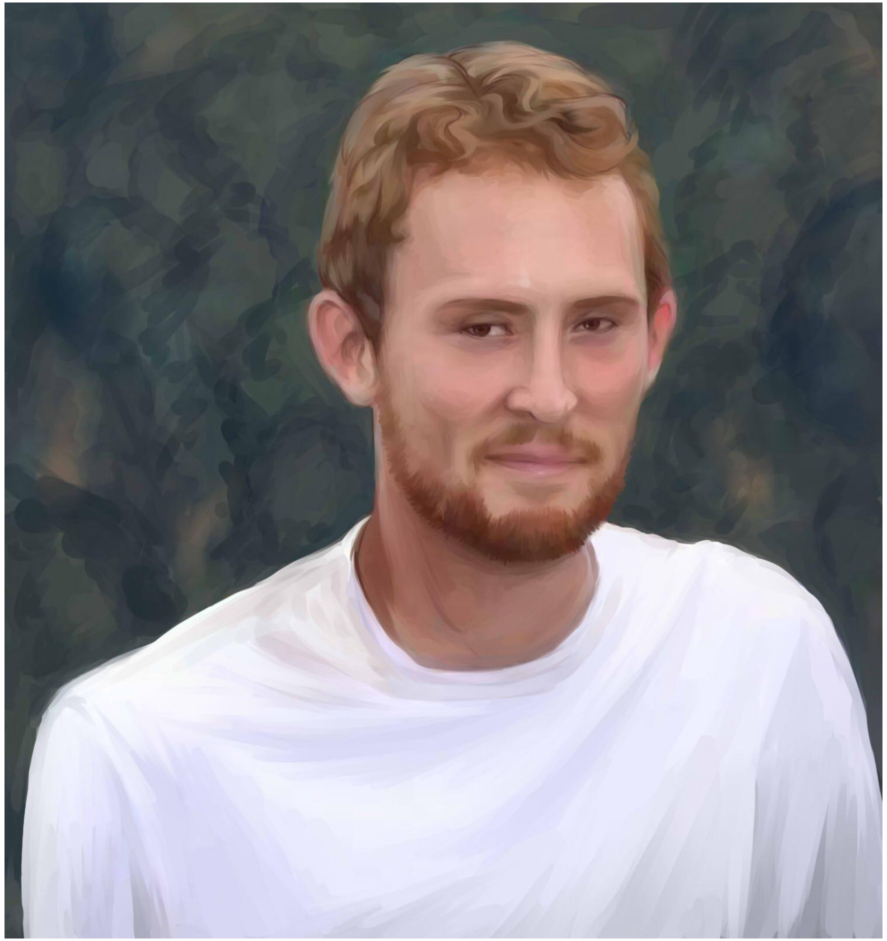
Lightning Network Smart Contracts



Alex Melville

Software Engineer
@BitGo

Digital Nomad

github.com/Melvillian

Main Goal: Expose you to SCRIPT Smart Contracts used in the Lightning Network Protocol

- Be a Core Dev
- SCRIPT Basics
- Example Commitment Transaction Output SCRIPT

Being a Core dev is easier than ever!!

- lnd (Golang)
  - Olaoluwa Osuntokun
  - https://github.com/lightningnetwork/lnd

- c-lightning (C)
  - Rusty Russell
  - https://github.com/ElementsProject/lightning
- eclair (Scala)
  - ACINQ
  - https://github.com/ACINQ/eclair

Lightning-related projects you can contribute to

https://github.com/bcongdon/awesome-lightning-network

Bitcoin’s SCRIPT

- Built-in programming language for Bitcoin
    - `OP_IF, OP_ELSE`
    - `OP_CHECKMULTISIG`
    - `OP_HASH160`

Bitcoin’s SCRIPT

- Built-in programming language for Bitcoin
  - `OP_IF, OP_ELSE`
  - `OP_CHECKMULTISIG`
  - `OP_HASH160`
- No looping, so it’s not Turing-Complete

Bitcoin’s SCRIPT

- Built-in programming language for Bitcoin
  - `OP_IF, OP_ELSE`
  - `OP_CHECKMULTISIG`
  - `OP_HASH160`
- No looping, so it’s not Turing-Complete
  - Simpler to analyze
  - Bounded execution

Bitcoin’s SCRIPT

- Built-in programming language for Bitcoin
  - `OP_IF, OP_ELSE`
  - `OP_CHECKMULTISIG`
  - `OP_HASH160`
- No looping, so it’s not Turing-Complete
  - Simpler to analyze
  - Bounded execution
- Stack based execution model
  - Again, simpler to analyze
  - Programs feel like they’re running in reverse

Bitcoin’s SCRIPT

- Built-in programming language for Bitcoin
    - `OP_IF, OP_ELSE`
    - `OP_CHECKMULTISIG`
    - `OP_HASH160`
- No looping, so it’s not Turing-Complete
    - Simpler to analyze
    - Bounded execution
- Stack based execution model
    - Again, simpler to analyze
    - Programs feel like they’re running in reverse
- Used to prove ownership of UTXO’s
    - If you’re able to run a SCRIPT program to completion, you own the bitcoin
- List of OP codes
    - https://github.com/bitcoinjs/bitcoin-ops/blob/master/index.json

What does Script Look like?

`OP_DUP OP_DUP OP_MUL OP_SUB OP_12 OP_EQUAL`

Equivalent to

$x^2 - x = 12$

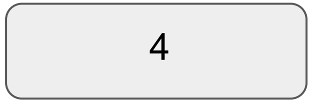
- scriptPubKey
    - This is the program
    - Example: `OP_DUP OP_DUP OP_MUL OP_SUB OP_12 OP_EQUAL`
- scriptSig
    - arguments to the program
    - Example: `4`

What does Script Look like?

- SCRIPT syntax is super simple, 2 types of things

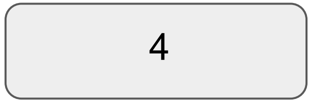
What does Script Look like?

- SCRIPT syntax is super simple, 2 types of things
- Elements:
    - Just a piece of data, like an x86 assembly number, register, or string
    - 4
    - <pubkeyHash>
    - <revocationKey>



What does Script Look like?

- SCRIPT syntax is super simple, 2 types of things
- Elements:
  - Just a piece of data, like an x86 assembly number, register, or string
  - 4
  - <pubkeyHash>
  - <revocationKey>

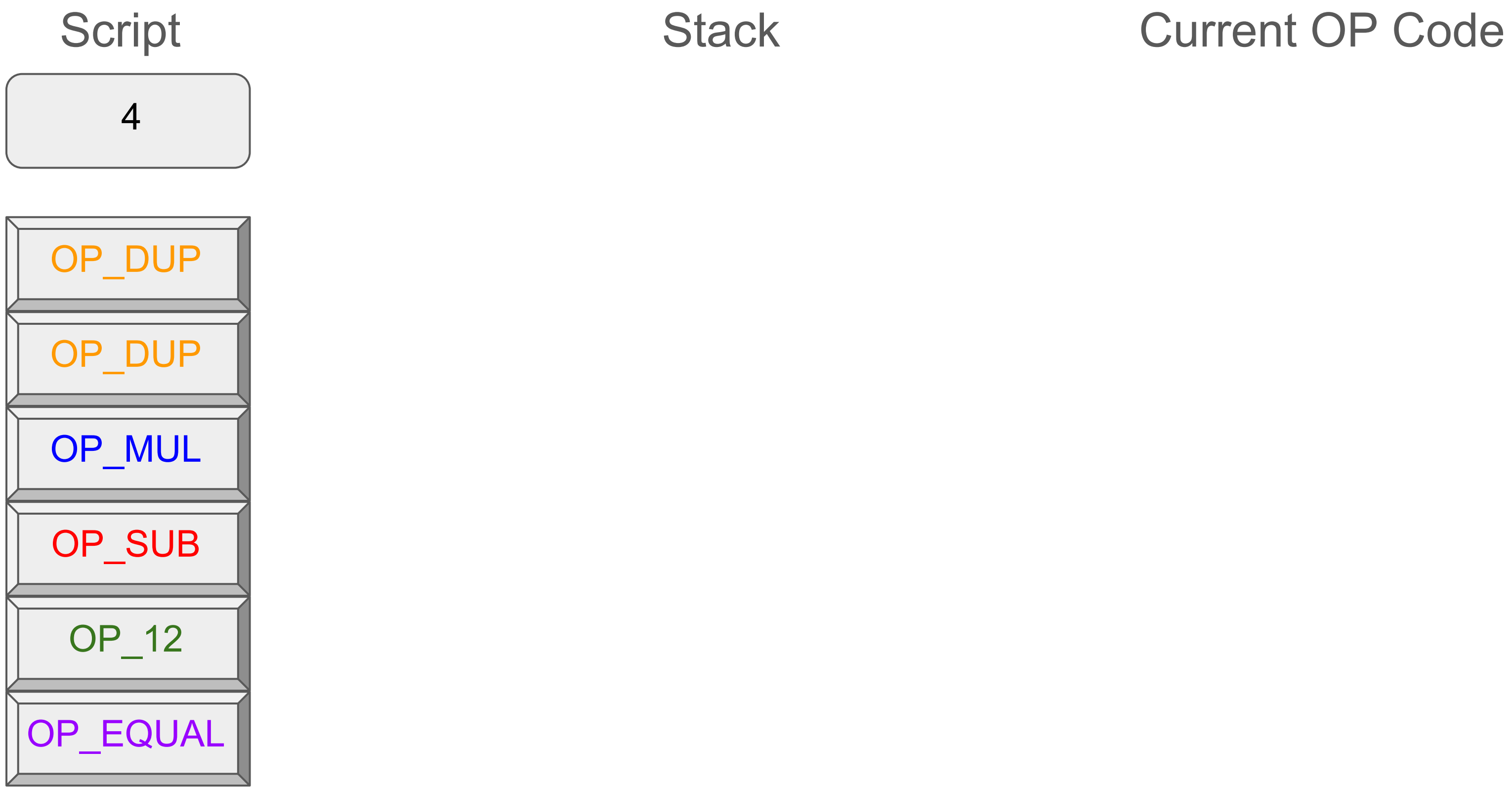


- OP codes:
  - A unit of execution, like an x86 assembly CPU instruction (mov, lea, push)
  - `OP_IF`
  - `OP_DUP`
  - `OP_CHECKMULTISIGVERIFY`

Note: I borrowed the following SCRIPT model from Jimmy Song

https://medium.com/@jimmysong/

What does Script Look like?



What does Script Look like?

Script

Stack

Current OP Code

4 } scriptSig (the program's arguments)

OP_DUP

OP_DUP

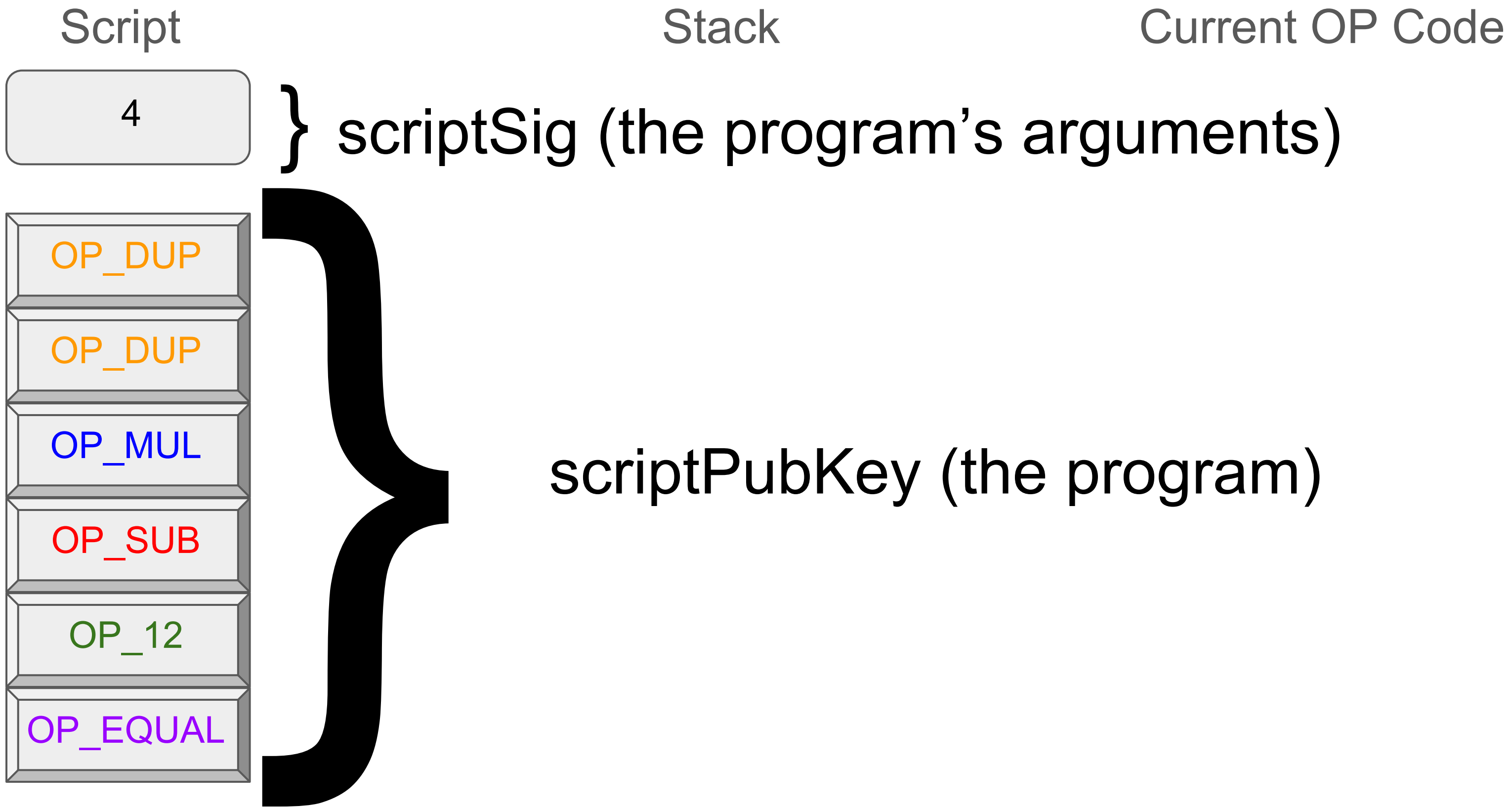
OP_MUL

OP_SUB

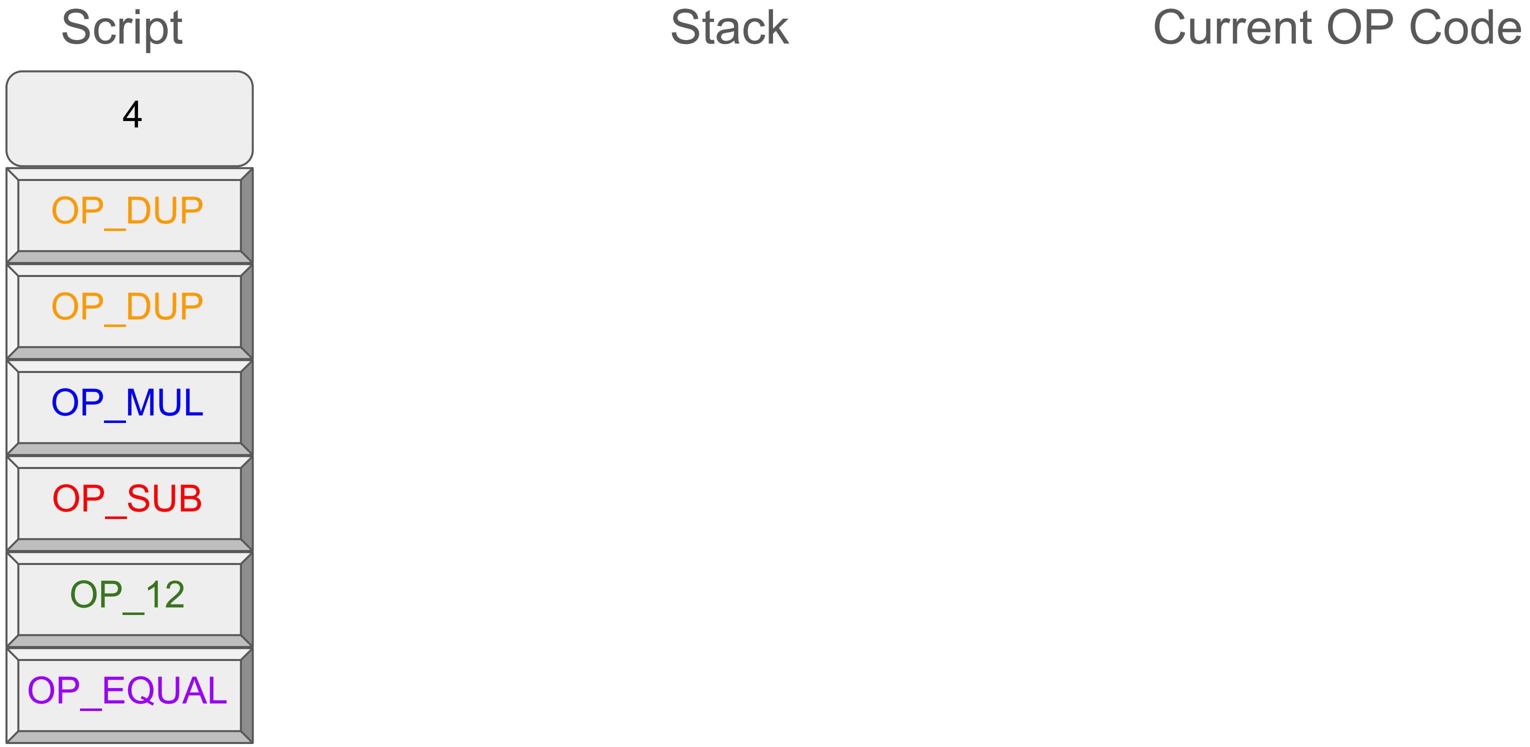
OP_12

OP_EQUAL

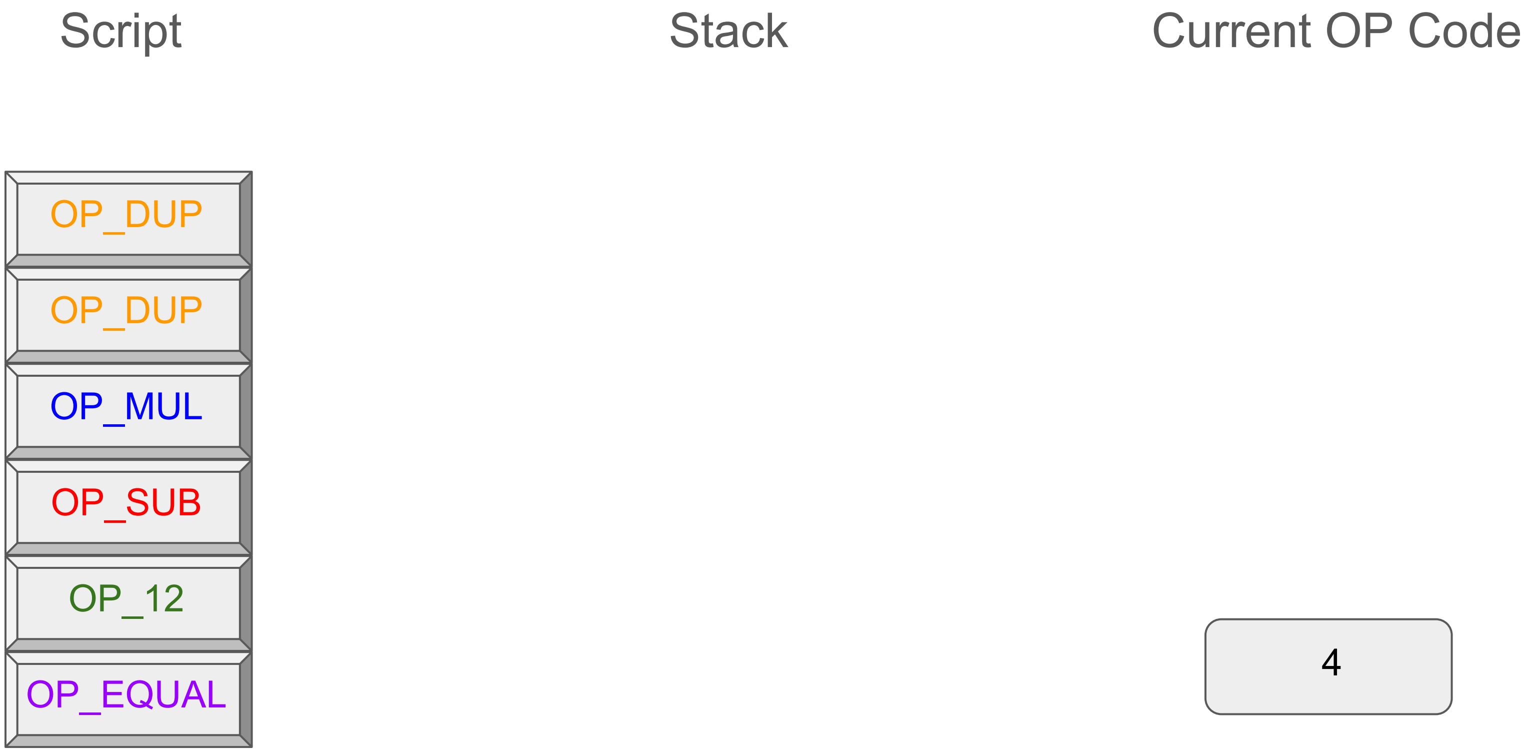
What does Script Look like?



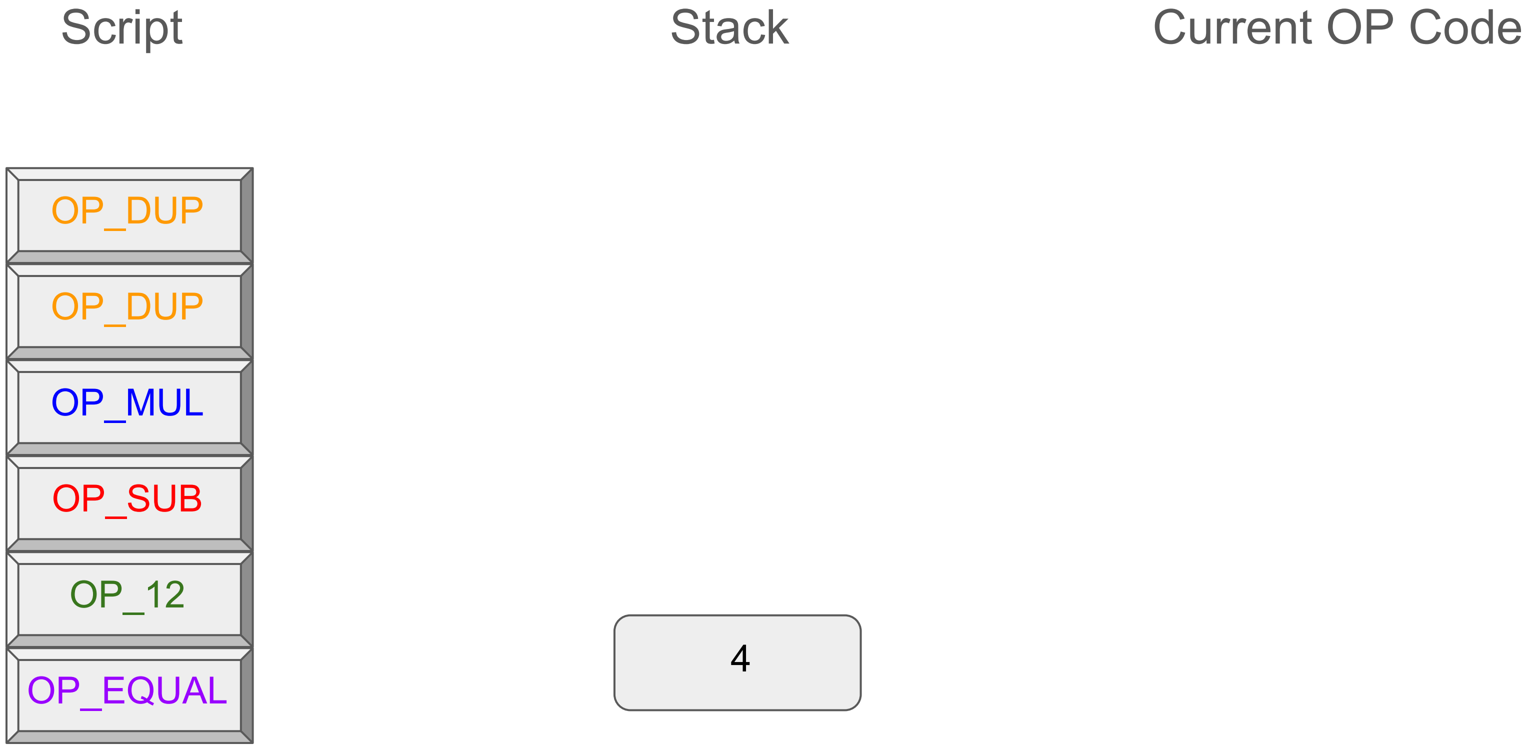
What does Script Look like?



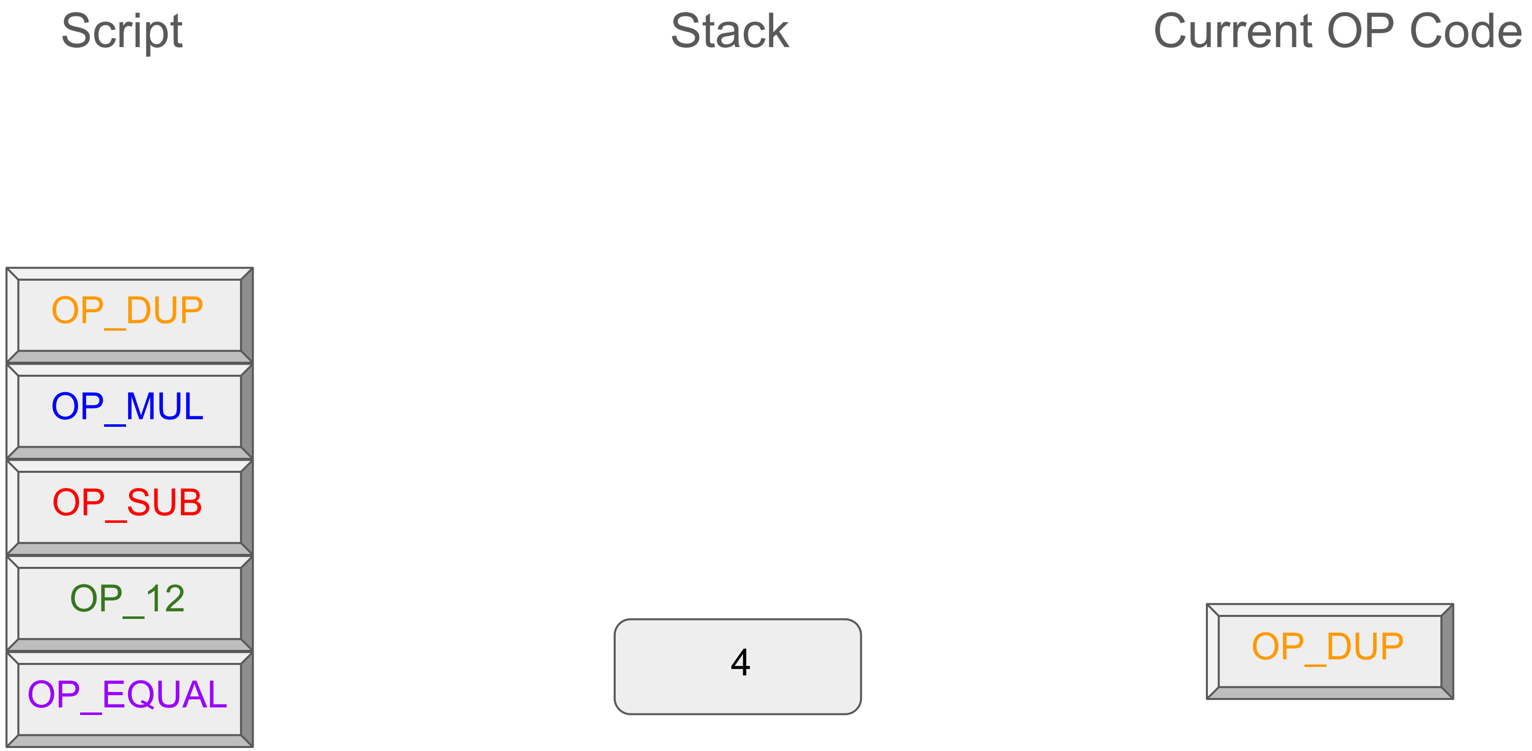
What does Script Look like?



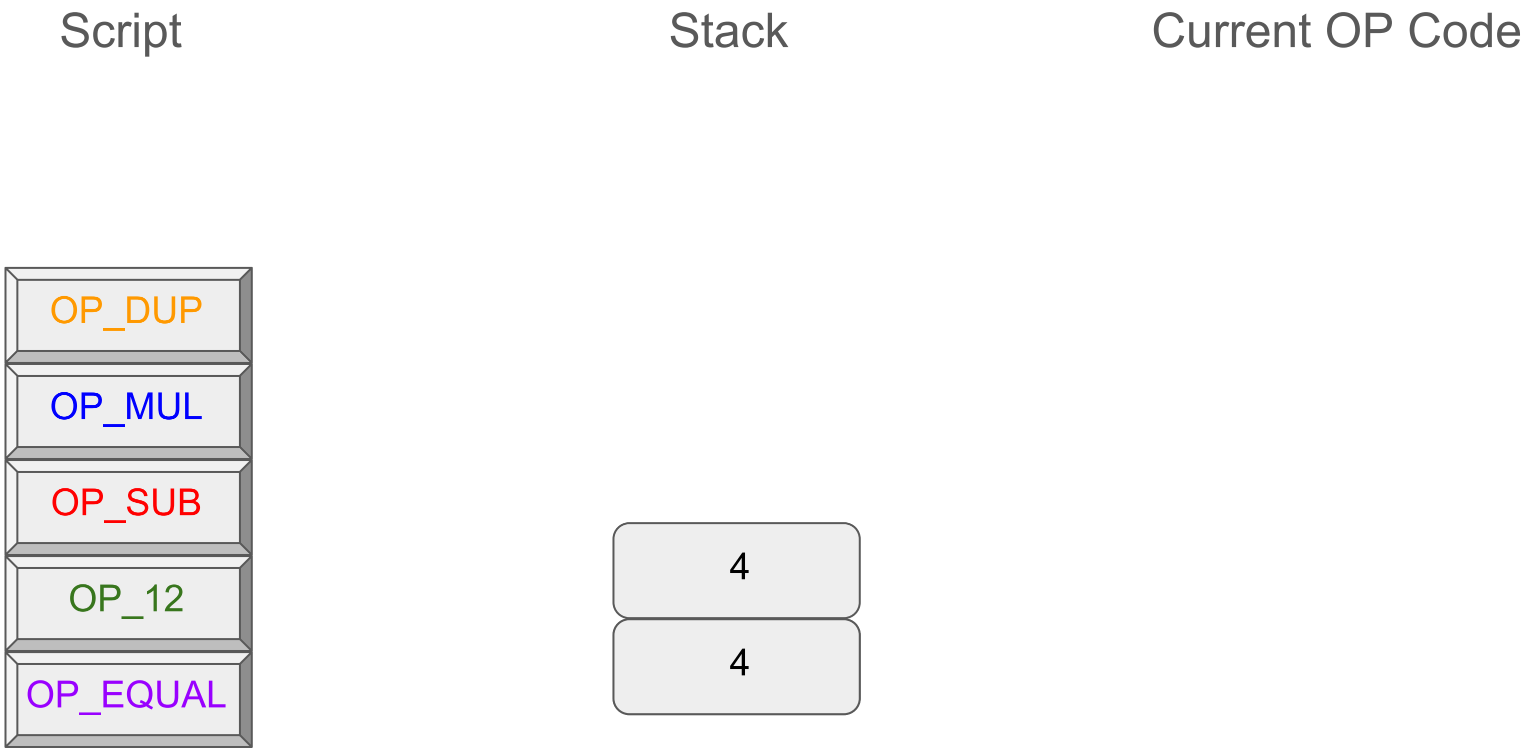
What does Script Look like?



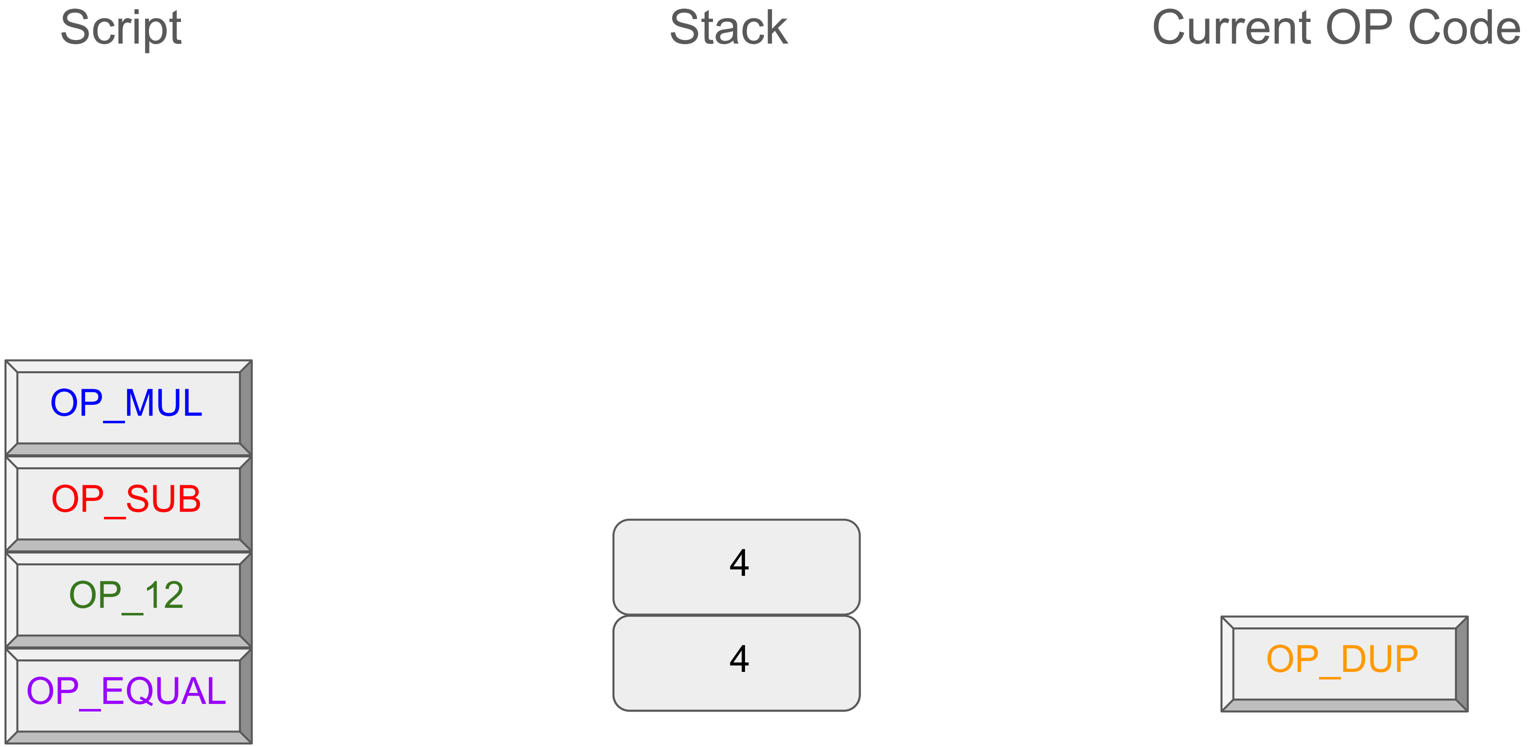
What does Script Look like?



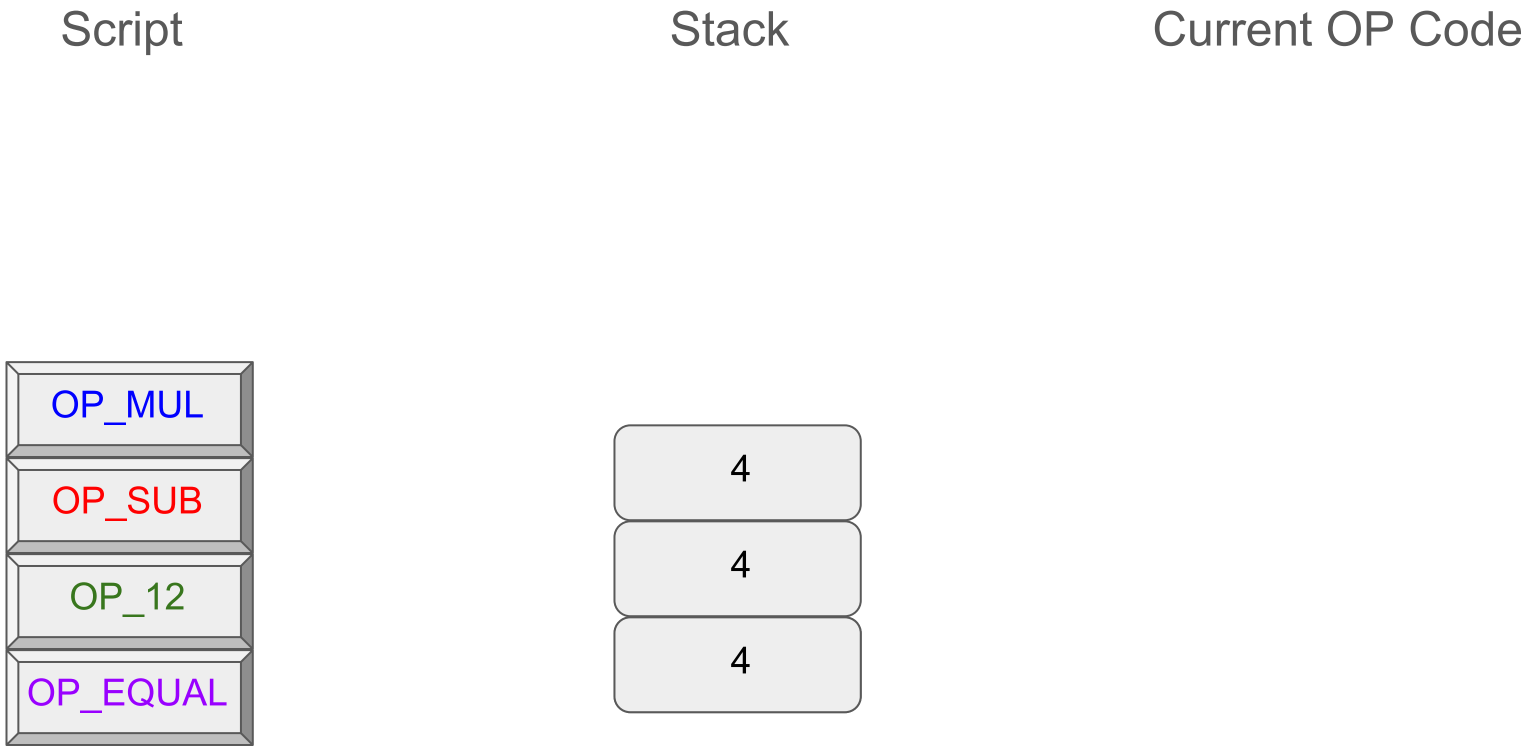
What does Script Look like?



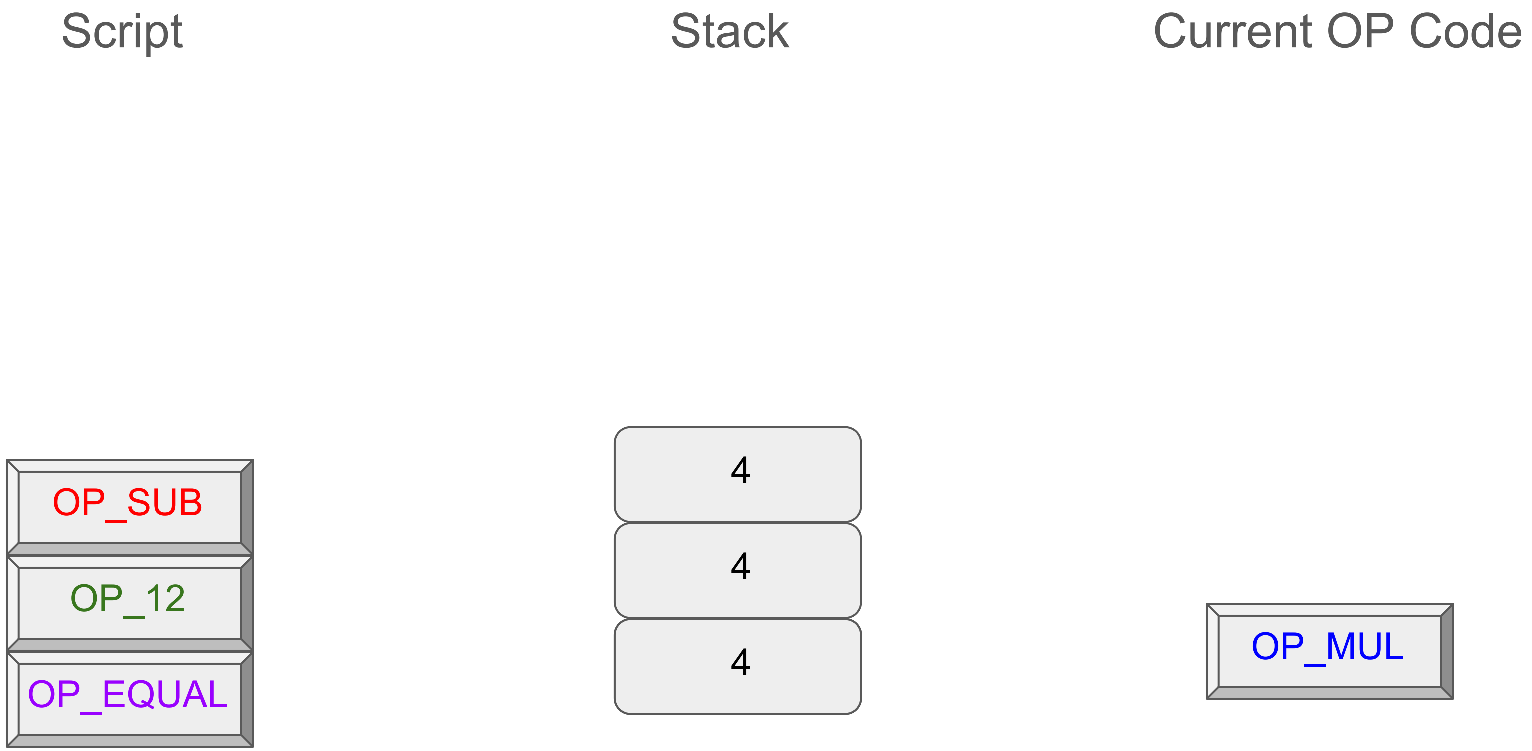
What does Script Look like?



What does Script Look like?



What does Script Look like?

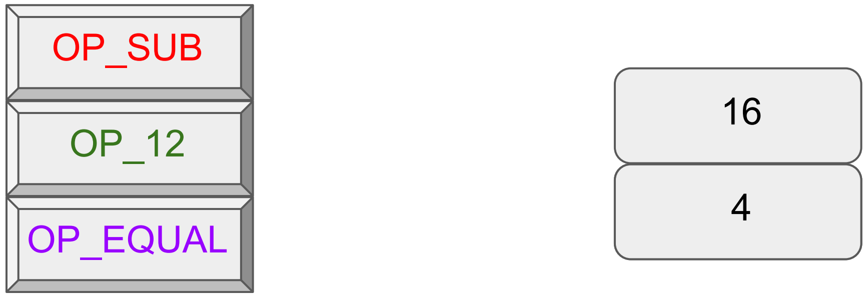


What does Script Look like?

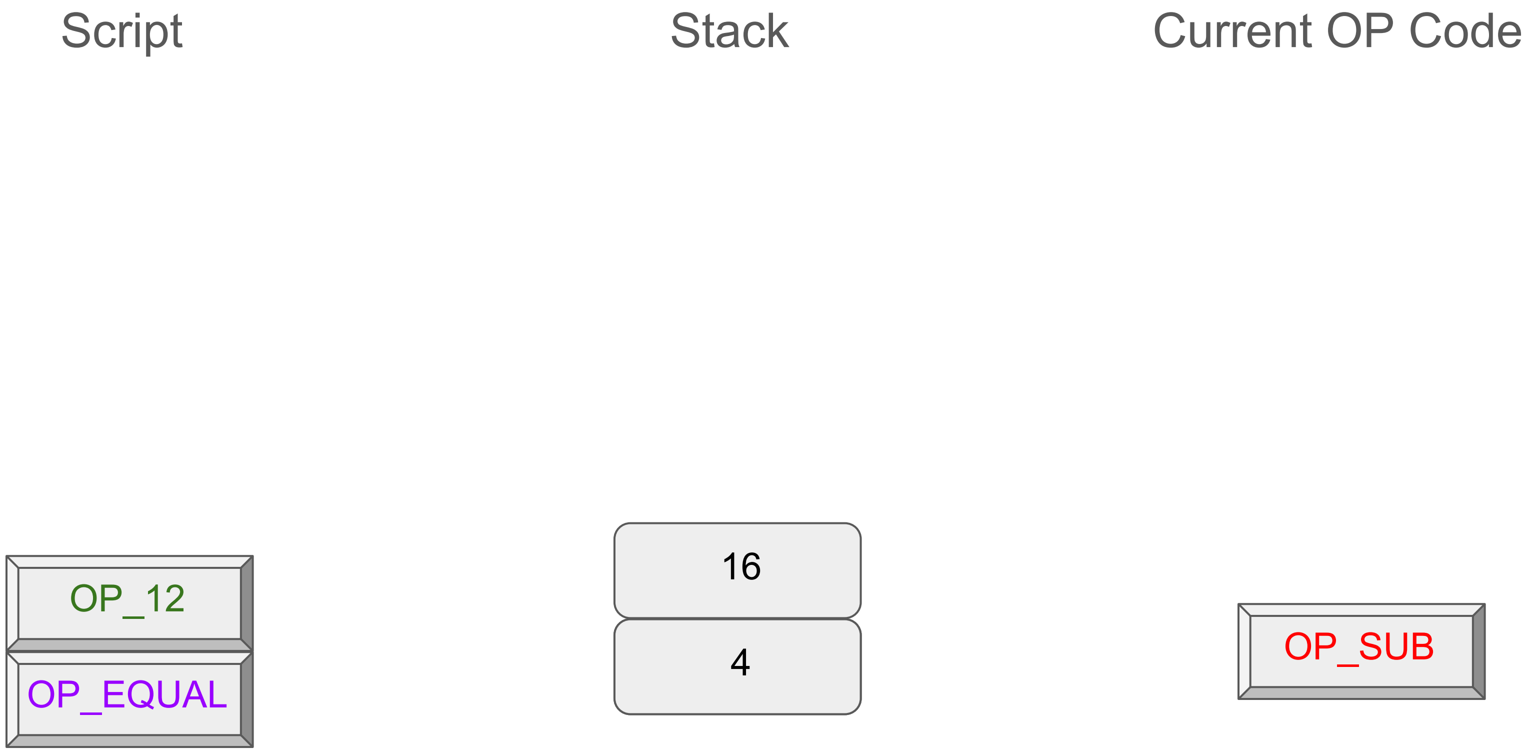
Script

Stack

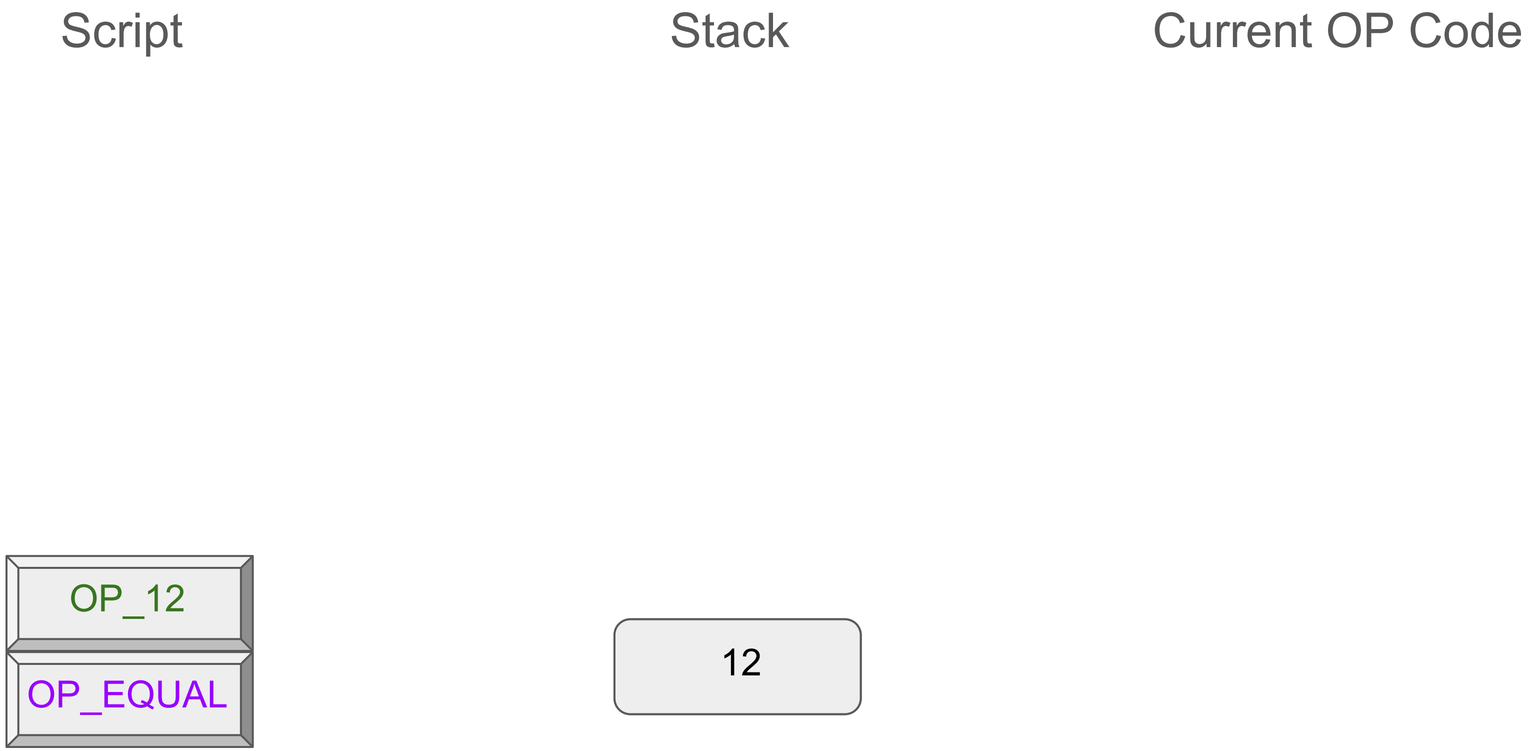
Current OP Code



What does Script Look like?



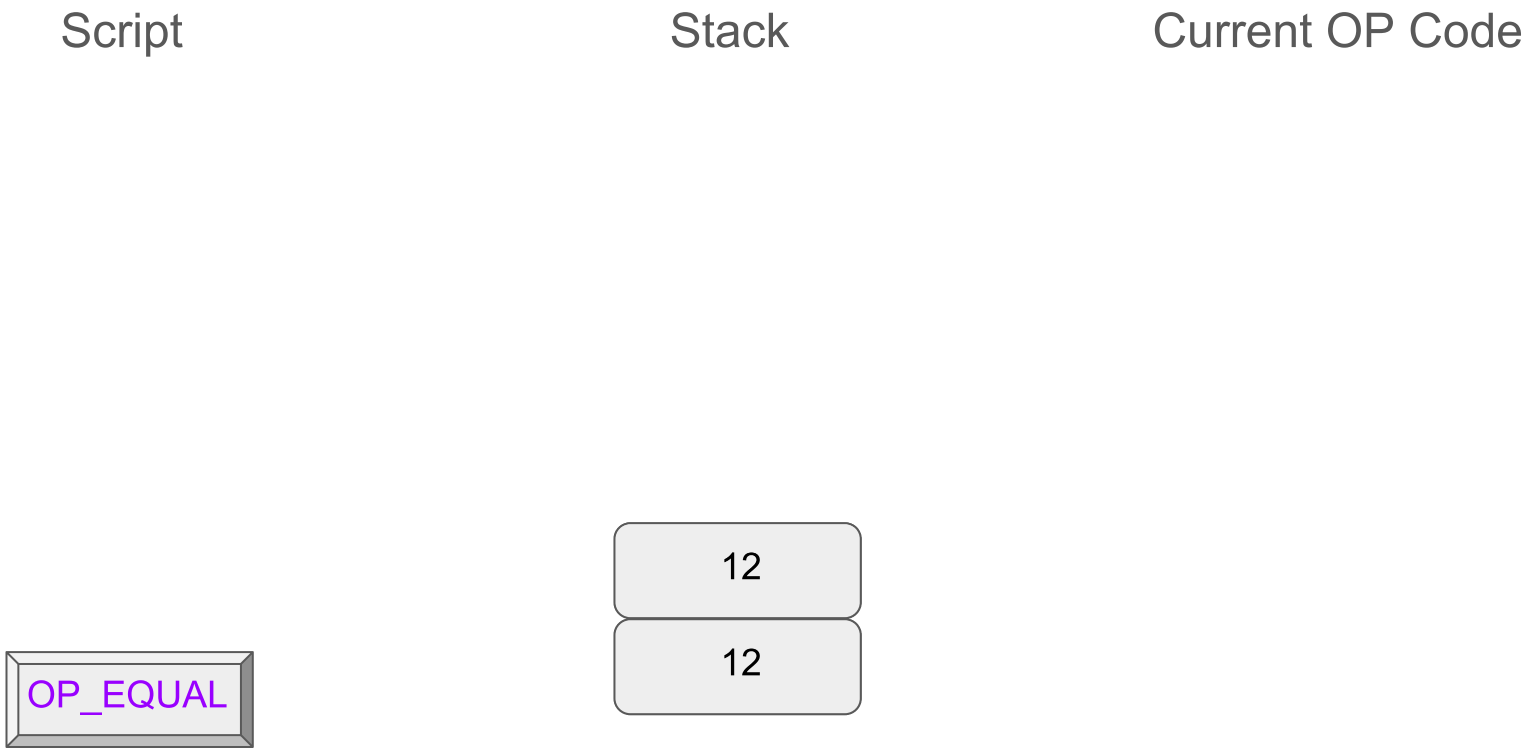
What does Script Look like?



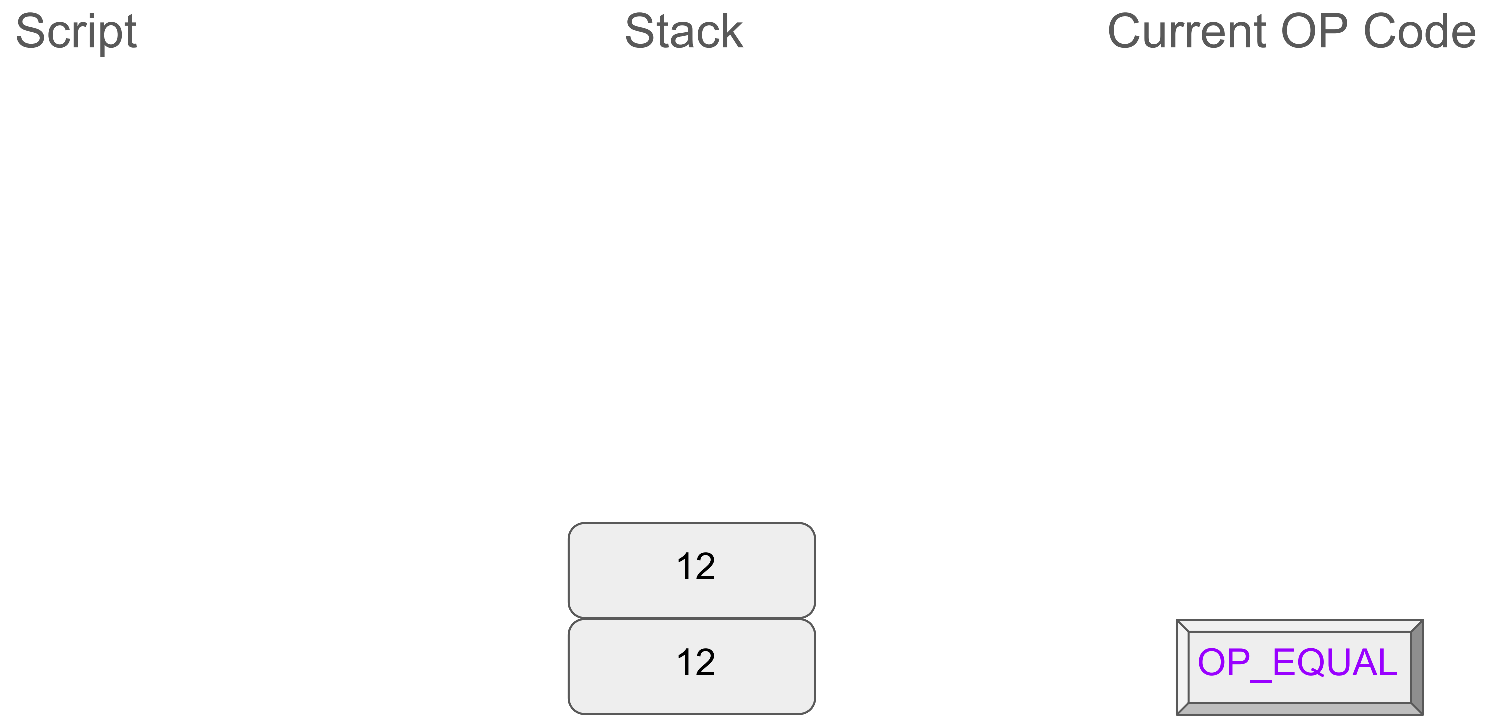
What does Script Look like?

Script	Stack	Current OP Code
OP_EQUAL	12	OP_12

What does Script Look like?



What does Script Look like?



What does Script Look like?

Script

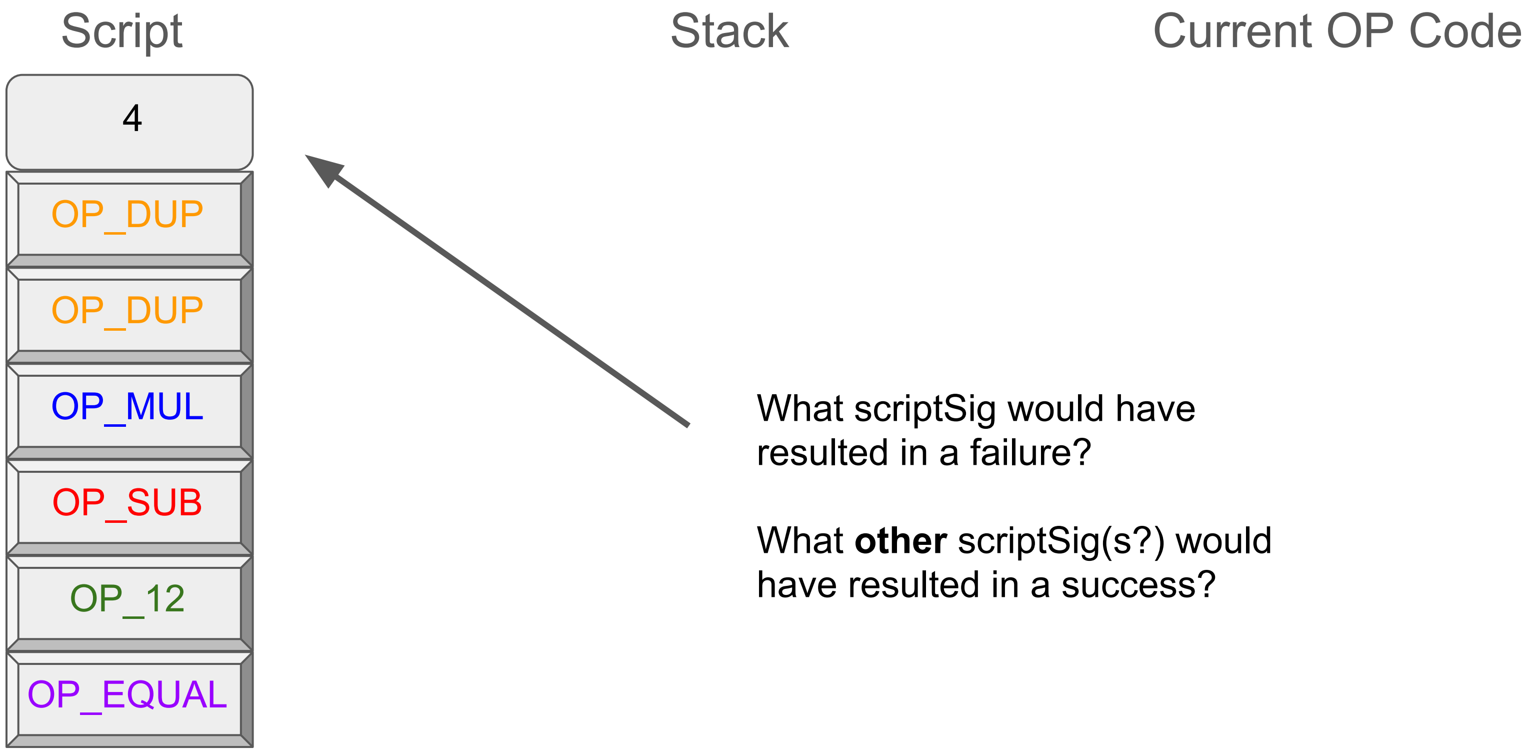
Stack

Current OP Code

SUCCESS!!

1

What does Script Look like?



What scriptSig would have resulted in a failure?

What **other** scriptSig(s?) would have resulted in a success?

Break...

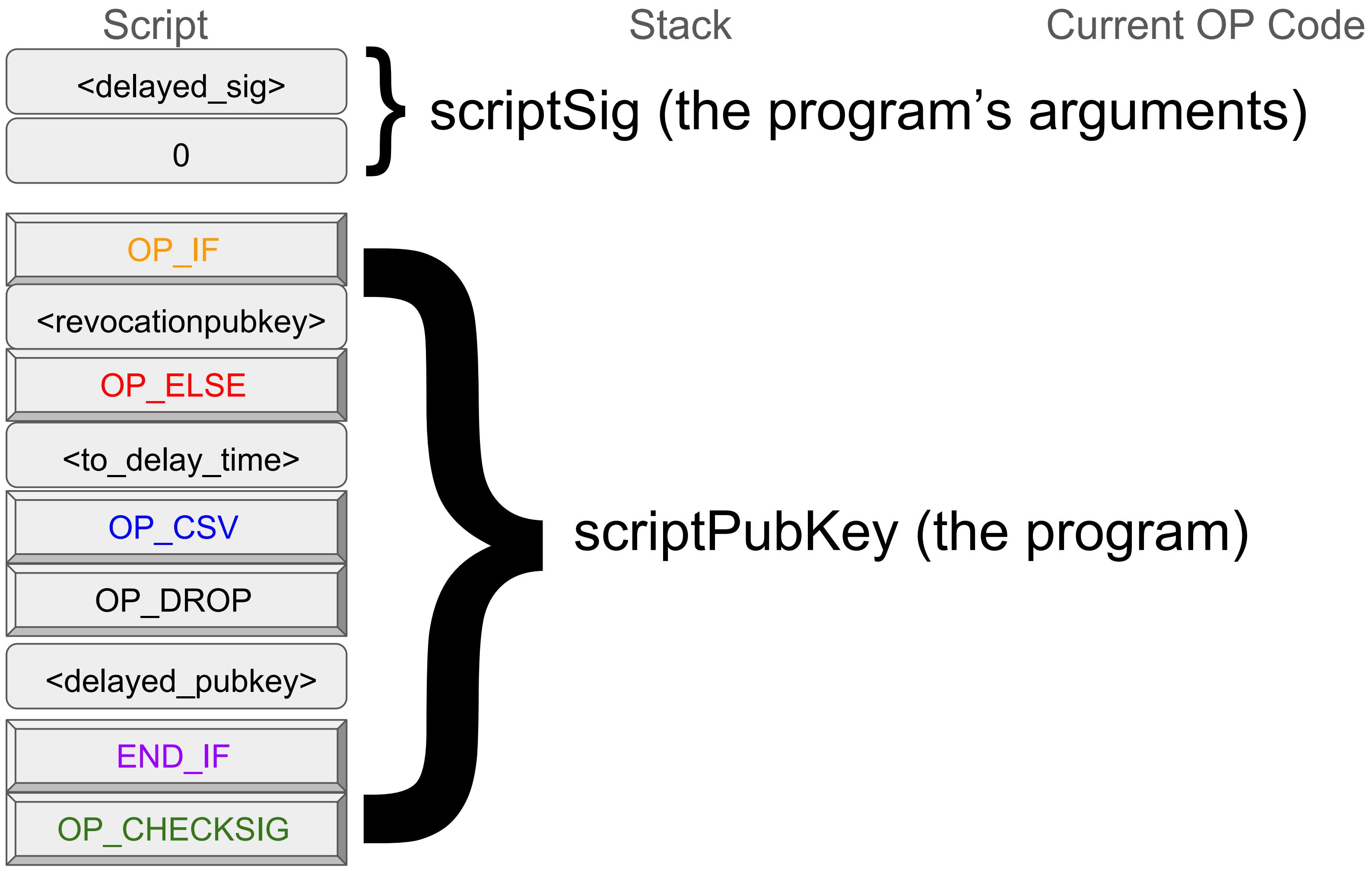
Commitment Transaction Output in Lightning

- Ensures a channel member can always get their funds (with some delay) if the other party disappears
  - OP_CSV
- Ensures it can be revoked, so we can update the channel to a new state (balance)
- When other party goes offline suddenly
  - `<local_delayedsig> 0`
- When othe party publishes revoked tx:
  - `<revocationsig> 1`

Commitment Transaction Output in Lightning

- Ensures a channel member can always get their funds (with some delay) if the other party disappears
  - OP_CSV
- Ensures it can be revoked, so we can update the channel to a new state (balance)
- When other party goes offline suddenly
  - `<local_delayedsig> 0`
- When othe party publishes revoked tx:
  - `<revocationsig> 1`

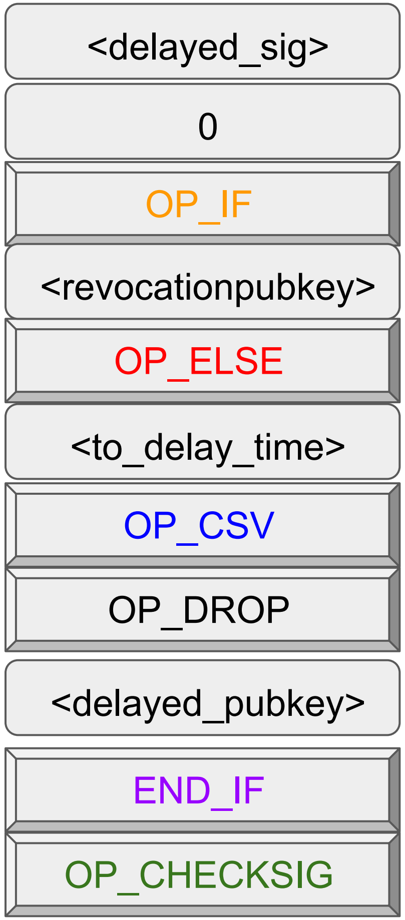
```
OP_IF
    <revocationkey> # Penalty transaction
OP_ELSE
    `to_self_delay`# number of blocks to delay
    OP_CSV
    OP_DROP
    <local_delayedkey>
OP_ENDIF
OP_CHECKSIG
```


Script
Stack
Current OP Code
<delayed_sig>
0
scriptSig (the program’s arguments)
OP_IF
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG
scriptPubKey (the program)

Script

Stack

Current OP Code



Script

```
0
OP_IF
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG
```

Stack

Current OP Code

```
<delayed_sig>
```

Script

```
0
OP_IF
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG
```

Stack

```
<delayed_sig>
```

Current OP Code

Script

OP_IF
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG

Stack

<delayed_sig>

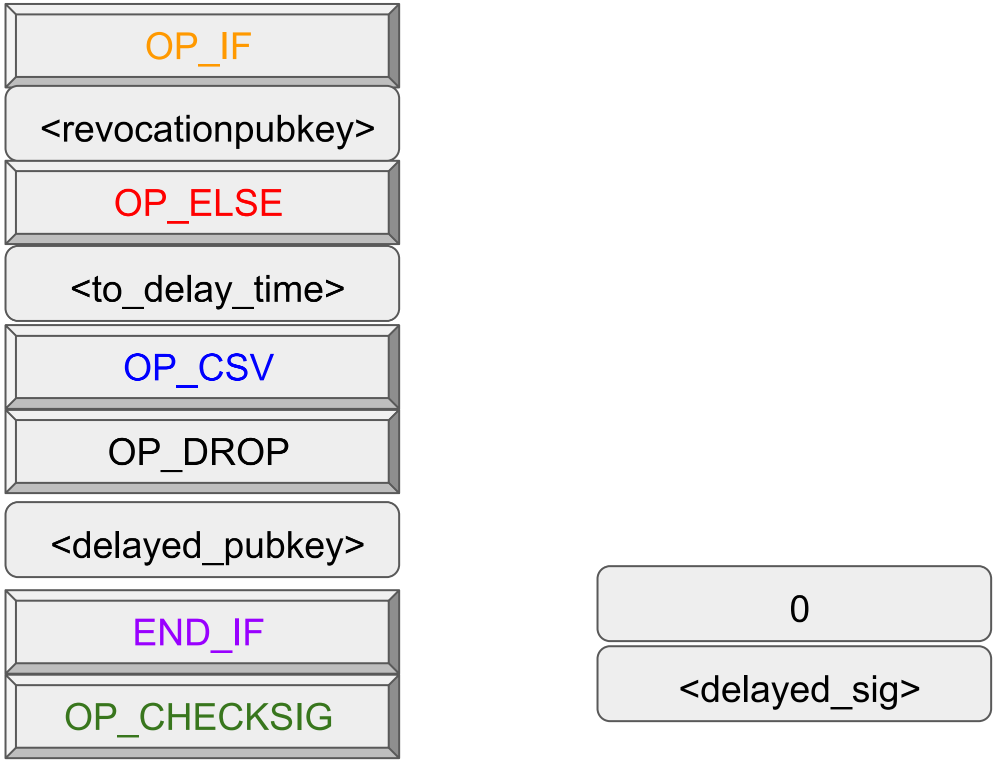
Current OP Code

0

Script

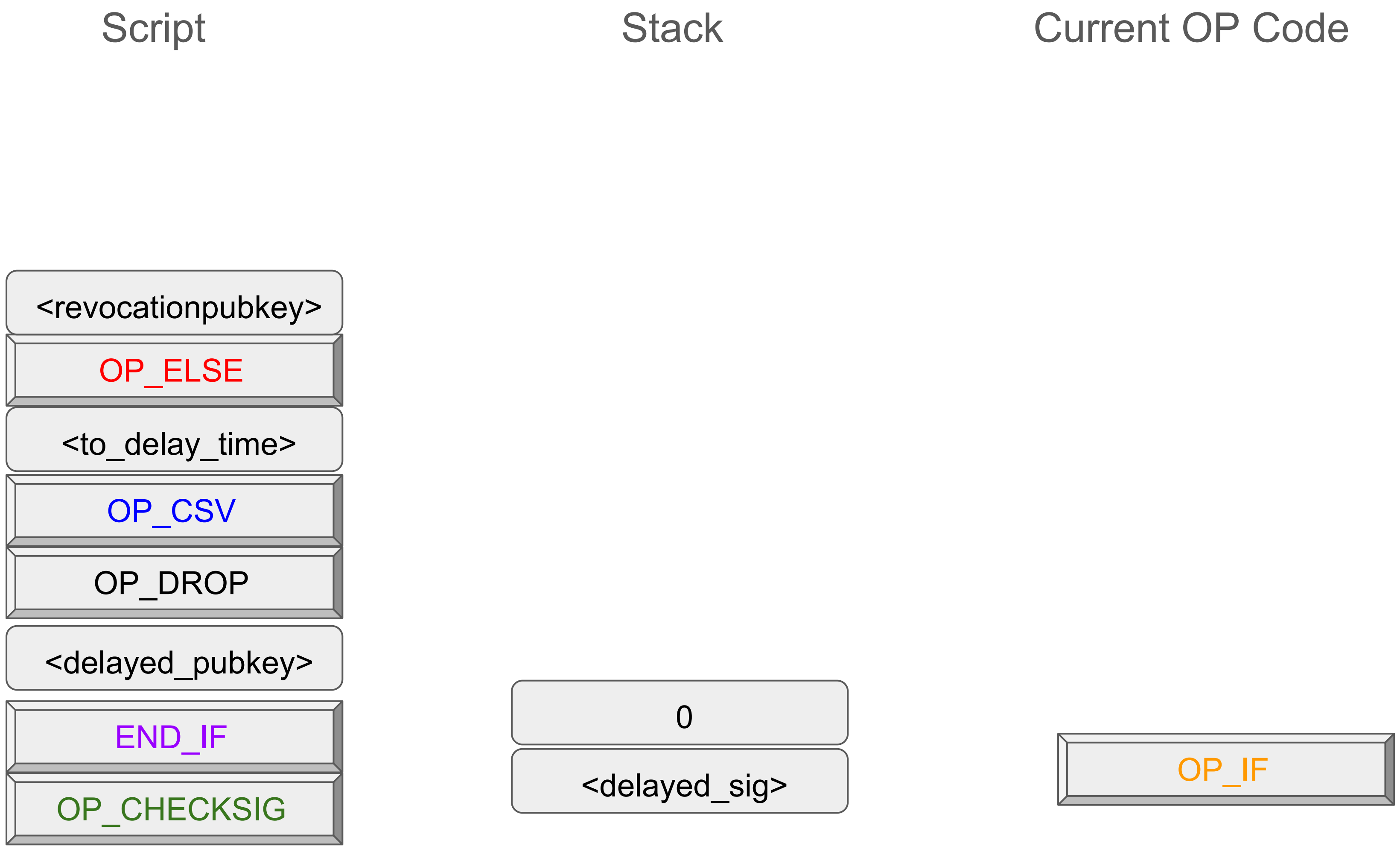
Stack

Current OP Code

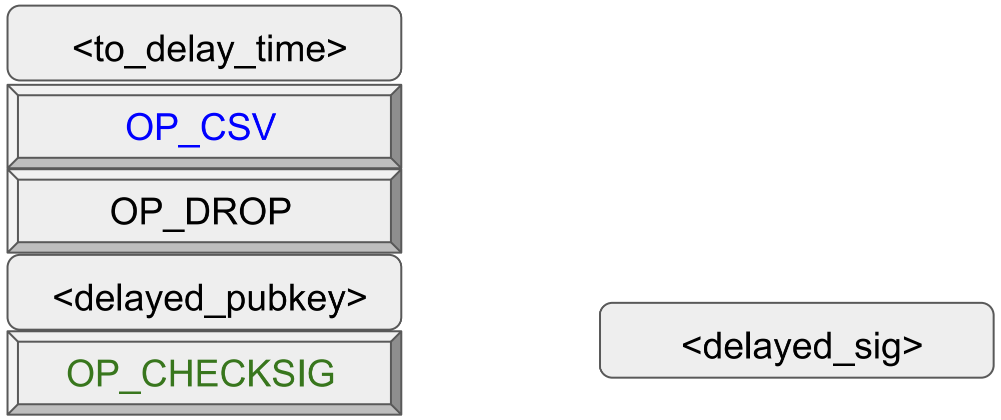


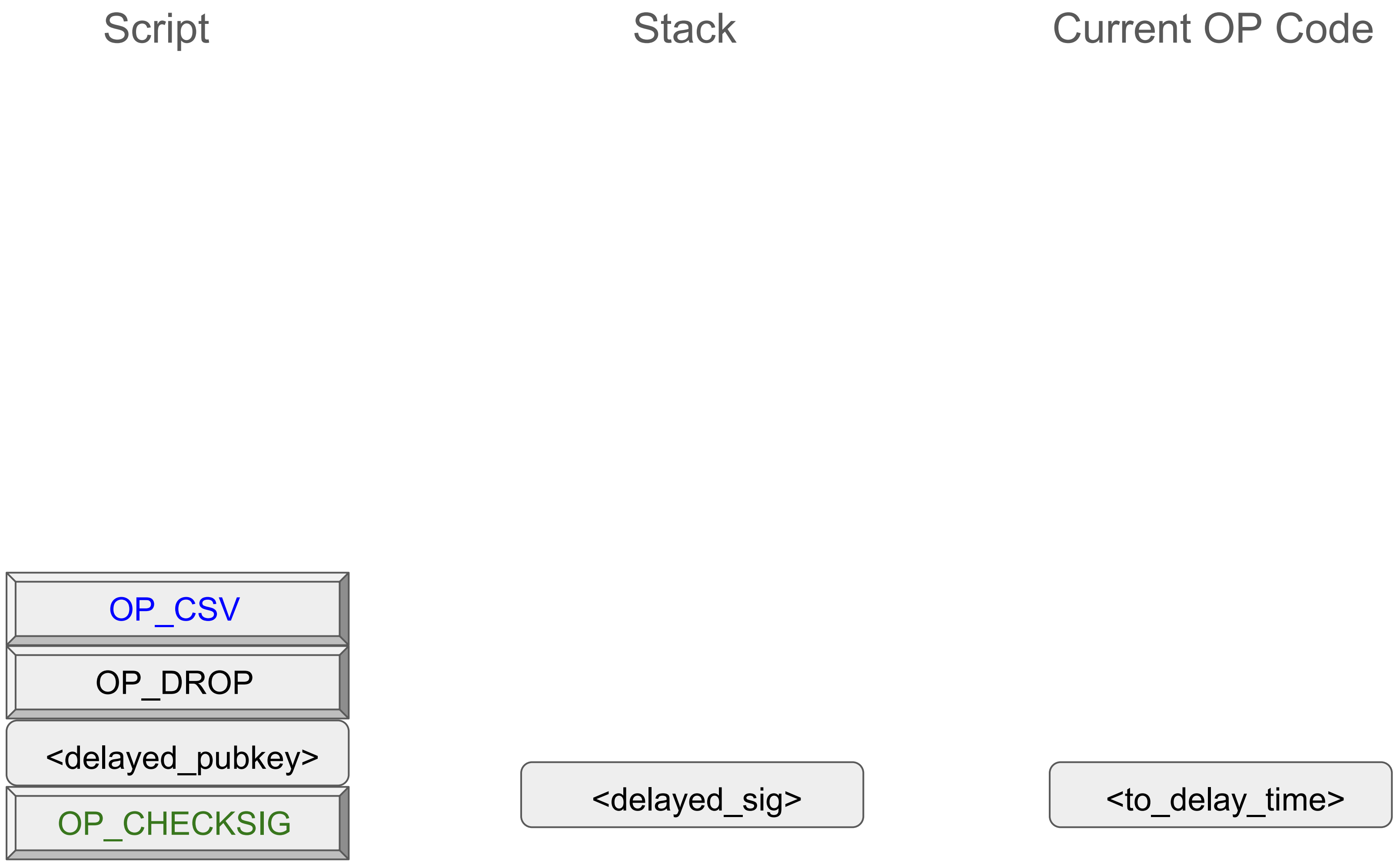
```
OP_IF
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG
```

```
0
<delayed_sig>
```


Script
Stack
Current OP Code
<revocationpubkey>
OP_ELSE
<to_delay_time>
OP_CSV
OP_DROP
<delayed_pubkey>
END_IF
OP_CHECKSIG
0
<delayed_sig>
OP_IF

Script Stack Current OP Code




Script
Stack
Current OP Code
OP_CSV
OP_DROP
<delayed_pubkey>
OP_CHECKSIG
<delayed_sig>
<to_delay_time>

Script	Stack	Current OP Code
OP_CSV		
OP_DROP		
<delayed_pubkey>	<to_delay_time>	
OP_CHECKSIG	<delayed_sig>	

Script

OP_DROP

<delayed_pubkey>

OP_CHECKSIG

Stack

<to_delay_time>

<delayed_sig>

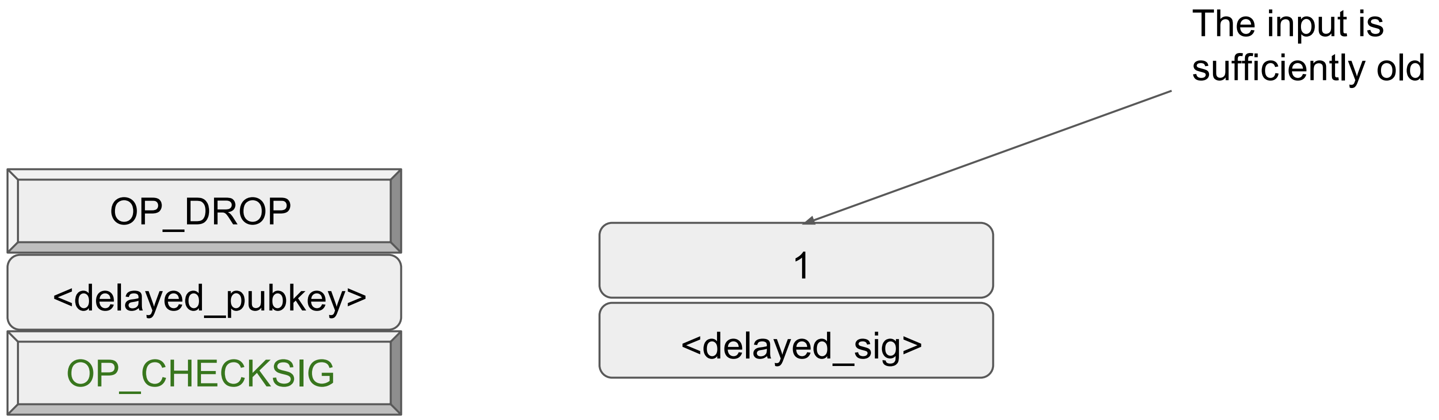
Current OP Code

Checks *nSequence*,
a field on the transaction

(oldestTx(inputs).nSequence + inputs[i].nSequence > currentBlock (if using block height based locking) or currentMedianTime (if using time based locking)

OP_CSV

Script Stack Current OP Code

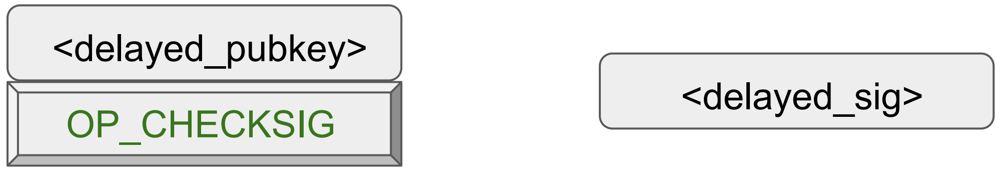


Script	Stack	Current OP Code
<delayed_pubkey>	1	OP_DROP
OP_CHECKSIG	<delayed_sig>	

Script

Stack

Current OP Code



Script	Stack	Current OP Code
OP_CHECKSIG	<delayed_sig>	<delayed_pubkey>

Script	Stack	Current OP Code
	<delayed_pubkey>	
OP_CHECKSIG	<delayed_sig>	

Script

Stack

<delayed_pubkey>

<delayed_sig>

Current OP Code

OP_CHECKSIG

Script

Stack

Current OP Code

SUCCESS!!

1

Thanks!