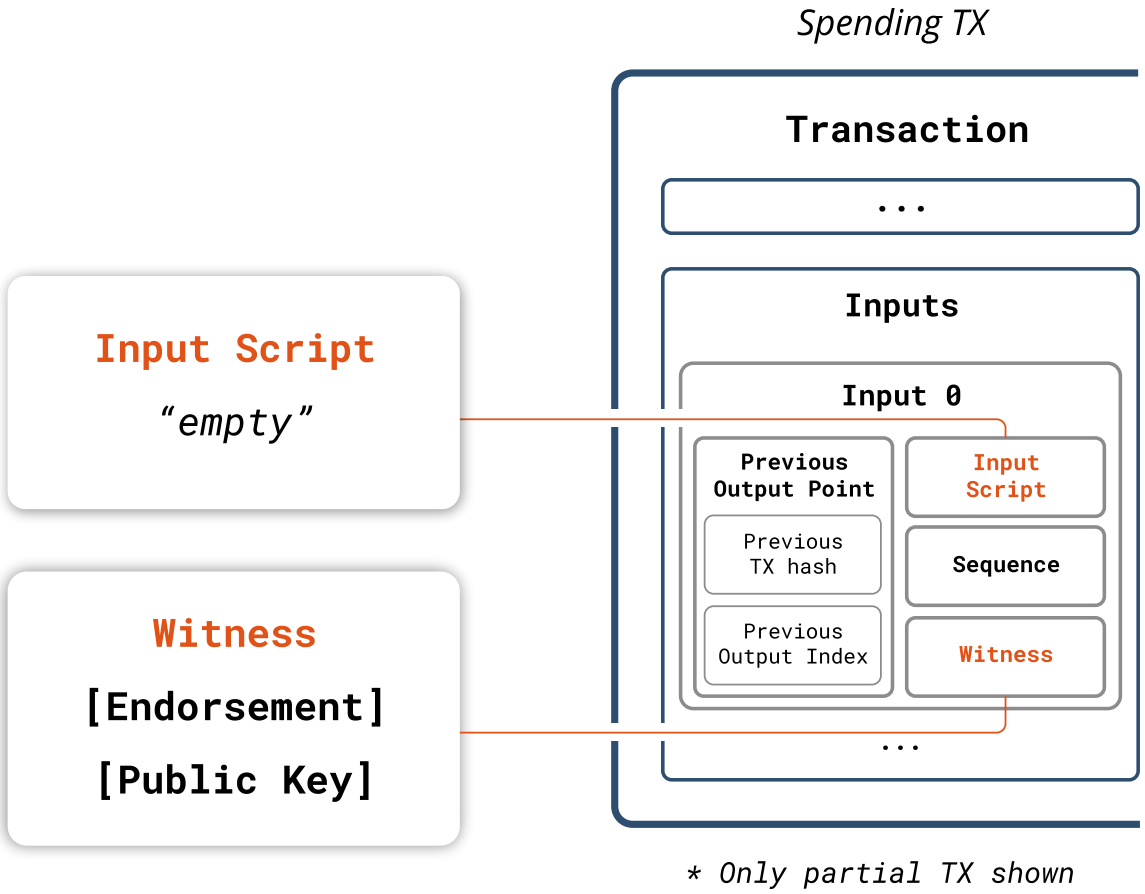
Segregated Witnesses

- Literally: Separating the unlocking script (scriptSig)
- An *output*-level concept, not transaction-level
- A softfork!
- BIPs 141, 143, 144, 145, 173
- Activation drama: UASF BIP 148, activated in August 2017
- Most important fix: transaction malleability
- Other minor cleanups
- Effective block size increase
- New address format
- New script versioning format ⇒ Taproot!

Transaction Malleability

- The possibility to change the TXID without altering its semantics
- A show-stopper for L2-solutions like LN (why?)
- How? Which part is inherently malleable without invalidating signatures?
- The signature
  - Signer: random k $\Rightarrow$ practically infinite valid signatures
  - Everybody: (r, s) a valid ECDSA signature => $(r, -s)$ also a valid (but different!) signature
  - Everybody: change data push opcodes: OP_48 == OP_4C [48] == OP_4D [4800]
    - Mt. Gox hack?

P2WPKH Transaction



The pay-to-witness-public-key-hash output can be spent without an endorsement in the input script.

Pay-to-Witness script begins with empty data push

- OP_0 pushes empty array to stack.
- Signals Version 0 witness script.

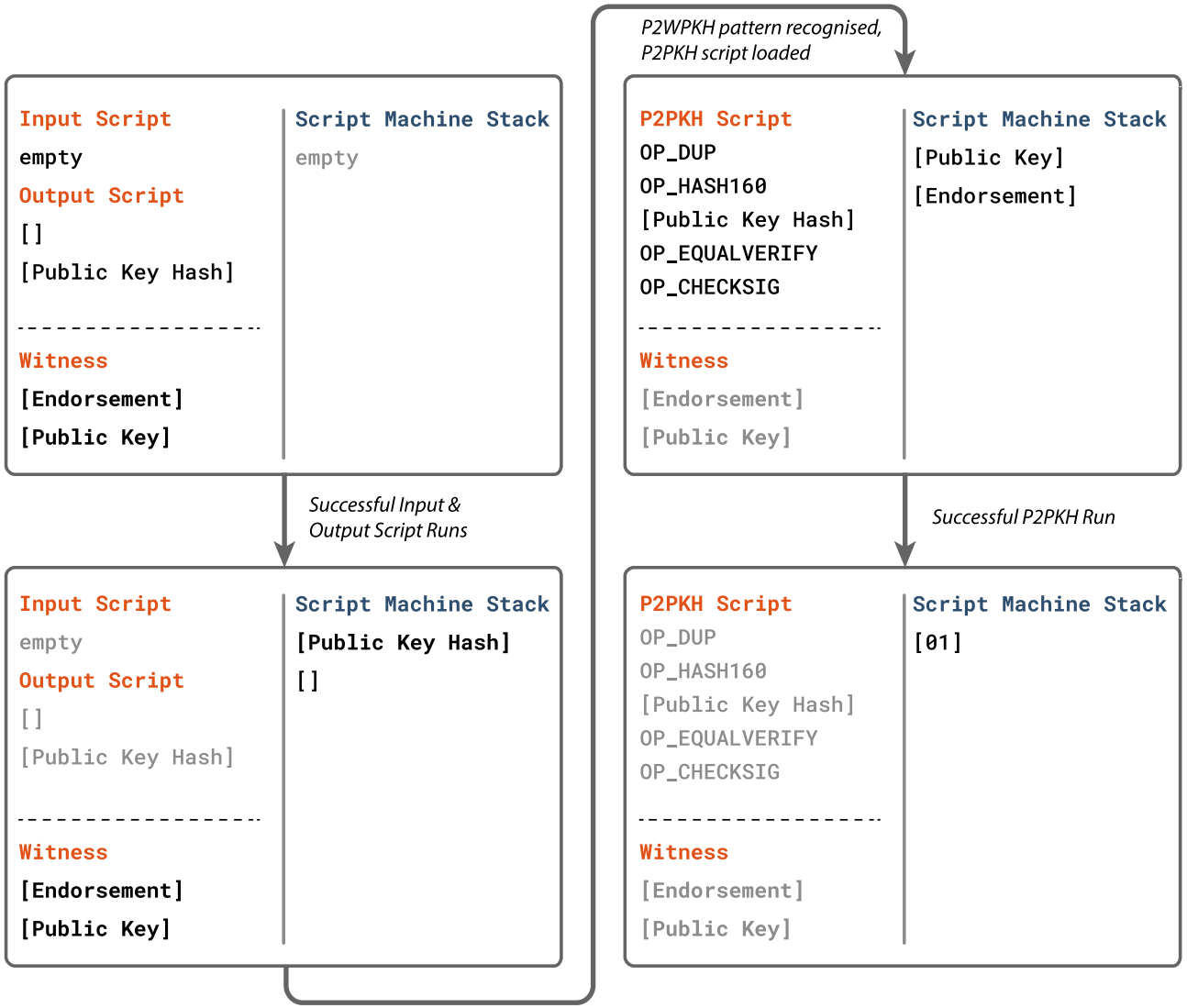
Followed by a 20-byte hash value

Spending TX input script is empty, endorsement is placed in witness

- Output is spendable by endorsement from private key of 20-Byte public key hash, equivalent to pay-to-public-key-hash output.
- Witness is an off-script/off-stack transaction element which is assessed during transaction verification.

How do unupgraded nodes interpret this output?

P2WPKH Script Run



1) Input & Output Scripts are run

- P2WPKH Witness script elements are pushed onto stack.

2) P2WPKH pattern is recognised on stack

- Pay-to-witness pattern with a 20-byte data-push is recognised as a P2WPKH script.

3) P2PKH script run

- The stack is cleared and witness elements are pushed on.
- A P2PKH script with the public key hash from the witness script is initiated and run.

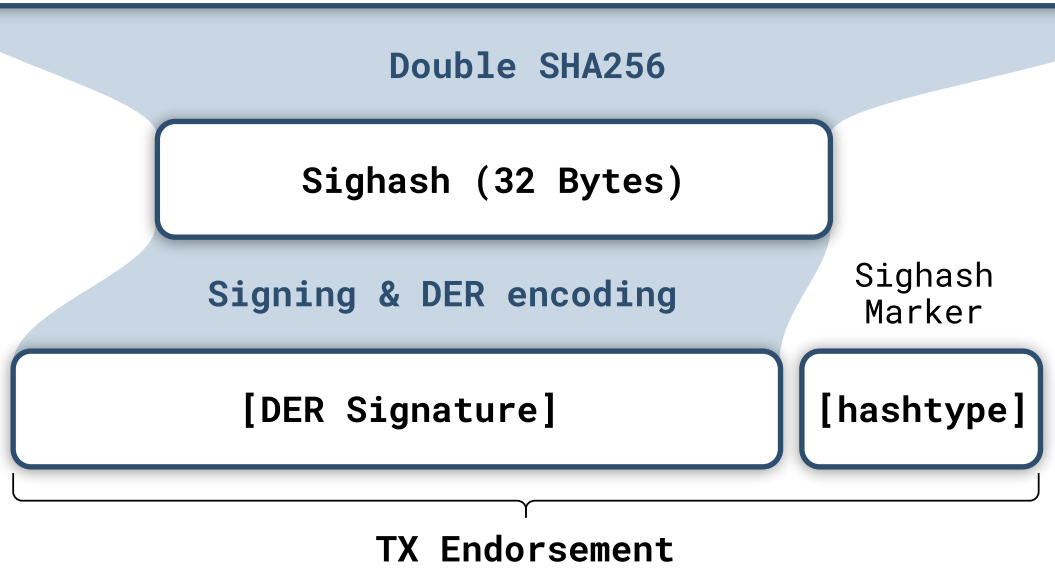
4) Final stack evaluation

Note: The pay-to-public-key-hash script is not explicitly expressed in the p2wpkh script, but rather, is implied by the pay-to-witness script pattern.

Pay-to-Witness **Endorsement**

Signature hash preimage for **input i** of transaction with **n inputs** and **m outputs**

```
[version]
[sha256([prev txid 0][prev index 0]...[prev txid n][prev index n])]
[sha256([sequence 0][sequence 1]...[sequence n])]
[prev txid i]
[script code i]
[amount of prev output i]
[sequence i]
[sha256([amount 0][output script 0]...[amount m][output script m])]
[locktime]
[hashtype]
```



BIP143 Signature Hash Algorithm for Pay-to-Witness Transactions

- Modified signature hash preimage compared to non-witness txid serialisation.
- O(n) computation time, n = length of transaction.
- For offline signers: Previous output(s) is now included.

Script Code for P2WPKH(public key) is the P2PKH(public key) Script

- Script code is part of signature hash preimage.

Commitment to Inputs and Outputs signaled by Sighash Marker

- Non-committed input, sequence and output fields are initialised to 0.

P2WSH Transaction

Input Script

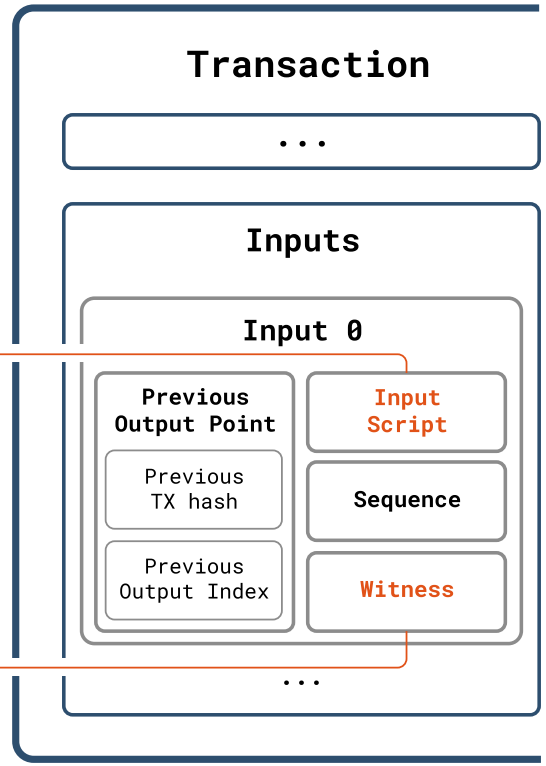
"empty"

Witness

unlocking script

[embedded script]

Spending TX



** Only partial TX shown*

Pay-to-Witness script begins with empty data push

- OP_0 pushes empty array to stack.
- Signals Version 0 witness script.

Followed by a 32-byte hash value

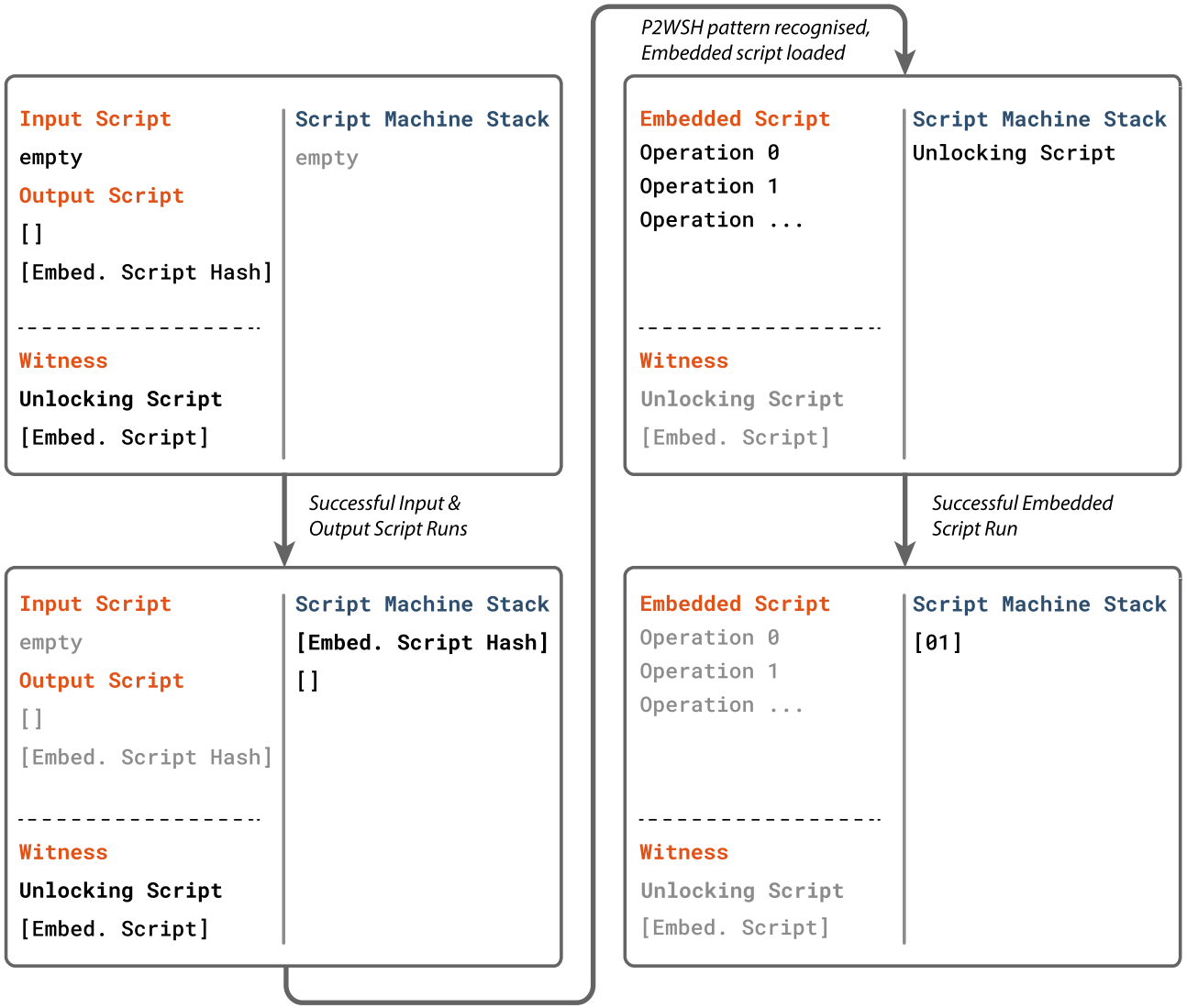
Output is spendable by Embedded Script together and its Unlocking Script

- Embedded script is sha256 hashed
- The spending TX witness includes the embedded script and its unlocking script.

Witness Elements:

- Unlocking script operations are expressed as individual witness elements.
- Embedded script is a single witness element.

P2WSH Script Run



1) Input & Output Scripts are run

- P2WSH witness script elements are pushed onto stack.

2) P2WSH pattern is recognised on stack

- Pay-to-witness pattern with a 32-byte data-push is recognised as a P2WSH script.
- Embedded script witness element is evaluated against the on-stack embedded script hash value.

3) Embedded script run

- If the embedded script hashes correctly, the stack is cleared.
- Unlocking script elements are pushed onto stack.
- Embedded script is run.

4) Final stack evaluation

Note: Evaluating the embedded script in the witness against its hash digest in the p2sh script is not explicitly expressed in the p2wsh script, but rather, is implied by the pay-to-witness script pattern.

Native SegWit Addresses

- bech32 addresses
  - BCH error detection: can also correct some errors
  - 32 lower-case characters alphabet
- P2WPKH: bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
- P2WSH: bc1qrp33g0q5c5txsp9arysrx4k6zdkfs4nce4xj0gdcccefvpysxf3qccfmv3
- Address parts
  - Human-readable part: bc / tb
  - separator: 1
  - data with six checksum chars (no "1", "b", "i", "o")

Step-wise SegWit onboarding

- Sender and receiver SegWit support?
  - Both: use it
  - None: don't use it
  - Sender only: Addresses
  - Receiver only: ?
    - P2SH! "Wrapped SegWit"
      - P2SH(P2WPKH)
      - P2SH(P2WSH)
    - Pre-BIP16 nodes vs. Pre-Segwit nodes vs. Post-Segwit nodes would all have different script runs, but all will accept the spend!

P2SH(P2WPKH) Transaction

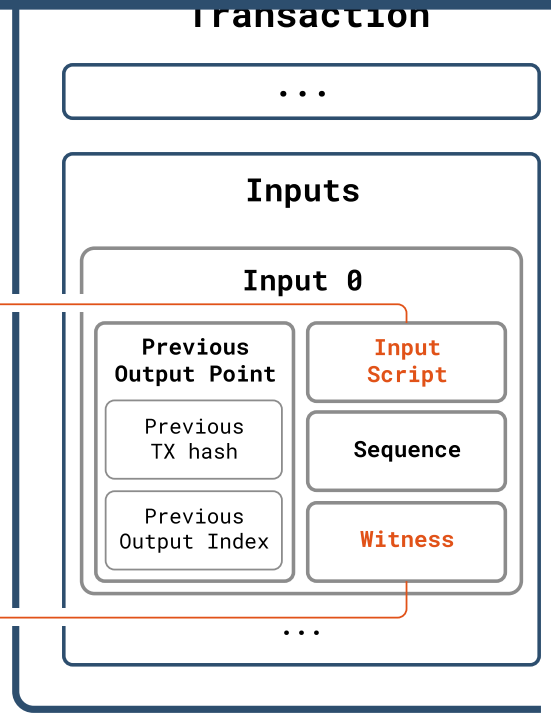
Input Script

[P2WPKH(Public Key)]

Witness

[Endorsement]

[Public Key]



** Only partial TX shown*

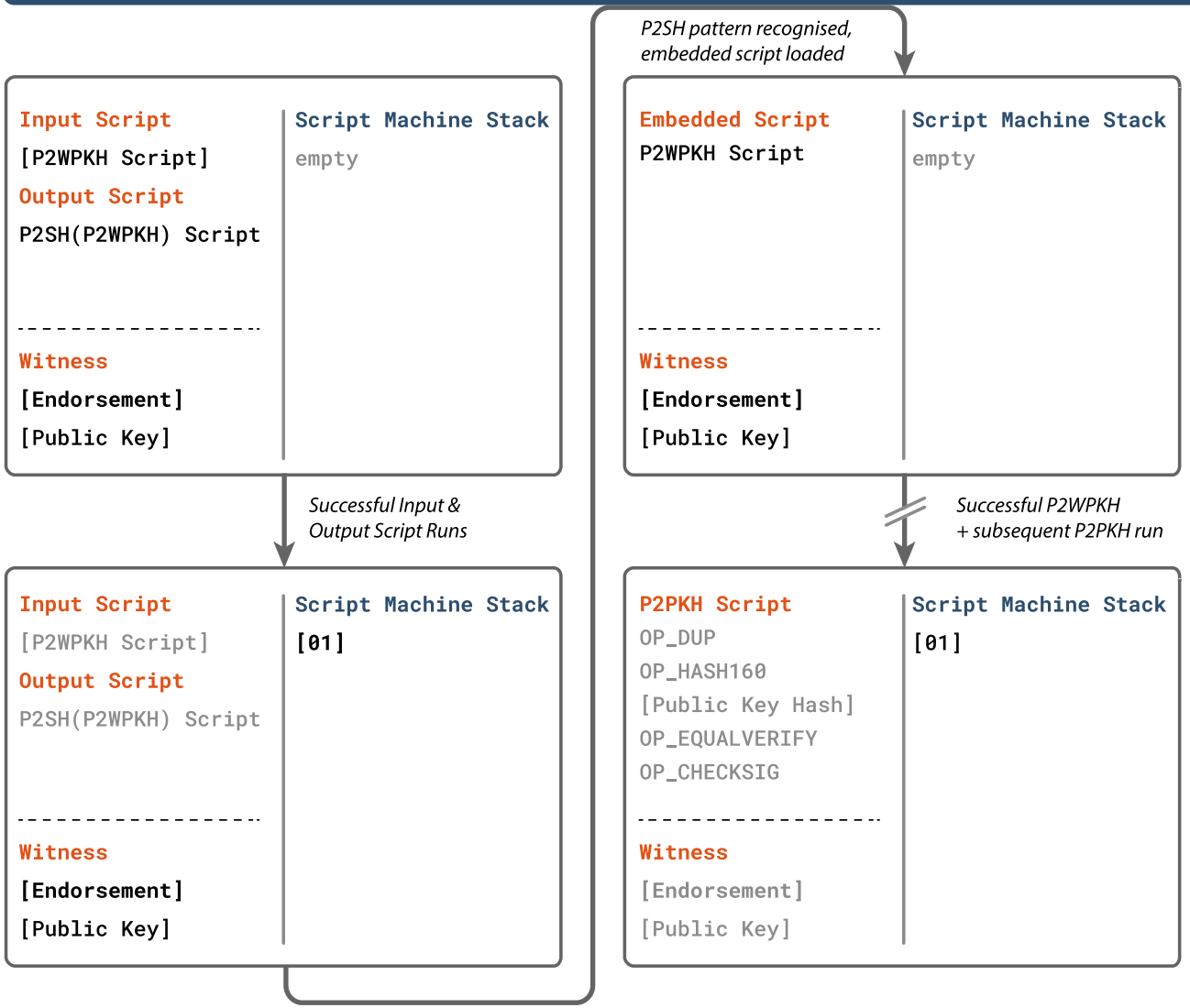
Pay-to-Witness outputs can be wrapped in a P2SH output script

- Non witness-enabled wallets can send to P2SH(P2WPKH) addresses.

Output is spendable by P2WPKH script and witness

- Input script must include P2WPKH(public key) script.
- Witness must include endorsement and public key for P2WPKH script runs.

P2SH(P2WPKH) Script Run



1) Input & Output Scripts are run

- Embedded P2WPKH in input script must hash correctly to hash digest in P2SH.

2) P2SH pattern is recognised on stack

- Embedded P2WPKH script is loaded.

3) P2WPKH & P2PKH script runs

- After a successful P2WPKH run, the witness elements are pushed onto the stack.
- Then, the P2PKH script is run.

4) Final stack evaluation

Note: See previous P2WPKH section for details on indivdiual P2WPKH and P2PKH script runs.

P2SH(P2WSH) Transaction

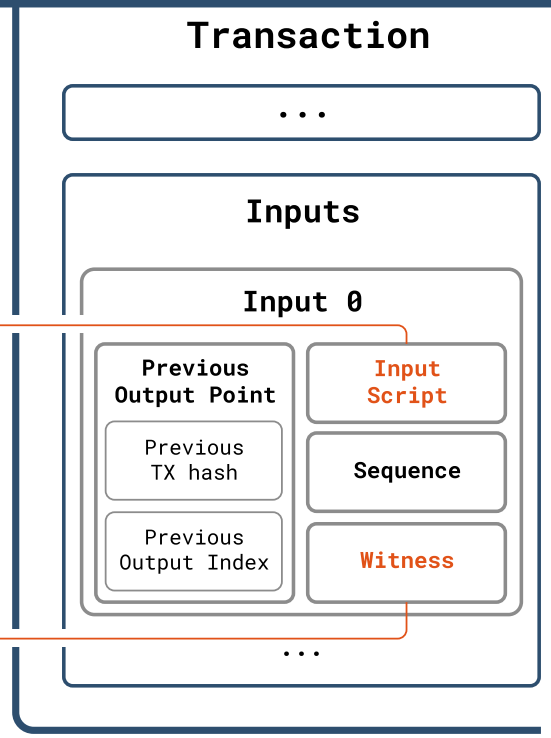
Input Script

[P2WSH(embedded script)]

Witness

unlocking script of embedded script*

[embedded script]



** Only partial TX shown*

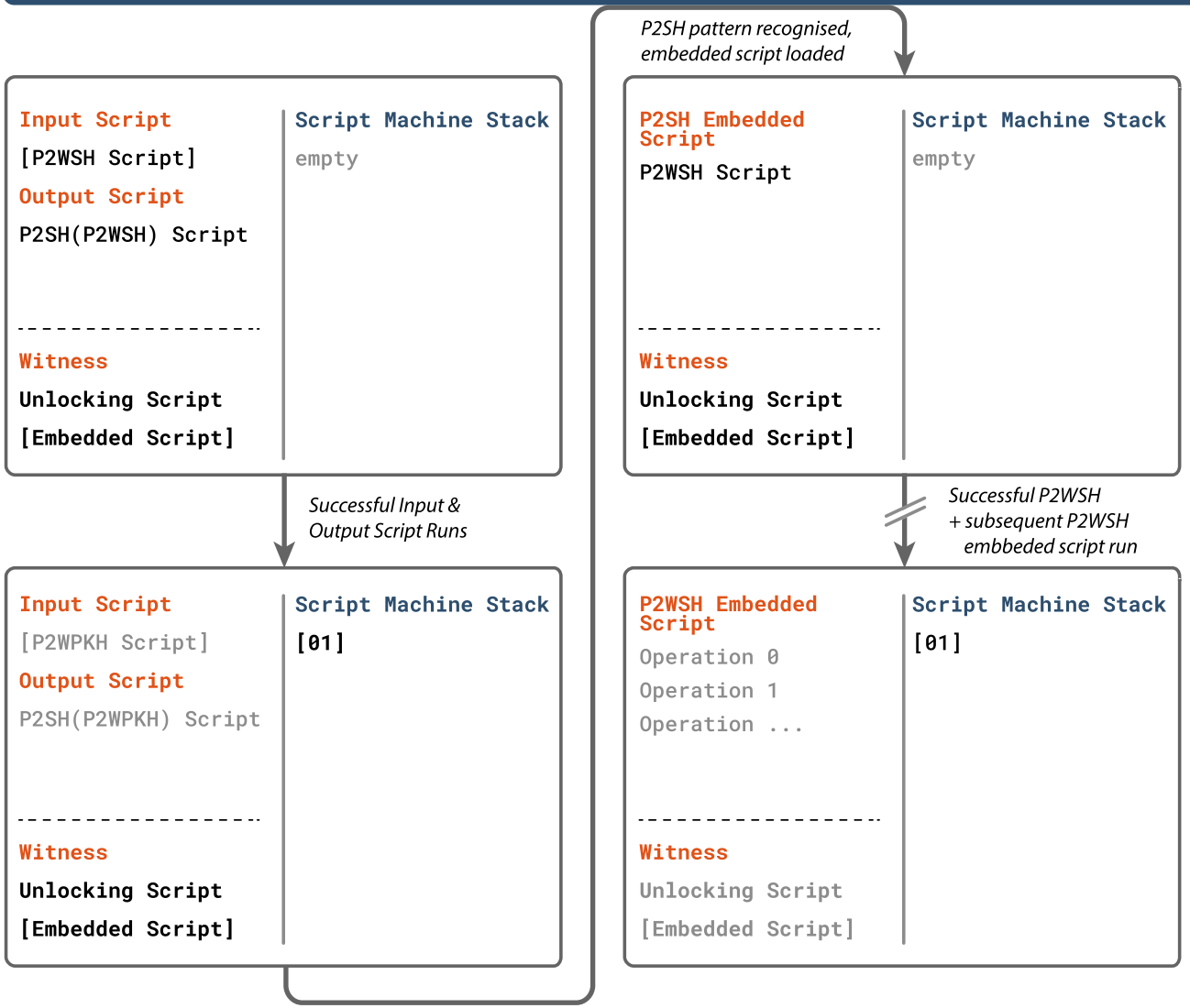
Pay-to-Witness outputs can be wrapped in a P2SH output script

- Non witness-enabled wallets can send to P2SH(P2WSH) addresses.

Output is spendable by P2WSH script and witness

- Input script must include P2WSH(Embedded Script).
- Witness must include *unlocking script operators of the embedded script as individual witness elements, as well the embedded script as a single witness element.

P2SH(P2WSH) Script Run



1) Input & Output Scripts are run

- Embedded P2WPKH in input script must hash correctly to hash digest in P2SH.

2) P2SH pattern is recognised on stack

- Embedded P2WSH script is loaded.

3) P2WPKH & P2PKH script runs

- After the successful P2WSH run, the unlocking script operations in the witness are pushed onto the stack.
- Finally, the P2WSH embedded script is run.

4) Final stack evaluation

Note: See previous P2WSH section for details on indivdiual P2WSH and P2WSH embedded script runs.

Effective block size increase

- What was Bitcoin's blocksize limit at inception?
  - ∞
  - 2010: 1M bytes
- Segregating witness data allowed for a softfork block size increase
  - Max. 4M WU
  - 1 byte of non-SegWit data: 4WU
  - 1 byte of SegWit data: 1WU

$$4(1-r)T + rT \leq 4 \times 10^6 \Rightarrow T \leq \frac{4 \times 10^6}{4-3r}$$

Table 10.2 Maximum block sizes for different ratios of witness data

r (witness bytes/total bytes)	Max total block size (bytes)
0	1,000,000
0.1	1,081,081
0.3	1,290,323
0.5	1,600,000
0.6	1,818,182
0.7	2,105,263
0.8	2,500,000

Other improvements

- Smarter hashing
  - Data hashing grew with $\mathcal{O}(n^2)$ in the number of SIGOPS
  - New digest allows for intermediate state to be reused (BIP 143)
- Output value uf UTXO being spent is committed to
  - Important for Harware wallets
- Script versioning scheme: Taproot is v1 Segwit (OP_1)