18 NOVEMBER 2025

The Death of the __gap

Custom Storage Layouts

WHOAMI

Eric Marti Haynes

Tokenization Tech Lead

@Nethermind

Upgradeable Smart Contracts

The Problem

Standard smart contracts are immutable. You can't patch vulnerabilities or add new features.

The Solution

Upgradeable pattern: use a Proxy contract that points to a separate Logic contract, that can be replaced.

The Benefit

The contract's address and all its data stay the same. You can change the contract's logic without changing its address.

BugCounter.sol (v1)

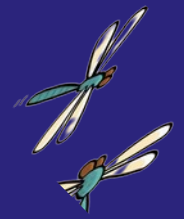
```sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {ArthropodCounter} from "./ArthropodCounter.sol";

contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public worms;

    constructor() {
        _disableInitializers();
    }

    function initialize(address _owner) public override initializer {
        __Ownable_init(_owner);
        __UUPSUpgradeable_init();
    }

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms;
    }

    function _authorizeUpgrade(address newImplementation) internal override onlyOwner {}
}
```

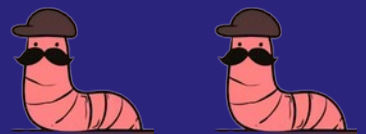
ArthropodCounter.sol (v1)

```sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;

    constructor() {
        _disableInitializers();
    }

    function initialize(address _owner) public virtual initializer {
        __Ownable_init(_owner);
    }

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function arthropods() public view returns (uint256) {
        return insects;
    }
}
```







Variable	Slot	Value
uint256 insects	0	0x07
uint256 worms	1	0x03

BEFORE

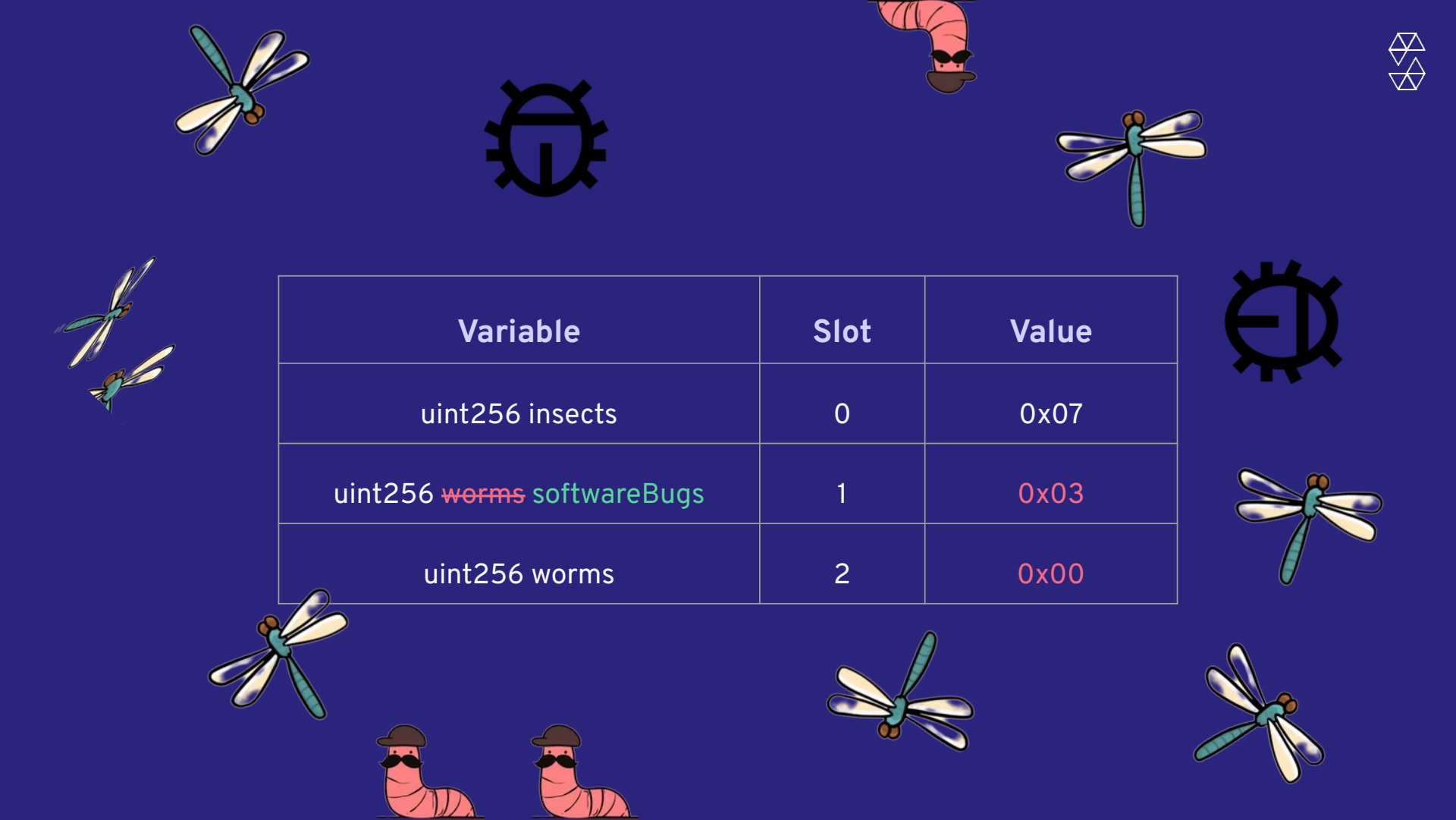
BugCounter.sol (v1)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public worms;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms;
    }
}
```

AFTER

BugCounter.sol (v2.1)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public softwareBugs;
    uint256 public worms;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount)
        external
        onlyOwner
    {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms + softwareBugs;
    }
}
```



Variable	Slot	Value
uint256 insects	0	0x07
uint256 ~~worms~~ softwareBugs	1	0x03
uint256 worms	2	0x00

BEFORE

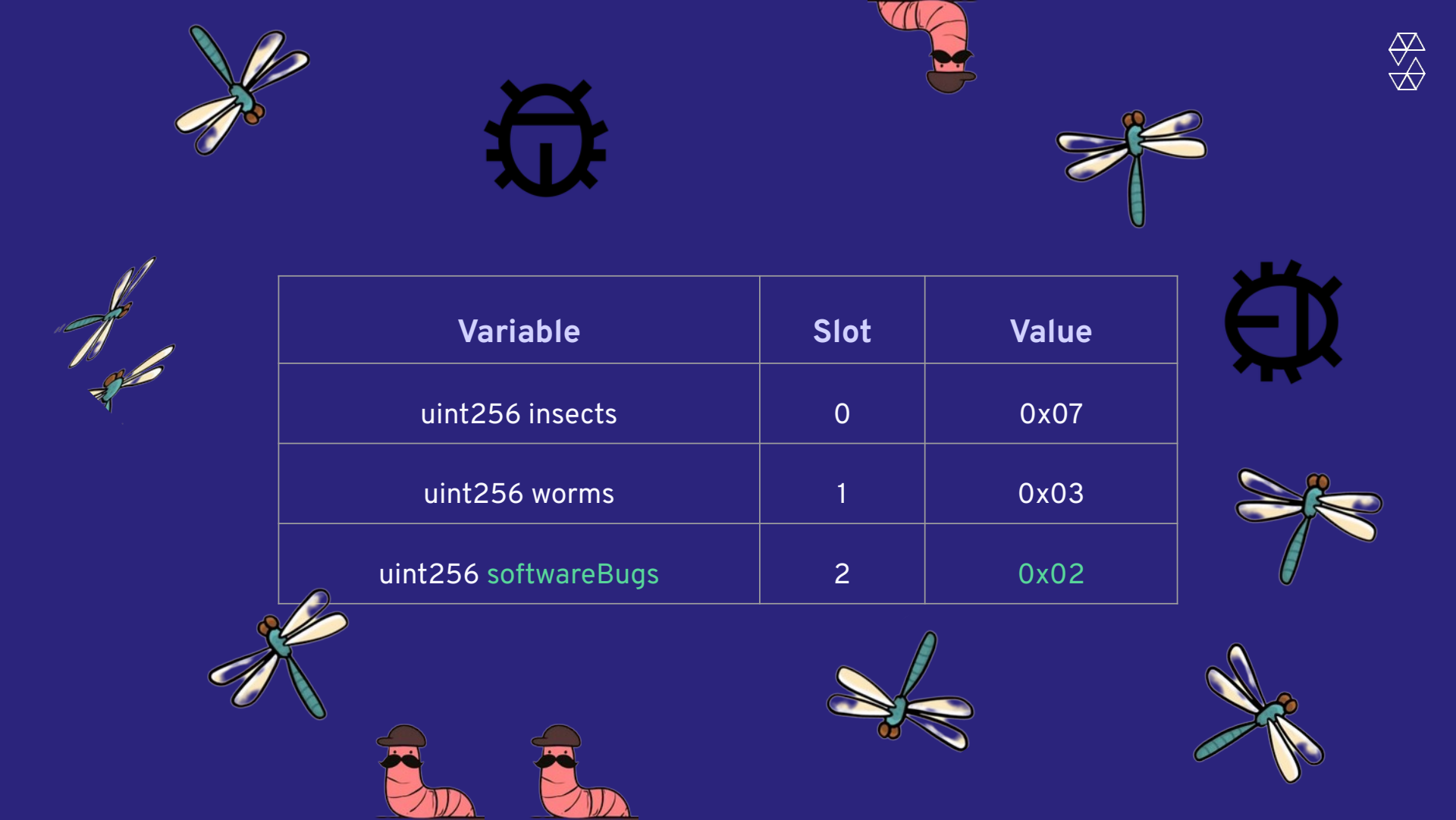
BugCounter.sol (v1)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public worms;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms;
    }
}
```

AFTER

BugCounter.sol (v2.2)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public worms;
    uint256 public softwareBugs;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount)
        external
        onlyOwner
    {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms + softwareBugs;
    }
}
```



Variable	Slot	Value
uint256 insects	0	0x07
uint256 worms	1	0x03
uint256 softwareBugs	2	0x02

BEFORE

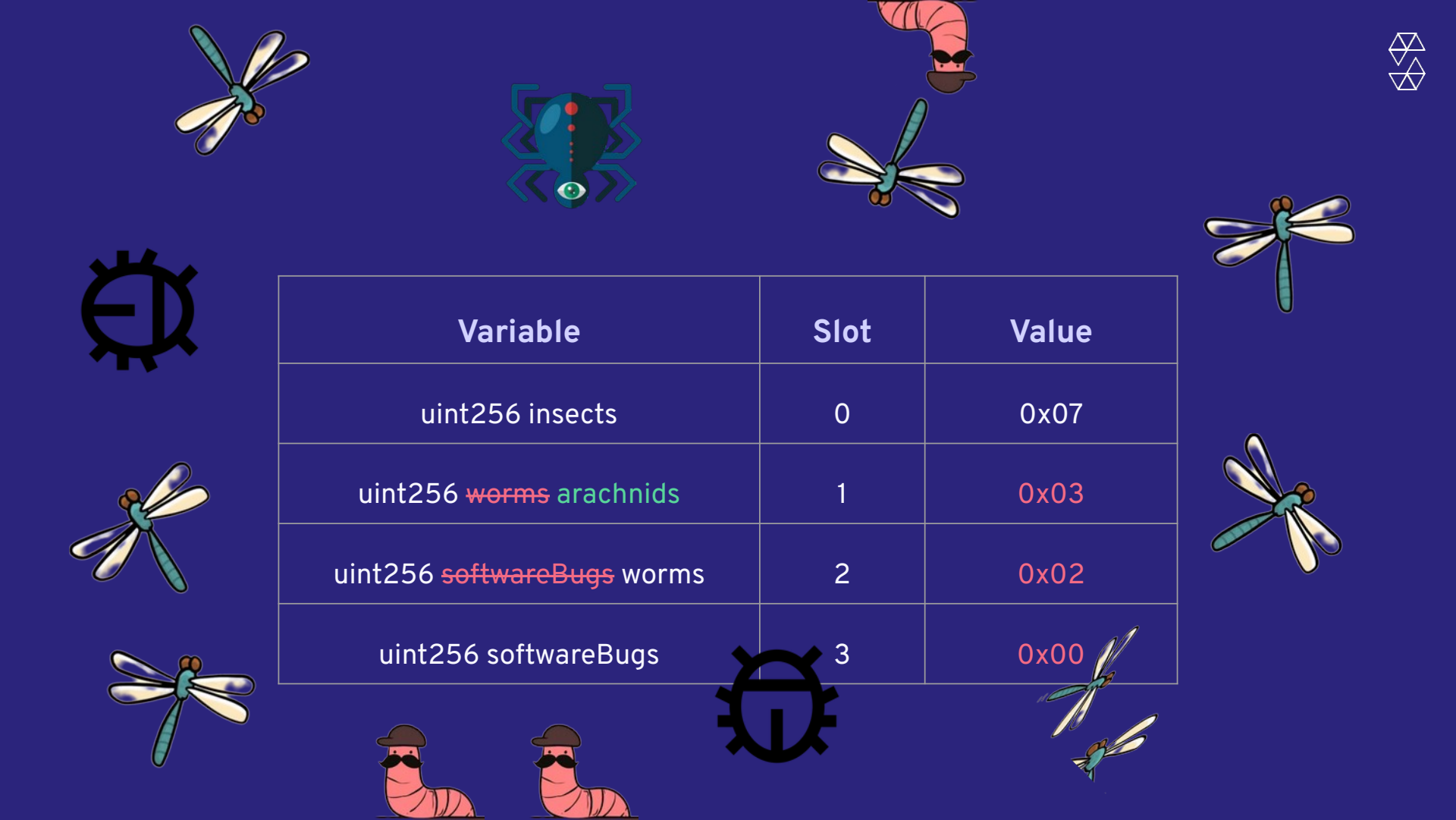
ArthropodCounter.sol (v1)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function arthropods() public view returns (uint256) {
        return insects;
    }
}
```

AFTER

ArthropodCounter.sol (v2.2)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;
    uint256 public arachnids;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function updateArachnids(uint256 _arachnidsCount)
        external
        onlyOwner
    {
        arachnids = _arachnidsCount;
    }

    function arthropods() public view returns (uint256) {
        return insects + arachnids;
    }
}
```



Variable	Slot	Value
uint256 insects	0	0x07
uint256 ~~worms~~ arachnids	1	0x03
uint256 ~~softwareBugs~~ worms	2	0x02
uint256 softwareBugs	3	0x00

Botched Smart Contract Upgrades

1. Additional time & resources
2. Corrupt storage & DoS
3. Exploits & loss of funds

> *Audius, web3's answer to Spotify, has fallen victim to a governance attack, losing $6M of its native token, AUDIO.*
>
> – REKT, 2022

AUDIUS

Storage Gaps

History

Introduced and popularized by the OpenZeppelin team around 2019.

Purpose

Preventing storage slot collisions in upgradeable proxy contracts.

Mechanism

Reserving empty storage slots for future variables, through large empty arrays at the end of contracts.

BEFORE

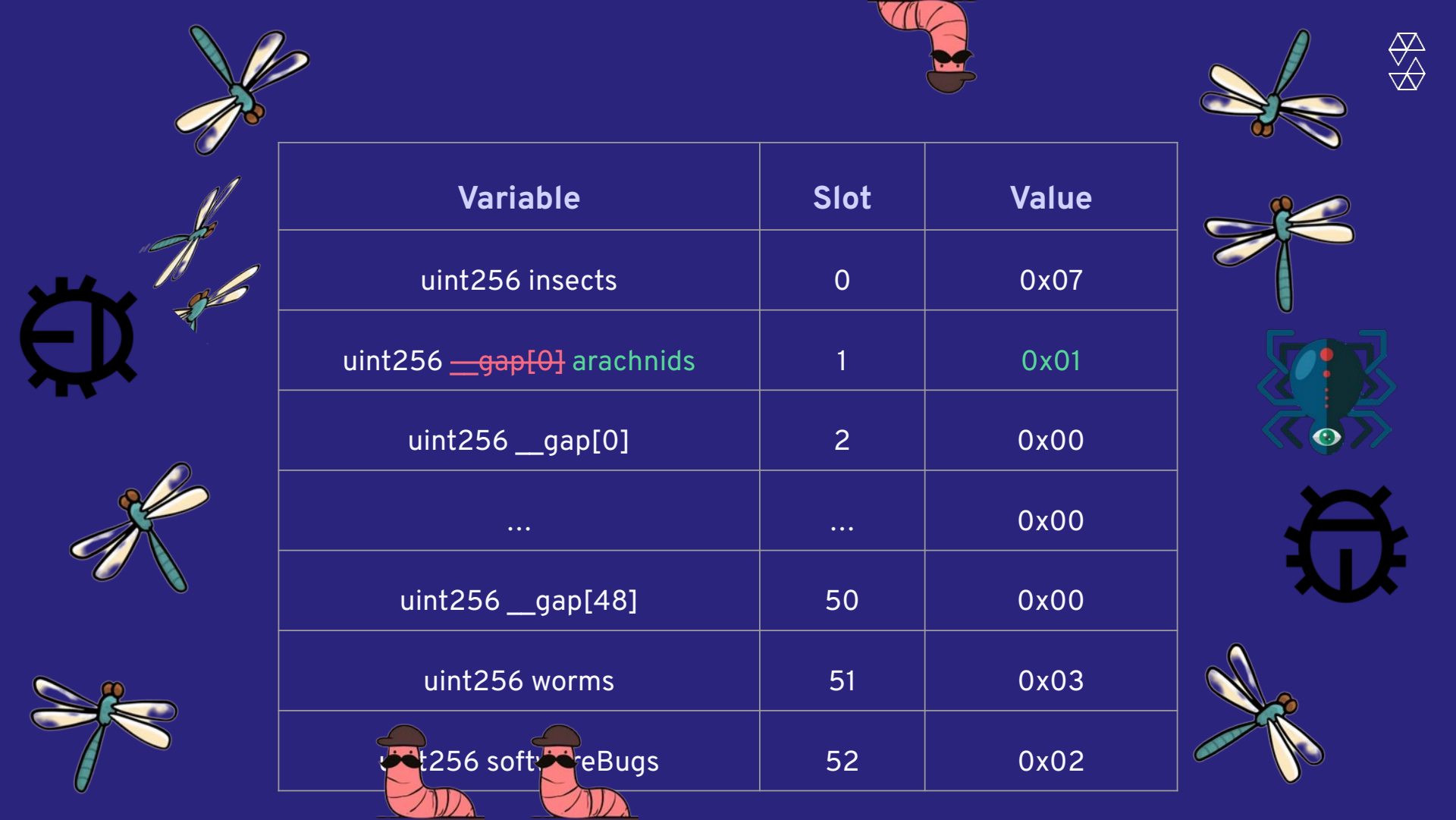
ArthropodCounter.sol (v1a)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;
    uint256[50] private __gap;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function arthropods() public view returns (uint256) {
        return insects;
    }
}
```

AFTER

ArthropodCounter.sol (v2.3)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;
    uint256 public arachnids;
    uint256[49] private __gap;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function updateArachnids(uint256 _arachnidsCount)
        external
        onlyOwner
    {
        arachnids = _arachnidsCount;
    }

    function arthropods() public view returns (uint256) {
        return insects + arachnids;
    }
}
```



Variable	Slot	Value
uint256 insects	0	0x07
uint256 ~~__gap[0]~~ arachnids	1	0x01
uint256 __gap[0]	2	0x00
...	...	0x00
uint256 __gap[48]	50	0x00
uint256 worms	51	0x03
uint256 softwareBugs	52	0x02

Storage Gap Issues

- Fixed Size
- Manual Updates
- Inheritance Tree

BEFORE

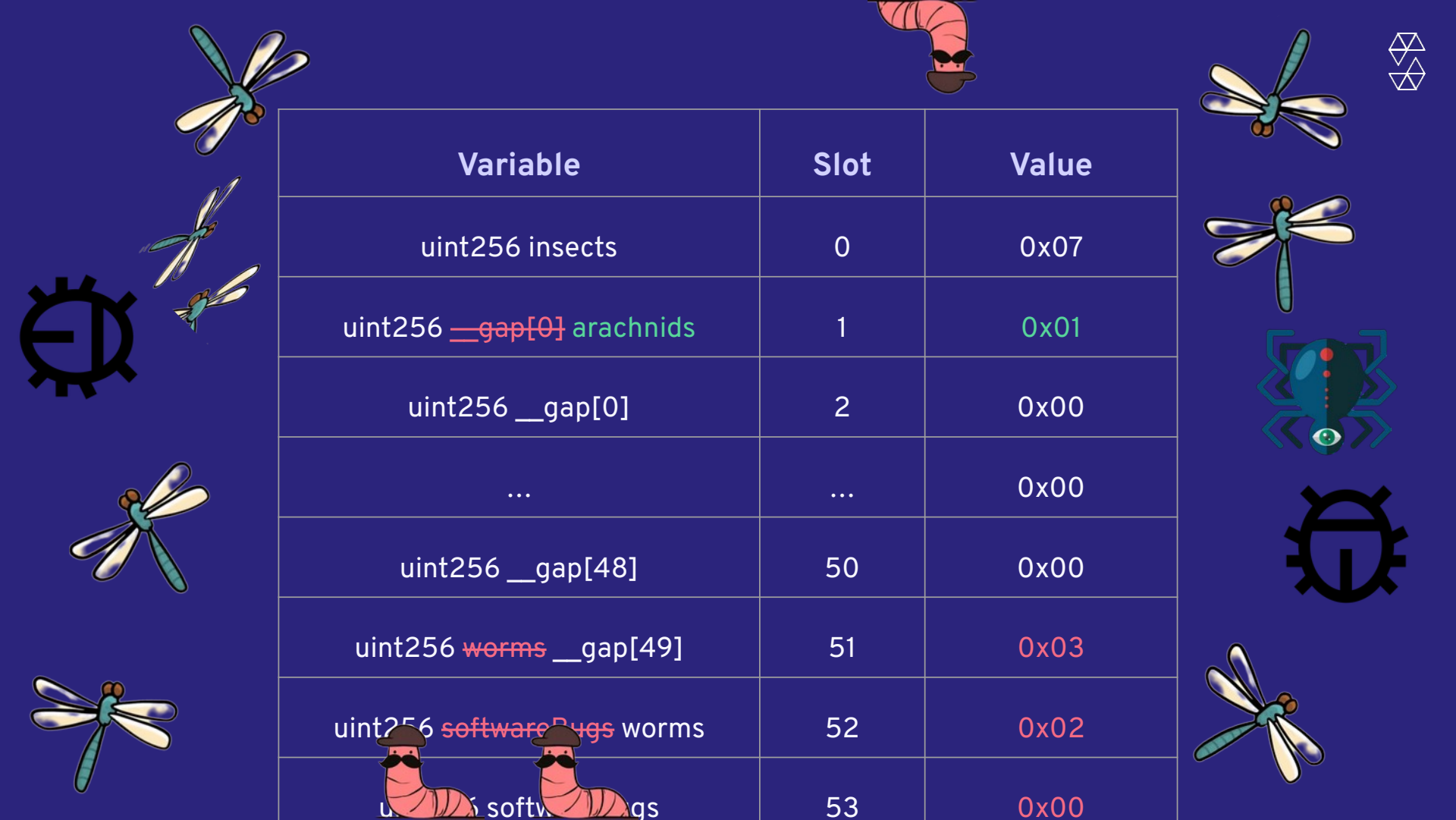
ArthropodCounter.sol (v1a)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;
    uint256[50] private __gap;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function arthropods() public view returns (uint256) {
        return insects;
    }
}
```

AFTER

ArthropodCounter.sol (v2.3a)

```
contract ArthropodCounter is OwnableUpgradeable {
    uint256 public insects;
    uint256 public arachnids;
    uint256[50] private __gap;

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function updateArachnids(uint256 _arachnidsCount)
        external
        onlyOwner
    {
        arachnids = _arachnidsCount;
    }

    function arthropods() public view returns (uint256) {
        return insects + arachnids;
    }
}
```



Variable	Slot	Value
uint256 insects	0	0x07
uint256 ~~__gap[0]~~ arachnids	1	0x01
uint256 __gap[0]	2	0x00
...	...	0x00
uint256 __gap[48]	50	0x00
uint256 ~~worms~~ __gap[49]	51	0x03
uint256 ~~softwareBugs~~ worms	52	0x02
u... soft...gs	53	0x00

BEFORE

BugCounter.sol (v2.3)

```sol
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    uint256 public worms;
    uint256 public softwareBugs;
    uint256[50] private __gap;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount)
        external
        onlyOwner
    {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + worms + softwareBugs;
    }
}
```

AFTER

BugCounter.sol (v3.1)

```sol
contract BugCounter is
    UUPSUpgradeable,
    ArthropodCounter,
    GastropodCounter
{
    uint256 public worms;
    uint256 public softwareBugs;
    uint256[50] private __gap;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount)
        external
        onlyOwner
    {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + gastropods() + worms + softwareBugs;
    }
}
```

GastropodCounter.sol (v3.1)

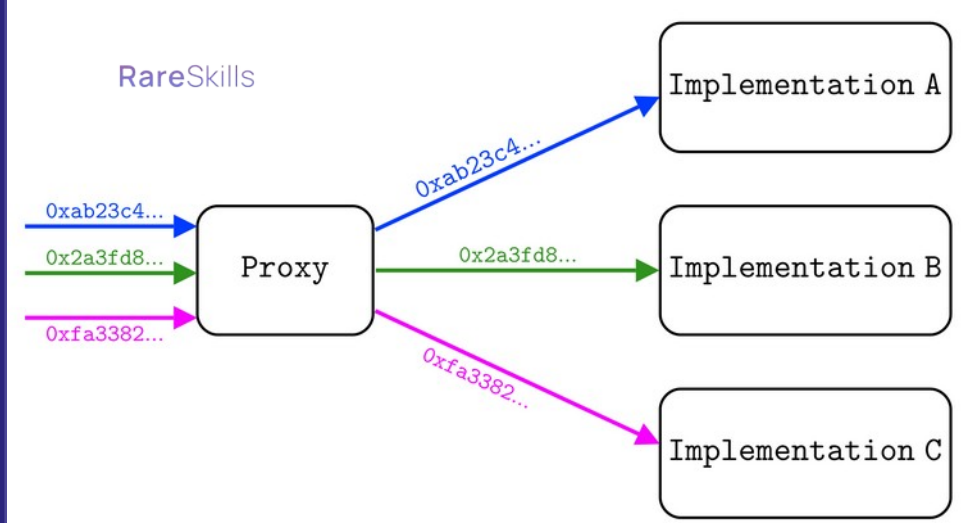
```sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract GastropodCounter is OwnableUpgradeable {
    uint256 public snails;
    uint256 public slugs;
    uint256[50] private __gap;

    constructor() {
        _disableInitializers();
    }

    function initialize(address _owner) public virtual initializer {
        __Ownable_init(_owner);
    }

    function updateSnails(uint256 _snailCount) external onlyOwner {
        snails = _snailCount;
    }

    function updateSlugs(uint256 _slugCount) external onlyOwner {
        slugs = _slugCount;
    }

    function gastropods() public view returns (uint256) {
        return snails + slugs;
    }
}
```

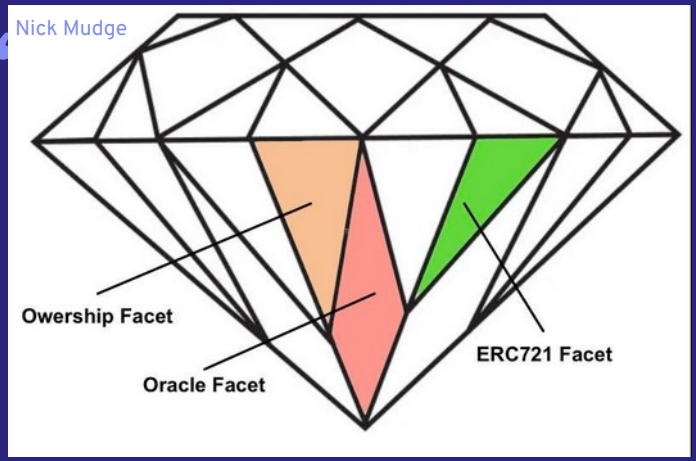
Variable	Slot	Value
uint256 insects	0	0x07
uint256 arachnids	1	0x01
uint256[49] __gap	2→50	0x00
uint256 ~~worms~~ snails	51	0x03
uint256 ~~softwareBugs~~ slugs	52	0x02
uint256[50] __gap	53→102	0x00
uint256 worms	103	0x0
u softwa	104	

ERC-2535: Diamonds, Multi-Facet Proxy

Create modular smart contract systems that can be extended after deployment

This proposal standardizes diamonds, which are modular smart contract systems that can be upgraded/extended after deployment, and have virtually no size limit. More technically, a diamond is a contract with external functions that are supplied by contracts called facets. Facets are separate, independent contracts that can share internal functions, libraries, and state variables.





Diamond Storage Formula

SLOT = keccak256("*myproject.erc20*")

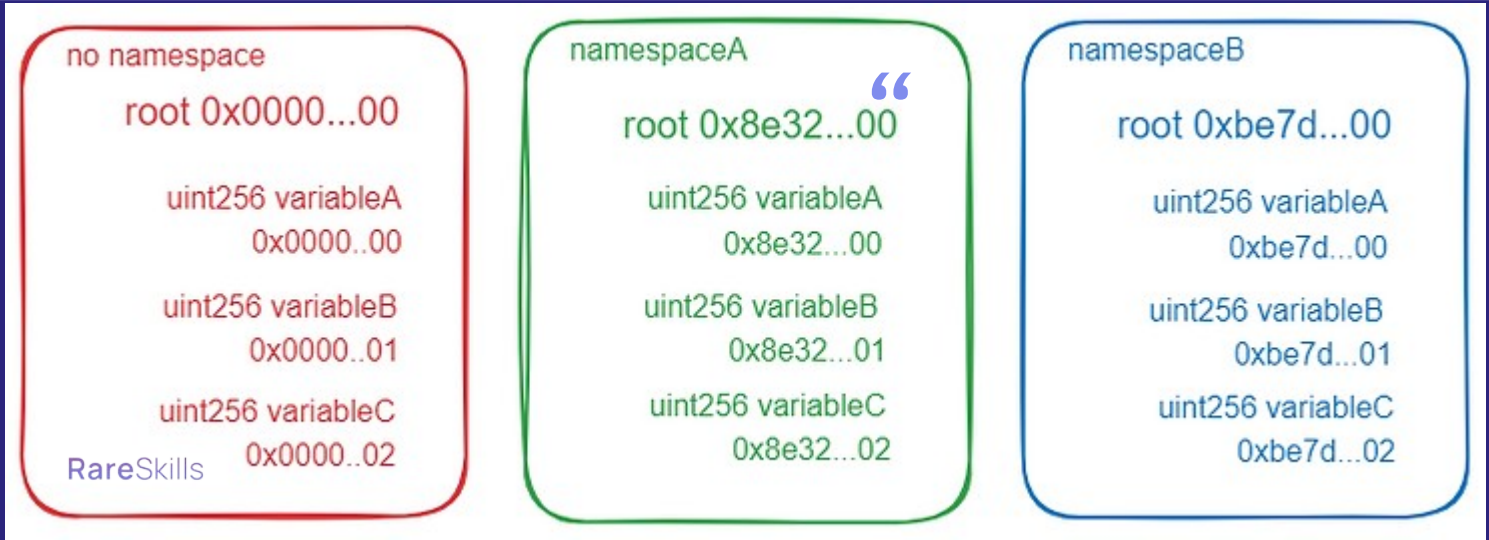
Diamond Storage

1. Storage identifiers based on simple string literals
2. Use not limited to Diamond Multi-Facet Proxies
3. Currently being standardized with ERC-8042
4. Can be used as an alternative to storage gaps

ERC-7201: Namespaced Storage Layout

Conventions for the storage location of structs in the namespaced storage pattern.

We define the NatSpec annotation @custom:storage-location *to document storage namespaces and their location in storage in Solidity or Vyper source code. Additionally, we define a formula to derive a location from an arbitrary identifier. The formula is chosen to be safe against collisions with the storage layouts used by Solidity and Vyper.*



Namespaced Storage Formula

SLOT = keccak256(keccak256(“*myproject.erc20*”) - 1) & ~0xff

Namespaced Storage

1. Multiple hashing guarantee that namespace slots are completely disjoint from standard storage
2. Namespace locations are aligned to 256 as a potential future gas optimization
3. Standardized with ERC-7201
4. Can be used as an alternative to storage gaps

BEFORE

BugCounter.sol (v3.1a)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter {
    /// @custom:storage-location erc7201:Bug.Storage
    struct BugStorage {
        uint256 worms;
        uint256 softwareBugs;
    }
    // keccak256(abi.encode(uint256(keccak256("Bug.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant BUG_STORAGE_SLOT =
0x7085942646c8710272fe508c90646d04206417736c67889e4571dd854ed65f00;
    function updateWorms(uint256 _wormCount) external onlyOwner {
        BugStorage storage bcs = _getBugStorage();
        bcs.worms = _wormCount;
    }
    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        BugStorage storage bcs = _getBugStorage();
        bcs.softwareBugs = _softwareBugCount;
    }
    function worms() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return bcs.worms;
    }
    function softwareBugs() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return bcs.softwareBugs;
    }
    function bugs() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return arthropods() + bcs.worms + bcs.softwareBugs;
    }
    function _getBugStorage() private pure returns (BugStorage storage $) {
        assembly {
            $.slot := BUG_STORAGE_SLOT
        }
    }
}
```

AFTER

BugCounter.sol (v3.2)

```sol
contract BugCounter is UUPSUpgradeable, ArthropodCounter, GastropodCounter {
    /// @custom:storage-location erc7201:Bug.Storage
    struct BugStorage {
        uint256 worms;
        uint256 softwareBugs;
    }
    // keccak256(abi.encode(uint256(keccak256("Bug.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant BUG_STORAGE_SLOT =
0x7085942646c8710272fe508c90646d04206417736c67889e4571dd854ed65f00;
    function updateWorms(uint256 _wormCount) external onlyOwner {
        BugStorage storage bcs = _getBugStorage();
        bcs.worms = _wormCount;
    }
    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        BugStorage storage bcs = _getBugStorage();
        bcs.softwareBugs = _softwareBugCount;
    }
    function worms() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return bcs.worms;
    }
    function softwareBugs() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return bcs.softwareBugs;
    }
    function bugs() public view returns (uint256) {
        BugStorage storage bcs = _getBugStorage();
        return arthropods() + gastropods() + bcs.worms + bcs.softwareBugs;
    }
    function _getBugStorage() private pure returns (BugStorage storage $) {
        assembly {
            $.slot := BUG_STORAGE_SLOT
        }
    }
}
```

Variable	Slot	Value
uint256 insects	0x9e...00	0x07
uint256 arachnids	0x9e...01	0x01
uint256 snails	0xc9...00	0x09
uint256 slugs	0xc9...01	0x05
uint256 worms	0x70...00	0x03
uint256 softwareBugs	0x70...01	0x02

ArthropodCounter.sol (v3.2)

```sol
contract ArthropodCounter is OwnableUpgradeable {
    /// @custom:storage-location erc7201:Arthropod.Storage
    struct ArthropodStorage{
        uint256 insects;
        uint256 arachnids;
    }
    // keccak256(abi.encode(uint256(keccak256("Arthropod.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant ARTHROPOD_STORAGE_SLOT =
0x9ea65aa6b13a05c6c8cef4cb75bc25aee688eda3792f8ab5c5de20a390adea00;
    function updateInsects(uint256 _insectCount) external onlyOwner {
        ArthropodStorage storage acs = _getArthropodStorage();
        acs.insects = _insectCount;
    }
    function updateArachnids(uint256 _arachnidsCount) external onlyOwner {
        ArthropodStorage storage acs = _getArthropodStorage();
        acs.arachnids = _arachnidsCount;
    }
    function insects() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.insects;
    }
    function arachnids() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.arachnids;
    }
    function arthropods() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.insects + acs.arachnids;
    }
    function _getArthropodStorage() private pure returns (ArthropodStorage storage $) {
        assembly {
            $.slot := ARTHROPOD_STORAGE_SLOT
        }
    }
}
```

GastropodCounter.sol (v3.2)

```sol
contract GastropodCounter is OwnableUpgradeable {
    /// @custom:storage-location erc7201:Gastropod.Storage
    struct GastropodStorage{
        uint256 snails;
        uint256 slugs;
    }
    // keccak256(abi.encode(uint256(keccak256("Gastropod.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant GASTROPOD_STORAGE_SLOT =
0xc92446913a76e5f4eee9a73f4f714d93468e0f6a6509e6fe302e36b80fa28d00;
    function updateSnails(uint256 _snailCount) external onlyOwner {
        GastropodStorage storage gcs = _getGastropodStorage();
        gcs.snails = _snailCount;
    }
    function updateSlugs(uint256 _slugCount) external onlyOwner {
        GastropodStorage storage gcs = _getGastropodStorage();
        gcs.slugs = _slugCount;
    }
    function snails() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.snails;
    }
    function slugs() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.slugs;
    }
    function gastropods() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.snails + gcs.slugs;
    }
    function _getGastropodStorage() private pure returns (GastropodStorage storage $) {
        assembly {
            $.slot := GASTROPOD_STORAGE_SLOT
        }
    }
}
```

Namespaced Storage Issues

- Manul Hardcoded Hash
- Usage of Assembly
- Verbose & Complex Syntax

Support for Custom Storage Layouts in Solidity

0.8.29

This Solidity release introduces syntax for relocating contract's storage variables to an arbitrary location.

EIP-7702

With the inclusion of *EIP-7702: Set EOA account code* in the Pectra upgrade, it has now become critical for safe implementation of account abstraction.

BugCounter.sol (v4)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter, GastropodCounter
    /// @custom:storage-location erc7201:Bug.Storage
    // keccak256(abi.encode(uint256(keccak256("Bug.Storage")) - 1)) & ~bytes32(uint256(0xff))
    layout at 0x7085942646c8710272fe508c90646d04206417736c67889e4571dd854ed65f00 {

    uint256 public worms;
    uint256 public softwareBugs;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + gastropods() + worms + softwareBugs;
    }
}
```

Variable	Slot	Value
uint256 insects	0x9e...00	0x07
uint256 arachnids	0x9e...01	0x01
uint256 snails	0xc9...00	0x09
uint256 slugs	0xc9...01	0x05
uint256 worms	0x70...00	0x03
uint256 softwareBugs	0x70...01	0x02


“
Currently the syntax is very limited: the base location can only be a literal expression and applies to the whole inheritance tree.
—SOLIDITY TEAM

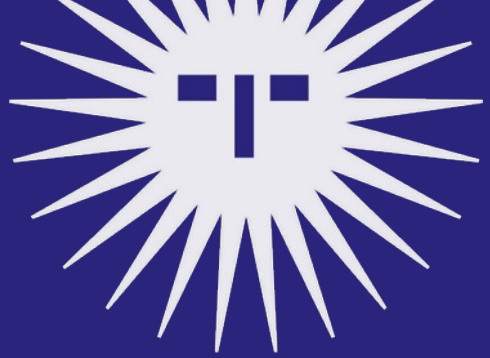
ArthropodCounter.sol (v4)

```sol
contract ArthropodCounter is OwnableUpgradeable {
    /// @custom:storage-location erc7201:Arthropod.Storage
    struct ArthropodStorage{
        uint256 insects;
        uint256 arachnids;
    }
    // keccak256(abi.encode(uint256(keccak256("Arthropod.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant ARTHROPOD_STORAGE_SLOT =
0x9ea65aa6b13a05c6c8cef4cb75bc25aee688eda3792f8ab5c5de20a390adea00;
    function updateInsects(uint256 _insectCount) external onlyOwner {
        ArthropodStorage storage acs = _getArthropodStorage();
        acs.insects = _insectCount;
    }
    function updateArachnids(uint256 _arachnidsCount) external onlyOwner {
        ArthropodStorage storage acs = _getArthropodStorage();
        acs.arachnids = _arachnidsCount;
    }
    function insects() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.insects;
    }
    function arachnids() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.arachnids;
    }
    function arthropods() public view returns (uint256) {
        ArthropodStorage storage acs = _getArthropodStorage();
        return acs.insects + acs.arachnids;
    }
    function _getArthropodStorage() private pure returns (ArthropodStorage storage $) {
        assembly {
            $.slot := ARTHROPOD_STORAGE_SLOT
        }
    }
}
```

GastropodCounter.sol (v4)

```sol
contract GastropodCounter is OwnableUpgradeable {
    /// @custom:storage-location erc7201:Gastropod.Storage
    struct GastropodStorage{
        uint256 snails;
        uint256 slugs;
    }
    // keccak256(abi.encode(uint256(keccak256("Gastropod.Storage")) - 1)) & ~bytes32(uint256(0xff))
    bytes32 private constant GASTROPOD_STORAGE_SLOT =
0xc92446913a76e5f4eee9a73f4f714d93468e0f6a6509e6fe302e36b80fa28d00;
    function updateSnails(uint256 _snailCount) external onlyOwner {
        GastropodStorage storage gcs = _getGastropodStorage();
        gcs.snails = _snailCount;
    }
    function updateSlugs(uint256 _slugCount) external onlyOwner {
        GastropodStorage storage gcs = _getGastropodStorage();
        gcs.slugs = _slugCount;
    }
    function snails() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.snails;
    }
    function slugs() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.slugs;
    }
    function gastropods() public view returns (uint256) {
        GastropodStorage storage gcs = _getGastropodStorage();
        return gcs.snails + gcs.slugs;
    }
    function _getGastropodStorage() private pure returns (GastropodStorage storage $) {
        assembly {
            $.slot := GASTROPOD_STORAGE_SLOT
        }
    }
}
```


“
As the next step we are planning to allow using constants and add special-purpose helpers for the most commonly used layouts (in particular *ERC-7201: Namespaced Storage Layout*)
—SOLIDITY TEAM



What full support for Custom Storage Layouts could look like:

BugCounter.sol (v5)

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
import {ArthropodCounter} from "./ArthropodCounter.sol";
import {GastropodCounter} from "./GastropodCounter.sol";

contract BugCounter is UUPSUpgradeable, ArthropodCounter, GastropodCounter
    layout
        ArthropodCounter at erc7201("Arthropod.Storage"),
        GastropodCounter at erc7201("Gastropod.Storage"),
        BugCounter at erc7201("Bug.Storage") {

    uint256 public worms;
    uint256 public softwareBugs;

    constructor() {
        _disableInitializers();
    }
    function initialize(address _owner) public override (ArthropodCounter, GastropodCounter) initializer {
        __Ownable_init(_owner);
        __UUPSUpgradeable_init();
    }
    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }
    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        softwareBugs = _softwareBugCount;
    }
    function bugs() public view returns (uint256) {
        return arthropods() + gastropods() + worms + softwareBugs;
    }
    function _authorizeUpgrade(address newImplementation) internal override onlyOwner {}
    function erc7201(string memory _namespaceId) private pure returns (bytes32) {
        return keccak256(abi.encode(uint256(keccak256(abi.encode(_namespaceId))) - 1)) &
~bytes32(uint256(0xff));
    }
}
```

BEFORE

BugCounter.sol (v1a)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter, GastropodCounter {

    uint256 public worms;
    uint256 public softwareBugs;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + gastropods() + worms + softwareBugs;
    }
}
```

AFTER

BugCounter.sol (v5)

```
contract BugCounter is UUPSUpgradeable, ArthropodCounter, GastropodCounter
    layout
        ArthropodCounter at erc7201("Arthropod.Storage"),
        GastropodCounter at erc7201("Gastropod.Storage"),
        BugCounter at erc7201("Bug.Storage") {

    uint256 public worms;
    uint256 public softwareBugs;

    function updateWorms(uint256 _wormCount) external onlyOwner {
        worms = _wormCount;
    }

    function updateSoftwareBugs(uint256 _softwareBugCount) external onlyOwner {
        softwareBugs = _softwareBugCount;
    }

    function bugs() public view returns (uint256) {
        return arthropods() + gastropods() + worms + softwareBugs;
    }

    function erc7201(string memory _namespaceId) private pure returns (bytes32) {
        return
            keccak256(abi.encode(
              uint256(keccak256(abi.encode(_namespaceId))) - 1))
            & ~bytes32(uint256(0xff));
    }
}
```

ArthropodCounter.sol (v5)

```sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract ArthropodCounter is OwnableUpgradeable {
        uint256 public insects;
        uint256 public arachnids;

    constructor() {
        _disableInitializers();
    }

    function initialize(address _owner) public virtual initializer {
        __Ownable_init(_owner);
    }

    function updateInsects(uint256 _insectCount) external onlyOwner {
        insects = _insectCount;
    }

    function updateArachnids(uint256 _arachnidsCount) external onlyOwner {
        arachnids = _arachnidsCount;
    }

    function arthropods() public view returns (uint256) {
        return insects + arachnids;
    }
}
```

GastropodCounter.sol (v5)

```sol
// SPDX-License-Identifier: MIT
pragma solidity 0.8.30;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract GastropodCounter is OwnableUpgradeable {
        uint256 public snails;
        uint256 public slugs;

    constructor() {
        _disableInitializers();
    }

    function initialize(address _owner) public virtual initializer {
        __Ownable_init(_owner);
    }

    function updateSnails(uint256 _snailCount) external onlyOwner {
        snails = _snailCount;
    }

    function updateSlugs(uint256 _slugCount) external onlyOwner {
        slugs = _slugCount;
    }

    function gastropods() public view returns (uint256) {
        return snails + slugs;
    }
}
```

Variable	Slot	Value
uint256 insects	0x9e...00	0x07
uint256 arachnids	0x9e...01	0x01
uint256 snails	0xc9...00	0x09
uint256 slugs	0xc9...01	0x05
uint256 worms	0x70...00	0x03
uint256 softwareBugs	0x70...01	0x02

UPGRADEABILITY TIMELINE

The Storage Gap is dying. Let's make sure native Solidity Custom Storage Layouts are the final nail in the coffin. In conjunction with patterns such as Namespaced Storage, let's work together to make the future of smart contract upgrades easy, simple, and safe.

PAST FUTURE

Nothing	Storage Gaps	ERC-7201	Custom Storage Layouts

GRACIAS

NETHERMIND

& Upgrade

Let's **build** together!

www.nethermind.io

ericmartihaynes

ericmartihaynes