# Making smart contract smarter

Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, Aquinas Hobor

**<EE817/IS893: Blockchain and Cryptocurrency>**

Presented by Daejun Kim (2019. 05)

# Index

- Background
- Introduction
- Security bugs in Ethereum
- Towards a better design
- The $Oyente$ Tool (compare with teEther)
- Conclusion
- Future Works
- Appendix

# Background

# Trend

- Academic Pedigree

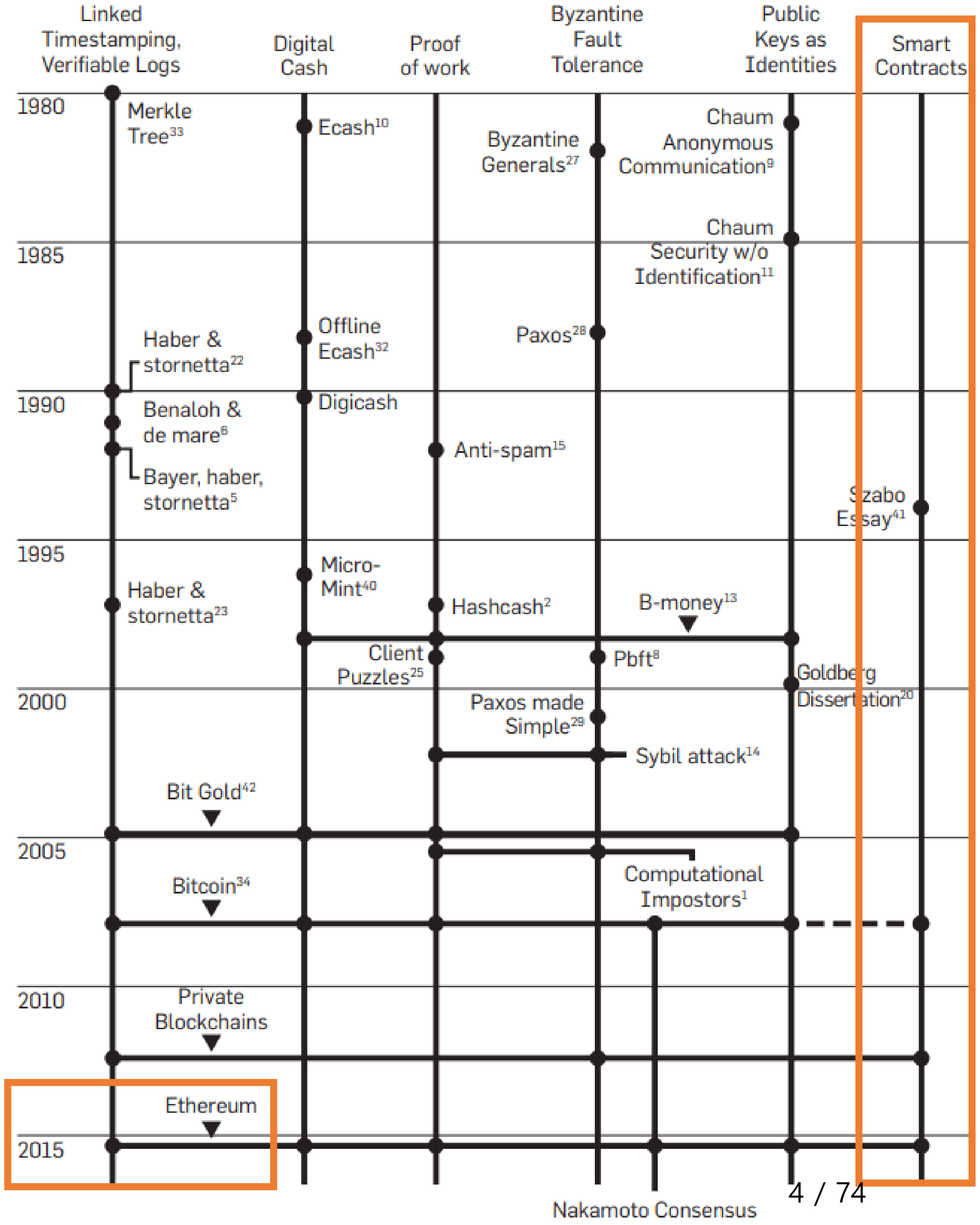



*Image from Narayanan, Arvind, and Jeremy Clark. "Bitcoin's academic pedigree." *Communications of the ACM* 60.12 (2017): 36-45.

# Trend

[2016]

- Luu, Loi, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena and Aquinas Hobor. "**Making smart contracts smarter.**" ACM CCS.

[2017]

- Trailofbits, https://github.com/trailofbits/**manticore**

- Trailofbits, https://github.com/ConsenSys/**mythril-classic**

# Trend

[2018] - Cont'd

- Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Mi chael Bailey. "**Erays:** reverse engineering ethereum's opaque smart contracts.", USENIX

- Sukrit Kalra, Seep Goel, Mohan Dhawan and Subodh Sharma. "**Zeus:** Analyz ing safety of smart contracts.", NDSS

- Krupp Johannes, and Christian Rossow. "**teether:** Gnawing at ethereum to a utomatically exploit smart contracts.", USENIX

# Trend

[2018]

- Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., & Vechev, M. “**Securify:** Practical security analysis of smart contracts.” ACM SIGSAC

- Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., & Alexandrov, Y. “**Smartcheck:** Static analysis of ethereum smart contracts.”. *WETSEB*

# Trend

- Symbolic Execution (This paper also uses the same methodology.)
- Slow.... But, targeting smart contracts is fast!

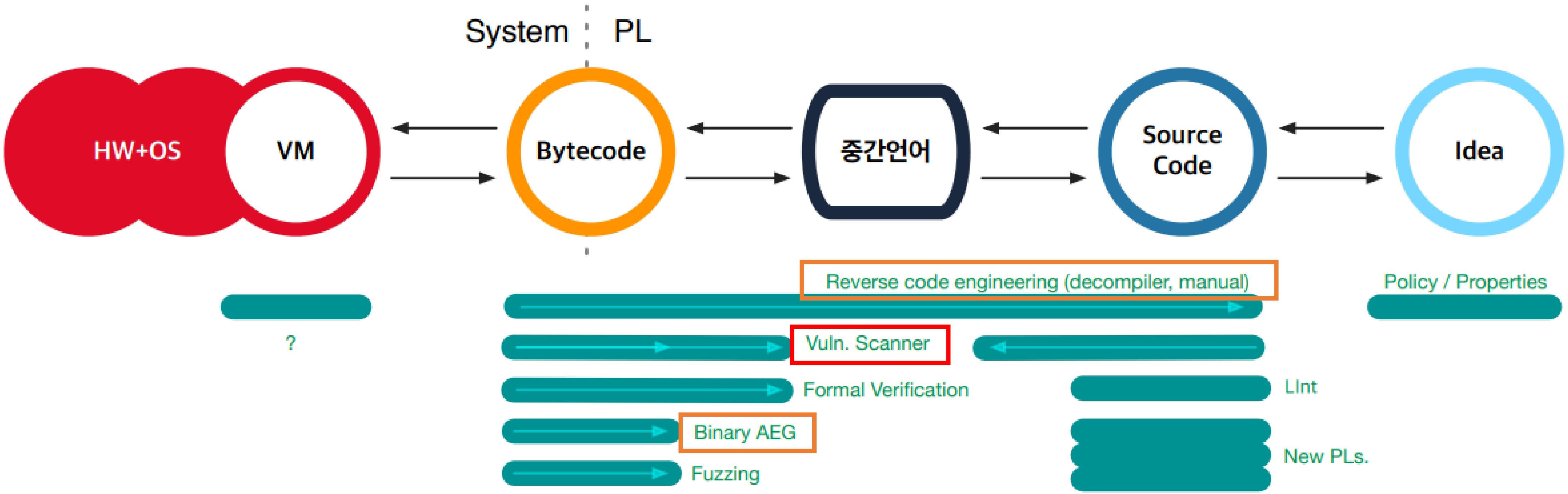



*Image from "Smart Contract 분석과 PL", Jonghyup Lee

# Ethereum

| | | | |
|---|---|---|---|
| Issued date | 2015. 07. | Market capitalization | ≈$18 billion (2019. 04) |
| Block Time | About 12 seconds | Block reward | 5 ETH (Ethereum) |
| Consensus Algorithm | PoW | | |

- "Ethereum is an open blockchain platform that lets anyone build and use decentralized applications that run on blockchain technology." (aka. 2nd generation cryptocurrency)

It can be a platform! <Smart contract>

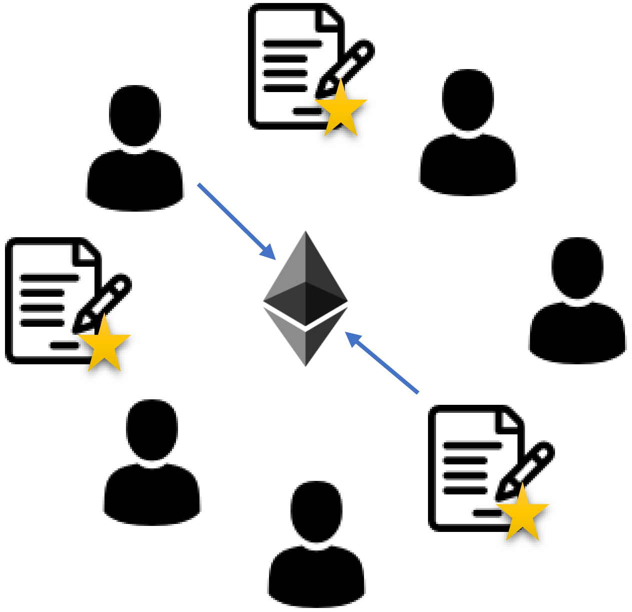
*Market capitalization from Coinmarketcap (https://coinmarketcap.com)

# Smart contract

- "A smart contract is a computerized transaction protocol that executes the terms of a contract." (Szabo, Nick. "Smart contracts." *Unpublished manuscript* (1994))

- Today, this is also called DApp (Decentralized application, Distributed application)

# Smart contract



- In Ethereum (Cont'd)
  - This program is run on block-chain nodes.
  - Executed on incoming transactions
    - from, to, value (ETH amount), gas (fee), data (argv)
    - "Conceptually, Ethereum can be viewed as a transaction based state-machine"
  - Turing complete (Turing, Alan. "On Computable Numbers, with an Application to the Entscheidungs problem, 1936." *B. Jack Copeland* (2004): 58.)

# Smart contract

- In Ethereum
  - Written in solidity
    - object-oriented, high-level language for implementing smart contracts
    - influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).
  - Usage
    - voting, crowdfunding, blind auctions, and multi-signature wallets.
  - Cannot patch 🤔

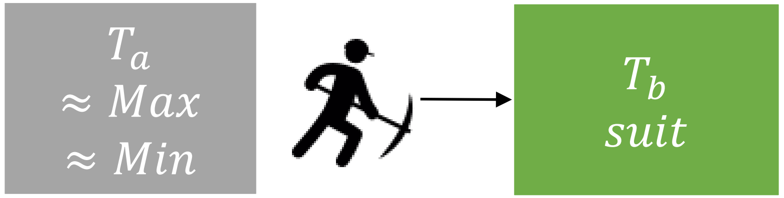
# Smart contract

- Gas (Cont'd)
  - "Gas is a unit that measures the amount of computational effort that it will take to execute certain operations."

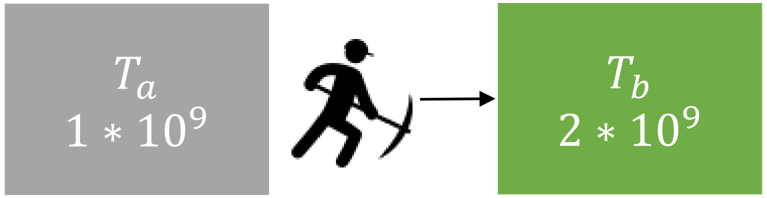
# Smart contract

- Gas (Cont’d)
  - Fee (Gas) = Gas limit * Gas price (FYI. 1 ETH = 1,000,000,000 $gwei$)
  - Gas Limit: Number of gases required for operation
  - Gas Price: Literally, gas price.
    - Affects mining time / order.

If $same$ $Gas\ Price$, Gas Limit comparison




If $same$ $Gas\ Limit$, Gas Price comparison

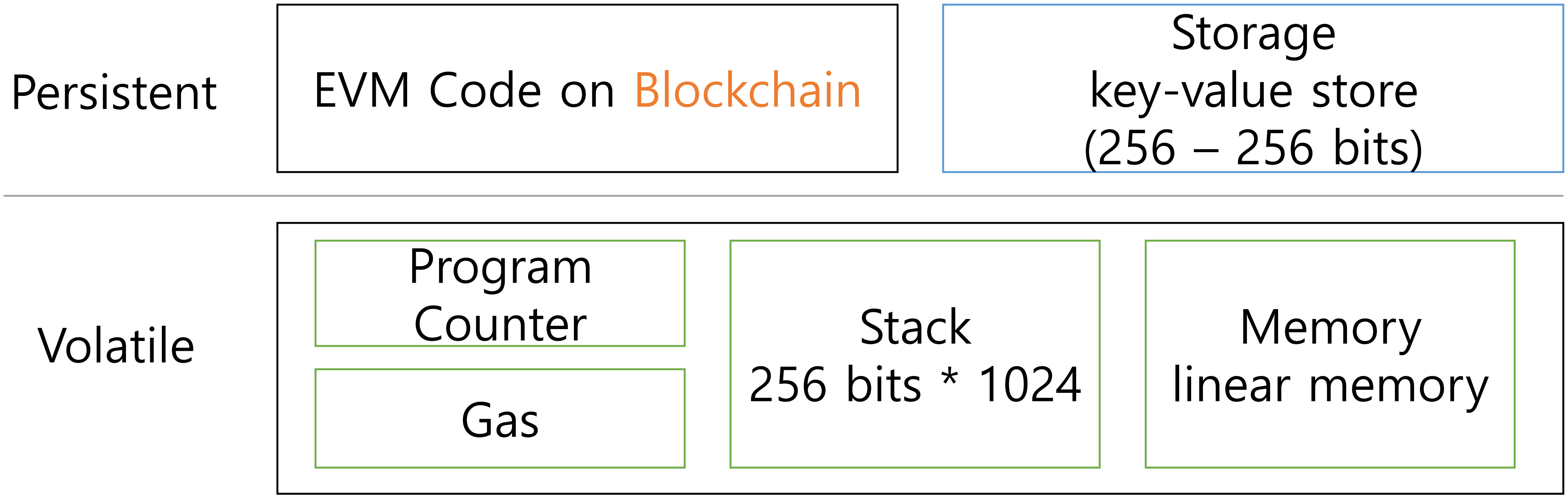


# Smart contract

- Gas
  - But, You do not consume too much gas in one transaction.
    - Block Gas Limit: The sum of the gases that can be contained in a block.
  - If fails, the state (σ) is reverted to the initial state and the sender pays all gas limit to the miner. (counter-measure against resource-exhausting attacks)

# Smart contract

- Ethereum Virtual Machine (EVM)



# Smart contract

- Ethereum Virtual Machine (EVM)
  - No register
  - Stack: PUSH/POP/COPY/SWAP
  - Memory: MSTORE/MLOAD
  - Storage: SSTORE/SLOAD
  - Gas consumes per opcode.

**EVM Code example**

```
Byte Code | Assembly
=====================
6009      | PUSH1 09
34        | CALLVALUE
6007      | PUSH1 07
57        | JUMPI
00        | STOP
5b        | JUMPDEST
56        | JUMP
5b        | JUMPDEST
00        | STOP
```

**Gas consumes**

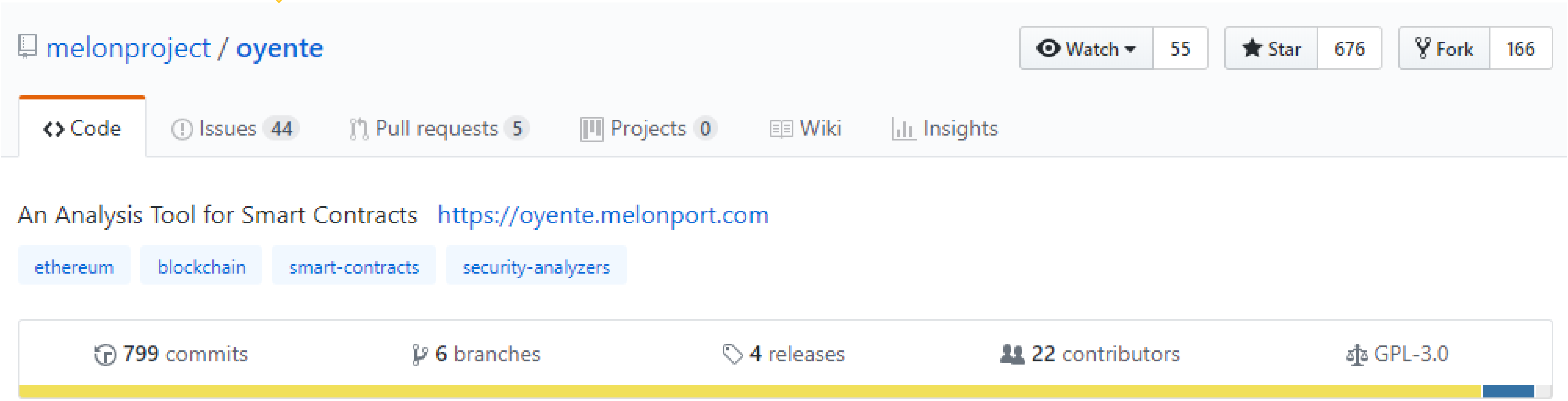
| Mnemonic | Gas Used |
|---|---|
| STOP | 0 |
| ADD | 3 |
| MUL | 5 |
| SUB | 3 |
| DIV | 5 |
| SDIV | 5 |
| MOD | 5 |
| SMOD | 5 |
| ADDMOD | 8 |
| MULMOD | 8 |

*Gas consumes: https://docs.google.com/spreadsheets/d/1n6mRqkBz3iWcOlRem_mO09GtSKEKrAsfO7Frgx18pNU/edit

# Introduction

# Introduction

- Goal & Approach: Finding bugs in Ethereum Smart Contract via symbolic execution tool.



# Introduction

- Contribution
  - Introducing several new classes of security bugs in the Ethereum Smart Contract
  - Formalize the "lightweight" semantics of Ethereum smart contract and propose recommendations as solutions for the documented bugs.
  - make & run $Oyente$, a symbolic execution tool which analyses Ethereum smart contracts to detect bugs, in real Ethereum network.

# Introduction

- Comparison (*Oyente* vs *Zeus*)
  - Kalra, Sukrit, et al. "Zeus: Analyzing safety of smart contracts." 25th Annual Network and Distributed System Security Symposium, NDSS. 2018.

| | |
|---|---|
| Transaction Order Dependence | 8,890 / 19,366 (45.9%, 1,758 unique contract) |
| Block / Transaction state dependence | |
| Unchecked send | |
| Reentrancy | |
| Failed send | 21,281 / 22,493 (94.6%, 1,524 unique contract) |
| Integer overflow / underflow | |

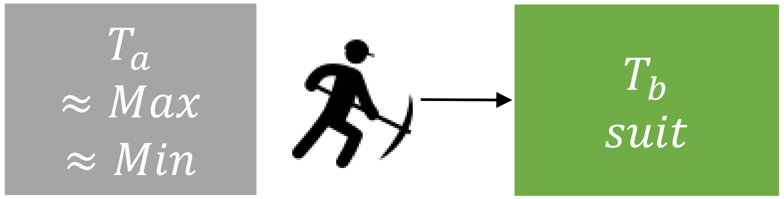
# Security bugs in Ethereum

# Security bugs in Ethereum

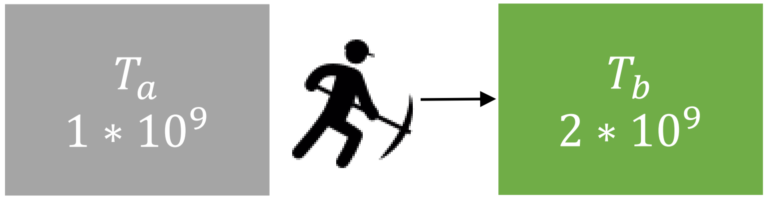
**Attack #1**. Transaction-Ordering Dependence (TOD)

- Did you remember the transaction ordering?

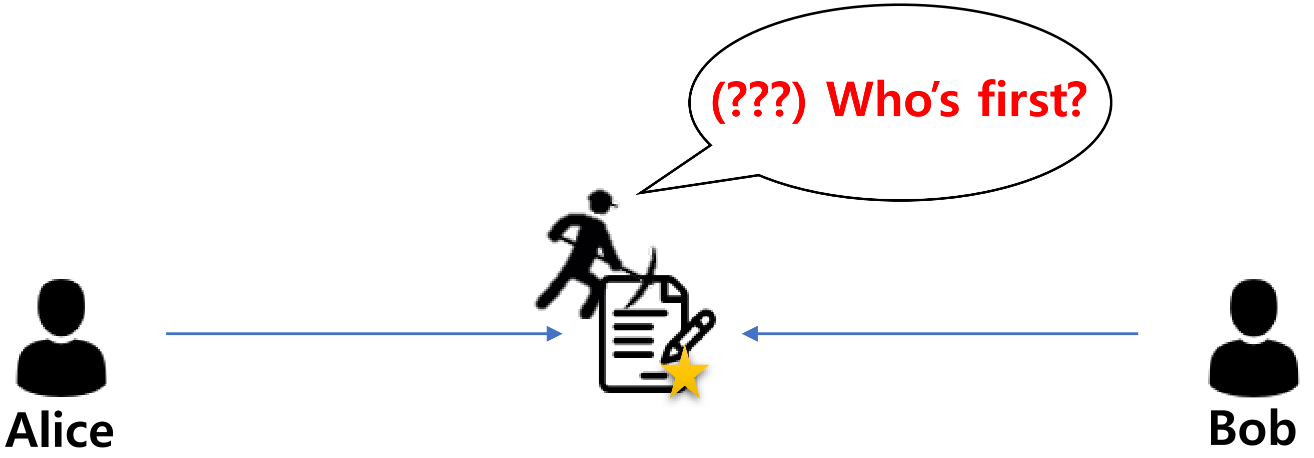
If *same Gas Price*, Gas Limit comparison




If *same Gas Limit*, Gas Price comparison




- OK, Let's think about the following situation.



# Security bugs in Ethereum

```
        if (sha256(msg.data) < diff){
          msg.sender.send(reward); //send reward
```

**Attack #1**. TOD

- Let's take a specific example.

- In this contract, you can **get a reward when you send the right answer.**

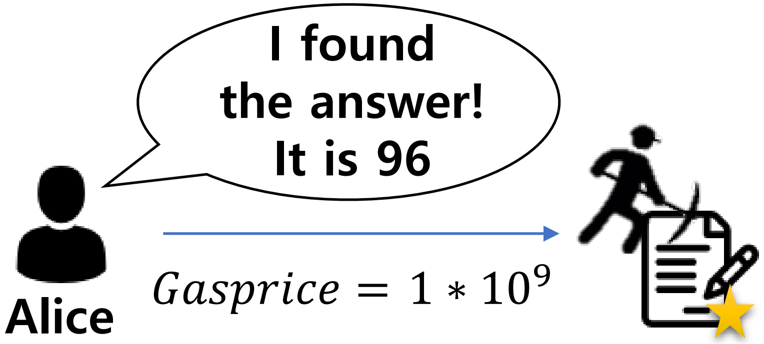
```
uint public reward;
bytes32 public diff;
bytes public solution;

function Puzzle() //constructor{
  owner = msg.sender;
  reward = msg.value;
  locked = false;
  diff = bytes32(11111); //pre-defined difficulty
}

function(){ //main code, runs at every invocation
  if (msg.sender == owner){ //update reward
    if (locked)
      throw;
    owner.send(reward);
    reward = msg.value;
  }
  else
    if (msg.data.length > 0){ //submit a solution
      if (locked) throw;
      if (sha256(msg.data) < diff){
        msg.sender.send(reward); //send reward
        solution = msg.data;
        locked = true;
      }}}
```

Figure 3: A contract that rewards users who solve a computational puzzle.

# Security bugs in Ethereum


```
if (sha256(msg.data) < diff){
  msg.sender.send(reward); //send reward
```


**Attack #1**. TOD - Example



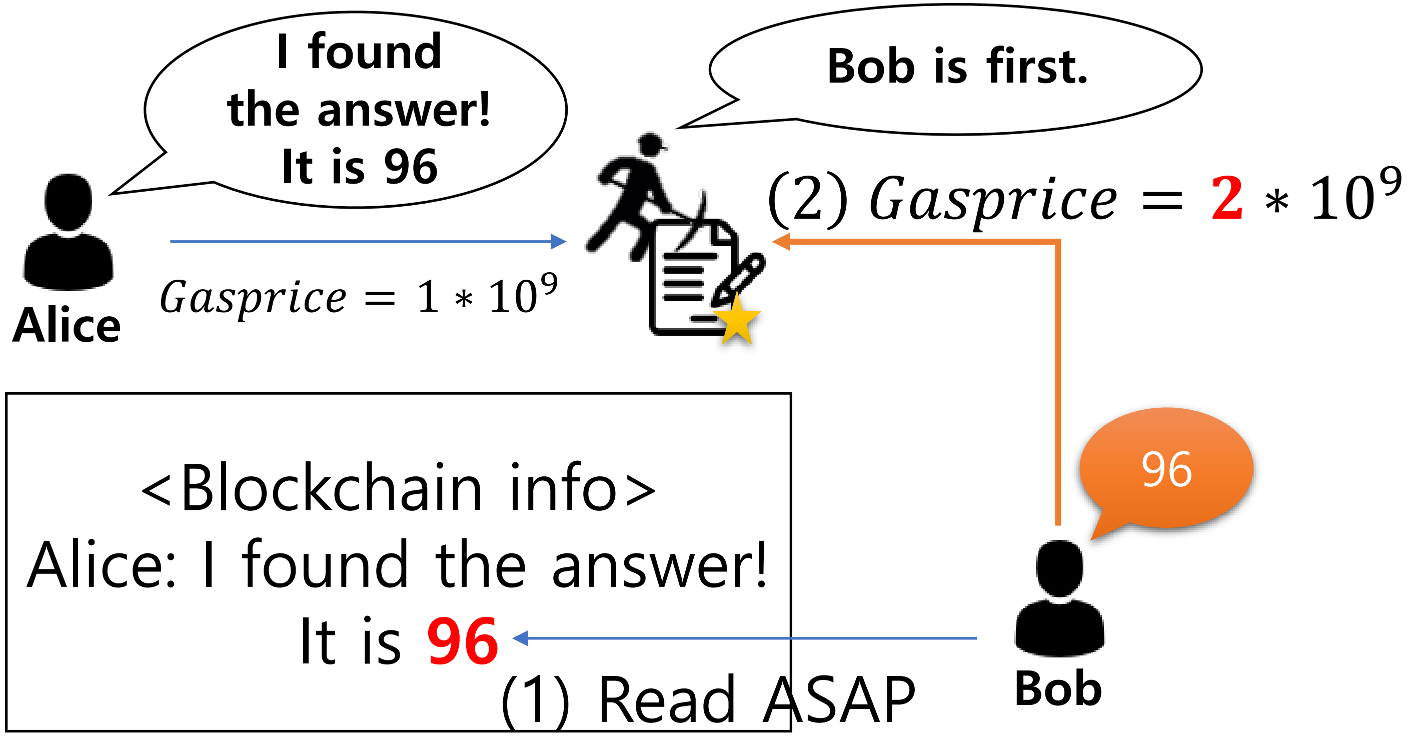

```
uint public reward;
bytes32 public diff;
bytes public solution;

function Puzzle() //constructor{
  owner = msg.sender;
  reward = msg.value;
  locked = false;
  diff = bytes32(11111); //pre-defined difficulty
}

function(){ //main code, runs at every invocation
  if (msg.sender == owner){ //update reward
    if (locked)
      throw;
    owner.send(reward);
    reward = msg.value;
  }
  else
    if (msg.data.length > 0){ //submit a solution
      if (locked) throw;
      if (sha256(msg.data) < diff){
        msg.sender.send(reward); //send reward
        solution = msg.data;
        locked = true;
      }}}
```

Figure 3: A contract that rewards users who solve a computational puzzle.

# Security bugs in Ethereum

```
if (sha256(msg.data) < diff){
  msg.sender.send(reward); //send reward
```

**Attack #1**. TOD - Example



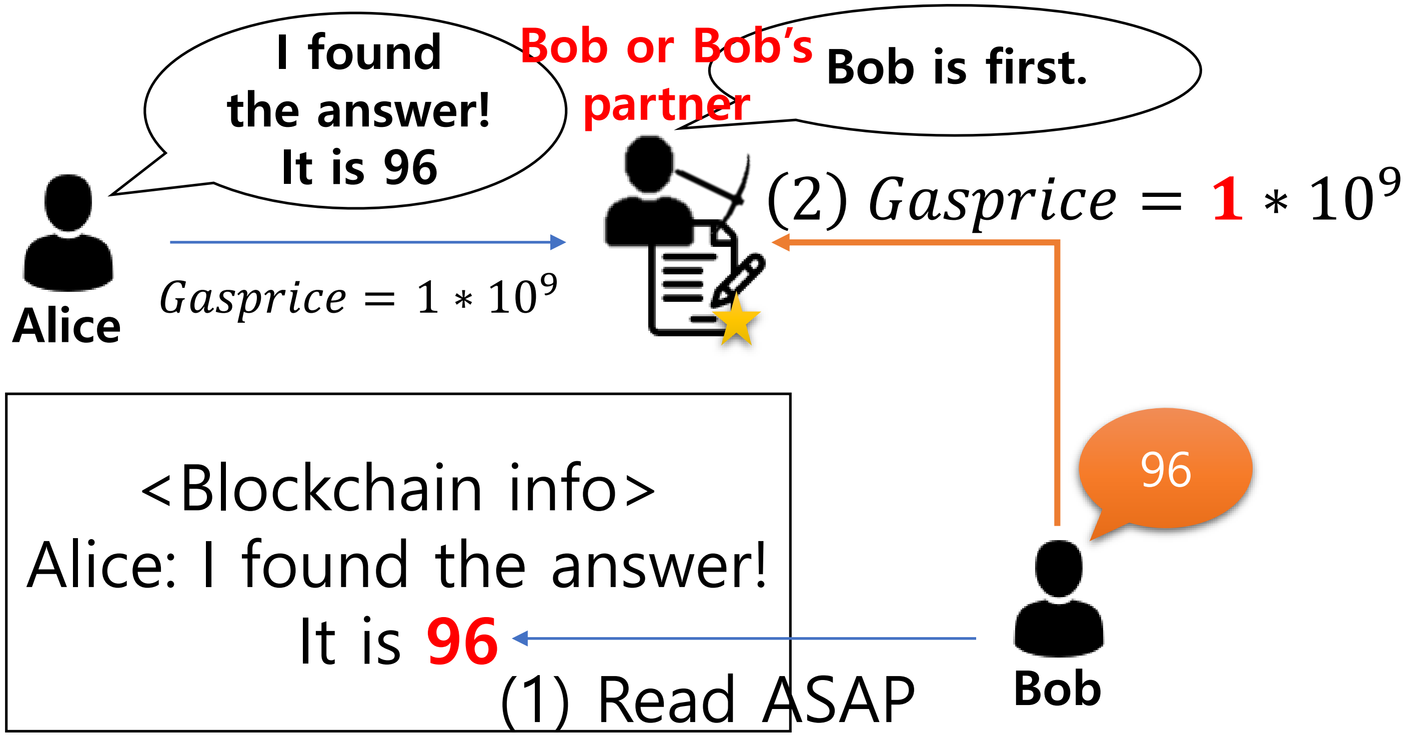

```
uint public reward;
bytes32 public diff;
bytes public solution;

function Puzzle() //constructor{
  owner = msg.sender;
  reward = msg.value;
  locked = false;
  diff = bytes32(11111); //pre-defined difficulty
}

function(){ //main code, runs at every invocation
  if (msg.sender == owner){ //update reward
    if (locked)
      throw;
    owner.send(reward);
    reward = msg.value;
  }
  else
    if (msg.data.length > 0){ //submit a solution
      if (locked) throw;
      if (sha256(msg.data) < diff){
        msg.sender.send(reward); //send reward
        solution = msg.data;
        locked = true;
      }}}}
```

Figure 3: A contract that rewards users who solve a computational puzzle.

# Security bugs in Ethereum


```
if (sha256(msg.data) < diff){
  msg.sender.send(reward); //send reward
```


**Attack #1**. TOD - Example




```
uint public reward;
bytes32 public diff;
bytes public solution;

function Puzzle() //constructor{
  owner = msg.sender;
  reward = msg.value;
  locked = false;
  diff = bytes32(11111); //pre-defined difficulty
}

function(){ //main code, runs at every invocation
  if (msg.sender == owner){ //update reward
    if (locked)
      throw;
    owner.send(reward);
    reward = msg.value;
  }
  else
    if (msg.data.length > 0){ //submit a solution
      if (locked) throw;
      if (sha256(msg.data) < diff){
        msg.sender.send(reward); //send reward
        solution = msg.data;
        locked = true;
      }}}}
```

Figure 3: A contract that rewards users who solve a computational puzzle.

# Security bugs in Ethereum

**Attack #2**. Timestamp Dependence

- The **timestamp** of the block is used to create a **random value.**

```
contract theRun {
  uint private Last_Payout = 0;
  uint256 salt = block.timestamp;
  function random returns (uint256 result){
    uint256 y = salt * block.number/(salt%5);
    uint256 seed = block.number/3 + (salt%300)
                   + Last_Payout +y;
    //h = the blockhash of the seed-th last block
    uint256 h = uint256(block.blockhash(seed));
    //random number between 1 and 100
    return uint256(h % 100) + 1;
  }}
```

Figure 5: A real contract which depends on block timestamp to send out money [22]. This code is simplified from the original code to save space.

# Security bugs in Ethereum

**Attack #2**. Timestamp Dependence

- The **timestamp** of the block is used to create a **random value.**
- local time manipulation with pre-computed value **(Randomness)**

**Bob or Bob's partner**

```
contract theRun {
  uint private Last_Payout = 0;
  uint256 salt =  block.timestamp;
  function random returns (uint256 result){
    uint256 y = salt * block.number/(salt%5);
    uint256 seed = block.number/3 + (salt%300)
                     + Last_Payout +y;
    //h = the blockhash of the seed-th last block
    uint256 h = uint256(block.blockhash(seed));
    //random number between 1 and 100
    return uint256(h % 100) + 1;
}}
```

Figure 5: A real contract which depends on block timestamp to send out money [22]. This code is simplified from the original code to save space.

block.timestamp <= now + 900 &&
block.timestamp >= parent.timestamp

# Security bugs in Ethereum

**Attack #2**. Timestamp Dependence

- The **timestamp** of the block is used to create a **random value.**
- local time manipulation with pre-computed value **(Randomness)**

**Bob or Bob's partner**

~~block.timestamp <= now + 900 &&~~
~~block.timestamp >= parent.timestamp~~

There is no time limit.

$H_s$ is the timestamp (in Unix's time()) of block $H$ and must fulfil the relation:

$$H_s > P(H)_{H_s} \quad (48)$$

Allow only 15 seconds. (geth code: consensys.go)

```
var (
        FrontierBlockReward    *big.Int = big.NewInt(5e+18)
        ByzantiumBlockReward   *big.Int = big.NewInt(3e+18)
        maxUncles                       = 2
        allowedFutureBlockTime          = 15 * time.Second
)
```

ref. from outdated whitepaper ☹
cuz of 3 years ago paper ☺

*Info ref. Wood, Gavin. "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER BYZANTIUM VERSION." Internet: https://github. com/ethereum/yellowpaper,[Apr. 17, 2019] (2019).
*geth is the the command line interface for running a full ethereum node implemented in Go (https://github.com/ethereum/go-Ethereum)

# Security bugs in Ethereum

**Attack #2**. Timestamp Dependence

- The **timestamp** of the block is used to create a **random value.**
- local time manipulation with pre-computed value **(Randomness)**

**Bob or Bob's partner**

~~block.timestamp <= now + 900 &&~~
~~block.timestamp >= parent.timestamp~~

There is no time limit.

$H_s$ is the timestamp (in Unix's time()) of block $H$ and must fulfil the relation:

$$H_s > P(H)_{H_s} \quad (48)$$

Allow only 15 seconds. (parity code: verification.rs)

```
const ACCEPTABLE_DRIFT: Duration = Duration::from_secs(15);
let max_time = SystemTime::now() + ACCEPTABLE_DRIFT;
let timestamp = UNIX_EPOCH + Duration::from_secs(header.timestamp());
if timestamp > max_time {
    return Err(From::from(BlockError::TemporarilyInvalid(OutOfBounc
}
```

ref. from outdated whitepaper ☹
cuz of 3 years ago paper ☺

*Info ref. Wood, Gavin. "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER BYZANTIUM VERSION." Internet: https://github. com/ethereum/yellowpaper,[Apr. 17, 2019] (2019).
*geth is the the command line interface for running a full ethereum node implemented in Go (https://github.com/ethereum/go-Ethereum)

# Security bugs in Ethereum

**Attack #3**. Mishandled Exception

```
function claimThrone(string name) {
  /.../
  if (currentMonarch.ethAddr != wizardAddress)
    currentMonarch.ethAddr.send(compensation);
  /.../
  // assign the new king
  currentMonarch = Monarch(
      msg.sender, name,
      valuePaid, block.timestamp);
```

# Security bugs in Ethereum

**Attack #3**. Mishandled Exception

- send reward -> assign the new king

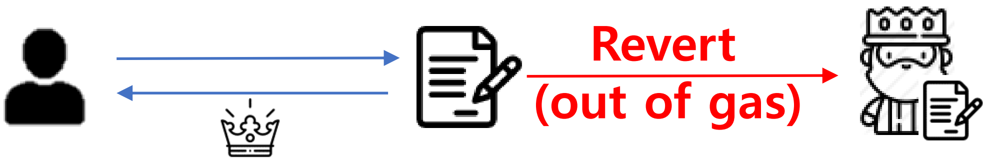

```
contract KingOfTheEtherThrone {
  struct Monarch {
    // address of the king.
    address ethAddr;
    string name;
    // how much he pays to previous king
    uint claimPrice;
    uint coronationTimestamp;
    }
  Monarch public currentMonarch;
  // claim the throne
  function claimThrone(string name) {
    /.../
    if (currentMonarch.ethAddr != wizardAddress)
      currentMonarch.ethAddr.send(compensation);
    /.../
    // assign the new king
    currentMonarch = Monarch(
        msg.sender, name,
        valuePaid, block.timestamp);
  }}
```

Figure 6: A code snippet of a real contract which does not check the return value after calling other contracts [12].

# Security bugs in Ethereum

**Attack #3**. Mishandled Exception

- send reward -> assign the new king




- 27.9% of the contract do not check the return values after calling other contracts via send.

```
contract KingOfTheEtherThrone {
  struct Monarch {
    // address of the king.
    address ethAddr;
    string name;
    // how much he pays to previous king
    uint claimPrice;
    uint coronationTimestamp;
    }
  Monarch public currentMonarch;
  // claim the throne
  function claimThrone(string name) {
    /.../
    if (currentMonarch.ethAddr != wizardAddress)
      currentMonarch.ethAddr.send(compensation);
    /.../
    // assign the new king
    currentMonarch = Monarch(
        msg.sender, name,
        valuePaid, block.timestamp);
  }}
```

Figure 6: A code snippet of a real contract which does not check the return value after calling other contracts [12].

# Security bugs in Ethereum

**Attack #4**. Reentrancy Vulnerability

- In Ethereum, when a contract calls another, the current execution waits for the call to finish.

```
contract Vulnerable {

    mapping (address => uint) public _balanceOf;

    function withdrawEquity() public returns (bool) {
        uint x = _balanceOf[msg.sender];
        msg.sender.call.value(x)();
        _balanceOf[msg.sender] = 0;
        return true;
    }

    //other functions

}
```

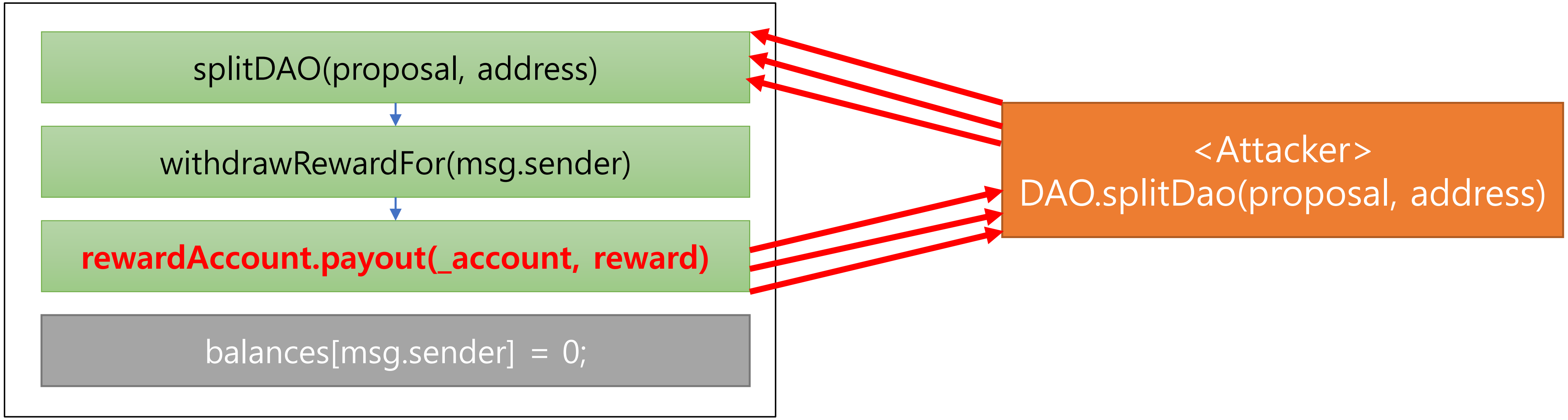
```
contract Malicious {

    address private _owner;
    address private _vulnerableAddr = 0x0;
    Vulnerable public vul = Vulnerable(_vulnerableAddr);

    function Malicious() public {
        _owner = msg.sender;
    }

    function () public payable {
        vul.withdrawEquity();
    }

    function winnerWinnerChickenDinner() public {
        _owner.transfer(this.balance);
    }

```

*code from https://hackernoon.com/smart-contract-security-part-1-reentrancy-attacks-ddb3b2429302

# Security bugs in Ethereum

**Attack #4**. Reentrancy Vulnerability

- The DAO Hack
- Most well-known smart contract vulnerability.
- The hacker stole over 3,600,000 ETH / 60,000,000 USD




*code from (TheDAO) https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413#code

# Towards a better design

# Towards a better design

- Operational Semantics of Ethereum

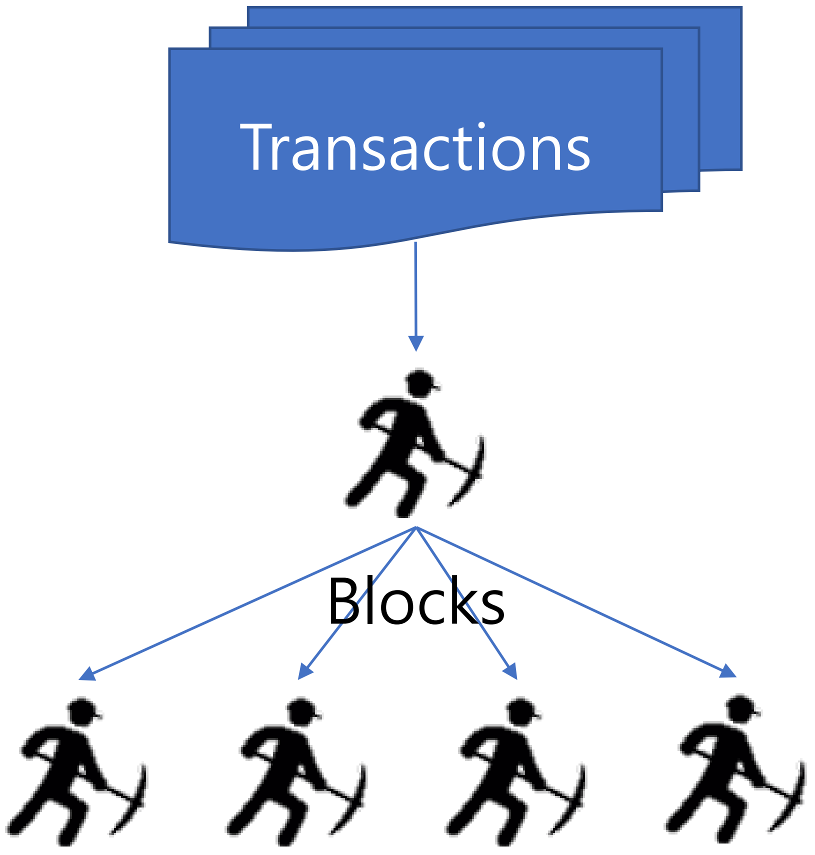
$$\textsc{Propose}\;\frac{\begin{array}{c}\texttt{TXs} \leftarrow \text{ Some transaction sequence } (T_1 \ldots T_n) \text{ from } \Gamma \\ B \leftarrow \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{proof-of-work}(B, BC) \\ \forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i\end{array}}{\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle}$$

$$\text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B$$

$$\textsc{Accept}\;\frac{\begin{array}{c}\text{Receive } B \equiv \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{TXs} \equiv (T_1 \ldots T_n) \\ \forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i\end{array}}{\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle}$$

$$\text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B$$

Figure 8: Proposing and Accepting a Block



# Towards a better design

- Transaction Execution

$$\text{TX-SUCCESS}\ \frac{\begin{array}{c} T \equiv \langle id, v, l \rangle \qquad M \leftarrow \text{Lookup}(\sigma, id) \\ \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle \langle M, 0, l, \epsilon \rangle \cdot \epsilon, \sigma' \rangle \rightsquigarrow^* \langle \epsilon, \sigma'' \rangle \end{array}}{\sigma \xrightarrow{T} \sigma''}$$

$$\text{TX-EXCEPTION}\ \frac{\begin{array}{c} T \equiv \langle id, v, l \rangle \qquad M \leftarrow \text{Lookup}(\sigma, id) \\ \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle \langle M, 0, l, \epsilon \rangle \cdot \epsilon, \sigma' \rangle \rightsquigarrow^* \langle \langle e \rangle_{exc} \cdot \epsilon, \bullet \rangle \end{array}}{\sigma \xrightarrow{T} \sigma}$$

# Towards a better design

- Recommendations for Better Semantics - Overview
  - Guard transactions
    - $g$ : guard condition
    - TX-Stale: current state $\sigma$ needs to satisfy $g$ for the execution of $T$

$$\text{TX-STALE}\ \frac{T \equiv \langle g, \bullet, \bullet, \bullet \rangle \qquad \sigma \not\vdash g}{\sigma \xrightarrow{T} \sigma}$$

$$\text{TX-SUCCESS}\ \frac{\begin{array}{c} T \equiv \langle g, id, v, l \rangle \qquad M \leftarrow \mathsf{Lookup}(\sigma, id) \\ \sigma \vdash g \qquad \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle \langle M, 0, l, \epsilon \rangle \cdot \epsilon, \sigma' \rangle \leadsto^{*} \langle \epsilon, \sigma'' \rangle \end{array}}{\sigma \xrightarrow{T} \sigma''}$$

$$\text{TX-EXCEPTION}\ \frac{\begin{array}{c} T \equiv \langle g, id, v, l \rangle \qquad M \leftarrow \mathsf{Lookup}(\sigma, id) \\ \sigma \vdash g \qquad \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle \langle M, 0, l, \epsilon \rangle \cdot \epsilon, \sigma' \rangle \leadsto^{*} \langle \langle e \rangle_{exc} \cdot \epsilon, \bullet \rangle \end{array}}{\sigma \xrightarrow{T} \sigma}$$

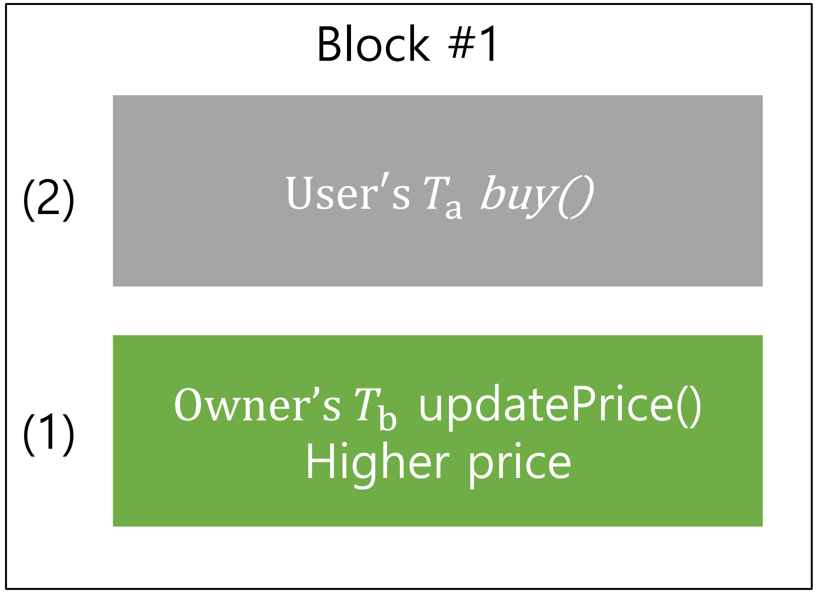
Figure 10: New Rules for Transaction Execution.

# Towards a better design

- Recommendations for Better Semantics - TOD
  - Guard transactions
    - $g$ : guard condition
    - TX-Stale: current state $\sigma$ needs to satisfy $g$ for the execution of $T$

```
contract MarketPlace{
  uint public price;
  uint public stock;
  /.../
  function updatePrice(uint _price){
    if (msg.sender == owner)
      price = _price;
  }
  function buy (uint quant) returns (uint){
    if (msg.value < quant * price || quant > stock)
      throw;
    stock -= quant;
    /.../
  }}
```

Figure 4: A contract that acts as a market place where users can buy/ sell some tokens. Due to TOD, some order may or may not go through.



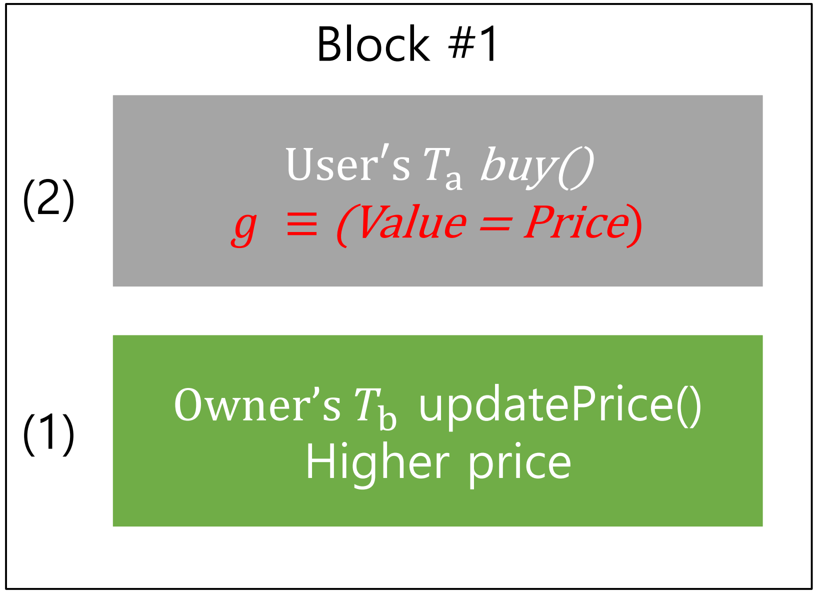

Vulnerable!

# Towards a better design

- Recommendations for Better Semantics - TOD
  - Guard transactions
    - $g$ : guard condition
    - TX-Stale: current state $\sigma$ needs to satisfy $g$ for the execution of $T$

```
contract MarketPlace{
  uint public price;
  uint public stock;
  /.../
  function updatePrice(uint _price){
    if (msg.sender == owner)
      price = _price;
  }
  function buy (uint quant) returns (uint){
    if (msg.value < quant * price || quant > stock)
      throw;
    stock -= quant;
    /.../
  }}
```

Figure 4: A contract that acts as a market place where users can buy/ sell some tokens. Due to TOD, some order may or may not go through.



# Towards a better design

- Recommendations for Better Semantics – Timestamp Dependence
  - Deterministic Timestamp
    - block timestamp is *essentially a redundant feature*
    - a new block is created approximately every 12 seconds in Ethereum
    - block.timestamp (X)
    - block number (O)

```
contract theRun {
  uint private Last_Payout = 0;
  uint256 salt =  block.timestamp;
  function random returns (uint256 result){
    uint256 y = salt * block.number/(salt%5);
    uint256 seed = block.number/3 + (salt%300)
                   + Last_Payout +y;
    //h = the blockhash of the seed-th last block
    uint256 h = uint256(block.blockhash(seed));
    //random number between 1 and 100
    return uint256(h % 100) + 1;
  }}
```

Figure 5: A real contract which depends on block timestamp to send out money [22]. This code is simplified from the original code to save space.

# Towards a better design

- Recommendations for Better Semantics – Mishandled exception
  - Better exception handling
  - “Make & Use Try-catch”

  - Info: catching exceptions is not yet possible in Solidity.

JavaScript Demo: Statement - Try...Catch

```
try {
  nonExistentFunction();
}
catch(error) {
  console.error(error);
  // expected output: ReferenceError: nonExistentFunction is not defined
  // Note - error messages will vary depending on browser
}

```

Run ›

```
> ReferenceError: nonExistentFunction is not defined
```

*Code from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch
*info from https://solidity.readthedocs.io/en/v0.5.7/control-structures.html#error-handling-assert-require-revert-and-exceptions

# Towards a better design

- Recommendations for Better Semantics – Mishandled exception
  - Error Handling.

## Error Handling

See the dedicated section on assert and require for more details on error handling and when to use which function.

`assert(bool condition)`:

causes an invalid opcode and thus state change reversion if the condition is not met - to be used for internal errors.

`require(bool condition)`:

reverts if the condition is not met - to be used for errors in inputs or external components.

`require(bool condition, string memory message)`:

reverts if the condition is not met - to be used for errors in inputs or external components. Also provides an error message.

`revert()`:

abort execution and revert state changes

`revert(string memory reason)`:

abort execution and revert state changes, providing an explanatory string

*info from https://solidity.readthedocs.io/en/v0.5.7/units-and-global-variables.html#error-handling

# Towards a better design

- Recommendations for Better Semantics
  - Reentrancy Vulnerability (Not covered in this paper.)
  - Call after update.

```
contract Vulnerable {

    mapping (address => uint) public _balanceOf;

    function withdrawEquity() public returns (bool) {
        _balanceOf[msg.sender] = 0;
        uint x = _balanceOf[msg.sender];
        msg.sender.call.value(x)();
        return true;
    }

    //other functions

}
```

```
contract Malicious {

    address private _owner;
    address private _vulnerableAddr = 0x0;
    Vulnerable public vul = Vulnerable(_vulnerableAddr);

    function Malicious() public {
        _owner = msg.sender;
    }

    function () public payable {
        vul.withdrawEquity();
    }

    function winnerWinnerChickenDinner() public {
        _owner.transfer(this.balance);
    }

```

*code from https://hackernoon.com/smart-contract-security-part-1-reentrancy-attacks-ddb3b2429302

# The $Oyente$ Tool

## Compare with teEther

Krupp, Johannes, and Christian Rossow. "teether: Gnawing at ethereum to automatically exploit smart contracts." *27th USENIX Security Symposium*. 2018.

# The *Oyente* Tool

- How to solve the problem in smart contract?
  - *Oyente* : An analysis tool for smart contract based upon symbolic execution
  - developers to write better contracts
  - users to avoid invoking problematic contracts

  - y = λ, x = χ
  - $(z \neq 1000)$ : ((χ * 2) ≠ 1000)
  - $(z = 1000,\ y \leq z)$ : (((χ * 2) = 1000) && λ ≤ (χ * 2))
  - $(z = 1000,\ y > z)$ : (((χ * 2) = 1000) && λ > (χ * 2))
  - Symbolically executing all feasible program paths does not scale to large programs. (But, targets of *Oyente* are smart contracts)

SymbolicExe.c

```c
#include <stdio.h>

void main(){
    int x,y,z;

    scanf("%d",&x);
    scanf("%d",&y);

    z = x * 2;

    if(z == 1000){
        if(y > z){
            printf("Nice!\n");
        }else{
            printf("Wrong!\n");
        }
    }
}
```

*Code from https://www.lazenca.net/pages/viewpage.action?pageId=6324534

# The *Oyente* Tool

- Z3
  - An efficient SMT Solver

```
(ee817) reset@DESKTOP-IP14NG0:~/test_oyente$ python
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for
>>> from z3 import *
>>> x = Int('x')
>>> y = Int('y')
>>> solve(x>5, y<1337, y > 1, x*y==20190508)
[y = 767, x = 26324]
```

**Z3: An Efficient SMT Solver**

Leonardo de Moura and Nikolaj Bjørner

Microsoft Research, One Microsoft Way, Redmond, WA, 98074, USA
{leonardo,nbjorner}@microsoft.com

**Abstract.** Satisfiability Modulo Theories (SMT) problem is a *decision problem* for logical first order formulas with respect to combinations of background theories such as: arithmetic, bit-vectors, arrays, and uninterpreted functions. Z3 is a new and efficient SMT Solver freely available from Microsoft Research. It is used in various software verification and analysis applications.

# The *Oyente* Tool

- *teEther*
  - The attacker of this paper is a weak attacker
  - The goal is to find a contract in which the attacker can call the money-related instruction
  - Ex. SELFDESTRUCT(address): sends all of the contract's current balance to address
  - Make Exploit automatically.

**TEETHER: Gnawing at Ethereum to Automatically Exploit Smart Contracts**

Johannes Krupp
*CISPA, Saarland University, Saarland Informatics Campus*

Christian Rossow
*CISPA, Saarland University, Saarland Informatics Campus*

**Abstract**

Cryptocurrencies like Bitcoin not only provide a decentralized currency, but also provide a programmatic way to process transactions. Ethereum, the second largest cryptocurrency next to Bitcoin, is the first to provide a Turing-complete language to specify transaction processing, thereby enabling so-called *smart contracts*. This provides an opportune setting for attackers, as security vulnerabilities are tightly intertwined with financial gain. In this paper, we consider the problem of automatic vulnerability identification and exploit generation for smart contracts. We develop a generic definition of vulnerable contracts and use this to build TEETHER, a tool that allows creating an exploit for a contract given only its binary bytecode. We perform a large-scale analysis of all 38,757 unique Ethereum contracts, 815 out of which our tool finds working exploits for—completely automated.

lion USD [1]. Although Bitcoin remains the predominant cryptocurrency, it also inspired many derivative systems. One of the most popular of these is Ethereum, the second largest cryptocurrency by overall market value as of mid 2018 [1].

Ethereum heavily extends the way consensus protocols handle transactions: While Bitcoin allows to specify simple checks that are to be performed when processing a transaction, Ethereum allows these rules to be specified in a Turing-complete language. This makes Ethereum the number one platform for so-called *smart contracts*.

A smart contract can be seen quite literally as a contract that has been formalized in code. As such, smart contracts can for example be used to implement fundraising schemes that automatically refund contributions unless a certain amount is raised in a given time, or shared wallets that require transactions to be approved of by multiple owners before execution. In Ethereum, smart

# The *Oyente* Tool

- *Oyente* Architecture
  - Overview

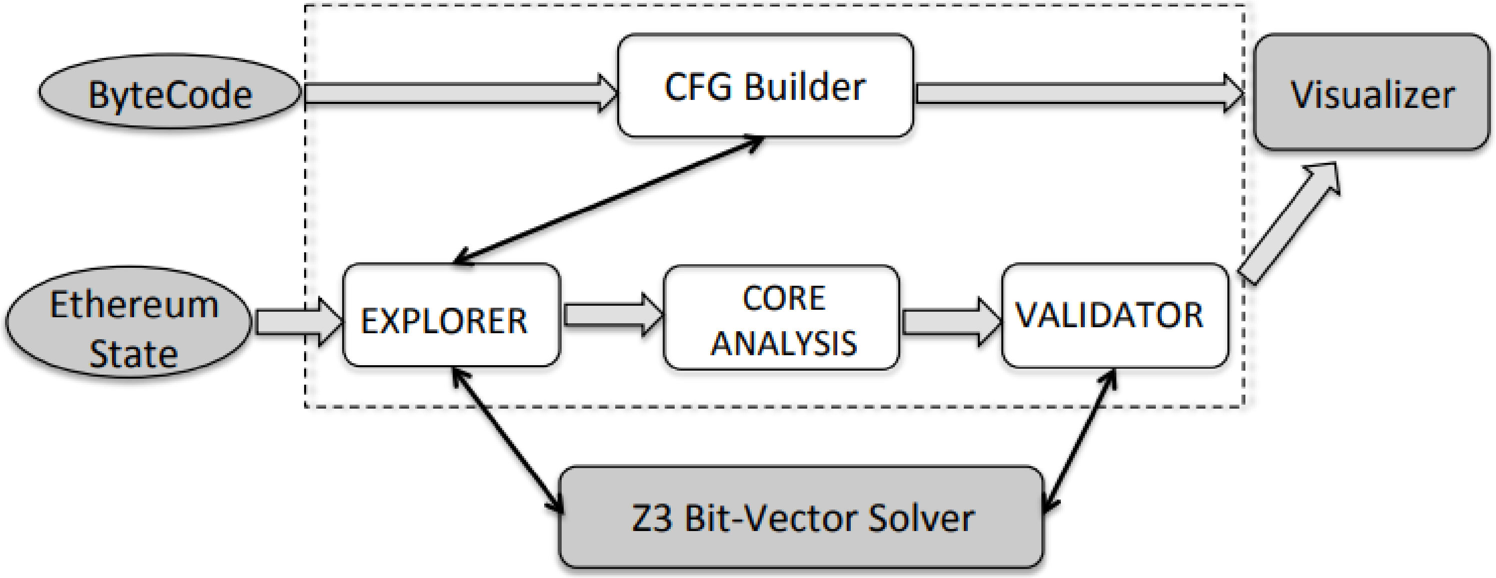



Figure 11: Overview Architecture of OYENTE. Main components are within the dotted area. Shaded boxes are publicly available.

# The *Oyente* Tool

- *teEther* Architecture
  - Overview

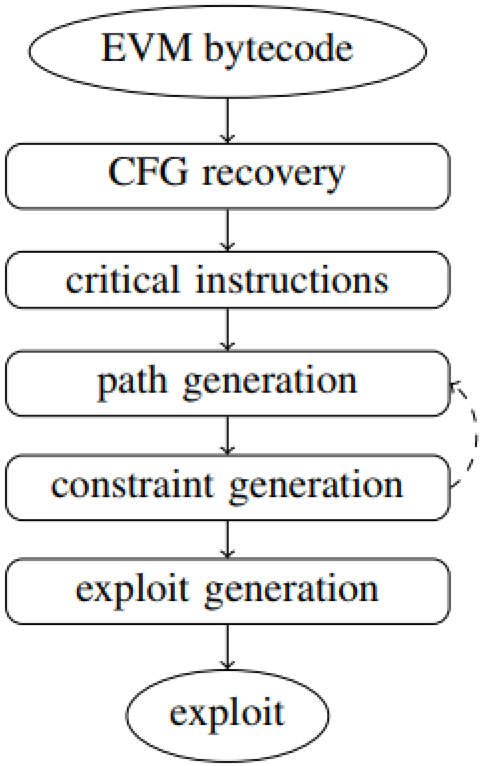


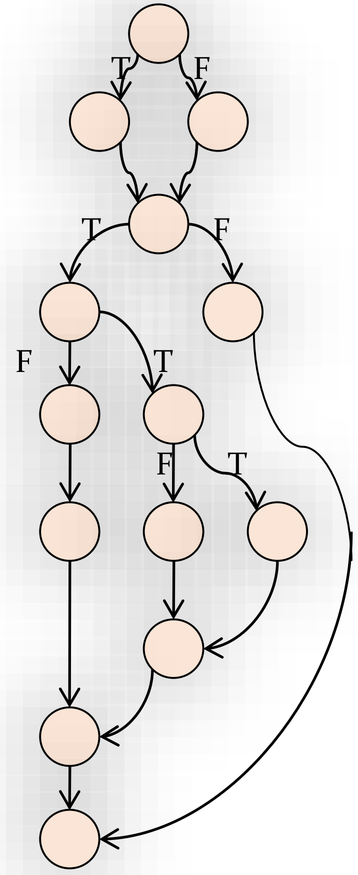

Figure 4: Architecture of TEETHER

*Krupp, Johannes, and Christian Rossow. "teether: Gnawing at ethereum to automatically exploit smart contracts." *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018.

# The $Oyente$ Tool

- $Oyente$ Architecture
    - CFG Recovery

6060604052123123123528.....




- 64 EVM Instructions.
- Block => Node
- Jump => Edge

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The *Oyente* Tool

- *teEther* Architecture
  - CFG Recovery

3460576060565b606060565b
50565b00151600…

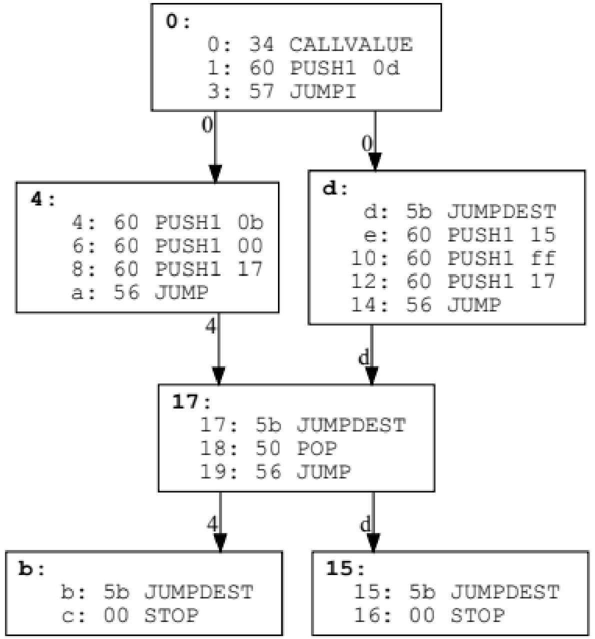



Figure 5: An example CFG with dependent edges

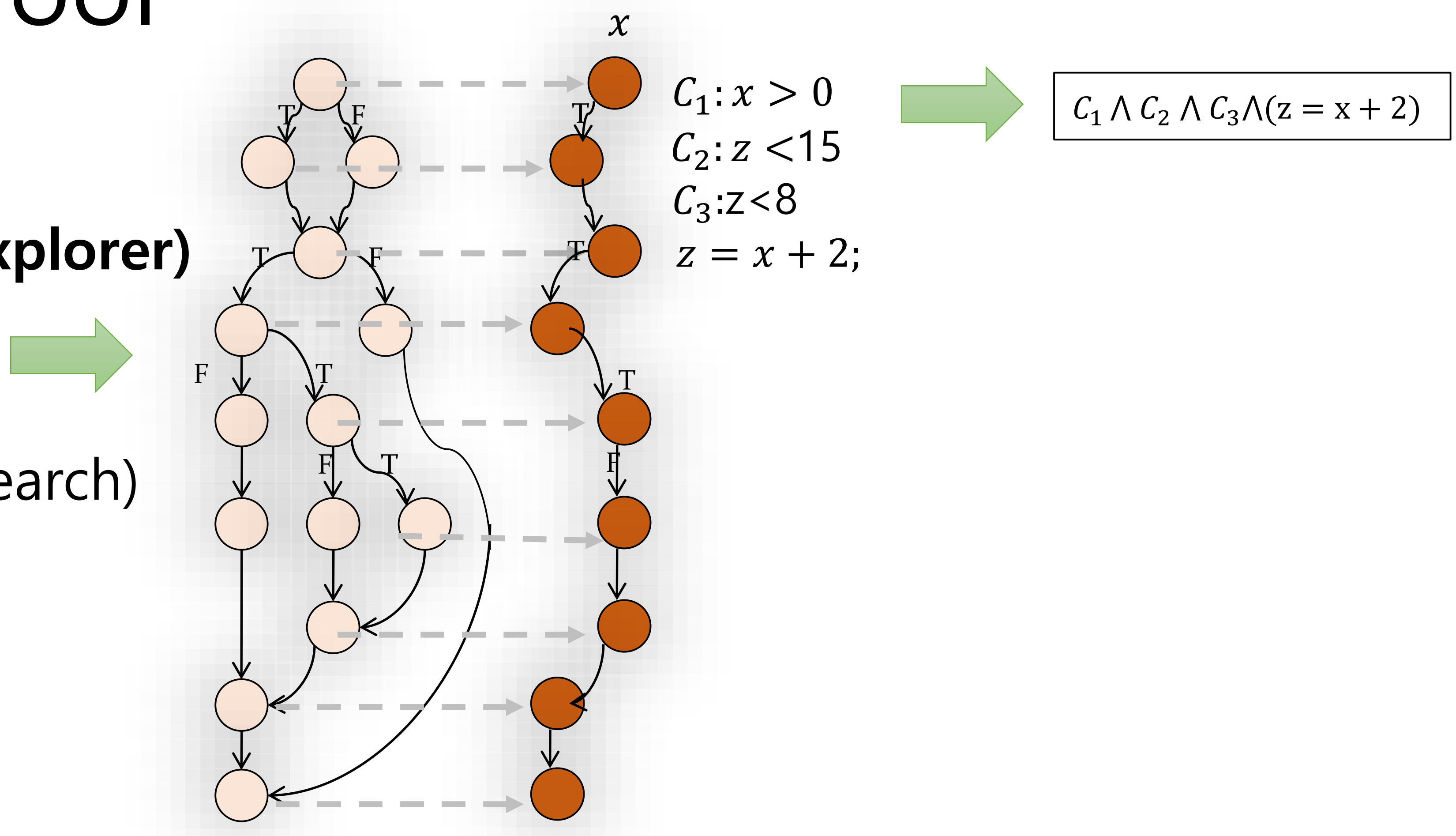
| | |
|---|---|
| JUMP | unconditional jump |
| JUMPI | conditional jump |
| JUMPDEST | Markers jump target |

*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The *Oyente* Tool

- *Oyente* Architecture
  - Execution Trace **(Explorer)**

6060604052123123123528.....



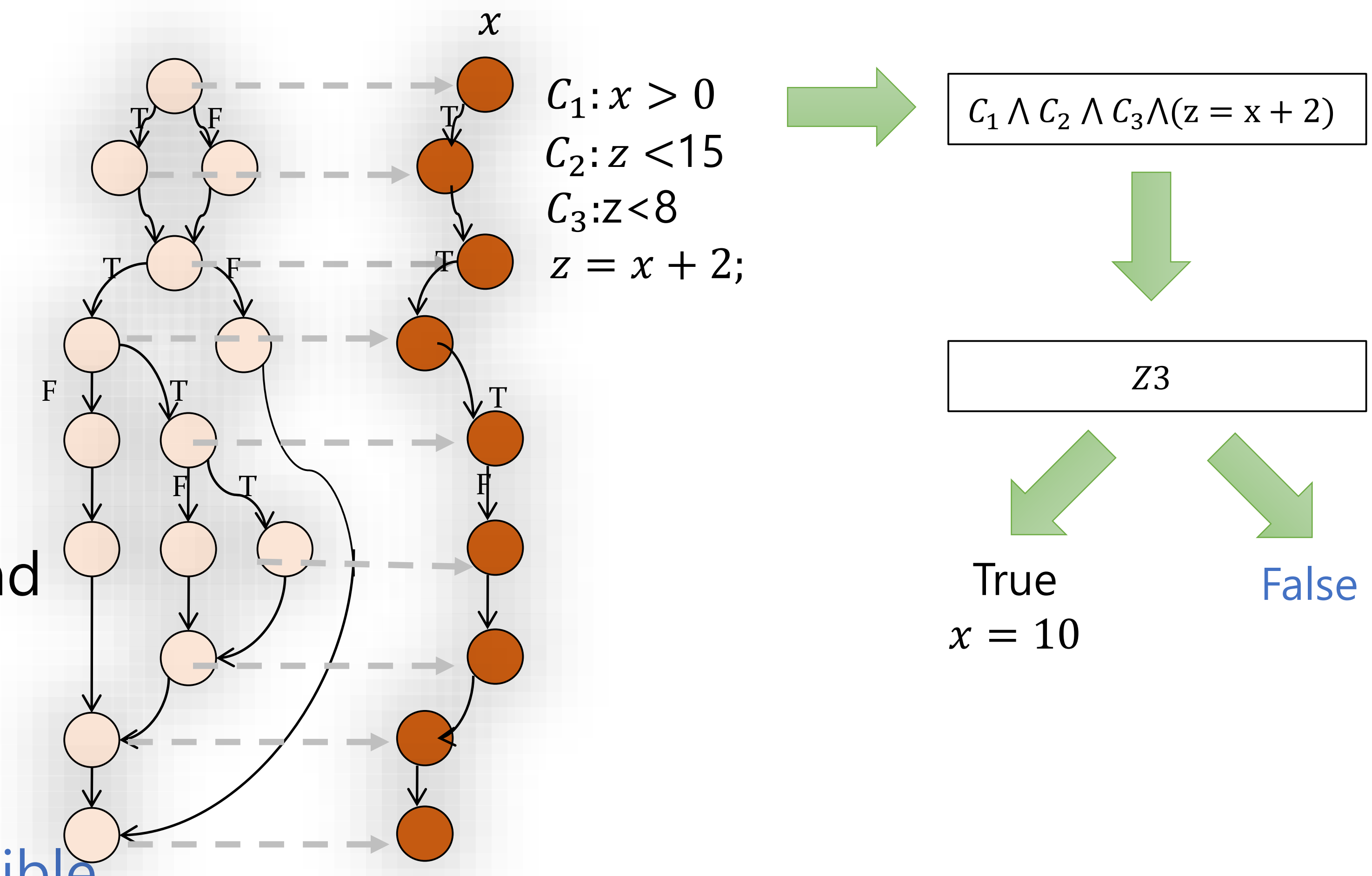

  - DFS (Depth Frist Search)

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The *Oyente* Tool

- *Oyente* Architecture
  - Theorem Prover

6060604052123123123528.....




  - Each trace is associated with a path constraint and auxiliary data that the analyses in later phase require.
  - Z3 in particular, helps us eliminate provably infeasible traces from consideration.

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The *Oyente* Tool

- *teEther* Architecture
  - Path generation

3460576060565b606060565b
50565b00151600…

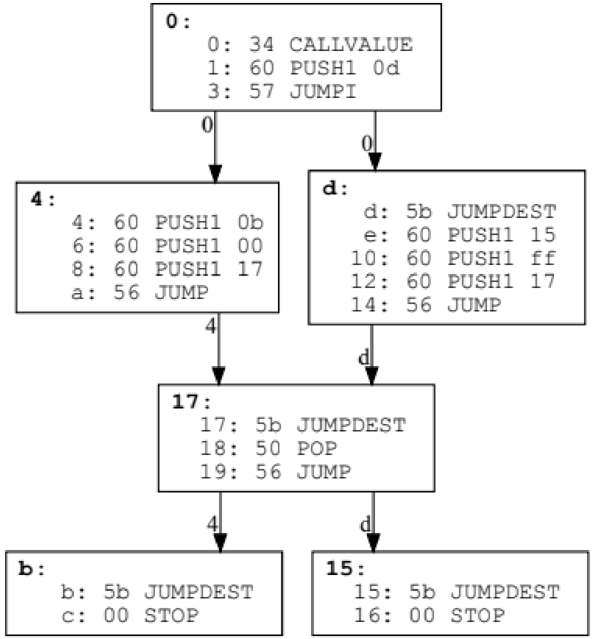
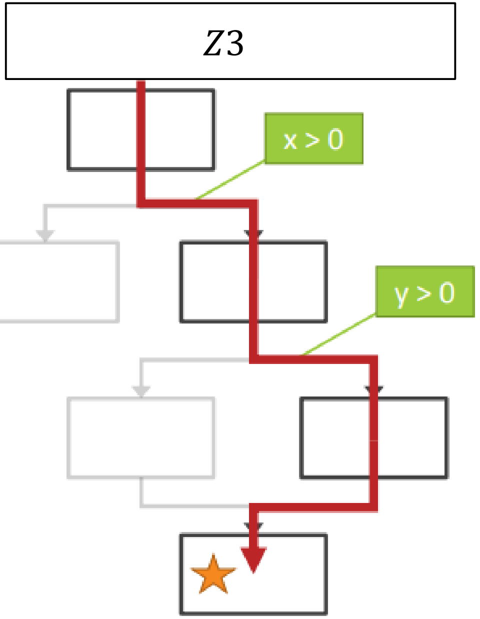



Figure 5: An example CFG with dependent edges




CRITICAL = ['CALL', 'DELEGATECALL', 'CALLCODE', 'SELFDESTRUCT']

  - Wait!
  - **There are some challenges.**

*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The *Oyente* Tool

- *teEther* Architecture
  - Path generation – Challenge #1. Contract state

```
contract Stateful{
    bool vulnerable = false;
    function exploit(address attacker){
        require(vulnerable);
        attacker.transfer(this.balance);
    }
    function makeVulnerable(){
        vulnerable = true;
    }
}
```

*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The *Oyente* Tool

- *teEther* Architecture
  - Path generation – Challenge #1. Contract state

    (1) mark SSTORE instructions
    (2) compute backward slices of argument(s)
    (3) generate path through a slice
    (4) execute path symbolically (collect path constraints)
      - **collect storage reads R & write W**
      - combine states changing paths + 1 critical path

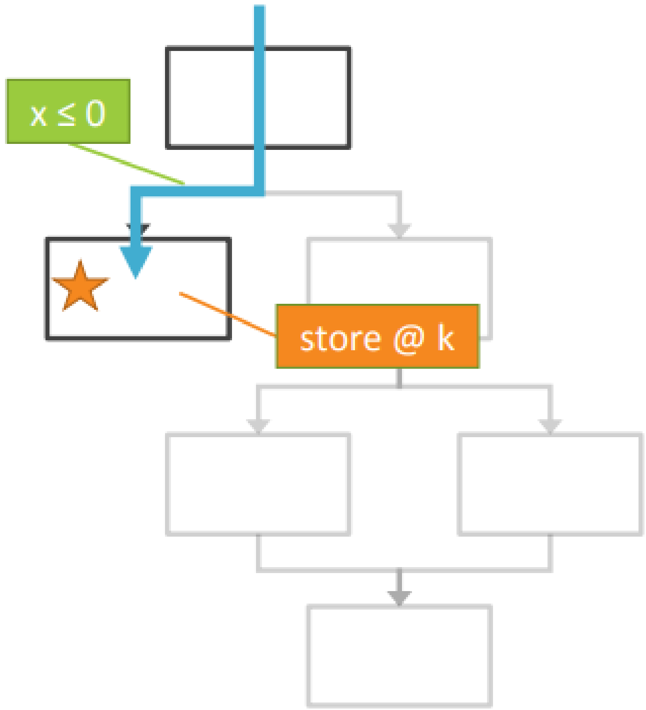



*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The *Oyente* Tool

- *teEther* Architecture
  - Path generation – Challenge #2. Hash Functions
  - EVM has **SHA hash** instructions.
  - Hash is a one-way function.

```
function check(bytes32 data, bytes32 check){
    require(data == "1337" && sha3(data) == check)
```

  - If the hash function is in the constraints, it is impossible to solve.

*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The *Oyente* Tool

- *teEther* Architecture
  - Path generation – Challenge #2. Hash Functions

(1) Remove dependent constraints
(2) Solve reduced set
(3) Compute hash values
(4) Replace dependent constraints
(5) Repeat.

dependent expression

$$C = \{data = "1337", \underbrace{sha3(data) = check}_{\text{dependent constraint}}\}$$

dependent constraint

$$C' = \{data = "1337"\}$$

$$sha3(data) \rightarrow 0x985d..$$

$$C' = \{data = "1337", \underbrace{0x985d.. = check}_{\text{independent}}\}$$

independent

*Image from "teether: Gnawing at ethereum to automatically exploit smart contracts (slide pptx)", Johannes.krupp@cispa, USENIX 2018

# The $Oyente$ Tool

- $Oyente$ Architecture
  - Core analysis – Transaction Ordering Dependence
  - [Remind]
    - **Explorer**: Returns a set of traces and the corresponding Ether flow for each trace.

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The *Oyente* Tool

- *Oyente* Architecture
  - if two different traces have different Ether flows => Vulnerable!

```
contract MarketPlace{
  uint public price;
  uint public stock;
  /.../
  function updatePrice(uint _price){
    if (msg.sender == owner)
      price = _price;
  }
  function buy (uint quant) returns (uint){
    if (msg.value < quant * price || quant > stock)
      throw;
    stock -= quant;
    /.../
  }}
```

1. Trace & Ether flow.

2. Trace & Ether flow.

Figure 4: A contract that acts as a market place where users can buy/ sell some tokens. Due to TOD, some order may or may not go through.

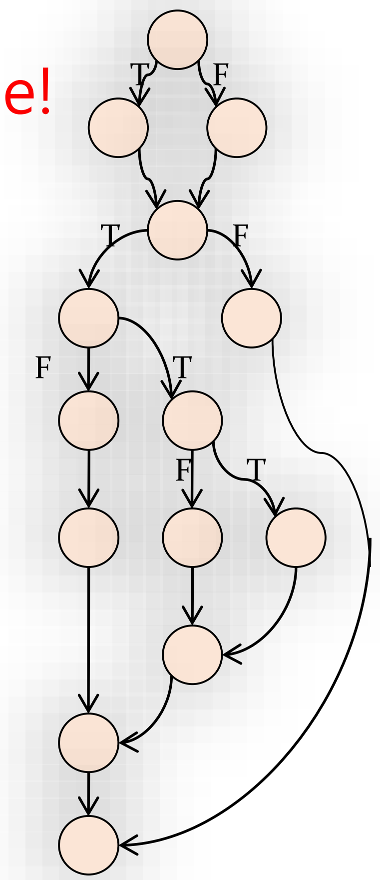



*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The $Oyente$ Tool

- $Oyente$ Architecture
  - Core analysis – Timestamp Dependency
  - Symbolize block.timestamp on Explorer. (Ex, $\theta$)
  - if this symbolic variable is included.
    A contract is flagged as timestamp-dependent vulnerability.

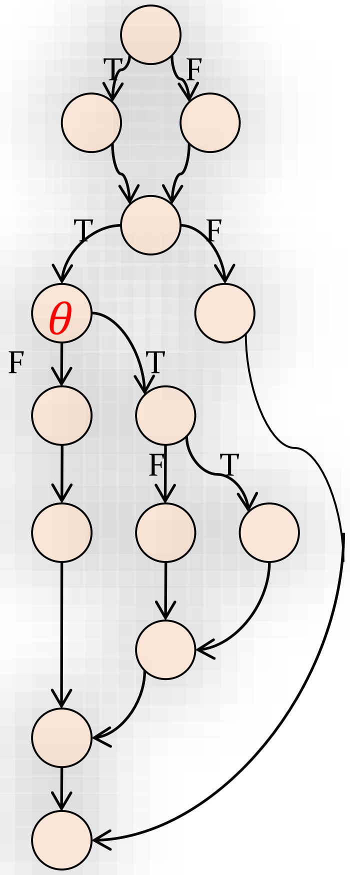
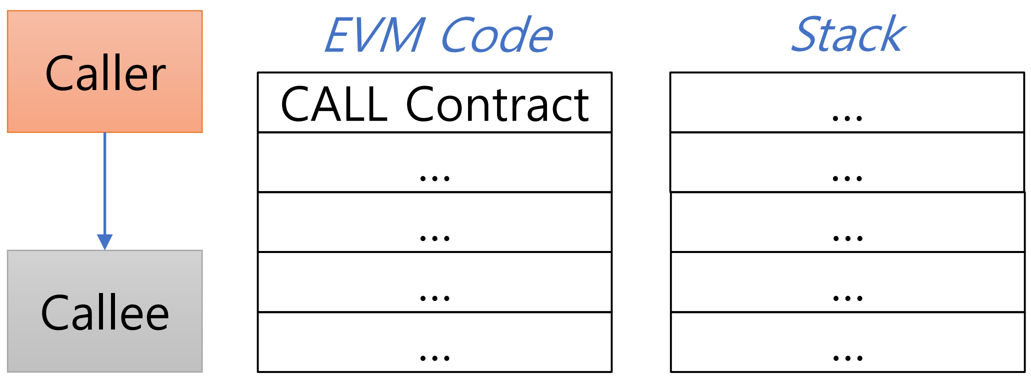



*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

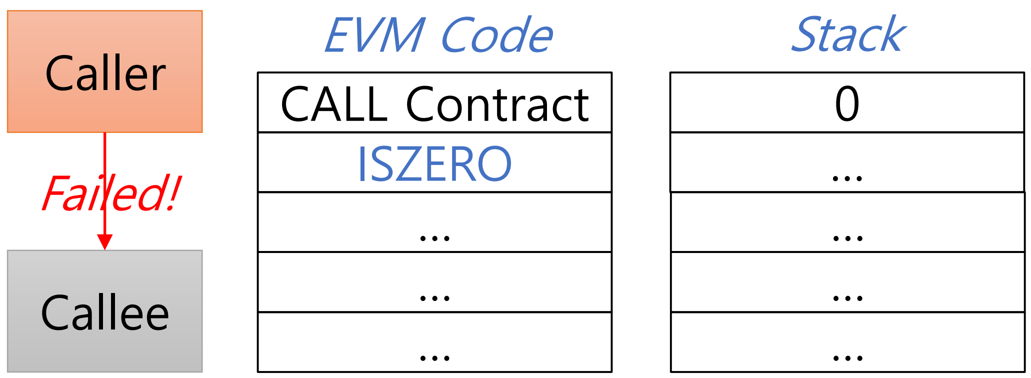
# The *Oyente* Tool

- *Oyente* Architecture
  - Core analysis – Mishandled Exception (send)



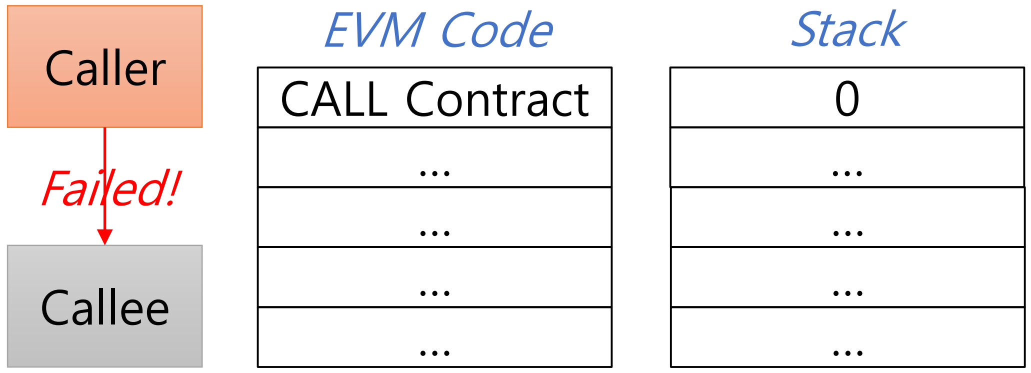
# The *Oyente* Tool

- *Oyente* Architecture
  - Core analysis – Mishandled Exception (send)
  - Safety



# The *Oyente* Tool

- *Oyente* Architecture
  - Core analysis – Mishandled Exception (send)
  - Vulnerable



# The $Oyente$ Tool

- $Oyente$ Architecture
  - Core analysis – Reentrancy Detection

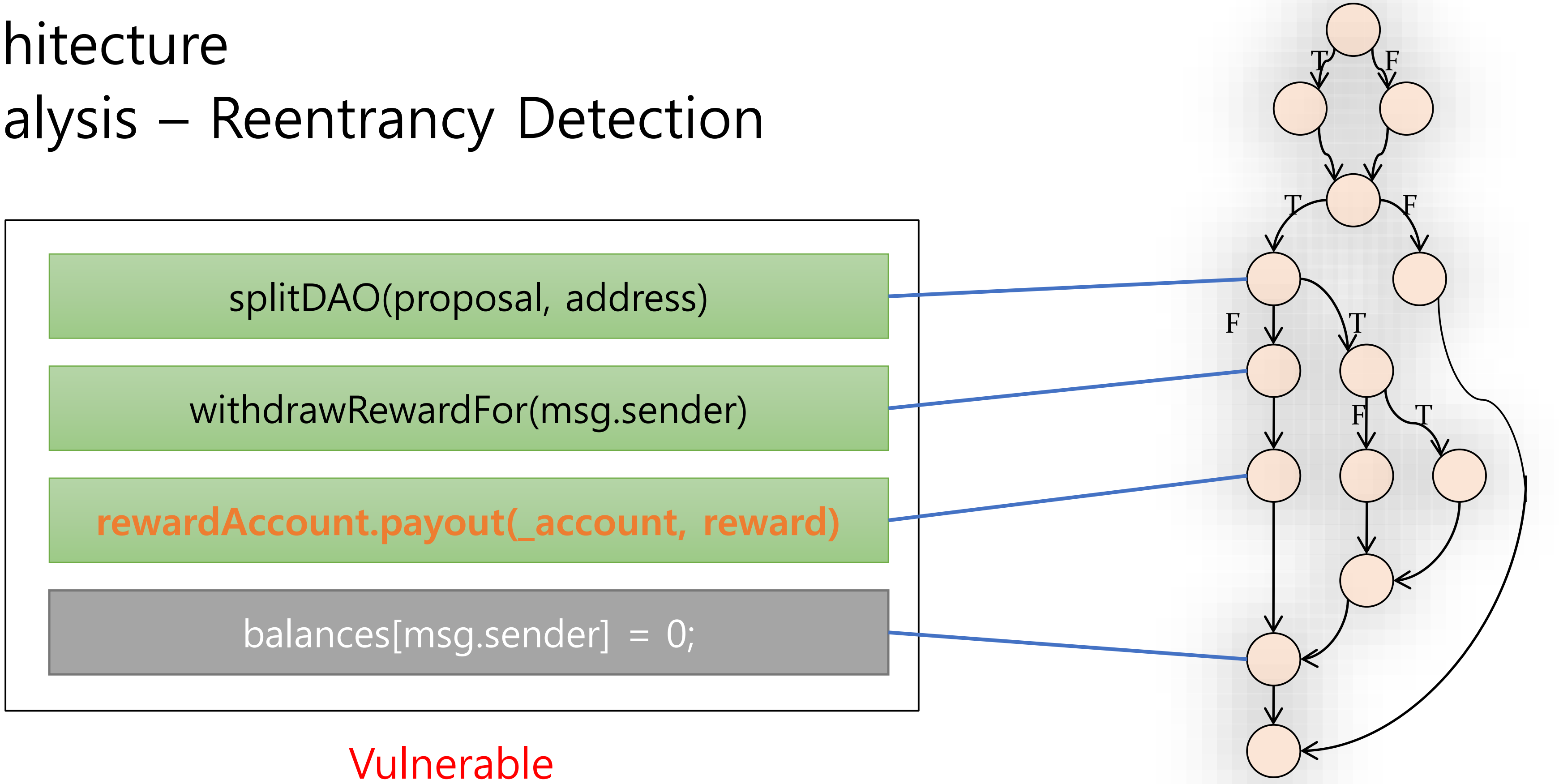
  - At each CALL that is encountered, they obtain the path condition for the execution before the CALL is executed.

  - check if such condition with updated variables (e.g., storage values) still holds (i.e., if the call can be executed again)

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# The *Oyente* Tool

- *Oyente* Architecture
    - Core analysis – Reentrancy Detection



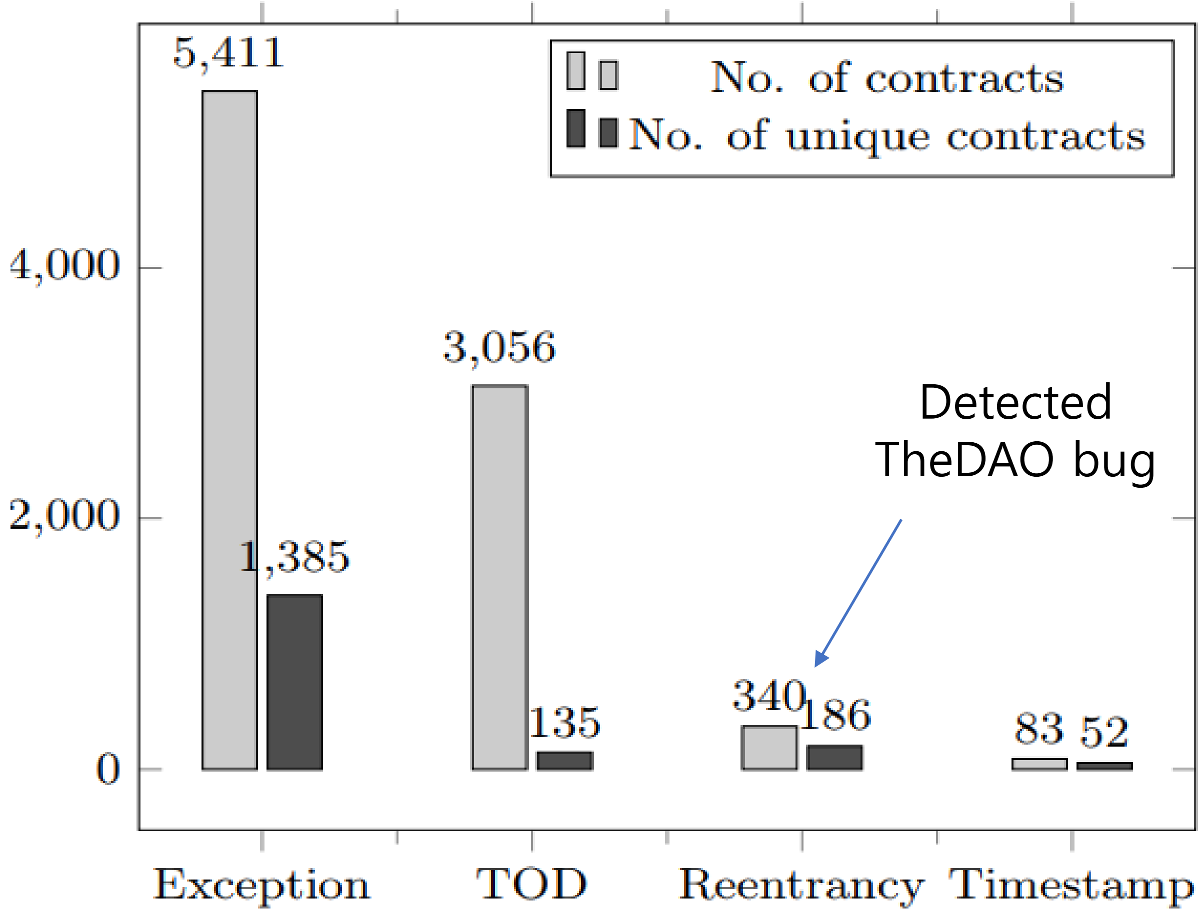

Vulnerable

*Image from "Making Smart Contracts Smarter: Oyente – Loi Luu (slide pptx)", Devcon 2016

# Conclusion

# Conclusion

- 19,336 Smart contracts (Mainnet)
- *Open-source!* (*Oyente*)




- *but for ethical reasons we do not conduct our attack confirmation on contracts*
- *False-Positive: Validator is far from being complete*

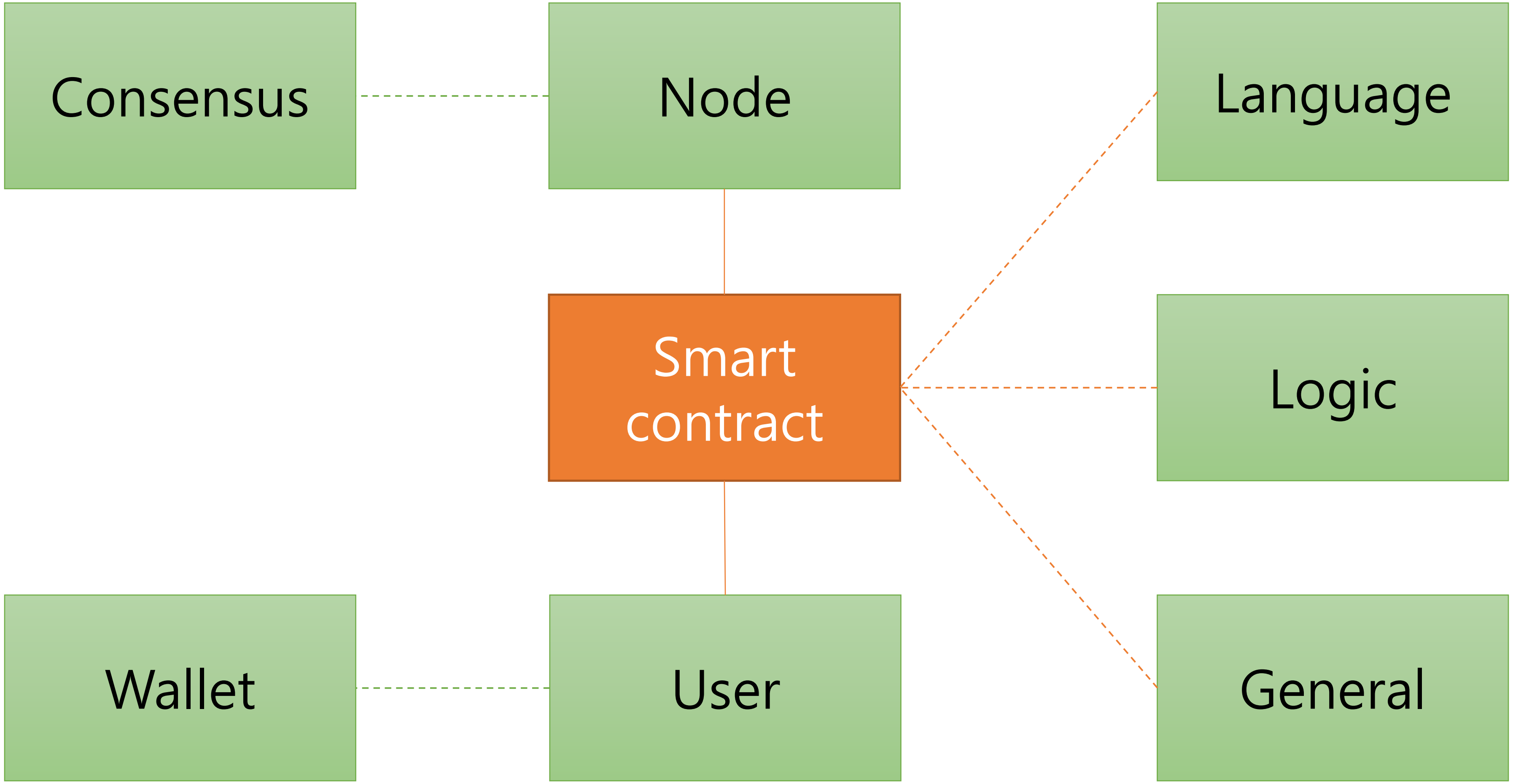
# Conclusion

- Contribution
  - Introducing several new classes of security bugs in the Ethereum Smart Contract
  - Formalize the “lightweight” semantics of Ethereum smart contract and propose recommendations as solutions for the documented bugs.
  - make & run $Oyente$, a symbolic execution tool which analyses Ethereum smart contracts to detect bugs, in real Ethereum network.

# Future Works

# Future Works

- Design defects due to component combination.



# END.

Thanks.

# Appendix

# Appendix - Towards a better design

- Operational Semantics of Ethereum - Denotation

  $\leftarrow$ assignment

  - an arbitrary element (The value that the program accesses during execution.)

  $\Downarrow$ big-step evaluation

  $\rightsquigarrow$ small-step evaluation

  $\sigma$ state (address and account state mapping)

  $\Gamma$ Transaction flow

  <BC, σ> Ethereum state as a pair <Blockchain, state>

  But, do not model miner rewards. (for simplicity)

# Appendix - Towards a better design

- Operational Semantics of Ethereum
  - ✓Only one "elected leader" executes the *Propose* rule at time.

$$
\textsc{Propose}\;
\frac{\begin{array}{c}
\texttt{TXs} \leftarrow \text{ Some transaction sequence } (T_1 \dots T_n) \text{ from } \Gamma \\
B \leftarrow \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\dots\rangle \\
\texttt{proof-of-work}(B, BC) \\
\forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i
\end{array}}{\langle BC, \sigma_0 \rangle \Downarrow \langle B \cdot BC, \sigma_n \rangle}
$$

$$\text{Remove } T_1 \dots T_n \text{ from } \Gamma \text{ and broadcast } B$$

$$
\textsc{Accept}\;
\frac{\begin{array}{c}
\text{Receive } B \equiv \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\dots\rangle \\
\texttt{TXs} \equiv (T_1 \dots T_n) \\
\forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i
\end{array}}{\langle BC, \sigma_0 \rangle \Downarrow \langle B \cdot BC, \sigma_n \rangle}
$$

$$\text{Remove } T_1 \dots T_n \text{ from } \Gamma \text{ and broadcast } B$$

Figure 8: Proposing and Accepting a Block

# Appendix - Towards a better design

- Operational Semantics of Ethereum
  - ✓Other miners use the *Accept* rule to "repeat" the transitions after the leader broadcasts block B ***(Timestamp-dependence)***

$$\textsc{Propose}\ \frac{\begin{array}{c}\texttt{TXs} \leftarrow \text{ Some transaction sequence } (T_1 \ldots T_n) \text{ from } \Gamma \\ B \leftarrow \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{proof-of-work}(B, BC) \\ \forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i\end{array}}{\begin{array}{c}\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle \\ \text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B\end{array}}$$

$$\textsc{Accept}\ \frac{\begin{array}{c}\text{Receive } B \equiv \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{TXs} \equiv (T_1 \ldots T_n) \\ \forall\, i, 1 \leq i \leq n : \sigma_{i-1} \xrightarrow{T_i} \sigma_i\end{array}}{\begin{array}{c}\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle \\ \text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B\end{array}}$$

Figure 8: Proposing and Accepting a Block

# Appendix - Towards a better design

- Operational Semantics of Ethereum
  - ✓some inevitable order among $T_i$ ***(Transaction-ordering dependence)***

$$
\textsc{Propose}\;\frac{\begin{array}{c}\texttt{TXs} \leftarrow \text{ Some transaction sequence } (T_1 \ldots T_n) \text{ from } \Gamma \\ B \leftarrow \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{proof-of-work}(B, BC) \\ \forall\, i, 1 \le i \le n : \boxed{\sigma_{i-1} \xrightarrow{T_i} \sigma_i}\end{array}}{\begin{array}{c}\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle \\ \text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B\end{array}}
$$

$$
\textsc{Accept}\;\frac{\begin{array}{c}\text{Receive } B \equiv \langle \texttt{blockid}\,;\texttt{timestamp}\,;\texttt{TXs}\,;\ldots\rangle \\ \texttt{TXs} \equiv (T_1 \ldots T_n) \\ \forall\, i, 1 \le i \le n : \boxed{\sigma_{i-1} \xrightarrow{T_i} \sigma_i}\end{array}}{\begin{array}{c}\langle BC, \sigma_0\rangle \Downarrow \langle B \cdot BC, \sigma_n\rangle \\ \text{Remove } T_1 \ldots T_n \text{ from } \Gamma \text{ and broadcast } B\end{array}}
$$

Figure 8: Proposing and Accepting a Block

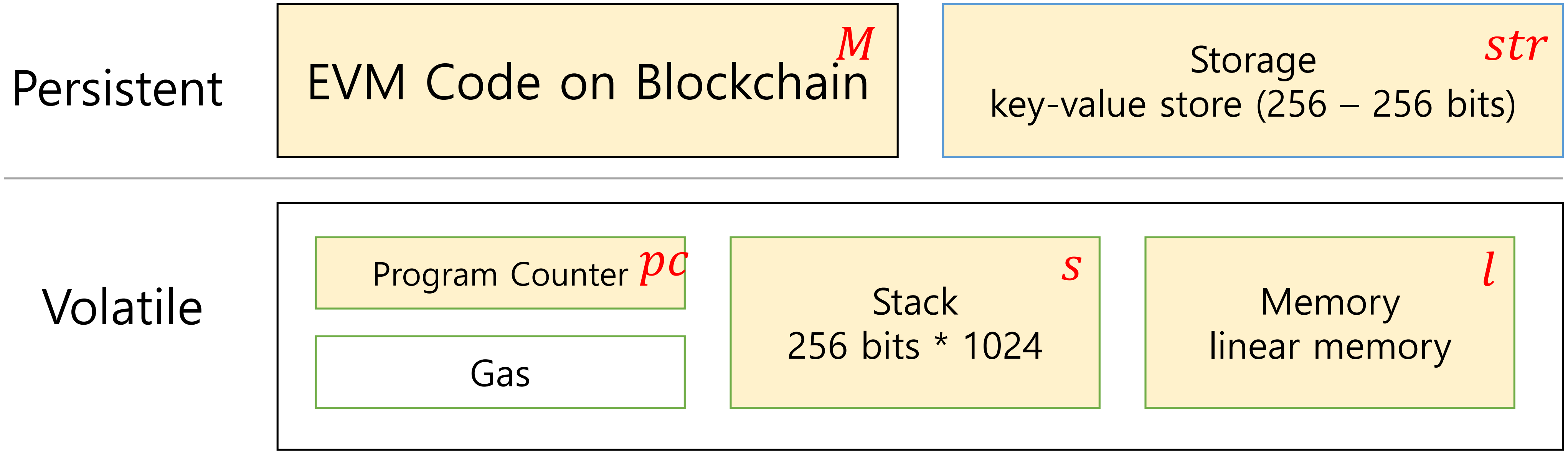
# Appendix - Towards a better design

- Transaction Execution – Denotation (Cont'd)
  - A transaction can activate the code execution of a contract.
  - execution can access to three types of space in which to store data
    - $s$ : LIFO Stack
    - $l$ : auxiliary memory (expandable array, input, output)
    - $str$ : long-term storage, part of σ[id]
    - $pc$ : Program counter
    - $M$ : the contract code array

# Appendix - Towards a better design

$$A \triangleq A_{normal} \mid \langle e \rangle_{exc} \cdot A_{normal}$$
$$A_{normal} \triangleq \langle M, pc, l, s \rangle \cdot A_{normal} \mid \epsilon$$

- Transaction Execution – Denotation
  - $A$ : Call stack of activation records
  - $\epsilon$ : empty call stack, $< \epsilon >_{exc}$ : exception thrown
  - $\mu = <A, \sigma>$: Virtual machine's execution state



# Appendix - Towards a better design

- Transaction Execution
  - $id$ : the identifier of the to-be-invoked contract
  - $v$ : the value to be deposited to the contract
  - $l$ : an data array capturing the values of input parameters
  - Transaction $= <id, v, l>$
  - features
    - Atomicity
    - Consistency

$$\textsc{TX-success}\ \frac{T \equiv \langle id, v, l\rangle \qquad M \leftarrow \mathsf{Lookup}(\sigma, id) \\ \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle\langle M, 0, l, \epsilon\rangle \cdot \epsilon, \sigma'\rangle \leadsto^{*} \langle \epsilon, \sigma''\rangle}{\sigma \xrightarrow{T} \sigma''}$$

$$\textsc{TX-exception}\ \frac{T \equiv \langle id, v, l\rangle \qquad M \leftarrow \mathsf{Lookup}(\sigma, id) \\ \sigma' \leftarrow \sigma[id][bal \mapsto (\sigma[id][bal] + v)] \\ \langle\langle M, 0, l, \epsilon\rangle \cdot \epsilon, \sigma'\rangle \leadsto^{*} \langle\langle e\rangle_{exc} \cdot \epsilon, \bullet\rangle}{\sigma \xrightarrow{T} \sigma}$$

# Appendix - Towards a better design

- Transaction Execution
  - $EtherLite$
  - $st$ : start address
  - $sz$ : size
  - $v \in values$

$\mu \rightsquigarrow \mu'$ per $M[PC]$

*Example*

| $M[pc]$ | Conditions | $\mu$ | $\mu'$ |
|---|---|---|---|
| push $v$ | | $\langle\langle M, pc, l, s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$ |
| pop | | $\langle\langle M, pc, l, v \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma\rangle$ |
| op | op: unary operator and $v' \leftarrow$ op $v$ | $\langle\langle M, pc, l, v \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v' \cdot s\rangle \cdot A, \sigma\rangle$ |
| op | op: binary operator and $v' \leftarrow v_1$ op $v_2$ | $\langle\langle M, pc, l, v_1 \cdot v_2 \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v' \cdot s\rangle \cdot A, \sigma\rangle$ |
| bne | $z = 0$ | $\langle\langle M, pc, l, \bullet \cdot z \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma\rangle$ |
| bne | $z \neq 0$ and $\lambda$ is a valid target | $\langle\langle M, pc, l, \lambda \cdot z \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, \lambda, l, s\rangle \cdot A, \sigma\rangle$ |
| bne | $z \neq 0$ and $\lambda$ is NOT a valid target | $\langle\langle M, pc, l, \lambda \cdot z \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle e\rangle_{exc} \cdot A, \sigma\rangle$ |
| mload | $v \leftarrow l[i]$ | $\langle\langle M, pc, l, i \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$ |
| mstore | $l' \leftarrow l[i \mapsto v]$ | $\langle\langle M, pc, l, i \cdot v \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l', s\rangle \cdot A, \sigma\rangle$ |
| sload | $id \leftarrow$ address of the executing contract<br>$v \leftarrow \sigma[id][i]$ | $\langle\langle M, pc, l, i \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$ |
| sstore | $id \leftarrow$ address of the executing contract<br>$\sigma' \leftarrow \sigma[id][i \mapsto v]$ | $\langle\langle M, pc, l, i \cdot v \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, s\rangle \cdot A, \sigma'\rangle$ |
| call | $id \leftarrow$ address of the executing contract<br>$a' \leftarrow \langle M, pc, l, s\rangle$<br>$M' \leftarrow$ Lookup$(\sigma, \gamma)$<br>$\sigma' \leftarrow \sigma[id][bal \mapsto \sigma[id][bal] - z]$<br>$\sigma'' \leftarrow \sigma'[\gamma][bal \mapsto \sigma[id][bal] + z]$ | $\langle\langle M, pc, l, \gamma \cdot z \cdot st \cdot sz \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M', 0, l', \epsilon\rangle \cdot a' \cdot A, \sigma''\rangle$ |
| call | $id \leftarrow$ address of the executing contract<br>$\sigma[id][bal] < v$ or $\lvert A\rvert = 1023$ | $\langle\langle M, pc, l, \bullet \cdot v \cdot \bullet \cdot \bullet \cdot \bullet \cdot \bullet \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, 0 \cdot s\rangle \cdot A, \sigma\rangle$ |
| return | | $\langle\langle M, pc, \bullet, \bullet\rangle \cdot \epsilon, \sigma\rangle$ | $\langle \epsilon, \sigma\rangle$ |
| return | $a' \equiv \langle M', pc', l'_0, st' \cdot sz' \cdot s'\rangle$<br>$n \leftarrow min(sz', sz)$<br>$0 \leq i < n : l'_{i+1} \leftarrow l'_i[st'+i \mapsto l[st+i]]$ | $\langle\langle M, pc, l, st \cdot sz \cdot s\rangle \cdot a' \cdot A, \sigma\rangle$ | $\langle\langle M', pc'+1, l'_n, 1 \cdot s'\rangle \cdot A, \sigma\rangle$ |
| EXC | exceptional halting of callee | $\langle\langle e\rangle_{exc} \cdot \langle M, pc, l, st \cdot sz \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, 0 \cdot s\rangle \cdot A, \sigma\rangle$ |
| suicide | [illegible]<br>$a' \equiv \langle M', pc', l'_0, \bullet \cdot \bullet \cdot s'\rangle$<br>$\sigma' \leftarrow \sigma[\gamma][bal \mapsto (\sigma[\gamma][bal] + \sigma[id][bal])]$<br>$\sigma'' \leftarrow \sigma'[id][bal \mapsto 0]$<br>Register $id$ for later deletion | $\langle\langle M, pc, \bullet, \gamma \cdot s\rangle \cdot a' \cdot A, \sigma\rangle$ | $\langle\langle M', pc'+1, l'_n, 1 \cdot s'\rangle \cdot A, \sigma\rangle$ |


| | | | |
|---|---|---|---|
| sload | $id \leftarrow$ address of the executing contract<br>$v \leftarrow \sigma[id][i]$ | $\langle\langle M, pc, l, i \cdot s\rangle \cdot A, \sigma\rangle$ | $\langle\langle M, pc+1, l, v \cdot s\rangle \cdot A, \sigma\rangle$ |

# References


- http://www.ethdocs.org/
- https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-krupp.pdf
- https://www.usenix.org/sites/default/files/conference/protected-files/security18_slides_krupp.pdf
- https://solidity.readthedocs.io/en/latest/
- https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf
- https://consensys.github.io/smart-contract-best-practices/recommendations/
- https://hackernoon.com/smart-contract-security-part-1-reentrancy-attacks-ddb3b2429302
- https://www.lazenca.net/pages/viewpage.action?pageId=6324534
- https://en.wikipedia.org/wiki/Symbolic_execution#Path_explosion
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch
- https://solidity.readthedocs.io/en/v0.5.7/control-structures.html#error-handling-assert-require-revert-and-exceptions
- https://solidity.readthedocs.io/en/v0.5.7/units-and-global-variables.html#error-handling
- https://users.encs.concordia.ca/~clark/papers/2017_cacm.pdf
- http://sigpl.or.kr/school/2018s/slides/0820-02-JonghyupLee.pdf