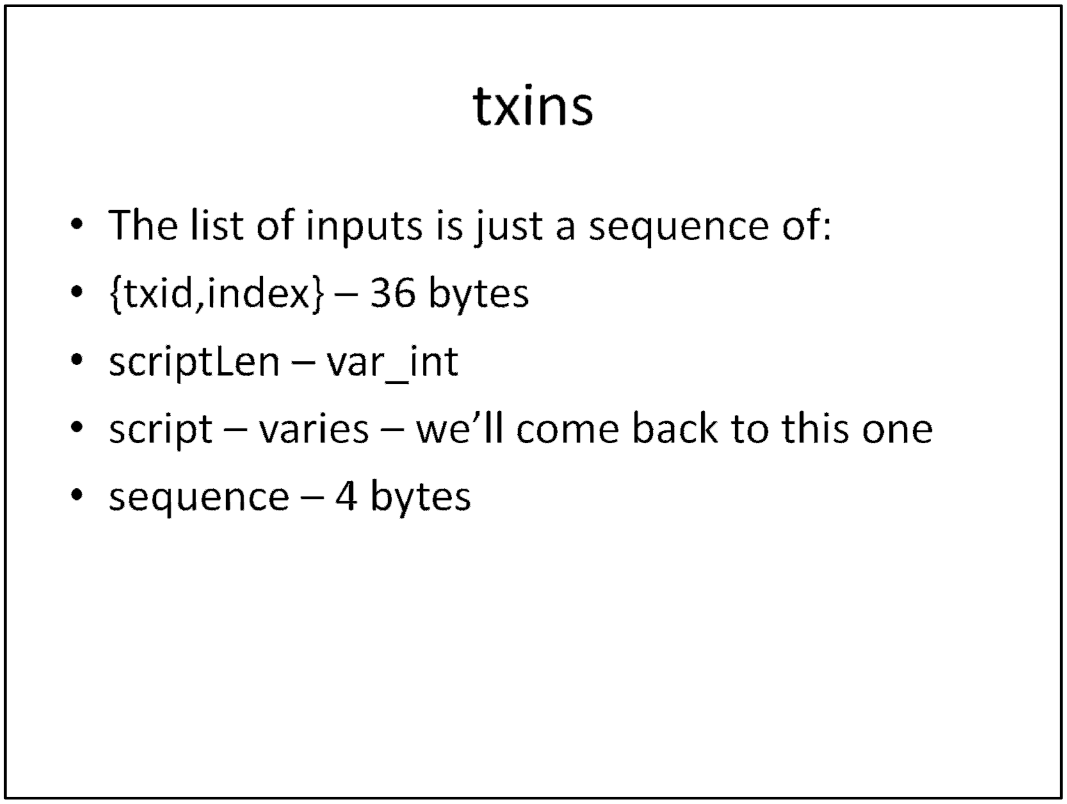
- Spends one or more previous **transaction outputs**
  - Transactions spend other transactions
  - addresses have no balances
  - transactions have no "sender" or "from" address
- Creates one or more new outputs
- Simple structure:
  - version (4 bytes)
  - number of inputs (var_int)
  - the inputs (varies)
  - number of outputs (var_int)
  - the outputs (varies)
  - lock_time (4 bytes)

One exception is the generate transaction, which collects fees and creates new coins through the subsidy

This slide was modified to clarify that transactions spend other transactions

transactions, 2

- TxID is the hash of the transaction
- specific outputs named by {txid,index}



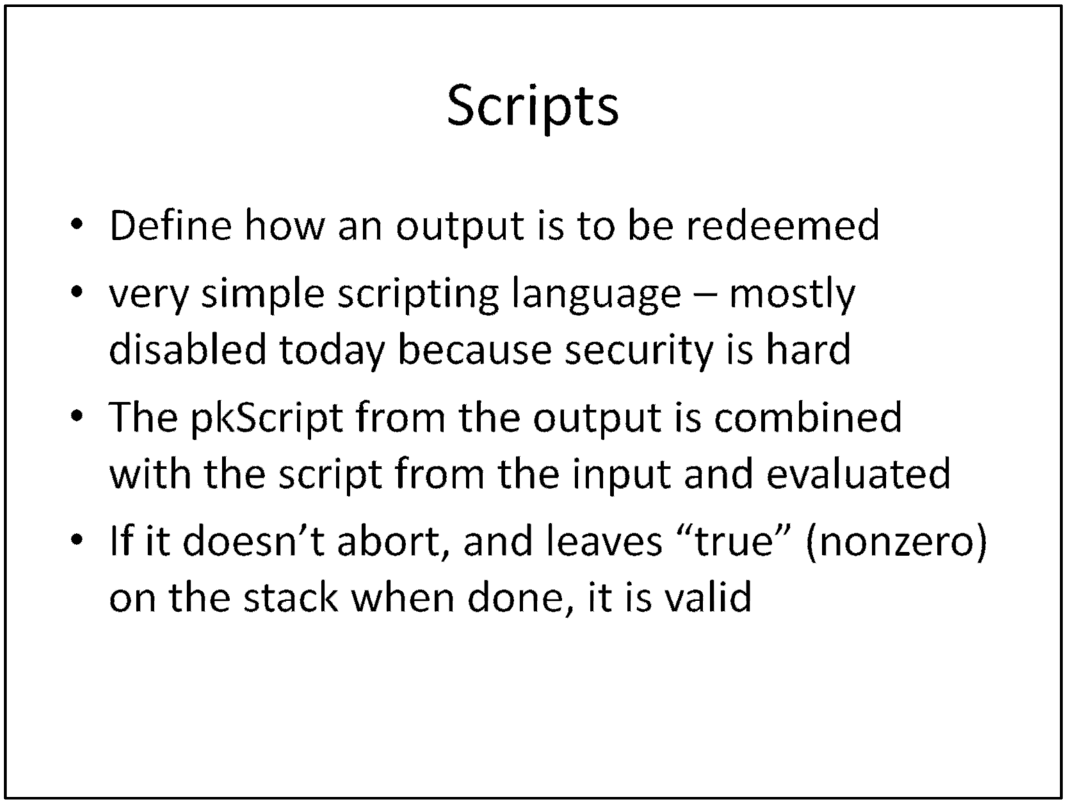
txins

- The list of inputs is just a sequence of:
- {txid,index} – 36 bytes
- scriptLen – var_int
- script – varies – we’ll come back to this one
- sequence – 4 bytes

sequence is for creating chains of transactions, it is typically the max value

txouts

- Likewise, the outputs are simple
- value – 64 bit unsigned integer
- pkScriptLen – var_int
- pkScript - varies



Scripts

- Define how an output is to be redeemed
- very simple scripting language – mostly disabled today because security is hard
- The pkScript from the output is combined with the script from the input and evaluated
- If it doesn't abort, and leaves "true" (nonzero) on the stack when done, it is valid

Soft disabled. There is a function that checks for standard-ness. If not standard, most people won't relay it, and most miners won't include it in a block.

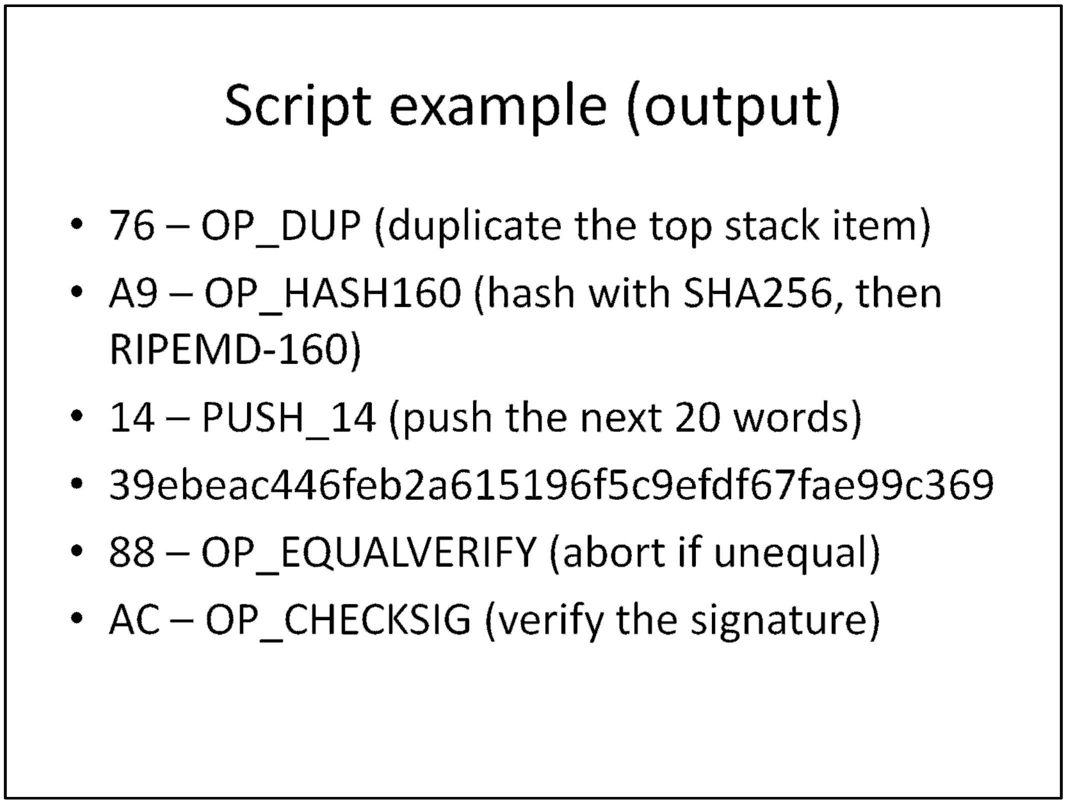
Typical script usage

For the typical case, we need to connect a transaction to a pubkey hash (address).

We need the pubkey to verify the signature, and the pubkey is not published in advance

Does this pubkey produce this signature?

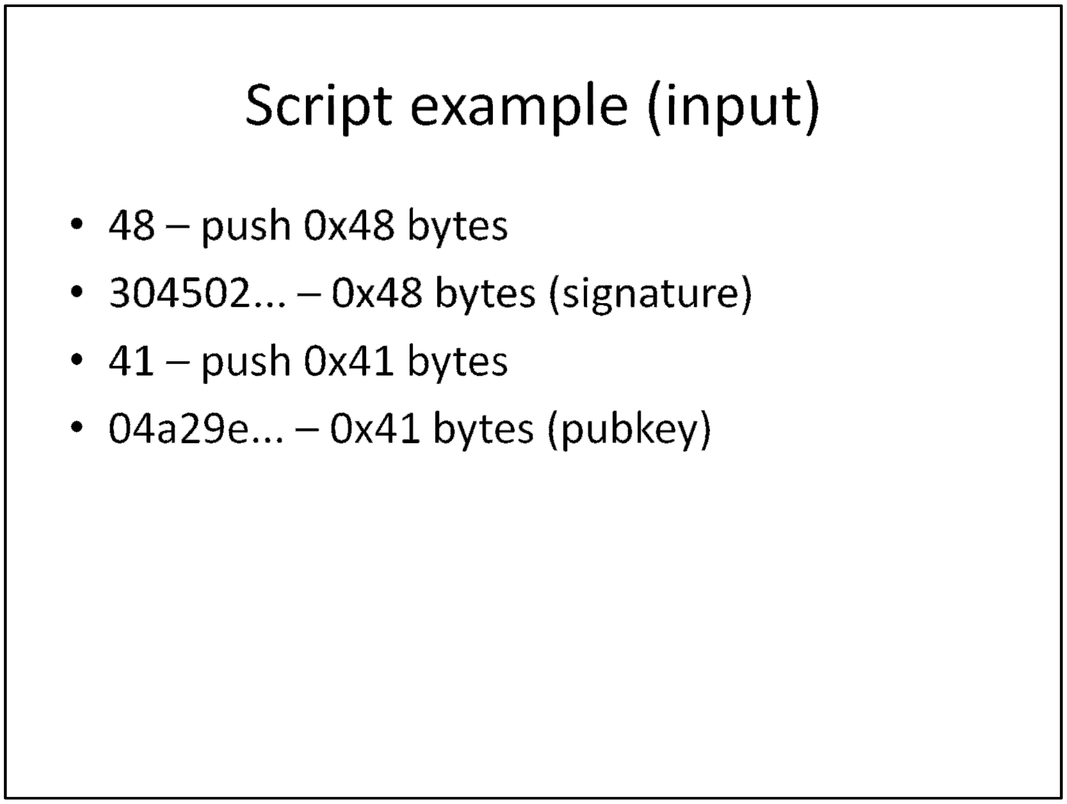
Does this pubkey hash to the hash recorded in advance?



Script example (output)

- 76 – OP_DUP (duplicate the top stack item)
- A9 – OP_HASH160 (hash with SHA256, then RIPEMD-160)
- 14 – PUSH_14 (push the next 20 words)
- 39ebeac446feb2a615196f5c9efdf67fae99c369
- 88 – OP_EQUALVERIFY (abort if unequal)
- AC – OP_CHECKSIG (verify the signature)

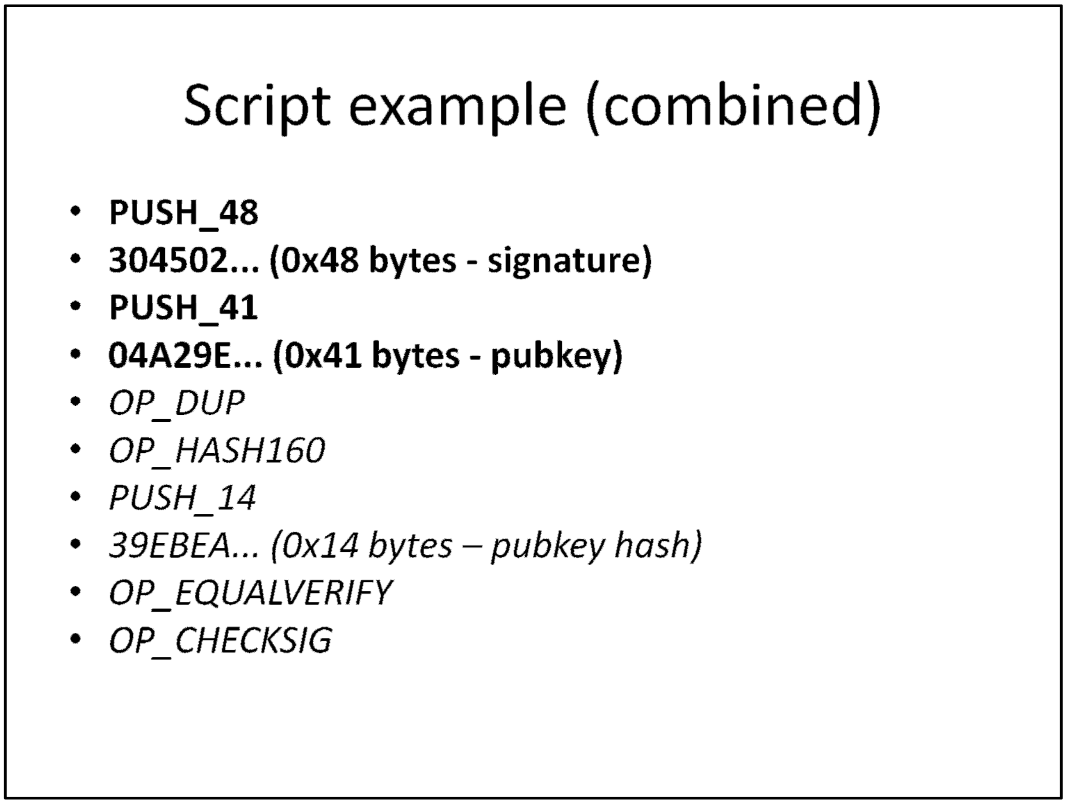
The 20 bytes pushed is the pubkey hash, extracted from an address



Script example (input)

- 48 – push 0x48 bytes
- 304502... – 0x48 bytes (signature)
- 41 – push 0x41 bytes
- 04a29e... – 0x41 bytes (pubkey)

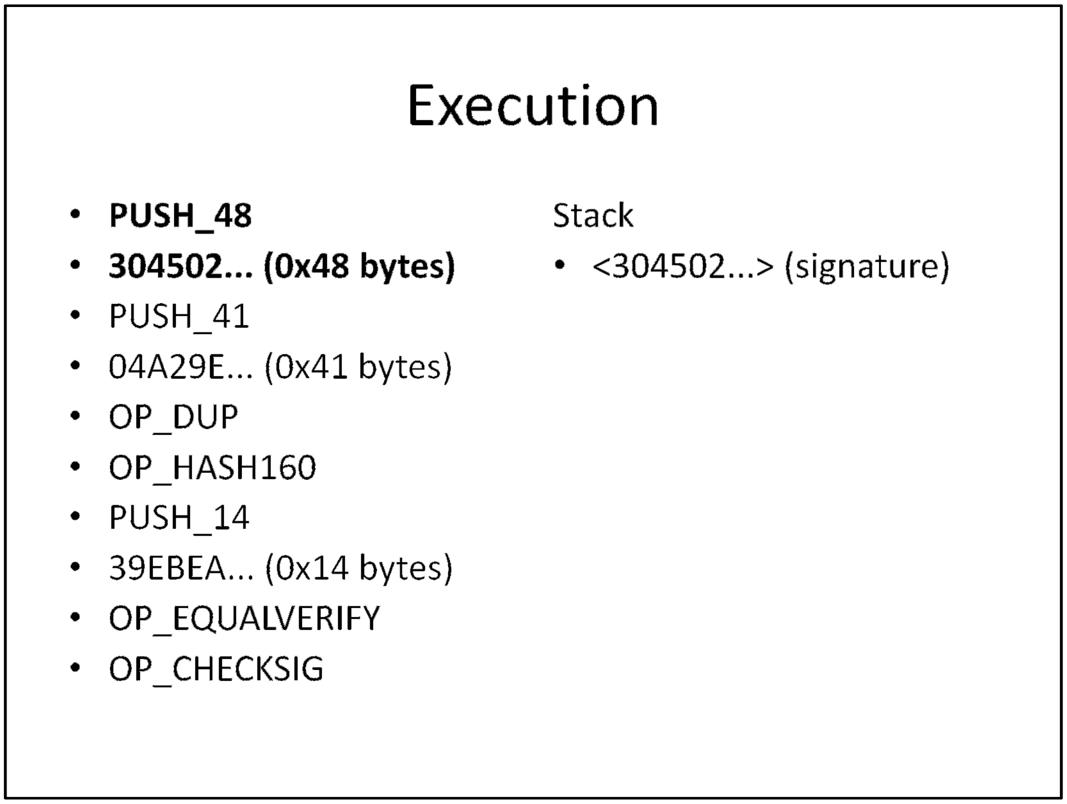
The 0x48 bytes is the signature
The 0x41 bytes is the pubkey



Script example (combined)

- **PUSH_48**
- **304502... (0x48 bytes - signature)**
- **PUSH_41**
- **04A29E... (0x41 bytes - pubkey)**
- *OP_DUP*
- *OP_HASH160*
- *PUSH_14*
- *39EBEA... (0x14 bytes – pubkey hash)*
- *OP_EQUALVERIFY*
- *OP_CHECKSIG*

Spender goes first. The person setting conditions for spending goes last.



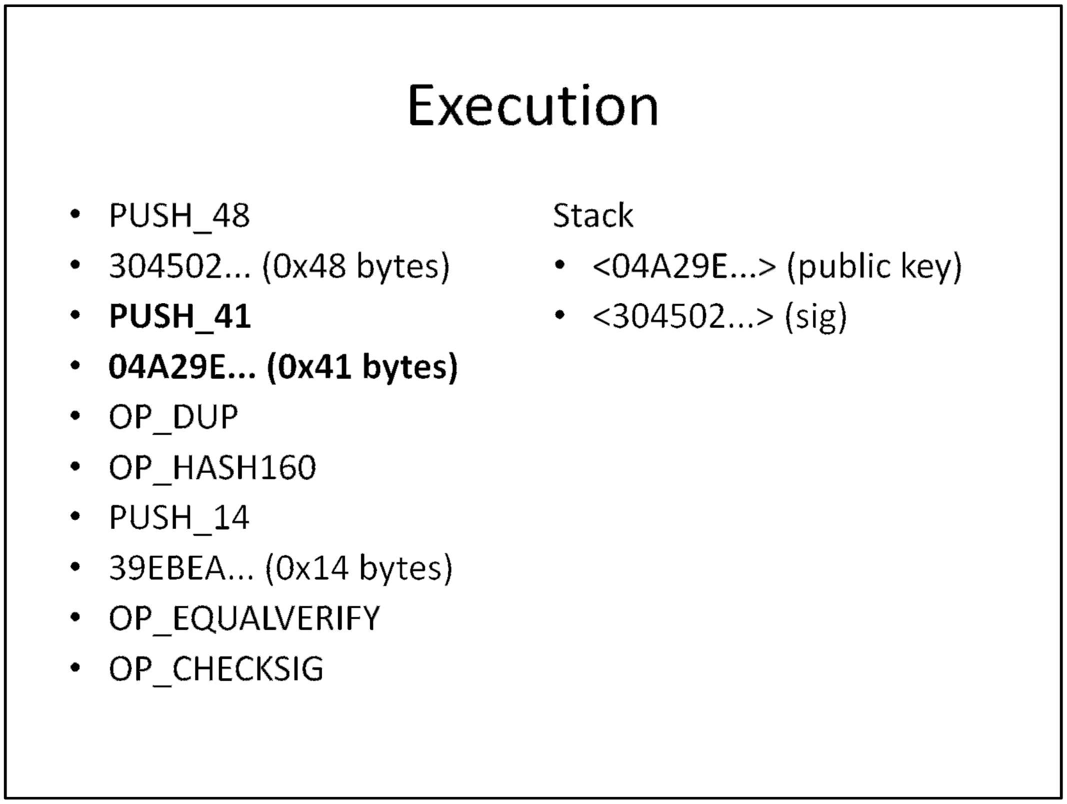
Execution

- **PUSH_48**
- **304502... (0x48 bytes)**
- PUSH_41
- 04A29E... (0x41 bytes)
- OP_DUP
- OP_HASH160
- PUSH_14
- 39EBEA... (0x14 bytes)
- OP_EQUALVERIFY
- OP_CHECKSIG

Stack

- <304502...> (signature)

First we push the signature onto the stack
The signature has a structure, basically the OpenSSL format for a EC signature



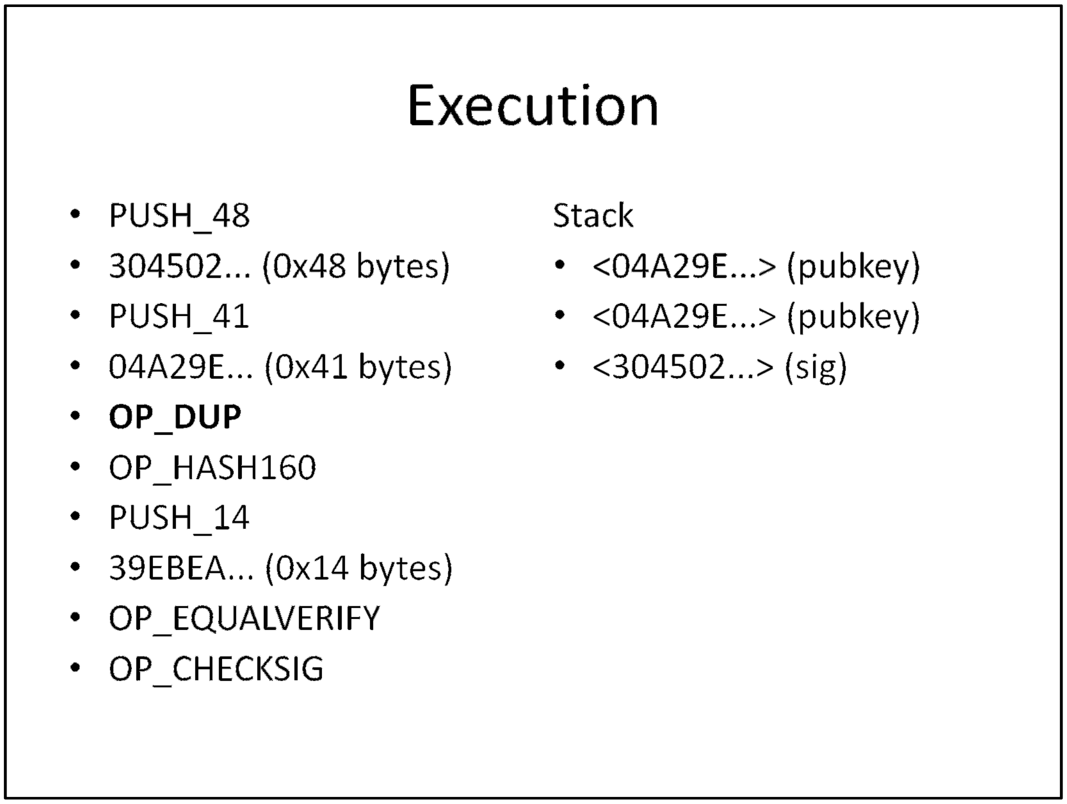
Execution

- PUSH_48
- 304502... (0x48 bytes)
- **PUSH_41**
- **04A29E... (0x41 bytes)**
- OP_DUP
- OP_HASH160
- PUSH_14
- 39EBEA... (0x14 bytes)
- OP_EQUALVERIFY
- OP_CHECKSIG

Stack

- <04A29E...> (public key)
- <304502...> (sig)

Second, we push the public key. This too has a structure, it is the encoded form from before

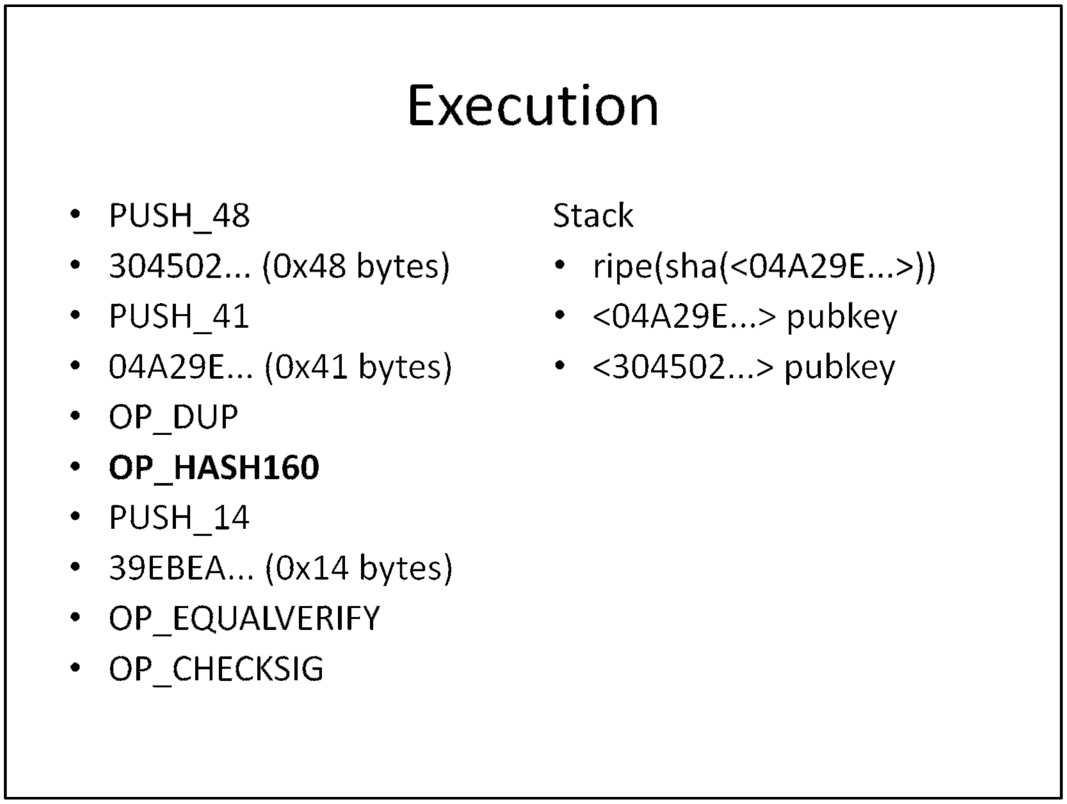


Execution

- PUSH_48
- 304502... (0x48 bytes)
- PUSH_41
- 04A29E... (0x41 bytes)
- **OP_DUP**
- OP_HASH160
- PUSH_14
- 39EBEA... (0x14 bytes)
- OP_EQUALVERIFY
- OP_CHECKSIG

Stack
- <04A29E...> (pubkey)
- <04A29E...> (pubkey)
- <304502...> (sig)

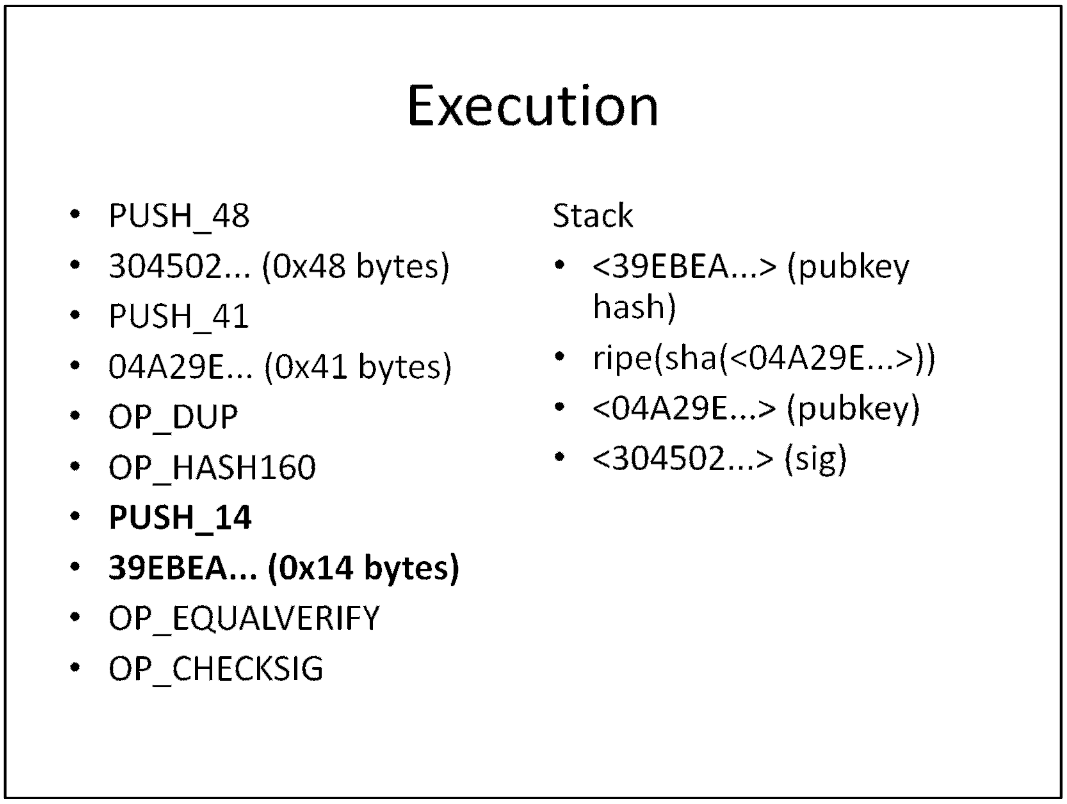
We duplicate the top item, for reasons that will become clear in a moment


Execution

- PUSH_48
- 304502... (0x48 bytes)
- PUSH_41
- 04A29E... (0x41 bytes)
- OP_DUP
- **OP_HASH160**
- PUSH_14
- 39EBEA... (0x14 bytes)
- OP_EQUALVERIFY
- OP_CHECKSIG

Stack
- ripe(sha(<04A29E...>))
- <04A29E...> pubkey
- <304502...> pubkey

Now we hash the pubkey



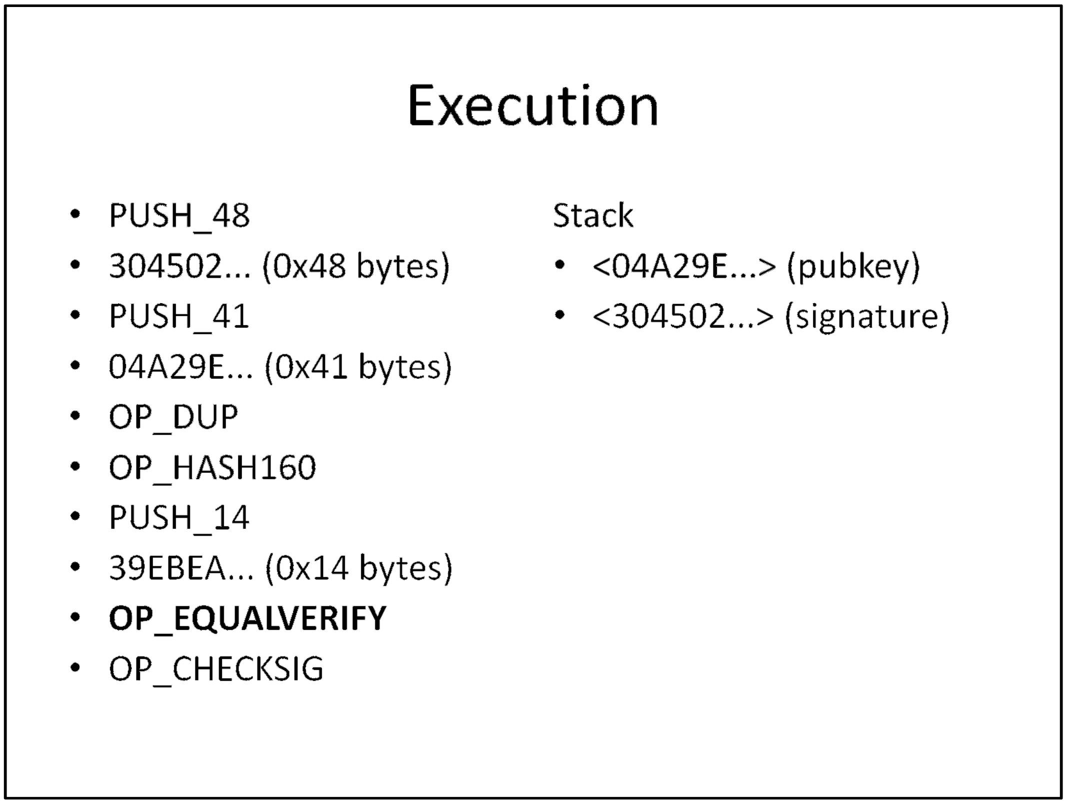
Execution

- PUSH_48
- 304502... (0x48 bytes)
- PUSH_41
- 04A29E... (0x41 bytes)
- OP_DUP
- OP_HASH160
- **PUSH_14**
- **39EBEA... (0x14 bytes)**
- OP_EQUALVERIFY
- OP_CHECKSIG

Stack

- <39EBEA...> (pubkey hash)
- ripe(sha(<04A29E...>))
- <04A29E...> (pubkey)
- <304502...> (sig)

Now we push the pubkey hash that was specified long ago when the txout was created



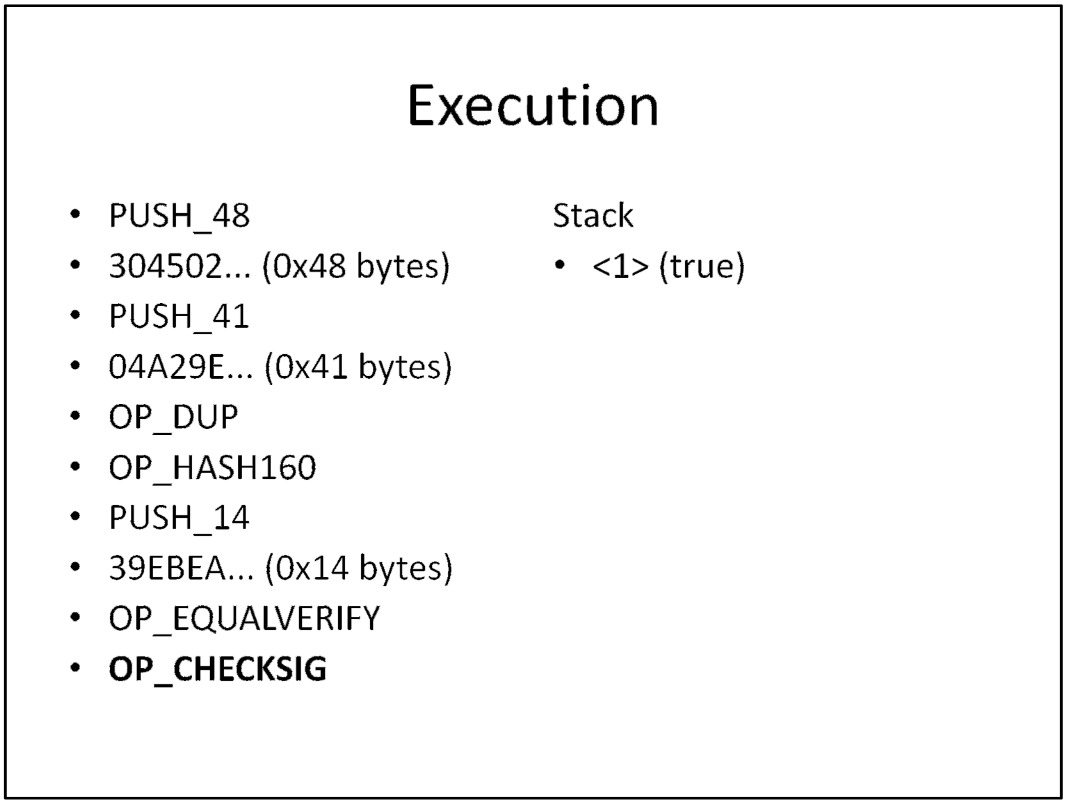
Execution

- PUSH_48
- 304502... (0x48 bytes)
- PUSH_41
- 04A29E... (0x41 bytes)
- OP_DUP
- OP_HASH160
- PUSH_14
- 39EBEA... (0x14 bytes)
- **OP_EQUALVERIFY**
- OP_CHECKSIG

Stack
- <04A29E...> (pubkey)
- <304502...> (signature)

OP_EQUALVERIFY does two things. It compares the top two stack items, and leaves either true (1) or false (0) on the stack in their place
Then it does a verify, if the top stack item is not true, it aborts execution

This lets the sender use the hash of the pubkey, carried as an address, rather than the pubkey itself. This is smaller, and more secure. If EC is broken some day, the hash is unlikely to also be broken



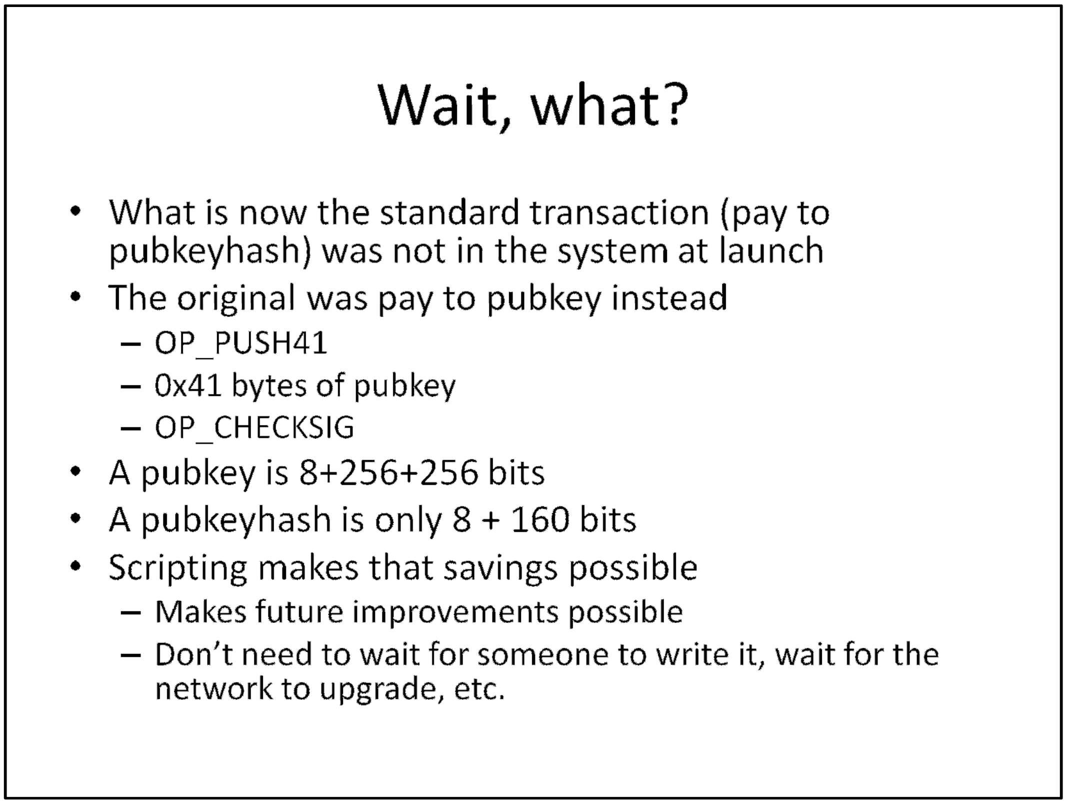
Execution

- PUSH_48
- 304502... (0x48 bytes)
- PUSH_41
- 04A29E... (0x41 bytes)
- OP_DUP
- OP_HASH160
- PUSH_14
- 39EBEA... (0x14 bytes)
- OP_EQUALVERIFY
- **OP_CHECKSIG**

Stack

- <1> (true)

And finally, OP_CHECKSIG does the whole signature calculation using the pubkey and compares it to the stored pubkey



Wait, what?

- What is now the standard transaction (pay to pubkeyhash) was not in the system at launch
- The original was pay to pubkey instead
  - OP_PUSH41
  - 0x41 bytes of pubkey
  - OP_CHECKSIG
- A pubkey is 8+256+256 bits
- A pubkeyhash is only 8 + 160 bits
- Scripting makes that savings possible
  - Makes future improvements possible
  - Don't need to wait for someone to write it, wait for the network to upgrade, etc.

Pubkey compression (y recovery) makes the savings a little less amazing
still gain security from not exposing your pubkey until redemption
Standard transaction types and disabled opcodes negated much of the advantage of scripting, but only temporarily

More

- P2SH/multisig
- payment protocol
- mining hardware
- transaction malleability
- signature malleability
- economics
- hardware wallets
- exchanges
- privacy
- coinjoin
- JSON RPC API