OpenZeppelin

Smart contract design

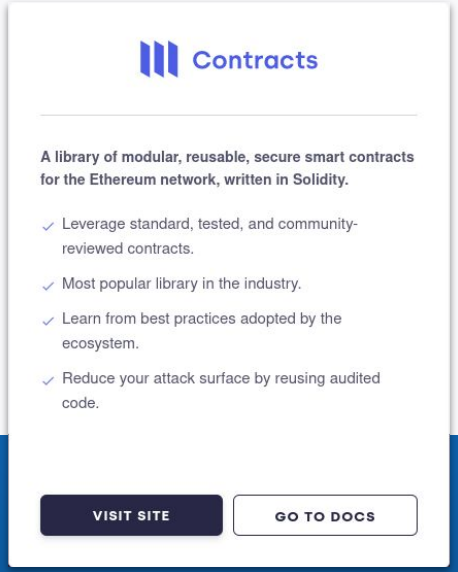
for a stateless EVM

Hadrien Croubois – Stateless summit – April 2026

Contracts

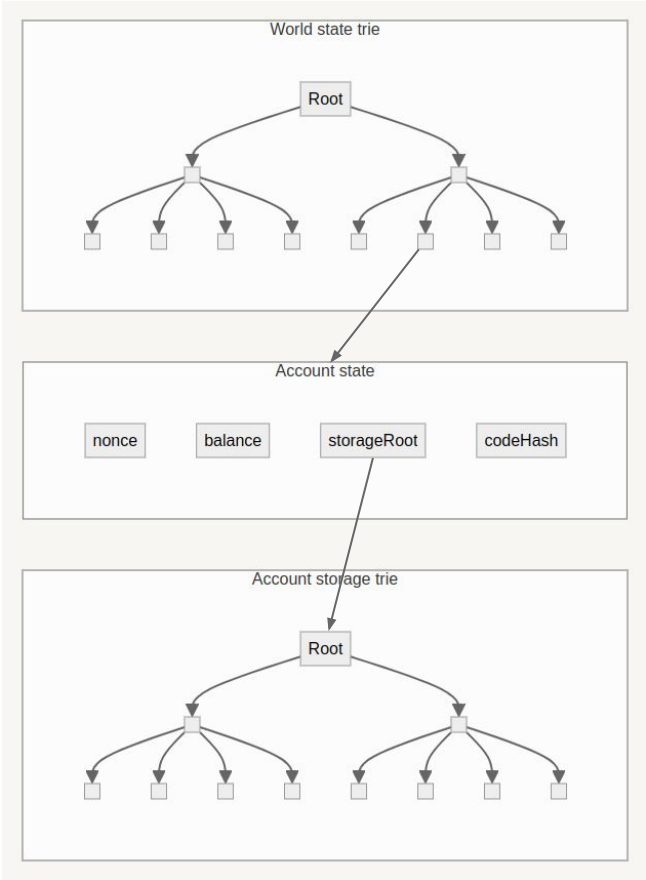
@openzeppelin/contracts@5.6.1

@openzeppelin/contracts-upgradeable@5.6.1



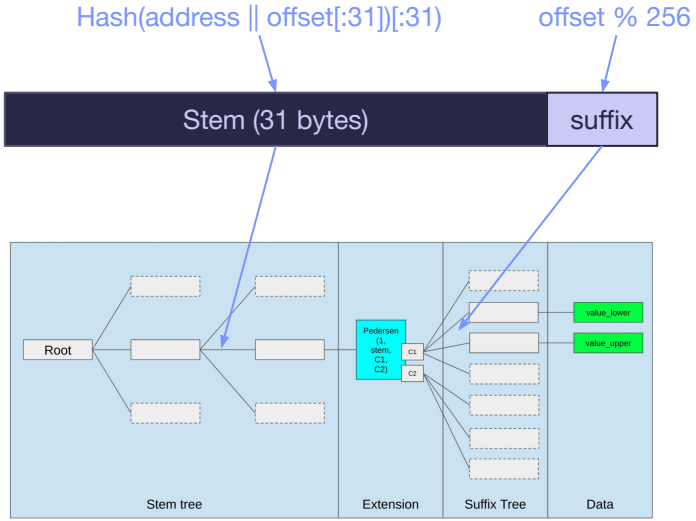
How smart contract state is stored

- **Each account comes with a storage trie**
- **Used for persistent data**
  ERC-20 balances, ERC-721 ownership, everything
- **Storage is divided in slots**
  Each account has 2^{256} slots available
- **Slots must be “warmed up” before use**
  You only pay the warm-up price once per tx
- **All slots are identical**



That all changes with the stateless

- **All storage in the same trie**
- **Position in that shared trie is a combination of account and offset (“slot number”)**
- **Slots are gathered in buckets**
  For a given account, consecutive slots are in the same bucket (up to 256 slots)
- **Buckets are “warmed up”, not slots**
  So reusing slots in a given bucket is efficient

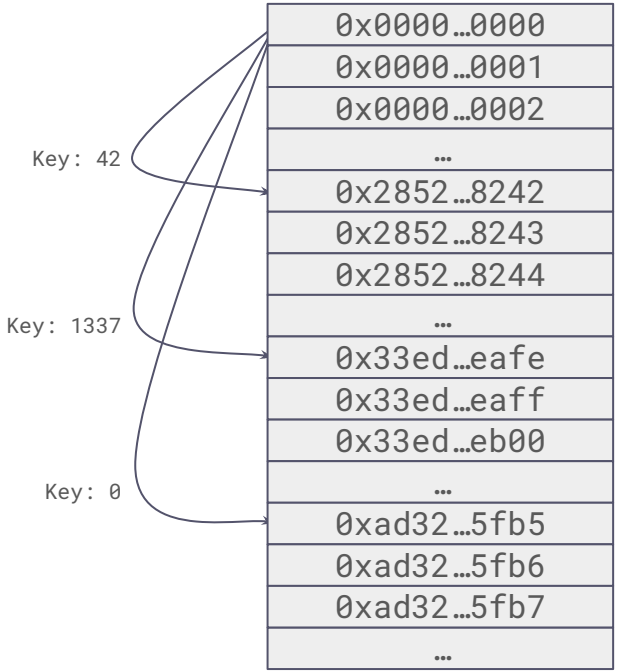


The grammar of storage layouts (Solidity & Vyper)

$$
\begin{aligned}
L_0 := 0 &\quad \leftarrow \text{Contract's root} \\
\mid L_0 + n &\quad \leftarrow \text{State variables, structs, array elements} \\
\mid \texttt{keccak256}(L_0) &\quad \leftarrow \text{Arrays derivation} \\
\mid \texttt{keccak256}(H(k) \oplus L_0) &\quad \leftarrow \text{Mappings (Solidity)} \\
\mid \texttt{keccak256}(L_0 \oplus H(k)) &\quad \leftarrow \text{Mappings (Vyper)}
\end{aligned}
$$

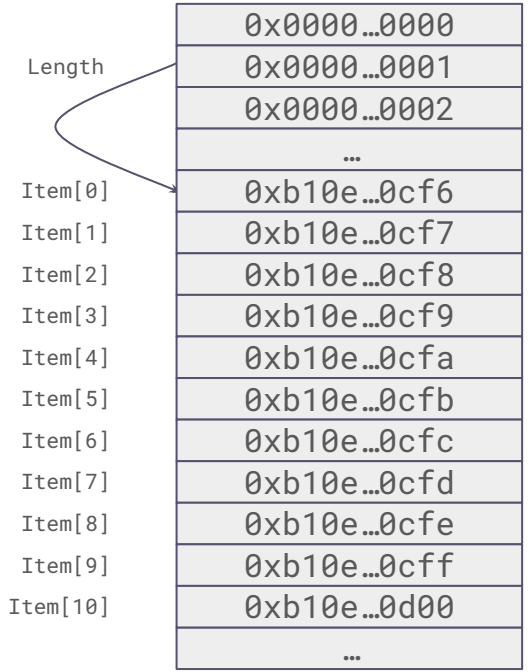
Where the grammar fails: Mappings

- **Mappings are extensively used in solidity**
  Balances, ownership, allowance, …
- **Mappings work by pointing to a pseudo-random location**
  Virtually impossible to get a collision + all slots are equally good
- **Data is scattered everywhere**
  Very poor data locality for verkle, each key point to a different bucket



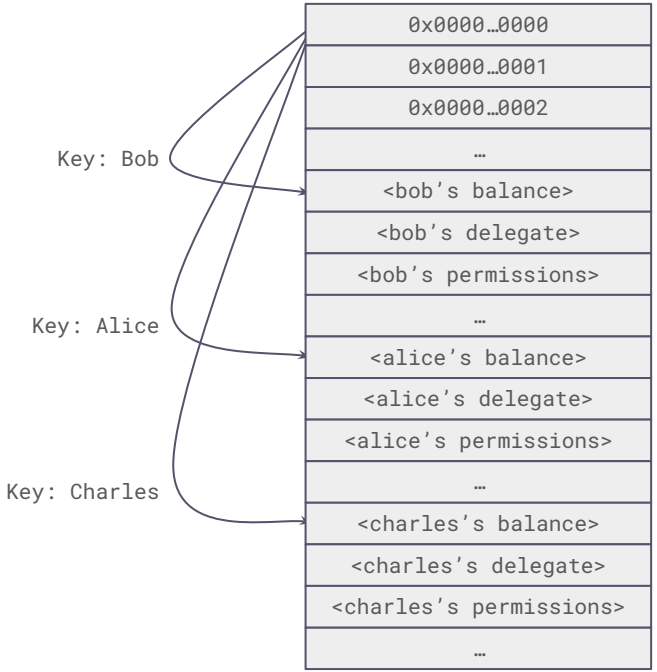
Where the grammar fails: Arrays

- **Arrays have way better data locality then mappings**
- **Each array access check that length is within bounds**
  Length information is located at the root (different bucket)
- **The start of the array is unlikely to coincide with the start of a bucket**
  Sub-optimal for small arrays, tapers off when the array length increases



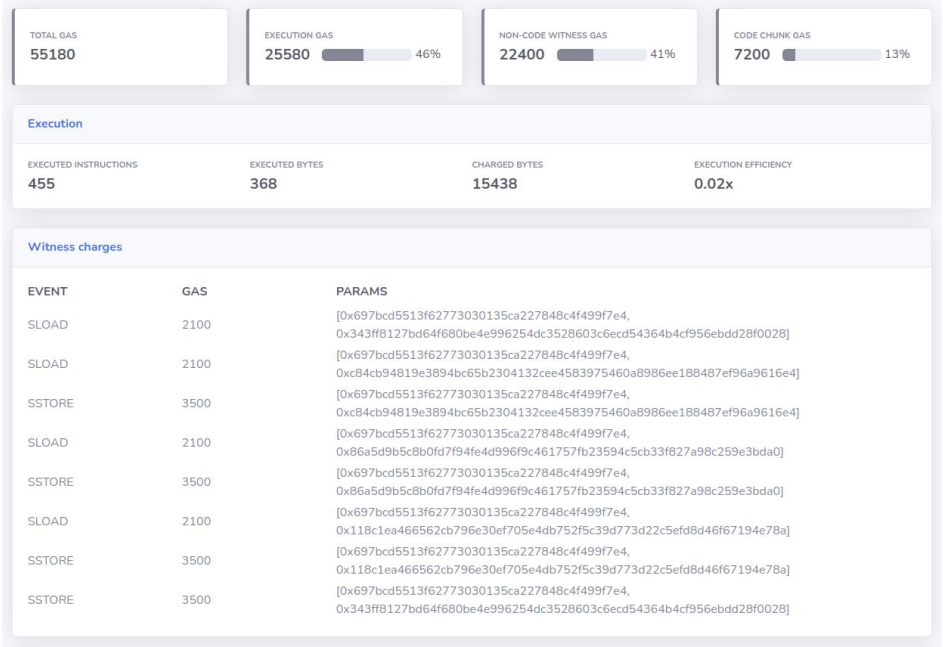
What you can do as a developer: consolidate mappings

- **Merge mappings that use the same key**
  Map to a structure that contains all the values
- **Easy to do, but will occasionally not work**
  If the key points to the end of a bucket, the structure may span over two buckets
- **Limited to some cases**
  In ERC-20, balances and approvals have different key patterns, so they cannot be consolidated

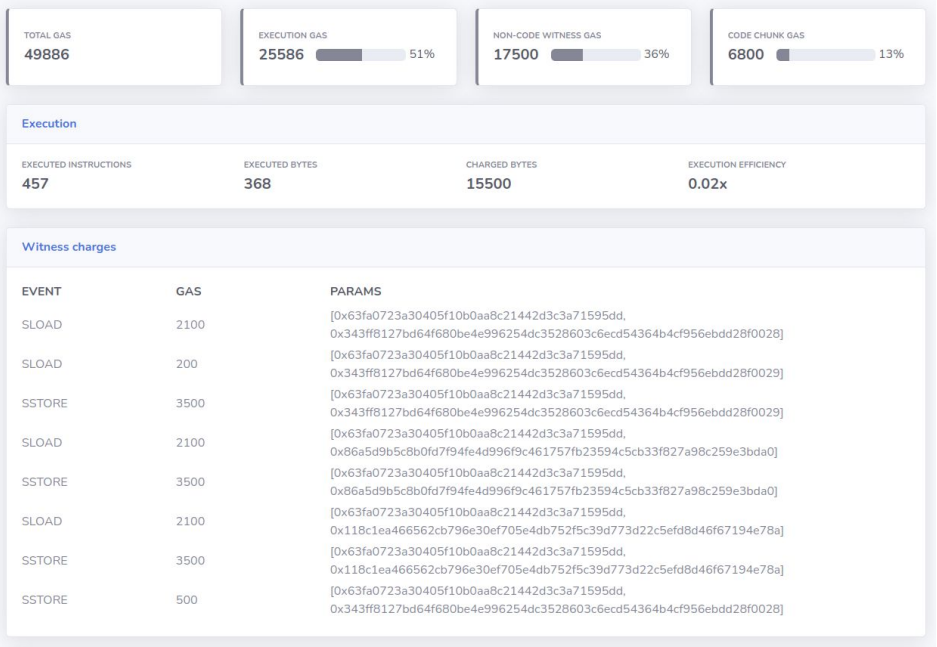


34 contracts/token/ERC721/ERC721.sol

```
@@ -25,13 +25,19 @@ abstract contract ERC721 is Context, ERC165, IERC721, IERC721Metadata, IERC721Er
25  25      // Token symbol
26  26      string private _symbol;
27  27
28      -   mapping(uint256 tokenId => address) private _owners;
    28  +   struct TokenDetails {
    29  +       address owner;
    30  +       address approval;
    31  +   }
29  32
30      -   mapping(address owner => uint256) private _balances;
    33  +   struct AccountDetails {
    34  +       uint256 balance;
    35  +       mapping(address => bool) operators;
    36  +   }
31  37
32      -   mapping(uint256 tokenId => address) private _tokenApprovals;
    38  +   mapping(uint256 tokenId => TokenDetails) private _tokens;
33  39
34      -   mapping(address owner => mapping(address operator => bool)) private _operatorApprovals;
    40  +   mapping(address owner => AccountDetails) private _accounts;
35  41
36  42      /**
37  43       * @dev Initializes the contract by setting a `name` and a `symbol` to the token collection.
@@ -58,7 +64,7 @@ abstract contract ERC721 is Context, ERC165, IERC721, IERC721Metadata, IERC721Er
58  64          if (owner == address(0)) {
59  65              revert ERC721InvalidOwner(address(0));
60  66          }
61      -       return _balances[owner];
    67  +       return _accounts[owner].balance;
62  68      }
63  69
64  70      /**
@@ -128,7 +134,7 @@ abstract contract ERC721 is Context, ERC165, IERC721, IERC721Metadata, IERC721Er
128 134       * @dev See {IERC721-isApprovedForAll}.
129 135       */
130 136      function isApprovedForAll(address owner, address operator) public view virtual returns (bool) {
131     -       return _operatorApprovals[owner][operator];
    137 +       return _accounts[owner].operators[operator];
132 138      }
```



[transferFrom on a “Normal ERC-721”]



[transferFrom on a “Packed ERC-721”]

What you can do as a smart contract developer: use (custom) arrays

Mapping

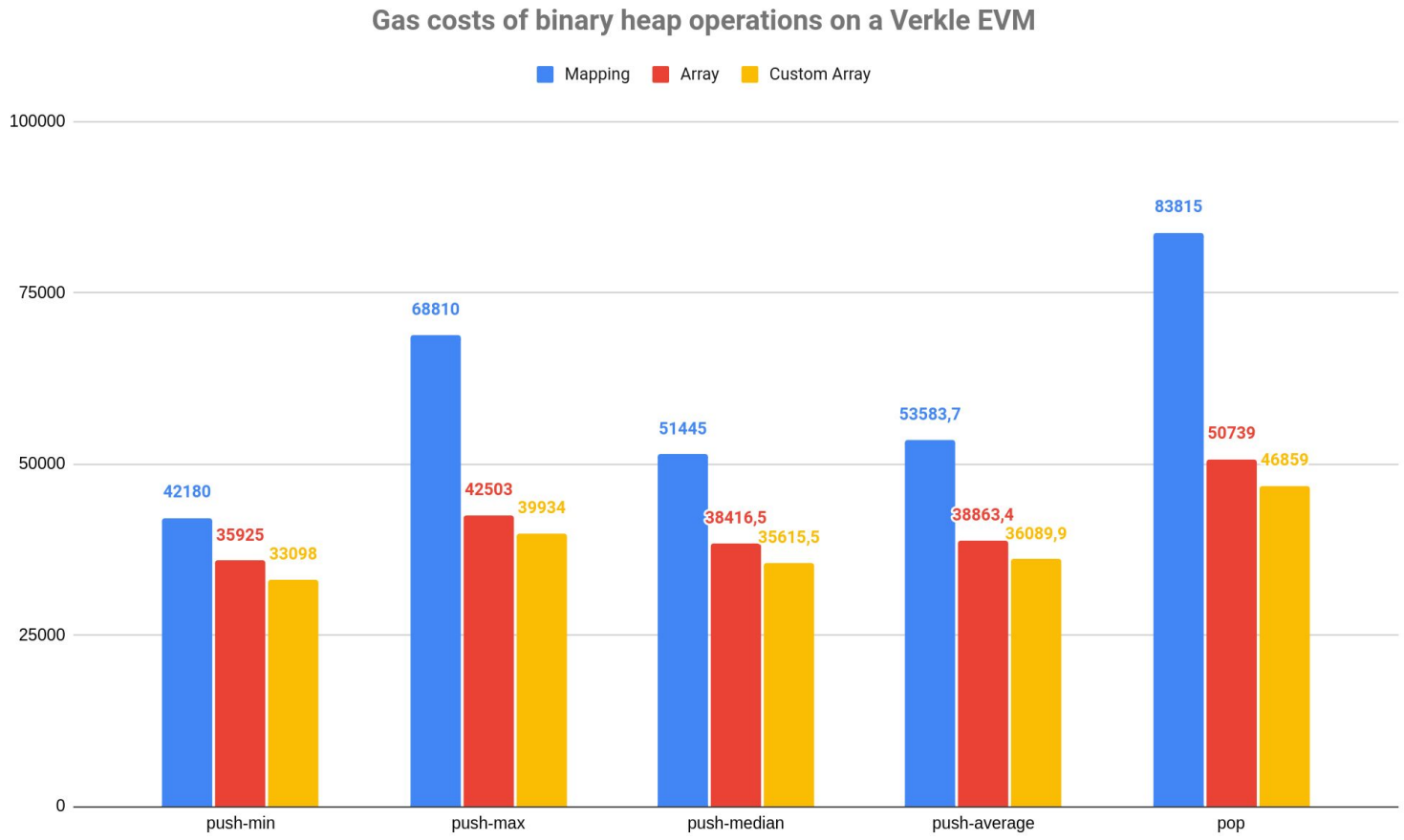
```
library HeapMapping {
    using SafeCast for *;

    struct Uint256Heap {
        mapping(uint32 => uint32) tree;
        mapping(uint32 => Node) items;
        uint32 size;
        uint32 nextItemIdx;
    }

    struct Node {
        uint256 value;
        uint32 heapIndex; // value -> position
    }
```

Array

```
library HeapArray {
    using SafeCast for *;

    struct Uint256Heap {
        Uint256HeapNode[] data;
    }

    struct Uint256HeapNode {
        uint256 value;
        uint32 index; // position -> value
        uint32 lookup; // value -> position
    }

    function _unsafeNodeAccess(
        Uint256Heap storage self,
        uint32 pos
    ) private pure returns (Uint256HeapNode storage result) {
        assembly ("memory-safe") {
            mstore(0x00, self.slot)
            result.slot := add(keccak256(0x00, 0x20), mul(pos, 2))
        }
    }
```

Custom array implementation

```
library HeapArray2 {
    using SafeCast for *;

    struct Uint256Heap {
        bytes32 _placeholder_do_not_use;
    }

    struct Uint256HeapNode {
        uint256 value;
        uint32 index; // position -> value
        uint32 lookup; // value -> position
    }

    struct Uint256HeapLength {
        uint256 value;
    }

    function _unsafeNodeAccess(
        Uint256Heap storage self,
        uint32 pos
    ) private pure returns (Uint256HeapNode storage result) {
        assembly ("memory-safe") {
            mstore(0x00, self.slot)
            result.slot := add(keccak256(0x00, 0x20), add(mul(pos, 2), 1))
        }
    }

    function _unsafeLengthAccess(
        Uint256Heap storage self
    ) private pure returns (Uint256HeapLength storage result) {
        assembly ("memory-safe") {
            mstore(0x00, self.slot)
            result.slot := keccak256(0x00, 0x20)
        }
    }
```



What we ultimately need: compiler support!

Compiler must evolve to optimize for stateless storage costs …

- **Replace “keccak256(...)” with “keccak256(...) & ~0xFF”**
  for Array and mapping derivation
- **Move the length of the array to the same “space” as the elements**
- **Provide in-language mechanism to “extend” a mapping from a parent contract**

… but this is breaking

```
abstract contract ERC721 is Context, ERC165, IERC721, IERC721Metadata, IERC721Errors {
    using Strings for uint256;

    // Token name
    string private _name;

    // Token symbol
    string private _symbol;

    mapping(uint256 tokenId => address) private _owners;

    mapping(address owner => uint256) private _balances;

    mapping(uint256 tokenId => address) private _tokenApprovals;

    mapping(address owner => mapping(address operator => bool)) private _operatorApprovals;


abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
    mapping(address owner => mapping(uint256 index => uint256)) private _ownedTokens;
    mapping(uint256 tokenId => uint256) private _ownedTokensIndex;

    uint256[] private _allTokens;
    mapping(uint256 tokenId => uint256) private _allTokensIndex;


abstract contract ERC721URIStorage is IERC4906, ERC721 {
    using Strings for uint256;

    // Interface ID as defined in ERC-4906. This does not correspond to a traditional interfac
    // defines events and does not include any external function.
    bytes4 private constant ERC4906_INTERFACE_ID = bytes4(0x49064906);

    // Optional mapping for token URIs
    mapping(uint256 tokenId => string) private _tokenURIs;
```

@openzeppelin/contracts

docs.openzeppelin.com

forum.openzeppelin.com