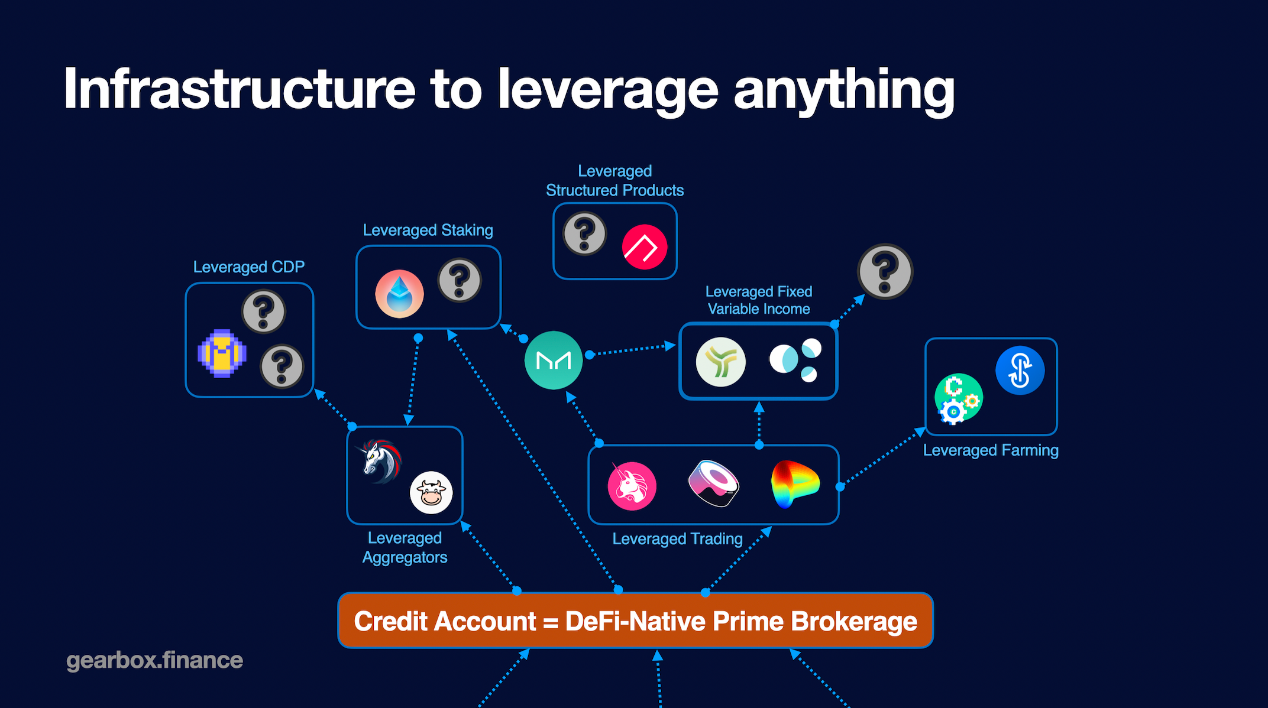
**Introduction**

Gearbox is a generalized leverage protocol: it allows anyone to take leverage in a DeFi-native way and then use it across various DeFi protocols. You take leverage with Gearbox and then use it on other protocols you already love. For example, you can leverage trade on Uniswap, leverage farm on Yearn or Curve and Convex, make complex delta-neutral strategies involving options and derivatives, get Leverage-as-a-Service for your structured product doing complex positions, etc.

The protocol has two sides to it: passive liquidity providers who earn higher APY by providing liquidity; - and active traders, farmers, or even other protocols who can borrow those assets to trade or farm with x4+ leverage.

That is possible thanks to Credit Accounts…

**Credit Accounts**



A Credit Account is an isolated smart contract which contains both the user funds and the borrowed funds. This is where your leverage is. After you open an account, all the operations go through this account and the assets stay on it as well. You can see a Credit Account as your automated DeFi wallet where you not only keep positions, but can also potentially program it the way you want.

Funds on Credit Accounts are used as collateral for debt, and users can operate these funds by sending financial orders to their Credit Accounts. That could be: margin trading on Uniswap or Sushiswap; leverage farming on Yearn; arbitraging pegged assets on Curve, and more!

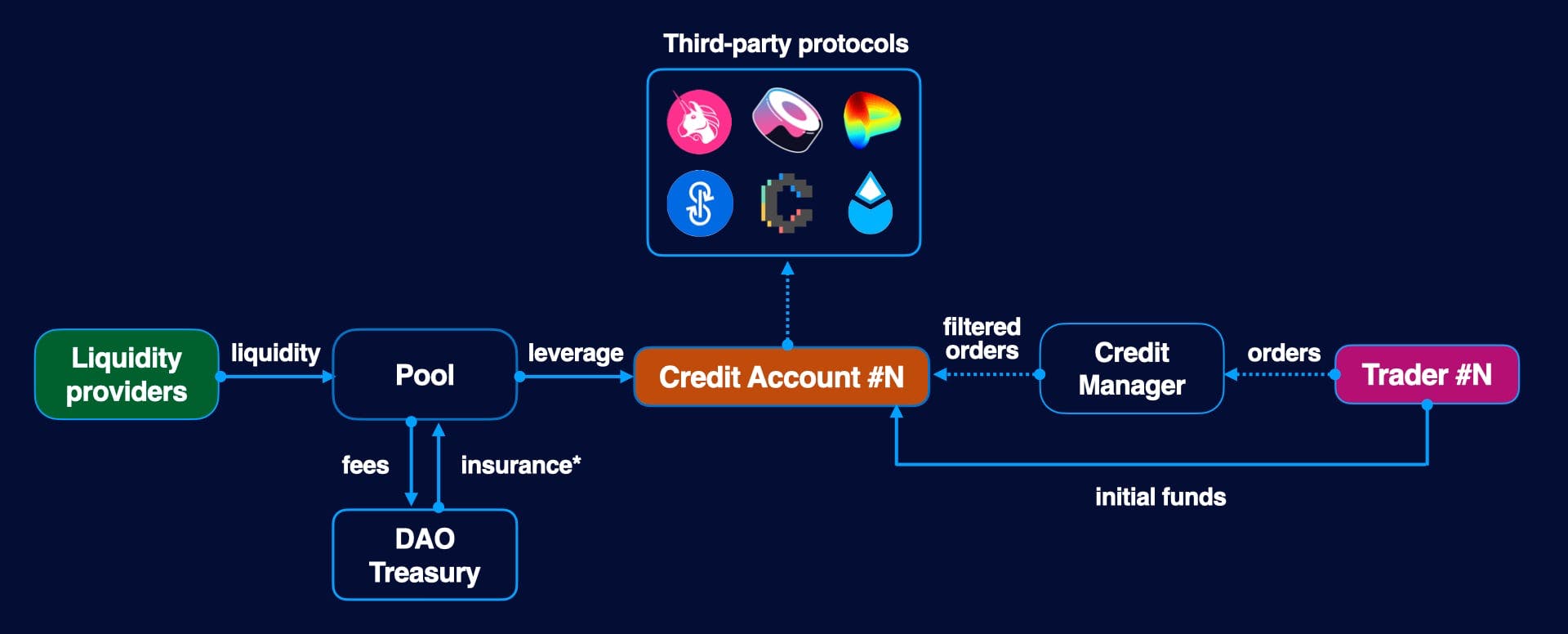
**Cool Features**

* **Composable**. Gearbox does not have its own order book or trading environment. The leverage you get is used across multiple DeFi protocols and assets, fully composable! For example, a yield aggregator can be on the liquidity provider side of Gearbox Protocol, as well as be an avenue for Gearbox users to deploy their leverage into.
* **0% Funding Rates**. The leverage offered is not based on synthetic positions but instead is executed with real assets on third-party protocols. Because Gearbox does not create its own trading pairs, there is no short-long ratio that needs to be maintained with funding rates.
* **Leverage as a Service**. Other protocols can offer leverage to their users with the help of Gearbox Protocol, without modifying anything in their own architecture. As such, they also get exposure to the user base of Gearbox.
* **Permissionless Strategies**. Positions and trades within Credit Accounts can be extended to include complex strategies, for example, making a short position farm in Yearn; or having LP tokens as collateral for more composable actions.
* **Low overhead on gas**. Due to how data and operations are processed across isolated smart contracts, gas usage overhead is reduced to an insignificant overhead.

[Gearbox dApp](https://gearbox.fi/)

# Architecture

## Overview



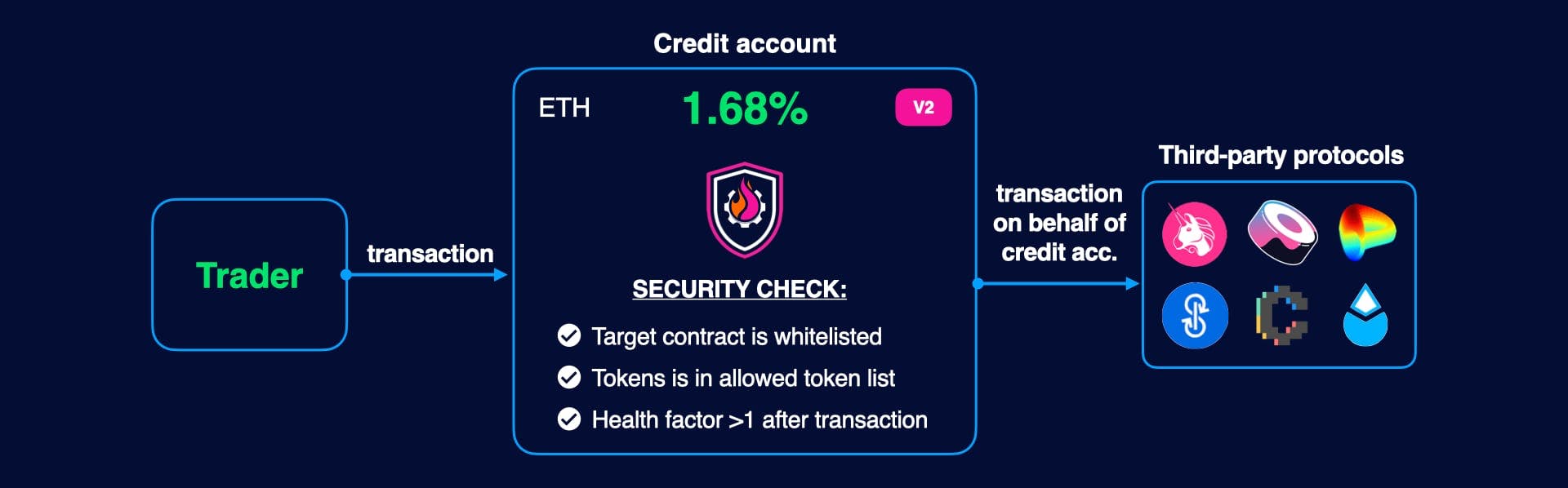
Gearbox is a two-sided protocol, with passive liquidity providers on one side and active traders and position managers on the other.

Liquidity providers can commit their funds into the liquidity pool to be borrowed and receive yield from interest. This works similarly in principle to other major lending protocols, such as Aave or Compound.

Active traders seek to get leverage and use it across different DeFi protocols. They open a Credit Account, providing initial funds, and additional funds are borrowed from the pool. The trader can then use their new Credit Account with borrowed funds to interact with connected protocols.

## Credit account

A Credit Account is new DeFi primitive - an isolated smart contract that allows to execute financial orders (interactions with third-party protocols), but doesn't provide direct access to funds contained within it.



Credit Accounts execute transactions on behalf of themselves, however, each executed transaction has to pass several checks:

* The target contract must be in a whitelist
* Incoming tokens must be in a whitelist
* The Credit Account's health factor must be more than 1 after the transaction (i.e., it must be able to repay its debt with interest)

More technically, Credit Accounts themselves are simple contracts that route calls, and all aforementioned policies are enforced by the Credit Manager.

[Credit Accounts](https://dev.gearbox.fi/credit/intro.md)

## Core contracts

Core contracts provide functionality necessary to interact with the protocol, including contract discovery, price feeds, access control, etc.



| **Contract** | **Responsibility** |
| --- | --- |
| AddressProvider | Keeps addresses of core contracts |
| AccountFactory | Supplies reusable credit accounts |
| ACL | Manages access control |
| ContractsRegister | Pools & CreditManagers contrascts register |
| DataCompressor | Prepares data for offchain services |
| PriceOracle | Provides price data for CreditManager |
| WETHGateway | Converts ETH to WETH and vice versa |

[Contracts Discovery](https://dev.gearbox.fi/architecture/discovery)

Architecture

Contracts discovery

# Contracts discovery

## Core contracts

Each interaction with Gearbox starts from retrieving the addresses of important contracts. AddressProvider stores the addresses of all core contracts.

| **function** | **return value** |
| --- | --- |
| getACL() | Address of ACL contract |
| getContractsRegister() | Address of ContractsRegister |
| getAccountFactory() | Address of AccountFactory |
| getDataCompressor() | Address of DataCompressor |
| getGearToken() | Address of GEAR token |
| getWethToken() | Address of WETH token |
| getWETHGateway() | Address of WETH Gateway |
| getPriceOracle() | Address of PriceOracle |
| getTreasuryContract() | Address of DAO Treasury Multisig |

:::note AddressProvider stores the most up-to-date addresses. It is recommended to avoid caching Gearbox-related addresses for a long time and refresh them periodically through AddressProvider, since they can be changed by the governance. :::

[List of all deployed contracts on mainnet](https://dev.gearbox.fi/docs/documentation/deployments/deployed-contracts)

## Getting list of pools & credit managers

ContractsRegister keeps a list of all active pools and Credit Managers:



## Versioning

Each contract in the protocol has a function version which returns the current version as a uint256 value. Contract ABIs can change between versions, so it is recommended to get and verify the value before interacting with a particular contract.

Code snippet from CreditManager:



Architecture

Account factory

# AccountFactory

## Reusable credit accounts

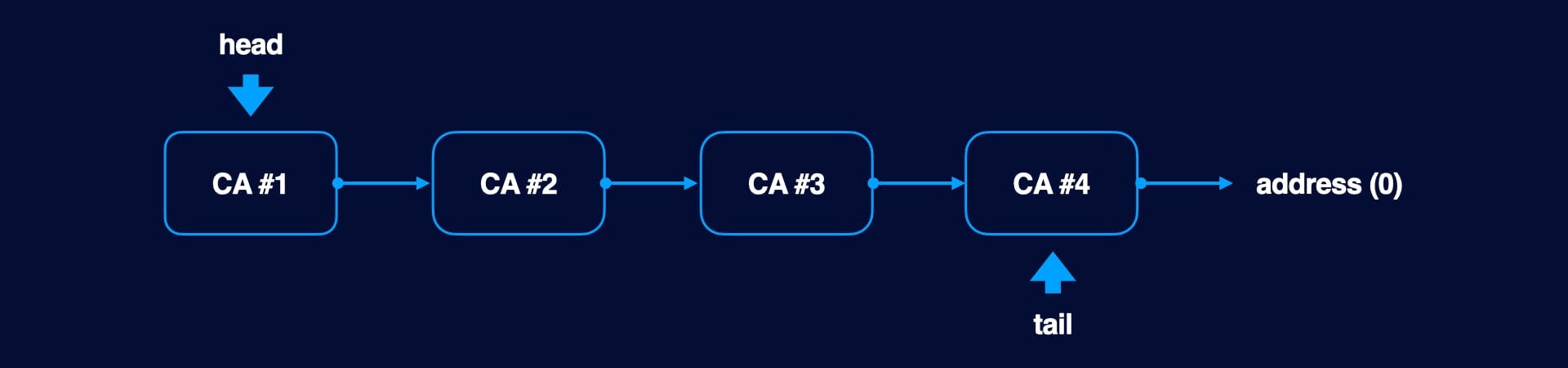
Each Credit account is implemented as an isolated smart contract. This helps to reduce gas consumption, as balances, account health and other data can be tracked naturally on a per-address basis, instead of being stored in a contract.



Reusability means that Credit Account contracts are deployed once, and then "rented" by a Credit Manager when a user opens an account. Once the account is closed or liquidated, the contract is returned. Users do not need to pay to deploy a new CreditAccount each time, which saves gas.

## Account Factory

AccountFactory is responsible for supplying Credit Accounts to Credit Managers. The pre-deployed Credit Accounts are organized into a linked list, and the current head is given each time a Credit Manager requests one. Returned accounts are appended to the list tail:



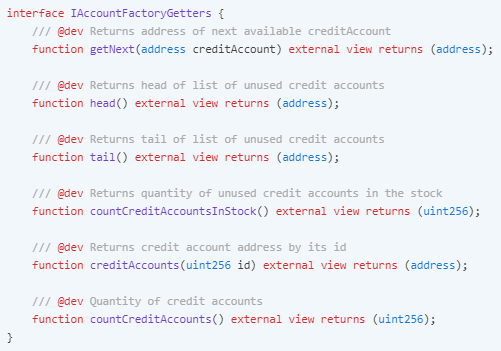
If the Account Factory has no pre-deployed contracts available and a user tries to open a new Credit Account, a new one is cloned using [EIP-1167(opens in a new tab)](https://eips.ethereum.org/EIPS/eip-1167).

## Advantages

* **Gas efficiency**  
  This solution is more gas-efficient compared to creating a new Credit Account, since deployment costs do not need to be paid as long as there are available pre-deployed contracts in the AccountFactory.
* **Resistance to insolvency contagion** Funds being kept on isolated accounts ensures that there is no system-wide contagion when a particular account, Credit Manager or integrated protocol is compromised. Collateralization and safety of Credit Accounts unrelated to the event is not affected.
* **Balance transparency on Etherscan**  
  Transactions for each account can be directly tracked on Etherscan - a user only needs to know the blocks in which the account was opened and closed in order to index transactions made by a particular borrower.
* **Ethereum network ecology**  
  The pattern generates significantly less data in comparison with deploying new accounts every time; and saves a considerable amount of gas both at account opening and during usage. As a result, less of an impact is made on Ethereum infrastructure.

## Getters

Despite AccountFactory being primarily for internal use, developers can retrieve information on the current account stock if they require it:



Architecture

Role model

# Role model

## Roles

ACL contract keeps global roles for whole Gearbox Protocol. There are 3 basic roles in the system:

| **Role** | **Responsibility** |
| --- | --- |
| Configurator | Role which has the most powerful roles for configuring system parameters |
| PausableAdmin | Role can pause contracts |
| UnPausableAdmin | Role can unpause contracts |

## Configurator role

Configurator is owner of ACL contract. This code snippet shows how to get the current adddress:

address configurator = ACL(addressProvider.getACL()).owner();

In current deployment, Configurator is managed by Technical Multisig, in the future, Gearbox will use Compound Bravo Governance.

### How to test your contracts

In many cases, developer needs to change system parameters:

Forge (solidity)

