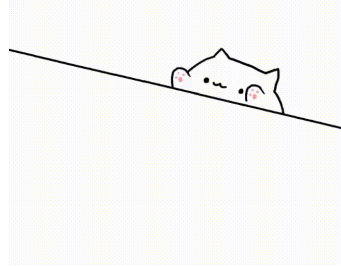
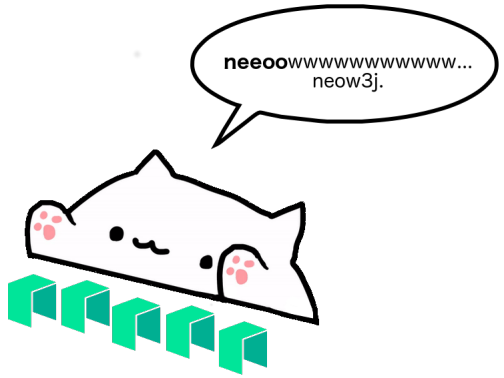


Neo Blockchain: Concepts & Coding

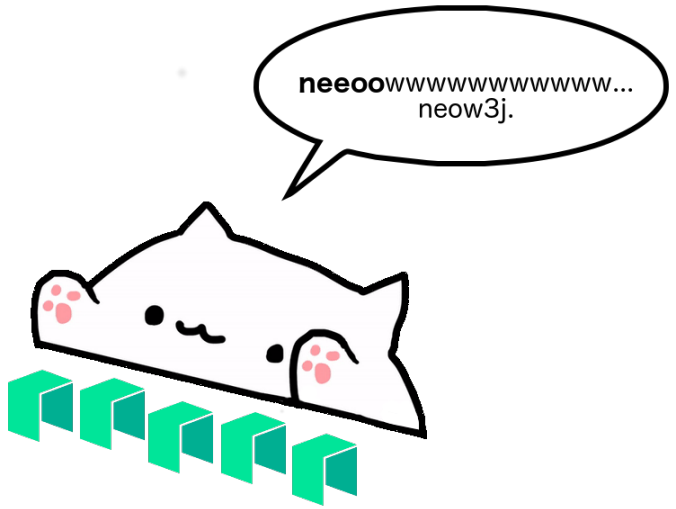
Dr. Guilherme Sperb Machado

AxLabs & neow3j



guil@axlabs.com

https://axlabs.com
https://neow3j.io

neo + 

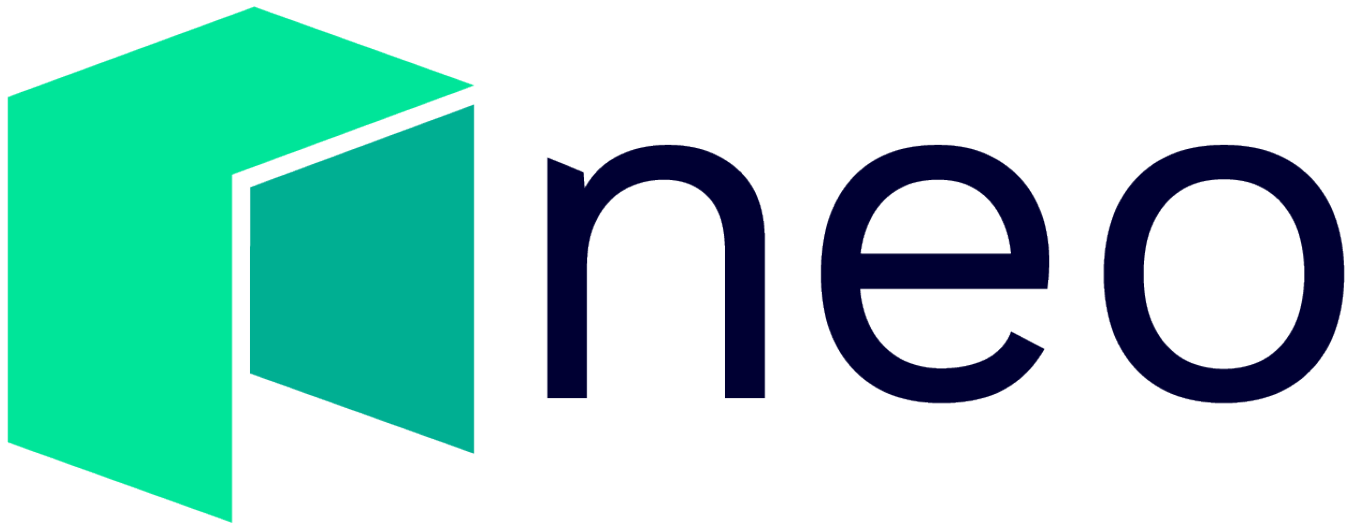
AXLABS

AxLabs

- Software Engineers, based in Zürich
- “Use bleeding-edge tech to build rock-solid [products, libraries, stuff, …]”
- Focus on open-source projects
  - neow3j: Java and Android library to interact with NEO blockchain
  - ICOnator: A secure, configurable, and scalable open source ICO engine
  - qry.sh: A Node Selection System based on DNS
- Neo community member

Agenda

- Part 1: Concepts
  - Neo Background
  - Neo Digital Assets
  - Addresses
  - UTXO Model
  - Transactions
  - Signature
  - Neo Nodes and Network
- Part 2: Demo & Coding
  - Smart Contracts Concepts
  - Set-up the environment
  - Deploy Smart Contract
  - Interact with the contract
  - Questions & Answers


neo

Neo

- Neo Project was founded in 2014
- Open-sourced on GitHub in June 2015
- Neo is a completely open-sourced global project
- Developers across the globe join and contribute to Neo

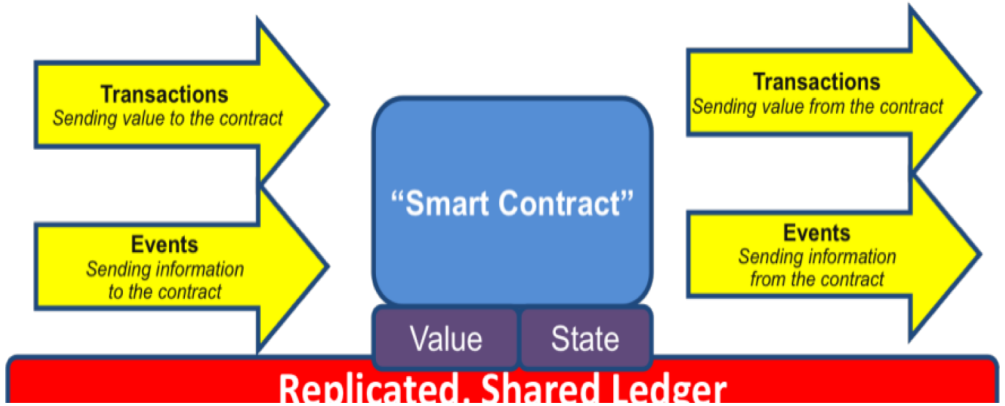
GitHub

https://github.com/neo-project/neo

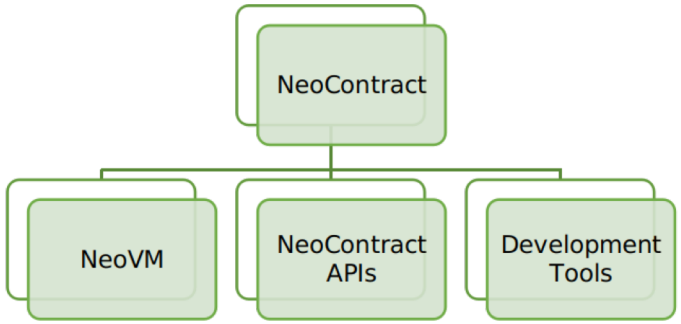
- Neo is a non-profit community blockchain
- Uses blockchain to digitalize assets and identity
- Uses smart contract to manage digital assets, automatically.
- Distributed network to realize the *"smart economy"*

Smart Contract Concept and Neo VM

- Smart Contracts
  - Idea posted by Nick Szabo
  - Set of commitments defined in digital form, including agreements on which contract participants can execute these commitments

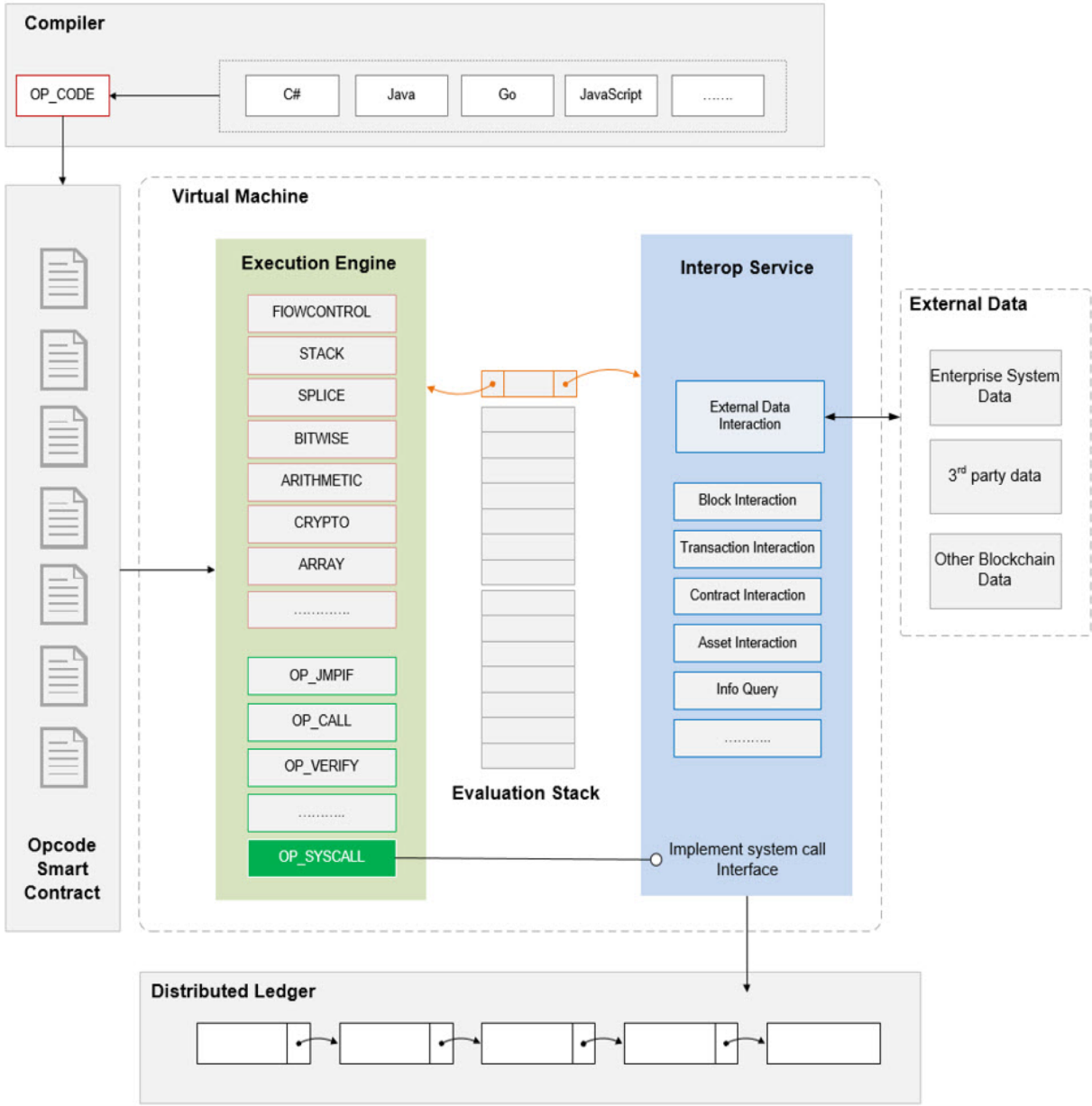


- Neo Smart Contracts
  - Compiled to binary format (.avm)
  - Run on Neo VM
  - APIs (i.e., ABI – Application Binary Interface)
  - Built with a set of development tools



Neo VM

- Stack-based Virtual Machine
- Executes a set of instructions
- Developers can use several general-purpose programming languages
  - Dev. tools convert high-level programming languages to Neo VM instructions

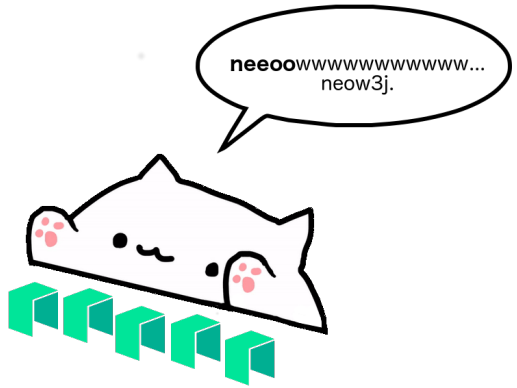


https://github.com/neo-project/neo-vm

Neo Global Community

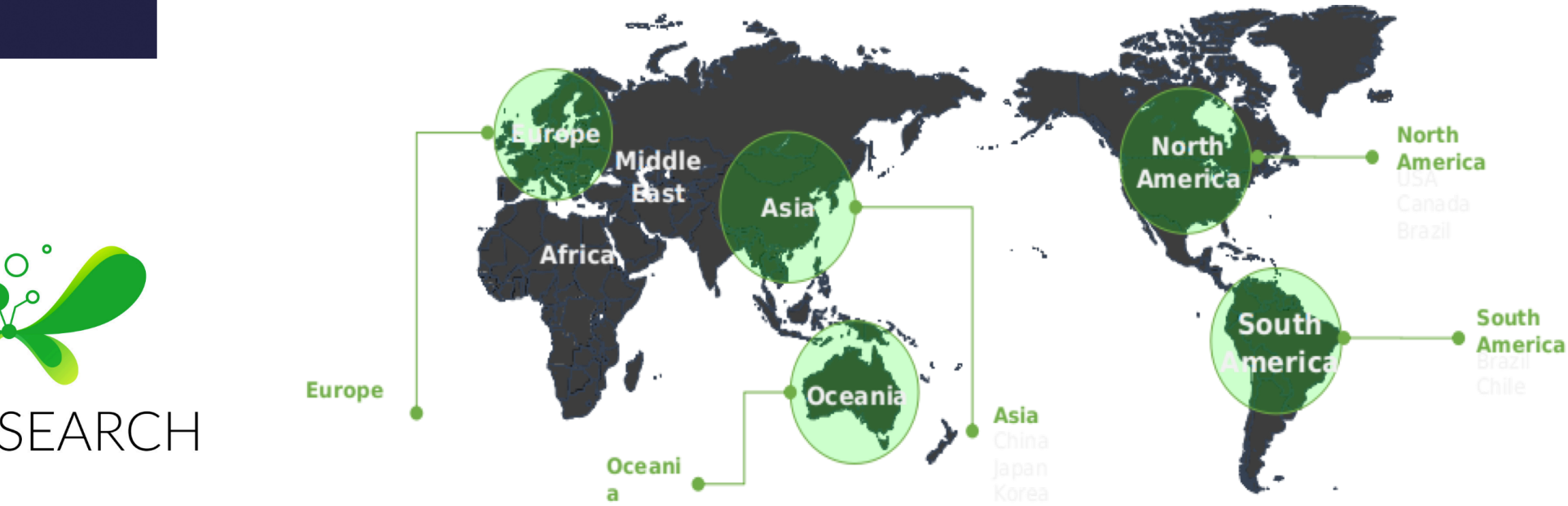
CITY OF ZION

NEL
NEWECONOLAB



NEO ONE

NEORESEARCH



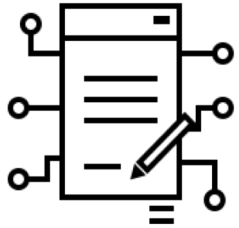
neo

Neo 3.0: The Future

TPS

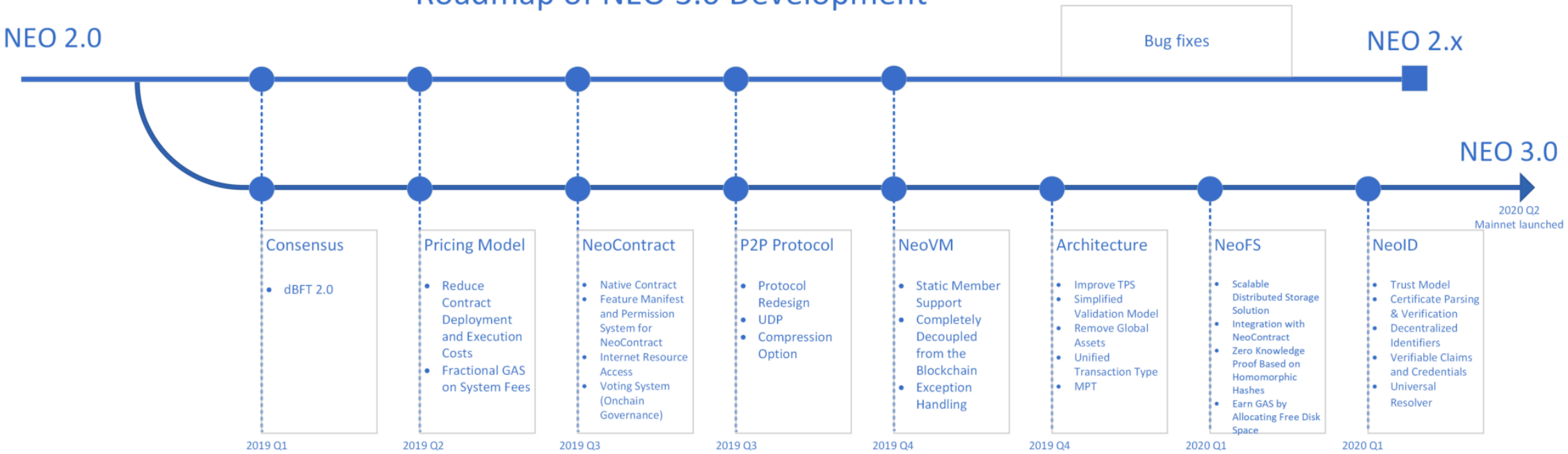
Economic Model

NEO VM

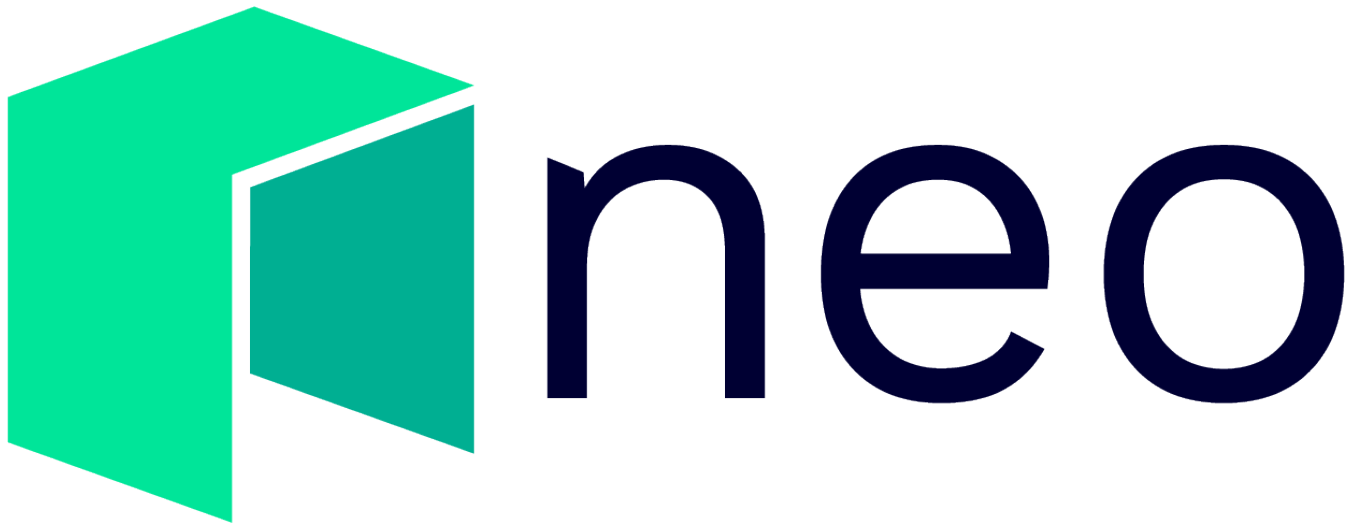


Smart Contracts

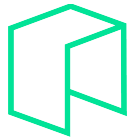
Roadmap of NEO 3.0 Development



neo


neo

Neo Global Assets

- NEO 
  - Cannot be mined
  - Not divisible
  - Max supply: 100,000,000
  - It represents a share in the NEO market
  - Voting rights to, e.g., elect validator nodes
  - Holders receive “dividends” as GAS
- GAS 
  - Divisible
  - Max supply: 100,000,000
  - Smart Contract processing fee
  - Generated at a rate of 8 GAS per block
    - Equally distributed to all NEO holders
  - Reduced by 1 token for every 2 million blocks generated

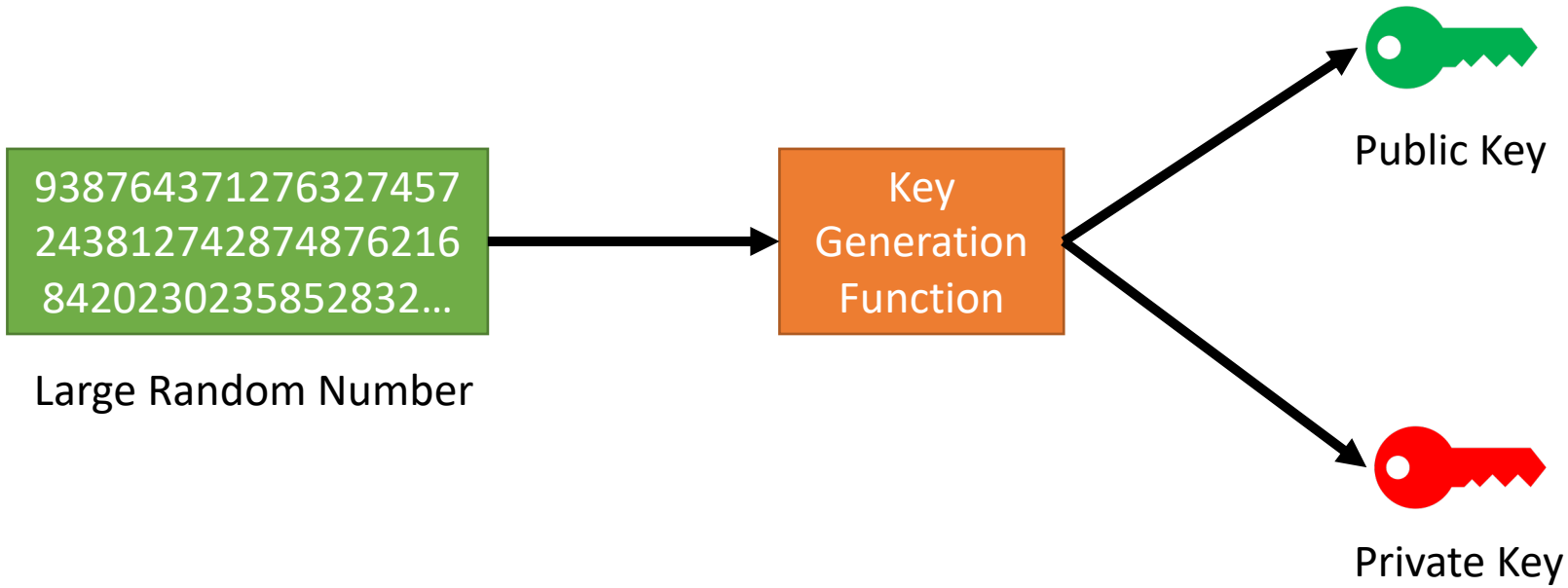
Wallets

- “Store” the private key
- Calculates the balances of NEO and GAS (due to UTXO model)
- Fetches balance for NEP-5 (account model)
  - NEP-5 <-> ERC-20

- Standards:
  - NEP-6: Wallet Standard
  - NEP-2: Passphrase-protected private key

Key Pair

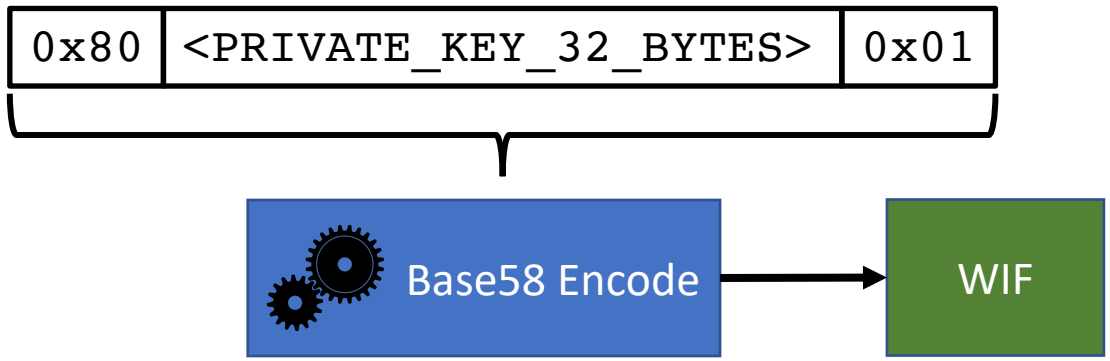
- Private key + Public key



Private Key

- Random value generated between 1 and N
  - N is a constant, (slightly) less than 2^256
  - Represented by a 256 bit (32 bytes) number

- WIF Format (Wallet Import Format):

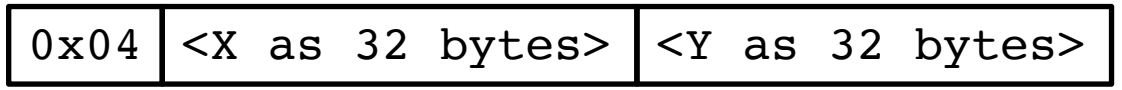


Format	Value
byte[]	[0xc7, 0x13, 0x4d, 0x6f, 0xd8, 0xe7, 0x3d, 0x81, 0x9e, 0x82, 0x75, 0x5c, 0x64, 0xc9, 0x37, 0x88, 0xd8, 0xdb, 0x09, 0x61, 0x92, 0x9e, 0x02, 0x5a, 0x53, 0x36, 0x3c, 0x4c, 0xc0, 0x2a, 0x69, 0x62]
Hex string	c7134d6fd8e73d819e82755c64c93788d8db0961929e025a53363c4cc02a6962
WIF	L3tgppXLgdaeqSGSFw1Go3skBiy8vQAM7YMXvTHsKQtE16PBncSU

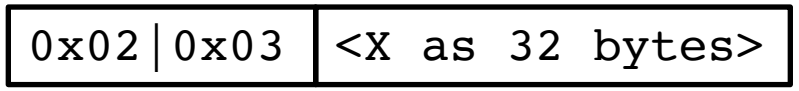
Public Key

- Point (x, y) obtained through the ECC algorithm with the private key
- Neo: secp256r1 curve (ECC algorithm)

- Uncompressed Public Key

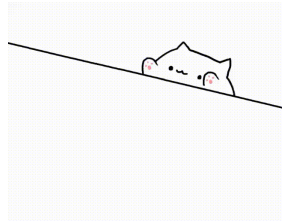


- Compressed Public Key



Format	Value
Private Key	c7134d6fd8e73d819e82755c64c93788d8db0961929e025a53363c4cc02a6962
Public Key (Compressed)	035a928f201639204e06b4368b1a93365462a8ebbff0b8818151b74faab3a2b61a
Public Key (Uncompressed)	045a928f201639204e06b4368b1a93365462a8ebbff0b8818151b74faab3a2b61a35dfabcb79ac492a2a88588d2f2e73f045cd8af58059282e09d693dc340e113f

Neo Addresses [1]

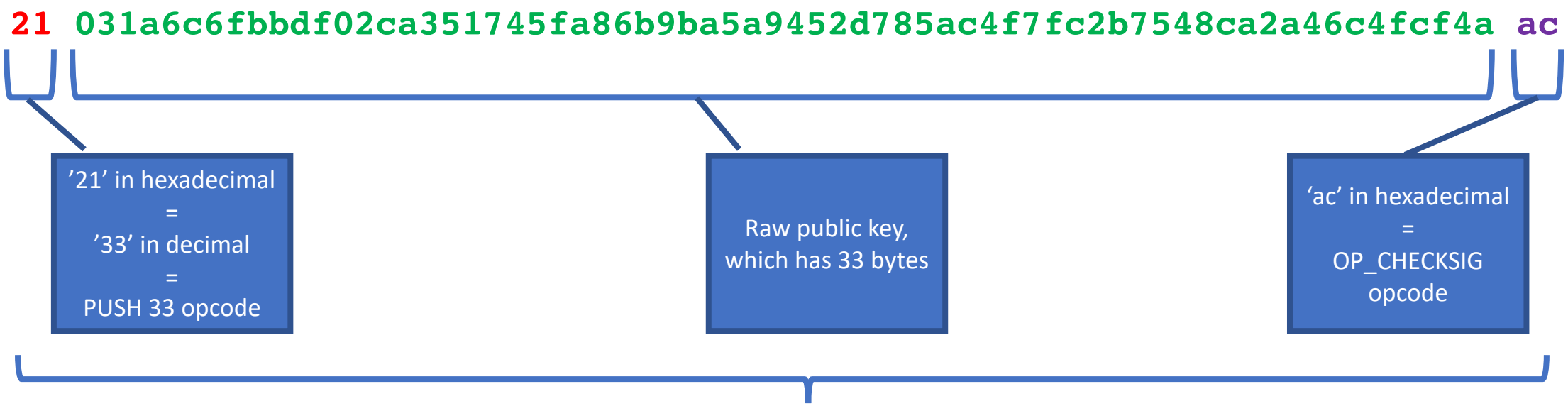


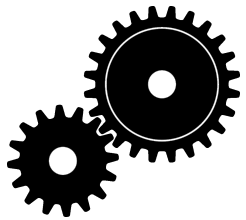
- Based on a “key pair”
  - More specifically, based on the **Public Key**
- Possible to have multiple addresses in one Neo wallet

- Address is a script!
  - Proving that such script can be “unlocked”, then it gives the ability to, e.g., move funds and/or perform other actions related to the address

`AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y`

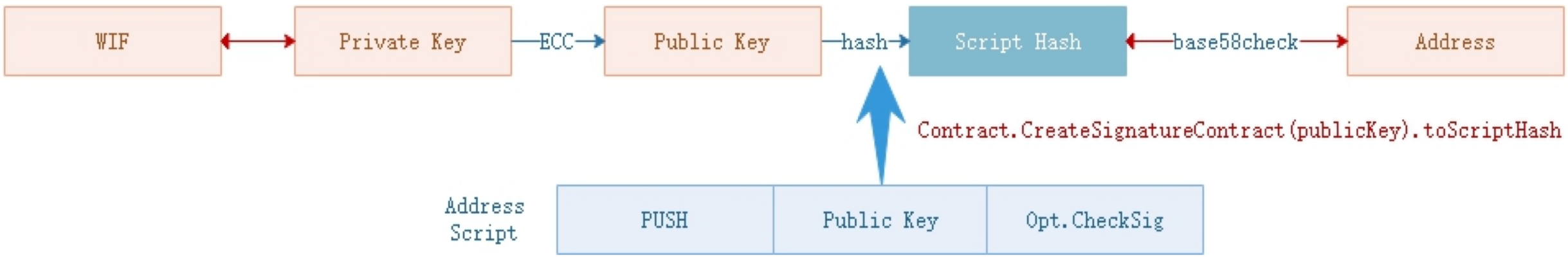
Neo Addresses [2]





- Generate the ScriptHash by hashing the "public key script"
  - sha256(ripemd160("public key script"))
- Concatenate 0x17 (NEO version) as prefix
- Calculate checksum, take only first 4 bytes, and concatenate with ScriptHash
- Perform Base58 of everything
- Result: `AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y`

Neo Addresses [3]

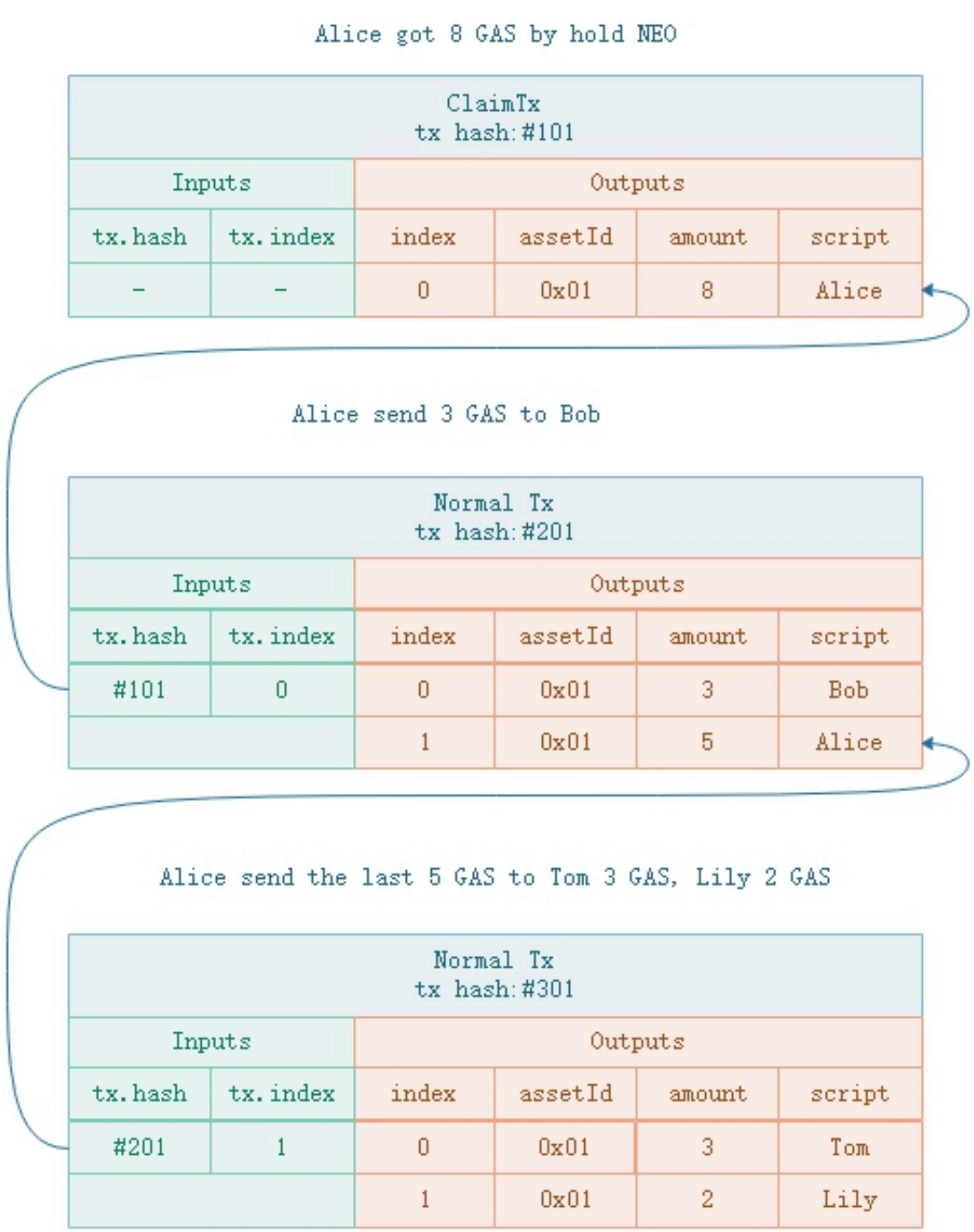


UTXO Model [1]

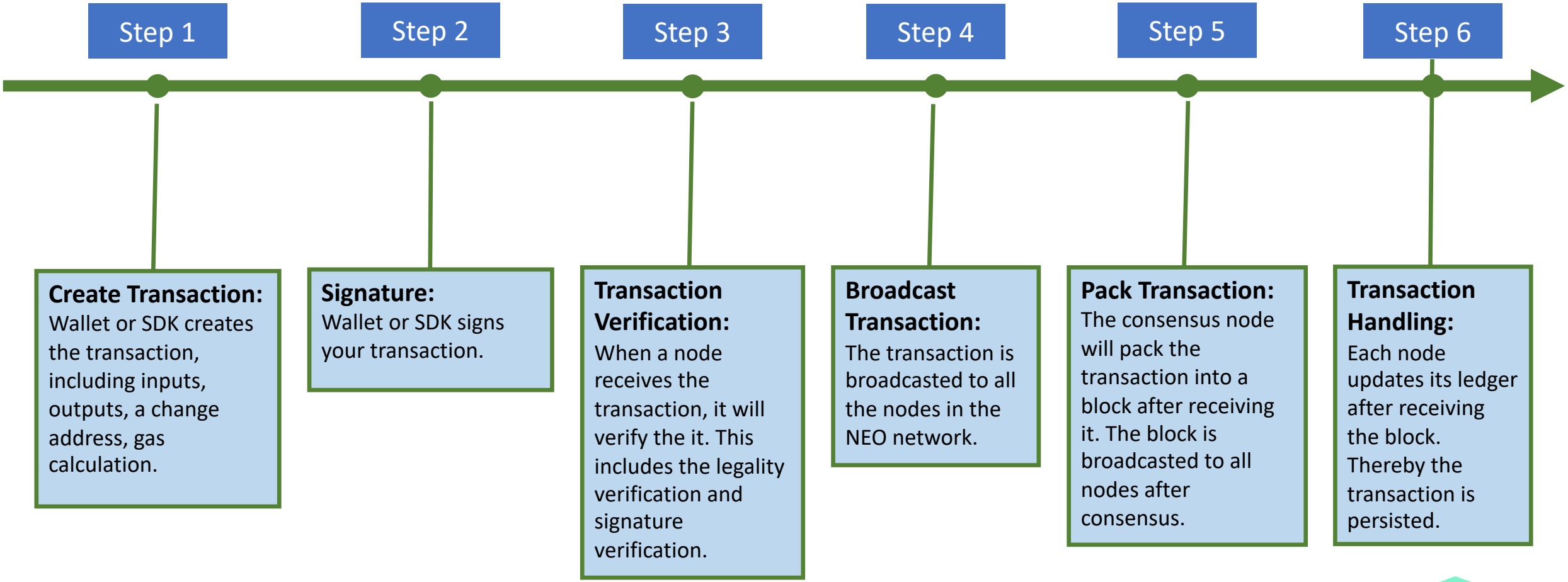
- Different concept than maintaining assets on an account
  - Account has no “balance”
- There’s a list of non-spent transactions
  - Analogy: concept of writing/receiving a check
- Inputs and outputs
  - Inputs indicate which previous transactions (outputs) should be spent
  - Outputs are the results of the transaction
- Signatures are the key

UTXO Model [2]

- Input (Origin of coins)
  - Transaction hash and
  - index of an output belonging to Alice
- Output (Destination of coins)
  - AssetId
  - Amount
  - Receipt address (ScriptHash)



Transactions [1]



Transactions [2]

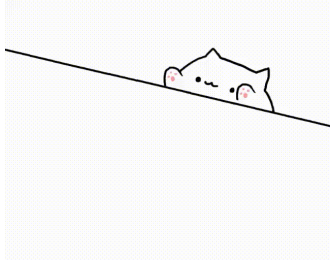
Property	Description
Type	The type of transaction
Version	The version number
Attributes	Set of transaction attributes
Inputs	Array of inputs • Information about past transactions serving as inputs to this one
Outputs	Array of outputs • Information about the receiver
Witness	Witness Script • The script that shows that the originator of the transaction is eligible to spend the inputs.

Witness is composed of:

- Invocation Script
- Verification Script

- PrevHash
- PrevIndex

- AssetId
- Value
- ScriptHash



AXLABS

Witness

- Composed of two parts:
  - Verification Script
    - Public key script
    - Example: `21031a6c6fbbdf02ca351745fa86b9ba5a9452d785ac4f7fc2b7548ca2a46c4fcf4aac`
    - This is basically the “address” of who’s doing the transaction
  - Invocation Script:
    - Signature of the transaction data
    - It should match the public key script

neo

Transaction Types

Name	System Fee	Description
MinerTransaction	0	Assign byte fees
IssueTransaction	500|0	Issuance of asset
ClaimTransaction	0	Assign GAS
EnrollmentTransaction	1000	Enrollment for validator
RegisterTransaction	10000	Assets register
ContractTransaction	0	Contract transaction
InvocationTransaction	0	Special transactions for calling Smart Contracts

Each transaction type might require special fields to be included in addition to the base transaction fields.

Invocation Transaction

Base Transaction Fields

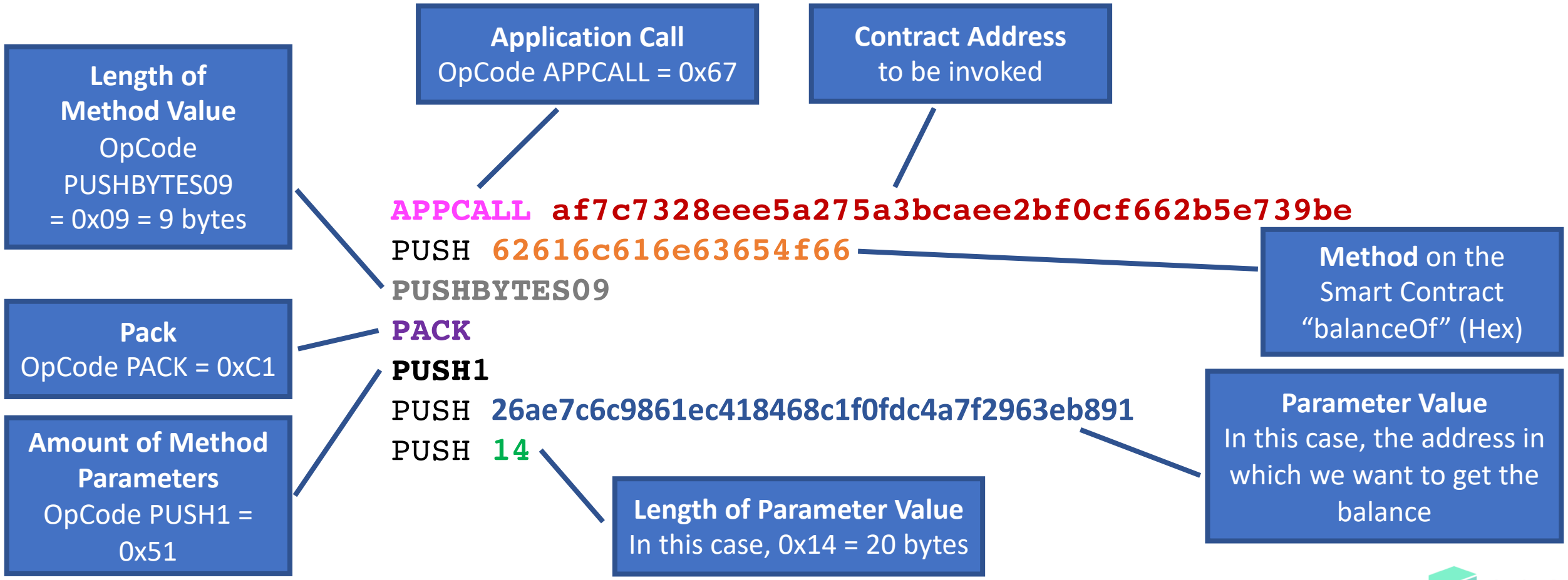
Specific Fields to Invocation Transaction

Property	Value
Type	InvocationTransaction (0xd1)
Version	0
Attributes	…
Inputs	…
Outputs	…
Witness	…
Script	Smart Contract -> AVM script
Gas	n

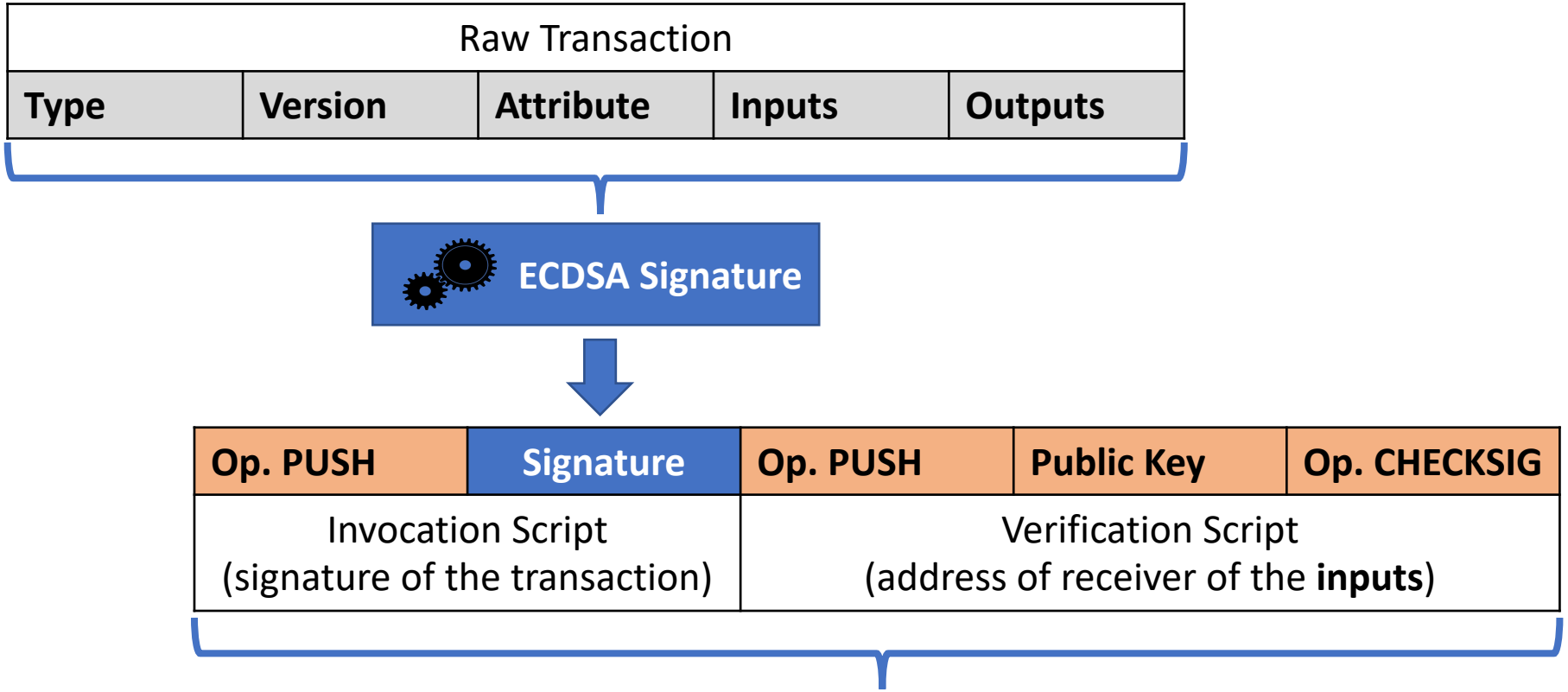
10 GAS is free

Invocation Transaction – Script

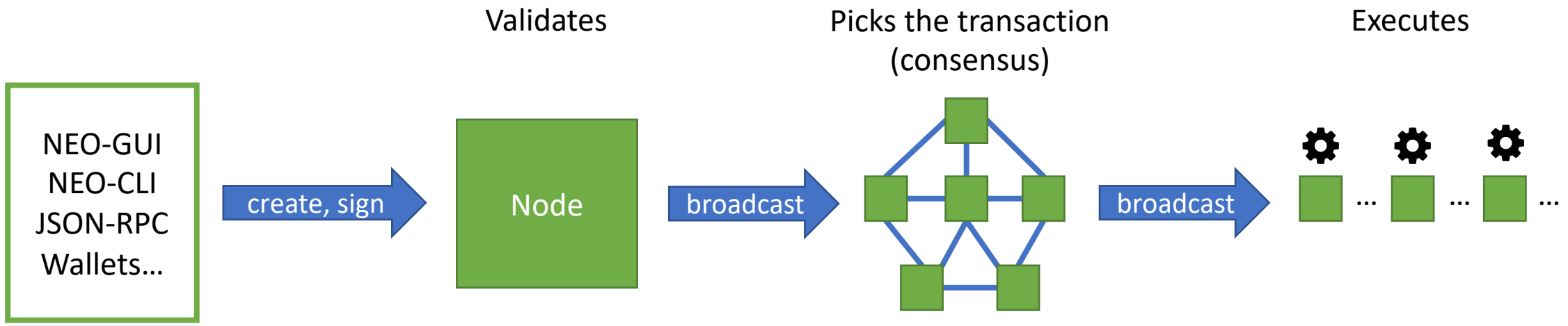
1426ae7c6c9861ec418468c1f0fdc4a7f2963eb89151c10962616c616e63654f6667be39e7b562f60cbfe2aebca375a2e5ee28737caf

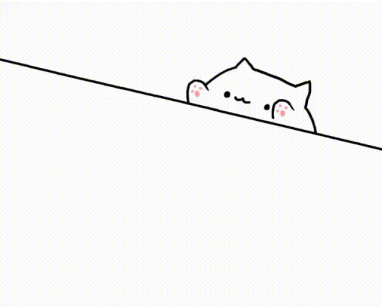


Signature


Raw Transaction
Type
Version
Attribute
Inputs
Outputs
ECDSA Signature
Op. PUSH
Signature
Op. PUSH
Public Key
Op. CHECKSIG
Invocation Script
(signature of the transaction)
Verification Script
(address of receiver of the inputs)
Type
Version
Attribute
Inputs
Outputs
Witnesses
Raw Transaction

Transaction Flow to the Network

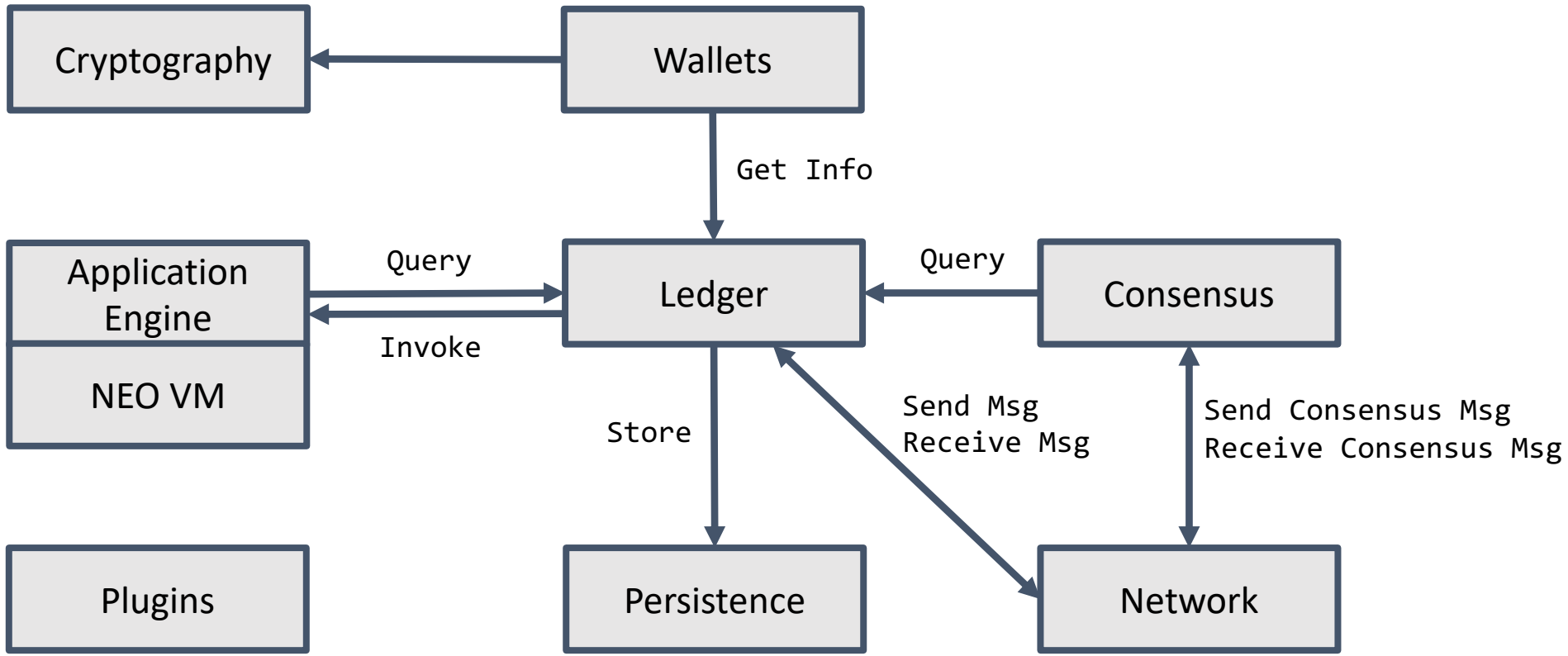




Legality Verification and Script Verification

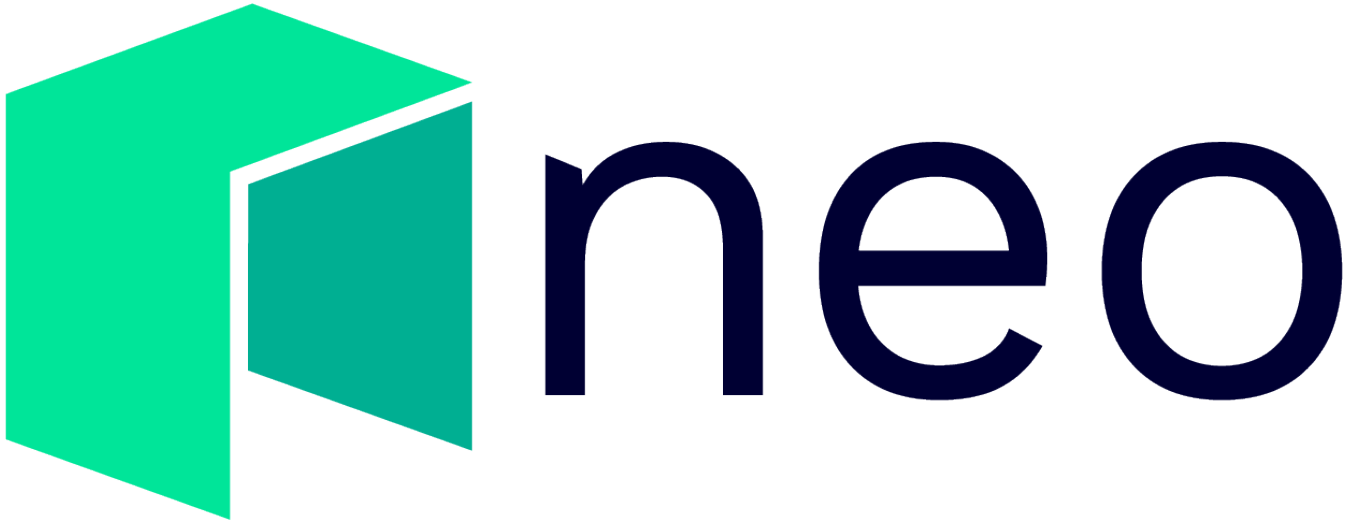
- Legality Verification
  - No duplicated inputs
  - No spent input
  - The input must be equal to output
  - The asset must be correct
  - …
- Script Verification
  - Unlock the UTXO
  - Build/check the “witness part”
  - …

NEO Full Node – Structure



NEO Node – Plugins

Name	Description
ApplicationLogs	Synchronizes the smart contract log (ApplicationLogs), and provide the logs/events in a JSON-RPC method.
CoreMetrics	Metrics from the Blockchain. This can involve reports through RPC calls or other forms of persistence.
ImportBlocks	Synchronizes the client using offline packages.
SimplePolicy	Enables policies for Consensus or Seed Nodes. In particular, policies for accepting transactions in the mempool and how to manage them.
RpcSecurity	Improves JSON-RPC security. Enables authentication for calls.
RpcSystemAssetTracker	Plugin that enables Assets tracker. In particular, it enables JSON-RPC calls for ”getunpents” and ”claimables”.
StatesDumper	Exports NEO-CLI status data (useful for debugging).
RpcNep5Tracker	Plugin that enables NEP5 tracking using LevelDB.



Part 2: Demo & Coding

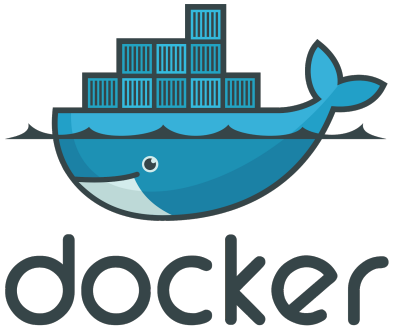
Useful Material for Demo & Coding

- https://github.com/AxLabs/neo-workshop

Set-Up the Basic Environment

- Install git

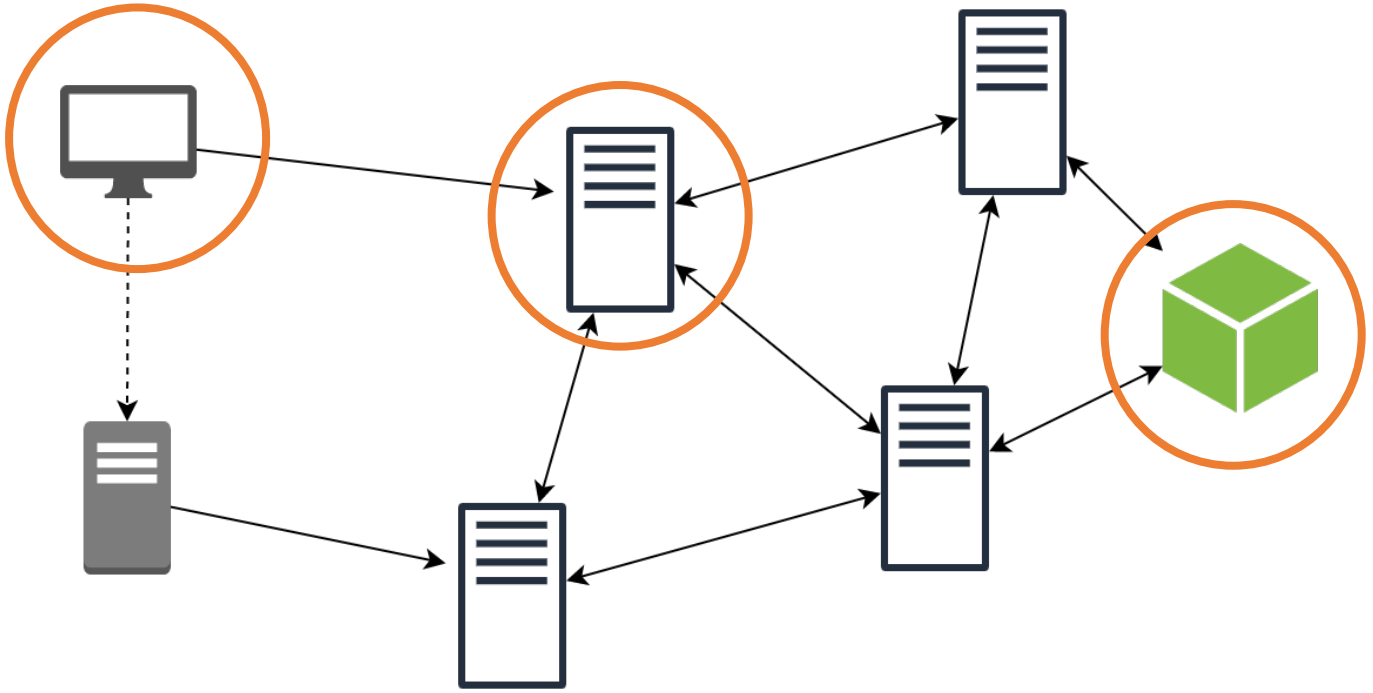
- Install docker
  - Windows: this and this can help you install it and solve pitfalls.
  - MacOS: it's pretty easy, this link is all you need.
  - Linux: also pretty easy, this link is all you need.

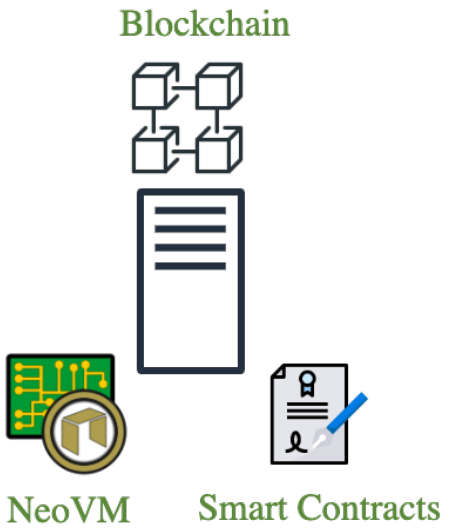


- We will use **Python**!
  - https://neo-python.readthedocs.io/en/latest/

- Optional:
  - Python 3.6+
  - PyCharm Community Edition (IDE)

Setting-up a private Network [1]

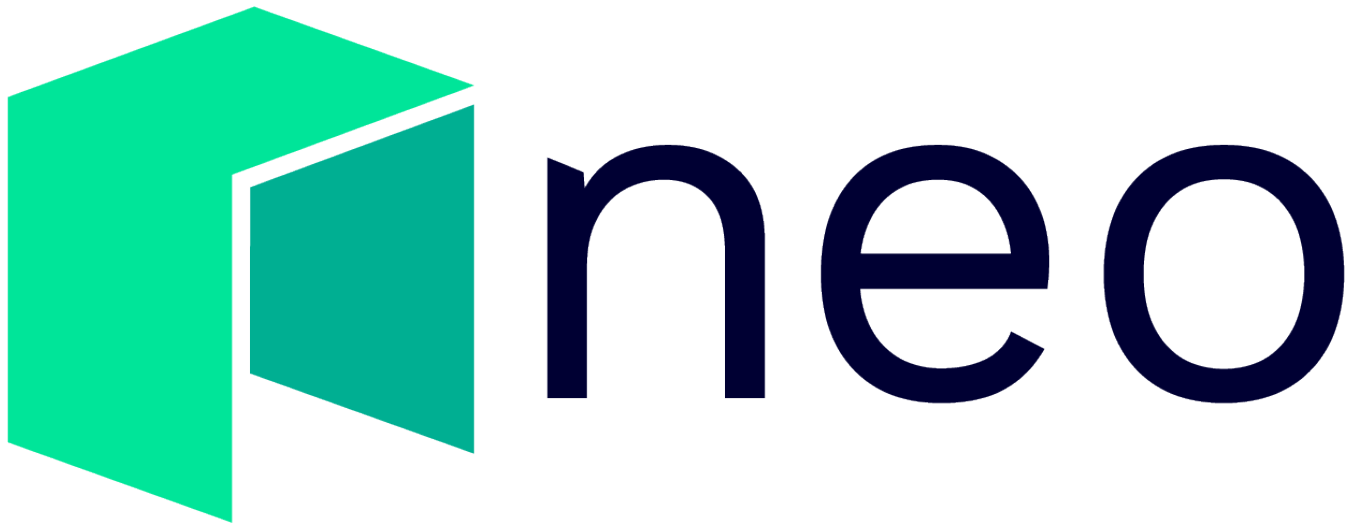




Setting-up a private Network [2]

```
$ git clone https://github.com/AxLabs/neo-local.git
$ cd neo-local
$ docker-compose up
```

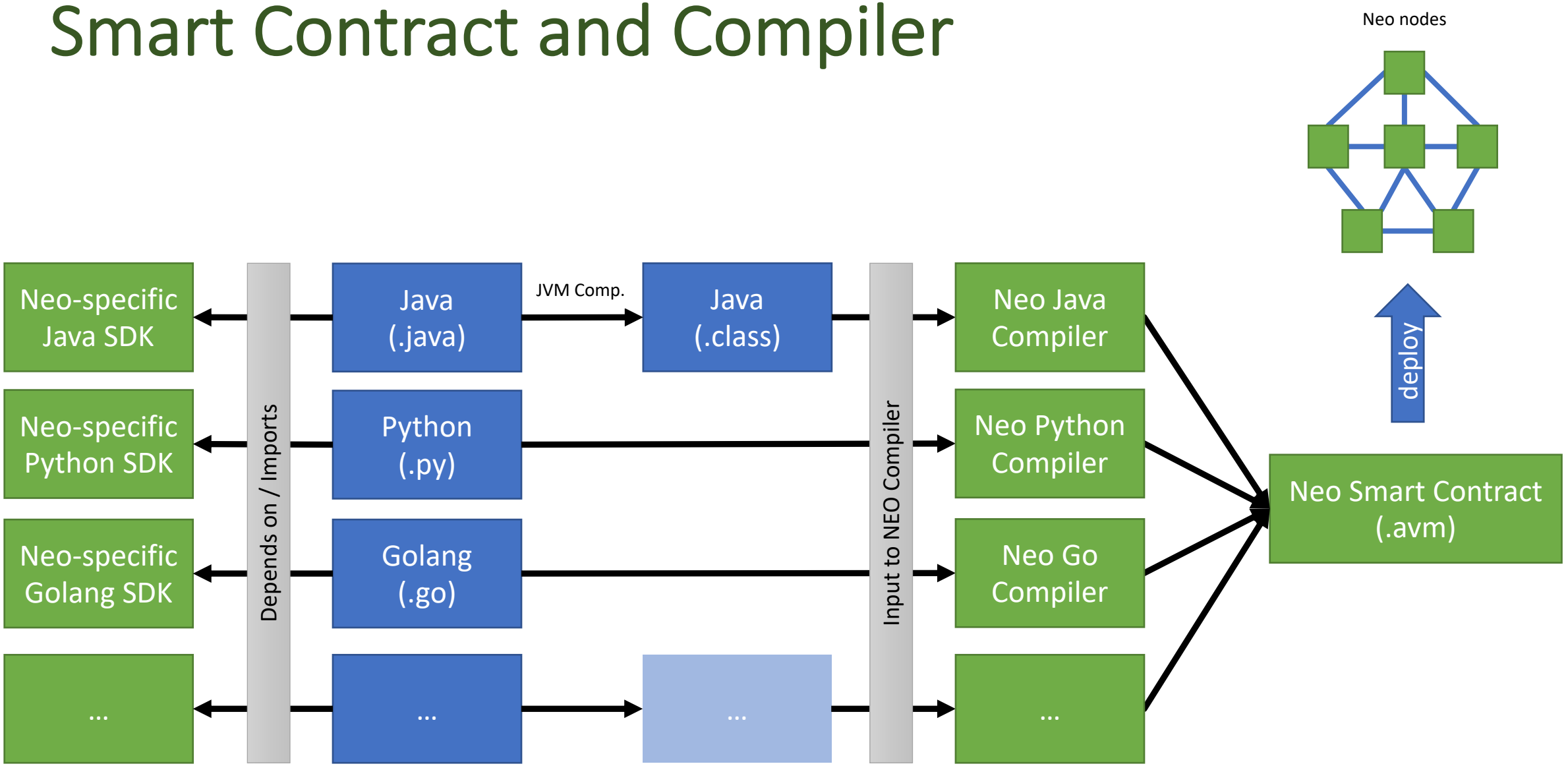
- Check if you can access http://localhost:4000 in the browser (neo-scan).
- The NEO node JSON-RPC interface is reachable on:
  - http://localhost:30333 (C# node, without open wallet)


neo

Smart Contract: Supported Languages

- C#
- Java, Kotlin
- Python
- Golang
- JavaScript (Typescript)

Smart Contract and Compiler



Language Features & Data Types Support

- Depends on the development language…
- Usually, Neo compilers support a sub-set of the language (built-ins), as well as data types

- For example, for **Python**:
  - Supported:
    - https://neo-boa.readthedocs.io/en/latest/overview.html#what-parts-of-python-are-supported
  - Not Supported
    - https://neo-boa.readthedocs.io/en/latest/overview.html#what-is-not-supported-and-why

Smart Contract in Python: “Hello World”

```
def Main():
    print('Hello World')
```

Main Method

- Smart Contracts can have any entry points
- But, it's recommended to use the "Main" function as the entry point
  - Easier invocation
- Usually, in the entry point, the smart contract has to handle the "trigger"
- Trigger: Verification and Application

Check Witness

- Is the caller really who he claims to be?
- “CheckWitness” command
- In more detail:
  - Verifies that the transactions or block of the *calling contract* has validated the required script hashes

```
# The trigger determines whether this smart
# contract is being run in 'verification' mode or
# 'application’
trigger = GetTrigger()

# 'Verification' mode is used when trying to
# spend assets (e.g., NEO, GAS) on behalf of
# this contract's address
if trigger == Verification():

    # if the script that sent this is the owner
    # we allow the spend
    if CheckWitness(OWNER):
        return True
    return False

# 'Application' mode is the main body of
# the smart contract
elif trigger == Application():
    if operation == 'balanceOf':
        return balanceOf()
    elif operation == 'totalSupply’:
        return totalSupply()
    ...
    return 'unknown operation'
```

Triggers

- Verification
  - Used to call the contract as a verification function
  - Multiple parameters as input, a Boolean value as output
- Application
  - Used to invoke the contract's actual application functionality
  - Accepts multiple parameters, changes the blockchain status, and returns values of any type

```
# The trigger determines whether this smart
# contract is being run in 'verification' mode or
# 'application'

trigger = GetTrigger()

# 'Verification' mode is used when trying to
# spend assets (e.g., NEO, GAS) on behalf of
# this contract's address

if trigger == Verification():

    # if the script that sent this is the owner
    # we allow the spend
    if CheckWitness(OWNER):
        return True
    return False

# 'Application' mode is the main body of
# the smart contract
elif trigger == Application():
    if operation == 'balanceOf':
        return balanceOf()
    elif operation == 'totalSupply':
        return totalSupply()
    ...
    return 'unknown operation'
```

First Smart Contract – “Address Name System”

- Convert names to Neo addresses (“ANS”)
  - Easy to remember
  - Ownership of a domain name
  - Query at any time

- Example:

Query:

test.neo

Result:

`AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y`

Smart Contract – “ANS” Template

```
from boa.interop.Neo.Runtime import CheckWitness
from boa.interop.Neo.Storage import Get, Put, Delete, GetContext


ctx = GetContext()


def Main(operation, name, addr):
    if operation == 'register’:
        return register(ctx, name, addr)
    elif operation == 'query’:
        return Get(ctx, name)
    elif operation == 'delete’:
        return delete(ctx, name)
    return False

def is_valid_addr(addr):
    if len(addr) == 20:
        return True
    return False
```

```
def register(ctx, name, addr):
    if not is_valid_addr(addr):
        return False
    if not CheckWitness(addr):
        return False
    addr_value = Get(ctx, name)
    if addr_value is not None:
        return False
    Put(ctx, name, addr)
    return True

def query(ctx, name):
    return Get(ctx, name)

def delete(ctx, name):
    owner = Get(ctx, name)
    if owner is None:
        return False
    if not CheckWitness(owner):
        return False
    Delete(ctx, name)
    return True
```

Starting two NEO Nodes – CLI

- We will use two NEO nodes connected to the private network (testing purposes)

- Open a new terminal (CLI 1):

```
$ docker exec -it neo-python1 np-prompt –p -v
```

- Open a new terminal (CLI 2):

```
$ docker exec -it neo-python2 np-prompt –p -v
```

Copy the Smart Contract to Container

- Assuming that you're in the neo-workshop directory (cloned git repo):

```
$ docker cp ./contracts/neo-ans.py neo-python1:/
```

Compile

- Go to the **CLI 1 terminal**, and compile the contract with the following command:

```
neo> sc build /neo-ans.py
```

- This command will create a file called “neo-ans**.avm**” on the docker container’s root directory

Open a Wallet

- On the **CLI 1 terminal**, open the wallet with the following command:

```
neo> wallet open /neo-python/neo-privnet.wallet
```

- This wallet was pre-generated with plenty of NEO and GAS for testing purposes
  - Address: **AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y**
  - Password: **coz**

Deploy

- On the **CLI 1 terminal**, deploy the compiled contract:

```
neo> sc deploy /neo-ans.avm True False False 070705 05
```

- Description of params:
  - Storage
  - Dynamic Invoke
  - Payable
  - Contract Parameters
  - Return Type

- **Important**
  - Copy the smart contract address and keep somewhere. We need to use that in upcoming commands.

Invoke (register a name!)

- Invoke → call the deployed smart contract

- On the **CLI 1 terminal**, run the following command to register a name:

```
neo> sc invoke 0xc4497e3f01d2efb02489be0b63072d5df8b1f8bc register
test.neo AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y
```

- Description of params:
  - Contract address
  - Operation (first parameter of the smart contract)
  - Name (second parameter of the smart contract)
  - Address (third parameter of the smart contract)

Testing: register operation really worked?

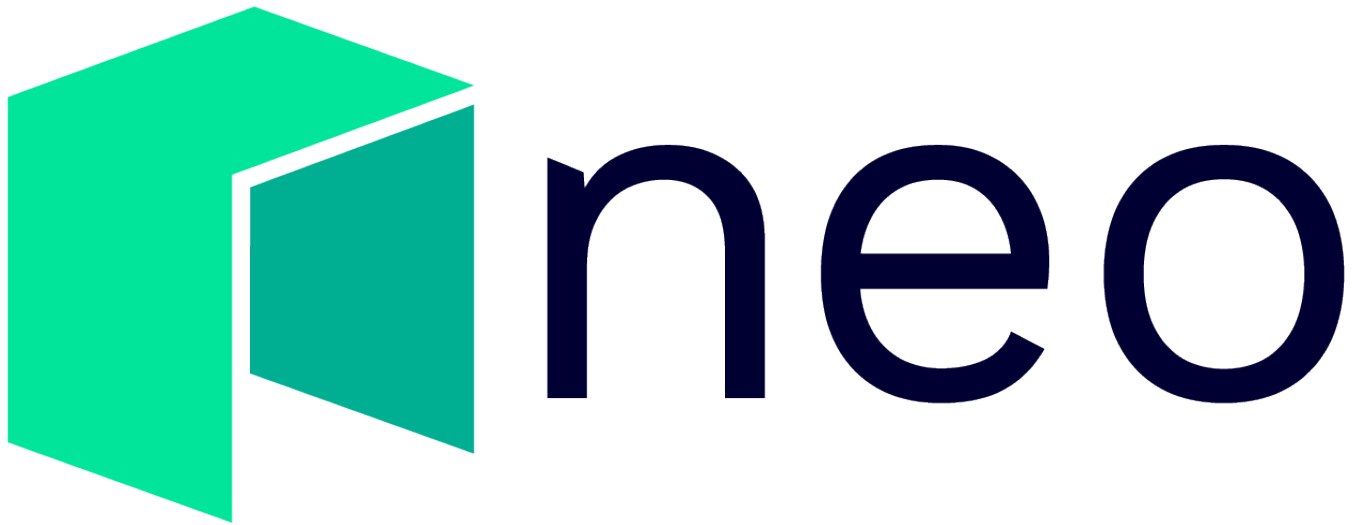
- Let's perform a query operation to check if "test.neo" really points to `AK2nJJpJr6o664CWJKi1QRXjqeic2zRp8y`
- Open a new terminal, and run the following cURL command:

```
curl -X POST http://127.0.0.1:30333/ \
  -H 'content-type: application/json' \
  -d '{
  "jsonrpc": "2.0",
  "method": "invoke",
  "params":[
    "0xc4497e3f01d2efb02489be0b63072d5df8b1f8bc",
    [{"type": "String", "value": "query"},{"type": "String", "value": "test.neo"},
{"type": "ByteArray", "value": ""}]
  ],
  "id": 1
}'
```

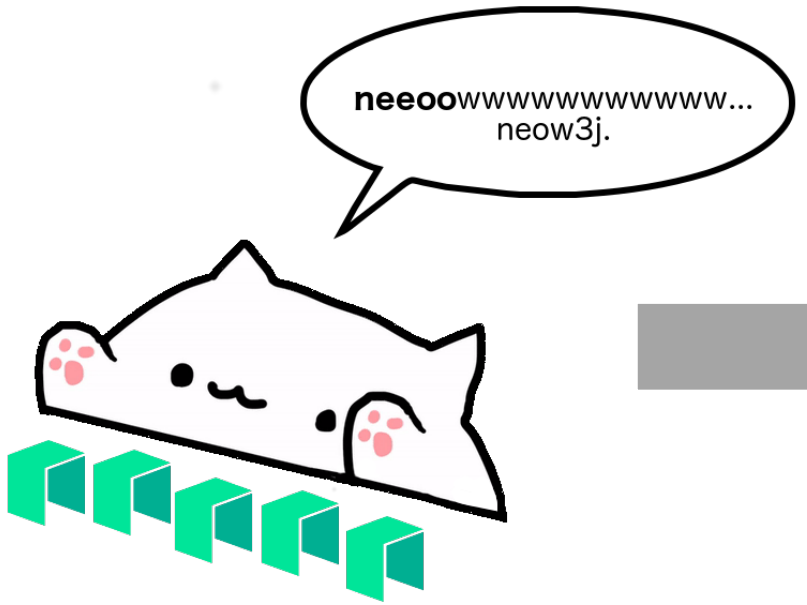
Smart Contract address

Operation (query)

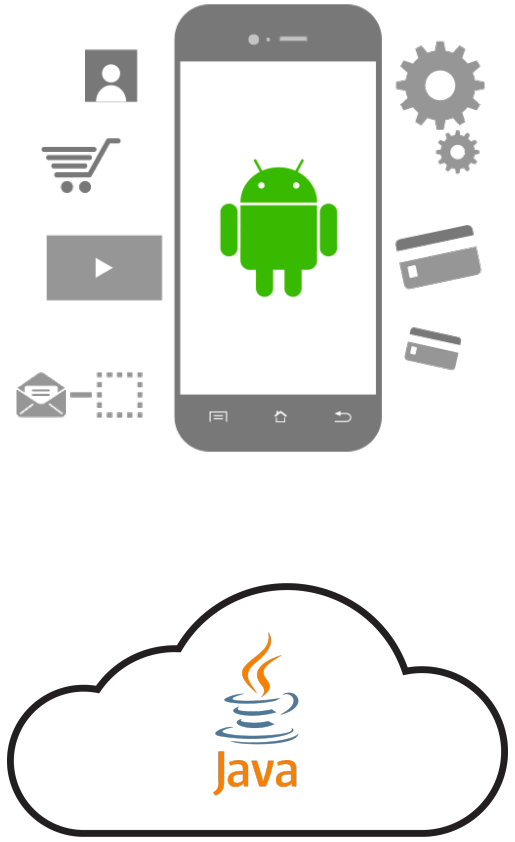
Name to query


neo

neow3j Project

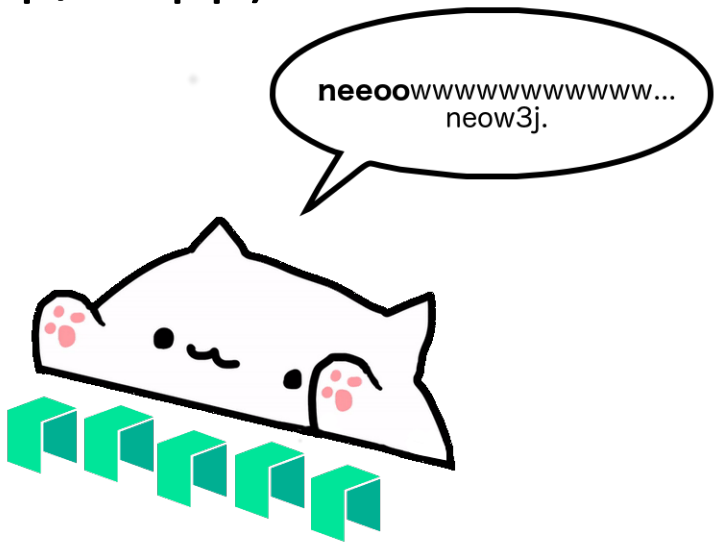


https://neow3j.io



Contribute to projects and be part of Neo!

- Have an idea to extend or enhance existing features of neow3j?
- Have an idea to build on top of neow3j (app/dApp)?

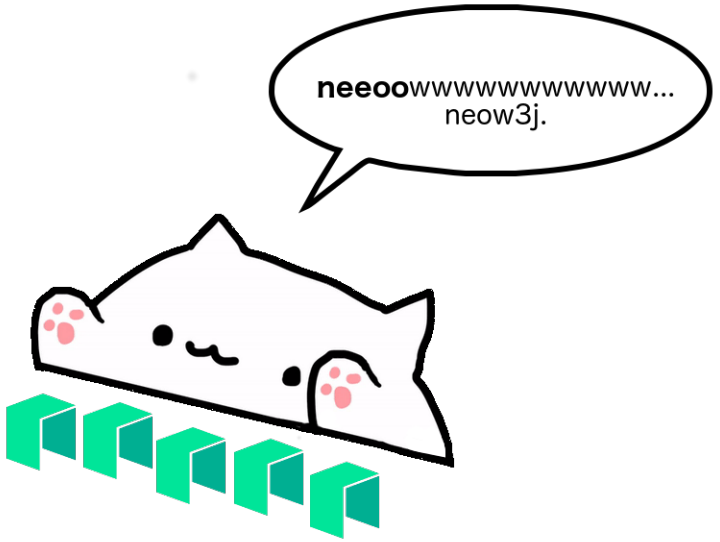


https://neow3j.io

- If you like the project, **give us a star on GitHub!**

Thanks!

- Questions?



https://neow3j.io

AXLABS

https://axlabs.com

neo

https://neo.org

References

- https://docs.neo.org/en-us/sc/introduction.html
- https://github.com/neo-ngd/NEO-Tutorial
- https://github.com/AxLabs/neo-workshop
- https://github.com/CityOfZion/python-smart-contract-workshop
- https://neo-boa.readthedocs.io/en/latest/overview.html