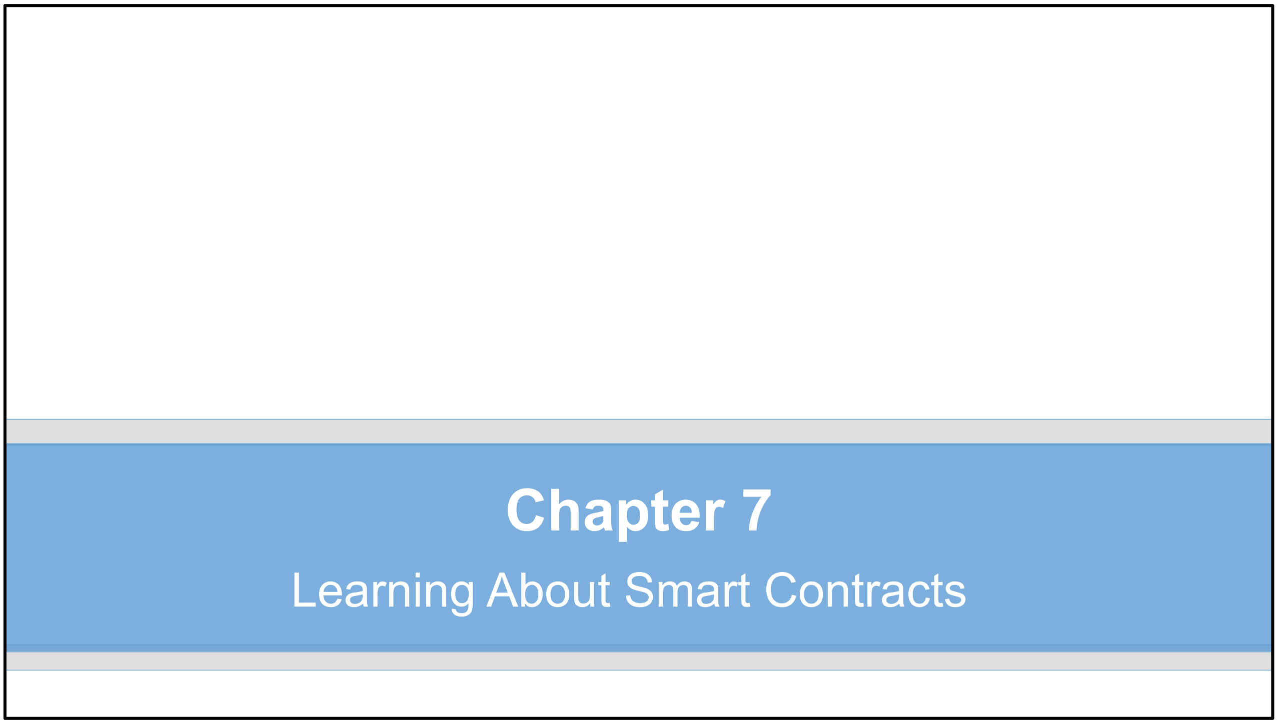
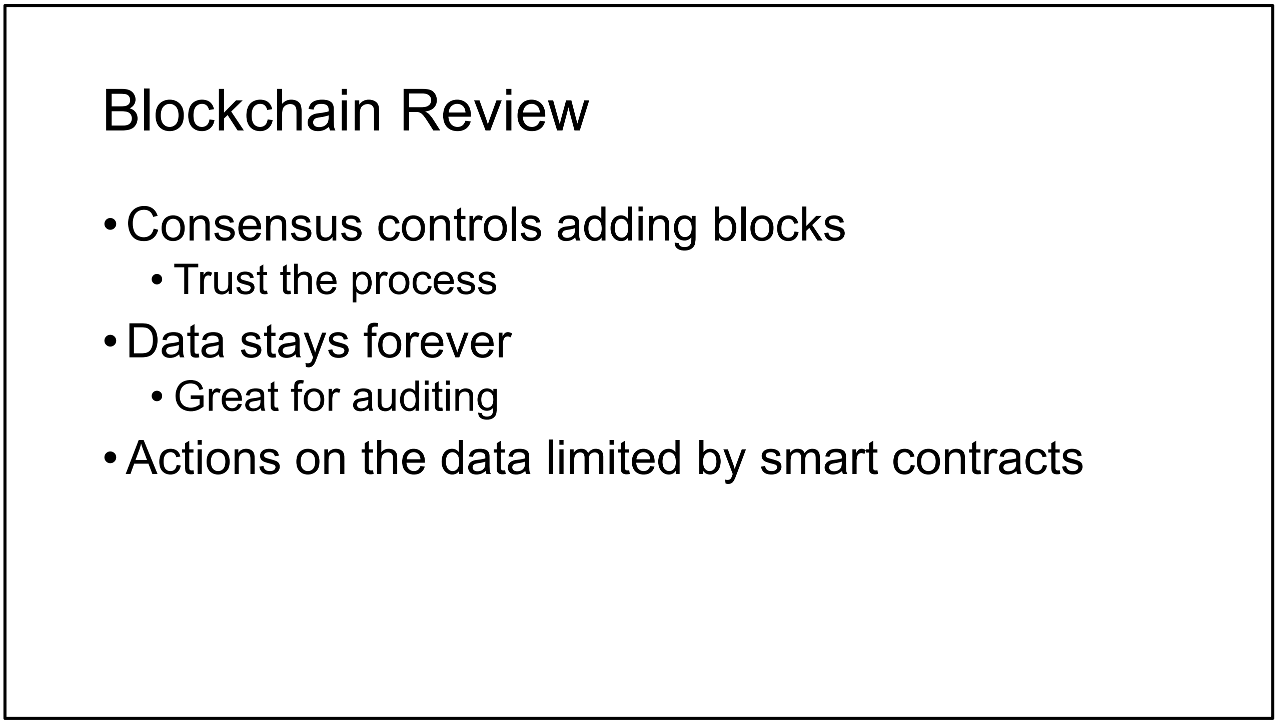


Chapter 7

Learning About Smart Contracts

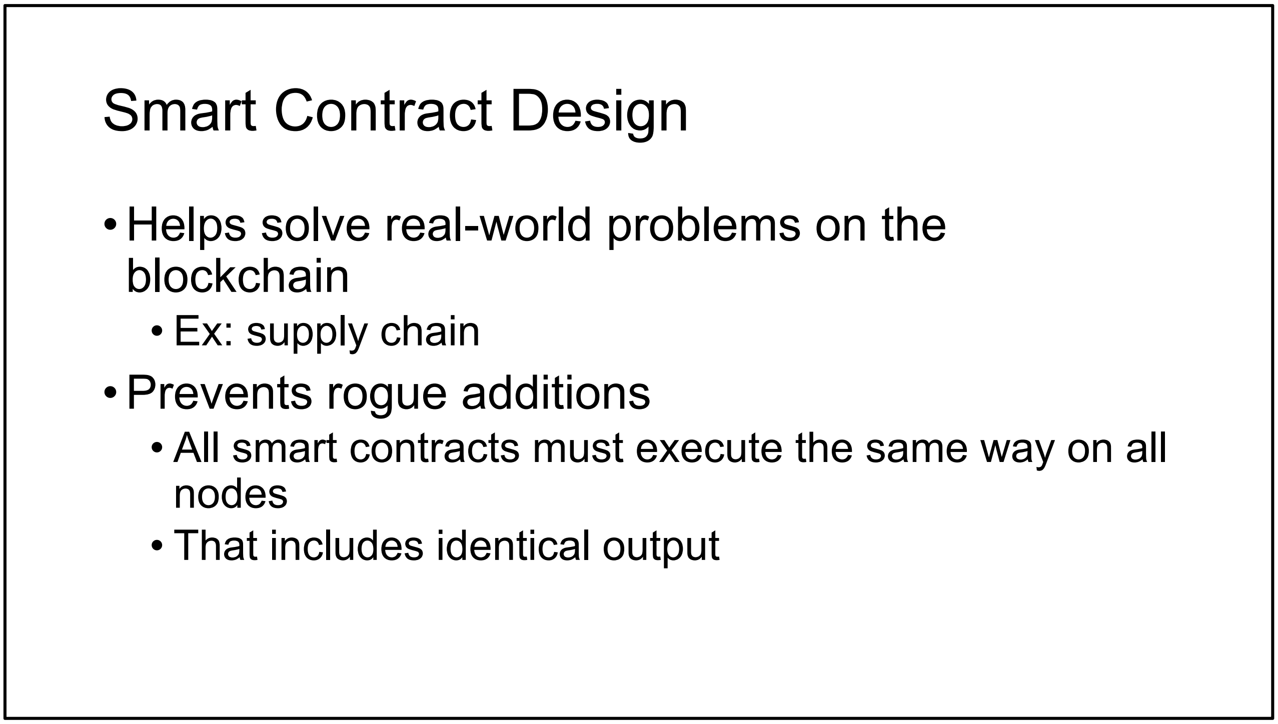
Episode 7.01

Smart Contracts Review



Blockchain Review

- Consensus controls adding blocks
  - Trust the process
- Data stays forever
  - Great for auditing
- Actions on the data limited by smart contracts

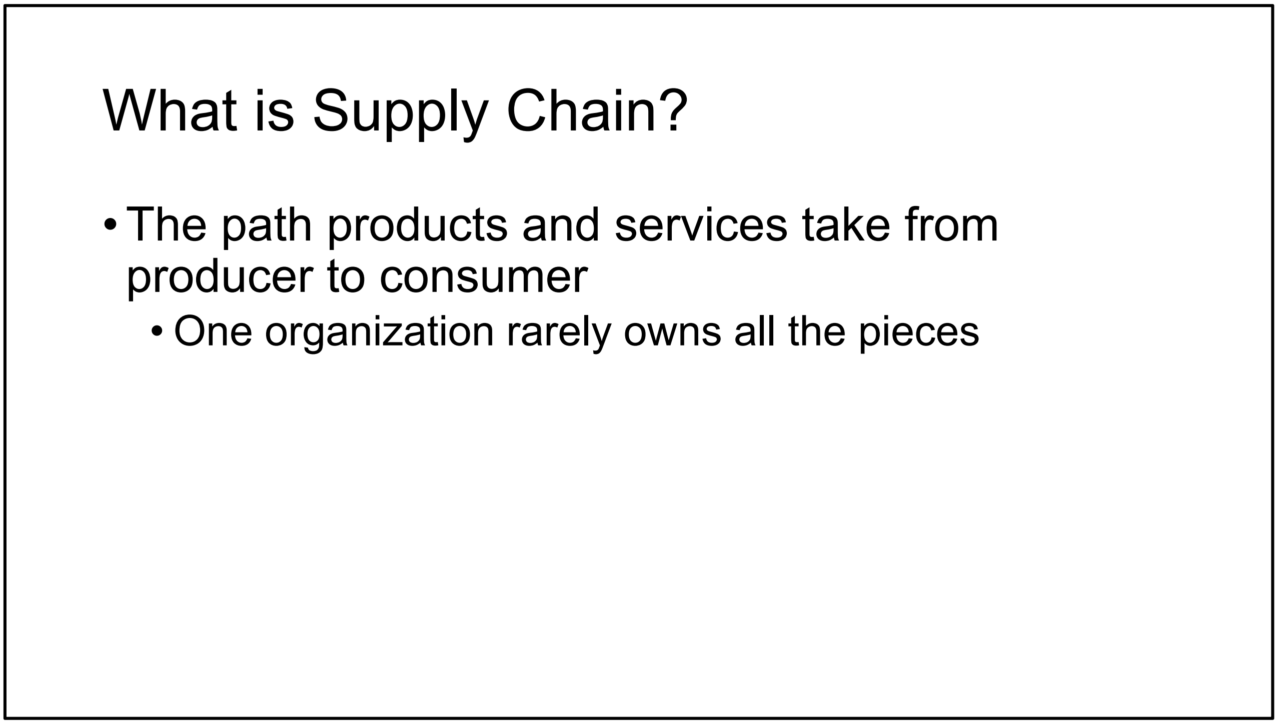


Smart Contract Design

- Helps solve real-world problems on the blockchain
  - Ex: supply chain
- Prevents rogue additions
  - All smart contracts must execute the same way on all nodes
  - That includes identical output

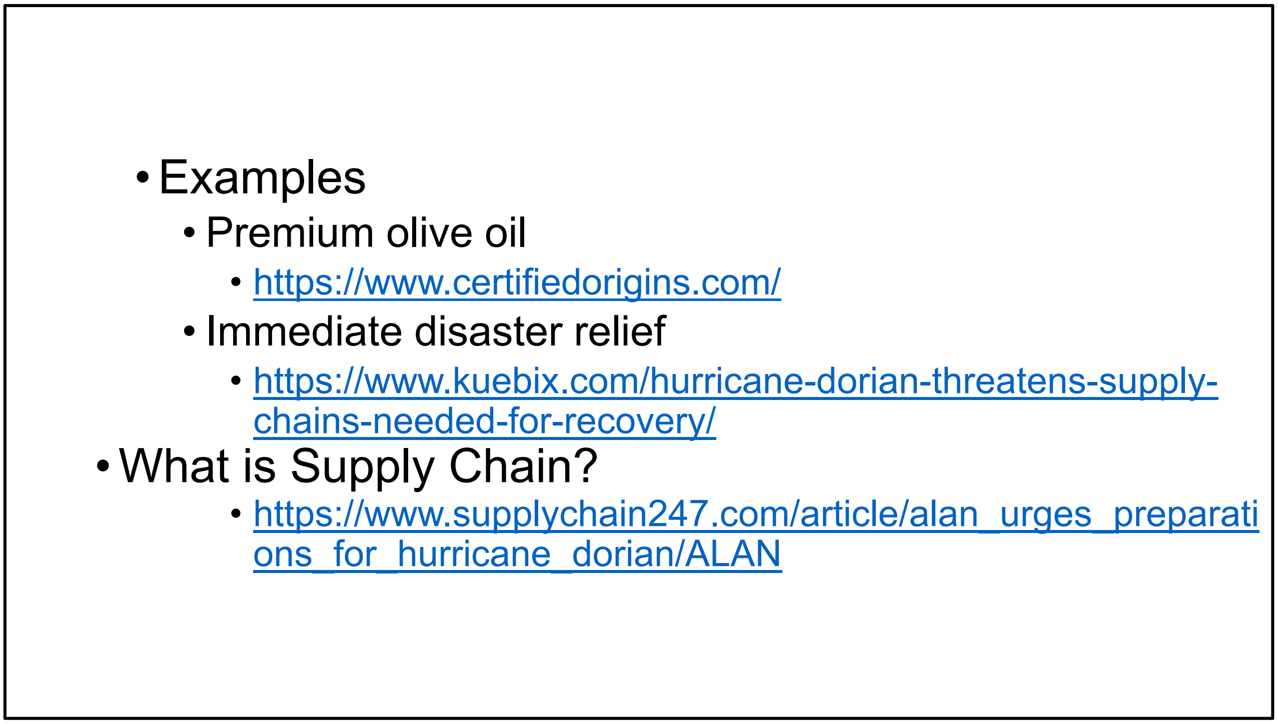
Episode 7.02

What is Supply Chain?



What is Supply Chain?

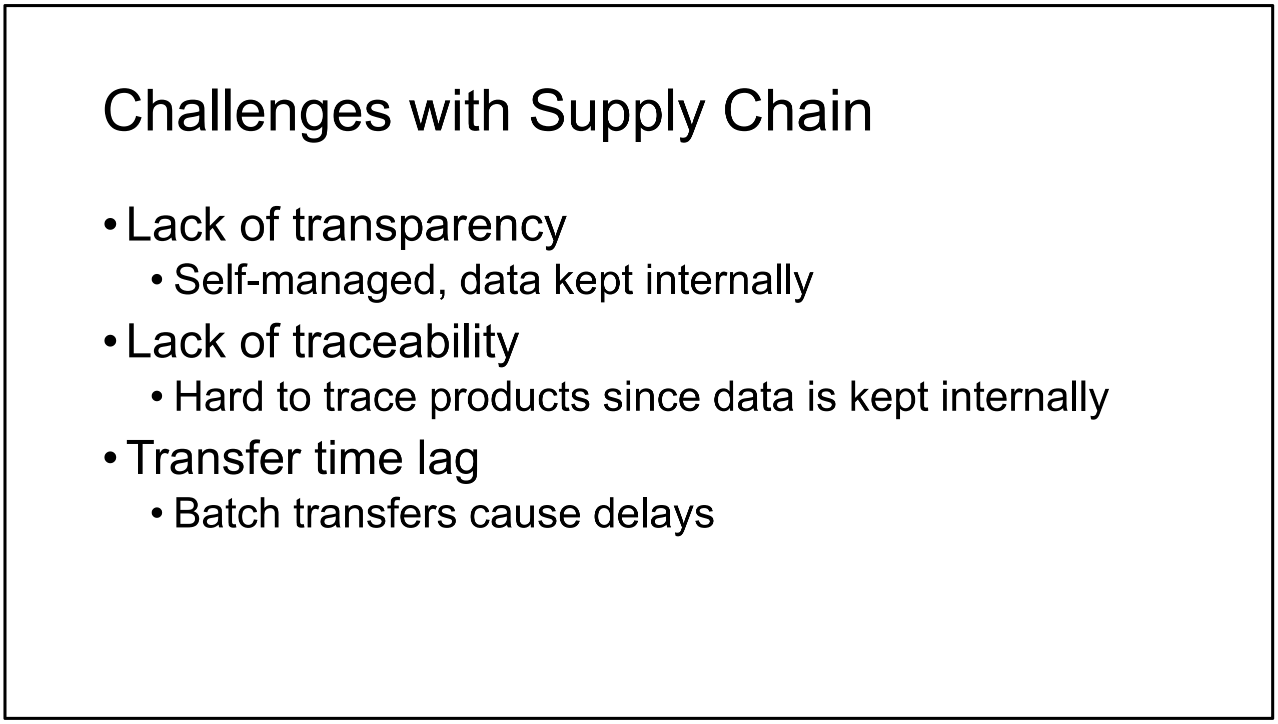
- The path products and services take from producer to consumer
  - One organization rarely owns all the pieces



- Examples
  - Premium olive oil
    - https://www.certifiedorigins.com/
  - Immediate disaster relief
    - https://www.kuebix.com/hurricane-dorian-threatens-supply-chains-needed-for-recovery/
- What is Supply Chain?
    - https://www.supplychain247.com/article/alan_urges_preparations_for_hurricane_dorian/ALAN

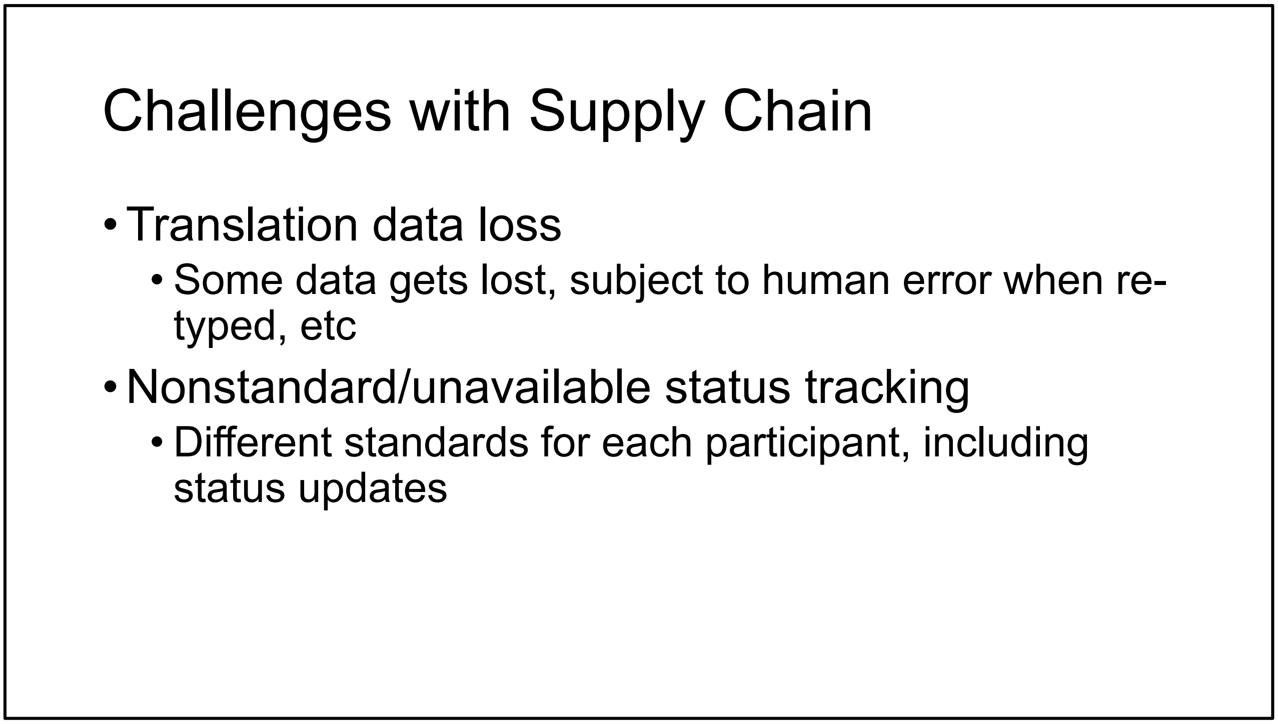
Episode 7.03

Supply Chain Challenges and Blockchain Solutions



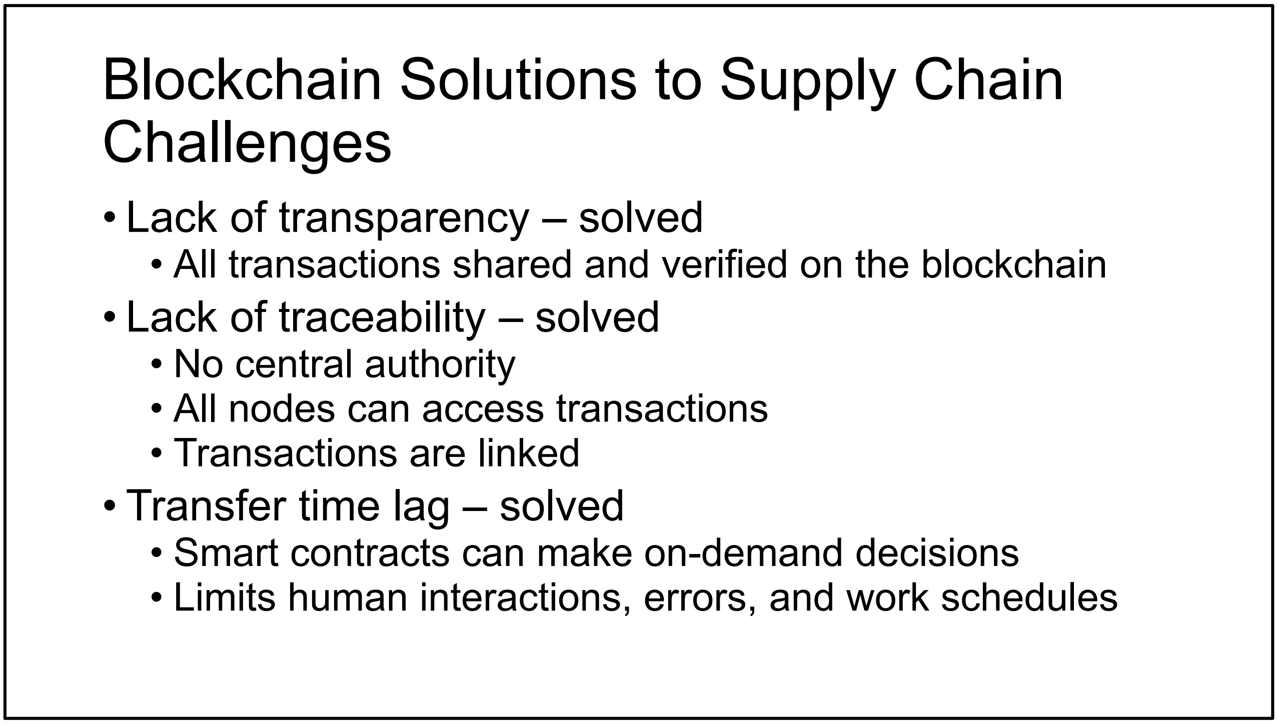
Challenges with Supply Chain

- Lack of transparency
  - Self-managed, data kept internally
- Lack of traceability
  - Hard to trace products since data is kept internally
- Transfer time lag
  - Batch transfers cause delays



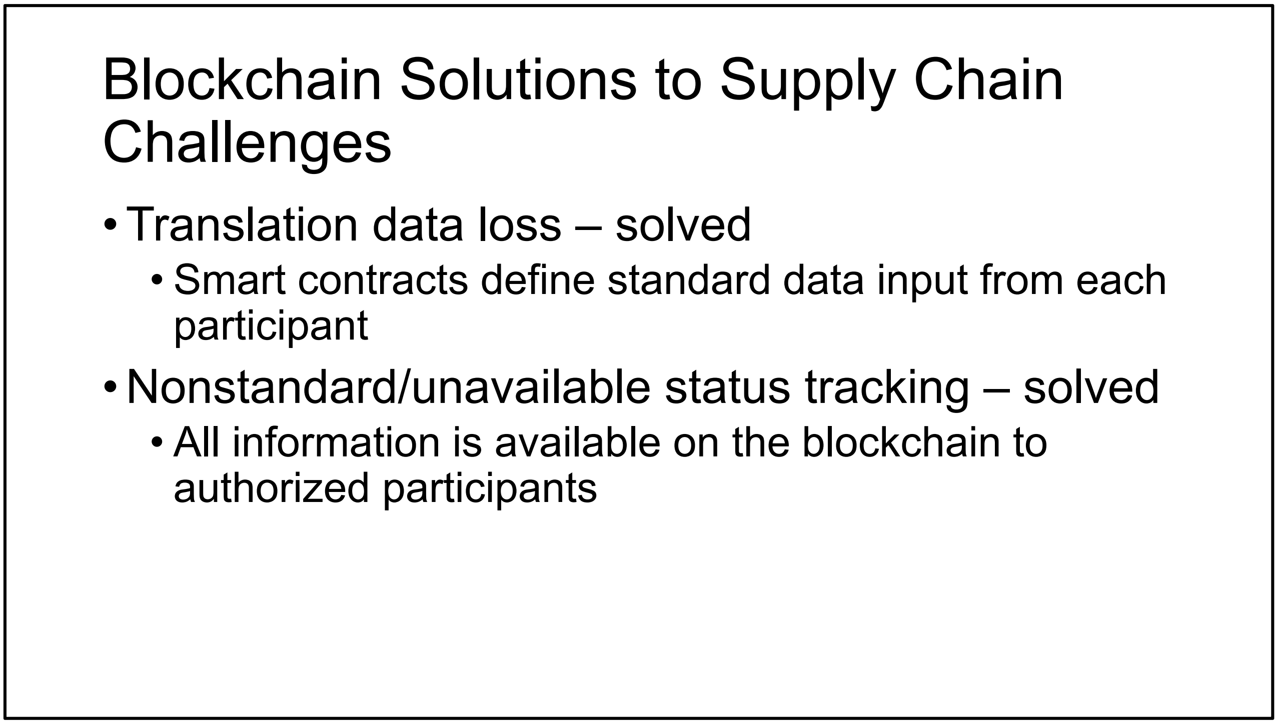
Challenges with Supply Chain

- Translation data loss
  - Some data gets lost, subject to human error when re-typed, etc
- Nonstandard/unavailable status tracking
  - Different standards for each participant, including status updates



Blockchain Solutions to Supply Chain Challenges

- Lack of transparency – solved
  - All transactions shared and verified on the blockchain
- Lack of traceability – solved
  - No central authority
  - All nodes can access transactions
  - Transactions are linked
- Transfer time lag – solved
  - Smart contracts can make on-demand decisions
  - Limits human interactions, errors, and work schedules



Blockchain Solutions to Supply Chain Challenges

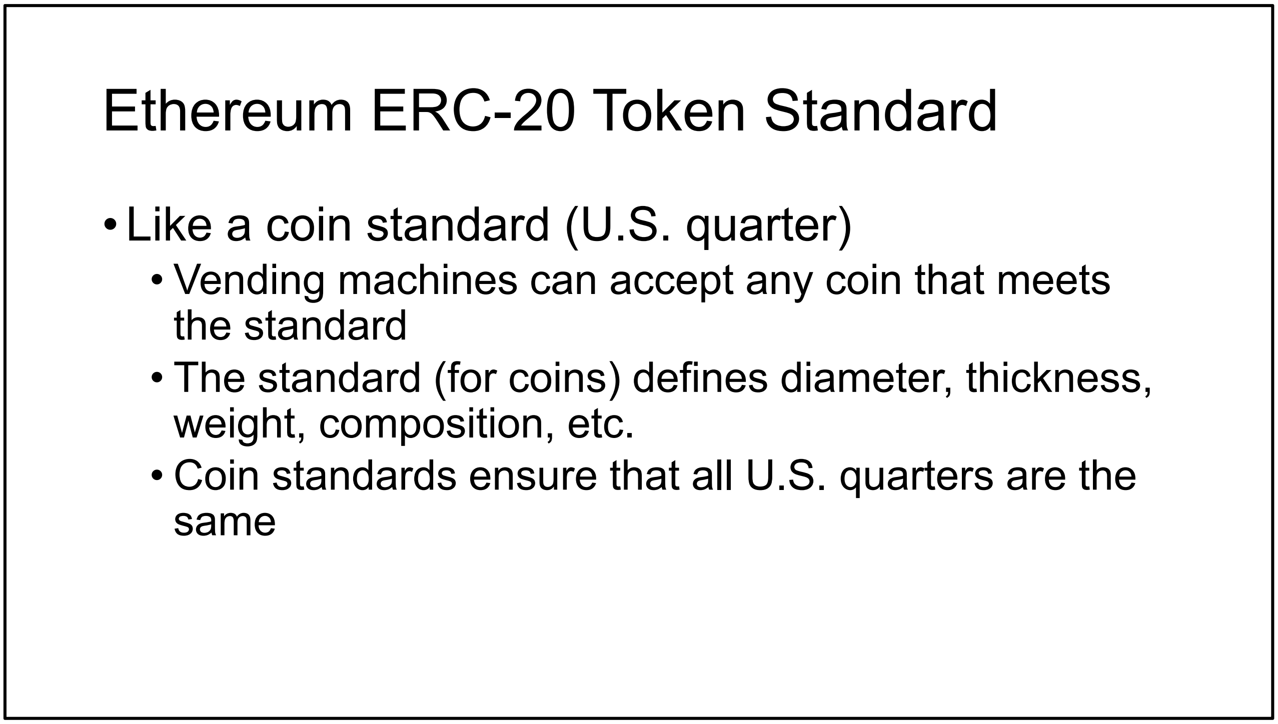
- Translation data loss – solved
  - Smart contracts define standard data input from each participant
- Nonstandard/unavailable status tracking – solved
  - All information is available on the blockchain to authorized participants

Episode 7.04

Blockchain Solution Examples

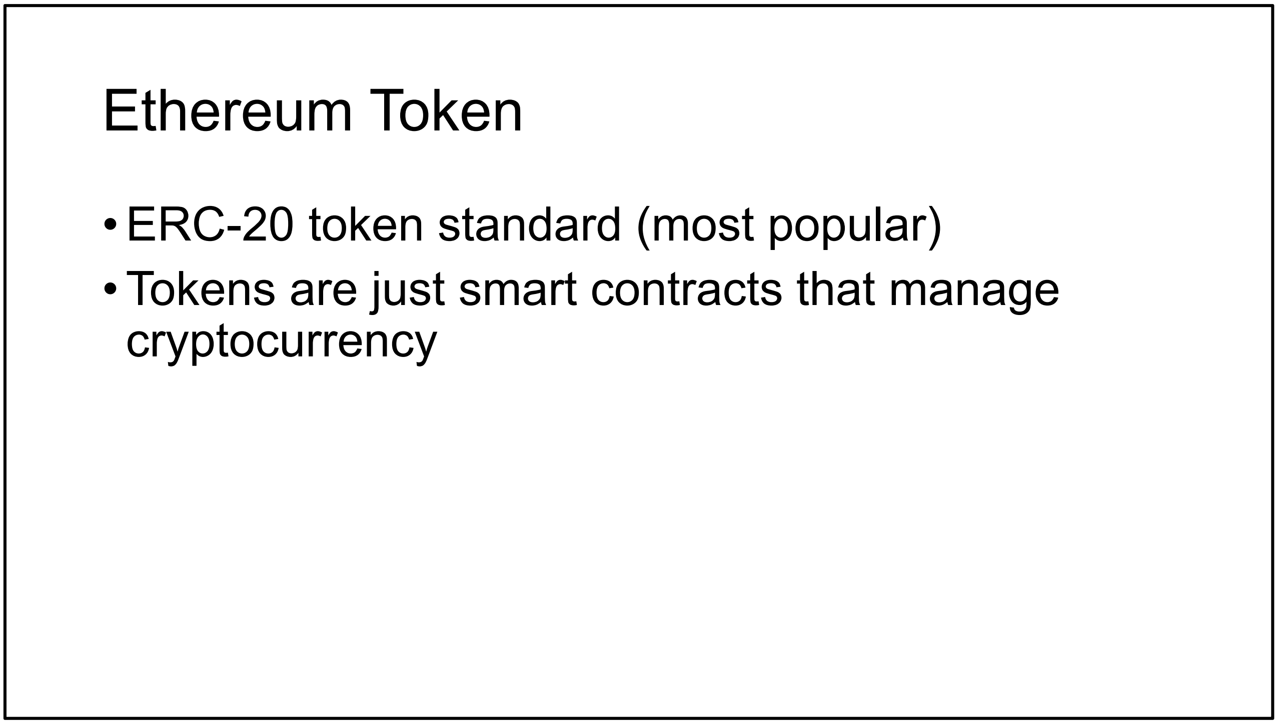
Episode 7.05

Ethereum Tokens



Ethereum ERC-20 Token Standard

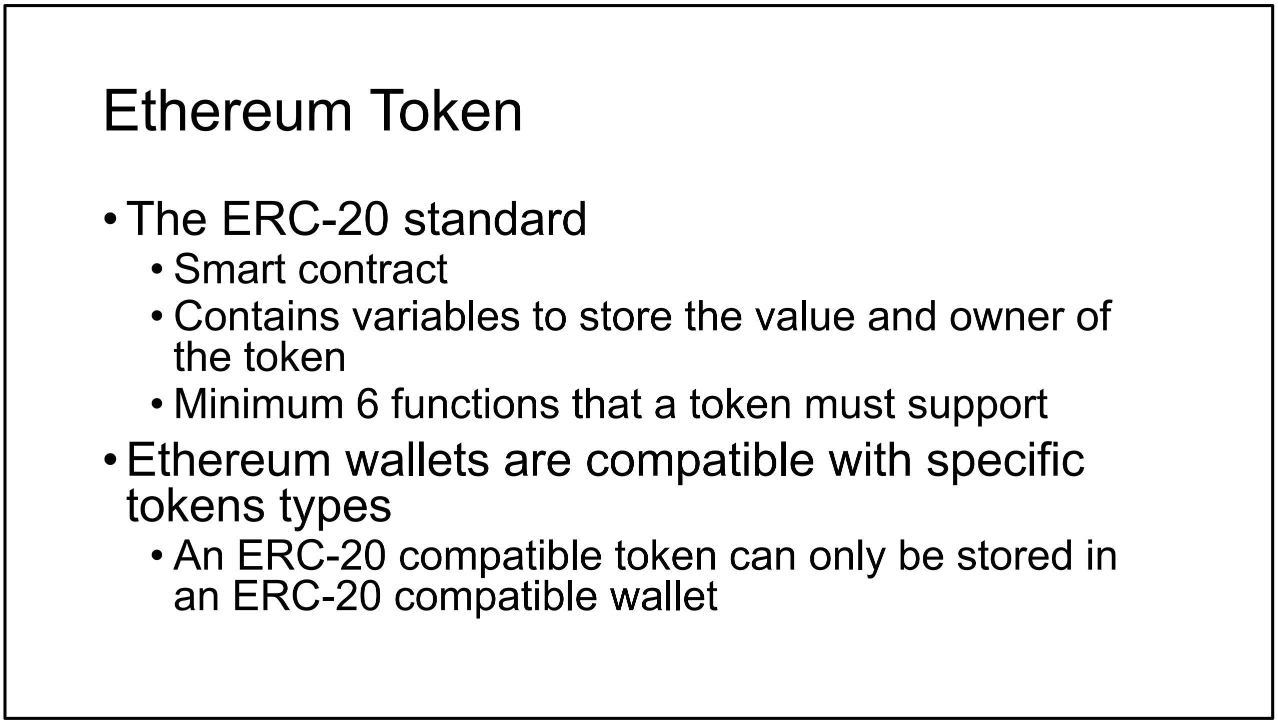
- Like a coin standard (U.S. quarter)
  - Vending machines can accept any coin that meets the standard
  - The standard (for coins) defines diameter, thickness, weight, composition, etc.
  - Coin standards ensure that all U.S. quarters are the same



Ethereum Token

- ERC-20 token standard (most popular)
- Tokens are just smart contracts that manage cryptocurrency

https://etherscan.io

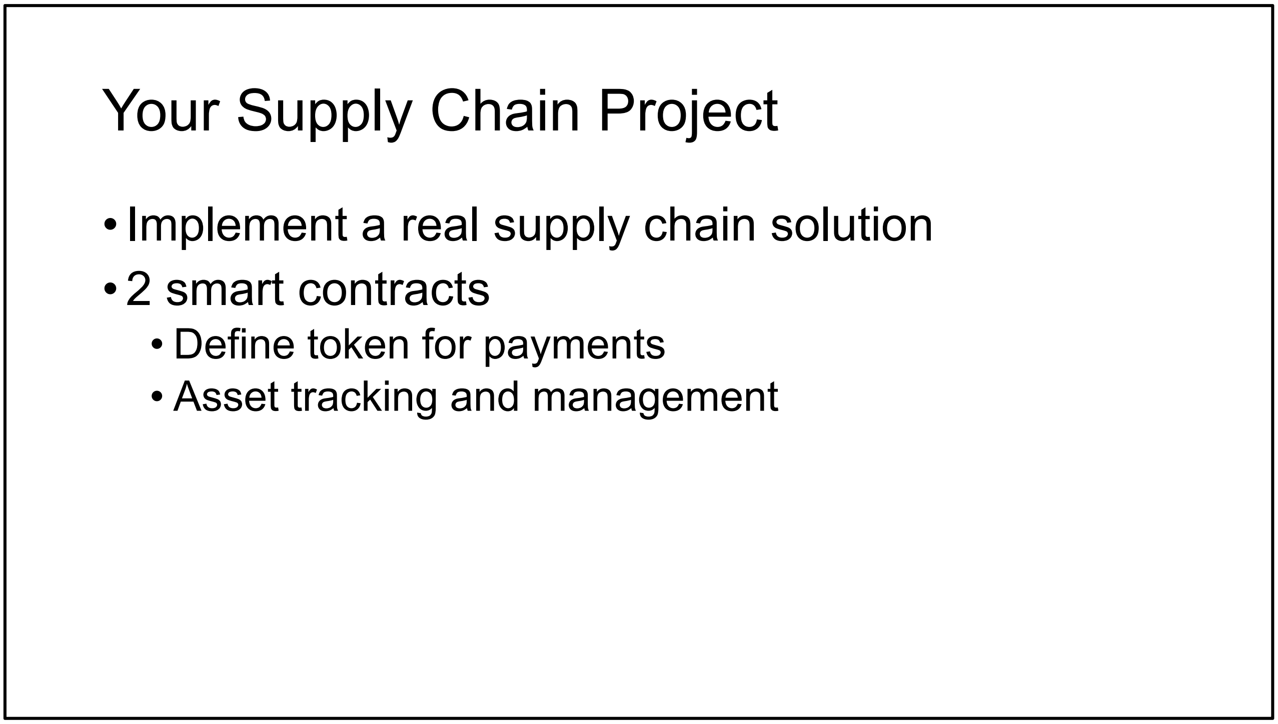


Ethereum Token

- The ERC-20 standard
  - Smart contract
  - Contains variables to store the value and owner of the token
  - Minimum 6 functions that a token must support
- Ethereum wallets are compatible with specific tokens types
  - An ERC-20 compatible token can only be stored in an ERC-20 compatible wallet

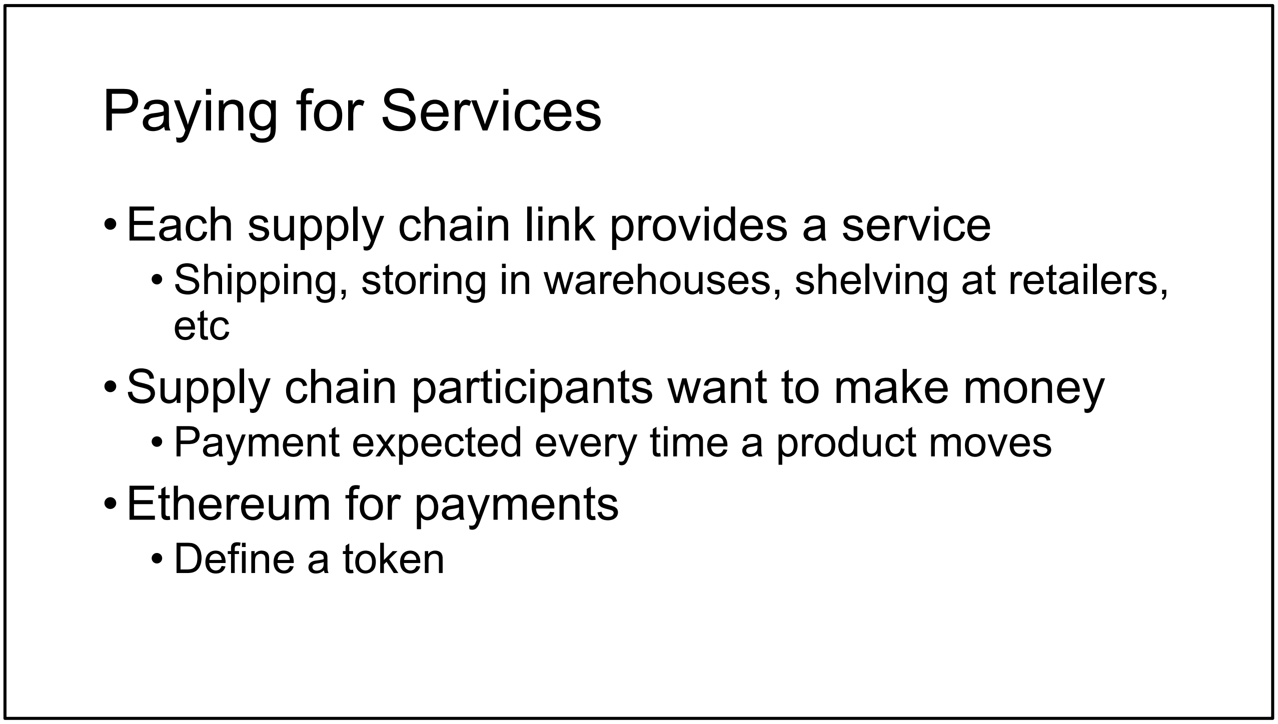
Episode 7.06

Your Supply Chain Project



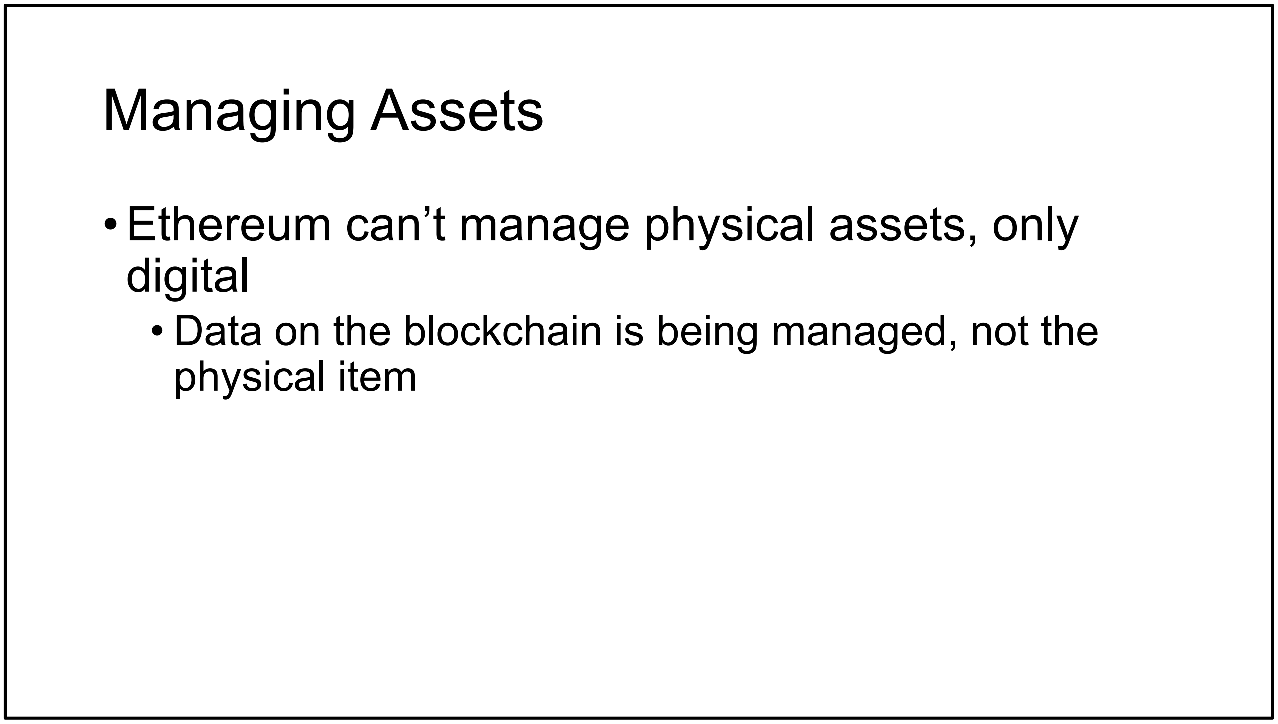
Your Supply Chain Project

- Implement a real supply chain solution
- 2 smart contracts
  - Define token for payments
  - Asset tracking and management



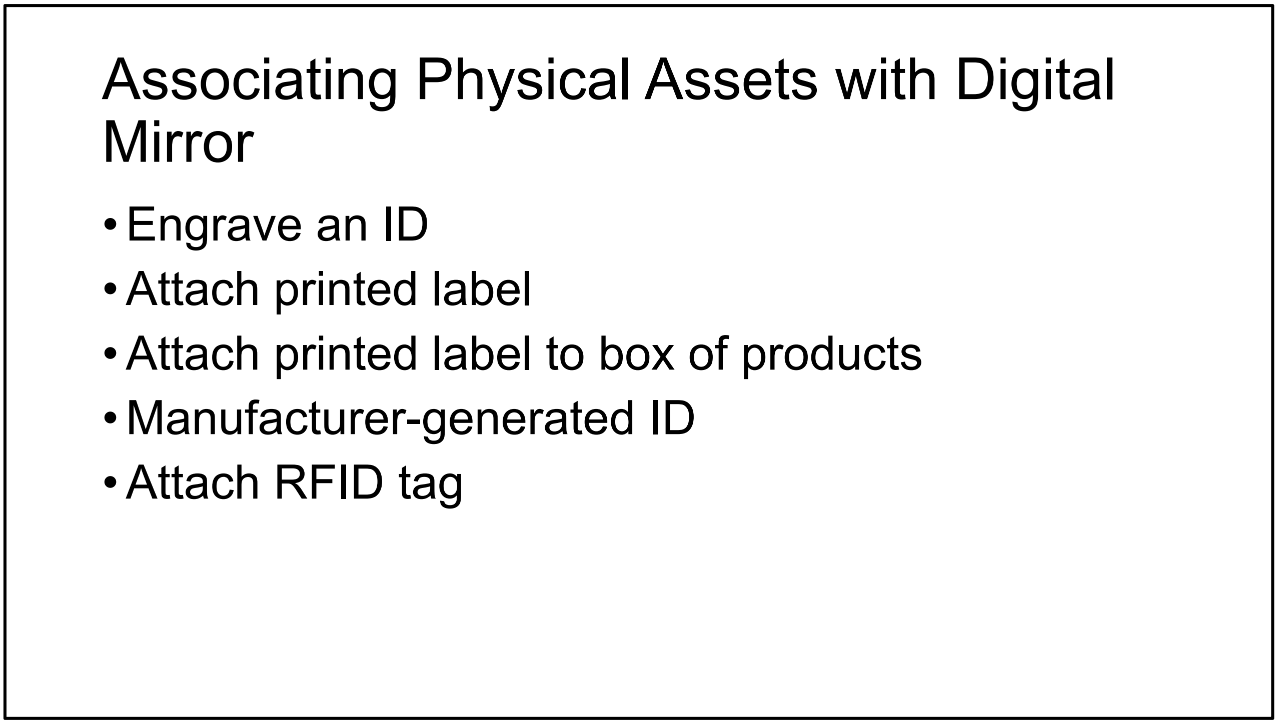
Paying for Services

- Each supply chain link provides a service
  - Shipping, storing in warehouses, shelving at retailers, etc
- Supply chain participants want to make money
  - Payment expected every time a product moves
- Ethereum for payments
  - Define a token



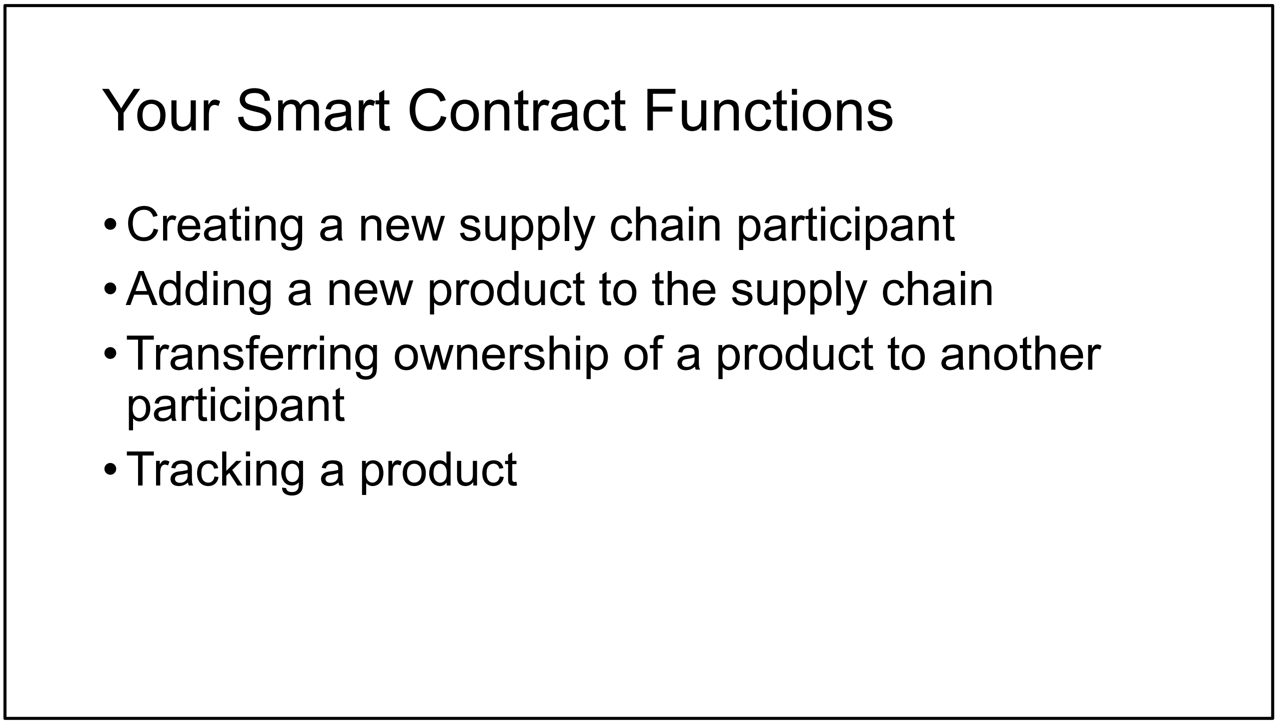
Managing Assets

- Ethereum can’t manage physical assets, only digital
  - Data on the blockchain is being managed, not the physical item



Associating Physical Assets with Digital Mirror

- Engrave an ID
- Attach printed label
- Attach printed label to box of products
- Manufacturer-generated ID
- Attach RFID tag

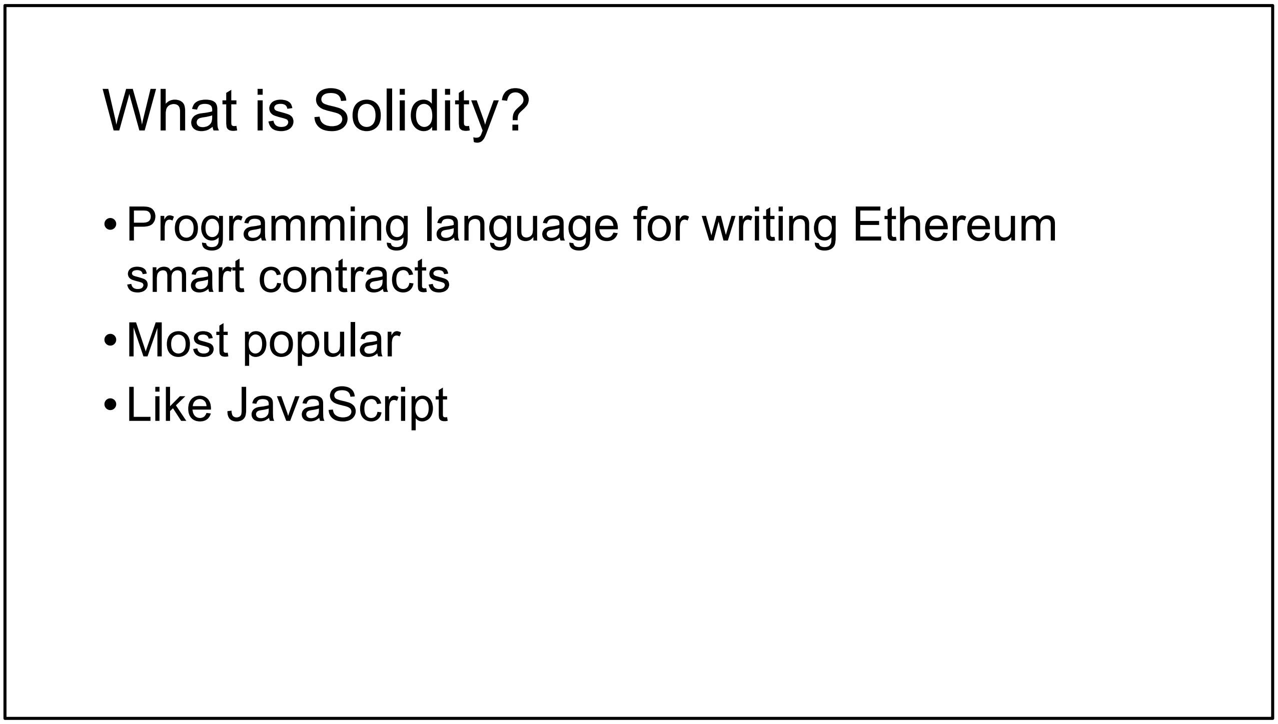


Your Smart Contract Functions

- Creating a new supply chain participant
- Adding a new product to the supply chain
- Transferring ownership of a product to another participant
- Tracking a product

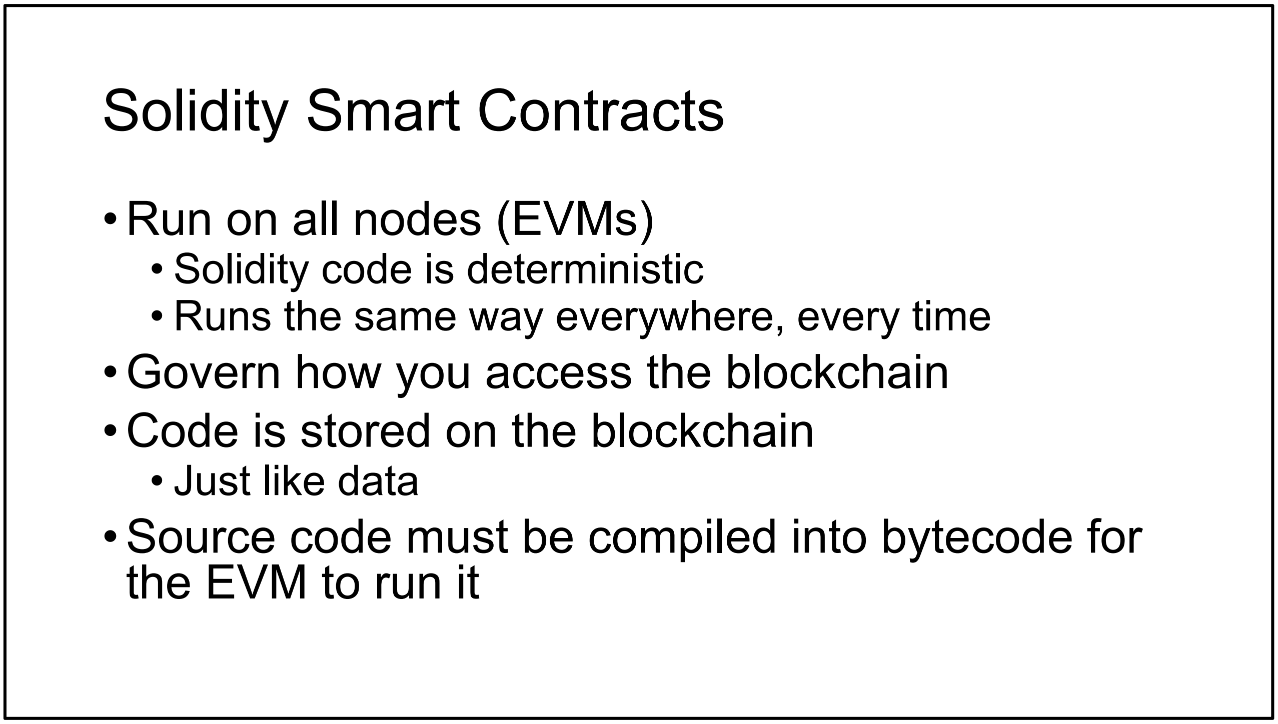
Episode 7.07

Exploring Solidity



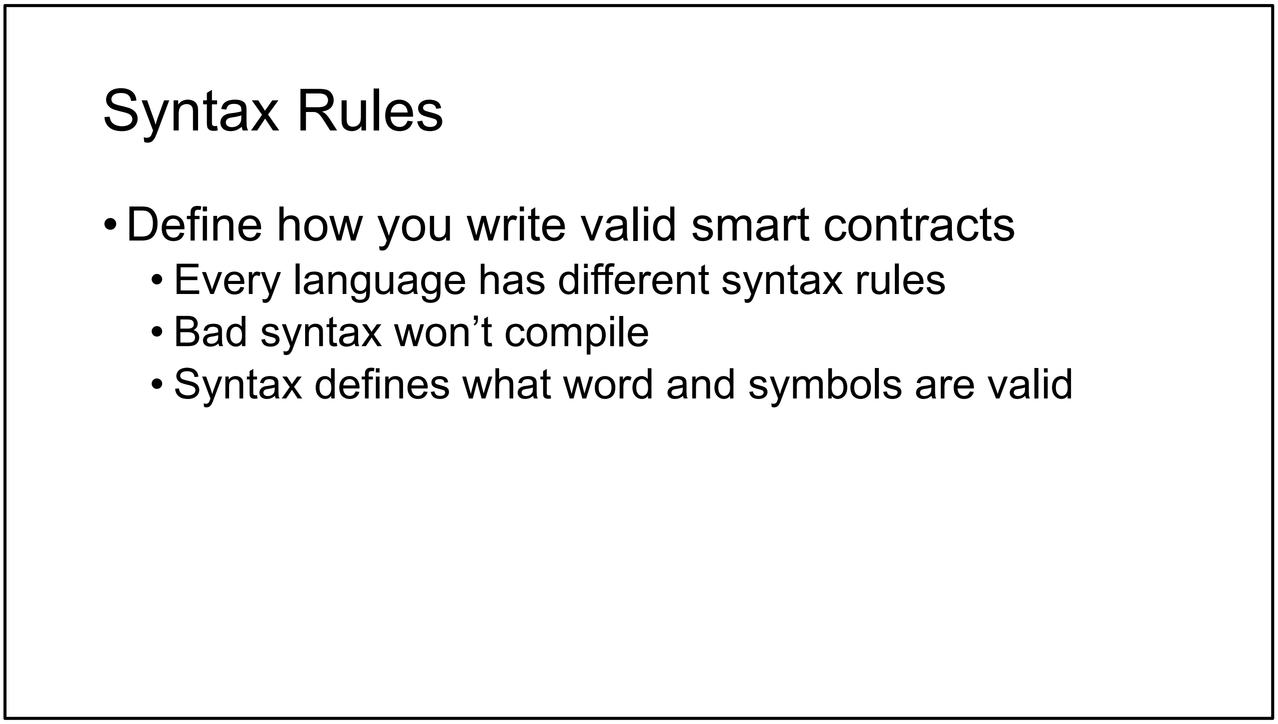
What is Solidity?

- Programming language for writing Ethereum smart contracts
- Most popular
- Like JavaScript



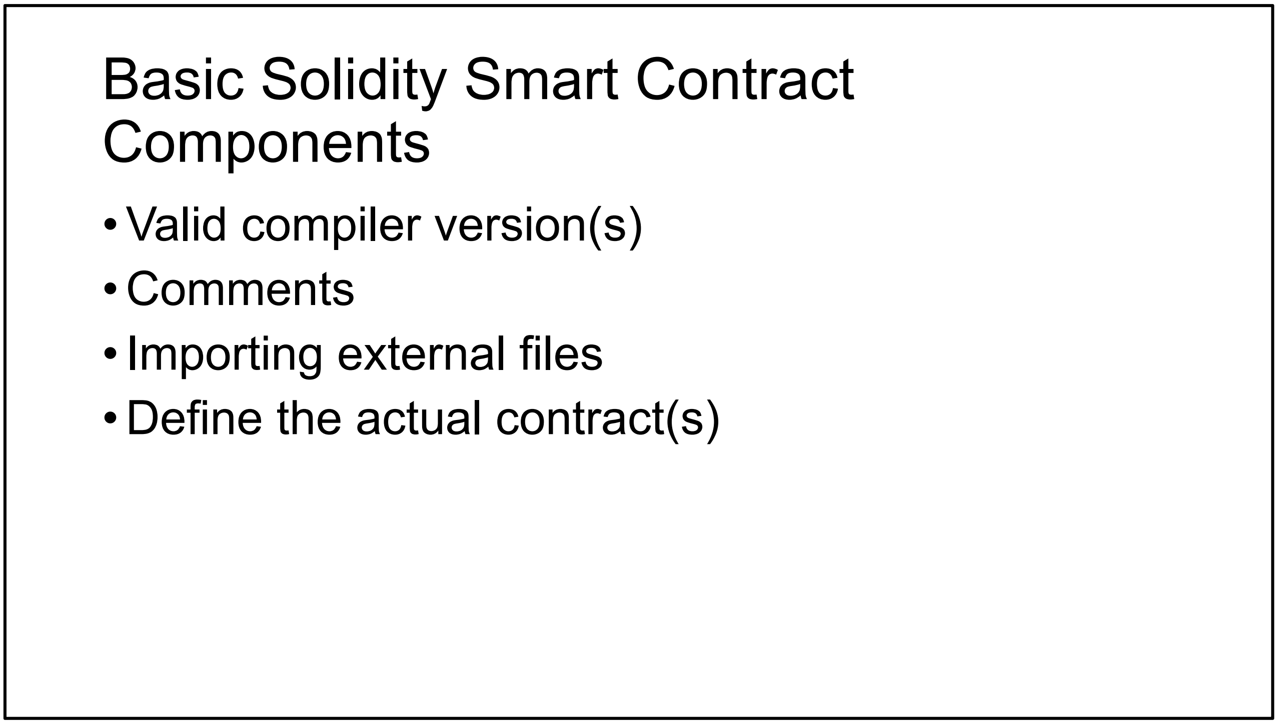
Solidity Smart Contracts

- Run on all nodes (EVMs)
  - Solidity code is deterministic
  - Runs the same way everywhere, every time
- Govern how you access the blockchain
- Code is stored on the blockchain
  - Just like data
- Source code must be compiled into bytecode for the EVM to run it



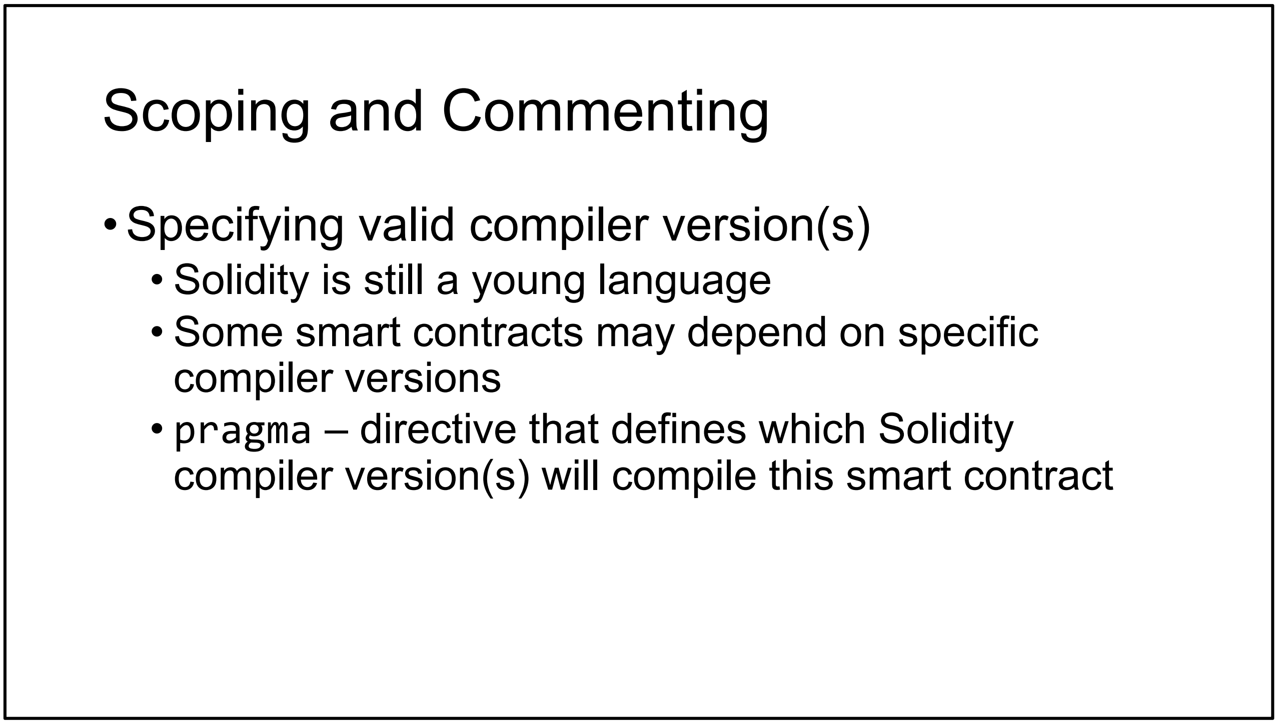
Syntax Rules

- Define how you write valid smart contracts
  - Every language has different syntax rules
  - Bad syntax won’t compile
  - Syntax defines what word and symbols are valid



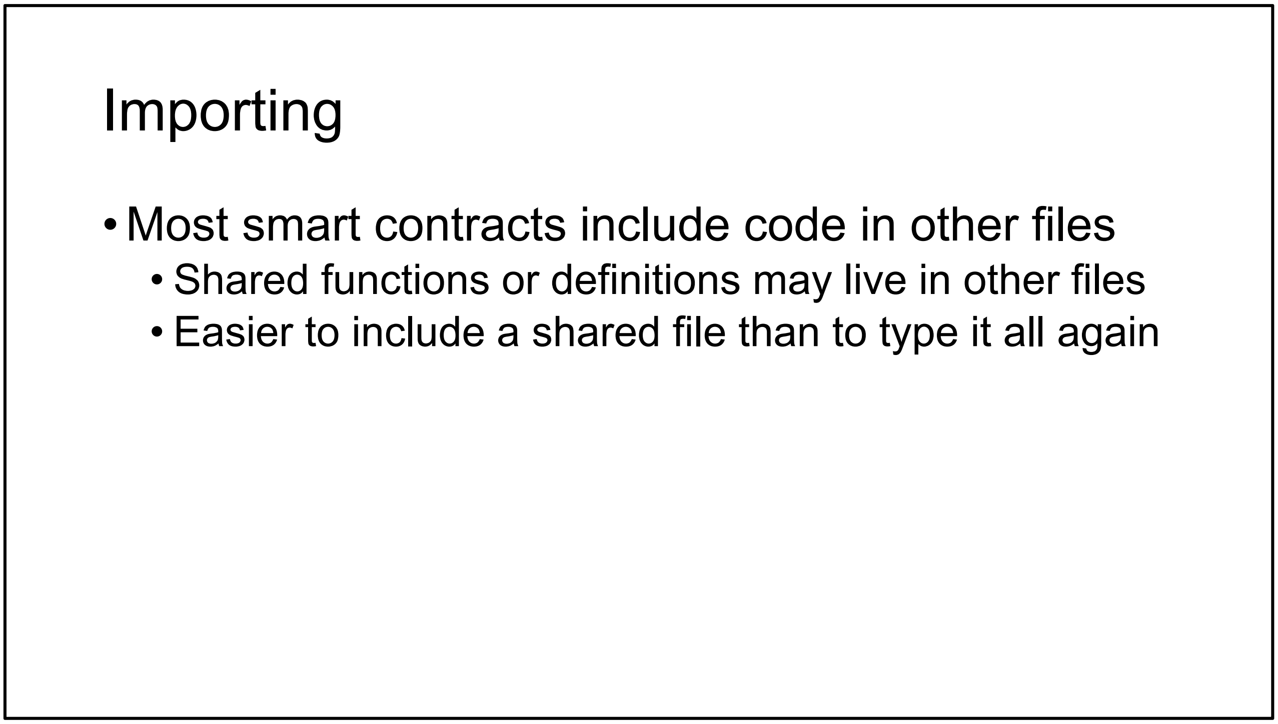
Basic Solidity Smart Contract Components

- Valid compiler version(s)
- Comments
- Importing external files
- Define the actual contract(s)



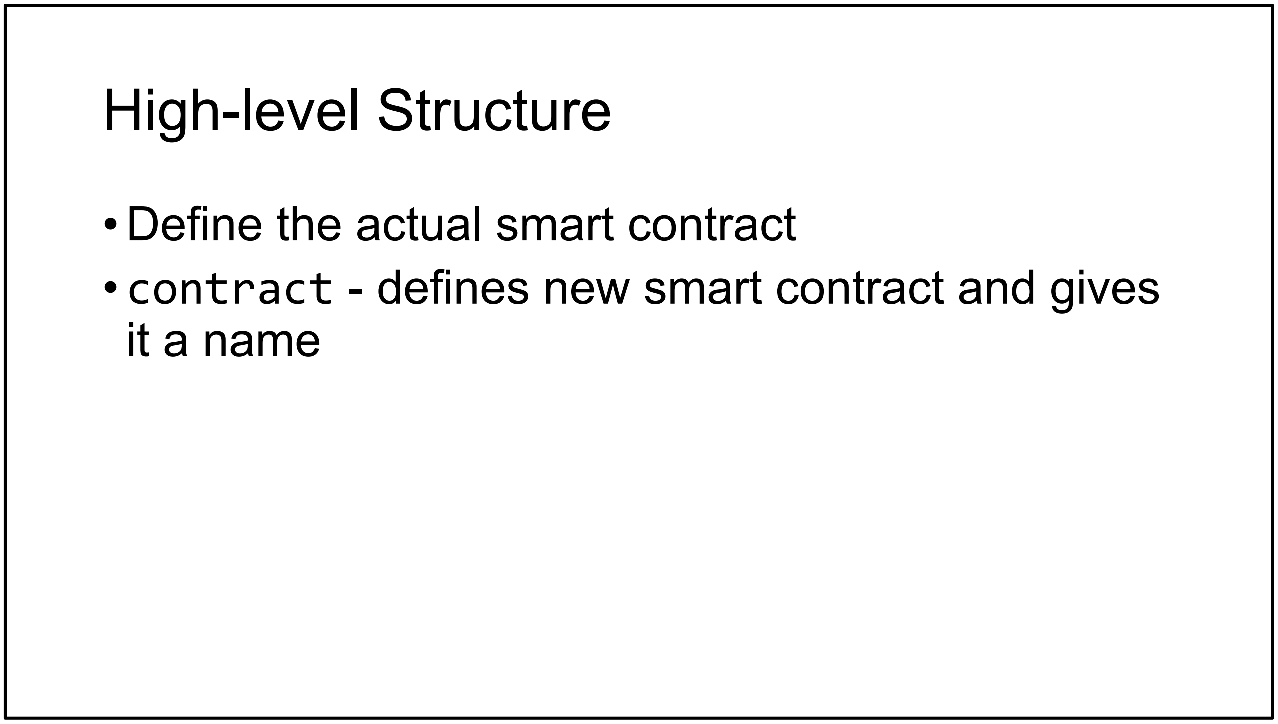
Scoping and Commenting

- Specifying valid compiler version(s)
  - Solidity is still a young language
  - Some smart contracts may depend on specific compiler versions
  - `pragma` – directive that defines which Solidity compiler version(s) will compile this smart contract



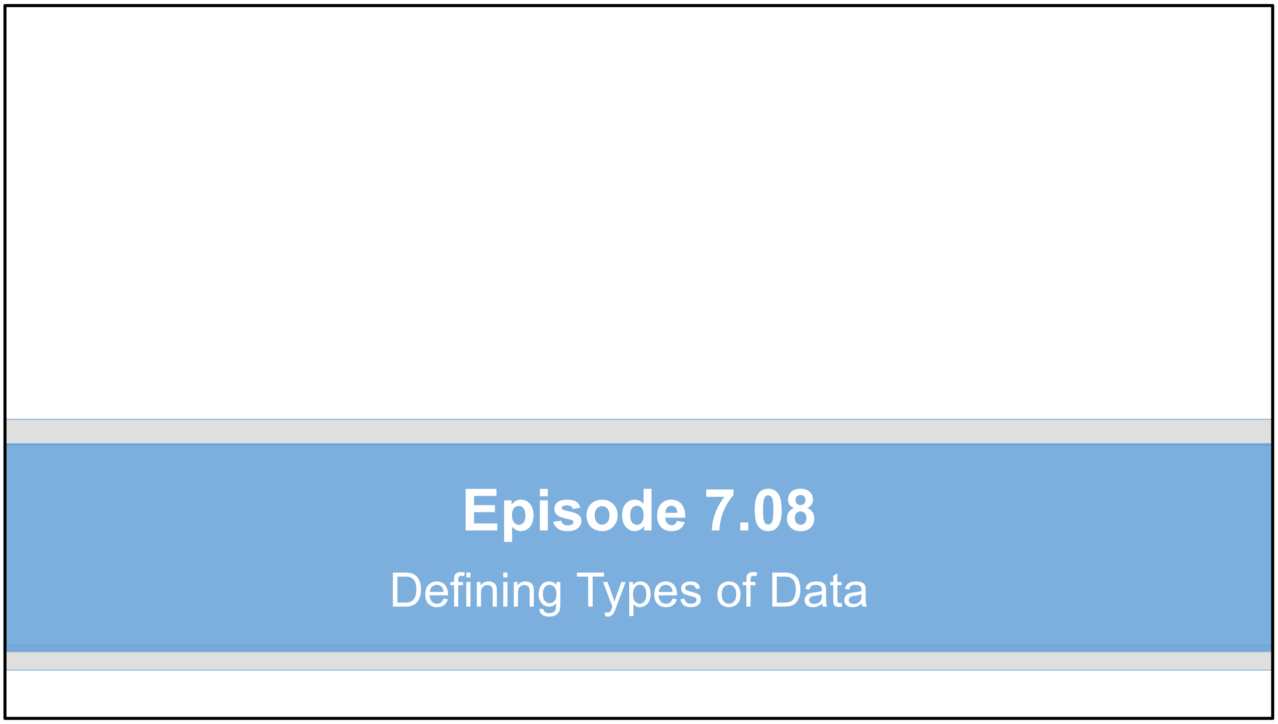
Importing

- Most smart contracts include code in other files
  - Shared functions or definitions may live in other files
  - Easier to include a shared file than to type it all again



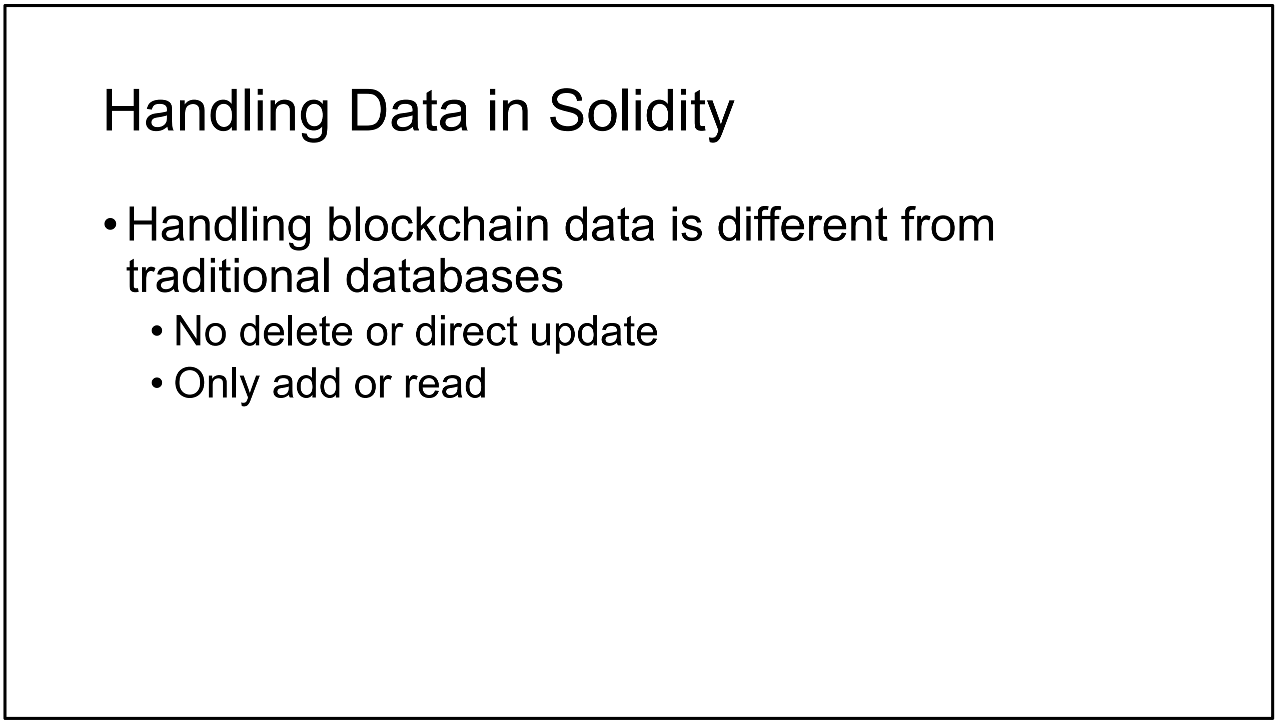
High-level Structure

- Define the actual smart contract
- `contract` - defines new smart contract and gives it a name



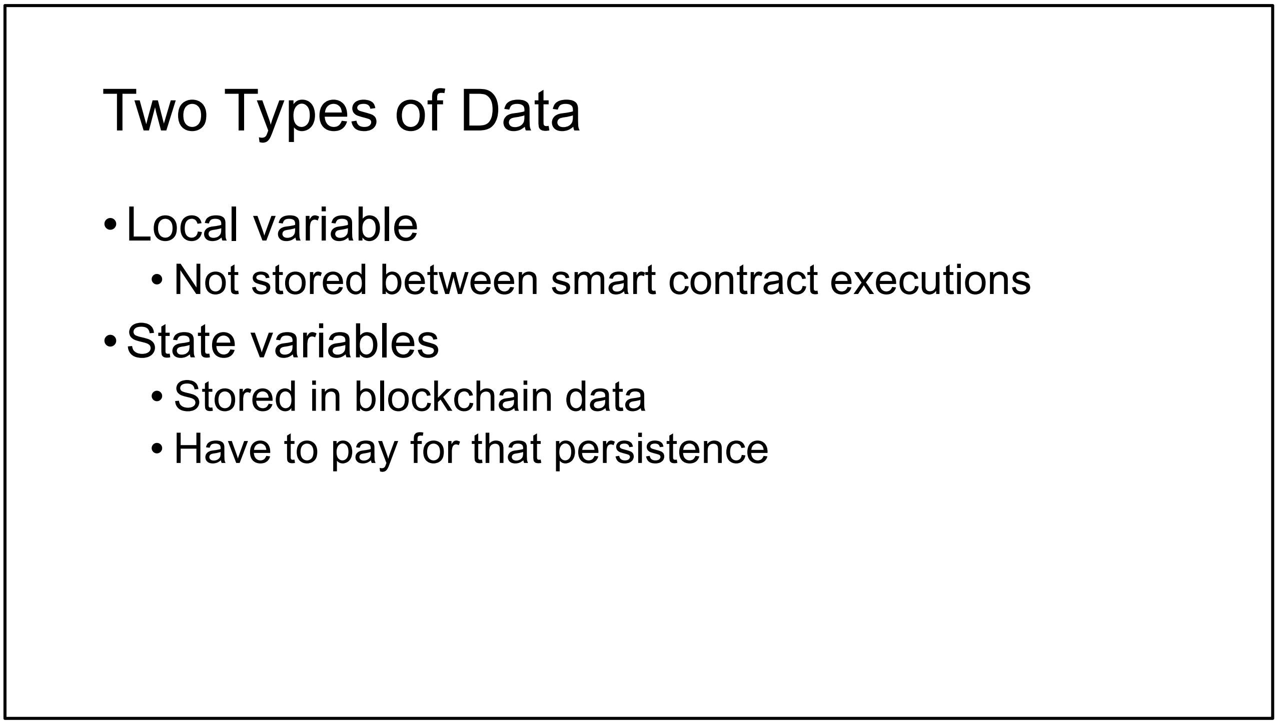
Episode 7.08

Defining Types of Data



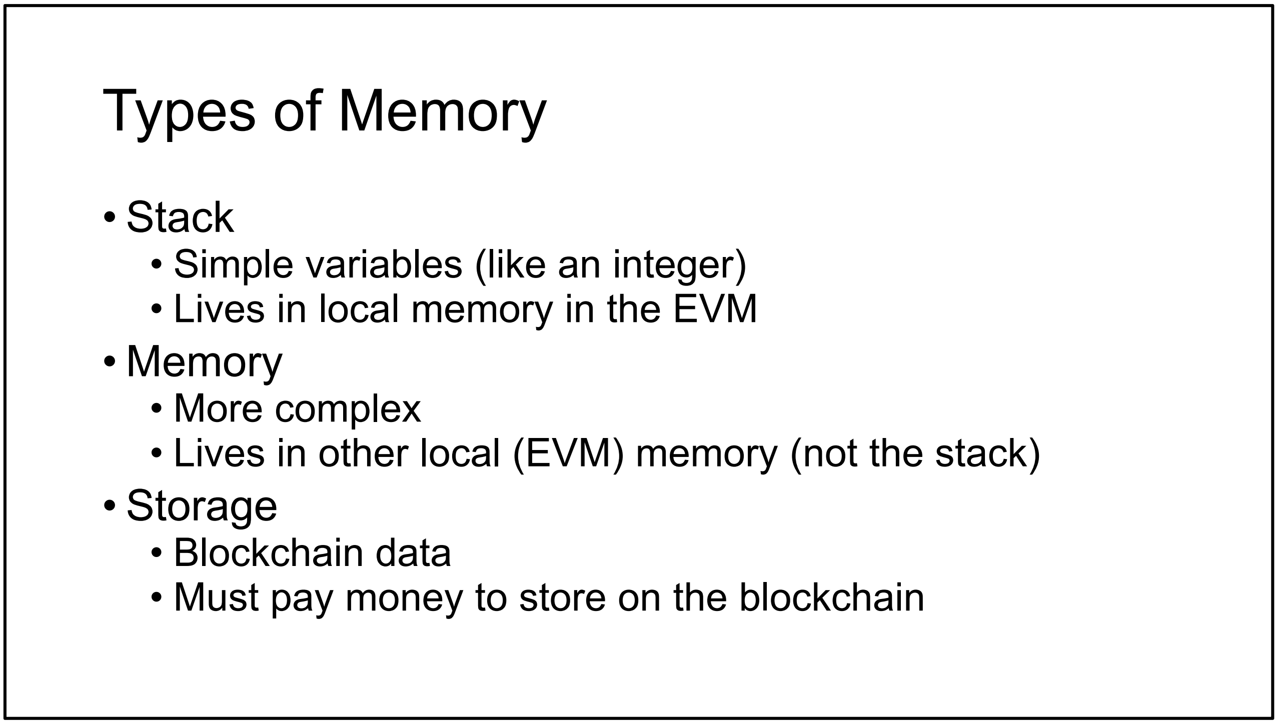
Handling Data in Solidity

- Handling blockchain data is different from traditional databases
  - No delete or direct update
  - Only add or read



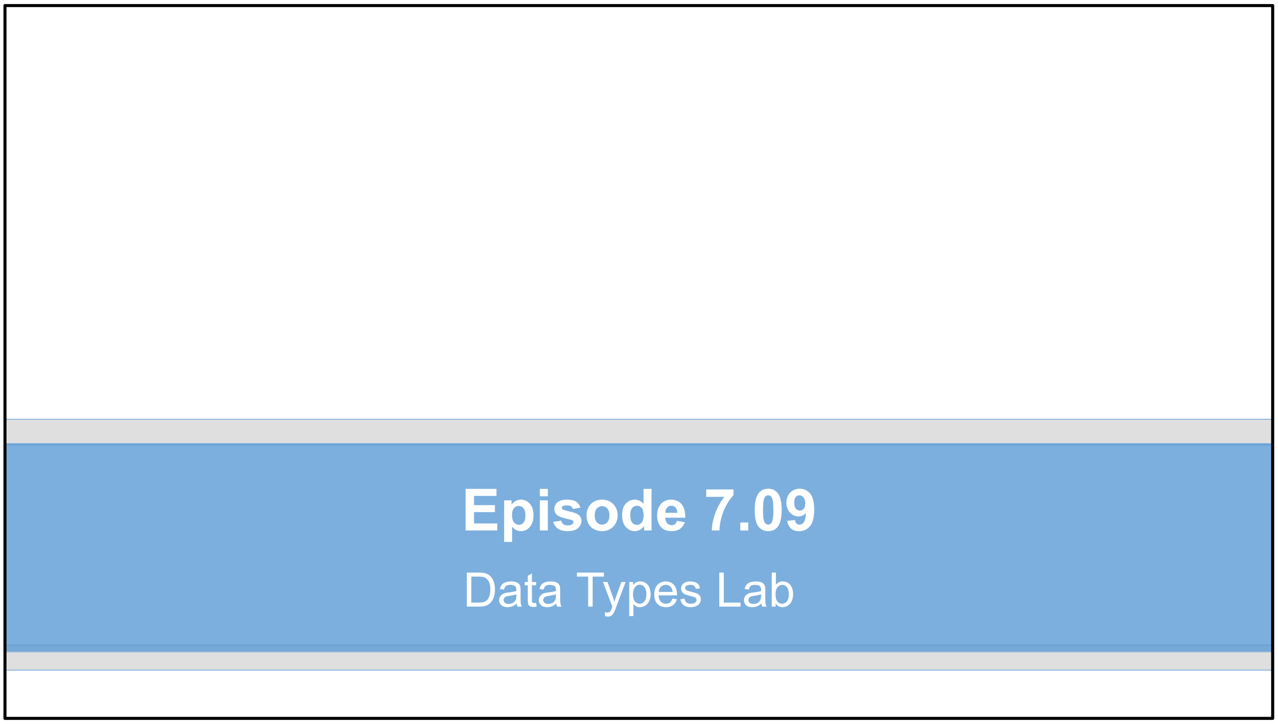
Two Types of Data

- Local variable
  - Not stored between smart contract executions
- State variables
  - Stored in blockchain data
  - Have to pay for that persistence



Types of Memory

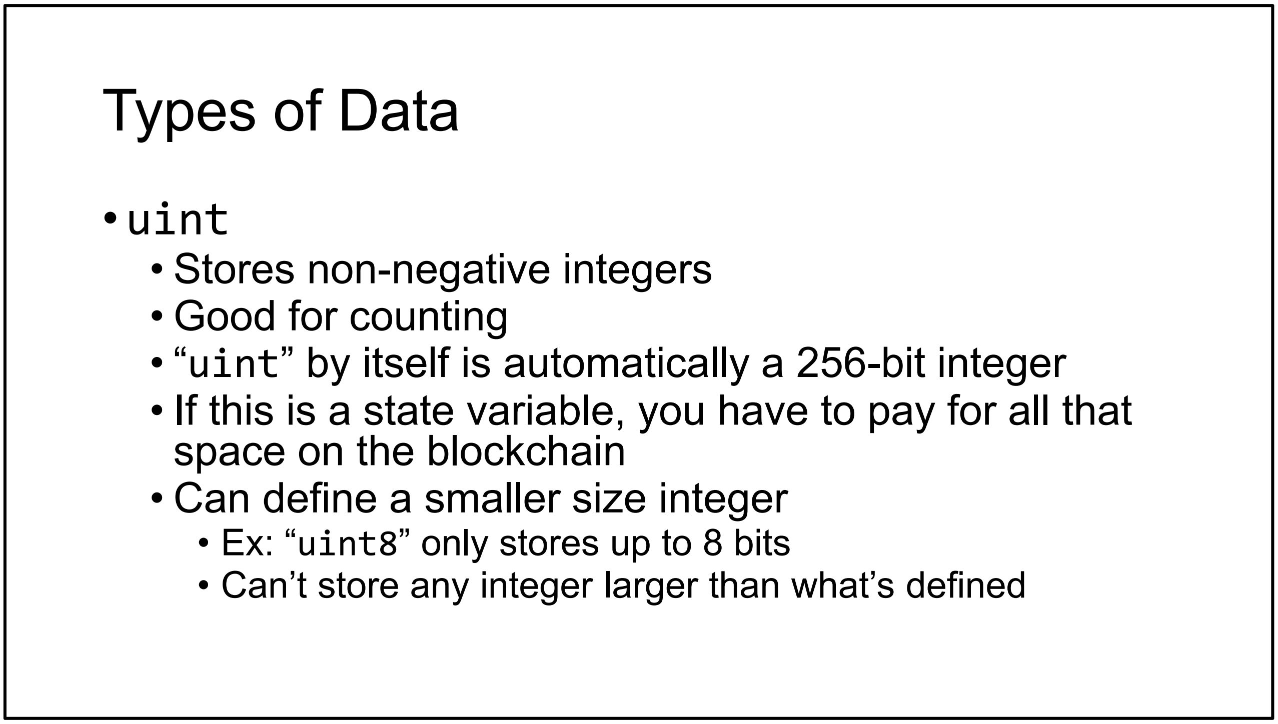
- Stack
  - Simple variables (like an integer)
  - Lives in local memory in the EVM
- Memory
  - More complex
  - Lives in other local (EVM) memory (not the stack)
- Storage
  - Blockchain data
  - Must pay money to store on the blockchain



Episode 7.09

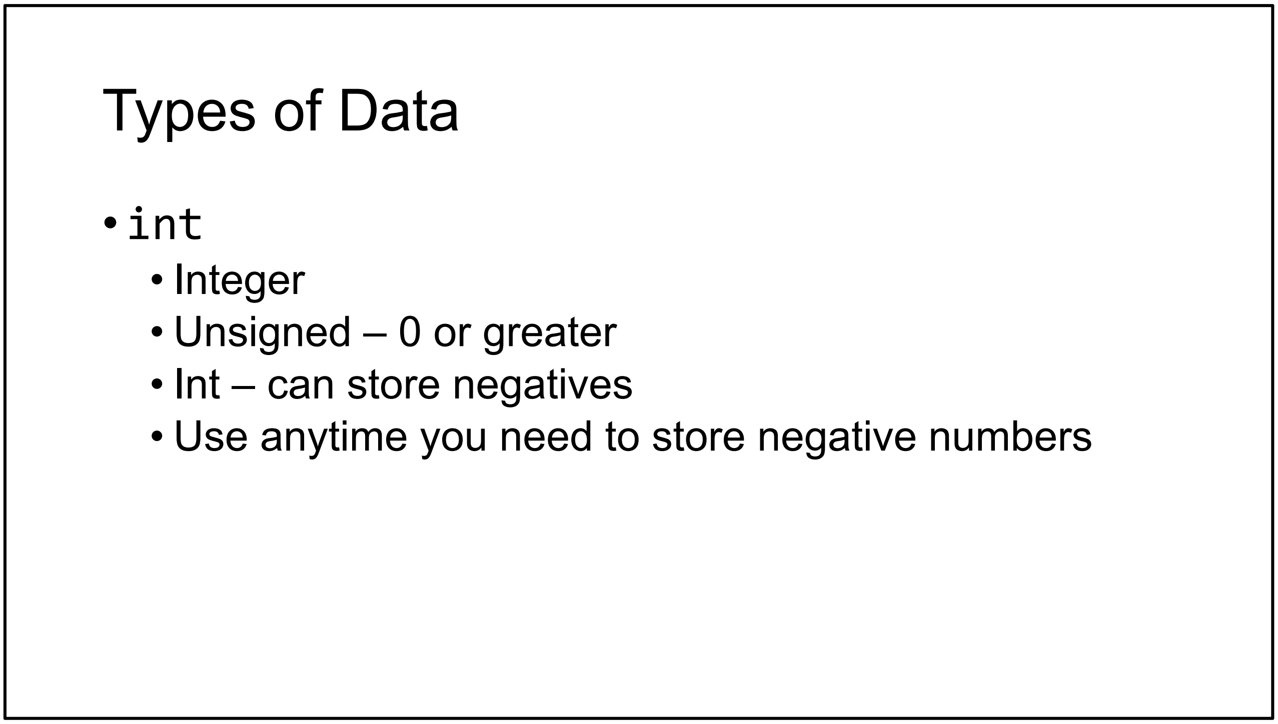
Data Types Lab

```
pragma solidity ^0.5.0;  // ^0.4.24;

/*
* @title Solidity data types
* @author Michael Solomon
* @notice A simple smart contract to demonstrate simple data types available in Solidity
*
*/

contract DataTypes {

    uint x = 9;
    int i = -68;
    uint8 j = 17;
    bool isEthereumCool = true;
    address owner = msg.sender; // msg.sender is the Ethereum address of the account that sent this transaction
    bytes32 bMsg = "hello";
    string sMsg = "hello";

    function getStateVariables() public view returns (uint, int, uint, bool, address, bytes32, string memory) {
        return (x, i, j, isEthereumCool, owner, bMsg, sMsg);
    }

}
```



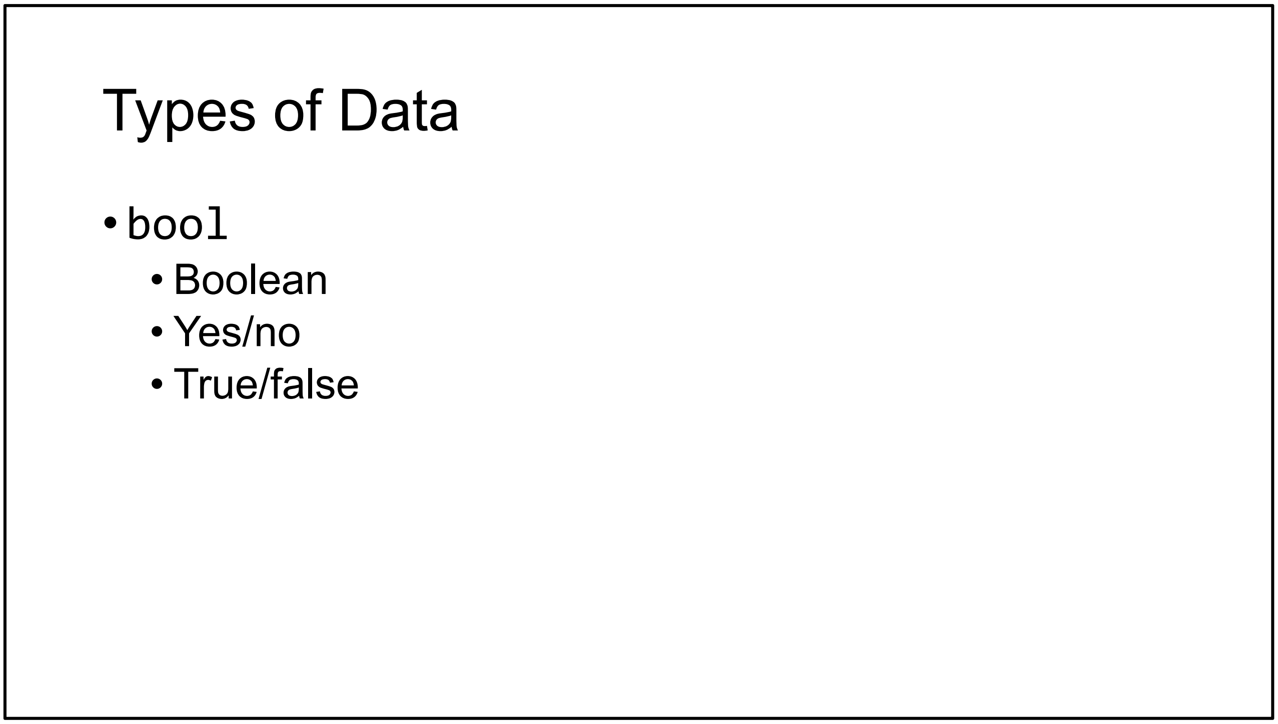
Types of Data

- `uint`
  - Stores non-negative integers
  - Good for counting
  - “`uint`” by itself is automatically a 256-bit integer
  - If this is a state variable, you have to pay for all that space on the blockchain
  - Can define a smaller size integer
    - Ex: “`uint8`” only stores up to 8 bits
    - Can’t store any integer larger than what’s defined



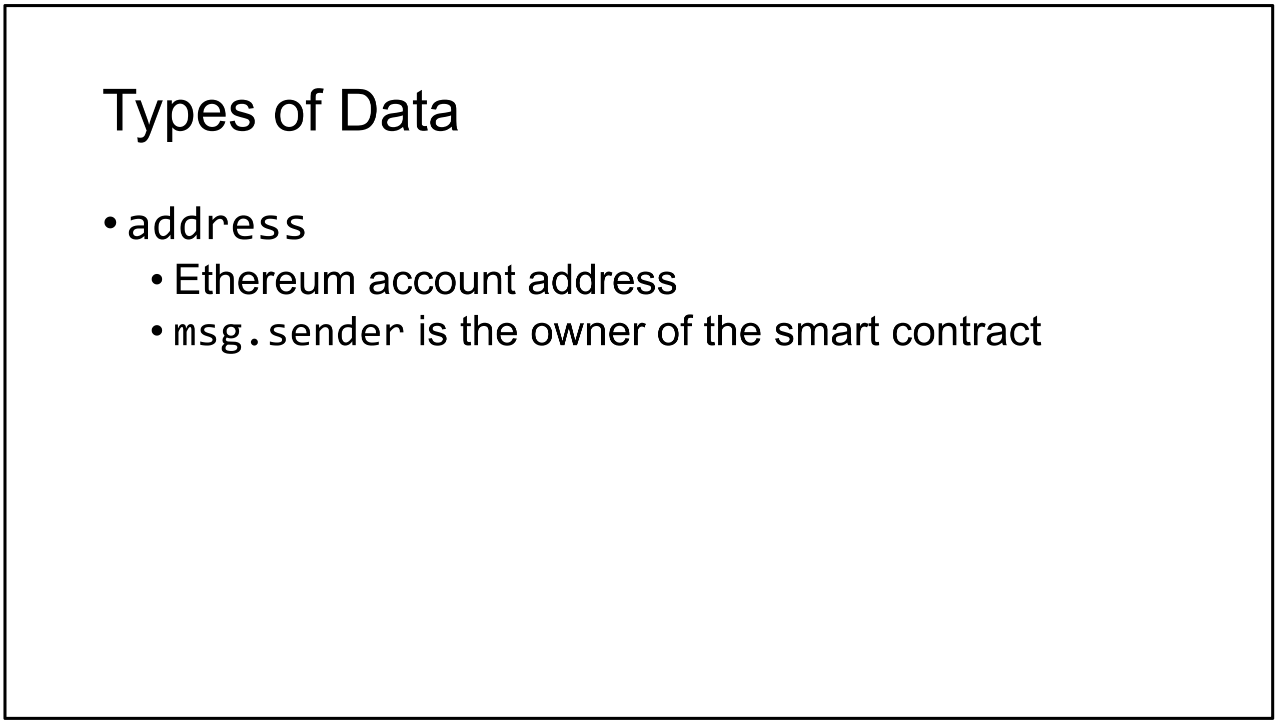
Types of Data

- `int`
  - Integer
  - Unsigned – 0 or greater
  - Int – can store negatives
  - Use anytime you need to store negative numbers



Types of Data

- `bool`
  - Boolean
  - Yes/no
  - True/false

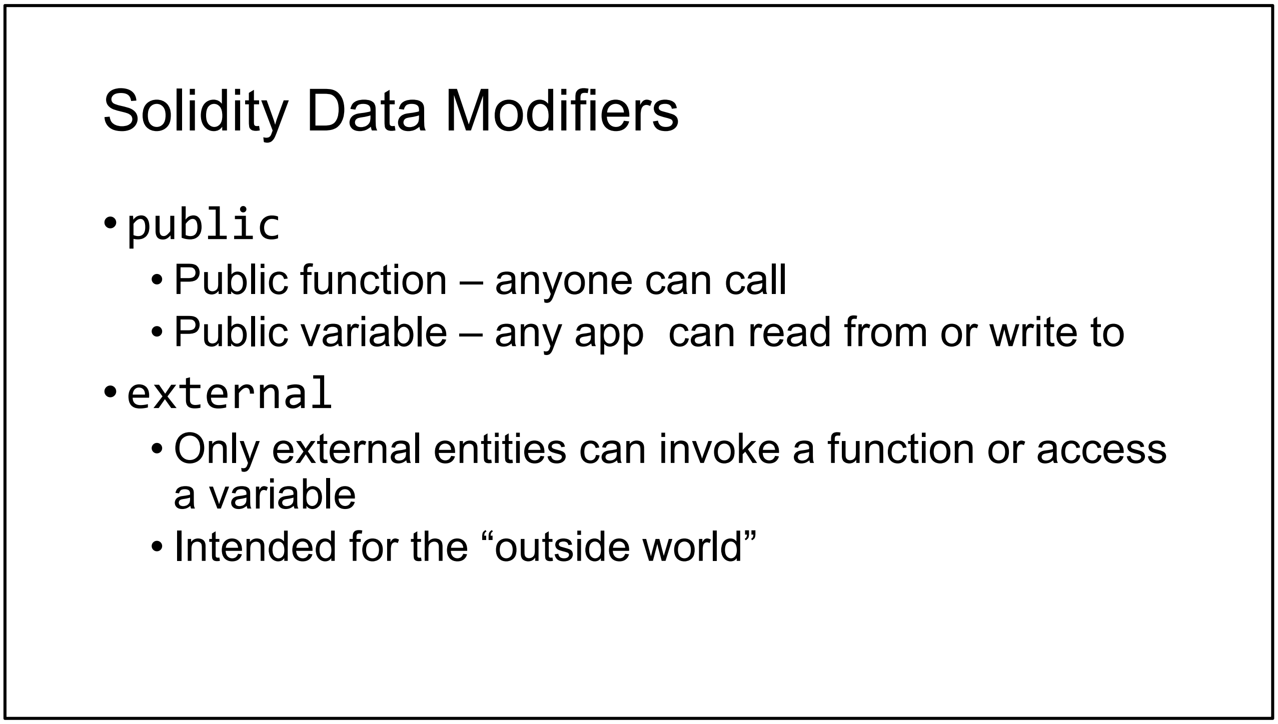


Types of Data

- `address`
  - Ethereum account address
  - `msg.sender` is the owner of the smart contract

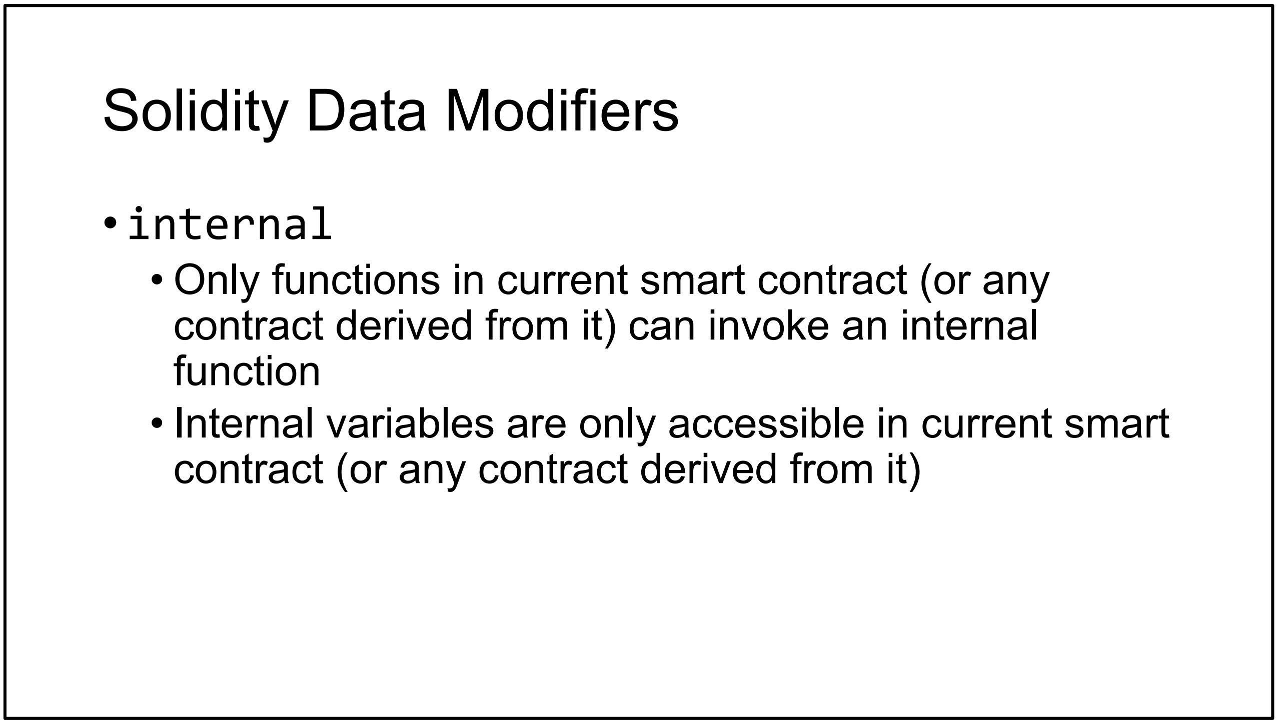
Episode 7.10

Solidity Data Modifiers, Part 1



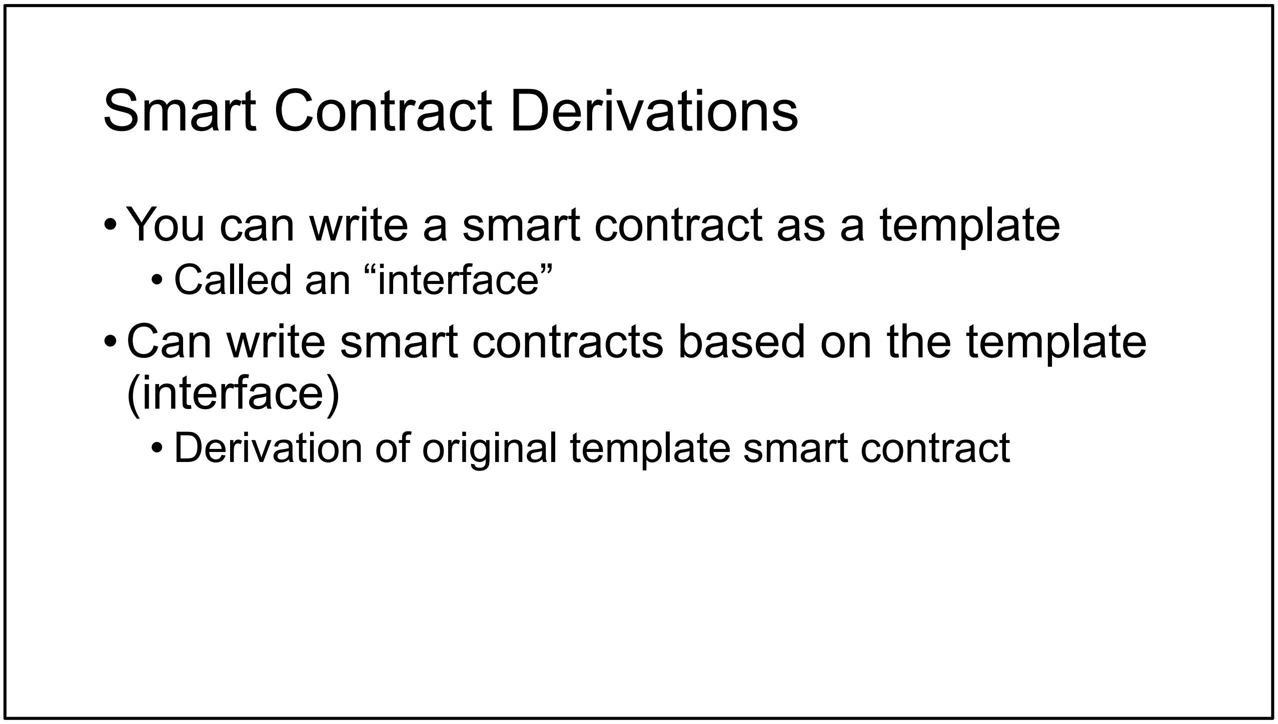
Solidity Data Modifiers

- `public`
  - Public function – anyone can call
  - Public variable – any app can read from or write to
- `external`
  - Only external entities can invoke a function or access a variable
  - Intended for the “outside world”



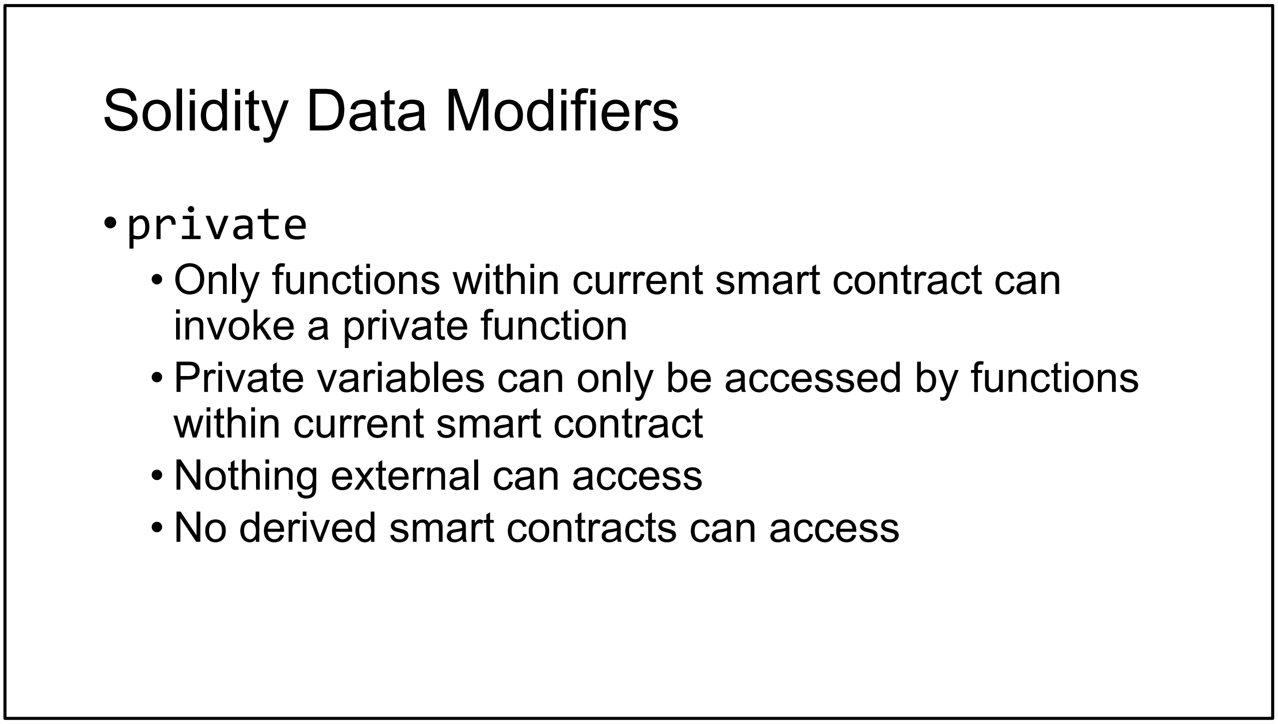
Solidity Data Modifiers

- `internal`
  - Only functions in current smart contract (or any contract derived from it) can invoke an internal function
  - Internal variables are only accessible in current smart contract (or any contract derived from it)



Smart Contract Derivations

- You can write a smart contract as a template
  - Called an “interface”
- Can write smart contracts based on the template (interface)
  - Derivation of original template smart contract



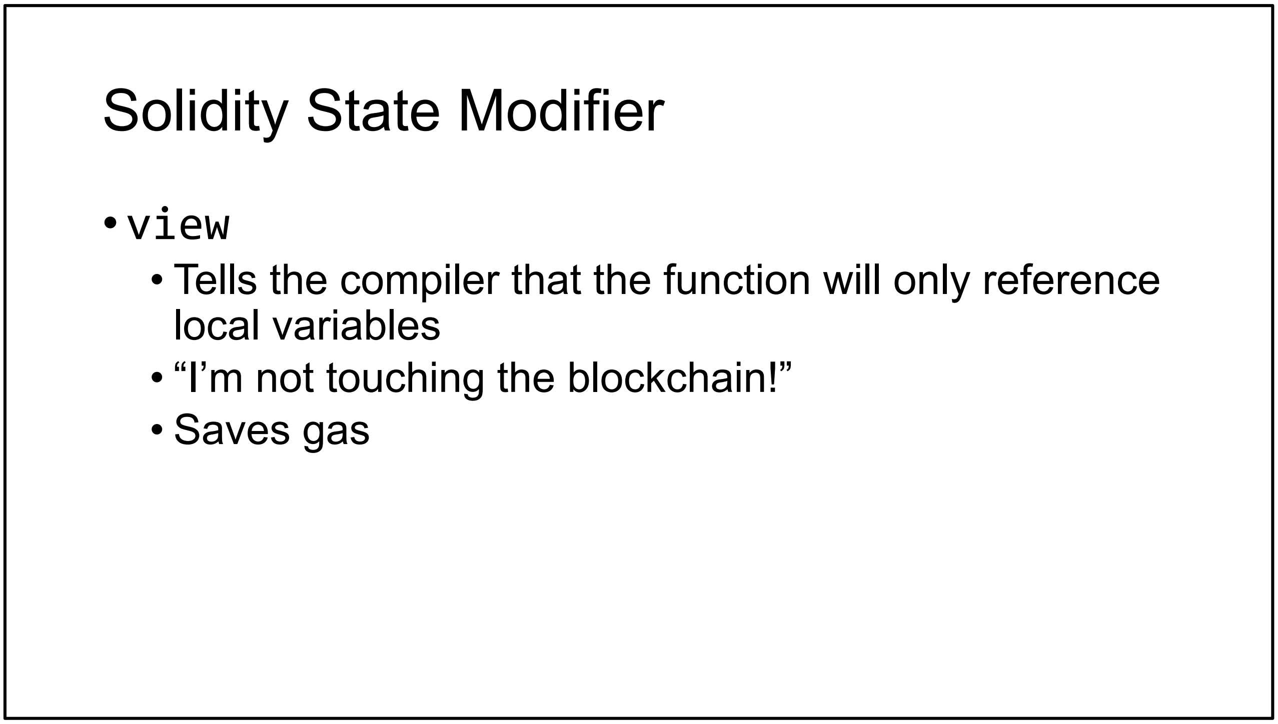
Solidity Data Modifiers

- `private`
  - Only functions within current smart contract can invoke a private function
  - Private variables can only be accessed by functions within current smart contract
  - Nothing external can access
  - No derived smart contracts can access

Episode 7.11

Solidity Data Modifiers, Part 2

Split this episode 10/17

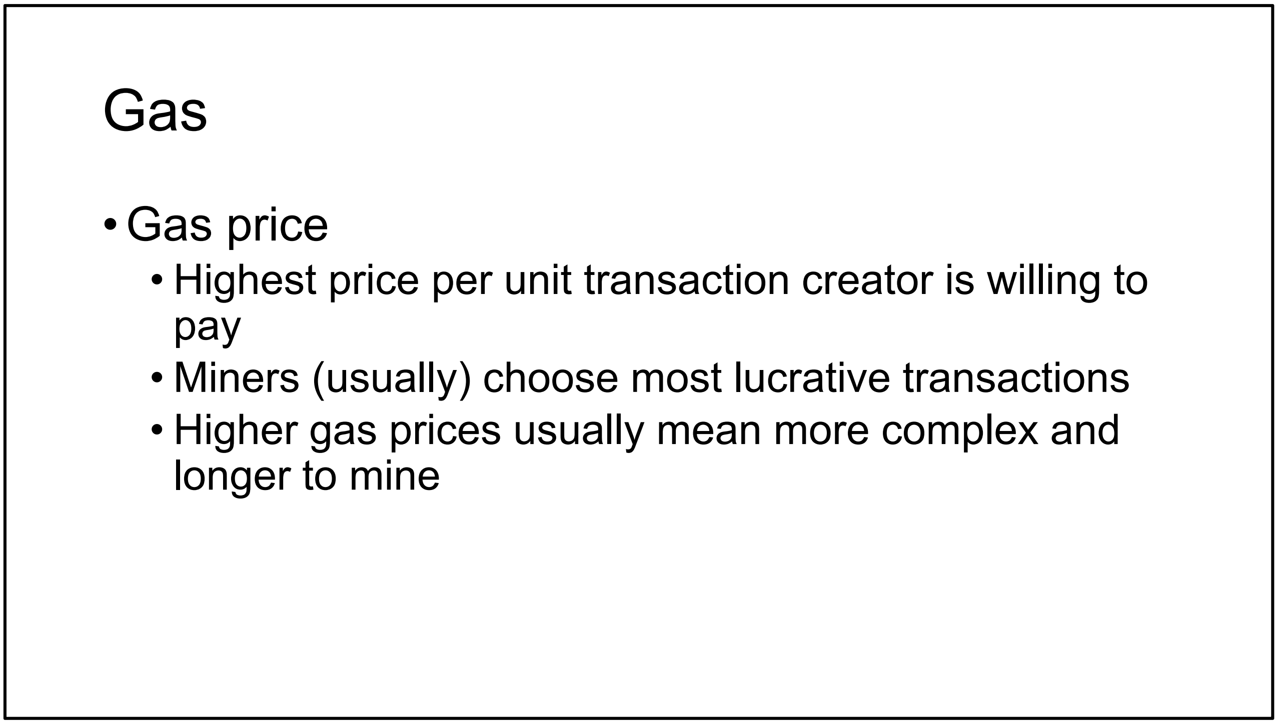


Solidity State Modifier

- `view`
  - Tells the compiler that the function will only reference local variables
  - “I’m not touching the blockchain!”
  - Saves gas

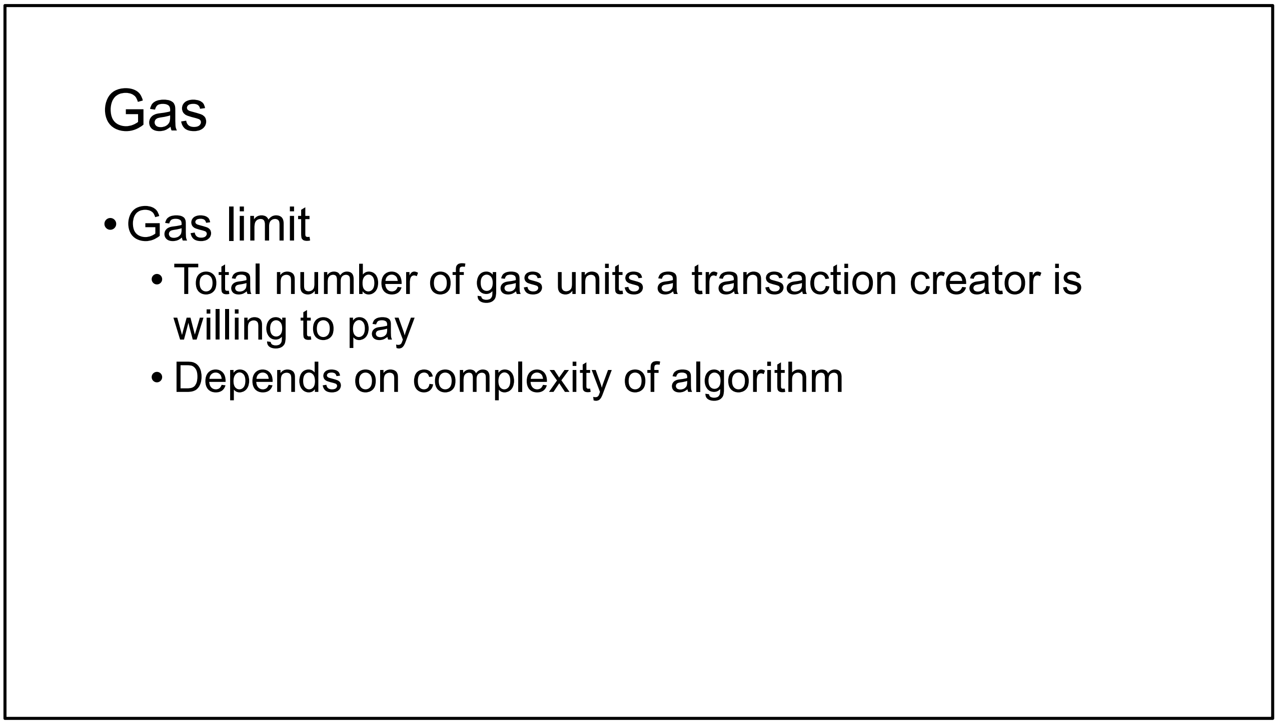
Episode 7.12

Revisiting Gas



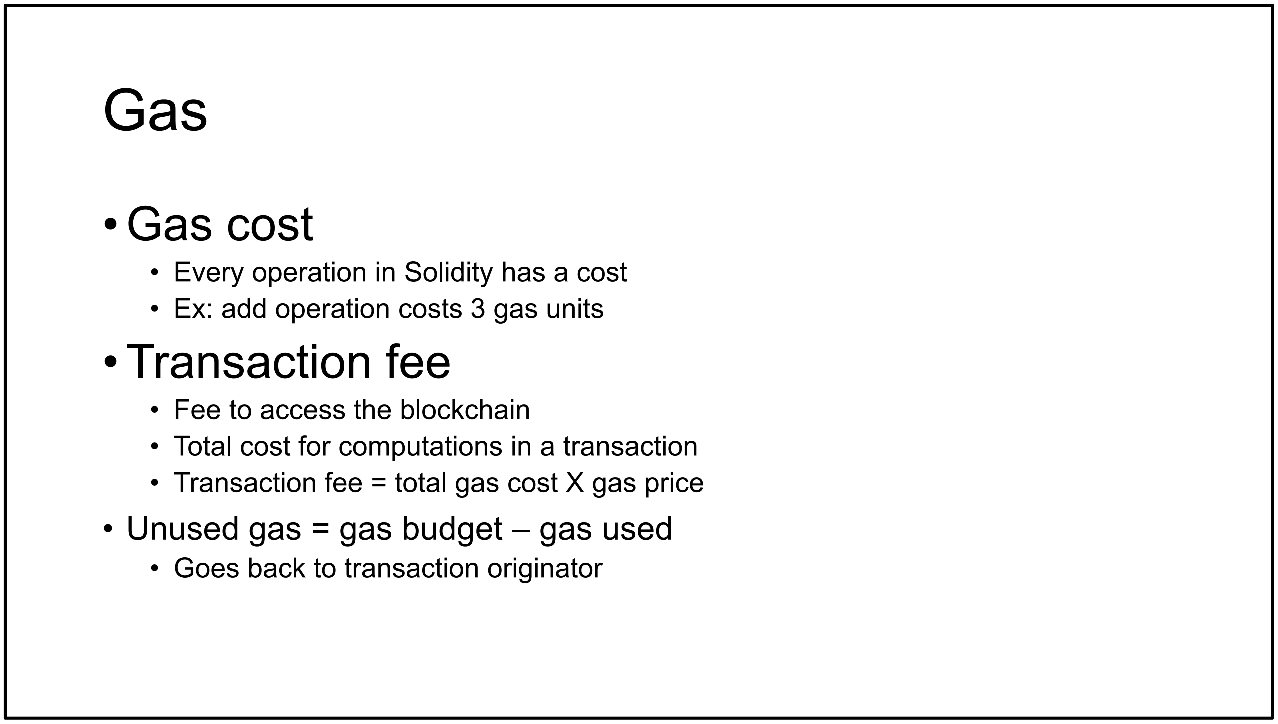
Gas

- Gas price
  - Highest price per unit transaction creator is willing to pay
  - Miners (usually) choose most lucrative transactions
  - Higher gas prices usually mean more complex and longer to mine



Gas

- Gas limit
  - Total number of gas units a transaction creator is willing to pay
  - Depends on complexity of algorithm

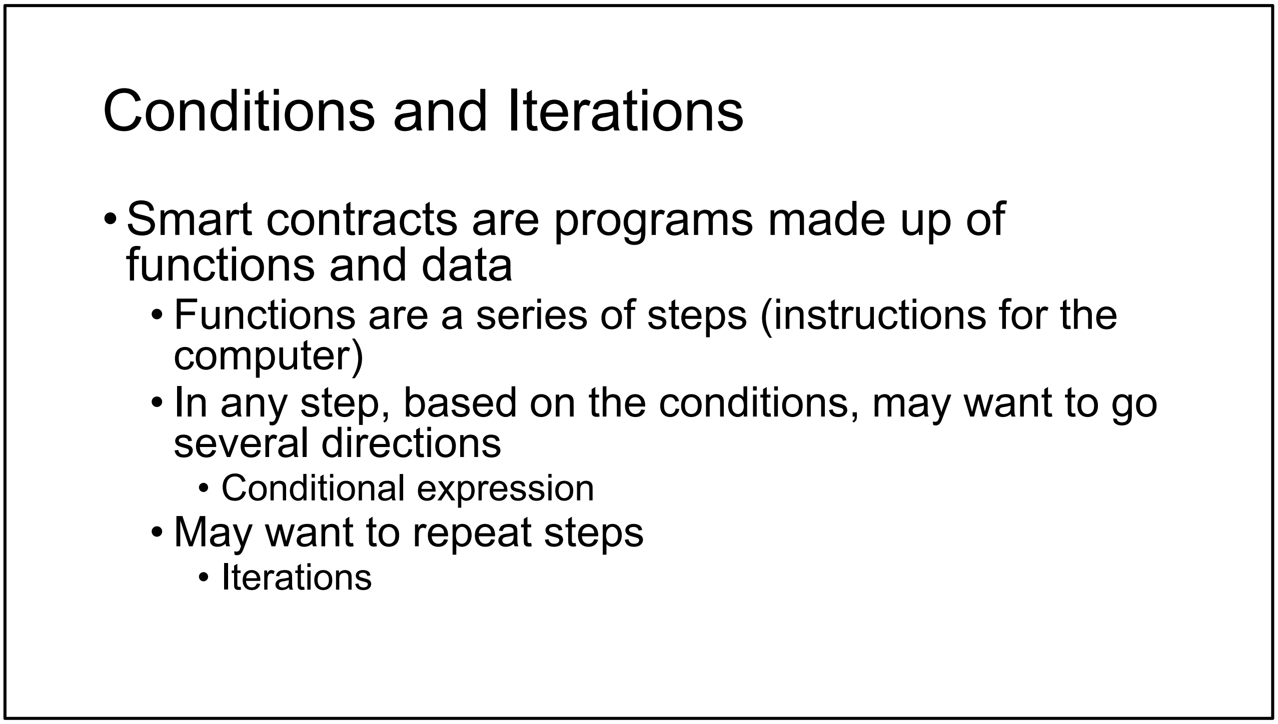


Gas

- Gas cost
  - Every operation in Solidity has a cost
  - Ex: add operation costs 3 gas units
- Transaction fee
  - Fee to access the blockchain
  - Total cost for computations in a transaction
  - Transaction fee = total gas cost X gas price
- Unused gas = gas budget – gas used
  - Goes back to transaction originator

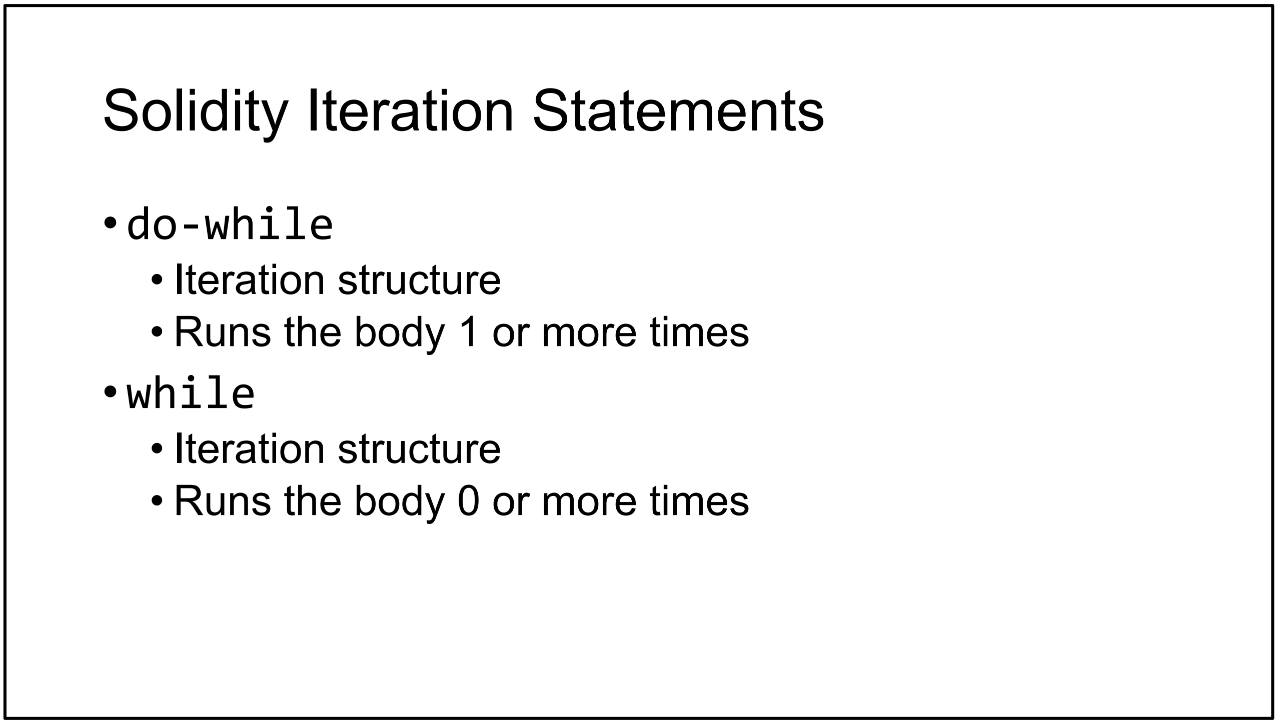
Episode 7.13

Controlling Flow



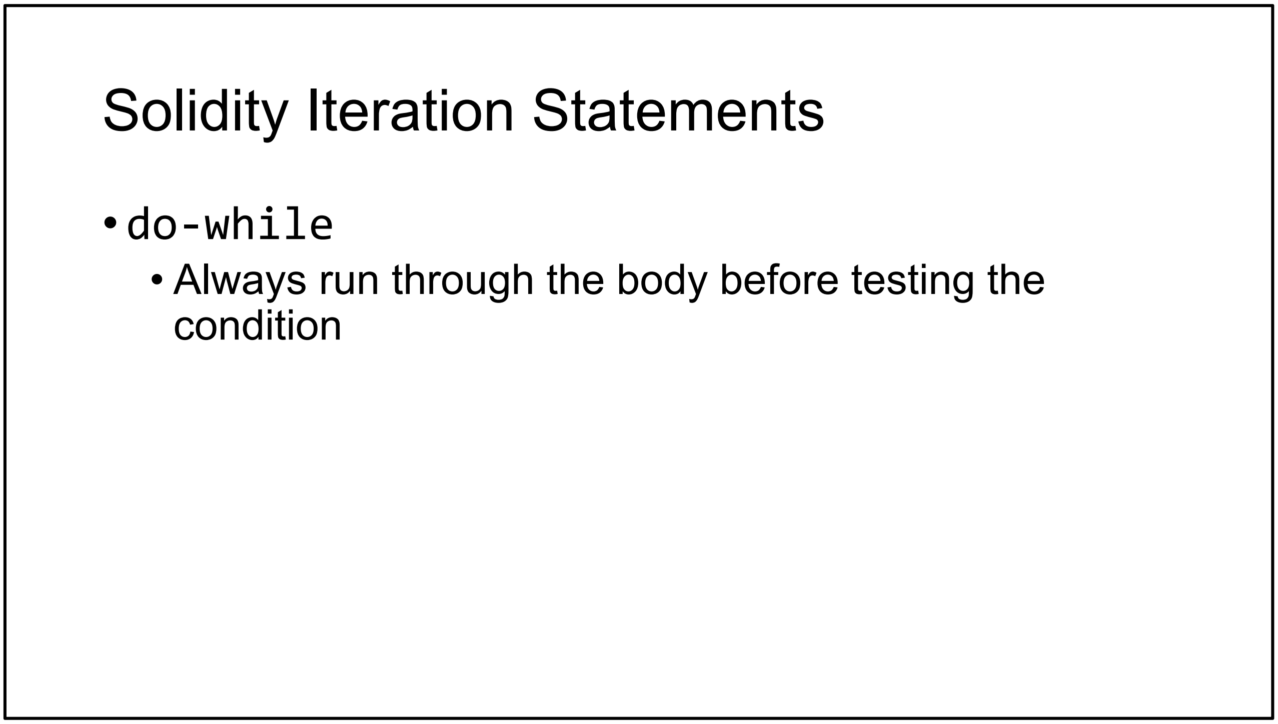
Conditions and Iterations

- Smart contracts are programs made up of functions and data
  - Functions are a series of steps (instructions for the computer)
  - In any step, based on the conditions, may want to go several directions
    - Conditional expression
  - May want to repeat steps
    - Iterations



Solidity Iteration Statements

- `do-while`
  - Iteration structure
  - Runs the body 1 or more times
- `while`
  - Iteration structure
  - Runs the body 0 or more times

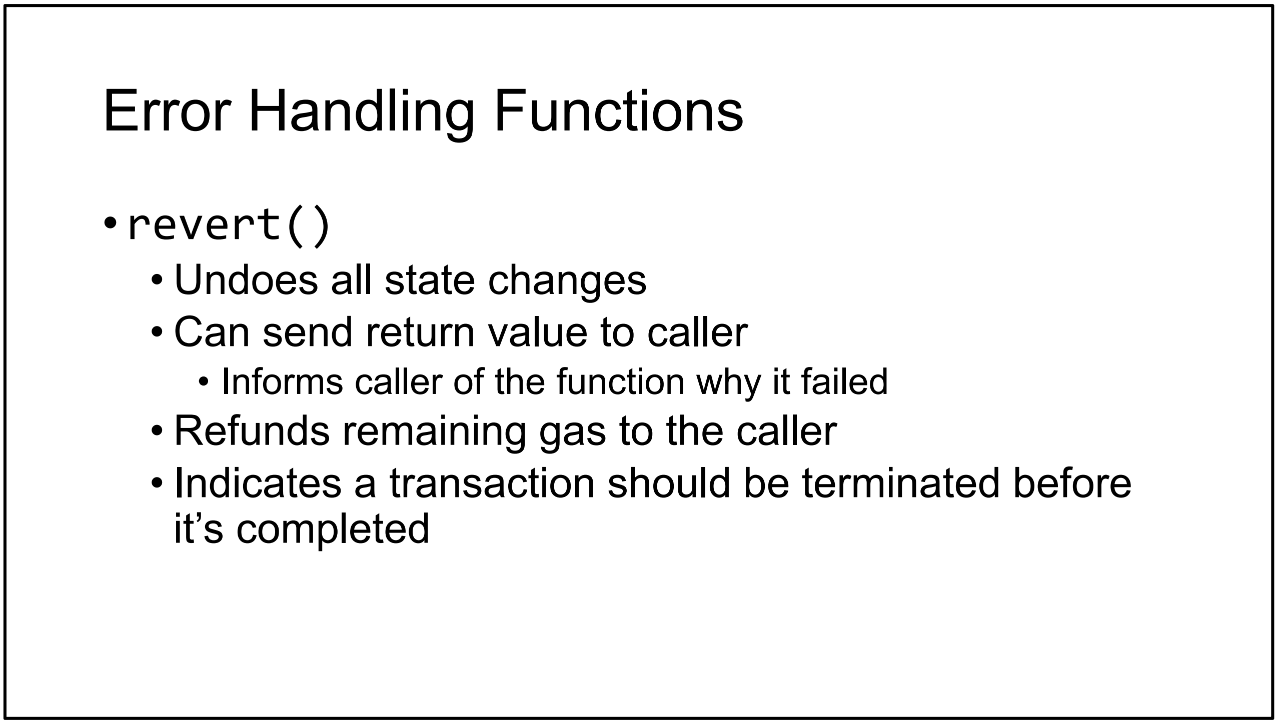


Solidity Iteration Statements

- `do-while`
  - Always run through the body before testing the condition

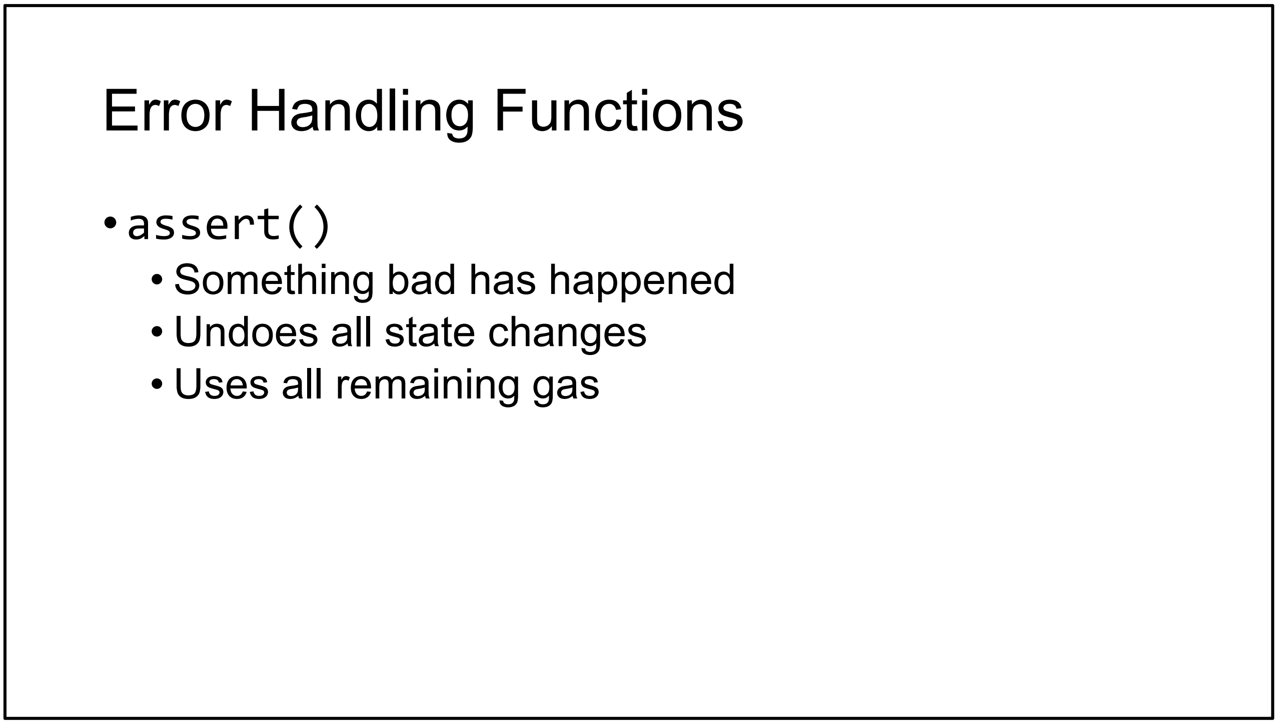
Episode 7.14

Handling Errors



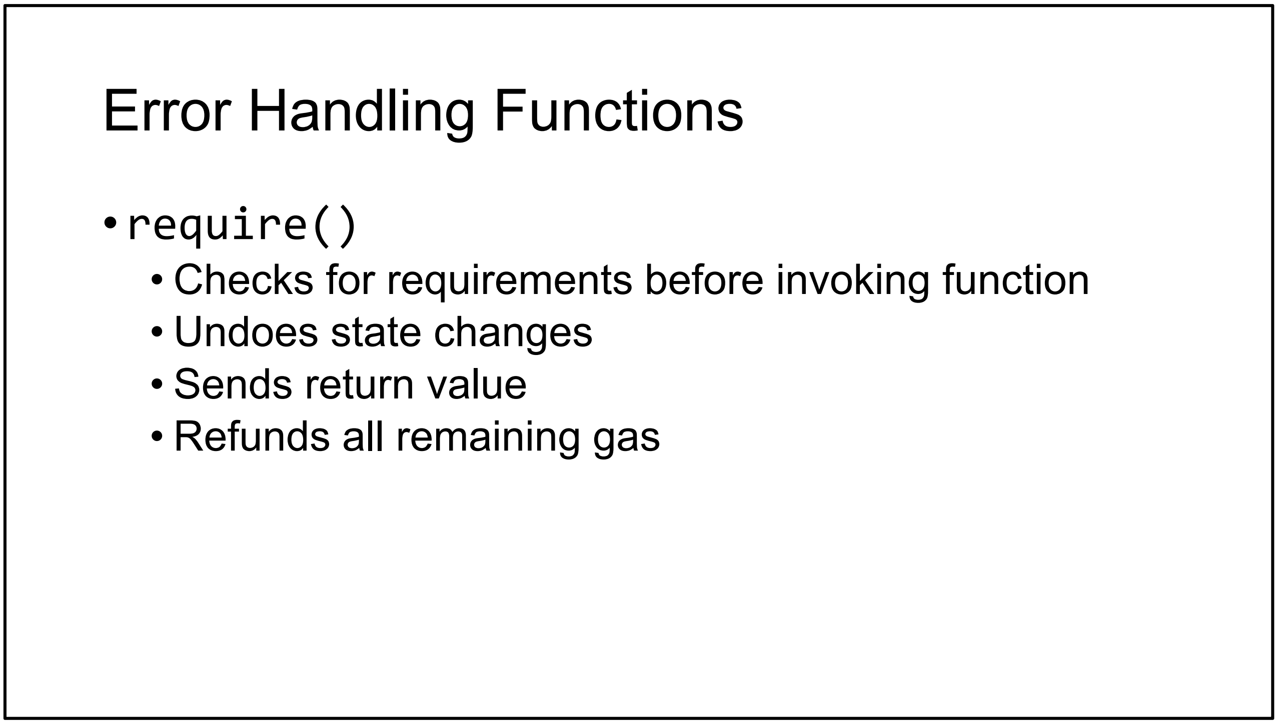
Error Handling Functions

- `revert()`
  - Undoes all state changes
  - Can send return value to caller
    - Informs caller of the function why it failed
  - Refunds remaining gas to the caller
  - Indicates a transaction should be terminated before it’s completed



Error Handling Functions

- `assert()`
  - Something bad has happened
  - Undoes all state changes
  - Uses all remaining gas



Error Handling Functions

- `require()`
  - Checks for requirements before invoking function
  - Undoes state changes
  - Sends return value
  - Refunds all remaining gas