Towards Finding Accounting Errors in Smart Contracts

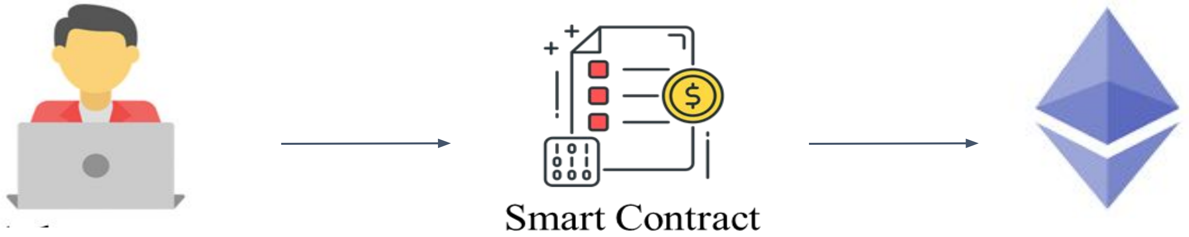
Brian Zhang

Reusable v1.1

Available v1.1

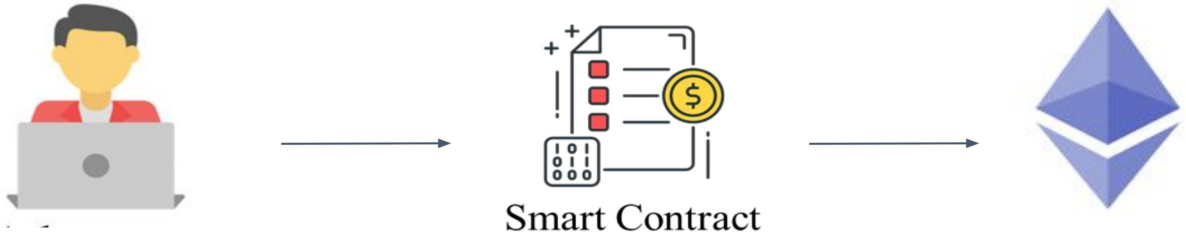
Smart Contracts

- **Blockchain-based application**
- Provide a wide variety of services:



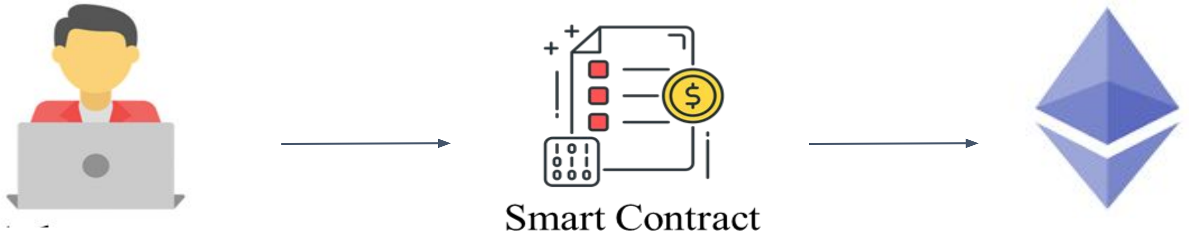
Smart Contracts

- **Blockchain-based application**
- Provide a wide variety of services:
  - Markets
  - Auctions
  - Gaming platforms



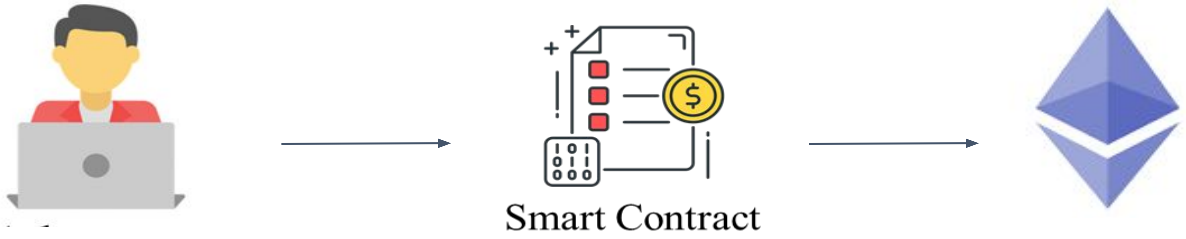
Smart Contracts

- **Blockchain-based application**
- Provide a wide variety of services:
  - Markets
  - Auctions
  - Gaming platforms
- Blockchains like **Ethereum** and **Polygon** support millions of transactions daily: (4.65B daily volume)
  - Tokens (WETH) are used instead of direct real money (USD)



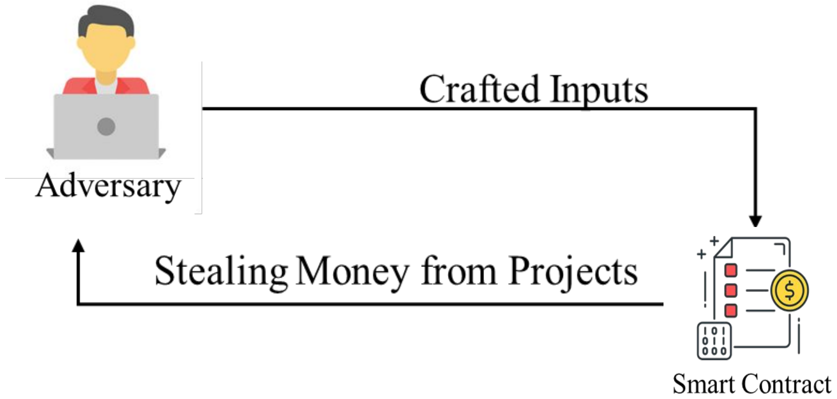
Smart Contracts

- **Blockchain-based application**
- Provide a wide variety of services:
  - Markets
  - Auctions
  - Gaming platforms
- Blockchains like **Ethereum** and **Polygon** support millions of transactions daily: (4.65B daily volume)
  - Tokens (WETH) are used instead of direct real money (USD)
- They rely on the Decentralized Finance (DeFi) principle



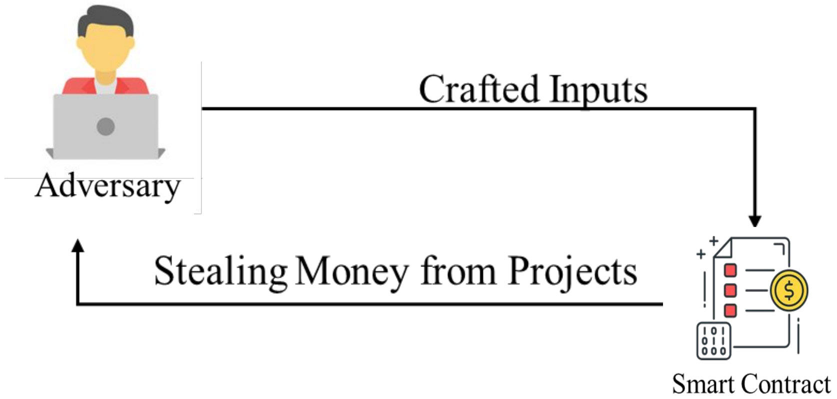
Smart Contract Exploits

- Smart contracts are developed by humans, and thus inevitably contain **vulnerabilities**



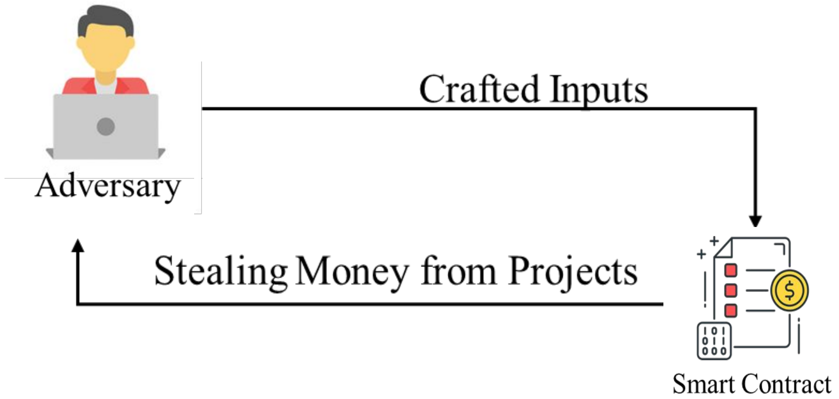
Smart Contract Exploits

- Smart contracts are developed by humans, and thus inevitably contain **vulnerabilities**
- Smart contracts are **lucrative targets** for malicious actors



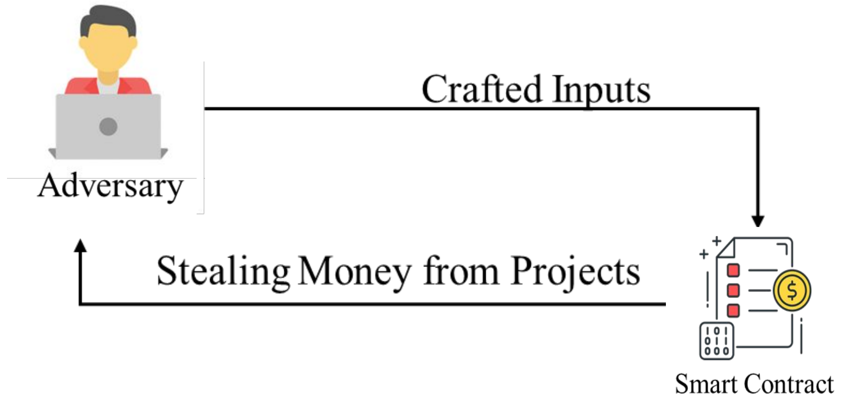
Smart Contract Exploits

- Smart contracts are developed by humans, and thus inevitably contain **vulnerabilities**
- Smart contracts are **lucrative targets** for malicious actors
- In Q2 of 2023, **212 exploits caused $300 million in damages**



Smart Contract Exploits

- Smart contracts are developed by humans, and thus inevitably contain **vulnerabilities**
- Smart contracts are **lucrative targets** for malicious actors
- In Q2 of 2023, **212 exploits caused $300 million in damages**
- Researchers have developed many techniques to prevent such exploits

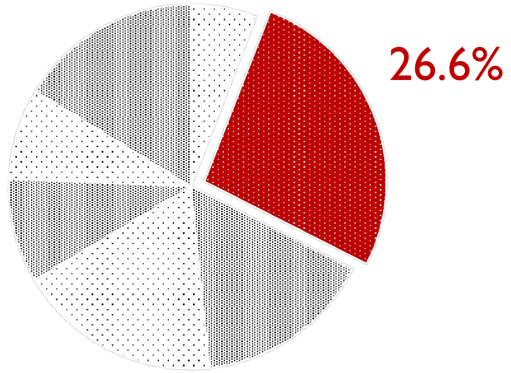


Accounting Errors in Smart Contracts

- A 2023 study of over 500 smart contract bugs found that **80% of exploitable bugs were beyond existing tools**

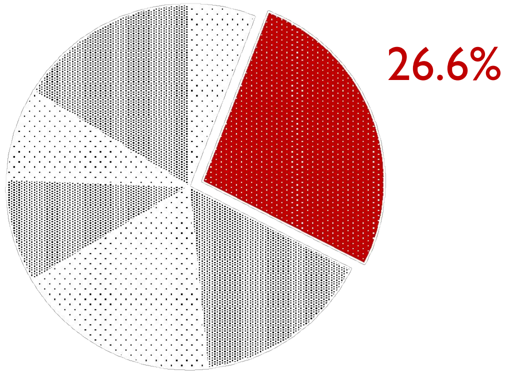
Accounting Errors in Smart Contracts

- A 2023 study of over 500 smart contract bugs found that **80% of exploitable bugs were beyond existing tools**
- Of the 80%, **exploits due to accounting errors made up 26.6%**
    - Accounting errors are the most popular category



Accounting Errors in Smart Contracts

- A 2023 study of over 500 smart contract bugs found that **80% of exploitable bugs were beyond existing tools**
- Of the 80%, **exploits due to accounting errors made up 26.6%**
  - Accounting errors are the most popular category
- *Accounting errors are incorrect implementations of domain-specific business models*



Accounting Errors in Smart Contracts

- A 2023 study of over 500 smart contract bugs found that **80% of exploitable bugs were beyond existing tools**
- Of the 80%, **exploits due to accounting errors made up 26.6%**
  - Accounting errors are the most popular category
- *Accounting errors are incorrect implementations of domain-specific business models*
- Uranium finance exploit caused **$87 million dollars** of damages due to two extra zeros
  - The bug survived multiple rounds of pre-deployment auditing

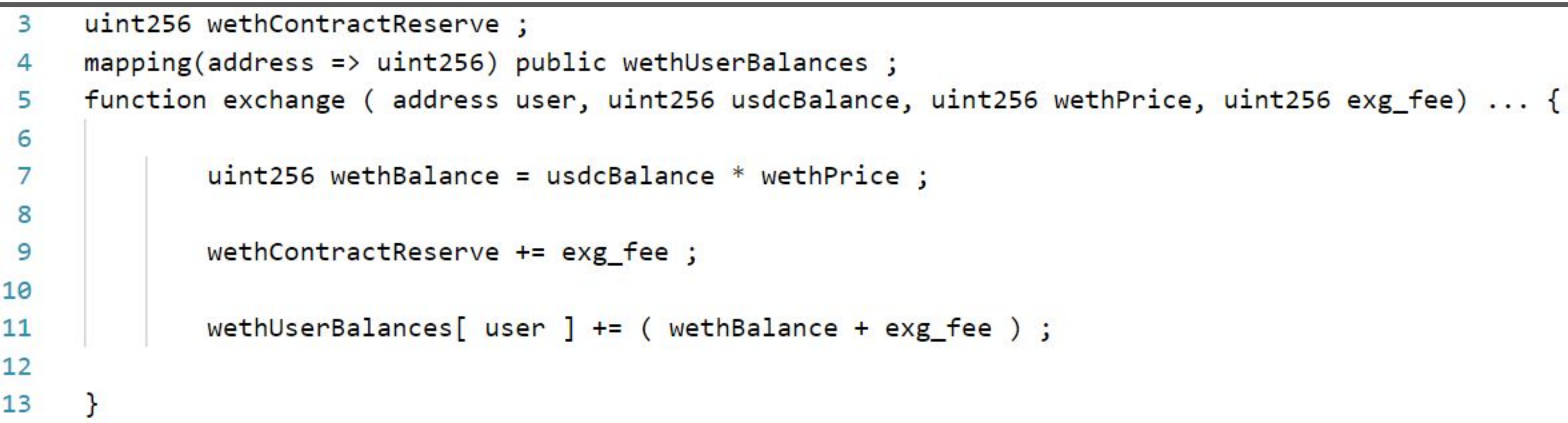
Motivating Example (from the *Tracer* Project)

- Performs an exchange from USD to WETH

```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example (from the *Tracer* Project)

- Performs an exchange from USD to WETH
- Collects an exchange fee



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

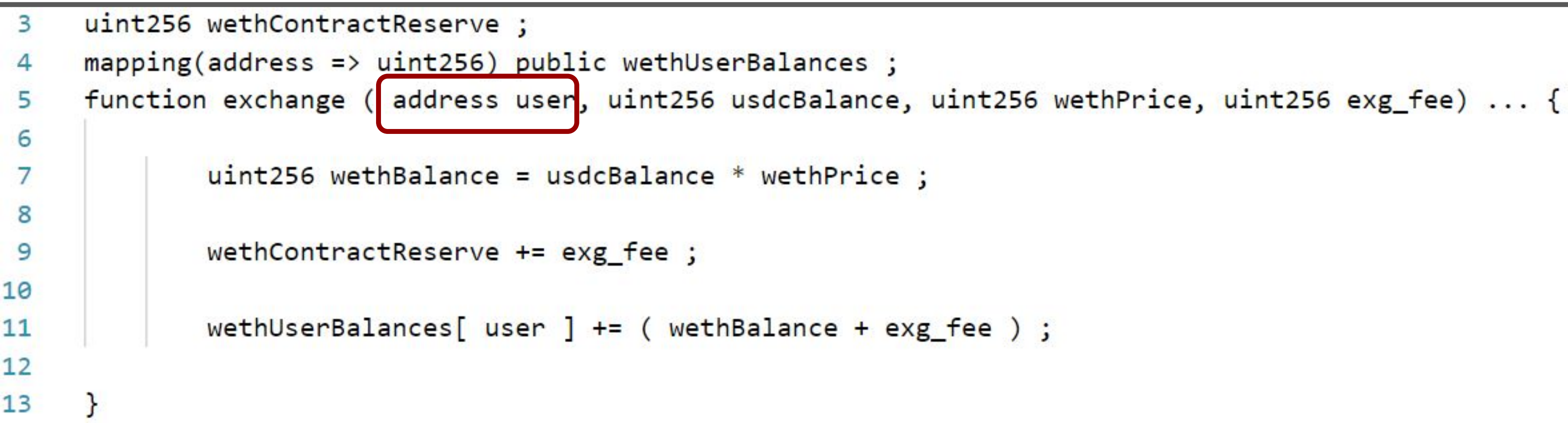
Motivating Example (from the *Tracer* Project)

- Performs an exchange from USD to WETH
- Collects an exchange fee
- Analogous to converting money at an ATM

```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example - Variables

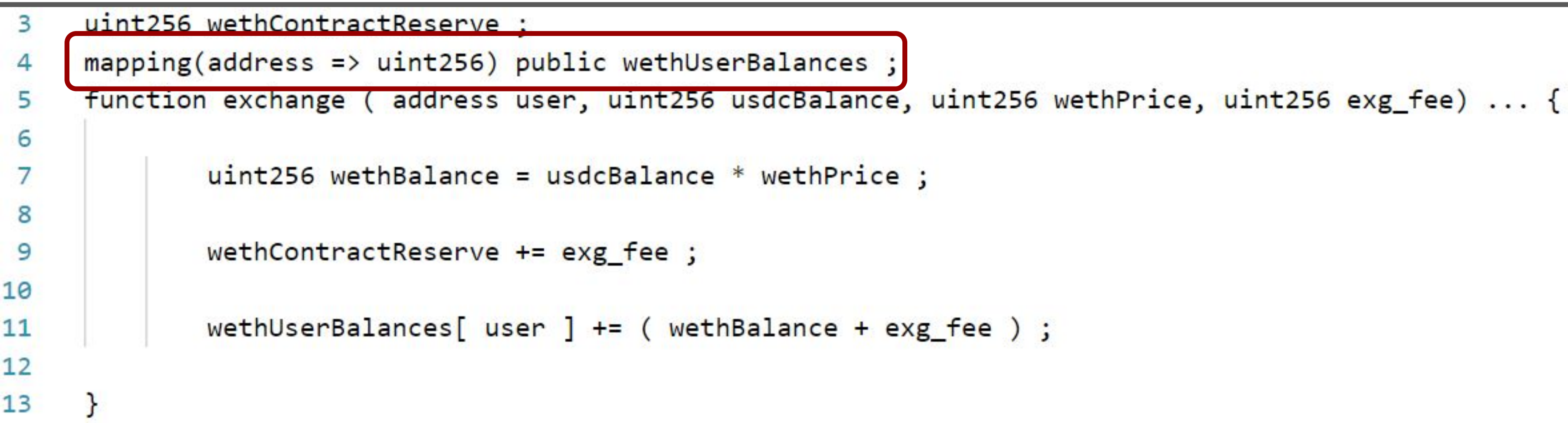
- "*user*" is the address for a user
  - Analogous to credentials stored on a credit card



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example - Variables

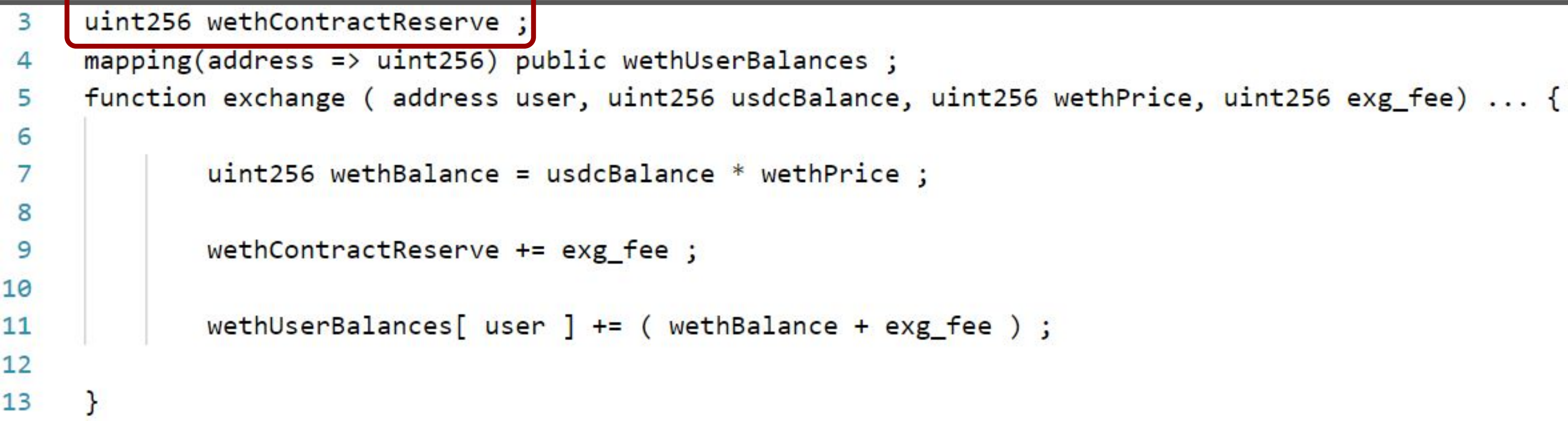
- "*user*" is the address for a user
- "*wethUserBalances*" is an array storing the accounts of all users (in WETH)
    - Analogous to bank accounts



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example - Variables

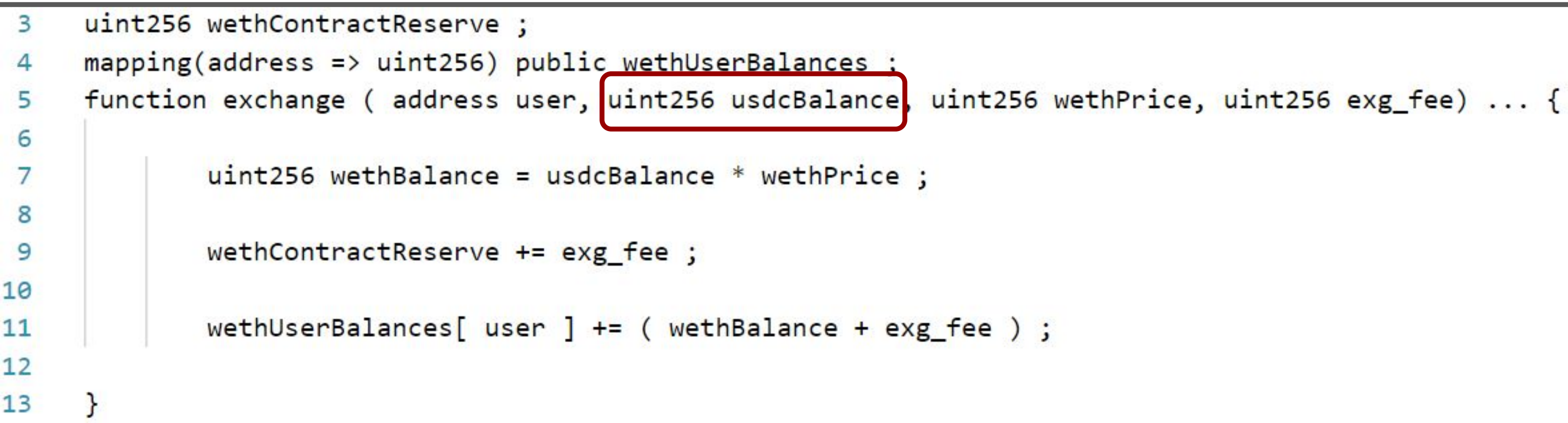
- "*user*" is the address for a user
- "*wethUserBalances*" is an array storing the accounts of all users (in WETH)
    - Analogous to bank accounts
- "*wethContractReserve*" is the reserve/account of the smart contract (in WETH)
    - Analogous to ATM reserves



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

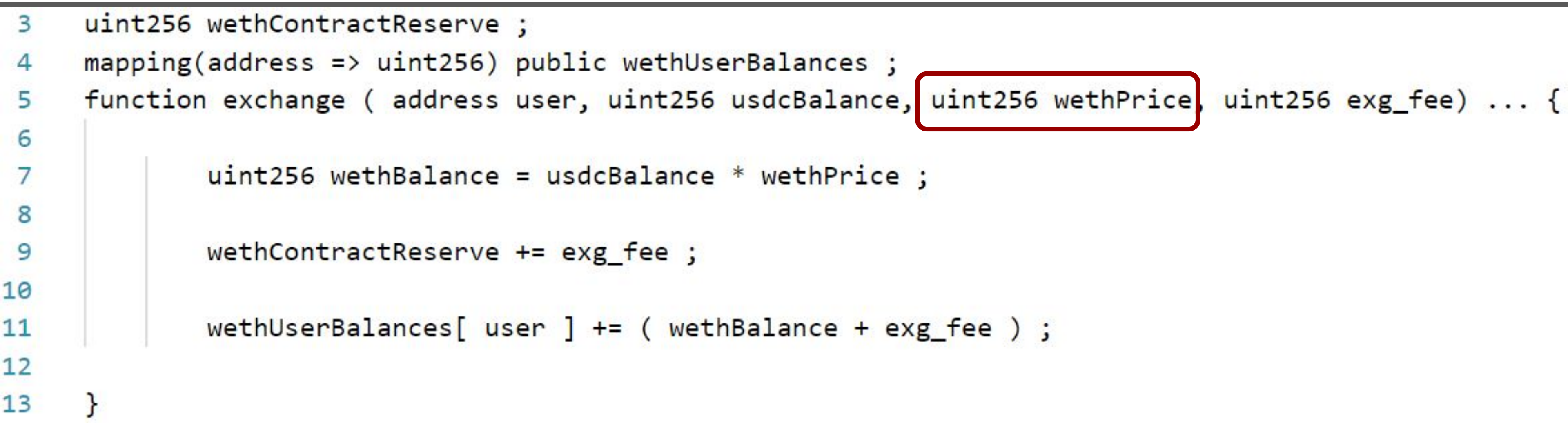
- "*usdcBalance*" is the amount of USD to be exchanged



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

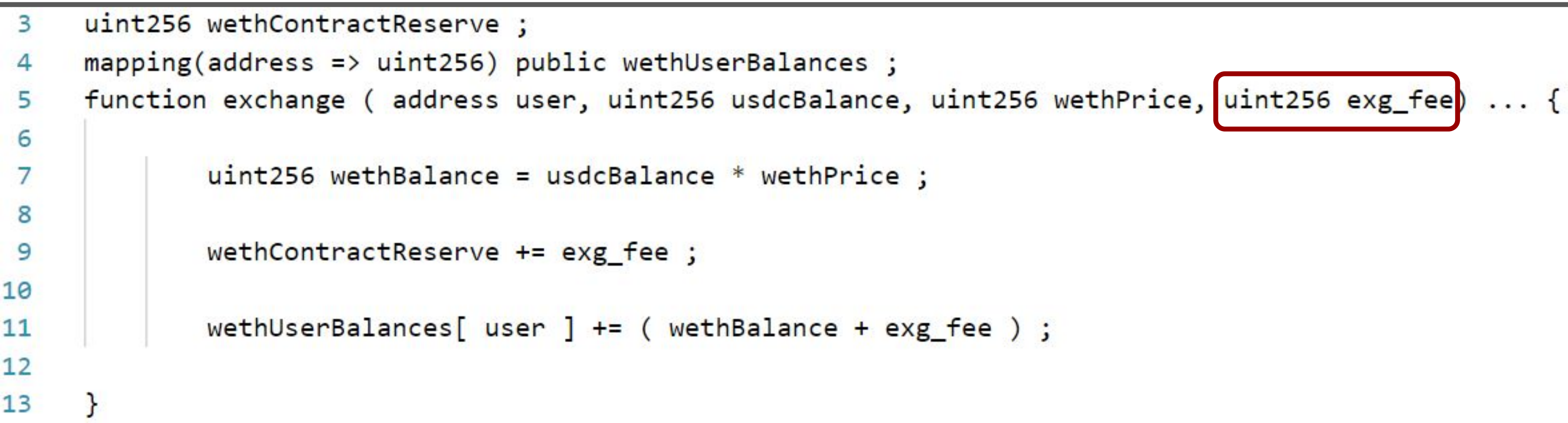
- “*usdcBalance*” is the amount of USD to be exchanged
- “*wethPrice*” is the conversion price of USD to WETH
    - Around $3,600



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

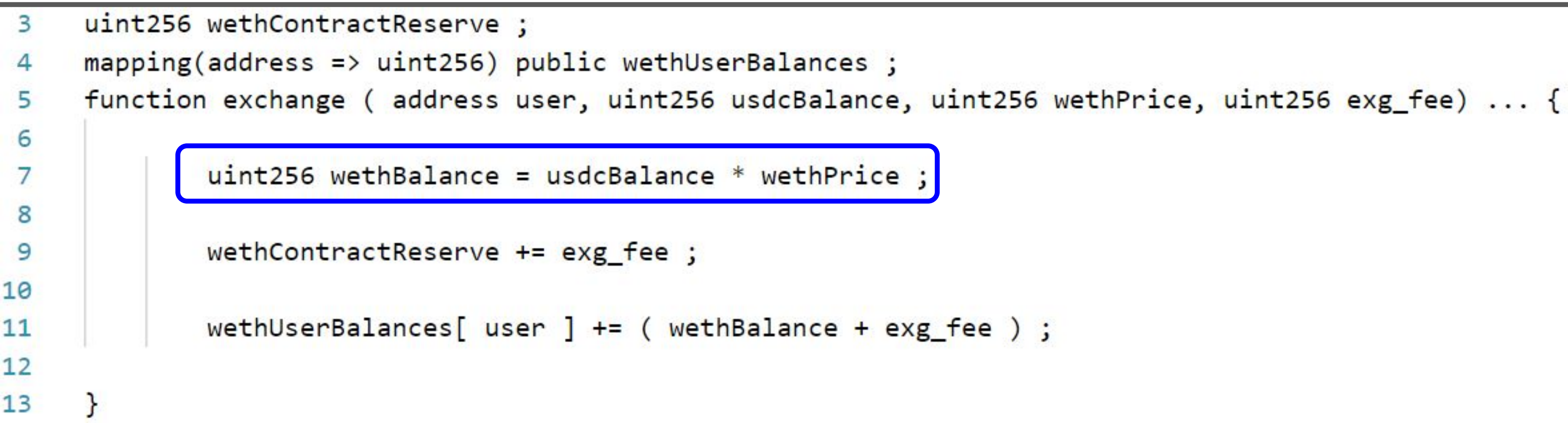
- "*usdcBalance*" is the amount of USD to be exchanged
- "*wethPrice*" is the conversion price of USD to WETH
    - Around $3,600 = 1 WETH
- "*exg_fee*" is the fee to be collected during the exchange



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve += exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

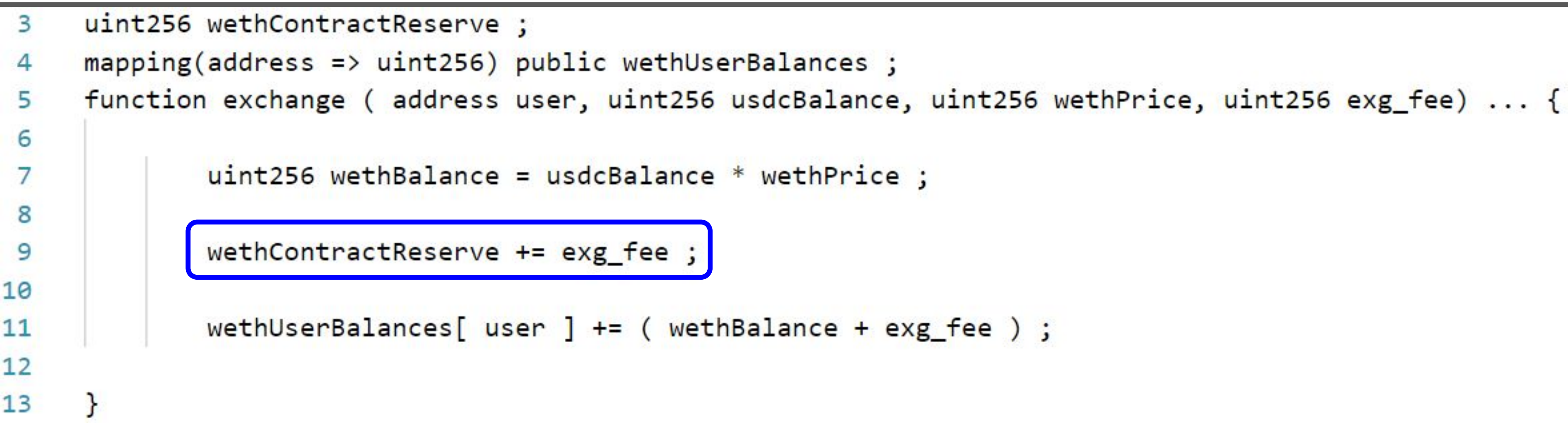
- Performs the exchange from USD to WETH on line **7**
  - Multiplies the amount of USD by the conversion price to WETH



```
 3  uint256 wethContractReserve ;
 4  mapping(address => uint256) public wethUserBalances ;
 5  function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
 6
 7          uint256 wethBalance = usdcBalance * wethPrice ;
 8
 9          wethContractReserve += exg_fee ;
10
11          wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13  }
```

Motivating Example

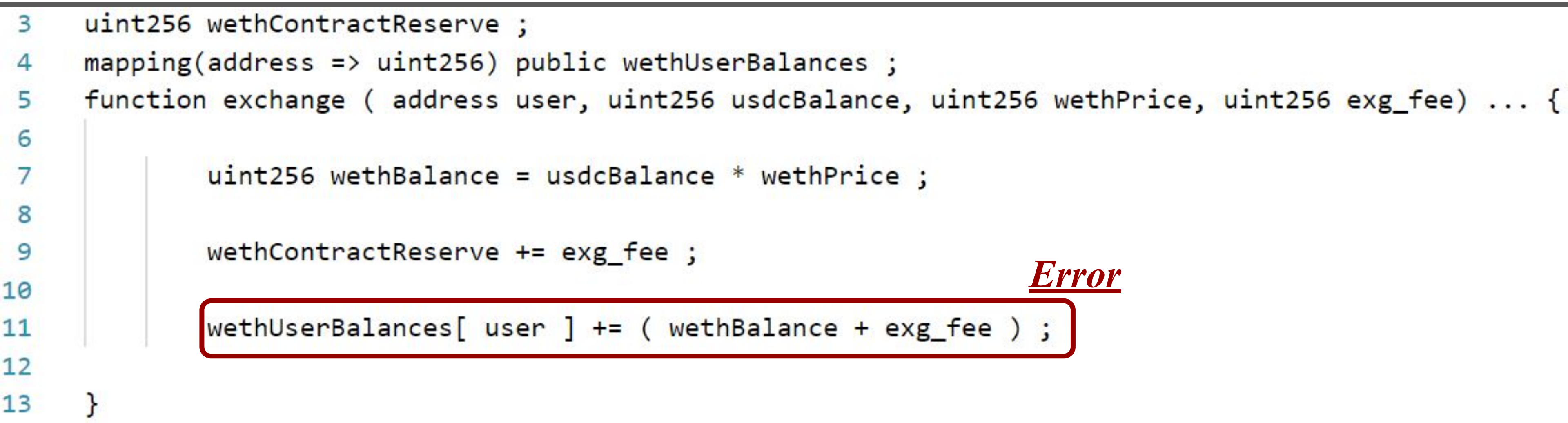
- Performs the exchange from USD to WETH on line **7**
  - Multiplies the amount of USD by the conversion price to WETH
- Adds the exchange fee to the contract reserves on line **9**



```
3   uint256 wethContractReserve ;
4   mapping(address => uint256) public wethUserBalances ;
5   function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
6
7           uint256 wethBalance = usdcBalance * wethPrice ;
8
9           wethContractReserve += exg_fee ;
10
11          wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13  }
```

Motivating Example

- Accounting error on line **11**
  - Should instead append by: “ *wethBalance - exg_fee*”

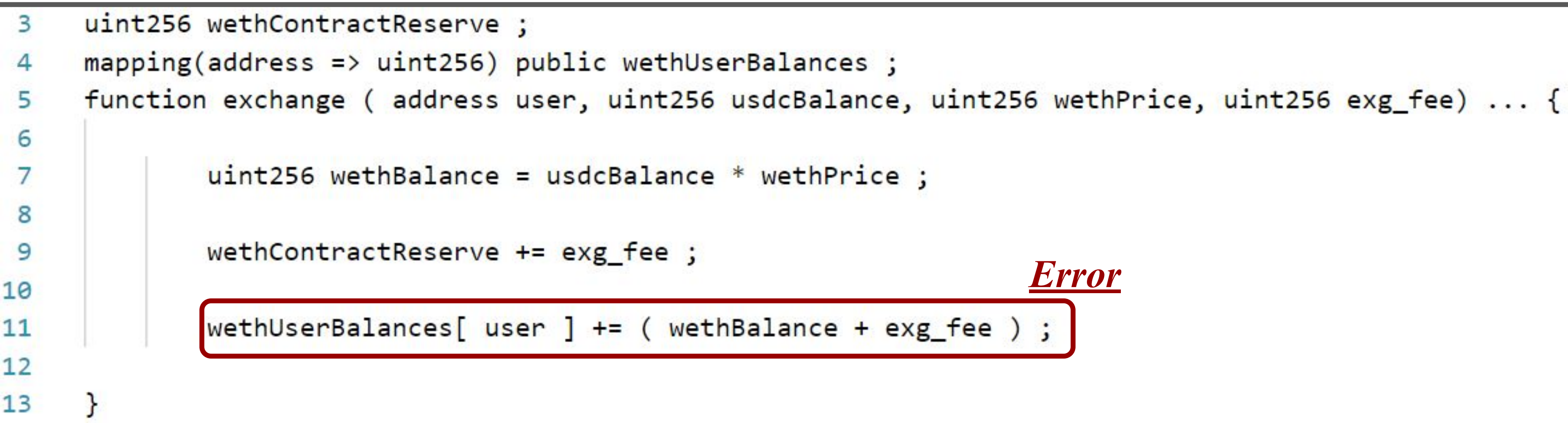


```
3    uint256 wethContractReserve ;
4    mapping(address => uint256) public wethUserBalances ;
5    function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
6
7            uint256 wethBalance = usdcBalance * wethPrice ;
8
9            wethContractReserve += exg_fee ;
10
11           wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13   }
```

Error

Motivating Example

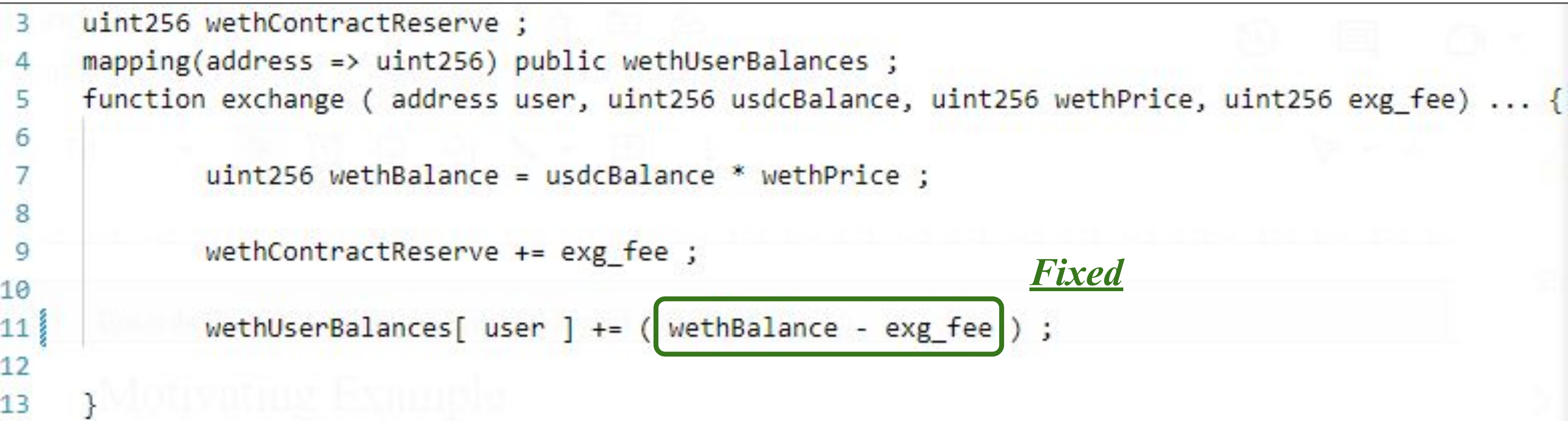
- Accounting error on line **11**
  - Should instead append by: " *wethBalance - exg_fee*"
  - Intuitively, it's adding the "*exg_fee*" to the user's account instead of decrementing



```
 3  uint256 wethContractReserve ;
 4  mapping(address => uint256) public wethUserBalances ;
 5  function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
 6
 7          uint256 wethBalance = usdcBalance * wethPrice ;
 8
 9          wethContractReserve += exg_fee ;
10
11          wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13  }
```

Motivating Example

- Accounting error on line **11**
  - Should instead append by: " *wethBalance - exg_fee*"
  - Intuitively, it's adding the "*exg_fee*" to the user's account instead of decrementing

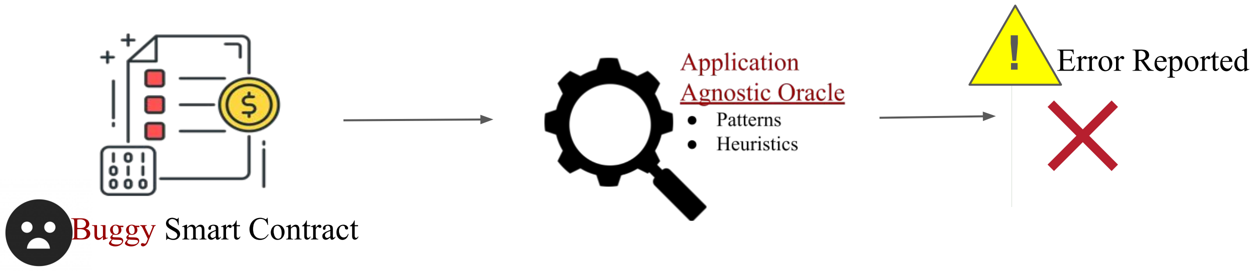


```
3   uint256 wethContractReserve ;
4   mapping(address => uint256) public wethUserBalances ;
5   function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
6
7           uint256 wethBalance = usdcBalance * wethPrice ;
8
9           wethContractReserve += exg_fee ;
10
11          wethUserBalances[ user ] += ( wethBalance - exg_fee ) ;
12
13  }
```

Fixed

Challenges to detecting Accounting Errors

- **Challenge 1:** No existing general-testing oracles
  - Oracles have made bugs such as **Reentrancy** and **Integer Overflow** obsolete



Challenges to detecting Accounting Errors

- **Challenge 1:** No existing general-testing oracles
  - Oracles have made bugs such as **Reentrancy** and **Integer Overflow** obsolete
- **Challenge 2:** Requires understanding the complex business logic of Smart Contracts

Key insights to ScType

- **Insight 1:** Many accounting errors manifest as *abstract type violations*

Key insights to ScType

- **Insight 1:** Many accounting errors manifest as *abstract type violations*
- **Insight 2:** All smart contracts can be instantiated as banks
    - Many basic operations are analogous
    - I.e. Depositing, Withdrawing, Loaning …

ScType

- ScType is a type-checking system
    - It is implemented on the Slither static analysis tool

ScType

- ScType is a type-checking system
  - It is implemented on the Slither static analysis tool
- It introduces an abstract type for Solidity variables, ExtendedType:

ScType

- ScType is a type-checking system
    - It is implemented on the Slither static analysis tool
- It introduces an abstract type for Solidity variables, ExtendedType:
    - **Financial Meaning**

ScType

- ScType is a type-checking system
  - It is implemented on the Slither static analysis tool
- It introduces an abstract type for Solidity variables, ExtendedType:
  - **Financial Meaning**
  - **Token Unit**

ScType

- ScType is a type-checking system
  - It is implemented on the Slither static analysis tool
- It introduces an abstract type for Solidity variables, ExtendedType:
  - **Financial Meaning**
  - **Token Unit**
  - **Scaling Factor**

ScType

- ScType is a type-checking system
  - It is implemented on the Slither static analysis tool
- It introduces an abstract type for Solidity variables, ExtendedType:
  - **Financial Meaning**
  - **Token Unit**
  - **Scaling Factor**
- Allows for type rules to be created that check **consistency** and **correctness** of Smart Contract operations

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank
- Variables and Operations in smart contracts can be assigned a “meaning” based on usage

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank
- Variables and Operations in smart contracts can be assigned a “meaning” based on usage
- Examples of Financial Meaning for variables:

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank
- Variables and Operations in smart contracts can be assigned a “meaning” based on usage
- Examples of Financial Meaning for variables:
  - “Raw Balance” - An amount of a currency owned by a user (that has not had fee applied to it)

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank
- Variables and Operations in smart contracts can be assigned a “meaning” based on usage
- Examples of Financial Meaning for variables:
  - “Raw Balance” - An amount of a currency owned by a user (that has not had fee applied to it)
  - “Price” - The ratio representing the transfer of one currency to another

ExtendedType - Financial Meaning

- We observe that most smart contract projects can be mapped to functions of a bank
- Variables and Operations in smart contracts can be assigned a “meaning” based on usage
- Examples of Financial Meaning:
  - “Raw Balance” - An amount of a currency owned by a user (that has not had fee applied to it)
  - “Price” - The ratio representing the transfer of one currency to another
  - “Reserve” - An amount of currency owned by the Smart Contract
  - …

ExtendedType - Financial Meaning

- Operations are constrained by the Financial Meanings of their operands
    - Certain operations do not make logical sense, nor result in meaningful output

ExtendedType - Financial Meaning

- Operations are constrained by the Financial Meanings of their operands
  - Certain operations do not make logical sense, nor result in meaningful output
- Provided is a table for "+" operations (Left Column + Top Row = Cell)

+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Reserve
Fee	🚫	🚫	Reserve	Fee

ExtendedType - Financial Meaning

- Operations are constrained by the Financial Meanings of their operands
  - Certain operations do not make logical sense, nor result in meaningful output
- Provided is a table for "+" operations (Left Column + Top Row = Cell)
  - **"Raw Balance" + "Reserve" = "Reserve"**

+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Reserve
Fee	🚫	🚫	Reserve	Fee

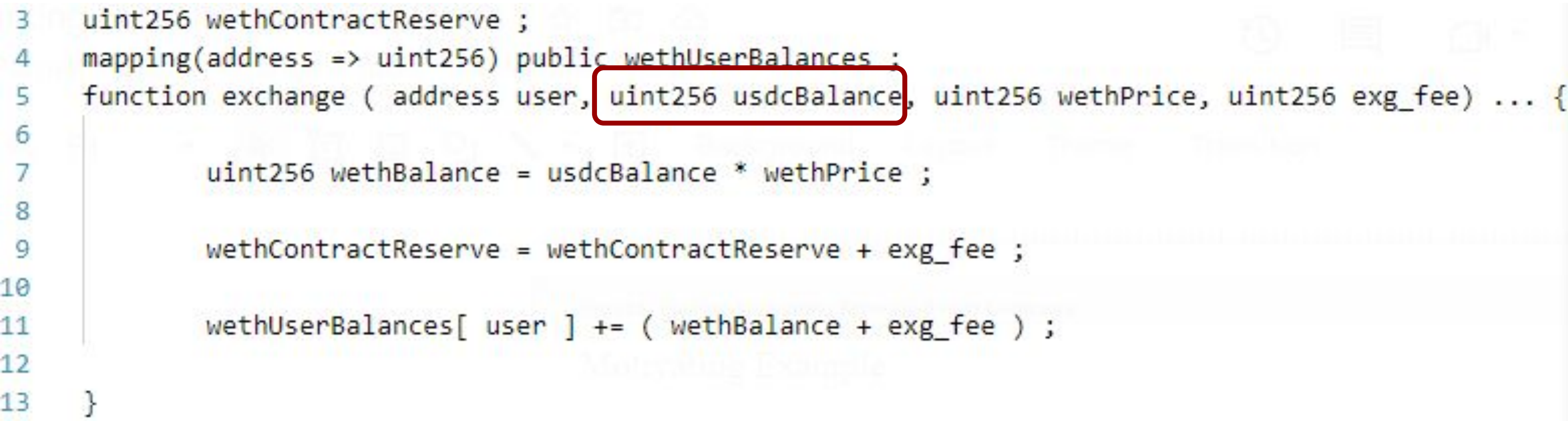
ExtendedType - Financial Meaning

- Operations are constrained by the Financial Meanings of their operands
  - Certain operations do not make logical sense, nor result in meaningful output
- Provided is a table for "+" operations (Left Column + Top Row = Cell)
  - **"Raw Balance" + "Reserve" = "Reserve"**
  - **"Raw Balance" + "Price" = Error**
  - The complete table can be found in our paper

+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Reserve
Fee	🚫	🚫	Reserve	Fee

Motivating Example

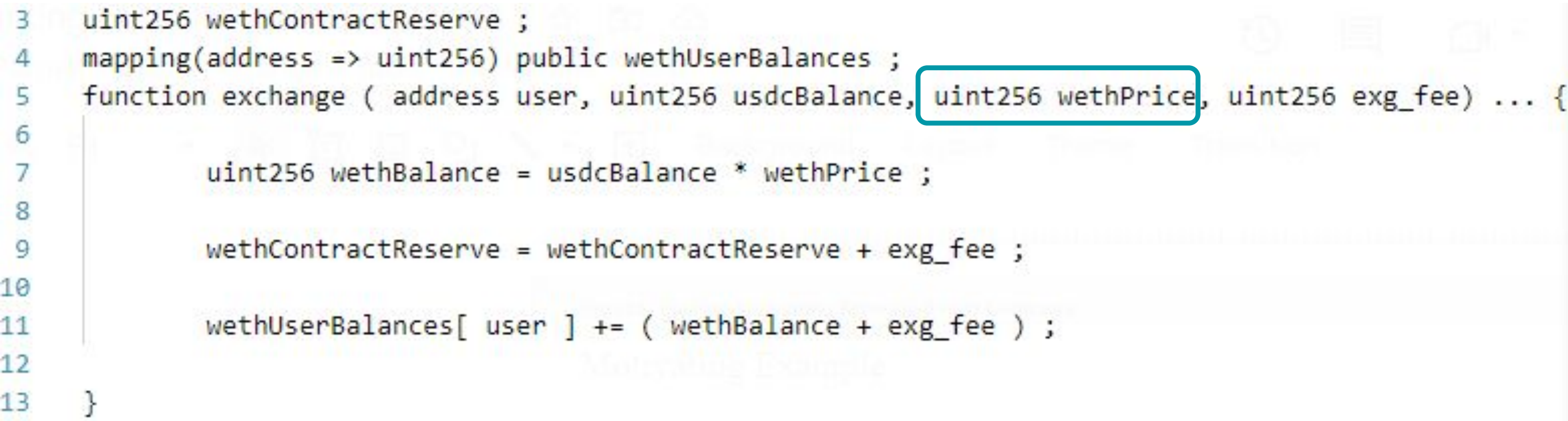
- "*usdcBalance*" has financial meaning "Raw Balance"
  - "Raw Balance": An amount of tokens owned by users



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

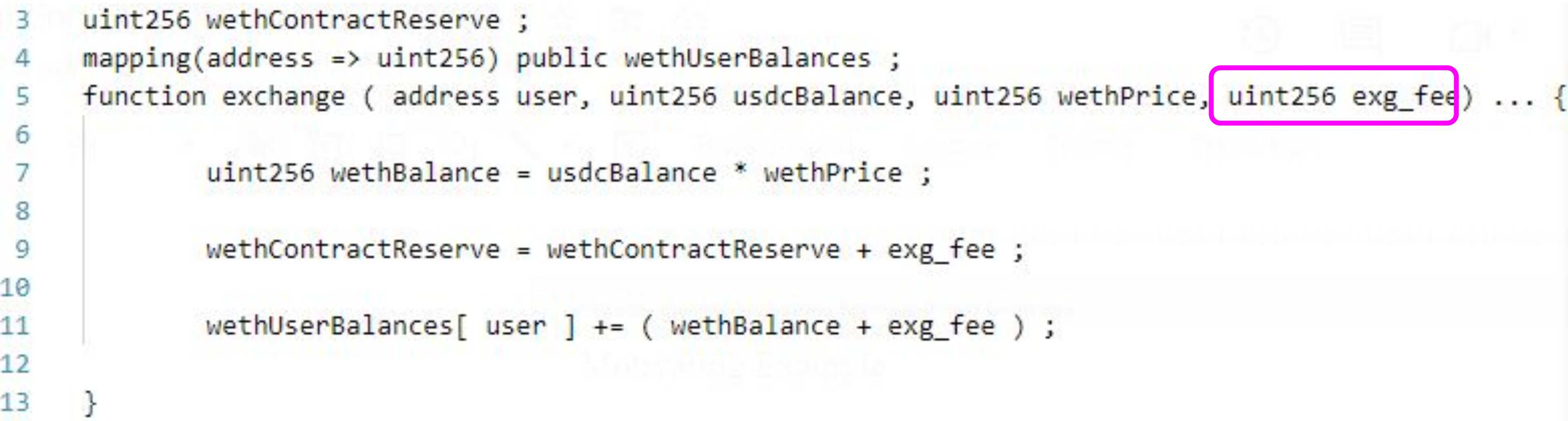
- "*usdcBalance*" has financial meaning "Raw Balance"
  - "Raw Balance": An amount of tokens owned by users
- "*wethPrice*" has financial meaning "Price"
  - "Price": An exchange rate from one token to another



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

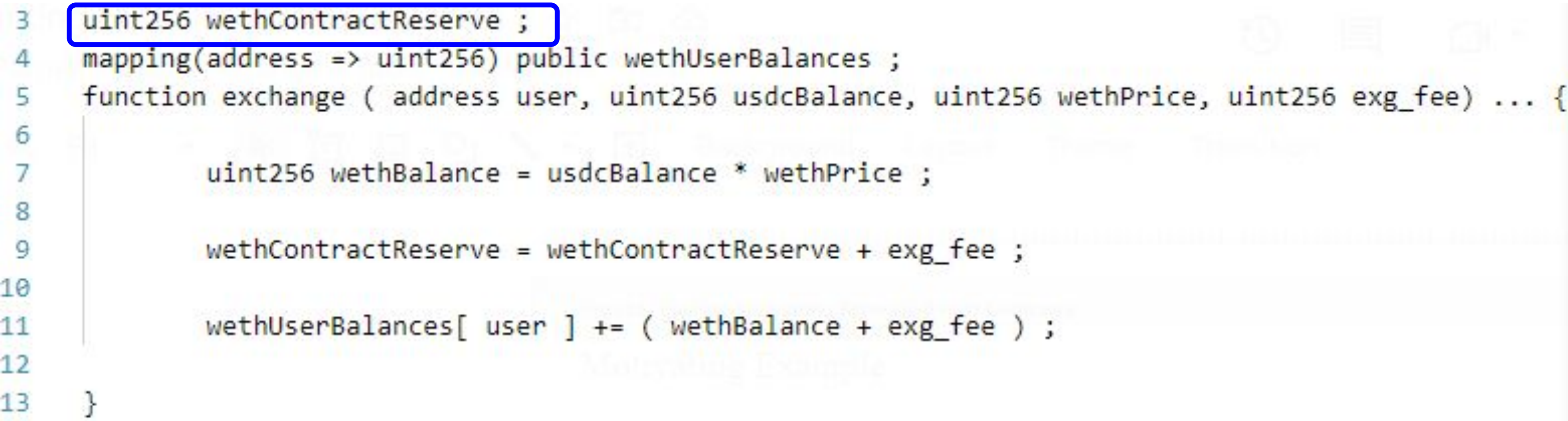
- "*exg_fee*" has financial meaning "Fee"
    - "Fee": An amount of tokens that are taken as fee for an operation



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

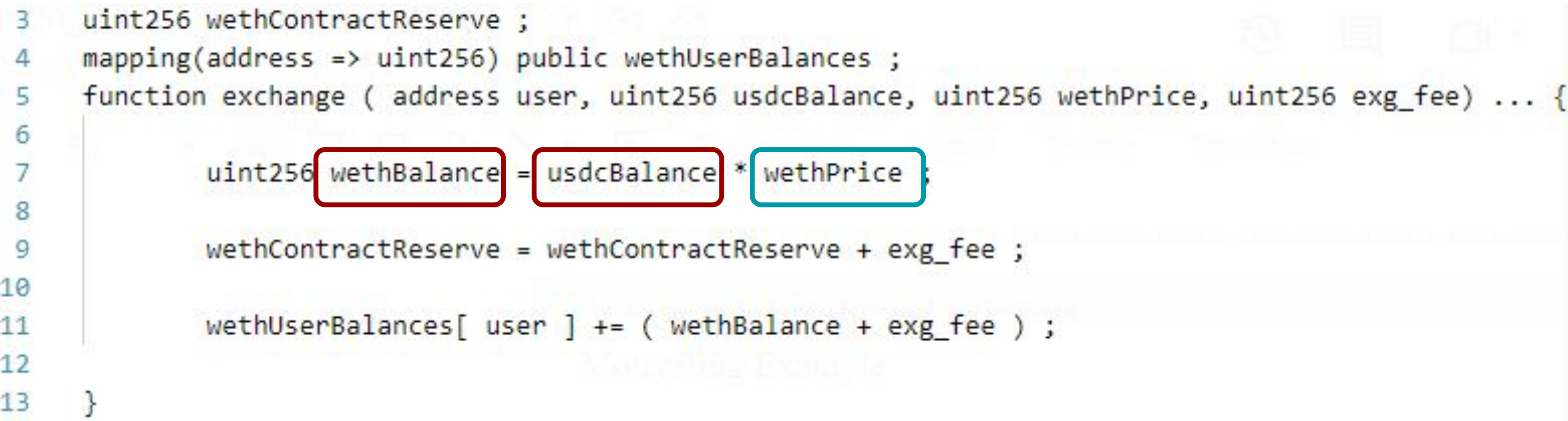
- "*exg_fee*" has financial meaning "Fee"
  - "Fee": An amount of tokens that are taken as fee for an operation
- "wethContractReserve" has financial meaning "Reserve"
  - "Reserve": An amount of tokens that are owned by the smart contract, not user



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

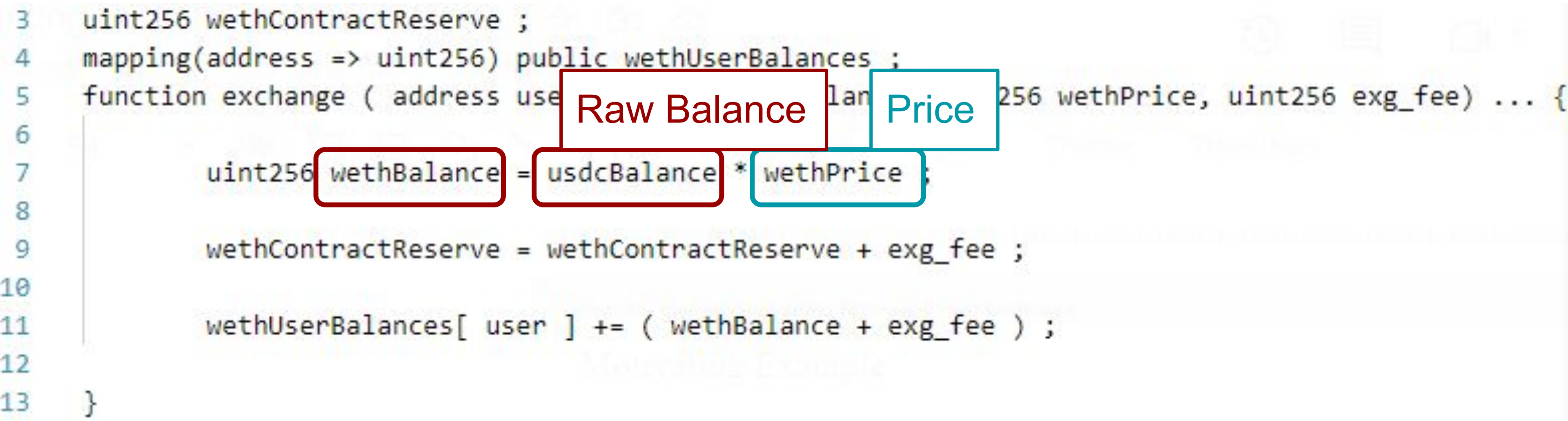
- “*wethBalance*” has financial meaning “Raw Balance”



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

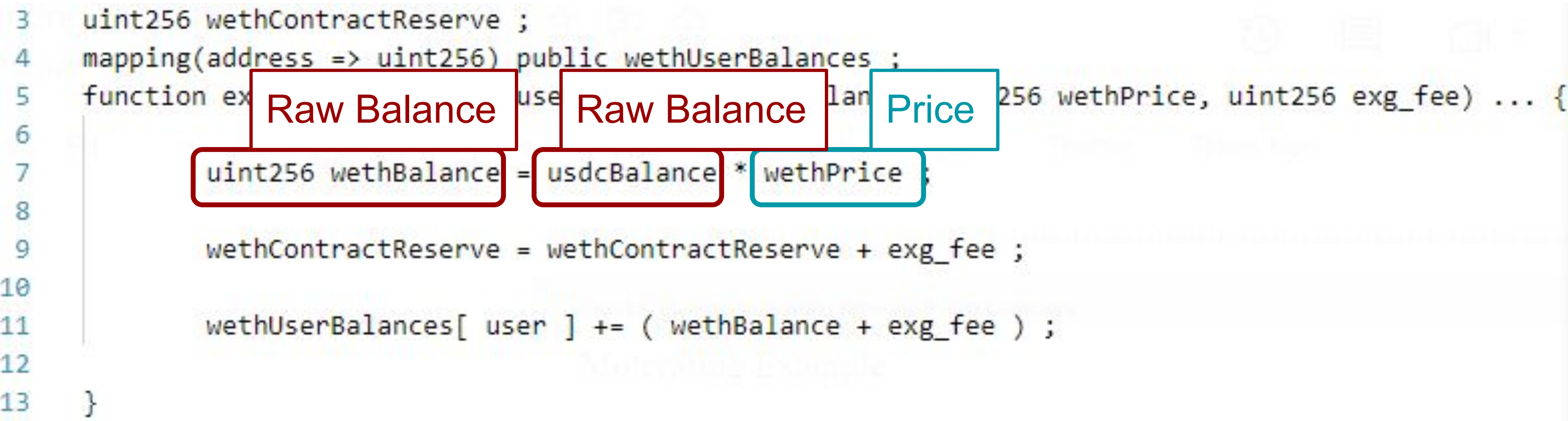
- “*wethBalance*” has financial meaning “Raw Balance”
    - “*usdcBalance*” (Raw Balance) * “*wethPrice*” (Price) = “*wethBalance*” (Raw Balance)



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address use                  lan          256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

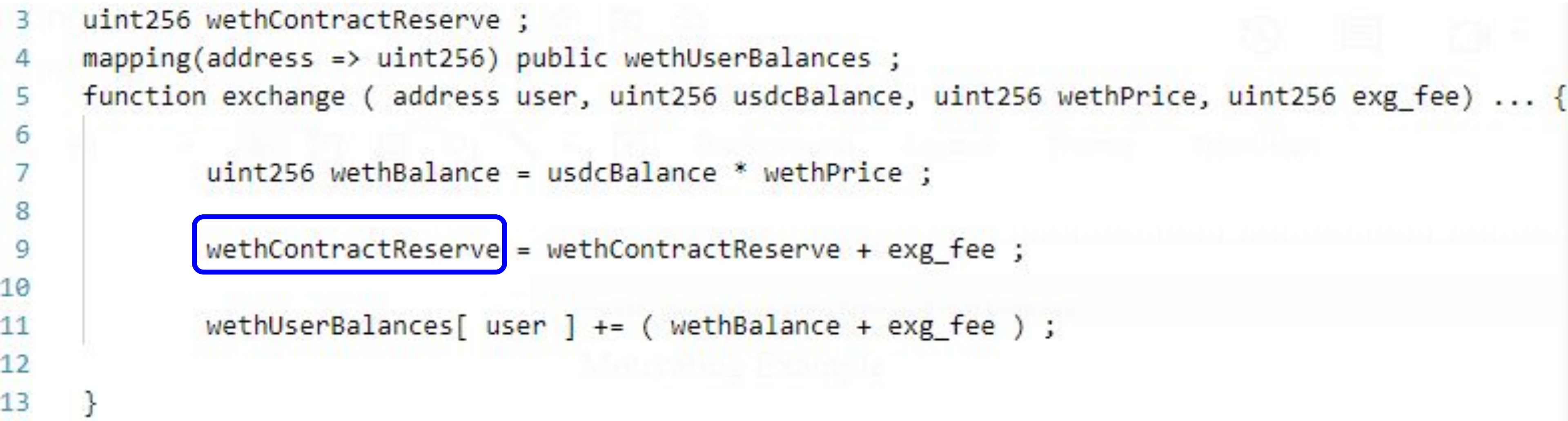
- "*wethBalance*" has financial meaning "Raw Balance"
  - "*usdcBalance*" (Raw Balance) * "*wethPrice*" (Price) = "*wethBalance*" (Raw Balance)
  - Intuitively, multiply by price only changes the token unit, not the meaning



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function ex                  use                  lan           256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

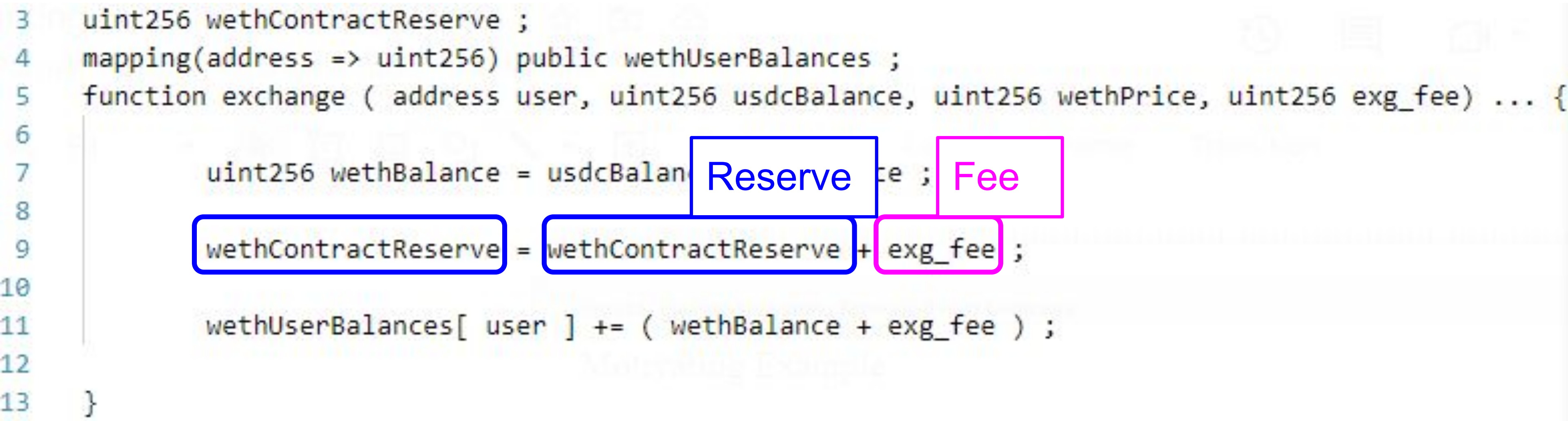
- "*wethContractReserve*" has financial meaning "Reserve"
  - "*wethContractReserve*" (Reserve) + "*exg_fee*" (Fee) = "*wethContractReserve*" (Reserve)



```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalance * wethPrice ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

- "*wethContractReserve*" has financial meaning "Reserve"
  - "*wethContractReserve*" (Reserve) + "*exg_fee*" (Fee) = "*wethContractReserve*" (Reserve)

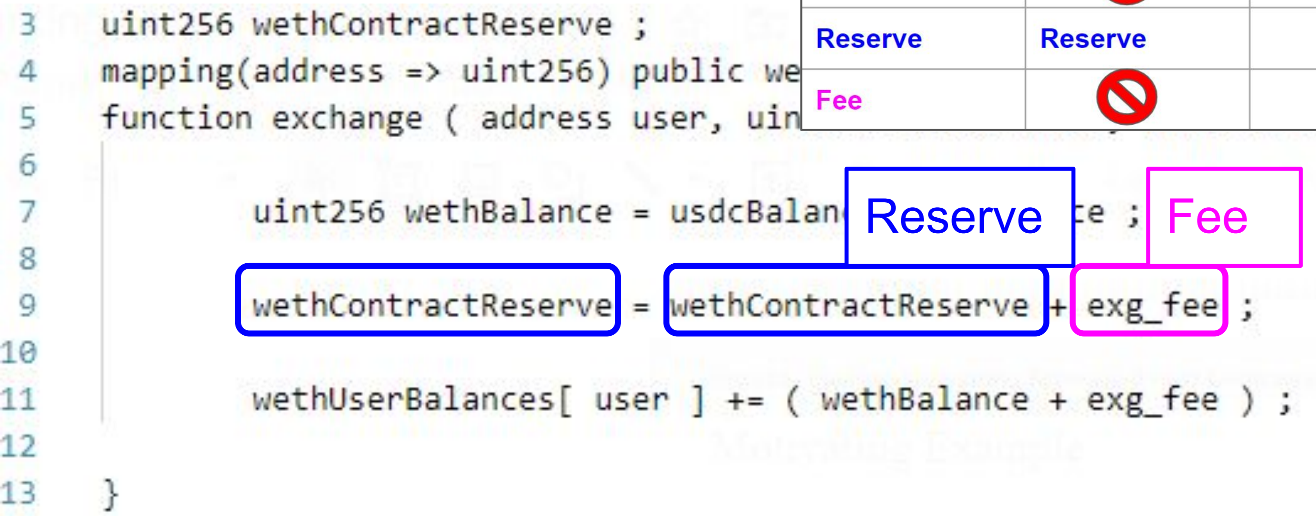


```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {

        uint256 wethBalance = usdcBalan                te ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

- “*wethContractReserve*” has financial meaning “Reserve”
  - “*wethContractReserve*” (Reserve) + “*exg_fee*” (Fee) = “*wethContractReserve*” (Reserve)

+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Reserve
Fee	🚫	🚫	Reserve	Fee

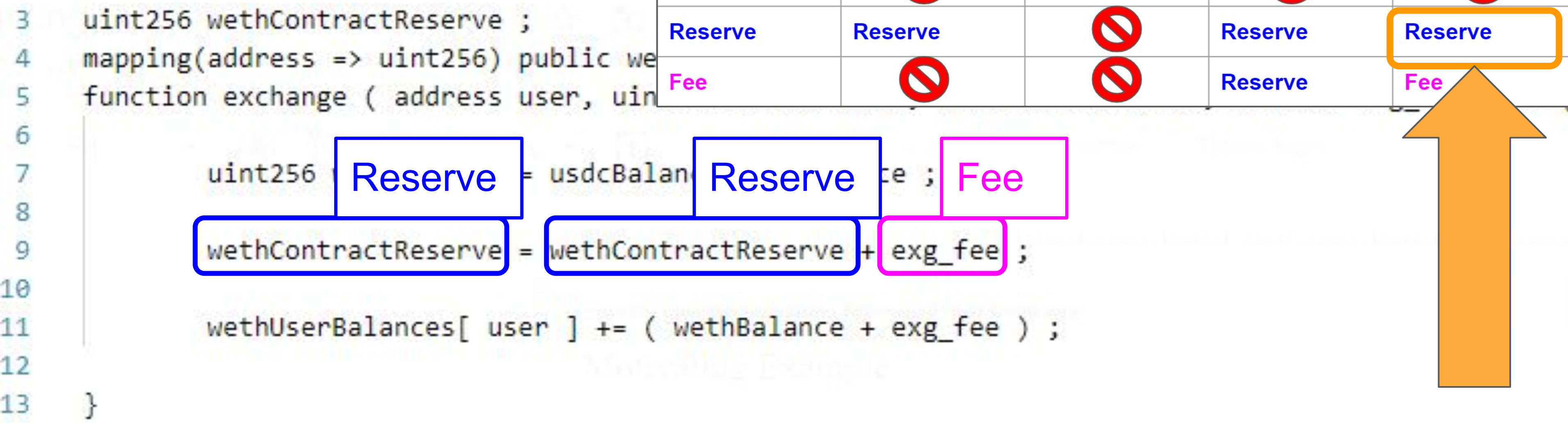


```
uint256 wethContractReserve ;
mapping(address => uint256) public we
function exchange ( address user, uin

        uint256 wethBalance = usdcBalan        te ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

- "*wethContractReserve*" has financial meaning "Reserve"
  - "*wethContractReserve*" (Reserve) + "*exg_fee*" (Fee) = "*wethContractReserve*" (Reserve)

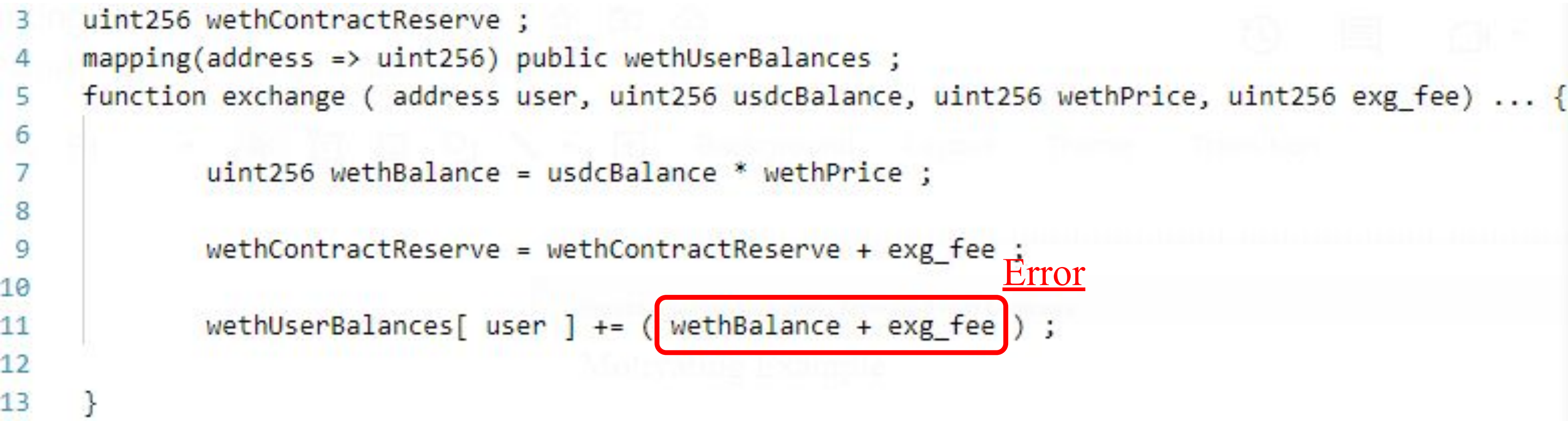
+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Reserve
Fee	🚫	🚫	Reserve	Fee



```
uint256 wethContractReserve ;
mapping(address => uint256) public we
function exchange ( address user, uin

        uint256         = usdcBalan         te ;

        wethContractReserve = wethContractReserve + exg_fee ;

        wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;

}
```

Motivating Example

- Error detected on line **11**
  - "*wethBalance*" (Raw Balance) + "*exg_fee*" (Fee) = Error
  - Intuitively, "Fee" should only be taken away from a user's "Raw Balance" and added to a contract's "Reserve"

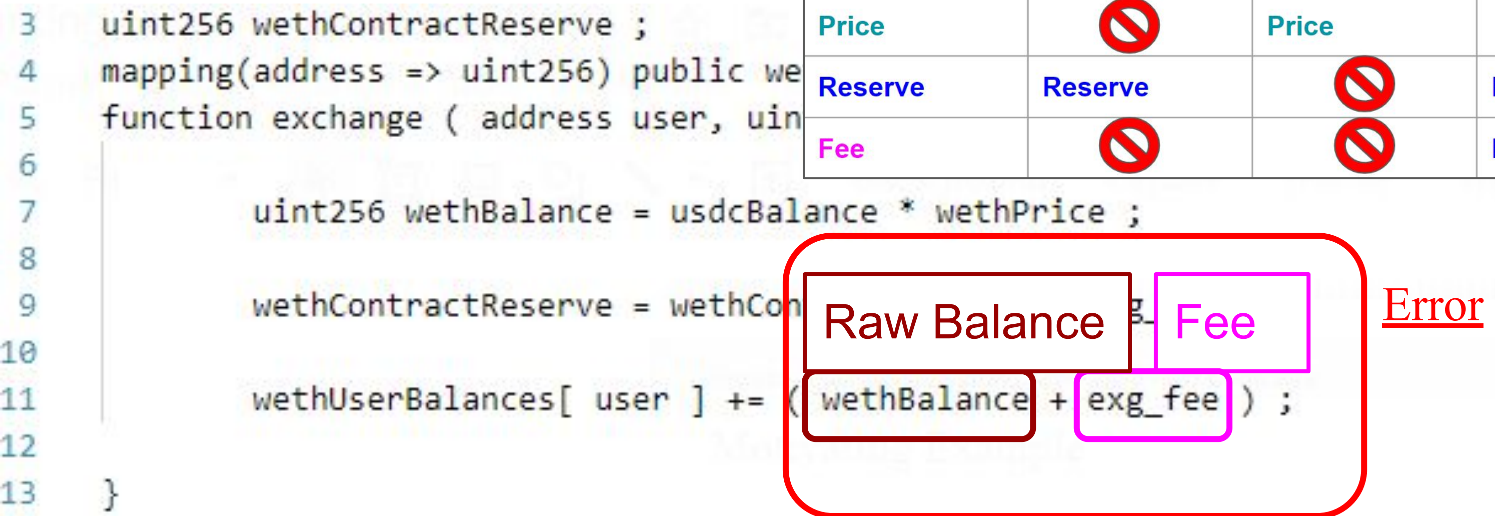


```
3   uint256 wethContractReserve ;
4   mapping(address => uint256) public wethUserBalances ;
5   function exchange ( address user, uint256 usdcBalance, uint256 wethPrice, uint256 exg_fee) ... {
6
7           uint256 wethBalance = usdcBalance * wethPrice ;
8
9           wethContractReserve = wethContractReserve + exg_fee ;
10
11          wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13  }
```

Motivating Example

- Error detected on line **11**
  - "*wethBalance*" (Raw Balance) + "*exg_fee*" (Fee) = Error
  - Intuitively, "Fee" should only be taken away from a user's "Raw Balance" and added to a contract's "Reserve"

+	Raw Balance	Price	Reserve	Fee
Raw Balance	Raw Balance	🚫	Reserve	🚫
Price	🚫	Price	🚫	🚫
Reserve	Reserve	🚫	Reserve	Rese
Fee	🚫	🚫	Reserve	Fee



```
3   uint256 wethContractReserve ;
4   mapping(address => uint256) public we
5   function exchange ( address user, uin
6
7           uint256 wethBalance = usdcBalance * wethPrice ;
8
9           wethContractReserve = wethCo                          g_
10
11          wethUserBalances[ user ] += ( wethBalance + exg_fee ) ;
12
13  }
```

Extended Type - Token Unit

- **Token Unit** represents the token denomination of a variable
  - Tokens are cryptocurrencies (Example 1: USDC, WETH)
- Intuitively, it is analogous to the symbols: “$” and “¥”

Extended Type - Scaling Factor

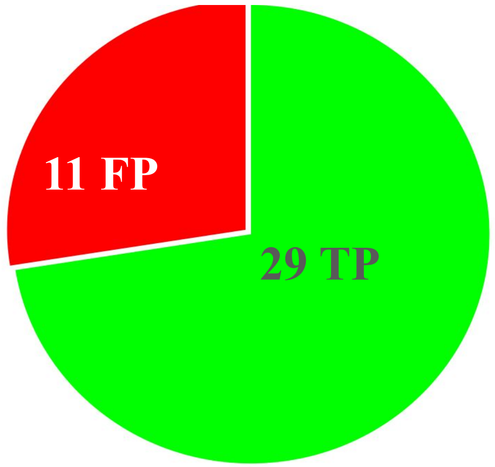
- Floating points are not supported in Solidity
  - Rely on scaling values by large factors (i.e. 1e18)
- Scaling Factor denotes how much a certain variable has been scaled

Research Questions

- **RQ1: How effective is ScType at disclosing accounting bugs?**
- **RQ2: How effective is ScType at disclosing zero-day vulnerabilities?**
- RQ3: How efficient is ScType?
- RQ4: What are the categories and distributions of accounting bugs?
- RQ5: What is the capacity of the type system?

RQ1: How effective is ScType at disclosing accounting bugs?

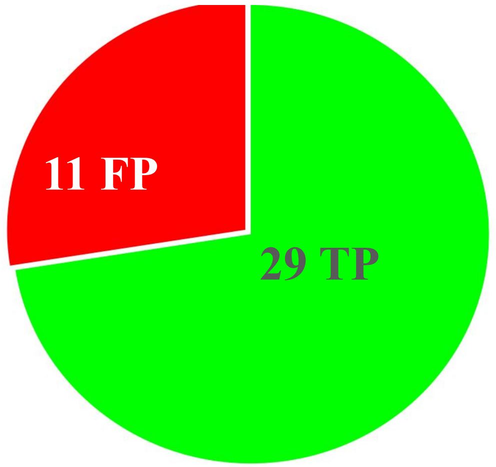
Ran ScType on **29 projects**, covering **57 accounting error bugs**



RQ1: How effective is ScType at disclosing accounting bugs?

Ran ScType on **29 projects**, covering **57 accounting error bugs**

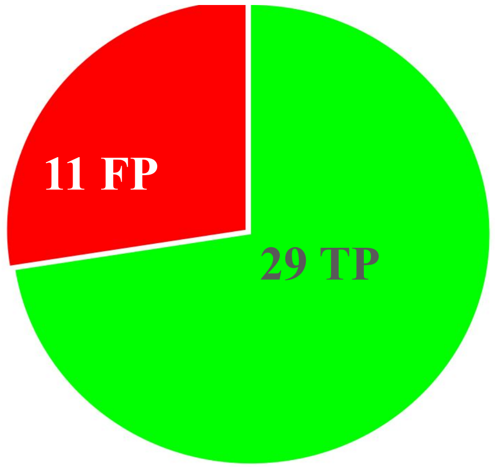
- Bugs taken from the previously mentioned Web3 Bug Database



RQ1: How effective is ScType at disclosing accounting bugs?

Ran ScType on **29 projects**, covering **57 accounting error bugs**

- Bugs taken from the previously mentioned Web3 Bug Database
- Of the 57 accounting error bugs, 24 are out of scope
  - Belong to other categories (i.e. Pure math errors)
- 57 -24 = 33 accounting error bugs are in scope

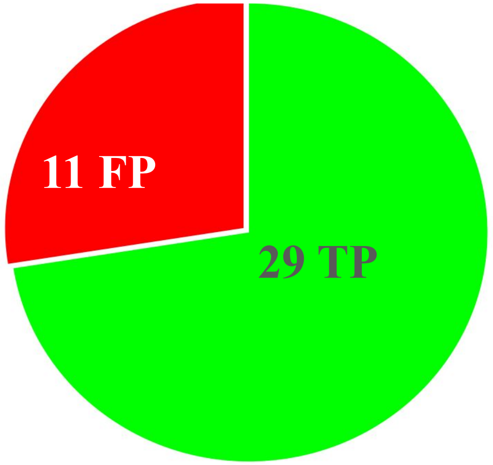


RQ1: How effective is ScType at disclosing accounting bugs?

Ran ScType on **29 projects**, covering **57 accounting error bugs**

- Bugs taken from the previously mentioned Web3 Bug Database
- Of the 57 accounting error bugs, 24 are out of scope
    - Belong to other categories (i.e. Pure math errors)
- 33 accounting error bugs are in scope

ScType reports 29 True Positives and 11 False Positives



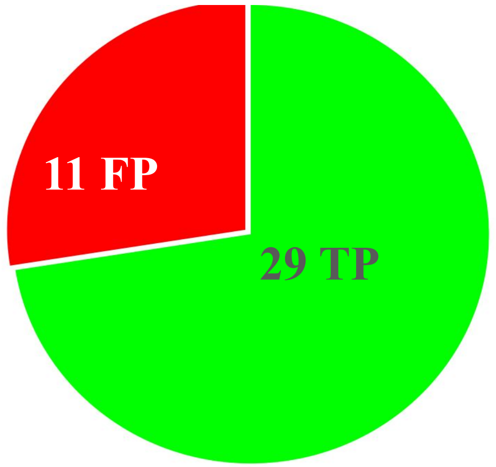
RQ1: How effective is ScType at disclosing accounting bugs?

Ran ScType on **29 projects**, covering **57 accounting error bugs**

- Bugs taken from the previously mentioned Web3 Bug Database
- Of the 57 accounting error bugs, 24 are out of scope
    - Belong to other categories (i.e. Pure math errors)
- 33 accounting error bugs are in scope

ScType reports <u>29 True Positives</u> and <u>11 False Positives</u>

It achieves an accuracy of: 29/33 = <u>**87.8%**</u>



RQ2: Effectiveness at disclosing zero-day vulnerabilities?

ScType was run on a large real-world contract, named Tapioca Dao through **Code4rena**

- Code4rena is a vendor for smart contract auditing competitions

RQ2: Effectiveness at disclosing zero-day vulnerabilities?

ScType was run on a large real-world contract, named Tapioca Dao through **Code4rena**

- Code4rena is a vendor for smart contract auditing competitions

ScType was run on 9 smart contracts

- Found **6 zero-day vulnerabilities**, 4 leading to direct fund loss
- Awarded **$6,000** as a result

Related Work

- **Liu and Y. Li, “Invcon: A dynamic invariant detector for ethereum smart contracts,” Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022.**
- **Y. Liu, Y. Li, S.-W. Lin, and R.-R. Zhao, “Towards automated verification of smart contract fairness,” Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020.**
- ***Smart Contract and DeFi Security Tools: Do They Meet the Needs of Practitioners?***

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics
- ScType is based on the insights that:

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics
- ScType is based on the insights that:
  - Mostly manifest as abstract type errors

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics
- ScType is based on the insights that:
  - Mostly manifest as abstract type errors
  - Smart contracts can be modeled in a way similar to banks

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics
- ScType is based on the insights that:
  - Mostly manifest as abstract type errors
  - Smart contracts can be modeled in a way similar to banks
- ExtendedType models Financial Meaning, Token Unit, and Scaling Factor
  - Capable of achieving 87% accuracy on the Benchmark

Take Away

We introduce **ScType**, an abstract type-checking tool, as a detector for Accounting Errors in Smart Contracts

- Accounting Errors are difficult to debug:
  - No general-testing oracles
  - Require understanding of complex business logics
- ScType is based on the insights that:
  - Mostly manifest as abstract type errors
  - Smart contracts can be modeled in a way similar to banks
- ExtendedType models Financial Meaning, Token Unit, and Scaling Factor
  - Capable of achieving 87% accuracy on the Benchmark
  - Caught 6 0-days with $6,000 in rewards

Thanks!

Reusable v1.1

Available v1.1

Github QRCode

Paper QRCode

Towards Finding Accounting Errors in Smart Contracts

- RQ2: How efficient is ScType?

TODO

Motivating Example 1

```
function getTotalBalance(address _user) external returns (uint256 total){
    uint256 USDCAmount = USDC.balanceOf(_user);
    uint256 WETHAmount = WETH.balanceOf(_user);
    uint256 totalAmount = USDCAmount + WETHAmount;
    USDC.transfer(_user, totalAmount);
}
```

Error should be reported here

[Corrected] Motivating Example 2

```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function applyTrade ( address user, uint256 usdcBalance, uint256 usdcToWethPrice, uint256 feeRate ) internal pure returns ... {
        uint256 wethBalance = usdcBalance * usdcToWethPrice / 1e18 ;
        uint256 fee = wethBalance * feeRate / 1e18 ;
        wethContractReserve += fee ;
        wethUserBalances[ user ] += ( wethBalance - fee ) ;
}
```

Motivating Example 2

```
uint256 wethContractReserve ;
mapping(address => uint256) public wethUserBalances ;
function applyTrade ( address user, uint256 usdcBalance, uint256 usdcToWethPrice, uint256 feeRate ) internal pure returns ... {

        uint256 wethBalance = usdcBalance * usdcToWethPrice / 1e18 ;

        uint256 fee = wethBalance * feeRate / 1e18 ;

        wethContractReserve += fee ;

        wethUserBalances[ user ] += ( wethBalance + fee ) ;

}
```

Error should be reported here

[ScType] Example 2

- RQ3: What are the categories and distributions of accounting bugs?

TODO

- RQ4: What is the capacity of the type system?

TODO

Towards Finding Accounting Errors in Smart Contracts

Slither?

Abstract typing?

RQ1: How effective is ScType at disclosing accounting bugs?

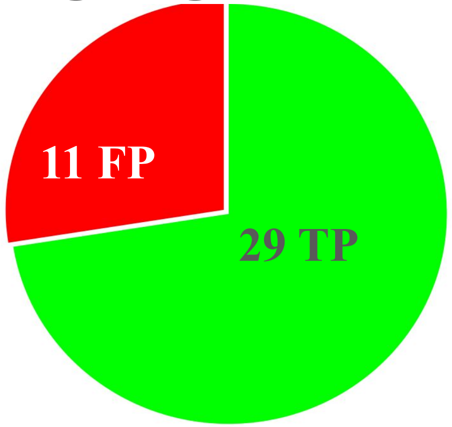
Ran ScType on 29 projects, covering 57 accounting error bugs

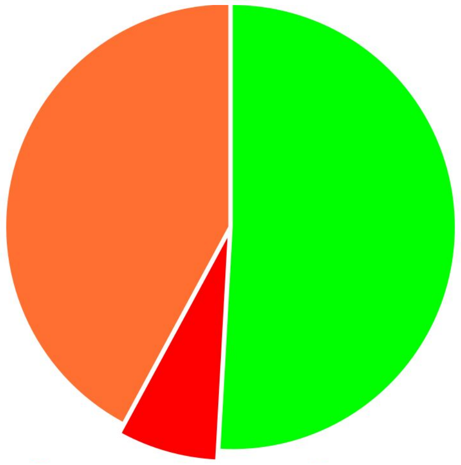
- Bugs taken from the previously mentioned Web3 Bug Database
- ScType reports 29 True Positives and 11 False Positives

Of the 57 accounting error bugs, 24 are out of scope

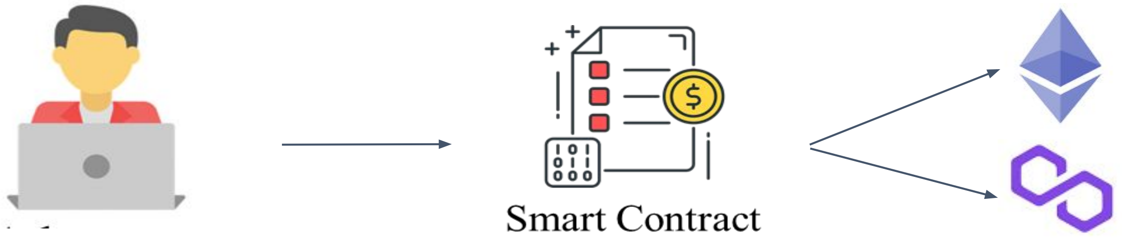
- Belong to other categories (i.e. Pure math errors)
- Hence, only 4 are not detected

ScType has an accuracy of 29/(29+4) = 87.9%





Spare pictures


Smart Contract