Smart Contract 분석과 PL

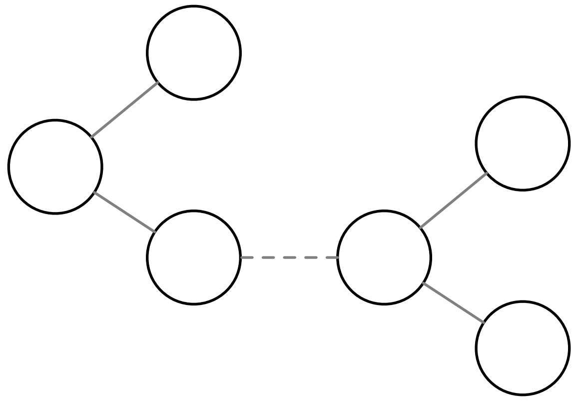
2018-08-20
SIGPL

이종협

Blockchain intro

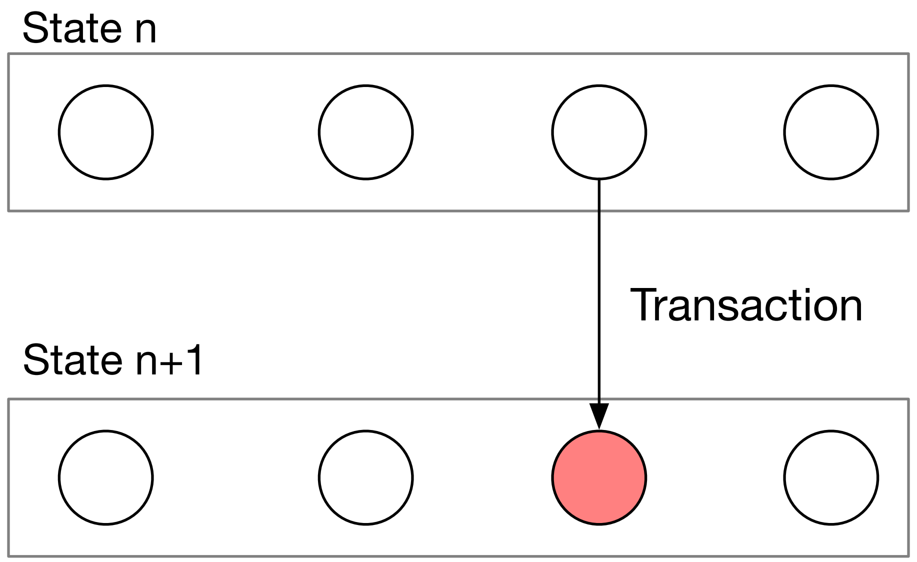
Bitcoin

Transaction Model



Ethereum

State + account model



+ EVM
(Ethereum Virtual Machine)

Hyperledger

Framework

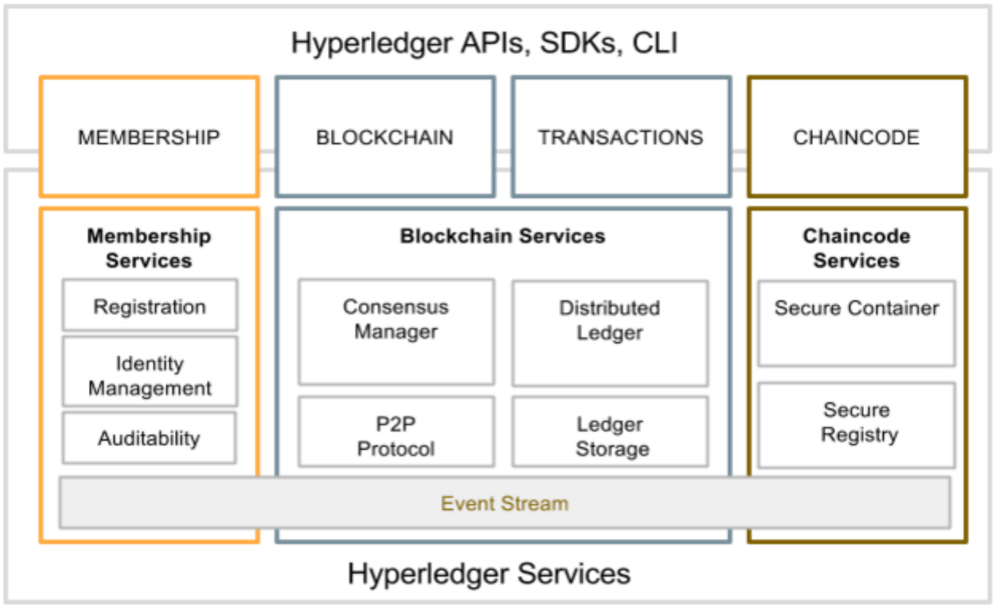


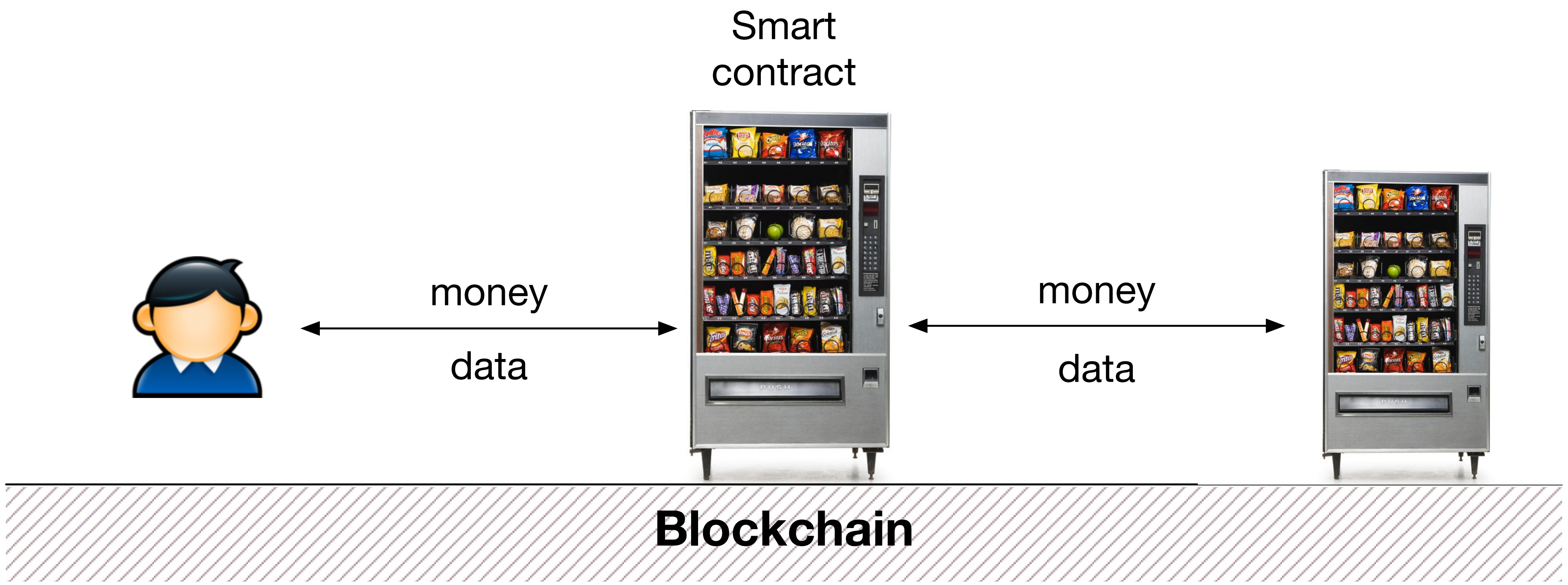
Figure 2: Hyperledger reference architecture

Smart contract

“Contract를 구현하고, 강제하고, 실행시켜 주는 code”

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

⋮



Solidity code

```
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 {
        ...
    }
}
```

Storage: the `mapping` declaration

Constructor: the `MyToken` function

Function (Public): the `transfer` function

Fallback function: the `function () payable` block

Smart contracts

어떻게 볼 것인가?

Vending machine

Balance (Money!)

Storage

Distributed objects

≀|

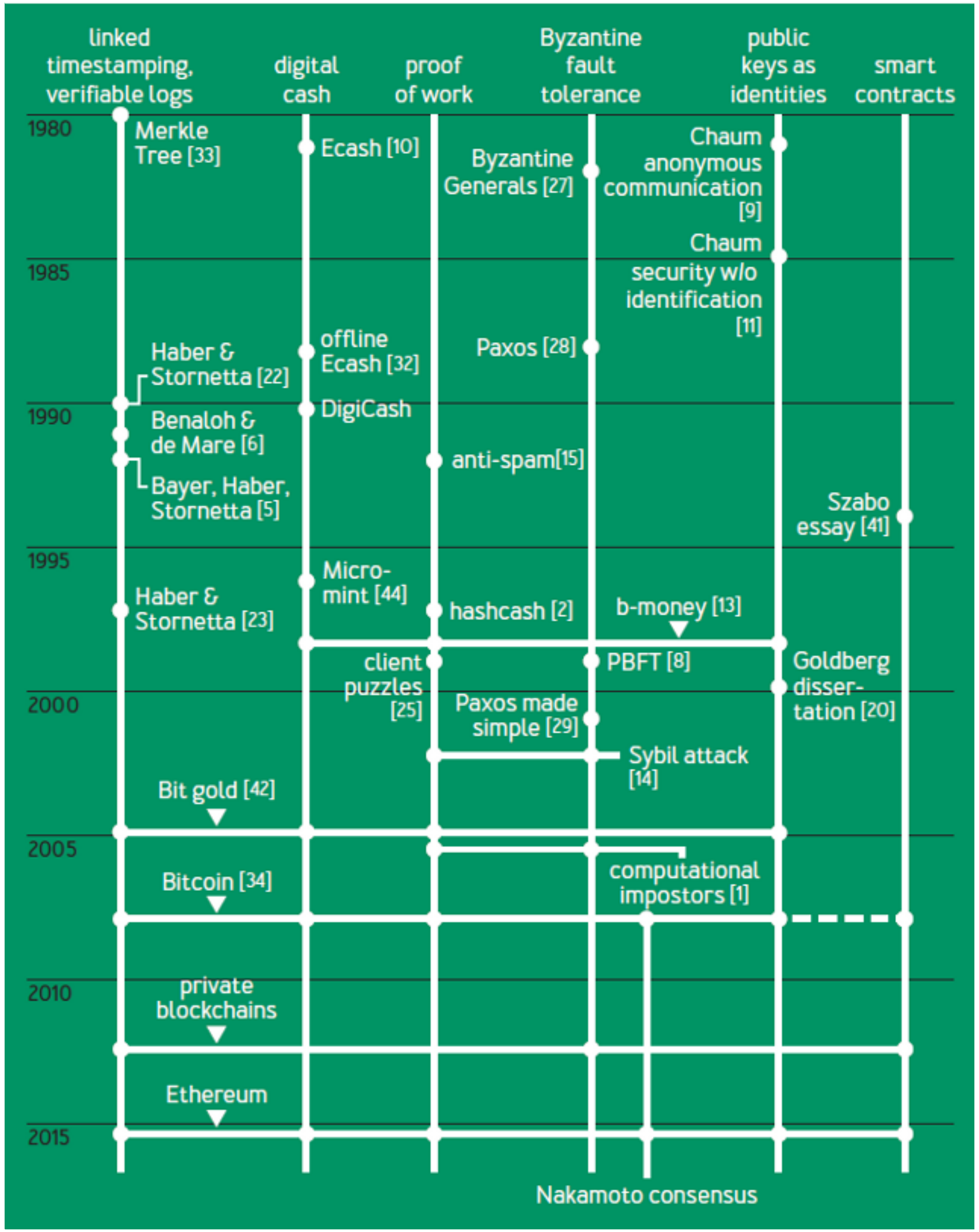
Threads using concurrent objects in shared memory

Secure execution (External)

!

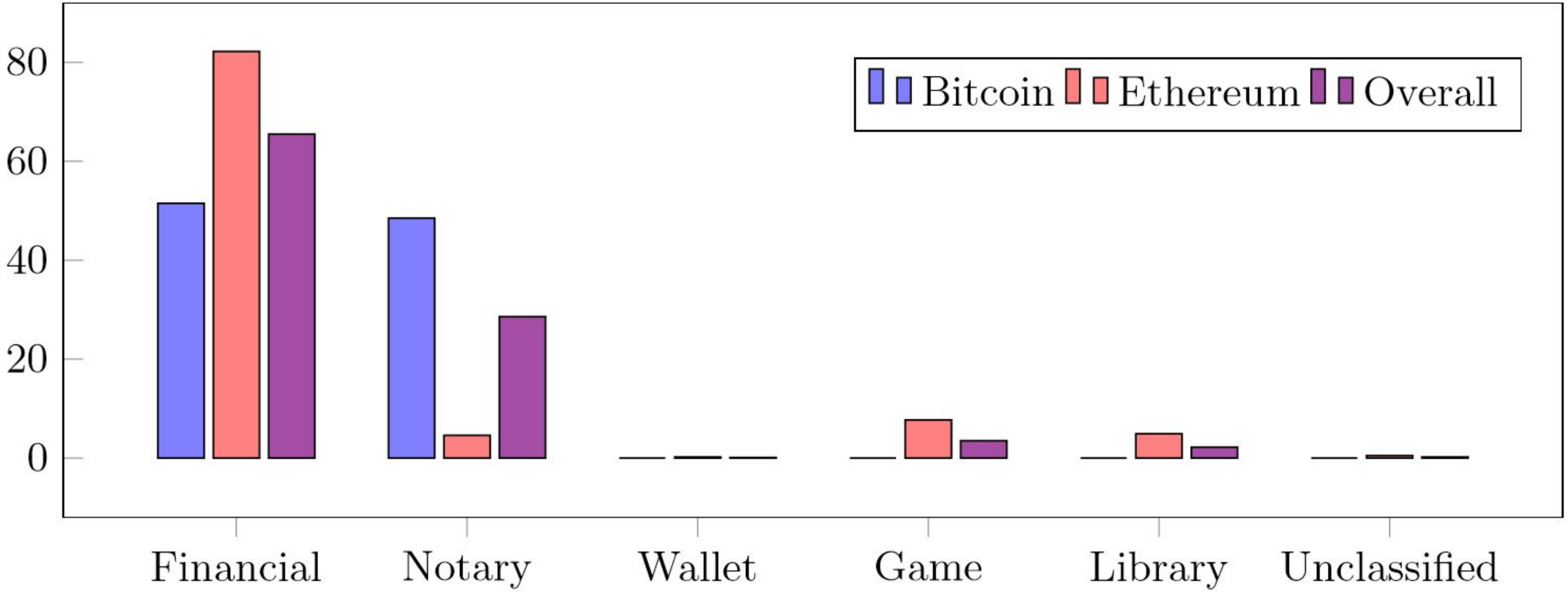
어떠한 의미를 가지는가?

Academic Pedigree



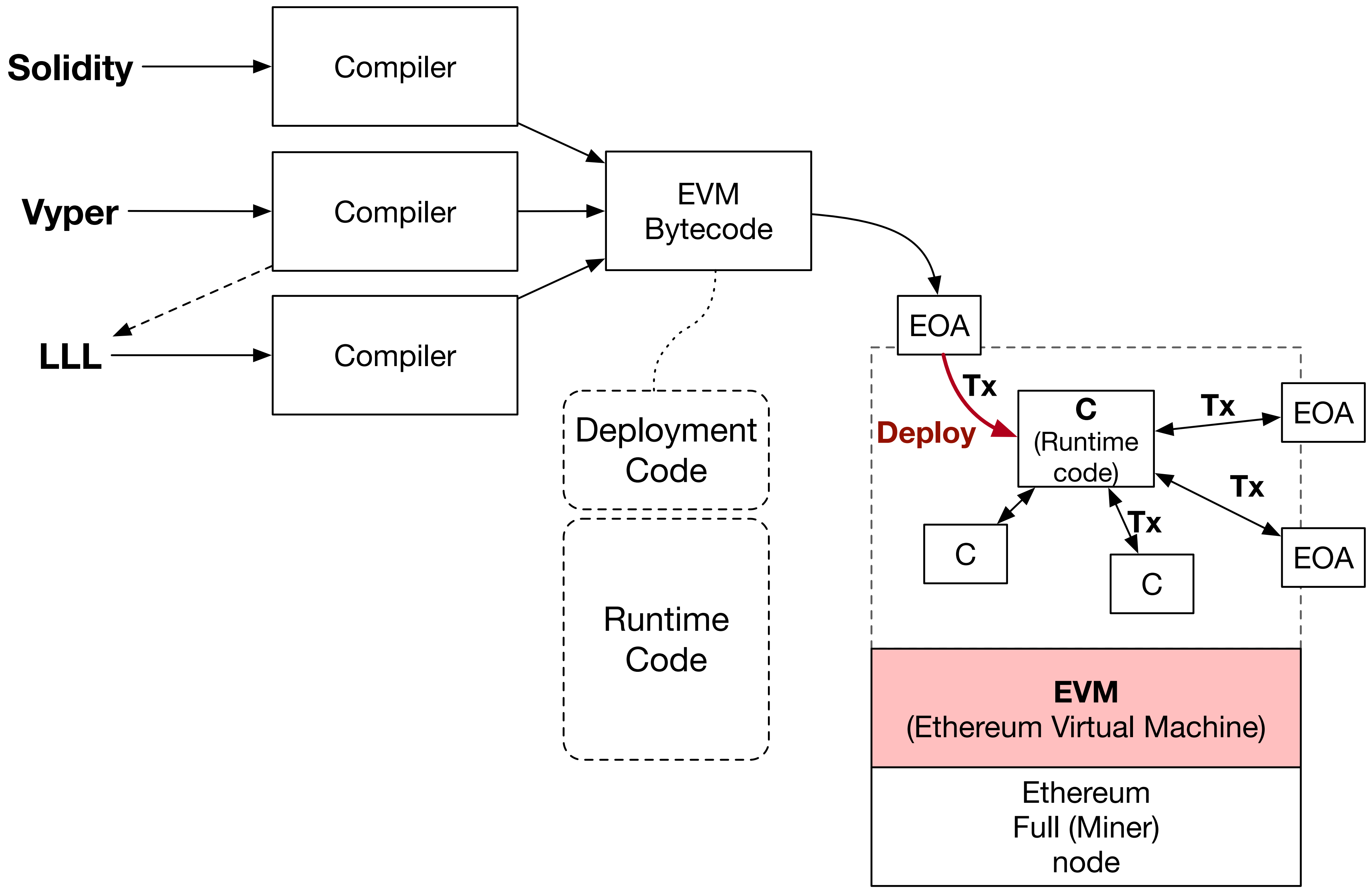
from “Bitcoin’s academic pedigree” Narayanan *et al.*

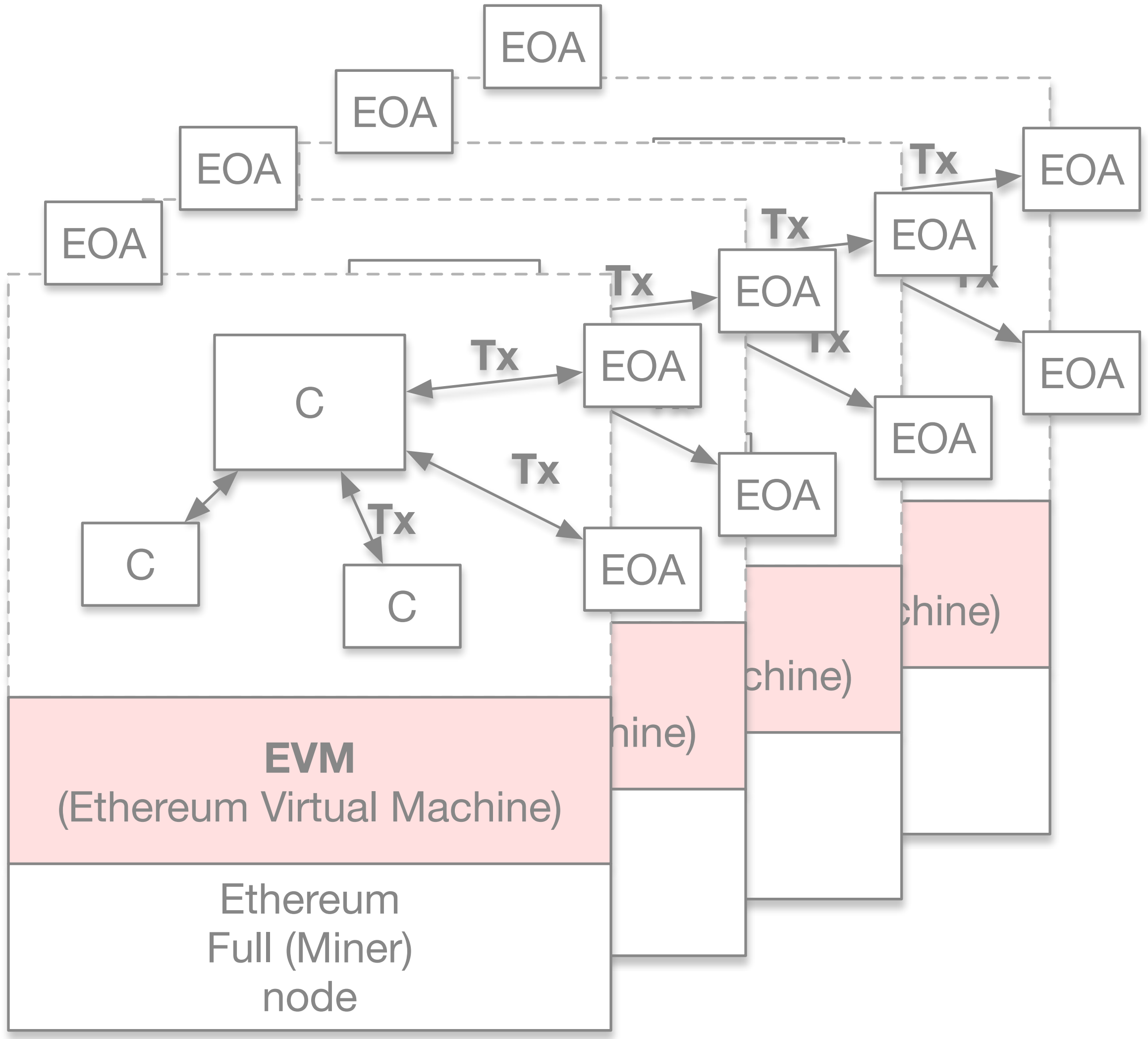
Smart contracts - category



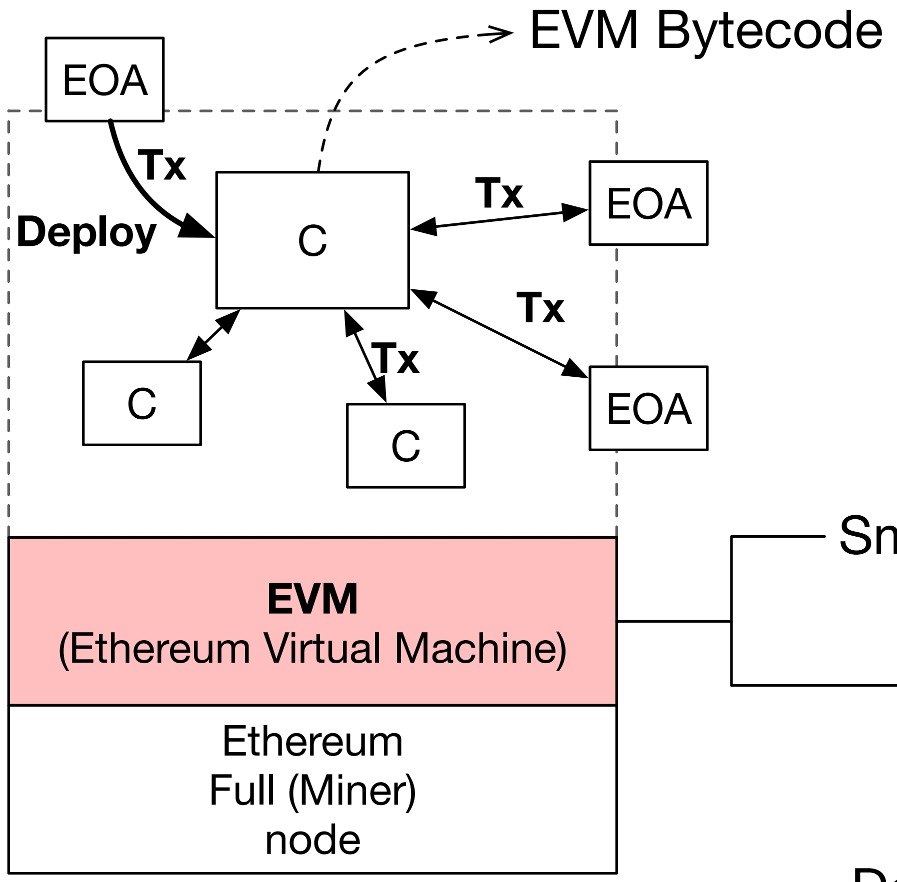
from “an empirical analysis of smart contracts” Bartoletti *et al.*

Smart contract lifecycle



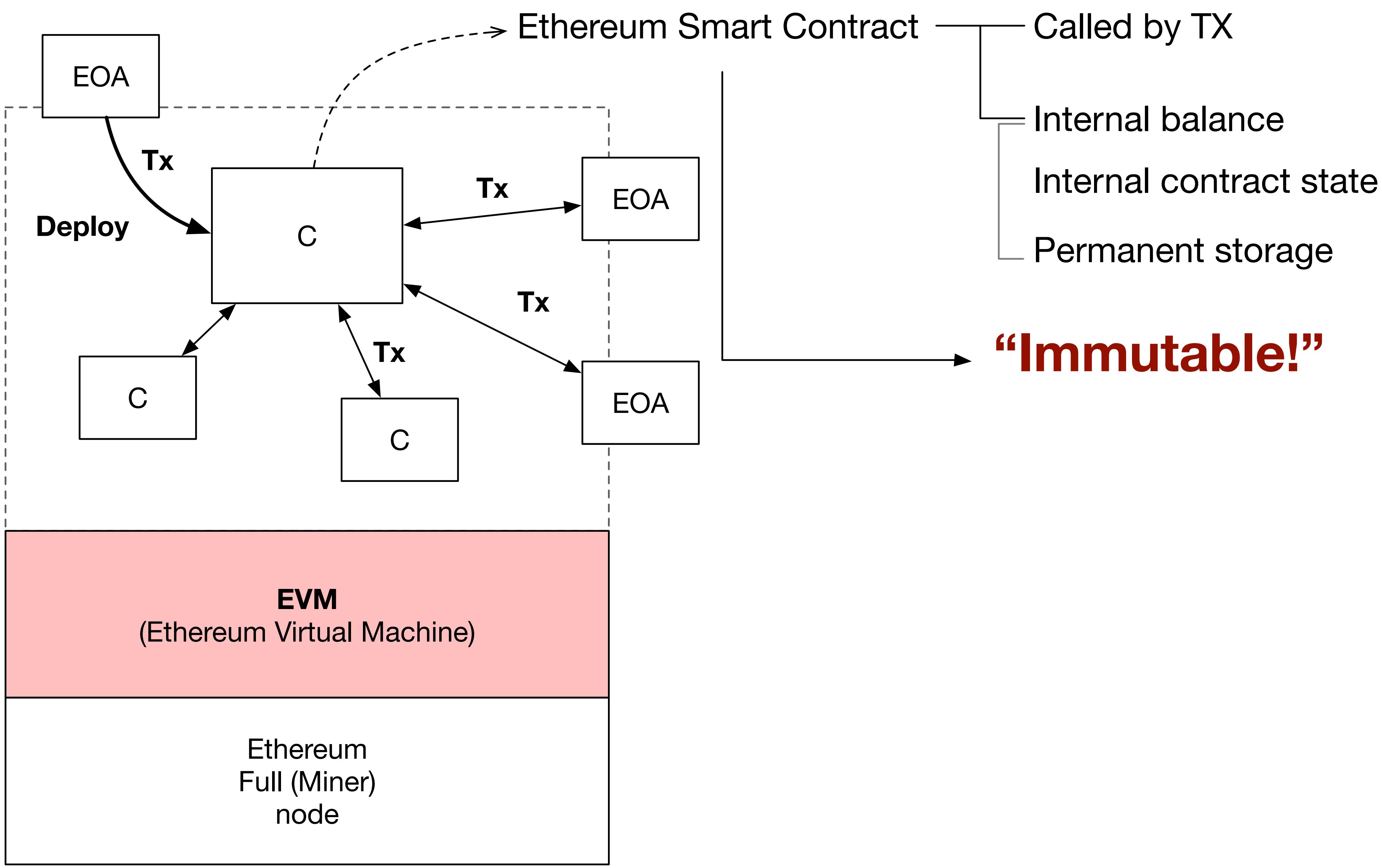


Ethereum Virtual Machine

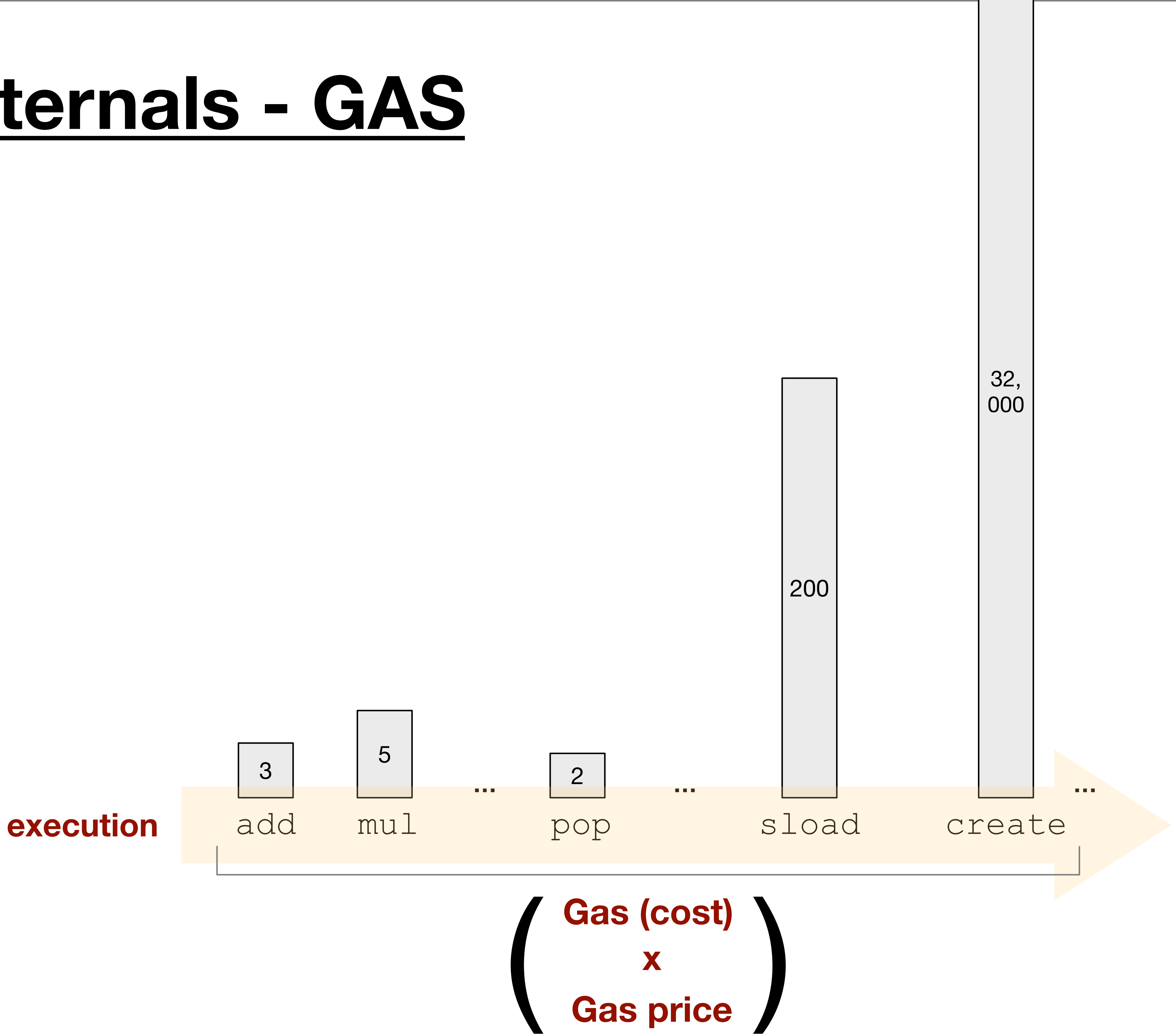


- Smart contract를 위한 execution model
  - Redundantly parallel
  - Turing complete!
  - GAS
- Design Goal
  - Simplicity
  - Space efficiency
  - Determinism?
  - Specialization
  - Security

Ethereum Virtual Machine



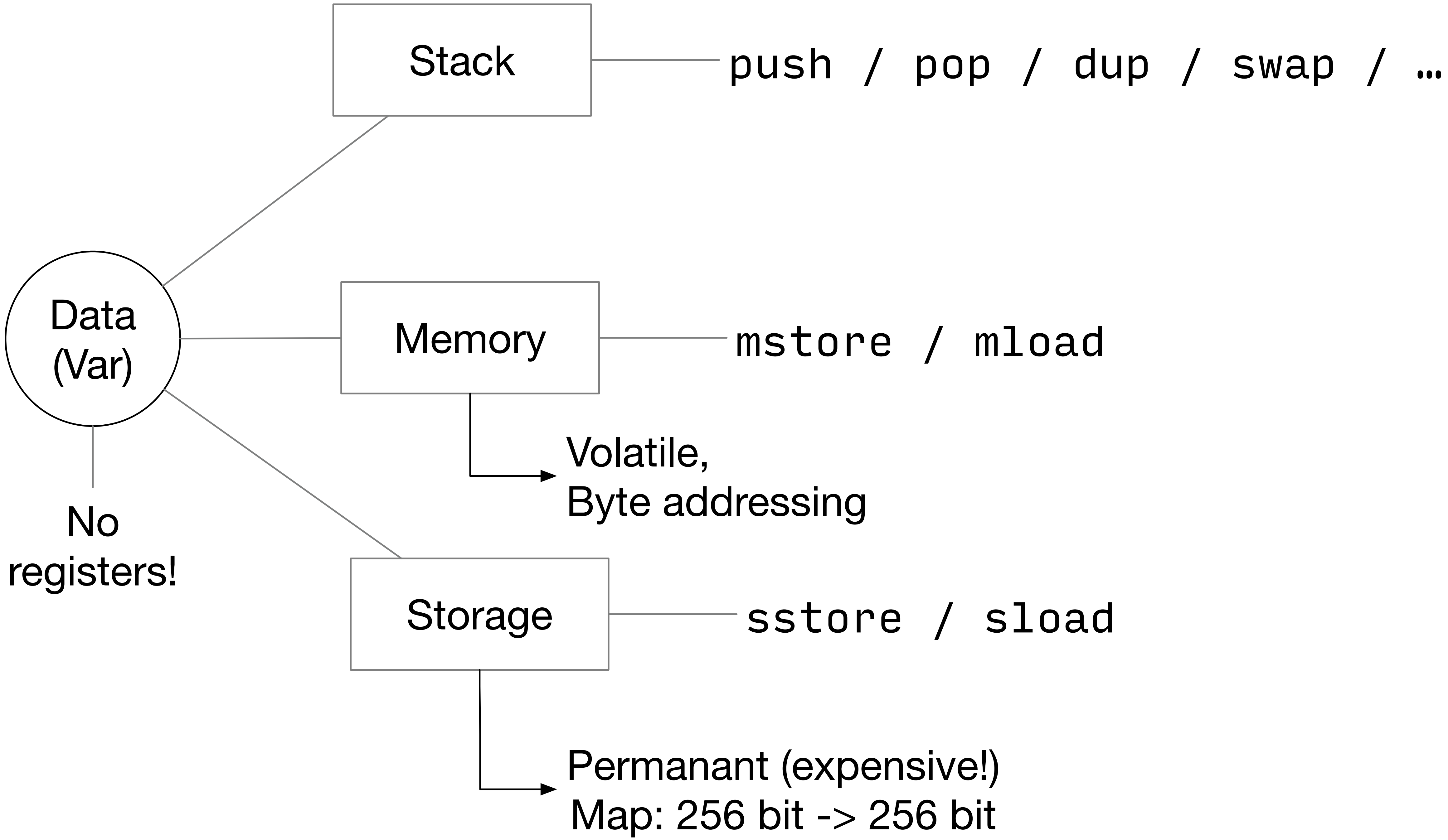
EVM internals - GAS



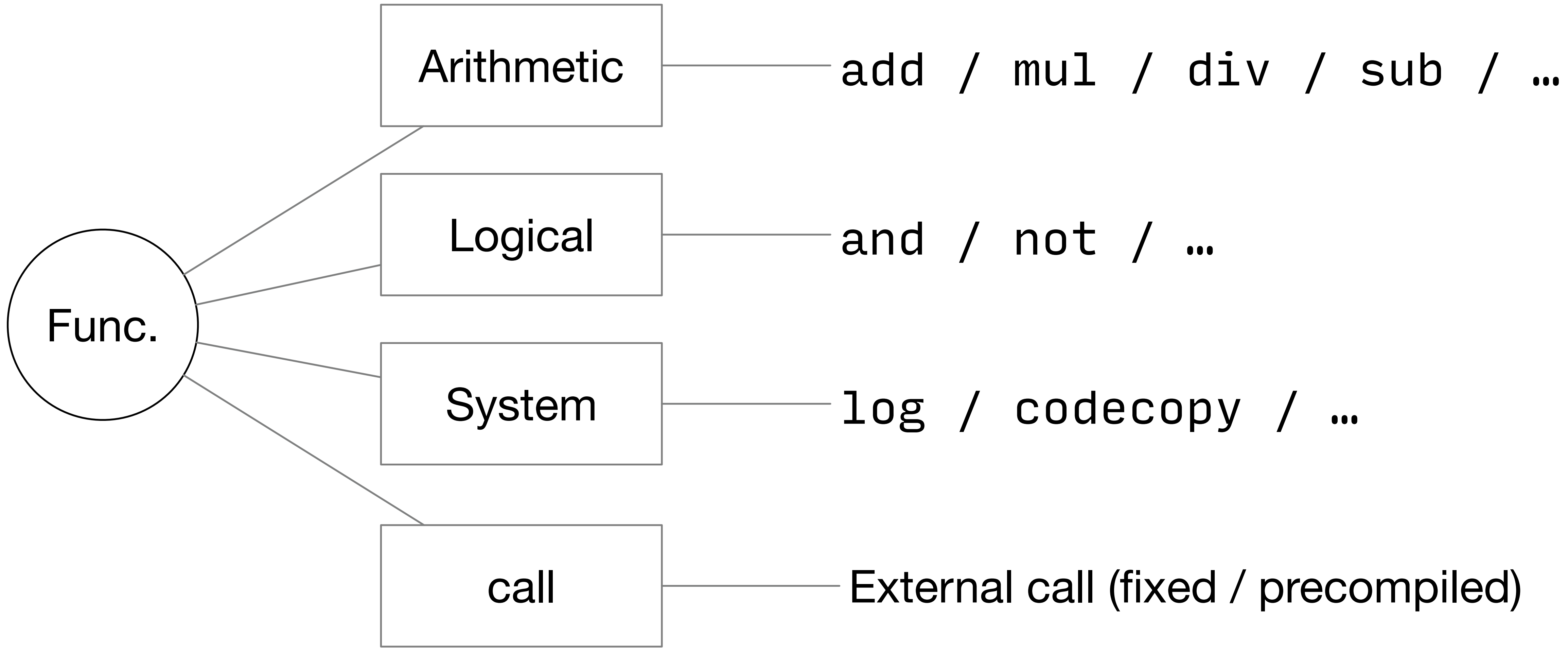
EVM assembly code

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

EVM internals - data

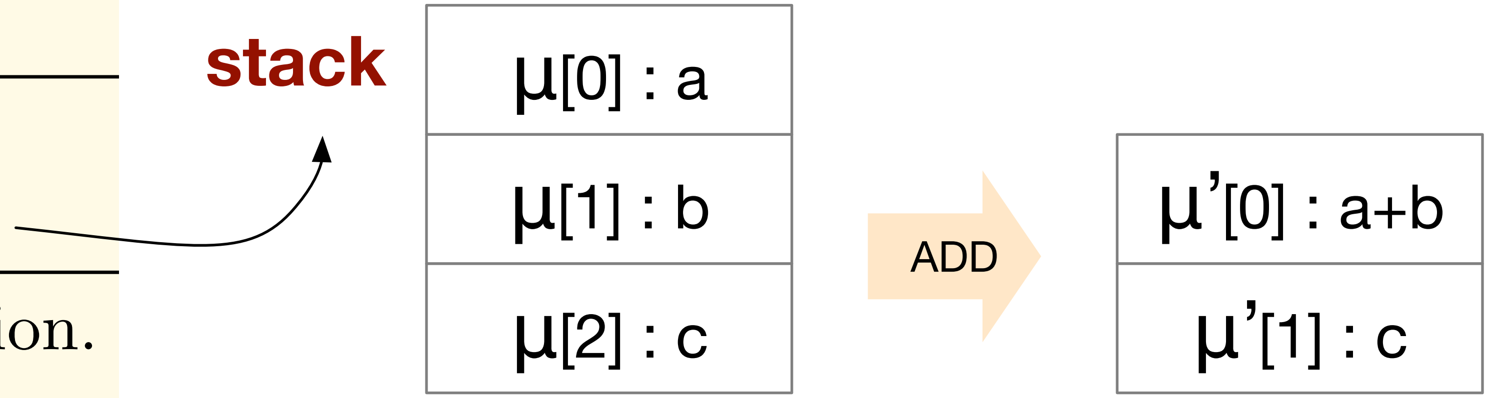


EVM internals - data



EVM instructions - “Yellow paper”

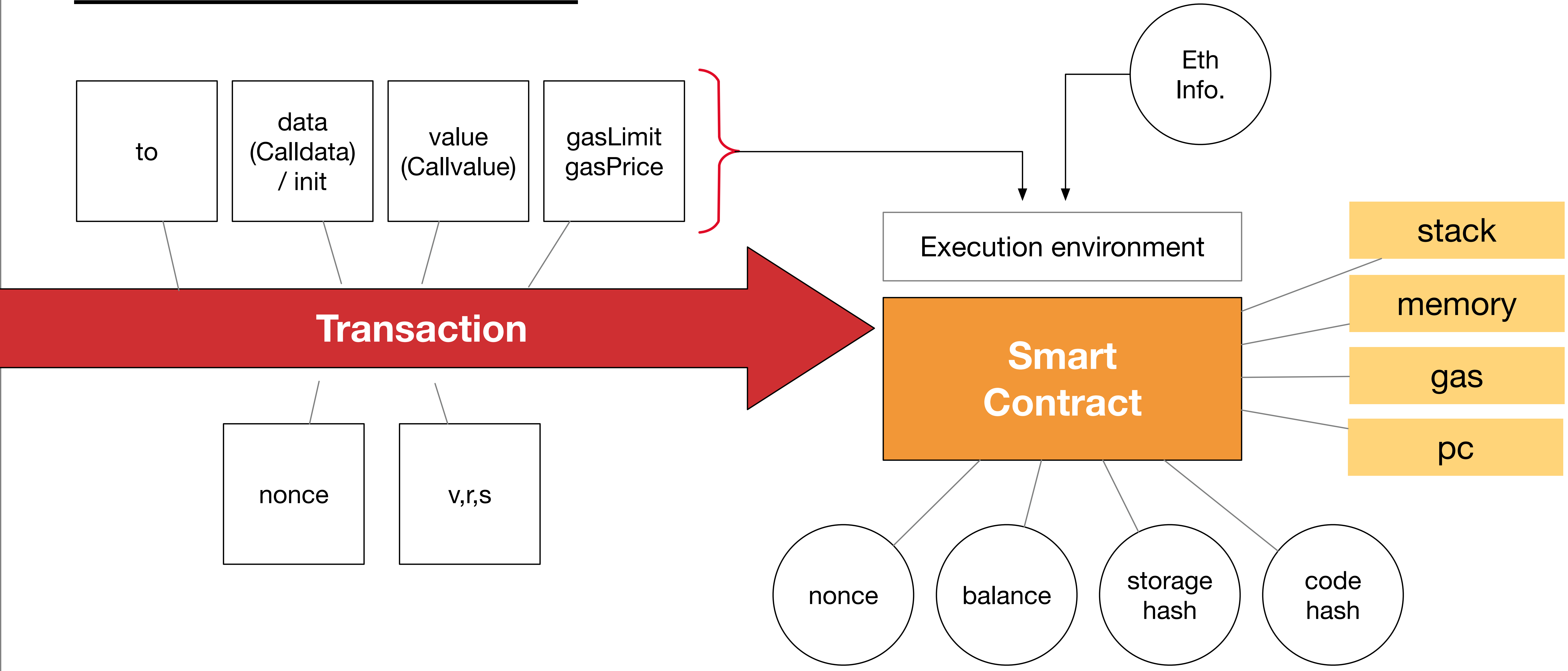
Value	Mnemonic	(pop) In δ	(push) out α	Description
0x00	STOP	0	0	Halts execution.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x02	MUL	2	1	Multiplication operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv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



μ: Machine state

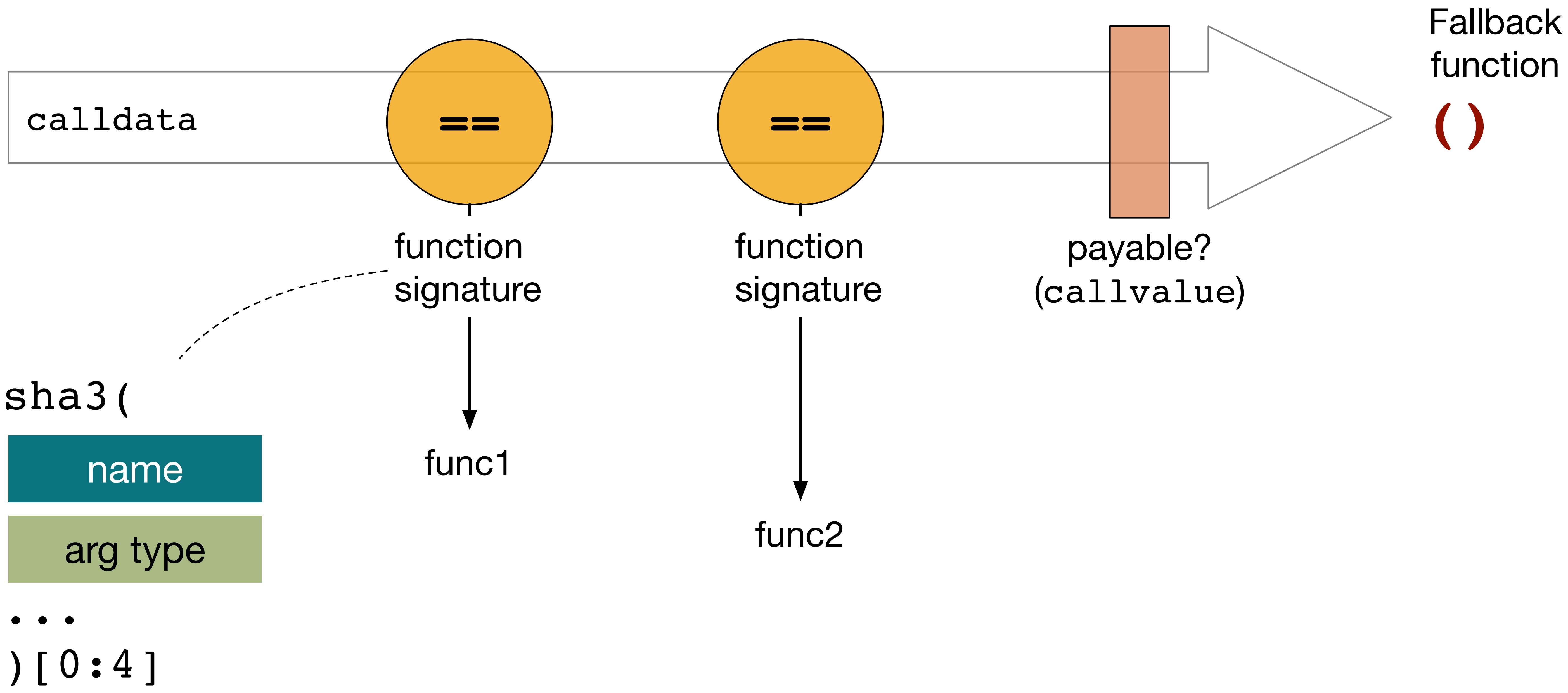
σ: World state

Execution model

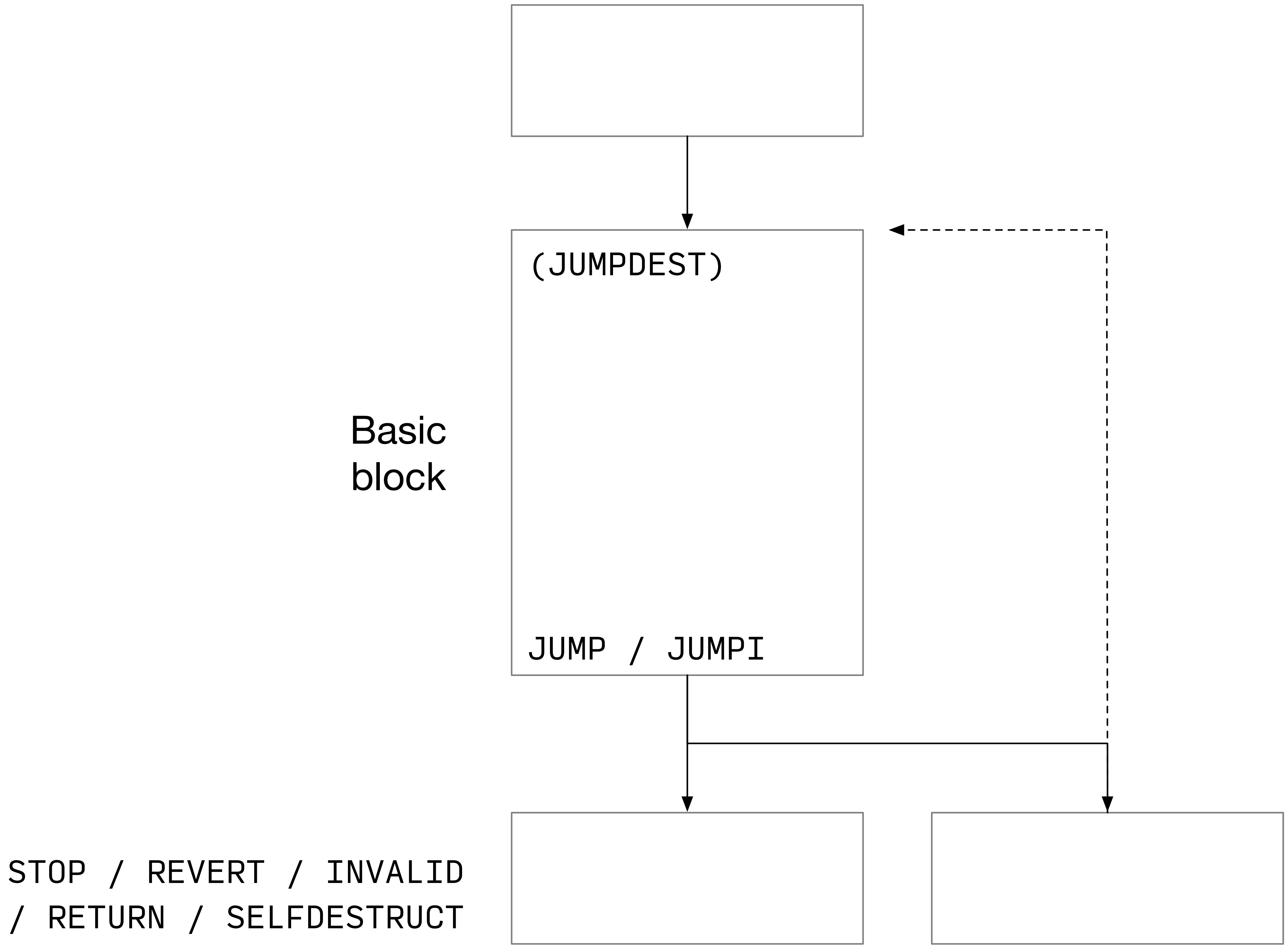


Function call handling

Function call과 fall back



EVM internals - control

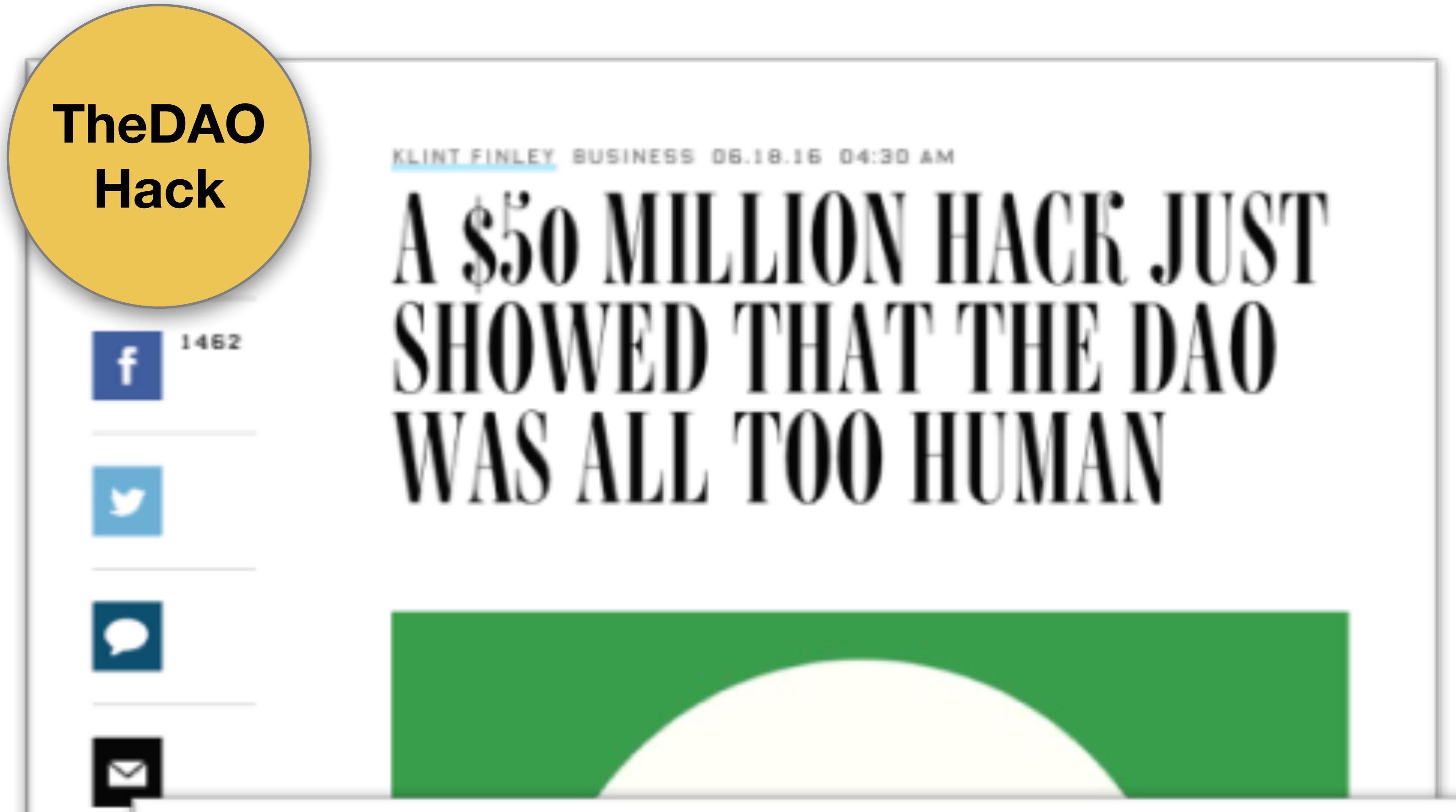


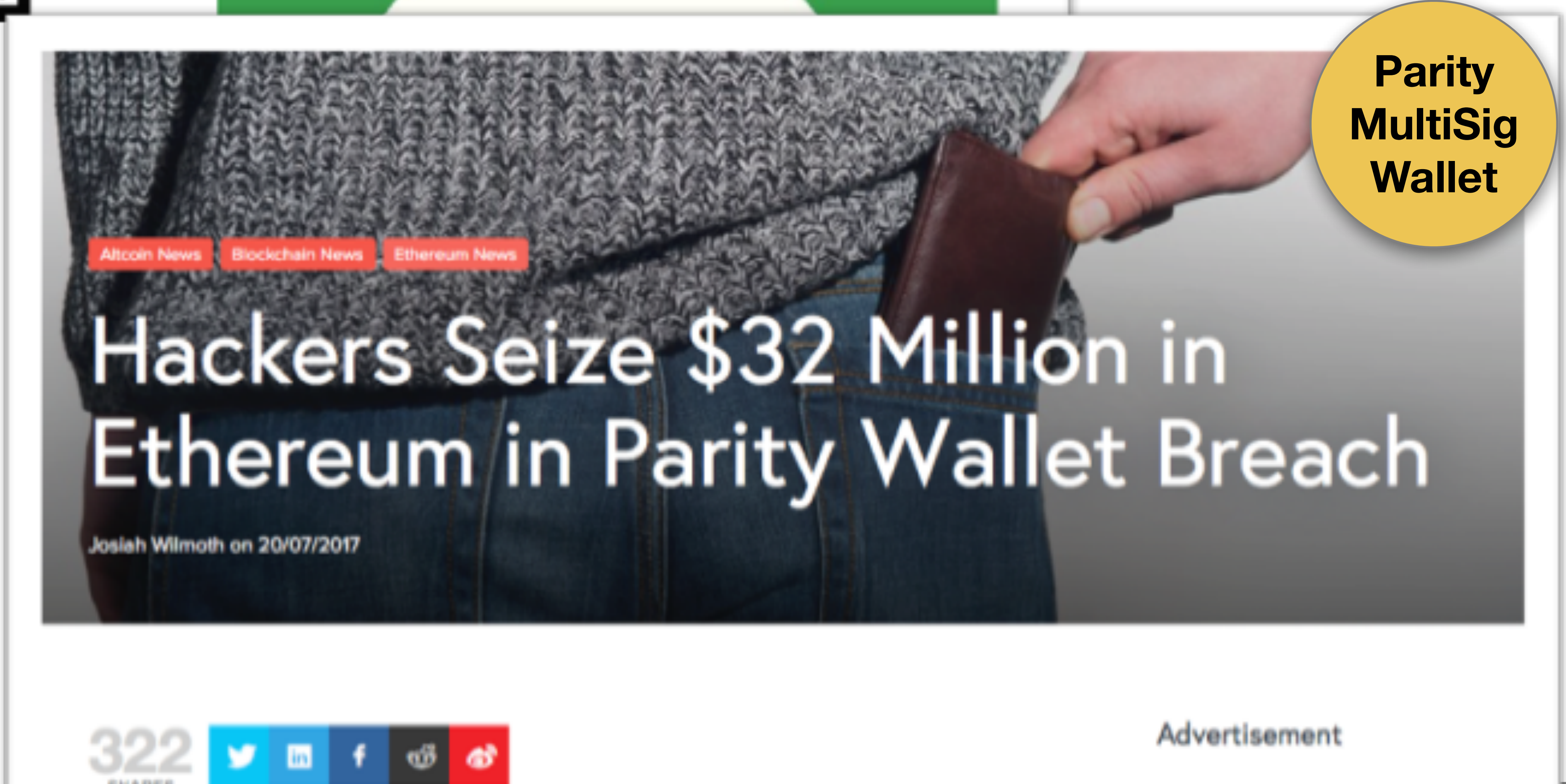
무엇이 문제인가?

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?

```
contract Wallet {
  mapping(address => uint) private userBalances;
  function withdrawBalance() {
    uint amountToWithdraw = userBalances[msg.sender];
    if (amountToWithdraw > 0) {
      msg.sender.call(userBalances[msg.sender]);
      userBalances[msg.sender] = 0;
    }
  }
  ...
}
```

```
contract AttackerContract {
  function () {
    Wallet wallet;
    wallet.withdrawBalance();
  }
}
```

Re-entrancy

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

Logic error

- prodigal SC
- suicidal SC
- greedy SC
- posthumous SC
- DoS (w/ deadlock)
- unprotected functions
- reentrancy

Undefined behaviors

- short address
- inconsistent view
- force transfer
- integer overflow
- DoS (w/ GAS)

EVM-level

- front running
- block state dep.

Unchecked send

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

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

Incorrect logic

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

Integer overflow

+

The Ethernaut:

https://ethernaut.zeppelin.solutions

어떻게 해결하는가?

무엇을 분석할 것인가?

어떻게 분석하는가?

왜 분석하는가?

지금 어디에 있는가?

근본적인 **고민** → 나름의 **문제** → 제대로 된 **해결책**

Dijkstra's <u>three golden rules</u> for successful scientific research

1. (…) Always try to work as closely as possible at the boundary of your abilities. Do this, because it is the only way of discovering how that boundary should be moved forward.

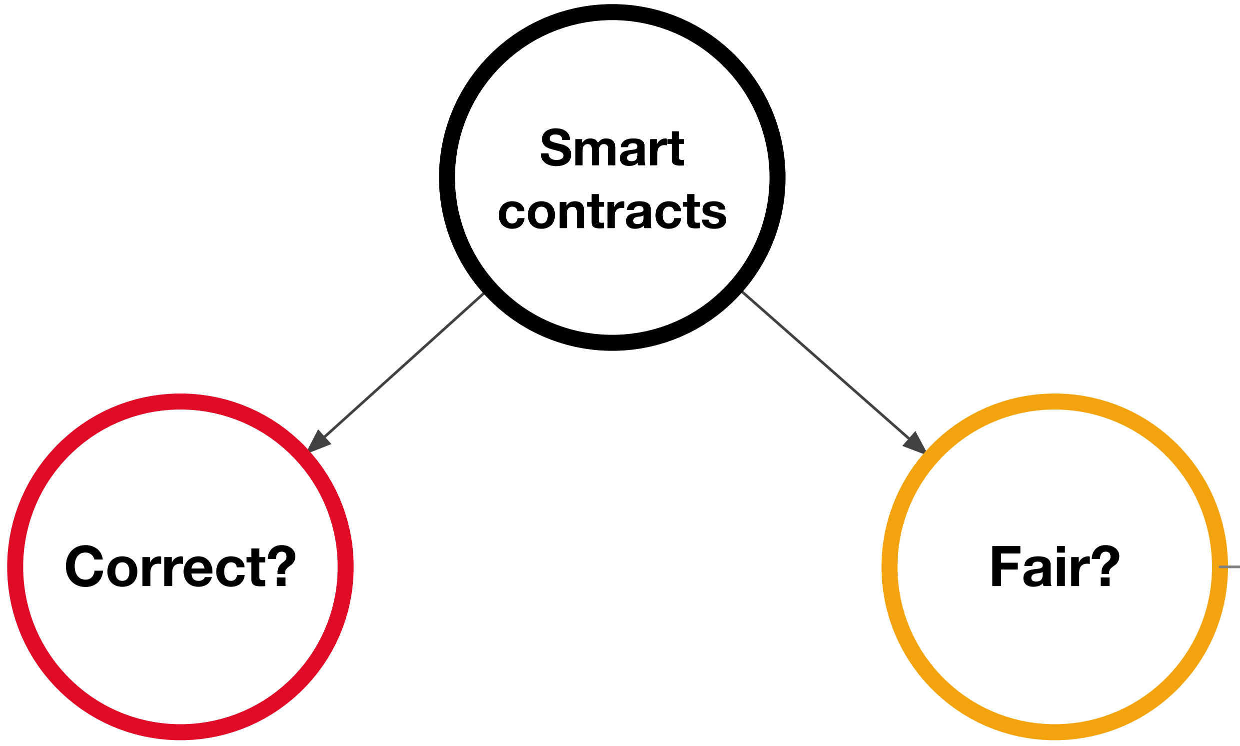
2. We all like our work to be socially relevant and scientifically sound. (…) If the two targets are in conflict with each other, let the requirement of scientific soundness prevail.

3. Never tackle a problem of which you can be pretty sure that it will be tackled by others who are, in relation to that problem, at least as competent and well-equipped as you.

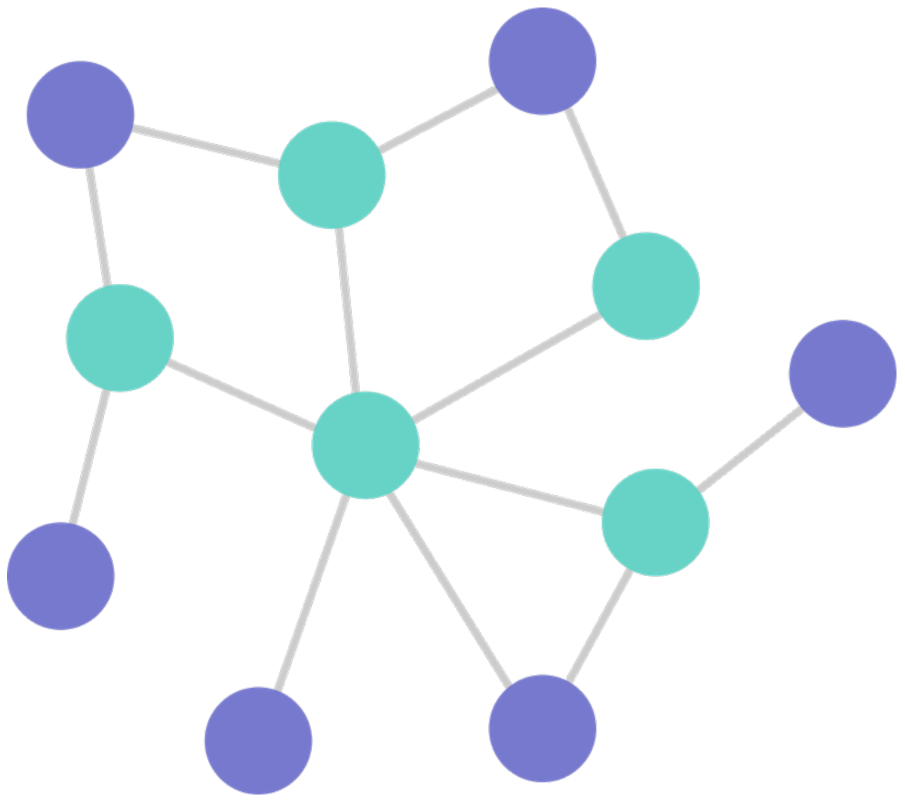
Blockchain에서 Smart contract란 어떤 의미인가?

Smart contract의 안전성이란?

Correctness와 fairness의 기준은 무엇인가?



무엇에
대하여?



Token economy

Decentralized governance 구조와
Incentive mechanism으로 정의

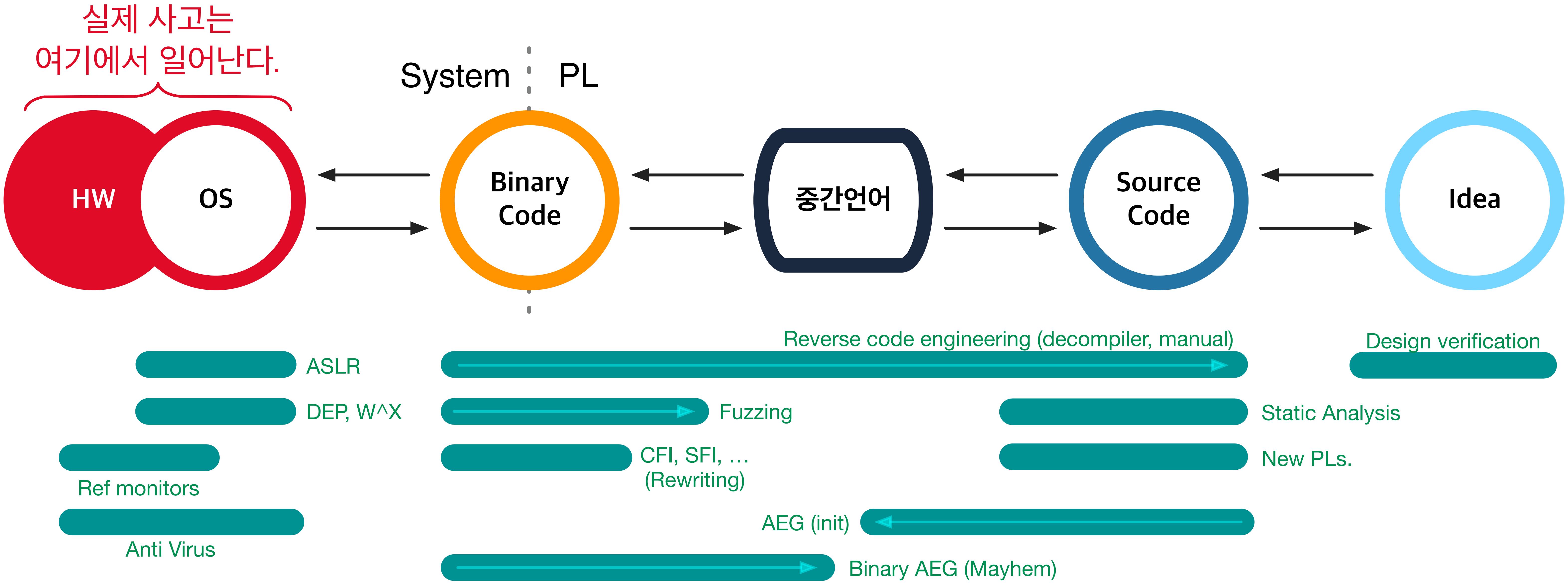
Smart contract가 이것을 위배하는가?

접근 방법의 변화

Smart Contracts 특징

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

Software security에서의 (기존) 접근 방법



반복되는
창과 방패의 싸움

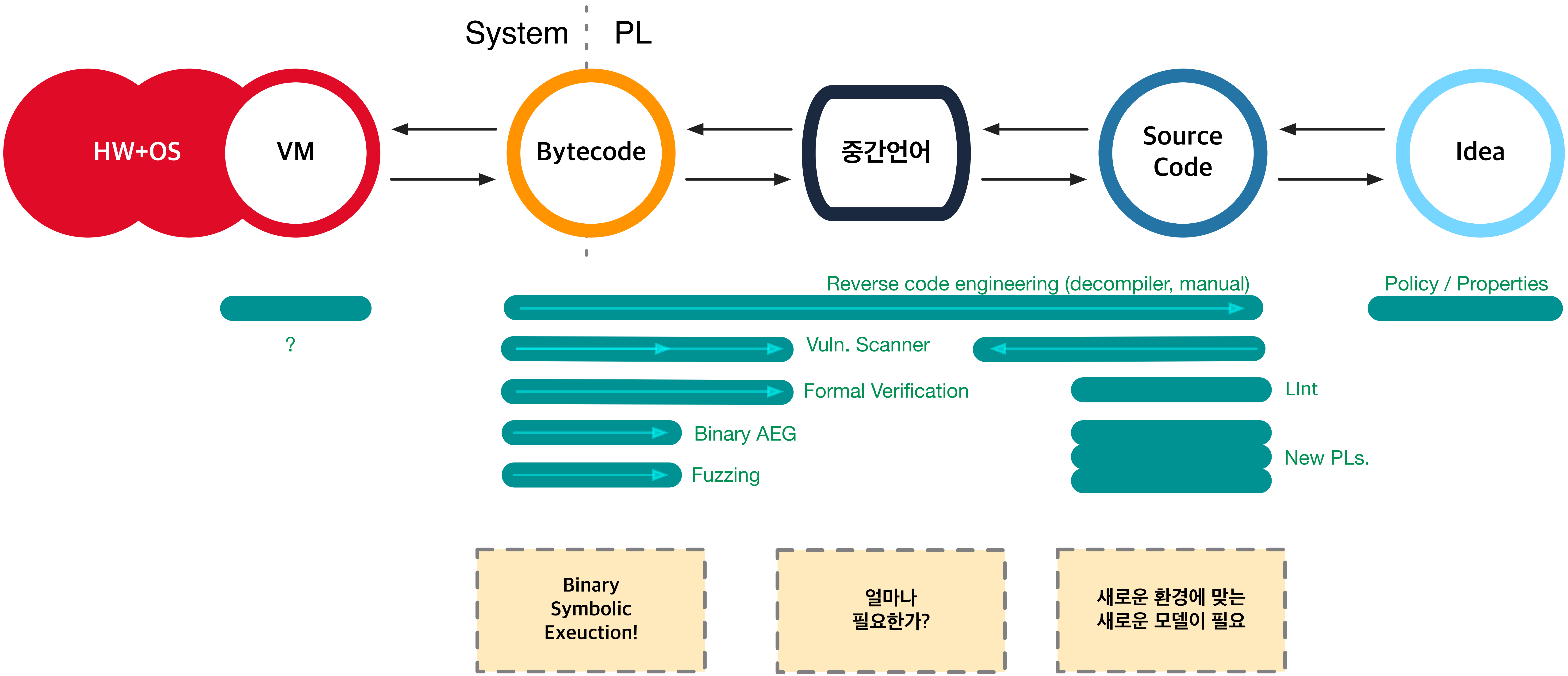
실제 문제는 여기에
있으나 너무 복잡하다.
(Heuristics의 세상)

Binary
분석의
마지노선

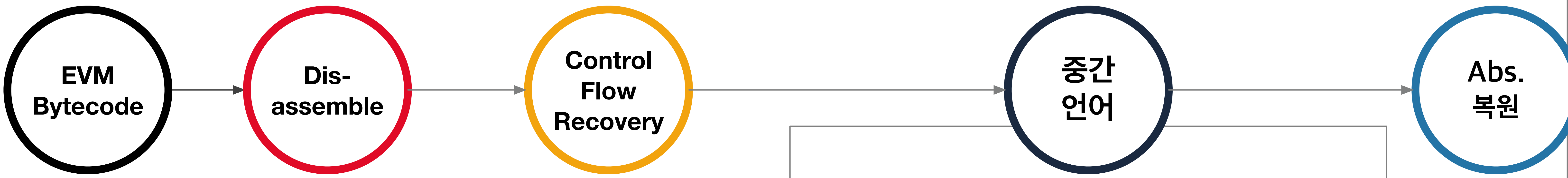
기존의 강자가
너무 세다.
(Parsing도 어렵다)

Mind
the gap!

Smart contract에 대한 현재 접근 방법

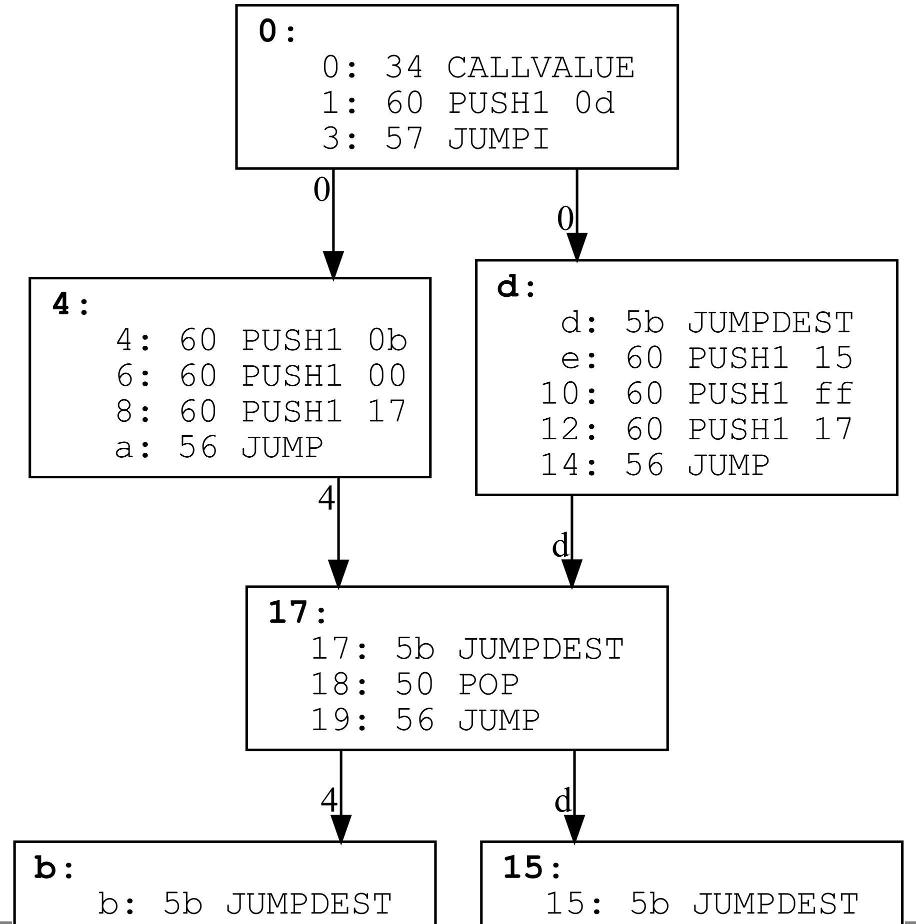


(자동화된) 분석의 시작



Linear sweep / Recursive traversal

Heuristics / Concrete execution / Abstract interpretation



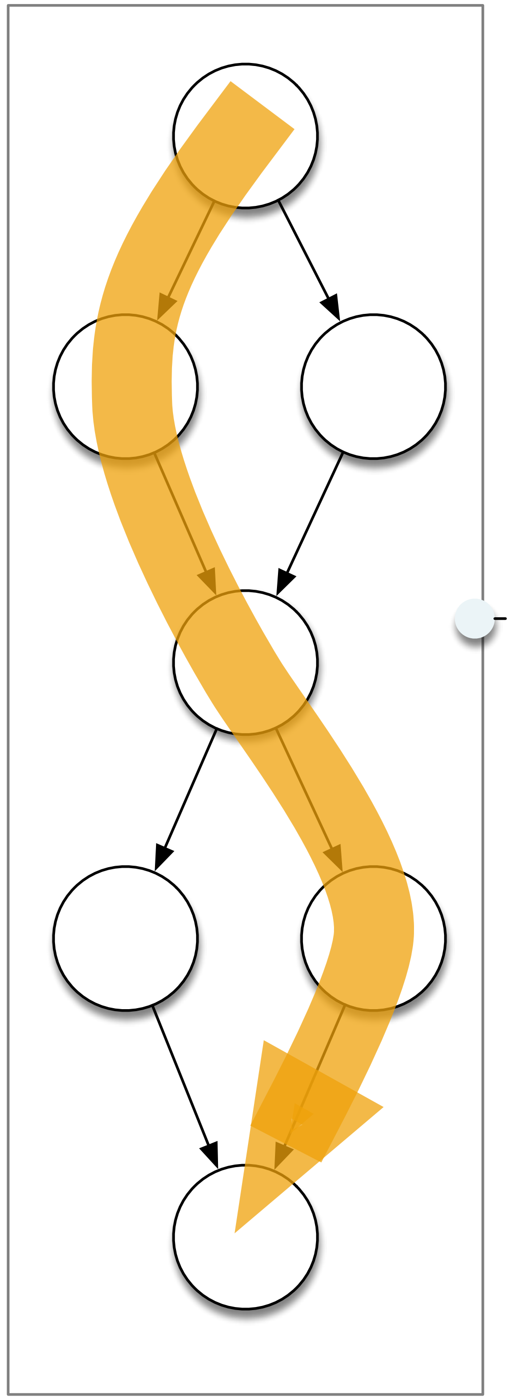

```
PUSH1 01
PUSH1 02
ADD
```

```
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?

Oyente
Mythril
Manticore
…

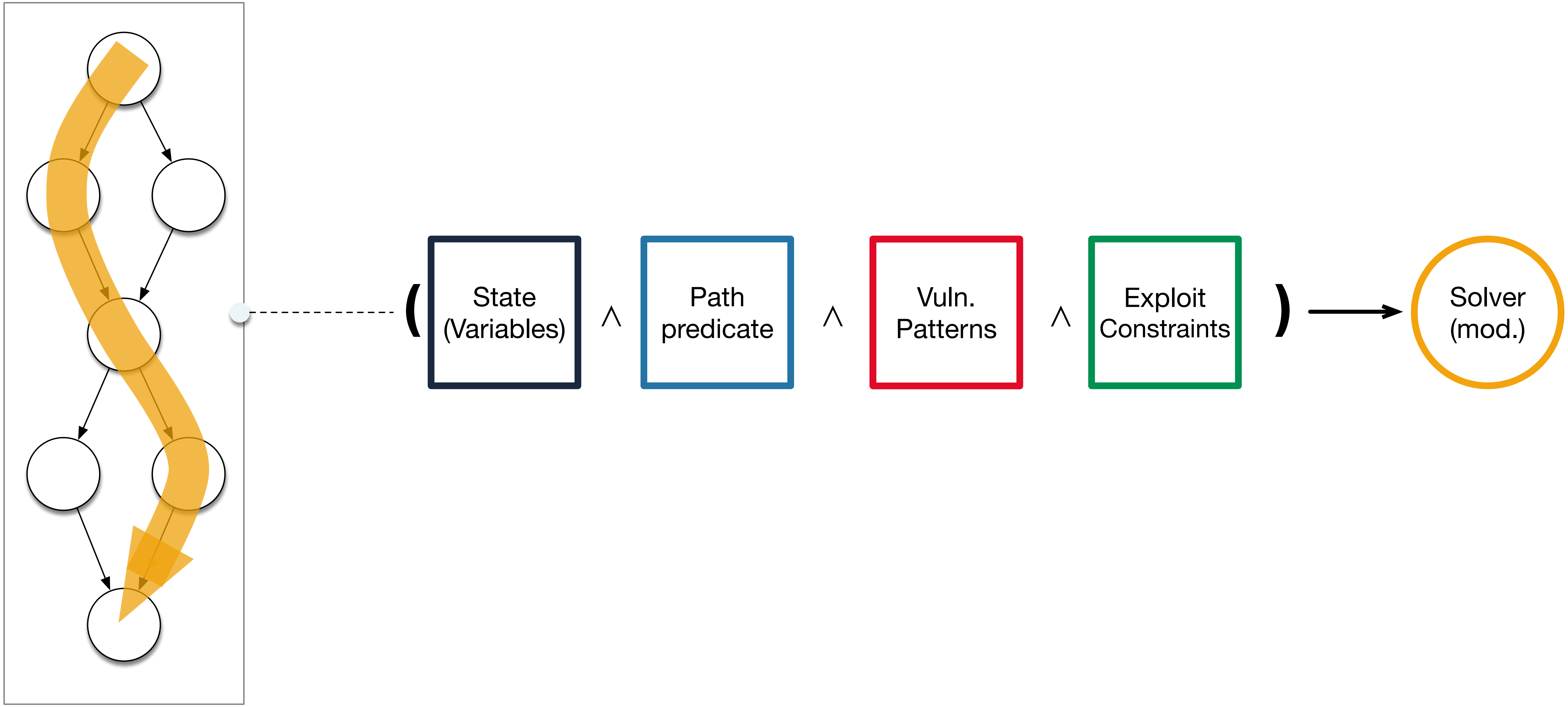
e.g.
Interger
over/underflow

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

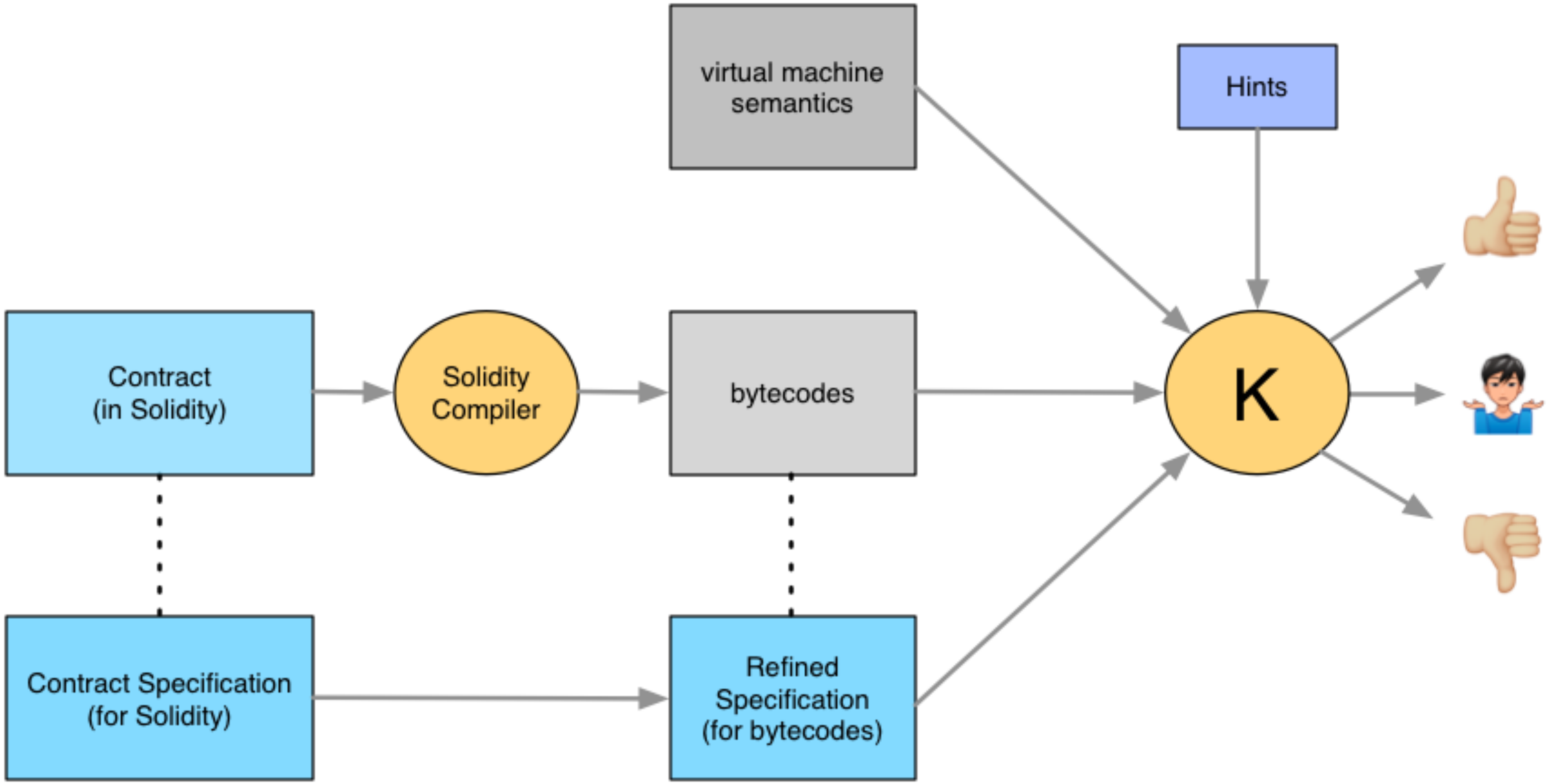
Bug Type	Benchmark	MythrilPip 0.17.12	ManticoreGit 2018-05-18 18:01:09	OyentePip 0.2.7
Integer Overflow	minimal	True Positive	True Positive	False Negative
Integer Overflow	add	True Positive	True Positive	Unsupported
Integer Overflow	mul	True Positive	True Positive	Unsupported
Integer Overflow	path 1	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



Formal Verfication



New Programming Languages

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

(From Vitalik Buterin's tweet)

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

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

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
```

감사합니다.

jonghyup@gmail.com