# ZEUS: Analyzing Safety of Smart Contracts

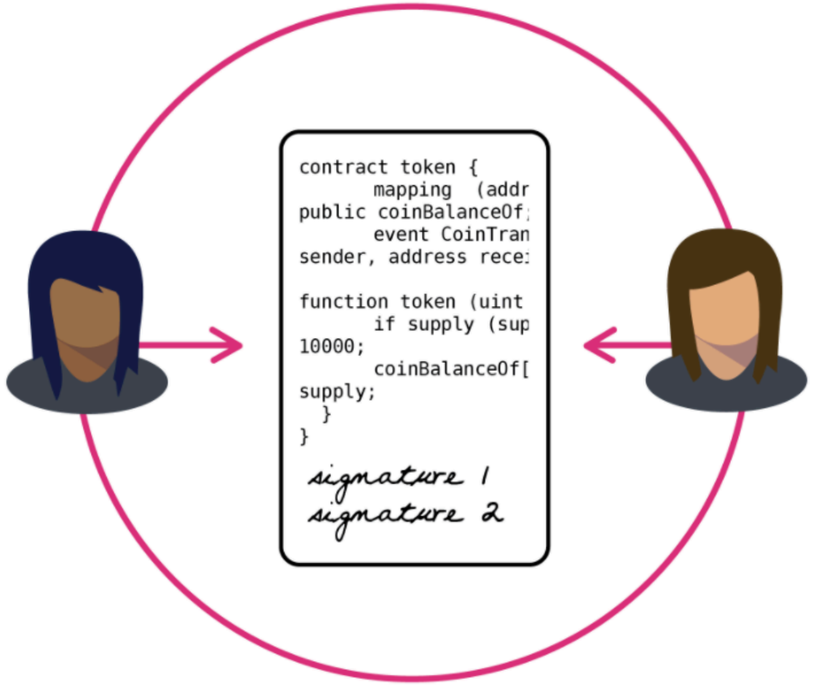

**Sukrit Kalra**
IBM Research

Seep Goel
IBM Research

Mohan Dhawan
IBM Research

Subodh Sharma
IIT-Delhi

# Smart Contracts




- Self-executing programs that encode the terms of interaction between multiple parties
- The code exists and runs on the blockchain network

# Smart Contracts

- The participating entities need to ensure:
  - **Correctness:** Syntactic implementation follows best practices
  - **Fairness:** Code adheres to higher-level business logic


# Correctness: The DAO

The New York Times

***A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency***

```
function withdrawRewardFor(address _account)
        returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```

# Correctness: The DAO

The New York Times

***A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency***

```
function withdrawRewardFor(address _account)
        returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```

# Correctness: The DAO

The New York Times

***A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency***

```
function withdrawRewardFor(address _account)
        returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```

# Fairness: AuctionHouse

- By law, auction can be of two types:
  - **With Reserve:** Seller is allowed to bid
  - **Without Reserve:** Seller is **not** allowed to bid

# Fairness: AuctionHouse

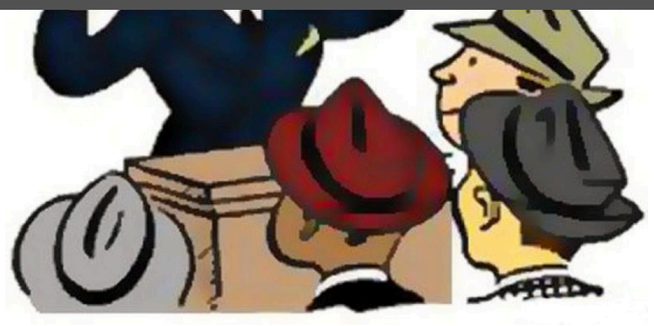
- By law, auction can be of two types:
  - **With Reserve:** Seller is allowed to bid
  - **Without Reserve:** Seller is **not** allowed to bid

# Fairness: AuctionHouse

- By law, auction can be of two types:

Seller shouldn't be able to bid on the auction #8

Open ericxtang opened this issue on Oct 16, 2016 · 2 comments bug



# Outline

- Overview
- **Motivation**
- **Zeus**
- **Implementation**
- **Evaluation**
- **Conclusion**

# Motivation


Michael del Castillo

Jun 17, 2016 at 14:00 UTC | Updated Jun 18, 2016 at 14:46 UTC

NEWS

The DAO, the distributed autonomous organization that had collected over $150m worth of the cryptocurrency ether, has reportedly been hacked, sparking a broad market sell-off.

A leaderless organization comprised of a series of smart contracts written on the ethereum codebase, The DAO has lost 3.6m ether, which is currently sitting in a separate wallet after being split off into a separate grouping dubbed a "child DAO"

# Motivation


Mail Online | TV&Showbiz | Femail | Health | Science | Money | Videc

Discounts

## £200 million worth of digital cryptocurrency is wiped out as bungling developer locks investors out while trying to stop hackers

Michael del C

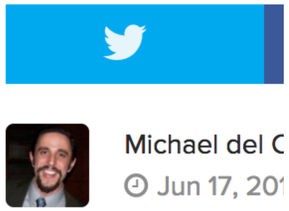
Jun 17, 201

- A developer was fixing a bug that let hackers steal funds from virtual wallets
- But the developer accidentally left a second flaw in its systems
- When the user tried to undo the damage by deleting the flaw in the code, this locked the funds in the wallets permanently
- The only way to reverse the issue is a 'hard-fork', but not everyone supports this

The DAO, the di
cryptocurrency e

A leaderless org
codebase, The D
split off into a se

# Motivation



Michael del C
Jun 17, 201

The DAO, the di
cryptocurrency e

A leaderless org
codebase, The D
split off into a se


# £200 mill cryptocu bungling while tryi

- A developer was f
- But the developer
- When the user trie locked the funds i
- The only way to re

BBC

# Code bug freezes $150m of Ethereum crypto-cash

9 November 2017

Share

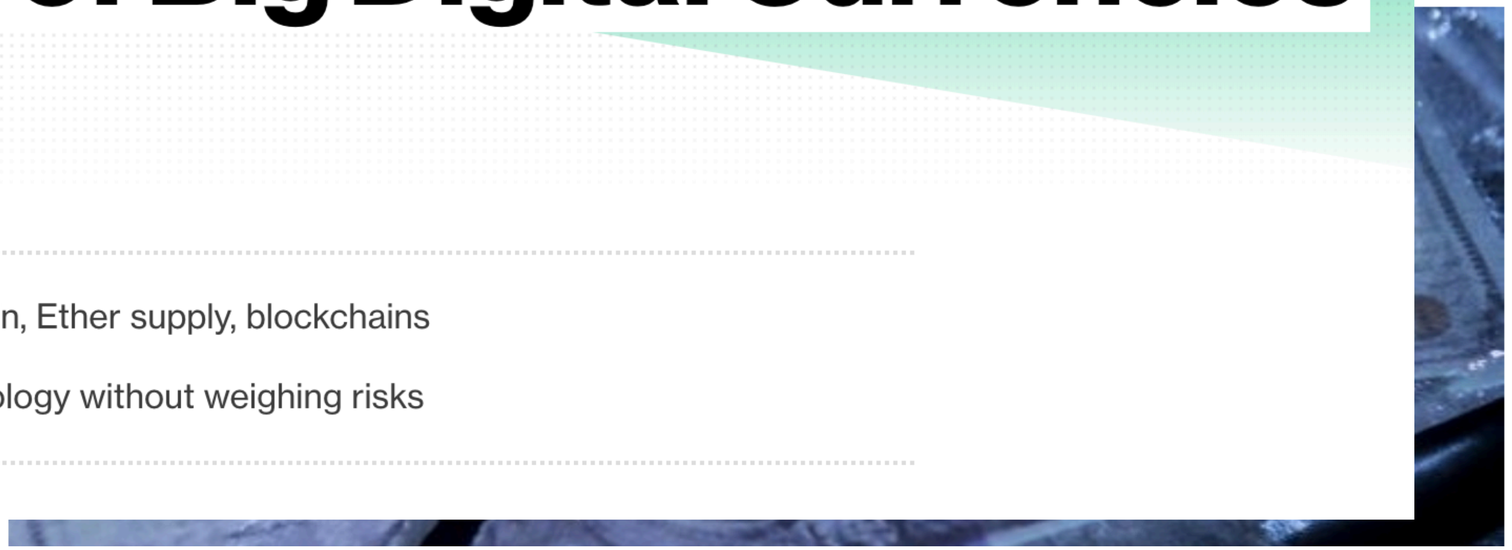
# Motivation

## Hackers Have Walked Off With About 14% of Big Digital Currencies

By **Olga Kharif**

January 18, 2018, 7:19 PM GMT+5:30

- Cybercriminals compromise Bitcoin, Ether supply, blockchains
- Crypto-crazed users adopt technology without weighing risks

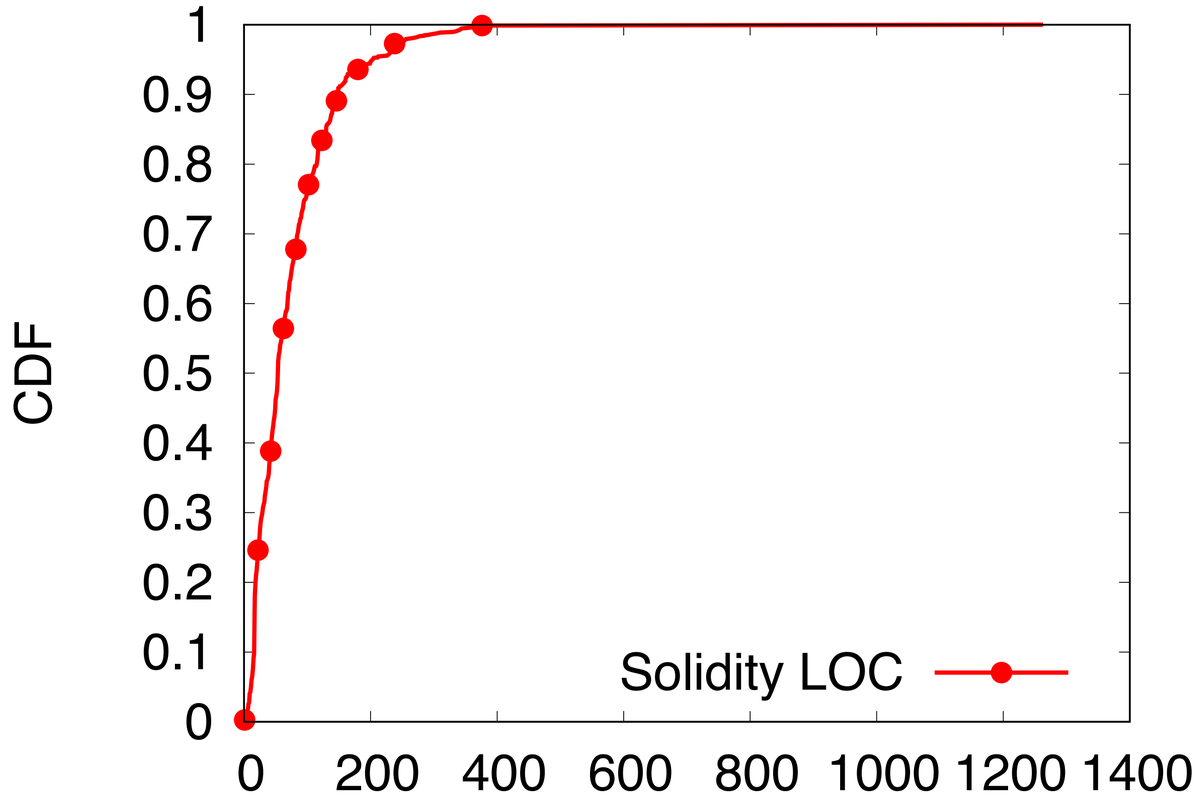


# Verification

- Verification suffers from **state space explosion!**

# Verification

- Verification suffers from **state space explosion!**



# Verification

- Verification suffers from **state space explosion!**


**Solidity contracts are small**

# Smart Contract Verification

- Oyente (CCS ’16) uses symbolic execution for **bug detection** at the bytecode level
  - Neither sound nor complete
  - Cannot handle fairness issues
- Bhargavan *et al.* (PLAS ’16) formally verify contracts written in a **subset of solidity** using F*
  - Require manual proofs
  - Important constructs unsupported

# Failed send()

```
for (uint i = 0; i < investors.length; i++) {
    if (investors[i].invested == minimum) {
        payout = investors[i].payout;
        if (!investors[i].address.send(payout))
            throw;
        investors[i] = newInvestor;
    }
}
```

- `send()` is used to transfer money in contracts
- A failed `send()` can lock contracts!

# Failed send()

```
for (uint i = 0; i < investors.length; i++) {
    if (investors[i].invested == minimum) {
        payout = investors[i].payout;
        if (!investors[i].address.send(payout))
            throw;
        investors[i] = newInvestor;
    }
}
```

- `send()` is used to transfer money in contracts
- A failed `send()` can lock contracts!

# Block State Dependence

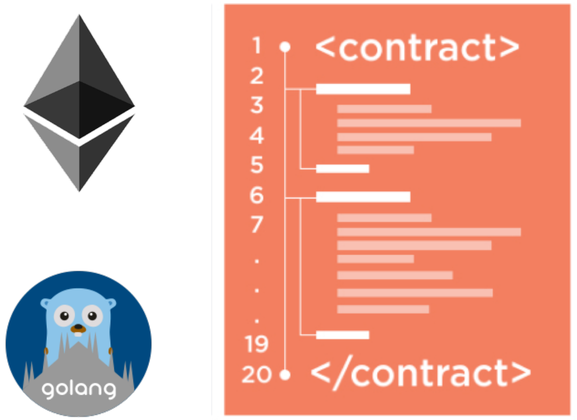
```
function resetInvestment() {
    if (block.timestamp <
            lastInvestment + ONE_MINUTE)
        throw;

    lastInvestor.send(jackpot);
    ...
}
```

- Block state variables are used to generate randomness
- They can be tampered by the miner for profit!

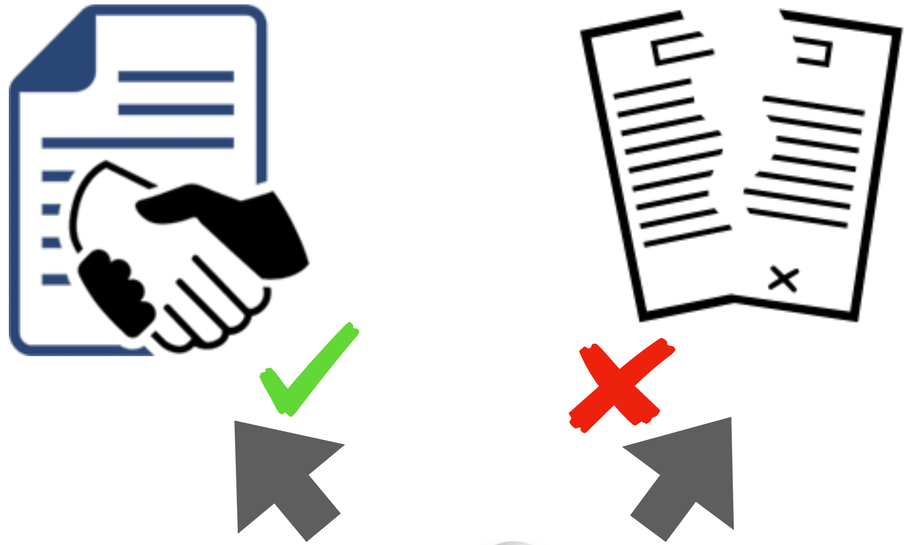
# Outline

- Overview
- Motivation
- **Zeus**
- **Implementation**
- **Evaluation**
- **Conclusion**

# Our Approach

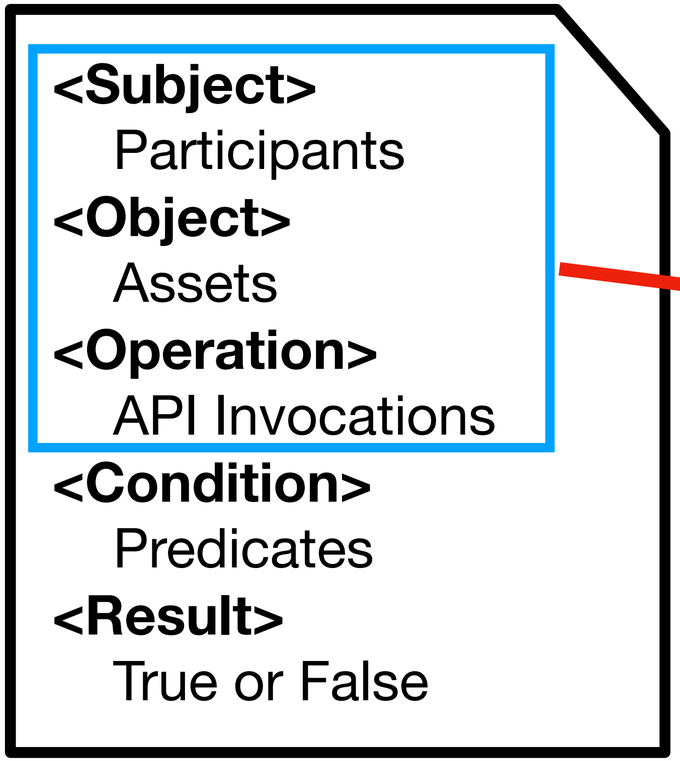




# Our Approach


Correct placement of asserts

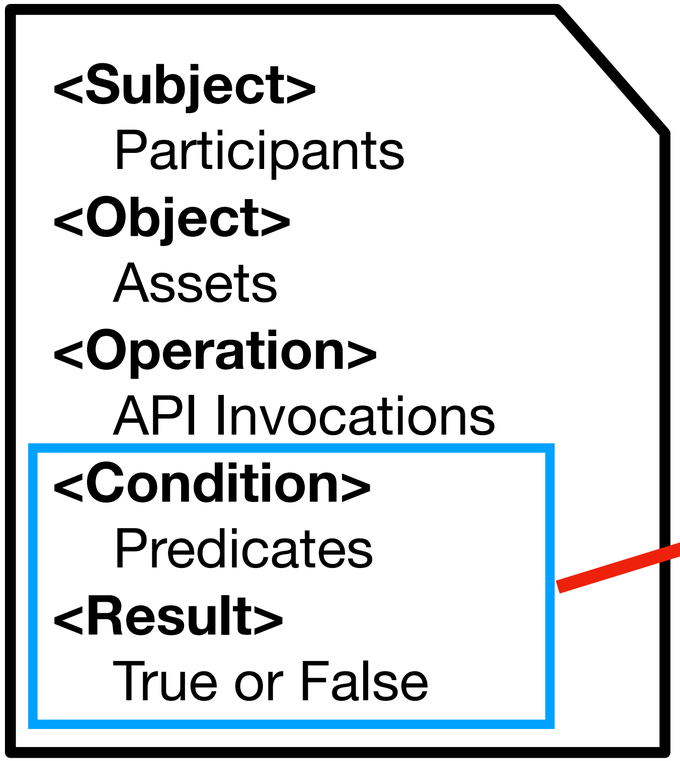

# Placement of asserts




```
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```

- **<Subject, Object, Operation>** determine the location

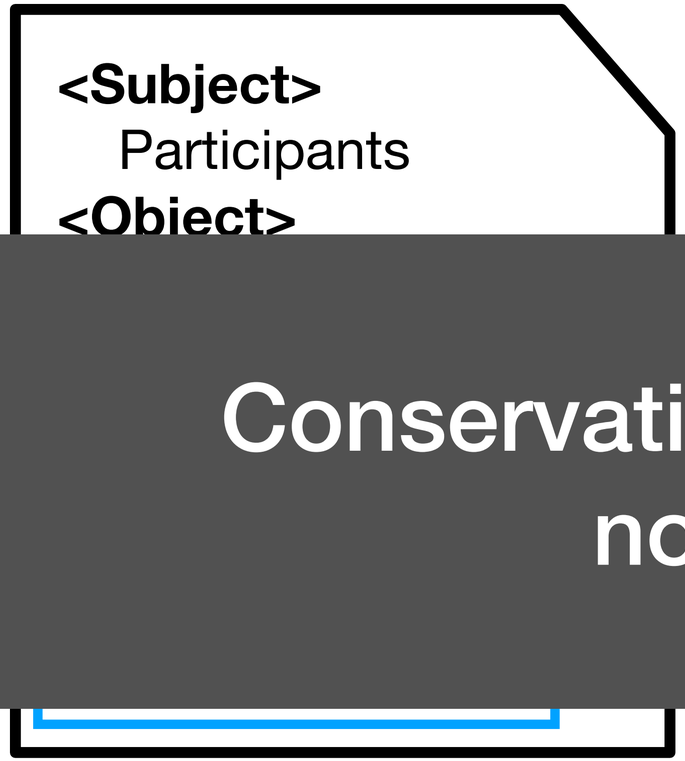
# Placement of asserts




```
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```

- **<Condition, Result>** determine the predicate

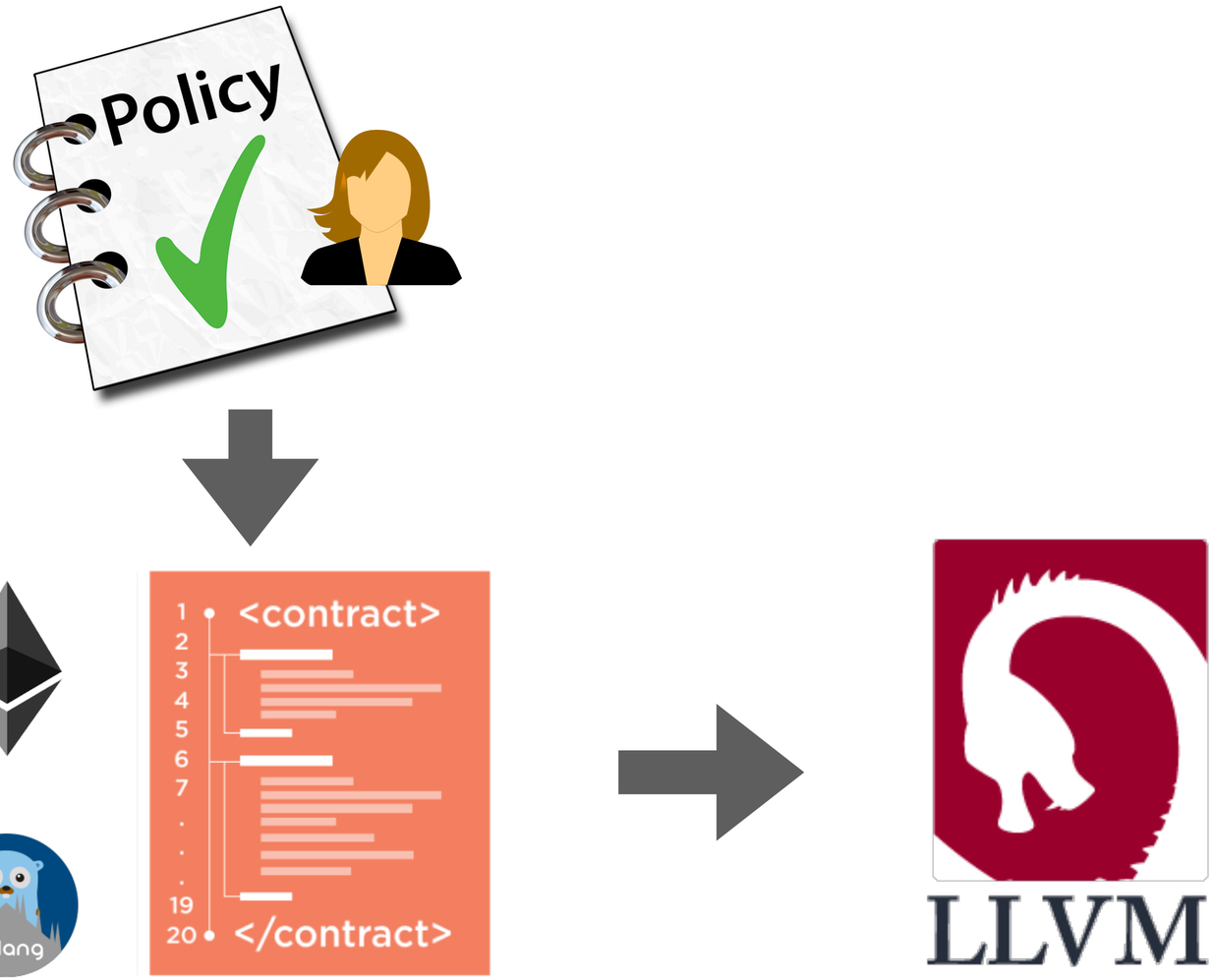
# Placement of asserts




Conservative taint analysis ensures
no false negatives

- **<Condition, Result>** determine the predicate

# Our Approach



# Our Approach


Sound over-approximation of the Solidity semantics



# Soundness

```
havoc(msg.value);

function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```

- `havoc(...)` expands the domain of legitimate values that a variable can take to the type-defined domain
- Ensures reasoning about all program paths

# Our Approach

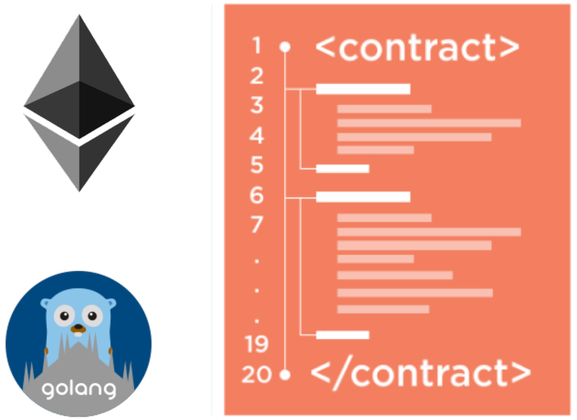

Protection against aggressive LLVM optimizations




# LLVM Optimizations

- Optimizations introduce architecture specific functions which may not be modeled in the verifier
  - `add` replaced with `uadd.with.overflow` which is not modeled
  - **Solution:** Enforce no optimizations
- Verifier eliminates non side-affecting variables and function calls
  - `send(...)` is optimized away
  - **Solution:** Global variable for external function returns

# Our Approach



# End-To-End Example


```
<Subject> msg.sender </Subject>
<Object> msg.value </Object>
<Operation trigger="pre"> send </Operation>
<Condition> msg.value <= balance </Condition>
<Result> True </Result>
```


```
function transfer() {
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```

# End-To-End Example



```
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```

# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

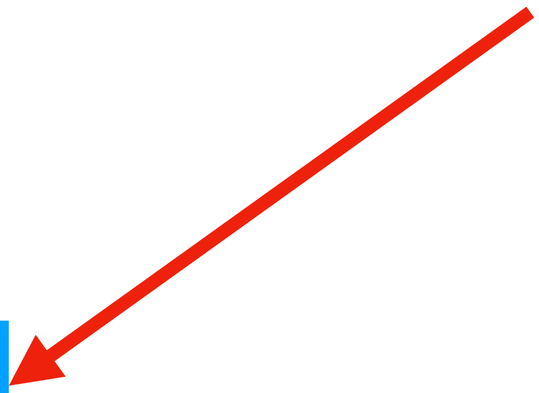
# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

An assert failure is modeled as a call to the verifier's error function

# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```



Globals are automatically `havoc`-ed to explore the entire data domain

# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %  sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

The return value of `send` is stored in a global variable

# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

# End-To-End Example

```
define void @transfer() {
entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* % __value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"

"74":                                             ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75":                                             ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

# Outline

- Overview
- Motivation
- Zeus
- **Implementation**
- **Evaluation**
- **Conclusion**

# Implementation

- Policy Builder
  - Extracts information from the AST nodes in `solc`
  - Taint analysis to retrieve the policy tuple (~500 LOC C++)
- Solidity to LLVM Translator
  - Generates the LLVM bitcode for the contract (~3000 LOC C++)
  - LLVM passes to automatically insert assertions for correctness bugs
- Verifier
  - Off-the-shelf model checkers that work with LLVM (Seahorn, SMACK)

# Outline

- Overview
- Motivation
- Zeus
- Implementation
- **Evaluation**
- **Conclusion**

# Methodology

- Study over 22.4K Solidity contracts (1524 unique)
- Verifier timeout threshold of 1 min
- Correctness bugs
  - Manually ascertain ground truth for 7 bug classes
- Fairness issues and case study on Hyperledger discussed in the paper

# Methodology

- Study over
- Verifier time
- Correctness
  - Manually
- Fairness iss

| Category | # Contracts | Lines of Code (K) | |
|---|---|---|---|
| | | Source | LLVM |
| DAO | 140 | 2.8 | 24.3 |
| Game | 244 | 23.3 | 609.2 |
| Token | 290 | 25.2 | 385.9 |
| Wallet | 72 | 10.8 | 105.9 |
| Misc. | 778 | 47.6 | 924.3 |
| **Total** | **1524** | **109.7** | **2049.6** |

the paper

# Correctness

| Bug | Zeus | | | | | | | Oyente (CCS '16) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm |
| Reentrancy | 1438 | 54 | 7 | 25 | 0 | 0 | 0.00 | 548 | 265 | 226 | 485 | 254 | 51 | 31.24 |
| Unchecked send | 1191 | 324 | 5 | 4 | 3 | 0 | 0.20 | 1066 | 112 | 203 | 143 | 89 | 188 | 7.56 |
| Failed send | 1068 | 447 | 3 | 6 | 0 | 0 | 0.00 | | | | | | | |
| Integer Overflow | 378 | 1095 | 18 | 33 | 40 | 0 | 2.72 | | | | | | | |
| Transaction State Dependence | 1513 | 8 | 0 | 3 | 0 | 0 | 0.00 | | | | | | | |
| Block State Dependence | 1266 | 250 | 3 | 5 | 0 | 0 | 0.00 | 798 | 15 | 226 | 485 | 2 | 84 | 0.25 |
| Transaction Order Dependence | 894 | 607 | 13 | 10 | 16 | 0 | 1.07 | 668 | 129 | 222 | 485 | 116 | 158 | 14.20 |

# Correctness

| Bug | Zeus | | | | | | | Oyente (CCS '16) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm |
| Reentrancy | 1438 | 54 | 7 | 25 | 0 | **0** | 0.00 | 548 | 265 | 226 | 485 | 254 | **51** | 31.24 |
| Unchecked send | 1191 | 324 | 5 | 4 | 3 | **0** | 0.20 | 1066 | 112 | 203 | 143 | 89 | **188** | 7.56 |
| Failed send | 1068 | 447 | 3 | 6 | 0 | **0** | 0.00 | | | | | | | |
| Integer Overflow | 378 | 1095 | 18 | 33 | 40 | **0** | 2.72 | | | | | | | |
| Transaction State Dependence | 1513 | 8 | 0 | 3 | 0 | **0** | 0.00 | | | | | | | |
| Block State Dependence | 1266 | 250 | 3 | 5 | 0 | **0** | 0.00 | 798 | 15 | 226 | 485 | 2 | **84** | 0.25 |
| Transaction Order Dependence | 894 | 607 | 13 | 10 | 16 | **0** | 1.07 | 668 | 129 | 222 | 485 | 116 | **158** | 14.20 |

# Correctness

| Bug | Zeus | | | | | | | Oyente (CCS '16) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm |
| Dependence | | | | | | | | | | | | | | |
| Block State Dependence | 1266 | 250 | 3 | 5 | 0 | **0** | 0.00 | 798 | 15 | 226 | 485 | 2 | **84** | 0.25 |
| Transaction Order Dependence | 894 | 607 | 13 | 10 | 16 | **0** | 1.07 | 668 | 129 | 222 | 485 | 116 | **158** | 14.20 |

Zeus has no false negatives

# Correctness

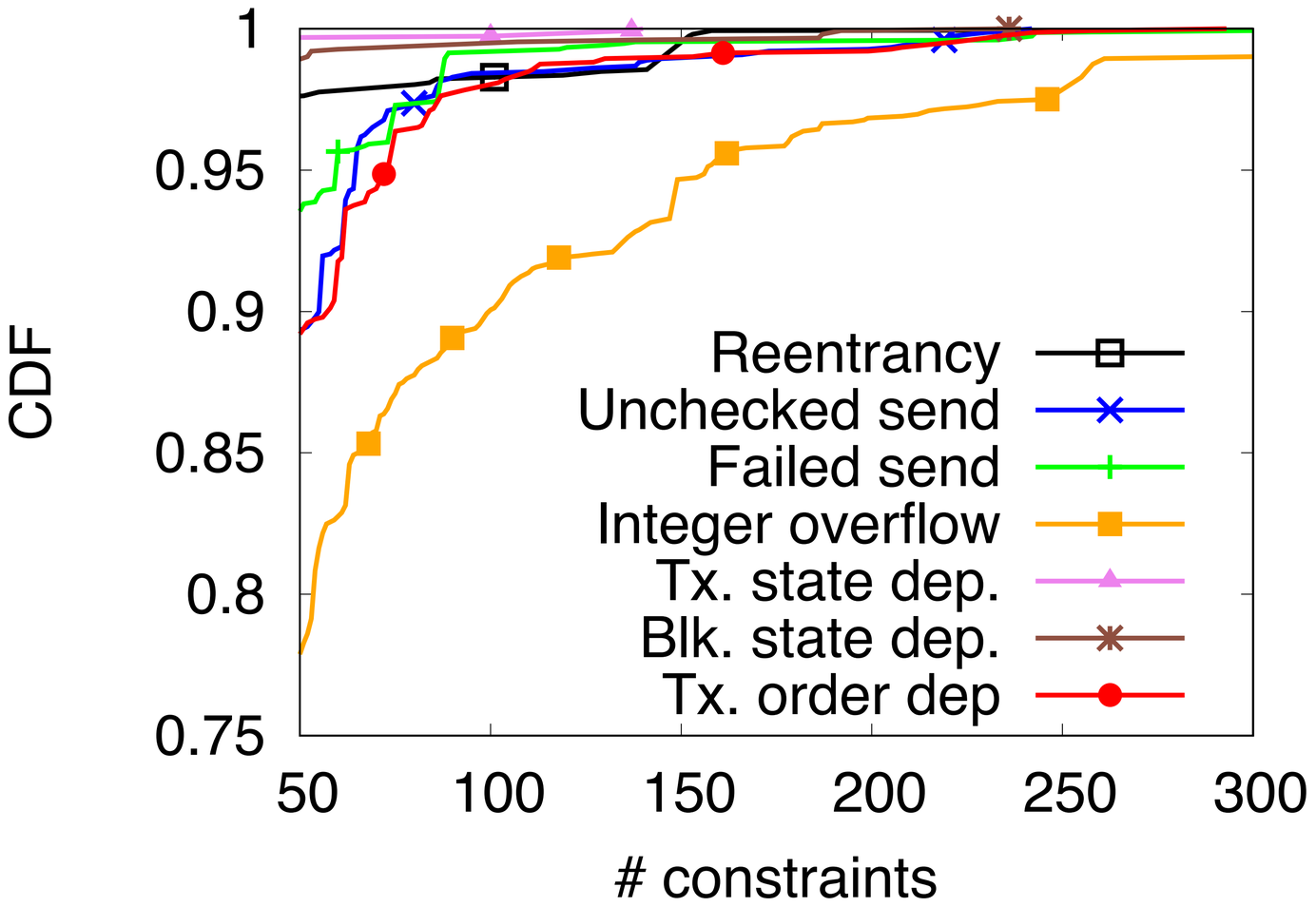
| Bug | Zeus | | | | | | | Oyente (CCS '16) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm |
| Reentrancy | 1438 | 54 | 7 | 25 | **0** | 0 | 0.00 | 548 | 265 | 226 | 485 | **254** | 51 | 31.24 |
| Unchecked send | 1191 | 324 | 5 | 4 | **3** | 0 | 0.20 | 1066 | 112 | 203 | 143 | **89** | 188 | 7.56 |
| Failed send | 1068 | 447 | 3 | 6 | **0** | 0 | 0.00 | | | | | | | |
| Integer Overflow | 378 | 1095 | 18 | 33 | **40** | 0 | 2.72 | | | | | | | |
| Transaction State Dependence | 1513 | 8 | 0 | 3 | **0** | 0 | 0.00 | | | | | | | |
| Block State Dependence | 1266 | 250 | 3 | 5 | **0** | 0 | 0.00 | 798 | 15 | 226 | 485 | **2** | 84 | 0.25 |
| Transaction Order Dependence | 894 | 607 | 13 | 10 | **16** | 0 | 1.07 | 668 | 129 | 222 | 485 | **116** | 158 | 14.20 |

# Correctness

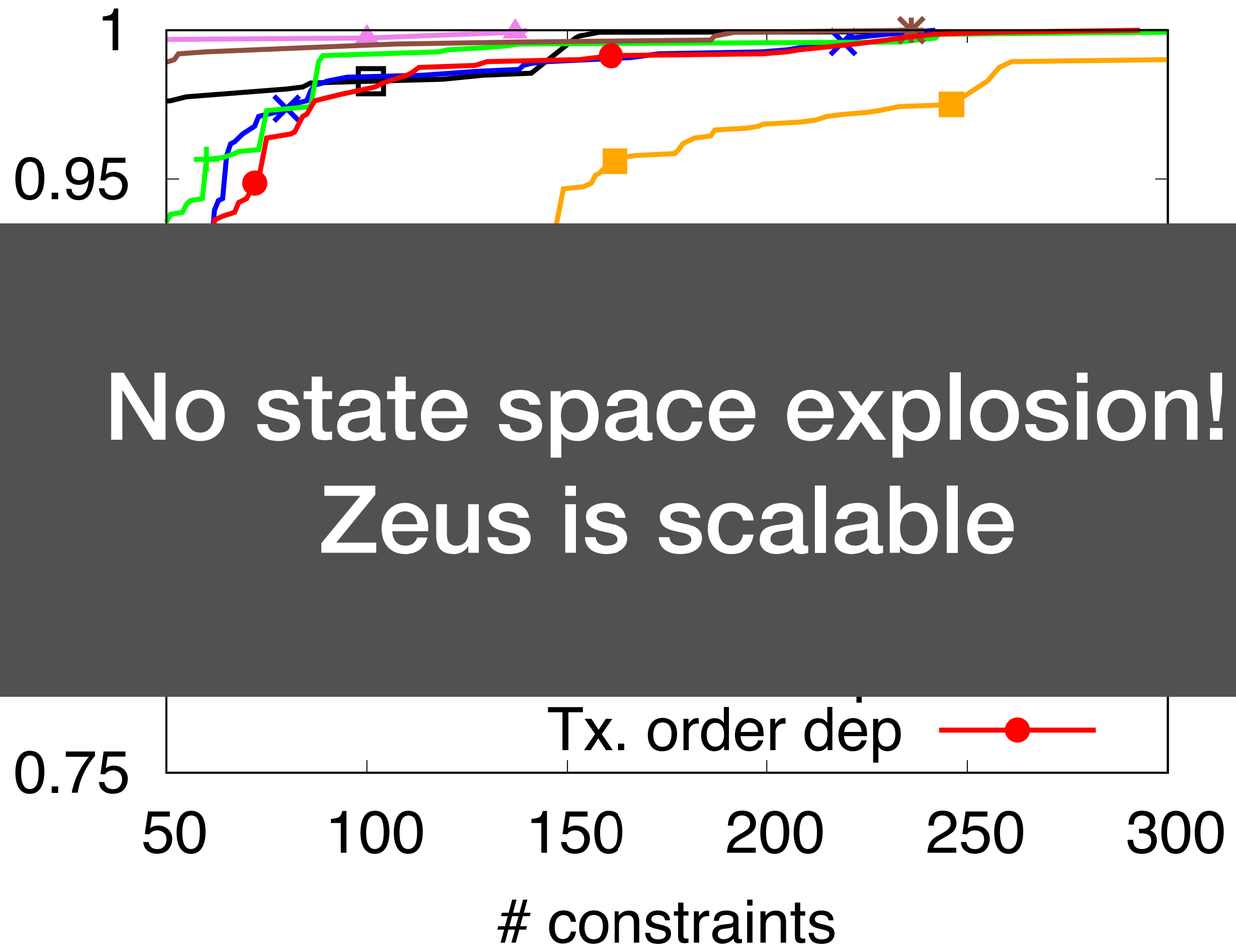
| Bug | Zeus | | | | | | | Oyente (CCS '16) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm | Safe | Unsafe | No Result | Time Out | False +ve | False -ve | % False Alarm |
| Dependence | | | | | | | | | | | | | | |
| Block State Dependence | 1266 | 250 | 3 | 5 | **0** | 0 | 0.00 | 798 | 15 | 226 | 485 | **2** | 84 | 0.25 |
| Transaction Order Dependence | 894 | 607 | 13 | 10 | **16** | 0 | 1.07 | 668 | 129 | 222 | 485 | **116** | 158 | 14.20 |

Zeus has lesser false positives

# Verification Complexity

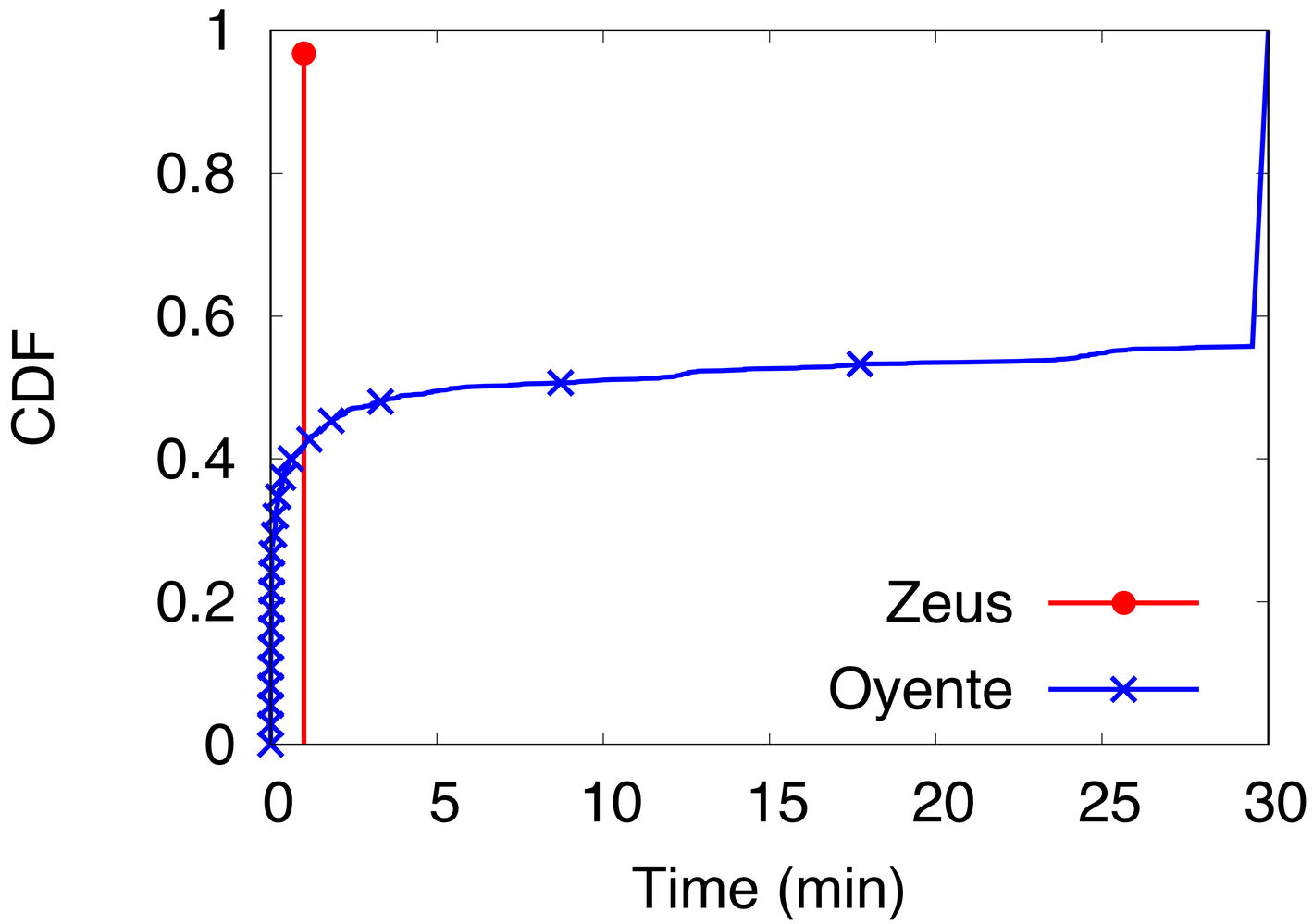


# Verification Complexity

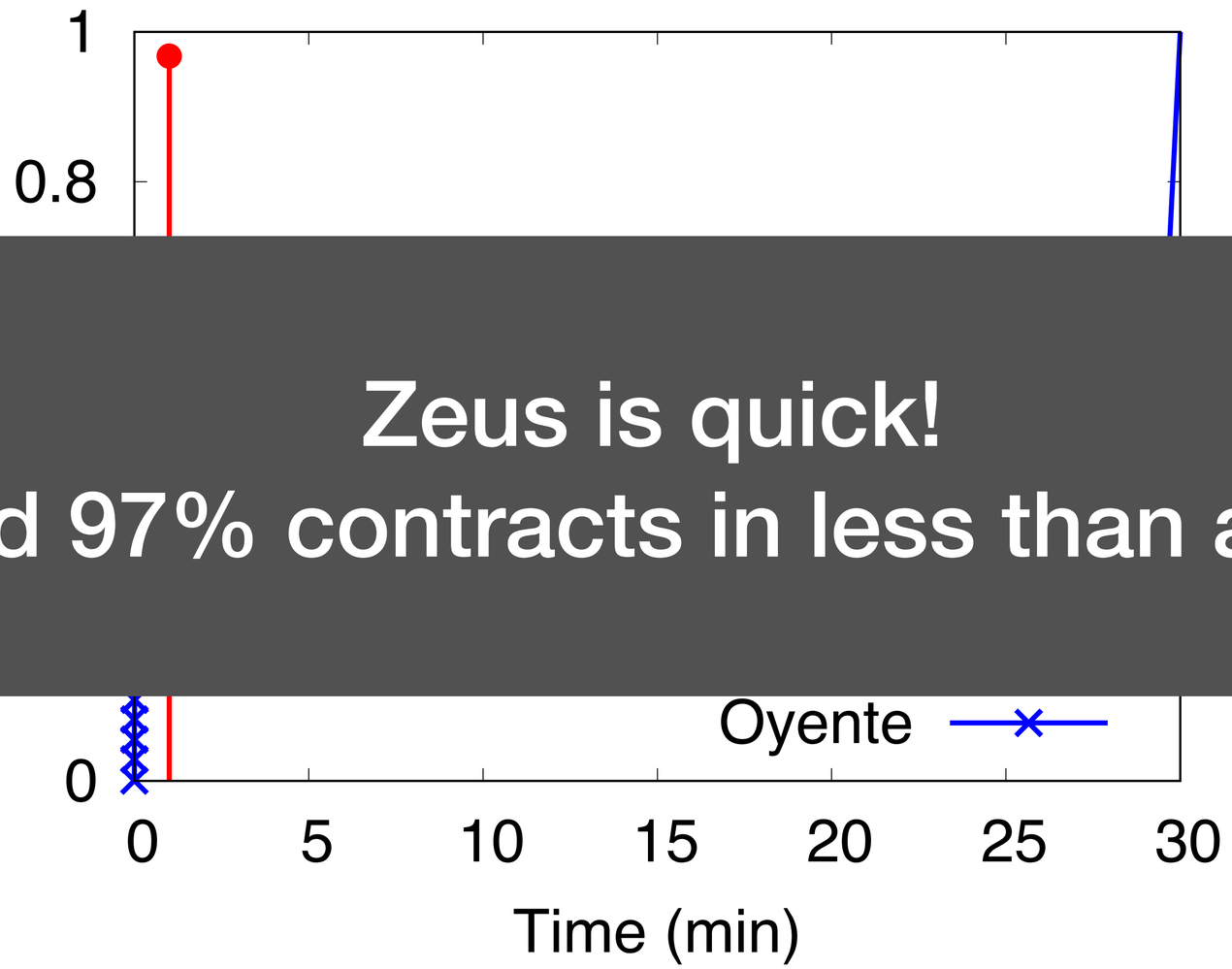



**No state space explosion!**
**Zeus is scalable**

# Verification Time



# Verification Time




**Zeus is quick!**
**Verified 97% contracts in less than a minute**

# Outline

- Overview
- Motivation
- Zeus
- Implementation
- Evaluation
- **Conclusion**

# Conclusion

- Smart contracts are buggy
  - Faithful execution ensured by consensus
  - Correctness and Fairness not guaranteed
- Zeus is a framework enabling verification of smart contracts
  - Works at scale
    - Study over 22.4K Solidity contracts (1524 unique)
    - Around 94% contracts vulnerable to correctness bugs
  - Sound with low verification overhead
    - Zero false negatives, lesser false positives
    - Takes under 1 min to analyze 97% contracts

# Thank You!

**Contact:** sukrit.kalra@in.ibm.com