Balancer Boosted Pools

x

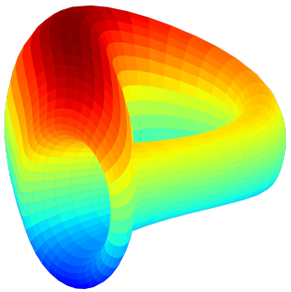
Compound Finance

Presentation by

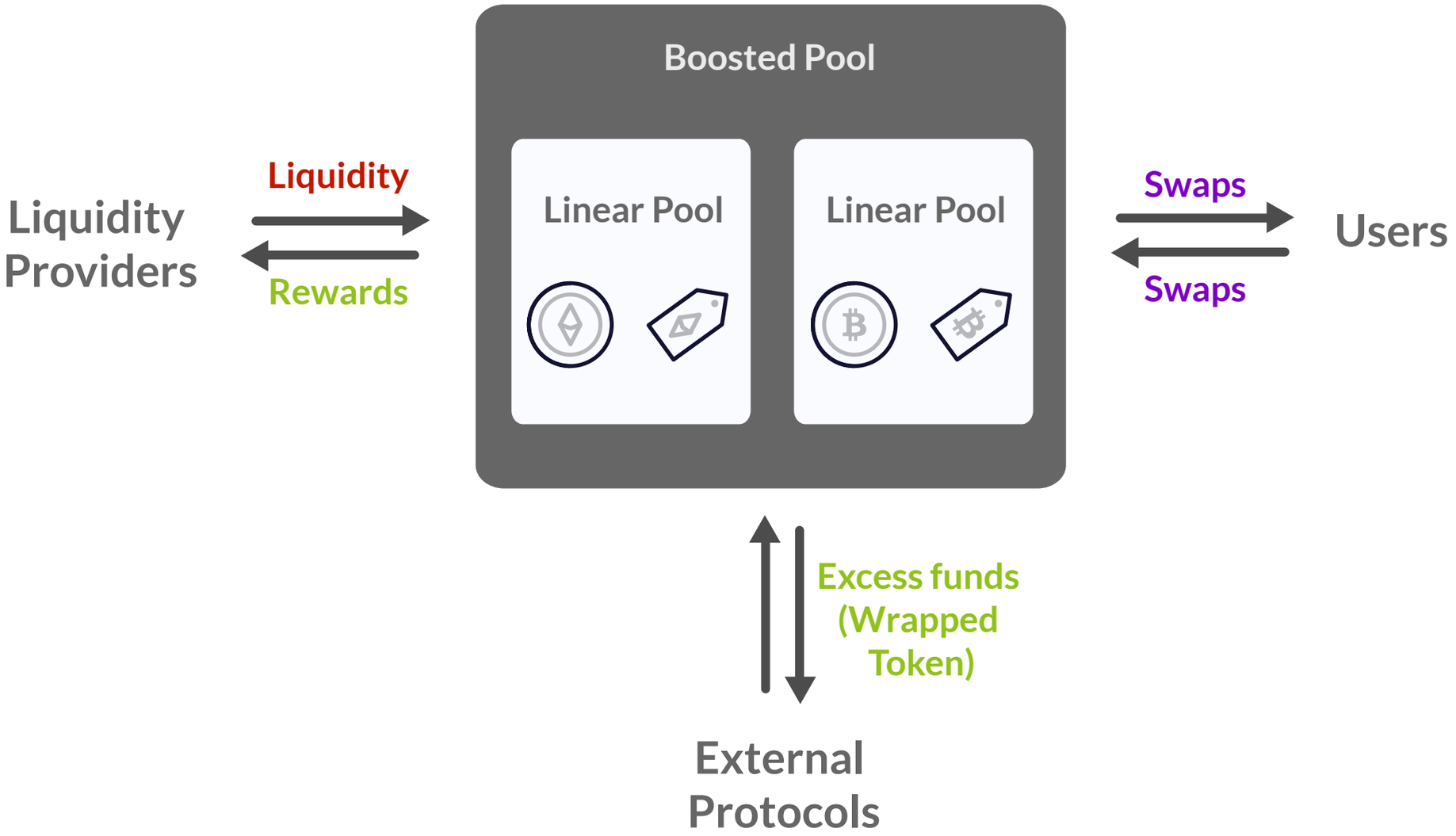
Bailey Spraggins

Automated Market Makers (AMMs)



Balancer Boosted Pools



Boosted Pools

Key Info

- Can be any type of pool
- Balancer Vault holds all tokens
- Vault is the entry point for swaps, joins, and exits
- Must contain at least one linear pool
- Linear Pool Contracts do the heavy lifting

Advantages

- Increased returns for LPs
- Better Exchange Rates
- Increased liquidity throughout the market
- Price Stability

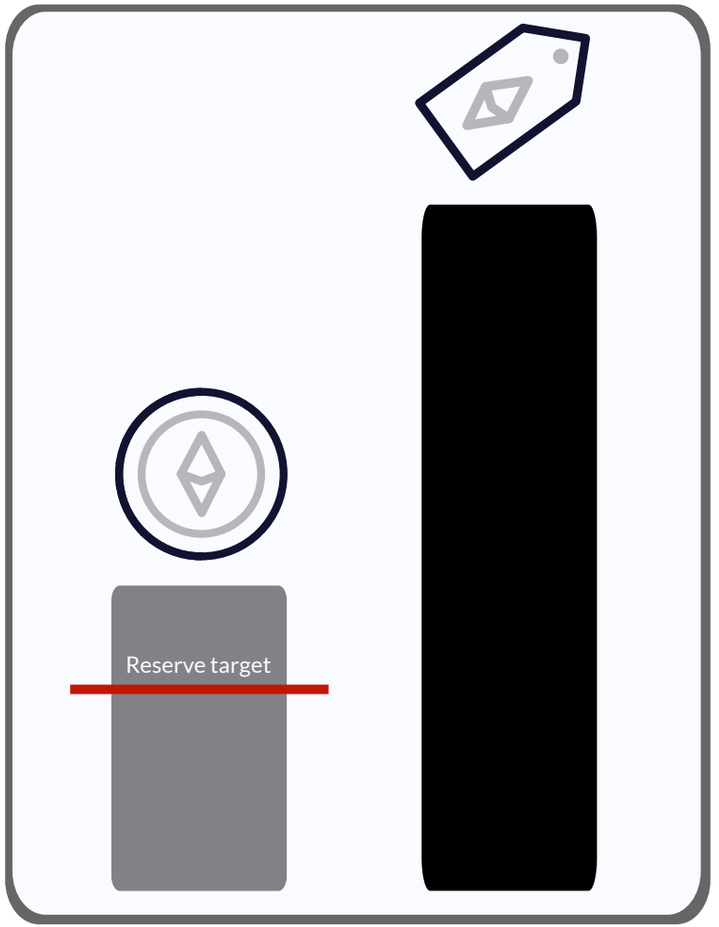
Balancer Linear Pools

Facilitates the exchange between the base asset and its corresponding derivative using linear math

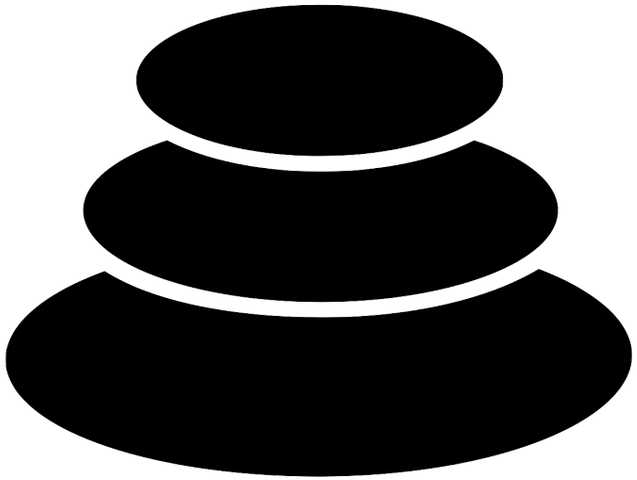
Positive/Negative Fee system to incentivize users to maintain optimal token ratios

$$\textbf{Amount}_A = \textit{rate} \times \textbf{Amount}_B$$

$$\textbf{Amount}_B = \frac{1}{\textit{rate}} \times \textbf{Amount}_A$$



Project Overview



X

Project Scope: Create a Balancer Boosted Pool that incorporates Compound Finance's Defi protocol

Compound Finance

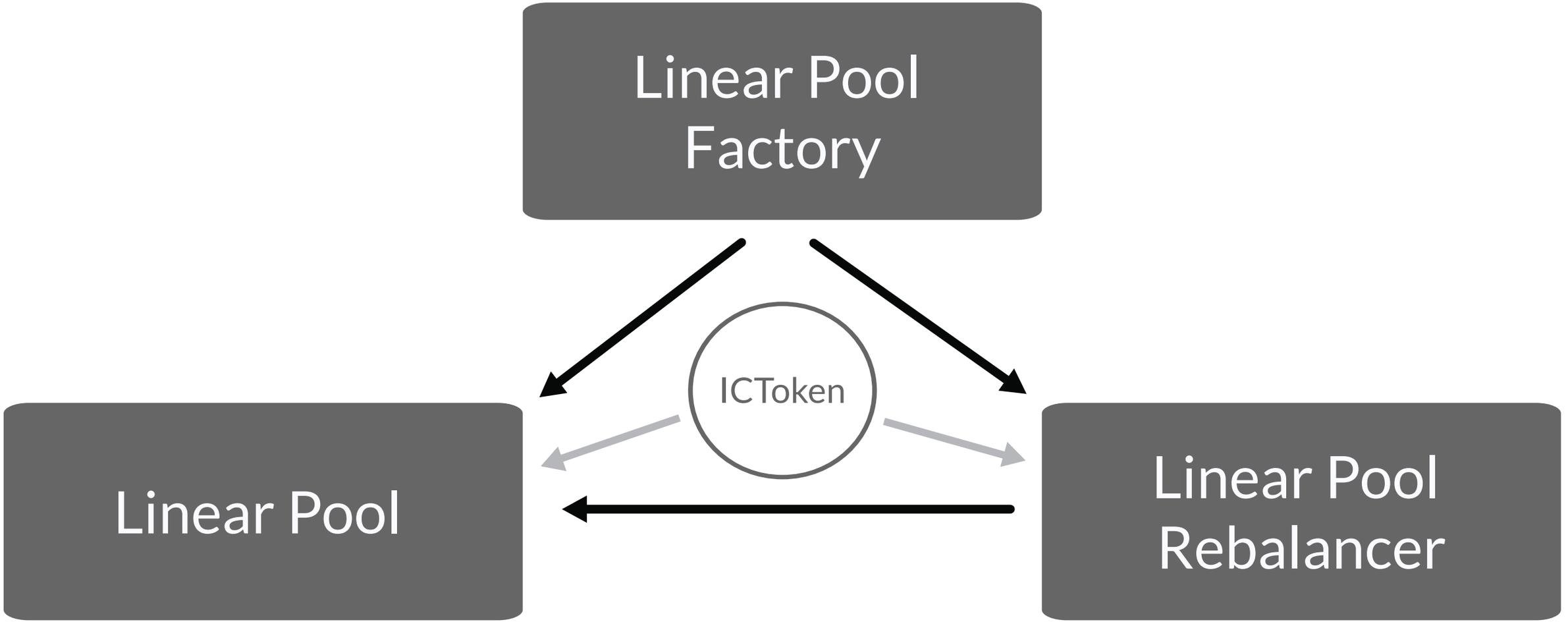
Compound Finance is a Defi lending protocol where users can stake assets to earn interest or borrow existing liquidity on the protocol.

cTokens:

Wrapped tokens give users the ability to earn interest through increasing exchange rates relative to the underlying token. They can also be used as collateral.

Compound Linear Pool Factory

The starting point for deploying Boosted Pools on-chain



Compound Linear Pool

The main logic of the linear pool sits within this contract

Necessary Imports

***Balancers LinearPool Contract*:**

- Reusable logic that is necessary for the creation of all Balancer Linear Pools.
- Functions for performing joins, exits, and various types of swaps.

***ICToken Interface*:**

- Necessary to interact with Compound Finance.
- Skeleton for cTokens created through the Linear Pool
- Function signatures match with necessary cToken functions in Compounds documentation v2.8

ICToken Interface

```
pragma solidity >=0.5.0 <0.9.0;

interface ICToken {
    /**
     * @dev returns the address of the cToken's underlying asset
     */
    function underlying() external view returns (address);

    /**
     * @dev Adds the wrapped tokens to compounds liquidity pool
     */
    function mint(uint256) external returns (uint256);

    /**
     * @dev Withdraws unwrapped tokens from compounds liquidity pool
     */
    function redeem(uint) external returns (uint);

    /**
     * @dev Gets the current exchange rate
     */
    function exchangeRateStored() external view returns (uint256);
}
```

Compound Linear Pool

```
// Returns the exchange rate for 1 wrapped token in main tokens
function _getWrappedTokenRate() internal view override returns (uint256) {
    uint256 rate = _cToken.exchangeRateStored();

    uint256 compoundScaling = SafeMath.add(10, _mainToken.decimals());

    int256 finalScaling = int256(compoundScaling) - 18;

    if (finalScaling < 0) {
        return rate * 10**abs(finalScaling);
    } else {
        return rate / 10**abs(finalScaling);
    }

}

// Returns the absolute value of a int256 and converts to a uint256
function abs(int256 number) private pure returns (uint256) {
    return number >=0 ? uint256(number) : uint256(-number);
}
```

Exchange Rate Calculation

- Crucial part in a successful Linear Pool creation.
- On-chain data from 3rd party protocols allow for trading to be performed smoothly.
- This function must always return an 18 fixed-point decimal.

Compound Exchange Rate Scaling

$$rate \times 10^{(18 - 8 + \text{Underlying Token Decimals})}$$

Compound Linear Pool Rebalancer

This contract makes sure that the tokens within the linear pool stay at an optimal ratio

Necessary Imports

LinearPoolRebalancer Contract:

- Reusable logic that is necessary for the creation of all Linear Pool Rebalancers.
- Uses swaps within private functions to manage token balances within the linear pool.

ICToken Interface:

- Redeem()
- Mint()
- ExchangeRateStored()
- Help execute the Rebalancer's logic with Compound Finance

Compound Linear Pool Rebalancer

```
contract CompoundLinearPoolRebalancer is LinearPoolRebalancer {
    // SafeERC20 wrapper that throw failures when contract returns false
    using SafeERC20 for IERC20;

    // These Rebalancers can only be deployed from a factory to work around a circular dependency: the Pool must know
    // the address of the Rebalancer in order to register it, and the Rebalancer must know the address of the Pool
    // during construction.
    constructor(IVault vault, IBalancerQueries queries)
        LinearPoolRebalancer(ILinearPool(ILastCreatedPoolFactory(msg.sender).getLastCreatedPool()), vault, queries)
    {
        // solhint-disable-previous-line no-empty-blocks
    }

    function _wrapTokens(uint256 amount) internal override {
        // No referral code, depositing from underlying (i.e. DAI, USDC, etc. instead of aDAI or aUSDC). Before we can
        // deposit however, we need to approve the wrapper in the underlying token.
        _mainToken.safeApprove(address(_wrappedToken), amount);
        ICToken(address(_wrappedToken)).mint(amount);
    }

    function _unwrapTokens(uint256 amount) internal override {
        // Withdrawing into underlying (i.e. DAI, USDC, etc. instead of aDAI or aUSDC). Approvals are not necessary here
        // as the wrapped token is simply burnt.
        ICToken(address(_wrappedToken)).redeem(amount);
    }

    function _getRequiredTokensToWrap(uint256 wrappedAmount) internal view override returns (uint256) {

        return ICToken(address(_wrappedToken)).exchangeRateStored() * wrappedAmount;

    }
}
```

Overview

- 3 virtual functions within the LinearPoolRebalancer that need to be overwritten.
- Constructor allows rebalancer to be grouped with the associated linear pool on-chain.

ABI & Bytecode extraction

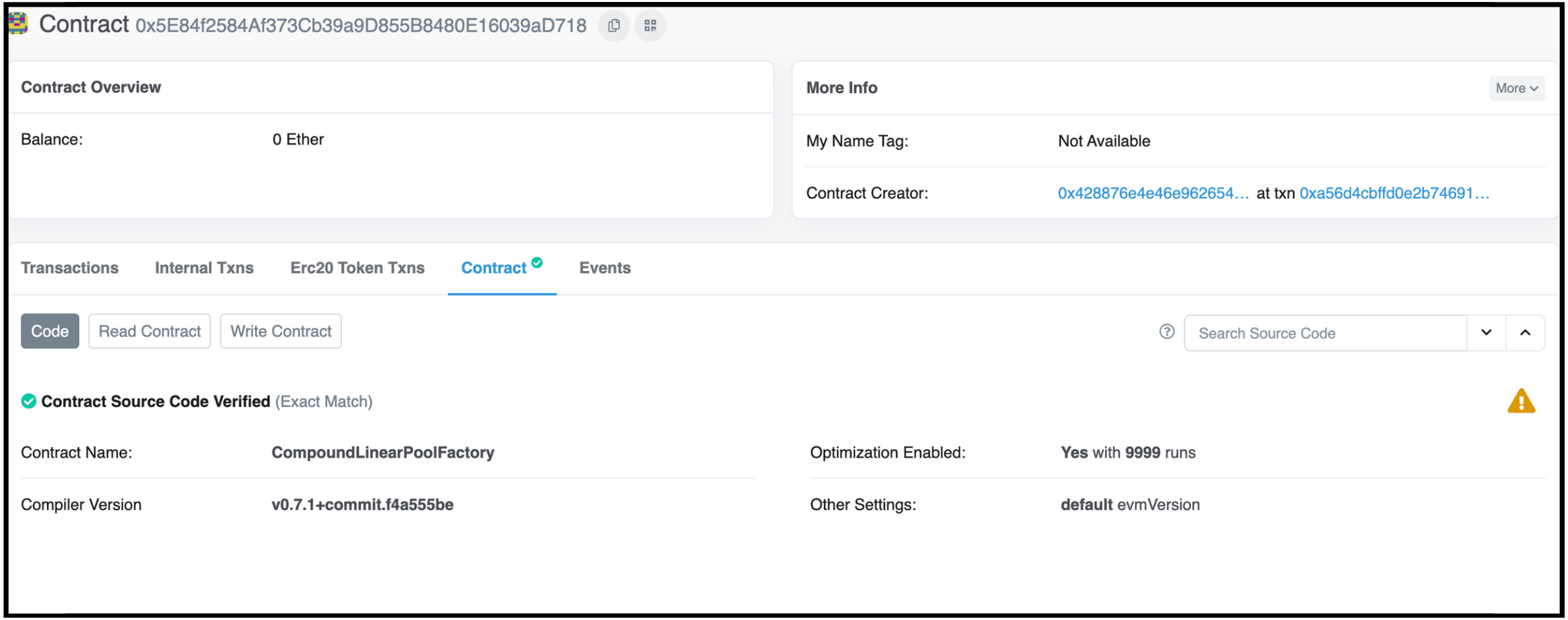
Blockchain deployment instructions:

Prerequisites: Please have testnet ETH and your network configuration complete and store at ~/.harhat/networks.json

1. Navigate to project folder and run the command `yarn hardhat compile`
2. Copy the json from the build-info folder within the artifacts folder
3. Navigate to the deployments folder and copy the build file into the folder associate with you project (create one with the date as a prefix if no folder exists)
4. Run the following commands (substitute your project and file names if you are not deploying the compound linear pool

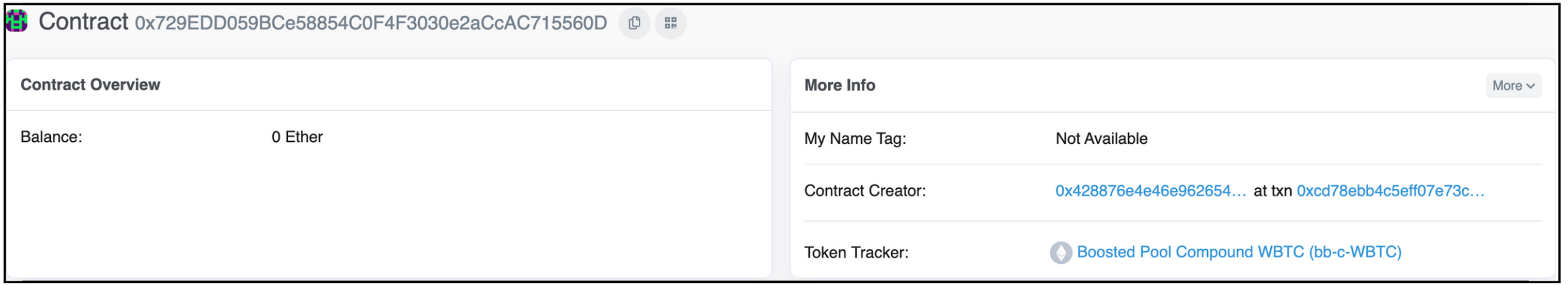
```
yarn extract-artifacts --id 20221111-compound-rebalanced-linear-pool --name CompoundLinearPoolFactory --file CompoundLinearPoolFactory
yarn extract-artifacts --id 20221111-compound-rebalanced-linear-pool --name CompoundLinearPool --file CompoundLinearPoolFactory
yarn extract-artifacts --id 20221111-compound-rebalanced-linear-pool --name CompoundLinearPoolRebalancer --file CompoundLinearPoolFactory
yarn hardhat deploy --id 20221111-compound-rebalanced-linear-pool --network goerli
```

Deployed & Verified Compound Linear Pool Factory



https://goerli.etherscan.io/address/0x5e84f2584af373cb39a9d855b8480e16039ad718

Created Compound Linear Pool



https://goerli.etherscan.io/address/0x729edd059bce58854c0f4f3030e2accac715560d