Velociraptor for Incident Response

ISACA®

San Francisco Chapter

2021 SF ISACA Fall Conference

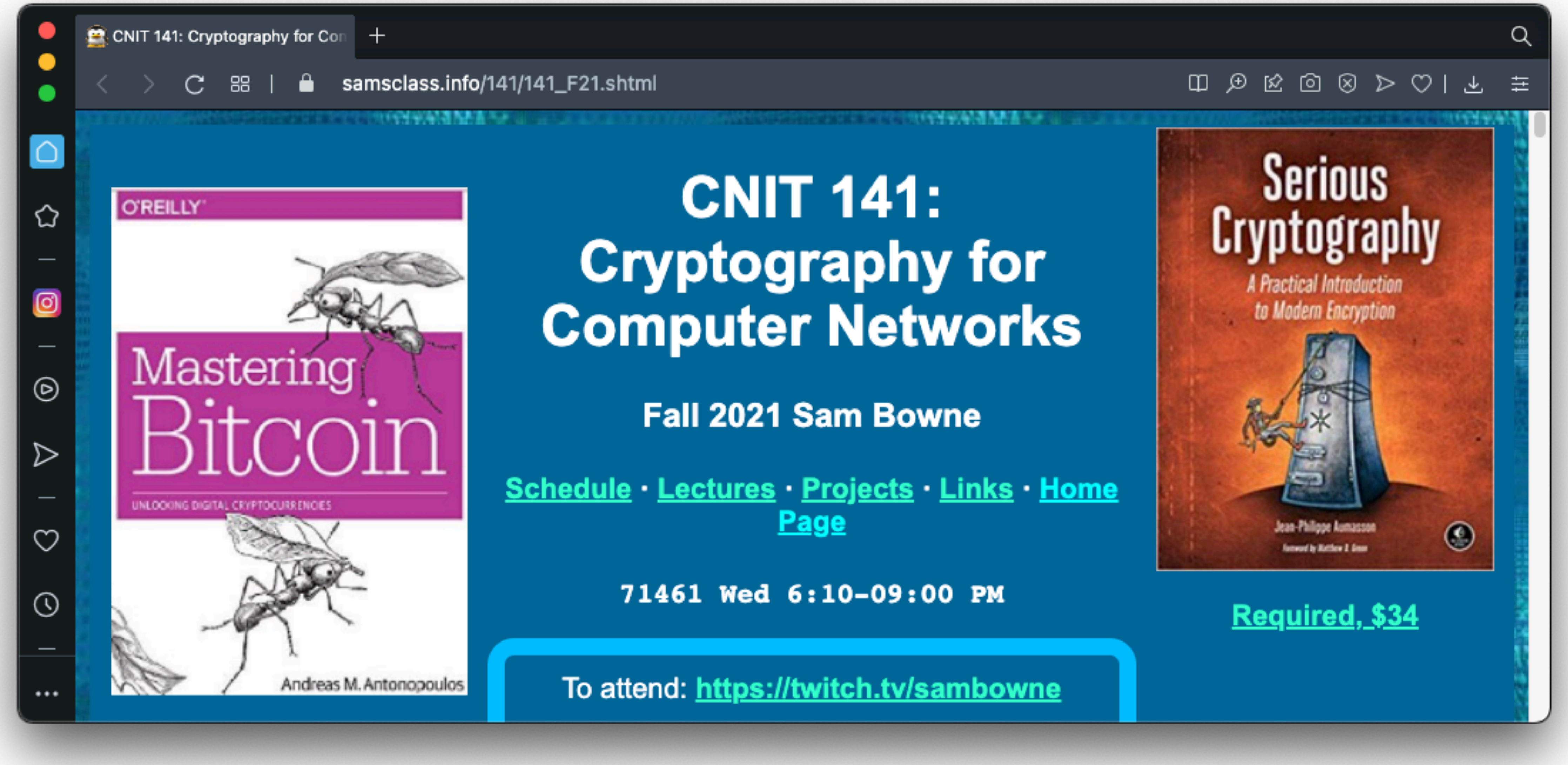
Sam Bowne, Oct 26, 2021

Bio

- Instructor at City College San Francisco
- Founder of Infosec Decoded, Inc.
  - Custom security training for corporations
- Presented at DEF CON, Black Hat, HOPE, etc.

Materials

- This talk and all the materials for these projects are freely available at **samsclass.info**



CNIT 141: Cryptography for Cor
samsclass.info/141/141_F21.shtml
O'REILLY
Mastering Bitcoin
UNLOCKING DIGITAL CRYPTOCURRENCIES
Andreas M. Antonopoulos
Serious Cryptography
A Practical Introduction to Modern Encryption
Jean-Philippe Aumasson

CNIT 141: Cryptography for Computer Networks

Fall 2021 Sam Bowne

Schedule · Lectures · Projects · Links · Home Page

71461 Wed 6:10-09:00 PM

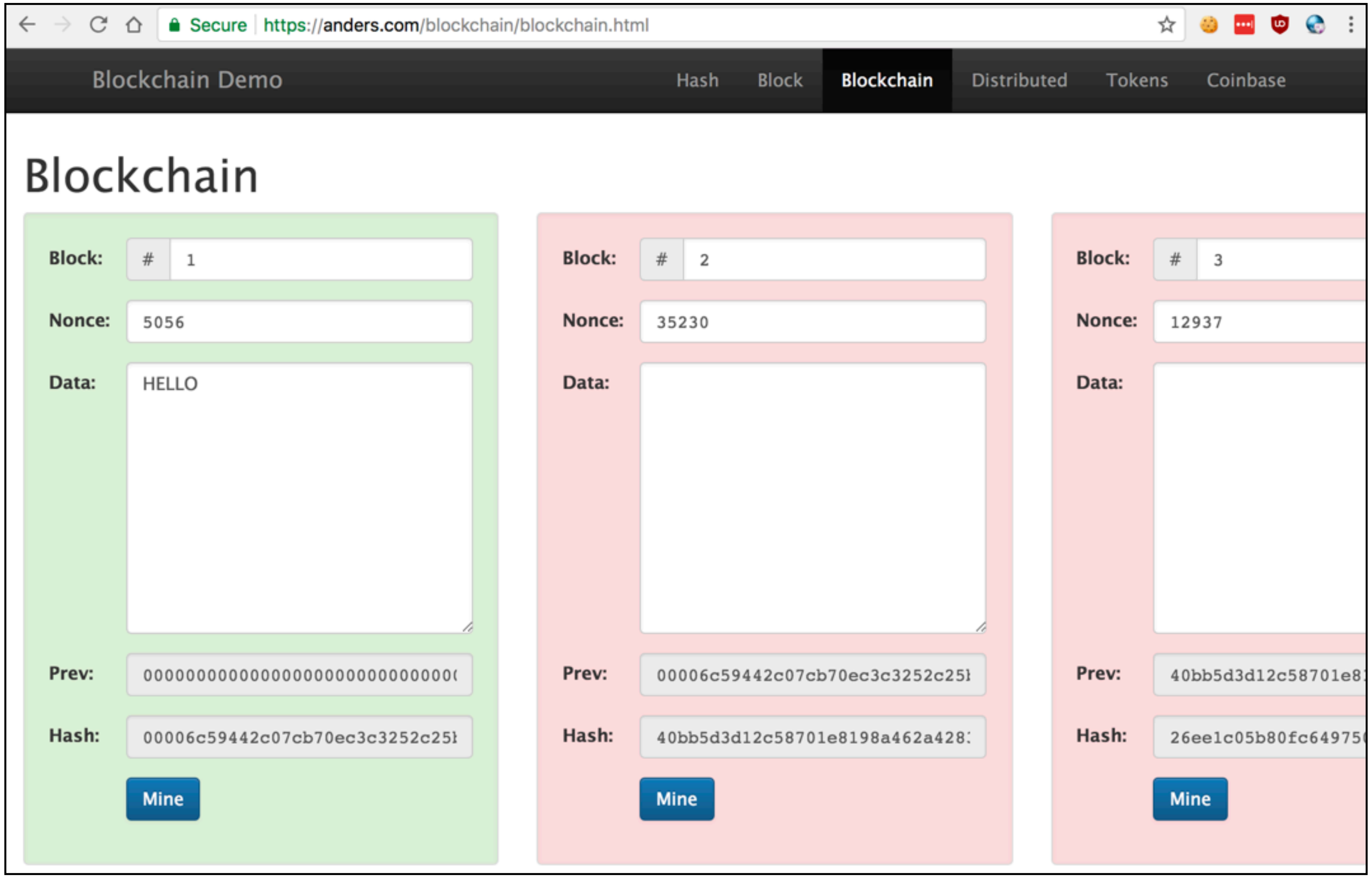
Required, $34

To attend: https://twitch.tv/sambowne

Summary

- **Blockchain**: a distributed database
- **Smart contract**: software running on the blockchain
- **Solidity**: popular smart contract programming language
  - Many security flaws
- **Glow**: new language, much safer

Demo: Blockchain



Blockchain

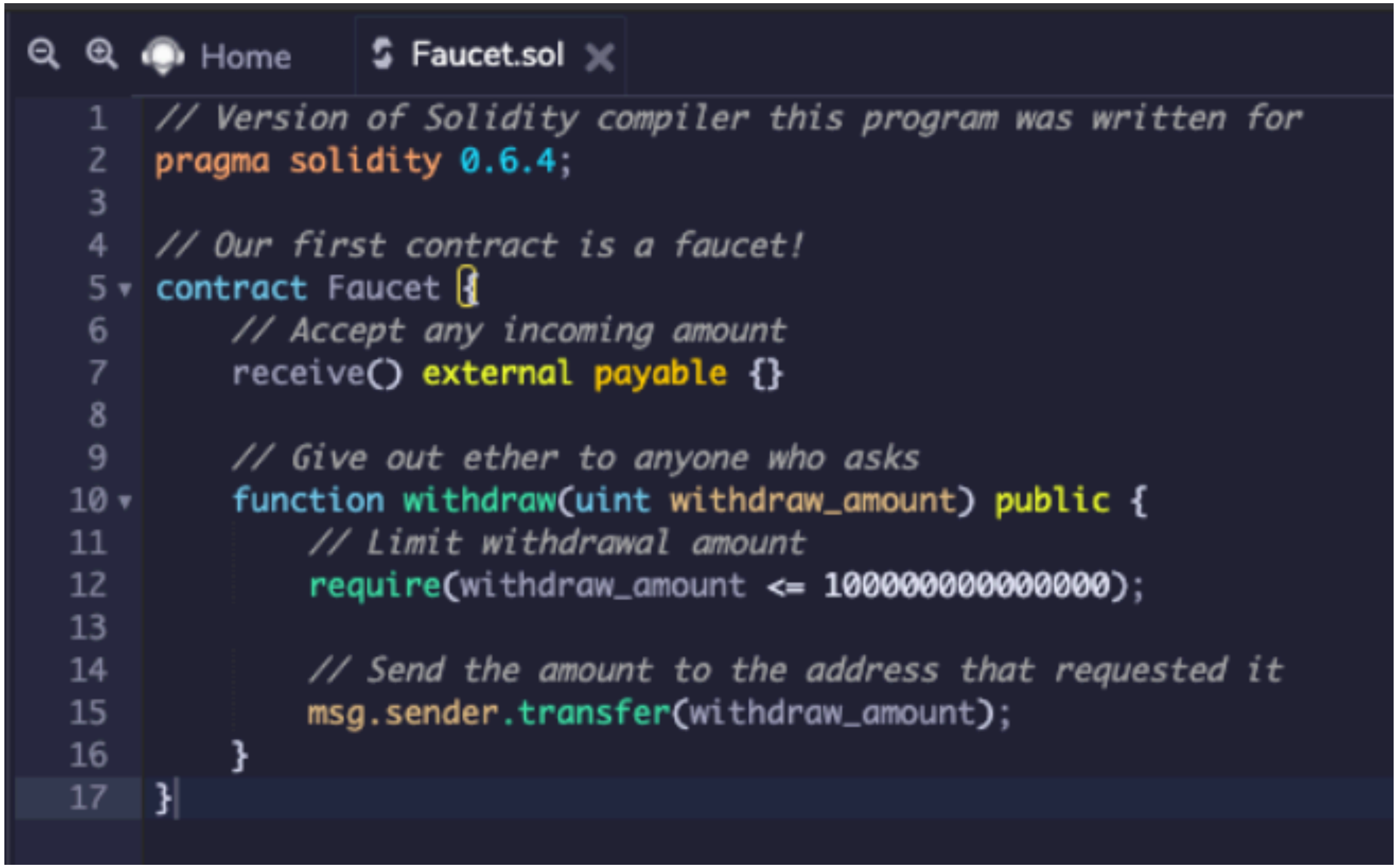
Block: # 1
Nonce: 5056
Data: HELLO
Prev: 00000000000000000000000000000
Hash: 00006c59442c07cb70ec3c3252c25

Block: # 2
Nonce: 35230
Data:
Prev: 00006c59442c07cb70ec3c3252c25
Hash: 40bb5d3d12c58701e8198a462a428

Block: # 3
Nonce: 12937
Data:
Prev: 40bb5d3d12c58701e8
Hash: 26ee1c05b80fc64975

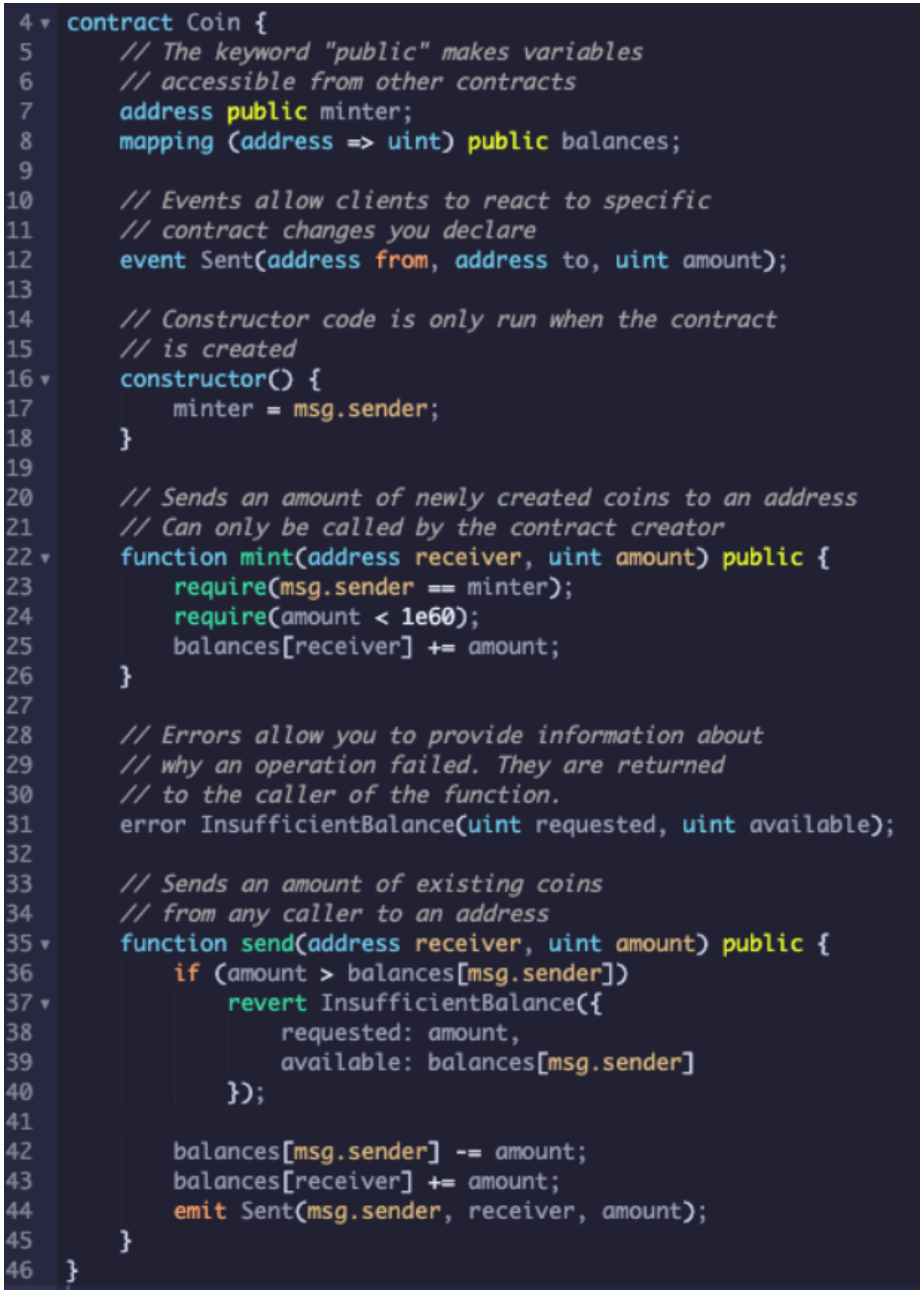
- https://andersbrownworth.com/blockchain/blockchain

Demo: Faucet



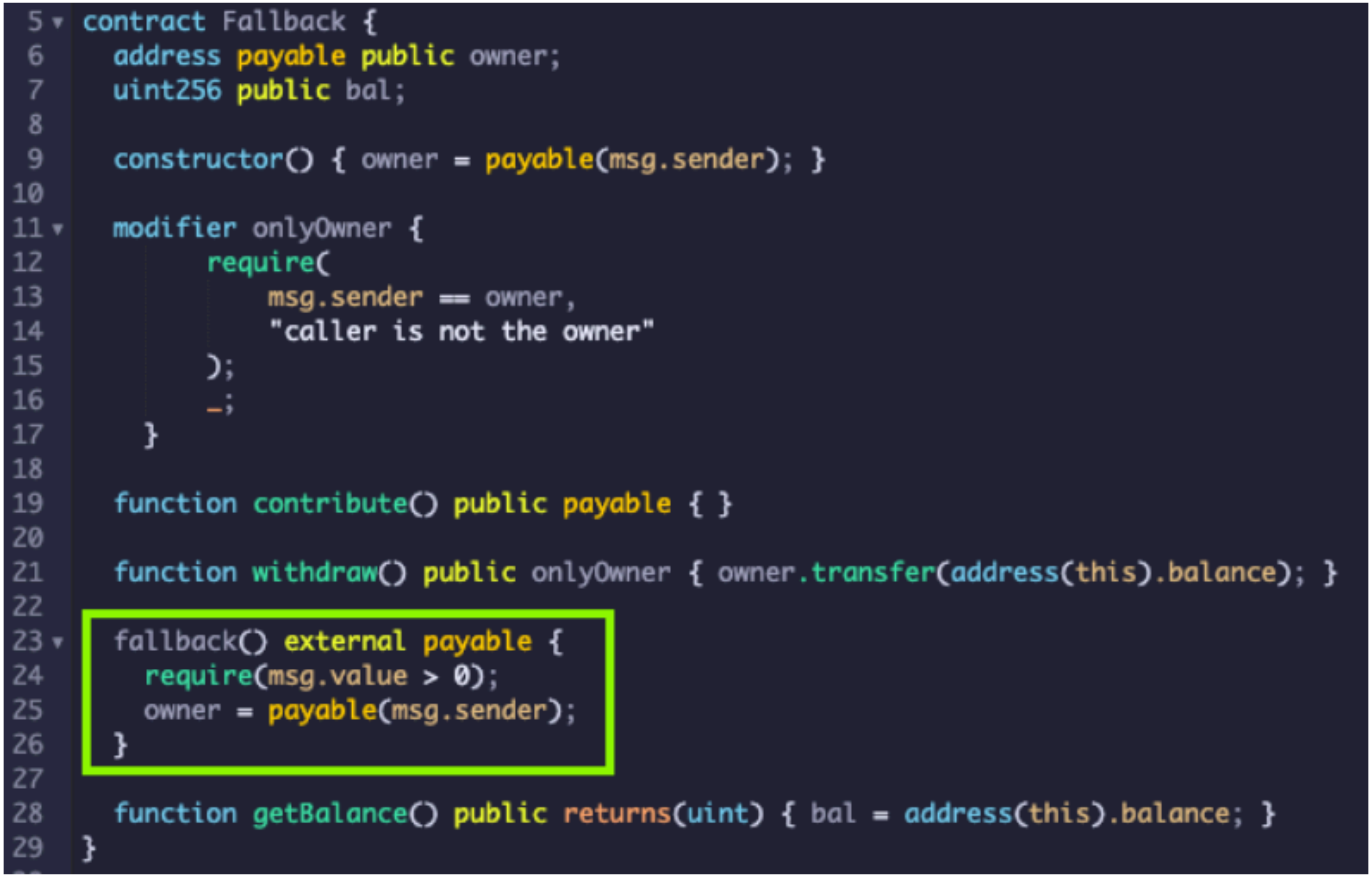
```
// Version of Solidity compiler this program was written for
pragma solidity 0.6.4;

// Our first contract is a faucet!
contract Faucet {
    // Accept any incoming amount
    receive() external payable {}

    // Give out ether to anyone who asks
    function withdraw(uint withdraw_amount) public {
        // Limit withdrawal amount
        require(withdraw_amount <= 100000000000000);

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

Demo: Minting a Coin



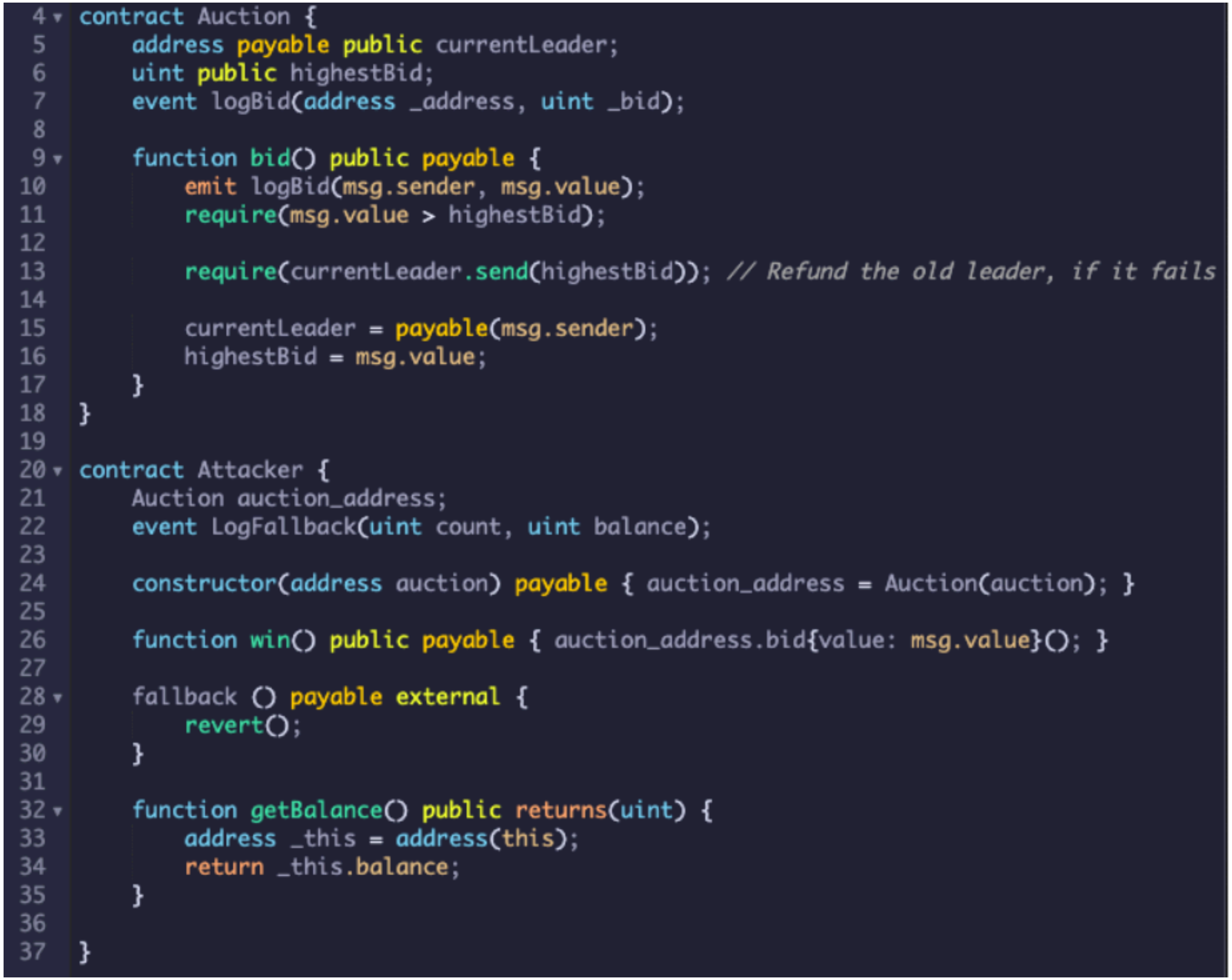
```
contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }

    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });

        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Demo: Fallback Function



```
contract Fallback {
  address payable public owner;
  uint256 public bal;

  constructor() { owner = payable(msg.sender); }

  modifier onlyOwner {
        require(
            msg.sender == owner,
            "caller is not the owner"
        );
        _;
    }

  function contribute() public payable { }

  function withdraw() public onlyOwner { owner.transfer(address(this).balance); }

  fallback() external payable {
    require(msg.value > 0);
    owner = payable(msg.sender);
  }

  function getBalance() public returns(uint) { bal = address(this).balance; }
}
```

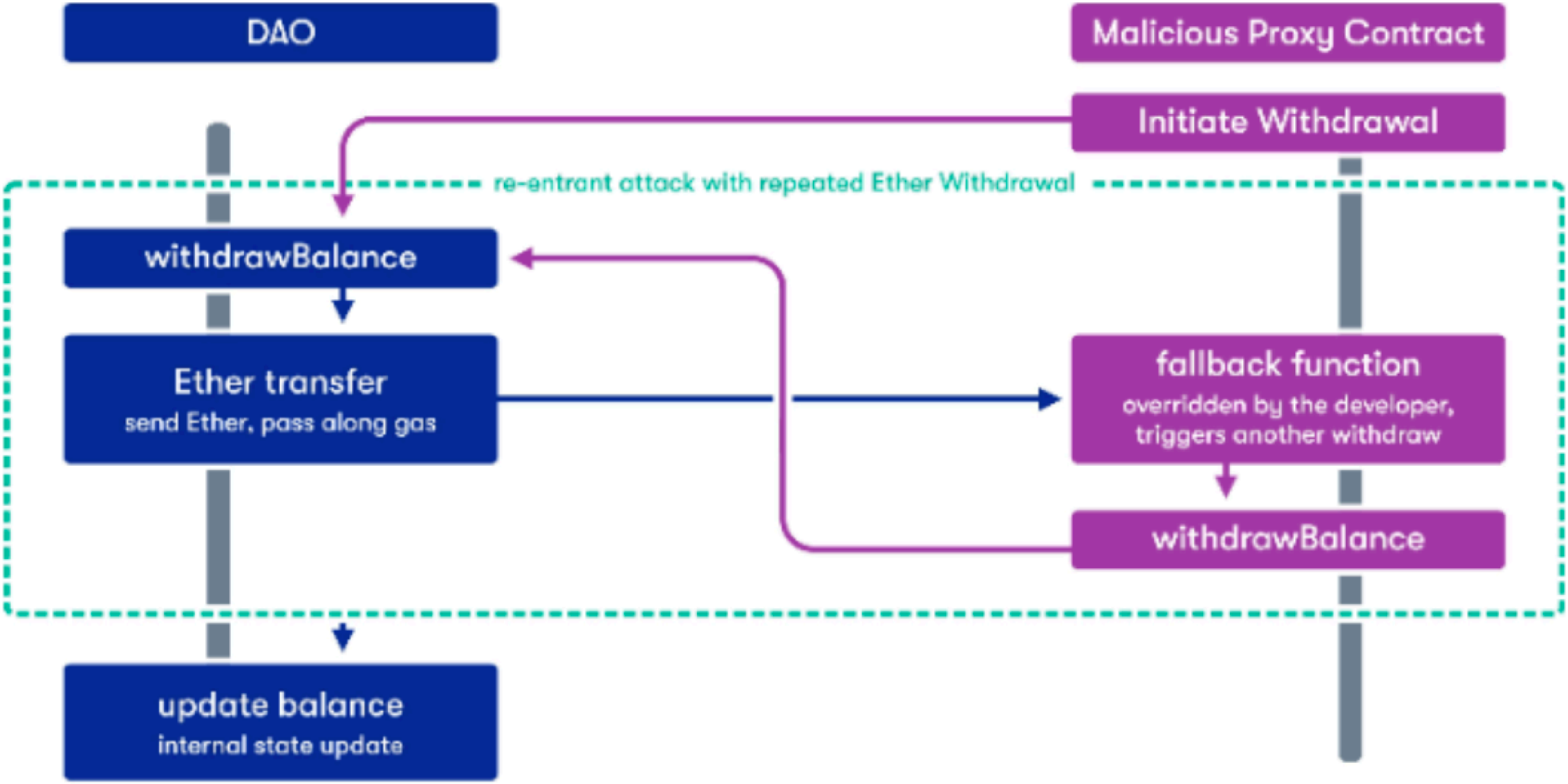
Demo: Auction



```
contract Auction {
    address payable public currentLeader;
    uint public highestBid;
    event logBid(address _address, uint _bid);

    function bid() public payable {
        emit logBid(msg.sender, msg.value);
        require(msg.value > highestBid);

        require(currentLeader.send(highestBid)); // Refund the old leader, if it fails

        currentLeader = payable(msg.sender);
        highestBid = msg.value;
    }
}

contract Attacker {
    Auction auction_address;
    event LogFallback(uint count, uint balance);

    constructor(address auction) payable { auction_address = Auction(auction); }

    function win() public payable { auction_address.bid{value: msg.value}(); }

    fallback () payable external {
        revert();
    }

    function getBalance() public returns(uint) {
        address _this = address(this);
        return _this.balance;
    }

}
```

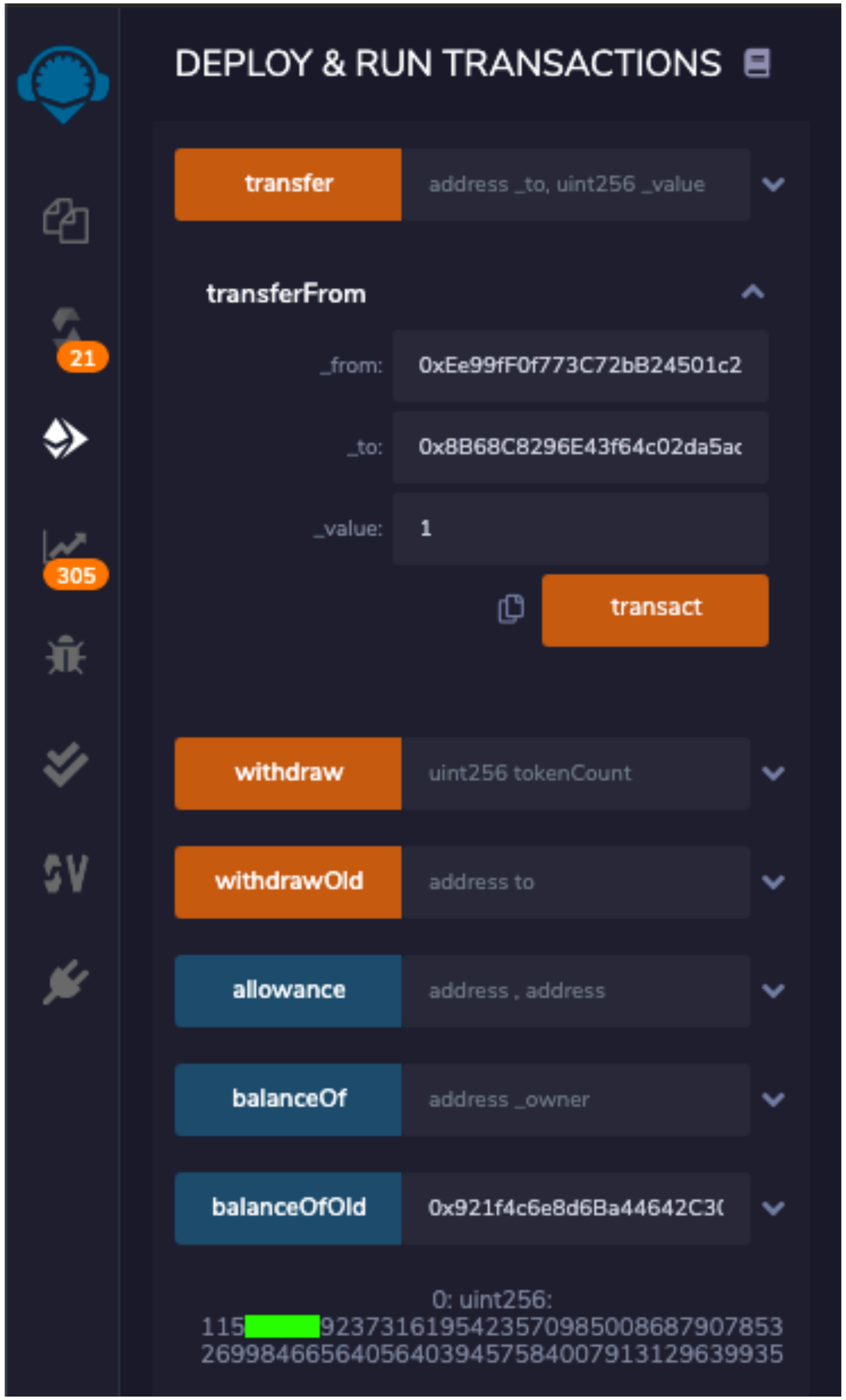
Demo: Reentrancy Attack

DAO: Over 1000 lines of Solidity



Demo: PoWHCoin Integer Underflow

Contract is 306 lines of Solidity



transferFrom

_from: 0xEe99fF0f773C72bB24501c2

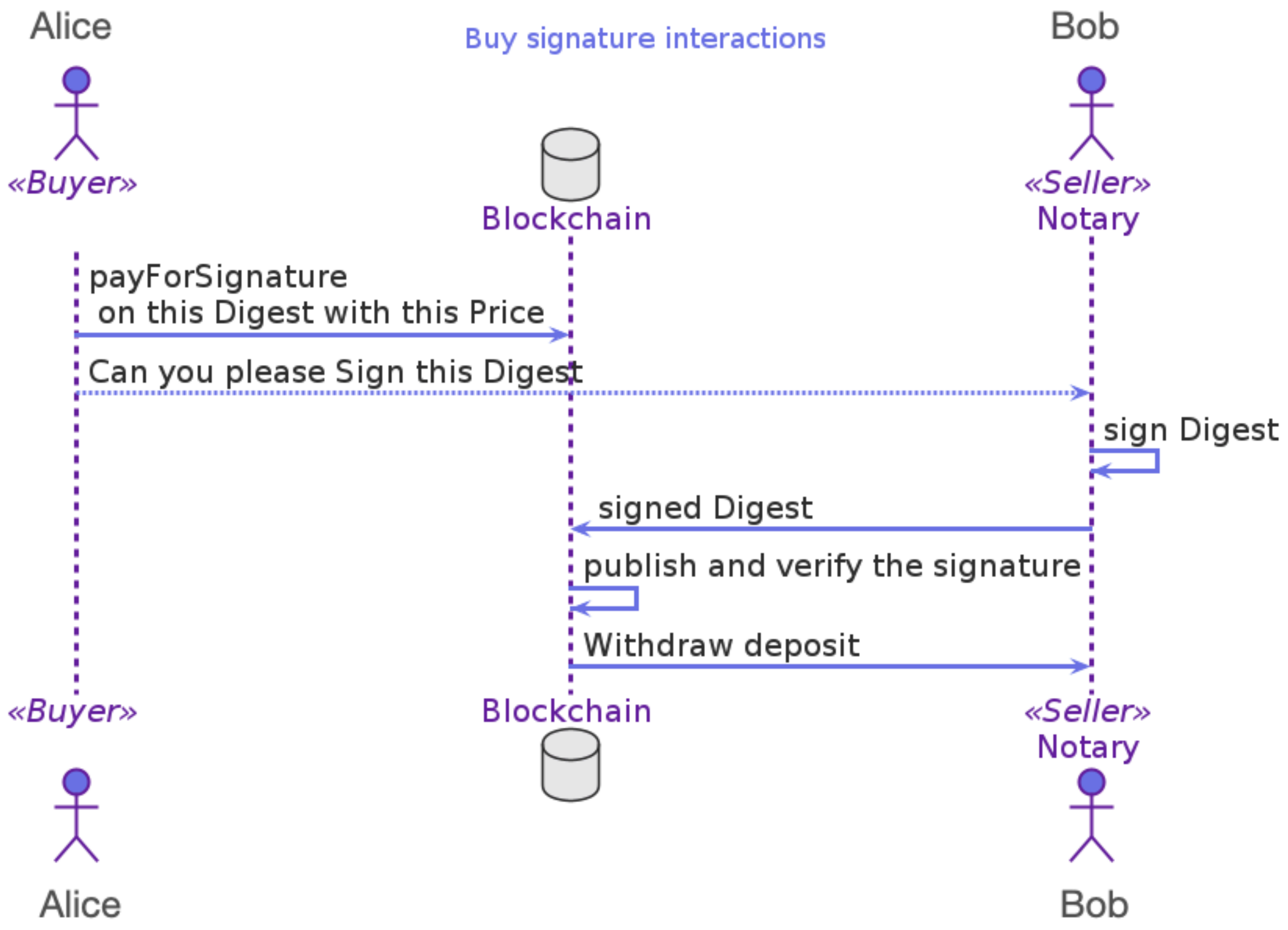
_to: 0x8B68C8296E43f64c02da5ac

_value: 1

balanceOfOld 0x921f4c6e8d6Ba44642C3(

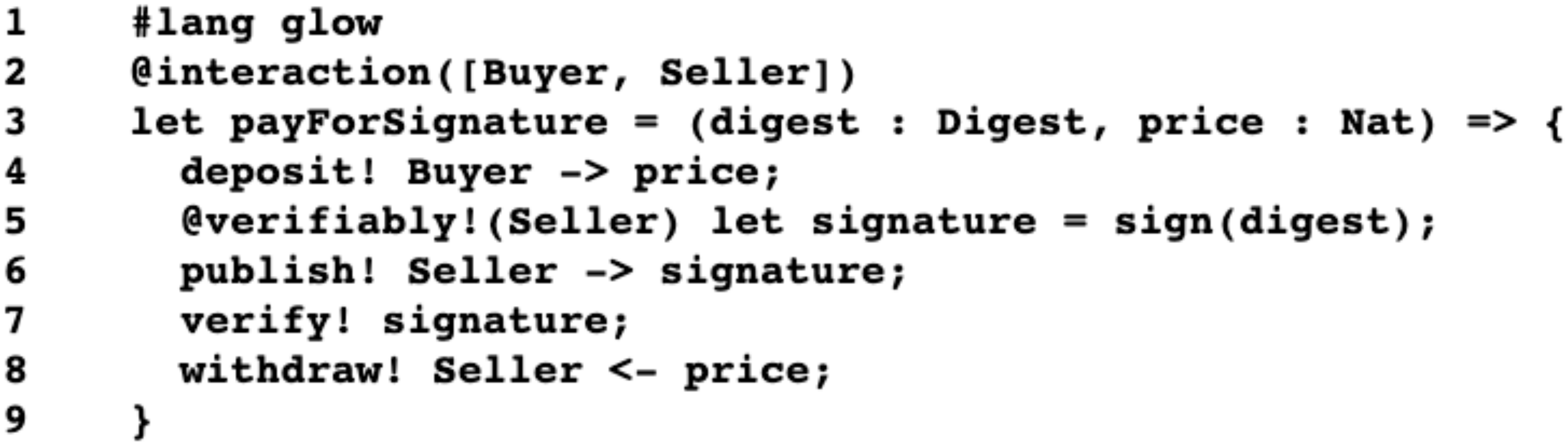
```
0: uint256:
115[redacted]9237316195423570985008687907853
269984665640564039457584007913129639935
```

Glow



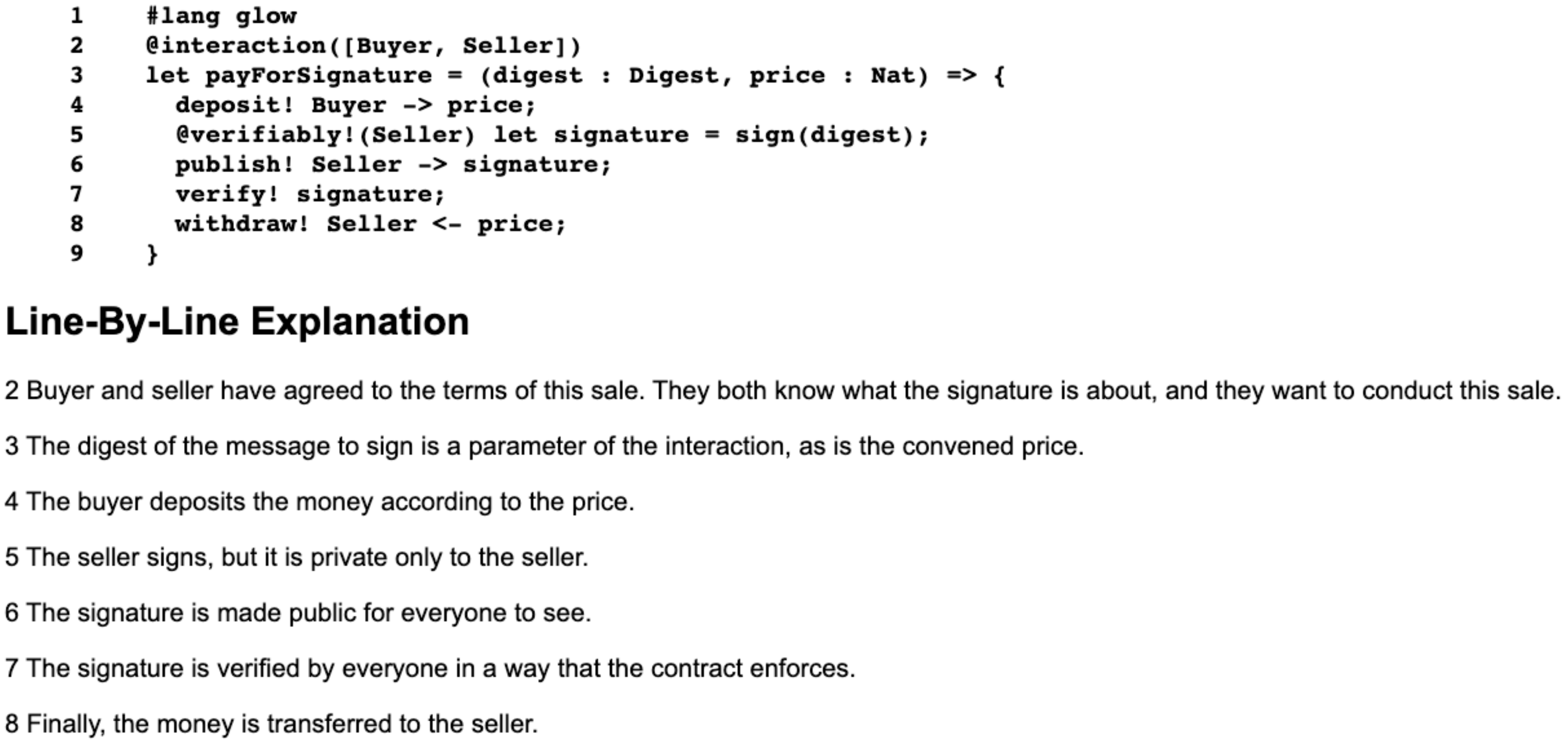
Buy signature interactions

Glow Contract



```
#lang glow
@interaction([Buyer, Seller])
let payForSignature = (digest : Digest, price : Nat) => {
  deposit! Buyer -> price;
  @verifiably!(Seller) let signature = sign(digest);
  publish! Seller -> signature;
  verify! signature;
  withdraw! Seller <- price;
}
```

Glow Contract



```
1    #lang glow
2    @interaction([Buyer, Seller])
3    let payForSignature = (digest : Digest, price : Nat) => {
4      deposit! Buyer -> price;
5      @verifiably!(Seller) let signature = sign(digest);
6      publish! Seller -> signature;
7      verify! signature;
8      withdraw! Seller <- price;
9    }
```

Line-By-Line Explanation

2 Buyer and seller have agreed to the terms of this sale. They both know what the signature is about, and they want to conduct this sale.

3 The digest of the message to sign is a parameter of the interaction, as is the convened price.

4 The buyer deposits the money according to the price.

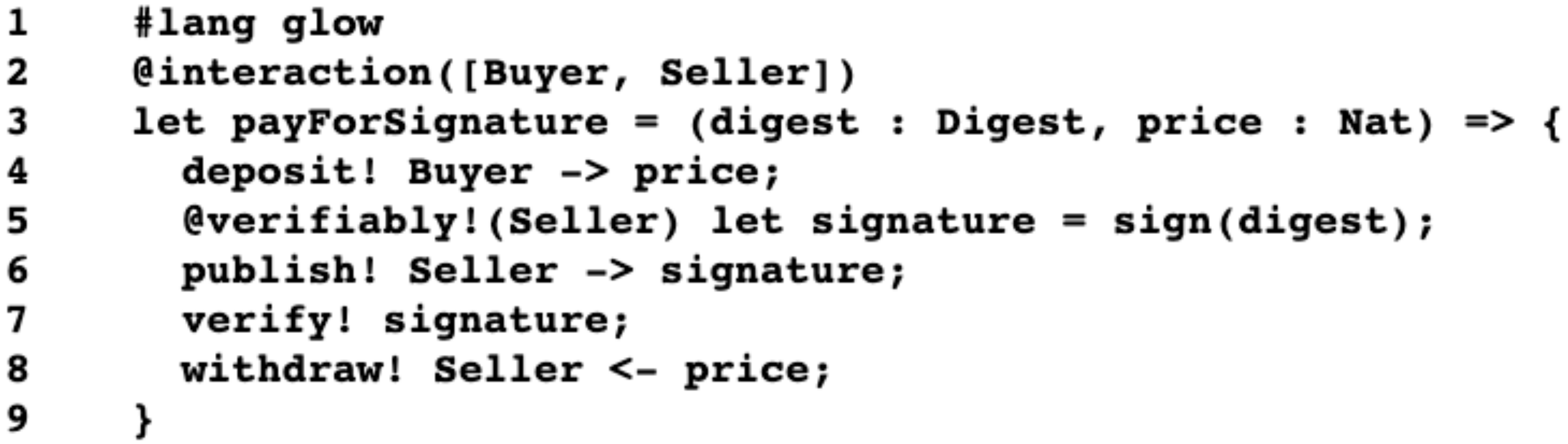
5 The seller signs, but it is private only to the seller.

6 The signature is made public for everyone to see.

7 The signature is verified by everyone in a way that the contract enforces.

8 Finally, the money is transferred to the seller.

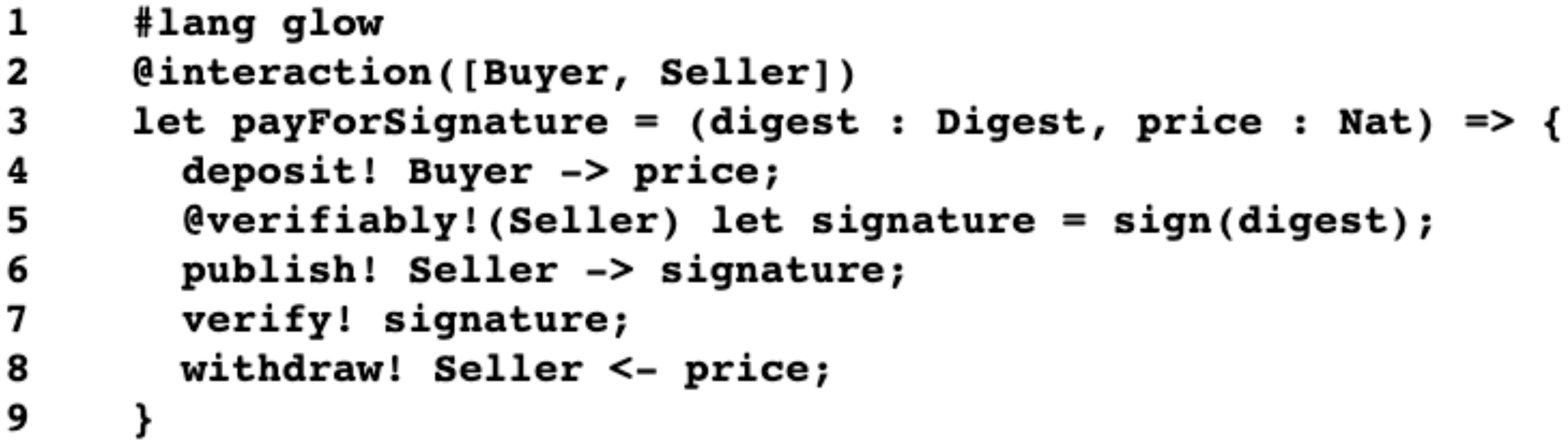
Failure Cases



```
1    #lang glow
2    @interaction([Buyer, Seller])
3    let payForSignature = (digest : Digest, price : Nat) => {
4      deposit! Buyer -> price;
5      @verifiably!(Seller) let signature = sign(digest);
6      publish! Seller -> signature;
7      verify! signature;
8      withdraw! Seller <- price;
9    }
```

- **Buyer Never Pays**
  - Process stops at line 4, so seller never creates signature. The interaction times out and is cancelled. No funds are exchanged.

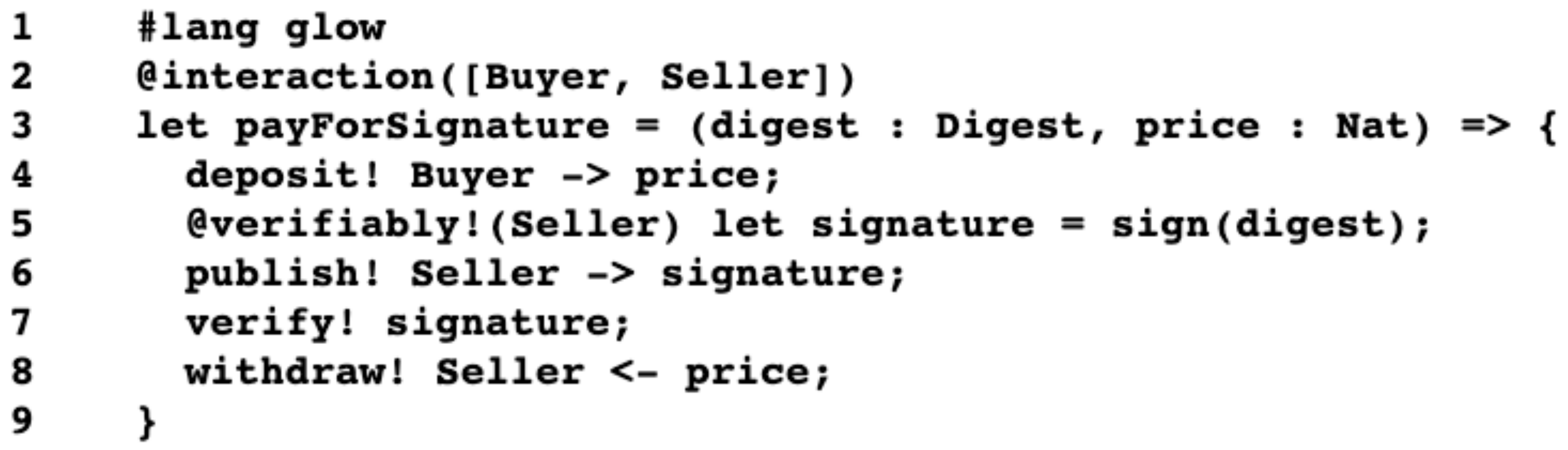
Failure Cases



```
1    #lang glow
2    @interaction([Buyer, Seller])
3    let payForSignature = (digest : Digest, price : Nat) => {
4      deposit! Buyer -> price;
5      @verifiably!(Seller) let signature = sign(digest);
6      publish! Seller -> signature;
7      verify! signature;
8      withdraw! Seller <- price;
9    }
```

- **Buyer Pays, but Seller Never Signs**
    - At line 4, the payment is deposited (into escrow).
    - The process stops at line 5.
    - The interaction times out and the funds are returned to the Buyer.

Failure Cases



```
1    #lang glow
2    @interaction([Buyer, Seller])
3    let payForSignature = (digest : Digest, price : Nat) => {
4      deposit! Buyer -> price;
5      @verifiably!(Seller) let signature = sign(digest);
6      publish! Seller -> signature;
7      verify! signature;
8      withdraw! Seller <- price;
9    }
```

- **Seller Sends Invalid Signature**
  - Process stops at line 7 because the signature does not validate.
  - The interaction times out and the funds are returned to the Buyer.

Questions