A possible roadmap to user-owned storage (and renting) led by contract development.

(Please let me know if I should go ahead and write detailed design and EIP for this.)

Goals:

- Data storage is owned by the beneficiary rather than application contracts.
- Owners cannot (U)pdate their data but can (D)elete them to manage the state size (and rent) and remove the spam data created by spammers.
- Application contracts have full CRUD control, while users can (D)elete their data without impacting the application's shared states or other users' states.
- Optionally, data resurrection can be developed by storage contracts, but there are many ways this roadmap can end up without resurrection at all.
- Owners cannot (U)pdate their data but can (D)elete them to manage the state size (and rent) and remove the spam data created by spammers.
- Application contracts have full CRUD control, while users can (D)elete their data without impacting the application's shared states or other users' states.
- Optionally, data resurrection can be developed by storage contracts, but there are many ways this roadmap can end up without resurrection at all.
- StorageManager

strategies are led by smart contract development and are open to the community.

- Different storage strategies can be developed and utilized by the applications simultaneously.
- Applications can choose and utilize the desired StorageManager

without worrying about the storage implementation.

- Each StorageManager

stores all of its application data in its trie and grants proper permissions to each user and application contract.

- Different storage strategies can be developed and utilized by the applications simultaneously.
- Applications can choose and utilize the desired StorageManager

without worrying about the storage implementation.

- Each StorageManager

stores all of its application data in its trie and grants proper permissions to each user and application contract.

- Execution clients can apply different storage optimization strategies without any protocol upgrade.
- Protocol upgrades can be planned in the endgame or not at all depends on the development of the previous steps. (E.g. rent and prune)

Notes

- Execution clients can optimize storage access by storing all data belonging to an account in a single trie, even across multiple StorageManager

contracts.

- Execution clients can incentivize applications to use StorageManager

by prioritizing transactions with StorageManager

in the EIP-2930 access list.

- Complex storage data rent, prune and resurrection strategies can be developed by the community.
- Data resurrection can be entirely omitted in favor of more socioeconomical mechanisms. For example, a close-to-be-pruned account can be preserved by anyone through payment, and later the preserver can benefit from various forms

of compensation provided by the account owner.

Example

The simplest StorageManager

implementation that is sufficient for execution clients' optimization and state pruning.

contract SimpleStorageManager { event Update(address indexed user, address indexed application, address indexed key, value);

mapping(bytes32 => bytes32) storage s_data;

// by user function delete(address application, bytes32 key) public { delete s_data[_key(msg.sender, application, key)]; emit Update(msg.sender, application, key, 0); }

// by application function update(address user, bytes32 key, bytes32 value) publlic { s_data[_key(user, msg.sender, key)] = value; emit Update(user, msg.sender, key, value); }

function read(address user, address application, bytes32 key) public view returns (bytes32) { return s_data[_key(user, application, key)]; }

function _key(address user, address application, bytes32 key) internal pure returns (bytes32) { return keccak256(abi.encodePacked(user, application, key)); } }