HILDA EXCHANGE

Description

This whitepaper covers all the needed milestones and prerequisites for building a successful, non-custodial, decentralized **Hilda Exchange** for instant ERC20 (TIP-3) token swapping.
The Hild protocol implements:

- Non-custodial censorship resistant instant exchange.
- Constant product automatic market making (**AMM**) liquidity pools for providing liquidity.
- Implements time-tested approaches from Uniswap, Curve, Balancer, Bancor and Moniswap exchanges.
- Rewards both the pools' liquidity providers and traders with a unique *deflationary* governance **Hilda Token**. This token lacks all the inflationary drawbacks known in the Ethereum Defi space and implements special fee-on-transfers tokenomics applied only to selling pressure (not buying), gradual issuance and open market buybacks from the **Treasury Fund.**
- Impermanent loss mitigation for liquidity providers by introducing free to use on-chain oracles like Keep3r. Since in an ordinary AMM, such as Uniswap, it is arbitrageurs and trading bots that come and buy/sell a token in the pool when price is low/high and take all the profits from liquidity providers.
- Customizable exchange fee for liquidity providers and slippage for traders.
- No strict 50/50% for pairs in the pools, you can customize the proportion to adjust risk exposure for any token, e.g. 98/2%.
- Instantly add/withdraw any token pair. No waiting for approval.
- **AragonDAO** for governance.

Abstract

This whitepaper focuses on the architecture and economical aspects of automatic market making non-custodial decentralized **Hilda Exchange**.

Chapter **Research** explains why Free TON ecosystem needs to concentrate more on liquidity hijacking from the Ethereum infrastructure rather than building its own one from scratch and why this approach along with building decentralized AMM exchanges like Hilda Exchange might be a life-blood to make Free TON eventually overcome Ethereum in terms of both usability, wide acceptance and native token value.

Chapter **Impermanent Loss Problem and Ways to Mitigate It** covers in detail the origin of the impermanent loss, under what circumstances it happens, what trading pairs suffer from it most and up-to-day methods used to fix it.

Chapter **Governance** includes the unique features of the deflationary governance **Hilda token** and the time-tested methods and techniques already implemented in the Ethereum decentralized finance (DeFi) which will help to massively attract liquidity providers, arbitrageurs, bots and traders from Ethereum to Free TON architecture.

The Solidity interfaces chapter contains technical description of Smart Contract's interfaces.

This whitepaper was made for the following contest: **Contest Proposal: FreeTon DEX Architecture & Design Proposal** [1].

Research

Decentralized exchanges (DEXes) began to appear simultaneously with centralized ones. But they lacked one important thing - liquidity. The problem was

that DEXes copied the order book matching architecture of the centralized model. The order book matching process is highly dependent on the execution speed. And since every new block on Ethereum is being mined roughly every 14 seconds, orders on decentralized exchanges were both slow and expensive. Not to speak about the problem of finding a counterpart for every working order in the order book. To fix this issue a normal exchange would attract professional agents, called *market makers* who would develop a sophisticated algorithm and buy expensive software/hardware to implement it properly. All the above turned into a high barrier to enter for ordinary investors, many of them, by the way, sitting on piles of idle crypto assets. Moreover, high frequency trading (HFT) was economically unprofitable on DEXes because of high gas fees. Therefore, many DEXes offered very thin liquidity, which was in fact their life blood, and things turned into a vicious circle. Traders were coming to a DEX, placed an order and had to wait for days, sometimes weeks, to get filled. The volume was miserable too. DEXes, like dating websites, need more matching pairs on the other side to function properly. When a person makes a search through a dating app and finds no counterpart to date he/she instantly leaves. And people went away, hopping from one DEX to another, eventually returning to some major centralized exchange (CEX), like Binance. So, CEXes flourished. They were like black holes in space devouring smaller ones and getting bigger and bigger.

Advent of Automatic Market Makers (AMM)

Things started to change when the AMM formula coupled with liquidity pools were used by the first newcomer to the crypto space - Bancor in 2017. This made it possible to trade cryptocurrency without relying on external data for pricing and there was no need in both the order book and order matching.
AMMs absolutely changed the way users swapped cryptocurrencies. New innovative AMM startups as Uniswap, Balancer, Curve, started springing up all over DeFi.
Liquidity Pools allowed users to easily switch (swap) between tokens in a fully

decentralized and non-custodial way. While, **Liquidity Providers** earned a passive income from trading fees that was based on the proportion of their contribution to the pool.

Alan Lu first used the constant-product invariant formula for prediction markets. Hayden Adams then used the beautiful formula for his AMM Uniswap. Uniswap is fully permissionless and can be funded by anyone. The Uniswap AMM works without any external feeds from oracles and basically has no other impacts on pricing apart from the trades executed against it. The elegant AMM formula reads as follows:

$x * y = k$,

where:

x — quantity of tokens x,

y — quantity of tokens y,

k — a fixed constant.

In other words, the equation expresses the dependency between quantities of two assets whose product has to remain constant. With this equation, an AMM is able to quote prices of two assets in a pool to sustain the product of their units equal to a constant.

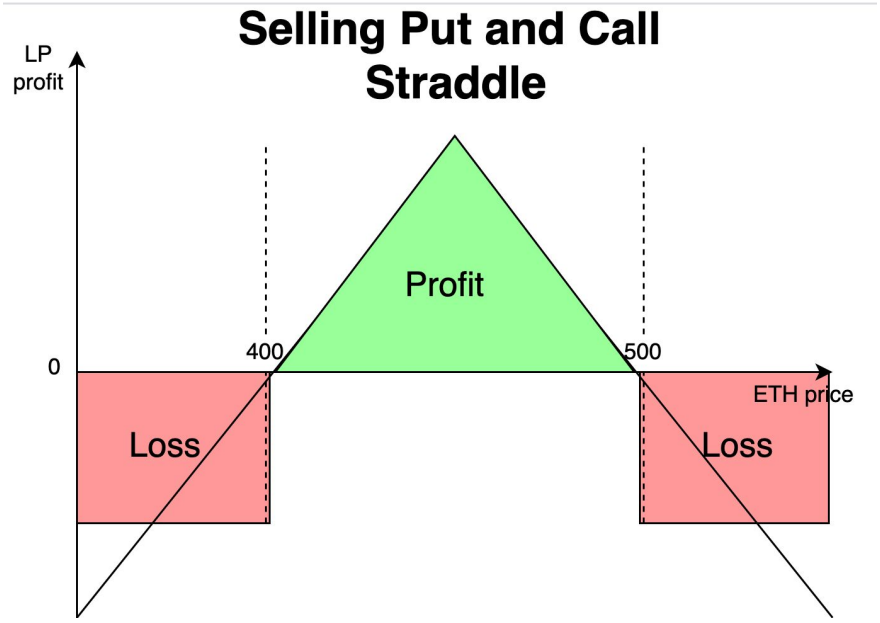
The idea was so brilliant and simple that it was taken up immediately. Now anybody holding ether or any ERC20 token was able to passively earn from 5% to over 60% annualized income. People started to rush adding liquidity into the pools. But in order to do that you have to divide your base asset into 50/50% with a paired token, e.g. if you like to supply liquidity into the ETH-DAI pool, you have to sell half of your ETH for DAI and only after that you can add both tokens into the pool.

So, from now on you become exposed to ETH price swings in both directions. Here the problems started to emerge. Liquidity providers began noticing that in fact they lost money whereas they were supposed to earn transaction fees.

Impermanent Loss Problem and Ways to Mitigate It

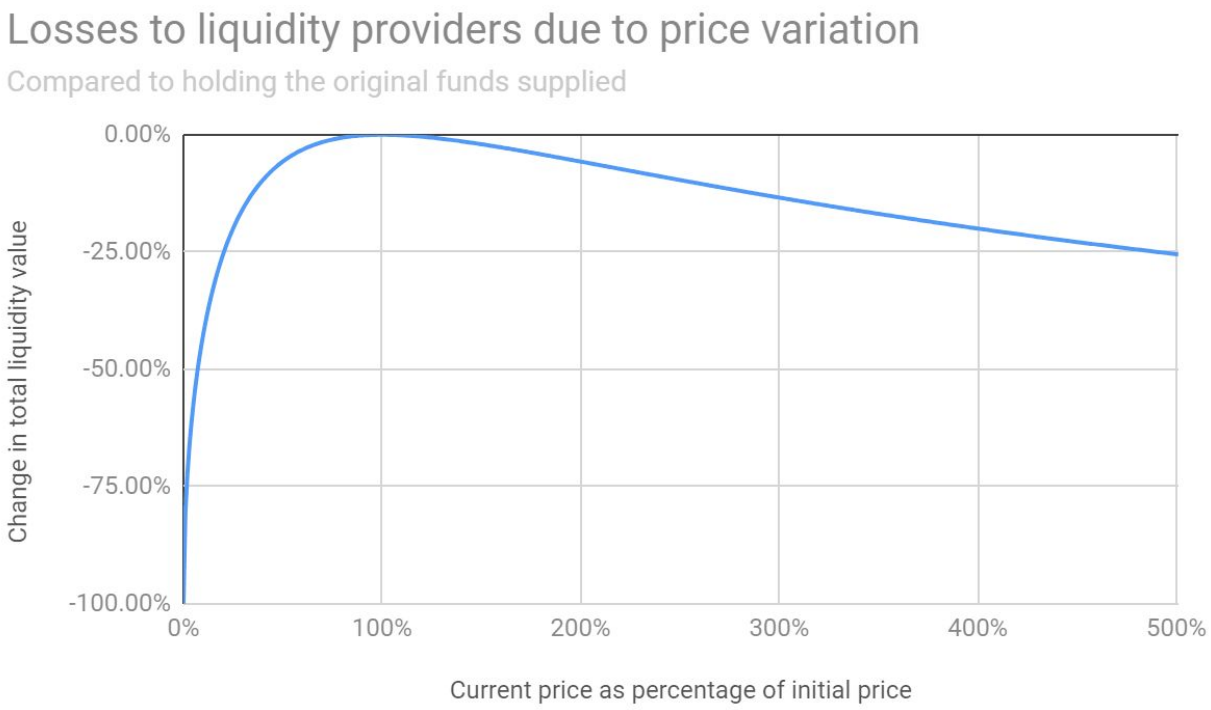
Since the prices in the liquidity pools were not updated by external forces from other open markets, when the prices on the latter changed, the assets in the pools became either under or overpriced. It was therefore not surprising that both trading bots and arbitrageurs came to those pools to take their share of profits.

But an exchange is a zero sum game. If one party wins the other always loses. So did the liquidity providers. Providing liquidity based on the Constant Product Market Maker formula was in fact so dangerous as to selling an option straddle, consisting of both selling a put and call option, if you are familiar with the subject. This structure only works as long as the prices of the underlying assets stay within a certain range, like on the picture below within the green area (ETH trading in the $400 - $500 range).



Pic. 1. Selling a straddle profile of profit and loss

If, say, you want to be long ETH and supply it to the pool at the price of $400 and the price moves to $500 or more, you will end up losing some percent of your portfolio compared to a scenario if you were holding both assets outside the pool in a cold wallet. So, in fact, you must wait for the price of ETH to return to $400 in order to successfully withdraw from the pool. If that doesn't happen your **Impermanent Loss** will turn into permanent and you will lose some amount of your ETH. In practice, you will earn a bit more DAI but that won't cover your loss in ETH. The chart below shows the extent of your losses.



Pic. 2. Impermanent loss chart

On the other hand, if you don't want to be exposed to ETH market price and only hold DAI, but the ETH falls from $400 to $300, you still suffer from the impermanent loss. You can click on the link AMM Impermanent Loss make a copy on your google

drive and play with it setting the price in J column to see how impermanent loss accumulates over time.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
2			TRADE			POOL								
3	Days		DAI to draw	ETH to draw	ETH/DAI price in the pool	DAI in the pool	ETH in the pool	Product	ETH/DAI	ETH/DAI market		HODL (denominated in DAI)	Total in the Pool denominated in Dai	ILoss Actual
4	0	Initial supply	100,000	1,000	100	100,000	1,000	100,000,000	100.00	100				
5	1	Swap	-22,474.5	183.5	122.5	122,474	816	100,000,000	150.00	150	1.50000	250,000	244,949	-2.02%
6	2	Swap	-407.6	2.7	150.5	122,882	814	100,000,000	151.00	151	1.51000	251,000	245,764	-2.09%
7	2	Swap	-406.2	2.7	151.5	123,288	811	100,000,000	152.00	152	1.52000	252,000	246,577	-2.15%
8	2	Swap	-404.9	2.7	152.5	123,693	808	100,000,000	153.00	153	1.53000	253,000	247,386	-2.22%
9	2	Swap	-403.6	2.6	153.5	124,097	806	100,000,000	154.00	154	1.54000	254,000	248,193	-2.29%
10	2	Swap	-402.3	2.6	154.5	124,499	803	100,000,000	155.00	155	1.55000	255,000	248,998	-2.35%
11	2	Swap	-401.0	2.6	155.5	124,900	801	100,000,000	156.00	156	1.56000	256,000	249,800	-2.42%
12	2	Swap	-399.7	2.6	156.5	125,300	798	100,000,000	157.00	157	1.57000	257,000	250,599	-2.49%
13	2	Swap	-398.4	2.5	157.5	125,698	796	100,000,000	158.00	158	1.58000	258,000	251,396	-2.56%
14	2	Swap	-397.2	2.5	158.5	126,095	793	100,000,000	159.00	159	1.59000	259,000	252,190	-2.63%

Pic. 3. Impermanent loss accumulation table

To force liquidity providers to keep their assets in the pools two ideas emerged. The first one was incentivising them with governance tokens. Compound was the first to introduce COMP token in March 2020 and the whole idea turned out to be quite successful followed by other projects like Balancer, Yearn (YFI), 1inch, Curve and by their countless forks. You probably heard the term **Yield Farming**. It was exactly the time when it was coined. Everybody and his dog then stampeded into a new gold rush for yields. Annualized return on investment called APY for short at times reached a whopping 5000% and people pursuing such abnormal high yields started to be called “degenerators” or “degen farmers”. Degenspartan twitter account was the Akkela of the wolf pack.

By the way, the Hilda Exchange logo has an element of the spartan helmet, since the Tolkien’s name Hilda denotes a woman, who sells Athelas plants and is an efficient market trader. The name also means “battle”.

Yield farmers rushed into every new project in order to catch a few hours of the maximum yields before others joined the pool and diluted their share. Then they dumped the earned governance token on the open market making its price fall,

sometimes in a death spiral. As a result, liquidity provision had progressively turned into a game of “hot money” where short-term LPs hopped from pool to pool trying to extract enough rewards to compensate for the risk of impermanent loss.

Some called this time a “Cambrian Explosion” of species, because it was the time when every day some new startup sprang up or a fork of another fork.

Since the governance tokens had the inflationary nature, it took the market about half a year to finally become sane, those tokens soon got devalued and everybody lost interest in such a business. But here is the catch.

To combat the impermanent loss Hilda Exchange introduces several ways.

1. A time-tested method - generously rewarding liquidity providers with a lucrative *deflationary* governance **Hilda Token**, lacking all the drawbacks of the inflationary ones and what’s more important it’s already successfully applied in a number of startups across the Ethereum ecosystem. It is described in detail in the corresponding **Governance** and **Hilda Token** section.
2. Inter-blockchain oracles to timely update the prices in the pools, which are covered in the **Inter-blockchain UniswapV2Sliding Oracles** chapter.

Single-Sided Liquidity

Another way to fight the impermanent loss was a method, called **Single-Sided Liquidity** introduced by Bancor in their v2.1 rollout. From the point of view of an ordinary investor, providing only a single side of the pool with a token of choice was the best solution. If you are long ETH and don’t want to lose on the upside, then you don’t have to sell half of your ETH for DAI in order to supply liquidity to the ETH-DAI pool. You just supply a single ETH!

It turned out to be great only in theory. We were the first to get disappointed by the actual implementation of this technique by Bancor and even dedicated a few Twitter posts to this. In reality the investor was 100% covered against the impermanent loss (IL) only after a three month holding period. There is no need to remind you what could happen on the market during such a large time frame.

The second setup was the form of compensating your loss. You think it was done by the token you supplied in the pool? Not at all. Where Bancor would get such amounts if they don't control the emission? Right! You could only be compensated by the native Bancor governance token BNT and, of course, you carried all the risks associated with its market price, which is highly volatile, by the way. We tried to ask a number of questions regarding the exact moment the liquidity provider was compensated: the moment he supplied or withdrew from the pool, only we didn't get any clear feedback on this.

Bancor was also the first to resolve high slippage on large trades and impermanent losses with **liquidity amplification** and **dynamic weights** mechanisms in their v2 update. But the floating weights solution for mitigating impermanent losses requires a smooth flow of price feeds from oracles that operate externally to the protocol. Bancor found this to be a critical drawback as oracles are vulnerable to front-running attacks.

Hard-pegged And Soft-pegged Tokens To Combat The Impermanent Loss

Quite soon traders realized that highly correlated tokens don't suffer from the impermanent loss at all. For example Tether USDT and Circle USDC both track the US dollar and can be called hard-pegged. MakerDAO DAI coin tracks the US dollar algorithmically and therefore is soft-pegged to USD. With the creation of Curve protocol giving better prices on stablecoins, Uniswap's retail traders and liquidity

providers started switching to more profitable sources. Curve developed a specific pricing function that consists of a constant product and constant sum. Thanks to the pool imbalance coefficient embedded in both sides of the equation, Curve's AMM was able to quote prices more effectively. The AMM switches between constant sum and constant product formulas based on the extent to which the pool is imbalanced at the moment of query. It was not a surprise that Curve's total locked value (TVL) soon reached and later exceeded one billion USD.

Curve's innovation greatly contributes to the DeFi space but does not solve some of the most notorious problems.

Prerequisites For Successful Liquidity Hijacking from Ethereum to Free TON

Free TON ecosystem doesn't need to build liquidity from scratch in order to take the top place in DeFi ratings. It can go the already beaten path other projects successfully did. Our Hilda Exchange team did a thorough research on this and is competent enough to implement already battle proven methods to migrate the Ethereum liquidity into the Free Ton version of DeFi. We are confident on the unchartered territory either.

Sushiswap hijacked more than a half of Uniswap's liquidity in a matter of days by just introducing the governance token. YFII successfully sucked out most of the YFI's liquidity to such an extent that Andre Cronje with YFY was forced to mine YFII token in order to keep them at bay. Project's leader learned a hard lesson to always keep the bar high in order to constantly attract and incentivise liquidity providers.

Still, there are some prerequisites to make this migration as smooth as possible for Hilda Exchange.

First, we need functioning ERC20 - TIP3 bridges. We know from experience that in order to succeed one must concentrate on one thing only and do it as best as they can. This is why we decided to concentrate on the Ethereum plefora of ERC20 tokens to make the migration to Free TON as smooth as possible for the end user. Some juries might object why we don't need BTC - TON bridges etc. Because, there is quite a simple way to swap BTC to renBTC using the REN project decentralized bridge, which we use ourselves on a regular basis. REN team is working hard at implementing bridges between other blockchains and Ethereum, even as hard as Monero.

Ideally, it should take only a click of a mouse for an Ethereum degen investor in the MetaMask browser plugin to swap, for example 10 000 CRV (Curve's native token) to exactly the same amount of 10 000 CRV TON minus commission. We see this process as the following.

The smart contract on the Ethereum side locks 10 000 CRV and signals to the smart contract on the Free TON side to mint 10 000 CRV TON which is instantly credited to the users wallet. From now one the user can easily swap this token on Hilda Exchange for any other token of his choice in pursuit of making alpha. Say, the user deemed the YFI TON (Yearn protocol) token undervalued in one of the Hilda Exchange pools and swapped 1000 CRV TON for 1 YFI TON. Then he goes back to the bridge and swaps his 1 YFI TON to 1 YFI Ethereum ERC20. The smart contract on the Free TON side burns 1 YFI TON and unlocks 1 YFI Ethereum ERC20. The user is happy and wants to do more swaps on Hilda Exchange in pursuit of profit.

Now he constantly monitors prices in Hilda pools to catch inefficiencies, even builds automatic trading bots, refers friends etc. Add to this our deflationary Hilda token rewards which is supposed to gradually rise in price and we have crowds of traders waiting in line to supply liquidity and make swaps on Hilda Exchange.

In case nobody builds a decent and satisfactory ERC20 - Free TON bridge by the time we make the alpha version of Hilda Exchange, we have plan B to build this bridge ourselves, that meets our specific requirements.

Requirements

- Smoothly functioning ERC20 Ethereum to Free TON ERC20 or TIP-3 bridge.
- FreeTON Wrapper Contracts functioning as a backend to implement all the logic of HildaFactory smart contract, HildaRouter, HildaToken and some extra logic to mitigate front-running (fully described in **Front-running and Ways to Prevent It** chapter) and use Hilda meta-pools and single-sided liquidity to their maximum.
- React or Vue front-end build both for the legacy web 2.0 and for web 3.0 IPFS (Interplanetary File System) website.
- A user-friendly GUI Free TON wallet, supporting ERC20 and TIP-3 assets to be a decent alternative to MetaMask browser plugin for smooth user interaction with Hilda Exchange.

Additionals

- Non-custodial decentralized approach.
- No need for trusted intermediaries.
- Censorship resistant.
- Open-sourced.
- Anyone can instantly create a new trading pair of any ERC20 tokens on Hilda Exchange.
- Customizable ratio of any two tokens when supplying liquidity in the pool. Not only 50-50% but any arbitrary ratio, like 98-2%. The latter is often needed to mitigate the risk of the volatile token in the pair.

- Customizable trading fees when creating a liquidity pool, ranging from 0 to 10%.
- Multiple tokens per pool (2=< Number Tokens =<8) as implemented in Balancer.
- To protect Hilda Exchange from any possible attack vectors we are going to make regular audits of our smart contracts by attracting authority third parties in the industry.
- Deflationary token which is built to sustain its price and gradually rise to reflect the rising value of Hilda Exchange.
- DAO community. When Hilda Exchange reaches a curtain milestone down its road map, we are going to burn our admin keys and transfer all the governance power to the community as successful projects do in the Ethereum ecosystem.
- A clear and sustainable economic model, where the portion of trading fees goes to the **Development Fund** - to hire the best devs in the sphere, **Treasury Fund** - to open-market buy **Hilda Token** at price levels we deem appropriate in order not to destroy our book value. And the **Emergency Fund** to deal with any unforeseen circumstances.
- Modular software approach .

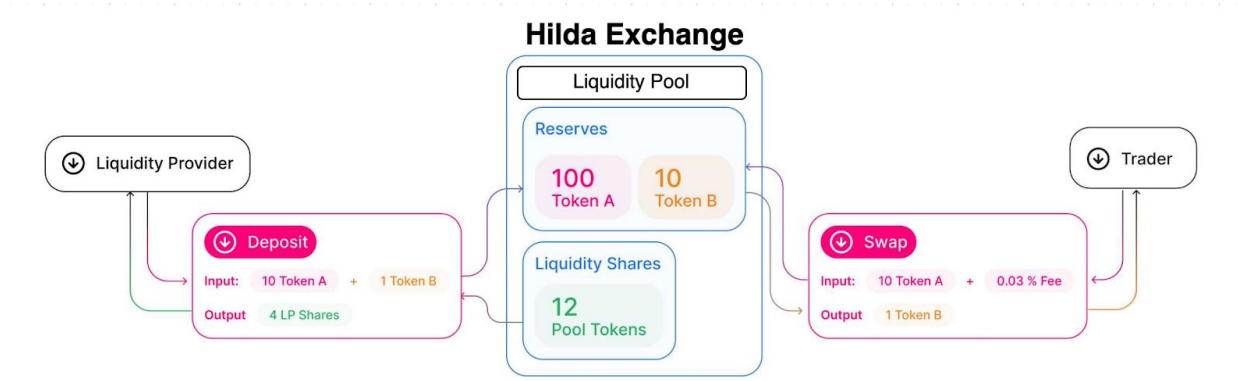
Hilda Exchange architecture

Basic Structure

Hilda Exchange smart contracts manage liquidity pools made up of reserves of two or more (up to eight) ERC-20 (TIP-3) tokens.

Anyone can become a liquidity provider (LP) for a pool by supplying an equivalent value of each underlying token in return for pool tokens. These tokens track pro-rata LP shares of the total reserves, and can be redeemed for the underlying assets at any

time.



Pic. 4. Architecture of AMM pool

Pairs act as automated market makers (AMM), standing ready to accept one token for the other as long as the “constant product” formula is preserved. This formula, x * y = k, states that trades must not change the product (k) of a pair’s reserve balances (x and y). Because k remains unchanged from the reference frame of a trade, it is often referred to as the invariant. This formula has the desirable property that larger trades (relative to reserves) execute at exponentially worse rates than smaller ones.

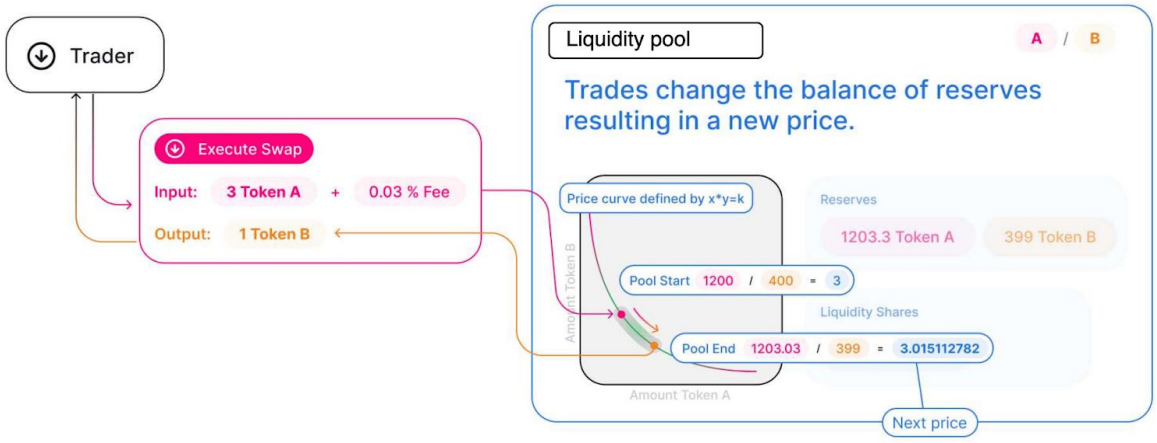
You can play with it yourself and see how the pool sets the price here AMM Impermanent Loss on the **getAmoutOut** tab. Make a copy of the file and change the price in the B:4 cell. Initially in the pool there were 200 of token A and 200 of token B as on the picture below. So the A/B price was 1.00 in the A:4. But if someone comes to the pool with 1000 of A tokens to buy as many B tokens as possible, he will be able to buy only 166.66 B tokens (G:4) at the 6.00 to 1 price (A:5).

	A	B	C	D	E	F	G	H
1		Without 0.3% Fee						
2		Input			Out			
3	A/B Initial Price	amountIn (Token A)	reserveIn (Token A)	reserveOut (Token B)	numerator	denominator	amountOut (Token B)	A/B Resulting Price
4	1.00	1000	200	200	200000	1200	166.6666667	6.00
5	6.00		1200	33.33333333				

Pic. 5. Price formation in the AMM pool

That's why it's practically impossible to buy out the whole pool, as some developers suggested on the Ton forum, in order to cope with liquidity dilution.

Hilda Exchange applies a 0-10% fee to trades, which is added to reserves. The fee is set by the initial creator of the pool. As a result, each trade actually increases k. This functions as a payout to LPs, which is realized when they burn their LP pool's tokens to withdraw their portion of total reserves.



Pic. 6. Trading process in the AMM pool

Because the relative price of the pair assets in the pool can only be changed through trading, divergences between the Hilda Exchange price and external prices create arbitrage opportunities. As was already mentioned in the **Research** section

arbitrageurs “rob” liquidity providers of their profits causing the impermanent loss. To prevent this we are going to implement the system of inter-blockchain oracles.

Inter-blockchain UniswapV2Sliding Oracles

We have two steps to implement here. At first it’s not crucially important to update the prices in Hilda Exchange pools. Market forces will make their job by bringing both traders and trading bots from Ethereum blockchain. It will serve as a bait to hijack Ethereum’s liquidity. All liquidity providers will be compensated for the impermanent loss with the deflationary **Hilda Token**. Once the platform accumulates enough liquidity and mutures we will start using the inter-blockchain **UniswapV2SlidingOracle** made by Keep3r team, which are based on Uniswap V2 oracles.

UniswapV2SlidingOracles are sliding window oracles that use observations collected over a window to provide moving price averages in the past *windowSize* with a precision of *windowSize / granularity*.

The *windowSize* is based on the *granularity* supplied by the user. There is a reading every *periodSize* minutes.

Price Feeds

Data Freshness

```
// returns the amount out corresponding to the amount in for a given token
using the moving average over the time
function current(address tokenIn, uint amountIn, address tokenOut) external
view returns (uint amountOut)
```

Example:

```
interface IUniswapV2Oracle {
  function current(address tokenIn, uint amountIn, address tokenOut)
external view returns (uint amountOut);
}

...

IUniswapV2Oracle public constant UniswapV2Oracle =
IUniswapV2Oracle(0xCA2E2df6A7a7Cf5bd19D112E8568910a6C2D3885);

...

uint _ethOut = UniswapV2Oracle.current(WETH, 1e18, YFI);
```

Security

A quote allows the caller to specify the *granularity* or amount of points to take. Each point is *periodSize*, so 24 points would be *24 * periodSize windowSize*

Data freshness is decreased for increased security

```
// returns the amount out corresponding to the amount in for a given token
using the moving average over the time taking granularity samples
function quote(address tokenIn, uint amountIn, address tokenOut, uint
granularity) external view returns (uint amountOut)
```

Price points

Returns a set of price points equal to *points * periodSize*, so for the points in the last 24 hours use *points = 48*

```
// returns an amount of price points equal to periodSize * points
prices(address tokenIn, uint amountIn, address tokenOut, uint points)
external view returns (uint[] memory)
```

Hourly

Returns a set of price points equal to **points * hours**, so for the points in the last 24 hours use points = 24

```
// returns an amount of price points equal to hour * points
function hourly(address tokenIn, uint amountIn, address tokenOut, uint
points) external view returns (uint[] memory)
```

Daily

```
// returns an amount of price points equal to days * points
function daily(address tokenIn, uint amountIn, address tokenOut, uint
points) external view returns (uint[] memory)
```

Smart Contract Structure

HildaCore

The **HildaCore** consists of a singleton **HildaFactory** and many **pairs**, which the factory is responsible for creating and indexing. These contracts are quite minimal, with a smaller surface area, less bug-prone, and more functionally elegant. Perhaps the biggest upside of this design is that many desired properties of the system can be asserted directly in the code, leaving little room for error. One downside, however, is that core contracts are somewhat user-unfriendly. In fact, interacting directly with these contracts is not recommended for most use cases. Instead, a periphery contract should be used.

HildaFactory

The factory holds the generic bytecode responsible for powering pairs. Its primary job is to create one and only one smart contract per unique token pair. It also contains logic to turn on the protocol charge.

HildaPairs

Pairs have two primary purposes: serving as automated market makers and keeping track of pool token balances. They also expose data which can be used to build decentralized price oracles.

HildaPeriphery

The periphery is a constellation of smart contracts designed to support domain-specific interactions with the core. Because of Hilda Exchange's permissionless nature, the contracts described below have no special privileges, and are in fact only a small subset of the universe of possible periphery-like contracts.

HildaRouter

The router, which uses the library, fully supports all the basic requirements of a front-end offering trading and liquidity management functionality. Notably, it natively supports multi-pair trades (e.g. x to y to z) and offers meta-transactions for removing liquidity.

WorkFlow

There are two main cases which should be considered: **Providing Liquidity** to pools and **Swapping** one asset (token) for another.

Providing Liquidity workflow:

1. Searches through the available pools on Hilda Exchange website to find a pair (or more) he wants to supply. For example, wrapped Bitcoin (WBTC) and wrapped ether (WETH).
2. Connects his wallet similar to MetaMask browser plugin.
3. If the WBTC-WETH pool is already present in the system, the investor supplies two equal amounts of both tokens denominated in US Dollars. For example 1 WBTC and 34 WETH.
4. Chooses maximum slippage like 0.2 - 0.5%. *We have strong hope we will be able to reduce this slippage close to as close to zero as possible on Free Ton architecture*.
5. In return the investor gets 1 **Hilda-LP token**, which represents his share in the pool.
6. If the investor doesn't hold both amounts needed to supply, he can opt for providing a single asset, like WBTC, which will be seamlessly swapped for the necessary amount of WETH and added into the pool, giving the investor the same 1 **Hilda-LP token**.
7. If the the WBTC-WETH pools doesn't exists, the investor simply **creates a new pool**, and does the following:
    a. Enters the number of desired tokens, in our case WBTC and WETH.
    b. Sets the ratio between tokens, for instance, 50% to 50%, but this can be any arbitrary number, like 98% and 2%. This might come in handy when the investor is afraid the second token might fall in price substantially, thus minimizing his risks.

   c. Sets the trading fee, choosing from 0 to 10% (subject to governance voting).
   d. Supplies liquidity as in item 3 above.
8. We reward liquidity providers with the deflationary **Hilda Token** for providing a **white listed number of tokens** which our community deems valuable to the Hilda ecosystem.
9. When the investor decides to withdraw, he simply goes to the Hilda Exchange website, connects his wallet and hits the *withdraw from the pool* button. All the Hilda-LP tokens are swapped back to the corresponding number of tokens he supplied to that pool. That's simple as this.

Swapping and Trading Workflow:

1. A trader wants to swap UNI to CRV.
2. He goes to the Hilda Exchange website, connects his wallet, in the *sell box* enters the amount of UNI he wants to sell and in the *buy box* just the ticker CRV.
3. The system calculates the price and offers to the investor the amount of CRV to buy.
4. The trader chooses the maximum slippage like 0.2-0.5% and signs the transaction in his wallet.
5. After the transaction is mined, he gets the desired number of CRV tokens right in his wallet. **No deposit and no withdrawal are necessary!** Everything is done instantly.
6. It might be possible from time to time at the will of our community to reward some traders with the deflationary **Hilda Token** for trading a **white listed number of tokens** which our community deems valuable to the Hilda ecosystem.

Governance

The ultimate goal down the road is creating a fully fledged Hilda DAO, where we completely transfer the control of smart contracts to the community. But this, based on our rich experience, is going to happen no sooner than the community matures, we get enough feedback and weed out all the bugs from the code.

Most likely we are going to use one of the already time-tested frameworks such as **AragonDAO**, but it will be discussed and decided later down the road map.

All holders of the governance **Hilda Token** are entitled to cast their vote in decentralized manner on all major changes and improvements of the Hilda protocol such as:

1. Setting/changing the size of the **Development Fund**, the proceeds of which are going to be used to hire the best talent in the industry.
2. Setting/changing the size of the **Treasury Fund**, used to buy back Hilda Token on the open market to support its price.
3. Setting the range of trading fees used in the pool. Currently at 0-10%.
4. Changing the amount of transaction fee on Hilda Token and the direction it is applied. Currently it is applied to reduce the selling pressure only.
5. Changing the number of tokens in the pool. Currently from 2 to 8.
6. Adding new tokens into the **White List** and removing those performing bad.
7. Deciding which pools should receive the most weight in Hilda Token allocation.
8. Deciding on the change of quorum of votes needed to pass a decision. Vote signaling times and final decision terms.
9. Setting/changing the admin fee from transactions going into the Dev fund.

Hilda Token is going to **be locked** in order to cast a vote to prevent misuse by bad players.

Of course, the killer feature of governance will be the **Hilda Token** with its unique deflationary characteristics described in detail in the **Hilda Token** section below. This token is designed from our rich experience in a number of other successful and not very so protocols, so we made a good home work to make everything as perfect as we can.

Hilda Exchange Economics

The Hilda team is here for the long term. So we designed the economics to make both short-term profits to finance high priority current operations as well long term plans in order to build a deep moat around our business and make it economically sustainable.

Liquidity providers earn profits in the form of fees set at the creation time of the pool. It is not possible to change the fee once the pool is created. Anyone can create a new pool with a different fee. Some may argue it is going to dilute liquidity, though our experience shows it is of minor significance at this early stage. Trading bots and aggregators have no problem collecting liquidity from all the pools and making swaps in one transaction for the end user.

Liquidity providers also earn the deflationary **Hilda Token** allocated to them based on their share in the pool.

Traders use Hilda Exchange to swap one token for another and pay commission for this based on the fee of a specific pool. These fees go to **Liquidity Providers.**

We divide the life cycle of our project into nascent and mature phases.

Nascent Phase

It is going to be a **liquidity hijacking** from Etherum into Free TON. We planned carefully the steps in such a way that our offer would become so irresistible to “degen farmers” that they would rush into Hilda Exchange in droves.

So we designed a transaction fee of 1-2% applied to swapping of Hilda Token when it is sold across our pools. 10% of it goes into the Development fund to finance R&D and hire top-notch developers. The rest 90% of profits is divided between a portion going back to pools to increase the intrinsic value of the Hilda Token and the rest is going to be used to buy backs of Hilda token in the open markets. By the way, it can even be outside markets in the Ethereum ecosystem.

We should be very flexible here in order to keep the delicate balance of psychological sentiment of the community and can change the ratio going between buybacks and into the pools.

Compensation with Hilda token is going to be the main force driving liquidity from Ethereum into the Free TON. Since it has a deflationary nature, making its supply steadily shrinking, its value is supposed to be steadily rising as well as the price. We will make our best to incentivise both liquidity providers and traders to buy and keep Hilda Token long-term as the YFI team successfully did.

As we accumulate enough liquidity, possibly from 100 to 500 mln, we will switch to the second phase.

Mature Phase

It would be marked by the creation of the **DAO** (most probably based on AragonDAO) run by the community. All the admin keys will be burned and access to the owner functions of the smart contracts will be transferred to the community

governance members. From now on all the decisions will be made by the community voting on-chain or any other more preferable way.

We will introduce the **Admin Fee** as a 5% from the fees collected from trading and swapping in all Hilda Pools. This will go into the Development Fund and Treasury Fund.

We may also put to vote the creation of the Emergency (Insurance) Fund in order to cover any unforeseen losses.

We may enable **one sided liquidity** additions by matching providers with each other. In addition to that, we mitigate impermanent loss further by allowing users to short or long as a “hedge” and cover the potential losses from market movements, creating an unprecedented safety to liquidity providing.

This feature removes impermanent loss by adding one sided liquidity exposure while earning liquidity fees. It allows shorting and longing positions in a decentralized environment by taking on the liquidity providers risk, bringing an important tool to DeFi.

If the community deems appropriate we may make additional emissions of Hilda Token. From experience we learned that sometimes the disagreement between members on this subject may lead to community separation and forks. Therefore it is a very peculiar subject and by no means it should influence the price of Hilda token.

Hilda Token

“Farming” tokens have a problem for their owners. To keep users farming, projects have to mint more ever more coins. This completely destroys the value of the underlying token, due to excessive inflation. It's easy to find examples of this

across the DeFi ecosystem (UNI, CRV, SWRV etc).
Our solution is called deflationary farming, and it is quite simple in only two steps:

1. Charge a fee on token transfers
2. Users can earn the fee by farming

This simple process means that those holding tokens are able to farm without infinite inflation.

The Black Hole of Liquidity

The architecture of Hilda Token creates several compounding loops inside the system which directly affect the token value. Every coin that runs through Hilda Token's ecosystem is charged with a fee which gradually raises the valuation of the whole system, namely pools where Hilda Token is swapped.

Since our token will be tradable on Ethereum first, we need to insure its price keeps stable and gradually rising. Uniswap has a transaction fee of 0.3% which it distributes to liquidity providers. Hilda Token's LP tokens have been modified to direct this fee into the liquidity pool where it is added to the total value locked (TVPL). This means that volume directly grows liquidity.

Each transfer of Hilda Token has a 1-2% fee attached to it. 50-80% of this fee is collected by the staked LP token holders, the 40-10% (to be voted) goes to the Treasury Fund used for buybacks and 10% goes to the dev fund. A positive feedback loop is generated when LP stakers sell their rewards. They generate additional fees which lock up more liquidity and drive up APY (return on investment). An increase of APY attracts additional minting of LP tokens which increases the TVPL as well as the value of the next liquidity addition.

Hilda Token is a pure utility governance token, which has zero initial value and its price is set solely by market forces such as supply and demand. The token gives

the right to participate in governance voting on major community decisions. Total amount yet to be disclosed due to severe competition and the initial tranche is going to be distributed in the following way:

1. 10-20 % for the developers team.
2. 60% to liquidity providers.
3. 20% to traders in the whitelisted pools.

Front-running And Ways to Prevent It

What is Front-running?

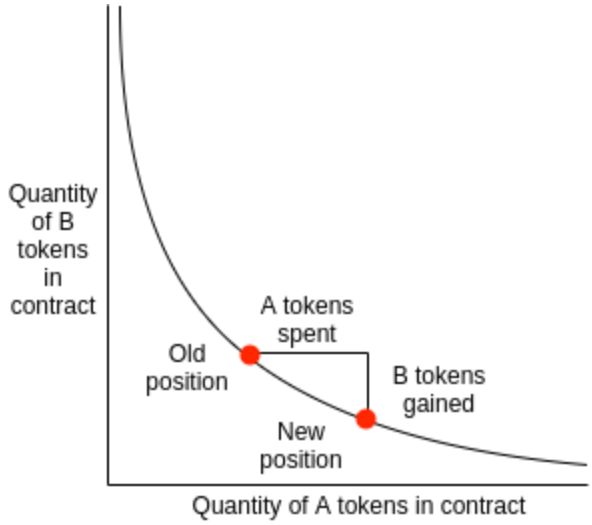
Front-running is simply trying to get in the line first, before another pending transaction is mined and make a profit from it. It is a way more serious problem for Ethereum than one may think. And the good news for Free TON? Lets see.

We'd like to point out **two levels of front-running**. The **higher level** happens on the level of the exchange itself, namely on the smart contract. For example, SushiSwap just locked up liquidity in the SushiLock smart contract to respond faster to new transactions than an ordinary trader trying to front-run by hand, since it is deemed closer to the original request. But in our view, solving a problem in such a way is like trying to strengthen the front door, while leaving open the windows.

The second, **low level** is much more serious and is inherent to the Ethereum Virtual Machine. Recent research has revealed that it is the miners who now became the most sophisticated front-runners. According to the common logic you may think that they must approve transactions with higher gas prices first. Far from it! They often push through transactions with lowest gas to be mined first, even ridiculously priced, wrapping other transactions in the pool with their own trying to make profit on both sides: mining and front-running.

Down below we offer a number of solutions to this which we hope will be implemented in the Free TON architecture sooner or later.

The idea is that you have a contract that holds x coins of token A and y coins of token B, and always maintains the invariant that x*y=k for some constant k . Anyone can buy or sell coins by essentially shifting the market maker's position on the x*y=k curve; if they shift the point to the right, then the amount by which they move it right is the amount of token A they have to put in, and the amount by which they shift the point down corresponds to how much of token B they get out.



Pic. 7. Front-running explained

Notice that, like a regular market, the more you buy the higher the marginal exchange rate that you have to pay for each additional unit (think of the slope of the curve at any particular point as being the marginal exchange rate). The nice thing about this kind of design is that it is provably resistant to money pumping. No matter how many people make what kind of trade, the state of the market cannot get off the curve. We can make the market maker profitable by simply charging a fee, eg. starting from 0.3% up to 10%.

However, there is a flaw in this design: it is vulnerable to **front running attacks**. Suppose that the state of the market is (10, 10), and I send an order to spend one unit of A on B. Normally, that would change the state of the market to (11, 9.090909), and I would be required to pay 1.00 A coin and get 0.909091 B coins in exchange. However, a malicious miner can “wrap” my order with two of their own orders, and get the following result:

1. Starting state: (10, 10)
2. Miner spends one unit of A: (11, 9.090909), gets 0.909091 units of B
3. I spend one unit of A: (12, 8.333333); I get 0.757576 units of B
4. Miner spends 0.757576 units of B: (11, 9.090909), gets 1 unit of A

The miner earns 0.151515 coins of profit, with zero risk, all of which comes out of my pocket.

Methods to Prevent Front-running

Now, how do we prevent this? One proposal is as follows. As part of the market state, we maintain two sets of “virtual quantities”: the A-side (x, y) and the B-side (x, y). Trades of B for A affect the A-side values only and trades of A for B affect the B-side values only.
Hence, the above scenario now plays out as follows:

1. Starting state: ((10, 10), (10, 10))
2. Miner spends one unit of A: ((11, 9.090909), (10, 10)), gets 0.909091 units of B
3. I spend one unit of A: ((12, 8.333333), (10, 10)); I get 0.757576 units of B
4. **Miner spends 1.111111 units of B: ((12, 8.333333), (9, 11.111111)), gets 1 unit of A**

You still lose 0.151515 coins, but the miner now loses 1.111111 - 0.909091 = 0.202020 coins; if the purchases were both infinitesimal in size, this would be a 1:1 griefing

attack, though the larger the purchase and the attack get the more unfavorable it is to the miner.

The simplest approach is to reset the virtual quantities after every block; that is, at the start of every block, set both virtual quantities to equal the new actual quantities. In this case, the miner could try to sell back the coins in a transaction in the next block instead of selling them in the same block, thereby recovering the original attack, but they would face competition from every other actor in the system trying to do the same thing; the equilibrium is for everyone to pay high transaction fees to try to get in first, with the end result that the attacking miner ends up losing coins on net, and all proceeds go to the miner of the next block.

In an environment where there is no sophisticated market of counter-attackers, we could make the attack even harder by making the reset period longer than one block. One could create a design that's robust in a wide variety of circumstances by maintaining a long-running average of how much total activity there is (ie. sum of absolute values of all changes to x per block), and allowing the virtual quantities to converge toward the real quantity at that rate; this way, the mechanism is costly to attack as long as arbitrageurs check the contract at least roughly as frequently as other users.

A more advanced suggestion would be as follows. If the market maker seems to earn profits from the implied spread from the difference between the virtual quantities, these profits could be allocated after the fact to users who seem to have bought at unfair prices. For example, if the price over some period goes from P1 to P2, but at times in between either exceeds P2 or goes below P1, then anyone who bought at that price would be able to send another transaction after the fact to claim some additional funds, to the extent that the market maker has funds available. This would make griefing even less effective, and may also resolve the issue that makes this kind of market maker fare poorly in handling purchases that are large relative to its liquidity pool.

Solidity interfaces

HildaFactory interface:

```
interface IHildaFactory {
	event PairCreated(address indexed token0, address indexed token1, address pair, uint);

	function feeTo() external view returns (address);
	function feeToSetter() external view returns (address);

	function getPair(address tokenA, address tokenB) external view returns (address pair);
	function allPairs(uint) external view returns (address pair);
	function allPairsLength() external view returns (uint);

	function createPair(address tokenA, address tokenB) external returns (address pair);

	function setFeeTo(address) external;
	function setFeeToSetter(address) external;
}
```

HildaRouter Interface:

```
interface IHildaRouter {

	function factory() external pure returns (address);

	function addLiquidity(
	address tokenA,
	address tokenB,
	uint amountADesired,
	uint amountBDesired,
	uint amountAMin,
	uint amountBMin,
	address to,
	uint deadline
```

```
    ) external returns (uint amountA, uint amountB, uint liquidity);

    function removeLiquidity(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
    ) external returns (uint amountA, uint amountB);

    function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);

    function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
    ) external returns (uint[] memory amounts);

    function swapTokensForExactTokens(
    uint amountOut,
    uint amountInMax,
    address[] calldata path,
    address to,
    uint deadline
    ) external returns (uint[] memory amounts);
```

```
      function swapExactTokensForTokensSupportingFeeOnTransferTokens(
      uint amountIn,
      uint amountOutMin,
      address[] calldata path,
      address to,
      uint deadline
      ) external;

      function quote(uint amountA, uint reserveA, uint reserveB) external
pure returns (uint amountB);

      function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
external pure returns (uint amountOut);

      function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
external pure returns (uint amountIn);

      function getAmountsOut(uint amountIn, address[] calldata path)
external view returns (uint[] memory amounts);

      function getAmountsIn(uint amountOu

}
```

HildaPair Interface

```
interface IHildaPair {
      event Approval(address indexed owner, address indexed spender, uint
value);
      event Transfer(address indexed from, address indexed to, uint value);

      function name() external pure returns (string memory);
      function symbol() external pure returns (string memory);
      function decimals() external pure returns (uint8);
      function totalSupply() external view returns (uint);
      function balanceOf(address owner) external view returns (uint);
      function allowance(address owner, address spender) external view
returns (uint);
```

```
    function approve(address spender, uint value) external returns
(bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external
returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint
deadline, uint8 v, bytes32 r, bytes32 s) external;

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1,
address indexed to);
    event Swap(
    address indexed sender,
    uint amount0In,
    uint amount1In,
    uint amount0Out,
    uint amount1Out,
    address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0,
uint112 reserve1, uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint
```

```
amount1);
      function swap(uint amount0Out, uint amount1Out, address to, bytes
calldata data) external;
      function skim(address to) external;
      function sync() external;

      function initialize(address, address) external;
}
```

Our View On Existing DEX Problems To Be Aware

Frontrunning (flash boys 2.0 and Mooniswap vs Uniswap value proposition). If it is not possible in Free TON (as it is), should be clearly stated why.

We covered front-running in detail in the corresponding section above offering a solution to be implemented on Free TON architecture.

Shortcomings of standard curves and its inflexible nature in liquidity based approach. These result in the issue - dilution of liquidity into several AMMs (general and specific ones with a more specific curve like Uniswap vs. Curve) which leads to non-optimal price execution.

Metapools are mainly used by Curve since they specialize exclusively on hard-pegged tokens like USDC and USDT and soft-pegged like DAI, since the prices in these pools vary in small amounts. Metapools allow for one token to seemingly trade with another underlying base pool. This means we could create for example the following pool: [GUSD, [3Pool]].

In this example users could seamlessly trade GUSD between the three coins in the 3Pool (DAI/USDC/USDT). This is helpful in multiple ways:

- Prevents diluting existing pools
- Allows to list less liquid assets
- More volume and more trading fees for the DAO

The Metapool in question would take GUSD and 3Pool LP tokens. This means that liquidity providers of the 3Pool **who do not provide liquidity in the GUSD Metapool are shielded from systemic risks from the Metapool.** That's one of the main advantages of metapools and down the road map we are going to implement them, though at the moment they are not of the first priority.

As for the **dilution of liquidity**, we don't see this as a large issue, since as we already mentioned in the paper, exchange aggregators like 1inch and Dex.ag pull the liquidity from all available sources and present it for the end user as one trade. Since most exchanges use liquidity pools (not an order book), you don't have to find a counterpart to match every order from every pool. You just quote the price in the pools and pull the liquidity.

We believe that market forces will do their job by gradually lowering the fees in pools and finally liquidity providers will migrate to those pools with the most liquidity and volume.

One-sided liquidity. Impairment loss problems. The market risk of two assets in the pool is widely discussed. These problems should be somehow covered or at least mentioned for further research.

Speaking briefly, we compensate for all the shortcomings of providing double-sided liquidity with deflationary Hilda Token rewards. This token has great tokenomics and as experience shows not only holds its price level but is also steadily rising.

But, we have plans to use a single-sided liquidity approach, described in the **Mature Phase** economics section, as the TON infrastructure develops and there will appear

decentralized futures trading. Then we would be able to hegde any position with long/short futures and create complex derivative trading instruments.

It is also worth to mention that we could also implement a Bancor-like approach and let users provide a single whitelisted token (**Single-side Liquidity**) in the pool and will compensate the loss with Hilda Token. But to do this we need to carefully weigh all pros and cons of such an endeavor. We believe Bancor hasn't done the whole mechanism properly. If we were to implement their approach, we would have introduced 100% insurance **instantly** for any liquidity provider instead of after a three month period, and only punish in case a provider withdraws from the pool earlier, by slashing some amount of cover, depending on the time spent in the pool.

References

1. Contest Proposal: FreeTon DEX Architecture & Design Proposal https://forum.freeton.org/t/contest-proposal-freeton-dex-architecture-design-proposal/3067