techpaper

abax lending protocol v1

Konrad Wierzbik
konrad.wierzbik@gmail.com

Łukasz Łakomy
lukasz.jan.lakomy@gmail.com

Abstract

In this paper, we describe the architecture of the abax lending protocol v1.

1 Introduction

The Abax Lending Protocol v1 is a smart contract system that allows users to seamlessly lend and borrow cryptocurrencies using the power of blockchain. The protocol is created with Rust ink! for substrate-based blockchains that support contract-pallet. Utilizing state-of-the-art liquidity pools, the protocol provides a peer-to-contract relationship for lending and borrowing PSP22 tokens with ease and efficiency.

Lenders can deposit their PSP22 tokens into the liquidity pool, and any user who provides adequate collateral can borrow the deposited tokens. Loans can be instantly accessed as long as there is available liquidity. Borrowers are subject to constantly accumulating interest on their loans, which is distributed to the lenders, creating a mutually beneficial experience for all parties involved.

The interest rate for borrowers and lenders is determined algorithmically to ensure fairness and efficiency. Borrowers' interest rates are based on the ratio of borrowed funds to supplied funds at a given time. As funds are borrowed from the pool, the available funds decrease, which raises the borrow interest rate. Lenders receive the generated income in the form of interest rates. This approach guarantees a balanced and secure lending and borrowing experience.

In addition, the protocol offers users various market rules to choose from. For instance, the standard market rule permits borrowing and using most assets as collateral, while the Stablecoin market only allows borrowing and using stablecoins as collateral. This is implemented to manage risks and facilitate more efficient borrowing of assets collateralized by other assets that have a strong correlation in their prices. This feature enables users to customize their borrowing experience according to their individual requirements.

1.1 General overview

At the core of the protocol, there is the Lending Pool contract which efficiently manages all deposits, debts, and interest rate calculations for all registered assets in the Lending Pool market. Users have the ability to easily deposit their collateral by calling the deposit function in the contract and transferring funds from the user to the contract. The protocol allows users to configure less risky assets as collateral to secure any potential debts. Users can borrow deposited funds at any time, provided that they pledge a sufficient amount of collateral.

The amount that can be borrowed is determined by the asset prices and two specific factors for each asset, the **Collateral Coefficient** (smaller than 1) and **Debt Coefficient**(greater than 1). These coefficients depend on the market rule a user has chosen. The **Collateral Power** of the user is calculated as the sum of deposited collateral values multiplied by the Collateral Coefficient, while the **Debt Power** is determined as the sum of borrowed asset values multiplied by the asset's borrow coefficient. To ensure safety for all parties, the user must manage his collaterals and debts to not allow for his Debt Power power to exceed Collateral Power or he will expose himself for liquidation. The majority of assets can be borrowed at variable rates. For now, stable rates are not available.

The Abax Lending Protocol ensures the safety of users' positions by allowing anyone to monitor for price fluctuations that could result in liquidation. Liquidation of a given user occurs when the user's Collateral Power drops below the user's Debt Power. In such cases, a user's position is considered undercollateralized, and any party has the opportunity to repay the debt in exchange for the collateral. The collateral received by the liquidator is equal in value to the repaid debt, plus an additional liquidation penalty paid by the user.

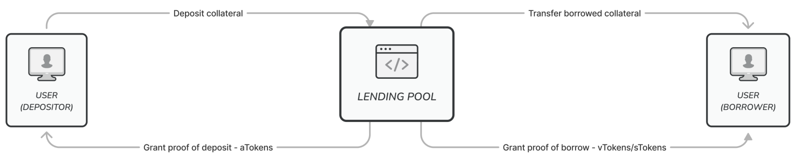
Deposit collateral
Transfer borrowed collateral
USER (DEPOSITOR)
LENDING POOL
USER (BORROWER)
Grant proof of deposit - aTokens
Grant proof of borrow - vTokens/sTokens

Figure 1: High-level overview of the protocol's supply/debt token flow

1.2 Formal definitions

ReserveData	A data structure that stores most of the data linked to one of the registered assets. ReserveData is stored in the `Mapping<AccountID(asset),ReserveData>`.

variable/parameter	**data type**	**symbol**	**describtion**
id	u32		The unique number assigned to the ReserveData and the corresponding registered asset. Denotes the position of data associated with this asset in Vectors.
activated	bool		Denoting if any actions with the use of the underlying asset are possible.
freezed	bool		Denotes if `deposit` and `borrow` actions on underlying asset are blocked.
decimal multiplayer	u128		The number of how many absolute tokens make one token. For example: 1AZERO = 10^{12}.
interest rate model e24	[u128;7]	$M[1-7]$	Slope of millisecond percentage rate for different utilization intervals: 0%-50%-60%-70%-80%-90%-95%-100%.
maximal total supply	Option <Balance>	ML	The maximal allowed **total supply**. Deposits that would increase **total supply** above this value will be reverted. If None then the **total supply** is uncapped.
maximal total debt	Option <Balance>	MD	The maximal allowed **total variable debt**. Borrows that would increase **total variable debt** above this value will be reverted. If None then the **total variable debt** is uncapped.
minimal collateral	Balance	mC	The minimal deposit of a user for which the user can use the asset as collateral. If `redeem` decreases the user deposit under this value, then the asset will be automatically removed from user collaterals. `Liquidate` can bring collateral under this value.
minimal variable debt	Balance	mD	The minimal (non-zero) debt a user can have after `borrow` and `redeem`. `Liquidate` can bring debt under this value.
income for suppliers part e6	u128	I	Part of the interest, paid by borrowers, distributed among suppliers .
flash loan fee e6	u128	F	Fee took during the `flash_loan`.
token price e8	u128		Price of the underlying asset in USD.
total supply	Balance	L	Sum of all deposits and accumulated earn interests.
cumulative earn rate index e18	u128	CI_L	An index used to keep track of accumulated interest. Whenever the reserve is updated it is multiplied by the accumulated interest rate.
current earn rate e24	u128	R_L	Current earn millisecond percentage rate. 10^{24} = 100% Millisecond Percentage Rate.
total debt	Balance	D_V	Sum of all variable debts together with the accumulated interests.
cumulative debt rate index e18	Balance	CI_V	An index used to keep track of accumulated interest. Whenever the reserve is updated it is multiplied by the accumulated interest rate.
current debt rate e24	u128	R_V	Current variable debt millisecond percentage rate. 10^{24} = 100% Millisecond Percentage Rate.
update timestamp	Timestamp	T_R	Timestamp of the last ReserveData update.
aToken address	AccountId		AccoundId of the aToken. PSP22 represents users' deposits.
vToken address	AccountId		AccoundId of the vToken. PSP22 represents users' debts.

Asset Rules	Data structures that store rules for borrowing a given asset at a given market rules. Asset Rules are stored in `Mapping<u64(market rule id), Vec<Option<AssetRule>>`. Position in the vector corresponds to the id in **Reserve Data** linked to the asset.

variable/parameter	**data type**	**symbol**	**description**
collateral coefficient e6	Option<u128>	CC	Coefficient used for **Collateral Power** calculation. If None then the asset can not be used as collateral.
debt coefficient e6	Option<u128>	DC	Coefficient used for **Borrow Power** calculation. If None then the asset can not be borrowed.
penalty e6	u128	P	Coefficient used to calculate penalty during liquidation.

UserConfig	Data structure storing information about which assets the user has deposited, the user uses as collateral, and is borrowing. It also stores the market rule id user has chosen.

variable/parameter	**data type**	**symbol**	**description**
deposits	Bitmap128		Each bit says if the user has deposited a given asset. The bit which corresponds to asset A is at ReserveData(A).id position counting from the right.
collaterals	Option<u128>		Each bit says if the user is using the asset as collateral. The bit which corresponds to asset A is at ReserveData(A).id position counting from the right.
borrows	u128		Each bit says if the user is borrowing the asset. The bit which corresponds to asset A is at ReserveData(A).id position counting from the right.
market rule id	u64	id	Index of the market rule a user has chosen.

UserReserveData	Data structures that store all data linked to the pair of one of the registered assets and one of the users. UserReserveData is stored in the `Mapping<(AccountId(asset), AccountId(user)),UserReserveData>`.

variable/parameter	**data type**	**symbol**	**description**
supplied	Balance	s	The amount supplied by the user together with the earned interest.
debt	Balance	d_V	The amount of debt at a variable rate together with the debt interest.
applied cumulative supply rate index e18	Balance	AI_L	An index to keep track of user earn interest.
applied cumulative debt rate index e18	Balance	AI_V	An index used to keep track of user debt interest.
use as collateral	bool		Denoting if a user is using the underlying asset as collateral.

Additional Variables	Additional variables used for calculations and the following explanations.

variable/parameter	**data type**	**symbol**	**description**
current timestamp	Timestamp	T	The current timestamp.
collateral power e6	u128	CP	Variable that represents users' debt power.
debt power e6	u128	DP	Variable that represents users used borrow power
health factor e6	u128	HF	Ratio of collateral power to debt power.
utilization rate e6	u128	U	Ratio of total debt to liquidity.
income per millisecond e24	u256	Q	Sum of variable and stable debt interests paid per millisecond.

After defining variables and parameters that appear in the protocol we can specify relations between them and some of the formulas used during calculations[1]. The formulas given below do not take into account differences in the variable's representation precision. For example, **debt coefficient e6** is a "real debt coefficient" multiplied by 10^6, while **current earn rate 24** is a "real current earn rate" multiplied by 10^{24}. Such representations are used in the contract because substrate blockchains don't support point numbers.

Utilization rate U is defined as:

$$U = \frac{D_V}{L}. \tag{1}$$

The formula for user's u **Collateral Power** CP is defined as:

$$CP(u) = \sum_{a \in Col(u)} s(a,u) \cdot CC(a, id(u)), \tag{2}$$

where $Col(u)$ is a set of user's collaterals and $CC(a, id(u)$ is **Collateral Coefficient** depending on a and market rule id chosen by the user $id(u)$. The formula for user's u **Debt Power** DP is defined as:

$$DP(u) = \sum_{a \in Bor(u)} d_v(a,u) \cdot DC(a, id(u)), \tag{3}$$

where $Bor(u)$ is a set of assets that the user is borrowing and $DC(a, id(u)$ is **Debt Coefficient** depending on a and market rule id chosen by the user $id(u)$. **Health Facotr** is defined as:

$$HF(u) = \frac{CP(u)}{DP(u)}. \tag{4}$$

Income Per Milisecond is defined as:

$$Q = D_V \cdot R_V \tag{5}$$

2 Lending Protocol Architecture

2.1 Lending Pool

- The Lending Pool contract is responsible for handling the protocol's core functionalities realizing LendingPool Trait:
  - `deposit` - depositing PSP22 tokens to the protocol and receiving proof of deposit PSP22 tokens - aToken.
  - `redeem` - redeeming PSP22 tokens for the corresponding proof of deposit PSP22 tokens - aToken.
  - `set as collateral` - setting some of deposited PSP22 tokens as a collateral for future borrows.
  - `choose market rule` - choosing a market rule user wants to use.
  - `borrow` - borrowing PSP22 tokens with variable(stable) interest rate and receiving a proof of debt PSP55 token - vToken(sToken).
  - `repay` - repaying PSP22 tokens while burning proof of debt PSP55 tokens.
  - `liquidate` - repaying the debt of an undercollateralized user and receiving some of his collateral.
- To take care of less used reserves contract can be extended by LendingPoolMaintain Trait:
  - `accumulate interest` - update ReserveData including current rates and cumulative indexes.
- As a Lending Pool Contract can possess a lot of liquidity, it can be a provider of flash loans by implementing LendingPoolFlashLoan Trait:
  - `flash loan` -allows users to perform flash loans - borrow PSP22 tokens, make use of them, and repay within a single transaction.
- Finally, the contract may be managed with LendingPoolManage Trait. The functions in this trait should be access controlled:
  - `register asset` - add a new underlying asset to the Lending Pool and create ReserveData and adjust its parameters.
  - `set reserve is active` - change the boolean ReserveData parameter **active** to turn on/off all actions that use the reserve.
  - `set reserve is freezed` - change the boolean ReserveData parameter **freezed** to turn on/off `deposits` and `borrows`.
  - `set reserve parameters` - change interest rate model, collateral and borrow coefficient, stable rate base, penalty, income for suppliers, and flash loan fee.
  - `add market rule` - adds new market rules under unique id.
  - `modify asset rule` - modifies asset rule for a given asset at a given market rule.
  - `take protocol income` - transfer protocol income to a given account.
- Upon performing the above operations the Lending Pool contract emits appropriate events.
- To perform the above the contract holds main chunk of the storage and performs various computations (updating user reserve data, reserve borrow/supply indexes, etc).
- Lending Pool serves as a storage for PSP22 proxies AToken, VToken (see **??**).

[1] More formulas are found in section (3)

2.1.1 Storage

The core storage structures of the Lending Pool storage are **Reserves Data**, **UserConfig**, **UserReserve Data**, and **MarketRule** which is Vec<Option<**AssetRules**>>. On top of it the **Registered Asset** Vec<AccountId> stores a list of all assets registered to the contract. The order in the list corresponds to the asset ids in **ReserveData**s.

As mentioned above, **Abax Lending Pool** keeps track of reserve datas by storing information about them in structures called **ReserveData**. **ReserveData** is uniquely linked to the registered **PSP22 token** (underlying token). These structures are stored in mapping `<asset's address => ReserveData>`.

The **Market Rules** are stored in mapping `<market rule id => Market Rule>` and contain a list of **AssetRules** that determine each asset's parameters for a given market.

Furthermore, **Abax Lending Pool** stores user data in structs called **UserConfig** and **UserReservesData**. They track the state corresponding to the user's general position and the pair (underlying asset, user). They account for all operations performed by the users. These structures are stored in mappings `<user's address => UserConfig>` and `<(asset's address, user's address) => UserReserveData>`

2.2 Abax Tokens

From the user's perspective, Abax Supply Token (AToken) behaves like a regular PSP22 token. It does allow for the usage of all of the actions specified by the PSP22 trait meaning the users can treat and use it like any other PSP22-compliant token. Having said that there are implementation differences.

- AToken does not store information about user balances. Nevertheless, it does keep track of user allowances allowing them to perform all of the operations implied by implementing the PSP22 trait.

Abax Variable Debt Token (VToken) derive some of the PSP22 trait behavior with one key difference. Similarly to AToken, they keep track of allowances. However, to perform the transfer the receiver has to approve the transferred amount (in a PSP22 scenario it is the sender that does have to approve said amount). This is due to VToken representing debt (a representation of a "negative value") and not a possession (a "positive value"). Having said all of the above, users may transfer not only proof of supply but also proof of debt.

2.3 Lending Pool and Abax Token interactions

Upon executing functions that perform a change in Abax Token storage-proxy state held by the Lending Pool contract (actions like `deposit`, `redeem`, `borrow`, `repay` or `liquidate`) Lending Pool performs a call to Abax Tokens to emit transfer events. For example, when a user makes a `deposit` of an asset the **supply** variable in the corresponding **UserReserveData** is increased. Exactly the same field represents the balance of proof of deposit PSP22 token - AToken. In other words, we can say that `deposit` mints Atokens but, the balance is not stored in AToken PSP22 token storage but in LendingPool. To remain in compliance with the PSP22 standard, on mint action, the transfer event inside PSP22 Contract should be emitted. To do so, LendingPool calls an accessed control function in AToken to emit an appropriate event.

Similarly, upon performing transfer operations invoked by a call to the PSP22 trait's 'transfer' function, Abax Token contracts perform a call to the Lending Pool contract. The callee does perform all of the checks and updates necessary to approve the transfer. Then it does update its state and returns to the Abax Token contract (caller) which emits a transfer event.

2.4 Configuration of the Lending Pool

The configuration of the Lending Pool contract is available only for permitted users. To restrict the calls to functions from **LendingPoolManage** trait we use the AccessControl module with the following roles:

- ASSET LISTING ADMIN - may call `register asset`.
- PARAMETERS ADMIN - may call `set reserve parameters`, `add market rule`, and `modify asset rule`.
- EMERGENCY ADMIN - may call `set reserve is active` and `set reserve is freezed`.
- GLOBAL ADMIN - may call any functions from **LendingPoolManage** expect `take protocol income`.
- TREASURY - may call `take protocol income`.
- ROLE ADMIN - manages which Account has which role.

3 Interest Rates

3.1 Interest Rate Model

3.1.1 Debt Rate

The **current debt rate** [Millisecond Percentage Rate] for each asset is determined by the **interest rate model** and the **utilization rate** [$\in [0,1]$] of the given asset in the following way:

$$R_V(M,U) = \begin{cases} M[1]\cdot U & U\in[0,0.5),\\ M[1]+10(M[2]-M[1])(U-0.5) & U\in[0.5,0.6),\\ M[2]+10(M[3]-M[2])(U-0.6) & U\in[0.6,0.7),\\ M[3]+10(M[4]-M[3])(U-0.7) & U\in[0.7,0.8),\\ M[4]+10(M[5]-M[4])(U-0.8) & U\in[0.8,0.9),\\ M[5]+20(M[6]-M[5])(U-0.9) & U\in[0.9,0.95),\\ M[6]+20(M[7]-M[6])(U-0.95) & U\in[0.95,1),\\ M[7]\cdot U & U\in[1,1]. \end{cases} \tag{6}$$

3.1.2 Earn Rate

The **current earn interest rate** R_L is calculated based on the income generated from variable and stable debts interest Q, supplied **liquidity** L, and **interest for suppliers part** I:

$$R_L = \frac{Q\cdot I}{L}. \tag{7}$$

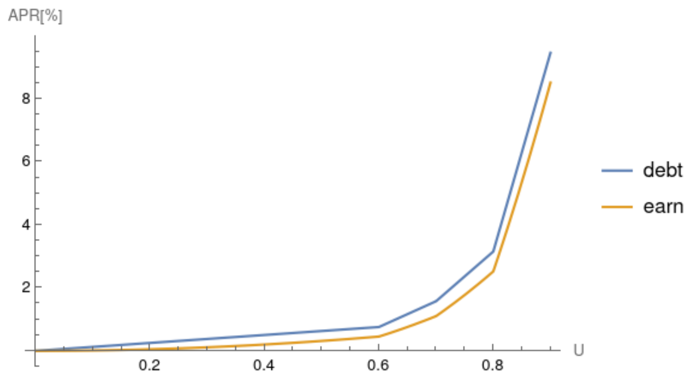

Figure 2: Possible Debt and Earn rates for Utilization Rate $U \in (0, 0.9)$.

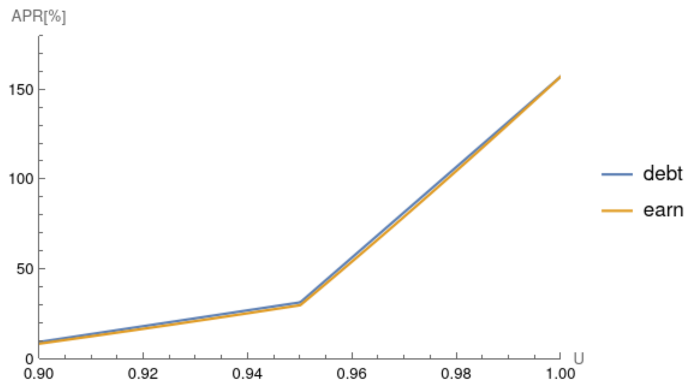

Figure 3: Possible Debt and Earn rates for Utilization Rate $U \in (0.9, 1)$

3.2 Accounting Interests

Accounting of interests is made in every function (like `deposit`) that modifies ReserveData or UserReserveData in two steps. First, the interest is accumulated based on the previous interest rates. Then Data is modified according to the called function (`deposit`). Eventually, new interest rates are calculated and they will be used in the next contract call for interest accumulation.

The accumulation of interests for every user is done any time ReserveData parameter **cumulative debt rate index** and **cumulative earn rate index** are updated.

3.2.1 Accumulate Interest

`accumulate interest` operation modifies the ReserveData variables: **total supply** L, **cumulative earn rate index** CI_L, **total debt** D_V, **cumulative debt rate index** CI_V, and **update timestamp** T_R based on the **current interest rate** R_L, **current debt rate** R_V and **current timestamp** T in the following way:

$$L(T) = L(T_R)(1+R_L(T-T_R)), \tag{8}$$

$$CI_L(T) = CI_L(T_R)(1+R_L(T-T_R)), \tag{9}$$

$$D_V(T) = D_V(T_R)(1+R_L(T-T_R)), \tag{10}$$

$$CI_V(T) = CI_V(T_R)(1+R_L(T-T_R)), \tag{11}$$

$$T_R(T) = T. \tag{12}$$

`accumulate user interest` operation uses already accumulated ReserveData to update UserReserveData variables: **supplied** s, **variable debt** d_V, **applied cumulative earn rate index** AI_L, and **applied cumulative debt rate index** AI_V in the following way:

$$s(new) = s(old)\cdot\frac{CI_L}{AI_L}, \tag{13}$$

$$AI_L(new) = CI_L, \tag{14}$$

$$d_V(new) = d_V(old)\cdot\frac{CI_V}{AI_V}, \tag{15}$$

$$AI_V(new) = CI_V, \tag{16}$$

3.2.2 Recalculate Current Rates

`recalculate current rates` is an operation on a ReserveData that updates the **current earn rate** R_L and **current debt rate** R_V as expressed in 3.1

4 Collateral

4.1 Market Rules

The **Market rule** is a set of rules that determine what assets users can use as collateral, which ones can be borrowed, and how the user's **Health Factor** is being calculated. Information about each of the registered assets are stored in **Market Rule** list entries Option<**Asset Rules**>. If the option is **None** then the asset is not accessible within this rule at all. The default **Market Rule** has the id 0 and is set for every user upon first interaction with **Abax Protocol**. This setting can be changed by each user by calling `choose market rule`. This design, of having multiple possible borrowing rules, allows the protocol to provide more efficiency for its users by specifying many possible rules.

4.2 Liquidation

`Liquidate` will work only if the liquidated user's **Health Factor** is below one. The liquidator specifies two assets, one that is borrowed (asset to repay - AR) and one that is used as collateral (asset to take - AT) by the liquidated user. Based on **penalty** parameters P_T and P_R in **Asset Rules**, of both AR and AT, and current asset prices V_R and V_T the maximal amount of AT to be taken during liquidation A_T is calculated based on the repaid amount of AR A_R:

$$A_T = \frac{V_R}{V_T} \cdot (1 + P_T + P_R) A_R. \tag{17}$$

As long as the Collateral Power of an asset to take CP_T and the Debt Power of an asset to repay DP_R satisfy the following conditions:

$$CP_T < 1 - P_T \quad \text{and} \quad DP_R > 1 + P_R \tag{18}$$

then one can show that after repaying A_R amount of AR and taking A_T amount of AT the Health Factor of the liquidated user will increase.

5 Lending Pool interactions

Since all of the core interactions (deposit, redeem, borrow and repay) have much in common regarding things such as the usage of storage, events emitting, or security checks they perform, they were structured in a way so they follow the same execution pattern.

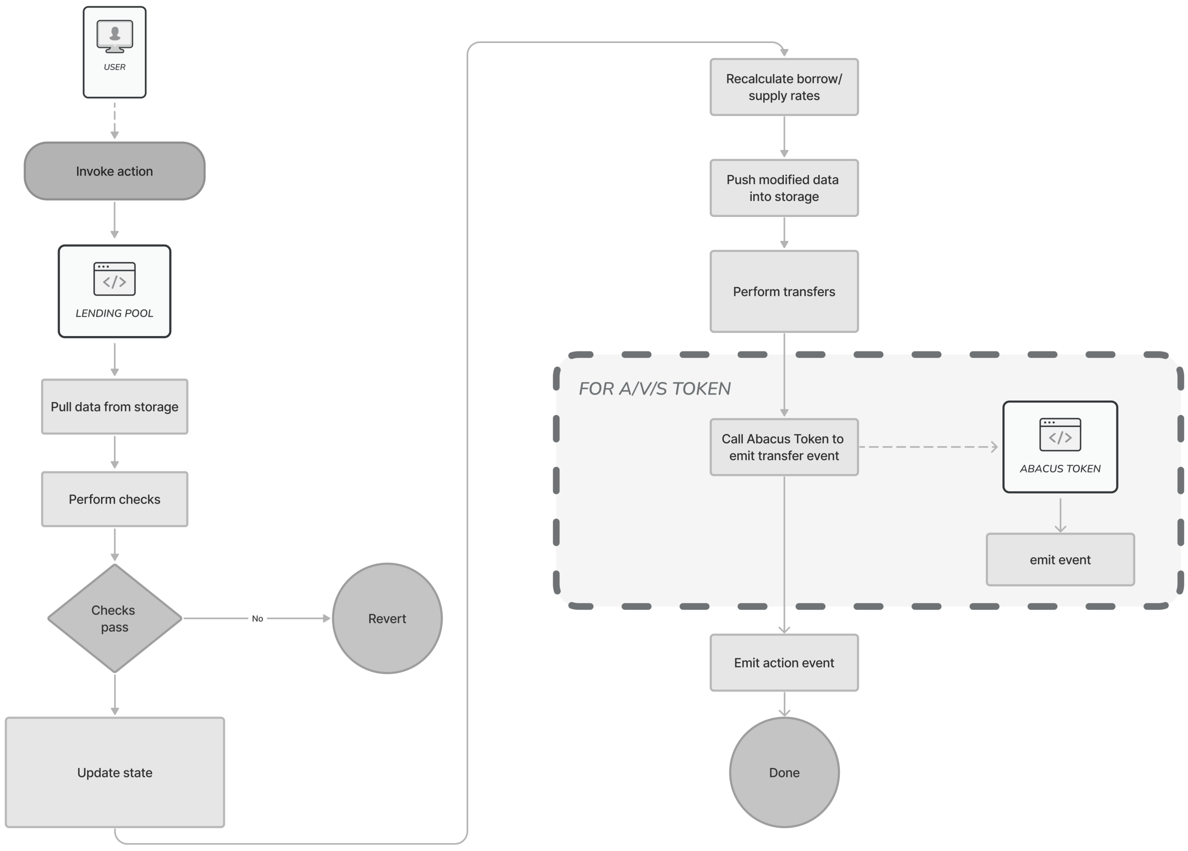

Figure 4: High-level structure of core interaction procedure flow.

5.1 Deposit

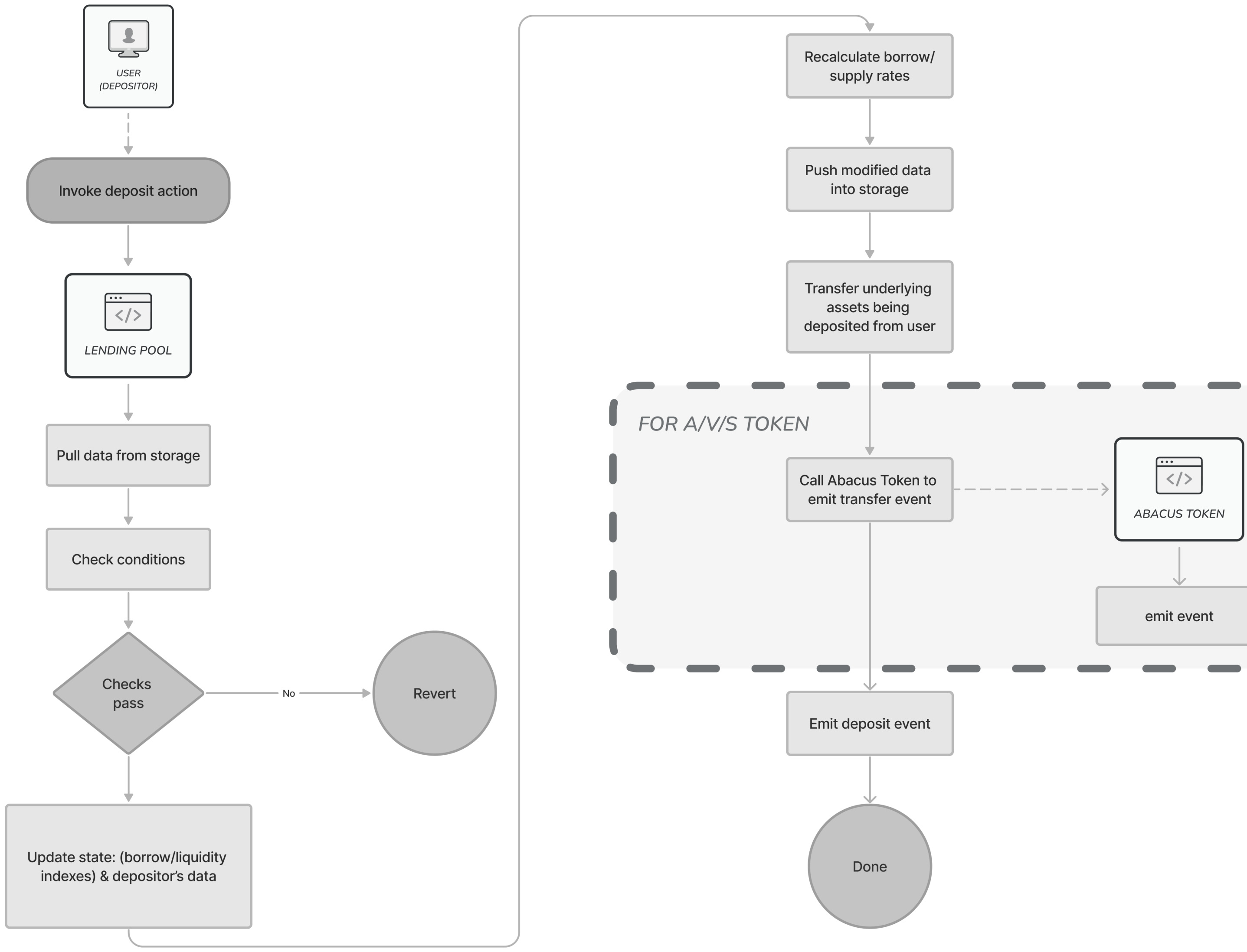
USER (DEPOSITOR)
Invoke deposit action
LENDING POOL
Pull data from storage
Check conditions
Checks pass
No
Revert
Update state: (borrow/liquidity indexes) & depositor's data
Recalculate borrow/ supply rates
Push modified data into storage
Transfer underlying assets being deposited from user
FOR A/V/S TOKEN
Call Abacus Token to emit transfer event
ABACUS TOKEN
emit event
Emit deposit event
Done

Figure 5: Deposit flow diagram.

5.2 Redeem

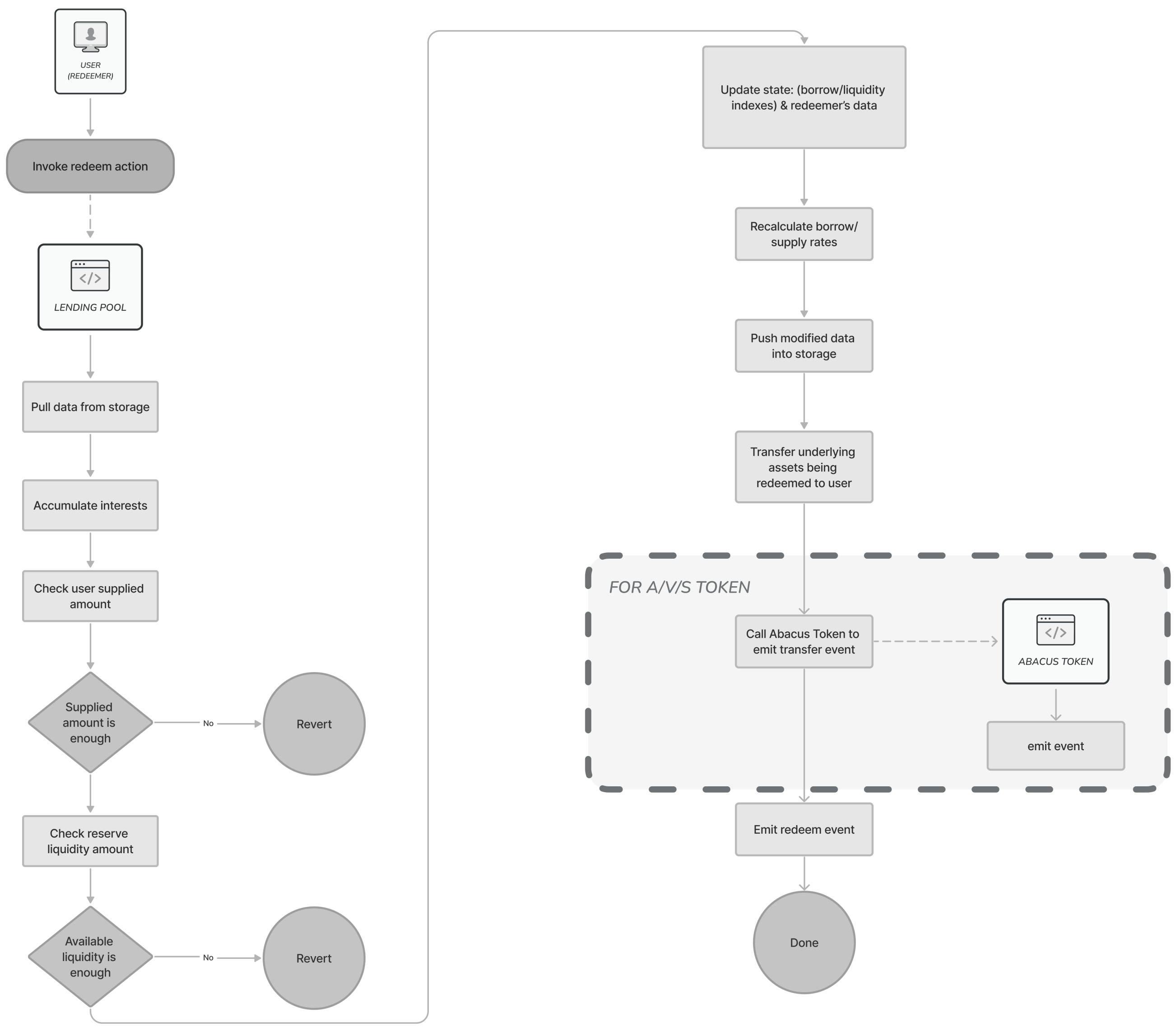
USER (REDEEMER)
Invoke redeem action
LENDING POOL
Pull data from storage
Accumulate interests
Check user supplied amount
Supplied amount is enough
No
Revert
Check reserve liquidity amount
Available liquidity is enough
No
Revert
Update state: (borrow/liquidity indexes) & redeemer's data
Recalculate borrow/ supply rates
Push modified data into storage
Transfer underlying assets being redeemed to user
FOR A/V/S TOKEN
Call Abacus Token to emit transfer event
ABACUS TOKEN
emit event
Emit redeem event
Done

Figure 6: Redeem flow diagram.

5.3 Borrow

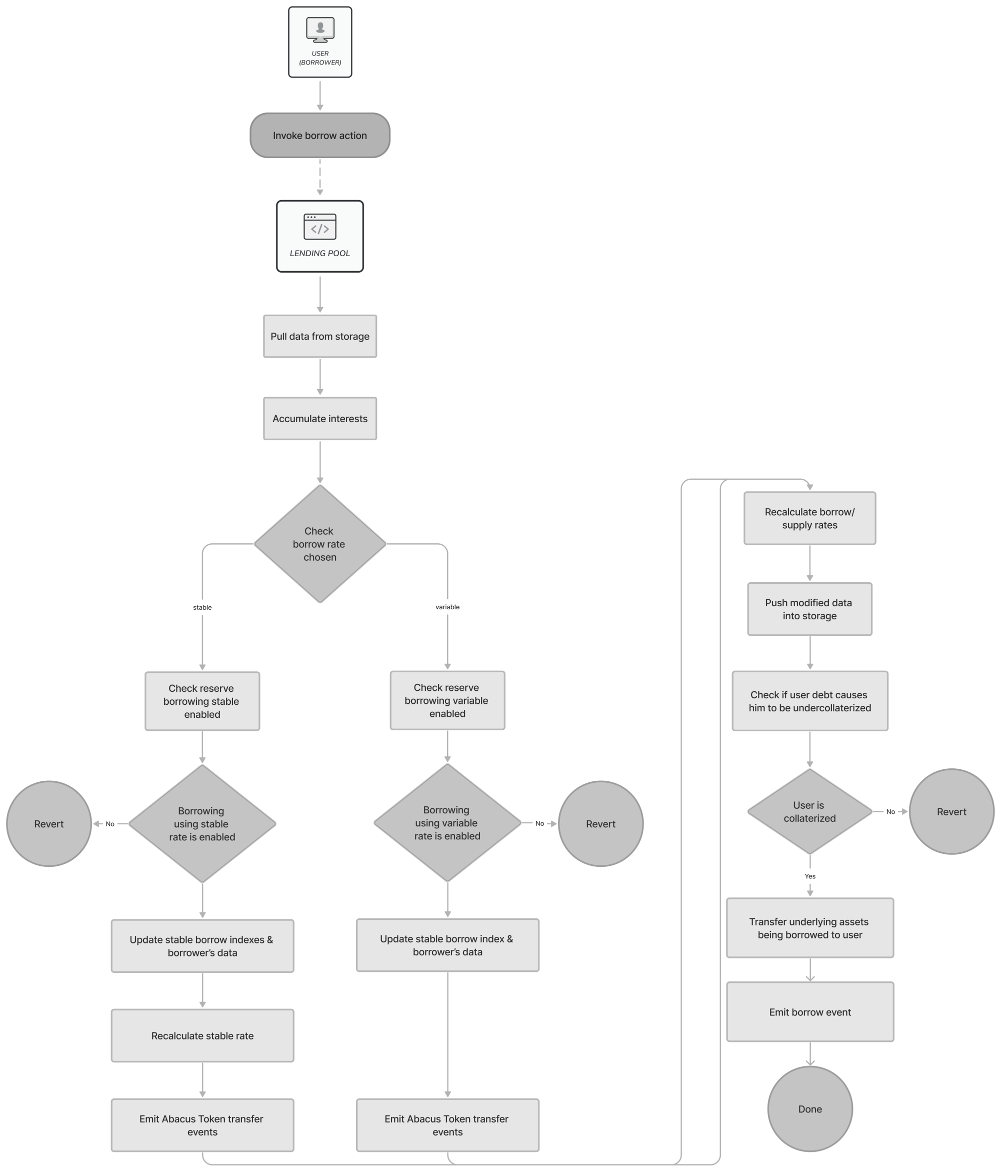
USER (BORROWER)
Invoke borrow action
LENDING POOL
Pull data from storage
Accumulate interests
Check borrow rate chosen
stable
variable
Check reserve borrowing stable enabled
Check reserve borrowing variable enabled
Revert
No
Borrowing using stable rate is enabled
Borrowing using variable rate is enabled
No
Revert
Update stable borrow indexes & borrower's data
Update stable borrow index & borrower's data
Recalculate stable rate
Emit Abacus Token transfer events
Emit Abacus Token transfer events
Recalculate borrow/ supply rates
Push modified data into storage
Check if user debt causes him to be undercollaterized
User is collaterized
No
Revert
Yes
Transfer underlying assets being borrowed to user
Emit borrow event
Done

Figure 7: Borrow flow diagram.

5.4 Repay

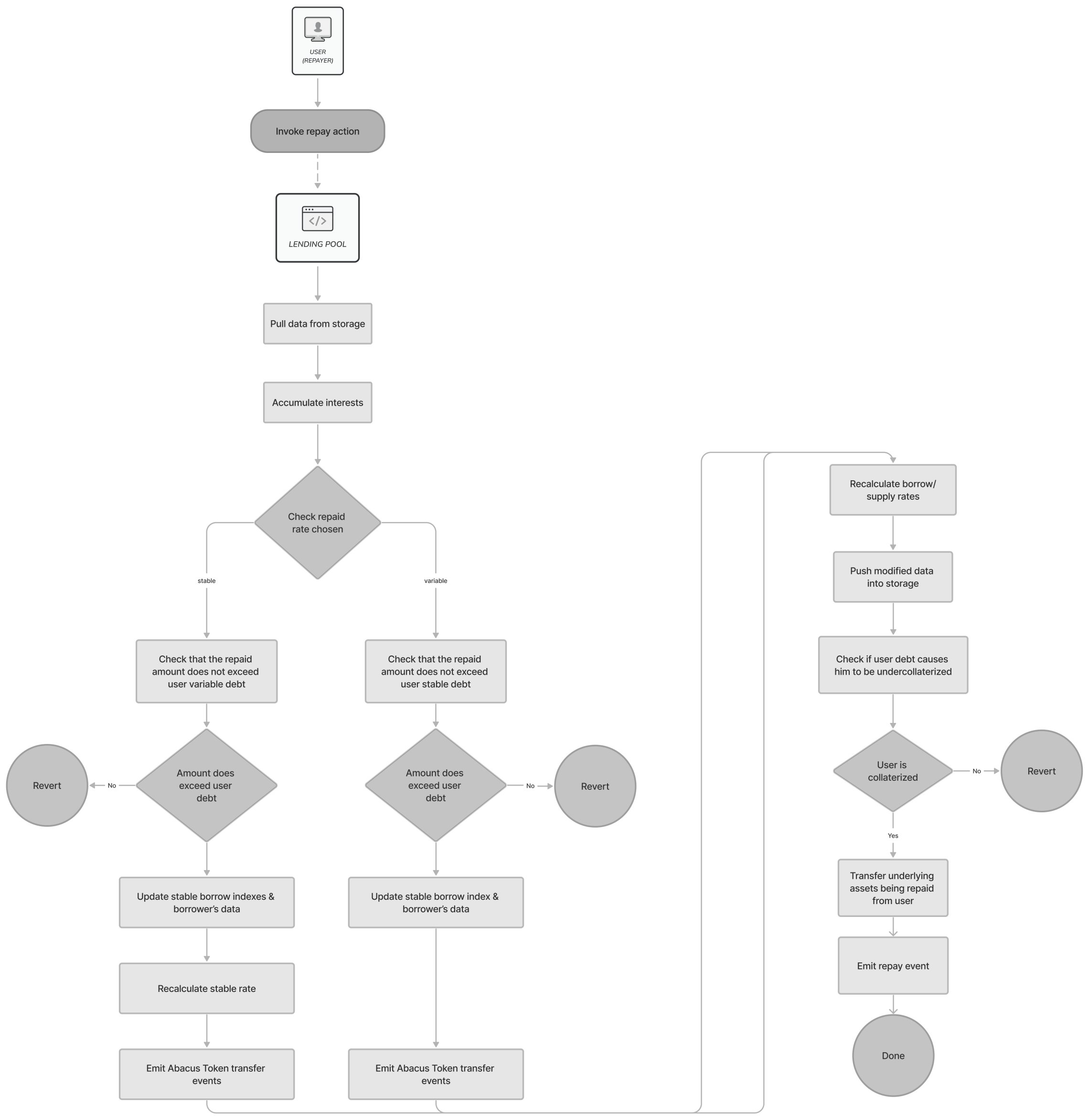
USER (REPAYER)
Invoke repay action
LENDING POOL
Pull data from storage
Accumulate interests
Check repaid rate chosen
stable
variable
Check that the repaid amount does not exceed user variable debt
Check that the repaid amount does not exceed user stable debt
Amount does exceed user debt
No
Revert
Amount does exceed user debt
No
Revert
Update stable borrow indexes & borrower's data
Update stable borrow index & borrower's data
Recalculate stable rate
Emit Abacus Token transfer events
Emit Abacus Token transfer events
Recalculate borrow/ supply rates
Push modified data into storage
Check if user debt causes him to be undercollaterized
User is collaterized
No
Revert
Yes
Transfer underlying assets being repaid from user
Emit repay event
Done

Figure 8: Repay flow diagram.

5.5 Flash Loan

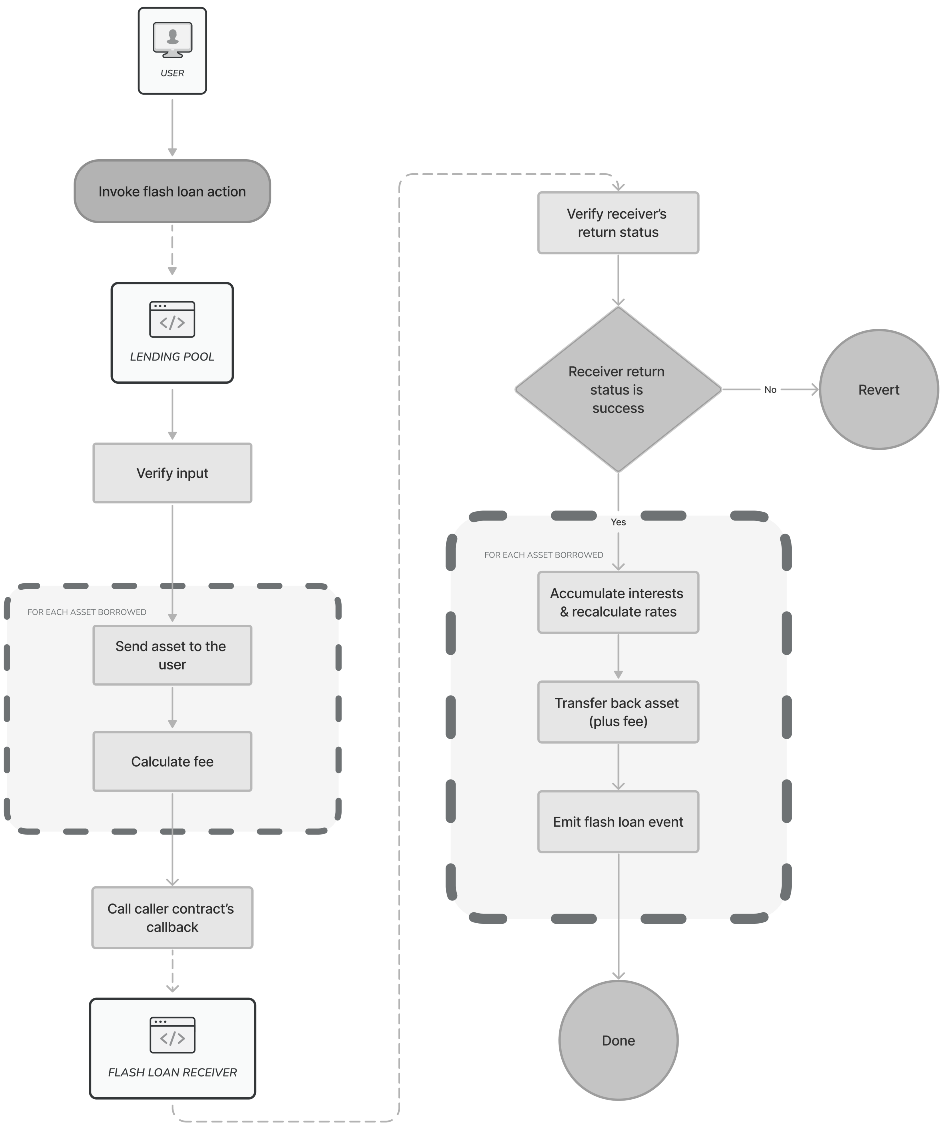

Figure 9: Flash loan flow diagram