StableGuard — Technical Whitepaper

A modular, MEV-aware stablecoin architecture for research

Martín Pérez

1. Table of Contents

2. Executive Summary

- StableGuard is a modular ERC20 stablecoin protocol focused on research and local prototyping, not production deployment.
- Core modules include a validated price oracle with fallback, collateral management, liquidation via direct and Dutch auctions, repeg coordination, arbitrage routing, and a timelock for sensitive operations.
- The protocol enforces safety through strict validations, LTV thresholds, rate-limiting, bounded actions, and delayed owner operations.
- MEV-aware design uses commit-reveal auctions and bounded execution to mitigate manipulation.
- Observability is prioritized via explicit events and predictable failure modes to aid monitoring and post-mortem analysis.
- The repository is self-contained: `forge install`, `forge build`, `forge test -vv`; dependencies are pinned in `foundry.lock`, and `lib/` is populated locally.

3. Abstract

StableGuard is a modular, gas-optimized ERC20 stablecoin protocol designed for local research and

prototyping. It demonstrates production-style architecture (owner controls, timelock, module wiring), robust pricing via a validated oracle with fallback logic, collateral accounting, liquidation through direct and Dutch auction flows, and optional arbitrage/repeg mechanisms. Safety is enforced through strict validation, rate limiting, timelock delays, and MEV-aware auction design. This whitepaper details design goals, architecture, modules, core flows, parameters, security model, invariants, and operational guidance. The repository runs locally without external infra and is not intended for mainnet deployment.

4. Scope & Assumptions

- Non-production educational scope: no governance/multisig, audit, or external orchestration.
- Deterministic local testing with mocks; ETH supported via `address(0)` conventions.
- Parameters are illustrative; tune per risk appetite (LTV, thresholds, cooldowns, limits).
- Collaborators run `forge install`; dependency pins are recorded in `foundry.lock`.

5. Design Goals

- Safety first: explicit validations, timelock for sensitive ops, bounded actions.
- Composability: clear interfaces and module boundaries for future extensions.
- Gas efficiency: storage packing, minimal copies, assembly in critical paths.

- Determinism: predictable state transitions and test reproducibility.
- Clarity: explicit events, well-defined responsibilities, and documented flows.

6. System Architecture

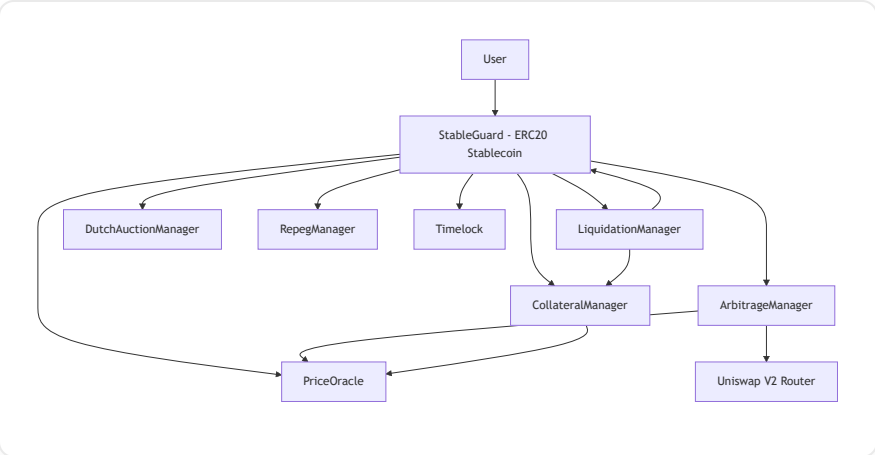
6.1 Key Flows Overview

This overview summarizes end-to-end process flows and how modules interact. Each item links to a detailed section with diagrams, preconditions, and invariants.

- Deposit & Mint (Deposit & Mint)
    - Trigger: user deposits supported collateral.
    - Steps: record deposit; fetch and normalize oracle price; compute collateral value; enforce LTV and per-operation caps; mint tokens; emit events.
    - Outcomes: supply increases; position updated and remains within safety limits; state is auditable.
- Burn & Withdraw (Burn & Withdraw)
    - Trigger: user burns stable tokens to withdraw collateral.
    - Steps: burn tokens; re-evaluate LTV and thresholds; release collateral within limits; emit events.
    - Outcomes: supply and debt reduce deterministically; position remains safe and consistent.
- Liquidation — Direct vs Dutch Auction (Liquidation — Direct vs Dutch Auction)

    - Trigger: position marked unsafe by LTV/threshold.
    - Path selection: direct liquidation under caps or schedule Dutch auction depending on size and configuration.
    - Outcomes: safety restored, penalties applied, balances settled, events emitted.
- Dutch Auction — Commit-Reveal (Dutch Auction — Commit-Reveal)
    - Phases: commit hashed bids within window; reveal bids; descending price discovers clearing; settle and emit events.
    - Guards: size caps, time windows, MEV-aware design, auditable via events.
- Arbitrage & Repeg Coordination (Arbitrage & Repeg Coordination) with (Arbitrage Flow) and (Repeg Flow)
    - Trigger: on-chain price dislocation or peg deviation relative to target.
    - Steps: monitor deviation; compute bounded action; apply buy/sell pressure under cooldowns and caps; route execution via DEX; evaluate peg and record telemetry.
    - Outcomes: gradual realignment while prioritizing safety and avoiding oscillations.
- Timelock — Delayed Sensitive Actions (Timelock — Delayed Sensitive Actions)
    - Steps: queue action; wait minimum delay; execute within grace period; optional cancel; emit Queue/Execute/Cancel events.

    - Outcomes: auditable, controlled parameter changes and emergency operations.
- Oracle Normalization & Fallback (Oracle Normalization & Fallback)
    - Steps: validate feed and decimals; normalize to 18-decimals; fallback to conservative price if invalid or stale; return safe price.
    - Outcomes: consistent and safe pricing for collateral and LTV calculations.
- Collateralization & Safety Guardrails (Collateralization & Safety Guardrails)
    - Steps: compute LTV; classify state (safe, warning, liquidation); enforce invariants, caps, and circuit-breakers.
    - Outcomes: predictable risk management, clear revert paths, and observable state changes.



6.2 Responsibilities

- StableGuard: core token; orchestrates deposit/mint, burn/withdraw; enforces ratios.

- PriceOracle: validated price retrieval; fallback prices; decimal normalization to 18-decimals.
- CollateralManager: deposit/withdraw accounting; per-token LTV and thresholds; ETH/ERC20 handling.
- LiquidationManager: safety checks; direct liquidations; settlement notifications to StableGuard.
- DutchAuctionManager: MEV-aware descending price curve with commit-reveal; fair price discovery.
- RepegManager: monitors deviation and applies bounded buy/sell pressure under guards.
- ArbitrageManager: detects on-chain price dislocations and coordinates execution via DEX.
- Timelock: queue/execute/cancel lifecycle for owner actions with enforced delay.

7. Actors & Roles

- User: deposits collateral, mints stable tokens, burns to withdraw.
- Liquidator: participates in direct liquidations or Dutch auctions to restore safety.
- Operator/Owner: adjusts parameters via timelock; runs maintenance and emergency actions.
- Oracle Provider: publishes external price feeds consumed and validated by `PriceOracle`.
- Observer/Analyst: consumes events and telemetry for monitoring and post-trade analysis.

8. Contracts & Interfaces

- `src/StableGuard.sol` — ERC20 stable token, mint/burn orchestration, module wiring, invariants enforcement.
- `src/PriceOracle.sol` — Aggregator-style price validation, fallback paths, normalization to 18-decimals.
- `src/CollateralManager.sol` — Collateral ledger per user/token; LTV and threshold enforcement.
- `src/LiquidationManager.sol` — Safety detection and direct liquidation with caps and validations.
- `src/DutchAuctionManager.sol` — Auction scheduling, commit-reveal bids, descending curve settlement.
- `src/RepegManager.sol` — Controlled peg realignment with cooldowns, limits, and circuit-breakers.
- `src/ArbitrageManager.sol` — DEX routing and guarded execution for arbitrage opportunities.
- `src/Timelock.sol` — Delayed owner actions for configuration changes and emergency operations.
- `src/Constants.sol` — Shared parameters and ratios; tuned per environment.
- `src/interfaces/*.sol` — Interfaces for oracle, collateral, liquidation, auction, repeg, arbitrage.

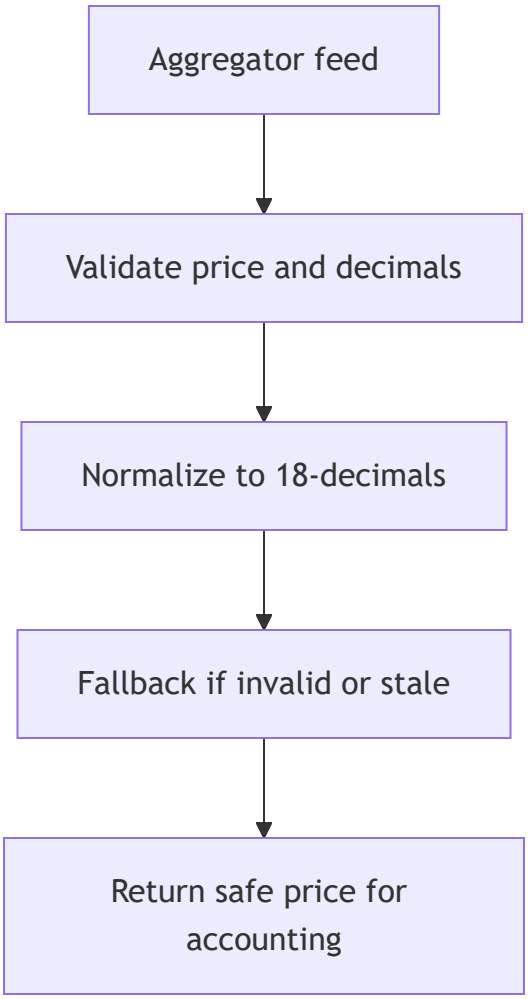
9. Data & Storage Layout

- Core mappings: balances and collateral per account; positions indexed by token.

- Packed structs: ratios, thresholds, cooldowns consolidated to minimize slots.
- Derived values: normalized prices and collateral valuations computed on demand.
- Assembly paths: critical functions use low-level checks to minimize overhead.
- Event surfaces: state changes emit explicit events for observability.

10. Core Mechanisms

10.3 Oracle Normalization & Fallback



- Validation: freshness, non-zero, bounds checks.
- Normalization: consistent 18-decimals for downstream calculations.
- Fallback: secondary price source or conservative constant if feed invalid.

Preconditions

- Price source accessible and expected format.
- Freshness thresholds and value bounds configured.

Steps

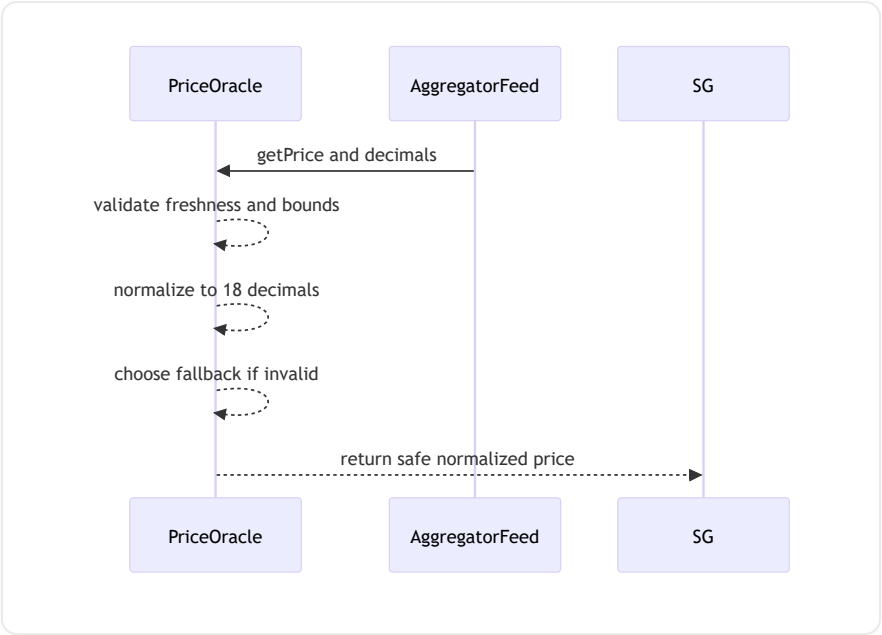
- Fetch price and decimals; validate freshness and non-zero.
- Normalize to 18 decimals; apply bounds if necessary.
- If invalid or out of range, use conservative fallback price.

Postconditions & Events

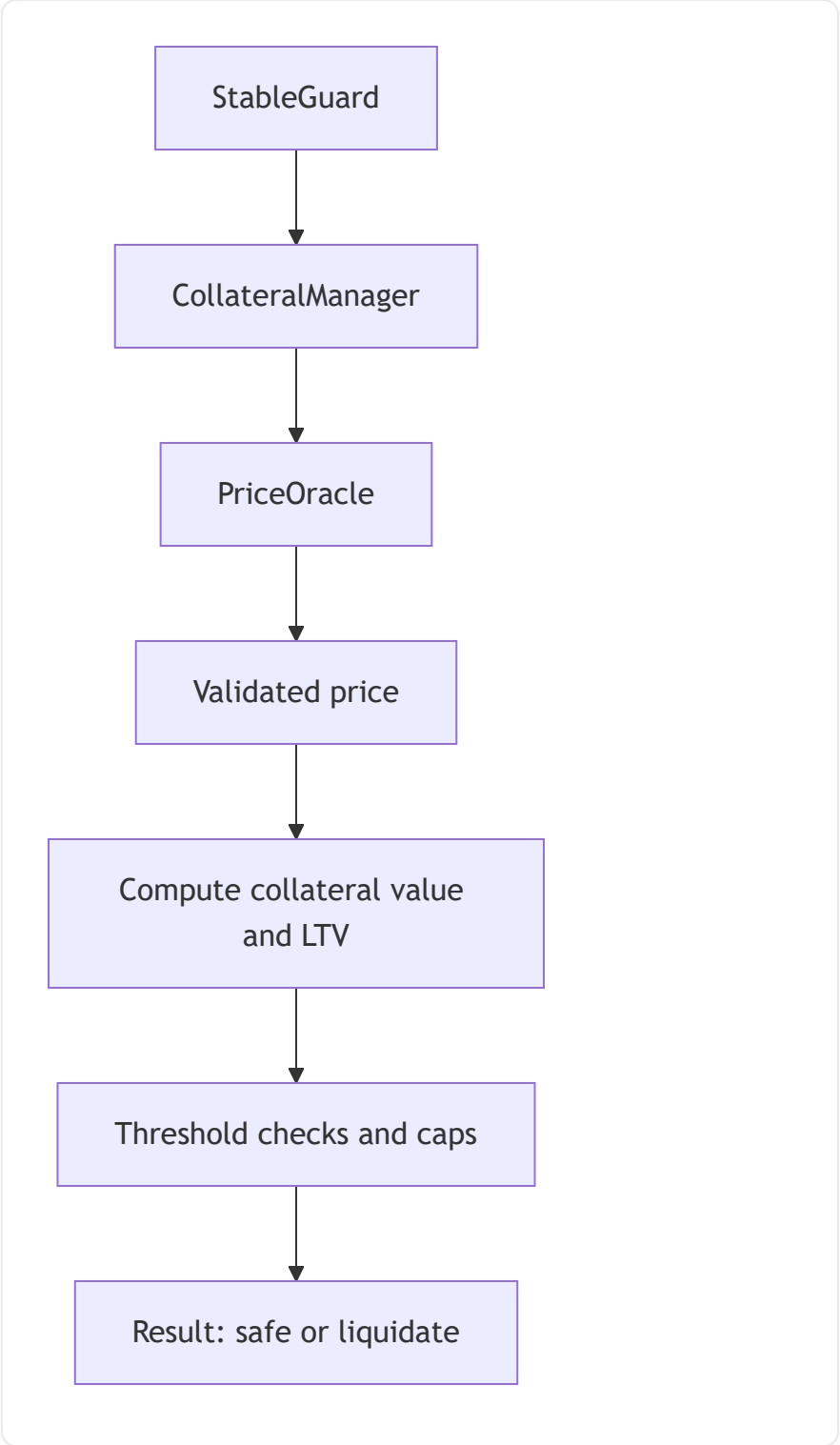
- Normalized price ready for collateral and LTV calculations.
- Emit price update event and fallback activation if applicable.

Failures & Reverts

- Revert if price unavailable and fallback not allowed.
- Record telemetry for diagnostics.


PriceOracle
AggregatorFeed
SG
getPrice and decimals
validate freshness and bounds
normalize to 18 decimals
choose fallback if invalid
return safe normalized price
PriceOracle
AggregatorFeed
SG

10.4 Collateralization & Safety Guardrails



- Per-token settings: LTV, liquidation thresholds, penalties.
- System caps: operation size limits, slippage bounds, circuit-breakers.

Preconditions

- Collateral registered per user and token.
- LTV parameters and liquidation thresholds defined.

Steps

- Normalize prices and compute collateral value per token.
- Compute LTV per position and at aggregate level if applicable.
- Evaluate safety thresholds and system limits.
- Determine state: safe, warning, or liquidation required.

Postconditions & Events

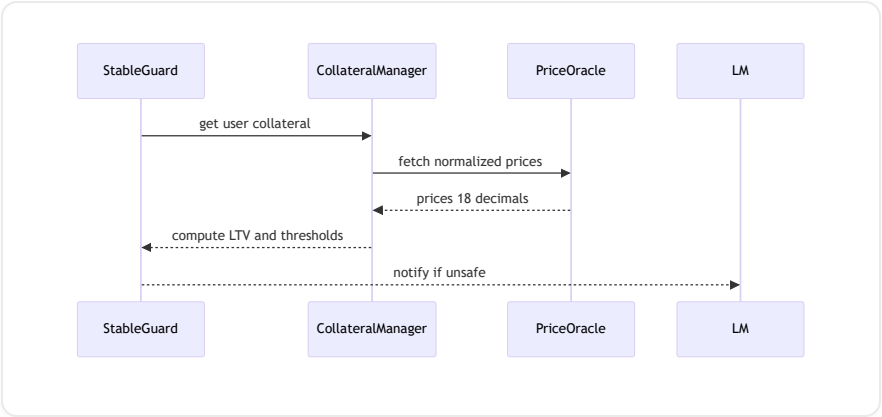
- Emit safety assessment and state-change events.
- Update indicators for liquidation managers.

Failures & Reverts

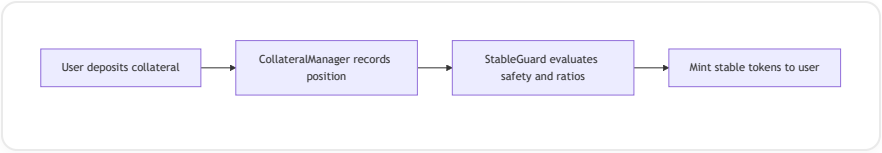
- Revert deposits or mints if new states violate invariants.
- Apply circuit-breakers if global limits are exceeded.

Invariants & Formula

- `LTV = valued_debt / normalized_collateral_value`
- Requires: `LTV ⩽ LTV_max` before mint and after key operations.



10.5 Deposit & Mint



Preconditions

- User with valid address; collateral token allowed.
- LTV parameters and operation limits configured.

Steps

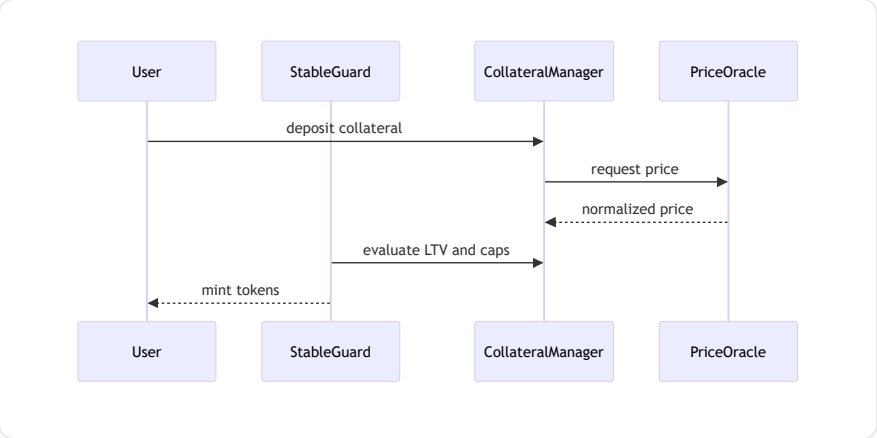
- Record deposit in CollateralManager (ETH or ERC20).
- Fetch normalized prices from the oracle.
- Compute collateral value and LTV; verify thresholds.
- Compute safe mint amount under caps and ratios.
- Mint to user account; update supply and debt.

Postconditions & Events

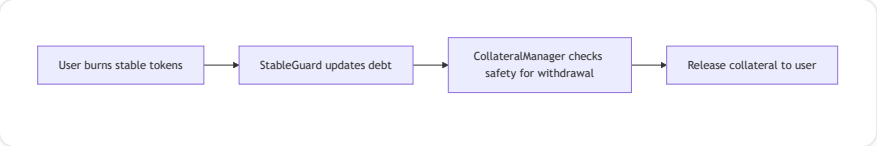
- Emit `Deposit` and `Mint` with metrics (value, LTV, caps applied).
- Protocol state consistent and auditable.

Failures & Reverts

- Revert if address/token invalid, price invalid/fallback not allowed, or LTV exceeded.
- Apply per-operation caps to prevent oversized mints.



10.6 Burn & Withdraw



Preconditions

- User holds sufficient tokens to burn.
- Collateral available without breaching thresholds after withdrawal.

Steps

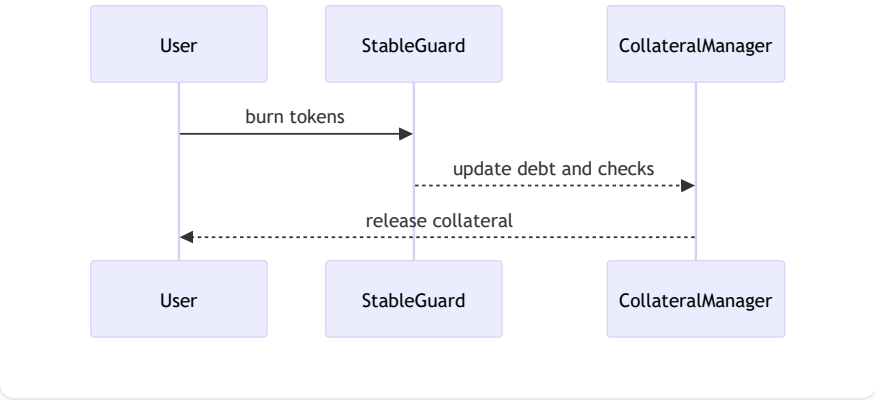
- Burn tokens and update user debt.
- Evaluate post-burn safety and withdrawal limits.
- Release collateral to user under constraints.

Postconditions & Events

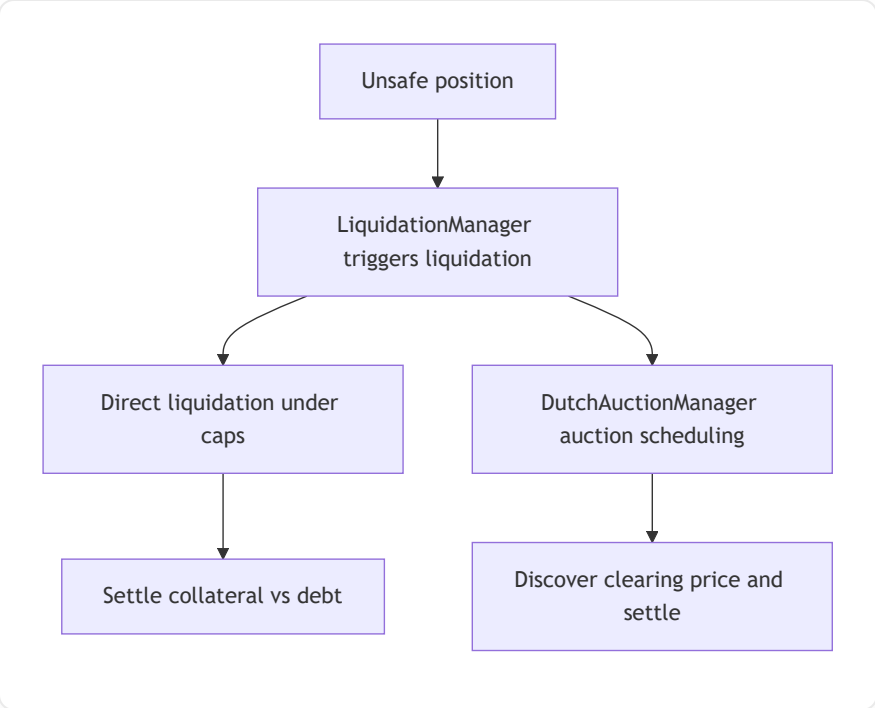
- Emit `Burn` and `Withdraw` with values and checks.

Failures & Reverts

- Revert if withdrawal would make the position unsafe or exceeds limits.



10.7 Liquidation — Direct vs Dutch Auction



Preconditions

- Position marked unsafe by LTV/threshold.
- Penalty and limit configuration active.

Steps — Direct liquidation

- Compute debt and collateral; apply penalties.
- Sell collateral or offset debt under caps.

- Settle balances and emit events.

Steps — Dutch auction

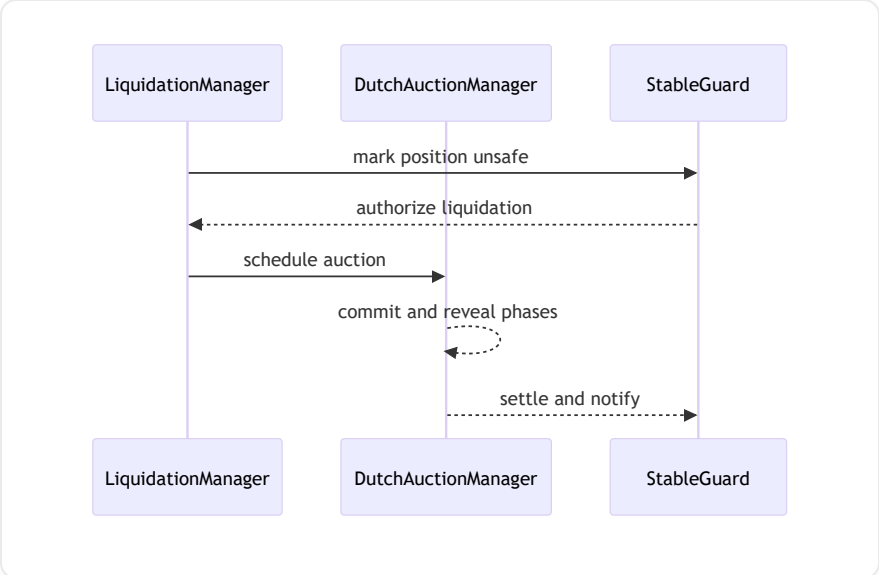
- Schedule auction with descending curve and fixed windows.
- Commit phase: register hashed bids.
- Reveal phase: validate and apply curve.
- Settlements and distribution of collateral and debt.

Postconditions & Events

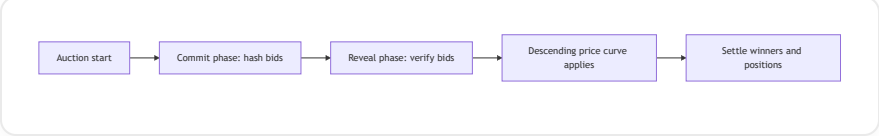
- Emit start, commit, reveal, settlement, and penalty events.

Failures & Reverts

- Revert invalid bids, inconsistent reveals, or exceeding system caps.



10.7.1 Dutch Auction — Commit-Reveal



- MEV awareness: commit-reveal prevents pre-trade exploitation.
- Price curve: decays over time to discover market-clearing value.

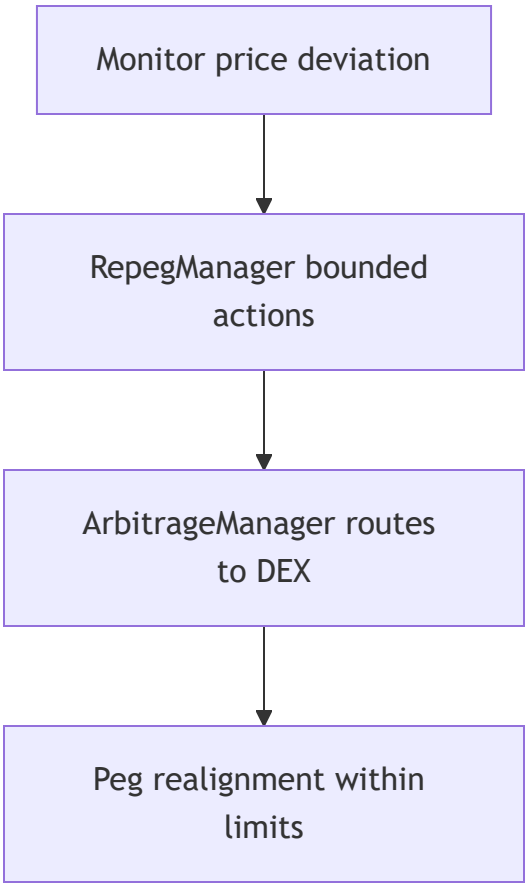
Key Parameters

- Time windows for commit and reveal.
- Time-parameterized descending price curve.
- Size caps per bid and per auction.

Failures & Mitigations

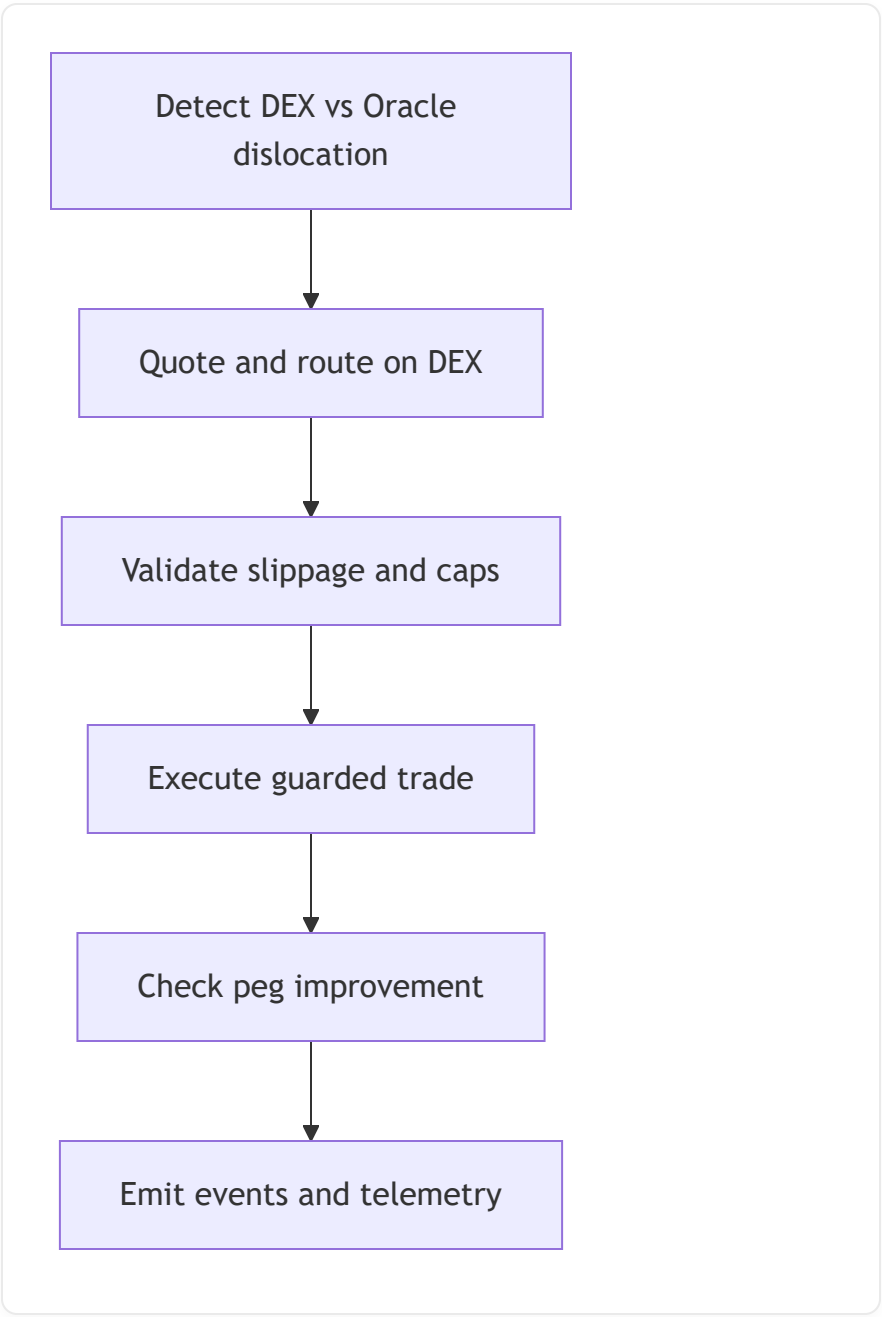
- Invalid reveal: reject; preserve commit evidence.
- Collusion: enforce limits and auditing via events.

10.8 Arbitrage & Repeg Coordination



- Bounded pressure: buy/sell under limits and cooldowns to avoid oscillations.
- Optional coordination: arbitrage aligns external pricing with oracle expectations.

10.9 Arbitrage Flow



- Dislocation detection: compare on-chain DEX prices vs validated oracle.

- Guarded execution: enforce slippage bounds, size caps, and cooldowns.
- Observability: emit events for monitoring and post-trade analysis.

Preconditions

- Oracle and DEX prices available and validated.
- Slippage, caps, and cooldown parameters active.

Steps

- Detect significant dislocation; compute route and quote.
- Validate limits and risk; execute trade under guards.
- Re-evaluate peg and record telemetry.

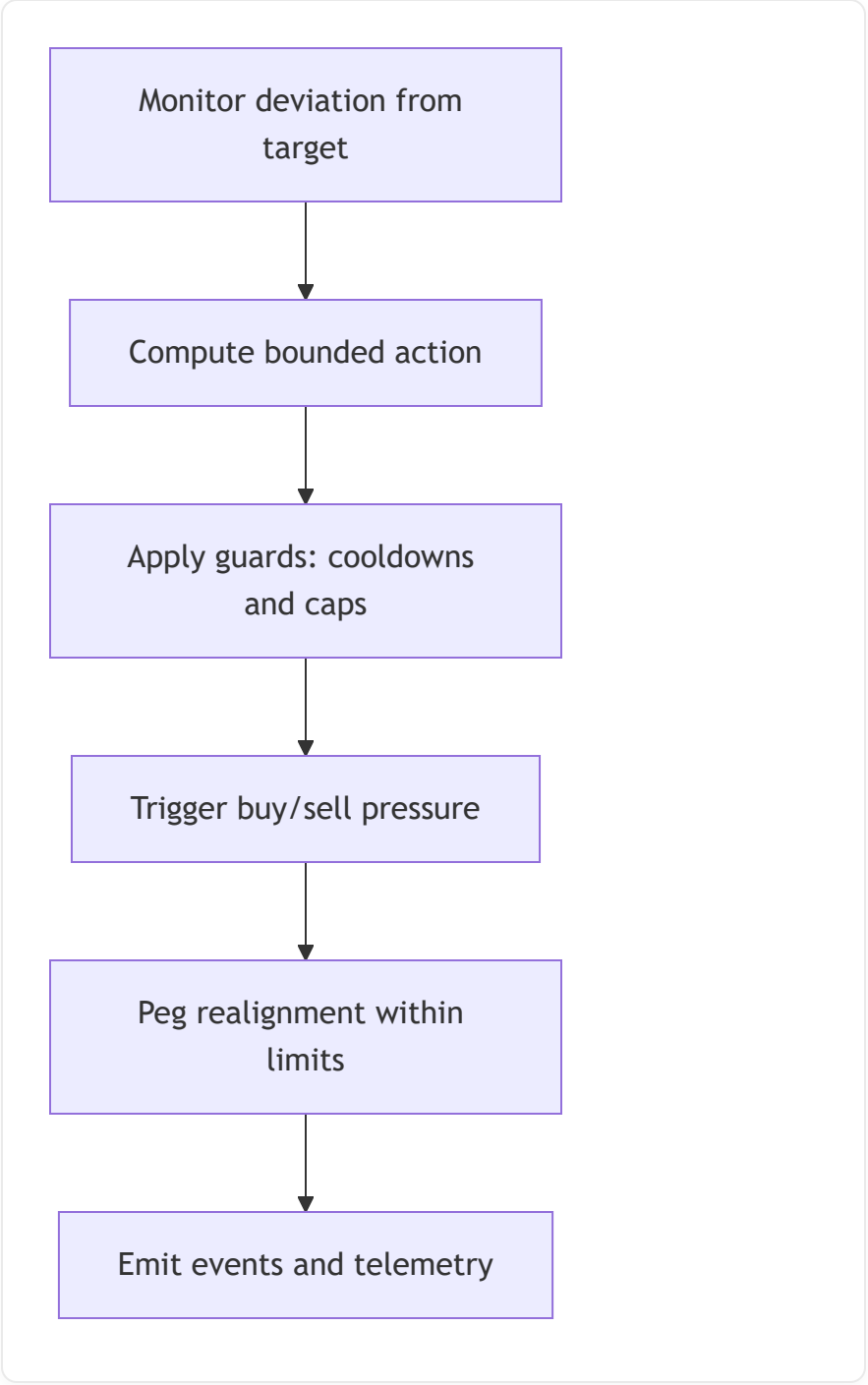
Postconditions & Events

- Emit detection, execution, and peg effect events.

Failures & Reverts

- Revert if slippage/caps violated or dislocation below minimum threshold.

10.10 Repeg Flow



- Bounded actions: avoid oscillations by limiting frequency and magnitude.
- Safety first: respect circuit-breakers and conservative defaults under stress.

Preconditions

- Deviation metrics configured; thresholds and limits defined.

Steps

- Measure deviation; decide bounded action (buy/sell).
- Apply cooldowns and caps; execute action.
- Measure peg improvement; adjust if necessary.

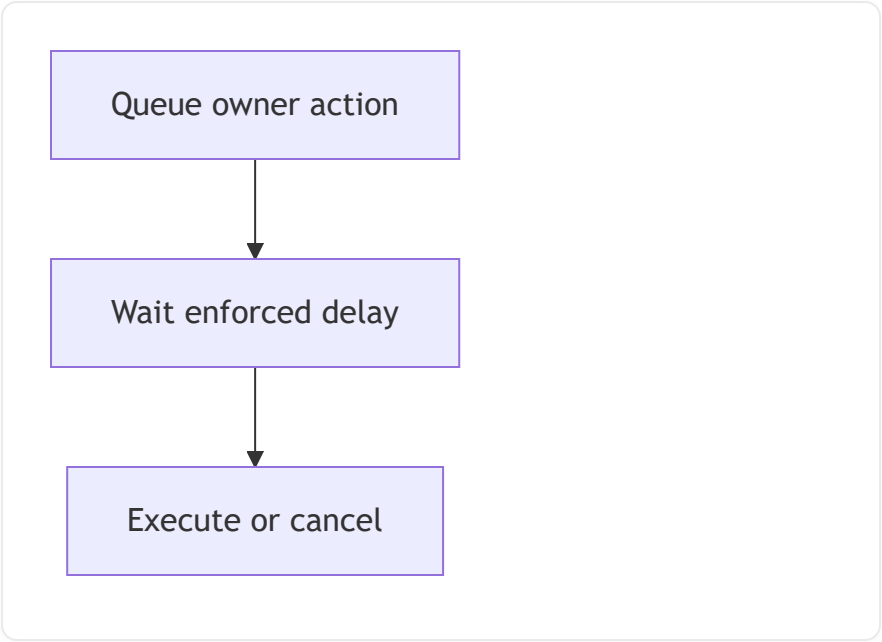
Postconditions & Events

- Emit events with action parameters and effects.

Failures & Reverts

- Do not execute if circuit-breaker active or deviation is transient.

10.11 Timelock — Delayed Sensitive Actions



- Governance-like safety: operators cannot instantly change critical parameters.

Preconditions

- Owner action recorded with hash, parameters, and target time.

Steps

- Queue action with `eta` and data; publish event.
- Wait minimum delay; allow cancellation within window.
- Execute within grace period; update state.

Postconditions & Events

- Emit `Queue`, `Execute`, and `Cancel` events.

Failures & Reverts

- Revert execution outside window or if parameters mismatch queued action.

11. Parameters & Configuration

- Ratios: LTV, liquidation thresholds, penalties.
- Limits: maximum operation sizes, slippage bounds, daily caps.
- Timing: cooldowns for repeg/arbitrage, timelock delays.
- Safety switches: circuit-breakers to pause modules under stress.
- Normalization: price decimals standardized to 18-decimals.

12. Governance & Parameterization

- Timelock authority: all sensitive parameter updates are queued and executed after a minimum delay,

with clear events for auditability.

- Parameter domains: ratios (LTV, penalties), thresholds (liquidation), execution limits (caps, slippage), timing (cooldowns, delays), and safety switches.
- Change management: adopt small, incremental changes and monitor outcomes; avoid large step changes that could destabilize the peg or liquidation efficiency.
- Emergency posture: use circuit-breakers to pause modules under stress; rely on conservative oracle fallback; prefer burns and deleveraging over aggressive repeg.
- Versioning: tag releases and track parameter sets; include revision notes in this whitepaper's history.

13. Security Model

- Access control: owner-only and role-guarded operations; restricted module wiring.
- Validation: strict checks on inputs and states to preempt unsafe transitions.
- Reentrancy protection: guarded functions, careful state ordering.
- Rate limiting: cooldowns and bounded actions to reduce abuse.
- Timelock defense: delays on sensitive actions for auditability and reaction time.
- Oracle safety: validation and fallback to trusted conservative prices.
- MEV protections: auction commit-reveal and descending curves mitigate manipulation.

- Error handling: revert paths on invalid states; predictable failure modes.

14. Economic Model

- Peg drivers: collateral valuation via normalized oracle prices and external liquidity via DEXs create a feedback loop where arbitrage aligns on-chain spot with target.
- Liquidation incentives: penalties and descending auction curves encourage timely participation and price discovery, reducing bad-debt risk.
- Bounded repeg: limited buy/sell actions apply gentle pressure to avoid overshooting and oscillations under adversarial conditions.
- Caps and slippage: operation size limits and slippage bounds internalize execution risk; large shifts are spread over time.
- Stress scenarios: under extreme volatility, minting is constrained, liquidations ramp up, and circuit-breakers favor stability over growth.

15. Protocol Invariants

- Collateralization: `collateral_value ⩾ required_collateral(value_of_debt, ratios)`.
- Minting constraints: mint only within configured safety bounds and caps.
- Burning constraints: debt reduces deterministically, supply updated consistently.

- Liquidation triggers: only when thresholds breached or invariants violated.
- Oracle usage: normalized and validated price or conservative fallback.
- Timelock guarantees: queued actions respect minimum delays before execution.

16. Threat Model & Mitigations

- Oracle failures: stale or invalid feeds — detect and switch to fallback; enforce bounds.
- Reentrancy attacks: guarded state transitions; no external calls before state updates.
- MEV exploitation: commit-reveal auctions; bounded operations; rate limiting.
- Parameter abuse: timelock delays and caps prevent sudden unsafe changes.
- Price manipulation: reconciliation with oracle and DEX signals; slippage controls.
- Denial of service: circuit-breakers and conservative defaults protect under stress.

17. Events & Telemetry

- Emissions on deposit, mint, burn, withdraw, liquidation steps, auction phases, repeg/arbitrage actions, oracle updates, and timelock lifecycle transitions.
- Observability allows external monitoring and post-mortem analysis of protocol decisions.

18. Performance & Gas Notes

- Storage packing: consolidate related values to reduce slot usage.
- Assembly in hot paths: minimize overhead for tight loops and validations.
- Avoid redundant normalization: cache normalized prices within a transaction when safe.
- Bounded iterations: guard against unbounded loops and heavy on-chain recomputation.

19. Testing Strategy

- Unit tests per module in `test/*.t.sol` covering normal and edge cases.
- Deterministic mocks for oracle and DEX interactions.
- Safety tests: threshold breaches, liquidation paths, timelock delays, fallback activation.
- Run locally: `forge install`, `forge build`, `forge test -vv`.

20. Repository Structure

```
 ├── .gitignore
├── LICENSE
├── README.md
├── foundry.lock
├── foundry.toml
├── src/
│   ├── ArbitrageManager.sol
│   ├── CollateralManager.sol
│   ├── Constants.sol
│   ├── DutchAuctionManager.sol
│   ├── LiquidationManager.sol
│   ├── PriceOracle.sol
│   ├── RepegManager.sol
│   ├── StableGuard.sol
│   ├── Timelock.sol
│   └── interfaces/
│       ├── AggregatorV3Interface.sol
│       ├── IArbitrageManager.sol
│       ├── ICollateralManager.sol
│       ├── IDutchAuctionManager.sol
│       ├── ILiquidationManager.sol
│       ├── IPriceOracle.sol
│       └── IRepegManager.sol
└── test/
    ├── ArbitrageManager.t.sol
    ├── CollateralManager.t.sol
    ├── DutchAuctionManager.t.sol
    ├── LiquidationManager.t.sol
    ├── PriceOracle.t.sol
    ├── RepegManager.t.sol
    ├── StableGuard.t.sol
    └── Timelock.t.sol
```

21. Operational Runbook

- Parameter updates: queue via timelock; wait the enforced delay; execute within grace period; verify events and resulting state.

- Oracle monitoring: track feed freshness and bounds; if invalid, ensure fallback engages; review telemetry for anomalies.
- Liquidation operations: monitor unsafe positions; trigger direct or auction flows; confirm settlements and penalty applications.
- Repeg/arbitrage cadence: respect cooldowns and caps; prefer small, frequent adjustments; halt under circuit-breaker.
- Incident response: pause modules via safety switches; communicate constraints; prioritize integrity (burns, deleveraging) over peg under sustained stress.

22. Build, Test, and Usage

- Dependencies: `forge install`
- Build: `forge build`
- Tests: `forge test -vv`
- Collaboration: `foundry.lock` pinned; `lib/` is not versioned and populated via `forge install`.

23. Glossary

- LTV: Loan-to-Value ratio.
- MEV: Miner/Maximal Extractable Value.
- Commit-Reveal: two-phase mechanism hiding bids until reveal.
- Timelock: enforced delay for sensitive actions.
- Fallback Price: conservative price used when primary feed fails validation.

24. References

- Foundry toolchain and testing.
- OpenZeppelin contracts and patterns.
- Chainlink-style aggregator interfaces and normalization practices.