Smart Contract 보안의 흥미로운 관점들

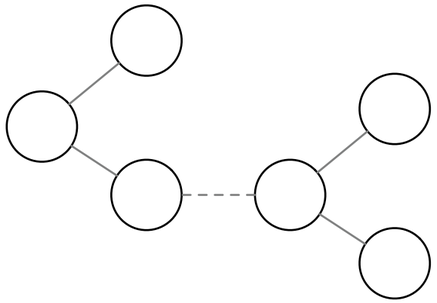
2018-09-18
카이스트 정보보호대학원 세미나

이종협

Blockchain intro

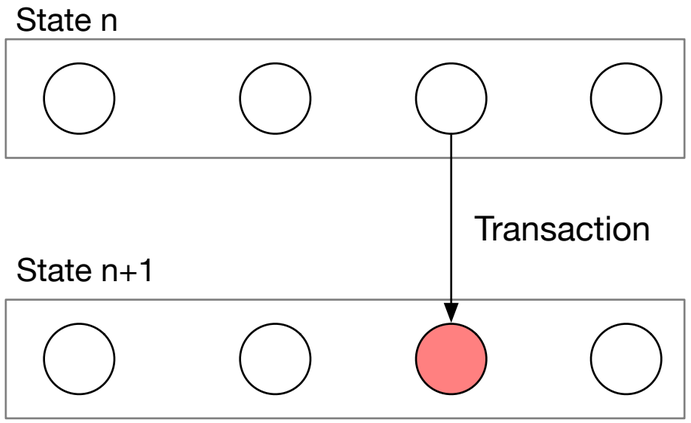
Bitcoin

Transaction Model



Ethereum

State + account model



+ EVM
  (Ethereum Virtual Machine)

Hyperledger

Framework

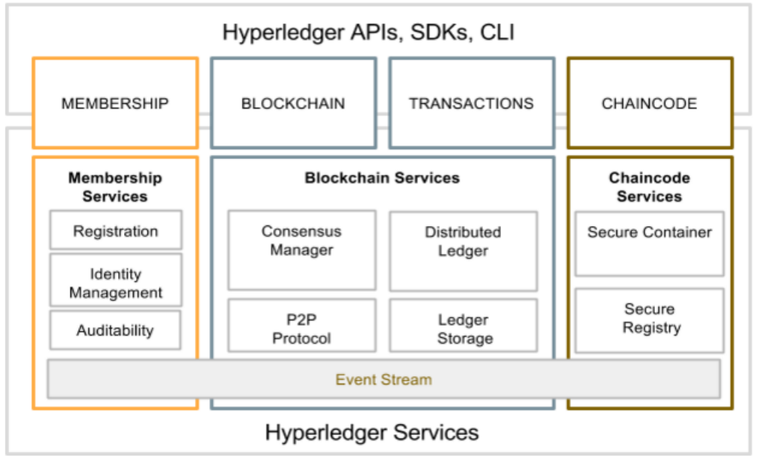
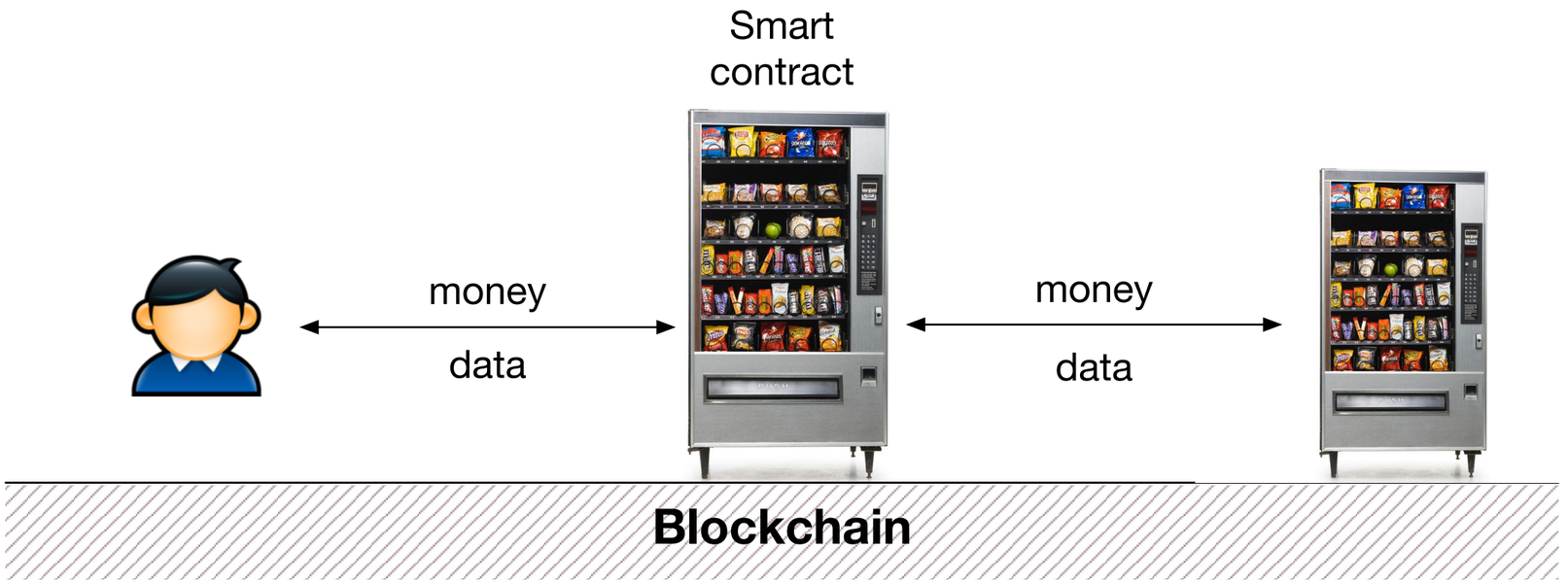


Figure 2: Hyperledger reference architecture

Smart contract

“Contract를 구현하고, 강제하고, 실행시켜 주는 code”

- 믿지 않는 사용자간의 agreement + coordination
- 블록체인에 복잡한 기능을 제공



Solidity code

Storage

Constructor

Function (Public)

Fallback function

```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken( uint256 initialSupply ) public {
    /* (or constructor ( uint256 initialSupply ) public { ) */
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }


    /* Send coins */
    function transfer(address _to, uint256 _value) public {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
    }

    /* Fallback */
    function () payable {
        ...
    }
}
```

Smart contracts

어떻게 볼 것인가?

Vending machine

Balance (Money!)

Storage

Distributed objects

≈

Threads using concurrent objects in shared memory

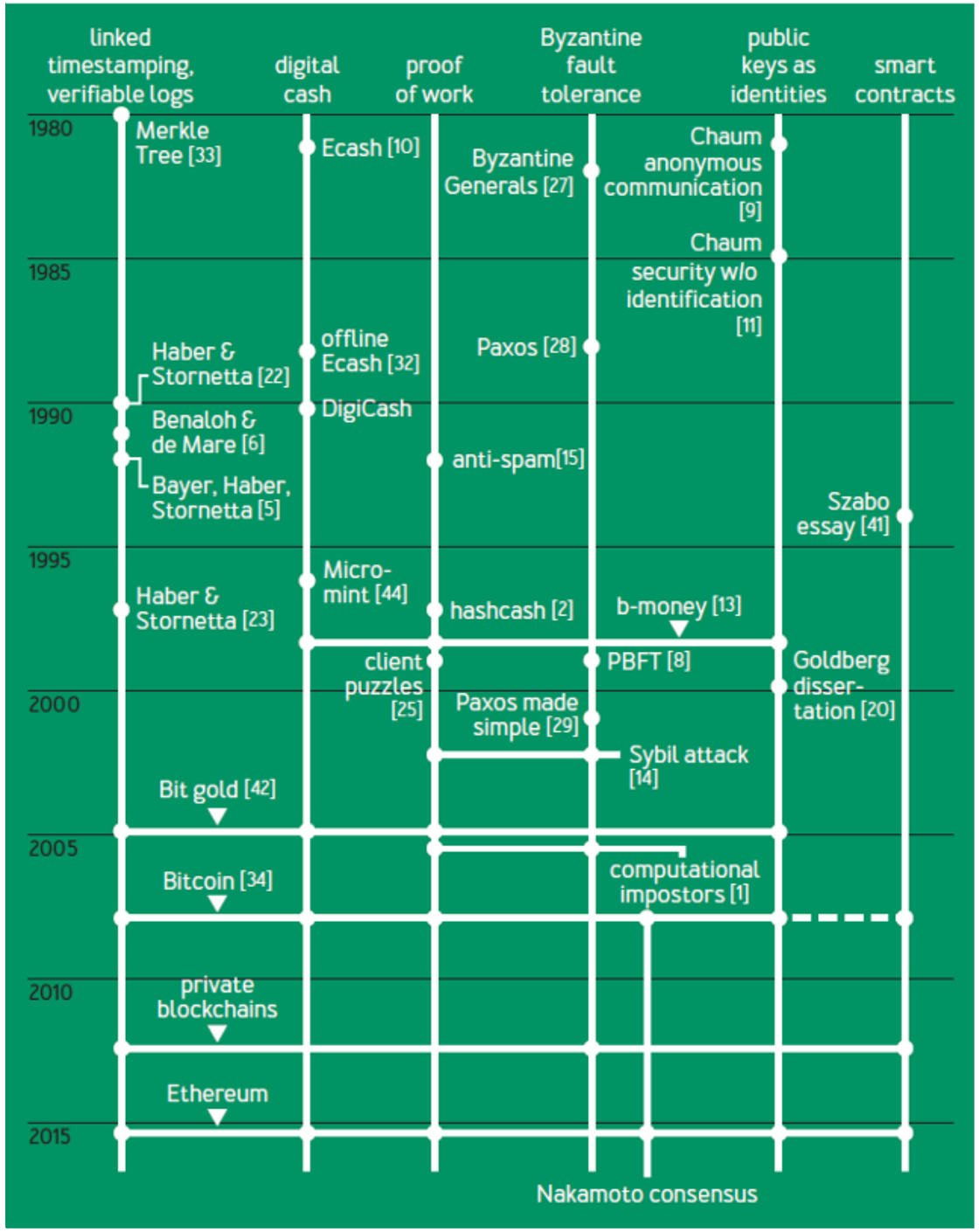
Secure execution (External)

!

어떠한 의미를 가지는가?

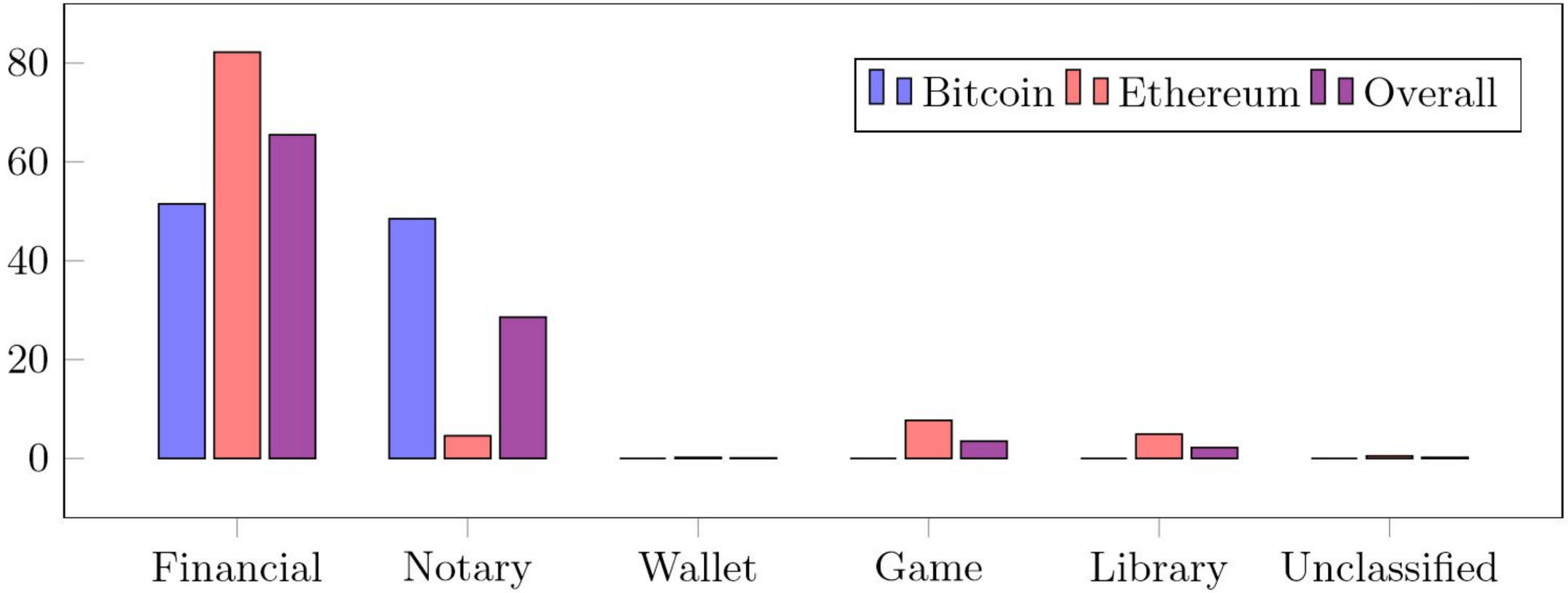
Blockchain에서 Smart contract란 어떤 의미인가?

Academic Pedigree



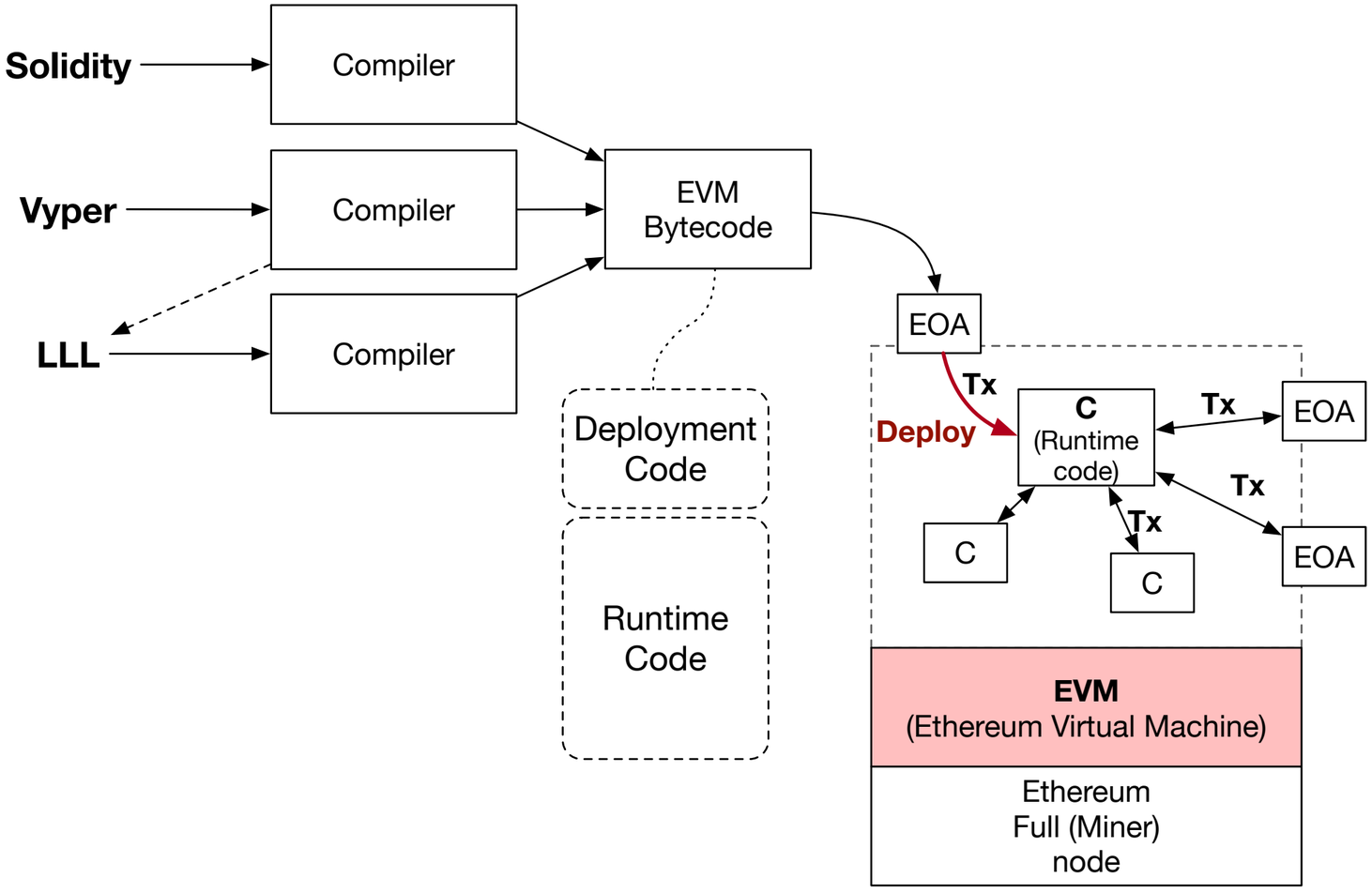
from “Bitcoin’s academic pedigree” Narayanan *et al.*

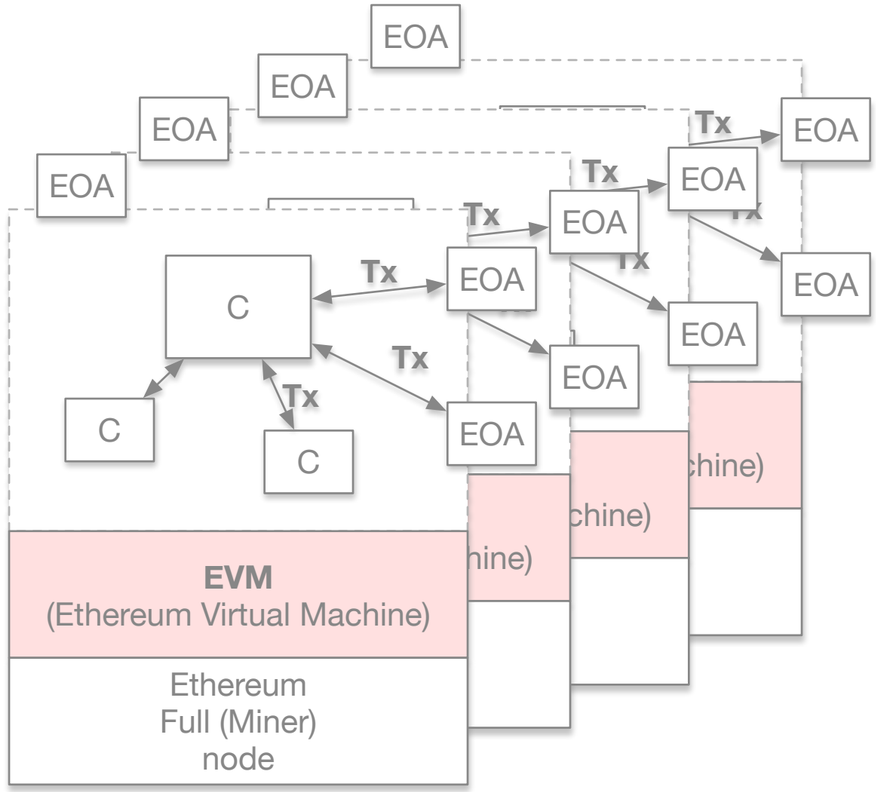
Smart contracts - category



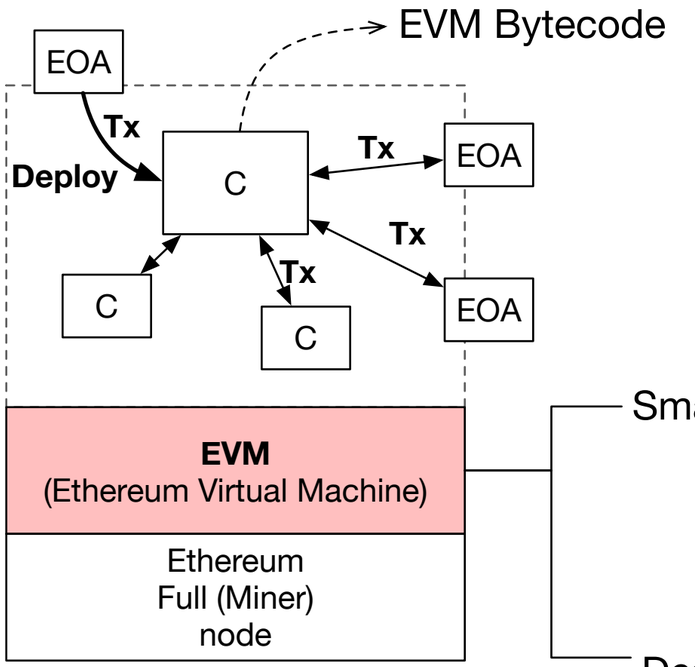
from “an empirical analysis of smart contracts” Bartoletti *et al.*

Smart contract lifecycle



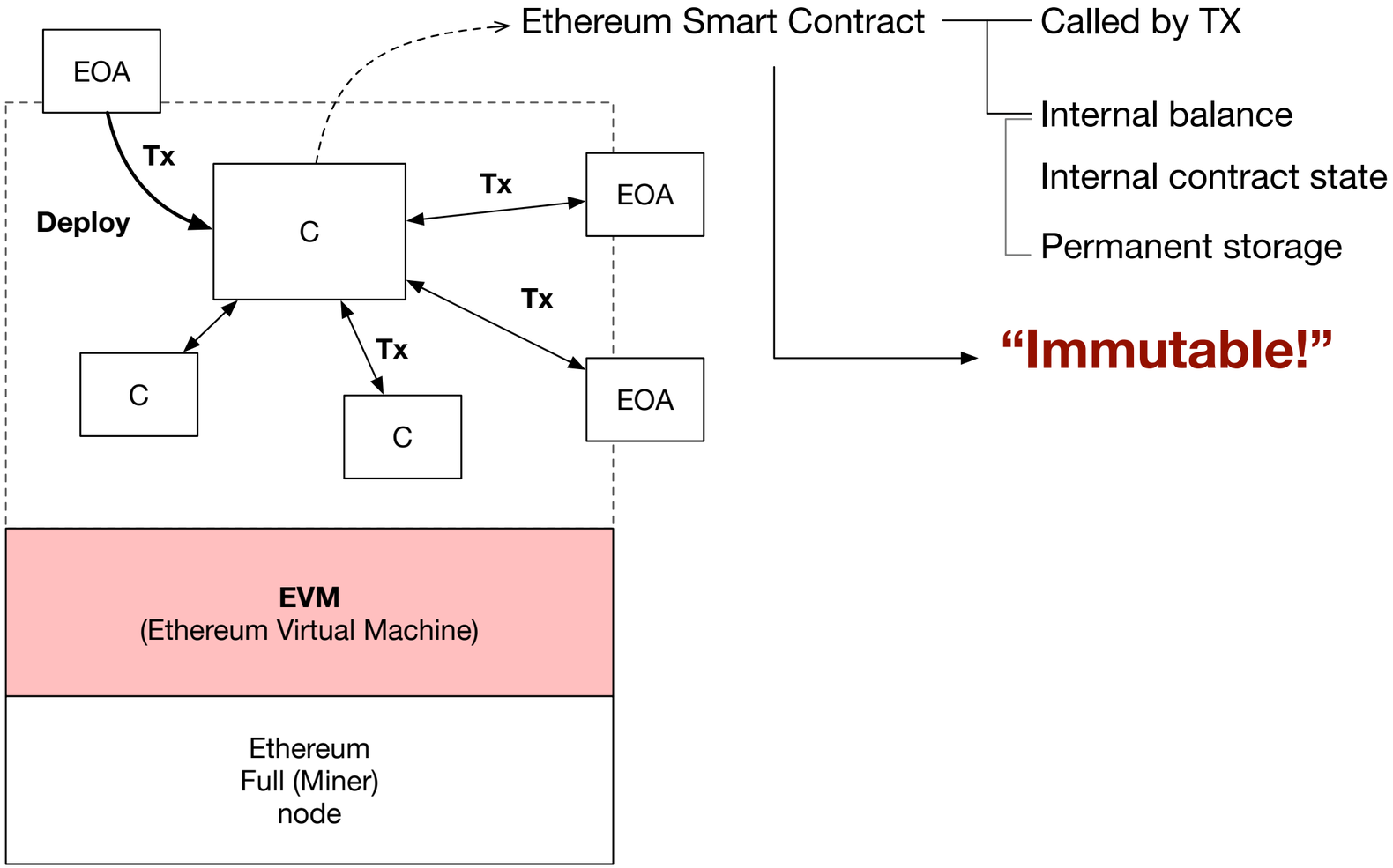


Ethereum Virtual Machine



- Smart contract를 위한 execution model
  - Redundantly parallel
  - Turing complete!
  - GAS
- Design Goal
  - Simplicity
  - Space efficiency
  - Determinism?
  - Specialization
  - Security

Ethereum Virtual Machine

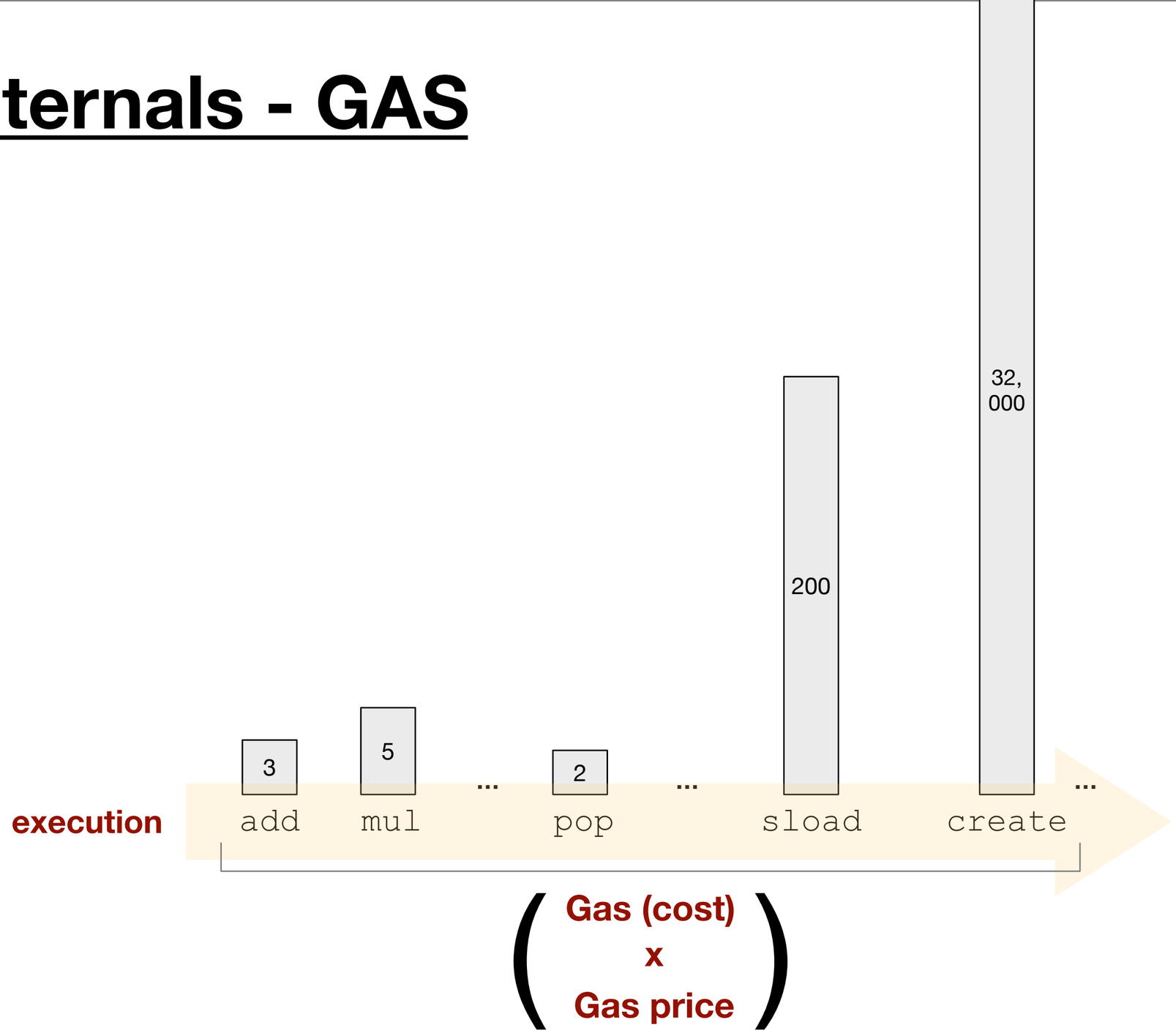


Ethereum Smart Contract

- Called by TX
- Internal balance
- Internal contract state
- Permanent storage

“Immutable!”

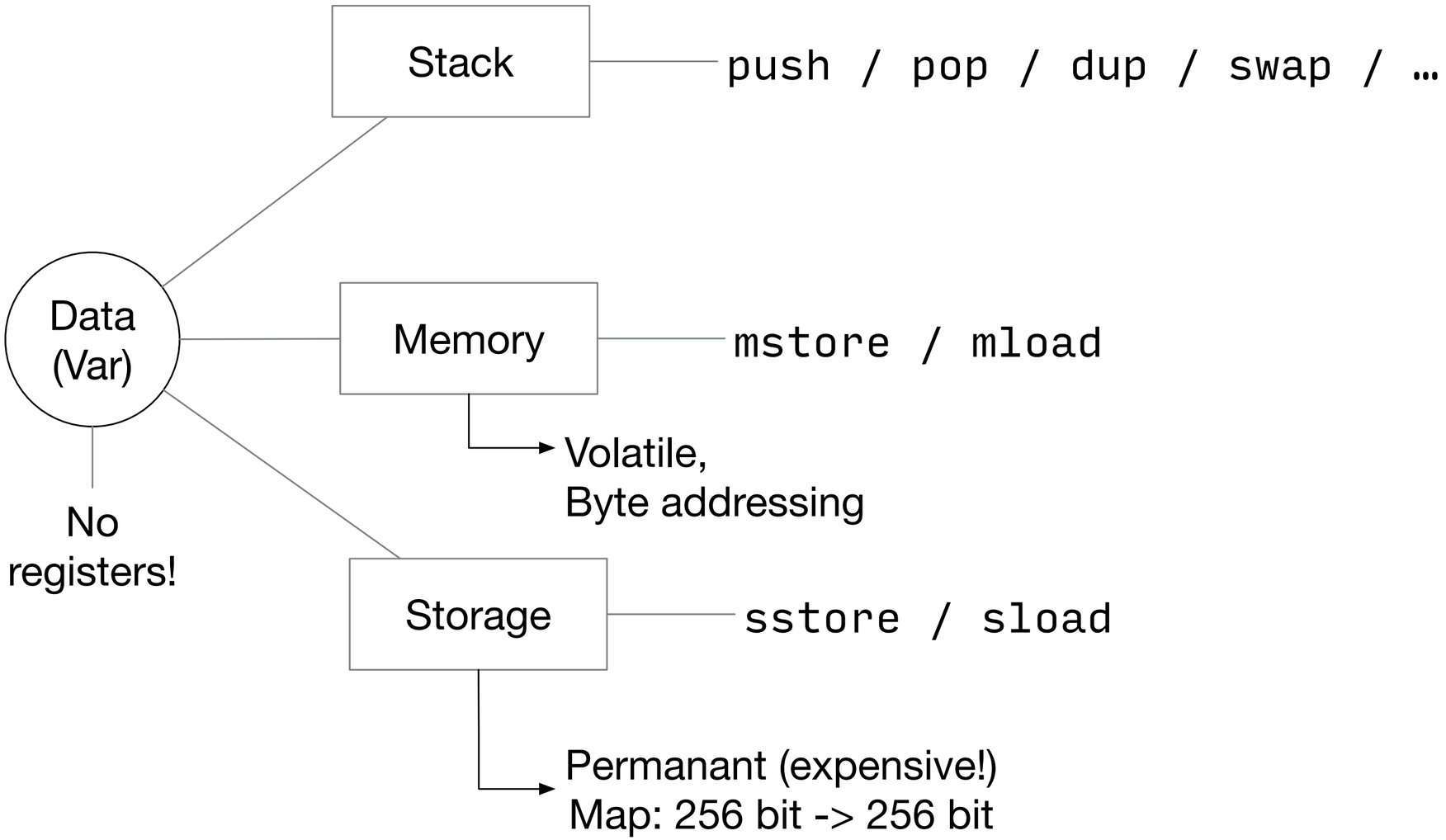
EVM internals - GAS



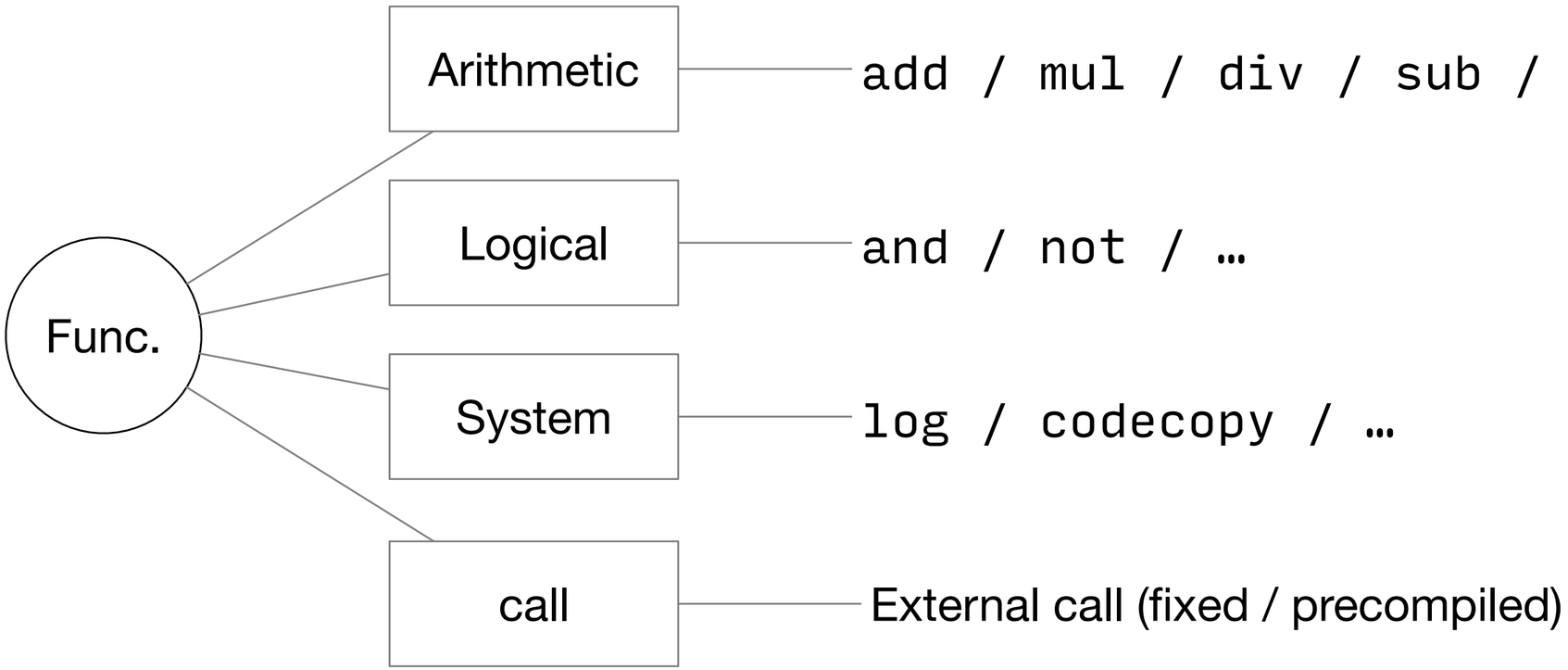
EVM assembly code

```
PUSH 0
DUP1
PUSH 100
EXP
DUP2
SLOAD
DUP2
PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00
MUL
NOT
AND
SWAP1
DUP4
PUSH FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
AND
MUL
OR
SWAP1
SSTORE
POP
```

EVM internals - data



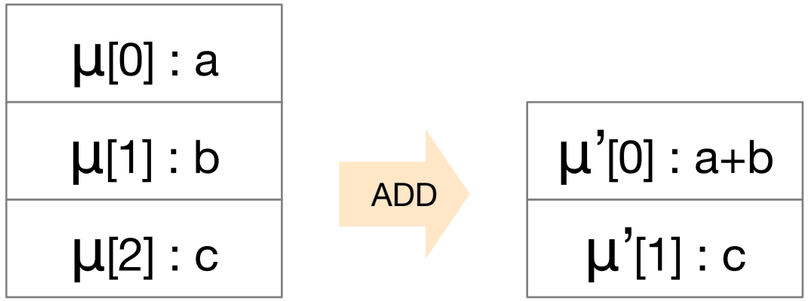
EVM internals - data



EVM instructions - “Yellow paper”

Value	Mnemonic	(pop) In δ	(push) out α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] \times \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x51	MLOAD	1	1	Load word from memory. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{m}}[\boldsymbol{\mu}_{\mathbf{s}}[0] \ldots (\boldsymbol{\mu}_{\mathbf{s}}[0] + 31)]$ $\boldsymbol{\mu}'_{\mathrm{i}} \equiv \max(\boldsymbol{\mu}_{\mathrm{i}}, \lceil (\boldsymbol{\mu}_{\mathbf{s}}[0] + 32) \div 32 \rceil)$
0x54	SLOAD	1	1	Load word from storage. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\sigma}[I_{\mathrm{a}}]_{\mathbf{s}}[\boldsymbol{\mu}_{\mathbf{s}}[0]]$

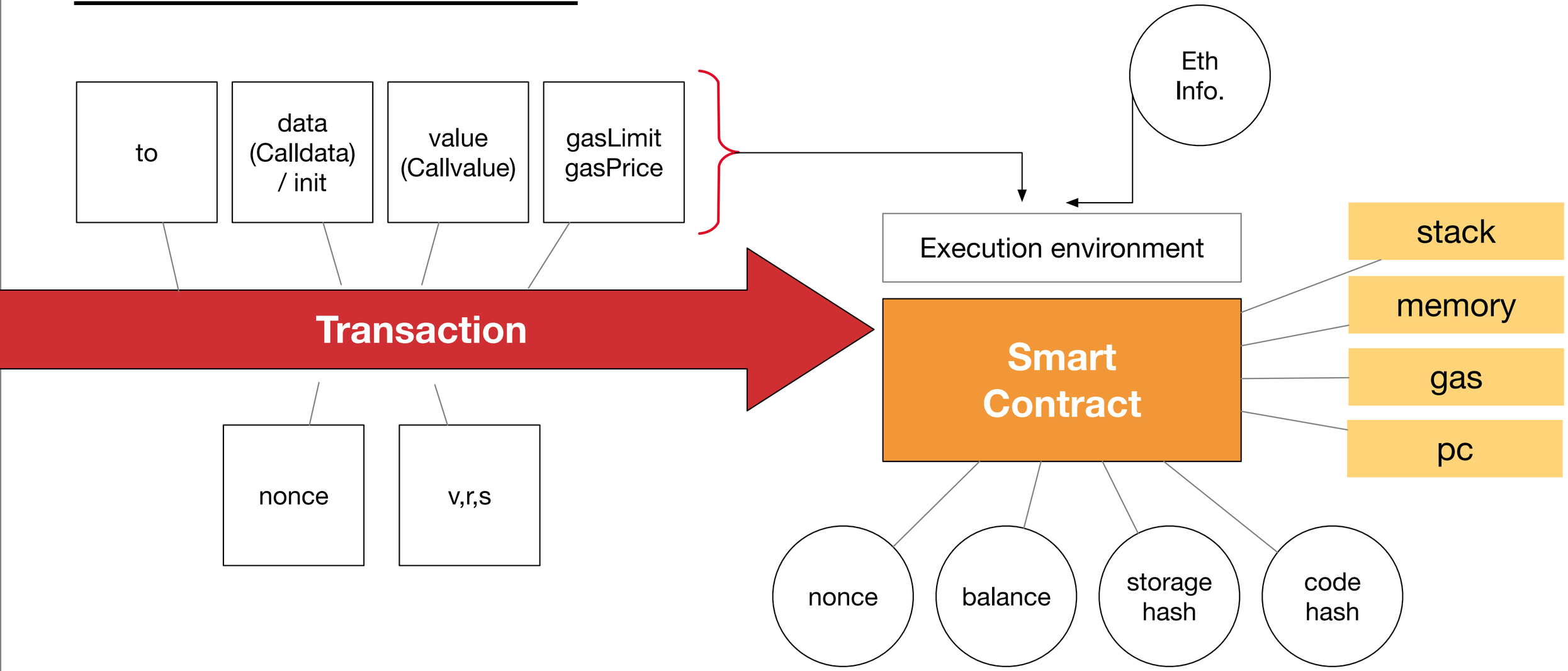
stack



μ: Machine state

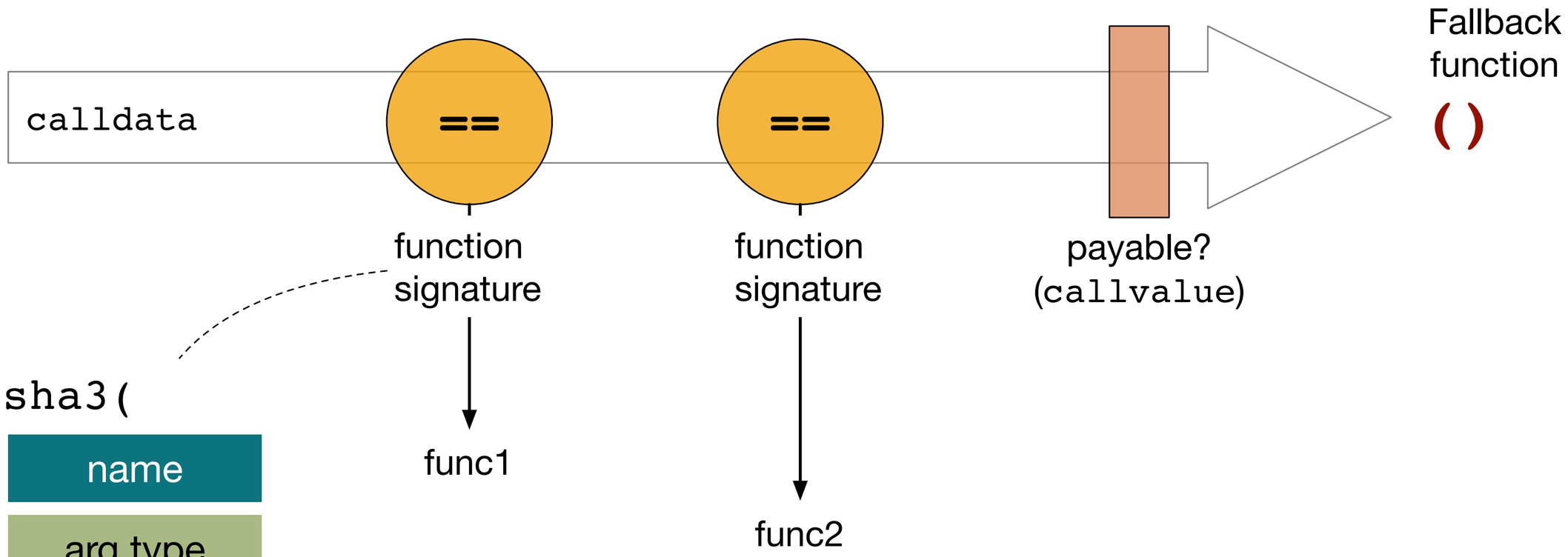
σ: World state

Execution model

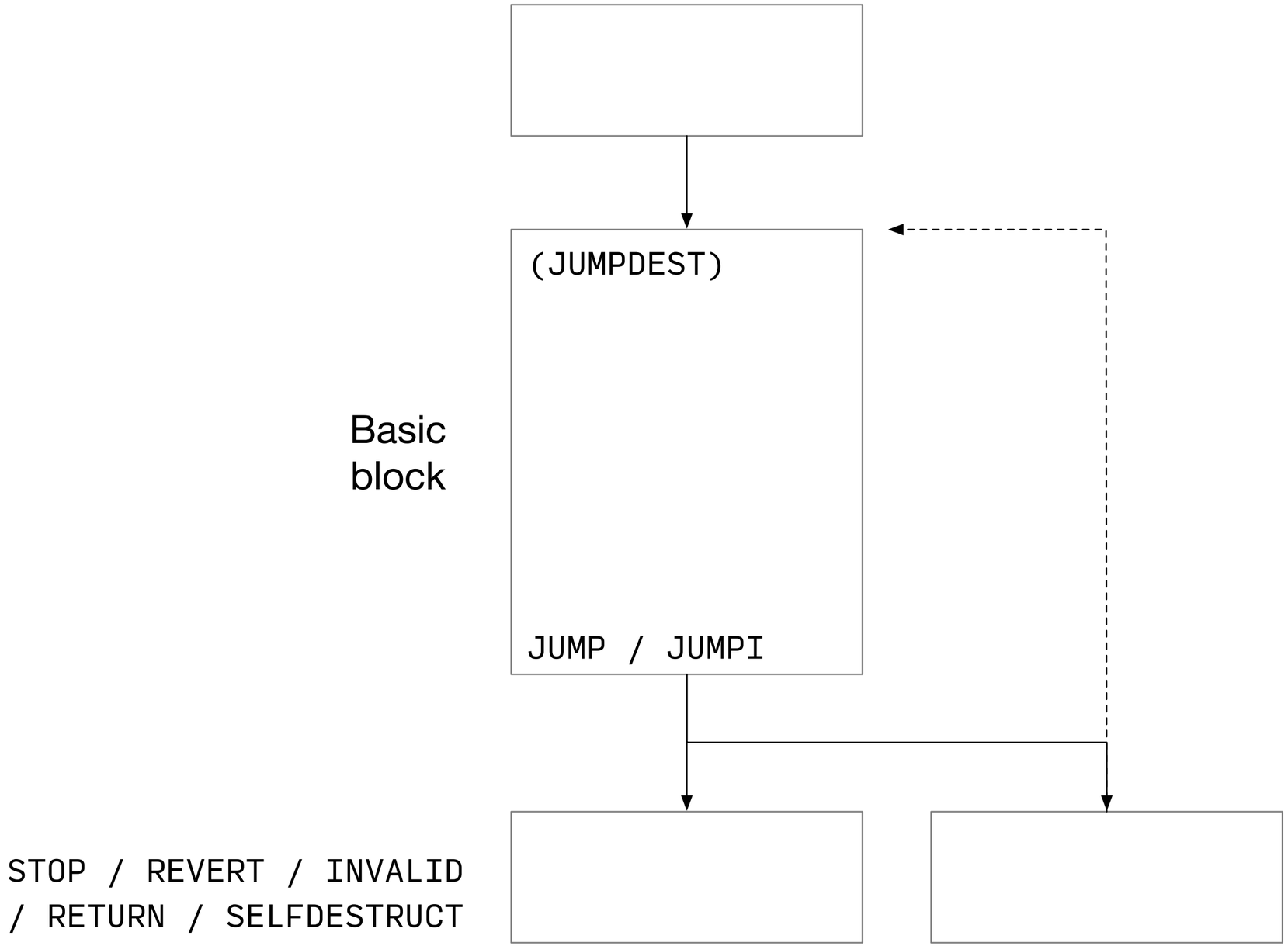


Function call handling

Function call과 fall back

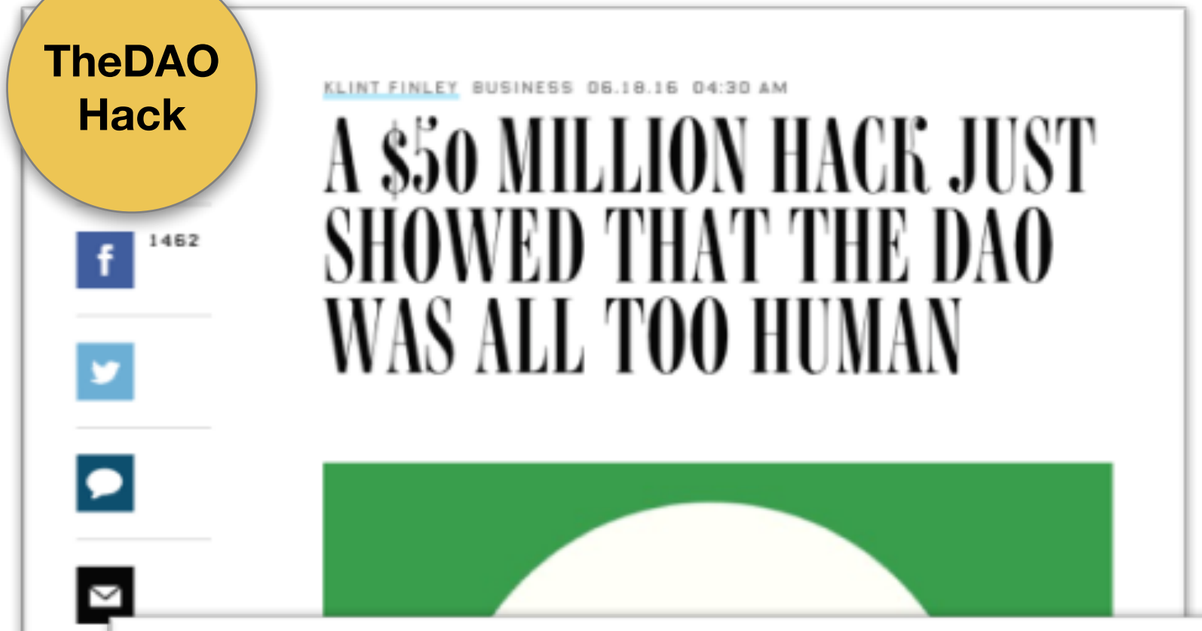


EVM internals - control



무엇이 문제인가?

TheDAO Hack

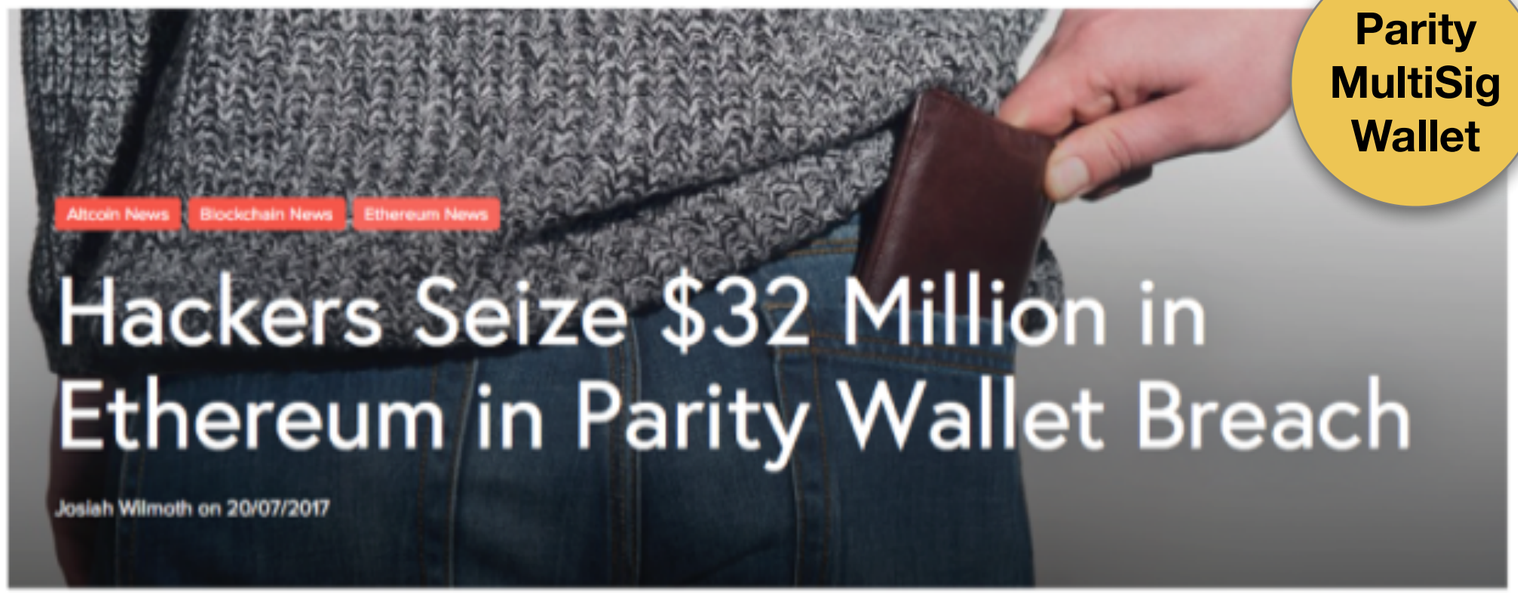


KLINT FINLEY BUSINESS 06.18.16 04:30 AM

A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN

1462

Parity MultiSig Wallet



322

Advertisement

왜 해킹의 대상이 되는가?

- Smart contract는 기본적으로 항상 online + open
- 공격자가 즉각적인 reward를 얻는다.
- Immutable!
- 개발자들에게도 생소한 execution model
- Solidity의 abstraction과 실제 EVM과의 mismatch

⋮

Smart contract를 작성한다는 것은..

> I want you to write a program that has to run in a concurrent environment under Byzantine circumstances where any adversary can invoke your program with any arguments of their choosing. The environment in which your program executes (and hence any direct or indirect environmental dependencies) is also under adversary control. If you make a single exploitable mistake or oversight in the implementation, or even in the logical design of the program, then either you personally or perhaps the users of your program could lose a substantial amount of money. Where your program will run, there is no legal recourse if things go wrong. Oh, and once you release the first version of your program, you can never change it. It has be right first time.

from *blog.acolyer.org*

취약점?

```
(1)   contract Wallet {
(2)     mapping(address => uint) private userBalances;
(3)     function withdrawBalance() {
(4)       uint amountToWithdraw = userBalances[msg.sender];
(5)       if (amountToWithdraw > 0) {
(6)         msg.sender.call(userBalances[msg.sender]);
(7)         userBalances[msg.sender] = 0;
(8)       }
(9)     }
(9)     ...
(10)  }
```

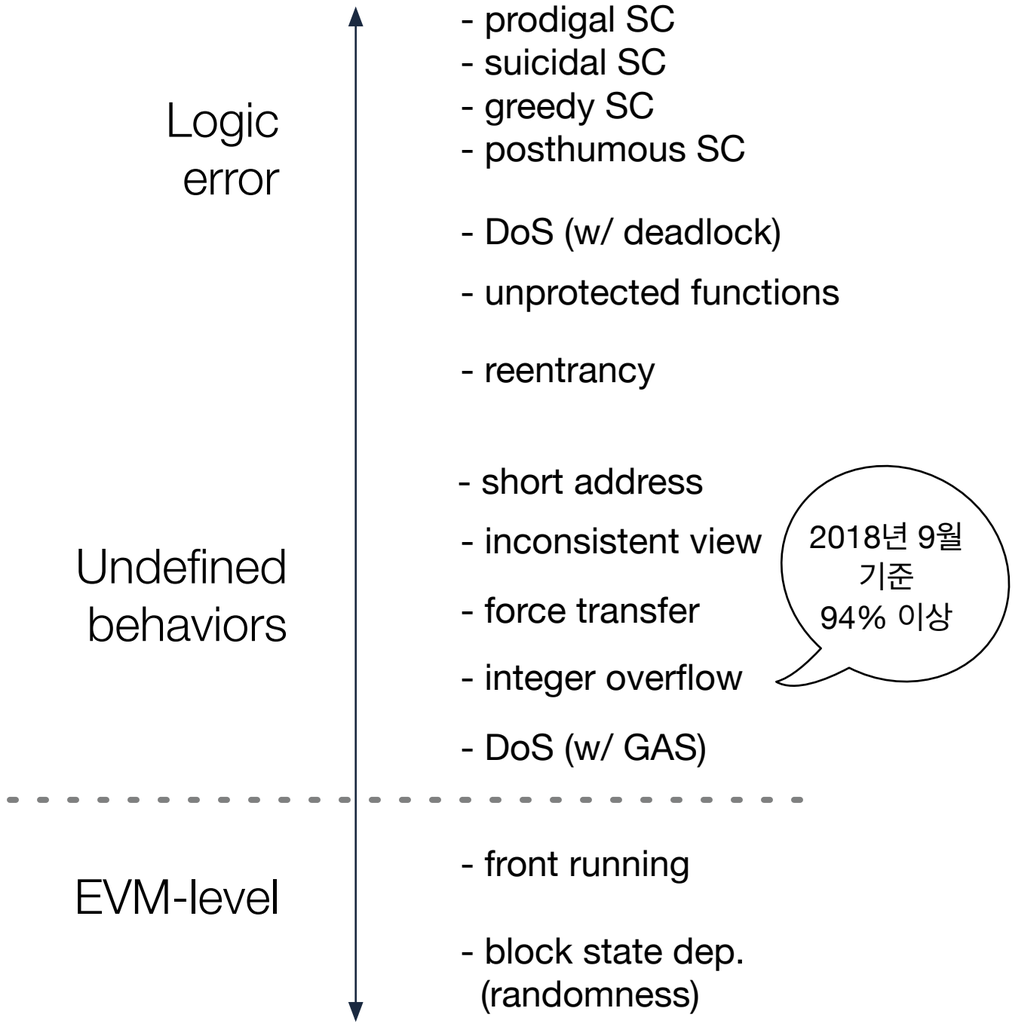
```
(1)   contract AttackerContract {
(2)     function () {
(3)       Wallet wallet;
(4)       wallet.withdrawBalance();
(5)     }
(6)   }
```

1. 조건을 확인하고
2. state를 변경하고
3. action

from “ZEUS: Analyzing Safety of Smart Contracts” Kalra *et al.*

from “ZEUS: Analyzing Safety of Smart Contracts” Kalra *et al.*

Smart contract 취약점



Unchecked send

```
(1)  if(gameHasEnded && !prizePaidOut) {
(2)    winner.send(1000); // send a prize to the winner
(3)    prizePaidOut = True;
(4)  }
```

```
(1)  while (balance >
             persons[payoutCursor_Id_].deposit/100*115) {
(2)    payout = persons[payoutCursor_Id_].deposit/100*115;
(3)    persons[payoutCursor_Id].EtherAddress.send(payout);
(4)    balance -= payout;
(5)    payoutCursor_Id_++;
(6)  }
```

Incorrect logic

```
(1)   uint payout = balance/participants.length;
(2)   for (var i = 0; i < participants.length; i++)
(3)     participants[i].send(payout);
```

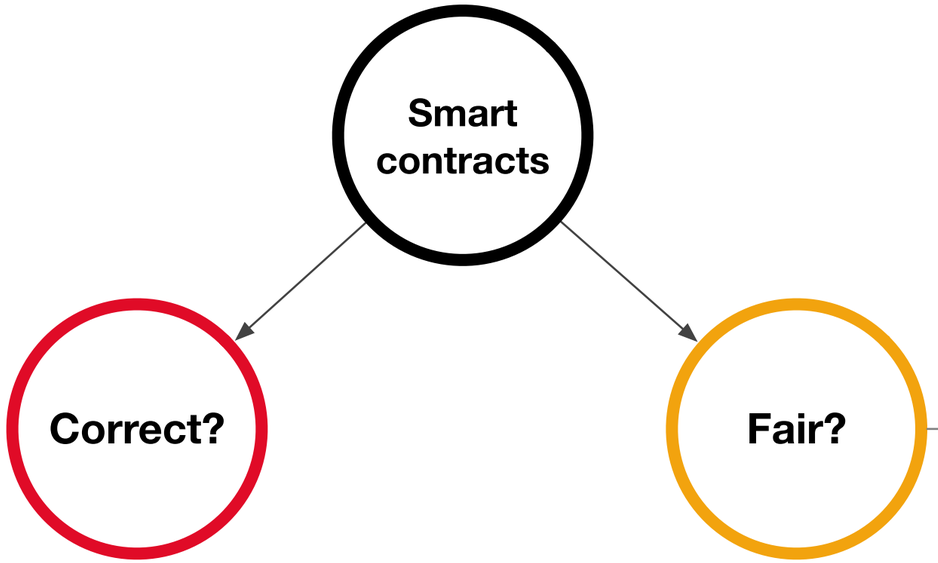
Integer overflow

+

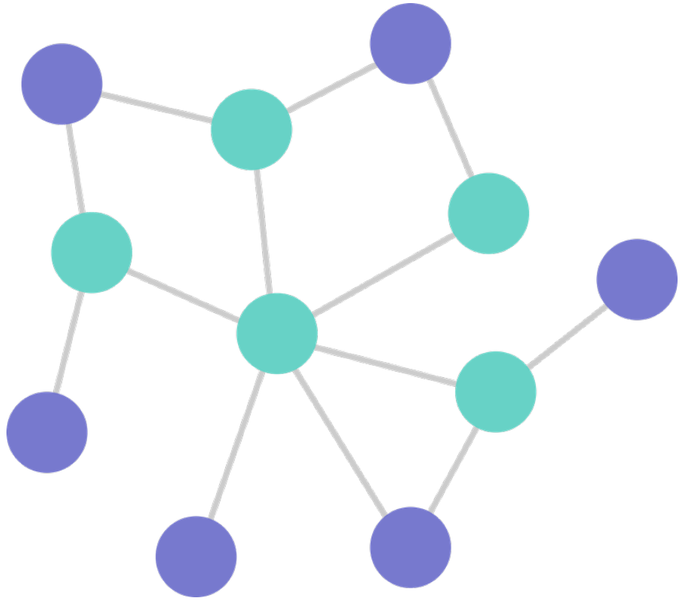
The Ethernaut:
https://ethernaut.zeppelin.solutions

Smart contract의 안전성이란?

Correctness와 fairness의 기준은 무엇인가?



무엇에 대하여?



Token economy

Decentralized governance 구조와
Incentive mechanism으로 정의

Smart contract가 이것을 위배하는가?

우선 지금 어떠한가?

```
modifier onlyFromWallet {
    require(msg.sender != walletAddress);
    _;
}
```

CVE-2018-14576

```
function mintTokens(address _to, uint256 _amount) {
  if (msg.sender != icoContractAddress) throw;
  if (restrictedAddresses[_to]) throw;
  if (balances[_to] + _amount < balances[_to]) throw;

  balances[_to] += _amount;
  supply += _amount;
  Mint(_to, _amount);
  Transfer(0x0, _to, _amount);
}
```

CVE-2018-14084

```
function sell(uint256 amount) public {
        require(this.balance >= amount * sellPrice);
        _transfer(msg.sender, this, amount);
        msg.sender.transfer(amount * sellPrice);
}
```

```
function transfer(address _to, uint256 _value) public {
    _transfer(msg.sender, _to, _value);
}
```

CVE-2018-10705

```
136       function unlockToken() onlyOwner {
137         locked = false;
138       }
139
140       bool public balancesUploaded = false;
141       function uploadBalances(address[] recipients, uint256[] balances) onlyOwner {
142         require(!balancesUploaded);
143         uint256 sum = 0;
144         for (uint256 i = 0; i < recipients.length; i++) {
145           balanceOf[recipients[i]] = safeAdd(balanceOf[recipients[i]], balances[i]);
146           sum = safeAdd(sum, balances[i]);
147         }
148         balanceOf[owner] = safeSub(balanceOf[owner], sum);
149       }
150       function lockBalances() onlyOwner {
151         balancesUploaded = true;
152       }
153   }
```

```
22  contract Owned {
23    address public owner;
24    function Owned() {
25      owner = msg.sender;
26    }
27    function setOwner(address _owner) returns (bool success) {
28      owner = _owner;
29      return true;
30    }
31    modifier onlyOwner {
32      require(msg.sender == owner);
33      _;
34    }
35  }
36
37  contract AURA is SafeMath, Owned {
```

(https://blog.peckshield.com/2018/05/03/ownerAnyone/)

Randomness?

```
contract NumberLottery
{
  uint256 private  randomNumber
          = uint256( keccak256(now) ) % 10 + 1;
  uint256 public prizeFund;
  uint256 public minBet = 0.1 ether;
  address owner = msg.sender;

  struct GameHistory
  {
    address player;
    uint256 number;
  }
```

```
function StartGame(uint256 _number) public payable
{
    if(msg.value >= minBet && _number <= 10)
    {
        GameHistory gameHistory;
        gameHistory.player = msg.sender;
        gameHistory.number = _number;
        log.push(gameHistory);

        if (_number == randomNumber)
        {
            msg.sender.transfer(this.balance);
        }

        randomNumber = uint256( keccak256(now) ) % 10 + 1;
        prizeFund = this.balance;
    }
}
```

(Contract: 0x2f069a1D7A052052458e8b5511E91221EB337c52)

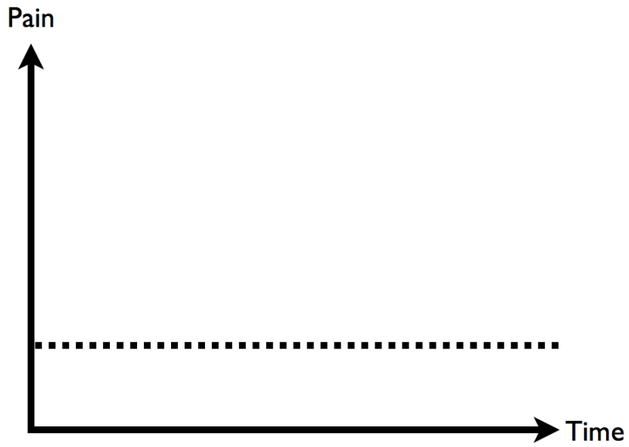
Fairness?

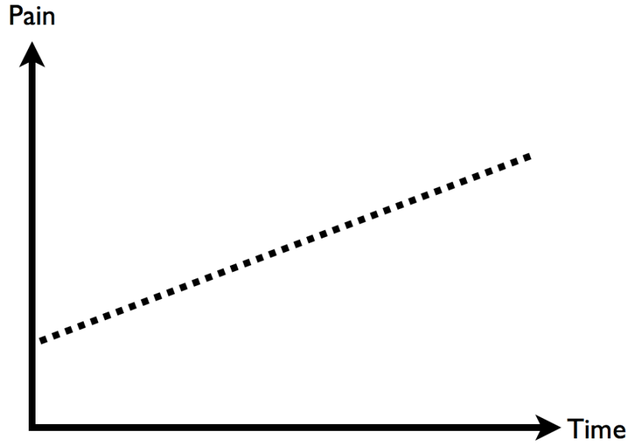
```
for(uint i = 0 ; i < NO_OF_SEATS_BID; i++){
  // Only 1-14 for sale
  assert( 0 < seats[i]  && seats[i] < 15);
  var seatNumber = uint8(seats[i]);

  var valueBid = bids[i];

  var existingSeat = table[seatNumber];
  // Min increase 1 ether
  if (existingSeat.cost + 1 ether <= valueBid){
    //Bidder takes the seat
    existingSeat.owner = bidder;
    existingSeat.cost = valueBid;
  }
  // else, money lost - medieval rules here
}
//Register how much the creator should have
creator_balance += 100 * bids.length;
// All money is stored in this contract until payout time
```

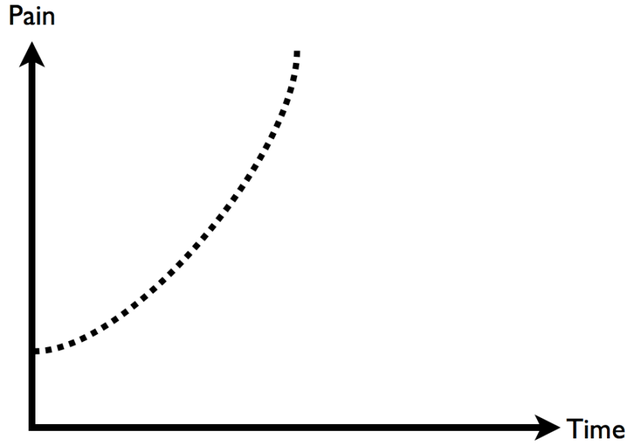
```
# '0x69f30401' for function 'bid(address,uint256[],uint256[])'
sig = "69f30401"



args = [ sig, #method
 "000000000000000000000000000000000000000000000000cafebabe", #bidder
 "0000000000000000000000000000000000000000000000000000000000000060", #datapart_1_param
 "00000000000000000000000000000000000000000000000000000000000000A0", #datapart_2_param
 "F0000000000000000000000000000000000000000000000000000000000000001", #len_of_array_1
 "0000000000000000000000000000000000000000000000000000000000000003", #seat 3 #array_1
 "F0000000000000000000000000000000000000000000000000000000000000001", #len_of_array_2
 "000000000000000000000000000000000000000000000000de0b6b3a7640000", # 1 ether #array_2
]
data = "".join(args)
```

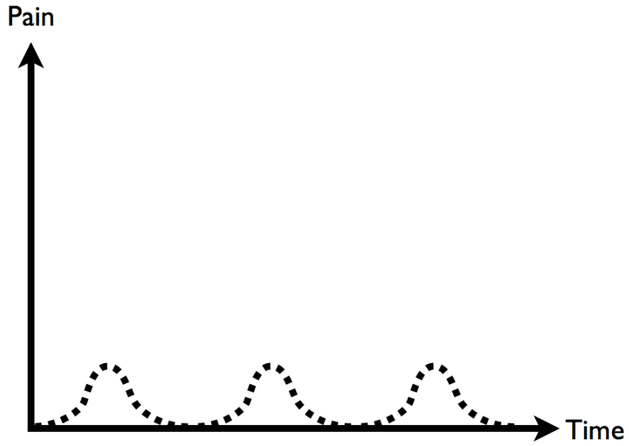
무엇인가
근본적으로
잘못되었다.

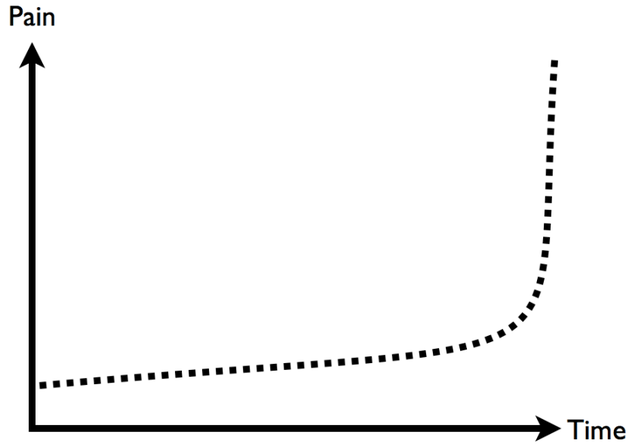
Problem shapes





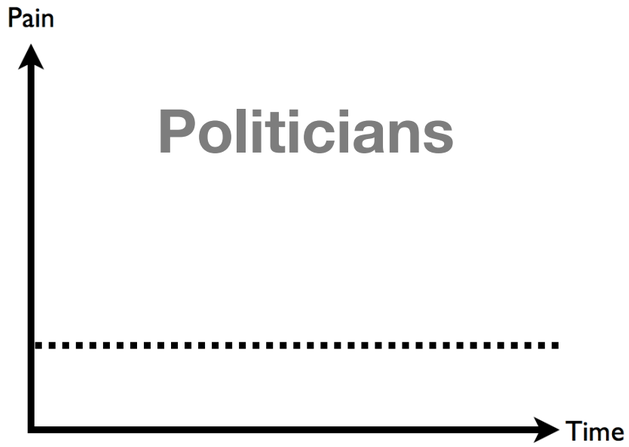


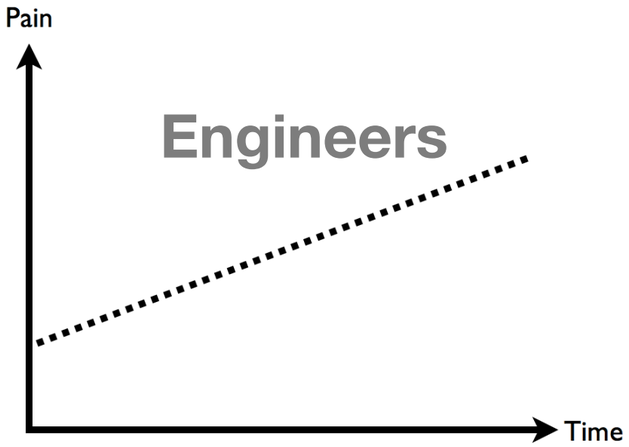


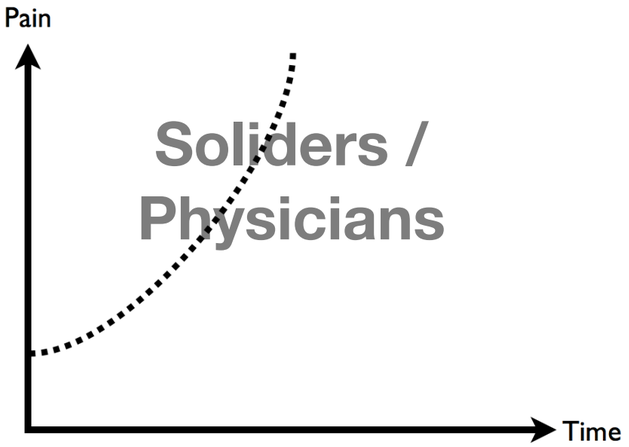


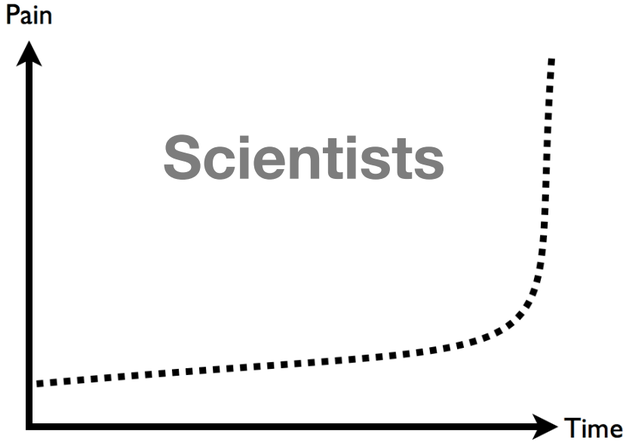
(http://matt.might.net/articles/problem-shapes/)

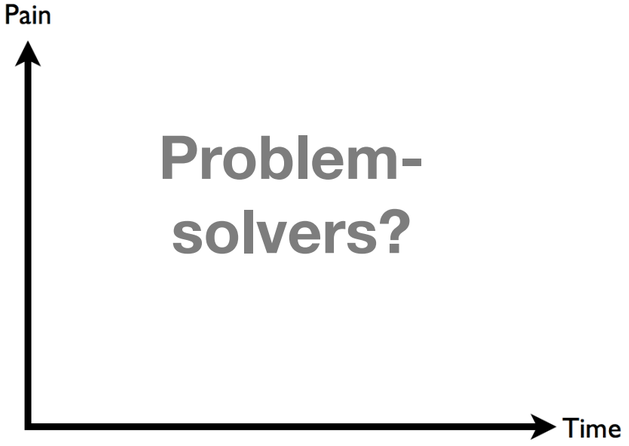
Problem shapes





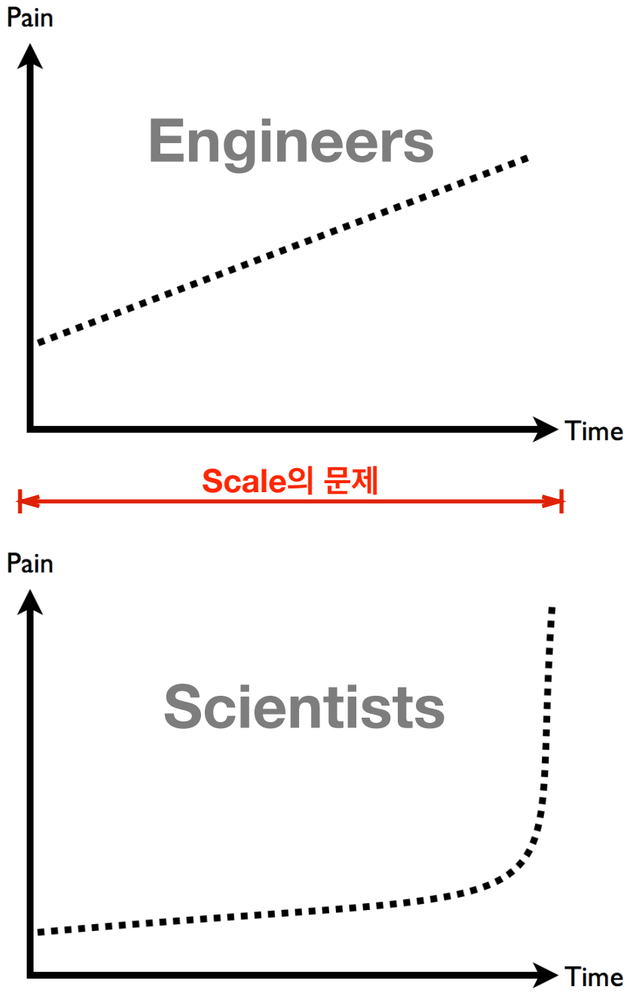


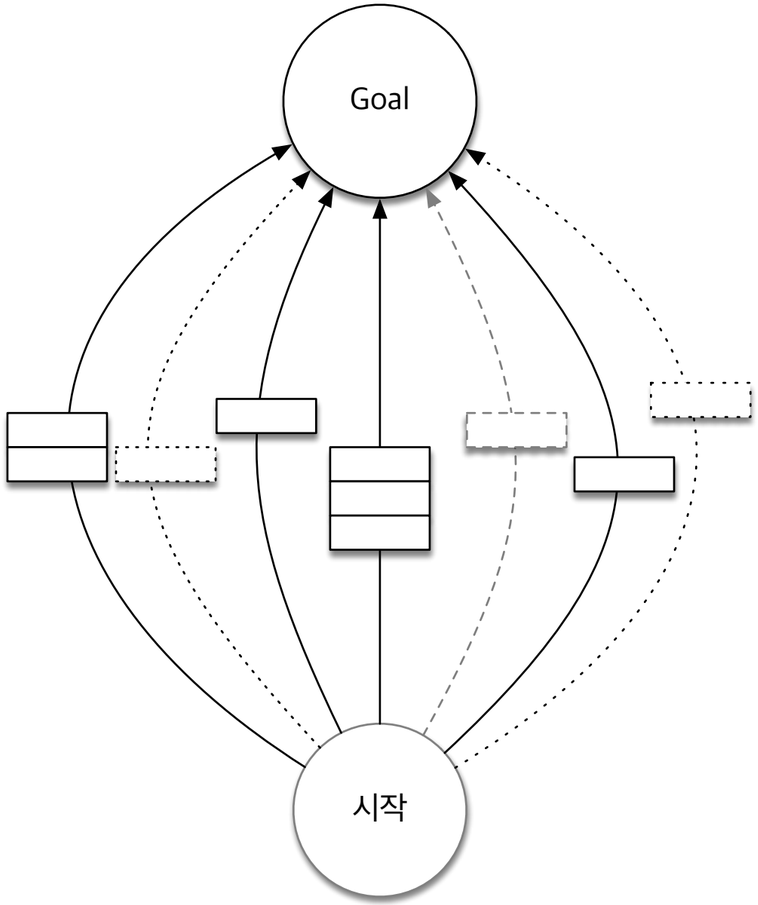




(http://matt.might.net/articles/problem-shapes/)

Science vs. Engineering

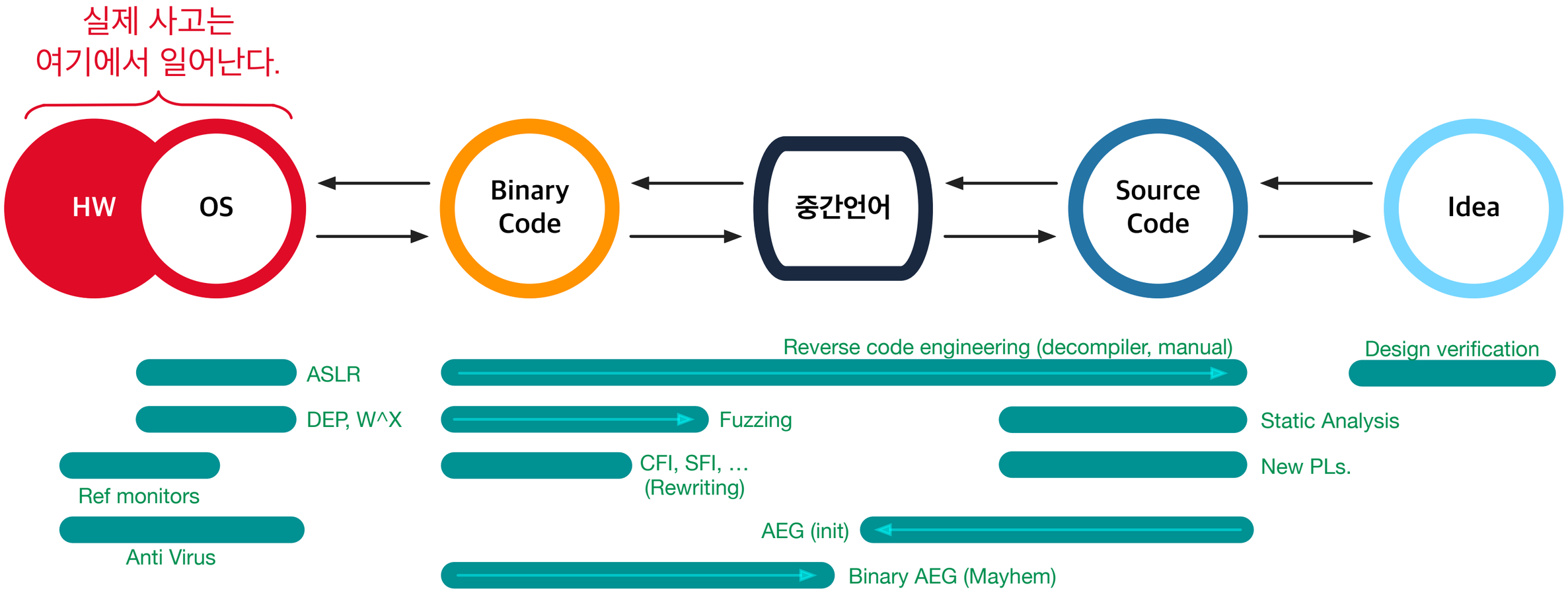




어떠한
가정, Constraints, 현실을
받아들일 것인가?

무엇을 어디에서 부터 고민하는가?

Software security에서의 (기존) 접근 방법



반복되는
창과 방패의 싸움

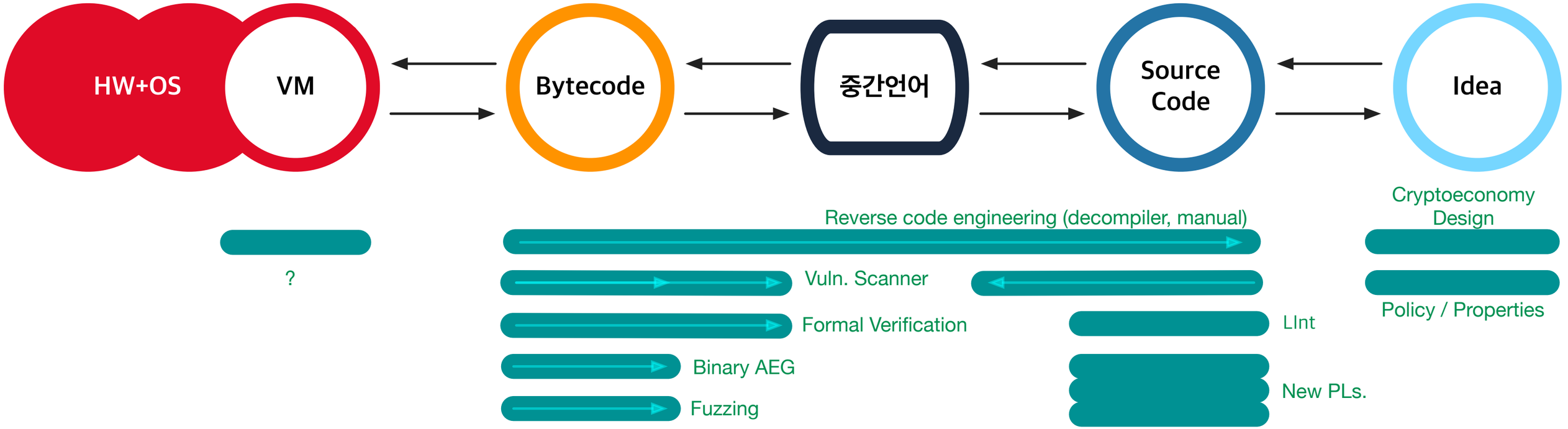
실제 문제는 여기에
있으나 너무 복잡하다.

Binary
분석의
마지노선

기존의 강자가
너무 쎄다.
(Parsing도 어렵다)

Mind
the gap!

Smart contract에 대한 현재 접근 방법



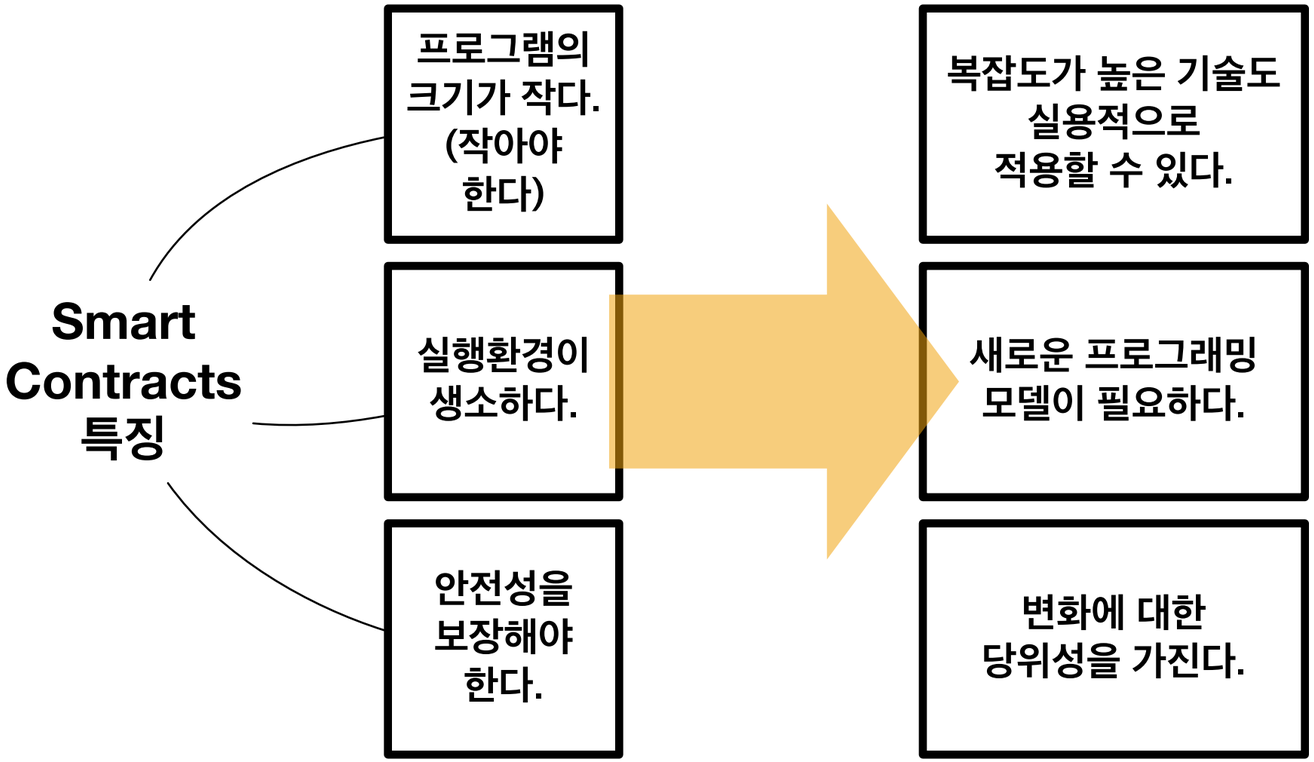
Binary Symbolic Exeuction!

얼마나 필요한가?

새로운 환경에 맞는 새로운 모델이 필요

Macro와 Micro적인 생각을 동시에 해야 한다.

기존 방식에서의 변화


Smart Contracts 특징
프로그램의 크기가 작다. (작아야 한다)
실행환경이 생소하다.
안전성을 보장해야 한다.
복잡도가 높은 기술도 실용적으로 적용할 수 있다.
새로운 프로그래밍 모델이 필요하다.
변화에 대한 당위성을 가진다.


“Symbolic execution”
“Formal verification”
“Model checking”
“Domain-specific …”

기존 방식의 연장선

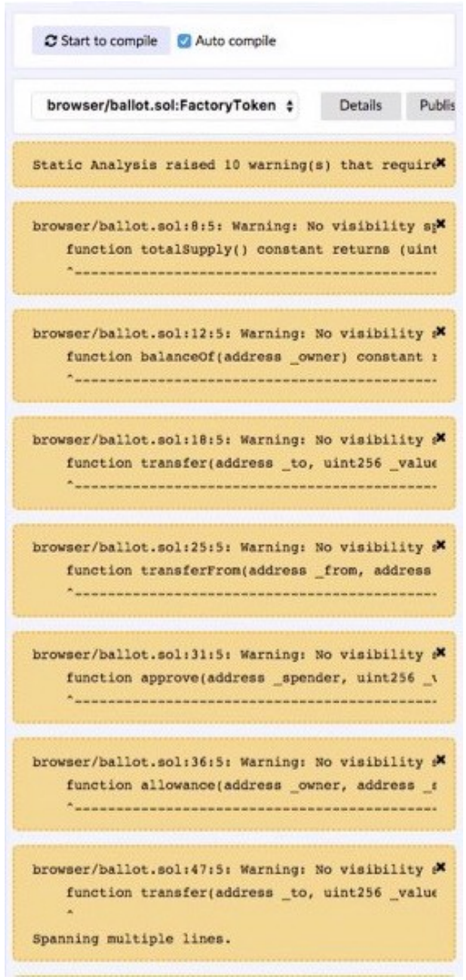
Code review (audit)

+

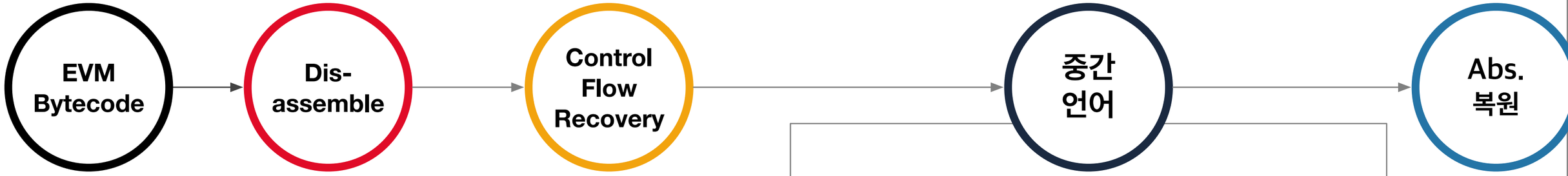
Linter (static analysis)

+

Reversing?

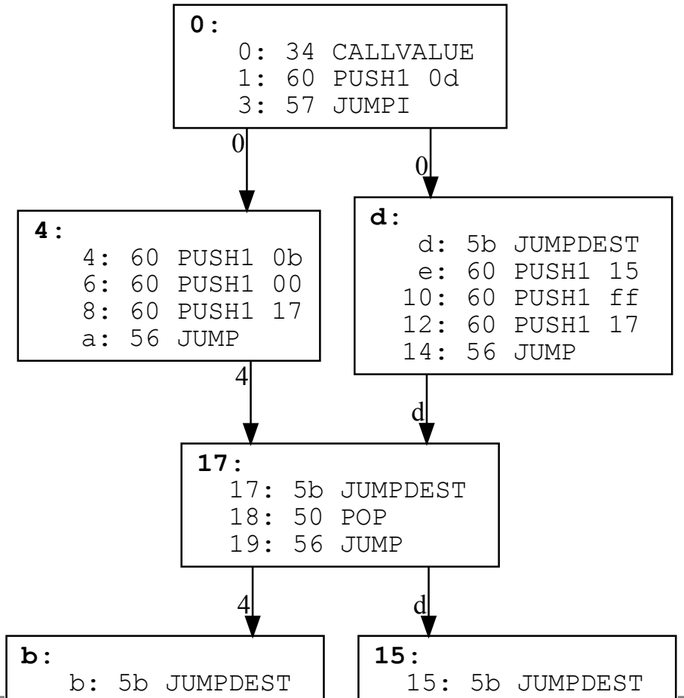


(자동화된) 분석의 시작



Linear sweep /
Recursive traversal

Heuristics /
Concrete execution /
Abstract interpretation



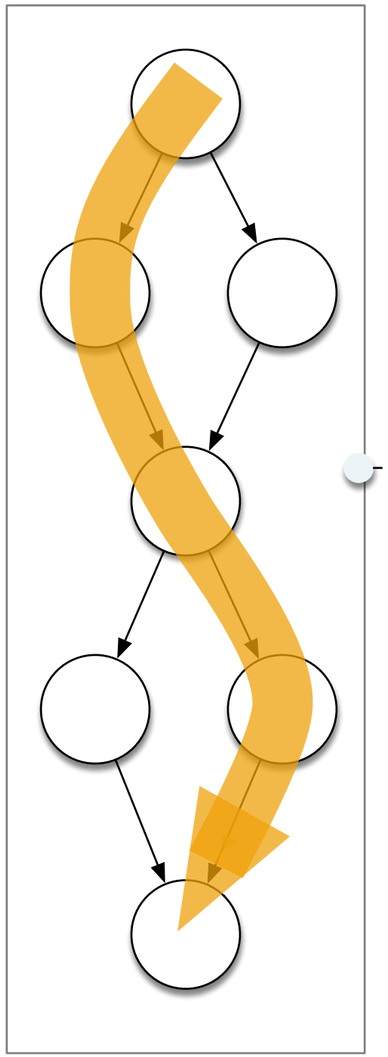

```
PUSH1 01
PUSH1 02
ADD
```

```
v1 = 01
v2 = 02
v3 = v1 + v2
```

```
v3 = 01+02
```

- Stack code의 동작을 explicit하게 보기 위하여 사용된다.
- 패턴 매칭을 고려해서 되도록 간단한 syntax로 정의한다.
- 일반적으로는 3 address code 이지만, 필요에 따라 nesting을 적용할 수 있다. (Multi-level IR)

Vulnerabilities Scanners



CFG는 정확한가?

Symbolic Execution

(State (Variables) ∧ Path predicate ∧ Vuln. Patterns) → Solver → SAT? UNSAT?

얼마나 잘 반영하는가?
모델과 분리되어 있는가?

e.g., Oyente
Mythril
Manticore
…
Zeus (Symbolic model checking)

e.g., Interger over/underflow

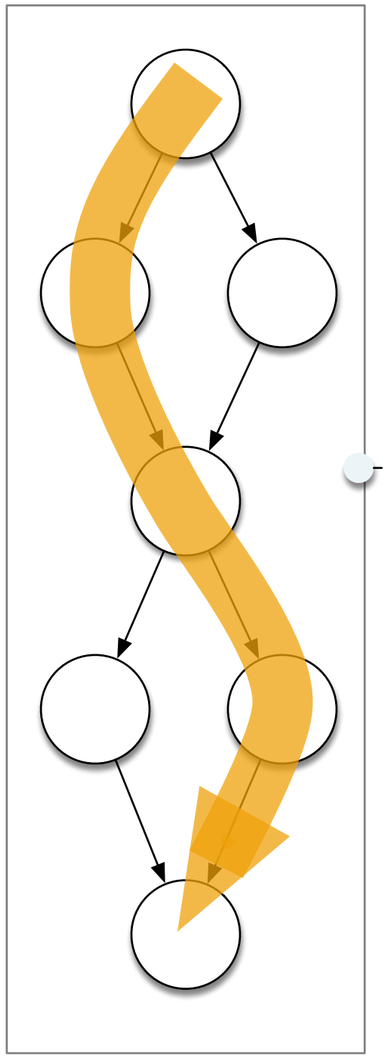
- SUB ∧ (OP1) < (OP2)
- ADD/MUL ∧ (OP1) + (OP2) > 2^{32} -1

Bug Type	Benchmark	MythrilPip 0.17.12	ManticoreGit 2018-05-18 18:01:09	OyentePip 0.2.7
Integer Overflow	minimal	True Positive	True Positive	False Negative
Integer Overflow	add	True Positive	True Positive	Unsupported
Integer Overflow	mul	True Positive	True Positive	Unsupported
Integer Overflow	path 1	True Negative	True Negative	Unsupported
Integer Overflow	benign 1	True Negative	False Positive	Unsupported
Integer Overflow	benign 2	False Positive	Unsupported	Unsupported
Integer Overflow	multi-tx 1	True Positive	False Negative	Unsupported
Integer Overflow	multi-tx 2	False Positive	Unsupported	Unsupported
Integer Overflow	multi-tx 3	True Positive	False Negative	Unsupported
Integer Overflow	storage inv	False Positive	True Negative	Unsupported
Integer Overflow	symbolic storage 1	True Positive	True Positive	Unsupported
Integer Overflow	symbolic storage 2	True Negative	True Negative	Unsupported
Integer Overflow	attribute store	False Positive	Analysis Failed	Unsupported
Integer Overflow	mapping string key	False Positive	Analysis Failed	Unsupported
Integer Overflow	fixed storage packing	True Negative	True Negative	Unsupported
	bytes			
Integer Overflow	parameter	False Positive	Analysis Failed	Unsupported
Integer Overflow	static array	True Negative	True Negative	Unsupported
Integer Overflow	mapping words	True Negative	True Negative	Unsupported
Integer Overflow	mapping structs 1	True Negative	True Negative	Unsupported
Integer Overflow	mapping structs 2	True Negative	False Positive	Unsupported
Integer Overflow	mapping static arr	True Negative	True Negative	Unsupported
Integer Overflow	dynamic array	False Positive	True Negative	Unsupported
Callback Effect-Free	dao	True Positive	False Negative	True Positive
Callback Effect-Free	dao fixed	False Positive	Unsupported	True Negative
Callback Effect-Free	effect-free	False Positive	Unsupported	True Negative
Assertion	minimal	True Positive	True Positive	True Positive
Assertion	constructor	False Negative	Analysis Failed	False Negative
Assertion	symbolic	True Positive	True Positive	True Positive
Assertion	require	True Negative	True Negative	True Negative
Assertion	multi tx 1	False Positive	Analysis Failed	False Positive
Assertion	multi tx 2	Unsupported	Analysis Failed	Unsupported
Eth Tx-Order Dependence	minimal 1	True Positive	False Negative	True Positive
Eth Tx-Order Dependence	minimal 2	False Positive	Unsupported	True Negative
Eth Tx-Order Dependence	multi tx 1	False Positive	Unsupported	False Positive
Eth Tx-Order Dependence	puzzle	True Positive	Analysis Failed	True Positive

https://consensys.net/diligence/evm-analyzer-benchmark-suite/

Automatic Exploit Generation

e.g., teEther



Symbolic Execution

(State (Variables) ∧ Path predicate ∧ Vuln. Patterns ∧ Exploit Constraints) → Solver (mod.)

Path stitching

Direct value transfer
Control hijacking

일종의 Harvard Architecture에서
Exploit 한다는 것은 무엇인가?
(Payload는 무엇인가?)

Transformation + Pattern/Policy

```
00: 60 04
02: 35 60
04: 08 56
06: 5B 00
08: 5B 60
0A: 00 56
0C: 60 00
0E: 55 56
    ⋮
```

EVM code

(1)

```
00: push 0x04
02: dataload
03: push 08
05: jump
06: jumpdest
07: stop
08: jumpdest
09: push 0x00
0B: sload
0C: push 0x00
0E: sstore
0F: jump
```

Parsed code

(2)

```
// entry
L1 a = 0x04
L2 b = dataload(a)
L3 ABI_9DA8(b)
L4 stop()

// method
   ABI_9DA8(b) {
L5  c = 0x00
    // write owner
L6  sstore(c, b);
   }
```

Decompiled code

(3)

```
MustFollow(L2, L1)
VarTag(a, const)
MayDepOn(b, dataload)
Eq(c, 0x00)
    ⋮
```

Semantic facts

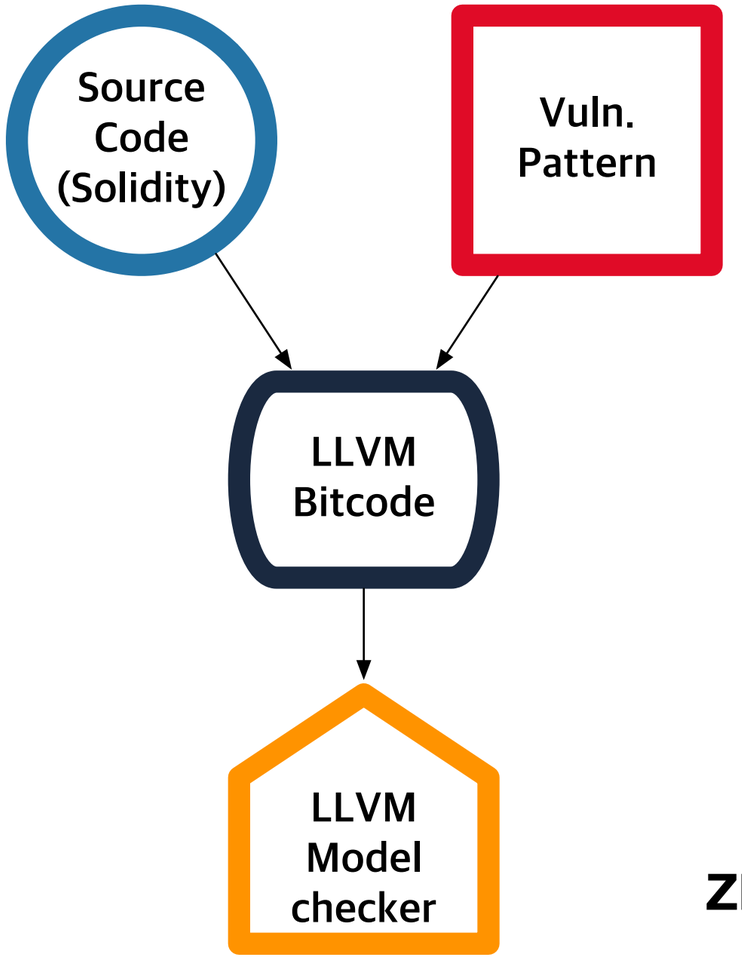
```
!MayDepOn(c, Caller)
&& !MayDepOn(6, Caller)
```

Restricted write violation pattern

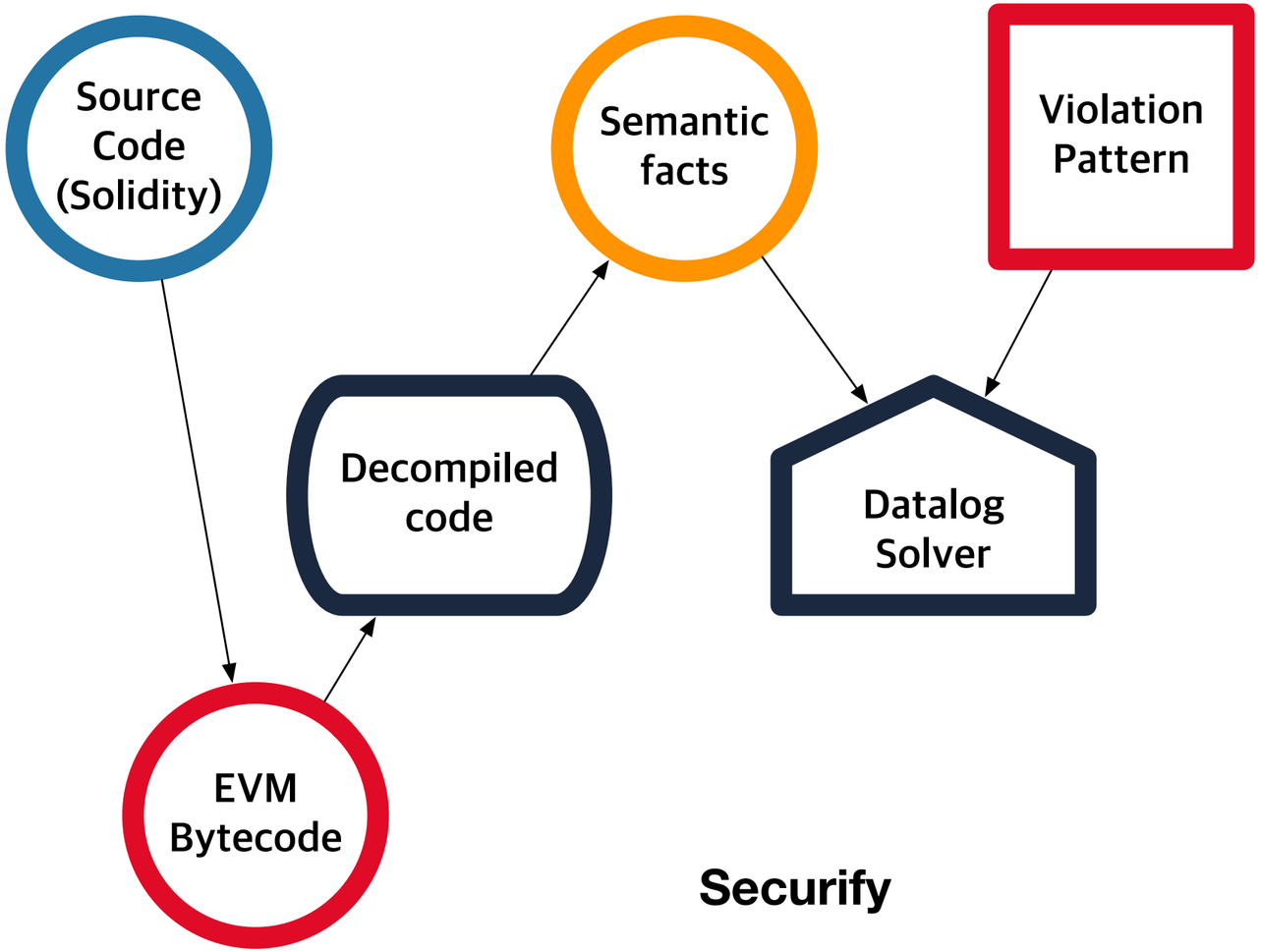
(4)

```
// entry
L1 a = 0x04
L2 b = dataload(a)
L3 ABI_9DA8(b)
L4 stop()

// method
   ABI_9DA8(b) {
L5  c = 0x00
    // write owner
L6  sstore(c, b);
   }
```

Matched pattern

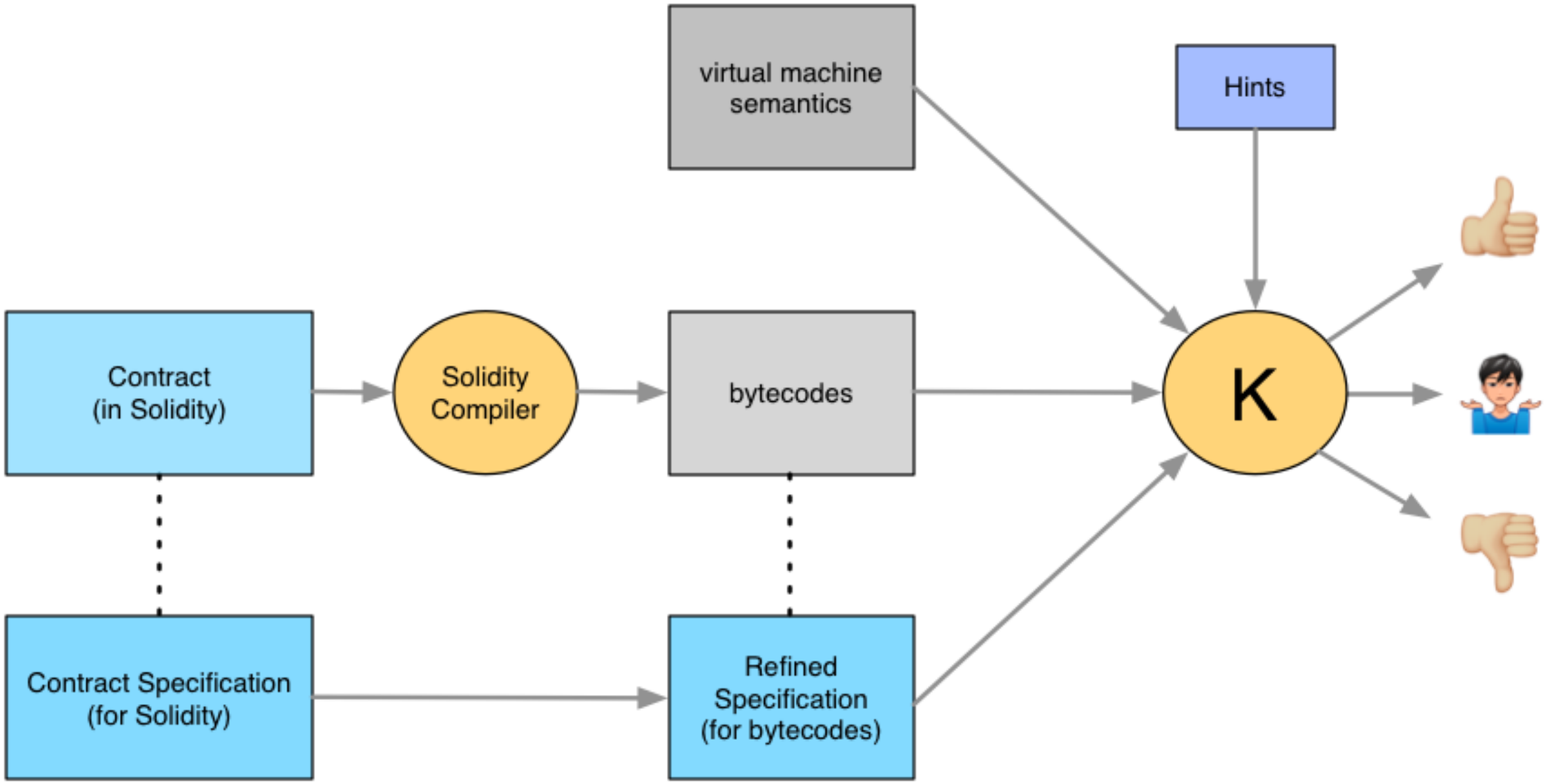


ZEUS



Securify

Formal Verfication



New Programming Languages

(From Vitalik Buterin's tweet)

"Mainstream 언어는 적합하지 않다." → Things contracts require that regular code does not:

* Very small code size
* Much higher focus on safety
* Much higher focus on auditability (misleading code very bad)
* Perfect determinism

C++ (EOS)

```
void add_balance( account_name payer, account_name to, uint64_t q ) {
   auto toitr = _accounts.find( to );
   if( toitr == _accounts.end() ) {
     _accounts.emplace( payer, [&]( auto& a ) {
        a.owner = to;
        a.balance = q;
     });
   } else {
     _accounts.modify( toitr, 0, [&]( auto& a ) {
        a.balance += q;
        eosio_assert( a.balance >= q, "overflow detected" );
     });
   }
}
```

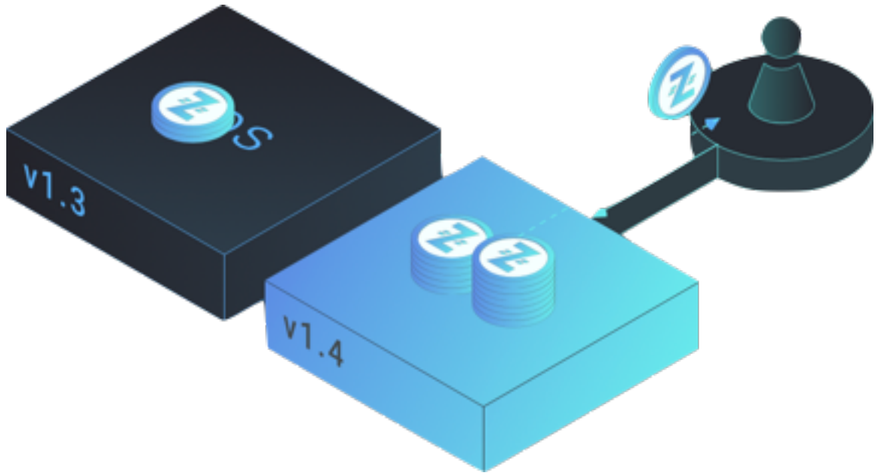
```
void transfer( account_name from, account_name to, uint64_t quantity ) {
   require_auth( from );

   const auto& fromacnt = _accounts.get( from );
   eosio_assert( fromacnt.balance >= quantity, "overdrawn balance" );
   _accounts.modify( fromacnt, from, [&]( auto& a ){ a.balance -= quantity; } );

   add_balance( from, to, quantity );
}
```

Vyper (Ethereum)

```
@public
def transfer(_to : address, _value : uint256(wei)) -> bool:
    _sender: address = msg.sender
    # Make sure sufficient funds are present implicitly through overflow protection
    self.balances[_sender] = self.balances[_sender] - _value
    self.balances[_to] = self.balances[_to] + _value
    # Fire transfer event
    log.Transfer(_sender, _to, _value)
    return True
```

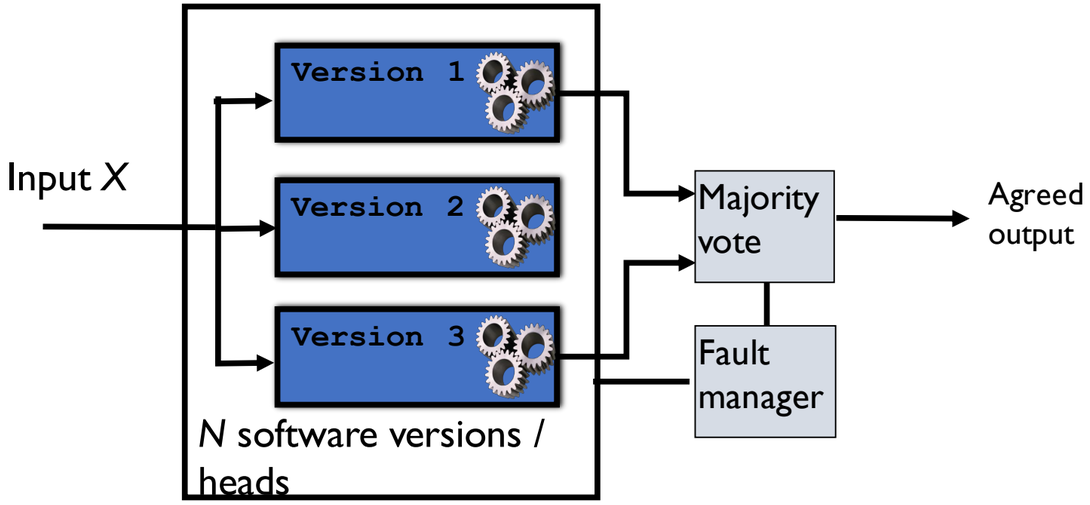
Bamboo, Babbage, Liquidity,
Michelson, OWL, Plutus
Rholang, Scilla, Simplicity
Solidity, Typecoin, Vyper
…

Crypto-economic Solution



ZepplinOS

(Up-to-date libraries)

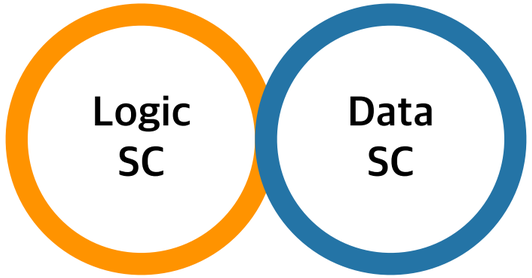


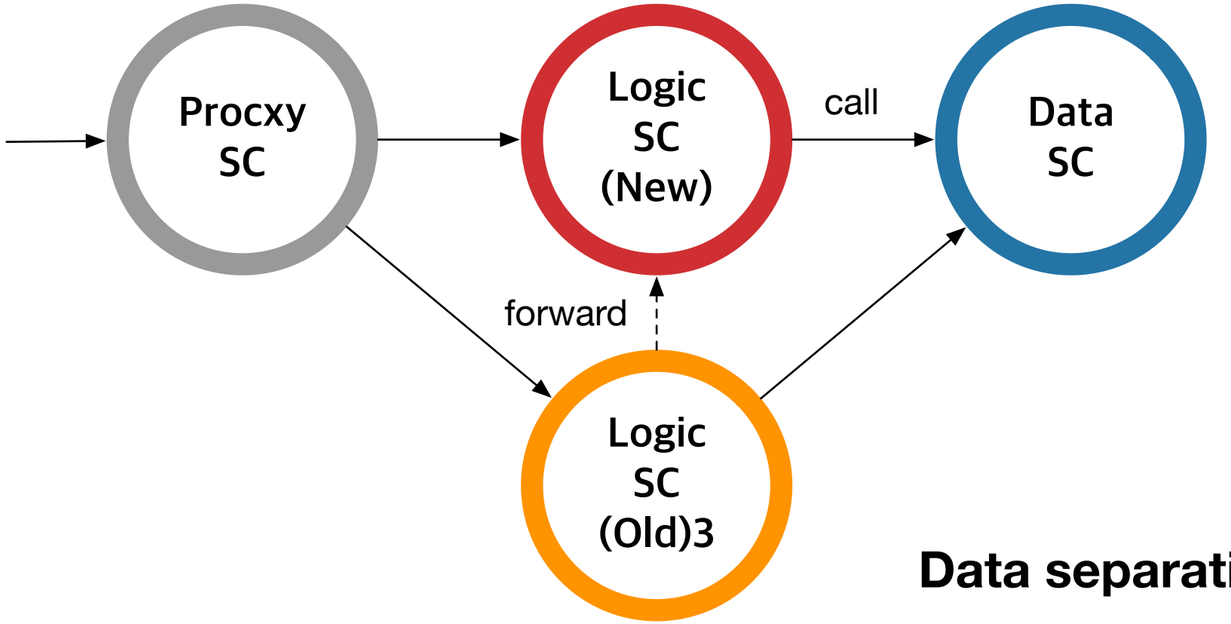
Hydra Framework

(Principled bug bounty)

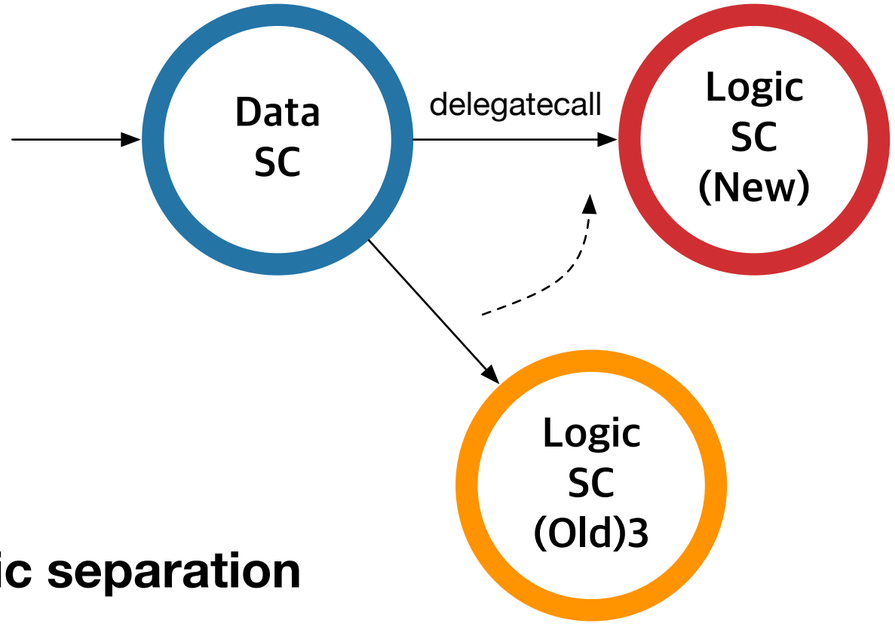
Updating smart contracts

Logic과 data의 분리





Data separation



Logic separation

감사합니다.

jonghyup@gmail.com