Deploying your first contract with Python

Jason Carver

Ethereum Foundation

Oct 31, 2018

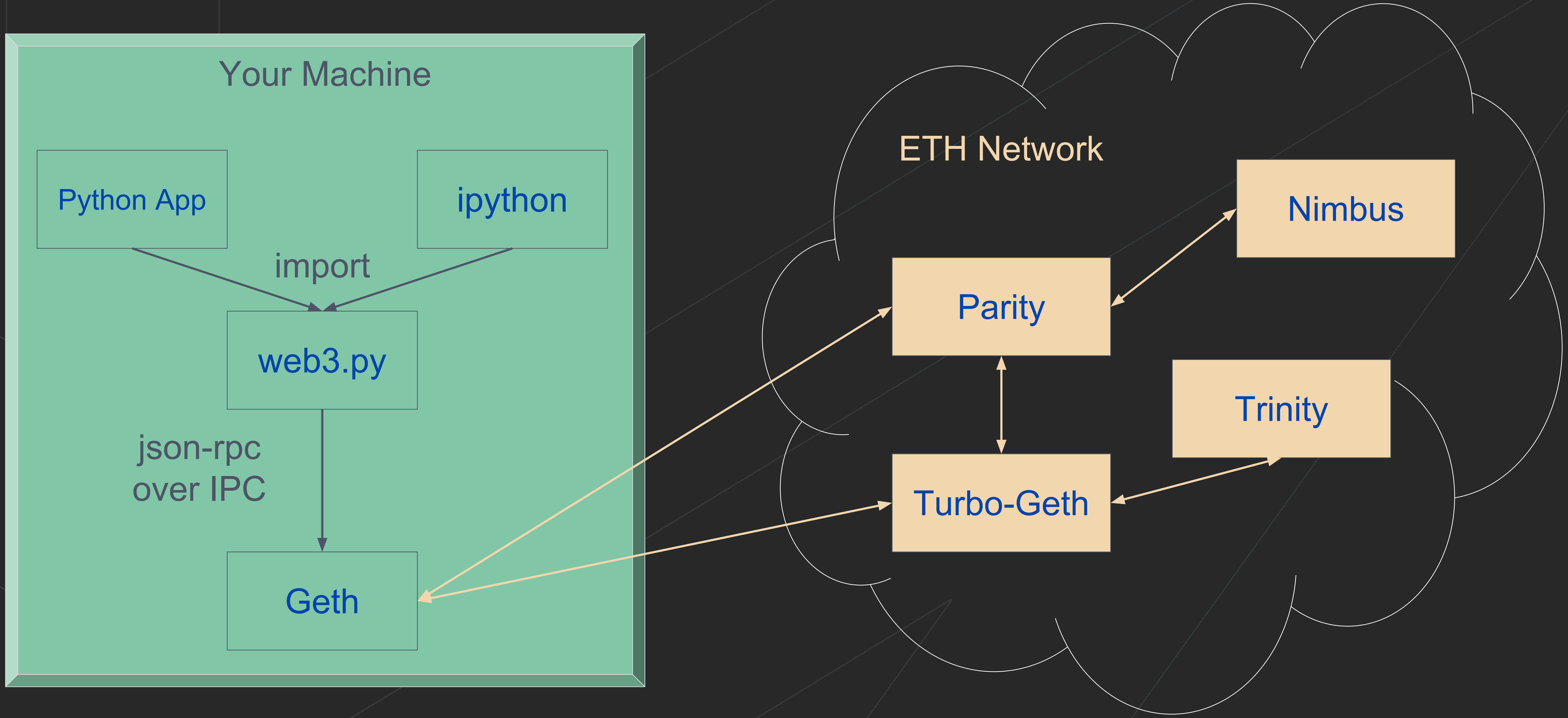
Primary Goals

- Install & use web3.py
- Deploy new token to testnet
- Trade tokens with neighbor

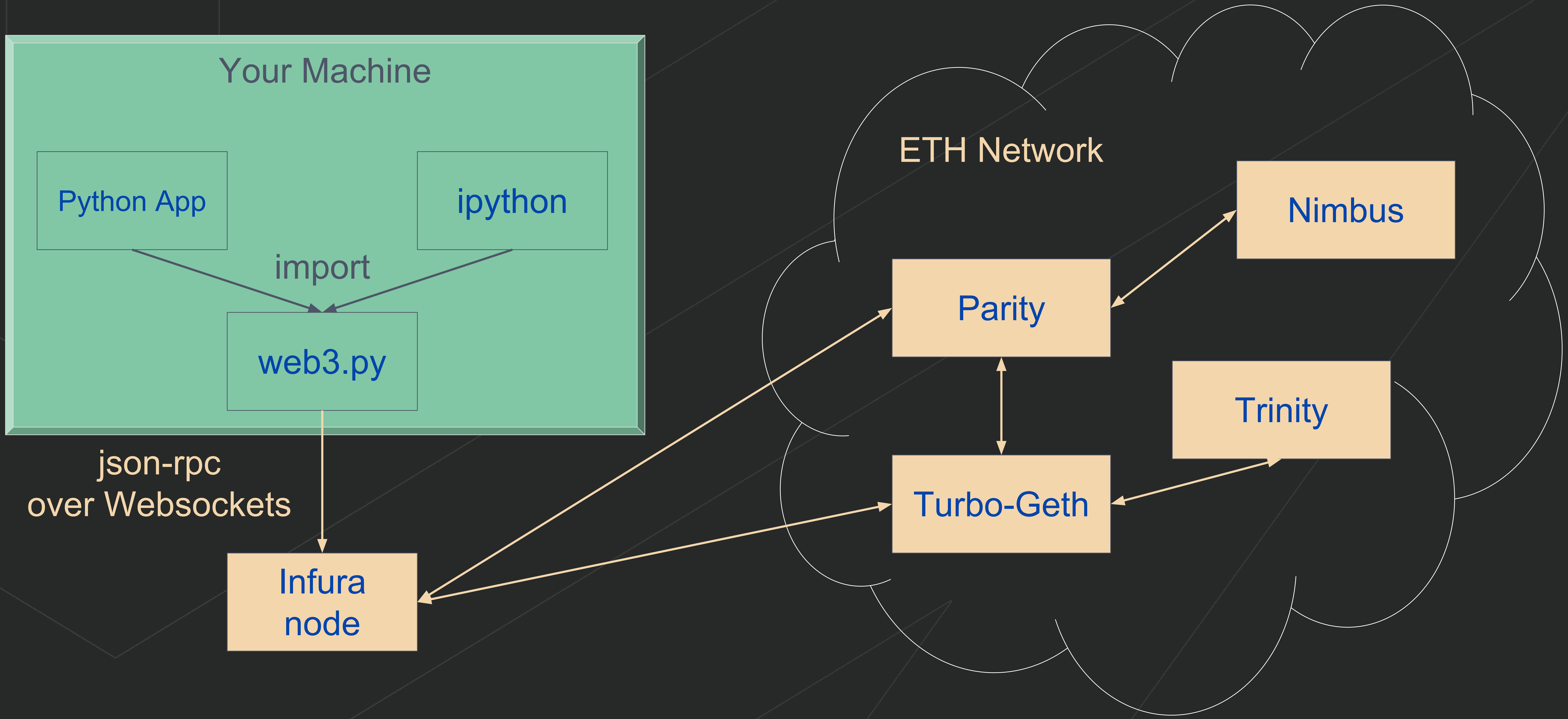
You should already be

- Familiar with Python
- Familiar with Ethereum
- Comfortable at your shell

What is web3?


Your Machine
Python App
ipython
import
web3.py
json-rpc
over IPC
Geth
ETH Network
Nimbus
Parity
Trinity
Turbo-Geth

Web3 with remote hosting



Why Web3.py?

- Rough API alignment with web3.js
- Good for: console, Django, etc
- Not good for: JS shops, DApps*

*except Shadowlands, someday

Follow Along

https://is.gd/LAEED9

Chat: gitter.im/ethereum-py/intro-workshop

This is a live-fire exercise.

Expect some snags.

Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Install web3.py

Install web3.py

- Install Python 3
- Create virtual environment
- `pip install web3`

Install Python 3

- See if you already have it

  ```
  $ which python3 || echo "No python3 found, try steps below"
  ```

- On GNU/Linux

  Debian-ish: `$ sudo apt-get install python3.7-dev python3-venv python3-pip`

  Other: build locally - see gist

- On Mac

  ```
  $ xcode-select --install
  $ brew install python --with-brewed-openssl
  ```

- On Windows 10 - VirtualBox with Ubuntu 18.10

Create virtual environment

- Create virtual environment

```
$ python3 -m venv ~/.venv-web3py
```

- Alternative: use pip and virtualenv

```
$ which pip3 || which pip || easy_install pip || sudo easy_install pip
$ which virtualenv || sudo $(which pip3 || which pip) install --upgrade virtualenv
$ virtualenv -p python3 ~/.venv-web3py
```

- Enter the virtual environment

```
$ source ~/.venv-web3py/bin/activate
```

- Upgrade your package management & CLI infrastructure

```
$ pip install --upgrade pip setuptools ipython certifi
```

pip install web3

- Make sure to get the latest

```
$ pip install --upgrade web3
```

- Dependencies requires extra libraries: setup docs

- Confirm installation

```
$ ipython
>>> from web3 import Web3
>>> Web3.toText(hexstr="0xf09fa684")
```

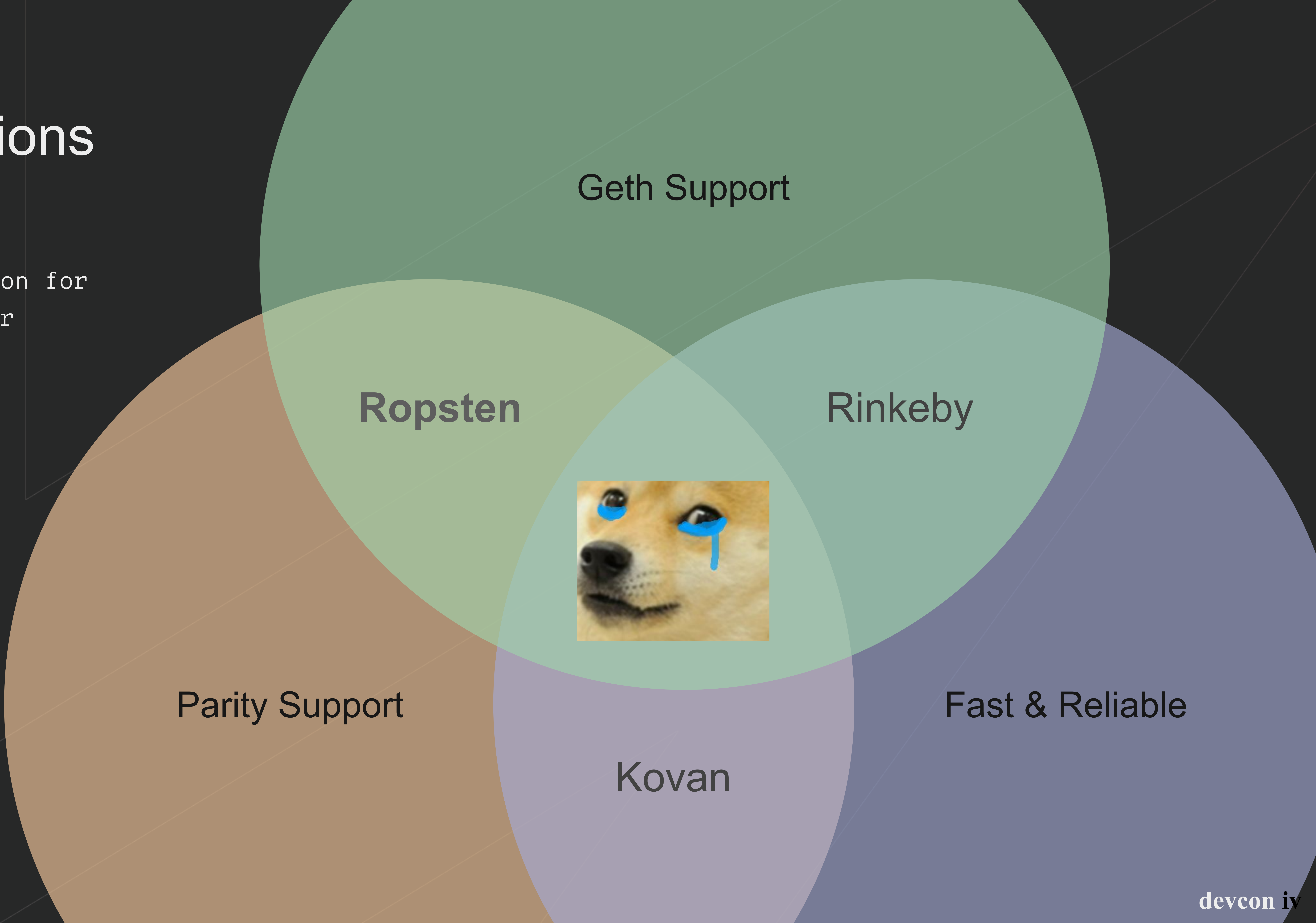
Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Connect to testnet

Testnet options

No single best option for testnet. Choose your favorite tradeoff.



Geth Support

Ropsten

Rinkeby

Parity Support

Fast & Reliable

Kovan

Connect to testnet

- Make sure you are in virtualenv

```
$ source ~/.venv-web3py/bin/activate
```

- Open python console

```
$ ipython
```

- Get web3 instance with Ropsten

```
>>> from web3.auto.infura.ropsten import w3
```

- Confirm connection & network

```
>>> block = w3.eth.getBlock(4281234)
>>> block.hash
HexBytes('0xc34ed62271c4d9e38e2d85fbe29ca4ce5c6a609b06d0fbb7ada21de79e9ade32')
```

Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Fund local account

Fund local account

- Use paper wallet
- Scan key
- Import private key
- Confirm balance

Use paper wallet

- For our purpose, a private key is 32 bytes of random data

- QR code is a hex-encoded copy of that key

- Paper wallets have very weak security properties

Scan key

- Open webqr.com
- Allow webcam usage, and hold paper wallet up to webcam
- Copy the private key from the scanned result

Import private key

- Convert private key to account

```
>>> acct = Account.privateKeyToAccount("0x34315962...")
>>> acct.address
"0x1234..."
```

Confirm Balance

- Show balance in Ropsten ETH

```
>>> balance = w3.eth.getBalance(acct.address)
>>> w3.fromWei(balance, "ether")
Decimal('1.337')
```

Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Deploy token contract

Deploy token contract

- Generate bytecode/ABI from source
- Build deployment transaction
- Sign & broadcast transaction
- Confirm token balance
- Verify source on Etherscan

Generate bytecode/ABI from source

- Copy source from this gist into remix.ethereum.org
- Copy Bytecode as-is into a variable

```
>>> bytecode = <PASTE_HERE>
```

- Extract object field

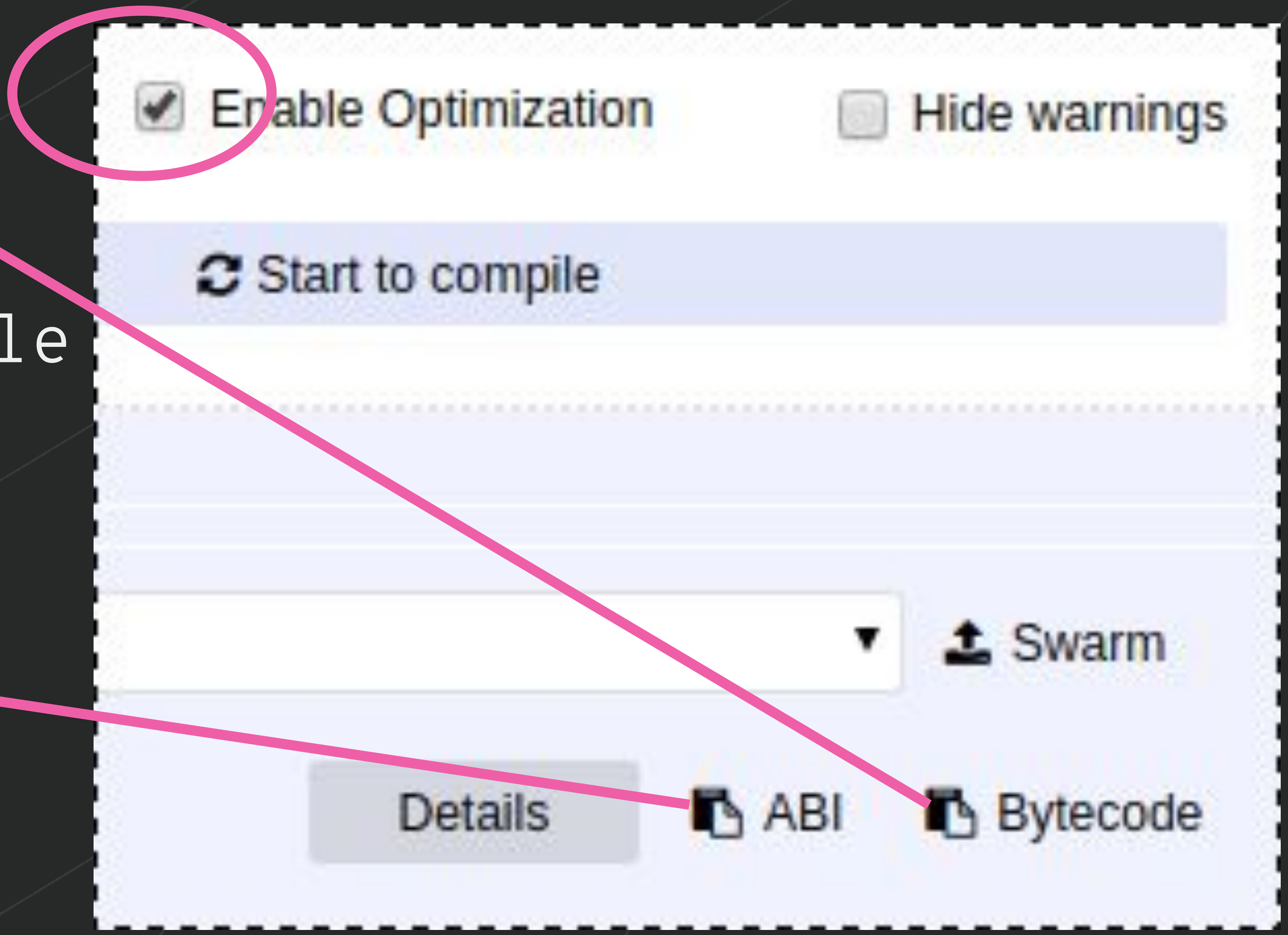
```
>>> token_code = bytecode['object']
```

- Copy ABI into a string variable

```
>>> token_abi = '''

<PASTE_HERE>

'''.strip()
```



☑ Enable Optimization □ Hide warnings

Build deployment transaction

- Build the contract object in preparation for deployment

```
>>> token_deployer = w3.eth.contract(abi=token_abi, bytecode=token_code)
```

- Build transaction

```
>>> init = token_deployer.constructor(1000)
>>> basic_txn = init.buildTransaction({'gas': 320000})
```

- Flesh out the transaction for local signing

```
>>> next_nonce = w3.eth.getTransactionCount(acct.address)
>>> signable_transaction = dict(
  basic_txn,
  nonce=next_nonce,
  gasPrice=w3.toWei(3, 'gwei'),
)
```

Sign and Broadcast Transaction

- Sign transaction

```
>>> signature_info = acct.signTransaction(signable_transaction)
```

- Broadcast transaction

```
>>> txn_hash = w3.eth.sendRawTransaction(signature_info.rawTransaction)
```

- Wait for the transaction to be mined

```
>>> receipt = w3.eth.waitForTransactionReceipt(txn_hash)
# ... console freezes until transaction is mined
```

Confirm token balance

- Get deployed contract

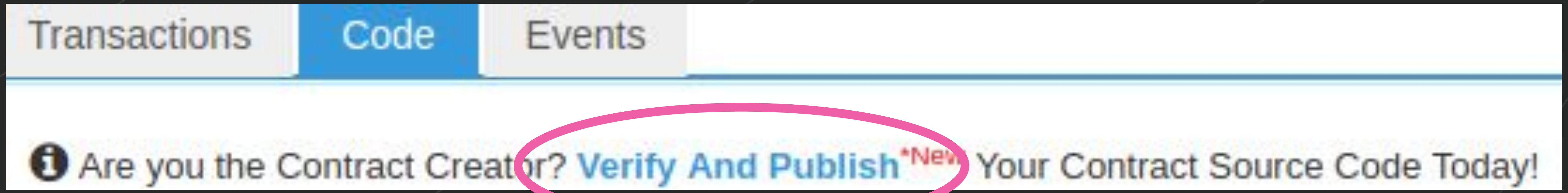
```
>>> token = w3.eth.contract(address=receipt.contractAddress, abi=token_abi)
```

- Check your balance

```
>>> token.functions.balanceOf(acct.address).call()
1000
```

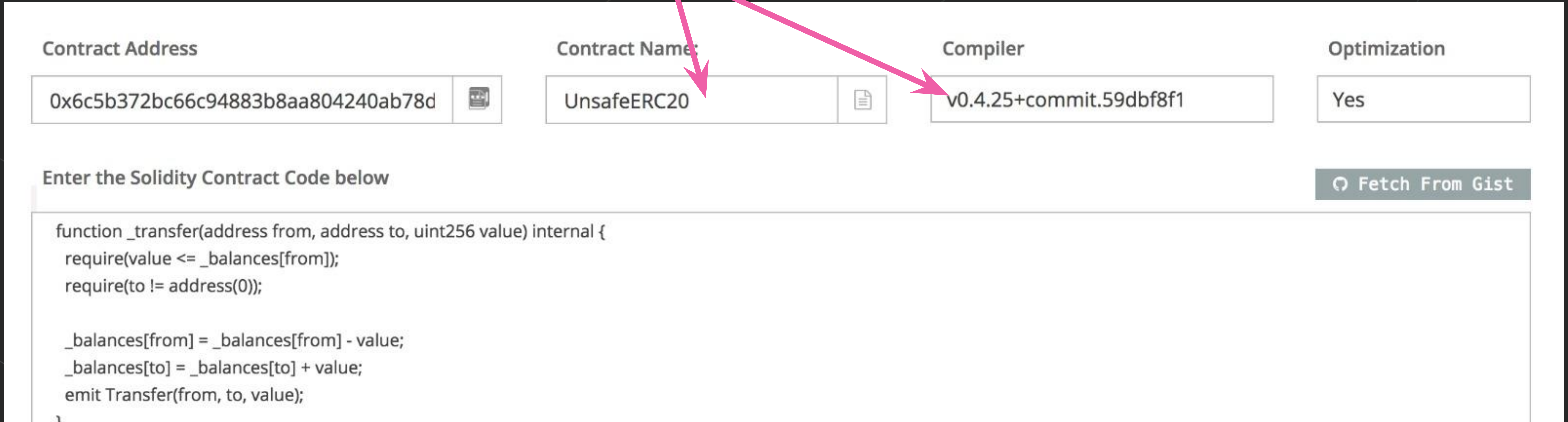
Verify source on Etherscan

- `Find contract at ropsten.etherscan.io/address/token.address`
- `Copy in source`



Are you the Contract Creator? Verify And Publish *New Your Contract Source Code Today!

- `Set verification parameters`



Contract Address: 0x6c5b372bc66c94883b8aa804240ab78d

Contract Name: UnsafeERC20

Compiler: v0.4.25+commit.59dbf8f1

Optimization: Yes

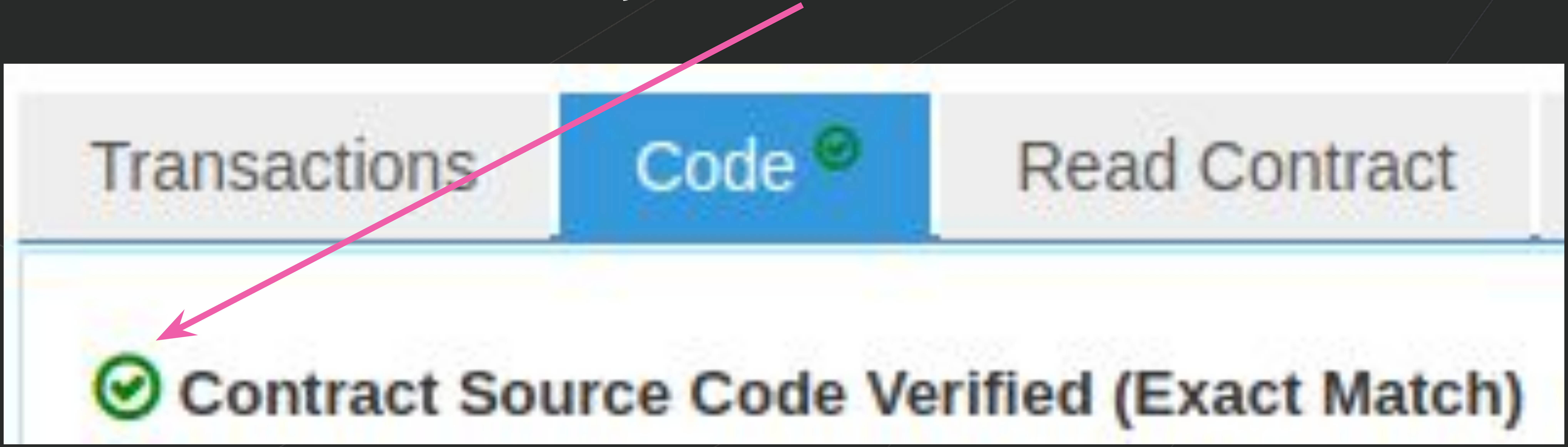
Enter the Solidity Contract Code below

```
function _transfer(address from, address to, uint256 value) internal {
  require(value <= _balances[from]);
  require(to != address(0));

  _balances[from] = _balances[from] - value;
  _balances[to] = _balances[to] + value;
  emit Transfer(from, to, value);
```

Verify source on Etherscan

- 3rd-party service is unfortunately the most convenient
- EthPM talk tomorrow for preview of trustless verification
- Until then, look for green check



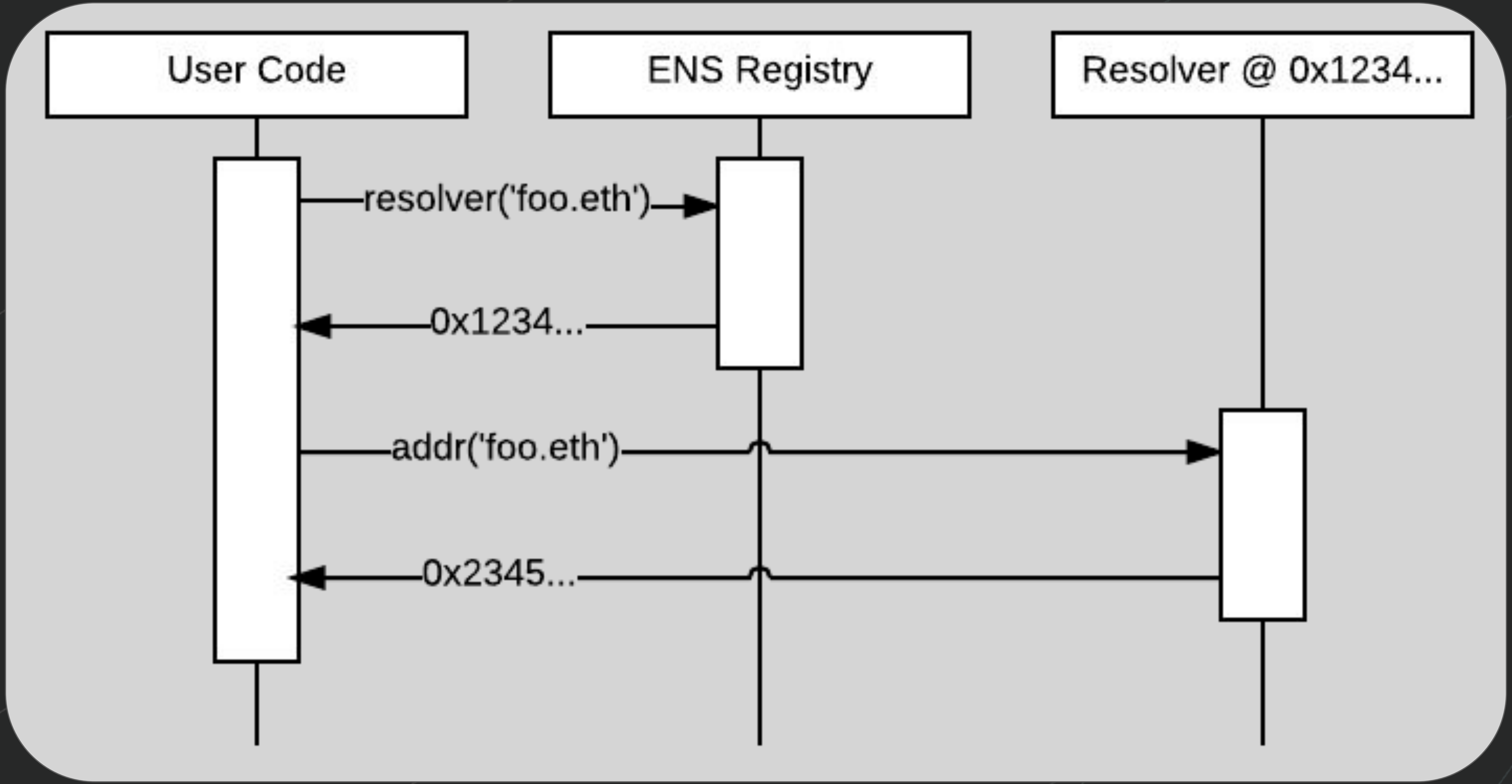
✓ Contract Source Code Verified (Exact Match)

Workshop Outline

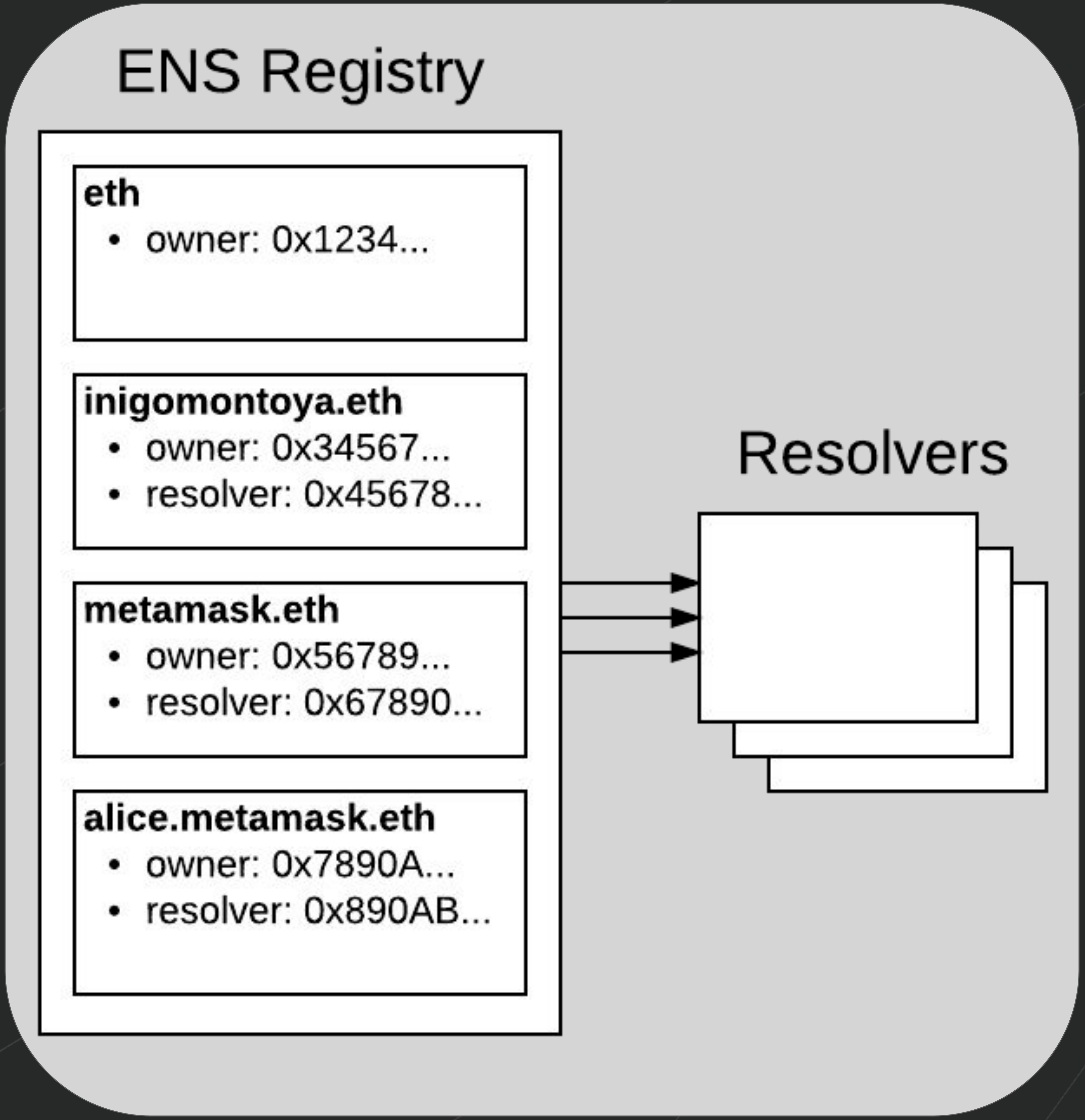
- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Assign an ENS name

How does ENS work?



How does ENS work?



ENS Registry

eth

- owner: 0x1234...

inigomontoya.eth

- owner: 0x34567...
- resolver: 0x45678...

metamask.eth

- owner: 0x56789...
- resolver: 0x67890...

alice.metamask.eth

- owner: 0x7890A...
- resolver: 0x890AB...

Resolvers

Assign an ENS name

- Connect to Ropsten ENS
- Acquire an ENS name
- Set up resolver
- Point name at contract address
- Give neighbor your contract ENS name, verify address

Connect to Ropsten ENS

- Build ENS object for custom Ropsten deployment

```
>>> from ens import ENS
>>> rns = ENS(
  w3.providers,
  addr="0xbaB9717617D7e50264dE6Ee0Ef152a7CA452CF9C")
```

- Verify connection by looking up an address

```
>>> rns.address('jarjar.test')
'0x13764E8D95F1a659E35274Cf7e8bDf7Cc05188D6'
```

Acquire an ENS name

- Connect to giveaway registrar

```
>>> reg_addr = rns.address('test')
>>> reg_abi = [{'inputs': [{'name': 'label', 'type': 'bytes32'},
   {'name': 'owner', 'type': 'address'}],
  'name': 'register', 'outputs': [], 'payable': False,
  'stateMutability': 'nonpayable', 'constant': False,
  'type': 'function'}]
>>> reg = w3.eth.contract(address=reg_addr, abi=reg_abi)
```

- Get a name

```
>>> my_hash = rns.labelhash('carvertoken')
>>> txn = reg.functions.register(my_hash, acct.address).buildTransaction…
```

Set up resolver

- Use public resolver for your new name

```
>>> resolver_addr = rns.address('resolver.eth')
>>> raw_ens = rns.ens._classic_contract
>>> txn = raw_ens.functions.setResolver(
  rns.namehash('carvertoken.test'),
  resolver_addr,
).build...
```

- Load resolver contract

```
>>> resolver_abi = [{"inputs": [
   {"name": "node", "type": "bytes32"}, {"name": "addr", "type": "address"}],
  "constant": False, "name": "setAddr", "outputs": [],
  "stateMutability": "nonpayable",
  "payable": False, "type": "function"}]
>>> resolver = w3.eth.contract(address=resolver_addr, abi=resolver_abi)
```

Point name at contract

- In public resolver, point name at contract

```
>>> namehash = rns.namehash('carvertoken.test')
>>> txn = resolver.functions.setAddr(namehash, token.address).buildTr...
```

- Confirm that your name now resolves to your contract

```
>>> rns.address('carvertoken.test') == token.address
True
```

Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

Swap tokens

Swap tokens with neighbor

- Get ENS name for local account
- Send token to neighbor's local account
- Verify token balance from neighbor
- Transaction "cancellation"
- Take-home exercise: Atomic Swap

Get ENS name for local account

- Get name from registrar, this time just "<username>.test"
- Set resolver for new name to the public resolver
- On the resolver contract, set address to your local account

```
>>> namehash = rns.namehash('carver.test')
>>> resolver.functions.setAddr(namehash, acct.address).buildTransaction...
```

Send token to neighbor

- Get ENS name to neighbor local account

- Look up neighbor's local account name

```
>>> neighbor_addr = rns.address('neighbor.test')
```

- Send random number of tokens to neighbor

```
>>> import random
>>> amt = random.randint(0, 9)
>>> txn = token.functions.transfer(neighbor_addr, amt).buildTransaction…
```

Verify token balance from neighbor

- Get neighbor's token address

```
>>> neighbor_token_addr = rns.address('neighbortoken.test')
```

- Load neighbor's token contract

```
>>> neighbor_token = w3.eth.contract(neighbor_token_addr, abi=token.abi)
```

- Check received tokens

```
>>> neighbor_token.functions.balanceOf(acct.address).call()
3
# check with neighbor to confirm how many tokens they sent
```

Transaction "cancellation"

- Send transaction with low gas price

```
>>> txn = token.functions.transfer(neighbor_addr, 20).buildTransaction…
>>> nonce = w3.eth.getTransactionCount(acct.address)
>>> price = 1
```

- Confirm on Etherscan that transaction is pending

- Send new transaction with same nonce & higher gas price

```
>>> signature_info = acct.signTransaction(dict(
  nonce=nonce,                       # same nonce as pending transaction above
  gasPrice=w3.toWei(5, 'gwei'),      # gas price must be at least 10% higher
  to=acct.address,
  value=0,
  gas=21000,
))
```

Take-home exercise: Atomic Swap

- Trade tokens without worry of being cheated
- Variant: swap tokens for ether
- There can be a few subtleties to get this swap correct
- See decentralized exchanges for a general solution

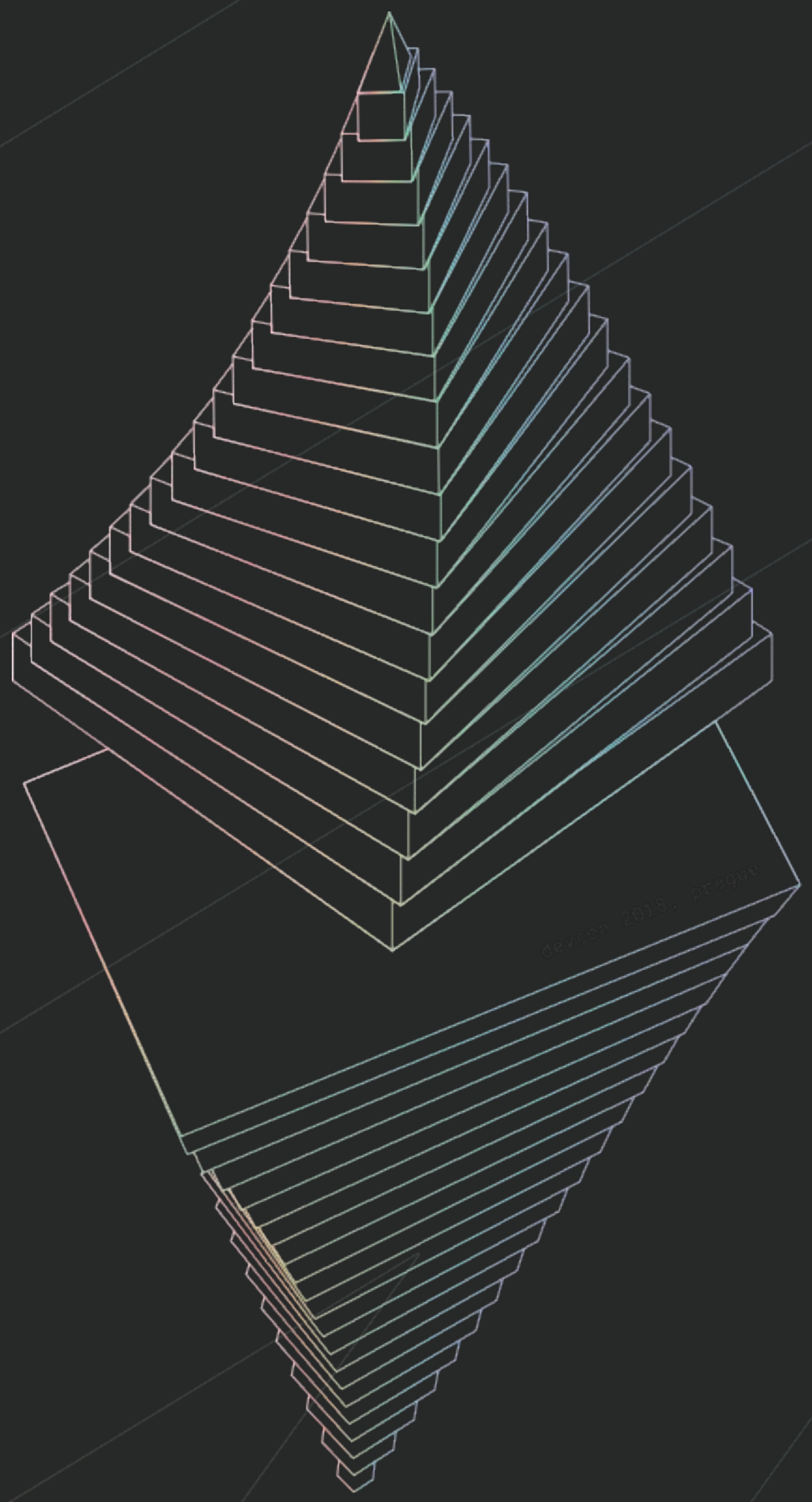
Workshop Outline

- Install web3.py
- Connect to testnet
- Fund local account
- Deploy token contract
- Assign an ENS name
- Swap tokens

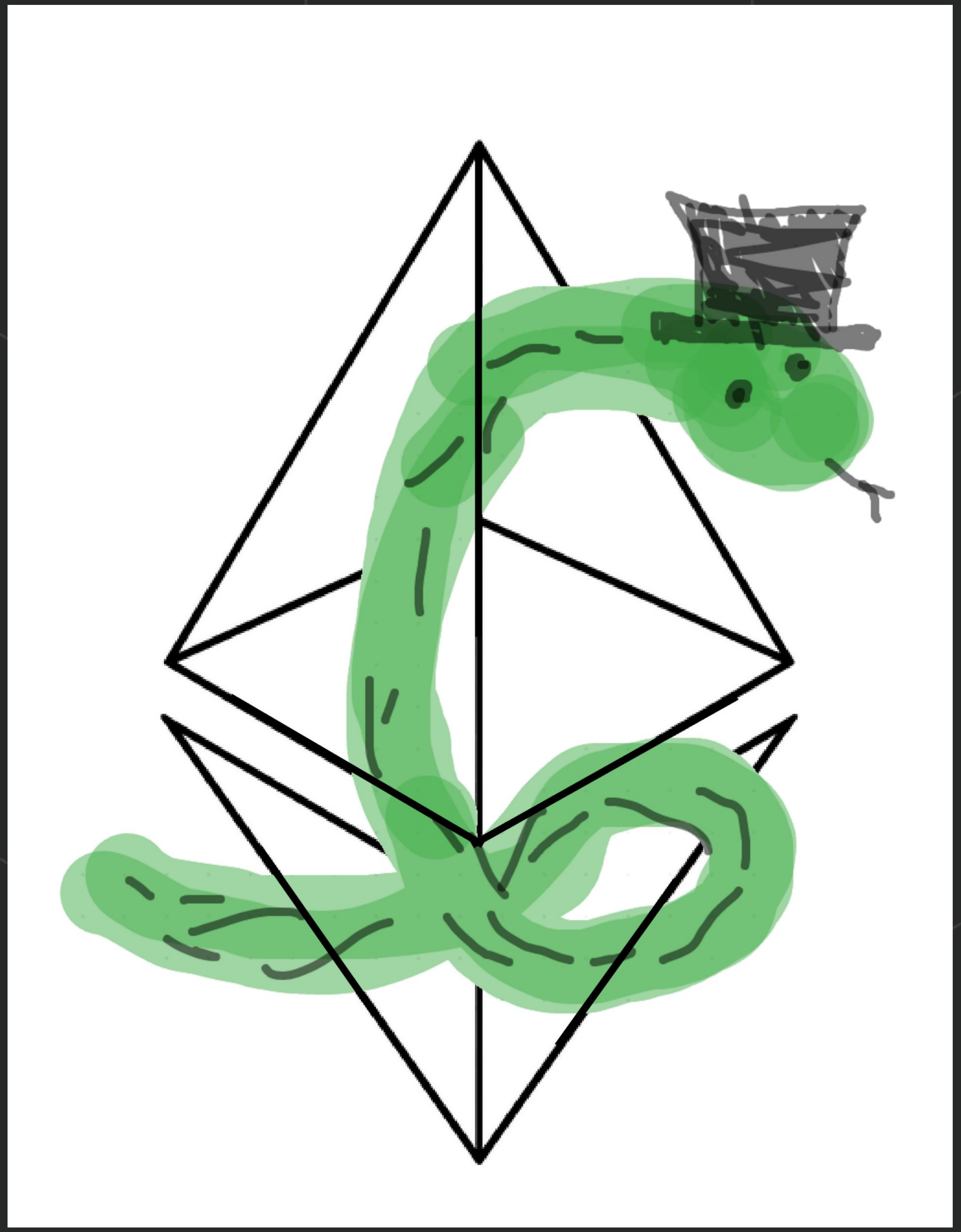
You feel more confident in your spell-casting skills

Huge thank you to the volunteers!

- Johns Beharry
- Dylan Wilson
- Bryant Eisenbach
- Christoph Burgdorf
- Nick Gheorghita
- Keri Clowes



Thank you!



Comments or suggestions for next time?

Open a ticket on:

github.com/ethereum/web3.py/issues

Jason Carver

@carver on github

Reference Slides

Getting your hex account address to your neighbor

- Print your local address

  ```
  >>> acct.address
  '0x1234...'
  ```

- Open webqr.com/create.html

- Copy address into field and click Create

- Have neighbor open webqr.com and scan it from your QR code

  ```
  >>> neighbor_addr = "0xaBcD..."
  ```

Connect to Infura over HTTPS

- Register with Infura for API key

- Set API key & https environment variables

```
$ export WEB3_INFURA_API_KEY="l33th4x0r"
$ export WEB3_INFURA_SCHEME="https"
```

- Launch ipython

```
$ ipython
```

- Load configured web3 instance

```
>>> from web3.auto.infura.ropsten import w3
```

Create local Ethereum account

- Create a local private key

```
>>> from eth_account import Account
>>> acct = Account.create('smash keyboard for bonus entropy in private key')
```

- Congratulations, you have an account at

```
>>> acct.address
'0x0123456789abcdef...'
```

Verify local account balance

- Check out some eth-account Account APIs:

```
>>> help(acct)
 |  encrypt(self, password)
 |  signHash(self, message_hash)
 |  signTransaction(self, transaction_dict)  # <- Today we'll use this one
```

- Check balance

```
>>> w3.eth.getBalance(acct.address)
0
```

- Naturally, account is empty

Fund local account: Android

- Import private key from paper wallet
- Make QR code for local address
- Send paper Ropsten ETH to local address

Import private key from paper wallet

- Sidebar -> Keys -> (*) ECDSA
- Scan QR code
- Switch network to Ropsten
- Sidebar -> Accounts -> Choose new account
- Confirm ETH balance on mobile

Make QR code for local address

- Display local address again

```
>>> acct.address
'0x8c9E19726f9a30aDE3B4b7d371761eA7dA35c1C5'
```

- Visit webqr.com/create.html

- Paste in address without quotes, like:

```
[ 0x8c9E19726f9a30aDE3B4b7d371761eA7dA35c1C5 ]
```

- Click Create

Send paper Ropsten ETH to local address

- In wallet view, click green arrow in top right
- Under To, click camera to scan QR code for local address
- Send 0.5 ETH
- Press green key circle
- Confirm ETH at local address

```
>>> balance = w3.eth.getBalance(acct.address)
>>> w3.fromWei(balance, 'ether')
Decimal('0.5')
```