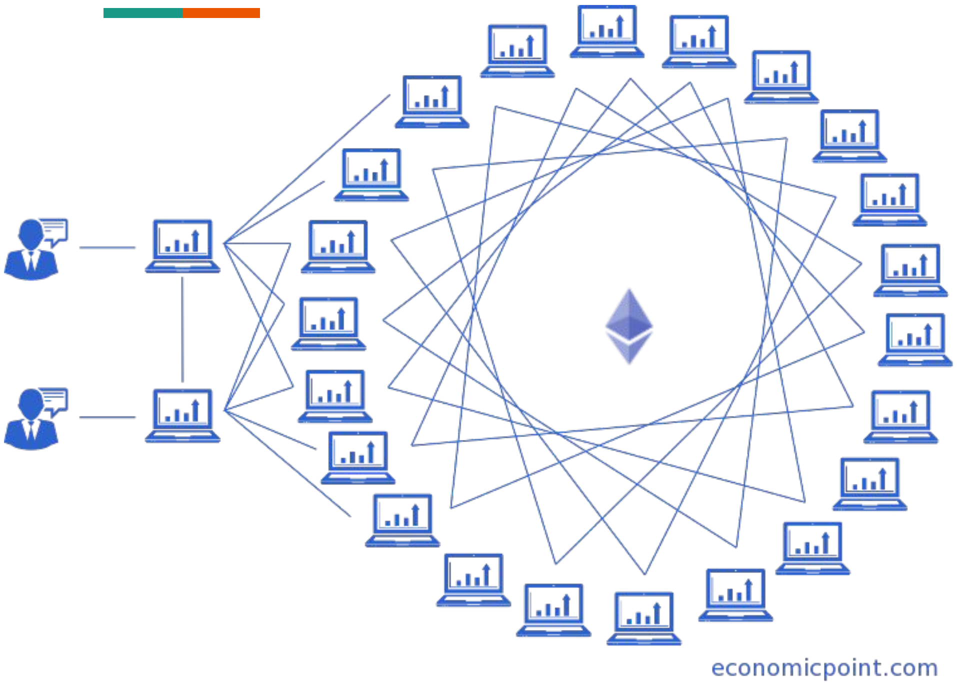
Contracts over Smart Contracts: Recovering from Violations Dynamically

Joshua Ellul
with: Christian Colombo, Gordon Pace

L-Università ta' Malta

Ethereum Blockchain Platform



Anyone can run a node (full node, or other)

Each node stores the Ethereum Ledger

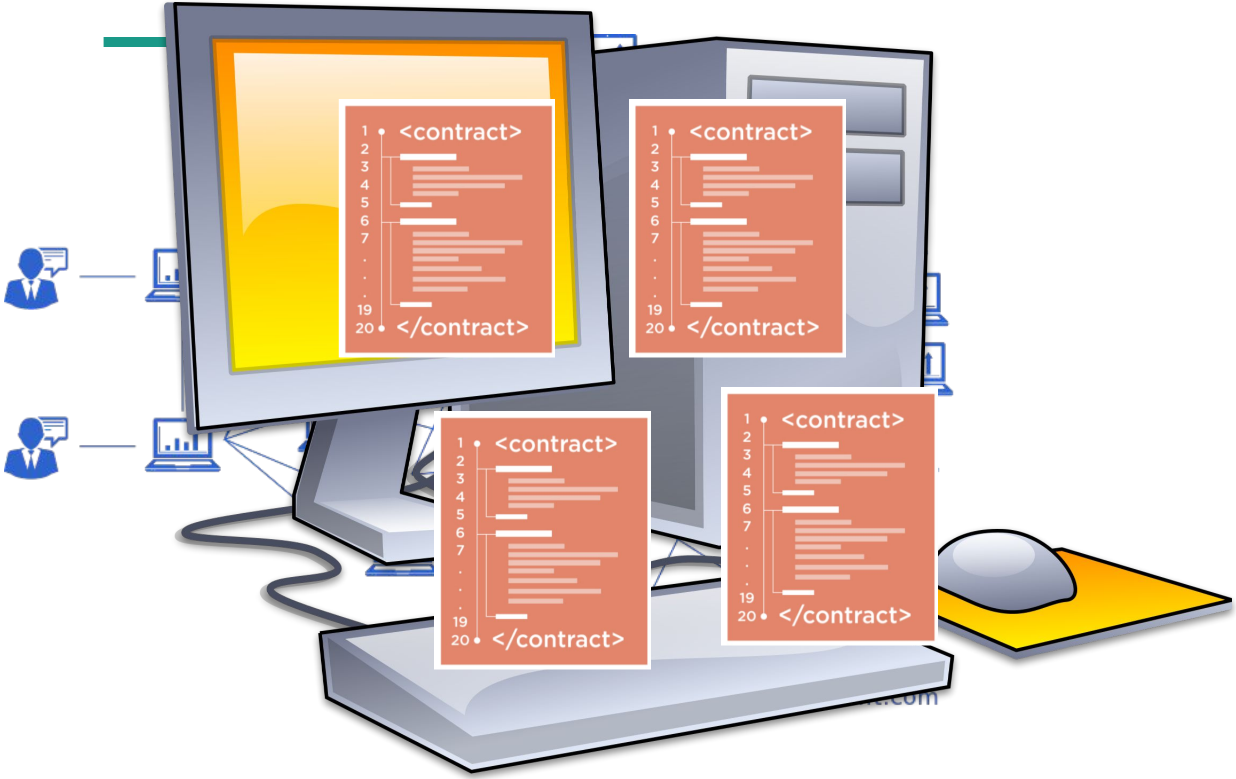
Consensus: Proof of Work

https://economicpoint.com/ethereum

Ethereum Blockchain Platform

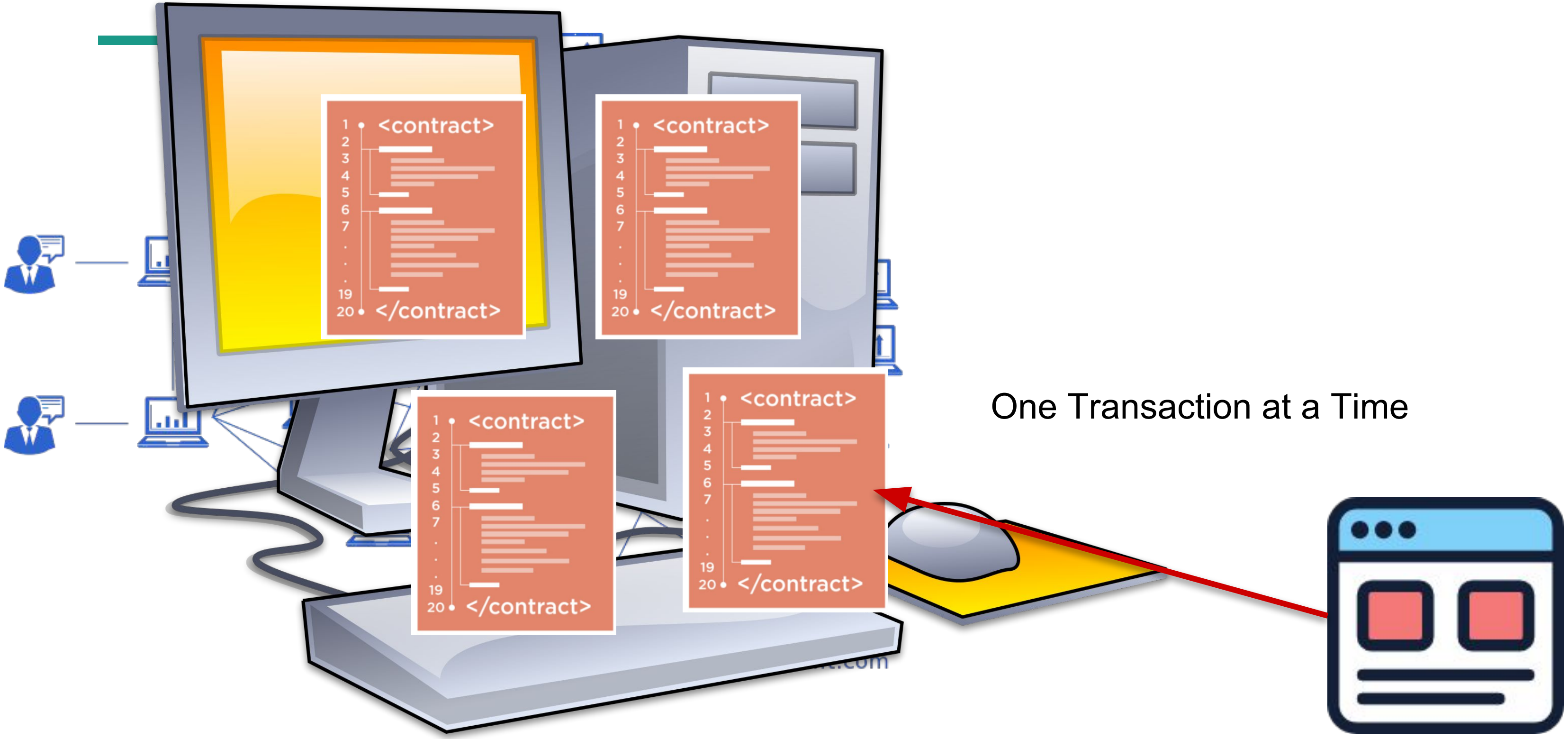
https://economicpoint.com/ethereum

Ethereum Blockchain Platform



https://economicpoint.com/ethereum

Ethereum Blockchain Platform



One Transaction at a Time

https://economicpoint.com/ethereum

Smart Contracts

Blockchain and Smart Contracts, enable:

- Decentralised, verifiable, enforceable automation of digital processes

Smart Contracts

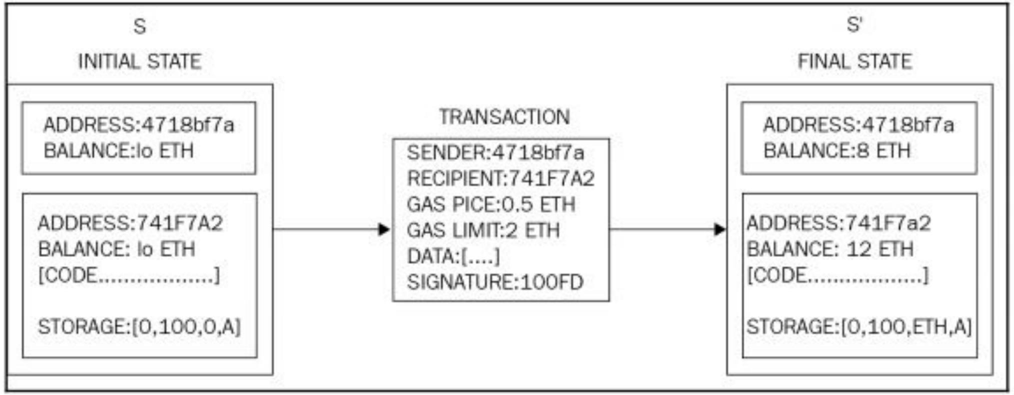
Blockchain and Smart Contracts, enable:
- Decentralised, verifiable, enforceable automation of digital processes

Different to contracts:
- obligations vs automated execution of obligations

Smart Contracts

One transaction at a time:

- Initial state + new Transaction (sender, receiver, data) => Final State



S

INITIAL STATE

ADDRESS:4718bf7a
BALANCE:lo ETH

ADDRESS:741F7A2
BALANCE: lo ETH
[CODE...................]

STORAGE:[0,100,0,A]

TRANSACTION

SENDER:4718bf7a
RECIPIENT:741F7A2
GAS PICE:0.5 ETH
GAS LIMIT:2 ETH
DATA:[....]
SIGNATURE:100FD

S'

FINAL STATE

ADDRESS:4718bf7a
BALANCE:8 ETH

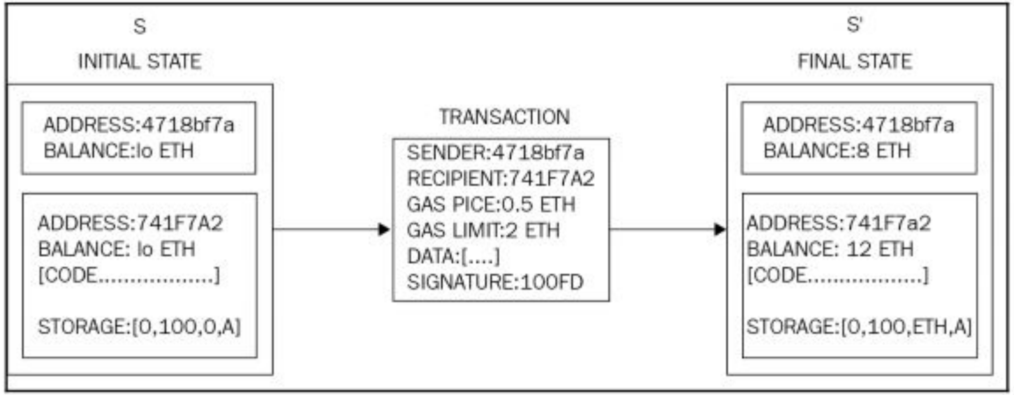
ADDRESS:741F7a2
BALANCE: 12 ETH
[CODE...................]

STORAGE:[0,100,ETH,A]

Smart Contracts

One transaction at a time:
- Initial state + new Transaction (sender, receiver, data) => Final State

Simple -- false sense of security?
Smart contract code uploaded is immutable



S
INITIAL STATE

ADDRESS:4718bf7a
BALANCE:lo ETH

ADDRESS:741F7A2
BALANCE: lo ETH
[CODE...................]

STORAGE:[0,100,0,A]

TRANSACTION

SENDER:4718bf7a
RECIPIENT:741F7A2
GAS PICE:0.5 ETH
GAS LIMIT:2 ETH
DATA:[....]
SIGNATURE:100FD

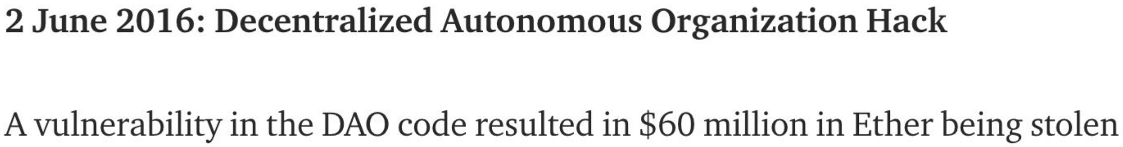
S'
FINAL STATE

ADDRESS:4718bf7a
BALANCE:8 ETH

ADDRESS:741F7a2
BALANCE: 12 ETH
[CODE...................]

STORAGE:[0,100,ETH,A]

Bugs

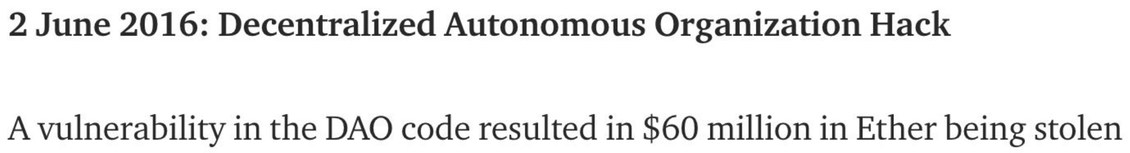


2 June 2016: Decentralized Autonomous Organization Hack

A vulnerability in the DAO code resulted in $60 million in Ether being stolen

https://medium.com/chainsecurity/the-5-most-costly-ethereum-security-bugs-616c649b6c86

Bugs



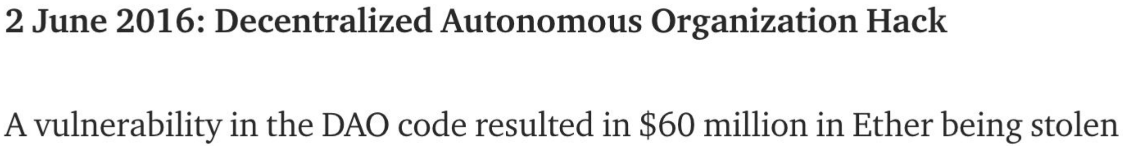
2 June 2016: Decentralized Autonomous Organization Hack

A vulnerability in the DAO code resulted in $60 million in Ether being stolen

3 July 2017 $30 Million: Ether Reported Stolen Due to Parity Wallet Breach

https://medium.com/chainsecurity/the-5-most-costly-ethereum-security-bugs-616c649b6c86

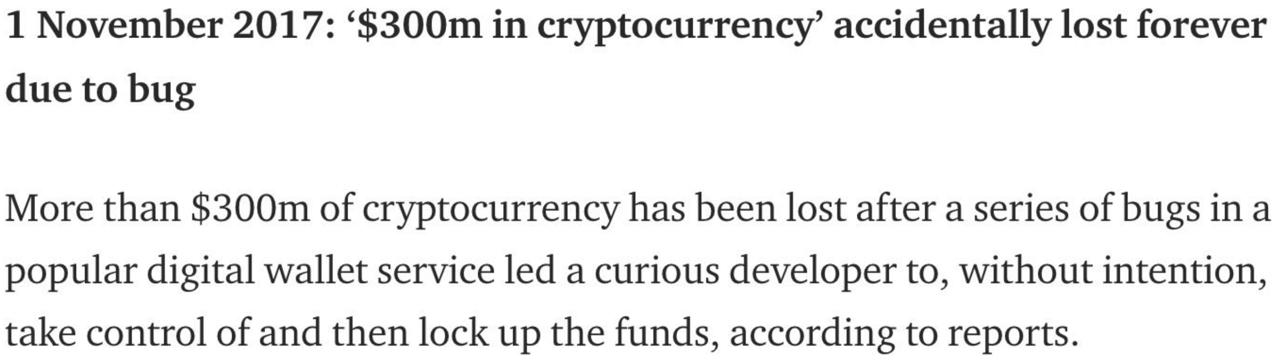
Bugs



2 June 2016: Decentralized Autonomous Organization Hack

A vulnerability in the DAO code resulted in $60 million in Ether being stolen

3 July 2017 $30 Million: Ether Reported Stolen Due to Parity Wallet Breach



1 November 2017: '$300m in cryptocurrency' accidentally lost forever due to bug

More than $300m of cryptocurrency has been lost after a series of bugs in a popular digital wallet service led a curious developer to, without intention, take control of and then lock up the funds, according to reports.

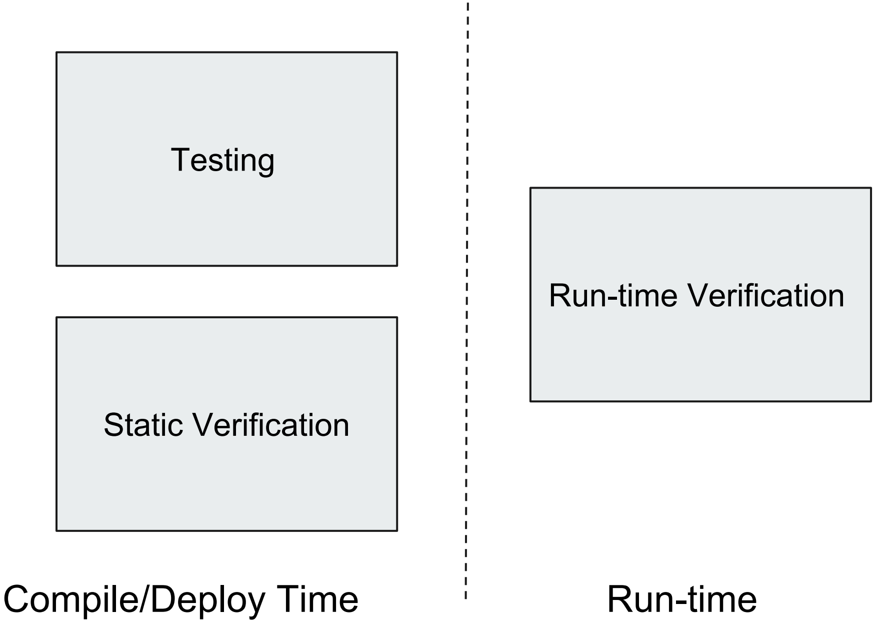
https://medium.com/chainsecurity/the-5-most-costly-ethereum-security-bugs-616c649b6c86

Challenge: Immutability

Contracts cannot be changed even if a bug is detected!

If a smart contract is doing something wrong… it’ll keep doing something wrong forever

Need for more assurances



Testing

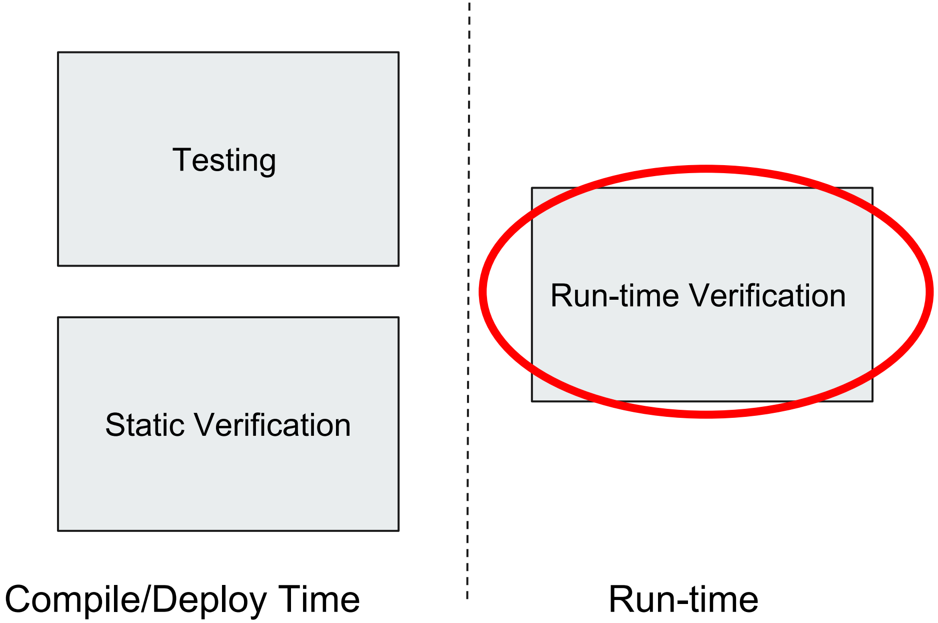
Static Verification

Run-time Verification

Compile/Deploy Time

Run-time

Need for more assurances



Verification

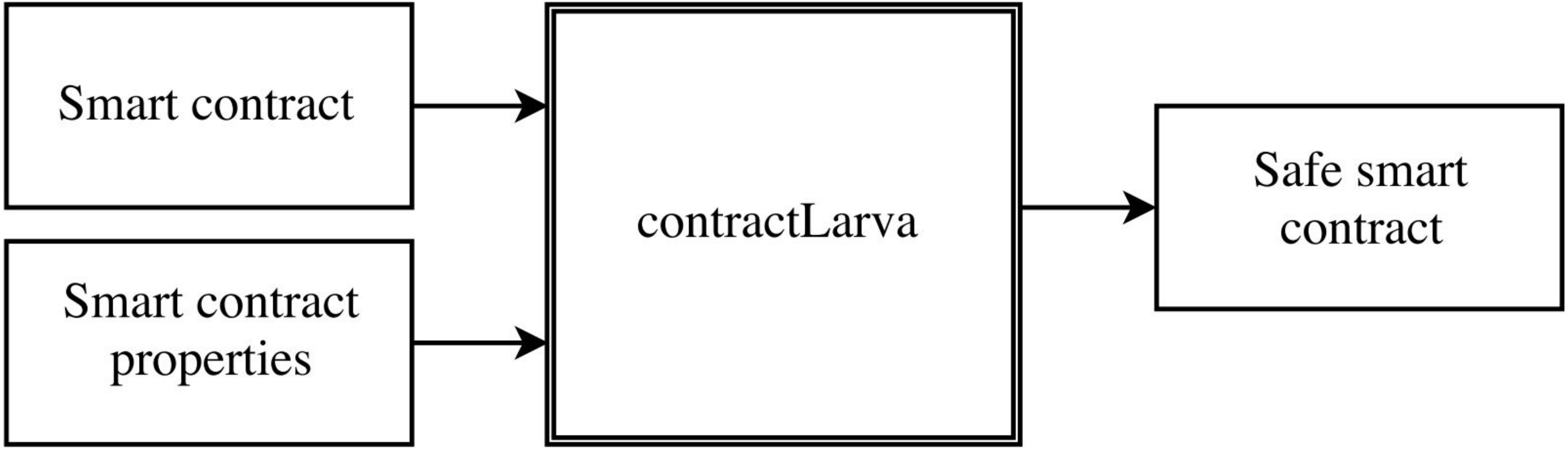
Static checking - ideal given immutability

- Solidity is not formally specified (yet?)

Runtime Verification

Checking the smart contract as it executes

ContractLarva



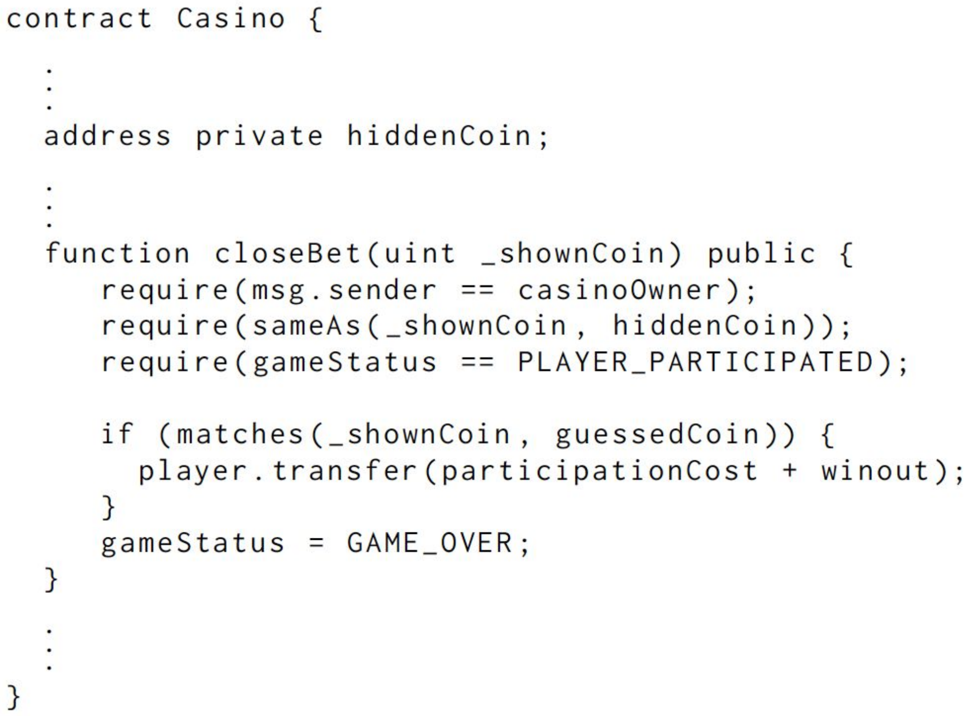
Smart contract

Smart contract properties

contractLarva

Safe smart contract

Coin flipping casino example (Solidity excerpt)



```
contract Casino {
  .
  .
  .
  address private hiddenCoin;
  .
  .
  .
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  .
  .
  .
}
```

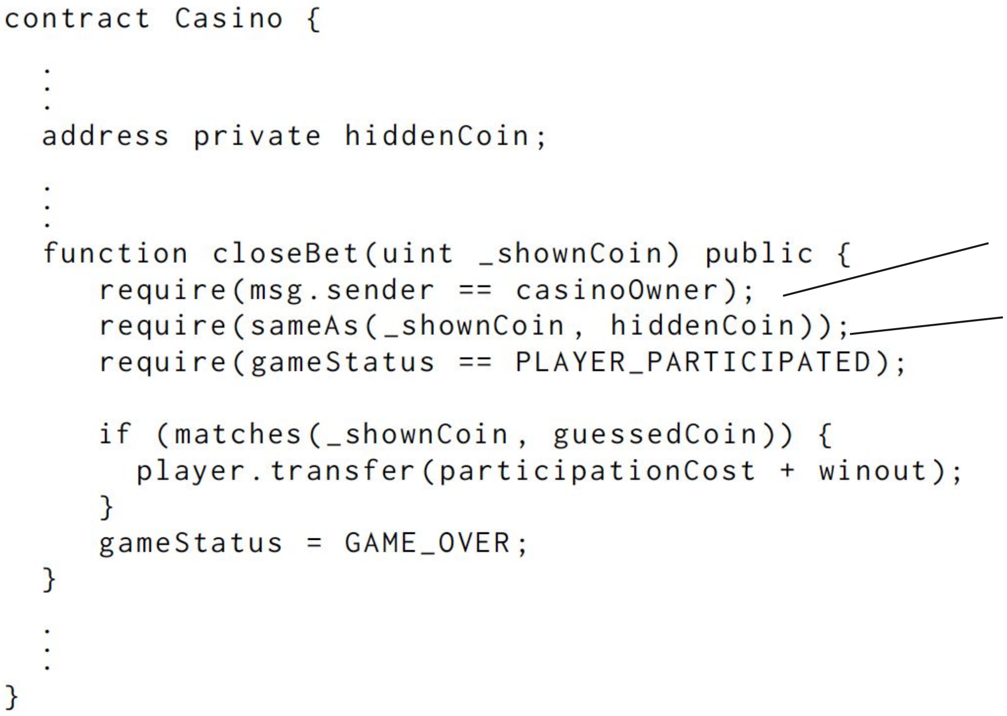
Coin flipping casino example (Solidity excerpt)

```
contract Casino {
  .
  .
  .
  address private hiddenCoin;
  .
  .
  .
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  .
  .
  .
}
```

Casino Owner is caller

Coin flipping casino example (Solidity excerpt)



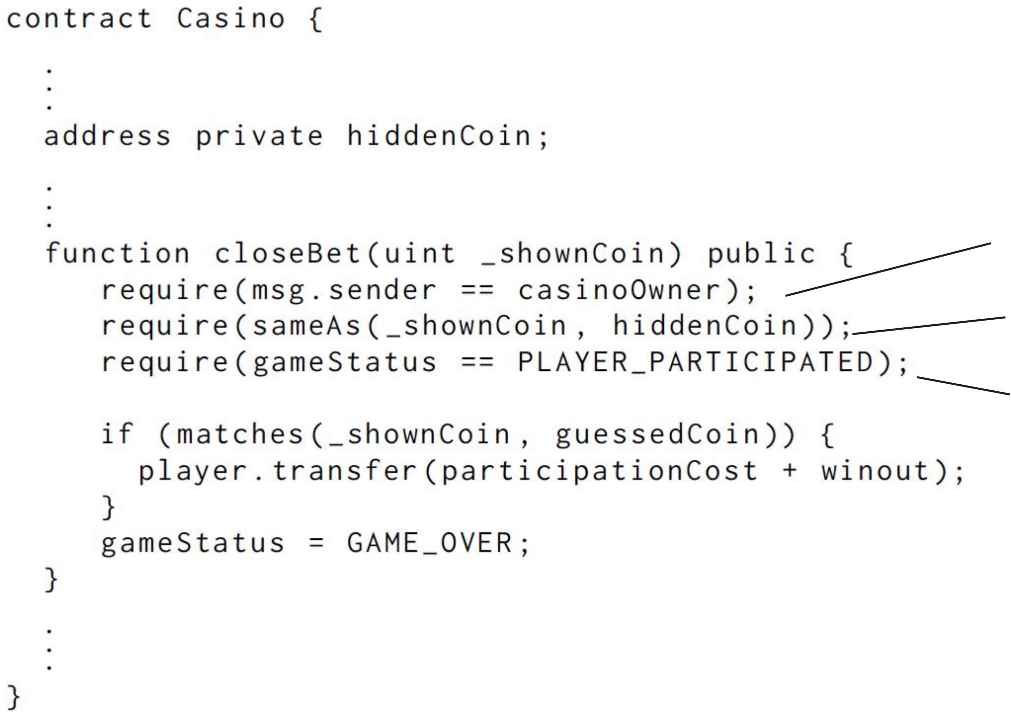
```
contract Casino {
  .
  .
  .
  address private hiddenCoin;
  .
  .
  .
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  .
  .
  .
}
```

Casino Owner is caller

Coin chosen initially is still the same

Coin flipping casino example (Solidity excerpt)



```
contract Casino {

  .
  .
  .
  address private hiddenCoin;

  .
  .
  .
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }

  .
  .
  .
}
```

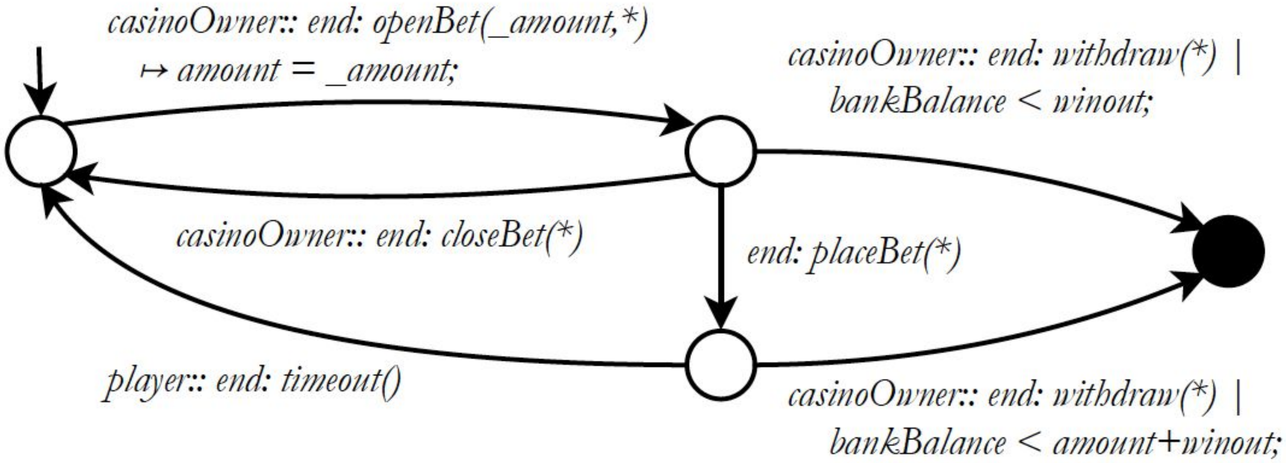
Casino Owner is caller

Coin chosen initially is still the same

At least 1 player played

Example property: Casino’s Bank can support bet

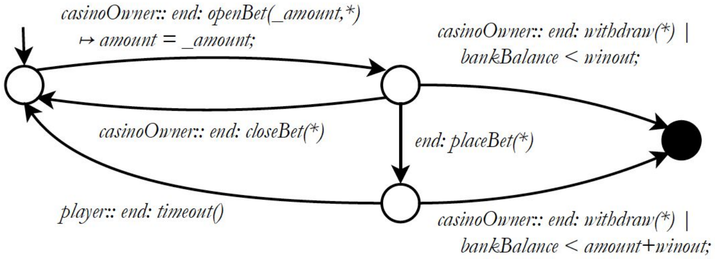
Dynamic Event Automaton:
DEA: event | condition => action
event: agent :: modality : solidity function

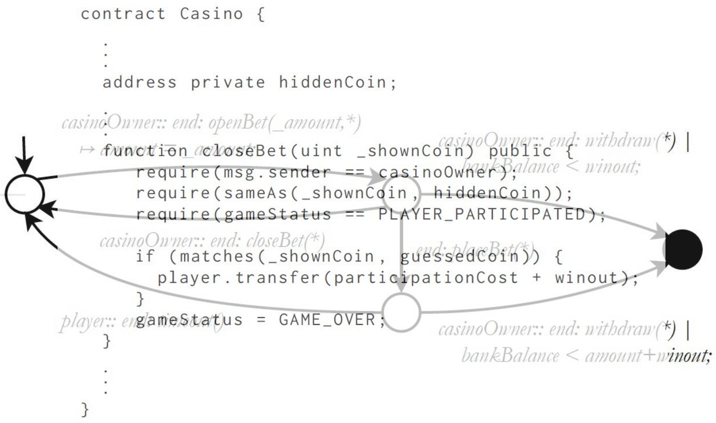

casinoOwner:: end: openBet(_amount,)*
↦ amount = _amount;
casinoOwner:: end: withdraw(*) |
bankBalance < winout;
casinoOwner:: end: closeBet(*)
end: placeBet(*)
player:: end: timeout()
casinoOwner:: end: withdraw(*) |
bankBalance < amount+winout;

ContractLarva

```
contract Casino {
  :
  :
  address private hiddenCoin;
  :
  :
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  :
  :
}
```


casinoOwner:: end: openBet(_amount,*)
↦ amount = _amount;
casinoOwner:: end: withdraw(*) |
bankBalance < winout;
casinoOwner:: end: closeBet(*)
end: placeBet(*)
player:: end: timeout()
casinoOwner:: end: withdraw(*) |
bankBalance < amount+winout;

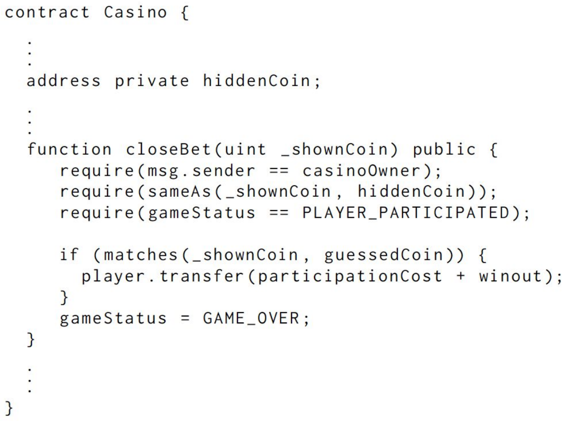


```
contract Casino {
  :
  :
  address private hiddenCoin;
  :
  :
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  :
  :
}
```

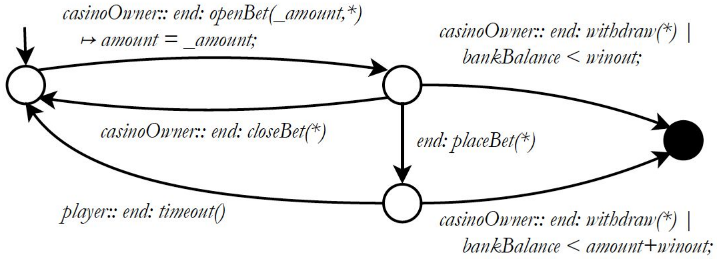
casinoOwner:: end: openBet(_amount,*)
↦ amount = _amount;
casinoOwner:: end: withdraw(*) |
bankBalance < winout;
casinoOwner:: end: closeBet(*)
end: placeBet(*)
player:: end: timeout()
casinoOwner:: end: withdraw(*) |
bankBalance < amount+winout;

ContractLarva



```
contract Casino {
  .
  .
  .
  address private hiddenCoin;
  .
  .
  .
  function closeBet(uint _shownCoin) public {
     require(msg.sender == casinoOwner);
     require(sameAs(_shownCoin, hiddenCoin));
     require(gameStatus == PLAYER_PARTICIPATED);

     if (matches(_shownCoin, guessedCoin)) {
       player.transfer(participationCost + winout);
     }
     gameStatus = GAME_OVER;
  }
  .
  .
  .
}
```


casinoOwner:: end: openBet(_amount,*)
↦ amount = _amount;
casinoOwner:: end: withdraw(*) |
bankBalance < winout;
casinoOwner:: end: closeBet(*)
end: placeBet(*)
player:: end: timeout()
casinoOwner:: end: withdraw(*) |
bankBalance < amount+winout;

Safe Smart Contract

Two challenges upon violation

BUT how do you deal with violations?

You cannot change the smart contract code!

When something goes wrong: Recovery action
Then, how to: Fix the code

Recovery

Immutability is not new

Other areas such as financial transactions already deal with immutability
* draw inspiration from existing work
(Colombo 2012)

‘Checkpointing’ in Ethereum

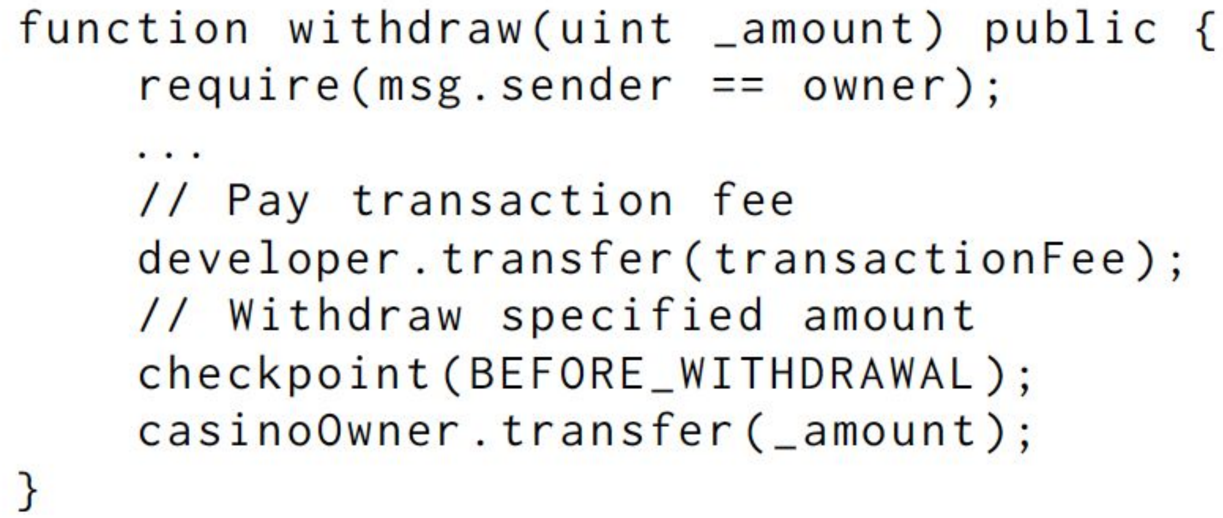
Ethereum natively supports checkpointing at the granularity of a function/transaction

If a violation is detected, reverting to initial state can be an option

This is useful but very coarse grained

Fine-grained checkpointing example

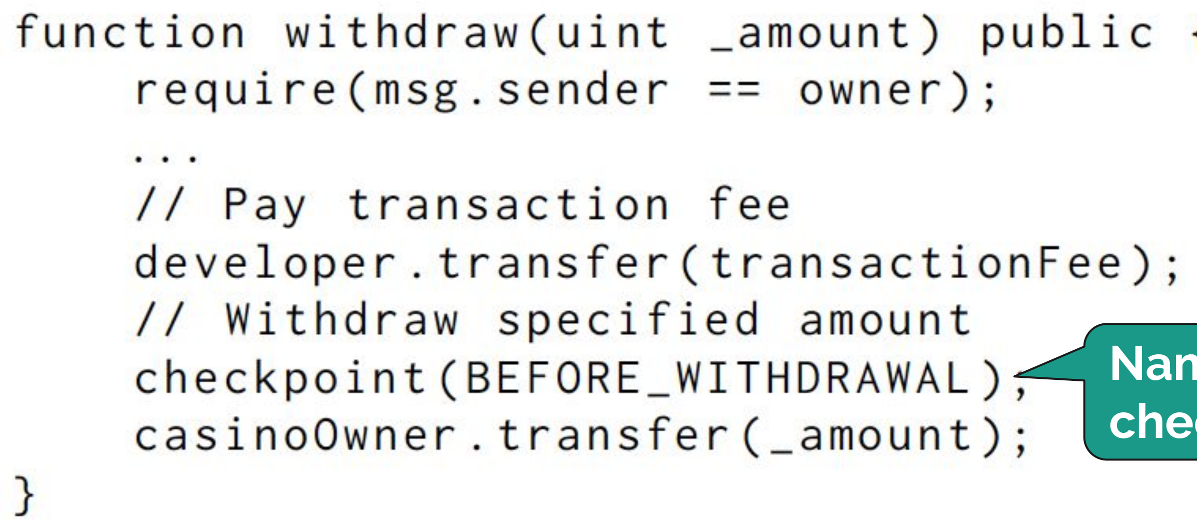
What if, you want to undo the transfer but keep the fee



```
function withdraw(uint _amount) public {
    require(msg.sender == owner);
    ...
    // Pay transaction fee
    developer.transfer(transactionFee);
    // Withdraw specified amount
    checkpoint(BEFORE_WITHDRAWAL);
    casinoOwner.transfer(_amount);
}
```

Fine-grained checkpointing example

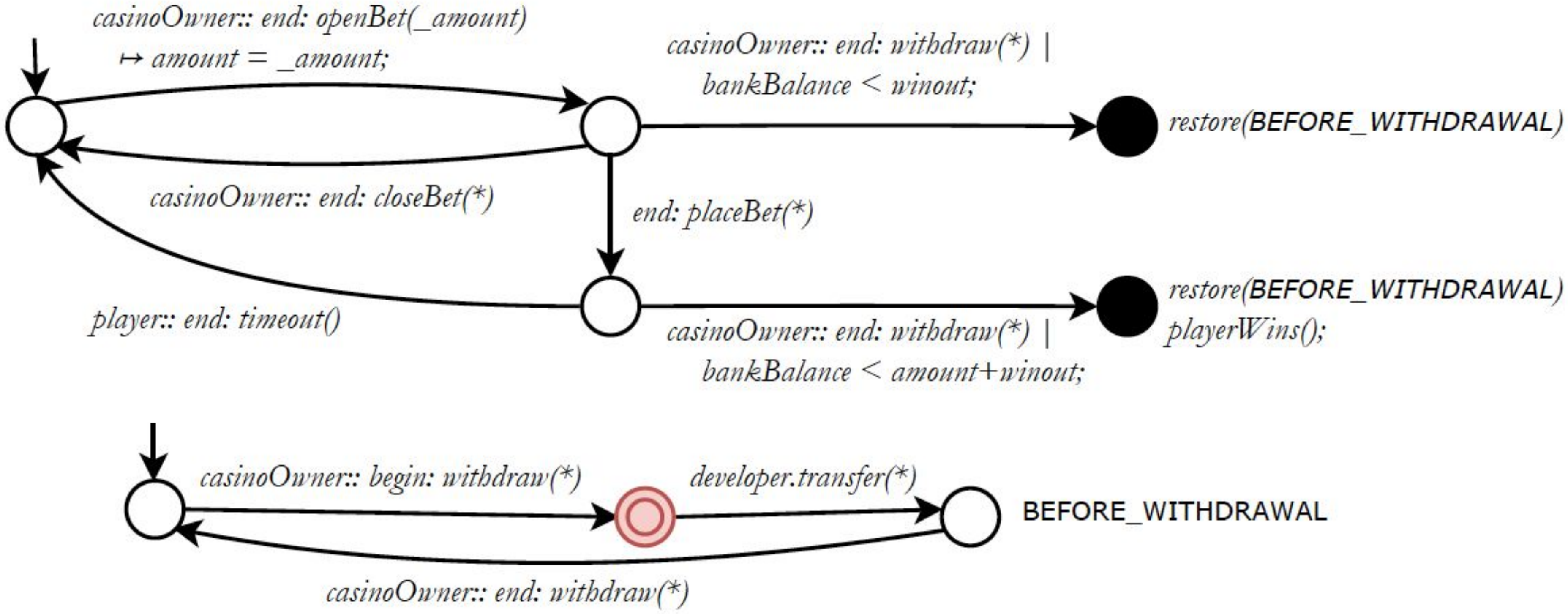
What if, you want to undo the transfer but keep the fee



```
function withdraw(uint _amount) public {
    require(msg.sender == owner);
    ...
    // Pay transaction fee
    developer.transfer(transactionFee);
    // Withdraw specified amount
    checkpoint(BEFORE_WITHDRAWAL);
    casinoOwner.transfer(_amount);
}
```

Named checkpoints

RV with checkpointing



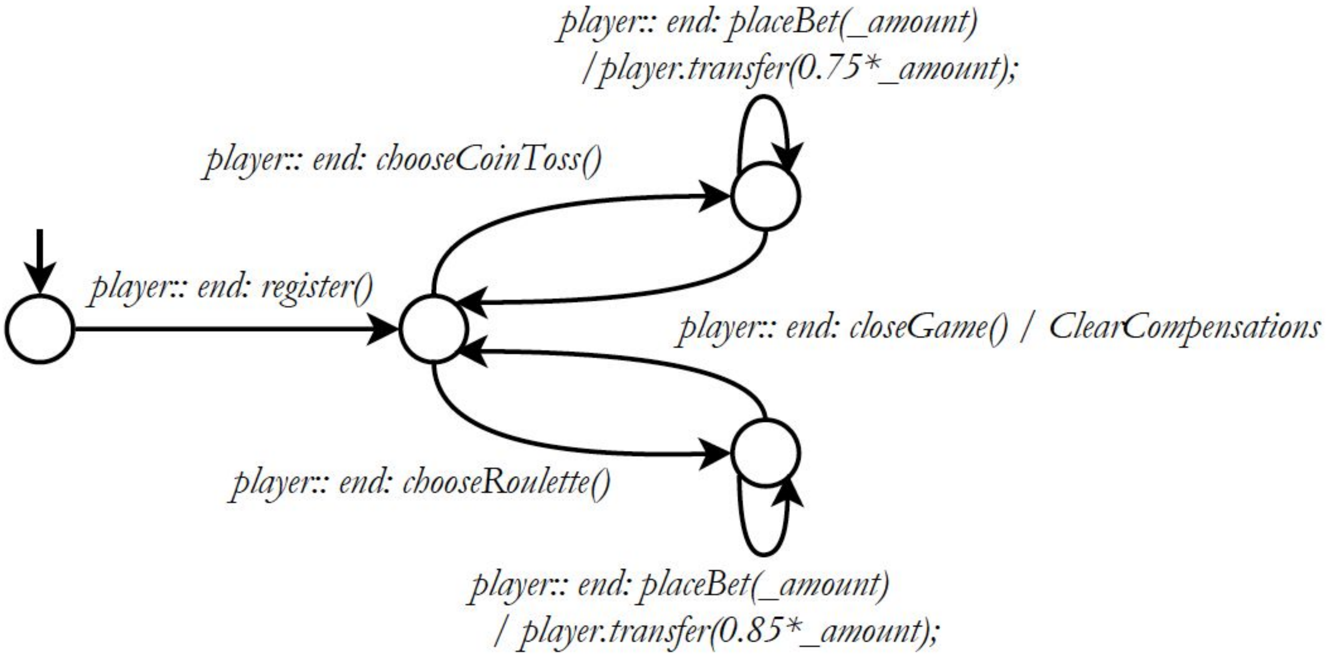
casinoOwner:: end: openBet(_amount)
↦ amount = _amount;
casinoOwner:: end: withdraw(*) |
bankBalance < winout;
restore(BEFORE_WITHDRAWAL)
casinoOwner:: end: closeBet(*)
end: placeBet(*)
restore(BEFORE_WITHDRAWAL)
playerWins();
player:: end: timeout()
casinoOwner:: end: withdraw(*) |
bankBalance < amount+winout;
casinoOwner:: begin: withdraw(*)
developer.transfer(*)
BEFORE_WITHDRAWAL
casinoOwner:: end: withdraw(*)

Compensations

Not all actions can be simply rolled back (as if they never happened)

At times preferable to run a “counter-action” - compensation

Compensations example


player:: end: placeBet(_amount)
/player.transfer(0.75*_amount);
player:: end: chooseCoinToss()
player:: end: register()
player:: end: closeGame() / ClearCompensations
player:: end: chooseRoulette()
player:: end: placeBet(_amount)
/ player.transfer(0.85*_amount);

Fixing code

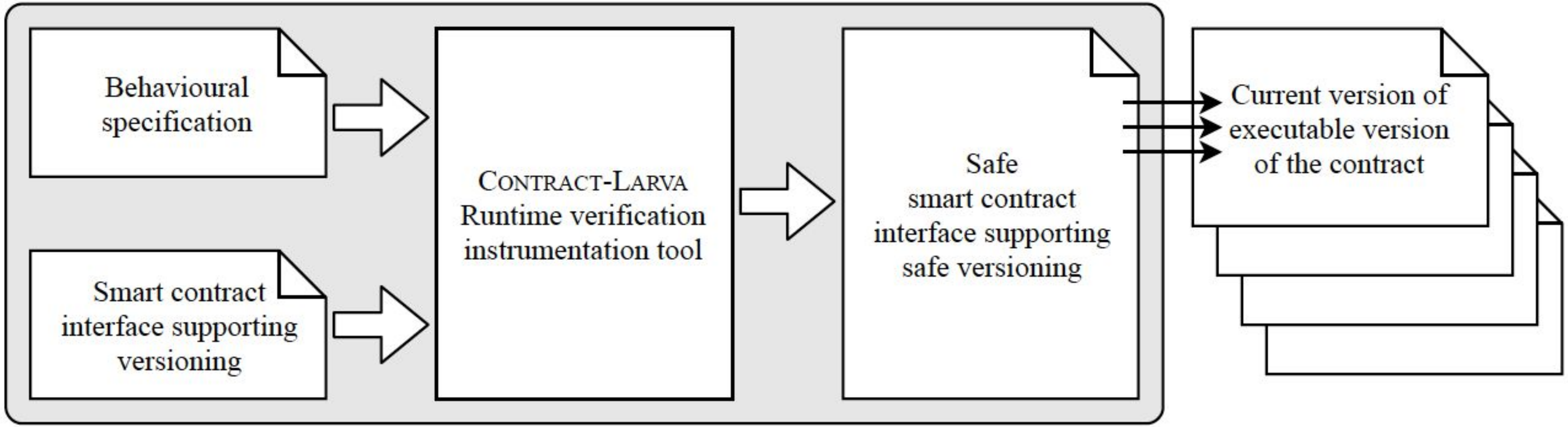
Fixing smart contract code

Once violation is detected (through RV) how can we fix the code for good?

RV can help again...

Specification-oriented approach

1. Expose an interface of the contract
2. Pass interface calls to the **current implementation** (can be updated)
3. Instrument implementation to **ensure specification is adhered to**



Behavioural specification
Smart contract interface supporting versioning
CONTRACT-LARVA Runtime verification instrumentation tool
Safe smart contract interface supporting safe versioning
Current version of executable version of the contract

ContractLarva

https://github.com/gordonpace/contractLarva

Conclusion

Smart contracts pose new challenges due to their immutability:

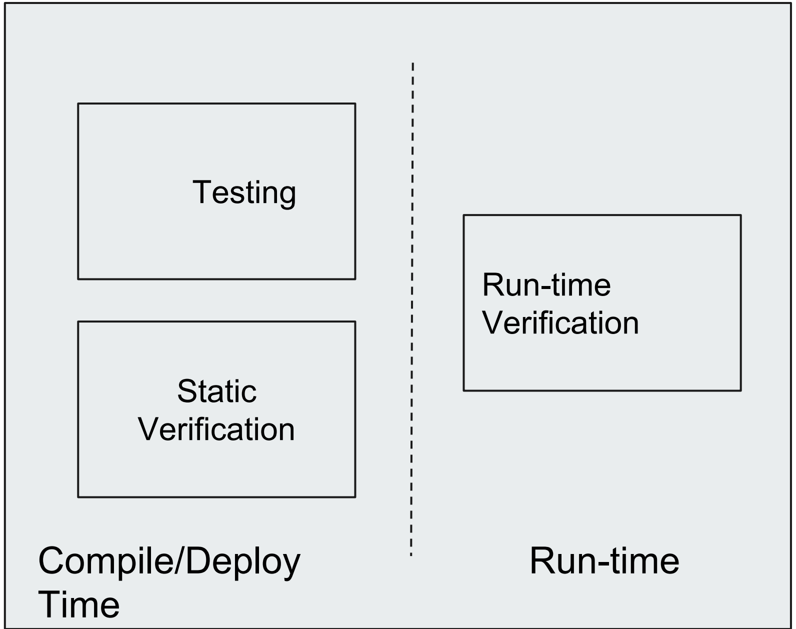
- Recovery
- Fixing code

Compensations can provide flexible yet automated recovery

RV can provide assurance that specification is respected even after code updates

Need for More Software Assurances

If a smart contract is doing something wrong, it'll keep doing so forever (bug, illegalities)



Testing

Static Verification

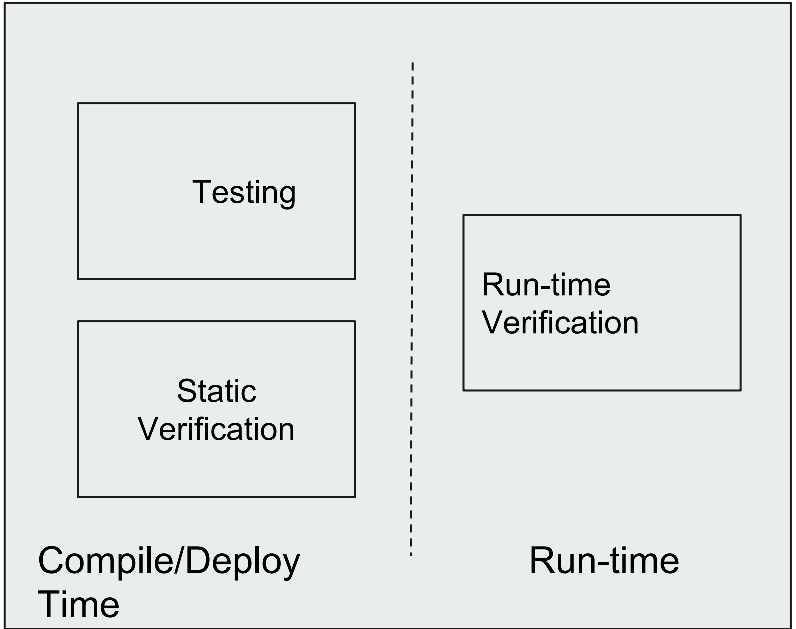
Run-time Verification

Compile/Deploy Time

Run-time

Need for More Software Assurances

If a smart contract is doing something wrong, it'll keep doing so forever (bug, illegalities)



Is this good enough?

- Static verification and RV are as good as the specification

Proxy calls?
Trade-offs? What guarantees are Users agreeing to?
Can ContractLarva-like specifications help here?

More testing?
More eyes?

Contact Us

Joshua Ellul	✉	joshua.ellul@um.edu.mt
Christian Colombo	✉	christian.colombo@um.edu.mt
Gordon Pace	✉	gordon.pace@um.edu.mt