HS19

DISTRIBUTED SYSTEMS ADVANCED & BLOCKCHAIN

Ethereum & Solidity

T. Bocek

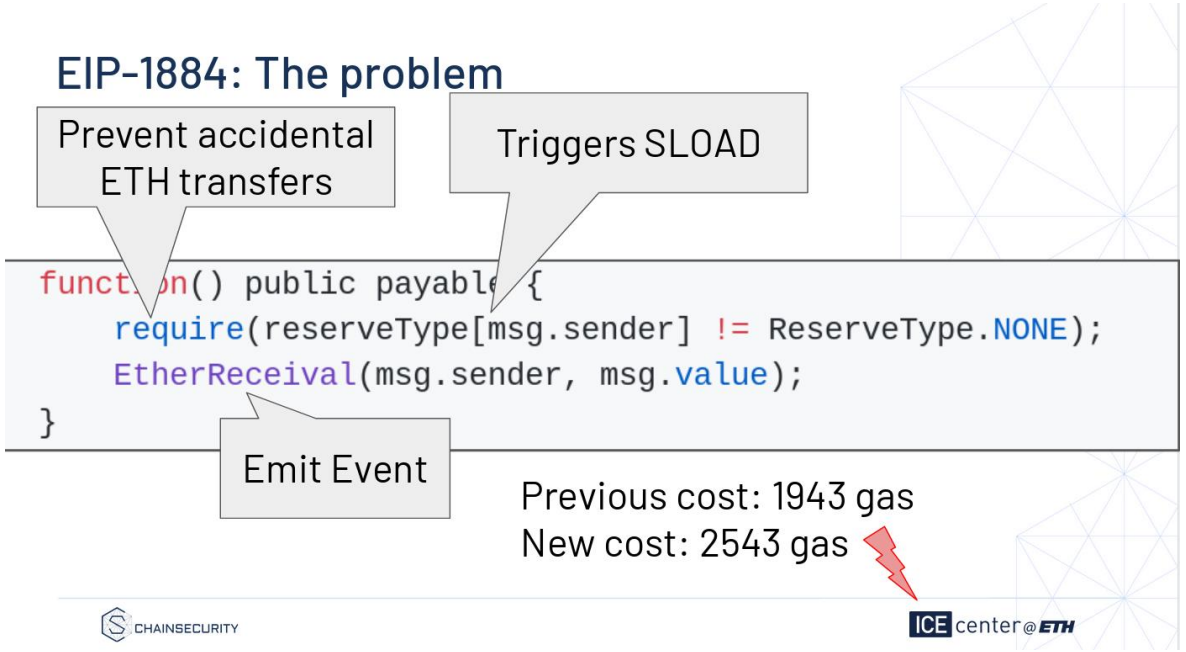
thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Distributed Systems - In the News

- **02.09.2019: Stop Using Solidity's transfer() Now**
  - Istanbul hard fork increases gas cost of SLOAD
  - EIP 1884: Repricing for disk access dependent opcodes
    - Balance between gas expenditure and resource consumption
    - “It is to be expected that it will rise again in the future, and may need future repricing, …”
  - Gas cost of SLOAD already changed in 2016 in EIP-150, due to spam tx attack



https://docs.google.com/presentation/d/1IiRYSjwle02zQUmWId06Bss8GrxGyw6nQAiZdCRFEPk/edit#slide=id.g60cc994f81_0_179

Distributed Systems - In the News

- **11.10.2019: Visa, Mastercard, eBay, Stripe Follow PayPal in Quitting Facebook's Libra Project**
  - Permissioned blockchain digital currency by Facebook (no mining)
  - Libra Blockchain is maintained by a distributed network of validator nodes
  - Launch planned in 2020, started in 2017
  - Libra token is stable coin, backed by basket of "regular" currencies
  - Project will not launch until all regulatory concerns have been met
  - Move smart contract language

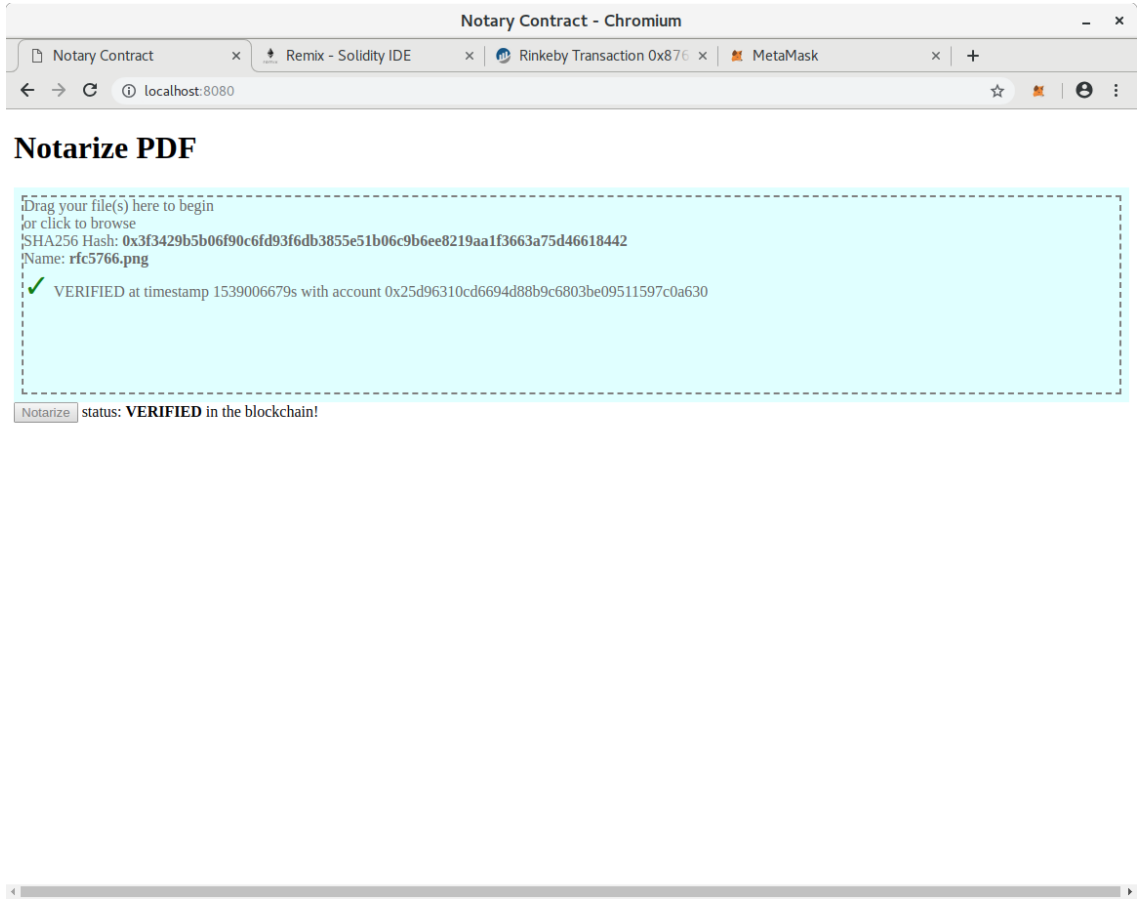
```
public main(payee: address, amount: u64) {
  let coin: 0x0.Currency.Coin = 0x0.Currency.withdraw_from_sender(copy(amount));
  0x0.Currency.deposit(copy(payee), move(coin));
}
```

```
public deposit(payee: address, to_deposit: Coin) {
  let to_deposit_value: u64 = Unpack<Coin>(move(to_deposit));
  let coin_ref: &mut Coin = BorrowGlobal<Coin>(move(payee));
  let coin_value_ref: &mut u64 = &mut move(coin_ref).value;
  let coin_value: u64 = *move(coin_value_ref);
  *move(coin_value_ref) = move(coin_value) + move(to_deposit_value);
}
```

```
public withdraw_from_sender(amount: u64): Coin {
  let transaction_sender_address: address = GetTxnSenderAddress();
  let coin_ref: &mut Coin = BorrowGlobal<Coin>(move(transaction_sender_address));
  let coin_value_ref: &mut u64 = &mut move(coin_ref).value;
  let coin_value: u64 = *move(coin_value_ref);
  RejectUnless(copy(coin_value) >= copy(amount));
  *move(coin_value_ref) = move(coin_value) - copy(amount);
  let new_coin: Coin = Pack<Coin>(move(amount));
  return move(new_coin);
}
```

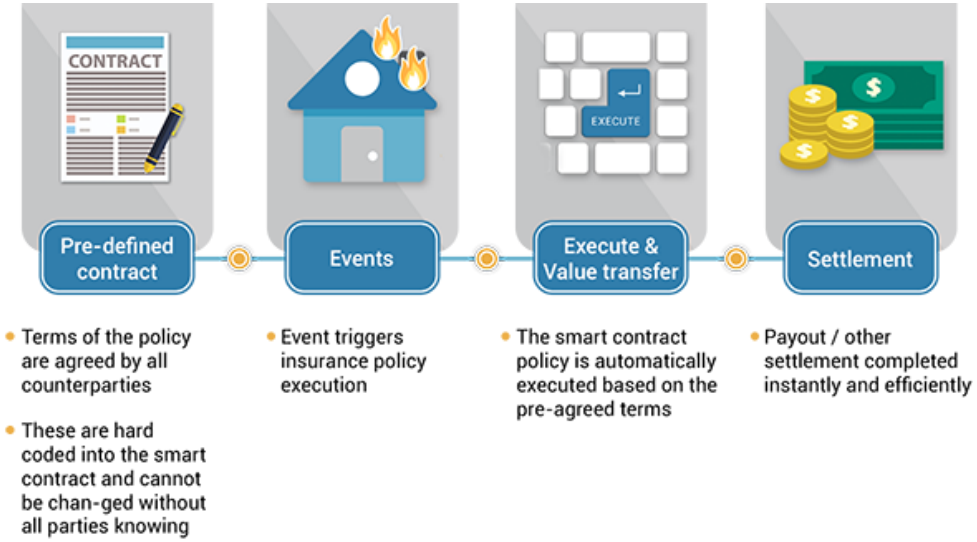
Recap: Java & JavaScript web3 examples

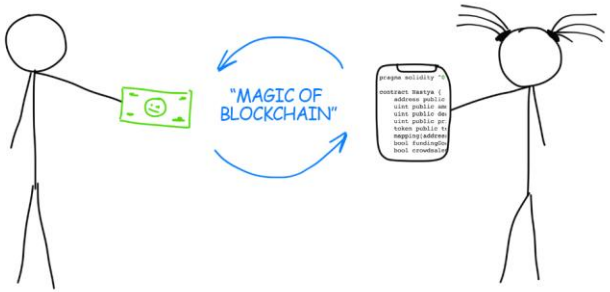
- **VSS recap**
  - Examples https://github.com/tbocek/VSS-WEB3
  - Install MetaMask or your own client
  - Run server locally
  - Notarize
- **Testonator**
  - Truffle/Ganache, but with JUnit
  - Runs with Solidity 0.5.x
  - Web3 – generate wrapper for SC
  - Testonator – use SC directly
  - Can run with rinkeby



What are Smart Contracts?

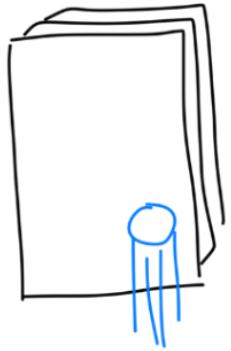
- **First proposed by Nick Szabo in 1994, inventor of “Bit Gold” in 1998**
- **Goal: a superior system for contractual agreements solely based on computer code**





Traditional vs. Smart Contracts

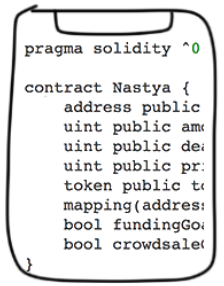
SILLY CONTRACT



- BORING OFFICIAL PAPER
- NO GUARANTEE
- KILL THE TREES
- NEEDS 50 LAWYERS

NOT YOUR BUDDY

SMART CONTRACT



- PERFECT CODE
- VERIFIED BY MATHS
- I'M A PROGRAMMER, YOU CAN'T FOOL ME
- ANYONE CAN WRITE HIS OWN

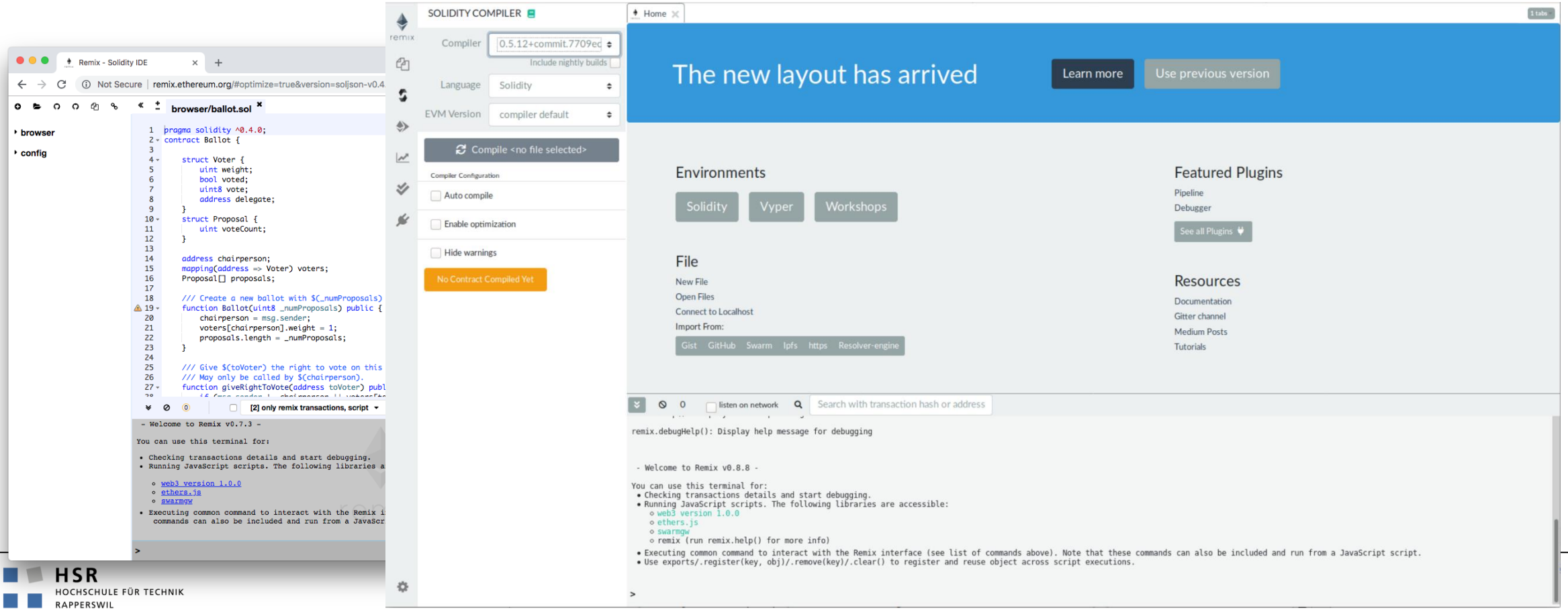
YOUR BUDDY

URL: b.socrative.com/student/

Room: HSR

Solidity IDE

- http://remix.ethereum.org **(Mixed Content: not https)**
- **Select Injected Web3 Provider (Metamask)**



Solidity - http://solidity.readthedocs.io

- **Version Pragma**

```
pragma solidity ^0.4.24; // not before 0.4.24, not after 0.5.0
```

- **Comments**

```
// This is a single-line comment.
/*
This is a
multi-line comment.
*/
```

- **Data Types**
  byte, bytes2 to bytes32, bytes (same as byte[], but more expensive), string, int8 to uint256, address, enum, bool

- **Contract**

```
contract SimpleStorage {
    uint256 storedData; // State variable
}
```

https://learnxinyminutes.com/docs/solidity/

Solidity IDE

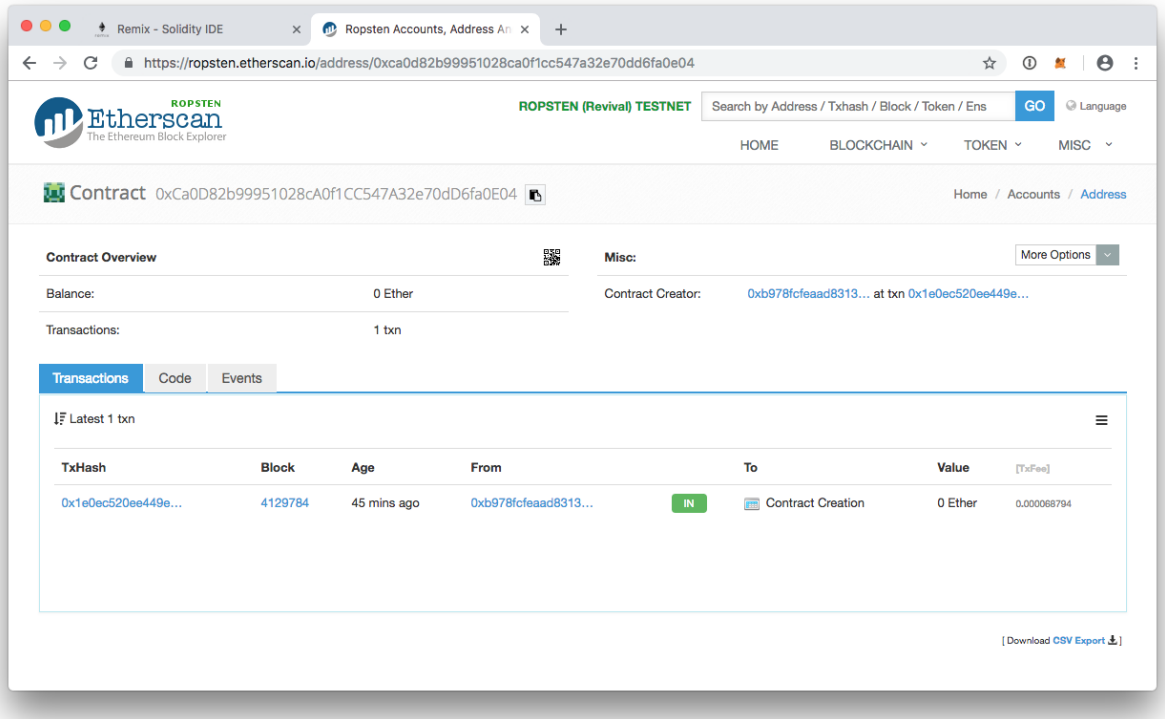
- **Create your first contract**

```
pragma solidity ^0.4.24;
// Minimal contract example
contract SimpleStorage {
    uint256 storedData; // State variable
}
```

- **Install MetaMask**
- **Compile**
- **Remix – Solidity IDE – Environment: Injected Web3 Provider**
- **Compile and push «Deploy»**

Check Deployment

- **You have mined you first contract!** 



- **Check on Etherscan**
- **How much did it cost?**

Gas: Ethereum's Fuel

- **Price that is paid for running a transaction or a contract**
- **Unit of measuring computational work**
- **Similar to the use of Kilowatts (kW) for measuring electricity in your house**
- **Current Price: https://ethgasstation.info**
  Current Gas Price (15.10.19): 1 Gwei
- **Every instruction needs to be paid for**
- **If you run out of gas, state is reverted, ETH gone**

Appendix G. Fee Schedule

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.

$W_{zero} = \{\text{STOP, RETURN}\}$
$W_{base} = \{\text{ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE, TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, POP, PC, MSIZE, GAS}\}$
$W_{verylow} = \{\text{ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}\}$
$W_{low} = \{\text{MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}\}$
$W_{mid} = \{\text{ADDMOD, MULMOD, JUMP}\}$
$W_{high} = \{\text{JUMPI}\}$
$W_{extcode} = \{\text{EXTCODESIZE}\}$

Solidity - http://solidity.readthedocs.io

- **Structs**

```
struct Account {
  string addr;
  uint256 amount;
}
```

Key	Value
0x3f51...	100
0x17aa...	0
0x4eb2...	85
...	...

- **Mapping (key-value pair, not iterable, but can be implemented), think of `Map<K,V>` in Java or `Dictionary<K,V>` in C#**

```
mapping(address => uint256) public balances;

balances[msg.sender] = 100; // Set balance of sender

uint256 balance = balances[msg.sender]; // Get balance of sender
```

Solidity – Functions

- **«Standard» Functions: Can read and modify the state**

```
uint256 counter;
function setCounter(uint256 _newValue) public {
        counter = _newValue;
}
```

- **View Functions: Do not modify the state (it's free!)**

```
uint256 counter;
function getCounter() public view returns (uint256) {
        return counter;
}
```

- **Pure Functions: Do not read from or modify the state.**

```
function multiply(uint256 a, uint256 b) public pure returns (uint256) {
        return a * b;
}
```

Solidity – Functions

- **Internal Functions: Only callable internally**

```
uint256 counter;
function setCounter(uint256 _newValue) internal {
        counter = _newValue;
}
```

- **External Functions: Only callable externally**
  Note: public and external differs in terms of gas usage

```
function setValues(uint256[20] _values) external {
        // do something
}
```

- **Payable Functions: Receives plain Ether**

```
function deposit() payable {
        // Access msg.value to get amount of Ether
}
```

Creating a smart contract that can get or set the balance of an address the *wrong* way.

```
contract BankExample {
  mapping(address => uint256) private balances;

  function getBalance(address _address) view public returns (uint256) {
    return balances[_address];
  }

  function setBalance(uint256 _newBalance, address _address) public {
    balances[_address] = _newBalance;
  }
}
```

Solidity – Basics

- **Often used Special Variables and Global Functions**
  - `block.number` (type of `uint256`): current block number
  - `gasleft()` (returns `uint256`): remaining gas
  - `msg.sender` (type of `address`): sender of the message
  - `msg.value` (type of `uint256`): number of wei sent with the message
  - `now` (type of `uint256`): current block timestamp

Update the previous smart contract such that only the sender of a transaction can get and set his own balance.

Use the `msg.sender` global variable to let only the sender of the transaction set his own balance.

```
contract BankExampleV2 {
  mapping(address => uint256) private balances;

  function getBalance() public view returns (uint256) {
    return balances[msg.sender];
  }

  function setBalance(uint256 _newBalance) public {
    balances[msg.sender] = _newBalance;
  }
}
```

Solidity – Ownership

- Who is the owner of a smart contract?
- How to transfer ownership?
- The *wrong* way:

```
contract OwnedContract {
   address owner;
   constructor() public {
     owner = msg.sender;
   }

   function transfer(address _newOwner) public {
     owner = _newOwner;
   }
 }
```

- Everybody can transfer the ownership
- Use `require` to test user inputs

```
if (msg.sender != owner) { throw; }

// Do something only the owner is allowed to
```

- behaves the same as:

```
require(msg.sender == owner);

// Do something only the owner is allowed to

//better: require(msg.sender == owner, "Unauthorized");
```

- The use of revert(), assert(), and require() in Solidity
  - Require, when errors can happen will not use all gas
  - Assert, when errors should not happen, will use all gas

Solidity – Ownership

```
contract OwnedContract {
   address owner;
   constructor() public {
     owner = msg.sender;
   }

   function transfer(address _newOwner) public {
     require(owner == msg.sender, "Sender not authorized");
     owner = _newOwner;
   }
}
```

- **Update the previous BankExample contract to let the owner set balances arbitrarily. Hint: Create a second function to set the balance, e.g.**

```
setBalance(uint256 _balance, address _address)
```

- **Create a second `setBalance` method which uses `require` to verify if the sender of the transaction is the owner.**

```
contract OwnedBankExample {
  ...

  function setBalance(uint256 _newBalance) public {
    balances[msg.sender] = _newBalance;
  }

  function setBalance(uint256 _newBalance, address _address) public {
    require(owner == msg.sender, "Sender not authorized");
    balances[_address] = _newBalance;
  }
}
```

Solidity – Events/Notifications

- **Events are a way for smart contracts written in Solidity to log that something has occurred**
- **Interested observers, notably JavaScript front ends for decentralized apps, can watch for events and react accordingly.**

transaction cost	43044 gas
execution cost	21580 gas
hash	0x306ba94b3681cc650cf62d55ae4a121aea240a5dd7077cb660c03f77a82ae537
input	0x82a...00064
decoded input	{ "uint256 newBalance": "100" }
decoded output	{}
logs	[{ "from": "0x692a70d2e424a56d2c6c27aa97d1a86395877b3a", "topic": "0x5f66d2a93b609bc6596b75c6dbb0e4f3f7cafd4b3b617157ff304d1076e58375", "event": "Update", "args": { "0": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "1": "100", "_user": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c", "_newBalance": "100", "length": 2 } }]
value	0 wei

Solidity – Events/Notifications

```
contract EventsExample {
   event OwnerChanged(
     address _oldOwner,
     address _newOwner
   );

   function transfer(address _newOwner) public {
     require(owner == msg.sender, "Sender not authorized");
     emit OwnerChanged(owner, _newOwner);
     owner = _newOwner;
   }
}
```

- **Update the previous BankExample. Emit an event when the balance of an address changes. The event should include the address, the old balance and the new balance.**

Solidity – Events/Notifications

```
event BalanceChanged(address indexed _address,
                     uint256 _oldBalance, uint256 _newBalance);

function setBalance(uint256 _newBalance) public {
   uint256 _oldBalance = balances[msg.sender];
   balances[msg.sender] = _newBalance;
   emit BalanceChanged(_address, _oldBalance, _newBalance);
}

function setBalance(uint256 _newBalance, address _address) public {
  require(owner == msg.sender, "Sender not authorized");
  uint256 _oldBalance = balances[_address];
  balances[_address] = _newBalance;
  emit BalanceChanged(_address, _oldBalance, _newBalance);
}
```

Solidity – Events/Notifications

```
event BalanceChanged(address indexed _address,
                     uint256 _oldBalance, uint256 _newBalance);

function setBalance(uint256 _newBalance) public {
  _setBalance(_newBalance, msg.sender);
}

function setBalance(uint256 _newBalance, address _address) public {
  require(owner == msg.sender, "Sender not authorized");
  _setBalance(_newBalance, _address);
}

function _setBalance(uint256 _newBalance, address _address) internal {
  uint256 _oldBalance = balances[_address];
  balances[_address] = _newBalance;
  emit BalanceChanged(_address, _oldBalance, _newBalance);
}
```

URL: b.socrative.com/student/

Room: HSR

ERC20 Token Standard

- **ERC20: token/cryptocurrency interface on Ethereum.**
  - As of April 2019, more than 181’000 ERC-20-compatible tokens exist on Ethereum main network
  - Tokens can be bought, sold, or traded
- **Describes the functions and events that an Ethereum token contract has to implement**

```
contract ERC20 {
    function totalSupply() public view returns (uint256 totalSupply);
    function balanceOf(address _owner) public view returns (uint256 balance);
    function transfer(address _to, uint _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
        public returns (bool success);
    function approve(address _spender, uint _value)
        public returns (bool success);
    function allowance(address _owner, address _spender) public view
        returns (uint remaining);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
    event Transfer(address indexed _from, address indexed _to, uint _value);
}
```

ERC20 Token Standard

- **Our token: only partial ERC20 support**

```
contract ERC20 {
    function totalSupply() public view returns (uint256 totalSupply);
    function balanceOf(address _owner) public view returns (uint256 balance);
    function transfer(address _to, uint _value) public returns (bool success);
    function transferFrom(address _from, address _to, uint _value)
        public returns (bool success);
    function approve(address _spender, uint _value)
        public returns (bool success);
    function allowance(address _owner, address _spender) public view
        returns (uint remaining);
    event Approval(address indexed _owner, address indexed _spender, uint _value);
    event Transfer(address indexed _from, address indexed _to, uint _value);
}
```

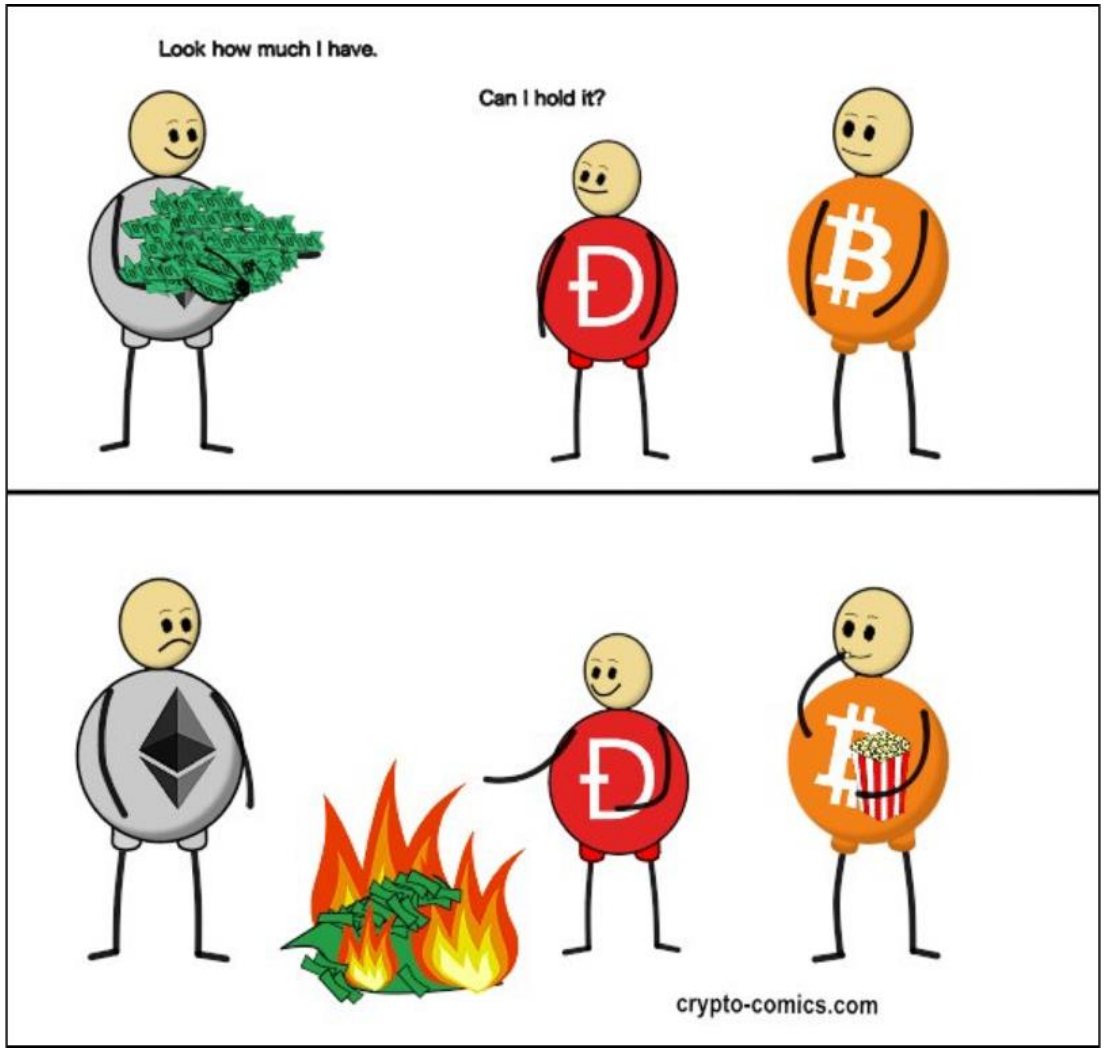
Lets do a token

```
pragma solidity ^0.5.12;
contract YourTestToken {
    string public constant name = "Your Token Name";
    string public constant symbol = "YTN";
    function totalSupply() public view returns (uint256 totalSupply) {
        // Your code
    }
    function balanceOf(address _owner) public view returns (uint256 balance) {
        // Your code
    }
    function transfer(address _to, uint256 _value) public returns (bool success) {
        // Your code
    }
}
```

- **Hint: you only need three state variables (not public):**
  - `address owner;`
  - `mapping (address => uint) balances;`
  - `uint256 totalSupply;`

Smart Contract Security

- **Smart contract programming requires a different engineering mindset**
- **Cost of failure can be high, and change can be difficult**
- **Notable Examples: Parity Bug, DAO Hack**



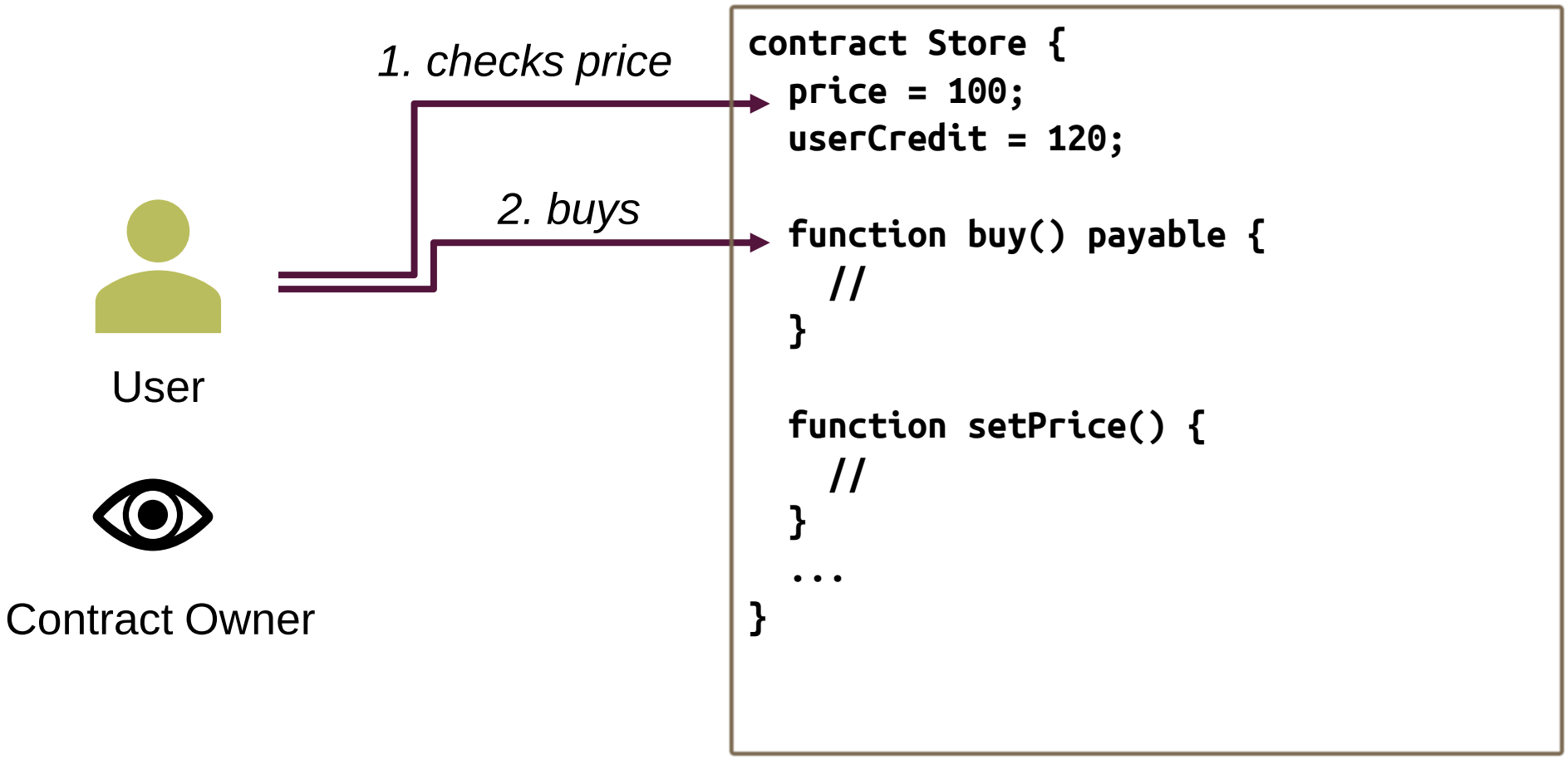
Smart Contract Security

- **Transaction Ordering**
  > Blockchain Shop

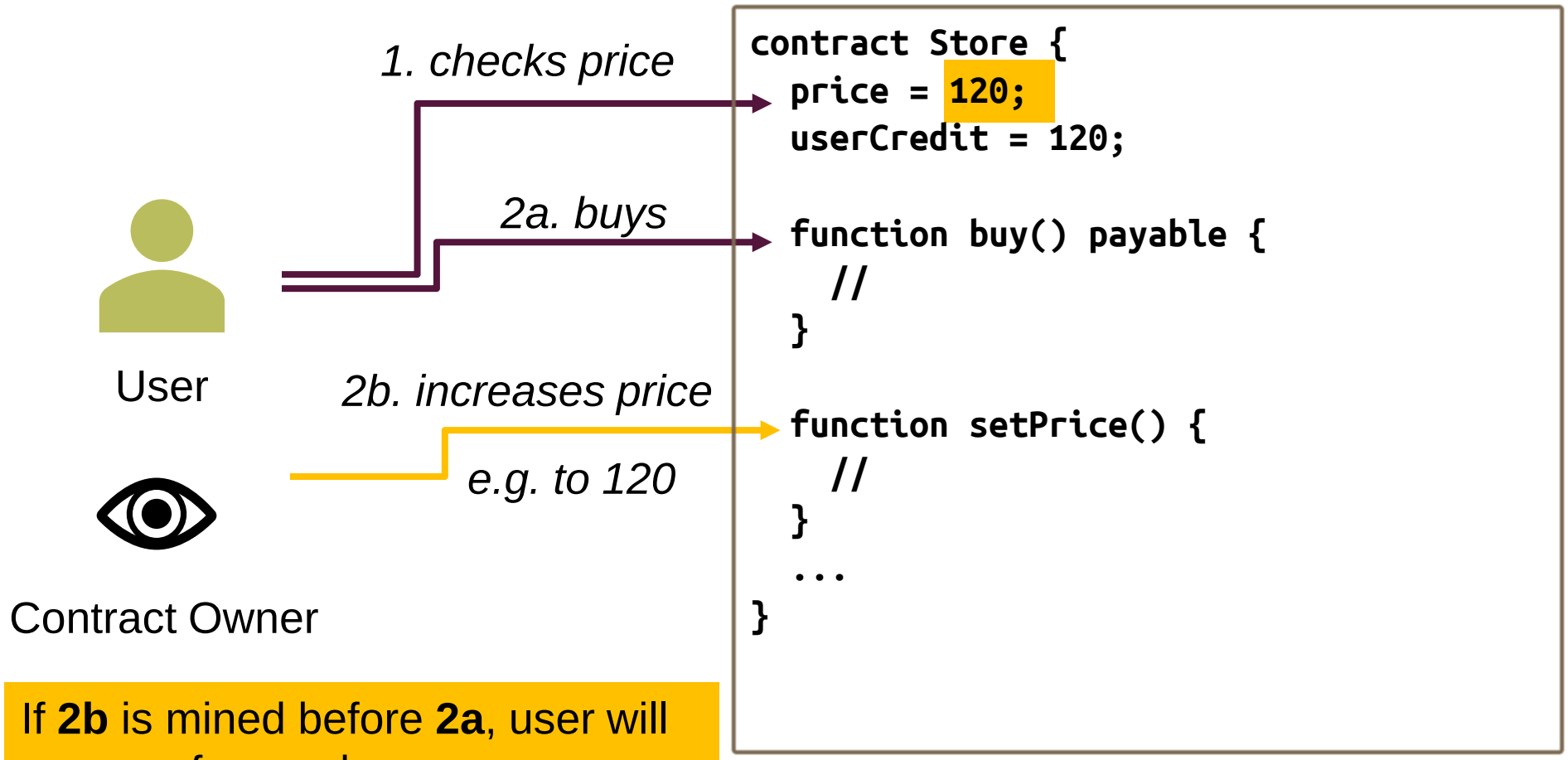
- **Reentrancy Attacks**
  > Good ATM | Bad ATM

Source: James Chiang

Transaction Ordering

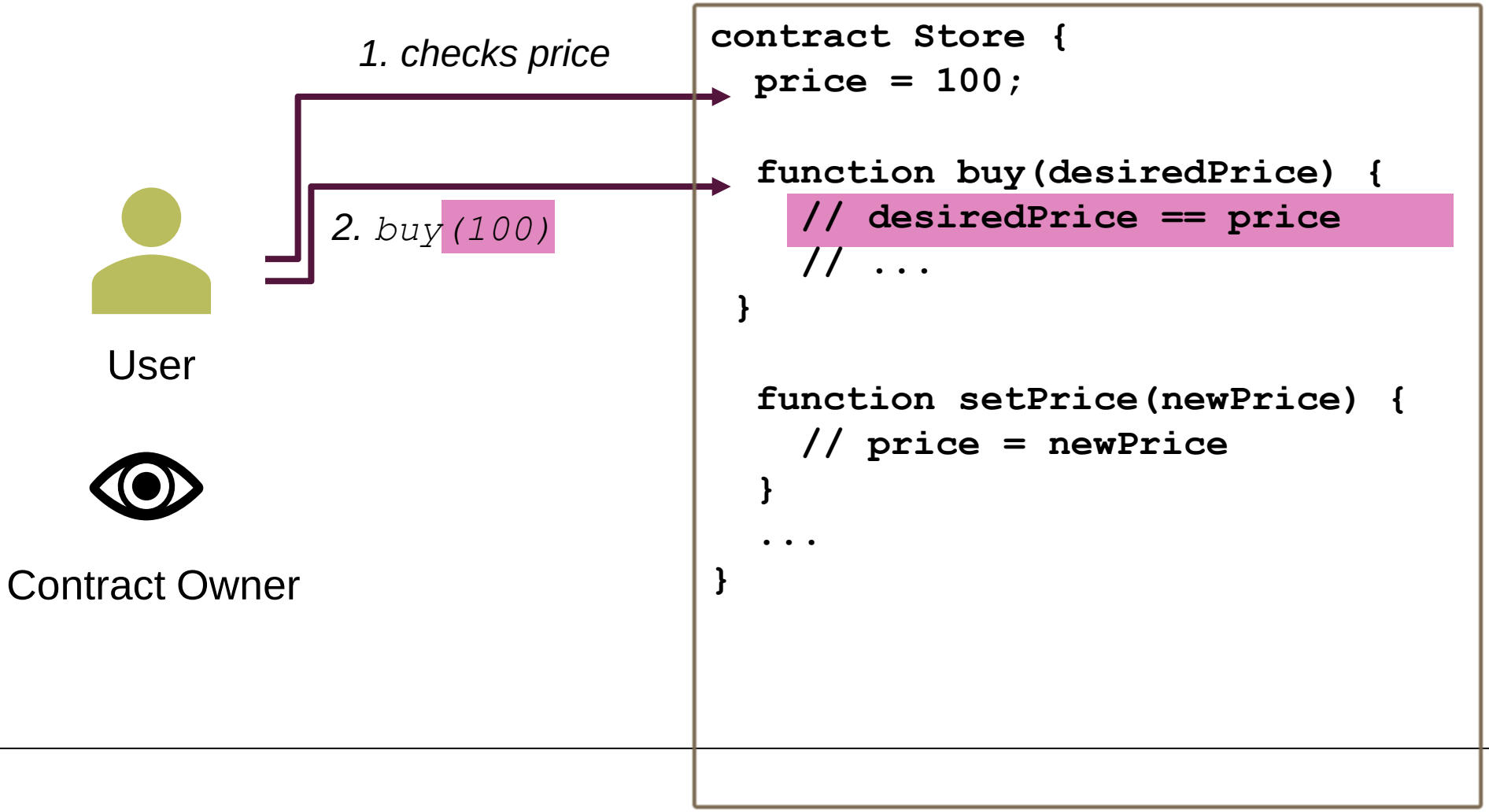


Transaction Ordering

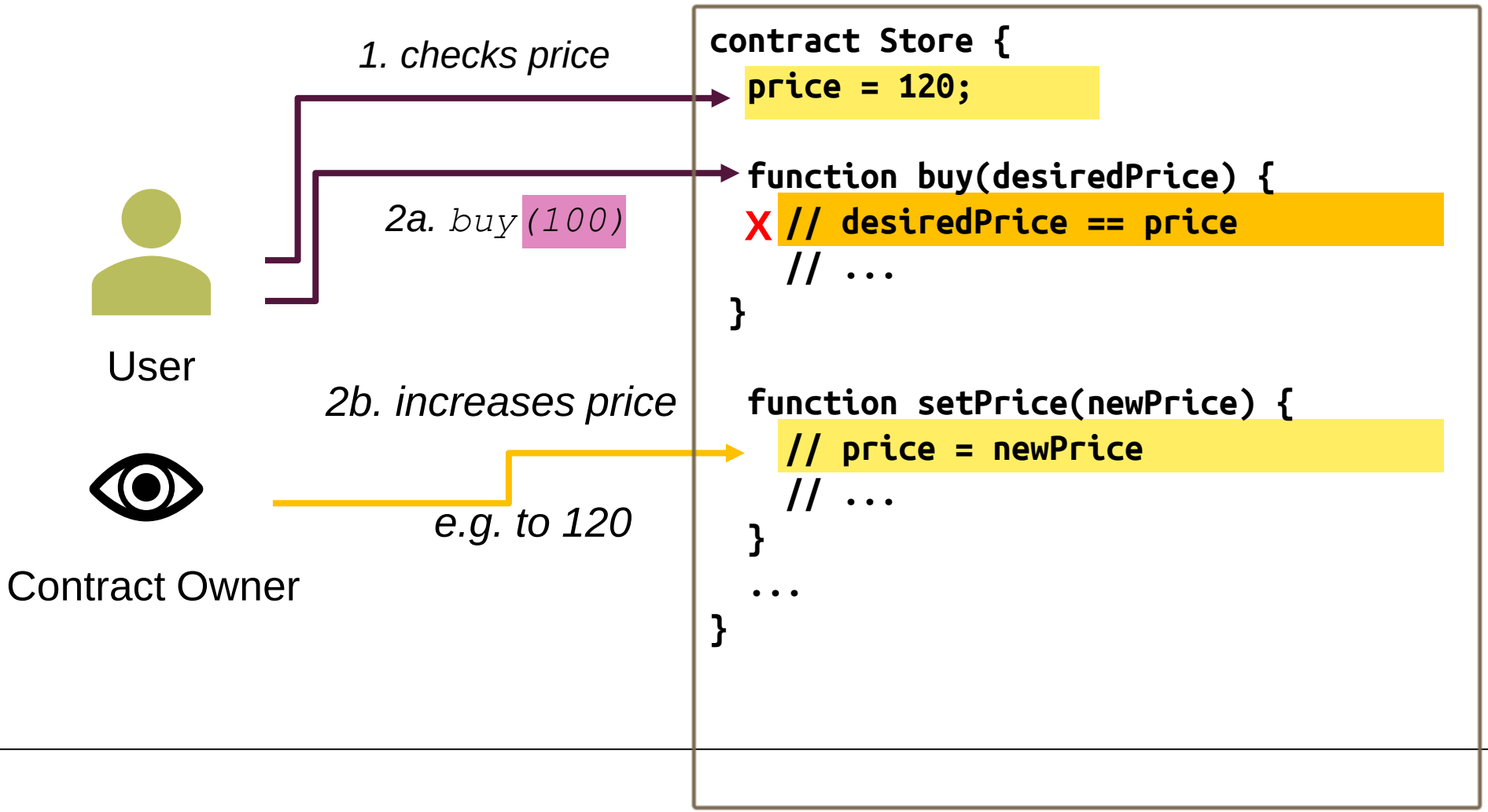


If **2b** is mined before **2a**, user will overpay for purchase

Transaction Order Guard



Transaction Order Guard

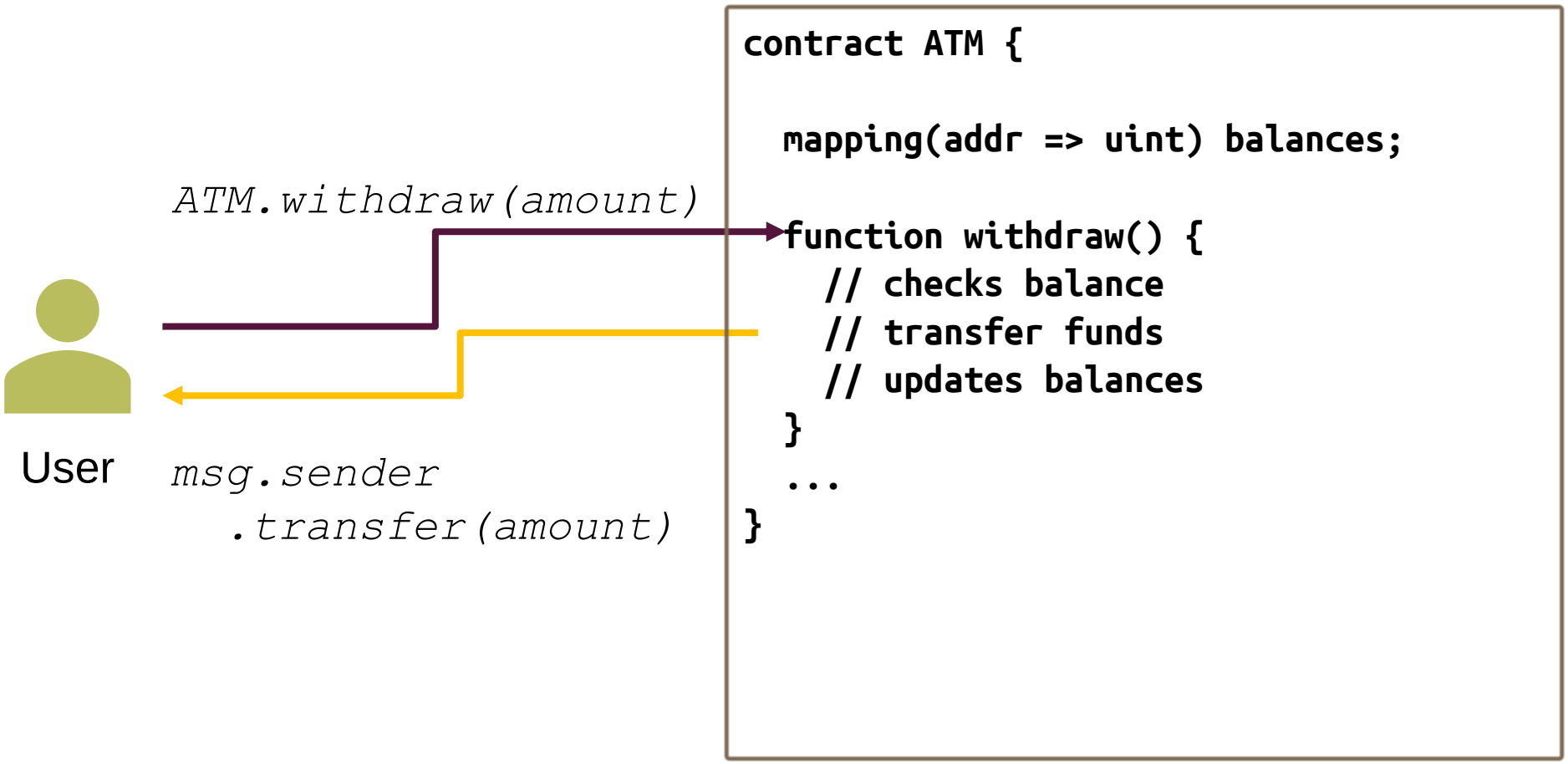


Smart Contract Security

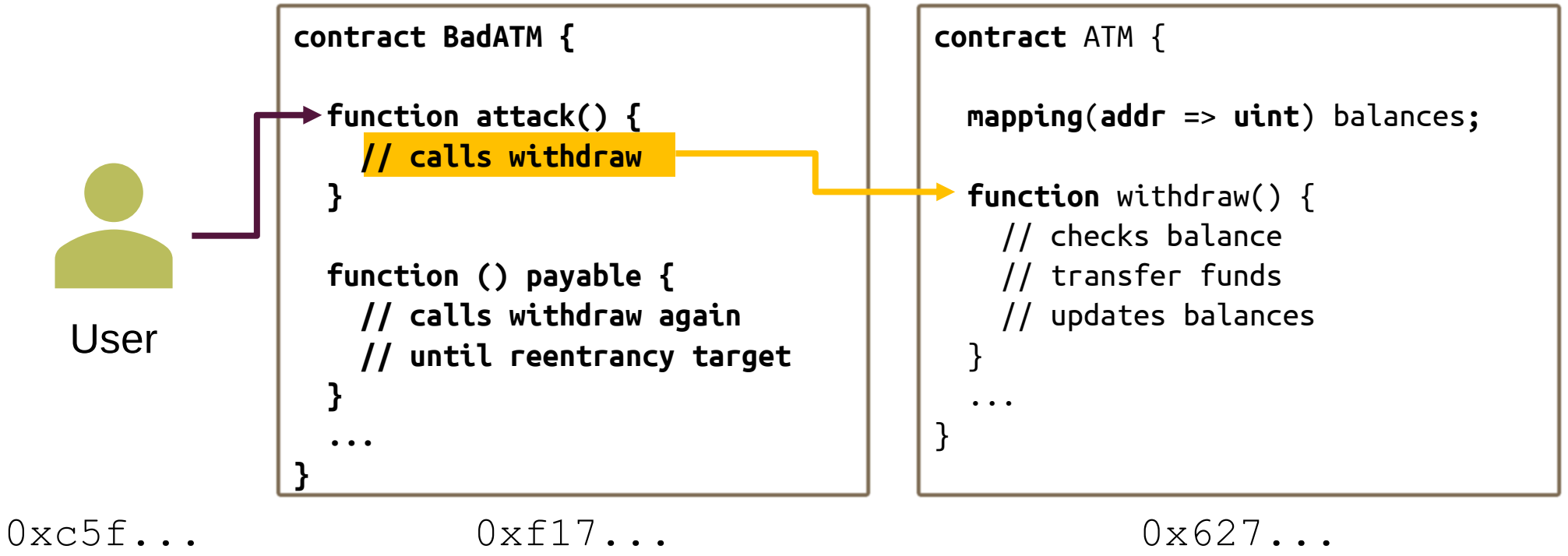
- **Transaction Ordering**
  > Blockchain Shop

- **Reentrancy Attacks**
  > Good ATM | Bad ATM

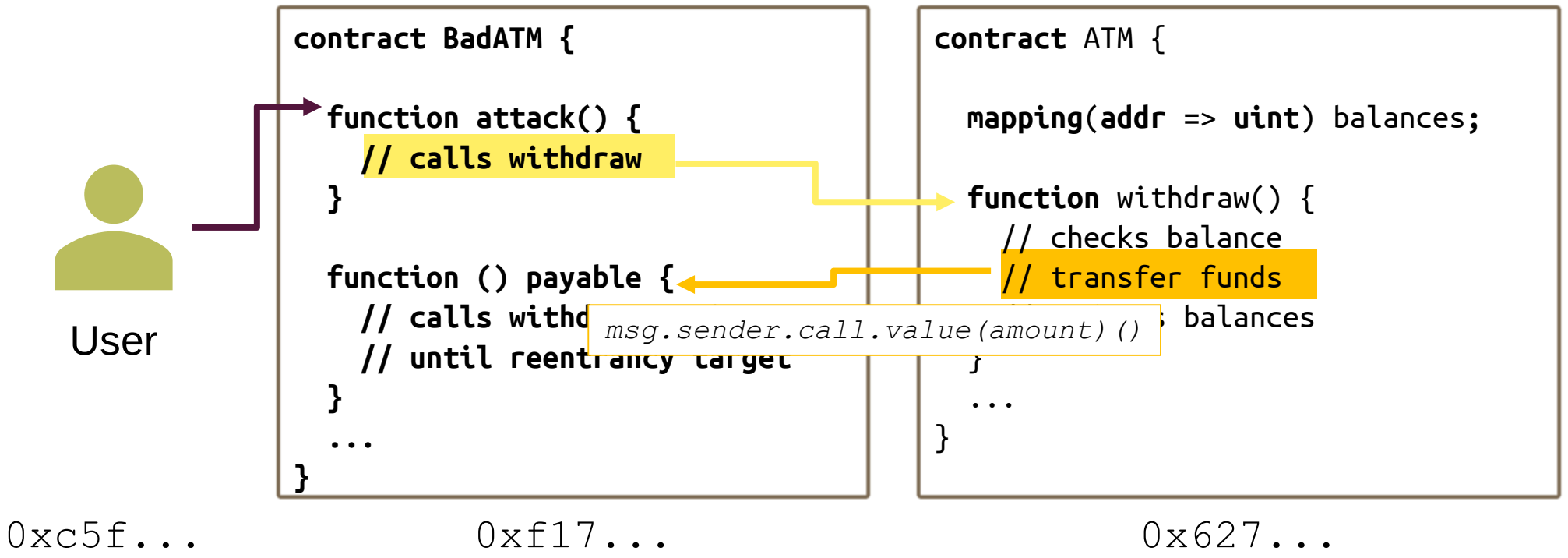
ATM Contract



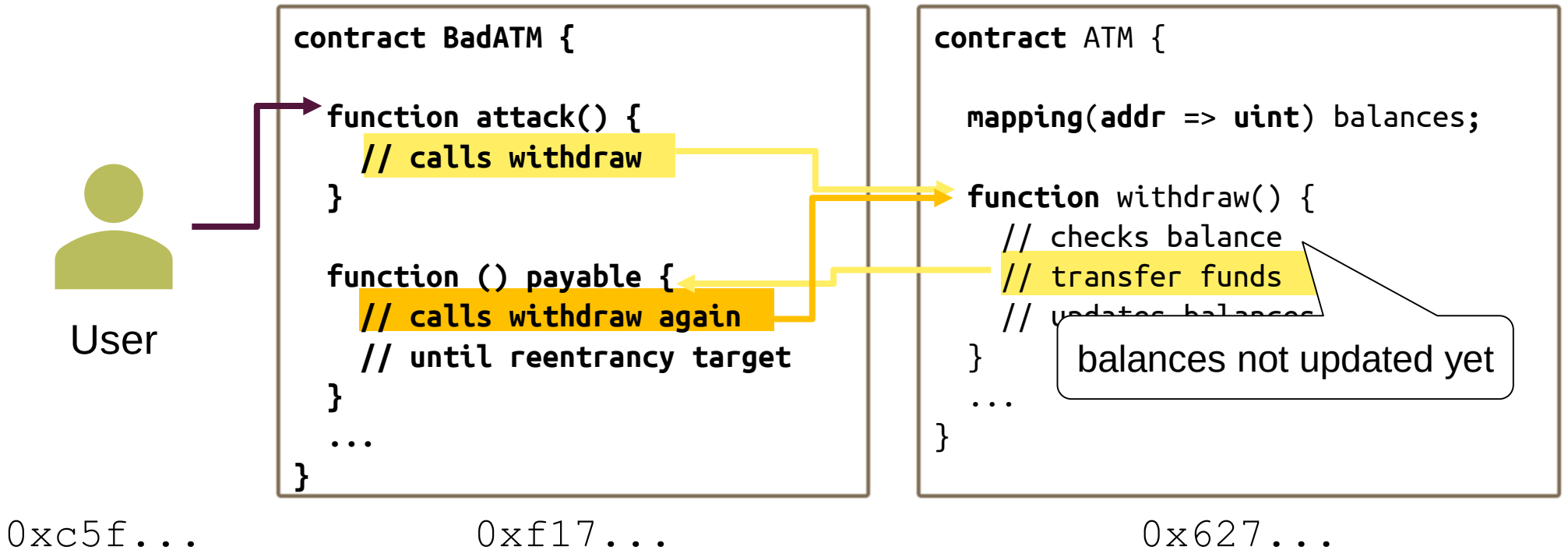
Reentrancy Attack



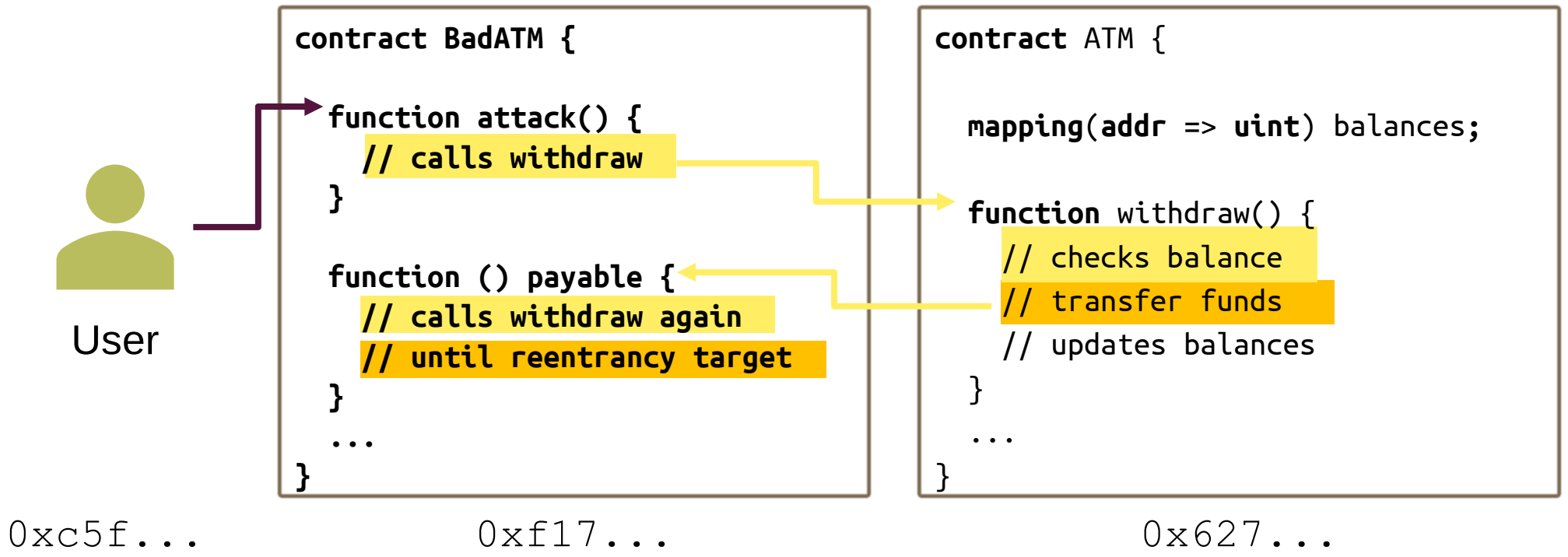
Reentrancy Attack



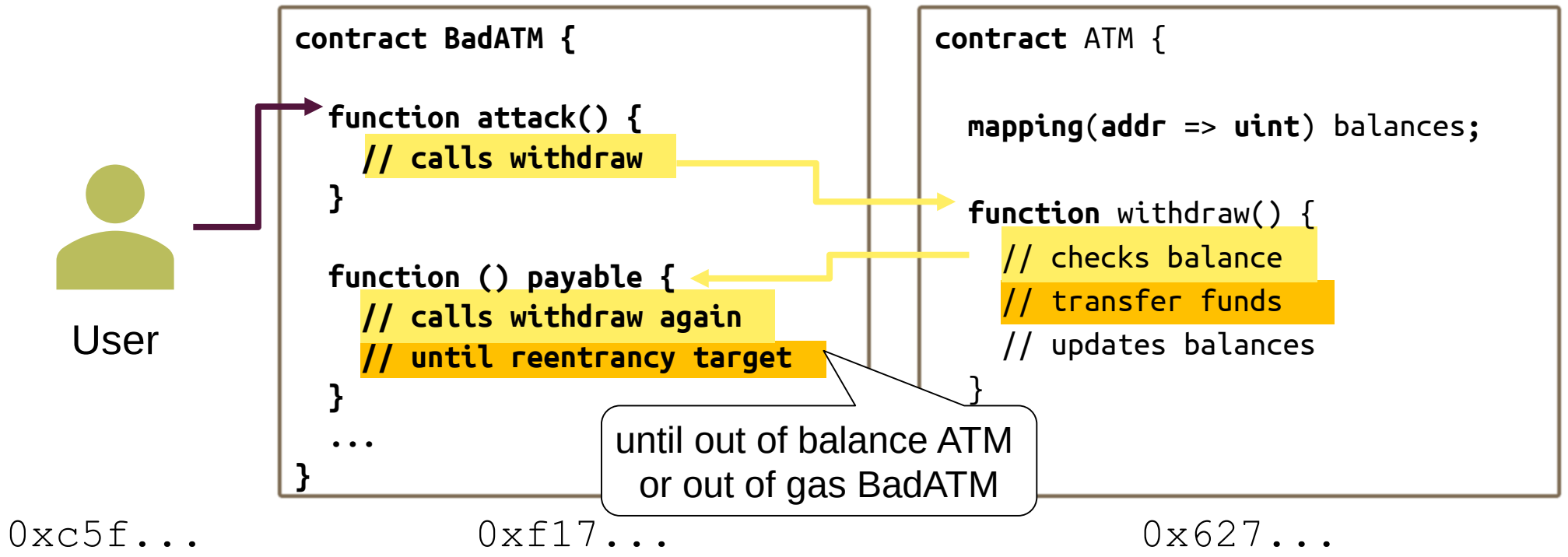
Reentrancy Attack



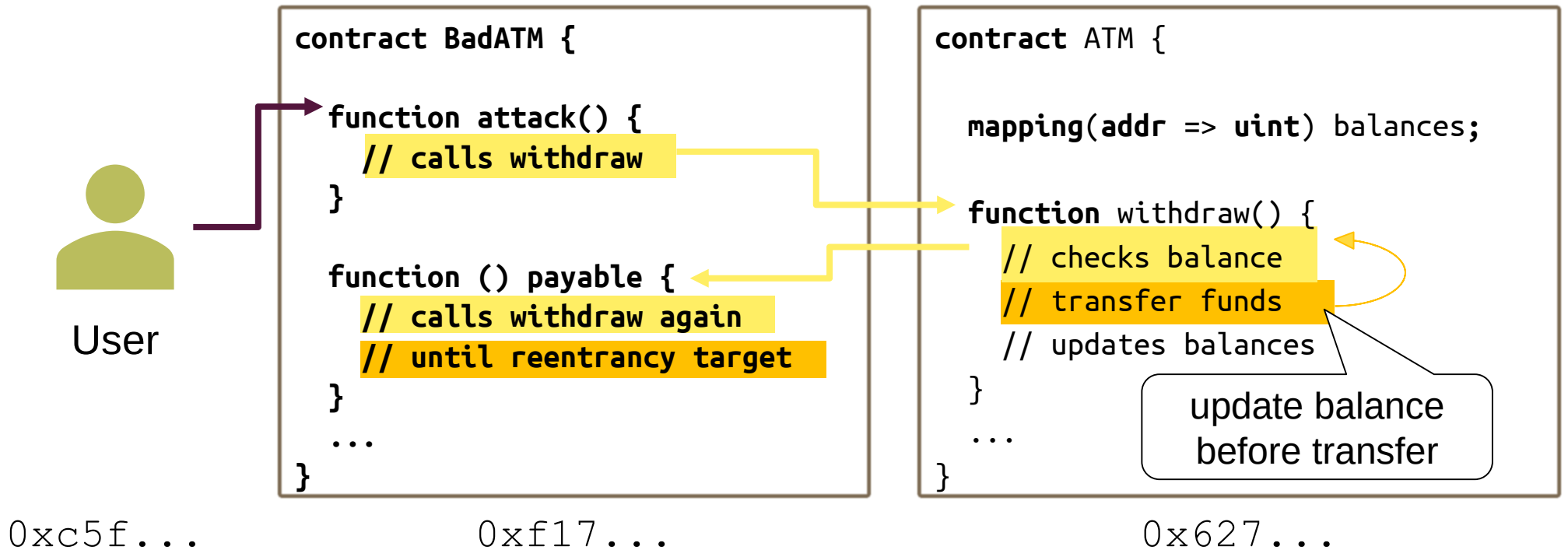
Reentrancy Attack



Reentrancy Attack



Reentrancy Attack



Contract Locks

User

0xc5f...

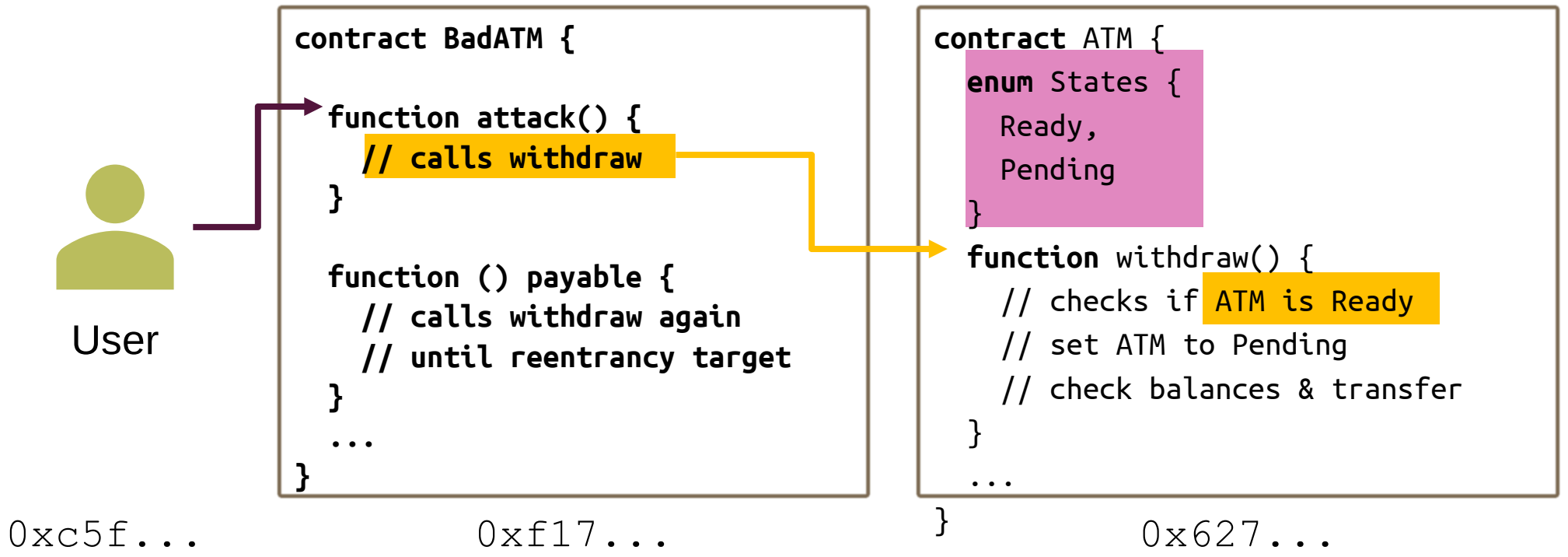
```
contract BadATM {

  function attack() {
    // calls withdraw
  }

  function () payable {
    // calls withdraw again
    // until reentrancy target
  }
  ...
}
```

0xf17...

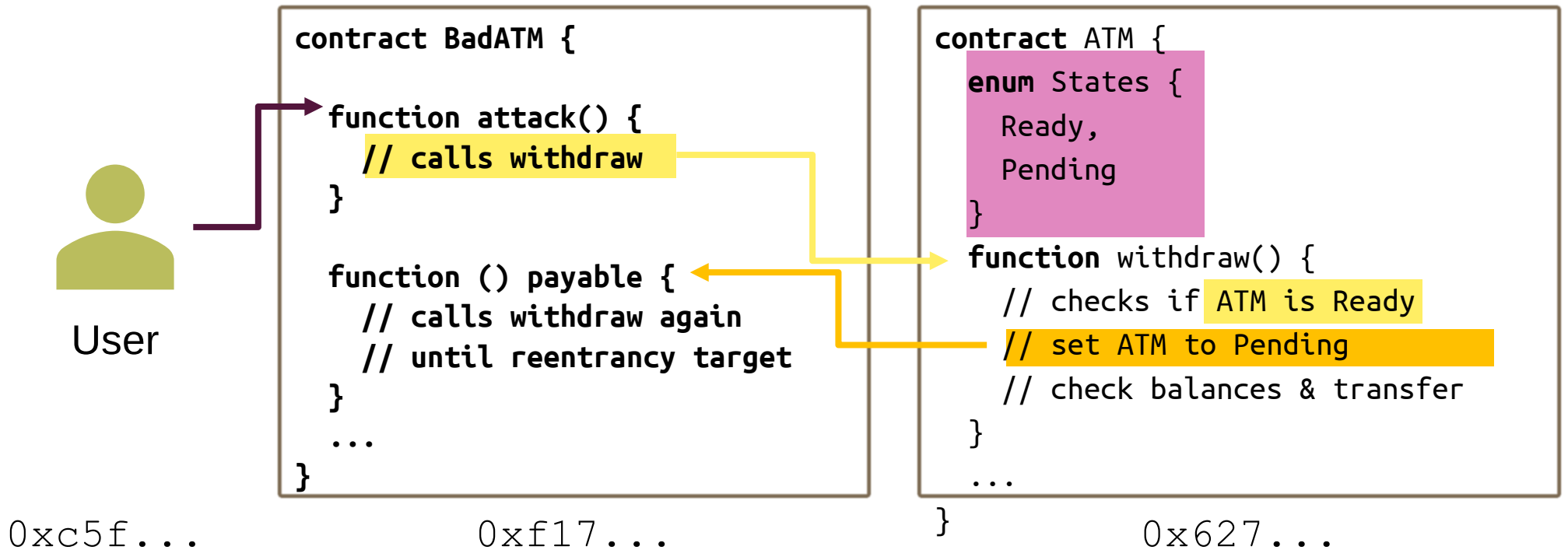
```
contract ATM {
  enum States {
    Ready,
    Pending
  }
  function withdraw() {
    // checks if ATM is Ready
    // set ATM to Pending
    // check balances & transfer
  }
  ...
}
```

0x627...

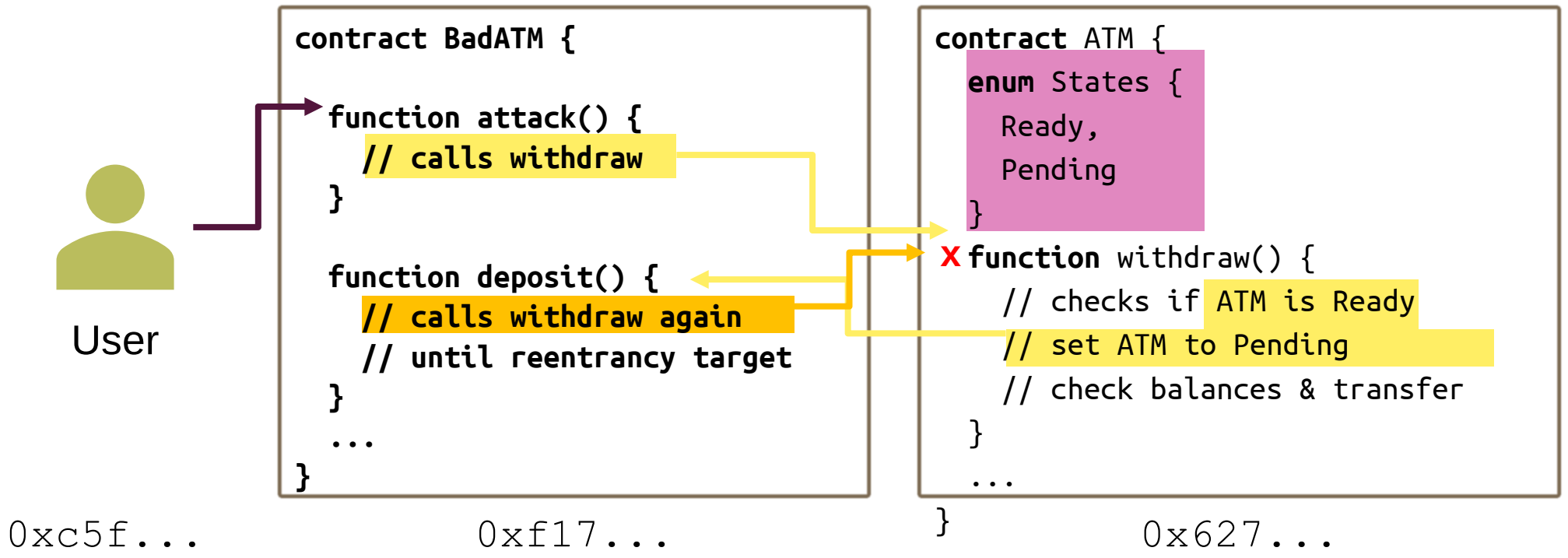
Contract Locks



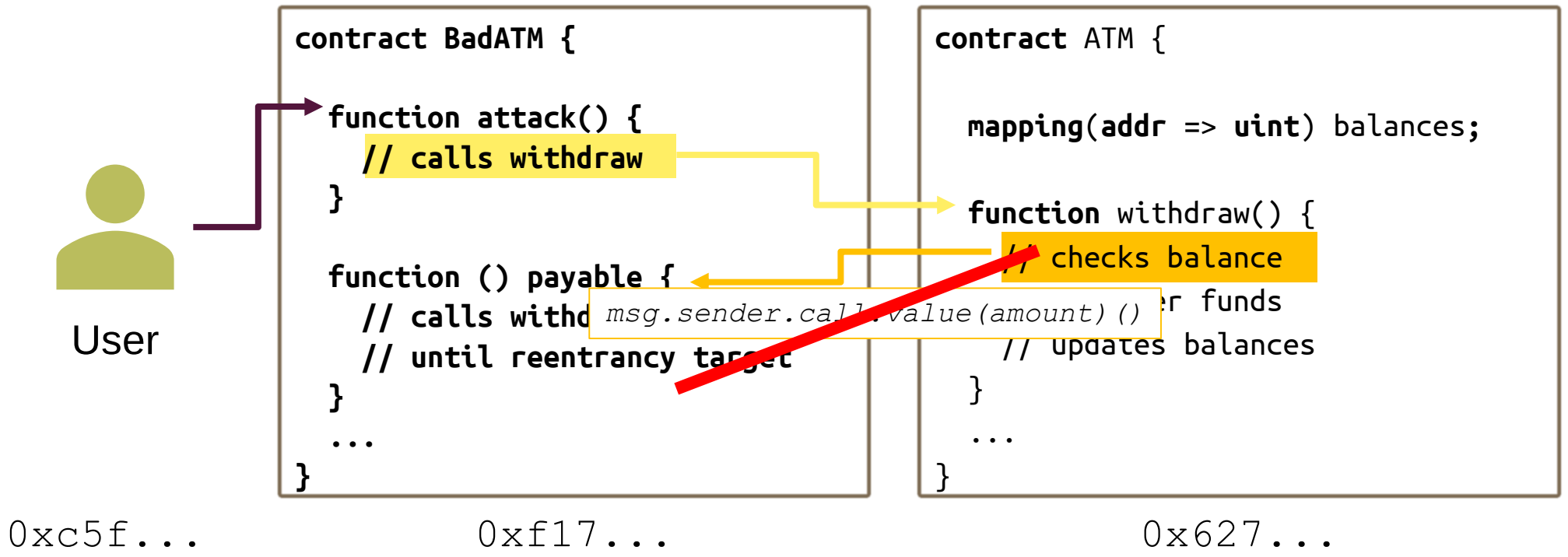
Contract Locks



Contract Locks

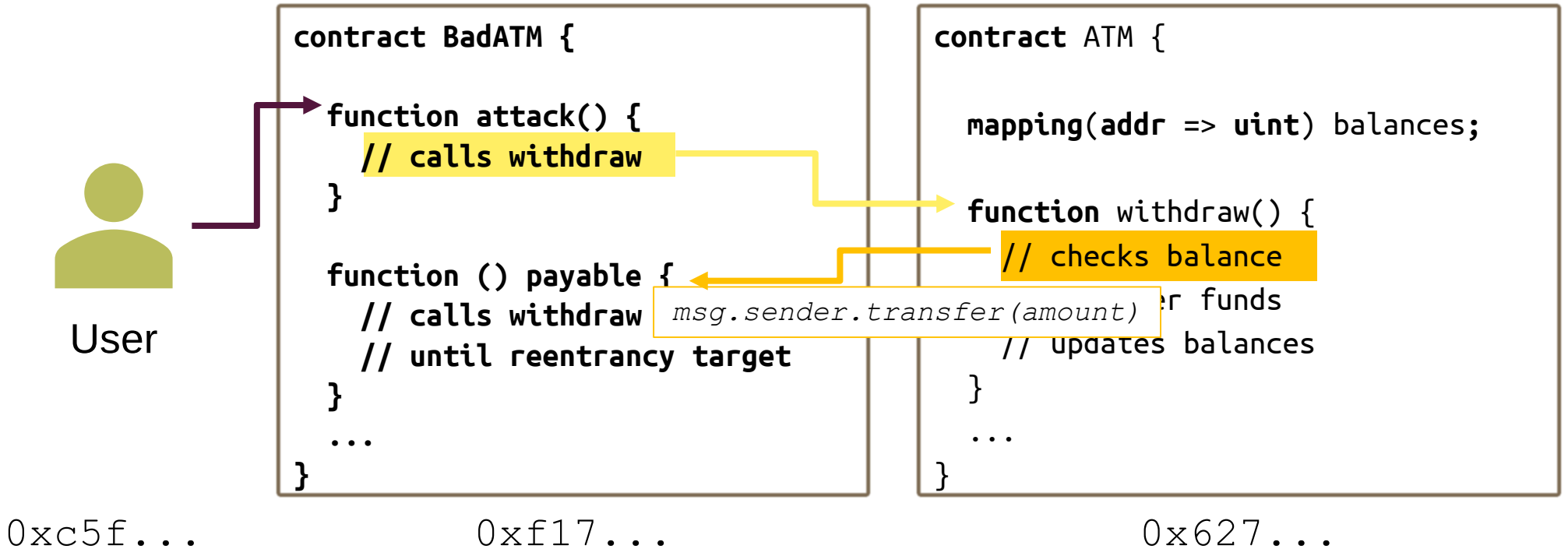


Reentrancy Attack



The transfer-Function

- **Only a small amount of gas is sent along (2300 gas).**
- **The receiver can only emit one single event at max, safe by “accident”.**
- **Remember 2nd slide: don’t use transfer(), pay attention to reentrancy!**



In Conclusion - Best Practices

- **Prepare for failure**
- **Rollout carefully**
- **Keep contracts simple**
- **Stay up to date**
- **Be aware of blockchain properties**
- **Others**
  - Safe Math (Overflow)
  - Error Handling (Revert / Require / Throw)
  - Best Practices e.g. Recommendation for Smart Contract Security in Solidity

URL: b.socrative.com/student/

Room: HSR

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)

```
function withdraw(uint _amount) {
        require(balances[msg.sender] >= _amount);
        msg.sender.call.value(_amount)();
        balances[msg.sender] -= _amount;
}

Contract AA {
function attack() {
        A a = A(addressOfA);
        a.withdraw(100);
}
function () payable {
        A a = A(addressOfA);
        a.withdraw(100);
}
}
function initContract() public {
        owner = msg.sender;
}

function withdraw(uint _amount) {
        require(balances[msg.sender] - _amount > 0);
        msg.sender.transfer(_amount);
        balances[msg.sender] -= _amount;
}
```

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)
  4. Unchecked Return Values For Low Level Calls
  5. Denial of Service (Parity 500K, 300M)
  6. Bad Randomness (400K)
     - Future blockhash as random source, but miner can influence it.

```
 function withdraw(uint256 _amount) public {
         require(balances[msg.sender] >= _amount);
         balances[msg.sender] -= _amount;
         etherLeft -= _amount;
         msg.sender.send(_amount);
 }

function getEtherBalanceIndex(address _address) internal
returns (uint256) {
        for (uint256 i = 0; i < investors.length; i++) {
            if (investors[i] == _address) {
                return i;
            }
        }
        return investors.length;
    }

function play() public payable {
        require(msg.value >= 1 ether);
        if (block.blockhash(blockNumber) % 2 == 0) {
                msg.sender.transfer(this.balance);
        }
}
```

Security Considerations

- **List of the top smart contract vulnerabilities**
  1. Reentrancy (DAO 3.5M, 50M)
  2. Access Control (Parity 150K, 30M)
  3. Arithmetic Issues (DAO)
  4. Unchecked Return Values For Low Level Calls
  5. Denial of Service (Parity 500K, 300M)
  6. Bad Randomness (400K)
  7. Front-Running
  8. Time Manipulation - A malicious miner sets own timestamp ~ 15s
  9. Short Address Attack – not yet exploited
  10. Unknown Unknowns

1. A smart contract publishes an RSA number (N = prime1 x prime2).
2. A call to its submitSolution() public function with the right prime1 and prime2 rewards the caller.
3. Alice successfully factors the RSA number and submits a solution.
4. Someone on the network sees Alice's transaction (containing the solution) waiting to be mined and submits it with a higher gas price.
5. The second transaction gets picked up first by miners due to the higher paid fee. The attacker wins the prize.

We believe more security audits or more tests would have made no difference. The main problem was that reviewers did not know what to look for.

Christoph Jentzsch

URL: b.socrative.com/student/

Room: HSR

ERC20 - Considerations

- **ERC865: Pay transfers in tokens instead of gas, in one transaction**
  - Delegate transfer of tokens to a third party
  - UX: pay service in tokens – user needs to have ETH **and** tokens
    - Create account at exchange, KYC, Bank transfer – and wait
- **Client could be offchain, third party not**
  - Make sure the signature is unique
- **ERC677**
  - Add delegatedTransferAndCall()

```
function delegatedTransfer(
          uint256 _nonce,
          address _from,
          address _to,
          uint256 _value,
          uint256 _fee,
          uint8 _v,
          bytes32 _r,
          bytes32 _s) public returns (bool) {

uint256 total = _value.add(_fee);
require(_from != address(0));
require(_to != address(0));
require(total <= balances[_from]);
require(_nonce > nonces[_from]);

address delegate = msg.sender;
address token = address(this);
bytes32 delegatedTxnHash = keccak256(delegate, token, _nonce, _from, _to,
_value, _fee);
address signatory = ecrecover(delegatedTxnHash, _v, _r, _s);
require(signatory == _from);

balances[_from] = balances[_from].sub(total);
balances[_to] = balances[_to].add(_value);
balances[delegate] = balances[delegate].add(_fee);
nonces[_from] = _nonce;

DelegatedTransfer(_from, _to, delegate, value, fee);
return true;
}
```

EVM Considerations - Random Numbers

- **Random Numbers**
  - There is no random number in Ethereum
  - Every EVM needs to come to the same result
- **Random Numbers from Oracle (External source)**
  - Or: commitment schemes
- **Or: use future blockhash**
  - Only up to 256 blockhashes from the past can be accessed.
  - Deduct / add tokens/ETH from the past address
  - Miner can influence the random value in a sense
    - Value needs to be low (miner gets 2 ETH)

```
function transfer(address _to, uint256 _value) public returns (bool) {
     …
        //since this is a lucky coin, the value transferred is not what you
expect
        luckyTransfer();
        previousTransferBlockNr = block.number;
        previousTransferAddress = msg.sender;
        …
        uint256 val = _value.sub(potIncrease);
       pot = pot.add(potIncrease);
}

function luckyTransfer() private {
        if(block.number != previousTransferBlockNr && (block.number -
previousTransferBlockNr) < 256 ) {
            uint256 rnd =
uint256(keccak256(block.blockhash(previousTransferBlockNr)));
            if(rnd % 200 == 0) { //.5%
                balances[previousTransferAddress] =
balances[previousTransferAddress].add(pot);
                Transfer(this, previousTransferAddress, pot);
                //tokens are from pot, thus no tokens are created from thin air
                pot = 0;
                previousTransferBlockNr = 0;
                previousTransferAddress = 0;
            }
        }
}
```

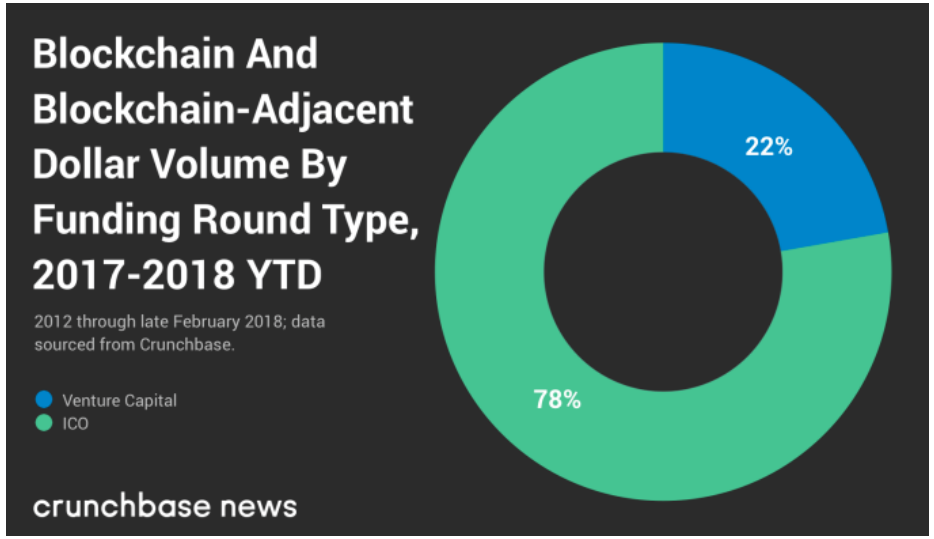
URL: b.socrative.com/student/

Room: HSR

What is an ICO

- **1st Blockchain Killer App (not anymore)**
  - Lots of tokens in Solidity / Ethereum, ERC20
- **An initial coin offering (ICO) is a means of crowdfunding**
  - Tokensale - release of a new cryptocurrency
  - 1490 listed cryptocurrencies with a marketcap, 4942 with coinlib.io
- **ICOs vs IPOs**
  - ICO little regulation, IPO are heavily regulated
  - ICO short in duration, IPO much longer
  - IPO are exclusive, ICO are open to anyone
- **TechCrunch: Over 1000 Crypto Projects Are Considered 'Dead' Now**

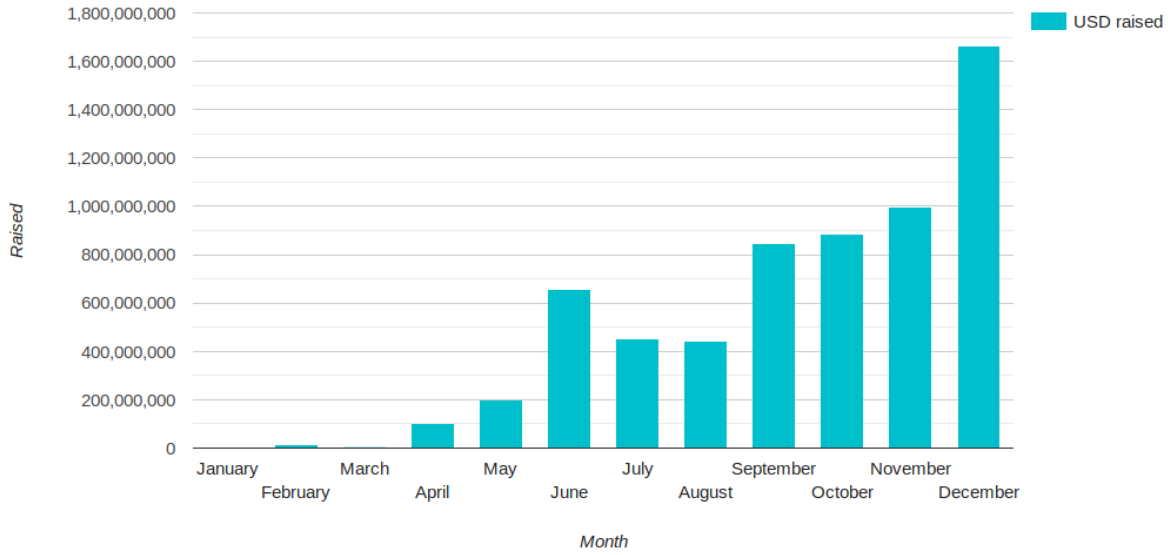
- **ICOs delivered at least 3.5x more capital to blockchain startups than VC since 2017**



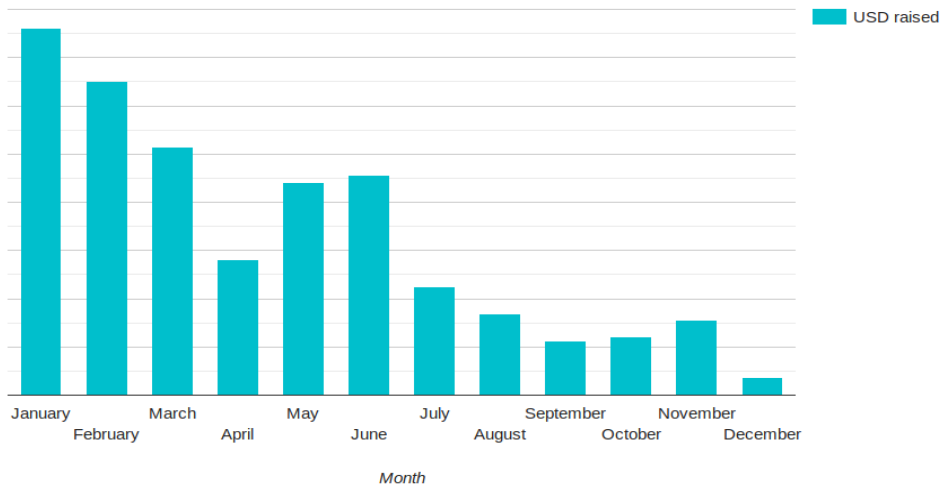
- **Wall Street Journal: 20 percent of all ICOs are fraudulent**
- **SATIS Group Report: '78% of ICOs are Scams'**

Total amount raised by all ICOs in total ($)

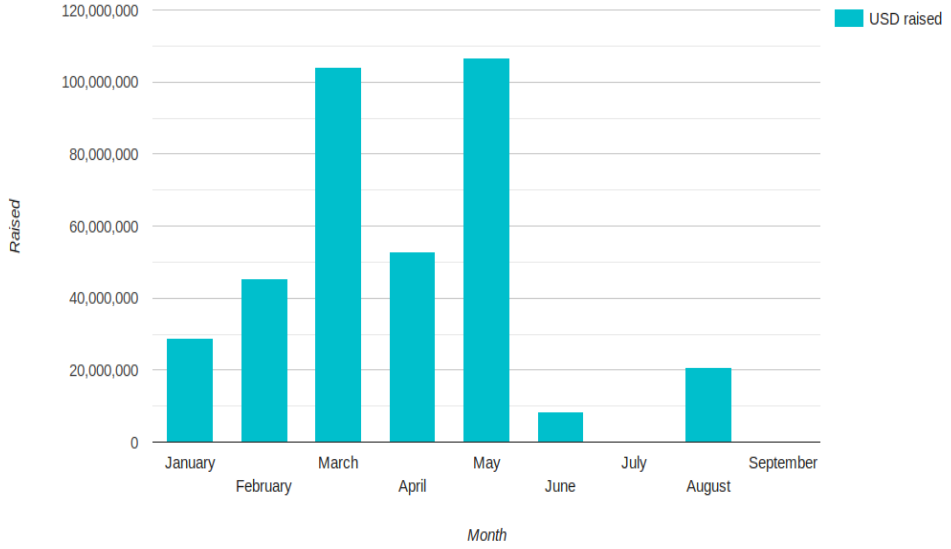
2017



2018



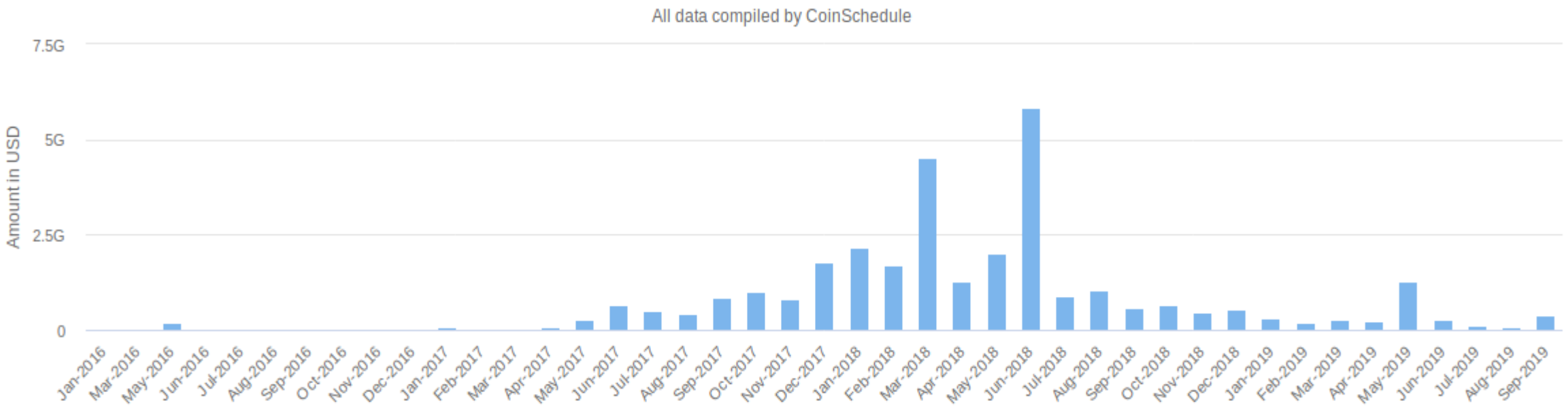
2019



Source: icodata.io

Total amount raised by all ICOs in total ($mln)

Source: Coinschedule



ICO countries and categories until 2018

Number of projects

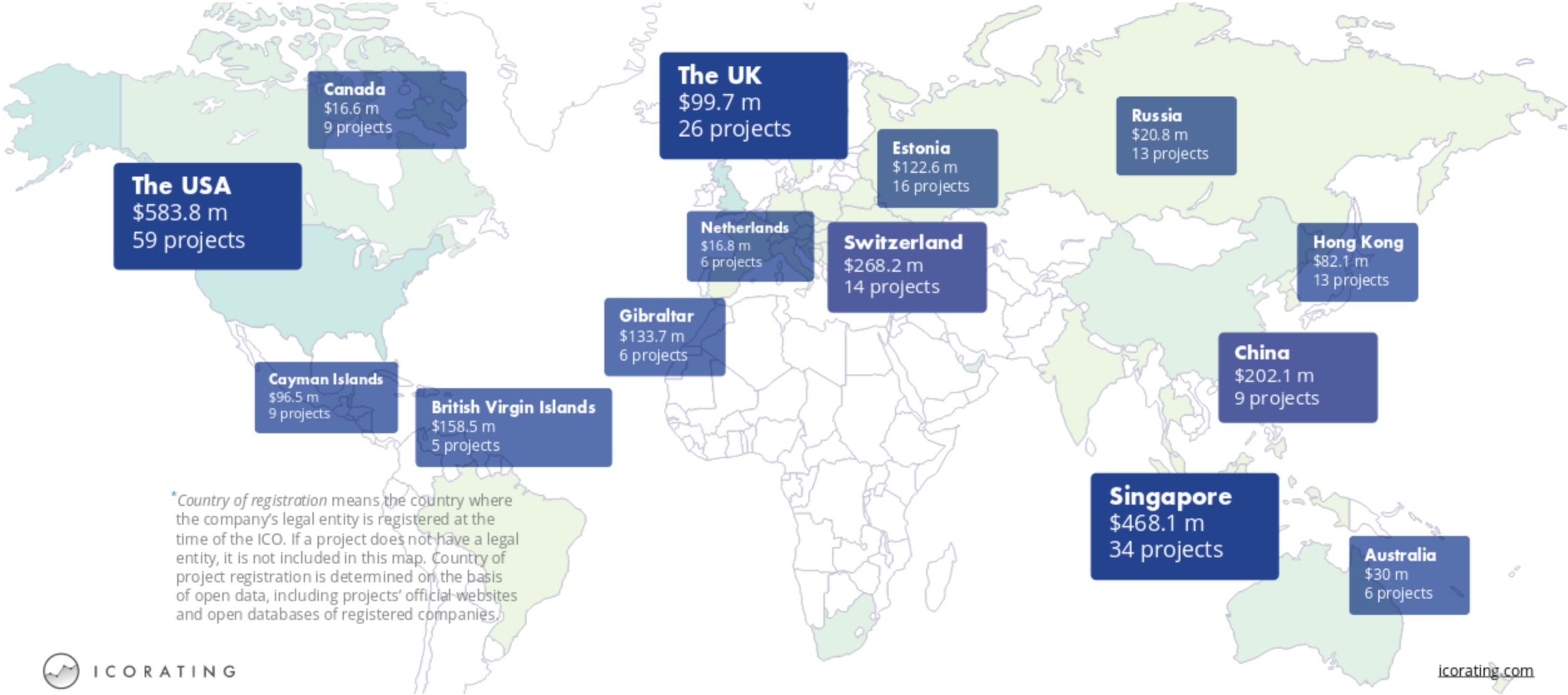
	Country	% of Projects	Total Raised
1	USA	15.94%	875m $
2	UK	9.6%	184m $
3	Singapore	8.3%	340m $
4	Russia	7.02%	1003m $
5	Switzerland	6.58%	547m $
6	Estonia	4.09%	34m $
7	Australia	3.36%	31m $

Total Raised

	Country	% of Projects	Total Raised
1	Russia	7.02%	1003m $
2	USA	15.94%	875m $
3	Switzerland	6.58%	547m $
4	Singapore	7.02%	340m $
5	UK	9.6%	184m $
6	Estonia	4.09%	34m $
7	Australia	3.36%	31m $

Source: icowatchlist

ICO countries and categories



Questions?

Dr. Thomas Bocek thomas.bocek@hsr.ch