Platform Architecture

By AdaLink

Table Of Content

Introduction

This chapter serves as an overarching introduction to the protocols and architecture discussed in subsequent chapters. It outlines the foundational concepts and objectives that underpin the AdaLink platform's design and operation.

Platform Overview

AdaLink is a groundbreaking initiative designed to forge connections and foster collaborations within the vibrant Cardano ecosystem. Serving as a nexus for Cardano businesses, influencers, NFT founders, marketers, and large capital delegators “i.e. whales”, AdaLink aims to revolutionize how partnerships are formed and sustained within the blockchain space.

Motivation

AdaLink aims to create an ecosystem where stake pool operators “i.e. SPOs” incentivize affiliates to attract new stakeholders to their pools. The primary motivation is to foster growth and participation within Cardano by rewarding affiliates who contribute to the success of stake pools.

Key Objectives

1. **Transparency and Fairness**: Ensure that reward distribution mechanisms are transparent and fair to all participants, including SPOs and affiliates.
2. **Trustlessness**: Minimize the need for trust between parties by leveraging smart contracts and automated protocols.
3. **Automation**: Streamline processes through automation to reduce manual intervention and increase efficiency.
4. **Security**: Implement robust security measures to protect the integrity of the platform and user data.

With these objectives in mind, AdaLink has developed a series of protocols and architectural components to achieve its goals. The following chapters will delve into the specifics of these protocols, including reward calculation, distribution, database management and smart contract design and implementation.

Reward Calculation Protocol

Introduction

This Chapter goes over the protocol responsible to calculate the rewards of all participating affiliates.

Protocol Overview

This protocol is the engine responsible for calculating the reward of each affiliate for participating in an incentive program and successfully bringing ADA to the stake pool. At the end of each epoch, the protocol query the blockchain and sort out the wallets delegating to SPOs with an active incentive campaign. Each delegated wallet gets associated with a registered affiliate. This is done by tracking the delegation transaction and confirming that the wallet delegated to the stake pool through a transaction having a special signature belongs to a registered affiliate, as will be seen in the next section.

Protocol Components

Affiliate ID

Every affiliate gets assigned a unique ID code by AdaLink when they register to the platform. This ID will be used in the next component, “Affiliate Transactions”, to associate the wallets that delegates through their affiliate links to them.

Affiliate Transactions

AdaLink offers API for registered affiliates that build them a delegation transaction that includes the affiliate ID within the first utxo’s datum of the transaction. These API calls/queries are the affiliate links that they offer to their followers to use in order to delegate to the SPO or one of the SPOs they enrolled with a program with.

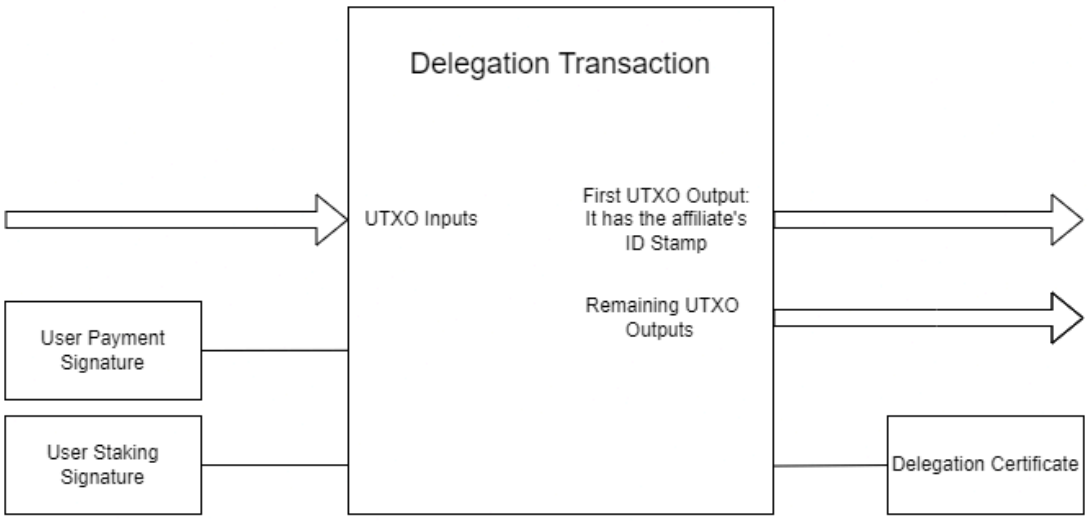

Figure 2-1: Affiliate Transaction Schematic.

Wallets Analyzer

At the end of each epoch an automated script in the backend of AdaLink initiates. It gets all the current active incentive programs from the database. For each program, it gets the stake pool of the program and query the blockchain for all delegated wallets. It goes through each wallet and query the blockchain for the delegated transaction (each wallet has to submit a delegation transaction when it desires to stake with a specific stake pool). Then, it checks the first utxo's datum and searches for an affiliate ID stamp. If found, it associates that delegation to the affiliate owning the unique ID, and adds the stake brought by that affiliate to the stake pool. At the end of each pool's list of delegated wallets, there will be a list of all participating affiliates and their corresponding brought stake.

Reward Calculator

The reward calculator sorts out the findings of the wallet analyzer component. For each incentive program, the percentage of each affiliate contribution to the overall stake is calculated and multiplied by the "maximum reward per epoch"[1] parameter. Lastly, each program will have a list of the corresponding affiliates with the reward of each one calculated.

[1] This parameter is defined in chapter 4 in detail.

Protocol Logic Flow

1. Initialization
    - At the end of each epoch, the protocol initialization begins, triggered by an automated script in the backend of AdaLink.
2. Incentive Programs Retrieval
    - The script retrieves all current active incentive programs from the database of AdaLink. Each program contains information about the associated stake pool and the maximum reward per epoch.
3. Delegated Wallets Query
    - For each incentive program, the script queries the blockchain to obtain the list of wallets that have been delegated to the corresponding stake pool.
4. Delegation Transaction Analysis
    - The protocol iterates through each delegated wallet and retrieves the delegation transaction associated with it.
    - It checks the first utxo's datum of each transaction to search for an affiliate ID stamp. If found, it associates that delegation with the affiliate owning the unique ID and adds the stake amount of the wallet to the overall brought stake by that affiliate to the stake pool.
5. Stake Accumulation
    - As the protocol progresses through the list of delegated wallets for each program, it accumulates the stake brought by each participating affiliate in the associated stake pool.
6. Reward Calculation
    - Once all delegated wallets have been analyzed, the reward calculator component begins.
    - For each incentive program, the percentage of each affiliate's contribution to the overall stake is calculated based on their accumulated brought stake.
    - This percentage is then multiplied by the maximum reward per epoch parameter to determine the reward amount for each affiliate.
7. Reward Assignment
    - The protocol assigns the calculated rewards to each affiliate participating in the incentive program.
    - Each program maintains a list of affiliates along with their corresponding reward amounts.

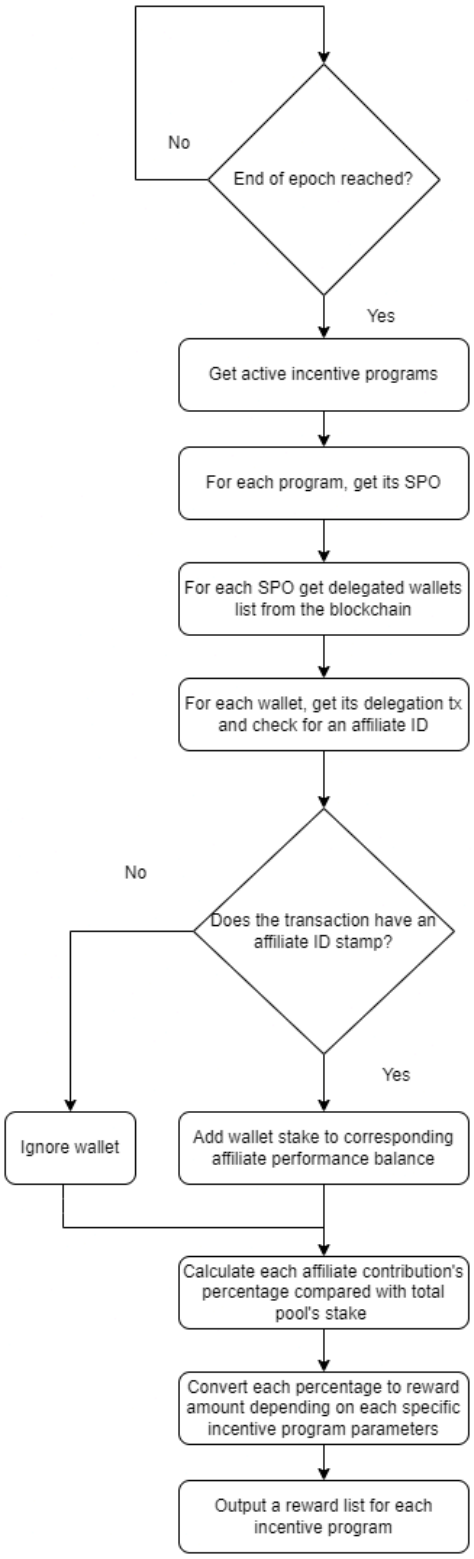

Figure 2-2: Flowchart of the reward distribution protocol.

Conclusion

The Reward Calculation Protocol serves as a vital component in the operation of incentive programs within the AdaLink platform. By systematically analyzing delegated wallets, associating them with registered affiliates, and calculating rewards based on stake contributions, the protocol ensures fair and transparent reward distribution. Through its automated execution at the end of each epoch, the protocol streamlines the reward calculation process, providing SPOs and affiliates with timely and accurate rewards for their participation in the ecosystem. Moving forward, continued optimization and refinement of the protocol will enhance its effectiveness in supporting the growth and sustainability of the AdaLink platform.

Selecting a Reward Distribution Protocol

Introduction

This chapter discusses several potential approaches to distribute the rewards to all affiliates from their corresponding SPOs. Each method has its own advantages and disadvantages that will be mentioned in detail. The main objective is to select one of the methods to be the adopted protocol for reward distribution.

Motivation

AdaLink's platform requires a reward distribution protocol, in which SPOs can incentivise other parties, referred to as affiliates, to spread awareness and market their stake pool to potential stakers, both in and outside the Cardano ecosystem. These rewards are in the form of ADA coins. Each affiliate is awarded proportionally to their contribution to the overall stake in the pool. The absolute amount of the reward is offered by the SPOs, each can allocate an arbitrary amount of ADA each epoch to be divided among the corresponding affiliates. Hence, the protocol has to satisfy these key points:

- Let SPOs allocate rewards for a given number of epochs.
- Let affiliates view SPOs incentives (reward amount per ADA staked).

Additionally, the protocol has to take into consideration the following quality measurements:

- Security for all parties (SPOs, affiliates and AdaLink).
- Convenient for all parties.
- Trustlessness: reduce the trust needed from any party as much as possible.
- Automation: reduce the manual interaction of any party after the initialization phase as much as possible.

Distribution Protocols

In this section three different approaches will be analyzed.

One Shot - SPO Controlled

This might be the simplest approach from a technical point of view. The algorithm of this protocol is as follows:

- SPOs launch new incentive programs at will.
- Later on, at the end of each epoch:
    - Get the SPO list for all registered SPOs in AdaLink platform.
    - Filter the ones who have an active reward program for affiliates.
    - For each SPO:
        - get the reward amount of their affiliates and order them in a list.
        - Construct a spending transaction that includes all affiliates with their respective reward amount.
        - Notify the SPO on the platform a reward payout is due.
        - Await for the SPO to confirm and sign the transaction.

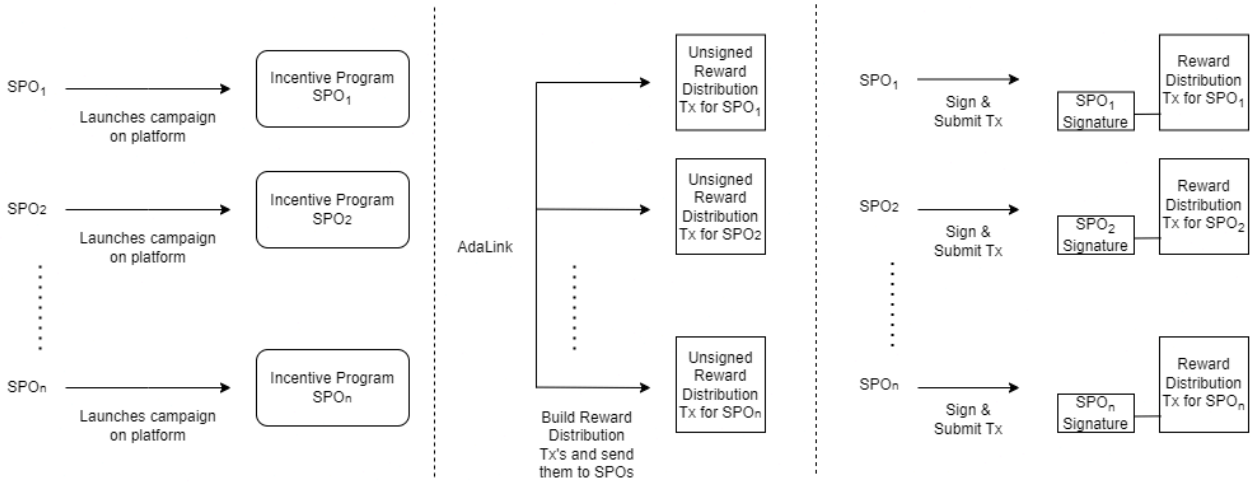

Figure 3-1: “One Shot - SPO Controlled” Method Schematic.

Advantages

- Simplicity: With a simple and straight forward approach there will be hardly any potential breaches in the protocol.
- SPOs do not need to deposit any amount of ADA beforehand.

Disadvantages

- Limited automated execution: Although AdaLink does all the heavy lifting needed to organize and put together all necessary parts, Each SPO has to manually go and manually sign the reward distribution transaction every epoch.
- Limited trustlessness: While SPOs don't have any issues in this approach regarding trusting any party (they can always verify before signing any transaction), affiliates and AdaLink have to trust on SPOs to sign and submit the transactions.

Two Stage - Spending Tx's

In this approach, the protocol asks the SPOs to lock in an amount of ADA corresponding to the amount of needed rewards for a given number of epochs. This deposit is controlled by AdaLink and is used to distribute the rewards on affiliates.The algorithm of this protocol is as follows:

- SPOs launch new incentive programs at will.
- SPOs lock in the maximum needed amount of ADA to fulfill a successful campaign.
- Later on, at the end of each epoch:
    - Get the SPO list for all registered SPOs in AdaLink platform.
    - Filter the ones who have an active reward program for affiliates.
    - For each SPO:
        - Get the reward amount of their affiliates and order them in a list.
        - Construct a spending transaction that includes all affiliates with their respective reward amount.
        - Automatically sign the transaction by AdaLink.
        - Notify the SPO on the platform a reward payout was distributed.

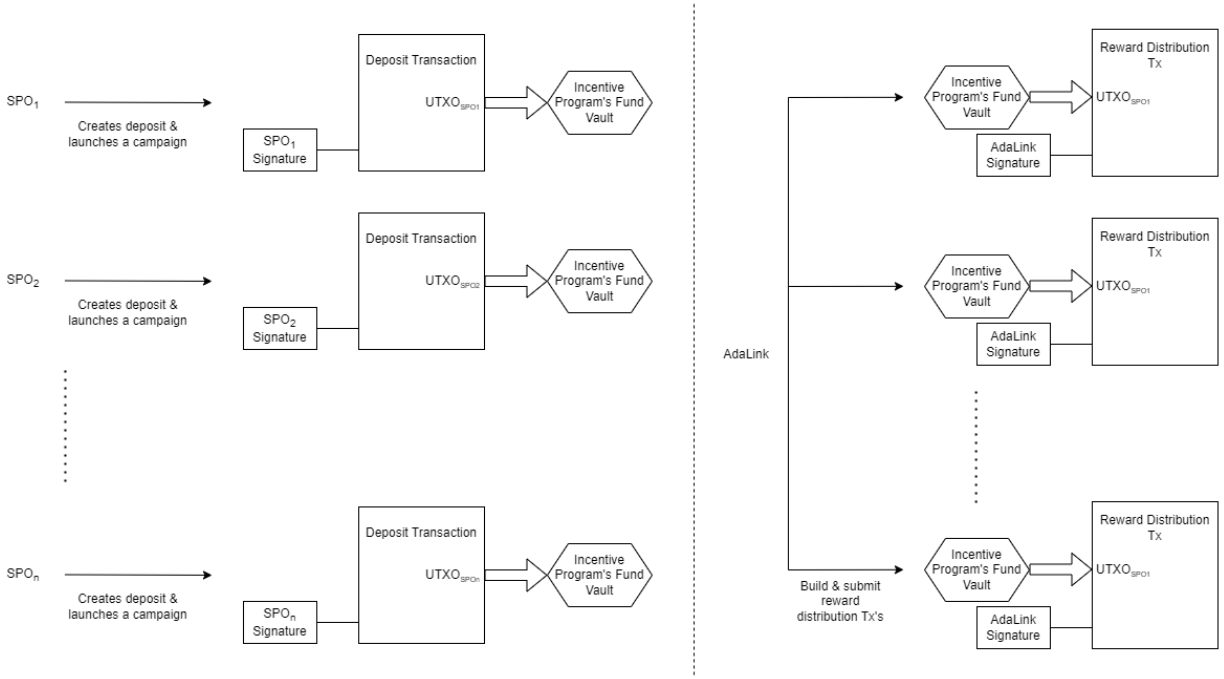

Figure 3-2: "Two Stage - Spending Tx's" Method Schematic.

Advantages

- Simplicity: With a simple approach there will be hardly any potential breaches in the protocol.
- Affiliates and AdaLink do not have to trust SPOs to deliver the rewards.
- Automated protocol: Each epoch there will be an automated reward distribution to all ongoing incentive programs.

Disadvantages

- SPOs have to trust AdaLink to rightfully distribute the rewards as intended.

Three Stage - Automated with Smart Contract Integration

In this method, the protocol asks the SPOs to lock in an amount of ADA corresponding to the amount of needed rewards for a given number of epochs. This deposit is put inside of a smart contract. This SC allows AdaLink to withdraw one epoch worth of maximum rewards once and only once an epoch. This portion is then relayed by AdaLink to corresponding affiliates via a spending transaction.The algorithm of this protocol is as follows:

- SPOs launch new incentive programs at will.
- SPOs lock in the maximum needed amount of ADA inside a smart contract to fulfill a successful campaign launch.
- Later on, at the end of each epoch:
    - Get the SPO list for all registered SPOs in AdaLink platform.
    - Filter the ones who have an active reward program for affiliates.
    - For each SPO:
        - Get the reward amount of their affiliates and order them in a list.
        - Withdraw maximum rewards per epoch from the smart contract.
        - Construct a spending transaction that includes all affiliates with their respective reward amount plus change back to the SPO if available.
        - Automatically sign the transaction by AdaLink.
        - Notify the SPO and their affiliates on the platform a reward payout was distributed.

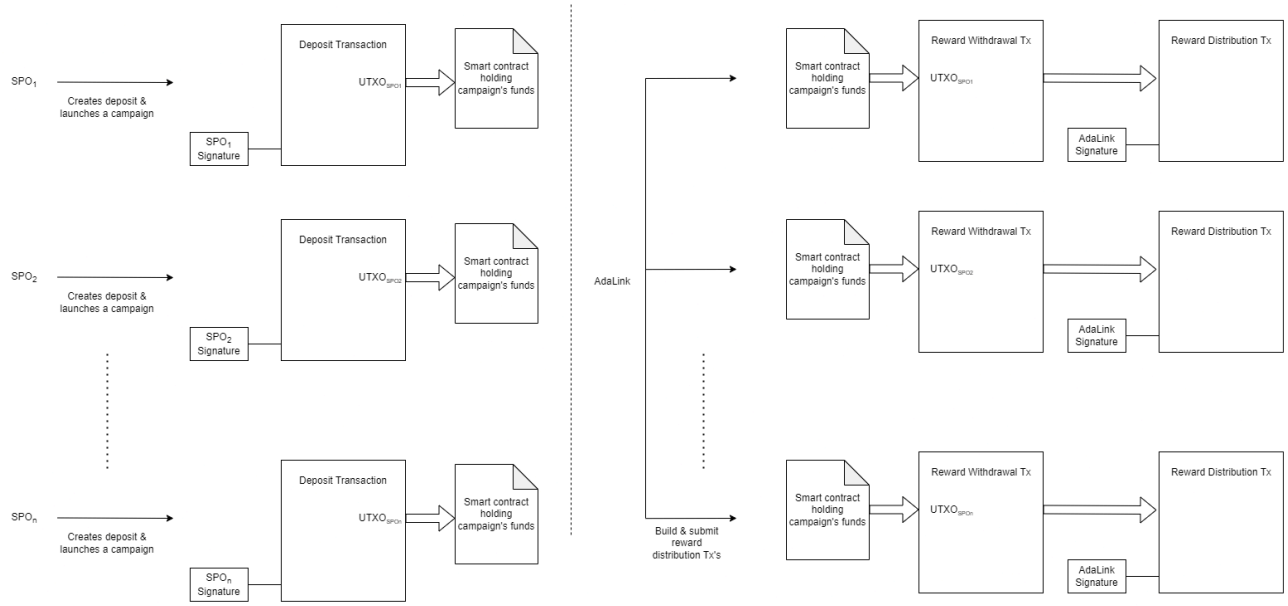

Figure 3-3: “Three Stage - Automated with Smart Contract” Method Schematic.

Advantages

- Affiliates and AdaLink do not have to trust SPOs to deliver the rewards.
- SPOs do not have to trust AdaLink on the deposited amount, as they can only take one epoch worth of rewards in ADA.
- Automated protocol: Each epoch there will be an automated reward distribution to all ongoing incentive programs.

Disadvantages

- Complexity: The overall design has several crucial parts. Such design needs to be well built and audited thoroughly to make sure it functions exactly as intended.

Selecting a Protocol for Reward Distribution

Comparing the pros and cons of each approach discussed above and using the requirements and considerations mentioned in the Motivation section. One can argue that the third method has the highest trustlessness, convenience and automation. Additionally, with careful design and thorough analysis, both security and robustness can as well be on the same level with the other two methods.

Reward Distribution Protocol

Introduction

This chapter goes over the protocol responsible to distribute the rewards among affiliates from their corresponding SPOs.

Protocol Overview

Paraphrase the following paragraph and make it as an overview for the protocol:
The protocol asks the SPOs to lock in an amount of ADA corresponding to the amount of needed rewards for a given number of epochs. This deposit is put inside of a smart contract. This SC allows AdaLink to withdraw one epoch worth of maximum rewards once and only once an epoch. This portion is then relayed by AdaLink to corresponding affiliates via a spending transaction.

Protocol Components

Incentive Program

A campaign launched by an SPO to encourage potential affiliates to help bring new stake to the stake pool. Each program is defined by four main parameters:

- Stake Pool: This is the stake pool running the campaign.
- Length of Campaign: Number of epochs the program is going to be active.
- Maximum Rewards per Epoch: An upper limit on the amount of ADA that is available per epoch. This amount corresponds to or equal to a reward if an affiliate or group of affiliates brings enough stake to the pool to reach 100% saturation from the initial saturation percentage at the beginning of the campaign.
- Reward in ADA per Brought ADA: The maximum reward per epoch parameter will be automatically converted to how much the stake pool will be rewarding affiliates in ADA for each brought ADA to the pool per epoch until end of campaign.

Example

Assuming current epoch is 469, and a stake pool with ticker WAPA that has a current saturation level of 30% (approximately 22 million ADA). The SPO of WAPA launches a new incentive program and desires the campaign to last 10 epochs starting from next epoch. Additionally, they set a maximum reward of 1000 ADA per epoch. The following are the parameters of the campaign:

- Stake Pool: WAPA (This is automatically entered by the platform)
- Duration: 10 Epochs [470-479]
- Maximum Rewards per Epoch: 1000 ADA for bringing 22 million ADA staked to 72 million ADA staked (amount at 100% saturation).
- Reward in ADA per Brought ADA: Mathematically, 1K/(72M-22M)=0.00002 ADA/Brought ADA or 20 Lovelace for each new ADA brought to the pool.

Continuing with the same example, assume an affiliate named Gisela joins the campaign during epoch 472, and she managed to bring 1,200,000 ADA from her followers base before epoch's end. She will receive 0.00002 ADA x 1.2M = 24 ADA per epoch from start of epoch 473 until start of epoch 480, given that the 1.2M remain staked during this period. Totalling 8 payments of 24 ADA each. Raising the brought ADA amount or lowering it will be reflected on next epoch's reward.

Reward Deposit Smart Contract

To eliminate the need to trust stake pools to provide the rewards they offered for their campaign, and simultaneously reduce the risk for SPOs by not needing to completely put the rewards in the hands of AdaLink, a smart contract sits between the two parties. This smart contract is designed to receive ADA from the SPO as a deposit. It allows only one wallet key, owned by AdaLink, to withdraw one epoch worth of rewards once every epoch.

Smart Contract Datum

Datum is the data attached when sending funds to a smart contract. In this context, the datum will include the incentive program parameters, namely, total deposit amount, maximum rewards per epoch, start and end epochs.

Smart Contract Script

The main logic running the smart contract. It will use the datum to validate any withdrawal actions. Only and only if the following points are satisfied the script gets validated:

1) Is the current epoch the same as the last epoch in the campaign?
    a) Yes: Number of outputs must be one and only one, and the receiving address is the known AdaLink address.
    b) No: Number of outputs must be two and only two. First one is the AdaLink address and its ADA amount is equal to the maximum rewards per epoch set from the datum. Second output is the script's own address.

2) Is end epoch minus current epoch times maximum allowed rewards per epoch equal to current ADA in the smart contract UTXO? This line of logic guarantees that only one withdrawal per epoch is allowed.

Reward Distributor Engine

AdaLink's backend contains a full running node. This engine is a group of scripts running in the backend. Its primary goal is to distribute the calculated rewards on affiliates at the end of every epoch. The engine itself consists of 3 subparts, organizer, submitter and notifier.

Organizer

Responsible for grouping each active incentive program with its SPO and corresponding affiliates. It is also in its duties to acquire the reward amount for each affiliate (from the Reward Calculation Protocol).

Submitter

Constructs a spending transaction for each incentive program that uses the withdrawn reward amount from the deposit smart contract as an input and distributes the reward of each affiliate as outputs.Then sign the transactions and submit them to the blockchain.

Notifier

Once the distribution transactions are confirmed, the notifier updates the database of the AdaLink's platform to notify users (SPO and affiliates) that a reward payout took place.

Protocol Logic Flow

1) Incentive Program Launching: A registered SPO starts a new incentive program on AdaLink's platform. The SPO launches a new incentive program and specifies its parameters. These parameters are:
    a) Length of program: Number of epochs the program is active.
    b) Maximum amount of rewards per epoch: Amount of ADA corresponds to the maximum amount of rewards to be distributed if the stake brought by affiliates results in 100% pool saturation.
2) Reward Depositing: To confirm the incentive program, the SPO is required to deposit the total amount of rewards of the program into the "Reward Deposit Smart Contract".

3) Epoch End Event: This is run automatically in the backend of AdaLink's servers at the end of each epoch.
    a) Run The Reward Calculation Protocol: This is discussed in detail in the "Reward Calculation Protocol" chapter. It results in calculating the ADA reward for each affiliate and the amount of ADA each SPO needs to distribute to their corresponding affiliates.
    b) Reward Distribution: For each SPO with an active incentive program:
        i) Get the reward amount of each affiliate participating in the program. This is available data from the "Reward Calculation Protocol".
        ii) Withdraw maximum rewards per epoch from the "Reward Deposit" smart contract.
        iii) Construct a spending transaction that includes all affiliates with their respective reward amount plus change back to the SPO if available.
        iv) Automatically sign the transaction and submit it to the chain.
        v) Notify the SPO and their affiliates on the platform a reward payout was distributed.

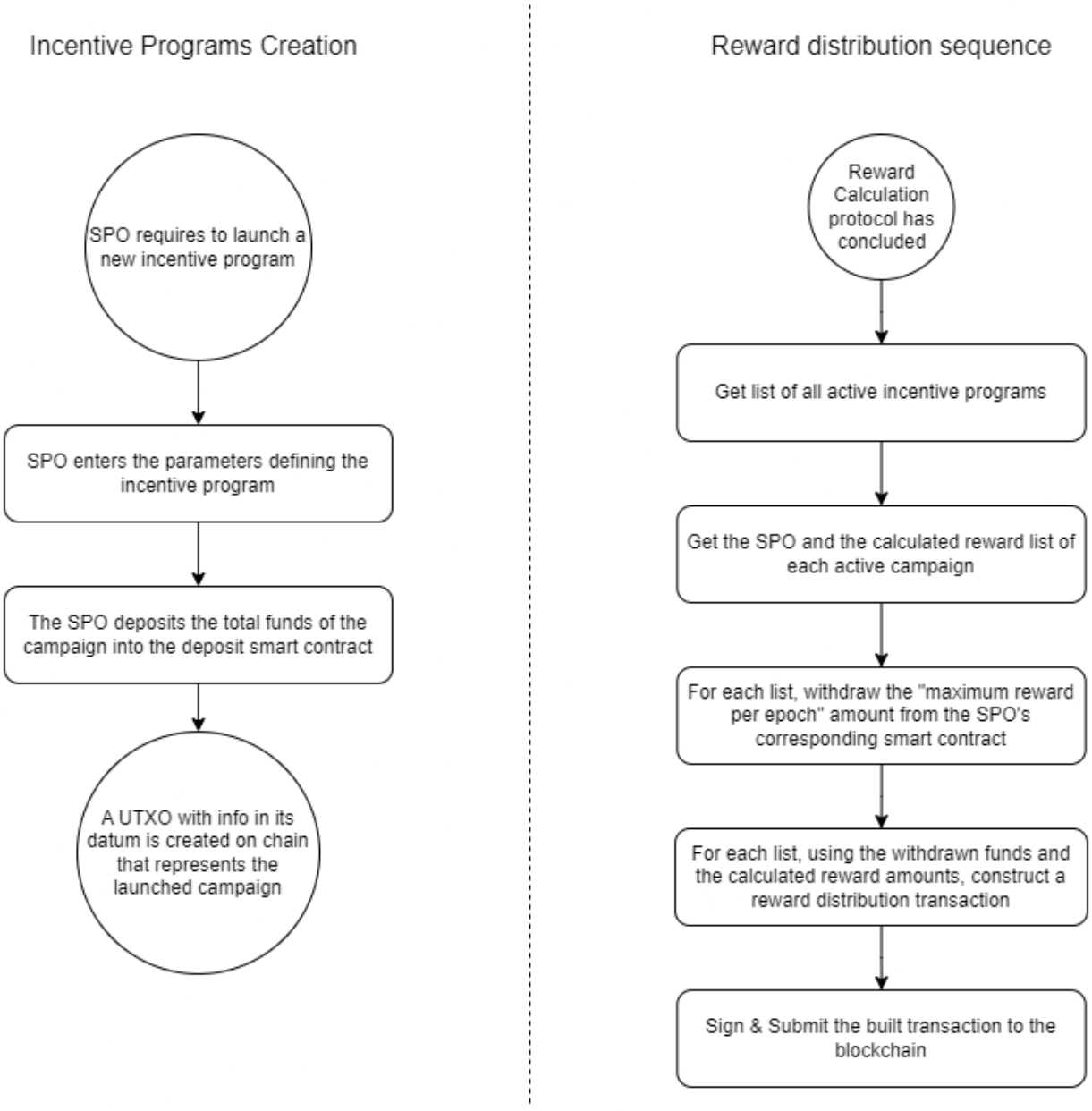

Figure 4-1: The flowchart of the Reward Distribution protocol.

Database Architecture and Data Handling

Introduction

This chapter discusses how AdaLink deals with all types of data collected, stored and produced by the platform. After understanding the main goal of the project and how the major protocols operate, one can now lay down the architecture of the database used to store and retrieve previously acquired data. Additionally, the different sources for querying data from the blockchain is discussed as well.

Database Architecture

The data used by the platform is divided into 3 categories. Namely, Stake pool's data, affiliates' data and incentive programs' data.

Stake Pools' Data

This database consists of a table that includes all related information about registered SPOs. This information is adequate to describe a stake pool to potential delegators or affiliates. The parameters and their data types are the following:

- Stake Pool's Ticker (Static): STRING data type.
- Stake Pool's Display Name (Static, but editable by user): STRING date type.
- Stake Pool's ID (Static): STRING data type.
- Stake Pool's Creation Date (Static): DATE data type.
- Stake Pool's Profile Picture (Static, but editable by user): URL data type.
- Stake Pool's Website (Optional): URL data type.
- Stake Pool's Social Link (Optional): URL data type.
- Stake Pool's Registered Wallet Address (Static): STRING data type.
- Stake Pool's Registered Wallet Stake Address (Static): STRING data type.
- Stake Pool's Active Stake (Dynamic, updates every epoch): BigInt data type.
- Stake Pool's Live Stake (Dynamic, updates every few minutes): BigInt data type.
- Stake Pool's Saturation (Dynamic, updates every few minutes): Float data type.
- Stake Pool's Fixed Fees (Dynamic, updates every few minutes): BigInt data type.
- Stake Pool's Margin (Dynamic, updates every few minutes): Float data type.
- Stake Pool's Recent ROA (Dynamic, updates every epoch): Float data type.
- Stake Pool's Lifetime ROA (Dynamic, updates every epoch): Float data type.
- Stake Pool's Blocks in Current Epoch (Dynamic, updates every few minutes): INT data type.
- Stake Pool's Blocks Lifetime (Dynamic, updates every epoch): INT data type.
- Stake Pool's Lifetime Luck (Dynamic, updates every epoch): Float data type.

- Stake Pool's Delegators (Dynamic, updates every few minutes): INT data type.
- Stake Pool's Pledge Leverage (Dynamic, updates every few minutes): Float data type.

Ticker	Display Name	ID	Pool Creation Date	.....	Current Epoch Blocks	Blocks Lifetime	Lifetime Luck	Delegators Count	Pledge Leverage

Figure 5-1: Illustration of the SPOs' table format within the platform database. Notice some data types have been omitted due to viewing space limitations.

Affiliates' Data

This database consists of a table that includes all related information about registered affiliates. This information is adequate to describe an affiliate for interested SPOs. The parameters and their data types are the following:

- Affiliate's Unique ID (Static): STRING data type.
- Affiliate's Display Name (Static): STRING data type.
- Affiliate's Type: Possible options are, influencer, web3 project, marketing agency (Static): STRING data type.
- Affiliate's Registered Wallet Address (Static): STRING data type.
- Affiliate's Registered Wallet Stake Address (Static): STRING data type.
- Affiliate's Profile Picture (Static, but editable by user): URL data type.
- Affiliate's Website (Optional): URL data type.
- Affiliate's Social Link (Optional): URL data type.

ID	Display Name	Type	Wallet Address	Stake Address	Profile Picture	Website	Social Link

Figure 5-2: Illustration of the Affiliates' table format within the platform database.

Incentive Programs' Data

This database consists of a table that includes all incentive programs' information, both active and ended ones. This information is adequate to describe an incentive program for interested affiliates. The parameters and their data types are the following:

- Incentive Program's SPO (Static): STRING data type.
- Incentive Program's Start Epoch (Static): INT data type.
- Incentive Program's End Epoch (Static): INTdata type.
- Incentive Program's Maximum Reward per Epoch (Static): BigInt data type.
- Incentive Program's Reward Rate (Static): Float data type. Measured in ADA per Brought ADA[(2)].
- Pool's Stake at Start (Static): BigInt data type. This is snapshotted from the stake pool information when the program launched.

With these parameters a handful of useful metrics can be deducted and shown/rendered on the platform as well. Some of these metrics are:

- Incentive Program's Status: values are active or ended.
- Incentive Program's Remaining Epochs: Can be shown for active programs.
- Incentive Program's Effect on Stake: Percent of brought ADA via affiliates to overall newly delegated ADA after program's start.

SPO ID	Start Epoch	End Epoch	Maximum Reward per Epoch	Reward Rate	Pool Initial Stake

Figure 5-3: Illustration of the Incentive Programs' table format within the platform database.

[2] Reward rate is calculated by dividing the maximum reward per epoch over the difference between the total stake at saturation and the total stake at the start of the campaign. It is referred to as ADA per brought ADA.

Data Collection

All data stored in the database are acquired from two sources. First one is user input, mostly during registration. Second source is the blockchain ledger itself.

Data Acquired from Users' Input

Typically these data are for users to introduce themselves to other users on the platform. No data from this type can result in any unwanted changes in the core outcomes from any of the protocols. These parameters are mostly collected during the registration stage. The following data parameters are examples of this type of data:

- Stake Pool's Display Name.
- Stake Pool's Profile Picture.
- Stake Pool's Website.
- Stake Pool's Social Link.
- Affiliate's Display Name.
- Affiliate's Type.
- Affiliate's Profile Picture.
- Affiliate's Website.
- Affiliate's Social Link.
- Incentive Program's Start Epoch.
- Incentive Program's End Epoch.
- Incentive Program's Maximum Reward per Epoch.

Data Handling Scripts Overview

The approach of collecting these types of data is done using PHP scripts in the backend server. These scripts are called by the frontend, making them local APIs, where they fill in or update the database with the data fed to them by the user. The logic flow for collecting this type is as follows:

- The user fills in some fields in the frontend, i.e. the platform webpage.
- The user submits their input values, mostly through a confirmation button.
- This frontend packs the inserted data and query the specified API(s) with this data.
- The script(s) in the backend depackage the incoming data and store them in their respective location within the platform's database.
- A feedback sent back to the frontend either confirming the process or reporting an error and requests a re-do.

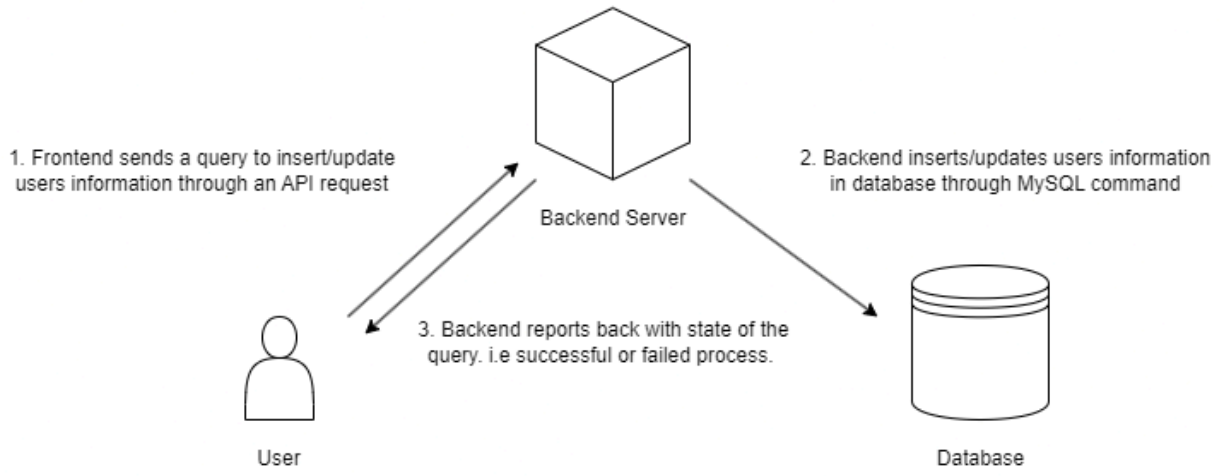

Figure 5-4: Illustration of handling the data acquired by users' inputs.

Data Acquired from the Blockchain

This data contains the majority of the information used on the platform. Adalink has two main gateways to query data from the blockchain. First one is the direct query from a full node installed in the backend server of the platform. Second source is Blockfrost API. All remaining data discussed in the previous section is retrieved by an API call in the backend or by directly querying the data from the installed node.

Data Handling Scripts Overview

This data type is retrieved from the blockchain via shell scripts within the backend and stored in the database. Later on, the stored/collected data can be acquired by both the backend protocols (i.e. via shell scripts) and the frontend web page (i.e. via PHP scripts).

Data Collecting Scripts

This is the stage of retrieving data from the blockchain. As stated earlier, this is done via shell scripts run within the backend. The logic flow can be summarized as follows:

- The script is triggered, i.e. by a timer event or external trigger.
- The script query either the installed node by using CLI commands or via API calls sent to Blockfrost database.
- Upon successful retrieval, the data is stored in its respective location within the platform's database.

Data Query Scripts - for Backend

These are shell scripts and are parts of the protocols that run in the backend. Their main purpose is to retrieve needed data from the database and use them in the protocol.

Data Query Scripts - for Frontend

These are PHP scripts that are triggered by the frontend through API calls. The scripts query the desired data from the database and send it back to the user. The logic flow of this process is below:

- The frontend requires a set of data via an API call.
- The API call triggers the corresponding script in the backend.
- The script pulls out the desired data values from the database.
- The script sends back the values of the desired data to the frontend.

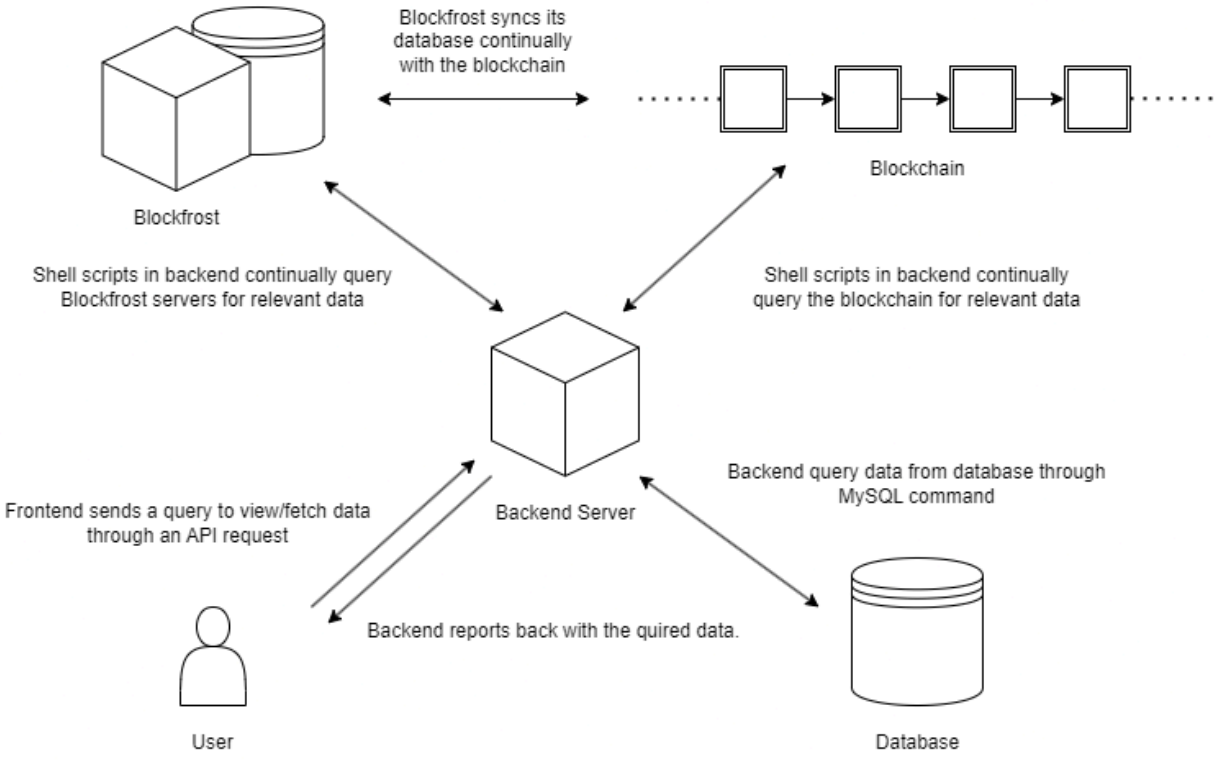

Figure 5-5: Illustration of handling the data acquired from the blockchain.

Conclusion

AdaLink's database architecture and data handling protocols provide a structured framework for managing stake pool, affiliate, and incentive program data efficiently. Through user input during registration and blockchain queries, the platform collects essential information. Backend shell scripts and PHP scripts facilitate data handling and retrieval, ensuring seamless communication between frontend and backend components. This robust system establishes AdaLink as a reliable and scalable platform.

Smart Contract Integration for Trustless Operations

Introduction

This chapter outlines the envisioned logic, structure, and functionality of the smart contract to be integrated with the platform. It aims to provide a clear understanding of how smart contract's principles could facilitate trustless interactions within the AdaLink Affiliate Network platform.

Motivation

From the research done in chapter 3 "Selecting a Reward Distribution Protocol" it was shown that with implementing a smart contract that act as the middle-man between SPOs and AdaLink, parties trust for each other to keep with their good intentions is reduced to a minimum without sacrificing on performance or security. In fact, it increases both security and performance of the platform. Security is enhanced by ensuring no outside party would be able to hack the funds deposited by SPOs as long as the smart contract logic is set properly. Performance of the platform is improved by making it easier to automate the reward distribution protocol.

Smart Contract Overview

The smart contract acts as an intermediary between stake pool operators (SPOs) and AdaLink. It is to be designed to manage reward distribution securely. It receives ADA payments as a form of deposits from SPOs and permits only one agreed upon AdaLink's wallet key to withdraw a sub amount of the deposits as a form of rewards to be later distributed on affiliates. These withdrawals are only available once an epoch with a maximum cap. The maximum cap amount coincides with the "Maximum Rewards per Epoch" parameter defined in chapter 4 "Reward Distribution Protocol".

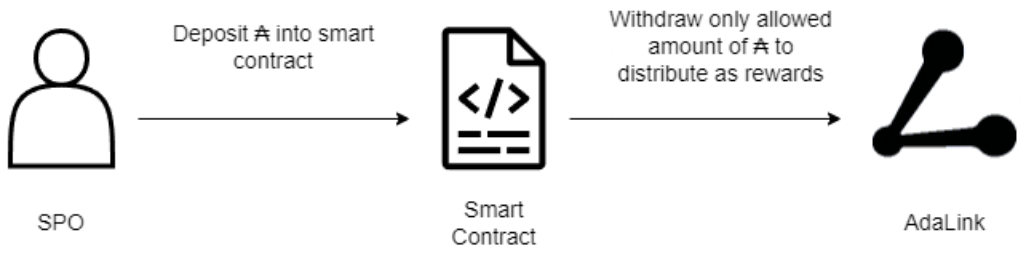

Figure 6-1: Illustration of the smart contract main objective.

Smart Contract Parts

Smart contracts in Cardano consist of three main parts. Namely, Redeemer, Datum and Script.

Redeemer

Redeemer is any set of data the user desires to include/feed into the smart contract when trying to execute a previously created instance of the smart contract. The expected user in this case is AdaLink. This part introduces flexibility to the smart contract in general. However, given that the overall objective in this specific case is to secure the deposited funds and narrow the ability to withdraw from the smart contract to only one wallet under a set of fixed rules, there is no need to add any form of flexibility to the contract. Therefore, the redeemer is assumed to be submitted empty and is ignored in the inner logic of the contract.

Datum

Datum is the set of data submitted to the contract by a user while creating a new instance of the said contract. The user creating a new instance is expected to be an SPO, since they are the ones depositing the total rewards amount of the campaign to be launched. The datum consists of the following data:

- Total deposit amount.
- Maximum rewards per epoch.
- Start epoch.
- End epoch.
- Public key hash of the depositor.

Script

Script is the logic of the contract presented as a compiled Plutus code. This core logic is the part that gets executed to check the validity of the transaction to either allow the funds to be withdrawn or not. Aiken language will be used to write the script code, which then will be compiled to Plutus code to be deployed later on to the blockchain.
The main logic of the smart contract can be described in the following bullet points:

- Calculate the current running epoch of the blockchain and check it with the last epoch of the incentive program.
- Check the number of outputs of the transaction. The number must fulfill the following constraints:
    - Number of outputs is two if the current running epoch is not the last epoch of the incentive program.

    - Number of outputs is one if the current running epoch is the last epoch of the incentive program.
- Check the details of the first output:
    - The receiving address must be AdaLink's whitelisted address.
    - The ADA amount must be equal to the "Maximum Reward Amount per Epoch" parameter.
- Check the details of the second output, if available:
    - The receiving address must be the smart contract address.
    - The ADA amount must be equal to the following:

$$F_i = F_t - \sum_{j=e_1}^{e_i} W$$

      Where, F_i is remaining funds sent back to the smart contract. F_t is the total initial deposit amount. e_i is the i^{th} epoch in the incentive program. W is the maximum reward amount per epoch.
- Validate the transaction only and only if all checks done in previous steps are satisfied.

Notice how the condition of the first output's ADA amount and the equation above together act as a guard by limiting the number of withdrawals per epoch to only one capped withdrawal action. A pseudo code describing the logic in the script is shown on the next page.

This is a pseudo code representing main key points of the script logic using syntax of Aiken language.

```
type Datum {
        total_deposit_amount,
        maximum_rewards_per_epoch,
        start_epoch,
        end_epoch,
        spo_public_key_hash
}

const reference_epoch = 474
const start_of_epoch_474_in_posix = 1710971125
const adalink_public_vkey = pUbLiCVeRiFiCaTiOnKeYOfAkNoWnWaLlEt

validator {
  fn vault (redeemer: Data,datum: Data,ctx: ScriptContext) -> Bool {
        let Datum dtm = datum
        let ScriptContext { transaction, purpose} = ctx
        let Interval { tx_lower_bound, tx_upper_bound }= transaction.validay_range
        let posix_difference = tx_lower_bound - start_of_epoch_474_in_posix
        let days_passed = floor(posix_difference/86400)
        let epoch_lower_bound = 474 + floor(days_passed/5)
        let posix_difference = tx_upper_bound - start_of_epoch_474_in_posix
        let days_passed = floor(posix_difference/86400)
        let epoch_upper_bound = 474 + floor(days_passed/5)
        let is_lower_and_upper_epoch_same = epoch_upper_bound == epoch_lower_bound
        let is_tx_outputs_count_correct = transaction.outputs.count < 3
        let is_first_output_valid = transaction.outputs[0].address.public_key_hash ==
adalink_public_vkey && lovelace_of(transaction.outputs[0].value) ==
dtm.maximum_rewards_per_epoch
        let is_second_output_valid = transaction.outputs.count == 1 || (
         transaction.inputs[0].address.public_key_hash ==
transaction.outputs[1].address.public_key_hash && lovelace_of(transaction.outputs[1].value) ==
dtm.total_deposit_amount - (epoch_upper_bound-reference_epoch)*maximum_rewards_per_epoch
        )
        Is_lower_and_upper_epoch_same && is_tx_outputs_count_correct && is_first_output_valid
&& is_second_output_valid
  }
}
```

Smart Contract Logic Flow

There are two main parts of the smart contract. The first one is the fund depositing action –by an SPO-. The second is the partial withdrawal of funds corresponding to the epoch's total reward amount. The second part is initiated by AdaLink's Reward Distribution protocol.

Funds Depositing

This is initiated by an SPO. When starting a new incentive program, AdaLink's platform will construct a deposit transaction for the SPO. This transaction creates a new output "i.e. a UTXO" that has the deposited funds and the correct datum to store the incentive program's details. This is equivalent to creating a new instance of the smart contract. In the next section a detailed discussion is presented describing the technicalities to achieve this objective.

Funds Withdrawal

This is initiated by AdaLink's Reward Distribution protocol at the end of each epoch. This transaction gets constructed in the backend of the platform. It uses the created output "i.e UTXO" from the deposit transaction as the main input. Depending on the current epochs order, if it is not the last epoch in the incentive program, the transaction divides the input into two outputs. First one is the partial withdrawal amount and is sent to AdaLink's protocol. This amount must be equal to the "Maximum Reward Amount per Epoch" defined in chapter 4. The second output is the remaining amount of the deposit sent back to the smart contract. The second output also includes the same datum as the input UTXO.

Notice how the newly created UTXO will be the input of the next withdrawal transaction "i.e. next epoch".

In the case of the last epoch of the incentive program, there will not be any remaining funds to be sent back to the contract. Therefore, the last epoch's withdrawal transaction has only one output which corresponds to the first output of the previous case.

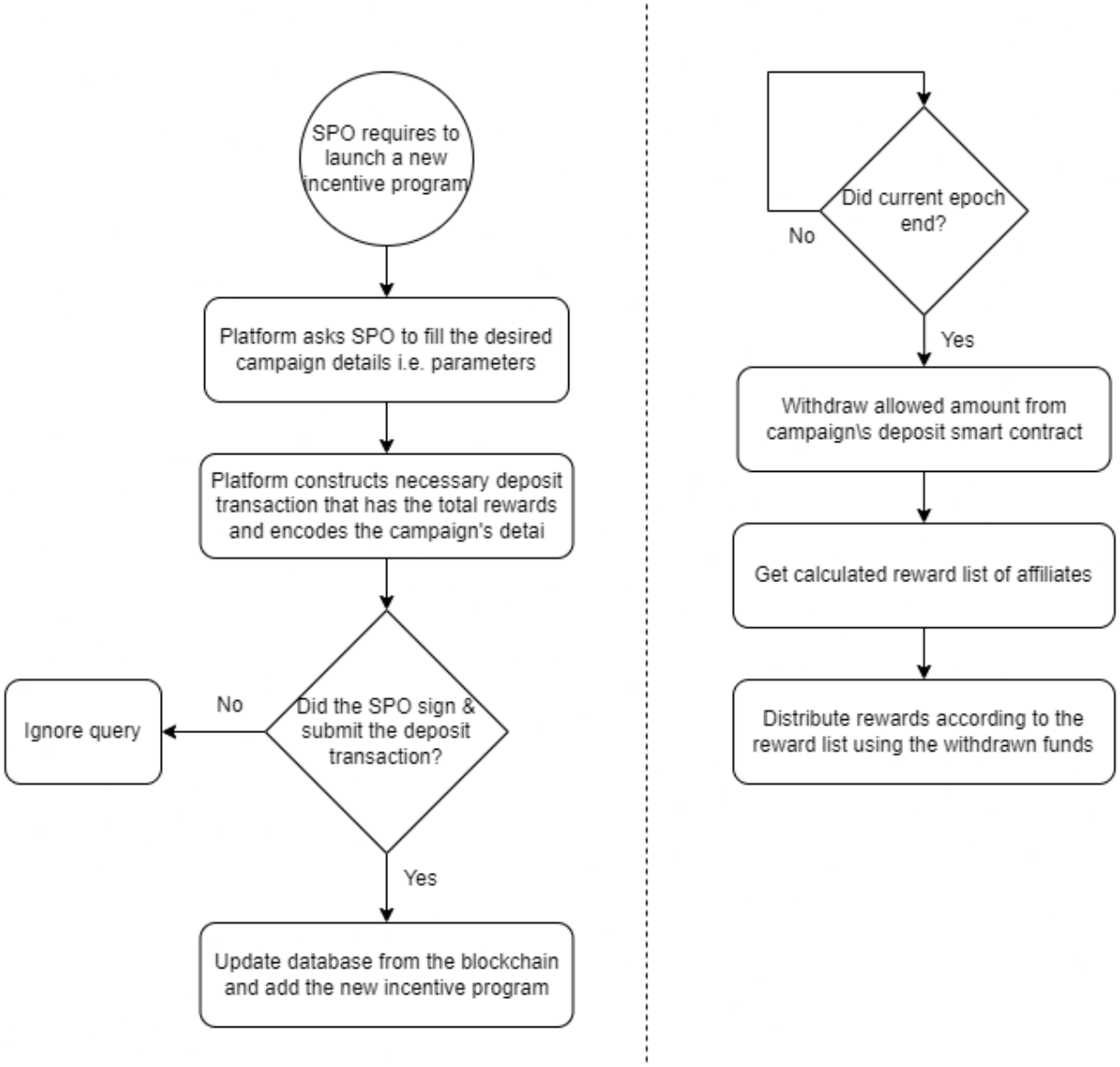

Figure 6-2: Flowchart representation of the logic flow of the smart contract's two main stages. Left one is the "Funds Depositing" phase and the right side part is the "Fund Withdrawal" stage.

Smart Contract Implementation

This section explains the way the smart contract is utilized within the platform. The implementation procedure is divided into four parts. Namely, development, deployment, creating of contract instances and executing contract instances.

Development

This part is in the pre-launch stage. It is when all technical aspects related to the smart contract are made. From brainstorming to finalized code.

Deployment

After developing the smart contract code. It is compiled to its raw form referred to as “Compiled Plutus Code”. This form is the actual string of code that the blockchain refers to while validating a transaction containing the smart contract.
In order to increase efficiency and performance, the compiled code is deployed into the blockchain via a deployment transaction. This transaction will have a major UTXO that will in itself contain the compiled code of the smart contract. Therefore, the platform will always refer to this UTXO while performing withdrawals actions, instead of attaching the whole contract code to every withdrawal transaction, resulting in a more efficient execution procedure.

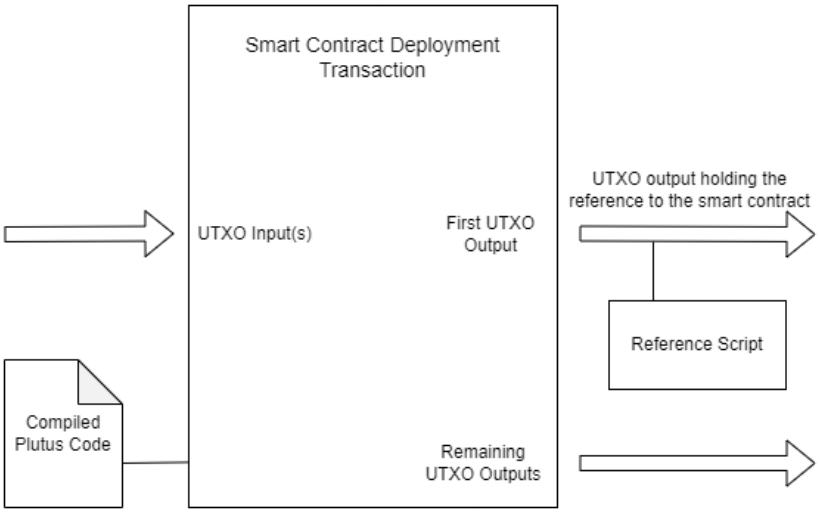

Figure 6-3: Illustration of the deployment transaction of the smart contract.

Creating of Contract Instances

One of the advantages of this particular design of the smart contract is that each incentive program uses a separated instance of the same contract. Therefore, there are no limits or concurrency issues of executing transactions that include the smart contract. Each time an SPO launches an incentive program, the platform will construct a deposit transaction. This transaction transfers the needed funds for the incentive program from the SPO into the smart contract address as a new UTXO. This UTXO can be thought of as an independent instance of the smart contract. Therefore, there can be as many UTXOs within the smart contract as needed, none of which collide or affect any other, allowing for fully parallelized operations.

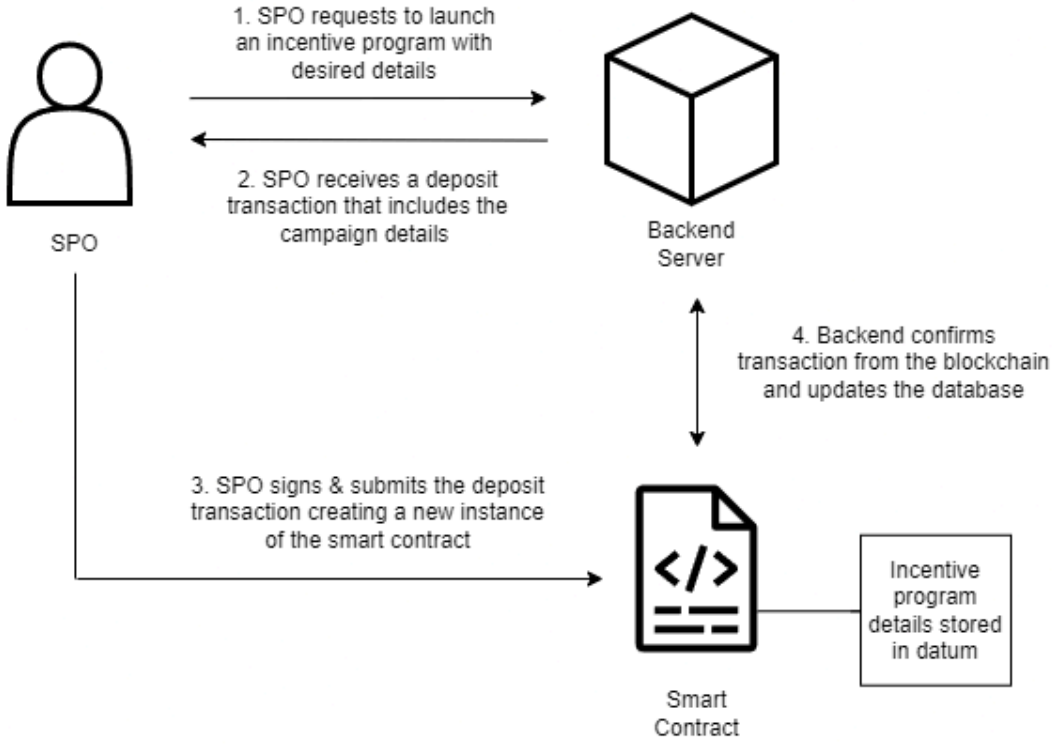

Figure 6-2: Illustration of implementing the smart contract in the initialization stage of incentive programs.

Executing Contract Instances

At the end of each epoch, Adalink's backend scripts will automatically withdraw the needed funds for each incentive program from its corresponding contract instance "i.e. corresponding contract's UTXO" in a separate transaction in a parallel fashion within the same block. These successfully withdrawn funds will then be distributed as rewards later on as discussed in the reward distribution chapter.

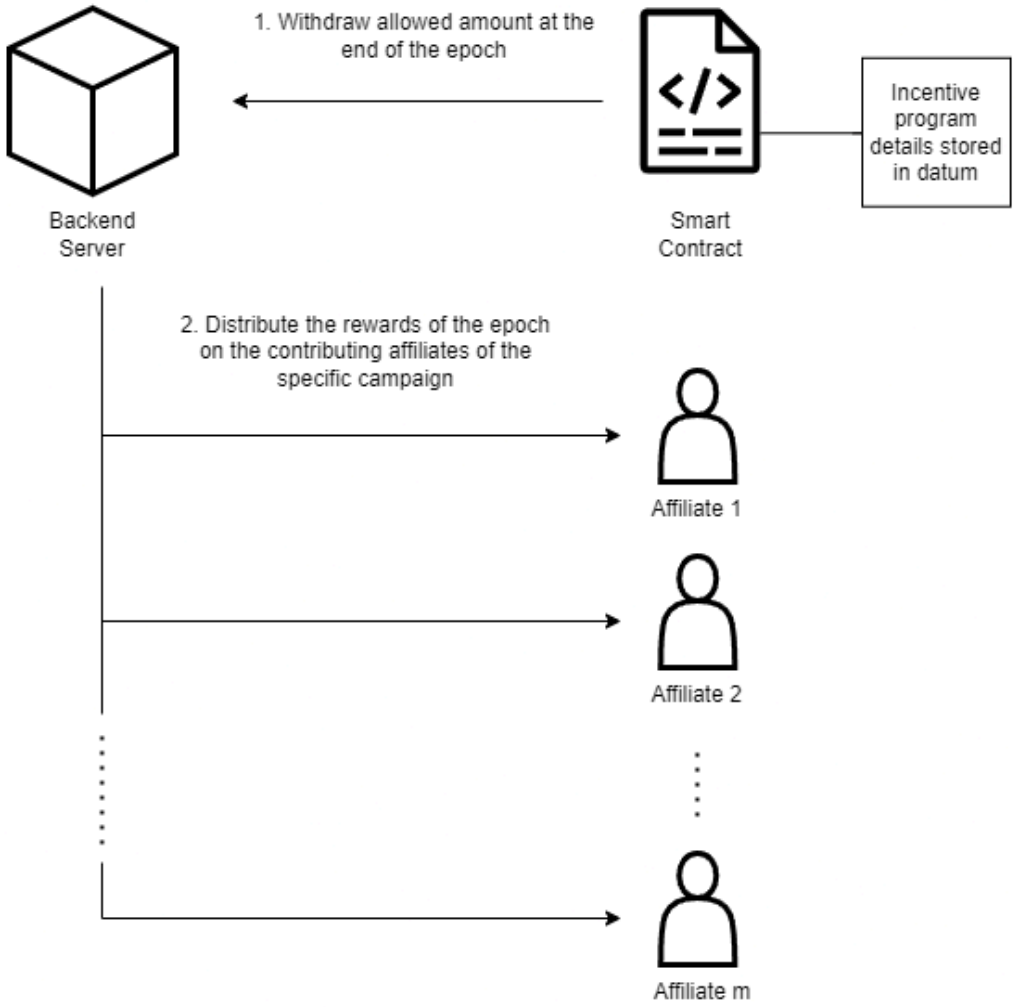

Figure 6-2: Illustration of implementing the smart contract in the reward distribution stage of incentive programs.