Rai: A Low Volatility, Trust Minimized Collateral for the DeFi Ecosystem

Stefan C. Ionescu, Ameen Soleimani

May 2020

Abstract. We present a governance minimized, decentralized protocol that automatically reacts to market forces in order to modify the value of its native collateralized asset. The protocol allows anyone to leverage their crypto assets and issue a “reflex bond” that follows the price of its underlying in a dampened way. We outline how bonds can be useful as universal, low volatility collateral which can protect its holders, as well as other decentralized finance protocols, from sudden market shifts. We outline our plans to help other teams launch their own synthetics by leveraging our infrastructure. Moreover, we offer alternatives to current oracle and governance structures that are often found in many DeFi protocols.

Contents

15. Future Research
16. Risks and Mitigation
17. Summary
18. References
19. Glossary

Introduction

Money is one of the most powerful coordination mechanisms humanity leverages in order to thrive. The privilege of managing the money supply has historically been kept in the hands of sovereign leadership and the financial elite while being imposed upon an unwitting general public. Where Bitcoin has demonstrated the potential for a grassroots protest in order to manifest a store-of-value commodity asset, Ethereum gives us a platform to build asset-backed synthetic instruments that can be protected from volatility and used as collateral, or pegged to a reference price and used as a medium-of-exchange for daily transactions, all enforced by the same principles of decentralized consensus.

Permissionless access to Bitcoin for storing wealth and properly decentralized synthetic instruments on Ethereum will lay the foundation for the coming financial revolution, providing those at the fringes of the modern financial system the means to coordinate around building the new one.

In this paper, we introduce a framework for building reflex bonds, a new asset type which will help other synthetics flourish and will establish a base money lego for the entire decentralized finance industry.

Overview of Reflex Bonds

In order to understand reflex bonds, we can compare their behaviour to that of stablecoins. Stablecoins have their redemption price fixed to an external asset such as the US dollar or the Euro (thus, their value should always be one). Reflex Bonds, on the other hand, start at an arbitrary value and are tuned to slowly follow the value of the underlying collateral, actively dampening any short-term market shocks along the way. In Section 4.1 we explain how exactly the redemption price changes in response to market forces.

The floating peg mechanism is not compatible with stablecoins because users would not have a specific reference price to target (the mechanism would not create a synthetic that is easy to understand and transact with such as a USD pegged coin). This is why we believe bonds and stablecoins are separate, albeit complementary assets that can form the foundation of DeFi.

Due to their dampened volatility, reflex bonds allow crypto asset holders to gain exposure to the crypto market without the same scale of risk as holding the actual underlying asset. Moreover, we believe RAI, our first bond, will have immediate utility for other teams issuing synthetics on Ethereum (e.g UMA [1], Synthetix [2] and MakerDAO's MCD [3]) because it gives their users more time to exit their positions.

Design Philosophy

Our design philosophy is to prioritize security, stability and speed of delivery.

Multi-Collateral DAI (MCD) was the natural place to start iterating on a new design for RAI. The system has been heavily audited and formally verified, it has minimal external dependencies and there is an already active community of experts who are familiar with it. To minimize development and communications effort, we want to make only the simplest changes to the original MCD codebase in order to achieve our implementation.

Our most important modifications include the addition of an autonomous rate setter, an Oracle Network Medianizer which is integrated with many independent price feeds and a governance minimization layer meant to isolate the system as much as possible from human intervention.

Monetary Policy Mechanisms

Redemption Rate Feedback Mechanism

The feedback mechanism is meant to change the bond's redemption price according to its deviation from the market price. If the bond's market price is too high compared to its redemption price, then the redemption price will start dropping over time, influenced by a *redemption rate*, discouraging people from holding the asset and encouraging CDP creators to generate more bonds, thus balancing out supply and demand. If the bond's market price is too low compared to redemption, the converse happens.

In the following scenario, we assume that the protocol uses a proportional-integral controller to change the redemption price. PI controllers calculate the current deviation between a target value and the current value of a parameter (the *P* term) and combine this with the sum of past deviations (the *I* term) in order to calculate a

correction, which, in our case, is the redemption rate that modifies the redemption price. The simplified algorithm works as follows:

- The reflex bond is launched with a random redemption price ‘rand’

- At some point, the bond’s market price rises from ‘rand’ to ’rand’ + x. After the feedback mechanism reads the new market price, it calculates a proportional term *p*, which in this case is -1 * ((‘rand’ + x) / ‘rand’). The proportional is negative in order to decrease the redemption price and in turn reprice the bonds so that they become cheaper. This allows CDP creators to generate more debt, even if the price of their collateral remains constant

- Governance can specify a per second integral term *i*. The effect of the integral will be stronger as time passes and the market price remains above redemption

- The mechanism sums the proportional and the integral and calculates a per-second redemption rate *r* that slowly starts to decrease the redemption price. As CDP creators realize they can generate more debt, they will flood the market with more bonds

- After *n* seconds, the mechanism detects that the deviation between the market and redemption prices is negligible (under a specified parameter *noise*). At this point, the algorithm sets r to zero and keeps the redemption price where it is

A similar scenario would happen if the market price was smaller than the redemption one. The rate *r* would be positive so that each CDP’s collateralization would start to decrease and users would be compelled to repay their loans.

In practice, the algorithm will be more robust and we will either make some variables immutable (e.g the *noise* parameter) or there will be strict bounds over what governance can change.

Money Market Setter

In RAI, we plan to keep the borrowing rate (interest rate applied when generating bonds) fixed or capped and only modify the redemption price, thus minimizing the complexity involved in modelling the feedback mechanism. Nevertheless, it is

possible to change both monetary levers with the help of a money market setter.

The money market changes the borrowing rate and the redemption price in a way that incentivizes CDP creators to generate more or less debt. If a bond's market price is above redemption, both rates will start to decrease, whereas if it is below redemption, the rates will increase.

Global Settlement

Global settlement is a method of last resort used to guarantee the redemption price to all reflex bond holders. It is meant to allow both reflex bond holders and CDP creators to redeem system collateral at its net value (amount of bonds per each collateral type, according to the latest redemption price). Anyone can trigger settlement after burning a certain amount of protocol tokens.

Settlement has three main phases:

- **Trigger**: settlement is triggered, users cannot create CDPs anymore, all collateral price feeds and the redemption price are frozen and recorded
- **Process**: process all outstanding auctions
- **Claim**: every reflex bond holder and CDP creator can claim a fixed amount of any system collateral based on the bond's last recorded redemption price

Governance

The vast majority of parameters will be immutable and the inner smart contract mechanics will not be upgradeable unless governance token holders deploy an entirely new system. We chose this strategy because we can eliminate the meta-game where people try to influence the governance process for their own benefit, thus damaging trust in the system. We establish the proper operation of the protocol without putting too much faith in humans (the "bitcoin effect") so that we maximize social scalability and minimize the risks for other developers who will want to use RAI as core infrastructure in their own projects.

For the few parameters that can be changed, we propose the addition of a Restricted Governance Module meant to delay or bound all possible system modifications.

Moreover, we present Governance Ice Age, a permissions registry that can lock some parts of the system from outside control after certain deadlines have passed.

Time Bounded Governance

Time Bounded Governance is the first component of the Restricted Governance Module. It imposes time delays between changes applied to the same parameter. An example is the possibility to change the addresses of the oracles used in the Oracle Network Medianizer (Section 6.2) after at least *T* seconds have passed since the last oracle modification.

Action Bounded Governance

The second component in the Restricted Governance Module is Action Bounded Governance. Every governable parameter has limits on what values it can be set to and how much it can change over a certain period of time. Notable examples are the initial versions of the Redemption Rate Feedback Mechanism (Section 4.1) which governance token holders will be able fine-tune.

Governance Ice Age

The Ice Age is an immutable smart contract that imposes deadlines on changing specific system parameters and on upgrading the protocol. It can be used in the case where governance wants to make sure they can fix bugs before the protocol locks itself and denies outside intervention. Ice Age will verify if a change is permitted by checking the parameter's name and the affected contract's address against a registry of deadlines. If the deadline has passed, the call will revert.

Governance may be able to delay Ice Age a fixed number of times if bugs are found close to the date when the protocol should start to lock itself. For example, Ice Age can only be delayed three times, each time for one month, so that the newly implemented bug fixes are tested properly.

Core Areas Where Governance Is Needed

We envision four areas where governance might be needed, especially in the early versions of this framework:

- **Adding new collateral types**: RAI will be backed only by ETH although some systems will be backed by multiple collateral types and governance will be able to diversify risk over time

- **Changing external dependencies**: oracles and DEXs that the system depends on can be upgraded. Governance can point the system to newer dependencies in order for it to continue functioning properly

- **Fine-tuning rate setters**: early monetary policy controllers will have parameters that can be changed within reasonable bounds (as described by Action and Time Bounded Governance)

- **Migrating between system versions**: in some cases, governance can deploy a new system, give it permission to print protocol tokens and withdraw this permission from an old system. This migration is performed with the help of the Restricted Migration Module outlined below

Restricted Migration Module

The following is a simple mechanism for migrating between system versions:

- There is a migration registry that keeps track of how many different systems the same protocol token covers and which systems can be denied the permission to print protocol tokens in a debt auction

- Every time governance deploys a new system version, they submit the address of the system's debt auction contract in the migration registry. Governance also needs to specify if they will ever be able to stop the system from printing protocol tokens. Also, governance can, at any time, say that one system will always be able to print tokens and thus it will never be migrated from

- There is a cooldown period between proposing a new system and withdrawing permissions from an old one

- An optional contract can be set up so that it automatically shuts down an old system after it is denied printing permissions

The migration module can be combined with an Ice Age that automatically gives specific systems the permission to always be able to print tokens.

Automatic System Shutdown

There are cases that the system can automatically detect and as a result trigger settlement by itself, without the need to burn protocol tokens:

- **Severe Price Feed Delays**: the system detects that one or more of the collateral or bond price feeds have not been updated in a long time

- **System Migration**: this is an optional contract that can shut down the protocol after a cooldown period passes from the moment when governance withdraws the ability of the debt auction mechanism to print protocol tokens (Restricted Migration Module, Section 5.4.1)

- **Consistent Market Price Deviation**: the system detects that the bond's market price has been *x%* deviated for a long time compared to the redemption price

Governance will be able to upgrade these autonomous shutdown modules while still being bounded or until the Ice Age starts to lock some parts of the system.

Oracles

There are three main asset types that the system needs to read price feeds for: the bond, the protocol token and every whitelisted collateral type. The price feeds can be provided by governance led oracles or by already established oracle networks.

Governance Led Oracles

Governance token holders or the core team that launched the protocol can partner with other entities who gather multiple price feeds off-chain and then submit a single transaction to a smart contract that medianizes all data points.

This approach allows for more flexibility on upgrading and changing the oracle infrastructure although it comes at the expense of trustlessness.

Oracle Network Medianizer

An oracle network medianizer is a smart contract that reads prices from multiple sources which are not directly controlled by governance (e.g Uniswap V2 pool between a bond collateral type and other stablecoins) and then medianizes all the results. ONM works as follows:

- Our contract keeps track of whitelisted oracle networks it can call in order to request collateral prices. The contract is funded by part of the surplus the system accrues (using the Surplus Treasury, Section 11). Each oracle network accepts specific tokens as payment so our contract also keeps track of the minimum amount and the type of tokens needed for each request

- In order to push a new price feed in the system, all the oracles need to be called beforehand. When calling an oracle, the contract first swaps some stability fees with one of the oracle's accepted tokens. After an oracle is called, the contract tags the call as "valid" or "invalid". If a call is invalid, the specific faulty oracle cannot be called again until all the other ones are called and the contract checks if there is a valid majority. A valid oracle call must not revert and it must retrieve a price that has been posted on-chain sometime in the last *m* seconds. "Retrieve" means different things depending on each oracle type:

    - For pull based oracles, from which we can get a result right away, our contract needs to pay a fee and directly fetch the price

    - For push based oracles, our contract pays the fee, calls the oracle and needs to wait a specific period of time *n* before calling the oracle again in order to get the requested price

- Every oracle result is saved in an array. After every whitelisted oracle is called and if the array has enough valid data points to form a majority (e.g the contract received valid data from 3/5 oracles), the results are sorted and the contract chooses the median

- Whether the contract finds a majority or not, the array with oracle results is cleared and the contract will need to wait *p* seconds before starting the entire process all over again

Oracle Network Backup

Governance can add a backup oracle option that starts to push prices in the system if the medianizer cannot find a majority of valid oracle networks several times in a row.

The backup option must be set when the medianizer is deployed as it cannot be changed afterwards. Furthermore, a separate contract can monitor if the backup has been replacing the medianization mechanism for too long and automatically shut down the protocol.

Collateralized Debt Positions

In order to generate bonds, anyone can deposit and leverage their crypto collateral inside Collateralized Debt Positions. While a CDP is opened, it will continue accruing debt according to the deposited collateral's borrowing rate. As the CDP creator pays back their debt, they will be able to withdraw more and more of their locked collateral.

CDP Lifecycle

There are four main steps needed for creating reflex bonds and subsequently paying back a CDP's debt:

- Deposit collateral in the CDP

  The user first needs to create a new CDP and deposit collateral in it.

- Generate bonds backed by the CDP's collateral

The user specifies how many bonds they want to generate. The system creates an equal amount of debt that starts to accrue according to the collateral's borrowing rate.

- Pay back the CDP debt

  When the CDP creator wants to withdraw their collateral, they have to pay back their initial debt plus the accrued interest.

- Withdraw collateral

  After the user pays back some or all of their debt, they are allowed to withdraw their collateral.

CDP Liquidation

In order to keep the system solvent and cover the value of the entire outstanding debt, each CDP can be liquidated in case its collateralization ratio falls under a certain threshold. Anyone can trigger a liquidation, in which case the system will confiscate the CDP's collateral and sell it off in a *collateral auction*.

Liquidation Insurance

In one version of the system, CDP creators can have the option to choose a *trigger* for when their CDPs get liquidated. Triggers are smart contracts that automatically add more collateral in a CDP and potentially save it from liquidation. Examples of triggers are contracts that sell short positions or contracts that communicate with insurance protocols such as Nexus Mutual [4].

Another method to protect CDPs is the addition of two different collateralization thresholds: *safe* and *risk*. CDP users can generate debt until they hit the safe threshold (which is higher than risk) and they only get liquidated when the CDP's collateralization goes below the risk threshold.

Collateral Auctions

To start a collateral auction, the system needs to use a variable called *collateralToSell* in order to determine the amount of collateral sold in each auction and the corresponding amount of debt to be covered. A *liquidation penalty* will be applied to every auctioned CDP.

Collateral Auction Parameters

Parameter Name	Description
bidIncrease	Minimum increase in the next submitted bid
bidDuration	How long the auction lasts after a new bid is submitted (in seconds)
totalAuctionLength	Total length of the auction (in seconds)
auctionsStarted	Number of auctions started until now
bidToMarketPriceRatio	Minimum size of the first bid compared to the collateral price multiplied by the amount of collateral being sold

Collateral Auction Mechanism

Each collateral auction has two distinct phases:

`increaseBidSize(uint auctionId, uint amountToBuy, uint bid)`: in the first phase, anyone can submit a higher bid for the same amount of collateral. The first bid needs to be higher or equal to *bidToMarketPriceRatio * collateralMarketPrice / 100.* Each subsequent bid needs to be higher or equal to *previousBid * bidIncrease / 100.* The first phase can last maximum *totalAuctionLength* seconds since the auction has started or it can end after *bidDuration* seconds have passed since the latest bid, with no other bid submitted in the meantime

`decreaseSoldAmount(uint auctionId, uint amountToBuy, uint bid)`: in the second phase, bidders accept lower amounts of collateral in return for paying the winning bid amount from the first phase. Similar to the first phase, the auction will end after maximum *totalAuctionLength* seconds have passed since it has started

If no bids were placed, the auction can be restarted. Otherwise the auction will settle and the winning bidder will receive the sold collateral.

Debt Auctions

In the scenario where a collateral auction cannot cover all the bad debt in a CDP and if the system does not have any surplus reserves, anyone can trigger a debt auction. Debt auctions are meant to mint more protocol tokens (Section 10) and sell them for bonds that can nullify the system's remaining bad debt.

In order to start a debt auction, the system needs to use two parameters:

- initialDebtAuctionAmount: the initial amount of protocol tokens to mint post-auction

- debtAuctionBidSize: the initial bid size (how many bonds must be offered in exchange for *initialDebtAuctionAmount* protocol tokens)

Autonomous Debt Auction Parameter Setting

The initial amount of protocol tokens minted in a debt auction can either be set through a governance vote or it can be automatically adjusted by the system. An automated version would need to be integrated with oracles (Section 6) from which the system would read the protocol token and reflex bond market prices. The system would then set the initial amount of protocol tokens (*initialDebtAuctionAmount*) that will be minted for *debtAuctionBidSize* bonds. *initialDebtAuctionAmount* can be set at a discount compared to the actual PROTOCOL/BOND market price in order to incentivize bidding.

Debt Auction Parameters

Parameter Name	Description
amountSoldIncrease	Increase in the amount of protocol tokens to be minted for the same amount of bonds
bidDecrease	Next bid's minimum decrease in the accepted amount of protocol tokens for the same amount of bonds
bidDuration	How long the bidding lasts after a new bid is submitted (in seconds)
totalAuctionLength	Total length of the auction (in seconds)
auctionsStarted	How many auctions have started until now

Debt Auction Mechanism

As opposed to collateral auctions, debt auctions only have one stage:

`decreaseSoldAmount(uint id, uint amountToBuy, uint bid)`: decrease the amount of protocol tokens accepted in exchange for a fixed amount of bonds.

The auction will be restarted if it has no bids placed. Every time it restarts, the system will offer more protocol tokens for the same amount of bonds. The new protocol token amount is calculated as *lastTokenAmount* * *amountSoldIncrease* / 100. After the auction settles, the system will mint tokens for the highest bidder.

Protocol Tokens

As described in earlier sections, each protocol will need to be protected by a token that is minted through debt auctions. Apart from protection, the token will be used to govern a few system components, unless we find a safe way to remove governance altogether. Also, the protocol token supply will gradually be reduced with the use of surplus auctions. The amount of surplus that needs to accrue in the

system before extra funds are auctioned is called the *surplusBuffer* and it is automatically adjusted as a percentage of the total debt issued.

Insurance Fund

Apart from the protocol token, governance can create an insurance fund that holds a wide array of uncorrelated assets and which can be used as a backstop for debt auctions.

Surplus Auctions

Surplus auctions sell stability fees accrued in the system for protocol tokens that are then burned.

Surplus Auction Parameters

Parameter Name	Description
bidIncrease	Minimum increase in the next bid
bidDuration	How long the auction lasts after a new bid is submitted (in seconds)
totalAuctionLength	Total length of the auction (in seconds)
auctionsStarted	How many auctions have started until now

Surplus Auction Mechanism

Surplus auctions have a single stage:

`increaseBidSize(uint id, uint amountToBuy, uint bid)`: anyone can bid a higher amount of protocol tokens for the same amount of bonds (surplus). Every new bid needs to be higher than or equal to *lastBid* * *bidIncrease* / 100. The auction will end after maximum *totalAuctionLength* seconds or after *bidDuration* seconds have passed since the latest bid and no new bids have been submitted in the meantime.

An auction will restart if it has no bids. On the other hand, if the auction has at least one bid, the system will offer the surplus to the highest bidder and will then burn all the gathered protocol tokens.

Surplus Bonds Management

Every time a user generates bonds and implicitly creates debt, the system starts applying a borrowing rate to the user's CDP. The accrued interest is pooled in two different smart contracts:

- The *accounting engine* used to trigger debt (Section 9.2) and surplus (Section 10.1) auctions
- The *surplus treasury* used to fund core infrastructure components and incentivize external actors to maintain the system

The surplus treasury is in charge of funding three core system components:

- Oracle module (Section 6). Depending on how an oracle is structured, the treasury either pays governance whitelisted, off-chain oracles or it pays for calls toward oracle networks. The treasury can also be set up to pay the addresses that spent gas to call an oracle and update it

- Market maker module (Section 12). The treasury can provide liquidity to DEXs where the reflex bonds are traded. This way the system can always read fresh price feeds from sources such as Uniswap V2 [5]

- In some cases, independent teams that maintain the system. Examples are teams who whitelist new collateral types or fine tune the system's rate setter (Section 4.1)

The treasury can be set up so that some surplus recipients will automatically be denied funding in the future and others can take their place.

Market Maker Module

The market maker module is meant to provide liquidity to BOND/COLLATERAL pairs in the early days post-launch so that our oracles can read fresh BOND feeds from

sources such as Uniswap V2. Market making can either be done programmatically by executing trades with the help of a smart contract and surplus coming from the treasury (Section 11) or by governance and other external actors.

In case market making is done with the help of a contract, governance can set a deadline on when the treasury will stop providing funds to it so that resources can be better used in other parts of the system.

External Actors

The system depends on external actors in order to function properly. These actors are economically incentivized to participate in areas such as auctions, global settlement processing, market making and updating price feeds in order to maintain the system's health.

We will provide initial user interfaces and automated scripts to enable as many people as possible to keep the protocol secure.

Addressable Market

We see RAI as being useful in two main areas:

- **Portfolio diversification**: investors use RAI in combination with a protocol such as Balancer [6] in order to create their own index fund. RAI offers them dampened exposure to an asset like ETH without the whole risk of actually holding ether

- **Collateral for synthetic assets**: RAI can offer protocols such as UMA, MakerDAO and Synthetix a lower exposure to the crypto market and give users more time to exit their positions in the case of scenarios such as Black Thursday from March 2020 when millions of dollars worth of crypto assets were liquidated

Future Research

To push the boundaries of decentralized money and bring further innovation in decentralized finance, we will continue to look for alternatives in core areas such as governance minimization and liquidation mechanisms.

We first want to lay the groundwork for future standards around protocols that lock themselves from outside control and for true “money robots” which adapt in response to market forces. Afterwards, we invite the Ethereum community to debate and design improvements around our proposals with a specific focus on collateral and debt auctions.

Risks and Mitigation

There are several risks involved in developing and launching a reflex bond, as well as subsequent systems that are built on top:

- **Smart contract bugs**: the greatest risk posed to the system is the possibility of a bug that allows anyone to extract all the collateral or locks the protocol in a state it cannot recover from. We plan to have our code reviewed by multiple security researchers and launch the system on a testnet before we commit to deploying it in production

- **Oracle failure**: we will aggregate feeds from multiple oracle networks and there will be strict rules in place for upgrading only one oracle at a time so that malicious governance cannot easily introduce false prices

- **Collateral black swan events**: there is the risk of a black swan event in the underlying collateral which can result in a high amount of liquidated CDPs. Liquidations may not be able to cover the entire outstanding bad debt and so the system will continuously change its surplus buffer in order to cover a decent amount of issued debt and withstand market shocks

- **Improper rate setter parameters**: autonomous feedback mechanisms are highly experimental and may not behave exactly like we predict during simulations. We plan to allow governance to fine-tune this component (while

still being bounded) in order to avoid unexpected scenarios

- **Failure to bootstrap a healthy liquidator market**: liquidators are vital actors that make sure all issued debt is covered by collateral. We plan to create interfaces and automated scripts so that as many people as possible can participate in keeping the system secure.

Summary

We have proposed a protocol that progressively locks itself from human control and issues a low volatility, collateralized asset called a reflex bond. We first presented the autonomous mechanism which incentivizes bond CDP creators to generate more or less debt and then described how several smart contracts can limit the power that token holders have over the system. We outlined a self-sustaining scheme for medianizing price feeds from multiple independent oracle networks and then finished by presenting the general mechanism for minting bonds and liquidating CDPs.

References

[1] “UMA: A Decentralized Financial Contract Platform”, https://bit.ly/2Wgx7E1

[2] Synthetix Litepaper, https://bit.ly/2SNHxZO

[3] “The Maker Protocol: MakerDAO’s Multi Collateral Dai (MCD) System”, https://bit.ly/2YL5S6j

[4] H. Karp, R. Melbardis, “A peer-to-peer discretionary mutual on the Ethereum blockchain”, https://bit.ly/3du8TMy

[5] H. Adams, N. Zinsmeister, D. Robinson, “Uniswap V2 Core”, https://bit.ly/3dqzNEU

[6] F. Martinelli, N. Mushegian, “A non-custodial portfolio manager, liquidity provider, and price sensor”, https://bit.ly/3bi54s2

Glossary

Reflex bond: a collateralized asset that dampens the volatility of its underlying

RAI: our first reflex bond

Redemption Price: the price that the system wants the bond to have. It changes, influenced by a redemption rate (computed by RRFM), in case the market price is not close to it. Meant to influence CDP creators to generate more or pay back some of their debt

Borrowing Rate: annual interest rate applied to all CDPs that have outstanding debt

Redemption Rate Feedback Mechanism (RRFM): an autonomous mechanism which compares the market and redemption prices of a reflex bond and then computes a redemption rate which slowly influences CDP creators to generate more or less debt (and implicitly tries to minimize the market/redemption price deviation)

Money Market Setter (MMS): a mechanism similar to RRFM which pulls multiple monetary levers at once. In the case of reflex bonds, it modifies both the borrowing rate and the redemption price

Oracle Network Medianizer (ONM): a smart contract that pulls prices from multiple oracle networks (which are not controlled by governance) and medianizes them if a majority (e.g 3 out of 5) returned a result without throwing

Restricted Governance Module (RGM): a set of smart contracts that bound the power that governance tokens holders have over the system. It either enforces time delays or limits the possibilities that governance has to set certain parameters

Governance Ice Age: immutable contract that locks most components of a protocol from outside intervention after a certain deadline has passed

Accounting Engine: reflex bond system component which triggers debt and surplus auctions. It also keeps track of the amount of currently auctioned debt, unactioned bad debt and the surplus buffer

Surplus Buffer: amount of interest to accrue and keep in the system. Any interest

accrued above this threshold gets sold in surplus auctions which burn protocol tokens

Surplus Treasury: contract which gives permission to different system modules to withdraw accrued interest (e.g ONM for oracle calls, Market Maker Module for adding liquidity to BOND/COLLATERAL pairs)

Market Maker Module: a contract that adds liquidity to BOND/COLLATERAL pairs and bootstraps bond liquidity in the early days post-launch