2019 NORTH AMERICA CACS™

AN ISACA EVENT

Blockchain, Blockchain Security and the Basics of Blockchain Auditing

May 11 - 12, 2019

Day 2

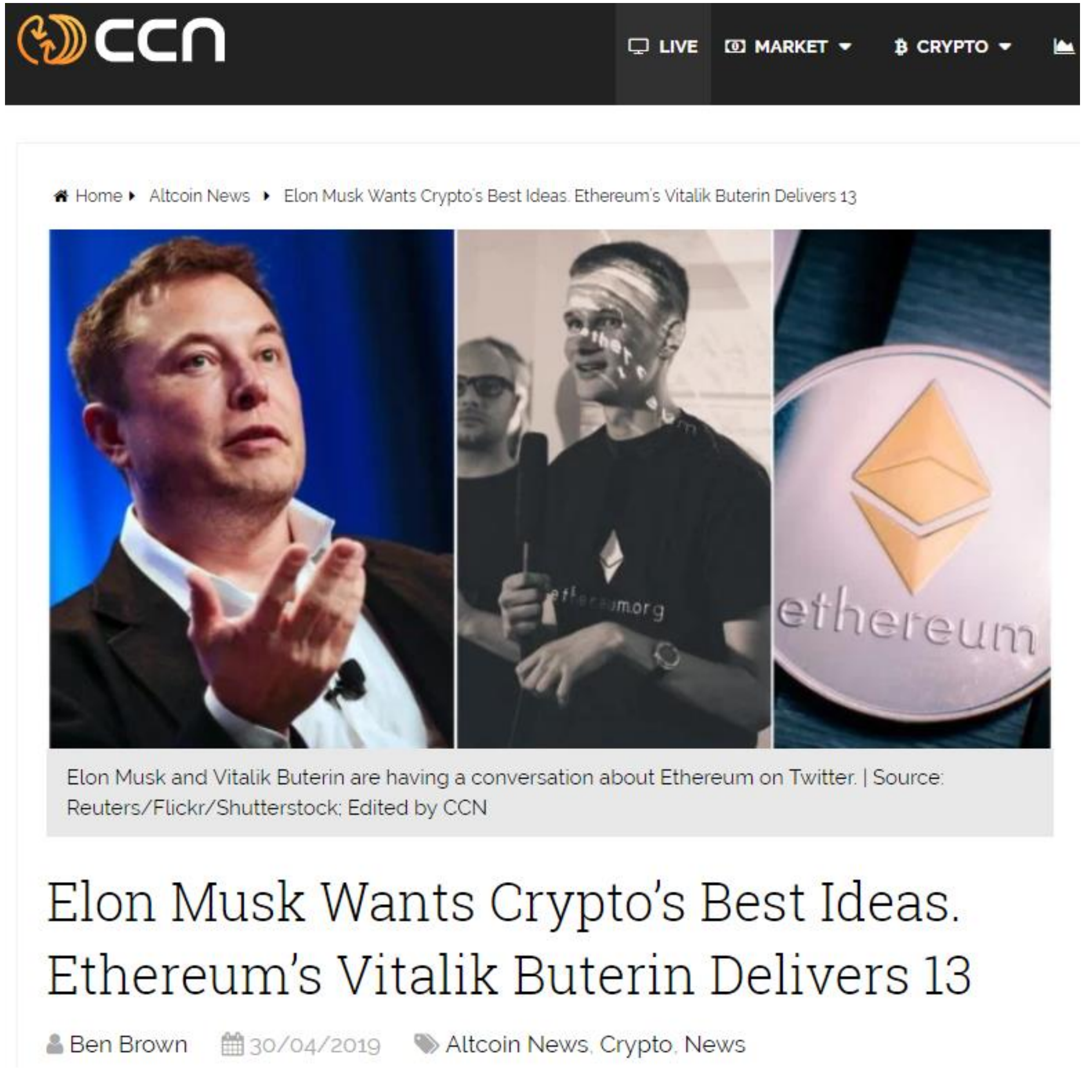
William Favre Slater, III

M.S., MBA, PMP, CISSP, CISA, SSCP, Security+, ITILv3

ISACA Gold Member

50 ISACA

Blockchain in the News – April 30, 2019



Home ▸ Altcoin News ▸ Elon Musk Wants Crypto's Best Ideas. Ethereum's Vitalik Buterin Delivers 13

Elon Musk and Vitalik Buterin are having a conversation about Ethereum on Twitter. | Source: Reuters/Flickr/Shutterstock; Edited by CCN

Elon Musk Wants Crypto's Best Ideas. Ethereum's Vitalik Buterin Delivers 13

Ben Brown 30/04/2019 Altcoin News, Crypto, News

Source: https://www.ccn.com/elon-musk-best-ethereum-ideas-vitalik-buterin

Blockchain in the News – April 30, 2019

Elon Musk Wants Crypto's Best Ideas. Ethereum's Vitalik Buterin Delivers 13

1. A Globally Accessible Financial System
2. "Sign in With Ethereum" Options
3. Secure & Transparent Registries
4. Experiment with New Forms of Governance & Human Organization
5. Micropayments
6. Markets for Personal Data
7. Spam Prevention in Social Networks
8. Micropayment Schemes for Publishers of Good Content
9. Testing Grounds for New Market Designs
10. Charity Stickers for Donations
11. Peer-to- Peer Marketplaces for Internet Connections
12. Identity, Reputation, And Credit Systems
13. Decentralized DNS Alternatives

Elon Musk and Vitalik Buterin are having a conversation about Ethereum on Twitter. | Source: Reuters/Flickr/Shutterstock; Edited by CCN

Elon Musk Wants Crypto's Best Ideas. Ethereum's Vitalik Buterin Delivers 13

Source: https://www.ccn.com/elon-musk-best-ethereum-ideas-vitalik-buterin

Agenda – Day 1 & Day 2

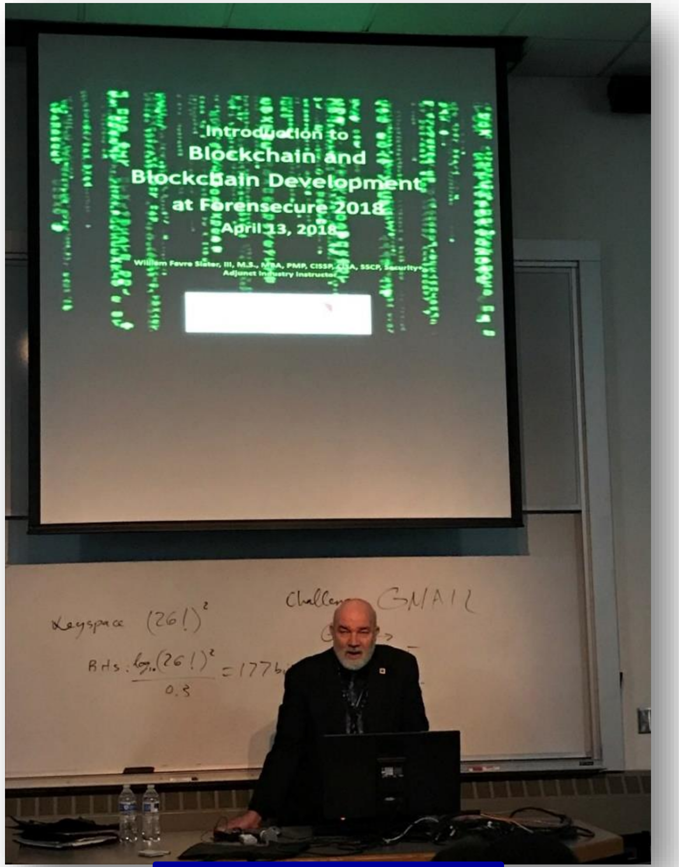
High-level Outline:

Day 1

Topic 1: History of Money and Conventional Ledger Functions
Topic 2: Bitcoin Basics
Topic 3: Tokenized Economy and Crypto Currency Concepts
Topic 4: Blockchain Technology
Topic 5: Ethereum Blockchain Technology
Topic 6: Blockchain Beyond Bitcoin
Topic 7: Blockchain Limits and Challenges
Topic 8: Blockchain Security
Topic 9: Examples of Real-world Blockchain Applications
Topic 10: The Ethereum EVM, Smart Contracts, and Solidity
Topic 11: How to Design and Implement a Blockchain Solution Project – an Organized High-Level Step-by-Step Approach
Topic 12: How to Help your Organization Rapidly Ramp Up Skills and Readiness for Blockchain Application Development

Day 2

Topic 1: Getting started with Blockchain Application Development – Setting up the Workbench
Topic 2: Truffle Framework Introduction
Topic 3: Example DApp using Truffle, HTML, CSS, Solidity, the EVM and Ethereum Blockchain
Topic 4: Solidity and Ethereum Blockchain Fundamentals
Topic 5: Javascript and Ethereum Blockchain Fundamentals
Topic 6: Example DApp using HTML, CSS, Solidity the EVM and the Ethereum Blockchain
Topic 7: Blockchain and Auditing
Topic 8: How to Secure Blockchain infrastructure and applications
Topic 9: How to perform Secure Software Development for Blockchain applications by design, coding practices, testing and verification
Topic 10: Concepts of Auditing the Data and Transactions in Blockchain Data Structures
Topic 11: Automating the Auditing of Blockchains and Blockchain Applications



William Favre Slater, III
Forensecure 2018

Agenda – Day 2

Day 2

Topic 1: Getting started with Blockchain Application Development – Setting up the Workbench

Topic 2: Truffle Framework Introduction

Topic 3: Example DApp using Truffle, HTML, CSS, Solidity, the EVM and Ethereum Blockchain

Topic 4: Solidity and Ethereum Blockchain Fundamentals

Topic 5: Javascript and Ethereum Blockchain Fundamentals

Topic 6: Example DApp using HTML, CSS, Solidity the EVM and the Ethereum Blockchain

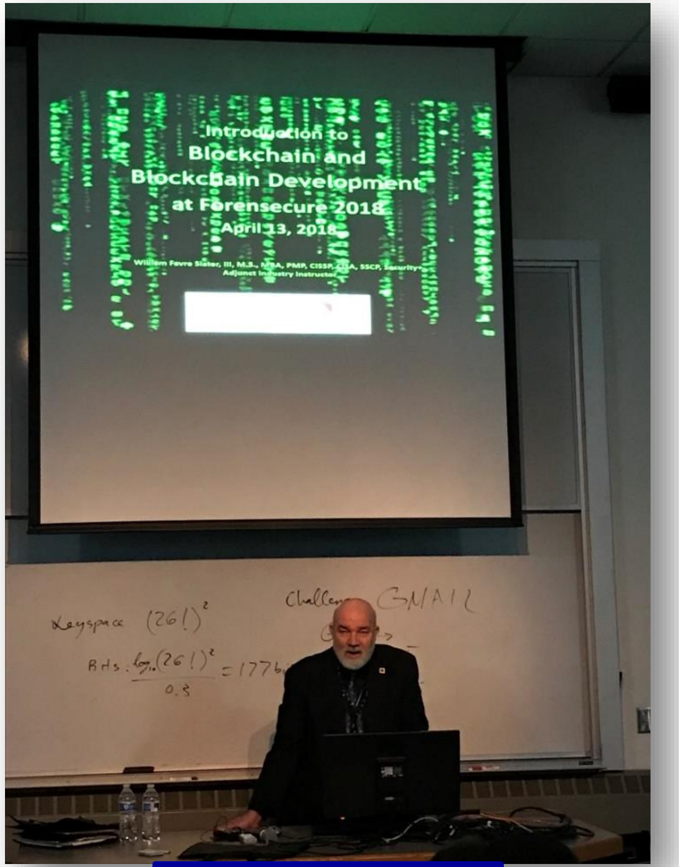
Topic 7: How to Secure Blockchain infrastructure and applications

Topic 8: How to perform Secure Software Development for Blockchain applications by design, coding

Topic 9: Blockchain and Auditing practices, testing and verification

Topic 10: Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Topic 11: Automating the Auditing of Blockchains and Blockchain Applications


Introduction to
Blockchain and
Blockchain Development
at Forensecure 2018
April 13, 2018
William Favre Slater, III, M.S., MBA, PMP, CISSP, CISA, SSCP, Security+
Adjunct Industry Instructor

William Favre Slater, III
Forensecure 2018

**** CAUTION ****

Blockchain and Blockchain DApp Development are sophisticated applied technologies that work together to provide trusted computing.

They are built on complex rules with the objective of providing reliable, trusted, anonymous transactions on decentralized distributed ledgers via the Internet.

It took the time, experience, knowledge and hard work of many geniuses to mature the technology.

It takes time, energy, patience and many hours of study to just begin to wrap your head around it.

If you are lazy or have a short attention span, or are overwhelmed after this presentation these topics are probably not a good career direction for you.

This path will not be easy, but it will be worthwhile if you are up for investing your time and energy to learn it.

As of February 2018, there are 14 open positions for every single Blockchain engineer who is looking for a job.

More Extremely Important Notes

This presentation is not about CRYPTOCURRENCY, only BLOCKCHAIN

Please clear your mind about everything you thought you knew about BLOCKCHAIN before this presentation.

BLOCKCHAIN MUCH bigger than you think.

Blockchain is moving SO FAST that a **“Blockchain Year”** is considered to be about 30 days

I have multiple decades of experience in software and application development. To say the **learning curve** “**humbling**” would be an ***understatement***.

The only way you will get to be excellent in this:

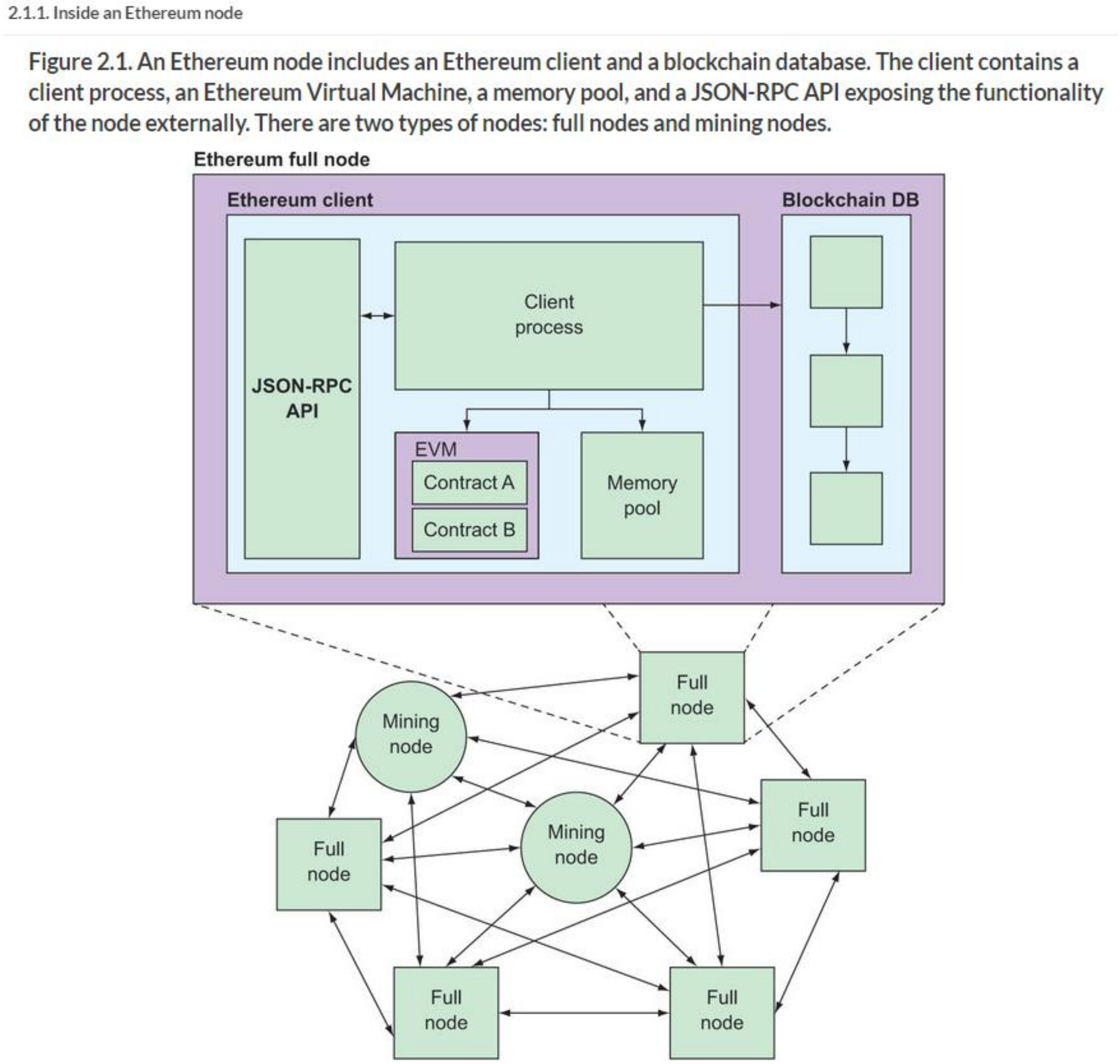
- ➢ **Hard Work & Perseverance** http://www.billslater.com/uop/persistence.htm
- ➢ **Read great Blockchain Development Resources and Authors**
- ➢ **Hands-on Practice**
- ➢ **Hanging out with Developers who are knowledgeable, kind, & sharing**
- ➢ **Participate in User Groups and Meet-ups that have excellent speakers and programs**
- ➢ **Don’t ever underestimate the difficulty and the level of effort required to become competent at this**

Topic 1: Getting Started with Blockchain Application Development - Setting up the Workbench

Workbench Decisions

- Decide WHY you are doing this.
- Get management support and a Budget (or bootstrap yourself at get the books recommended)
- Choose your Development Speed and Product Quality
  - Crawl, Walk, Run?
  - Start simple and work at the command line and/or the Remix compiler
  - Experiment, Prototype, Proof of Concept, Production?
- Ethereum Accounts and EOA accounts
- Choose your type of Key Pair & Management
- Choose your Wallet (recommend MetaMask)
- Choose your Blockchain Platform
  - Rinkeby (https://www.rinkeby.io/#stats – Test)
  - Ganache
  - Parity
  - Ethereum
  - Factom
  - NEM
  - AWS
  - Azure
  - Hyperledger
- Choose your Blockchain Type
  - Public, Permissionless
  - Public, Permissioned
  - Private, Permissionless
  - Private, Permissioned

Ethereum Full Nodes Have a Blockchain Database & the EVM



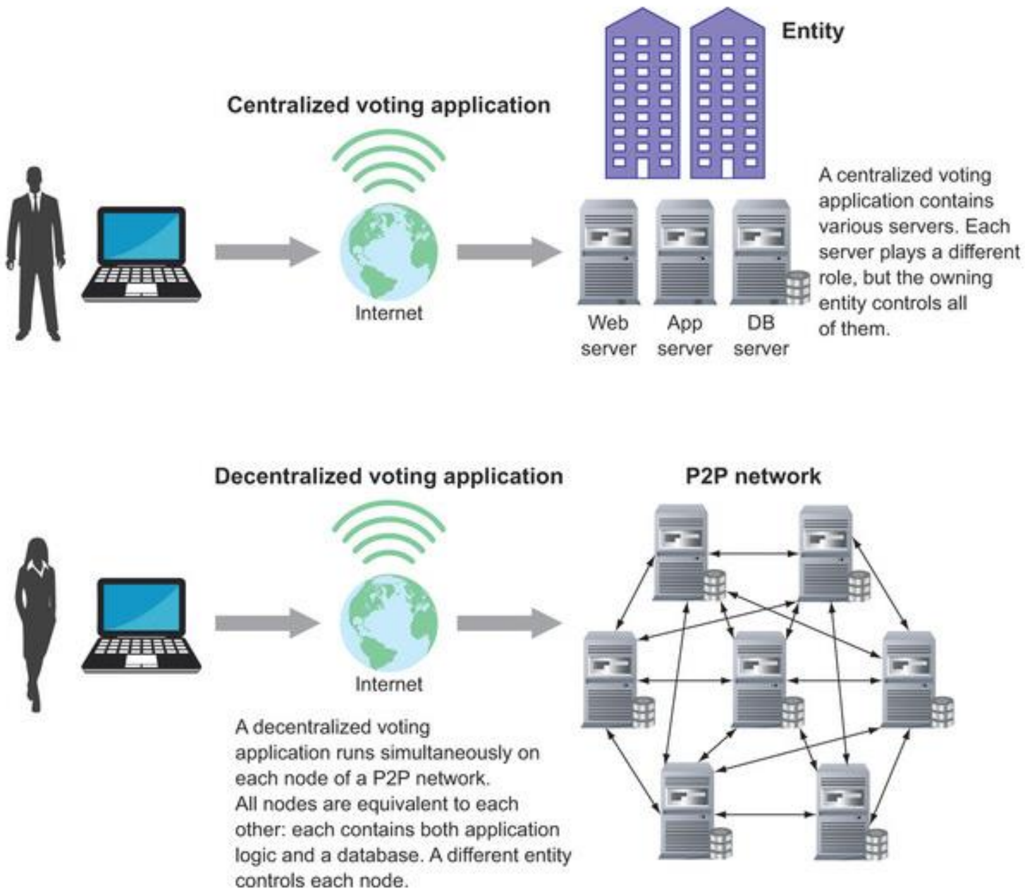
2.1.1. Inside an Ethereum node

Figure 2.1. An Ethereum node includes an Ethereum client and a blockchain database. The client contains a client process, an Ethereum Virtual Machine, a memory pool, and a JSON-RPC API exposing the functionality of the node externally. There are two types of nodes: full nodes and mining nodes.

Source: Roberto Infante, Building Ethereum DApps, 2019

Comparing a Centralized Application to a Decentralized Application

Figure 1.2. Comparison of a centralized voting application with a decentralized one. One institution owns all servers of a centralized application. A decentralized voting application runs simultaneously on multiple nodes of a network that different entities own.


Entity
Centralized voting application
Internet
Web server
App server
DB server
Decentralized voting application
P2P network
Internet

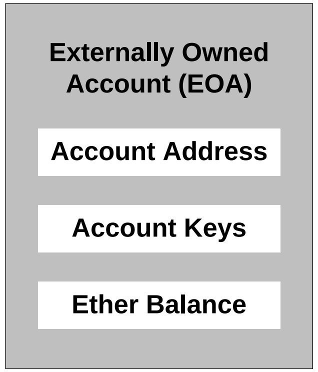
A centralized voting application contains various servers. Each server plays a different role, but the owning entity controls all of them.

A decentralized voting application runs simultaneously on each node of a P2P network. All nodes are equivalent to each other: each contains both application logic and a database. A different entity controls each node.

Source: Roberto Infante, Building Ethereum DApps, 2019

Ethereum Accounts and EOA Accounts

- You need either an Externally Owned Account (EOA) or a Contract Account to interact with a Blockchain System
- EOA
  - Accounts owned and controlled by the users. Each EOA has an Ether balance associated with it, but does not have any code associated with it. All transactions on the Ethereum network are initiated by EOAs. These accounts can send transactions to other EOSs or contract accounts
- Contract Account
  - Contract Accounts are controlled by the associated contract code which is stored with the account. Each Contract Account has an Ether balance associated with it. The contract code execution is triggered by transactions sent by an EOAs or messages sent by other contracts.
- Keypairs
  - Each EOA has a public private keypair associated with it. The account address is derived from the public key. When a new EOA is created, a JSON keyfile is created which has the public and private keys associated with the account. The private key is encrypted with the password which is provided while creating the account. For sending transactions to other accounts, the private key and the public key are required.

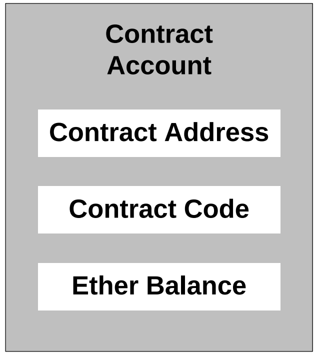


Externally Owned Account (EOA)

Account Address

Account Keys

Ether Balance



Contract Account

Contract Address

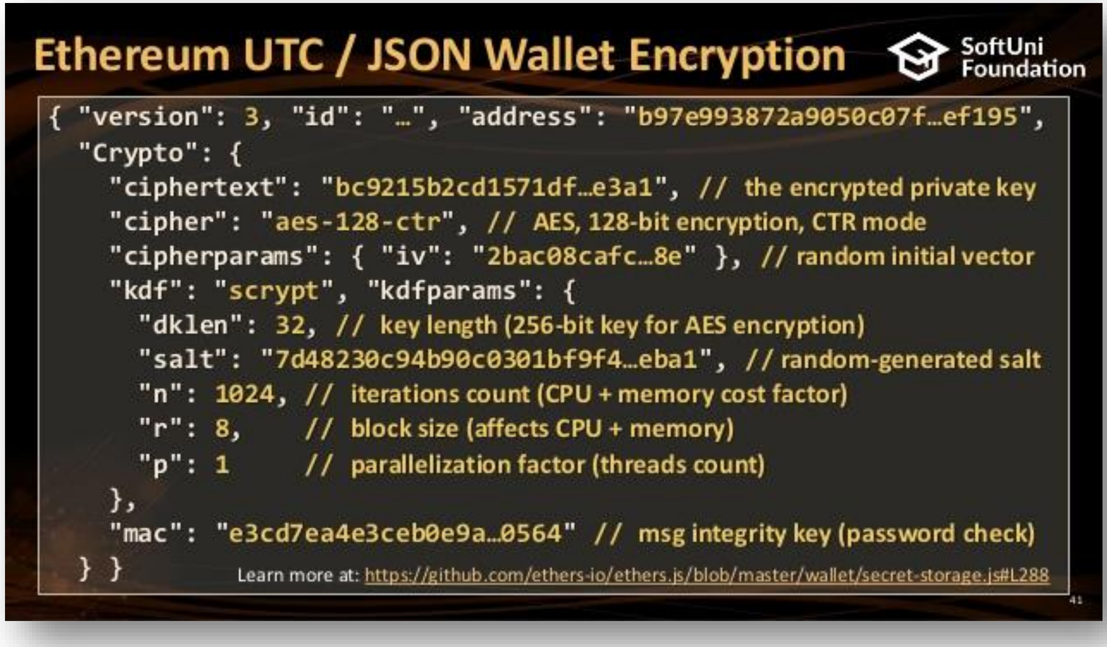
Contract Code

Ether Balance

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

Keypairs and KDF

- Keyfiles are stored in the keystore directory. To encode the private key, first a key derivation function (KDF) is used to generate a derived key. The supported KDFs include PBKDF2 and Scrypt (mentioned as *kdf* in the JSON file). The *kdfparams* field in the JSON file lists the KDF-dependent static and dynamic parameters. The account password (pw) is passed to the KDF along with the *kdfparams* (derived key = kdfeval(pw, kdfparams)). The crypto algorithm used for these keyfiles is AES-128-CTR (mentioned as cipher in JSON file. The cipher parameters (*cipherparams*) include a 128-bit initialization vector (iv) for the cipher. The key for the cipher 9enckey) is the leftmost 16 bytes of the derived key (enckey= derivedkey[;16]). The ciphertext is computed by passing the private key (*priv*), the encryption key (*enkey*) and the *cipherparams* to the encryption function of the cipher (c = aes_ctr_encrypt(priv, enckey, cipherparams)). The mac field is computed taking the SHA3 hash of the second-leftmost 16 bytesof the derived key concatenated with *ciphertext* (mac = sha3(derivedkey[:16:32] + c).
- To decode the private key from the JSON file the above steps are reversed.



Ethereum UTC / JSON Wallet Encryption

```
{ "version": 3, "id": "…", "address": "b97e993872a9050c07f…ef195",
  "Crypto": {
    "ciphertext": "bc9215b2cd1571df…e3a1", // the encrypted private key
    "cipher": "aes-128-ctr", // AES, 128-bit encryption, CTR mode
    "cipherparams": { "iv": "2bac08cafc…8e" }, // random initial vector
    "kdf": "scrypt", "kdfparams": {
      "dklen": 32, // key length (256-bit key for AES encryption)
      "salt": "7d48230c94b90c0301bf9f4…eba1", // random-generated salt
      "n": 1024, // iterations count (CPU + memory cost factor)
      "r": 8,     // block size (affects CPU + memory)
      "p": 1      // parallelization factor (threads count)
    },
    "mac": "e3cd7ea4e3ceb0e9a…0564" // msg integrity key (password check)
} }
```

Learn more at: https://github.com/ethers-io/ethers.js/blob/master/wallet/secret-storage.js#L288

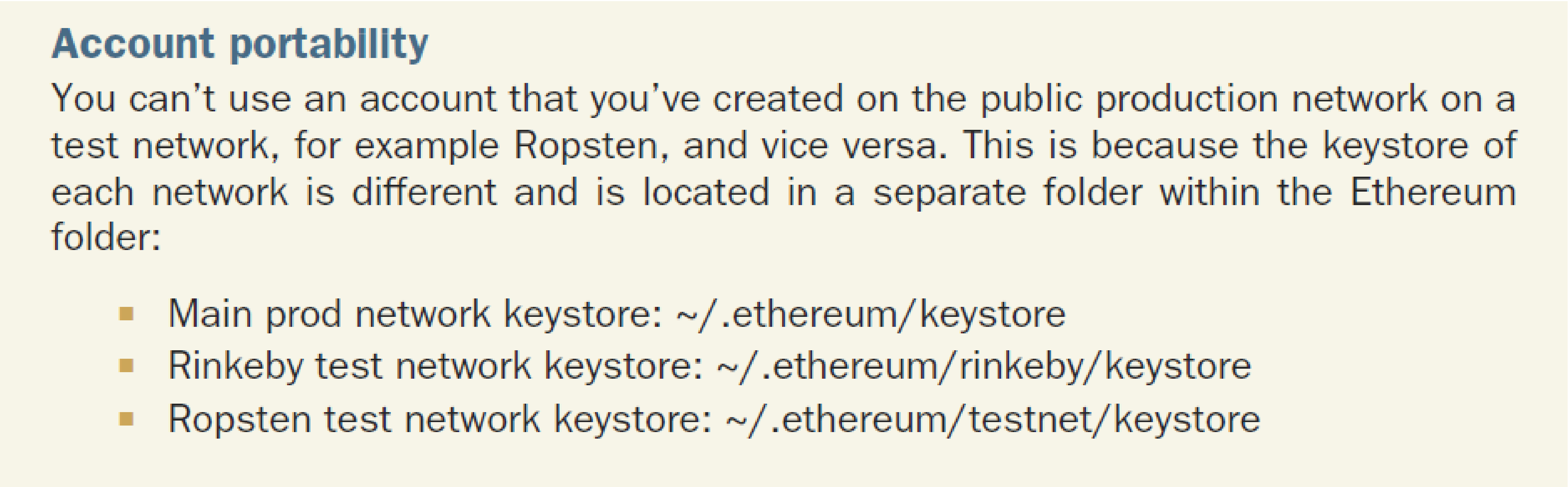
- If this hurts your brain to think about, just know that it is the basis for securely authenticating with and signing blockchain transactions using your public and private keys. It is a fundamental part of the cryptographic security foundation of blockchain, and it happens automatically behind the scenes, once you have set up your wallet with your public key, private key and password.

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti AND SoftUni Foundation

Blockchain Workshop Student Public & Private Key Pairs

Name	Public Key	Private Key
Student 01	0xdF4f021Ec048b84431252604c8Cca48d74bb5068	754a147fcacf6c3b5b9a77e423435a37a4c4dd5e0d4b76004c676f2bce2efc27
Student 02	0x03088FDaee150D6635b8fF0e646138e444f3c9D3	d4d85585f29108d648b86583624f304697fd7cca43ee2d0032ffa5b0b7dec284
Student 03	0x83654fb6D9ea75fFfE78234c367dCd080E4e3078	6660a87c16cc8bd3c0744c928c332f7a39ea0ee79efccca8009c0671618f6a40
Student 04	0xEf962e3f5233b43192AF41177576516e2aBbf8CC	bb04eba43710b39d576dbe227461df65eb6b1de8aac2338bcc59d50e7aedd2d3
Student 05	0xf84D6e63A2DaBFEE1c216E0514CC72d86436A9F6	6c373921fd6208eb6cd1ffdc56209008b2542c257e12a1453b69f308a6628b73
Student 06	0x96fC53dee2439ba514CEa7C7152675695A392044	7c801705d8840a9713190b48d9c2705f06a3fdbbeeda4a0a43775b227df3e5bc
Student 07	0x564e7AD25bEb06EbAF81D77f9F5DC290aFBd3474	e0484046c0e90ffbbf8004d452aeb53378f8275ab0b0a64278166a2bfd98b891
Student 08	0x30291Ad653C031094093C61C13042eb669595498	1c05967a3c47ee7688ac49157c862abe2158b812dea5e900ec248d04f1e0452c
Student 09	0x1C2bCf127A6d2604705863CEAf6907b895aBC2b6	c81865c9dea5796c182b953b2b00d5c202d95eccefa7c50b0cc89cae4cfb9c18
Student 10	0xe4320AE19d2b47e6e4467252f1eFA0880Fd60DED	bcdc174b5a9b6cb61a7bce7e41232c6110508f196725637c3edc1901eea7c943
Student 11	0xBc464518fa18ff56B271d2eF17184590Ddde382E	d9f8fbedc7b5b0efeb58ad45502ebe48cac5d2765d97d7c1118952724c30e95a
Student 11	0x64137db23217972e692239960a94aEA44B18d3a4	f55ed6ed125f89c466ce83244395ad9a1fb792a5af1f833c9e068210b8f9250f
Student 12	0xaC130fa9E334C9A72e59AceF59aCe53d03A1dce4	63a9e54c43646de10cbda338b4fd608e2afb7d651f014ac6d8ff67601d808a50
Student 13	0xBACfa76ae12d64C5BcD57548386E0488190C23dd	defb5992635bd222da23d7d88c716ab6e9d629e7e1206197f48009f01728c4ec
Student 14	0xa724a1B81B9eD2EcE7Fc2206F54129aa6F8C65C8	dcc4f9671fa868928a700f0cbc6479445e838c2dab89f68c15d9b746274c5ff4
Student 14	0x254503Df4492Bc6e57365bD078672741be1972BD	a780dd1c91cccfcb03fd788f07900df56707f8946a916b55e49731122ad20d72
Student 15	0xb6Fb168790AF529CF2073d18779aFF4a96d4BDa2	56b089efa9363bd95a3a537f295b24bb183e3d615268cd7516700a222e758fb1
Student 16	0xDf579E2577e46d32f50afa55AB06bcBdf1ff8A2d	df20fca7a4a73f33a35799268334eb2d9ce9cbef2a2aa4b6b8233e86142921f2
Student 17	0x7793A496b792942116734FC9996fbeE2eD455e22	4680993b4d586098803e9285ce5dcb167f06c1ddfa1a33d2522ee6225d4c4084
Student 18	0xf199C3e47ECb13D1267904cF8A3A063b437cFe76	fa7b1f6c081895b644ccb788a69187812ab7f6b98d0438ca0c2c8ab740da12f1
Student 19	0x948B5F7A88eB06a1AecabE009F55475c3943729E	7df890271e9d8b29dc6ee7eb8403635139a310dfa8dd7d44fb5aa824ec9988eb
Student 20	0x8198DF0ec9D63b1aF545981Eb26AB29Ba84f2513	04ab78ddba34d0c122d52b7cec73b9f3c7b6b4e972cf581fb3431ca39d673927

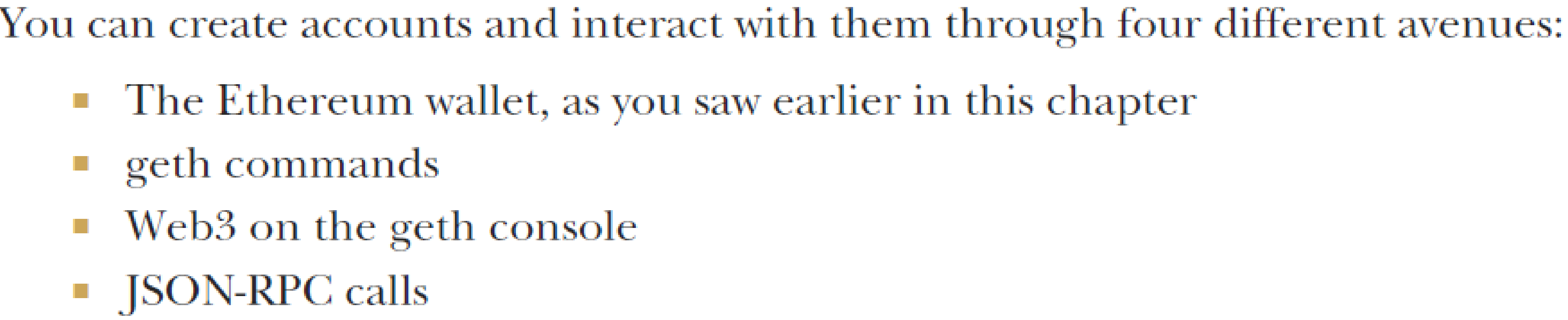
Other Ways to Get Ethereum Login Credentials



Account portability

You can't use an account that you've created on the public production network on a test network, for example Ropsten, and vice versa. This is because the keystore of each network is different and is located in a separate folder within the Ethereum folder:

- Main prod network keystore: ~/.ethereum/keystore
- Rinkeby test network keystore: ~/.ethereum/rinkeby/keystore
- Ropsten test network keystore: ~/.ethereum/testnet/keystore

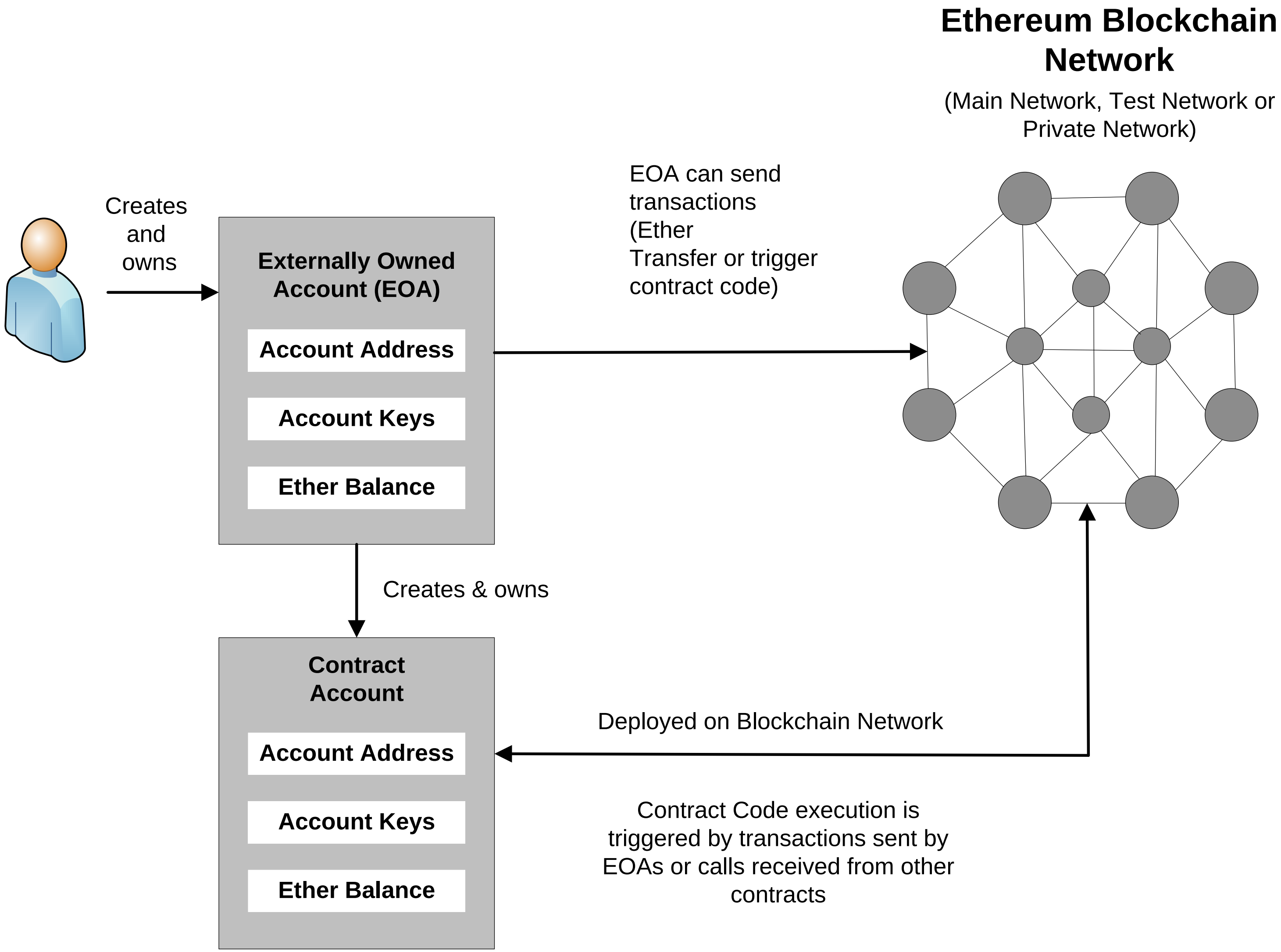


You can create accounts and interact with them through four different avenues:

- The Ethereum wallet, as you saw earlier in this chapter
- geth commands
- Web3 on the geth console
- JSON-RPC calls

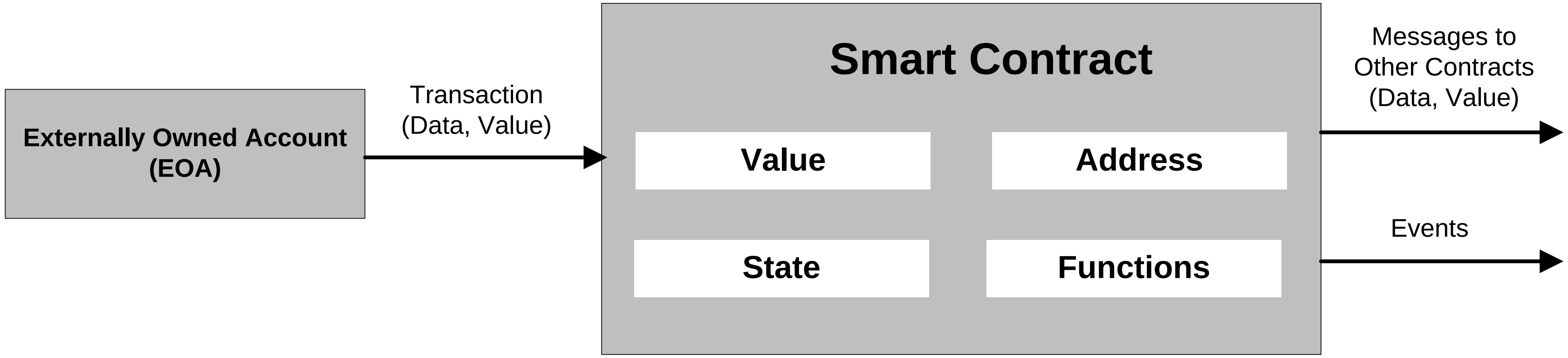
Source: Roberto Infante, Building Ethereum DApps, 2019

EOA Accounts


Ethereum Blockchain Network
(Main Network, Test Network or Private Network)
Creates and owns
Externally Owned Account (EOA)
Account Address
Account Keys
Ether Balance
EOA can send transactions (Ether Transfer or trigger contract code)
Creates & owns
Contract Account
Account Address
Account Keys
Ether Balance
Deployed on Blockchain Network
Contract Code execution is triggered by transactions sent by EOAs or calls received from other contracts

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

EOA and Smart Contract Execution

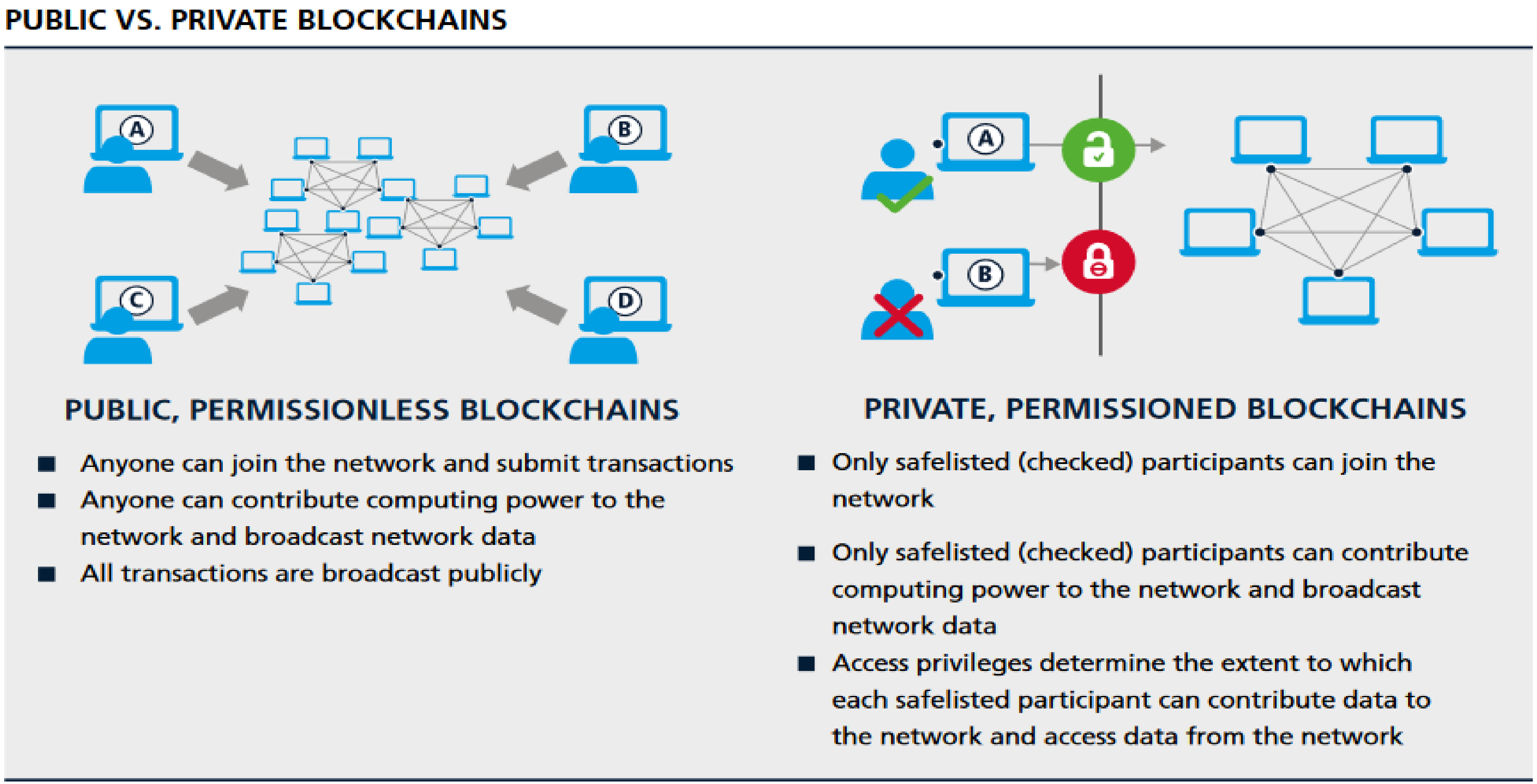


Externally Owned Account (EOA)
Transaction (Data, Value)
Smart Contract
Value
Address
State
Functions
Messages to Other Contracts (Data, Value)
Events

Smart Contract Execution

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

Public vs. Private



PUBLIC VS. PRIVATE BLOCKCHAINS

PUBLIC, PERMISSIONLESS BLOCKCHAINS

- Anyone can join the network and submit transactions
- Anyone can contribute computing power to the network and broadcast network data
- All transactions are broadcast publicly

PRIVATE, PERMISSIONED BLOCKCHAINS

- Only safelisted (checked) participants can join the network
- Only safelisted (checked) participants can contribute computing power to the network and broadcast network data
- Access privileges determine the extent to which each safelisted participant can contribute data to the network and access data from the network

Key differences between public, permissionless blockchains and private, permissioned blockchains; **Source:** Accenture

Important Blockchain Architecture Decision

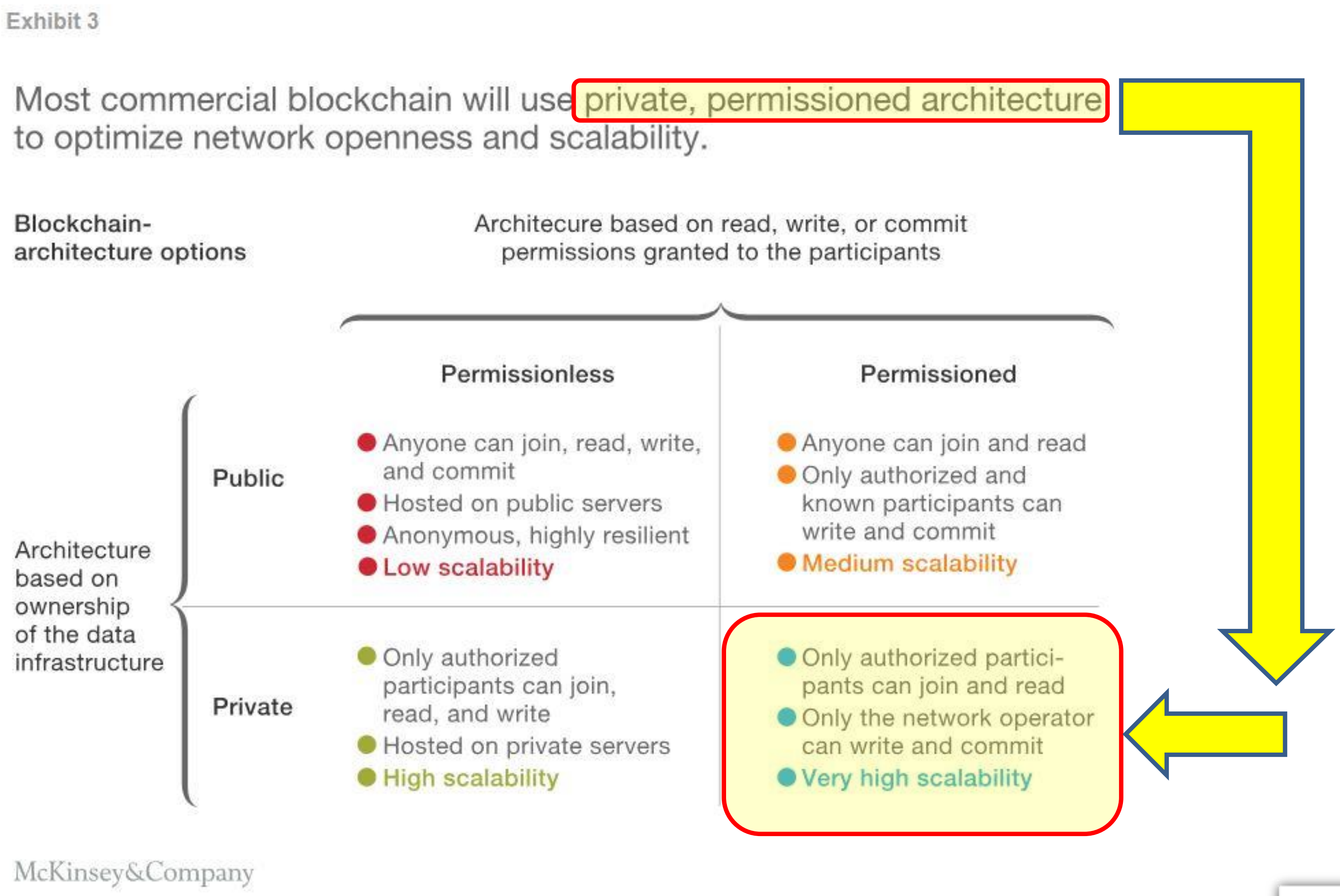


Exhibit 3

Most commercial blockchain will use private, permissioned architecture to optimize network openness and scalability.

Blockchain-architecture options

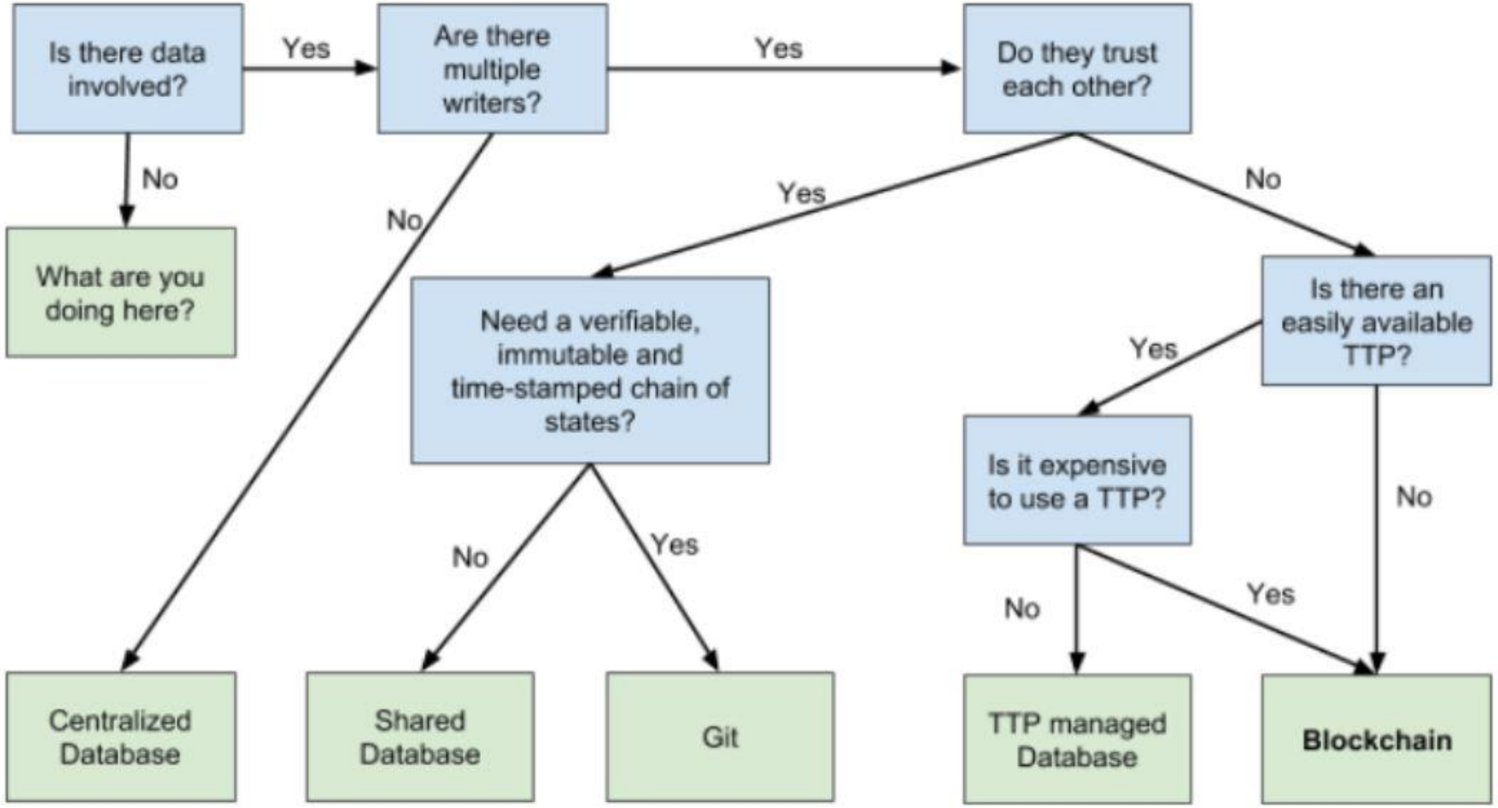
Architecure based on read, write, or commit permissions granted to the participants

Architecture based on ownership of the data infrastructure		Permissionless	Permissioned
	Public	● Anyone can join, read, write, and commit ● Hosted on public servers ● Anonymous, highly resilient ● Low scalability	● Anyone can join and read ● Only authorized and known participants can write and commit ● Medium scalability
	Private	● Only authorized participants can join, read, and write ● Hosted on private servers ● High scalability	● Only authorized participants can join and read ● Only the network operator can write and commit ● Very high scalability

McKinsey&Company

To Blockchain or Not to Blockchain

If you are a little lost, don't worry, here is a visual framework that will help you assess whether a Blockchain is something you should be looking into:



Is there data involved?
Yes → Are there multiple writers?
No → What are you doing here?

Are there multiple writers?
Yes → Do they trust each other?
No → Centralized Database

Do they trust each other?
Yes → Need a verifiable, immutable and time-stamped chain of states?
No → Is there an easily available TTP?

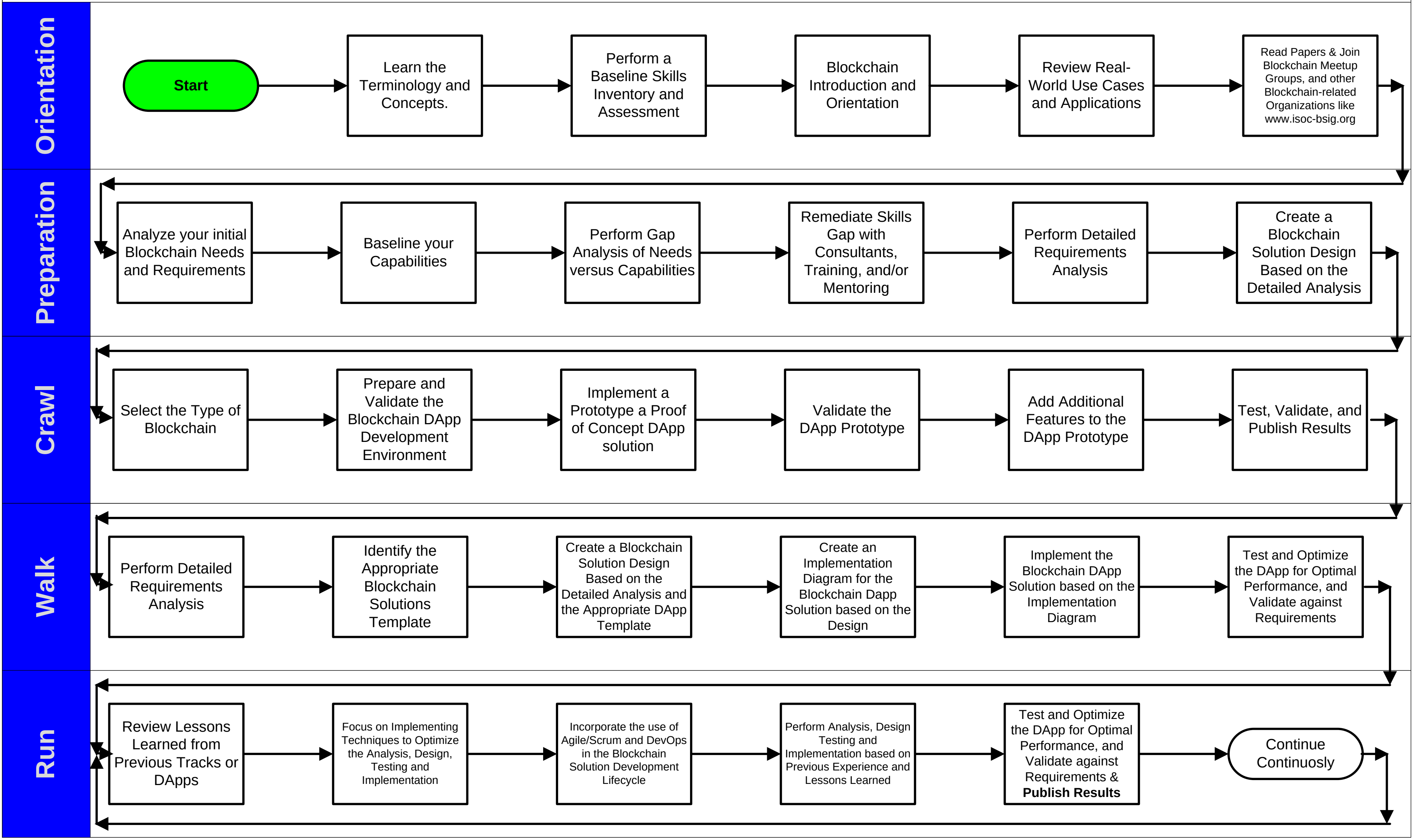
Need a verifiable, immutable and time-stamped chain of states?
No → Shared Database
Yes → Git

Is there an easily available TTP?
Yes → Is it expensive to use a TTP?
No → Blockchain

Is it expensive to use a TTP?
No → TTP managed Database
Yes → Blockchain

Source: To Blockchain or not to Blockchain? https://medium.com/causys/to-blockchain-or-not-to-blockchain-aed05bf08150 Hats off to the author, Thomas Ferry of Causys

Roadmap to "Blockchain" Your IT Organization: How to Help Your IT Staff Go from Square One to Competence & Dominance in Blockchain Technologies



Orientation

Start

Learn the Terminology and Concepts.

Perform a Baseline Skills Inventory and Assessment

Blockchain Introduction and Orientation

Review Real-World Use Cases and Applications

Read Papers & Join Blockchain Meetup Groups, and other Blockchain-related Organizations like www.isoc-bsig.org

Preparation

Analyze your initial Blockchain Needs and Requirements

Baseline your Capabilities

Perform Gap Analysis of Needs versus Capabilities

Remediate Skills Gap with Consultants, Training, and/or Mentoring

Perform Detailed Requirements Analysis

Create a Blockchain Solution Design Based on the Detailed Analysis

Crawl

Select the Type of Blockchain

Prepare and Validate the Blockchain DApp Development Environment

Implement a Prototype a Proof of Concept DApp solution

Validate the DApp Prototype

Add Additional Features to the DApp Prototype

Test, Validate, and Publish Results

Walk

Perform Detailed Requirements Analysis

Identify the Appropriate Blockchain Solutions Template

Create a Blockchain Solution Design Based on the Detailed Analysis and the Appropriate DApp Template

Create an Implementation Diagram for the Blockchain Dapp Solution based on the Design

Implement the Blockchain DApp Solution based on the Implementation Diagram

Test and Optimize the DApp for Optimal Performance, and Validate against Requirements

Run

Review Lessons Learned from Previous Tracks or DApps

Focus on Implementing Techniques to Optimize the Analysis, Design, Testing and Implementation

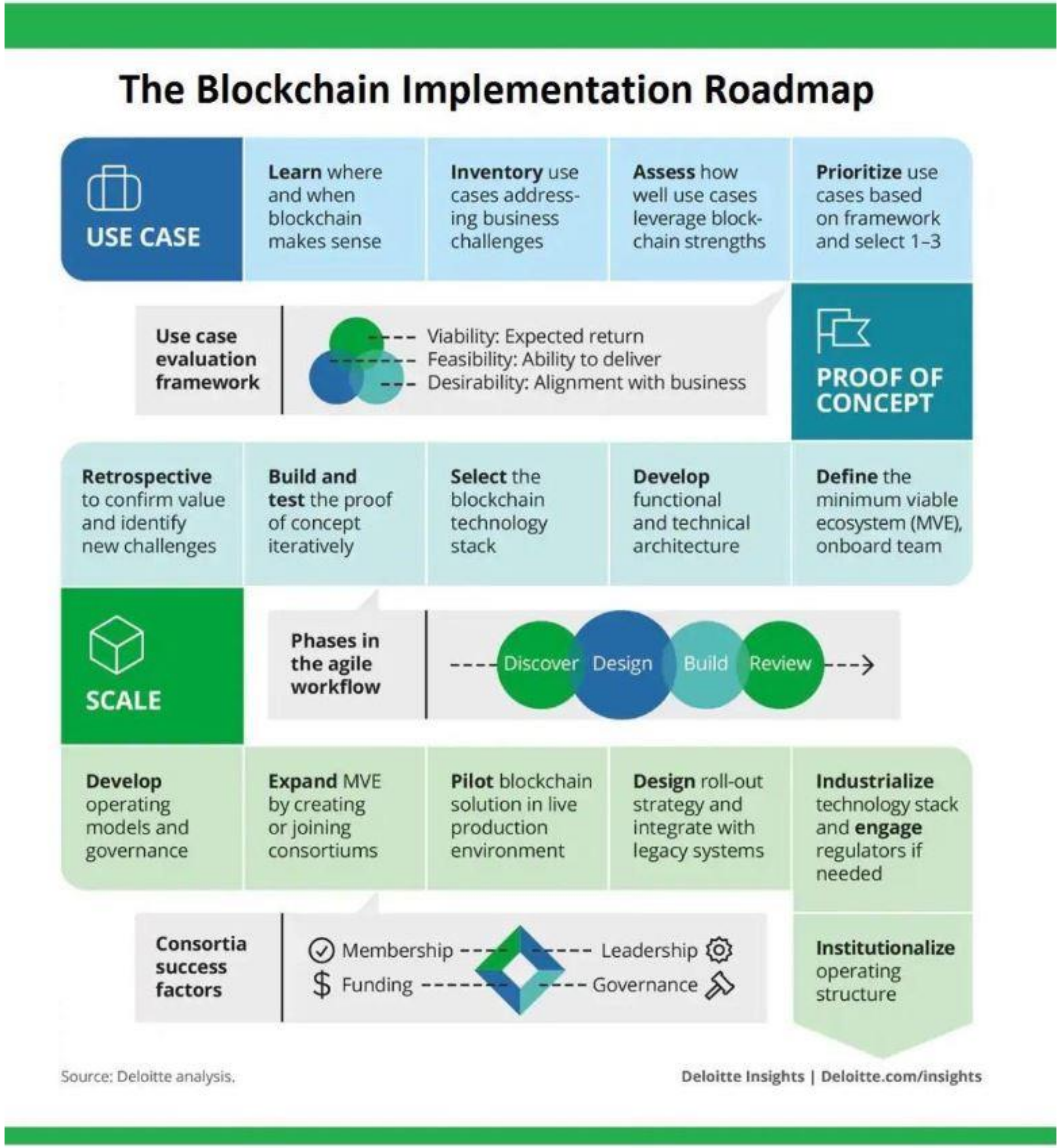
Incorporate the use of Agile/Scrum and DevOps in the Blockchain Solution Development Lifecycle

Perform Analysis, Design Testing and Implementation based on Previous Experience and Lessons Learned

Test and Optimize the DApp for Optimal Performance, and Validate against Requirements & **Publish Results**

Continue Continuosly

Blockchain Implementation Roadmap



The Blockchain Implementation Roadmap

USE CASE

Learn where and when blockchain makes sense

Inventory use cases addressing business challenges

Assess how well use cases leverage blockchain strengths

Prioritize use cases based on framework and select 1–3

Use case evaluation framework

- Viability: Expected return
- Feasibility: Ability to deliver
- Desirability: Alignment with business

PROOF OF CONCEPT

Define the minimum viable ecosystem (MVE), onboard team

Develop functional and technical architecture

Select the blockchain technology stack

Build and test the proof of concept iteratively

Retrospective to confirm value and identify new challenges

SCALE

Phases in the agile workflow

Discover → Design → Build → Review

Develop operating models and governance

Expand MVE by creating or joining consortiums

Pilot blockchain solution in live production environment

Design roll-out strategy and integrate with legacy systems

Industrialize technology stack and **engage** regulators if needed

Consortia success factors

- Membership
- Funding
- Leadership
- Governance

Institutionalize operating structure

Source: Deloitte analysis.

Deloitte Insights | Deloitte.com/insights

Ethereum EVM

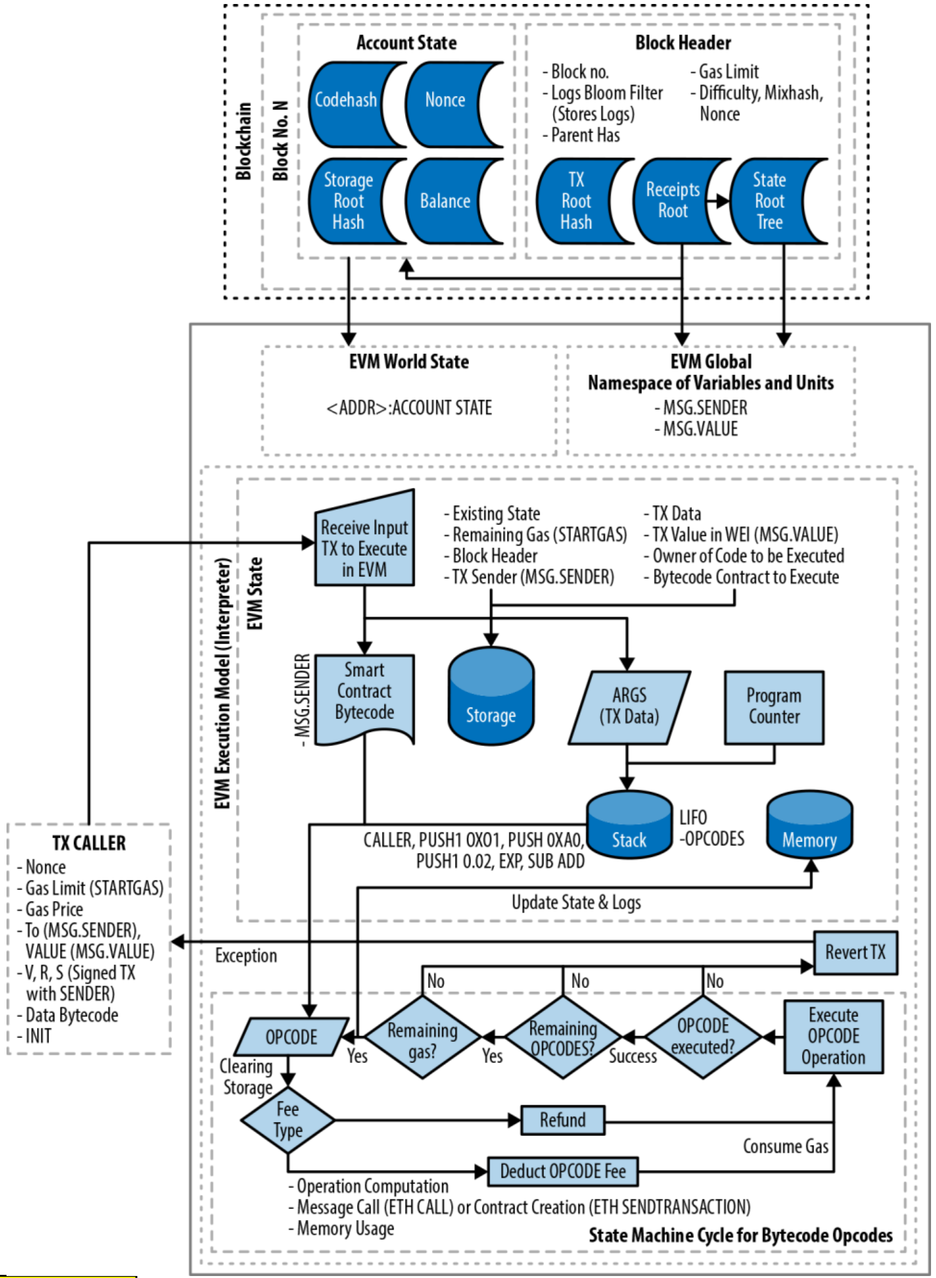
This is the Ethereum Virtual Machine.

The EVM is also known as "The World Computer"

The EVM executes compiled Smart Contract code.

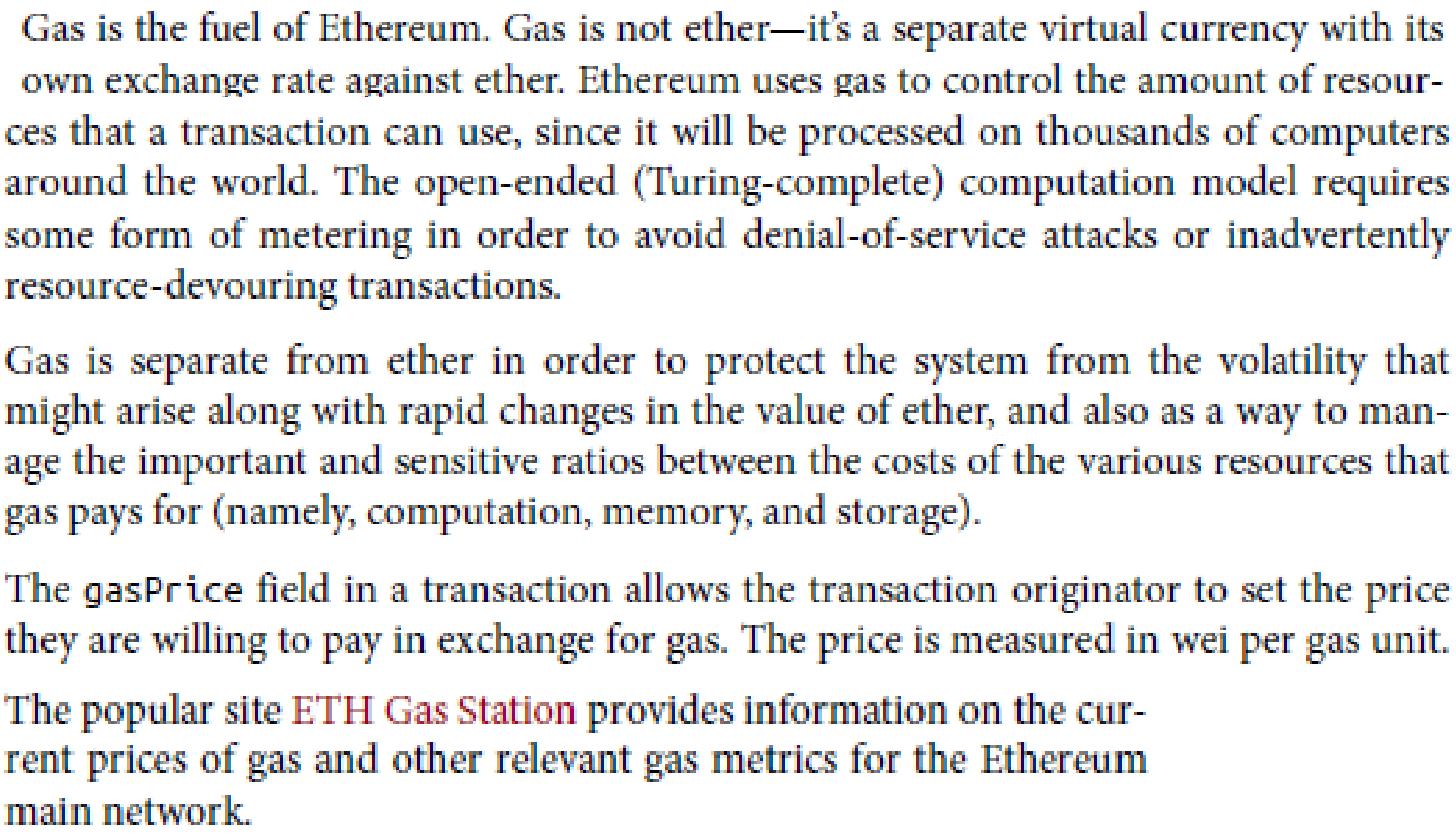
Smart Contracts:

1) **Require "Gas"**
2) **Become permanent on the Ethereum Blockchain**



Source: Mastering Ethereum by Andreas Antonopolous & Gavin Wood

Gas



Gas is the fuel of Ethereum. Gas is not ether—it's a separate virtual currency with its own exchange rate against ether. Ethereum uses gas to control the amount of resources that a transaction can use, since it will be processed on thousands of computers around the world. The open-ended (Turing-complete) computation model requires some form of metering in order to avoid denial-of-service attacks or inadvertently resource-devouring transactions.

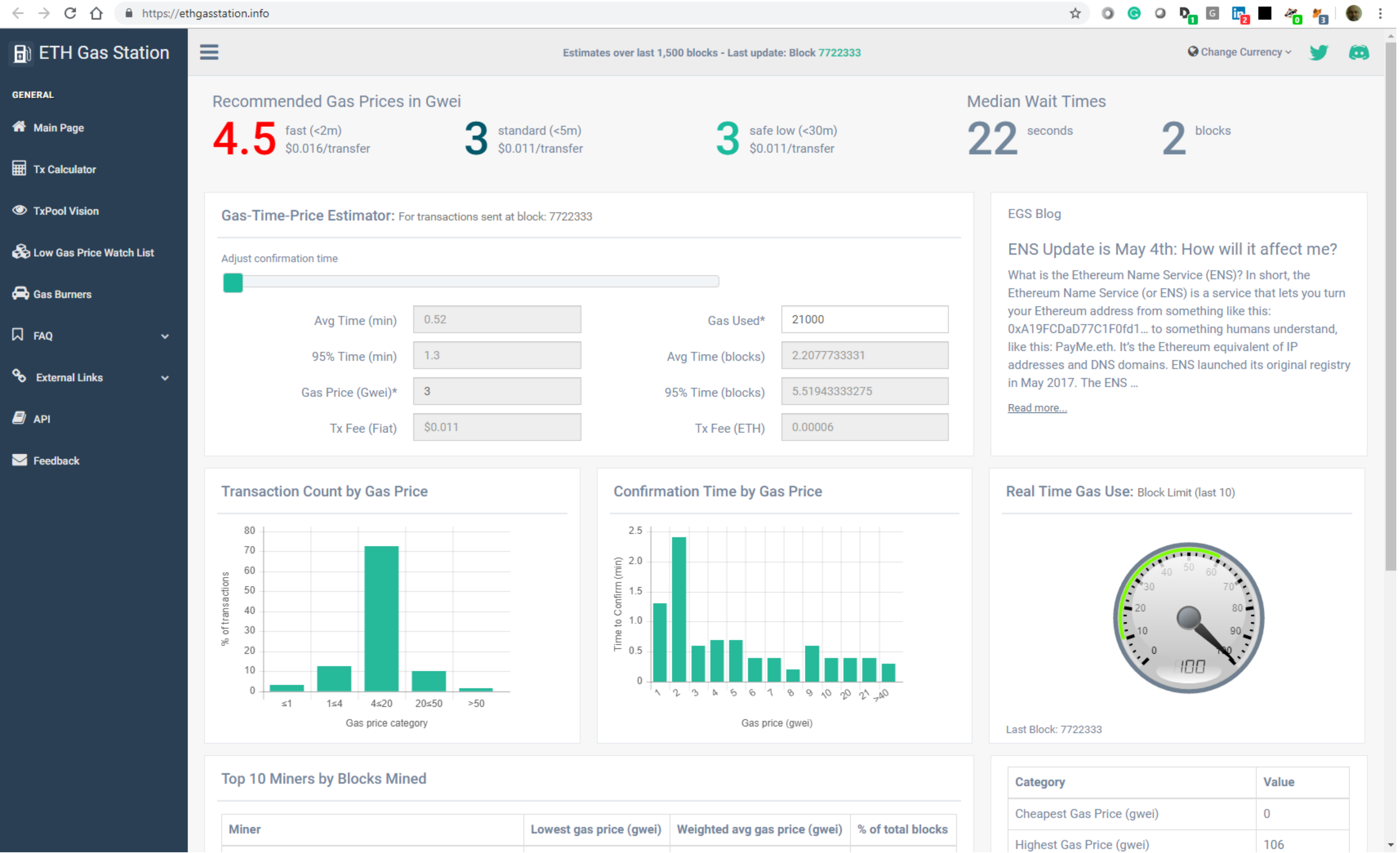
Gas is separate from ether in order to protect the system from the volatility that might arise along with rapid changes in the value of ether, and also as a way to manage the important and sensitive ratios between the costs of the various resources that gas pays for (namely, computation, memory, and storage).

The `gasPrice` field in a transaction allows the transaction originator to set the price they are willing to pay in exchange for gas. The price is measured in wei per gas unit.

The popular site ETH Gas Station provides information on the current prices of gas and other relevant gas metrics for the Ethereum main network.

Source: Mastering Ethereum by Andreas Antonopoulos & Gavin Wood

Gas



https://ethgasstation.info

ETH Gas Station

GENERAL

- Main Page
- Tx Calculator
- TxPool Vision
- Low Gas Price Watch List
- Gas Burners
- FAQ
- External Links
- API
- Feedback

Estimates over last 1,500 blocks - Last update: Block 7722333

Change Currency

Recommended Gas Prices in Gwei

4.5 fast (<2m) $0.016/transfer

3 standard (<5m) $0.011/transfer

3 safe low (<30m) $0.011/transfer

Median Wait Times

22 seconds

2 blocks

Gas-Time-Price Estimator: For transactions sent at block: 7722333

Adjust confirmation time

Avg Time (min)	0.52	Gas Used*	21000
95% Time (min)	1.3	Avg Time (blocks)	2.2077733331
Gas Price (Gwei)*	3	95% Time (blocks)	5.51943333275
Tx Fee (Fiat)	$0.011	Tx Fee (ETH)	0.00006

EGS Blog

ENS Update is May 4th: How will it affect me?

What is the Ethereum Name Service (ENS)? In short, the Ethereum Name Service (or ENS) is a service that lets you turn your Ethereum address from something like this: 0xA19FCDaD77C1F0fd1... to something humans understand, like this: PayMe.eth. It's the Ethereum equivalent of IP addresses and DNS domains. ENS launched its original registry in May 2017. The ENS ...

Read more...

Transaction Count by Gas Price

Confirmation Time by Gas Price

Real Time Gas Use: Block Limit (last 10)

Last Block: 7722333

Top 10 Miners by Blocks Mined

Miner	Lowest gas price (gwei)	Weighted avg gas price (gwei)	% of total blocks

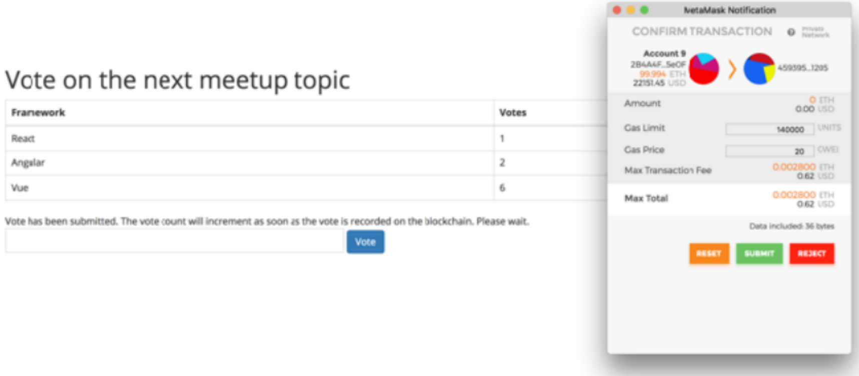
Category	Value
Cheapest Gas Price (gwei)	0
Highest Gas Price (gwei)	106

ETH Gas Station

Source: https://ethgasstation.info/

ETH + Gas

- “It costs money to interact with the blockchain. This money goes to miners who do all the work to include your code in the blockchain.”



Vote on the next meetup topic

Framework	Votes
React	1
Angalar	2
Vue	6

Vote has been submitted. The vote count will increment as soon as the vote is recorded on the blockchain. Please wait.

MetaMask Notification

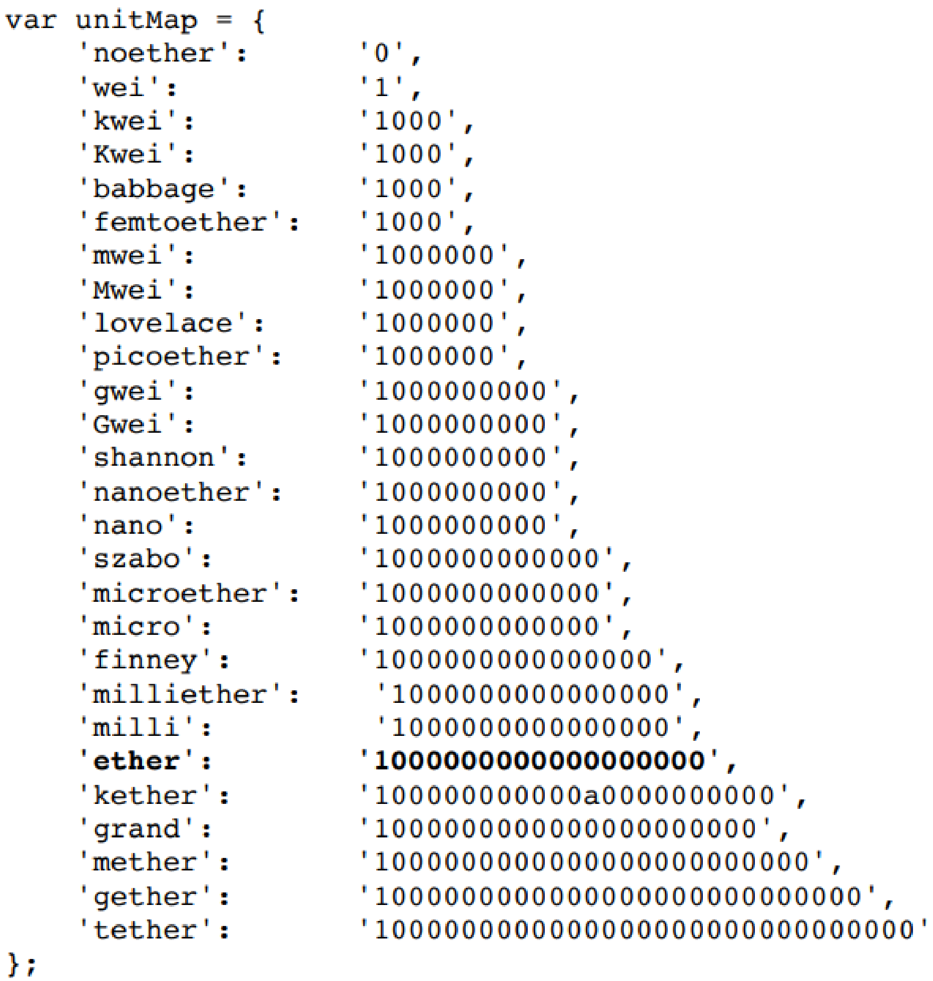
CONFIRM TRANSACTION

Account 9
2B4A4F...5e0F
99.994 ETH
22151.45 USD

459395...1205

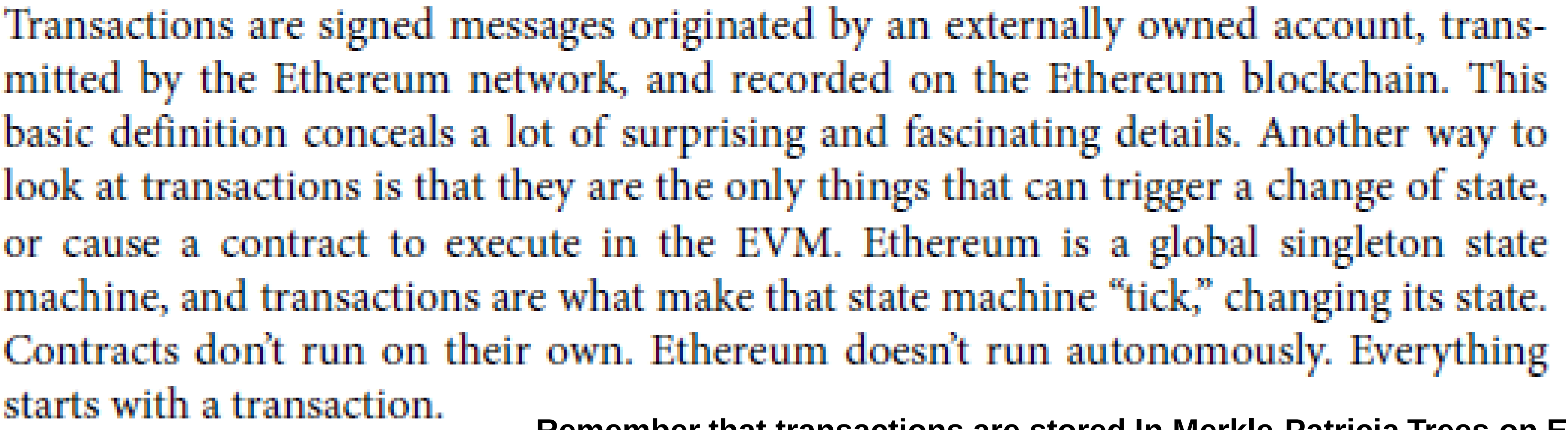
Amount: 0 ETH 0.00 USD
Gas Limit: 140000 UNITS
Gas Price: 20 GWEI
Max Transaction Fee: 0.002800 ETH 0.62 USD
Max Total: 0.002800 ETH 0.62 USD
Data included: 36 bytes

1 Ether = 10 ^{18} wei



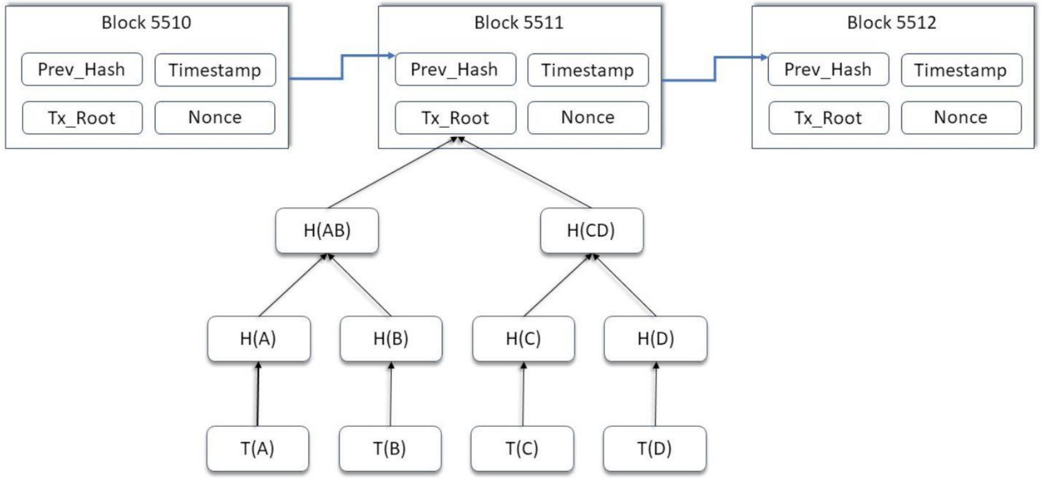
```
var unitMap = {
    'noether':       '0',
    'wei':           '1',
    'kwei':          '1000',
    'Kwei':          '1000',
    'babbage':       '1000',
    'femtoether':    '1000',
    'mwei':          '1000000',
    'Mwei':          '1000000',
    'lovelace':      '1000000',
    'picoether':     '1000000',
    'gwei':          '1000000000',
    'Gwei':          '1000000000',
    'shannon':       '1000000000',
    'nanoether':     '1000000000',
    'nano':          '1000000000',
    'szabo':         '1000000000000',
    'microether':    '1000000000000',
    'micro':         '1000000000000',
    'finney':        '1000000000000000',
    'milliether':     '1000000000000000',
    'milli':          '1000000000000000',
    'ether':         '1000000000000000000',
    'kether':        '100000000000a0000000000',
    'grand':         '1000000000000000000000',
    'mether':        '1000000000000000000000000',
    'gether':        '1000000000000000000000000000',
    'tether':        '1000000000000000000000000000000'
};
```

What Is a Transaction?



Transactions are signed messages originated by an externally owned account, transmitted by the Ethereum network, and recorded on the Ethereum blockchain. This basic definition conceals a lot of surprising and fascinating details. Another way to look at transactions is that they are the only things that can trigger a change of state, or cause a contract to execute in the EVM. Ethereum is a global singleton state machine, and transactions are what make that state machine "tick," changing its state. Contracts don't run on their own. Ethereum doesn't run autonomously. Everything starts with a transaction.

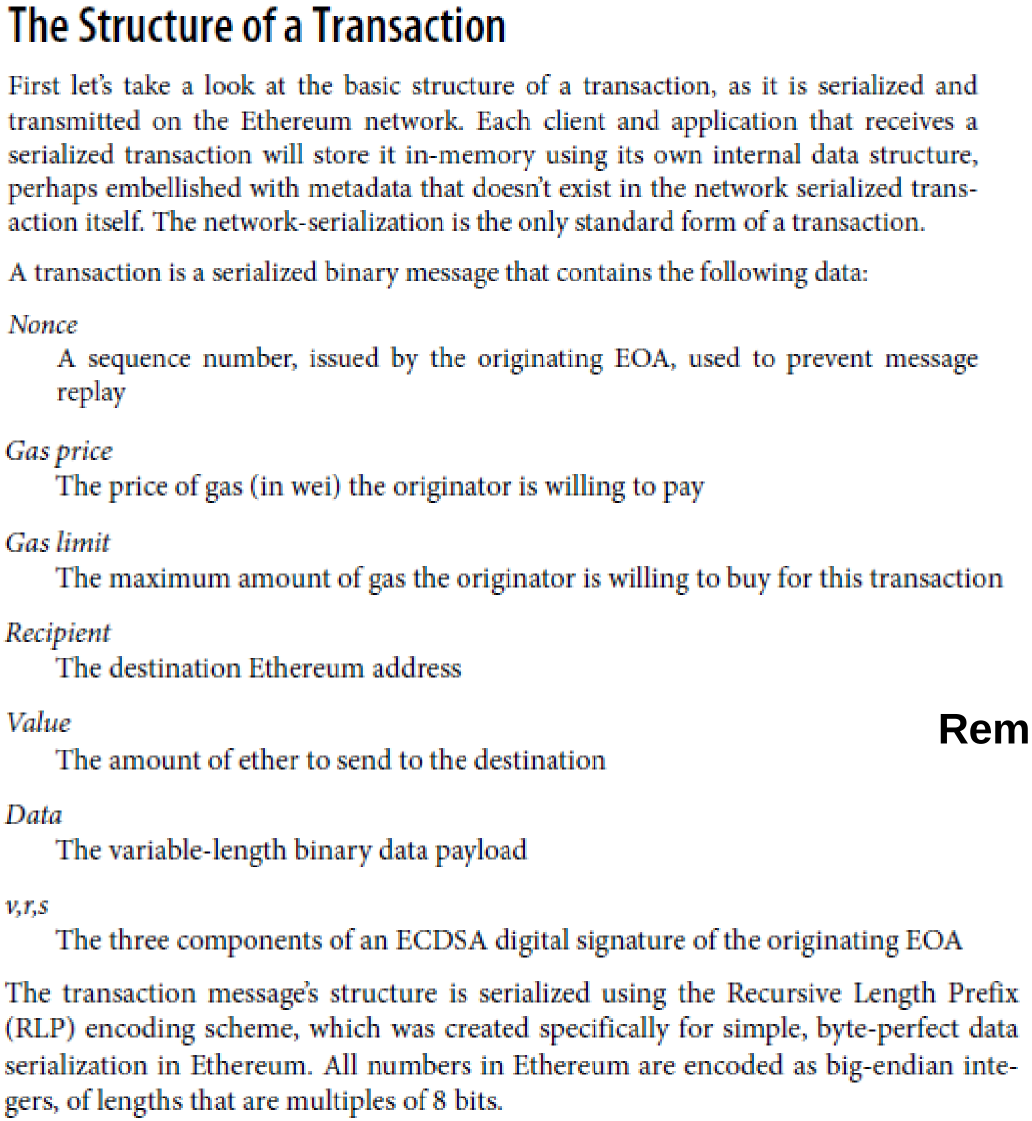
Remember that transactions are stored In Merkle-Patricia Trees on Ethereum Blocks



blockchain data structure, by sombando, shared under a Creative Commons (BY-SA) license

Source: Mastering Ethereum by Andreas Antonopoulos & Gavin Wood

What Is the Structure of a Transaction?



The Structure of a Transaction

First let's take a look at the basic structure of a transaction, as it is serialized and transmitted on the Ethereum network. Each client and application that receives a serialized transaction will store it in-memory using its own internal data structure, perhaps embellished with metadata that doesn't exist in the network serialized transaction itself. The network-serialization is the only standard form of a transaction.

A transaction is a serialized binary message that contains the following data:

Nonce
: A sequence number, issued by the originating EOA, used to prevent message replay

Gas price
: The price of gas (in wei) the originator is willing to pay

Gas limit
: The maximum amount of gas the originator is willing to buy for this transaction

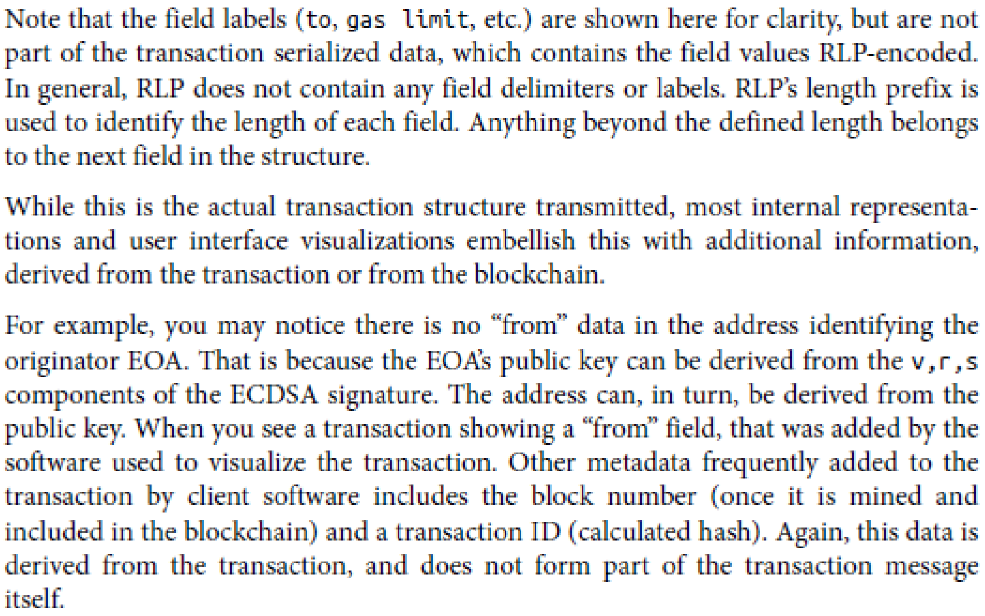
Recipient
: The destination Ethereum address

Value
: The amount of ether to send to the destination

Data
: The variable-length binary data payload

v,r,s
: The three components of an ECDSA digital signature of the originating EOA

The transaction message's structure is serialized using the Recursive Length Prefix (RLP) encoding scheme, which was created specifically for simple, byte-perfect data serialization in Ethereum. All numbers in Ethereum are encoded as big-endian integers, of lengths that are multiples of 8 bits.

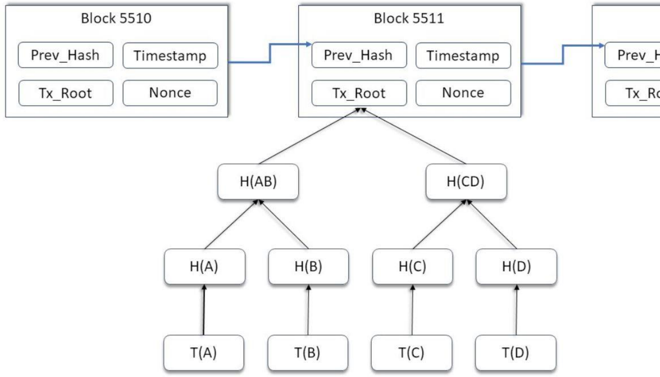


Note that the field labels (`to`, `gas limit`, etc.) are shown here for clarity, but are not part of the transaction serialized data, which contains the field values RLP-encoded. In general, RLP does not contain any field delimiters or labels. RLP's length prefix is used to identify the length of each field. Anything beyond the defined length belongs to the next field in the structure.

While this is the actual transaction structure transmitted, most internal representations and user interface visualizations embellish this with additional information, derived from the transaction or from the blockchain.

For example, you may notice there is no "from" data in the address identifying the originator EOA. That is because the EOA's public key can be derived from the `v,r,s` components of the ECDSA signature. The address can, in turn, be derived from the public key. When you see a transaction showing a "from" field, that was added by the software used to visualize the transaction. Other metadata frequently added to the transaction by client software includes the block number (once it is mined and included in the blockchain) and a transaction ID (calculated hash). Again, this data is derived from the transaction, and does not form part of the transaction message itself.

Remember that transactions are stored In Merkle-Patricia Trees



blockchain data structure, by sombando, shared under a Creative Commons (BY-SA) license

Source: Mastering Ethereum by Andreas Antonopoulos & Gavin Wood

What Is a DApp?

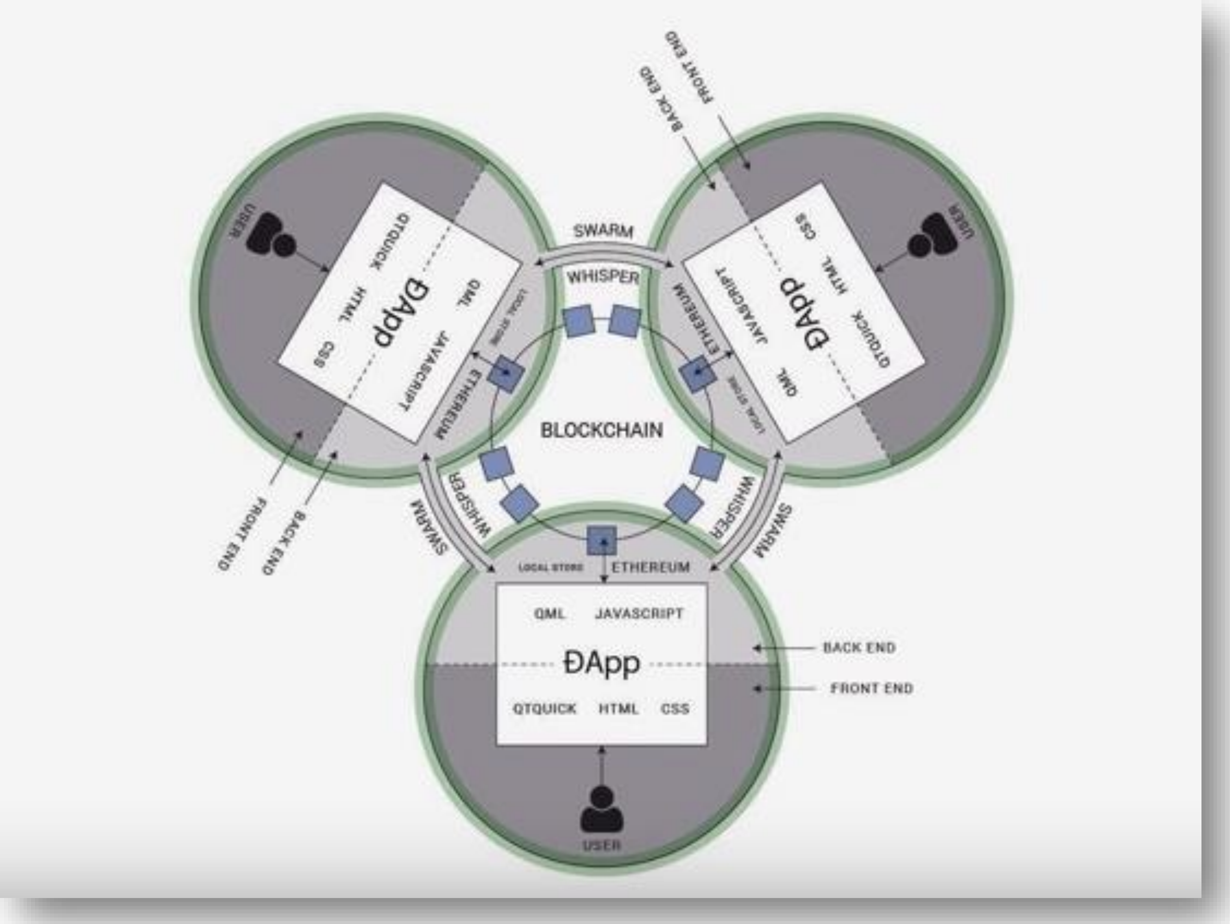
DApp is an abbreviated form for decentralized application.

A DApp has its backend code running on a decentralized peer-to-peer network. Contrast this with an app where the backend code is running on centralized servers.

A DApp can have frontend code and user interfaces written in any language (just like an app) that can make calls to its backend. Furthermore, its frontend can be hosted on decentralized storage such as Swarm or IPFS.

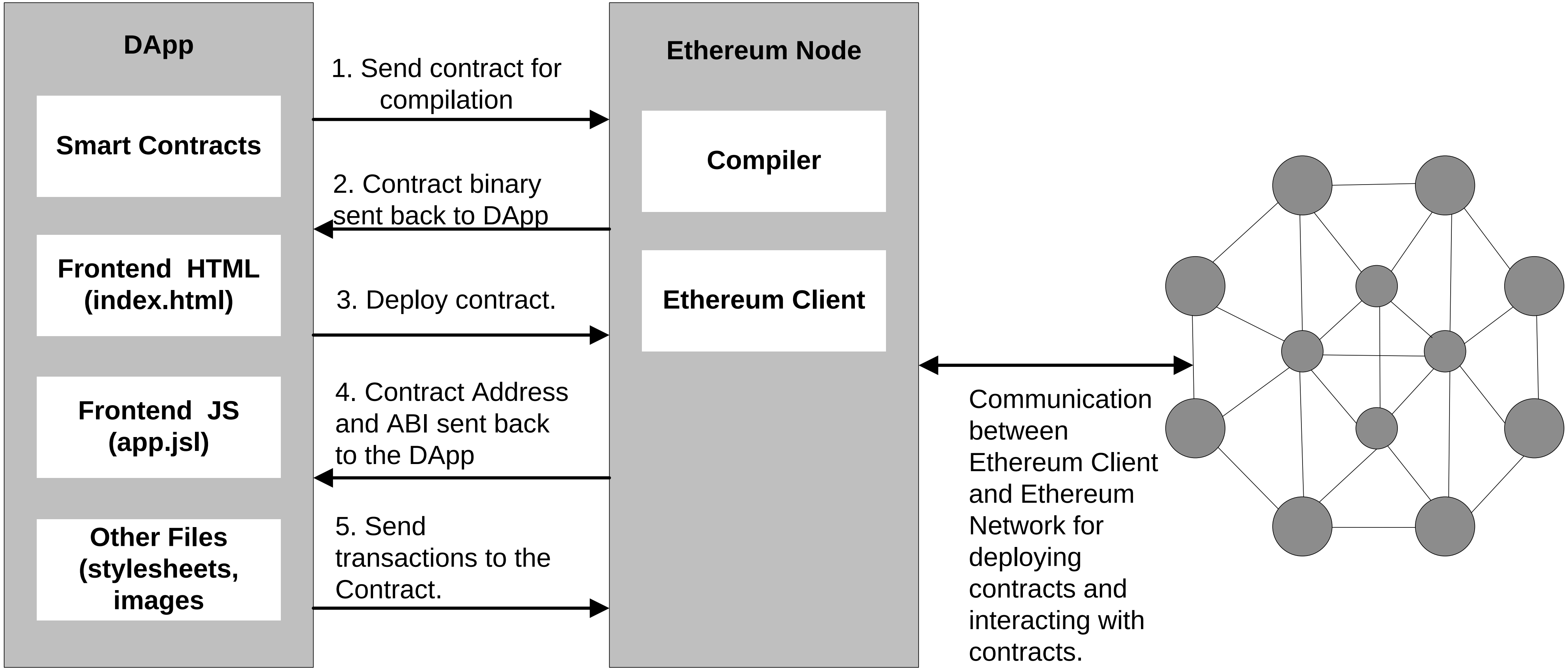
If an app= frontend + server, since Ethereum contracts are code that runs on the global Ethereum decentralized peer-to-peer network, then:

DApp = frontend + contracts



Source: https://ethereum.stackexchange.com/questions/383/what-is-a-dapp

DApp Creation and Execution Workflow (high-level)



DApp

Smart Contracts

Frontend HTML (index.html)

Frontend JS (app.jsl)

Other Files (stylesheets, images

1. Send contract for compilation

2. Contract binary sent back to DApp

3. Deploy contract.

4. Contract Address and ABI sent back to the DApp

5. Send transactions to the Contract.

Ethereum Node

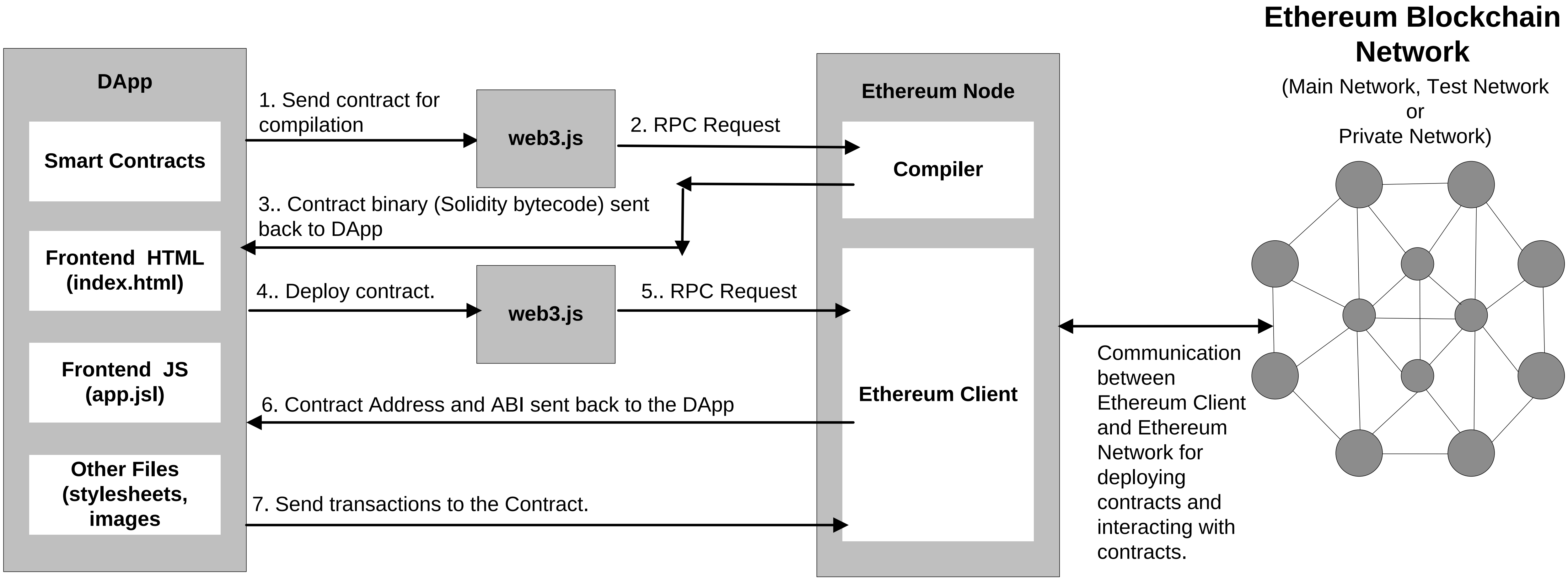
Compiler

Ethereum Client

Communication between Ethereum Client and Ethereum Network for deploying contracts and interacting with contracts.

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

DApp Creation and Execution Workflow (high-level with web3.js)


DApp
Smart Contracts
Frontend HTML (index.html)
Frontend JS (app.jsl)
Other Files (stylesheets, images
1. Send contract for compilation
web3.js
2. RPC Request
3.. Contract binary (Solidity bytecode) sent back to DApp
4.. Deploy contract.
web3.js
5.. RPC Request
6. Contract Address and ABI sent back to the DApp
7. Send transactions to the Contract.
Ethereum Node
Compiler
Ethereum Client
Communication between Ethereum Client and Ethereum Network for deploying contracts and interacting with contracts.
Ethereum Blockchain Network
(Main Network, Test Network or Private Network)

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

Ethereum Web3.js Tech Stack

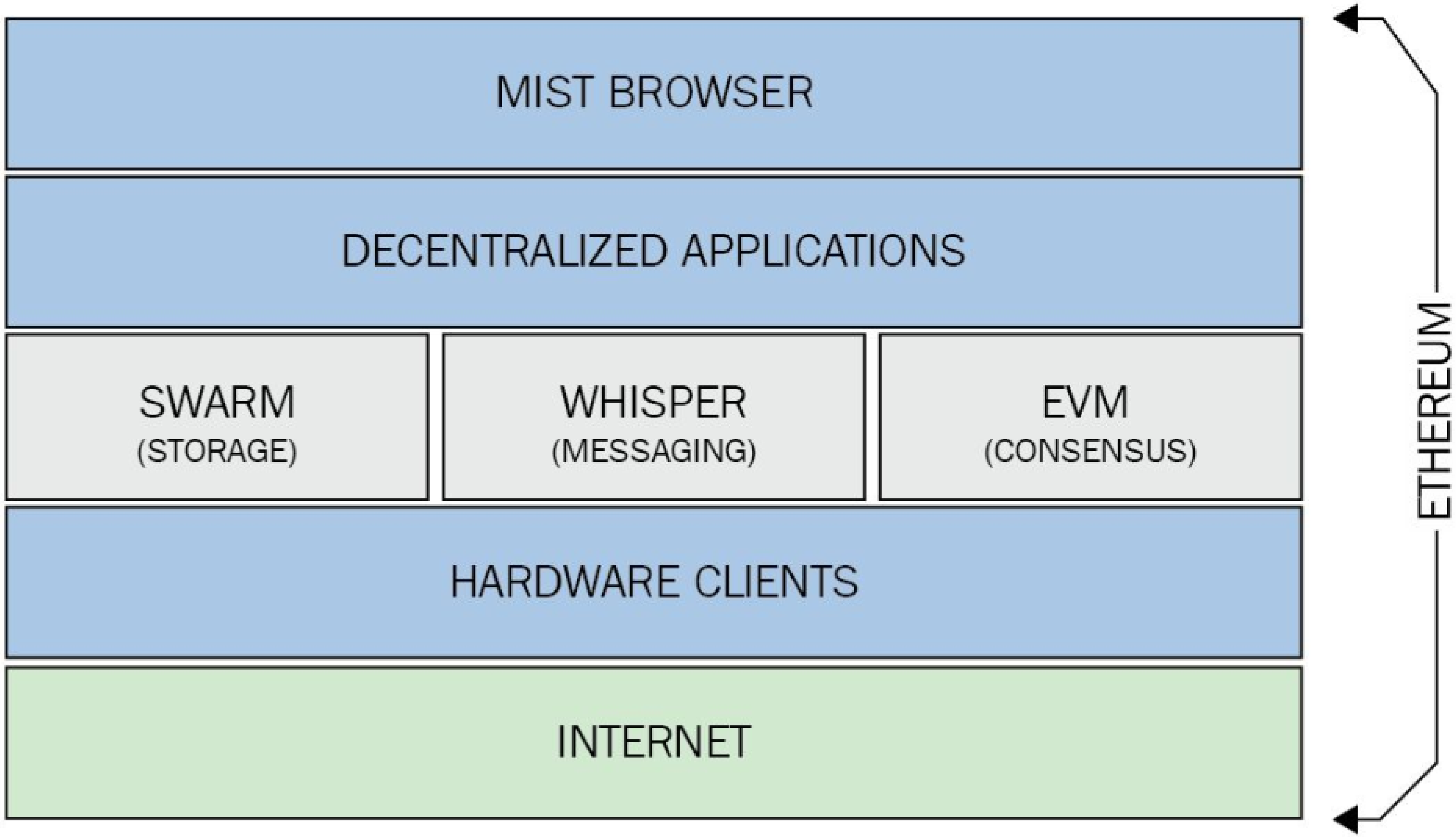
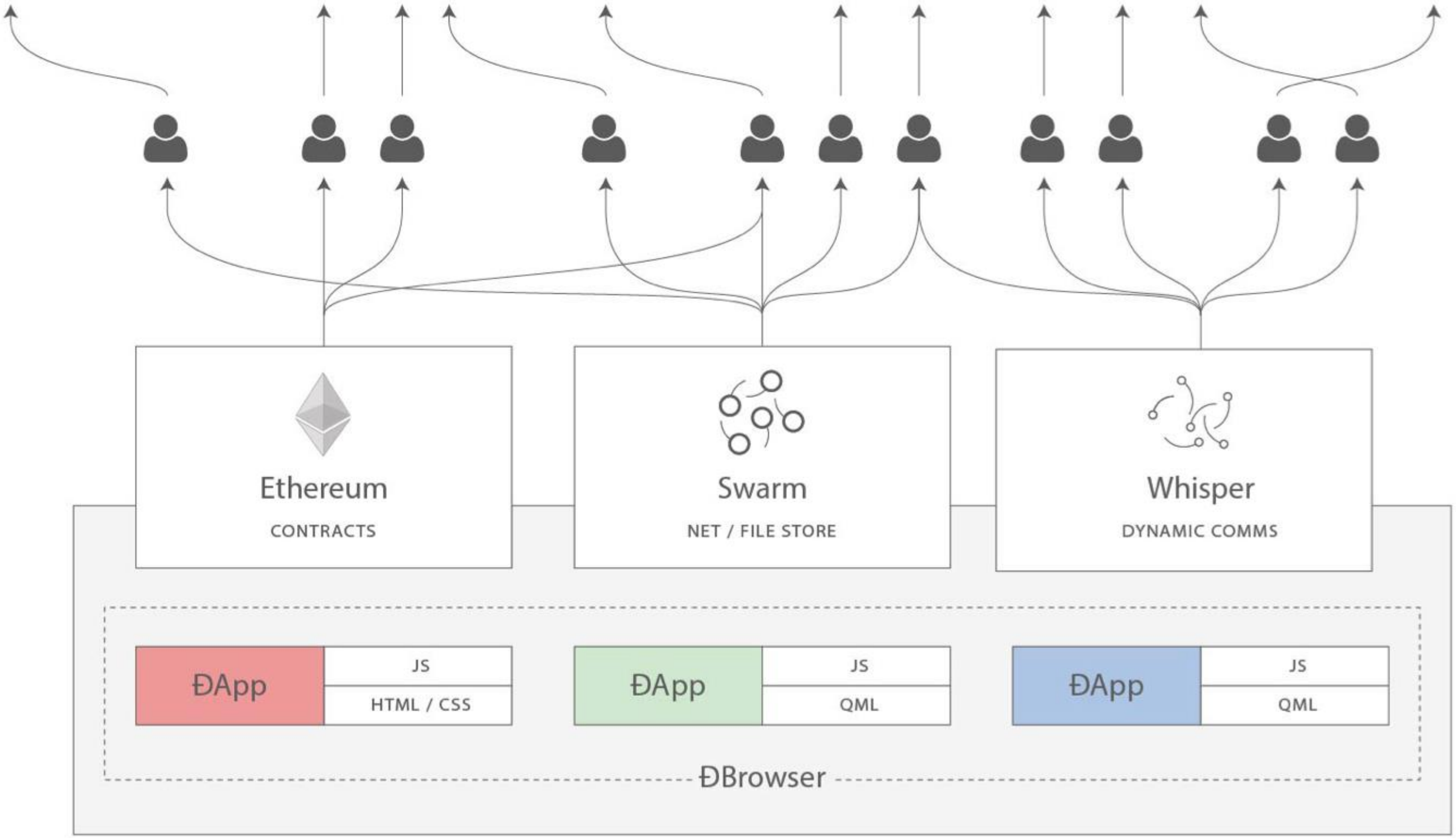


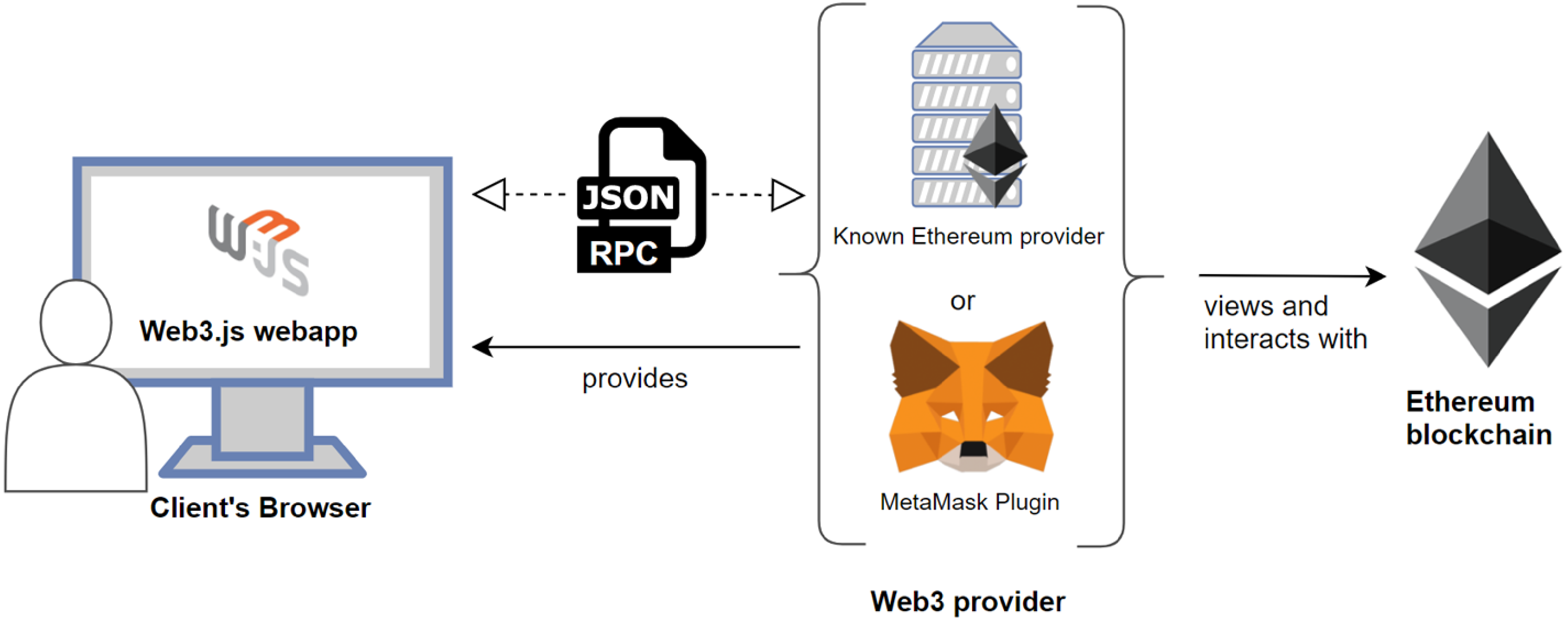
Figure 2.4: Web 3.0 tech stack for Ethereum, Source: Ethereum stack exchange

Web3 API Interaction Capabilities



Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Web3 and DApps



Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Three Types of DApps

Johnston states that there are three types of DApps.

1. **Type I decentralized applications have their own block chain, such as Bitcoin.**
2. **Type II decentralized applications use the blockchain of a type I decentralized application but are “protocols and have tokens that are necessary for their function” like the Omni Protocol.**
3. **Type III decentralized applications use the protocol of a type II decentralized application and “are protocols and have tokens that are necessary for their function,” such as the SAFE Network that uses the Omni Protocol to issue ‘safecoins.”**

Think of DApps as an operating system like Windows, Mac OS X, Linux, Android, iOS as a Type I classification. The programs on these systems, such as a word processor or Dropbox, would be Type II. A Type III example would then be a blogging platform that integrates Dropbox.

Source: Moses Sam Paul. https://medium.com/heptagon/step-by-step-guide-to-build-a-dapp-a-homo-sapiens-2-day-love-affair-with-ethereum-dapp-de2b0dea12f1

How a DApp Works with the Ethereum Ecosystem

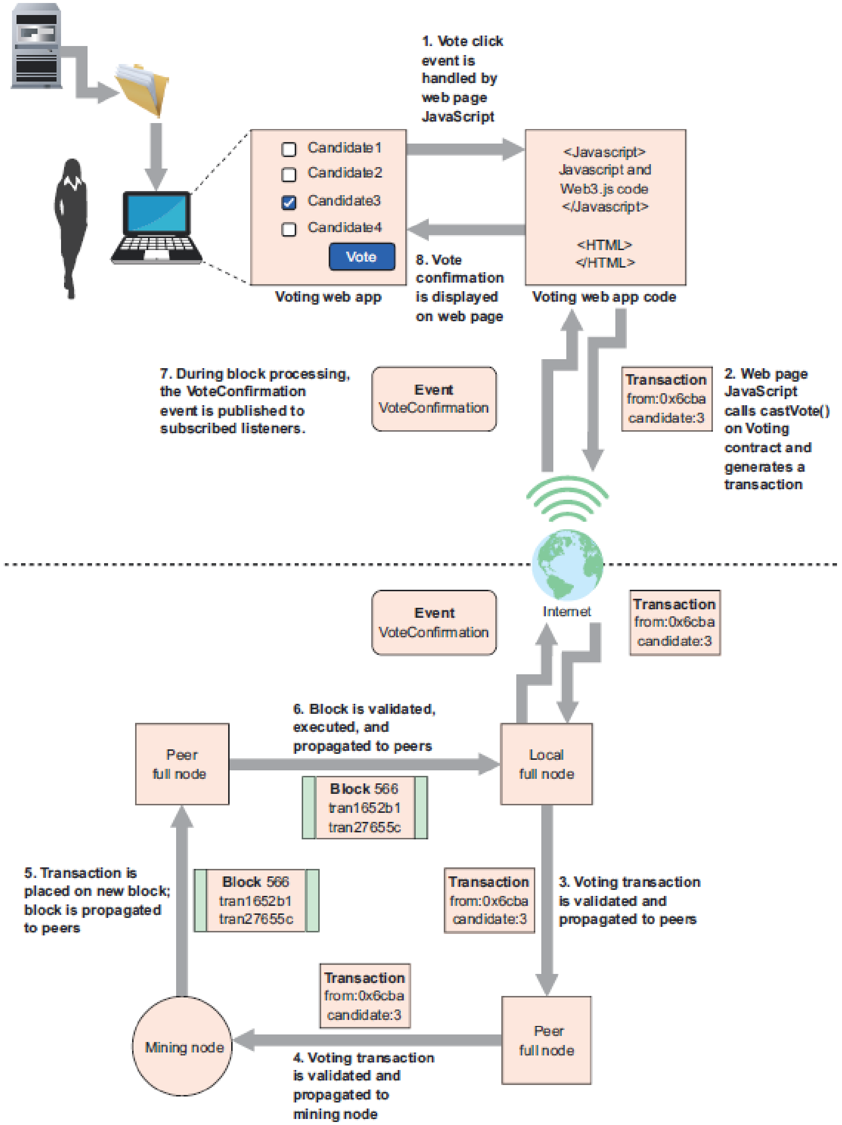
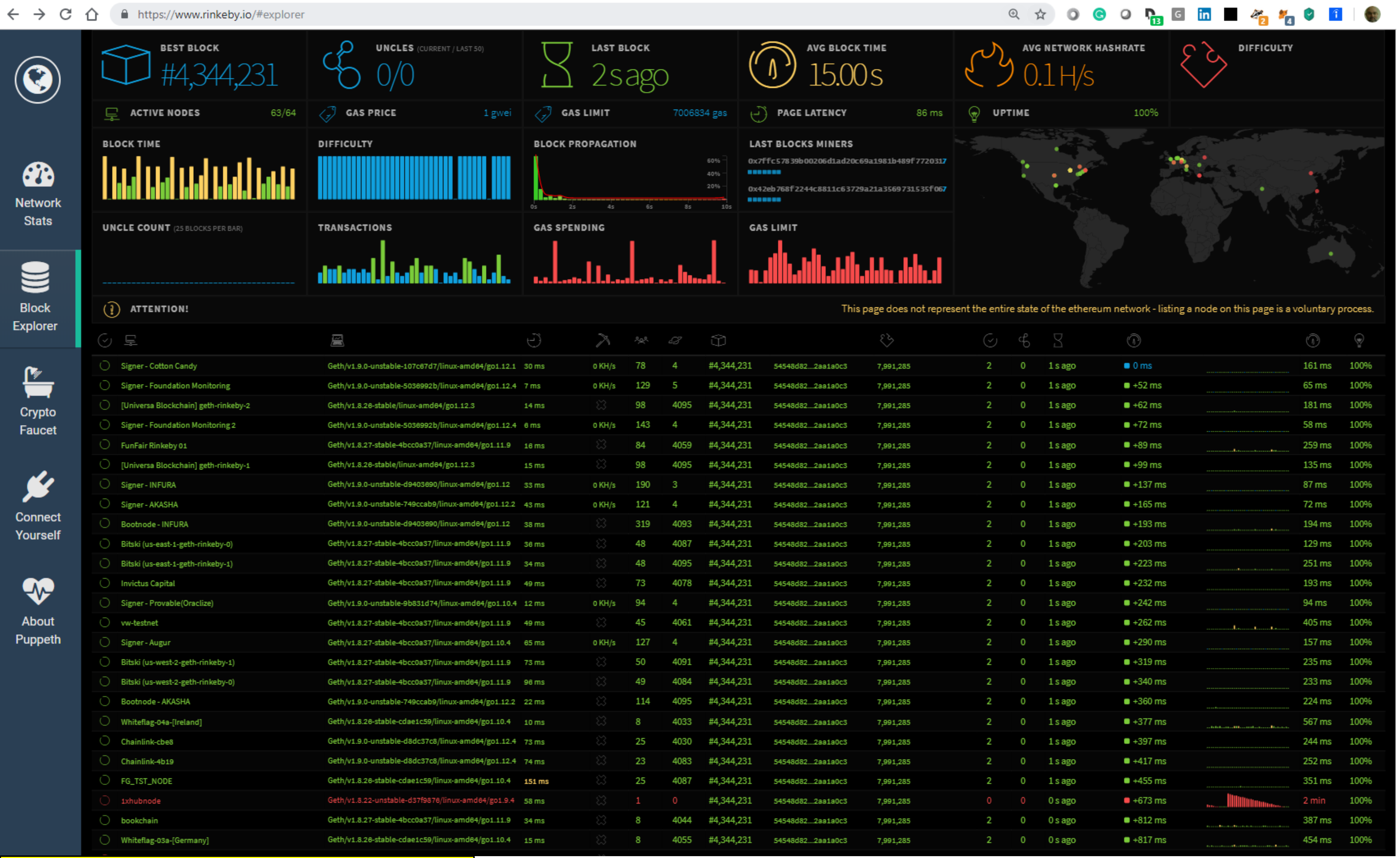

1. Vote click event is handled by web page JavaScript
□ Candidate1
□ Candidate2
☑ Candidate3
□ Candidate4
Vote
Voting web app
<Javascript> Javascript and Web3.js code </Javascript>
<HTML> </HTML>
Voting web app code
8. Vote confirmation is displayed on web page
7. During block processing, the VoteConfirmation event is published to subscribed listeners.
Event VoteConfirmation
Transaction from:0x6cba candidate:3
2. Web page JavaScript calls castVote() on Voting contract and generates a transaction
Internet
Event VoteConfirmation
Transaction from:0x6cba candidate:3
6. Block is validated, executed, and propagated to peers
Peer full node
Local full node
Block 566 tran1652b1 tran27655c
5. Transaction is placed on new block; block is propagated to peers
Block 566 tran1652b1 tran27655c
Transaction from:0x6cba candidate:3
3. Voting transaction is validated and propagated to peers
Transaction from:0x6cba candidate:3
Mining node
Peer full node
4. Voting transaction is validated and propagated to mining node

Figure 1.8 The lifecycle of a voting transaction. A voting transaction is created when a voter browser invokes the `castVote()` function on the Voting smart contract on a local node of the Ethereum network. This is then validated and propagated throughout the network until it's included on a new blockchain block by a mining node. The new block is propagated throughout the network, and then it finally gets back to the local node.

In the Ethereum Blockchain Ecosystem a new Block is mined about every 14 to 17 seconds

Source: Roberto Infante, Building Ethereum DApps, 2019

Rinkby Ethereum Test Blockchain Explorer



BEST BLOCK #4,344,231

UNCLES (CURRENT / LAST 50) 0/0

LAST BLOCK 2s ago

AVG BLOCK TIME 15.00s

AVG NETWORK HASHRATE 0.1 H/s

DIFFICULTY

ACTIVE NODES 63/64

GAS PRICE 1 gwei

GAS LIMIT 7006834 gas

PAGE LATENCY 86 ms

UPTIME 100%

BLOCK TIME

DIFFICULTY

BLOCK PROPAGATION

LAST BLOCKS MINERS

0x7ffc5783 9b00206d1ad20c69a1981b489f7720317

0x42eb768f2244c8811c63729a21a3569731535f067

UNCLE COUNT (25 BLOCKS PER BAR)

TRANSACTIONS

GAS SPENDING

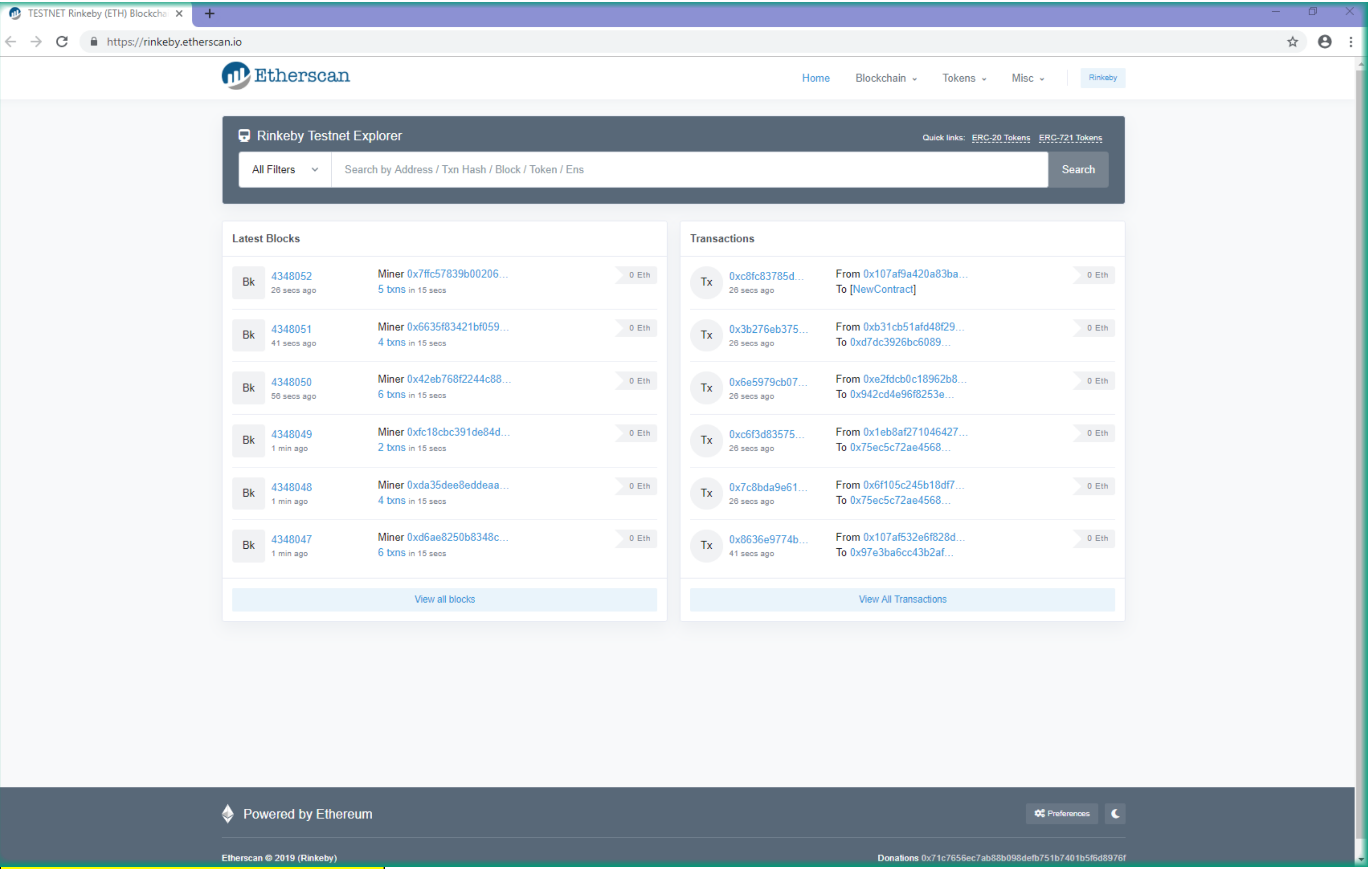
GAS LIMIT

ATTENTION! This page does not represent the entire state of the ethereum network - listing a node on this page is a voluntary process.

○	Signer - Cotton Candy	Geth/v1.9.0-unstable-107c67d7/linux-amd64/go1.12.1	30 ms	0 KH/s	78	4	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	0 ms	161 ms	100%
○	Signer - Foundation Monitoring	Geth/v1.9.0-unstable-5036992b/linux-amd64/go1.12.4	7 ms	0 KH/s	129	5	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+52 ms	65 ms	100%
○	[Universa Blockchain] geth-rinkeby-2	Geth/v1.8.26-stable/linux-amd64/go1.12.3	14 ms	✕	98	4095	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+62 ms	181 ms	100%
○	Signer - Foundation Monitoring 2	Geth/v1.9.0-unstable-5036992b/linux-amd64/go1.12.4	6 ms	0 KH/s	143	4	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+72 ms	58 ms	100%
○	FunFair Rinkeby 01	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	16 ms	✕	84	4059	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+89 ms	259 ms	100%
○	[Universa Blockchain] geth-rinkeby-1	Geth/v1.8.26-stable/linux-amd64/go1.12.3	15 ms	✕	98	4095	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+99 ms	135 ms	100%
○	Signer - INFURA	Geth/v1.9.0-unstable-d9403690/linux-amd64/go1.12	33 ms	0 KH/s	190	3	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+137 ms	87 ms	100%
○	Signer - AKASHA	Geth/v1.9.0-unstable-749ccab9/linux-amd64/go1.12.2	43 ms	0 KH/s	121	4	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+165 ms	72 ms	100%
○	Bootnode - INFURA	Geth/v1.9.0-unstable-d9403690/linux-amd64/go1.12	38 ms	✕	319	4093	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+193 ms	194 ms	100%
○	Bitski (us-east-1-geth-rinkeby-0)	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	36 ms	✕	48	4087	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+203 ms	129 ms	100%
○	Bitski (us-east-1-geth-rinkeby-1)	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	34 ms	✕	48	4095	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+223 ms	251 ms	100%
○	Invictus Capital	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	49 ms	✕	73	4078	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+232 ms	193 ms	100%
○	Signer - Provable(Oraclize)	Geth/v1.9.0-unstable-9b831d74/linux-amd64/go1.10.4	12 ms	0 KH/s	94	4	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+242 ms	94 ms	100%
○	vw-testnet	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	49 ms	✕	45	4061	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+262 ms	405 ms	100%
○	Signer - Augur	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.10.4	65 ms	0 KH/s	127	4	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+290 ms	157 ms	100%
○	Bitski (us-west-2-geth-rinkeby-1)	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	73 ms	✕	50	4091	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+319 ms	235 ms	100%
○	Bitski (us-west-2-geth-rinkeby-0)	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	96 ms	✕	49	4084	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+340 ms	233 ms	100%
○	Bootnode - AKASHA	Geth/v1.9.0-unstable-749ccab9/linux-amd64/go1.12.2	22 ms	✕	114	4095	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+360 ms	224 ms	100%
○	Whiteflag-04a-[Ireland]	Geth/v1.8.26-stable-cdae1c59/linux-amd64/go1.10.4	10 ms	✕	8	4033	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+377 ms	567 ms	100%
○	Chainlink-cbe8	Geth/v1.9.0-unstable-d8dc37c8/linux-amd64/go1.12.4	73 ms	✕	25	4030	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+397 ms	244 ms	100%
○	Chainlink-4b19	Geth/v1.9.0-unstable-d8dc37c8/linux-amd64/go1.12.4	74 ms	✕	23	4083	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+417 ms	252 ms	100%
○	FG_TST_NODE	Geth/v1.8.26-stable-cdae1c59/linux-amd64/go1.10.4	151 ms	✕	25	4087	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	1 s ago	+455 ms	351 ms	100%
○	1xhubnode	Geth/v1.8.22-unstable-d37f9876/linux-amd64/go1.9.4	58 ms	✕	1	0	#4,344,231	54548d82...2aa1a0c3	7,991,285	0	0	0 s ago	+673 ms	2 min	100%
○	bookchain	Geth/v1.8.27-stable-4bcc0a37/linux-amd64/go1.11.9	34 ms	✕	8	4044	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	0 s ago	+812 ms	387 ms	100%
○	Whiteflag-03a-[Germany]	Geth/v1.8.26-stable-cdae1c59/linux-amd64/go1.10.4	15 ms	✕	8	4055	#4,344,231	54548d82...2aa1a0c3	7,991,285	2	0	0 s ago	+817 ms	454 ms	100%

Source: https://www.rinkeby.io/#explorer

Etherscan Ethereum Blockchain Explorer


TESTNET Rinkeby (ETH) Blockcha
https://rinkeby.etherscan.io
Etherscan
Home
Blockchain
Tokens
Misc
Rinkeby

Rinkeby Testnet Explorer

Quick links: ERC-20 Tokens ERC-721 Tokens

All Filters
Search by Address / Txn Hash / Block / Token / Ens
Search

Latest Blocks

Bk	4348052 26 secs ago	Miner 0x7ffc57839b00206... 5 txns in 15 secs	0 Eth
Bk	4348051 41 secs ago	Miner 0x6635f83421bf059... 4 txns in 15 secs	0 Eth
Bk	4348050 56 secs ago	Miner 0x42eb768f2244c88... 6 txns in 15 secs	0 Eth
Bk	4348049 1 min ago	Miner 0xfc18cbc391de84d... 2 txns in 15 secs	0 Eth
Bk	4348048 1 min ago	Miner 0xda35dee8eddeaa... 4 txns in 15 secs	0 Eth
Bk	4348047 1 min ago	Miner 0xd6ae8250b8348c... 6 txns in 15 secs	0 Eth

View all blocks

Transactions

Tx	0xc8fc83785d... 26 secs ago	From 0x107af9a420a83ba... To [NewContract]	0 Eth
Tx	0x3b276eb375... 26 secs ago	From 0xb31cb51afd48f29... To 0xd7dc3926bc6089...	0 Eth
Tx	0x6e5979cb07... 26 secs ago	From 0xe2fdcb0c18962b8... To 0x942cd4e96f8253e...	0 Eth
Tx	0xc6f3d83575... 26 secs ago	From 0x1eb8af271046427... To 0x75ec5c72ae4568...	0 Eth
Tx	0x7c8bda9e61... 26 secs ago	From 0x6f105c245b18df7... To 0x75ec5c72ae4568...	0 Eth
Tx	0x8636e9774b... 41 secs ago	From 0x107af532e6f828d... To 0x97e3ba6cc43b2af...	0 Eth

View All Transactions

Powered by Ethereum

Etherscan © 2019 (Rinkeby)

Donations 0x71c7656ec7ab88b098defb751b7401b5f6d8976f

Sourcehttps://rinkeby.etherscan.io

Set up and Test Geth

Geth Workout

Download and install Geth, the Ethereum Blockchain software

(Written for Windows Users)

- Visit this website, to download Geth:
- https://geth.ethereum.org/downloads/

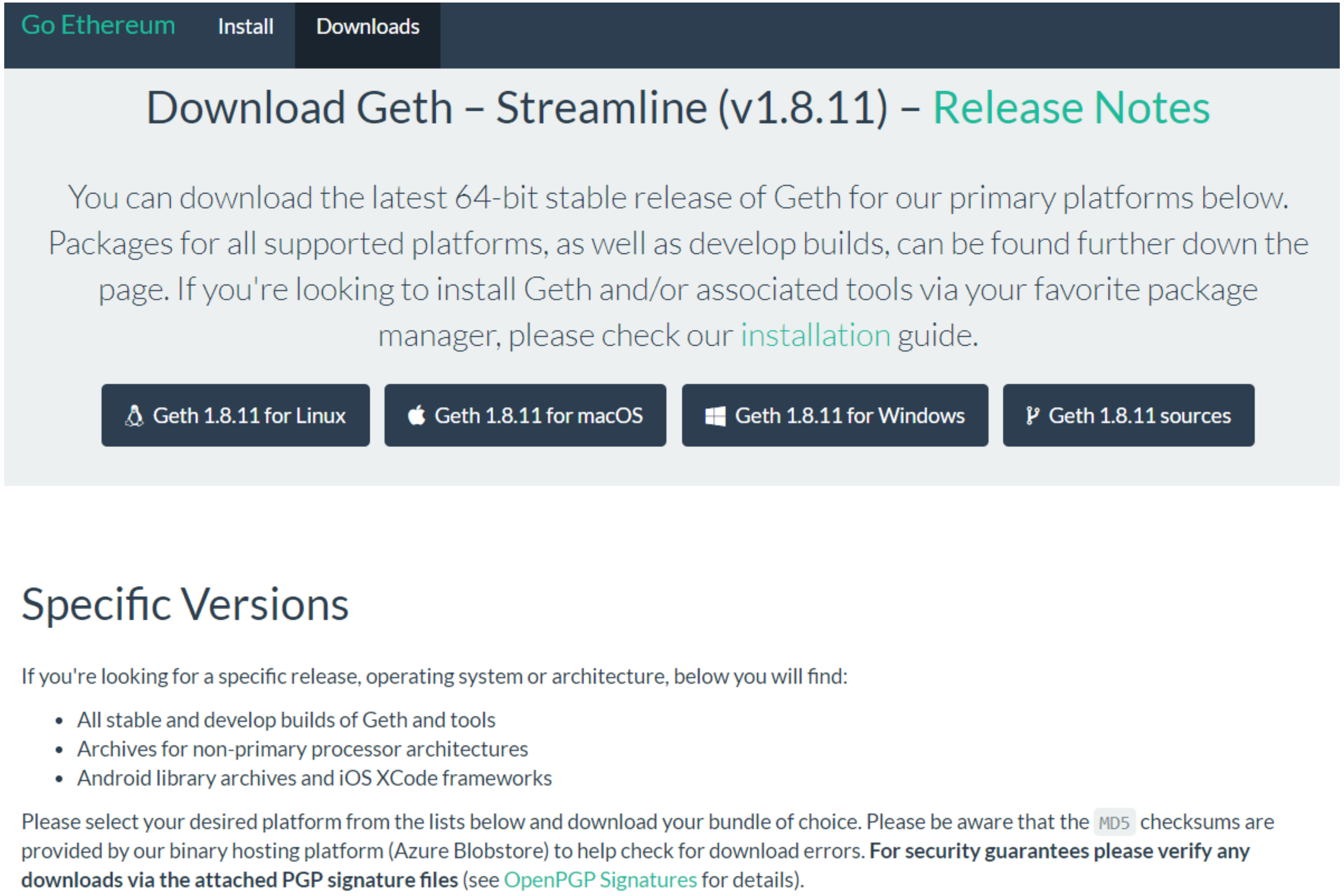
2. Install Geth into a directory you will create: c:\ethereum

3. At the command line, launch Geth in testnet mode

4. Switch to miner mode

5. Extra Credit: if you set up an Ethereum Account, you can actually write data (like your name) to the Ethereum Blockchain and view it

Download Geth



Download Geth – Streamline (v1.8.11) – Release Notes

You can download the latest 64-bit stable release of Geth for our primary platforms below. Packages for all supported platforms, as well as develop builds, can be found further down the page. If you're looking to install Geth and/or associated tools via your favorite package manager, please check our installation guide.

Specific Versions

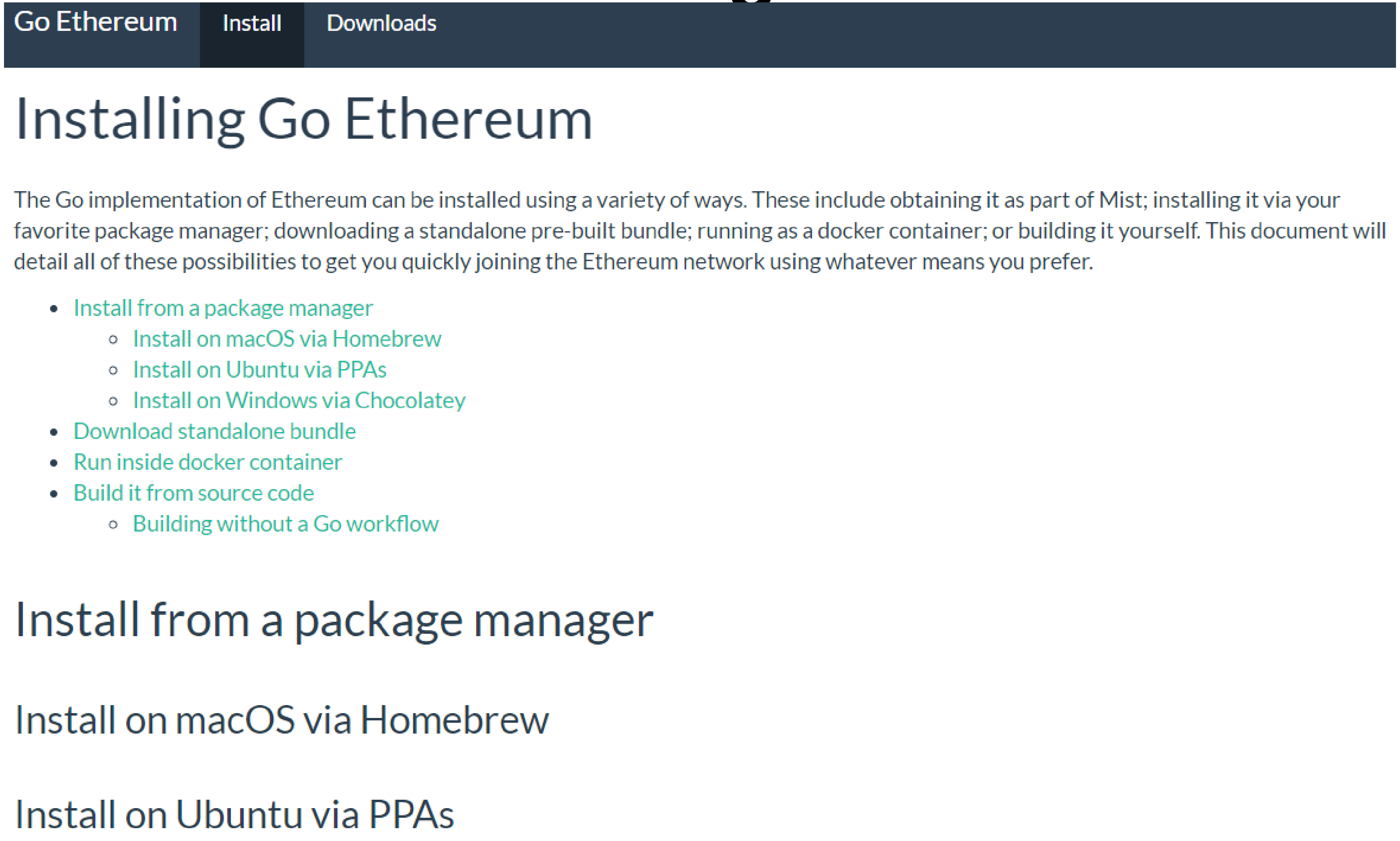
If you're looking for a specific release, operating system or architecture, below you will find:

- All stable and develop builds of Geth and tools
- Archives for non-primary processor architectures
- Android library archives and iOS XCode frameworks

Please select your desired platform from the lists below and download your bundle of choice. Please be aware that the `MD5` checksums are provided by our binary hosting platform (Azure Blobstore) to help check for download errors. **For security guarantees please verify any downloads via the attached PGP signature files** (see OpenPGP Signatures for details).

Source: https://geth.ethereum.org/downloads/

Installing Geth



Installing Go Ethereum

The Go implementation of Ethereum can be installed using a variety of ways. These include obtaining it as part of Mist; installing it via your favorite package manager; downloading a standalone pre-built bundle; running as a docker container; or building it yourself. This document will detail all of these possibilities to get you quickly joining the Ethereum network using whatever means you prefer.

- Install from a package manager
  - Install on macOS via Homebrew
  - Install on Ubuntu via PPAs
  - Install on Windows via Chocolatey
- Download standalone bundle
- Run inside docker container
- Build it from source code
  - Building without a Go workflow

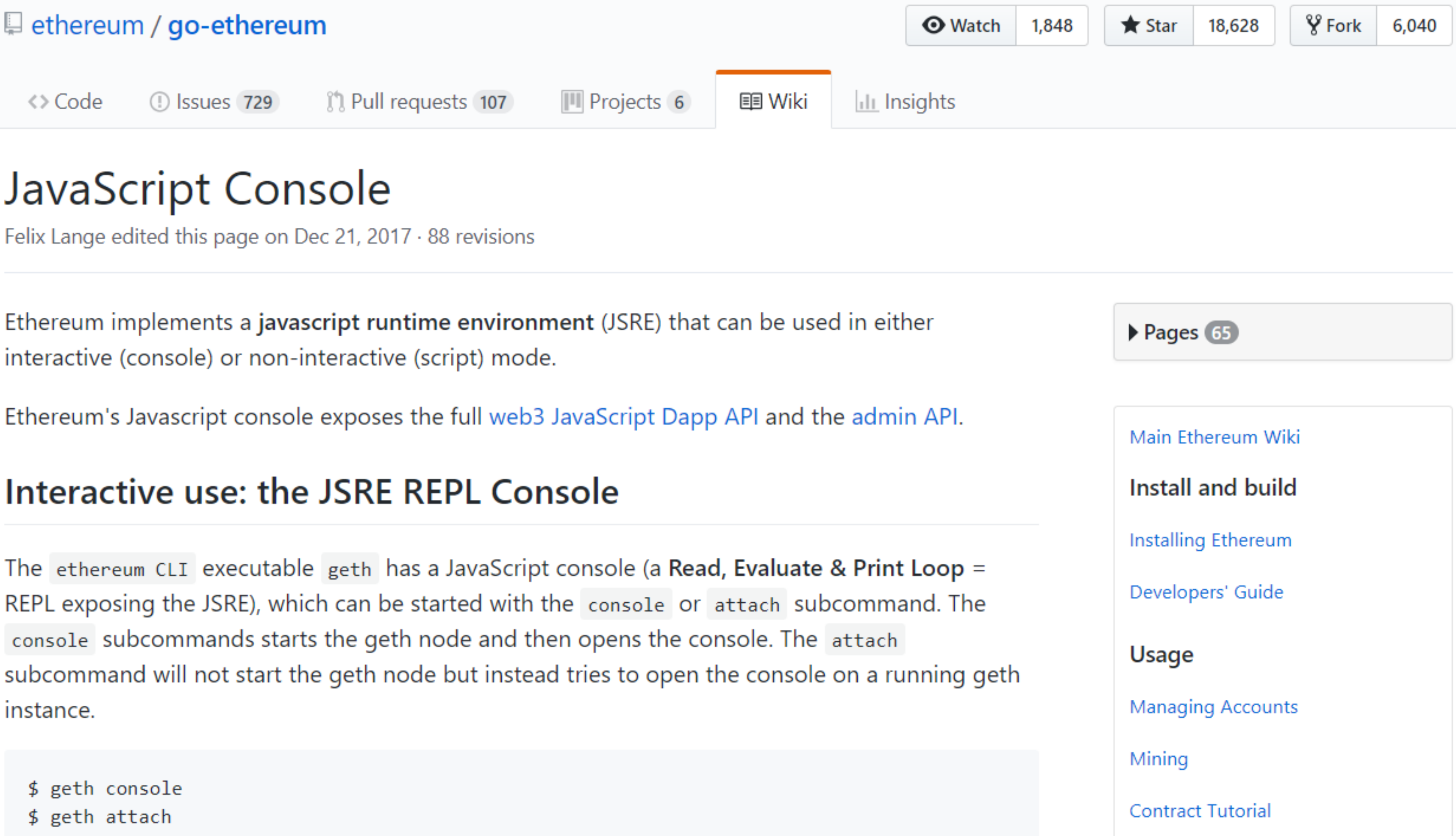
Install from a package manager

Install on macOS via Homebrew

Install on Ubuntu via PPAs

Source: https://geth.ethereum.org/downloads/

Starting the Javascript Console



JavaScript Console

Felix Lange edited this page on Dec 21, 2017 · 88 revisions

Ethereum implements a **javascript runtime environment** (JSRE) that can be used in either interactive (console) or non-interactive (script) mode.

Ethereum's Javascript console exposes the full web3 JavaScript Dapp API and the admin API.

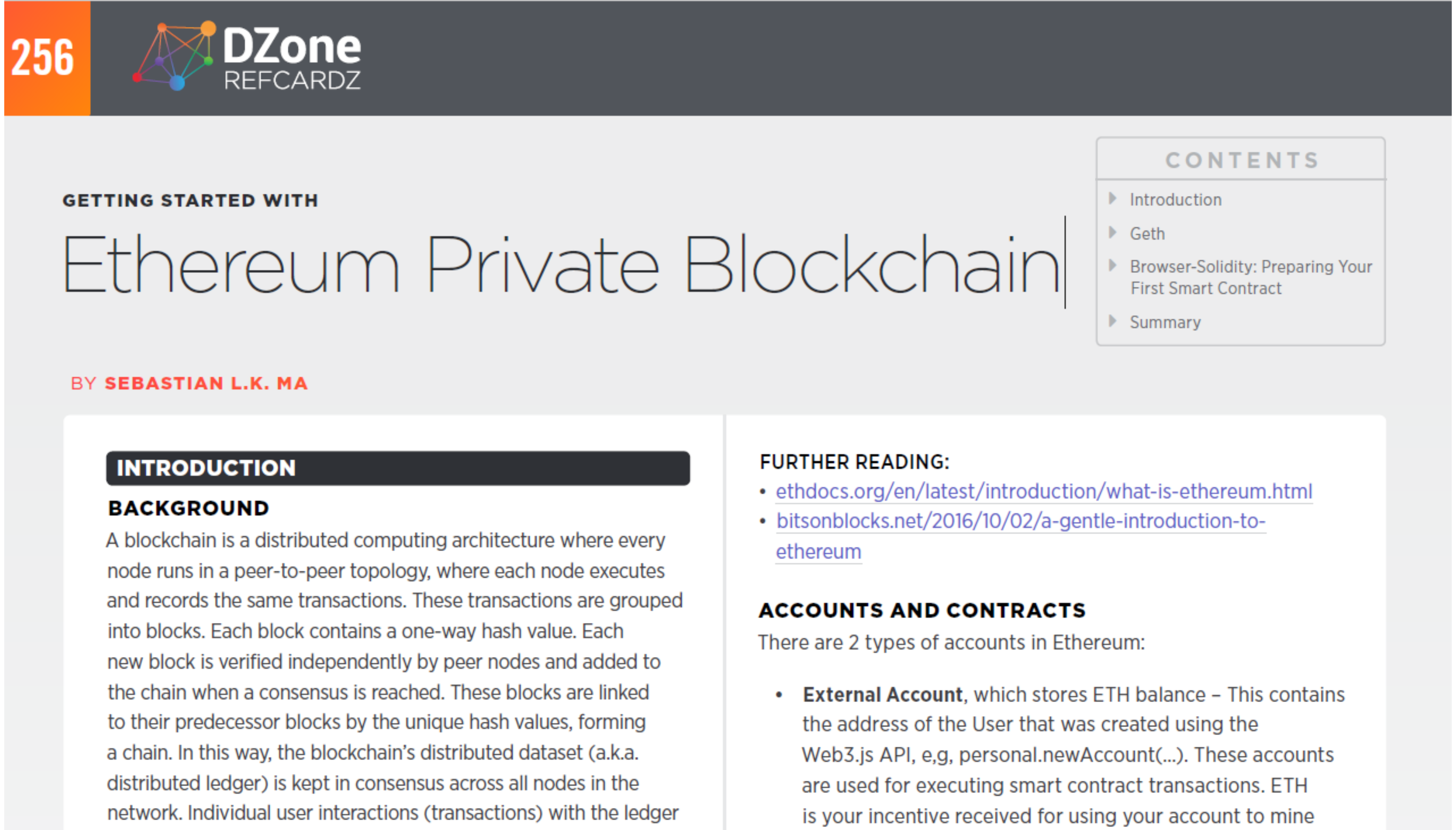
Interactive use: the JSRE REPL Console

The `ethereum CLI` executable `geth` has a JavaScript console (a **Read, Evaluate & Print Loop** = REPL exposing the JSRE), which can be started with the `console` or `attach` subcommand. The `console` subcommands starts the geth node and then opens the console. The `attach` subcommand will not start the geth node but instead tries to open the console on a running geth instance.

```
$ geth console
$ geth attach
```

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Getting Started with Ethereum Private Blockchain



256 DZone REFCARDZ

GETTING STARTED WITH

Ethereum Private Blockchain

BY SEBASTIAN L.K. MA

INTRODUCTION

BACKGROUND

A blockchain is a distributed computing architecture where every node runs in a peer-to-peer topology, where each node executes and records the same transactions. These transactions are grouped into blocks. Each block contains a one-way hash value. Each new block is verified independently by peer nodes and added to the chain when a consensus is reached. These blocks are linked to their predecessor blocks by the unique hash values, forming a chain. In this way, the blockchain's distributed dataset (a.k.a. distributed ledger) is kept in consensus across all nodes in the network. Individual user interactions (transactions) with the ledger

FURTHER READING:

- ethdocs.org/en/latest/introduction/what-is-ethereum.html
- bitsonblocks.net/2016/10/02/a-gentle-introduction-to-ethereum

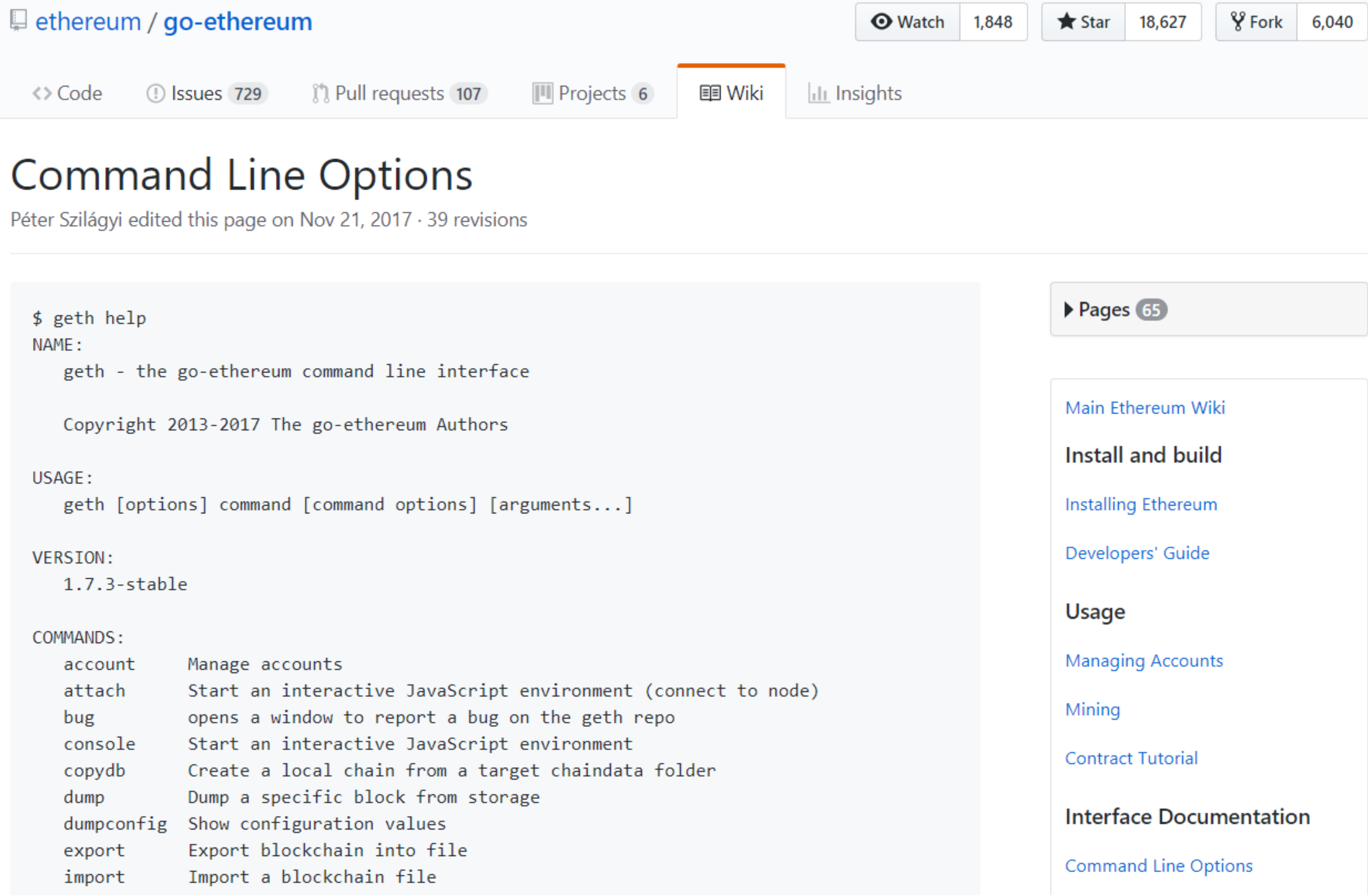
ACCOUNTS AND CONTRACTS

There are 2 types of accounts in Ethereum:

- **External Account**, which stores ETH balance – This contains the address of the User that was created using the Web3.js API, e,g, personal.newAccount(...). These accounts are used for executing smart contract transactions. ETH is your incentive received for using your account to mine

Source: https://dzone.com/refcardz/getting-started-with-etherium-private-blockchain?chapter=1/

Geth Command Line



Command Line Options

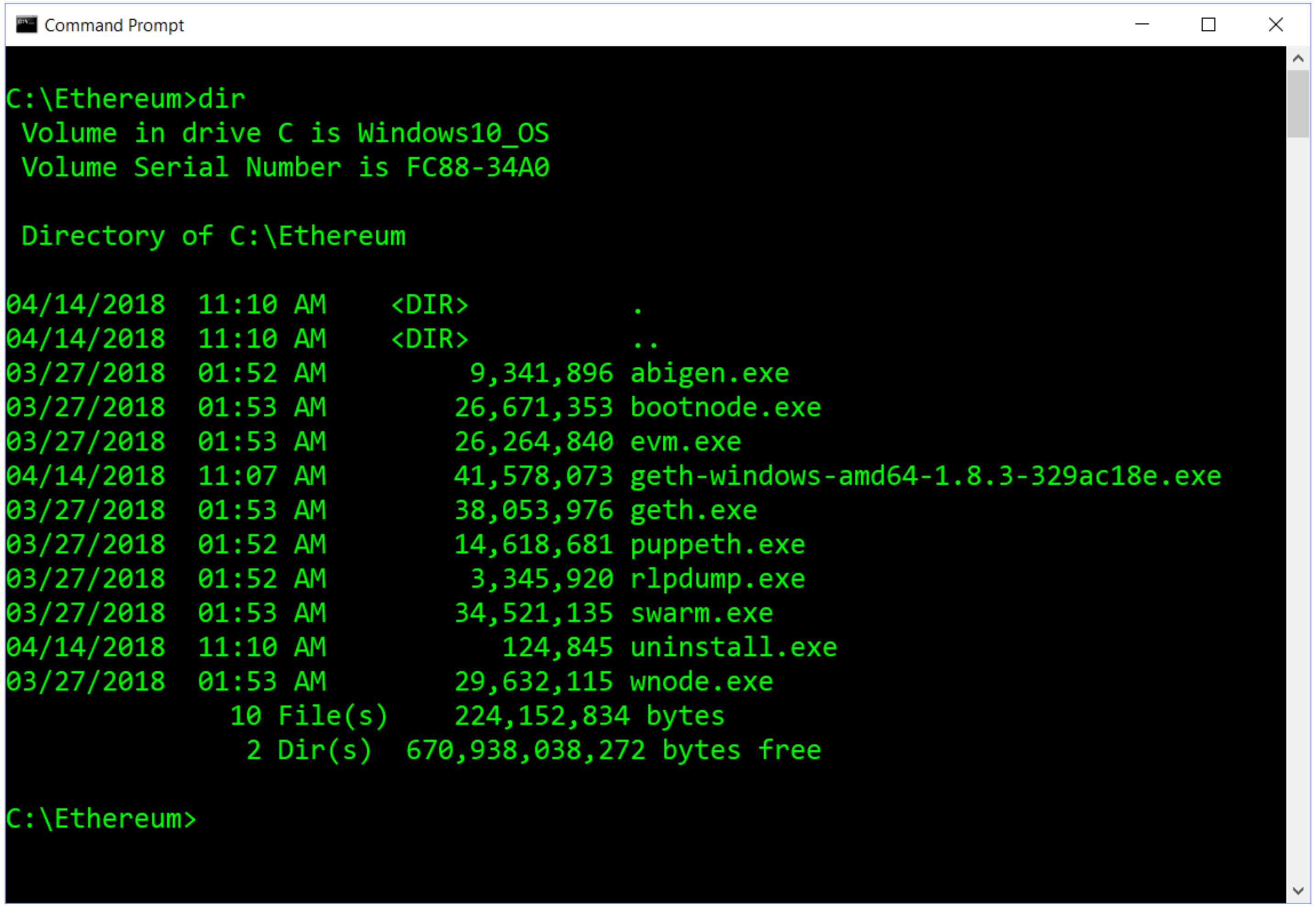
Péter Szilágyi edited this page on Nov 21, 2017 · 39 revisions

```
$ geth help
NAME:
   geth - the go-ethereum command line interface

   Copyright 2013-2017 The go-ethereum Authors

USAGE:
   geth [options] command [command options] [arguments...]

VERSION:
   1.7.3-stable

COMMANDS:
   account     Manage accounts
   attach      Start an interactive JavaScript environment (connect to node)
   bug         opens a window to report a bug on the geth repo
   console     Start an interactive JavaScript environment
   copydb      Create a local chain from a target chaindata folder
   dump        Dump a specific block from storage
   dumpconfig  Show configuration values
   export      Export blockchain into file
   import      Import a blockchain file
```

Source: https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options

Geth Workout

In Windows, Geth at the Command



```
C:\Ethereum>dir
 Volume in drive C is Windows10_OS
 Volume Serial Number is FC88-34A0

 Directory of C:\Ethereum

04/14/2018  11:10 AM    <DIR>          .
04/14/2018  11:10 AM    <DIR>          ..
03/27/2018  01:52 AM         9,341,896 abigen.exe
03/27/2018  01:53 AM        26,671,353 bootnode.exe
03/27/2018  01:53 AM        26,264,840 evm.exe
04/14/2018  11:07 AM        41,578,073 geth-windows-amd64-1.8.3-329ac18e.exe
03/27/2018  01:53 AM        38,053,976 geth.exe
03/27/2018  01:52 AM        14,618,681 puppeth.exe
03/27/2018  01:52 AM         3,345,920 rlpdump.exe
03/27/2018  01:53 AM        34,521,135 swarm.exe
04/14/2018  11:10 AM           124,845 uninstall.exe
03/27/2018  01:53 AM        29,632,115 wnode.exe
              10 File(s)    224,152,834 bytes
               2 Dir(s)  670,938,038,272 bytes free

C:\Ethereum>
```

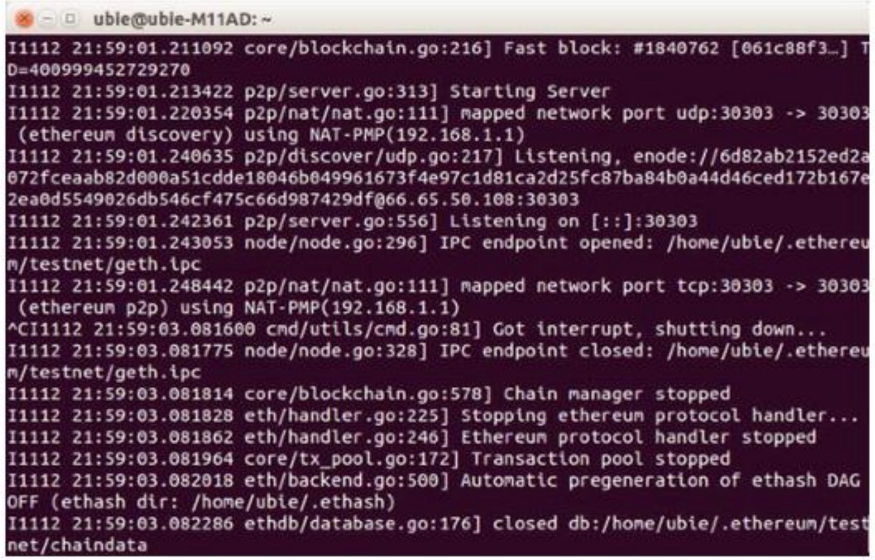
Geth Workout

In Windows, Geth at the Command Line

To start Geth on the testnet , type this:

geth --testnet

You'll see text output similar to the screen in Figure 6-6, except that this mining is taking place on the testnet. Press Control+C to stop it.



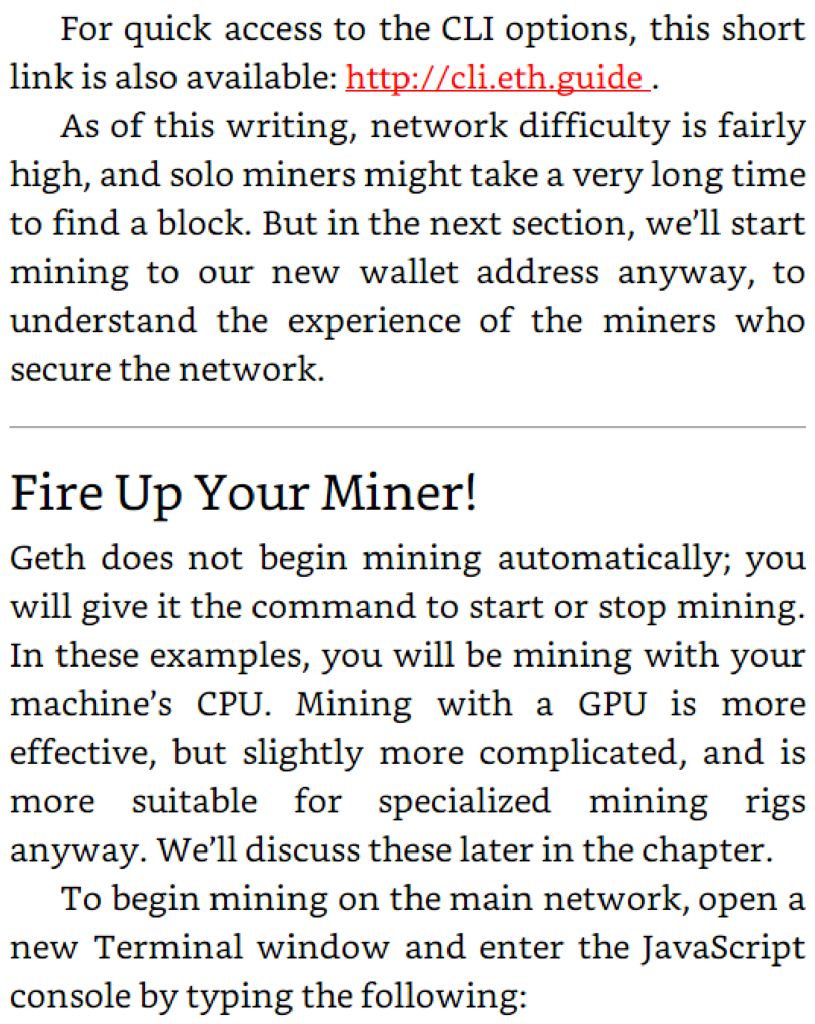
```
I1112 21:59:01.211092 core/blockchain.go:216] Fast block: #1840762 [061c88f3…] T
D=400999452729270
I1112 21:59:01.213422 p2p/server.go:313] Starting Server
I1112 21:59:01.220354 p2p/nat/nat.go:111] mapped network port udp:30303 -> 30303
 (ethereum discovery) using NAT-PMP(192.168.1.1)
I1112 21:59:01.240635 p2p/discover/udp.go:217] Listening, enode://6d82ab2152ed2a
072fceaab82d000a51cdde18046b049961673f4e97c1d81ca2d25fc87ba84b0a44d46ced172b167e
2ea0d5549026db546cf475c66d987429df@66.65.50.108:30303
I1112 21:59:01.242361 p2p/server.go:556] Listening on [::]:30303
I1112 21:59:01.243053 node/node.go:296] IPC endpoint opened: /home/ubie/.ethereu
m/testnet/geth.ipc
I1112 21:59:01.248442 p2p/nat/nat.go:111] mapped network port tcp:30303 -> 30303
 (ethereum p2p) using NAT-PMP(192.168.1.1)
^CI1112 21:59:03.081600 cmd/utils/cmd.go:81] Got interrupt, shutting down...
I1112 21:59:03.081775 node/node.go:328] IPC endpoint closed: /home/ubie/.ethereu
m/testnet/geth.ipc
I1112 21:59:03.081814 core/blockchain.go:578] Chain manager stopped
I1112 21:59:03.081828 eth/handler.go:225] Stopping ethereum protocol handler...
I1112 21:59:03.081862 eth/handler.go:246] Ethereum protocol handler stopped
I1112 21:59:03.081964 core/tx_pool.go:172] Transaction pool stopped
I1112 21:59:03.082018 eth/backend.go:500] Automatic pregeneration of ethash DAG
OFF (ethash dir: /home/ubie/.ethash)
I1112 21:59:03.082286 ethdb/database.go:176] closed db:/home/ubie/.ethereum/test
net/chaindata
```

Figure 6-6. Output from testnet

Source: Introducing Ethereum and Solidity – by Chris Dannen (Published by Apress)

Geth Workout

In Windows, Geth at the Command Line



For quick access to the CLI options, this short link is also available: http://cli.eth.guide .

As of this writing, network difficulty is fairly high, and solo miners might take a very long time to find a block. But in the next section, we'll start mining to our new wallet address anyway, to understand the experience of the miners who secure the network.

Fire Up Your Miner!

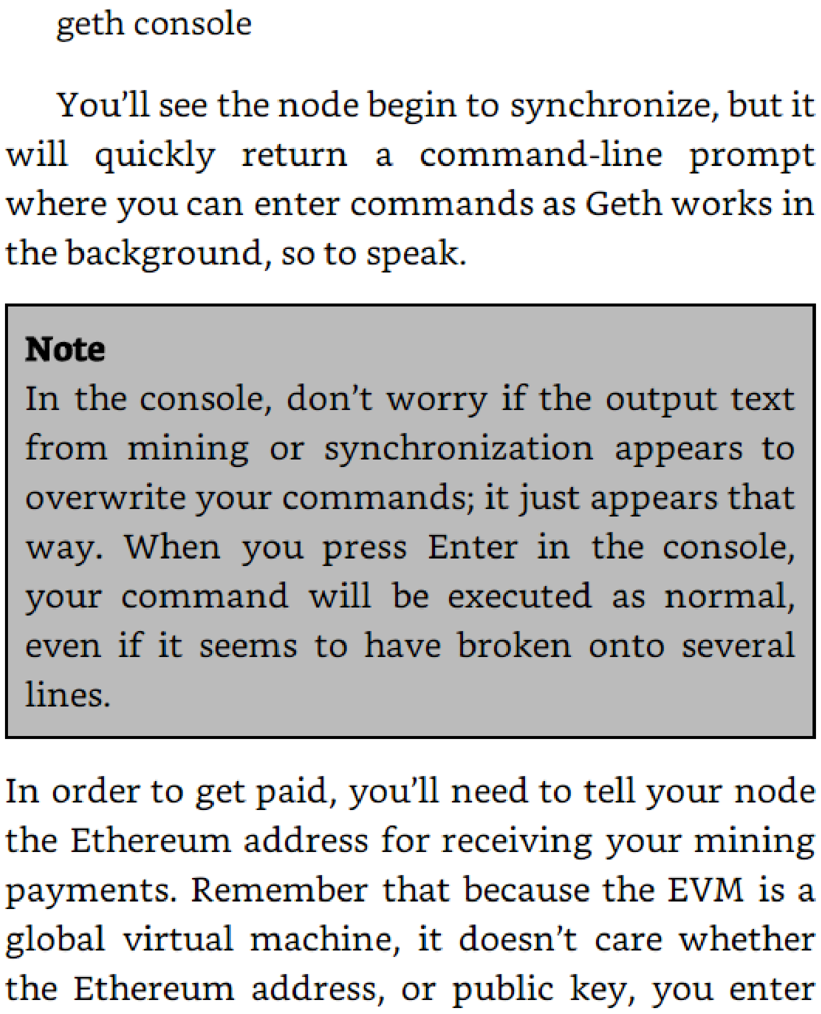
Geth does not begin mining automatically; you will give it the command to start or stop mining. In these examples, you will be mining with your machine's CPU. Mining with a GPU is more effective, but slightly more complicated, and is more suitable for specialized mining rigs anyway. We'll discuss these later in the chapter.

To begin mining on the main network, open a new Terminal window and enter the JavaScript console by typing the following:

Source: Introducing Ethereum and Solidity – by Chris Dannen (Published by Apress)

Geth Workout

In Windows, Geth at the Command Line



```
geth console
```

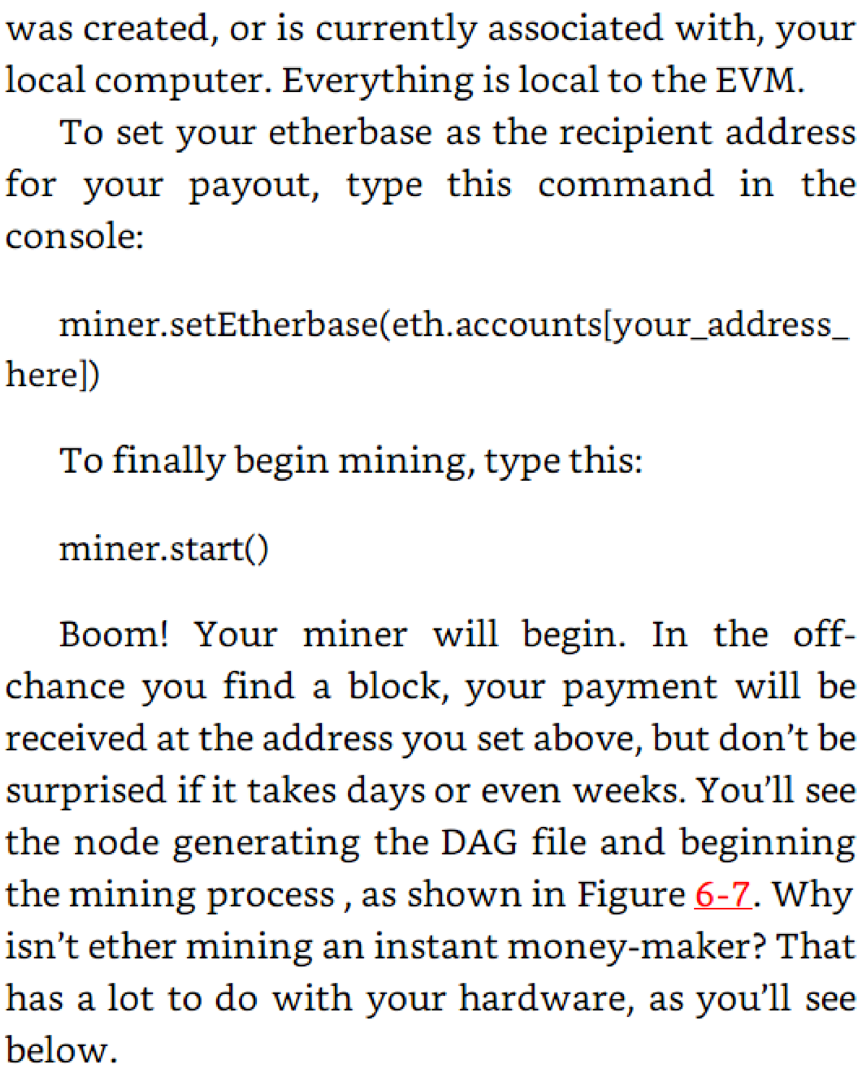
You'll see the node begin to synchronize, but it will quickly return a command-line prompt where you can enter commands as Geth works in the background, so to speak.

> **Note**
> In the console, don't worry if the output text from mining or synchronization appears to overwrite your commands; it just appears that way. When you press Enter in the console, your command will be executed as normal, even if it seems to have broken onto several lines.

In order to get paid, you'll need to tell your node the Ethereum address for receiving your mining payments. Remember that because the EVM is a global virtual machine, it doesn't care whether the Ethereum address, or public key, you enter

Source: Introducing Ethereum and Solidity – by Chris Dannen (Published by Apress)

Geth Workout



was created, or is currently associated with, your local computer. Everything is local to the EVM.

To set your etherbase as the recipient address for your payout, type this command in the console:

```
miner.setEtherbase(eth.accounts[your_address_
here])
```

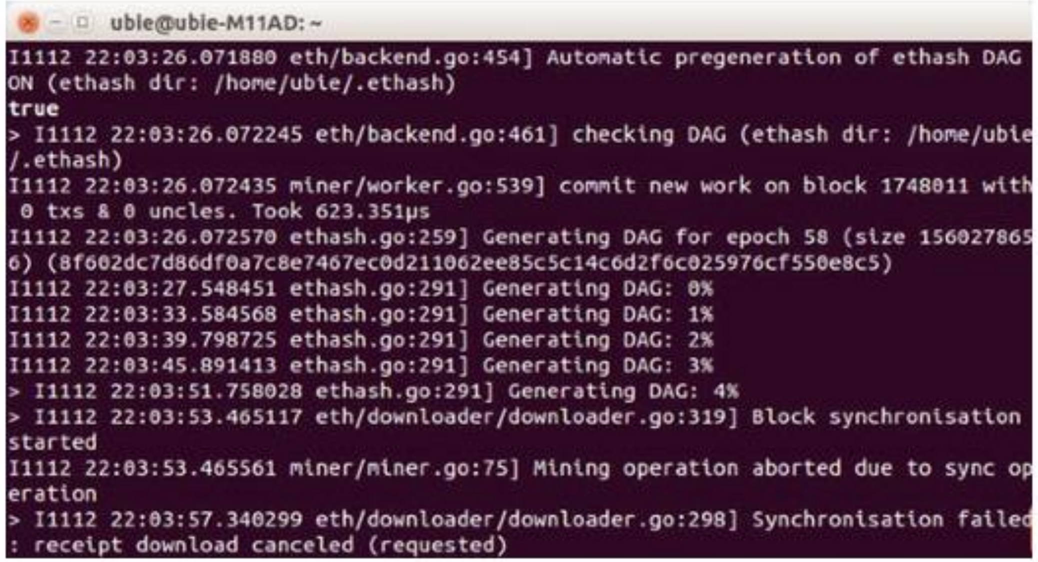
To finally begin mining, type this:

```
miner.start()
```

Boom! Your miner will begin. In the off-chance you find a block, your payment will be received at the address you set above, but don't be surprised if it takes days or even weeks. You'll see the node generating the DAG file and beginning the mining process , as shown in Figure 6-7. Why isn't ether mining an instant money-maker? That has a lot to do with your hardware, as you'll see below.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



```
I1112 22:03:26.071880 eth/backend.go:454] Automatic pregeneration of ethash DAG
ON (ethash dir: /home/ubie/.ethash)
true
> I1112 22:03:26.072245 eth/backend.go:461] checking DAG (ethash dir: /home/ubie
/.ethash)
I1112 22:03:26.072435 miner/worker.go:539] commit new work on block 1748011 with
 0 txs & 0 uncles. Took 623.351µs
I1112 22:03:26.072570 ethash.go:259] Generating DAG for epoch 58 (size 156027865
6) (8f602dc7d86df0a7c8e7467ec0d211062ee85c5c14c6d2f6c025976cf550e8c5)
I1112 22:03:27.548451 ethash.go:291] Generating DAG: 0%
I1112 22:03:33.584568 ethash.go:291] Generating DAG: 1%
I1112 22:03:39.798725 ethash.go:291] Generating DAG: 2%
I1112 22:03:45.891413 ethash.go:291] Generating DAG: 3%
> I1112 22:03:51.758028 ethash.go:291] Generating DAG: 4%
> I1112 22:03:53.465117 eth/downloader/downloader.go:319] Block synchronisation
started
I1112 22:03:53.465561 miner/miner.go:75] Mining operation aborted due to sync op
eration
> I1112 22:03:57.340299 eth/downloader/downloader.go:298] Synchronisation failed
: receipt download canceled (requested)
```

Figure 6-7. The miner gets ready to mine

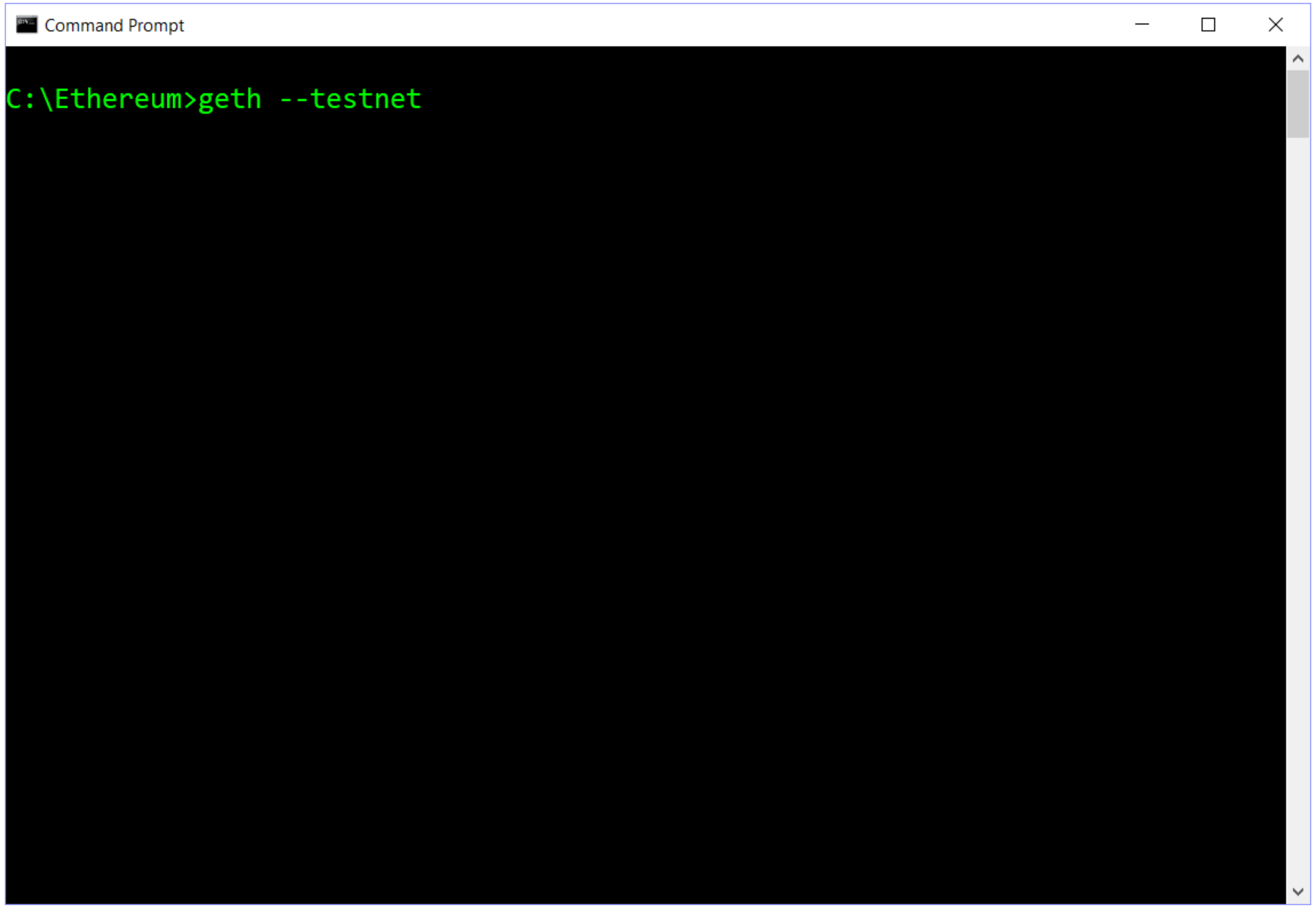
You can stop this process by typing the following:

```
miner.stop()
```

Next, you'll put a personal tag on the blocks you mine, just because.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

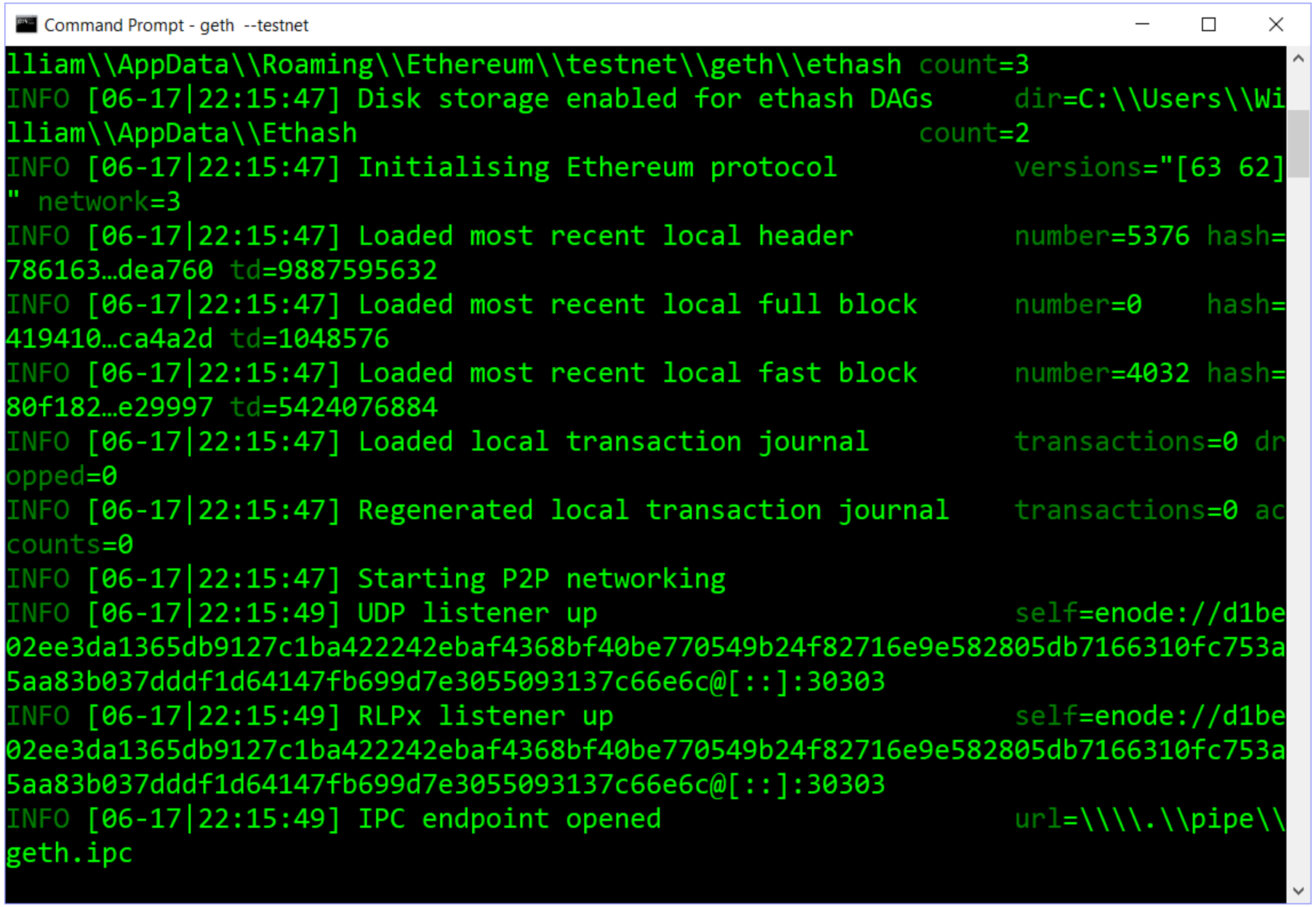
Geth Workout



```
C:\Ethereum>geth --testnet
```

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

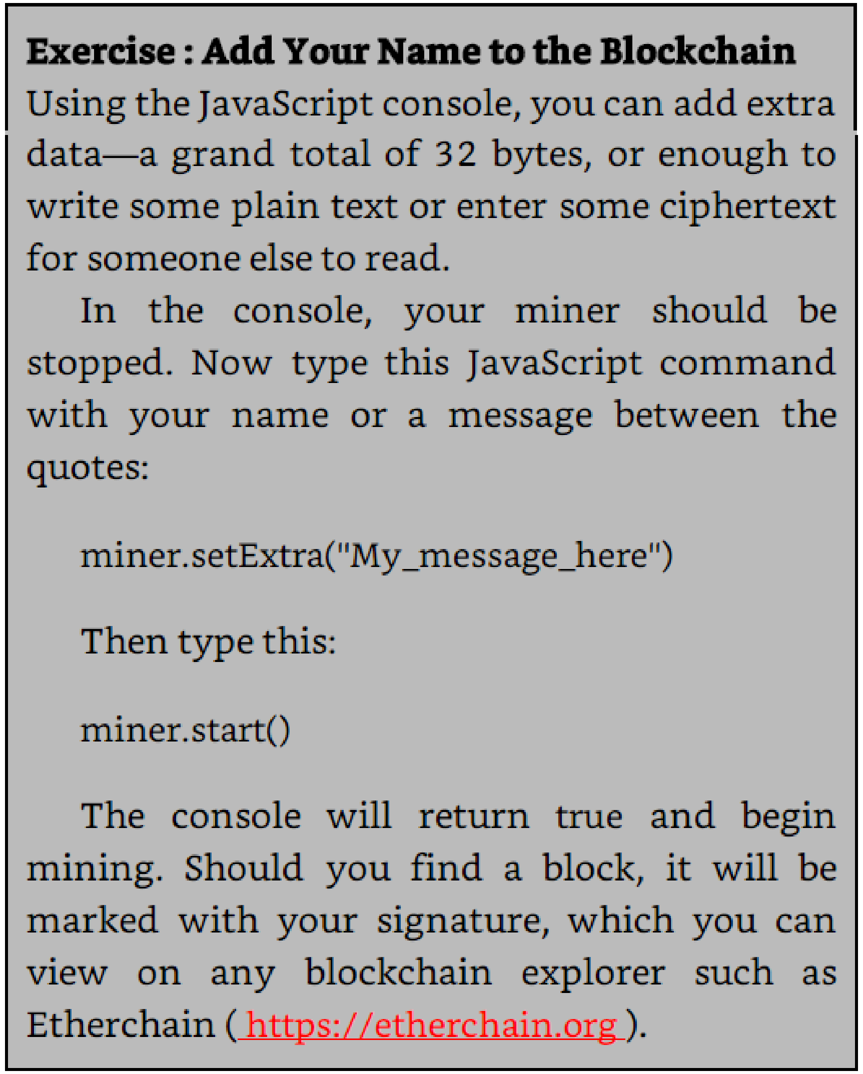
Geth Workout



```
lliam\\AppData\\Roaming\\Ethereum\\testnet\\geth\\ethash count=3
INFO [06-17|22:15:47] Disk storage enabled for ethash DAGs     dir=C:\\Users\\Wi
lliam\\AppData\\Ethash                                   count=2
INFO [06-17|22:15:47] Initialising Ethereum protocol           versions="[63 62]
" network=3
INFO [06-17|22:15:47] Loaded most recent local header          number=5376 hash=
786163…dea760 td=9887595632
INFO [06-17|22:15:47] Loaded most recent local full block      number=0    hash=
419410…ca4a2d td=1048576
INFO [06-17|22:15:47] Loaded most recent local fast block      number=4032 hash=
80f182…e29997 td=5424076884
INFO [06-17|22:15:47] Loaded local transaction journal         transactions=0 dr
opped=0
INFO [06-17|22:15:47] Regenerated local transaction journal    transactions=0 ac
counts=0
INFO [06-17|22:15:47] Starting P2P networking
INFO [06-17|22:15:49] UDP listener up                          self=enode://d1be
02ee3da1365db9127c1ba422242ebaf4368bf40be770549b24f82716e9e582805db7166310fc753a
5aa83b037dddf1d64147fb699d7e3055093137c66e6c@[::]:30303
INFO [06-17|22:15:49] RLPx listener up                         self=enode://d1be
02ee3da1365db9127c1ba422242ebaf4368bf40be770549b24f82716e9e582805db7166310fc753a
5aa83b037dddf1d64147fb699d7e3055093137c66e6c@[::]:30303
INFO [06-17|22:15:49] IPC endpoint opened                      url=\\\\.\\pipe\\
geth.ipc
```

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



Exercise : Add Your Name to the Blockchain

Using the JavaScript console, you can add extra data—a grand total of 32 bytes, or enough to write some plain text or enter some ciphertext for someone else to read.

In the console, your miner should be stopped. Now type this JavaScript command with your name or a message between the quotes:

```
miner.setExtra("My_message_here")
```

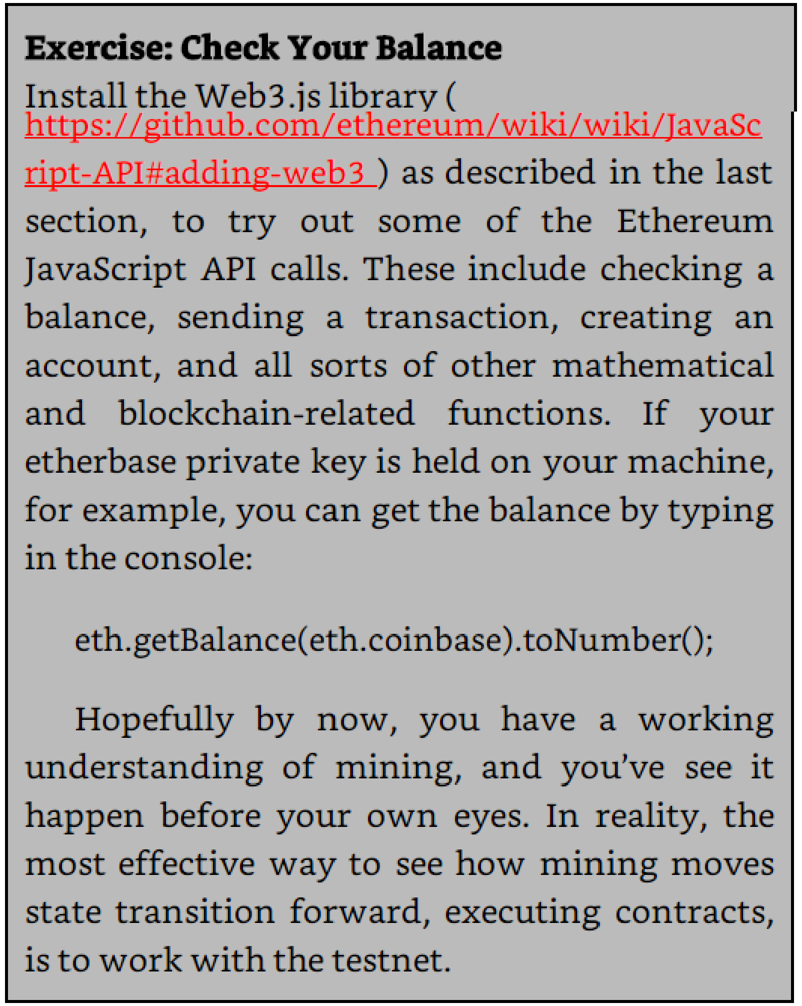
Then type this:

```
miner.start()
```

The console will return true and begin mining. Should you find a block, it will be marked with your signature, which you can view on any blockchain explorer such as Etherchain (https://etherchain.org).

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



Exercise: Check Your Balance

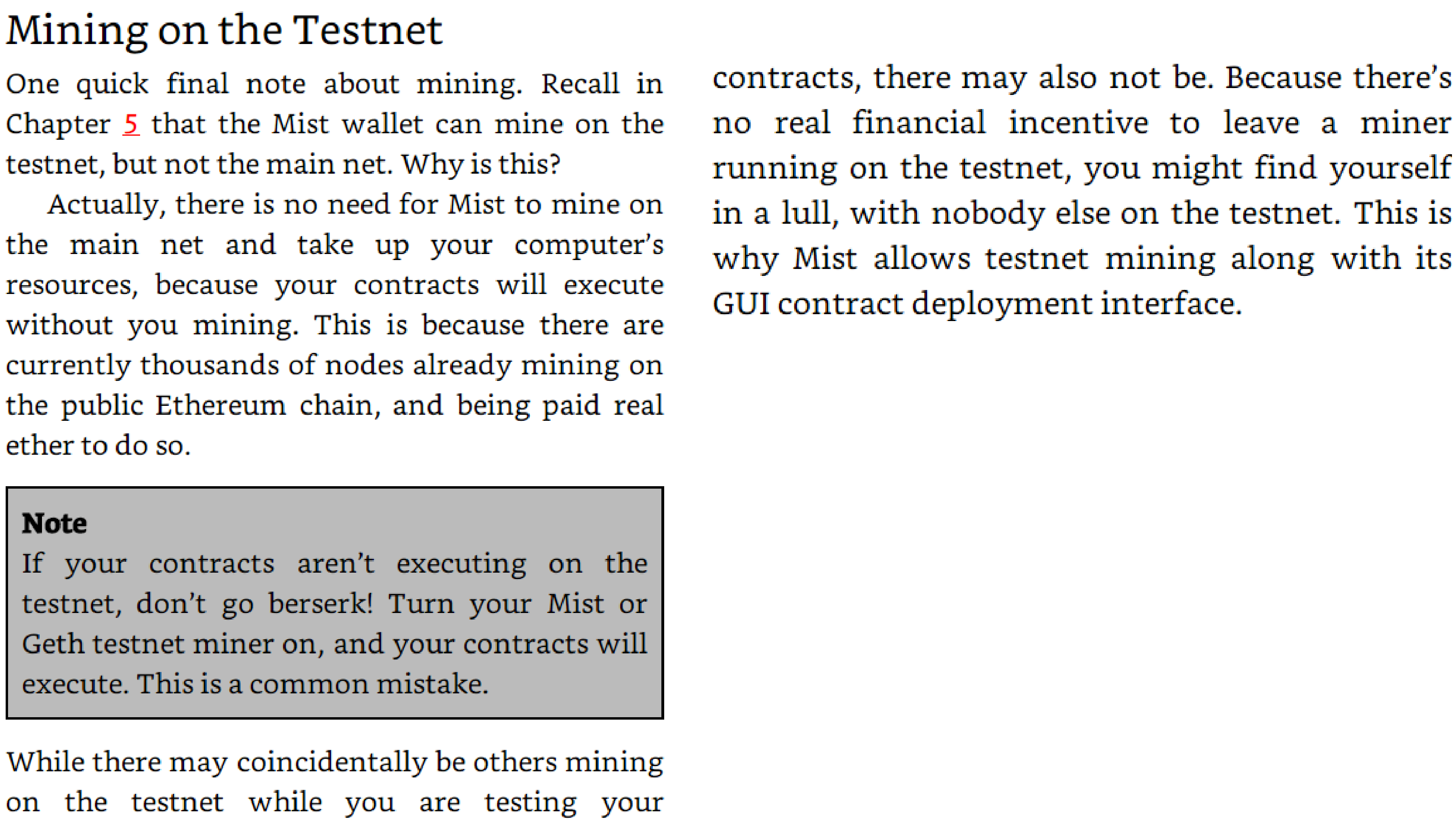
Install the Web3.js library (https://github.com/ethereum/wiki/wiki/JavaScript-API#adding-web3) as described in the last section, to try out some of the Ethereum JavaScript API calls. These include checking a balance, sending a transaction, creating an account, and all sorts of other mathematical and blockchain-related functions. If your etherbase private key is held on your machine, for example, you can get the balance by typing in the console:

```
eth.getBalance(eth.coinbase).toNumber();
```

Hopefully by now, you have a working understanding of mining, and you've see it happen before your own eyes. In reality, the most effective way to see how mining moves state transition forward, executing contracts, is to work with the testnet.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



Mining on the Testnet

One quick final note about mining. Recall in Chapter 5 that the Mist wallet can mine on the testnet, but not the main net. Why is this?

Actually, there is no need for Mist to mine on the main net and take up your computer's resources, because your contracts will execute without you mining. This is because there are currently thousands of nodes already mining on the public Ethereum chain, and being paid real ether to do so.

Note
If your contracts aren't executing on the testnet, don't go berserk! Turn your Mist or Geth testnet miner on, and your contracts will execute. This is a common mistake.

While there may coincidentally be others mining on the testnet while you are testing your contracts, there may also not be. Because there's no real financial incentive to leave a miner running on the testnet, you might find yourself in a lull, with nobody else on the testnet. This is why Mist allows testnet mining along with its GUI contract deployment interface.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Topic 2: Truffle Framework Introduction

What Is Truffle?

Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life as an Ethereum developer easier. It is one of the most widely used IDEs in the Ethereum community. Developers can use it to build and deploy DApps for testing purposes with many features that make it more attractive to users with a Web 3.0 dev background.

Features:

- Automated contract testing with Mocha and Chai.
- A configurable build pipeline that supports both web apps and console apps.
- Generators for creating new contracts and tests (like rails generate)
- Instant rebuilding of assets during development (truffle watch)
- Console to easily work with your compiled contracts (truffle console)
- Script runner that lets you run JS/Coffee files with your contracts included (truffle exec)
- Contract compilation and deployment using the RPC client of your choice.
- Support for JavaScript, CoffeeScript, SASS, ES6 and JSX built-in.

Free at https://truffleframework.com/truffle

Free tutorials also.

Source: https://ethereum.stackexchange.com/questions/1030/what-is-truffle

Why Truffle?

Rapid Dapp development and DApp software assembly

- Generators for creating new contracts and tests (like rails generate)
- Instant rebuilding of assets during development (truffle watch)
- Console to easily work with your compiled contracts (truffle console)
- Script runner that lets you run JS/Coffee files with your contracts included (truffle exec)
- Contract compilation and deployment using the RPC client of your choice.
- Support for JavaScript, CoffeeScript, SASS, ES6 and JSX built-in.

Setting up Truffle

Three Options:

1. Download and install the version for your Windows Operating System
   - ❑ https://truffleframework.com/docs/truffle/getting-started/installation
2. Download and install the version for your Linux Operating System
   - ❑ https://medium.com/@techgeek628/how-to-install-and-execute-truffle-on-an-ubuntu-16-04-7ebb3444707e
3. Install VMWare Workstation for your Operating System and add and configure it with an Ubuntu image that already has a) NodeJS; b) git; c) Ethereum Client, MetaMask, and Truffle for Linux all installed. The images for VMWare Workstation and the Ubuntu VM are at https://tinyurl.com/y46paxkg |

00 Day 02 Materials | 00 VMWare Workstation Images

00 Day 02 Materials | 00 Blockchain Dev Platforms | 00 Ubuntu VM

Install Metamask

Metamask will become your Ethereum “Wallet” for your Smart Contract and DApp development activities. It will store your Ether, and your public and private keys.

Use your keypair from the provided list in this class.

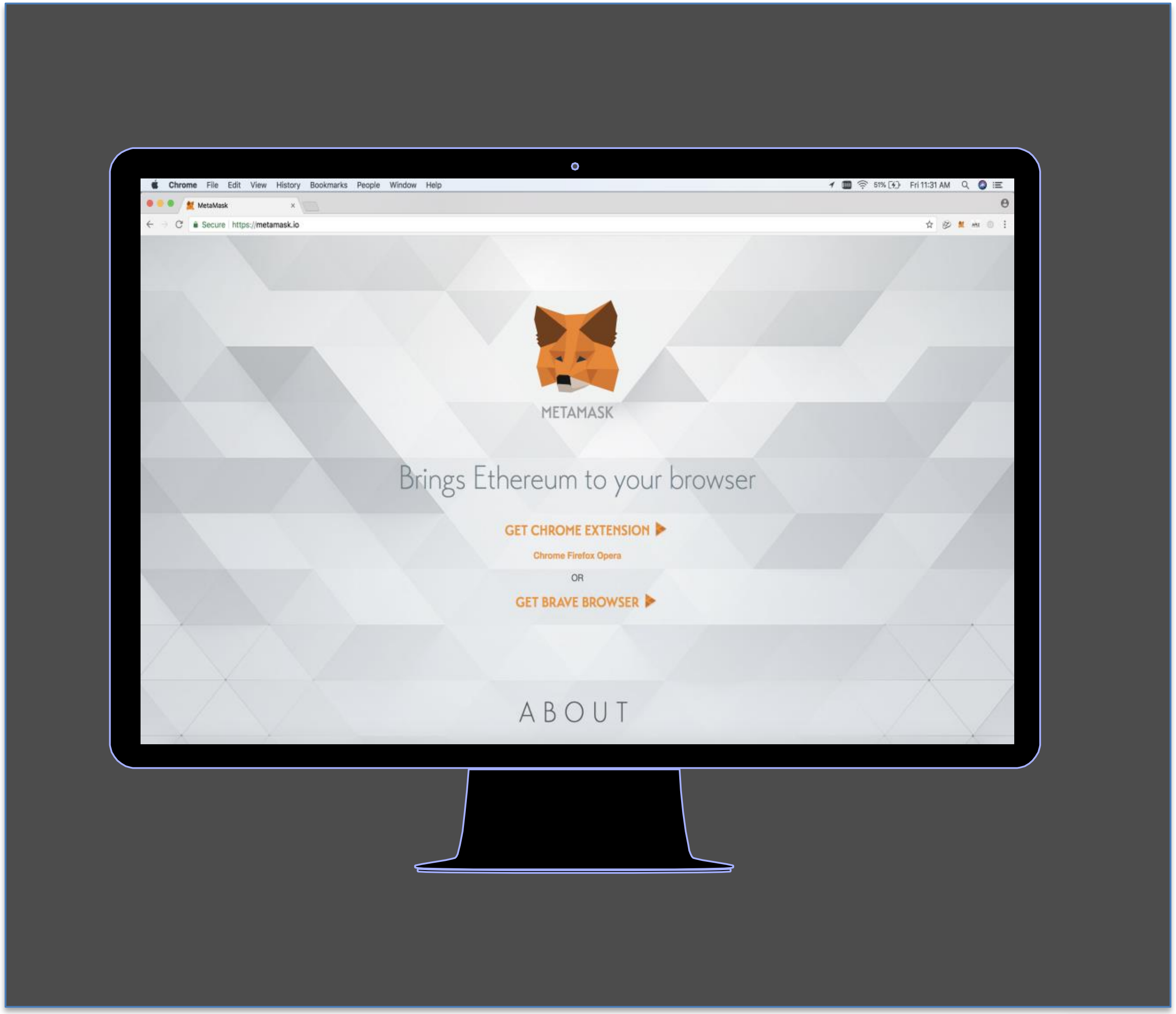
Go to https://metamask.io

Install Metamask as a Browser plugin, and provide your public and private key as well as a password you decide you want to use.

Accept ToS

Create Password

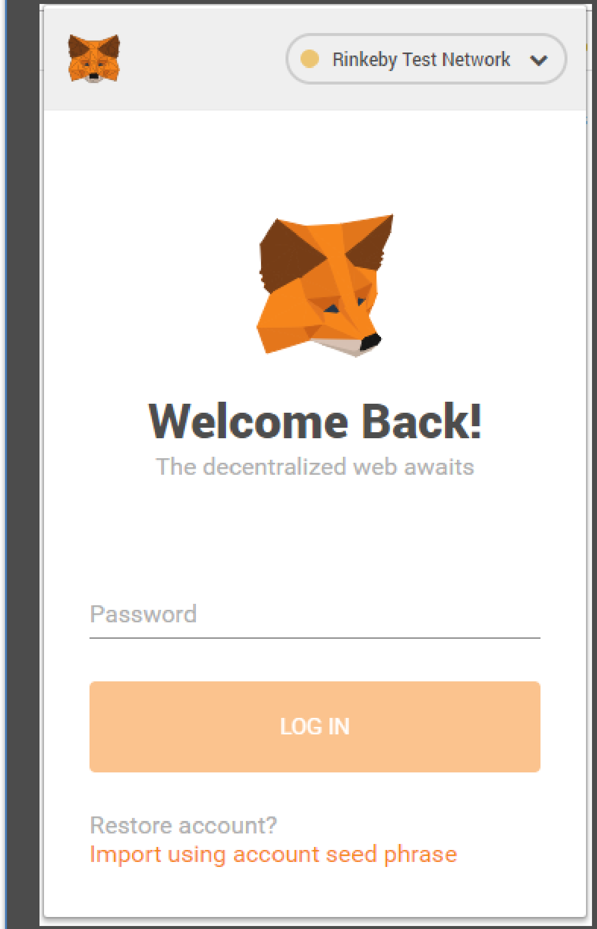
Save Seed


Chrome File Edit View History Bookmarks People Window Help
MetaMask
Secure https://metamask.io
METAMASK
Brings Ethereum to your browser
GET CHROME EXTENSION
Chrome Firefox Opera
OR
GET BRAVE BROWSER
ABOUT

Startup Metamask

Metamask will become your Ethereum "Wallet" for your Smart Contract and DApp development activities. It will store your Ether, and your public and private keys.

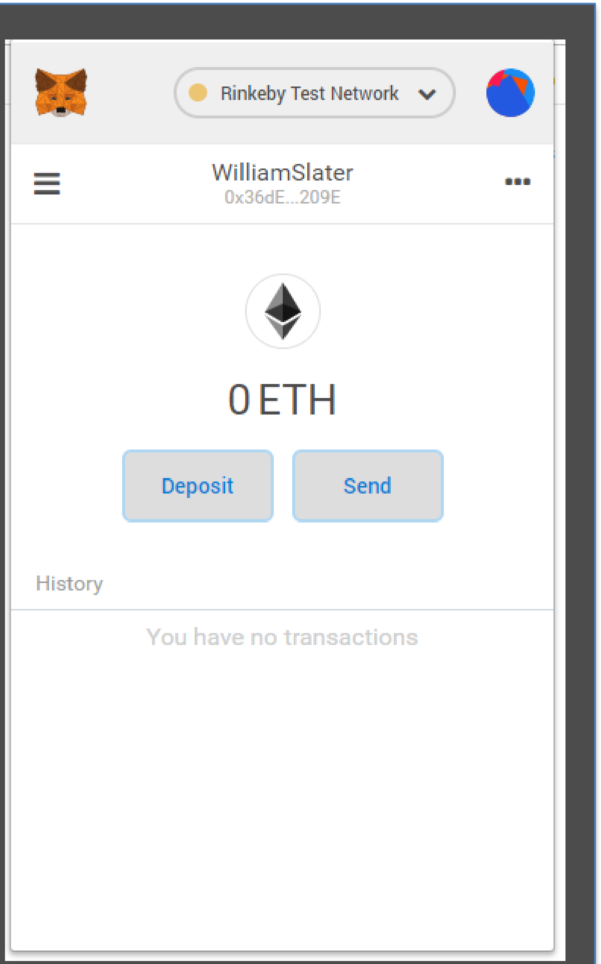
To start up Metamask, click on the small Foxhead icon on the upper right.



Welcome Back!
The decentralized web awaits

Restore account?
Import using account seed phrase

Before Login



WilliamSlater
0x36dE...209E

0 ETH

You have no transactions

After Login

Installing Truffle on Wintel

First install MetaMask and use the Public and Private Key Pairs Provided (write down your password and seed phrase)

To install Truffle for Wintel

Visit http://www.chocolatey.org and install Chocolatey for Windows

In a new Powershell Window, running with Administrator Privileges

choco install nodejs.install –y

choco install git –y

choco VisualStudioCode –y #optional

In a new Powershell Window, running with Administrator Privileges

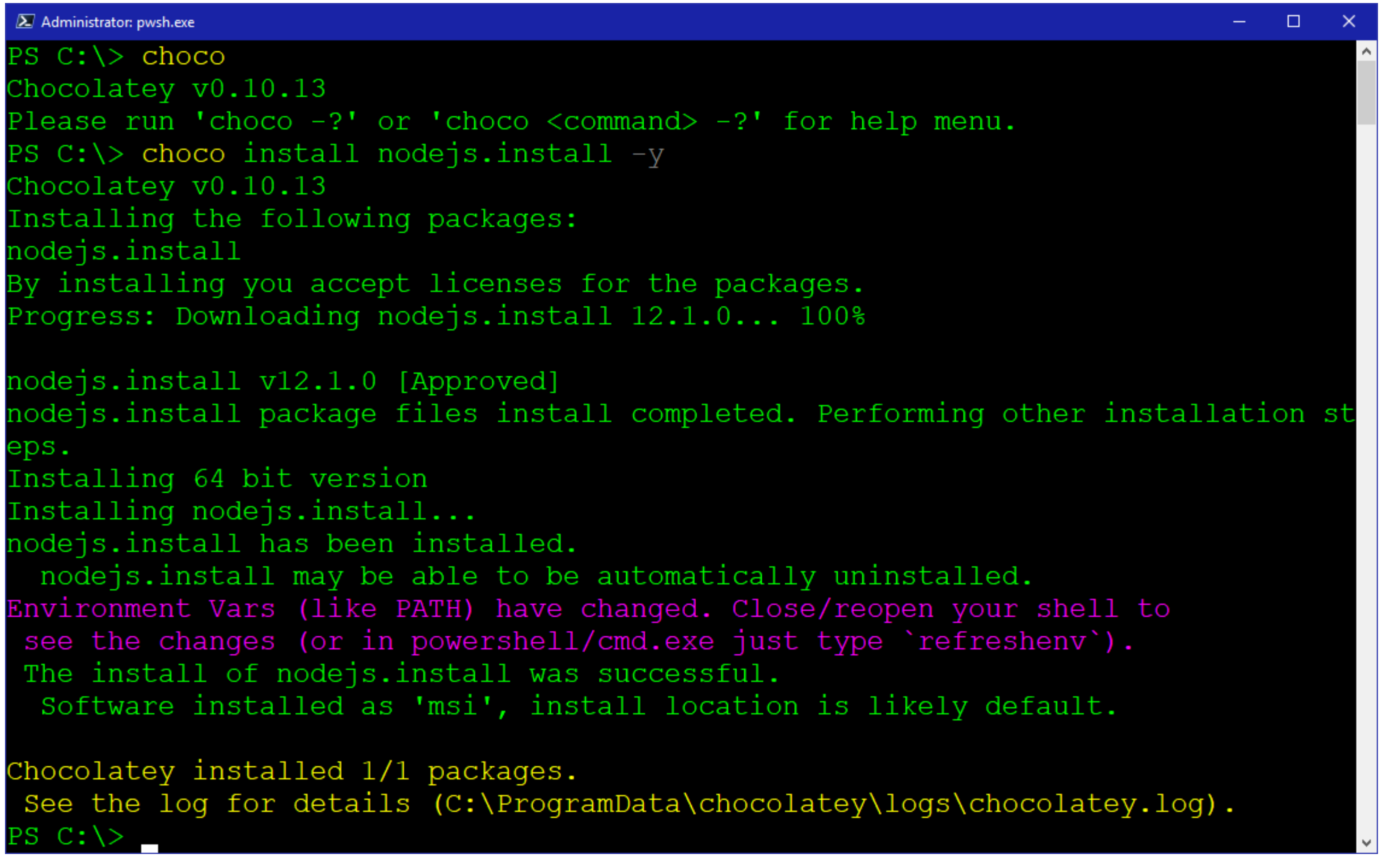
npm install -g npm

npm install -g --production windows-build-tools

npm install -g ethereumjs-testrpc truffle

Installing Truffle – First, install Chocolaty On Wintel

Download Chocolatey from http://chocolatey.org
Install it
Go to Powershell with Admin privilege and Run Choco to install NodeJS



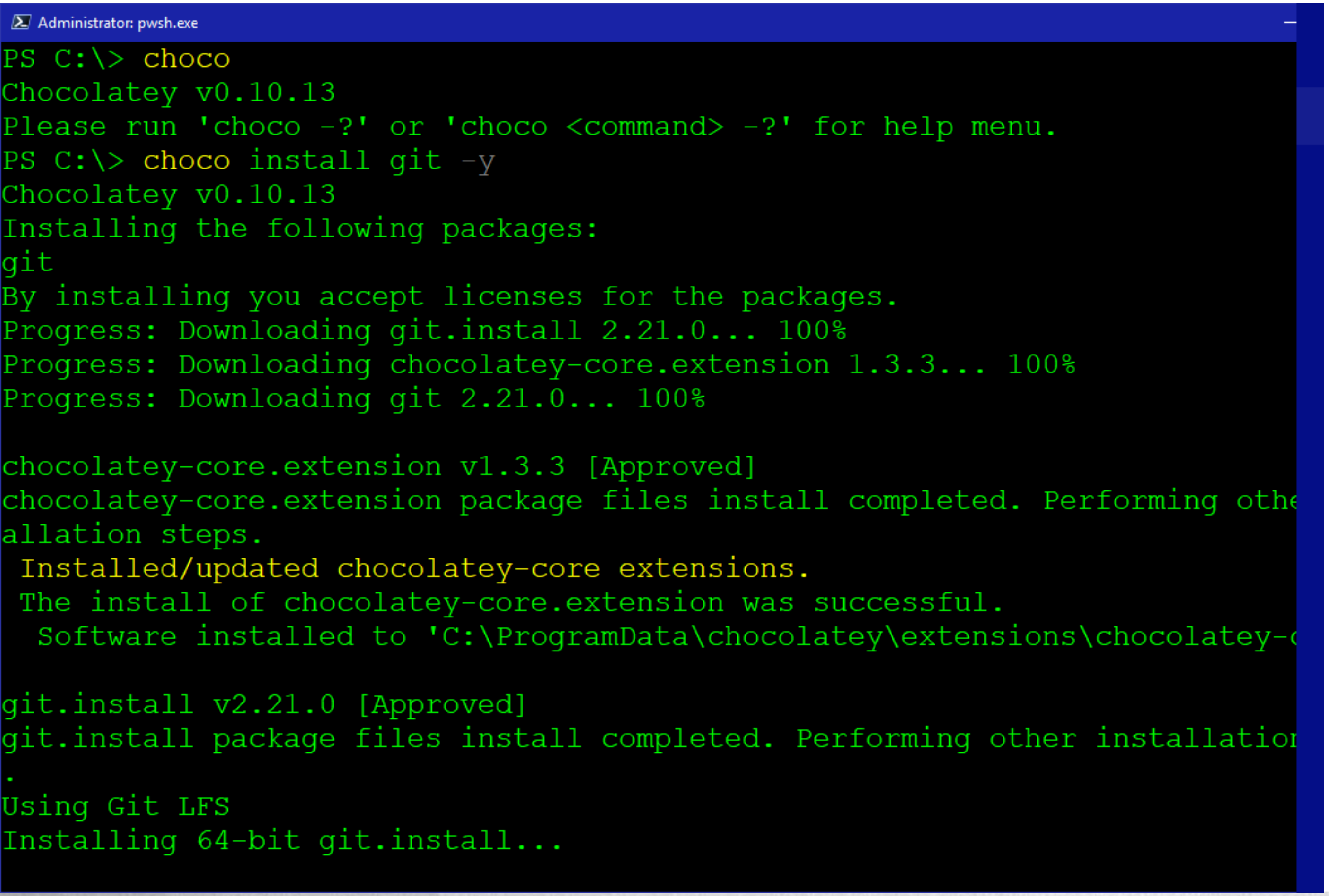
```
PS C:\> choco
Chocolatey v0.10.13
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\> choco install nodejs.install -y
Chocolatey v0.10.13
Installing the following packages:
nodejs.install
By installing you accept licenses for the packages.
Progress: Downloading nodejs.install 12.1.0... 100%

nodejs.install v12.1.0 [Approved]
nodejs.install package files install completed. Performing other installation st
eps.
Installing 64 bit version
Installing nodejs.install...
nodejs.install has been installed.
  nodejs.install may be able to be automatically uninstalled.
Environment Vars (like PATH) have changed. Close/reopen your shell to
 see the changes (or in powershell/cmd.exe just type `refreshenv`).
 The install of nodejs.install was successful.
  Software installed as 'msi', install location is likely default.

Chocolatey installed 1/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
PS C:\>
```

Install Chocolatey via https://chocolatey.org/
Open a PowerShell prompt as Administrator

choco install nodejs.install -y
choco install git -y
choco install VisualStudioCode -y
#optional

Chocolaty On Wintel

Go to Powershell with Admin privilege and Run Choco to install Git

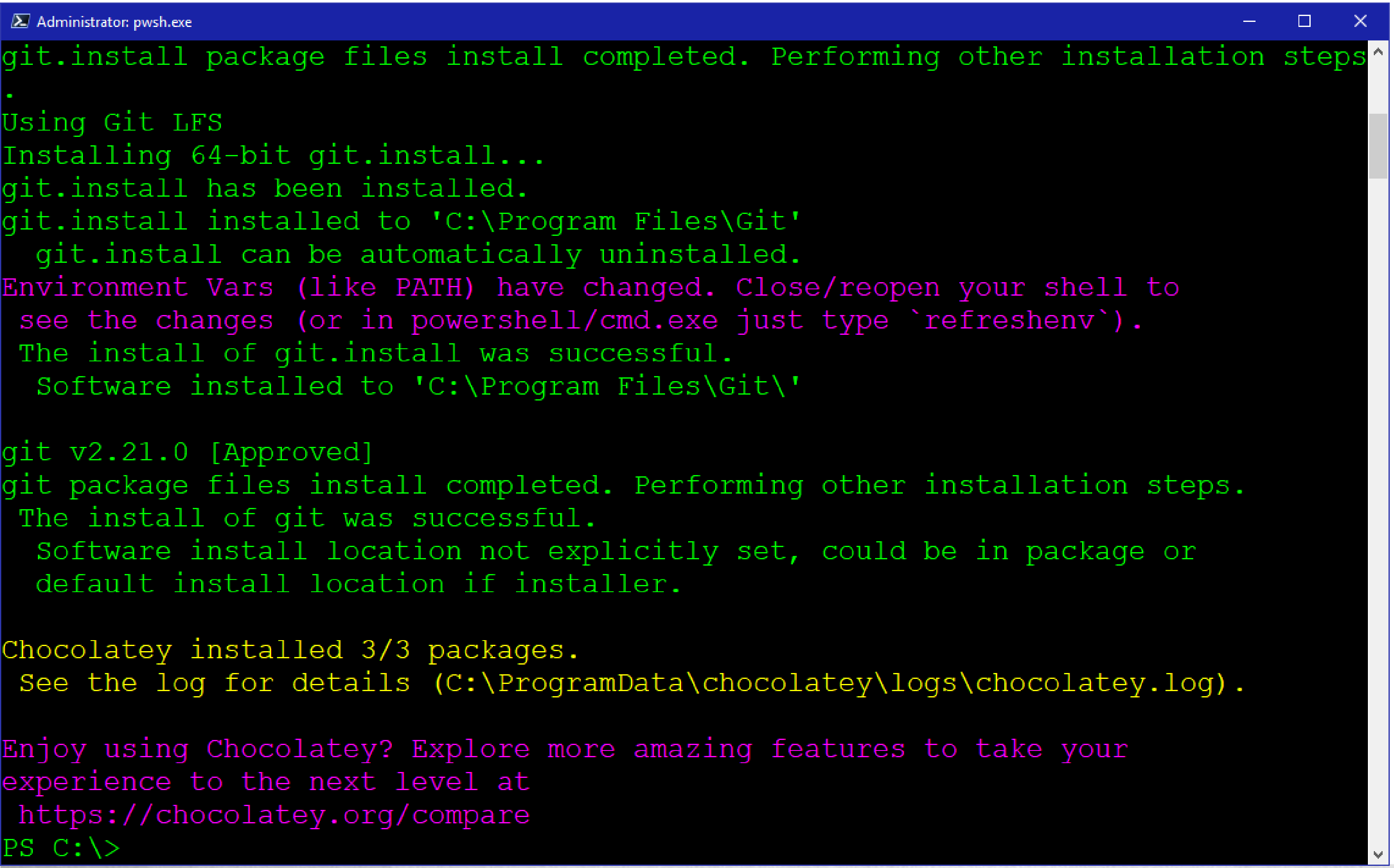


```
PS C:\> choco
Chocolatey v0.10.13
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\> choco install git -y
Chocolatey v0.10.13
Installing the following packages:
git
By installing you accept licenses for the packages.
Progress: Downloading git.install 2.21.0... 100%
Progress: Downloading chocolatey-core.extension 1.3.3... 100%
Progress: Downloading git 2.21.0... 100%

chocolatey-core.extension v1.3.3 [Approved]
chocolatey-core.extension package files install completed. Performing othe
allation steps.
 Installed/updated chocolatey-core extensions.
 The install of chocolatey-core.extension was successful.
  Software installed to 'C:\ProgramData\chocolatey\extensions\chocolatey-c

git.install v2.21.0 [Approved]
git.install package files install completed. Performing other installatio
.
Using Git LFS
Installing 64-bit git.install...
```

choco install git -y

Chocolaty On Wintel

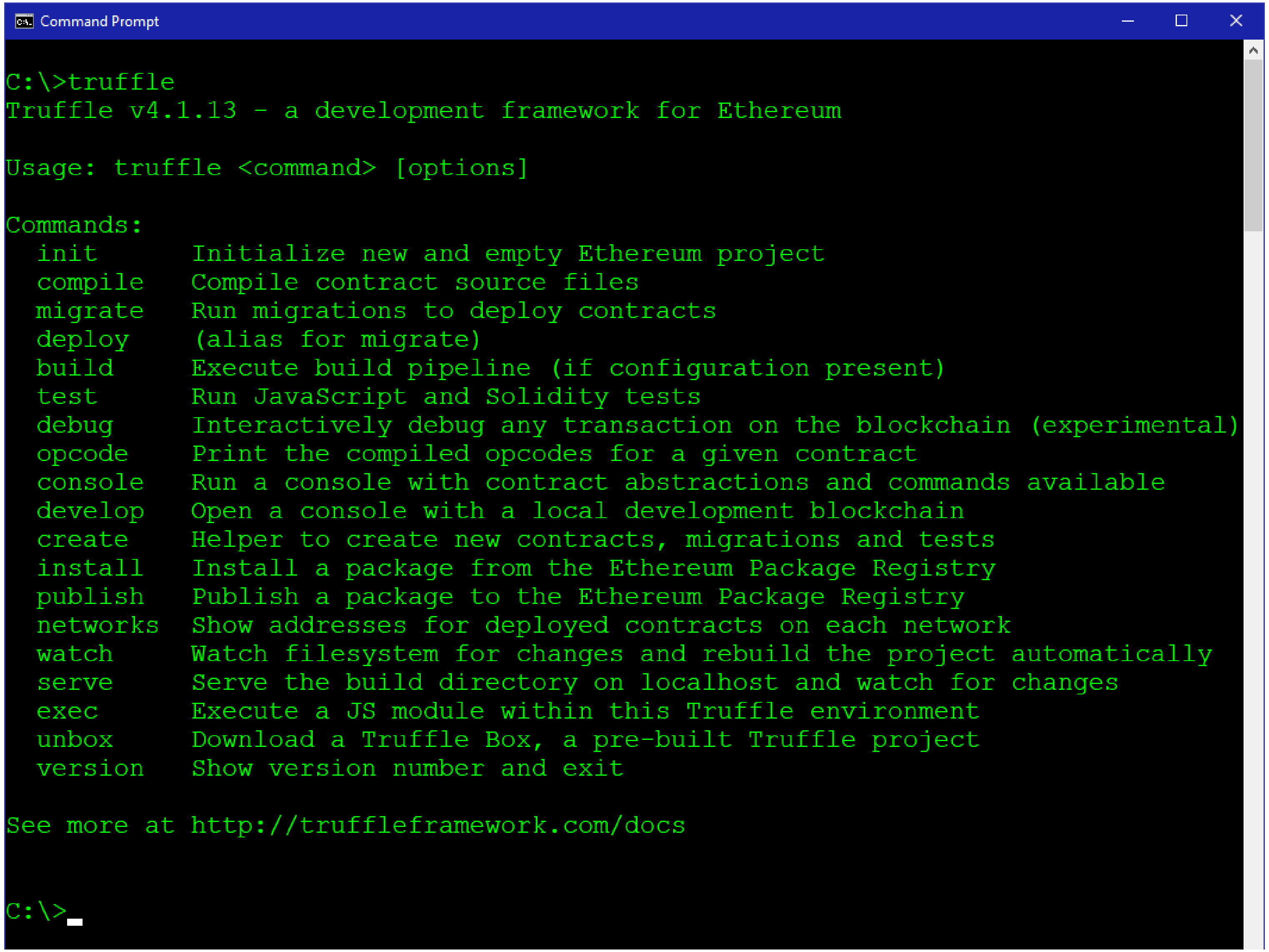
Go to Powershell with Admin privilege and Run Choco to install the Visual Studio Add-ins



```
git.install package files install completed. Performing other installation steps
.
Using Git LFS
Installing 64-bit git.install...
git.install has been installed.
git.install installed to 'C:\Program Files\Git'
  git.install can be automatically uninstalled.
Environment Vars (like PATH) have changed. Close/reopen your shell to
 see the changes (or in powershell/cmd.exe just type `refreshenv`).
 The install of git.install was successful.
  Software installed to 'C:\Program Files\Git\'

git v2.21.0 [Approved]
git package files install completed. Performing other installation steps.
 The install of git was successful.
  Software install location not explicitly set, could be in package or
  default install location if installer.

Chocolatey installed 3/3 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
 https://chocolatey.org/compare
PS C:\>
```

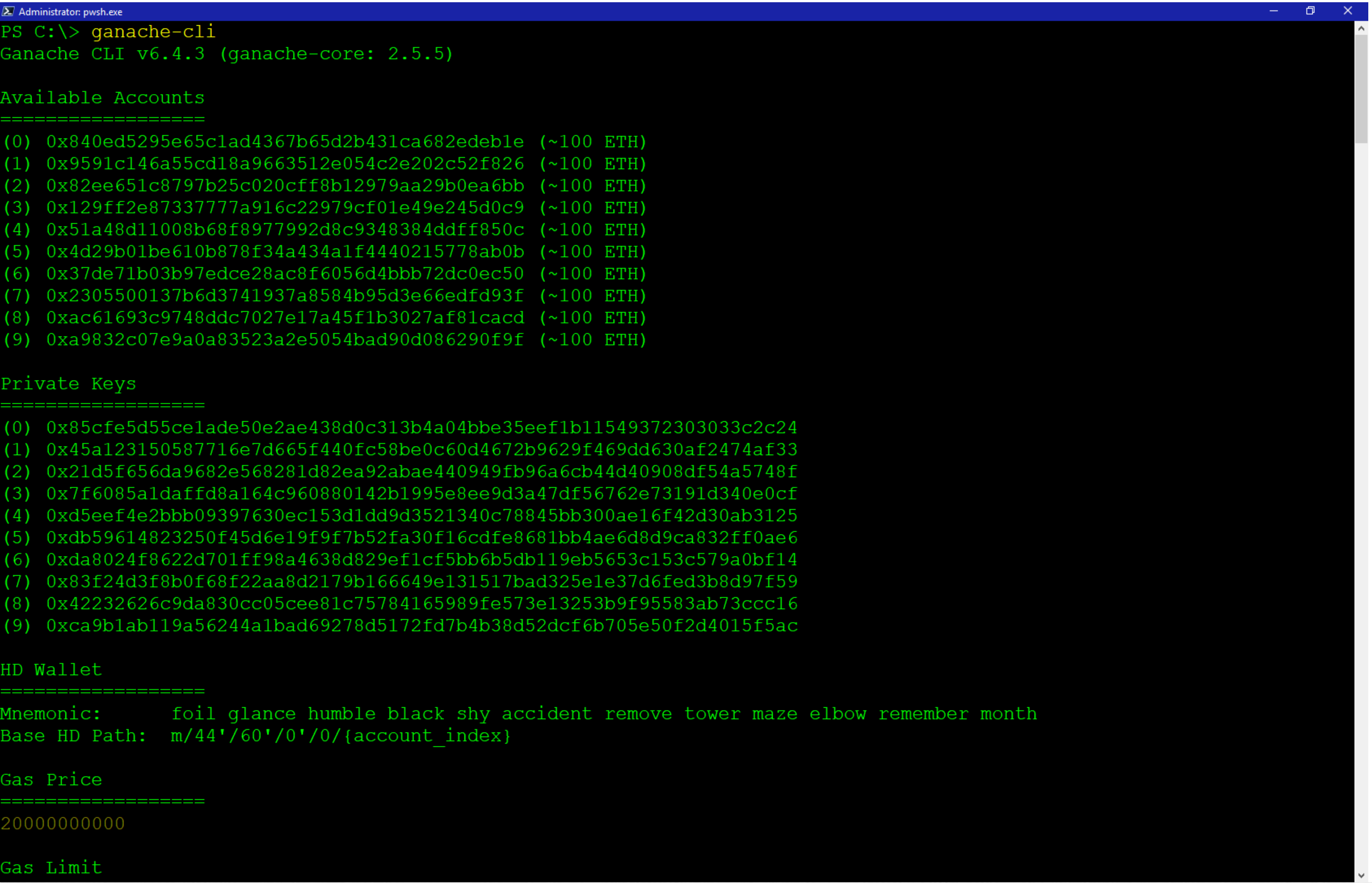
choco install VisualStudioCode -y #optional

Truffle Commands On Wintel



```
C:\>truffle
Truffle v4.1.13 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  init      Initialize new and empty Ethereum project
  compile   Compile contract source files
  migrate   Run migrations to deploy contracts
  deploy    (alias for migrate)
  build     Execute build pipeline (if configuration present)
  test      Run JavaScript and Solidity tests
  debug     Interactively debug any transaction on the blockchain (experimental)
  opcode    Print the compiled opcodes for a given contract
  console   Run a console with contract abstractions and commands available
  develop   Open a console with a local development blockchain
  create    Helper to create new contracts, migrations and tests
  install   Install a package from the Ethereum Package Registry
  publish   Publish a package to the Ethereum Package Registry
  networks  Show addresses for deployed contracts on each network
  watch     Watch filesystem for changes and rebuild the project automatically
  serve     Serve the build directory on localhost and watch for changes
  exec      Execute a JS module within this Truffle environment
  unbox     Download a Truffle Box, a pre-built Truffle project
  version   Show version number and exit

See more at http://truffleframework.com/docs


C:\>_
```

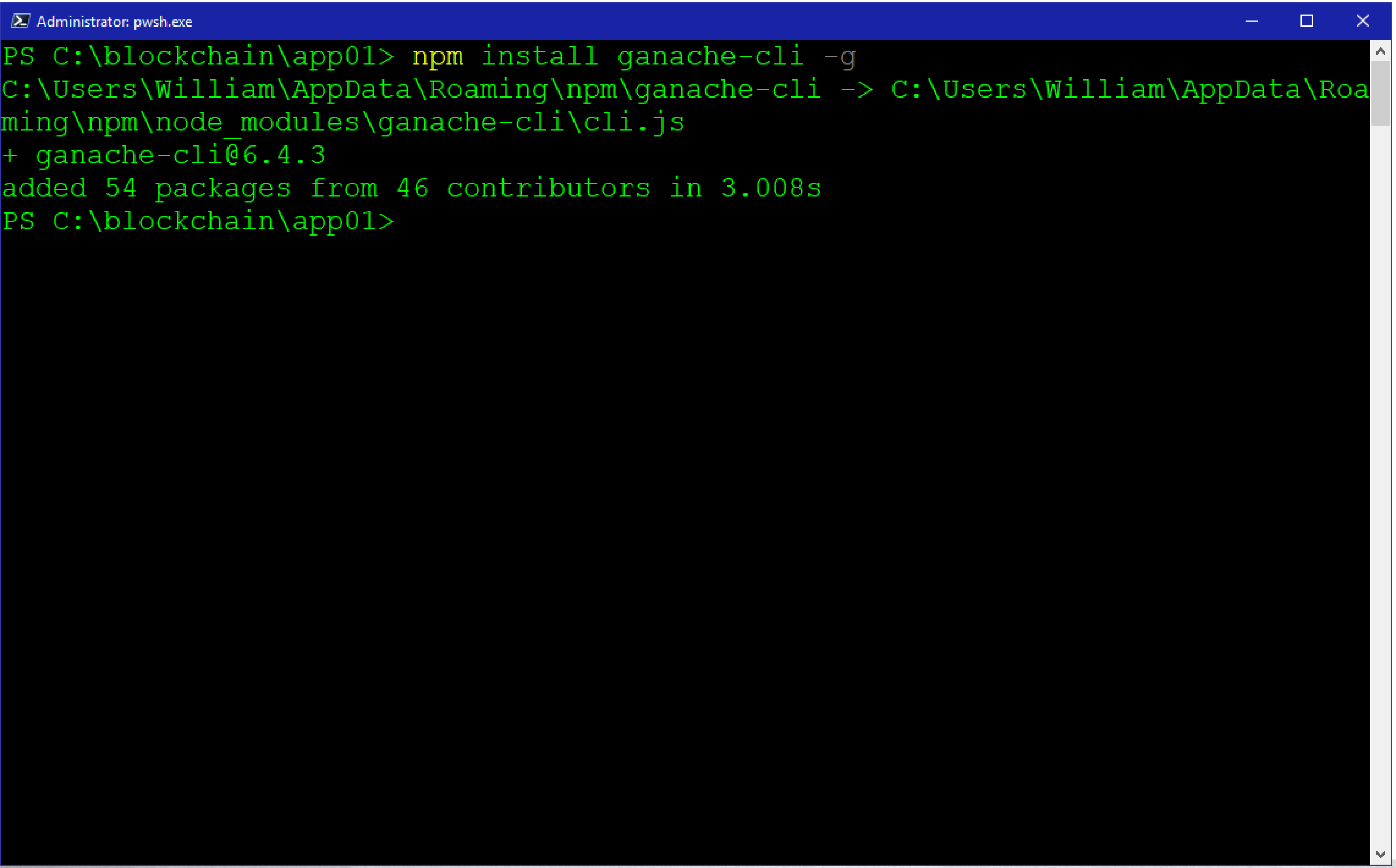
Ganache On Wintel



```
PS C:\> ganache-cli
Ganache CLI v6.4.3 (ganache-core: 2.5.5)

Available Accounts
==================
(0) 0x840ed5295e65c1ad4367b65d2b431ca682edeb1e (~100 ETH)
(1) 0x9591c146a55cd18a9663512e054c2e202c52f826 (~100 ETH)
(2) 0x82ee651c8797b25c020cff8b12979aa29b0ea6bb (~100 ETH)
(3) 0x129ff2e87337777a916c22979cf01e49e245d0c9 (~100 ETH)
(4) 0x51a48d11008b68f8977992d8c9348384ddff850c (~100 ETH)
(5) 0x4d29b01be610b878f34a434a1f4440215778ab0b (~100 ETH)
(6) 0x37de71b03b97edce28ac8f6056d4bbb72dc0ec50 (~100 ETH)
(7) 0x2305500137b6d3741937a8584b95d3e66edfd93f (~100 ETH)
(8) 0xac61693c9748ddc7027e17a45f1b3027af81cacd (~100 ETH)
(9) 0xa9832c07e9a0a83523a2e5054bad90d086290f9f (~100 ETH)

Private Keys
==================
(0) 0x85cfe5d55ce1ade50e2ae438d0c313b4a04bbe35eef1b11549372303033c2c24
(1) 0x45a123150587716e7d665f440fc58be0c60d4672b9629f469dd630af2474af33
(2) 0x21d5f656da9682e568281d82ea92abae440949fb96a6cb44d40908df54a5748f
(3) 0x7f6085a1daffd8a164c960880142b1995e8ee9d3a47df56762e73191d340e0cf
(4) 0xd5eef4e2bbb09397630ec153d1dd9d3521340c78845bb300ae16f42d30ab3125
(5) 0xdb59614823250f45d6e19f9f7b52fa30f16cdfe8681bb4ae6d8d9ca832ff0ae6
(6) 0xda8024f8622d701ff98a4638d829ef1cf5bb6b5db119eb5653c153c579a0bf14
(7) 0x83f24d3f8b0f68f22aa8d2179b166649e131517bad325e1e37d6fed3b8d97f59
(8) 0x42232626c9da830cc05cee81c75784165989fe573e13253b9f95583ab73ccc16
(9) 0xca9b1ab119a56244a1bad69278d5172fd7b4b38d52dcf6b705e50f2d4015f5ac

HD Wallet
==================
Mnemonic:      foil glance humble black shy accident remove tower maze elbow remember month
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
```

ganache-cli

Ganache On Wintel

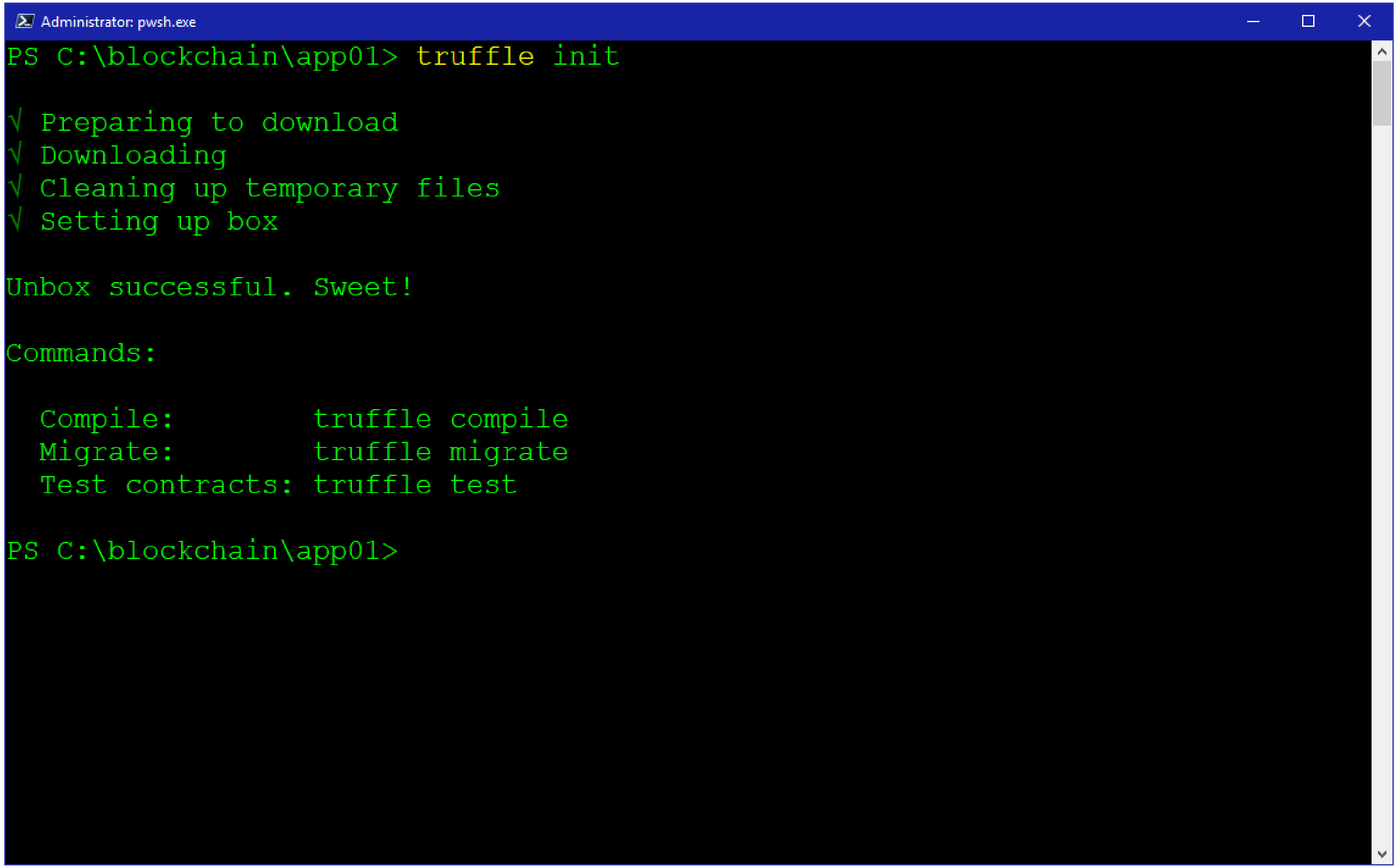


```
PS C:\blockchain\app01> npm install ganache-cli -g
C:\Users\William\AppData\Roaming\npm\ganache-cli -> C:\Users\William\AppData\Roaming\npm\node_modules\ganache-cli\cli.js
+ ganache-cli@6.4.3
added 54 packages from 46 contributors in 3.008s
PS C:\blockchain\app01>
```

npm install ganache-cli -g

Truffle On Wintel

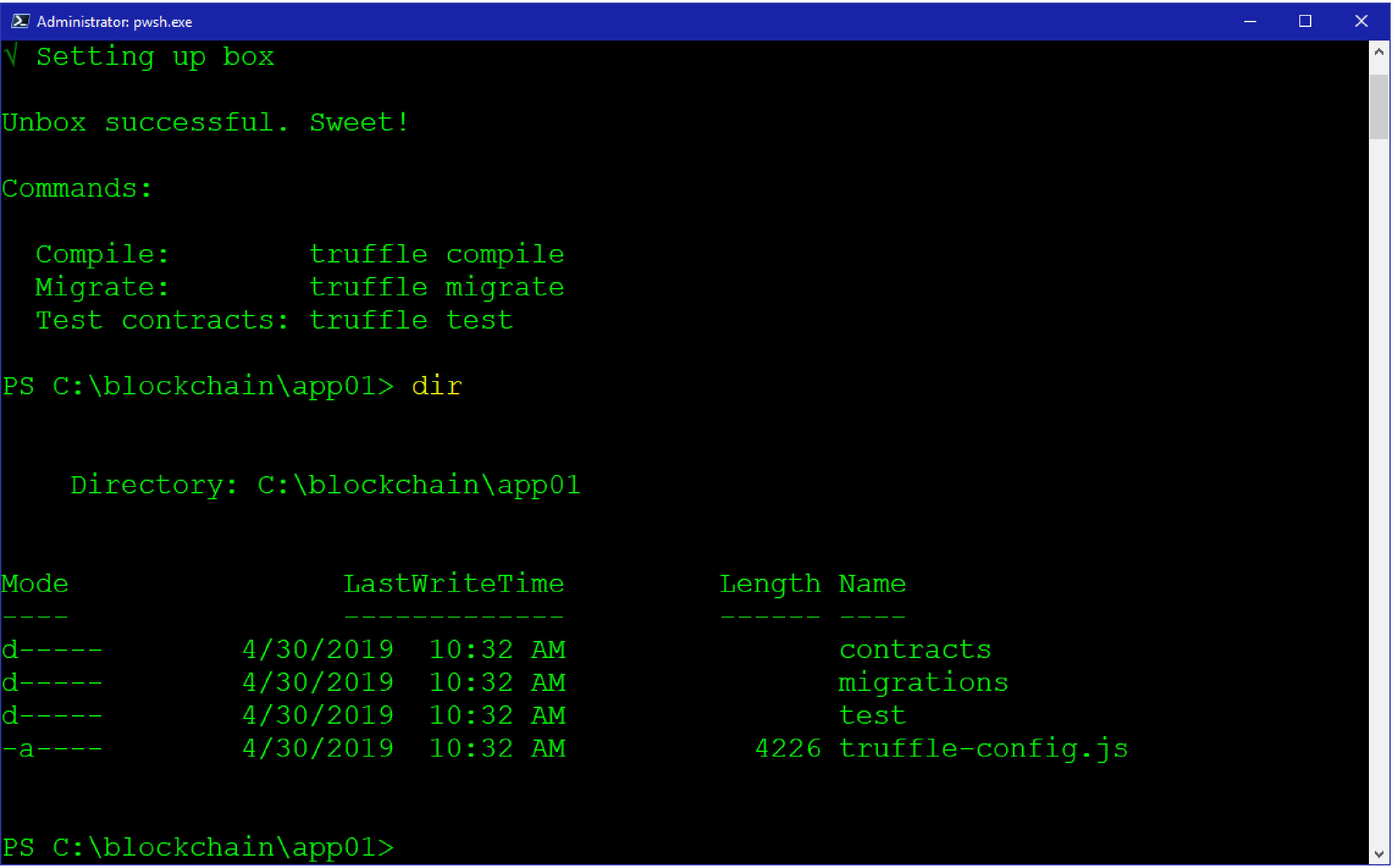
Create a directory structure something like c:\blockchain\app01
Go to Powershell with Admin privilege, navigate to that directory
and type **truffle init**, and you should see this:



```
PS C:\blockchain\app01> truffle init

√ Preparing to download
√ Downloading
√ Cleaning up temporary files
√ Setting up box

Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test

PS C:\blockchain\app01>
```

Truffle On Wintel

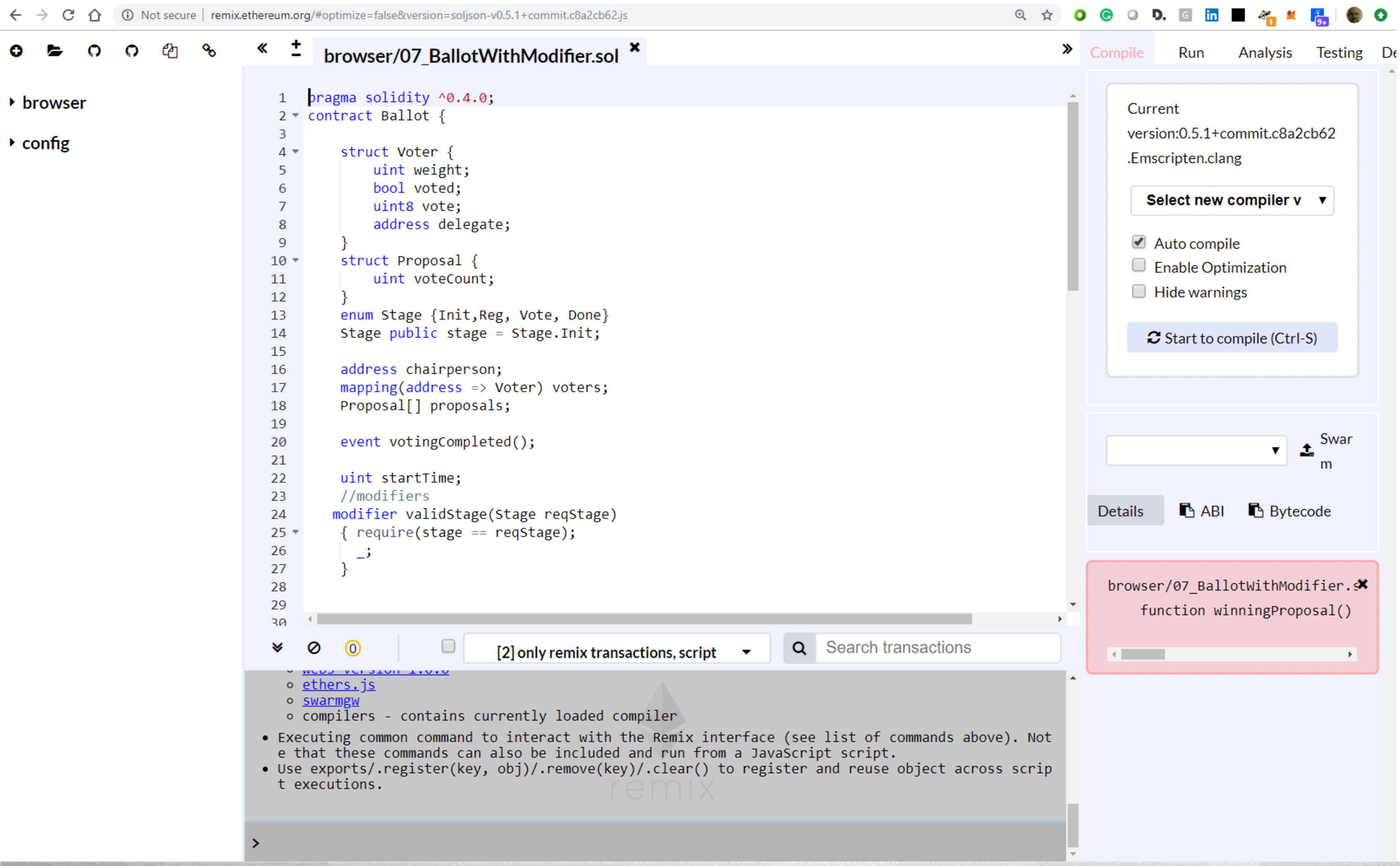
After all that, type **dir** you should see this:



```
√ Setting up box

Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test

PS C:\blockchain\app01> dir


    Directory: C:\blockchain\app01


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        4/30/2019  10:32 AM                contracts
d-----        4/30/2019  10:32 AM                migrations
d-----        4/30/2019  10:32 AM                test
-a----        4/30/2019  10:32 AM           4226 truffle-config.js


PS C:\blockchain\app01>
```

Remix Tips

Remix Compiler – located at http://remix.ethereum.org


Not secure | remix.ethereum.org/#optimize=false&version=soljson-v0.5.1+commit.c8a2cb62.js
browser
config
browser/07_BallotWithModifier.sol
Compile
Run
Analysis
Testing
De
[2] only remix transactions, script
Search transactions
remix

```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }
    enum Stage {Init,Reg, Vote, Done}
    Stage public stage = Stage.Init;

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    event votingCompleted();

    uint startTime;
    //modifiers
   modifier validStage(Stage reqStage)
    { require(stage == reqStage);
      _;
    }
```

```
    o web3 version 1.0.0
    o ethers.js
    o swarmgw
    o compilers - contains currently loaded compiler
 • Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
 • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Current version:0.5.1+commit.c8a2cb62.Emscripten.clang

Select new compiler v

☑ Auto compile
☐ Enable Optimization
☐ Hide warnings

Start to compile (Ctrl-S)

Swarm

Details | ABI | Bytecode

browser/07_BallotWithModifier.s
function winningProposal()

Free
IDE for
Solidity
With access to
Scores of
Different
Versions of
the Solidity
Compiler
With Debugger

Remix Compiler – Best Practices

- Ensure that you use the *correct version* of the Solidity Compiler
- If you are working in multiple directories, ensure you have the correct source code file version
- Pay attention to Remix Static Analysis
- Pay attention to remix Console Details (including Details on ABI, Bytecode, Errors, etc.)
- Review compile details
- Use the Remix Transaction Log for Debugging

DApp Creation Steps (Under Unix & Linux)

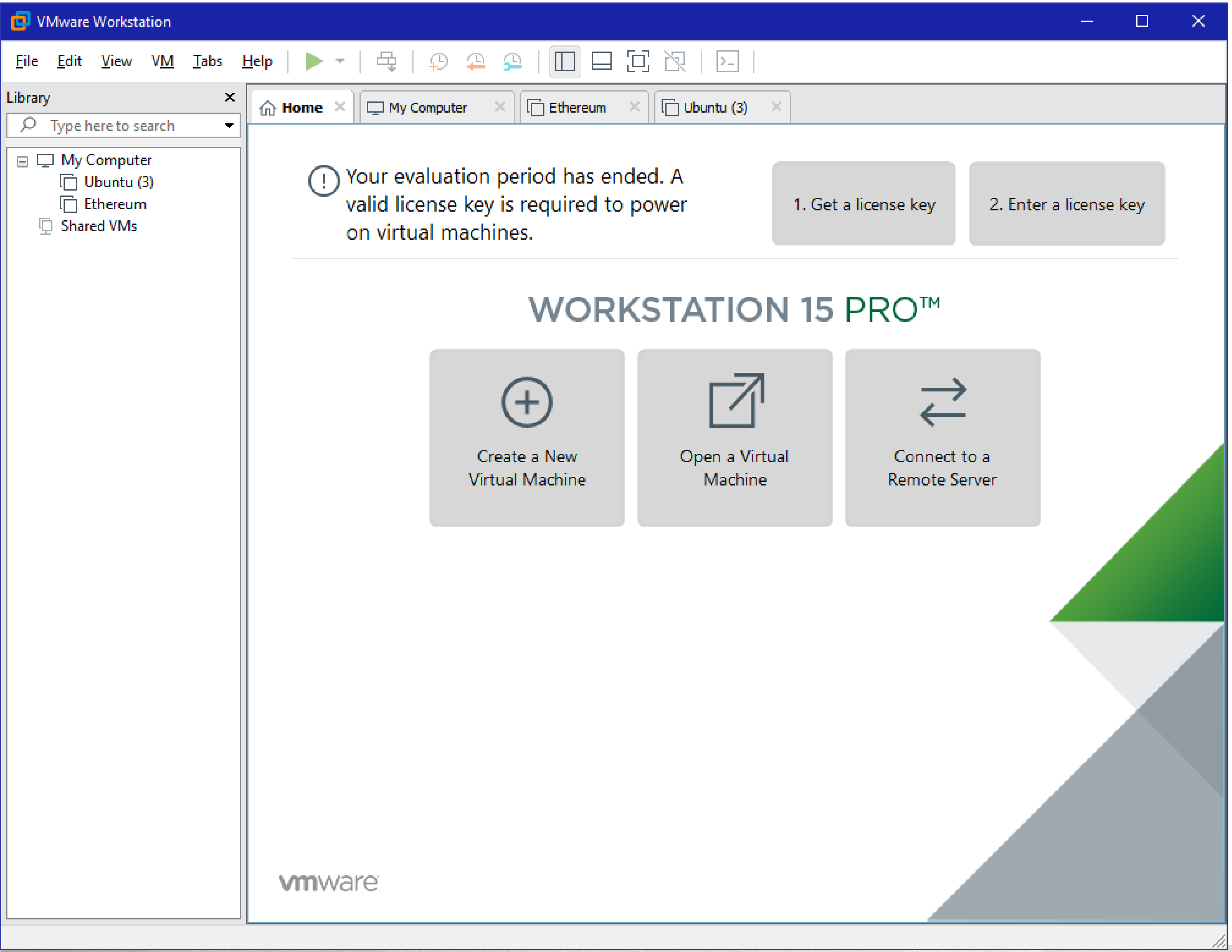
(Assume a good Internet Connection and competence at the Command Line)

1. Install Home Brew
2. Install Node.js
3. Install Git
4. Install TestRPC
5. Install Truffle
6. Optional but Recommended: Install Geth
7. Design and Create a User Interface in HTML
8. Design and Create Smart Contracts using Solidity and/or Javascript
9. Test
10. Deploy

Source: Moses Sam Paul. https://medium.com/heptagon/step-by-step-guide-to-build-a-dapp-a-homo-sapiens-2-day-love-affair-with-ethereum-dapp-de2b0dea12f1

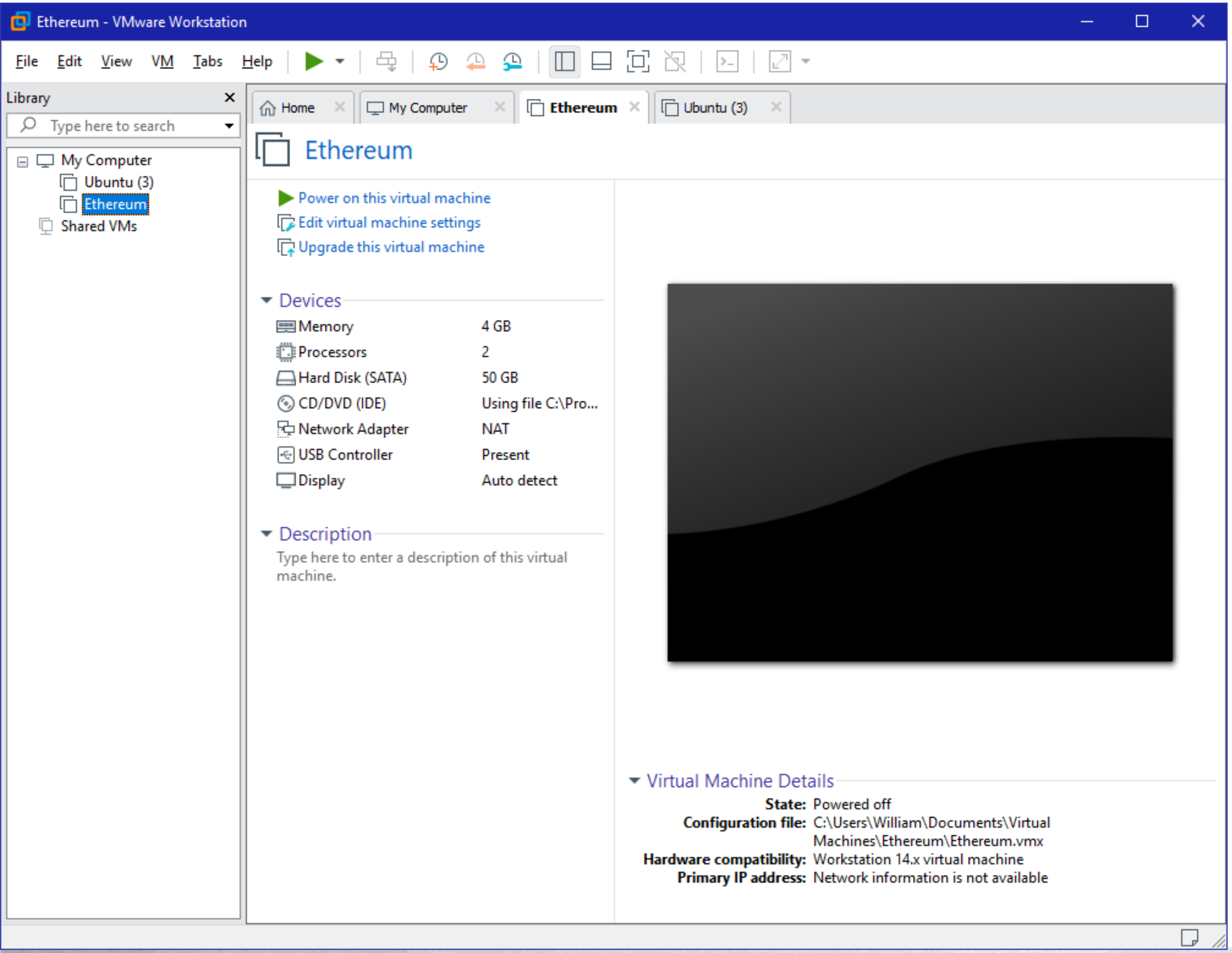
Using a VM to Work with Truffle

VMWare Professional Workstation 15 Pro



Your evaluation period has ended. A valid license key is required to power on virtual machines.

Select the VM Disk Image that Includes Ubuntu, Ethereum, and Truffle



Ethereum

Power on this virtual machine
Edit virtual machine settings
Upgrade this virtual machine

Devices

Memory	4 GB
Processors	2
Hard Disk (SATA)	50 GB
CD/DVD (IDE)	Using file C:\Pro...
Network Adapter	NAT
USB Controller	Present
Display	Auto detect

Description

Type here to enter a description of this virtual machine.

Virtual Machine Details

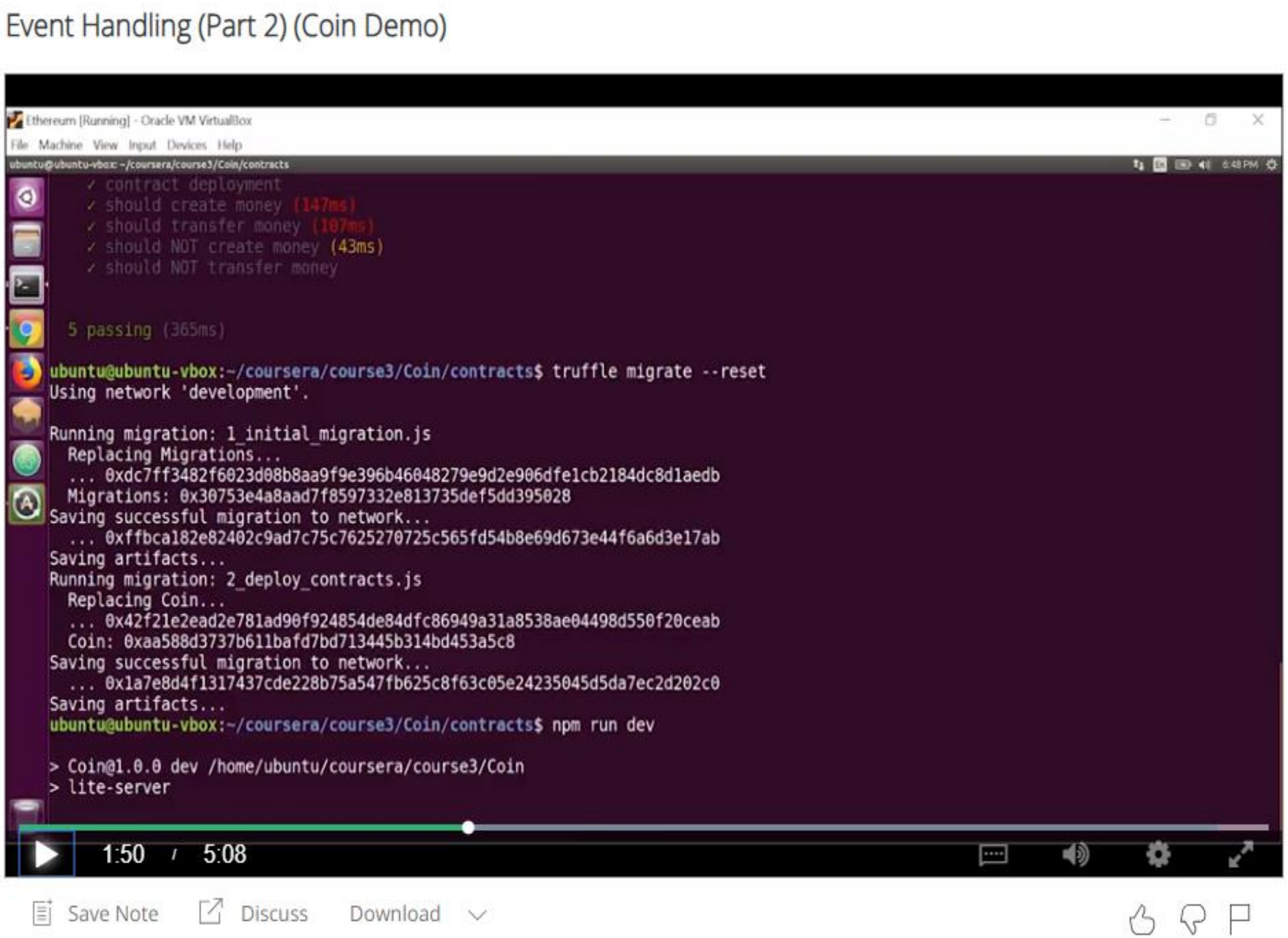
State: Powered off
Configuration file: C:\Users\William\Documents\Virtual Machines\Ethereum\Ethereum.vmx
Hardware compatibility: Workstation 14.x virtual machine
Primary IP address: Network information is not available

Install VMWare Workstation for your Operating System and add and configure it with an Ubuntu image that already has a) NodeJS; b) git; c) Ethereum Client, MetaMask, and Truffle for Linux all installed. The images for VMWare Workstation and the Ubuntu VM are at https://tinyurl.com/y46paxkg |

00 Day 02 Materials | 00 VMWare Workstation Images
00 Day 02 Materials | 00 Blockchain Dev Platforms | 00 Ubuntu VM

Note: You will need to apply for
A 30-day Free License Key.

Select the VM Disk Image that Includes Ubuntu, Ethereum, and Truffle



Event Handling (Part 2) (Coin Demo)

Ethereum [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
ubuntu@ubuntu-vbox: ~/coursera/course3/Coin/contracts
6:48 PM
1:50 / 5:08
Save Note
Discuss
Download

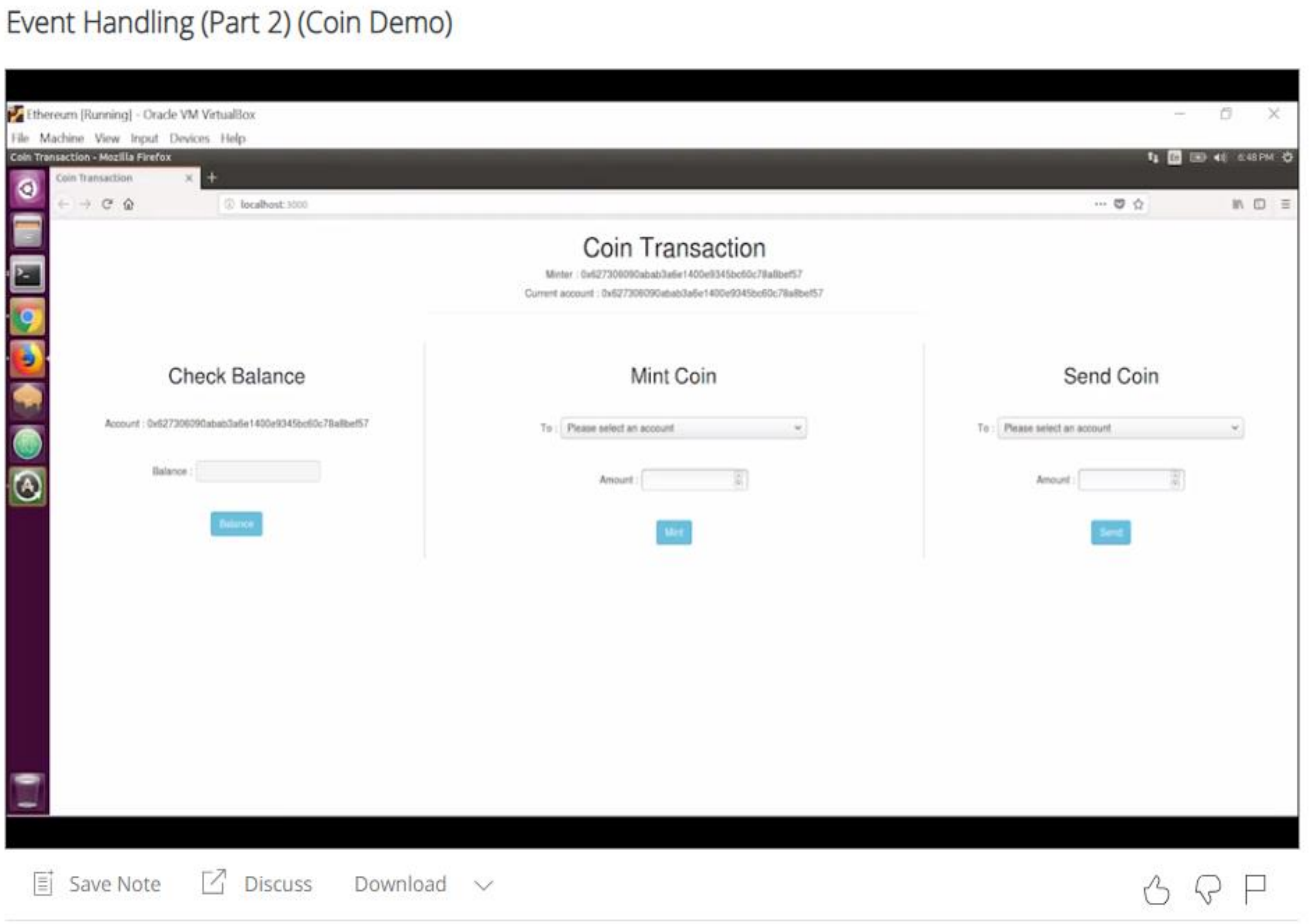
```
    ✓ contract deployment
    ✓ should create money (147ms)
    ✓ should transfer money (107ms)
    ✓ should NOT create money (43ms)
    ✓ should NOT transfer money


  5 passing (365ms)

ubuntu@ubuntu-vbox:~/coursera/course3/Coin/contracts$ truffle migrate --reset
Using network 'development'.

Running migration: 1_initial_migration.js
  Replacing Migrations...
  ... 0xdc7ff3482f6023d08b8aa9f9e396b46048279e9d2e906dfe1cb2184dc8d1aedb
  Migrations: 0x30753e4a8aad7f8597332e813735def5dd395028
Saving successful migration to network...
  ... 0xffbca182e82402c9ad7c75c7625270725c565fd54b8e69d673e44f6a6d3e17ab
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Replacing Coin...
  ... 0x42f21e2ead2e781ad90f924854de84dfc86949a31a8538ae04498d550f20ceab
  Coin: 0xaa588d3737b611bafd7bd713445b314bd453a5c8
Saving successful migration to network...
  ... 0x1a7e8d4f1317437cde228b75a547fb625c8f63c05e24235045d5da7ec2d202c0
Saving artifacts...
ubuntu@ubuntu-vbox:~/coursera/course3/Coin/contracts$ npm run dev

> Coin@1.0.0 dev /home/ubuntu/coursera/course3/Coin
> lite-server
```

Install VMWare Workstation for your Operating System and add and configure it with an Ubuntu image that already has a) NodeJS; b) git; c) Ethereum Client, MetaMask, and Truffle for Linux all installed. The images for VMWare Workstation and the Ubuntu VM are at https://tinyurl.com/y46paxkg |

00 Day 02 Materials | 00 VMWare Workstation Images
00 Day 02 Materials | 00 Blockchain Dev Platforms | 00 Ubuntu VM

Note: You will need to apply for
A 30-day Free License Key.

Select the VM Disk Image that Includes Ubuntu, Ethereum, and Truffle


Event Handling (Part 2) (Coin Demo)
Ethereum [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Coin Transaction - Mozilla Firefox
Coin Transaction
localhost:3000
Check Balance
Mint Coin
Send Coin
Balance
Mint
Send
Save Note
Discuss
Download

Coin Transaction

Minter : 0x627306090abab3a6e1400e9345bc60c78a8bef57

Current account : 0x627306090abab3a6e1400e9345bc60c78a8bef57

Check Balance

Account : 0x627306090abab3a6e1400e9345bc60c78a8bef57

Balance :

Mint Coin

To : Please select an account

Amount :

Send Coin

To : Please select an account

Amount :

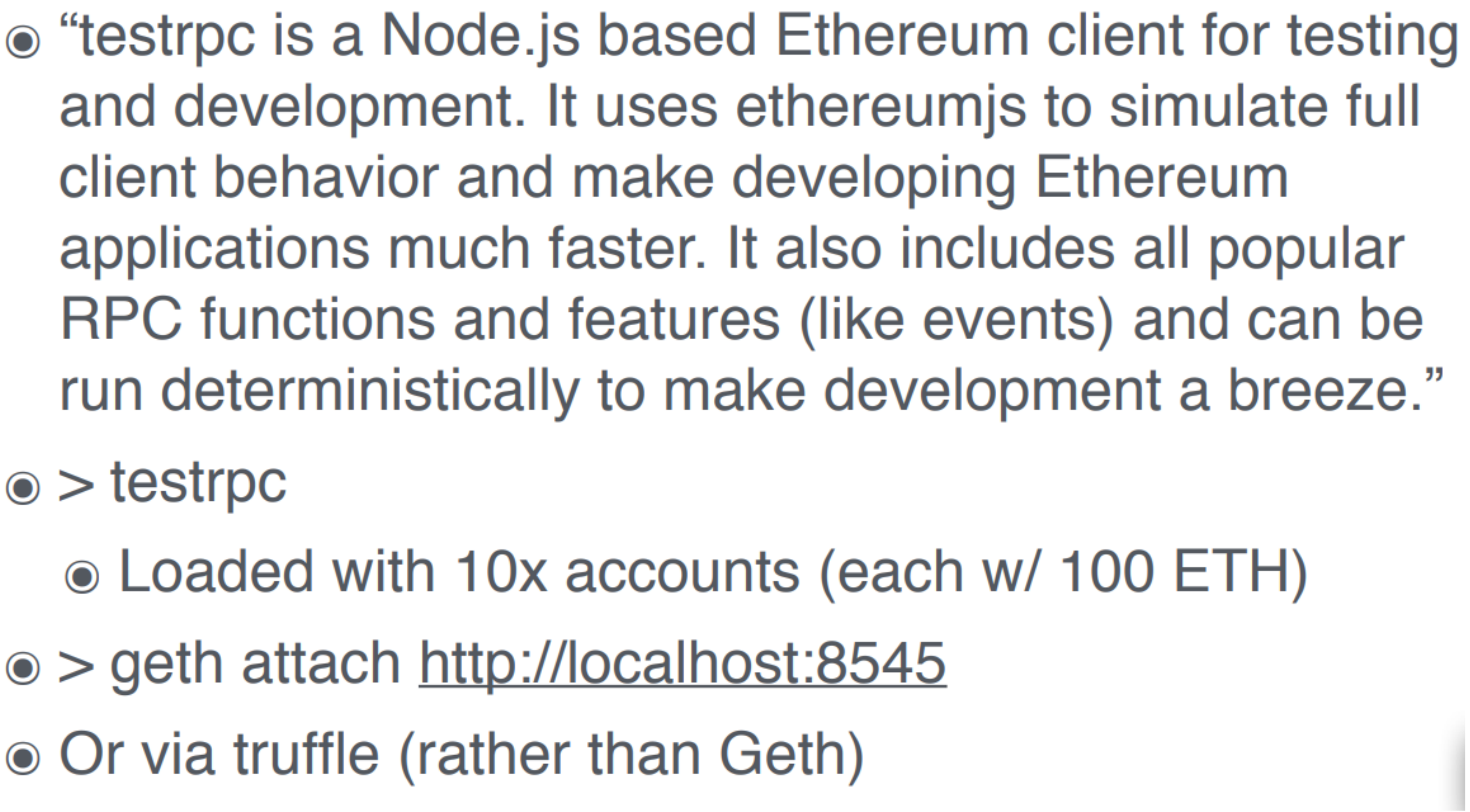
Install VMWare Workstation for your Operating System and add and configure it with an Ubuntu image that already has a) NodeJS; b) git; c) Ethereum Client, MetaMask, and Truffle for Linux all installed. The images for VMWare Workstation and the Ubuntu VM are at https://tinyurl.com/y46paxkg |

00 Day 02 Materials | 00 VMWare Workstation Images
00 Day 02 Materials | 00 Blockchain Dev Platforms | 00 Ubuntu VM

Note: You will need to apply for
A 30-day Free License Key.

TestRPC

TestRPC

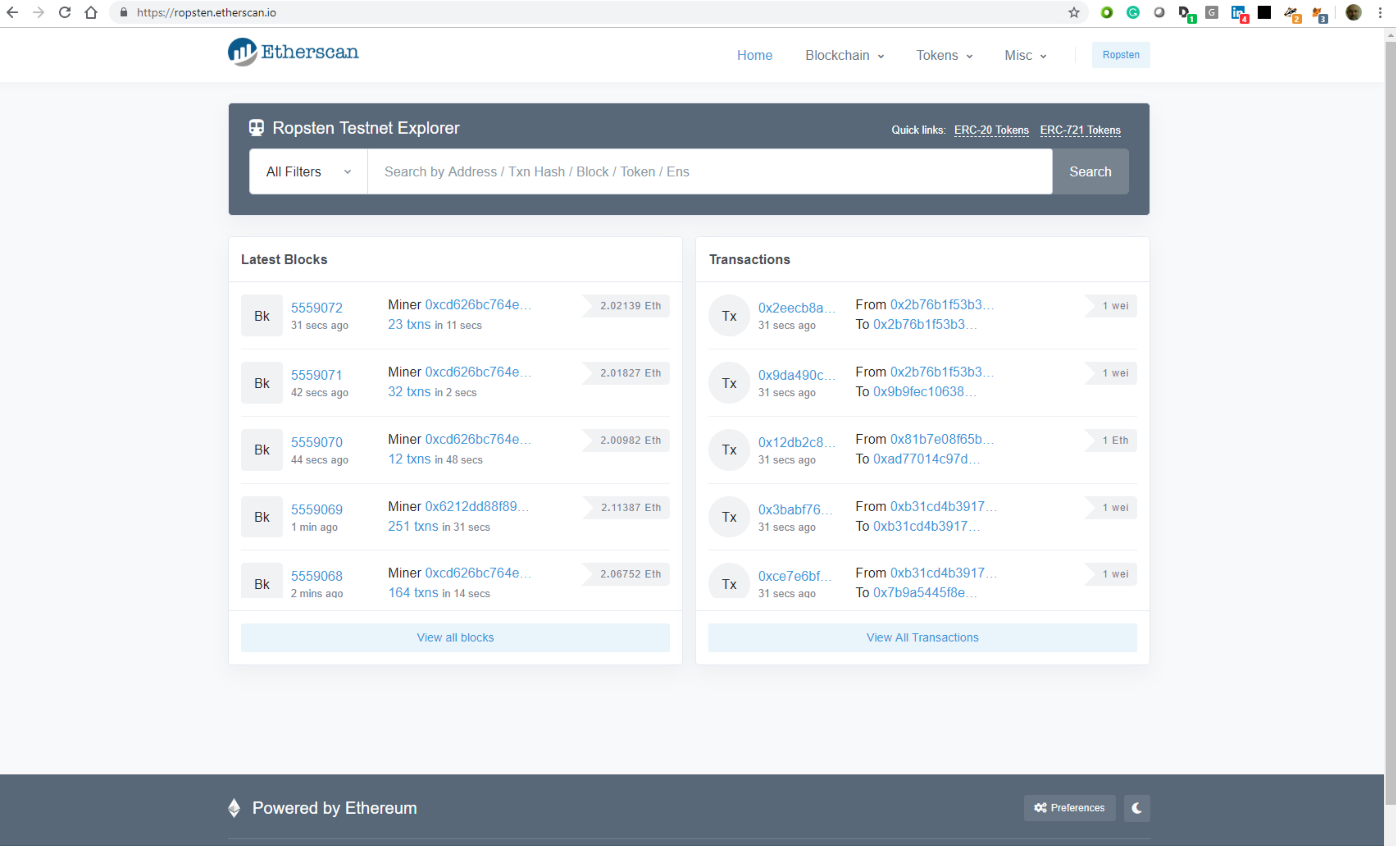


- "testrpc is a Node.js based Ethereum client for testing and development. It uses ethereumjs to simulate full client behavior and make developing Ethereum applications much faster. It also includes all popular RPC functions and features (like events) and can be run deterministically to make development a breeze."
- > testrpc
  - Loaded with 10x accounts (each w/ 100 ETH)
- > geth attach http://localhost:8545
- Or via truffle (rather than Geth)

TestNets

- https://ropsten.etherscan.io/
- https://ropsten.etherscan.io/address/0x1cda2ea9673146dc4bf55662fe14bef11c22ea78

Ropsten in Etherscan Blockchain Explorer



https://ropsten.etherscan.io

Etherscan

Home | Blockchain | Tokens | Misc | Ropsten

Ropsten Testnet Explorer

Quick links: ERC-20 Tokens ERC-721 Tokens

All Filters | Search by Address / Txn Hash / Block / Token / Ens | Search

Latest Blocks

Bk	5559072 31 secs ago	Miner 0xcd626bc764e... 23 txns in 11 secs	2.02139 Eth
Bk	5559071 42 secs ago	Miner 0xcd626bc764e... 32 txns in 2 secs	2.01827 Eth
Bk	5559070 44 secs ago	Miner 0xcd626bc764e... 12 txns in 48 secs	2.00982 Eth
Bk	5559069 1 min ago	Miner 0x6212dd88f89... 251 txns in 31 secs	2.11387 Eth
Bk	5559068 2 mins ago	Miner 0xcd626bc764e... 164 txns in 14 secs	2.06752 Eth

View all blocks

Transactions

Tx	0x2eecb8a... 31 secs ago	From 0x2b76b1f53b3... To 0x2b76b1f53b3...	1 wei
Tx	0x9da490c... 31 secs ago	From 0x2b76b1f53b3... To 0x9b9fec10638...	1 wei
Tx	0x12db2c8... 31 secs ago	From 0x81b7e08f65b... To 0xad77014c97d...	1 Eth
Tx	0x3babf76... 31 secs ago	From 0xb31cd4b3917... To 0xb31cd4b3917...	1 wei
Tx	0xce7e6bf... 31 secs ago	From 0xb31cd4b3917... To 0x7b9a5445f8e...	1 wei

View All Transactions

Powered by Ethereum

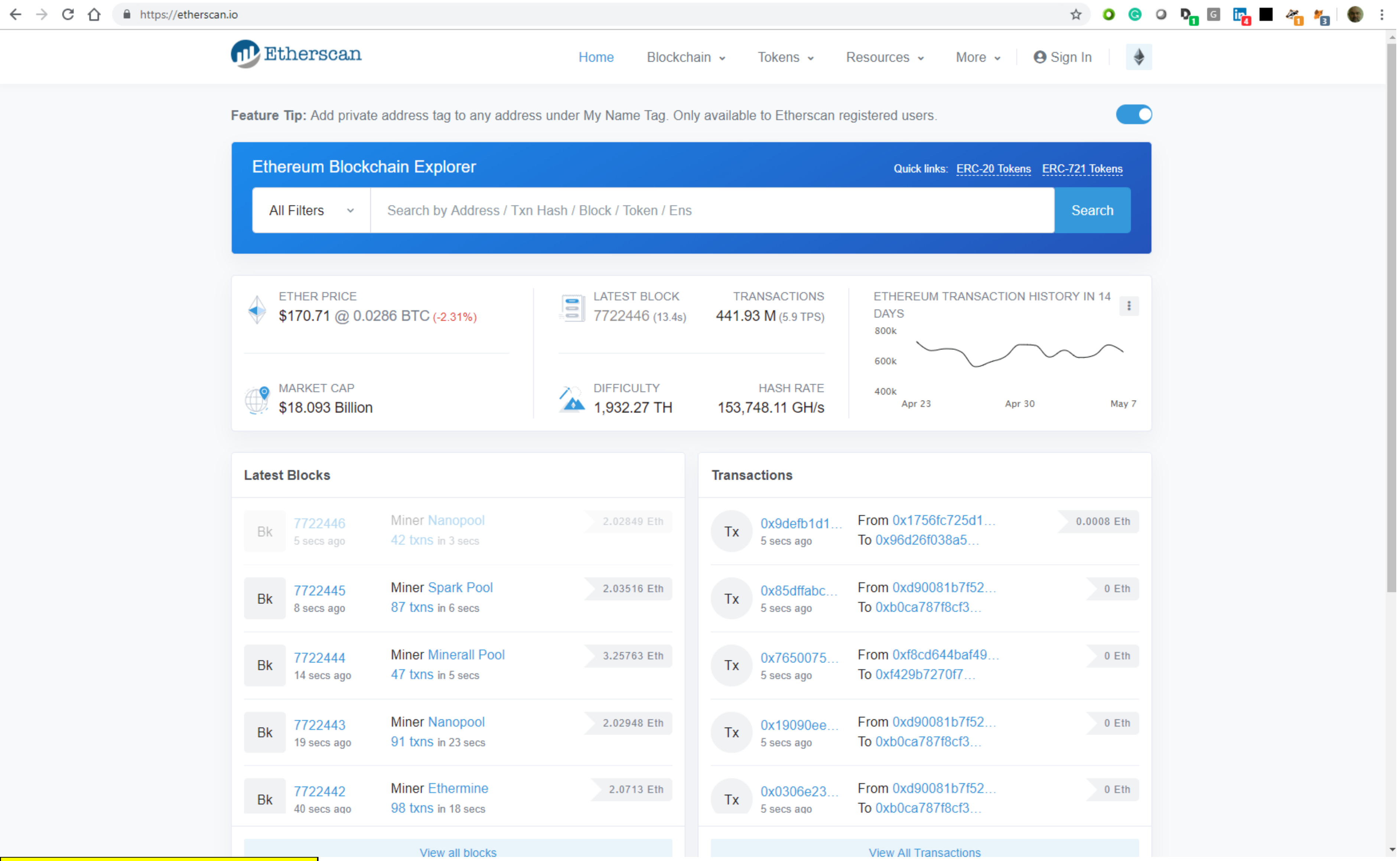
Preferences

Sourcehttps://ropsten.etherscan.io

Ethereum MainNet

- https://etherscan.io/
- https://etherscan.io/address/0x830e3a6766c753e041aa5b78e94213972a99d400

Ethereum Mainnet in Etherscan Blockchain Explorer



https://etherscan.io

Etherscan

Home | Blockchain | Tokens | Resources | More | Sign In

Feature Tip: Add private address tag to any address under My Name Tag. Only available to Etherscan registered users.

Ethereum Blockchain Explorer

Quick links: ERC-20 Tokens ERC-721 Tokens

All Filters | Search by Address / Txn Hash / Block / Token / Ens | Search

ETHER PRICE
$170.71 @ 0.0286 BTC (-2.31%)

LATEST BLOCK
7722446 (13.4s)

TRANSACTIONS
441.93 M (5.9 TPS)

ETHEREUM TRANSACTION HISTORY IN 14 DAYS

MARKET CAP
$18.093 Billion

DIFFICULTY
1,932.27 TH

HASH RATE
153,748.11 GH/s

Latest Blocks

Bk	7722446 5 secs ago	Miner Nanopool 42 txns in 3 secs	2.02849 Eth
Bk	7722445 8 secs ago	Miner Spark Pool 87 txns in 6 secs	2.03516 Eth
Bk	7722444 14 secs ago	Miner Minerall Pool 47 txns in 5 secs	3.25763 Eth
Bk	7722443 19 secs ago	Miner Nanopool 91 txns in 23 secs	2.02948 Eth
Bk	7722442 40 secs ago	Miner Ethermine 98 txns in 18 secs	2.0713 Eth

View all blocks

Transactions

Tx	0x9defb1d1... 5 secs ago	From 0x1756fc725d1... To 0x96d26f038a5...	0.0008 Eth
Tx	0x85dffabc... 5 secs ago	From 0xd90081b7f52... To 0xb0ca787f8cf3...	0 Eth
Tx	0x7650075... 5 secs ago	From 0xf8cd644baf49... To 0xf429b7270f7...	0 Eth
Tx	0x19090ee... 5 secs ago	From 0xd90081b7f52... To 0xb0ca787f8cf3...	0 Eth
Tx	0x0306e23... 5 secs ago	From 0xd90081b7f52... To 0xb0ca787f8cf3...	0 Eth

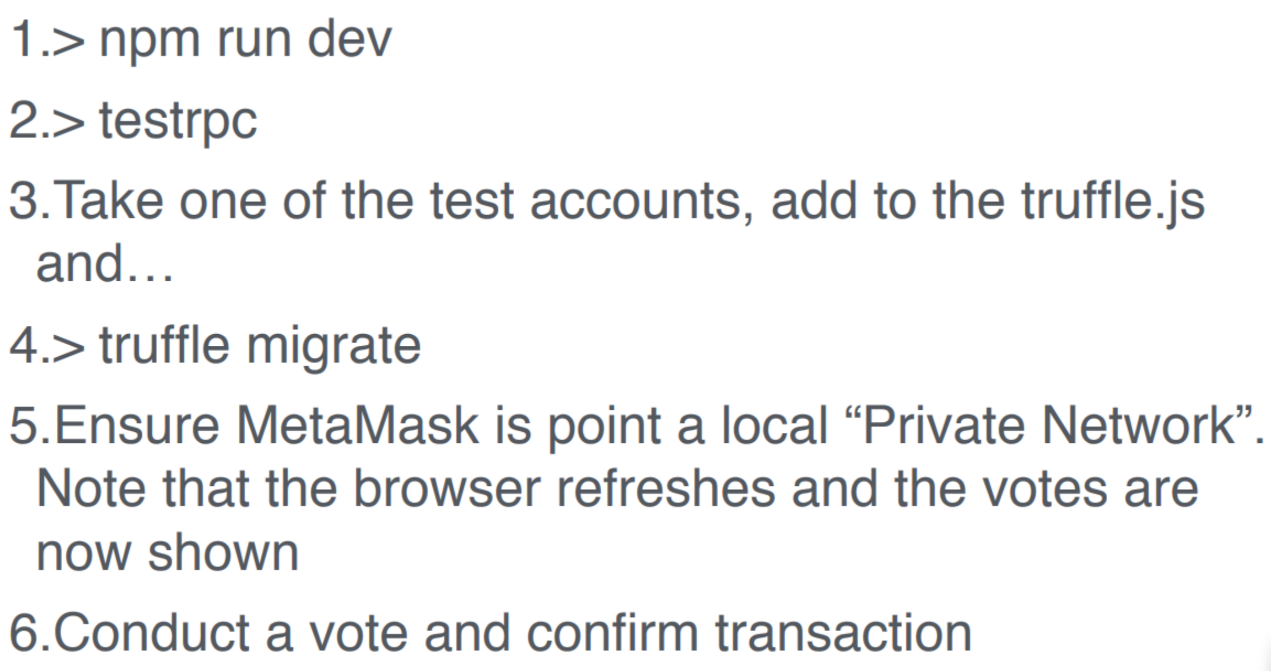
View All Transactions

Source: https://.etherscan.io

Things to Note

- The backend is Ethereum's blockchain
- In this case a local test network (e.g. dev env)
- Web3.js is used as the bridge from the client to the blockchain
- MetaMask is used to connect a wallet

Setup Steps



1.> npm run dev

2.> testrpc

3.Take one of the test accounts, add to the truffle.js and…

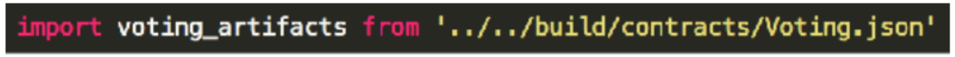
4.> truffle migrate

5.Ensure MetaMask is point a local “Private Network”. Note that the browser refreshes and the votes are now shown

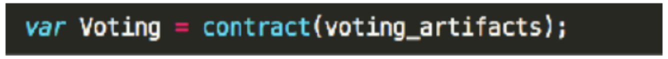
6.Conduct a vote and confirm transaction

Reviewing the Web3.js

- The ABI (Application Binary Interface)

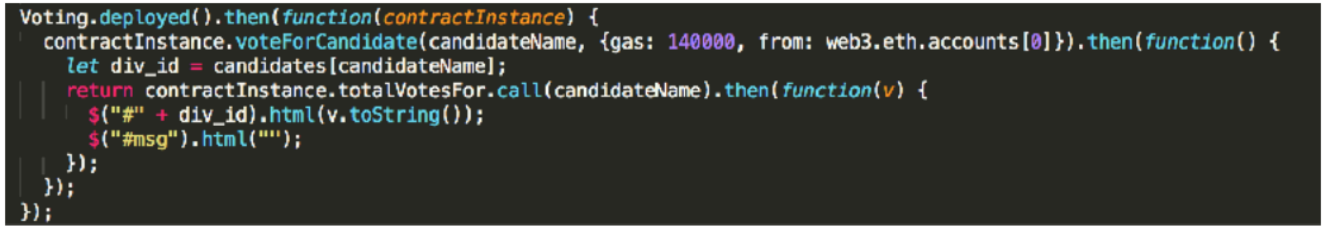


```
import voting_artifacts from '../../build/contracts/Voting.json'
```



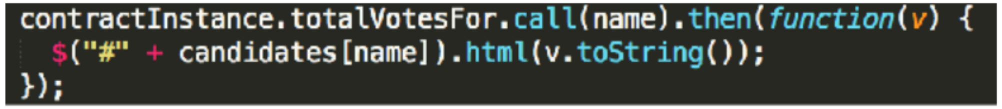
```
var Voting = contract(voting_artifacts);
```

- Voting (note the name, gas, and account params)



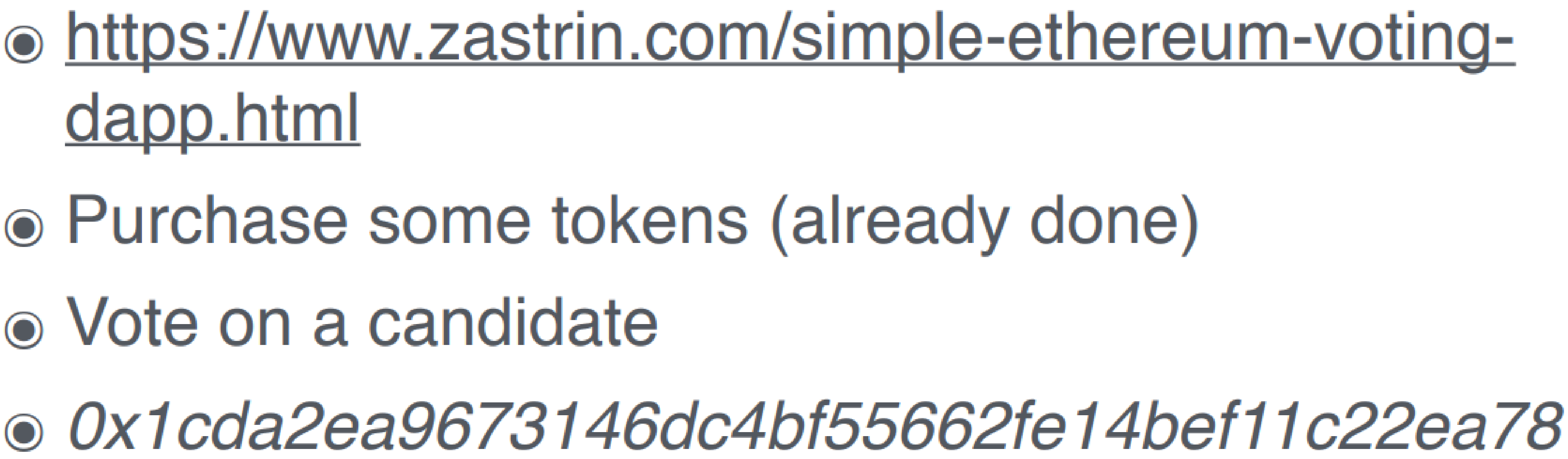
```
Voting.deployed().then(function(contractInstance) {
  contractInstance.voteForCandidate(candidateName, {gas: 140000, from: web3.eth.accounts[0]}).then(function() {
    let div_id = candidates[candidateName];
    return contractInstance.totalVotesFor.call(candidateName).then(function(v) {
      $("#" + div_id).html(v.toString());
      $("#msg").html("");
    });
  });
});
```

- Retrieving the current votes (reading doesn't need gas)



```
contractInstance.totalVotesFor.call(name).then(function(v) {
  $("#" + candidates[name]).html(v.toString());
});
```

Decentralized Voting Application

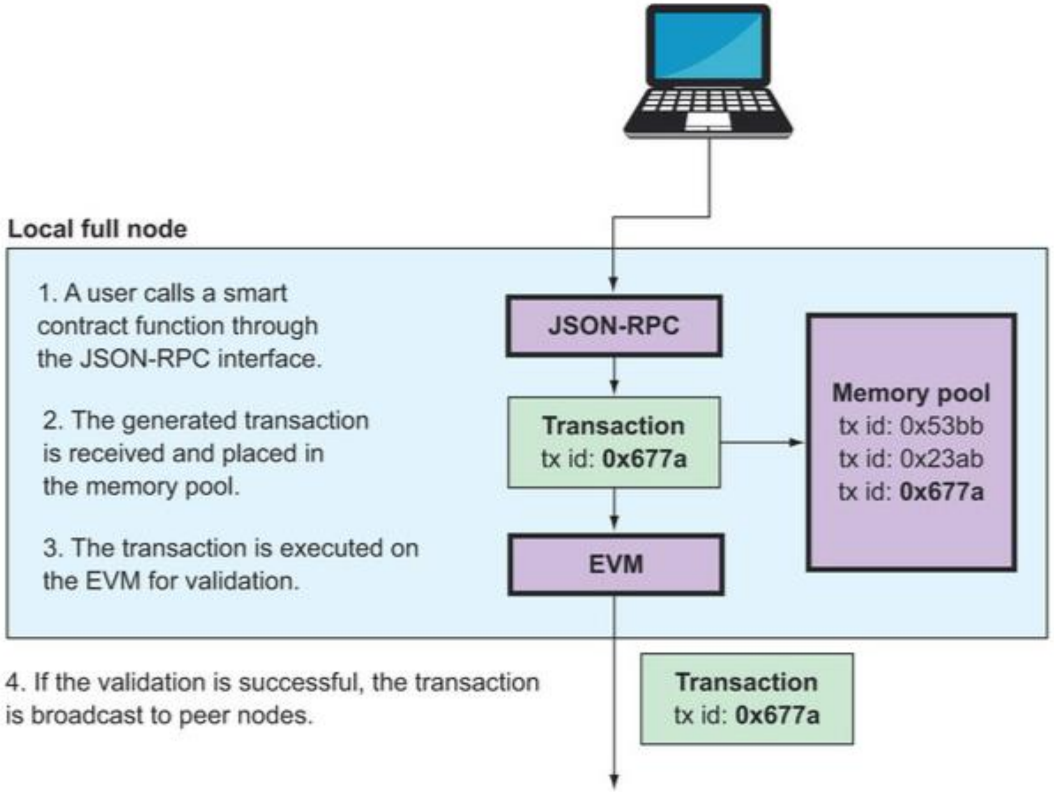


- https://www.zastrin.com/simple-ethereum-voting-dapp.html
- Purchase some tokens (already done)
- Vote on a candidate
- *0x1cda2ea9673146dc4bf55662fe14bef11c22ea78*

Source: https://kevin.bluer.com/downloads/first-ethereum-dapp.pdf

DApp or Smart Contract Transaction Lifecycle

Figure 2.2. The lifecycle of a transaction. A voting transaction is created when a function is invoked on a smart contract on a chosen Ethereum node through the JSON-RPC interface. The node places the transaction in the memory pool and executes it on the EVM for validation. If the validation is successful, the transaction is broadcast to peer nodes until it reaches a mining node; otherwise, it dies out.



1. A user calls a smart contract function through the JSON-RPC interface.

2. The generated transaction is received and placed in the memory pool.

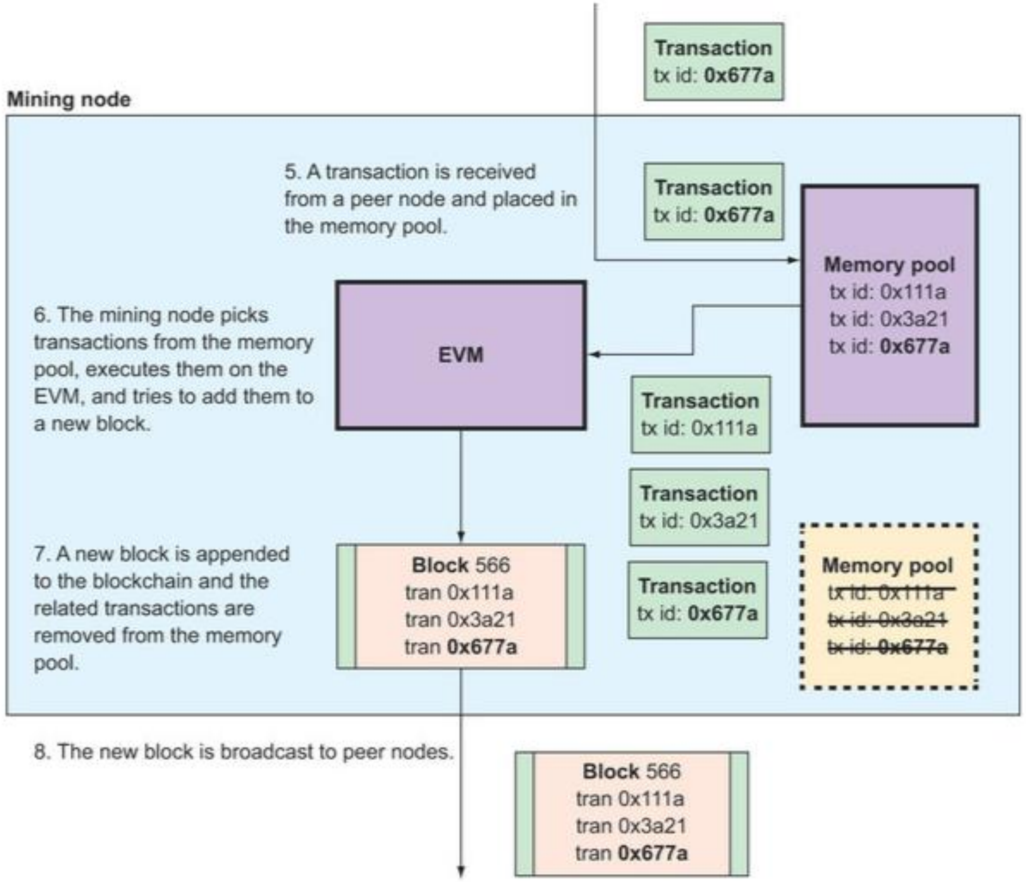
3. The transaction is executed on the EVM for validation.

4. If the validation is successful, the transaction is broadcast to peer nodes.

Source: Roberto Infante, Building Ethereum DApps, 2019

Ethereum Mining Node

Figure 2.3. A mining node receives the transaction from a peer node and places it in its memory pool. The node later picks it and executes it on the EVM, among other transactions, to place it on a new block. If the block is appended on the blockchain, the transaction is removed from the memory pool and the block is broadcast to peer nodes.



5. A transaction is received from a peer node and placed in the memory pool.

6. The mining node picks transactions from the memory pool, executes them on the EVM, and tries to add them to a new block.

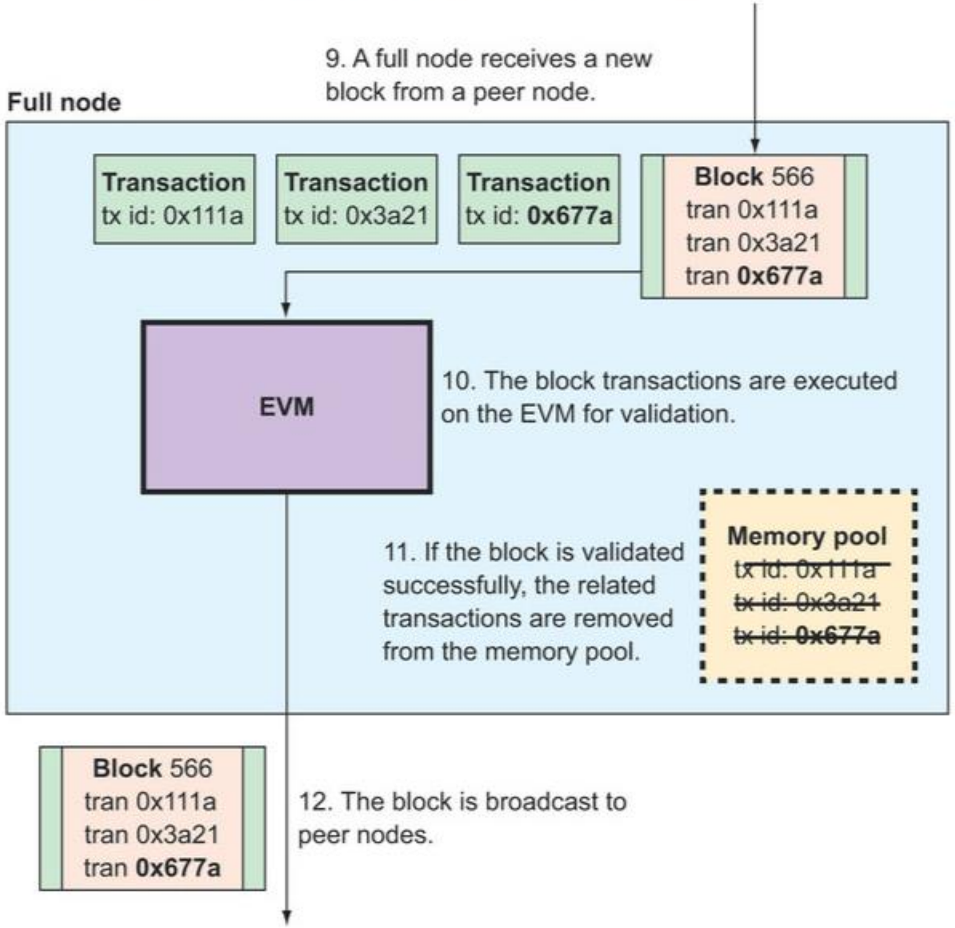
7. A new block is appended to the blockchain and the related transactions are removed from the memory pool.

8. The new block is broadcast to peer nodes.

Source: Roberto Infante, Building Ethereum DApps, 2019

Ethereum Full Node’s Process

Figure 2.4. The full node’s process, from when it receives the new block to when it processes all its transactions on the EVM for validation, then, if validation is successful, removes the related transactions from the memory pool and propagates the block further into the network



9. A full node receives a new block from a peer node.

Full node

Transaction
tx id: 0x111a

Transaction
tx id: 0x3a21

Transaction
tx id: **0x677a**

Block 566
tran 0x111a
tran 0x3a21
tran **0x677a**

EVM

10. The block transactions are executed on the EVM for validation.

11. If the block is validated successfully, the related transactions are removed from the memory pool.

Memory pool
~~tx id: 0x111a~~
~~tx id: 0x3a21~~
~~tx id: **0x677a**~~

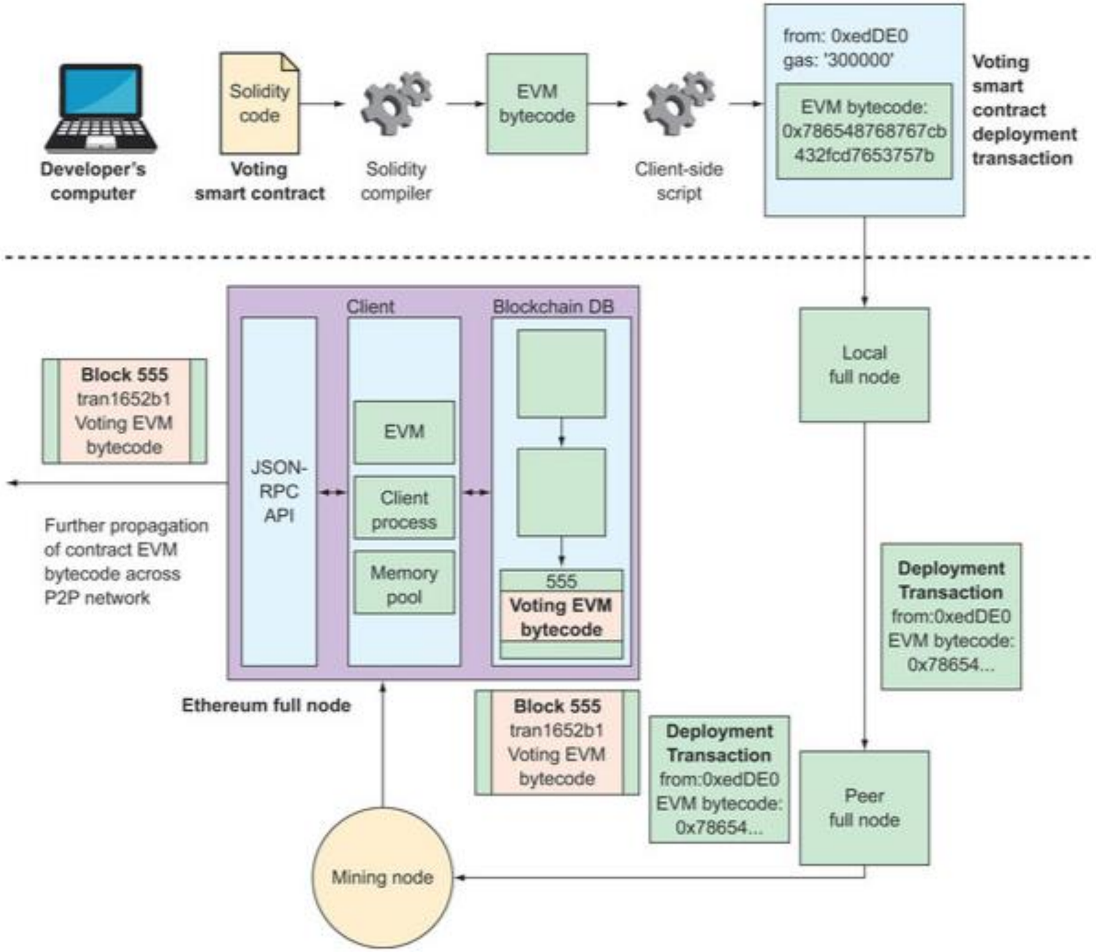
Block 566
tran 0x111a
tran 0x3a21
tran **0x677a**

12. The block is broadcast to peer nodes.

Source: Roberto Infante, Building Ethereum DApps, 2019

Development Lifecycle from Full Node to Mining

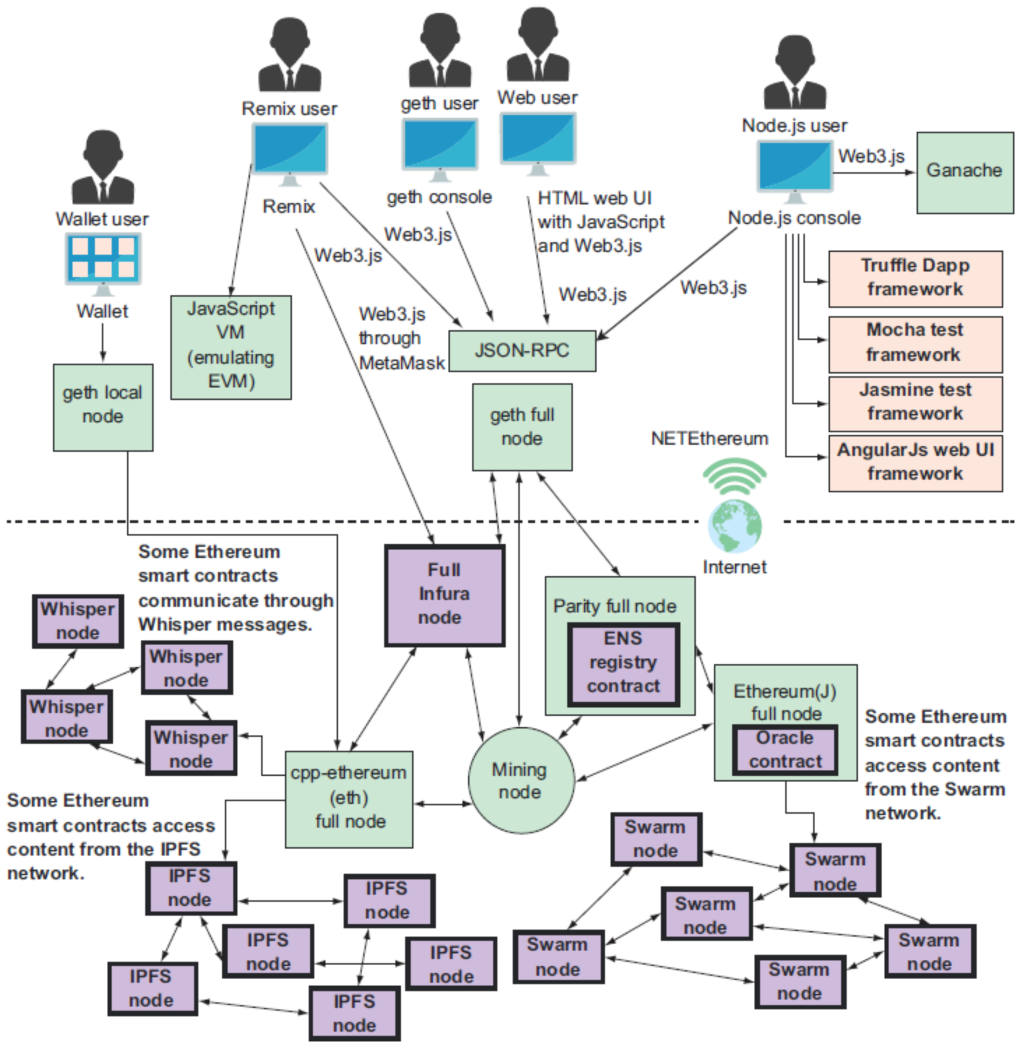
Figure 2.5. A developer writes the voting smart contract in the Solidity language, then compiles it into EVM bytecode and inserts it into a contract deployment transaction. This is pushed to the local Ethereum node and propagated throughout the network. It's then mined and appended to the blockchain.



Source: Roberto Infante, Building Ethereum DApps, 2019. https://www.manning.com/books/building-ethereum-dapps

The Current Ethereum Ecosystem

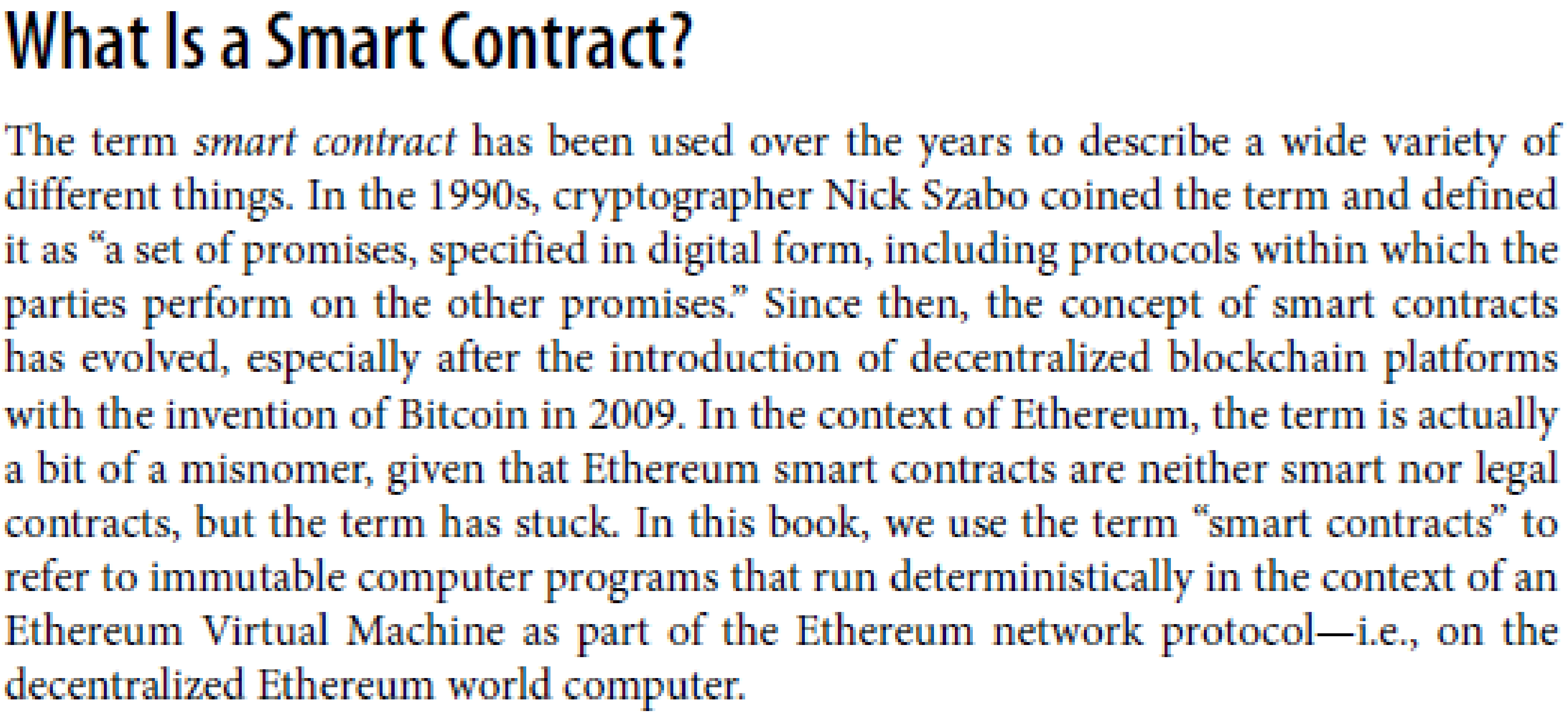
Full View of the Current Ethereum Ecosystem



Source: Roberto Infante, Building Ethereum DApps, 2019. https://www.manning.com/books/building-ethereum-dapps

Smart Contract

Smart Contract Definition

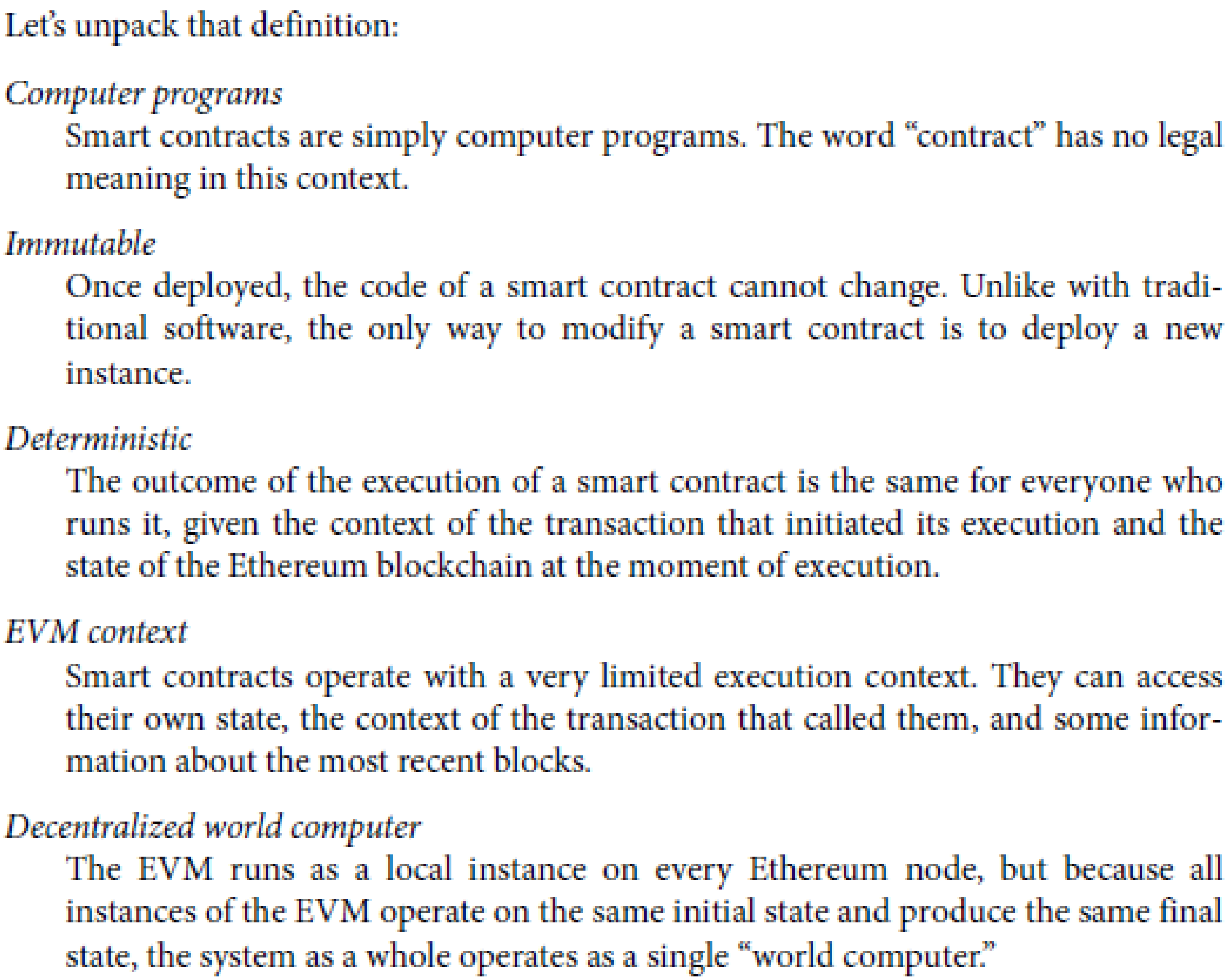


What Is a Smart Contract?

The term *smart contract* has been used over the years to describe a wide variety of different things. In the 1990s, cryptographer Nick Szabo coined the term and defined it as “a set of promises, specified in digital form, including protocols within which the parties perform on the other promises.” Since then, the concept of smart contracts has evolved, especially after the introduction of decentralized blockchain platforms with the invention of Bitcoin in 2009. In the context of Ethereum, the term is actually a bit of a misnomer, given that Ethereum smart contracts are neither smart nor legal contracts, but the term has stuck. In this book, we use the term “smart contracts” to refer to immutable computer programs that run deterministically in the context of an Ethereum Virtual Machine as part of the Ethereum network protocol—i.e., on the decentralized Ethereum world computer.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Smart Contract Definition - Unpacked



Let's unpack that definition:

Computer programs

Smart contracts are simply computer programs. The word "contract" has no legal meaning in this context.

Immutable

Once deployed, the code of a smart contract cannot change. Unlike with traditional software, the only way to modify a smart contract is to deploy a new instance.

Deterministic

The outcome of the execution of a smart contract is the same for everyone who runs it, given the context of the transaction that initiated its execution and the state of the Ethereum blockchain at the moment of execution.

EVM context

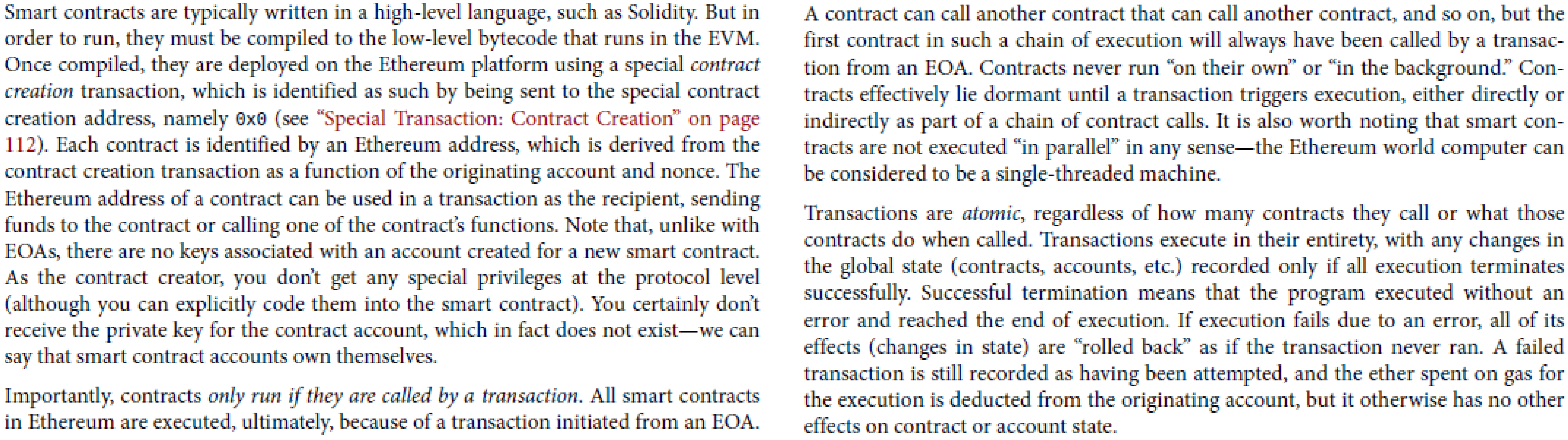
Smart contracts operate with a very limited execution context. They can access their own state, the context of the transaction that called them, and some information about the most recent blocks.

Decentralized world computer

The EVM runs as a local instance on every Ethereum node, but because all instances of the EVM operate on the same initial state and produce the same final state, the system as a whole operates as a single "world computer."

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Smart Contract Lifecycle



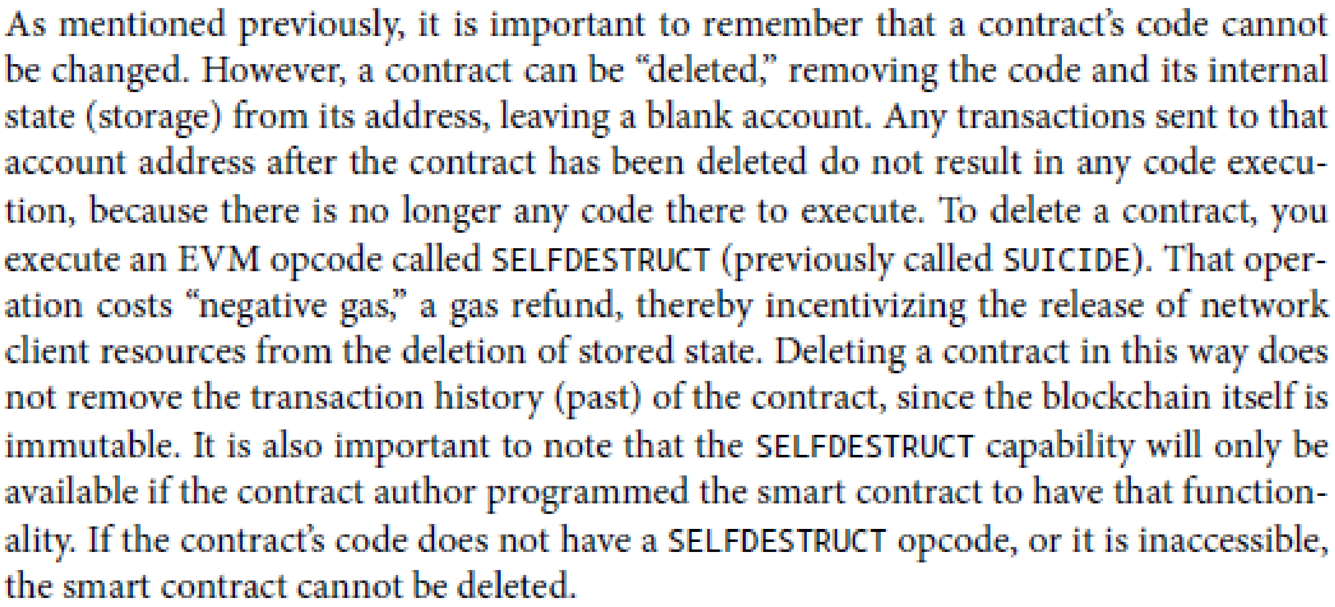
Smart contracts are typically written in a high-level language, such as Solidity. But in order to run, they must be compiled to the low-level bytecode that runs in the EVM. Once compiled, they are deployed on the Ethereum platform using a special *contract creation* transaction, which is identified as such by being sent to the special contract creation address, namely `0x0` (see "Special Transaction: Contract Creation" on page 112). Each contract is identified by an Ethereum address, which is derived from the contract creation transaction as a function of the originating account and nonce. The Ethereum address of a contract can be used in a transaction as the recipient, sending funds to the contract or calling one of the contract's functions. Note that, unlike with EOAs, there are no keys associated with an account created for a new smart contract. As the contract creator, you don't get any special privileges at the protocol level (although you can explicitly code them into the smart contract). You certainly don't receive the private key for the contract account, which in fact does not exist—we can say that smart contract accounts own themselves.

Importantly, contracts *only run if they are called by a transaction*. All smart contracts in Ethereum are executed, ultimately, because of a transaction initiated from an EOA. A contract can call another contract that can call another contract, and so on, but the first contract in such a chain of execution will always have been called by a transaction from an EOA. Contracts never run "on their own" or "in the background." Contracts effectively lie dormant until a transaction triggers execution, either directly or indirectly as part of a chain of contract calls. It is also worth noting that smart contracts are not executed "in parallel" in any sense—the Ethereum world computer can be considered to be a single-threaded machine.

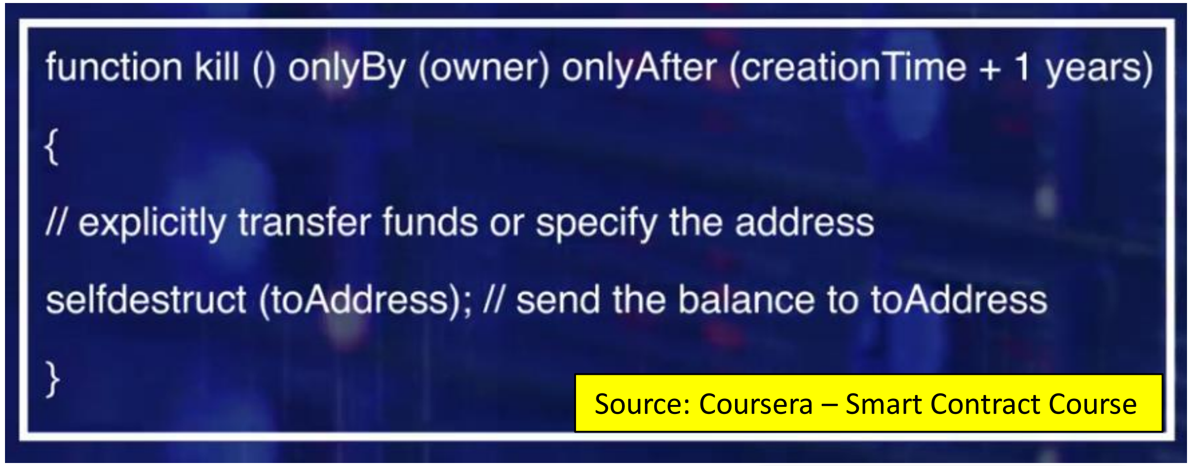
Transactions are *atomic*, regardless of how many contracts they call or what those contracts do when called. Transactions execute in their entirety, with any changes in the global state (contracts, accounts, etc.) recorded only if all execution terminates successfully. Successful termination means that the program executed without an error and reached the end of execution. If execution fails due to an error, all of its effects (changes in state) are "rolled back" as if the transaction never ran. A failed transaction is still recorded as having been attempted, and the ether spent on gas for the execution is deducted from the originating account, but it otherwise has no other effects on contract or account state.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Smart Contract Lifecycle



As mentioned previously, it is important to remember that a contract's code cannot be changed. However, a contract can be "deleted," removing the code and its internal state (storage) from its address, leaving a blank account. Any transactions sent to that account address after the contract has been deleted do not result in any code execution, because there is no longer any code there to execute. To delete a contract, you execute an EVM opcode called `SELFDESTRUCT` (previously called `SUICIDE`). That operation costs "negative gas," a gas refund, thereby incentivizing the release of network client resources from the deletion of stored state. Deleting a contract in this way does not remove the transaction history (past) of the contract, since the blockchain itself is immutable. It is also important to note that the `SELFDESTRUCT` capability will only be available if the contract author programmed the smart contract to have that functionality. If the contract's code does not have a `SELFDESTRUCT` opcode, or it is inaccessible, the smart contract cannot be deleted.



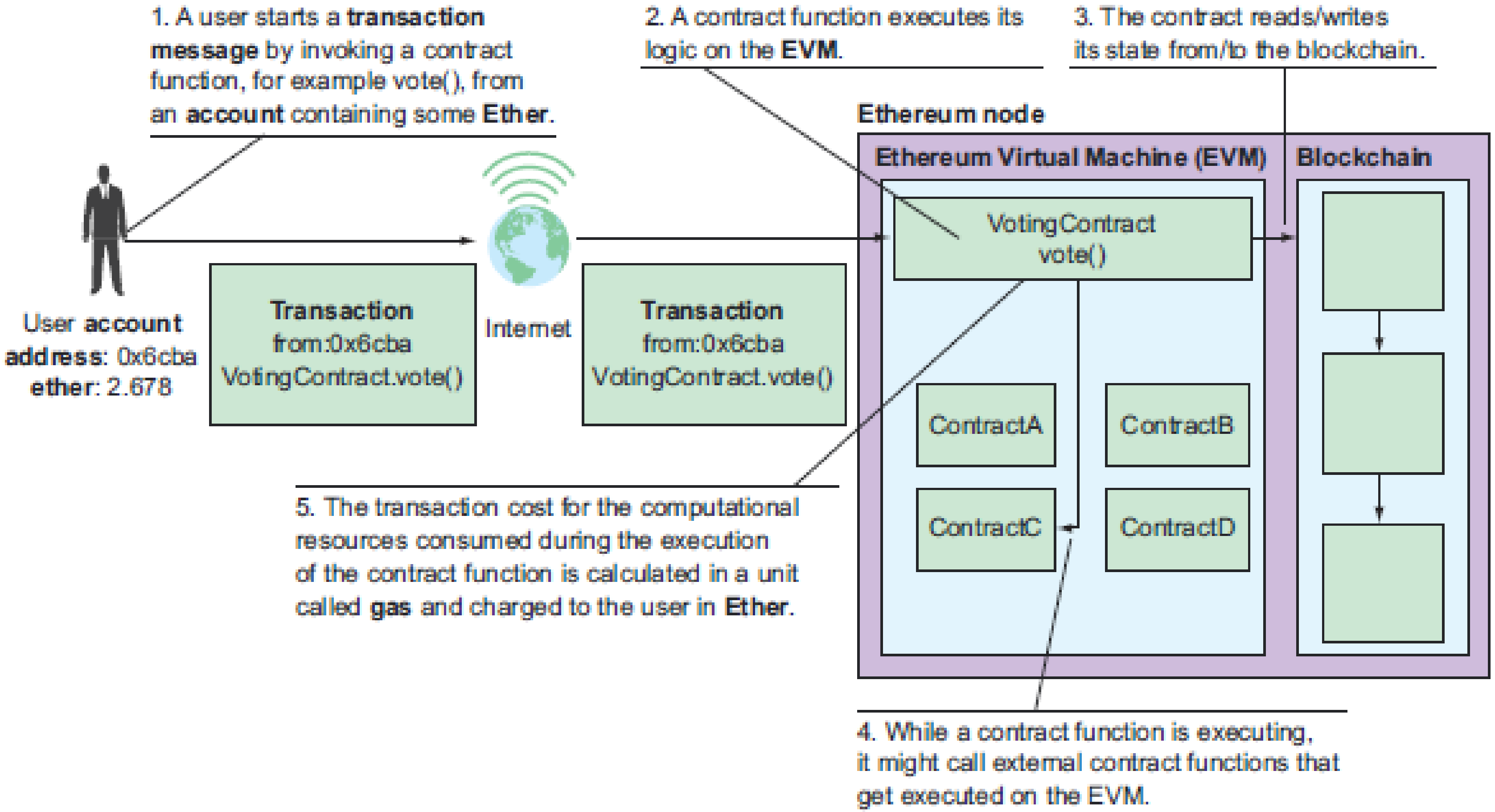
```
function kill () onlyBy (owner) onlyAfter (creationTime + 1 years)
{
// explicitly transfer funds or specify the address
selfdestruct (toAddress); // send the balance to toAddress
}
```

Source: Coursera – Smart Contract Course

Note: This is irreversible.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Visualizing a Smart Contract Execution



1. A user starts a **transaction message** by invoking a contract function, for example vote(), from an **account** containing some **Ether**.

2. A contract function executes its logic on the **EVM**.

3. The contract reads/writes its state from/to the blockchain.

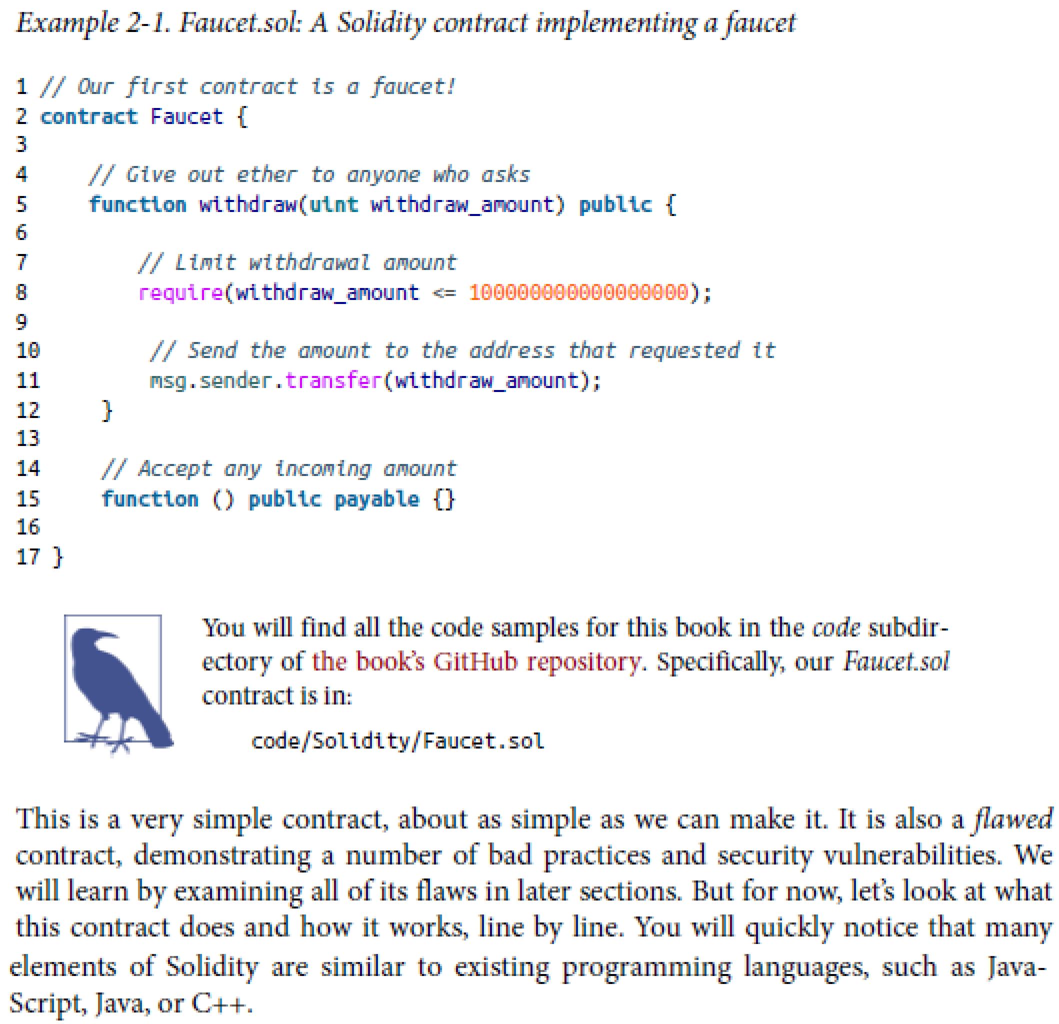
4. While a contract function is executing, it might call external contract functions that get executed on the EVM.

5. The transaction cost for the computational resources consumed during the execution of the contract function is calculated in a unit called **gas** and charged to the user in **Ether**.

Figure 3.8 An Ethereum contract receives a transaction message from a user account. Its logic is executed on the Ethereum Virtual Machine (EVM); then the successful miner calculates the cost for the computational and network resources used, in a unit called gas, and charges the user account in Ether.

Source: Roberto Infante, Building Ethereum DApps, 2019

Faucet – First Solidity Smart Contract Program



Example 2-1. Faucet.sol: A Solidity contract implementing a faucet

```
// Our first contract is a faucet!
contract Faucet {

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {

        // Limit withdrawal amount
        require(withdraw_amount <= 100000000000000000);

         // Send the amount to the address that requested it
         msg.sender.transfer(withdraw_amount);
     }

     // Accept any incoming amount
     function () public payable {}

}
```

You will find all the code samples for this book in the *code* subdirectory of the book's GitHub repository. Specifically, our *Faucet.sol* contract is in:

```
code/Solidity/Faucet.sol
```

This is a very simple contract, about as simple as we can make it. It is also a *flawed* contract, demonstrating a number of bad practices and security vulnerabilities. We will learn by examining all of its flaws in later sections. But for now, let's look at what this contract does and how it works, line by line. You will quickly notice that many elements of Solidity are similar to existing programming languages, such as JavaScript, Java, or C++.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Smart Contract Transactions – Reading & Writing

INTERACTING WITH YOUR CONTRACTS

Introduction

If you were writing raw requests to the Ethereum network yourself in order to interact with your contracts, you'd soon realize that writing these requests is clunky and cumbersome. As well, you might find that managing the state for each request you've made is *complicated*. Fortunately, Truffle takes care of this complexity for you, to make interacting with your contracts a breeze.

Reading and writing data

The Ethereum network makes a distinction between writing data to the network and reading data from it, and this distinction plays a significant part in how you write your application. In general, writing data is called a **transaction** whereas reading data is called a **call**. Transactions and calls are treated very differently, and have the following characteristics.

Transactions

Transactions fundamentally change the state of the network. A transaction can be as simple as sending Ether to another account, or as complicated as executing a contract function or adding a new contract to the network. The defining characteristic of a transaction is that it writes (or changes) data. Transactions cost Ether to run, known as "gas", and transactions take time to process. When you execute a contract's function via a transaction, you cannot receive that function's return value because the transaction isn't processed immediately. In general, functions meant to be executed via a transaction will not return a value; they will return a transaction id instead

Source: https://truffleframework.com/docs/truffle/getting-started/interacting-with-your-contracts

Smart Contract Transactions – Reading & Writing

So in summary, transactions:

Cost gas (Ether)

Change the state of the network

Aren't processed immediately

Won't expose a return value (only a transaction id).

Calls

Calls, on the other hand, are very different. Calls can be used to execute code on the network, though no data will be permanently changed. Calls are free to run, and their defining characteristic is that they read data. When you execute a contract function via a call you will receive the return value immediately. In summary, calls:

Are free (do not cost gas)

Do not change the state of the network

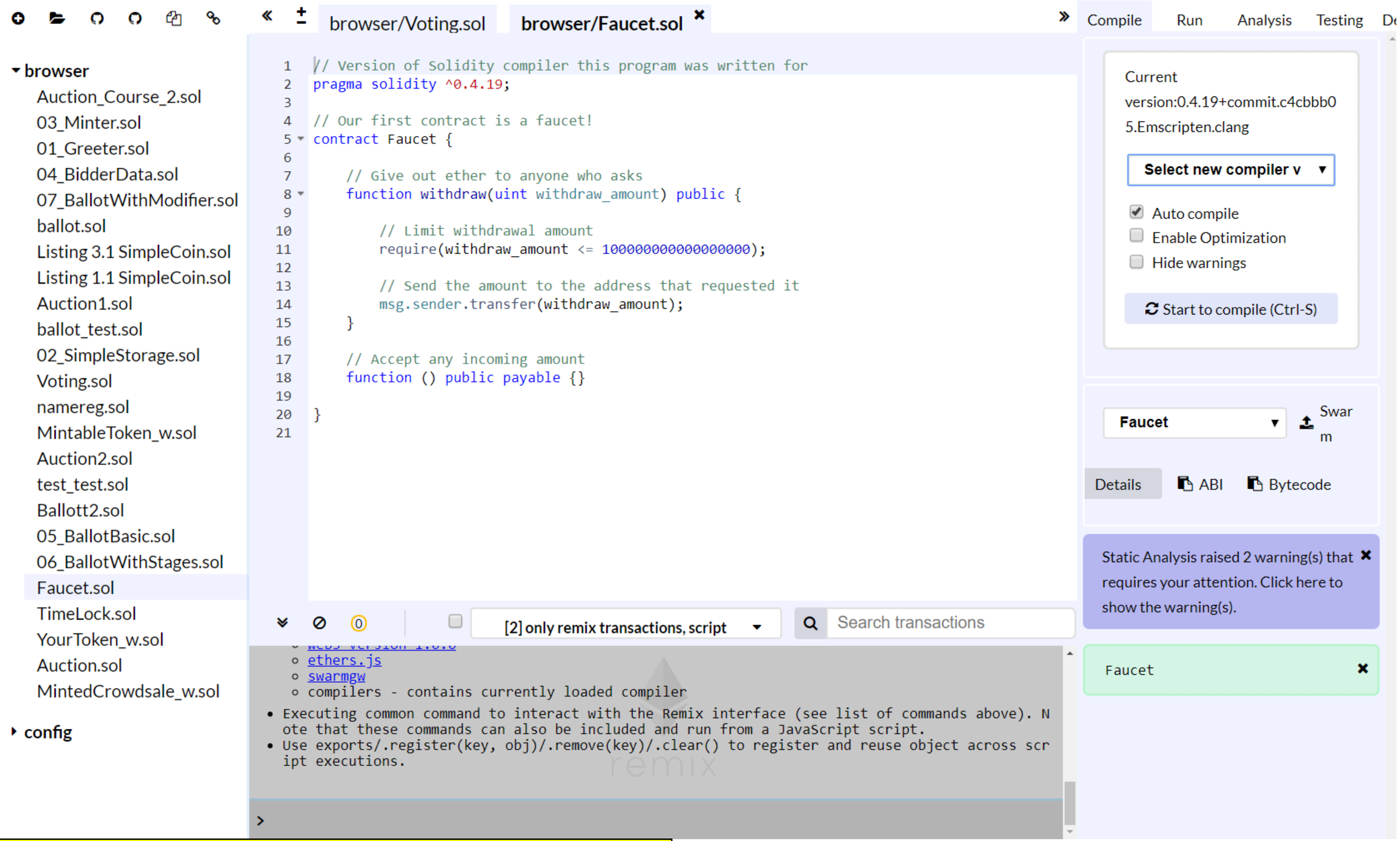
Are processed immediately

Will expose a return value (hooray!)

Choosing between a transaction and a call is as simple as deciding whether you want to read data, or write it.

Source: https://truffleframework.com/docs/truffle/getting-started/interacting-with-your-contracts

Faucet – in Remix IDE to Get Ether for Your Development and Testing



```
// Version of Solidity compiler this program was written for
pragma solidity ^0.4.19;

// Our first contract is a faucet!
contract Faucet {

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {

        // Limit withdrawal amount
        require(withdraw_amount <= 100000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }

    // Accept any incoming amount
    function () public payable {}

}
```

Current version:0.4.19+commit.c4cbbb05.Emscripten.clang

☑ Auto compile
☐ Enable Optimization
☐ Hide warnings

Static Analysis raised 2 warning(s) that requires your attention. Click here to show the warning(s).

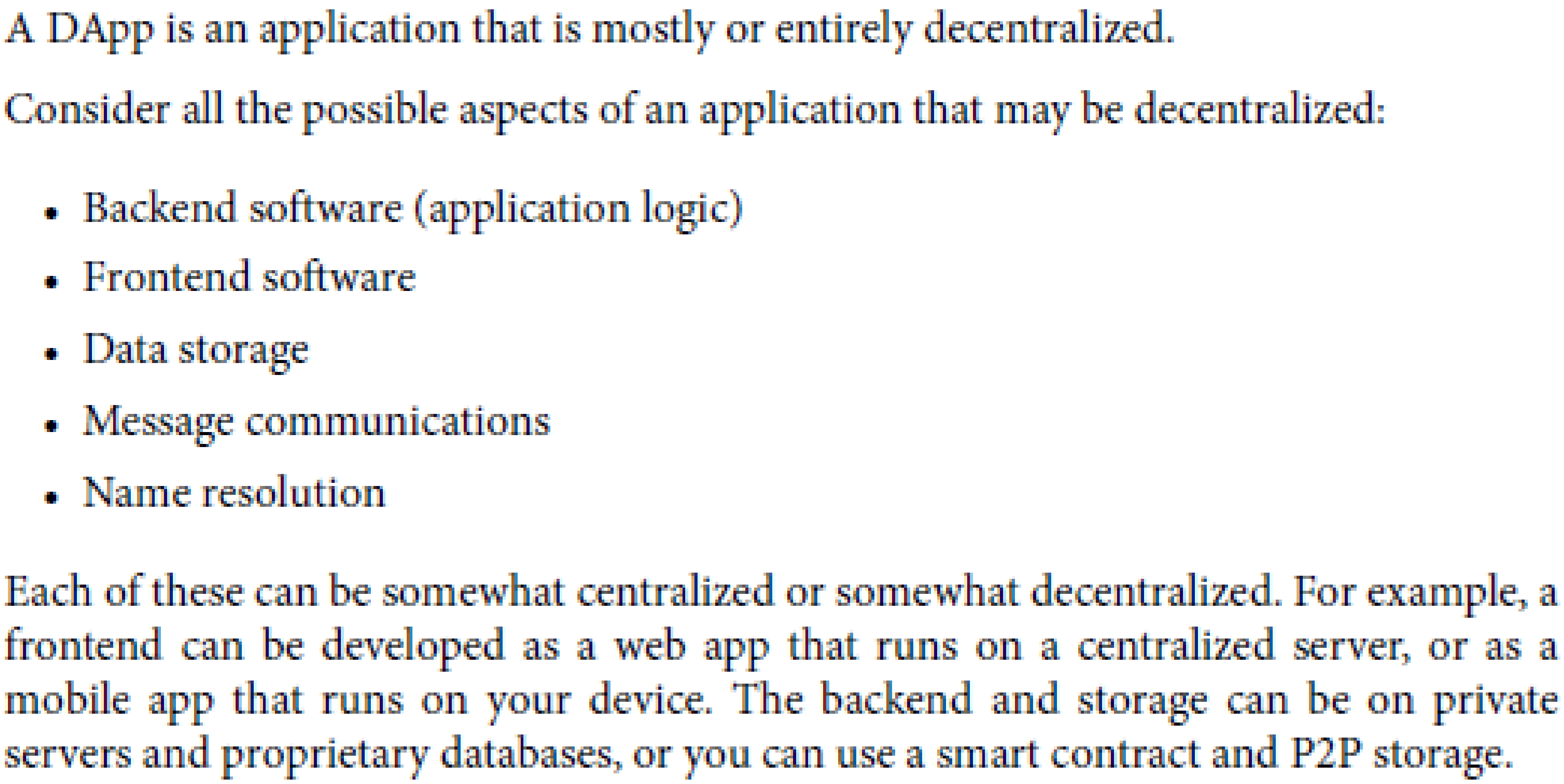
Faucet

```
    o ethers.js
    o swarmgw
    o compilers - contains currently loaded compiler
  • Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
  • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

What Is a DApp?

What is a DApp?



A DApp is an application that is mostly or entirely decentralized.

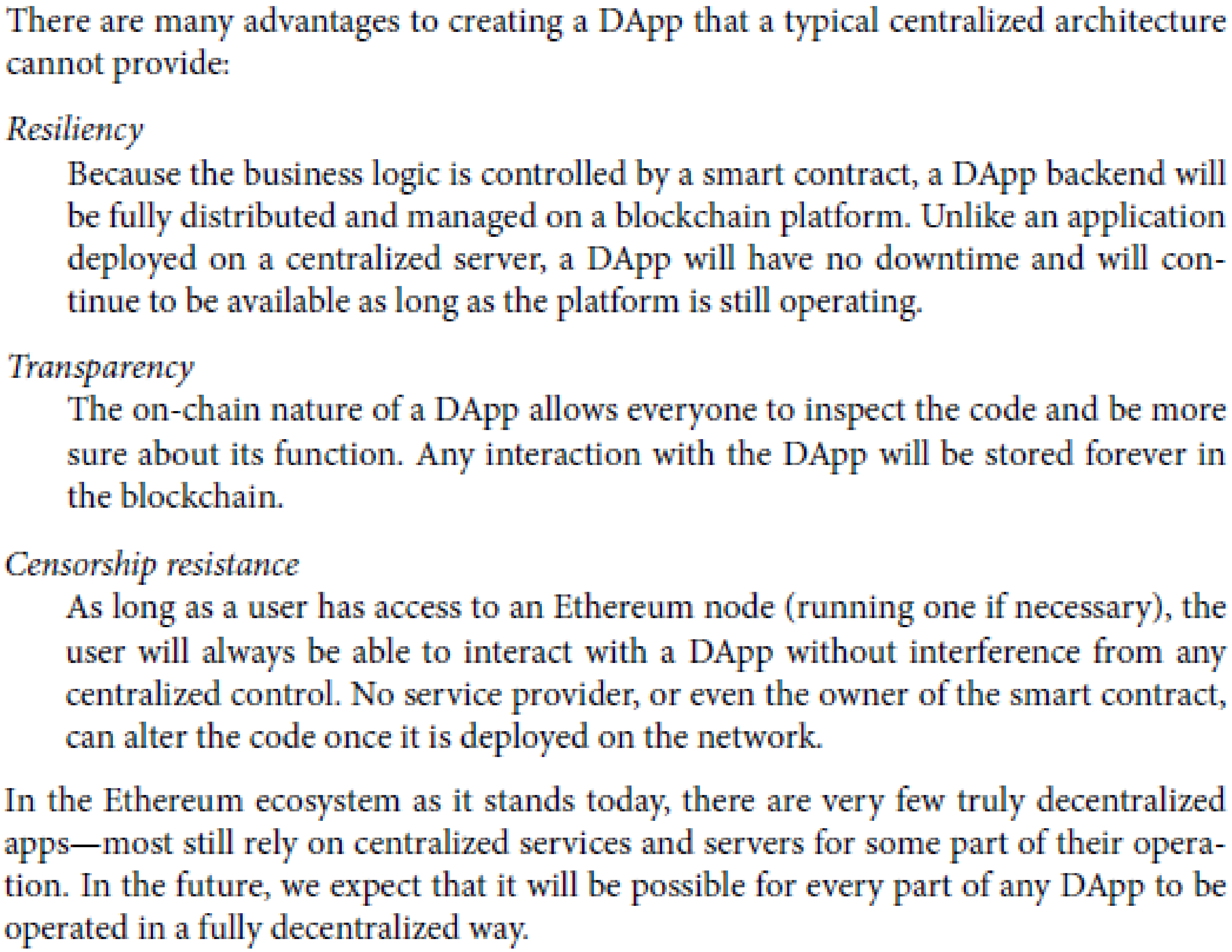
Consider all the possible aspects of an application that may be decentralized:

- Backend software (application logic)
- Frontend software
- Data storage
- Message communications
- Name resolution

Each of these can be somewhat centralized or somewhat decentralized. For example, a frontend can be developed as a web app that runs on a centralized server, or as a mobile app that runs on your device. The backend and storage can be on private servers and proprietary databases, or you can use a smart contract and P2P storage.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

DApp Advantages



There are many advantages to creating a DApp that a typical centralized architecture cannot provide:

Resiliency

Because the business logic is controlled by a smart contract, a DApp backend will be fully distributed and managed on a blockchain platform. Unlike an application deployed on a centralized server, a DApp will have no downtime and will continue to be available as long as the platform is still operating.

Transparency

The on-chain nature of a DApp allows everyone to inspect the code and be more sure about its function. Any interaction with the DApp will be stored forever in the blockchain.

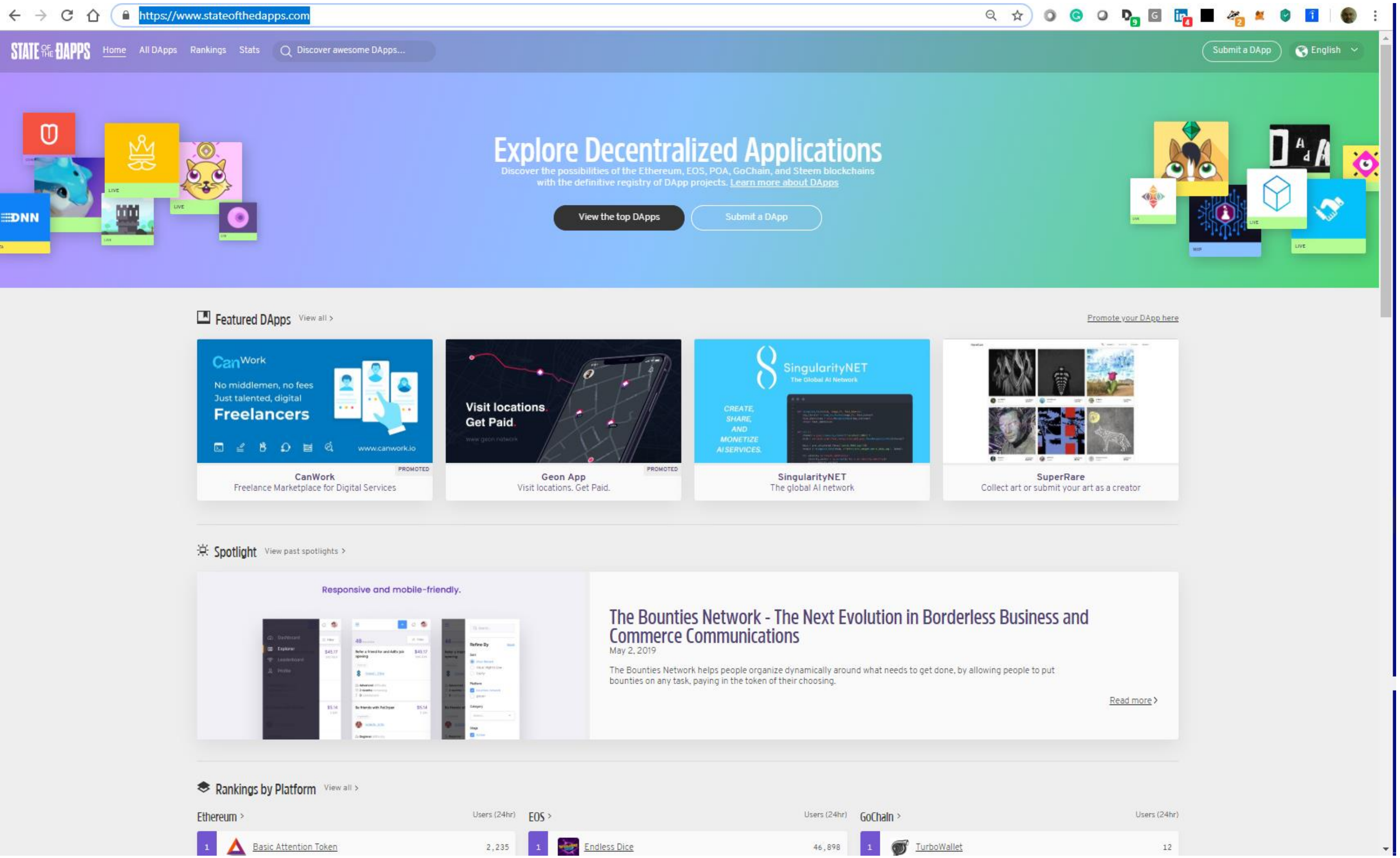
Censorship resistance

As long as a user has access to an Ethereum node (running one if necessary), the user will always be able to interact with a DApp without interference from any centralized control. No service provider, or even the owner of the smart contract, can alter the code once it is deployed on the network.

In the Ethereum ecosystem as it stands today, there are very few truly decentralized apps—most still rely on centralized services and servers for some part of their operation. In the future, we expect that it will be possible for every part of any DApp to be operated in a fully decentralized way.

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

State of the DApps



Explore Decentralized Applications

Discover the possibilities of the Ethereum, EOS, POA, GoChain, and Steem blockchains with the definitive registry of DApp projects. Learn more about DApps

Featured DApps View all >

Promote your DApp here

CanWork	Freelance Marketplace for Digital Services
Geon App	Visit locations. Get Paid.
SingularityNET	The global AI network
SuperRare	Collect art or submit your art as a creator

Spotlight View past spotlights >

The Bounties Network - The Next Evolution in Borderless Business and Commerce Communications

May 2, 2019

The Bounties Network helps people organize dynamically around what needs to get done, by allowing people to put bounties on any task, paying in the token of their choosing.

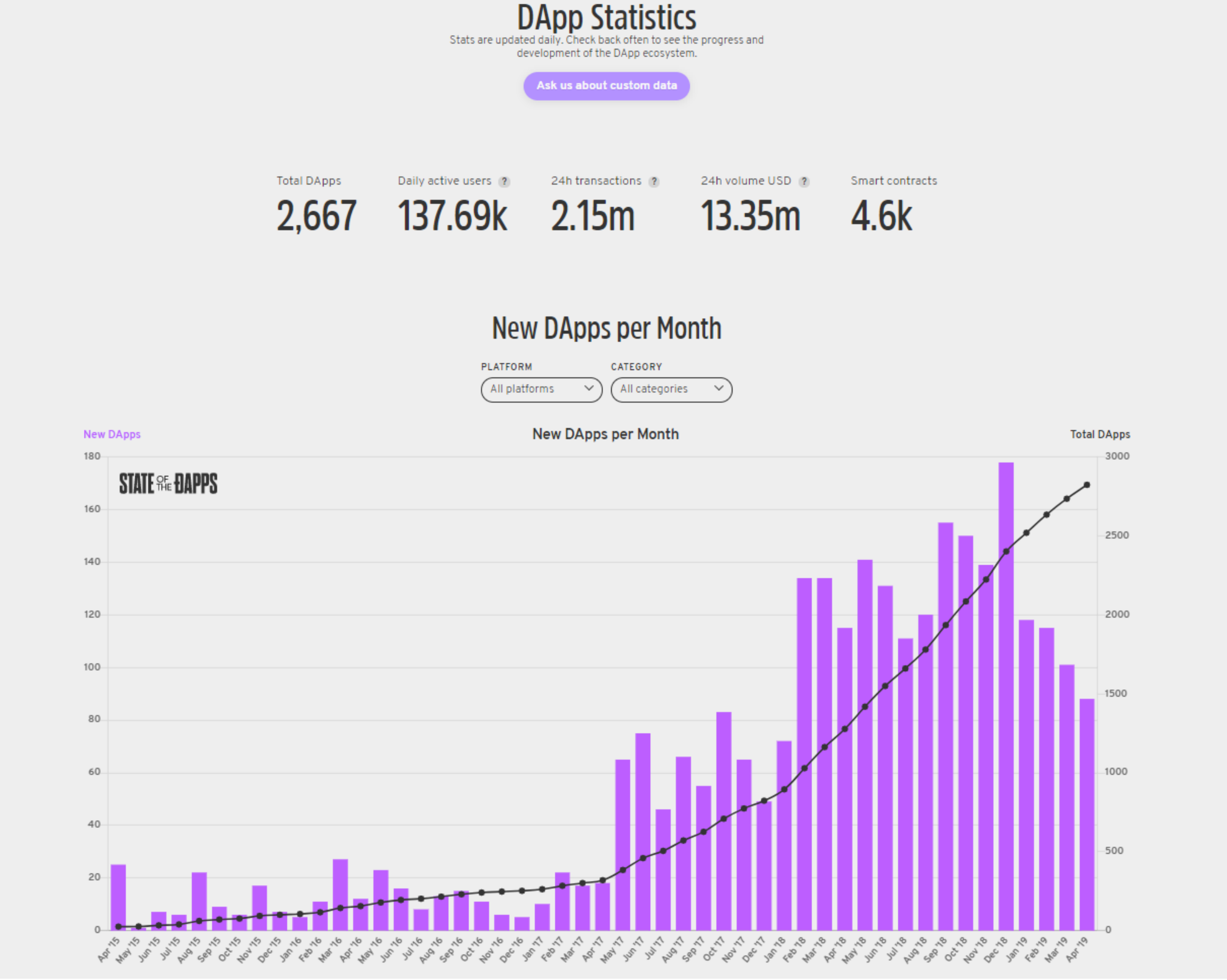
Read more >

Rankings by Platform View all >

Ethereum >	Users (24hr)	EOS >	Users (24hr)	GoChain >	Users (24hr)
1 Basic Attention Token	2,235	1 Endless Dice	46,898	1 TurboWallet	12

Source: https://www.stateofthedapps.com/

State of the DApps



DApp Statistics

Stats are updated daily. Check back often to see the progress and development of the DApp ecosystem.

Ask us about custom data

Total DApps	Daily active users ?	24h transactions ?	24h volume USD ?	Smart contracts
2,667	137.69k	2.15m	13.35m	4.6k

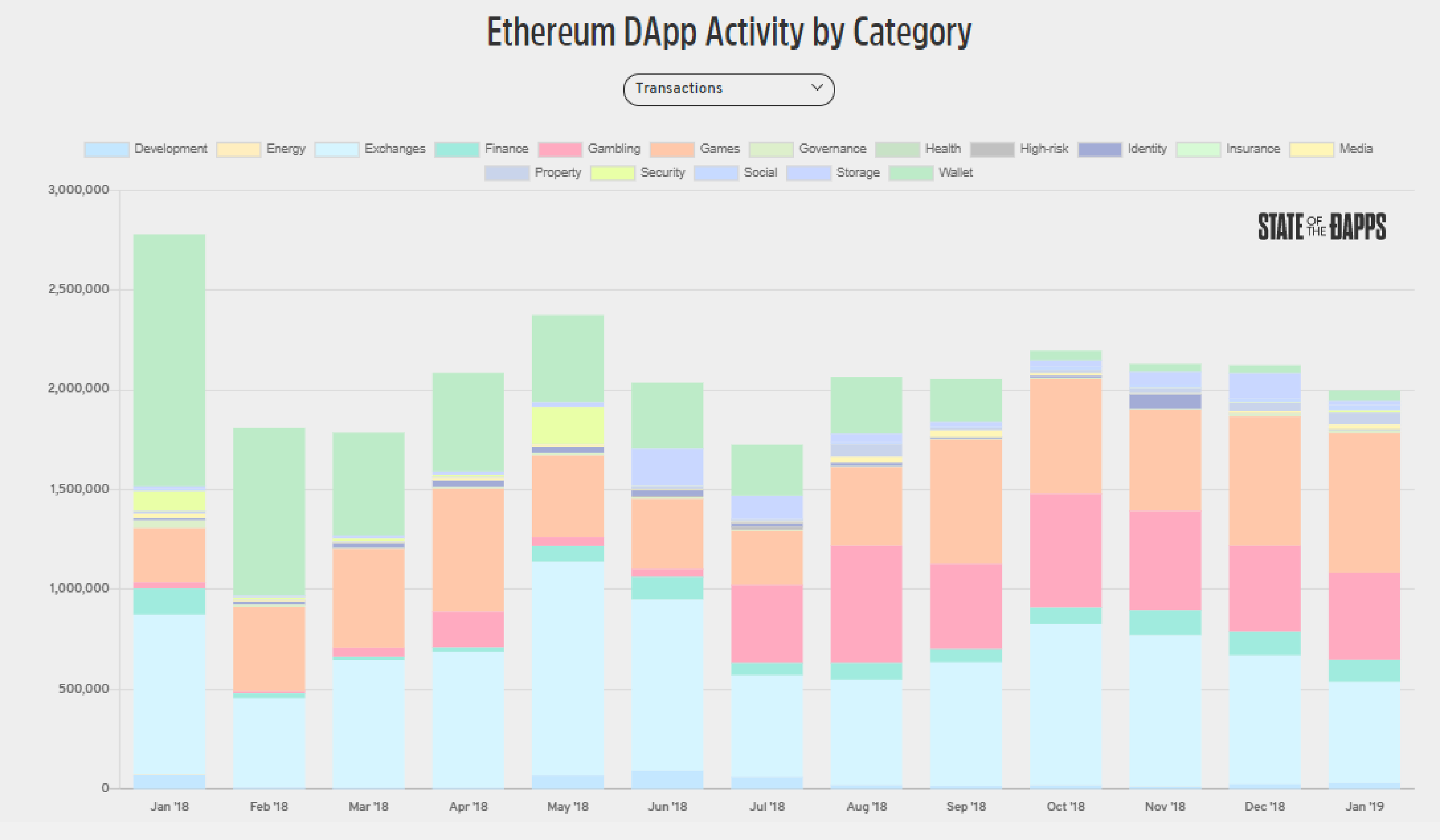
New DApps per Month

PLATFORM: All platforms

CATEGORY: All categories

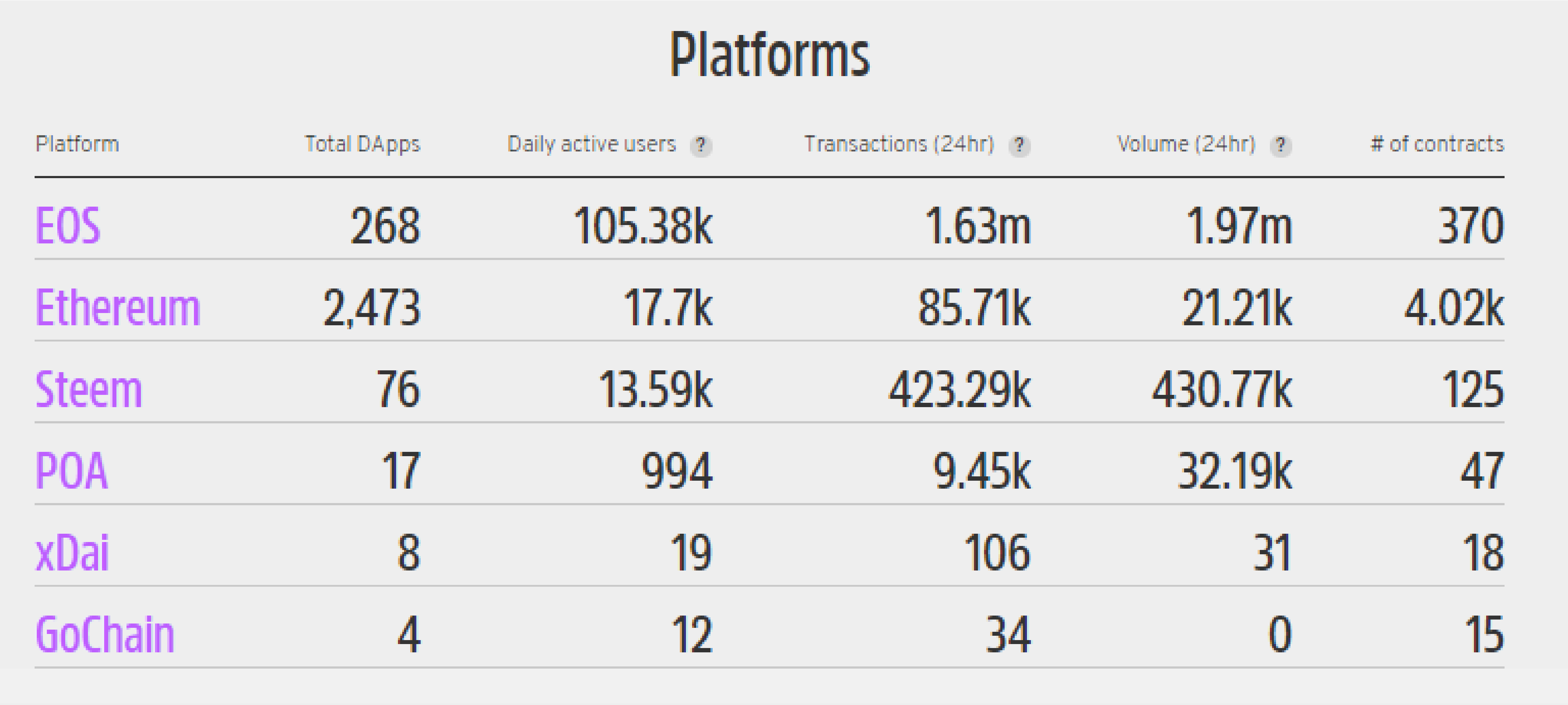
Source: https://www.stateofthedapps.com/

State of the DApps



Source: https://www.stateofthedapps.com/

State of the DApps

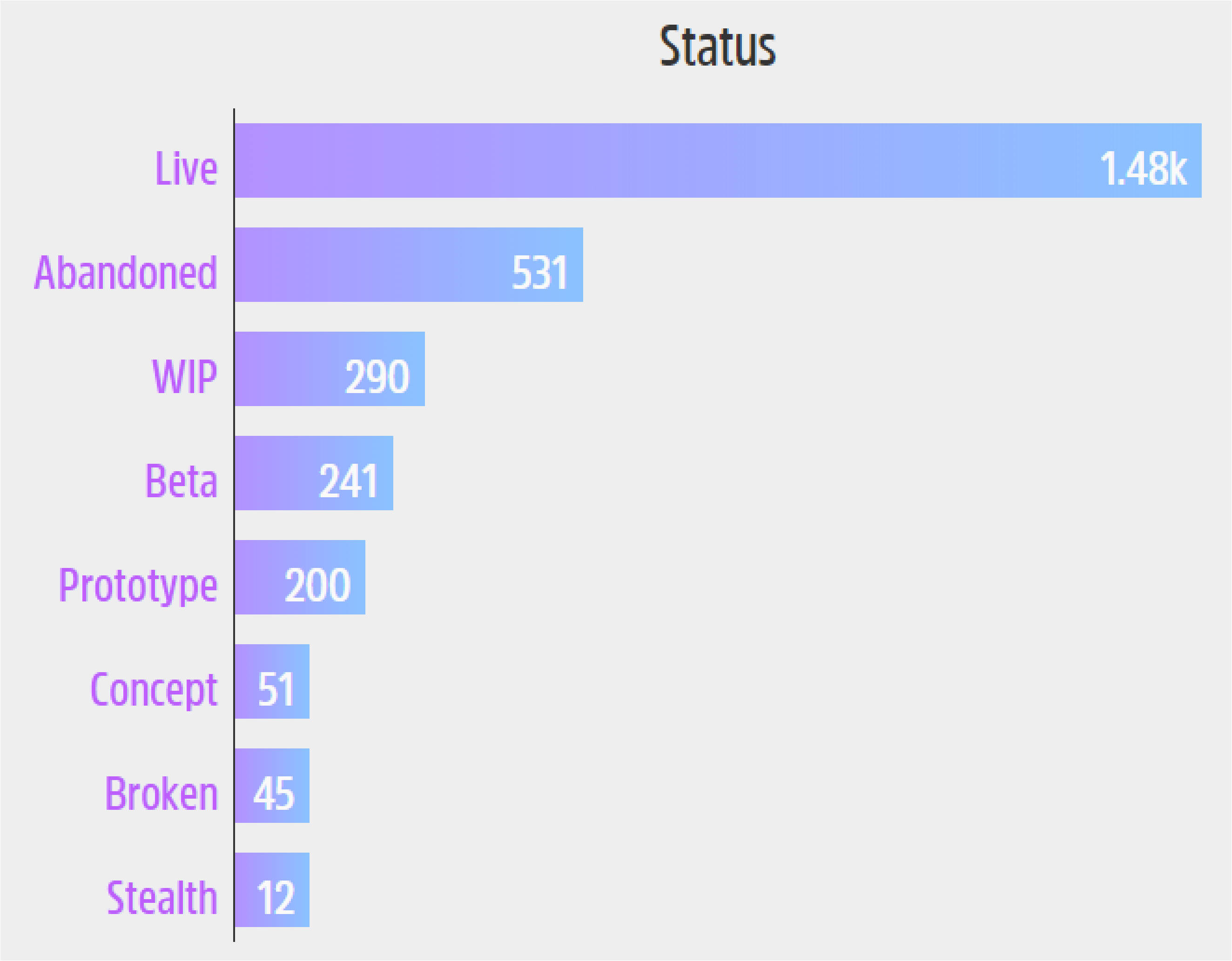


Platforms

Platform	Total DApps	Daily active users ?	Transactions (24hr) ?	Volume (24hr) ?	# of contracts
EOS	268	105.38k	1.63m	1.97m	370
Ethereum	2,473	17.7k	85.71k	21.21k	4.02k
Steem	76	13.59k	423.29k	430.77k	125
POA	17	994	9.45k	32.19k	47
xDai	8	19	106	31	18
GoChain	4	12	34	0	15

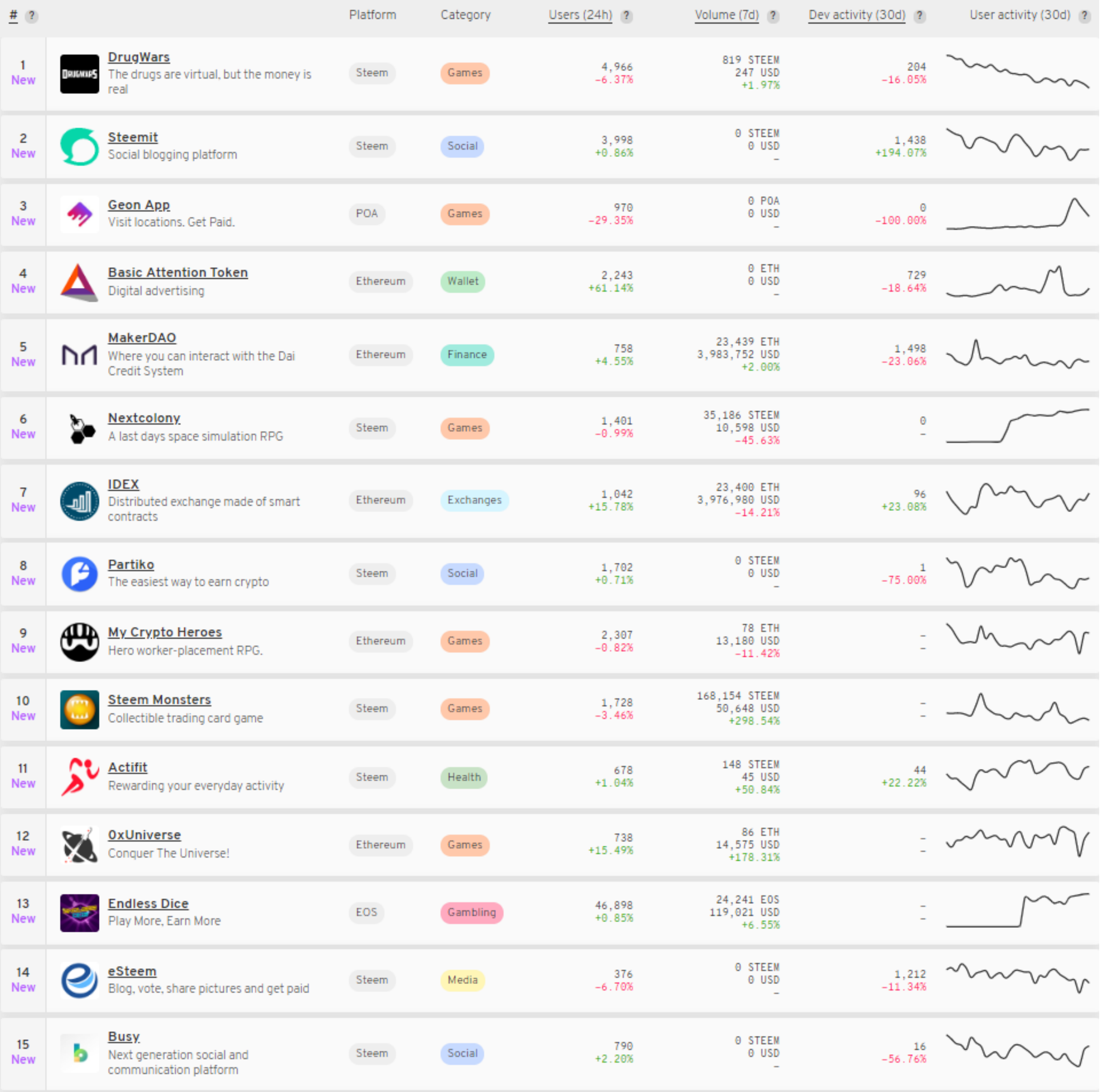
Source: https://www.stateofthedapps.com/

State of the DApps



Source: https://www.stateofthedapps.com/

State of the DApps



#		Platform	Category	Users (24h)	Volume (7d)	Dev activity (30d)	User activity (30d)
1 New	DrugWars The drugs are virtual, but the money is real	Steem	Games	4,966 -6.37%	819 STEEM 247 USD +1.97%	204 -16.05%	
2 New	Steemit Social blogging platform	Steem	Social	3,998 +0.86%	0 STEEM 0 USD -	1,438 +194.07%	
3 New	Geon App Visit locations. Get Paid.	POA	Games	970 -29.35%	0 POA 0 USD -	0 -100.00%	
4 New	Basic Attention Token Digital advertising	Ethereum	Wallet	2,243 +61.14%	0 ETH 0 USD -	729 -18.64%	
5 New	MakerDAO Where you can interact with the Dai Credit System	Ethereum	Finance	758 +4.55%	23,439 ETH 3,983,752 USD +2.00%	1,498 -23.06%	
6 New	Nextcolony A last days space simulation RPG	Steem	Games	1,401 -0.99%	35,186 STEEM 10,598 USD -45.63%	0 -	
7 New	IDEX Distributed exchange made of smart contracts	Ethereum	Exchanges	1,042 +15.78%	23,400 ETH 3,976,980 USD -14.21%	96 +23.08%	
8 New	Partiko The easiest way to earn crypto	Steem	Social	1,702 +0.71%	0 STEEM 0 USD -	1 -75.00%	
9 New	My Crypto Heroes Hero worker-placement RPG.	Ethereum	Games	2,307 -0.82%	78 ETH 13,180 USD -11.42%	- -	
10 New	Steem Monsters Collectible trading card game	Steem	Games	1,728 -3.46%	168,154 STEEM 50,648 USD +298.54%	- -	
11 New	Actifit Rewarding your everyday activity	Steem	Health	678 +1.04%	148 STEEM 45 USD +50.84%	44 +22.22%	
12 New	0xUniverse Conquer The Universe!	Ethereum	Games	738 +15.49%	86 ETH 14,575 USD +178.31%	- -	
13 New	Endless Dice Play More, Earn More	EOS	Gambling	46,898 +0.85%	24,241 EOS 119,021 USD +6.55%	- -	
14 New	eSteem Blog, vote, share pictures and get paid	Steem	Media	376 -6.70%	0 STEEM 0 USD -	1,212 -11.34%	
15 New	Busy Next generation social and communication platform	Steem	Social	790 +2.20%	0 STEEM 0 USD -	16 -56.76%	

Source: https://www.stateofthedapps.com/

Example Code Files

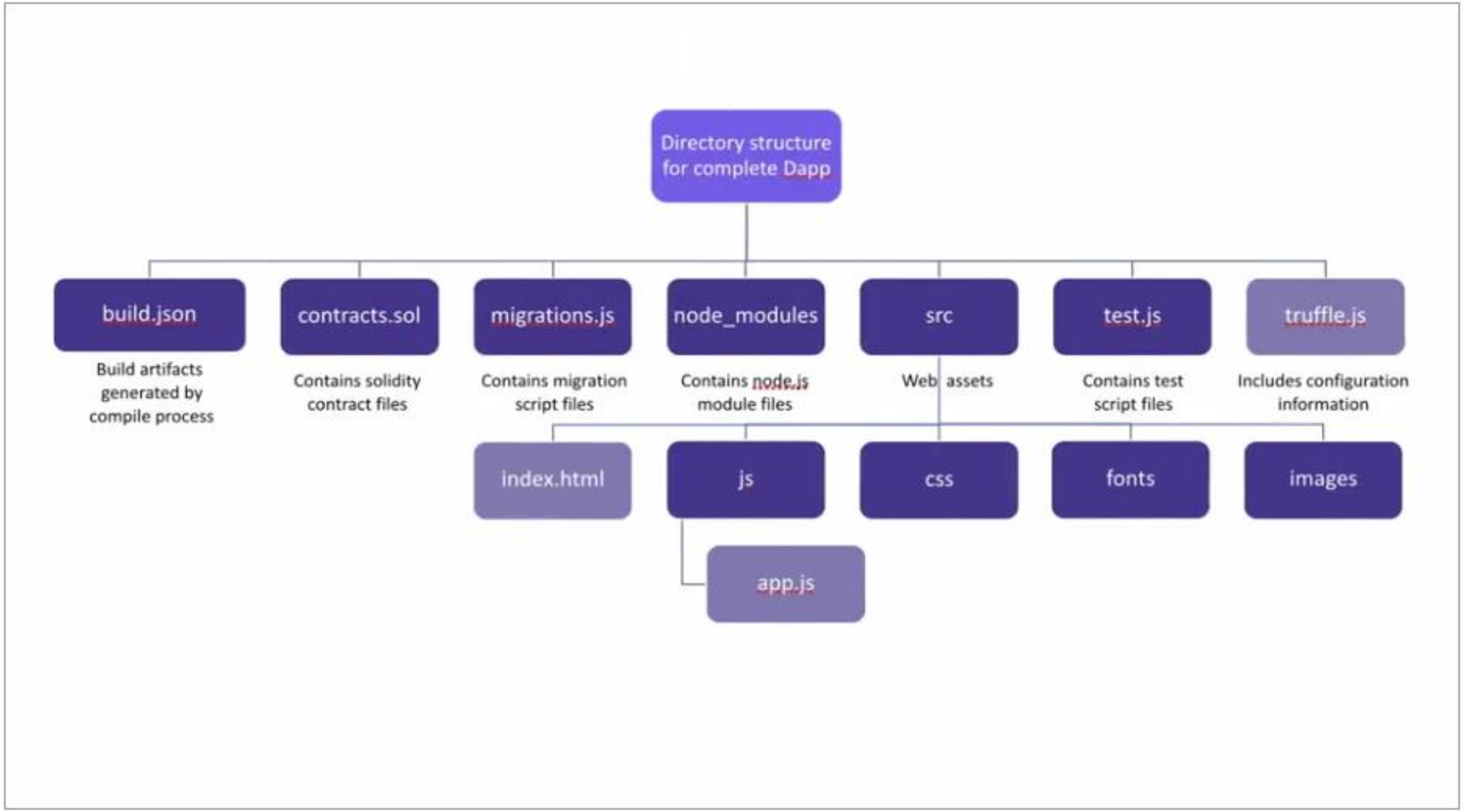
Example Code Files

Number	Workshop Topic	Source File	Location on OneDrive at 00 2019 ISACA CACS Blockchain Workshop
1	3	Greeter.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 01
2	3	Ballot.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 02
3	3	SimpleCoin.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 03
4	4	Faucet.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 04
5	4	Coin.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 05
6	6	AuctionRepository.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 06
7	6	DeedRepository.sol	00 Day 02 Materials | 00 Blockchain Workshop DApp Examples | 06

Topic 3: Example DApp using Truffle, HTML, CSS, Solidity, the EVM and Ethereum Blockchain

Truffle DApp Structure

Web Interface & Testing (Part 1) (Front-End Demo)



Directory structure for complete Dapp

build.json
Build artifacts generated by compile process

contracts.sol
Contains solidity contract files

migrations.js
Contains migration script files

node_modules
Contains node.js module files

src
Web assets

test.js
Contains test script files

truffle.js
Includes configuration information

index.html

js

app.js

css

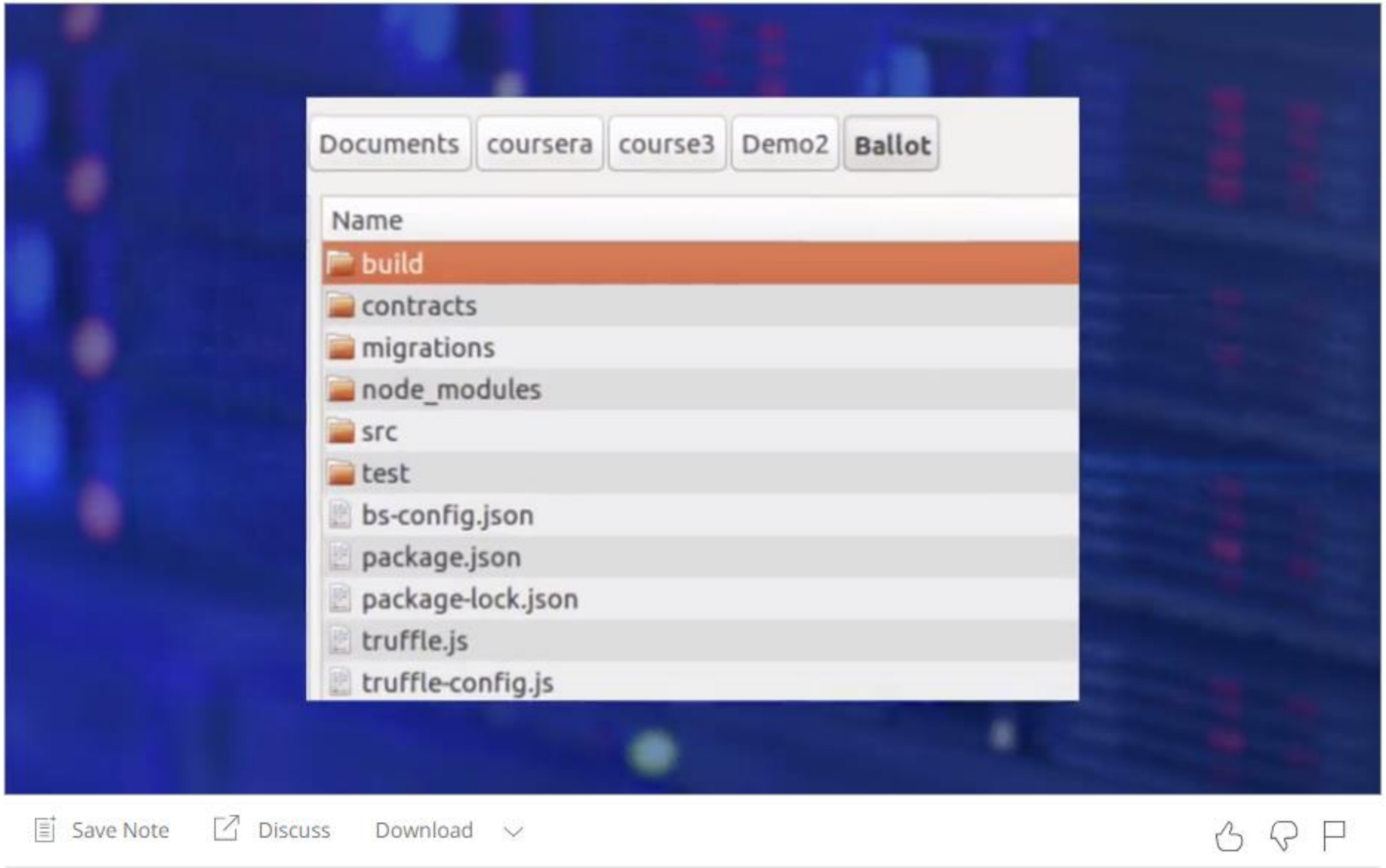
fonts

images

Source: Coursera – DApp Development Course

Truffle DApp Structure

Web Interface & Testing (Part 1) (Front-End Demo)



Documents | coursera | course3 | Demo2 | **Ballot**

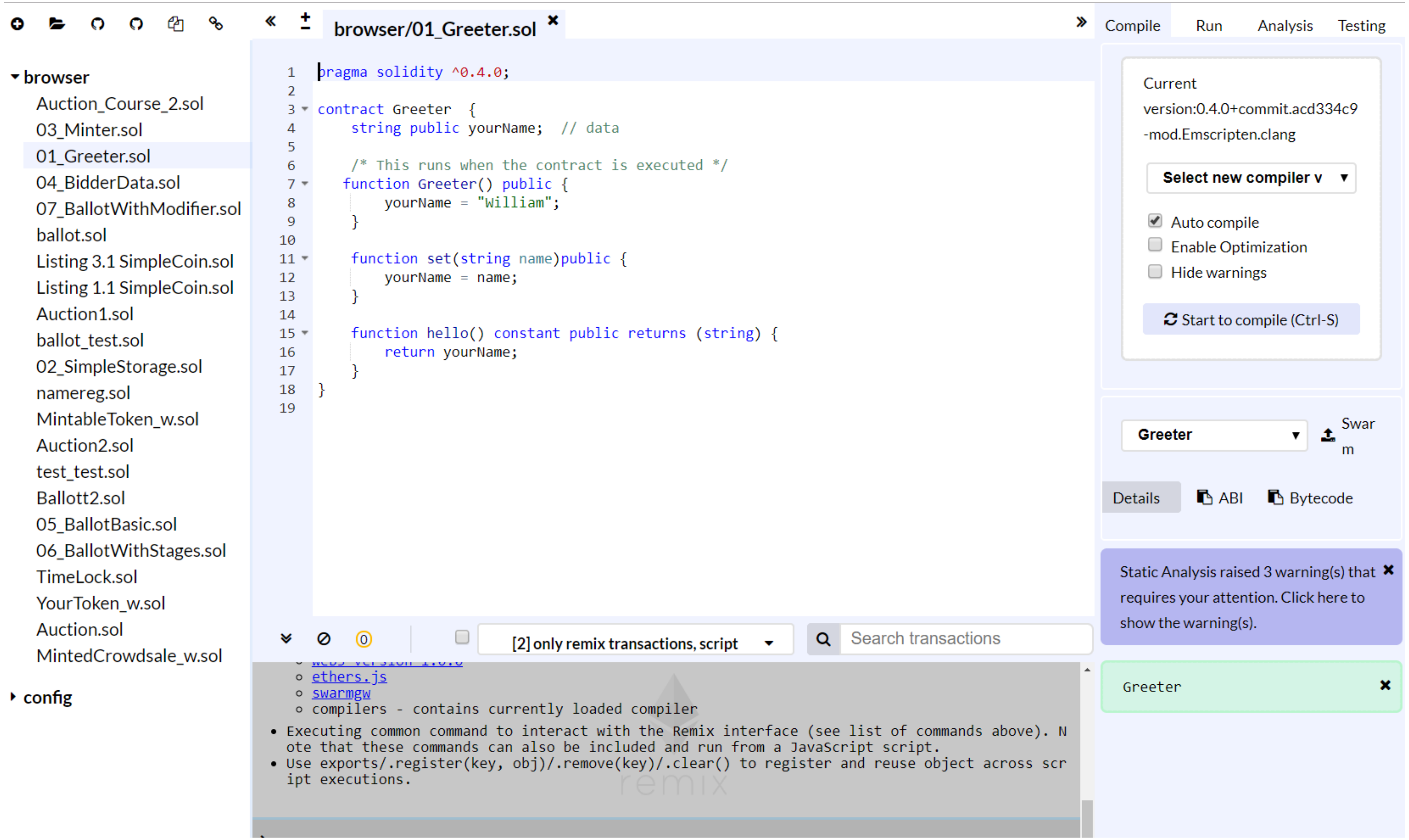
Name

- build
- contracts
- migrations
- node_modules
- src
- test
- bs-config.json
- package.json
- package-lock.json
- truffle.js
- truffle-config.js

Source: Coursera – DApp Development Course

Example 1

Greeter



browser/01_Greeter.sol

```
pragma solidity ^0.4.0;

contract Greeter  {
    string public yourName;  // data

    /* This runs when the contract is executed */
   function Greeter() public {
        yourName = "William";
    }

    function set(string name)public {
        yourName = name;
    }

    function hello() constant public returns (string) {
        return yourName;
    }
}

```

- browser
  - Auction_Course_2.sol
  - 03_Minter.sol
  - 01_Greeter.sol
  - 04_BidderData.sol
  - 07_BallotWithModifier.sol
  - ballot.sol
  - Listing 3.1 SimpleCoin.sol
  - Listing 1.1 SimpleCoin.sol
  - Auction1.sol
  - ballot_test.sol
  - 02_SimpleStorage.sol
  - namereg.sol
  - MintableToken_w.sol
  - Auction2.sol
  - test_test.sol
  - Ballott2.sol
  - 05_BallotBasic.sol
  - 06_BallotWithStages.sol
  - TimeLock.sol
  - YourToken_w.sol
  - Auction.sol
  - MintedCrowdsale_w.sol
- config

Compile | Run | Analysis | Testing

Current version:0.4.0+commit.acd334c9-mod.Emscripten.clang

Select new compiler v

☑ Auto compile
☐ Enable Optimization
☐ Hide warnings

Start to compile (Ctrl-S)

Greeter ▾ | Swarm

Details | ABI | Bytecode

Static Analysis raised 3 warning(s) that requires your attention. Click here to show the warning(s).

Greeter

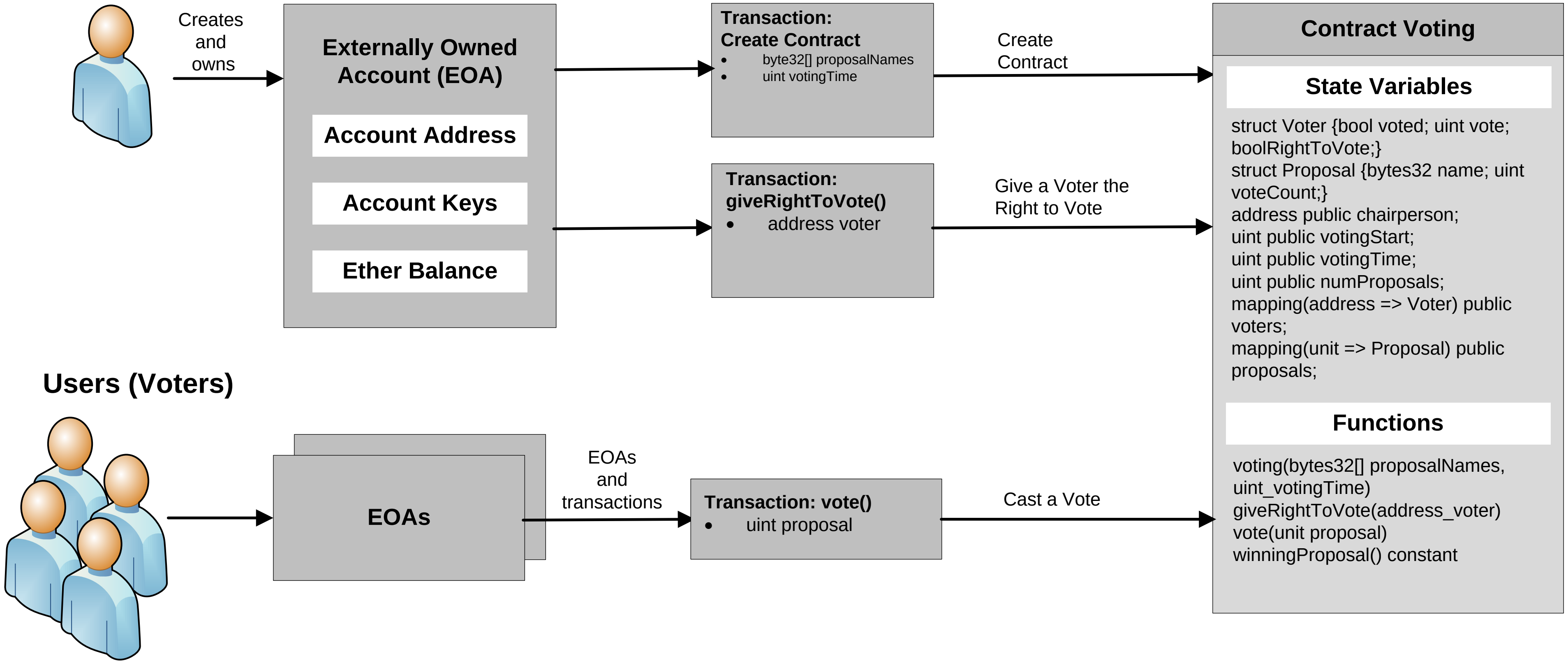
☐ [2] only remix transactions, script | Search transactions

```
    o web3 version 1.0.0
    o ethers.js
    o swarmgw
    o compilers - contains currently loaded compiler
  • Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
  • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Source: Coursera – Smart Contract Course

Example 2 – Voting DApp

Voting DApp



Creates and owns

Externally Owned Account (EOA)

Account Address

Account Keys

Ether Balance

Transaction: Create Contract
- byte32[] proposalNames
- uint votingTime

Create Contract

Transaction: giveRightToVote()
- address voter

Give a Voter the Right to Vote

Contract Voting

State Variables

struct Voter {bool voted; uint vote; boolRightToVote;}
struct Proposal {bytes32 name; uint voteCount;}
address public chairperson;
uint public votingStart;
uint public votingTime;
uint public numProposals;
mapping(address => Voter) public voters;
mapping(unit => Proposal) public proposals;

Functions

voting(bytes32[] proposalNames, uint_votingTime)
giveRightToVote(address_voter)
vote(unit proposal)
winningProposal() constant

Users (Voters)

EOAs

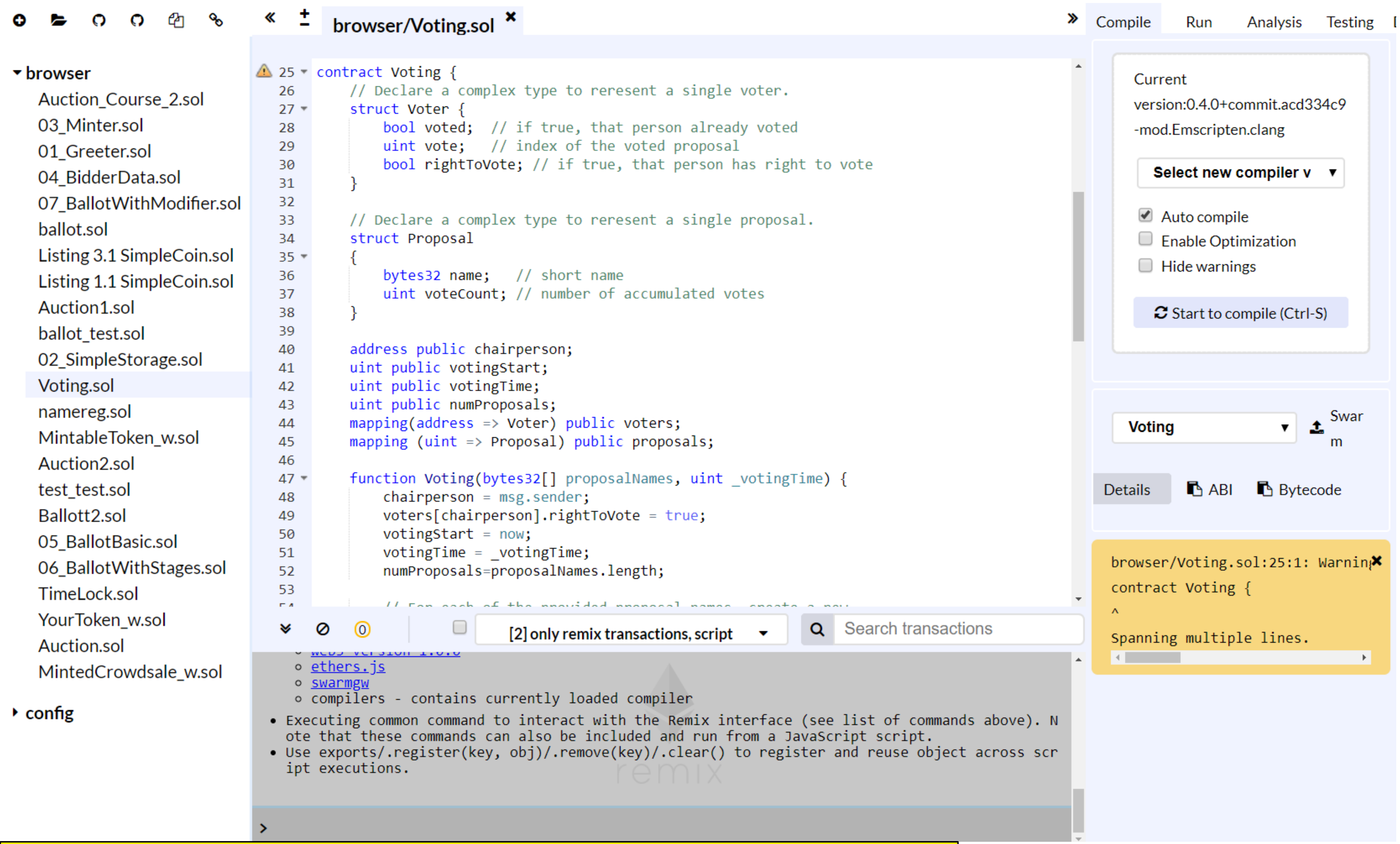
EOAs and transactions

Transaction: vote()
- uint proposal

Cast a Vote

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

Example 2 – Voting DApp



```
contract Voting {
    // Declare a complex type to reresent a single voter.
    struct Voter {
        bool voted;  // if true, that person already voted
        uint vote;   // index of the voted proposal
        bool rightToVote; // if true, that person has right to vote
    }

    // Declare a complex type to reresent a single proposal.
    struct Proposal
    {
        bytes32 name;   // short name
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;
    uint public votingStart;
    uint public votingTime;
    uint public numProposals;
    mapping(address => Voter) public voters;
    mapping (uint => Proposal) public proposals;

    function Voting(bytes32[] proposalNames, uint _votingTime) {
        chairperson = msg.sender;
        voters[chairperson].rightToVote = true;
        votingStart = now;
        votingTime = _votingTime;
        numProposals=proposalNames.length;

        // For each of the provided proposal names, create a new
```

Current version:0.4.0+commit.acd334c9-mod.Emscripten.clang

Select new compiler v

☑ Auto compile
□ Enable Optimization
□ Hide warnings

Start to compile (Ctrl-S)

Voting ▾ Swarm

Details ABI Bytecode

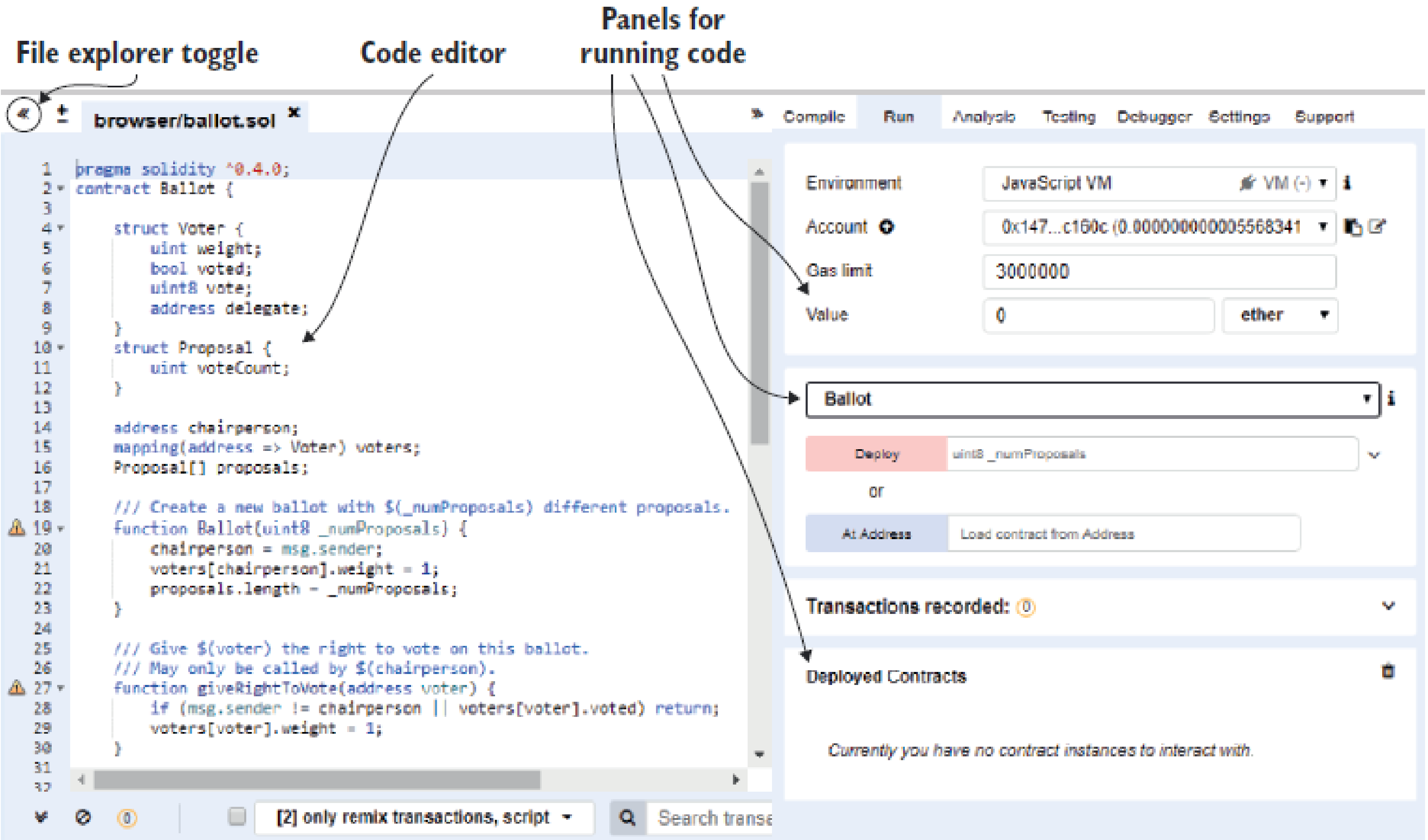
```
browser/Voting.sol:25:1: Warning
contract Voting {
^
Spanning multiple lines.
```

□ [2] only remix transactions, script ▾ Search transactions

```
   o web3 version 1.0.0
   o ethers.js
   o swarmgw
   o compilers - contains currently loaded compiler
 • Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
 • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Source: Blockchain Applications: A Hands-on Approach by Arsheep Bahga and Vijay Madisetti

Example 3 – ballot.sol



```
pragma solidity ^0.4.0;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        address delegate;
    }
    struct Proposal {
        uint voteCount;
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    /// Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        proposals.length = _numProposals;
    }

    /// Give $(voter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function giveRightToVote(address voter) {
        if (msg.sender != chairperson || voters[voter].voted) return;
        voters[voter].weight = 1;
    }
```

Environment: JavaScript VM — VM (-)

Account: 0x147...c160c (0.000000000005568341

Gas limit: 3000000

Value: 0 ether

Ballot

Deploy: uint8 _numProposals

or

At Address: Load contract from Address

Transactions recorded: 0

Deployed Contracts

Currently you have no contract instances to interact with.

Figure 1.10 Screenshot of the Remix opening screen, with the code on the left and the code execution panels on the right. I've hidden the file explorer by clicking the double arrow toggle at the top left.

Source: Roberto Infante, Building Ethereum DApps, 2019

Basic Truffle Steps

Reference: https://truffleframework.com/docs/truffle/getting-started/creating-a-project

Init

Compile

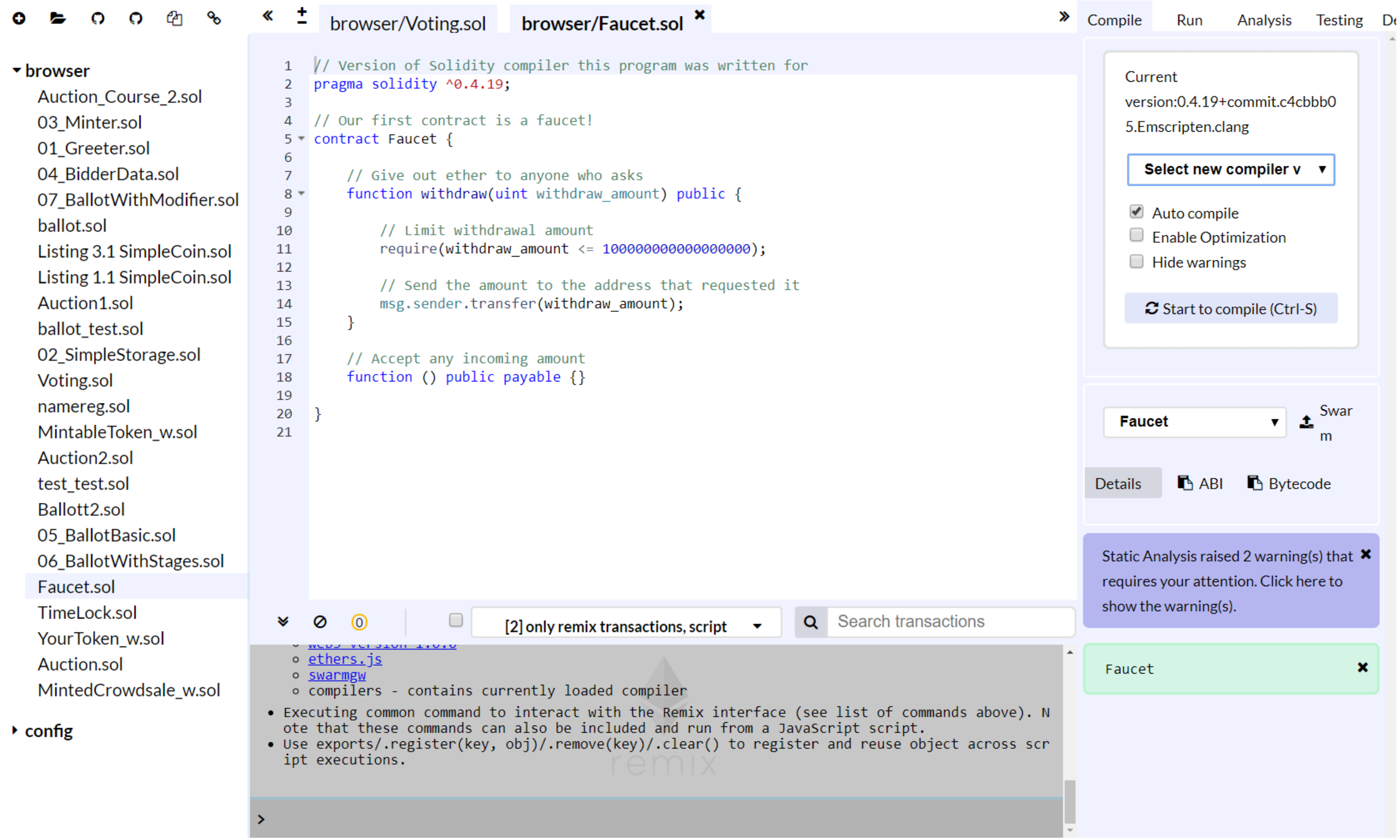
Deploy

Migrate

Run

Topic 4: Solidity and Ethereum Blockchain Fundamentals

Faucet



```
// Version of Solidity compiler this program was written for
pragma solidity ^0.4.19;

// Our first contract is a faucet!
contract Faucet {

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {

        // Limit withdrawal amount
        require(withdraw_amount <= 100000000000000000);

        // Send the amount to the address that requested it
        msg.sender.transfer(withdraw_amount);
    }

    // Accept any incoming amount
    function () public payable {}

}
```

Current version:0.4.19+commit.c4cbbb05.Emscripten.clang

☑ Auto compile
☐ Enable Optimization
☐ Hide warnings

Static Analysis raised 2 warning(s) that requires your attention. Click here to show the warning(s).

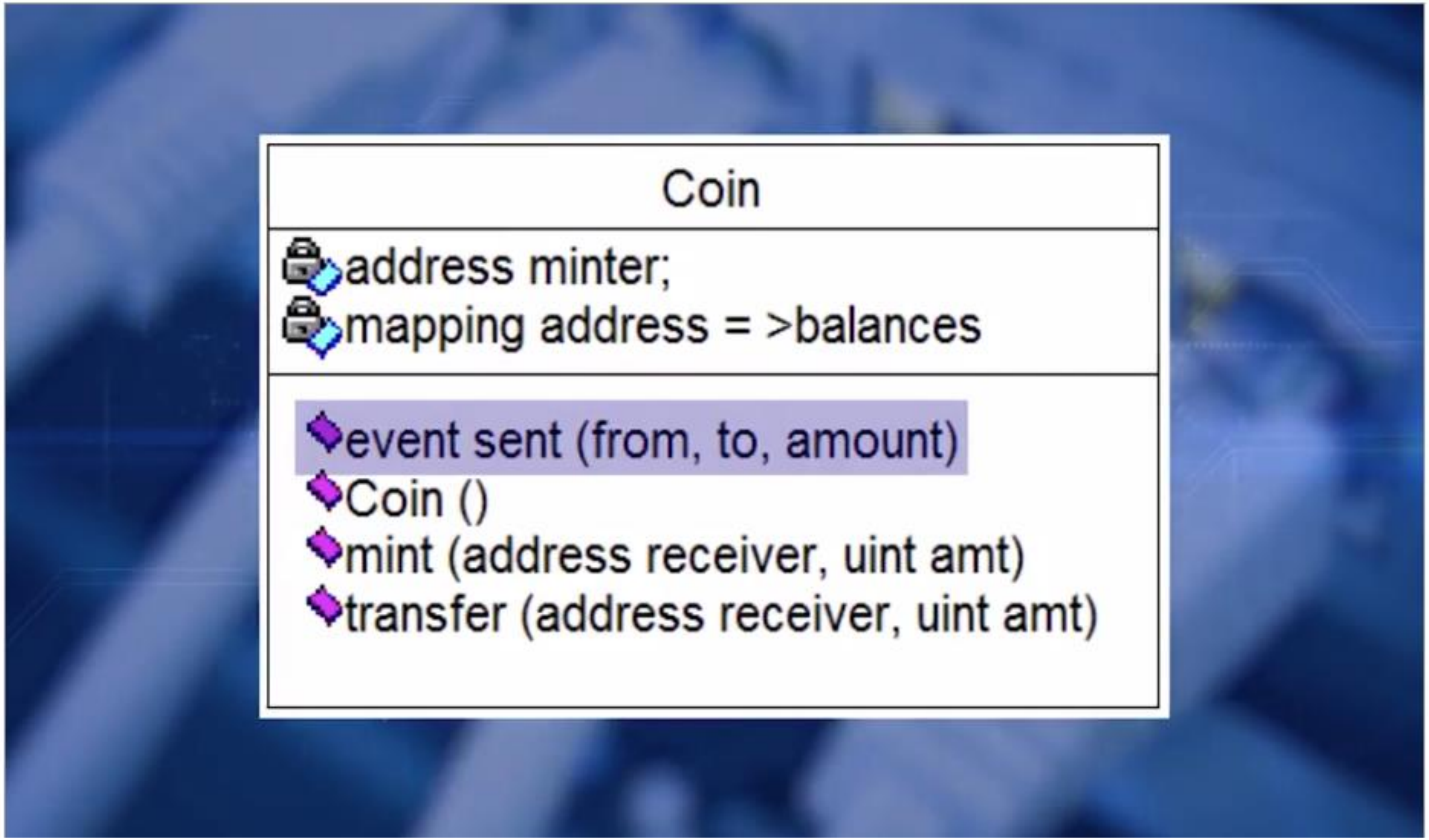
Faucet

```
    o ethers.js
    o swarmgw
    o compilers - contains currently loaded compiler
  • Executing common command to interact with the Remix interface (see list of commands above). N
    ote that these commands can also be included and run from a JavaScript script.
  • Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across scr
    ipt executions.
>
```

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

DApp Design - Coin

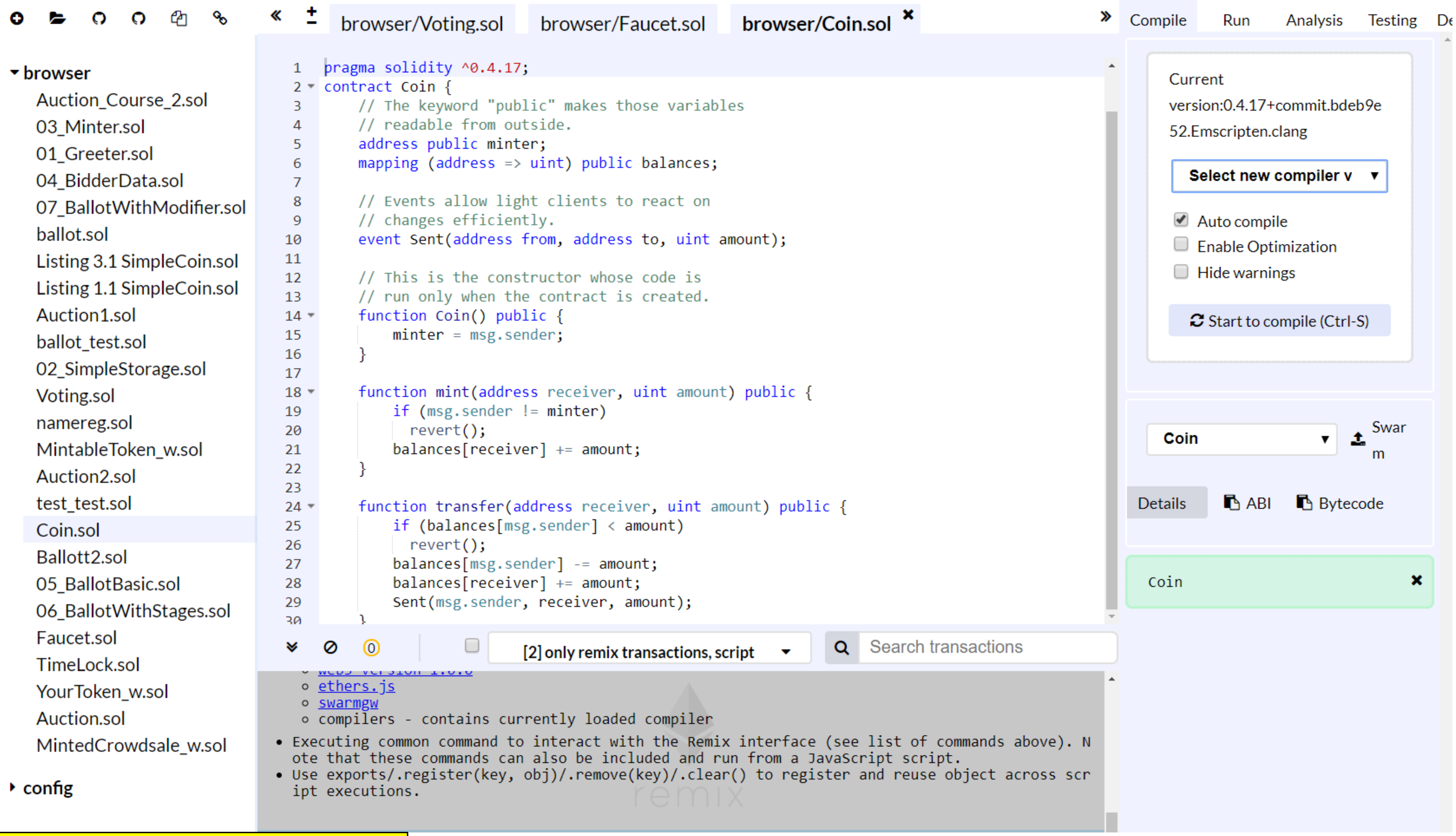
Event Handling (Part 1)



```
Coin
address minter;
mapping address = >balances
event sent (from, to, amount)
Coin ()
mint (address receiver, uint amt)
transfer (address receiver, uint amt)
```

Source: Coursera – DApp Development Course

Coin



```
pragma solidity ^0.4.17;
contract Coin {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    function Coin() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter)
          revert();
        balances[receiver] += amount;
    }

    function transfer(address receiver, uint amount) public {
        if (balances[msg.sender] < amount)
          revert();
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
    }
```

Current version:0.4.17+commit.bdeb9e52.Emscripten.clang

☑ Auto compile
☐ Enable Optimization
☐ Hide warnings

Coin

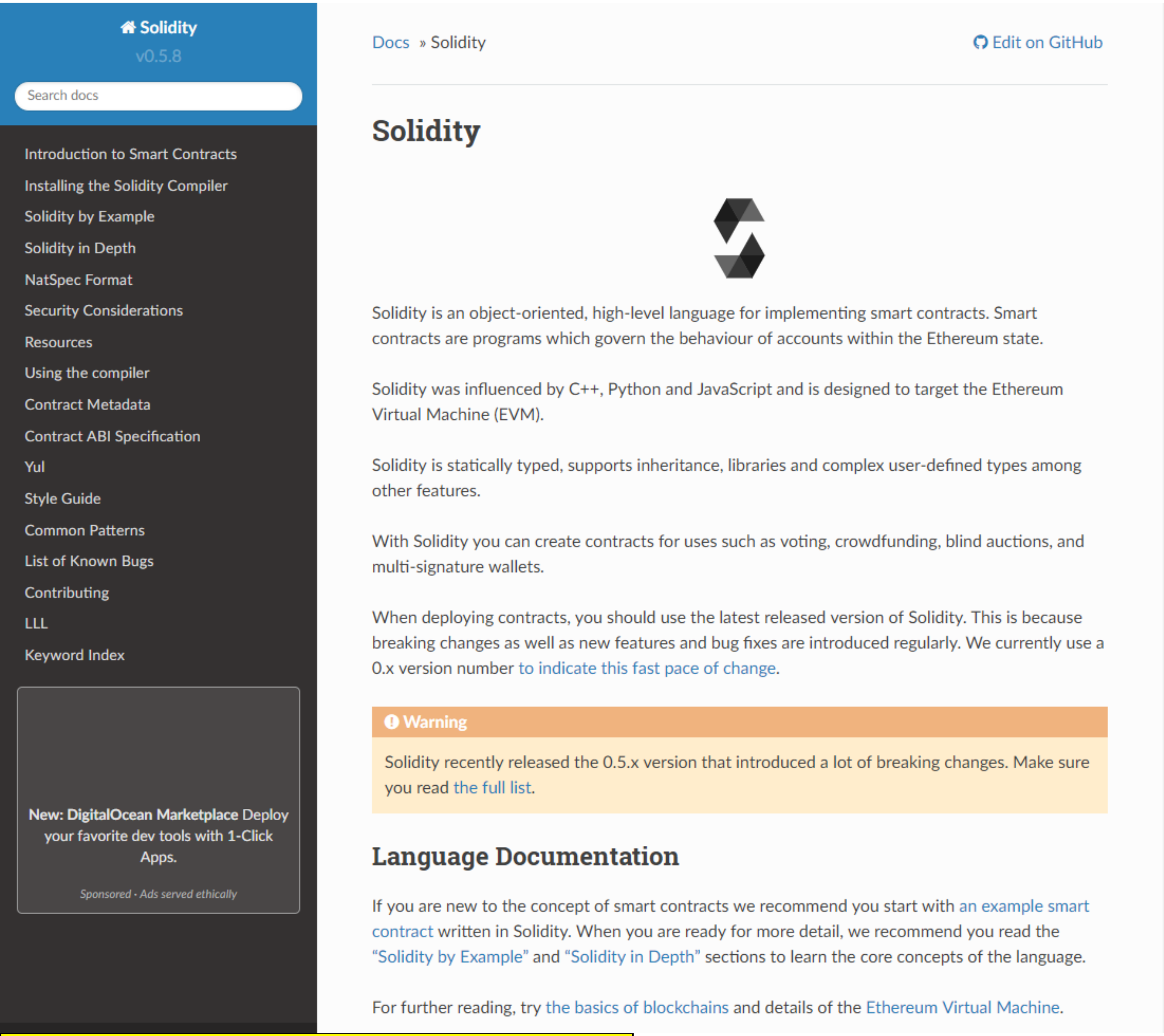
Coin

☐ [2] only remix transactions, script

```
    o web3 version 1.0.0
    o ethers.js
    o swarmgw
    o compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Source: Coursera – Smart Contract Course

Solidity Fundamentals



Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Smart contracts are programs which govern the behaviour of accounts within the Ethereum state.

Solidity was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).

Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

With Solidity you can create contracts for uses such as voting, crowdfunding, blind auctions, and multi-signature wallets.

When deploying contracts, you should use the latest released version of Solidity. This is because breaking changes as well as new features and bug fixes are introduced regularly. We currently use a 0.x version number to indicate this fast pace of change.

Warning

Solidity recently released the 0.5.x version that introduced a lot of breaking changes. Make sure you read the full list.

Language Documentation

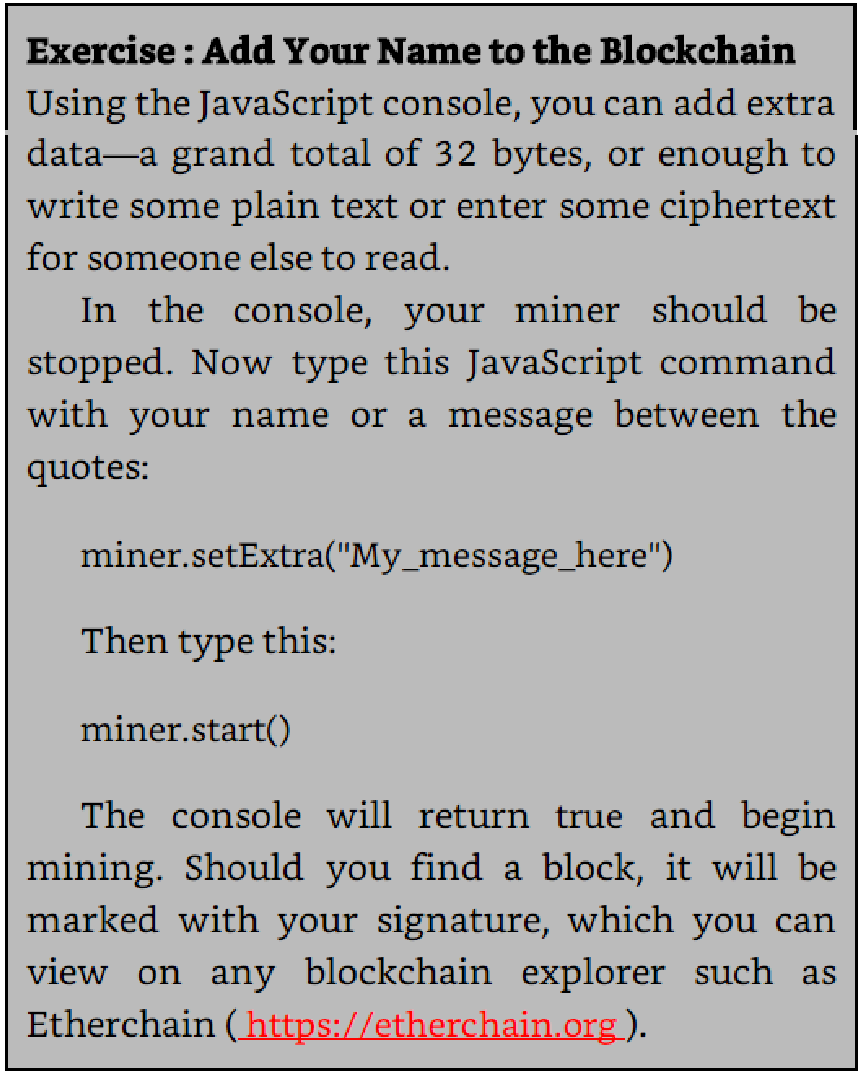
If you are new to the concept of smart contracts we recommend you start with an example smart contract written in Solidity. When you are ready for more detail, we recommend you read the "Solidity by Example" and "Solidity in Depth" sections to learn the core concepts of the language.

For further reading, try the basics of blockchains and details of the Ethereum Virtual Machine.

Source: https://solidity.readthedocs.io/en/v0.5.8/

Topic 5: Javascript and Ethereum Blockchain Fundamentals

Geth Workout



Exercise : Add Your Name to the Blockchain

Using the JavaScript console, you can add extra data—a grand total of 32 bytes, or enough to write some plain text or enter some ciphertext for someone else to read.

In the console, your miner should be stopped. Now type this JavaScript command with your name or a message between the quotes:

```
miner.setExtra("My_message_here")
```

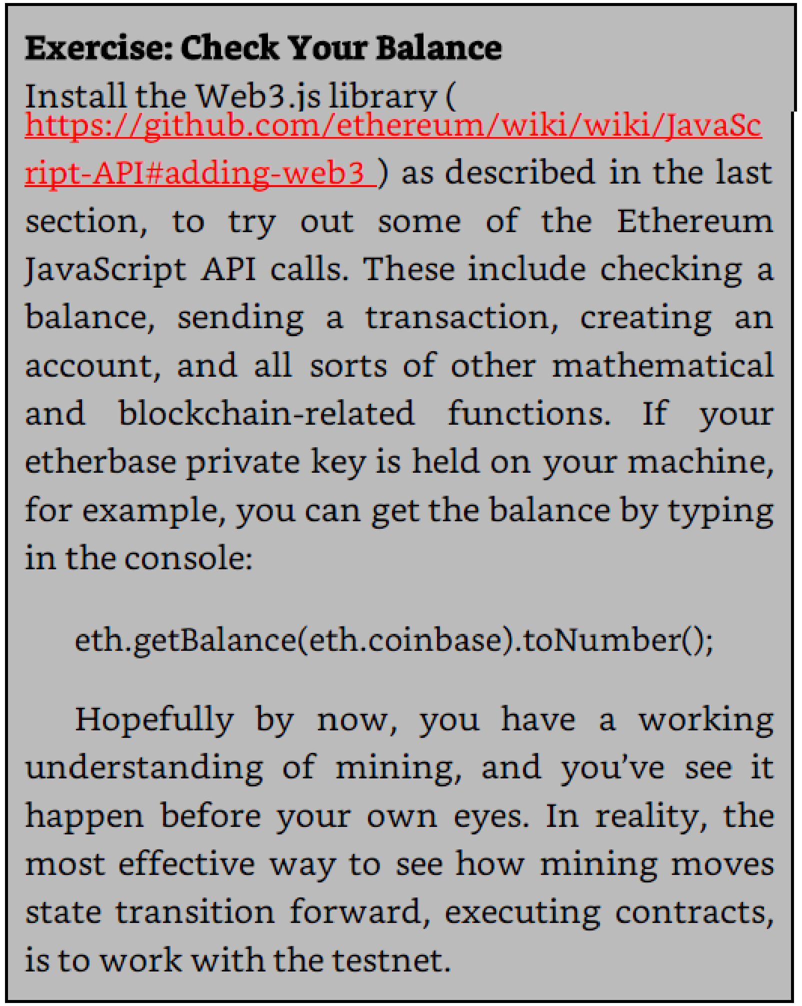
Then type this:

```
miner.start()
```

The console will return true and begin mining. Should you find a block, it will be marked with your signature, which you can view on any blockchain explorer such as Etherchain (https://etherchain.org).

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



Exercise: Check Your Balance

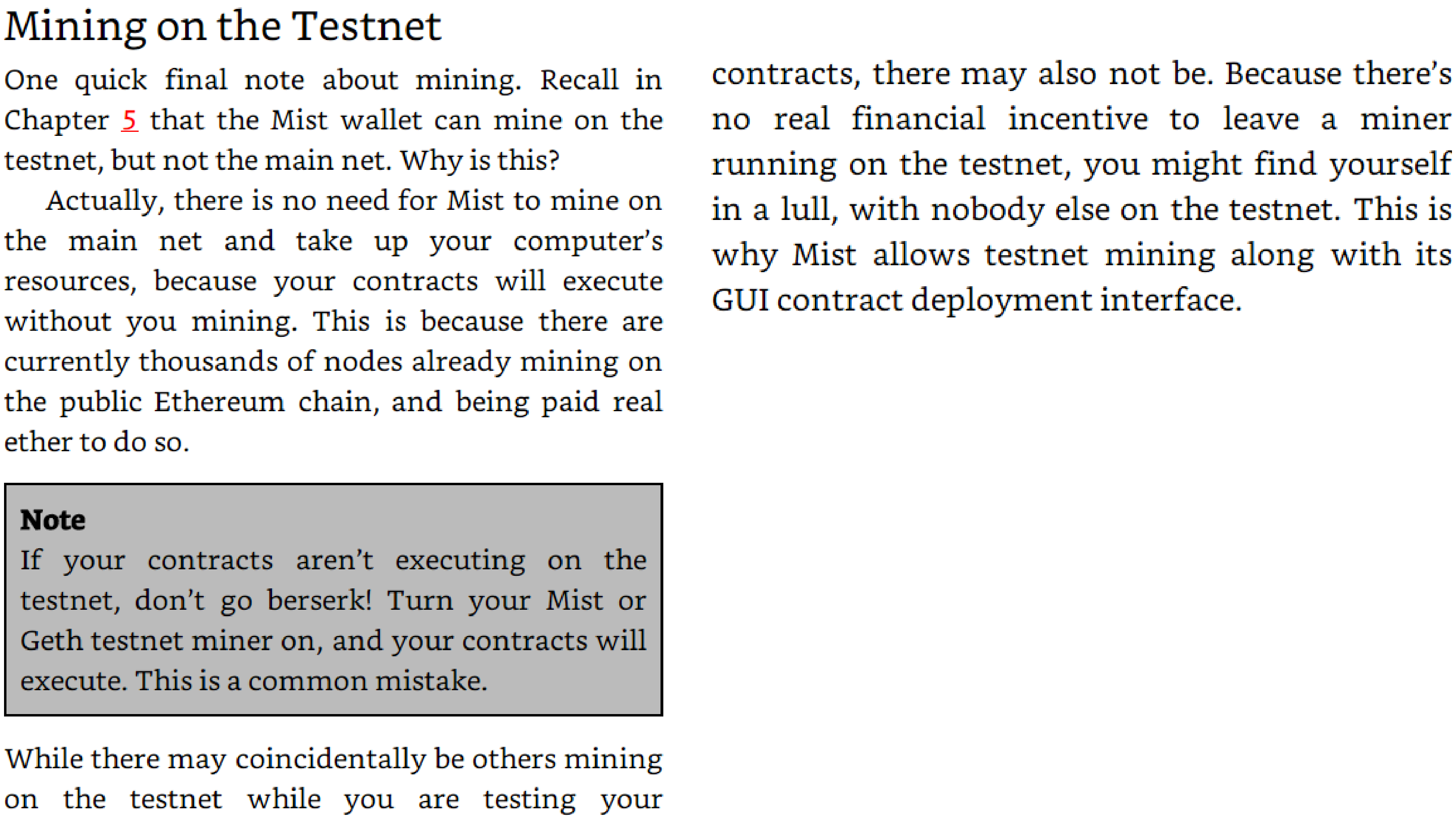
Install the Web3.js library (https://github.com/ethereum/wiki/wiki/JavaScript-API#adding-web3) as described in the last section, to try out some of the Ethereum JavaScript API calls. These include checking a balance, sending a transaction, creating an account, and all sorts of other mathematical and blockchain-related functions. If your etherbase private key is held on your machine, for example, you can get the balance by typing in the console:

```
eth.getBalance(eth.coinbase).toNumber();
```

Hopefully by now, you have a working understanding of mining, and you've see it happen before your own eyes. In reality, the most effective way to see how mining moves state transition forward, executing contracts, is to work with the testnet.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

Geth Workout



Mining on the Testnet

One quick final note about mining. Recall in Chapter 5 that the Mist wallet can mine on the testnet, but not the main net. Why is this?

Actually, there is no need for Mist to mine on the main net and take up your computer's resources, because your contracts will execute without you mining. This is because there are currently thousands of nodes already mining on the public Ethereum chain, and being paid real ether to do so.

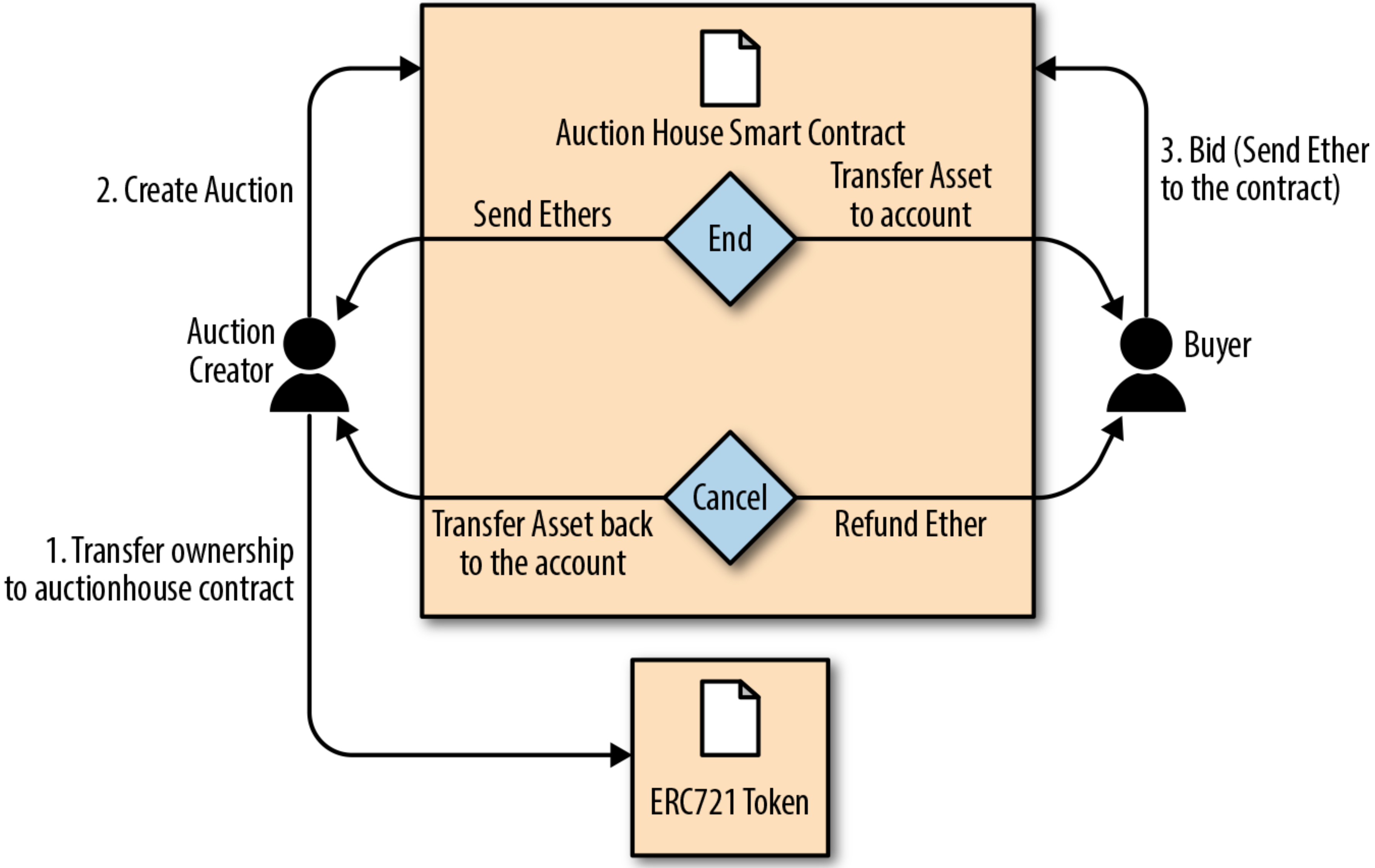
> **Note**
> If your contracts aren't executing on the testnet, don't go berserk! Turn your Mist or Geth testnet miner on, and your contracts will execute. This is a common mistake.

While there may coincidentally be others mining on the testnet while you are testing your contracts, there may also not be. Because there's no real financial incentive to leave a miner running on the testnet, you might find yourself in a lull, with nobody else on the testnet. This is why Mist allows testnet mining along with its GUI contract deployment interface.

Source: https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console

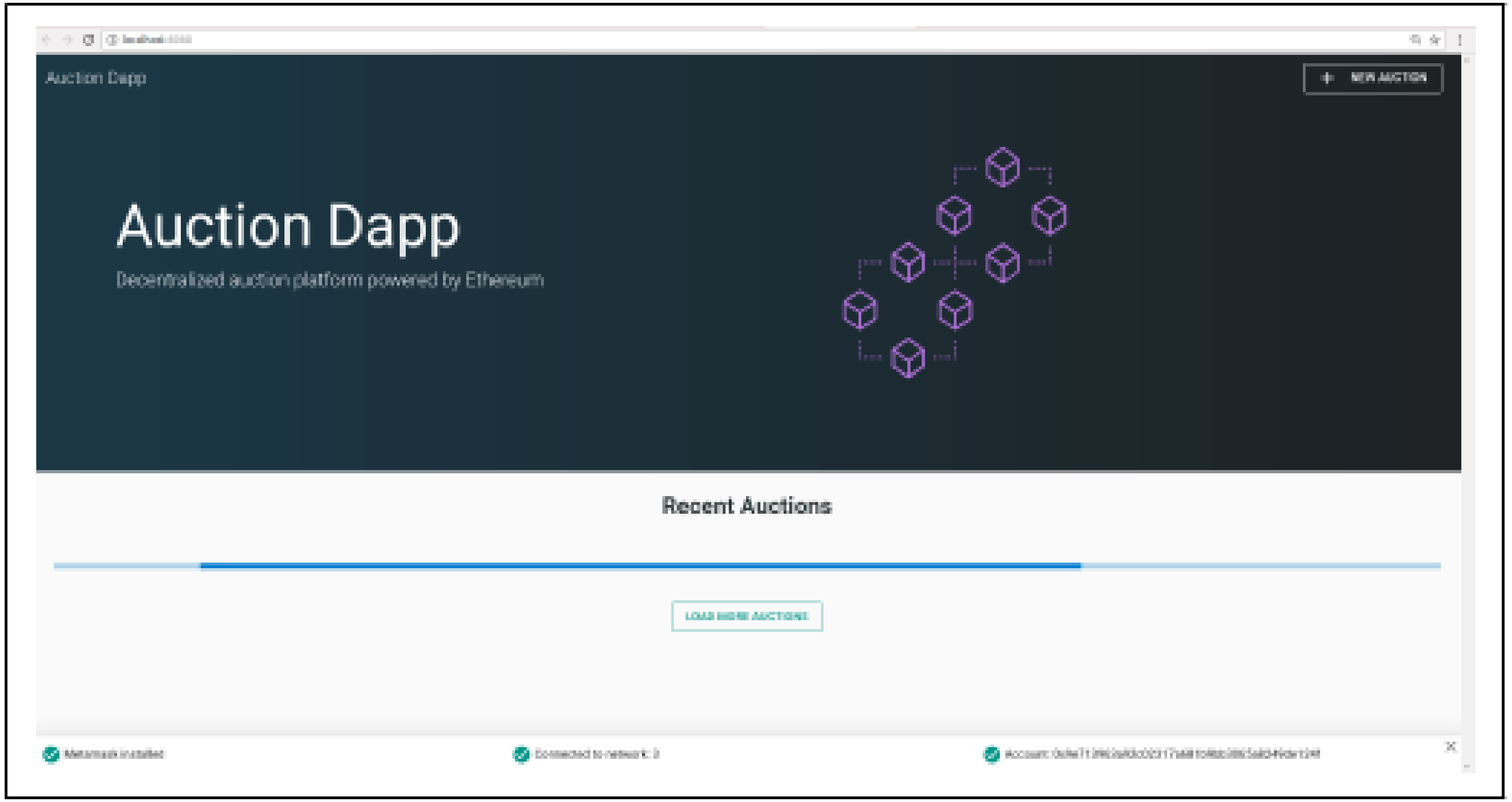
Topic 6: Example DApp using HTML, CSS, Solidity the EVM and the Ethereum Blockchain

Auction – The Functional Model Diagram


Auction House Smart Contract
2. Create Auction
Send Ethers
End
Transfer Asset to account
3. Bid (Send Ether to the contract)
Auction Creator
Buyer
Cancel
Transfer Asset back to the account
Refund Ether
1. Transfer ownership to auctionhouse contract
ERC721 Token

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction – The DApp User Interface – Part 1



Auction Dapp

Decentralized auction platform powered by Ethereum

Recent Auctions

✓ Metamask installed

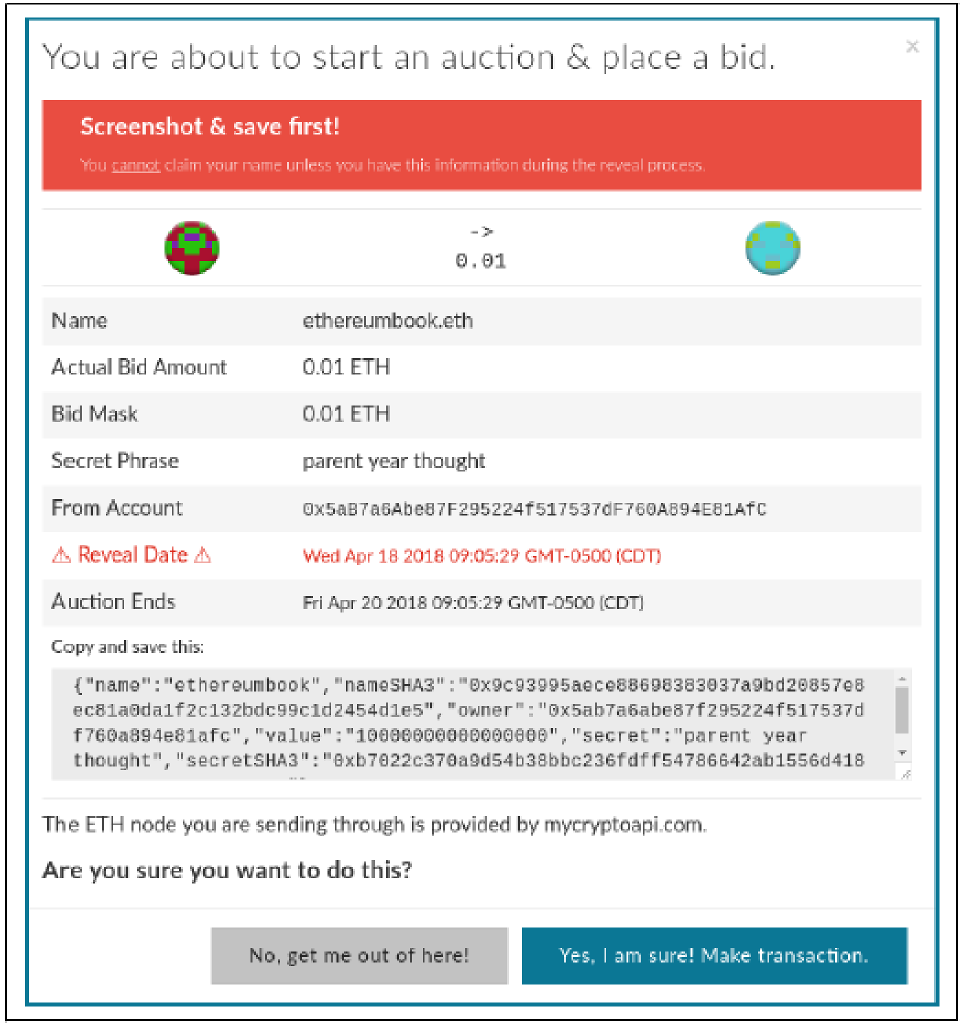
✓ Connected to network: 3

✓ Account: [illegible]

Figure 12-3. Auction DApp user interface

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction – The DApp User Interface – Part 2



You are about to start an auction & place a bid.

Screenshot & save first!

You cannot claim your name unless you have this information during the reveal process.

Name	ethereumbook.eth
Actual Bid Amount	0.01 ETH
Bid Mask	0.01 ETH
Secret Phrase	parent year thought
From Account	0x5aB7a6Abe87F295224f517537dF760A894E81AfC
⚠ Reveal Date ⚠	Wed Apr 18 2018 09:05:29 GMT-0500 (CDT)
Auction Ends	Fri Apr 20 2018 09:05:29 GMT-0500 (CDT)

Copy and save this:

```
{"name":"ethereumbook","nameSHA3":"0x9c93995aece88698383037a9bd20857e8
ec81a0da1f2c132bdc99c1d2454d1e5","owner":"0x5ab7a6abe87f295224f517537d
f760a894e81afc","value":"10000000000000000","secret":"parent year
thought","secretSHA3":"0xb7022c370a9d54b38bbc236fdff54786642ab1556d418
```

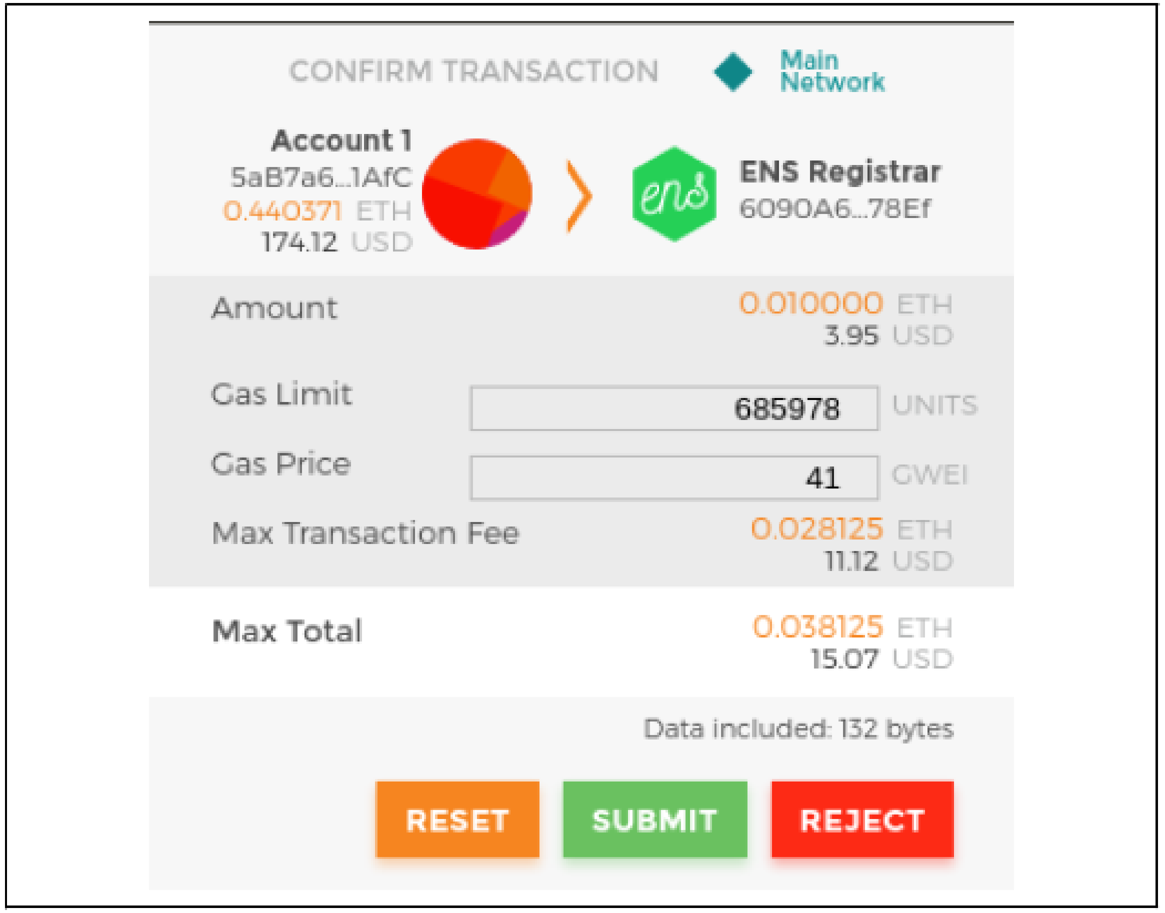
The ETH node you are sending through is provided by mycryptoapi.com.

Are you sure you want to do this?

Figure 12-8. Placing a bid for an ENS name

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction – The DApp User Interface – Part 3



CONFIRM TRANSACTION

Main Network

Account 1
5aB7a6...1AfC
0.440371 ETH
174.12 USD

ENS Registrar
6090A6...78Ef

Amount 0.010000 ETH
3.95 USD

Gas Limit 685978 UNITS

Gas Price 41 GWEI

Max Transaction Fee 0.028125 ETH
11.12 USD

Max Total 0.038125 ETH
15.07 USD

Data included: 132 bytes

Figure 12-9. MetaMask transaction containing your bid

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction – The DApp Architecture Diagram

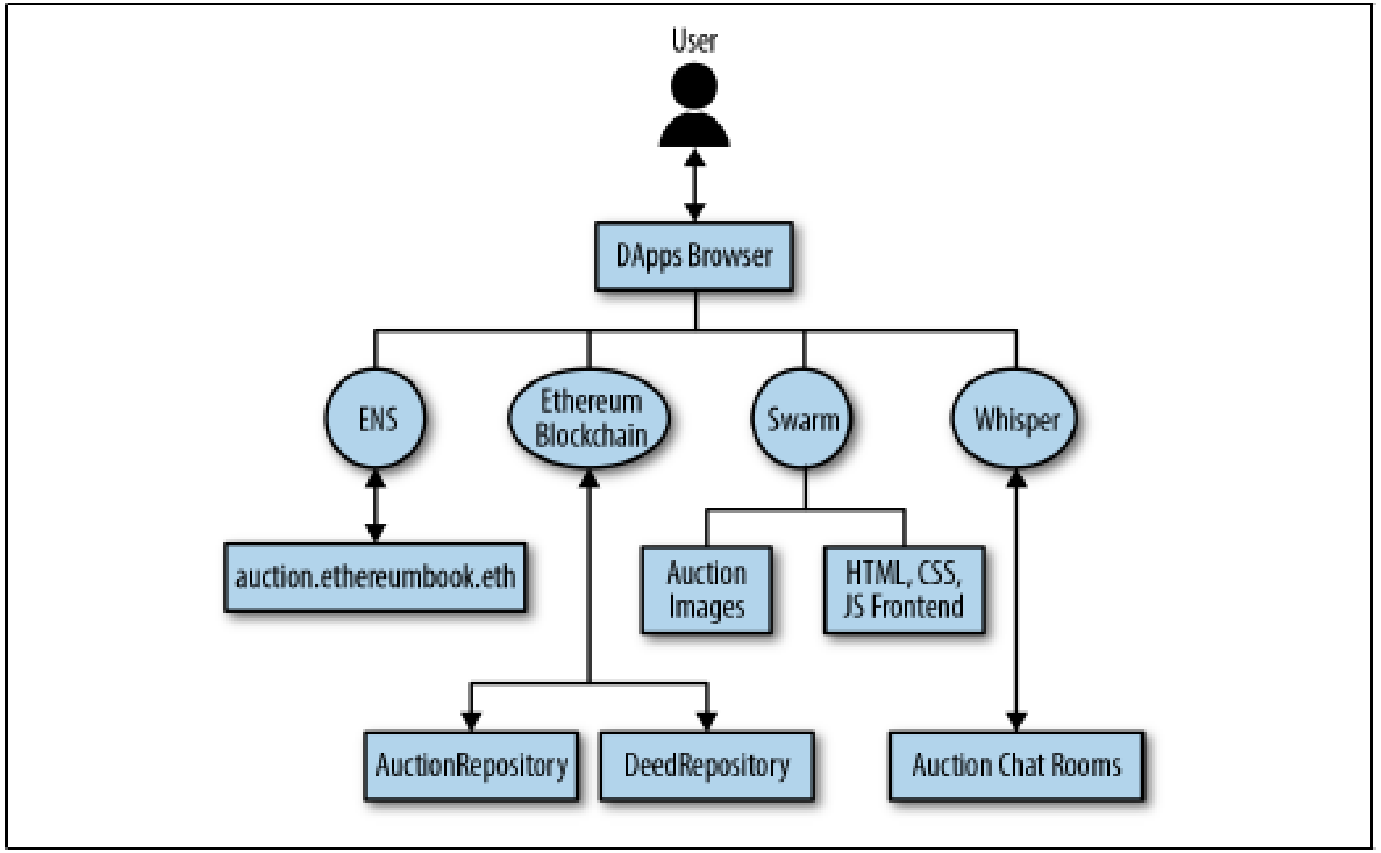
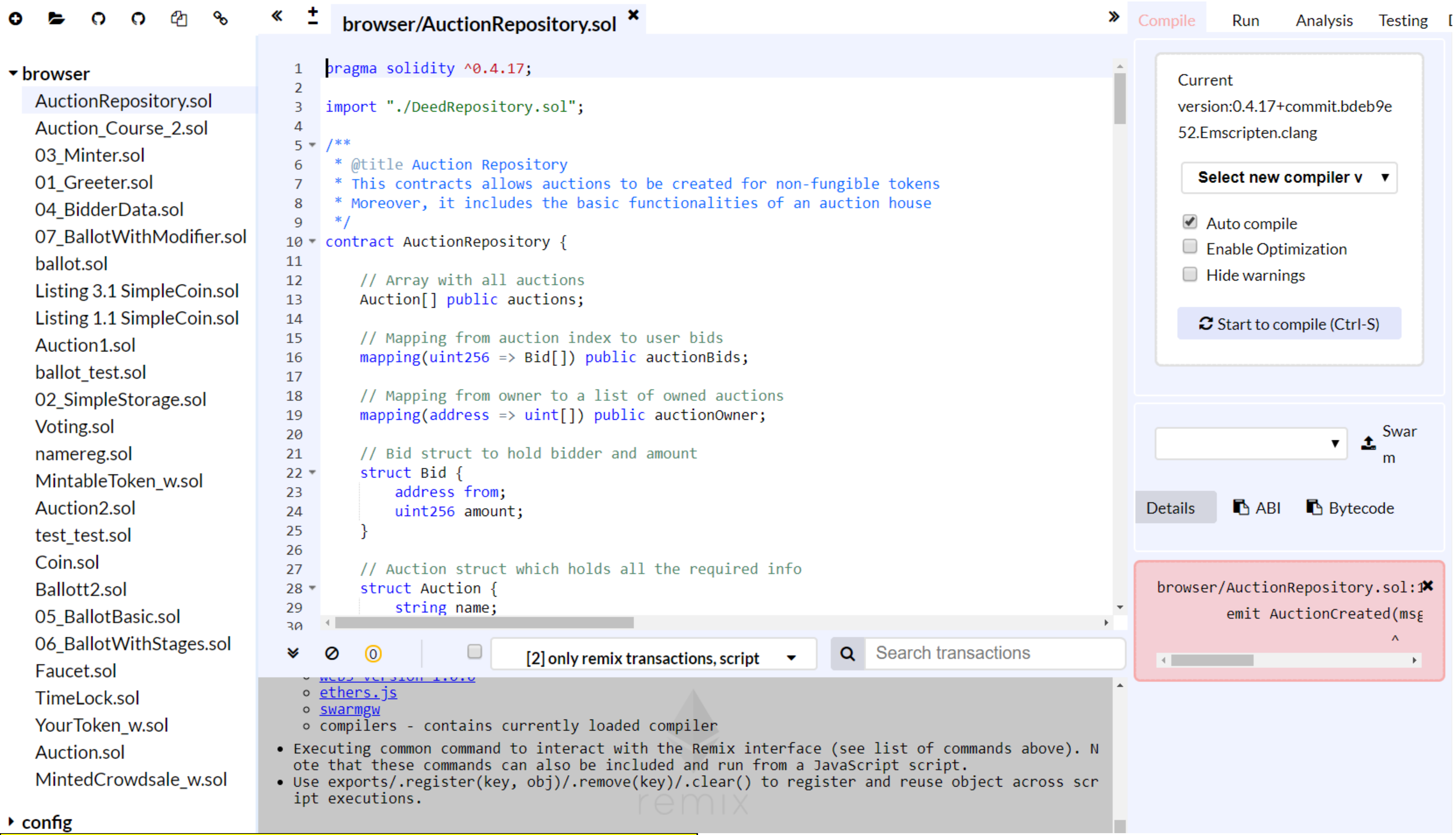


Figure 12-14. Auction DApp architecture

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction - the AuctionRepository Code



```
pragma solidity ^0.4.17;

import "./DeedRepository.sol";

/**
 * @title Auction Repository
 * This contracts allows auctions to be created for non-fungible tokens
 * Moreover, it includes the basic functionalities of an auction house
 */
contract AuctionRepository {

    // Array with all auctions
    Auction[] public auctions;

    // Mapping from auction index to user bids
    mapping(uint256 => Bid[]) public auctionBids;

    // Mapping from owner to a list of owned auctions
    mapping(address => uint[]) public auctionOwner;

    // Bid struct to hold bidder and amount
    struct Bid {
        address from;
        uint256 amount;
    }

    // Auction struct which holds all the required info
    struct Auction {
        string name;
```

Current version:0.4.17+commit.bdeb9e52.Emscripten.clang

☑ Auto compile

□ Enable Optimization

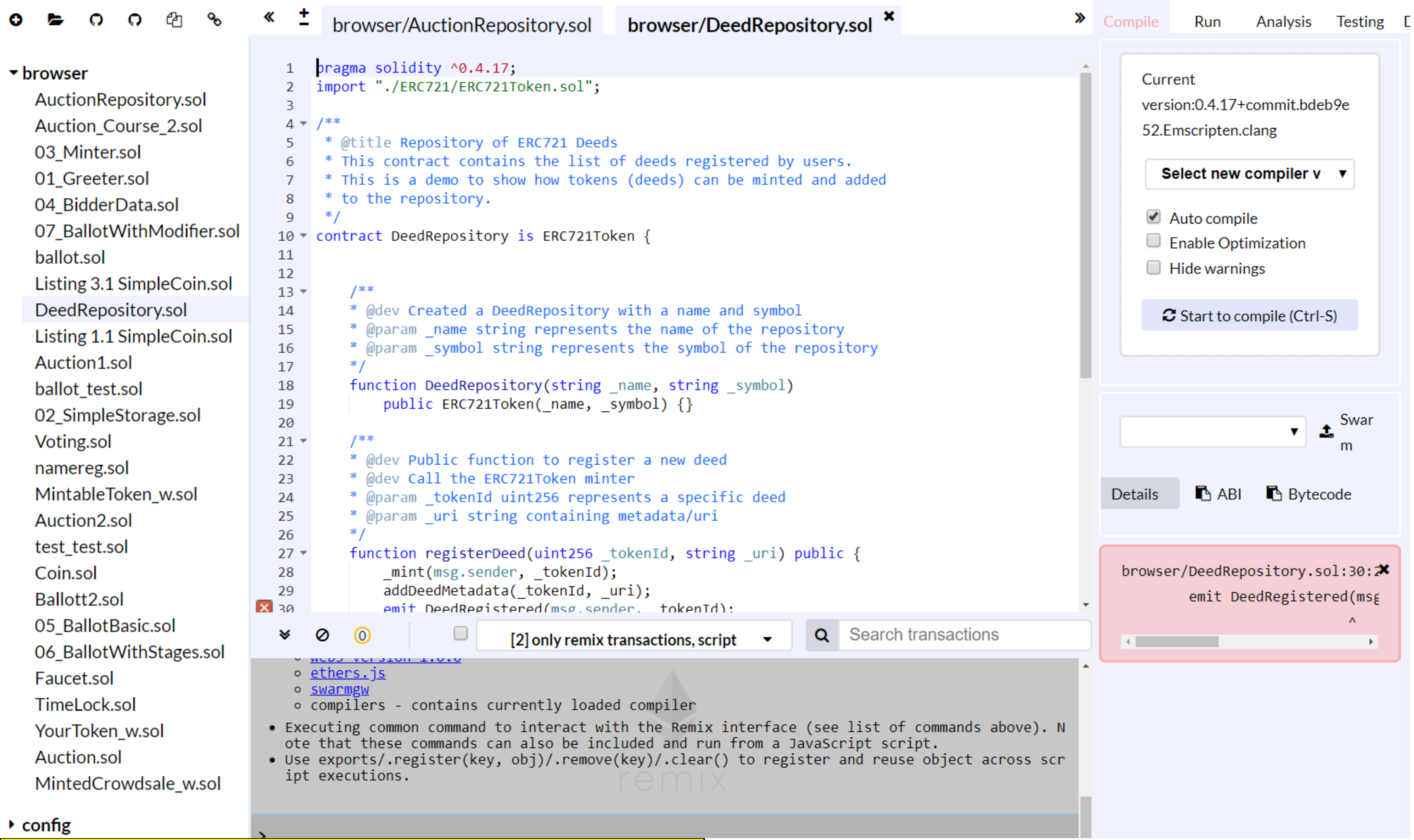
□ Hide warnings

```
browser/AuctionRepository.sol:1
    emit AuctionCreated(msg
                      ^
```

```
  o ethers.js
  o swarmgw
  o compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). Note that these commands can also be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across script executions.
```

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Auction - the DeedRepository Code



browser/AuctionRepository.sol
browser/DeedRepository.sol
Compile
Run
Analysis
Testing
Select new compiler v
Start to compile (Ctrl-S)
Swarm
Details
ABI
Bytecode
[2] only remix transactions, script
Search transactions
remix

- browser
  - AuctionRepository.sol
  - Auction_Course_2.sol
  - 03_Minter.sol
  - 01_Greeter.sol
  - 04_BidderData.sol
  - 07_BallotWithModifier.sol
  - ballot.sol
  - Listing 3.1 SimpleCoin.sol
  - DeedRepository.sol
  - Listing 1.1 SimpleCoin.sol
  - Auction1.sol
  - ballot_test.sol
  - 02_SimpleStorage.sol
  - Voting.sol
  - namereg.sol
  - MintableToken_w.sol
  - Auction2.sol
  - test_test.sol
  - Coin.sol
  - Ballott2.sol
  - 05_BallotBasic.sol
  - 06_BallotWithStages.sol
  - Faucet.sol
  - TimeLock.sol
  - YourToken_w.sol
  - Auction.sol
  - MintedCrowdsale_w.sol
- config

```
pragma solidity ^0.4.17;
import "./ERC721/ERC721Token.sol";

/**
 * @title Repository of ERC721 Deeds
 * This contract contains the list of deeds registered by users.
 * This is a demo to show how tokens (deeds) can be minted and added
 * to the repository.
 */
contract DeedRepository is ERC721Token {


    /**
    * @dev Created a DeedRepository with a name and symbol
    * @param _name string represents the name of the repository
    * @param _symbol string represents the symbol of the repository
    */
    function DeedRepository(string _name, string _symbol)
        public ERC721Token(_name, _symbol) {}

    /**
    * @dev Public function to register a new deed
    * @dev Call the ERC721Token minter
    * @param _tokenId uint256 represents a specific deed
    * @param _uri string containing metadata/uri
    */
    function registerDeed(uint256 _tokenId, string _uri) public {
        _mint(msg.sender, _tokenId);
        addDeedMetadata(_tokenId, _uri);
        emit DeedRegistered(msg.sender, _tokenId);
```

Current version:0.4.17+commit.bdeb9e52.Emscripten.clang

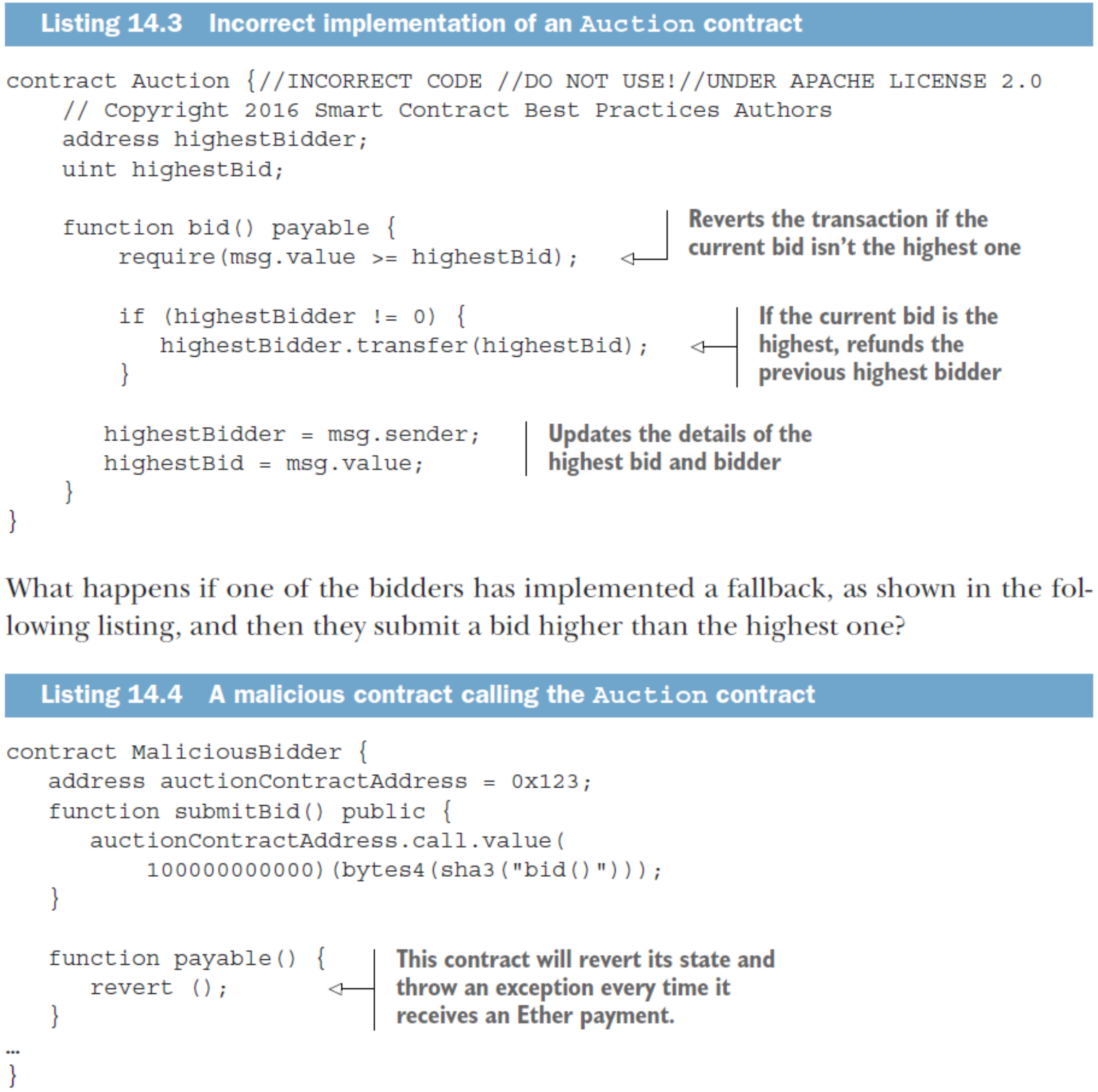
☑ Auto compile
□ Enable Optimization
□ Hide warnings

```
browser/DeedRepository.sol:30:2
    emit DeedRegistered(msg
                            ^
```

```
  o web3 version 1.0.0
  o ethers.js
  o swarmgw
  o compilers - contains currently loaded compiler
• Executing common command to interact with the Remix interface (see list of commands above). N
  ote that these commands can also be included and run from a JavaScript script.
• Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse object across scr
  ipt executions.
```

Source: Mastering Ethereum, by Andreas Antonopolous and Gavin Hill

Bad Auction



Listing 14.3 Incorrect implementation of an `Auction` contract

```
contract Auction {//INCORRECT CODE //DO NOT USE!//UNDER APACHE LICENSE 2.0
    // Copyright 2016 Smart Contract Best Practices Authors
    address highestBidder;
    uint highestBid;

    function bid() payable {
        require(msg.value >= highestBid);

        if (highestBidder != 0) {
           highestBidder.transfer(highestBid);
        }

       highestBidder = msg.sender;
       highestBid = msg.value;
    }
}
```

Reverts the transaction if the current bid isn't the highest one

If the current bid is the highest, refunds the previous highest bidder

Updates the details of the highest bid and bidder

What happens if one of the bidders has implemented a fallback, as shown in the following listing, and then they submit a bid higher than the highest one?

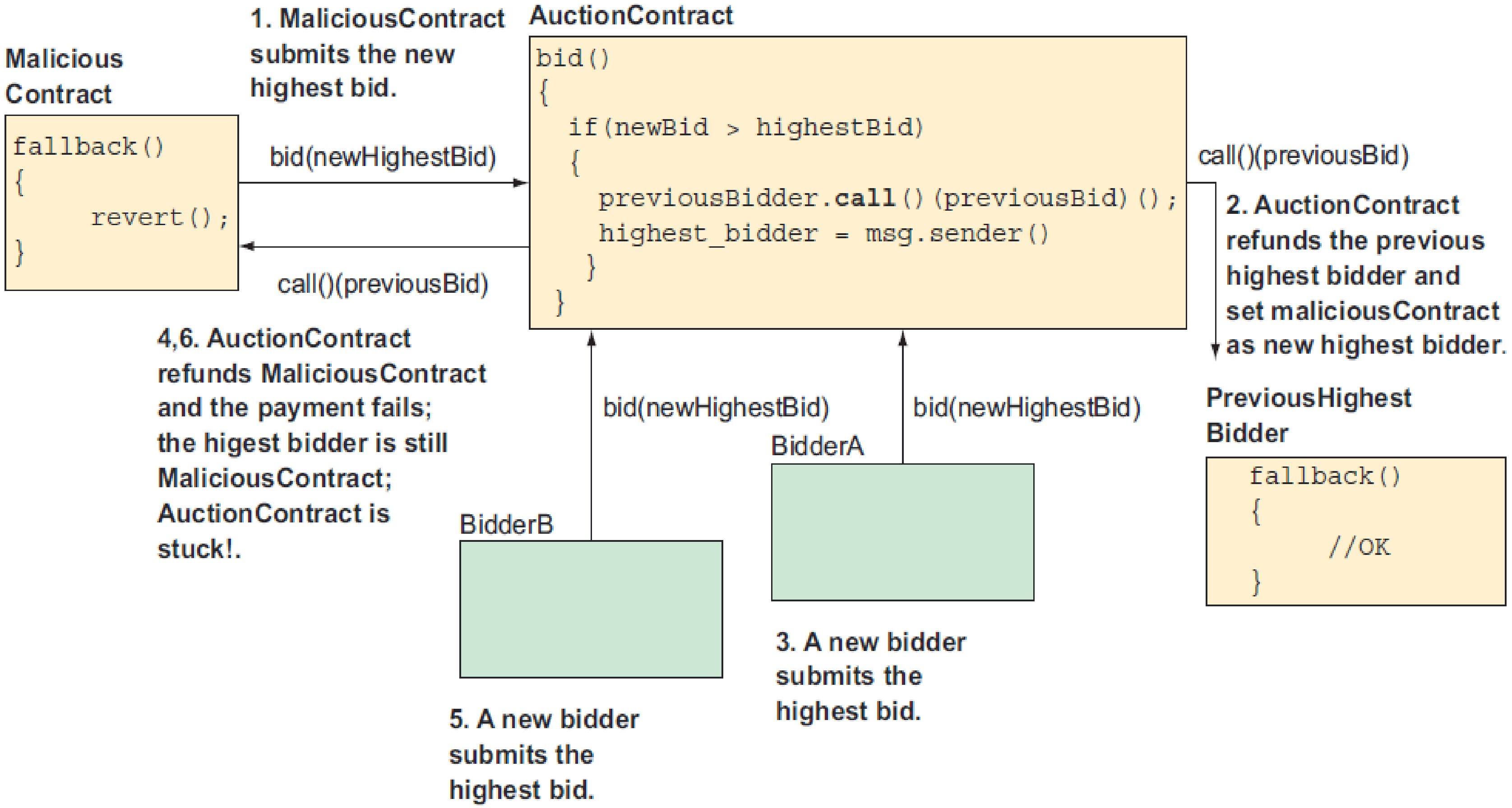
Listing 14.4 A malicious contract calling the `Auction` contract

```
contract MaliciousBidder {
   address auctionContractAddress = 0x123;
   function submitBid() public {
      auctionContractAddress.call.value(
          100000000000)(bytes4(sha3("bid()")));
   }

   function payable() {
      revert ();
   }
...
}
```

This contract will revert its state and throw an exception every time it receives an Ether payment.

Source: Roberto Infante, Building Ethereum DApps, 2019

Bad Auction



```
fallback()
{
      revert();
}
```

```
bid()
{
  if(newBid > highestBid)
  {
    previousBidder.call()(previousBid)();
    highest_bidder = msg.sender()
  }
}
```

```
fallback()
{
      //OK
}
```

1. MaliciousContract submits the new highest bid.

2. AuctionContract refunds the previous highest bidder and set maliciousContract as new highest bidder.

3. A new bidder submits the highest bid.

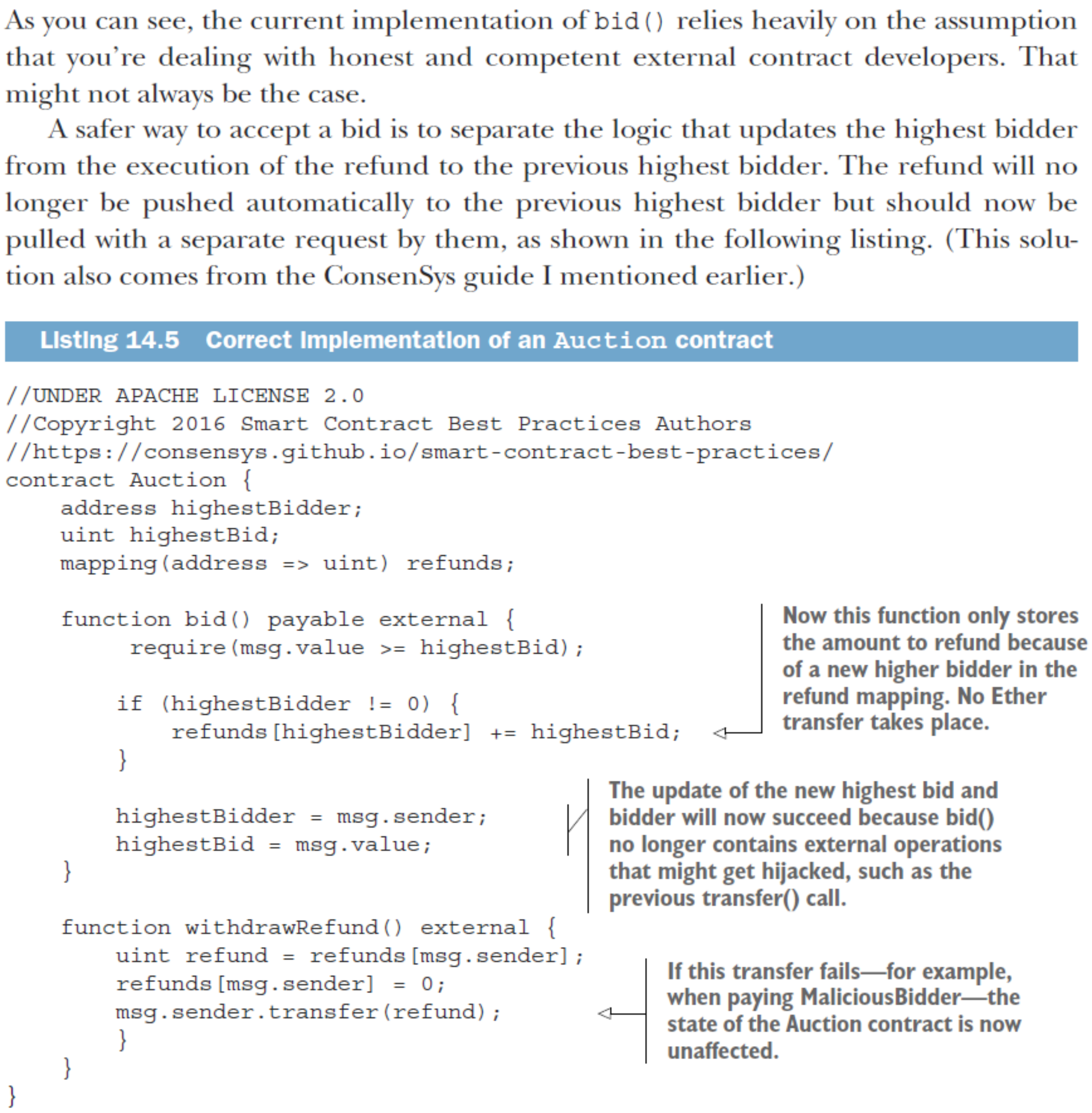
4,6. AuctionContract refunds MaliciousContract and the payment fails; the higest bidder is still MaliciousContract; AuctionContract is stuck!.

5. A new bidder submits the highest bid.

Figure 14.8 After the malicious contract has become the highest bidder, the `Auction` contract becomes unusable because it will unsuccessfully try to refund the malicious contract at every new higher bid and will never be able to set the new highest bidder.

Source: Roberto Infante, Building Ethereum DApps, 2019

Bad Auction



As you can see, the current implementation of `bid()` relies heavily on the assumption that you're dealing with honest and competent external contract developers. That might not always be the case.

A safer way to accept a bid is to separate the logic that updates the highest bidder from the execution of the refund to the previous highest bidder. The refund will no longer be pushed automatically to the previous highest bidder but should now be pulled with a separate request by them, as shown in the following listing. (This solution also comes from the ConsenSys guide I mentioned earlier.)

Listing 14.5 Correct implementation of an `Auction` contract

```
//UNDER APACHE LICENSE 2.0
//Copyright 2016 Smart Contract Best Practices Authors
//https://consensys.github.io/smart-contract-best-practices/
contract Auction {
    address highestBidder;
    uint highestBid;
    mapping(address => uint) refunds;

    function bid() payable external {
         require(msg.value >= highestBid);

        if (highestBidder != 0) {
            refunds[highestBidder] += highestBid;
        }

        highestBidder = msg.sender;
        highestBid = msg.value;
    }

    function withdrawRefund() external {
        uint refund = refunds[msg.sender];
        refunds[msg.sender] = 0;
        msg.sender.transfer(refund);
        }
    }
}
```

Now this function only stores the amount to refund because of a new higher bidder in the refund mapping. No Ether transfer takes place.

The update of the new highest bid and bidder will now succeed because bid() no longer contains external operations that might get hijacked, such as the previous transfer() call.

If this transfer fails—for example, when paying MaliciousBidder—the state of the Auction contract is now unaffected.

Source: Roberto Infante, Building Ethereum DApps, 2019

Topic 7: How to Secure Blockchain Infrastructure and Applications

How to Secure Blockchain Infrastructure and Applications

Build and lead Teams of experienced, dedicated workers

Design securely

Implement securely

Document ***everything***

Verify ***everything*** on the system throughput, from the keyboard to the blockchain transaction log.

Test security on everything

Routinely test vulnerabilities (at least quarterly)

- https://tinyurl.com/y292y3yf

Penetration test semi-annually

- https://tinyurl.com/yya4vtac

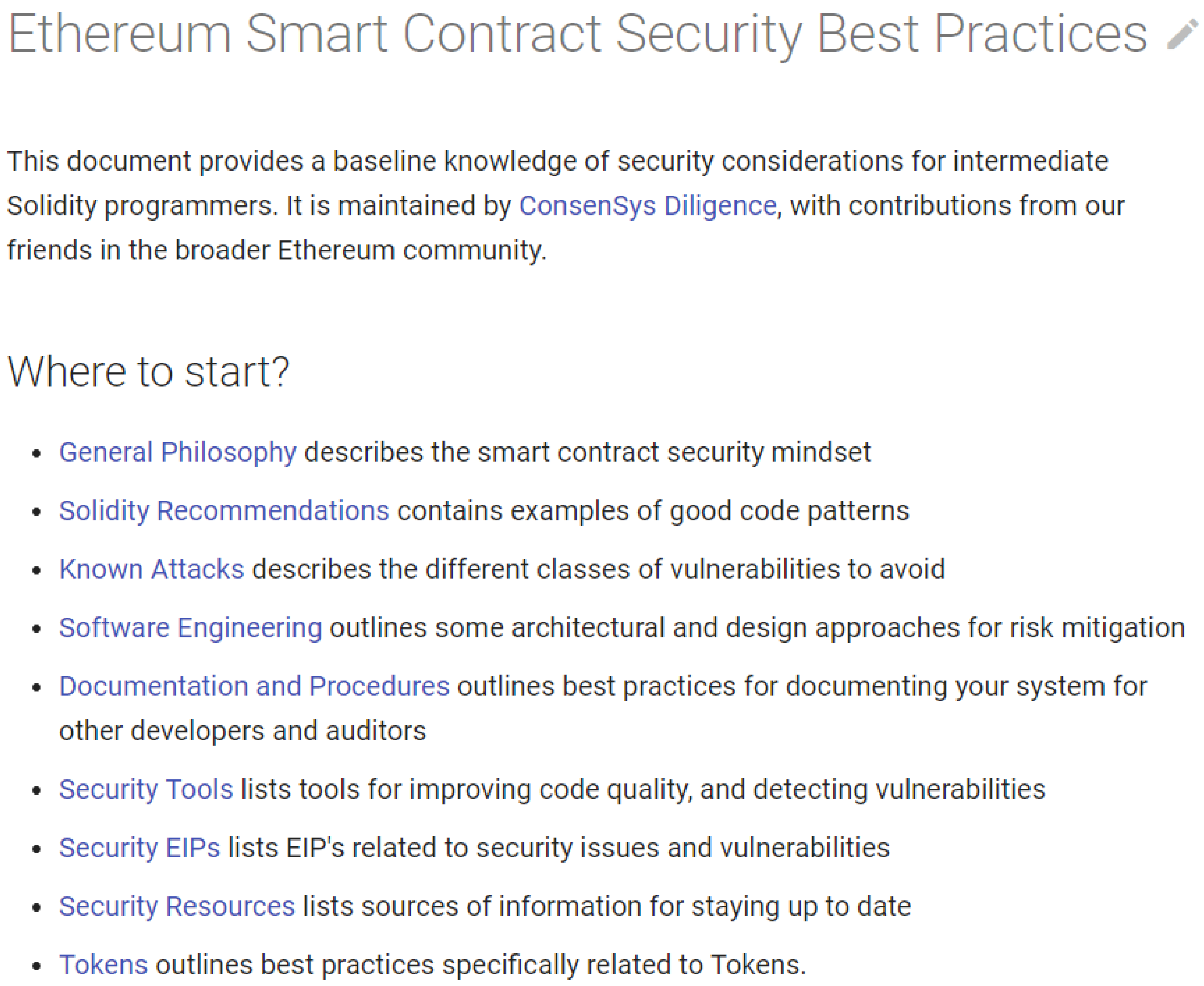
Test and document performance

- https://tinyurl.com/yxpwszj7

Do Threat Management

Continuously review for upgrading

Smart Contract Security Best Practices



Ethereum Smart Contract Security Best Practices

This document provides a baseline knowledge of security considerations for intermediate Solidity programmers. It is maintained by ConsenSys Diligence, with contributions from our friends in the broader Ethereum community.

Where to start?

- General Philosophy describes the smart contract security mindset
- Solidity Recommendations contains examples of good code patterns
- Known Attacks describes the different classes of vulnerabilities to avoid
- Software Engineering outlines some architectural and design approaches for risk mitigation
- Documentation and Procedures outlines best practices for documenting your system for other developers and auditors
- Security Tools lists tools for improving code quality, and detecting vulnerabilities
- Security EIPs lists EIP's related to security issues and vulnerabilities
- Security Resources lists sources of information for staying up to date
- Tokens outlines best practices specifically related to Tokens.

Source: Smart Contract Security: https://consensys.github.io/smart-contract-best-practices/

Topic 8: How to Perform Secure Software Development for Blockchain Applications by Design, Coding

How to Perform Secure Software Development for Blockchain Applications by Design, Coding practices, Testing and Verification

Design as a Decentralized App First

Define and develop tests

Code and test incrementally

Verify Infrastructure

Manage keys securely

Experienced DApp developers

Test-driven Development

Code reviews, by multiple experienced developers

Understand and remediate the weakest security points, especially protection of private keys and sensitive data.

Implement the tests on test net and understand exactly how the code will behave prior to moving to main net

Automate Smart Contract testing when possible

Using Best Practices in Coding Solidity to Achieve Secure Blockchain Applications that Perform Optimally

- Keep the smart contract code simple, coherent, and auditable
- Use the right sized variables. Ex. Don’t declare a 256-bit uint when a 16-bit uint will do.
- Pay attention to order of statements in functions
- Use Modifier declarations for implementing business rules and controlling the execution of programs
- Carefully differentiate between what should be stored “on-chain” and “off-chain”
- Use (reliably sourced) Oracles for external data sources when possible
- Only call external contracts from reliable sources and ensure they are tested first and have a good reputation for reliable results
- Maintain a standard order of funtions: The recommended order for functions within a smart contract are; constructor, fallback function, external, public, internal, private. Within a grouping, play the constant functions last.

Source: Coursera – Smart Contract Course

Using Best Practices in Coding Solidity to Achieve Secure Blockchain Applications that Perform Optimally

- Understand the public visibility modifier for data. All state variables are created as private. Any variable on the block chain is viewable to all, irrespective of the visibility modifier. You have to explicitly state that a variable is public. When a variable is declared public, the Solidity compiler automatically creates a *getter method* to view the value of the variable. Internal to the smart contract, the variable is accessed as *data*, externally it is accessed as a *function*. Be aware of this difference in accessing the public data and the getter method.
- Multiple modifiers that apply to a function, by specifying them in a white space separated list and are evaluated in the order presented. Hence, if the output of one modifier depends on the other, make sure you order them in the right sequence. For example, function buy, has three modifiers specified in the following order; payable, enoughMoney, item available. Use Solidity-defined payable modifier when sending value. Only through payable functions, can an account send value to another address. Payable is a reserved keyword, you may use payable as an addition to an existing function.

Source: Coursera – Smart Contract Course

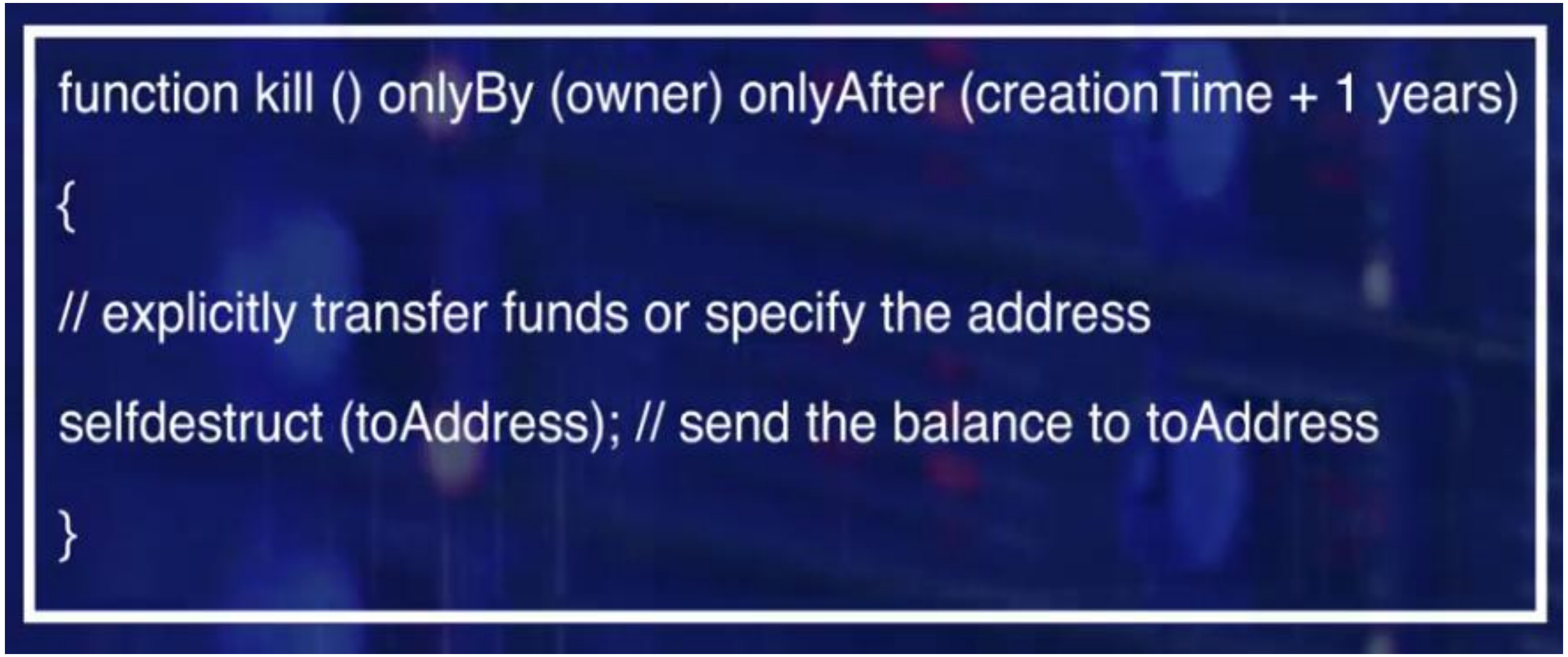
Using Best Practices in Coding Solidity to Achieve Secure Blockchain Applications that Perform Optimally

- Use Function Modifiers for
  - Implementing Rules, Policies, and/or Regulations
  - Implementing Common Rules for all who may access a Function
  - Declaratively validating application-specific conditions
  - Providing auditable elements to allow verification of the correctness of a smart contract
- Using Events for Notification
- Beware of the now() time variable
- Use secure hashing for protecting data

Source: Coursera – Smart Contract Course

Using Best Practices in Coding Solidity to Achieve Secure Blockchain Applications that Perform Optimally

Killing a Smart Contract



```
function kill () onlyBy (owner) onlyAfter (creationTime + 1 years)
{
// explicitly transfer funds or specify the address
selfdestruct (toAddress); // send the balance to toAddress
}
```

Note: This is irreversible.

Source: Coursera – Smart Contract Course

Topic 9: Blockchain and Auditing Practices, Testing and Verification

Blockchain and Auditing Practices, Testing and Verification

Blockchain Integrity and Security

Coding and assembly of Smart Contracts and DApps

Infrastructure

Physical Security

Key Management Practices

Concepts of Auditing Blockchain Applications

Data should be validated and verified prior to committing as a Blockchain transaction because once written to the Blockchain it is immutable.

Sample transactions should be verified from the DApp as successfully written to the Blockchain.

Use Solidity Events and Blockchain Logs

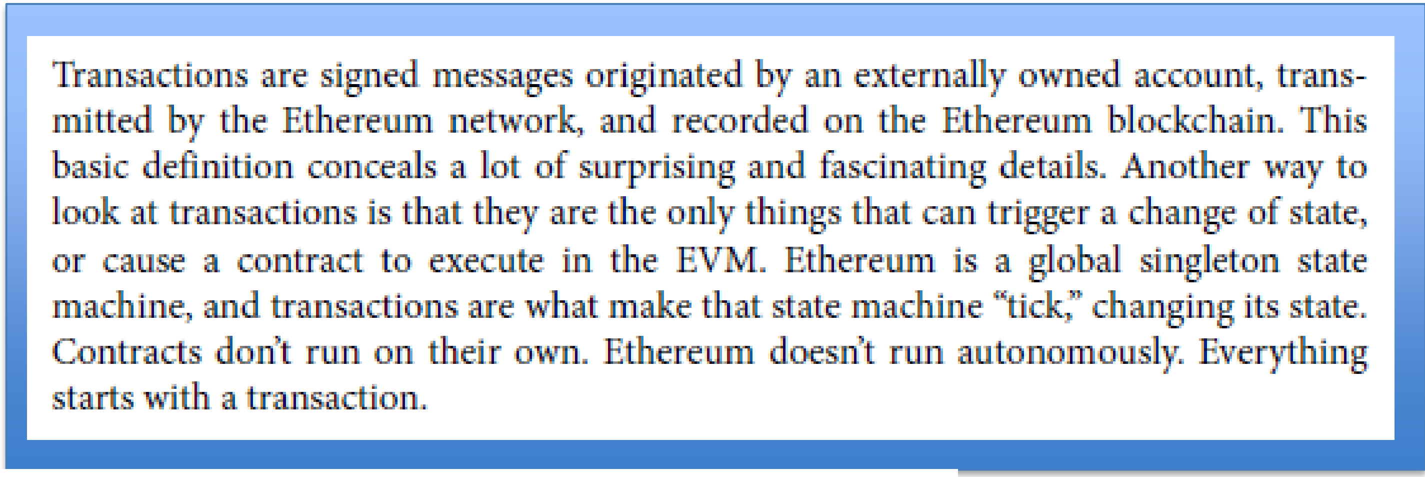
Transactions in Blockchain Structures

Blockchain Log Entries on geth

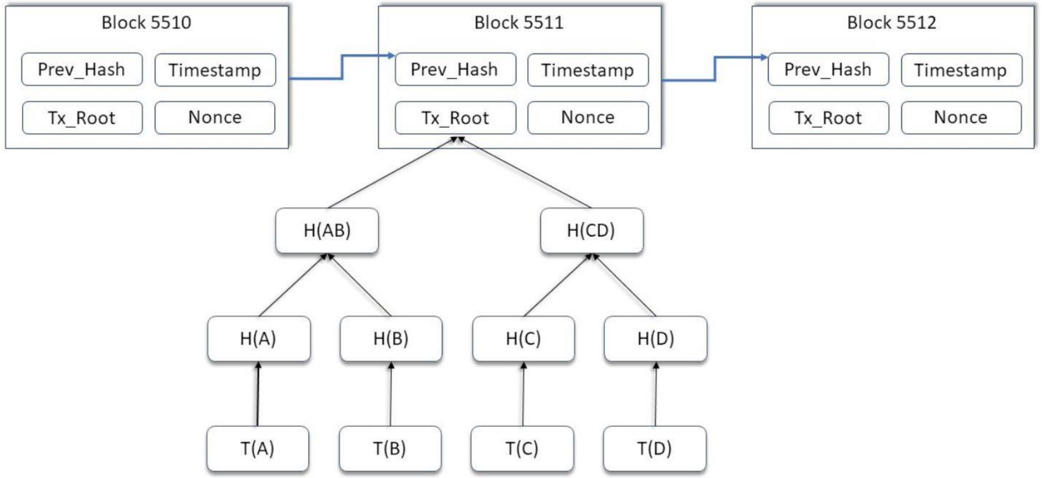
Examine using Javascript in geth console using *web3.eth.filter()*

Log object fields you can examine include

- **logIndex:** Log index position of the block.
- **transactionIndex:** Transaction index position the log was created from.
- **transactionHash:** Hash of the transaction this log was created from.
- **blockHash:** Hash of the block this log was in.
- **blockNumber:** Block number where this log was in.
- **address:** Address from which this log originated.
- **data:** Includes non-indexed arguments of the log.
- **topics:** Includes indexed log arguments.



Transactions are signed messages originated by an externally owned account, transmitted by the Ethereum network, and recorded on the Ethereum blockchain. This basic definition conceals a lot of surprising and fascinating details. Another way to look at transactions is that they are the only things that can trigger a change of state, or cause a contract to execute in the EVM. Ethereum is a global singleton state machine, and transactions are what make that state machine "tick," changing its state. Contracts don't run on their own. Ethereum doesn't run autonomously. Everything starts with a transaction.



blockchain data structure, by sombando, shared under a Creative Commons (BY-SA) license

Source: Mastering Ethereum by Andreas Antonopoulos & Gavin Wood

Topic 10: Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Data should be validated and verified prior to committing as a Blockchain transaction because once written to the Blockchain it is immutable.

Sample transactions should be verified from the DApp as successfully written to the Blockchain.

Use Blockchain Logs

Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Blockchain Log Entries on geth

Examine using Javascript in geth console using *web3.eth.filter()*

Options include:

fromBlock: Number of the earliest block for fetching the logs or use ‘latest’ or ‘pending’

toBlock: Number of the latest block for fetching the logs or use ‘latest’ or ‘pending’

address: An address or list of addresses to only get logs from particular accounts

topics: List of log topics

When *web3.eth.filter()* is set to ‘pending’ it returns a transaction hash of the most recent pending transaction.

Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Blockchain Log Entries on geth

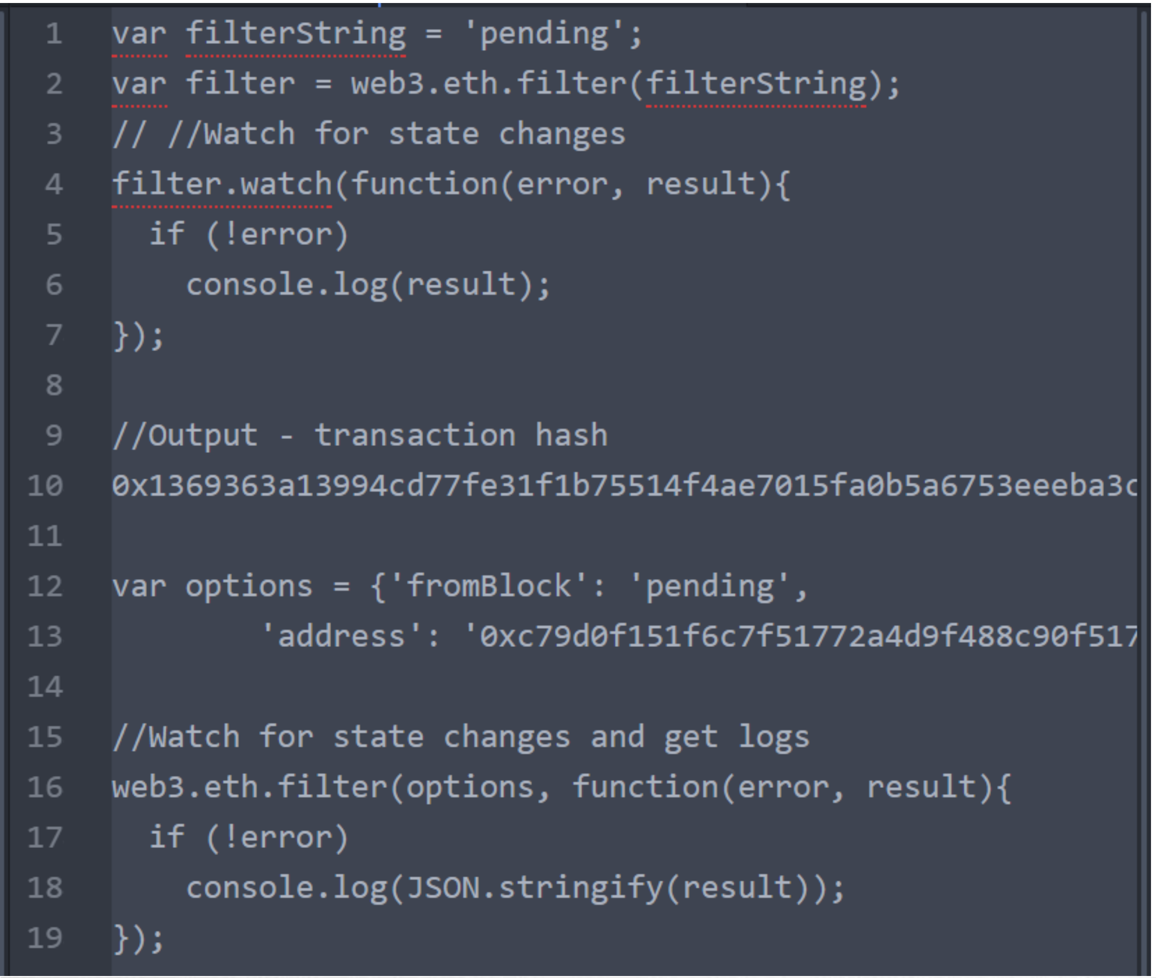
Examine using Javascript in geth console using *web3.eth.filter()*

Log object fields you can examine include

- **logIndex:** Log index position of the block.
- **transactionIndex:** Transaction index position the log was created from.
- **transactionHash:** Hash of the transaction this log was created from.
- **blockHash:** Hash of the block this log was in.
- **blockNumber:** Block number where this log was in.
- **address:** Address from which this log originated.
- **data:** Includes non-indexed arguments of the log.
- **topics:** Includes indexed log arguments.

Concepts of Auditing the Data and Transactions in Blockchain Data Structures

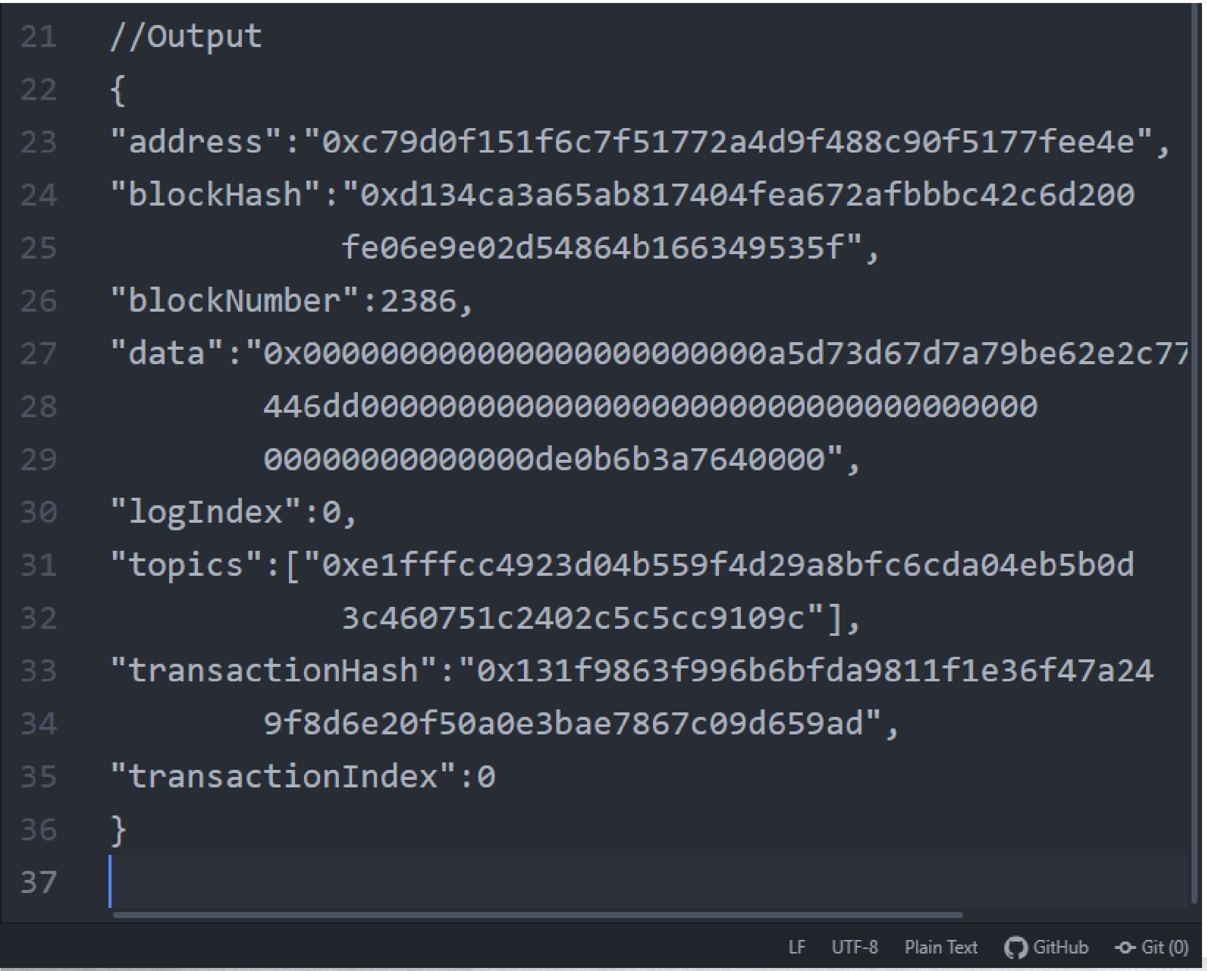
Example Log review code using Javascript



```
var filterString = 'pending';
var filter = web3.eth.filter(filterString);
// //Watch for state changes
filter.watch(function(error, result){
  if (!error)
    console.log(result);
});

//Output - transaction hash
0x1369363a13994cd77fe31f1b75514f4ae7015fa0b5a6753eeeba3c

var options = {'fromBlock': 'pending',
        'address': '0xc79d0f151f6c7f51772a4d9f488c90f517

//Watch for state changes and get logs
web3.eth.filter(options, function(error, result){
  if (!error)
    console.log(JSON.stringify(result));
});
```

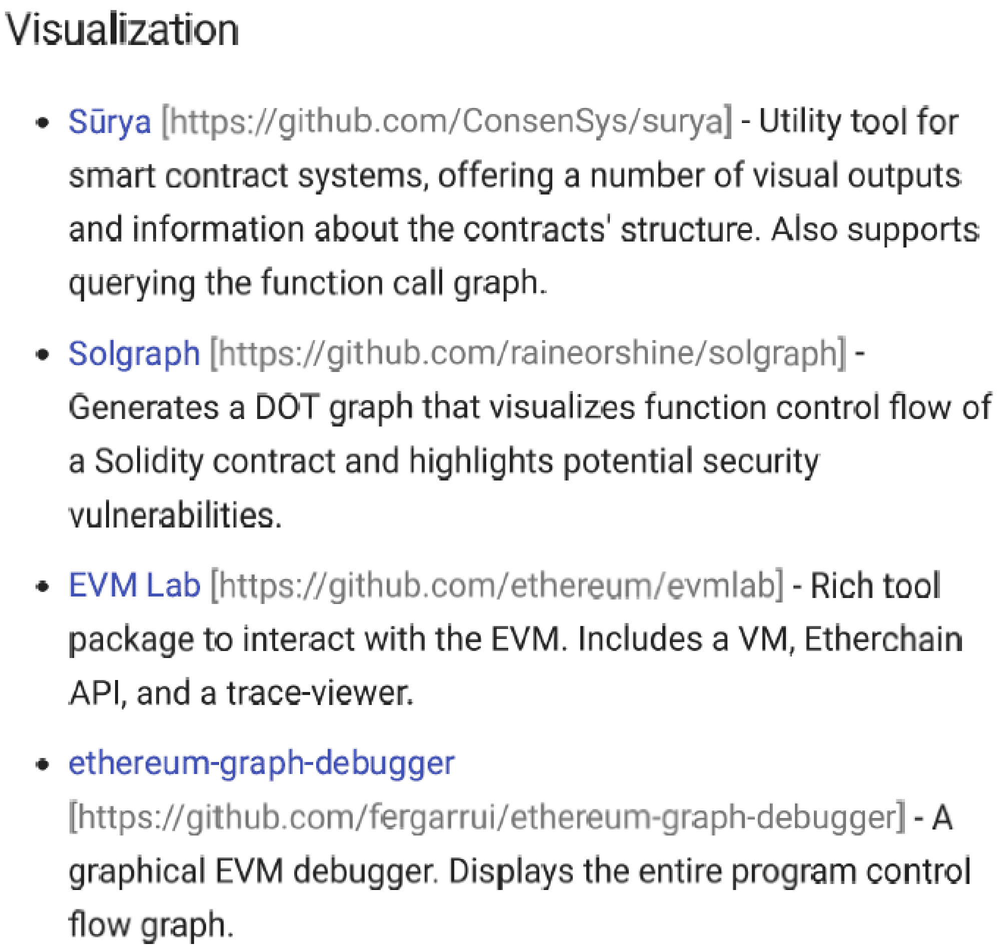
Concepts of Auditing the Data and Transactions in Blockchain Data Structures

Example Log review code using Javascript



```
//Output
{
"address":"0xc79d0f151f6c7f51772a4d9f488c90f5177fee4e",
"blockHash":"0xd134ca3a65ab817404fea672afbbbc42c6d200
            fe06e9e02d54864b166349535f",
"blockNumber":2386,
"data":"0x000000000000000000000000a5d73d67d7a79be62e2c77
        446dd00000000000000000000000000000000000
        00000000000000de0b6b3a7640000",
"logIndex":0,
"topics":["0xe1fffcc4923d04b559f4d29a8bfc6cda04eb5b0d
            3c460751c2402c5c5cc9109c"],
"transactionHash":"0x131f9863f996b6bfda9811f1e36f47a24
        9f8d6e20f50a0e3bae7867c09d659ad",
"transactionIndex":0
}
```

Security Tools for Auditing & Visualizing Transactions in Blockchain Data Structures



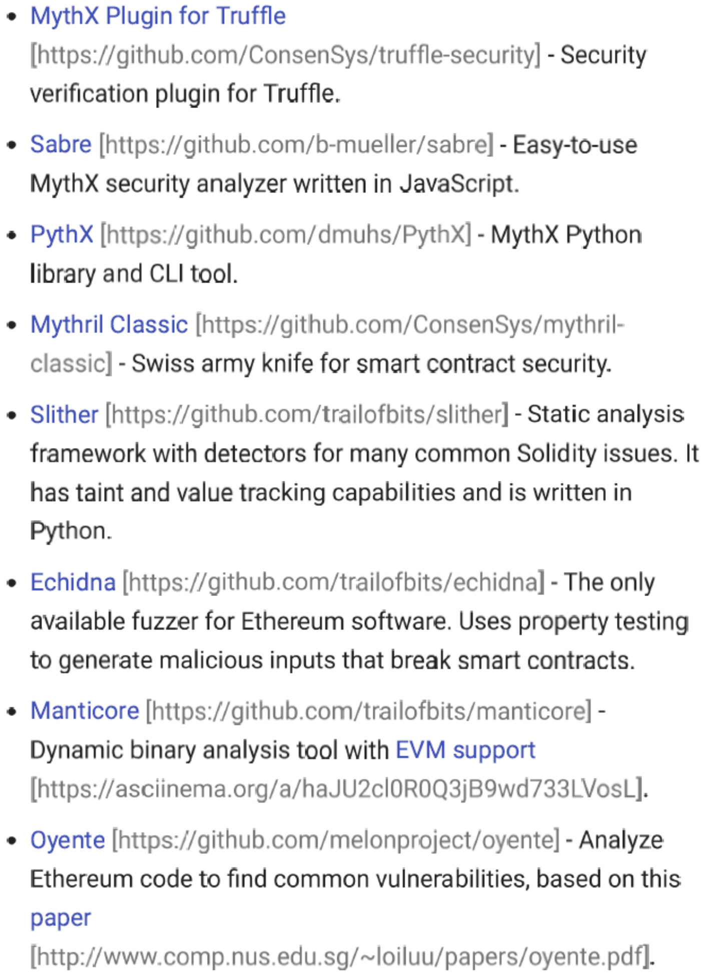
Visualization

- Sūrya [https://github.com/ConsenSys/surya] - Utility tool for smart contract systems, offering a number of visual outputs and information about the contracts' structure. Also supports querying the function call graph.
- Solgraph [https://github.com/raineorshine/solgraph] - Generates a DOT graph that visualizes function control flow of a Solidity contract and highlights potential security vulnerabilities.
- EVM Lab [https://github.com/ethereum/evmlab] - Rich tool package to interact with the EVM. Includes a VM, Etherchain API, and a trace-viewer.
- ethereum-graph-debugger [https://github.com/fergarrui/ethereum-graph-debugger] - A graphical EVM debugger. Displays the entire program control flow graph.

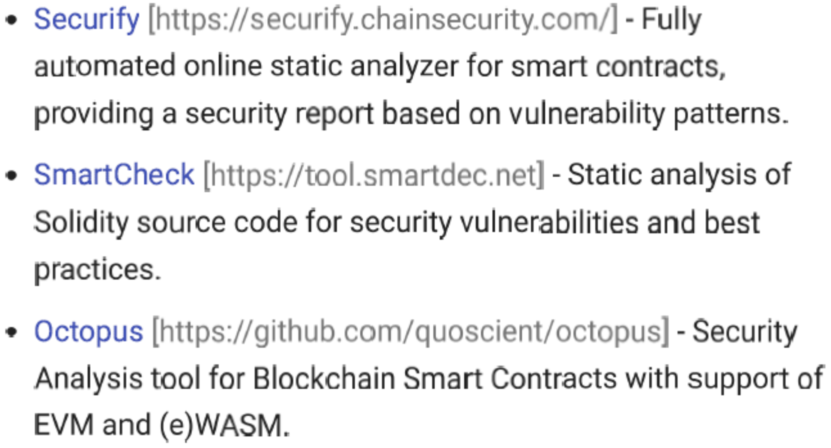
Source: Smart Contract Security: https://consensys.github.io/smart-contract-best-practices/

Security Tools for Auditing & Visualizing Transactions in Blockchain Data Structures

Static & Dynamic Analysis



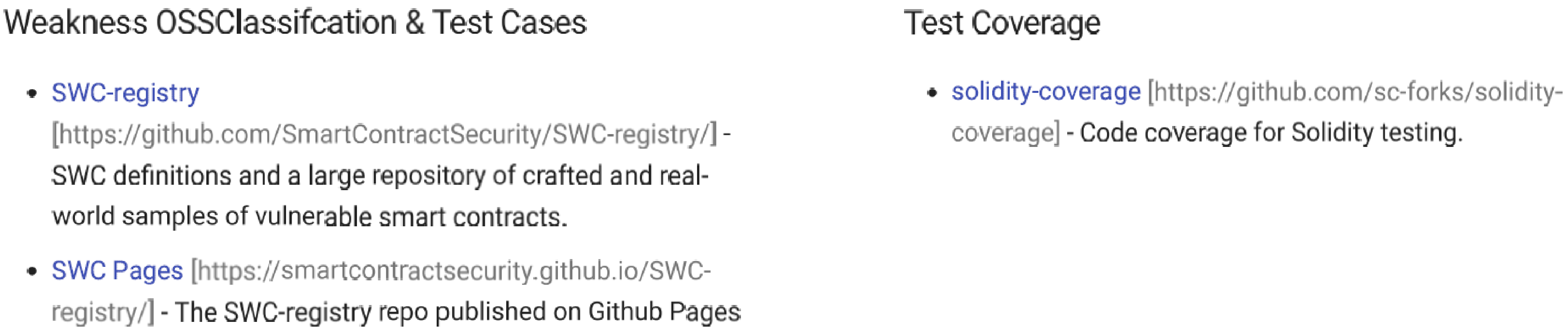
- MythX Plugin for Truffle [https://github.com/ConsenSys/truffle-security] - Security verification plugin for Truffle.
- Sabre [https://github.com/b-mueller/sabre] - Easy-to-use MythX security analyzer written in JavaScript.
- PythX [https://github.com/dmuhs/PythX] - MythX Python library and CLI tool.
- Mythril Classic [https://github.com/ConsenSys/mythril-classic] - Swiss army knife for smart contract security.
- Slither [https://github.com/trailofbits/slither] - Static analysis framework with detectors for many common Solidity issues. It has taint and value tracking capabilities and is written in Python.
- Echidna [https://github.com/trailofbits/echidna] - The only available fuzzer for Ethereum software. Uses property testing to generate malicious inputs that break smart contracts.
- Manticore [https://github.com/trailofbits/manticore] - Dynamic binary analysis tool with EVM support [https://asciinema.org/a/haJU2cl0R0Q3jB9wd733LVosL].
- Oyente [https://github.com/melonproject/oyente] - Analyze Ethereum code to find common vulnerabilities, based on this paper [http://www.comp.nus.edu.sg/~loiluu/papers/oyente.pdf].



- Securify [https://securify.chainsecurity.com/] - Fully automated online static analyzer for smart contracts, providing a security report based on vulnerability patterns.
- SmartCheck [https://tool.smartdec.net] - Static analysis of Solidity source code for security vulnerabilities and best practices.
- Octopus [https://github.com/quoscient/octopus] - Security Analysis tool for Blockchain Smart Contracts with support of EVM and (e)WASM.

Source: Smart Contract Security: https://consensys.github.io/smart-contract-best-practices/

Security Tools for Auditing & Visualizing Transactions in Blockchain Data Structures



Weakness OSSClassifcation & Test Cases

- SWC-registry [https://github.com/SmartContractSecurity/SWC-registry/] - SWC definitions and a large repository of crafted and real-world samples of vulnerable smart contracts.
- SWC Pages [https://smartcontractsecurity.github.io/SWC-registry/] - The SWC-registry repo published on Github Pages

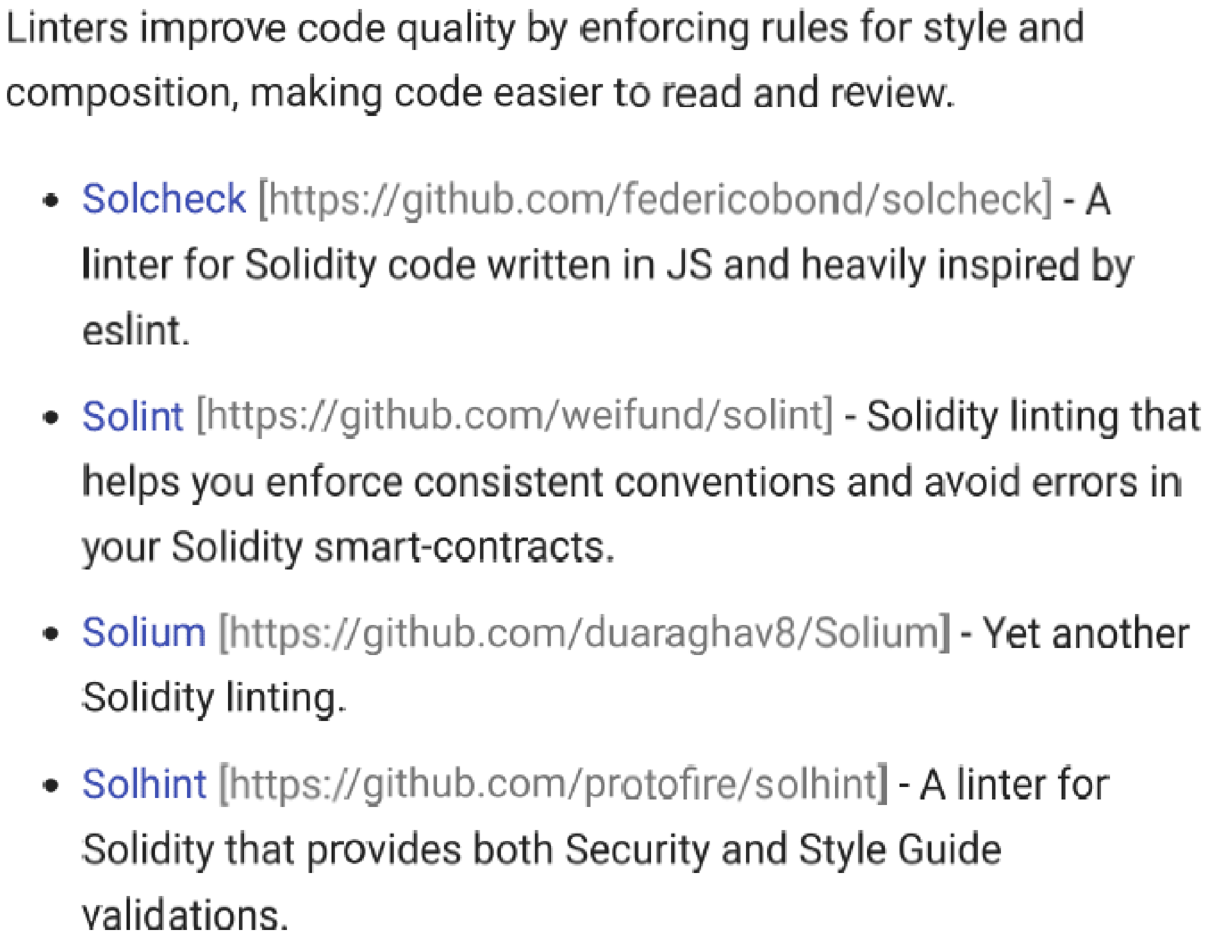
Test Coverage

- solidity-coverage [https://github.com/sc-forks/solidity-coverage] - Code coverage for Solidity testing.

Source: Smart Contract Security: https://consensys.github.io/smart-contract-best-practices/

Security Tools for Auditing & Visualizing Transactions in Blockchain Data Structures

Linters



Linters improve code quality by enforcing rules for style and composition, making code easier to read and review.

- Solcheck [https://github.com/federicobond/solcheck] - A linter for Solidity code written in JS and heavily inspired by eslint.
- Solint [https://github.com/weifund/solint] - Solidity linting that helps you enforce consistent conventions and avoid errors in your Solidity smart-contracts.
- Solium [https://github.com/duaraghav8/Solium] - Yet another Solidity linting.
- Solhint [https://github.com/protofire/solhint] - A linter for Solidity that provides both Security and Style Guide validations.

Source: Smart Contract Security: https://consensys.github.io/smart-contract-best-practices/

Topic 11: Automating the Auditing of Blockchains and Blockchain Applications

Automating the Auditing of Blockchains and Blockchain Applications

In February 2018, ***Maian***, an open source tool to monitor Smart Contracts for being Greedy, Prodigal, or Suicidal was announced.

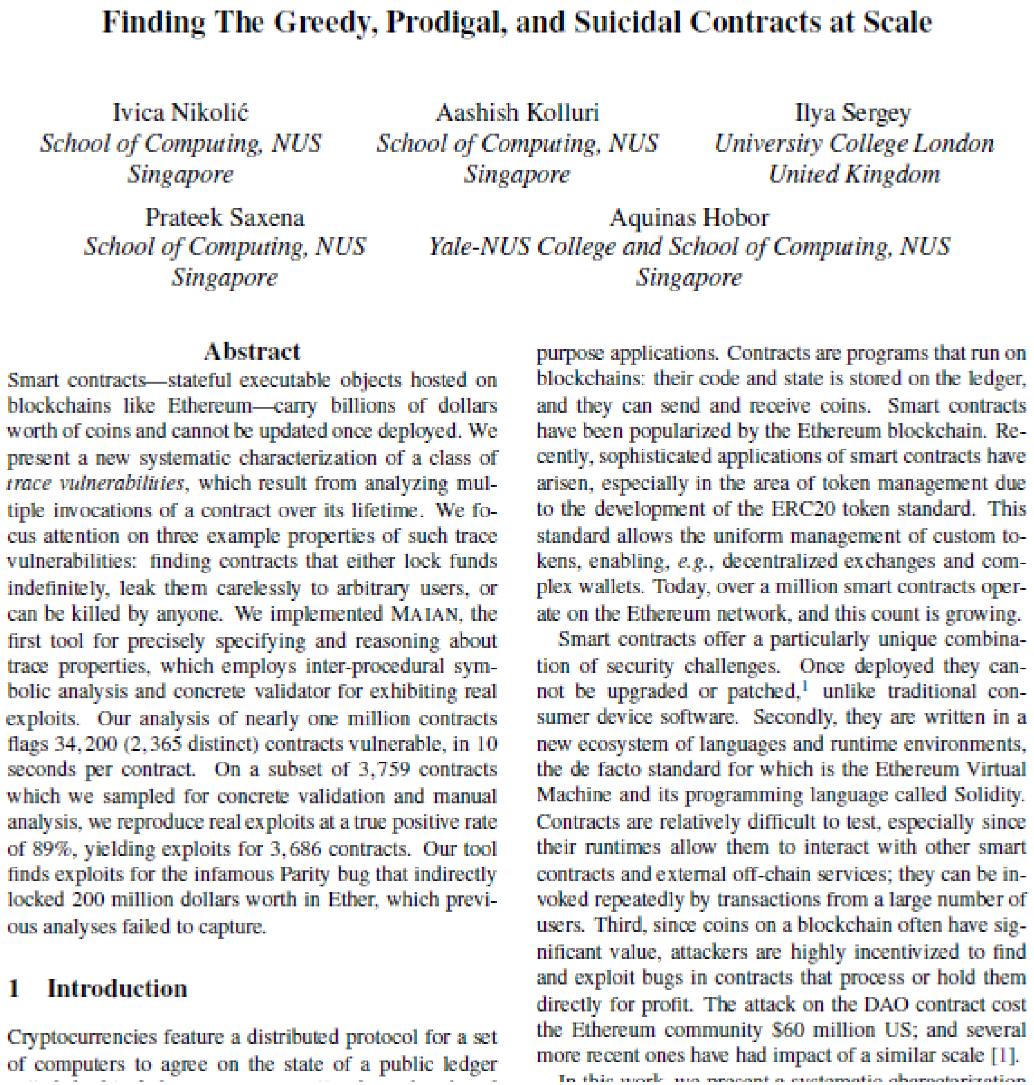
As of April 2018, EY has Blockchain Auditing tools and technology.

https://www.ey.com/en_gl/news/2018/04/ey-announces-blockchain-audit-technology

As of October 2018, How Big Four Auditors Delve Into Blockchain: PwC, Deloitte, EY and KPMG Approaches Compared

https://cointelegraph.com/news/how-big-four-auditors-delve-into-blockchain-pwc-deloitte-ey-and-kpmg-approaches-compared

Auditing Smart Contracts at Scale



Finding The Greedy, Prodigal, and Suicidal Contracts at Scale

Ivica Nikolić
School of Computing, NUS
Singapore

Aashish Kolluri
School of Computing, NUS
Singapore

Ilya Sergey
University College London
United Kingdom

Prateek Saxena
School of Computing, NUS
Singapore

Aquinas Hobor
Yale-NUS College and School of Computing, NUS
Singapore

Abstract

Smart contracts—stateful executable objects hosted on blockchains like Ethereum—carry billions of dollars worth of coins and cannot be updated once deployed. We present a new systematic characterization of a class of *trace vulnerabilities*, which result from analyzing multiple invocations of a contract over its lifetime. We focus attention on three example properties of such trace vulnerabilities: finding contracts that either lock funds indefinitely, leak them carelessly to arbitrary users, or can be killed by anyone. We implemented MAIAN, the first tool for precisely specifying and reasoning about trace properties, which employs inter-procedural symbolic analysis and concrete validator for exhibiting real exploits. Our analysis of nearly one million contracts flags 34,200 (2,365 distinct) contracts vulnerable, in 10 seconds per contract. On a subset of 3,759 contracts which we sampled for concrete validation and manual analysis, we reproduce real exploits at a true positive rate of 89%, yielding exploits for 3,686 contracts. Our tool finds exploits for the infamous Parity bug that indirectly locked 200 million dollars worth in Ether, which previous analyses failed to capture.

1 Introduction

Cryptocurrencies feature a distributed protocol for a set of computers to agree on the state of a public ledger

purpose applications. Contracts are programs that run on blockchains: their code and state is stored on the ledger, and they can send and receive coins. Smart contracts have been popularized by the Ethereum blockchain. Recently, sophisticated applications of smart contracts have arisen, especially in the area of token management due to the development of the ERC20 token standard. This standard allows the uniform management of custom tokens, enabling, *e.g.*, decentralized exchanges and complex wallets. Today, over a million smart contracts operate on the Ethereum network, and this count is growing.

Smart contracts offer a particularly unique combination of security challenges. Once deployed they cannot be upgraded or patched,[1] unlike traditional consumer device software. Secondly, they are written in a new ecosystem of languages and runtime environments, the de facto standard for which is the Ethereum Virtual Machine and its programming language called Solidity. Contracts are relatively difficult to test, especially since their runtimes allow them to interact with other smart contracts and external off-chain services; they can be invoked repeatedly by transactions from a large number of users. Third, since coins on a blockchain often have significant value, attackers are highly incentivized to find and exploit bugs in contracts that process or hold them directly for profit. The attack on the DAO contract cost the Ethereum community $60 million US; and several more recent ones have had impact of a similar scale [1].

In this work, we present a systematic characterization

February 2018 Technical paper about flaws in How Ethereum and EVM handle Smart Contracts. Worth your time

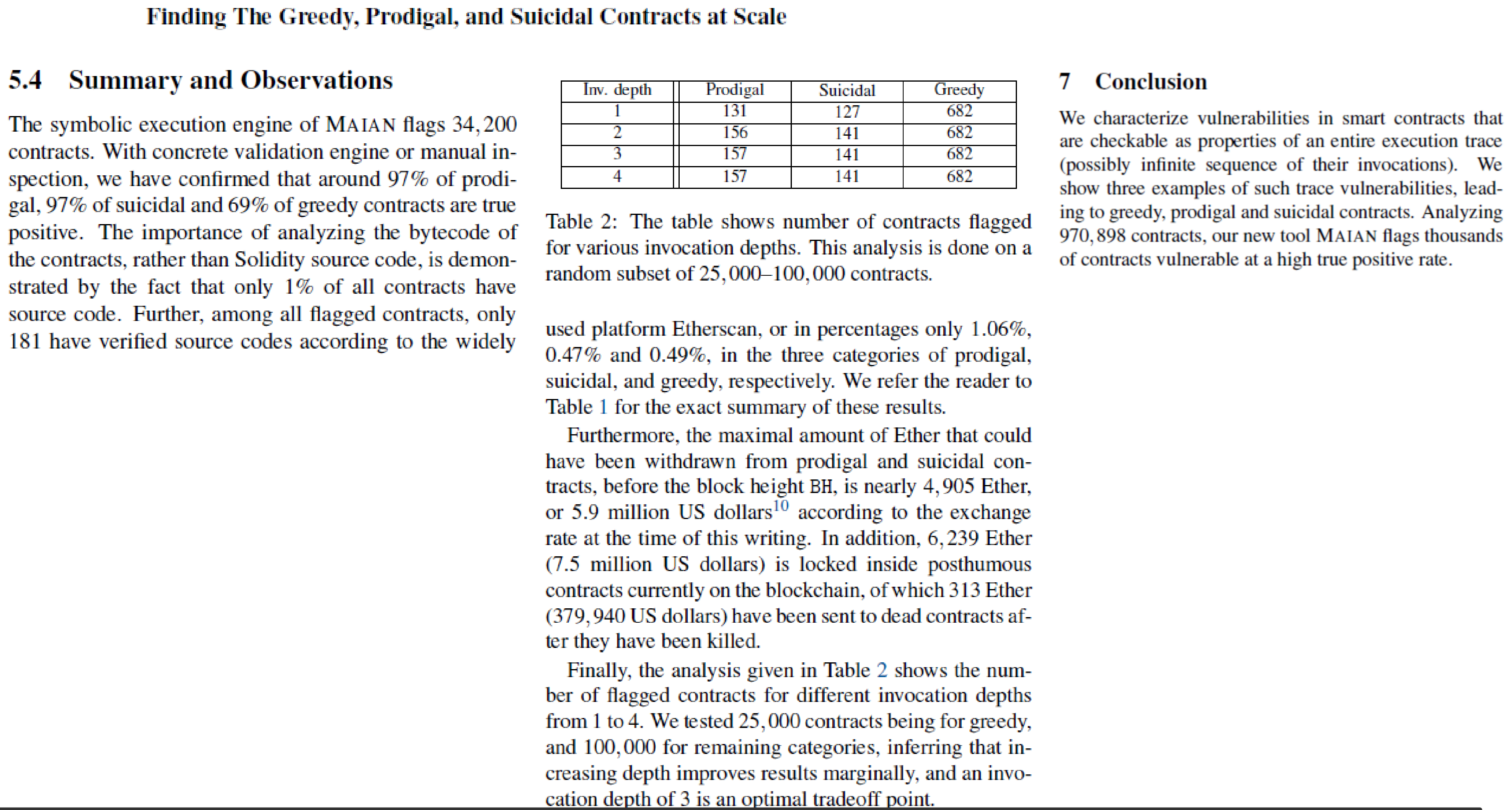
Prodigal - Leak them carelessly to arbitrary users

Suicidal - Can be killed by anyone

Greedy - Lock funds Indefinitely

Source: https://www.reddit.com/r/Bitcoin/comments/7ys5nq/pdf_finding_the_greedy_prodigal_and_suicidal/

Auditing Smart Contracts at Scale



Finding The Greedy, Prodigal, and Suicidal Contracts at Scale

5.4 Summary and Observations

The symbolic execution engine of MAIAN flags 34,200 contracts. With concrete validation engine or manual inspection, we have confirmed that around 97% of prodigal, 97% of suicidal and 69% of greedy contracts are true positive. The importance of analyzing the bytecode of the contracts, rather than Solidity source code, is demonstrated by the fact that only 1% of all contracts have source code. Further, among all flagged contracts, only 181 have verified source codes according to the widely used platform Etherscan, or in percentages only 1.06%, 0.47% and 0.49%, in the three categories of prodigal, suicidal, and greedy, respectively. We refer the reader to Table 1 for the exact summary of these results.

Inv. depth	Prodigal	Suicidal	Greedy
1	131	127	682
2	156	141	682
3	157	141	682
4	157	141	682

Table 2: The table shows number of contracts flagged for various invocation depths. This analysis is done on a random subset of 25,000–100,000 contracts.

Furthermore, the maximal amount of Ether that could have been withdrawn from prodigal and suicidal contracts, before the block height BH, is nearly 4,905 Ether, or 5.9 million US dollars[10] according to the exchange rate at the time of this writing. In addition, 6,239 Ether (7.5 million US dollars) is locked inside posthumous contracts currently on the blockchain, of which 313 Ether (379,940 US dollars) have been sent to dead contracts after they have been killed.

Finally, the analysis given in Table 2 shows the number of flagged contracts for different invocation depths from 1 to 4. We tested 25,000 contracts being for greedy, and 100,000 for remaining categories, inferring that increasing depth improves results marginally, and an invocation depth of 3 is an optimal tradeoff point.

7 Conclusion

We characterize vulnerabilities in smart contracts that are checkable as properties of an entire execution trace (possibly infinite sequence of their invocations). We show three examples of such trace vulnerabilities, leading to greedy, prodigal and suicidal contracts. Analyzing 970,898 contracts, our new tool MAIAN flags thousands of contracts vulnerable at a high true positive rate.

Prodigal - Leak them carelessly to arbitrary users

Suicidal - Can be killed by anyone

Greedy - Lock funds Indefinitely

Bottom Line: three to four percent of the smart contracts on Ethereum's blockchain still contain trace vulnerabilities, according to the researchers' new analysis methodology.

Sources: https://www.reddit.com/r/Bitcoin/comments/7ys5nq/pdf_finding_the_greedy_prodigal_and_suicidal/ and https://bitsonline.com/singapore-research-ethereum/

Auditing Smart Contracts at Scale

Opacity Is Hampering Ethereum Security

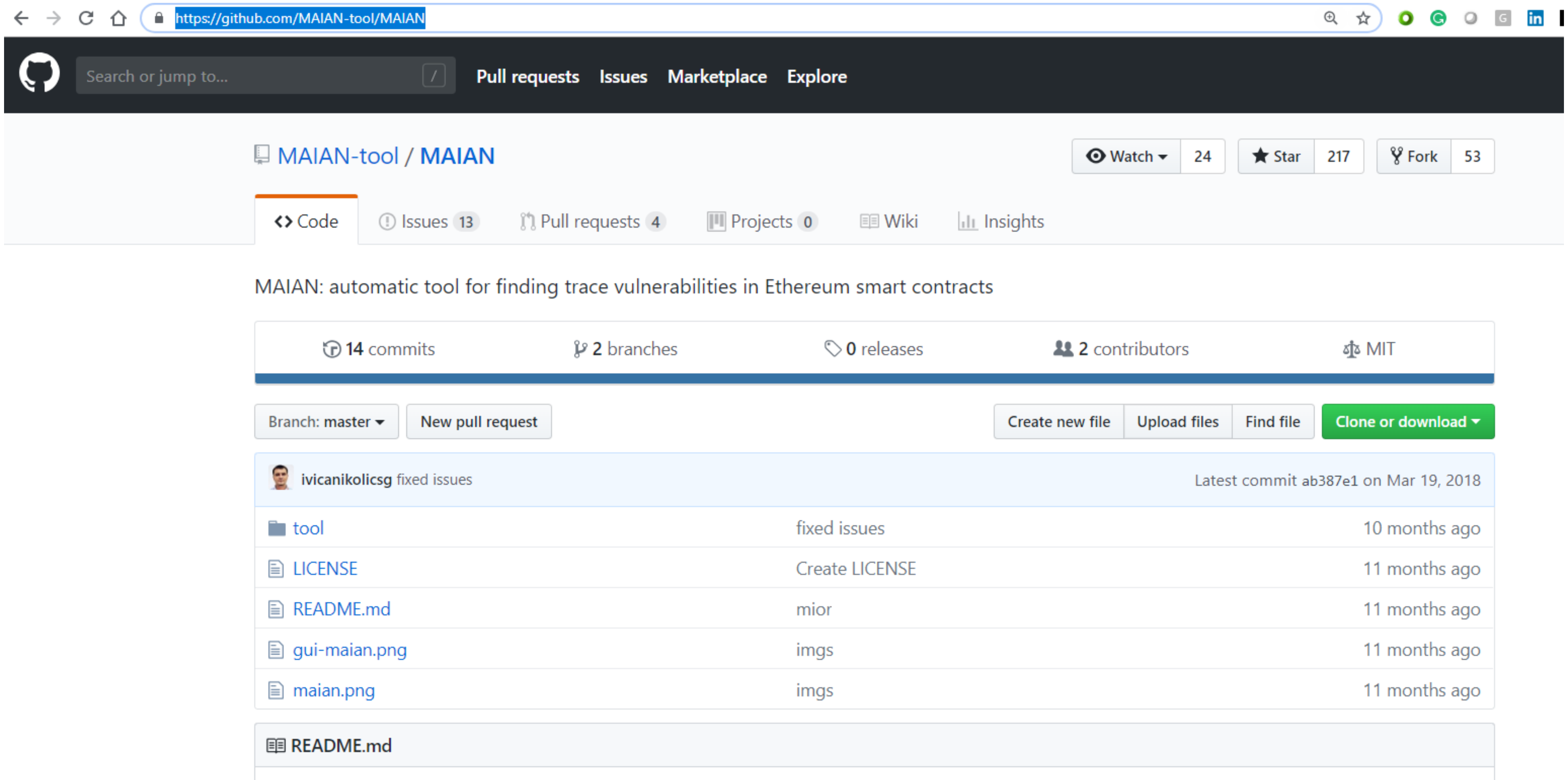
Another interesting point raised in the paper is the unavailability of smart contract source code for Ethereum smart contracts, estimating the number at only one percent of the 970 thousand contracts they analyzed.

Fixing serious security vulnerabilities at scale requires **peer review**, and the **culture of propriety on the Ethereum network** forced the research team to directly analyze EVM bytecode instead of the sources to complete their research. Were the source code for these contracts more available and reviewed, Trace Vulnerabilities on Ethereum may not have proliferated in the first place.

Bottom Line: three to four percent of the smart contracts on Ethereum's blockchain still contain trace vulnerabilities, according to the researchers' new analysis methodology.

Sources: https://www.reddit.com/r/Bitcoin/comments/7ys5nq/pdf_finding_the_greedy_prodigal_and_suicidal/ and https://bitsonline.com/singapore-research-ethereum/

MAIAN



https://github.com/MAIAN-tool/MAIAN

MAIAN-tool / MAIAN

MAIAN: automatic tool for finding trace vulnerabilities in Ethereum smart contracts

14 commits · 2 branches · 0 releases · 2 contributors · MIT

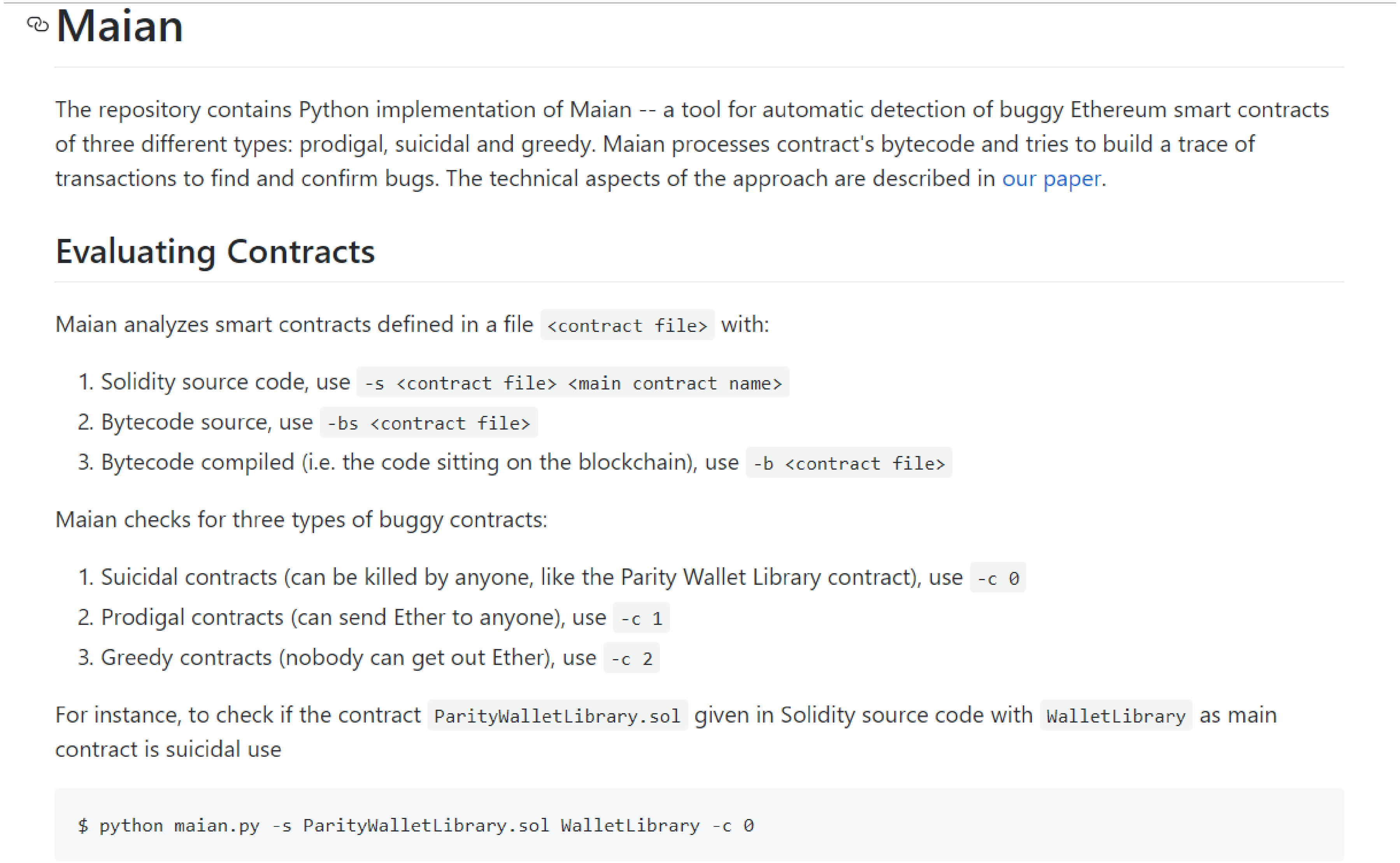
ivicanikolicsg fixed issues — Latest commit ab387e1 on Mar 19, 2018

tool	fixed issues	10 months ago
LICENSE	Create LICENSE	11 months ago
README.md	mior	11 months ago
gui-maian.png	imgs	11 months ago
maian.png	imgs	11 months ago

README.md

Source https://github.com/MAIAN-tool/MAIAN

MAIAN



Maian

The repository contains Python implementation of Maian -- a tool for automatic detection of buggy Ethereum smart contracts of three different types: prodigal, suicidal and greedy. Maian processes contract's bytecode and tries to build a trace of transactions to find and confirm bugs. The technical aspects of the approach are described in our paper.

Evaluating Contracts

Maian analyzes smart contracts defined in a file `<contract file>` with:

1. Solidity source code, use `-s <contract file> <main contract name>`
2. Bytecode source, use `-bs <contract file>`
3. Bytecode compiled (i.e. the code sitting on the blockchain), use `-b <contract file>`

Maian checks for three types of buggy contracts:

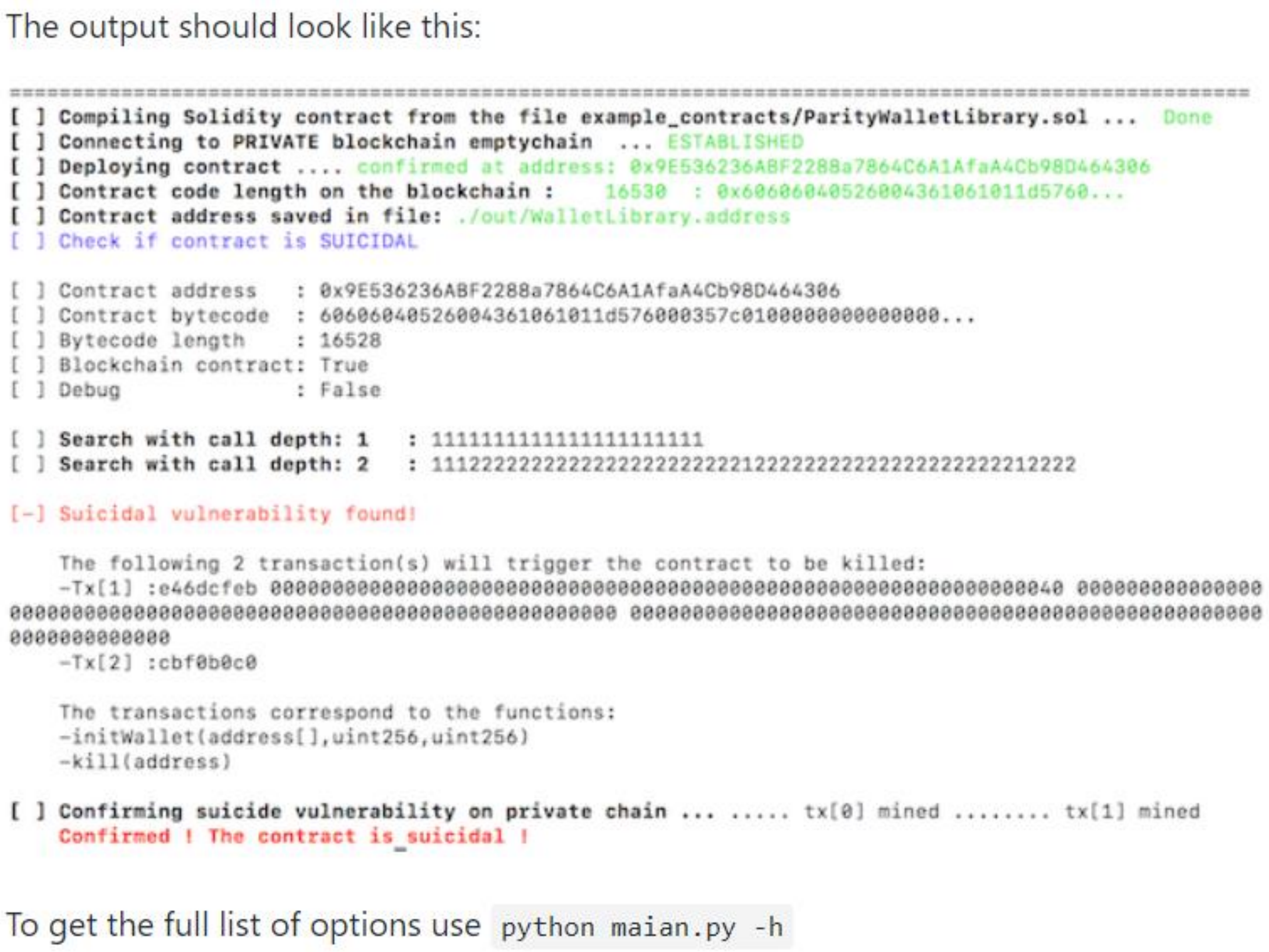
1. Suicidal contracts (can be killed by anyone, like the Parity Wallet Library contract), use `-c 0`
2. Prodigal contracts (can send Ether to anyone), use `-c 1`
3. Greedy contracts (nobody can get out Ether), use `-c 2`

For instance, to check if the contract `ParityWalletLibrary.sol` given in Solidity source code with `WalletLibrary` as main contract is suicidal use

```
$ python maian.py -s ParityWalletLibrary.sol WalletLibrary -c 0
```

Source https://github.com/MAIAN-tool/MAIAN

MAIAN



The output should look like this:

```
====================================================================================================
[ ] Compiling Solidity contract from the file example_contracts/ParityWalletLibrary.sol ...  Done
[ ] Connecting to PRIVATE blockchain emptychain  ... ESTABLISHED
[ ] Deploying contract .... confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain :    16530  : 0x60606040526004361061011d5760...
[ ] Contract address saved in file: ./out/WalletLibrary.address
[ ] Check if contract is SUICIDAL

[ ] Contract address   : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode  : 60606040526004361061011d576000357c0100000000000000...
[ ] Bytecode length    : 16528
[ ] Blockchain contract: True
[ ] Debug              : False

[ ] Search with call depth: 1   : 1111111111111111111111
[ ] Search with call depth: 2   : 1112222222222222222222222212222222222222222222212222

[-] Suicidal vulnerability found!

    The following 2 transaction(s) will trigger the contract to be killed:
    -Tx[1] :e46dcfeb 0000000000000000000000000000000000000000000000000000000000000040 000000000000000
0000000000000000000000000000000000000000000000000 000000000000000000000000000000000000000000000000000
0000000000000
    -Tx[2] :cbf0b0c0

    The transactions correspond to the functions:
    -initWallet(address[],uint256,uint256)
    -kill(address)

[ ] Confirming suicide vulnerability on private chain ... ..... tx[0] mined ........ tx[1] mined
    Confirmed ! The contract is_suicidal !
```

To get the full list of options use `python maian.py -h`

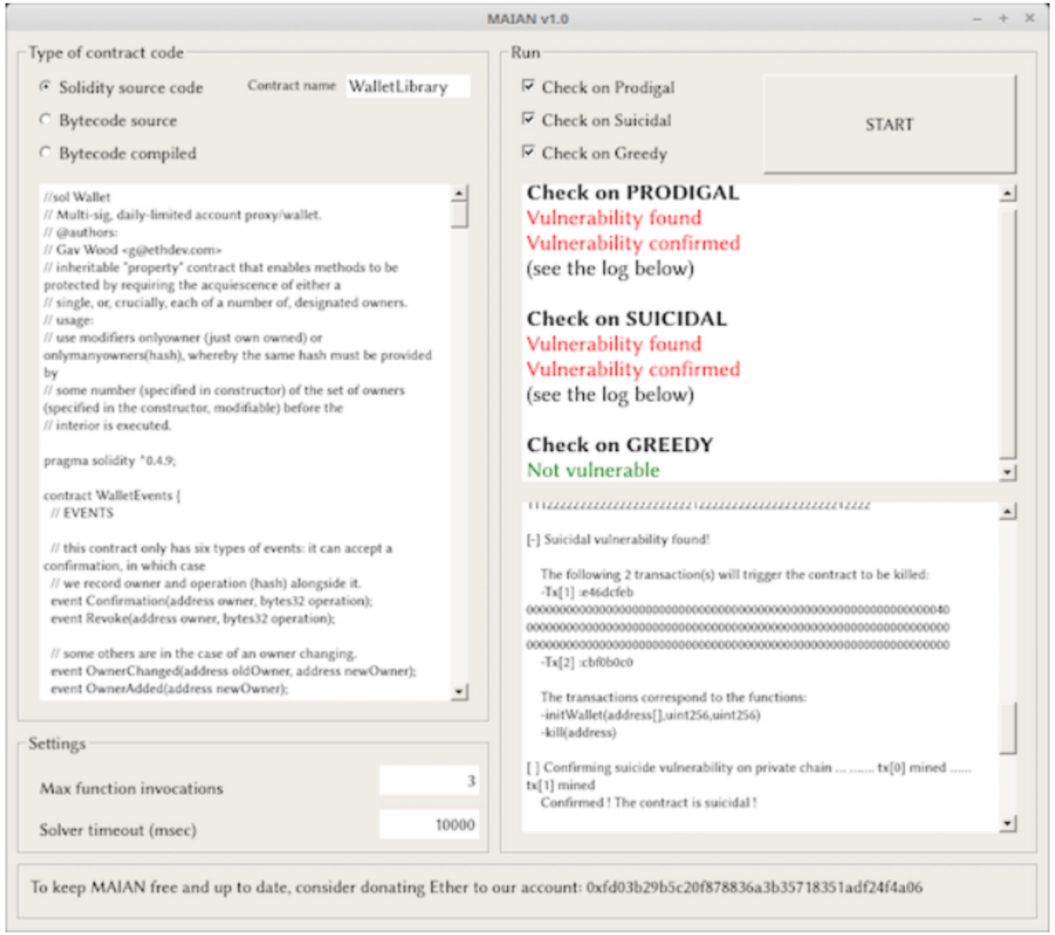
Source https://github.com/MAIAN-tool/MAIAN

#NACACS

MAIAN

GUI

For GUI inclined audience, we provide a simple GUI-based Maian. Use `python gui-maian.py` to start it. A snapshot of one run is given below



● Solidity source code
○ Bytecode source
○ Bytecode compiled

Contract name: WalletLibrary

☑ Check on Prodigal
☑ Check on Suicidal
☑ Check on Greedy

```
//sol Wallet
// Multi-sig, daily-limited account proxy/wallet.
// @authors:
// Gav Wood <g@ethdev.com>
// inheritable "property" contract that enables methods to be
protected by requiring the acquiescence of either a
// single, or, crucially, each of a number of, designated owners.
// usage:
// use modifiers onlyowner (just own owned) or
onlymanyowners(hash), whereby the same hash must be provided
by
// some number (specified in constructor) of the set of owners
(specified in the constructor, modifiable) before the
// interior is executed.

pragma solidity ^0.4.9;

contract WalletEvents {
  // EVENTS

  // this contract only has six types of events: it can accept a
confirmation, in which case
  // we record owner and operation (hash) alongside it.
  event Confirmation(address owner, bytes32 operation);
  event Revoke(address owner, bytes32 operation);

  // some others are in the case of an owner changing.
  event OwnerChanged(address oldOwner, address newOwner);
  event OwnerAdded(address newOwner);
```

```
Check on PRODIGAL
Vulnerability found
Vulnerability confirmed
(see the log below)

Check on SUICIDAL
Vulnerability found
Vulnerability confirmed
(see the log below)

Check on GREEDY
Not vulnerable
```

```
[-] Suicidal vulnerability found!

   The following 2 transaction(s) will trigger the contract to be killed:
   -Tx[1] :e46dcfeb
0000000000000000000000000000000000000000000000000000000000000040
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
   -Tx[2] :cbf0b0c0

   The transactions correspond to the functions:
   -initWallet(address[],uint256,uint256)
   -kill(address)

[ ] Confirming suicide vulnerability on private chain ... ....... tx[0] mined ......
tx[1] mined
   Confirmed ! The contract is suicidal !
```

Settings

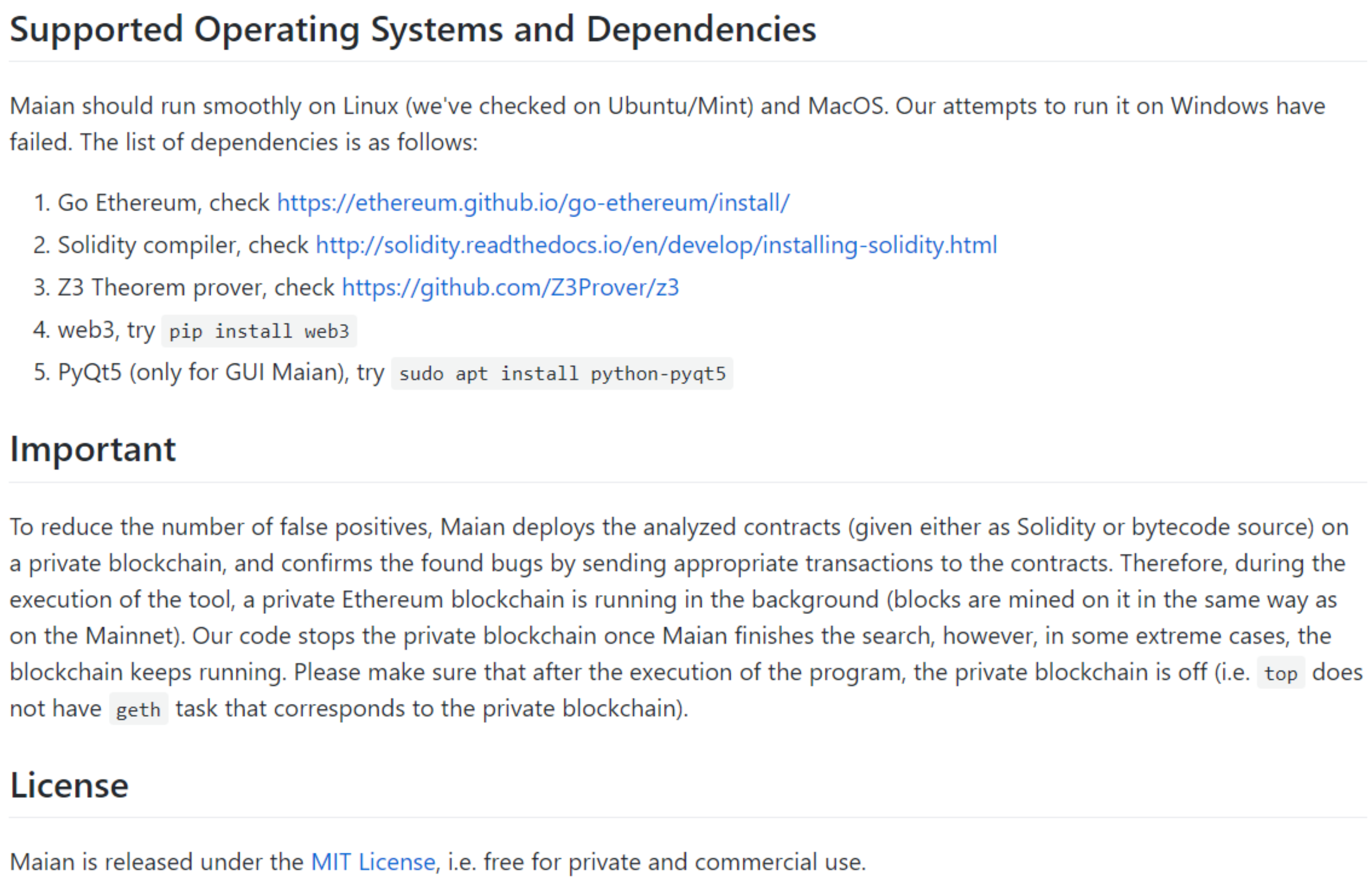
Max function invocations: 3

Solver timeout (msec): 10000

To keep MAIAN free and up to date, consider donating Ether to our account: 0xfd03b29b5c20f878836a3b35718351adf24f4a06

Source https://github.com/MAIAN-tool/MAIAN

MAIAN



Supported Operating Systems and Dependencies

Maian should run smoothly on Linux (we've checked on Ubuntu/Mint) and MacOS. Our attempts to run it on Windows have failed. The list of dependencies is as follows:

1. Go Ethereum, check https://ethereum.github.io/go-ethereum/install/
2. Solidity compiler, check http://solidity.readthedocs.io/en/develop/installing-solidity.html
3. Z3 Theorem prover, check https://github.com/Z3Prover/z3
4. web3, try `pip install web3`
5. PyQt5 (only for GUI Maian), try `sudo apt install python-pyqt5`

Important

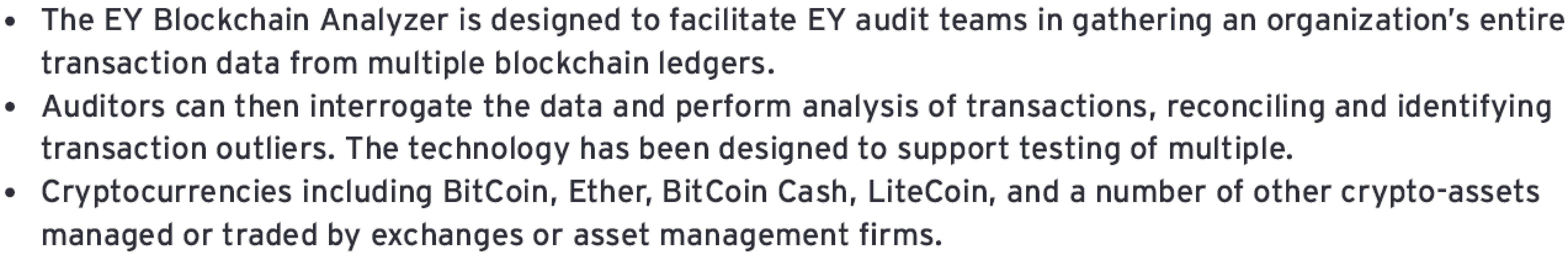
To reduce the number of false positives, Maian deploys the analyzed contracts (given either as Solidity or bytecode source) on a private blockchain, and confirms the found bugs by sending appropriate transactions to the contracts. Therefore, during the execution of the tool, a private Ethereum blockchain is running in the background (blocks are mined on it in the same way as on the Mainnet). Our code stops the private blockchain once Maian finishes the search, however, in some extreme cases, the blockchain keeps running. Please make sure that after the execution of the program, the private blockchain is off (i.e. `top` does not have `geth` task that corresponds to the private blockchain).

License

Maian is released under the MIT License, i.e. free for private and commercial use.

Source https://github.com/MAIAN-tool/MAIAN

EY has a new Tool, Blockchain Analyzer with the Capability to Automate the Auditing of Blockchain Applications



- The EY Blockchain Analyzer is designed to facilitate EY audit teams in gathering an organization's entire transaction data from multiple blockchain ledgers.
- Auditors can then interrogate the data and perform analysis of transactions, reconciling and identifying transaction outliers. The technology has been designed to support testing of multiple.
- Cryptocurrencies including BitCoin, Ether, BitCoin Cash, LiteCoin, and a number of other crypto-assets managed or traded by exchanges or asset management firms.

Conclusion

Conclusion

We covered:

- Getting started with Blockchain Application Development – Setting up the Workbench
- Truffle Framework Introduction
- Example DApp using Truffle, HTML, CSS, Solidity, the EVM and Ethereum Blockchain
- Solidity and Ethereum Blockchain Fundamentals
- Javascript and Ethereum Blockchain Fundamentals
- Example DApp using HTML, CSS, Solidity the EVM and the Ethereum Blockchain
- How to Secure Blockchain infrastructure and applications
- How to perform Secure Software Development for Blockchain applications by design, coding
- Blockchain and Auditing practices, testing and verification
- Concepts of Auditing the Data and Transactions in Blockchain Data Structures
- Automating the Auditing of Blockchains and Blockchain Applications

Conclusion

BLOCKCHAIN MUCH bigger than you think.

Blockchain is moving SO FAST that a **“Blockchain Year”** is considered to be about 30 days

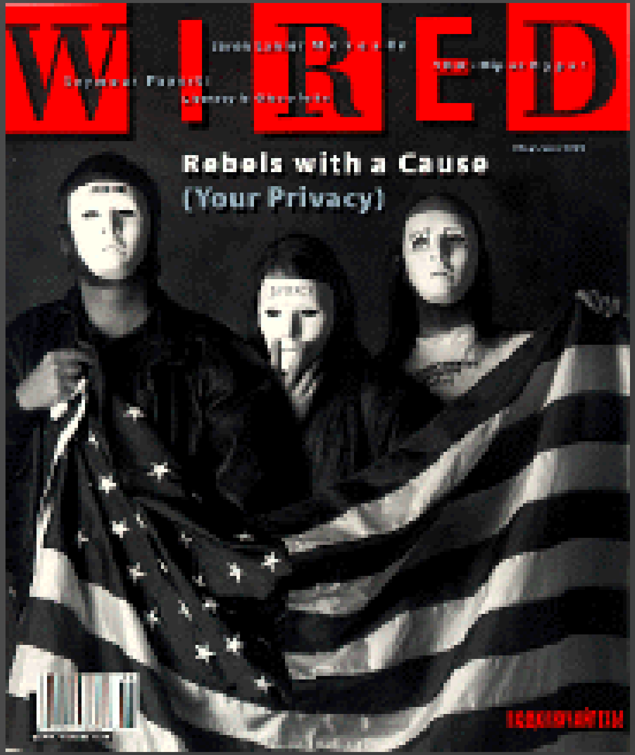
I have multiple decades of experience in software and application development. To say the **learning curve** “**humbling**” would be an ***understatement***.

The only way you will get to be excellent in this:

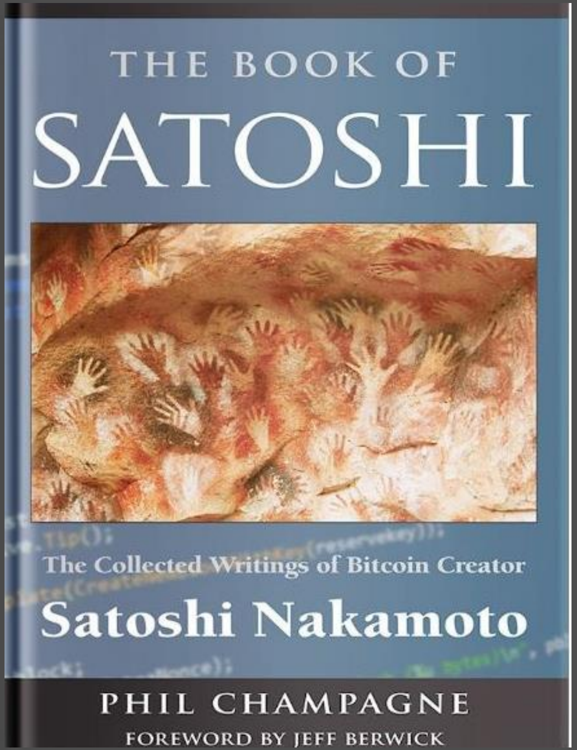
- **Hard Work & Perseverance** http://www.billslater.com/uop/persistence.htm
- **Read great Blockchain Development Resources and Authors**
- **Hands-on Practice**
- **Hanging out with Developers who are knowledgeable, kind, & sharing**
- **Participate in User Groups and Meet-ups that have excellent speakers and programs**
- **Don’t ever underestimate the difficulty and the level of effort required to become competent at this**

Questions

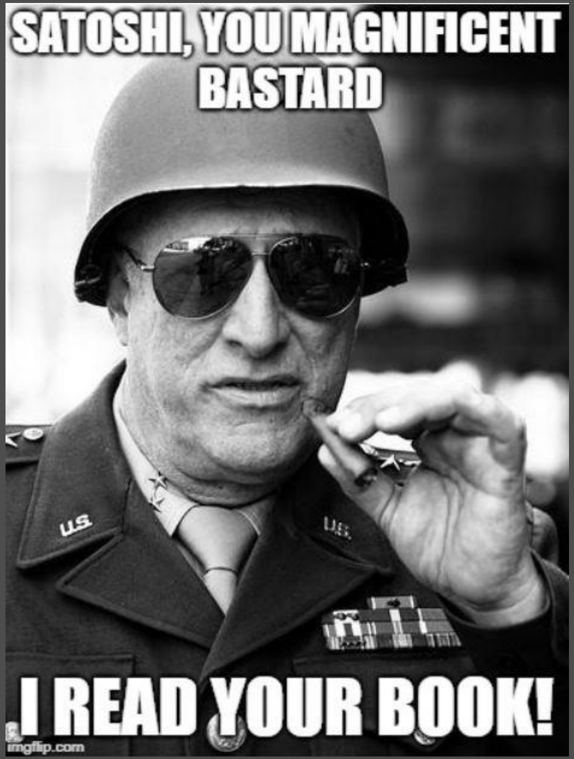
Questions?



Crypto Rebels Revealed
Wired Magazine, February 1993



Book of Satoshi
Collected Writings
Of Satoshi Nakamoto



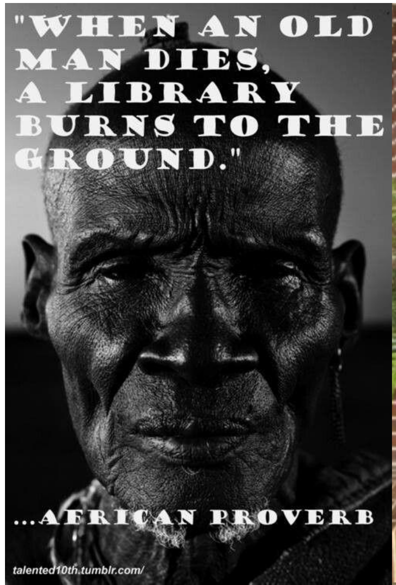
General George S. Patton

Dedication & Thanks

Dedication

This work is dedicated with love, admiration, gratitude, and great respect to ***James P. Jarnagin*** (January 25, 1935 – December 2, 2018), the Man who was my Mentor and Father-figure since March 1985. He is one of the biggest reasons for my career success and personal success. What I owe him can never be repaid.

We'll meet again, Jim. You can count on it…



"WHEN AN OLD MAN DIES, A LIBRARY BURNS TO THE GROUND."

...AFRICAN PROVERB

talented10th.tumblr.com/

June 2013

June 1994

October 2015

Special Thanks To:

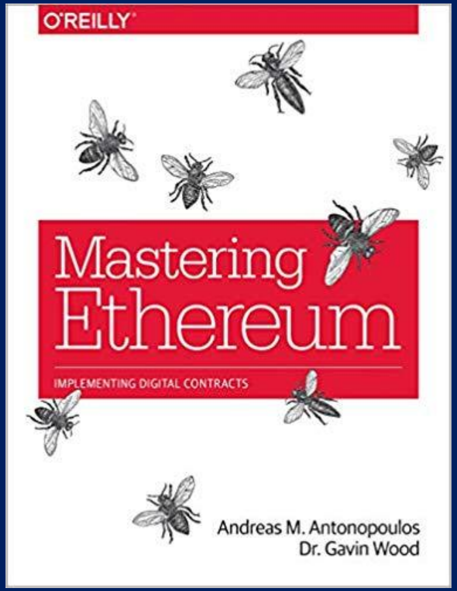
CBP
CHICAGO BLOCKCHAIN PROJECT

Joe Hernandez
Co-Founder of the
Chicago Blockchain Project

COMMUNITY CHICAGO · VIRES IN NUMERIS
B.O.B.
BITCOIN AND OPEN BLOCKCHAIN

Hannah Rosenburg
Director at the Chicago Blockchain Institute an
Co-Founder of the
Chicago Bitcoin and Open
Blockchain Meetup (3800 Members!)

Special Thanks To:



**Andreas Antonopoulos
and Dr. Gavin Wood
Co-authors of
Mastering Ethereum**

Special Thanks To:

Vitalik Buterin
Inventor of Ethereum
@VitalikButerin on
#Twitter

References

References

- Antonopoulos, A. M. (2018). Mastering Bitcoin: Programming the Open Blockchain, second edition. Sebastopol, CA: O'Reilly Media, Inc.
- Antonopoulos, A. M. and Wood, G. (2019). Mastering Ethereum: Building Smart Contract sand DApps. Sebastopol, CA: O'Reilly Media, Inc.
- Associated Press. (2014). Mt. Gox finds 200,000 missing bitcoins. Retrieved from http://money.msn.com/business-news/article.aspx?feed=AP&date=20140321&id=17454291 on March 21, 2014.
- Bahga, A. and Madisetti, V. (2017). Blockchain Applications: A Hands-On Approach. Published by Arshdeep Bahga and Vijay Madisetti. www.blockchain-book.com .
- Bambara, J. J. and Allen P. R. (2018). Blockchain: A Practical Guide to Developing Business, Law, and Technology Solutions. New York, NY: McGraw-Hill Education.
- Bashir, I. (2018). Mastering Blockchain, second edition. Birmingham, UK: Packt Publishing Ltd.
- BBC. (2014). Troubled MtGox Bitcoin boss emerges after shut down Retrieved from http://www.bbc.com/news/technology-26352442 on February 26, 2014.
- Bitcoin.org. (2014). Bitcoin.org FAQs.. Retrieved from https://bitcoin.org/en/faq on April 10, 2014.
- Bitcoin Scammers. (2014). Bit Coin Scammers. Retrieved from http://bitcoinscammers.com/ on April 9, 2014.
- Casey, M. J. and Vigna, P. (2018). The Truth Machine: The Blockchain Reference and the Future of Everything. New York, NY: St. Martin's Press.
- Caughey, M. (2013). Bitcoin Step by Step, second edition. Amazon Digital Services.

References

•Champagne, P. (2014). The Book of Satoshi: The Collected Writings of Bitcoin Creator Satoshi Nakamoto. Published by E53 Publishing, LLC.

•Dannen, C. (2017). Introducing Ethereum and Solidity: Foundations of Crytocurrency and Blockchain Programming for Beginners. New York, NY: Apress

•De Filippi, P. and Wright, A. (2018). Blockchain and the Law: the Rule of Code. Cambridge, MA: President and Fellows of Harvard College.

•De Havilland, P. (2018). Greedy, Prodigal, and Suicidal — Hosho to Save Smart Contracts From Three Deadly Sins. An article published at Bitsonline.com on September 3, 2018. Retrieved from https://bitsonline.com/greedy-prodigal-suicidal-hosho-smart-contracts/ on February 27, 2019.

•Dhillon, V., Metcalf, D., and Hooper, M. (2017). Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Nake It Work for You. New York, NY: Apress.

•Drescher, D. (2017). Blockchain Basics. Frankfort am Main, Germany: Apress.

•Eddison, L. (2017). Ethereum: A Deep Dive into Ethereum. Published by Leonard Eddison.

•Etwaru, R. (2017). Blockchain Trust Companies. Indianapolis, IN: Dog Ear Publishing.

•Ferry, T. (2019). To Blockchain or not to Blockchain. An article publsihed at Medium.com on June 8, 2018. Retrieved on January 13, 2019 from https://medium.com/causys/to-blockchain-or-not-to-blockchain-aed05bf08150 .

•Gerard, D. (2107), Attack of the 50 Foot Blockchain: Bitcoin, Blockchain, Ethereum, and Smart Contracts. Published by David Gerard. www.davidgerard.co.uk/blockchain .

•GreenBerg, A. (2019). A BlockchainBandit Is Guessing Private Keys and Scoring Millions, An article published on April 23, 2019 at Wired.com and retrieved from https://www.wired.com/story/blockchain-bandit-ethereum-weak-private-keys/ on April 23, 2019.

References

•Hornyak, T. (2014). 'Malleability' attacks not to blame for Mt. Gox's missing bitcoins, study says. Retrieved from http://www.pcworld.com/article/2114200/malleability-attacks-not-to-blame-for-mt-goxs-missing-bitcoins-study-says.html on March 27, 2014.

•Incencio, R. (2014). Ransomware and Bitcoin Theft Combine in BitCrypt. Retrieved from http://blog.trendmicro.com/trendlabs-security-intelligence/ransomware-and-bitcoin-theft-combine-in-bitcrypt/ on March 27, 2014.

•Laurence, T. (2017). Blockchain for Dummies. Hoboken, NJ: John Wiley & Sons, Inc.

•Lee, T. B. (2013). 12 questions about Bitcoin you were too embarrassed to ask. Retrieved from http://www.washingtonpost.com/blogs/the-switch/wp/2013/11/19/12-questions-you-were-too-embarrassed-to-ask-about-bitcoin/ on November 19, 2013.

•Ma, M. (2017). Blockchain Design Sprint: An Agile Innovation Workbook to Implement an Agile Design Sprint for your Blockchain Business. Published by Future Lab www.futurelabconsulting.com .

•Markowitz, E. (2014). Cryptocurrencies Are the New Spam Frontier. Retrieved from http://www.vocativ.com/tech/bitcoin/cryptocurrencies-new-spam-frontier/ on March 28, 2014.

•Nakamoto. S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved from https://bitcoin.org/bitcoin.pdf on November 1, 2013.

•Nguyen, J. (2019). Blockchain still vulnerable to hacks despite security hype, but here are some solutions. Retrieved from https://e27.co/blockchain-still-vulnerable-to-hacks-despite-security-hype-but-here-are-some-solutions-20190212/ on February 13, 2019.

•O'Ham, T. (2018). Singapore Research Team Codifies 3 new Ethereum VM Vulnerabilities. An article published at Bitsonline.com on February 21, 2018. Retrieved from https://bitsonline.com/singapore-research-ethereum/ on February 27, 2019.

•Orcutt, M. (2019). Once Hailed as Unhackable, Blockchains Are now Getting Hacked. An article in MIT Review. Published February 19, 2019. Retrieved from https://www.technologyreview.com/s/612974/once-hailed-as-unhackable-blockchains-are-now-getting-hacked/ on February 24, 2019.

References

•Popper, N. (2013). Into the Bitcoin Mines, Retrieved from http://dealbook.nytimes.com/2013/12/21/into-the-bitcoin-mines/?hp&_r=0 on December 21, 2013.

•Prusty, N. (2017). Building Blockchain Projects: Building Decentralized Blockchain Applications with Ethereum and Solidity. Birmingham, UK: Pact Publishing.

•Ramone, A. D. (2019). How to Secure a Blockchain: 3 Things Business Leaders Know. An article publisged at Techrepublic.com on April 18, 2019. Retrieved from https://www.techrepublic.com/article/how-to-secure-a-blockchain-3-things-business-leaders-need-to-know/ on April 23, 2019.

•SCGNEWS. (2014). The IRS Just Declared War on Bitcoin - Retroactively. Retrieved from http://scgnews.com/the-irs-just-declared-war-on-bitcoin-retroactively on March 27, 2014.

•Sharkey, T. (2014. Inside Bitcoins NYC Day 1: Bitcoin 2.0 Takes Center Stage. Retrieved from http://www.coindesk.com/inside-bitcoins-nyc-day-1-bitcoin-2-0-takes-center-stage/ on April 8, 2014.

•Zenko, M. (2017). Bitcoins for Bombs – a Blog published at the Council on Foreign Relations on August 17, 2017. Retrieved from https://www.cfr.org/blog/bitcoin-bombs on February 13, 2019.

References – Best Blockchain Books

- **Mastering Ethereum**

– by Andreas M. Antonopoulos and Dr. Gavin Wood

- **Blockchain Applications: A Hands-On Approach**

–by Arshdeep Bahga and Vijay Madisetti

- **Building Ethereum DApps**

–By Roberto Infante

- **Truffle Quick Start Guide**

–by Nikhil Bhaskar

- **Mastering Blockchain - Second Edition**

–by Imran Bashir

- **Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners**

–By Chris Dannen

- **Ethereum, Tokens & Smart Contracts: Notes on getting started**

–by Eugenio Noyola

- **Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You**

–by Vikram Dhillon, David Metcalf, Max Hooper

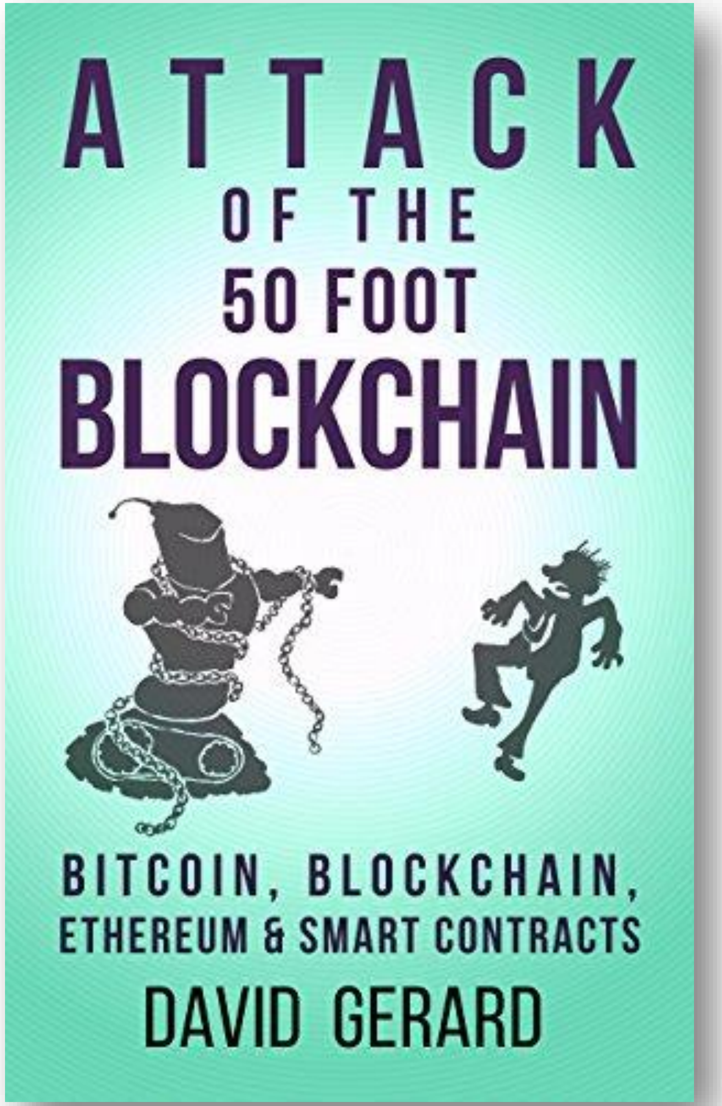
- **Foundations of Blockchain**

–By Koshik Raj

- **The Book of Satoshi: The Collected Writings od Bitcoin Creator Satoshi Nakamoto**

–By Phil Champagne

References – For a Cynical & Humorous View of Blockchain



References – 12 Free Blockchain Resources

1. William Slater’s Blockchain Resource Page http://billslater.com/blockchain
2. Factom University http://www.factom.com/university
3. Ethereum 101 http://www.ethereum101.org
4. Build on Ripple http://ripple.com/build
5. Programmable money by Ripple https://goo.gl/g8vFPL
6. DigiKnow https://youtu.be/scr68zFddso
7. Blockchain University http://blockchainu.co
8. Bitcoin Core https://bitcoin.org
9. Blockchain Alliance http://www.blockchainalliance.org
10. Multichain Blog http://www.mutichain,com/blog
11. HiveMind http://bitcoinhivemind.com
12. Chicago Blockchain Project http://chicagoblockchainproject.com/
13. Chicago Bitcoin and Open Blockchain Meetup Group https://www.meetup.com/Bitcoin-Open-Blockchain-Community-Chicago/

References – 10 Rules to Never Break the Blockchain

1. Don’t use Cryptocurrency or Blockchain to Skirt the Law
2. Keep your contracts as simple as possible
3. Publish with great caution
4. Back Up, Back Up, Back Up Your Private Keys
5. Triple-check the Address Before Sending Currency
6. Take Care When Using Exchanges
7. Beware Wi-Fi
8. Identify Your Blockchain Dev
9. Don’t Get Suckered
10. Don’t Trade Tokens Unless You Know What You’re Doing

References – 10 Free Blockchain Projects

- The R3 Consortium http://www.r3cev.com
- T ZERO: Overstocking the Stock Market http://www.overstock.com
- Blockstream's Distributed Systems http://www.blockstream.com
- OpenBazaar's Blockchain http://www.openbazaar.com
- Code Valley: Find Your Coder http://www.codevalley.com
- Bitfury's Digital Assets http://www.bitfury.com
- Any Coin Can Shapeshift http://www.shapeshift.io
- Machine-Payable Apps on 21 http://www.21.co
- Anonymous Transactions on Dash http://www.dash.org
- ConsenSys: Decentralized Applications: http://www.consensys.net

William Favre Slater, II

- 312-758-0307
- slater@billslater.com
- williamslater@gmail.com
- http://billslater.com/interview
- 1515 W. Haddon Ave., Unit 309
  Chicago, IL 60642
  United States of America

2019 NORTH AMERICA CACS™ AN ISACA EVENT

Thank You!

Now, Let’s Go Build Something Beautiful on or for Blockchain…

PEOPLE WILL FORGET
WHAT YOU SAID.
PEOPLE WILL FORGET
WHAT YOU DID.
BUT PEOPLE WILL
NEVER FORGET HOW
YOU MADE THEM FEEL.

Maya Angelou

January 16, 2019