Ethereum and Smart Contracts

Thierry Sans

The Bitcoin Inspiration

What if you could program money and decentralized logic into a blockchain?

Ethereum in a Nutshell

- Uses Elliptic Curve Public Keys (secp256k1) and ECDSA signature algorithm
- Consensus :
  (before) Proof of Work
  (since 2022) Proof of Stake
- Block time : ~12 seconds
- `ETH` are created through staking rewards and transaction fees
- Account-based blockchain **+ programmable smart contracts**

➡ Not just a cryptocurrency
it's a decentralized computing platform to automate trustless transactions
a.k.a decentralized applications (dApp)

Ethereum Accounts & Transactions

Different types of Ethereum **accounts** (all associated with an address):

- Externally Owned Accounts (EOAs)
- Contract Accounts
- Account abstractions (newest - will be covered in another lecture)

Different types of Ethereum **transactions**

- transfer ETH from EOA accounts to Ethereum addresses
- deploy smart contracts
- call methods of a deployed smarts

Smart Contracts

What is a smart contract?
A computer program (EVM bytecode) deployed on the blockchain that defines 1) a set of state variables and 2) methods to read/write these state variables

Can a smart contract hold ETH?
Yes, a smart contract has an address and can hold ETH but there is no private key associated with that address

How to write a smart contract?
Either write an EVM bytecode program directly
or use a high-level language (e.g. *Solidity*) that compiles programs into EVM bytecode

How to deploy a smart contract?
By sending a transaction that will write the EVM bytecode on the Ethereum blockchain

Can you change the code of a smart contract once deployed?
(short answer) no, the code is immutable
However, the contract state can change (by modifying contract state variables) when smart contract methods are called

How to call a method of a deployed smart contract?
Either directly using EOA account (sending a transaction) or from another contract

EVM code

What can the code do?

- Perform Arithmetic, Logical Operations, Bit Operations plus conditionals and loops (Turing complete)
- Store data (through contract state variables)
- Read transaction and block data
- Transfer ETH (held by the contract) to other another address
- Receive ETH (and execute some logic when funds are received)
- Call methods from other deployed contracts
- Emits events (logs that will be written on the blockchain)
- Self-destruct

Execution and Gas Fee

Who executes smart-contracts?
The Ethereum nodes that process transactions

When is the smart contract executed?

- When the transaction is received (unconfirmed mempool), the code is executed (by the node) but the contract's state is not modified (dry-run)
- When the transaction is confirmed (into a block), the code is executed (by the node chosen to confirmed the next block) and the contract's state is modified (i.e written to the blockchain)

➡ Deterministic execution: given the sequences of transaction and the blockchain state, the outcome can be determined

If the code has loops, how do we ensure that the execution will terminate?
In a nutshell, calling a smart contract method costs money (a.k.a gas). Whoever calls a smart contract method must pay some fee that will reward the node (selected to confirm the next block) for executing the smart contract

What happen when a method call fails or does not terminate because it runs out of gas?
The transaction is confirmed as a failed transaction. The contract state is not updated (full reverse) but the gas fee is not returned to the caller but kept by the node.

Gas Fee Calculation

$$\text{Total Fee} = (\text{Base Fee} + \text{Priority Fee}) \times \text{Gas Used}$$

- **Base Fee:** set by the protocol, dynamically adjusted based on network congestion
- **Priority Fee:** the tip paid to miners/validators as an incentive to prioritize the transaction
- **Gas Used:** the amount of gas consumed by the smart contract execution
  Each operation (storage, computation, external calls) consumes gas
  Examples :
    - Writing a new storage variable: 20,000 gas
    - Modifying an existing storage variable: 5,000 gas
    - Simple arithmetic operation: ~3 gas
    - Sending ETH: ~21,000 gas

➡ If the caller supplies more gas than actually needed, the excess of gas is refunded once the transaction processed

In summary

In summary, what data is written onto the Ethereum blockchain?

Transactions, smart contract code, smart contract state variables and events

Why are smart-contracts useful?

Automates agreements without intermediaries by enabling trustless transactions

Examples of dApps

- Payment Automation
- Tokens (Fungible) including Stablecoins, NFTs (Non-Fungible) and RWAs (Real-World Assets)
- Funds and Assets Management
- Decentralized Exchanges and Decentralized Marketplaces
- Lending and Borrowing Platforms
- Insurance and Derivatives
- Governance (DAO - Decentralized Autonomous Organization)
- Supply Chain Management

Benefits and Risks of Smart Contracts

Benefits:

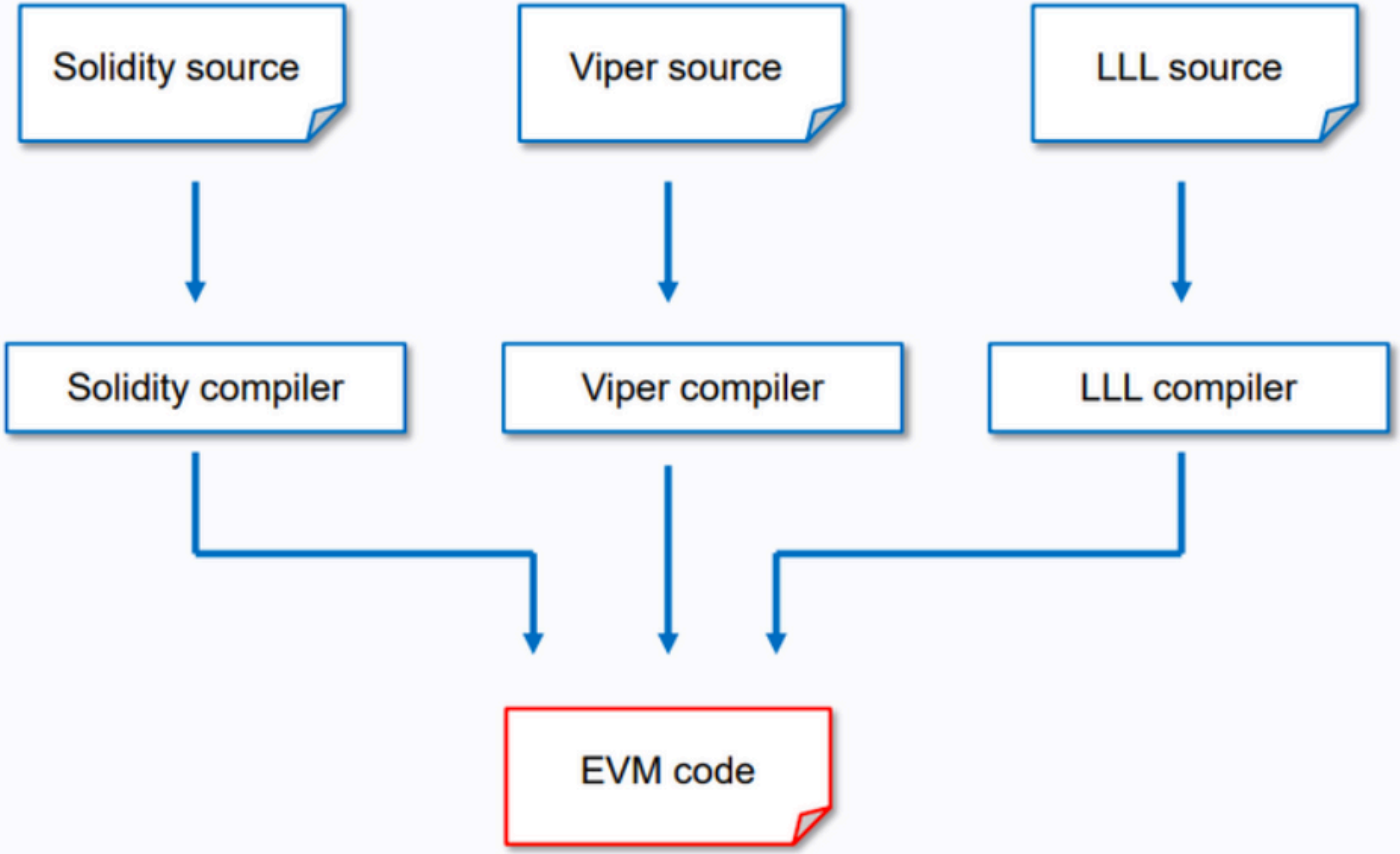
- Trustlessness
- Automation
- Transparency

Risks:

- Security vulnerabilities
- Immutable bugs
- Gas cost considerations

Solidity

EVM code generation



Solidity source

Viper source

LLL source

Solidity compiler

Viper compiler

LLL compiler

EVM code

Ethereum virtual machine code

Introduction to Solidity

High-level, contract-oriented language for Ethereum

Evolved alongside Ethereum to meet dApp needs (through EIP)

Similar in syntax to JavaScript/C++

Solidity Language Constructs

Data types

uint, address, bool, string, byte, enum, struct, array, mapping, ...

Structure

state variables, functions, events, modifiers

Code organization

contracts, inheritance, libraries, interfaces

Development Tools for Solidity

Remix IDE for quick prototyping

Frameworks such as Truffle, **Hardhat**, Foundry for development and testing

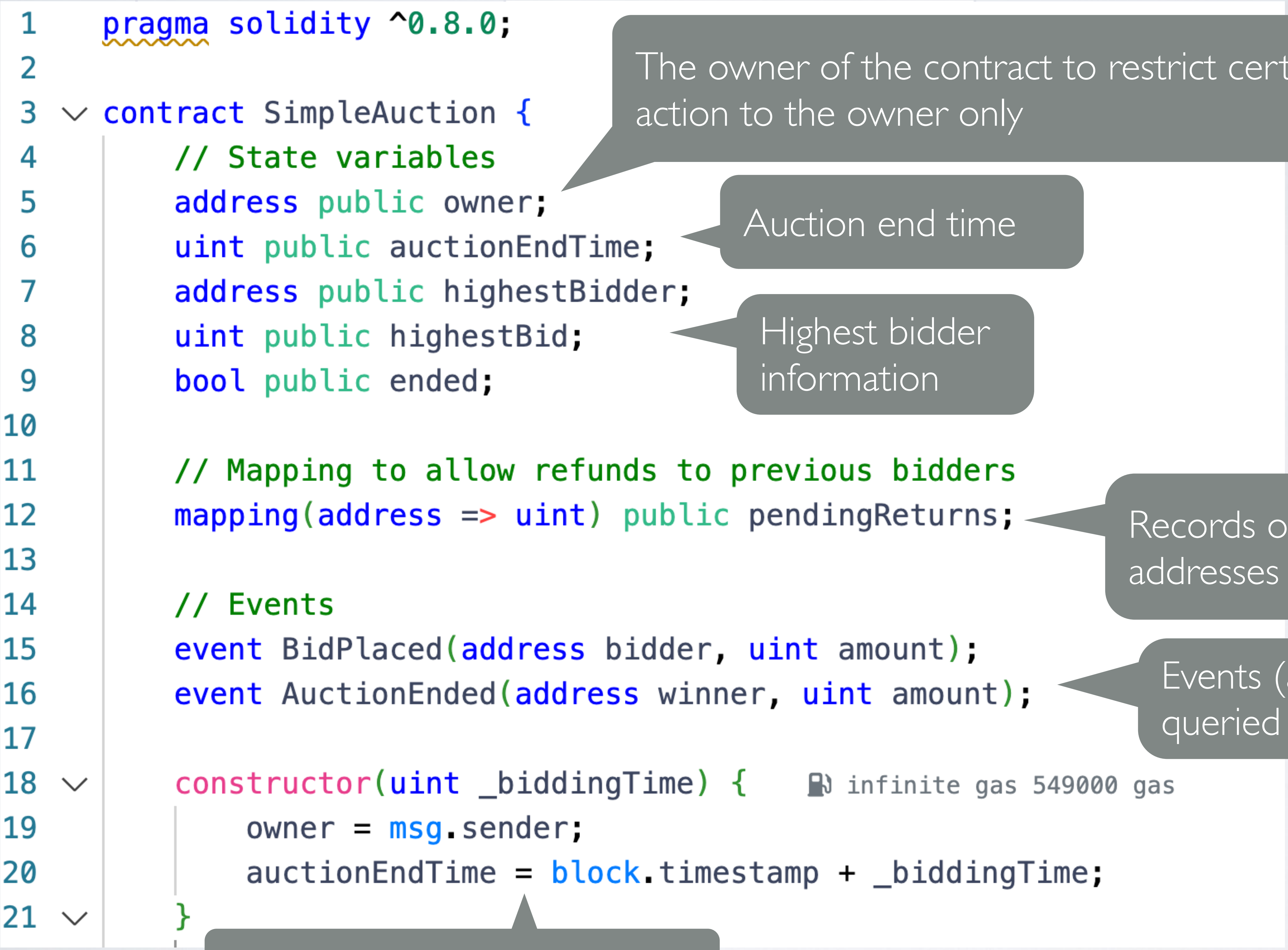
Example

Requirements for a Simple Auction Contract

An auction dApp:

- Allow an admin to create an auction and control the timing (start/end bid)
- Allow users to place bids by depositing ETH onto the auction contract
- Allow users to withdraw their funds if they were outbidded
- Allow the admin to transfer the highest-bid price at the end of the auction

Code Walkthrough – Contract Setup



```solidity
pragma solidity ^0.8.0;

contract SimpleAuction {
    // State variables
    address public owner;
    uint public auctionEndTime;
    address public highestBidder;
    uint public highestBid;
    bool public ended;

    // Mapping to allow refunds to previous bidders
    mapping(address => uint) public pendingReturns;

    // Events
    event BidPlaced(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    constructor(uint _biddingTime) {    infinite gas 549000 gas
        owner = msg.sender;
        auctionEndTime = block.timestamp + _biddingTime;
    }
```

The owner of the contract to restrict certain action to the owner only

Auction end time

Highest bidder information

Records outbidded addresses and amounts

Events (a.k.a logs) that can be queried by a client

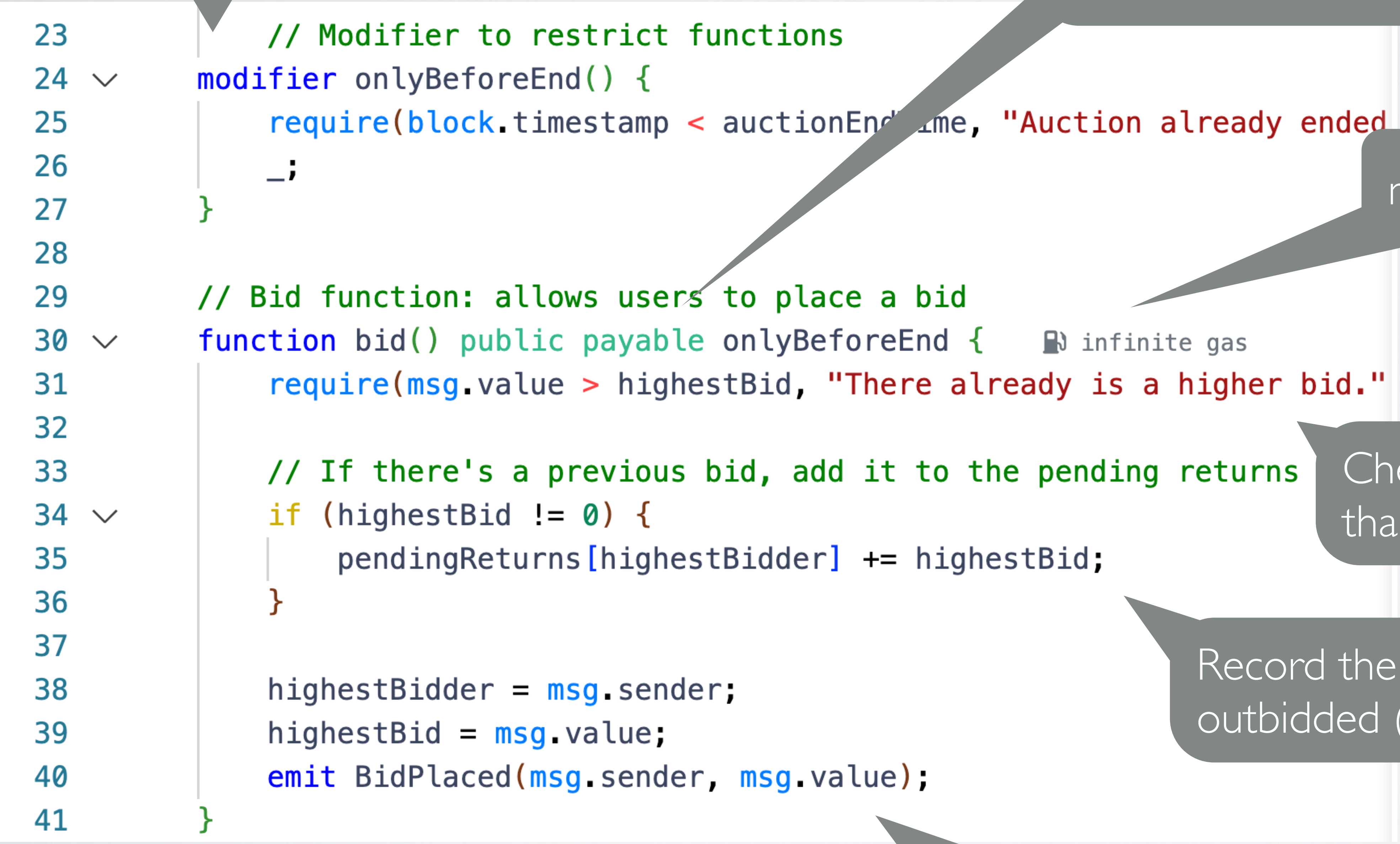
The constructor is called when the contract is deployed

Code Walkthrough – The Bidding Function

Code that can run before (or after) a function

allows someone to send ETH to the contract when calling the function (deposit)

modifier is called here



```
    // Modifier to restrict functions
modifier onlyBeforeEnd() {
    require(block.timestamp < auctionEndTime, "Auction already ended
    _;
}

// Bid function: allows users to place a bid
function bid() public payable onlyBeforeEnd {
    require(msg.value > highestBid, "There already is a higher bid."

    // If there's a previous bid, add it to the pending returns
    if (highestBid != 0) {
        pendingReturns[highestBidder] += highestBid;
    }

    highestBidder = msg.sender;
    highestBid = msg.value;
    emit BidPlaced(msg.sender, msg.value);
}
```

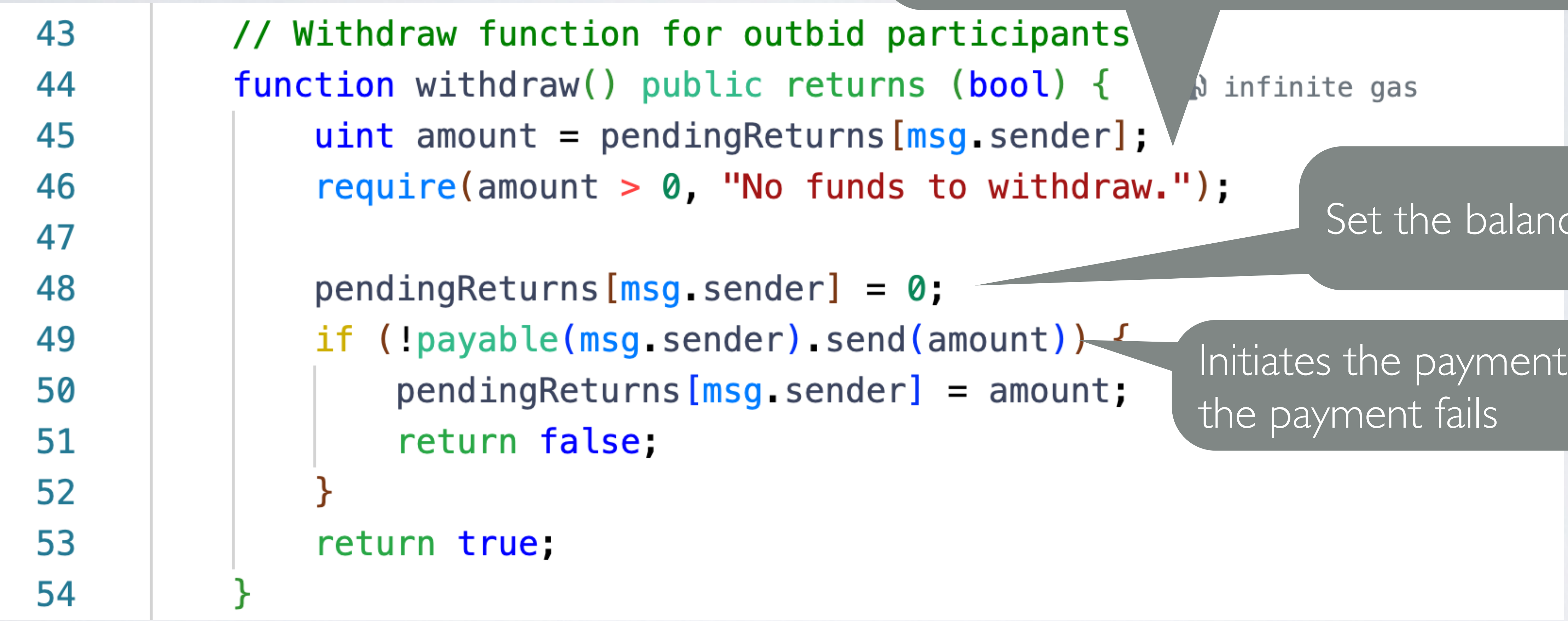
Check if the deposit is higher than current highest bid

Record the previous bidder as outbidded (to allow refund)

Record the new bidder as the highest bidder and emit an event

Code Walkthrough – Refunds

Checks if caller's address has been recorded in the outbidded mapping and that the balance is positive

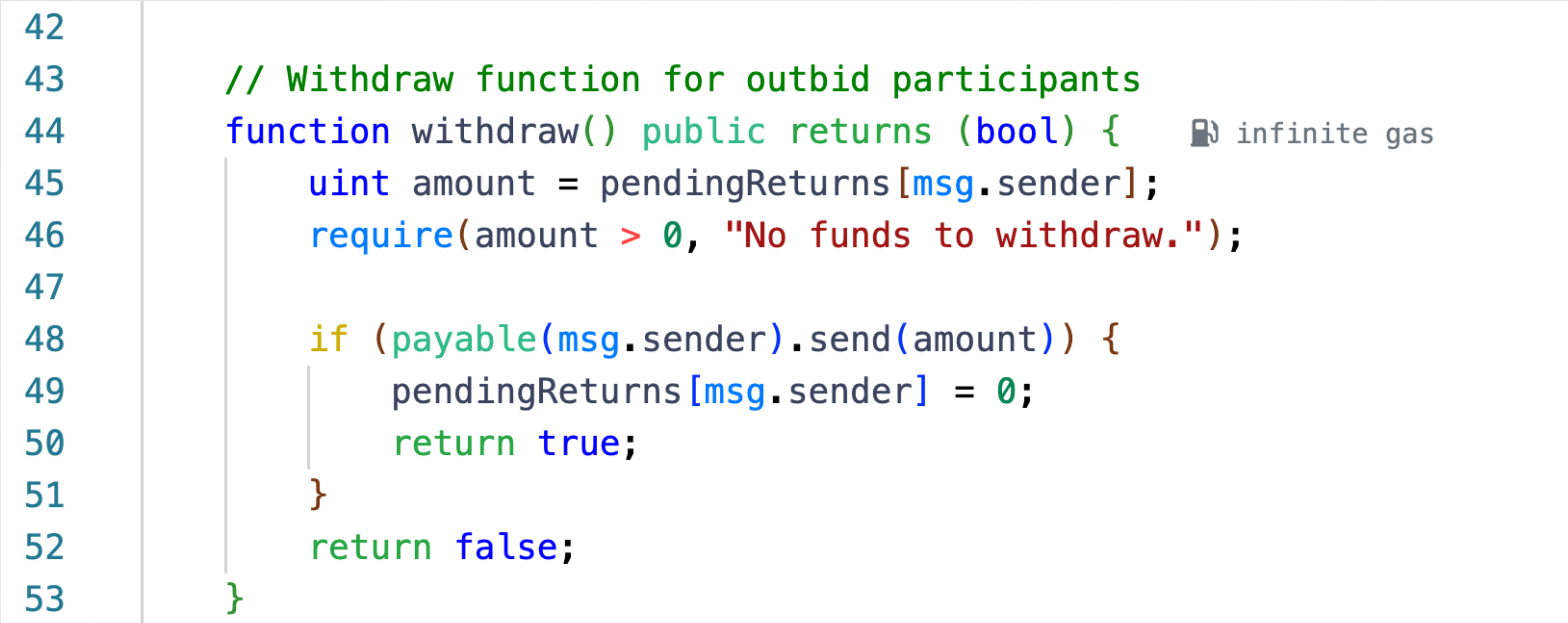


```
    // Withdraw function for outbid participants
    function withdraw() public returns (bool) {      infinite gas
        uint amount = pendingReturns[msg.sender];
        require(amount > 0, "No funds to withdraw.");

        pendingReturns[msg.sender] = 0;
        if (!payable(msg.sender).send(amount)) {
            pendingReturns[msg.sender] = amount;
            return false;
        }
        return true;
    }
```

Set the balance to 0

Initiates the payment and revert if the payment fails

.... why not simplifying the code

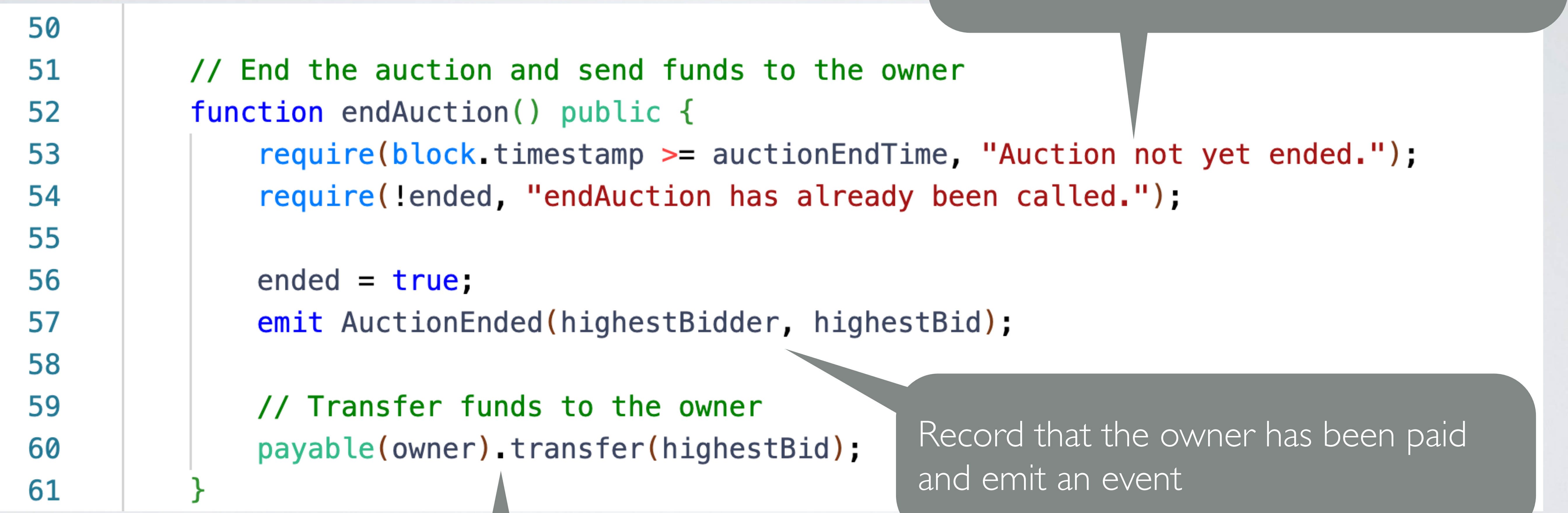


```
    // Withdraw function for outbid participants
    function withdraw() public returns (bool) {
        uint amount = pendingReturns[msg.sender];
        require(amount > 0, "No funds to withdraw.");

        if (payable(msg.sender).send(amount)) {
            pendingReturns[msg.sender] = 0;
            return true;
        }
        return false;
    }
```

➡ This is huge vulnerability (called reentrency attack) that would allow the attacker to withdraw all funds from the contract (more later in the "smart contract security" lecture)

Code Walkthrough
– Withdrawing at the end of the auction

Check that the auction has ended and that owner has not been paid yet

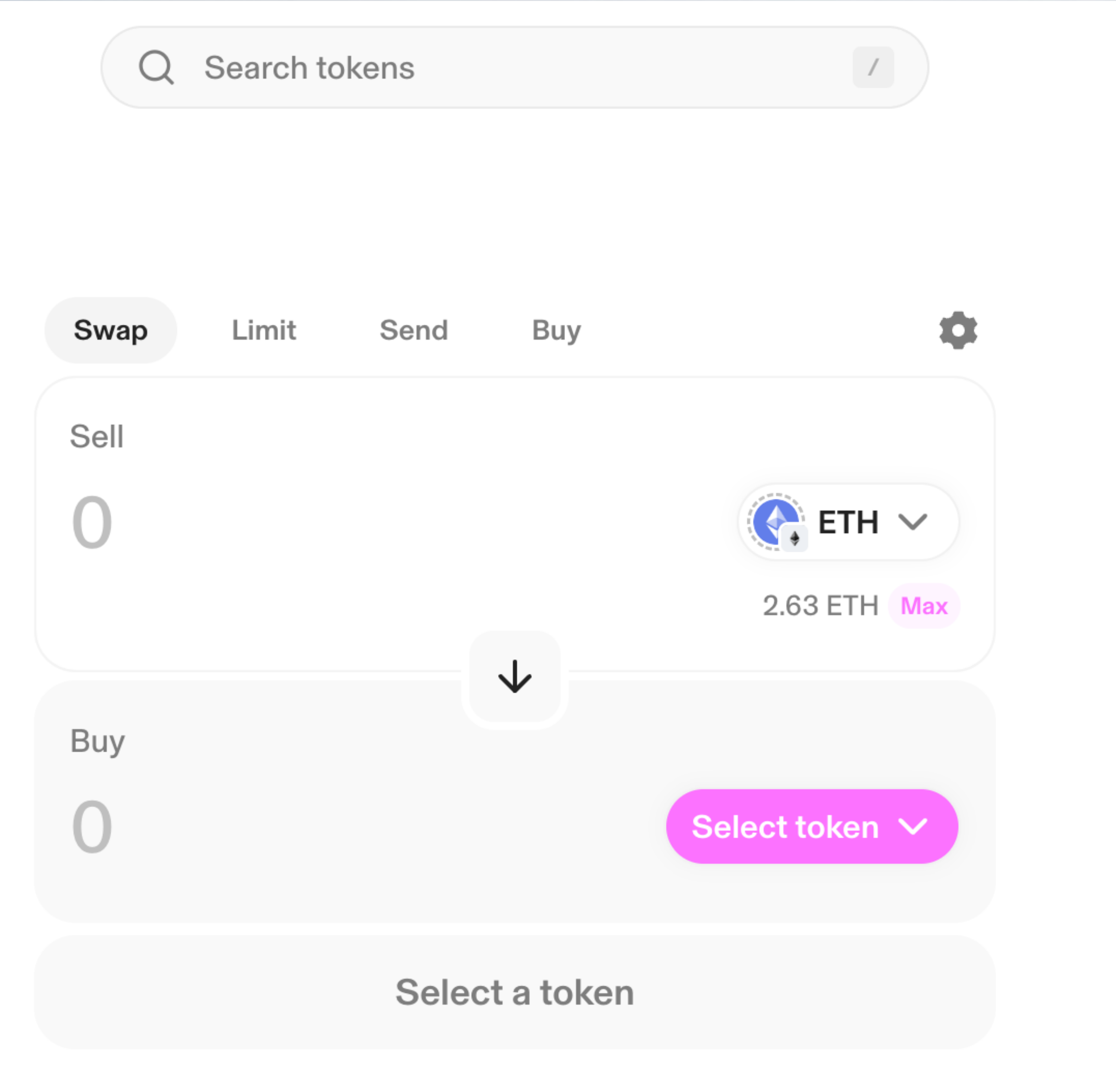


```
    // End the auction and send funds to the owner
    function endAuction() public {
        require(block.timestamp >= auctionEndTime, "Auction not yet ended.");
        require(!ended, "endAuction has already been called.");

        ended = true;
        emit AuctionEnded(highestBidder, highestBid);

        // Transfer funds to the owner
        payable(owner).transfer(highestBid);
    }
```

Record that the owner has been paid and emit an event

The highest bid amount is transferred to the owner

Anatomy of a dApp



Search tokens /

0x3b72...0624

Swap Limit Send Buy

Sell

0

ETH

2.63 ETH Max

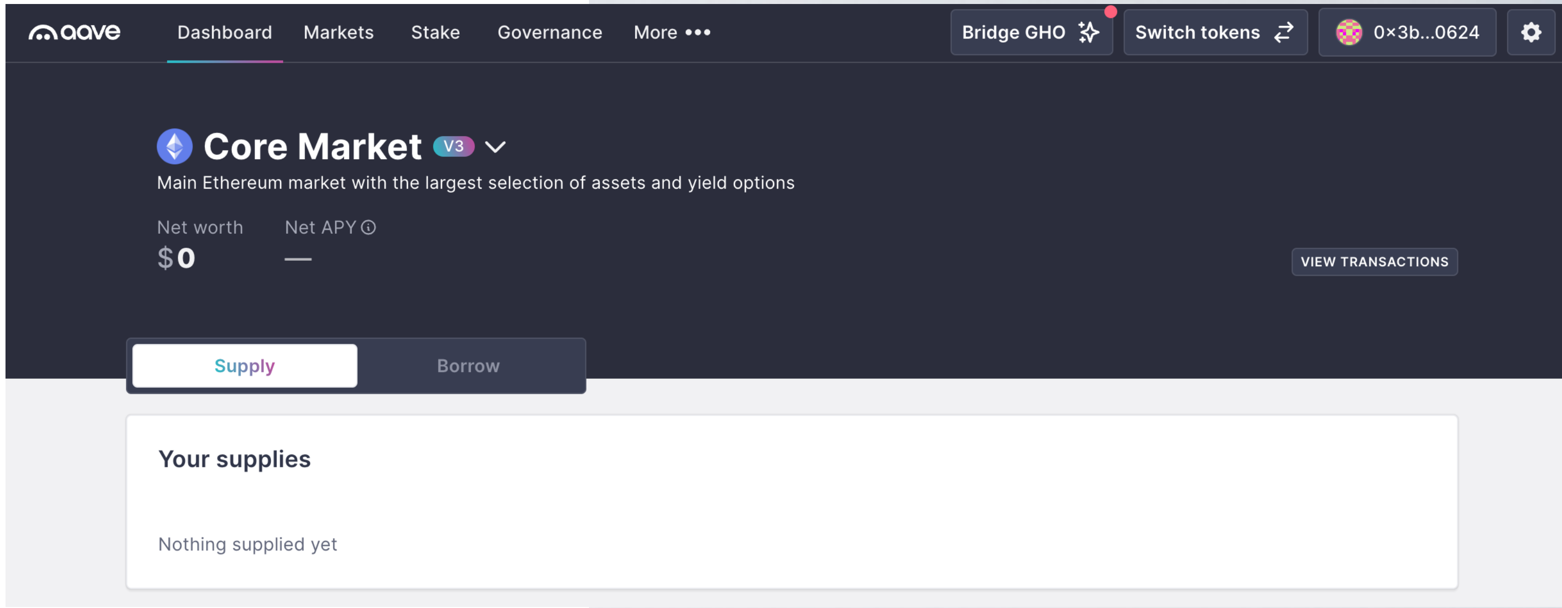
Buy

0

Select token

Select a token

Aave (Lending and Borowing)



aave Dashboard Markets Stake Governance More •••

Bridge GHO Switch tokens 0x3b...0624

Core Market V3

Main Ethereum market with the largest selection of assets and yield options

Net worth $0

Net APY —

VIEW TRANSACTIONS

Supply Borrow

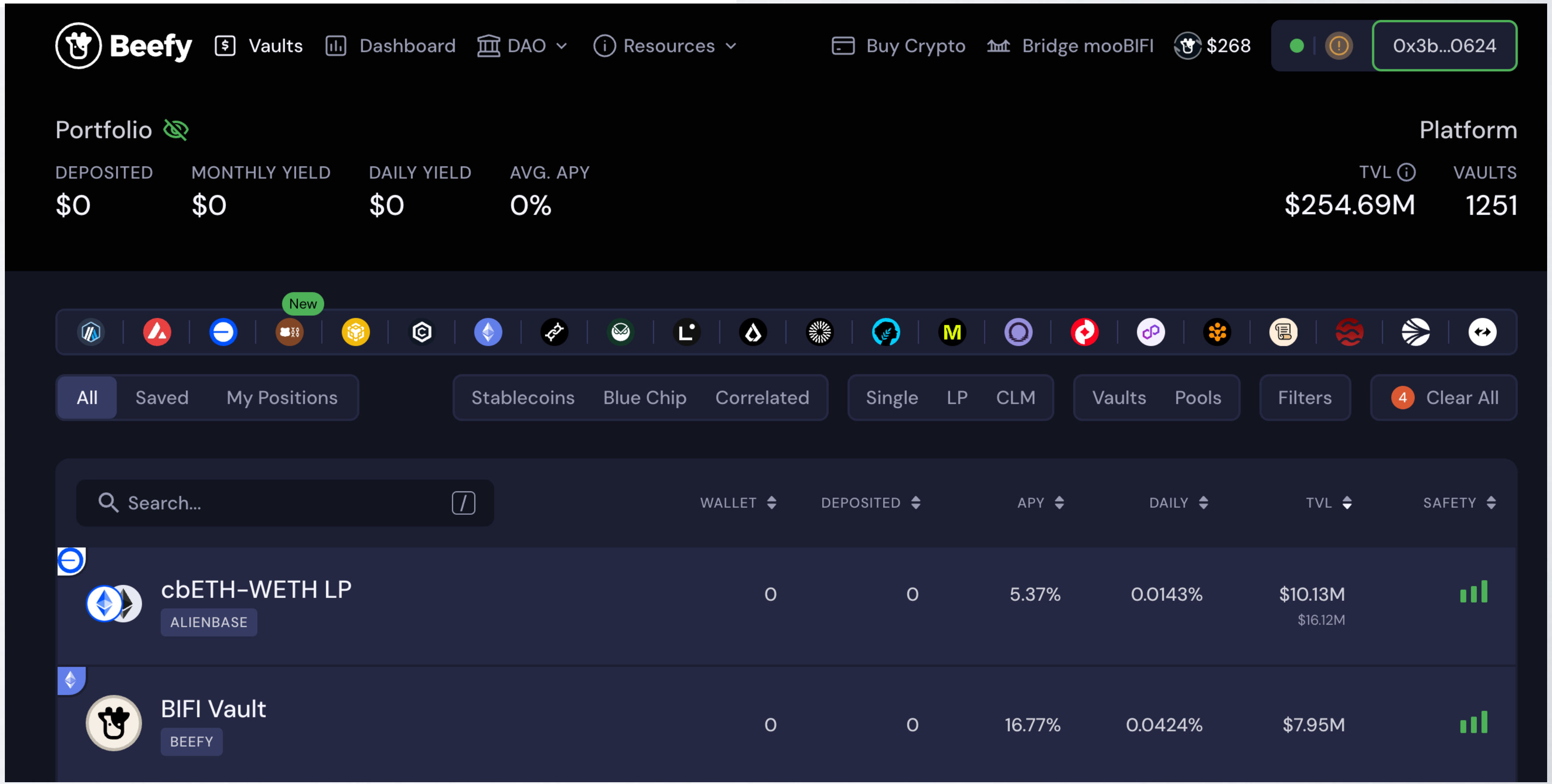
Your supplies

Nothing supplied yet

Beefy Finance (Yield Farming)

Uniswap (DEX)

Examples



Beefy Vaults Dashboard DAO Resources Buy Crypto Bridge mooBIFI $268 0x3b...0624

Portfolio

DEPOSITED	MONTHLY YIELD	DAILY YIELD	AVG. APY
$0	$0	$0	0%

Platform

TVL	VAULTS
$254.69M	1251

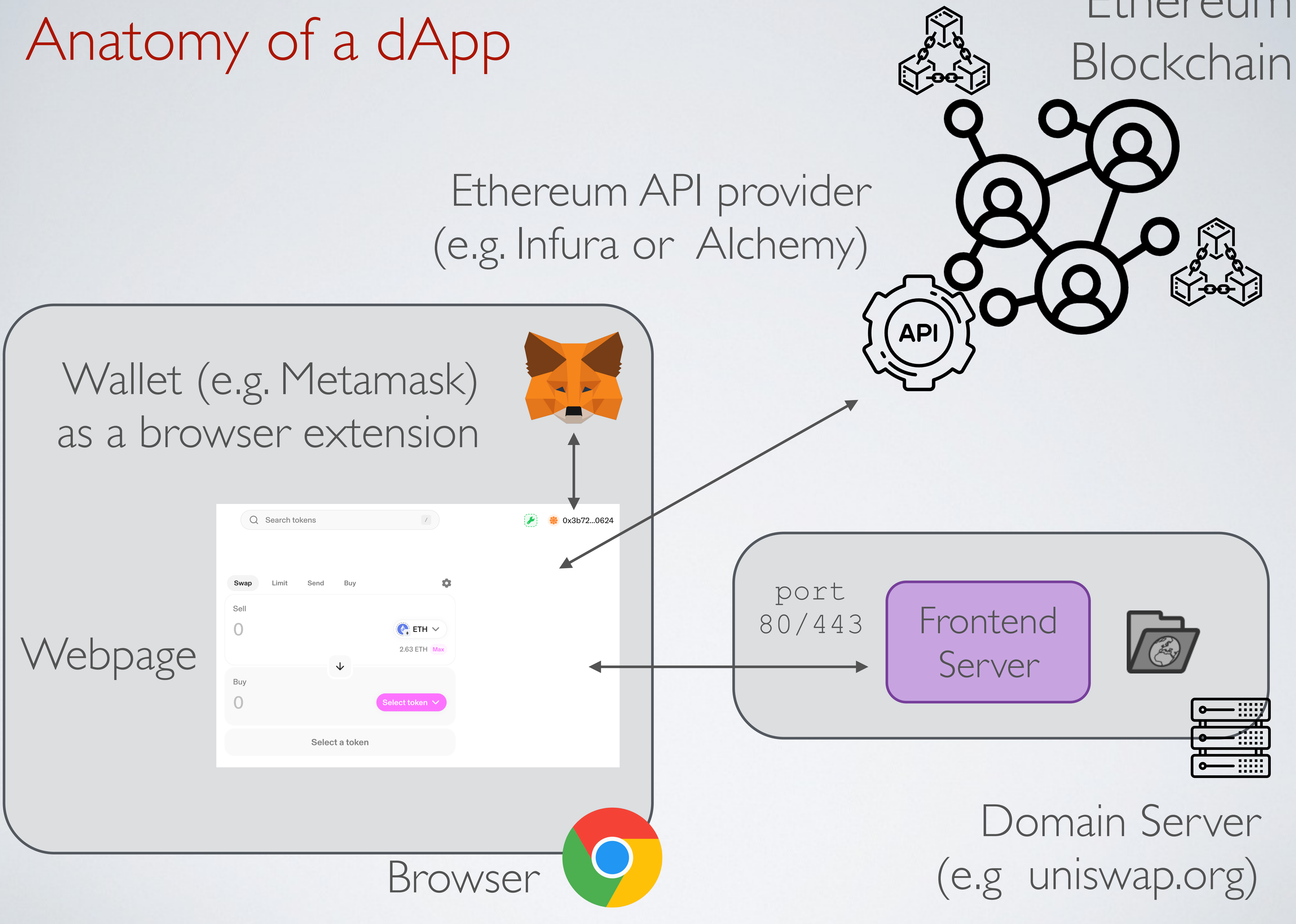
New

All Saved My Positions Stablecoins Blue Chip Correlated Single LP CLM Vaults Pools Filters 4 Clear All

Search... /

	WALLET	DEPOSITED	APY	DAILY	TVL	SAFETY
cbETH-WETH LP ALIENBASE	0	0	5.37%	0.0143%	$10.13M $16.12M	
BIFI Vault BEEFY	0	0	16.77%	0.0424%	$7.95M	

Anatomy of a dApp



Ethereum Blockchain

Ethereum API provider
(e.g. Infura or Alchemy)

Wallet (e.g. Metamask)
as a browser extension

Webpage

Browser

port
80/443

Frontend
Server

Domain Server
(e.g uniswap.org)

Development Lifecycle

1. Local Development Chain
   using hardhat chain running locally

2. Deploy on `Testnet` chain
   either using a real testnet (e.g *Sepolia*)
   or cloud-based testnets (*BuildBear* for instance)

3. Deploy on `Mainnet` chain
   (either Ethereum, Binance, Base, Polygon and so on)