Demystifying Fuzzing

Nat Chin

Hi! I’m Nat!

- Blockchain Security Engineer at Trail of Bits
- I figure out where things can break
- Fell down the rabbit hole in 2017
- Former smart contract developer & blockchain professor
- Author of solc-select
- Twitter: @0xicingdeath

Agenda

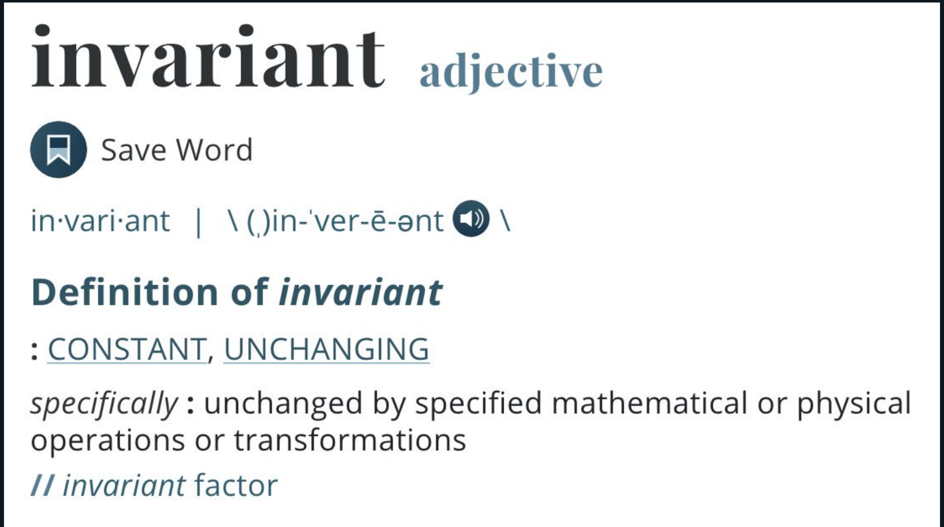
- Defining Invariants
- Writing properties
- Fuzzing!
- *Finding fun bugs*

Fuzzing

- Define assumptions meant to hold true
- Exploration of contracts with randomized arguments
- Checks dangerous contract states

What’s an invariant?

- “Invariant Testing” = “Property Testing”
- System properties that should always be true



invariant adjective

Save Word

in·vari·ant | \ (ˌ)in-ˈver-ē-ənt 🔊 \

Definition of *invariant*

: CONSTANT, UNCHANGING

specifically : unchanged by specified mathematical or physical operations or transformations

// *invariant* factor

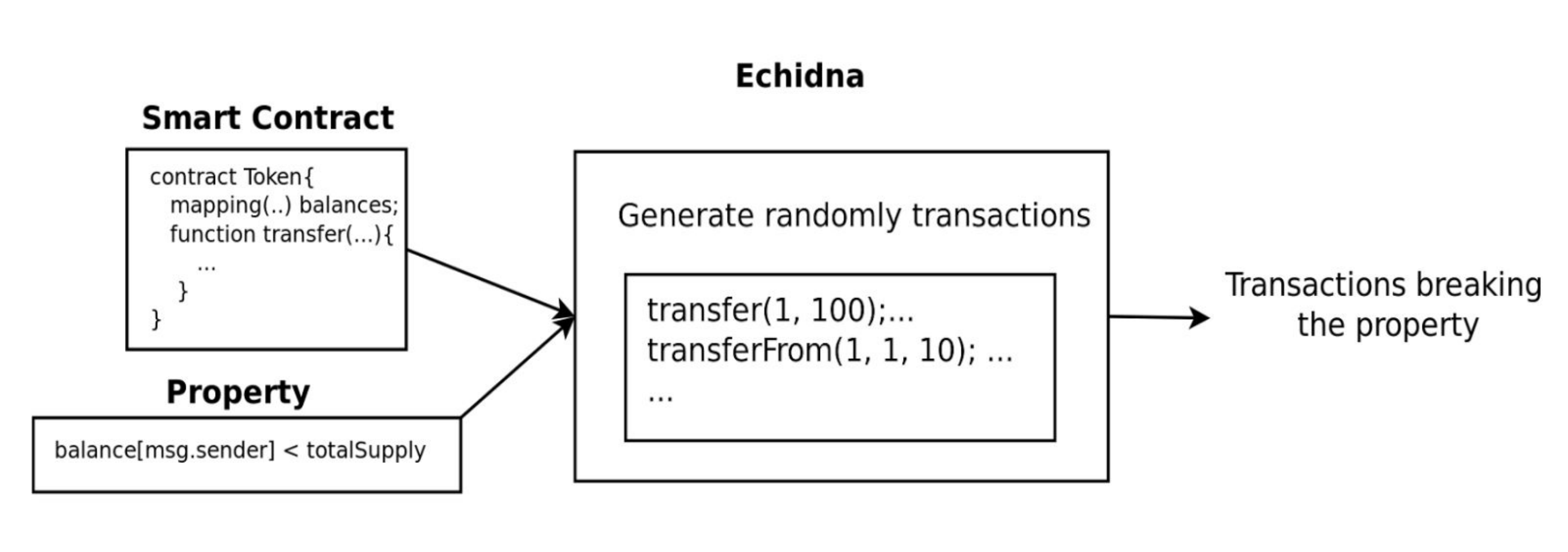
Fuzzing is Easy!

```
for value in [0, 255] {
    call function;
    if invariant is broken {
        profit
    }
}
```

But It’s Also Hard!

- Even with constraint smart contracts, astronomical search space
- What if the invariant is only broken for a single, unique input?
- Multiple accounts/contracts interacting with each other?

Echidna



Echidna

Smart Contract

```
contract Token{
  mapping(..) balances;
  function transfer(...){
     ...
   }
}
```

Property

```
balance[msg.sender] < totalSupply
```

Generate randomly transactions

```
transfer(1, 100);...
transferFrom(1, 1, 10); ...
...
```

Transactions breaking the property

How do I start?

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

1: Identify your Invariants

IN ENGLISH WORDS.

Invariants

- They’re everywhere!
- Token Invariants
- Mathematical invariants

Token Invariants – Total Supply

User balance never exceeds total supply

Token Invariants – Transfer

Users cannot transfer more than they own

Mathematical Invariant – Association

$$1 + 2 = 3$$

$$2 + 1 = 3$$

Mathematical Invariant – Identity

$$1 * 2 = 2$$

$$0 + x = x$$

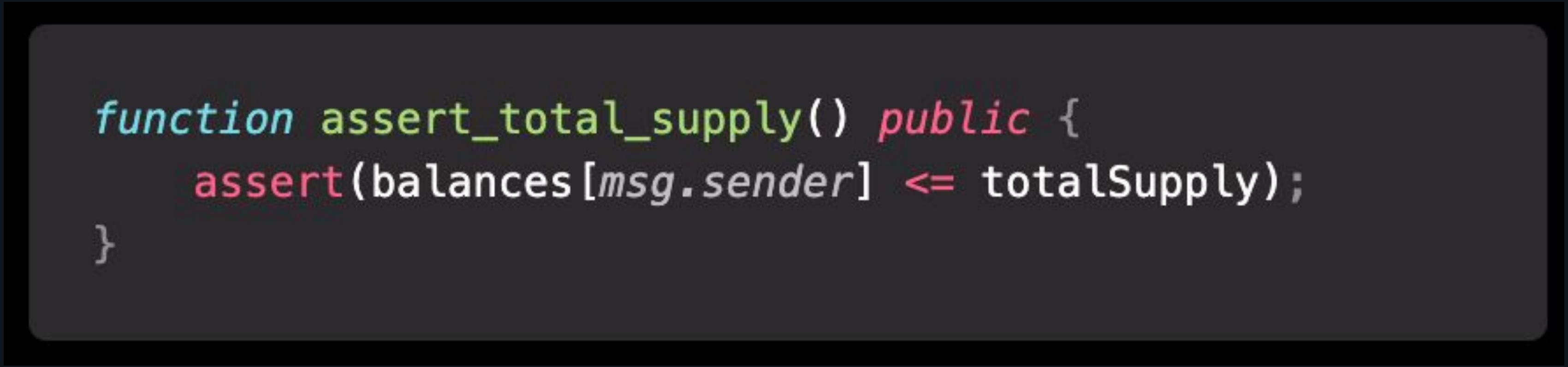
Mathematical Invariant – Addition / Subtraction

$$x + 5 - 5 = x$$

2: Convert into Code

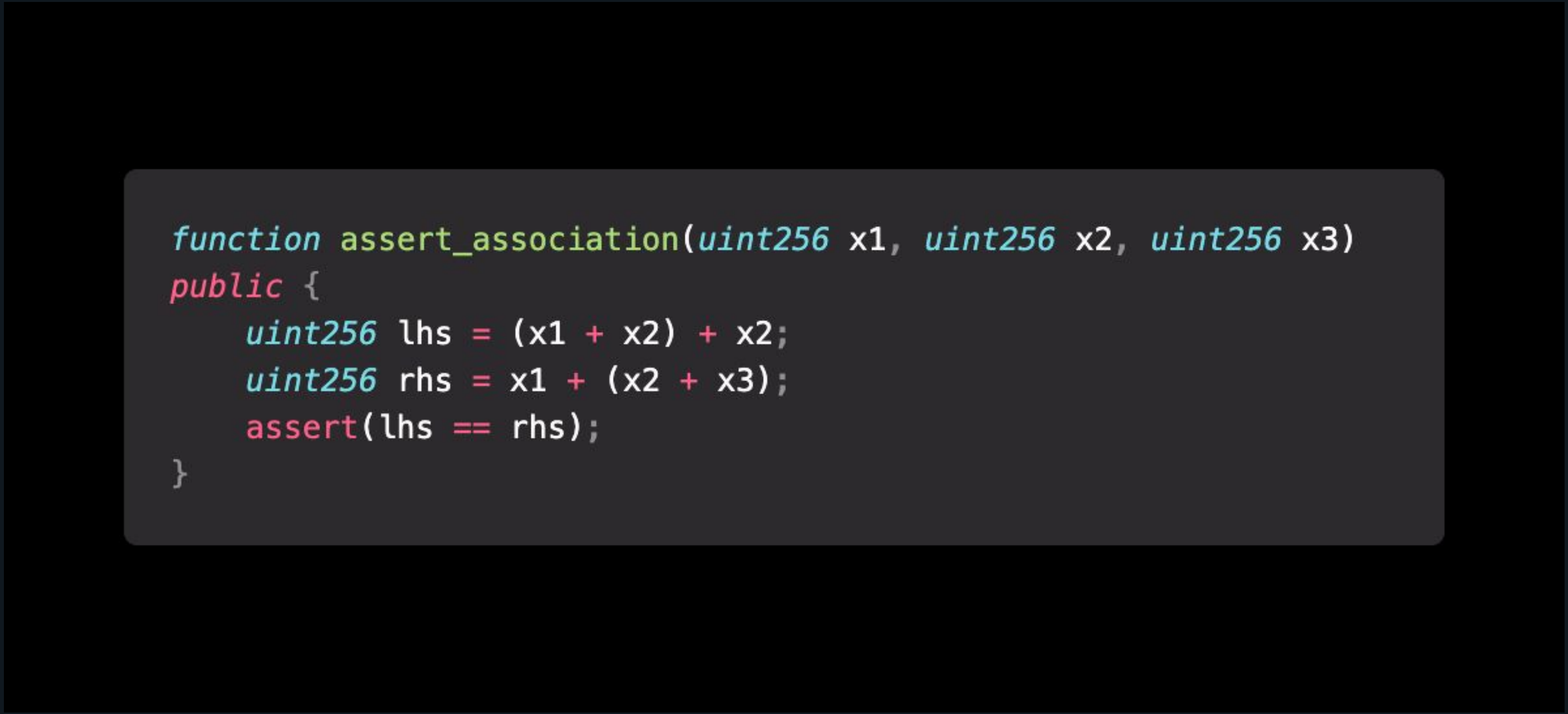
IT’S EASIER THAN IT SOUNDS.

Token Invariants – Total Supply



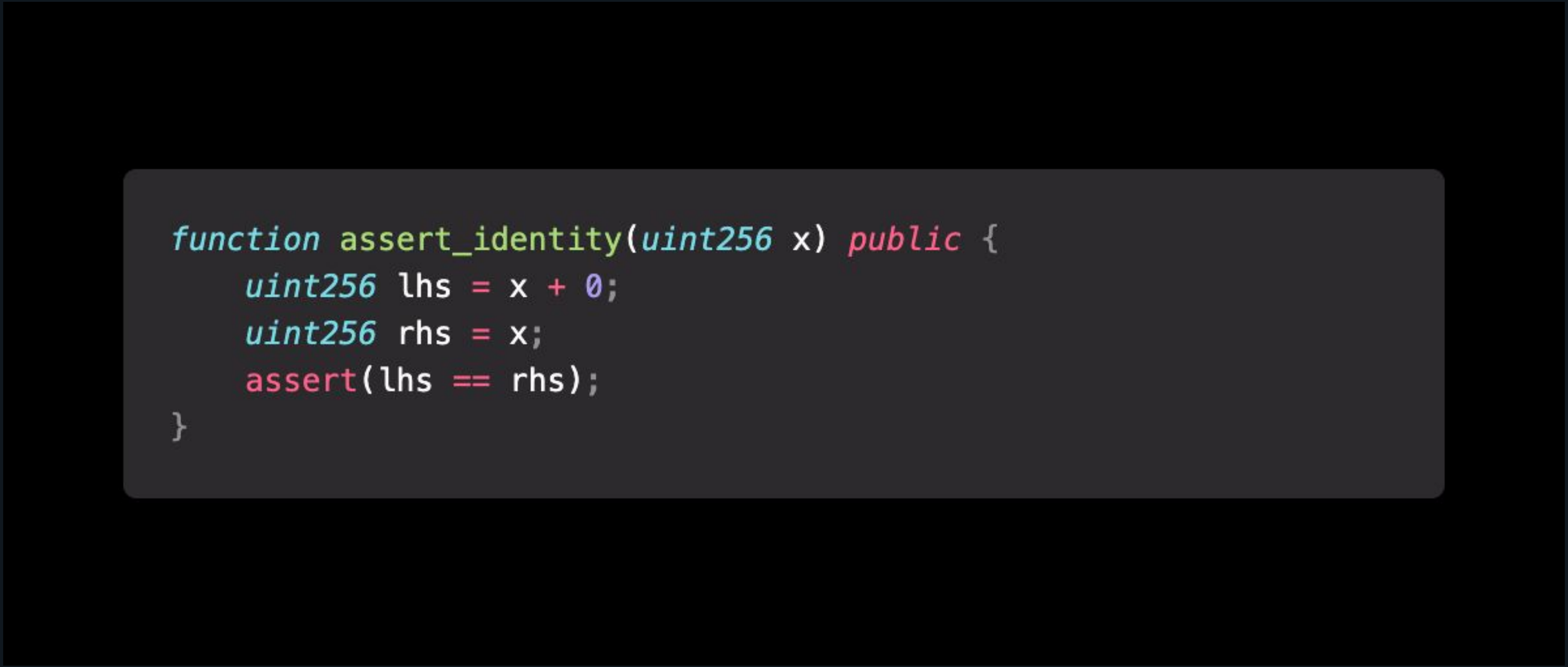
```
function assert_total_supply() public {
    assert(balances[msg.sender] <= totalSupply);
}
```

Mathematical Invariant – Association



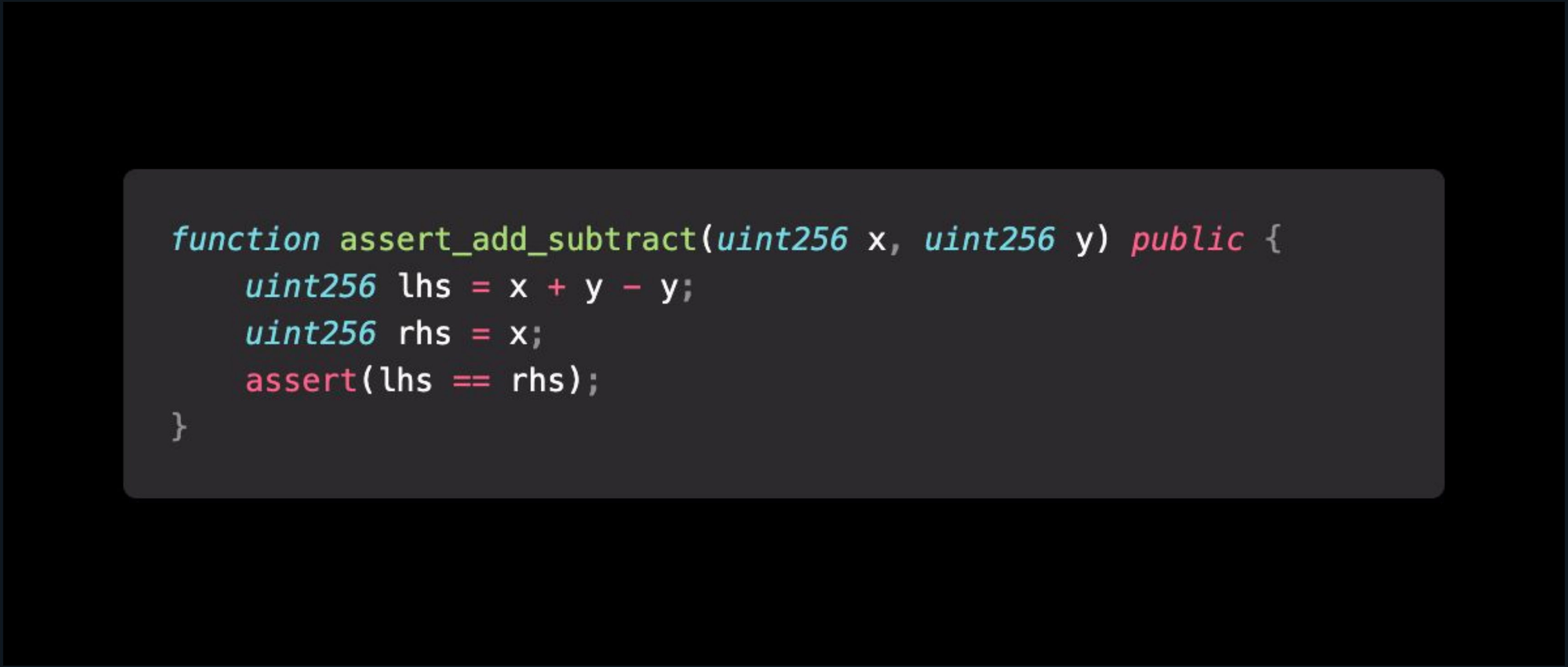
```
function assert_association(uint256 x1, uint256 x2, uint256 x3)
public {
    uint256 lhs = (x1 + x2) + x2;
    uint256 rhs = x1 + (x2 + x3);
    assert(lhs == rhs);
}
```

Mathematical Invariant – Identity



```
function assert_identity(uint256 x) public {
    uint256 lhs = x + 0;
    uint256 rhs = x;
    assert(lhs == rhs);
}
```

Mathematical Invariant – Addition / Subtraction



```
function assert_add_subtract(uint256 x, uint256 y) public {
    uint256 lhs = x + y - y;
    uint256 rhs = x;
    assert(lhs == rhs);
}
```

Example - rmm-core

Liquidity Pools

- Allocate assets into the pool
- Remove assets from the pool
- Swap assets

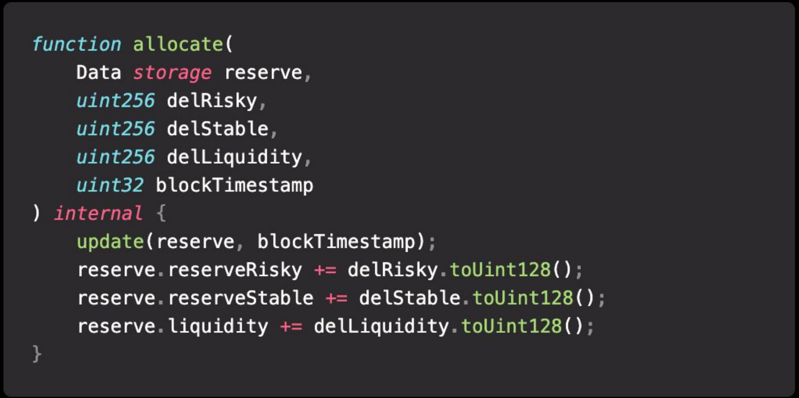
Liquidity Pools

- Initial pool balance: x
- Deposit: 5
- Withdraw: 5

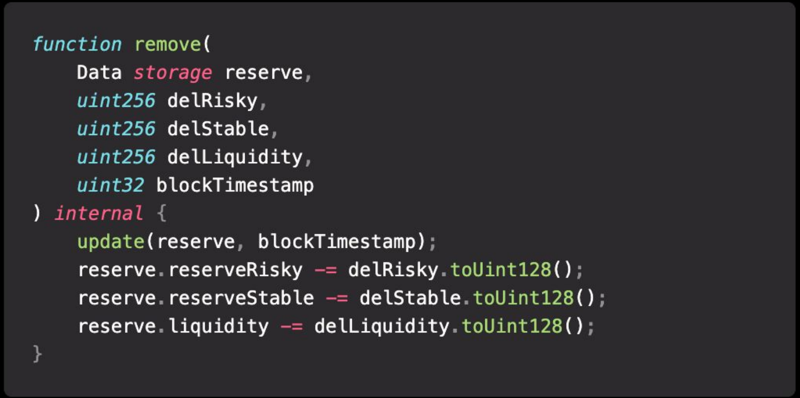
$$\begin{array}{r} x \\ +\,5 \\ (5) \\ \hline x \end{array}$$

What value do you expect the pool balance to be?

Allocate/Remove Functions



```
function allocate(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky += delRisky.toUint128();
    reserve.reserveStable += delStable.toUint128();
    reserve.liquidity += delLiquidity.toUint128();
}
```

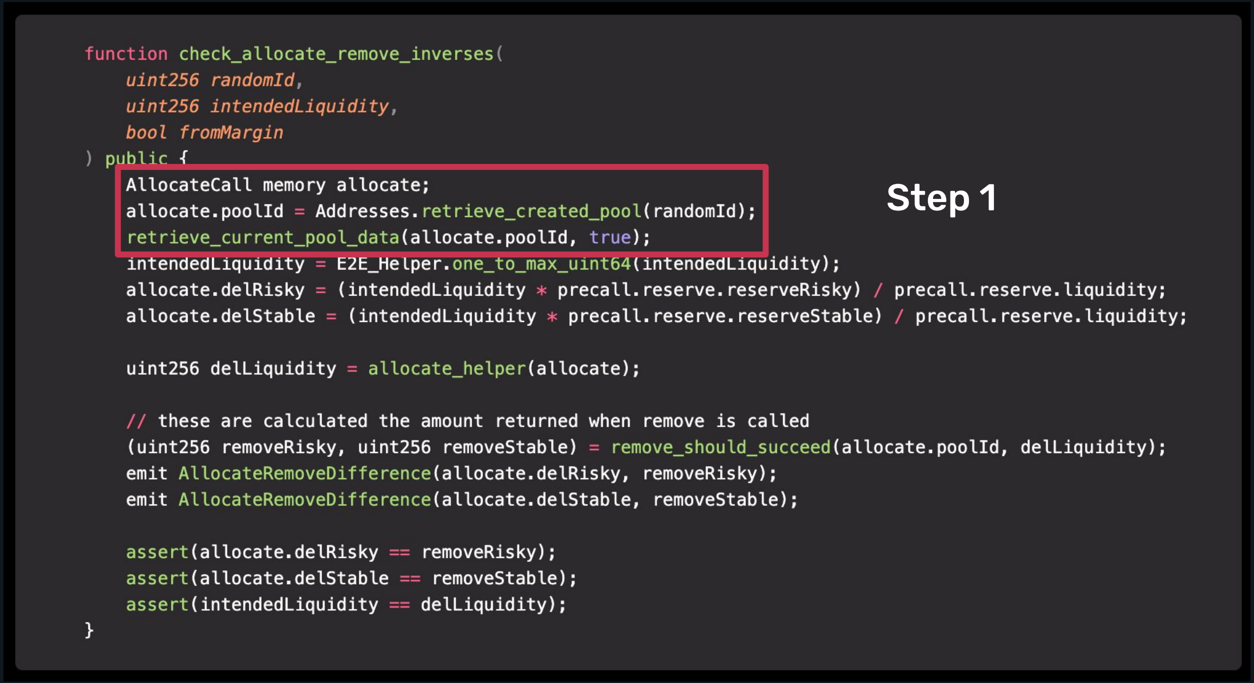


```
function remove(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky -= delRisky.toUint128();
    reserve.reserveStable -= delStable.toUint128();
    reserve.liquidity -= delLiquidity.toUint128();
}
```

What should the test do?

1. Start with initial reserve and liquidity balance
2. Allocate funds into the system
3. Remove funds from the system
4. Balance before and after transactions should be equal

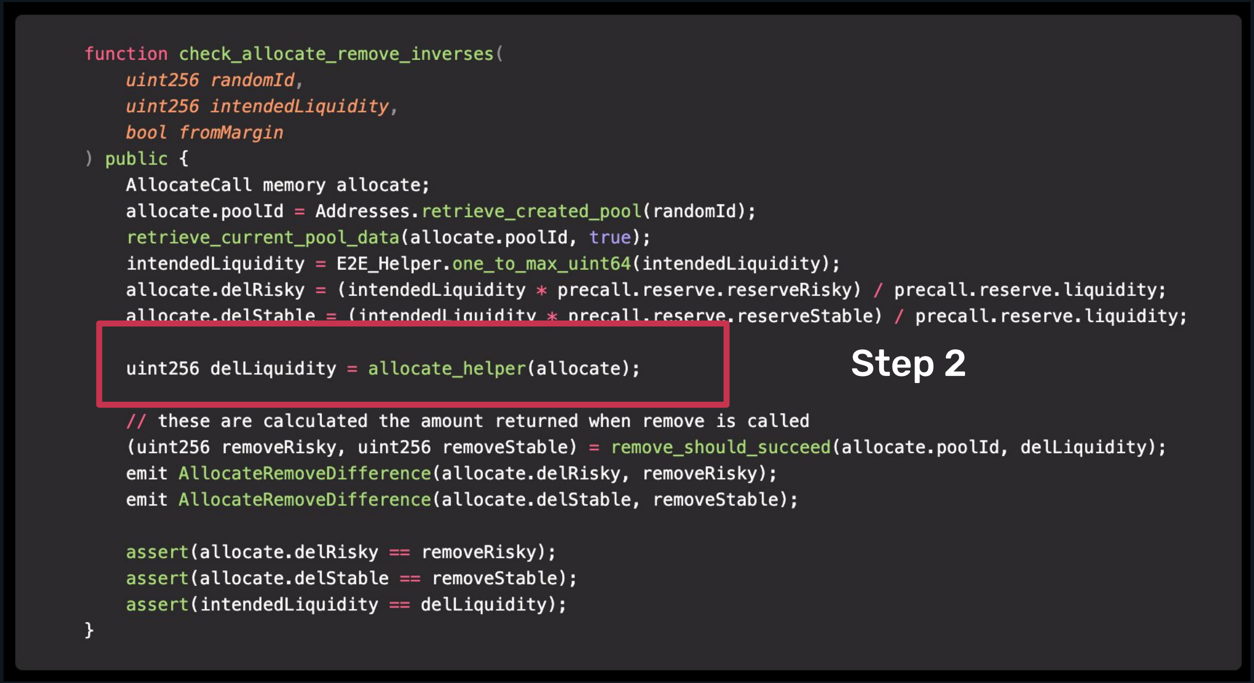
Invariant Test



```
function check_allocate_remove_inverses(
    uint256 randomId,
    uint256 intendedLiquidity,
    bool fromMargin
) public {
    AllocateCall memory allocate;
    allocate.poolId = Addresses.retrieve_created_pool(randomId);
    retrieve_current_pool_data(allocate.poolId, true);
    intendedLiquidity = E2E_Helper.one_to_max_uint64(intendedLiquidity);
    allocate.delRisky = (intendedLiquidity * precall.reserve.reserveRisky) / precall.reserve.liquidity;
    allocate.delStable = (intendedLiquidity * precall.reserve.reserveStable) / precall.reserve.liquidity;

    uint256 delLiquidity = allocate_helper(allocate);

    // these are calculated the amount returned when remove is called
    (uint256 removeRisky, uint256 removeStable) = remove_should_succeed(allocate.poolId, delLiquidity);
    emit AllocateRemoveDifference(allocate.delRisky, removeRisky);
    emit AllocateRemoveDifference(allocate.delStable, removeStable);

    assert(allocate.delRisky == removeRisky);
    assert(allocate.delStable == removeStable);
    assert(intendedLiquidity == delLiquidity);
}
```

Step 1

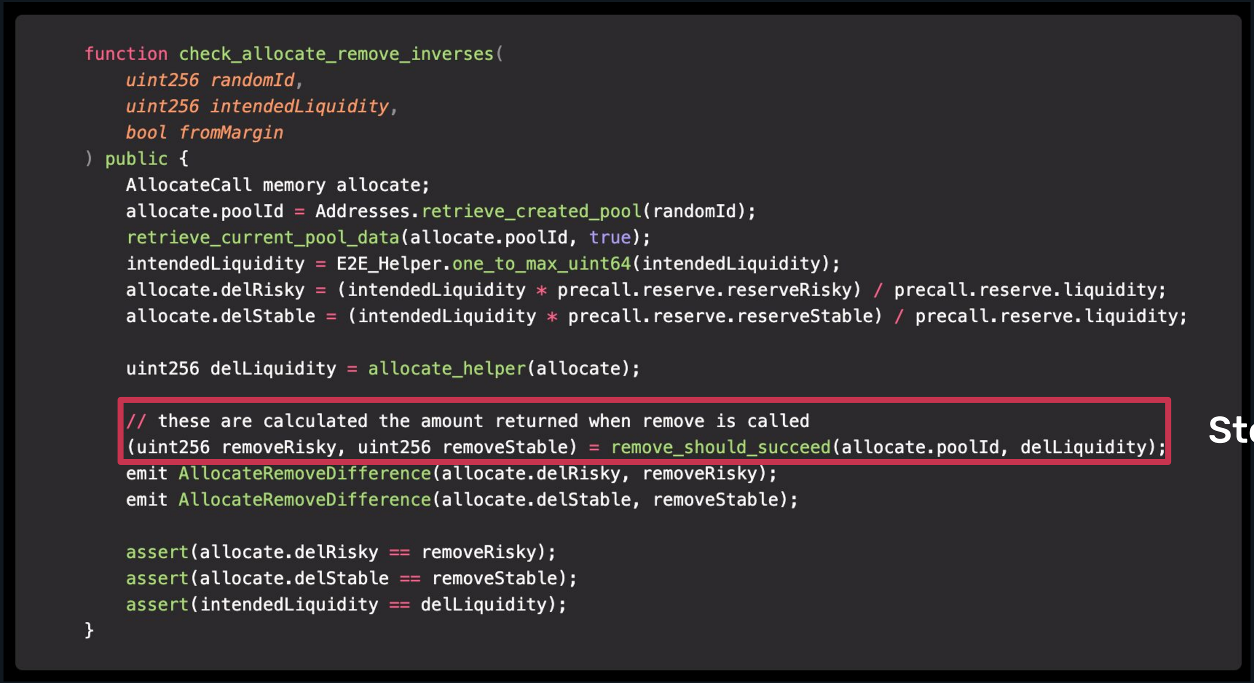
Invariant Test



```
function check_allocate_remove_inverses(
    uint256 randomId,
    uint256 intendedLiquidity,
    bool fromMargin
) public {
    AllocateCall memory allocate;
    allocate.poolId = Addresses.retrieve_created_pool(randomId);
    retrieve_current_pool_data(allocate.poolId, true);
    intendedLiquidity = E2E_Helper.one_to_max_uint64(intendedLiquidity);
    allocate.delRisky = (intendedLiquidity * precall.reserve.reserveRisky) / precall.reserve.liquidity;
    allocate.delStable = (intendedLiquidity * precall.reserve.reserveStable) / precall.reserve.liquidity;

    uint256 delLiquidity = allocate_helper(allocate);

    // these are calculated the amount returned when remove is called
    (uint256 removeRisky, uint256 removeStable) = remove_should_succeed(allocate.poolId, delLiquidity);
    emit AllocateRemoveDifference(allocate.delRisky, removeRisky);
    emit AllocateRemoveDifference(allocate.delStable, removeStable);

    assert(allocate.delRisky == removeRisky);
    assert(allocate.delStable == removeStable);
    assert(intendedLiquidity == delLiquidity);
}
```

Step 2

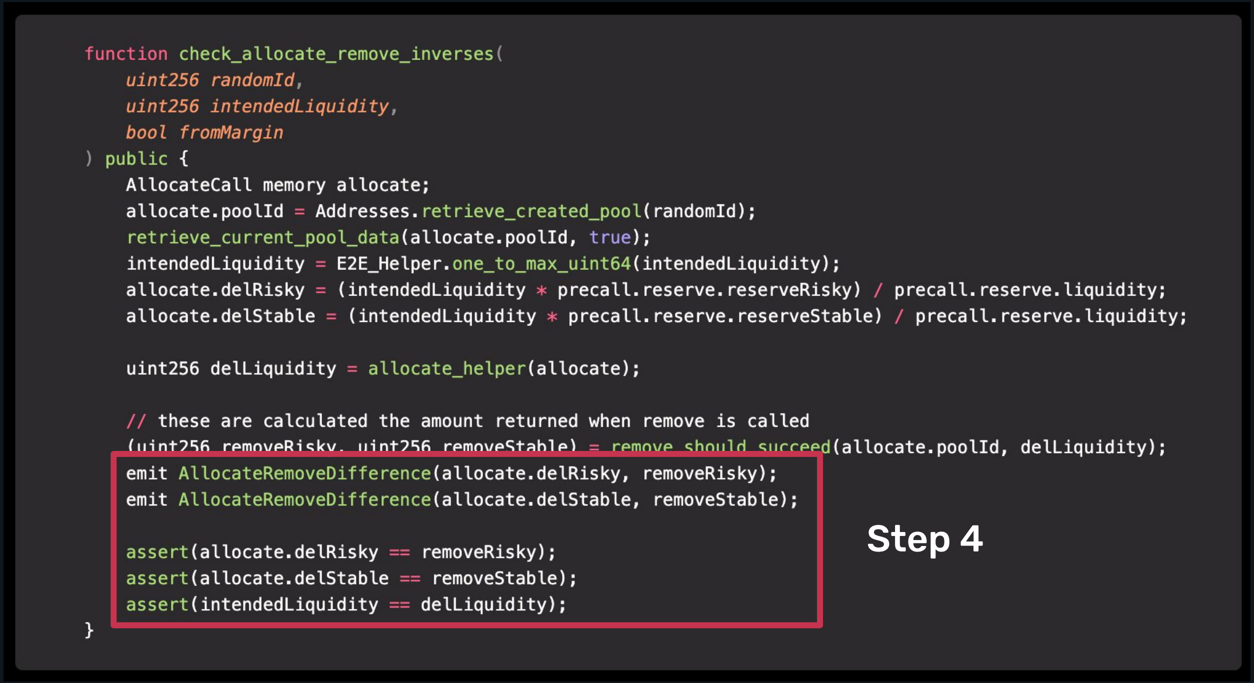
Invariant Test



```
function check_allocate_remove_inverses(
    uint256 randomId,
    uint256 intendedLiquidity,
    bool fromMargin
) public {
    AllocateCall memory allocate;
    allocate.poolId = Addresses.retrieve_created_pool(randomId);
    retrieve_current_pool_data(allocate.poolId, true);
    intendedLiquidity = E2E_Helper.one_to_max_uint64(intendedLiquidity);
    allocate.delRisky = (intendedLiquidity * precall.reserve.reserveRisky) / precall.reserve.liquidity;
    allocate.delStable = (intendedLiquidity * precall.reserve.reserveStable) / precall.reserve.liquidity;

    uint256 delLiquidity = allocate_helper(allocate);

    // these are calculated the amount returned when remove is called
    (uint256 removeRisky, uint256 removeStable) = remove_should_succeed(allocate.poolId, delLiquidity);
    emit AllocateRemoveDifference(allocate.delRisky, removeRisky);
    emit AllocateRemoveDifference(allocate.delStable, removeStable);

    assert(allocate.delRisky == removeRisky);
    assert(allocate.delStable == removeStable);
    assert(intendedLiquidity == delLiquidity);
}
```

Step 3

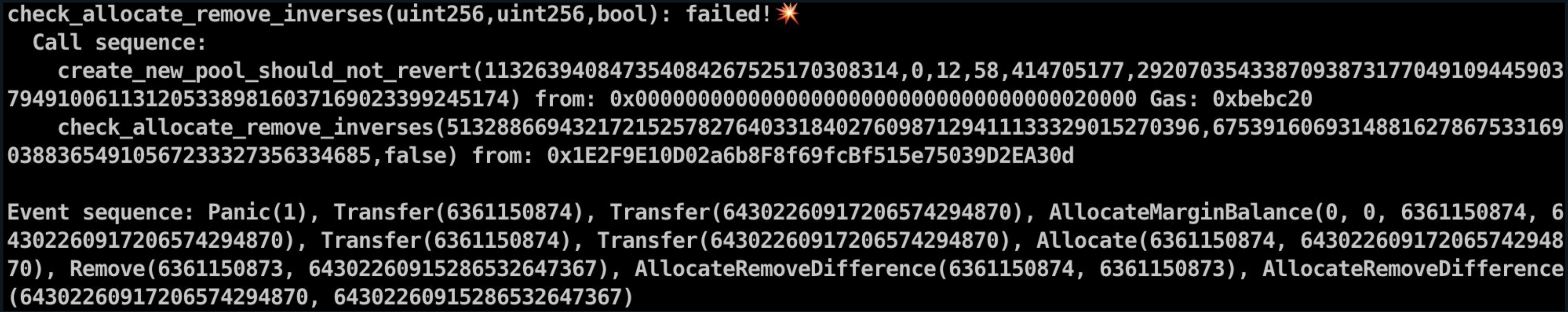
Invariant Test



```
function check_allocate_remove_inverses(
    uint256 randomId,
    uint256 intendedLiquidity,
    bool fromMargin
) public {
    AllocateCall memory allocate;
    allocate.poolId = Addresses.retrieve_created_pool(randomId);
    retrieve_current_pool_data(allocate.poolId, true);
    intendedLiquidity = E2E_Helper.one_to_max_uint64(intendedLiquidity);
    allocate.delRisky = (intendedLiquidity * precall.reserve.reserveRisky) / precall.reserve.liquidity;
    allocate.delStable = (intendedLiquidity * precall.reserve.reserveStable) / precall.reserve.liquidity;

    uint256 delLiquidity = allocate_helper(allocate);

    // these are calculated the amount returned when remove is called
    (uint256 removeRisky, uint256 removeStable) = remove_should_succeed(allocate.poolId, delLiquidity);
    emit AllocateRemoveDifference(allocate.delRisky, removeRisky);
    emit AllocateRemoveDifference(allocate.delStable, removeStable);

    assert(allocate.delRisky == removeRisky);
    assert(allocate.delStable == removeStable);
    assert(intendedLiquidity == delLiquidity);
}
```

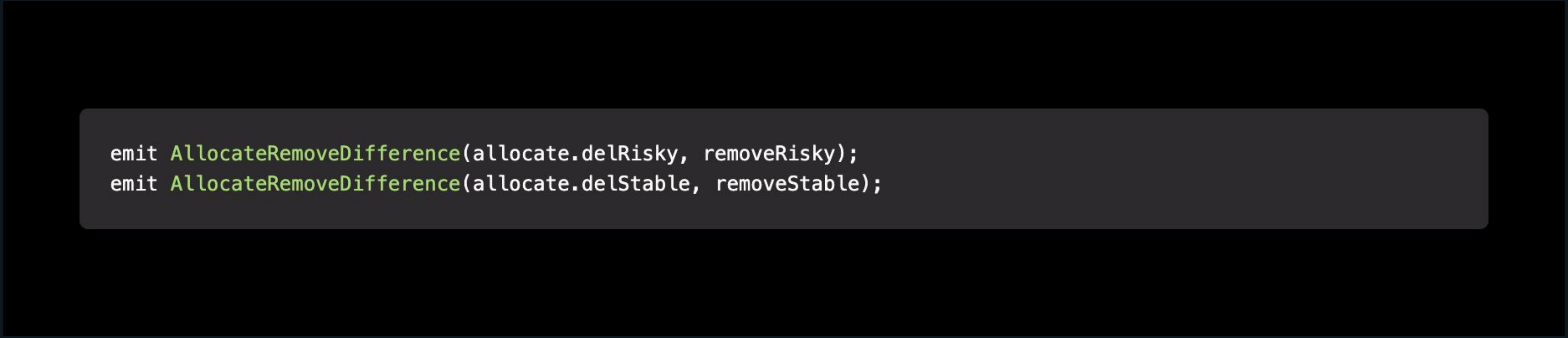
Step 4

Echidna Results



```
check_allocate_remove_inverses(uint256,uint256,bool): failed!💥
  Call sequence:
    create_new_pool_should_not_revert(113263940847354084267525170308314,0,12,58,414705177,2920703543387093873177049109445903
7949100611312053389816037169023399245174) from: 0x0000000000000000000000000000000000020000 Gas: 0xbebc20
    check_allocate_remove_inverses(513288669432172152578276403318402760987129411133329015270396,6753916069314881627867533169
03883654910567233327356334685,false) from: 0x1E2F9E10D02a6b8F8f69fcBf515e75039D2EA30d

Event sequence: Panic(1), Transfer(6361150874), Transfer(64302260917206574294870), AllocateMarginBalance(0, 0, 6361150874, 6
4302260917206574294870), Transfer(6361150874), Transfer(64302260917206574294870), Allocate(6361150874, 643022609172065742948
70), Remove(6361150873, 64302260915286532647367), AllocateRemoveDifference(6361150874, 6361150873), AllocateRemoveDifference
(64302260917206574294870, 64302260915286532647367)
```

Events



```
emit AllocateRemoveDifference(allocate.delRisky, removeRisky);
emit AllocateRemoveDifference(allocate.delStable, removeStable);
```

Event Results

	Amount allocated	Amount removed	Delta
Token 1	6361150874	6361150873	1

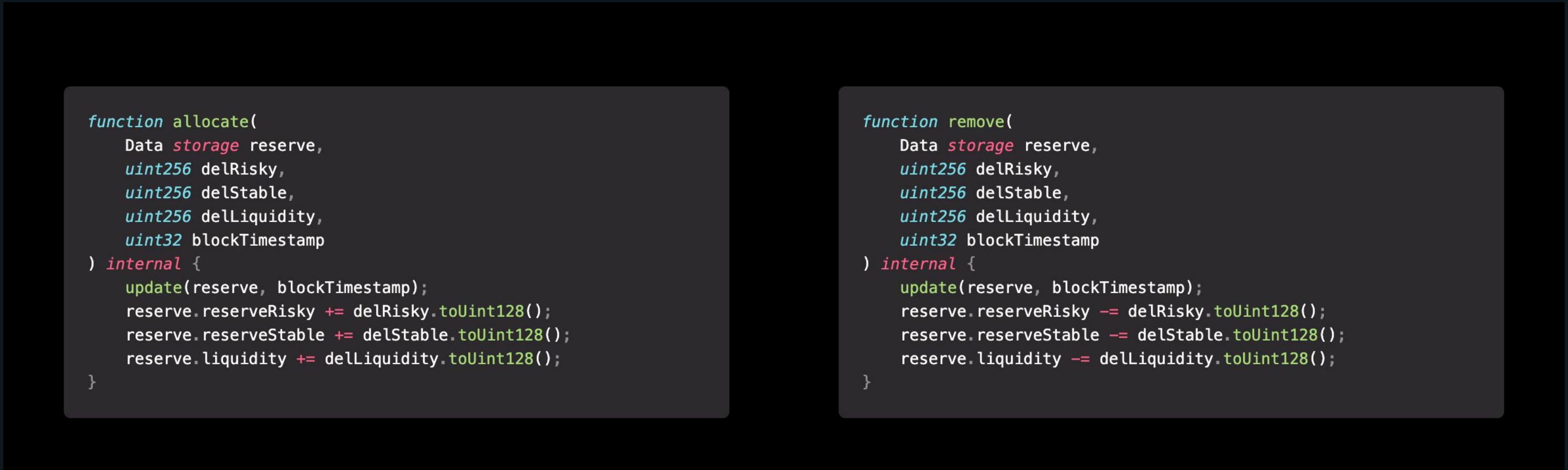
Event Results

	Amount allocated	Amount removed	Delta
Token 1	6361150874	6361150873	1
Token 2	64,302,260,917,206,574,294,870	64302260915286532647367	1,920,041,647,503

What does it mean?

- **Adding and removing funds are not exact inverses**
- **Users will actually receive 1,920,041,647,503 *less***

Why is there a delta?



```
function allocate(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky += delRisky.toUint128();
    reserve.reserveStable += delStable.toUint128();
    reserve.liquidity += delLiquidity.toUint128();
}
```

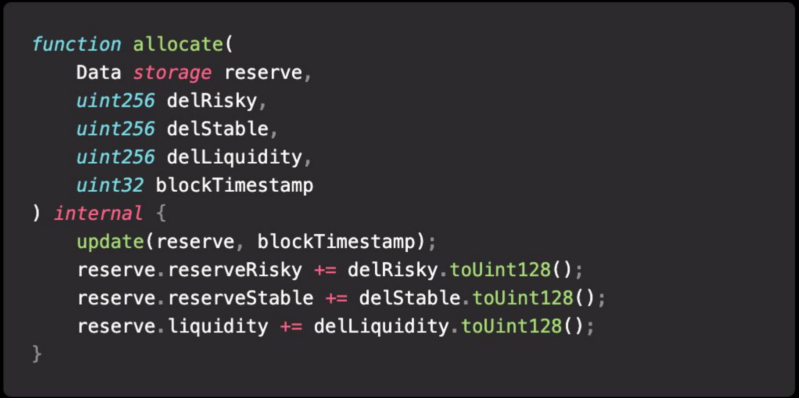
```
function remove(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky -= delRisky.toUint128();
    reserve.reserveStable -= delStable.toUint128();
    reserve.liquidity -= delLiquidity.toUint128();
}
```

`toUint128()`

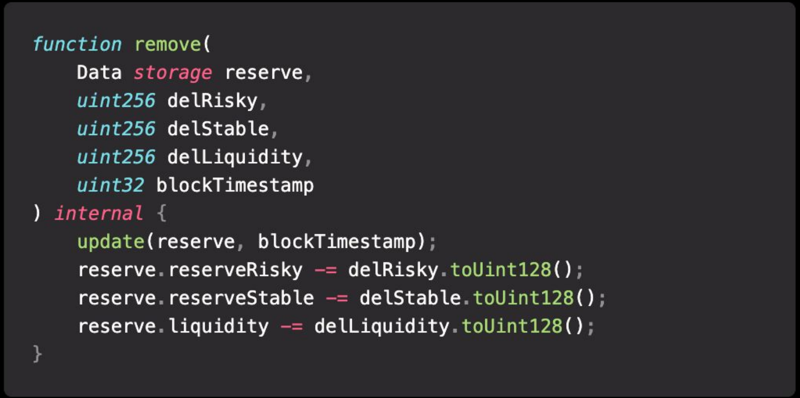
Why is `toUint128()` important?

- Converts `FixedPoint 64x64` to `uint128`
- Truncates numbers too large
- Used in *both* allocation and removal functions

With that in mind....



```
function allocate(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky += delRisky.toUint128();
    reserve.reserveStable += delStable.toUint128();
    reserve.liquidity += delLiquidity.toUint128();
}
```



```
function remove(
    Data storage reserve,
    uint256 delRisky,
    uint256 delStable,
    uint256 delLiquidity,
    uint32 blockTimestamp
) internal {
    update(reserve, blockTimestamp);
    reserve.reserveRisky -= delRisky.toUint128();
    reserve.reserveStable -= delStable.toUint128();
    reserve.liquidity -= delLiquidity.toUint128();
}
```

How can it be fixed?

It can’t, but….it can be *mitigated*

- **Defining an acceptable delta**
- **Round in a direction to benefit a pool**

Only the tip of the iceberg…

- Access Controls
- Correct Bookkeeping
- Token balances
- Differential Fuzzing

What next?

- Talk to us!
- Go through Echidna tutorials on building-secure-contracts
- Use Echidna on your codebase
- Join Empire Hacking