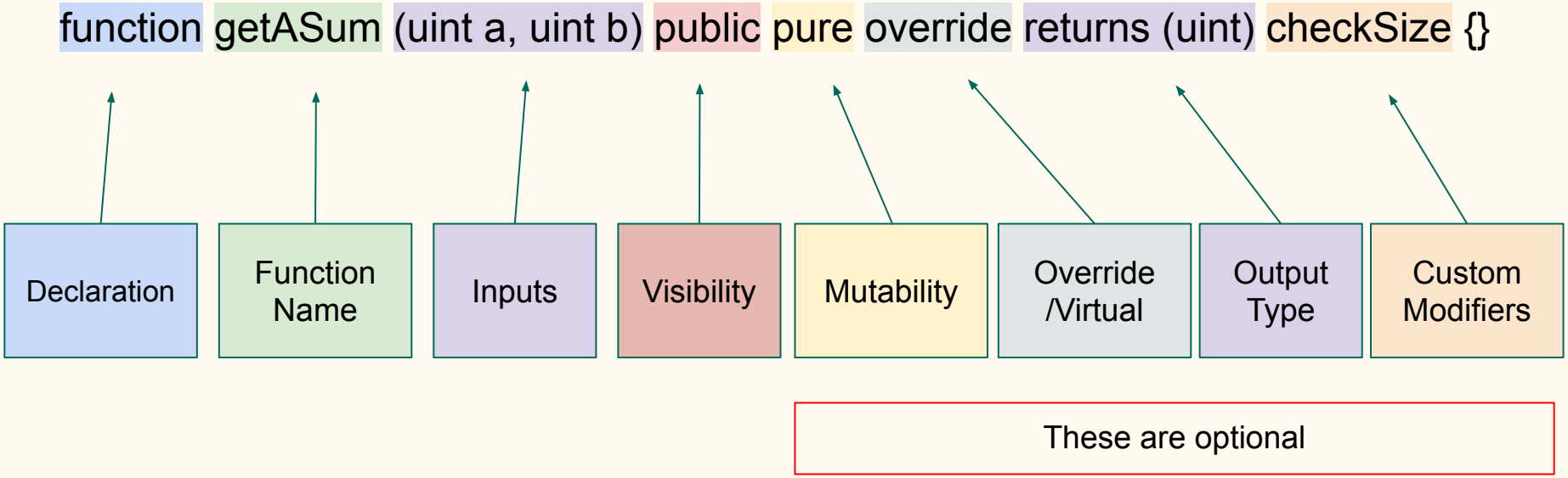
Lecture 3

Contracts and Functions

Function Deep Dive

Function Convention



Function Visibility

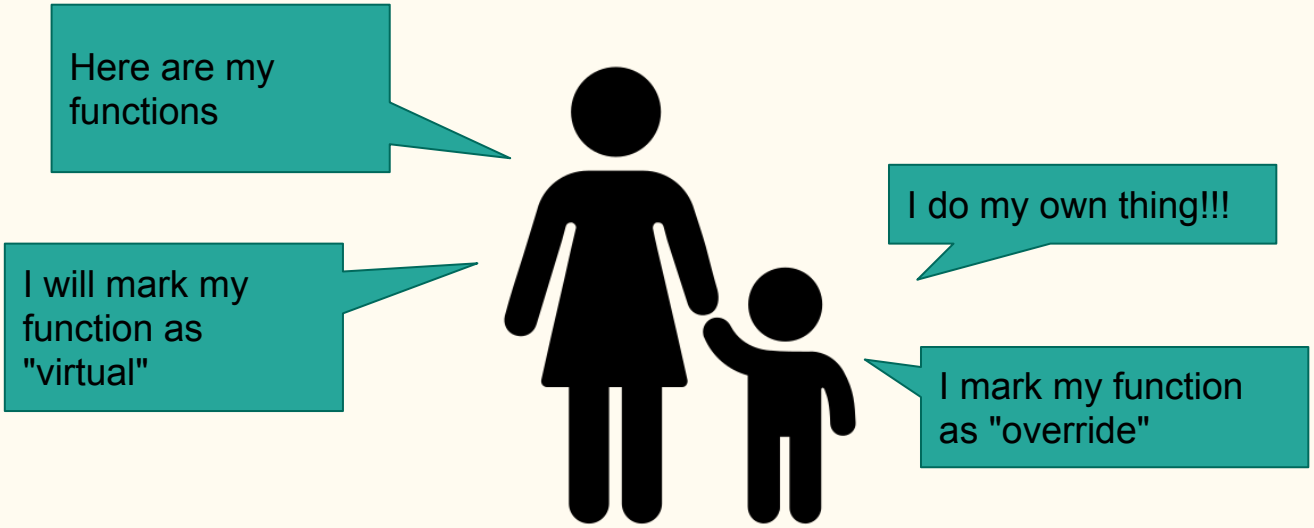
public	**private**
Function is callable by anyone, including other contracts and itself.	Function can only be executed within its contract. Cannot be inherited or externally called.
internal	**external**
Function can only be called by its contract and the contracts which inherit the contract it lives in.	Function will only ever be called by external users and contract, not by the contract itself internally. It is cheaper than public functions as it accesses calldata.

Function Mutability

pure	view
No state variable will be changed or read.	State variable will be read but not changed.

Combined with "external", the function is free!

Function inheritance - virtual / override



Interface Functions are automatically virtual

Private Functions cannot be inherited

In the case of **Multiple Inheritance**, with parents who have functions with the same name, child contract MUST override the function with the same name!

Custom Function Modifiers

```
pragma solidity ^0.5.0;

contract Owner {
    address owner;
    constructor() public {
        owner = msg.sender;
    }
    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }
    modifier costs(uint price) {
        if (msg.value >= price) {
            _;
        }
    }
```

Declare using the "**modifier**" keyword.
_; signals start of main function

Modifiers properties:

- Can be inherited
- Can take arguments
- Many in same function
- Can be overriden

Function Overloading

```
function sameName(int a){}

function sameName(int a, string b){}

function sameName(int a, int b){}
```

A function with multiple, different inputs but the same function name is an overloaded function.

Functions may be overload with:

- A different number of inputs
- Different types of inputs

How does this work?

Application Binary Interface

Encoding communication between SCs

ABI Encoding standard

- First 4 Bytes, always the hash of function signature. (Function clash possible!)
  - FuncName(type1, type2) : HelloWorld(uint, bool)
- Following that, the input encoding, different for value and reference types.
  - Value types: each input allocated 32 Bytes, value directly passed in
  - Depending on different type, either left or right aligned.
  - Reference types: First the memory location is passed in, length, then the values.

Example: How many Bytes would these functions have?

HelloWorld(uint, bytes[2], bool) -> Input 1993, "hi" , true
HelloWorld(uint[], bool) -> [1993, 1994], true

Example Solution

HelloWorld(uint, uint, bool): 4 + 3 x 32 = 100 Bytes

Function Selector: bytes4(keccak256(bytes(function_signature)))
1993: 00000000000000000000000000000000000000000000000000000000000007c9
hi: 6869000000000000000000000000000000000000000000000000000000000000
True: 0000000000000000000000000000000000000000000000000000000000000001

HelloWorld(uint[], bool): 4 + 32 + 32 + 32 + 2x32 = 164 Bytes

Function selector + memory location of unit[] + bool + length of array + values in array

Useful ABI functions

abi.encode() // abi.decode()

abi.encodePacked() -> no zero padding, no length, no memory offset, no decode equivalent due to ambiguity

abi.encodeWithSignature() -> very useful in contract calls!

Special Functions

Send, Transfer, Call - low level functions

receivingAddress.transfer(amount);

- Success - 2300 gas unit payment
- Failure - Revert

receivingAddress.send(amount);

- Success - 2300 gas unit payment
- Failure - Returns bool. Handled by initiating SC

(bool success, bytes memory data)= receivingAddress.call{gas: limit; value: wei}("");

- Can call specific function signature (ie. different payment methods)
- Empty function directly calls fallback or receive function
- Can specify gas limit or use all gas in calling contract. Enables complex operations
- Function params encoded with abi.encodeWithSignature
- **What happens when the called contract makes a call to the original calling contract?**

Fallback, Receive

receive() external payable{}

- First line of defense

fallback() external payable{}

- Second line of defense

- Does NOT contain the function keyword
- MUST be external and payable
- Worst case scenario: 2300 Gcallstipend
- Can be virtual overridden and contain modifiers

See it in action with Solidity Scripting

Validation and Assertion

- require(cond), require(cond, error_msg) is used to check inputs
- revert(), revert(msg) is used to abort execution given internal errors
- assert(cond) is used to prevent logical errors like integer overflow

```
function withdraw(uint256 amount) public {
    require(balance[msg.sender] >= amount, "insufficient balance");
    if (!msg.sender.send(amount)) {
        revert("transfer failed");
    }
    balance[msg.sender] -= amount;
}
```

Source: https://www.linkedin.com/pulse/require-vs-revert-assert-solidity-arslan-maqbool/

Globally Available Variables

Block, Transaction

Block properties

blockhash(uint blockNumber) returns (bytes32): hash of the given block when blocknumber is one of the 256 most recent blocks; otherwise returns zero

block.basefee (uint): current block's base fee (EIP-3198 and EIP-1559)

block.blobbasefee (uint): current block's blob base fee (EIP-7516 and EIP-4844)

block.chainid (uint): current chain id

block.coinbase (address payable): current block miner's address

block.difficulty (uint): current block difficulty (EVM < Paris).

block.gaslimit (uint): current block gaslimit

block.number (uint): current block number

block.prevrandao (uint): random number provided by the beacon chain (EVM >= Paris)

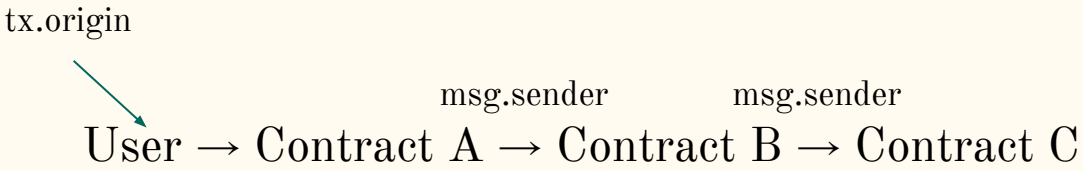
block.timestamp (uint): current block timestamp as seconds since unix epoch

Transaction Properties

tx.gasprice (uint): gas price of the transaction

tx.origin (address): sender of the transaction (full call chain)

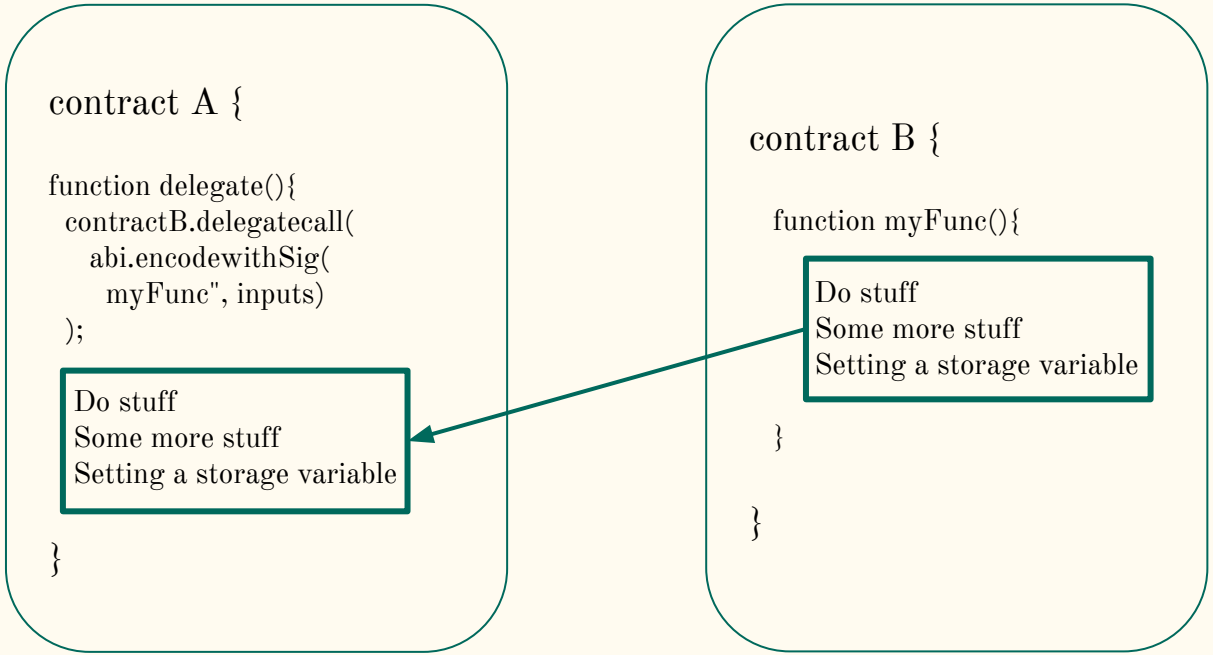
msg.data (bytes calldata): complete calldata
msg.sender (address): sender of the message (current call)
msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)
msg.value (uint): number of wei sent with the message



Delegatecall - A Special Call

contractB.delegatecall(abi.encodeWithSignature("myFunc()", inputs));

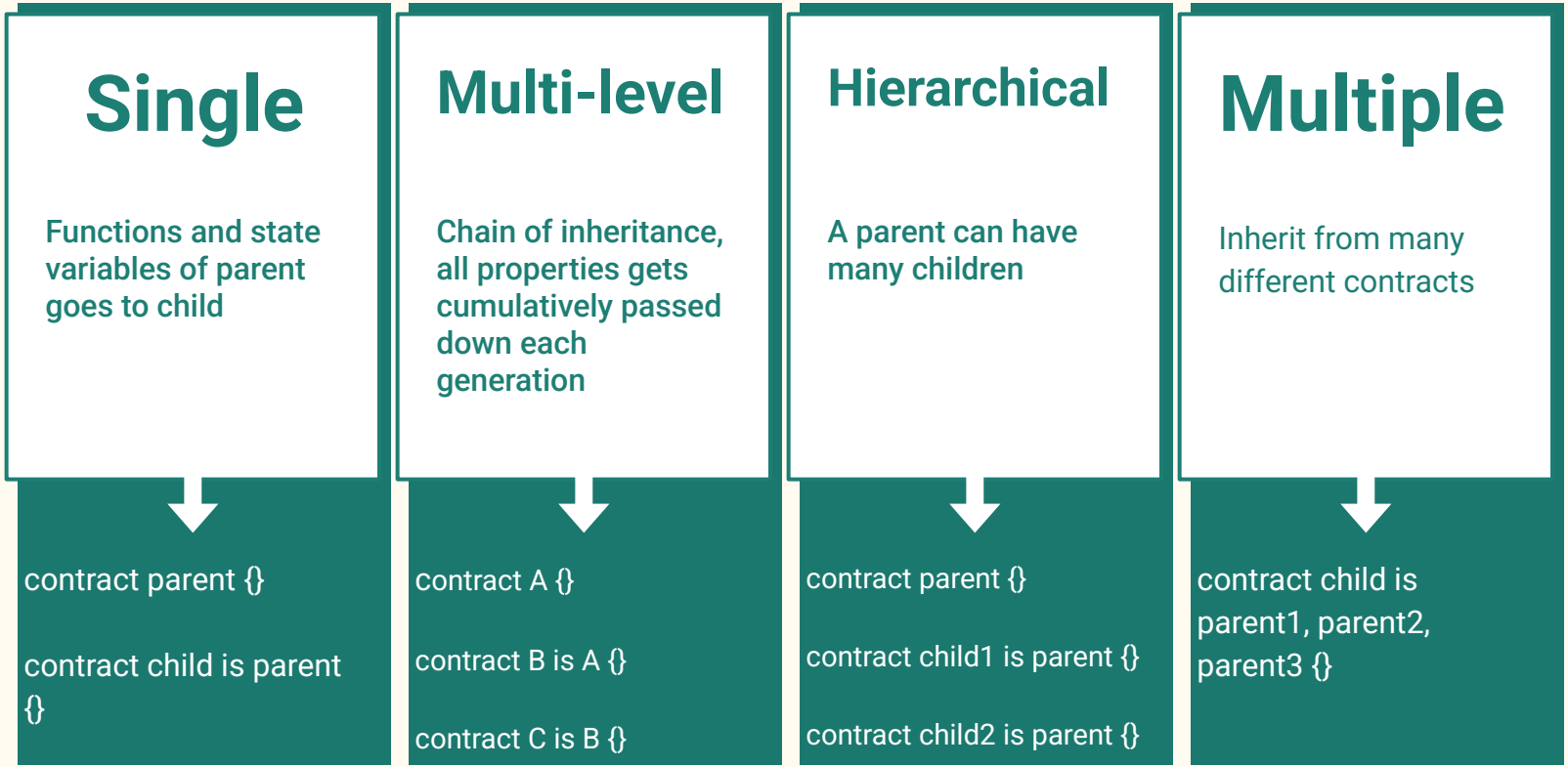
tx.origin = msg.sender



Smart Contract properties

Inheritance and Memory

Inheritance - sharing is caring



Inheritance - Interfaces and Abstract Contracts

Interface

- Only used as templates
- No function implementation logic
- No constructors or state variables
- Only external functions, automatically virtual
- Cannot inherit

Abstract Contract

- Used as base for other contracts. is itself a contract
- Mix of implemented and unimplemented functions
- Can have constructor and state variable
- can inherit

```
interface InheritMe{
        function getBalance(address _user) external view{}
}
```

Interfaces - Birds of a feather, work together

Variable Scope in Smart Contracts

```
contract MyContract {
        int public a;
        mapping(address => int) ownerBalances;

        function checkBalance(address _owner) external view {
                int b = 5;
                return b + ownerBalances[_owner];
        }
}
```

- ➔ State variables are always stored onchain.
- ➔ They are inherited

- ➔ Local variables are in transient memory.
- ➔ They are not inherited

Memory Management in Smart Contracts

Memory

- Lasts for the length of a function call
- Stored on EVM
- Values are mutable
- Most versatile

Calldata

- Extremely tiny
- User/contract inputs
- Immutable
- Use as safe guard

Storage

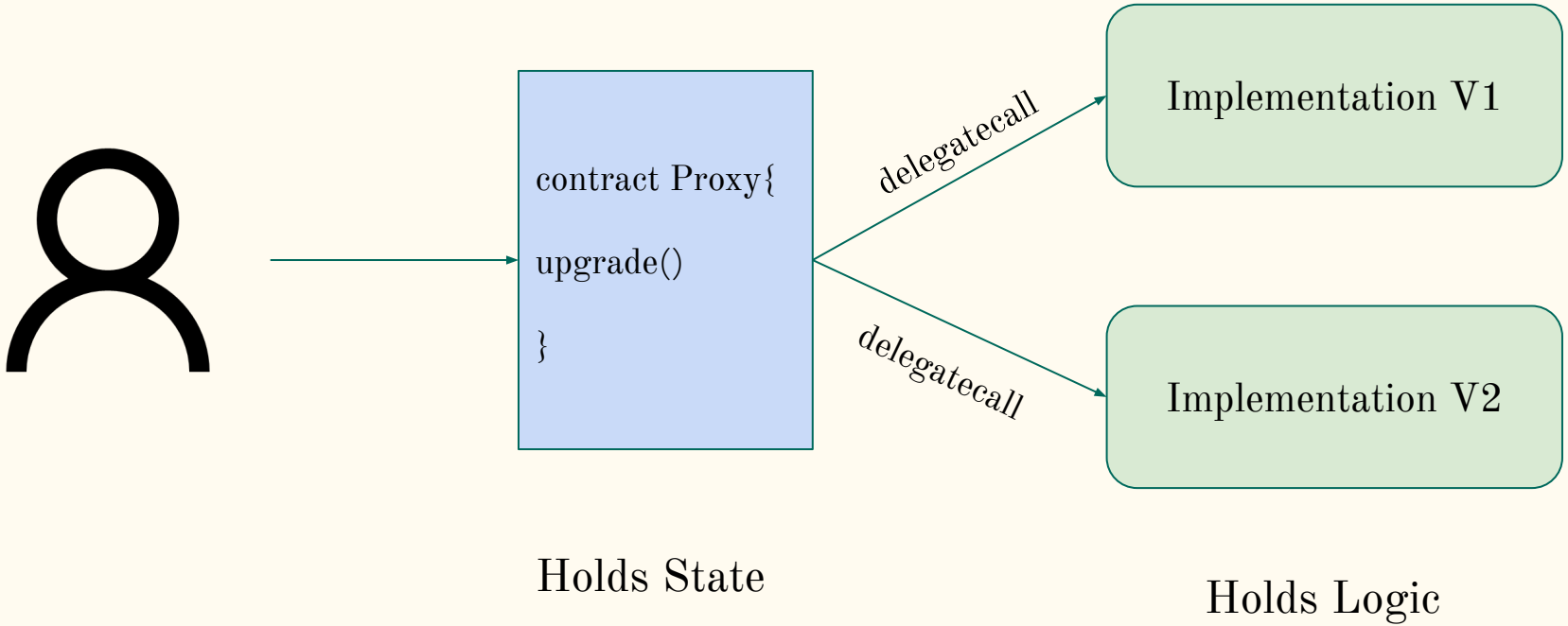
- Permanent on chain storage
- Extremely expensive!!
- Can use push function
- Use for state variables only

Stack

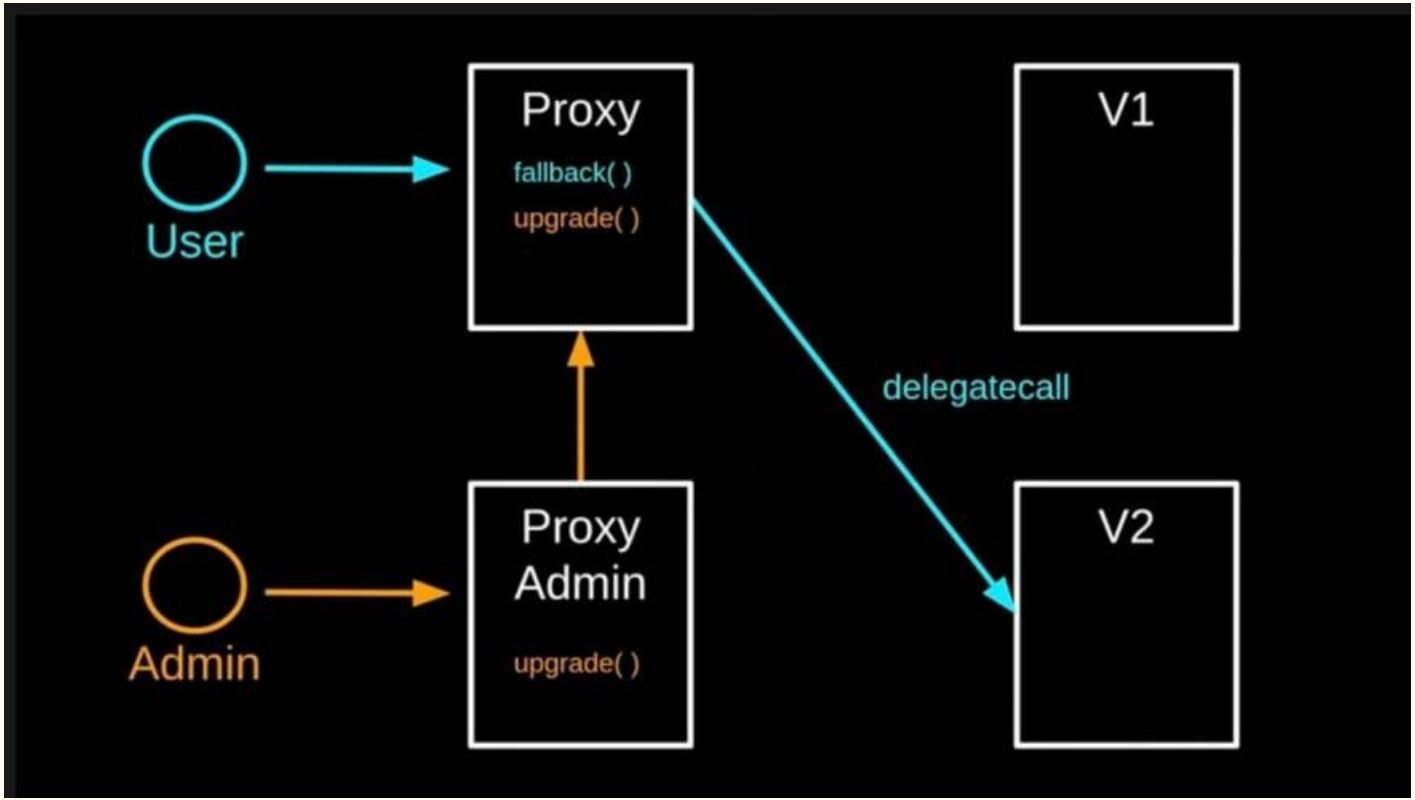
- Very small
- Stores things for immediate execution
- Gas Efficient
- **Extremely painful** -> inline assembly

Upgradeable Contracts

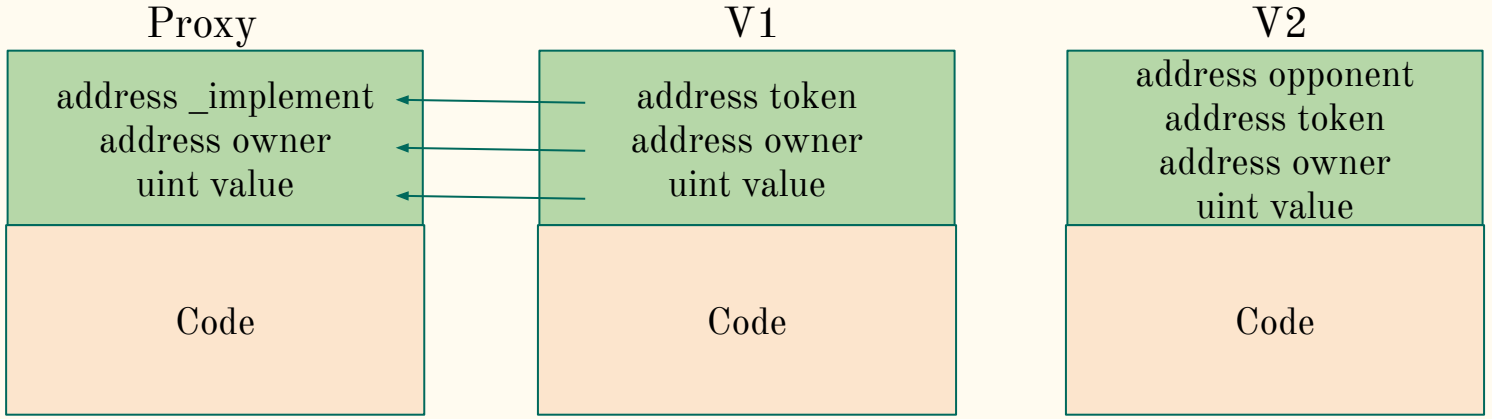
Upgradable Proxy Architecture



Transparent Proxy Model



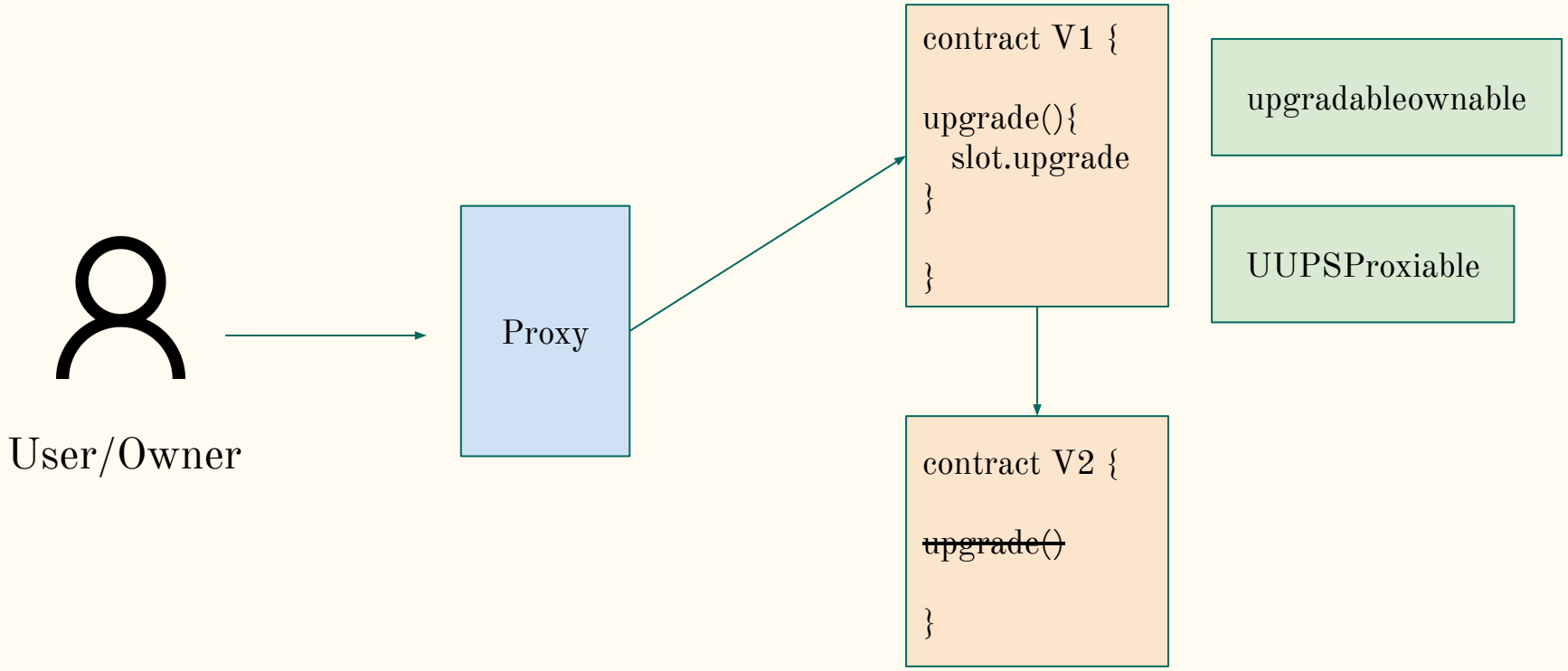
Storage Collisions




```
bytes32 private constant implementationPosition = bytes32(uint256(
  keccak256('eip1967.proxy.implementation')) - 1
));
```

Always append state vars, not modify

UUPS Proxy Model - EIP 1822

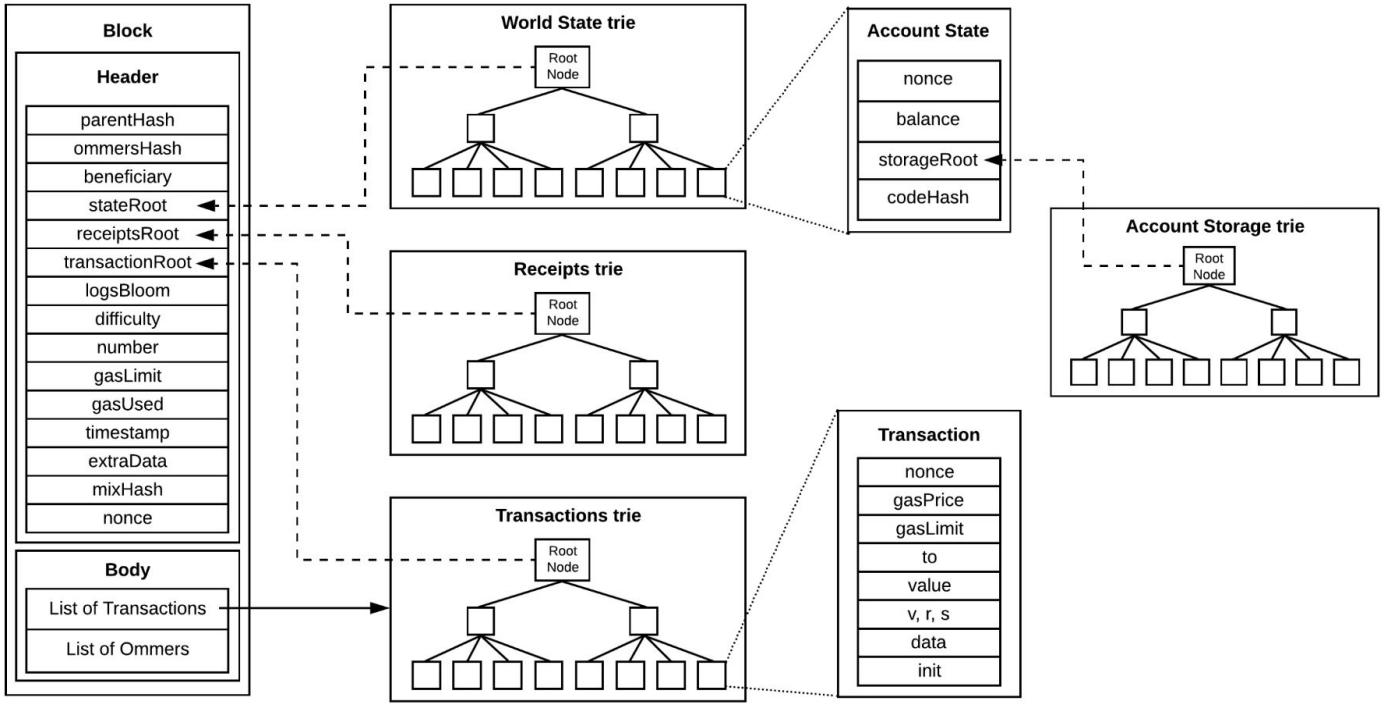


Cost Comparison

	Transparent	UUPS
Proxy Deployment	740k + 480k ProxyAdmin	390k
Implementation Deployment	+ 0	+ 320k
Runtime Overhead	7.3k	4.9k

Solidity Events

Ethereum Accounting - Tries



Block, transaction, account state objects and Ethereum tries

Events - Transaction Record

- Security Audits
  - Traditional accounting - does the stored variable add up to all events?

- User Interface
  - Confirm a transaction has happened
  - Change the interface after confirmation

- Cryptographic checks
  - Bloom filters

Event - Declaration and Actualization

```
contract Transaction {
    event makeATransfer(address indexed _from, address indexed _to, uint amount);

    function payRent(address receiver, uint deposit) external {
        require(msg.sender.balance >= msg.value);
        emit makeATransfer(msg.sender, receiver, amount);
    }
}
```

Events - ABI representation


```
{
        "returnValues": {
                "_from": "0x1111...FFFFCCCC",
                "_to": "0x50...sd5adb20",
                 "amount": "0x420042"
        },
        "raw": {
                "data": "0x7f...91385",
                "topics": ["0xfd4...b4ead7", "0x7f...1a91385","0xf28...d21297" ]
        }
}
```