SICE

Hooiser Tycoons

Building a Decentralized Application with Ethereum’s Smart Contracts

Student: Christopher East – Computer Science

Mentor: Dr. Haghverdi

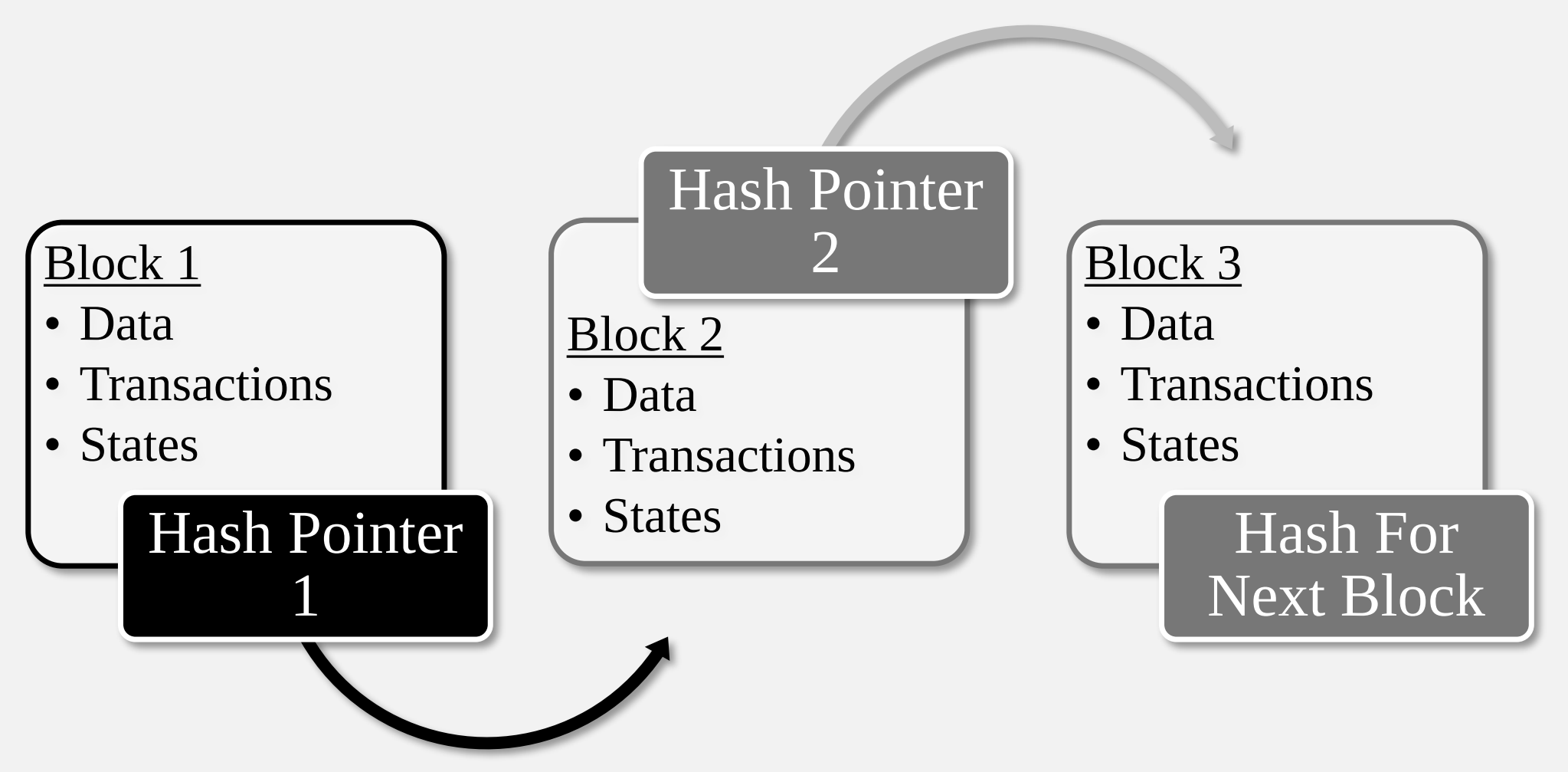
Indiana University SICE, Bloomington Indiana

Introduction

In the last few years, the concept of cryptocurrency has taken the world by storm. Many flock to cryptocurrencies like Bitcoin for their promise as a high return investment. However, the underlying technologies of blockchain, smart contracts, and distributed consensus.

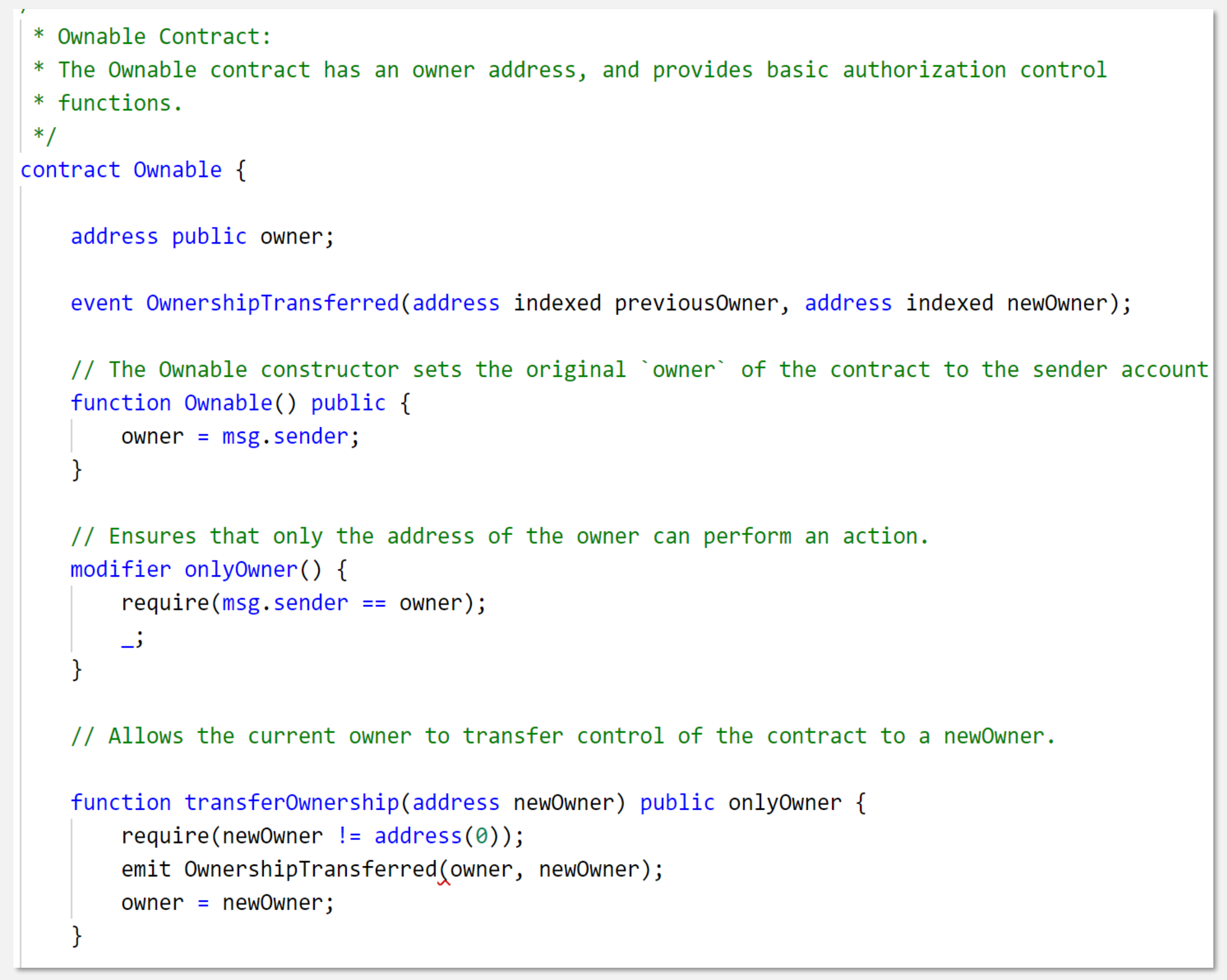
Blockchain

A Blockchain is a cryptographically secured linked list. Each block is a public listing of transactions containing a cryptographic hash pointer to the previous block. Hash algorithms like SHA-256 are nearly collision free (i.e. $SHA_{256}(X) = SHA_{256}(Y) \leftrightarrow X = Y$). This property allows programs to validate the chain of blocks. To add a invalid block to the chain would require alternating the previous block’s hash pointer, which additionally requires alternating the hash of it’s previous block and so on down the entire chain.



Smart Contracts

A “smart contracts” are self-executing programs with strongly defined conditions/terms of the agreement. The difference here is that the code and its conditions exist across a distributed and decentralized blockchain network. Once “deployed” to a blockchain the Smart Contract is immutable and public. Thus allowing smart contracts to act as a trusted intermediates that facilitate traceable and irreversible transactions between anonymous parties in a trustless environment. The most popular platform to develop smart contracts on is the Ethereum. Ethereum offers a “Turing-Complete” virtual machine. Their Solidity language allows for the development of contracts used in decentralized applications.

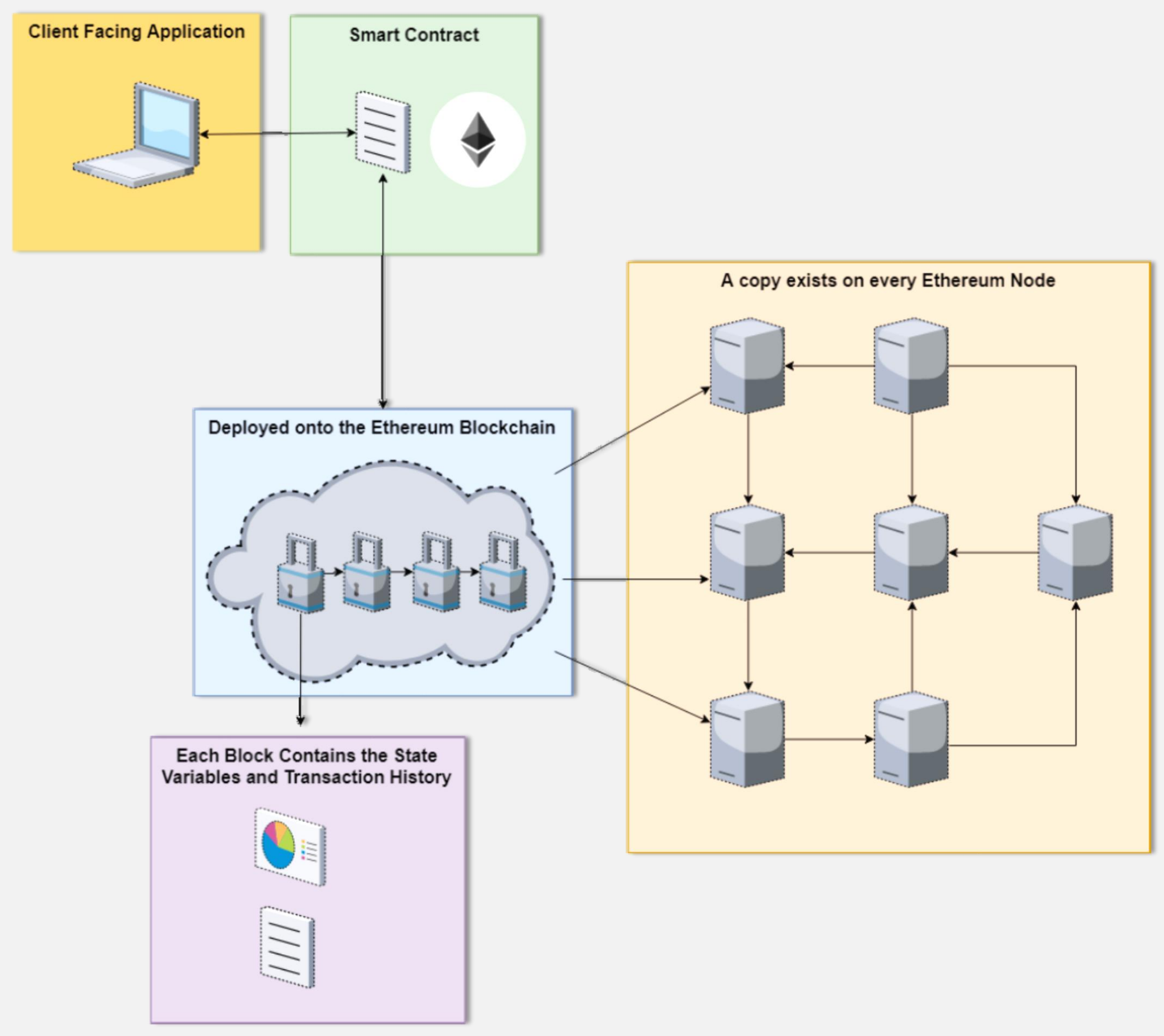


```
 * Ownable Contract:
 * The Ownable contract has an owner address, and provides basic authorization control
 * functions.
 */
contract Ownable {

    address public owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    // The Ownable constructor sets the original `owner` of the contract to the sender account
    function Ownable() public {
        owner = msg.sender;
    }

    // Ensures that only the address of the owner can perform an action.
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }

    // Allows the current owner to transfer control of the contract to a newOwner.

    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0));
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }
```

Decentralized Applications

A Decentralized Application or Dapp, is an application that exists on a blockchain in the form of a smart contract. Modern tools called Web 3.0 allow front end apps to transfer data to and from a blockchain backend. A dapp works by:

1. Taking an essential function and placing it in a smart contract
2. Deploying the smart contract to a blockchain to a special location called an ‘address’
3. Storing variable and state values in the blockchain.


Client Facing Application
Smart Contract
A copy exists on every Ethereum Node
Deployed onto the Ethereum Blockchain
Each Block Contains the State Variables and Transaction History

ERC721 Token

ERC’s (Ethereum Request for Comments) are a set of standards that lay out a set of required function that be implemented. For example, the most common standard is ERC20, which requires methods like “balance of” and “total supply”. An ERC20 token is a smart contract that implements the ERC20 standard. During my research the most interesting token was the ERC721.

ERC721 Tokens are special, because they act like collectible/unique assets much like a collectible baseball card for the following reasons:

- They are non-fungible, meaning they aren’t divisible (you can’t split a card into parts) and their value can’t be used to buy another item of equal value (a card worth $1 can’t be used at a store for another item that is a $1)
- Each individual token has an element that makes it unique from all other tokens (each card has a unique player, stats, rarity, or year)
- The value of a collectible token to a collector is connected to its rareness and individual traits among other items.
- Each token has a distinct and immutable owner

My Dapp - *Hoosier Tycoons*

For my project I set out to develop a Decentralized Application that took advantage of the ERC721 Standard.

Concept:

My focus was to create a novel use case for a ERC721 Token. The idea I decided on was to demonstrate the unique characteristics of a ‘collectible token’ by tokenizing IU Buildings and Landmarks. Each building is uniquely represented by a mixture of real world attributes (GPS coordinates, Name) and generated attributes (Rank, Price, Rent). These unique attributes dictate how often and how much rent the owner receives. Rent is paid directly to the owner’s account by the smart contract.

Hoosier Tycoons is a smart contract that controls:

- The validation and creation of new properties
- The transfer of bought/sold properties
- Handles all escrow and payment functions
- Ensuring proper ownership

Value:

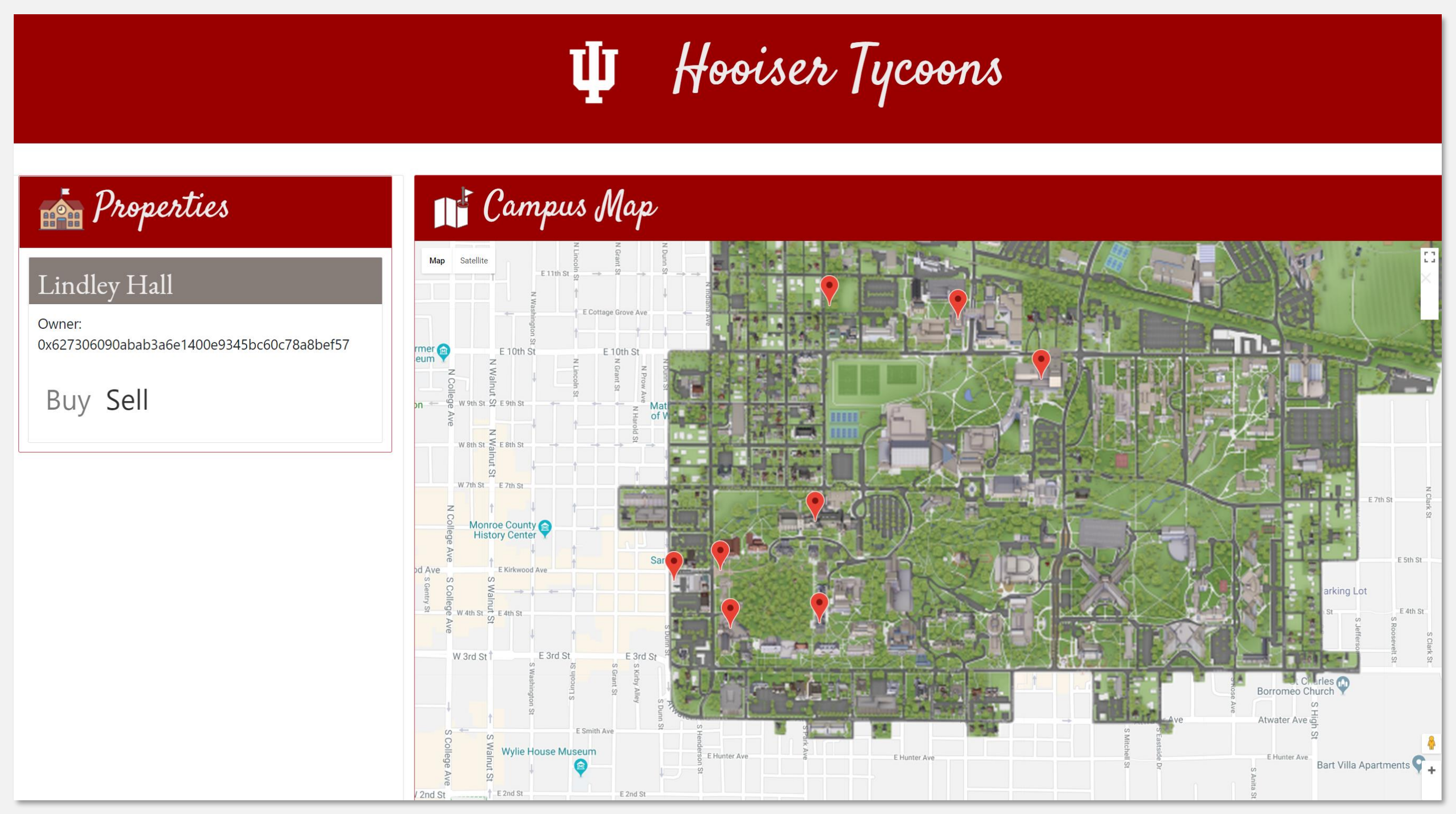
In the collectible model value is driven by desirability and intrinsic value. Some IU Buildings like the Stadium would logically have more intrinsic vale and generate more Rent than others like Lindley. However, some buildings have a desirability that might out weight the Rent they produce. For instance some may compete for the bragging rights to be the sole owner of iconic IU traditions like Kilroy’s, Little 500, or Sample Gates.

Technologies Used:

- Ethereum protocol
- Smart Contracts written in Solidity
- Truffle
- Web3

Future Upgrades:

- Use IPFS (interplanetary file system) to decentralized the storage of the front end
- Build a proprietary Cryptocurrency that’s used to buy tokens and what Rent is paid out in.


Hooiser Tycoons
Properties
Campus Map
Lindley Hall
Owner:
0x627306090abab3a6e1400e9345bc60c78a8bef57
Buy
Sell
Map
Satellite