Introduction

Morpho is a decentralized lending protocol with different entities and individuals contributing to its development and adoption. As a result, the documentation refers to different areas of “Morpho” which are worth distinguishing.

- The Morpho Protocol:
  A decentralized, noncustodial lending protocol implemented for the Ethereum Virtual Machine. The protocol had two main steps in its evolution with two independent versions: Morpho Optimizers and Morpho.
- The Morpho Interface:
  A web interface allowing easy interaction with the Morpho protocol. This interface is one of many ways to interact with the Morpho protocol. More details in the Morpho Interface section.
- Morpho Governance:
  A governance system for governing the Morpho Protocol, enabled by the MORPHO token.
- Morpho Labs:
  The company that developed the Morpho Protocol, Interface, and many developer tools.
- Morpho Association:
  A France-registered association that regroups the main contributors like Morpho Labs to promote the development and the decentralization of the Morpho Protocol. The Morpho Association hosts the Morpho Interface.

Morpho

Morpho is the most recent protocol version and is independent of Morpho Optimizers.

Morpho is a simple lending primitive layer that allows the creation of immutable and efficient lending markets in a permissionless way.

The protocol comes with EVM smart contracts which facilitate interactions and integrations.

What is Morpho?

Morpho is a trustless and efficient lending primitive with permissionless market creation.

It enables the deployment of minimal and isolated lending markets by specifying:

- one collateral asset,
- one loan asset,
- a Liquidation Loan To Value (LLTV),
- an Interest Rate Model (IRM),
- and an oracle.

The protocol is trustless and was designed to be more efficient and flexible than any other decentralized lending platform.

Morpho Markets

As a reminder, Morpho has a permissionless market creation mechanism, where anyone can create a market with a different collateral token, borrowable token and oracle (IRM & LLTV are allowed by governance).

Such as Uniswap, you can be vulnerable to loss of funds if you use a random market where the oracle or the tokens involved in the market can be dumb. So, we recommend you use only the markets that are listed in the Morpho Interface.

Overview

Morpho is a decentralized protocol enabling the overcollateralized lending and borrowing of crypto assets (ERC20 & ERC4626 Tokens) on the Ethereum Virtual Machine. The protocol is implemented as an immutable smart contract, engineered to serve as a trustless base layer for lenders, borrowers, and applications.

Morpho is licensed under a dual license (BUSL-1.1 and GPLv2) which you can find here. Once deployed, Morpho will function in perpetuity, provided the existence of the Ethereum blockchain.

Decentralized, overcollateralized lending and borrowing

A decentralized, overcollateralized lending and borrowing protocol is an autonomous system that allows users to borrow assets by providing more collateral than the value of the borrowed assets and lenders to earn interest on supplied assets.

Basic features and components:

1. Collateralization: To borrow assets, a user must provide collateral as a crypto asset supported by the protocol.
2. Liquidation Loan-To-Value (LLTV): The protocol specifies the minimum value of collateral required relative to the borrowed assets. For example, if this ratio is

90%, the value of borrowed assets must not exceed 90% of the value of the collateral, or the position is eligible for liquidation.

3. Borrowing: A user initiates the borrowing process by interacting with the protocol. They specify the amount of the asset they want to borrow and provide the required collateral.
4. Interest Rates: Borrowers pay interest on the borrowed amount. The amount of interest paid is based on the interest rate model used by the protocol. Interest accrues over time and is payable when the borrower repays the loan.
5. Repayment: Borrowers can repay the loan at any time by returning the borrowed assets plus accrued interest. The borrower can retrieve his collateral from the smart contract once the repayment is confirmed on the blockchain.
6. Liquidation Mechanism: To mitigate the risk of default, protocols include a liquidation mechanism. Suppose the value of the borrowed assets exceeds the LLTV (due to market fluctuations or interest accrual). The position may be liquidated in part or full to repay the loan and any outstanding interest.
7. Lending: A user initiates the lending process by interacting with the protocol. They specify the amount of the asset they want to lend and transfer these assets to the smart contract.
8. Withdrawal: Lenders can withdraw their loan assets plus accrued interest at any time, assuming the market has enough liquidity.

Permissionless market creation

A distinctive feature of Morpho is permissionless market creation: the protocol allows users to create isolated markets consisting of one loan asset, one collateral asset, a Liquidation Loan-To-Value (LLTV), an oracle, and an interest rate model (IRM).

This a departure from the existing paradigm and traditional lending platforms which:

1. Require governance approval for asset listing and parameter changes.

2. Pool assets into a single lending pool, sharing risk across the entire protocol.

In Morpho, each parameter is selected at market creation and persists in perpetuity. Or, in other words, are immutable. The LLTV and interest rate model must be chosen from a set of options approved by Morpho Governance.

Once a market is created, users can either lend/borrow assets to/from it with certainty that:

1. It will persist as long as Ethereum, Base or any other network exists.
2. The parameters of the market will never change.

Morpho Markets

In Morpho, markets are named based on their individual parameters in the following format:

`CollateralAsset/LoanAsset (LLTV, ORACLE, IRM)`

Using the following parameters as an example:
CollateralAsset: `wstETH` LoanAsset: `WETH` LLTV: `94.5%` Oracle: `ChainlinkOracle` IRM: `AdaptiveCurveIRM`

The market would be named `wstETH/WETH (94.5%, ChainlinkOracle, AdaptiveCurveIRM)`

and will have an `id`, which is a hash of the market parameters as defined in the technical reference section.

Externalized Risk Management

Traditional lending platforms incorporate mechanisms that allow token holders to participate in risk management on behalf of users. However, this approach restricts the number of listed assets, confines users to a single risk profile, and is not scalable. Morpho removes these limitations by separating risk management from the protocol.

Permissionless Risk Management

Instead of relying on governance to list assets and manage parameters, Morpho is designed to leave choices up to the users. It allows anyone to create markets with any loan asset, collateral asset, risk parameters, or oracle and for users to interact with any deployed market.

Under this approach, users can independently assess and manage their risk and return without restrictions imposed by governance. This allows Morpho to satisfy broader risk appetites and support diverse use cases.

Advanced lenders benefit from the added flexibility, but for users accustomed to risk management being done for them, interacting with Morpho directly can be too complex.

However, Morpho was designed so that risk management layers can be rebuilt on top of the protocol to simplify the user experience.

One example are Morpho Vaults: a permissionless risk management protocol that facilitates the creation of lending vaults on top of Morpho markets. A curator will allow the vault to supply to multiple markets and depositors can delegate risk management by supplying to Morpho via Morpho Vaults. Passive lenders can supply WETH to a Morpho Vault. That vault allocates WETH liquidity to any number of Morpho markets with WETH as the loan asset. Instead of the user assessing which markets have the appropriate collateral, LLTV, oracle, and IRM, the vault does it on their behalf.

Morpho Vaults are only one example of permissionless risk management on top of Morpho. Any entity, DAO, or protocol can build services and tools that help manage risk on behalf of users.

Importantly, risk management is performed externally from Morpho. So any adverse outcomes, like fund losses from technical issues or poor management, do not affect Morpho itself.

Minimizing Governance

Morpho is designed to be a base layer for decentralized lending. Alongside immutability, removing the ability for governance to halt or modify markets is key to enabling completely trustless lending and borrowing. Learn more in the Governance & Fees section.

Liquidation

Morpho has a liquidation mechanism to mitigate the risk of default and protect lenders' capital.

When an account becomes unhealthy, meaning its Loan-To-Value (LTV) on a given market exceeds the market's Liquidation Loan-To-Value (LLTV), the account's position can be liquidated. Anyone can perform this liquidation by repaying the account's debt in exchange for the equivalent amount in the market collateral asset, along with an incentive.

To avoid being liquidated, borrowers can track their LTV in real-time and either repay their loan, partially or entirely, either add more collateral to their position.

How to calculate the LTV of a position?

To compute the LTV of a position on Morpho, use the following formula:

LTV=

$$BORROWED\_AMOUNT * ORACLE\_PRICE / COLLATERAL\_AMOUNT * ORACLE\_PRICE\_SCALE$$

where:

- $BORROWED\_AMOUNT$
- $BORROWED\_AMOUNT$ = the amount of borrowed assets of the user,
- $ORACLE\_PRICE$
- $ORACLE\_PRICE$ = the oracle price returned by the oracle of the market,
- $COLLATERAL\_AMOUNT$
- $COLLATERAL\_AMOUNT$ = the amount of collateral assets provided by the user,
- $ORACLE\_PRICE\_SCALE$

- $ORACLE\_PRICE\_SCALE$ =
- 1e36
- $1e36$.

How Liquidations Work

Liquidations on Morpho are quite simple: liquidators can liquidate up to 100% of the account's debt and receive the corresponding collateral value, plus the relative incentive.

No fee is taken by the Morpho protocol at this level. The entire Liquidation Incentive Factor (LIF) goes to the liquidator.

The LIF depends on the LLTV of the market, according to the following formula:

$LIF=min(M,\ 1/\ \beta*LLTV+(1-\beta))$,

with

$\beta=0.3$ and $\mathrm{M}=1.15$

Example

Liquidation Calculus

- A borrower provided \$100 of collateral in market A, that has an LLTV of 80%.
- The Liquidation Incentive Factor (LIF) is approximately 1.06 (as per the calculation or the picture above).

If the borrower’s debt reaches just above $80 (like $80.00001), the position is liquidatable as the Loan-To-Value (LTV) is just above 80%.
A liquidator can repay the borrower’s debt of approximately $80 and:

$$\mathrm{SeizableAssets}=\mathrm{debtAmount}*\mathrm{LIF}=80*1.06=84.8$$

$$SeizableAssets=debtAmount*LIF=80*1.06=84.8$$

The Liquidator will thus seize up to $84.8 of collateral, leaving the borro

Oracle-Agnostic Pricing

For a lending protocol to operate, it needs an accurate notion of price.

Oracles are the primary method for querying a price feed from an external source. They provide frequent and accurate pricing, taking into account many data points and diverse feeds. Determining a price without using oracles requires a lending protocol to have internal trading markets, also known as an oracle-less approach. However, this brings additional complexity, increasing gas consumption and compromising auditability and security.

To avoid compromising efficiency and security, Morpho is oracle-agnostic rather than oracle-less. The protocol has no oracle or trading mechanism built into it. Rather, anyone can create a market by specifying an address that returns the price for loan and collateral assets. Some markets can feature oracles such as Chainlink, Redstone or Uniswap, while other markets may have prices hardcoded or use a mechanism similar to the one seen in Ajna.

With this design, Morpho remains simple, trustless, and flexible while ensuring users can still benefit from the efficiency of oracles.

Interest Rate Models

Morpho is an Interest Rate Model (IRM) agnostic protocol, meaning it can support any interest rate model for its markets. In Morpho, the interest borrowers pay in a given market is defined by the IRM chosen at market creation among a governance-approved set.

Initially, this set is composed of one immutable IRM, the AdaptiveCurveIRM.

The AdaptiveCurveIRM

The AdaptiveCurveIRM is engineered to maintain the ratio of borrowed assets over supplied assets, commonly called utilization, close to a target of 90%.

In Morpho, the collateral supplied is not rehypothecated. Removing this systemic risk removes the liquidity constraints imposed by liquidation needs. It enables more efficient markets with higher target utilization of capital and lower penalties for illiquidity, resulting in better rates for both lenders and borrowers.

As with every parameter of a Morpho Market, the IRM address is immutable. This means that neither governance nor market creators can change it at any given time. As such, the AdaptiveCurveIRM is designed to adapt autonomously to market conditions, including changes in interest rates on other platforms and, more broadly, any shifts in supply and demand dynamics.

Its adaptability enables it to perform effectively across any asset, market, and condition, making it highly suitable for Morpho's permissionless market creation.

How It Works

The model can be broken down into two complementary mechanisms:

1. The Curve Mechanism This mechanism is akin to the interest rate curve in traditional lending pools. It manages short-term utilization effectively, maintaining capital efficiency while avoiding excessively high utilization zones that could lead to liquidity issues.
    -

2. The Adaptive Mechanism This mechanism fine-tunes the curve over time to keep the range of rates in sync with market dynamics. It achieves this by adjusting the value of $r90\%$ which in turn shifts the entire curve:

- When utilization exceeds the target, the curve continuously shifts upward. This incentivizes loan repayment and thus decreases utilization.
- When utilization falls below the target, the curve continuously shifts downward. This incentivizes borrowing and thus increases utilization.

The speed at which the curve adjusts is determined by the distance of current utilization to the target: the further it is, the faster the curve shifts. This incremental adjustment of the curve allows for rate exploration, ultimately stabilizing when the interest rate at the target utilization aligns with the market equilibrium.

Positions

Overview

On Morpho, a position is created whenever a user supplies, provides collateral, borrows, or engages in both activities on a Morpho market.

Unlike protocols like Aave or Compound, positions in Morpho are not tokenized as aTokens or cTokens. Instead, they are tracked using a mapping at the singleton smart contract level. This approach simplifies the system and reduces dependency on ERC20 token standards, offering a more efficient way to manage user positions.

Details

Supply, Borrow, Withdraw, Repay

In Morpho, the operations of supply, borrow, withdraw, and repay do not involve minting ERC20 tokens like Compound's cTokens. Instead, the system tracks users' positions using a mapping at the smart contract level. It uses shares for the loan asset.

Why Use Shares?

Shares are used to manage and track the distribution of assets within the Morpho protocol. This design helps ensure precise accounting and mitigate potential issues. Here are the main points:

1. Precision in Accounting:

- Shares allow for high precision in calculating the value of assets and liabilities, ensuring accurate representation of supply and borrow balances, even with fractional values.
- `SharesMathLib` provides functions (`toSharesDown`, `toSharesUp`, `toAssetsDown`, `toAssetsUp`) to convert between `assets` and `shares`, rounding up or down as needed. This ensures precise handling of all operations.

2. Mitigation of Share Price Manipulation:

- To prevent manipulation of the share price, an offset with virtual shares and virtual assets is used. This concept, borrowed from OpenZeppelin's ERC4626 standard, helps mitigate inflation attacks.
- These virtual shares and assets ensure that even when the market is empty, a conversion rate between shares and assets is maintained.

3. Flexible Operations:

- Users can choose to specify the amount in either assets or shares, with the other parameter set to zero. This flexibility allows users to supply or borrow exactly the amount they need.
- For instance, if a user wants to borrow a specific amount of assets, they can specify that amount in the assets parameter and set shares to 0. Conversely, if they want to specify the amount in terms of shares, they can set assets to 0. Note that it is particularly useful to fully close a position with a full repay by providing the number of shares, same for withdrawals.

4. Simplified Interest Accrual:

- Shares remain constant for a user if there are no interactions, simplifying the handling of interest accrual. This makes it easier to repay the exact amount without worrying about accruing interest on the assets.

- This approach is particularly useful for withdrawals, as users can specify the amount in shares to avoid issues with rounding or changing asset values.

Example Usage

When someone supplies assets to a Morpho market (see the `supply` function in Morpho.sol):

Either assets or shares should be zero.

- If assets > 0, the system calculates the corresponding shares.
- If shares > 0, the system calculates the corresponding assets. Similarly, when borrowing, repaying or withdrawing from a market (see the respective function in Morpho.sol).

Advanced Concepts

Singleton Contract

Morpho is designed as a singleton contract. This means all markets are contained within one single smart contract, rather than each market having a separate contract. This design simplifies the protocol and significantly reduces gas consumption when users interact with multiple markets.

Benefits of the Singleton Contract

- Simplified Interactions: Users can interact with multiple markets within a single transaction, saving on gas fees.
- Concentrated Liquidity: The singleton contract concentrates liquidity for flash loans, making it more efficient and accessible.
- Unified Logic: All core functionalities and user-facing entry points are contained within one file, Morpho.sol, making the protocol more readable and efficient.

Key Features

- User-Facing Entry Points: Includes functions like `supply`, `withdraw`, `borrow`, `repay`, `supplyCollateral`, `withdrawCollateral`, and `liquidate`.
- Governance Functions: Includes functions for enabling new Interest Rate Models (IRMs), Loan-to-Value (LLTV) ratios, and a fee switch capped at 25%.
- Free Flash Loans: Allows borrowing from all markets simultaneously, provided they are repaid in the same transaction. This is useful for liquidations, leverage positions, and arbitrage.
- Callbacks: Each entry point where tokens move from the user to Morpho (e.g. `supply`, `supplyCollateral`, `repay`, and `liquidate`) supports callbacks with arbitrary data, removing dependencies on external flash loans and reducing gas costs.

Advanced Features

- Bad Debt: Mechanisms to handle bad debts within the protocol.
- Callbacks: Detailed explanation of how callbacks work within the entry points.
- Flash Loans: Overview of the free flash loan functionality.

- Account Management: Authorization system that allows users to grant permissions to other addresses for borrowing and withdrawing on their behalf.

Additional Information

- Interest Rate Models and Oracle Adapters: The contract includes various IRMs and Oracle adapters, which are crucial for the functioning of Morpho. These are covered in the IRM and Oracles sections.

Bad Debt

Morpho has different mechanisms for accounting for bad debts.

Morpho Vaults v1.0 - Bad debt realization

The Morpho Vaults created with the MetaMorpho Factory v1.0 have a mechanism to account for and realize bad debt in the event it arises.

Typically, in other lending pool designs, accrued bad debt remains in the market forever until manual intervention to pay down the bad debt. If small enough, the markets can continue functioning. If it is significant, the market becomes unusable.

Morpho Vaults v1.0 treat bad debt differently. When a liquidation leaves an account with some remaining debt, and without collateral to cover it, the loss is realized and shared proportionally between all lenders.

As bad debt is realized at the time it occurs, a market can be used in perpetuity.

Morpho Vaults v1.1 - No bad debt realization

The Morpho Vaults created with the MetaMorpho Factory v1.1 have a different mechanism to account for as it does not realize bad debt and behave as other lending pools with accrued debt remaining in the market forever until manual intervention to pay down the bad debt.

Callbacks

A callback is a piece of code that is passed as an argument to another function. The purpose of a callback is to be run during the execution of the function it is attached to.

Morpho has callbacks on the following functions:

1. `supply`
2. `supplyCollateral`
3. `repay`
4. `liquidate`

Developers can leverage callbacks to execute custom logic between the accounting of these functions and the actual transfer of assets, allowing for more advanced operations to occur on Morpho.

For more information and examples, feel free to visit the technical guide of callbacks.

Flash Loans

Flash loans are loans that can be taken without any collateral if the borrowed assets are repaid in the same transaction. Flash loans are free in Morpho, and thanks to the singleton architecture, users of the flash loan function can access the liquidity of all markets simultaneously.

Free Flash Loans

Morpho's singleton has a free flash loan function allowing one to borrow from all markets simultaneously, as long as they are repaid in the same transaction. This is typically useful in DeFi for liquidations, setting up leverage positions, and onchain arbitrages.

Overview

Flash loans are a unique concept in decentralized finance (DeFi) that allows users to borrow funds instantly without providing collateral. Aave is credited with creating the concept of flash loans. These loans are executed within a single blockchain transaction and must be repaid within the same transaction. They have been popularized by Aave and dYdX and are primarily designed for developers and users with technical knowledge.

If you are already familiar with the concept, jump into the implementation section or the technical reference of the `flashLoan` function.

Use case

Flash loans have various use cases, such as arbitrage, swapping collateral, and self-liquidation. However, they have also been associated with security implications, as they have been used in highly profitable DeFi exploits. Despite this, they are considered a groundbreaking innovation in the DeFi space.

○

Account Management

Morpho has an authorization system enabling users to grant permission for an address to act on their behalf. An authorized address can borrow on behalf, withdraw on behalf, and withdraw collateral on behalf of the authorizer.

Authorizations can be managed via:

1. A classic function call,
2. Message signature following EIP-712 standard.

Account management allows users of externally owned accounts (EOAs) to batch interactions using bundler contracts and smart contract wallets.

It is a primitive authorization system in the sense that it allows the implementation of tailored, granular management systems to be built on top.

For more information and examples, feel free to visit the technical guide of the account managment.

Account Management Overview

Morpho has an authorization system enabling users to grant any address the permission to manager their position. This can be done by calling `setAuthorization` or by passing a signed message (similar to permit signatures for ERC20 tokens) through the `setAuthorizationWithSig` function.

Use case

The account management system unlocks the following use cases (not exhaustive):

- Batching: By setting authorization for a contract to manage their positions, EOAs can batch transactions. The bundler's contracts leverage this feature to batch transactions and reduce gas costs.
- Granular management: Users can grant specific addresses the permission to manage their position. This can be used to implement granular management systems, such as a DAO managing a user's positions.
- Automated leverage/deleverage: Users can grant specific contracts with specific logic the permission to leverage or deleverage their position depending on market conditions.

Functions

`setAuthorization`

This function allows a user to set authorization for another address to manage its positions directly. It updates the `isAuthorized` mapping and emits a `SetAuthorization` event for tracking.

`setAuthorizationWithSig`

This function enables authorization setting through a signed message, removing one call for EOAs. It checks for signature validity, ensuring that it hasn't expired and that the correct authorizer signs the message. Upon successful validation, it updates the authorization status and emits a `SetAuthorization` event for tracking.

Governance & Fees

Governance

The role of governance in Morpho is purposely minimized to prioritize a trustless experience. Morpho governance cannot halt the operation of a market or modify its LLTV, IRM, or oracle. However, governance is responsible for whitelisting new LLTVs and IRMs that users can select at market creation.

Fees

Morpho has a fee switch built into the protocol. Governance can enable a fee ranging from 0% to 25% of the total interest amount paid by borrowers for a given market.

Incentives

Context

Unlike traditional lending pools, Morpho allows projects to take a more targeted approach to incentivizing specific use cases. On Morpho, they can incentivize a specific pair of assets with a strong use case that will help drive further adoption of their token. As a result, many projects will want to distribute incentives on Morpho.

The Universal Rewards Distributor (URD) is a central component of this incentive distribution mechanism.

Universal Rewards Distributor (URD)

Overview

The Universal Rewards Distributor (URD) is a bespoke smart contract tailored for the distribution of incentives, usable on Morpho markets. Its primary function is to manage the distribution of incentives, focusing on specific asset pairs to stimulate token adoption and market activity.

Purpose of URD

The URD is engineered to bring uniformity and efficiency to the incentive distribution process. It caters to both external project rewards and MORPHO tokens, addressing the common challenges and complexities associated with rewards distribution in DeFi.

How URD Functions

Offchain Computation: The process starts with an offchain script that computes a Merkle tree, which outlines the rewards each user is entitled to.

Smart Contract Utilization

Initially under the governance of the Morpho DAO, this contract allows the Morpho Operator to periodically update the Merkle root, ensuring the rewards distribution remains accurate and up-to-date.

Claiming Rewards:

Upon each update of the Merkle root within the contract (after the end of an epoch), users can claim their designated rewards.

Additional Features and Considerations:

For a comprehensive insight into the URD's technical makeup, including aspects like owner and updater management, updater specifications, Merkle tree intricacies, and the

specificities of the claim process, please refer to the Technical Reference section.
It's crucial to acknowledge and understand the inherent limitations of the URD, such as the absence of a queue mechanism for the pending root. Appropriate strategies and planning are necessary to navigate these limitations and ensure smooth and efficient operations.

For additional details and guidance on how to integrate with the URD, please consult the official URD documentation.

Rewards Emission Data Provider

The rewards emission data provider allows the setup and adjustment of reward emissions rates with precision, catering to the nuanced needs of different market conditions and strategic goals.

Jump on the rewards section for further details.

Benefits of Morpho

Morpho is a trustless lending protocol that offers greater efficiency and flexibility than existing lending platforms. Its primitive design makes it the ideal building block for users and applications.

Trustless

Immutable: Morpho is not upgradable. The protocol will run and behave the same way forever.

Governance-minimized: Morpho Governance cannot halt the operation of a market or manage funds on users' behalf, nor does it impose specific oracle implementations.

Simple: The protocol consists of only 650 lines of Solidity code. This simplicity makes it particularly easy to understand and safe.

Efficient

Higher collateralization factors: Morpho's Lending markets are isolated. Unlike multi-asset pools, liquidation parameters for each market can be set without consideration of the most risky asset in the basket. Therefore, suppliers can lend at a much higher LLTV while being exposed to the same market risk as when supplying to a multi-asset pool with a lower LLTV.

Improved interest rates: Collateral assets are not lent out to borrowers. This alleviates the liquidity requirements for liquidations to function properly in current lending platforms and allows Morpho to offer higher capital utilization. Moreover, Morpho is fully autonomous, so it does not need to introduce fees to cover costs for platform maintenance.

Low gas consumption: Morpho is a remarkably simple protocol built in a singleton smart contract that groups every possible primitive market in the same place. This reduces gas consumption by 50% compared to existing lending platforms.

Flexible

Permissionless market creation: Morpho features permissionless asset listing. Markets with any collateral and loan assets and any risk parametrization can be created. The protocol also supports permissioned markets, enabling a broader range of use cases, including RWAs and institutional markets.

Permissionless risk management: Morpho allows for additional layers of logic to enrich its markets. More specifically, risk management can be built on top of the protocol to simplify the user experience for lenders.

For example, risk experts could build noncustodial vaults for lenders to earn yield passively. These vaults can allocate depositors liquidity to multiple markets on their behalf. This allows users to pass off the responsible of risk management to the vault rather than doing it themselves.

Developer-friendly: Morpho features several modern smart contract patterns. Callbacks enable liquidators and sophisticated users to chain advanced actions without any flash loans. Account management facilitates gasless interactions and account abstractions. Free flash loans on the singleton contract allow anyone to access the assets of all markets simultaneously with a single call, as long as they are repaid in the same transaction.

Security

Morpho is known for its industry-leading security practices and follows a multi-faceted approach to security.

Morpho security practices include formal verification, mutation tests, fuzzing, unit testing, and peer reviews that can be found within respective Github repositories. External measures include professional security reviews, contests, and pre/post-deployment bounties.

A whole article was dedicated to the Morpho Security Framework here.

Over time, Morpho has been audited 27 times by 12 different security firms. This covers Morpho Optimizers, Front-ends, Morpho Vaults and Morpho, and to our knowledge, makes Morpho the most audited project in the world. Feel free to refer to the following 3 sections for more details about respective security reviews:

- Morpho
- Morpho Vaults
- Morpho Optimizers

Morpho Optimizers is renowned for achieving the highest score (98%) out of 300+ protocols on DeFiSafety, an independent rating agency that assesses how closely a protocol follows best practices.

More information on specific security practices is shared in the following sections.

Security Reviews

Here is the list of all security reviews conducted on Morpho smart contracts.

The code is immutable and public-source. Multiple independent security firms and individual auditors have audited the protocol.

The following table lists all security reviews conducted on Morpho smart contracts.

Smart contracts security reviews

Auditors	Scope	Date	Report
Cantina Contest	Morpho	2023-11 to 2023-12	cantina-competition

Spearbit	Morpho	2023-10-16	cantina-managed-review
Open Zeppelin	Morpho	2023-10-13	open-zeppelin

Bug Bounty

Bug bounties encourage security researchers and whitehat hackers to find and report potential vulnerabilities.

There are 2 ongoing bug bounty programs:

1. Immunefi - $2,500,000,
2. Cantina - $2,500,000.

Formal Verification

Formal verification is the process of mathematically checking that the behavior of a system, described using a formal model, satisfies a given property. In essence, it proves the smart contract performs as it should.

Morpho’s core properties have been formally verified using CVL, Certora's Verification Language. It is done by giving a high-level description of the verification and then describing the folder and file structure of the specification files.

For all the formal verification performed on Morpho to date, visit the dedicated folder in the Github repository.

The following table lists all formal verifications conducted on Morpho smart contracts.

Formal verifications

Formally Proven	Scope	Date	Tool Used
morpho-token	ERC20 and delegation logic	December 2024	Certora
universal-rewards-distributor	Merkle tree and claim function	April 2024	Certora & custom checker
morpho-blue	core logic	December 2023	Certora & Halmos
semitransferable-token (morpho token)	authorization system	August 2022	Certora

Risk Documentation

Morpho is committed to use industry-leading security practices. Yet, there are still a number of risks associated with the use of Morpho that users must be aware of.

Smart contract risk

There is an inherent risk that the protocol could contain a smart contract vulnerability or bug.

Several security measures are employed to mitigate this risk:

- Core contracts are immutable,
- It is a simple and public-sourced code base that avoids complexities,
- The code has been audited by OpenZeppelin and Cantina, in addition to a competitive security review.
- Formal verification has been applied using Certora,
- 2 ongoing bug bounty programs:
    i. Immunefi - $2,500,000,
    ii. Cantina - $2,500,000.

More details about Morpho's security and security reviews can be found in the dedicated section.

Oracle risk

Every Morpho market is connected to an oracle, established at market creation. It is important to understand that no oracle is immune to price manipulation, which can lead to liquidations or even bad debt. However, some oracles will be more resistant and resilient than others.

When assessing the reliability of an oracle, consider factors such as safety and liveness, particularly if the oracle is centralized. Also, take into account the settings and processes pertaining to the definition and frequency of price updates.

Counterparty risk

Before entering a market, it's crucial to conduct thorough due diligence on the loan asset and the collateral asset to understand who holds power over them. Factors to consider include centralization, as a centralized governance could blacklist a specific user or even Morpho, resulting in a loss of funds. The distribution of the asset is also important, as a high concentration can cause extreme price fluctuations.

Liquidation risk

Liquidation risk (for borrowers)

Each Morpho market is linked to an immutable Liquidation Loan-to-Value (LLTV). If the Loan-To-Value of your position exceeds this LLTV, you will face liquidation. When borrowing on Morpho, carefully select the market and diligently manage the health of your position.

Bad debt risk (for lenders)

There could be circumstances in which the collateral's value for a position drops below the borrowed amount before liquidators can close the position. In such cases, the borrower holding this position has no incentive to repay the debt. Morpho has different mechanisms for accounting for bad debts. You can read more about it in the bad debt section.

Liquidity risk (for lenders)

Liquidity refers to the access to supplied assets. A lack of liquidity can prevent suppliers from withdrawing their assets for a certain period of time. Liquidity issues are tackled through the interest rate model. Before providing liquidity, it's essential to understand the market's interest rate model. This understanding will help you estimate the level of liquidity you can expect in that market.

Morpho Vaults

What are Morpho Vaults?

Morpho Vaults is a protocol for permissionless lending vaults built on top of the Morpho protocol.

Morpho Vaults Overview

Morpho Vaults is an open-source (GPL) protocol for permissionless risk curation on top of Morpho. The MetaMorpho Factory is a contract deploying ERC4626-compliant Morpho Vaults. Each vault has one loan asset and can allocate deposits to multiple Morpho markets. Morpho Vaults are noncustodial and immutable instances. Morpho Vaults offer users a way to provide liquidity and earn interest passively. Vaults have a system to automate risk management so that users are not required to make these decisions. Instead, the vault actively curates a risk exposure for all deposited assets.

The vaults operate in a noncustodial way and users maintain full control over their assets. Users can look at the state of the vault at any time and withdraw their liquidity at their discretion.

Motivation

Morpho is a trustless primitive prioritizing efficiency and flexibility over end-user experience. Lending to Morpho markets is more complex than on Aave and Compound. On Morpho, suppliers must consider multiple factors, including collateral assets, liquidation LTV, oracles, and caps. Compared with Aave where all the decisions are made by governance on the suppliers' behalf. As such, interacting with Morpho directly is more suited to sophisticated lenders than passive lenders. Morpho Vaults facilitate the supply of liquidity to Morpho markets. Users can delegate risk management to a vault which can automate and decentralize these decisions, similar to platforms like Aave or Compound, making for a more passive experience.

Permissionless Risk Management

Anyone can use the MetaMorpho Factory to create a Morpho Vault. There is no restriction on who can create a vault. Risk experts, DAO, protocols, etc. can all leverage the open infrastructure to provide passive users with a simple lending experience.

Morpho Vaults Core

Risk curation

Each Morpho Vault can be curated to reflect a different risk profile based on the goal and value proposition of the vault.

For example, a Morpho Vault dedicated to lending against Liquid Staking Tokens (LSTs) collateral(s) on Morpho would expose its users to a specific curated set of LSTs collateral assets. A user depositing this vault would not be exposed to any other collateral asset.

This differs from traditional lending platforms that do not offer the same flexibility. Users only have one choice: to be exposed to every asset listed in the pool. Risk curation offer users the ability to select a vault that aligns with their desired balance between risk and return and that reflects their individual risk appetite.

Timelock

Timelocks preserve the noncustodial property of Morpho Vaults, ensuring users maintain control of the assets at any time and can withdraw them if they want to.

A timelock is a mechanism that introduces a delay or waiting period before certain functions or actions can be executed. This adds a layer of security, giving users of a Morpho Vault time to review and react to proposed changes.

Starting from Morpho Vaults v1.1, at the initial setup of a vault, the timelock can be set to 0 to allow the curator to configure the vault. Once the configuration is complete, the curator can adjust the timelock within the bounds of 1 day to 2 weeks (expressed in seconds). This timelock applies to subsequent updates that may significantly impact the vault's risk profile, such as enabling high LLTV markets or increasing supply caps. This mechanism ensures users have sufficient time to exit their positions if they disagree with proposed risk changes.

To inspect the `timelock` value, one can jump into the "Read Contract" section of any Morpho Vault, for instance here.

Versions Overview

Morpho Vaults are available in two versions: v1.0 and v1.1. The latest v1.1 version introduces important improvements focused on enhancing integration capabilities and operational flexibility. Most notably, v1.1 vaults handle bad debt differently, making them compatible with flash loans and other DeFi primitives. All new vaults should be deployed using v1.1 to take advantage of these improvements through the Morpho Vault Factory v1.1.

Feature	v1.0	v1.1
Bad Debt Mechanism	Realizes bad debt	Does not realize bad debt
Initial Timelock	Minimum 24h required	Can be set to 0 for initial setup
Vault Name/Symbol	Immutable	Mutable, can be updated
Market Reallocation	Allows reallocation even if market disabled	Always reverts if market is disabled

Morpho Vaults Roles

Owner

The owner of a Morpho Vault can range from a DAO, traditional asset management, crypto-native risk expert, RWA risk expert, or even an autonomous algorithm.

Each vault can only have one owner and this address can set additional roles including one curator, one guardian, and multiple allocators to help manage the vault.

Curator

The Curator is responsible for setting the bounds of the vault by enabling or disabling certain Morpho markets and modifying supply caps.

Each vault can have one Curator.

Allocator(s)

An allocator is responsible for optimizing APY on the assets deposited in the vault by dynamically supplying and rebalancing across markets within the bounds set by the Curator.

A vault can have multiple Allocator roles. This role can be performed manually or automated using bots.

Guardian

The Guardian can revoke any action in the pending timelock including a change to the guardian, by the owner, and modifications to supply caps. The Guardian could be a DAO made up of vault depositors.

A vault can only have one Guardian which is responsible for protecting the interests of vault depositors.

For more on each of these roles and their permissions, visit the Morpho Vaults README.

Governance & Fees

The Morpho Vaults protocol itself is governance-less, meaning Morpho governance is not involved in the operations of the protocol or individual Morpho Vaults.

A Morpho Vault is governed by the owner of the smart contract. As explained in the Role section, the owner can also assign one curator, one guardian, and multiple allocators to help manage the vault.

Morpho DAO can’t take fees on Morpho Vaults but Vaults owners can set a performance fee to their Morpho Vault. The fee works by taking a % of interest generated by the Vault. The maximum performance fee is 50%.

Previous

Benefits of using Morpho Vaults

Morpho Vaults serve two types of users. The MetaMorpho Factory allows risk experts to spin up noncustodial lending vaults, and Morpho Vaults provide users with a simpler way to lend on Morpho.

For Lenders

Morpho Vaults offer users the same passive experience as traditional lending pools with the following notable improvements.

Curated Risk Profiles

Morpho Vaults offer a range of vaults with various risk levels instead of the traditional one-size-fits-all approach. This allows users to choose vaults that align with their general risk preferences/appetite. In traditional lending, pools are exposed to the riskiest asset in the pool. With Morpho Vaults, a lender can avoid exposure to long-tail assets by depositing in a vault that only lends against blue chip collateral.

Better Yields

Earn better yields from Morpho’s capital-efficient markets. Interest earned by suppliers are optimized by dynamically rebalancing across multiple markets with the best risk-return ratio.

Transparent

Morpho Vaults are noncustodial with immutable logic and verifiable onchain allocations. Risk curators are encouraged to share their processes with users, giving them more insight into the risk models and computations.

For Risk Experts/Protocols

Morpho Vaults pave the way for a new approach to risk management that is better for both risk experts and users.

Service users, not a DAO

Morpho Vaults enable risk protocols and experts to serve users directly instead of consulting for DAOs. This approach is a more scalable business model and is better aligned with users.

Permissionless infrastructure

Requires no code and is audited, lowering barriers to entry. Risk experts can create revenue-generating products with limited costs and effort. Benefit from Morpho SDK, front-end, rewards distribution, and other tooling.

Shared Liquidity

Morpho vaults combine the best of isolated markets and multi-asset lending pools to create a better way to lend. Over time, it is anticipated that Morpho vaults will emerge as the standard lending solution.

The Morpho Approach

There are two main approaches to structuring lending markets in decentralized finance: lending pools and isolated markets. The former excels in providing a straightforward user experience and aggregates liquidity but lacks the efficiency and flexibility needed for significant scalability. Conversely, the latter unlocks notably higher levels of efficiency and flexibility but introduces a more complex user experience and liquidity fragmentation.

Thus, the question arises: How does one combine the simple user experience and aggregated liquidity of lending pools with the efficiency and flexibility of isolated markets?

The answer: Morpho Vaults & Morpho.

Morpho is a remarkably simple and immutable lending primitive that enables the permissionless creation of isolated markets. Morpho Vault is a separate protocol for creating Morpho Vaults (lending vaults) on top of Morpho.

Although built independently, Morpho Vaults are fully integrated into Morpho. At the base layer, Morpho provides efficient, secure, and flexible isolated markets. Built on top, Morpho Vaults simplify the lending user experience and aggregate liquidity offering users the best of both isolated markets and lending pools.

Simplifying Isolated Markets

First, lending to isolated markets needs to be as simple as a lending pool.

When using a multi-asset lending pool, a lender only has one option, however, with isolated markets there could be several markets for one loan asset. For example, Morpho could have five options to lend USDC: sDAI/USDC, wstETH/USDC, wbIB01/USDC, WBTC/USDC, and WETH/USDC.

In other words, lenders must manage their own risk by choosing their collateral exposure, Liquidation Loan-To-Value (LLTV) ratios, oracle, caps, etc.

Morpho Vaults eliminate the complexity of risk management by creating a single point of entry. Rather than requiring users to make multiple decisions, they can simply deposit USDC into a USDC Morpho Vault to allocate liquidity.

Not only does it make it easier to supply, but it helps to improve yield. As market conditions change, a vault can rebalance across markets to optimize interest earned by lenders.

In the end, Morpho Vaults provide users with the same simple user experience as a multi-asset lending pool with the benefits of lending to isolated markets.

Enabling Diverse Risk Profiles

Going beyond one-size-fits all

Every user has their own tolerance for risk, while each collateral asset has a set of associated risks, some with more than others.

For that reason, users with a lower risk tolerance may prefer to avoid lending against certain collateral assets, whereas it may be the opposite for users with a greater appetite for risk.

However, a core limitation of multi-asset lending pools is that all users are forced into a one-size-fits-all risk profile, regardless of their risk appetite.

The above illustrates how every lender to a multi-asset pool is exposed to every collateral asset in the pool: wstETH, WBTC, LINK, USDT, sDAI, DAI, rETH, RPL, AAVE, USDC, LINK, etc. There is no option to lend against specific collateral or a combination of collateral assets aligned with one's tolerance for risk.

Catering to any risk profile

Morpho Vaults are built on top of Morpho’s isolated markets with each vault having a unique risk profile determined by the markets they lend to.

Lenders can choose to deposit in vaults that best align with their risk appetite. Users with a higher risk tolerance can deposit in a vault that lends to “riskier” markets and vice versa.

For example, the chart shows three different lenders depositing into three different vaults:

- USDC Vault lending to wstETH/USDC & wbIB01/USDC.
- ETH Vault lending to wstETH/WETH & sDAI/WETH.
- ETH Vault lending to weETH/WETH, osETH/WETH, and swETH/WETH.

Each lender can deposit into one or more vaults to tailor their risk exposure:

- Lender #1 has exposure to wstETH (crypto) and wbIB01 (RWA) markets.
- Lender #2 has exposure to wstETH and sDAI markets.
- Lender #3 has exposure to weETH, osETH, and swETH markets.

Here, lenders can opt in and out of certain markets by selecting specific vaults. This architecture enables a level of flexibility unattainable by traditional lending pools. It can scale to any number of vaults, markets, and risk profiles catering to any risk profile imaginable.

Aggregating then Amplifying Liquidity For Lenders

This part explains one the most powerful features of the Morpho approach: how Morpho Vaults aggregate and then amplify withdrawable liquidity to give lenders a better liquidity profile than multi-asset lending pools.

We break down this part into three sections:

1. Why isolated markets fragment liquidity
2. How Morpho Vaults reaggregate liquidity to match the liquidity profile of a lending pool
3. How multiple Morpho Vaults sharing liquidity amplifies withdrawable liquidity beyond what is possible in lending pools

1. Isolated markets fragment liquidity

One of the main drawbacks of isolated markets is liquidity fragmentation.

For lending protocols, liquidity is the assets available for users to withdraw or borrow from the market immediately. It is defined by:

$$\mathrm{liquidity} = \mathrm{totalsupply} * (1 - \mathrm{UtilizationRate})$$

$$liquidity = totalsupply * (1 - UtilizationRate)$$

For example, a market with $1000 supply and 90% utilization rate has $100 liquidity.

When $1000 is supplied to a single lending pool versus equally across five isolated markets, the lending pool would have much more liquidity ($100) than individual markets ($20).

2. Reaggregating liquidity

Lending to isolated markets via a Morpho Vault solves liquidity fragmentation. The chart below illustrates how liquidity from each market is aggregated resulting in users having the same liquidity profile as a multi-asset lending pool, despite the underlying markets remaining isolated.

3. Shared Liquidity amplifies liquidity beyond lending pools

Now, the key realization: With Morpho Vaults, the liquidity profile for lenders is even better than a lending pool. This works as liquidity from each vault is is aggregated on Morpho and therefore shared by anyone lending to the same markets.

This chart illustrates how liquidity increases when there is a second Morpho Vault.

Let us break it down:

1. Morpho Vault #2 supplies an additional $500 to market #5.

2. Total supply on market #5 increases from $200 to $700 ($630 borrowed + $70 liquid)
3. The liquidity in market #5 increases from $20 to $70

Next, the crucial part:

1. Liquidity available in Morpho Vault #1 increases 50% from $100 to $150 because Morpho Vault #2 supplied to market #5.
2. Liquidity available in Morpho Vault #2 is $80 rather than $60 or 33% higher than it would be if Morpho Vault #1 was not supplying on market #5.

This effect is called 'liquidity amplification’, which occurs when multiple vaults share liquidity from the same markets. The benefits of liquidity amplification grow with the number of vaults, leading to greater liquidity, efficiency, and scalability.

Security

Morpho is known for its industry-leading security practices and follows a multi-faceted approach to security.

Morpho security practices include formal verification, mutation tests, fuzzing, unit testing, and peer reviews that can be found within respective Github repositories. External measures include professional security reviews, contests, and pre/post-deployment bounties.

A whole article was dedicated to the Morpho Security Framework here.

Over time, Morpho has been audited 27 times by 12 different security firms. This covers Morpho Optimizers, Front-ends, Morpho Vaults and Morpho, and to our knowledge, makes Morpho the most audited project in the world. Feel free to refer to the following 3 sections for more details about respective security reviews:

- Morpho
- Morpho Vaults
- Morpho Optimizers

Morpho Optimizers is renowned for achieving the highest score (98%) out of 300+ protocols on DeFiSafety, an independent rating agency that assesses how closely a protocol follows best practices.

More information on specific security practices is shared in the following sections.

Security Reviews

Here is the list of all security reviews conducted on Morpho Vaults' smart contracts.

The code is immutable and open-source. Multiple independent security firms and individual auditors have audited the protocol.

The following table lists all audits conducted on Morpho Vaults' smart contracts.

Smart contracts security reviews

Auditors	Scope	Date	Report
Spearbit	Morpho Periphery	2024-10-29	spearbit-report
ABDK Consulting	Morpho Periphery	2024-11-01	ABDK-report

Spearbit	Morpho Periphery	2024-11-23	cantina-managed-review
Open Zeppelin	Morpho Periphery	2024-11-16	MetaMorpho Diff Audit
Cantina Contest	Morpho Periphery	2023-11 to 2023-12	cantina-competition
Spearbit	Morpho Periphery	2023-11-16	cantina-managed-review
Open Zeppelin	Morpho Periphery	2023-11-16	open-zeppelin

Formal Verification

Formal verification is the process of mathematically checking that the behavior of a system, described using a formal model, satisfies a given property. In essence, it proves the smart contract performs as it should.

Morpho Vaults' core properties have been formally verified using CVL, Certora's Verification Language. It is done by giving a high-level description of the verification and then describing the folder and file structure of the specification files.

For all the formal verification performed on Morpho Vaults to date, visit the dedicated folder in the Github repository.

The following table lists all formal verifications conducted on Morpho Vaults smart contracts.

Formal verifications

Formally Proven	Scope	Date	Tool Used
metamorpho	core logic	March 2024	Certora

Risk Documentation

Morpho is committed to use industry-leading security practices. Yet, there are still a number of risks associated with the use of Morpho Vaults that users must be aware of.

Morpho risks

Morpho Vaults are built on Morpho. Therefore, using Morpho Vaults comes with the risks associated with using Morpho. Find more info on them in the dedicated section.

Smart Contracts risks

There is an inherent risk that the Morpho Vaults smart contracts could contain a vulnerability or bug. Several security measures are employed to mitigate this risk:

- Core contracts are immutable,
- It is an open-sourced code base that avoids complexities,
- The code has been audited by OpenZeppelin and Cantina, in addition to a competitive security review
- Formal verification has been applied using Certora,

- 2 ongoing bug bounty programs:
    1. Immunefi - $1,500,000,
    2. Cantina - $1,500,000.

More details about Morpho Vaults' security and security reviews can be found in the dedicated section.

Vaults risks

Key roles within a Morpho Vault wield significant power, impacting user interests:

- The Owner has the ability to set performance fees, appoint curators and allocators, and adjust various other settings. Morpho Vaults impose a timelock on actions that may affect users' interests.
- The Curator can enable/disable markets. A timelock allows users to react to changes initiated by the curator.
- The Allocators determine markets supply/withdrawal order, influencing returns and liquidity for suppliers.
- The Guardian has the ability to revoke timelocked actions, providing an additional layer of protection for users.

When investing in a Morpho Vault, it is important to conduct thorough due diligence on the vault's settings and its allocation strategy, as well as to stay up to date with its changes.

Become a Risk curator

Setup

The vault curator can setup a few things, let's detail them one by one.

Attribute roles

Curator

Execute `setCurator(0xNewCurator)`

Allocator

Execute `setIsAllocator(0xNewAllocator, true)`. One can set the second attribute to false to revoke the allocation capacity.

Guardian

Note that if no guardian is set, (which is the case at vault creation), submitting a new guardian sets it immediately as the guardian. Execute `submitGuardian(0xNewGuardian)`

Fee

If one wants to set the performance fees:

1. Execute `setFeeRecipient(0xNewFeeRecipient)`
2. Execute `setFee(10000000000000000)` // 1%, FEE variable has 18 decimals.

Skim recipient

The skim recipient will receive all rewards that may have been allocated to the vault, while the latter earned them by allocating liquidity on Morpho markets.

Execute `setSkimRecipient(0xNewSkimRecipient)`

Set Market Caps

The process to set caps is:

1. `submitCap`
2. `acceptCap`

Let's detail them:

1 - Submit the cap

Fill the arguments of the `submitCap`. On the addresses section of the documentation, you have the markets created on Morpho, let's take as an example the market: WETH/wstETH (94.5%, ChainlinkOracle, AdaptiveCurveIRM).
Execute `submitCap` as in the picture below. Keep in mind that te `newSupplyCap` is in decimals of the asset (loan token).
Submit Cap.

2 - Accept the cap

Wait for timelock to elapse and execute `acceptCap()`

Execute the `acceptCap()` with the market parameters of the market, which in this case are retrievable thanks to the `id`of the market:
`0xc54d7acf14de29e0e5527cabd7a576506870346a78a11a6762e2cca66322ec41`

How to get the id from a market?

Morpho Labs team may provide an annex contract, in the meantime, one can retrieve the id from markets either:

- On the addresses section of the documentation, on the column `id`,
- Or at market creation as the event is emitting the `id`, and the markets params alongside as stated in this section,
- Or via this TypeScript script.

Accept Cap.

Do this for as many markets as you expect the vault to deposit liquidity.

In the end, do not forget to submit and accept the cap for what we call the "idle" market.

The idle market is a Morpho market with the asset of the Morpho Vault as `loanToken`, the `address(0)` as the `collateralToken`, the `address(0)` as `irm`, the `address(0)` as `oracle` and `0` as `lltv` so that it is possible to supply on this market, but not to borrow. You might have to create a such market (with the `createMarket` function of the Morpho contract) if none of the already created markets of Morpho matches these parameters.

The cap for this market is supposed to be `type(uint184).max`, to always allow deposits into the vault.

3 - Set the `supplyQueue`

Now one has to set the `supplyQueue` accordingly via the `setSupplyQueue` function, by giving as input a list of Ids of markets as follows:
[0xc54d7acf14de29e0e5527cabd7a576506870346a78a11a6762e2cca66322ec41,0x_MARKET_ID_2,...]

And execute with an address that has the `onlyAllocatorRole`:

- Owner,

- Curator,
- Allocator.

Idle market

It is recommended to put the idle market as the end of the `supplyQueue` such that any extra deposit after all caps are reached will go into this market.
For the `withdrawQueue` (see next section), the idle market should be set as the first one.

4 - Deposit liquidity

It is important to deposit a bit of liquidity (can be even 1$) such that the native rate of the vault is not 0 if the markets where liquidity flows is not 0. Any interface, scripts, or bots reading the market APYs will thus have a rate different than 0.

Manage exposure

Be sure to have read the previous section and that you are aware of the different functions behavior.

Remove a market

To force remove a market, one can call `submitMarketRemoval`.

Warning

- Submits a forced market removal from the vault, eventually losing all funds supplied to the market.
- Note that funds can be recovered by enabling this market again and withdrawing from it (using `reallocate`), but funds will be distributed pro-rata to the shares at the time of withdrawal, not at the time of removal.
- This forced removal is expected to be used as an emergency process in case a market constantly reverts.
- To softly remove a sane market, the curator role is expected to bundle a reallocation that empties the market first (using `reallocate`), followed by the removal of the market (using `updateWithdrawQueue`).
- Reverts for non-zero cap or if there is a pending cap. Successfully submitting a zero cap wil prevent such reverts.

Gating a Morpho Vault

This tutorial explains how to gate a Morpho Vault by controlling access to deposits through the supply queue.

Prerequisites

- You have the necessary permissions (onlyAllocatorRole) to call `setSupplyQueue` functions on the vault contract. The users allowed to do so are: the `owner`, the `curator`, and any `allocator`.

- You already have approved the Morpho Vault to spend the ERC20-compliant token. Otherwise, you will have to add it into the batch of operations.
- You have the ID of the market you want to allow deposits into.
- The supply queue is empty.

Optional: Approving ERC20 Token Spending (if not done previously)

If you haven't already approved the Morpho Vault to spend your ERC20 tokens, you'll need to add this step to your batch transaction before the deposit. Here's how to do it:

1. Approve token spending
   Call the `approve` function on the ERC20 token contract:
2. *function* approve(address spender, uint256 amount) *external returns* (bool);
3. Example:
4. IERC20(tokenAddress).approve(vaultAddress, amountToDeposit);
5. Make sure to replace `tokenAddress` with the address of the ERC20 token you're depositing, `vaultAddress` with the address of the Morpho Vault, and `amountToDeposit` with the amount you want to deposit.
6. Add to batch transaction
   Include this approval step at the beginning of your batch transaction, before the `setSupplyQueue` and `deposit` calls.

Remember that approvals are typically persistent, so you only need to do this once for each token-vault pair unless you've set a specific allowance that has been exhausted.

Roles & Capabilities

This section describes the different roles and capabilities of the Morpho Vaults Contract.

Owner

- Only a single address can have this role.
- Capabilities:
    - Change owner (2 steps: the new owner has to accept ownership).
    - Renounce ownership.
    - Set the curator.
    - Add/remove addresses with the allocator role (including the Public Allocator).
    - [Time-locked] Set the guardian.
    - Increase the timelock duration for every time-locked function.
    - [Time-locked if already set] Decrease the timelock duration for every time-locked function.
    - Set the performance fee.
    - Set the fee recipient.
    - Set the rewards distributor address.
    - All the capabilities of the Curator, the Allocator and the Guardian.

Curator

- Only a single address can have this role.
- Capabilities:
    - Decrease a supply cap on a Morpho market.
    - [Time-locked] Increase a supply cap on a Morpho market, which includes enabling a new market (by setting a non-zero cap on a not yet enabled market).
    - [Time-locked] Submit the forced removal of a market.
    - Revoke the pending supply cap on a Morpho market.
    - Revoke the pending removal of a Morpho market.
    - All the capabilities of the Allocator.
- Note: the Curator can’t pause the withdrawal of funds.

Allocator

- Multiple addresses can have this role.
- Capabilities:
    - Can modify the allocation between markets and the idle supply in the vault within the bounds set by the Curator.
    - Set the supply queue to some arbitrary queue of markets.
    - Re-order the withdraw queue by applying a permutation to it. Can omit markets on which the vault has 0 supply and 0 cap to remove it form the withdraw queue.

Guardian

- Only a single address can have this role.
- Capabilities:

    - Can revoke a pending timelock decrease until the previous timelock ends and the new timelock is accepted (by the Owner).
    - Can revoke a pending guardian until the timelock ends and the new guardian is accepted (by the Owner).
    - Can revoke each pending market cap increase until the timelock ends and the new market cap is accepted (by the Owner or the Curator).
    - In particular, it cannot revoke a pending fee (submitted by the Owner).

Any address

- Can accept the new cap after timelock.
- Can accept the new fee after timelock.
- Can accept the new guardian after timelock.
- Can accept the new Timelock value after the current timelock duration.

Vaults as asset

In this section, we outline security considerations and recommendations for listing 4626 vaults as an asset, collateral, or loan asset.

Vault as collateral

Liquid & flashloanable 4626 assets with share price decrease

ERC4626 vaults with the following properties can incur a loss of funds.

- Most of the shares of the vault can be flash loaned.
- Most of the vault is liquid.
- The vault's share price can decrease notably and instantly.

This is not specific to Morpho but a general statement for all ERC4626 vaults. Since assets deposited on Morpho can always be flash loaned, it is important to keep in mind.

The loss is bounded by:

$$\text{totalLoss} \leq \delta * (1 / 1 - \min(AW/A, SL/S)$$

where:

- δ
- δ is the loss on the vault (that creates the share price decrease)
- AW/A is the "liquid portion of the vault"
- SL/S
- is the "loanable portion of the vault"

Recommendations for Morpho Vaults used as collateral

For Morpho Vaults with bad debt realization (created with [the MetaMorpho Factory v1.0]()), the share price can decrease in the event of a bad debt realization in one of the listed markets. If the vault is highly liquid, the supply queue has a specific order, and most of the shares can be flashloaned, then the liquidator of the market could amplify bad debt realized by the vault. Since using vaults as collateral increases the proportion of the vault that can be flashloaned, it could favor this amplification. Although very unlikely, it is thus not recommended using those vaults as collateral.

For Morpho vaults without bad debt realization (created with the MetaMorpho Factory v1.1), the share price of the vault can't decrease, and the above-described scenario can't happen. That said, the vault users won't have bad debt realized in their vault.

Credit to 100proof on helping the Morpho team providing those security guidelines to vault curators.

Vault as loan asset

Pricing method

One of the main consideration when onboarding an asset is what price it should have. A natural way to price vault shares is by using the exchange rate (supported by the Morpho Oracle V2). We summarise here the risks associated with using this pricing method and recommendations when doing so.

Manipulations & share price changes

General manipulations of an ERC4626 are detailed in an Euler article. Additionally, we should take into account the other possible ways that the share price of a vault can change.

We have in the general case:

1. donations in general to ERC4626 vaults can cause a sudden price increase. Vaults can have mitigations to this, but most are vulnerable. Morpho Vaults can be affected because of supply on behalf.

2. rounding errors (including stealth donations) can change the share price. They are not expected to be significant, including for Morpho Vaults when the vault has a large enough total assets.

And the ones specific to each vaults. For example in Morpho vaults:

3. “economic”, known scenarios could decrease the share price on Morpho Vaults. They should be accepted by the user for what they are, so they are not considered as blocking. It includes bad debt realization for Morpho Vaults v1.0 (note that public reallocation can be an aggravating factor in this scenario) and forced market removal.

Conclusions

The price change by donation causes a price increase, so it naturally is an issue for listing a vault as a loan asset. In particular, it leads to:

- the possibility of a future attack like the Cream hack, as soon as the position of those shares is priced
- liquidations having an extra incentive that is extremely difficult to mitigate, resulting in borrowers taking unpredictable risks (or not joining at all).

Because of those risks, it is not recommended to list any ERC4626 vault as a loan asset in a Morpho market at the moment.

Emergency

An enabled market is now considered unsafe

If an enabled market is considered unsafe (e.g. risk too high), the curator/owner may want to disable this market in the following way:

1. Revoke the pending cap of the market with the `revokePendingCap` function (this can also be done by the guardian).
2. Set the cap of the market to 0 with the `submitCap` function. To ensure that submit cap does not revert because of a pending cap, it is recommended to batch the two previous transactions, for example using the multicall function of Morpho Vaults.
3. Withdraw all the supply of this market with the `reallocate` function. If there is not enough liquidity on the market, remove the maximum available liquidity with the `reallocate` function, then put the market at the beginning of the withdraw queue (with the `updateWithdrawQueue` function).
4. Once all the supply has been removed from the market, the market can be removed from the withdraw queue with the `updateWithdrawQueue` function.

An enabled market reverts

If an enabled market starts reverting, many of the vault functions would revert as well (because of the call to `totalAssets`). To turn the vault back to an operating state, the market must be forced removed by the owner/curator, who should follow these steps:

1. Revoke the pending cap of the market with the `revokePendingCap` function (this can also be done by the guardian).
2. Set the cap of the market to 0 with the `submitCap` function. To ensure that submit cap does not revert because of a pending cap, it is recommended to

batch the two previous transactions, for example using the multicall function of Morpho Vaults.

3. Submit a removal of the market with the `submitMarketRemoval` function.
4. Wait for the timelock to elapse
5. Once the timelock has elapsed, the market can be removed from the withdraw queue with the `updateWithdrawQueue` function.

Warning: Funds supplied in forced removed markets will be lost, this is why only markets expected to always revert should be disabled this way (because funds supplied in such markets can be considered lost anyway).

Curator takeover

If the curator starts to submit positive caps for unsafe markets that are not in line with the vault risk strategy, the owner of the vault can:

1. Set a new curator with the `setCurator` function.
2. Revoke the pending caps submitted by the curator (this can also be done by the guardian or the new curator).
3. If the curator had the time to accept a cap (because `timelock` has elapsed before the guardian or the owner had time to act), the owner (or the new curator) must disable the unsafe market (see above).

Allocator takeover

If one of the allocators starts setting the withdraw queue and/or supply queue that are not in line with the vault risk strategy, or incoherently reallocating the funds, the owner of the vault should:

1. Deprive the faulty allocator from his privileges with the `setIsAllocator` function.
2. Reallocate the funds in a way consistent with the vault risk strategy with the `reallocate` function (this can also be done by the curator or the other allocators).
3. Set a new withdraw queue that is in line with the vault risk strategy with the `updateWithdrawQueue` function (this can also be done by the curator or the other allocators).
4. Set a new supply queue that is in line with the vault risk strategy with the `setSupplyQueue` function (this can also be done by the curator or the other allocators).

Morpho Vaults

Overview

- Morpho Vaults are ERC4626 compliant vaults with permit (ERC-2612).
- One Morpho Vault is dedicated to one loan asset (e.g USDC) that users can supply or withdraw at any time depending on the available liquidity.
- Permissionless, free, and immutable onchain factory contract, deploying immutable onchain instances.
- A maximum of 30-ish markets can be listed on a given vault.

- The vault manager can extract a performance fee (up to 50%) from the total interest generated.
- Each market has a supply cap configured onchain that guarantees lenders a maximum absolute exposure to the specific market.
- Allocation of the vault’s liquidity can be done in multiple ways:
  - Manual Allocation: Asynchronously via a permissioned function, specifying how much liquidity is withdrawn from which markets (up to their respective available liquidity) and how much liquidity is then supplied to each market (up to their respective caps).
    The gas cost of this operation is paid by the caller of the function, which is restricted and is expected to be under the vault manager’s responsibility.
  - Supply queue: Atomically upon each deposit by specifying an order in which markets will be supplied to (up to their respective caps).
    The remaining funds after having reached all market caps of the supply queue are always left in the vault, as idle liquidity.
    The gas cost of iterating through the supply queue and supplying to markets is paid by depositors.
    The supply queue can be empty to disable this behavior and save gas for depositors, in which case all funds deposited are left idle in the vault and must be supplied to markets asynchronously to generate interest.
  - Withdraw queue: Atomically upon each withdrawal by specifying an order in which markets are withdrawn from (up to their respective liquidity).
    The funds are always taken first from the idle liquidity left in the vault.
    The gas cost of iterating through the withdraw queue and withdrawing from markets is paid by depositors.
    The withdraw queue must always contain each market that has either a non-zero cap or some liquidity supplied, to guarantee lenders they can withdraw from the vault up to each market’s liquidity at any given time.

If a market is broken or forever illiquid, the allocator can put the market at the end of the withdraw queue.

Bundlers

What are Bundlers?

Bundlers are [audited]() smart contracts that let users combine multiple actions into a single transaction—making interactions with Morpho and other protocols more efficient. They help reduce waiting times between separate transactions and can lower gas costs by using a `multicall` function to batch various actions at once.

Bundler3

Bundler3 is the latest and most versatile version, offering additional features like:

- Callbacks and approvals during a single atomic transaction.
- Adapters for advanced integrations and custom logic.

Morpho Bundlers

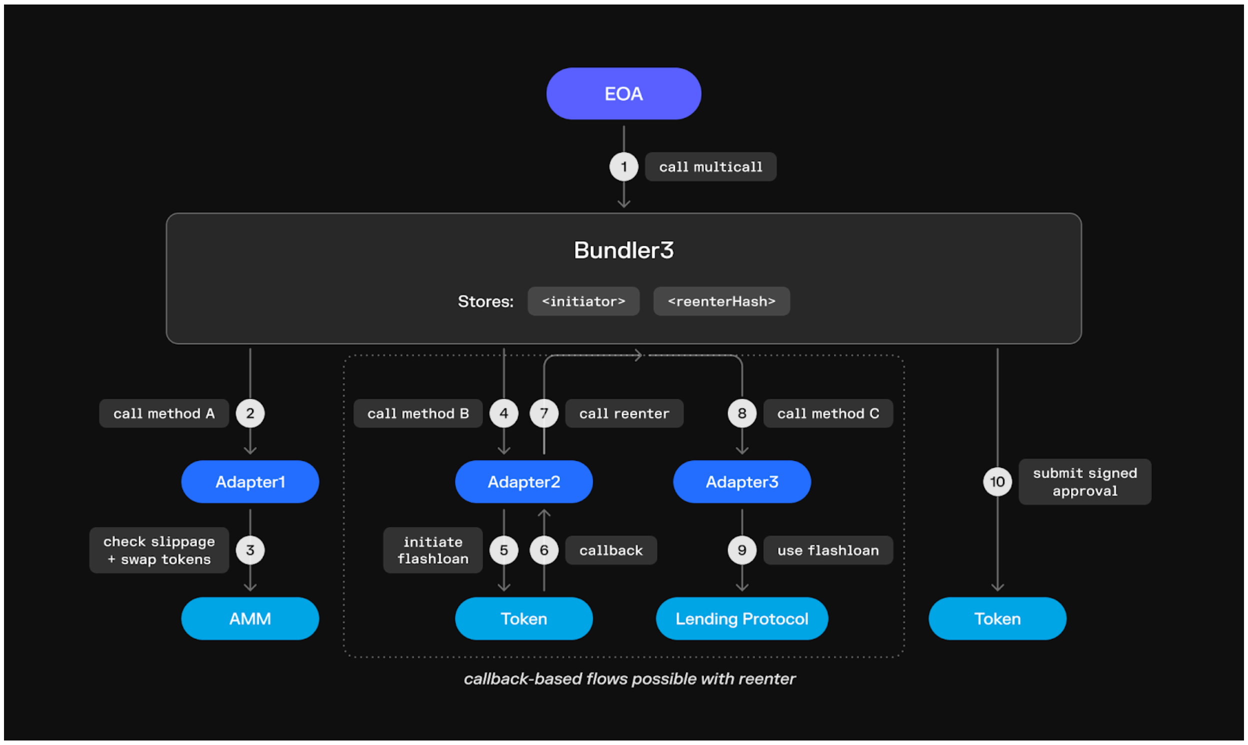
What are they?

Morpho Bundlers allow externally owned accounts (EOAs) to bundle multiple operations into a single transaction. This reduces both the time needed between steps and overall gas costs. A bundler exposes a `multicall` function where each call can trigger different on-chain actions in one go.

Bundler3

Bundler3 is the newest release, designed for complex DeFi use cases. It supports:

- Atomic checks, approvals, and callbacks in a single transaction.
- Adapter-based integrations (e.g. token swaps, bridging, slippage checks).



A core innovation in Bundler3 is the introduction of adapters, specialized “wrappers” that manage DeFi interactions like transfers, slippage checks, or token approvals in a safe, atomic manner. Out of the box, Bundler3 comes with:

- GeneralAdapter1 — Covers common operations (e.g. ERC20 transfers, wrap/unwrap, ERC4626 vault interactions, Morpho actions).

- EthereumGeneralAdapter1 — Extends the general adapter with Ethereum-specific features (e.g. staking ETH, wstETH).
- ParaswapAdapter — Integrates with ParaSwap to enable swapping tokens (buy/sell flows) within a Bundler3 transaction.

All of these adapters inherit from a CoreAdapter contract that ensures only Bundler3 can call them, preserving user approvals and preventing malicious usage. If an adapter needs to perform additional steps—like a flash-loan scenario—it can call back into Bundler3 via the `reenter` function, all within the same atomic transaction.

Earlier versions like `MorphoBundler`, `PermitBundler`, and more specialized ones (e.g. `WstEthBundler`, `ERC4626Bundler`) remain available for simpler use cases. However, Bundler3 is recommended for its flexibility and robust security model.

Security

Morpho employs a rigorous security approach, including:

- Formal verification, continuous testing, and peer reviews.
- Regular professional audits by leading firms (see security reviews).
- Bug bounties and post-deployment monitoring.

Bundler3 follows these same high standards, and its code is publicly available for transparency and community inspection.

Bundler2

Prior to Bundler3, Morpho offered several bundlers tailored to specific needs, such as:

- MorphoBundler (for core Morpho interactions)
- TransferBundler (transferring ETH or ERC20 tokens)
- Permit(2)Bundler (token approvals using EIP-2612 or Permit2)
- WstEthBundler (Lido-related staking and wrapping)
- ERC4626Bundler (interactions with ERC4626 vaults)
- UrdBundler (claiming rewards via a UniversalRewardsDistributor)
- WNativeBundler (wrapping/unwrapping the chain’s native token)

These can still be used as standalone or in conjunction with each other. For newer projects or advanced functionality, Bundler3 is the recommended upgrade.

Understanding and Implementing Bundler3

and Adapters in

Solidity

This tutorial walks you through the implementation and usage of Morpho's Bundler3 contract along with GeneralAdapter1 (one of the adapters taken as example), explaining key concepts and functionalities.

1. Overview

`Bundler3` is Morpho's latest multicall contract, allowing you to batch multiple operations in a single transaction. Unlike simpler multicall implementations, `Bundler3` supports callbacks, authorizations, and slippage checks (via adapters), making it more powerful and flexible for advanced DeFi scenarios.

`GeneralAdapter1` is a chain-agnostic adapter that integrates `Bundler3` with various protocols and token standards, including:

- ERC20 wrappers
- ERC4626 vaults
- Morpho protocol interactions
- Permit2 transfers

- Wrapped native token operations

By combining `Bundler3` with `GeneralAdapter1`, you can create seamless one-click flows—e.g., wrapping tokens, supplying to Morpho, repaying debt, or migrating positions—all within a single atomic transaction.

---

4. Understanding Bundler3's Core Features

4.1 Transient Storage & Callbacks

`initiator` (transient) ensures adapters know the original EOA interacting, so that approvals and ownership checks can be enforced. `reenterHash` ensures that any nested call (like a flash loan callback) is pre-authorized by the original transaction data. This prevents malicious reentry with altered parameters.

4.2 Atomic Multicalls

`Bundler3`'s `multicall` allows you to sequence calls in a single transaction—complete with fallback logic. For example, you can do:

1. Wrap ETH → WETH
2. Approve WETH → Morpho
3. Supply WETH as collateral
4. Borrow DAI

5. Swap DAI for USDC
6. Repay an existing USDC debt

All in one shot, with each step optionally rolling back if an intermediate check fails—unless `skipRevert` is set to true.

5. Security Considerations

5.1 Reentrancy Protection

`Bundler3` uses a `reenterHash` mechanism to enforce that only the expected callback can reenter. `Adapters` use `onlyBundler3` to block unauthorized direct calls.

5.2 Slippage & Max Amounts

Many adapter functions accept `type(uint).max` to mean "use the entire balance." Slippage checks (e.g. `require(suppliedAssets.rDivUp(...) <= maxSharePriceE27)`) ensure you don't supply or borrow at an unfavorable rate.

5.3 Approvals

By default, `Bundler3` is "unowned" and shouldn't hold any permanent user approvals. `Adapters` handle approvals as needed, typically forceApproveing tokens only at the moment of the operation.

5.4 Callback Hash

If you’re building more advanced flows (like a flash loan from Morpho), make sure your `callbackHash` is set correctly in the call struct. A mismatch will cause the transaction to revert for security reasons.

6. Best Practices

6.1 Validate Inputs Thoroughly

require(receiver != address(0), ErrorsLib.ZeroAddress());
require(assets != 0, ErrorsLib.ZeroAmount());

6.2 Use Slippage Checks

Always set `maxSharePriceE27` or `minSharePriceE27` to guard against unexpected prices.

6.3 Careful with Call Ordering

If you rely on newly wrapped tokens in a second call, ensure your first call does that wrapping.

6.4 Don’t Expose Bundler3 to Unlimited Approvals

The recommended approach is to pass tokens to an adapter (which then uses the bundler’s initiator context), not to approve `Bundler3` permanently.

6.5 Plan for Re-Entrant Calls

If a protocol calls back into your adapter (e.g. after flash-loan funds are delivered), ensure you understand the reenter flow and set the correct `callbackHash`.

7. Next Steps

Check out additional adapters like `EthereumGeneralAdapter1` (handles stETH, wstETH), or `ParaswapAdapter` for token swaps. Explore the Migration Adapters for seamless position transfers (e.g. from Aave or Compound to Morpho). Integrate advanced flows, such as flash loans, by leveraging the `onMorphoFlashLoan` callback in `GeneralAdapter1`. For deeper references, see the `Bundler3` repository and examine the unit tests under `./test` to see real usage examples.

By understanding these concepts and carefully structuring your calls, you can build efficient and secure DeFi flows using `Bundler3` and its adapters.

Public Allocator

What is the Public Allocator?

The Public Allocator is an audited smart contract that allows anyone to reallocate liquidity to specific markets.

Curators of Morpho Vaults provide this contract with the role of [`allocator`](). This helps borrowers get the liquidity they need from a market by reallocating funds, making borrowing more efficient and increasing available liquidity.

The Public Allocator is natively integrated in [the Morpho Interface]().

Here is a video that explains how the Public Allocator works:

PublicAllocator

Overview

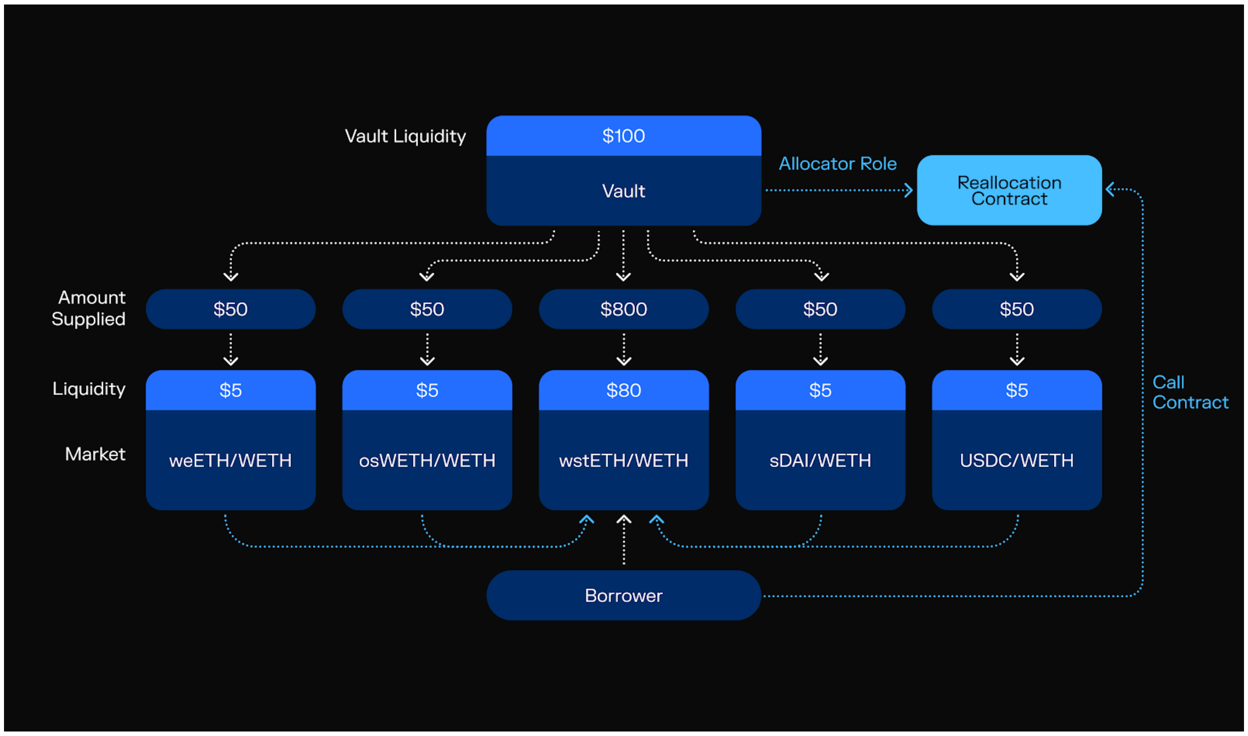
Similarly to withdrawable liquidity, when borrowing on a Morpho market, at first glance, liquidity is fragmented.

This fragmentation is not a problem for lenders thanks to Morpho Vaults as explained in the shared liquidity section. However, Morpho Vaults do not solve the problem for borrowers.

In the example above the borrower can only access $20.

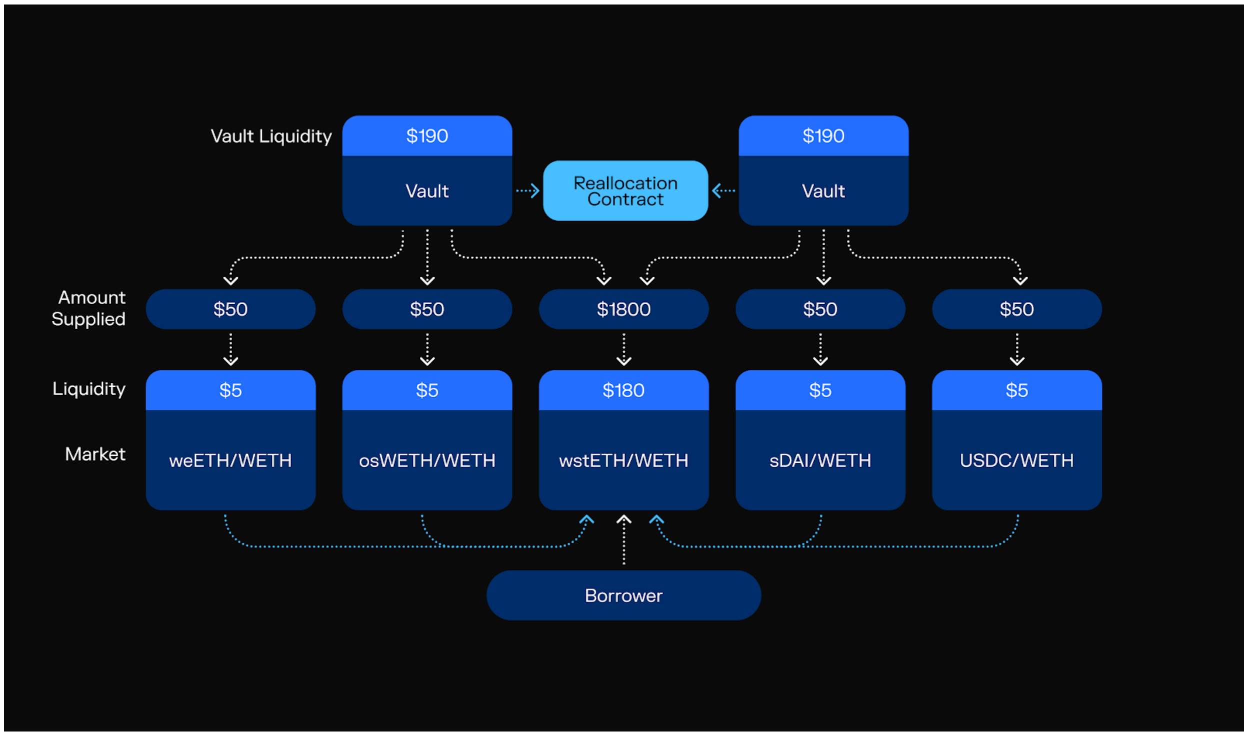
This is where the `PublicAllocator` contract comes in. With this contract, Morpho Vaults can define how liquidity can flow between their markets to increase the instantly borrowable liquidity on markets. By giving the Allocator role to this `PublicAllocator` contract, anyone can reallocate liquidity of the vault up to the bounds set by the curator.

As such, a borrower can amplify the liquidity in one market by calling the reallocation contract. Of course, the extra call to the `PublicAllocator` is abstracted in the front end.



A borrower on Morpho using the Reallocation contract of Morpho Vaults experiences the same liquidity as in a multi asset pool like Aave.

Now, as for withdrawable-liquidity, let’s project what happens as soon as you have multiple vaults enabling the reallocation contract.



Borrowers can reallocate liquidity from all the vaults supplying into this market, amplifying the borrowable-liquidity to levels that were not possible before in multi-asset lending pools like Aave.

Set Up as Allocator

Overview

As the vault curator of a Morpho Vault, you can designate the `PublicAllocator` address as an allocator of the vault. This will allow borrowers or Morpho's interface to trigger reallocations, amplifying the available liquidity for end users while limiting the flow of tokens that can be supplied or withdrawn from a market listed in the vault.

Prerequisites

Jump on the Morpho Vault contract you are curating and follow the instructions.

As a `curator` of your own vault, provide the `allocator` role to the `PublicAllocator` contract as described in this section, via the `setIsAllocator` function (0xNewAllocator, true).

Providing the `allocator` role to the `PublicAllocator` contract on the Morpho Vault contract
Now you can jump on the `PublicAllocator` contract to set it up.

Setting an admin

By default, only the owner of the vault can configure the `PublicAllocator` for a given vault. However, the owner can appoint an additional admin by triggering the `setAdmin` function.

Setting the fee

The `PublicAllocator` has a fee that is charged on every reallocation. The fee is paid in ETH and can be changed at any time by the owner of the vault or the admin by triggering the `setFee` function.

This fee prevents griefing attacks and helps the vault curators to cover the gas costs reallocations in the case the reallocation through the `PublicAllocator` has created an imbalance.

Setting the flow caps

Flow caps determine the maximum amount of tokens that can be supplied or withdrawn from a market listed in the vault. The flow caps can be set by the owner of the vault or the admin for a specific market by triggering the `setFlowCap` function.

A flow cap is composed of the `maxIn` and `maxOut` in this order. The `maxIn` is the maximum amount of tokens that can be supplied to a market and the `maxOut` is the maximum amount of tokens that can be withdrawn from a market.

For example, if amount `x` is withdrawn from a market during reallocation, the market's `maxIn` increases by `x`, while `maxOut` decreases by `x`. Conversely, if `x` is supplied to the market, `maxIn` decreases by `x` and `maxOut` increases by `x`.

This mechanism helps maintain balance and limits excessive token flows in and out of the market.

Public Allocator

Overview

The `PublicAllocator` contract is a publicly callable allocator for Morpho Vaults. It allows for the efficient reallocation of assets within the vault, enabling borrowers or Morpho's interface to trigger reallocations and amplify the available liquidity for end users while ensuring the flow caps are respected.

Code

Public Allocator Github repository

Rewards

What are Rewards?

Rewards are incentives provided by the Morpho DAO, market creators and/or curators to users, encouraging them to interact with Morpho markets.

The claimability of rewards earned through Morpho's rewards programs is based on an epoch system. When an epoch ends, the rewards accumulated during the epoch become claimable. Currently, an epoch lasts 2 weeks, this period is due to be shortened in the near future. There is no deadline to claim the rewards earned.

Below is an image of app.morpho.org showing the rewards earn-able on a given Morpho Vault. Users might also earn rewards by borrowing on some Morpho markets.

Overview

Guides

- Claim your rewards

- Incentivize Morpho Markets

Core concepts

- Programs: Strategies used to compute rewards.
    - Market programs: Incentivize market supply, borrow, and/or collateral using a linear distribution over a timeline.
    - Vault programs: Incentivize vault supply using a linear distribution over a timeline.
    - Uniform rate programs: Apply a uniform reward rate for each dollar supplied in a curated market.
- Computation: An offchain mechanism to attribute rewards.
    - Source of data: Subgraph Data used to compute rewards.
    - Data manipulation (Blacklisting and reallocation): offchain tools to perform manipulation on data to provide useful features.
    - Offchain script: offchain script soon open sourced to attribute rewards.
- Distribution: An onchain mechanism to distribute rewards.
    - Universal Reward Distributor (URD): A contract used to claim rewards.

Useful link

What are Programs?

A reward program defines a strategy to incentivize Morpho users.

- This section describes the process to create a reward program.
- You can access to all published program using rewards.morpho.org/v1/programs

Market programs

A market program incentivize supply and/or borrow and/or collateral using a linear distribution during a timeline. It is set for Morpho users.

Vault programs

A vault program incentivise vault supply using a linear distribution during a timeline. It is set at the level of the Morpho Vaults users.

Uniform rate programs

The following information are retrievable in [this forum post].

A uniform rate program applies a consistent reward rate for each dollar supplied in a curated market.

This uniform rate will be determined by the formula:

The formula ensures that each new dollar of supply is rewarded at a consistent MORPHO rate, up to a predetermined limit to cap the total daily distribution amount.

Below is a graphical representation of the uniform rate rewards:

can be adjusted by governance as the protocol expands, and customized for each chain where Morpho is deployed. The values and scope can be monitored on the [New scalable rewards model forum thread].

Source of data

To compute rewards, we need user participation data over time within the protocol. We obtain this data using the following subgraph: Morpho Points Mainnet Subgraph.

Data manipulation

To provide advanced features on top of the programs, we need to perform data manipulation. This manipulation is carried out using offchain tools (which will be open-sourced soon).

We have implemented two key features: forwarding and blacklisting.

Forwarding

This feature is used to distribute Market programs. Since we have vaults on top of Morpho to supply in markets, without reallocation, rewards would be distributed to vaults. By implementing reallocation, we reallocate vault positions to vault suppliers, allowing us to directly distribute market rewards to the vault suppliers.

Blacklisting:

This feature removes the positions of blacklisted users. It allows protocols that bootstrap their markets to avoid incentivizing themselves.

Distribution

The distribution is done by a distributor which defines the strategy to distribute rewards to users. Actually we are using one type of distributor which is the Universal Rewards Distributor (URD).

The distribution is done manually around every month.

You can access to a user distributed rewards using:

`https://rewards.morpho.org/v1/users/[0xUser1]/distributions`

Universal Rewards Distributor

The Universal Rewards Distributor (URD), is a smart contract allowing the distribution of multiple ERC20 tokens from a single offchain computed Merkle tree.

Full repository here.

Each URD contract has an owner and a group of updaters (chosen by the owner). Values submitted by updaters are timelocked and can be revoked by the owner or overriden by another updater. However, this timelock can be set to 0 if the URD owner does not need it.

Use Case Example

Assume the Owner is a DAO with a periodic rewards mechanism. Each month, a Gelato bot runs a script to create a Merkle tree that distributes TokenA and TokenB.

During the setup, the DAO configures the timelock based on the risk of updater corruption, let's say 3 days, and adds a Gelato bot as an updater. If the DAO already has a distribution at the time of the URD deployment, it can define an initial root, or use an empty root if not.

Each month, the Gelato bot proposes a new root. For 3 days, the DAO has the opportunity to run checks on this root. After these 3 days, if the DAO did not revoke the root, anyone can accept this value.

The DAO must transfer the correct amount of tokens to the URD to allow all claimants to claim their rewards. If the DAO does not provide enough funds, the erc20 transfer will fail.

Attaching an IPFS Hash

- Using a Merkle tree delegates all root computation offchain, leaving no information about the tree onchain (except for the root). This makes it challenging for an integrator (or a claimer) to understand the Merkle tree or to know which proof to use. To address this, each root can be linked to an IPFS hash, which can link to any data. We recommend that all users follow the same format for the IPFS hash to facilitate integration. The suggested format is as follows:
- We recommend not including spaces or new lines when uploading to IPFS, to ensure a consistent method of uploading your JSON file.
- We also recommend sorting the tree by the account address and the reward token address, to ensure a consistent order.

Owner Specifications

- The URD is an owner-managed contract, meaning the owner has full control over its distribution. Specifically, the owner can bypass all timelocked functions, modify the timelock, add or remove updaters, and revoke the pending value at their discretion.

- If the owner is set to the zero address, the contract becomes ownerless. In this scenario, only updaters can submit a root. Furthermore, the `timelock` value becomes unchangeable, and pending values can still be overridden by updaters.
- If there are neither owner nor updaters, the URD becomes trustless. This is useful for creating a one-time distribution that must remain unchanged. This can be accomplished at the contract's creation by providing only a Merkle root and an optional IPFS hash. After this, the contract's sole functionality is to claim rewards.
- It is possible to create a URD with no root, owner, or updaters. While this might seem pointless, it is the URD creator's responsibility to configure the URD correctly.

Updaters Specifications

- Multiple updaters can be defined for a single URD. The owner can add or remove updaters at any time, instantly, without a timelock.
- All updaters share a single pending value. This means they can override the pending value (if any) at any time.
- If a pending value is not yet used as the main root, any submissions by updaters will override the pending value.

Having multiple updaters can lead to situations where the pending values are subject to multiple concurrent propositions. The owner must manage this scenario to maintain the URD's functionality. We recommend not having different reward distribution strategies in a single URD. All updaters should agree on a distribution mechanism and a root. The use case for having multiple updaters is to enhance resilience and security.

Considerations for Merkle Tree

- The claimable amount for a given user must always exceed the amount provided in the previous Merkle tree. If a claimer has claimed an amount higher than the `claimable` amount in the Merkle tree (if claimed from a previous root), the transaction will revert with an underflow error.
- We recommend merging all the (reward, user) pairs into a single leaf. If you wish to have two different leaves for one (reward, user) pair, the user will be able to claim the larger amount from the two leaves, not the sum of the two.
- Merkle trees can be generated with Openzeppelin library.
- The leaves of the merkle tree have to be hashed twice. It is natively supported by the Openzeppelin library.
- The URD supports empty root. This means that, at any time, updaters or owner can submit a 0 hash root. It can be used to deprecate the URD.

Claim Rewards

- The arguments for the claim function must be provided off-chain. If an IPFS hash is given, the IPFS content likely includes the parameters.
- Anybody can make a claim on behalf of someone else. This means you can claim for multiple users in one transaction by using a multicall.
- When the root is updated the proof will change even if the claimable amount of the user is not updated.
- A call to the claim function will claim only one token. If you wish to claim multiple tokens, use a multicall.

The factory

You can create a URD using a factory. This factory simplifies the indexing of the URD offchain and validates any URDs created through it. While its use is optional, it offers a convenient alternative to directly deploying a URD by submitting the bytecode.

Skim non claimed rewards

A rewards program can have a deadline for users to claim tokens. After this deadline, the owner can skim the rewards that were not claimed. To do so, the owner has to create a Merkle tree that is sending the rewards to the owner. Then, the owner can submit this Merkle tree to the URD. The URD will then allow the owner to claim the rewards that were not claimed by the users.

Limitations

The pending root is not a queue

The pending root does not have a queue mechanism. As a result, when a root updater pushes a root to the pending root in a distribution with a timelock, it effectively erases the previous root. This implies that a compromised root updater can continuously suggest a root, resetting the timelock of the pending root.

Furthermore, if the pending root is ready to be accepted but a root updater suggests a new root simultaneously, the pending root is erased and the timelock restarts. This situation can lead to endless loops of the pending root, especially if an automation mechanism suggests a root at intervals shorter than the timelock.

This behavior is acknowledged and should be considered when designing a strategy using the URD. Ensure the epoch interval for updating the root is longer than the timelock, and define these parameters accordingly.

An alternative solution is to build a queue mechanism on top of the URD with a 0 timelock distribution. Here, the root updater could be a queue designed as the Delay Modifier of Zodiac.

How to Incentivize Morpho Markets or Vaults?

To incentivize users, you will need to create a program. You will find here an overview of what a program is and the various options available to you.

This Morpho reward programs repository allows you to create your own reward program by submitting a pull request.

Universal Rewards Distributors

Morpho DAO

This section covers the reasoning behind progressive and modular decentralization.

The Morpho Association believes that a progressive and modular approach to decentralization is the key to Morpho’s success. By organizing governance into various components, we can ensure that each one advances at its own pace, according to what is relevant at the time. The governance architecture must adapt to meet unique needs to navigate different stages of the protocol best.

In the early stages, having clear direction and the ability to make quick decisions is crucial. But even while operating in a not fully decentralized environment, the association prioritizes transparent communication, comprehensive protocol documentation, and open-source code. The DAO must be involved in all changes made to the protocol through a community vote triggered by a Safe 5/9.

As the first version of the protocol matures, the power of the Safe will gradually fade, transitioning first to a veto-only role, then to no possibility of acting on the protocol, leading the protocol to be fully decentralized. The multisig remains in place for the DAO to design MORPHO token distribution and initiate future protocol versions distinct from the Morpho Optimizer.

The ambition and commitment are to achieve full decentralization by 2025, with a DAO equipped with all the tools needed to fulfill its mission.

Morpho Association

This page explains the difference between the Morpho Association and contributors to the Morpho protocol.

Morpho Association

The Morpho Association is a French nonprofit entity grouping the main contributors and users of the Morpho protocol.

Its aim is to develop the Morpho protocol by funding contributors, or by facilitating access to the protocol for users and developers. In practice, the association:

- Hosts a front end to the protocol to facilitate the access of new users to Morpho: morpho.org
- Hosts technical documentation to facilitate the work of new Morpho developers: docs.morpho.org

- Holds the intellectual property of open-source codebases such as morpho-optimizers and hosts it on the morpho-org GitHub under a GPL3 license.

The association does not research or develop Morpho software, contributors such as Morpho Labs or Institut Louis Bachelier (ILB) do.

Contributors

Morpho Labs is a core contributor to the Morpho protocol.

The company researches and develops new protocols to improve Morpho Additionally, it creates various tools to enhance the functionality and capabilities of Morpho. For example, the company:

- Researches, formalizes, and proves advanced decentralized lending mechanisms.
- Develops, tests, and secures the formally described protocols.
- Develops the necessary tooling, e.g., to host a front end or to format protocol data.
- Holds code being developed for Morpho in a private repositories until it is finished. Upon completion, ownership of the code is transferred to the Association under GPL3 license.
- Does not host any front end to the Morpho protocol.

Contracts Architecture

Past & Current Infrastructure

This section focuses on the DAOs previous and current architecture, explaining precisely how governance is executed in practice and who is involved in that process.

Step 1: Initial Deployment ☑️

The Morpho Association comprises researchers, engineers, and contributors who first imagined the Morpho Protocol.

The Morpho Association deployed the foundational Multisig of the Morpho DAO. The signers include nine association members and require five signatures to approve a transaction, making it a 5/9 Multisig.

In June 2022, the association deployed the first instance of the Morpho Protocol and transferred its ownership to the 5/9 Multisig.

Step 2: Facilitate Offchain Voting ☑️

The next natural step in the evolution of Morpho's governance is to enable any MORPHO token holder to vote for the evolution of the DAO.

Snapshot, a native web3 service that provides voting interfaces for those participating in DAO, is utilized for offchain voting for Morpho Governance.

Implementing offchain votes and governance proposals still requires the 5/9 Multisig to execute the transaction, and while it is not a fully trustless process but a step in the right direction.

Step 3: Establish Emergency Operator ☑️

In DeFi, A 5/9 Multisig is considered one of the most secure setups. However, with added security comes a more laborious and time-consuming execution. This is particularly evident when a quick, non-critical execution such as pausing a market is required. Having the capability to execute such transactions puts the protocol in a better position to handle potential emergencies.

Hence, a second 3/9 Multisig, Morpho Operator, was deployed. Using Zodiac Role Modifier, the Morpho Protocol can give 3/9 Multisig an operator role, allowing it to perform emergency functions on behalf of the 5/9 Multisig:

- (Un)pause a given market.
- Enable/Disable P2P matching for a given market.
- Update non-critical matching engine parameters.

The Morpho Operator can also:

- Set the `p2pIndexCursor` for a given market.
- Transfer the fees generated by a given market to 5/9 Multisig.

Establish Emergency Operator

Step 4: Introduce Modular Decentralization and Deploy Delay ☑️

This is the current state of the DAO's architecture.

The motivation behind the next evolution of Morpho’s Governance was two parts: an opportunity to implement modular decentralization and the need for a timelock.

Ownership of core contracts was transferred from the original 5/9 Morpho DAO Multisig to a new ownerless safe, a Delay Modifer module was deployed, and an Invalidator role was given to 5/9 multisig.

This architecture allows fully fledged contracts such as Morpho AaveV2 Optimizer to be decentralized while maintaining the option to stay lean for nascent deployments and other aspects of Morpho Governance. The timelock provides users and integrators with an added layer of security.

At this point, the governance system is rather complex. The below diagram shows the architecture of the Morpho Governance:

- Utilizing a Guard Scope to eliminate signers, an ownerless Safe Multisig (Morpho V1 Admin) is the owner of Morpho core contracts:
    i. Morpho AaveV2 Optimizer
    ii. Morpho CompoundV2 Optimizer
    iii. Morpho Proxy Admin
    iv. Morpho Vaults
- Delay Modifer introduced a 24-hour timelock on transactions initiated by Morpho DAO.
- Only Morpho DAO, a 5/9 Multisig, can initiate transactions to the delay modifier. It has an Invalidator role allowing it to invalidate a transaction in the delay modifier if the transaction was proposed initially.
- Only the two modules - Delay Modifier & Role Modifier - can interact with this Morpho V1 Admin.
- Morpho Operator, a separate 3/9 Multisig designed to be more nimble, has an Operator Role allowing it to perform emergency functions as mentioned in Step 3.

Future Decentralization

This section outlines possible implementations or changes to the current governance infrastructure to introduce further decentralization.

Enable Governance Initiated Transactions ⚙️

Under the current architecture, MORPHO holders can vote on proposals, but they cannot enforce the implementation of proposals onchain. It relies on the 5/9 Multisig bridging offchain votes to the practical onchain implementation. Therefore, Multisig signatories could choose not to implement a proposal even if approved by Governance.

One solution is the Zodiac Reality Module: a governance module that enables the onchain execution of offchain decisions through a bonding mechanism. More on the Reality Module can be found here.

Another option is the Zodiac Governor Module, which enables a seamless transition from a Multisig into a DAO with onchain voting while maintaining flexibility and optionality. More on the Governor Module can be found here.

Either option allows both holders, through a vote, and the 5/9 Multisig to send transactions to the Delay Modifier to update core Morpho contracts.

Revoke Multisig Transactions ⚙️

This would be done by introducing a transaction guard on the 5/9 Multisig, preventing it from submitting transactions to the Delay Modifier. The 5/9 Multisig would maintain the Invalidator Role.

The result: only approved snapshot votes can transmit updates to the core contracts through the Delay Modifier, but the 5/9 Multisig can invalidate transactions during the timelock period.

Depreciate Multisig ⚙️

Above all, the final step in achieving complete decentralization is depreciating the 5/9 Multisig and leaving the DAO with full control of the protocol. It will likely only be implemented once the community is comfortable with DAO infrastructure and its decentralization.

The MORPHO token

MORPHO is the Morpho Protocol's governance token. The Morpho DAO — made up of MORPHO holders and delegators — governs the Morpho Protocol. The governance system uses a weighted voting system in which the number of MORPHO tokens held determines voting power.

MORPHO holders can vote on changes or improvements to the protocol, including:

- Future initiatives to grow the Morpho protocol;
- Deployment and ownership of Morpho smart contracts;
- Turning on/off the fee switch built into the Morpho smart contracts;
- Govern the DAO Treasury.

Legacy and Wrapped MORPHO

The original, now legacy, MORPHO token was deployed as an immutable contract and lacked functionality associated with onchain vote accounting. The Morpho DAO voted to create a contract to wrap the legacy tokens into wrapped MORPHO to enable onchain vote tracking functionality in MIP-75.

In addition to onchain vote tracking, using an upgradeable token makes it possible to conform to a crosschain interoperability standard in the future, minimizing friction for MORPHO holders who want to move their tokens between chains.

Although legacy MORPHO tokens can be freely converted to wrapped MORPHO 1:1 via the wrapper contract, there could be a risk that legacy tokens might be used mistakenly in external integrations, such as exchanges. To prevent this, only the wrapped MORPHO will be transferable. Users can easily wrap their existing legacy MORPHO tokens on the Morpho App.

MORPHO Token Addresses

Wrapped Token

Ticker: MORPHO

Ethereum contract address: 0x58D97B57BB95320F9a05dC918Aef65434969c2B2

Base contract address: 0xBAa5CC21fd487B8Fcc2F632f3F4E8D37262a0842

Deployed: November 10, 2024

Maximum Supply: 1,000,000,000 MORPHO

Legacy Token

Ticker: MORPHO

Contract Address: 0x9994E35Db50125E0DF82e4c2dde62496CE330999

Deployed: 24 June 2022

Maximum Supply: 1,000,000,000 MORPHO

Wrapper contract

Address: 0x9D03bb2092270648d7480049d0E58d2FcF0E5123 (only on Ethereum)

Token Distribution & Vesting

The overall distribution of MORPHO tokens, including vested and unvested allocations, as of 7 November 2024.

Morpho DAO

35.4% of MORPHO tokens are owned and controlled by the Morpho DAO. Holders of the MORPHO token can vote on how these tokens are used.

Users & Launch Pools

4.9% of MORPHO tokens have been distributed from the Morpho DAO to users of the Morpho Protocol and launch pool participants. MORPHO tokens will continue to be distributed under the scalable rewards model until the Morpho DAO votes to amend it.

Morpho Association

6.3% of MORPHO tokens are allocated to the Morpho Association for ecosystem development. This allocation can be used to fund partnerships, contributors, and any other initiatives that help grow the Morpho Protocol and advance Morpho’s network.

Reserve for Contributors

5.8% of MORPHO tokens are reserved for contributors to the Morpho Protocol for their role in the development and growth of the network. This reserve includes unallocated tokens set aside by the Morpho DAO for future contributors to the Morpho Protocol such as employees of Morpho Labs, service providers, contractors, and research institutes.

Strategic Partners

27.5% of MORPHO tokens have been allocated to Strategic Partners tokens in exchange for providing support — monetary or otherwise — to the Morpho Protocol. These MORPHO tokens are distributed according to three vesting schedules based on the time these Strategic Partners joined the Morpho ecosystem.

Cohort 1: 4.0% allocated over a 3 year vest, with a 6 month lockup from when the MORPHO token contract was deployed on 24 June 2022.

Cohort 2: 16.8% originally allocated over a 3 year vest, with a 6 month lockup from 24 June 2022. However, these strategic partners have agreed to relock to a 6 month linear vesting, following a 6 month lockup from the earliest of any future transferability date or 3 October 2024. This means 100% is vested by 3 October 2025 at the latest.

Cohort 3: 6.7% allocated over 2 year linear vest, following a 1 year lockup from the earliest of any future transferability date or 17 May 2025. This means 100% is vested by 17 May 2028 at the latest.

Founders

15.2% of MORPHO tokens have been allocated to Morpho’s founders. The tokens were originally allocated over 3 year vest, with a 1-year lockup from when the token was deployed on 24 June 2022. However, all co-founders have agreed to relock to additional 2 year linear vest, following a 1 year lockup from the earliest of any future transferability date or May 17th 2025. This means 100% is vested by 17 May 2028 at the latest.

Early Contributors

4.9% of MORPHO tokens have been allocated to early contributors, including Morpho Labs’ contributors, independent researchers, and advisors, in exchange for contributions to the Morpho Protocol. These tokens have been subject to either:

- A 3-year vesting schedule with a 6-month lockup, or
- A 4-year vesting schedule with a 4-month lockup.

Circulating Supply

The circulating supply on transferability date is expected to be approximately 11.2%.

The chart below shows how the circulating supply of MORPHO tokens might evolve over time:

Note: the evolution of the MORPHO circulating supply over time is subject to change as the Morpho DAO makes future decisions on use of reserves and rate of distributions.

Initial Non-Transferability Phase

While token-based governance allows anyone to own a share of the Morpho network, many token launches are very unfair because the launch date & price are often a centralized decision. Moreover, there is usually a large information asymmetry between the initial team/investors and the potential buyers.

The MORPHO token was launched as non-transferable to tackle those issues, allowing the DAO to turn on transferability anytime. This enabled the protocol to reach meaningful traction prior to a decentralized token launch process.

Under this vision, a good equilibrium for the control of a DAO is a clear distribution between:

1. Users: To own a share of the network and to provide a positive feedback loop on the product.
2. Contributors: To keep improving and building based on the user's feedback and on their technical vision.
3. Strategic Partners: To provide capital and guidance to fuel those contributions.

The current state of distributions in categories 2. and 3. are described in the previous section. As for 1., it is described in the Rewards section.

Morpho DAO Voted to Enable Token Transferability

Although initially deployed as a non-transferable token for the reasons mentioned above, the Morpho DAO voted to enable the transferability of the MORPHO token to advance Morpho’s mission to make financial infrastructure a public good. Since launching in June 2022, the DAO has progressively distributed ownership to the protocol’s users, strategic partners, risk curators, and other contributors aligned with Morpho's long-term vision. Now, by enabling token transferability, the Morpho DAO is opening ownership of the Morpho network to anyone.

As discussed in the community forum, this move is a key step toward realizing the protocol’s mission of transforming financial infrastructure into a public good. It will empower a broader range of participants to engage with and contribute to the Morpho ecosystem, bringing new perspectives and ensuring that DAO decisions reflect the interests of a more diverse community.

A Community-Driven Approach

The Morpho DAO took a unique approach with the MORPHO token, addressing common criticisms of token launches often controlled by a small group or centralized entity. By initially deploying MORPHO as a non-transferable token, the DAO allowed itself—not any centralized entity or individual—to decide when to enable transferability, ensuring a decentralized and community-driven process.

During the non-transferability phase, the DAO prioritized distributing ownership across three key pillars: users, contributors, and strategic partners:

- Users gained ownership in the network and helped shape its evolution,
- Contributors enhanced the technology, network, and accessibility based on user feedback,
- Strategic partners offered capital and guidance.

Now, with a robust ecosystem of contributors, tens of thousands of users, and a network poised to become a cornerstone of decentralized finance, the DAO agreed to take the next step: making the MORPHO token transferable to further its mission of transforming financial infrastructure into a public good.

How to vote

To vote on Morpho governance, use vote.morpho.org. Here, you will find the list of all closed, ongoing, and upcoming proposals for the Morpho DAO.

Voting power

One's voting power in Morpho's governance was previously calculated using this script. Users received governance power by having MORPHO Ownership.

Validation of the proposal

For proposals to pass, they must reach quorum. Currently, this parameter is set to 500,000 MORPHO. Abstention is not taken into account in this quorum at the moment.

Voting period for general proposals is 72 hours + a 24-hour timelock.

Currently, no initial duration is set before the start of the vote. Yet, it is advised to post your proposal on the governance forum before submitting a vote.

Delegation

Delegation allows you to transfer the voting power associated with your tokens to another member of the Morpho community who has volunteered to actively participate in governance. These individuals are referred to as “delegates.”

You retain the ability to change your designated delegate at any time.

How do I delegate?

1. Go to the Delegation Platform: Visit delegate.morpho.org
2. Connect Your Wallet: Use the connect option to link your wallet to the delegation platform
3. Select a Delegate: Choose the address you wish to delegate your voting power to
4. Confirm Delegation: Review and confirm the delegation to set your chosen delegate

You can find more info about Delegation in the FAQ.

Create a Proposal

To increase the likelihood of a successful proposals, it’s recommended to follow this workflow:

1. Prior to initiating a vote, create a draft proposal on forum.morpho.org for community feedback
2. After gathering feedback, make any desired edits to the original post
3. Once complete, the proposer can create an official vote on vote.morpho.org

In order to create a proposal on Morpho's Snapshot page, you need to have at least 500,000 MORPHO.