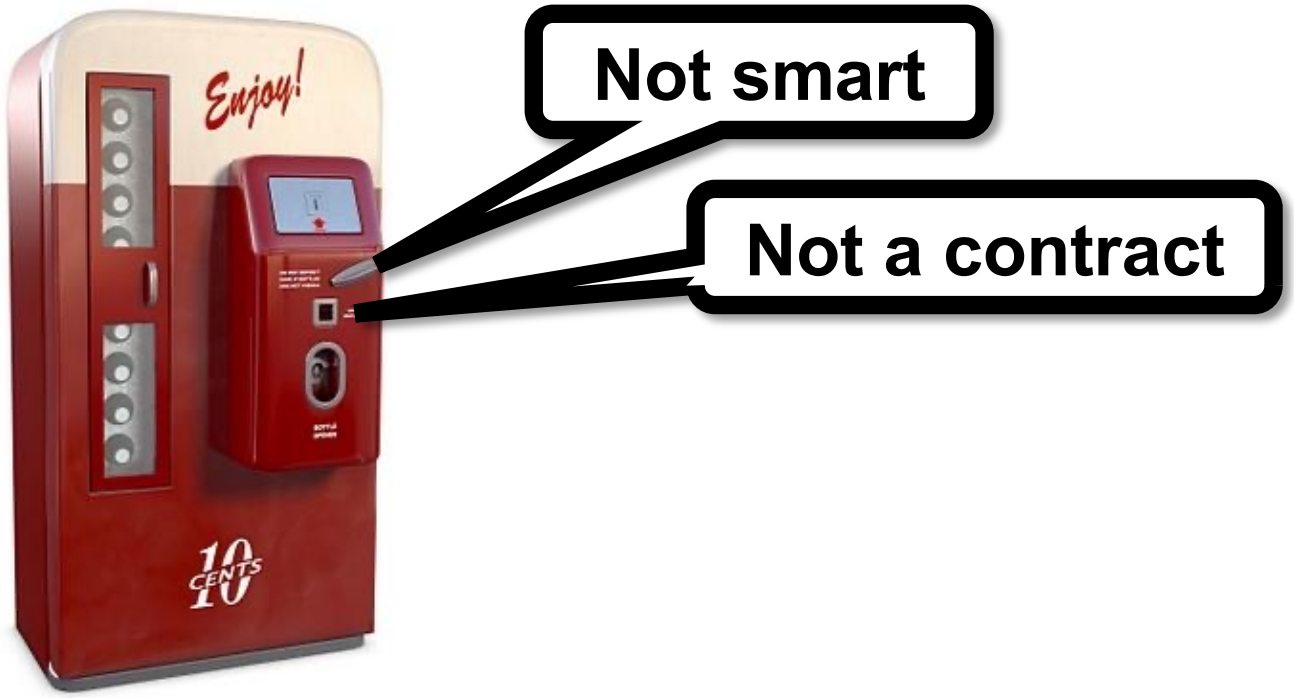
TÉCNICO LISBOA

Solidity

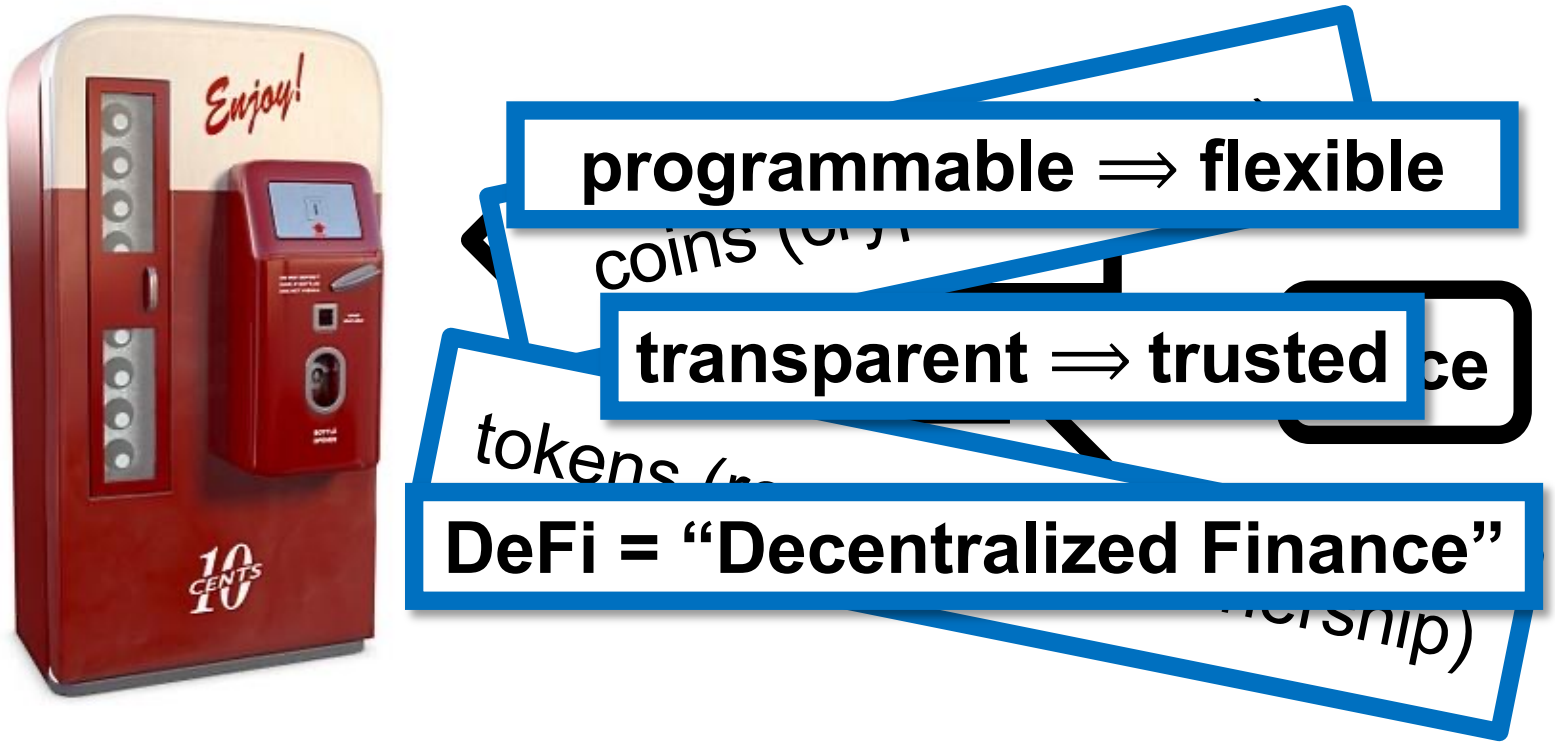
Highly dependable systems – 2024/25

Lecture 9

Smart Contracts



Smart Contracts



Solidity

- Solidity is a high-level language for the EVM
- Based on Javascript
- In the lecture we won't focus on how to program in Solidity
- But rather on the design objectives and challenges
  - Recall: Smart Contracts are executed in a Byz. environment

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 …
}
```

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  mapping (address => uint) balances;
  …
}
```

This program needs compiler version 0.5.2 or later

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  address minter;
  m                                    s;
  …
}
```

A contract is like a class.
This contract manages a simple coin

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  address minter;
  mapping (address => uint) balances;
  …
}
```

long-lived state in EVM storage

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
}
```

Name of an account (person or contract).

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
}
```

Name of an account (person or contract).

Here, only one allowed to mint new coins.

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  address minter;
  mapping (address => uint) balances;
  …
}
```

Like a hash table

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  address minter;
  mapping (address => uint) balances;
  …
}
```

Like a hash table

Initially, every address maps to zero

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
  address minter;
  mapping (address => uint) balances;
  …
}
```

Like a hash table

Initially, every address maps to zero

Tracks client balances

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 constructor() public {
   minter = msg.sender;
 }
```

Initializes a contract

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 constructor() public {
   minter = msg.sender;
 }
}
```

address of contract making the call

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 constructor() public {
  minter = msg.sender;
 }
```

Remember the creator's address.

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 ..
 function mint(address owner, uint amount)
  public {
    if (msg.sender != minter) return;
    balances[owner] += amount;
  }
 ..
}
```

A function is like a method.

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 …
 f
amount)
   public {
    if (msg.sender != minter) return;
    balances[owner] += amount;
 }
 …
}
```

If the caller is not authorized, just return.

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 …
 function mint(address owner, uint
   public {
    if (msg.sender != minter) return;
    balances[owner] += amount;
  }
 …
}
```

Otherwise credit the amount to the owner's balance

Coin example

```
pragma solidity ^0.5.2;
```

One party transfers coins to counterparty

```
 mapping (address => uint) balances;
 …
 function send(address receiver, uint amount)
  public {
    if (balances[msg.sender] < amount) return;
    balances[msg.sender] -= amount;
    balances[receiver] += amount;
 }
…
}
```

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 …
 function send(address receiver, uint amount)
  public {
   if (balances[msg.sender] < amount) return;
   balances[msg.sender] -= amount;
   balances[receiver] += amount;
 }
…
}
```

Quit if insufficient funds

Coin example

```
pragma solidity ^0.5.2;
contract Coin {
 address minter;
 mapping (address => uint) balances;
 .
 f
  public {
   if (balances[msg.sender] < amount) return;
   balances[msg.sender] -= amount;
   balances[receiver] += amount;
 }
…
}
```

transfer amount from one account to the other

Coin Flipping

Alice and Bob want to flip a fair coin

Each one is willing to cheat

Obvious sources of randomness:
computer time, block hashes, etc. don't work

Let's try something else

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
…
```

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
```

declare contract and compiler version

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
```

will want to restrict interactions to the two players

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
…
```

each player contributes to the result

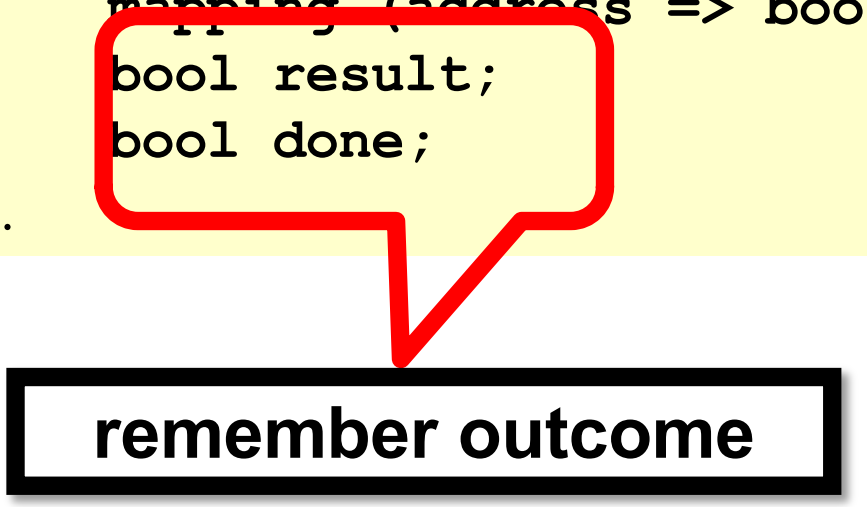
Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
…
```

each player gets one move only

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
    address[2] player;
    mapping (address => uint) vote;
    mapping (address => bool) played;
    bool result;
    bool done;
…
```



Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
…
 constructor (address alice, address bob)
 public {
        player[0] = alice;
        player[1] = bob;
    }
…
```

Coin Flipping

```
pragma solidity ^0.5.1;
contract CoinFlipMaybe {
…
 constructor (address alice, address bob)
 public {
      player[0] = alice;
      player[1] = bob;
    }
…
```

constructor just remembers players

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
                vote[player[1]]) % 2) == 1;
  }
}
```

Coin Flipping

```
function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  played[msg.sender] = true;
  vote[msg.sender] = _vote;
  if (played[player[0]] && played[player[1]]) {
    result = ((vote[player[0]] ^
                vote[player[1]]) % 2) == 1;
  }
}
```

require(condition)
revert computation if condition not satisfied.
Like throwing an exception

Coin Flipping

```
function play(uint _vote) public {
 require(msg.sender == player[0] ||
             msg.sender == player[1]);
 require(!played[msg.sender]);
 vote[msg.sender] = _vote;
 played[msg.sender] = true;
 if (played[player[0]] && played[player[1]]) {
  done = true;
  result = ((vote[player[0]] +
             vote[player[1]]) % 2) == 1;
 }
}
```

Only Alice and Bob allowed to play

Coin Flipping

```
function play(uint _vote) public {
 require(msg.sender == player[0] ||
                msg.sender == player[1]);
 require(!played[msg.sender]);
 vote[msg.sender] = _vote;
 played[msg.sender] = true;
 if (played[player[0]] && played[player[1]]) {
  done = true;
  result = ((vote[player[0]] ^
             vote[player[1]]) % 2) == 1;
 }
}
```

Each player can play only once

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
                vote[player[1]]) % 2) == 1;
  }
}
```

record player's vote

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
                vote[player[1]]) % 2) == 1;
  }
}
```

record that player voted

Coin Flipping

```
function play(uint _vote) public {
 require(msg.sender == player[0] ||
              msg.sender == player[1]);
 require(!played[msg.sender]);
 vote[msg.sender] = _vote;
 played[msg.sender] = true;
 if (played[player[0]] && played[player[1]]) {
  done = true;
  result = ((vote[player[0]] ^
             vote[player[1]]) % 2) == 1;
 }
}
```

we are done if both players voted

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
              vote[player[1]]) % 2) == 1;
  }
}
```

remember we are done

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
              vote[player[1]]) % 2) == 1;
  }
}
```

take XOR of votes

Coin Flipping

```
 function play(uint _vote) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!played[msg.sender]);
  vote[msg.sender] = _vote;
  played[msg.sender] = true;
  if (played[player[0]] && played[player[1]]) {
   done = true;
   result = ((vote[player[0]] ^
                vote[player[1]]) % 2) == 1;
  }
}
```

flip value is parity of XOR of votes

Coin Flipping

```
function getResult() public view returns (bool) {
      require (done);
      return result;
   }
```

Coin Flipping

```
function getResult() public view returns (bool) {
    require (done);
    return result;
}
```

anyone can request outcome

Coin Flipping

```
function getResult() public view returns (bool) {
    require (done);
    return result;
}
```

make sure outcome is ready

Coin Flipping

This contract is insecure!

Why?

Coin Flipping

Suppose Alice plays first

Bob then observes contract state on Etherscan.io

Picks his vote after seeing hers

Bob controls coin flip outcome

Commit-Reveal Pattern

Step One: commit to a value without revealing it

you cannot change your value

Step Two: Wait for others to commit

Step Three: reveal your value

you can check validity

Commit-Reveal Pattern

Alice generates a random value r

Alice commits to r by sending hash(r) to contract

Alice waits for Bob to commit

Alice reveals r to contract

Compute outcome when Bob reveals

Commit-Reveal Pattern

```
contract CoinFlipCR {
    address[2] player;
    mapping (address => uint) commitment;
    mapping (address => uint) revelation;
    mapping (address => bool) committed;
    mapping (address => bool) revealed;
    bool result;
    bool done;
```

Commit-Reveal Pattern

```
contract CoinFlipCR {
    address[2] player;
    mapping (address => uint) commitment;
    mapping (address => uint) revelation;
    mapping (address => bool) committed;
    mapping (address => bool) revealed;
    bool result;
    bool done;
```

track both commit and reveal

Commit-Reveal Pattern

```
function commit(uint _commit) public {
      require(msg.sender == player[0] ||
            msg.sender == player[1]);
      require(!committed[msg.sender]);
      commitment[msg.sender] = _commit;
      committed[msg.sender] = true;
   }
```

Commit-Reveal Pattern

```
function commit(uint _commit) public {
    require(msg.sender == player[0] ||
            msg.sender == player[1]);
    require(!committed[msg.sender]);
    commitment[msg.sender] = _commit;
    committed[msg.sender] = true;
}
```

members only

Commit-Reveal Pattern

```
function commit(uint _commit) public {
      require(msg.sender == player[0] ||
              msg.sender == player[1]);
      require(!committed[msg.sender]);
      commitment[msg.sender] = _commit;
      committed[msg.sender] = true;
   }
```

You only commit once

Commit-Reveal Pattern

```
function commit(uint _commit) public {
      require(msg.sender == player[0] ||
            msg.sender == player[1]);
      require(!committed[msg.sender]);
      commitment[msg.sender] = _commit;
      committed[msg.sender] = true;
    }
```

close the deal

```
 function reveal(uint _reveal) public {
  require(msg.sender == player[0] ||
              msg.sender == player[1]);
  require(!revealed[msg.sender]);
  require(
    commitment[msg.sender]==hash(_reveal));
  revelation[msg.sender] = _reveal;
  revealed[msg.sender] = true;
  if (revealed[player[0]] &&
      revealed[player[1]]) {
   done = true;
   result = ((revelation[player[0]] ^
              revelation[player[1]])
                   % 2) == 1;
  }
}
```

```
function reveal(uint _reveal) public {
  require(msg.sender == player[0] ||
              msg.sender == player[1]);
  require(!revealed[msg.sender]);
  require(
    commitment[msg.sender]==hash(_reveal));
  revelation[m
  revealed[msg.sender] = true;
  if (revealed[player[0]] &&
      revealed[player[1]]) {
   done = true;
   result = ((revelation[player[0]] ^
              revelation[player[1]])
                   % 2) == 1;
  }
}
```

Usual preconditions

```
 function reveal(uint _reveal) public {
  require(msg.sender == player[0] ||
                msg.sender == player[1]);
  require(!revealed[msg.sender]);
  require(
    commitment[msg.sender]==hash(_reveal));
  revelation[msg.sender] = _reveal;
  revealed[msg.sender] = true;
  if (revealed[player[0]] &&
      revealed[player[1]]) {
   done = true;
   result = ((revelation[player[0]] ^
              revelation[player[1]])
                   % 2) == 1;
  }
}
```

make sure reveal is sincere

```
function reveal(uint _reveal) public {
 require(msg.sender == player[0] ||
             msg.sender == player[1]);
 require(!revealed[msg.sender]);
 require(
   commitment[msg.sender]==hash(_reveal));
 revelation[msg.sender] = _reveal;
 revealed[msg.sender] = true;
 if (revealed[player[0]] &&
     revealed[player[1]]) {
  done = true;
  result = ((r
             revelation[player[1]])
                  % 2) == 1;
 }
}
```

record revelation

```
function reveal(uint _reveal) public {
 require(msg.sender == player[0] ||
                 m
 require(!reve
 require(
   commitment[msg.sender]==hash(_reveal));
 revelation[msg.sender] = _reveal;
 revealed[msg.sender] = true;
 if (revealed[player[0]] &&
     revealed[player[1]]) {
  done = true;
  result = ((revelation[player[0]] ^
             revelation[player[1]])
                  % 2) == 1;
 }
}
```

If done, compute outcome as before

Smart Contract Programming

- Adversarial environment
- Modularity and information hiding often not enough
  - Attacker does not respect abstraction boundaries
- Smart contracts need to withstand attacks from motivated attackers
- Potential large gains for successful exploits

Javascript

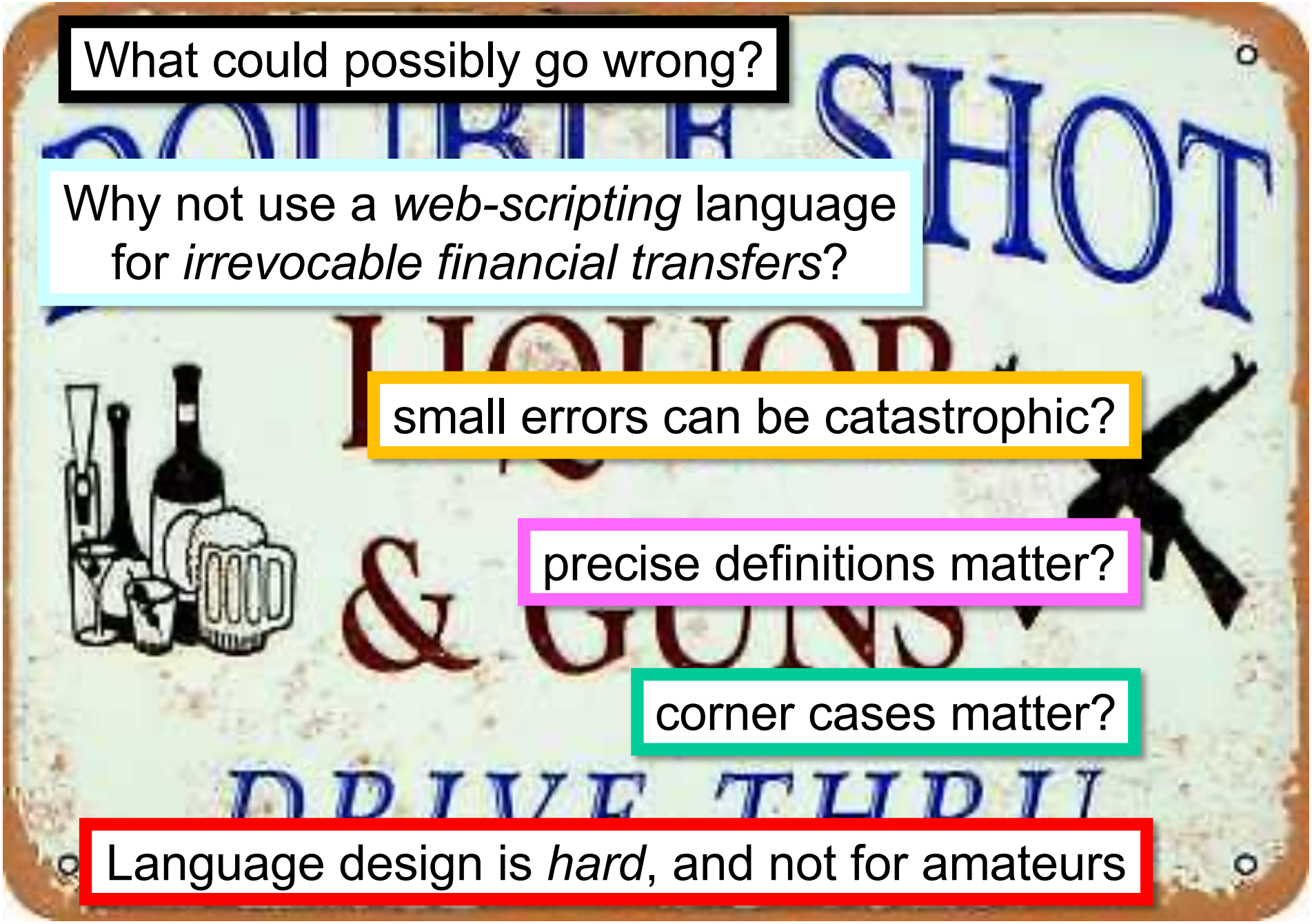
Intended for web page presentation

Odd, complex behavior considered normal

No precise notion of correctness

Must be “easy” for non-experts

No actual adversary


What could possibly go wrong?
Why not use a *web-scripting* language for *irrevocable financial transfers*?
small errors can be catastrophic?
precise definitions matter?
corner cases matter?
Language design is *hard*, and not for amateurs

Rest of this lecture

Review several vulnerabilities

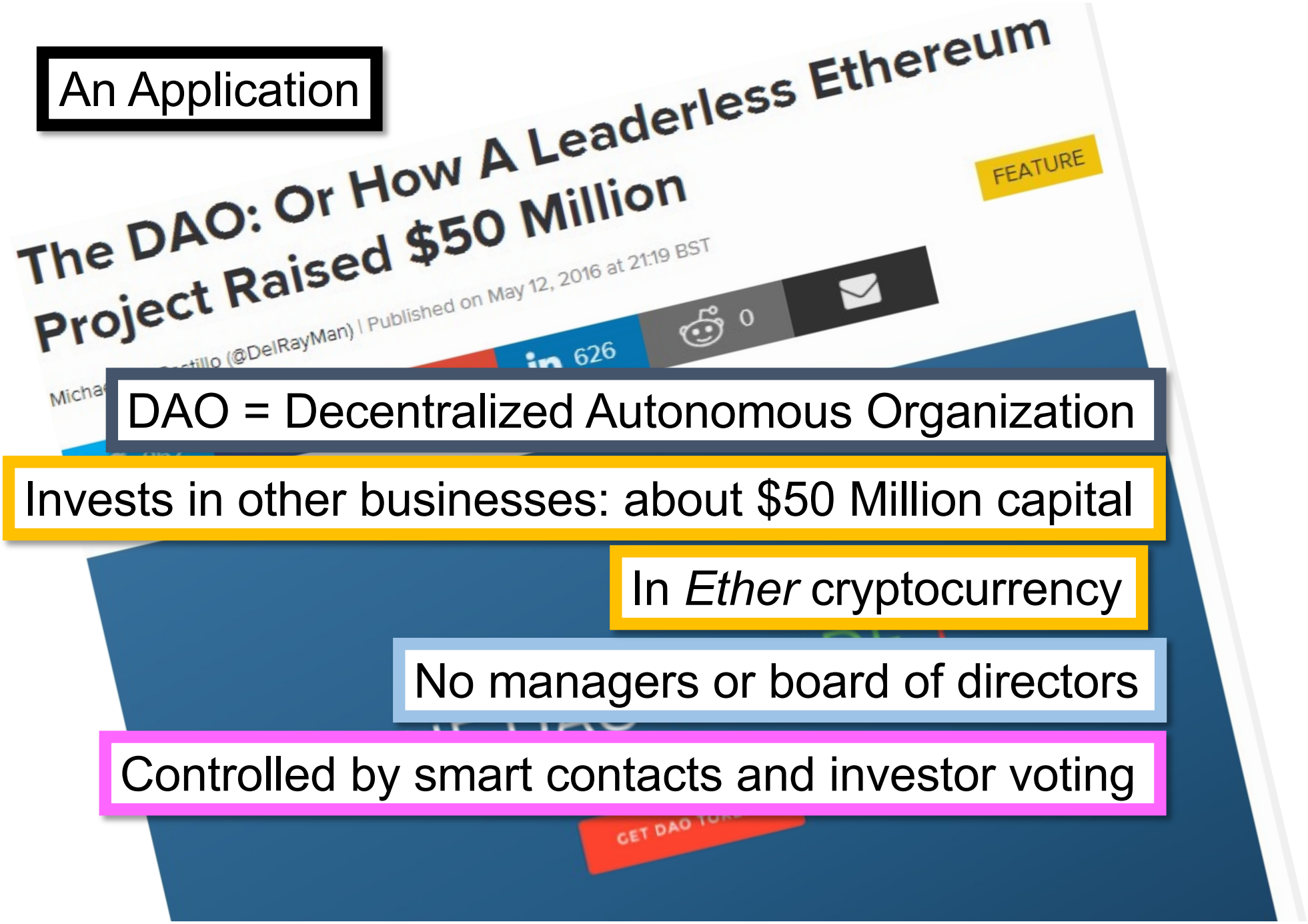
Some common to all blockchains

Some are Solidity idiosyncrasies …

"nonsense is nonsense, but the history of nonsense is scholarship"

Re-Entrancy Attack

An Application



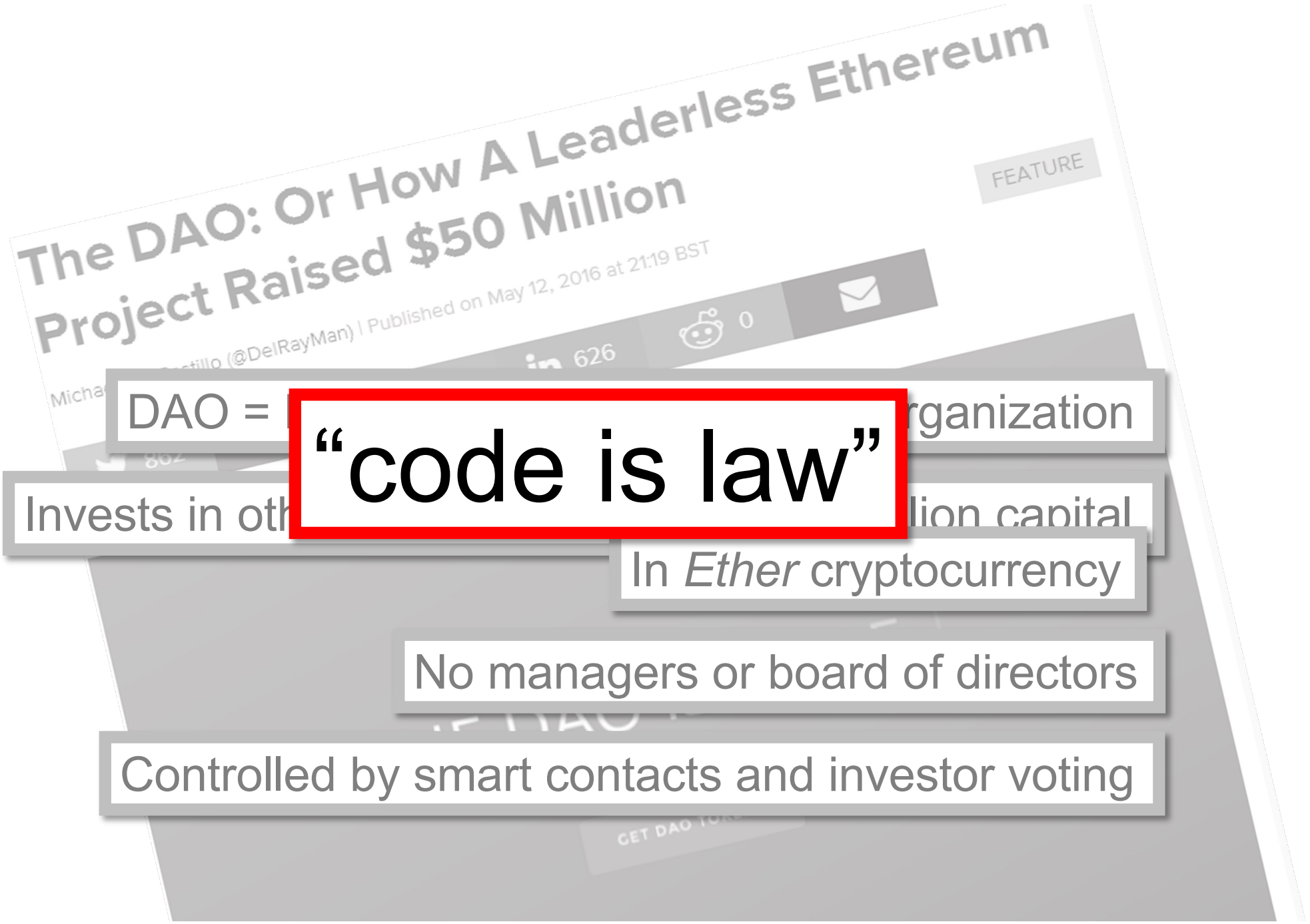
DAO = Decentralized Autonomous Organization

Invests in other businesses: about $50 Million capital

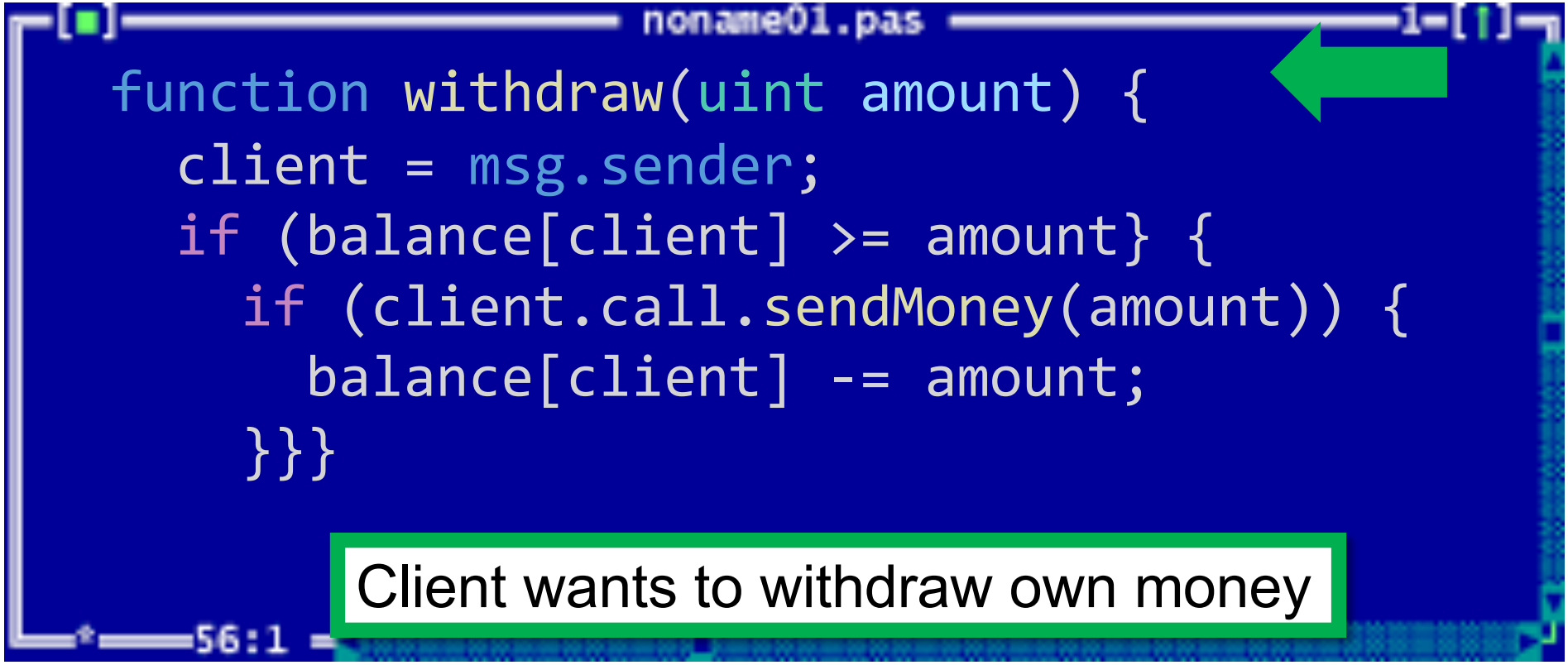
In *Ether* cryptocurrency

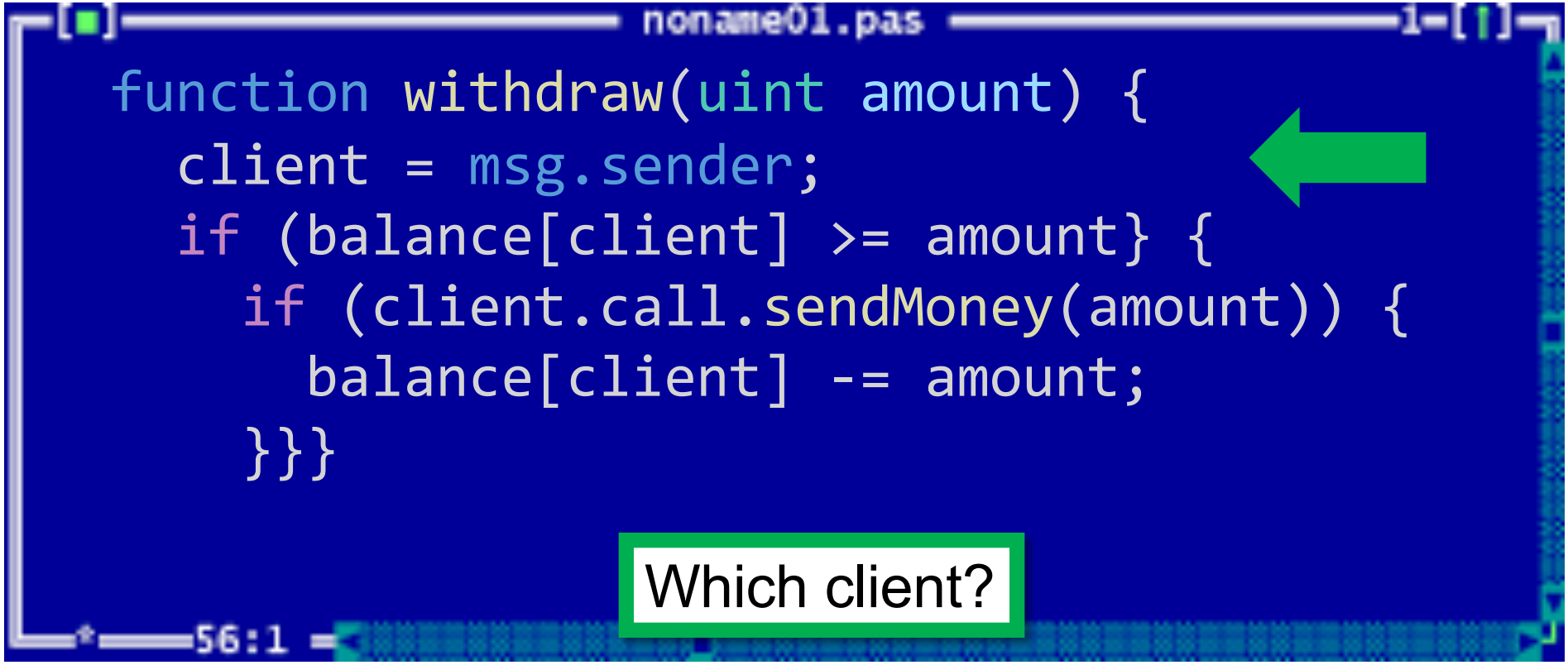
No managers or board of directors

Controlled by smart contacts and investor voting

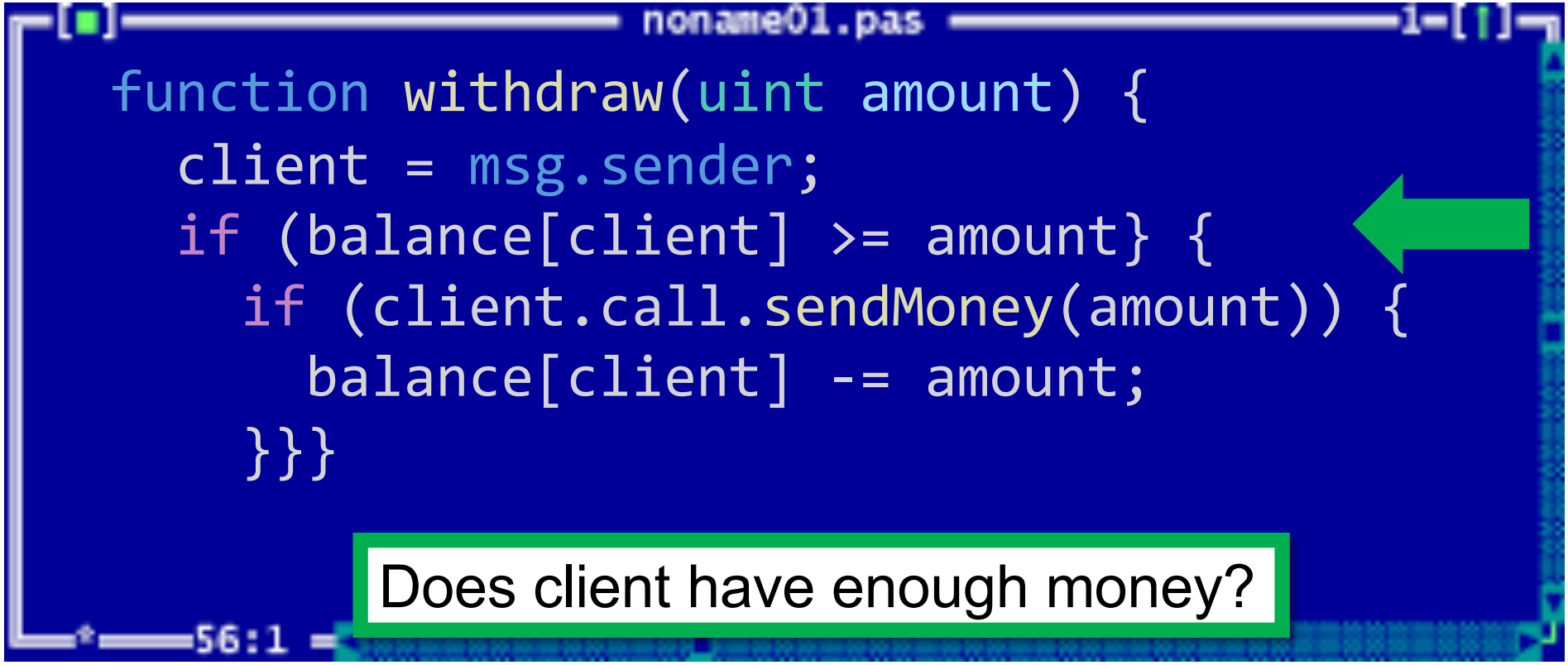

The DAO: Or How A Leaderless Ethereum Project Raised $50 Million
FEATURE
Published on May 12, 2016 at 21:19 BST
626
0
DAO =
rganization
Invests in oth
lion capital
“code is law”
In *Ether* cryptocurrency
No managers or board of directors
Controlled by smart contacts and investor voting


The DAO: a radical experiment that could be the future of decentralised governance
May 10, 2016
The DAO is a New Dow
Nolan Bauerle | Published on May 22, 2016 at 17:22 BST
OPINION
235
17
240
22
Why the DAO Ethereum is Revoluti
By Adam Hayes, CFA | May 16, 2016 — 2:02 PM EDT
On April 30, 2016, a brand new organizational stru
organization, or DAO. This organization, buil
blockchain has already raised over $4
project to date. What is the DA
What Is the DAO?
The blockchain is a sh
permanent re
most well-kno
blockchain fro
to be coded dire
disparate, anony
enforcement
BLOCKCHAIN REVOLUTION
the future of business a company
kers, managers, or a
Much hyperventilation about effect on future of finance


noname01.pas
function withdraw(uint amount) {
client = msg.sender;
if (balance[client] >= amount} {
if (client.call.sendMoney(amount)) {
balance[client] -= amount;
}}}
Client wants to withdraw own money
56:1


noname01.pas
function withdraw(uint amount) {
client = msg.sender;
if (balance[client] >= amount} {
if (client.call.sendMoney(amount)) {
balance[client] -= amount;
}}}
Which client?
56:1

72


noname01.pas
function withdraw(uint amount) {
client = msg.sender;
if (balance[client] >= amount} {
if (client.call.sendMoney(amount)) {
balance[client] -= amount;
}}}
Does client have enough money?
56:1

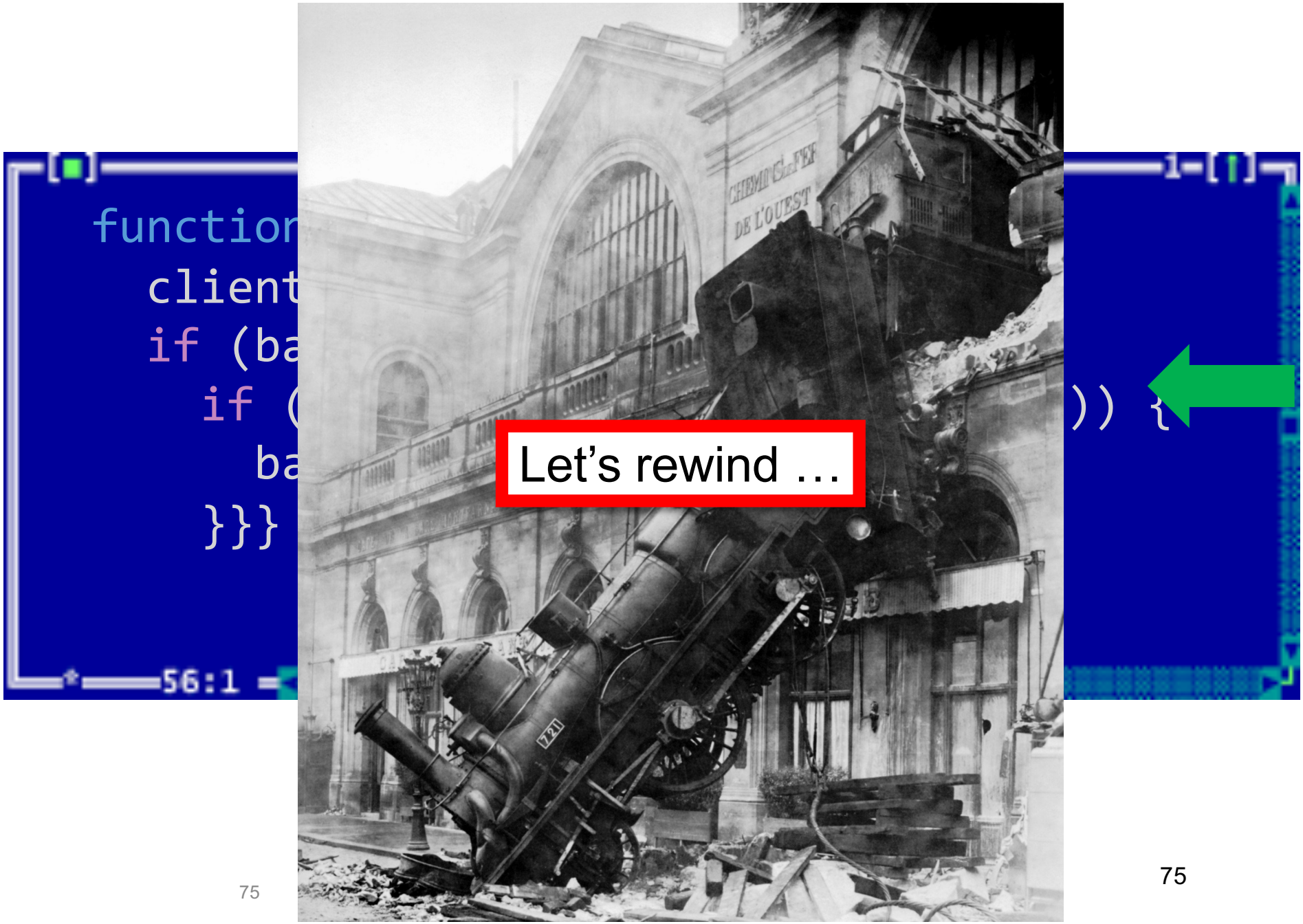
73

noname01.pas

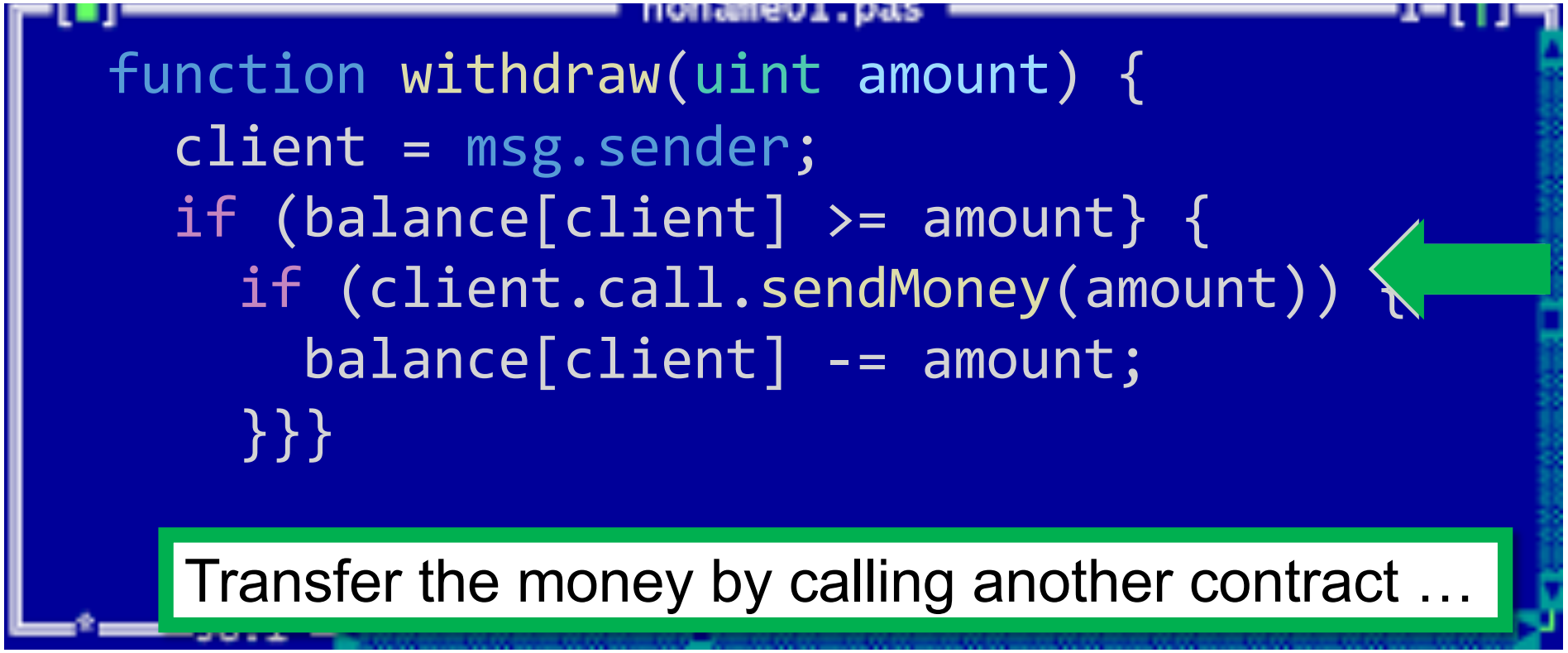
```
function withdraw(uint amount) {
  client = msg.sender;
  if (balance[client] >= amount} {
    if (client.call.sendMoney(amount)) {
      balance[client] -= amount;
    }}}
```

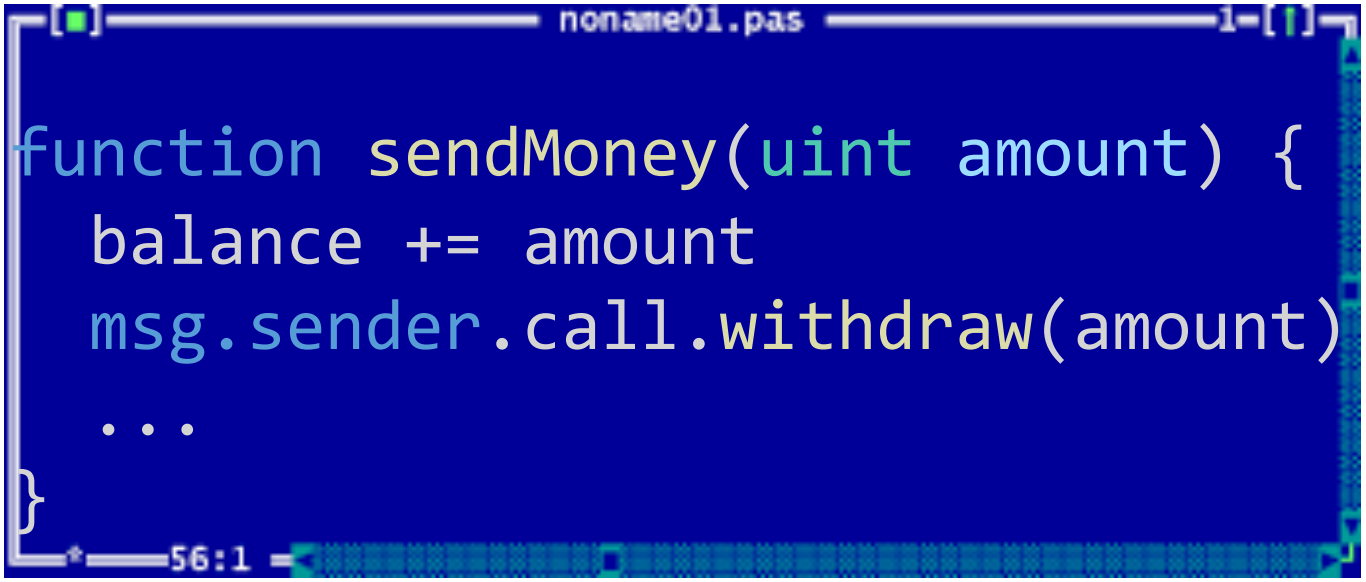
Transfer the money by calling a function in another contract …

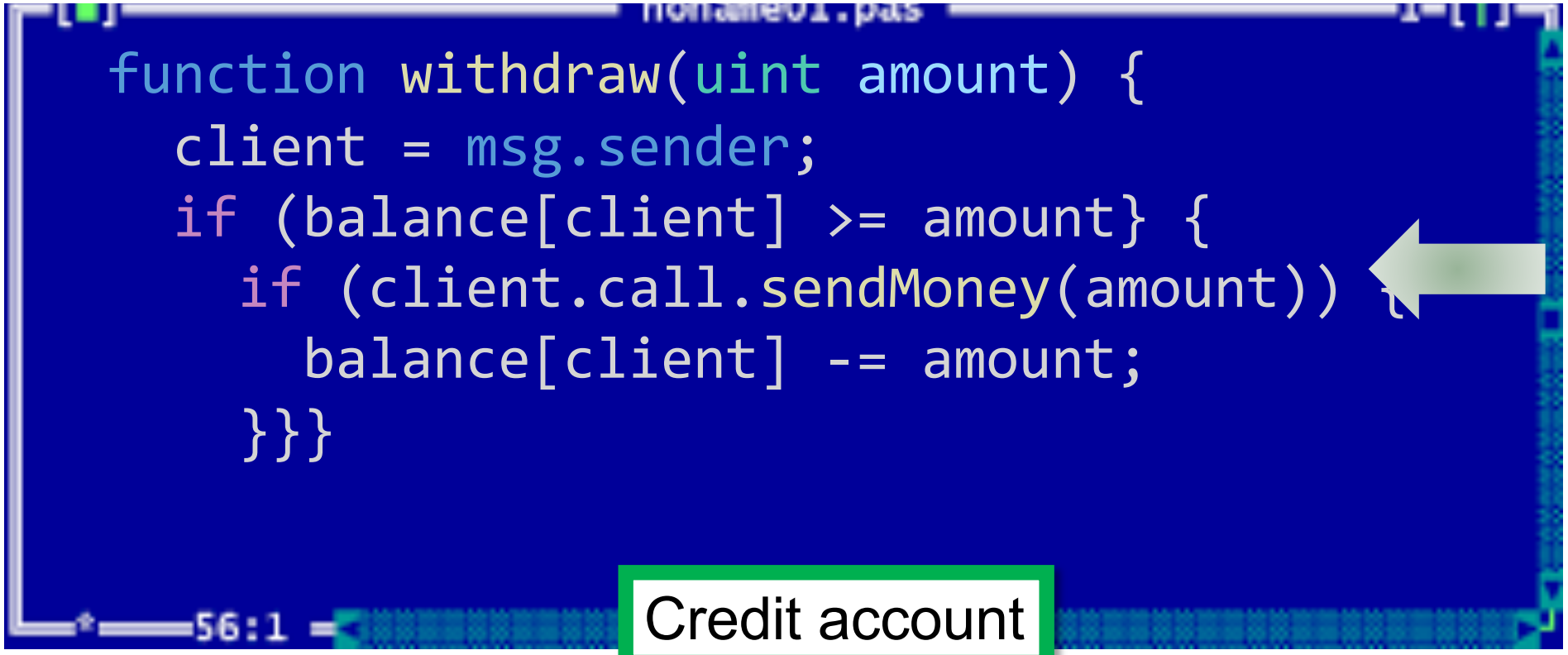
74


function
client
if (ba
if (
ba
}}}
56:1
CHEMINS DE FER
DE L'OUEST
721
Let's rewind …

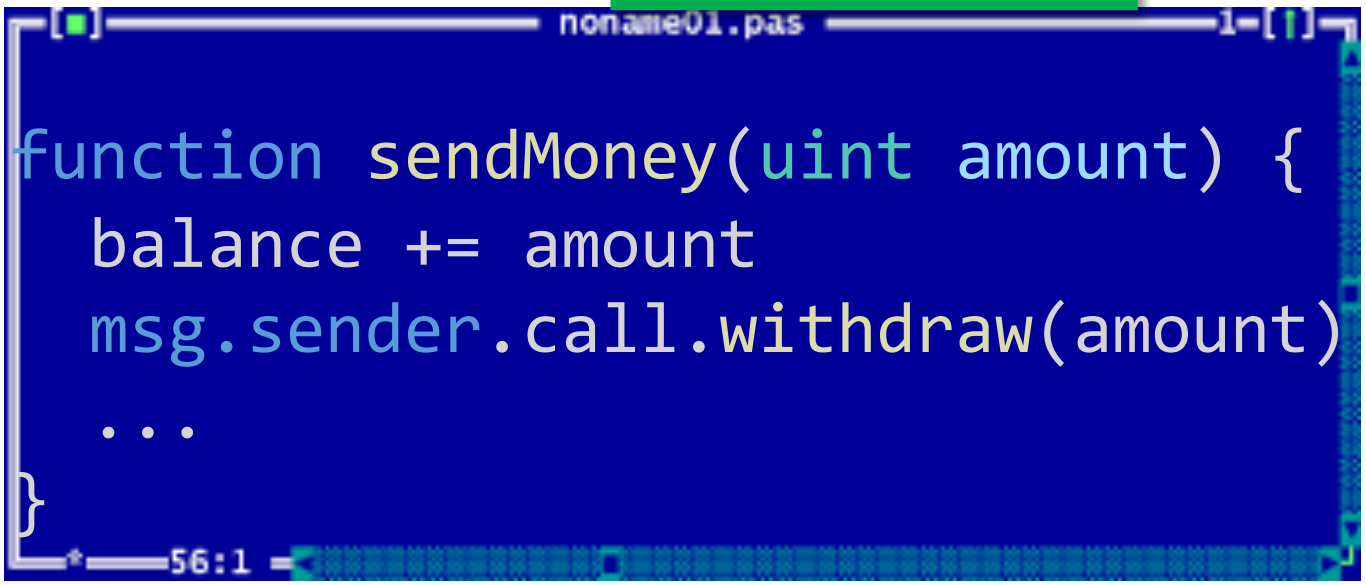
75


function withdraw(uint amount) {
client = msg.sender;
if (balance[client] >= amount} {
if (client.call.sendMoney(amount)) {
balance[client] -= amount;
}}}
Transfer the money by calling another contract …


noname01.pas
function sendMoney(uint amount) {
balance += amount
msg.sender.call.withdraw(amount)
...
}
56:1



Credit account

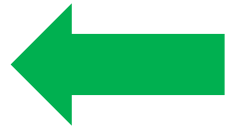


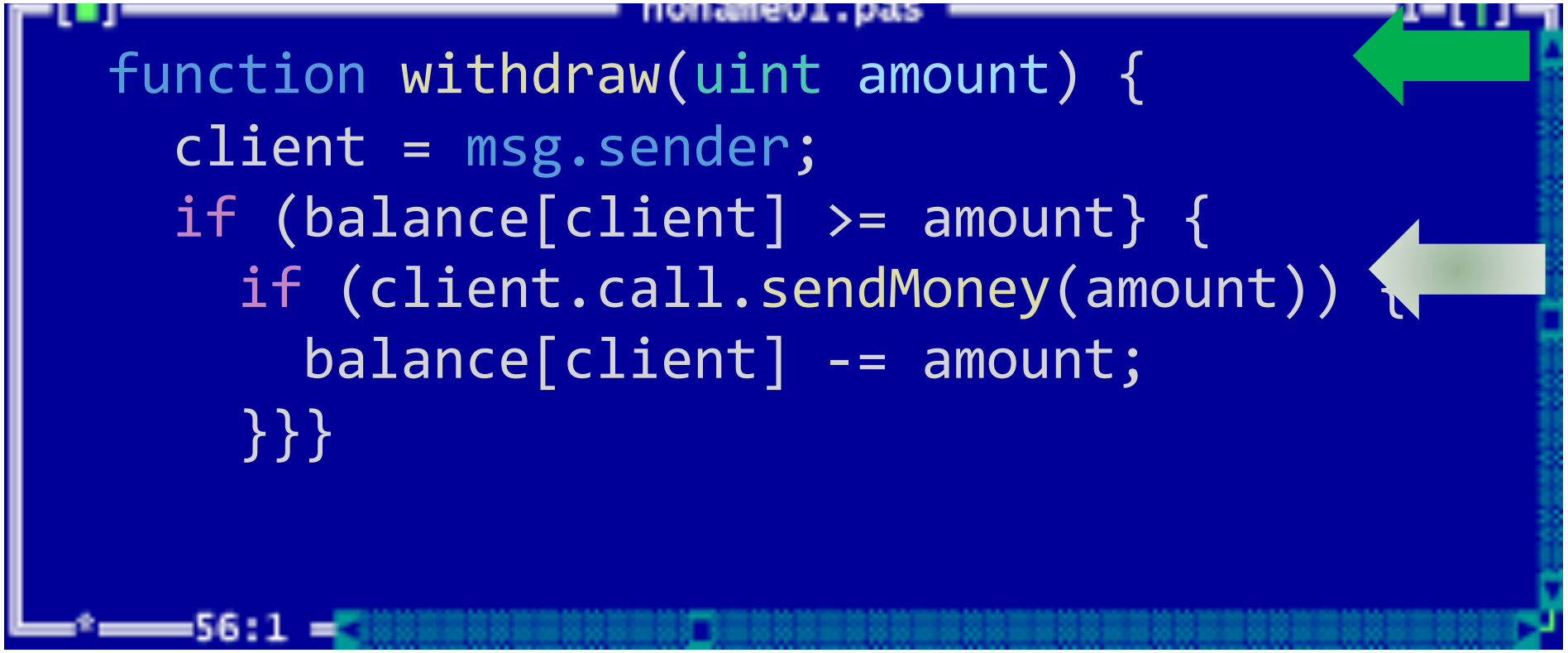
```
function withdraw(uint amount) {
  client = msg.sender;
  if (balance[client] >= amount} {
    if (client.call.sendMoney(amount)) {
      balance[client] -= amount;
    }}}
```

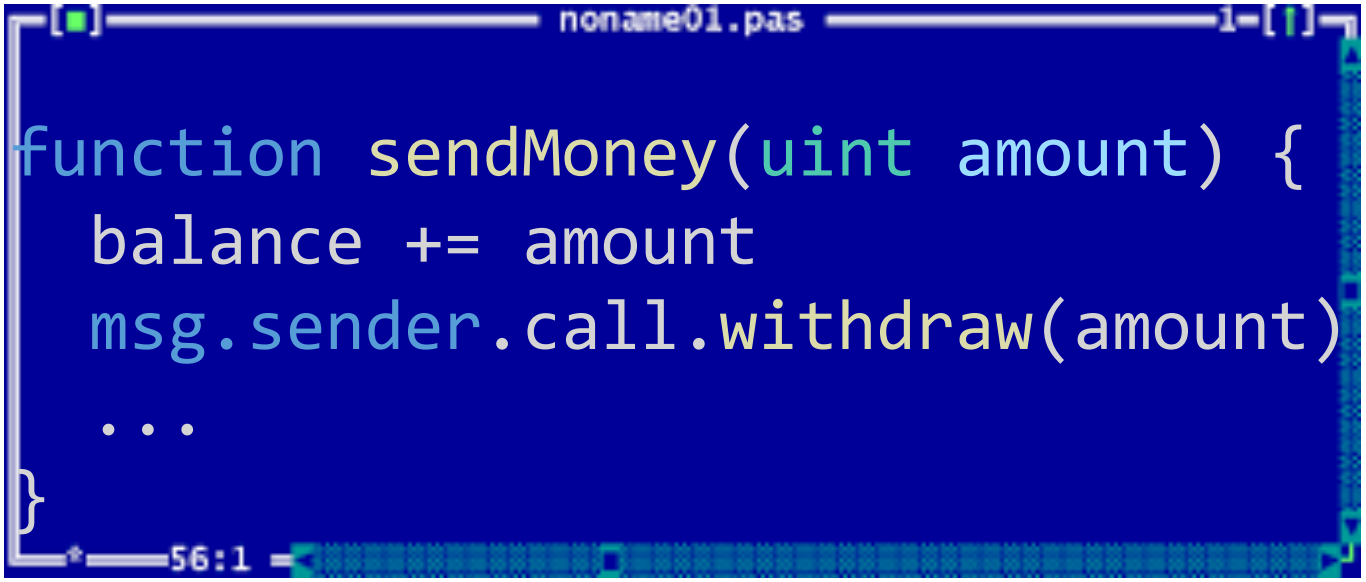
Wait, what?

Client makes “re-entrant” withdraw request!

```
function sendMoney(uint amount) {
  balance += amount
  msg.sender.call.withdraw(amount)
  ...
}
```




function withdraw(uint amount) {
client = msg.sender;
if (balance[client] >= amount} {
if (client.call.sendMoney(amount)) {
balance[client] -= amount;
}}}
56:1


noname01.pas
function sendMoney(uint amount) {
balance += amount
msg.sender.call.withdraw(amount)
...
}
56:1

```
function withdraw(uint amount) {
  client = msg.sender;
  if (balance[client] >= amount} {
    if (client.call.sendMoney(amount)) {
      balance[client] -= amount;
    }}}
```

Second time around, balance still looks OK …

```
function sendMoney(uint amount) {
  balance += amount
  msg.sender.call.withdraw(amount)
  ...
}
```

```
function withdraw(uint amount) {
  client = msg.sender;
  if (balance[client] >= amount} {
    if (client.call.sendMoney(amount))
      balance[client] -= amount;
    }}}
```

56:1

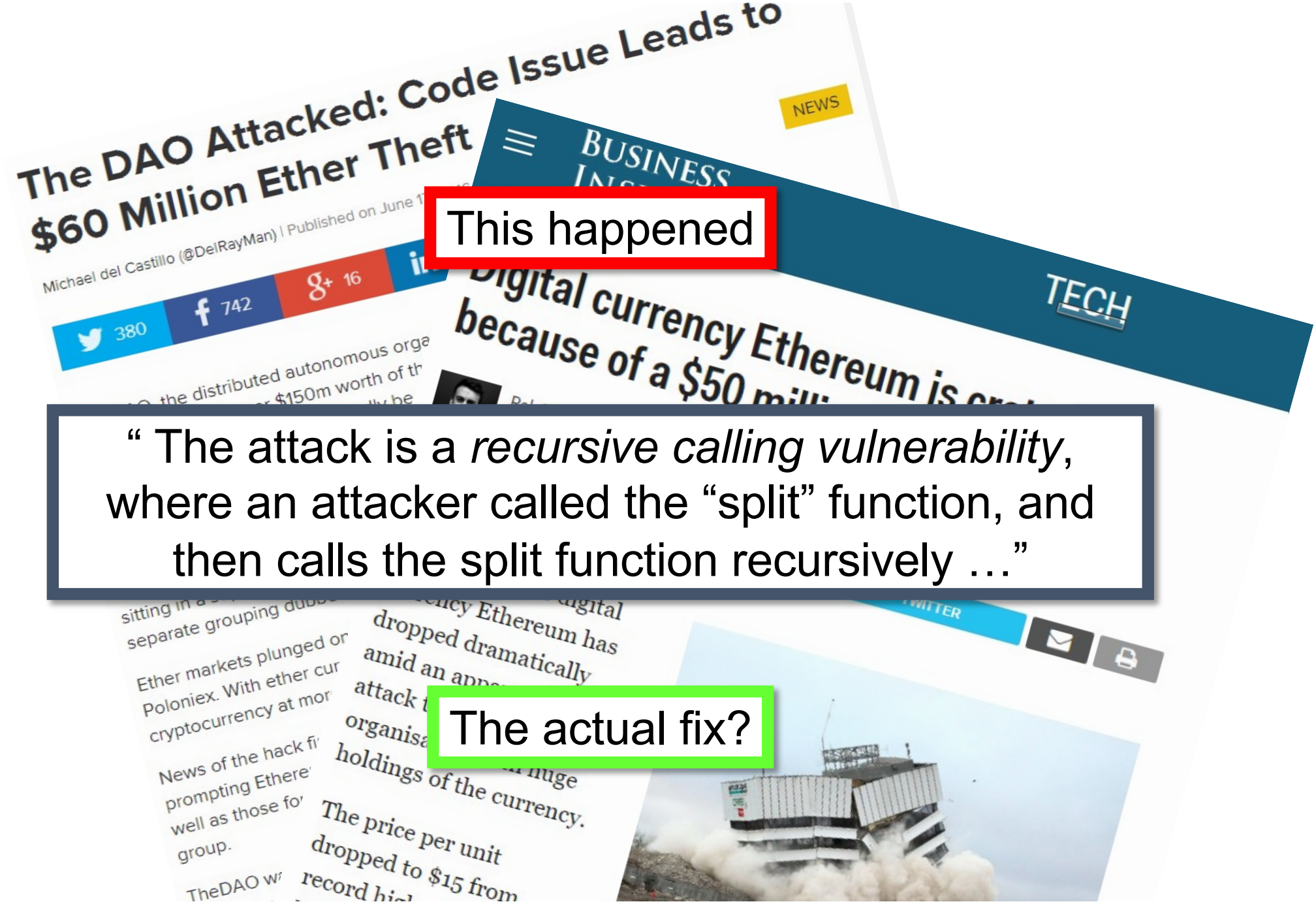
Send money again …

and again and again …

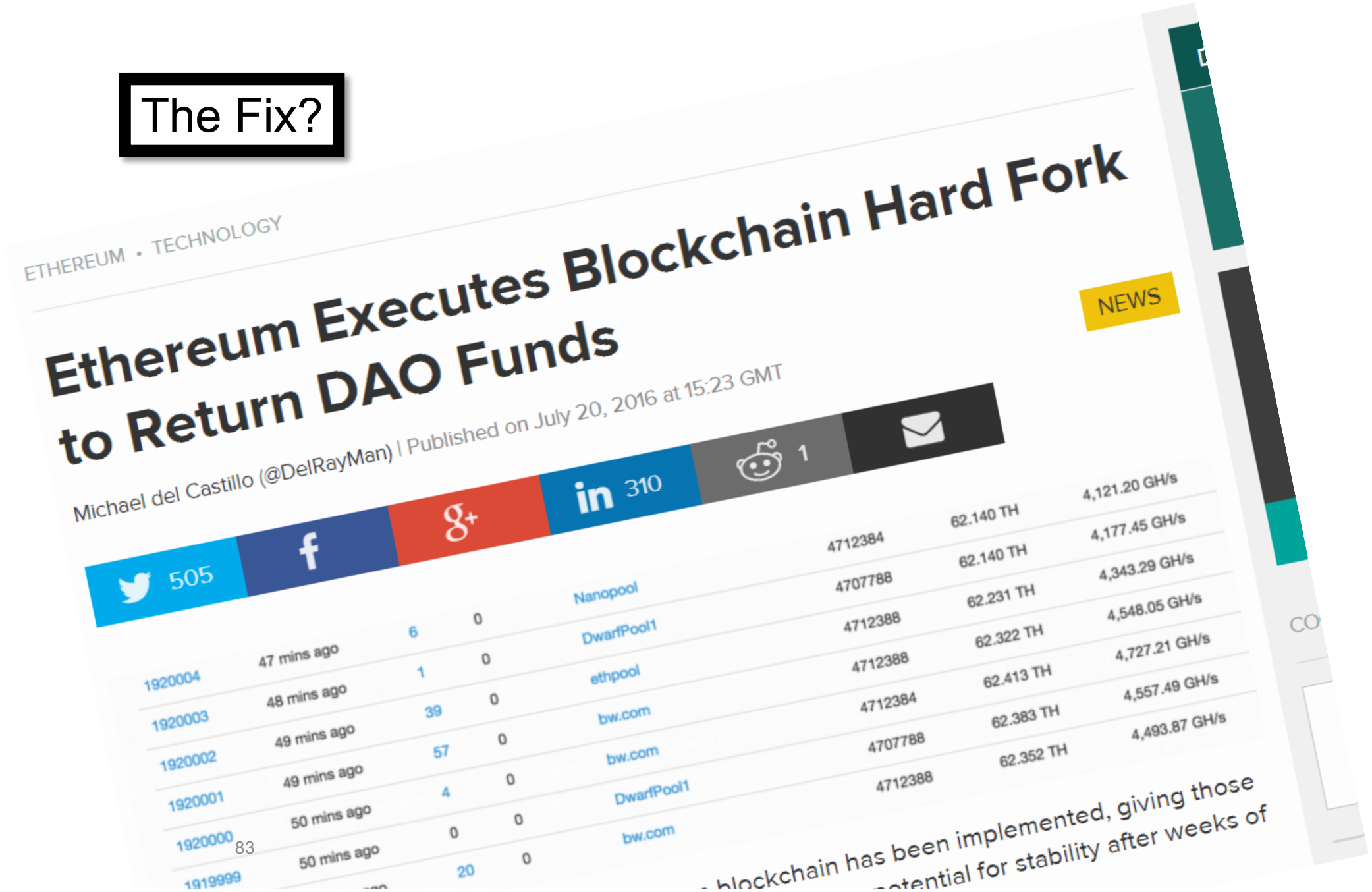
```
function sendMoney(uint amount) {
  balance += amount
  msg.sender.call.withdraw(amount)
  ...
}
```

81

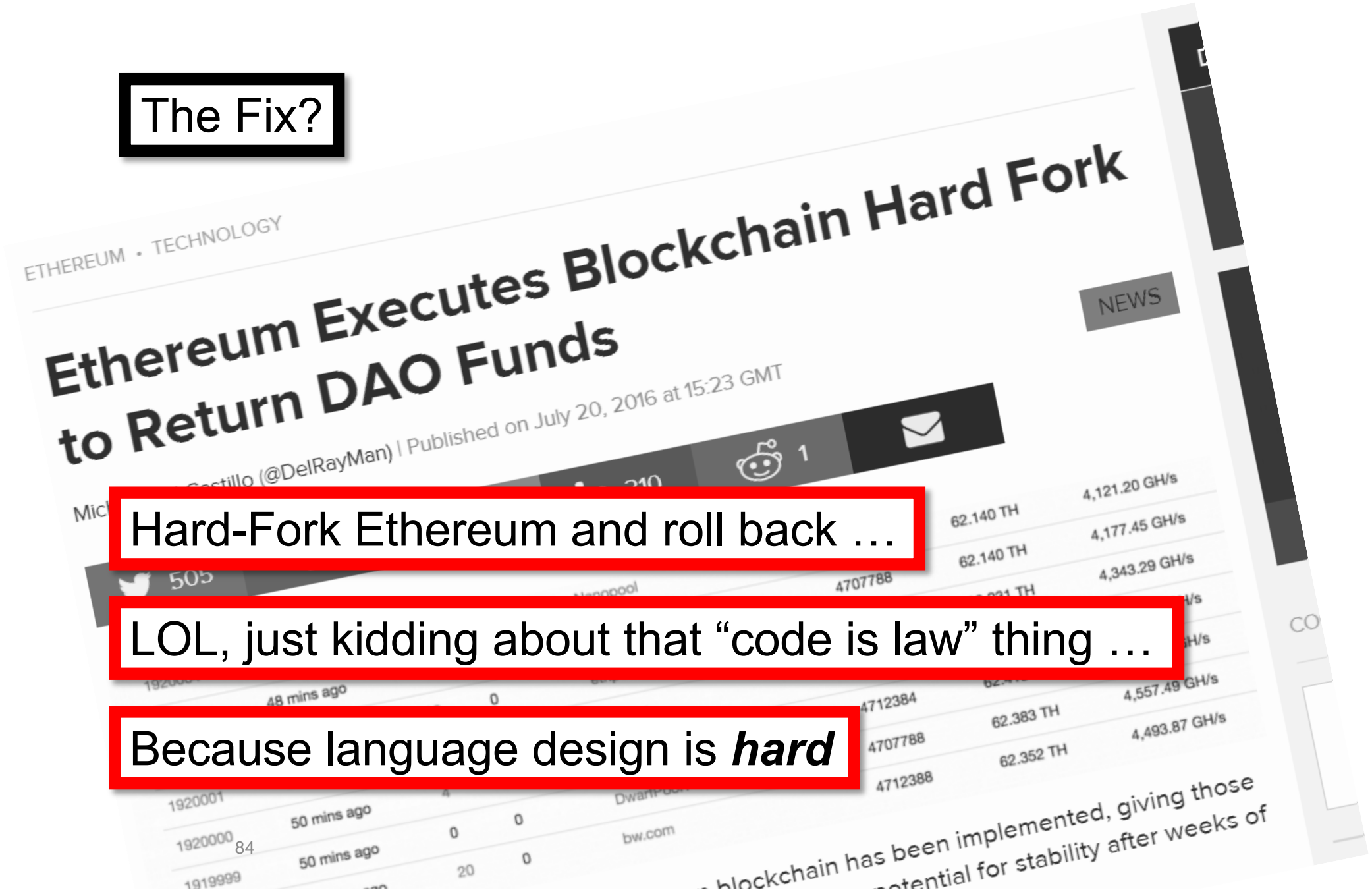
56:1


The DAO Attacked: Code Issue Leads to $60 Million Ether Theft
NEWS
Michael del Castillo (@DelRayMan) | Published on June
380
742
16
BUSINESS
TECH
This happened
Digital currency Ethereum is
because of a $50
" The attack is a *recursive calling vulnerability*, where an attacker called the "split" function, and then calls the split function recursively …"
separate grouping
Ether markets plunged or
Poloniex. With ether cur
cryptocurrency at mor
News of the hack fi
prompting Ethere
well as those fo
group.
TheDAO w
dropped dramatically
amid an app
attack
organisa
holdings of the currency.
The price per unit
dropped to $15 from
record hig
The actual fix?

The Fix?


ETHEREUM • TECHNOLOGY
Ethereum Executes Blockchain Hard Fork to Return DAO Funds
NEWS
Michael del Castillo (@DelRayMan) | Published on July 20, 2016 at 15:23 GMT
505
310
1
1920004 47 mins ago 6 0 Nanopool 4712384 62.140 TH 4,121.20 GH/s
1920003 48 mins ago 1 0 DwarfPool1 4707788 62.140 TH 4,177.45 GH/s
1920002 49 mins ago 39 0 ethpool 4712388 62.231 TH 4,343.29 GH/s
1920001 49 mins ago 57 0 bw.com 4712388 62.322 TH 4,548.05 GH/s
1920000 50 mins ago 4 0 bw.com 4712384 62.413 TH 4,727.21 GH/s
1919999 50 mins ago 0 0 DwarfPool1 4707788 62.383 TH 4,557.49 GH/s
20 0 bw.com 4712388 62.352 TH 4,493.87 GH/s
blockchain has been implemented, giving those
potential for stability after weeks of

The Fix?


ETHEREUM • TECHNOLOGY
Ethereum Executes Blockchain Hard Fork to Return DAO Funds
NEWS
Published on July 20, 2016 at 15:23 GMT
Hard-Fork Ethereum and roll back …
LOL, just kidding about that “code is law” thing …
Because language design is ***hard***
blockchain has been implemented, giving those

Rationale for Hard Fork

Client did something stupid?

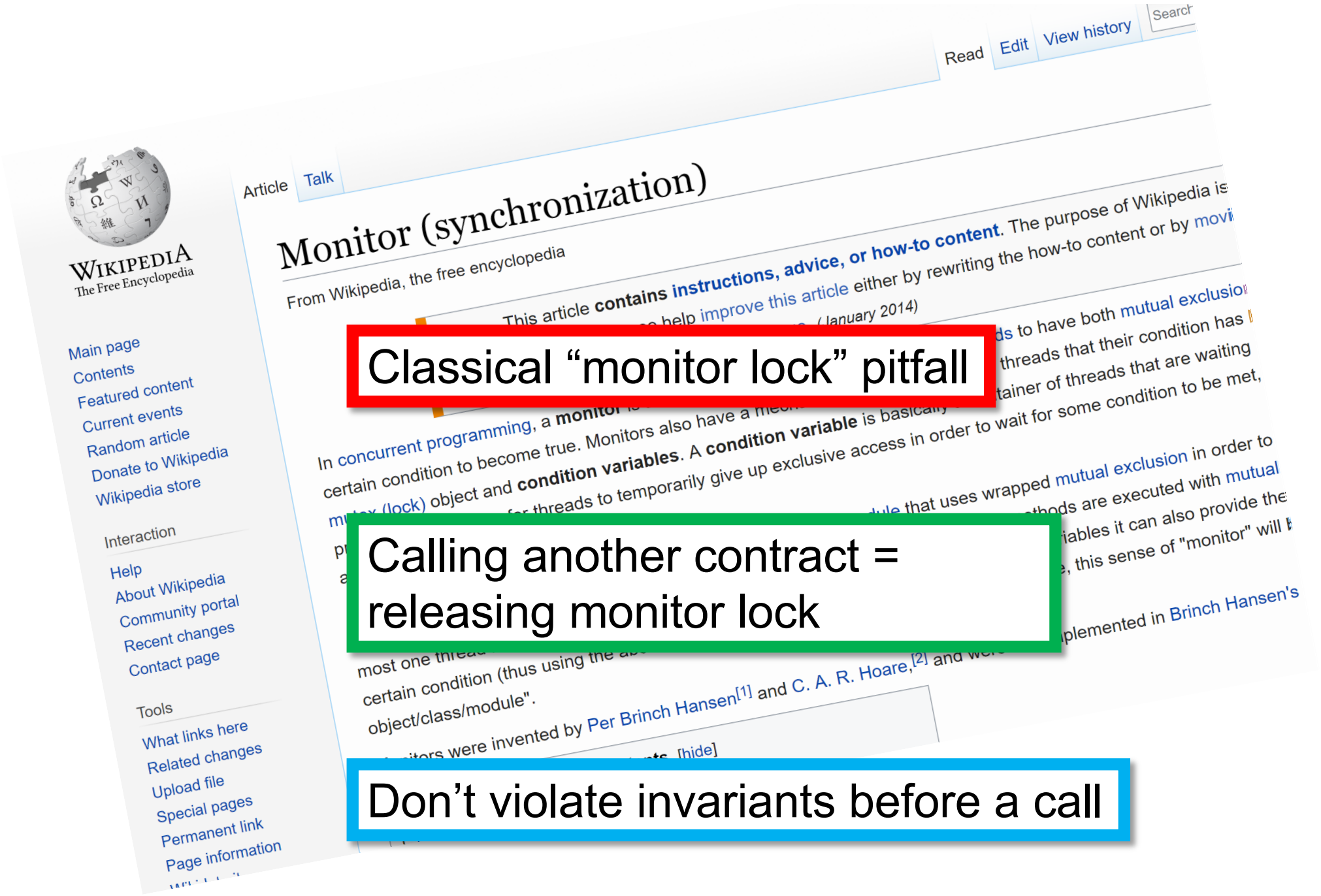
Client's fault, no refund.

Bug in Ethereum run-time system?

Ethereum's fault, refund.

No warning about reentrancy vulnerability?

Ethereum's fault, refund.

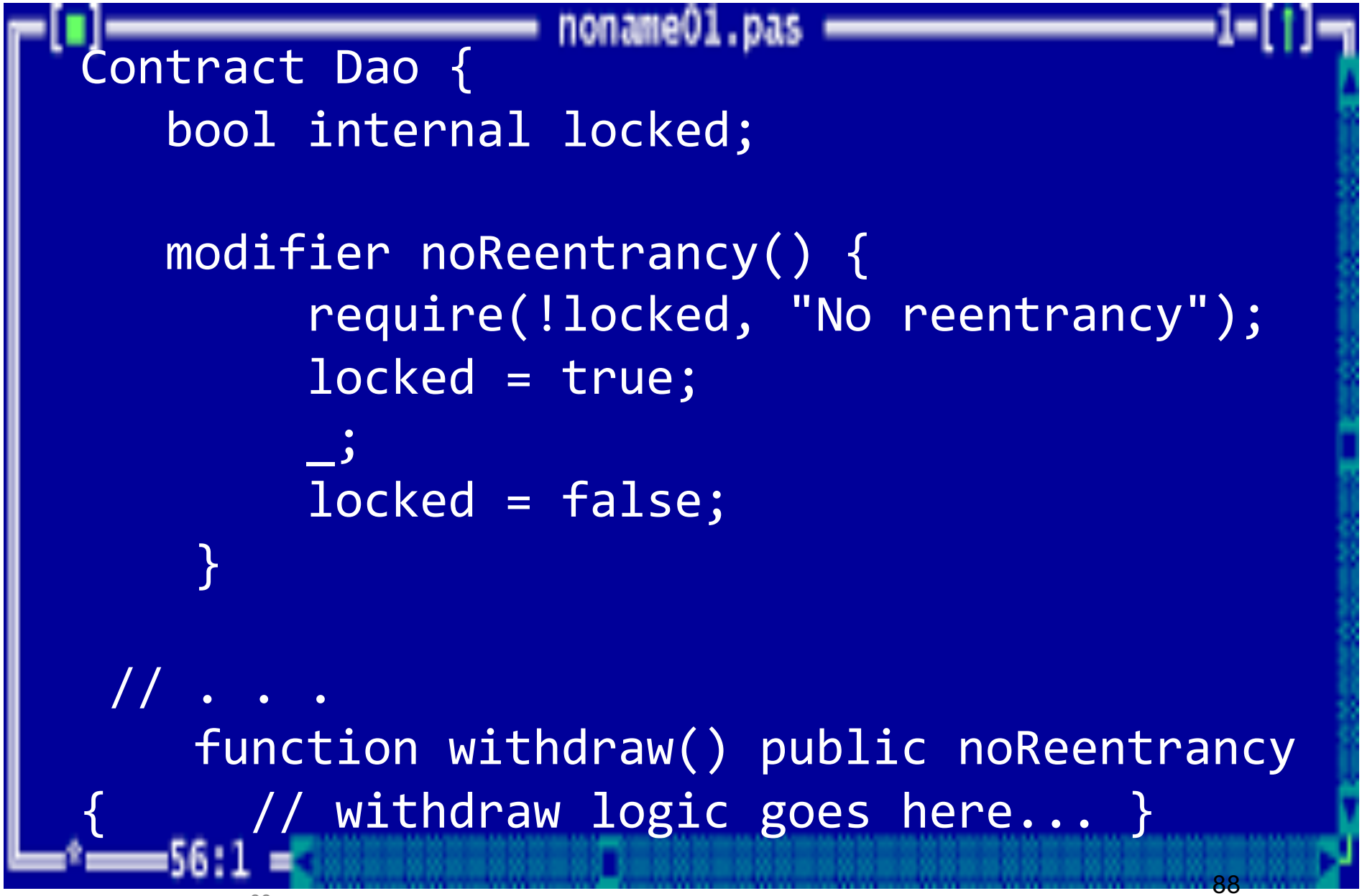

Monitor (synchronization)
Article
Talk
Read
Edit
View history
From Wikipedia, the free encyclopedia
Classical “monitor lock” pitfall
Calling another contract = releasing monitor lock
Don’t violate invariants before a call

Prevention

Change state *before* any external calls
("Checks-Effects-Interactions" pattern)

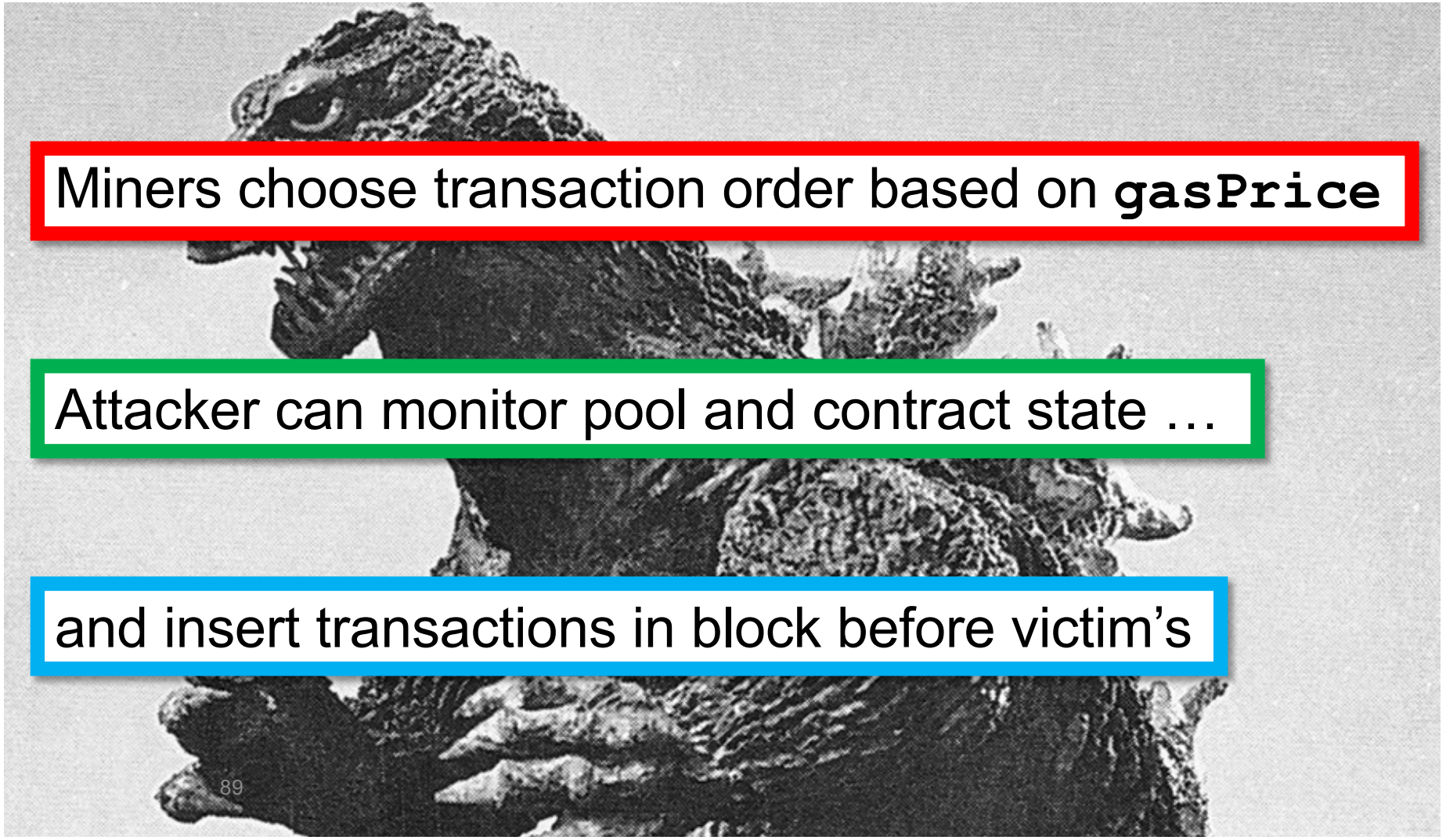
Use a mutex

Make the function non-reentrant


noname01.pas
Contract Dao {
bool internal locked;
modifier noReentrancy() {
require(!locked, "No reentrancy");
locked = true;
_;
locked = false;
}
// . . .
function withdraw() public noReentrancy
{ // withdraw logic goes here... }
56:1

88

Race Conditions



“Find This Hash” Game

```
contract FindThisHash {
  bytes32 constant public hash =
0xb5b5b97fafd9855eec9b41f74dfb6c38f59511
41f9a3ecd7f44d5479b630ee0a;
  constructor() public payable {}
  function solve(string solution)
   public {
    require(hash == sha3(solution));
    msg.sender.transfer(1000 ether);
 }
}
```

Load with ether

“Find This Hash” Game

```
contract FindThisHash {
  bytes32 constant public hash =
0xb5b5b9                                    511
41f9a3ecd7144d5479b630ee0a;
  constructor() public payable {}
  function solve(string solution)
   public {
    require(hash == sha3(solution));
    msg.sender.transfer(1000 ether);
  }
}
```

Earn 1000 ether if you can find preimage of hash

Alice realizes solution is “Ethereum!”

She calls
`FTHContract.solve("Ethereum!")`

Bob sees her transaction, validates answer, and resubmits with much higher **`gasPrice`**

Most likely, miners will order Bob’s transaction before Alice’s

Prevention

Contracts can put upper bound on **`gasPrice`**

Does not protect against abuse by miners

Prevention

Parties use commit-reveal scheme

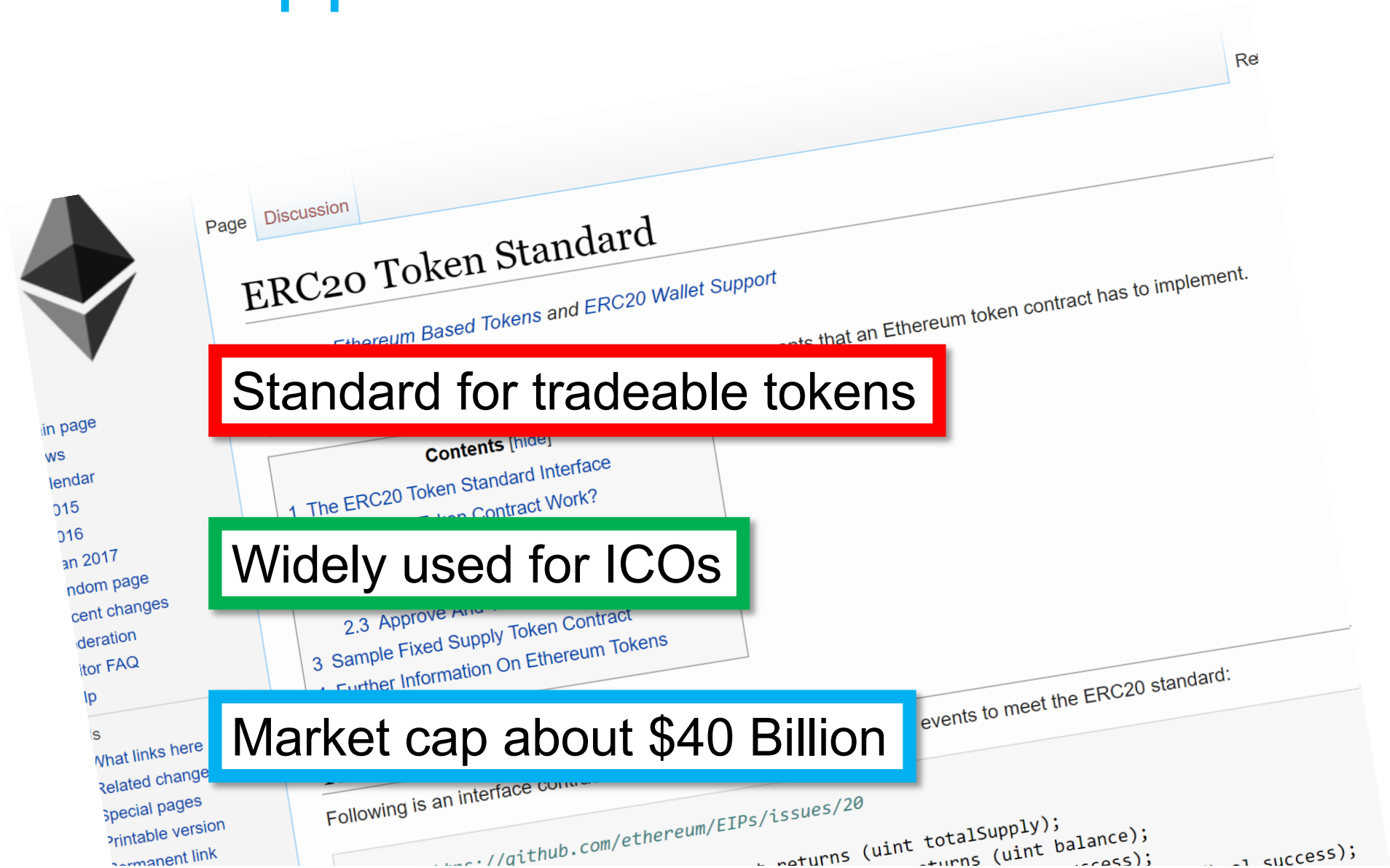
Commit to bid sending `hash(bid)`

After contract accepts commitment, reveal actual bid

Prevents both miners and users from front-running

Does not hide transaction value

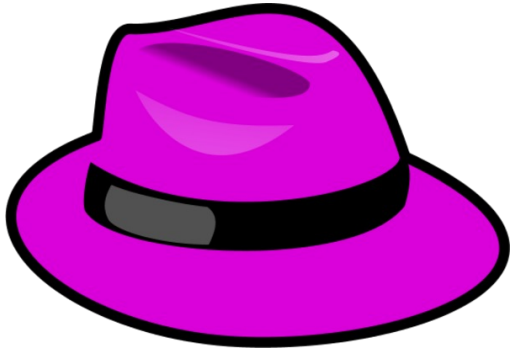
This Happened




I am willing to allow this party …
… to withdraw this amount …
}
function approve(address _spender, uint256 _value) returns (bool success) {
allowed[msg.sender][_spender] = _value;
return true;
}
function allowance(address _owner, address _spender) constant returns (uint256 remaining) {
… from my account.

Alice

Alice
authorizes
Bob
to withdraw
$100

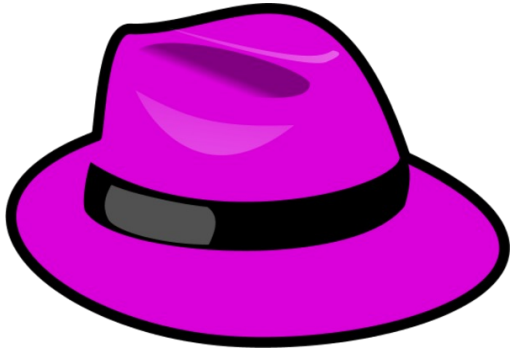


Bob

miner

Alice

Alice authorizes Bob to withdraw $100



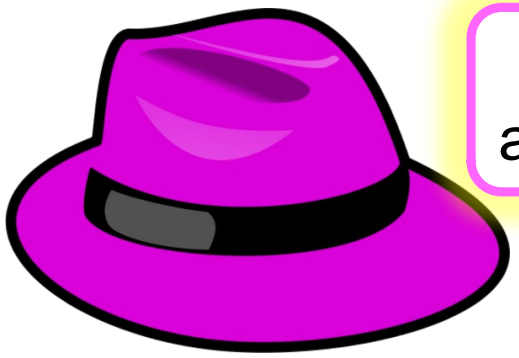
Bob

I am Bob's secret friend

miner

Alice

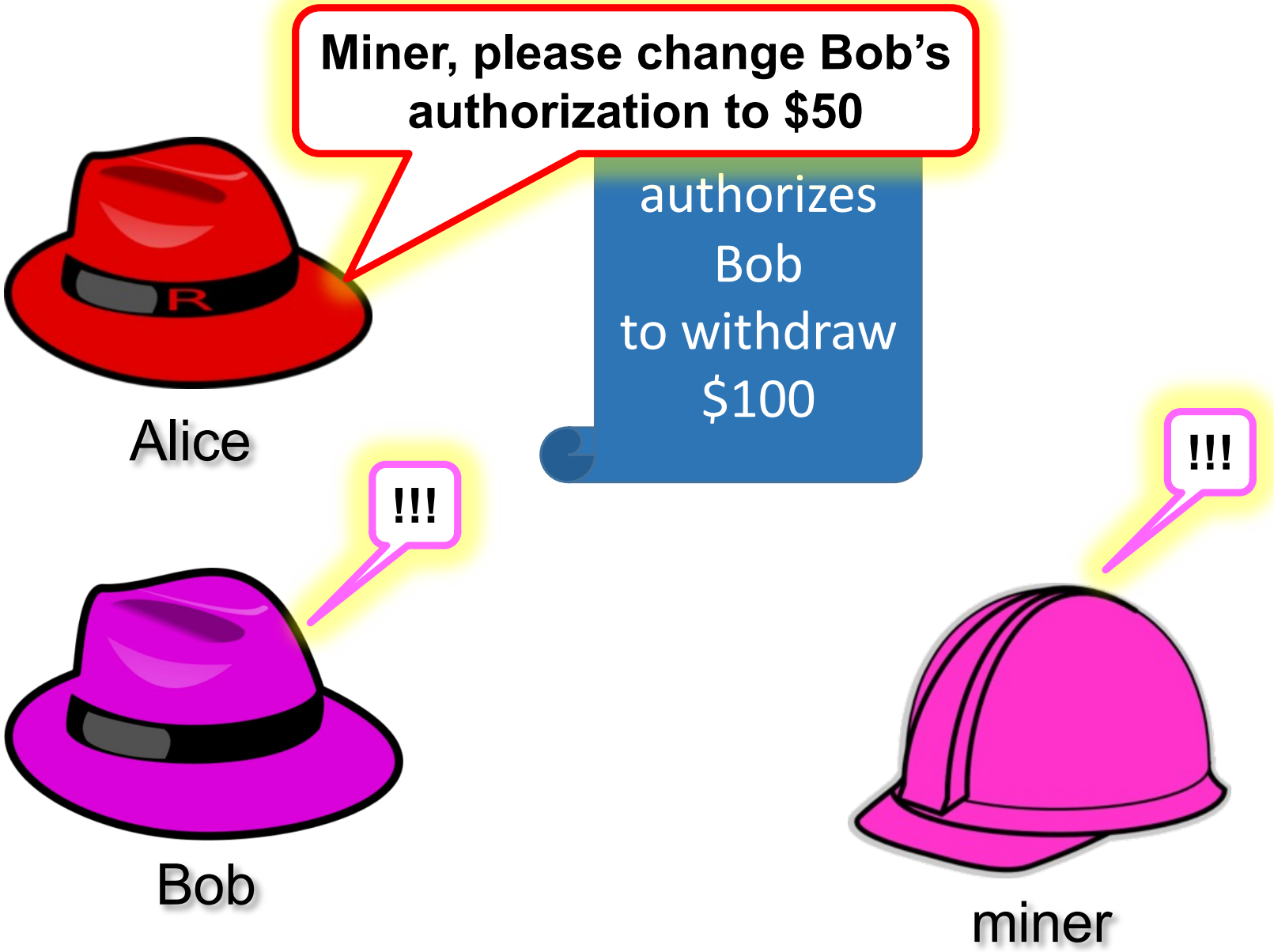
Alice
authorizes
Bob
to withdraw
$100

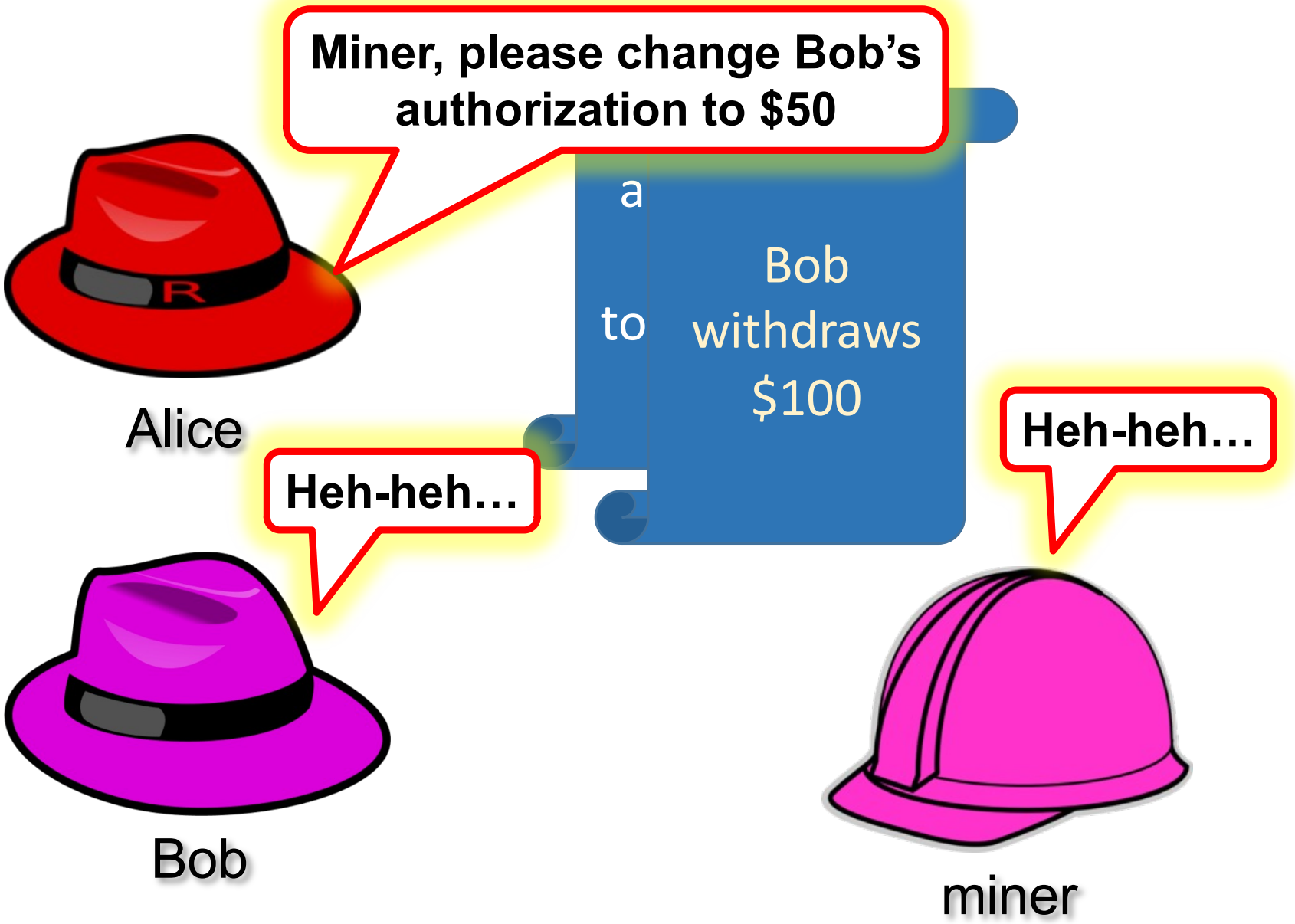


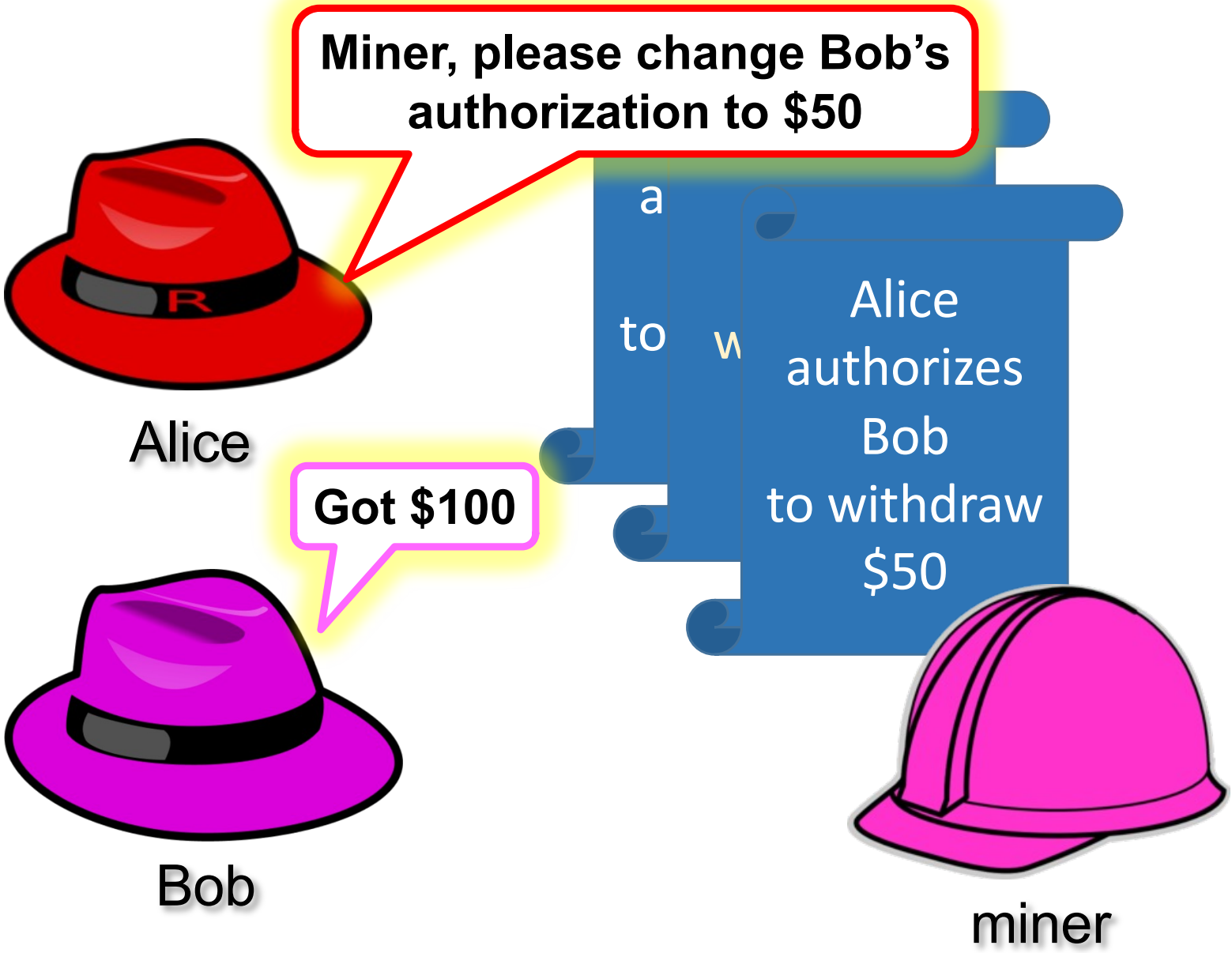
Bob

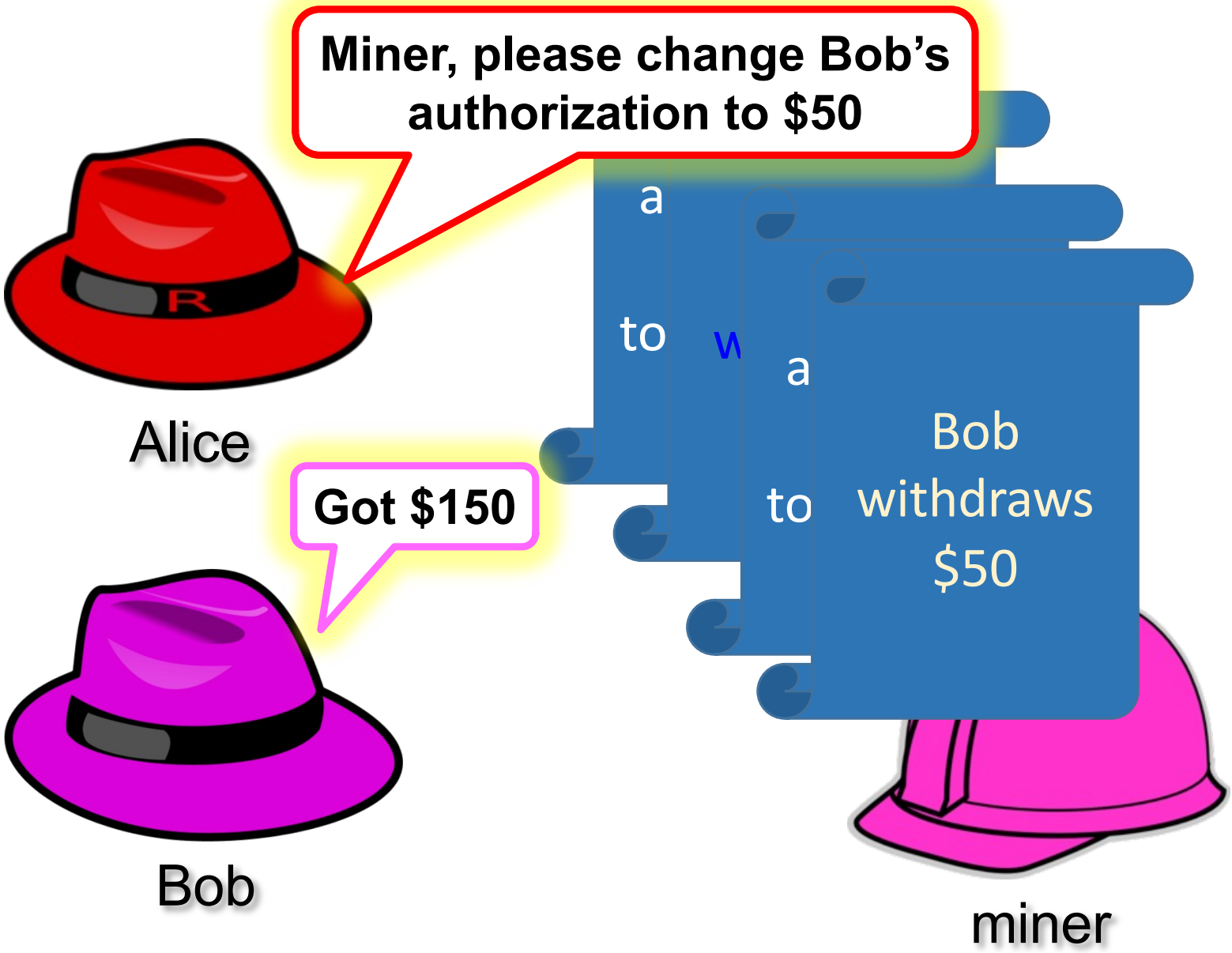
Think of me as an
adversarial scheduler

miner


Miner, please change Bob's authorization to $50
authorizes Bob to withdraw $100
Alice
!!!
Bob
!!!
miner


Miner, please change Bob's authorization to $50
a
to
Bob withdraws $100
Alice
Heh-heh…
Heh-heh…
Bob
miner


Miner, please change Bob's authorization to $50
Alice
Got $100
Bob
a
to
w
Alice authorizes Bob to withdraw $50
miner


Miner, please change Bob's authorization to $50
Alice
Got $150
Bob
a
to
a
to
Bob withdraws $50
miner

The problem

```
}

function approve(address _spender, uint256 _value) returns (bool success) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
}

function allowance(address _owner, address _spender) constant returns (uint256 remaining) {
```

Problem is overwriting
(instead of an atomic
Read-modify-write)

The fix

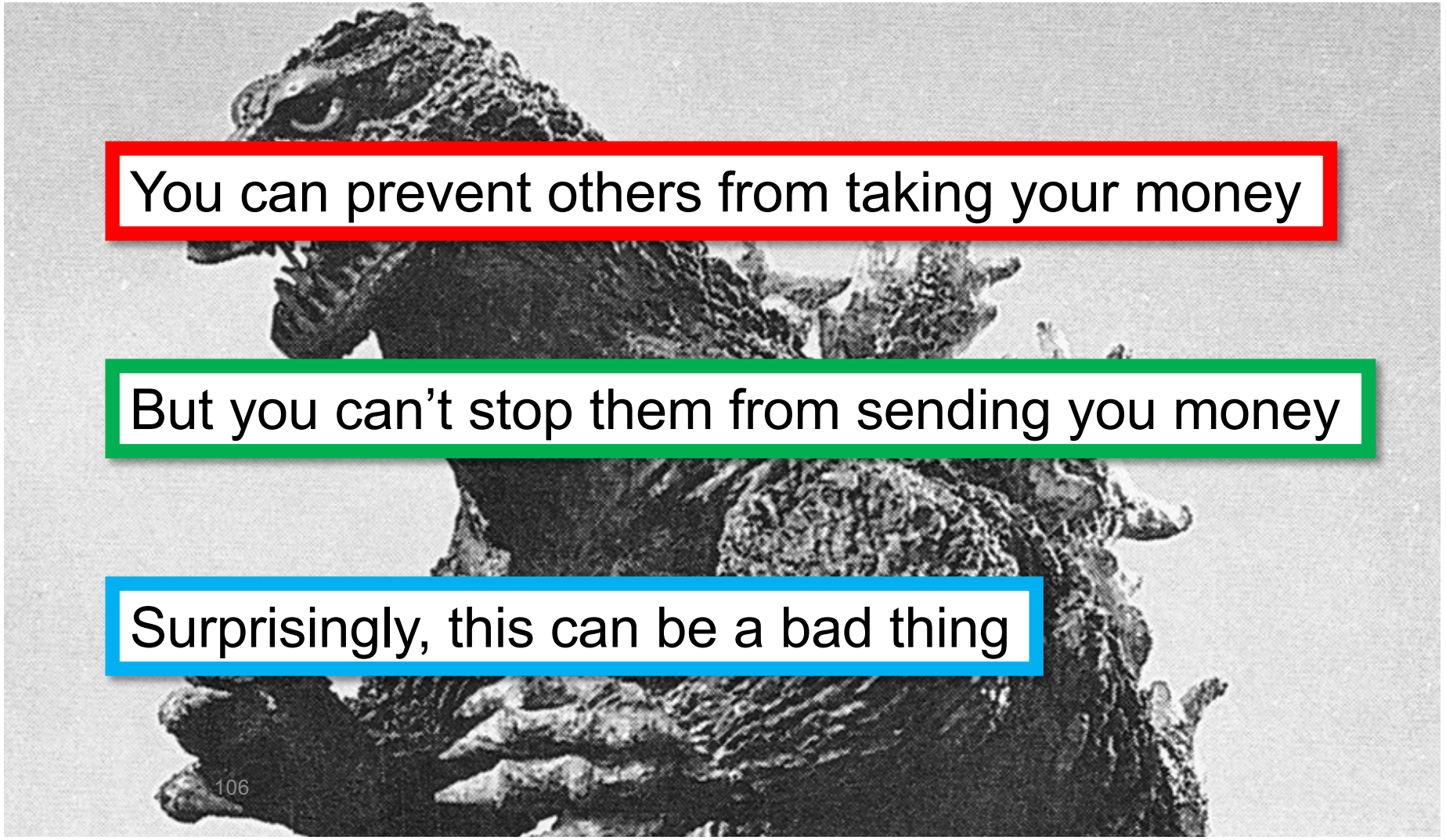
```
}

function approve(address _spender, uint256 _value) returns (bool success) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
}

fu                                                          nt256 remaining) {
```

(1) Could require approval to be: 100 → 0 → 50.
(2) Standard implementation of ERC20 created increaseAllowance() and decreaseAllowance() to prevent this

Unexpected Ether Attack



Sending Ether

```
destination.foo.value(1000)(arg)
```


```
contract destination {
  function foo(uint arg) public payable {
    …
  }
}
```

Sending Ether

```
destination.transfer(1000);
```


```
contract destination {
  fallback() external payable {
    // Code to execute when Ether is sent to the contract without a specific function call
  }
}
```

Controlling Ether Receipt

Ether receipt triggers …

Function, named or fallback

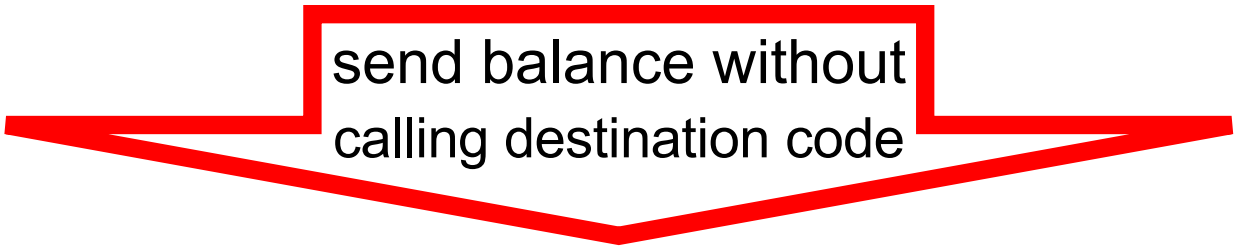
Possible to refuse ether by

reverting (throwing)

not being payable

Ether You Cannot Refuse

```
selfdestruct(destination);
```




```
contract destination {
        …
  }
```

can forcibly send Ethers to even the contract that does not implement the receive or fallback function

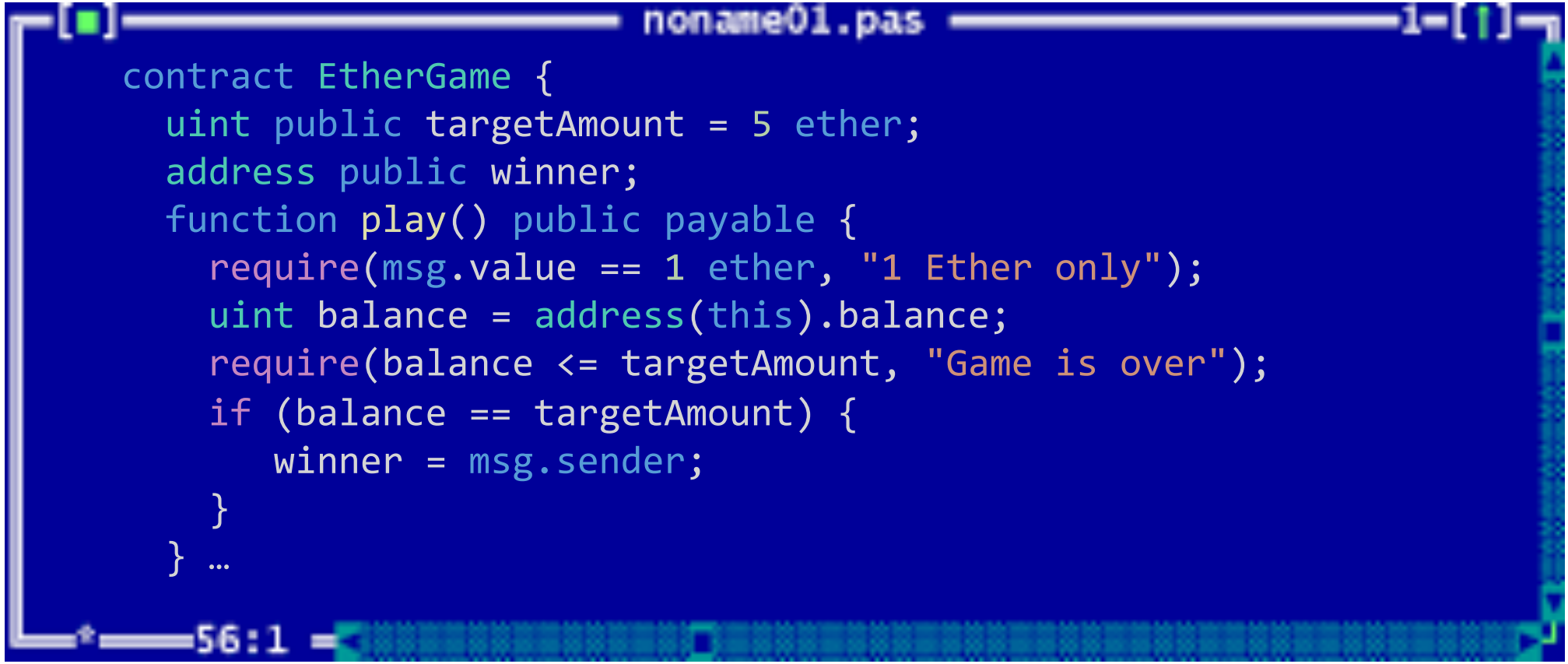
Example

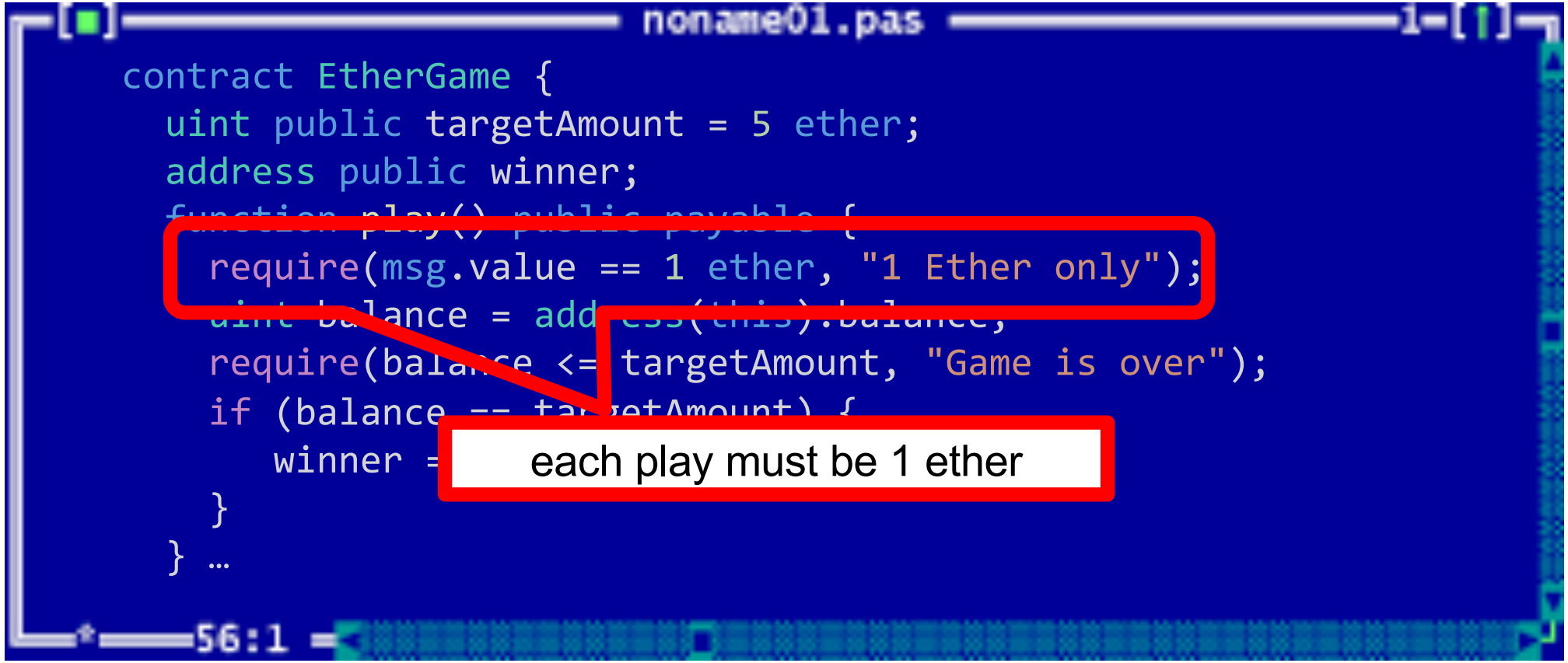
Simple gambling game

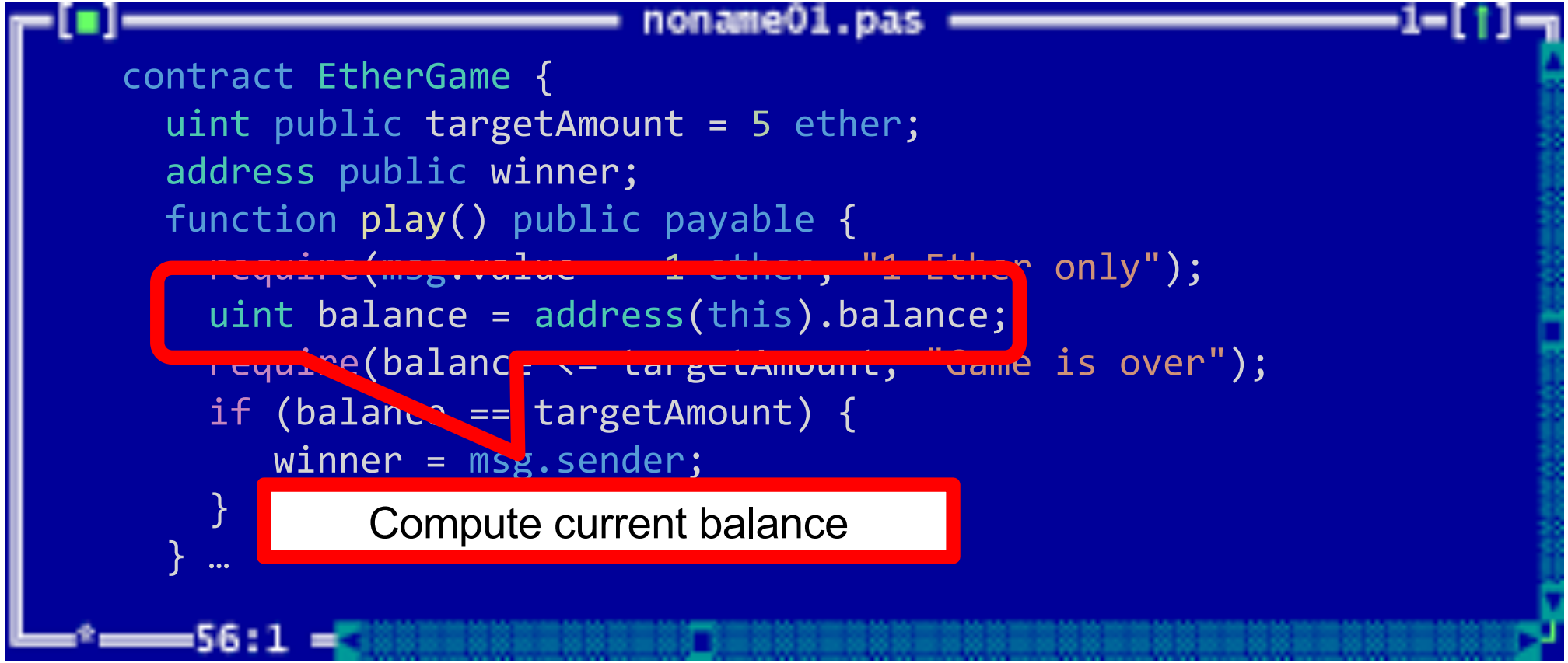
Sequence of milestones (amounts)

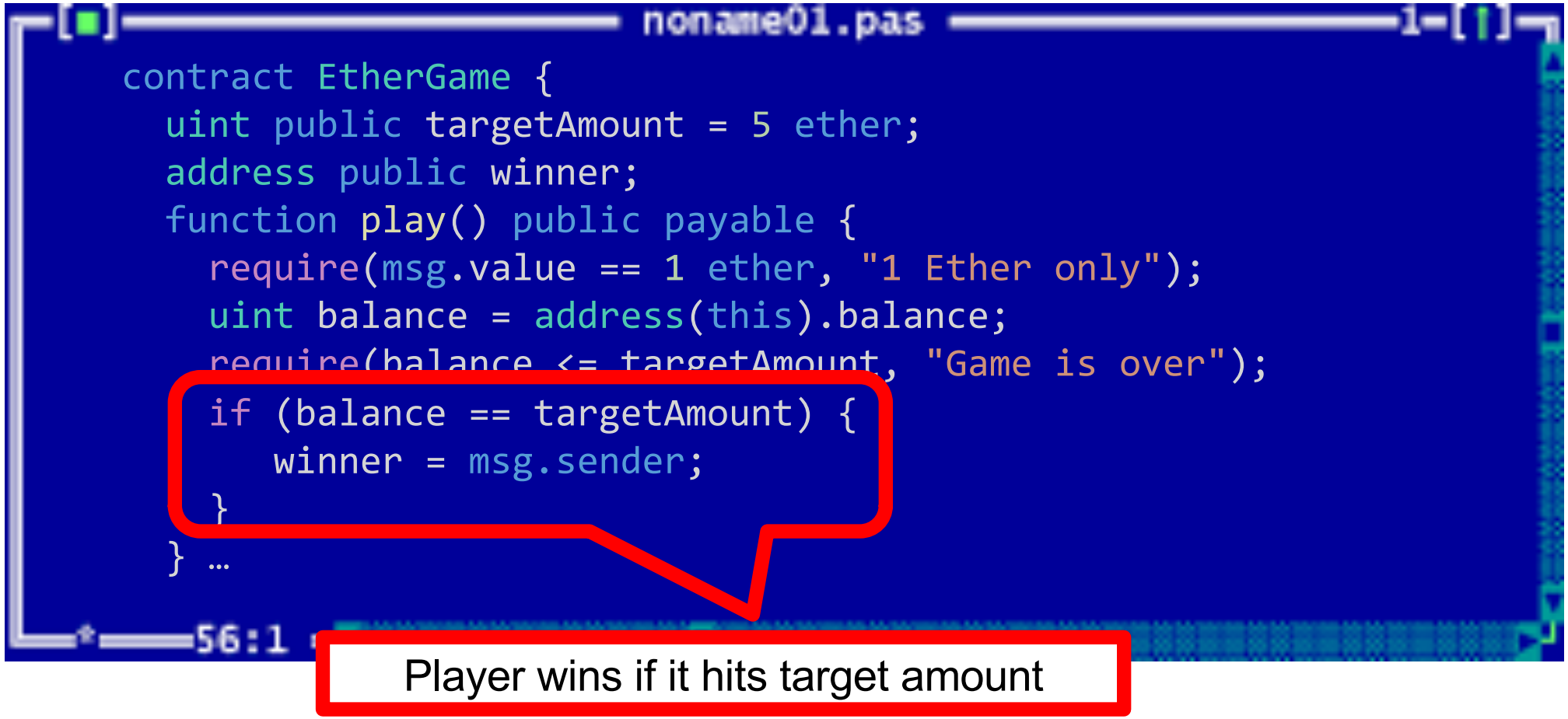
Players send small ether amounts …

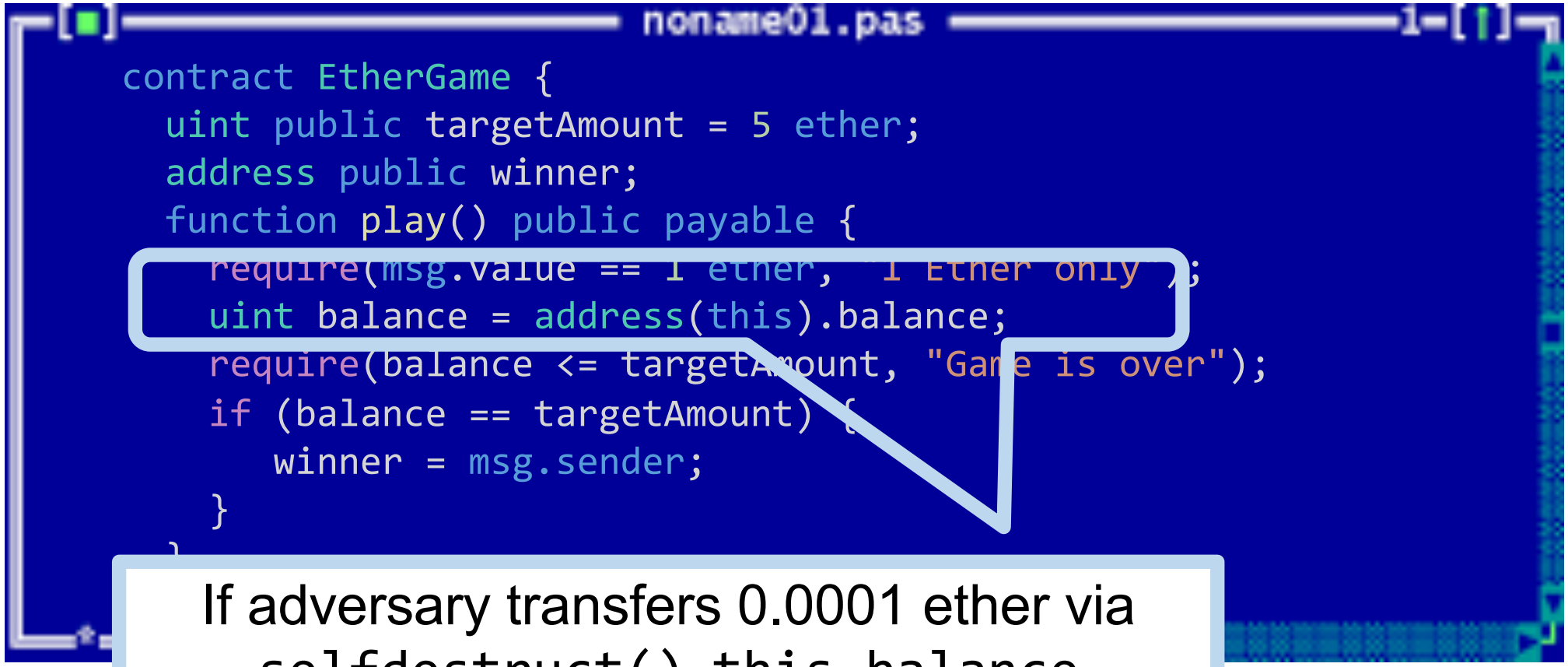
First player to reach each milestone wins!


noname01.pas
contract EtherGame {
uint public targetAmount = 5 ether;
address public winner;
function play() public payable {
require(msg.value == 1 ether, "1 Ether only");
uint balance = address(this).balance;
require(balance <= targetAmount, "Game is over");
if (balance == targetAmount) {
winner = msg.sender;
}
} …
56:1


noname01.pas
contract EtherGame {
uint public targetAmount = 5 ether;
address public winner;
require(msg.value == 1 ether, "1 Ether only");
require(balance <= targetAmount, "Game is over");
if (balance
winner =
each play must be 1 ether
} …
56:1


noname01.pas
contract EtherGame {
uint public targetAmount = 5 ether;
address public winner;
function play() public payable {
uint balance = address(this).balance;
if (balance == targetAmount) {
winner = msg.sender;
}
} …
Compute current balance
56:1


noname01.pas
contract EtherGame {
uint public targetAmount = 5 ether;
address public winner;
function play() public payable {
require(msg.value == 1 ether, "1 Ether only");
uint balance = address(this).balance;
require(balance <= targetAmount, "Game is over");
if (balance == targetAmount) {
winner = msg.sender;
}
} …
56:1
Player wins if it hits target amount



If adversary transfers 0.0001 ether via `selfdestruct()`, `this.balance` becomes slightly off, & comparison fails!

The attack

1. Deploy the Attack contract specifying the EtherGame contract address as the contract deployment argument
2. Supply some Ethers for attacking and invoke the Attack.attack() function

```
contract Attack {
    address immutable eGame;

    constructor(address _eGame) {
        eGame = _eGame;
    }

    function attack() external payable {
        require(msg.value != 0, ”Need some Ether to attack");

        address payable target = payable(eToken);
        selfdestruct(target);
    }
}
```

56:1

Prevention

Never rely on exact value of `this.balance`

`Replace == with >= or range [5,6[`

Recommended reading

https://docs.soliditylang.org/en/v0.8.19/introduction-to-smart-contracts.html
https://docs.soliditylang.org/en/v0.8.19/solidity-by-example.html
https://blog.chain.link/reentrancy-attacks-and-the-dao-hack/
https://medium.com/valixconsulting/solidity-smart-contract-security-by-example-08-unexpected-ether-with-forcibly-sending-ether-e13be2c6b985
https://www.adrianhetman.com/unboxing-erc20-approve-issues/

Further reading (comprehensive list of attacks)

https://blog.sigmaprime.io/solidity-security.html

Acknowledgements

Slides adapted from Maurice Herlihy, Brown University, available under CC BY-NC 4.0 (https://creativecommons.org/licenses/by-nc/4.0/)