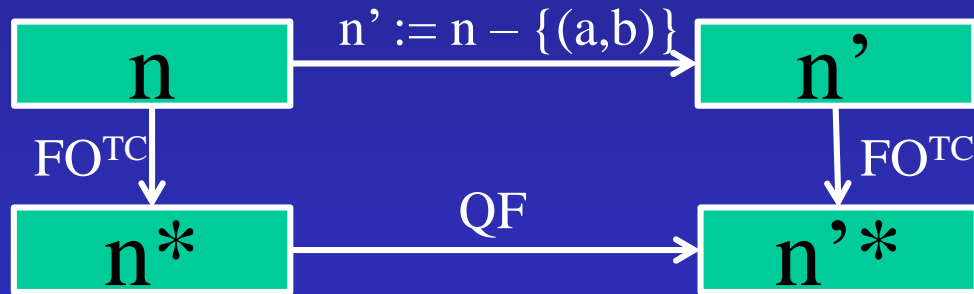


The magic behind FO modeling (take 1)

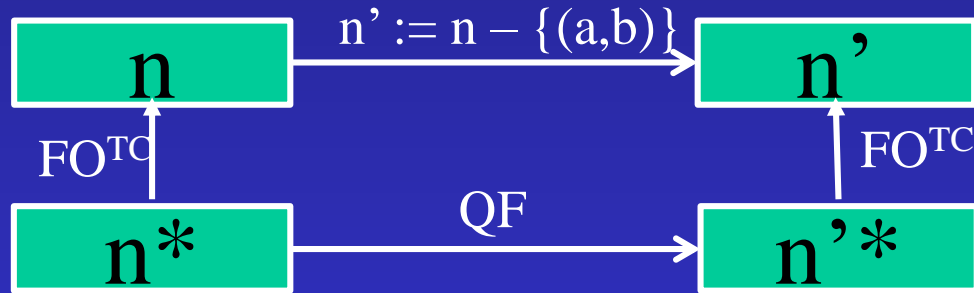
- Employ incremental updates (n is deterministic)



Neil Immerman: Descriptive complexity. Graduate texts in computer science, Springer 1999, ISBN 978-1-4612-6809-3, pp. I-XVI, 1-268

The magic behind FO modeling (take 2)

- Invert $n^* \rightarrow n$ (deterministic)



$$\forall X. n^*(X, X)$$

$$\forall X, Y. n^*(X, Y) \wedge n^*(Y, X) \Rightarrow X=Y$$

$$\forall X, Y, Z. n^*(X, Y) \wedge n^*(Y, Z) \Rightarrow n^*(X, Z)$$

$$\forall X, Y, Z. n^*(X, Y) \wedge n^*(X, Z) \Rightarrow n^*(Y, Z) \vee n^*(Z, Y)$$

$$\forall X, Y. n(X, Y) \Leftrightarrow n^*(X, Y) \wedge \forall Z. n^*(X, Z) \wedge Z \neq X \Rightarrow n^*(Y, Z)$$

What can get wrong with Ivy?

- Design
 - Axioms
 - Transition Relation
 - Initial state
 - Safety Property
 - Inductive invariants
 - Outside EPR
 - Z3 bugs
- Implementation
 - Data structure does not refine design
 - Outside FAU
 - Inefficient data structure
 - Wrong data structure implementation
 - Z3 bugs

transition $t_1 t_2 \dots t_k$

bmc k



CERTORA

Building Trust in Software
The Smart Contract Spec-hub

AUGUST 2019

The Team



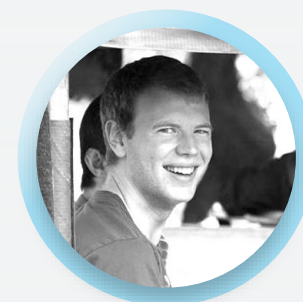
Mooly Sagiv
CEO & Founder



Shelly Grossman
Chief Scientist & Founder



Marcelo Taube
Product Architect



James Wilcox
CTO



Alexander Nutz
Researcher



Pain Point: Buggy & Untrusted Software Components

- Software is Eating the World
- Software applications are built of numerous disparate sources
 - unknown
 - untrusted
 - constantly evolving
- Correctness of code = safety, money, human life, ...
- Even worse in blockchain
 - Immutability: code is law
 - Cryptocurrency: code is money

The Business Case

- **Problem:** Automated financial contracts
 - Bugs in contracts = money lost to adversaries for ever
- **Pain:** Very hard to find bugs in the contracts
 - Lots of examples where people have lost large amounts of money
 - Customers are willing to ≥ 6 figures for solutions
- **Solution:** Automatic Verification
 - Find bugs, or prove the absence of bugs (one or the other!)
 - Key enabler: **higher level specifications that can be checked**
 - Developers are willing to do most of the work, for reward
 - Standards ERC20, ERC721

Business Snapshot

- Compound Finance

“We installed Certora's technology and it is used daily by our software engineers to locate mind blowing bugs” **Geoff Hayes** CTO of Compound Finance

- Coinbase

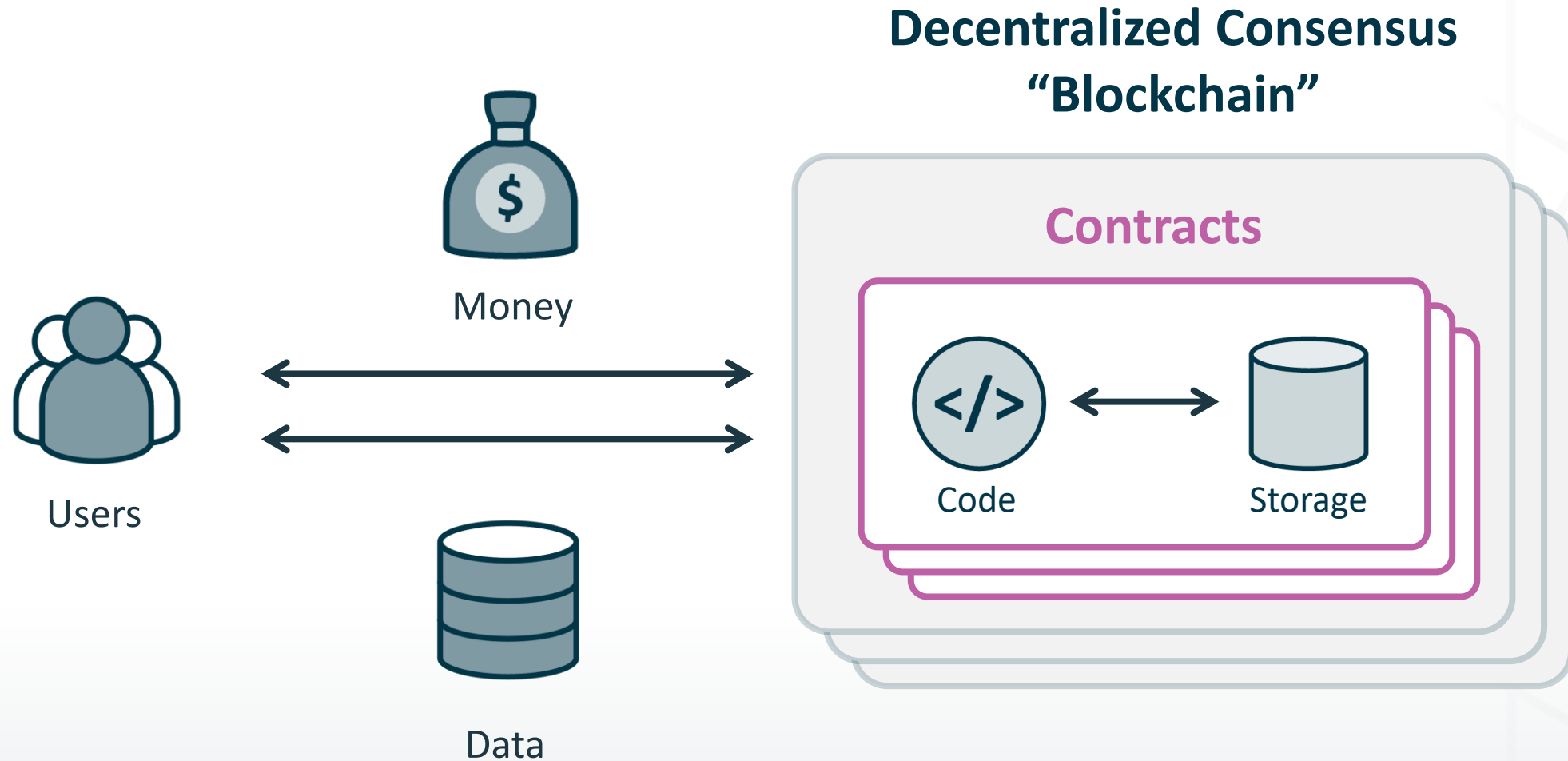
“The Certora ASA surfaces problems before a contract is available on our platforms, to help us better inform our customers of risk. The ASA has already surfaced significant problems missed by expensive and unscalable manual audits.”

Shamiq Islam Head of Security at Coinbase Global

- Celo

“We have decided to deploy Certora’s ASA to continuously verify the code of our governance protocol during the development process. Certora’s ASA tool has already uncovered a number of nuanced bugs, including one in our sorted linked-list, and also mathematically proved interesting properties of linked lists.” **Marek Olszweski** Founder, Celo

Smart Contracts: user-defined programs running on top of a blockchain

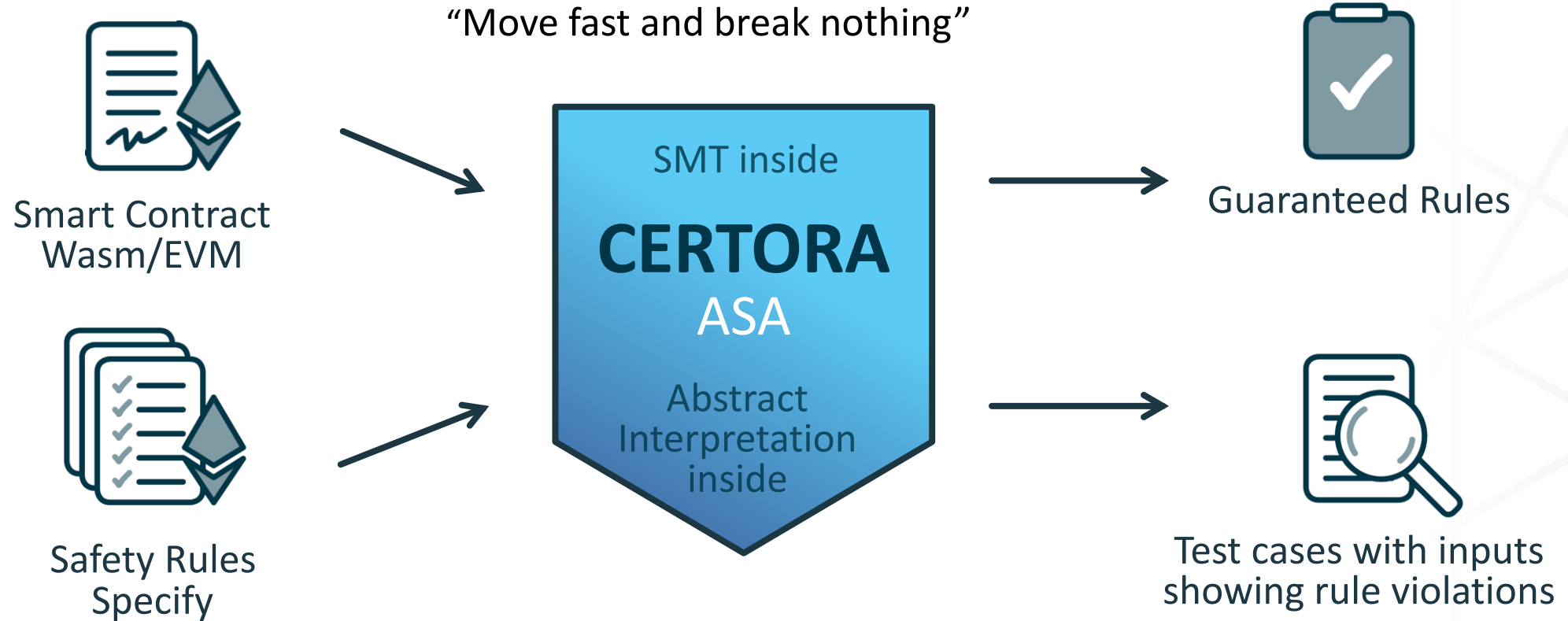


Smart Contract Example (very high level)



Other examples abound:
Auctions, elections, lotteries, escrow, ...

Formal Verification of Smart Contracts



Benefits

Superior Accuracy

Most accurate method to detect bugs

Minimal False bugs

All reported errors are real

0 Missed Bugs

All errors are eventually detected

Scalability

Every contract is verified separately

Buggy Code

```
pragma solidity ^0.4.0;
contract TicketBuy {

    uint64 public ticketPrice;
    uint16 public ticketsRemaining;

    constructor(uint16 _numTickets, uint64 _price) {
        ticketsRemaining = _numTickets;
        ticketPrice = _price;
    }

    function getBalance(address x) returns (uint256) {
        return x.balance;
    }

    function updatePrice(uint64 newPrice) {
        ticketPrice = newPrice;
    }

    function buyNewTicket() payable returns (uint16 ticketID){
        require (ticketsRemaining > 0);
        require(msg.value >= ticketPrice);

        ticketID = ticketsRemaining--;
        return ticketID;
    }
}
```

Buggy Code

```
pragma solidity ^0.4.0;
contract TicketBuy {

    uint64 public ticketPrice;
    uint16 public ticketsRemaining;

    constructor(uint16 _numTickets, uint64 _price) {
        ticketsRemaining = _numTickets;
        ticketPrice = _price;
    }

    function getBalance(address x) returns (uint256) {
        return x.balance;
    }

    function updatePrice(uint64 newPrice) {
        ticketPrice = newPrice;
    }

    function buyNewTicket() payable returns (uint16 ticketID){
        require (ticketsRemaining > 0);
        require(msg.value >= ticketPrice);

        ticketID = ticketsRemaining--;
        return ticketID;
    }
}
```

Fixed Code

```
pragma solidity ^0.4.0;
contract TicketBuy {

    uint64 public ticketPrice;
    uint16 public ticketsRemaining;

    constructor(uint16 _numTickets, uint64 _price) {
        ticketsRemaining = _numTickets;
        ticketPrice = _price;
    }

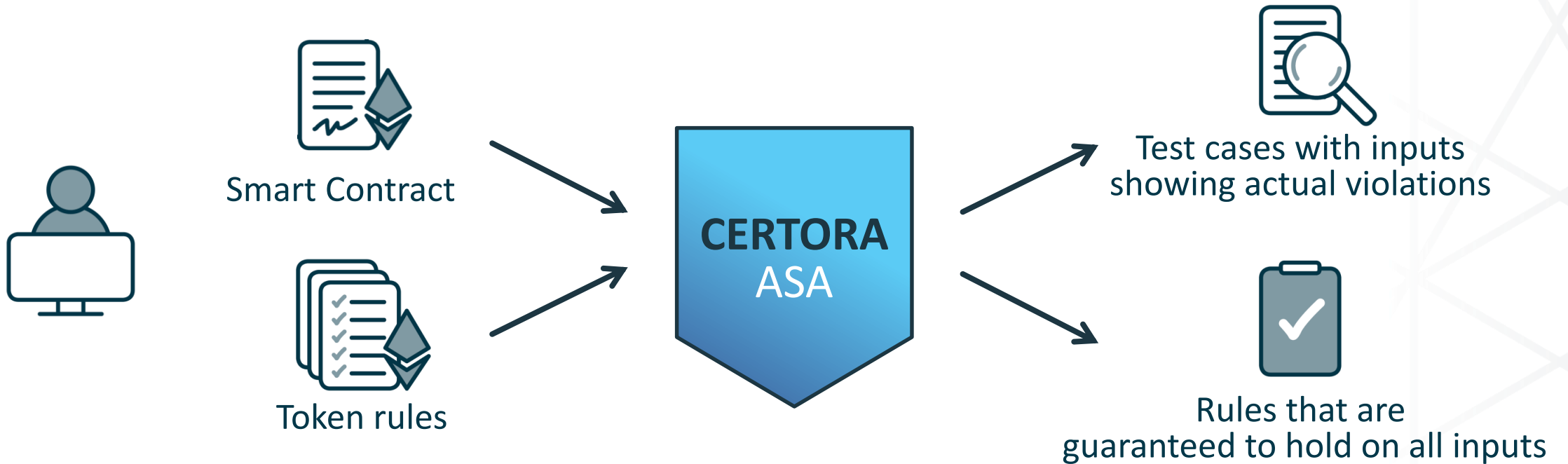
    function getBalance(address x) returns (uint256) {
        return x.balance;
    }

    function updatePrice(uint64 newPrice) {
        ticketPrice = newPrice;
    }

    function buyNewTicket() payable returns (uint16 ticketID){
        require (ticketsRemaining > 0);
        require(msg.value == ticketPrice);

        ticketID = ticketsRemaining--;
        return ticketID;
    }
}
```

Product: Asset Scanner



Token Scanning

Rules	Token 1	Token 2	Token 3	Token 4	Token 5
1	Green	Green	Green	Red	Green
2	Green	Green	Green	Red	Red
3 Burning: No one can burn arbitrary account's funds	Red	Green	Green	Red	Red
4	Green	Red	Green	Red	Green
5 Ownership transfer is restricted to privileged party	Green	Green	Green	Red	Green
6	Green	Green	Green	Green	Red
7 Cannot transfer without proper authorization	Green	Green	Green	Red	Green
8	Green	Green	Green	Green	Green
9	Green	Green	Green	Green	Green
10	Green	Green	Green	Green	Red
11	Green	Green	Green	Red	Green
12	Green	Green	Green	Green	Green
13	Green	Green	Green	Green	Green
14	Green	Green	Green	Green	Green
15	Red	Red	Red	Red	Red
16	Red	Green	Green	Red	Red
17	Red	Green	Green	Red	Red
18	Green	Green	Green	Red	Green
19 No external calls	Red	Red	Green	Red	Red

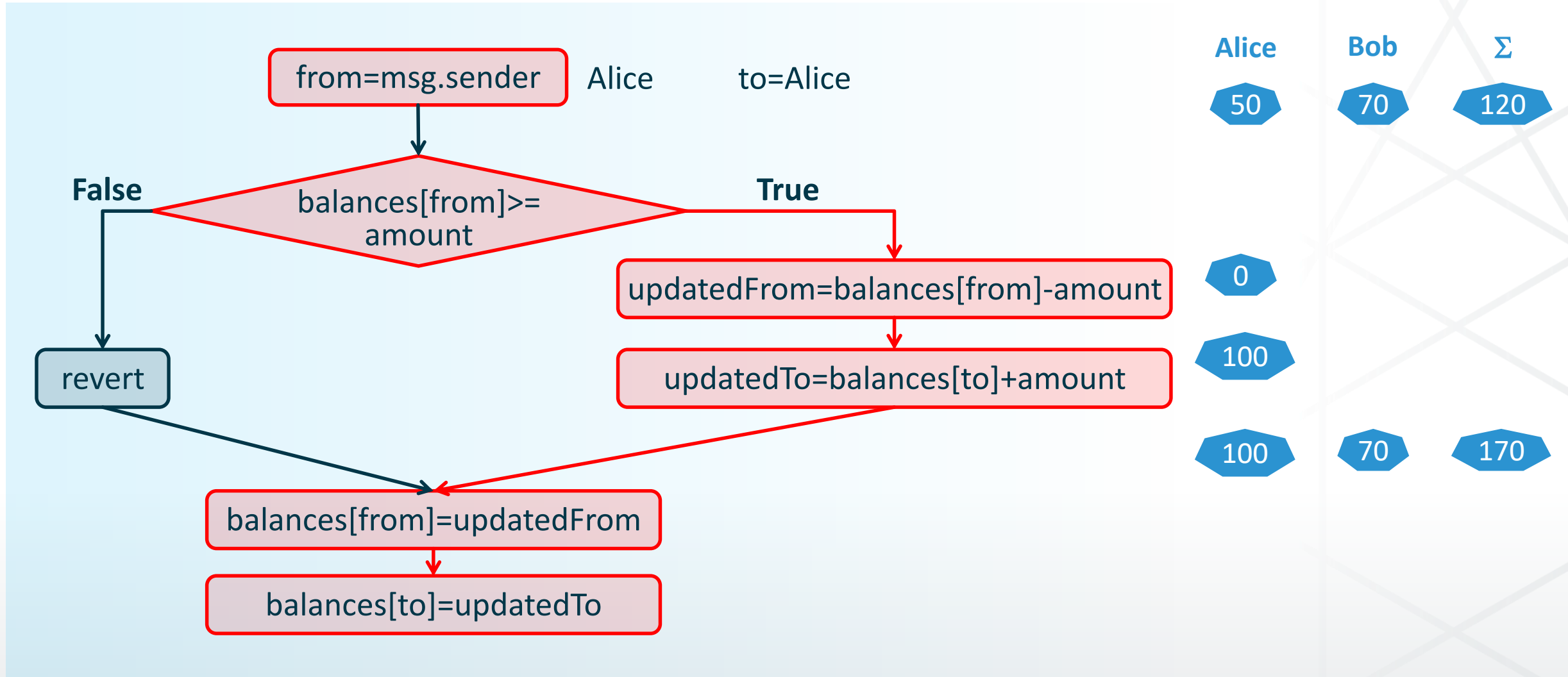
Toy ERC20 token

invariant $\sum_{a: \text{address}} \text{balances}[a]$

```
contract toyERC20 {
    mapping (address => uint) balances;
    constructor(address bank, uint initial_amount) {
        balances[bank] = initial_amount;
    }
    function transfer(address to, uint amount) {
        uint updatedFrom;
        uint updatedTo;
        address from = msg.sender;
        if (balances[from] >= amount) {
            updatedFrom = balances[from] - amount;
            updatedTo = balances[to] + amount;
        } else { revert(); }
        balances[from] = updatedFrom;
        balances[to] = updatedTo;
    }
}
```

Toy ERC20 token

invariant $\sum_{a: \text{address}} \text{balances}[a]$



Fixed Toy ERC20 token



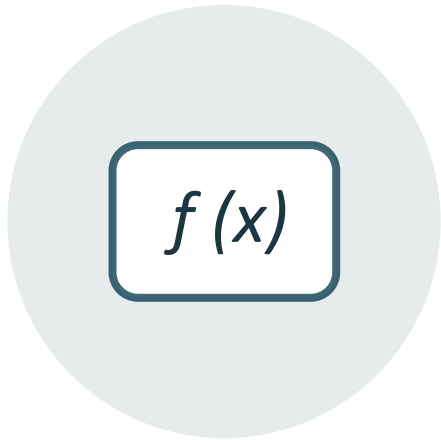
invariant $\sum_{a: \text{address}} \text{balances}[a]$

```
contract toyERC20 {
  mapping (address => uint) balances;
  constructor(address bank, uint initial_amount) {
    balances[bank] = initial_amount;
  }
  function transfer(address to, uint amount) {
    uint updatedFrom;
    uint updatedTo;
    address from = msg.sender;
    require from != to ;
    if (balances[from] >= amount) {
      updatedFrom = balances[from]-amount;
      updatedTo = balances[to] + amount;
    } else { revert(); }
    balances[from] = updatedFrom;
    balances[to] = updatedTo;
  }
}
```

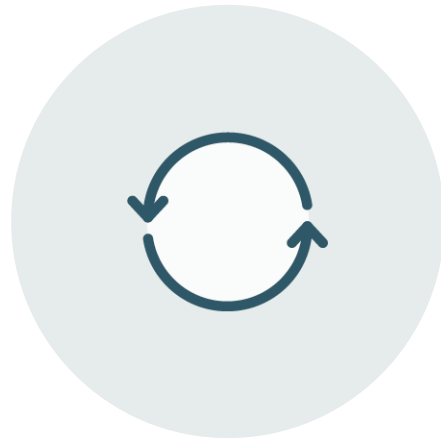
The Specification Mission

- **Develop a library of reusable correctness rules**
- **Community effort**
 - Reward developers for correctness rules
 - ~~Code is the law~~ → Spec is the law
 - No overdraft
 - If no transaction is executed then no cost
 - No radical currency changes

Formal Specification Challenges



Mathematical
thinking



Reuse



Correctness
of the specification



Tooling

Principles of Specify

- **Math is the law:** powerful expressions
 - $\sum x$
 - $\exists X.r(X)$ *FO^{Z,Agg}*
 - $\forall X$
 - $X \rightarrow Y$
- **Modularity**
 - The correctness of every contract is defined separately using global contract invariants (mostly)
 - Enabled by checking isolation between contracts which prevent bugs like DAO and others
- **Separate** code from the specification
- Check specifications



Dines Bjørner

Specification



Code

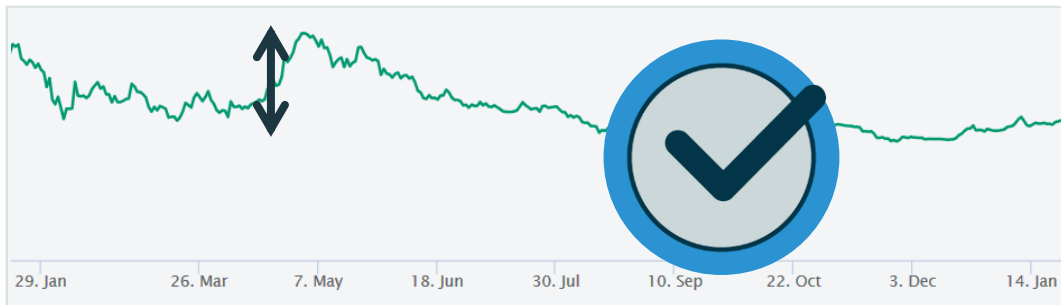
Principle 1: Math is the law



Geoff Hayes | CTO



- **Goal:** enable human mitigation of money theft
- **Requirement:** Price changes must be less than 10% every hour



For all t_1, t_2 . $|t_2 - t_1| < 1 \text{ hour}$, $|p_2 - p_1| < 0.1p_1$

```
(err, onePlusMaxSwing) = addExp(one, maxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// max = anchorPrice * (1 + maxSwing)
(err, max) = mulExp(anchorPrice, onePlusMaxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// If price > anchorPrice * (1 + maxSwing)
// Set price = anchorPrice * (1 + maxSwing)
if (greaterThanExp(price, max)) {
    return (Error.NO_ERROR, true, max);
}

(err, oneMinusMaxSwing) = subExp(one, maxSwing);
if (err != Error.NO_ERROR) {
    return (err, false, Exp({mantissa : 0}));
}

// min = anchorPrice * (1 - maxSwing)
(err, min) = mulExp(anchorPrice, oneMinusMaxSwing);
// We can't overflow here or we would have already overflowed
assert(err == Error.NO_ERROR);

// If price < anchorPrice * (1 - maxSwing)
// Set price = anchorPrice * (1 - maxSwing)
if (lessThanExp(price, min)) {
    return (Error.NO_ERROR, true, min);
}
```

Correctness rules for Debt

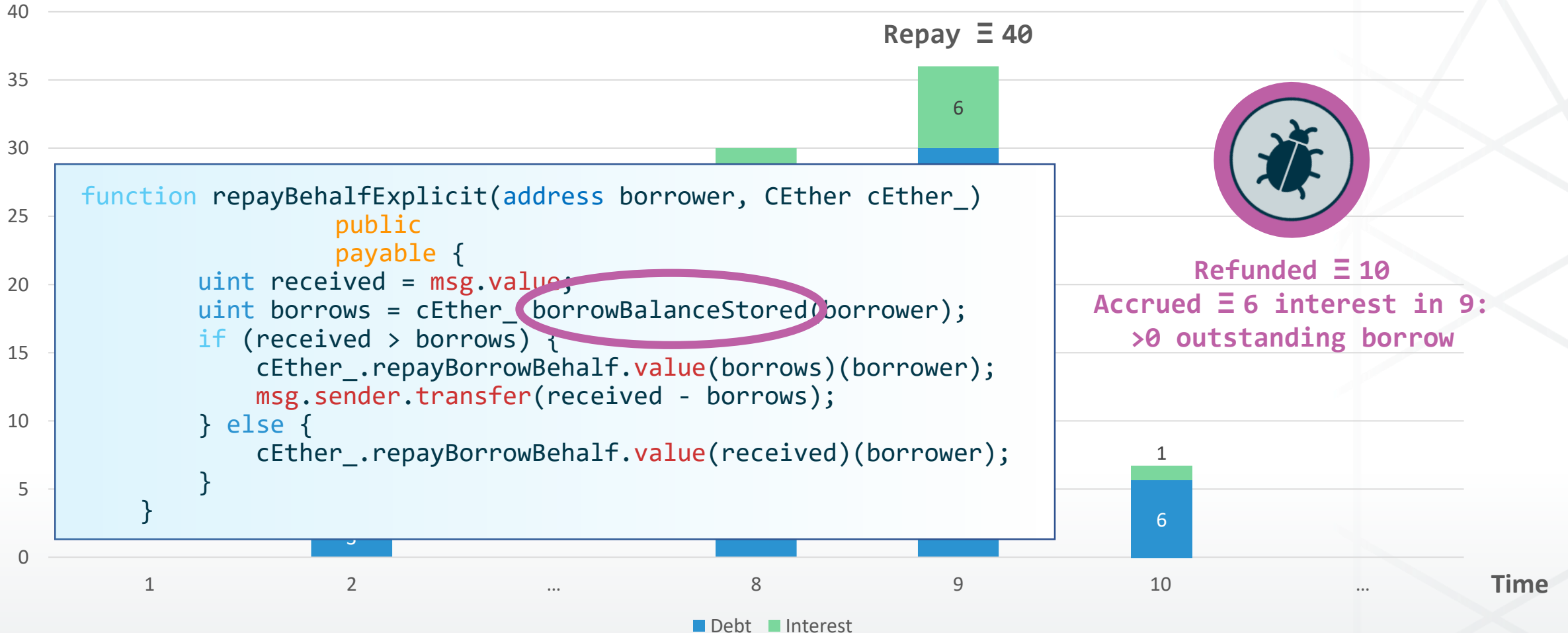
Any debt can be paid off: $\text{repayAmount} \geq \text{borrowed} \rightarrow \text{newBorrow} = 0$

```
function repayBehalfExplicit(address borrower, CEther cEther_)
    public
    payable {
    uint received = msg.value;
    uint borrows = cEther_.borrowBalanceStored(borrower);
    if (received > borrows) {
        cEther_.repayBorrowBehalf.value(borrows)(borrower);
        msg.sender.transfer(received - borrows);
    } else {
        cEther_.repayBorrowBehalf.value(received)(borrower);
    }
}
```

Correctness rules for Debt

Money

Any debt can be paid off: $\text{repayAmount} \geq \text{borrowed} \rightarrow \text{newBorrow} = 0$



Correctness rules for Debt

Any debt can be paid off: $\text{repayAmount} \geq \text{borrowed} \rightarrow \text{newBorrow} = 0$

```
function repayBehalfExplicit(address borrower, CEther cEther_)
    public
    payable {
    uint received = msg.value;
    uint borrows = cEther_.borrowBalanceCurrent(borrower);
    if (received > borrows) {
        cEther_.repayBorrowBehalf.value(borrows)(borrower);
        msg.sender.transfer(received - borrows);
    } else {
        cEther_.repayBorrowBehalf.value(received)(borrower);
    }
}
```

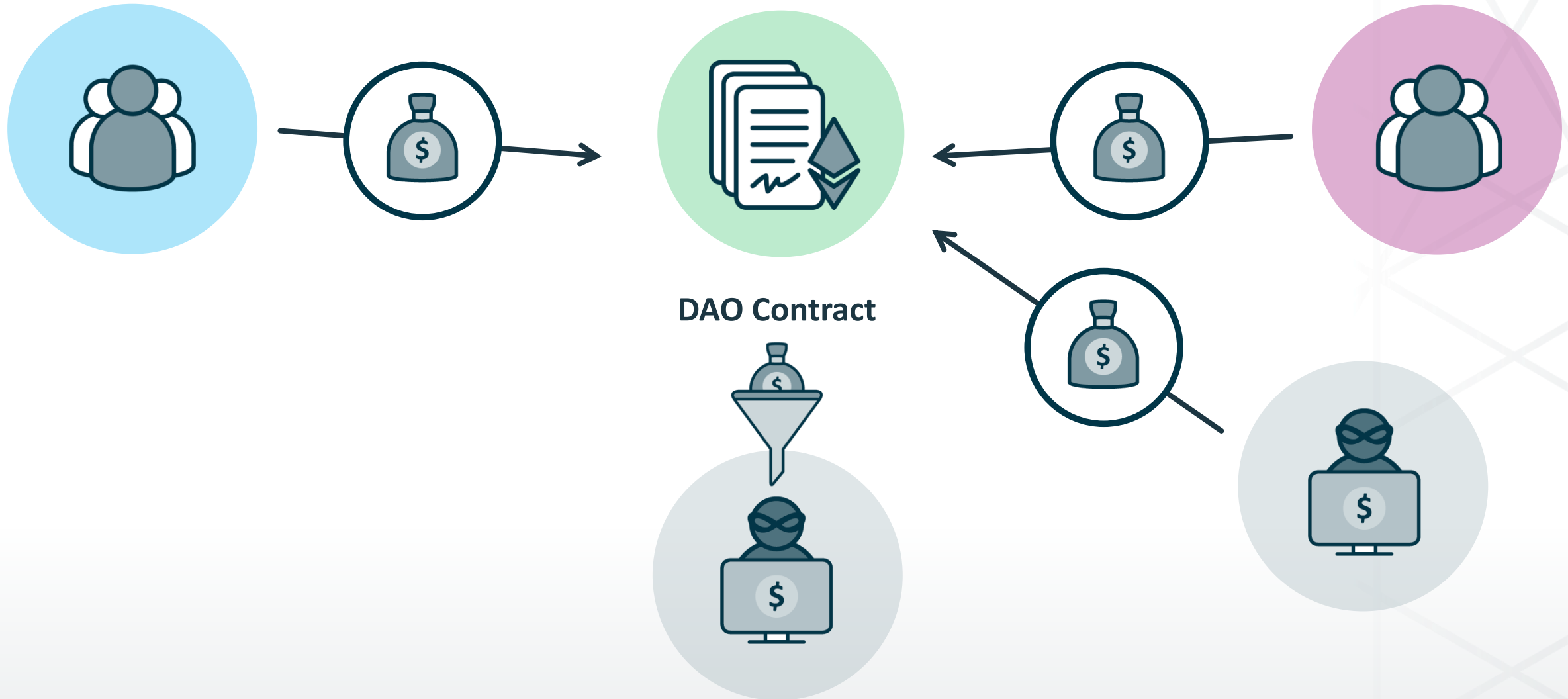


Principle 2: Modularity



- Enforces global contract invariants
- Checked modularly
 - Can this be always done?

Reentrancy attacks



Reentrancy attacks

```
DAO_withdraw(to) {  
  if (shares[to] > 0) {  
    to.send(shares[to]);  
    shares[to] = 0 ;  
  }  
}
```

```
f () {  
  DAO(x).withdraw(me)  
}
```



Immune Reentrancy attacks(Atomicity)

```
DAO_withdraw(to) {  
  if (shares[to] > 0) {  
    to.send(shares[to]);  
    shares[to] = 0 ;  
  }  
}
```

```
DAO_withdraw(to) {  
  if (shares[to] > 0) {  
    temp = shares[to];  
    shares[to] = 0 ;  
    to.send(temp);  
  }  
}
```

Atomicity[POPL'18]

- Contracts that are vulnerable to reentrancy attacks are dangerous to use
 - Sensitive to changes in the EVM
 - Constantinople fork postponement
- **Most precise method**
- **Enables modular reasoning**
- Guarantee atomicity in presence of callbacks

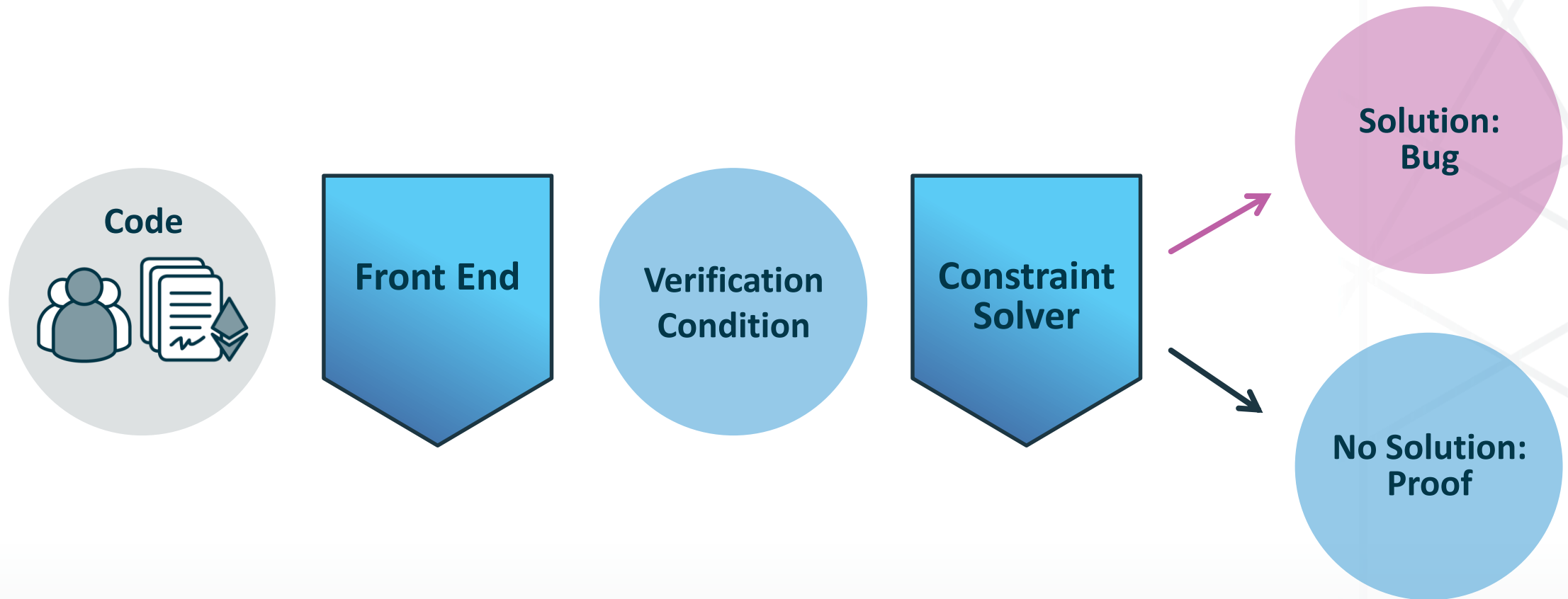


Principle 3: Separate Code from Spec

```
test checkTransferReversibility() {
  uint amount ; // arbitrary
  address to; // arbitrary
  transaction[t1] {transfer(to, amount)} ;
  transaction [t2] {transfer(t1.msg.sender, amount)}
    where t2.msg.sender == to ;
  assert  $\forall a:\text{address. balances}[a]== t1.\text{before}.\text{balances}[a]$ ;
}
```

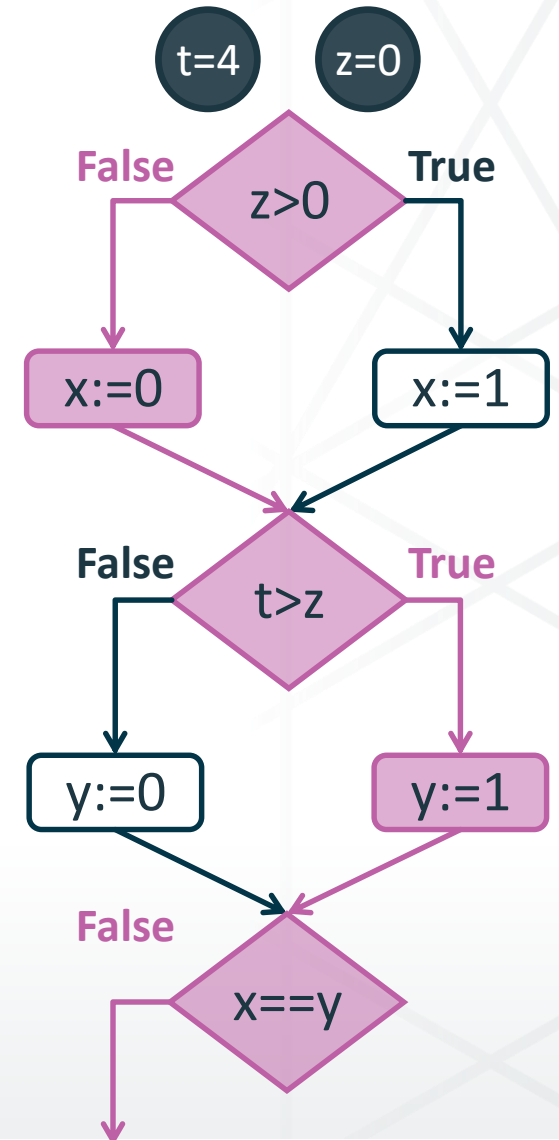
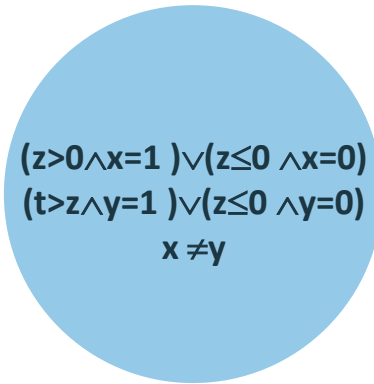
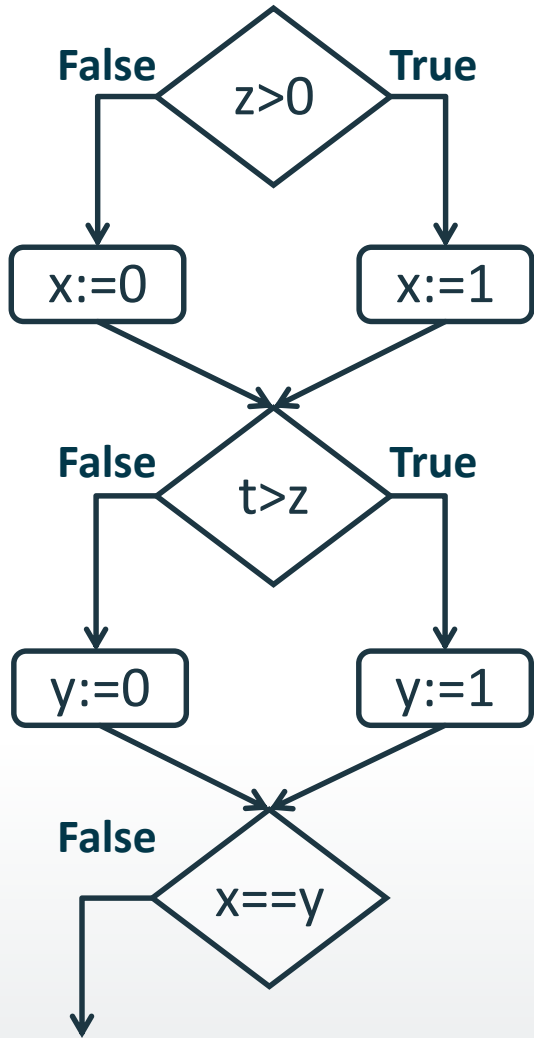
SPC

ASA via Constraint Solving

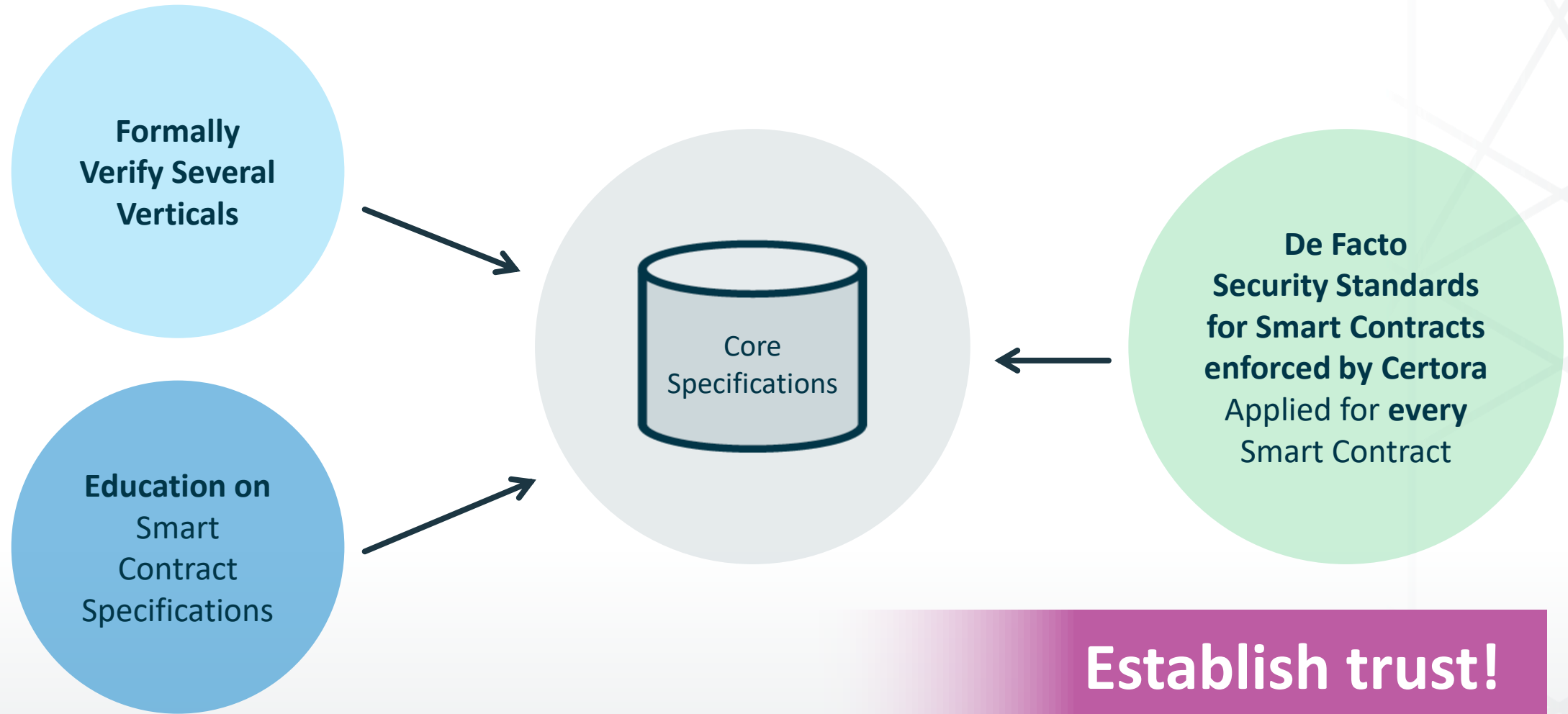


[Z3] Microsoft Research, [CVC4] Stanford University, [Yices] Stanford Research Institute [SMT*](#)

ASA via Constraint Solving



Towards Trusted Blockchain



Summary

- Smart contracts are interesting for formal verification
 - Immutable
 - Value
 - Open environment
 - Non determinism is a blessing
- Reasoning about low level programs
 - Array axioms are hard
 - Connection with cryptographic primitives
- Develop reusable specifications is an ongoing challenge

Alien Abduction

By Bart Laubsch, DE
In the [total randomness Collection](#)

 Creative Commons

spiral

By Hawraa Alsalman, GB

 Creative Commons

spam email

By ProSymbols, US
In the [Cyber Robbery, Hacker Line Icons Collection](#)

 Creative Commons

money savings

By ProSymbols, US
In the [Cyber Robbery, Hacker Line Icons Collection](#)

 Creative Commons

Description	English: This diagram describes Continuous Delivery process
Date	4 October 2015
Source	This file was derived from: Continuous Delivery process diagram.png
Author	Grégoire Détrez, original by Jez Humble