



Designing Resilient Stablecoins with M0: Best Security and Stability Practices

Antonina Norair,
CTO M0 Labs
X: @0xt0n1

Where are we now ?

Market Cap [?]

\$297.72B

▼ 0.47% from 30d ago

Monthly Transfer Volume [?]

\$4.94T

▲ +18.44% from 30d ago

Monthly Active Addresses [?]

39.85M

▲ +32.56% from 30d ago

Holders

203.30M

▲ +3.03% from 30d ago

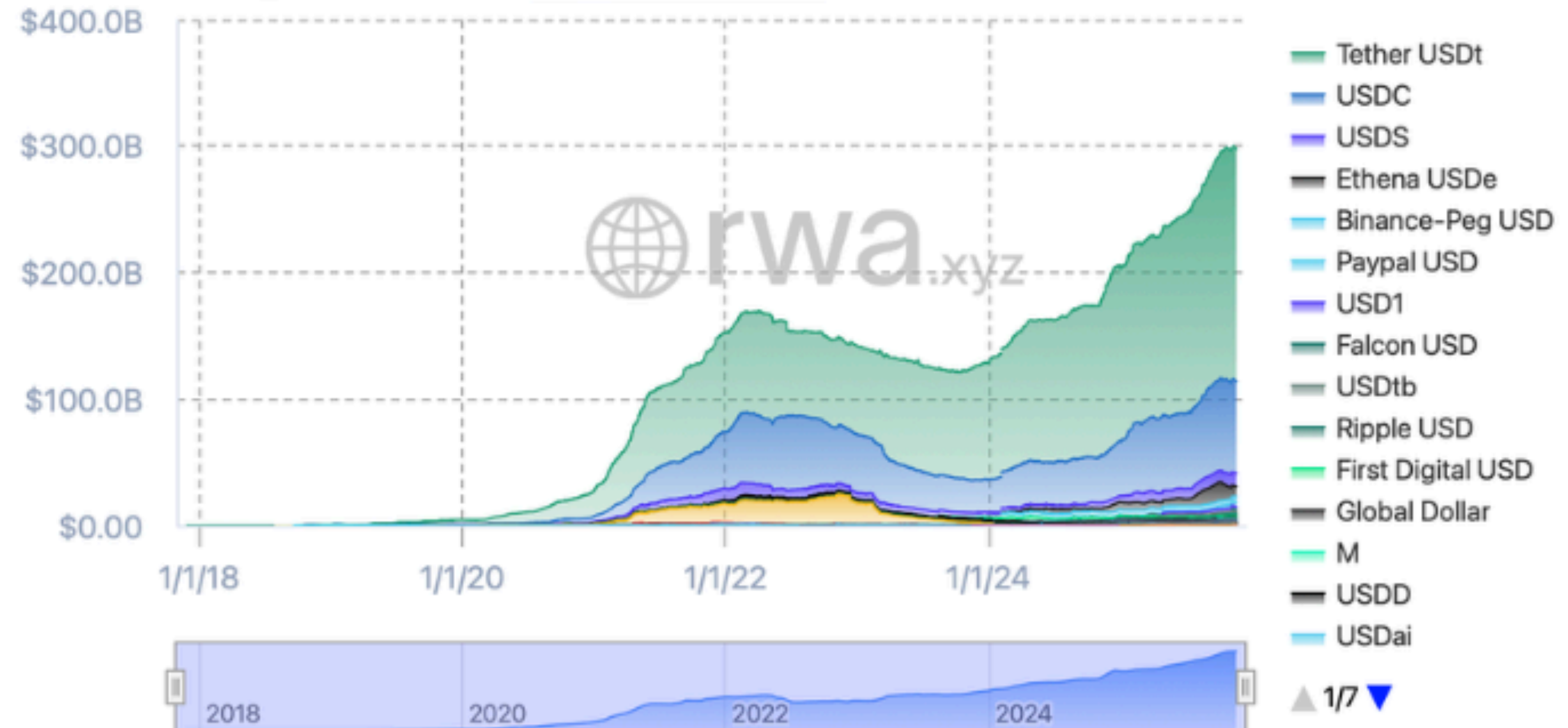
Stablecoin Metrics

Download ...

Market Caps Transfer Volume Active Addresses Mint / Burn Volumes Transfer Count Holders

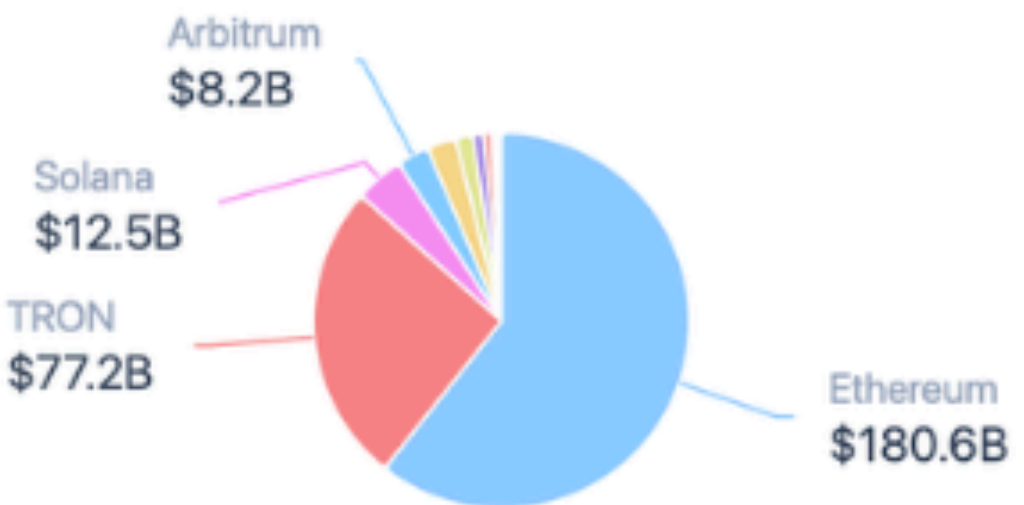
Market Caps shows the scale and power law distribution between all stablecoin issuers.

Show % of Total ☐ Group By Asset



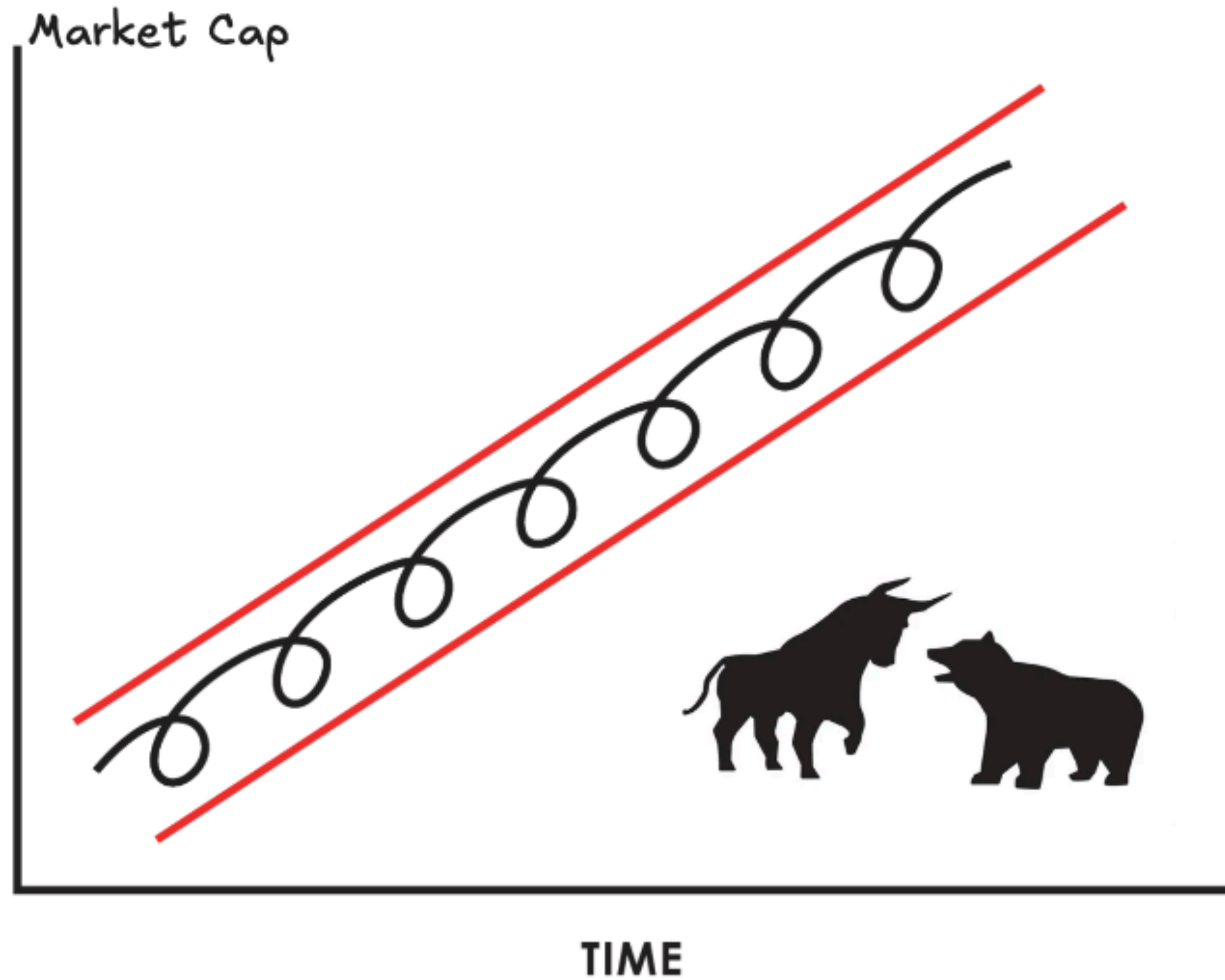
Stablecoin Market Caps

GROUP BY Network



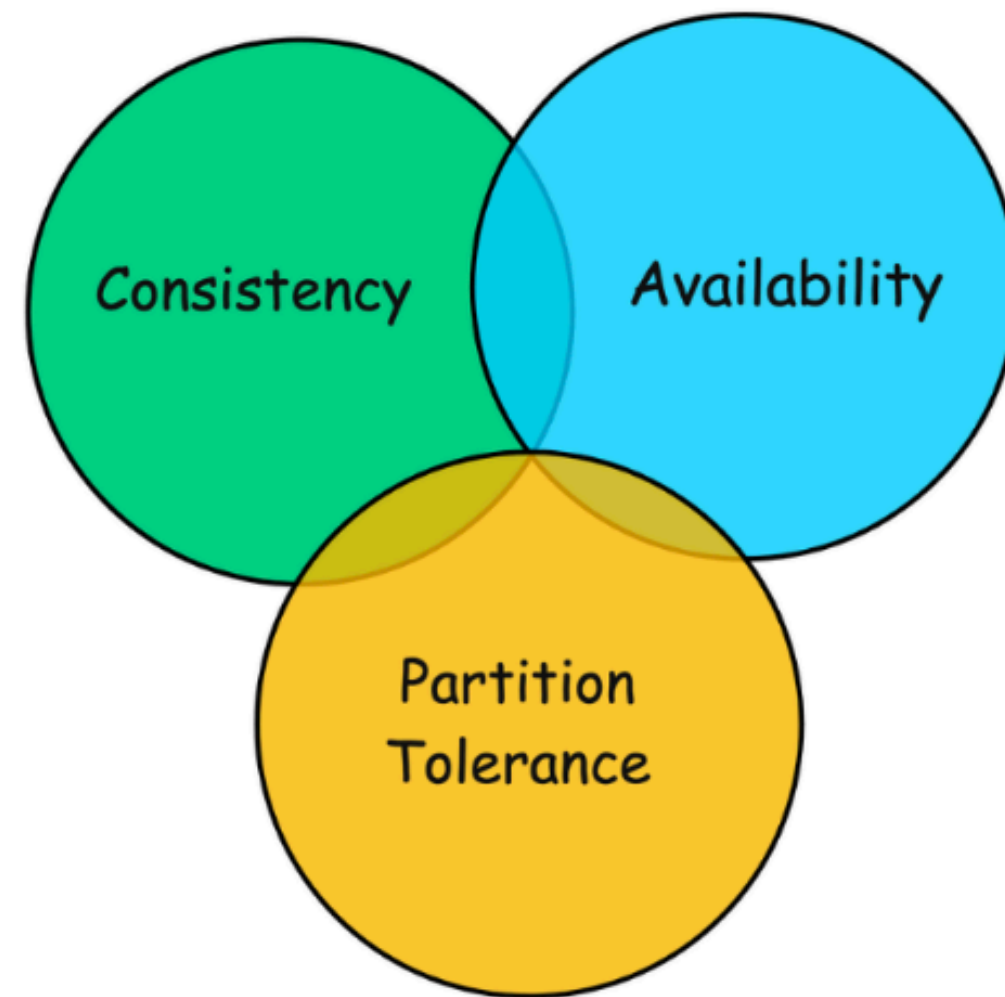
rwa.xyz
as of 11/21/2025

Where are we going ?

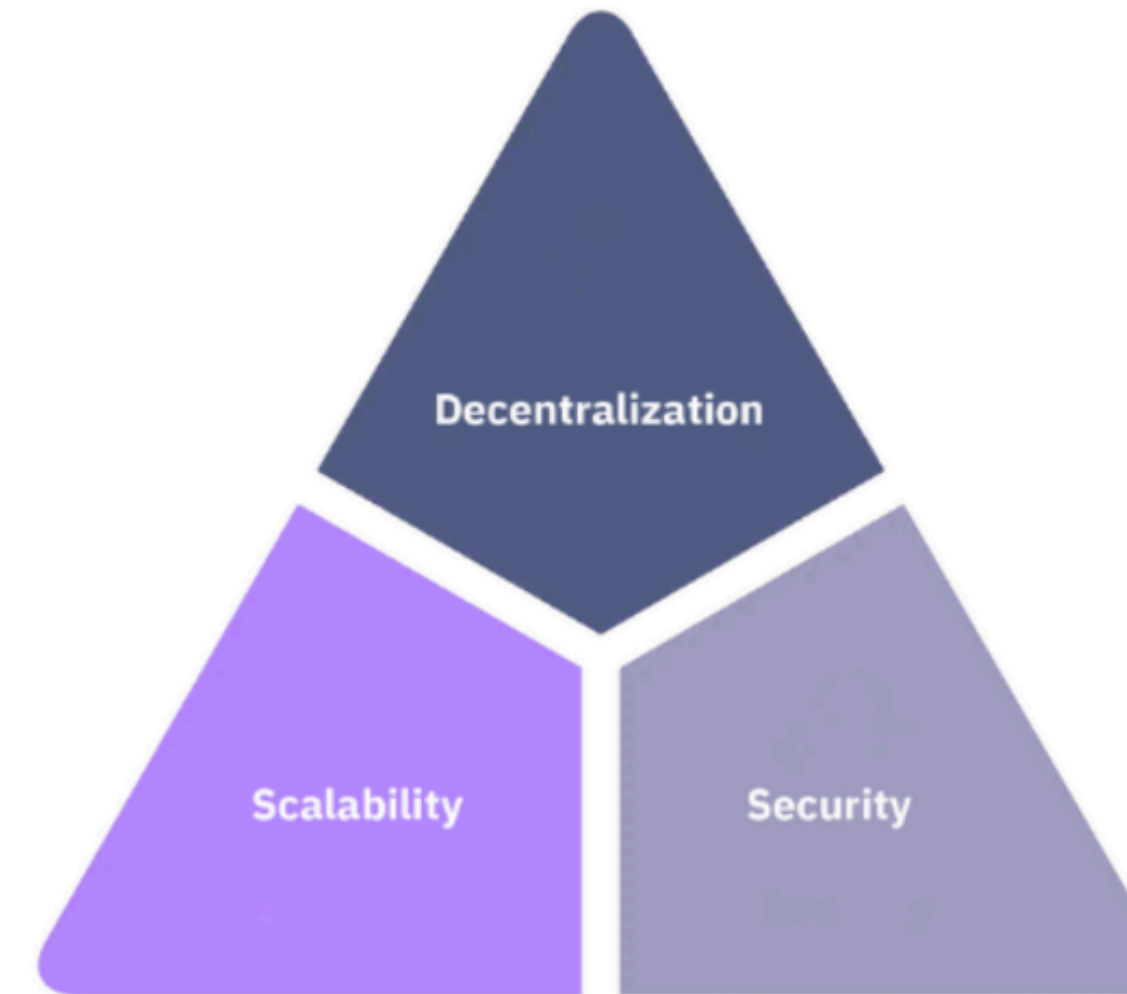


Trilemmas

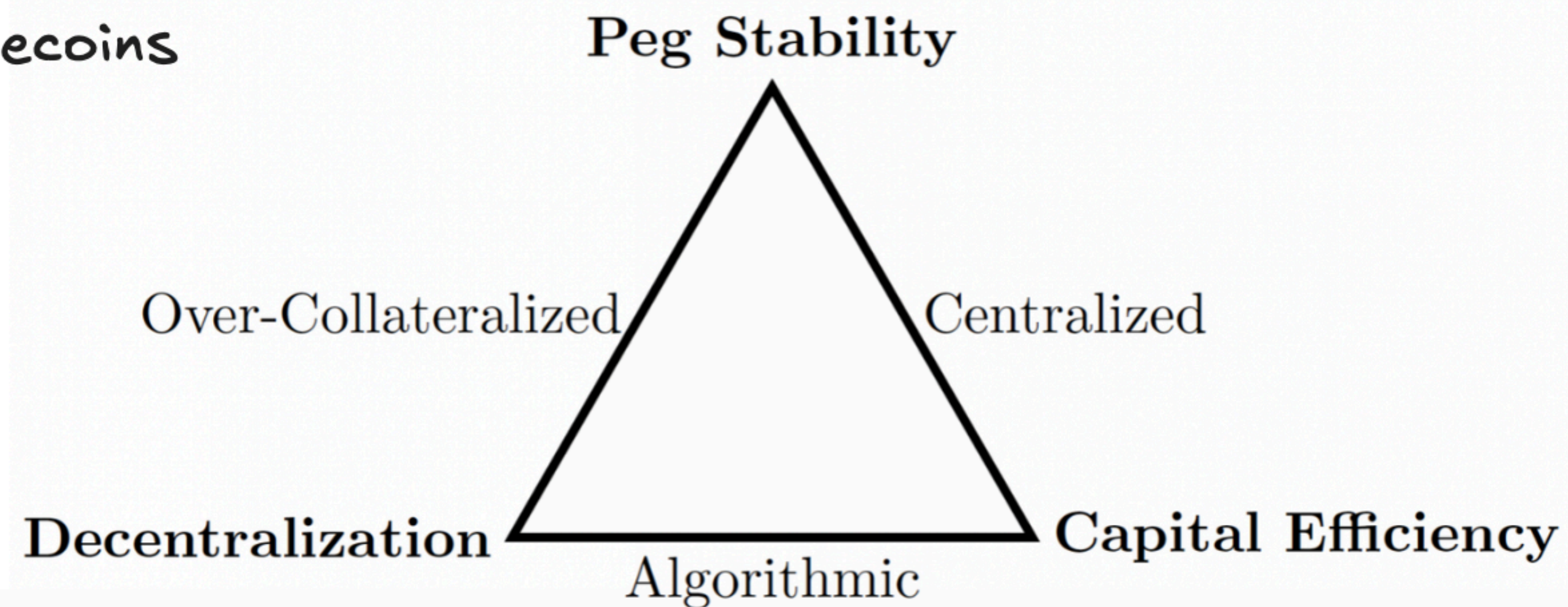
Distributed systems



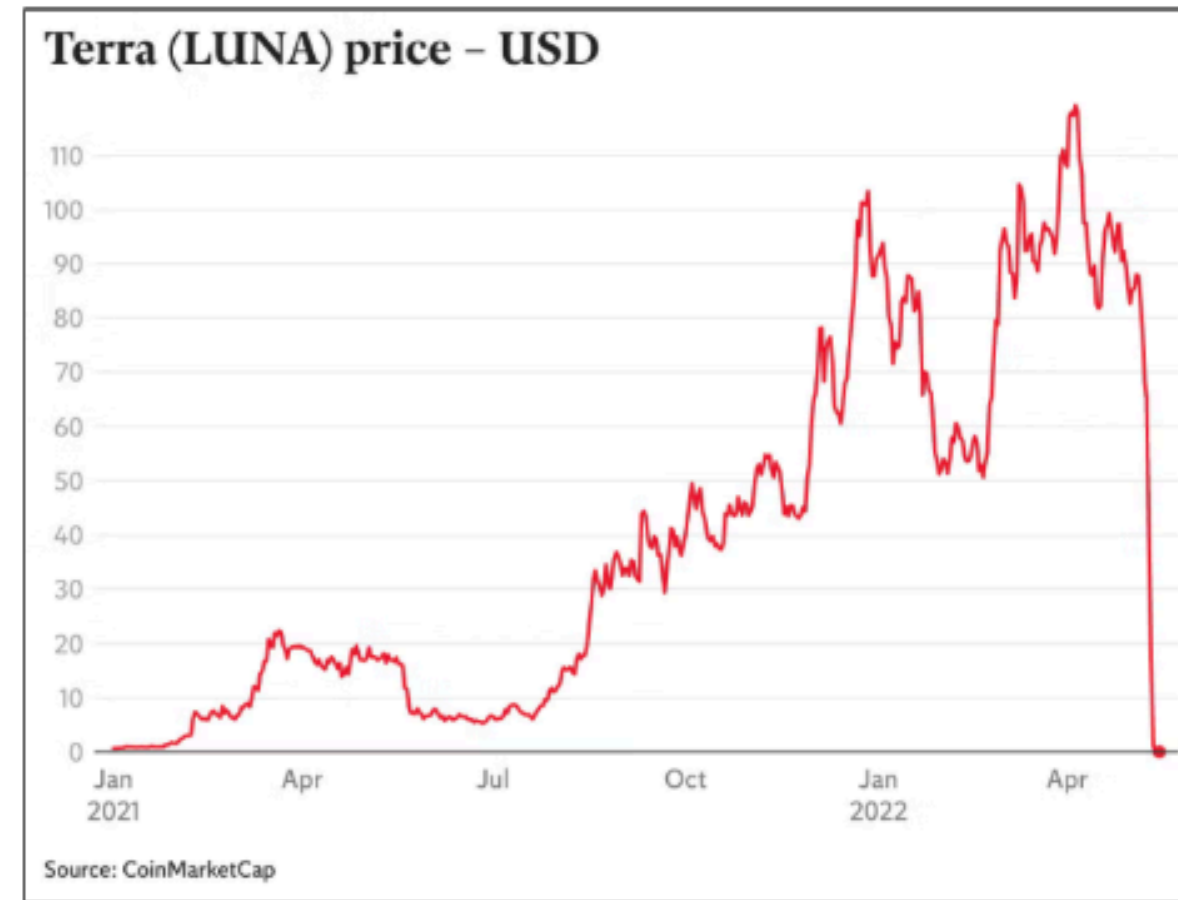
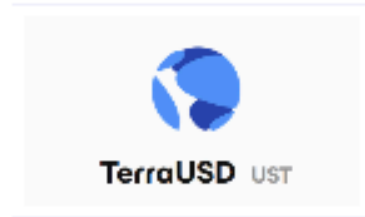
Blockchain



Stablecoins



History Lessons



“We never thought it would happen, but it just did. We just experienced the world’s first large-scale crypto bank run.”



Bybit Update: Ethena Labs Addresses Bybit Hack Concerns, Confirms USDe Reserves Are Secure



Binance News

Feb 21 · Verified Binance official account

Follow



Collateral Damage from Stream Finance Hack Estimated at \$285 Million

Losses on major credit and collateral positions exceed \$280 million.

TRADING 12 MIN READ EASY

Did USDe Really Depeg? The Truth Behind Binance’s \$0.65 Crash Explained

Resilient Stablecoin Designs



DON'TS

1. Relying on arbitrage as a source of stability
2. Diversifying collateral (not 100% in the *MOST SECURE ASSET* class)
3. Collateralized by assets Protocols control, seigniorage
4. Inheriting risks of other stables as collateral, loops & leverage
5. Manual liquidity operations
6. Fragile Oracle design relying on volatile low liquidity secondary market, hardcoded \$1

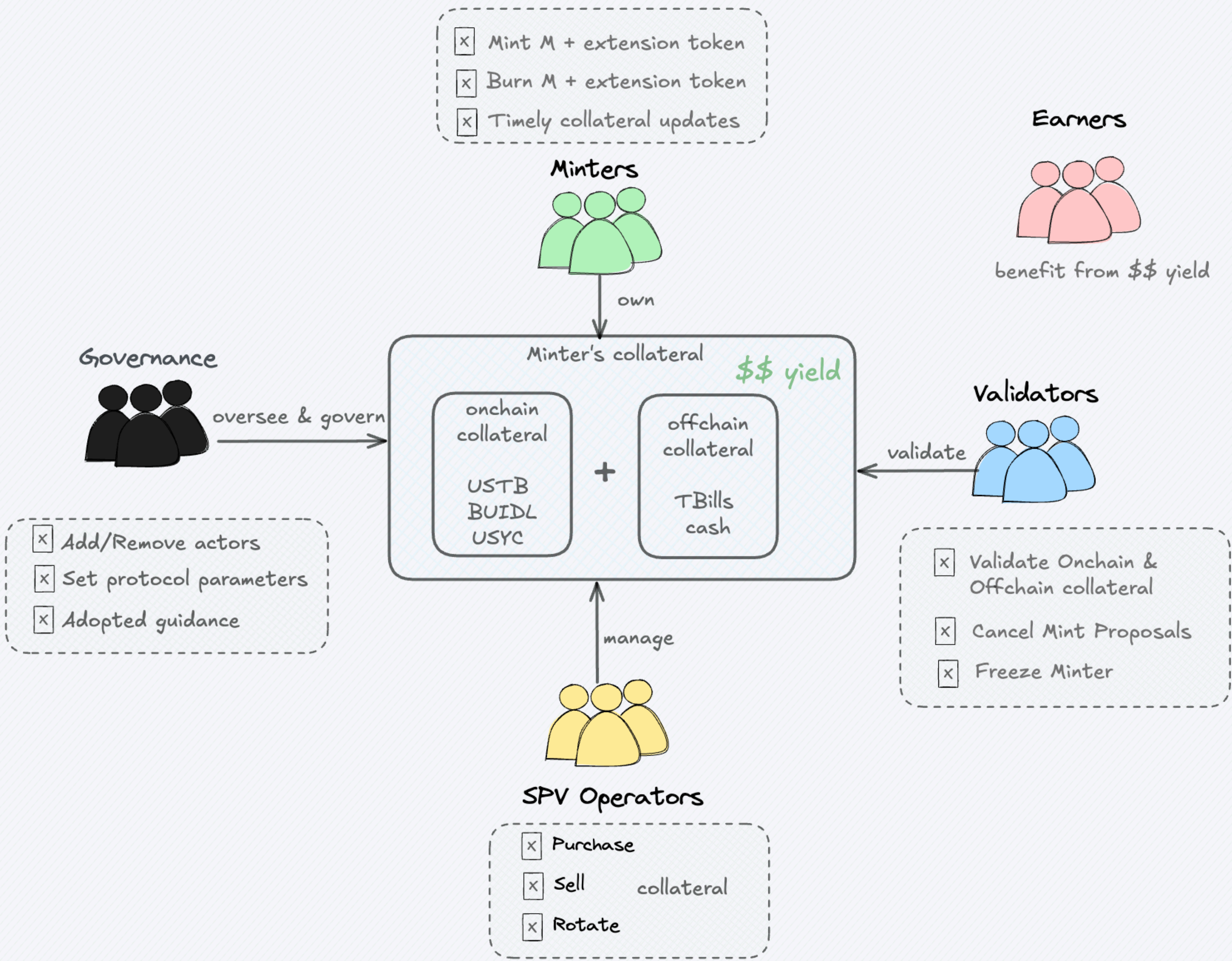


DOs

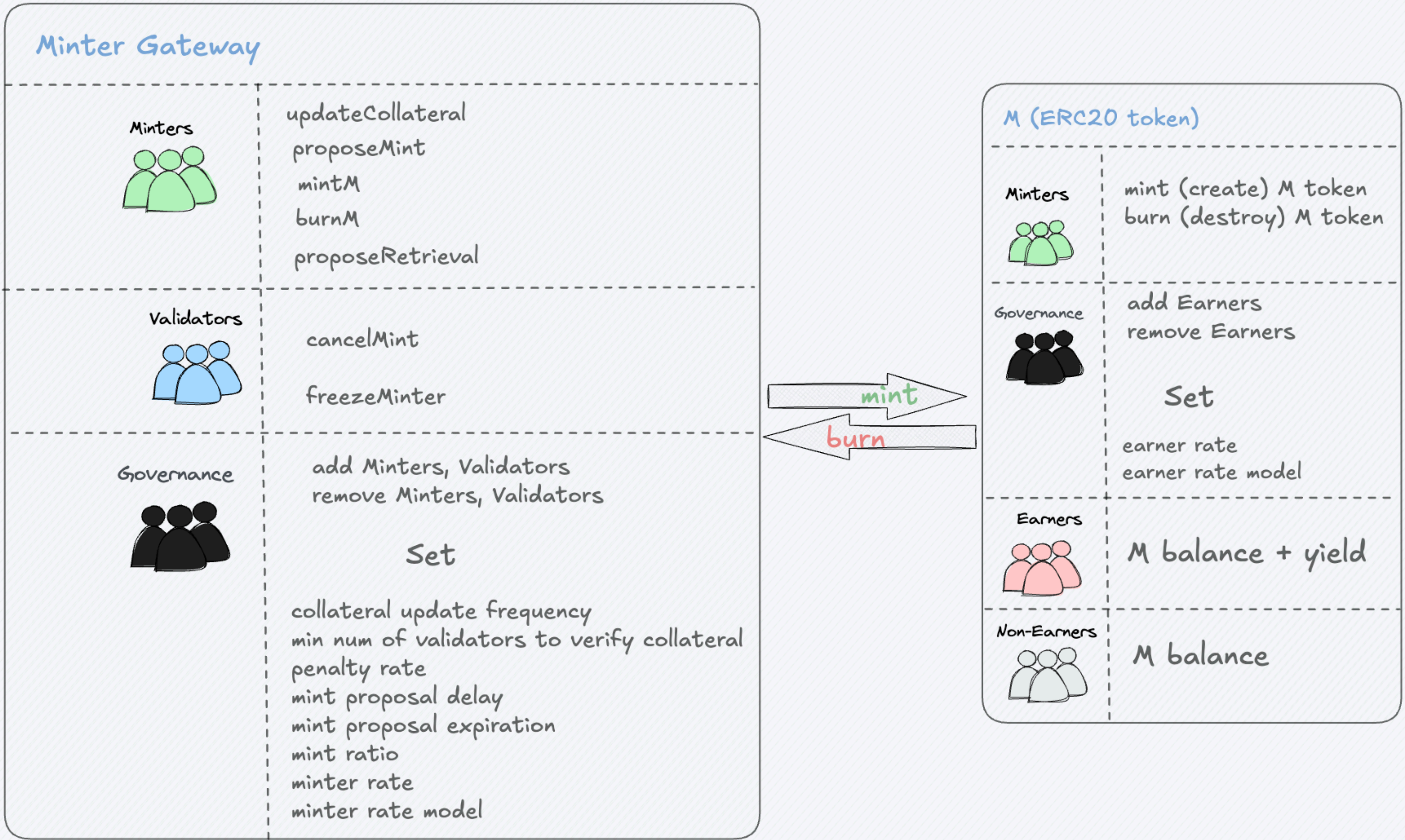
1. High quality collateral with risk-free rate
2. Algorithms for yield accrual, not peg stability
3. Automatic liquidity management with clear expectations
4. transparent *NAV* oracles vs hardcoded \$ 1

MO Core Ecosystem Network

MO Approach - stable collateral, automated yield distribution



MO Core Protocol



MO Core Protocol Interest Accrual

Protocol rate accruing logic

Minter Principal += Minted M Amount / Index(t2)

at t2:

$$\text{Index}(t2) = \text{Index}(t1) * e^{(\text{Minter Rate} * (t2 - t1))}$$

at tn with Index(tn):

$$\text{Minter Owed M}(tn) = \text{Minter Principal} * \text{Index}(tn)$$

$$\text{Earner Rate} = \min(\text{Max Earner Rate}, \text{Safe Earner Rate})$$

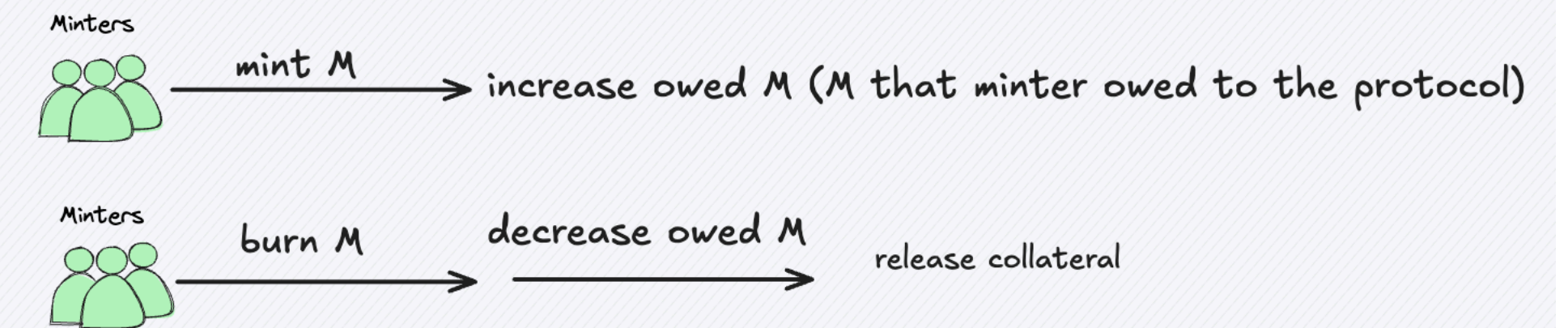
$$\text{Safe Earner Rate} = \text{fn}(\text{Minter rate}, \text{owed M}, \text{Earning M Supply})$$



$$\text{total owed M} * (e^{(\text{minter rate} * dt)} - 1) \geq \text{earning M supply} * (e^{(\text{earner rate} * dt)} - 1)$$

approximation:

$$\text{total active owed M} * \text{minter rate} \geq \text{total earning supply} * \text{earner rate}$$



MO Core Protocol Interest Accrual

Protocol rate accruing logic

Minter Principal += Minted M Amount / Index(t2)

at t2:

$$\text{Index}(t2) = \text{Index}(t1) * e^{(\text{Minter Rate} * (t2 - t1))}$$

at tn with Index(tn):

$$\text{Minter Owed M}(tn) = \text{Minter Principal} * \text{Index}(tn)$$

$$\text{Earner Rate} = \min(\text{Max Earner Rate}, \text{Safe Earner Rate})$$

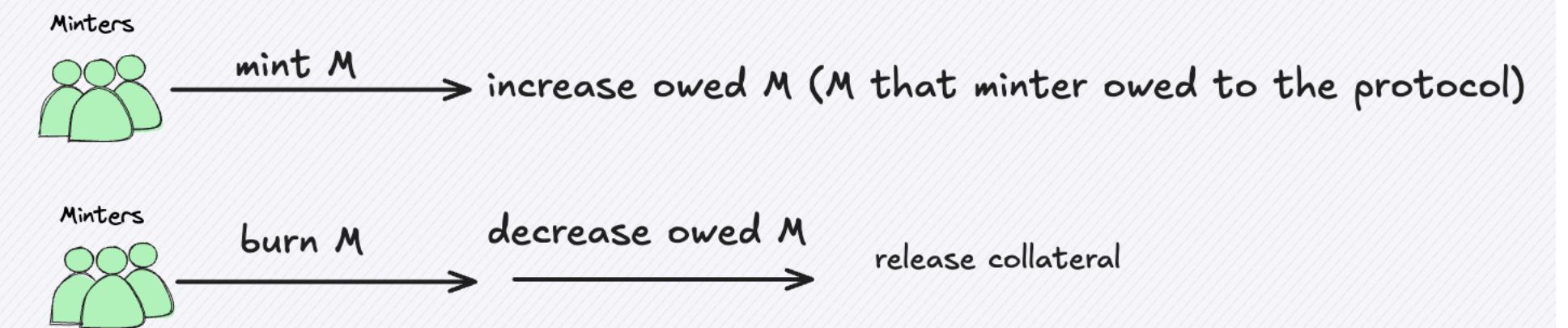
$$\text{Safe Earner Rate} = \text{fn}(\text{Minter rate}, \text{owed M}, \text{Earning M Supply})$$



$$\text{total owed M} * (e^{(\text{minter rate} * dt)} - 1) \geq \text{earning M supply} * (e^{(\text{earner rate} * dt)} - 1)$$

approximation:

$$\text{total active owed M} * \text{minter rate} \geq \text{total earning supply} * \text{earner rate}$$



Earner Rate Model, Safe Earner Rate calculations

```
function getSafeEarnerRate(
    uint240 totalActiveOwedM_,
    uint240 totalEarningSupply_,
    uint32 minterRate_
) public pure returns (uint32) {
    // solhint-disable max-line-length
    // When `totalActiveOwedM_ >= totalEarningSupply_`, it is possible for the earner rate to be higher than the
    // minter rate and still ensure cashflow safety over some period of time (`RATE_CONFIDENCE_INTERVAL`). To ensure
    // cashflow safety, we start with `cashFlowOfActiveOwedM >= cashFlowOfEarningSupply` over some time `dt`.
    // Effectively:  $p1 * (\exp(\text{rate1} * dt) - 1) \geq p2 * (\exp(\text{rate2} * dt) - 1)$ 
    // So:  $\text{rate2} \leq \ln(1 + (p1 * (\exp(\text{rate1} * dt) - 1)) / p2) / dt$ 
    // 1.  $\text{totalActive} * (\text{delta\_minterIndex} - 1) \geq \text{totalEarning} * (\text{delta\_earnerIndex} - 1)$ 
    // 2.  $\text{totalActive} * (\text{delta\_minterIndex} - 1) / \text{totalEarning} \geq \text{delta\_earnerIndex} - 1$ 
    // 3.  $1 + (\text{totalActive} * (\text{delta\_minterIndex} - 1) / \text{totalEarning}) \geq \text{delta\_earnerIndex}$ 
    // Substitute `delta_earnerIndex` with `exponent((earnerRate * dt) / SECONDS_PER_YEAR)`:
    // 4.  $1 + (\text{totalActive} * (\text{delta\_minterIndex} - 1) / \text{totalEarning}) \geq \text{exponent}(\text{earnerRate} * dt) / \text{SECONDS\_PER\_YEAR}$ 
    // 5.  $\ln(1 + (\text{totalActive} * (\text{delta\_minterIndex} - 1) / \text{totalEarning})) \geq (\text{earnerRate} * dt) / \text{SECONDS\_PER\_YEAR}$ 
    // 6.  $\ln(1 + (\text{totalActive} * (\text{delta\_minterIndex} - 1) / \text{totalEarning})) * \text{SECONDS\_PER\_YEAR} / dt \geq \text{earnerRate}$ 

    // When `totalActiveOwedM_ <= totalEarningSupply_`, the instantaneous earner cash flow must be less than the
    // instantaneous minter cash flow. To ensure instantaneous cashflow safety, we use the derivatives of the
    // previous starting inequality, and substitute `dt = 0`.
    // Effectively:  $p1 * \text{rate1} \geq p2 * \text{rate2}$ 
    // So:  $\text{rate2} \leq p1 * \text{rate1} / p2$ 
    // 1.  $\text{totalActive} * \text{minterRate} \geq \text{totalEarning} * \text{earnerRate}$ 
    // 2.  $\text{totalActive} * \text{minterRate} / \text{totalEarning} \geq \text{earnerRate}$ 
    // solhint-enable max-line-length

    if (totalActiveOwedM_ == 0 || minterRate_ == 0) return 0;

    if (totalEarningSupply_ == 0) return type(uint32).max;

    if (totalActiveOwedM_ <= totalEarningSupply_) {
        // NOTE: `totalActiveOwedM_ * minterRate_` can revert due to overflow, so in some distant future, a new
        // rate model contract may be needed that handles this differently.
        return uint32((uint256(totalActiveOwedM_) * minterRate_) / totalEarningSupply_);
    }
}
```

Padé(4/4) Euler number calculations

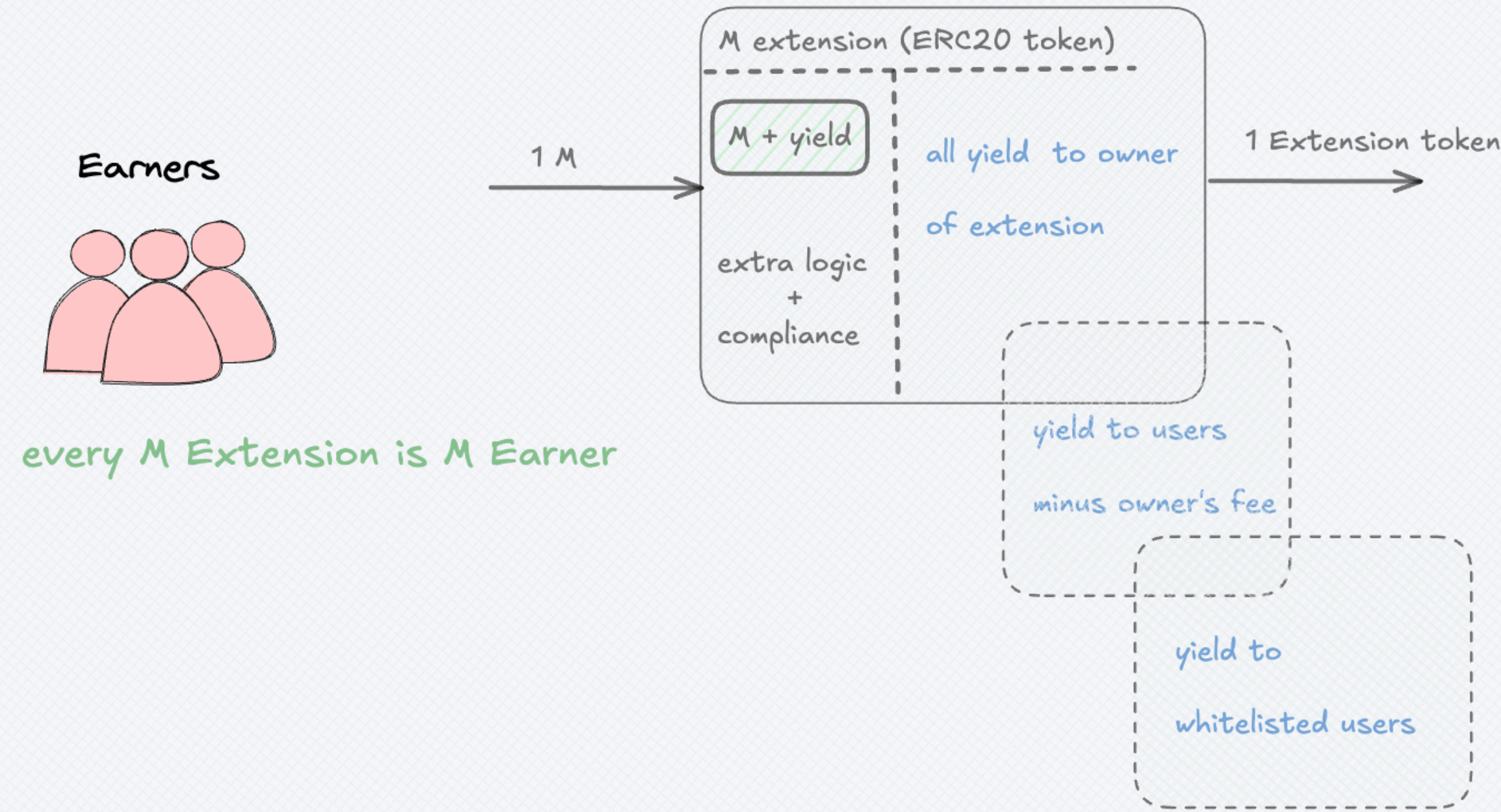
```
/**
 * @notice Helper function to calculate  $y = e^x$  using R(4,4) Padé approximation:
 *  $e(x) = (1 + x/2 + 3(x^2)/28 + x^3/84 + x^4/1680) / (1 - x/2 + 3(x^2)/28 - x^3/84 + x^4/1680)$ 
 * See: https://en.wikipedia.org/wiki/Pad%C3%A9\_table
 * See: https://www.wolframalpha.com/input?i=Pad%C3%A9Approximant%5Bexp%5Bx%5D%2C%7Bx%2C0%2C%7B4%2C+4%7D%5D
 * Despite itself being a whole number, `x` represents a real number scaled by `EXP_SCALED_ONE`, thus
 * allowing for  $y = e^x$  where  $x$  is a real number.
 * @dev Output `y` for a `uint72` input `x` will fit in `uint48`
 */
function exponent(uint72 x) internal pure returns (uint48 y) {
    // NOTE: This can be done unchecked even for `x = type(uint72).max`.
    // Verify by removing `unchecked` and running `test_exponent()`.
    unchecked {
        uint256 x2 = uint256(x) * x;

        // `additiveTerms` is  $(1 + 3(x^2)/28 + x^4/1680)$ , and scaled by `84e27`.
        // NOTE: `84e27` the cleanest and largest scalar, given the various intermediate overflow possibilities.
        // NOTE: The resulting  $(x2 * x2) / 20e21$  term has been split up in order to avoid overflow of  $x2 * x2$ .
        uint256 additiveTerms = 84e27 + (9e3 * x2) + ((x2 / 2e11) * (x2 / 1e11));

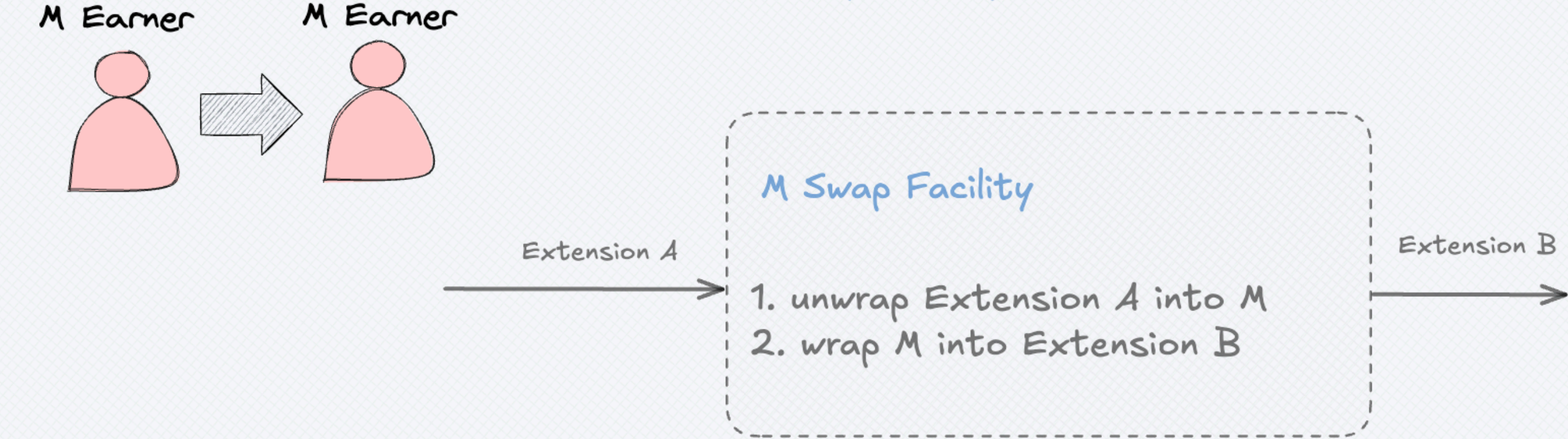
        // `differentTerms` is  $(-x/2 - x^3/84)$ , but positive (will be subtracted later), and scaled by `84e27`.
        uint256 differentTerms = uint256(x) * (42e15 + (x2 / 1e9));

        // Result needs to be scaled by `1e12`.
        // NOTE: Can cast to `uint48` because contents can never be larger than `type(uint48).max` for any `x`.
        // Max `y` is  $\sim 200e12$ , before falling off. See links above for reference.
        return uint48(((additiveTerms + differentTerms) * 1e12) / (additiveTerms - differentTerms));
    }
}
```

M Extension Framework



Built-in Interoperability



- USDai
USDai
- Usual
UsualM
- MetaMask USD
mUSD
- Noble Dollar
USDN
- \$M (wrapped)
wM
- USDhl
USDHL
- KAST Dollar
USDK
- Dfns Rewards
Ofns
- Braid Dollar
USDZ



Designing Resilient Stablecoins with M0: Best Security and Stability Practices

Antonina Norair,
CTO M0 Labs
X: @0xt0n1