Governance-Minimized Credit Market Protocol (Ecru)

By Ecruware

July 6, 2023

Abstract

We propose a novel credit market that supports different internal and external use cases for the protocol-created credit. Internally, credit can be extended as a credit line to borrow vaults facilitating specialized credit markets. Externally, credit can be minted in the form of a protocol-issued stablecoin and utilized within the greater DeFi ecosystem. The bootstrapping of new credit markets around the proposed protocol is further facilitated by the permissionless onboarding of borrow vaults.

1 Introduction

Stablecoins of all configurations have proven their product-market fit for DeFi. Hundreds of billions of dollars' worth settled trillions in transaction volume in 2022, clearly highlighting the value that the market ascribes to purchasing power preservation and frictionless exchange. Their adoption has been driven by reserve-backed stablecoins like USDT and USDC, with decentralized alternatives growing alongside, albeit at a smaller scale.

Today's decentralized stablecoins have pioneered valuable mechanism designs - collateralized debt positions, stability pools, automated market operations - that make it possible to transmute volatile collateral into stable credit. Yet their credit creation has remained limited in scope, with few primary use cases emerging outside of leveraging deeply liquid collateral. Some incumbents have addressed this reality by parlaying proprietary liquidity into new initiatives, but fundamental innovation around the creation of stablecoin credit has undoubtedly slowed.

Ecru aims to challenge this status quo with the FIAT II protocol. The protocol is a rethinking of stablecoin credit creation as a function of marketplace interactions rather than as an exercise in analogue consensus formation. Ecru allows users to lend out their ability to mint the stablecoin, making it a modular framework for allocating and pricing credit. Its protocol design boasts a number of emergent properties, including permissionless collateral onboarding and undersecured lending. And while active protocol governance remains a part of its architecture, its role will prove to be less about binary decision-making and more so about granular calibration.

All of this is to say: Ecru is learning from the lessons of incumbent DeFi stablecoins and money markets to build the most flexible and comprehensive credit protocol to date.

2 Credit Creation

Credit is the unit of account for minting power over the system's stablecoin. The *credit and debt module* (CDM) allows for the creation, transfer and eventual burning of credit. It therefore extends credit lines to designated accounts called *vaults*. Credit lines are capped at a debt ceiling up to which the vault may borrow credit from the CDM. These credit lines are the sole mechanism of the system to create credit. Credit in the CDM can then be used towards the minting of the system's stablecoin or, alternatively, it can be freely transferred to another account. A specific use case for this is the delegation of credit to other vaults by which this vault's credit line is extended effectively lending the delegated credit to the vault. An account in the CDM can thus be described by the triple $(C_i, D_i, \bar{D}_i)$ where C_i and D_i represent its credit and debt balance and $\bar{D}$ the debt ceiling. Furthermore, the account's credit line, L_i, is determined by

$$L_i = C_i + \bar{D}_i - D_i. \tag{1}$$

Thus, an account has two options to receive a credit line; it may attract credit delegations from other accounts or it receives a non-zero debt ceiling. Since existing credit can be transferred freely among accounts, the extension of non-zero debt ceilings is the systems's main risk control mechanism.

2.1 Create Credit

When utilizing its credit line, an account's available credit is always prioritized over the creation of new credit. Thus, when borrowing the amount b of credit the account's balances are updated as follows

$$\begin{aligned} C_i' &= max(C_i - b, 0) \\ D_i' &= D_i + max(b - C_i, 0) \\ C_j' &= C_j + b \end{aligned}$$

with j the account that is credited with the borrowed credit. We further keep track of the total (global) debt, D, by

$$D' = D + max(b - C_i, 0)$$

When creating credit the global debt ceilings are always enforced by verifying the following conditions

$$\begin{aligned} D_i' &< \bar{D}_i \\ D' &< \bar{D} \end{aligned}$$

with global debt ceiling D'.

2.2 Repay Credit

Similarly, when repaying credit we first honor any debt the account has. Thus, consider that account j repays credit amounting to b on behalf of account i. The updated account and global debt balances thus are

$$\begin{aligned} D_i' &= max(D_i - b, 0) \\ C_i' &= C_i + max(b - D_i, 0) \\ C_j' &= C_j - bD' = D - max(b - D_i, 0) \end{aligned}$$

2.3 Bad Debt

Governance may bail out a CDM borrower, such as a CDP vault, at any time in an ad-hoc restructuring intervention raising unbacked credit that is allocated to a designated *buffer* account.

3 Borrow Vaults

The system allows for the creation of credit by extending credit lines to designated accounts. Generally speaking, these accounts can be thought of borrow facilities specialized for some use case. This makes the system extremely flexible in terms of the supported use cases including but not limited to the following

- Over-collateralized Lending
- Unsecured Lending
- Peer-to-peer Lending

While technically any vault implementation can attract credit delegations in a permissionless manner and thus receive a non-zero credit line, credit is only created from a non-zero debt ceiling $\bar{d}_i > 0$. Thus, the bootstrapping of credit is orchestrated through a set of governance-approved vaults that implement a strict over-collateralization regime on borrow positions in order to minimize the risk of accumulating unbacked credit supply, i.e. credit c_i that is not backed by the value of a collateral asset. In this section we discuss the mechanisms implemented by these borrow vaults.

3.1 Collateralized Debt

Over-collateralized lending is based on the concept of *collateralized debt positions* (CDP) which can be seen as individual loans that are backed by the value of a collateral asset. The vault mechanisms thereby ensure that this backing is always maintained. Therefore, consider a vault i with credit line L_i and a specific collateral asset that is accepted as backing for debt. This vault then allows any account j to draw from the vault's credit line by depositing the collateral asset. Account j's debt position is thus represented by the tuple (n_j, a_j) where n_j is the borrowed normal debt and a_j the deposited collateral assets. The position's normal debt is subject to an interest accumulation process expressed in the position's (interest) rate accumulator $I_j \geq 1$ and discussed in more detail in Section 3.6. The total debt due at position repayment is then defined by

$$d_j = I_j n_j \tag{2}$$

3.2 Graceful Liquidations

The objective of over-collateralization is to enforce that each credit unit borrowed, and potentially in circulation, is backed by at least the same value in the collateral asset locked in the vault. The vault therefore implements a graceful liquidation mechanism that allows external liquidators to repay an under-collateralized position's debt in exchange for receiving its collateral assets to the extent that the position is appropriately collateralized again. Therefore, the vault defines a minimal collateral ratio, $\underline{cr}$, that defines the health of vault positions in terms of the liquidation price

$$\underline{p} = \frac{p}{\underline{cr}} \tag{3}$$

with the current collateral asset spot price p. Note that we simplify the notation by avoiding the vault indexes for both the minimal collateral ratio and spot price as this shouldn't lead to confusion. Based on the liquidation price position (n_j, a_j)'s health factor is defined as

$$h_j \overset{\text{def}}{=\!=} h((n_j, a_j)) = \frac{a_j \underline{p}}{d_j} \tag{4}$$

The health factor thus serves as a standardized metric for the *distance to liquidation* (or safety) of a position. This distance is characterized by the liquidation threshold

$$h_j \leq 1 \longrightarrow \text{liquidate position } (n_j, a_j). \tag{5}$$

Unsafe positions, positions with $h_j \leq 1$, are then available to be liquidated whereby this liquidation is permissionless such that it can be executed by anyone. Specifically, the liquidator is offered the position's collateral assets at a discount in order to repay position debt and increase position health.

3.2.1 The Class of Linear, Graceful Liquidations

While different implementations of the liquidation mechanism outlined above exist, we consider the specific class of linear, *graceful* liquidations only. Therefore, we define a graceful liquidation as the subset where only parts of the position collateral is sold such that the resulting position health factor reaches a predefined target health factor, $\bar{h} > 1$. We can describe this class of liquidations by the mapping

$$\mathcal{L} : (n_j, a_j) \to (n'_j, a'_j) \quad \text{s.t.} \quad h((n'_j, a'_j)) = \bar{h} \tag{6}$$

We further restrict the acceptable implementations by those adhering to the linear form

$$\mathcal{L}((n_j, a_j)) = \begin{pmatrix} d_j - (\Delta d + b) \\ a_j - \Delta a \end{pmatrix} \tag{7}$$

where b is any residual debt, that cannot be offset with the available collateral assets and instead is accounted for as bad debt. Thus, the liquidation strategy allows an external liquidator to buy Δa units of collateral assets at the price of Δd units of credit. In order to create an economic incentive to capture this opportunity the exchange of credit for collateral assets is offered at a discounted spot price pe with *liquidation discount factor* $e \in (0, 1)$. The smaller the liquidation discount is, the higher is the potential profit captured by the liquidator

$$p\Delta a - pe\Delta a - g = (1-e)p\Delta a - g \tag{8}$$

where g is the gas fee paid by the liquidator to the Ethereum network. Note that this is an unrealized profit as it does not reflect the sell price the bidder achieves when liquidating the offset collateral on the market. Disregarding this effect for now, the bidder's profit is positive as long as $(1-e)p\Delta a > g$ and so calibration of the liquidation discount e to the Ethereum gas market is of great importance.

In each processed liquidation, the system takes a liquidation fee, expressed in the liquidation penalty $f \in (0, 1)$, on the repaid credit amount $x = pe\Delta a$. This fee is used towards the settling of accumulated bad debt and hence increases the system's overall resilience. On the other hand, the liquidation penalty reduces the credit used to offset the liquidated position's debt accordingly

$$\Delta d = xf \tag{9}$$

3.2.2 Solving the Liquidation Problem

The objective function of a liquidation according to Equation 6 is for the position health to be reset to the vault's target health factor $\bar{h}$. Based on this criteria and the above equations we are able to establish a system of equations with the liquidation parameters $x, \Delta d$ and Δa as the target variables;

$$\begin{cases} \bar{h} & = \frac{(a_j - \Delta a)\underline{p}}{d_j - \Delta d} \\ \Delta a & = \frac{x}{pe} \\ \Delta d & = xf \end{cases} \tag{10}$$

Note that this system is fully defined such that a unique solution exists. We therefore substitute the second and third equation into the first

$$\bar{h} = \frac{(a_j - \frac{x}{pe})\underline{p}}{d_j - xf}$$

Solving for the repay amount x yields

$$x = \frac{\bar{h}d_j - a_j\underline{p}}{\bar{h}f - \frac{1}{\underline{cr}e}} \tag{11}$$

Note that Equation 11 implies a condition on the values of $\bar{h}, f, \underline{cr}$ and e in order for the solution to be defined (x is only defined in the positive, non-∞ range). This condition translates to a restriction on the chosen vault parameters

$$\bar{h} \geq \frac{1}{\underline{cr}ef} \tag{12}$$

On the other hand, since only positions with $h_j < 1$ are liquidated we know that the numerator in Equation 11 too is strictly positive resulting in only valid solutions for x. Once x is fixed we find the solutions for Δa and Δd by plugging x into the respective Equations in system 10.

3.2.3 Bad Debt

Bad debt is defined as the residual debt b of a position after all of the position's collateral has been used to offset debt in a liquidation. Thus, bad debt accumulates whenever the position collateral, a_j, is insufficient to honor the collateral to be sold to the liquidator, Δa, in order to offset the required position debt, Δd, or

$$a_j < \Delta a \longrightarrow \text{accumulate bad debt} \tag{13}$$

Specifically, if this condition is met the accumulated bad debt is

$$b = d_j - pefa_j \tag{14}$$

Bad debt is then allocated to the vault's unbacked debt balance and the respective position is deleted.

3.3 Credit Delegation

Credit delegation allows anyone in the system with a positive credit balance to deposit credit to a vault account thereby effectively increasing its credit line L_i. When delegating credit to a vault, delegators in exchange receive vault shares that represent a claim on the vault's total credit. The vault maintains this claim in different forms including a (liquid) credit balance, position debt or future interest payments. More formally, vault i's total credit claim, V_i, is computed by

$$V_i = C_i^R + C_i^W + I_i * N_i - D_i - F_i \tag{15}$$

with

- C_i^R is the credit balance - C_i^W is the credit withheld for unfixed epochs (see Section below) - I_i is the vault's global rate accumulator - N_i is the total normal debt - D_i is the total debt owed to the CDM - F_i are the total accrued (but not collected) protocol fees

In the forthcoming sections we will discuss how vault shares are minted and burned as delegators deposit and withdraw credit to/from a vault. We thereby omit again the vault index i in the quantities where this does not lead to confusion.

3.3.1 Delegate Credit

When delegating v_j units of credit, an account j deposits credit to a vault and is minted vault shares in exchange. The amount of vault shares minted, s_j, is computed as

$$s_j = \begin{cases} v_j & \text{if} \quad S = 0 \\ \frac{v_j}{V} S & \text{else if} \quad D + F < C^R + C^W + I * N \\ \text{revert} & \text{else} \end{cases} \tag{16}$$

with S the total shares outstanding in the vault at the time of credit delegation. The first case in Equation 16 initializes the vault shares if the total supply of shares is zero (e.g. before the very first credit delegation), the second case reflects the vault state for regular deposits and the third case applies for *insolvent* vaults (see Section below).

Inline with Section 2.2, the delegated credit is then used to repay any debt the vault may have in the CDM and the residual amount is credited to the vault's credit balance. A credit delegation thus results in the following updates of the vault balances

$$\begin{aligned} C^{R\prime} &= C^R + max(v_j - D, 0) \\ D' &= max(D - v_j, 0) \\ S' &= S + s_j \end{aligned}$$

3.3.2 Undelegate Credit

When undelegating credit, vault shares are burned in exchange for withdrawing the implied credit claim. Therefore, the vault's credit reserves need to be sufficient to honor the credit claim. This may result in race conditions should multiple share holders want to undelegate at the same time. In order to ensure an orderly undelegation process we thus introduce the concept of undelegation epochs. The undelegation process spans over multiple epochs:

- Epoch 1 (Undelegate): Queue vault shares for undelegation
- Epoch 2 (Fix Claim): Fix credit claim for all undelegated claims
- Epoch 2+ (Claim): Claim credit for fixed epochs

More formally, let $\mathbf{E} = e_0, e_1, e_2, \dots$ define a partition of undelegation epochs $e_k, k = 0, 1, 2 \dots$ We require that the epochs are separated by a constant epoch duration, i.e. $e_j - e_{j-1} = e_k - e_{k-1}, \forall j, k$.

Epoch 1: Undelegate Shares

Holder j may then undelegate shares $q_j(k)$ at any time t_1 such that the shares are queued with epoch $e_k, e_k \le t_1 < e_{k+1}$, undelegation queue $\mathbf{Q}(e_k) = (q_0, q_1, \dots)$. When queued, shares $q_j(e_k)$ and the associated estimated credit claim are locked in queue $\mathbf{Q}(e_k)$ in order to prevent double-spending of shares and reflect the undelegation in the vault credit utilization (and possibly the vault's interest rate). An undelegation thus results in the following balance updates

$$\begin{aligned} s'_j &= s_j - q_j(e_k) \\ \mathbf{Q}'(e_k) &= \mathbf{Q}(e_k) \cup q_j(e_k) \\ C^{R\prime} &= C^R - \frac{q_j(e_k)}{S(t_1)} V(t_1) \\ C^{W\prime} &= C^W + \frac{q_j(e_k)}{S(t_1)} V(t_1) \end{aligned}$$

with $S(t_1)$ and $V(t_1)$ the total shares and vault credit claim at the time of undelegation. While locked, undelegated shares $q_j(e_k)$ and the associated credit claim are still considered regular shares and hence earn interest and are exposed to bad debt risk. Consequently, the undelegation of shares does not affect the total and per-share credit claim for the vault.

Epoch 2: Fix Claim

The undelegation claim associated with queued shares in $\mathbf{Q}(e_k)$ can be fixed during the fixing period (e_{k+e^D}, e_{k+e^T}) or at any epoch after the *epoch fix delay*, e^D, and before the *epoch fix timeout*, e^T. Epochs for which the undelegation claim is not fixed within this period are considered *stale epochs* and shares queued in a stale epoch have to be unlocked and transferred back to a user's shares balance manually.

At any time $t_2, e_{k+e^D} \le t_2 \le e_{k+e^T}$, we then fix the undelegation claim associated with epoch e_k by first computing the total shares queued and the associated total credit claim, or

$$Q(e_k) = \sum_{q_j \in \mathbf{Q}(e_k)} q_j \tag{17}$$

$$W(e_k) = \frac{Q(e_k)}{S(t_2)} V(t_2) \tag{18}$$

where $S(t_2)$ and $V(t_2)$ are the total shares and vault credit claim at the time of fixing the claim. Further, we fix the actual vault credit available to honor undelegations. This consists of the vault's current credit line and credit currently withheld for unfixed epochs

$$C(e_k) = L(t_2) + C^W(t_2) \tag{19}$$

The mismatch of the epoch's total credit claim and available credit results in an adjustment of the amount of shares for which the undelegation can be honored. We express this adjustment in a *claim ratio* computed as follows

$$R(e_k) = \frac{C(e_k)}{max(W(e_k), C(e_k))} \tag{20}$$

Therewith, we are able to fix epoch e_k's adjusted total shares, $Q'(e_k)$, and credit claim, $W'(e_k)$, honored for redemption

$$Q'(e_k) = R(e_k)Q(e_k)$$
$$W'(e_k) = R(e_k)W(e_k)$$

The adjusted undelegated shares and associated credit claim are immediately deducted from the vault's total shares and credit reserves. Thereby, epoch e_k's fixed undelegation claim is locked and not subject to the accumulation of future bad debt.

Epoch 2+: Withdraw Claim

Vault shares queued in undelegation epoch $\mathbf{Q}(e_k)$ and with claims fixed at time t_2 can be claimed at any time $t_3 > t_2$. Thereby, the honored shares, $q'_j(e_k)$, and withdrawable credit claim, $w'_j(e_k)$, are computed using epoch e_k's claim ratio

$$q'_i(e_k) = R(e_k)q_i(e_k)$$
$$w_i(e_k) = \frac{q'_i(e_k)}{Q'(e_k)}W'(e_k)$$

Upon withdrawal the credit claim is transferred to account j and the shares are burned from the account's claimable balance. Note that the total vault shares have already been reduced by epoch e_k's adjusted total shares, $Q'(e_k)$, at the claim fixing time t_2 and so do not have to be adjusted when the claim is withdrawn.

3.4 Vault Solvency

A vault's total credit claimable by share holders, V, implies a solvency criteria: We say a vault is *solvent* as long as the credit claim is positive, or $V \geq 0$. Vault solvency is an important property because it ensures that economic incentives exist for delegators to deposit credit. On the other hand, if a vault turns insolvent, or $V < 0$, depositing credit effectively translates to bailing out the vault and its existing delegators. Based on Equation 15 we can describe the insolvency condition in terms of a vault's credit reserves, total normal debt, rate accumulator, total debt in the CDM and fees accumulated but not yet collected

$$\begin{cases} C^R + C^W + I * N \geq D + F & \text{Vault solvent} \\ C^R + C^W + I * N < D + F & \text{Vault insolvent} \end{cases} \tag{21}$$

As no incentive exists for delegators to bail out an insolvent vault, restructuring requires an ad-hoc governance intervention.

3.5 Credit Redemption

The ability to offer stablecoin redemptions at a fixed, internal price has proven to be an effective mechanism for enhancing peg stability. Centralized stablecoins such as USDC offer redemption of the stablecoin against fiat USD while decentralized stablecoins like agEUR can be redeemed against the systems crypto asset reserves. Similar to Liquity's LUSD, we offer credit redemptions against the collateral assets locked in CDP style borrow vaults. However, other than LUSD, we give the collateral owner full control over the terms under which she is willing to facilitate redemptions. Specifically, the owner of a position (n_j, a_j) is able to offer her position for redemption at a fixed exchange price.

More formally, let there be a discrete partition of tick prices $\Pi = (\pi_1, \pi_2, \dots, \pi_M)$ such that $0 < \pi_1 < \pi_2 \dots < \pi_M$. The partition does not have to be equidistant but we allow for $\pi_k - \pi_{k-1} \neq \pi_i - \pi i - 1$ for $k \neq i$. Each tick price π_k then translates to an exchange price $p_k = p\pi_k$, with p expressing the current oracle price, at which a user is offering her position collateral to be swapped to credit. In other words, prices p_k represent the *ask prices* to a limit order placed by the position owner. In a redemption, these limit orders are executed executed sequentially in the following order

1. Best price
2. First-in-first-out (FIFO)

We can thus describe this market as a partial (ask orders only) limit order book (PLOB). Each CDP vault implements its own PLOB potentially with a collateral asset specific configuration of the tick price partition Π.

3.5.1 Create Limit Order

New limit orders are associated with an existing position (n_j, a_j) and tick price π_k. We can thus describe a limit order by $o_{j,k}$ where the indices j and k refer to the associated position and tick price. When created, the limit order is then inserted in the tick price π_k's FIFO queue. Furthermore, when creating a limit order, the position receives a maker rebate that applies to the interest charged on the position's debt (see Section below).

3.5.2 Execute Limit Order

Limit orders $o_{j,k}$ can be executed in a permissionless manner by anyone accepting the offered ask price π_k. A redemption is then initiated by submitting a redemption order to the vault's PLOB indicating the exchange parameters:

- *max credit amount*, m, or the maximal amount of credit to be exchanged to collateral
- *max limit price*, $\bar{\pi}$, the maximal tick price at which the exchange can be executed

For the execution, the PLOB then iterates through all limit orders $o_{j,k}$ with tick prices $\pi_k \leq \bar{\pi}$ and in increasing order of tick price as well as position in the FIFO queue until either of the following conditions is met:

$$\pi_k > \bar{\pi} \quad \text{max execution price reached} \tag{22}$$

$$\sum_{j \in \mathbf{J}} c_j > m \quad \text{max credit amount reached} \tag{23}$$

where $\mathbf{J}$ represents the set of positions executed as part of the redemption. Thus, a redemption may be honored by the PLOB in full or the redemption order may be executed partially only, if not enough volume is available at the indicated limit prices.

3.5.3 Cancel Limit Order

Active limit orders may be cancelled at any time by the position owner. Thereby, the position cannot be redeemed against anymore and, in exchange, it also does not receive a maker rebate going forward.

3.5.4 Maker Rebates

Similar to liquidity makers in a central limit order book, limit orders increase liquidity for redeemers and thus improve peg stability. In order to incentivize liquidity provisioning at low tick prices, limit orders earn a maker rebate expressed in the rebate factor r_j. This rebate is a function of the order's tick price, or

$$r(o_{j,k}) = 1 - \frac{1}{a + b(\pi_k - 1)} \tag{24}$$

with floor parameter $a \in (1, \infty)$ and scale parameter $b \in (0, \infty)$. Note that the floor parameter inversely relates to the maximal rebate given which is derived by $\underline{r} = \frac{1}{1-a}$. An implementation of this function for demonstration purposes can be found in this Desmos graph https://www.desmos.com/calculator/jnekvt6ri4.

3.6 Interest Rate Model

As mentioned above, a position's normal debt, n_j, accrues interest over time. We here discuss how this interest accumulates per position and for the vault overall. Therefore, let j refer to an individual position and $\mathbf{J}$ to the set of all positions in the vault such that $j \in \mathbf{J}$. Furthermore, we find the vault's total normal debt by

$$N = \sum_{j \in \mathbf{J}} n_j \tag{25}$$

Moreover, we recall that each position is associated with a limit order $o_{j,k}$ in the vault's PLOB and a respective maker rebate r_j. As users make changes to their position, e.g. reduce or increase normal debt or change the limit order ask price, this creates a discrete time partition $\mathbf{S} = s_0, s_1, s_2, \ldots,$ with $s_0 < s_1 < s_2 < \ldots$ with s_i reflecting the timestamp of the block within which the update was made. Note that partition $\mathbf{S}$ is specific to the position for which updates are tracked. We will thus use the notation $\mathbf{S}_j$ in order to reference a specific position's update time partition if needed. The union of all the position update times then creates a new, discrete time partition that reflects all interactions with the vault that result in a position state change

$$\mathbf{T} = \bigcup_{j \in \mathbf{J}} \mathbf{S}_j = t_0, t_1, t_2 \ldots \quad \text{with} \quad t_0 < t_1 < t_2 \ldots \tag{26}$$

Along this partition, vaults accumulate and distribute interest from borrowers based on a (variable) base interest rate $i(t_k) \in \mathbf{R}^+$ that expresses a non-compounding, annual rate. Based on this rate, interest accumulates according to the difference operator

$$A(t_{k-1}, t_k) = i(t_{k-1}) y(t_{k-1}, t_k) \tag{27}$$

with year fraction $y(t_{k-1}, t_k) = \frac{t_k - t_{k-1}}{\text{YEAR}}$ for any two timestamps $t_k \geq t_{k-1}$ and YEAR $= 366 * 86400$, i.e. the number of seconds of a full year. However, in order to derive a position's effective interest accrual process we have to discuss the different regimes governing the dynamics of the base rate $i(t_k)$ and account for a position's maker rebate.

3.6.1 Interest Rate Regimes

Each vault implements its own interest rate model defining the dynamics of the base rate $i(t_k)$. We here propose two distinct models. While future vault implementations may incorporate a different model, the models discussed here have found widespread adoption in DeFi and thus serve as a good starting point for further innovation.

Static Rate

This model considers the rate $i(t_k)$ to be a parameter that may be immutable or controlled by protocol governance. As a result, interest rate changes are ad-hoc and implemented by protocol governance.

Utilization Based Rate

The interest rate $i(t_k)$ in this model is a function of a vault's credit utilization. The rational for this is that credit utilization can be thought of the supply and demand curve for credit allocated to a specific borrow vault and the interest rate expresses the current equilibrium in the market. For this purpose, a vault's credit utilization at time t_k, $U(t_k)$, is defined as

$$U(t_k) = \frac{I(t_k) N(t_k)}{I(t_k) N(t_k) + C(t_k) + \bar{D} - D(t_k)} \tag{28}$$

where $I(t_k)$ is the vault's global rate accumulator (see below). The interest rate is then informed by this credit utilization according to a stepwise linear curve as a function of buffer utilization as modeled here.

$$i(t_k) = \begin{cases} i_{min} + \frac{i_{target} - i_{min}}{U_{target}} U(t_k) & \text{if} \quad U(t_k) \le U_{target} \\ i_{target} + \frac{i_{max} - i_{target}}{1 - U_{target}} (U(t_k) - U_{target}) & \text{else} \end{cases} \tag{29}$$

An implementation of this model for demonstration purposes can be found in this Desmos graph https://www.desmos.com/calculator/jsvvcxfqz4.

This curve spans the range between zero and 100% utilization in order to charge interest between a minimum and maximum rate, i_{min} and i_{max}. It features a steep inflection point occurring at a target utilization (U_{target}) where a target rate, i_{target}, is applied.

3.6.2 Maker Rebates

Positions earn a maker rebate, $r_j(t_k) \in (0, 1)$, depending on the ask price of their limit order. Note that a position's ask price in the Exchange module may be updated dynamically which is why the rebate factor $r_j(t_k)$ itself is a discrete process on partition $\mathbf{T}$. The rebate applies to the interest difference operator and proportionally reduces the interest due yielding a new, position specific, interest accumulator

$$I_j(t_k) = I_j(t_{k-1}) + r_j(t_{k-1}) A(t_{k-1}, t_k) \tag{30}$$

Expanding the recursive form we find the position's reduced interest accumulation process in terms of the applicable rebate and interest rate

$$I_j(t_k) = I_j(t_{l^*}) + \sum_{i=l^*+1}^{k} r_j(t_{i-1}) A(t_{i-1}, t_i), \quad \text{with} \quad I_j(t_{l^*}) = I(t_{l^*}) \tag{31}$$

with $I(t_{l^*})$ the global interest accumulator and t_{l^*} the timestamp at which the position was created. It is easy to see that the accumulator remains constant, i.e. no interest accumulation, if the position earns a 100% rebate, $r_j(t_k) = 0, \forall t_k$. On the other hand, the position accumulates interest according to the vault's interest rate $i(t_k)$ if no rebate is earned, or $r_j(t_k) = 1$.

Therewith, we are able to compute a position's accrued interest over two subsequent times t_{k-1} and t_k as

$$a_j(t_{k-1}, t_k) = (I_j(t_k) - I_j(t_{k-1})) n_j(t_{k-1}) \tag{32}$$

3.6.3 Interest Collection

We want to collect interest globally, on the vault level, instead for positions individually in order to allow for a more continuous interest distribution among beneficiaries. Therefore, we note that, according to Equations 30 and 32, interest accrued by the position over the period (t_{k-1}, t_k) is derived as

$$a_j(t_{k-1}, t_k) = r_j(t_{k-1}) A(t_{k-1}, t_k) n_j(t_{k-1}) \tag{33}$$

Building the sum over all positions yields the total interest accrued by the vault

$$a(t_{k-1}, t_k) = \sum_{j \in \mathbf{J}} r_j(t_{k-1}) A(t_{k-1}, t_k) n_j(t_{k-1}) \tag{34}$$

We can factor out the base interest accumulation difference operator and restate the accrued interest for the vault as

$$a(t_{k-1}, t_k) = \bar{r}(t_{k-1}) A(t_{k-1}, t_k) N(t_{k-1}) \tag{35}$$

with the normal debt weighted average rebate

$$\bar{r}(t_{k-1}) = \frac{1}{N(t_{k-1})} \sum_{j \in \mathbf{J}} r_j(t_{k-1}) n_j(t_{k-1}) \tag{36}$$

The normalized total accrued interest, $\frac{a(t_{k-1},t_k)}{N(t_{k-1})}$, directly translates to the difference operator of the global, reduced interest accumulator which can be written recursively as

$$I(t_k) = I(t_{k-1}) + \bar{r}(t_{k-1}) A(t_{k-1}, t_k) \tag{37}$$

Expanding the recursive form shows that, similar to the position accumulator, the global accumulator accrues interest based on the base interest accumulation difference operator reduced by the average maker rebate

$$I(t_k) = I(t_0) + \sum_{i=1}^{k} \bar{r}(t_{i-1}) A(t_{i-1}, t_i), \quad \text{with} \quad I(t_0) = 1 \tag{38}$$

Finally, a note should be taken on the feasibility of computing a vault's average maker rebate. Clearly, simply building the sum over all normal debt weighted position maker rebates does not scale. In order to translate this to an $O(1)$ problem we resort to the iterative updating algorithm

$$\bar{r}(t_k) = \begin{cases} 1 & \text{if} \quad N(t_k) = 0 \\ \frac{1}{N(t_k)} \left(N(t_{k-1}) \bar{r}(t_{k-1}) - r_j(t_{k-1}) n_j(t_{k-1}) + r_j(t_k) n_j(t_k) \right) & \text{else} \end{cases} \tag{39}$$

Using the global rate accumulator we can restate the accrued interest for the vault as

$$a(t_{k-1}, t_k) = (I(t_k) - I(t_{k-1})) N(t_{k-1}) \tag{40}$$

3.7 Failure Modes

We characterize a system failure by a state that puts the system at risk of accumulating bad debt. There are two categories of such failures which are explained in more detail below. It is important to note that these failures always occur on the level of an individual borrow vault and not globally. In other words, global accounting, aka the CDM, is not affected by the failure types and modes discussed here.

3.7.1 Failure Types

Technical Failures

Technical failures are characterized by flaws in the implementation aka software bugs and are generally protected against through thorough audits. Nonetheless, the occurrence of technical failures cannot be excluded and mitigation relies on a manual intervention. This manual intervention is achieved by setting the vault in *Pause* state (see below).

Economic Failures

An economic failure results in a vault being undercollateralized and thus at risk of not being able to offset its debt with deposited collateral. Among other reasons, economic failures may be due to a sharp decline of collateral value, failing liquidations (e.g. due to collateral illiquidity) or targeted market manipulation resulting in an oracle failure.

We are thus able to describe an economic failure as the vault state where the global collateral ratio falls below a certain threshold. More formally, the economic failure state can be expressed as the condition

$$\frac{pA}{IN} < \text{ECR} \tag{41}$$

where A, I, N are the vault's total collateral balance, rate accumulator and total normal debt, p is the collateral's spot price and ECR is the vault's *Emergency Collateral Ratio*. As alluded to previously, the implications of an economic failure are that the system is at risk of the total collateral value falling below the debt borrowed and thus a settlement of this debt resulting in bad debt.

Other Failures

The previous failure types cover most failures but not all. As an example, the characterization of *Economic Failures* does not cover failures resulting in the overestimation of collateral value (e.g. through oracle manipulation). Instead, this failure type as well as all other uncovered types have to be identified and mitigated in an ad-hoc fashion similar to technical failures.

3.7.2 Failure Modes

Failure modes describe the system's response to an identified failure. The objective of these responses always is to secure user funds and protect the system against bad debt.

Pause

The Pause mode temporarily locks the vault such that a failure (technical, economic or other) cannot be exploited by a third party. In the Pause mode the following operations are locked on the vault:

- modifyCollateralAndDebt
- delegate
- undelegate
- liquidate
- createLimitOrder
- cancelLimitOrder
- exchange

Settlement

The Settlement mode permanently locks the vault and initiates an orderly settlement of the vault's total debt. This settlement process distributes a vault's collateral among three groups of stakeholders with decreasing priority

1. Vault Borrowers
2. Vault Delegators
3. External Liquidators

Therefore, settlement proceeds in three epochs of 1 Month each during which the respective stakeholder group is able to claim a proportional share of the collateral by repaying an amount d of outstanding debt. The claimable quantities are determined by stakeholder group as follows:

	Borrowers	Delegators	Liquidators
Debt repaid	d	d	d
Collateral received	$\frac{d}{D_0}A_0$	$\frac{d}{D_1}A_1$	$dp(t), p(0) = \frac{A_2}{D_2}$
Max debt repaid	$\frac{a}{A_0}D_0$	$\frac{s}{S}D_1$	$\frac{A(t)}{p(t)}$, $A(0) = A_2$

where the involved quantities are defined as follows:

- D_0, A_0: the total debt and collateral balance of the vault when entering the settlement mode
- D_1, A_1: the remaining total debt and collateral balance after borrowers claimed their shares
- D_2, A_2: the remaining total debt and collateral balance after delegators claimed their shares
- a is the borrower's collateral balance (sum of position collateral and cash) when the vault entered the settlement mode

- s and S are the delegator's vault shares and total vault shares
- $p(t)$ and $A(t)$ are the collateral auction price and remaining collateral balance at any time t during the auction process

It is important to note that no oracle price is utilized making the settlement process immune against an oracle failure.

3.7.3 Failure Mode Transitions

The following table gives an overview of how the failure modes are entered (or exited) for different vault types and in particular vaults that are owned by governance and such that are not.

	Pause		Settlement	
	Enter	Exit	Enter	Exit
Owned	Owner or Economic Failure	Owner	Owner *when-paused-only*	Never
Not-owned	Economic Failure	Never	Public *when-paused-only*	Never