today
new types of scripts
MAST
taproot
graftroot
script types
mostly P2PKH or segwit equivalent

OP_DUP OP_HASH160 <pkh>
OP_EQUALVERIFY OP_CHECKSIG

OP_0 <pkh>

(segwit saves 3 bytes)
script types
P2SH or segwit equivalent

P2SH: OP_HASH160 <sh> OP_EQUAL

P2WSH: OP_0 <sh>

(distinguished from P2WPKH by data size (20 vs 32 bytes))

mostly used for multisig
script types
multisig:
OP_2 <pkA> <pkB> <pkC> OP_3
OP_CHECKMULTISIG
to spend:
OP_0 <sigA> <sigC>
output vs input size

pay to pubkey:

<pk> OP_CHECKSIG

34 bytes in output script (+10), but saves 33 bytes in signature! Overall 23 bytes smaller!
output vs input size
keep output sizes small as they are in the utxo DB. Need to be randomly read.

Signatures not in DB, only blocks, linear read and latency is OK
output vs input size

similarly, could put full scripts (like multisig) in the output field

space savings overall, but better to keep output size small
big scripts
what if we want really big scripts
2 of 3 multisig, just show all 3 keys, 33 bytes of extra data
2 of 50 multisig...?
big scripts commit, only reveal part of commitment
...the cause of, and solution to, all a blockchain's problems!
merkle trees!
MAST
merkelized abstract syntax tree
make every opcode a leaf in a tree
perhaps overkill, simpler is "P2SMR"
pay to script merkle root
MAST
make a bunch of scripts
make a merkle tree of them
send to the root
MAST
to spend, reveal which you're spending
MAST
to spend, reveal
which you're
spending
MAST
to spend, reveal which you're spending
and reveal the path to the root
MAST
to spend, reveal which you're spending
and reveal the path to the root
MAST
to spend, reveal which you're spending
and reveal the path to the root
MAST to spend, reveal which you're spending and reveal the path to the root
MAST for big multisig
in the case of 2 of 50, it's
50 choose 2 = 1225 scripts,
tree height 11
proof size 11*32 = 352 bytes
raw is 50*33 = 1650 bytes
MAST for big multisig

25 of 50? 50 choose 25 = ~100T scripts, tree height 47

proof size 22*32 = 1504 bytes

raw is 50*33 = 1650 bytes

not much better. Also have to compute 200 trillion hashes.
MAST deployment

P2SMR, or tail call?

tail call: if there are 2 items left on the stack, treat the top as the MR, and the bottom as the proof & arguments
intermission

$1 << 7$ sec timeout
OP_RETURN
seems unconnected...
people use OP_RETURN to put data in the blockchain.
But why?
OP_RETURN
seems unconnected...
people use OP_RETURN to put data in the blockchain.
But why?
to prove it's there
0 byte OP_RETURN
want to prove knowledge of some data before a blockheight
with 0 bytes overhead...
0 byte OP_RETURN
want to prove knowledge of some data
before a blockheight
with 0 bytes overhead...
put it in the signature!
P2CH
pay to contract hash
Poelstra like a year ago?
weird name as it's undetectable
signature is:
s = k - h(m, R)a
sG = R - h(m, R)A
P2CH

\[ s = k - h(m, R)a \]

\[ k = j + h(\text{data}, jG)G \]

\[ s = j + h(\text{data}, jG)G - h(m, kG)a \]

to verify, still

\[ sG = R - h(m, R)A \]
P2CH

sig: (R, s)  pubkey: A  message: m

sG = R - h(m, R)A

but signer can prove that R is not kG!

(also, never reveal k, even later)
P2CH

sig: (R, s)  pubkey: A  message: m

\[ sG = R - h(m, R)a \]

\[ R = J + h(data, J)G \]

no way to prove this after the fact

\[ J = h(data, J)G - R \ldots? J = h(J) \]
P2CH
put data inside a signature's R point
can even do it with other people's signatures! Just hand them the data, they give you the proof (just J)
OP_RETURN in 0 bytes -- nifty
taproot
ML post by Greg a few months ago uses P2CH
same equation, but somehow took us a year or two to find this :(
taproot

motivation: P2PKH and P2SH look different. Different is bad.

can use P2SH for everything?

often, scripts OR "everyone signs"
in 2 of 50 multisig... 50 of 50 is also fine
taproot
merge P2PKH and P2SH
make key J, script z.  Send to key C

C = J + hash(z, J)G
taproot

\[ C = J + \text{hash}(z, J)G \]

treat as p2pkh: sign with

c = j + \text{hash}(z, J)

treat as p2sh: reveal \((z, J), \) arguments, and run script
taproot

$P = \text{sum of everyone's keys}$

$n \text{ of } n \rightarrow 1 \text{ sig for schnorr (not ECDSA)}$

most smart contracts have an "all participants sign" clause

if everyone agrees, don't even show the contract
taproot

weird trick: can make a pubkey and prove there is no known private key

C = J + hash(z, J)G

interactive: use someone elses J

non-interactive:

show pre-image of J's x-coordinate
taproot

note that anyone can make a key and script and send to it

only pubkeys needed

which differs from the next cool thing which is...
graftroot
Maxwell, 2 months ago
Allow lots of scripts with $O(1)$ proof size
merkle proofs grow in $\log(n)$
proof that grows $O(1)$...?
graftroot
Maxwell, 2 months ago
Allow lots of scripts with $O(1)$ proof size
merkle proofs grow in $\log(n)$
proof that grows $O(1)$...?
signature
graftroot
key or script, but many scripts
send to key C
p2pkh: spend from C
p2sh: show script s, signature from C on message s, script arguments
graftroot
root key must sign every script
need to use private keys to create an address
overhead is 1 signature, to endorse the script being executed
graftroot

overhead is 1 signature, to endorse the script being executed

64 bytes? overhead is 33 bytes; can aggregate the s values (more on that next time)
graftroot

simple! more scripts can be added any time. O(1) scaling. A million scripts in 32 bytes

can be threshold of many parties

signature can be aggregated within tx
downside: interactive setup
all together
unified output script:
OP_5 <pubkey>
to spend:
all together
to spend:

<sig> P2PKH mode

<J> <script> []<args>
taproot; verify commitment, execute

<C> <sig on script> <script> []<args>
graftroot; verify sig, execute
not implemented
there's code out there, but none of this is in Bitcoin, or any coin
maybe this year? next year?
If interested... start coding it!
(Also... use cases!)
MAST vs graftroot vs both