Declarative Liquidity Manager (DLM)

Alex Euler

September 22, 2023

Abstract

DLM is a modern DeFi protocol that offers users a flexible platform for decentralized asset management. The system autonomously handles DeFi liquidity for users and supports both fungible and non-fungible tokens.

Contents

1 Overview

1.1 Evolution of Liquidity Management in DeFi

The origins of DeFi can be traced back to the creation of Bitcoin in 2009, which introduced a decentralized, peer-to-peer network for transferring value without intermediaries. However, it was not until the emergence of Ethereum in 2015 that the foundations for DeFi were truly laid. With Ethereum's support for smart contracts, developers were able to build decentralized applications that could execute complex financial transactions on the blockchain.

The first wave of DeFi asset management protocols appeared in 2016, with the creation of decentralized exchanges such as EtherDelta and 0x. These DEXs allowed users to trade cryptocurrencies without the need for a centralized exchange, enabling a more secure and transparent trading environment.

In 2017, the DeFi landscape expanded with the introduction of decentralized lending platforms, such as MakerDAO and Compound. These platforms allowed users to lend and borrow cryptocurrency in a decentralized and trustless manner, without the need for a traditional financial intermediary.

In 2018, the first decentralized asset management protocol, Melonport, was launched. Melonport provided a platform for users to create and manage their own investment funds, giving investors greater control over their assets and removing the need for a centralized fund manager.

The DeFi asset management space continued to evolve in 2019, with the launch of new protocols such as Set Protocol and PieDAO. Set Protocol allowed users to create and manage baskets of assets, known as "sets", while PieDAO in-

troduced the concept of community-led asset management, where the investment decisions of the fund were made through a decentralized governance process.

In 2020, the DeFi space experienced a massive surge in activity, with the TVL in DeFi protocols increasing from less than $1 billion to over $15 billion in just a few months. This growth was partly fueled by the emergence of new DeFi asset management protocols such as Yearn Finance and Enzyme, which introduced novel concepts such as yield farming and automated asset management.

In 2022, during a larger crypto-economic boom, the new innovative protocols that expanded the horizons of liquidity management and financial decentralization. Key players, such as Mellow, Gamma, and Brahma, emerged during this period, each bringing their unique offerings and architectures.

1.2 Account abstraction

Ethereum has two types of accounts: Externally Owned Accounts (EOAs) and Contract Accounts. EOAs are controlled by private keys and are used primarily for sending and receiving Ether. In contrast, Contract Accounts are governed by their code and are responsible for the logic and operations of smart contracts.

Account abstraction is, at its essence, an attempt to merge the distinctions between these two types of accounts. By doing so, it seeks to offer users better UX and provide developers with more flexibility in coding contracts. This initiative doesn't just make Ethereum simpler; it also paves the way for a host of new features. For instance, with account abstraction, smart contracts can directly pay for gas fees and eliminating the need for users to hold Ether in their wallets for transaction costs.

1.3 Intent-based architectures

Traditionally, users interact with Ethereum by creating and signing transactions that the Ethereum Virtual Machine (EVM) uses to execute a state transition. However, crafting these transactions can be intricate, forcing users to navigate a complex web of smart contracts. To simplify this, "intents" were introduced. An intent is essentially a signed set of constraints allowing users to delegate the creation of transactions to third parties, without giving them full control. While a traditional transaction requires a specific computational path, an intent provides flexibility within specified boundaries, permitting various pathways as long as certain constraints are met.

Clever use of intents can elegantly simplify several processes:

- **Limit Orders**: Specifying conditions for buy/sell transactions
- **CowSwap style Auctions**: Using third-parties for optimal order matching
- **Gas Sponsorship**: Allowing for gas payments in alternative tokens
- **Delegation**: Limiting interactions to pre-authorized methods
- **Transaction Batching**: Grouping multiple intents for efficiency

Intents can also play a pivotal role in asset management. By leveraging the intent-based paradigm, asset managers can streamline their operations by specifying investment objectives without dictating each transaction's details. For instance, instead of specifying purchases of specific stocks or assets at particular prices, an asset manager can express an intent like, "Optimize my portfolio for maximum growth over the next quarter, keeping the risk moderate," and let the underlying system or third-party providers decide the best trades or adjustments to achieve that intent.

2 Declarative Liquidity Manager

2.1 Introduction

In this paper, we introduce a novel design for a Declarative Liquidity Manager (DLM). The distinguishing features of DLM are outlined as follows:

1. **Intuitive Asset Allocation:** DLM offers an easily definable asset allocation function
2. **Adaptive Flexibility:** The asset allocation function is highly flexible, capable of incorporating any current or historical onchain data
3. **Efficiency and Deployment:** DLM has been optimized for minimal gas consumption, and its smart wallet can be deployed swiftly and inexpensively
4. **Cost-effective Rebalancing:** One of DLM's core advantages is its capability to rebalance assets at an optimal cost

2.2 High level architecture

At the heart of the proposed system is the smart wallet. This wallet is a lightweight smart contract designed to store fungible, non-fungible, and native

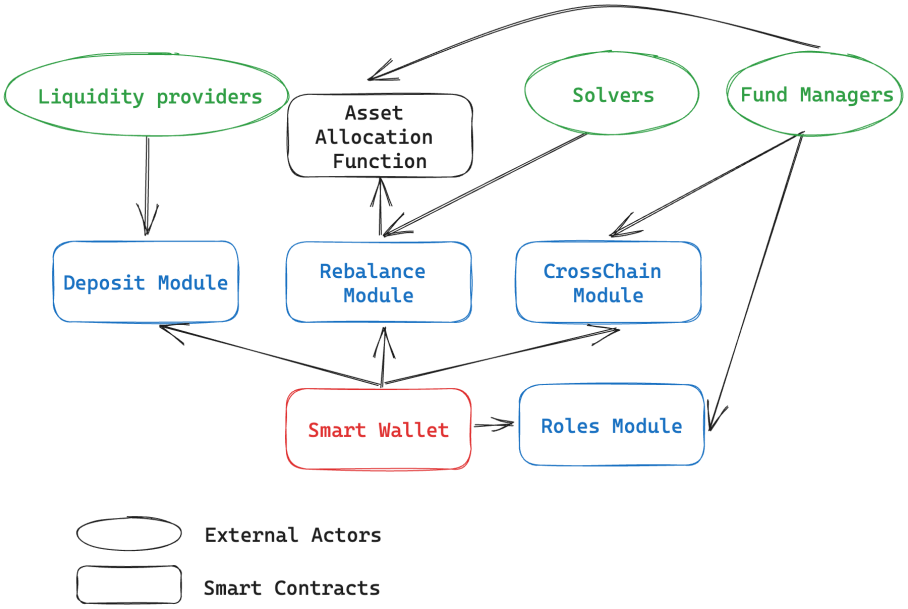

Figure 1: High level architecture

tokens. (see Figure 1). It can also integrate various smart contract modules, enhancing its utility beyond mere token storage. The concept assumes that each asset management strategy will possess its dedicated smart wallet. This wallet will safeguard the tokens of liquidity providers, which are then orchestrated by specific modules.

The proposed architecture is based on a modular design that is inherently extensible. Initially, we anticipate the integration of the following modules:

- **Roles Module**: Facilitates Role-Based Access Control within the smart wallet
- **TVL Module**: Quantifies the smart wallet's TVL in USD or native tokens such as ETH, MATIC, etc.
- **Deposit Module**: Tasked with overseeing the deposits and withdrawals made by Liquidity Providers
- **Rebalance Module**: Provides solvers with the capability to rebalance assets within the smart wallet in accordance with a predetermined asset distribution

2.3 Wiring framework

Vaults and smart wallets systems are being developed for many years, more active during the recent DeFi summer. It didn't take long for vault developers to recognize the importance of a modular approach to smart wallet functionality. However, the mechanics of attaching the modules to smart wallets is not yet standardized.

First attempt at module decomposition was made in ERC-2535 (diamond pattern). Later developments include ERC-4337 (Account Abstraction), ERC-6900 (Modular Smart Contract Accounts and Plugins) and Safe protocol.

We suggest considering one of two proposed solutions:

- **Safe{Core} AA SDK**: A modular smart wallet protocol that uses Account Abstraction to build a wide range of wallets and other solutions through a shared plugin interface
- **Crystal Pattern**: An alternative to ERC-2535 diamond pattern focused on cheaper calls and smart wallet deployments

2.3.1 Safe{Core} AA SDK

The *Safe{Core} Protocol* is an open-source framework for modular smart accounts that are composable, portable, and secure. The protocol is inspired by the learnings from the Safe Smart Account but is fundamentally vendor-agnostic and meant as an ecosystem initiative, solving the following challenges:

- **Fragmentation:** As various smart accounts are developed by vendors, tooling and applications have to prioritize which ones to be compatible with. This may result in major fragmentation of the smart account ecosystem and lack of component reuse. This degradation of both developer and user experience reduces the overall growth and utility of smart accounts
- **Vendor Lock-In:** Externally owned accounts are a widely accepted standard, allowing portable accounts across wallets and applications. The fragmentation of smart accounts, however, may create vendor lock-in due to wallets creating proprietary account implementations that reduce portability. End users may face restrictions when trying to move their assets or interact with different services
- **Security Risks:** Smart accounts introduce additional smart contract risk, similar to other smart contract-based systems like bridges or DeFi protocols. Without sufficient security, users will not feel comfortable leveraging the full potential of smart accounts

To address these challenges the *Safe{Core} Protocol* aims to achieve maximum reuse and interoperability of components, retain portability, and establish stronger security properties for smart accounts. Developers adopting the *Safe{Core} Protocol* benefit from increased security, interoperability, and composability with the wider smart account ecosystem, as well as means to charge fees in the future.

2.3.2 Crystal Pattern

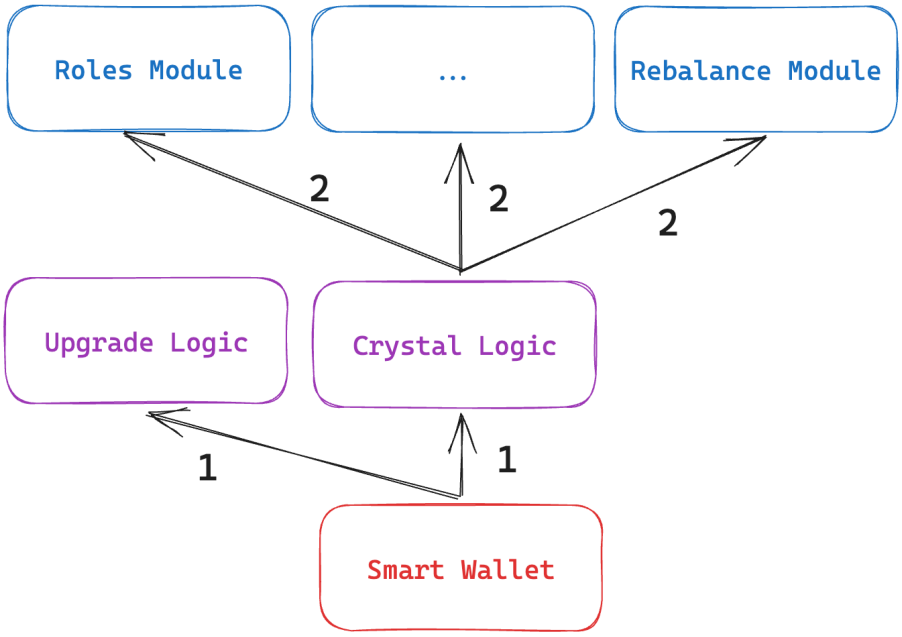

Figure 2: Crystal pattern

2.3.2.1 Smart Wallet

In this article, we introduce The Crystal Pattern. It is engineered for minimal gas consumption during deployment and calls, all while maintaining wallet upgradability. To achieve this goal, we employ a combination of several established proxy standards:

- ERC-7229: Minimal upgradable proxy contract
- ERC-1822: Universal Upgradeable Proxy Standard (UUPS)

Table 1: Crystal Smart Wallet Opcodes

Opcode	**Opcode + Arguments**	**Description**	**Stack View**
Constructor code			
0x60	0x60 0x00	PUSH1 0x00	0
0x73	0x73 impl	PUSH20 impl	impl 0
0x81	0x81	DUP2	0 impl 0
0x19	0x19	NOT	0xffff..ffff impl 0
0x55	0x55	SSTORE	0
0x60	0x60 0x50	PUSH1 80	80 0
0x80	0x80	DUP1	80 80 0
0x60	0x60 0x23	PUSH1 35	35 80 80 0
0x83	0x83	DUP4	0 42 80 80 0
0x39	0x39	CODECOPY	80 0
0x81	0x81	DUP2	0 80 0
0xf3	0xf3	RETURN	0
Deployed code			
0x60	0x60 0x00	PUSH1 0	0
0x36	0x36	CALLDATASIZE	csize 0
0x81	0x81	DUP2	0 csize 0
0x80	0x80	DUP1	0 0 csize 0
0x37	0x37	CALLDATACOPY	0
0x80	0x80	DUP1	0 0
0x80	0x80	DUP1	0 0 0
0x36	0x36	CALLDATASIZE	csize 0 0 0
0x81	0x81	DUP2	0 csize 0 0 0
0x33	0x33	CALLER	caller 0 csize 0 0 0
0x73	0x73 admin	PUSH20 admin	admin caller 0 csize 0 0 0
0x14	0x14	EQ	false 0 csize 0 0 0
0x60	0x60 0x2a	PUSH1 42	42 false 0 csize 0 0 0
0x57	0x57	JUMPI	0 csize 0 0 0
0x80	0x80	DUP1	0 0 csize 0 0 0
0x19	0x19	NOT	0xffff..ffff 0 csize 0 0 0
0x54	0x54	SLOAD	impl 0 csize 0 0 0
0x60	0x60 0x40	PUSH1 64	64 impl 0 csize 0 0 0
0x56	0x56	JUMP	impl 0 csize 0 0 0
0x5b	0x5b	JUMPDEST	impl 0 csize 0 0 0
0x5a	0x5a	GAS	gas impl 0 csize 0 0 0
0xf4	0xf4	DELEGATECALL	success 0
0x3d	0x3d	RETURNDATASIZE	rsize success 0
0x82	0x82	DUP3	0 rsize success 0
0x80	0x80	DUP1	0 0 rsize success 0
0x3e	0x3e	RETURNDATACOPY	success 0
0x3d	0x3d	RETURNDATASIZE	rsize success 0
0x82	0x82	DUP3	0 rsize success 0
0x82	0x82	DUP3	success 0 rsize success 0
0x60	0x60 0x4e	PUSH1 78	78 success 0 rsize success 0
0x57	0x57	JUMPI	0 rsize success 0
0x5b	0x5b	JUMPDEST	0 rsize success 0
0xf3	0xf3	RETURN	success 0

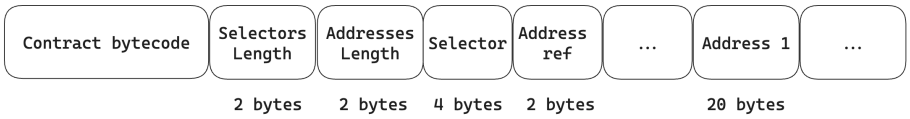

Figure 3: Crystal Logic layout

- TransparentUpgradeableProxy

First, we establish a *Crystal Logic Contract* that contains all potential modules. The Smart Wallet is designed as a Minimal Upgradeable Proxy. This proxy delegates to the Upgrade Logic when the caller is identified as the proxy admin. In other scenarios, it delegates to the Crystal Logic.

The deployment code details are provided in Table 1. The gas required for deploying this Smart Wallet is approximately 130k.

2.3.2.2 Crystal Logic

In the conventional Diamond pattern, every selector is directly mapped to its respective logic address. This configuration implies that introducing a new method incurs a minimum deployment gas cost of 21k. For a smart wallet having a relatively conservative 50 selectors, this equates to a baseline gas expenditure of 1 million for each deployment.

The *Crystal Logic* tackles this limitation by employing contract bytecode for selectors mapping, as illustrated in Figure 3. During deployment, we organize all selectors in a sorted sequence and append them to the end of the file in the format: `[selector, reference to contract]`. Subsequently, the addresses section enumerates all the associated execution addresses.

During call execution, the metadata structure of the *Crystal Logic* is read. A binary search is subsequently conducted over the selectors using the `CODECOPY` code. Then the address reference is resolved to the actual execution address.

The associated computational cost is $O(\log_2 N)$ for the `CODECOPY` execution. As an illustrative example, for 50 selectors, there would be a requirement for 1 metadata read, 6 selector reads, and a single address lookup, resulting in a gas expenditure of 24 units. In contrast, the Diamond pattern would have necessitated a significantly higher cost, at a minimum of 2100 gas units.

For deployment, the associated cost is computed as $200 \times (N \times 6 + M \times 20)$, where N represents the number of selectors and M denotes the number of execution addresses. Assuming an average of 10 selectors per contract, the cost becomes $1600 \times N$. This is approximately 13 times more cost-effective than the $21000 \times N$ required for diamonds.

2.4 TVL estimate

Each smart wallet has a list of tokens (fungible, non-fungible or native) known in advance. The TVL estimate is boiled down to 2 questions:

1. What is the base currency for estimation
2. What is the price of each token in base currency

While our default base currency is USD, certain tokens might first be estimated in USDC or ETH, depending on which is more apt, before being translated into USD.

Token prices can be classified into two primary categories:

- Spot prices
- Derivative prices

2.4.1 Spot prices

Spot prices are used for tokens like UNI, LINK, YFI, etc. The estimated value of these tokens is derived from their trading activity against other notable tokens, like ETH, USDC. Our primary source of reference is the Uniswap V3 spot prices, which is further confirmed with the Uniswap V3 oracle. The workflow for this process is as follows:

1. When token pair is added, the oracle analyzes current liquidity for different fee tiers for this pair and picks the pool with the highest liquidity
2. When the price is queried, the oracle reads the current spot price from the pool
3. This spot price is then cross-verified with the oracle using the intra-block volatility threshold (IBVT). The IBVT represents the maximum allowed price deviation between two consecutive blocks. If the deviation exceeds this threshold, the transaction will be reverted.

2.4.2 Derivative prices

Derivative oracles are used for tokens such as Wrapped Staked Ether (wstETH), Aave USDC (aUSDC), and Uniswap V3 NFT. The underlying principle is that the price of these tokens can be precisely determined based on their underlying assets (which in turn can be priced using Spot Oracles or other Derivative oracles). For every derivative token, creating such an oracle is a straightforward process.

2.4.3 Estimation of Smart Wallet TVL

To determine the TVL within the Smart Wallet, we use the following methodology:

1. Initialize a comprehensive registry encompassing all spot and derivative oracles
2. Query the balances of all tokens within the smart wallet.
3. Classify each token based on whether it is resolved by a spot oracle or a derivative oracle
4. If the token is resolved through a derivative oracle:
    - Resolve its price relative to its underlying assets
    - Append the underlying assets (along with their respective quantities) to the inventory of tokens in the smart wallet
    - Go to step 3
5. If resolved by a spot oracle, convert its value equivalently to USDC or ETH and incrementally add this to the TVL
6. If there are still tokens except USDC and ETH, go to step 3
7. Finally, convert the values of ETH and USDC to USD utilizing the Chainlink Oracle

2.5 Rebalance mechanics

Every smart wallet is equipped with a distinct asset allocation function, designated by the wallet's owner. This function specifies the proportions of various assets the wallet should hold, contingent upon available onchain data. To illustrate, consider a scenario where the asset allocation responds to the ETH/USDC exchange rate:

- If the ETH/USDC price falls below 2000, the prescribed allocation is 30% ETH and 70% USDC.
- Conversely, should the price surpass 2000, an even split is recommended, allocating 50% to both ETH and USDC.

Additionally, the asset allocation function defines the conditions for an "out-of-balance situation". For instance, consider the following scenarios:

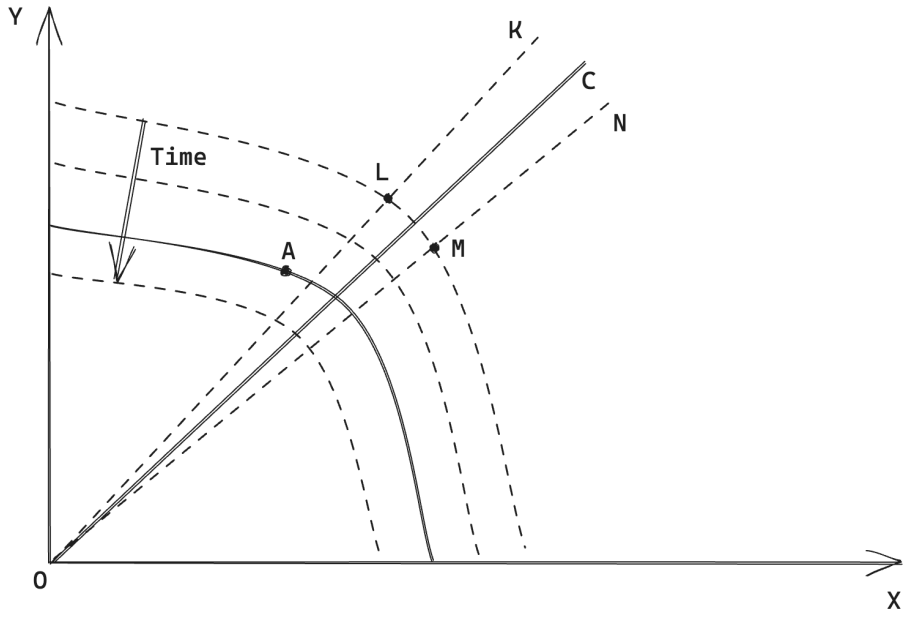

Figure 4: Rebalance mechanics

- If the ETH/USDC price falls below 2000, and the ETH share is either below 25% ETH or above 35%.
- If the ETH/USDC price surpass 2000, and the ETH share is either below 45% ETH or above 55%.

When the system detects an *out-of-balance* situation, it initiates the Dutch auction on TVL. The TVL at any given time t is represented by the equation:

$$TVL(t) = p(t) \cdot TVL_0 \tag{1}$$

Where:

- TVL_0 denotes the current estimated value of the wallet's TVL.
- $p(t)$ is a time-dependent declining function. For instance, it could start at 1.01 and decrease to 0.99 over a span of 1 hour.

A rebalance can be executed by any external account or smart contract. The executing entity has the capability to drain all assets from the vault, make the appropriate swaps, and then return the assets. However, the transaction must adhere to the following conditions:

1. All operations are completed within a single transaction.
2. After all operations, the wallet remains **in balance** (i.e., it is **not** "out-of-balance").
3. After all operations, the TVL remains greater than or equal to $TVL(t)$.

The mechanics of rebalancing are illustrated in Figure 4 for two tokens, namely X and Y. The target asset allocation function is represented by OC. The region defined by KON defines the in-balance state. Conversely, any state not within this region is considered out-of-balance. For instance, point A lies outside the in-balance region, triggering the start of the Dutch auction. The $KLMN$ region defines permissible asset allocations post-rebalance. As time progresses without a rebalance, this region expands, broadening the rebalancing possibilities.

From the perspective of a rebalancer, rebalancing is executed as soon as a profit opportunity is identified. Consider the following example:

Suppose the current vault's TVL stands at \$1 million, and $TVL(t) = \$990k$. If the rebalancer can extract assets from the vault, conduct the necessary swaps, and restore balance at a TVL of \$990.5k, then a profit scenario arises. In this case, the rebalancer would perform the rebalance, return assets worth only \$990k to the vault, and retain the remaining \$0.5k as profit.

From the viewpoint of the vault owner, numerous rebalancers competing for optimal execution. The first to successfully rebalance achieves the highest TVL for the wallet owner, ensuring the most efficient rebalancing process.

2.6 Intent-based architectures

Conventionally, interactions with dApps in the DeFi space have followed an imperative style. An example of such an interaction is a user stating, "I want to swap X token on Uniswap." The introduction of intents has pivoted this narrative towards a more declarative style, where a user might state, "I want to swap X for at least Y tokens, irrespective of the exchange medium used."

In this system, these declarative intentions, or *intents*, are subsequently executed by entities called *solvers* who act on behalf of the intent's originator.

At present, intents are predominantly associated with swap operations in platforms such as UniswapX and Cowswap. Yet, the concept is not limited to just these exchanges.

The *Declarative Liquidity Manager* extends the intent idea to the domain of liquidity management. Within the liquidity management sphere, the imme-

diacy of execution is often not critical. This difference from the swap-focused platforms offers several unique advantages:

- **Open Solver Market:** There are no explicit trust or staking prerequisites for solvers. This openness allows any software developer to join the solver market.
- **Blockchain-native Architecture:** The system is wholly integrated with the blockchain, eliminating the need for potentially vulnerable off-chain services which could act as a single point of failure.

Collectively, these characteristics contribute to a competitive intent execution environment, ensuring an efficient and effective rebalancing process.

3 Use Cases

Fundamentally, DLM serves as a robust and adaptable framework designed for the efficient storage and dynamic allocation of crypto assets. Its very versatile and have a myriad of applications that can push DeFi ecosystem forward.

In this section, we will highlight a few applications, all of which are made viable through the capabilities of the DLM:

1. **Portfolio Management**: DLM facilitates users to organize, monitor, and strategically reallocate their assets, all in alignment with market flux and individual investment objectives
2. **Lending Markets**: Using DLM, it is feasible to architect platforms which bridge the gap between lenders and borrowers very effectively
3. **Cost effective Call/Put Options**: Using the DLM, one can employ a payoff replicating strategy to emulate the payoff of Call/Put options. This approach effectively offers you the benefit of a call option at a fair (and low) premium price, all without the necessity of collateral

3.1 Portfolio Management

The application of the DLM in portfolio management is very straightforward. By utilizing the DLM, investors can easily establish a target asset allocation function for their portfolios. Once set, the DLM takes over, ensuring that the portfolio is rebalanced in line with the predetermined strategy and market prices.

Below are some examples of asset allocation strategies:

1. **Markowitz Efficient Frontier**: This is a foundational concept in modern portfolio theory, proposed by Harry Markowitz. It aims to construct portfolios to optimize or maximize expected return based on a given level of market risk.

2. **Equal Weighting**: This is a simple approach where each asset in the portfolio is assigned the same weight. If there are 10 assets, each would have a 10% allocation.

3. **Value Weighting (or Market Cap Weighting)**: Assets are weighted based on their market capitalization

4. **Risk Parity**: This approach aims to allocate weights to assets based on the inverse of their volatility. Assets with higher volatility are assigned a lower weight, and vice versa.

5. **Inverse Volatility Weighting**: Similar to risk parity, this strategy assigns weights based on the inverse of historical volatility.

6. **Momentum-based Strategies**: Assets are weighted based on their recent performance. It's based on the belief that assets that performed well in the recent past will continue to do so in the near future.

7. **Minimum Variance Portfolio**: This strategy aims to construct a portfolio with the lowest possible volatility by considering the correlation between assets.

8. **Maximum Diversification**: A strategy that seeks to maximize the diversification ratio of a portfolio.

3.2 Lending markets

The DLM offers an approach to lending based on the CDP (Collateralized Debt Position) structures, frequently seen in decentralized finance.

Let's consider a practical scenario for clarity:

Suppose you have an Ethereum (ETH) portfolio and you're aiming to borrow 100 USDC against it, as illustrated in Figure 5. With DLN's adaptive mechanism, once the USDC value of the collateralized portfolio drops below 150 USDC, the system begins a gradual process of converting ETH to USDC. This ensures that by the end of the selling process, your wallet holds a minimum of 125 USDC, providing an adequate cushion to secure the initially borrowed 100 USDC.

It's important to understand a distinct advantage here: the liquidation process is based exclusively on price fluctuations. Unlike conventional liquidation systems where a portion of the collateral might be forfeited to a third-party

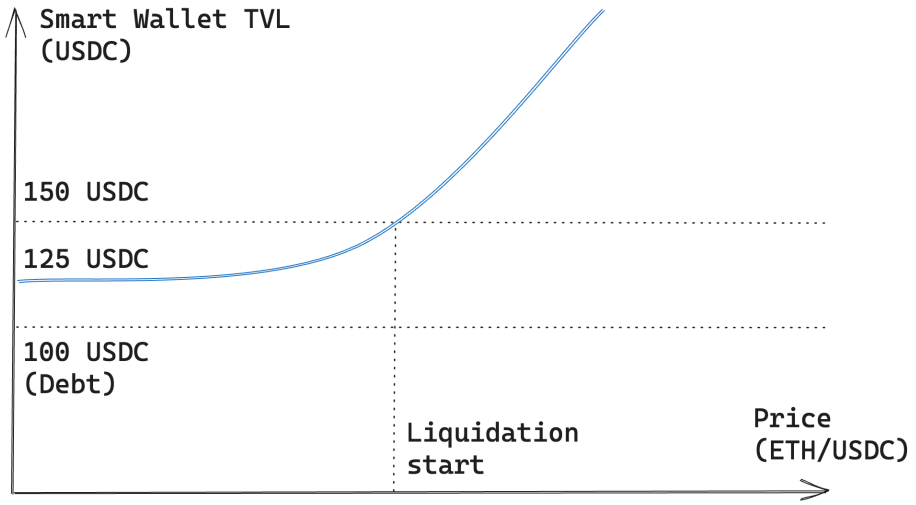

Figure 5: CDP

liquidator, the DLN model eliminates such concerns. This results in a more user-centric and efficient liquidation process.

3.3 Cost effective Call/Put Options

Options are financial instruments that derive their value from an underlying asset. One of the foundational principles in financial mathematics is that certain options can be replicated by a dynamic combination of the underlying asset and a risk-free bond (or cash). Option payoff replication is simply the act of DLN orchestrating a portfolio, based on underlying price shifts, as described in the sections below.

Call Option Replication

Consider a European Call option with an exercise price of K and an expiration time T. To replicate this Call option, an investor can follow these steps:

1. Buy Δ units of the underlying asset, where Δ is the option's delta, representing the change in option price for a unit change in the underlying price.
2. Borrow the present value of ($K - \Delta \times$ current underlying price) at the risk-free rate.

The combination of the underlying asset and the borrowed amount will, at expiration, provide the same payoff as the Call option.

Put Option Replication

To replicate a European Put option with an exercise price of K and an expiration time T, the strategy differs:

1. Short Δ units of the underlying asset.
2. Invest the proceeds, plus the present value of ($K+\Delta\times$current underlying price), in a risk-free bond.

The combination of the short position in the underlying and the risk-free bond will, at expiration, mimic the payoff of the Put option.

4 Summary

The Declarative Liquidity Manager (DLM) represents a significant step forward in the evolution of the DeFi landscape. Through the application of the Safe{Core} SDK and Crystal Pattern systems, DLM offers a more gas-efficient and flexible alternative to traditional smart wallet implementations. These foundational components not only address the persistent challenges of gas costs and upgradeability but also pave the way for modular and scalable DeFi solutions.

Key to DLM's functionality is its rebalancing mechanism, which automatically adjusts asset allocations in response to market conditions, ensuring optimized returns and asset security. By leveraging a Dutch auction system, the rebalance process is made efficient, competitive, and profitable for rebalancers, ensuring the smart wallet remains balanced while maximizing the TVL.

The introduction of intent-based architectures further emphasizes DLM's commitment to user-centricity, shifting from an imperative interaction paradigm to a more declarative one. This approach empowers users to express their desired outcomes, which are then optimally executed by solvers. This system is not only user-friendly but also highly adaptable, making it applicable beyond simple swaps to the broader domain of liquidity management.

DLM's potential applications are vast. From facilitating dynamic portfolio management, optimizing lending markets, to introducing cost-effective financial instruments such as Call/Put Options.