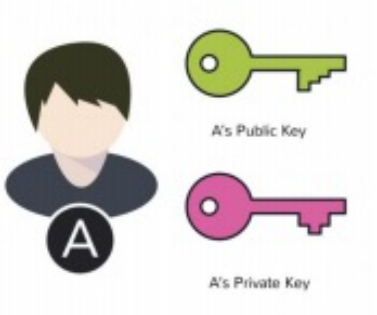
# Empowering Decentralization through Smart Contracts

**Texas A&M Blockchain Day**
**May 1, 2023**

**Oshani Seneviratne**
Associate Director, Tetherless World Constellation
Assistant Professor, Department of Computer Science
Rensselaer Polytechnic Institute, Troy NY USA

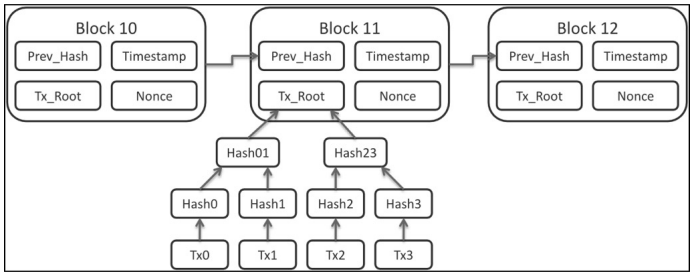
# Key Build Blocks of Blockchain Technologies

Blockchain technologies are built on top of the following:




Cryptographic Hashes and Identities

Consensus Protocol




Ledger aka “Chain”

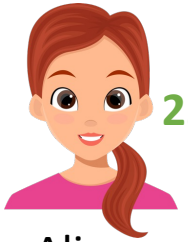
Smart Contract


2

Rensselaer

# Example (Bitcoin) Transactions


25 coins



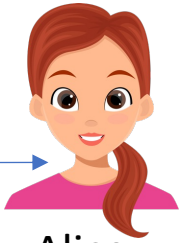
Alice

**25 coins**

Carol

5 coins

**13 coins**



Alice

**8 coins**

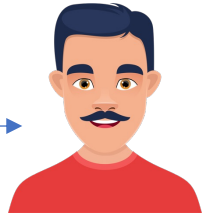
Alice

17 coins

Bob

Alice

15 coins



David

**13 coins < 15 coins**

**Can Alice perform this last transaction?**

Bob

8 coins

Carol

# Bitcoin Transactions

One way to organize a ledger

| |
|---|
| Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$ |
| Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$ |
| Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$ |
| Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$ |
| Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$ |

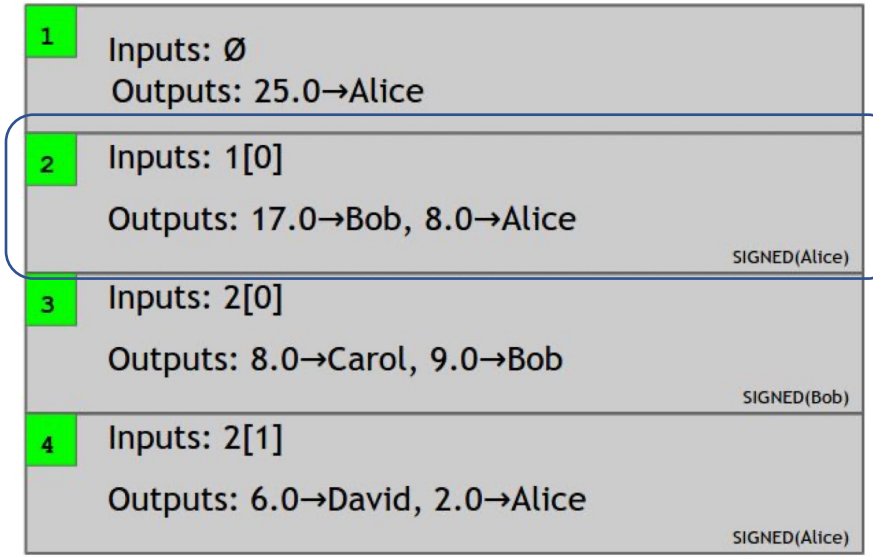
How can we build a currency on top of such a ledger?

The downside to this way of doing things is that anyone who wants to determine if a transaction is valid will have to keep track of these account balances.

Rensselaer

# Bitcoin's way of organizing the ledger

Bitcoin doesn't use an account-based model.




Bitcoin uses a ledger that just keeps track of transactions.

Transactions specify a number of inputs and a number of outputs

You can think of the inputs as coins being consumed (created in a previous transaction) and the outputs as coins being created

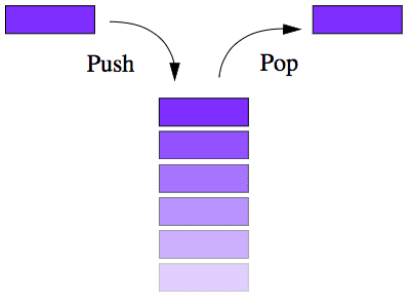
Why does Alice have to send money to herself?

When a new tx is added, how easy is it to check if it is valid?

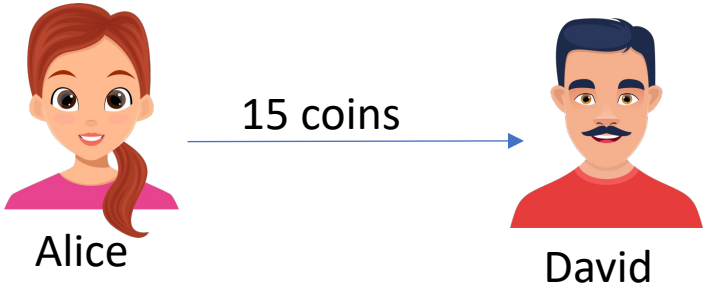
# An Actual Bitcoin Transaction

# Bitcoin Scripting Language

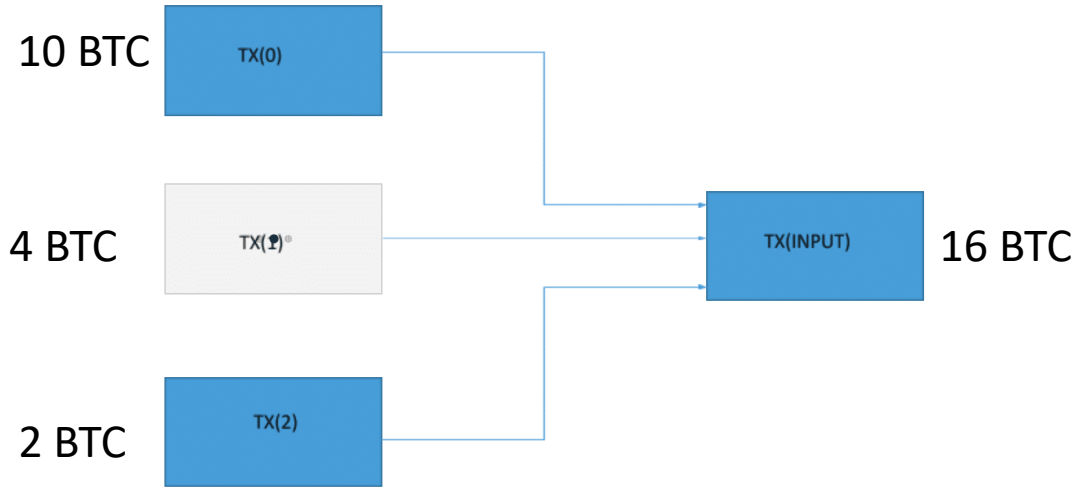
- Has a fixed set of “Op Codes” or instructions:
  - A total of 256 – 15 are disabled, 75 are reserved
  - Basic functions – arithmetic, conditionals
  - Crypto functions – hash functions, signature verifications
- Turing Incomplete
  - Does not allow infinite loops
  - Advantage: does not run malformed/malicious scripts
  - Disadvantage: does not allow for complex logic to build applications on the blockchain
- Reverse-Polish Notation
  - The operators follow operands, e.g., “1 + 2” is written as “1 2 +”
- Stack-based
  - Last-In-First-Out (LIFO)



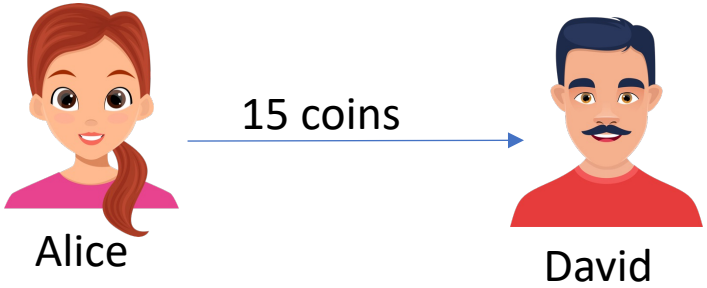
# Bitcoin Scripts in Action



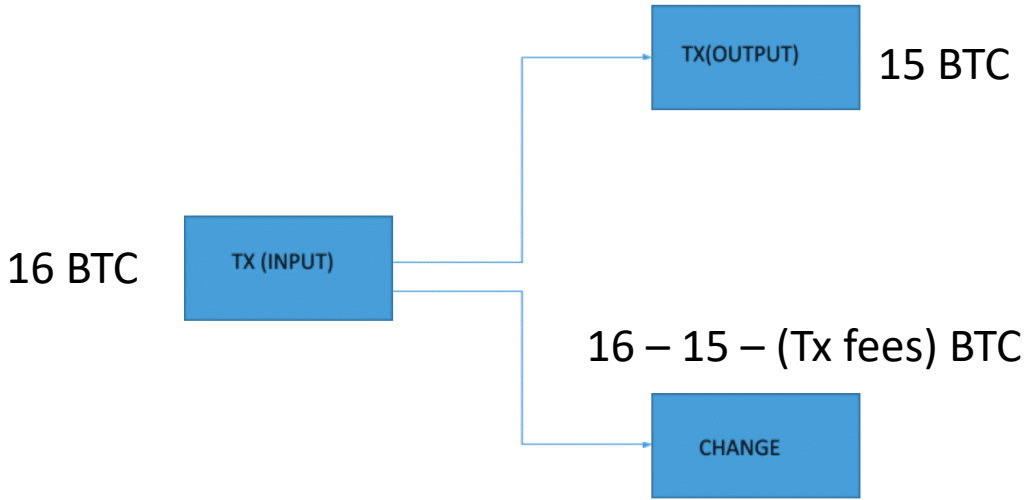

- Transaction Input
  - Alice needs to get bitcoins which she has received from various previous transactions.
  - Suppose Alice needs to pull bitcoins from the following transactions which we shall name TX(0), TX(1) and TX(2)



# Bitcoin Scripts in Action

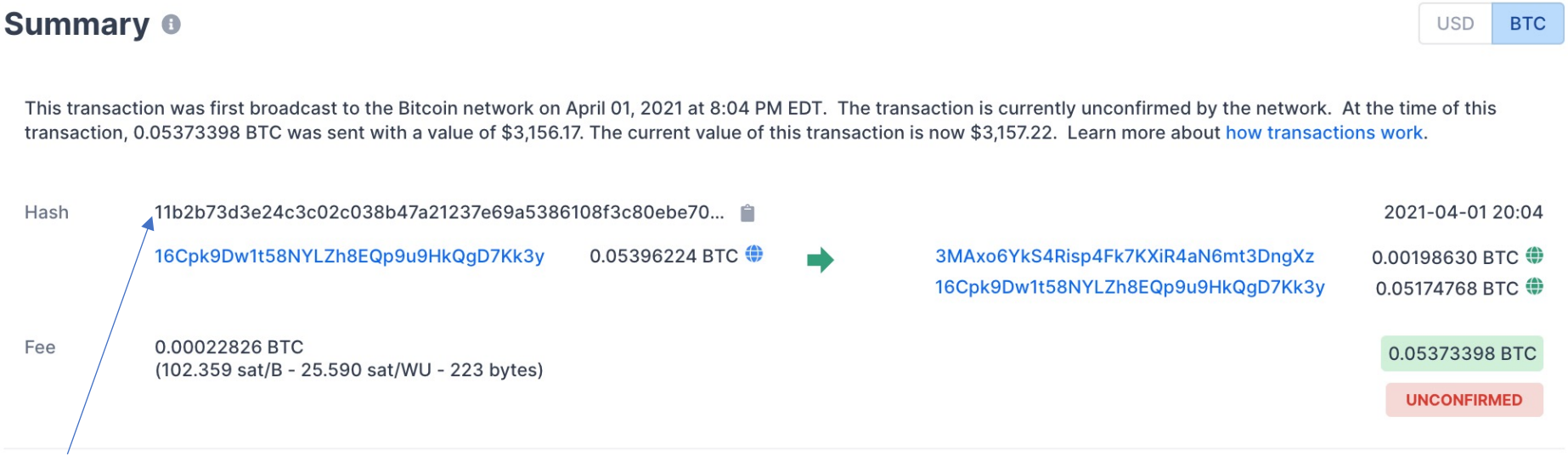



- Transaction Output will have the number of bitcoins that Bob will possess, post-transaction.
- Any remaining change that is left over is sent back to Alice.
- Conditions of a transaction
  - *TX(Input) > TX(output)*
  - Transaction fees = *TX(Input) – (TX(output) + Change).*



Rensselaer

# Bitcoin Scripts in Action: Behind the Scenes




Name of the transaction

https://www.blockchain.com/btc/tx/11b2b73d3e24c3c02c038b47a21237e69a5386108f3c80ebe7051eb939cbaf38

# Bitcoin Scripts in Action: Behind the Scenes

## Inputs

HEX | ASM

| | | | |
|---|---|---|---|
| Index | 0 | Details | Output |
| Address | 16Cpk9Dw1t58NYLZh8EQp9u9HkQgD7Kk3y | Value | 0.05396224 BTC |
| Pkscript | OP_DUP<br>OP_HASH160<br>39150f1cf405f2f81258b3cba0f959d850fc0412<br>OP_EQUALVERIFY<br>OP_CHECKSIG | | |
| Sigscript | 304402204e8250d2b7ca777a4244653fb0100e0b7ce334acce55bd89b7c403dddadd0b21022077e794195826aedb7cedcf40b4514f389d41034b638b53817a5f70226d81b8d801<br>020c5b01a17d444499b91b42cdf311f7c17fcf3e8249cae1ffe694e45ab282d5c2 | | |

https://www.blockchain.com/btc/tx/11b2b73d3e24c3c02c038b47a21237e69a5386108f3c80ebe7051eb939cbaf38

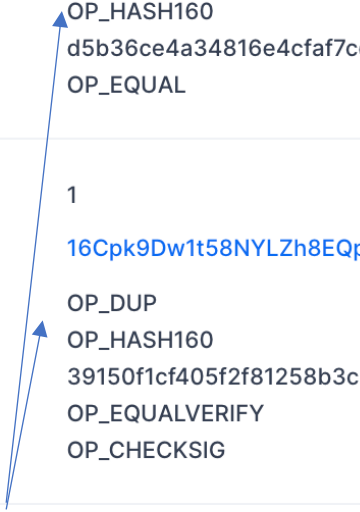
Rensselaer

# Bitcoin Scripts in Action: Behind the Scenes

## Outputs

| | | | |
|---|---|---|---|
| Index | 0 | Details | Unspent |
| Address | 3MAxo6YkS4Risp4Fk7KXiR4aN6mt3DngXz | Value | 0.00198630 BTC |
| Pkscript | OP_HASH160<br>d5b36ce4a34816e4cfaf7cde8dbf8619dcbc3b5c<br>OP_EQUAL | | |

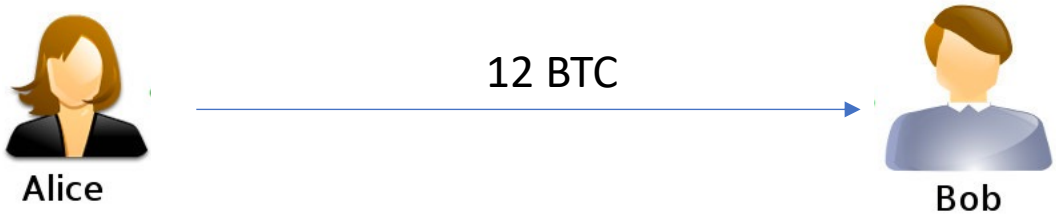
| | | | |
|---|---|---|---|
| Index | 1 | Details | Unspent |
| Address | 16Cpk9Dw1t58NYLZh8EQp9u9HkQgD7Kk3y | Value | 0.05174768 BTC |
| Pkscript | OP_DUP<br>OP_HASH160<br>39150f1cf405f2f81258b3cba0f959d850fc0412<br>OP_EQUALVERIFY<br>OP_CHECKSIG | | |



Script to ”unlock” the unspent transaction outputs (UTXO)

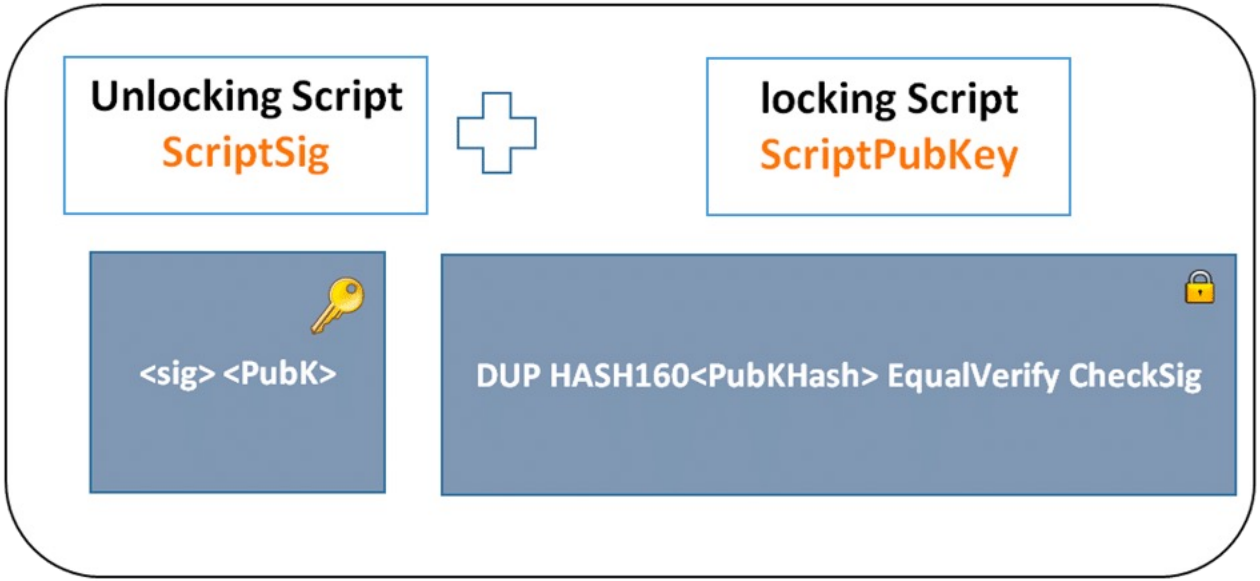
https://www.blockchain.com/btc/tx/11b2b73d3e24c3c02c038b47a21237e69a5386108f3c80ebe7051eb939cbaf38

# Bitcoin Scripts in Action




- Alice sends Bob an output which has the scriptPubKey, which includes Bob's address.
- Bob unlocks the input using his signature of scriptSig which includes his signature and his public key.
- scriptPubKey = OP_DUP OP_HASH160 <Bob's public address> OP_EQUALVERIFY OP_CHECKSIG
- scriptSig = <Bob's signature> <Bob's public key>

# Bitcoin Scripts in Action: A Game of Locking and Unlocking



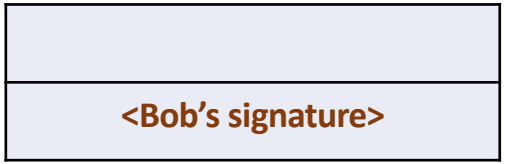

Source: https://www.cryptocompare.com/wallets/guides/bitcoin-transactions-pay-to-address-pay-to-public-key-hash/

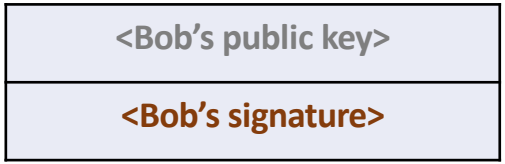
**Script**

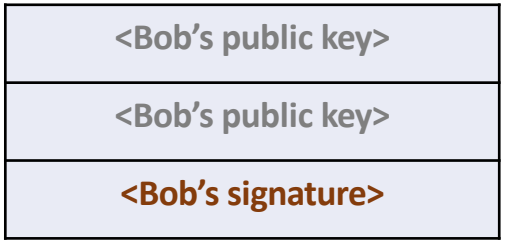
**<Bob's signature>** <Bob's public key> **OP_DUP OP_HASH160 <Bob's public address> OP_EQUALVERIFY OP_CHECKSIG**

# Bitcoin Scripts in Action: Verification

- **<Bob's signature>** <Bob's public key> **OP_DUP** **OP_HASH160** **<Bob's public address>** **OP_EQUALVERIFY** **OP_CHECKSIG**
- For **OP_DUP** pop <Bob's public key> , duplicate it and push it back



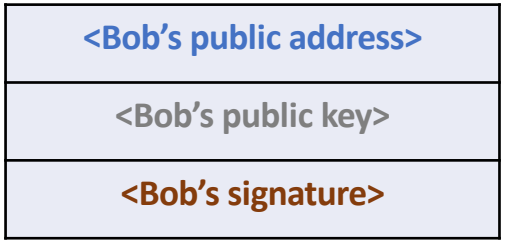



Rensselaer

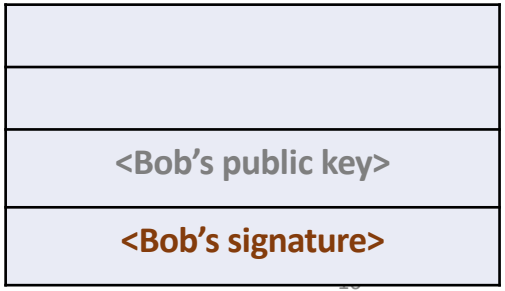
# Bitcoin Scripts in Action: Verification

**<Bob's signature>** **<Bob's public key>** **OP_DUP**
**OP_HASH160** **<Bob's public address>**
**OP_EQUALVERIFY** **OP_CHECKSIG**

- **OP_HASH160** pops **<Bob's public key>** runs it through SHA256 followed by RIPEMOD160 to get **<Bob's public address>**



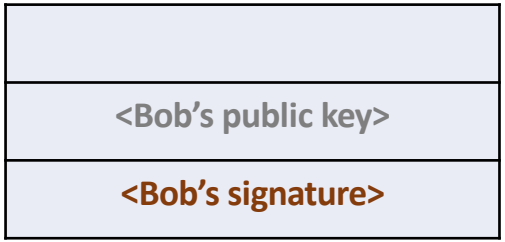

- **OP_EQUALVERIFY** pops the last two elements in the stack and check to see if they are equal or not



# Bitcoin Scripts in Action: Verification

**<Bob's signature>** <Bob's public key> **OP_DUP**
**OP_HASH160** **<Bob's public address>**
**OP_EQUALVERIFY** **OP_CHECKSIG**

- **OP_CHECKSIG** pops <Bob's public key> and **<Bob's signature>** and checks their validity.
  - This is where the Elliptical Curve Digital Signature Algorithm (ECDSA) is used.



# Summary of Bitcoin Scripts

- Stack-based
- Data in the script is enclosed in <>: <sig>, <pubkey>, etc.
- Opcodes: commands or functions
  - Arithmetic, e.g., OP_ABS, OP_ADD
  - Stack, e.g., OP_DROP, OP_SWAP
  - Flow control, e.g., OP_IF, OP_ELSE
  - Bitwise logic, e.g., OP_EQUAL, OP_EQUALVERIFY
  - Hashing, e.g., OP_SHA1, OP_SHA256
  - (Multiple) Signature Verification, e.g., OP_CHECKSIG, OP_CHECKMULTISIG
  - Locktime, e.g., OP_CHECKLOCKTIMEVERIFY, OP_CHECKSEQUENCEVERIFY

# Bitcoin's Scripting Language Limitations

- Lack of Turing completeness: No loops
- Lack of state: Cannot keep internal state.
- Value-blindness: Cannot denominate the amount being sent
- Blockchain-blindness: Cannot access block header values such as nonce, timestamp, and previous hash block.

# Extending Bitcoin Functionality

- Building a protocol on top of Bitcoin:
  - Pros:
    - Take advantage of the underlying network and mining power.
    - Very low development cost.
  - Cons:
    - No flexibility.
- Build an independent network:
  - Pros:
    - Easy to add and extend new opcodes.
    - Flexibility.
  - Cons:
    - Need to attract miners to sustain the network.
    - Difficult to implement

# Alternative (Early) Blockchain Applications

- Namecoin:
  - Bitcoin fork: Currency NMC
  - Decentralized name registration database: DNS, identities etc
- Colored Coins:
  - On top of Bitcoin
  - Allows people to create their own digital currencies
- OmniLayer (formerly Mastercoin)
  - On top of Bitcoin
  - Distributed exchange, smart property, distributed e-commerce, etc
- OpenBazaar
  - On top of Bitcoin
  - Decentralized marketplace

# Better Blockchain Programming Models

# Smart Contracts

- Programatically enforced state updates
  - You can add any functionality you want!
- Can facilitate access to and distribution of funds based on specified conditions
- Can create, transfer, and alter arbitrary digital assets
- Interact with other contracts to create robust, interoperable applications
- Base layer for the Internet of Value

# How Smart Contracts Work

**Define**

*Terms and conditions are agreed by all parties involved*

**Trigger**

*Execution of the contract is triggered by an event*

**Execute**

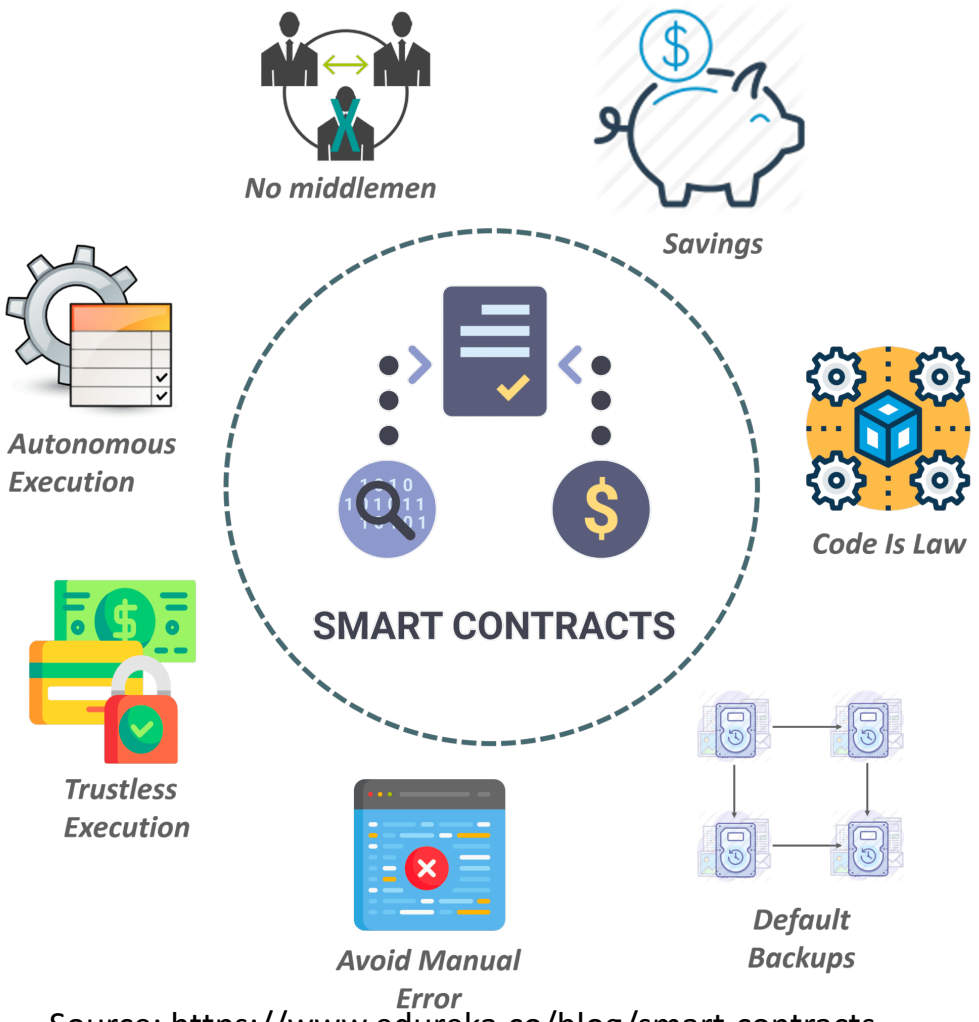
*The smart contract is executed automatically*

**Settle**

*Transaction results are settled on the blockchain*

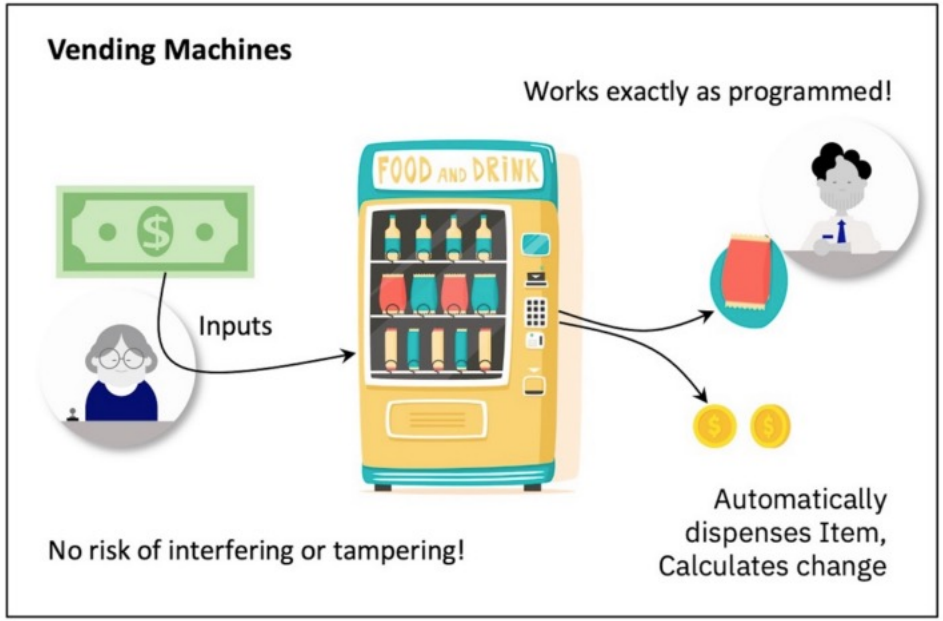
Rensselaer

# What are some advantages of smart contracts?




Source: https://www.edureka.co/blog/smart-contracts

# Vending Machines as a Smart Contract

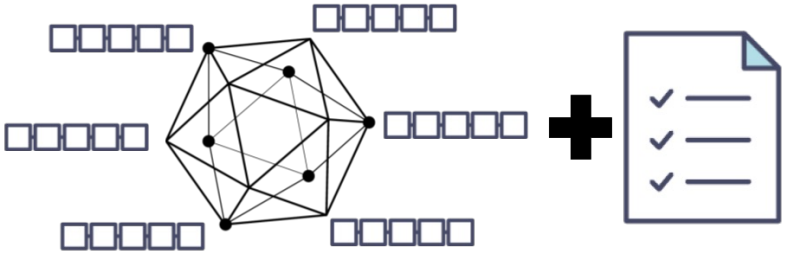



- Buyer selects an item on the screen and agrees to the specified payment
- Buyer inputs cash into the vending machine
- The machine recognizes the payment, confirms its validity, and drops the buyer what they picked from the machine.

# History of Ethereum

- Russian-Canadian Vitalik Buterin co-founded Ethereum when he was 19 years old
- Whitepaper in 2013
  - 'A Next-Generation Smart Contract and Decentralized Application Platform' Ethereum
- Genesis block July 2015
- Important Concepts
  - Blockchain
    - Accounts
    - Wallets
    - Transactions
  - Smart Contracts
  - Tokens
  - Decentralized Applications


Source: https://en.wikipedia.org/wiki/Vitalik_Buterin



Blockchain / Distributed ledger

Smart Contract

# Ethereum Blockchain

- Blockchain as a Fully “Distributed Database”
  - Stores data
  - Transactions/messages alter the data Ethereum
- The “data” can be any digital asset/token
- Ethereum uses **smart contracts** to dramatically expand transaction capabilities
- What are smart contracts?
  - “A set of promises, specified in digital form, including protocols within which the parties perform on these promises.” Nick Szabo, 1996

However ….

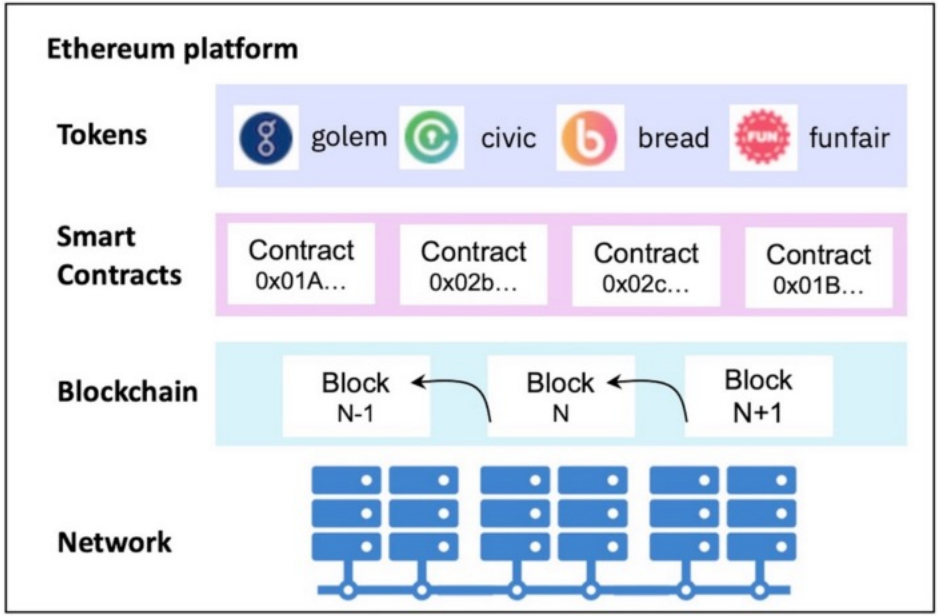
- Smart Contracts may not be ‘Smart’
- Smart Contracts may not be ‘Contracts’

# Decentralized Applications (DApps)

- Goal is totally distributed application
  - No point of failure
  - No censorship
  - Totally transparent
- App logic via smart contract
- App data via decentralized storage like Inter-Planetary File System (IPFS) or Swarm
- Name resolution via Ethereum Name Service (ENS)
- Messaging via Whisper (decentralized SMS - or message calls between applications)
- Backend (or legacy application) integration via Web3

# Programming Dapps in Ethereum

- Using a special programming language called Solidity
- It uses a syntax that resembles JavaScript




Ethereum platform architecture

# Ethereum Under the Hood

- Blocks created faster than BTC and reward is different
  - Every 15 seconds
  - ~ 2 ETH main reward
  - Different mining algorithm, i.e., Keccak 256
  - The same ECDSA used to generate public keys
- Blocks keep track of balances – not UTXO like BTC
- Transitioned from Proof of Work to Proof of Stake on Sep 15, 2022
  - https://ethereum.org/en/upgrades/merge

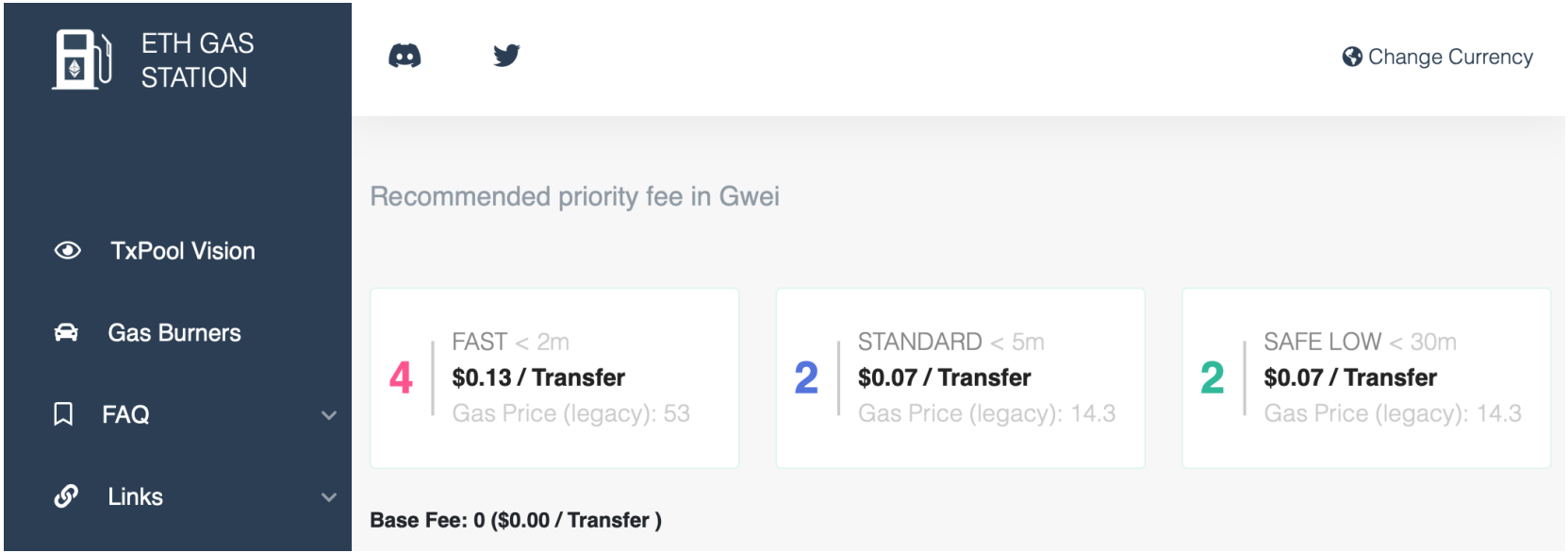
# Gas

- Halting problem:
  - Cannot tell whether or not a program will run infinitely from compiled code (infinite loop)
  - Solution: charge fee per computational step to limit infinite loops and stop flawed code from executing
- Every transaction needs to specify an estimate of the amount of gas it will spend
- Essentially a measure of how much one is willing to spend on a transaction, even if buggy

# Gas Cost

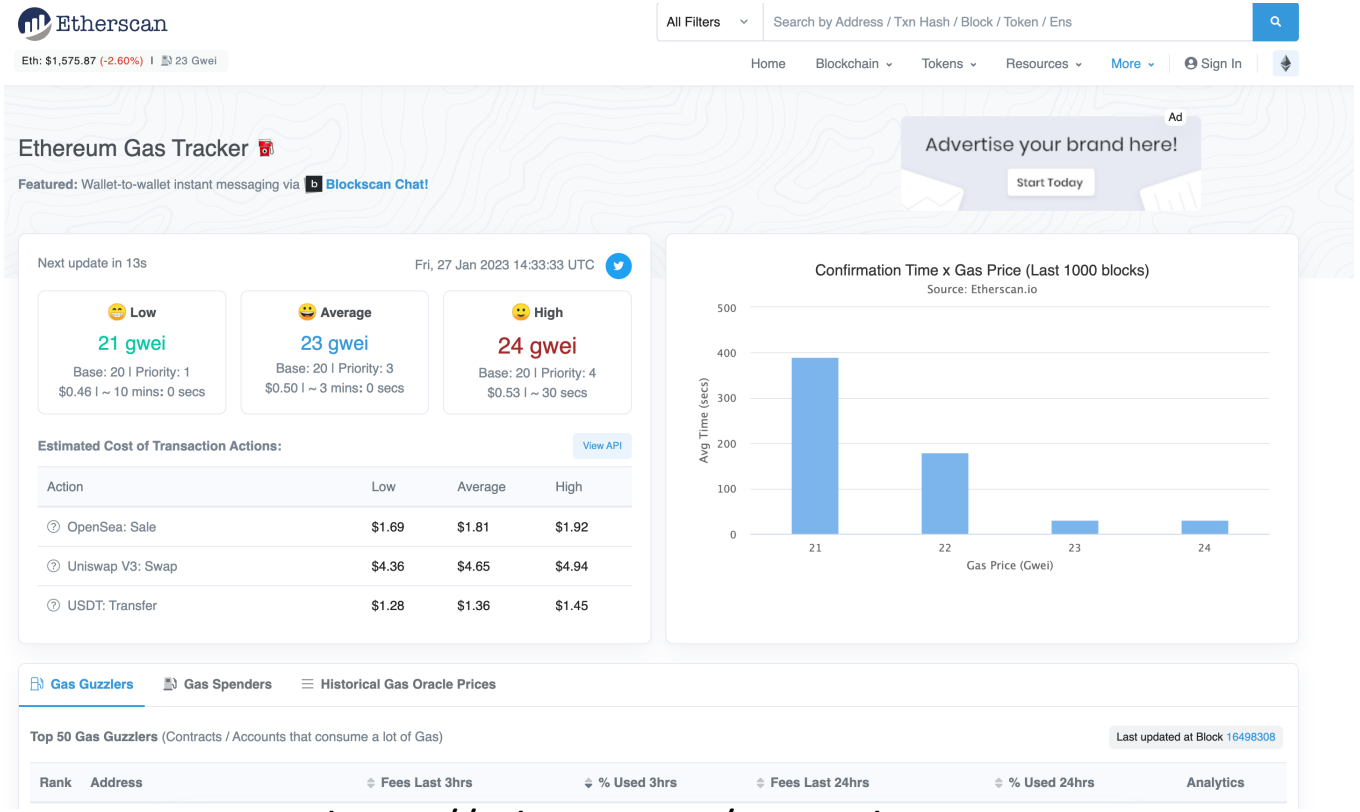
- Helps to regulate load on network
- Gas Price
  - Current market price of a unit of Gas (in Wei)
  - Check gas price here: https://ethgasstation.info
  - Gas price is always set before a transaction by user
- Gas Limit
  - The maximum amount of Gas user is willing to spend
  - All blocks have a Gas Limit (maximum Gas each block can use)
- Gas Cost
  - Used when sending transactions
  - Calculated by gasLimit*gasPrice.

# Each transaction on the public Ethereum network has to pay a gas fee




More complicated transactions consume more gas, so they cost more.
See ”Gas Burners” for such transactions.

# Ethereum Gas Tracker



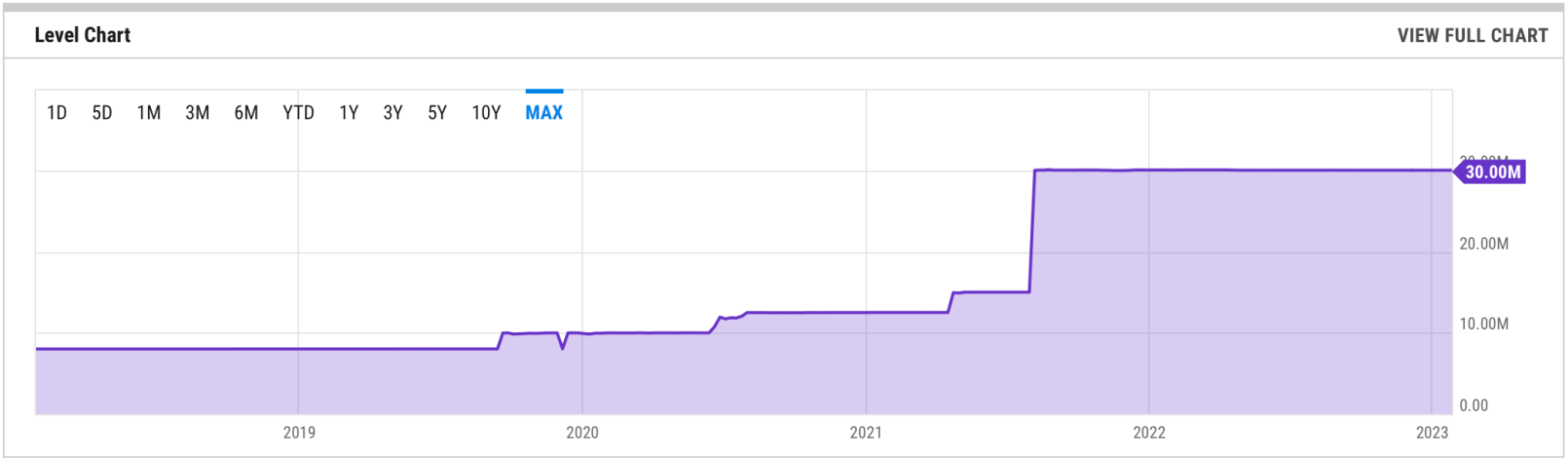

https://etherscan.io/gastracker

# Miners limited by a global limit on gas per block

**Ethereum Average Gas Limit**

30.00M for Jan 26 2023

Overview | Interactive Chart




There's a limit to **how much gas can be consumed in each block**, i.e., a limit on **how many smart contract program statements can be evaluated** on each block. It has been increasing, but at any given time, there's a limit– currently 30 MWei

# Ethereum Lingo

**Ether Denominations**

- Wei - lowest denomination
  - Named after Wei Dai - author of b-money paper (http://www.weidai.com/bmoney.txt, 1998), many core concepts used in BTC implementation
  - 1/1,000,000,000,000,000,000 (quintillion)
- Szabo
  - Named after Nick Szabo - author of Bit-Gold and the person who coined the phrase “smart contracts”
- Finney
  - Named after Hal Finney - received first Tx from Nakamoto

| Unit | Wei Value | Wei |
|---|---|---|
| wei | 1 wei | 1 |
| Kwei (babbage) | 1e3 wei | 1,000 |
| Mwei (lovelace) | 1e6 wei | 1,000,000 |
| Gwei (shannon) | 1e9 wei | 1,000,000,000 |
| microether (szabo) | 1e12 wei | 1,000,000,000,000 |
| milliether (finney) | 1e15 wei | 1,000,000,000,000,000 |
| ether | 1e18 wei | 1,000,000,000,000,000,000 |

Do you recognize names behind some of the other denominations?

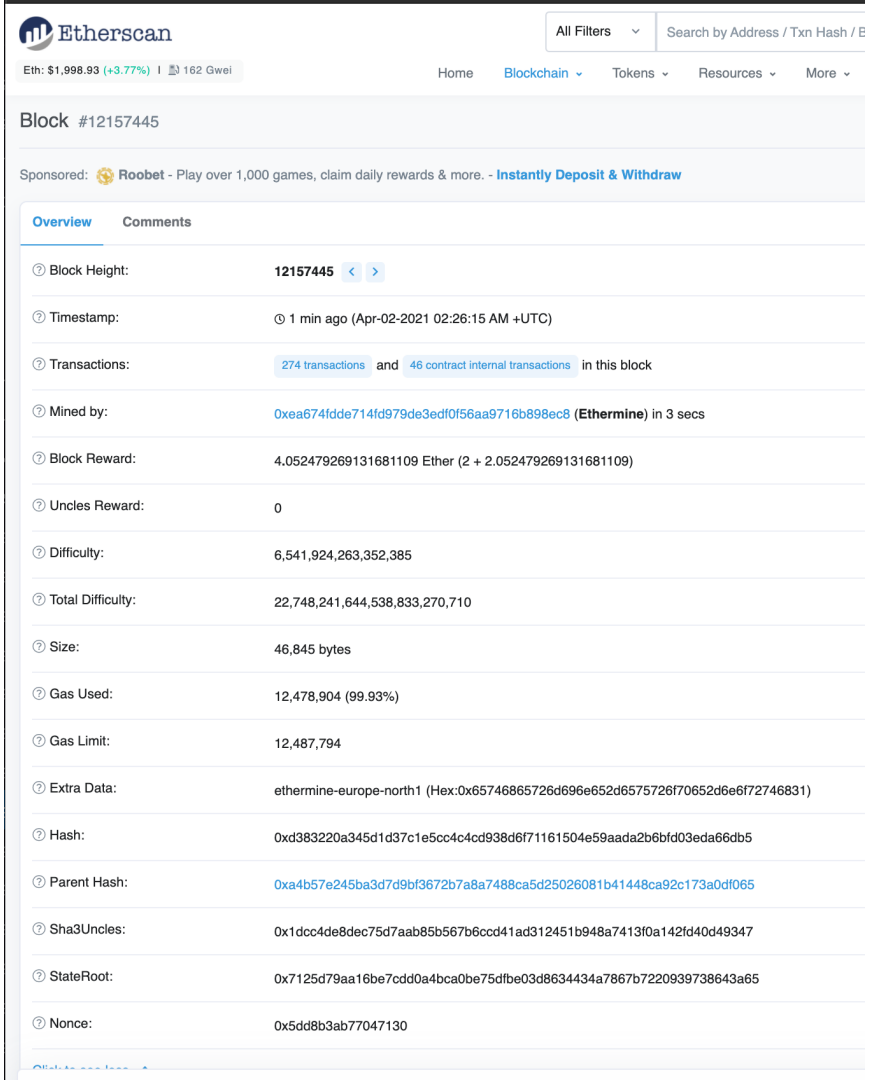
# Ethereum Accounts

- All accounts have equal access to interacting with Ethereum
- External Owned Accounts (EOA)
  - Human account
  - Public/private keys used to send/validate transactions
- Contract Accounts
  - Completely run by code once deployed
  - Can hold and transfer ETH or other tokens
  - Unchangeable outside of what is coded

Rensselaer

# Etherscan

- All blocks visible like BTC
- However, blocks have a different structure than BTC
- https://etherscan.io

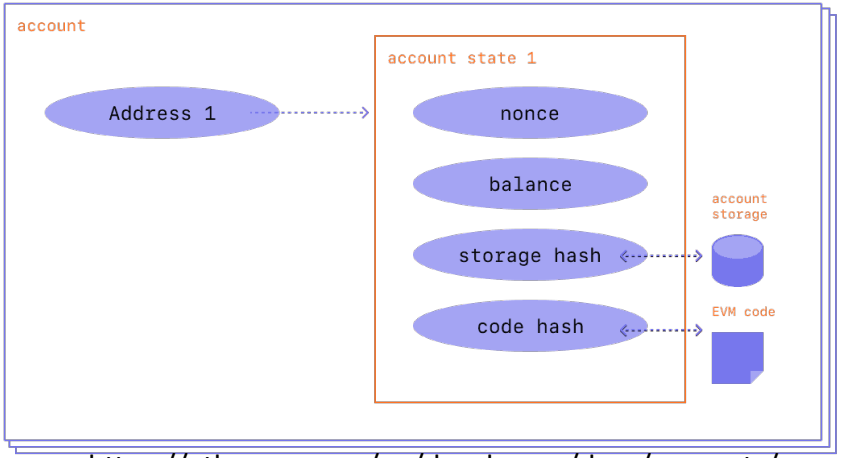


# Wallets

- A set of one or more external accounts
- Used to store/transfer ether
  - Can also hold other tokens
- Manages Public/Private keys for you
  - Usually opened with a password
  - Provides back up phrase for keys
- X of Y Multisig wallets (e.g., need 2 of 3 to sign off on a transaction)

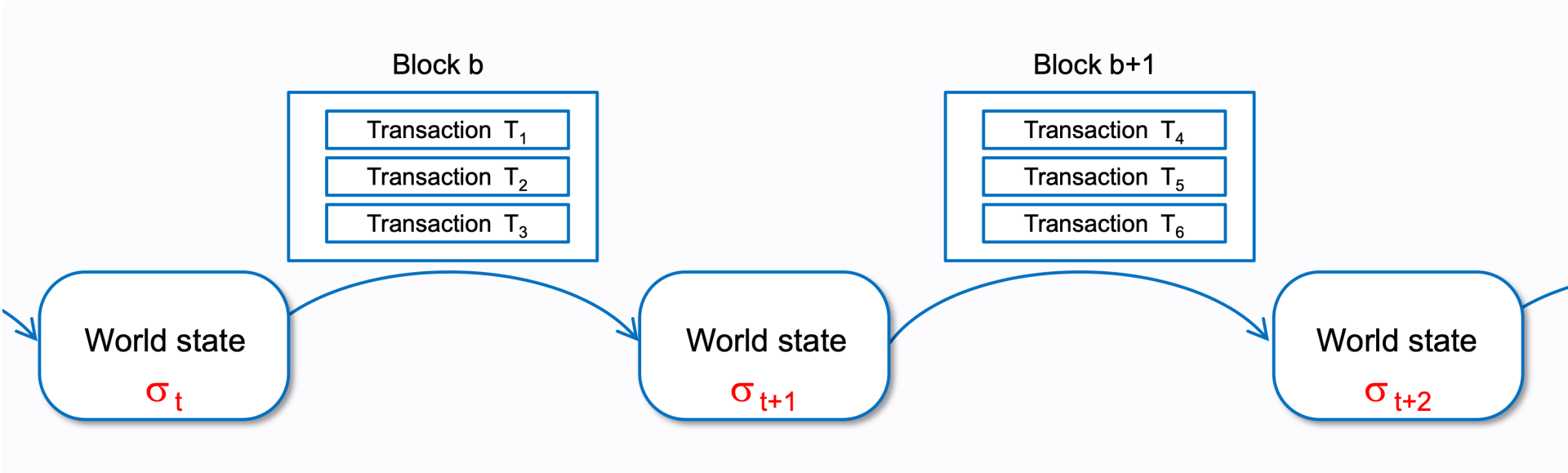
# Ethereum Accounts

- **Externally-owned account** (EOA) – controlled by anyone with the private keys
- **Contract account** – a smart contract deployed to the network, controlled by code.
- Both account types have the ability to:
  - Receive, hold and send ETH and tokens
  - Interact with deployed smart contracts

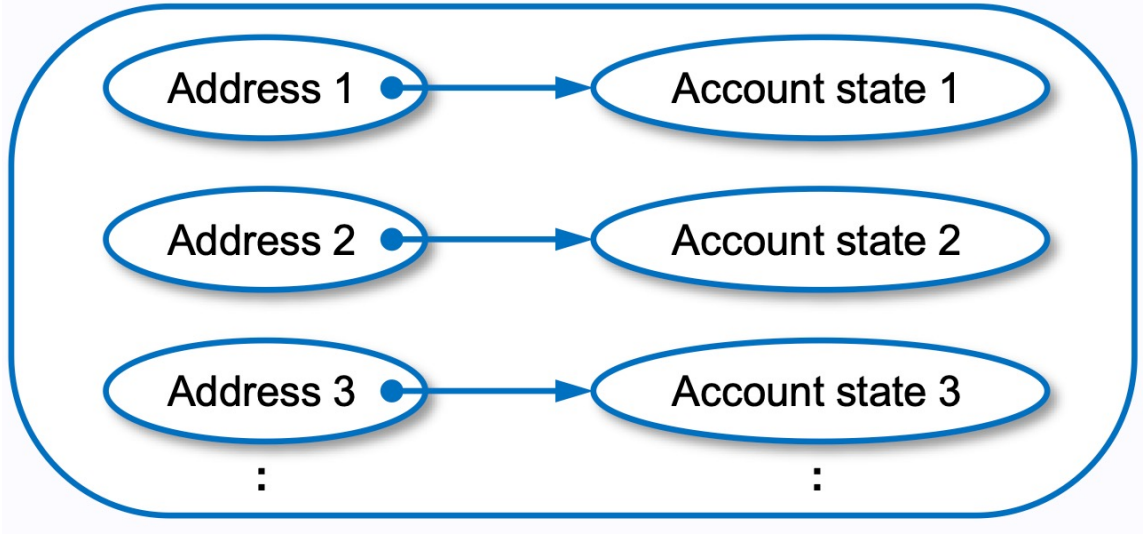



Source: https://ethereum.org/en/developers/docs/accounts/

# Ethereum can be seen as a “chain of states”



# “World State”




The world state is a mapping between address and account state.

# Tokens

- Digital assets which live on a blockchain not its own
- Can have utility in context of a DApp, represent a physical good, or be a digital collectible
- ERC-20: Fungible Ethereum Token spec
  - All tokens are interchangeable, i.e., non-unique (like money)
  - Divisible
  - Examples: Binance, Tether, Uniswap, Chainlink, USDC
- ERC-721: Non-Fungible Ethereum Token spec
  - Each token unique (like collectibles or title deeds)
  - Examples: Cryptokitties, Ethereum Name Service (ENS)
- Any idea where these numbers (20, 721) come from?
- The number 20 simply refers to the $20^{th}$ ERC that was posted by someone. That person proposed a general interface for a fungible token.

# What is ERC?

- Stands for “Ethereum Request for Comments”
- Open and public mechanism inspired by the well-known IETF Request for Comments (RFC)
- The mechanism for standardization of tokens
  - So that one token can be traded for another in the Ethereum ecosystem
- ERC's are now called EIPs: Ethereum Improvement Proposals
  - Because the majority of newcomers did not understand any difference between EIPs and ERCs they were merged.

# An ERC-20 Token Example

Use a library such as OpenZepplin's ERC.sol

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol

The constructor takes a name and a symbol.

The visibility specifier for mint defines the function as **internal**, which means only derived contracts can call this.
Sets the totalSupply.
Updates the balances

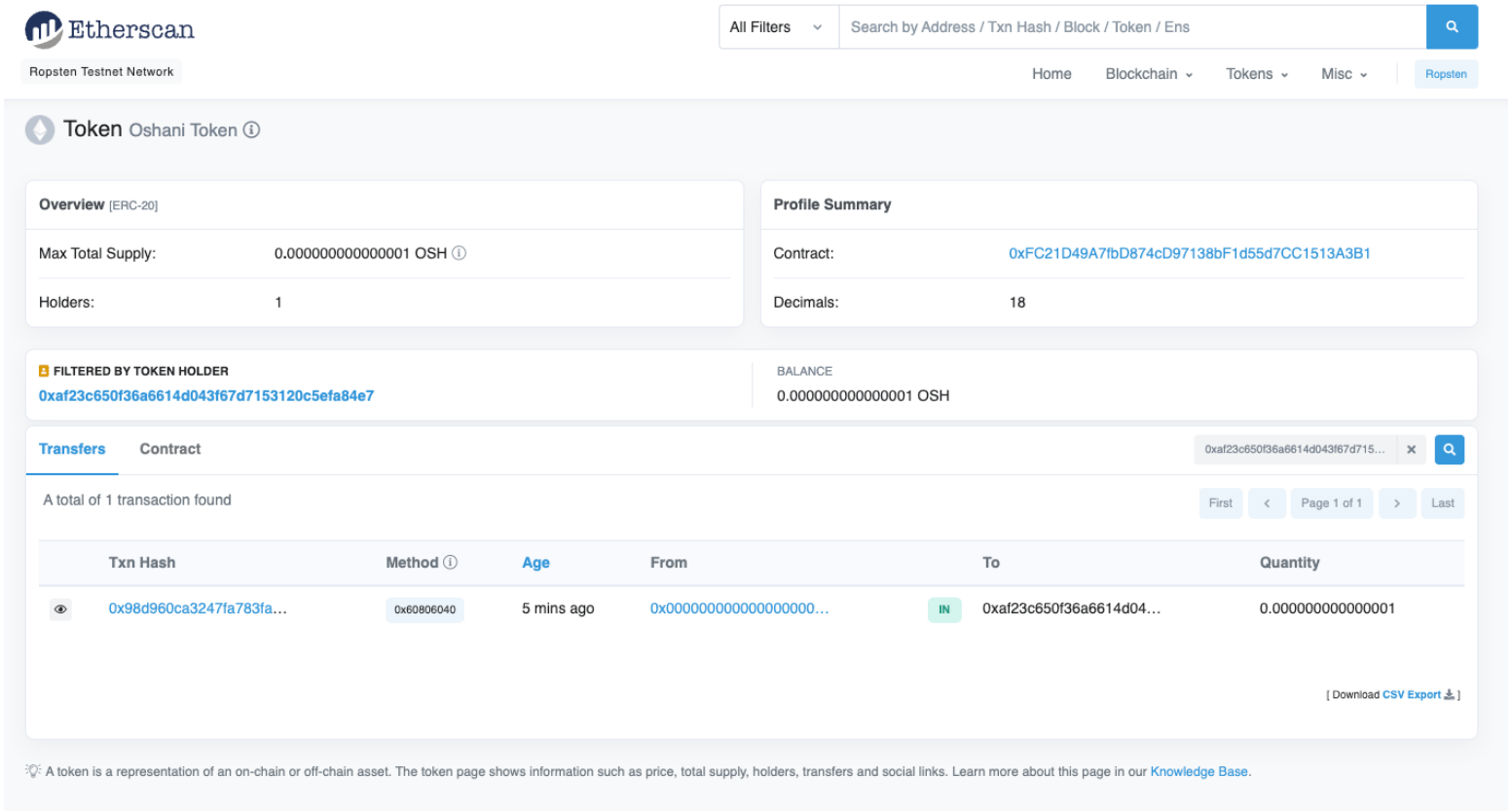
```
1   pragma solidity ^0.8.4;
2
3   // The ERC-20 spec is implemented in ERC20. sol, by importing
4   // it, we avoid duplicating a great deal of code here
5   import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
6
7   // OshaniToken is meant to be a very simple example of an ERC20 token.
8   // In the example all tokens are pre-assigned to the creator.
9   // Can later distribute these tokens using `transfer` and other ERC20 functions
10  contract OshaniToken is ERC20 {
11
12      // "super constructor" for OshaniToken, which calls on ERC20 contructor
13      // passing in token name   = "Oshani Token" and symbol = "OSH";
14
15      constructor() ERC20("Oshani Token", "OSH") {
16
17          // Mints 1,000 tokens to your wallet's address
18          _mint(msg.sender, 1000);
19      }
20      // so much more can be done here.
21  }
```

# Minting Tokens

- Fixed Supply Tokens
  - The mint function is callable only in the constructor once.
  - Once the token is deployed, there is no more access to the internal mint functionality, the supply of tokens remains fixed.

- Variable Supply Tokens
  - Possible to mint more tokens after the contract is deployed.

# Token is Minted!



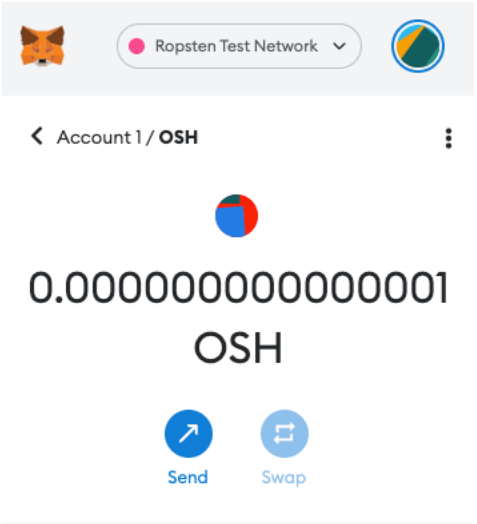

https://ropsten.etherscan.io/token/0xFC21D49A7fbD874cD97138bF1d55d7CC1513A3B1?a=0xaf23c650f36a6614d043f67d7153120c5efa84e7

Rensselaer

# Import into Your Wallet

You can specify the contract address and import the tokens to Metamask.

Now ready for transactions!



# Token Contracts

- A token can be created (minted) by a smart contract
- The minting process follows a set of rules specified in ERC-X (or EIP-X) standard, that dictates what it means to create, transfer, and keep track of account balances
- To purchase tokens, a buyer sends Ether to the smart contract affiliated with that token
- There are many online marketplaces and exchanges to buy and sell tokens
  - For e.g., opensea.io is for the exchange of NFTs

Rensselaer

# Non Fungible Tokens (NFT)

**The first big NFT!**


https://www.cryptokitties.co

Rensselaer

# Cryptokitties are Based on Dutch Auctions

The “Buy it Now” price is initially set at the largest value.
As time goes on, the “Buy it Now” price is lowered
As soon as someone is ready to buy it, they announce their bid and win.


Example: https://solidity-by-example.org/app/dutch-auction

# What if I want to create an NFT?

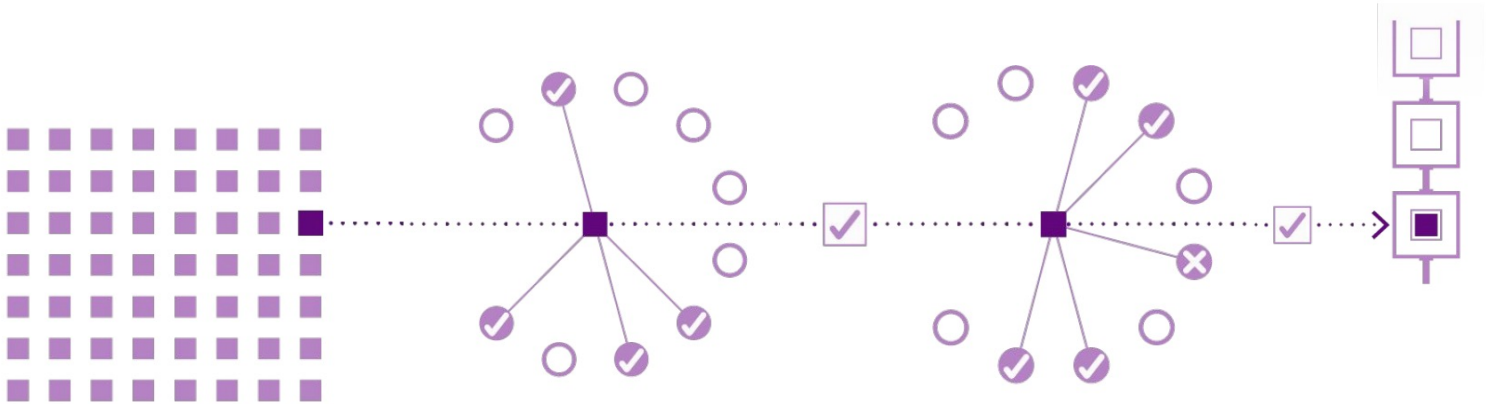
- ERC-721 further extends the ERC-20 token specification by enabling the definition of unique, non-fungible tokens
- The primary difference is EC-721's addition of the following:
  - Unique token identification number (tokenID)
  - External (off-chain) link that references a collection of data (metadata) that represents the unique properties of this token (tokenURI)
- Several token builder tools allow for web-based creation of ERC-721 tokens without coding, e.g., opensea.io
  - The NFT is given a **name and description** with a means to set the **offering price** of the NFT along **with options of blockchain platforms** in which it will be deployed and run.
  - For example, if Ethereum is its destination, it will **auto-generate the Solidity smart contract** and compile and deploy it with a simple push of a button.
  - Then the NFT appears as a web page and enables you to "mint" a new token or transfer ownership to another user's address.

# Algorand

# Algorand's Founder

- Silvio Micali
  - Professor MIT
  - Turing Award, Gödel Prize
  - Scientific Contributions:
    - Digital Signatures
    - Probabilistic Encryption
    - Zero-Knowledge Proofs
    - Verifiable Random Functions
    - Many other primitives of modern cryptography…
- The consensus mechanism used in Algorand is his brainchild:
  - Sortition: select a random small constant-size committee that proposes blocks and votes on blocks
  - Scales with millions of participation nodes!

# Pure Proof of Stake



**Block proposal:** Accounts propose new blocks to the network

**Soft vote:** Committee votes on proposals and filters down to one

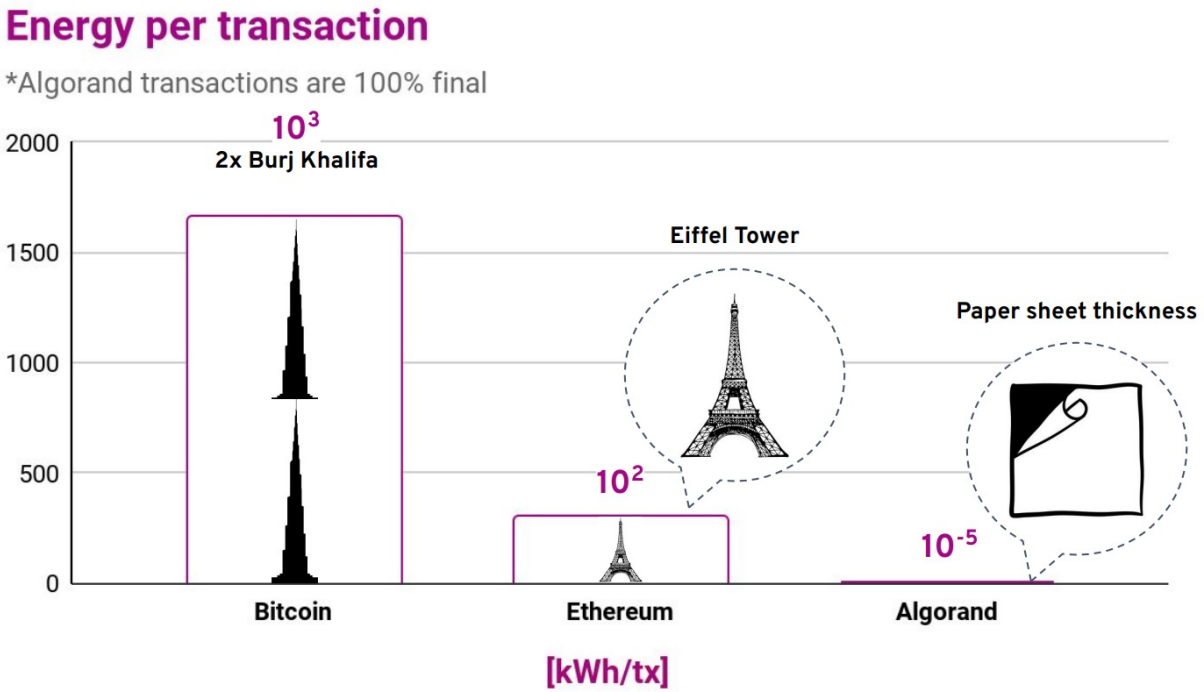
**Certify vote:** A separate committee votes to certify the block

The new block is appended to the blockchain

# Why Algorand?

- Block time < 4 seconds
  - Ethereum = 12 seconds
  - Bitcoin = 10 minutes
- Immediate finality, i.e., never forks!
  - Once a block is added, it can never be removed
  - Compare with Ethereum where Coinbase waits 14 blocks, i.e., about 3 minutes
- High throughput: 6000 transactions per second
  - Compare with Ethereum, which is < 30 transactions per second
- Easy to develop
  - No need to develop smart contracts in many cases (ASA, NFT, atomic transfers)
  - Smart contract languages use python
  - Official SDK for python, Javascript, Java and Go
  - Community SDK for dot Net

# Algorand Sustainability




"Permission-less" is not "Responsibility-less"

Source: https://prismic-io.s3.amazonaws.com/algorandfoundationv2/d5407f96-8e7d-4465-9656-2abb558850a9_Proof+of+Stake+Blockchain+Efficiency+Framework.pdf

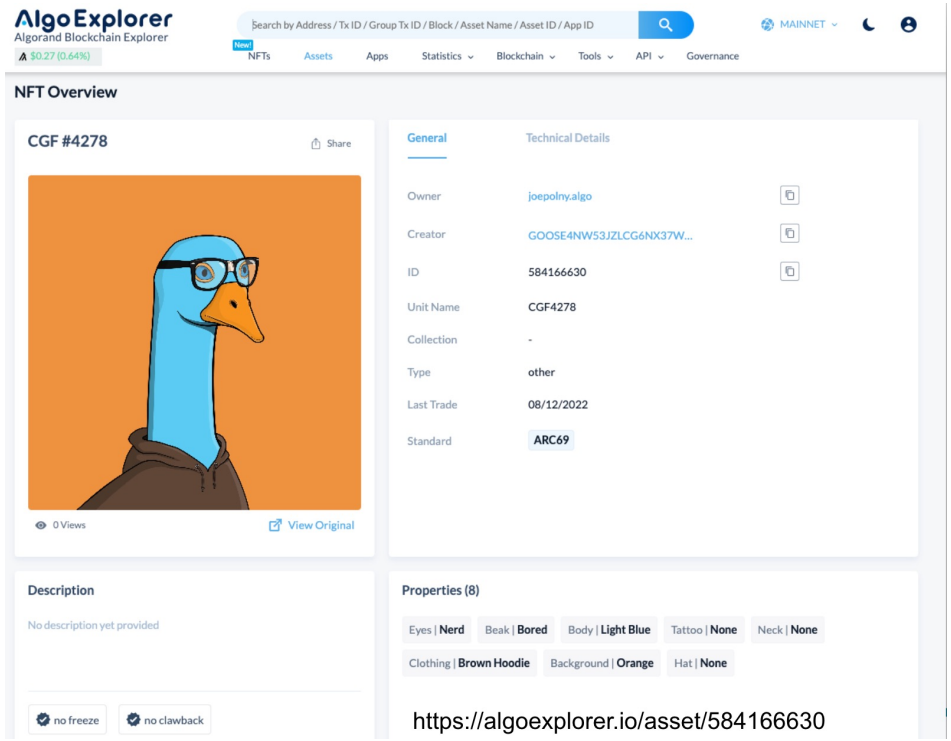
# Algorand Standard Assets (ASA)

- Algo = native token
- ASA = custom token
  - Anyone can create their own ASA
  - Same transaction fee as the Algo
  - Same throughput/latency
  - Examples: reward/loyalty token, USDC, NFT ..
- Create your own token in a few seconds at:
  - https://app.algodesk.io

- Comparison with Ethereum
  - Similar to ERC-20/ ERC-721
  - Lower transaction fee
  - No smart contract

Rensselaer

# NFT = ASA with supply of just one!




Developer portal contains everything you need!

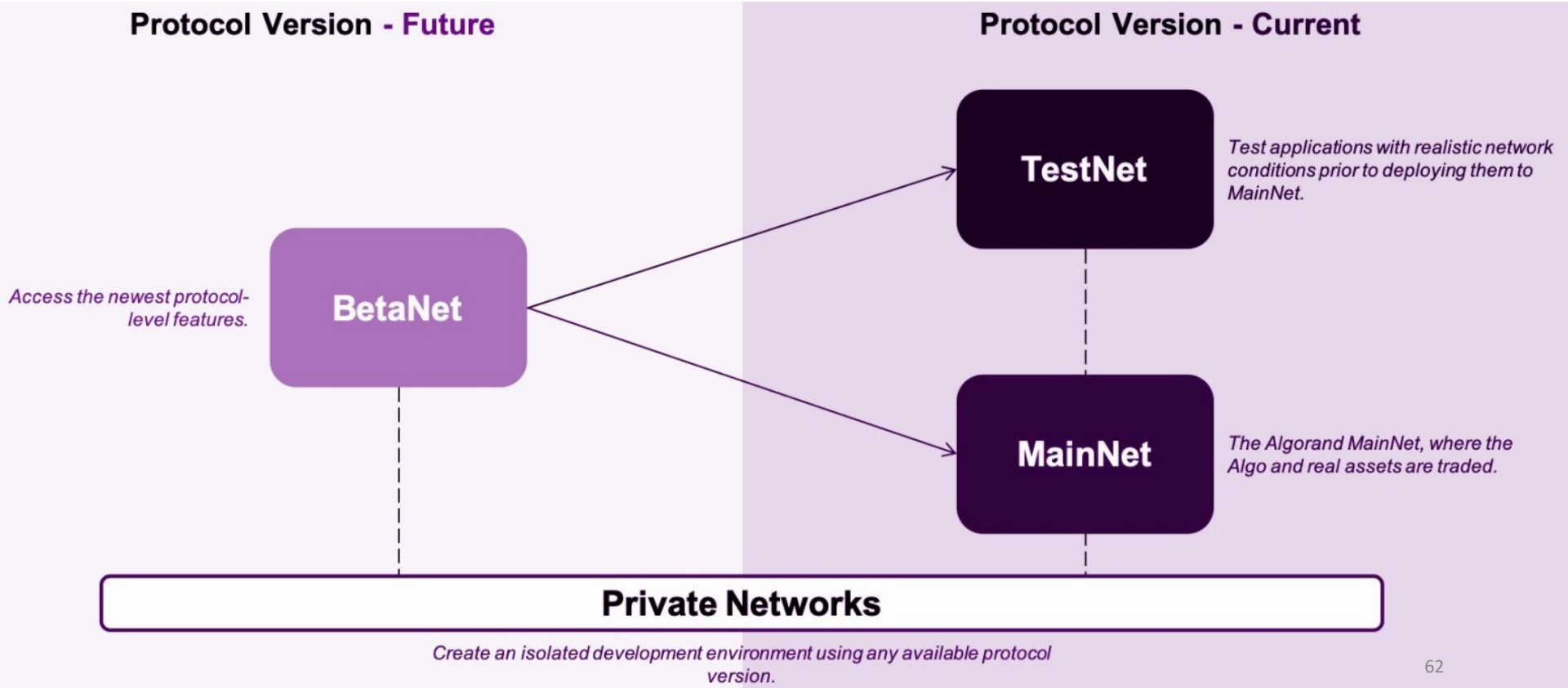
https://developer.algorand.org

Websites to create NFTs just in 1-click

https://arcminter.daotools.org

Https://arc3.xyz

# Algorand Networks



# Algorand Nodes

- Non-Relay Nodes
  - Participation Nodes:
    - Participate in the PPoS consensus (verify the blocks and the transactions to ensure the network is safe)
    - Light Configuration: store just the latest 1000 blocks (~20 GB)
    - Recommended Specs: 8vCPU, 16GB RAM, 500 GB, 1GBPS broadband
  - Archival Nodes:
    - Store all the chain since the genesis block (~1TB)
    - Required for indexer, which is used to query the blockchain
- Relay Nodes
  - Communication routing to a set of connected Non-Relay Nodes
  - Connect both with Non-Relay Nodes and Relay Nodes
  - Route blocks to all connected Non-Relay Nodes
  - Highly efficient communication paths, reducing communication hops
  - Recommended Specs: 16 vCPU, 32 GB RAM, 3GB SSD, 30 TB/month egress, 1 GBPS broadband

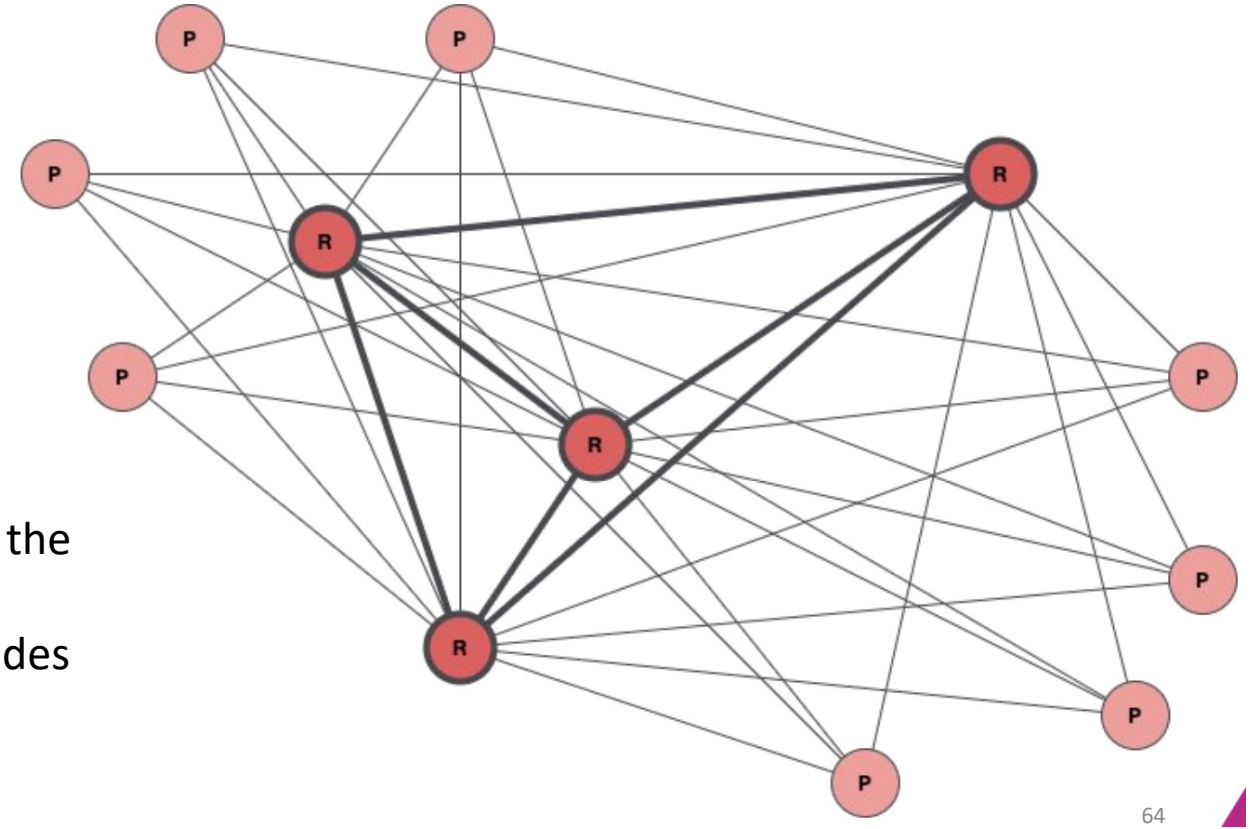
# Algorand Network Topology

**Participation Nodes:**

- ~200 unique accounts participating
- ~1.5B ALGOs online
- Permissionless

**Relay Nodes:**

- ~120 nodes
- Default relays chosen by the Algorand foundation
- Anyone can relay, but nodes must point to it



# Algorand Nodes

- Running a node:
  - Install the Algod (Algorand Deamon)
  - Choose a network (MainNet, TestNet, BetaNet, PrivateNet)
  - Start & Sync with the network
- Interacting with nodes:
  - CLI utilities: goal, kmd and algokey
  - REST API interface: algod V2, kmd, indexer
  - Algorand SDKs: JavaScript, Python, Java, Go

# Software

- algod – Algorand Daemon
  - The node software that connects with the rest of the network
  - HTTP endpoints for submitting transactions and reading state
- kmd – Key Management Daemon
  - Responsible for the wallet management
  - Manages account keys
  - HTTP endpoints for managing and querying local accounts
- indexer
  - Software that can run alongside an archival node
  - Saves blockchain state in SQL database
  - Provides HTTP endpoints specifically for querying on-chain data (for e.g., to query the balance of a particular account

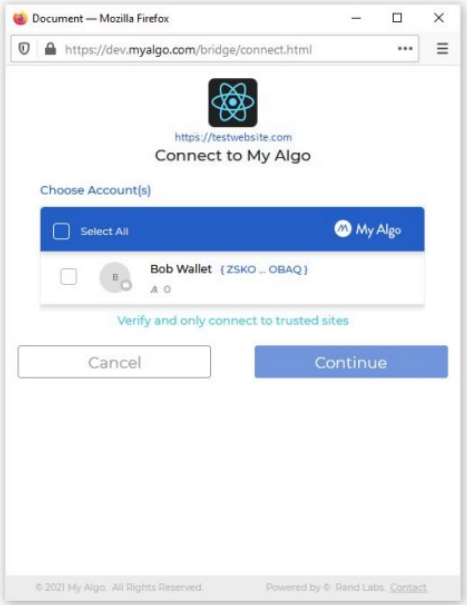
# Interacting with Nodes

- **goal**
  - Command-line utility for interacting with algod and kmd programmatically
- **SDKs**
  - Leverage HTTP endpoints to interact with algod and kmd
  - Essentially, programmable wrappers
  - Python SDK, Javascript SDK, Java SDK
- **Public API services**
  - Services that expose HTTP endpoints for Algorand nodes publicly
  - Useful when you don’t want to run your own node

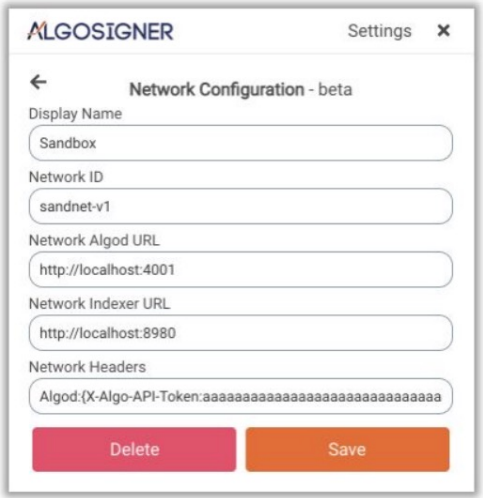
Rensselaer

# Algorand Wallets


Mobile Wallet + Wallet Connect




MyAlgo Wallet



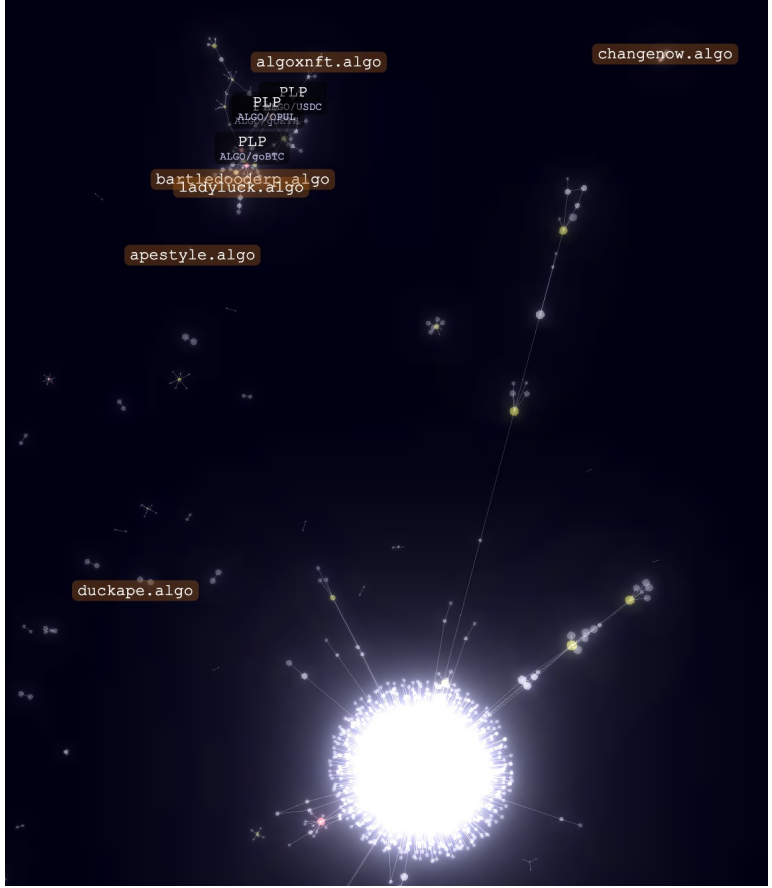

Algosigner

pera

Pera Wallet

# Algorand Explorers

- AlgoExplorer
- Goalseeker
- NFTExplorer
- Algorand Ballet - Algorand accounts' 2D graphs.
- Algorand Multiverse - Algorand accounts' 3D graphs.
- Algoscan - Algoscan is a Blockchain Explorer and Analytics Platform. Built on top of the Algorand Network.
- Asalytic - Analyze the Algorand NFT space.
- Dappflow - Algorand Private Network Explorer (supports Sandbox in localhost).




Algorand Multiverse at https://algo3d.live

# Algorand Transactions

- Transactions are the core element of blocks, which define the evolution of distributed ledger state.
- There are six transaction types in the Algorand Protocol:
  1. **Payment** (sends Algos from one account to another)
  2. **Key Registration** (register an account to participate in Algorand Consensus).
  3. **Asset Configuration** (create an asset, modify certain parameters of an asset, or destroy an asset)
  4. **Asset Transfer** (receive a specific type of Algorand Standard Asset, transfer an Algorand Standard asset, or revoke an Algorand Standard Asset from a specific account)
  5. **Asset Freeze** (asset receiver address losing or being granted the ability to send or receive the frozen asset)
  6. **Application Call** (Smart contract logic identified by an AppId and an OnComplete method. The AppId specifies which App to call and the OnComplete method is used in the contract to determine what branch of logic to execute.)
- These six transaction types can be specified in particular ways that result in more granular perceived transaction types.

# Example Algorand Transaction


```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwdvd7S12BL5FaOP20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```


Transaction that sends 5 Algos from one account to another on MainNet.

# Example Algorand Transaction


```
{
  "txn": {
    "apar": {
      "am": "gXHjtDdtVpY7IKwJYsJWdCSrnUyRsX4jr3ihzQ2U9CQ=",
      "an": "My New Coin",
      "au": "developer.algorand.org",
      "c": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
      "dc": 2,
      "f": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
      "m": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
      "r": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
      "t": 50000000,
      "un": "MNC"
    },
    "fee": 1000,
    "fv": 6000000,
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 6001000,
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "acfg"
  }
}
```


Asset parameters struct that includes all the initial configurations for the asset

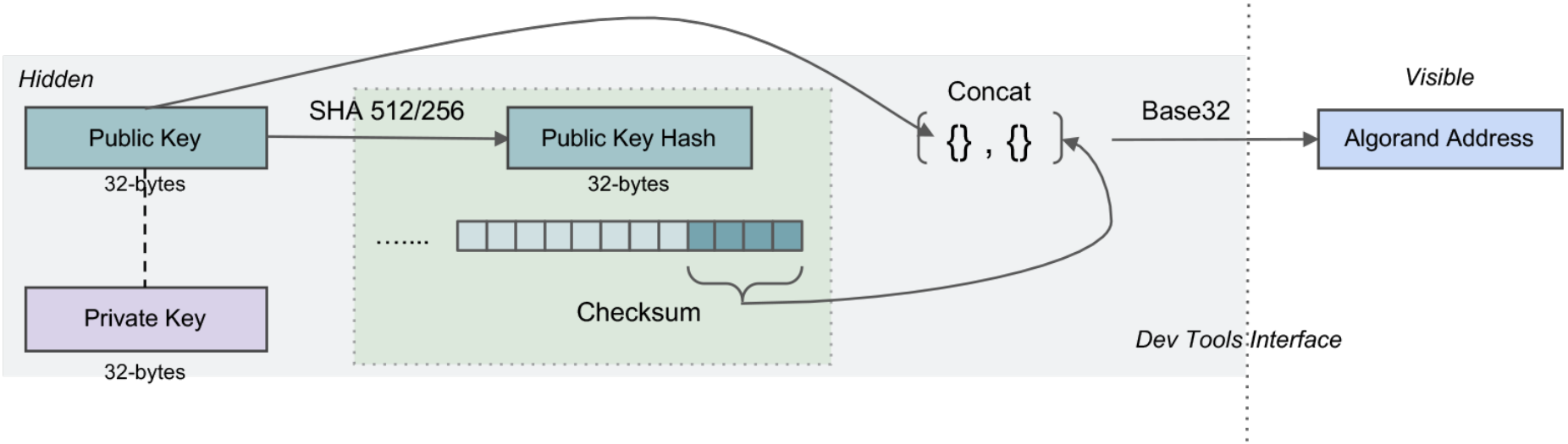
Asset creation transaction

# Multi-Signature Transactions in Algorand


```
{
  "msig": {
    "subsig": [
      {
        "pk": "SYGHTA2DR5DYFWJE6D4T34P4AWGCG7JTNMY4VI6EDUVRMX7NG4KTA2WMDA"
      },
      {
        "pk": "VBDMPQACQCH5M6SBXKQXRWQIL7QSR4FH2UI6EYI4RCJSB2T2ZYF2JDHZ2Q"
      },
      {
        "pk": "W3KONPXCGFNUGXGDCOCQYVD64KZOLUMHZ7BNM2ZBK5FSSARRDEXINLYHPI"
      }
    ],
    "thr": 2,
    "v": 1
  },
  "txn": {
    "amt": 10000000,
    "fee": 1000,
    "fv": 4694301,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 4695301,
    "rcv": "QC7XT7QU7X6IHNRJZBR67RBMKCAPH67PCSX4LYH4QKVSQ7DQZ32PG5HSVQ",
    "snd": "GQ3QPLJL4VKVGQCHPXT5UZTNZIJAGVJPXUHCJLRWQMFRVL4REVW7LJ3FGY",
    "type": "pay"
  }
}
```

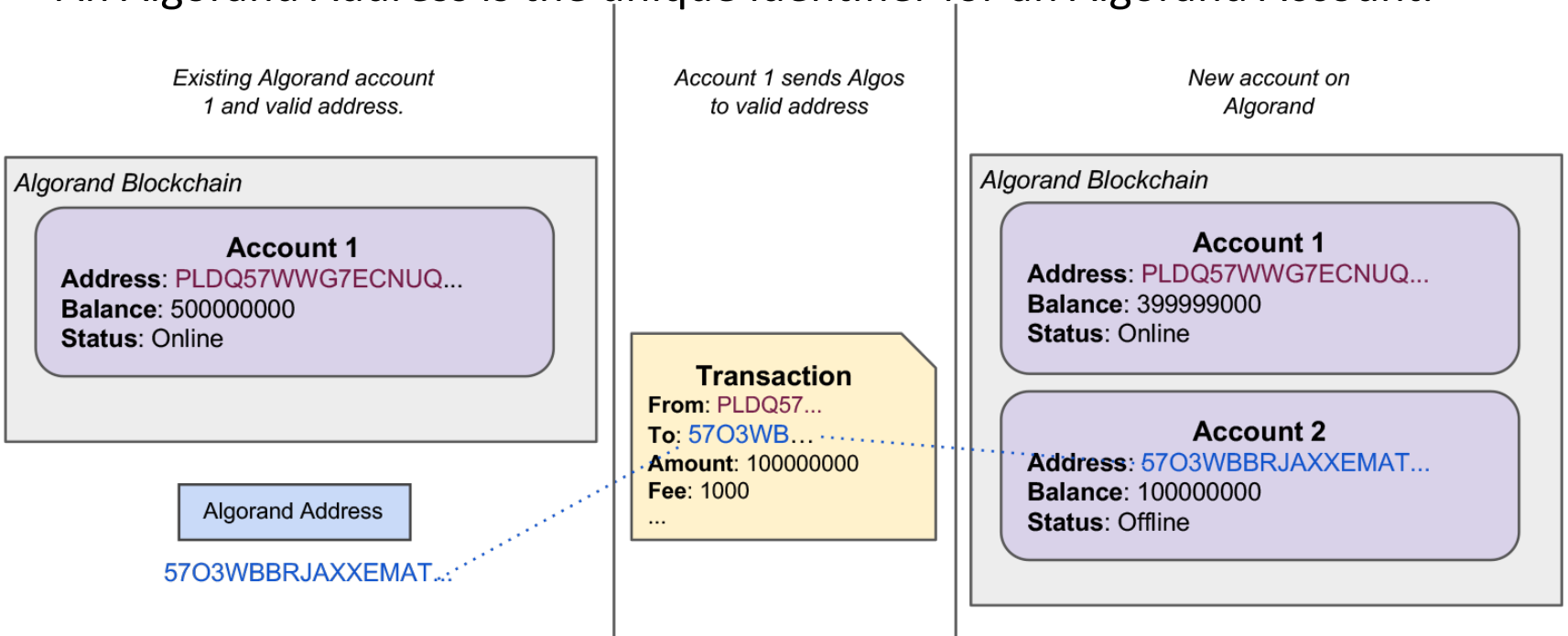
Rensselaer

# Algorand Addresses



# Algorand Accounts

- Accounts are entities on the Algorand blockchain associated with specific on-chain local state.
- An Algorand Address is the unique identifier for an Algorand Account.



# Smart Contracts

- Flat fee (0.001 ALGO) until congestion
- Turing complete language (TEAL)
  - Hard-coded limitations to keep complexity in check
- Can read/write blockchain state and send transactions

# Smart Contract Tech Stack in Algorand

- Algorand Virtual Machine (AVM)
  - Running on every node
  - Not compatible with Ethereum Virtual Machine
- Transaction Execution Approval Language (TEAL)
  - Assembly-like language for writing smart contracts
- PyTeal, Beaker, and AlgoKit
  - Python library and framework for writing Algorand smart contracts
  - Ultimately compiles down to TEAL

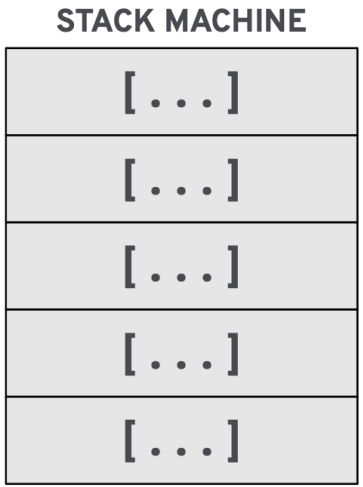
# Algorand Virtual Machine (AVM)

- Available data
  - Transaction information: sender, fee, amount
  - Global variables: current round, latest timestamp
- The Algorand Virtual Machine is a **Turing-complete** secure execution environment that runs on the Algorand consensus layer.
- AVM **approves** transactions' effects if and only if:
  - There is a single non-zero value on top of AVM's stack
- AVM **rejects** transactions' effects if and only if:
  - There is a single zero (false) value on top of AVM's stack
  - There are multiple values on the AVM's stack
  - There is no value on the AVM's stack
- The AVM runs on **every node** in the Algorand blockchain.
- It contains a **stack** that evaluates smart contracts and smart signatures.

# How does the AVM work?

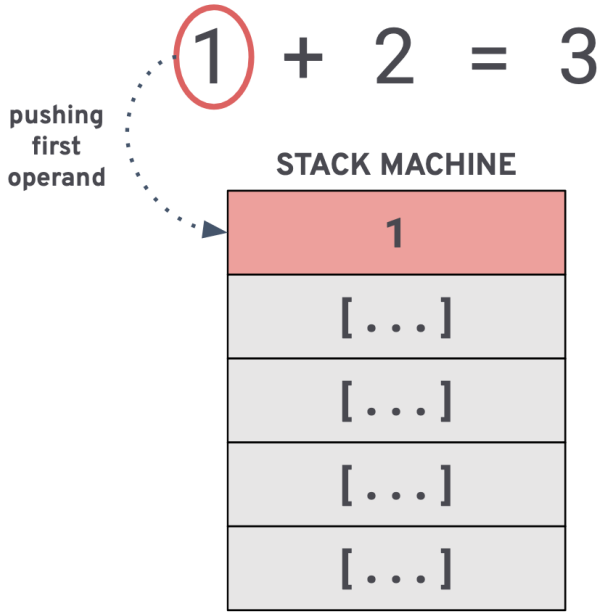
- Suppose we want the AVM to check the following assertion:

$$1 + 2 = 3$$



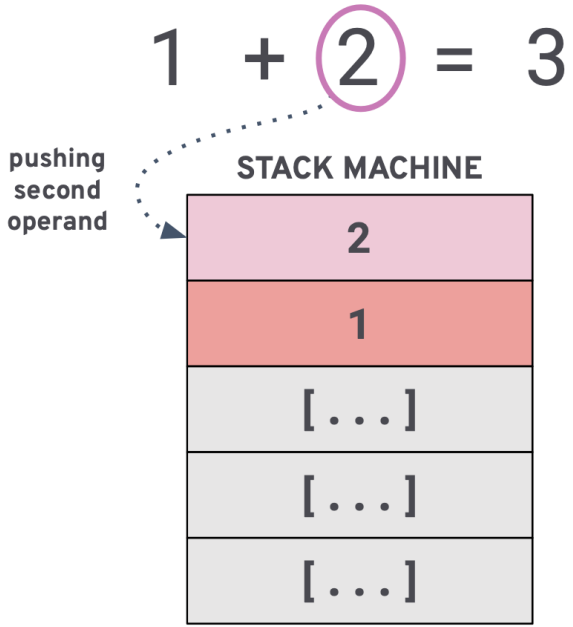
# How does the AVM work?

- Suppose we want the AVM to check the following assertion:



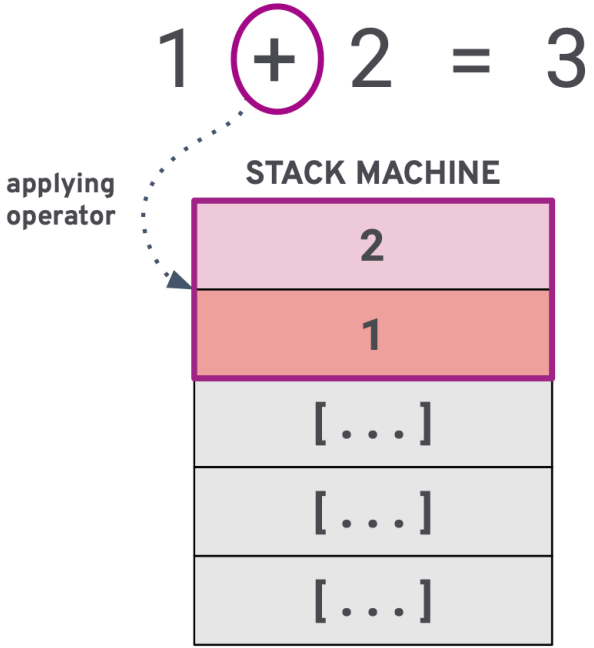
# How does the AVM work?

- Suppose we want the AVM to check the following assertion:



# How does the AVM work?

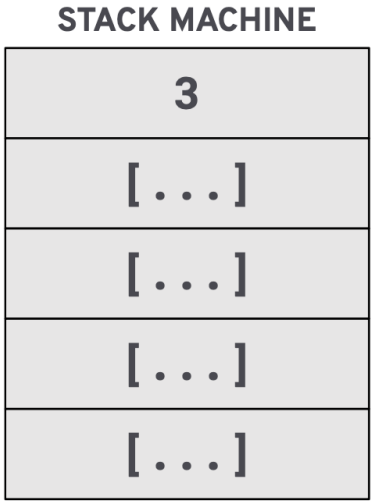
- Suppose we want the AVM to check the following assertion:



# How does the AVM work?

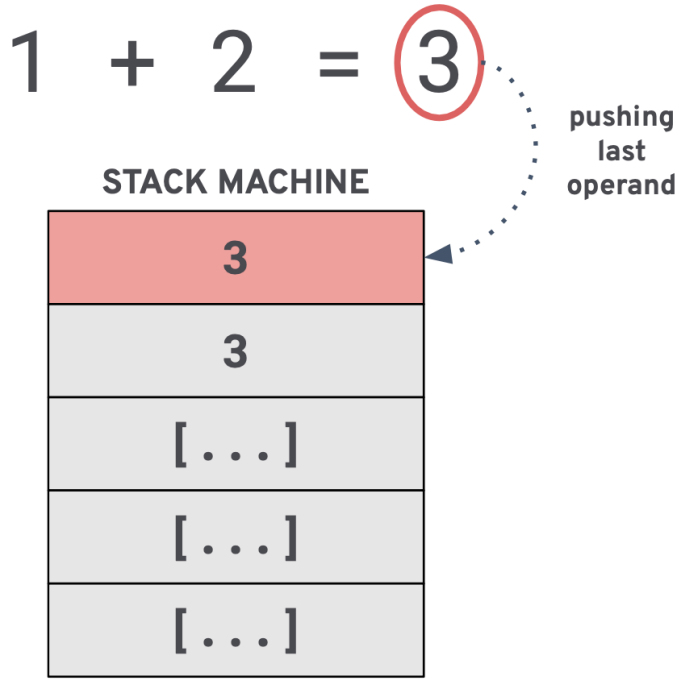
- Suppose we want the AVM to check the following assertion:

$$1 + 2 = 3$$



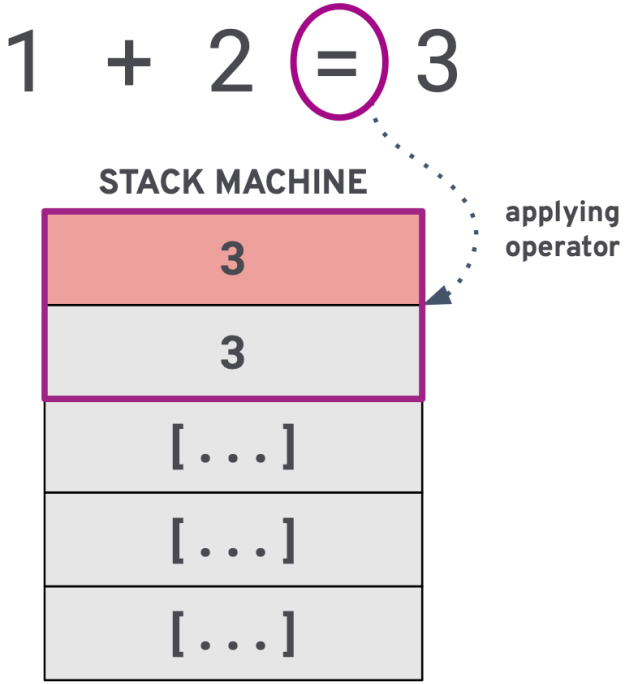
# How does the AVM work?

- Suppose we want the AVM to check the following assertion:



# How does the AVM work?

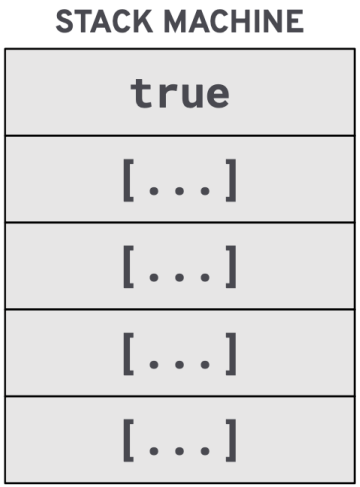
- Suppose we want the AVM to check the following assertion:



# How does the AVM work?

- Suppose we want the AVM to check the following assertion:

$$1 + 2 = 3$$

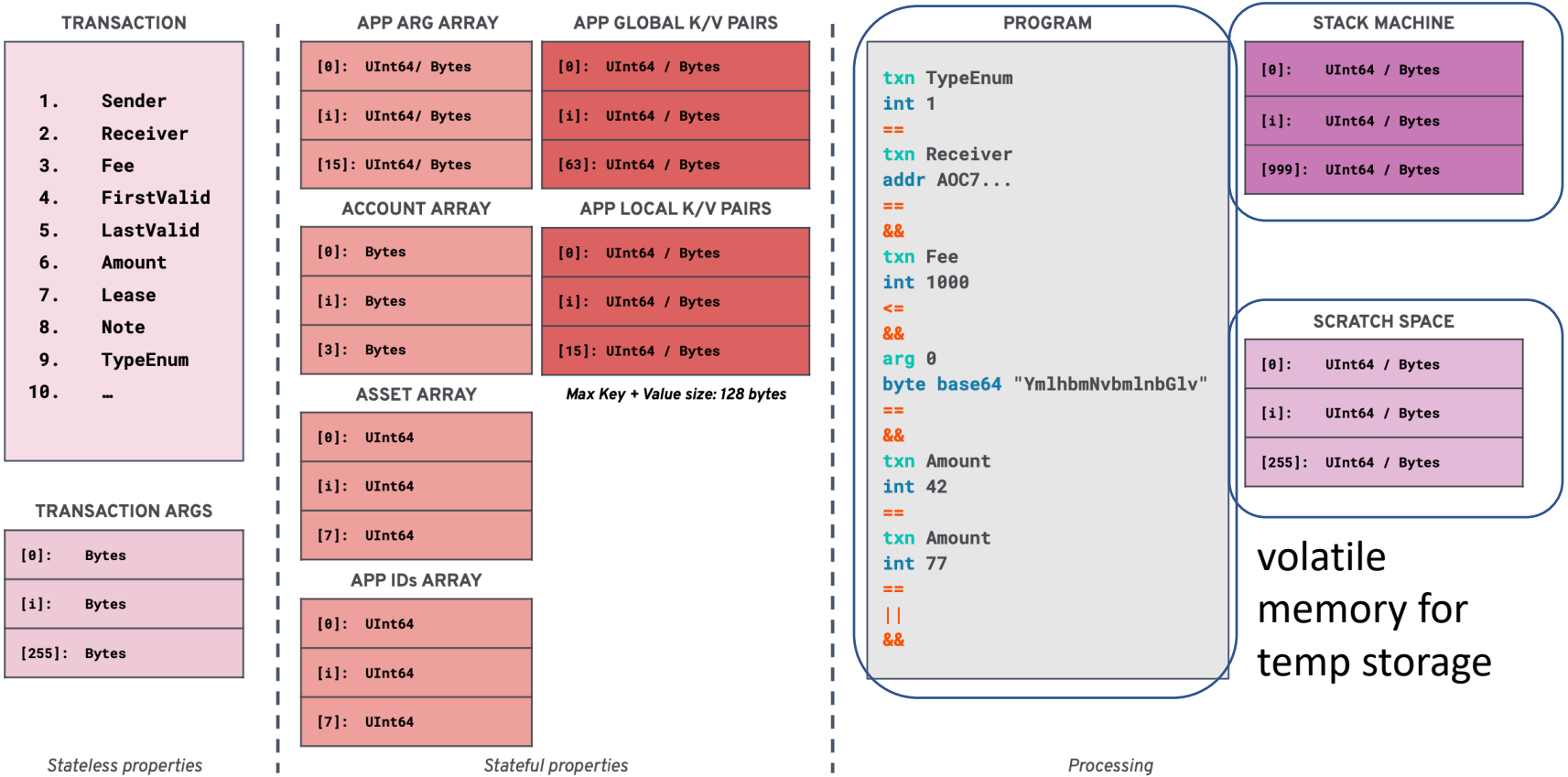


# Transaction Execution Approval Language (TEAL)

- AVM interprets an assembler-like language called TEAL.
- TEAL can be thought of as syntactic sugar for AVM bytecode.
- TEAL programs are comprised of a set of operation codes (opcodes).
- These opcodes are used to implement the logic of smart contracts and smart signatures.
- While it is possible to write TEAL directly, a developer may prefer to use the PyTeal Python library, which provides a more familiar syntax.

Rensselaer

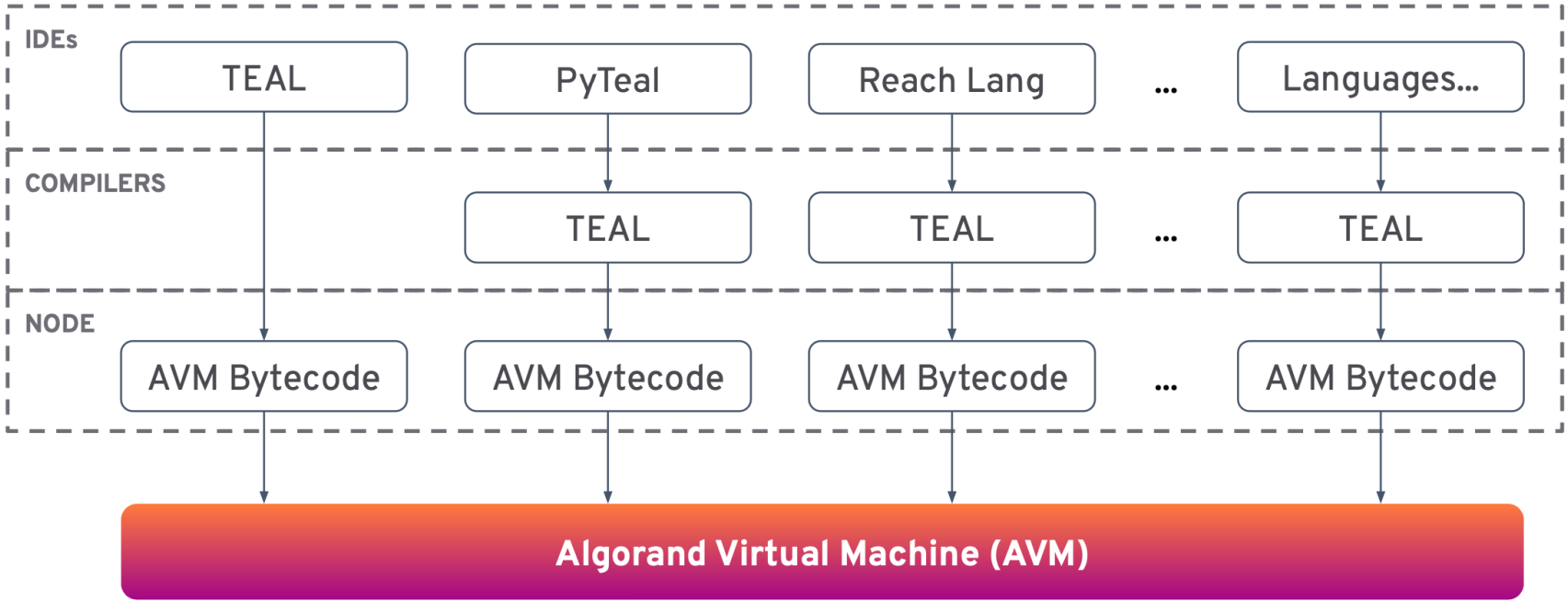
# AVM Architecture



# AVM vs EVM

| | Algorand Virtual Machine | Ethereum Virtual Machine |
|---|---|---|
| **TURING COMPLETENESS** | YES | YES |
| **EXECUTION SPEED** | ~ 4.5 sec<br>regardless dApp complexity | > 20 sec<br>depends on dApp complexity |
| **ENERGY EFFICIENCY** | ~$10^{-5}$ [kWh/txn]<br>all final | ~$10^{2}$ [kWh/txn]<br>not all final |
| **EXECUTION COSTS** | Flat Fee for Smart Contract Calls<br>and Inner Transactions | Depends on dApp complexity |
| **INTEROPERABILITY** | native interoperability<br>ASA, AT, MultiSig, RekeyTo... | user defined solutions / standards |
| **EFFECTS FINALITY** | instant | ~ 6 blocks |
| **MATHEMATICAL PRECISION** | 512 bits | 256 bits |
| **PROGRAMMABILITY** | TEAL, PyTEAL, Reach, ... | Solidity, Viper, Reach, ... |

# Algorand Programming Ecosystem



# Application State

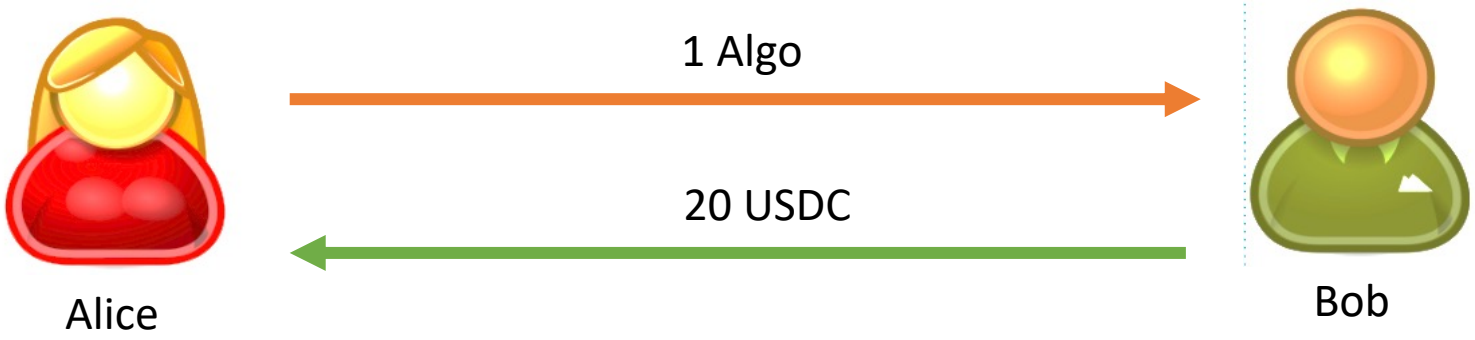
- **Global Storage**
  - 64 key/value pairs
  - Limited to 128 bytes per key/value pair
  - Any app on-chain can read it
  - Minimum Balance Requirement (MBR) funded during the app creation process by the smart contract creator
    - Proportional to the amount of storage used
- **Local Storage**
  - 16 key/value pairs per account
  - Limited to 128 bytes per key/value pair
  - It can be read by any app on-chain
  - MBR funded during opt-in by end-user
    - Proportional to the amount of storage used
  - Accounts must opt-in before the end-user uses the application
  - Can be cleared by the end-user at any time
    - So, do not use local storage for any mission critical data

- **Box Storage**
  - “Unlimited” named storage segments
    - Flexible in terms of types of data and how much data you can store
  - Up to 32kb per box
  - Can only be read by the app that created the box
  - MBR funded during box creation by contract account
    - Proportional to box size

# Transactions

- An application can send any transaction type
  - This includes application calls
- An application can send up to 16 transactions
  - Inner transactions are atomic with the outer transactions
  - One failure will cause all to fail
- Every application has its own contract address it can send transactions from

# Atomic Transfers / Group Transactions




**This is a transaction**

Any transaction can be part of **atomic transfer**, which can include up to 16 transactions.
Either **all** transactions **succeed** or **all** transactions **fail**!

# Creating a Smart Contract with TEAL

- Suppose we want to develop a Smart Contract that approves a transaction if and only if:
  1. Is “Payment” type transaction;
  2. The receiver is a specific “ADDR”;
  3. Fees are less or equal to “1000 microALGO”;
  4. First argument is equal to “bianconiglio”;
  5. Amount is equal to “42 ALGO”;
  6. Or amount is equal to “77 ALGO”;
- To translate those 6 semantically defined example conditions into TEAL, we need to check which transaction fields are going to be controlled by Smart Signature’s logic.

PROGRAM

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
&&
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
txn Amount
int 42
==
txn Amount
int 77
==
||
&&
```

# Translating Conditions into TEAL

1. is “Payment” type transaction;

| | | | | |
|---|---|---|---|---|
| TxType | *required* | string | `"type"` | Specifies the type of transaction. This value is automatically generated using any of the developer tools. |

```
txn TypeEnum
int 1
==
```

1

# Translating Conditions into TEAL

2. the receiver is a specific “ADDR”;

| | | | | |
|---|---|---|---|---|
| Receiver | *required* | Address | `"rcv"` | The address of the account that receives the amount. |

```
txn Receiver                    2
addr AOC7...
==
```

# Translating Conditions into TEAL

3. fees are less or equal to “1000 microALGO”;

| | | | | |
|---|---|---|---|---|
| Fee | *required* | uint64 | `"fee"` | Paid by the sender to the FeeSink to prevent denial-of-service. The minimum fee on Algorand is currently 1000 microAlgos. |

```
txn Fee
int 1000
<=
```

3

# Translating Conditions into TEAL

4. first argument is equal to “bianconiglio”;

| | |
|---|---|
| `arg` | push Args[N] value to stack by index |

```
arg 0                                    4
byte base64 "YmlhbmNvbmlnbGlv"
==
```

# Translating Conditions into TEAL

5. amount is equal to “77 ALGO”;

| | | | | |
|---|---|---|---|---|
| Amount | *required* | uint64 | `"amt"` | The total amount to be sent in microAlgos. |

```
txn Amount
int 77000000
==
```

6

# Translating Conditions into TEAL

6. or amount is equal to “42 ALGO”;

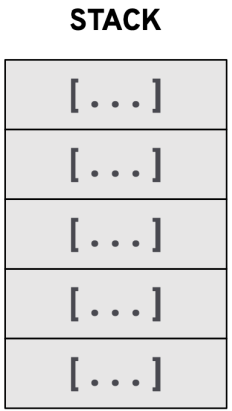
| | | | | |
|---|---|---|---|---|
| Amount | *required* | uint64 | `"amt"` | The total amount to be sent in microAlgos. |

```
txn Amount                5
int 42000000
==
```

# Logic Connectors

This is probably the most complex phase in TEAL programming because you need to keep in mind the state of the stack.

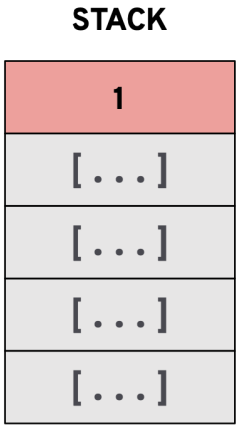
```
txn TypeEnum
int 1
==                                  1

txn Receiver
addr AOC7...
==                                  2

txn Fee
int 1000
<=                                  3

arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==                                  4

txn Amount
int 42000000
==                                  5

txn Amount
int 77000000
==                                  6
```




This phase is drastically simplified with the use of **PyTEAL**, Python binding for TEAL, which automatically performs this concatenation, saving us the effort of thinking about the state of the stack

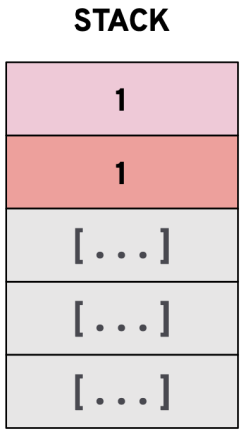
# Execution – Step 1

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



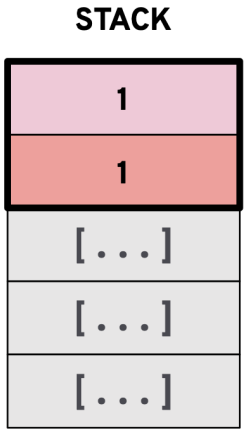
# Execution – Step 2

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



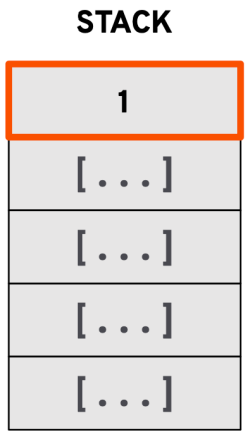
# Execution – Step 3

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



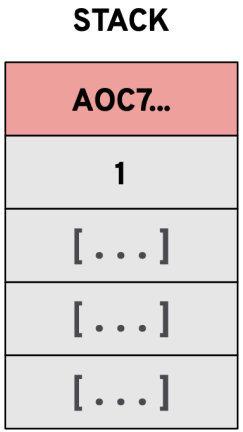
# Execution – Step 4

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



# Execution – Step 5

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



# Execution – Step 6

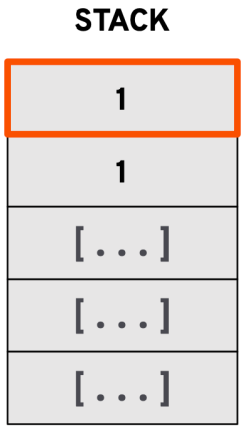
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

# Execution – Step 7

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

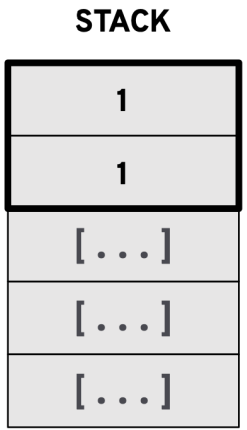
# Execution – Step 8

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



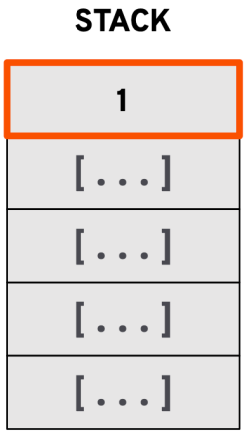
# Execution – Step 9

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



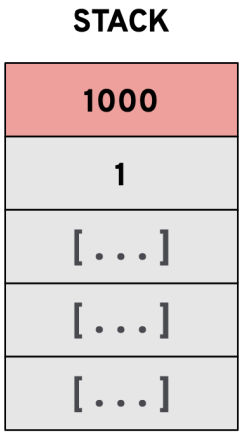
# Execution – Step 10

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



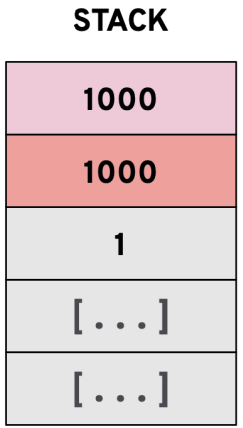
# Execution – Step 11

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



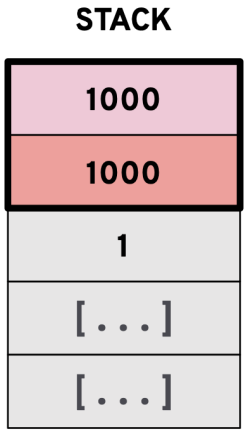
# Execution – Step 12

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



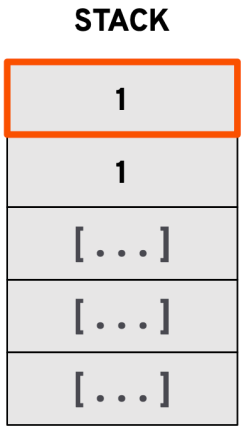
# Execution – Step 13

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



# Execution – Step 14

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



# Execution – Step 15

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

**STACK**

| bianconiglio |
| --- |
| 1 |
| 1 |
| [ . . . ] |
| [ . . . ] |

# Execution – Step 16

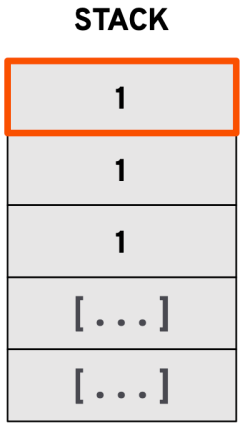
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

# Execution – Step 17

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

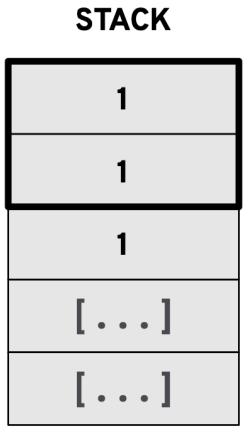
# Execution – Step 18

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



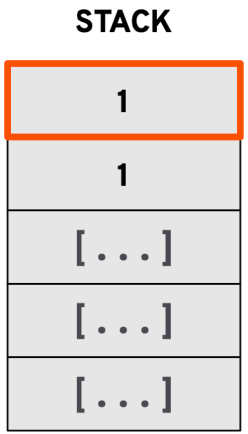
# Execution – Step 19

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



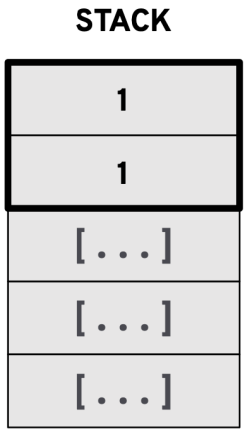
# Execution – Step 20

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



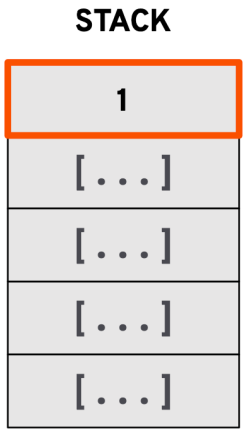
# Execution – Step 21

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



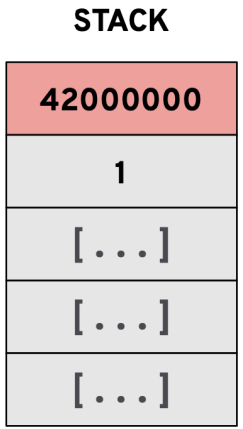
# Execution – Step 22

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



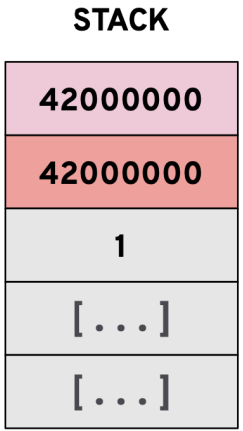
# Execution – Step 23

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



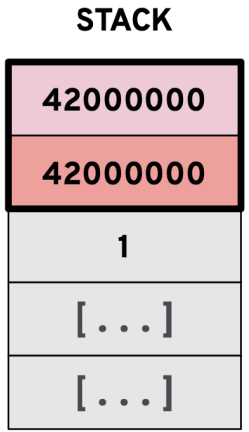
# Execution – Step 24

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



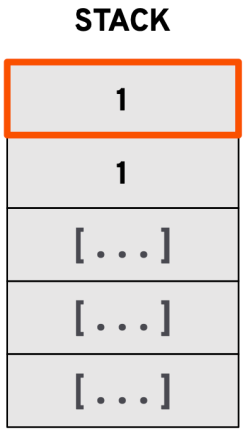
# Execution – Step 25

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



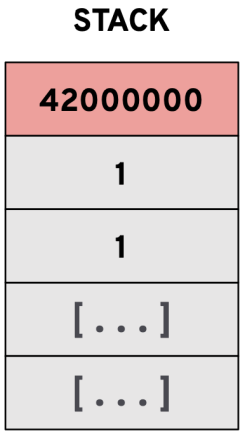
# Execution – Step 26

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



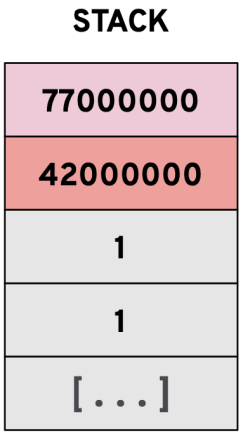
# Execution – Step 27

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



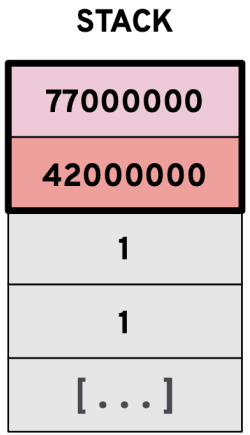
# Execution – Step 28

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



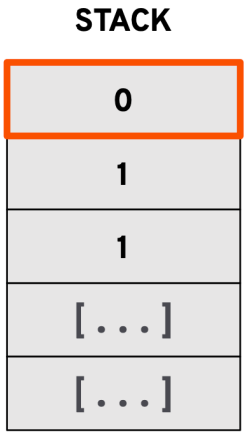
# Execution – Step 29

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



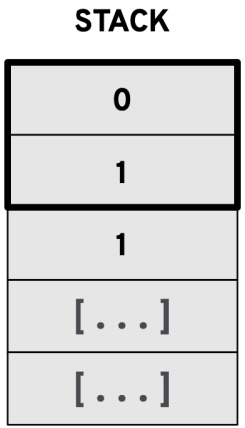
# Execution – Step 30

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



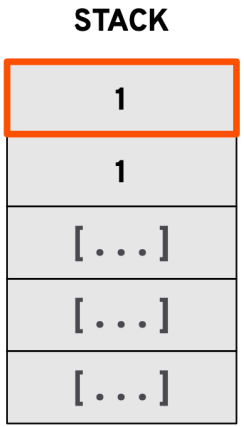
# Execution – Step 31

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



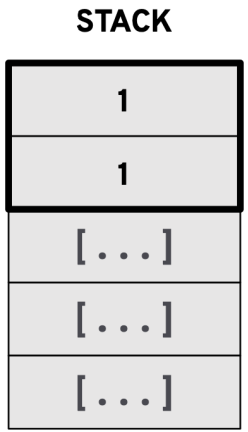
# Execution – Step 32

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



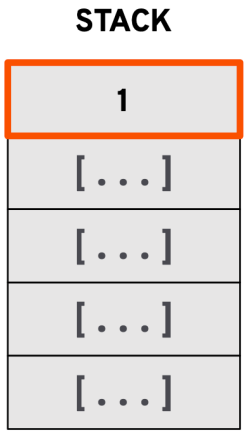
# Execution – Step 33

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



# Execution – Step 34

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



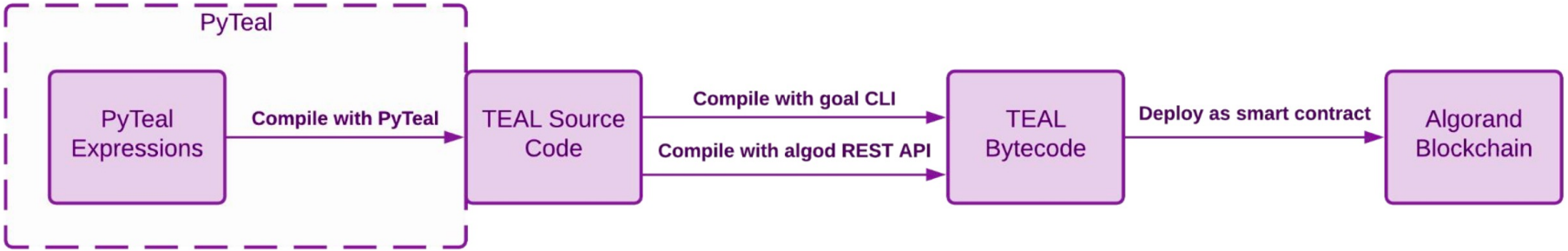
# Conclusion

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "Yml...nb...v"
==
&&
&&
txn Amount
int 4...00
==
txn Amount
int 77000...00
==
||
&&
```

APPROVED

# PyTEAL

- PyTEAL is a Python language binding for Algorand Virtual Machine.
- Allows Smart Contracts and Smart Signatures to be written in Python and then compiled to TEAL
- PyTEAL expressions represent an abstract syntax tree (AST).
- Basically, use Python code to produce TEAL code.

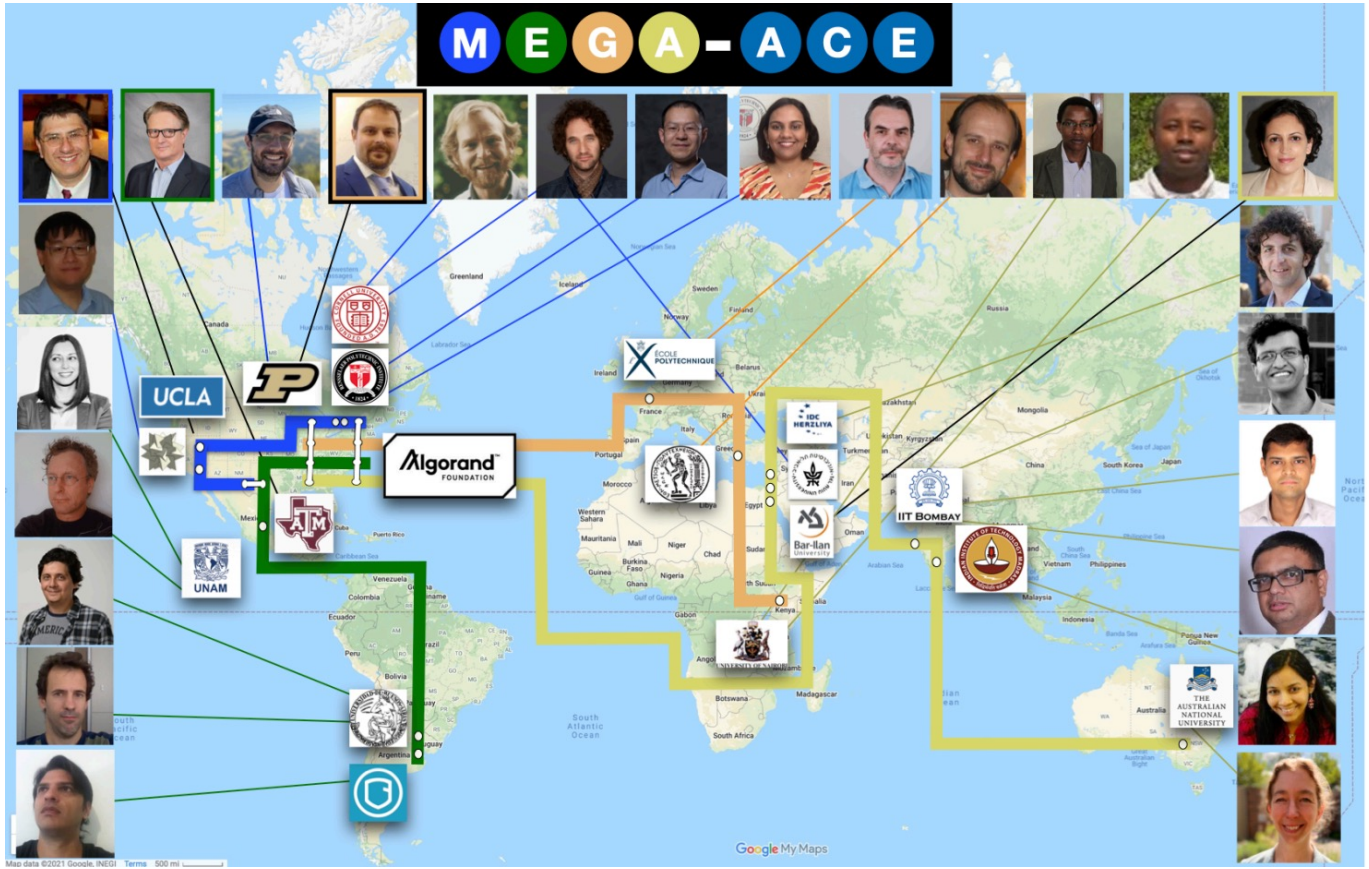


# Algorand Resources

- AlgoDevs on Youtube: https://www.youtube.com/@algodevs
- ACE: https://www.algorand.foundation/ace-learning-resources
- Algokit: https://github.com/algorandfoundation/algokit-cli/blob/main/docs/tutorials/intro.md

Rensselaer

# Multidisciplinary Educational Global Alliance for Algorand Center of Excellence




**mega-ace.org**

# Smart Contract Research

# Swarm Contracts

**Swarm Contracts: Smart Contracts in Robotic Swarms with Varying Behavior;** *Jonathan Grey, Isuru Godage, Oshani Seneviratne. IEEE Blockchain Conference 2020.*

- Allow for heterogenous and multi-stakeholder swarms
  - Robots, AI services, or even humans
- Create a system which is more robust through decentralization and voluntarism
- Generalizable to different applications
- Incentivizes cooperative behavior over long term
- Disincentivizes adversarial behavior over long term

**Potential Applications**

Disaster recovery

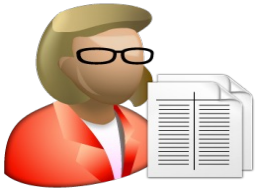
Pooling Resources for Scientific Endeavors

Rensselaer

# Swarm Contract Features

**Agents**

Worker



Chief

Adjudicator

**Public Information**

*Set by contract creator:*

- Adjudicators
- Contract reward
- Job data (will vary depending on application)
- Deadline

*Set later:*

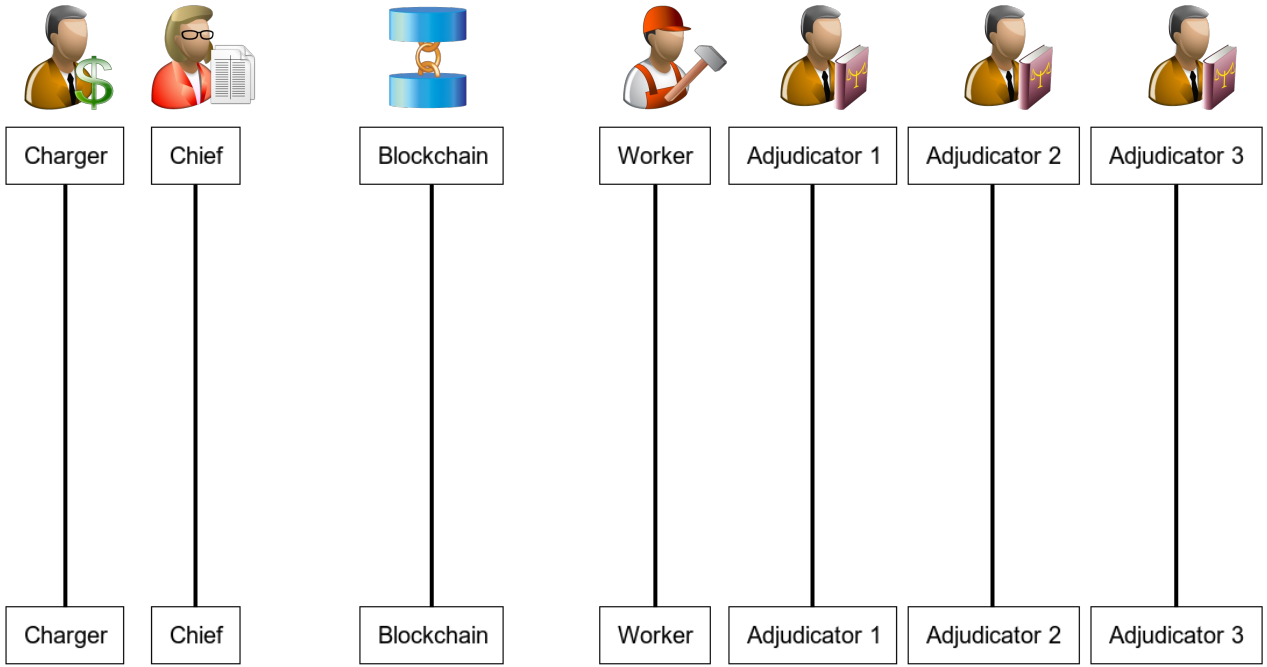
- Acceptor
- Judgments

**Functions**

- Accept
  - Accept the contract by paying insurance
- Adjudicate
  - Submit a judgment, final judgment pays out contract
- Revoke
  - If no one accepts, reclaim funds
- Surrender
  - Back out of contract, for a price

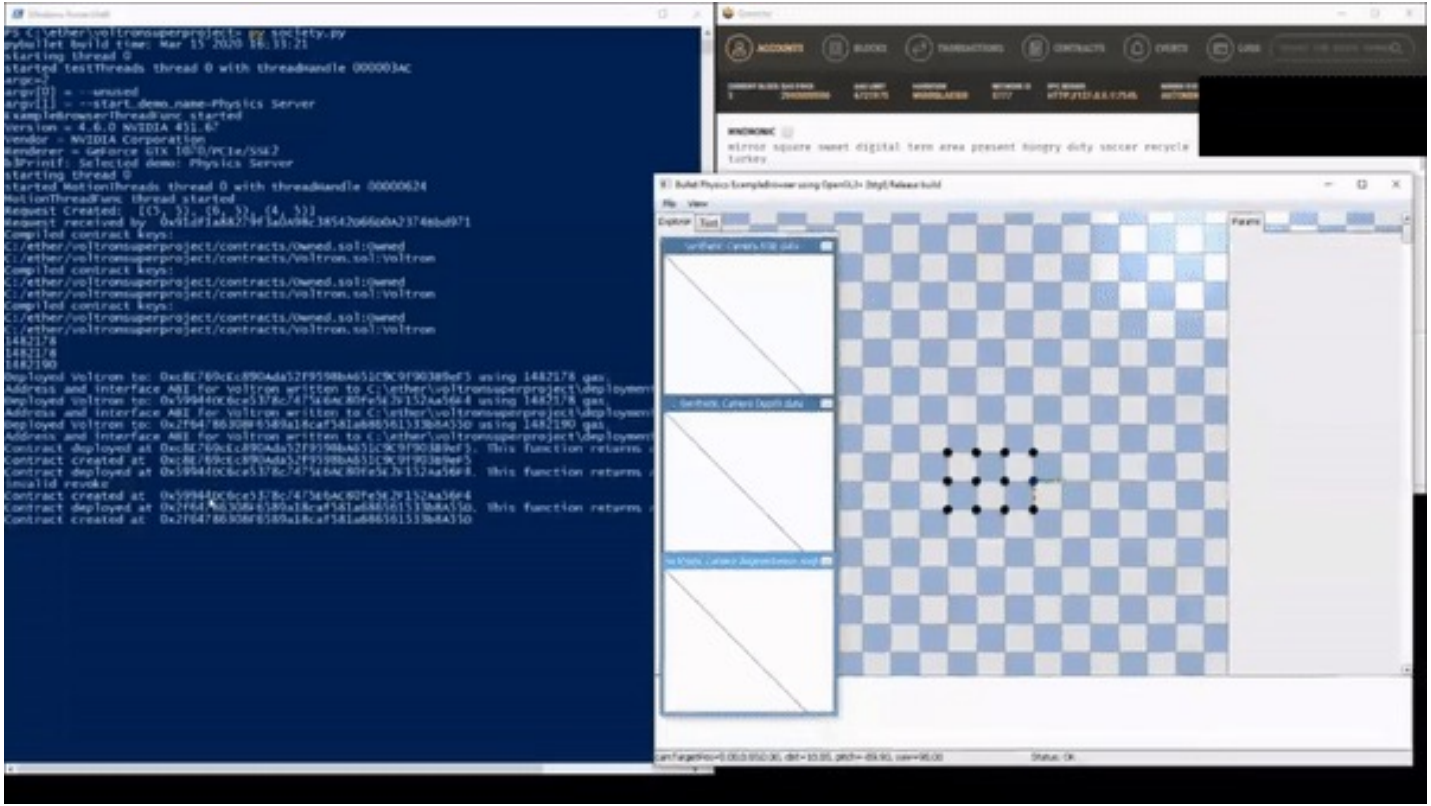
**Agent Sub Types:** Fair Adversarial

# Sequence Diagram



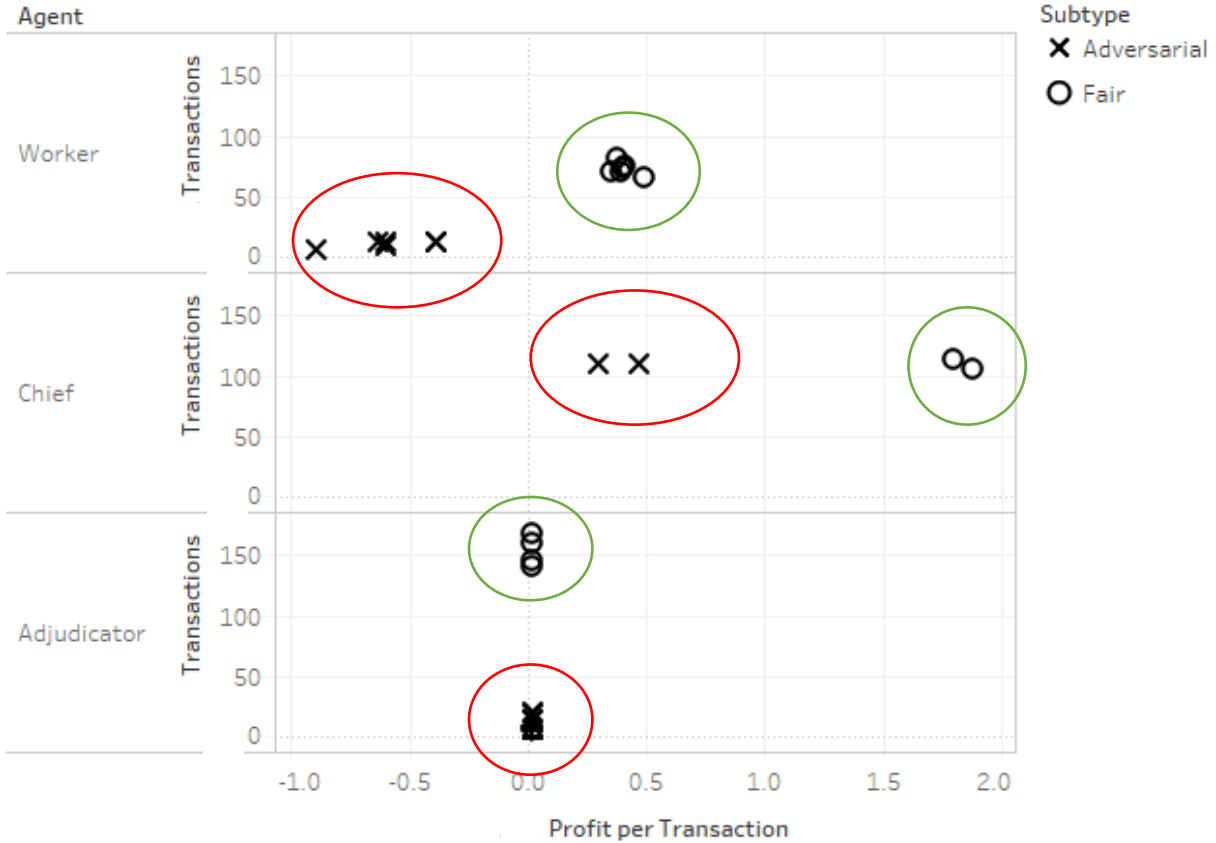
Rensselaer

# Swarm Contract Simulation



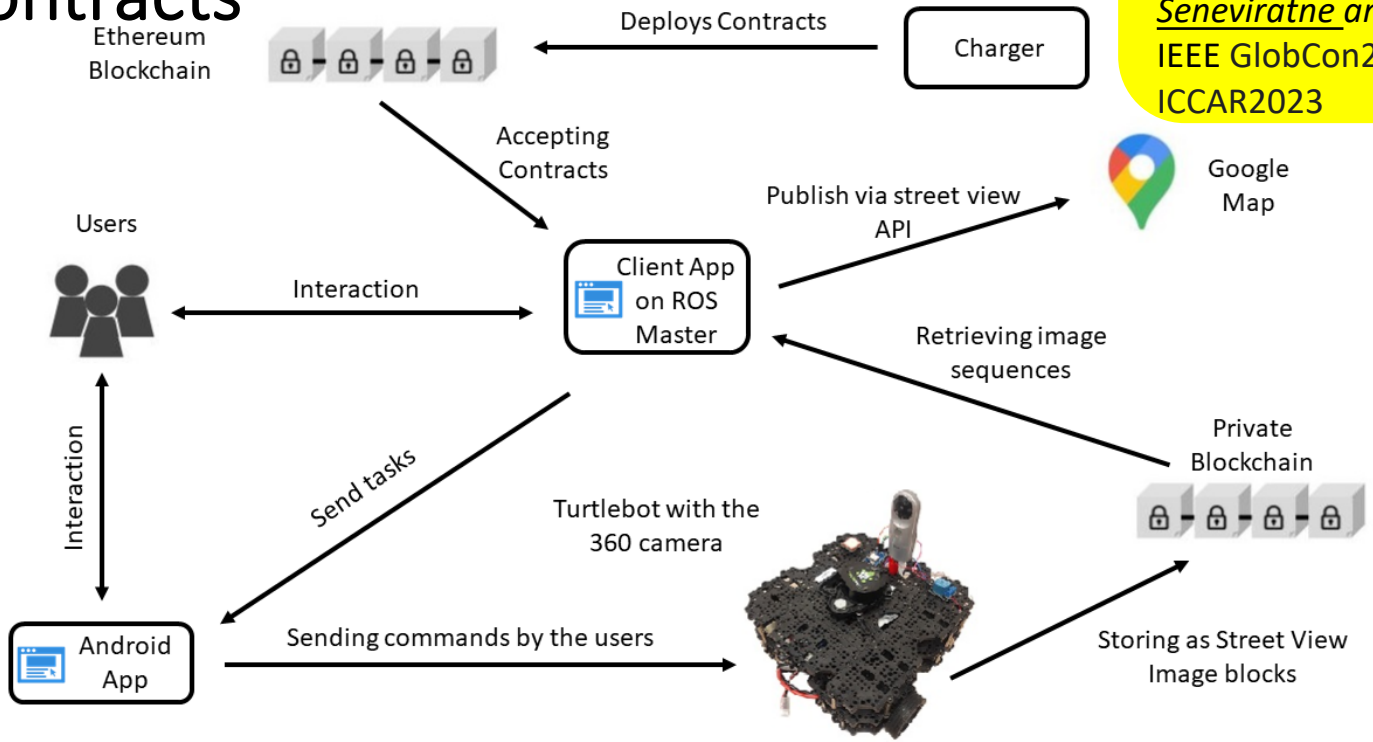
Rensselaer

# Swarm Contract Performance

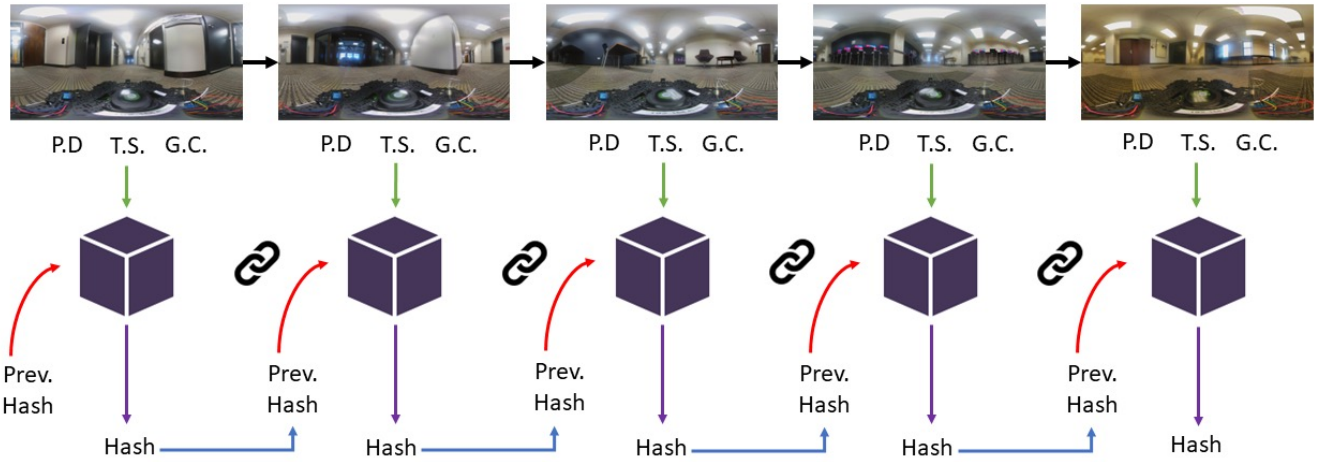


# Collecting Data for Decentralized Knowledge Graph with Smart Contracts

**Decentralized Framework for Collection and Secure Storage of Google Street View Data: Case Study;** *Sanjaya Mallikarachchi, Bonnie Ho, Iyad Kanj, Oshani Seneviratne and Isuru Godage;* IEEE GlobCon2023 and ICCAR2023



# Private Ledger Based Store for Storing Image Sequences




**Legend**

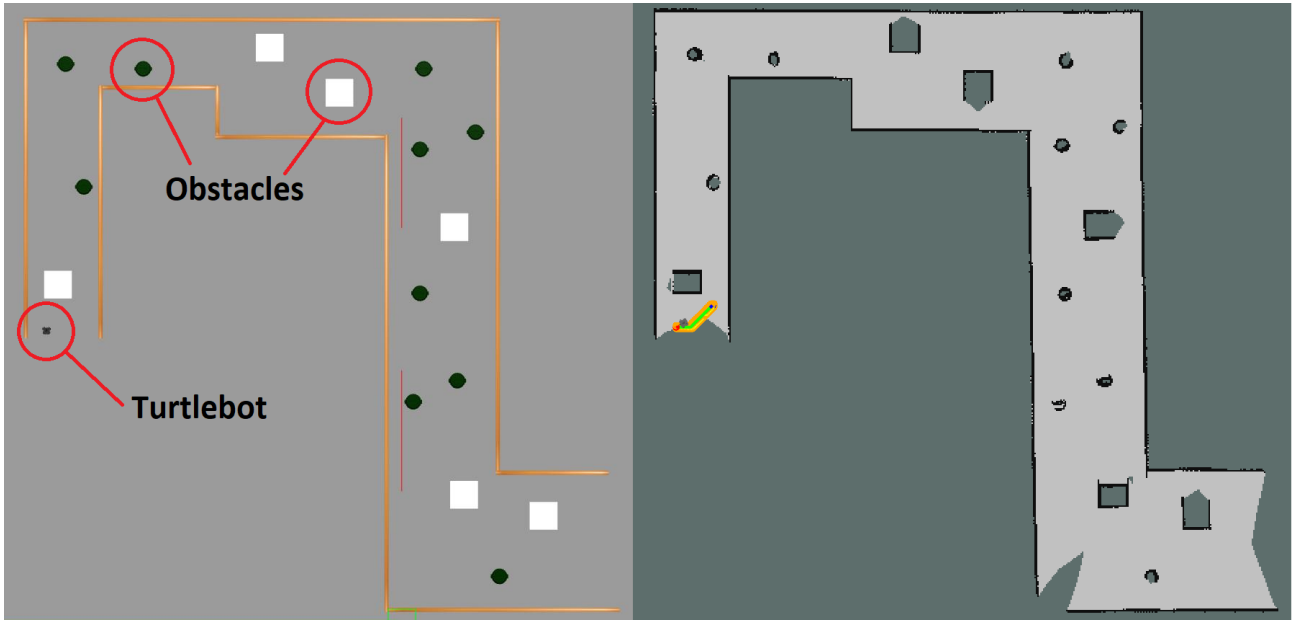
P.D. - Pixel Data

T.S. – Timestamp

G.C. - GPS Coordinate

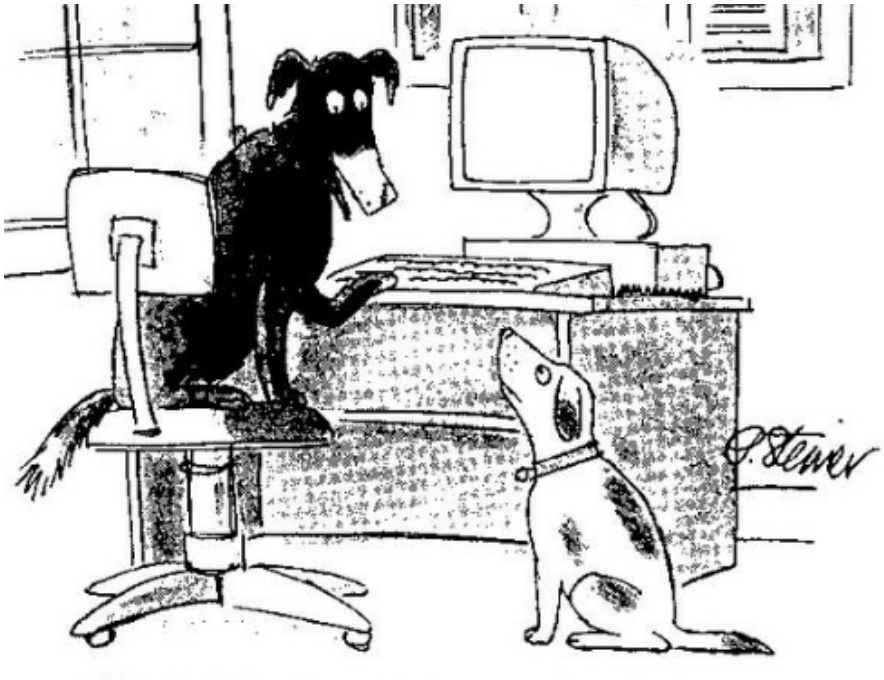
- Each hash is an identifier of the single points of data the agent collects, and they form the basis of **identities in a decentralized knowledge graph**.
- We are expanding this to include knowledge about Points of Interest to create a comprehensive **multi-modal decentralized knowledge graph**.

# Application – Building Indoor Maps

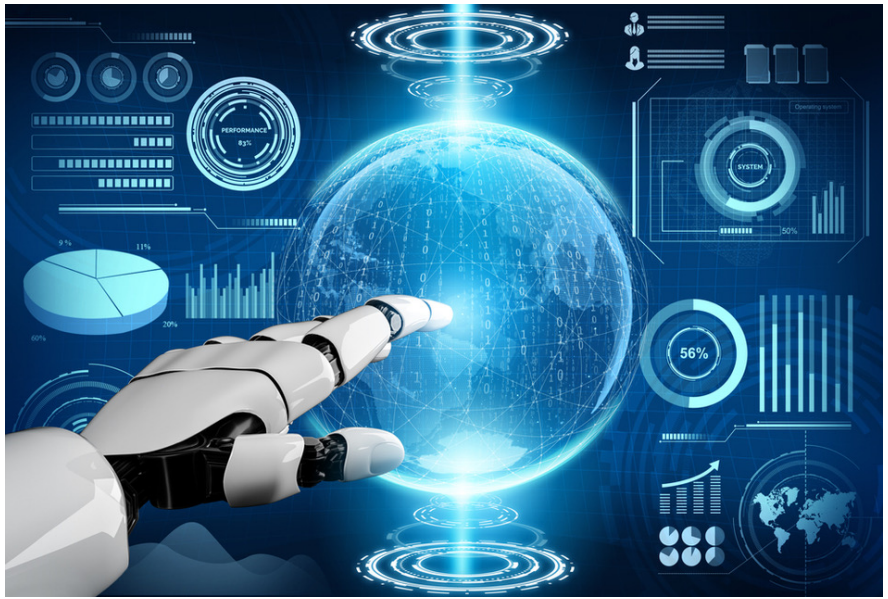



Top-level generated map

Lidar Map

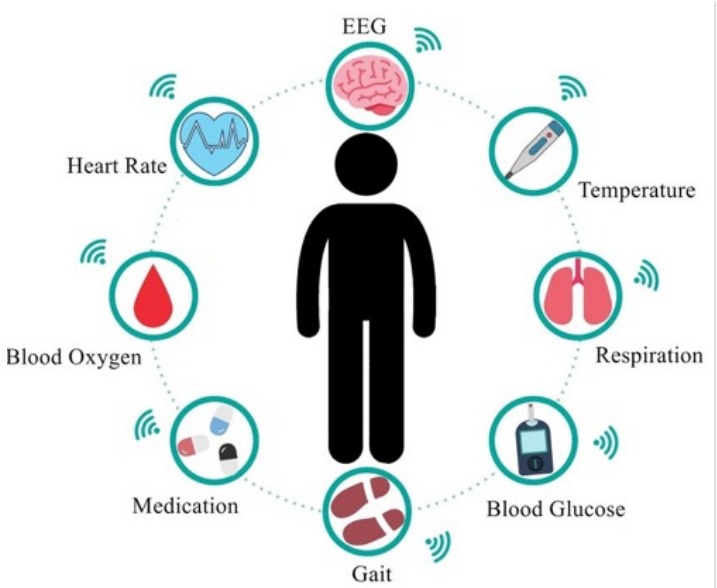


On the Web, nobody knows you are a dog!



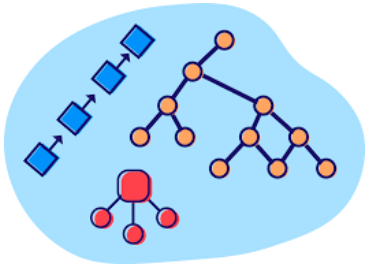

On the Blockchain, nobody knows you are an AI!

# Using Smart Contracts in Data and Computation Heavy Applications

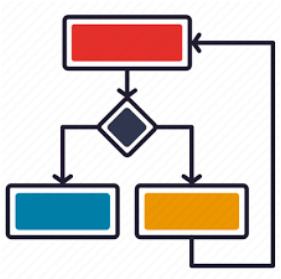



For example, decentralized health applications may need data from various sources with complex schemas where the data streams are fast changing.

Limitations in smart contract programming languages



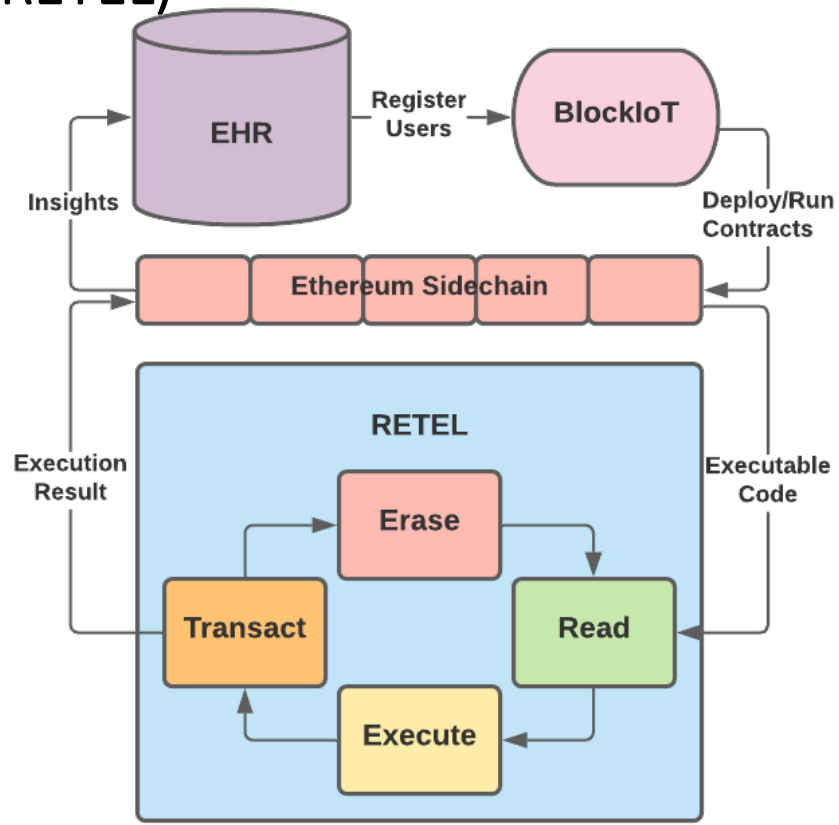
Lack of support for complex data structures



Control structures incur very high gas costs

Rensselaer

# Read-Execute-Transact-Erase-Loop (RETEL)




**BlockIoT-RETEL: Blockchain and IoT Based Read-Execute-Transact-Erase-Loop Environment for Integrating Personal Health Data;** *Manan Shukla, Jianjing Lin, Oshani Seneviratne. IEEE Blockchain Conference 2021.*

**Read:** the high-level python commands for executing smart contract code

**Execute/ Transact:** execute the commands, and input and output history of the execution flow is provided through the RETEL Interpreter

**Erase:** the python interpretation of the solidity smart contract is deleted in preparation for the next smart contract execution

**Loop:** the interpreter will then iterate towards the next smart contract

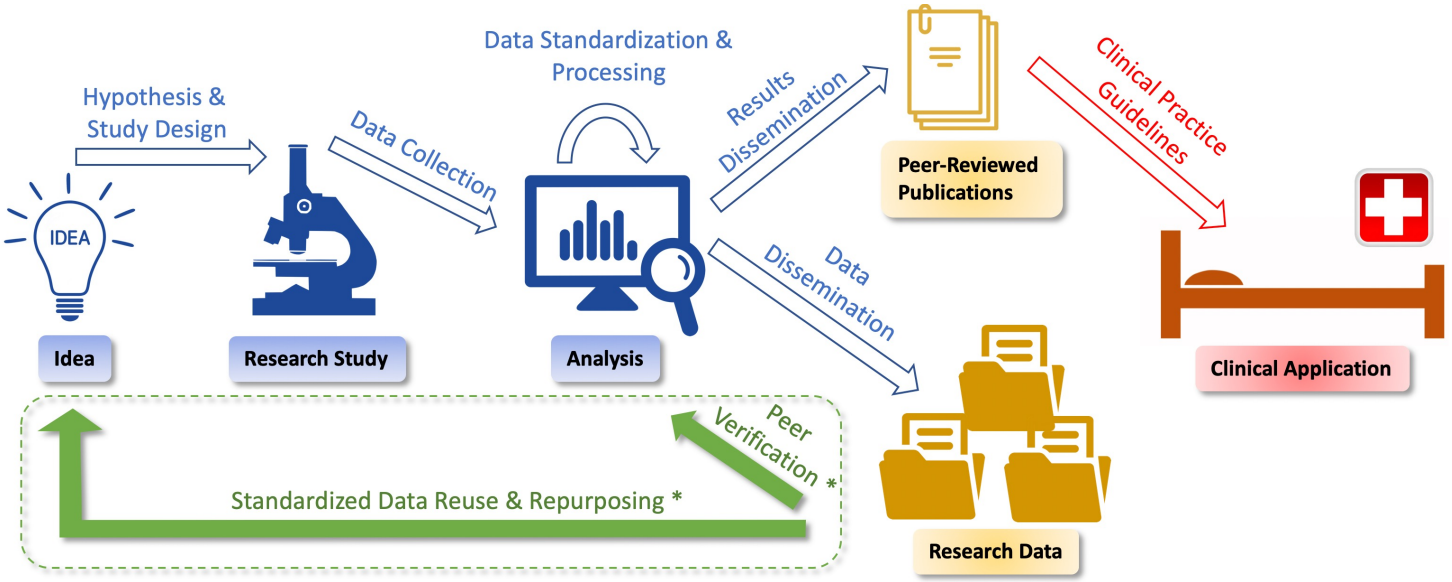
# Incentivized Accountable Research Data Sharing Ecosystem

**Accountable Bench-to-Bedside Data-Sharing Mechanism for Researchers;**
*Oshani Seneviratne, Deborah McGuinness; Transactions on Social Computing, 2023.*

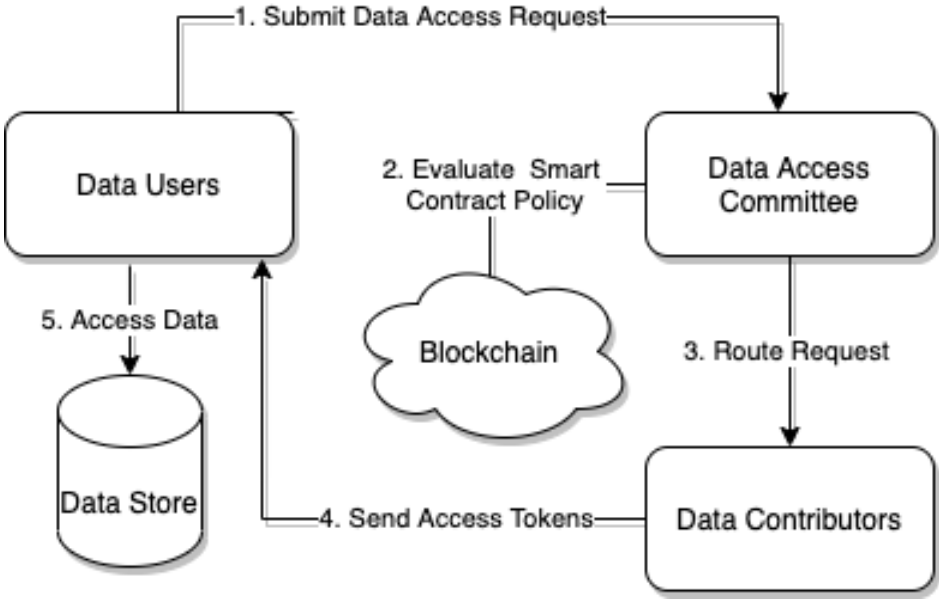
**Goals:**

- Handle "Researcher's Dilemma"
- Reward Reproducible Research and Peer Verification
- Tokenization of Rewards

**Bench-to-Bedside Biomedical Research Scenario:**



# Use Case: COVID-19 National Collaboration (N3C)

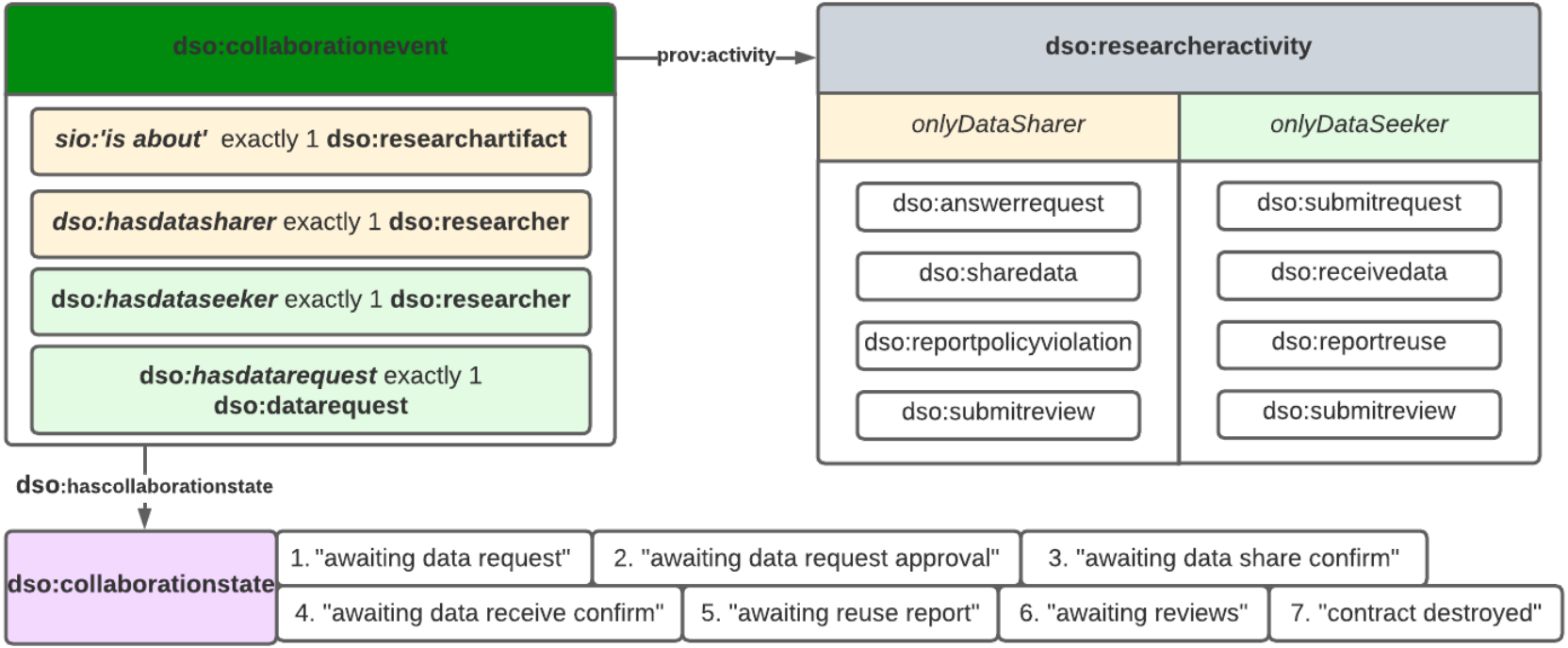



- Data contributors are rewarded for their contributions with “data credits” NFT token.
- Data users must use the data for research purposes.
- The N3C Data Access Committee reviews data access requests.
- We captured these usage requirements and data credits generation in smart contracts.

Rensselaer

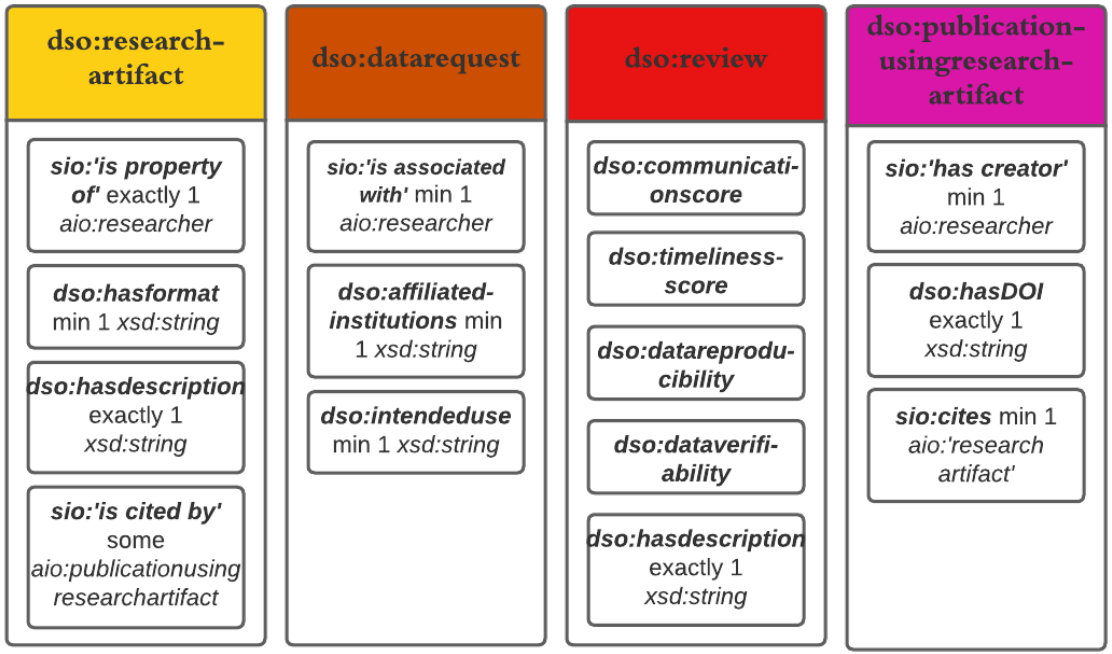
# Semantics-based Framework for Incentivized Research Data Sharing

**Semantics-based Framework for Incentivized Research Data Sharing;** *Kacy Adams, Deborah McGuinness, Oshani Seneviratne; FLAIRS'23.*




Data Sharing Ontology

# Semantics-based Framework for Incentivized Research Data Sharing (contd.)




Data Sharing Template

**Addresses the following competency questions:**

Q1: What does the data seeker intend to use the data for?

Q2. Has the data exchanged hands between the collaborating researchers?

Q3. Which publication by the data seeker uses the shared dataset?

Q4. How many researchers have cited a dataset listed on the protocol?

Q5. Why has a researcher left a specific review?

Rensselaer

# Rewarding Reproducible Research with the SCIENCE Index

**Assessing Scientific Contributions in Data Sharing Spaces;** *Kacy Adams, Fernando Spadea, Conor Flynn, Oshani Seneviratne; Sci-K'23.*

**SCIENCE**

**C**apability-based

**I**ntention-centric

**E**xperiment-oriented

**N**etworked

**C**ollaborative

**E**xpression

- Mechanism to reward researchers for their data contributions
- Supplements the h-index
- To overcome the "cold-start" problem in our data-sharing dApp, we bootstrapped the SCIENCE-index with:
  - Publication data from the Microsoft Academic Graph
  - Data citations from DataCite

Rensselaer

# Questions?

Please feel free to email me at **senevo@rpi.edu**

Twitter: @oshaniws

LinkedIn: https://www.linkedin.com/in/oshani