Deco Protocol: Decomposing Yield-Bearing Assets into Fixed Rate and Yield Rate Instruments in DeFi

Vamsi Alluri (hi@vamsiraju.com)

August 2021

Abstract

The Decentralized Finance (DeFi) ecosystem requires the ability to hedge against rate volatility of all yield bearing assets in order to serve the more conservative user and expand the value proposition of this nascent industry to a wider audience. The DECO Protocol is a novel decentralized protocol which decomposes a yield bearing asset into a zero-yield and a pure-yield instrument. In this paper, the protocol is implemented for various yield tokens deployed for ETH, DAI, USDC and BTC. The Protocol makes it possible for a token holder to receive a fixed savings rate over a fixed term by purchasing the zero-yield, and for the risk-prone to take on the risk of rate volatility by purchasing a pure-yield, fixed term instrument. Deco is a simple and flexible protocol that can be attached to any yield protocol to decompose its yield token into fixed term assets which benefits both the protocol as well as the user.

Contents

List of Figures

1 Introduction

Protocols that can generate yield for base assets such as ETH, DAI, USDC, and BTC have proliferated over the past year on the Ethereum network and other blockchains. Yield protocols such as Automated Market Makers (AMMS) [18], Staking protocols (ETH) [1], and other lending protocols [14] [12] promote the deposit of base assets for additional yield, and all will benefit from the Deco Protocol. Yield protocols are easy to build on top of, by integrating with yield token balances such as cDAI, yDAI, aUSDC, et cetera. Yield protocols receive deposits of base assets, and in exchange issue additional tokens to depositors. We refer to these tokens from the yield protocol as *yield tokens*. Yield token holders expect to withdraw both their original base asset deposit, as well as the additional yield earned over the *term* of the deposit period.

Yield protocols are usually designed to be responsive to market conditions, and to generate the best possible return for the yield token holder. In doing so, the protocols are generally unable to promise a *fixed rate* of yield over a specific time period and are able to permit the user to withdraw the deposit at any time. Deco makes it possible for a user to partition into two assets and trade them freely; one token representing the original base asset deposit, and the other token representing an ongoing accrual of interest in the yield token. Deco brings rate stability to yield protocols by creating a fixed maturity version of a specific yield token, providing the holder a *fixed rate* over a stated period. On the other hand, the yield portion may be held by a less risk averse holder, who will receive the variable yield over a fixed term. Fixed rate assets are useful to those seeking a fixed yield, but also allows market participants with a different outlook on future rates to choose a floating yield counterpart. Deco is a simple and flexible protocol that can be attached to any yield token to decompose it into a *zero-yield* asset which guarantees a *fixed rate* and another *pure-yield* asset which promises only the yield.

Deco is compelling because it gives users flexibility when dealing with assets that come with a fixed maturity date attached. Additionally, users can choose from a diverse selection of liquidity sources based upon their individual needs.

Deco also mitigates risk to the underlying yield protocol and its users; it always affords the free movement of yield token deposits in and out of the core Deco protocol during the contract period, even before the *maturity* date. Consequently, yield protocols are not at risk of liquidity constraints when integrating with Deco.

Unlike other fixed rate protocols, Deco contains advanced safety mechanisms that can quickly unwind a Deco *instance* and unlock and return all yield token deposits to users at any time if the need arises. With *emergency shutdown* as a central feature, Deco keeps both users as well as the attached yield protocol safe.

While the designed protocol works in a fully trustless manner for most yield protocols, governance still plays an irreplaceable role in keeping users safe, increasing efficiency of the protocol, and serving as a trustworthy conduit for information in instances of ambiguity. For instance, in the event that the protocol is unable to easily read or compute information directly from the chain, a Deco instance can employ both trustless (governance free) or trusted (governance inclusive) functions as required for redundancy.

2 Protocol

2.1 Yield Mechanics

2.1.1 Yield Distribution

Deco supports a diverse number of yield protocol structures. Yield token protocols may use different mechanisms internally, including the method of yield accrual and distribution, not just the way the yield is distributed, but also in how the yield is accrued from an accounting standpoint, and then passed on to the *base asset* depositor.

Blockchain transactional throughput is a rate limiting factor in yield distribution and does not permit yield to be distributed individually to each depositor, even at large time intervals. Individualized yield distribution is not currently practical as it is cost prohibitive even when servicing a modest number of depositors.

Rather than create individual distributions, yield tokens distribute yield to all depositors simultaneously. This is accomplished by continuously increasing a user's share of total reserves. The user's share is a static claim on a growing pool. In other words, depositors receive a static yield token balance, which remains unchanged from initial deposit until withdrawal. What grows in response to the yield is the share of reserves this static balance claims. With this design, yield is distributed to all depositors, even every *block*. Only one transactional update is required to increase the common reserves for everyone. This mechanism results in a scenario where yield tokens are priced in terms of the *base asset*, which continuously increases over time. [5]

Deco utilizes the following two components of a Yield protocol: a) yield token balance, and b) its *price* vs the *base asset* increasing over time, as can be seen in Figure 1.

Deco is a general protocol and applies to a wide variety of yield tokens. Deco relies on and tracks the external market *price* of the yield token with regard to the yield protocol's own *base asset* deposit. Depending on the yield protocol, sometimes earned yield could vary based on the size of the withdrawal amount [10], with further slippage as a risk, but Deco can determine an appropriate price in these scenarios with the aid of governance.

2.2 General Fixed Rate Protocol Requirements

Fixed rate layers for yield tokens should have the following characteristics.

- **Guaranteed Fixed Rate.** A *fixed rate* protocol must be able to guarantee a *fixed rate* irrespective of market conditions or the actions (or inaction) of other users. A *zero-yield* structure gives users a guaranteed *fixed rate* with an imputed interest. The fixed interest comes from a discounted purchase price to face value and redemption of the *zero* at full *face value* upon maturity. [11]

- **No Solvency Issues** Some competing *fixed rate* protocols offer a guaranteed *fixed rate* to buyers by simply holding onto the yield component of the asset. [7] These protocols bear the risk of actual yield (rate volatility) dipping below the promised fixed yield over the duration of *fixed rate* term. Solvency issues will occur when the yield earned is insufficient to cover the guaranteed *fixed rate*. Deco assures solvency.

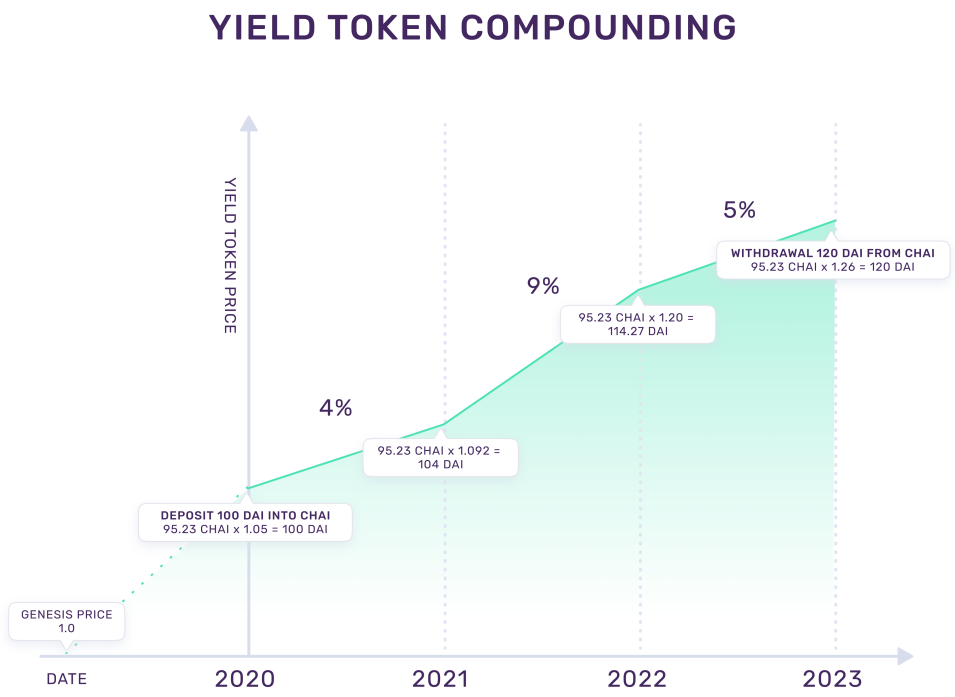

Figure 1: Price of a yield token (CHAI) compounding at various rates over time vs its *base asset* (DAI).

- **Zero Redemption Risk.** In order to ensure that there is no redemption risk, a *fixed rate* protocol must retain all of the yield token deposit, thus ensuring that it can settle all its obligations at any time, and not just at maturity. While yield tokens can face solvency issues, this will not impact a *fixed rate* protocol which is able to settle its obligations with the yield token deposits it holds. Promises regarding actual yield, and the ability to withdraw the *base asset* deposit itself at a future date are unsecured promises, currently being made by some *fixed rate* protocols [17] [6]; in contrast, Deco has zero redemption risk by always retaining yield token deposits.

- **Rate Choice.** It is important to create assets that are flexible enough to allow a market to form around the expectations of future rates. A *fixed rate* protocol must offer more than one *fixed rate* to its users. Users should be given the ability to enter into agreements with counterparties based on a mutually agreed upon *fixed rate*. Deco provides this flexibility.

- **Minimal Burden.** A *fixed rate* protocol should not add additional constraints to the operation of the underlying Yield protocol. For example, *fixed rate* protocols should allow early redemption to ensure that the yield protocols are not adversely impacted by the *base asset* deposits being locked until a specified maturity date. A protocol must enable sufficient liquidity so that all early withdrawals can be accommodated prior to the stated maturity date.

- **Instant Shutdown.** A governance mechanism must be able to shut down the *fixed rate* protocol when unexpected market conditions create a concern of solvency. Under such circumstances, redemptions must still be allowed on assets which have not yet reached their maturity date. Of course, the redemption must be at the then fair market value. Deco provides this flexibility.

2.3 Fraction Snapshots

Yield tokens typically do not store historical yield values on-chain, but this historical information is imperative to settling *fixed maturity* assets at some future date. Before we discuss the core lifecycle from *issuance* to *settlement* in Deco, we will describe how it is able to maintain a record of historical yield data over time for use in the settlement of transactions on maturity.

The difference in the *base asset price* between time of deposit and time of valuation, tells us the amount of yield a user has earned. As a reminder, a user deposits a fixed quantity of yield token into Deco. Yield is then accounted for by an increase in the *base asset price*, while *base asset* quantity remains stable (Figure 2). The *price* is therefore denoted in terms of the *base asset*. We have created a mechanism which allows anyone, (when yield token *price* information is available to be read on-chain), or governance (trusted *price* information input), or both, to take a *snapshot* of the yield token *price* and store it within Deco at various timestamps.

Currently, *issuance* and *settlement* timestamps are set for all fixed-maturity asset balances at their *issuance*. Various functions in the Deco protocol can look up the *price* information for these timestamps from a common *snapshot* mapping and use it for settlement any time after the *price* information is recorded.

The Deco snapshot mechanism also operates to handle *fixed-term* assets issued for many different *maturity* timestamps from a single deployed contract. Assets that share the same timestamps are able to automatically utilize the same stored *price* information, which means that a single yield *price snapshot* at a timestamp can be used to settle all balances relying on the timestamp at any point in the future.

Let us take the yield token CHAI [16] which is the yield token wrapper for the base asset DAI [4] as an example. Let us assume ρ_t denotes the price of CHAI at time t. ρ_t varies with time as yield is earned. At time t, if the yield protocol receives B amount of *base asset* deposit (DAI) from a user, it will issue Y amount of yield tokens (CHAI) back to them according to the equation,

$$Y = \frac{B}{\rho_t} \tag{1}$$

Accumulated Interest rate return over time is reflected by an increase in *price* of the yield token relative to the *base asset* over this duration. The price of the yield token at withdrawal then represents the amount for which the yield token can be redeemed. Yield earned is the difference between the yield token value at withdrawal and deposit, and may be calculated as follows,

$$\text{Yield} = (\text{Y} \times \rho_{t2}) - (\text{Y} \times \rho_{t1}) \tag{2}$$

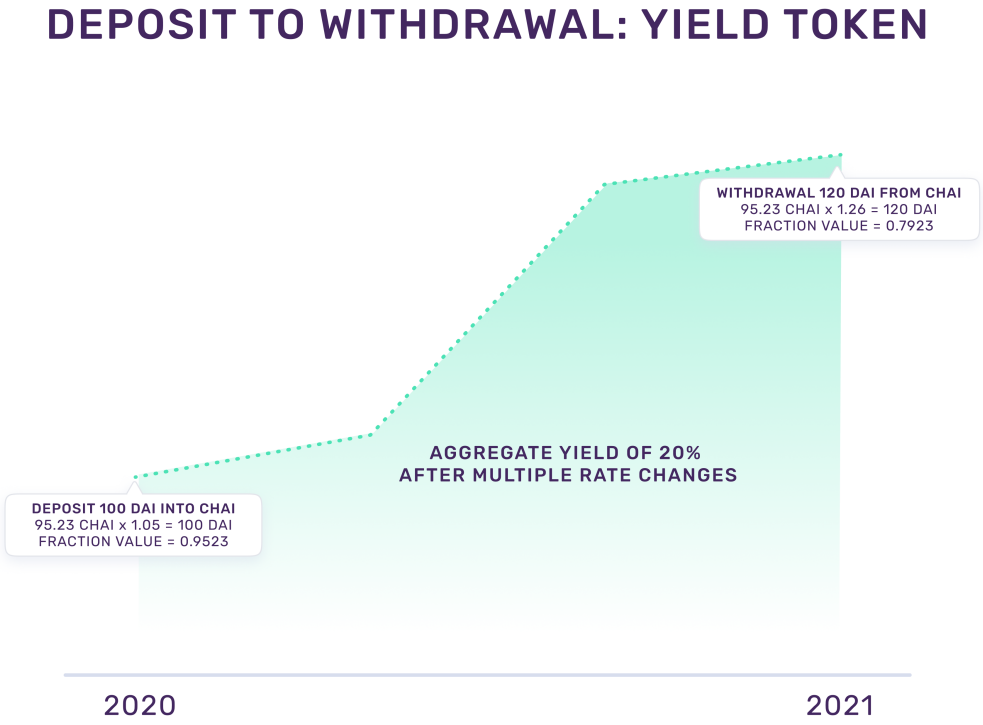

Figure 2: Yield token performance between deposit and withdrawal from the yield token protocol, in this case, CHAI as the yield token and DAI as the *base asset*.

where Y is the amount of yield tokens, ρ_{t2} is the price of the yield token at withdrawal time, and ρ_{t1} is the price of the yield token at deposit time.

We shall refer to the inverse of this yield token *price*, $1/\rho_t$, as a *"Fraction."* *Fraction* Values are the *price* of a yield token priced in terms of the *base asset*. *Fraction* values start at 1 and decrease continuously when yield accrues. No change in *fraction* value between deposit and maturity suggests no yield accrued, while a smaller fraction value at maturity suggests the *price* of the *base asset* has increased relative to the fixed yield token amount and therefore yield has accrued. Within the smart contract, we have chosen to store the *fraction* value, instead of the yield token *price*, as an optimization to reduce on-chain division operations during both *issuance* and *settlement* of Deco assets. We will discuss the exact method in which *fraction* values are used within Deco in later sections.

On-chain yield token pricing may be unavailable for some protocols, especially those utilizing a pooled deposit model, for example Uniswap LP tokens. Deco utilizes two functions to be implemented to store these *fraction* snapshots: *snapshot* and *insert* to handle both scenarios.

Anyone can execute a public *snapshot* function to read and store the latest *fraction* value at the current timestamp when an on-chain *price* value is available.

When an on-chain *price* value is unavailable, or when it must be adjusted before it can be stored, a governance mechanism for the Deco instance can use the restricted *insert* function to input the *fraction* value directly at any timestamp.

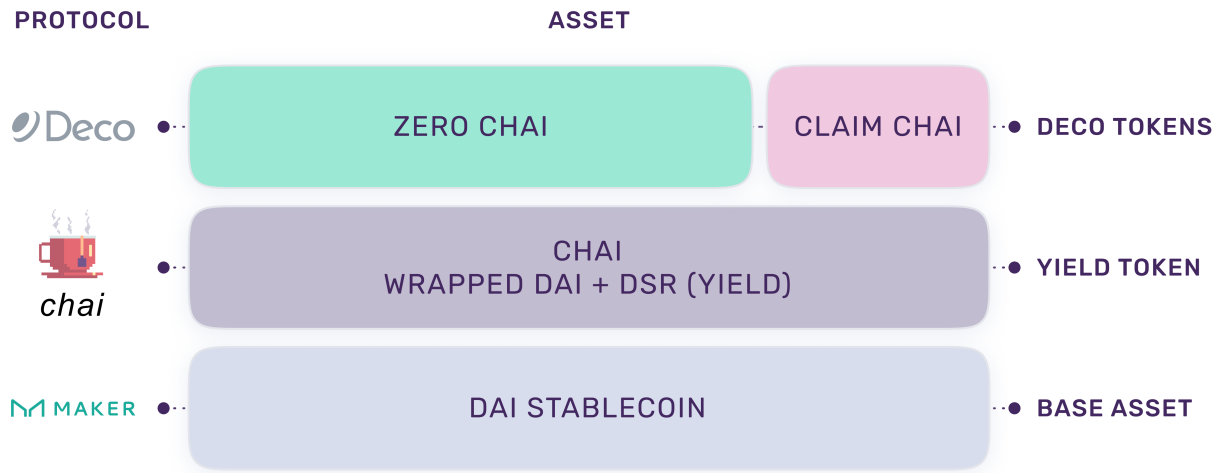

Figure 3: Protocol Overview.

2.4 Core Lifecycle

Deco is able to create the *zero-yield* version for yield tokens by moving the risk of earning the volatile rate to another *pure-yield* asset whose value will track exactly the yield earned. The core lifecycle of Deco consists of *issuance* (mint), which brings two new assets into circulation, *zero* and *claim*, and *settlement* (burn), which removes both components from circulation. Asset balances are always attached to a *maturity* date at *issuance* which establishes a bench-mark future settlement date. Deco permits Settlement prior to the bench-mark settlement date, or even after with continued accrual of interest.

To better describe the mechanics of *issuance* and *settlement*, we will use the CHAI token and a Deco protocol deployment for it as reference. CHAI can be thought of as a yield token wrapper for the yield produced by the Dai Savings Rate (DSR) [3] in the Maker protocol. The assets Deco issues will be referred to as ZERO-CHAI and CLAIM-CHAI (Figure 3).

Below we examine the lifecycle of a *zero*, using DAI, CHAI and ZERO-CHAI as an example (Figure 4). Here a user purchases 100 ZERO-CHAI from the market for 80.95 CHAI. The corresponding quantity of CHAI within Deco is 95.23, with a price of 1.05 DAI/CHAI in the pool values it at 100 DAI (95.23 CHAI $\times$ 1.05 DAI/CHAI). The user expects to *redeem* this 100 ZERO-CHAI in 1 year for 100 DAI, with an expected return of 15 DAI or a 17.6% annualized rate. At redemption, the user redeems the 100 ZERO-CHAI for 79.36 CHAI from Deco, since the *price* of CHAI is 1.26 DAI/CHAI in the pool at the final time point (Figure 4). The user then withdraws 79.36 CHAI for 100 DAI from the CHAI smart contracts. The remaining portion of the initial CHAI deposit (95.23 CHAI - 79.36 CHAI) is reserved for the CLAIM-CHAI token redemption, and represents the yield accrued during the maturation window (Figure 5).

Deco manages all its internal accounting using the yield token balance. At issuance, users exchange their yield token balance for a corresponding amount of *zero* and *claim* balances. Later, users are also able to exchange these two assets back for their due portion of the yield token deposit and accrued yield. How the original yield token balance deposit gets split

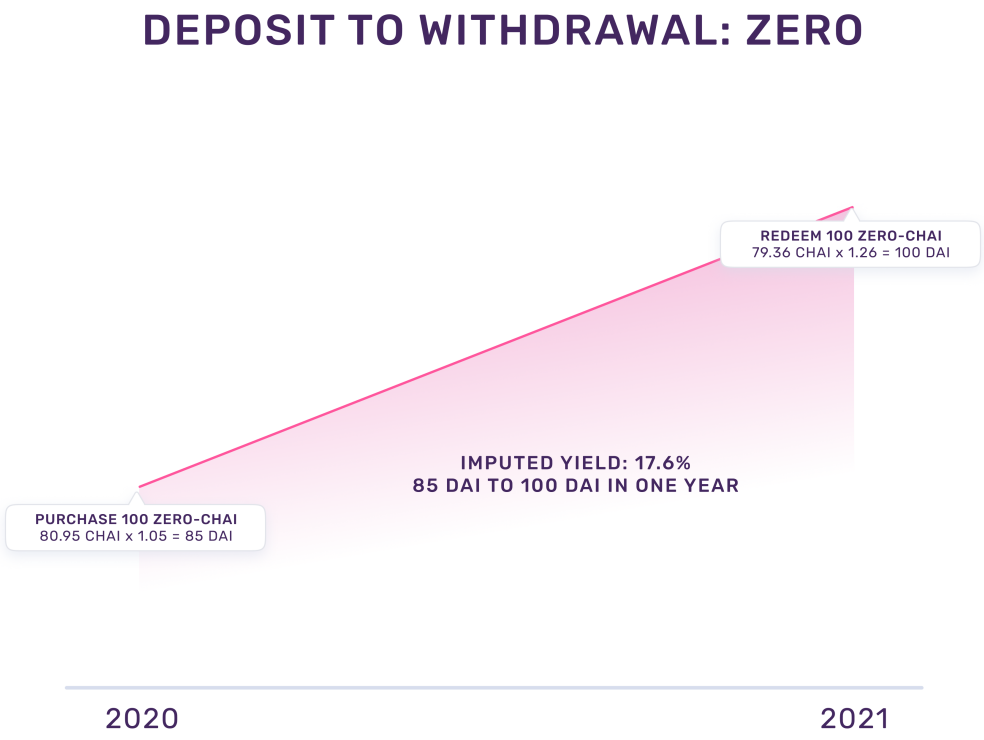

Figure 4: Zero token performance between its purchase and redemption at maturity.

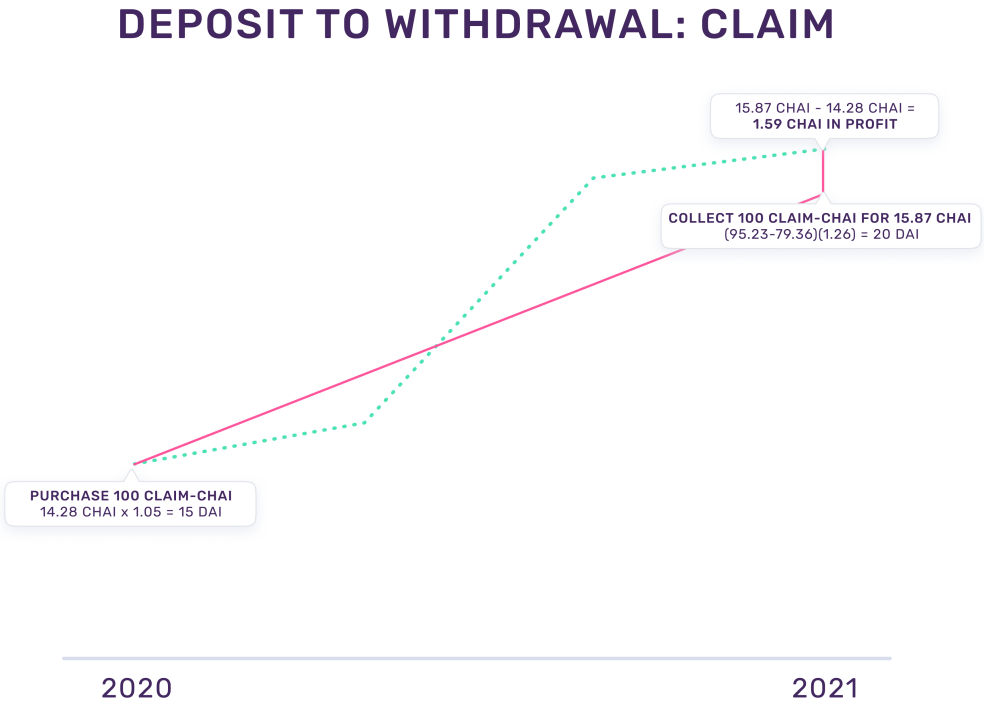

Figure 5: Claim token performance between its purchase and when it collects all yield earned between issuance and maturity.

Figure 6: Zeros and Claims with Yield.

between them after maturity is decided based on the observed yield (change in *price* of yield token) earned by the yield token balance between *issuance* and *settlement*. In other words, if a yield is high, the *zero* will receive a smaller proportion of the original deposit, if the yield is low, the *zero* will receive a larger portion.

An amount of yield tokens sufficient to claim exactly the yield earned in *base asset* will go to the *claim* holder, and the *zero* holder will receive the remaining yield token balance, which will be sufficient to claim the original *glsnotionalamount* of *base asset* that the yield token had at issuance.

For example, let's assume 100 DAI worth of CHAI was deposited to issue ZERO-CHAI and CLAIM-CHAI balances, and about 20 DAI in additional yield was accrued to the deposit. Upon maturity, the ZERO-CHAI balance holder will be able to exchange their balance with Deco to receive enough CHAI that can claim the principal amount deposited which is 100 DAI. Similarly, the CLAIM-CHAI balance holder can exchange it to receive enough CHAI that can claim the yield amount portion of 20 DAI. The *snapshots* of *fraction* values derived from the *price* of CHAI stored within Deco at both *issuance* and *maturity* timestamps allow the entire CHAI balance deposited at issuance to be split between the *zero* and *claim* balance holders which then allows them to withdraw 100 DAI and 20 DAI respectively from the CHAI yield protocol (Figure 6).

We can make two general observations about the performance of these assets under various market conditions. First, in the most extreme scenario, when no yield is earned, the deposited yield token balance at issuance will always be sufficient to allow the *zero* holder to *redeem* the entire amount owed at maturity. Also, the entire risk of yield volatility and actual yield earned by the yield protocol in the time period between *issuance* and *maturity* is borne by the *claim* holder, who could come out with a profit or loss based on their own rate forecast (Figure 7). Deco protocol can settle all its future obligations to both *zero* and *claim* holders without having to rely on any collateral liquidation mechanisms to protect users and is designed to avoid *solvency* issues.

The ideal path will be for the *zero* asset holder to earn slightly less than the market yield over the time period in exchange for the certainty of *fixed-yield*, while the *claim* asset holder earns slightly more yield than what was originally paid to purchase the asset and for taking on this risk. These dynamics allow a cooperative system to emerge between professional

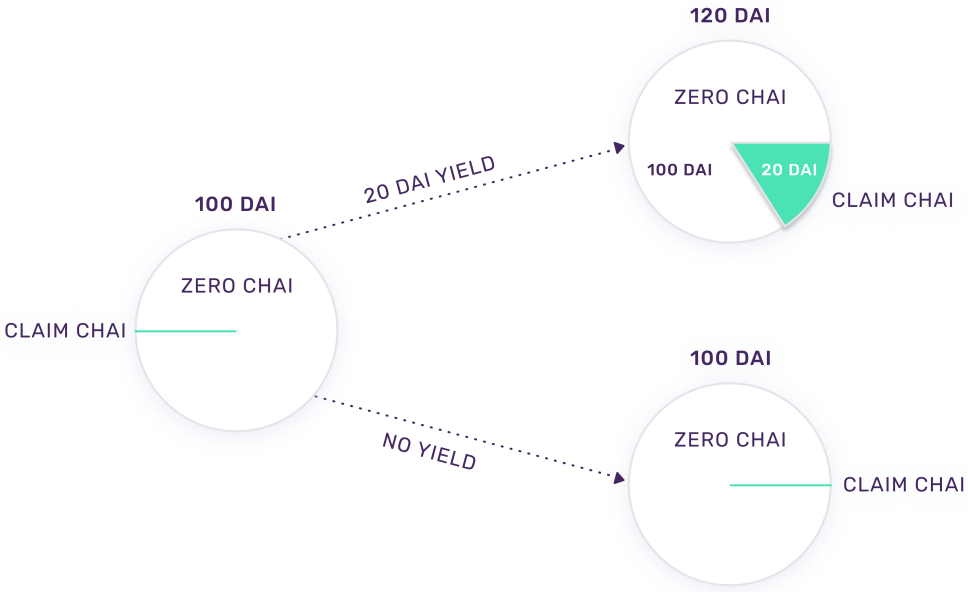

Figure 7: Zeros and Claims under Two Yield Conditions.

forecasters, and yield token holders seeking to lower their risk through *fixed rates*, with each achieving their financial objectives.

Since Deco always settles its future obligations with *zero* and *claim holders* in the yield token balance itself, it is up to the yield protocol to allow these users to immediately redeem their yield tokens for the underlying *base asset*. Deco does not handle underlying base assets as a feature, and does not involve itself in *base asset* redemption, which allows yield protocols to operate unencumbered by Deco. It is important to keep in mind that solvency issues at the yield protocol level can prevent users from withdrawing the yield token balance they receive from Deco (CHAI) into the *base asset* (DAI). While receiving a *fixed rate* is an important feature, we believe that Deco users should not face increased risk of loss due to solvency issues at the yield protocol level. In order to avoid this scenario, Deco allows users to quickly react and withdraw their deposit from the yield protocol at any time. Deposits are not unconditionally locked until a maturity date. To free the underlying yield tokens, all one needs to do is deposit both a *claim* and *zero* to *withdraw* the underlying yield token. Deco's design has several safety features which are described in later sections which should allay any fears that asset holders may normally have.

Figure 8 provides an outline of the *issuance* and *settlement* process of Deco assets which shall be explained in the next two sections. The market maker persona (light blue) is a neutral actor who momentarily steps in to facilitate the sale of *zero* and *claim* balances to two holders. Of course, either the *zero* or *claim* balance holder may perform the initial *issuance* instead of a market maker. The holder would simply hold on to one asset and sell the other to another party to complete the *issuance* step.

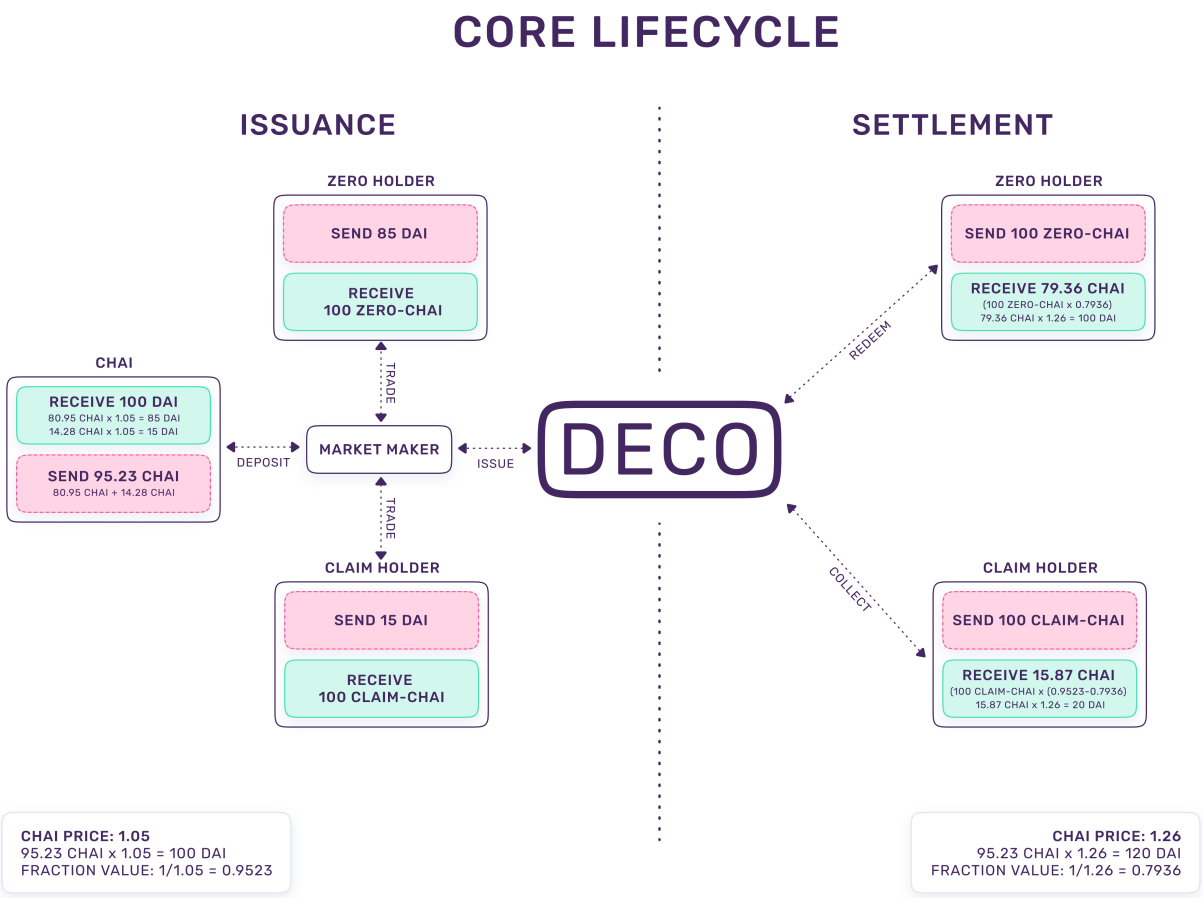

Figure 8: Core lifecycle of a yield token balance between issuance and settlement within Deco Protocol.

2.4.1 Issuance

Let's assume the *price* of CHAI in terms of the underlying *base asset* deposit DAI is 1.05 DAI/CHAI. This *price* includes the yield earned in DAI by DSR for its depositors since the original deployment of DSR.

When a user deposits 100 DAI today into the yield protocol CHAI, they will receive a yield token balance of 95.23 CHAI in return. This balance will be sufficient to immediately claim back the 100 DAI deposit if needed based on the current price of 1.05 DAI/CHAI.

Before *issuance* can occur at a timestamp, a *fraction* value needs to be set in the *snapshot* mapping. The current price of 1.05 DAI/CHAI is translated into a *fraction* value (1 CHAI/1.05 DAI) of 0.9523 CHAI/DAI.

When a user deposits 95.23 CHAI into Deco and issues at the timestamp where *fraction* value is 0.9523 CHAI/DAI, they will receive 100 ZERO-CHAI and 100 CLAIM-CHAI balances back in exchange for their CHAI deposit. The *zero* and *claim* balances are calculated by dividing the CHAI balance being deposit, and the underlying *fraction* value, and as we can see this translates into the *notional amount* of *base asset* that the yield protocol has received as deposit to generate yield.

At time $t1$, user deposits Y amount of CHAI into Deco protocol and receives Z amount of ZERO-CHAI and C amount of CLAIM-CHAI in return which can be calculated with the following equations,

$$Z = \rho_{t1} \times Y \tag{3}$$

$$C = \rho_{t1} \times Y \tag{4}$$

where ρ_{t1} denotes the price of CHAI at *issuance* timestamp $t1$.

Zero and *claim* balances use the standard 18 decimal balance format, and Deco can automatically detect the number of decimal places a yield token utilizes and adjust accordingly when dealing with it.

Zeros and *claims* are assets with different characteristics, and we have different settlement methods for each of them.

2.4.2 Settlement: Zero

The function used for settlement of *zero* assets is called *redeem*. *Zero* assets trade at a discount to *full face* value today for which they can be redeemed at *maturity*. The growth in value over the time duration gives them an imputed *fixed yield* which we can calculate using the *yield-to-maturity* formula shown below:

$$\text{Yield To Maturity} = \left(\frac{\text{Face Value}}{\text{Current Bond Price}}\right)^{1/\text{Years To Maturity}} - 1 \tag{5}$$

For example, 100 ZERO-CHAI issued today with a *maturity* date set one year from now could be sold in the market today at 80.95 CHAI (or 85 DAI) with the holder receiving an imputed yield of 17.6% over the course of the year when they are able to accrue sufficient yield tokens to claim 100 DAI at redemption.

Zero balance is tied to a *maturity* timestamp and can be settled at a *fraction* value right at or after maturity. Instead of settling automatically, using the latest *fraction* value snapshot available in the protocol, *zeros* are automatically settled at *maturity*, permitting subsequent accrual of yield on the *zero* balance. This method passes on all the savings accrued to the redeemed balance even when the *redeem* transaction is executed by the holder a long time after the actual *maturity*. This mechanism also avoids a rush of transactions and does not penalize those who may have forgotten to *redeem* their *zero* balance for a long period of time after *maturity*.

Z amount of *zero* balance with a *maturity* timestamp set to $t2$ is redeemed for Y amount of yield token balance as given by the equation,

$$Y = Z \times \frac{1}{\rho_{t2}} \tag{6}$$

where ρ_{t2} denotes the *price* of CHAI at *maturity* timestamp $t2$.

For example, let us continue with the issuance example (Figure 8) where ZERO-CHAI was issued at a CHAI price of 1.05 DAI/CHAI. A CHAI price of 1.26 DAI/CHAI at maturity of the ZERO-CHAI balance and would provide a fraction value at maturity of 0.7936 CHAI/DAI. Redemption of the 100 ZERO-CHAI to be redeemed for $100 \times 0.7936 = 79.36$ CHAI.

At redemption, sufficient yield token balance is sent back to the *zero* holder to redeem the same *notional amount* of *base asset* today from the yield protocol. 79.36 CHAI is sufficient to withdraw the full-face value of the ZERO-CHAI balance at the current price of 1.26 DAI/CHAI which is 100 DAI.

As you can see, since the *price* of the yield token has increased and a fraction of the yield token balance (79.36 CHAI) is sufficient to give back enough CHAI to allow the *zero* holder to *redeem* the original notional amount (100 DAI) deposited at issuance time.

The *fraction* value set at *maturity* is used to settle the ZERO-CHAI holder. With a *fraction* value of 0.7936 CHAI/DAI, 100 ZERO-CHAI will settle to $100 \times 0.7936 = 79.36$ CHAI. ZERO-CHAI holder has spent 85 DAI(80.95 CHAI $\times$ 1.05 DAI/CHAI) at issuance to receive 100 DAI(79.36 CHAI $\times$ 1.26 DAI/CHAI) at *maturity* which means the implied yield they were promised and received at maturity was 17.6%.

In an alternate scenario where the yield protocol did not earn any yield, the price would remain constant as well at 1.05 DAI/CHAI which means the entire 95.23 CHAI would be sent to the ZERO-CHAI holder to allow them to withdraw 100 DAI (95.23 CHAI @ 1.05 DAI/CHAI) at *maturity* and thereby still give them the promised fixed yield of 17.6% (85 DAI invested at *issuance* to receive 100 DAI at *maturity*).

2.4.3 Settlement: Claim

The function used for settlement of *claim* assets is called *collect*. In Figure 8, the user receives 100 CLAIM-CHAI at *issuance* with the yield token *price* at 1.05 DAI/CHAI, and the balance has reached *maturity* after a year with the yield token *price* at 1.26 DAI/CHAI.

Claim balances are tied to both an *issuance* and *maturity* timestamp and allow the user to *collect* the yield earned on the deposit in this period at settlement. *Claim* holders pay a

fixed price upfront to purchase their balance and take on the risk of a volatile rate in the hope of collecting a greater amount of the yield at settlement.

C amount of *claim* balance with *issuance* timestamp at $t1$ and *maturity* timestamp set to $t2$ can collect Y amount of yield token balance as given by the equation,

$$Y = C \times \left(\frac{1}{\rho_{t1}} - \frac{1}{\rho_{t2}} \right) \tag{7}$$

where ρ_{t1} denotes the *price* of CHAI at *issuance* timestamp $t1$ and ρ_{t2} denotes the *price* of CHAI at*maturity* timestamp $t2$.

For example, the *claim* holder has purchased a 100 CLAIM-CHAI balance for 14.28 CHAI (15 DAI) at *issuance*. At *maturity*, the yield token *price* has increased to 1.26 DAI/CHAI. The *fraction* values at *issuance* and *maturity* will be 0.9523 CHAI/DAI (1 CHAI/1.05 DAI) and 0.7936 CHAI/DAI (1 CHAI/1.26 DAI) respectively.

The user executes *collect* on this balance after *maturity* to receive (0.9523 - 0.7936) × 100 = 15.87 CHAI, which at the latest *price* of 1.26 DAI/CHAI can be used to collect 20 DAI from the CHAI protocol. As one can see, the CLAIM-CHAI holder has earned an additional 5 DAI over the original spend, but one can also see how misjudging future returns at the time of initial purchase may result in losses to the *claim* holder as well.

The settlement process concludes when the ZERO-CHAI holder redeems the principal amount of 100 DAI with the 79.36 CHAI and the CLAIM-CHAI holder collects the yield earned of 20 DAI with the 15.87 CHAI received from the Deco instance. By retiring both *zero* and *claim* asset balances, the Deco protocol can successfully distribute the entire CHAI amount originally deposited with it at issuance of 95.23 CHAI or 120 DAI.

Fraction value snapshots taken during *issuance* (0.9523 CHAI/DAI), and once again at the *maturity* date (0.7936 CHAI/DAI) make the settlement process possible. A fully trustless Deco instance relies on the protocol's ability to capture *fraction* value snapshots, on-chain and in a trustless manner. Due to the variations in *blocktimes*, as well as the times at which *snapshot* transactions are mined, a fully trustless implementation would be unable to *insert* a *fraction* value exactly at the *maturity* timestamp for the *zero* and *claim* assets. Since *redeem* can be executed only at a *fraction* snapshot at or after maturity for a *zero*, and *claim* can *collect* yield only at a *fraction* snapshot before or at maturity, this results in some CHAI being permanently locked within Deco. Deco instances with governance can be utilized to *insert* fraction values exactly at the *maturity* timestamp and avoid these losses.

2.5 Usage

Yield token holders seeking a *fixed rate* of return are target users for holding *zeros*. Those who are interested in higher yields, and are less risk averse, would choose *claims* which are more susceptible to rate volatility and consequently has the potential for higher returns.

Other candidates for accepting the risk of volatile rates would be yield token issuers like MakerDAO (DAI/DSR), Aave (aTokens), or Compound (cTokens) themselves. These protocols can hold on to *claims* and give their depositors rate protection by permitting them to exchange their yield tokens for *zeros*. In exchange, the issuer gains more certainty that the *base asset* is deposited in their protocols for a specific duration. The Deco protocol can play an important role in helping protocols thwart vampire attacks [8] from various

competitors by appropriately incentivizing liquidity providers to lock in their deposits for extended periods. Protocols will now better control the risk of rate volatility by being able to offer a *fixed rate* for a stated *term* up front; the exchange is then guaranteed locked liquidity for a stated period, and thus mitigates the risk created by demand for instant redemption of deposits which they currently face.

Redemption of the underlying *base asset* requires that both the variable interest component, and the *fixed rate* component be removed from circulation on a certain maturity date. Consequently, *zeros* must stay in circulation until removed. This removal from circulations locks the *base asset* deposit into the yield protocol for a defined duration. Locking in deposits for extended durations helps yield protocols avoid uncertainty about liquidity, particularly in cases where the protocol does not have withdrawal restrictions in place.

3 Safety

In this section, we discuss the various user levels and protocol level safety mechanisms that have been incorporated into Deco, so that assets can be issued safely, and with very long maturities, instead of being restricted, and short terms, 3-6 months, and still avoid risk.

3.1 Withdraw

The philosophy underpinning the design of Deco is that there must always be access to a withdrawal mechanism, and that the underlying yield token should never be locked by default into the maturity date. This flexibility is not only central to the philosophy of DeFi but is the core of Deco.

By making issuance reversible at any time through the *withdraw* function, Deco asset holders have greater comfort which allows them to confidently participate and promote liquidity in Deco assets (Figure 9). Holding an equal balance of both *zero* and *claim* assets of the same maturity date is economically equivalent to holding the yield token but cannot serve as an actual substitute. From a functional standpoint there must be a trustless mechanism which permits withdrawal at any point in the life cycle of the yield protocol.

Current protocols issuing yield tokens are unable to address the negative impact on users of having their yield tokens locked in for an extended period of time without the ability of redemption prior to maturity. Issuing *zeros* and *claims* does not create liquidity issues for underlying protocols but rather enhances them because Deco provides flexibility to the user. For example, DAI might be off-peg and it may be better for a ZERO-CHAI balance holder to *withdraw* their CHAI early for DAI and sell it on the open market and claim the premium offered for correcting the peg, rather than staying invested until maturity to collect the fixed yield.

Deco can make early withdrawal work by simply allowing users to take an equal amount of zero and claim balances of the same maturity date out of circulation and *withdraw* the entire yield token balance deposited without delay.

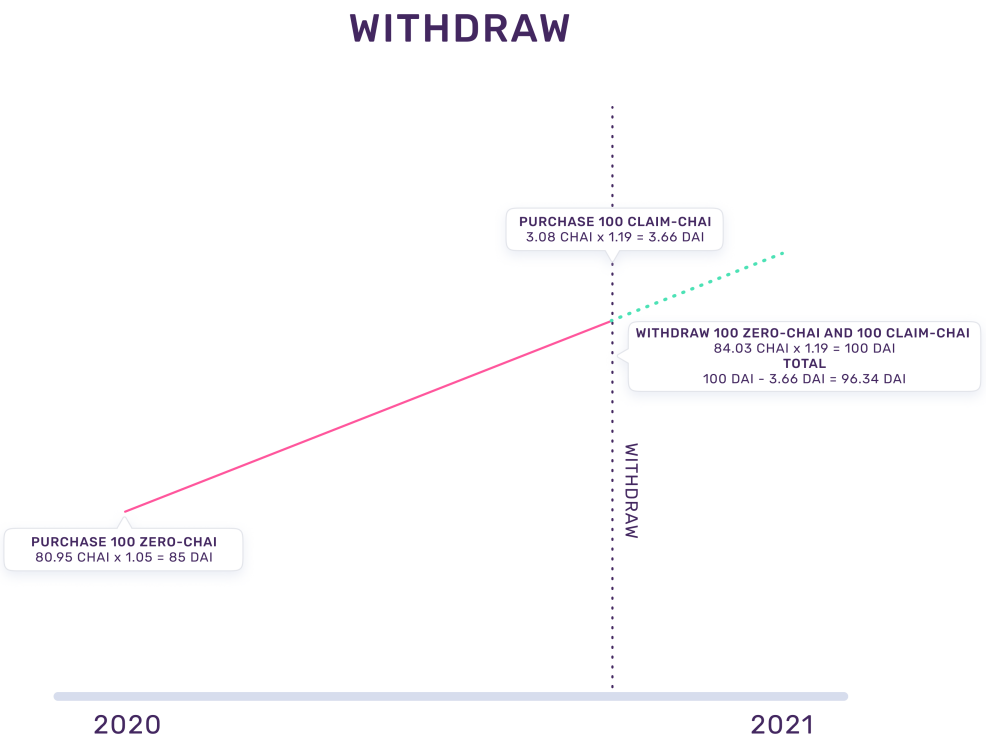

Figure 9: Withdrawal of a Zero balance for its underlying yield token balance before its maturity date.

3.2 Close

We have designed Deco instances to be capable of closing at any time and to permit all current and future obligations to be fairly settled independently for each individual holder.

There are several unexpected and troublesome scenarios that can occur in a yield protocol underlying a Deco instance. These problematic scenarios can be technical migrations, instability, or even protocol losses causing a complete shutdown, et cetera. While the Deco early withdrawal mechanism can help users to co-ordinate and *withdraw* on their own, ideally both *zero* or *claim* holders should also be able to independently withdraw the yield token value due, without the need to purchase the other asset they do not have (*zero* or *claim*) in their wallets under extraordinary circumstances.

In troublesome scenarios, Deco does not simply give *zero* asset holders the entire *notional amount* back, even when it seems like this is the right approach, since the future yield is essentially going to drop to zero for *claim* holders. If it did, this would incentivize *zero* holders to first buy their own balances at a discount to face value, and then try to immediately shut down Deco or even the underlying yield protocol in the hope of retrieving the full-face value without waiting until the *maturity* date. The *Close* mechanism has been carefully designed to remove the possibility of such *early settlement* attacks by *zero* asset holders. Deco instances will either encode a fair future valuation mechanism or lean on governance to pass a fair value back to both *zero* and *claim* holders. This promotes re-establishing positions in an equivalent yield protocol or a new deployment of the same yield protocol.

Some yield protocols, like the *Maker* protocol/*DSR*, contain an *emergency shutdown* mechanism which can result in the stop of all future yield accrual. Under such circumstances, unless there is a mechanism to address the risk of distortion of value, no matter how modest, will be priced in permanently. Deco is designed so that at closure there is a split of the value of locked yield deposit between both *zero* and *claim* holders fairly to prevent the market from pricing the possibility of *emergency shutdown* permanently into asset valuation.

Close can be triggered on a Deco instance by anyone when there is a clear on-chain signal demonstrating that the yield protocol is at risk. For example, the *Maker* protocol has a clear on-chain indicator called the *live* value and *close* can be triggered on a Deco instance deployed for CHAI when an aberrant change in this value is detected.

Deco governance can also be given permission to safely trigger *close* on a Deco instance when appropriate.

After *close*, a fair split of the locked yield token balance at each *maturity* date will be distributed between both *zero* and *claim* holders. The valuation of *zeros* and *claims* with a future *maturity* date can be performed either with an on-chain valuation calculation setup within Deco, or be input by governance. The split *ratio* is entered for each *maturity* timestamp which allows all *zero* and *claim* balances for that *maturity* date to *cash* out their balances independently for a portion of the yield token balance that is locked.

Instead of implementing a blanket freeze on all core functions after *close*, Deco uses a novel approach that permits assets to be settled based on their timestamps. This method allows assets that have their maturities before the *close* timestamp, and for which *fraction* data is already available, to simply settle using the regular settlement functions, even after *close* has been executed.

Cash functions are activated to allow the holders to settle their *zero* and *claim* balances

whose maturities fall after the *close* timestamp based on the valuation *ratios* that are stored.

The novel *close* mechanism allows Deco to translate a variety of uncertain conditions for various yield protocols, each with their own constraints, back into a single timestamp and a *last* fraction value to denote the end. This safely settles all existing balances for any *maturity* timestamp past or future.

In scenarios where Deco instances are deployed for a single round of issuance and settlement occurs once at a one-off maturity date, a single *close* transaction can be used to settle all the asset balances. With Deco, there is no need to capture multiple *fraction* values over time, allowing *close* to play the role of a primary settlement mechanism.

4 Liquidity

We have also developed a variety of features that enhance the liquidity profile of Deco assets and allow users greater flexibility without compromise.

4.1 Asset Design

One Deco instance deployment for a yield protocol is sufficient to manage balance issuances for any maturity date. There is no need to deploy new Deco instances to specifically handle maturity dates as time passes, for example at 3 months after issuances, 6 months after issuance, et cetera. This approach significantly lowers operational costs. Timestamps may be agreed upon as standard and may be communicated to everyone by the Deco instance operator. This method allows market participants to issue their balances and align on standard dates.

4.1.1 Class

All *zero* and *claim* balances for an *issuance* and *maturity* date are held in a single smart contract. We have created a mechanism called *class* to simultaneously allow asset balances that have common characteristics to be fungible with each other and keep those that differ separate.

Zero balance amounts with the same *maturity* date are fungible with each other since they can be redeemed after the date for the same amount of yield token. *Claim* balances that share the same *issuance* date and *maturity* date are also perfectly fungible since they collect the same amount of yield token based on yield earned between stated dates. Deco operations like *collect*, *slice*, *activate*, and *merge*, when applied to a *zero* or *claim* balance, change the character of the balance and also update its *class* immediately to prevent fungibility issues from emerging.

Class value is derived by taking a solidity *keccak256* hash of the *maturity* date for *zero* balances, and both the *issuance* and *maturity* dates for *claim* balances.

4.1.2 Balance Adapters

Individual Deco asset holders may choose between *ERC20* [13] or *ERC721* [19] token standards while issuing their *zero* or *claim* balances through *balance adapters*. Users can simply

stick to holding the asset balances internally within the main Deco contract to also save on gas costs if they do not have a particular need to tokenize their balances.

Deco instance operators will pay for gas costs to deploy *ERC20* token contracts for popular maturity dates in the market, e.g., end of month, quarter, and year. This helps channel usage and liquidity into more concentrated buckets. Users are also allowed to deploy their own *ERC20* token for a nonstandard maturity date if they need one without having to obtain permission from the operator. Users also have the choice to issue their asset balances with "non-standard" maturity dates as an *NFT* from the deployed *ERC721* contract at a much lower gas cost. Both token standards have robust exchange and wallet infrastructure and offer many other benefits to meet user needs.

4.1.3 Approvals

Approvals allow a balance holder to delegate the management of their asset balance to other addresses. This system is most commonly used to allow other smart contract protocols to operate on a user's balance to deposit or withdraw it or perform tasks like token trades on their behalf. Users holding Deco asset balances in *ERC20* or *ERC721* token form can use their respective *approval* systems to authorize other smart contracts or addresses to operate on their balance.

We have incorporated a *binary approval* system for internal asset balances held within the Deco core contract to give users additional flexibility. This allows users to *approve* or *disapprove* another address from operating on all their asset balances for the entire balance amount without an ability to set limits like with *ERC20* tokens. Approved addresses are allowed to take all actions like *issue*, *redeem*, *join*, or *exit* balances on a user's behalf. This internal *approval* system can be especially useful when a user wants to set an *approval* to their own *proxy contract* wallet to broadly authorize it to take various actions on all their Deco asset balances.

4.2 Collect

In the previous section, we explained how *collect* can be used by *claim* holders to settle and withdraw their yield tokens earned at maturity.

When a user deals with larger amounts of *claim* balances, the yield earned before maturity can be significant. It may be inefficient for a user to retrieve yield earned only at maturity. We have designed the *collect* mechanism so that it can be executed by a *claim* balance holder any number of times before maturity to allow periodic retrieval of accrued yield any number of times.

Let's use a 100 CLAIM-CHAI balance with issuance on January 1st 2020 and maturity set to December 31st 2020 as an example. At a point of time in between, let's say September 1st, the valuation of this *claim* balance would comprise of the known yield already earned, and the expectation of the amount of yield it will be earning in the remaining time period from September 1st until its maturity on December 31st. Let's assume the price of the yield token CHAI at issuance on January 1st was 1.05 DAI/CHAI and 1.19 DAI/CHAI on September 1st 2020 (Figure 10).

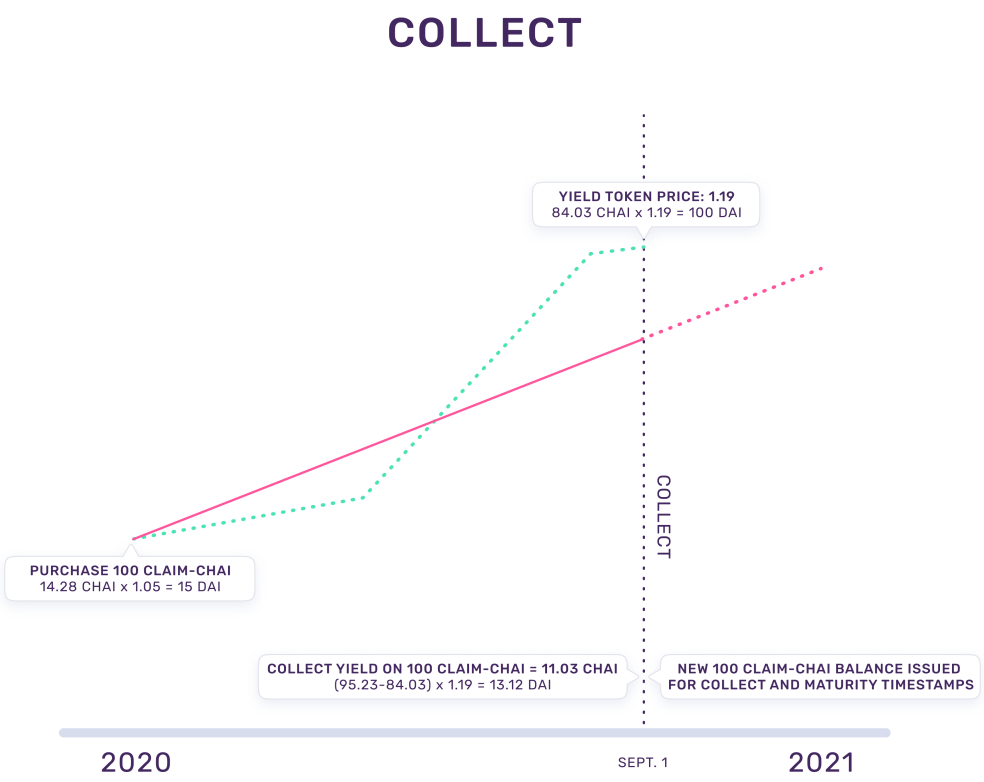

Figure 10: Collecting yield earned by a claim balance early before its final maturity date.

When *collect* is executed by the user, the 100 CLAIM-CHAI balance is burned and removed from circulation and Deco sends the user 11.03 CHAI which is the entire yield earned until then. Deco also issues a new 100 CLAIM-CHAI balance but with a new issuance date set to September 1st and the maturity date set again to December 31st to give the user back a balance they can use to collect the remaining future yield.

We can see that every time *collect* is executed, the *issuance* timestamp on the *class* of the same *claim* balance amount is updated to the *collect* timestamp and all yield earned by the balance up to that point is sent. This step automatically updates the *class* of the *claim* balance to continue to preserve fungibility and separate this CLAIM-CHAI balance from those that have not collected their yield so far, and wish to do so later. This automatically prevents new *subclasses* from forming within *claim* balances that have the same *issuance* and *maturity* timestamps but have collected their yield at different timestamps.

This feature can also be beneficial to *claim* holders who work with market-standard *issuance* and *maturity* dates as well. They can collectively retrieve earned yield on a weekly basis and move the entire market periodically to a new standard *issuance* date for the same *maturity* date. This ensures that most of the value of *claim* balances in circulation is due to the potential value of future yield and not based on past yield already earned.

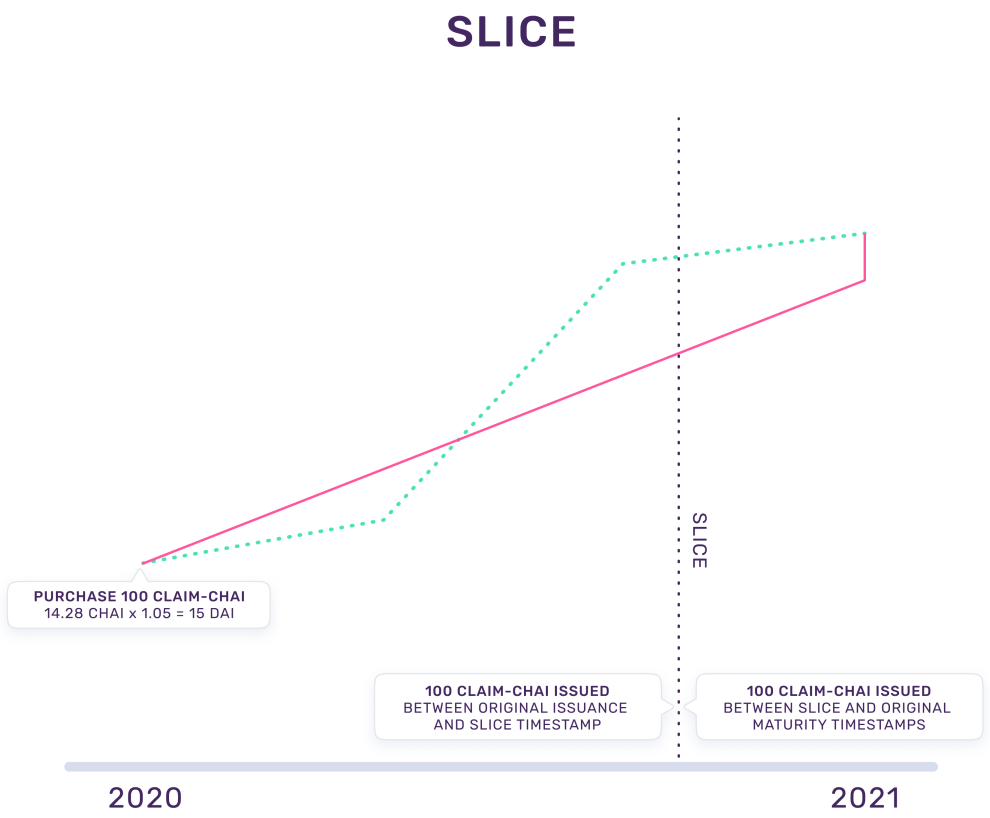

Figure 11: Slice one claim balance into two token balances which collect yield for their respective time periods.

4.3 Rewind

Deco implements a function called *rewind* to ensure the *issuance* of a timestamp can be moved back to another past timestamp when needed, especially to facilitate a non-standard timestamp *claim* issuance, or to convert itself to a standard timestamp that is being used for most of the trading activity. *Rewind* enables fungibility across timestamp *classes* of the same asset.

4.4 Slice

Claim holders can also split their term into two parts by using the *slice* function. *Slice* partitions the total time between *issuance* and *maturity* into two parts at a *slice* timestamp in the middle, thereby issuing two new *claim* balances that represent new time periods. This feature can be helpful when the original *claim* balance is for a long maturity period, e.g., a year, and the holder wants to extract a *claim* balance for only the first quarter to be able to trade in markets while holding the remaining time for future activation. We have incorporated a *merge* feature which permits sliced *claim* balances into a single continuous time period (Figure 11).

Collect, *rewind*, *slice*, *activate*, and *merge* collectively stop any issuances that are consid-

ered non-standard by the market from being hit with an illiquidity premium by allowing the holders to transform their balances to match a standard issuance either in the past or the future which enjoys greater liquidity on the market.

4.5 Exchanges

Primary trading pairs for *zeros* and *claims* will be with their respective yield tokens on various exchanges.

Auctions have the potential to be a dominant mechanism for *zeros* because most traders are searching to get the best rate possible versus holding the yield token itself. Users who place a trade between zeros and the yield token or vice versa are not affected by price volatility when they place orders in auctions which typically run longer durations. Deco asset balances that are held as *NFTs* can trade on auction infrastructure like OpenSea (Wyvern protocol) [9], and those held as *ERC20* tokens can use the new Gnosis Auction Protocol V2 [2].

Order book exchanges like OasisDEX, 0x, and their regulated counterparts like Oasis Pro Markets can also serve as great venues for liquidity.

Automated Market Makers (AMMs) can also be used by traders to supply liquidity and YieldSpace [15] is a great fit for liquidity providers to set up pools with a *zero* at a certain *maturity* date and its yield token counterpart.

5 Governance

Operating a Deco instance successfully for a yield token requires the following external data to be input into it on a regular basis:

- *Fraction value* snapshots derived from the yield token price over time.
- Monitoring the safety and solvency of a yield protocol to determine whether the attached Deco instance should be closed proactively to protect users.
- Adding *zero* and *claim* valuation data to process withdrawals of some balances after *close*.

Governance mechanisms are set up to input this data whenever Deco instances are not able to capture it themselves directly from the yield protocol. Even in scenarios where the data can be captured in a trustless manner, it might only be available on-chain for a very short duration which could easily result in a failure to capture it within Deco at the right time. Governance mechanisms serve as a crucial backup mechanism who can be trusted to input this data ex post facto in such scenarios and prevent inevitable losses for users.

5.1 Capturing Snapshots

Yield protocols only make their latest price value available on-chain but trying to capture and store it within Deco by submitting a blockchain transaction can make the process unpredictable due to network congestion and variability as to when the transactions are mined

and confirmed by the blockchain. Governance mechanisms face no such issues since they can *insert* fraction values at precise maturity timestamps, unlike trustless mechanisms that are only able to capture them at the block timestamps of mined transactions.

Required *fraction* values can also be lost forever in extreme situations as when a *snapshot* transaction is not mined before the maturity timestamp. Governance mechanisms are able to backfill fraction values at past maturity timestamps to help *zero* and *claim* holders settle their positions and prevent losses. These errors can easily occur in fully trustless and ungoverned deployments.

5.2 Close

Governance can consider a broad range of issues, both on-chain and off-chain, and *close* the Deco deployment to allow all asset holders to immediately settle their positions for safety reasons. Yield protocols can face solvency issues at any time which can prompt yield token holders to withdraw their deposits immediately. Only a trusted governance mechanism can quickly react and *close* the Deco instance and allow *zero* and *claim* holders to immediately withdraw their deposits like any other yield token holder and prevent losses.

After *close*, valuations of the future time portions of assets are required to settle them correctly. When an on-chain valuation calculation cannot be implemented, or if it is expensive to do so, governance can again be used as a trusted source to perform these calculations off-chain and input this data as well.

It is important to keep in mind that all Deco deployments are independent which gives each one the ability to select where it wants to be on the trustless spectrum and select the trusted governance body. It is possible for multiple Deco instances to be deployed for the same yield protocol and allow each one to operate at different parts of the trust spectrum to serve differing market participants' needs.

6 Future Work

6.1 Leverage

Protocols like Maker, Aave, and Compound can benefit from facilitating leverage for Deco users by on-boarding various *zero* assets as collateral and enabling leverage. The primary source of volatility for stablecoins deposited into yield protocols is their rate of return. When rate volatility is low, *zeros* can be set up as collateral with high leverage factors desired by rate speculators making bets with additional leverage. Additional research in this area will make it both easy and safe to on-board various *zero* assets as collateral into multiple lending protocols.

6.2 Negative Rates

When the yield token has a negative rate, *zeros* may be purchased at a premium over *notional amount* and its holder will receive the lower *notional amount* upon maturity implying a negative yield over the time from the original purchase price. Ex: At a negative rate of -5%,

100 ZeroUSD which can be used to redeem 100 USD at a future maturity will be sold at a higher price of 105.25 USD today.

The role of *claim* holders under a negative rate regime will shift from collecting yield to topping up Deco with lost value as the negative rate gets applied continuously to bring the *notional amount* back up allowing the *zero* holder to redeem the value in full at maturity in the future. *Claim* holders can plug the gap either with a credible mechanism that can draw the lost value from the source protocol continuously or by leaving an excess deposit that is sufficient to settle the future obligations to *zero* holders under known maximum negative rates.

With slight modifications, we can convert the same design that was outlined above to work with yield tokens that operate with a volatile negative rate but want to offer *zeros* with a fixed negative yield.

7 Conclusion

Deco is a safe and flexible fixed-rate protocol to issue *zeros* and *claims* for any yield token present on Ethereum and other blockchains. The protocol is flexible and offers certainty as to yield based upon individual risk tolerance.

Our novel asset balance management approach with the use of *classes* to separate any number of balances within Deco totally removes the need to deploy new smart contracts continuously and vastly reduces on-going operator involvement.

Deco is an expansive and meticulously designed protocol, going beyond the basics of *issuance* and *settlement*, and introduces additional flexibility into the management of assets. Deco allows the reversal of *issuance*, and *maturity* timestamps which can be seamlessly modified and consequently providing superior liquidity.

Deco instances can be deployed to operate at various levels of the trust spectrum, based on user choice. Also, it can rely on governance for crucial settlement information to keep the protocol both efficient and safe. The novel shutdown mechanism keeps all *zero* and *claim* holders safe, without making them rely on others to trade, as one cannot assume market efficiency.

Acknowledgement

The author would like to thank *Andrew Dawson*, *Rohit Kapoor*, and *Suman Bhunia* for their immense help in bringing this whitepaper to shape.

Glossary

approvals Approvals allow users to delegate full management of their Deco balances to other Ethereum addresses. 18

balance adapters Balance adapter smart contracts allow Deco balances to conform to either ERC20 or ERC721 NFT token standards based on user choice. 17

base asset Token deposited into a yield protocol. 1

blocktime Timestamp at which a block is mined in a blockchain. 13

cash Cash functions are activated after a Deco instance is closed to allow users to immediately settle their Deco balances at future maturity dates for yield token balances. 16

CLAIM-CHAI Claim balance in a Deco instance set up for CHAI. CLAIM-CHAI balance is created when a user executes the issue function on a Deco instance deployed for the CHAI protocol. 6

class Deco balances are separated with a class value derived from their issuance and maturity dates to only allow fungibility among balances that share the same characteristics. 17

close Close function stops new Zero and Claim issuance and allows existing Zero and Claim balances for all maturity dates, past or future, to be immediately settled for their share of the yield token deposits held by Deco. 16

collect Collect allows Claim balance holders to withdraw the yield earned on their balance. 12

DAI Stablecoin which is pegged to USD issued by the Maker protocol. 1

Dai Savings Rate Yield protocol managed by the Maker protocol to give all DAI holders additional yield when they lockup. 6

emergency shutdown Maker protocol feature which stops all new issuance of Dai and allows holders to redeem it for various amounts of collateral held within. 16

fixed rate An exact amount of yield earned by holders for a pre-defined time period on their deposit. 1

fraction Fraction value is the amount of yield tokens a unit of the base asset can purchase. For example, the amount of CHAI one DAI can purchase. It is typically 1 or lower since the yield token almost always is worth more in terms of the base asset due to the additional yield it accrues over time in the base asset itself. 5

governance Body of users who use coordination methods like voting, delegation, et cetera to take actions they are authorized to on a Deco instance. 21

insert Insert allows governance to input fraction values for various timestamps to be used for settlement purposes of balance holders. 5

instance A Deco instance is a deployment of the Deco protocol smart contracts to handle the needs of a specific yield token. 1

issuance Users generate Zero and Claim balances by making a yield token deposit into a Deco instance. 4

leverage Ability to lock a token balance as collateral and borrow another asset as debt to be repaid back later. 22

liquidity provider Users who deposit base assets into protocols that in turn provide various services like exchange, staking, lending, et cetera to other end-users. 14

Maker protocol Decentralized lending protocol which allows users to borrow against a variety of collateral and generate the USD pegged stablecoin DAI. 6

market maker User who specializes in supplying trading liquidity for a variety of assets on exchanges. 9

maturity Timestamp after which assets can be exchanged back for their promised shares of the locked yield token balance from Deco. 1

price Value of the yield token in base asset terms. For example, the amount of DAI a CHAI balance can withdraw from the CHAI protocol. 2

pure-yield Asset that promises its holder only the yield earned at settlement. 1

redeem Zero balances are redeemed for their share of the yield token deposit in Deco after their maturity date. 6

rewind Issuance of any Claim balance can be set to a previous timestamp provided a fraction value snapshot is available at that timestamp. This is typically used to adjust a Claim balance from a non-standard issuance date to a standard date that exists in the past to take advantage of available ERC20 token adapter and exchange liquidity. 20

settlement Process of giving back the entire yield token balance deposited at issuance back to Zero and Claim holders based on the terms attached. Zero and Claim balance holders use functions like redeem, collect, withdraw, or cash based on the circumstance for settlement. 4

slice Splitting a Claim balance at a timestamp between its issuance and maturity timestamps. 20

snapshot Snapshot function captures the fraction value of a yield token at the current timestamp and stores it for future reference during settlement. 4

solvency Yield protocols are considered solvent when they can honor redemption of all base asset deposits and accrued yields. 8

term Time between issuance and maturity timestamps. 1

trustless Trustless functions in Deco instances can capture data directly from the yield protocol's smart contracts and allow anyone to execute them. 1

withdraw Function which allows early redemption of a Zero balance before its maturity timestamp. 14

yield Amount of additional base asset balance accrued after a certain duration which the original deposit can claim from the yield protocol. 1

yield protocol Protocol which takes in a base asset deposit from liquidity providers and passes them a yield generated from the fees charged to users for providing a service. 1

yield token Yield protocols give liquidity providers a yield token in exchange for their base asset deposits. Yield token balances can typically be exchanged back again for the base asset deposit as well as any additional yield earned by it from the yield protocol. 1

ZERO-CHAI Zero balance in a Deco instance set up for CHAI. ZERO-CHAI balance is created when a user executes the issue function on a Deco instance deployed for the CHAI protocol. 6

zero-yield An asset that trades at a discount to its face value for which it settles at on the maturity date and without making any explicit yield payments to the holder during the term. 2

References

[1] Ethereum 2.0 staking. https://ethereum.org/en/eth2/.

[2] Gnosis protocol v2. https://github.com/gnosis/gp-v2-contracts/blob/main/docs/architecture.md.

[3] Maker protocol: Dai savings rate. https://community-development.makerdao.com/en/learn/Dai/dsr/.

[4] The maker protocol: Makerdao's multi-collateral dai (mcd) system. https://makerdao.com/en/whitepaper/.

[5] Maker protocol rates module. https://archive.is/AcEpY.

[6] Notional protocol. https://docs.notional.finance/developers/whitepaper/.

[7] Smart yield bonds. https://archive.is/QhMsm.

[8] Vampire attack. https://finematics.com/vampire-attack-sushiswap-explained/.

[9] Wyvern protocol. https://wyvernprotocol.com/docs#matching-orders.

[10] Yearn finance: yvaults. https://docs.yearn.finance/yearn-finance/yvaults/overview.

[11] Zero-coupon bond. https://www.investopedia.com/terms/z/zero-couponbond.asp.

[12] Aave protocol, January 2020.

[13] Fabian Vogelsteller, Vitalik Buterin. Erc-20 token standard. https://eips.ethereum.org/EIPS/eip-20.

[14] Robert Leshner, Geoffrey Hayes. Compound: The money market protocol, February 2019.

[15] Allan Niemerg, Dan Robinson, Lev Livnev. Yieldspace: An automated liquidity provider for fixed yield tokens, August 2020.

[16] Martin Lundfall, Lucas Vogelsang, Lev Livnev. Chai – a simple erc20 wrapper over the dai savings rate. https://chai.money/about.html.

[17] Dan Robinson, Allan Niemerg. The yield protocol: On-chain lending with interest rate discovery, April 2020.

[18] Hayden Adams, Noah Zinsmeister, Dan Robinson. Uniswap v2 core, March 2020.

[19] William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs. Erc-721 non-fungible token standard. https://eips.ethereum.org/EIPS/eip-721.