TRAIL OF BITS

Advanced DeFi Invariants

Upcoming workshops

Beginner

- Part 1: The Basics
- Part 2: Breaking ABDKMath (Week of Nov 21, 2022)

Intermediate

- Part 3: Breaking Uniswap I (Week of Nov 28, 2022)
- Part 4: Breaking Uniswap II (Week of Dec 5, 2022)

Advanced

- Part 5: Advanced DeFi Invariants I (Week of Dec 12, 2022)
- **Part 6: Advanced DeFi Invariants II (Week of Dec 19, 2022)**

Who am I?

Nat Chin, Security Engineer II

Who You Should Follow

- Troy Sargent (@0xalpharush)
- Josselin Feist (@montyly)
- Anish Naik (@anishrnaik)
- Justin Jacob (@technovision99)

Who are we?

Trail of Bits (@trailofbits)

- We help developers to build safer software
- R&D focused: we use the latest program analysis techniques
- Slither, Echidna, Tealer, Amarna, solc-select, ..

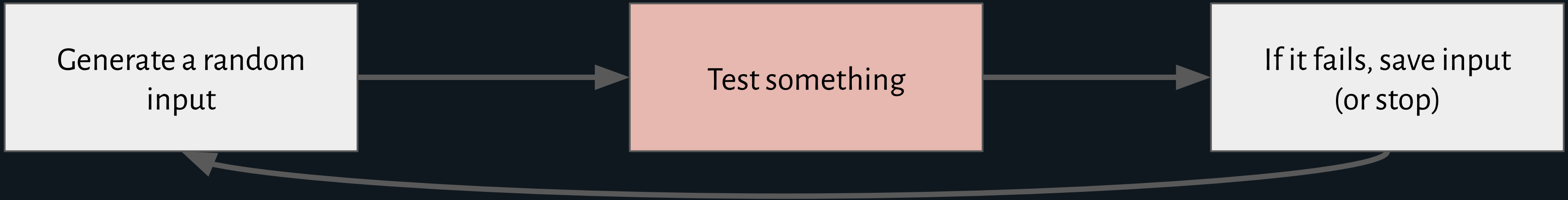
Agenda

- Recap: Fuzzing
- Recap: Primitive
- Architecture
- Finding invariants
- Writing basic invariants

So…how do I start fuzzing?

1. **Identify your invariants / system properties in English**
2. **Convert your properties to code**
3. **Run Echidna**
4. **FIND BUGS**

And… what is fuzzing?



Echidna vs Other Fuzzers

- **Echidna is more mature**
- **Allows testing of high gas assumptions**
- **Works with any compilation framework**
- **Supports various API's for testing**
- **Supports hevm/dapptool cheatcodes**

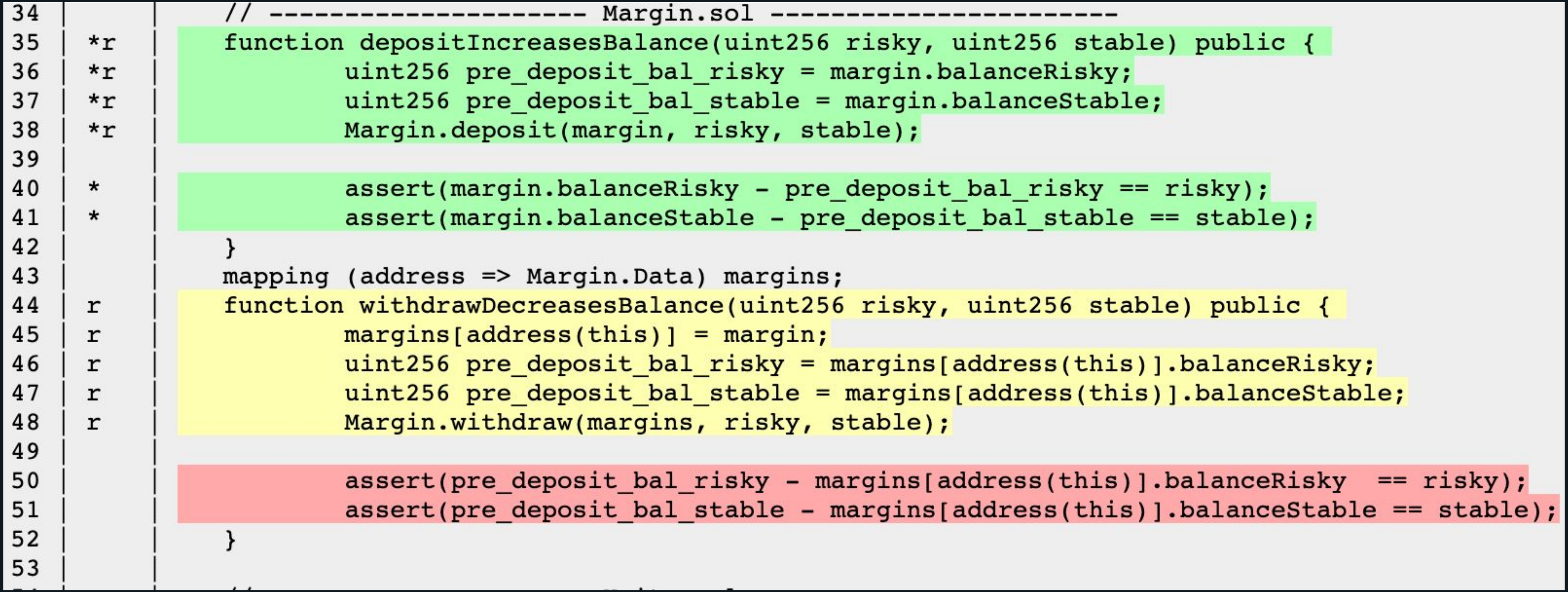
Tips on Identifying Invariants

- Start with the smallest component first
- Analyze all preconditions and postconditions
- Determine safe bounds of inputs
- Identify inversely-related functions
- Focus on the happy and unhappy paths

Useful Optimizations

- **Tests should have precondition, action, postcondition**
  - Pre-conditions: Scope the input space
  - Action: What we are testing
  - Post-conditions: The “truths” after the action

Coverage **is** your friend! Especially Echidna 2.0.4



```
34 |      |     // -------------------- Margin.sol ----------------------
35 | *r   |     function depositIncreasesBalance(uint256 risky, uint256 stable) public {
36 | *r   |             uint256 pre_deposit_bal_risky = margin.balanceRisky;
37 | *r   |             uint256 pre_deposit_bal_stable = margin.balanceStable;
38 | *r   |             Margin.deposit(margin, risky, stable);
39 |      |
40 | *    |             assert(margin.balanceRisky - pre_deposit_bal_risky == risky);
41 | *    |             assert(margin.balanceStable - pre_deposit_bal_stable == stable);
42 |      |     }
43 |      |     mapping (address => Margin.Data) margins;
44 | r    |     function withdrawDecreasesBalance(uint256 risky, uint256 stable) public {
45 | r    |             margins[address(this)] = margin;
46 | r    |             uint256 pre_deposit_bal_risky = margins[address(this)].balanceRisky;
47 | r    |             uint256 pre_deposit_bal_stable = margins[address(this)].balanceStable;
48 | r    |             Margin.withdraw(margins, risky, stable);
49 |      |
50 |      |             assert(pre_deposit_bal_risky - margins[address(this)].balanceRisky  == risky);
51 |      |             assert(pre_deposit_bal_stable - margins[address(this)].balanceStable == stable);
52 |      |     }
53 |      |
```

Disclaimer

- **This is complicated ✨**
- **Clone and follow along!**

Primitive

Uniswap vs Primitive

Uniswap

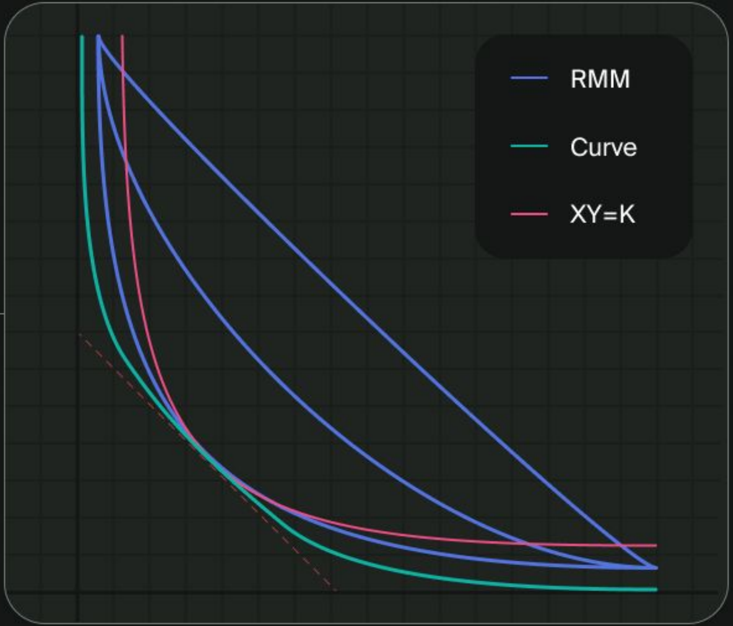
- Price changes on swap
- Pools don’t have a concept of time

Primitive

- Price changes on swap and over time
- At expiry, pool consists of an underlying token

Constant Function Market Makers

Tradng Functions

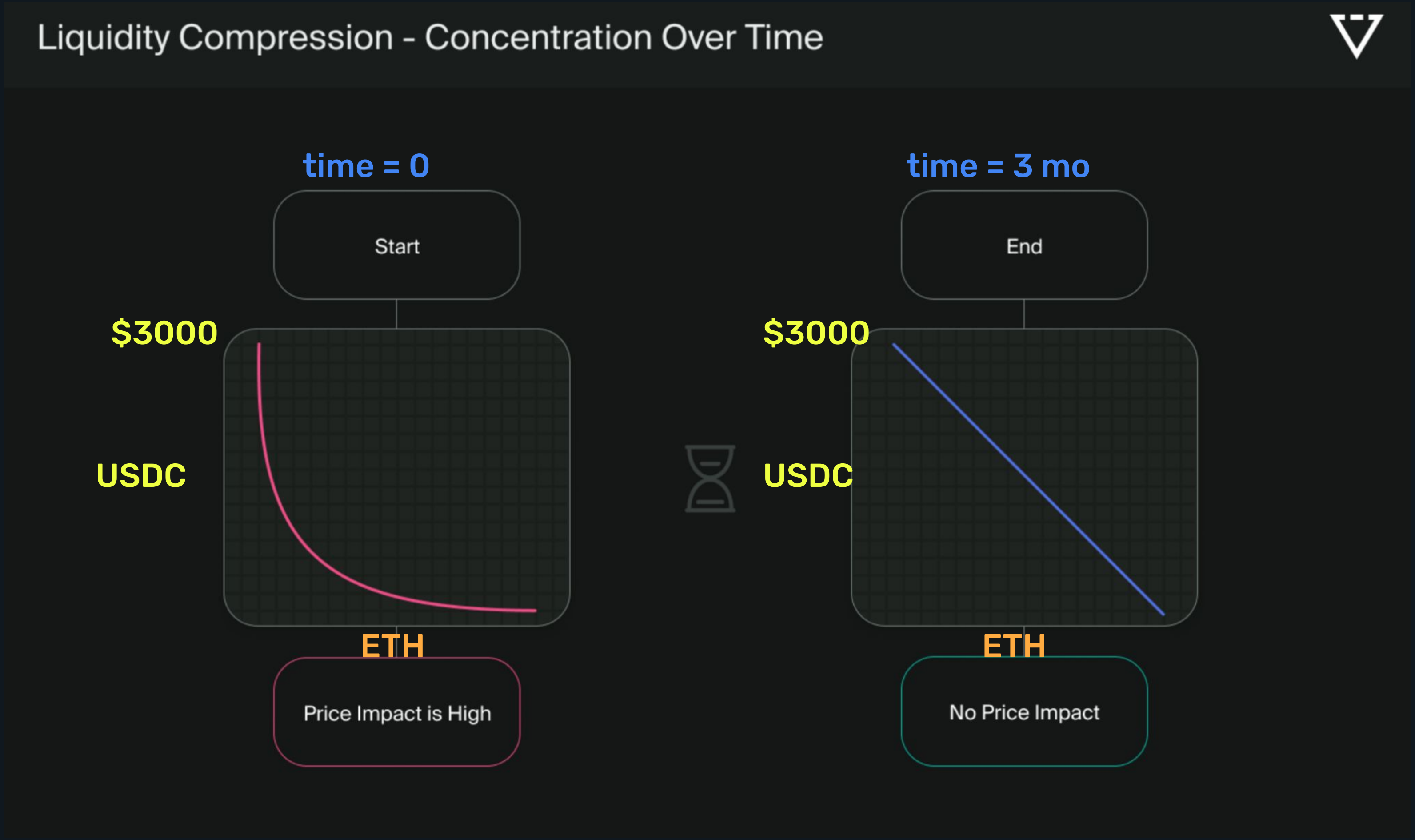


Features

- Allows creation of pools against 2 tokens
- Relies on spot price – **no oracles**
- Price curve continues to changes until expiry
- Price will converge to strike price at maturity

Concrete Example

- Pool consists of **USDC (underlying) - ETH (quote)***
- Strike price = 3000 USDC
- Maturity of **100 days**
- Implied volatility **150%**



Liquidity Compression - Concentration Over Time

time = 0

Start

Price Impact is High

time = 3 mo

End

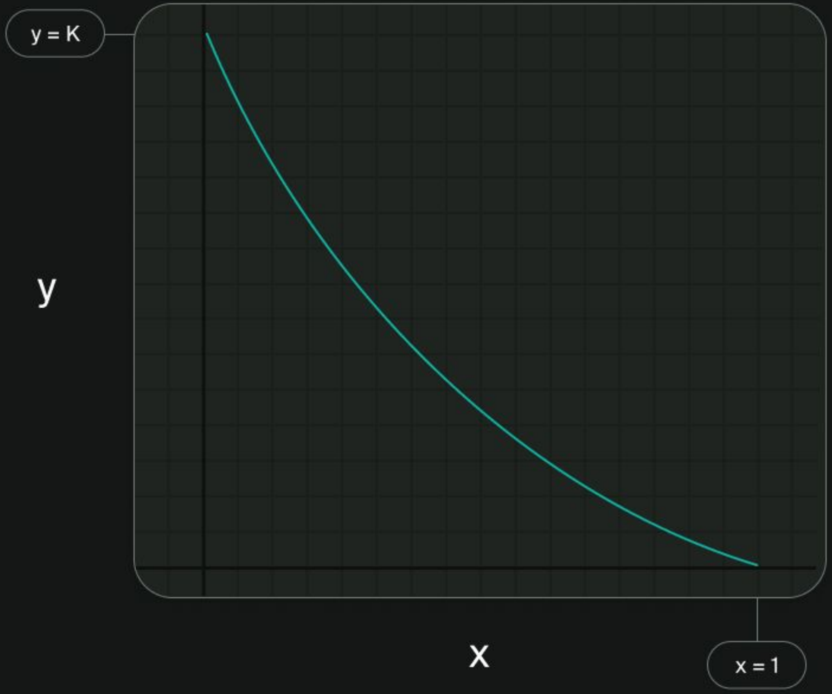
No Price Impact

Primitive RMM Curve

K	Strike Price	x	Underlying Asset Reserve
σ	Implied Volatility	y	Quote Asset Reserve
τ	Time until Expiry	Φ	CDF
k	Invariant	Φ^{-1}	Inverse CDF

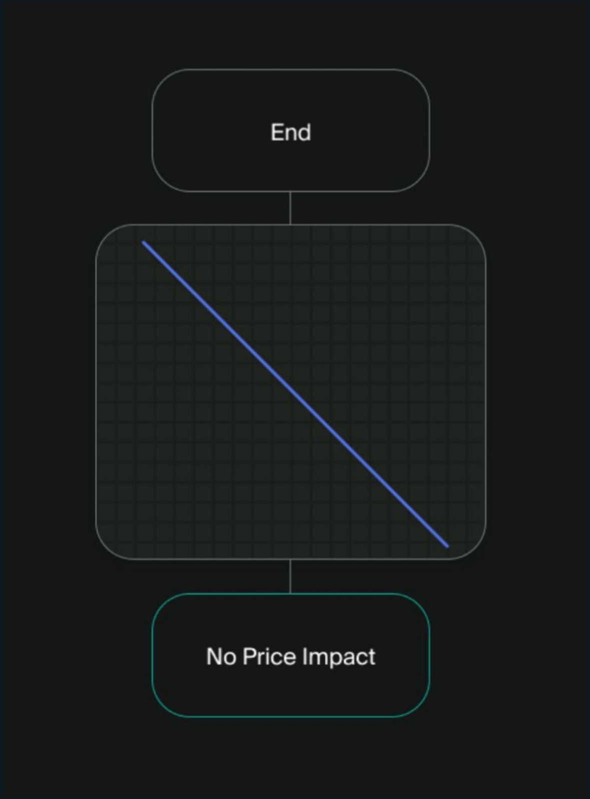
— $$y - K\Phi(\Phi^{-1}(1 - x) - \sigma\sqrt{\tau}) = k$$

Trading Function



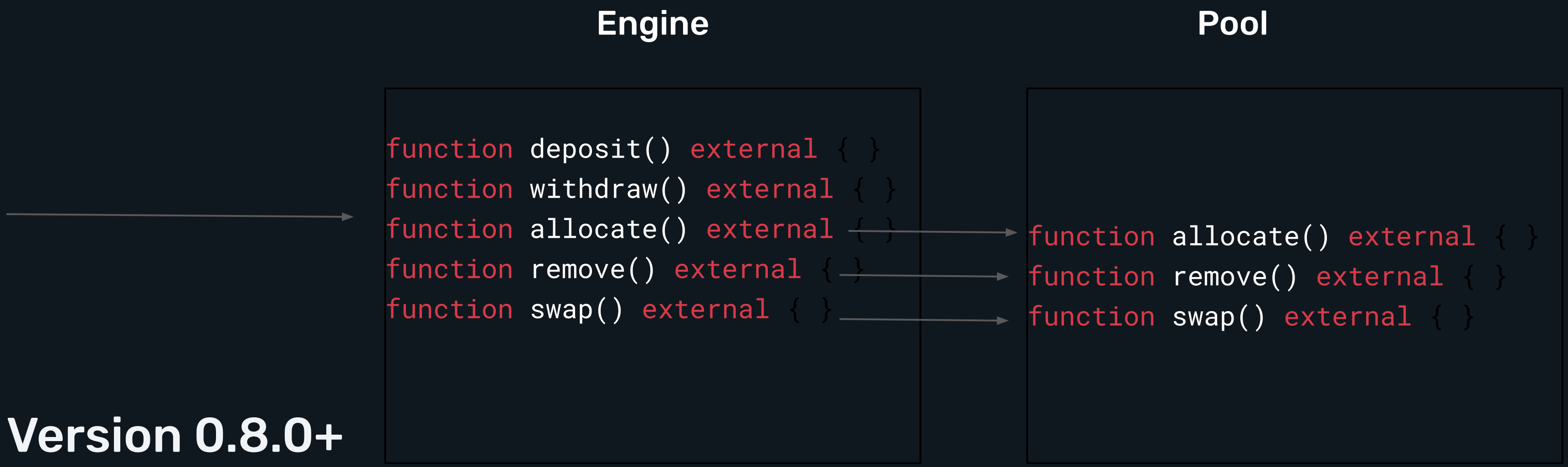
At maturity

- **Assets cannot be bought/sold**
- **Sells the second asset**
- **Pool prices at the strike price**



System Architecture

System Architecture



Engine

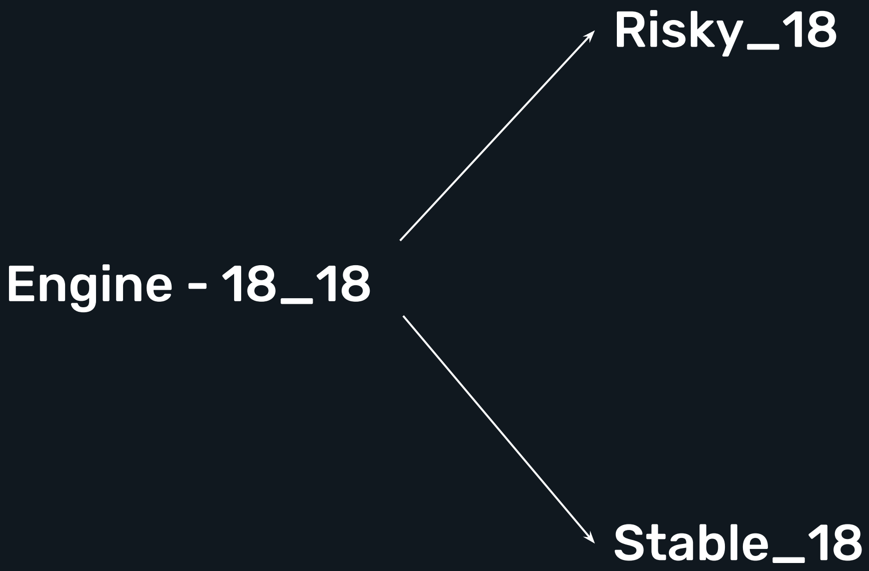
```
function deposit() external { }
function withdraw() external { }
function allocate() external { }
function remove() external { }
function swap() external { }
```

Pool

```
function allocate() external { }
function remove() external { }
function swap() external { }
```

Version 0.8.0+

Engine

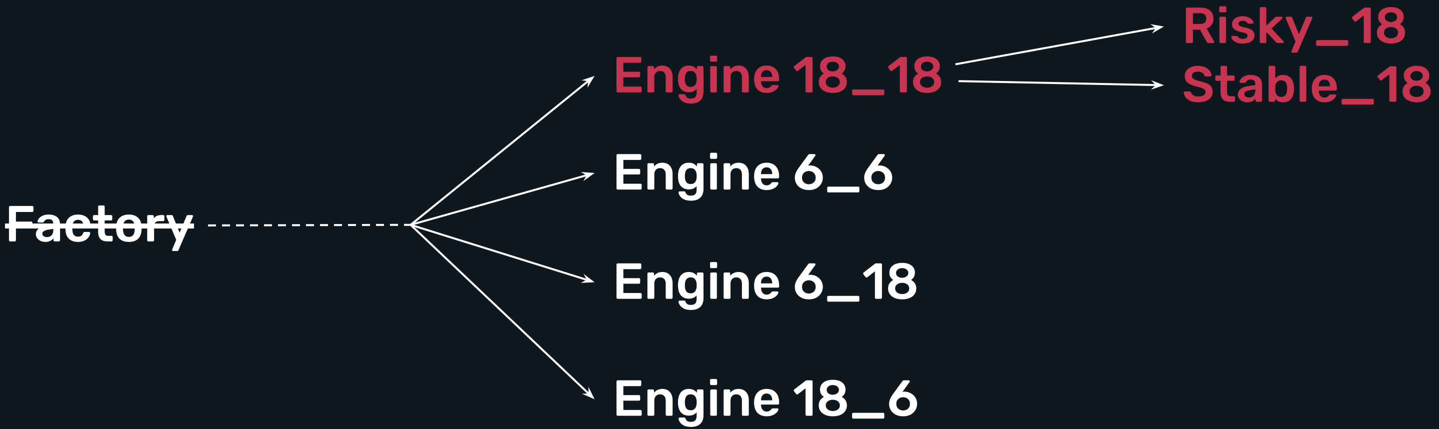


Engine - 18_18

Risky_18

Stable_18

Factory



~~Factory~~

Engine 18_18

Risky_18

Stable_18

Engine 6_6

Engine 6_18

Engine 18_6

System Setup

Etheno Setup

- Install Etheno: https://github.com/crytic/etheno
- Allows us to track deployments
- Test against an E2E script

```
pip3 install --user etheno
```

Why is it useful?

- Run a deployment script of the repo
- Save the addresses
- Tests against the instance each time

Internal Tests vs E2E Approach

Internal Tests

- Test in isolation
- Code auto-updates* *(usually)*

E2E Approach

- Allows you to explore entire code flows
- Requires re-initialization when target code changes

Primitive – Deposits/Withdrawals

Actors

- **Sender (msg.sender)**
- **Recipient**
- **Engine**

State Deposits – Withdrawals

Pre-Deposit	**Post-Deposit**
margins[recipient] = x	margins[recipient] = $x+\Delta$
sender risky, stable = a, b	sender risky, stable = $a-\Delta r$, $b-\Delta s$
engine risky, stable = c, d	engine risky, stable = $c+\Delta r$, $d+\Delta s$

No Changes in State

- No change in engine margins
- No change in sender margins
- No change in recipient token balance

No State Changes – Withdrawal

Pre-Withdraw	Post-Withdraw
margins[sender] = x	margins[sender] = x-Δ
recipient risky, stable = a, b	recipient risky, stable = a+Δr, b+Δs
engine risky, stable = c, d	engine risky, stable = c–Δr, d–Δs

No Changes in State – Withdrawal

- **No change in engine margins**
- **No change in recipient margins**
- **No change in sender token balance**

🎉🎉

Lessons Learned

- **Start with a monolithic contract**
- **Take a step back – and *think first***
- **Add preconditions on custom Natspec for clarity***

So on your own time……

- Implement more Primitive invariants
- Create PR's against echidna-streaming-series
- Check out Primitive's full E2E

Thank you for tuning in!

- **Look out for some new streams in the future!**
- **Use Echidna on your own codebase**