Solidity

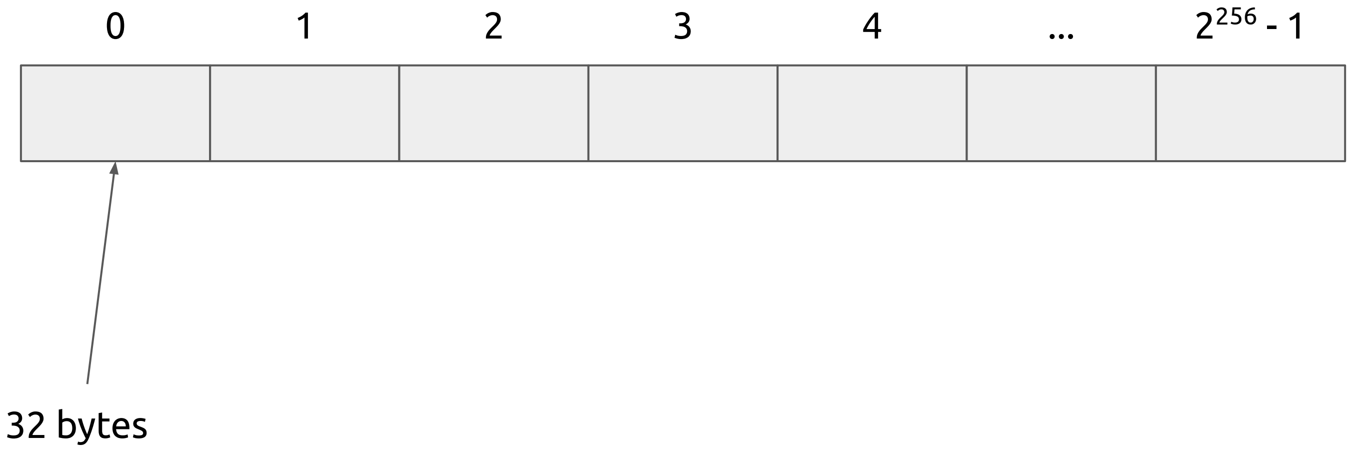
Aggelos Kiayias

Dionysis Zindros, Christos Nasikas

Overview

- Storage layout
- Security of Smart Contracts
- Known attacks
- Best practices

Storage Layout


0
1
2
3
4
...
$2^{256} - 1$
32 bytes

Storage Layout: Fixed-Sized Values

```
contract Storage {

     uint256 x;

     uint256[2] y;

}
```

Address (slot)	Contents
0	x
1	y
2	

Storage Layout: Fixed-Sized Values

```
contract Storage {
	uint256 x;
	uint256[2] y;

	struct Row {
		uint256 id;
		uint256 value;
	}

	Row z;
}
```

Address (slot)	Contents
0	x
1	y
2	
3	z
4	

Adapted and modified from: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

Storage Layout: Dynamically-Sized Arrays (1D)

```
contract Storage {
     uint256[] x;
}
```

Address (slot)	Contents
0	x.length
...	...
keccak256(0)	x[0]
keccak256(0) + 1	x[1]
...	....

Adapted and modified from: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

Storage Layout: Dynamically-Sized Arrays (2D)

```
contract Storage {
    uint256[][] y;
}
```

Address (slot)	Contents
0	y.length
...	....
keccak256(0)	y[0].length
keccak256(0) + 1	y[1].length
...	...

Address (slot)	Contents
keccak256(keccak256(0))	y[0][0]
keccak256(keccak256(0)) + 1	y[0][1]
...	...
keccak256(keccak256(0) + 1)	y[1][0]
keccak256(keccak256(0) + 1) + 1	y[1][1]

Adapted and modified from: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA '18.

Storage Layout: Calculate slot

```
function get1DArrayPos(uint256 slot, uint256 index, uint256 size)
    public
    pure
    returns (uint256) {
    return uint256(keccak256(slot)) + (index * size);
}
```

Storage Layout: Mappings

```
contract Storage {
    mapping (address => uint) balances;
}
```

Address (slot)	Contents
0	empty
...	...
keccak256(0xc5, 0)	balances[0xc5]
...	....
keccak256(0xfa, 0)	balances[0xfa]

Adapted and modified from: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

Storage Layout: Calculate slot

```
function getMapPos(uint256 key, uint256 slot) public pure returns
(uint256) {
    return uint256(keccak256(key, slot));
}
```

Security of Smart Contracts

Know Attacks

DoS

DoS: Unbounded operation

```
// INSECURE
contract NaiveBank {
  struct Account {
      address addr;
      uint balance;
  }

  Account accounts[];
  function applyInterest() returns (uint) {
      for (uint i = 0; i < accounts.length; i++) {
            // apply 5 percent interest
            accounts[i].balance = accounts[i].balance * 105 / 100;
      }
      return accounts.length;
  }

  function openAccount() returns (uint) { … }
}
```

Source: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

DoS: Unbounded operation

```
// INSECURE
contract NaiveBank {
  struct Account {
      address addr;
      uint balance;
  }

  Account accounts[];
  function applyInterest() returns (uint) {
      for (uint i = 0; i < accounts.length; i++) {
            // apply 5 percent interest
            accounts[i].balance = accounts[i].balance * 105 / 100;
      }
      return accounts.length;
  }

  function openAccount() public returns (uint) { … }
}
```

Source: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

DoS: Wallet Griefing

```
// INSECURE
for (uint i = 0; i < investors.length; i++) {
  if (investors[i].invested == min_investment) {

    if (!(investors[i].addr.send(investors[i].dividendAmount))) {
        revert();
    }

    investors[i] = newInvestor;
  }
}
```

Source: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

DoS: Wallet Griefing

```
// INSECURE
for (uint i = 0; i < investors.length; i++) {
  if (investors[i].invested == min_investment) {

    if (!(investors[i].addr.send(investors[i].dividendAmount))) {
        revert();
    }

    investors[i] = newInvestor;
  }
}
```

Source: MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA ‘18.

Forcibly Sending Ether to a Contract

Forcibly Sending Ether to a Contract

- Exploits **misuse** of `this.balance`
- How can you **send ether** to a contract **without** firing contact’s **fallback** function ?
    - `selfdestruct(victim)`
    - Anyone can **calculate** a contract’s address **before** it is **created** (contract addresses generation is **deterministic**) and sent ether to that address.

Reentrancy

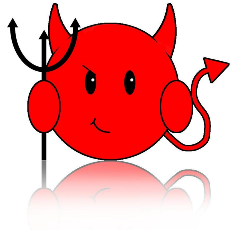
Reentrancy

Fallback function

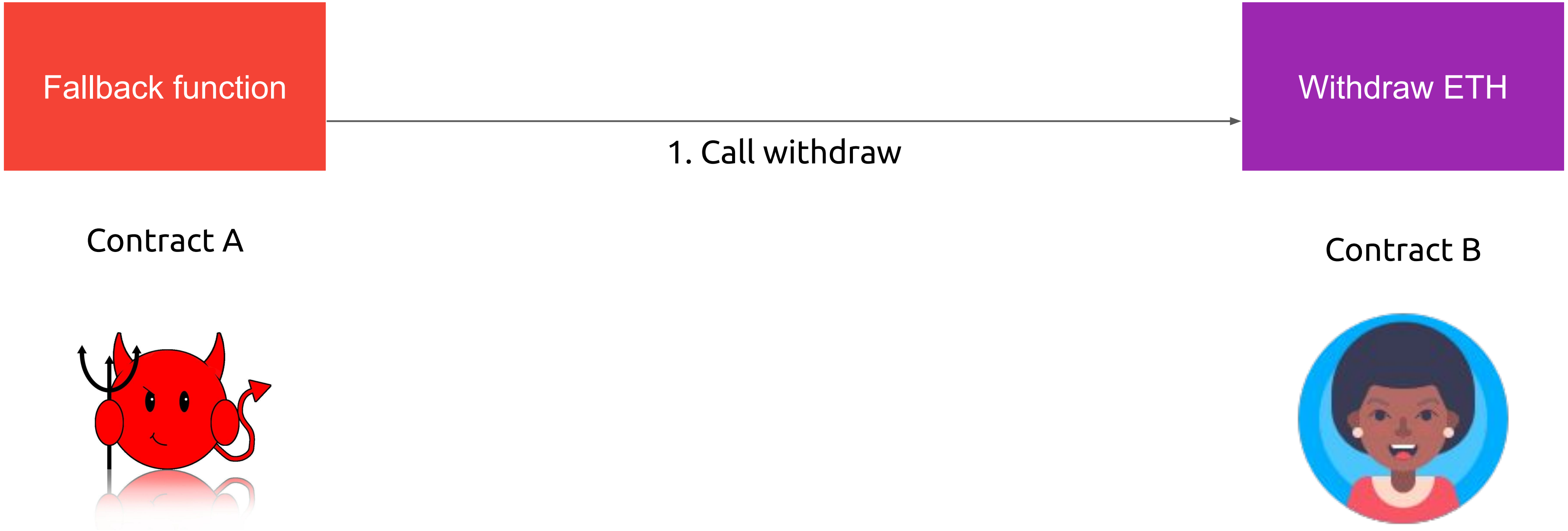
Withdraw ETH

Contract A

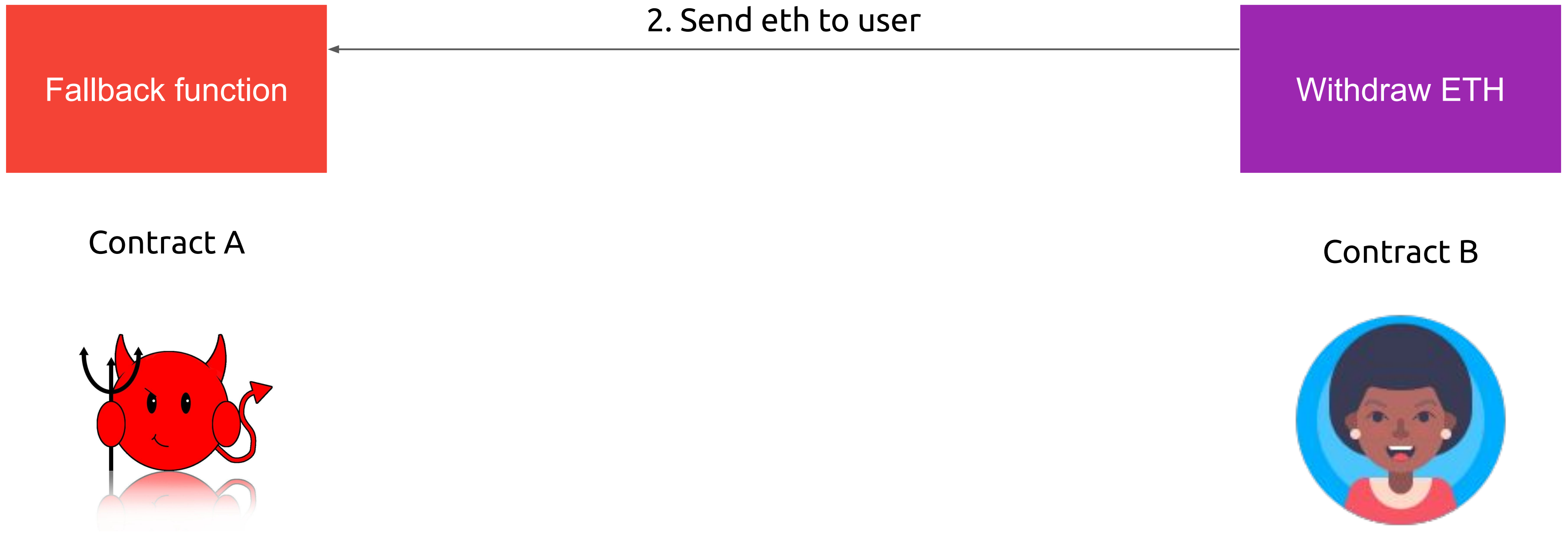
Contract B



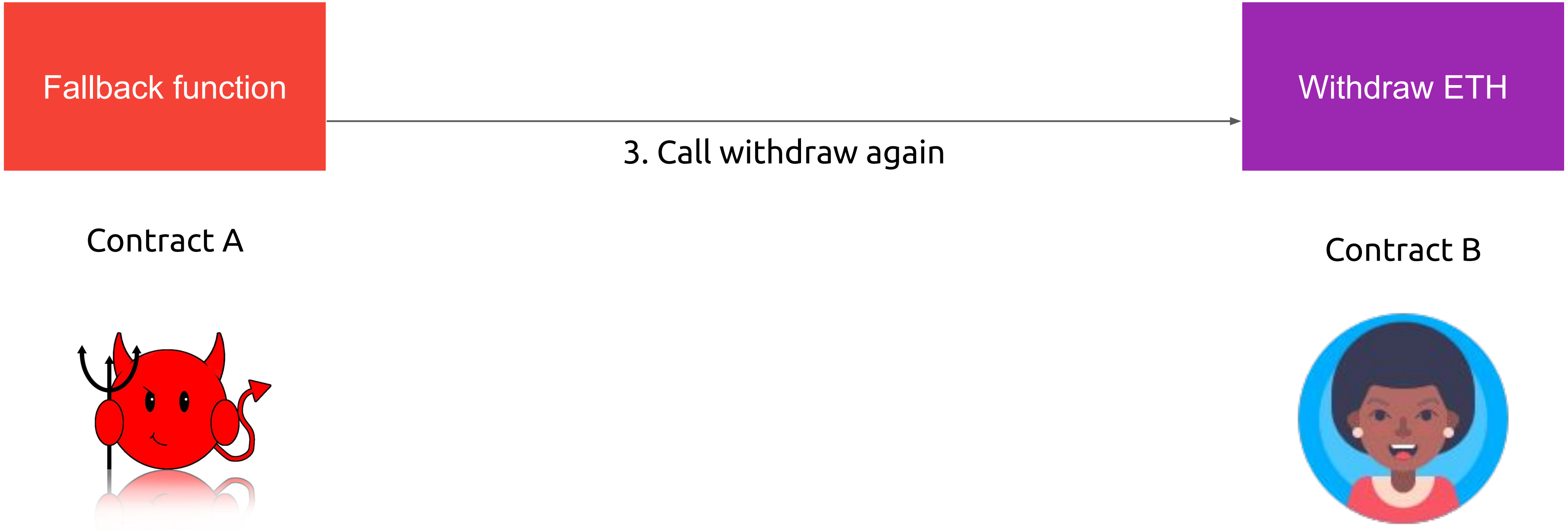
Reentrancy


Fallback function
Withdraw ETH
1. Call withdraw
Contract A
Contract B

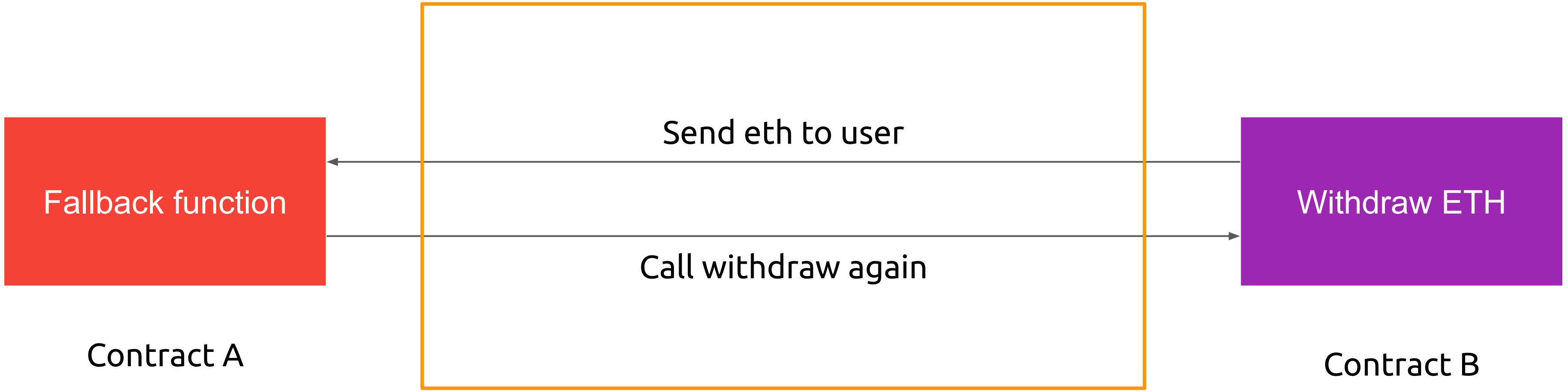
Reentrancy

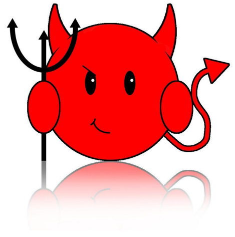

2. Send eth to user
Fallback function
Withdraw ETH
Contract A
Contract B

Reentrancy


Fallback function
Withdraw ETH
3. Call withdraw again
Contract A
Contract B

Reentrancy





Loop of function calls

Reentrancy

```
// INSECURE

mapping (address => uint) private userBalances;

function withdrawBalance() public {

      uint amountToWithdraw = userBalances[msg.sender];

      require(msg.sender.call.value(amountToWithdraw)());

      userBalances[msg.sender] = 0;

}
```

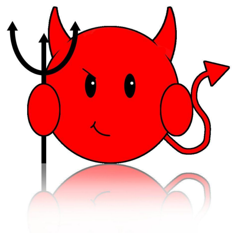
Reentrancy

```
// INSECURE

mapping (address => uint) private userBalances;

function withdrawBalance() public {

    uint amountToWithdraw = userBalances[msg.sender];

    require(msg.sender.call.value(amountToWithdraw)());

    userBalances[msg.sender] = 0;

}
```

Reentrancy

```
// INSECURE

mapping (address => uint) private userBalances;

function withdrawBalance() public {

      uint amountToWithdraw = userBalances[msg.sender];

      require(msg.sender.call.value(amountToWithdraw)());

      userBalances[msg.sender] = 0;

}
```

```
function () payable {

      if (victim.balance >= msg.value) {

            victim.withdrawBalance();

      }

   }
```

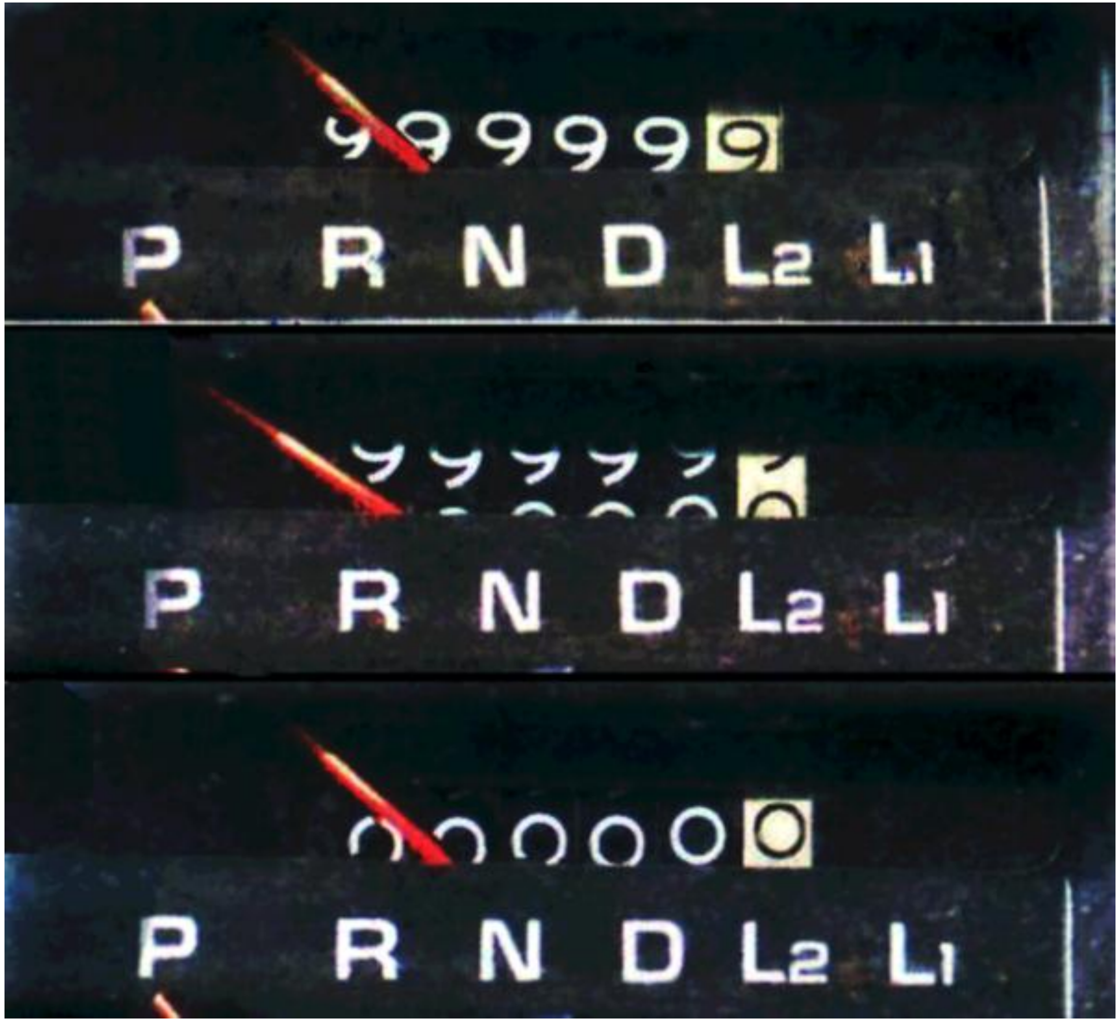


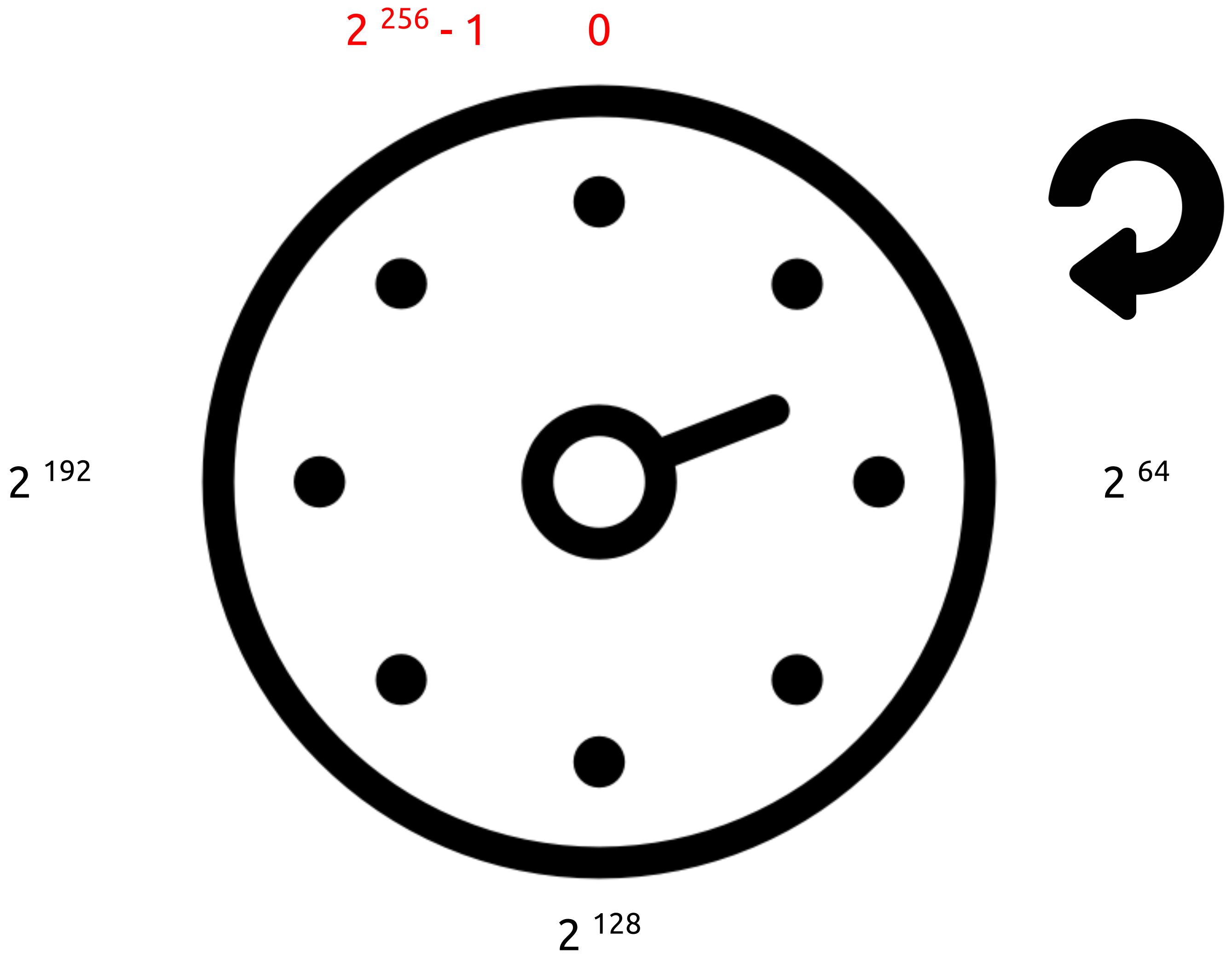
Reentrancy: solutions

```
// SECURE

mapping (address => uint) private userBalances;

function withdrawBalance() public {

      uint amountToWithdraw = userBalances[msg.sender];

      userBalances[msg.sender] = 0;

      msg.transfer(amountToWithdraw);

}
```

- Use `transfer` or `send` instead of `call`
- Finish all internal work (ie. state changes) and then call external functions
- Checks-Effects-Interactions Pattern
- Mutexes
- Pull-push pattern

Integer Overflow and Underflow


999999
P R N D L2 L1
P R N D L2 L1
000000
P R N D L2 L1


$2^{256} - 1$
0
2^{64}
2^{128}
2^{192}

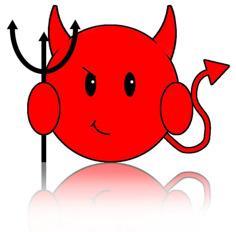
Integer Overflow and Underflow

```
// INSECURE
function withdraw(uint256 _value) {
    require(balanceOf[msg.sender] >= _value);
    msg.sender.call.value(_value)();
    balanceOf[msg.sender] -= _value;
}
```

Integer Overflow and Underflow

```
// INSECURE

function withdraw(uint256 _value) {

    require(balanceOf[msg.sender] >= _value);

    msg.sender.call.value(_value)();

    balanceOf[msg.sender] -= _value;

}
```

Integer Overflow and Underflow



```
// INSECURE

function withdraw(uint256 _value) {

    require(balanceOf[msg.sender] >= _value);

    msg.sender.call.value(_value)();

    balanceOf[msg.sender] -= _value;

}
```

```
function attack() {

    victim.donate.value(1)();

    victim.withdraw(1);

}

function() {

    if (performAttack) {

        performAttack = false;

        victim.withdraw(1);

    }

 }
```

Integer Overflow and Underflow: solutions

- Use OpenZeppelin’s SafeMath library

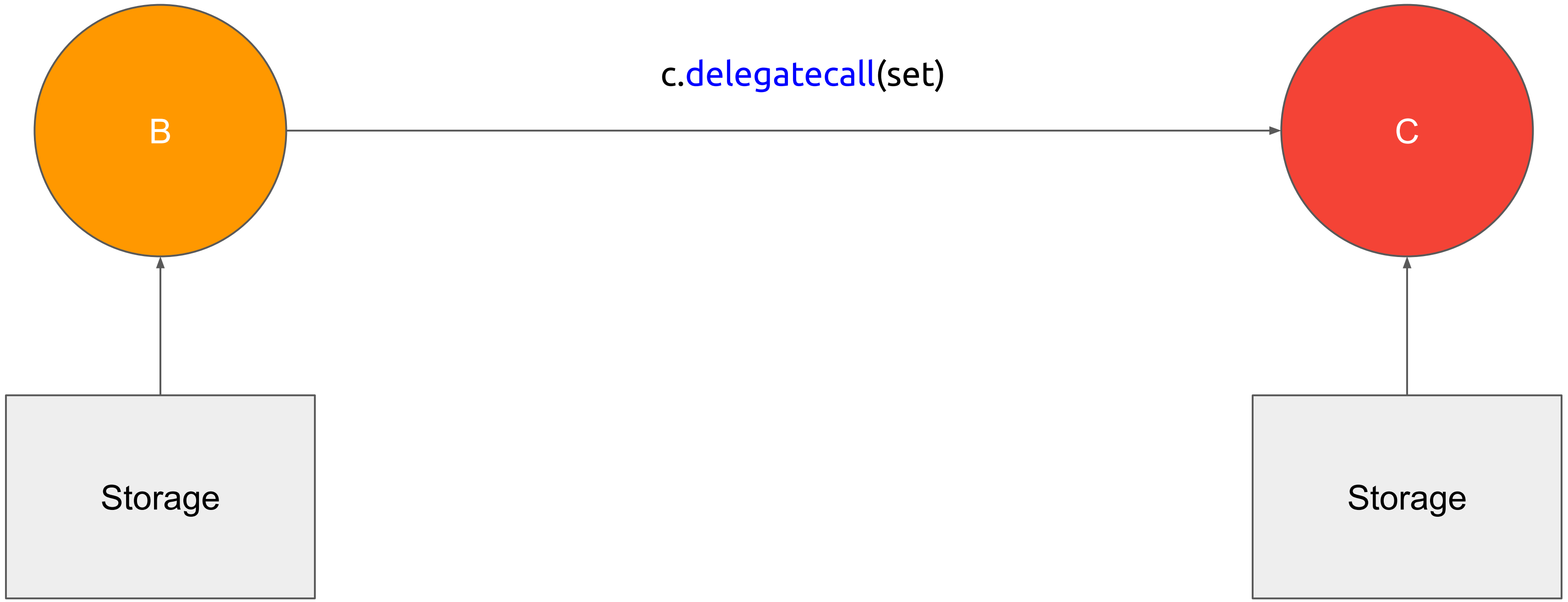
```
// OpenZeppelin: SafeMath.sol

function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");

      return c;
}

function sub(uint256 a, uint256 b) internal pure returns
(uint256) {
      require(b <= a, "SafeMath: subtraction overflow");
      uint256 c = a - b;

      return c;
}
```

Delegate call

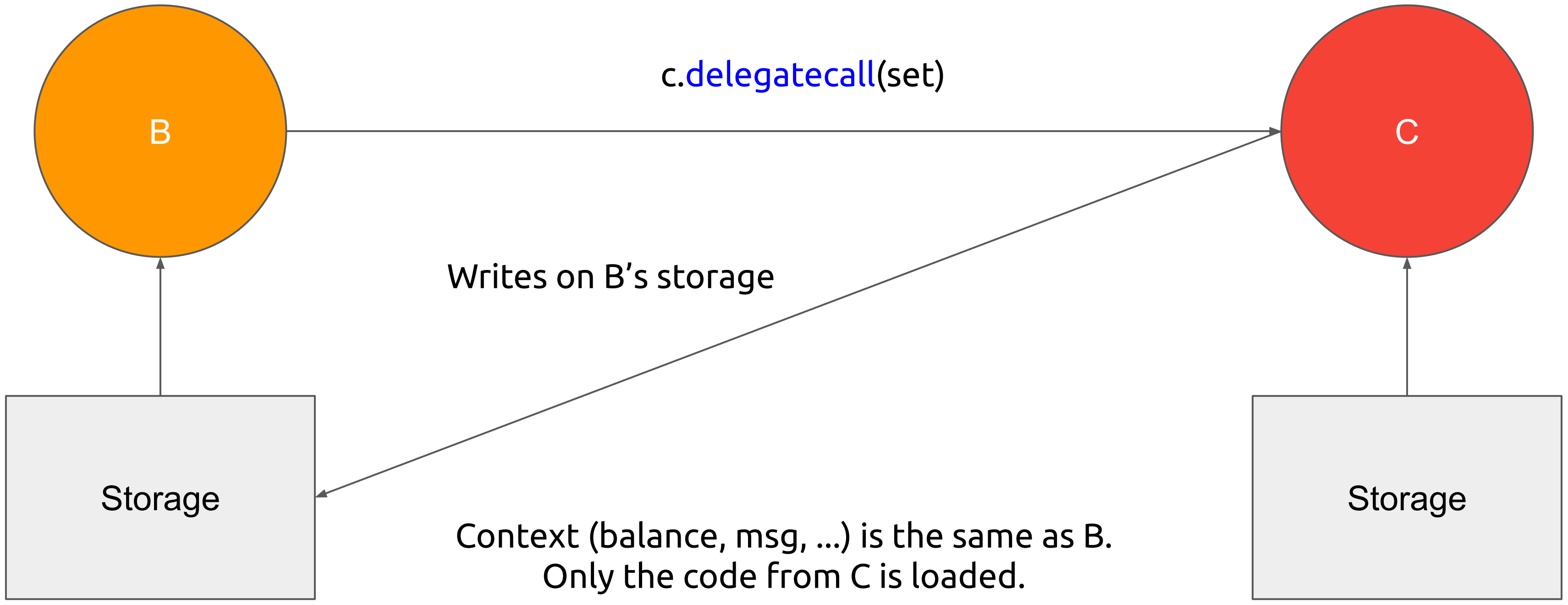
Delegate call

B
Storage
C
Storage

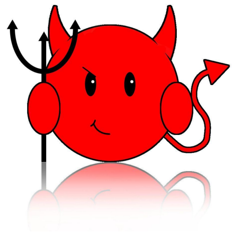
Delegate call



Delegate call


c.delegatecall(set)
B
C
Writes on B's storage
Storage
Storage
Context (balance, msg, ...) is the same as B.
Only the code from C is loaded.

Delegate call



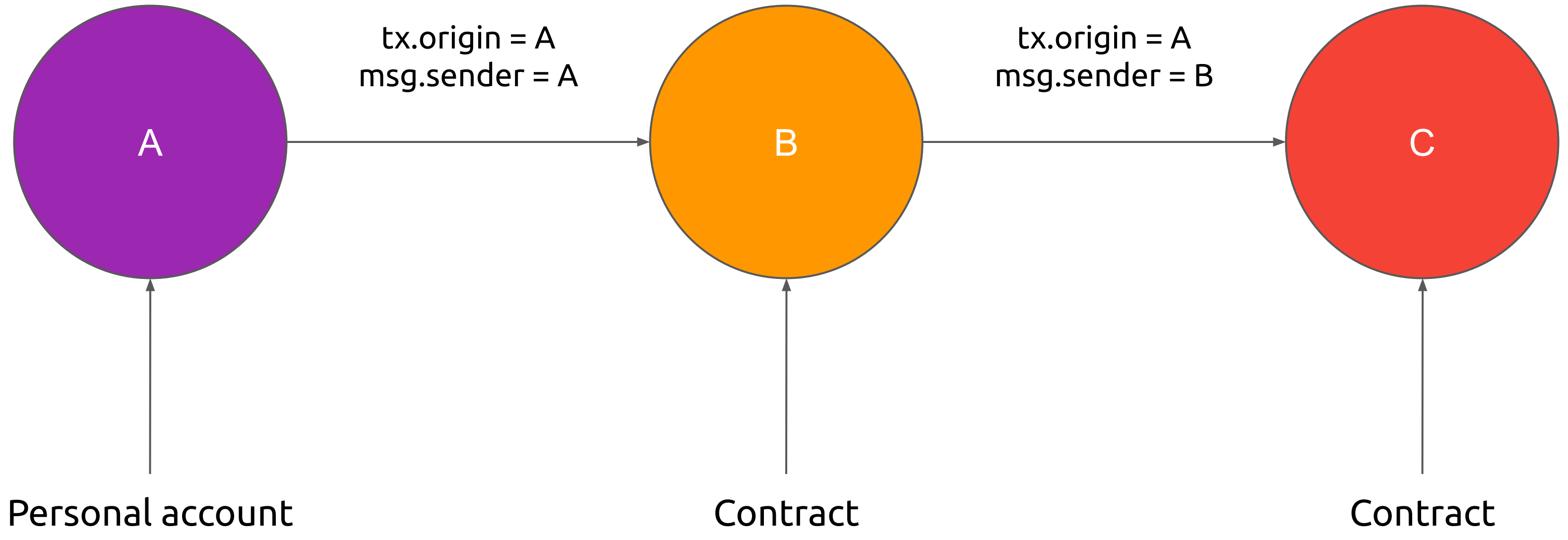
```
// INSECURE
address public owner;

Library library =

function() public {
      require(library.delegatecall(msg.data));
}
```

```
  address public owner;

  constructor (address _owner) public {
    owner = _owner;
  }

  function pwn() public {
    owner = msg.sender;
  }
}
```

Source: https://ethernaut.zeppelin.solutions

Use of tx.origin

Use of tx.origin



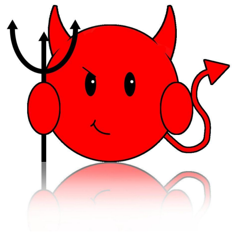
Use of tx.origin

```
// INSECURE
contract Bank {

    address owner;

    constructor() public {
        owner = msg.sender;
    }

    function sendTo(address receiver, uint amount) public {
        require(tx.origin == owner);
        receiver.call.value(amount)();
    }

}
```

Use of tx.origin

```
// INSECURE
contract Bank {

    address owner;

    constructor() public {
        owner = msg.sender;
    }

    function sendTo(address receiver, uint amount) public {
        require(tx.origin == owner);
        receiver.call.value(amount)();
    }

}
```

Use of tx.origin



```
// INSECURE
contract Bank {

    address owner;

    constructor() public {
        owner = msg.sender;
    }

    function sendTo(address payable receiver, uint amount)
public {
        require(tx.origin == owner);
        receiver.call.value(amount)();
    }

}
```

```
function() external payable {

  victim.sendTo(attacker,msg.sender.balance);

}
```

Checks-Effects-Interactions Pattern

1. **Perform** some **checks** (e.g sender, value, arguments ect).
2. Update **state**.
3. **Interaction** with other **contracts** (external function calls or send ether).

Pull over push

- **Do not transfer ether** to **users** (push) but **let** the **uses withdraw** (pull) their funds.
- **Isolates** each **external call** into its own transaction.
- **Avoids** multiple `send()` calls in a single transaction.
- **Reduces** problems with **gas limits**.
- **Trade-off** between **security** and **user experience**.

Pull over push: example

```
// INSECURE

function bid() payable {
      require(msg.value >= highestBid);

      if (highestBidder != address(0)) {
            highestBidder.transfer(highestBid);
      }

      highestBidder = msg.sender;
      highestBid = msg.value;
}
```

```
// SECURE

function bid() payable external {
      require(msg.value >= highestBid);

      if (highestBidder != address(0)) {
            refunds[highestBidder] += highestBid;
      }

      highestBidder = msg.sender;
      highestBid = msg.value;
}

function withdrawRefund() external {
      uint refund = refunds[msg.sender];
      refunds[msg.sender] = 0;
      msg.sender.transfer(refund);
}
```

Source: https://consensys.github.io/smart-contract-best-practices

Keep fallback function simple

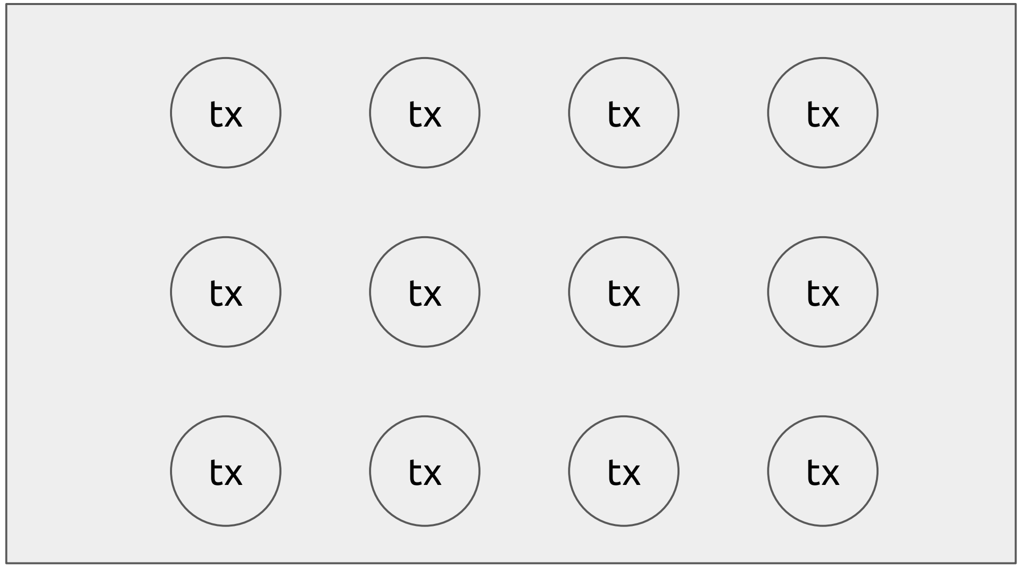
```
// BAD

function() payable {
      balances[msg.sender] += msg.value;
}
```

```
// GOOD

function deposit() payable external {
      balances[msg.sender] += msg.value;
}

function() payable {
      require(msg.data.length == 0);
      emit LogDepositReceived(msg.sender);
}
```

Source: https://consensys.github.io/smart-contract-best-practices

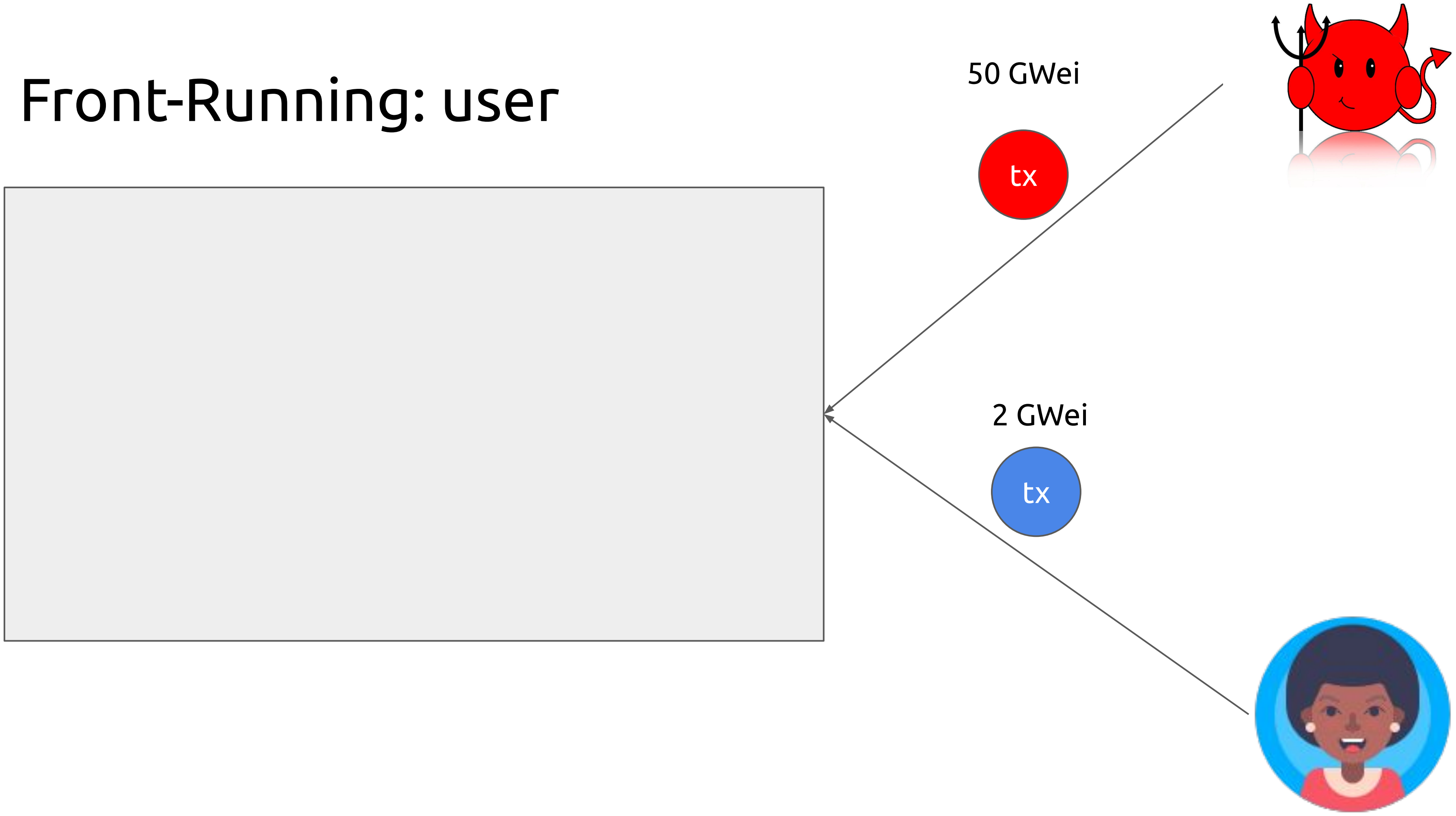
Front-Running

Front-Running

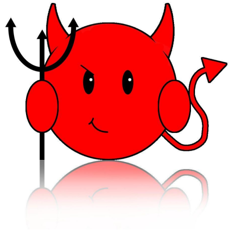


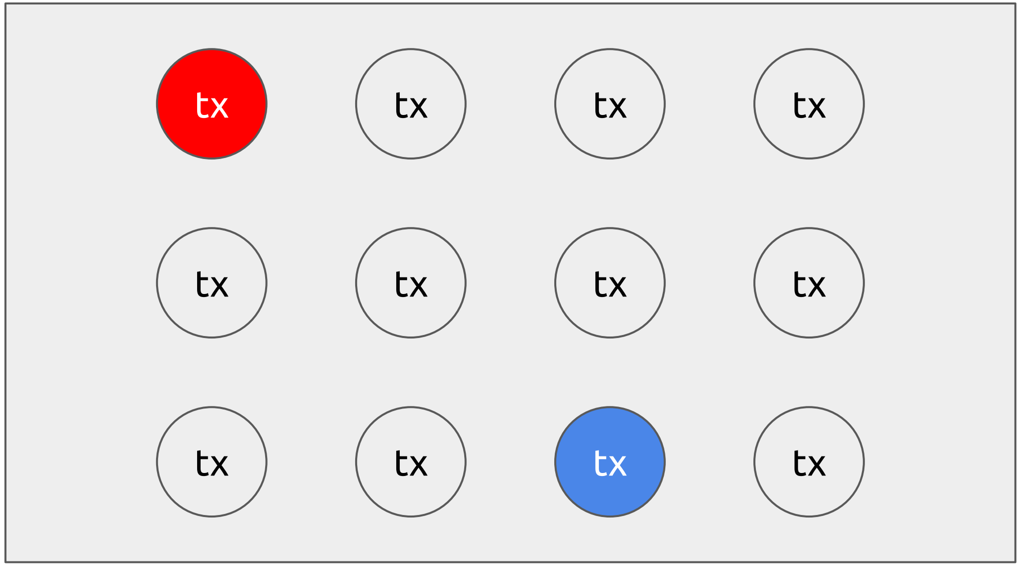
```
sortByGasPrice(txs, ‘desc’)
```

Front-Running: user

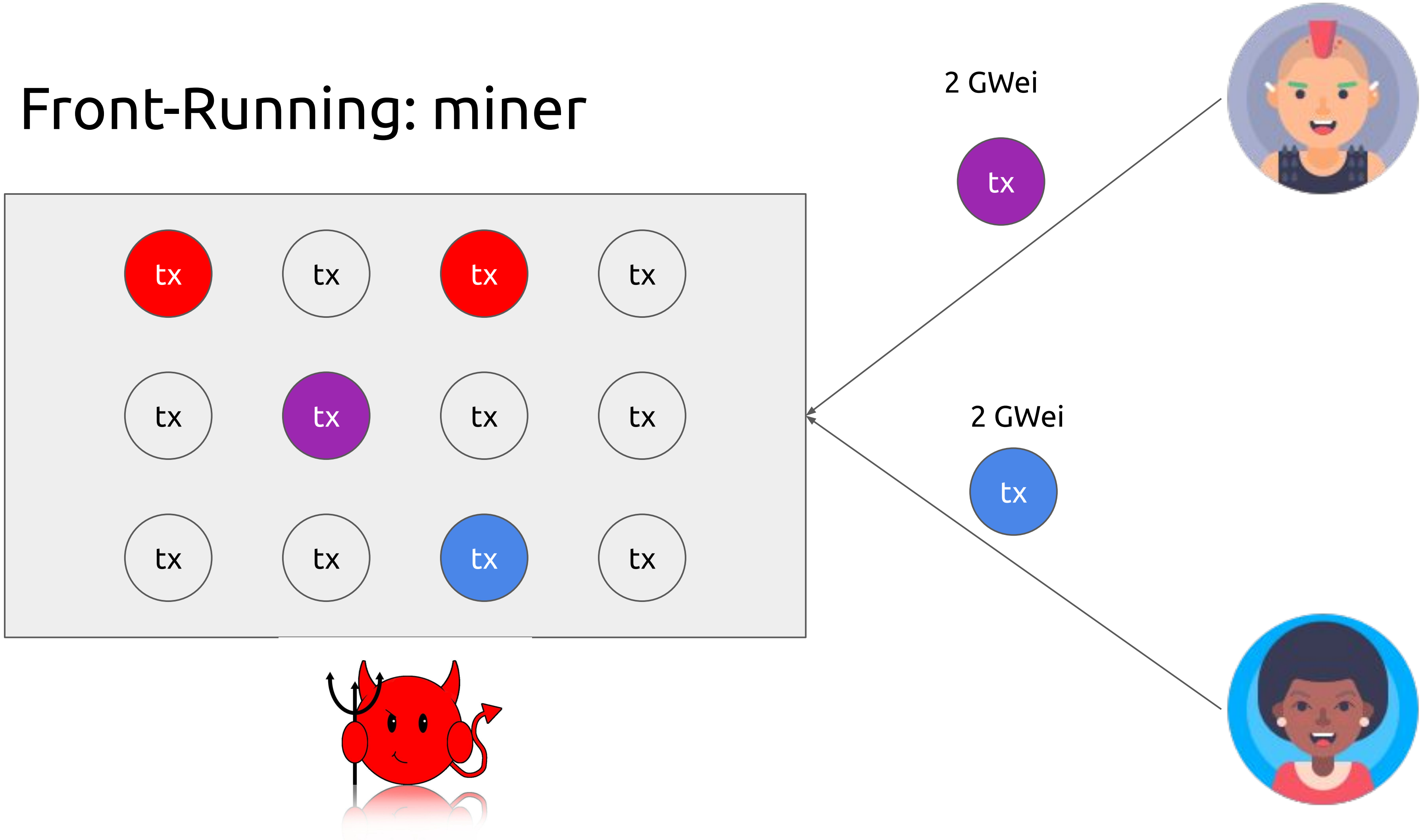


Front-Running: user





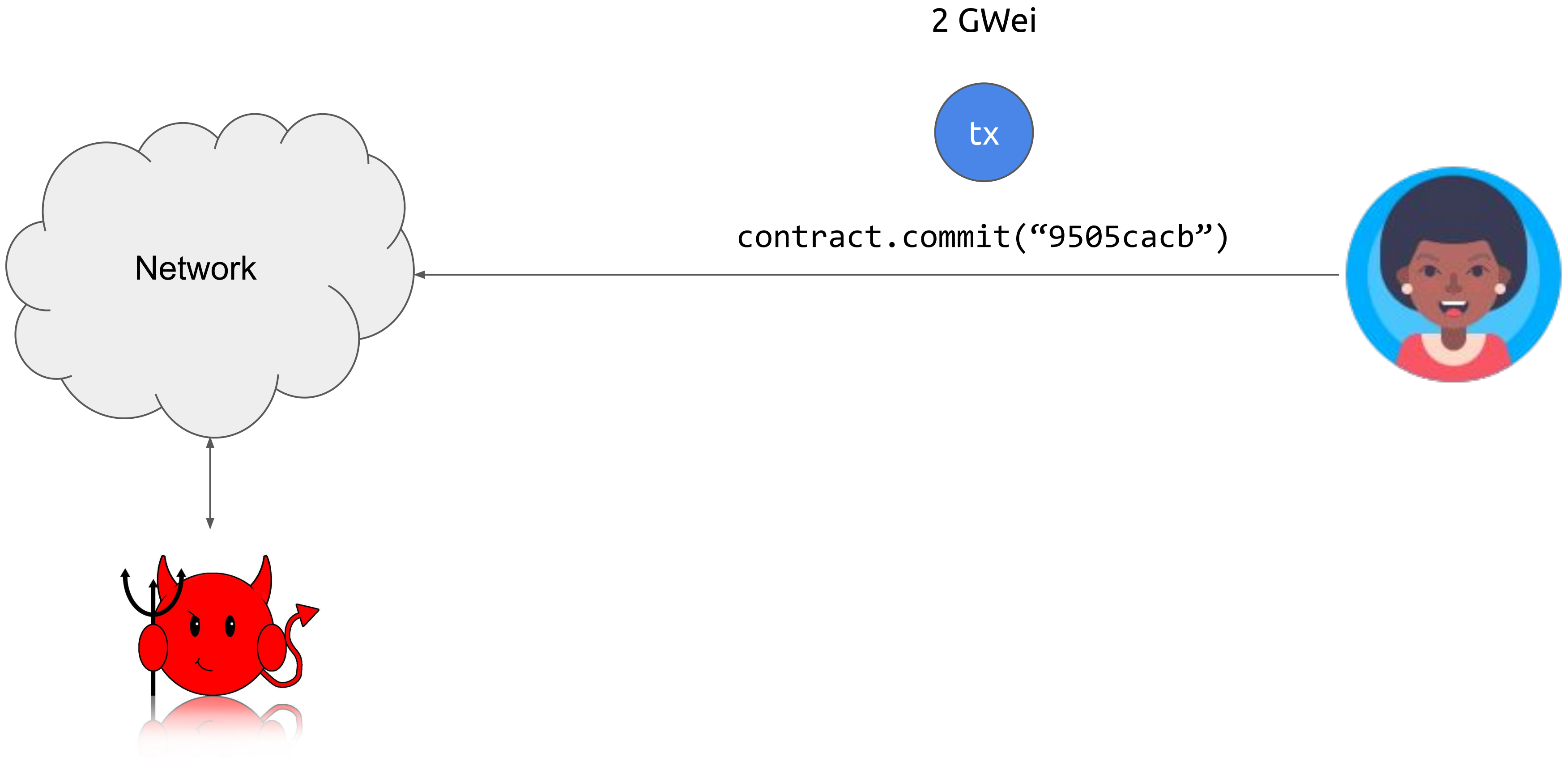
Front-Running: miner



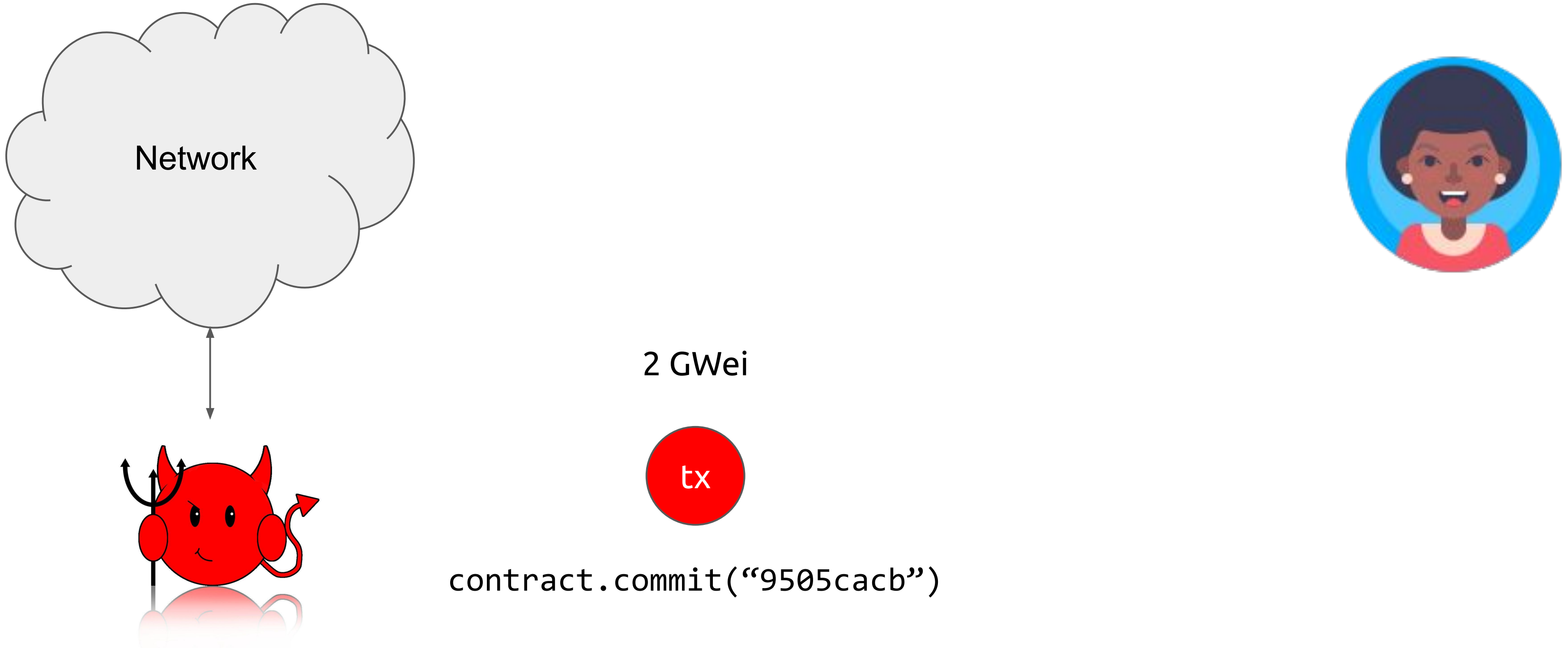
Front-Running: example

```
// INSECURE

function commit(bytes32 commit) public {

    commitments[commitment] = msg.sender;

}



function registerName(bytes32 name, bytes32 nonce) public {

    require(commitments[makeCommitment(name, nonce)] == msg.sender, “Commitment not
found!”);

     names[name] = msg.sender;

}
```

Front-Running: example

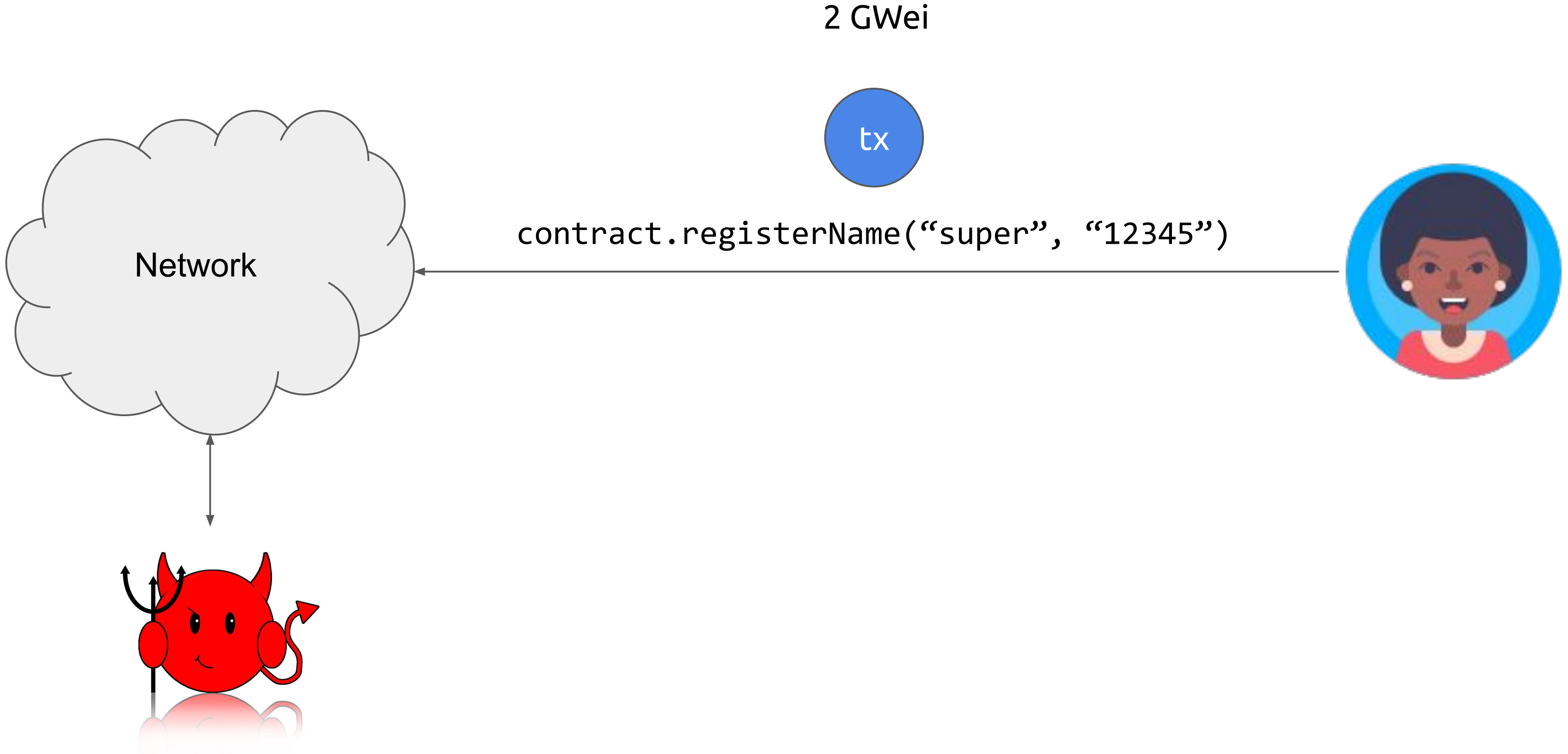


Front-Running: example



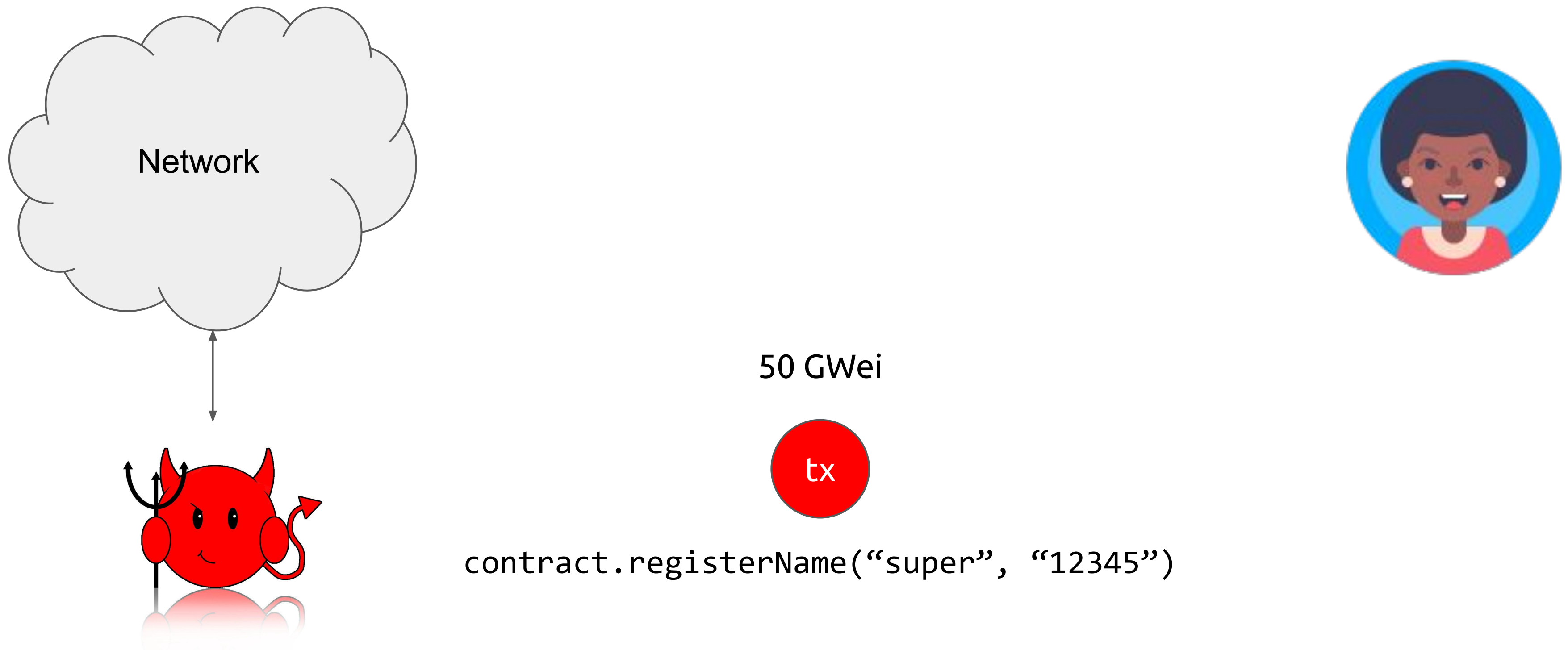
```
contract.commit(“9505cacb”)
```

Front-Running: example



```
contract.registerName("super", "12345")
```

Front-Running: example



```
contract.registerName(“super”, “12345”)
```

Randomness

Randomness: sources (?)

- block.number
- block.timestamp
- block.hash
- block.difficulty
- block.coinbase
- block.gasLimit
- now
- msg.sender

uint(keccak256(	timestamp	msg.sender	hash	...	)) % n

Randomness: sources (?)

- block.number
- block.times p
- block.has
- block.diff y

- lock.coinbase
- k.gasLimit
- m
- m ender

They can be manipulated by a malicious miner.
They are shared within the same block to all users.

Randomness

```
// INSECURE
bool won = (block.number % 2) == 0;



// INSECURE
uint random = uint(keccak256(block.timestamp)) % 2;



// INSECURE
address seed1 = contestants[uint(block.coinbase) % totalTickets].addr;
address seed2 = contestants[uint(msg.sender) % totalTickets].addr;
uint seed3 = block.difficulty;
bytes32 randHash = keccak256(seed1, seed2, seed3);
uint winningNumber = uint(randHash) % totalTickets;
address winningAddress = contestants[winningNumber].addr;
```

Randomness: blockhash

Not that private :)

```
// INSECURE
uint256 private _seed;

function random(uint64 upper) public returns (uint64 randomNumber) {
        _seed = uint64(keccack256(keccack256(block.blockhash(block.number), _seed), now));
        return _seed % upper;
}
```

Randomness: blockhash

Not that private :)

```
// INSECURE

uint256 constant private FACTOR =
1157920892373161954235709850086879078532699846656405640394575840079131296399;

function rand(uint max) constant private returns (uint256 result) {
        uint256 factor = FACTOR * 100 / max;
        uint256 lastBlockNumber = block.number - 1;
        uint256 hashVal = uint256(block.blockhash(lastBlockNumber));
        return uint256((uint256(hashVal) / factor)) % max;
}
```

Randomness: attack pattern

```
if (replicatedVictimConditionOutcome() == favorable)
    victim.tryMyLuck();
```

Source: Bad Randomness Is Even Dicier than You Think. Yannis Smaragdakis

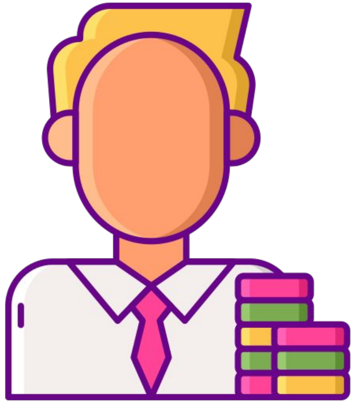
Randomness: intra-transaction information leak

```
victim.tryMyLuck();
require(victim.conditionOutcome() == favorable);
```

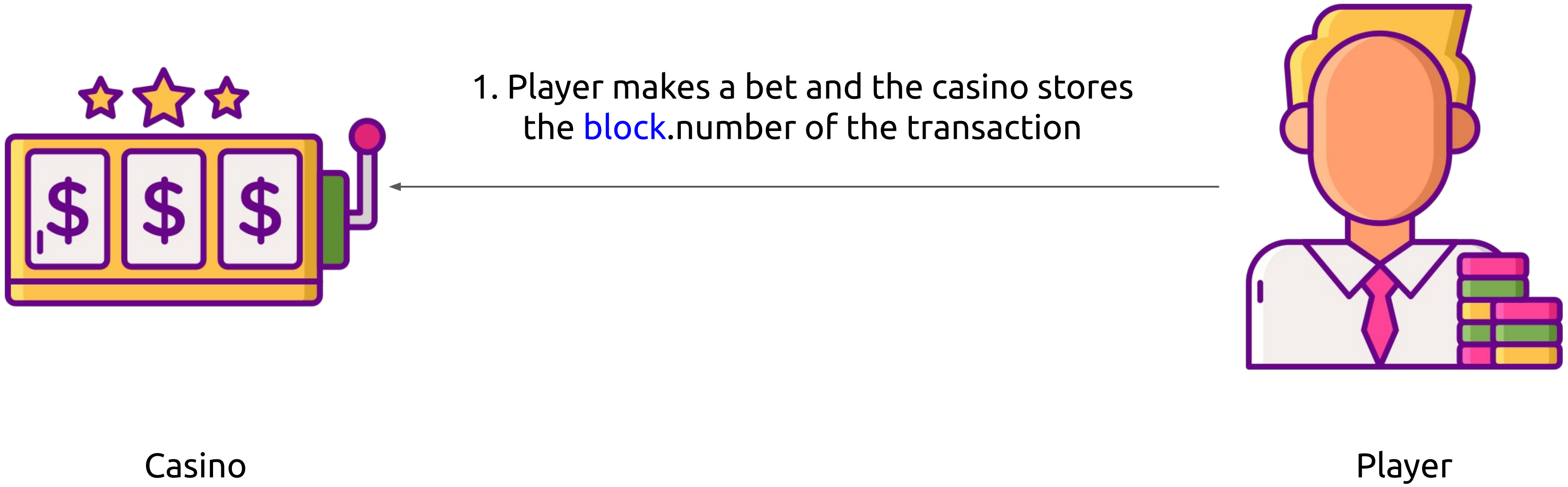
Source: Bad Randomness Is Even Dicier than You Think. Yannis Smaragdakis

What about future blocks ?

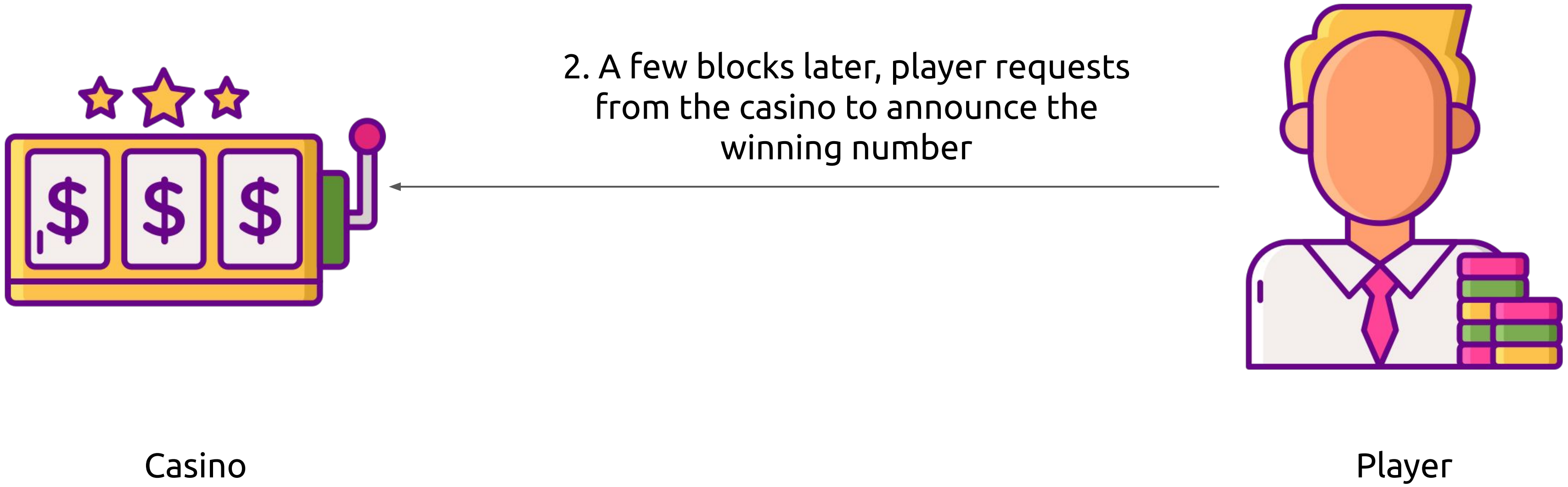
Casino



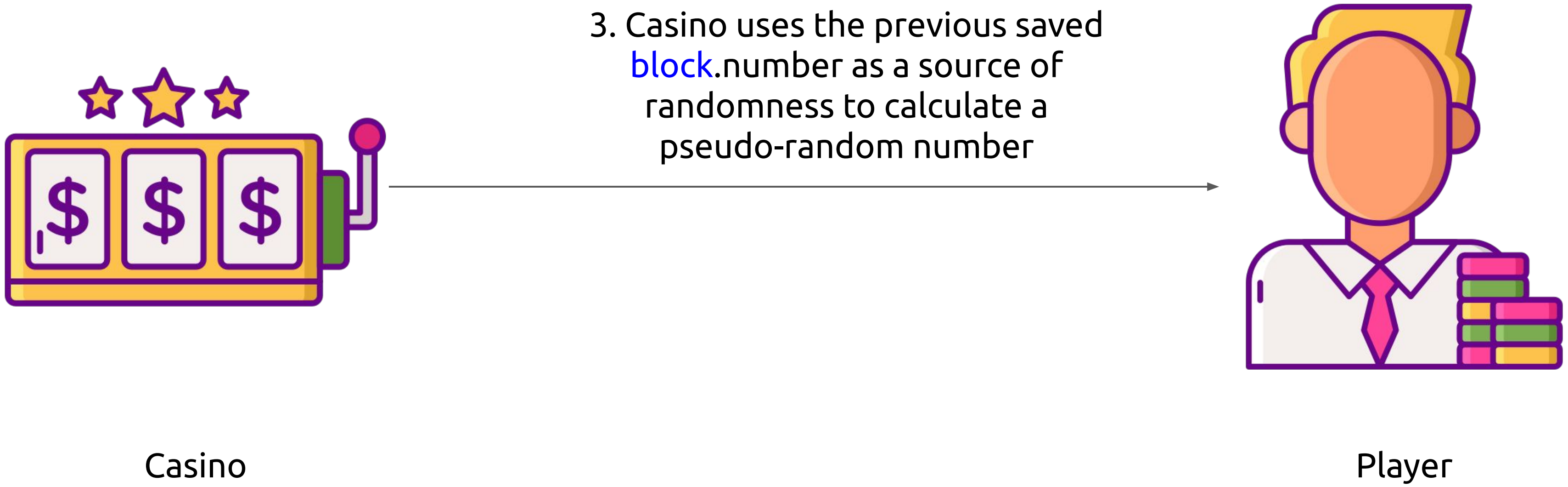
Player



1. Player makes a bet and the casino stores the block.number of the transaction



2. A few blocks later, player requests from the casino to announce the winning number


3. Casino uses the previous saved block.number as a source of randomness to calculate a pseudo-random number

Is the hash of a block in the future a good source of randomness against a malicious miner ?

Randomness: towards safer PRNG

- Commit–reveal schemes
- Example:
  - Casino and player commit each to a random value.
  - Casino and player reveal their random values.
  - Casino XORs the random values and take a seed. The seed can be combined with the hash of a future block.
- RANDAO (decentralized)

On-chain data is public

- Applications such as games and auction mechanisms required data to be private up until some point in time.
- Best strategy: commitment schemes:
  - Commit phase: Submit the hash of the value.
  - Reveal phase: Submit the value.
- Be aware of front-running!

References

- MadMax: surviving out-of-gas conditions in Ethereum smart contracts. Grech N., Kong M., Jurisevic A., Brent L., Scholz B., and Smaragdakis Y. OOPSLA '18.
- Bad Randomness Is Even Dicier than You Think. Yannis Smaragdakis
- https://consensys.github.io/smart-contract-best-practices/
- https://github.com/OpenZeppelin/openzeppelin-solidity
- https://github.com/ethereum/wiki/wiki/Safety
- https://ethernaut.zeppelin.solutions
- https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620
- https://github.com/slotthereum/source/issues/1

Thank you!