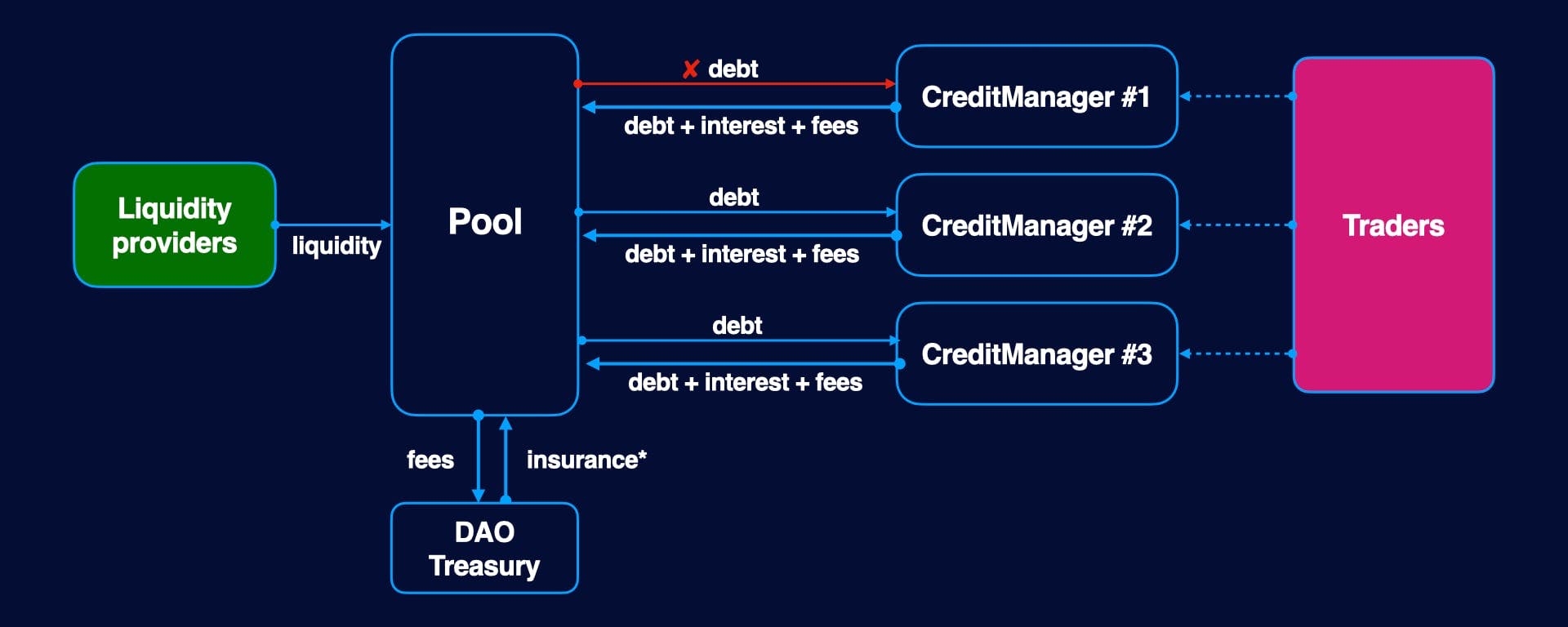
Pools

Overview

**Pools**

Capital is required for traders to get leverage on the platform. For this, there are Liquidity Pools(LPs): anyone can become a liquidity provider by depositing funds in the Liquidity Pool.



The profitability of LPs depends on the pool utilization ratio - the higher utilization, the higher interest rate. Each pool has an underlying asset and risk parameters such as: allowed trading tokens, allowed DEXes, stable coin pool, and others.

In the current version, we implement linear extrapolation for interest rate calculation as Aave did, in v2 we are going to use a specially designed curve - see the [note(opens in a new tab)](https://colab.research.google.com/drive/1bjBWHNGHiSDd27_WsINQLXa3ImhTrt-W).

**Diesel (LP) tokens**

Each pool has its own diesel (LP) tokens. Each time, when a liquidity provider adds money to the pool, he gets diesel tokens back (like c-tokens in Compound).



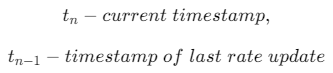
Liquidity providers get profits from holding diesel tokens because they grow with expected interest. LP can keep diesel tokens on their wallets and then withdraw the deposit + interest or can use them as collateral in lending protocols or even sell them on the secondary market. Diesel tokens are 100% liquid yield-generating assets.

**Basic parameters**

* EL(t) - expected liquidity
* B(t) - total borrowed
* r(t) - borrow rate
* d(t) - diesel rate
* CI(t) - cumulative index (it shows value of money at moment t)

**Periods and timestamp**

All functions are piecewise linear functions. Each change in available liquidity or borrowed amount [updates rate parameters](https://dev.gearbox.fi/). In follow formulas we use the convention:



**Available liquidity**

The amount of money available in pool.

**EL(t) - Expected Liquidity**

The amount of money should be in the pool if all users close their Credit Accounts and return debt. If no action happens during *tn*−1​ and *tn*​, then the equation of *EL* should be

*EL*(*tn*​)=*EL*(*tn*−1​)+*B*(*tn*−1​)∗*r*(*tn*−1​)∗(*tn*​−*tn*−1​)

Beside, [Add Liquidity](https://dev.gearbox.fi/) and [Remove Liquidity](https://dev.gearbox.fi/) will have a new formula of *EL*.

**B(t) - Total borrowed**

Represents total borrowed amount without accrued interest rate:

*B*(*t*)=∑*bi*​

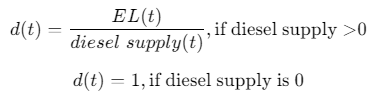
**r(t) - Borrow APY**

Represents current borrow APY. Depends on pool utilisation parameter and computed independently using [Interest rate model](https://dev.gearbox.fi/).

**d(t) Diesel rate**

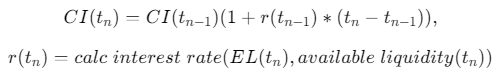
Liquidity providers get profits from holding diesel tokens because they grow with expected interest. LP can keep diesel tokens on their wallets and then withdraw the deposit + interest.

Diesel Rate is the price of Diesel token (LP token).



**Cumulative Index**

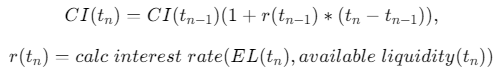
Cumulative Index is aggregated variable that shows value of borrowing money.



**Rate parameters update**

Updates borrow rate & cumulative index. Called each time when borrowed amount or available liquidity is changed:

* Add liquidity
* Remove liquidity
* Credit account manager lends money
* Credit account manager repays debt



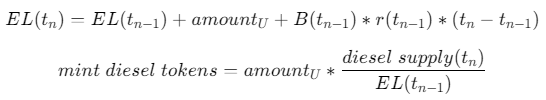
Pools

Liquidity

**Liquidity**

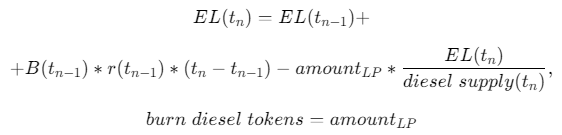
**Liquidity operations**

**Add liquidity**



where amount\_U - is amount of added underlying liquidity. Then called Pool Update().

**Remove liquidity**



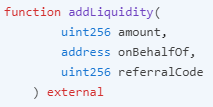
where amount\_LP - amount of removed LP tokens.Then call Pool Update().

**Adding / removing liquidity programatically**

**addLiquidity**

Adds liquidity to pool and send diesel (LP) tokens back to LP:

* Transfers underlying asset to pool
* Mints diesel (LP) token with current diesel rate
* Updates expected liquidity
* Updates borrow rate



| **Parameter** | **Description** |
| --- | --- |
| amount | Amount of tokens to be transfer |
| onBehalfOf | The address that will receive the diesel tokens, same as msg.sender if the user wants to receive them on his own wallet, or a different address if the beneficiary of diesel tokens is a different wallet |
| referralCode | Code used to register the integrator originating the operation, for potential rewards. 0 if the action is executed directly by the user, without any middle-man |

**removeLiquidity**



| **Parameter** | **Description** |
| --- | --- |
| amount | Amount of tokens to be transfer |
| to | Address to transfer liquidity |

It burns user diesel (LP) tokens and returns underlying tokens:

* Transfers to LP *underlying account* = amount \* diesel rate
* Burns diesel tokens
* Decreases underlying amount from total\_liquidity
* Updates borrow rate

Pools

Interest rate model

**LinearInterestRateModel**

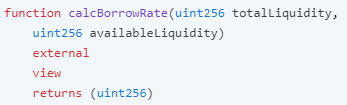
Linear interest rate model, similar to which Aave uses: [https://docs.aave.com/risk/asset-risk/methodology(opens in a new tab)](https://docs.aave.com/risk/asset-risk/methodology). In the next version, we are going to use a specially designed curve - see the [paper(opens in a new tab)](https://colab.research.google.com/drive/1UciFX7BQ62mtme0r84FO95FfA8C_zQKI).

LinearInterestRateModel implements IInterestRateModel

**Getters**

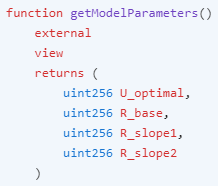
**calcBorrowRate**

Calculates borrow rate



**getModelParameters**

Gets interest rate model parameters



Pools

Fee collection

**Fees collection**

Protocol collects following fees:

* Interest rate fee, when user close credit account, or it's liquidated
* Liquidation fee, if account was liquidated
* Withdraw fee, when user removes liquidity

The fees remain in the pool, instead, the DAO treasury receives the pool's LP tokens. The protocol has a feature to leave part of the fees in the pool, thereby increasing the price of diesel tokens, that is, increasing the APY of the pool.

Pools

Insurance

**Insurance and rebalancing**

**Closing account**

When user closes account or his account is liquidated:

PnL - is result of repaying:

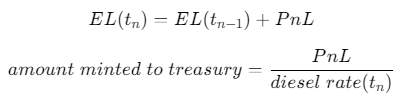
*PnL*=*totalfunds*−*amountB*​−*interest*

Updating total borrowed amount:

*B*(*tn*​)=*B*(*tn*−1​)−*amountB*​

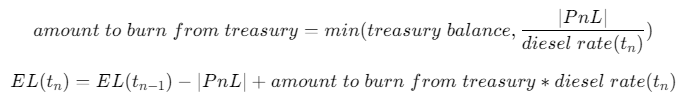
**PnL>=0:**

This case means that return value > borrowed amount + expected interest accrued. Interest accrued is already included in expected liquidity. At this point, the protocol keeps all funds in the pool and mint diesel tokens to treasury fund.



**PnL<0:**

This case means, that returned value < borrowed amount + expected interest. At this cases, pool uses treasury fund as insurance and burn tokens to keep *diesel rate* on the same level.



**Insurance example**

In some rare cases, the remaining funds after paying liquidation premium could be less than borrowed amount + interest rate + fee. In this case, the protocol uses treasury to compensate for the shortage by burning diesel tokens to keep the diesel rate as it should be.

Let's consider an example:

1) LP adds 1000 DAI to the pool. Diesel rate =1, he got 1000 dDAI (diesel DAI tokens). Total liquidity = 2000, and diesel supply = 2000.

2) Trader borrowed 1000 DAI and used them for some period. Let's assume that interest accrued is 100DAI. So, at time of return: total liquidity = 2100, diesel rate= 2100 / 2000 = 1.05

3) LP earns on diesel rate, so their interest rate income is already accounted in 1.05 rate.

4) Trader's account was liquidated, and the remaining funds are 1000 DAI only. So, if there is no insurance fund was there, the diesel rate should return to 1 which means that LP earns nothing. However, in this case, protocol burns treasure's tokens to keep the diesel rate on the same level:



5) Let's check that burning 95.23 will keep diesel rate as it was before this accident:



So, in this case, the treasury was used to cover some losses and behave like an insurance fund.

Pools

Economy

# Pools

PoolSevice contract provides API for managing liquidity and computing interest rates. PoolService lends money to connected credit account managers. Let's take a look at Pool Economy first. The [Pool operations](https://dev.gearbox.fi/pools/economy#pool-operations) are the main functions of PoolService contract.

## Pool Economy

Capital is required for traders to get leverage on the platform. For this, there are Liquidity Pools(LPs): anyone can become a liquidity provider by depositing funds in the Liquidity Pool.

The profitability of LPs depends on the pool utilization ratio - the higher utilization, the higher interest rate. Each pool has an underlying asset and risk parameters such as: allowed trading tokens, allowed DEXes, stable coin pool, and others.

In the current version, we implement linear extrapolation for interest rate calculation as Aave did, in v2 we are going to use a specially designed curve - see the [note(opens in a new tab)](https://colab.research.google.com/drive/1bjBWHNGHiSDd27_WsINQLXa3ImhTrt-W).

### Diesel tokens

Each pool has its own diesel (LP) tokens. Each time, when a liquidity provider adds money to the pool, he gets diesel tokens back (like c-tokens in Compound).



Liquidity providers get profits from holding diesel tokens because they grow with expected interest. LP can keep diesel tokens on their wallets and then withdraw the deposit + interest or can use them as collateral in lending protocols or even sell them on the secondary market. Diesel tokens are 100% liquid yield-generating assets.

### Basic parameters

* EL(t) - [expected liquidity](https://dev.gearbox.fi/pools/economy#expected-liquidity)
* B(t) - total borrowed
* r(t) - borrow rate
* d(t) - diesel rate
* CI(t) - cumulative index (it shows value of money at moment t)

#### ****Periods and timestamp****

All functions are piecewise linear functions. Each change in available liquidity or borrowed amount [updates rate parameters](https://dev.gearbox.fi/pools/economy#rate-parameters-update). In follow formulas we use the convention:

��−�������  ���������,*tn*​−*currenttimestamp*,

��−1−���������  ��  ����  ����  ������*tn*−1​−*timestampoflastrateupdate*

#### Available liquidity

The amount of money available in pool.

#### EL(t) - Expected Liquidity

The amount of money should be in the pool if all users close their Credit Accounts and return debt. If no action happens during ��−1*tn*−1​ and ��*tn*​, then the equation of ��*EL* should be

��(��)=��(��−1)+�(��−1)∗�(��−1)∗(��−��−1)*EL*(*tn*​)=*EL*(*tn*−1​)+*B*(*tn*−1​)∗*r*(*tn*−1​)∗(*tn*​−*tn*−1​)

Beside, [Add Liquidity](https://dev.gearbox.fi/pools/economy#add-liquidity) and [Remove Liquidity](https://dev.gearbox.fi/pools/economy#remove-liquidity) will have a new formula of ��*EL*.

#### B(t) - Total borrowed

Represents total borrowed amount without accrued interest rate:

�(�)=∑��*B*(*t*)=∑*bi*​

#### r(t) - Borrow APY

Represents current borrow APY. Depends on pool utilisation parameter and computed independently using [Interest rate model](https://dev.gearbox.fi/pools/economy#linear-interest-rate-model).

#### d(t) Diesel rate

Liquidity providers get profits from holding diesel tokens because they grow with expected interest. LP can keep diesel tokens on their wallets and then withdraw the deposit + interest.

Diesel Rate is the price of Diesel token (LP token).

�(�)=��(�)������  ������(�),if diesel supply >0*d*(*t*)=*dieselsupply*(*t*)*EL*(*t*)​,if diesel supply >0

�(�)=1,if diesel supply is 0*d*(*t*)=1,if diesel supply is 0

#### Cumulative Index

Cumulative Index is aggregated variable that shows value of borrowing money.

��(��)=��(��−1)(1+�(��−1)∗(��−��−1)),*CI*(*tn*​)=*CI*(*tn*−1​)(1+*r*(*tn*−1​)∗(*tn*​−*tn*−1​)),

�(��)=����  ��������  ����(��(��),���������  ���������(��))*r*(*tn*​)=*calcinterestrate*(*EL*(*tn*​),*availableliquidity*(*tn*​))

#### Rate parameters update

Updates borrow rate & cumulative index. Called each time when borrowed amount or available liquidity is changed:

* Add liquidity
* Remove liquidity
* Credit account manager lends money
* Credit account manager repays debt

��(��)=��(��−1)(1+�(��−1)∗(��−��−1)),*CI*(*tn*​)=*CI*(*tn*−1​)(1+*r*(*tn*−1​)∗(*tn*​−*tn*−1​)),

�(��)=����  ��������  ����(��(��),���������  ���������(��))*r*(*tn*​)=*calcinterestrate*(*EL*(*tn*​),*availableliquidity*(*tn*​))

### Pool operations

#### Add liquidity

��(��)=��(��−1)+�������+�(��−1)∗�(��−1)∗(��−��−1)*EL*(*tn*​)=*EL*(*tn*−1​)+*amountU*​+*B*(*tn*−1​)∗*r*(*tn*−1​)∗(*tn*​−*tn*−1​)

����  ������  ������=�������∗������  ������(��)��(��−1)*mintdieseltokens*=*amountU*​∗*EL*(*tn*−1​)*dieselsupply*(*tn*​)​

where amount\_U - is amount of added underlying liquidity. Then called Pool Update().

#### Remove liquidity

��(��)=��(��−1)+*EL*(*tn*​)=*EL*(*tn*−1​)+

+�(��−1)∗�(��−1)∗(��−��−1)−��������∗��(��)������  ������(��),+*B*(*tn*−1​)∗*r*(*tn*−1​)∗(*tn*​−*tn*−1​)−*amountLP*​∗*dieselsupply*(*tn*​)*EL*(*tn*​)​,

����  ������  ������=��������*burndieseltokens*=*amountLP*​

where amount\_LP - amount of removed LP tokens.Then call Pool Update().

#### Lend Credit Account

�(��)=�(��−1)+�������*B*(*tn*​)=*B*(*tn*−1​)+*amountB*​

where amount\_B - borrowed amount. Then Pool Update().

#### Repay Credit Account

PnL - is result of VA repaying:

���=�����  �����−�������−��������*PnL*=*totalfunds*−*amountB*​−*interest*

Updating total borrowed amount:

�(��)=�(��−1)−�������*B*(*tn*​)=*B*(*tn*−1​)−*amountB*​

**PnL>=0:**

This case means that return value > borrowed amount + expected interest accrued. Interest accrued is already included in expected liquidity. At this point, the protocol keeps all funds in the pool and mint diesel tokens to treasury fund.

��(��)=��(��−1)+���*EL*(*tn*​)=*EL*(*tn*−1​)+*PnL*

������  ������  ��  ��������=���������  ����(��)*amountmintedtotreasury*=*dieselrate*(*tn*​)*PnL*​

**PnL<0:**

This case means, that returned value < borrowed amount + expected interest rate. At this cases, pool uses treasury fund as insurance and burn tokens to keep diesel rate on the same level.

������  ��  ����  ����  ��������=���(��������  �������,∣���∣������  ����(��))*amounttoburnfromtreasury*=*min*(*treasurybalance*,*dieselrate*(*tn*​)∣*PnL*∣​)

��(��)=��(��−1)−∣���∣+������  ��  ����  ����  ��������∗������  ����(��)*EL*(*tn*​)=*EL*(*tn*−1​)−∣*PnL*∣+*amounttoburnfromtreasury*∗*dieselrate*(*tn*​)

### Fees distribution

Each time the trader/farmer closes the credit account, he pays back to the pool:

* Borrowed amount
* Interest accrued
* Fees

The fees remain in the pool, instead, the treasury receives the pool's LP tokens. The protocol has a feature to leave part of the fees in the pool, thereby increasing the price of diesel tokens, that is, increasing the APY of the pool.

### Insurance and rebalancing

In some rare cases, the remaining funds after paying liquidation premium could be less than borrowed amount + interest rate + fee. In this case, the protocol uses treasury to compensate for the shortage by burning diesel tokens to keep the diesel rate as it should be.

Let's consider an example:

1) LP adds 1000 DAI to the pool. Diesel rate =1, he got 1000 dDAI (diesel DAI tokens). Total liquidity = 2000, and diesel supply = 2000.

2) Trader borrowed 1000 DAI and used them for some period. Let's assume that interest accrued is 100DAI. So, at time of return: total liquidity = 2100, diesel rate= 2100 / 2000 = 1.05

3) LP earns on diesel rate, so their interest rate income is already accounted in 1.05 rate.

4) Trader's account was liquidated, and the remaining funds are 1000 DAI only. So, if there is no insurance fund was there, the diesel rate should return to 1 which means that LP earns nothing. However, in this case, protocol burns treasure's tokens to keep the diesel rate on the same level:

�����  ��  ����=2000  ∗(1.05−1)/1.05≈95.23*tokentoburn*=2000∗(1.05−1)/1.05≈95.23

5) Let's check that burning 95.23 will keep diesel rate as it was before this accident:

������  ����=20002000−95.23≈1.05*dieselrate*=2000−95.232000​≈1.05

So, in this case, the treasury was used to cover some losses and behave like an insurance fund.

### Linear Interest Rate Model

## LinearInterestRateModel

Linear interest rate model, similar to which Aave uses: [https://docs.aave.com/risk/asset-risk/methodology(opens in a new tab)](https://docs.aave.com/risk/asset-risk/methodology). In the next version, we are going to use a specially designed curve - see the [note(opens in a new tab)](https://colab.research.google.com/drive/1bjBWHNGHiSDd27_WsINQLXa3ImhTrt-W).

LinearInterestRateModel implements [IInterestRateModel.sol(opens in a new tab)](https://github.com/Gearbox-protocol/gearbox-v2/blob/master/contracts/interfaces/IInterestRateModel.sol" \t "_blank).

## Configuration

For the current version, pools use Immutable configuration, which means that all parameters should be set before contract deployment.