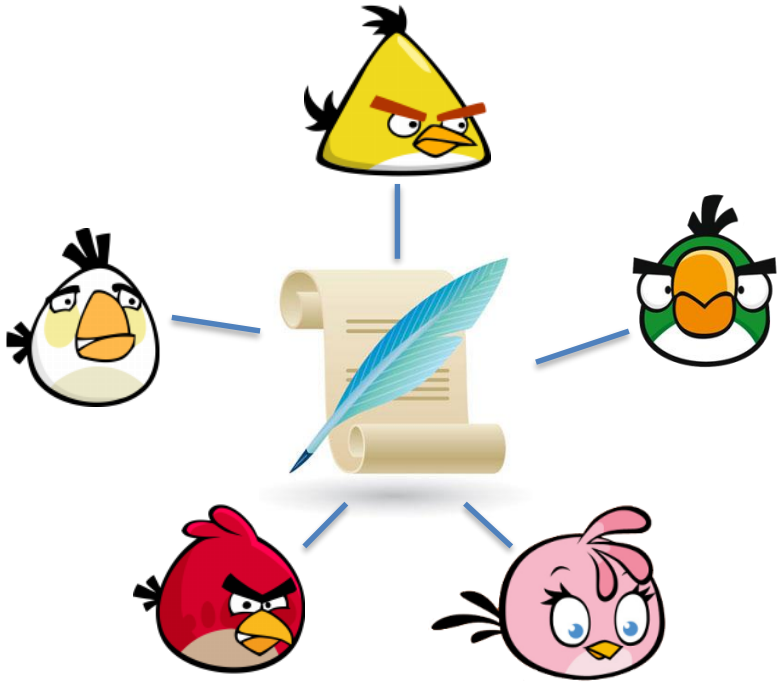
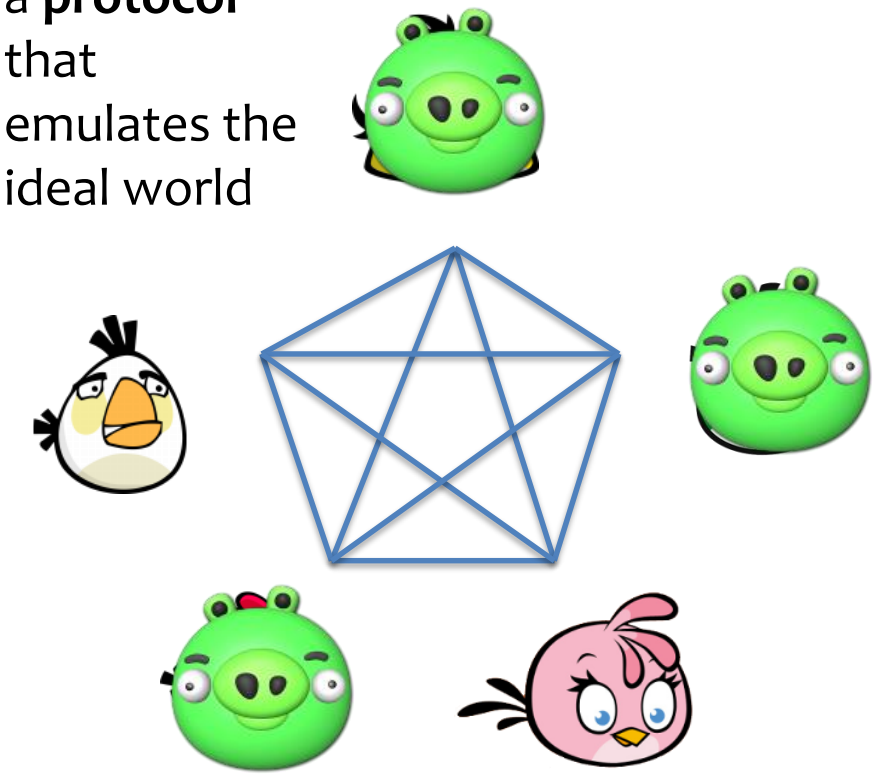
Stefan
Dziembowski

Trusted bulletin-board emulation

the "ideal" world



a **protocol**
that
emulates the
ideal world



Main difficulty: Some parties can cheat.

Classical result: simulation is possible if the "majority is honest".

For example for **5** players we can tolerate at most **2** "cheaters".

Formally Verifying Smart Contracts

Mooly Sagiv

Tel Aviv University

And also…

Shelly Grossman

Ittai Abraham

Shachar Itzhaky

David Dill

Yan Michalevsky

Noam Rinetzky

Guy Golan-Gueta

Yoni Zohar

Smart Contracts

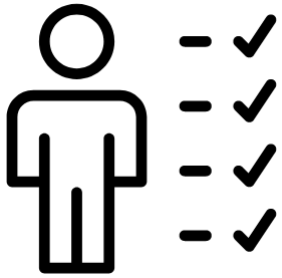
- Transactions in bitcoin are limited
  - Transfer ‘X’ bitcoins from ‘Y’ to ‘Z’
- More powerful transactions
  - Exchange
  - Auction
  - Games
  - Bets
  - Legal agreements
- Solution
  - Store smart contracts on the blockchain
  - Computer programs implement transactions
  - Immutability guarantees persistence

THE PROBLEM

Massive Losses due to Bugs

CURRENT SOLUTIONS

Auditing



Testing

Manual

Costly

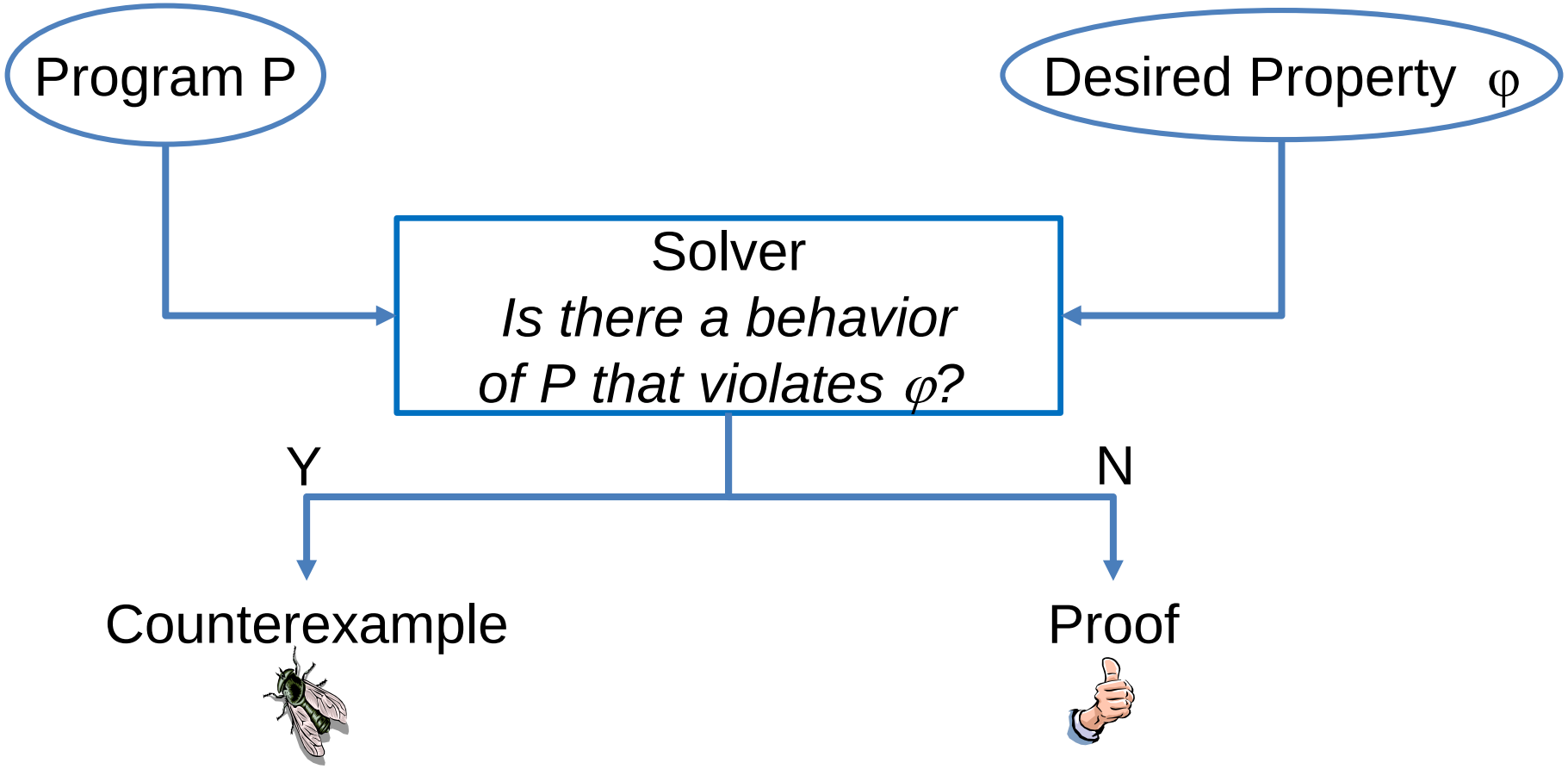
Incomplete

AUDITING IS INSUFFICIENT

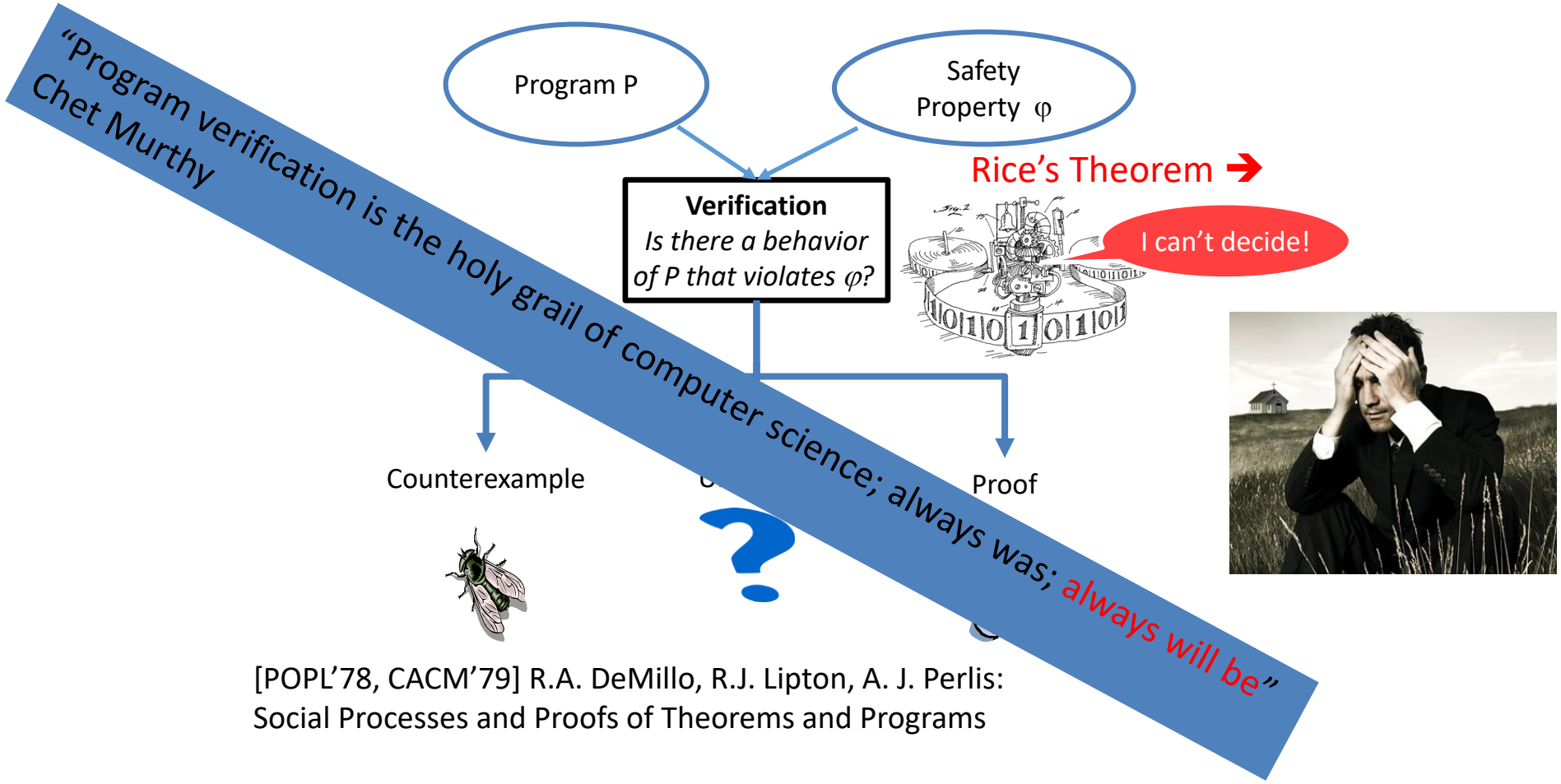
From "A Postmortem on the Parity Multi-Sig Library Self-Destruct":

… multi-sig wallet code was created and audited by the Ethereum Foundation's DEV team, Parity technology and others in the community

Automatic software verification



Disillusionment in program verification 80's



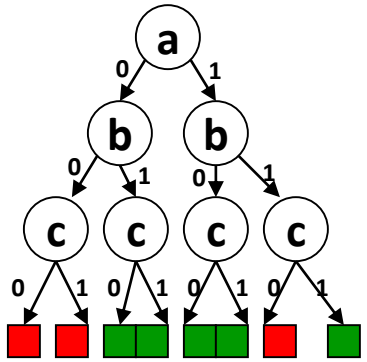
[POPL'78, CACM'79] R.A. DeMillo, R.J. Lipton, A. J. Perlis: Social Processes and Proofs of Theorems and Programs

Challenges in program verification

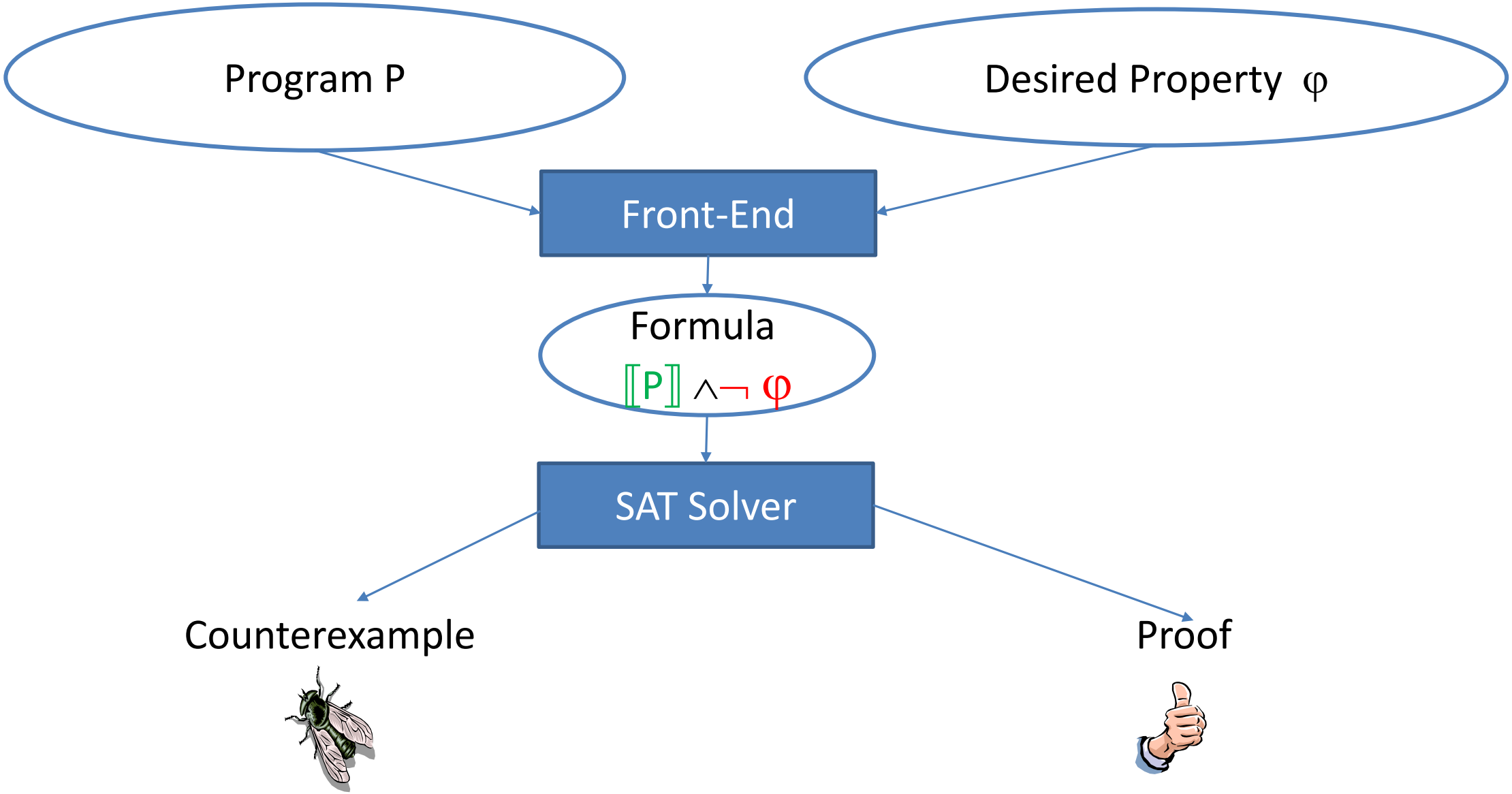
- Specifying program behavior
- Complexity of program verification
  - The halting problem
  - Rice theorem
  - The ability of simple programs to represent complex behaviors
- Complexity of realistic systems
  - Huge code
  - Heterogeneous code
  - Missing code

The SAT Problem

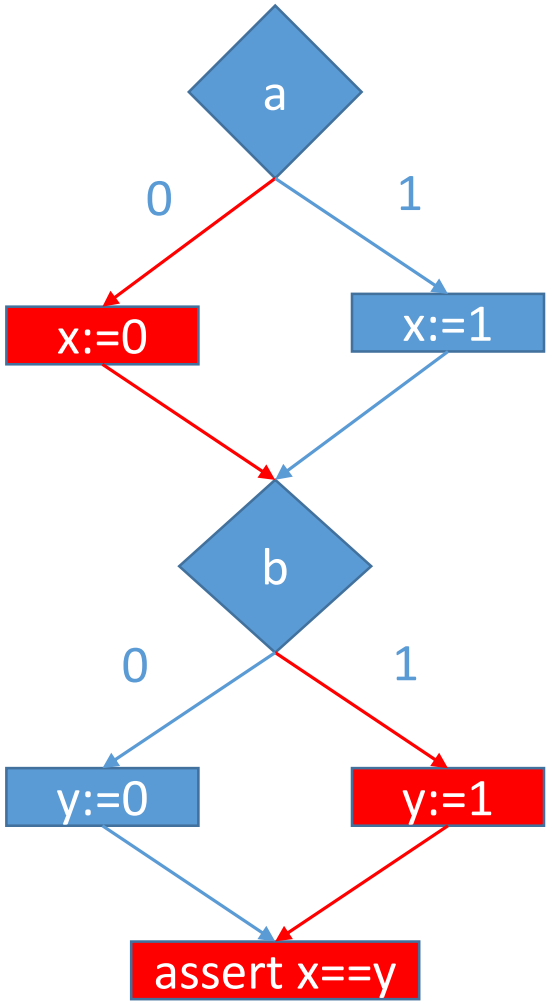
- Given a propositional formula (Boolean function)
  - φ = **(a** $\vee$ **b)** $\wedge$**(**$\neg$ **a**$\vee$ $\neg$**b** $\vee$ **c)**
- Determine if φ is satisfiable
  - Find a satisfying assignment or report that such does not exist
- For *n* variables, there are 2^n possible truth assignments to be checked
- Tools exist: Z3, Yices, CVC, …



Verification by reductions to SAT



Verification by reduction to SAT



SAT Query:

$((a \wedge x) \vee (\neg a \wedge \neg x))$

$\wedge$

$((b \wedge y) \vee (\neg b \wedge \neg y))$

$\wedge$

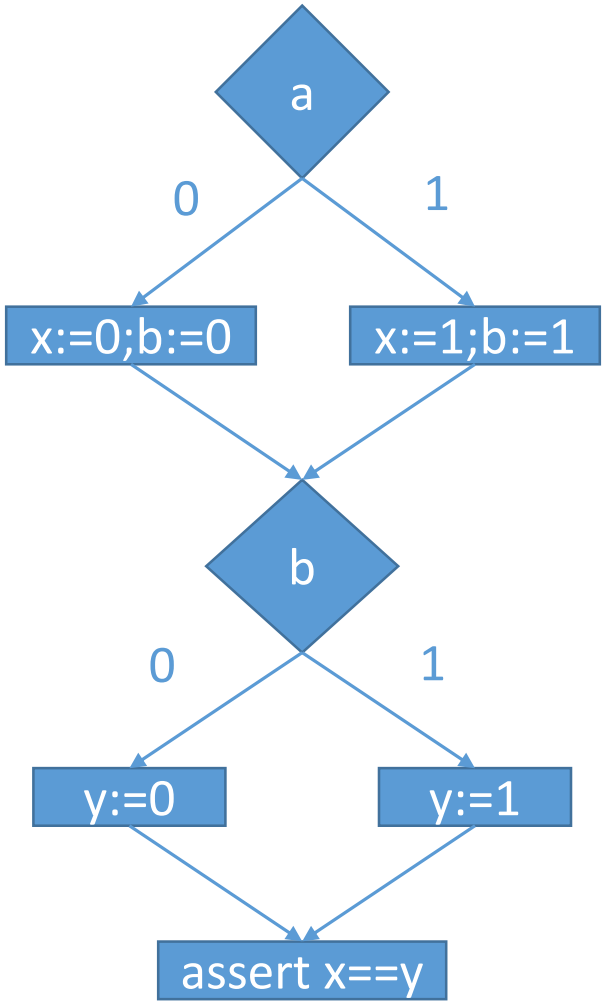
$((x \wedge \neg y) \vee (\neg x \wedge y)$

?

SAT Answer:

Satisfiable by a=0, b = 1

Verification by reduction to SAT



SAT Query:

$((a \wedge x \wedge b) \vee (\neg a \wedge \neg x \wedge \neg b))$

$\wedge$

$((b \wedge y) \vee (\neg b \wedge \neg y))$

$\wedge$

$((x \wedge \neg y) \vee (\neg x \wedge y)$

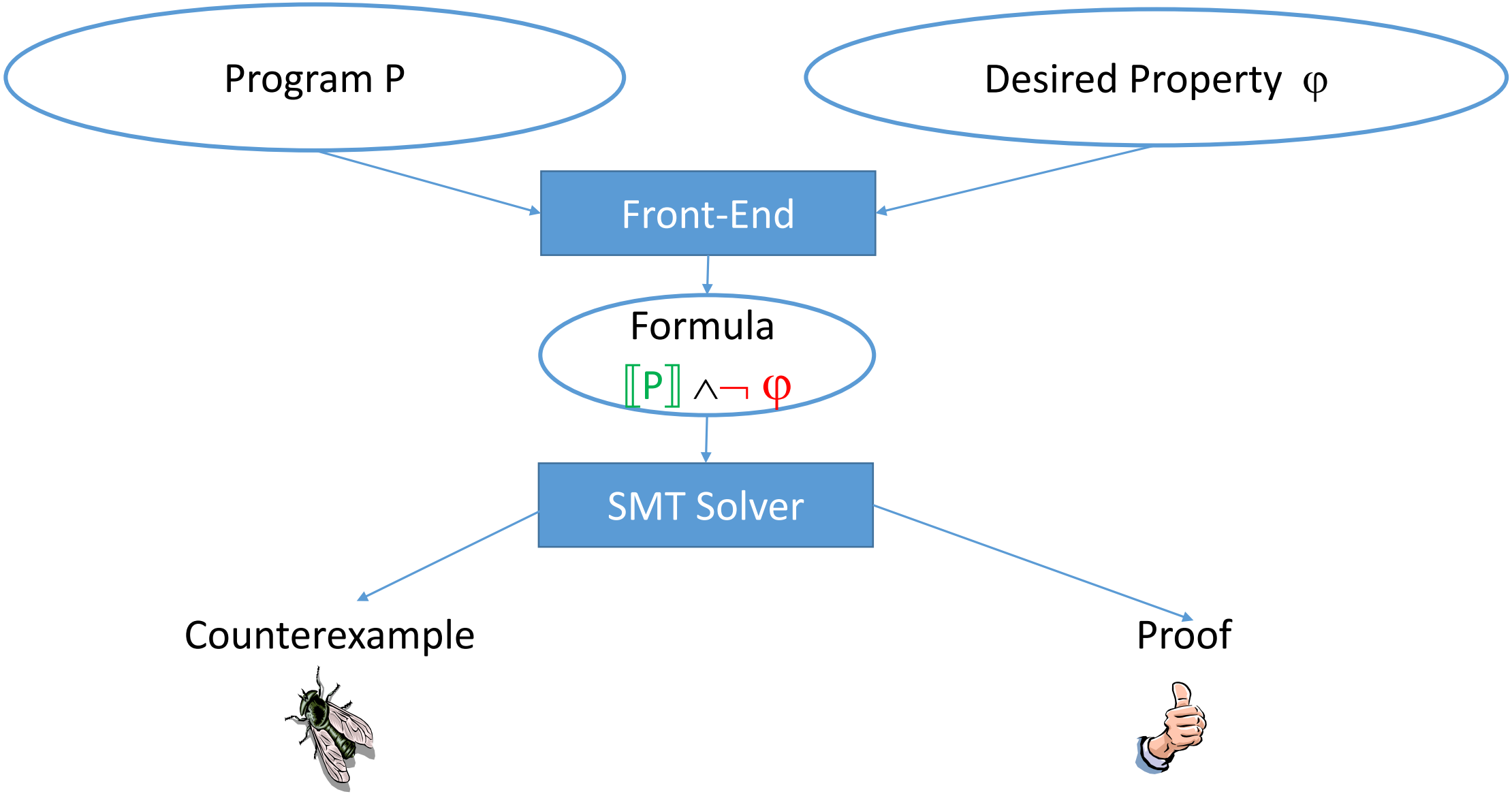
?

SAT Answer:
Unsatisfiable

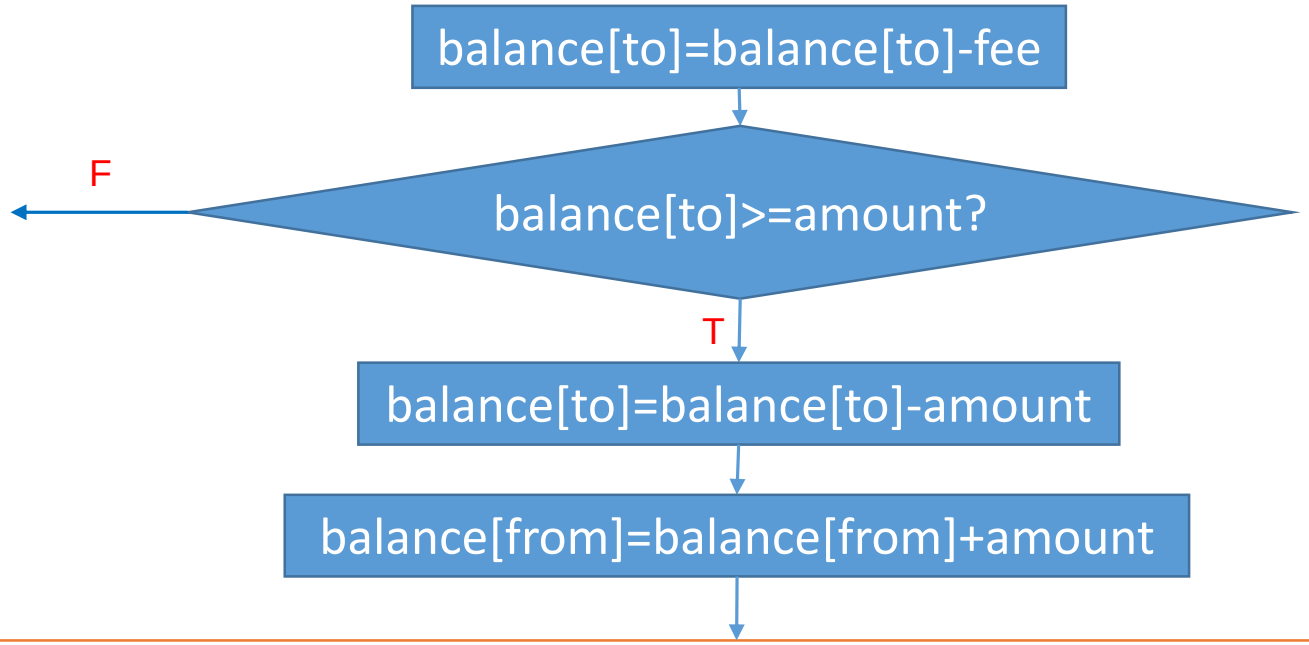
The SMT(Sat Modulo Theory) Problem

- Given a ground first order formula over theories(Boolean function)
  - $\varphi = \exists x, y: 2x + y \geq 5 \wedge y < 3$
- Determine if φ is satisfiable
  - Find a satisfying assignment or report that such does not exist
- Satisfiability becomes harder
- But tools exist: Yices, Z3, CVC, …

Verification by reductions to SMT



Simple Example Token (buggy)

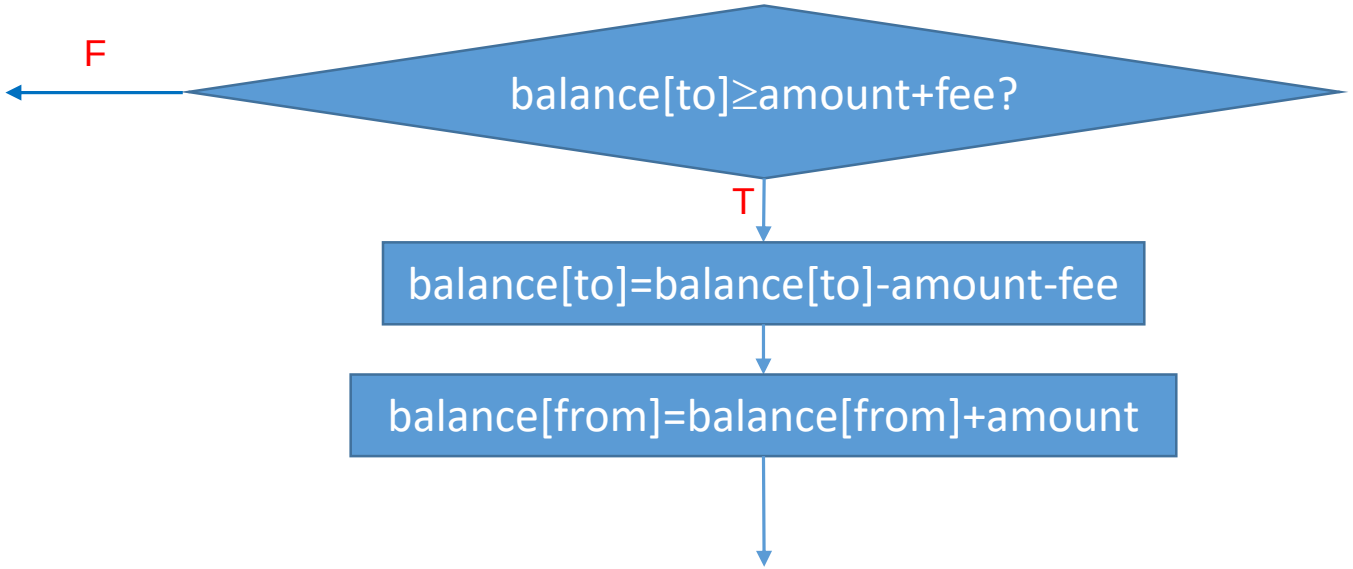


SMT Answer:
Satisfiable by
balance[to]=10,
fee=5,
amount=6
balance[from]=100

assert

$$\forall x.\ (x \neq to \wedge x \neq from) \Rightarrow b'[x]=b[x] \wedge$$
$$b[to] \geq amount+fee \Rightarrow (b'[to]=b[to]-amount-fee \wedge b'[from]=b[from]+amount) \wedge$$
$$b[to]<amount+fee \Rightarrow (b'[to]=b[to] \wedge b'[from]=b[from])$$

Simple Example Token (corrected)



SAT Answer:
Unsatisfiable

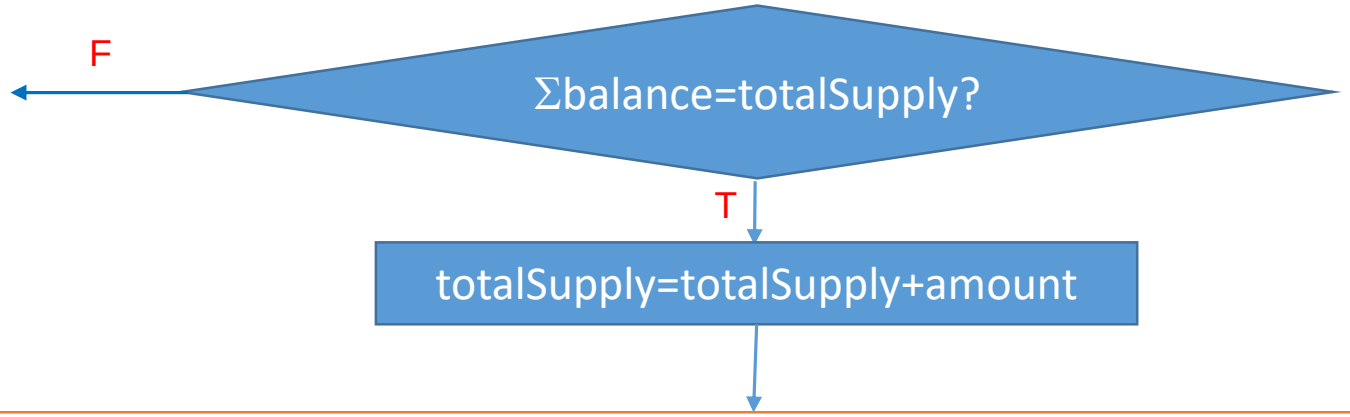
assert

$$\forall x.\ (x \neq to \wedge x \neq from) \Rightarrow b'[x]=b[x] \wedge$$
$$b[to] \geq amount+fee \Rightarrow (b'[to]=b[to]-amount-fee \wedge b'[from]=b[from]+amount) \wedge$$
$$b[to]<amount+fee \Rightarrow (b'[to]=b[to] \wedge b'[from]=b[from])$$

More interesting contracts

- Unbounded participants
- Complicated specifications
  - Higher order reasoning
- Need to handle loops

Minting Tokens - buggy

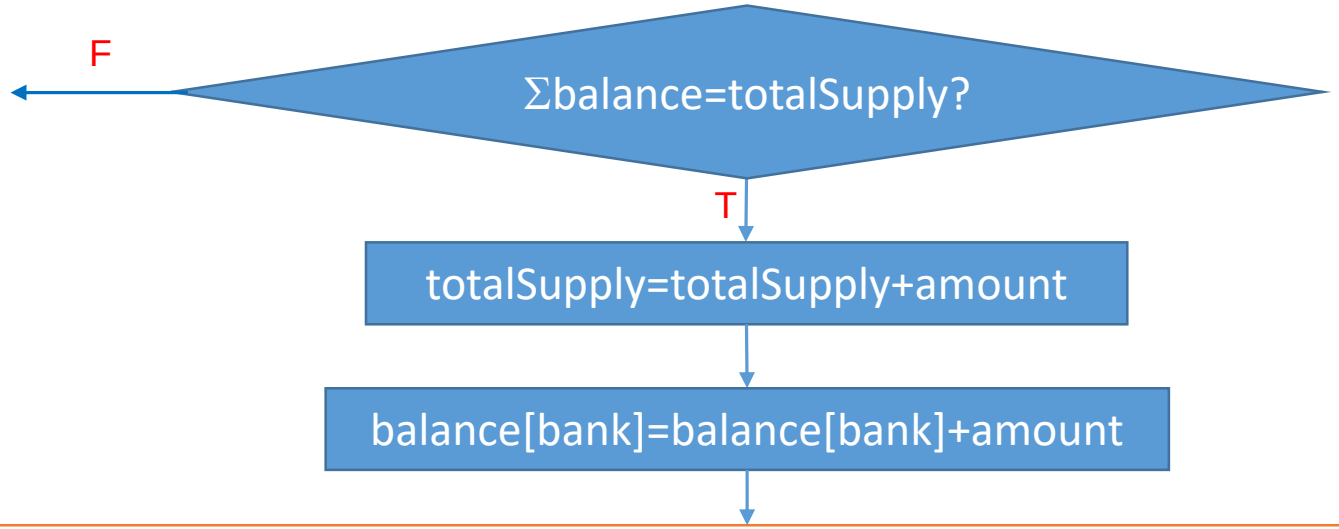


SAT Answer:
Satisfiable by
Σbalance=10,
totalSupply=10,
amount=5

assert

Σbalance’=totalSupply’ ∧ totalSupply’=totalSupply+amount

Minting Tokens - corrected



SAT Answer:
Unsatisfiable

assert

$\Sigma balance' = totalSupply' \wedge totalSupply' = totalSupply + amount$

Challenge: Handling Loops

- Bounded loop instantiation
  - CBMC
  - Scaling
- User specified loop invariants
  - Powerful
  - But requires careful insights
- Automatic loop invariants inference
  - Ultimately limited
  - Even when checking is possible
- Limited loops

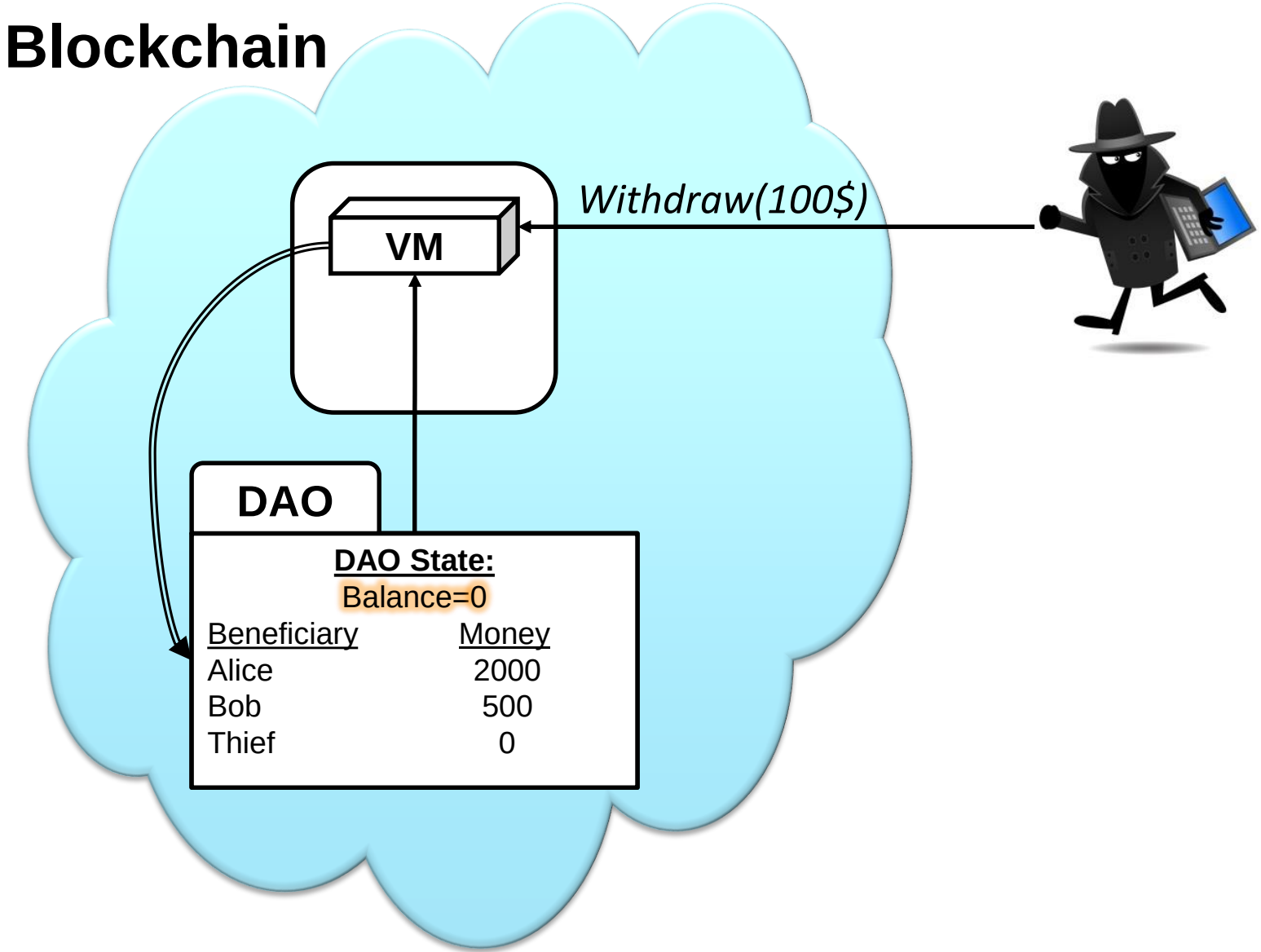
Summary thus far

- Program verification is powerful
- But hard to apply to complicated systems
- Modularity helps

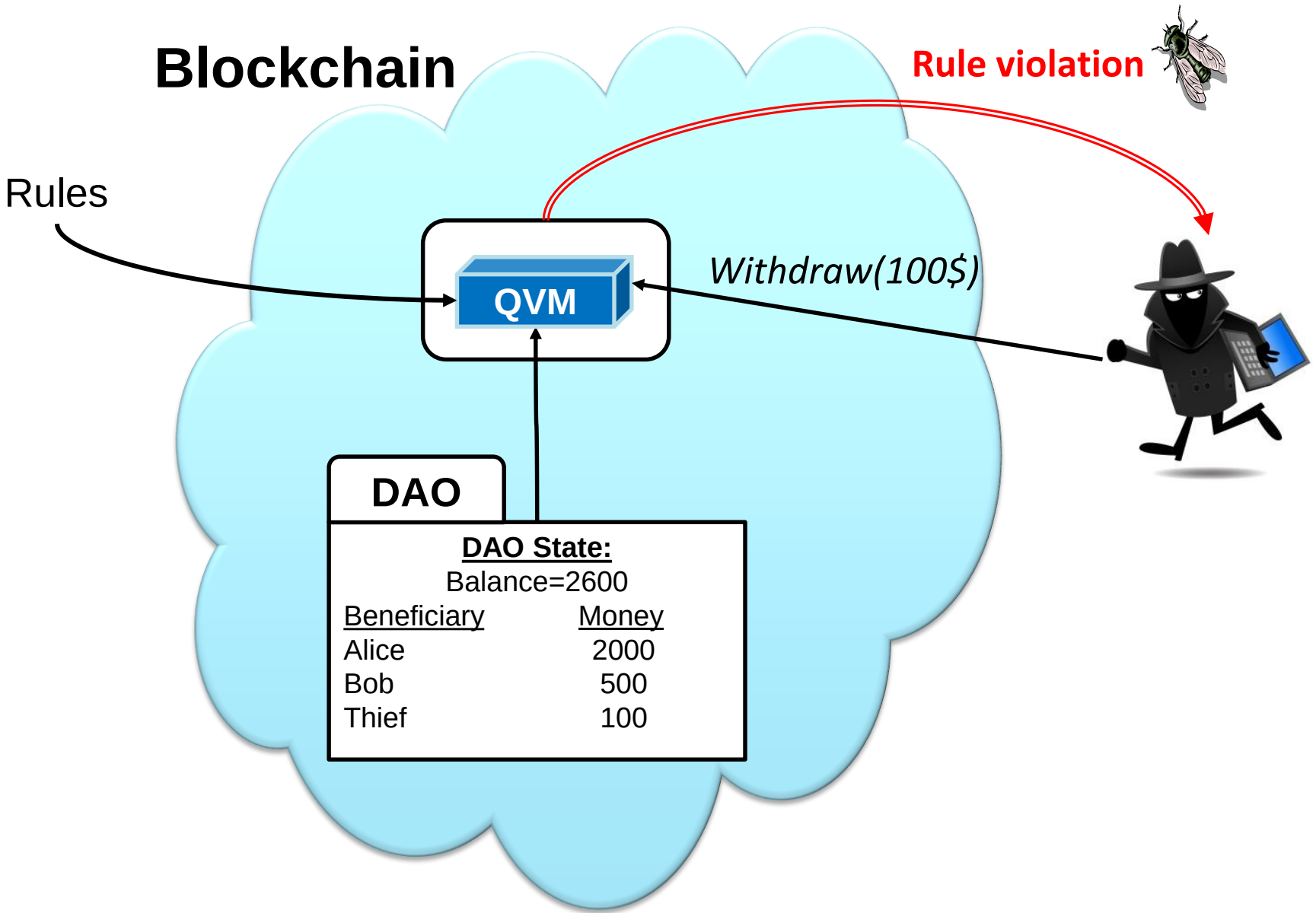
Runtime Monitoring

- Enforce correctness at runtime
- Especially useful with generic required properties
- Java properties
  - No out of bound array accesses
  - No null dereferences
  - ….
- Can we do the same for contracts?

MOTIVATION: EXISTING VM



Quality VM


Blockchain
Rule violation
Rules
QVM
Withdraw(100$)
DAO
DAO State:
Balance=2600
Beneficiary
Money
Alice
2000
Bob
500
Thief
100

Some Generic Correctness Rules

- Effectively Callback Free (ECF) transactions
  - Eliminate the DAO bug
- Immutable Ownership
  - Parity #1
- Prevent Bad upgrades
  - Monitor code changes and signed whitelists
  - Parity #2
- A flexible framework for arbitrary rules

Effective Callback Freedom – the DAO bug

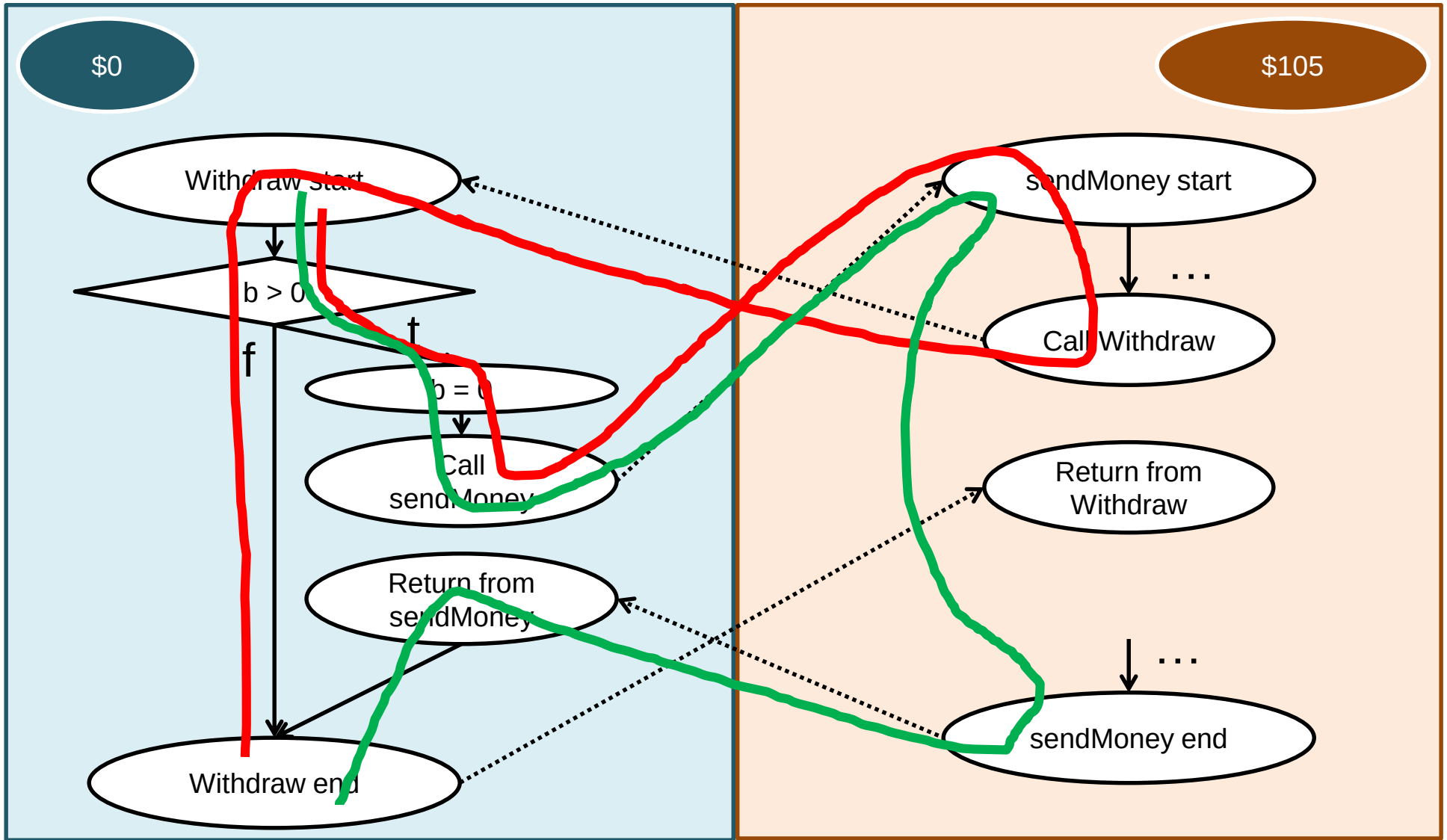
```
DAO::withdraw(to) {
 if b[to] > 0 {
  sendMoney(to, b[to]);
  b[to] = 0;
 }
}

Thief::uponTransfer(a) {
 DAO::withdraw(Thief)
}
```

coins[Thief]=205

b[Thief]=100

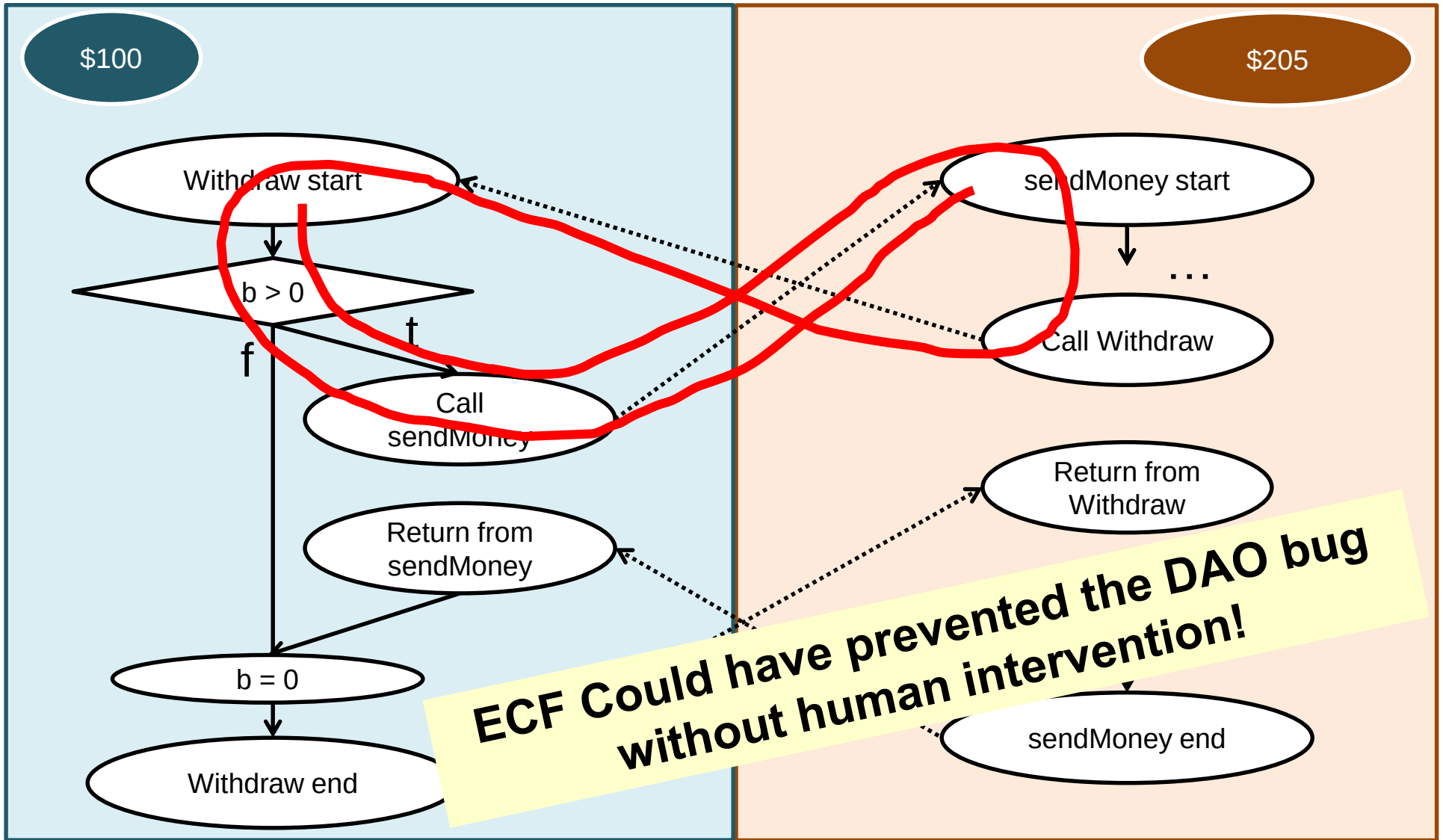
EFFECTIVE CALLBACK FREEDOM (ECF)

For every path there is a path without callbacks with same effect



GIST OF DAO ATTACK

~~For every path there is a path without callbacks with same effect~~



Empirical Results (POPL'18)

Ethereum (7/2015 — 6/2017)

Blockchain	Contracts	Executions	Non-ECF (%)
Ethereum	342K	96M	3,321 (0.003%)
Ethereum Classic	91K	32M	2,288 (0.007%)

Each Non-ECF is an actual attack (0% False positive)

Miniscule performance overhead*

Could have prevented the DAO bug without human intervention!

*3.38% in time executing EVM alone – drops further in real settings

THE THREE ENABLERS

- Relatively small number of generic required properties are needed
    - Not per-contract
- Restricted domain
    - Small contracts
    - Modularity due to ECF
- Feasibility of defensive checking

Contract Verification != Software Verification

Complementary Approaches

- Concolic execution
- Restricted programs
-

Summary

- Virtualization is powerful
- Program verification is powerful
- Program verification is expensive
- Few Success stories
  - Hardware Verification
  - Operating System
  - Device drivers
  - Packet Filters
  - Distributed protocols
- Contract verification
- Higher order programming reduces errors and enables verification