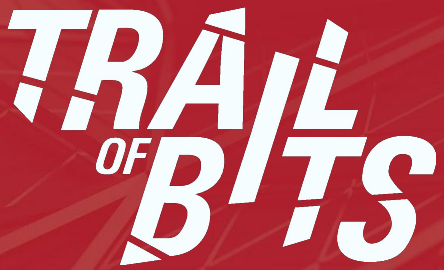


Echidna Workshop Series: Part 3

Justin Jacob - Security Engineer I

Previous & Upcoming Workshops

Beginner

- Part 1: The Basics (Week of Nov 16, 2022)
- Part 2: Breaking ABDK Math (Week of Nov 21, 2022)

Intermediate

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

Advanced

- Part 5: Breaking Primitive Finance I (Week of Dec 12, 2022)
- Part 6: Breaking Primitive Finance II (Week of Dec 19, 2022)

Who am I?

Justin Jacob, Security Engineer I

Who You Should Follow

- Troy Sargent (@0xalpharush)
- Josselin Feist (@montyly)
- Nat Chin (@0xicingdeath)
- Anish Naik (@anishrnaik)

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, ..

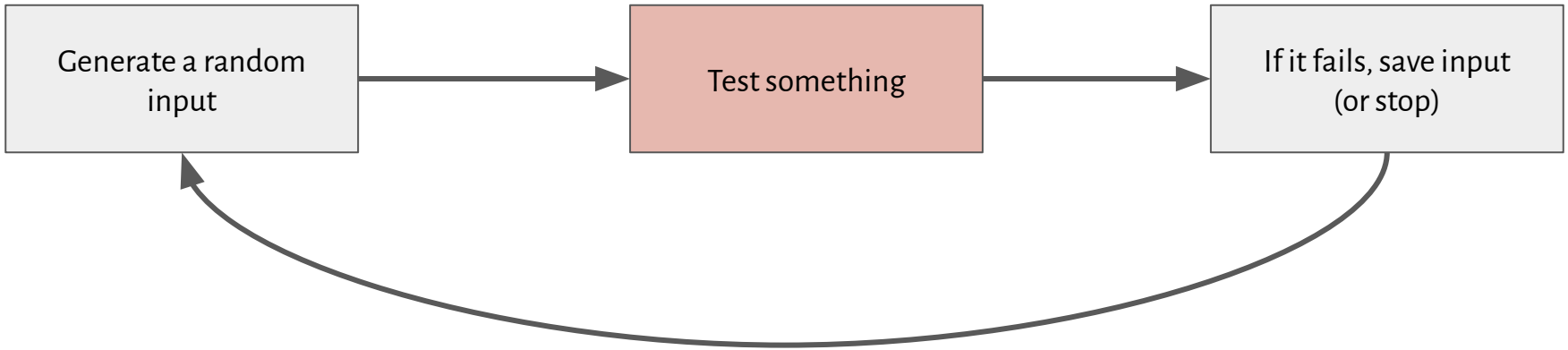
Today's Agenda:

- **Recap**
- **What is an AMM?**
  - How does it work?
  - Swaps, Liquidity Provisions
- **Go through the codebase**
- **Invariants and testing**

Recap -

- What is fuzzing?
- How do we fuzz contracts?
- What do we need to keep in mind when using Echidna?

What is fuzzing?



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

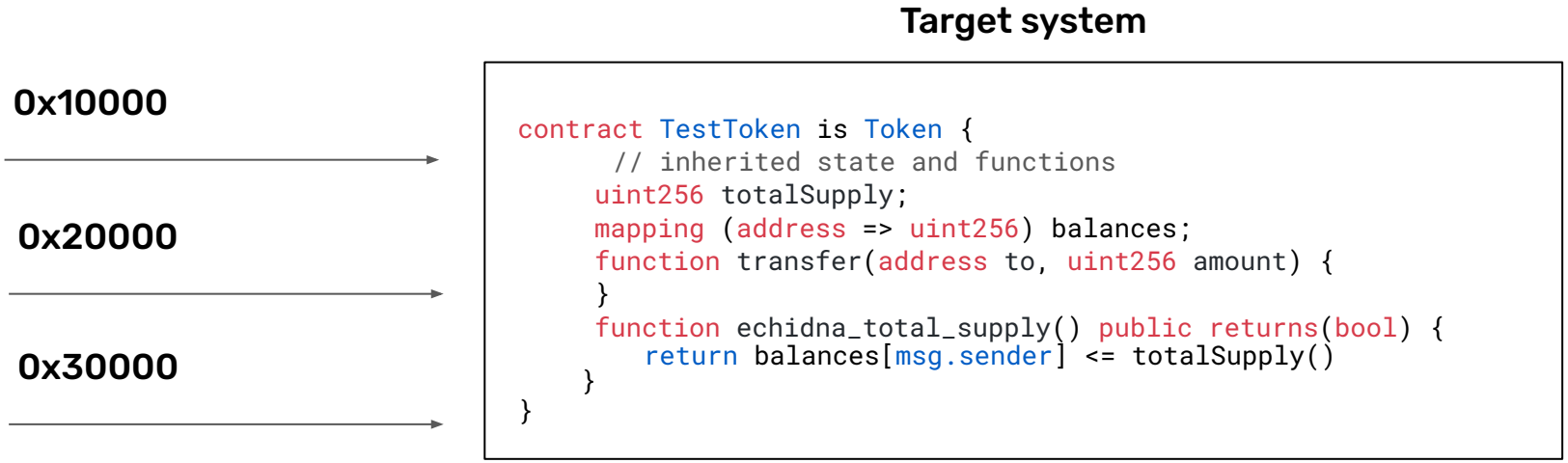
Useful optimizations:

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

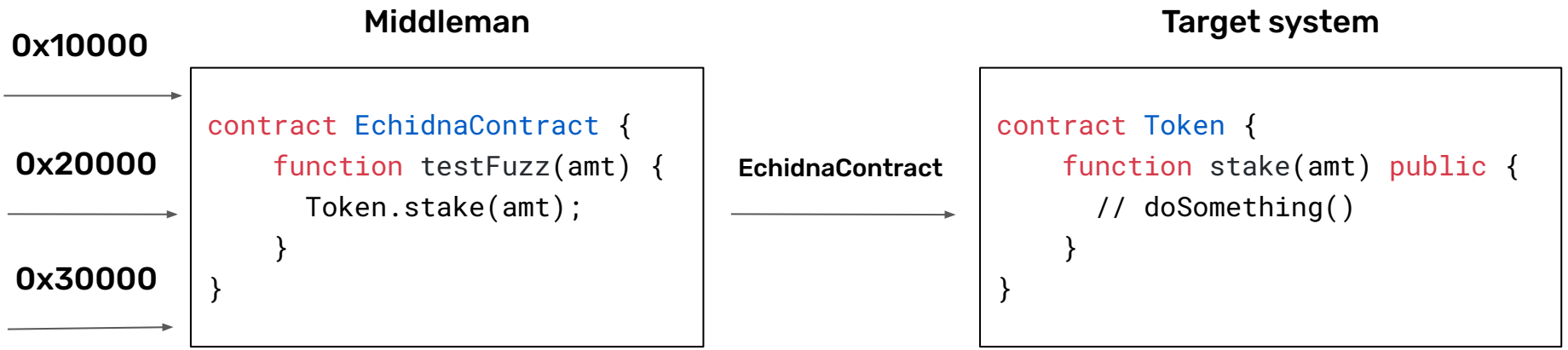
Useful optimizations:

- **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!**
  - Echidna saves coverage in corpus

Visualizing internal testing

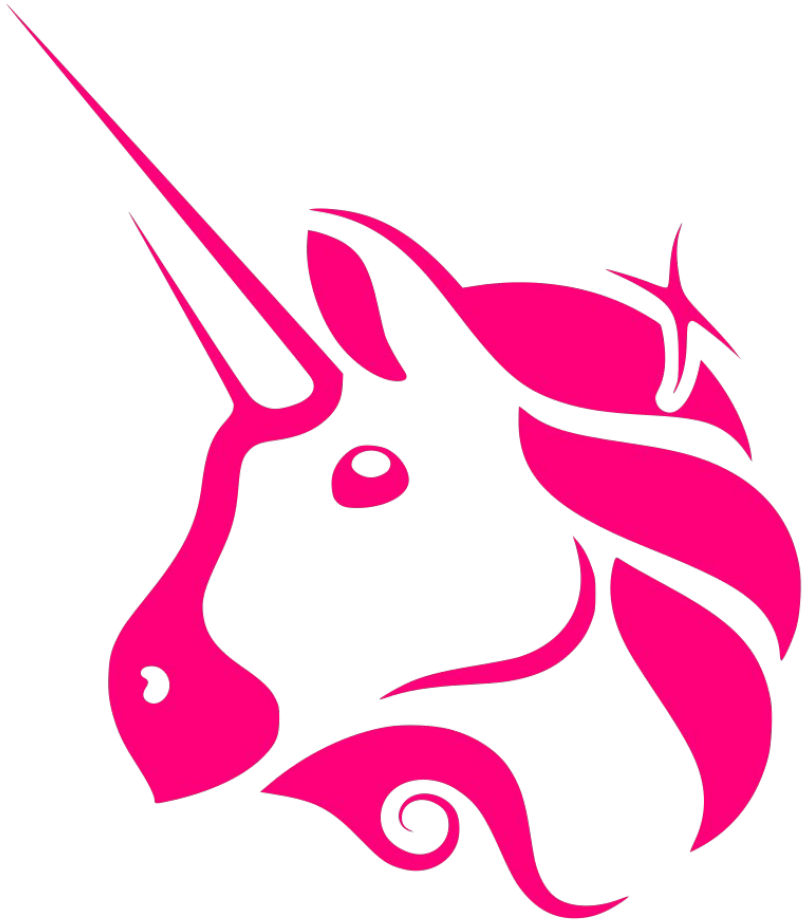


Visualizing External Testing



Today

Finding invariants of something very important...



What is an AMM?

- **Traditional Orderbook Model:**
  - Match buyers ⟷ sellers
  - Bid Price: max price buyer will pay
  - Ask Price: lowest price seller will sell
  - Market makers: add orders to orderbook
  - Market takers: execute orders

What is an AMM?

- **AMM Model:**
  - Exchange without orderbook
  - Pricing is based on pool's liquidity formula
    - Simplest example: xy = k (Uniswap!)
    - Price is calculated as ratio between two assets
  - Exchanges keep k (pool invariant) constant

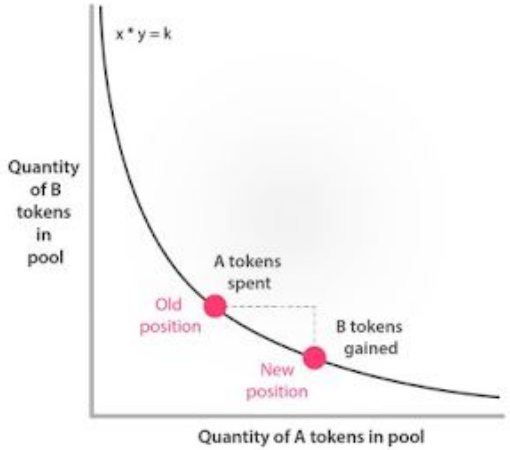
But why tho?

Orderbook model is really inefficient for blockchain!

- Have to find counterparties every time!
  - Keep track of all pending orders
  - Uses storage
  - Costs a lot of gas
- Need third party (market makers, takers)

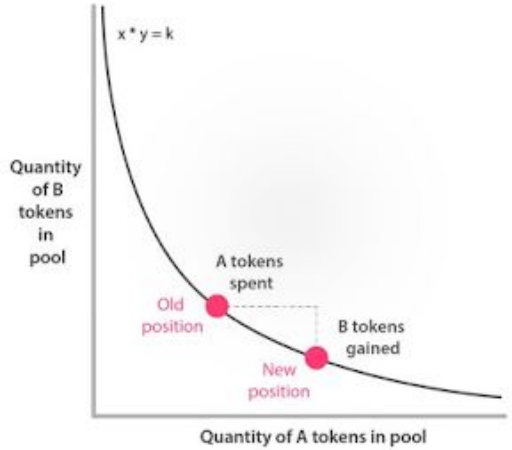
Example

- Swap x amt of tokenA for tokenB
- <u>How much of token B do we get out?</u>



Example

- **Swap x amt of tokenA for tokenB**
- **<u>How much of token B do we get out?</u>**
- **<u>Solution:</u>**
  - We have x*y=k
  - All trades keep k constant
  - Therefore …



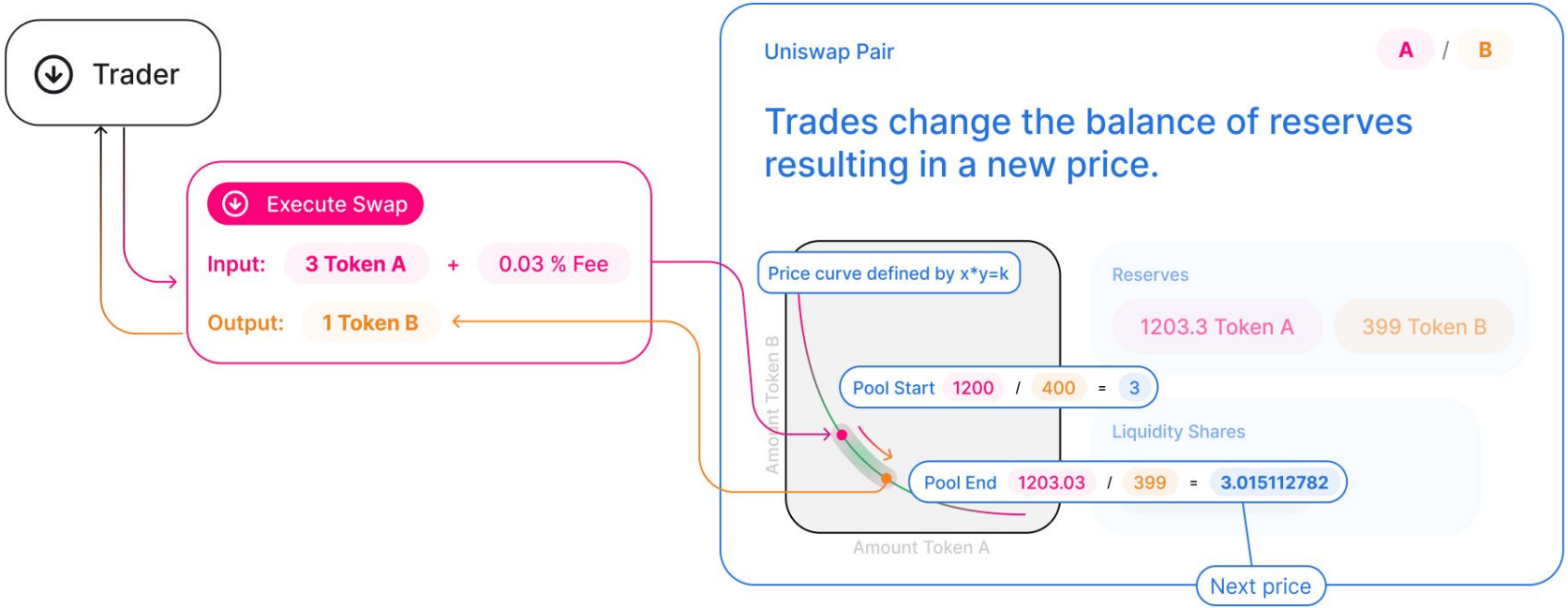
Solution:

Let Δx be the amount we are swapping token A for Δy of token b.

x * y = k

(x + Δx) * (y - Δy) = k (the pool gains Δx, loses Δy)

Δy = y - (k / (x + Δx))


Trader
Execute Swap
Input:
3 Token A
+
0.03 % Fee
Output:
1 Token B
Uniswap Pair
A
/
B
Trades change the balance of reserves resulting in a new price.
Price curve defined by x*y=k
Amount Token B
Amount Token A
Reserves
1203.3 Token A
399 Token B
Pool Start
1200
/
400
=
3
Liquidity Shares
Pool End
1203.03
/
399
=
3.015112782
Next price

Liquidity Providers (LPs)

- People provide ratio of tokens to the pool, get minted special LP token
- This LP token represents provided liquidity to the pool
- Initial LP Provider sets k => sets token price
- To get tokens back, must burn these LP tokens

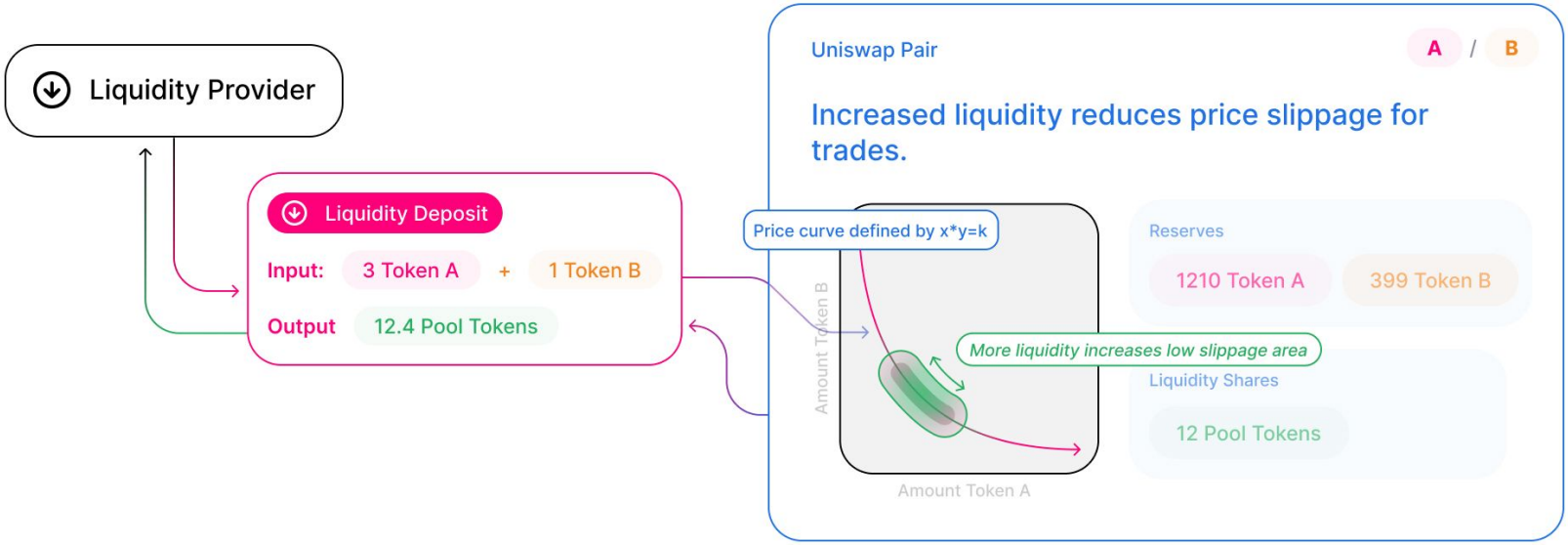
Why should I?

- **In practice, there are fees on each swap**
- **This fee goes back to the LPs**
  - Passive way to make money!

Example

- **Bob wants to create a DAI/WETH pool:**
  - Uni V2
  - He puts up 100000 DAI and 1 WETH
  - Therefore he gets minted sqrt(100000 *1) ≈ 316 DAI-WETH LP tokens

- **Bob is encouraged to put up “fair price” of DAI/WETH**
  - If not, there is an arbitrage opportunity and pool will be rebalanced


Liquidity Provider
Liquidity Deposit
Input:
3 Token A
+
1 Token B
Output
12.4 Pool Tokens
Uniswap Pair
A
/
B
Increased liquidity reduces price slippage for trades.
Price curve defined by x*y=k
Amount Token B
More liquidity increases low slippage area
Amount Token A
Reserves
1210 Token A
399 Token B
Liquidity Shares
12 Pool Tokens

Impermanent Loss

- Impermanent loss: $ value of the LP's share decrease if the tokens change price
- In practice, trading fees should account for this

Now...

Let’s examine the code!

Core

- **Where the main logic of uniswap resides**
- **Extremely minimal and simplistic**
  - Uniswap themselves say it is “quite minimal, possibly even brutalist”

Core

- Two contracts: factory and pairs

Core

- **Two contracts: factory and pairs**
- **Factory: creates pairs**
  - Creates unique pair contracts for each pool via CREATE2
  - Also has logic to turn on fees

Core

- **Two contracts: factory and pairs**
- **Factory: creates pairs**
  - Creates unique pair contracts for each pool via CREATE2
  - Also has logic to turn on fees
- **Pairs:**
  - Represent liquidity pool, keep track of token balances
  - Also an ERC20 token
  - Contains the basic swapping logic

Now for the fun part...

Let’s find some invariants!

LP Invariants:

- **Providing liquidity increases invariant**
  - x*y =k => increasing x and y increases k!

Homework:

Think of more invariants!

Write your own tests and make PRs!

Thanks everyone for attending!

See you next week!