Indigo V2

Synthetic Assets on Cardano

Indigo Laboratories, Inc.
info@indigo-labs.io

March 2024, v2.0.0 (Draft)

Table of Contents

1 Motivation

For most of the world's population, important financial tools are inaccessible. This is exemplified by the fact that two people can share the same education, perform the same work, and put in the same amount of effort, yet not have the same development possibilities. One could have access to a share of the global economy's growth, while the other may be left out.

It's brutal and unfair. Until now, borders have limited human development. With the advent of blockchain technology, we are amid a global switch in the financial foundation we use as a society to trade and transact. At the forefront of this transformation, we present a new solution to equalize the playing field by bringing the world's assets to the blockchain. This solution allows anyone to access and participate in new financial markets and take control of their own financial destiny, paving the way for a new mantra: Tokenize Everything.

2 Introduction

This document (the "Indigo paper") presents the Indigo Protocol (the "protocol" or "Indigo"), a synthetic assets protocol built for Cardano[1]. Combining the benefits of a white paper[2] and a yellow paper[3], the Indigo paper provides both a high-level and detailed protocol specification for educating the Indigo community. The Indigo paper serves as the basis to introduce Indigo and kickstart complete community management.

Indigo has been committed to a Fair Launch to bootstrap the protocol from the ground up. As part of this initiative, no minting, pre-sale, or distribution of tokens related to the protocol have been undertaken. This ensures that starting from launch, Indigo will be community managed.

2.1 Synthetic Assets

Indigo creates synthetic assets which are known in the protocol as iAssets (i.e., "Indigo Assets"). iAssets are cryptocurrency assets that derive their prices from tracked assets. Prices of iAssets are influenced via protocol rules with the intention of matching the prices of the tracked assets. One example of an iAsset is iBTC,

[1]Cardano is a public blockchain that supports smart contracts and custom tokens utilizing an eUTXO architecture, an extension of UTXO.
[2]A white paper is a marketing tool typically used to attract investors.
[3]A yellow paper typically contains complete specification details.

representing a synthetic version of Bitcoin (BTC); it is designed to mimic the price action of BTC – an asset that lives in separate ecosystem than Indigo.

2.2 Indigo Protocol

Indigo is a decentralized synthetics protocol for on-chain exposure to assets with publicly verifiable prices. Using Cardano Plutus[4] smart contracts, the protocol enables the creation of iAssets. Prices of iAssets are soft pegged[5] to external tracked assets; iAssets are overcollateralized in the form of a decentralized Collateralized Debt Position ("CDP"). The protocol enforces liquidations to ensure iAssets always maintain overcollateralization, meaning the value of the collateral in the CDP exceeds the intended value of the iAsset. In the event of a CDP becoming undercollateralized, a liquidation reestablishes overcollateralization by confiscating the collateral of the undercollateralized CDP and replacing it with another user's overcollateralized CDP.

While minting an iAsset requires opening a CDP, after iAssets are minted they are freely exchangeable. Anyone with a Cardano wallet can send or receive iAssets, regardless of whether they have an open CDP.

2.3 Benefits of iAssets

Users can gain some benefits of owning an asset without being required to obtain or own the asset themselves. This can be useful in cases where assets are difficult for a user to obtain or for assets that live elsewhere yet users desire to utilize on the Cardano blockchain.

iAssets can be used as building pieces to be included in a wider financial strategy. This could include being part of derivative contracts or constructing a widely diversified portfolio in one easy to use system. Users can make trades without requiring the underlying supply. For example, more iAsset could exist than total supply of the real asset, allowing for leveraged trades that wouldn't be possible to be settled using the real underlying assets.

iAssets have the following properties:

- Tracking different type of assets and statistics; which allows the creation of many new asset classes for emerging industries.
- No custodians; iAsset creation is fully decentralized.
- Low barrier to entry; anyone with cryptocurrency can use Indigo to mint new synthetic assets or buy and trade them on the open market.
- Composability; iAssets can be used as a lego block, enabling their integration into a larger financial ecosystem.

Table 1: Examples of possible iAssets

Name	**Description**
iBTC	Tracks the price of BTC on the Bitcoin blockchain
iETH	Tracks the price of ETH on the Ethereum blockchain
iUSD	Tracks the price of dollar-denominated stablecoins on any blockchain
iCPI	Tracks the change of the Consumer Price Index over time

Generally, an iAsset names begin with the letter "i," followed by the name of the tracked asset.

2.3.1 Obtaining iAssets

There are two ways to obtain iAssets:

- **Buying iAssets** – Users directly purchase iAssets via an exchange (centralized or decentralized), thus gaining exposure without having to request any loan.
- **Minting iAssets** – Users make overcollateralized loans against their cryptocurrency assets.

[4] Plutus is a smart contract platform for Cardano.
[5] A soft peg is a strategy of maintaining the value of an asset against another asset by utilizing an exchange rate mechanism.
[5] The Consumer Price Index is a measure of the average change over time in the prices of goods and services.

Users can buy iAssets from any exchange that has available supply. After buying an iAsset, the user gains full control of the iAsset and can reap benefits from price possible appreciation. Users can be assured that iAssets maintain their intended pegged prices due to Indigo's liquidation process.

The second way for users to obtain iAssets is by minting them within the Indigo Web App by depositing collateral and creating a loan.

2.4 Collateralized Debt Positions

Every iAsset is backed by collateral held in a Collateralized Debt Position (a "CDP"). A CDP is created by a user locking collateral (a minimum of 10 ADA) into Indigo to mint a new iAsset. An iAsset is borrowed against the **collateral**, creating a **debt**, and this **position** is watched by liquidators to ensure overcollateralization.

The value of the collateral in a CDP should always exceed a governance-based Liquidation Ratio (a "LR"). Each iAsset type has its own LR. Both the value of the collateral and iAsset price can fluctuate over time, potentially causing a CDP to become undercollateralized. A CDP is considered undercollateralized when its collateral ratio (the "CR") falls below the iAsset's LR. The CR is the ratio of the collateral value relative to the minted iAsset value, and can be calculated using the formula:

$$c = \frac{a-i}{md}$$

or:

$$c = \frac{(a-i)b}{mp}$$

Where:

- c is the CR used to determine solvency
- a is the amount of ADA locked in the CDP
- i is the amount of ADA accrued interest on the CDP
- b is the dollar-denominated price of ADA
- p is the dollar-denominated price of the iAsset's tracked asset
- d is the ADA-denominated price of the iAsset's tracked asset
- m is the amount of iAsset minted from the CDP

When CR drops below LR, the CDP is considered insolvent and eligible for being frozen, which can then lead to liquidation to ensure the reestablishment of solvency.

A CDP opening is also subjected to Maintenance Ratio (the "MR"). The position will not be able to open a position with a collateral ratio below this maintenance ratio. In a period where the iAssets are above the intended peg reference rate. This ratio can also be lowered to LR which will allow arbitrage and maintain the hardpeg.

2.4.1 CDP and iAsset Example

As an example, assume Violet wants to mint 100 iDOT (m). DOT is trading for \$15 (p). Violet has 2,000 ADA (a) she's willing to use as collateral to borrow iDOT. ADA is trading for \$1.28 (b).

Violet deposits 2,000 ADA into Indigo to mint 100 iDOT. A CDP is created consisting of 2,000 ADA. Violet now owns 100 iDOT and owes 100 iDOT to Indigo. Violet can still earn staking rewards from her 2,000 ADA, but cannot transfer it because it now is used as collateral. To regain control of her ADA, Violet must return 100 iDOT.

Violet's CR is ˜171%:

$$c = \frac{ab}{mp} \therefore \frac{2000 \times 1.28}{100 \times 15} = \sim 1.71$$

As the price of either DOT or ADA changes, CR changes too. When CR drops below the iDOT's LR, Violet's CDP is subject to liquidation.

If the price of ADA increases to \$1.40 (b) and DOT increases to \$19 (p), then Violet's CR drops to ˜147%:

$$\frac{2000 \times 1.4}{100 \times 19} = \sim 1.47$$

If the iDOT LR was 150%, Violet's CDP could be liquidated. Upon liquidation, Violet would lose her 2,000 ADA collateral deposit. Violet could still have her 100 iDOT, worth $1,900 ($19 x 100). The 2,000 ADA she lost would be worth $2,800 ($1.40 x 2000). Therefore, Violet could have lost $900 of value ($2,800 - $1,900).

To have prevented liquidation, Violet needed to either add more ADA into her CDP to increase its CR, or close the CDP by returning the 100 iDOT she borrowed.

2.4.2 CDP Actions and States

Several actions can be taken against a CDP by users of the protocol:

- **Open Position** – Creates a CDP by a user depositing a minimum of 10 ADA as collateral, and associates it with an iAsset type that can be minted. The user who creates the CDP becomes the CDP's owner.
- **Deposit Collateral** – An owner can increase CR by depositing more collateral.
- **Withdraw Collateral** – An owner can lower CR by withdrawing part or all the collateral. Collateral cannot be withdrawn if it brings CR below the iAsset's MR. If a CDP has no debt (i.e., no minted iAsset) and all collateral is withdrawn, then the CDP is closed.
- **Borrow iAsset** – An owner can lower CR by minting an iAsset. This increases the amount of debt against the CDP. More iAsset cannot be minted if it brings CR below the iAsset's MR.
- **Repay Debt** – An owner can increase the CR by repaying debt in the form of iAsset. When the debt is repaid, the iAsset is burned (i.e., destroyed). More iAsset cannot be burned than debt owed by the CDP.
- **Freeze** – If CR is below the iAsset's LR, any user can submit a transaction for that CDP to be frozen. Upon freezing, a CDP is no longer usable or interactable by its former owner. The former owner loses all access and rights to the CDP.
- **Liquidate** – If a CDP is frozen, any user can submit a transaction for that CDP to be liquidated. Upon liquidation, CDP debt is repaid by withdrawing iAsset from a Stability Pool. As debt is repaid, collateral is withdrawn from the CDP. If all debt is repaid, then all collateral is withdrawn, and the CDP is closed.
- **Merge** – If two or more CDPs are frozen, any user can submit a transaction for them to be merged into one CDP. Upon merging, all but one of the CDPs requested to be merged are closed, and their debt and collateral are transferred into a single CDP.
- **Redeem** – If CR is below the iAssets redemption margin ratio, any user can redeem the debt of that CDP for some of it's collateral at the current oracle price.

A CDP has the following states:

- **Open** – A CDP that is fully collateralized, with its CR value above the iAsset's LR. Open CDPs remain fully usable by their owners.
- **Insolvent** – A CDP that is undercollateralized, with its CR value below the iAsset's LR. Insolvent CDPs remain fully usable by their owners but eligible to be frozen by any user.
- **Frozen** – A CDP that has been confiscated by the protocol and no longer has an owner. A CDP becomes frozen after a user successfully submits a request against an insolvent CDP. Frozen CDPs cannot be used by their former owners.
- **Closed** – A CDP whose CR value is zero, no longer having any collateral or debt. A CDP is closed after all its debt is repaid and its collateral is withdrawn.

2.4.3 CDP Liquid Staking

Indigo supports liquid staking of ADA collateral within CDPs, allowing users to continue earning ADA rewards from the Cardano network on top of utilizing the benefits of iAsset minting. This improves capital efficiency and doubles reward capabilities – rewards earned from Cardano, and rewards earned from Indigo. Liquid staking is a unique capability offered by Indigo and will help attract liquidity from outside of the Cardano ecosystem to encourage more participation, bringing iAssets to a wider audience.

To use liquid staking, users must first have their Cardano wallet staked to their preferred stake pool[6]. The Indigo Web App automatically attaches the user's staking key when creating a CDP. All ADA deposited into that CDP will continue to earn staking rewards from the user's chosen stake pool, accruing in the user's wallet.

[6]Indigo supports any Cardano stake pool. A stake pool is a Cardano network node that forms the basis for consensus on the blockchain. Users can delegate their ADA to stake pools to earn ADA rewards from the Cardano network.

If the user delegates their wallet to a new stake pool after creating the CDP, the CDP will automatically earn rewards from the new stake pool.

2.4.4 CDP Redemption

With the transition to V2, Indigo Protocol will be introduce a variant of redemption. This allows for a direct path to arbitrage the peg.

iAssets have an adjustable parameter Redemption Margin Ratio. This is a limit based parameter which will allow more flexibility for both users and protocol to maintain their needs and requirements and can serve as micro adjustments to the peg together with other pegging mechanisms.

Redemptions work by allowing a user to bring their owned iAssets and redeeming those iAssets for the collateral of redeemable CDPs. CDPs are redeemable when their collateral ratio is below the Redemption Margin Ratio. When doing a redemption, the redeemer is burning an iAsset, essentially paying off the debt of a CDP by withdrawing an equal value (based on oracle price) of collateral from that CDP. Redeemers are only able to redeem enough to bring the redeemed CDP's collateral ratio at or below the RMR.

To calculate the amount of iAsset redeemable, you can use this formula:

$$x = \frac{(\frac{-c}{p+r*m})}{r-1}$$

Where:

- x is the amount iAsset redeemable for this CDP
- c is the amount of collateral attached to the redeemed CDP
- m is the amount of debt attached to the redeemed CDP
- p is the oracle price (in lovelaces) of the iAsset
- r is the redemption margin ratio (in percentage) set for the iAsset

Redemption CDP Reimbursement Fee At the time of redemption, a small fee is taken from the redeemed collateral and returned back to the CDP as an incentive for CDP owners to allow their CDPs to be redeemed. This value is adjustable by the DAO by adjusting the iAsset parameters.

Redemption INDY Staker Fee Similar to the Redemption CDP Reimbursement Fee, when a CDP has been redeemed, a percentage of the the redeemed collateral is sent the the Collector contract to be distributed to INDY stakers as a protocol fee sharing mechanism.

2.4.5 CDP Interest

While redemption offers a hard peg on the floor of iAsset prices, it isn't unlimited in nature as well as resulting in changes in users CDP position which can reduce the use case of a CDP.

Further incentives for peg to stay above this hard peg will also help to prevent excessive redemption.

Staking of Cardano provides a risk-free rate to depositors. Allowing users leverage for free without concern for the risk free rate can result in too many people trying to leverage the system traditionally.

With an interest mechanism, this interest rate seek to influence the peg via interest rate.

With interest charged on loans, this allows for positions to gravitate towards their liquidation point and prevents excessive leverage.

2.5 INDY

The Indigo DAO Token ("INDY") is a Cardano native asset that can be owned, held, or transferred by any user. INDY serves as Indigo's utility token, with one of its key purposes being to allow on-chain voting on DAO proposals (a "proposal")[7]. The total supply of INDY is 35M with a 6 decimal precision. INDY's monetary policy disallows future minting and burning, therefore making the total supply constant and unchanging. Indigo is undergoing a Fair Launch, therefore there has been no pre-sale nor private distribution to investors prior to launch.

INDY will be distributed every Cardano epoch (five days), over a period of five years. There will be three distribution schedules for the community:

- **Governance Distribution** – Users who opt to stake their INDY into Indigo and participate in DAO Governance by voting on proposals will be eligible for INDY rewards proportionally to their pro-rata share of staked INDY.
- **Stability Pool Distribution** – Users who stake their iAssets in Stability Pools to ensure the protocol's solvency will be eligible for INDY rewards proportionally to their pro-rata share of staked iAssets.
- **Liquidity Distribution** – Users who provide liquidity in DEXes will be eligible for INDY rewards proportionally to their pro-rata share of staked LP tokens.

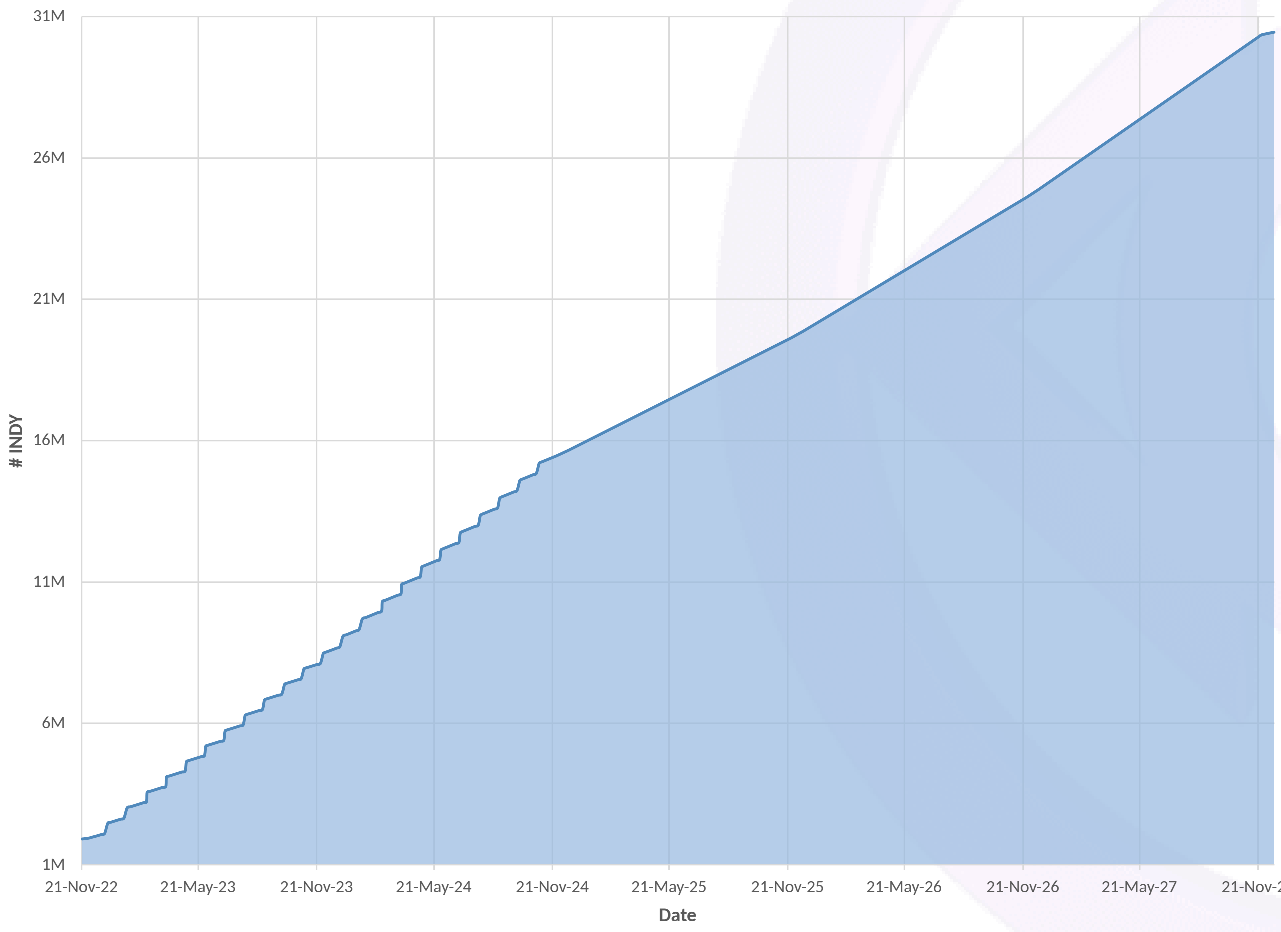


Figure 1: Distribution of INDY over five years

[7]The usage of INDY and the process of voting is described further in the Indigo DAO Constitution and Voting Procedures.

2.5.1 Fair Launch

Indigo has approached its tokenomics and launch from a new perspective, with a focus on gaining community trust first, allowing the protocol to be built with a vision of fairness. Early supporters of Indigo will be among the first receivers of INDY for use within the protocol. INDY will be distributed predominantly to users of the project, rather than investors or special insiders.

After being in development for almost two years without investor funding, the initial Core Contributors of Indigo who have built the codebase – and will continue to improve, optimize, and develop new features – will receive tokens vested over two years beginning the day of mainnet launch.

Indigo has not minted, sold, allocated, distributed, or promised any tokens to third parties. The purpose of INDY is to be used within the protocol; until the launch of mainnet, there is or has been no use for INDY to be distributed or sold. Indigo's Fair Launch has helped alleviate community concerns over rug-pulling or the team not delivering a useful and highly functional product. No purchasing of tokens will be possible until the community has an opportunity to see and use Indigo for themselves.

Indigo's Fair Launch is a novel approach to bootstrapping liquidity, allowing the Indigo community to become highly collaborative, driven, and vibrant. This is evidenced by the Indigo DAO Kickstart – an effort to decentralize the launch of Indigo – which has received wide praise. This approach bolsters Indigo's core tenet of decentralization, making the launch itself a decentralized decision involving possibly thousands of individuals from around the world. Indigo will be governed by the community immediately upon launch. There will be no barriers for use. Anyone, regardless of traits, will be able to gain benefit from Indigo's iAssets. Indigo has established a new framework to allow for community-led projects to come to life, which will be used to generate INDY in as fair of a manner as possible.

2.5.2 Token Generation Event

Indigo's Token Generation Event (the "TGE") will occur upon the beginning of, and at no point prior to, deployment of the Indigo Protocol to mainnet (which is currently anticipated to be November 20th, 2022). Upon minting of INDY, the Initial Token Distribution (the "ITD") will be as follows:

- 350,000 INDY to two or three DEXs approved by the Indigo community
- 350,000 INDY to participants within the Indigo community
- 21,000,000 INDY to one or more wallets (administrated by Indigo Laboratories, Inc. at the direction of the Indigo Foundation on behalf of the Indigo DAO) to be used for the sole purpose of community rewards distributions (Stability Pools, Liquidity, and Governance)
- 4,550,000 INDY to the DAO Treasury Reserve
- 8,750,000 INDY will be allocated to Indigo Laboratories, Inc. for future building, administering, and further developing the protocol, with 7,875,000 being distributed to team members under a two-year monthly vesting schedule

At launch, the circulating supply of INDY[8] will be 1,903,125; 350,000 of which being allocated to Cardano DEXs via an Initial Liquidity Event.

2.5.3 Initial Liquidity Event

Indigo's Initial Liquidity Event (the "ILE") will distribute and make INDY publicly available. The ILE will consist of three phases in conjunction with the launch of Indigo:

1. Airdrop
2. Liquidity Bootstrapping Event (the "LBE")
3. Liquidity Pool Creation

[8] A full detailed spreadsheet of the distribution of INDY with specific dates and allocations can be found in the open source indy-tokenomics project.

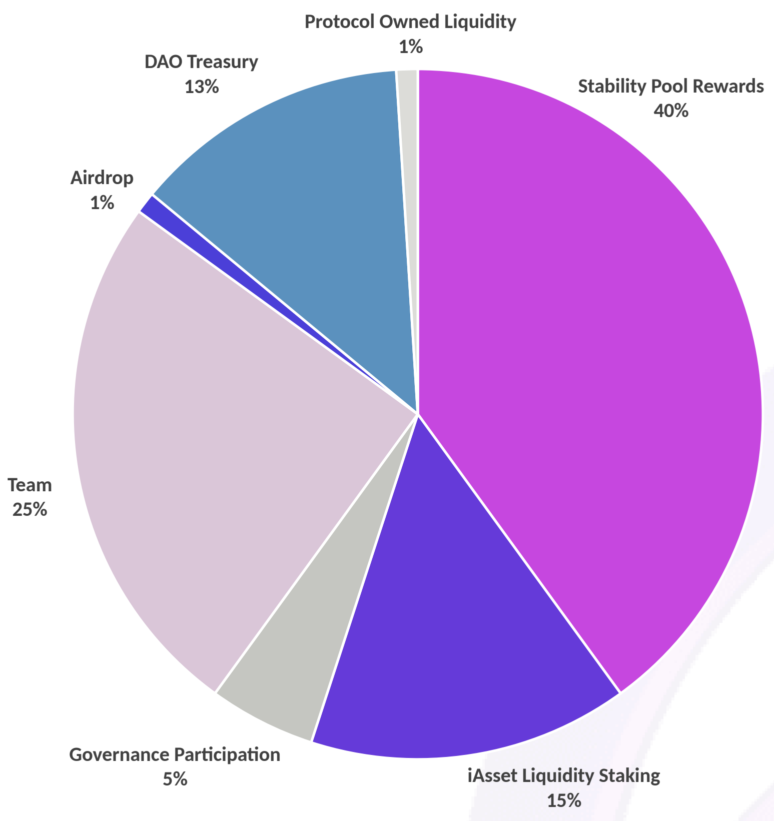


Figure 2: Allocation of INDY

Indigo Airdrop Indigo's airdrop will distribute 350,000 INDY to participants within the Indigo community. The airdrop will consist of two phases:

1. Distribution to early participants of the Indigo community
2. Distribution to stakers supporting the decentralization of Cardano and Indigo

Each phase will be distributed 175,000 INDY. Cardano wallet addresses have been collected by Indigo Laboratories, Inc. (the "Labs") and will be forwarded to Vending Machine.[9] The Labs will send 350,000 INDY to Vending Machine, who will subsequently distribute INDY to qualified recipients via the Indigo Web App.

To redeem airdropped INDY, qualified users will need to connect their wallet to the Indigo Web App and follow the in-app instructions to withdraw INDY into their wallets. Users will be able to determine whether they qualify for the airdrop upon connecting their wallets and navigating to the appropriate reward page. Users will have until March 31st 2023 to withdraw their INDY rewards into their wallets. Any INDY not withdrawn by this time will not be eligible to be withdrawn by users and instead will be subject to redistribution by the Labs.

Members or affiliates of the Labs or Indigo Foundation make no promises on the distribution of tokens. No action or series of actions guarantees a user to receive INDY.

Airdrop 1: Distribution to Early Participants Qualified participants for Airdrop 1 fit into either one of two categories:

1. Participants who showed their interest by successfully completing each of the processes, which were:
   (a) Participate in Indigo's first temperature check in the Indigo Forum
   (b) Connect their Indigo Forum account with their Discord account
   (c) Complete the Indigo Quiz to become an Indigo Guru
2. Participants who aided the Indigo community, as identified by the Labs' team

172,751.924982 INDY is to be distributed to wallets that fit into the first category, and 2,248.07304 INDY is to be distributed to wallets that fit into the second category. A total of 3,458 wallets qualified for the first category, and 30 wallets qualified for the second category.

Addresses deemed to be suspicious or fraudulent were removed from the first category.

[9] Vending Machine is a Cardano token distribution system.

Airdrop 2: Distribution to Decentralization Stakers 175,000 INDY will be distributed as a reward to users who helped boost decentralization of the Cardano network by staking with a member of the Cardano Single Pool Alliance (CSPA). To have qualified for receiving this reward, a user had to have been staking a minimum of 10 ADA in one of 357 pools on November 6th, 2022. A total of 79,679 wallets qualified to be eligible to withdraw rewards. Each user who connects a qualified wallet to the Indigo Web App will be eligible for a one-time withdrawal of 5 INDY on a first come first serve basis.

Indigo Liquidity Bootstrapping Event and Liquidity Pool Creation In partnership with Minswap, Indigo will begin a Liquidity Bootstrapping Event (the "LBE") on November 14th 2022[10]. The goal of the LBE is to use a decentralized and transparent process to discover a fair price for INDY. After the LBE starts, users can deposit ADA into the Minswap Launch Bowl. Deposited ADA will be used to create INDY/ADA Liquidity Pools (a "LP").

The Minswap LP will consist of 75% of deposited ADA in the LBE paired with 262,500 INDY. Depending on slippage analysis at the time of the LBE end date of November 20th 2022, 25% of deposited ADA in the LBE paired with 87,500 INDY will be used to create LPs on either one or two DEXs approved by the Indigo community.

2.5.4 Indigo Rewards

Stability Pool stakers contribute to maintaining the solvency of the protocol and the iAsset pegs. In return for staking their iAsset, Indigo offers rewards in the form of ADA from liquidated CDPs and INDY. INDY is rewarded each Cardano epoch (every five days) and determined by the market cap of the iAsset as well as how much iAsset is being staked relative to other iAssets. The less iAsset that is staked in a Stability Pool relative to the total number of iAsset minted, the higher the INDY reward; the more iAsset that's staked, the less the INDY reward.

A benefit of iAsset composability is that they can be provided as liquidity to any Decentralized Exchange (a "DEX"). Having iAssets available on several DEXs is a key factor to promote Indigo's integration into the broader ecosystem, allowing other users to obtain and use iAssets without having to manage a CDP.

Users who provide liquidity to DAO voted DEXs and participate in the DEX farming program. Indigo rewards users who provide iAsset liquidity by having the DAO to vote on rewards allocation towards choice DEXs and receive INDY rewards.

Members who participate in Governance by casting a vote at least once every ninety days (configurable itself by Member vote) are rewarded with INDY each epoch. Each epoch, INDY is unlocked and distributed to all qualifying Members. The amount of INDY each Member receives is based on the ratio of a Member's stake relative to the total amount of INDY staked

$$a = \frac{bc}{\sum_{i=1}^{|m|} m_i}$$

Where:

- a is the amount of INDY a Member is rewarded
- b is the amount of INDY a Member has staked
- c is the amount of INDY rewarded to all Members for the epoch
- m is the collection of INDY amounts staked by all Members

Table 2: Distribution schedule of INDY unlocked every epoch for Stability rewards

Category	**# INDY per Epoch**
Liquidity	33563
Governance	2398

Users can withdraw their accumulated INDY staking rewards (the sum of a for each epoch they're owed rewards) via the Indigo Web App. Unclaimed rewards are withdrawable for three months. Any rewards not claimed within three months after being rewarded are redistributed by DAO vote.

[10]More information about Indigo's LBE will be available on Indigo's Medium.

2.6 Stability Pools

A Stability Pool (a "SP") helps maintain iAsset solvency by acting as the source of liquidity to repay debt from liquidated CDPs, thus intending all minted iAsset supply to remain overcollateralized.

Every supported iAsset has its own SP (e.g., iBTC SP, iETH SP). A user can deposit corresponding iAsset into a SP to become a SP staker (a "SP staker"). SP stakers provide stability to the protocol by offering their iAssets to be used for liquidations.

SP Liquidation ("SPL") is the process of utilizing a SP to liquidate a CDP, where iAsset deposited in a SP are burned to repay the debt of an undercollateralized CDP. In exchange, SP stakers earn a share of the collateral that was confiscated from liquidated CDPs. When CR falls below the iAsset LR, the CDP is considered insolvent and subject to liquidation, which amounts to canceling the debt where:

1. the same amount of iAsset debited by the CDP is burned from the corresponding SP; and

2. the collateral from the CDP is proportionally distributed to SP stakers.

As CDPs become liquidated, SP stakers lose a pro-rata share of their iAsset deposits while gaining a pro-rata share of the liquidated collateral. An incentive for SP stakers to participate in SPL is the possibility of earning net gains from liquidations. Under normal circumstances, the value of the collateral earned may be greater than the value of the canceled debt, because a liquidated CDP is likely to have a CR value above 100% (the value of the iAsset).

SPL first requires that CDPs are frozen. Each liquidation request of a CDP is executed against its iAsset's associated single SP. Optionally, users can make requests for CDPs to be merged. As illustrated in the CDP merge figure, three CDPs could be merged into a single CDP. The resulting merged CDP can then be liquidated against the SP. While only a single liquidation can occur per SP at once, multiple CDPs can be merged in parallel. Merging CDPs effectively enables multiple frozen CDPs to be liquidated simultaneously.

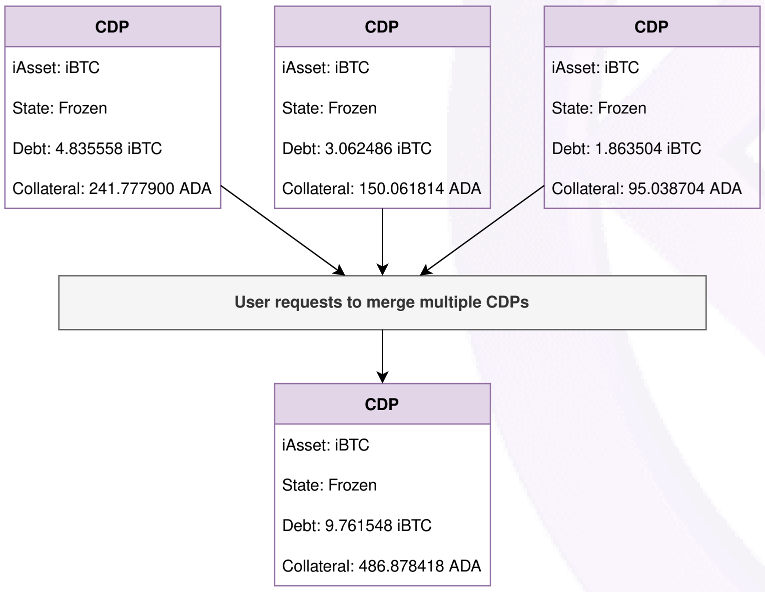


Figure 3: Three CDPs being merged into one

Indigo allows for both full and partial liquidations. A full liquidation, as illustrated in the SPL figure, repays all debt of a CDP and closes the CDP. A partial liquidation, as illustrated in the partial SPL figure, repays some debt of a CDP and keeps the remaining position frozen. If a CDP debt is higher than the entire amount of iAssets in the related SP, the protocol attempts to cancel as much debt as possible with the iAsset supply available. Any remaining non-liquidated collateral and debt of the CDP remains frozen until more iAsset is deposited into the SP and another liquidation is initiated.

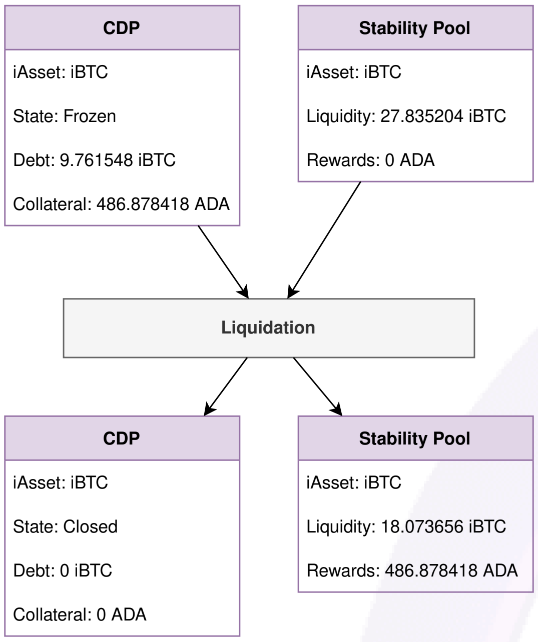


Figure 4: Illustration of a full liquidation where sufficient funds are present in the SP

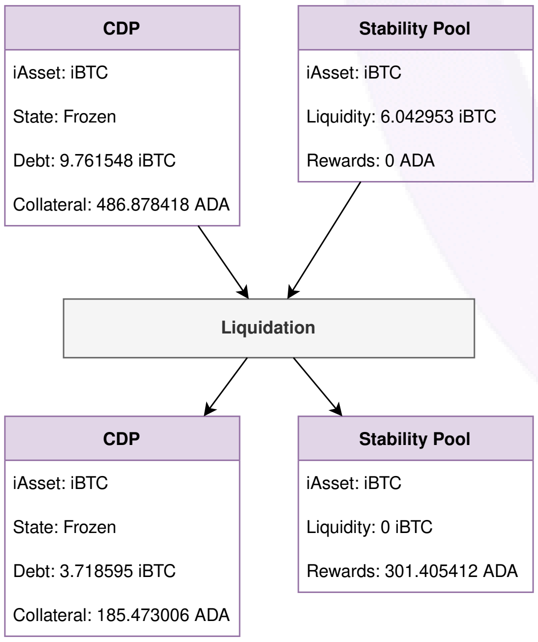


Figure 5: Illustration of a partial liquidation where there are insufficient funds present in the SP

Values for the liquidated CDP and associated SP can be calculated using:

$$w = a - min\{a, b\}$$
$$x = d - min\{a, e\}\frac{d}{a}$$
$$y = e - min\{a, e\}$$
$$z = g + min\{a, e\}\frac{d}{a}$$

Where:

- w is the updated debt of the CDP after liquidation
- x is the updated amount of collateral in the CDP after liquidation
- y is the updated amount of iAsset in the SP after liquidation
- z is the updated amount of ADA rewarded to the SP after liquidation
- a is the amount of debt of the CDP before liquidation
- b is the amount of iAsset in the SP
- d is the amount of collateral in the CDP before liquidation
- e is the amount of iAsset in the SP before liquidation
- g is the amount of ADA rewarded to the SP before liquidation

2.6.1 Stability Pool Staking Fees

Users can stake and unstake iAssets from SPs at any time. To stake iAsset, a user needs to create a SP account by depositing 7 ADA and the amount of iAsset they desire to stake. 2 ADA is returnable to the user upon closing the SP account, which involves withdrawing all their iAsset and earned rewards. 5 ADA is taken as a fee. Users pay a 1 ADA fee for each new iAsset deposit or withdraw against their SP account.

SP fees are collected and distributed to all SP stakers as part of liquidation rewards.

2.6.2 Stability Pool Withdraw Fee

Users can also be charged a withdrawal fee modifiable by DAO for the Stability Pool to counteract potential reward gaming and ensure efficient processing of liquidations. This fee could be redistributed as a loyalty bonus to other Stability Pool Providers deposited in the pool at the time of withdrawal.

For instance an inconvenience fee of 0.5% is charged for withdrawal.

Given the withdrawal is in iAssets, the fee can be charged, and retain each individual iAssets Stability pool to the DAO UTXO.

$$a = b * c$$

- a is the fee retained in the Stability Pool
- b is the current withdrawal fee constant
- c is the amount of iAsset withdrawn from the SP

2.6.3 Stability Pool Liquidation Rewards

As liquidations occur, SP stakers lose a pro-rata share of iAsset deposits and gain a pro-rata share of ADA rewards. A SP "product constant" maintains mathematical state of liquidations occurred. When a SP is first created, its product constant is set to one. Upon liquidation, the product constant can be calculated using the formula:

$$c = a\left(1 - \frac{b}{d}\right)$$

Where:

- c is the new product constant
- a is the current product constant
- b is the amount of iAsset debited from the SP for the liquidation
- d is the total amount of iAsset in the SP

A SP "compounded constant" maintains the mathematical state of rewards earned from liquidations relative to the product constant. When a SP is first created, its compounded constant is zero. Upon liquidation, the compounded constant can be calculated using the formula:

$$r = a + \frac{bc}{d}$$

Where:

- r is the new compounded constant
- a is the current compounded constant
- b is the amount of ADA earned during the liquidation
- c is the product constant before the liquidation
- d is the total amount of iAsset in the SP before the liquidation

When an action is taken against a SP, such as a deposit of an iAsset or a liquidation, its state is updated. The SP state data structure – represented in the SP state table – is stored within the UTXO of the SP; "iAsset Deposit" records the number of iAsset in the SP deposited by all users.

A SP epoch ends when all iAsset from a SP is drained via liquidations. Epoch is a running tally of the number of occurrences there have been when the SP's total iAsset deposit reached zero. Upon updating the SP state, if the total iAsset in the SP is to be set to zero, then this marks the end of an epoch. At the end of an epoch, the following occurs:

- Epoch is recorded in a UTXO paired with the compounded constant value after the latest liquidation
- The SP state is updated with the values:
  - epoch incremented by one;
  - product constant set to one; and
  - compounded constant set to zero.

Table 3: State stored upon updates to a SP

Name	**Description**
Product Constant	The new product constant (c)
Compounded Constant	The new compounded reward (r)
iAsset Deposit	The updated amount of iAsset deposited into the SP
Epoch	The current epoch

When a user deposits iAsset into a SP, a SP staker "account record" is created or updated for that user's account. The account record is represented the same as SP state and stored within the UTXO of the SP staker's position; iAsset Deposit records the number of iAssets owned individually by the SP staker. All other values for the account record are copied from the SP state.

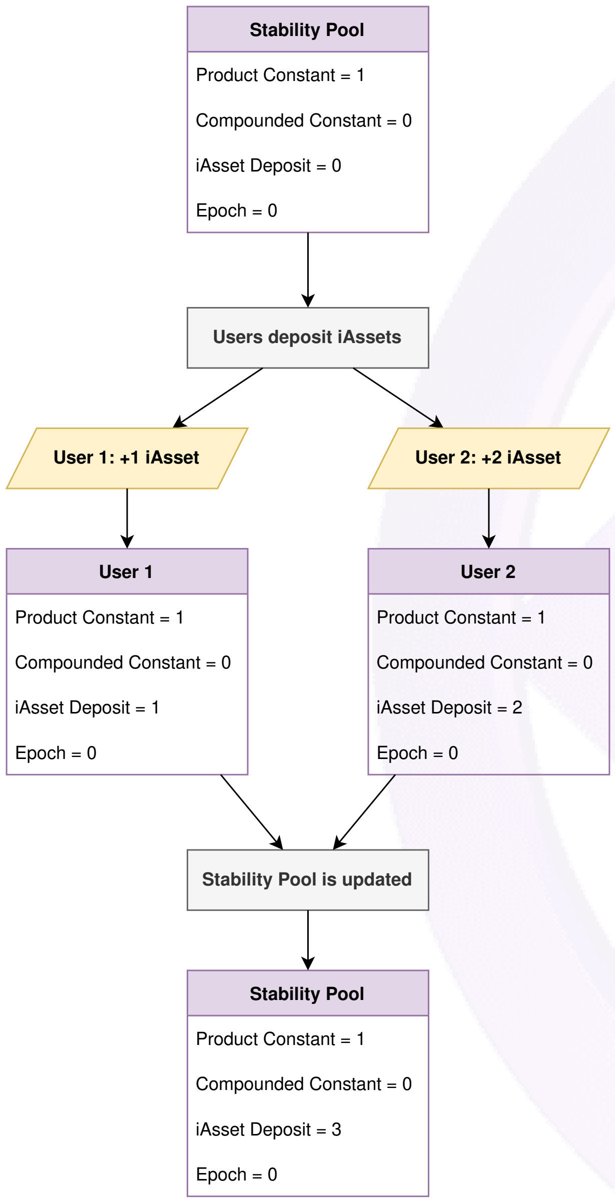


Figure 6: iAsset being deposited into a new SP

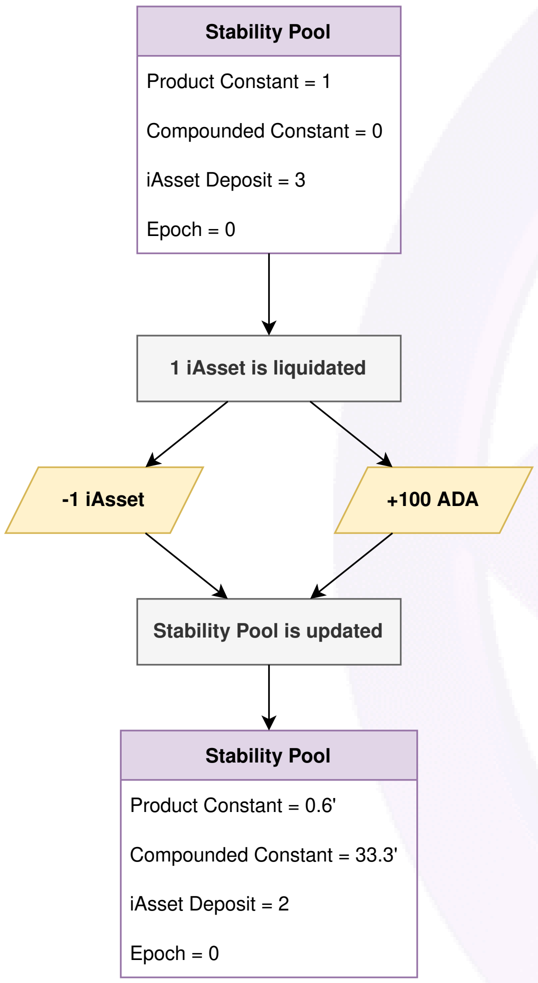


Figure 7: SP state being updated after a liquidation occurs

During a liquidation, iAsset is extracted from a SP. Proportionally, the ownership share of the iAsset within each SP staker's position is reduced. If the epoch in the account record matches the epoch in the SP state, the amount of iAsset an individual SP staker holds can be calculated using:

$$m = a\frac{c}{b}$$

Where:

- m is the amount of iAsset owed to the SP staker
- a is the amount of iAsset the SP staker deposited (retrieved from the account record)
- c is the current product constant (retrieved from the SP state)
- b is the product constant when the SP staker deposited their iAsset (retrieved from the account record)

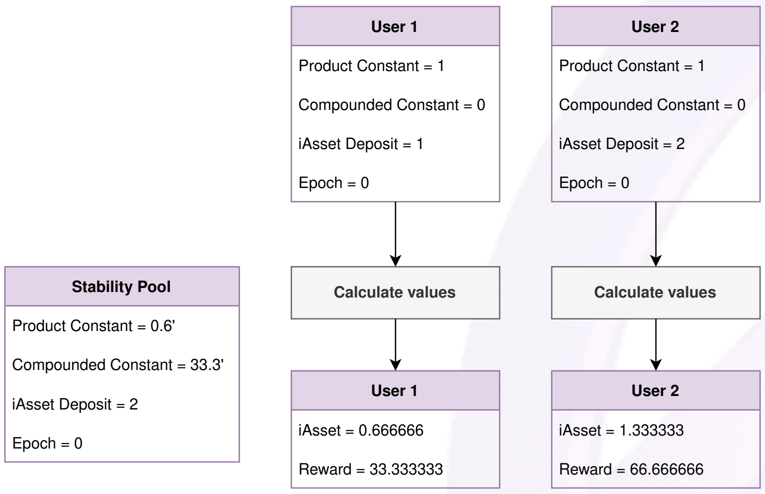


Figure 8: SP staker rewards after a liquidation has occurred

If the epoch in the account record does not match the epoch in the SP state, then the amount of iAsset owned to the SP staker (m) is zero. This is due to all the user's iAsset having been burned during a previous epoch.

During a liquidation, an ADA reward is deposited into the SP. Proportionally, the share of ADA rewards each SP staker is owed increases. The formula to calculate how much ADA an individual SP staker is rewarded from the SP is:

$$l = a\frac{r-d}{b}$$

Where:

- l is the amount of ADA owed to the SP staker
- a is the amount of iAsset the SP staker deposited (retrieved from the account record)
- r is the current compounded constant (retrieved from the SP state or recorded compounded constant for the matching epoch)
- d is the compounded constant when the SP staker deposited their iAsset (retrieved from the account record)
- b is the product constant when the SP staker deposited their iAsset (retrieved from the account record)

If an account record's epoch does not match the epoch of the SP state, then r is set to the latest recorded compounded constant for the epoch. This is due to the compounding constant resetting to zero after an epoch ends, therefore all SP staker positions during that epoch would be closed because all their iAsset would have been utilized during liquidations.

When a SP staker modifies their position, either by depositing or withdrawing iAsset or ADA reward, then their previous position is considered closed, and a new position is created. If a user withdraws all their iAsset, then a new position is not opened. The SP state is also updated to reflect the new deposit or withdrawal, i.e., the iAsset Deposit is updated by the amount of iAsset deposited or withdrawn.

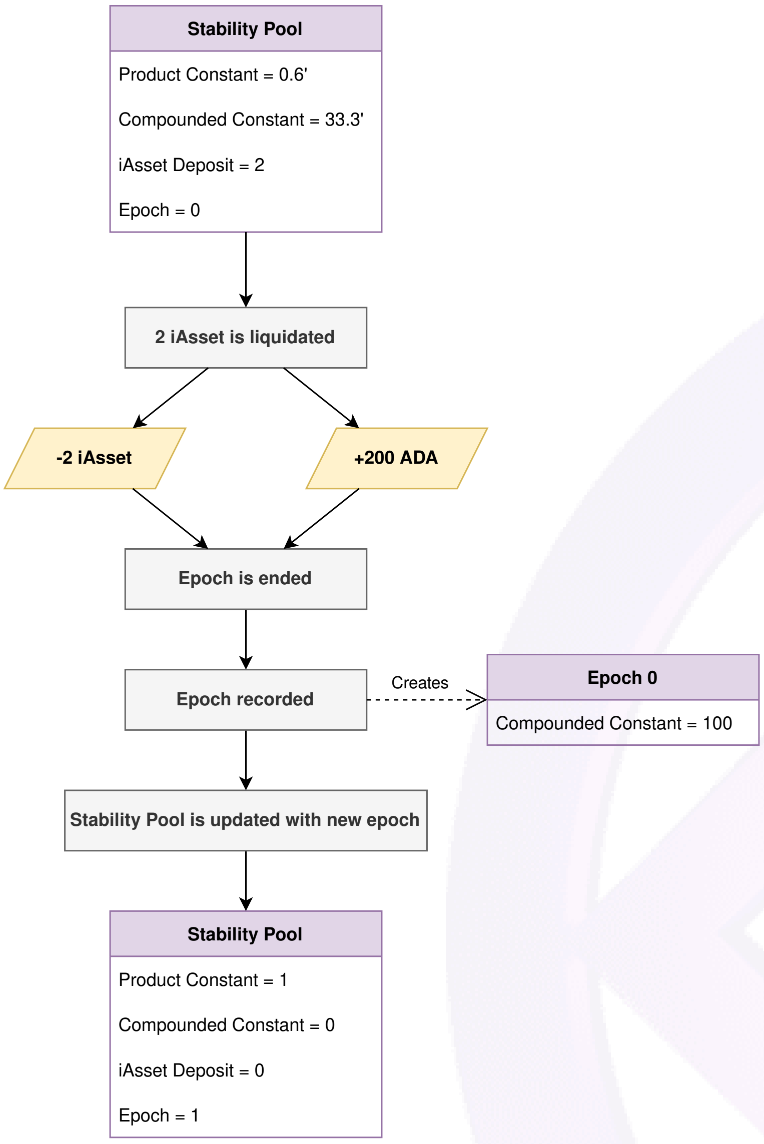


Figure 9: Illustration of a new SP epoch beginning after a liquidation drains all iAsset

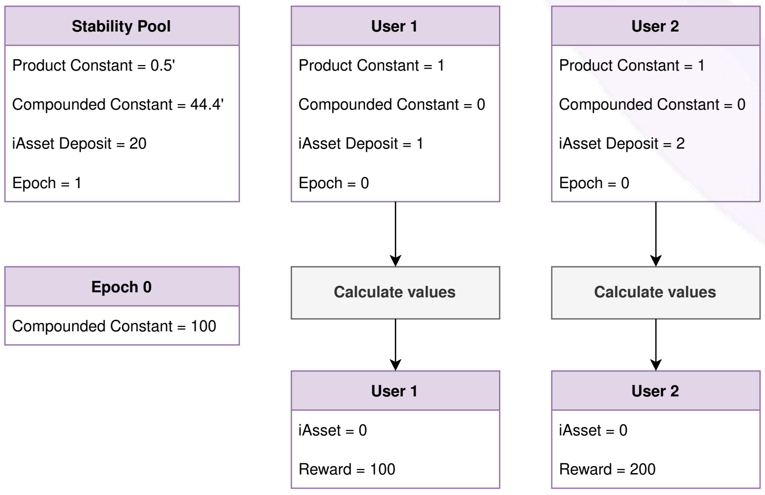


Figure 10: SP staker rewards after SP has been drained and a new epoch has begun

2.6.4 Liquidity State Transitions

The Stability Pool mechanism requires the use of a single UTxO to store the state of the pool. This global state leads to contention against the Stability Pool UTxO. Liquidity State Transitions is a design pattern implemented for the Stability Pool contracts to essentially eliminate the contention of users who are attempting to interact with the Stability Pool.

Liquidity State Transitions take the approach of separating the action of the user into two steps: a request transaction and an execution transaction.

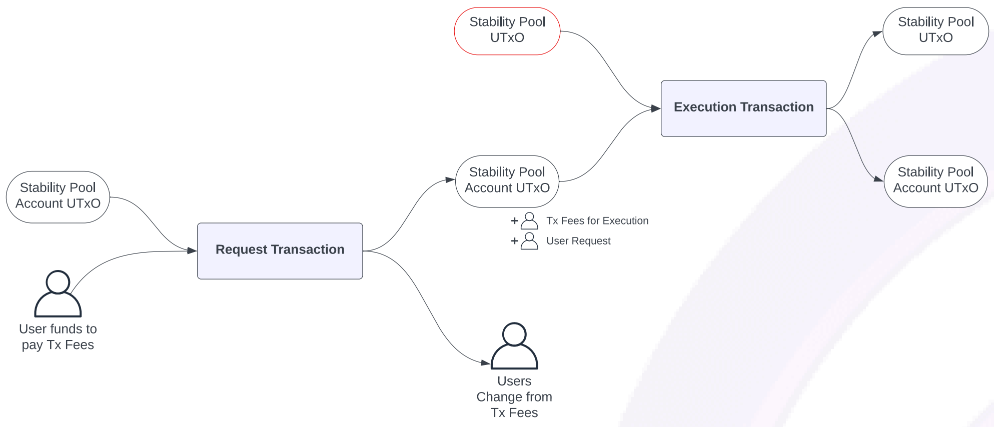

Stability Pool UTxO
Execution Transaction
Stability Pool UTxO
Stability Pool Account UTxO
Stability Pool Account UTxO
+ Tx Fees for Execution
+ User Request
Stability Pool Account UTxO
Request Transaction
User funds to pay Tx Fees
Users Change from Tx Fees

Figure 11: Transaction flow of a Liquidity State Transition transaction

Referencing the above figure, instead of processing the request and execution of the stability pool adjustment in the same transaction, we separate it into two transactions. This method greatly increases the assurance that the user's intended action will be executed on the Cardano blockchain, rather than being impeded due to the submission of a contentious transaction.

Once the request is recorded on-chain, it can then be executed by any user who wants to process the request against the Stability Pool, including the requestor.

Additionally, there are two ancillary benefits with LST:

1. Executing requests can be transaction-chained against other executions; and
2. The user never has to re-sign an execution with their wallet if a preceding transaction fails due to contention.

Since the execution doesn't require a wallet signature and has the potential to be transaction-chained, we increase our throughput by processing multiple executions against a Stability Pool in a single block.

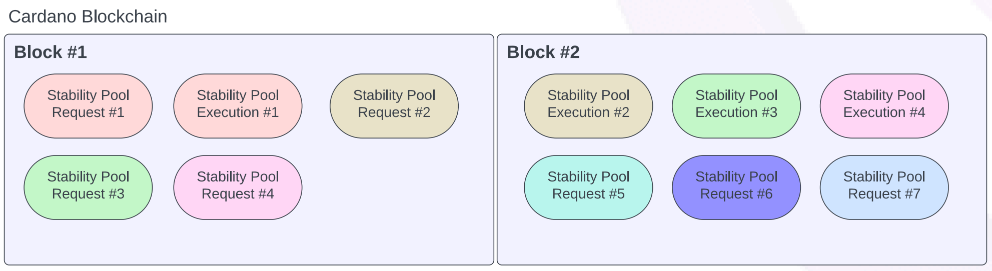

Cardano Blockchain
Block #1
Stability Pool Request #1
Stability Pool Execution #1
Stability Pool Request #2
Stability Pool Request #3
Stability Pool Request #4
Block #2
Stability Pool Execution #2
Stability Pool Execution #3
Stability Pool Execution #4
Stability Pool Request #5
Stability Pool Request #6
Stability Pool Request #7

Figure 12: Example of how multiple Stability Pool transactions can be stored in a single block

As shown in the above figure, submitting requests and executing requests can both fit into a single block, which means that a request can be executed just as it is now, however, it simply occurs in two separate transactions.

2.7 Oracles

To determine the value of collateral held within CDPs and the intended prices of iAssets, Indigo makes use of Oracles[11] available on Cardano. An Oracle queries external data sources for information and makes that information available on-chain.

[11]Oracles provide a way for decentralized blockchain applications to access existing data sources.

Indigo is designed to be Oracle agnostic, meaning that it can support any Oracle that publishes data on the Cardano blockchain so long as the data format conforms with the protocol's specifications defined in the CDP section.

2.8 iAsset Price Stability

iAssets are pegged to tracked assets. To maintain price pegs, Indigo relies on protocol rules to incentivize arbitrageurs and market forces to stabilize prices. These rules ensure that iAssets are always fully collateralized, giving further confidence to users that iAsset prices will match their counterparts.

Periodically, Indigo receives price data from the outside world via Oracles. The rate at which price feeds are updated is configurable, and at launch will be set to once per hour. After price is updated, CRs are adjusted across the protocol, allowing for liquidations to occur for CDPs whose CR falls below the iAsset's LR.

2.8.1 Managing Liquidation Ratio for Peg

If an iAsset drops in price relative to its peg, it provides CDP owners an opportunity to buy the iAsset to repay their loan at a discount. This can cause buying pressure on the iAsset to rise its price. If there is an abundance of iAsset supply, Indigo can increase LR towards the iAsset mode CR.

Each CDP has its own CR. The iAsset mode CR represents the most frequent CR value users select for their CDPs. By moving LR towards the mode CR, probability of liquidation increases, incentivizing users to close their CDPs, which can cause iAsset buying pressure and reduced iAsset supply.

A higher LR results in a higher liquidation risk, reduces the maximum leverage utilizable, and increases the margin of arbitrage value for Stability Pool stakers. This encourages CDP owners to be more prudent with their positions and incentivise CDP owners to reduce their debts.

A low LR reduces the cost of minting iAsset supply and maximizes the leverage utilizable. This creates an incentive for users to create new iAsset supply, hence is why Indigo's iAssets will initially be launched with a LR of 110%. An iAsset LR of 110% forces the price of the iAsset to be no more than 10% above its peg by creating an arbitrage opportunity. Users at any time can mint iAsset at a cost of 10% higher than the iAsset's pegged price, allowing iAsset to be immediately sold if the market premium is higher than 10%. iAsset trading above its peg also offers an opportunity to borrow at a lower cost, further incentivizing more supply to be minted and possibly creating additional sell pressure if users choose to take advantage of the leverage.

2.8.2 Managing Maintenance Ratio for Peg

Maintenance Ratio, a parameter attached to the iAsset, caps the maximum permissible debt that can be secured against a given amount of collateral. The MR can be set to be equal to the Liquidation Ratio (LR) to effectively disable it, allowing its activation only when necessary to control excessive minting. This approach mirrors the impact of minting fee adjustments, but specifically targets positions with lower collateralization.

If there is an abundance of iAsset demand and limited supply, Indigo can decrease MR towards LR%. This in turn reduces the cost of minting iAsset, pushing the price of the iAsset down. Indigo's quick liquidation mechanism via Stability Pools allows for high capital efficiency and support for very low collateralization while still providing incentive for users to participate in arbitrage. MR value setting considers the LR of iAssets, ensuring that iAssets are always overcollateralized irrespective of any market conditions or possible future events

2.8.3 Managing Interest Rates for Peg

Implementing an interest rate on CDPs adds a crucial layer of stability, encouraging responsible borrowing and timely repayment. If the peg oscillates frequently around the redemption rate. The base interest can be increased. By increasing or decreasing interest, the Indigo DAO can choose whether to incentivize or disincentivize the minting or spot buying of $iUSD and by how much. A heightened interest rate applies when redemption occurs implying depeg situation.

Having a cost associated with opening debt encourages debtors to repay their debts.

2.8.4 Managing Redemptions for Peg

Redemption Margin Ratio for all assets which can be adjusted individually. This mandatory RMR can act as a safeguard to maintain peg stability, reducing the risk of total collateral losses without reducing capital efficiency.

Raising the RMR has a milder effect on positions compared to raising the LR. For example, if a position is at 200%, raising the LR to 200% would liquidate the position, resulting in a complete loss of net equity (50%). Conversely, increasing the RMR to 200% only exposes the position to potential net equity returns of 50%.

2.9 Governance

Governance is the decentralized voting process through which *proposals* for updating the protocol are introduced and either accepted or rejected by the community (collectively known as the "Indigo DAO"). All change to the protocol must go through governance.

Indigo has a 3-pillar structure built for long term sustainability:

1. **Indigo DAO** – Decentralized association of members governing the protocol.
2. **Indigo Foundation** – Foundation Company incorporated in the Cayman Islands for interacting with the real-world on behalf of the Indigo DAO.
3. **Indigo Laboratories, Inc.** – A Wyoming corporation contracted by the Indigo Foundation responsible for development of Indigo and blockchain technologies.

2.9.1 Indigo DAO

The Indigo DAO (the "DAO") is an informal non-jurisdictional, non-hierarchical, and nonprofit association of fluctuating individuals and entities who are uncoordinated and act together using a token. The DAO owns and controls the Indigo Protocol. All changes to the protocol must go through governance. Governance is the decentralized voting process through which proposals for updating the protocol are introduced and either accepted or rejected by the Indigo DAO Members.

INDY serves as Indigo DAO's utility token with one of its purposes being to allow voting on DAO proposals. Users who stake their INDY in Indigo's governance thereby become a DAO Member (a "Member") and can vote on proposals.

Members who wish to assist in managing the administrative and technical operations of Indigo (e.g.: organizing meetings of Members, submitting governance Proposals, or leading Working Groups) can be elected by other Members and become Core Contributors.

Protocol Working Group Protocol working group (PWG) is elected by the community, the members can be expanded or removed via a DAO vote. The role of PWG involves gathering of community feedback and assisting to setup proposals

Technical Working Group Technical working group (TWG) is elected by the community, the members can be expanded or removed via a DAO vote.

The role of TWG is to provide technical input on behalf of the community

2.9.2 Indigo Foundation

The Indigo Foundation (the "Foundation") entity provides an extremely flexible framework that supports off-chain functions necessary for executing the intent of the Indigo DAO. While the Indigo DAO is not a legal entity, the Foundation is, and therefore can enter into legal agreements with other real-world entities. The Foundation is established to help implement approved actions of the DAO that cannot otherwise be implemented in an automated or computational manner. The Foundation can engage with governmental authorities (for tax, regulatory, or other purposes), contract with vendors, and educate the community about Indigo – all as directed by the DAO.

The Foundation's authority is limited to implementing the votes of the DAO and otherwise supporting Indigo. The DAO may vote to amend the responsibilities of the Foundation at any time. The Foundation does not have possession of or control of any Indigo or user funds. The DAO is required to fund the Foundation and provide the Foundation with any tokens needed to make payments to third party vendors.

2.9.3 Governance Process

An owner of INDY who chooses to stake INDY within Indigo becomes a Member and obtains the right to vote on proposals. A vote can be either in the form of yes, indicating favor of passing the proposal, or no, indicating favor of rejecting the proposal. Each Member receives voting power weighted by their amount of INDY staked.

The Governance Process consists of three phases.

Step 1 – Temperature Check: A user creates and submits their idea to the Indigo Forum. The idea will be reviewed by Moderators and Indigo Forum users for consistency with the Indigo DAO Constitution. Forum users will review and provide comments or suggested improvements to the idea, and eventually vote on it within the Forum.

Step 2 – Proposal: If a Temperature Check results in a positive outcome, a user needs to deposit INDY to submit a proposal on-chain. In addition, the user submitting the proposal should also create voting shards by depositing some ADA. Voting shards will maintain a record of votes and are meant to enhance on-chain voting performance. Members can vote on the proposal using their staked INDY. Indigo's Adaptive Quorum Biasing mechanism automatically adjusts the threshold to determine how many positive votes are required for the proposal to pass.

Step 3 – Execution: After a proposal's Voting Period ends, it moves to the execution phase. If the proposal passed, users could execute it and the proposal creator can retrieve their INDY deposit as well as their ADA deposit within each voting shard.

To stop malicious users from spamming proposals on-chain and creating a headache for INDY DAO Members to manage, an exponential INDY deposit is used when multiple proposals are taking place at the same time. When a proposal is created the number of active proposals at that time is taken into account to calculate the total required deposit of INDY:

$$d = p * 2^a$$

Where:

- d is the required INDY deposit
- p is the proposal deposit determined by the protocol parameters
- a is the number of active proposals at the time of proposal creation

If the proposal fails, the proposal is closed and the proposal creator loses their INDY deposit. The INDY is instead sent to the Treasury.

2.9.4 Staking

Users who stake their INDY in Indigo's governance (thereby becoming a "Member") can vote on proposals. A vote can be either in the form of *yes*, indicating favor of passing the proposal, or *no*, indicating favor of rejecting the proposal. Each INDY staker receives voting power weighted by their amount of INDY staked and must either use either all or none of their voting power.

When a Member votes on a proposal, their INDY stake is locked until that proposal's Voting Period has concluded (i.e., either approved, rejected, or expired). Locked INDY cannot be withdrawn from the protocol. If a Member votes on multiple proposals, their INDY is unlocked after the most recently created proposal they voted on concludes. If their INDY is in an unlocked state, then users can withdraw their INDY stake.

After casting a vote, it cannot be changed or undone. Voting power is set to the total amount of INDY staked at the time of casting. If an Member deposits additional INDY into their position, they can use that INDY in addition to the existing locked INDY to vote on another proposal but cannot use that additional INDY to vote on a proposal they've already voted on. If the user deposits INDY after casting a vote and before casting another vote, then the INDY can be withdrawn. After depositing INDY and casting a vote for another proposal, all deposited INDY becomes locked and cannot be withdrawn until the end of the proposal.

2.9.5 Governance Rewards

Members who participate in Governance by casting a vote at least once every ninety days (configurable itself by DAO vote) are rewarded with INDY each epoch. Each epoch, INDY is unlocked and distributed to all qualifying Members. The amount of INDY each Member receives is based on the ratio of a Member's stake relative to the total amount of INDY staked, and can be calculated using:

$$a = \frac{bc}{\sum_{i=1}^{|m|} m_i}$$

Where:

- a is the amount of INDY a Member is rewarded
- b is the amount of INDY a Member has staked
- c is the amount of INDY rewarded to all Members for the epoch
- m is the collection of INDY amounts staked by all Members

Table 4: Distribution schedule of INDY unlocked every epoch for Governance rewards

INDY per Epoch
2,398

2.9.6 Adaptive Quorum Biasing

A proposal is considered passed when the ratio of *yes* votes over *no* votes exceeds the quorum threshold. Indigo uses a dynamic vote-threshold mechanism called Adaptive Quorum Biasing ("AQB") to calculate the quorum threshold value. AQB lowers the quorum threshold as more INDY is used to vote. If voter participation is low, then a high majority of those votes must be in favor of the proposal. If voter participation is high, then a lower majority of those votes must be in favor of the proposal. Always at least 50% of votes must be in favor of a proposal for it to pass.

In addition to AQB, the Indigo DAO has the ability to set a global parameter on the minimum amount of INDY needed for a vote to pass as well. This mechanism is meant as a safety parameter and is not to restrict the passing of proposals but to instead limit the ability of proposals to have little or no votes and creating a passing proposal.

For example, if 29% of all circulating supply of INDY is used to vote during a proposal's Voting Period, the quorum threshold for that proposal would be set to 66%. This means that 66% or more of the total INDY used for voting would be required to vote *yes* for the proposal to be considered passed. If more than 34% of the total INDY used for voting voted *no*, then the proposal would fail.

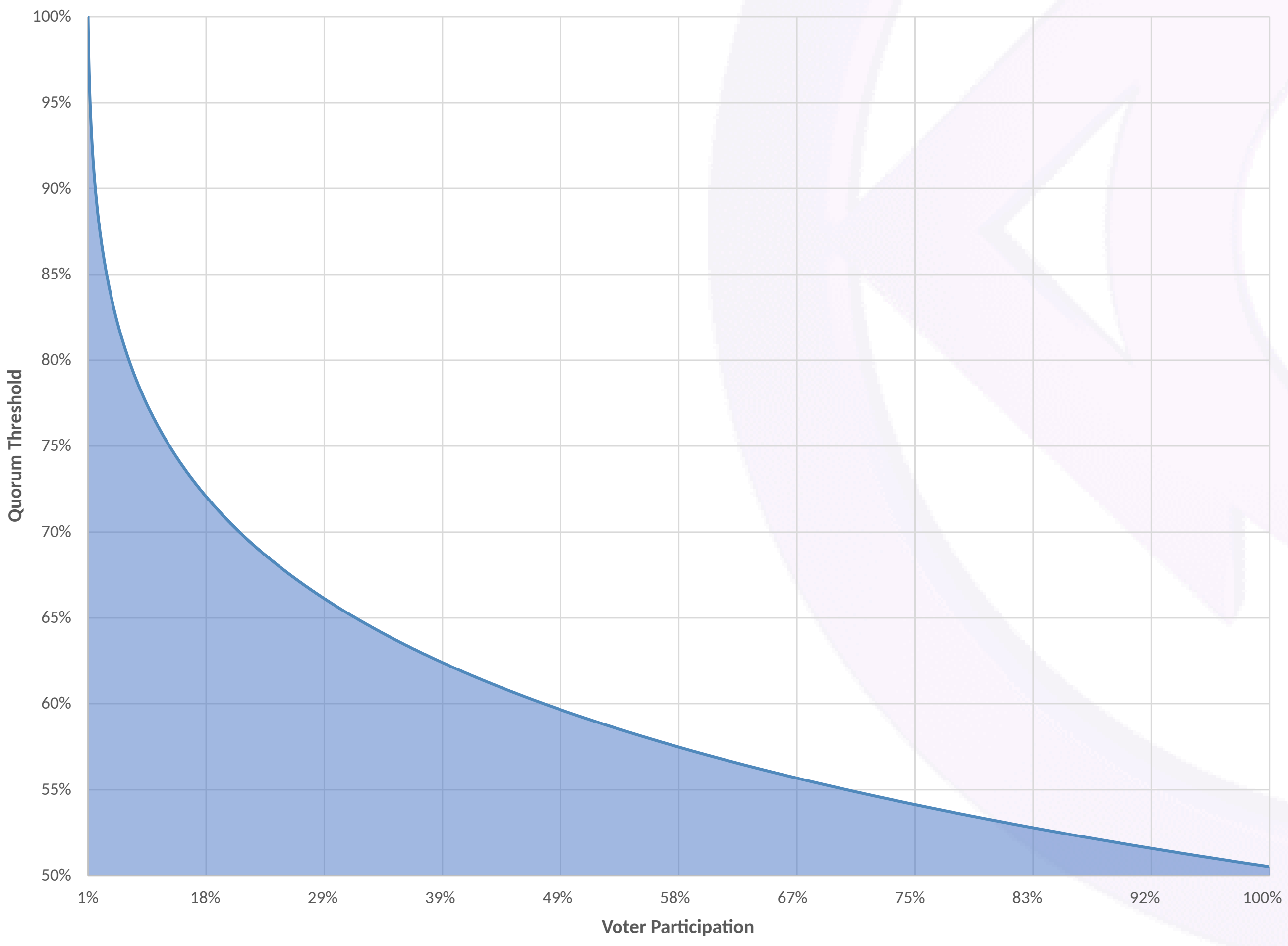


Figure 13: Illustration of quorum threshold decreasing as voter participation increases

To determine if a proposal is approved, the electorate (e) first needs to be calculated. e is INDY circulating supply at the time of a proposal's conclusion. INDY has a fixed distribution schedule, so e can be derived by taking the launch time of the protocol, the end time of the proposal, and other values related to Indigo's token distribution schedule set at the time of protocol launch.

To calculate e the following logic can be used:

$$e = \left(\begin{array}{c} f : (a,b) \mapsto \left(\begin{array}{c} \text{let } x \text{ equal } \min\left\{ \left\lfloor \frac{d-l}{5} \right\rfloor - a + 1, 73\left|b\right| \right\} \\ \begin{cases} \sum_{i=1}^{x} \left(\begin{array}{c} \text{let } y \text{ equal } \left\lfloor t \frac{b_{\left\lceil \frac{i}{73} \right\rceil}}{73} \right\rfloor \\ \begin{cases} y+1 & \text{if } i-1 < \left\lfloor t \sum b - \sum_{j=1}^{|b|} 73 \left\lfloor \frac{tb_j}{73} \right\rfloor \right\rfloor \\ y & \text{otherwise} \end{cases} \end{array} \right) & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \end{array} \right) \\ \text{let } z \text{ equal } \left(\begin{array}{c} \text{let } x \text{ equal } \min\left\{ \left\lfloor \frac{d-l-o}{365 \div 12} \right\rfloor + 1, q \right\} \\ \begin{cases} 0 & \text{if } x < 0 \\ \left\lfloor \frac{tp}{q} \right\rfloor & \text{if } x = 0 \text{ and } d - l \geq 0 \\ \left\lfloor \frac{xtp}{q} \right\rfloor & \text{otherwise} \end{cases} \end{array} \right) \\ \sum_{i=1}^{|a|} f\left(a_i, b_i\right) + z + \begin{cases} c & \text{if } d \geq l \\ 0 & \text{if } d < l \end{cases} \end{array} \right)$$

Where:

- a is a set of delays for token distribution schedules (set at protocol launch)
- b is a set of vesting distribution schedules (set at protocol launch)
- c is the amount of INDY unlocked upon Indigo mainnet launch (set at protocol launch)
- d is the date of the proposal's conclusion
- l is the date of the first epoch after the launch of Indigo mainnet (set at protocol launch)
- o is the offset for the start of Indigo's team distribution (set at protocol launch)
- p is the percentage of INDY total supply allocated to the Indigo team (set at protocol launch)
- q is the total number of months the Indigo team distribution lasts for (set at protocol launch)
- t is the total supply of INDY (set at protocol launch)

Vesting schedules defined by b are represented as a set of sets containing the percentage of token supply to be distributed per year, with each value in the subset representing an individual year. For example, consider the following set:

$$b = \{\{0.01, 0.02, 0.03\}, \{0.05, 0.1\}\}$$

This defines two vesting schedules (two being the size of the set b). The first vesting schedule in b, referenced as b_1, describes a three-year vesting schedule (three being the size of the subset b_1), with the first year distributing 1% (0.01 being 1%) of total token supply, the second 2%, and the third year 3%, for a total of 6% (0.06 being the sum of all values in the subset b_1) of tokens distributed over the three years.

Knowing e, a proposal's approval status can be calculated using the formula:

$$q = \left\lfloor \frac{v_y}{\sqrt{e}} - \frac{v_n}{\sqrt{v_y + v_n}} \right\rfloor$$

Where:

- q is the vote threshold
- e is the amount of INDY in circulation at time of the proposal's conclusion
- v_y is the number of *yes* votes
- v_n is the number of *no* votes.

If q is larger than 0, the proposal is passed. If q is equal to or less than 0, the proposal is failed.

2.9.7 Governance Sharding

Upon creation of a proposal, multiple voting UTXOs can be created to maintain records of votes. Each voting UTXO represents a shard. The total number of shards that can be created is defined by the *Total Shards* protocol parameter.

After creating a proposal, the proposal's creator can create shards, up to the number of Total Shards, by depositing ADA and submitting transactions. If, after the proposal creation time plus the time defined by the *Proposing Period* protocol parameter, there are fewer shards created than Total Shards, then the proposal is considered expired.

The amount of ADA required to deposit to create an individual shard is x, as calculated and described in the Minimum ADA to Create UTXO section. The proposal creator is required to deposit x ADA to create an individual shard. To prevent a proposal from expiring before all votes can be submitted, the proposal creator must deposit ADA totaling x multiplied by Total Shards. The deposited ADA is later returnable upon following correct voting procedures, as described in the Governance Proposal Process section.

To vote, an INDY staker selects a shard to track their allocation. Each shard records the total number of *yes* and *no* votes from users who voted using that shard. A shard can only record a vote from one user at a time. If a shard is in use by another user, then the user must select an alternative shard to use. If all shards are in use, then the user must wait until a shard becomes available.

At the end of the Voting Period, the shards can be closed. Upon closing, all votes from each shard can be tallied, and the final vote counts can be used to calculate whether the proposal has passed.

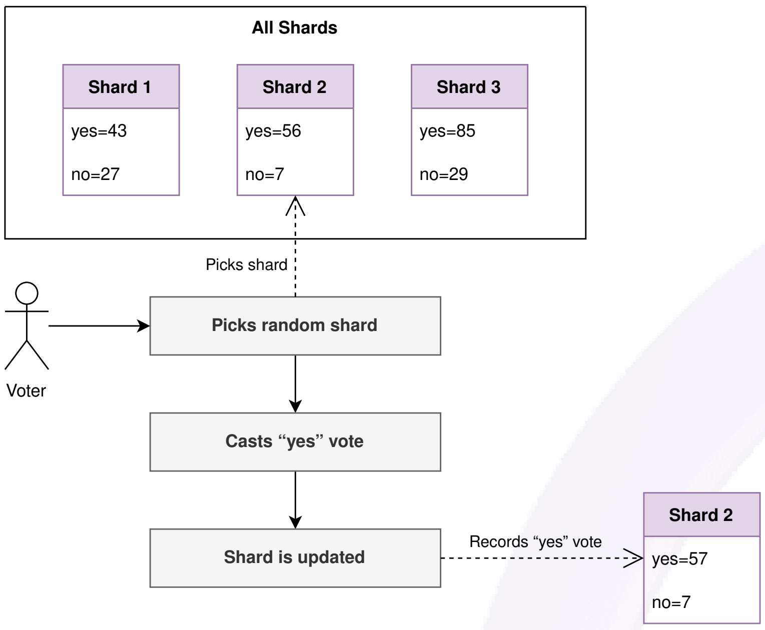


Figure 14: A voter selecting and casting their vote using a shard

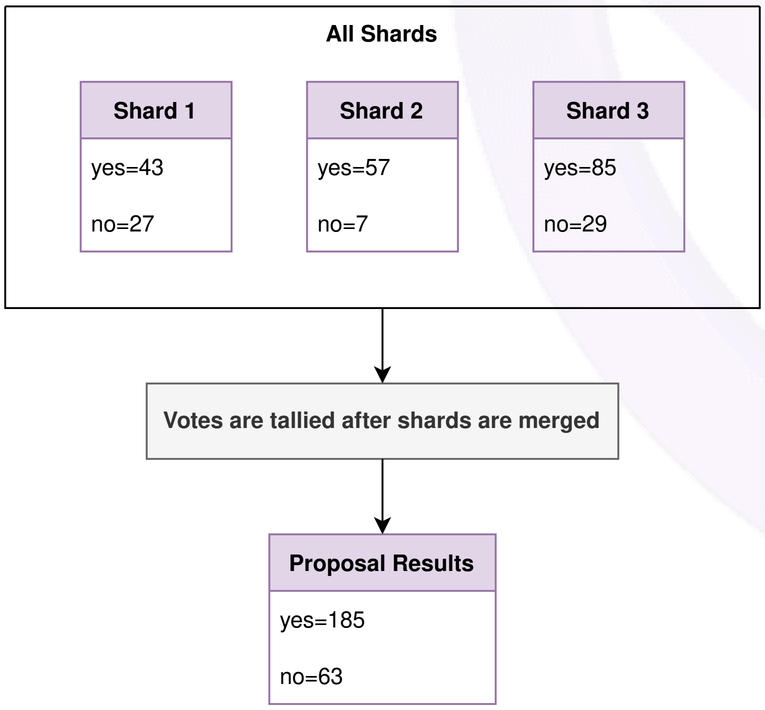


Figure 15: Shards being merged to tally votes after a Voting Period has ended

2.9.8 Governance Proposal Types

Users can submit the following type of proposals:

- **Whitelist an iAsset** – Propose that a new iAsset type be supported by the protocol. Attributes such as the LR and Oracle price feed must be provided.
- **Update an iAsset** – Propose that an existing iAsset's LR and/or Oracle price feed be updated. Nullifying an iAsset's Oracle price feed causes that iAsset to be no longer mintable; therefore, it is delisted from the protocol.
- **Text** – Propose that the Indigo DAO should adopt a proposal described textually. This formally records the DAO's intent on the blockchain but is not executed computationally, i.e., the proposal's executable message is non-actionable. A hash is stored on-chain, with the hash able to represent a Content Identifier (CID)[12] that references data on an external storage network.
- **Upgrade Protocol** – Propose that the protocol should be upgraded to a new version.
- **Update Protocol Parameters** – Propose that parameters describing protocol behavior be updated. Updateable parameters are shown in the Protocol Parameters table.

2.9.9 Protocol Parameters

Protocol parameters are updateable via proposals and define some behaviors of the protocol. They exist as a map of values inside a UTXO. Users and protocol functions can reference the values of the latest defined protocol parameters to utilize within transactions.

Table 5: Parameters that are able to be updated via an Update Protocol Parameters Governance Proposal

Parameter Name	**Description**
Effective Delay	The number of seconds after a passed proposal closes before it becomes eligible for execution.
Expiration Period	The maximum number of seconds allowed after a passed proposal closes for it to be executed. If the proposal isn't executed in time, then the proposal is considered expired.
Proposal Deposit	The amount of INDY that is required to be deposited to create a proposal. If a proposal passes, the INDY deposit is returnable to the owner. If a proposal fails, the INDY deposit is non-returnable, and instead is only transferable to the DAO Treasury.
Proposing Period	The maximum number of seconds allowed after a proposal is created for its shards to be created. If shards are not created by this time then the proposal fails and the creator loses their deposit.
Collateral Fee Percentage	The percentage of ADA to take as a fee when withdrawing collateral from CDPs or redeeming SPL rewards.
Total Shards	The total number of Governance Shards to utilize during Voting Periods of proposals.
Voting Period	The number of seconds a proposal remains open for voting after being created.
Minimum Quorum	The minimum number of INDY for a proposal to be able to be passed.
Max Treasury Lovelace Spend	The maximum amount of ADA (in lovelaces) allowed to be withdrawn from the treasury.
Max Treasury INDY Spend	The maximum amount of INDY (in INDYlaces) allowed to be withdrawn from the treasury.

[12] A CID is a self-describing content-addressed identifier containing 32 characters. A CID can be used to lookup data stored on decentralized networks such as Filecoin.

2.9.10 Governance Proposal Process

Any user can create a proposal by depositing a fixed amount of INDY into the protocol. The amount of INDY required is determined by the value of the *INDY Deposit* protocol parameter.

Once submitted, the proposal becomes eligible for the proposal creator to create shards. After one or more shards are created for a proposal, it can be voted on by INDY stakers until that proposal's Voting Period has concluded.

Proposals are recorded on-chain with an executable message encoding the specific effects of each one. Upon execution, the proposal will be processed with the full privileges of the governance contracts.

The following steps outline the proposal lifecycle:

1. A user creates a new proposal by depositing an amount of INDY that equals the Proposal Deposit.
2. The proposal creator creates one or more shards, up to a maximum of Total Shards, by depositing ADA. All shards must be created before the Proposing Period ends for the proposal to pass.
3. The proposal enters the voting phase, where INDY stakers can vote (*yes*/*no*) using their staked INDY positions. INDY of the INDY stakers who vote remains locked until the *Voting Period* ends.
4. The Voting Period ends after more time has passed than the proposal's creation time, plus time defined by the Voting Period protocol parameter.
5. After the Voting Period has ended, the proposal can be closed by its creator.
6. If the proposal passes, its executable contents can be executed by users after a delay defined by the *Effective Delay* protocol parameter. The proposal must be executed prior to the time described by the *Expiration Period* protocol parameter; otherwise, the proposal will be considered expired and no longer executable.

Several actions can be taken against a proposal by users:

- **Create** – Creates a proposal conforming to one of the allowed Governance Proposal Types.
- **Create Shard** – The owner of the proposal is expected to – and can – create one or more shards, up to a maximum of *Total Shards.* For a proposal to be eligible to pass, the number of shards created must equal Total Shards. A shard is created by depositing ADA alongside a request to create one. Shards can only be created from the creation of the proposal up until the *Proposing Period* ends. Creating shards after the Proposing Period will cause the transaction to fail.
- **Merge Shards** – Users can merge two or more shards created after the proposal's Voting Period ends and before the proposal is closed. Upon merging, the owner is eligible to receive back the ADA that was deposited to create each merged shard after the proposal is closed.
- **Close** – The owner of the proposal can close the proposal after its Voting Period ends if the number of shards created is equal to Total Shards, and after all shards have been merged. If the number of shards created is less than Total Shards, then the proposal cannot be closed until after the proposal expires. After the owner closes their proposal, they receive back any ADA that was deposited to create each shard. If a proposal expires before the owner closes the proposal, then any user can close the proposal.
- **Execute** – If a proposal is closed and has passed, any user can execute it. Upon execution, the protocol runs the executable message embedded within the proposal to apply changes to the protocol.

A proposal has the following states:

- **Created** – After a proposal is created it is available for the owner to create shards.
- **Open** – When a proposal has one or more shards available then it becomes available for INDY holders to vote on. If a proposal has at least one shard but less than T*otal Shards*, the proposal is *Open.*
- **Active** – When a proposal has shards that equal *Total Shards*, all shards were created before the Proposing Period, and time has not exceeded its *Voting Period*, then the proposal is *Active.*
- **Ended** – When a proposal has exceeded its *Voting Period*, then the proposal is *Ended.*
- **Merged** – When all the proposal's shards have been merged, then the proposal is *Merged.*
- **Closed** – When a proposal is *Ended,* and after a user has made a submission for the proposal to close, then the proposal is *Closed.*

- **Passed** – When a proposal is *Closed* and the number of *yes* votes exceeds the quorum threshold, then the proposal is *Passed*.
- **Failed** – When a proposal is *Closed* and the number of *yes* votes does not exceed the quorum threshold, then the proposal is *Failed*.
- **Expired** – When a proposal has exceeded its *Execution Period* without being executed, if the created shards are fewer than Total Shards after the Voting Period, or if shards have been created after the Proposing Period, then the proposal is *Expired*.
- **Executed** – When a proposal is *Passed* and not *Expired*, then any user can execute the proposal. The proposal then becomes *Executed*.

2.9.11 Indigo DAO Treasury

The Indigo DAO owns and controls a DAO Treasury (the "Treasury"). Upon minting of INDY, a portion of INDY (the amount is defined at protocol launch) is sent to the Treasury.

To permanently identify the Indigo DAO on the Cardano blockchain, a NFT is minted as the official Indigo DAO identity token ("identity token") and held in the Treasury. The identity token is transferred to wherever the latest version of the Treasury lives. The protocol transfers the identity token and INDY in the Treasury upon future protocol upgrades.

The primary purpose of the Treasury is to support the ongoing development and maintenance of the Indigo Protocol. The assets are earmarked to be allocated to various essential activities (all subject to refinement and revision by vote of the DAO):

1. Maintenance: A substantial portion of the Treasury's funds are earmarked for maintaining and upgrading the protocol's infrastructure. On-going support of upgrades and maintenance of the existing Protocol is essential to ensure a robust, secure, and efficient Protocol for users.
2. Development: Continuing the technical development and innovation of the Indigo Protocol is paramount for its success and relevance. The Treasury has earmarked funds to pay developers and other technical service providers in order to continuously fuel innovation and help ensure that the Protocol remains at the forefront of technological advancement.
3. Foundation Operations: The Indigo Foundation is a Cayman Islands foundation company with limited liability. It was created to be the voice of the DAO and oversee the Protocol's governance and strategic direction, as well as implement votes of the DAO (such as to hire developers to provide maintenance or development services, or vendors to provide audits, accounting, legal or other services). To fulfill its mission, the Foundation requires substantial resources on an annual basis to cover normal business expenses. A portion of the Treasury is earmarked to provide the necessary financial support for these operations.

There is a limit of INDY and ADA that can be withdrawn from the Treasury. This limit can be adjusted by a DAO vote to adjust protocol parameters.

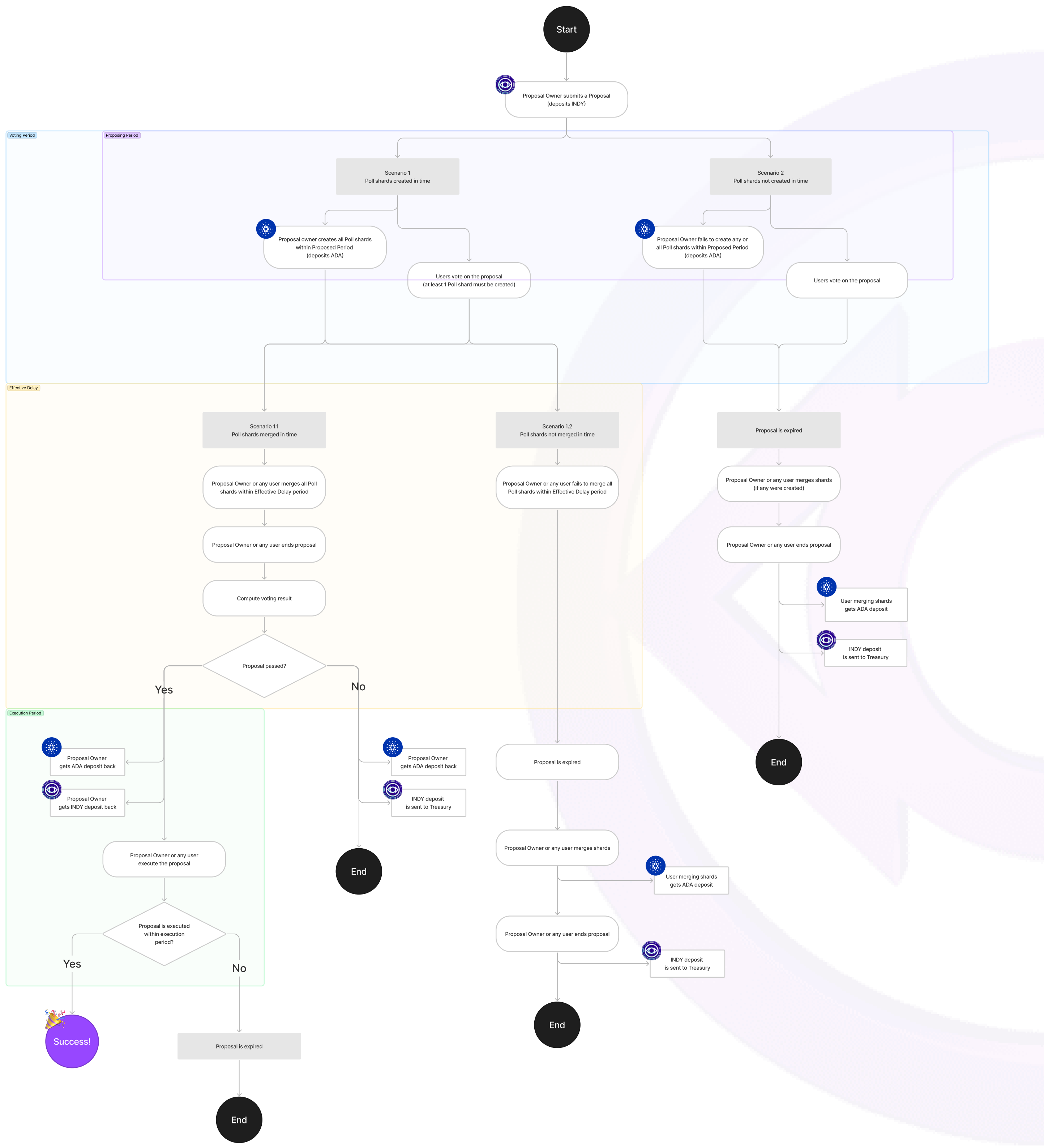
2.9.12 Protocol Upgrade

Indigo is designed to be continually and incrementally upgraded. Instead of releasing distinct protocols that users may interact with individually, the Indigo Protocol exists as a singular protocol whose underlying validators may periodically be updated. From a user's perspective, the interaction is seamless, since they will only interact with one protocol, regardless of the version of Indigo Protocol that is live on the Cardano blockchain.

A single protocol has been launched, and new features will be added to Indigo via approval from Members. Protocol upgrades are driven by the governance process. To suggest new features, a Text proposal and development request must first be approved and authorized by the DAO. A development firm such as Indigo Laboratories will then begin work on building software to implement the new features.

When software is ready for deployment, a request to upgrade is submitted to Indigo. Members can inspect the new code requested to be deployed and either approve or reject the proposal. Upon approval, the developing entity of the software can deploy the code to Cardano, and Indigo will be automatically upgraded to a new version. However, the code must match the code approved by Members, otherwise Indigo will not recognize the new features as authentic, and no upgrade will take place.

After deployment and approval, individual user positions can be migrated from the old version to the new one. Some features may not be available until the user has migrated their positions. For example, if a user owns a CDP, they will be unable to add collateral to their CDP until they migrate their CDP to the new version of



Start
Proposal Owner submits a Proposal (deposits INDY)
Voting Period
Proposing Period
Scenario 1 Poll shards created in time
Scenario 2 Poll shards not created in time
Proposal owner creates all Poll shards within Proposed Period (deposits ADA)
Users vote on the proposal (at least 1 Poll shard must be created)
Proposal Owner fails to create any or all Poll shards within Proposed Period (deposits ADA)
Users vote on the proposal
Effective Delay
Scenario 1.1 Poll shards merged in time
Scenario 1.2 Poll shards not merged in time
Proposal is expired
Proposal Owner or any user merges all Poll shards within Effective Delay period
Proposal Owner or any user fails to merge all Poll shards within Effective Delay period
Proposal Owner or any user merges shards (if any were created)
Proposal Owner or any user ends proposal
Proposal Owner or any user ends proposal
Compute voting result
User merging shards gets ADA deposit
INDY deposit is sent to Treasury
Proposal passed?
Yes
No
Execution Period
Proposal Owner gets ADA deposit back
Proposal Owner gets INDY deposit back
Proposal Owner gets ADA deposit back
INDY deposit is sent to Treasury
Proposal is expired
End
Proposal Owner or any user execute the proposal
End
Proposal Owner or any user merges shards
User merging shards gets ADA deposit
Proposal is executed within execution period?
Proposal Owner or any user ends proposal
INDY deposit is sent to Treasury
Yes
No
End
Success!
Proposal is expired
End

Figure 16: Illustration of the proposal lifecycle

Indigo. To migrate a CDP, a user will have to pay a small transaction fee in the form of ADA and submit the migration request via the Indigo Web App. If a user chooses not to migrate a CDP, they will not be able to deposit more collateral or mint more iAsset; their CDP may become at risk of liquidation. Another user may opt to migrate a CDP subject to liquidation to perform the liquidation and confiscate the underlying collateral, with the original CDP owner losing their collateral.

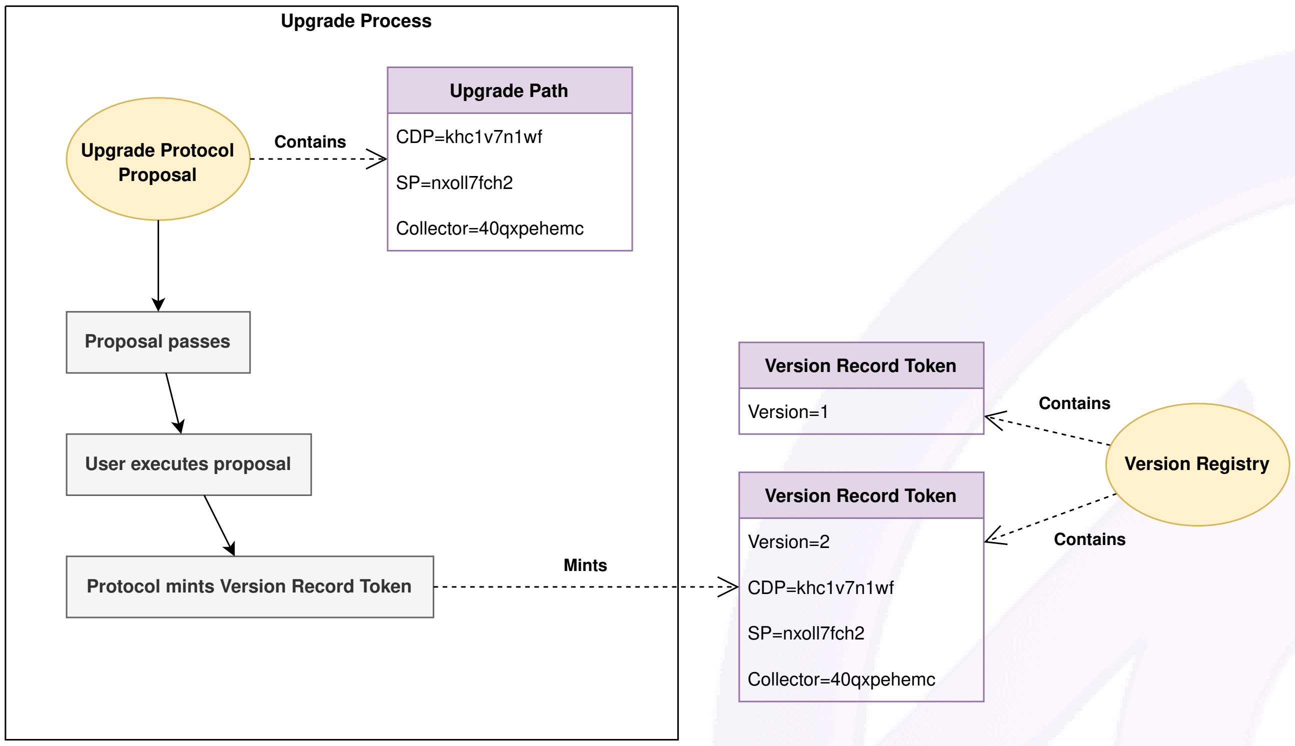


Figure 17: Illustration of an Upgrade Protocol proposal upgrading the CDP, SP, and Collector contracts

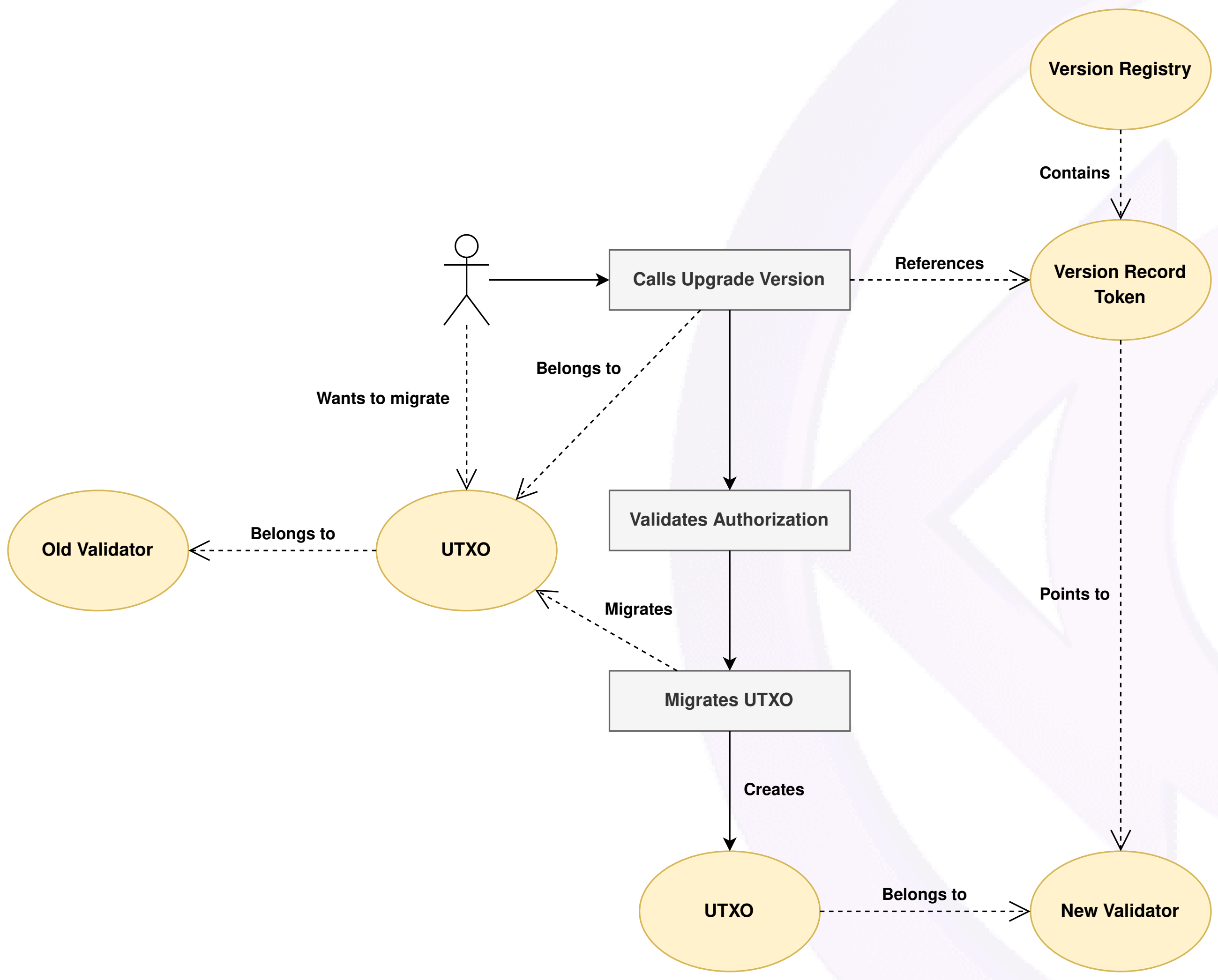


Figure 18: Illustration of a UTXO migrating from an old validator to a new validator

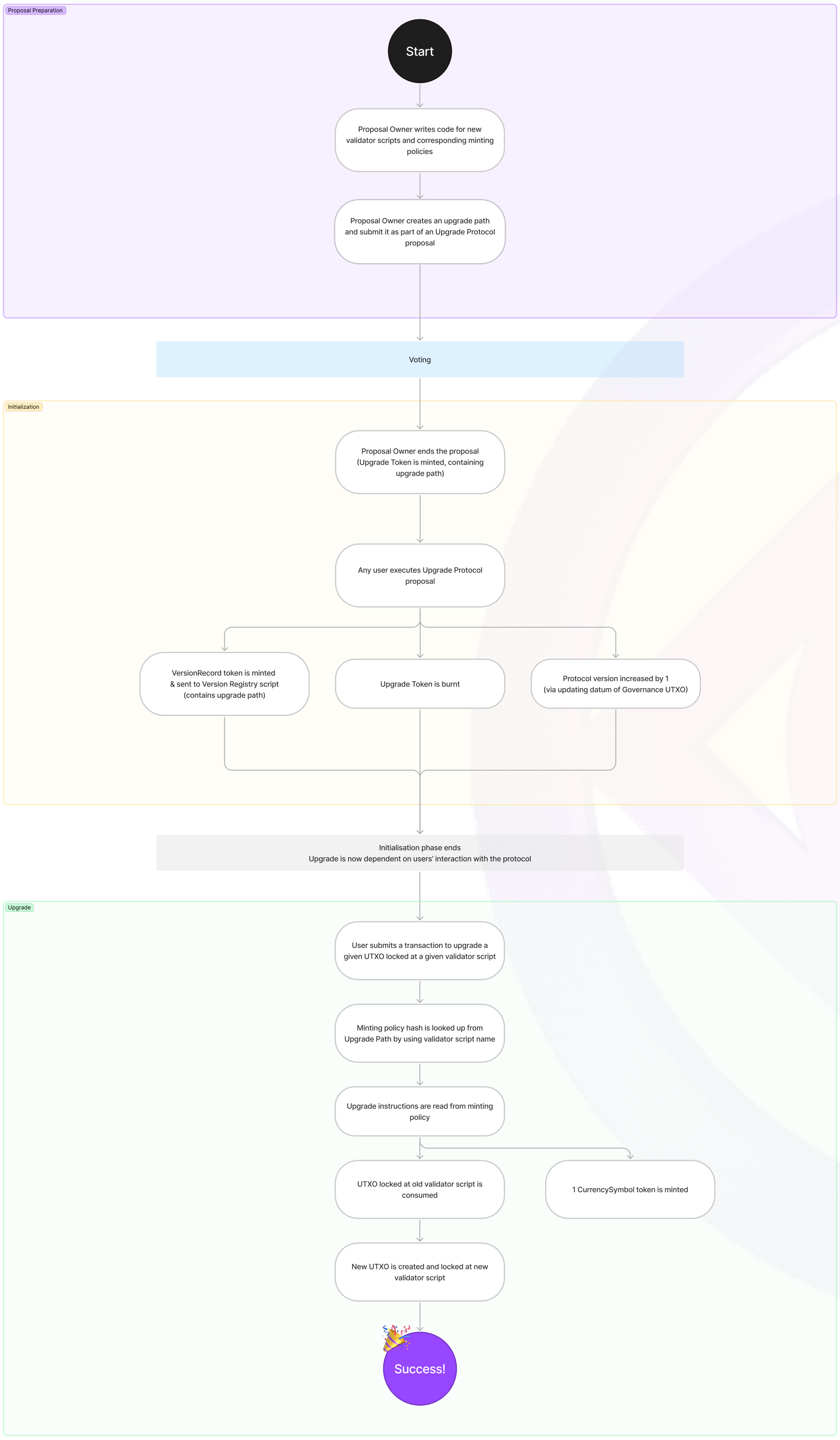

Proposal Preparation
Start
Proposal Owner writes code for new validator scripts and corresponding minting policies
Proposal Owner creates an upgrade path and submit it as part of an Upgrade Protocol proposal
Voting
Initialization
Proposal Owner ends the proposal (Upgrade Token is minted, containing upgrade path)
Any user executes Upgrade Protocol proposal
VersionRecord token is minted & sent to Version Registry script (contains upgrade path)
Upgrade Token is burnt
Protocol version increased by 1 (via updating datum of Governance UTXO)
Initialisation phase ends
Upgrade is now dependent on users' interaction with the protocol
Upgrade
User submits a transaction to upgrade a given UTXO locked at a given validator script
Minting policy hash is looked up from Upgrade Path by using validator script name
Upgrade instructions are read from minting policy
UTXO locked at old validator script is consumed
1 CurrencySymbol token is minted
New UTXO is created and locked at new validator script
Success!

Figure 19: The process to upgrade the protocol

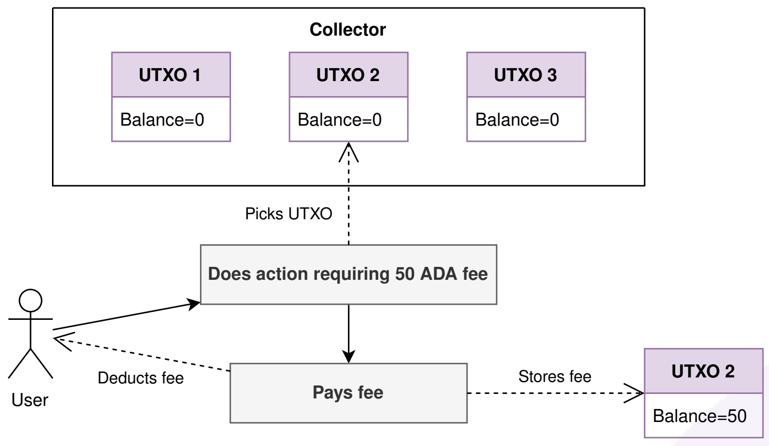


Figure 20: A user paying a fee to the Collector

2.10 Protocol Profit Sharing

As users create and close CDPs, and as CDPs are liquidated, a fee is collected. Members are rewarded by receiving a share of the collected fees. The Debt minting fee and Liquidation Fee are parameters assigned to iAssets that are modifiable by DAO vote.

Debt Minting Fee:

$$a = b * c$$

- a is the ADA fee charged for minting debt
- b is the current Debt minting fee constant
- c is the amount of iAsset minted

Liquidation Fee:

$$a = b * c$$

- a is the Liquidation fee upon liquidation
- b is the current Liquidation fee constant
- c is the amount of iAsset liquidated

When a fee is collected, it is sent to the Collector smart contract. The Collector's purpose is to collect protocol fees and distribute them to INDY stakers. Users who stake their INDY are eligible to a share of all collected protocol fees, proportional to their share of total INDY staked.

The Collector maintains a collection of UTXOs that can be used to store ADA. When a protocol fee is collected, such as during withdrawal of a liquidation reward, the user selects a UTXO from the Collector to send the fee to. The amount of ADA required to deposit to create a Collector UTXO is x, as calculated and described in the Minimum ADA to Create UTXO section. A Collector UTXO can be created by any user who deposits x ADA. Once deposited, a new UTXO is added to the Collector and the ADA cannot be withdrawn.

Users can request to gather fees from Collector UTXOs and collectively send them to the Staking Manager who is responsible for allowing INDY stakers to withdraw their share of owed fees, and will only accept deposits of fees if there are one or more INDY stakers. If no INDY is staked, then user requests to transfer fees from the Collector to the Staking Manager will fail.

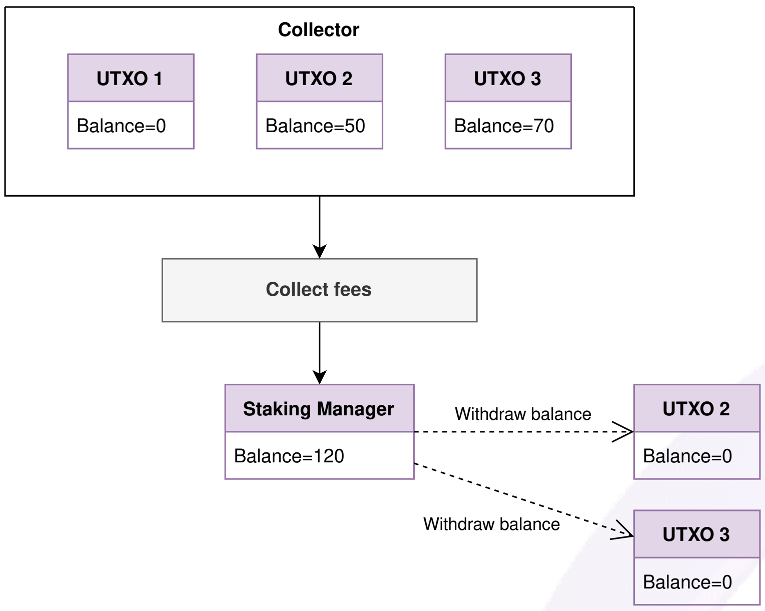


Figure 21: Transferring collected fees to the Staking Manager

The Staking Manager keeps track of the number of INDY that are staked as well as a snapshot value. The snapshot value is a running total (with a precision of six decimals) of reward deposits updated each time ADA is transferred from the Collector to the Staking Manager, and can be calculated using:

$$a = b + \frac{c}{d}$$

Where:

- a is the new snapshot value to be stored by the Staking Manager, truncated to six decimals
- b is the current snapshot value stored by the Staking Manager
- c is the amount of ADA deposited into the Staking Manager from the Collector
- d is the total amount of INDY staked in the Staking Manager

The snapshot value is initially set to zero. When a user stakes INDY, the current snapshot value is stored in the INDY staker's position, and the total amount of INDY staked is updated in the Staking Manager. When an INDY staker updates or closes their position, all rewards are withdrawn. INDY staker rewards can be calculated using:

$$a = d\,(b - c)$$

Where:

- a is the amount of ADA reward the user is owed
- b is the current snapshot value stored by the Staking Manager
- c is the snapshot value when the user staked their INDY
- d is the amount of INDY the user has staked

3 Smart Contract Design

In Cardano's eUTXO model[13], each transaction has inputs and outputs. An input is a UTXO that is an output of another transaction. Users interact with the protocol by performing actions and submitting transactions containing those actions to Protocol Endpoints. Submitted transactions are validated by the protocol's smart contracts (also known as validators). If a transaction is successfully validated (i.e., permitted), then an action is put into effect by the transaction's execution.

[13]Cardano utilizes the eUTXO model to perform arbitrary logic permitted by smart contracts.

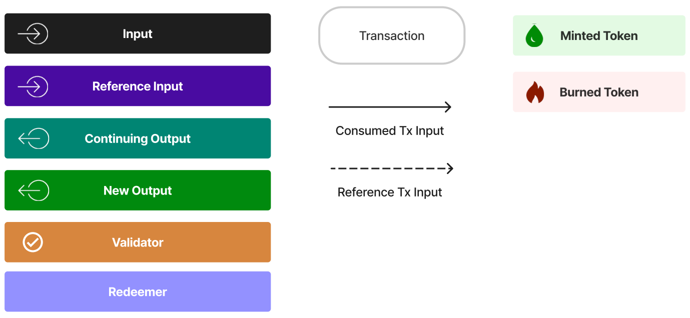


Figure 22: Legend for Protocol Endpoint transaction examples

Protocol Endpoints allow users to interact with the protocol by performing a specific action such as opening a CDP, submitting a proposal, depositing iAssets in a SP, etc. A Protocol Endpoint can take input in the form of UTXOs. Input is provided either by consuming or referencing. To consume a UTXO is to spend the UTXO in whole within the transaction. By consuming the UTXO it allows change to the state of that UTXO, such as updating the balance. To reference a UTXO is to read the UTXO without change. Only one user can consume a single UTXO at a time, whereas many users can simultaneously reference UTXOs.

Protocol Endpoints may perform actions in the form of minting or burning. Minting a token creates a new token and allows it to be used as input. Upon minting, the token may be stored in a UTXO containing datum that can be read for additional information. Burning a token destroys an existing token, making it no longer usable as input.

Outputs are UTXOs that are created as an effect of a transaction. For example, a Protocol Endpoint may create an output to represent a user position or a pool of tokens. After an output is created it can be used as an input.

Following are details for each Indigo smart contract, their tokens issued, parameter inputs, and outputs.

For the described smart contract parameters, token types are in the form of $Value.AssetClass$[14]. The smart contracts look for the UTXO with the token type and may read the datum of that UTXO for additional information.

3.1 CDP

The CDP contracts are used to store the collateral used to mint iAssets. There are two contracts for managing CDPs: CDPCreator and CDP. The CDPCreator validates the creation of a user's CDP UTXO. The CDP contract is used to manage a user's individual position by validating actions such as storing collateral, minting iAssets, and performing SPL.

Table 6: CDP native tokens

Name	Description	Minting Policy
CDPCreatorNFT	Identifies the authentic CDPCreator output Validators ensure that this NFT always stays at the CDPCreator output	The protocol mints more than 1 token at initialization
CDPToken	Identifies an authentic CDP output	The transaction must spend CDPCreatorNFT or consume a CDPToken
iAssetToken	Identifies an authentic iAsset output, where datum is stored defining iAsset information including the OracleAssetNFT used to reference the latest price Validators ensure that this token always stays at an iAsset output	The transaction must consume GovNFT

[14] An asset class is identified by currency symbol and token name.

Table 6: CDP native tokens

Name	Description	Minting Policy
iAssets (iBTC, iETH, etc.)	Synthetic version of BTC, ETH, etc.	The transaction must consume a CDPToken

Table 7: CDP reference inputs

Type	Description	Datum
OracleAssetNFT	The NFT managed by an Oracle provider that's used to record price information for an iAsset	*price*: The price with six decimals of precision *expiration*: The timestamp in which the oracle price expires

3.1.1 CDPCreator Parameters

- `cdpCreatorNFT :: CDPCreatorNFT`. NFT for identifying authentic CDPCreator output.
- `cdpAssetCs :: CurrencySymbol`. Currency symbol for the minting policy of iAssets.
- `cdpAuthTk :: CDPToken`. Token for identifying authentic CDP output.
- `iAssetAuthTk :: iAssetToken`. Token for identifying authentic iAsset output including datum with the iAsset name, LR, and OracleAssetNFT reference to find the latest price for the asset.
- `versionRecordToken :: VersionRecordToken`. Token for identifying the version record for a protocol upgrade.
- `cdpScriptHash :: ValidatorHash`. Hash of CDP script, used for verifying the output of a CDP.
- `minCollectoralInLovelace :: Integer`. The minimum allowed lovelaces in a CDP.
- `biasTime :: PosixTime`. The range of time that a transaction is valid onchain.

3.1.2 CDP Parameters

- `cdpAuthToken :: CDPToken`. Token for identifying authentic CDP output.
- `cdpAssetSymbol :: CurrencySymbol`. Currency symbol for the minting policy of iAssets.
- `iAssetAuthToken :: iAssetToken`. Token for identifying authentic iAsset output.
- `stabilityPoolAuthToken :: StabilityPoolToken`. Token identifying authentic SP output.
- `versionRecordToken :: VersionRecordToken`. Token for identifying the version record for a protocol upgrade.
- `upgradeToken :: UpgradeToken`. Token for identifying proposal Upgrade tokens to update iAsset output.
- `collectorValHash :: ValidatorHash`. The validator hash for the Collector contract.
- `govNFT :: GovNFT`. NFT for identifying authentic governance parameters.
- `spValHash :: ValidatorHash`. The validator hash for the SP contract.
- `minCollateralInLovelace :: Integer`. The minimum allowed lovelaces in a CDP.
- `partialRedemptionExtraFeeLovelace :: Integer`. The fee for processing a partial redemption.
- `biasTime :: PosixTime`. The range of time that a transaction is valid onchain.
- `treasuryValHash :: ValidatorHash`. The validator hash for the Treasury contract.

Table 8: CDP outputs

Type	**Description**	**Datum**	**Values**
CDPCreator	Many CDPCreator outputs exist for the protocol To create a CDP output, this output must be consumed		*CDPCreatorNFT*: 1
CDP	Each CDP output represents an individual position	*owner*: The public key hash that owns this CDP *asset*: The type of iAsset associated with this CDP *minted_amount*: Amount of iAsset minted from this position *accumulated_fees*: Total amount of fees collected from interest and Liquidation Processing Fee	*CDPToken*: 1 *ADA*: collateral locked in this position

Table 8: CDP outputs

Type	Description	Datum	Values
iAsset	Each iAsset output represents an iAsset	*name*: the name of iAsset *price_info*: Either the final price for the delisted asset or the OracleAssetNFT used to reference the price feed *redemption_ratio*: The ratio at which a CDP can be redeemed. *maintenance_ratio*: The minimum that the user can set their CDP collateral ratio to. *liquidation_ratio*: If a CDP goes below this ratio, it is susceptible to liquidations. *debt_minting_fee_percentage*: The percentage of fee to take in lovelaces when minting iAsset *liquidation_processing_fee_percentage*: The percentage of fee to take in lovelaces when liquidating an iAsset *redemption_reimbursement_percentage*: This fee is charged to the redeemer's redemption value. This percentage of ADA is returned to the redeemed CDPs collateral. *redemption_indy_staker_percentage*: This fee is charged to the redeemer's redemption value. This percentage of ADA is sent to INDY stakers. *base_rates*: Interest rates in an ordered list, from newest to oldest. *first_iasset*: Marks if this asset is the first iAsset. *next_iasset*: An optional field if there is an iAsset after this one in alphabetical order.	*iAssetToken*: 1

3.1.3 CDP Endpoints

CDP: Open Creates a CDP associated with an iAsset type

Type	Amount	Description
Redeemer	N.A.	CreateCDP, takes as parameters a public key hash corresponding to a user's wallet, amount of iAssets to mint, and ADA collateral to deposit
Redeemer	N.A.	Collect, collects value from the Collector UTxO to distribute the debt minting fee.

Type	Amount	Description
Consume	1	CDPCreator UTXO
Consume	1	Collector UTXO
Consume	1+	ADA to be used as collateral
Reference	1	iAsset UTXO that identifies the iAsset to mint
Reference	1	UTXO containing the OracleAssetNFT with a datum describing the iAsset price
Mint	∞	The minted iAsset tokens (dependent on the ADA deposited, iAsset LR determined from the iAsset UTXO, and iAsset price)
Mint	1	CDPToken that identifies a user's position
Output	1	CDPCreator UTXO
Output	1	CDP UTXO that represents a user's CDP
Output	1	Collector UTXO that collects the debt minting fee
Output	1	The UTXO sent to the user's wallet containing the minted iAsset

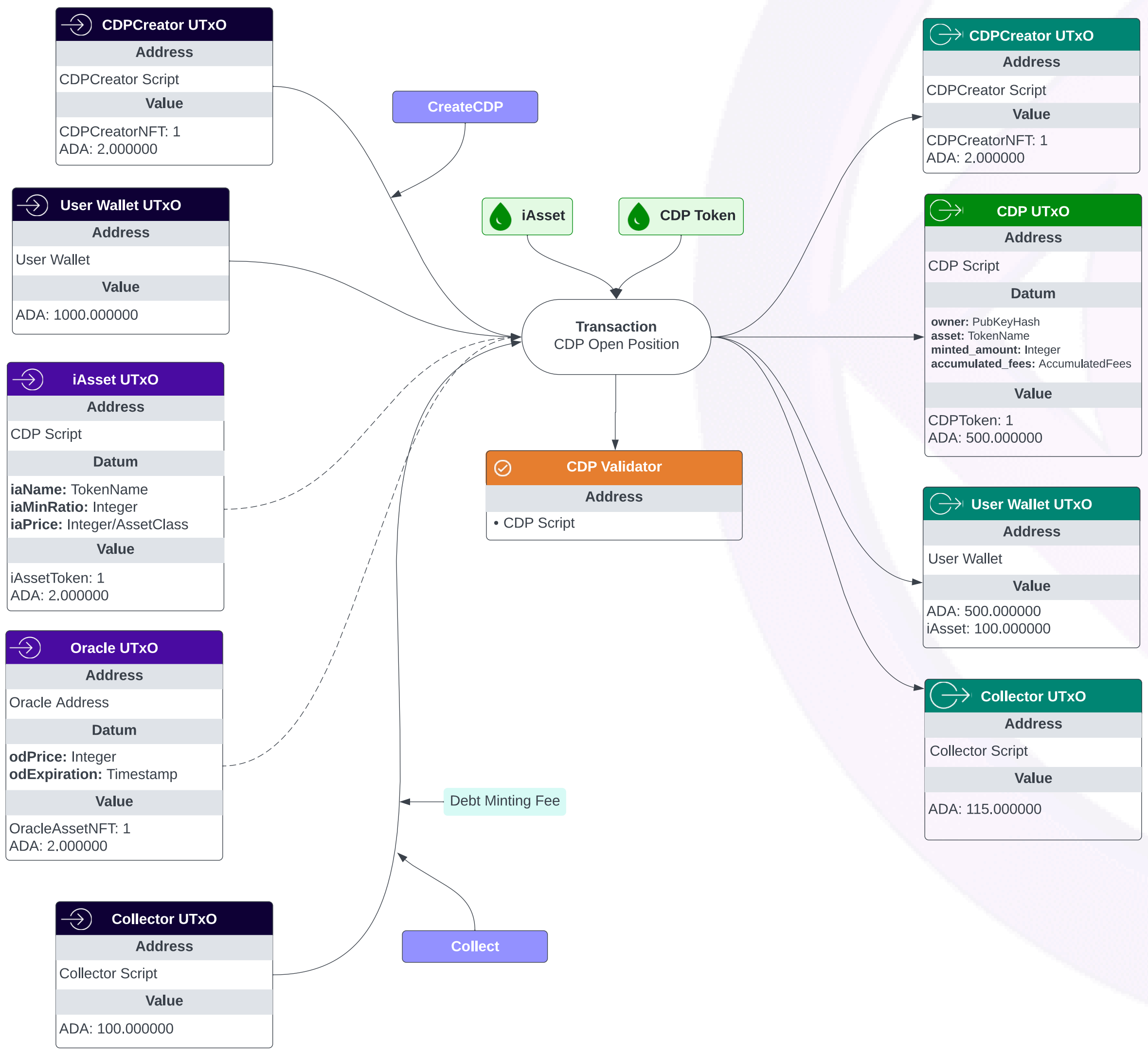


Figure 23: Example of creating a CDP with 500 ADA and minting 100 iAsset

CDP: Deposit Collateral Deposit ADA collateral into an existing CDP

Type	Amount	Description
Redeemer	N.A.	AdjustCDP, CDP Input
Consume	1	CDP UTXO that represents the user's current position
Consume	1+	UTXOs containing ADA from the user's wallet to be used as collateral
Reference	1	iAsset UTXO that serves to identify the iAsset that the CDP is for
Output	1	CDP UTXO that represents the user's adjusted CDP
Output	1	New UTXO to the user wallet returning change (if any)

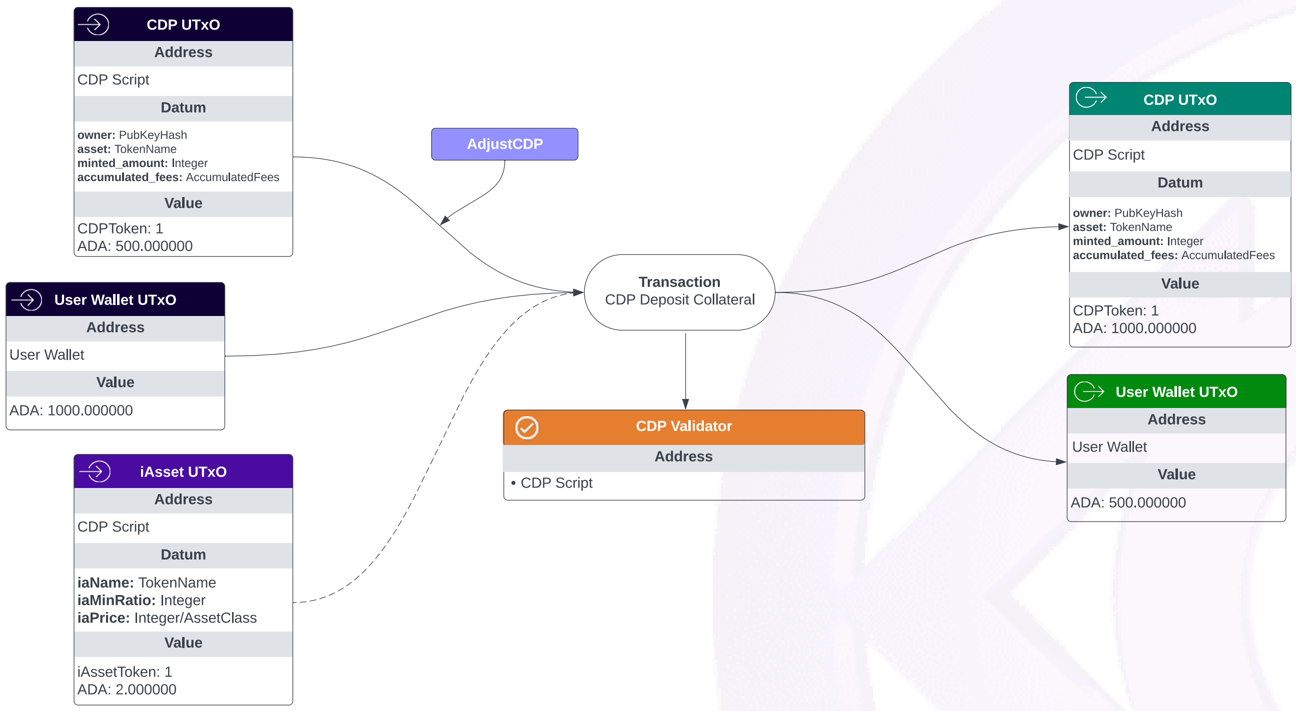


Figure 24: Example of depositing an additional 500 ADA into an existing CDP

CDP: Withdraw Collateral Withdraw ADA collateral from an existing CDP

Type	Amount	Description
Redeemer	N.A.	AdjustCDP, CDP Input
Redeemer	N.A.	Collect, Collector Input
Consume	1	CDP UTXO that represents the user's current position
Consume	1	Collector UTXO that may already contain fees previously collected
Reference	1	iAsset UTXO that serves to identify the iAsset should be minted
Reference	1	UTXO containing the OracleAssetNFT with a datum describing the iAsset price
Output	1	CDP UTXO that represents the user's adjusted position
Output	1	Collector UTXO that contains a portion of the withdrawn collateral (taken as the collector fee)
Output	1	A new UTXO to the user wallet containing the withdrawn collateral

CDP: Close Closes an existing CDP

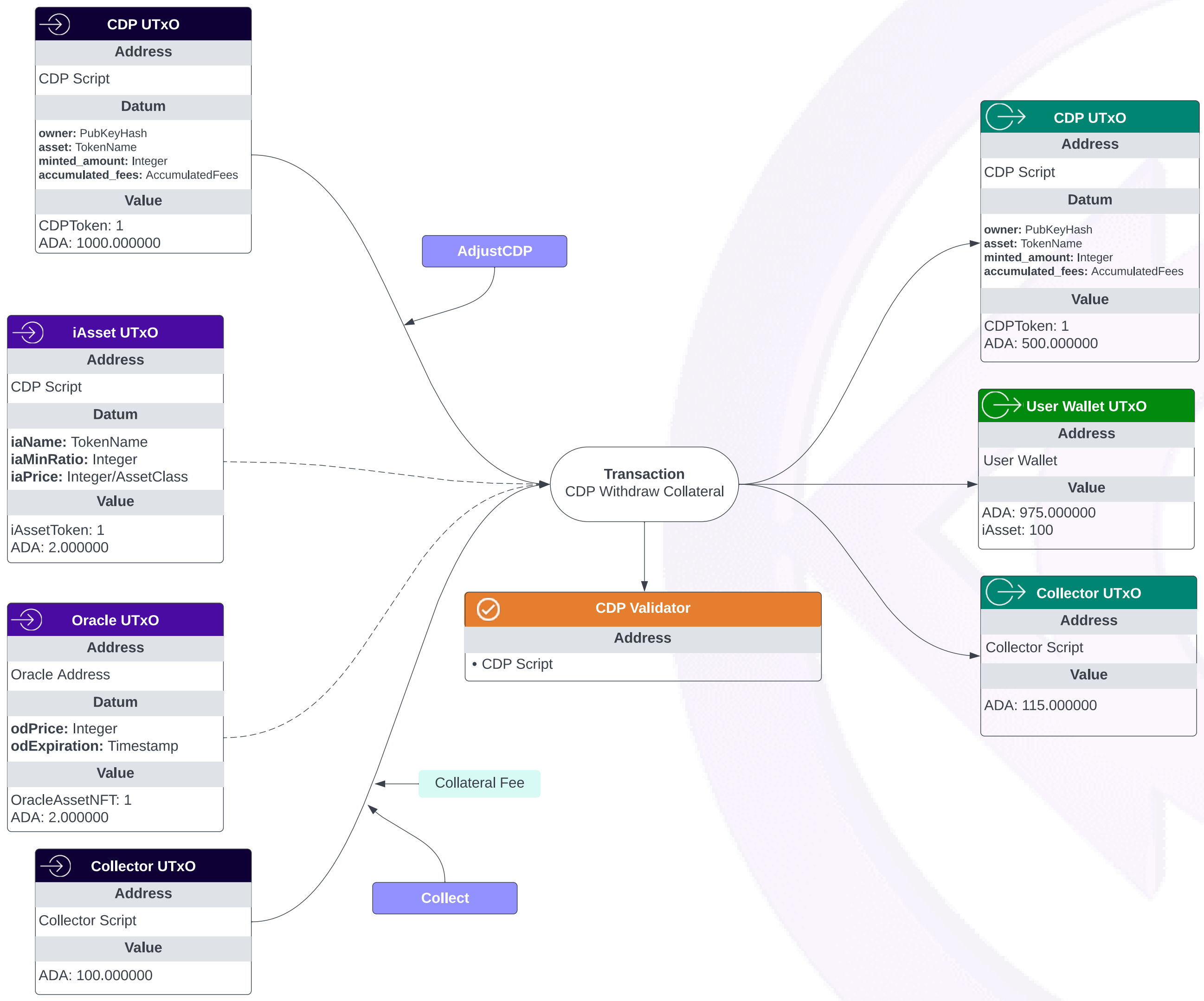


Figure 25: Example of withdrawing 500 ADA from a CDP and paying a 10 ADA fee

Type	Amount	Description
Redeemer	N.A.	CloseCDP
Redeemer	N.A.	Collect
Consume	1	CDP UTXO that represents the user's current position
Consume	1	Collector UTXO that may already contain fees previously collected
Consume	1+	UTXOs from the user's wallet containing iAsset tokens of the same type as the CDP
Reference	1	iAsset UTXO that serves to identify the iAsset the CDP is for
Reference	1	UTXO containing the OracleAssetNFT with a datum describing the iAsset price
Burn	∞	iAssets that were sent by the user
Burn	1	CDPtoken
Output	1	Collector UTXO that contains a portion of the CDP collateral (taken as the collateral fee)
Output	1	A new UTXO to the user wallet containing the total collateral (minus the fee)

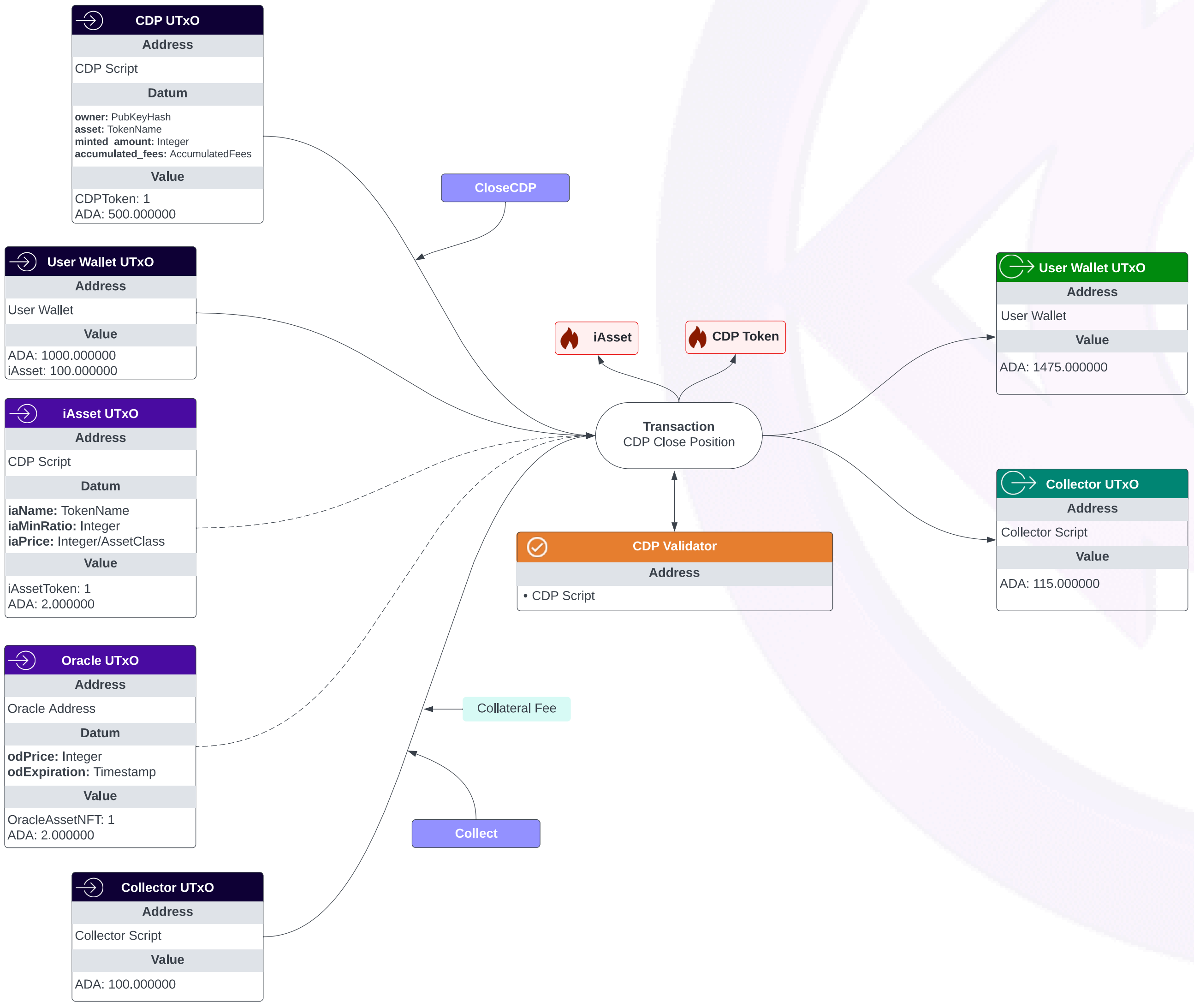


Figure 26: Example of closing a CDP and paying a 10 ADA fee

CDP: Mint iAsset Mints iAsset using an existing CDP

Type	Amount	Description
Redeemer	N.A.	AdjustCDP, CDP Input
Redeemer	N.A.	Collect, Collector Input
Consume	1	CDP UTXO that represents the user's current position
Consume	1	Collector UTXO that may already contain fees previously collected
Reference	1	iAsset UTXO that serves to identify the iAsset the CDP is for
Reference	1	UTXO containing the OracleAssetNFT with a datum describing the iAsset price
Mint	∞	iAsset tokens the user selected to mint
Output	1	CDP UTXO that represents the user's adjusted CDP
Output	1	Collector UTXO that contains the debt minting fee
Output	1	A new UTXO to the user wallet containing the newly minted iAsset tokens

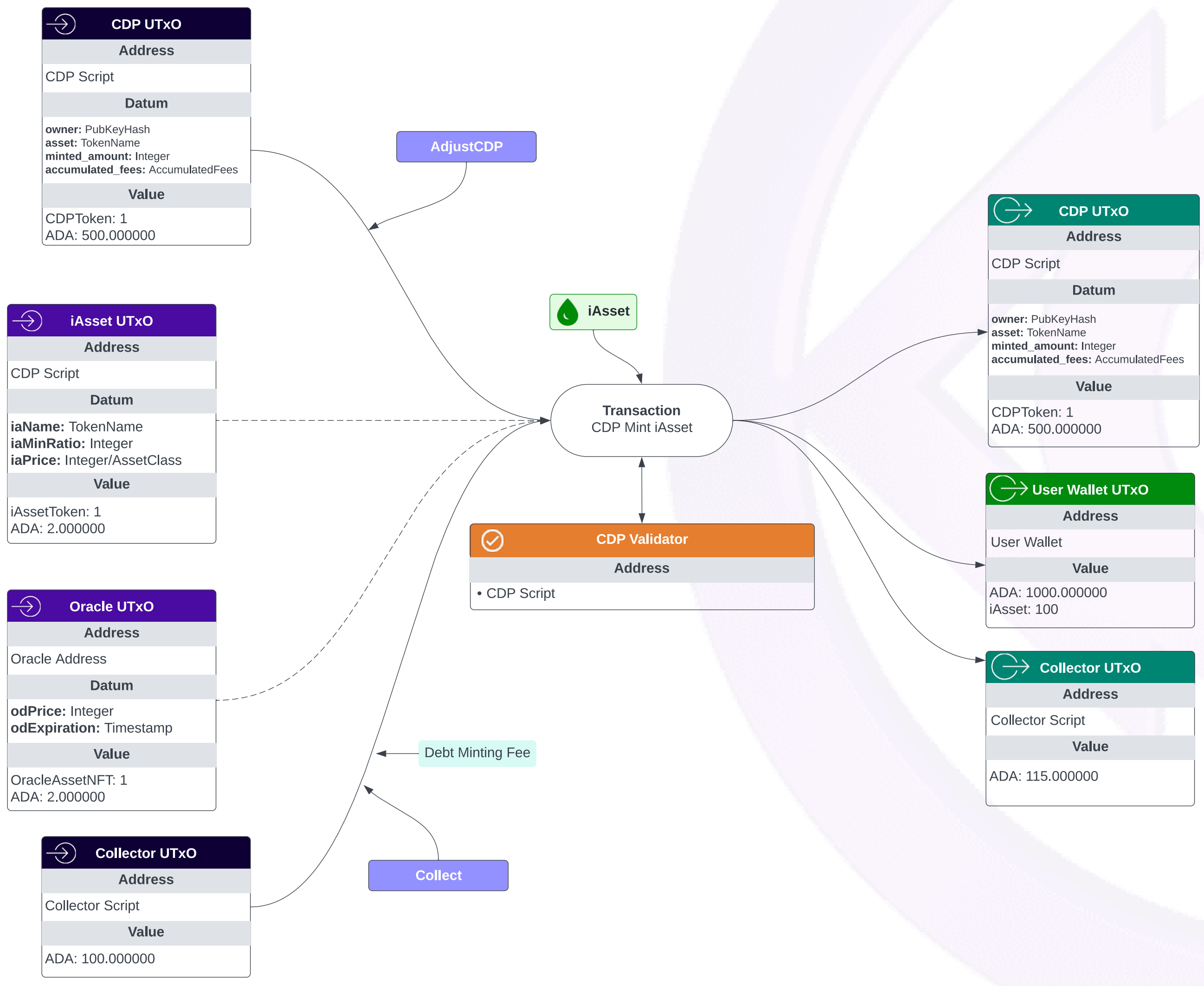


Figure 27: Example of using a CDP to mint 100 iAsset

CDP: Burn iAsset Burns iAsset using an existing CDP

Type	Amount	Description
Redeemer	N.A.	AdjustCDP

Type	Amount	Description
Consume	1	CDP UTXO that represents the user's current position
Consume	1+	UTXOs from the user's wallet containing the iAsset tokens to be burned
Reference	1	iAsset UTXO that serves to identify the iAsset that the CDP is for
Burn	∞	The iAsset tokens the user requested to burn
Output	1	CDP UTXO that represents the user's adjusted position
Output	1	New UTXO to the user wallet returning change (if any)

CDP: Freeze Makes an existing CDP no longer interactable by its creator if it is insolvent

Type	Amount	Description
Redeemer	N.A.	FreezeCDP
Consume	1	CDP UTXO that represents the user's current position
Reference	1	iAsset UTXO that serves to identify the iAsset the CDP is for
Reference	1	UTXO containing the OracleAssetNFT with a datum describing the iAsset price
Output	1	CDP UTXO that represents the frozen CDP
Output	1	New UTXO to the user wallet returning change (if any)

CDP: Liquidate Withdraws ADA collateral from a CDP and transfers it to a SP if the CDP is frozen

Type	Amount	Description
Redeemer	N.A.	Liquidate, CDP Input
Redeemer	N.A.	LiquidateCDP, Stability Pool Input
Redeemer	N.A.	CollectADA, Treasury Input
Redeemer	N.A.	Collect, Collector Input
Reference	1	iAsset UTXO that serves to identify the iAsset the CDP is for
Consume	1	CDP UTXO that represents the frozen CDP to liquidate
Consume	1	SP UTXO that contains iAsset tokens to repay the debt
Consume	1	Treasury UTXO that contains existing lovelaces
Consume	1	Collector UTXO that contains existing lovelaces
Burn	0/1	If all debt is repaid, then the CDPToken of the frozen CDP is burned
Output	1	SP UTXO that with the added collateral from the frozen CDP
Output	1	Treasury UTXO that has the included ADA for CDP interest
Output	1	Collector UTXO that has the included ADA for INDY Stakers Liquidation Processing Fee and Collateral Fee

CDP: Merge Closes one or more CDPs and transfers all CDP state into a single CDP

Type	Amount	Description
Redeemer	N.A.	MergeCDPs
Redeemer	N.A.	MergeAuxiliary, takes a CDP UTXO as a parameter which identifies the main UTXO to keep and have others UTXOs merged into

Type	Amount	Description
Consume	2+	CDP UTXOs of the frozen CDPs to merge
Output	1	CDP UTXO representing all the frozen CDPs combined

CDP: Redemption Allows a user to spend some iAsset to redeem for a CDPs collateral by repaying their debt

Type	Amount	Description
Redeemer	N.A.	RedeemCDP, CDP Input
Redeemer	N.A.	Collect, Collector Input
Reference	1	iAsset UTXO that serves to identify the iAsset the CDP is for
Consume	1	CDP UTXO to redeem
Consume	1	Collector UTXO that contains existing lovelaces
Consume	1+	UTXOs from the user's wallet containing the iAsset tokens to be redeemed against
Output	1	CDP UTXO that has been redeemed against
Output	1	UTXO to the user wallet returning the redeemed ADA
Output	1	Collector UTXO that has the included ADA for Redemption INDY Staker fee

3.2 Stability Pool

The SP contract is used as a pool of iAssets to be used for liquidation. It is important to understand how the Snapshot works to understand how the liquidations and account withdrawals work. The Stability Pool utilizes a pattern called Liquidity State Transitions to allow a user to create a request transaction that can then be used in an execution transaction to process the appropriate action.

Table 19: Stability Pool native tokens

Name	Description	Minting Policy
StabilityPoolToken	Identify the authentic StabilityPool output	The transaction must spend GovNFT
AccountToken	Identify an authentic StabilityPoolAccount output	The transaction must spend StabilityPoolToken
EpochToScaleToSumToken	Identify an authentic SnapshotEpochToScaleToSum output	The transaction must spend StabilityPoolToken
RequestToken	Identify an authentic StabilityPoolRequest output	The Request Policy rules must be satisfied, see Request Policy smart contract rules

3.2.1 Stability Pool Parameters

- `assetSymbol :: CurrencySymbol`. The minting policy for iAssets.
- `stabilityPoolToken :: StabilityPoolToken`. The token identifying an authentic SP output.
- `accountToken :: AccountToken`. The token identifying an authentic SP Account output.
- `snapshotEpoch2Scale2SumToken :: EpochToScaleToSumToken`. The token identifying an authentic SP Account output.
- `requestToken :: RequestToken`. The token identifying an authentic Stability Pool Request output.

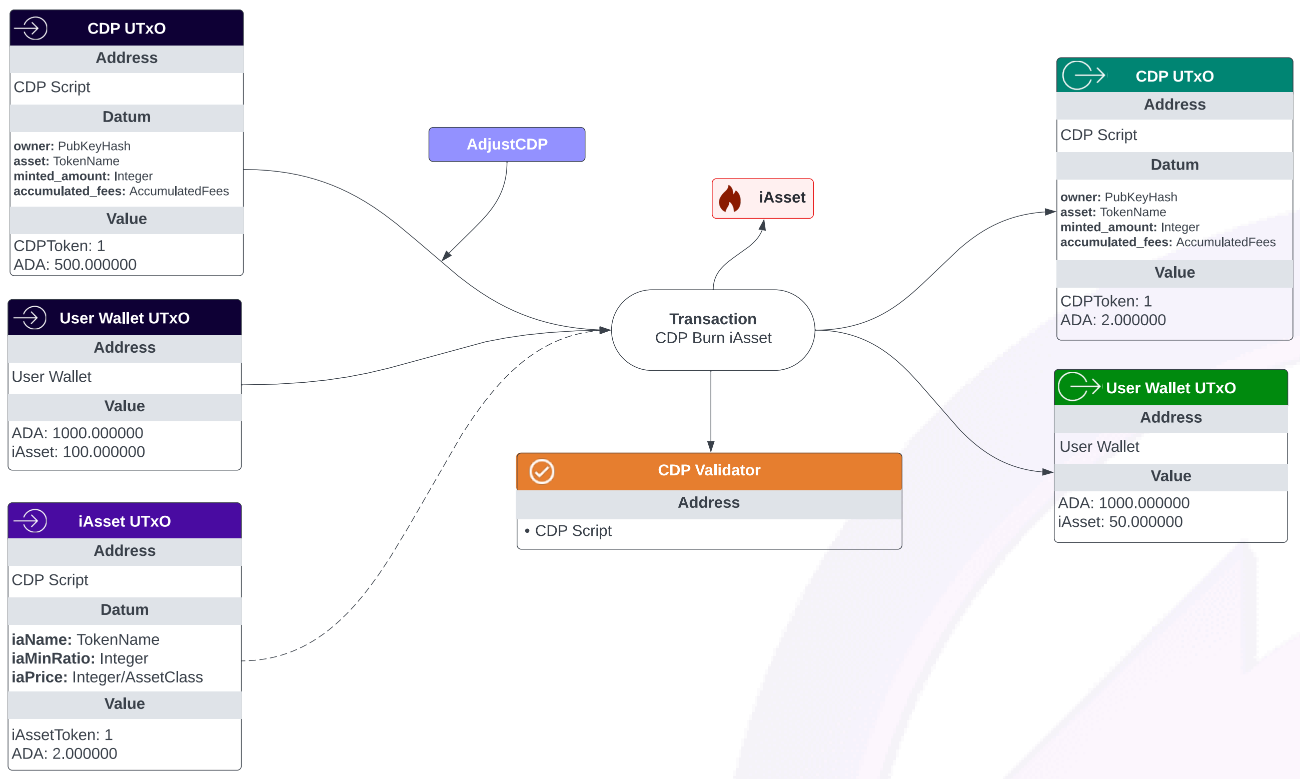


Figure 28: Example of using a CDP to burn 50 iAsset

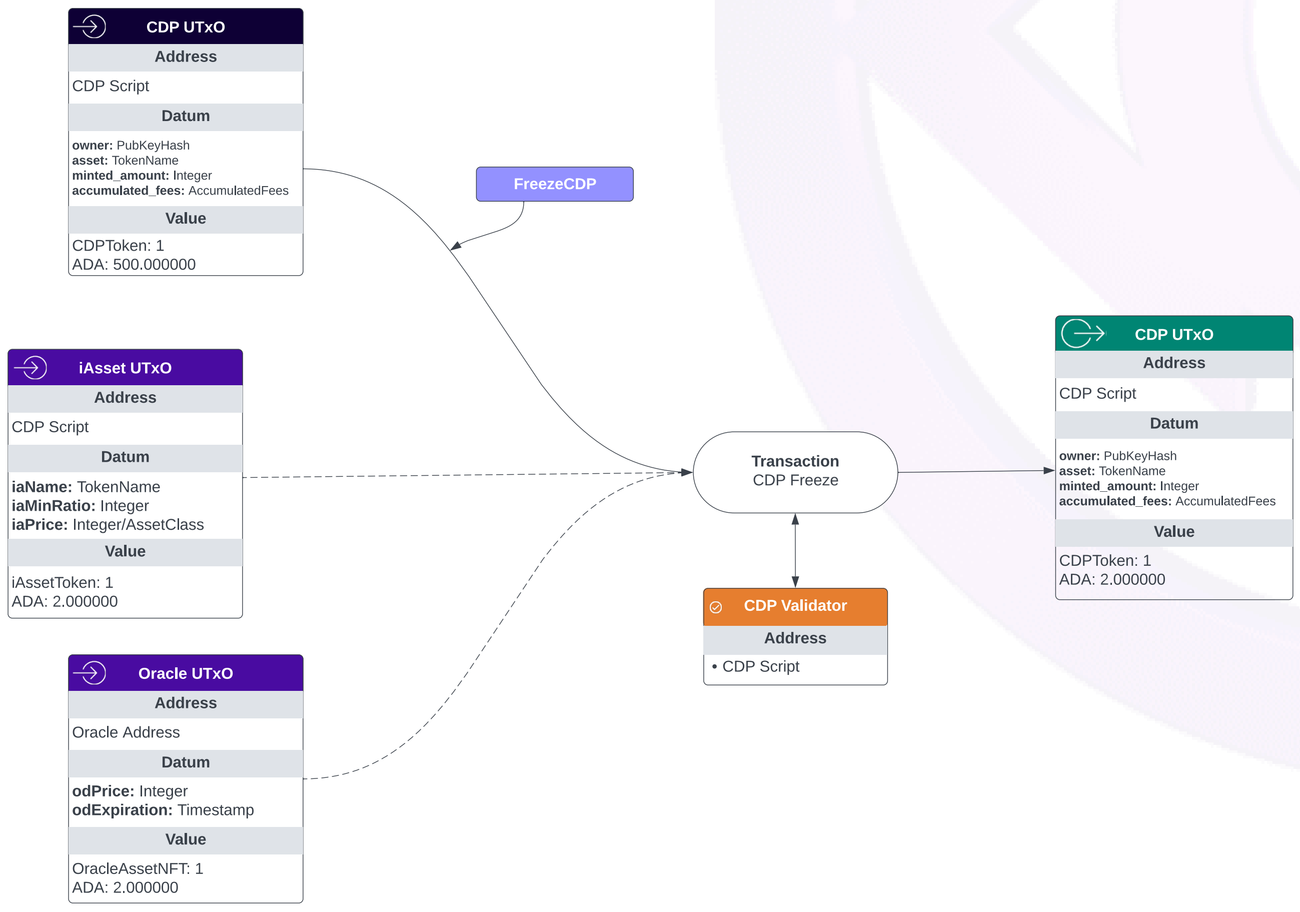


Figure 29: Example of freezing a CDP, thereby removing the creator as an owner

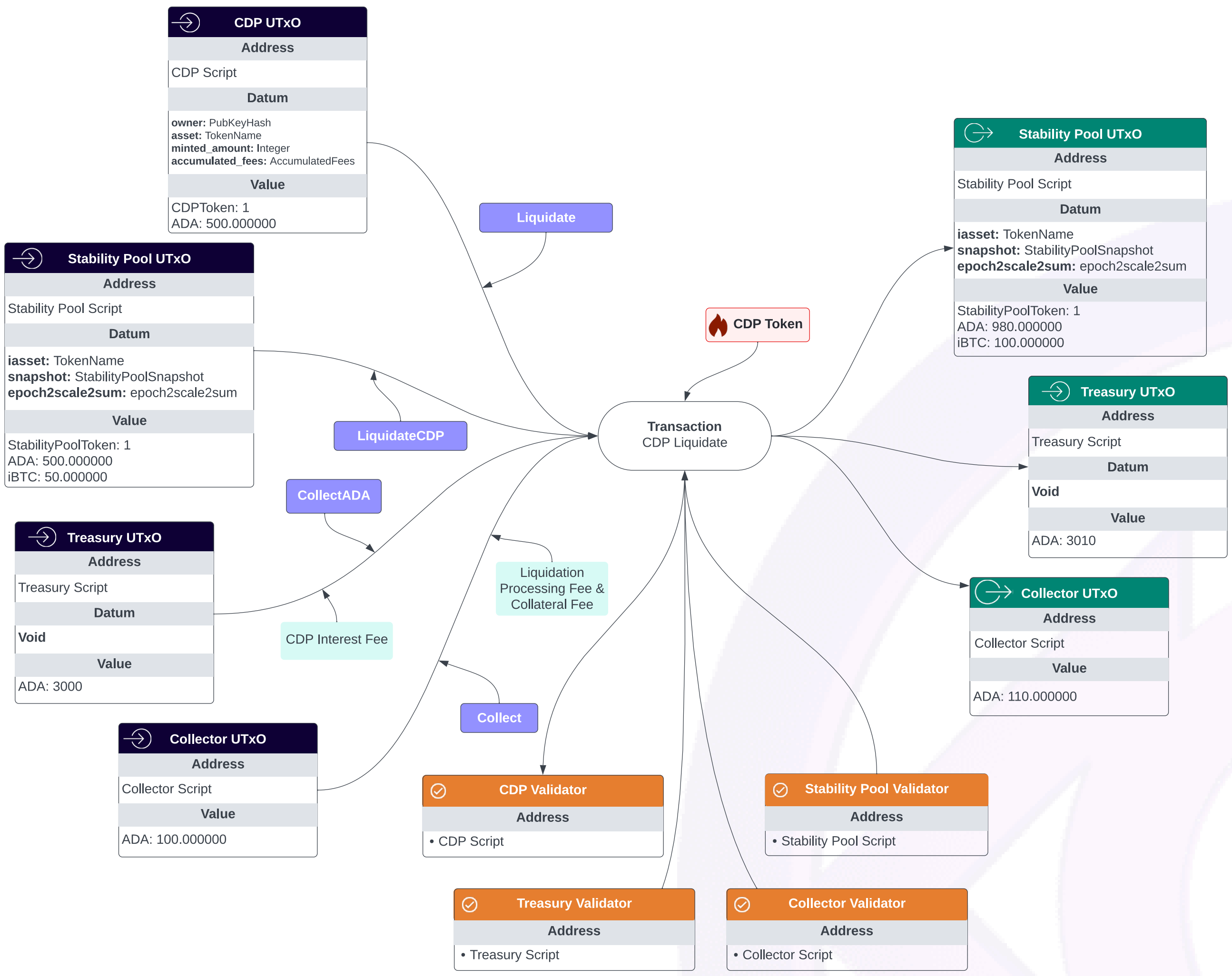


Figure 30: Example of a CDP with a debt of 50 iAsset and collateral of 500 ADA being liquidated, with the collateral being transferred to the Stability Pool, and the iAsset from the Stability Pool being burned

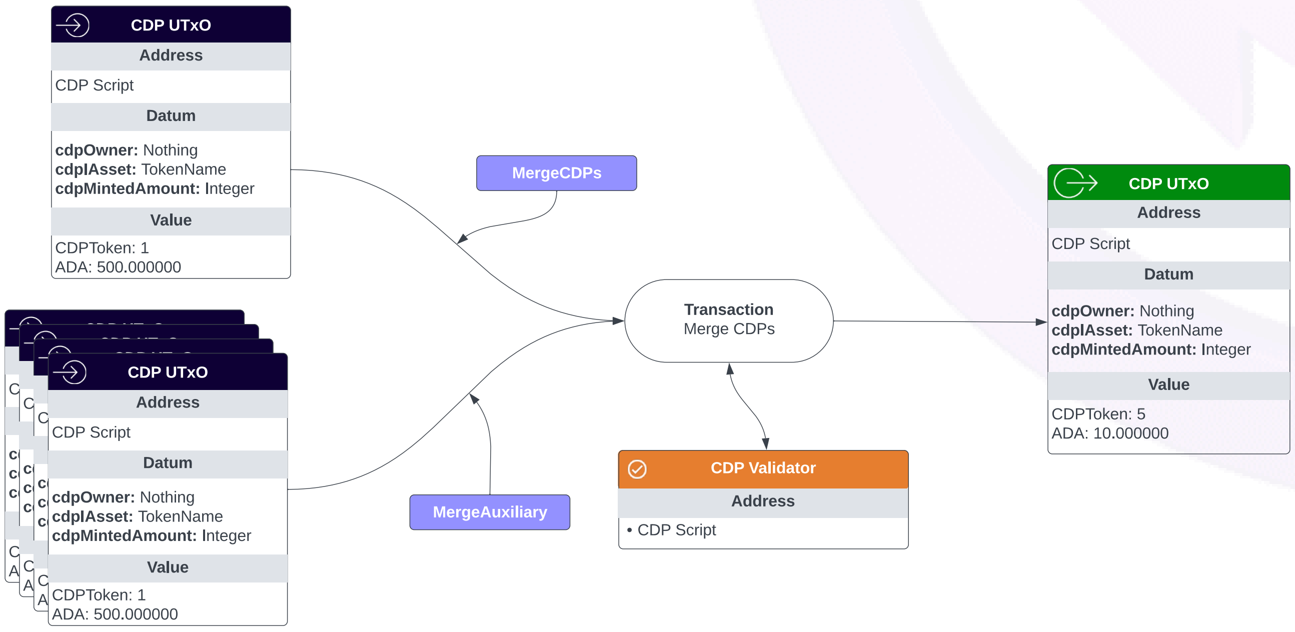


Figure 31: Example of 5 CDPs being merged together

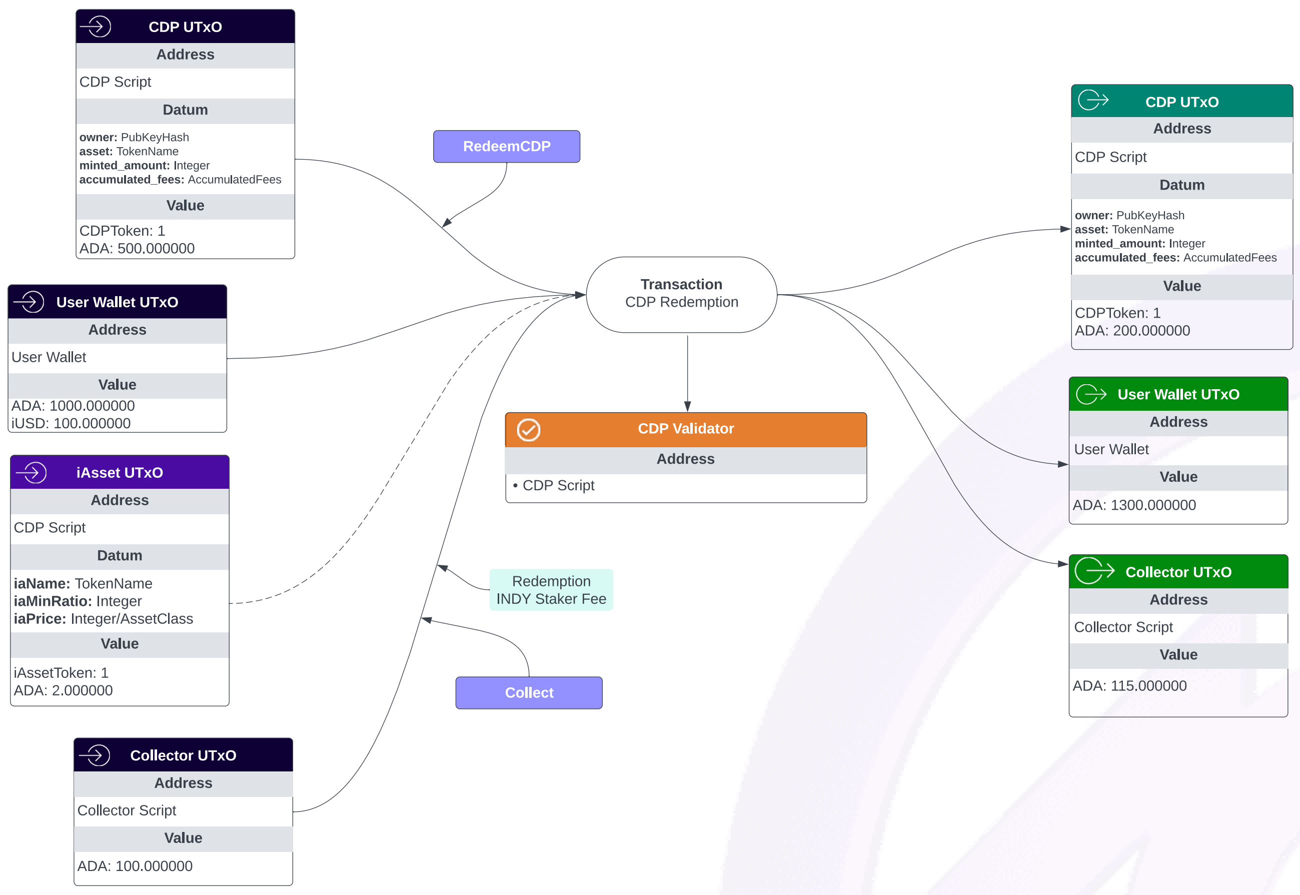


Figure 32: Example of a CDP being redeemed for 300 ADA

- `cdpToken :: CDPToken`. Token for identifying authentic CDP output.
- `versionRecordToken :: VersionRecordToken`. Token for identifying the version record for a protocol upgrade.
- `collectorValHash :: ValidatorHash`. The validator hash for the collector contract.
- `govNFT :: GovNFT`. NFT for identifying authentic governance parameters
- `accountCreateFeeLovelaces :: Integer`. Amount of lovelaces required to pay as an account creation fee
- `accountAdjustmentFeeLovelaces :: Integer`. Amount of lovelaces required to pay as an account adjustment fee
- `requestCollateralLovelaces :: Integer`. Minimum amount of lovelaces required to pay as collateral for a request

Table 20: Stability Pool outputs

Type	**Description**	**Datum**	**Values**
StabilityPool	Each StabilityPool output holds iAssets to be used for liquidations	*iasset*: The name of the iAsset that this SP is for *snapshot*: The snapshot of funds for the SP. See Snapshot *epoch2scale2sum*: A map of the sum of funds for a particular epoch and scale	*StabilityPoolToken*: 1 *iAsset*: Funded by stability providers *ADA*: Collateral transferred to SP from liquidated CDPs
EpochToScaleToSum	Archives EpochToScaleToSum records	*snapshot*: A snapshot of EpochToScaleToSum *asset*: The name of the iAsset that this snapshot is for	*SnapshotToken*: 1
Account	Designates an active Stability Pool Account	*owner*: The owner of the SP Account *iasset*: The name of the iAsset that this SP Account is for *snapshot*: The snapshot of funds from the SP at the time of deposit	*AccountToken*: 1
Request	Designates an active Stability Pool Request	*owner*: The owner of the SP Request *request*: The action intended to be done by this request *iasset*: The name of the iAsset that this SP Request is for *output_address*: The address to send the leftover collateral to	*RequestToken*: 1 *iAsset*: optional, The iAssets meant to be deposited to the Stability Pool

3.2.2 Stability Pool Endpoints

SP: Create Account (Request) Creates a request to create an account with a stability pool

Type	**Amount**	**Description**
Consume	1	The users wallet with the iAssets to deposit in the created account
Output	1	Stability Pool Account UTXO with the requested action
Output	1	A UTxO with the users wallet change

SP: Create Account (Execution) Executes a request to create a Stability Pool Account

Type	**Amount**	**Description**
Redeemer	N.A.	ProcessRequest, the stability pool input
Redeemer	N.A.	ProcessRequest, the request input
Consume	1	Stability Pool UTxO
Consume	1	Stability Pool Account UTxO

Type	Amount	Description
Mint	1	Account Token representing the user's new Stability Pool account
Output	1	Stability Pool UTXO with the updated snapshot
Output	1	Stability Pool Account UTXO with the snapshot and account owner
Output	1	A UTxO with the requested users wallet change

SP: Account Adjust (Request) Creates a request to adjust an existing stability pool account

Type	Amount	Description
Redeemer	N.A.	RequestAction, the account input
Consume	1	The users wallet with the iAssets to deposit in the created account
Consume	1	Stability Pool Account associated with the account owner
Output	1	Stability Pool Account associated with the account owner
Output	1	A UTxO with the users wallet change

SP: Adjust Account (Execution) Executes a request to adjust a Stability Pool Account

Type	Amount	Description
Redeemer	N.A.	ProcessRequest, the stability pool input
Redeemer	N.A.	ProcessRequest, the stability pool account input
Consume	1	Stability Pool UTxO
Consume	1	Stability Pool Account UTxO
Output	1	Stability Pool UTXO with the updated snapshot
Output	1	Stability Pool Account UTXO with the updated snapshot and account owner
Output	1	A UTxO with the requested users wallet change

SP: Account Close (Request) Creates a request to close an existing stability pool account

Type	Amount	Description
Redeemer	N.A.	RequestAction, the account input
Consume	1	Stability Pool Account associated with the account owner
Output	1	Stability Pool Account associated with the account owner
Output	1	A UTxO with the users wallet change

SP: Close Account (Execution) Executes a request to close a Stability Pool Account

Type	Amount	Description
Redeemer	N.A.	ProcessRequest, the stability pool input
Redeemer	N.A.	ProcessRequest, the stability pool account input
Consume	1	Stability Pool UTxO
Consume	1	Stability Pool Account UTxO
Output	1	Stability Pool UTXO with the updated snapshot

Type	Amount	Description
Output	1	Stability Pool Account UTXO with the updated snapshot and account owner
Output	1	A UTxO with the requested users wallet change

SP: Cancel Account Creation Cancels the account creation

Type	Amount	Description
Redeemer	N.A.	AnnulRequest, the stability pool account input
Consume	1	Stability Pool Account UTxO
Output	1	A UTxO with the requested users wallet change

SP: Cancel Account Request Cancels a request on an existing account

Type	Amount	Description
Redeemer	N.A.	AnnulRequest, the stability pool account input
Consume	1	Stability Pool Account UTxO
Output	1	Stability Pool Account UTXO with no requested action
Output	1	A UTxO with the requested users wallet change

3.3 Staking

The Staking contract is used primarily by the Indigo DAO Governance package for proving the ownership of INDY tokens and locking those tokens upon voting. The Staking contract also includes functionality to collect protocol fees from Collector UTXOs.

Table 29: Staking native tokens

Name	Description	Minting Policy
StakingManagerNFT	The NFT identifies the authentic StakingManager output The NFT must be stored in the StakingManager output Validator scripts ensure that this NFT always stays at the StakingManager output	The protocol mints exactly 1 token, before launch
StakingToken	Identify the authentic StakingPosition output	The transaction must consume a StakingManagerNFT or a StakingToken

3.3.1 Parameters

- `stakingManagerNFT :: StakingManagerNFT`. NFT of StakingManager.
- `stakingToken :: StakingToken`. Token for identifying authentic Staking Position output.
- `indyToken :: INDY`.
- `pollToken :: PollToken`. Token identifying authentic Poll output.
- `versionRecordToken :: VersionRecordToken`. Token identifying the VersionRegistry output.
- `collectorValHash :: ValidatorHash`. The collector script, used as a bridge between Staking and Poll Script.

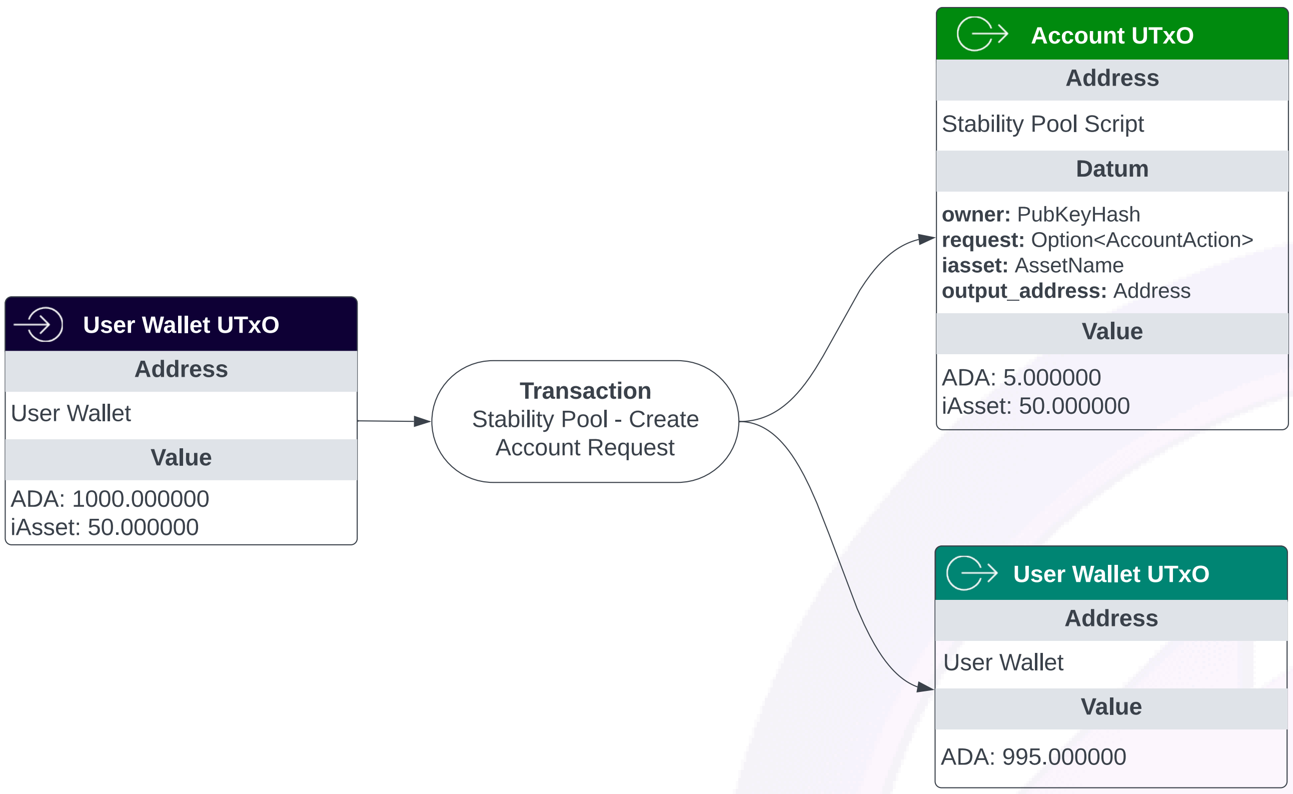


Figure 33: Example of a user creating a request to make their first deposit into a Stability Pool, depositing 50 iAsset, 2 ADA, and paying a 5 ADA fee

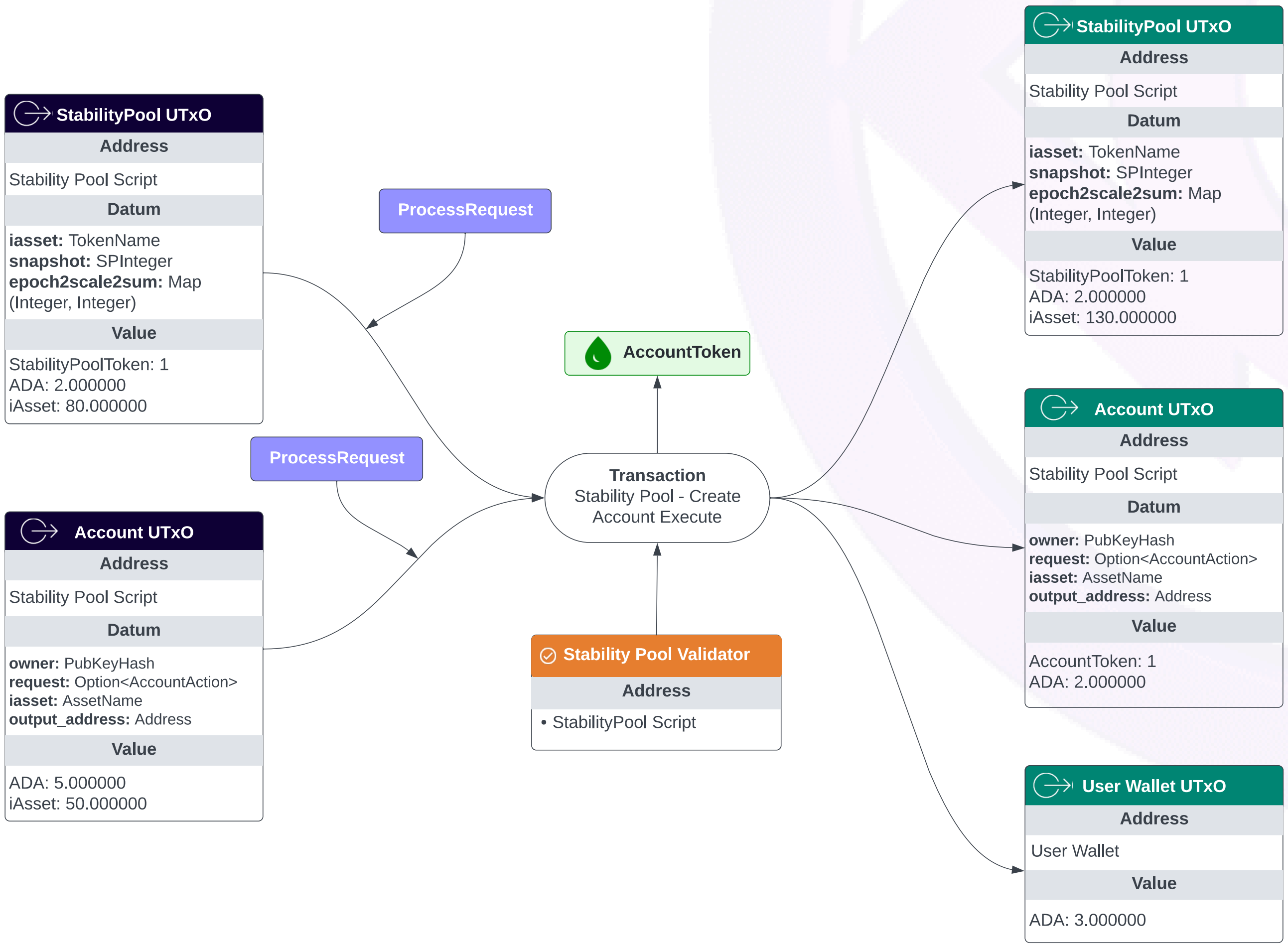


Figure 34: Example of an execution of a request of a user to make their first deposit into a Stability Pool, depositing 50 iAsset, 2 ADA, and paying a 5 ADA fee

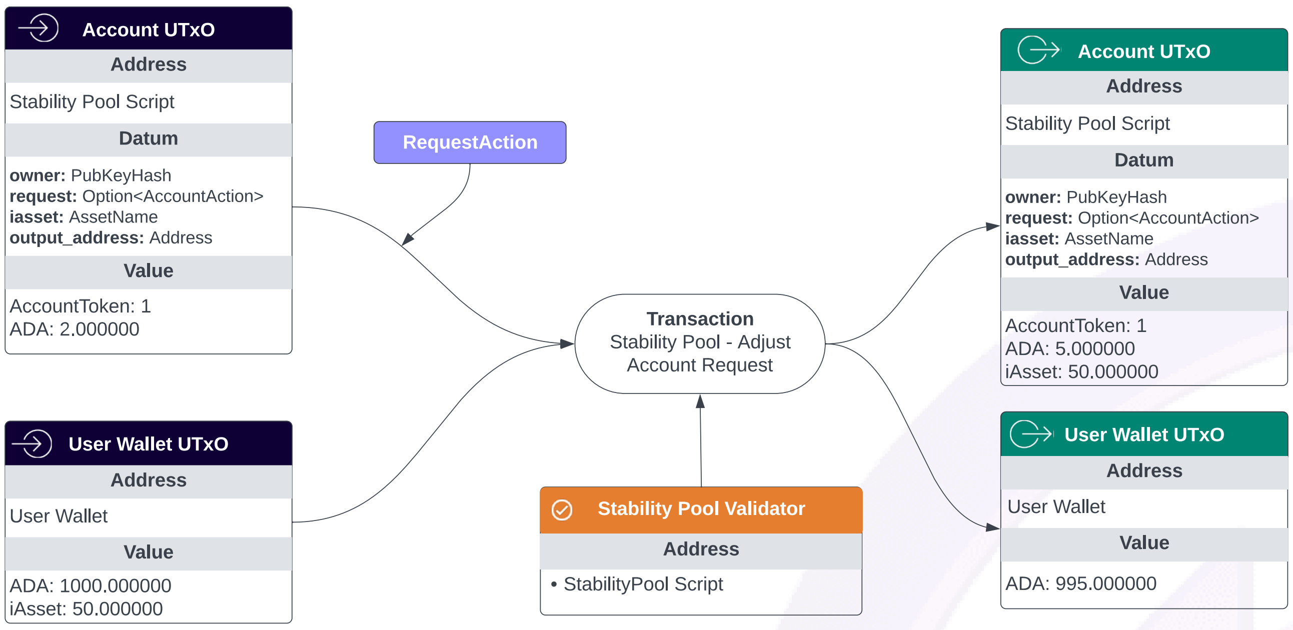


Figure 35: Example of a user creating a request to adjust their deposit in a Stability Pool account, depositing 50 iAsset

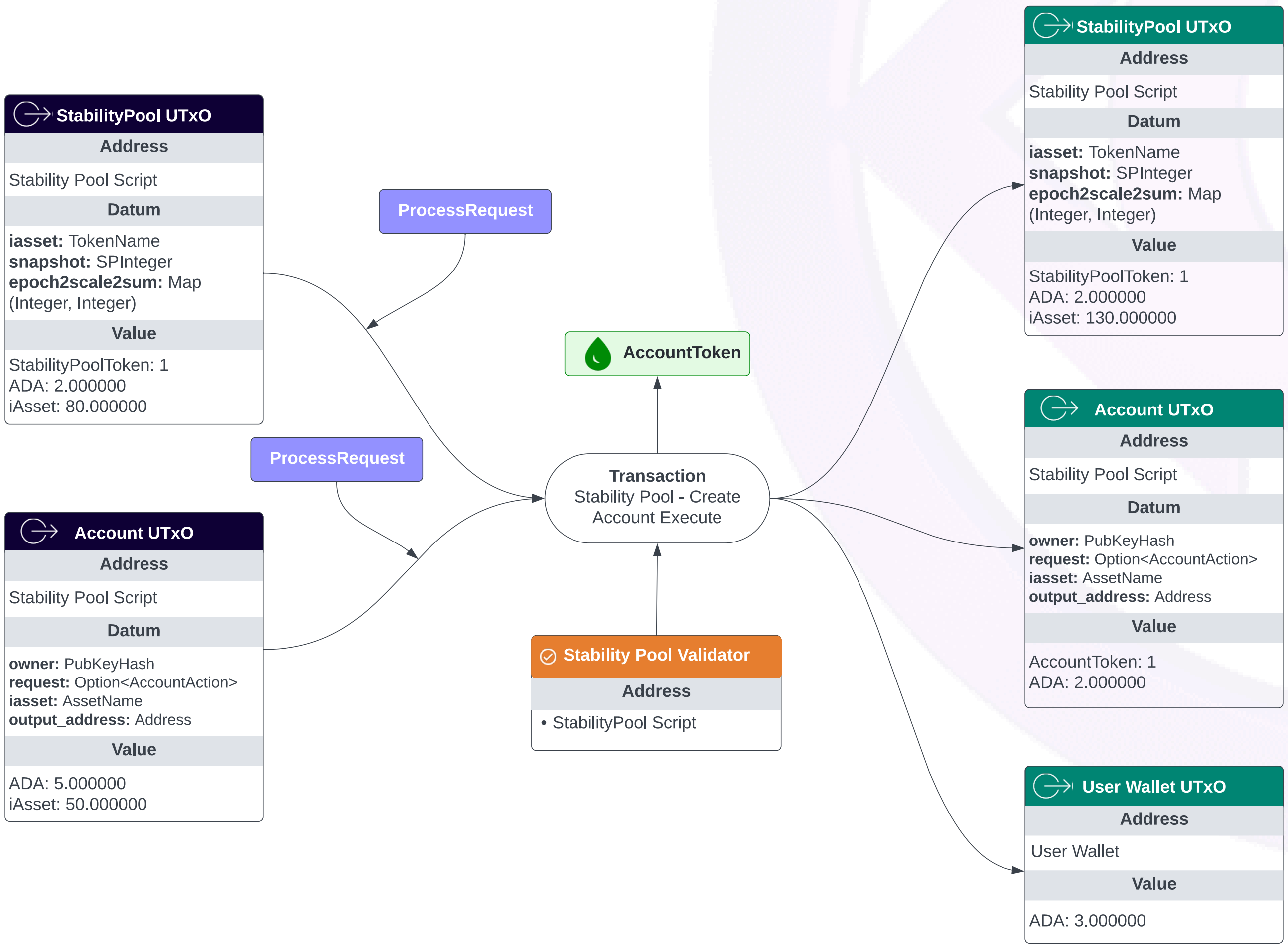


Figure 36: Example of an execution of a request of a user to make their first deposit into a Stability Pool, depositing 50 iAsset, 2 ADA, and paying a 5 ADA fee

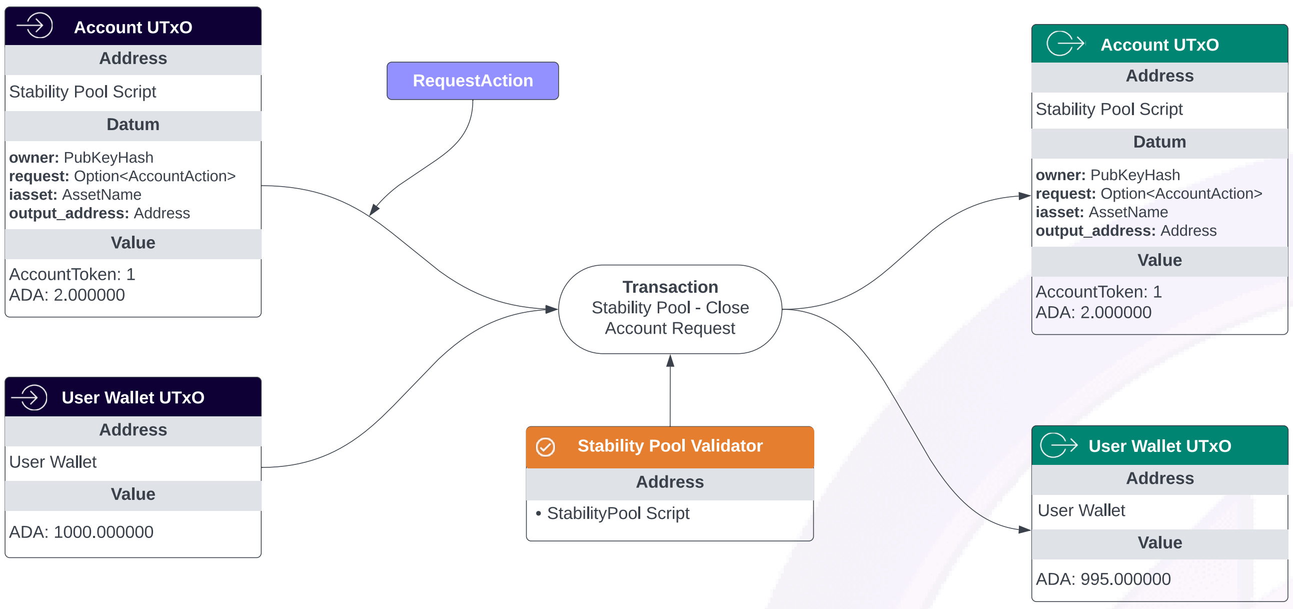


Figure 37: Example of a user creating a request to close their Stability Pool account

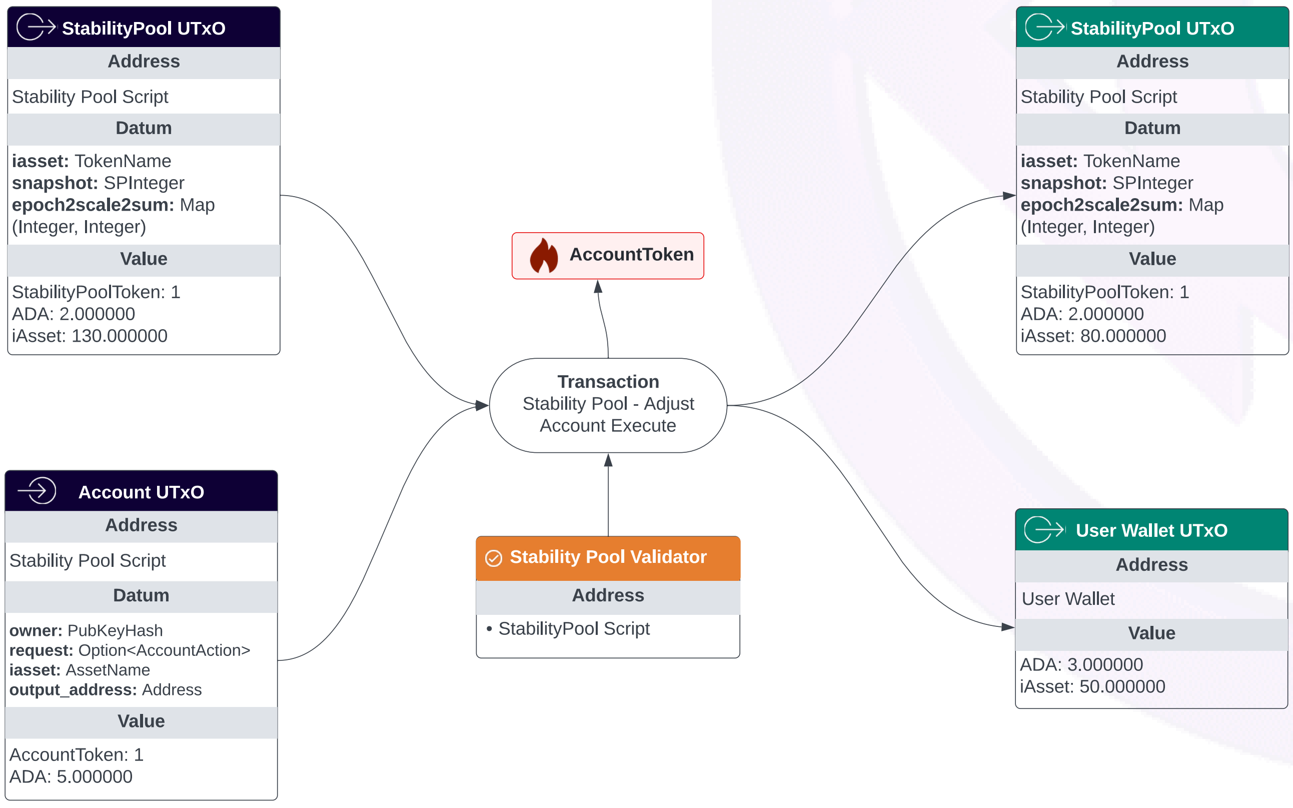


Figure 38: Example of an execution of a request of a user closing a Stability Pool Account fee

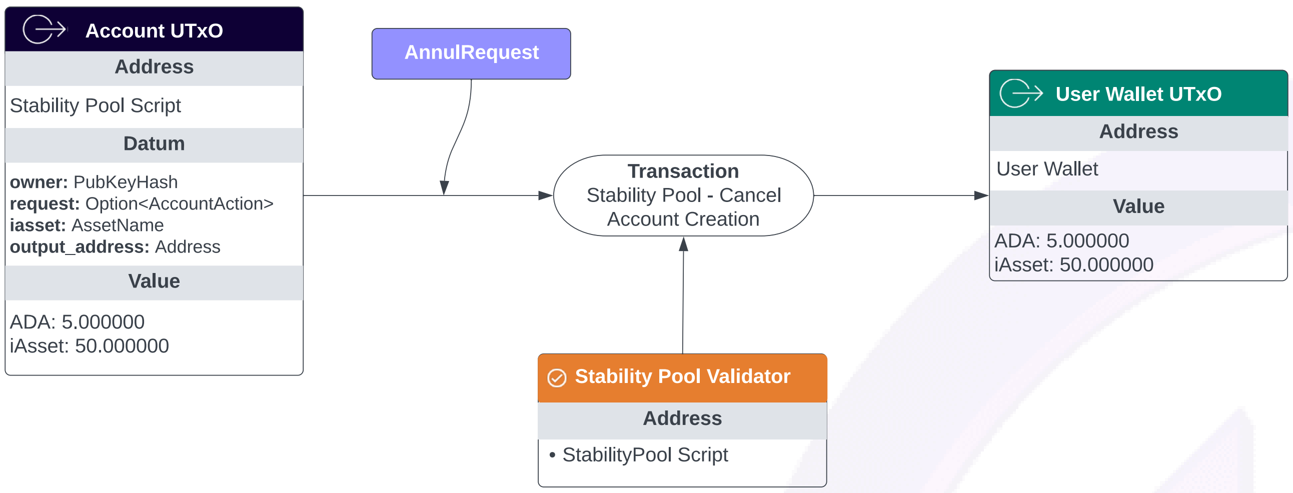


Figure 39: Example of a user canceling their request to create a Stability Pool Account

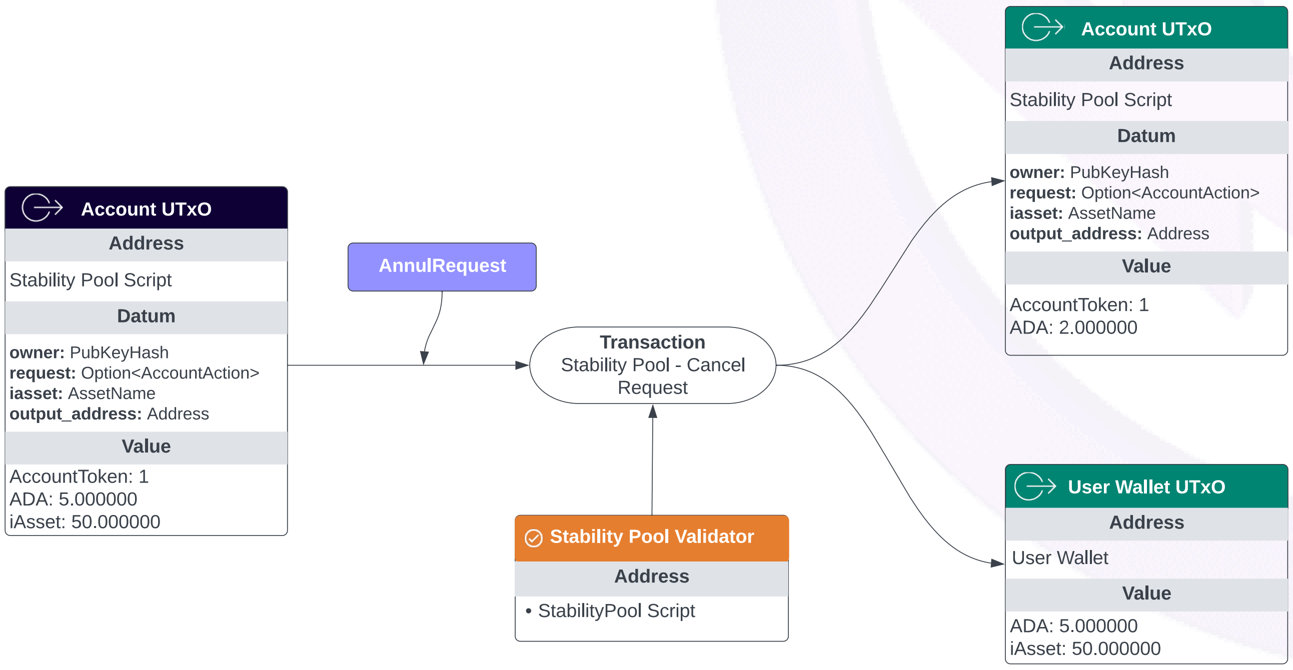


Figure 40: Example of an execution of a request of a user canceling their Account Request

- `cdpToken :: CDPToken`. Necessary for OffChain Endpoint to construct CollectorScriptParams.

Table 30: Staking outputs

Type	Description	Datum	Values
StakingManager	Only one output of this type is stored in the script To create a StakingPosition output, the user must consume this output in the transaction	*totalStake*: The total amount of staked INDY *mSnapshot*: The snapshot of ADA rewards for INDY stakers	*StakingManagerNFT*: 1
StakingPosition	An individual user's INDY staking position	*owner*: The owner of the staking position *lockedAmount*: A map of Poll ID to (Vote Amount, Proposal End Time) *pSnapshot*: The snapshot of ADA rewards for the INDY staker	*StakingToken*: 1

3.3.2 Staking Endpoints

Staking: Create Creates a user's staking position

Type	Amount	Description
Redeemer	N.A.	CreateStakingPosition
Consume	1	Staking Manager UTXO representing the global state of staking positions
Consume	1+	UTXOs containing the user's INDY to be staked
Mint	1	Staking Token representing the user's staking position
Output	1	Staking Manager UTXO with the updated global state
Output	1	Staking Position UTXO holding the user's Staking Token

Staking: Unstake Unstakes a user's staking position

Type	Amount	Description
Redeemer	N.A.	UpdateTotalStake
Redeemer	N.A.	Unstake
Consume	1	Staking Manager UTXO representing the global state of staking positions
Consume	1	Staking Position UTXO representing the user's staking position
Burn	1	Staking Token representing the user's former staking position
Output	1	Staking Manager UTXO with the updated global state
Output	1	UTXOs containing the user's previously staked INDY

Staking: Stake Adds more INDY to a user's staking position

Type	Amount	Description
Redeemer	N.A.	UpdateTotalStake
Redeemer	N.A.	AdjustStakedAmount

Type	Amount	Description
Consume	1	Staking Manager UTXO representing the global state of staking positions
Consume	1	Staking Position UTXO representing the user's staking position
Consume	1+	UTXOs containing the INDY to be staked
Output	1	Staking Manager UTXO with the updated global state
Output	1	Staking Position UTXO representing the user's updated staking position

Staking: Distribute Distributes fees from the Collector to the Staking Manager

Type	Amount	Description
Redeemer	N.A.	Distribute
Redeemer	N.A.	Collect
Consume	1	Staking Manager UTXO representing the global state of staking positions
Consume	1+	Collector UTXOs containing the fees to distribute
Output	1	Staking Manager UTXO with the updated global state

Staking: Withdraw Rewards Withdraw ADA rewards allocated to a user's staking position

Type	Amount	Description
Redeemer	N.A.	UpdateTotalStake
Redeemer	N.A.	AdjustStakedAmount
Consume	1	Staking Manager UTXO representing the global state of staking positions
Consume	1	Staking Position UTXO representing the user's staking position
Output	1	Staking Manager UTXO with the updated global state
Output	1	Staking Position UTXO representing the user's staking position

Table 36: Governance native tokens

Name	Description	Minting Policy

3.4 Governance

Governance is a group of several contracts: Gov, Poll, Execute, and VersionRegistry. The Gov contract stores protocol parameters and controls the creation/ending of a Governance Poll. The Poll contract handles the creation of vote shards, voting, merging of vote shards, AQB calculations, and the ending of a Poll. The Execute contract takes the result of a passed proposal and applies the appropriate action to the contracts. The VersionRegistry contract handles the creation of Version Records, which can be used by other protocol scripts to find an upgrade path.

Table 36: Governance native tokens

Name	Description	Minting Policy
GovNFT	Identify authentic Governance output Governance script ensures that this NFT always stays at the Governance output	The protocol mints exactly 1 token at initialization
PollToken	Identifies an authentic proposal Validator scripts ensure that this token always stays at Poll output	The transaction must consume GovNFT
UpgradeToken	Identifies a passed proposal and the upgrade contract Validator scripts ensure that this token always stays at Execute output	The transaction must consume a PollToken
VersionRecordToken	Identifies a potential upgrade path for a contract Validator scripts ensure that this token always stays at VersionRegistry output	The transaction must consume UpgradeToken

3.4.1 Execute Script Parameters

- `govNFT :: GovNFT`. NFT for identifying authentic Governance Script output.
- `upgradeToken :: UpgradeToken`. The asset class for identifying a valid upgrade token.
- `iAssetToken :: iAssetToken`. Token for identifying authentic iAsset output.
- `stabilityPoolToken :: StabilityPoolToken`. Token for identifying authentic SP output.
- `versionRecordToken :: VersionRecordToken`. Token for identifying the version record for a protocol upgrade.
- `cdpValHash :: ValidatorHash`. Hash of CDP script, used for verifying the output of a CDP.
- `sPoolValHash :: ValidatorHash`. Hash of SP script, used for verifying the output of a SP.
- `versionRegistryValHash :: ValidatorHash`. Hash of Version Registry script, used for verifying the output of a Version Registry.
- `maxInterestPeriods :: Integer`. The maximum number of interest periods that can be stored in IAsset datum.

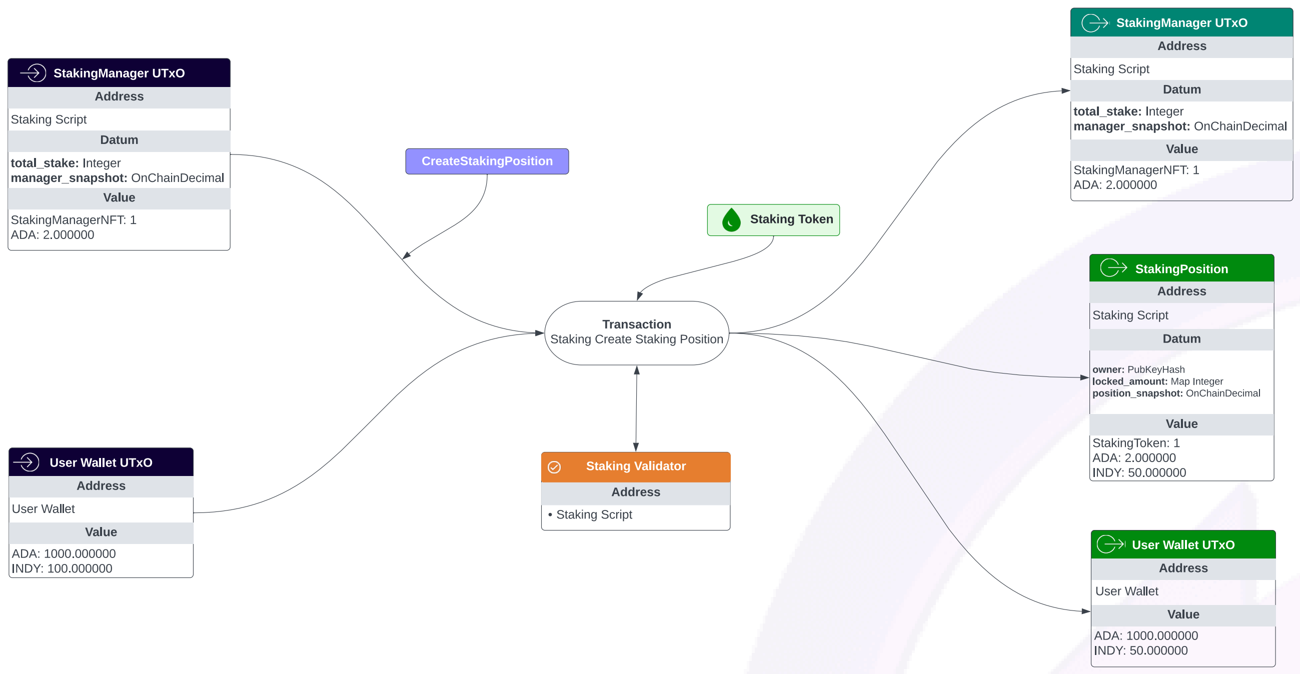


Figure 41: Example of a user staking INDY for the first time, depositing 50 INDY

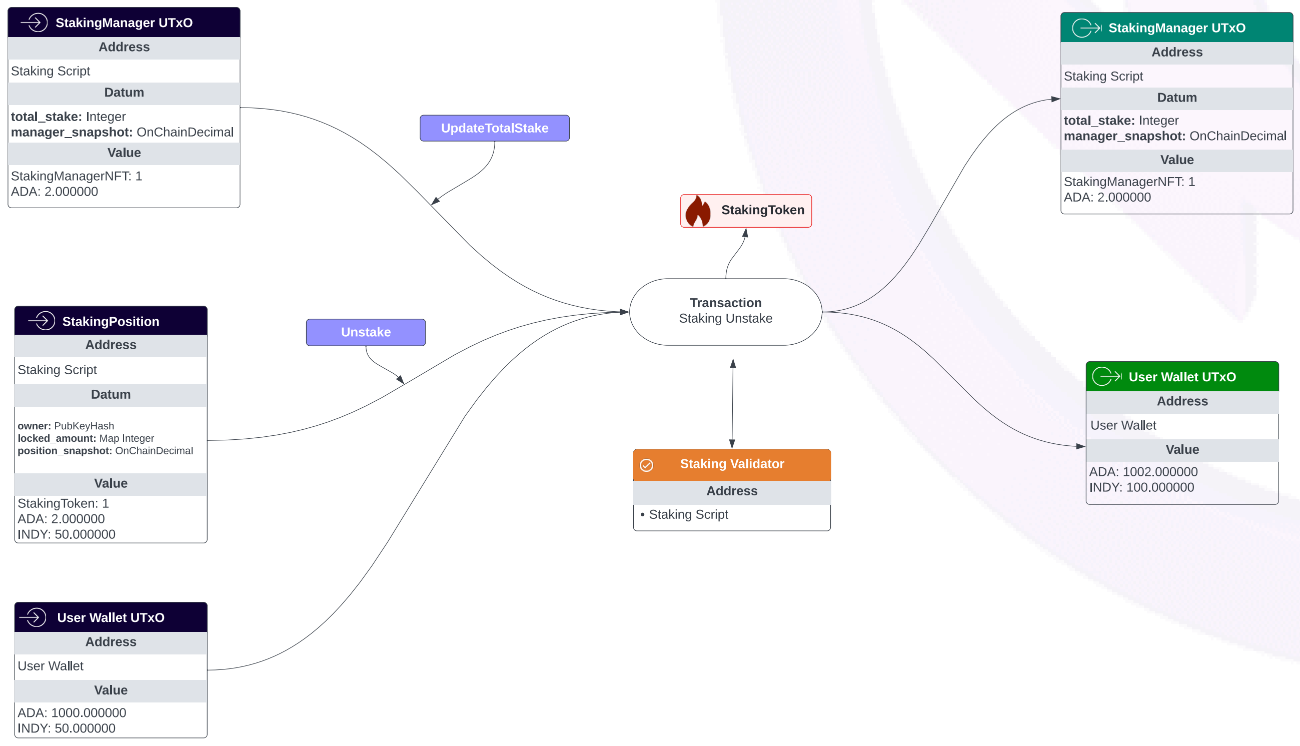


Figure 42: Example of a user unstaking 50 INDY

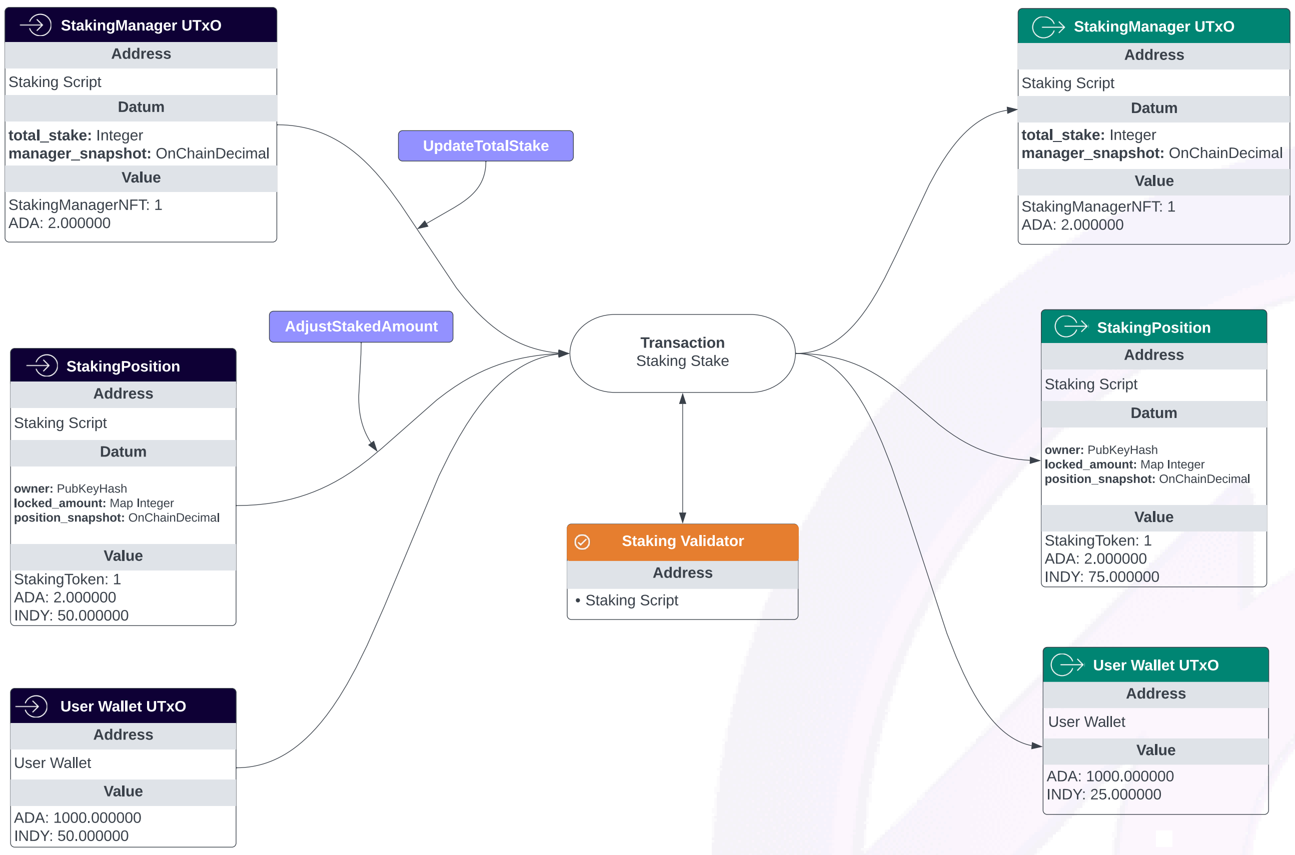


Figure 43: Example of a user staking an additional 25 INDY

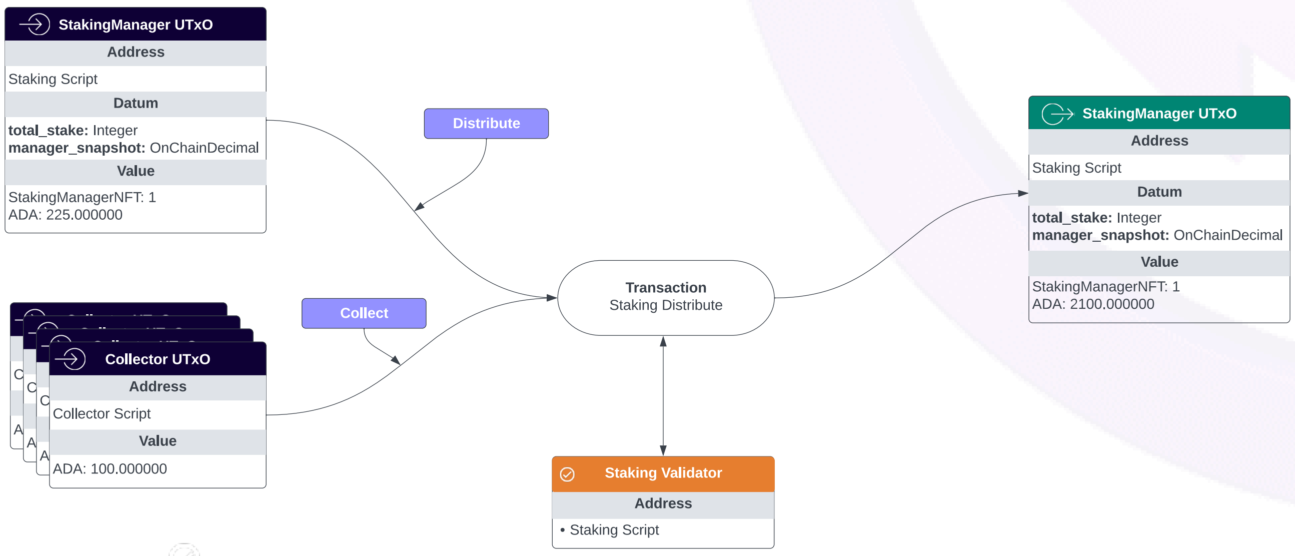


Figure 44: Example of fees from the Collector being distributed to the Staking Manager

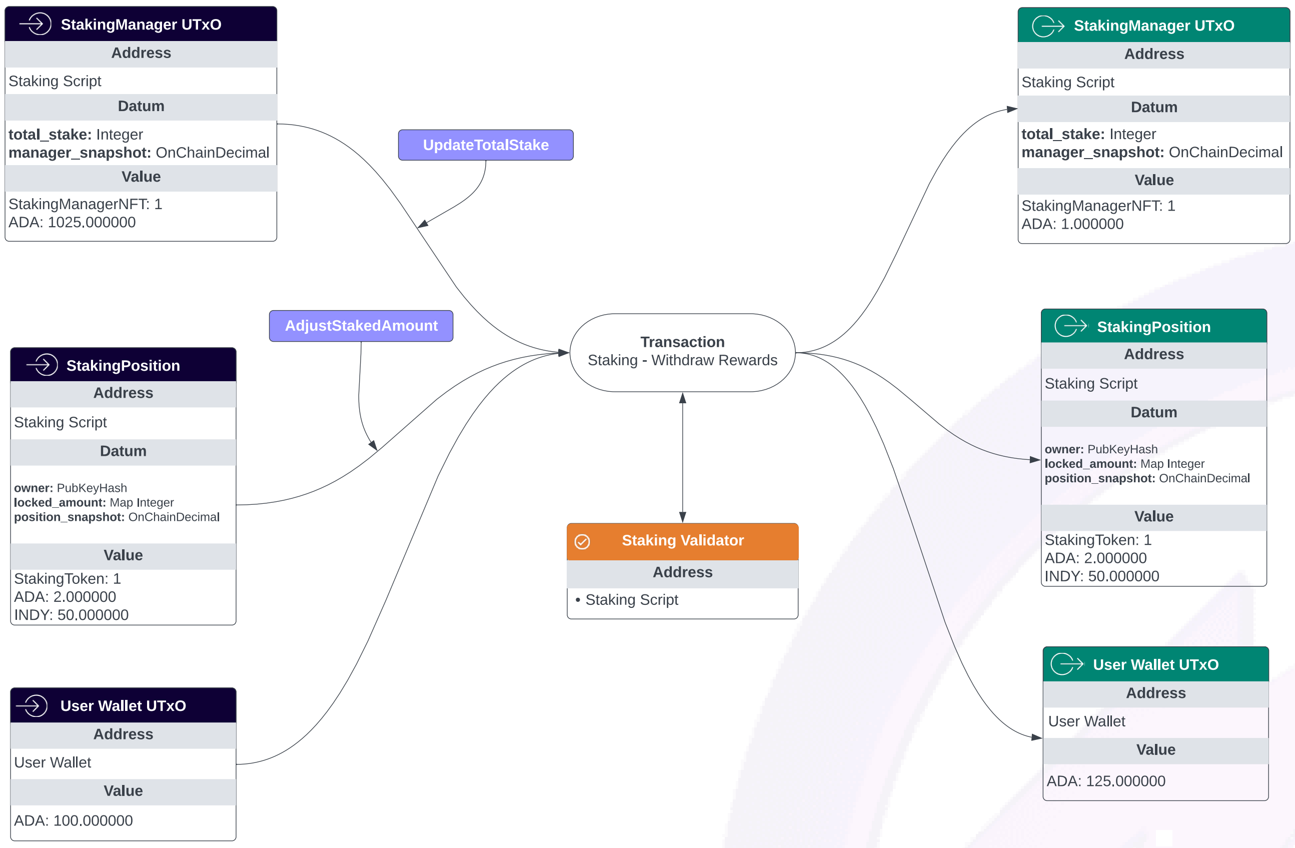


Figure 45: Example of a user withdrawing a 25 ADA reward

3.4.2 Gov Script Parameters

- `govNFT :: GovNFT`. NFT for identifying authentic Governance Script output.
- `pollToken :: PollToken`. The asset class for identifying a valid Poll token.
- `upgradeToken :: UpgradeToken`. The asset class for identifying a valid Upgrade token.
- `indyAsset :: INDY`.
- `versionRecordToken :: VersionRecordToken`. Token for identifying the version record for a protocol upgrade.
- `pollManagerValHash :: ValidatorHash`. Hash of Poll Manager script, used for verifying the output of a Poll.
- `gBiasTime :: POSIXTime`. Used to apply some leverage to the voting procedures.
- `iassetAuthToken :: AssetAuthToken`. The token that authenticates a valid iAsset UTxO.

3.4.3 Poll Manager Script Parameters

- `govNFT :: GovNFT`. NFT for identifying authentic Governance Script output.
- `pollToken :: PollToken`. The asset class for identifying a valid Poll token.
- `upgradeToken :: UpgradeToken`. The asset class for identifying a valid Upgrade token.
- `indyAsset :: INDY`.
- `govExecuteValHash :: ValidatorHash`. Hash of Execute script, used for verifying the output of a Upgrade token.
- `pBiasTime :: POSIXTime`. Used to apply some leverage to the voting procedures.
- `shardsValHash :: ValidatorHash`. Hash of the poll shards script.

- `treasuryValHash :: ValidatorHash`. Hash of the treasury script.
- `initialIndyDistribution :: Integer`. Used by the electorate calculation for the ITD value.

3.4.4 Poll Shard Script Parameters

- `pollToken :: PollToken`. The asset class for identifying a valid Poll token.
- `stakingToken :: StakingToken`. The asset class for identifying a valid Staking Position token.
- `indyAsset :: INDY`.
- `stakingValHash :: ValidatorHash`. Hash of Staking script, used for verifying the output of the Staking token.

3.4.5 Version Record Script Parameters

- `upgradeToken :: UpgradeToken`. The asset class for identifying a valid Upgrade token.

Table 37: Governance outputs

Type	Description	Datum	Values
Governance	Only one output of this type is stored in the script To create a Poll output, the user must consume this output in the transaction To store the protocol parameters	*currentProposal*: The number of opened proposals *protocolParams*: The parameters of the protocol *currentVersion*: The current version of the protocol, starting at 0 *iassetsCount*: The number of active iassets (including delisted) *activeProposals*: The number of active proposals	*GovNFT*: 1

Table 37: Governance outputs

Type	**Description**	**Datum**	**Values**
Poll Manager	The Poll Manager acts as a central UTXO that manages the content of the poll	*pollId*: The identifying key for this particular proposal *pollOwner*: The pub key hash of the owner of the poll *pollContent*: The intended action of this poll: ProposeAsset, MigrateAsset, ModifyProtocolParams, UpgradeProtocol, and TextProposal *pollTreasuryWithdrawal*: The value allowed to be withdrawn from the treasury *pollStatus*: The count of yes and no votes *pollEndTime*: The time in which the poll should be ended *pollCreatedShards*: The number of shards created *pollTalliedShards*: The number of shards tallied and merged into Poll Manager *pollTotalShards*: The number of shards in total *pollProposeEndTime*: The time in which all of the poll shards must be created within *pollExpirationTime*: The time in which the poll should expire *pollProtocolVersion*: The protocol version at the time the poll UTXO was created *pollMinimumQuorum*: The minimum amount of INDY for a proposal to be possible to pass.	*PollToken*: 1
Poll Shard	A derivation of the Poll Manager that stores some votes	*pollId*: The identifying key for this particular proposal *pollStatus*: The count of yes and no votes *pollEndTime*: The time in which the poll should be ended *pollManagerAddress*: The address of the poll manager script	*PollToken*: 1

Table 37: Governance outputs

Type	Description	Datum	Values
Upgrade	This output can be consumed to process a passed proposal	*uId*: The identifying key for the passed proposal this upgrade was derived from *uContent*: The intended action of this upgrade: ProposeAsset, MigrateAsset, ModifyProtocolParams, UpgradeProtocol, and TextProposal *uPassedTime*: The time in which the poll was passed *uEndTime*: The time in which the upgrade should be deemed ”expired” *uProtocolVersion*: The protocol version at the time the upgrade UTXO was created	*UpgradeToken*: 1
VersionRecord	Given a particular version id, the path for upgrading to a new validator	*versionId*: The version that the record is associated with. Version starts at 0 at genesis and works up *versionPaths*: A map of the validator name that should be upgraded and the currency symbol that can be used to process the upgrade	*VersionRecordToken*: 1

3.4.6 Governance Endpoints

Governance: Create Proposal Creates a proposal to enact changes

Type	Amount	Description
Redeemer	N.A.	CreatePoll, takes as parameters the time the poll voting period should end, a public key hash corresponding to a user's wallet, and the Poll's type (e.g.: ProposeAsset, MigrateAsset, etc.)
Consume	1	Governance UTXO
Consume	1+	User Wallet UTxO for INDY to be deposited to create the proposal
Mint	1	Poll Token representing the newly created proposal
Output	1	Governance UTXO
Output	1	Poll Manager UTXO that represents the proposal

Governance: Vote Vote on an open proposal

Type	Amount	Description
Redeemer	N.A.	Vote, takes as a parameter the vote choice (yes or no)
Redeemer	N.A.	Lock
Consume	1	Poll Shard UTXO to cast the vote with
Consume	1	Staking Position UTXO representing the user's voting power

Type	Amount	Description
Output	1	Poll Shard UTXO with the vote recorded
Output	1	Staking Position UTXO representing the user's voting power

Governance: Create Shards Create one or more shards to allow users to vote on proposals

Type	Amount	Description
Redeemer	N.A.	CreateShards, takes as a parameter the time the poll voting period should end
Consume	1	Poll Manager UTXO that represents the proposal
Output	∞	Poll Shard UTXOs to record votes

Governance: Merge Shards Merges one or more shards so that votes can be tallied

Type	Amount	Description
Redeemer	N.A.	MergeShardsManager, takes as a parameter the time the poll voting period should end
Redeemer	N.A.	MergeShards
Consume	1	Poll Manager UTXO that represents the proposal
Consume	∞	Poll Shard UTXOs to merge
Output	1	Poll Manager UTXO with the updated vote count

Governance: End Proposal Passed End a proposal that has passed

Type	Amount	Description
Redeemer	N.A.	EndPoll, takes as a parameter the time the poll voting period should end
Consume	1	Poll Manager UTXO that represents the proposal
Reference	1	Governance UTXO
Mint	1	Upgrade Token
Burn	1	Poll Token
Output	1	Upgrade UTXO

Governance: End Proposal (Failed or Expired) End a proposal that has failed or expired

Type	Amount	Description
Redeemer	N.A.	EndPoll, takes as a parameter the time the poll voting period should end
Consume	1	Poll Manager UTXO that represents the proposal
Reference	1	Governance UTXO
Burn	1	Poll Token
Output	1	Treasury UTXO containing the INDY deposited when the proposal was created

Governance: Execute Text Proposal Execute a passed proposal containing text adopted by the DAO

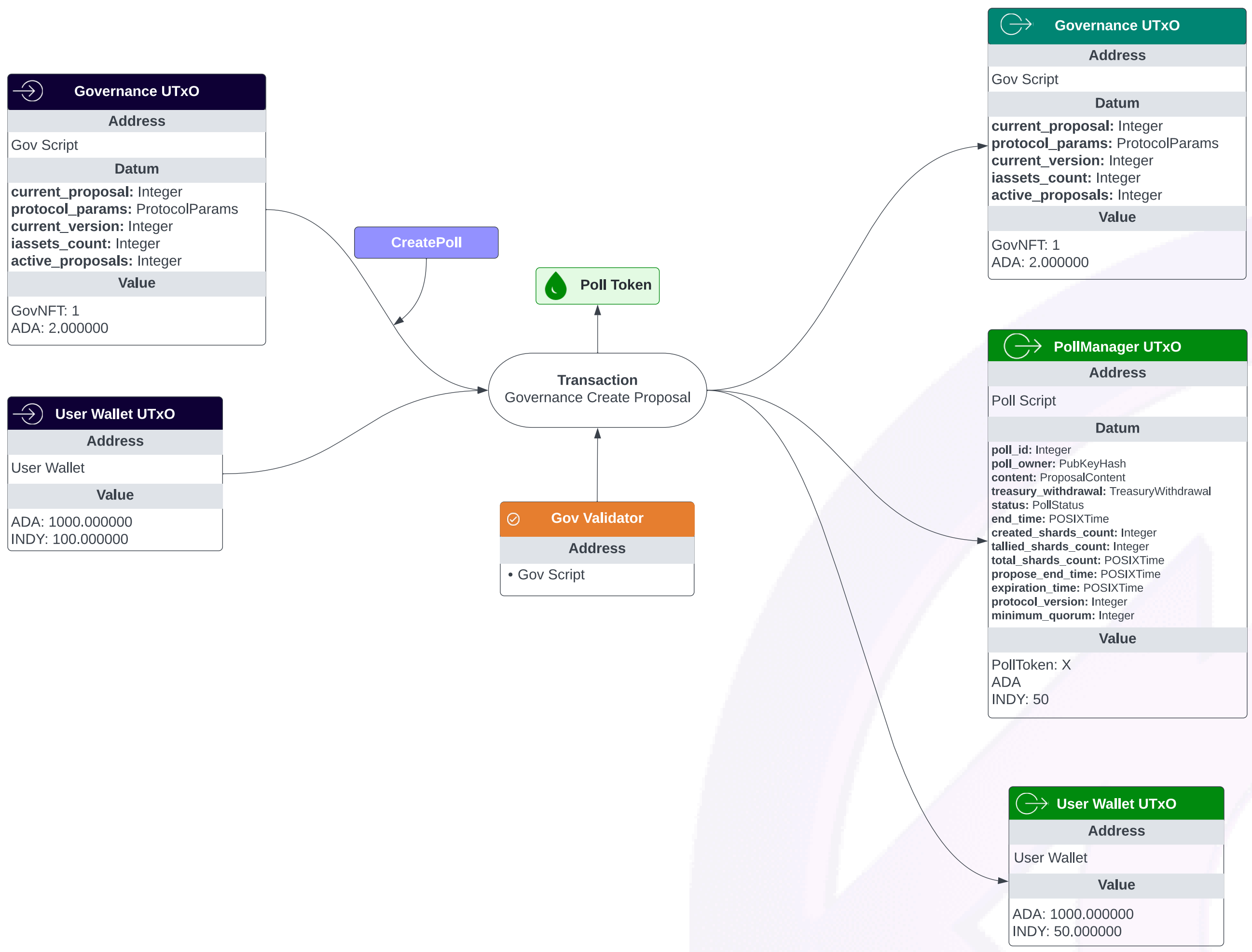


Figure 46: Example of a user depositing 50 INDY and 2 ADA to create a proposal

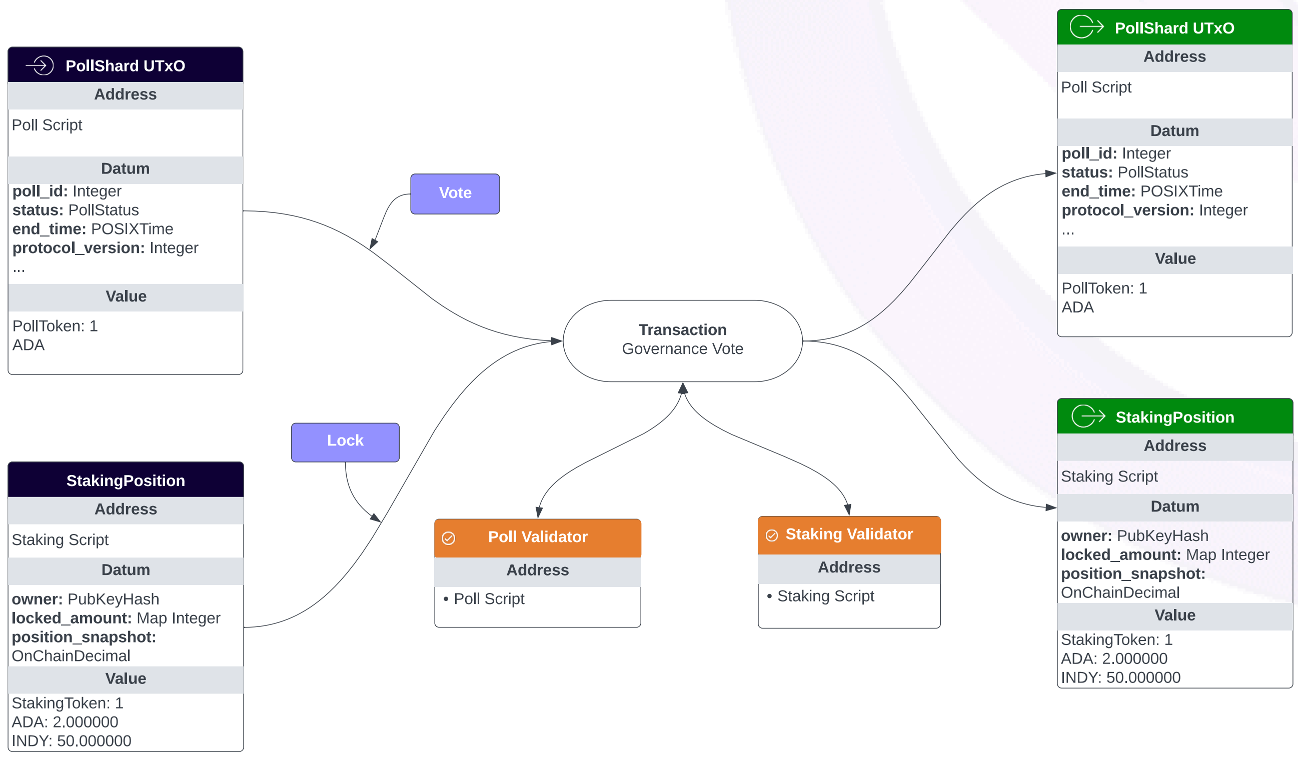


Figure 47: Example of a user casting their vote

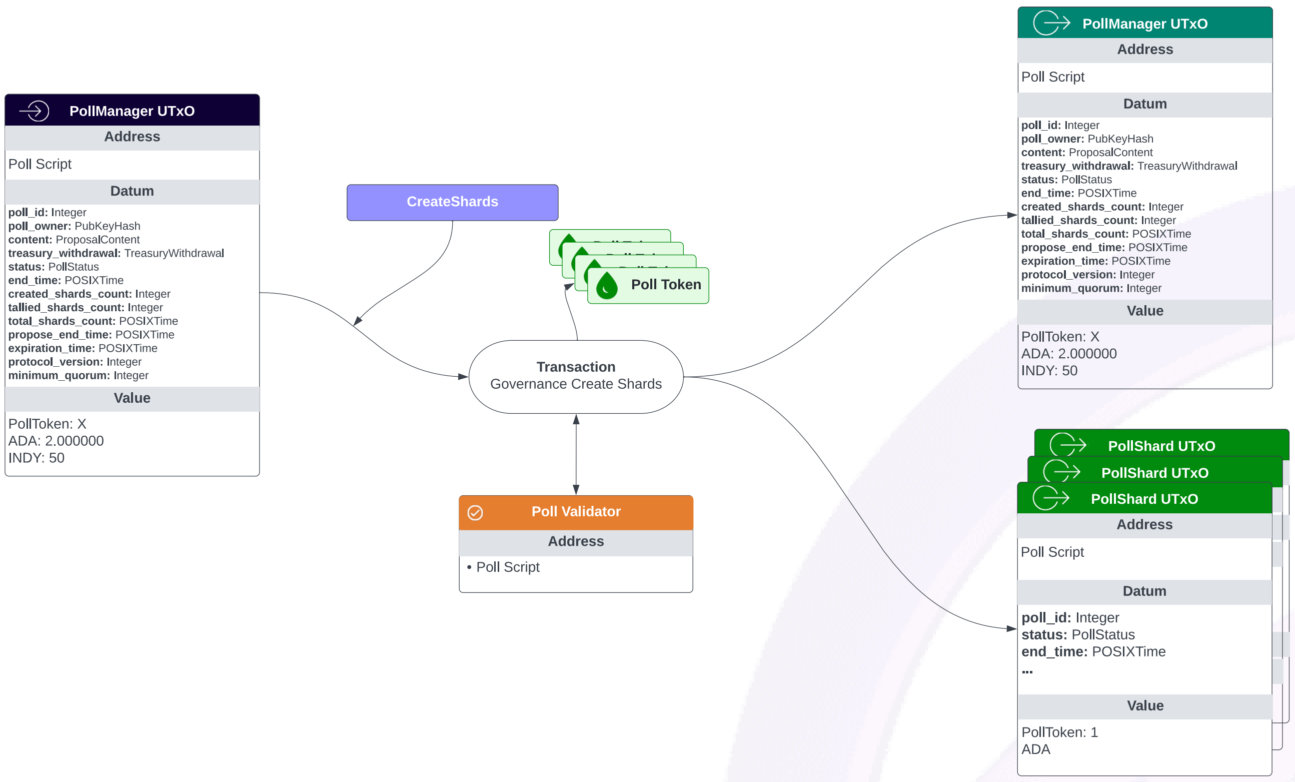


Figure 48: Example of a user creating three vote shards for their proposal and depositing a refundable 6 ADA

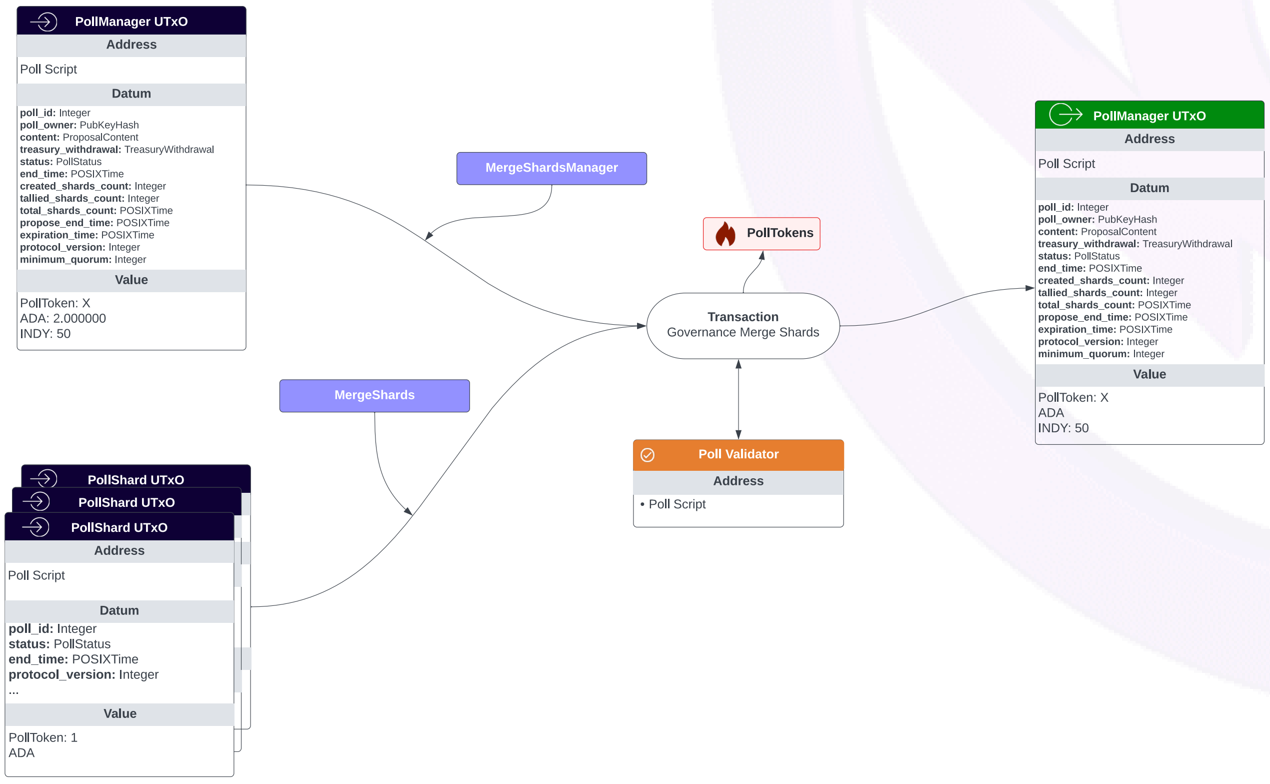


Figure 49: Example of a user merging three vote shards for their proposal and retrieving their original 6 ADA deposit

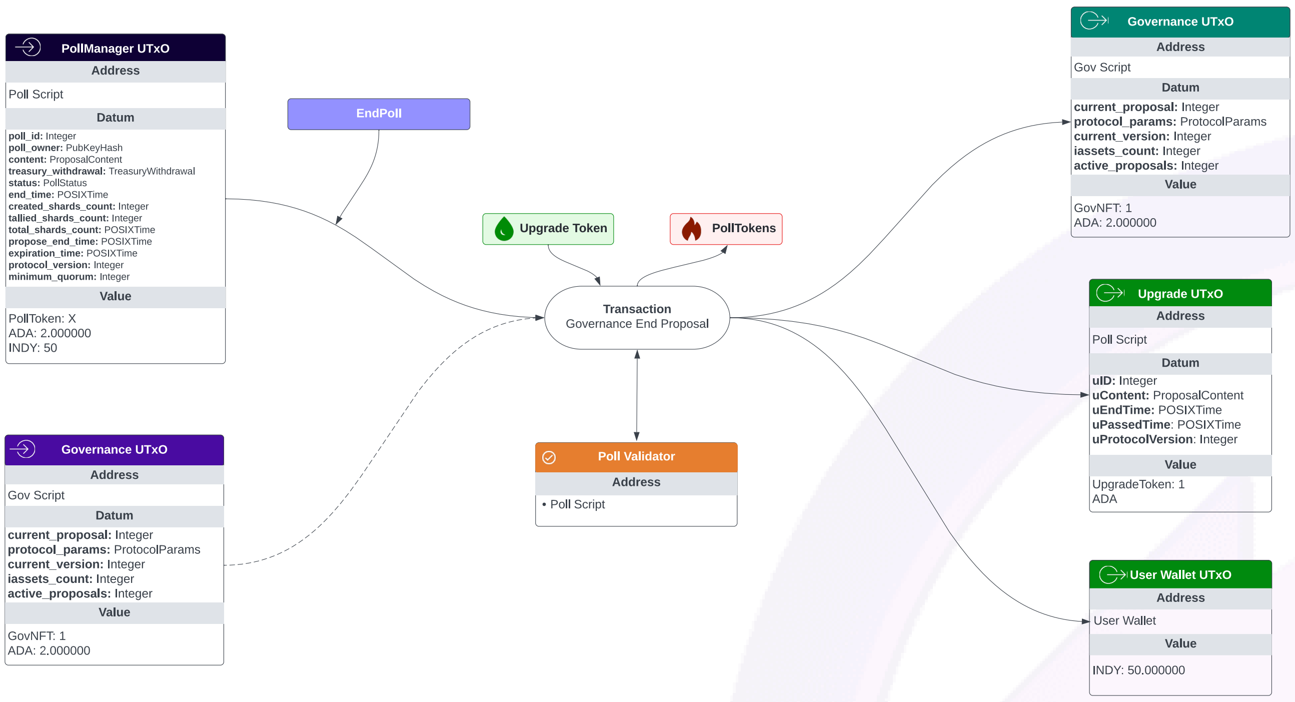


Figure 50: Example of a user ending their proposal that passed and retrieving their original 50 INDY deposit

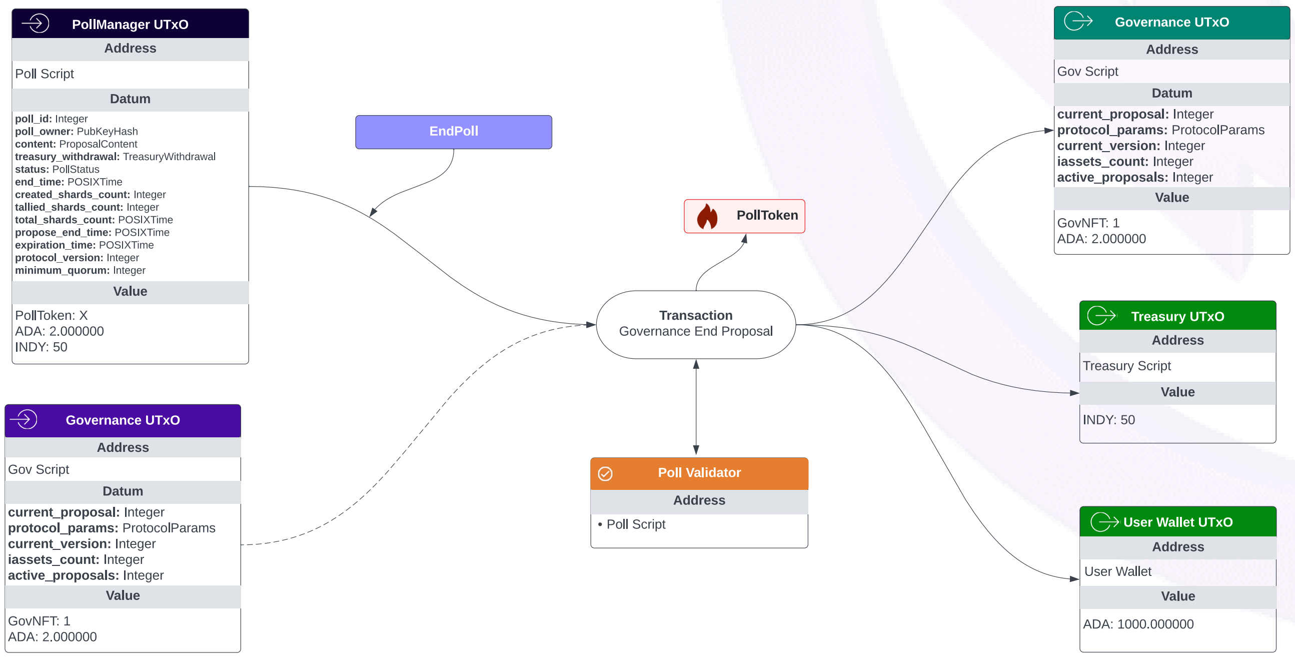


Figure 51: Example of a user ending a failed or expired proposal and sending the deposited 50 INDY to the Treasury

Type	Amount	Description
Redeemer	N.A.	EndPoll, takes as a parameter the time the poll voting period should end
Redeemer	N.A.	Execute
Consume	1	Upgrade UTXO containing the Upgrade Token for the passed proposal
Burn	1	Upgrade Token

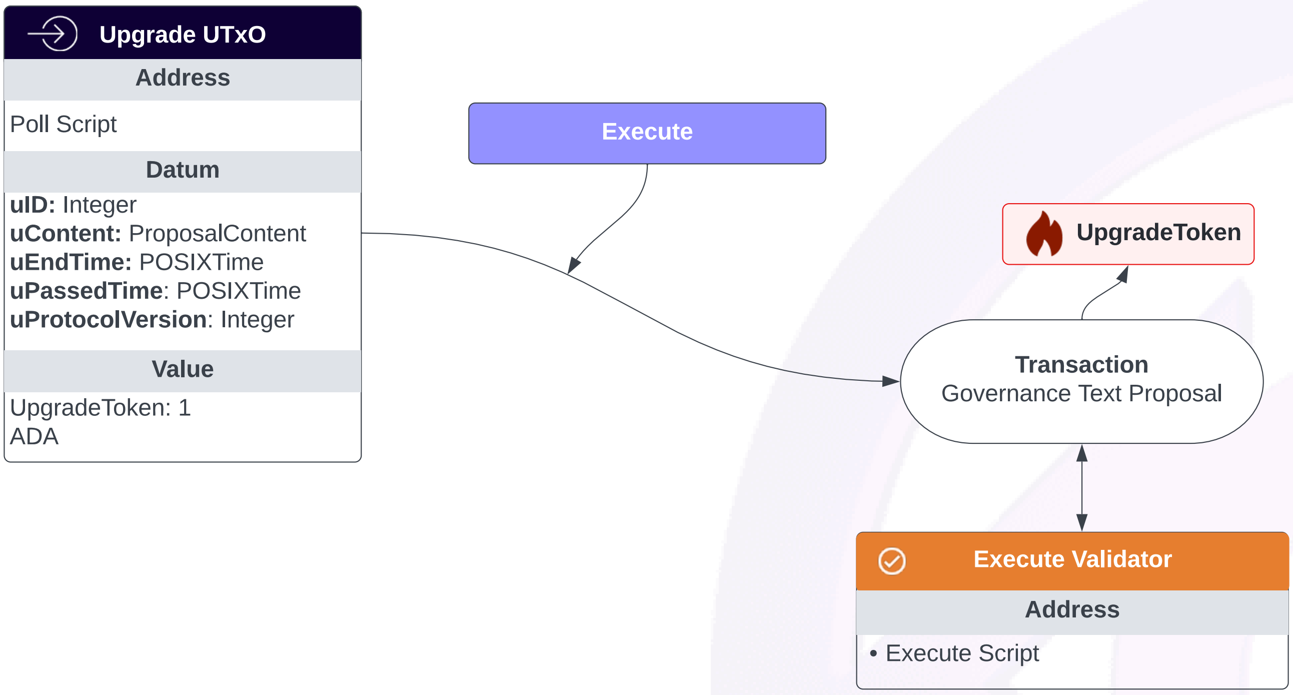


Figure 52: Example of a user executing their passed text proposal and retrieving their original 2 ADA deposit

Governance: Execute Propose Asset Execute a passed proposal adopted by the DAO to whitelist a new iAsset

Type	Amount	Description
Redeemer	N.A.	Execute
Consume	1	Upgrade UTXO containing the Upgrade Token for the passed proposal
Burn	1	Upgrade Token
Mint	1	iAsset Token
Mint	1	Stability Pool Token
Output	1	iAsset UTXO representing the new whitelisted iAsset
Output	1	Stability Pool UTXO representing the Stability Pool for the new whitelisted iAsset

Governance: Migrate Asset Execute a passed proposal adopted by the DAO to update an existing iAsset

Type	Amount	Description
Redeemer	N.A.	Execute
Redeemer	N.A.	UpgradeAsset

Type	Amount	Description
Consume	1	Upgrade UTXO containing the Upgrade Token for the passed proposal
Consume	1	iAsset UTXO representing the iAsset to update
Burn	1	Upgrade Token
Output	1	iAsset UTXO representing the updated iAsset

3.5 Collector

The Collector contract is an intermediary contract between protocol fee collection and distribution. The collection of funds can occur by sending funds directly to the Collector, or consuming an existing Collector and the output being more funds than were input. To distribute the funds, the Staking Manager can consume a Collector UTXO and use it to send funds to INDY stakers.

3.5.1 Parameters

- `stakingManagerNFT :: StakingManagerNFT`. NFT of StakingManager.
- `stakingToken :: StakingToken`. Token for identifying authentic Staking Position output.
- `versionRecordToken :: VersionRecordToken`. Token identifying the VersionRegistry output

3.5.2 Collector Endpoints

Collector: Collect Collect Protocol Fees upon withdrawing a CDP's collateral or closing a CDP

Type	Amount	Description
Redeemer	N.A.	Collect
Consume	1	Collector UTXO which may already contain previously collected protocol fees
Consume	1	CDP UTXO that represents a user's CDP
Output	1	Collector UTXO updated with the collected fee

3.6 Treasury

The purpose of this contract is to hold the DAO Treasury funds. The DAO Treasury will contain INDY that's vested over time according to Indigo's tokenomics model. The funds in the DAO Treasury are intended to be spent to help further develop, maintain and enhance the Indigo Protocol for the betterment of its users and INDY holders. The Treasury can be spent by creating a DAO proposal that includes a TreasuryWithdrawal value.

3.6.1 Parameters

- `versionRecordToken :: VersionRecordToken`. Token identifying the VersionRegistry output.
- `upgradeToken :: UpgradeToken`. Token identifying an Upgrade output.
- `treasuryUtxosStakeCredential :: Option<StakeCredential>`. An optional Stake Credential to attach to Treasury UTxOs.

Table 48: Treasury outputs

Type	Description	Values
Treasury	This output stores the DAO Treasury tokens	*INDY*: The INDY stored in the Treasury *IdentityToken*: 1

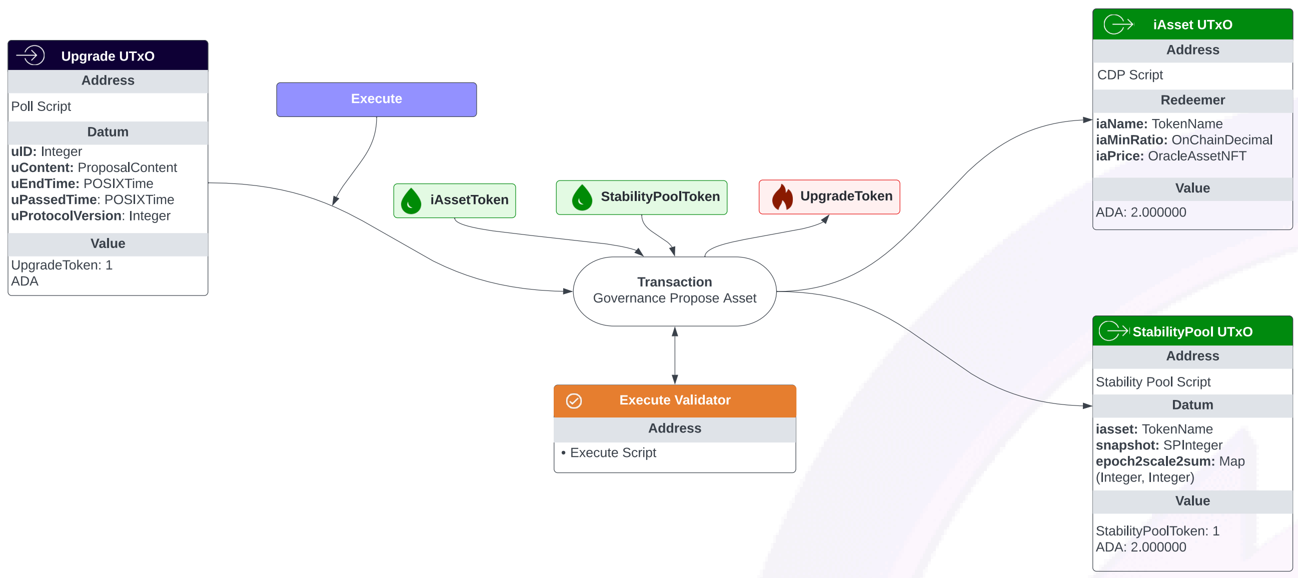


Figure 53: Example of a user executing their passed whitelist iAsset proposal, enabling a new iAsset within the protocol, creating a new Stability Pool, and retrieving their original 2 ADA deposit

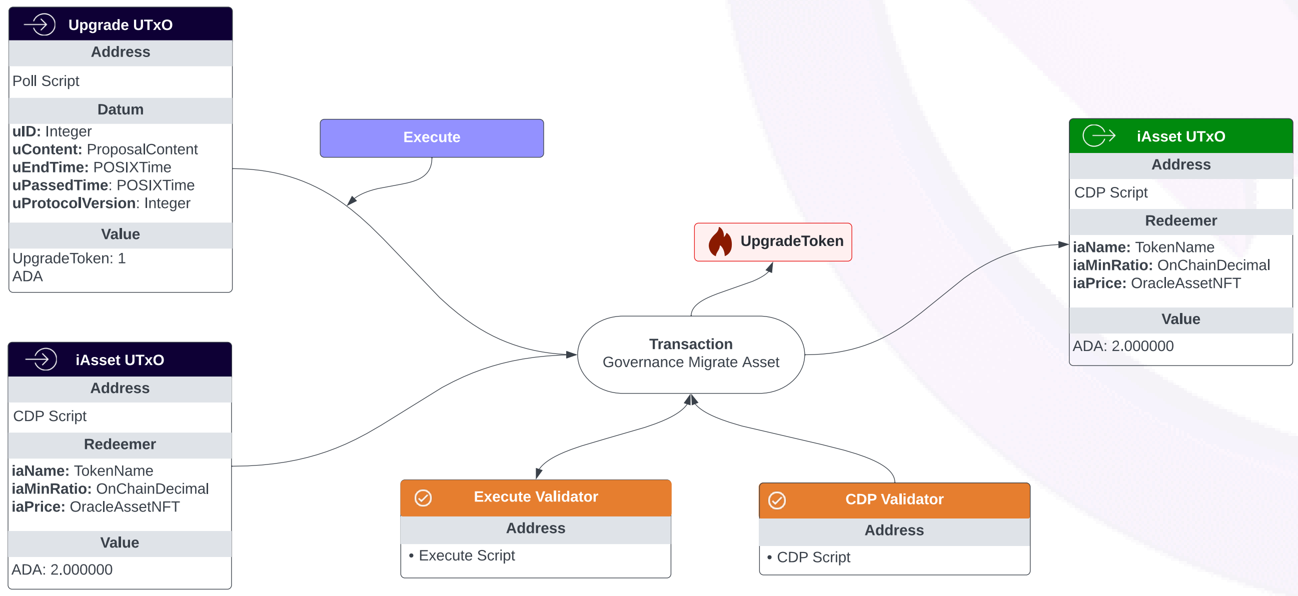


Figure 54: Example of a user executing their passed proposal to update an iAsset, and retrieving their original 2 ADA deposit

3.6.2 Treasury Endpoints

Treasury: Split Ensures that it is possible to have a structure of 1 Asset ¡–¿ 1 UTxO. This is useful to separate out assets cramped into a single UTxO (e.g. as a result of donations or failed multi-assets withdrawals) and increase their availability.

Type	Amount	Description
Redeemer	N.A.	Split, Treasury Input
Consume	1+	Treasury UTXOs with lovelaces and Multi-Assets
Output	1	Treasury UTXO with the lovelaces of the treasury input
Output	1+	Treasury UTXO with each Multi-Asset type

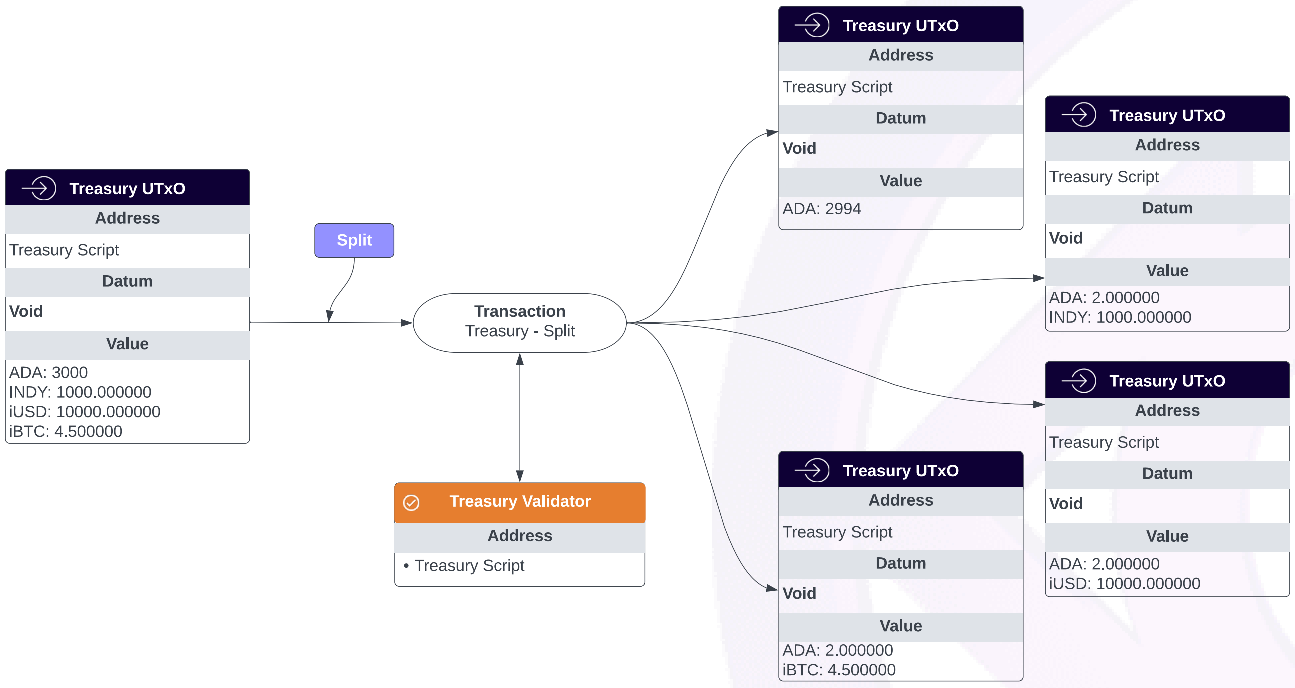


Figure 55: Example of a user executing their passed proposal to update an iAsset, and retrieving their original 2 ADA deposit

Treasury: Merge Defragments a single asset spread across multiple UTXOs locked in the treasury script (INDY from failed proposals and other tokens from strategic movements such as LP tokens or donations).

Type	Amount	Description
Redeemer	N.A.	Merge, Treasury Input
Consume	1+	Treasury UTXOs with lovelaces and Multi-Assets
Output	1	Treasury UTXO with all the value of the merged Treasury Inputs

Treasury: Prepare Withdrawal To prepare a single UTxO to be spent from the treasury following a request voted and passed by the DAO. This is to make sure that multi-asset withdrawals can be processed.

Type	Amount	Description
Redeemer	N.A.	PrepareWithdrawal, Treasury Inputs
Consume	1+	Treasury UTXOs with lovelaces and Multi-Assets

Type	Amount	Description
Reference	1	Upgrade UTXO with the expected withdrawal value
Output	1	Treasury UTXO with all the value of intended Withdrawal
Output	1	Treasury UTXO with all the change

4 Known Protocol Limitations

4.1 Governance Contention

Users can deposit their INDY into the protocol to become Members and gain access to privileges such as voting rights and reward collection. The Staking Manager UTXO is responsible for managing staking positions of users. The Staking Manager must be updated upon the following actions:

1. Create a staking position (i.e., deposit INDY into governance)
2. Adjust a staking position (i.e., deposit INDY into or withdraw INDY from governance)
3. Deposit staking reward (i.e., collect an ADA protocol fee)
4. Withdraw staking reward (i.e., redeem ADA reward for staking)

Interacting with the Staking Manager causes contention because only one update can be made per block. As a mitigation effort, the Collector can bundle staking rewards collected by the protocol to reduce the number of staking reward transactions deposited into the Staking Manager. Contention still exists for INDY stakers depositing or withdrawing INDY or withdrawing ADA rewards.

Additionally, contention exists for recording governance votes. To improve scalability, votes are recorded using individual shards. Users can pick unused shards to record their votes. While, theoretically, an unlimited number of shards can be configured, the Cardano blockchain is limited in the number of shards that can record votes per block. If there are insufficient available shards, then users will have to wait for a shard to become available before voting.

Shard collision can occur when two or more users select the same shard to vote with; only one user will succeed with recording the vote, the other users using the same shard will experience transaction errors. If a user explicitly checks for shard availability before submitting a transaction, another user may also select that same shard before the transaction is processed in a block, thereby possibly resulting in collision and transaction failure for either user.

5 Glossary

Term	Description
Debt Minting Fee	Paid when iAsset is minted, benefits INDY stakers and varies according to market conditions, affecting debt minting volumes.
Redemption INDY Staker Fee	Imposed on redemption's, paid by the redeemer to INDY Staker's.
Redemption Reimbursement Fee	Imposed on redemption's, paid by the redeemer back to the redeemed Collateralized Debt Position.
Stability Pool Fee	A fee that is taken whenever a Stability Pool Provider withdraws iAssets from the Stability Pool. Those iAssets are distributed across the remaining Stability Pool Providers.
CDP Collateral Fee	A fee applies with withdrawing collateral from your CDP or upon closure. This fee is distributed to INDY Stakers.

6 Definitions for Mathematical Notations

Throughout this document, references are made to mathematical equations. Below is a summary of notations that may be used and their associated meanings.

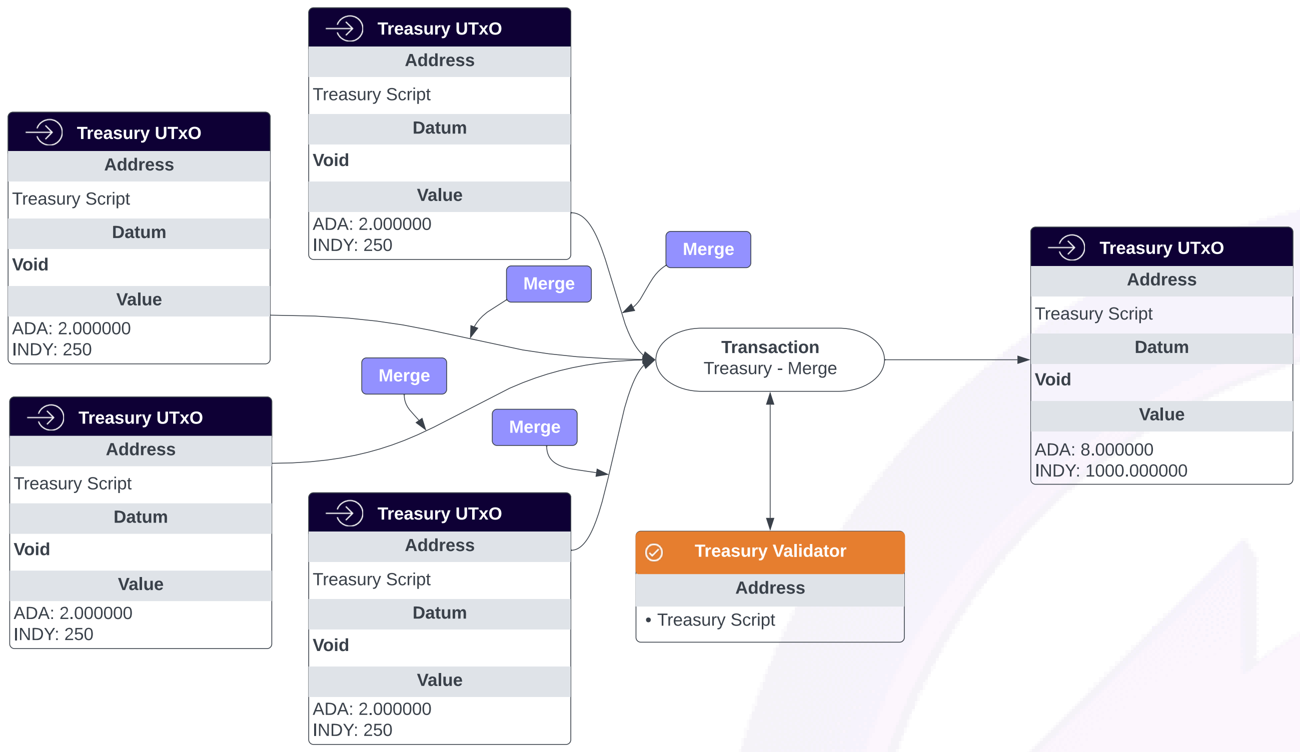


Figure 56: Example of a ADA, INDY, iUSD, iBTC Treasury UTxO being merged into a single UTxO

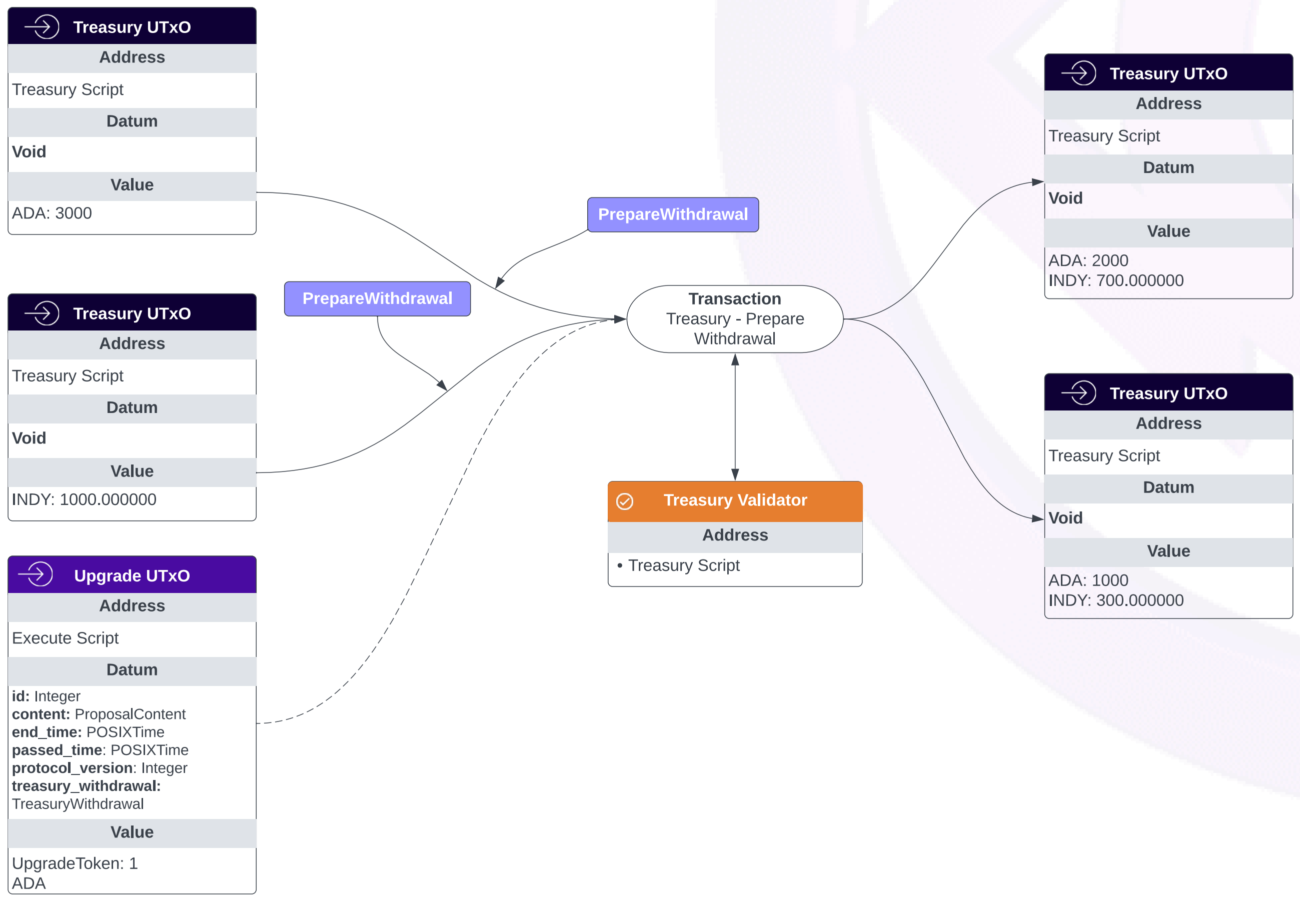


Figure 57: Example of a ADA and INDY Treasury UTxO being merged into a single UTxO based on a governance proposal withdrawal of 300 INDY and 1000 ADA

6.1 Sets

Values enclosed in { } are a unique assortment of values. Each value is separated by a comma (,).
$\{10, 20, 30, 40, 50\}$ means five values incrementing in tens, beginning at 10 and ending at 50.

6.2 Summation

The $\sum$ represents a sum of values. It can either be in the form of $\sum x$ or $\sum_{i=1}^{n} i$.
$\sum x$ means to sum all values of a set. If x is a set of $\{1, 2, 3\}$, then:

$$\sum x = 1 + 2 + 3 = 6$$

$\sum_{i=1}^{n} i$ means to iterate n times and sum the result of x. i begins at 1 and increments until i equals n. If n is 3, then:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 = 6$$

6.3 Length of Sets

A set enclosed within | | represents the length of the set.
$|x|$ means the length of set x. If x is $\{5, 10, 15\}$, then $|x| = 3$ because it contains 3 elements in the set.

6.4 Indexes

A subscript (x_i) represents an associated variable or a value within a set.
If x is a set and i is a number, then x_i means the i^{th} element of the set x. If x is $\{3, 6, 9\}$, then x_1 is 3, x_2 is 6, and x_3 is 9. Thus, if i is 2, then x_i is 6 because it's the 2^{nd} element of x.

6.5 Mean of Sets

A set with (a bar) above it represents the mean (average) of the set.
$\overline{x}$ means the mean of set x, which is the sum of all elements in the set divided by the length of the set. Alternatively, $\overline{x}$ can be expressed as:

$$\frac{\sum_{i=1}^{|x|} x_i}{|x|}$$

If x is $\{10, 30, 20, 40\}$, then:

$$\overline{x} = \frac{10 + 30 + 20 + 40}{4} = 25$$

6.6 Rounding

Values enclosed in $\lceil\ \rceil$ or $\lfloor\ \rfloor$ represent the value either rounded up or down to the nearest whole number.
$\lceil x \rceil$ means "ceil," or to round up to the nearest whole number. If x is 0.5, then: $\lceil x \rceil = 1$.
$\lfloor x \rfloor$ means "floor," or to round down to the nearest whole number. If x is 0.5, then: $\lfloor x \rfloor = 0$.

6.7 Scoped Variables

Sometimes equations may be simplified and made more readable using scoped variables.
$x = \begin{pmatrix} \text{let } y \text{ equal } 1 \\ y \end{pmatrix}$ means to create a variable called y with a value of 1, which can then be referenced throughout any component within the ( ) it's defined within. Therefore, x is 1 because the bottom-most statement is y and y is 1.

6.8 Conditional Statements

A statement proceeding $\{$ without an enclosing $\}$ is conditional. Conditional statements take the form of $\begin{cases} x & \text{if } a > 0 \\ y & \text{otherwise} \end{cases}$. They can have two or more conditions, such as: $\begin{cases} x & \text{if } a > 0 \text{ and } a < 1 \\ y & \text{if } a > 50 \\ z & \text{otherwise} \end{cases}$.

$\begin{cases} x & \text{if } a > 0 \text{ and } a < 1 \\ y & \text{if } a \geq 1 \\ z & \text{otherwise} \end{cases}$ means that the value is determined by the truthfulness of three conditions. If a is between 0 and 1, then the statement is x. If a is 1 or larger, then the statement is y. The only other possibility is a is 0 or smaller, in which case the statement is z.

"otherwise" means if no other condition matches.

"and" means that both conditions must be true.

"or" means that either condition must be true.

6.9 Functions

Statements proceeding $f : (\ ) \mapsto$ represent a callable function that can be referenced.

$f : (a, b) \mapsto a + b$ means that f takes two values and adds them together to determine the value. A reference to $f\,(1, 2)$ equates to 3.

6.10 Minimum and Maximums

Minimum and maximum values within sets can be referenced using $min\,\{\ \}$ and $max\,\{\ \}$ respectively.

$min\,\{100, 10, 1000\}$ means the lowest value out of the set $\{100, 10, 1000\}$, therefore: 10.

$max\,\{100, 10, 1000\}$ means the highest value out of the set $\{100, 10, 1000\}$, therefore: 1000.

7 Minimum ADA to Create UTXO

To create a UTXO on Cardano, a minimum amount of ADA is required to be locked into the transaction. The amount of ADA deposit required to create a UTXO is calculated using the formula:

$$x = ab + 160b$$

Where:

- x is the amount of ADA required to create a shard
- a is the size of the UTXO of the transaction
- b is the *coinsPerUTxOByte* parameter of the Cardano blockchain[15]

Upon closing the UTXO, the deposited ADA can be unlocked.

[15] Calculating required ADA for UTXOs is described by the UTXO inference rules on page 16 of the formal Cardano specification.