Binary analysis, meet the blockchain

Presented by: Mark Mossberg

Felipe Manzano, Yan Ivnitskiy, Mark Mossberg

Blockchain

Blockchain

"...is a decentralized, distributed and public **digital ledger** that is used to **record transactions**..."

(Wikipedia)

Blockchains have useful properties

- Resilient
- Verifiable
- Transparent
- Immutable

Ethereum

Ethereum

Blockchain-based, decentralized
computation platform

Ethereum

- Second largest cryptocurrency by valuation
- Peak market cap $100 billion+ (Jan 2018)


Ether Historical Market Capitalization Chart (USD)
Source: Etherscan.io
Click and drag in the plot area to zoom
Monday, January 15, 2018
[Cap Value : 123960214855.64719]
Market Cap: USD 123.96 (Billion)
Avg Price/Ether: $1,278.69
Ether Market Cap (USD)
150 B
125 B
100 B
75 B
50 B
25 B
0 B
Jan '16
Jul '16
Jan '17
Jul '17
Jan '18

Ethereum

- Second largest cryptocurrency by valuation
- Peak market cap $100 billion+ (Jan 2018)
- “**Smart contract**” framework

Smart contracts

"applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference" (Ethereum.org)

Smart contracts

- Ethereum application layer programs
  - Asset management
  - Voting
  - Auctions
  - Crowdfunding
  - ….

Smart contracts

- Ethereum application layer programs
  - Asset management
  - Voting
  - Auctions
  - Crowdfunding
  - ….
- Can have bugs

Smart contracts can have bugs

KLINT FINLEY BUSINESS 06.18.16 04:30 AM

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

Mitch Brenner Follow
Sep 12, 2017 · 7 min read

How I Snatched 153,037 ETH After A Ba Tinder Date

Security

Parity's $280m Ethereum wallet freeze was no accident: It was a hack, claims angry upstart

Smart contracts

- Ethereum application layer programs
  - Asset management
  - Voting
  - Markets
  - Crowdfunding
  - ....
- Can have bugs

Need analysis tooling!

Symbolic Execution

- Powerful program analysis technique
- Proven utility in software security, testing fields
- Could be useful for Ethereum?

Agenda

- Symbolic Execution
- Ethereum Internals
- Symbolic Execution + Ethereum

Symbolic Execution

Programming Languages

B. Wegbreit Editor

Symbolic Execution and Program Testing

James C. King
IBM Thomas J. Watson Research Center

(1975)

This paper describes the symbolic execution of programs. Instead of supplying the normal inputs to a program (e.g. numbers) one supplies symbols representing arbitrary values. The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols. The difficult, yet in-

KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs

Cristian Cadar, Daniel Dunbar, Dawson Engler *

(KLEE, 2008)

BAP: A Binary Analysis Platform

David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz

(BAP, 2011)

Automated Whitebox Fuzz Testing

Patrice Godefroid
Microsoft (Research)
pg@microsoft.com

Michael Y. Levin
Microsoft (CSE)
mlevin@microsoft.com

David Molnar*
UC Berkeley
dmolnar@eecs.berkeley.edu

(SAGE, 2012)

Unleashing MAYHEM on Binary Code

Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley
Carnegie Mellon University
Pittsburgh, PA
{*sangkilc, thanassis, alexandre.rebert, dbrumle*

(MAYHEM, 2012)

CYBER GRAND_CHALLENGE

(2016)

KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs

Cristian Cadar, Daniel Dunbar, Dawson Engler *
Stanford University

(KLEE, 2008)

(BAP, 2011)

David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz

Automated Whitebox Fuzz Testing

(SAGE, 2012)

Patrice Godefroid
Microsoft (Research)
pg@microsoft.com

Michael Y. Levin
Microsoft (CSE)
mlevin@microsoft.com

David Molnar*
UC Berkeley
dmolnar@eecs.berkeley.edu

Unleashing MAYHEM on Binary Code

(MAYHEM, 2012)

Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley
Carnegie Mellon University
Pittsburgh, PA
{sangkilc, thanassis, alexandre.rebert, dbrumley}@cmu.edu

(2016)

CYBER GRAND_CHALLENGE

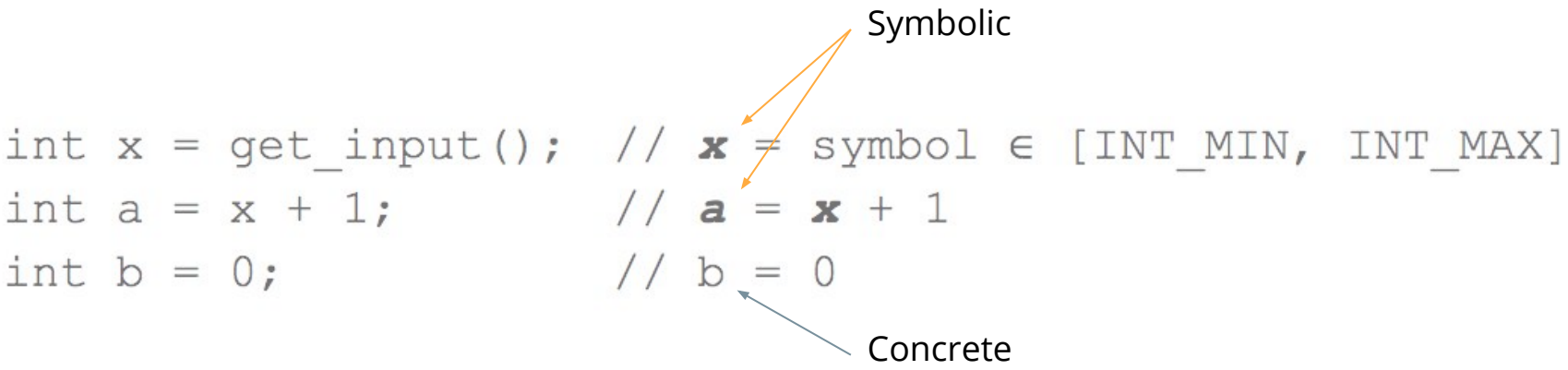
Concrete Execution

```
int x = get_input();  // x = 42
int a = x + 1;        // a = 43
int b = 0;            // b = 0
```

Symbolic Execution

```
int x = get_input();  // x = symbol ∈ [INT_MIN, INT_MAX]
int a = x + 1;        // a = x + 1
int b = 0;            // b = 0
```

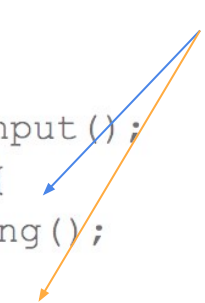
Symbolic Execution



Symbolic Execution: State Forking

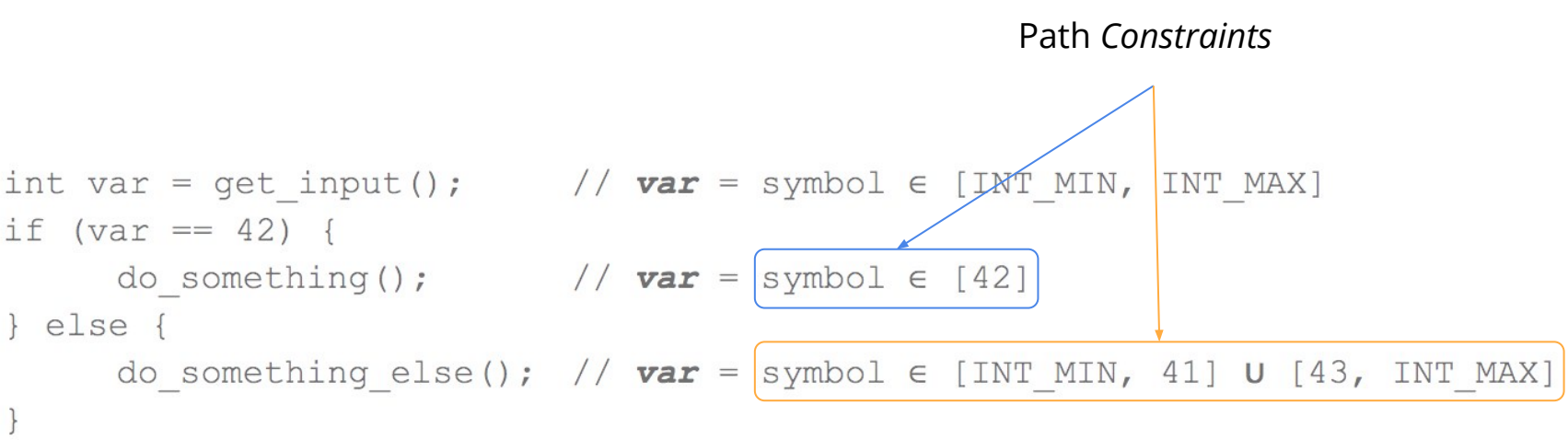
```
int var = get_input();       // var = symbol ∈ [INT_MIN, INT_MAX]
if (var == 42) {
      do_something();        // var = symbol ∈ [42]
} else {
      do_something_else();   // var = symbol ∈ [INT_MIN, 41] U [43, INT_MAX]
}
```

Symbolic Execution: State Forking




```
int var = get_input();       // var = symbol ∈ [INT_MIN, INT_MAX]
if (var == 42) {
     do_something();         // var = symbol ∈ [42]
} else {
     do_something_else();    // var = symbol ∈ [INT_MIN, 41] U [43, INT_MAX]
}
```

Symbolic Execution: State Forking



Symbolic execution example

```
1. int a = α, b = β, c = γ;
2.                  // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```

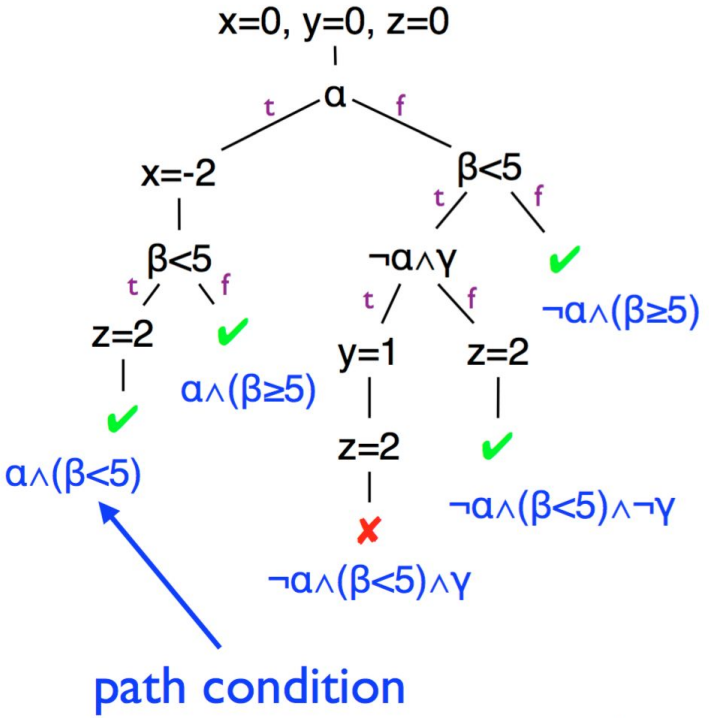


Image from Michael Hicks, University of Maryland

Constraint Solvers

$\neg\alpha \wedge (\beta < 5) \wedge \gamma$

SAT, α=false, β=3, γ=true

z3 Microsoft Research

Constraint Solvers

$\neg a \wedge a$

Microsoft Research

z3

UNSAT

Challenges

- Path explosion
- Symbolic memory indexing
- Symbolic/infinite loops

Challenges

- Path explosion
- Symbolic memory indexing
- Symbolic/infinite loops

```
...
mov rbx, [rax] // symbolic rax
...
```

Challenges

- Path explosion
- Symbolic memory indexing
- Symbolic/infinite loops

```
int var = get_input();
for (int i = 0; i < var; i++) {
    ...
}
```

Using Symbolic Execution, we can:

- Test many paths in the program
- Systematically analyze new program code
- Prove properties about programs

Ethereum Internals

Ethereum

- Decentralized computation platform
- Includes a “cryptocurrency” implemented using a blockchain (ether)
- Includes a **virtual machine**

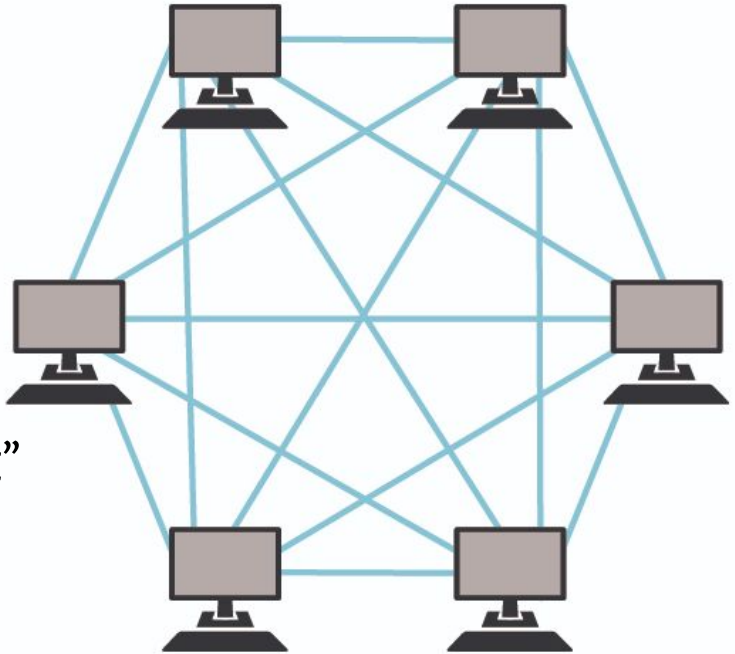
Ethereum Internals

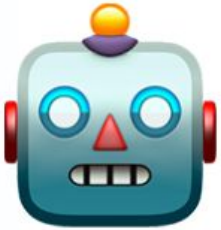
- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

Ethereum Internals

- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

Ethereum entities





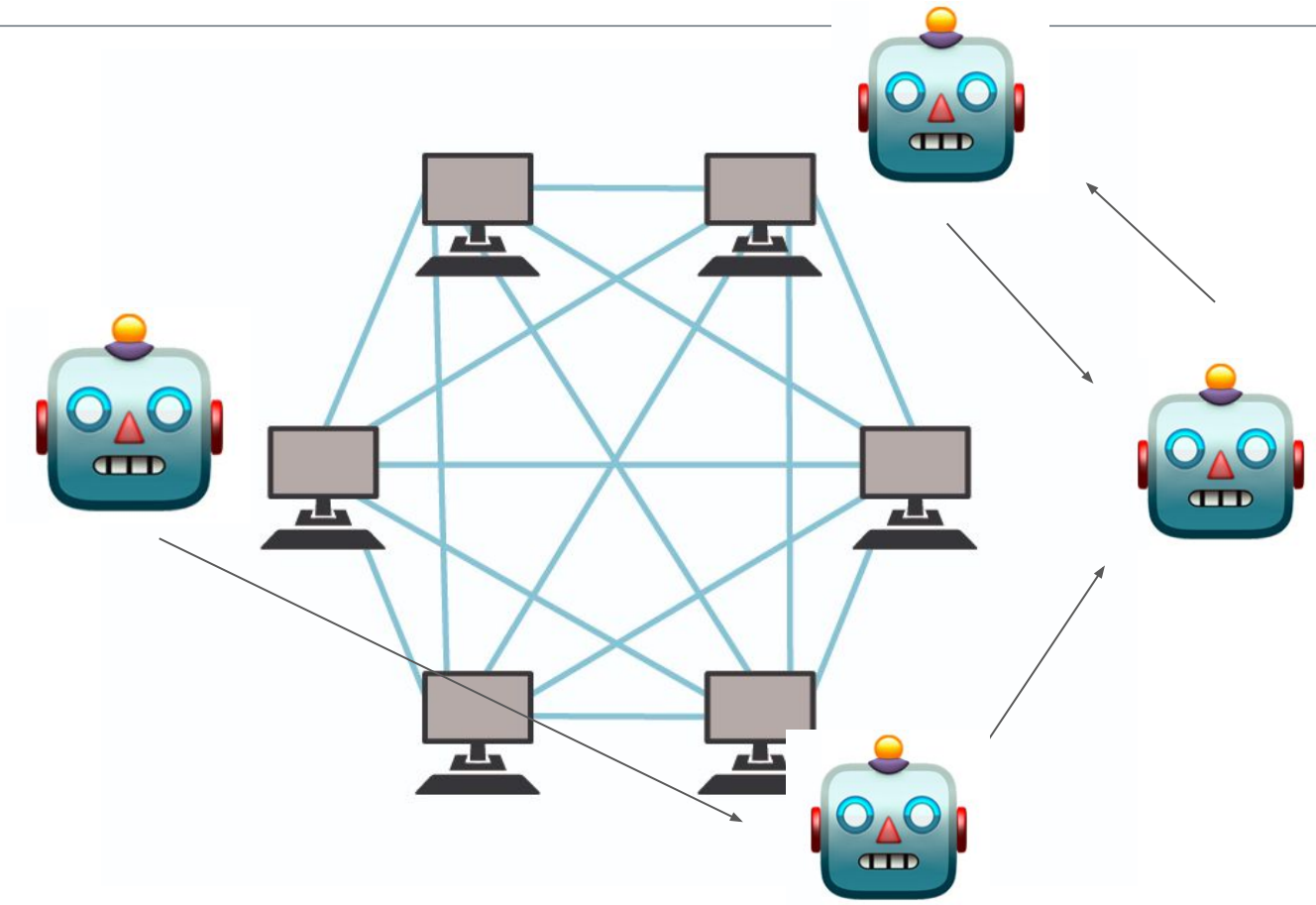
“External Account”

- balance

“Contract Account”

- balance
- code

Contracts can call other contracts



Smart contracts

- Consist of state variables & functions
- Effectively encode state machines
- Commonly contain many assertions
  - State rollback mechanism for error handling
- Programmed in “Solidity”
  - Source code generally unavailable
- Execute when the contract account receives a transaction

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

    /* Initializes contract with initial supply tokens to
    function MyToken(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
    }
}
```

Smart contracts

- Consist of state variables & functions
- Effectively encode state machines
- Commonly contain many assertions
  - State rollback mechanism for error handling
- Programmed in “Solidity”
  - Source code generally unavailable
- Execute when the contract account receives a transaction

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

    /* Initializes contract with initial supply tokens to
    function MyToken(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
    }
}
```

Smart contracts

- Consist of state variables & functions
- Effectively encode state machines
- Commonly contain many assertions
  - State rollback mechanism for error handling
- Programmed in “Solidity”
  - Source code generally unavailable
- Execute when the contract account receives a transaction

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

    /* Initializes contract with initial supply tokens to
    function MyToken(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
    }
}
```

Smart contracts

- Consist of state variables & functions
- Effectively encode state machines
- Commonly contain many assertions
  - State rollback mechanism for error handling
- Programmed in “Solidity”
  - Source code generally unavailable
- Execute when the contract account receives a transaction

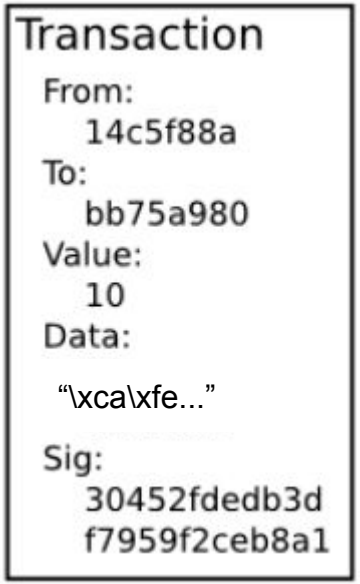
```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to
    function MyToken(
        uint256 initialSupply
        ) public {
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
    }
}
```

Ethereum Internals

- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

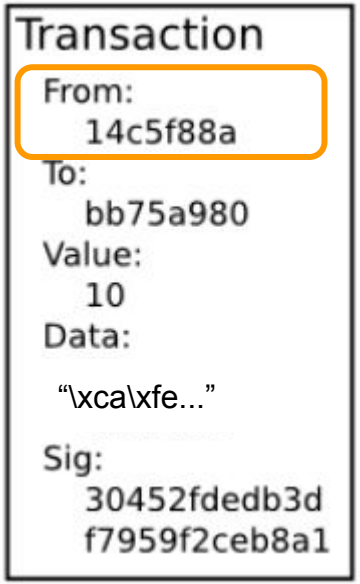
Ethereum transactions

- Fundamental communication interface
  - Transfer ether
  - Deploy contracts
  - Interact with contracts



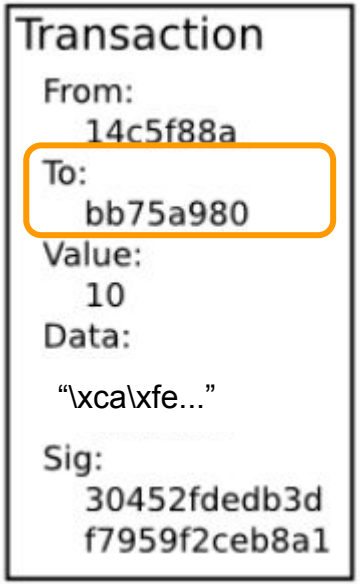
Ethereum transactions

- From/To: Account address
- Value: Ether amount to send
- Data: Arbitrary data buffer
  - Used when interacting with contracts



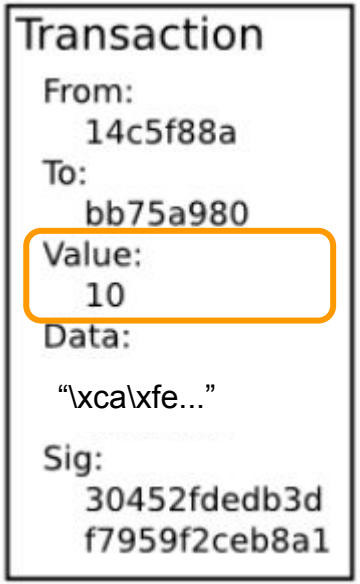
Ethereum transactions

- From/To: Account address
- Value: Ether amount to send
- Data: Arbitrary data buffer
  - Used when interacting with contracts



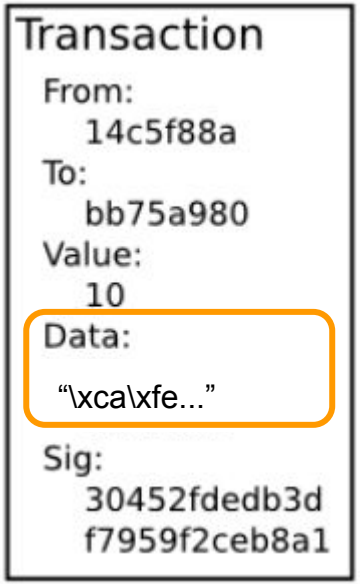
Ethereum transactions

- From/To: Account address
- Value: Ether amount to send
- Data: Arbitrary data buffer
  - Used when interacting with contracts



Ethereum transactions

- From/To: Account address
- Value: Ether amount to send
- Data: Arbitrary data buffer
  - Used when interacting with contracts



Ethereum Internals

- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

Ethereum Virtual Machine

- Stack machine
- **256 bit** native word size
- ~181 instructions
  - Arithmetic
  - Control Flow
  - Memory
  - Domain Specific
- Gas: Instruction execution cost
- Compilation target for Solidity

Ethereum Virtual Machine

- Stack machine
- **256 bit** native word size
- ~181 instructions
  - Arithmetic
  - Control Flow
  - Memory
  - Domain Specific
- Gas: Instruction execution cost
- Compilation target for Solidity

0x01	ADD	Addition operation
0x02	MUL	Multiplication operation
0x03	SUB	Subtraction operation
0x04	DIV	Integer division operation
0x05	SDIV	Signed integer division operation (truncated)
0x06	MOD	Modulo remainder operation
0x07	SMOD	Signed modulo remainder operation
0x08	ADDMOD	Modulo addition operation
0x09	MULMOD	Modulo multiplication operation
0x0a	EXP	Exponential operation
0x0b	SIGNEXTEND	Extend length of two's complement signed integer

https://github.com/trailofbits/evm-opcodes

Ethereum Virtual Machine

- Stack machine
- **256 bit** native word size
- ~181 instructions
  - Arithmetic
  - Control Flow
  - Memory
  - Domain Specific
- Gas: Instruction execution cost
- Compilation target for Solidity

`0x56`	JUMP	Alter the program counter
`0x57`	JUMPI	Conditionally alter the program counter

https://github.com/trailofbits/evm-opcodes

Ethereum Virtual Machine

- Stack machine
- **256 bit** native word size
- ~181 instructions
  - Arithmetic
  - Control Flow
  - Memory
  - Domain Specific
- Gas: Instruction execution cost
- Compilation target for Solidity

0x51	MLOAD	Load word from memory
0x52	MSTORE	Save word to memory
0x53	MSTORE8	Save byte to memory
0x54	SLOAD	Load word from storage
0x55	SSTORE	Save word to storage

https://github.com/trailofbits/evm-opcodes

Ethereum Virtual Machine

0x30	ADDRESS	Get address of currently executing account
0x31	BALANCE	Get balance of the given account
0x32	ORIGIN	Get execution origination address
0x33	CALLER	G
0x34	CALLVALUE	G re
0x35	CALLDATALOAD	G

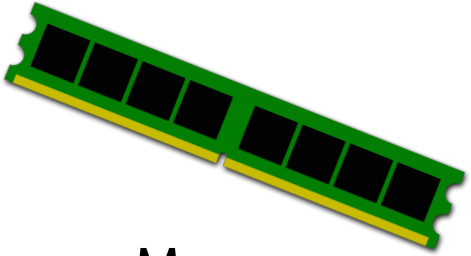
0x40	BLOCKHASH	Get the hash of one of the 256 most recent complete blocks
0x41	COINBASE	Get the block's beneficiary address
0x42	TIMESTAMP	Get the block's timestamp
0x43	NUMBER	Get the block's number
0x44	DIFFICULTY	Get the block's d
0x45	GASLIMIT	Get the block's g

0xfd	REVERT	Stop execution and re consuming all provide
0xfe	INVALID	Designated invalid ins
0xff	SELFDESTRUCT	Halt execution and re

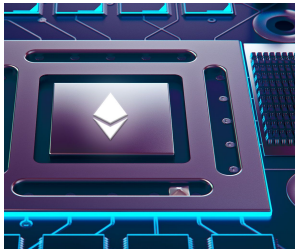
https://github.com/trailofbits/evm-opcodes

EVM: Memory Regions

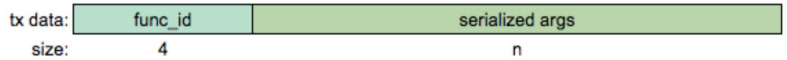
Storage



Memory



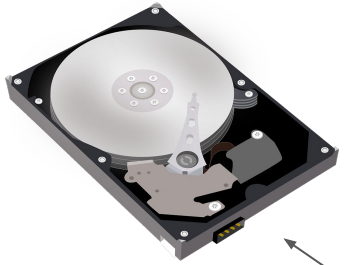
EVM



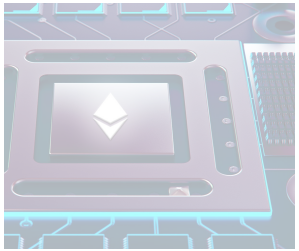
Calldata

Stack

EVM: Memory Regions



Storage

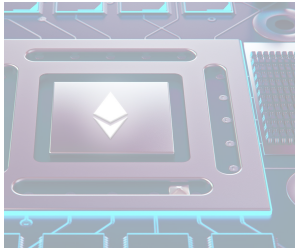


Persistent, “infinite” virtual disk space

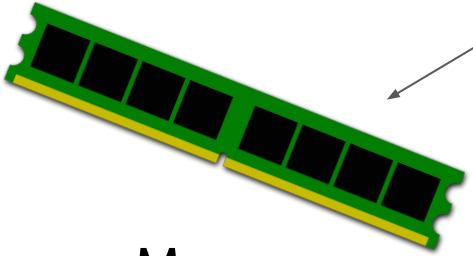
(256 bit addressable)

EVM: Memory Regions

tx data:	func_id	serialized args
size:	4	n



EVM

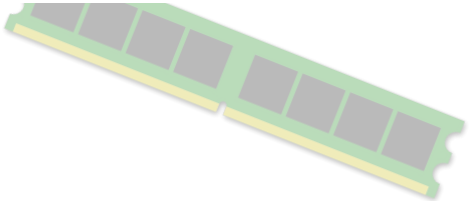


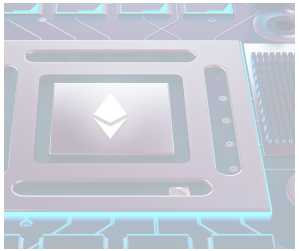
Memory

Volatile, heap-like virtual memory region

EVM: Memory Regions

Transaction data buffer





EVM

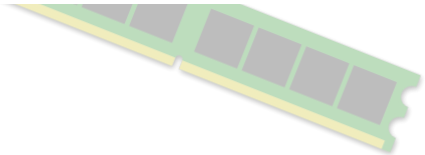
Calldata

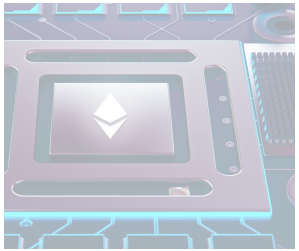
EVM: Memory Regions

tx data:	func_id	serialized args
size:	4	n

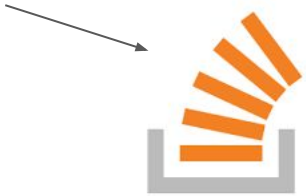
Direct CPU Storage

(1024 word capacity)





EVM



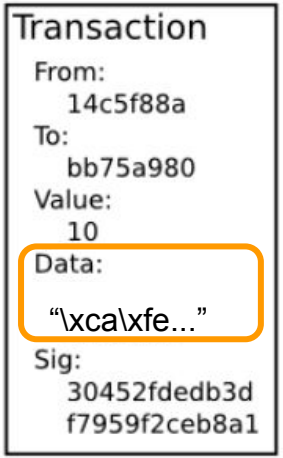
Stack

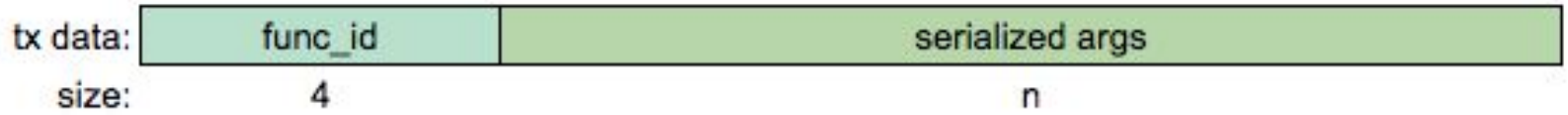
Ethereum Internals

- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

Ethereum ABI

- Specifies how function call information is serialized
  - Function id being called
  - Arguments passed to function
- Serialized call info passed via transaction data field
- ABI spec required to call contract

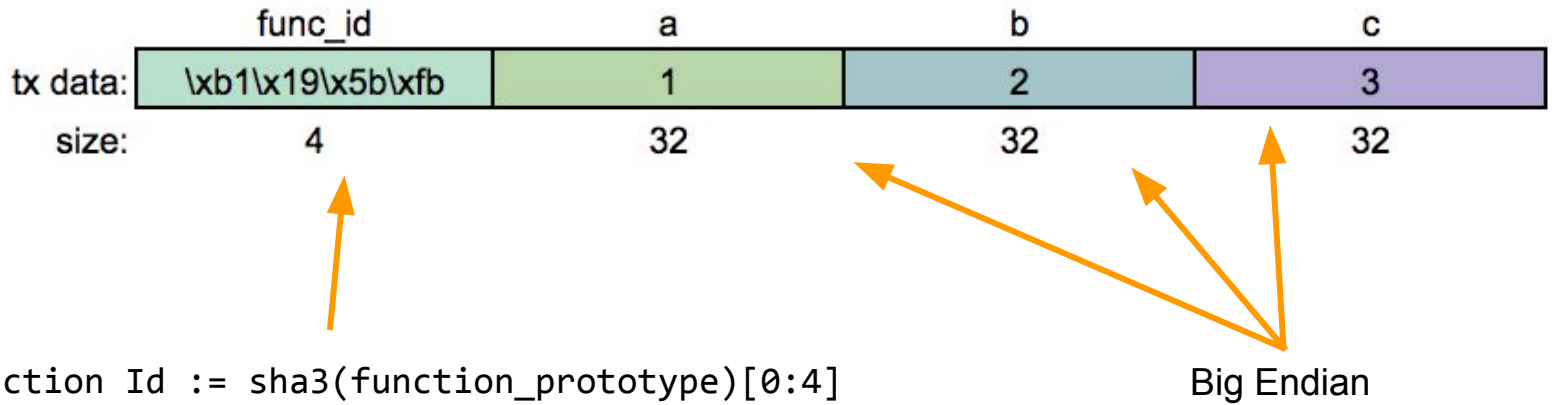




Ethereum ABI Example: Simple Types

```
my_function(uint256 a, uint256 b, uint256 c);
```

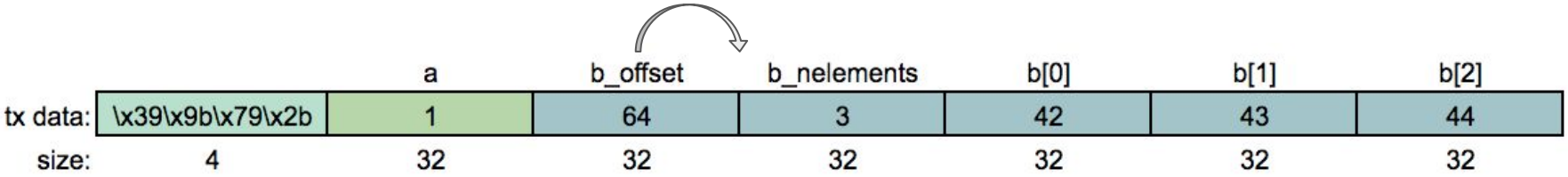
```
my_function(1, 2, 3);
```




```
Function Id := sha3(function_prototype)[0:4]
```

Ethereum ABI Example: Dynamic Types

```
my_function(uint256 a, uint256[] b);

my_function(1, [42, 43, 44]);
```

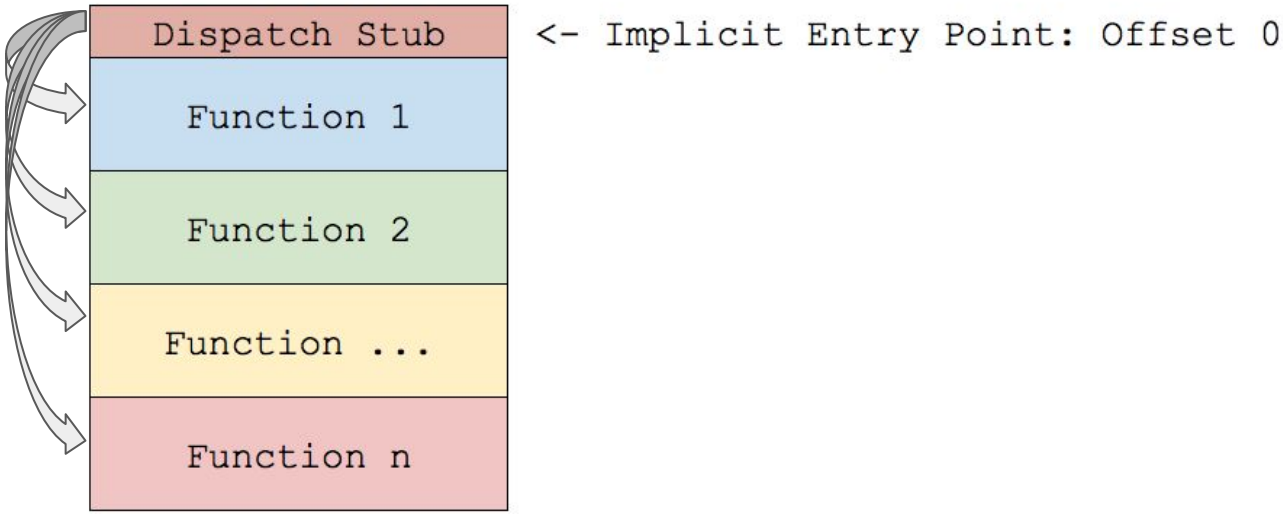


Ethereum Internals

- Smart contracts
- Transactions
- Virtual Machine
- Application Binary Interface
- Bytecode format

EVM Bytecode

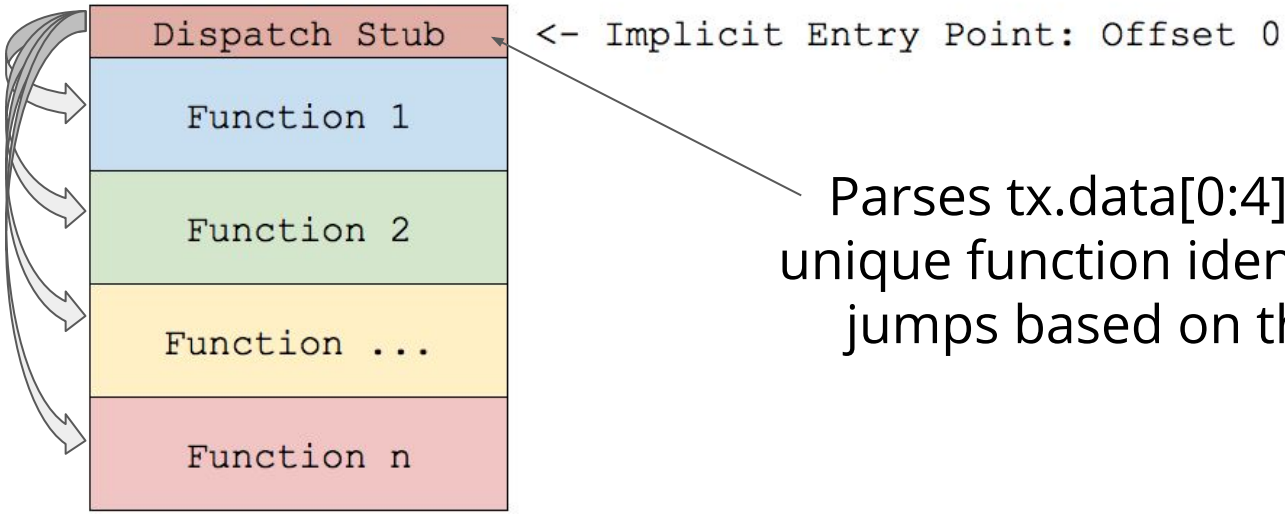
EVM Runtime Bytecode Format



*as of solc 0.4.17

EVM Bytecode

EVM Runtime Bytecode Format

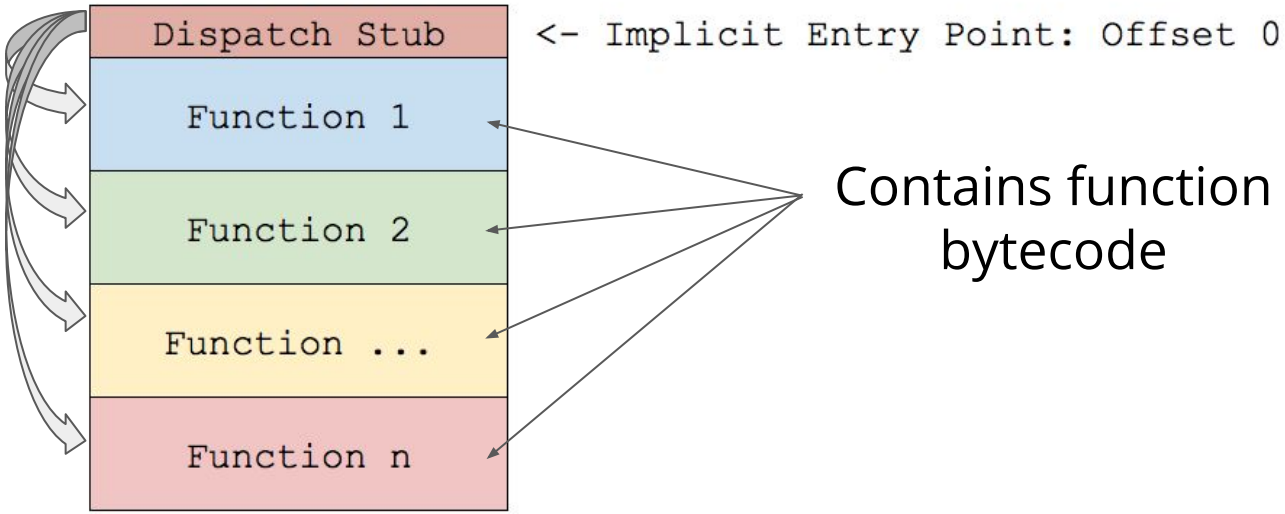


Parses tx.data[0:4] for unique function identifier, jumps based on that

*as of solc 0.4.17

EVM Bytecode

EVM Runtime Bytecode Format



*as of solc 0.4.17

Dispatch Stub

```
_dispatcher:
00000000  6060                PUSH1  0x60
00000002  6040                PUSH1  0x40
00000004  52                  MSTORE
00000005  6000                PUSH1  0x0
00000007  35                  CALLDATALOAD
00000008  7c0100000000000…    PUSH29 0x1000000000000000000000000000000000000000000000000000000000
00000026  90                  SWAP1
00000027  04                  DIV
00000028  63ffffffff          PUSH4  0xffffffff
0000002d  16                  AND
0000002e  80                  DUP1
0000002f  631822932c          PUSH4  0x1822932c
00000034  14                  EQ
00000035  6046                PUSH1  0x46
00000037  57                  JUMPI
```

```
00000038  80                  DUP1
00000039  6390cc0276          PUSH4  0x90cc0276
0000003e  14                  EQ
0000003f  6058                PUSH1  0x58
00000041  57                  JUMPI
```

```
00000046  5b                  JUMPDEST
{ Falls through into 0x1822932c }
```

```
00000042  6000                PUSH1  0x0
00000044  80                  DUP1
00000045  fd                  REVERT
```

```
00000058  5b                  JUMPDEST
{ Falls through into 0x90cc0276 }
```

(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Dispatch Stub

```
_dispatcher:
00000000  6060              PUSH1  0x60
00000002  6040              PUSH1  0x40
00000004  52                MSTORE
00000005  6000              PUSH1  0x0
00000007  35                CALLDATALOAD
00000008  7c0100000000000…  PUSH29 0x100000000000000000000000000000000000000000000000000000
00000026  90                SWAP1
00000027  04                DIV
00000028  63ffffffff        PUSH4  0xffffffff
0000002d  16                AND
0000002e  80                DUP1
0000002f  631822932c        PUSH4  0x1822932c
00000034  14                EQ
00000035  6046              PUSH1  0x46
00000037  57                JUMPI
```

Access tx.data

```
00000038  80                DUP1
00000039  6390cc0276        PUSH4  0x90cc0276
0000003e  14                EQ
0000003f  6058              PUSH1  0x58
00000041  57                JUMPI
```

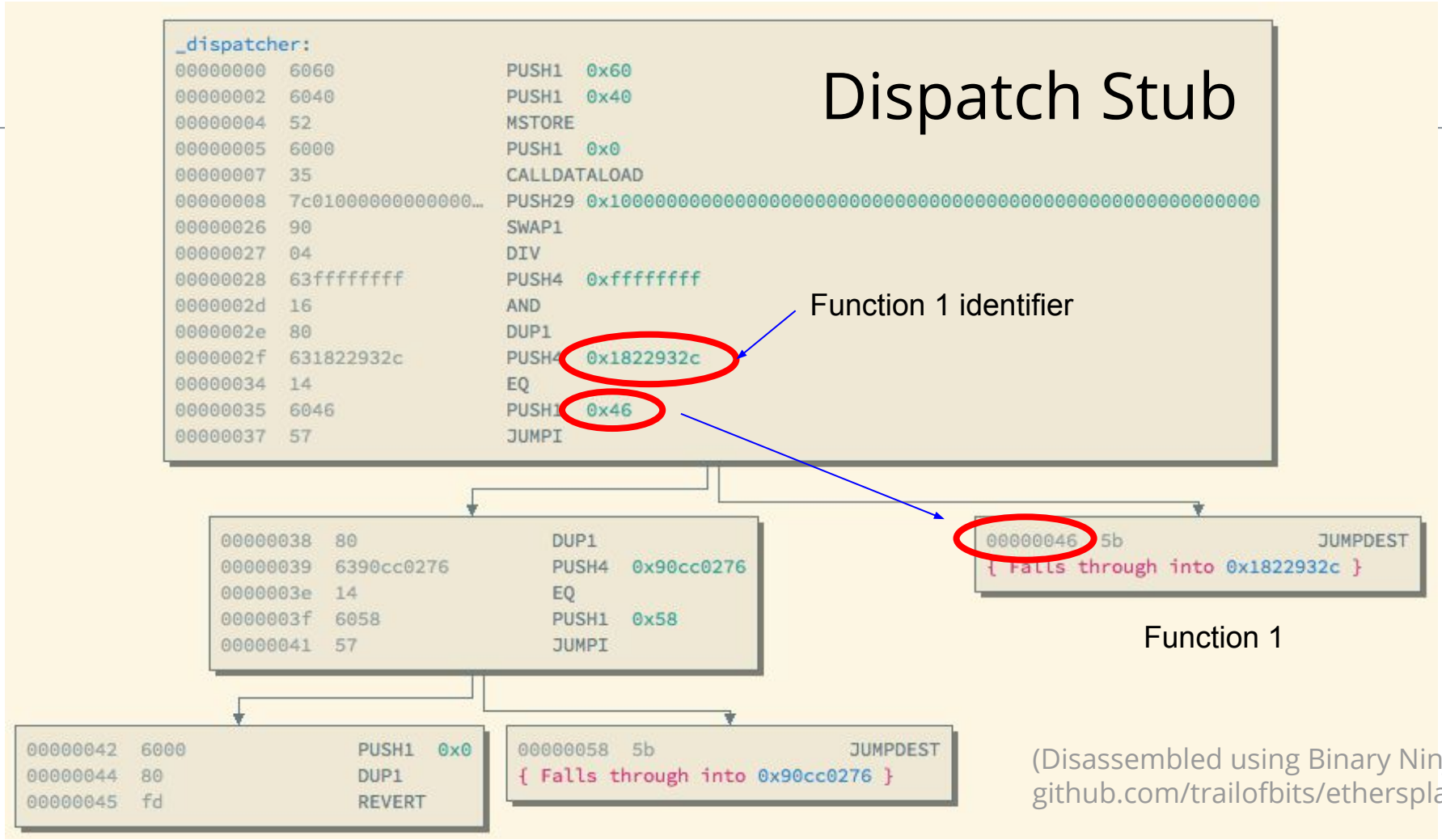
```
00000046  5b                JUMPDEST
{ Falls through into 0x1822932c }
```

```
00000042  6000              PUSH1  0x0
00000044  80                DUP1
00000045  fd                REVERT
```

```
00000058  5b                JUMPDEST
{ Falls through into 0x90cc0276 }
```

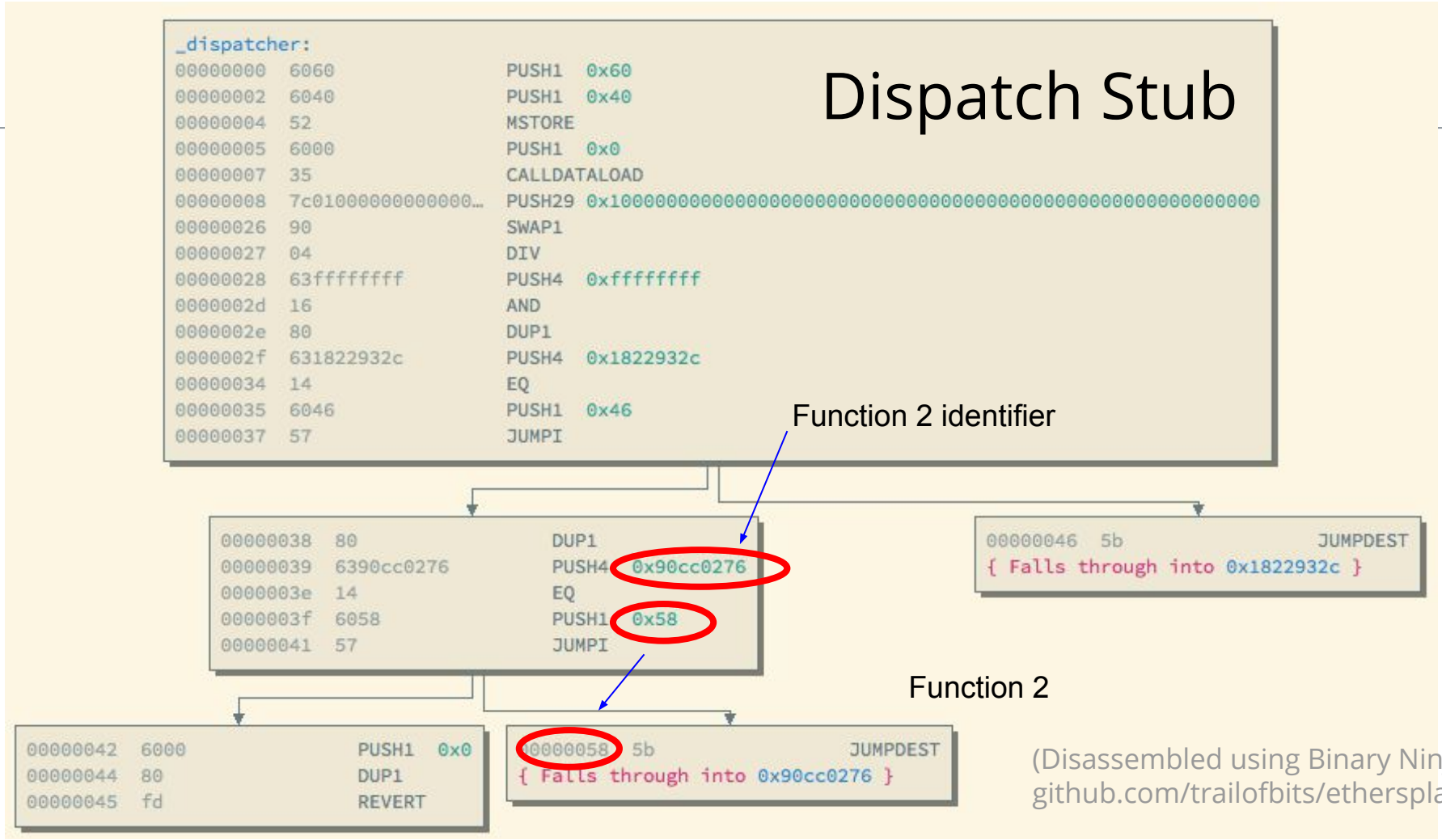
(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Dispatch Stub



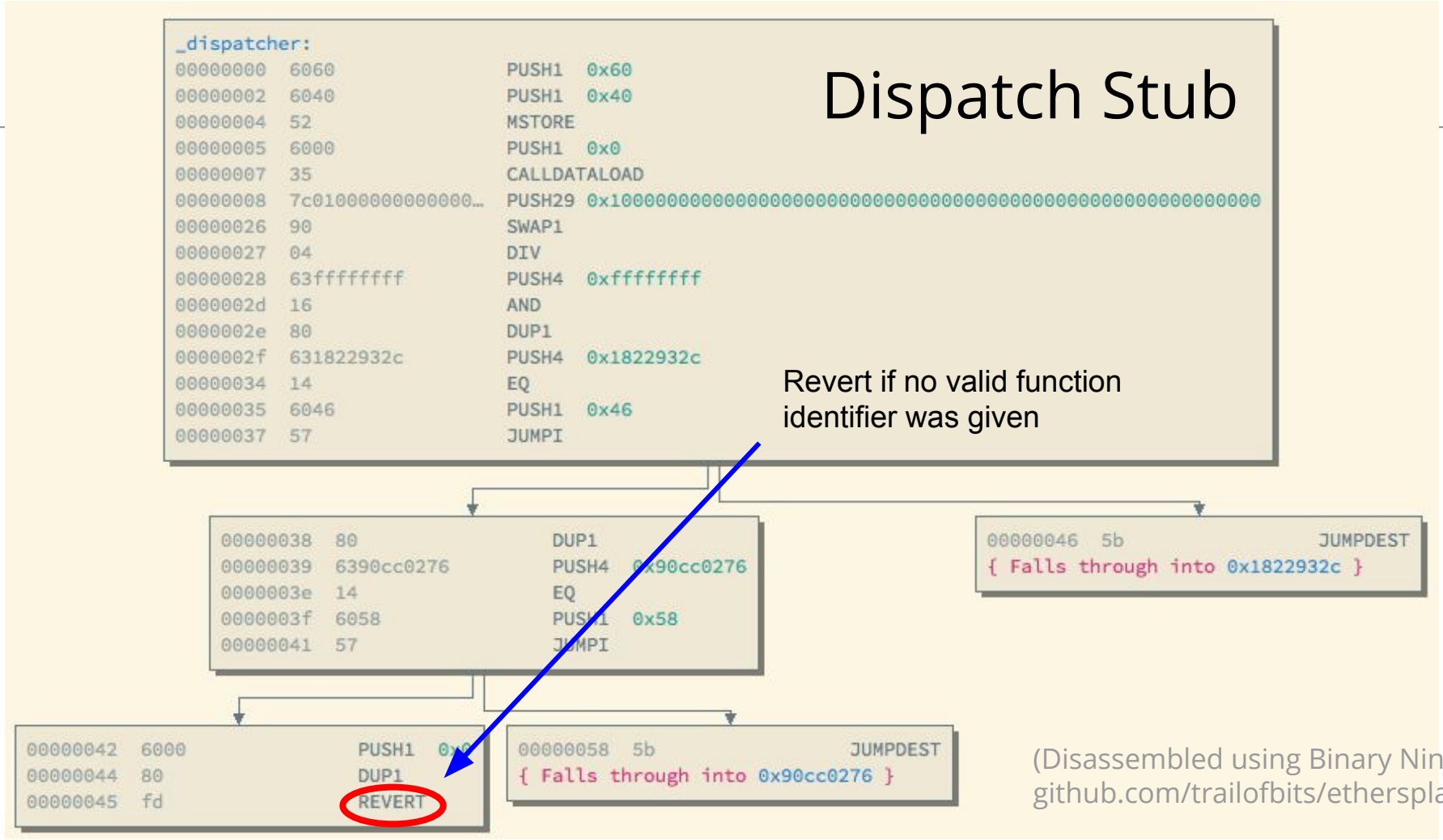
(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Dispatch Stub



(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Dispatch Stub



(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Ethereum

- Decentralized virtual-machine based computation platform
- “Smart Contract” applications: deployed state machines interacted with via **transactions**

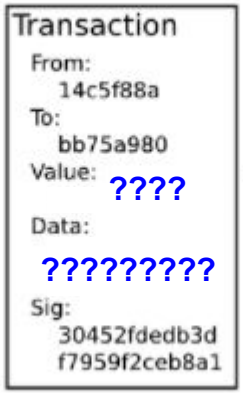
Symbolic Execution + Ethereum

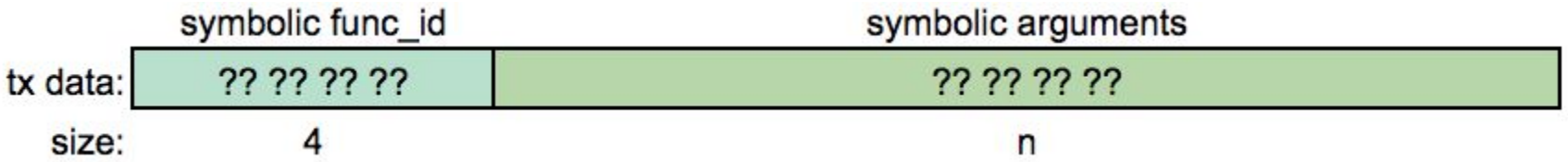
Goals

- Generate inputs that exercise contract functionality
- Enumerate state space & discover failure states
- Allow humans to prove properties about contract

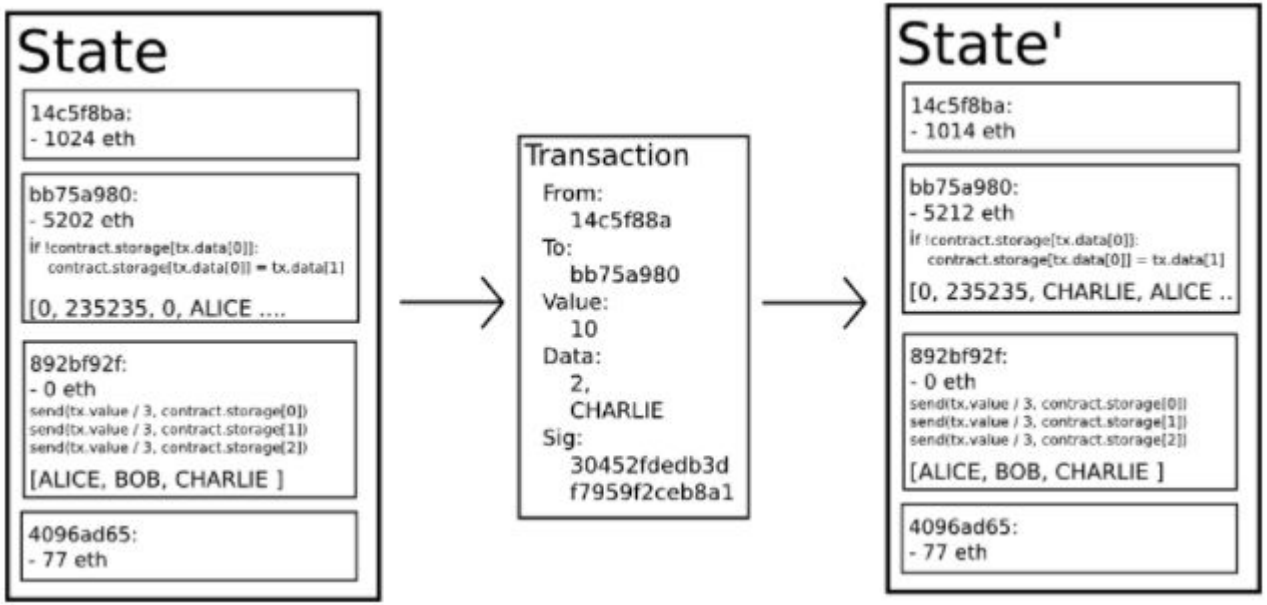
Methodology

- Implement symbolic EVM interpreter
- Execute contracts with symbolic input
  - Symbolic transaction value
  - **Symbolic transaction data buffer**

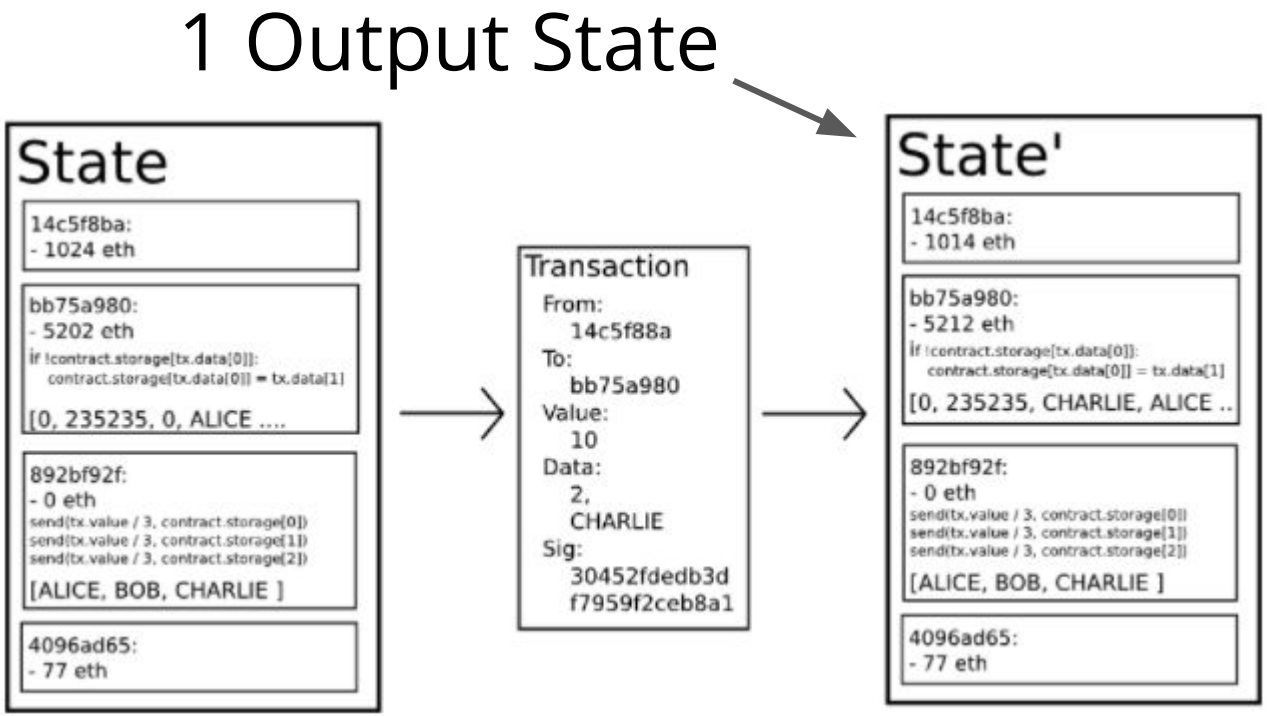




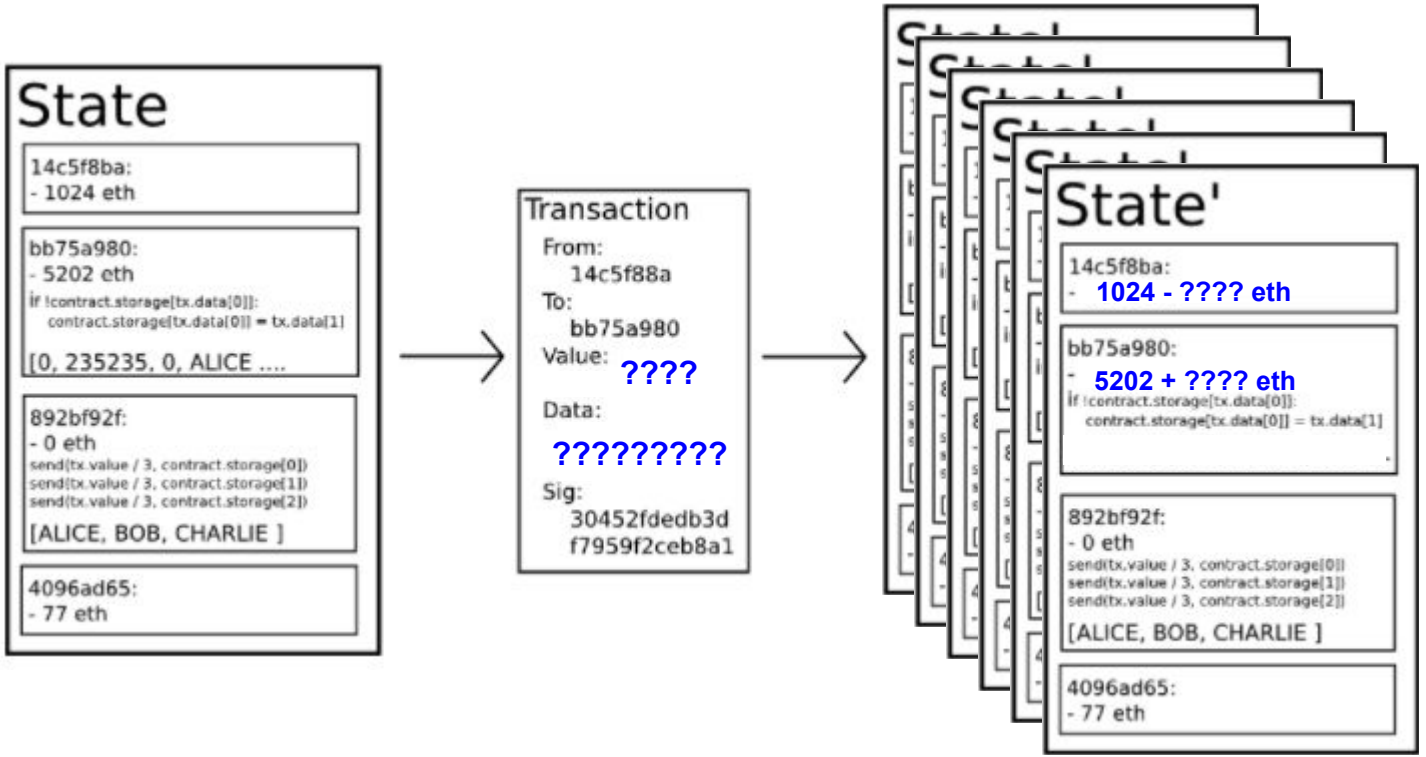
Concrete Transaction



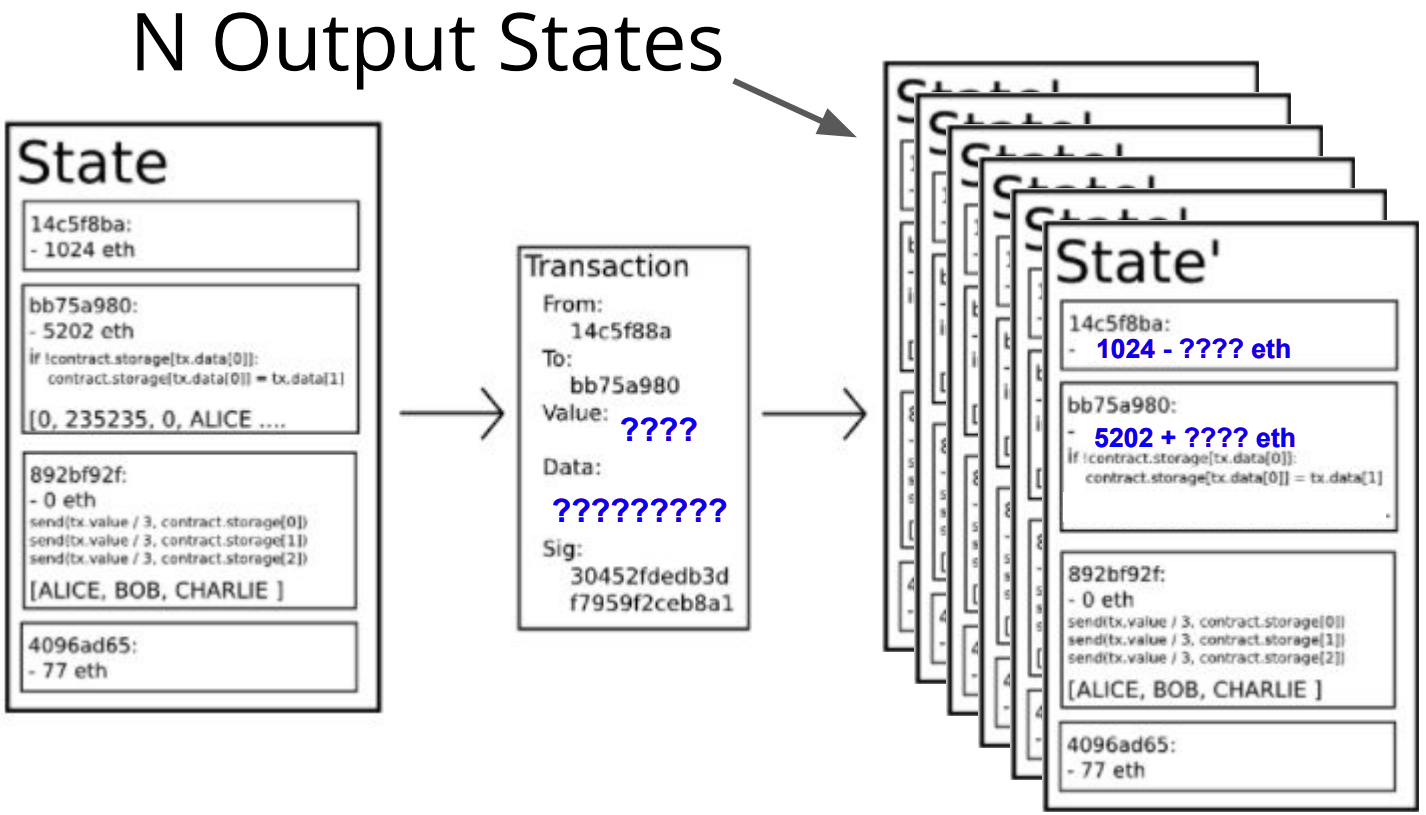
Concrete Transaction



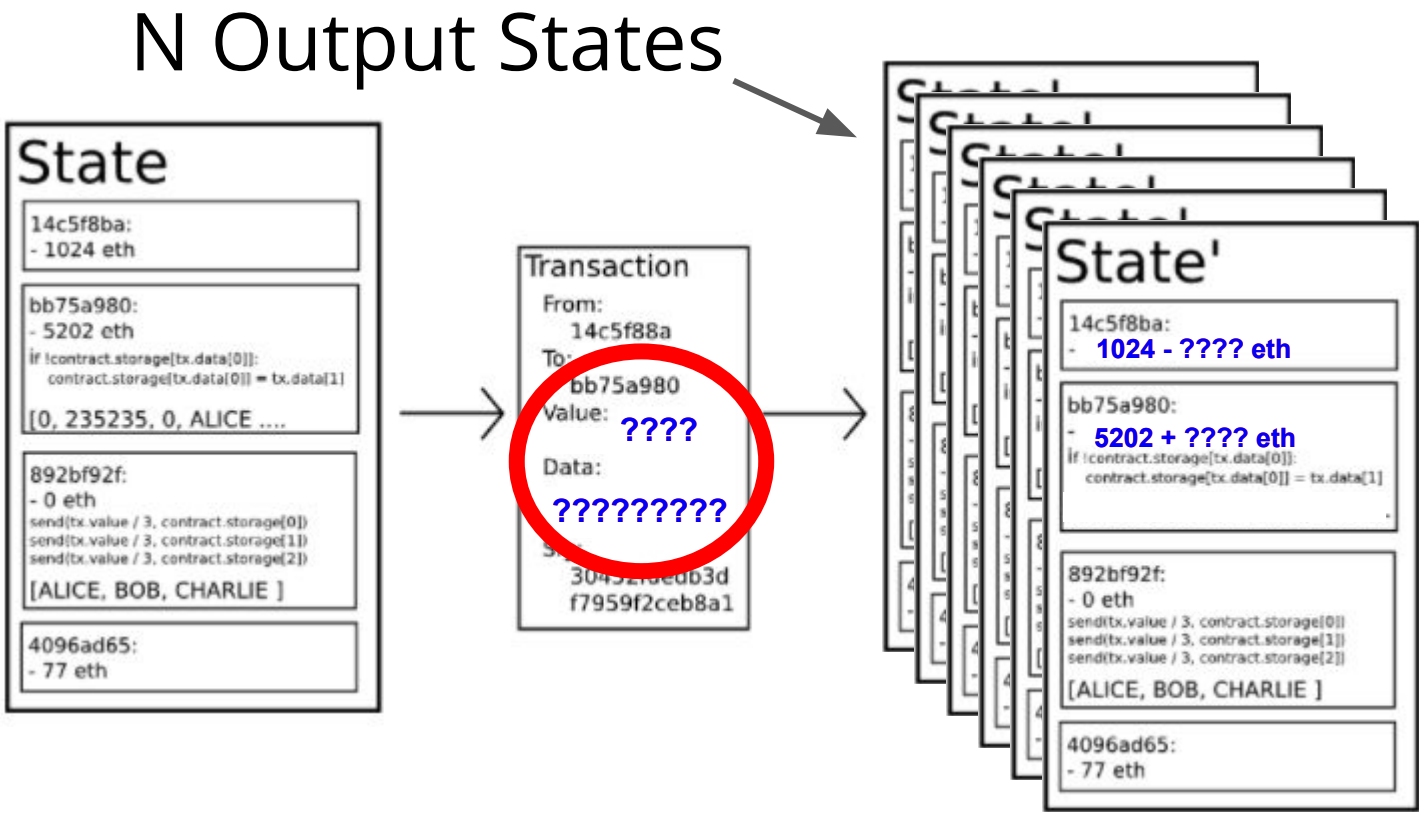
Symbolic Transaction



Symbolic Transaction

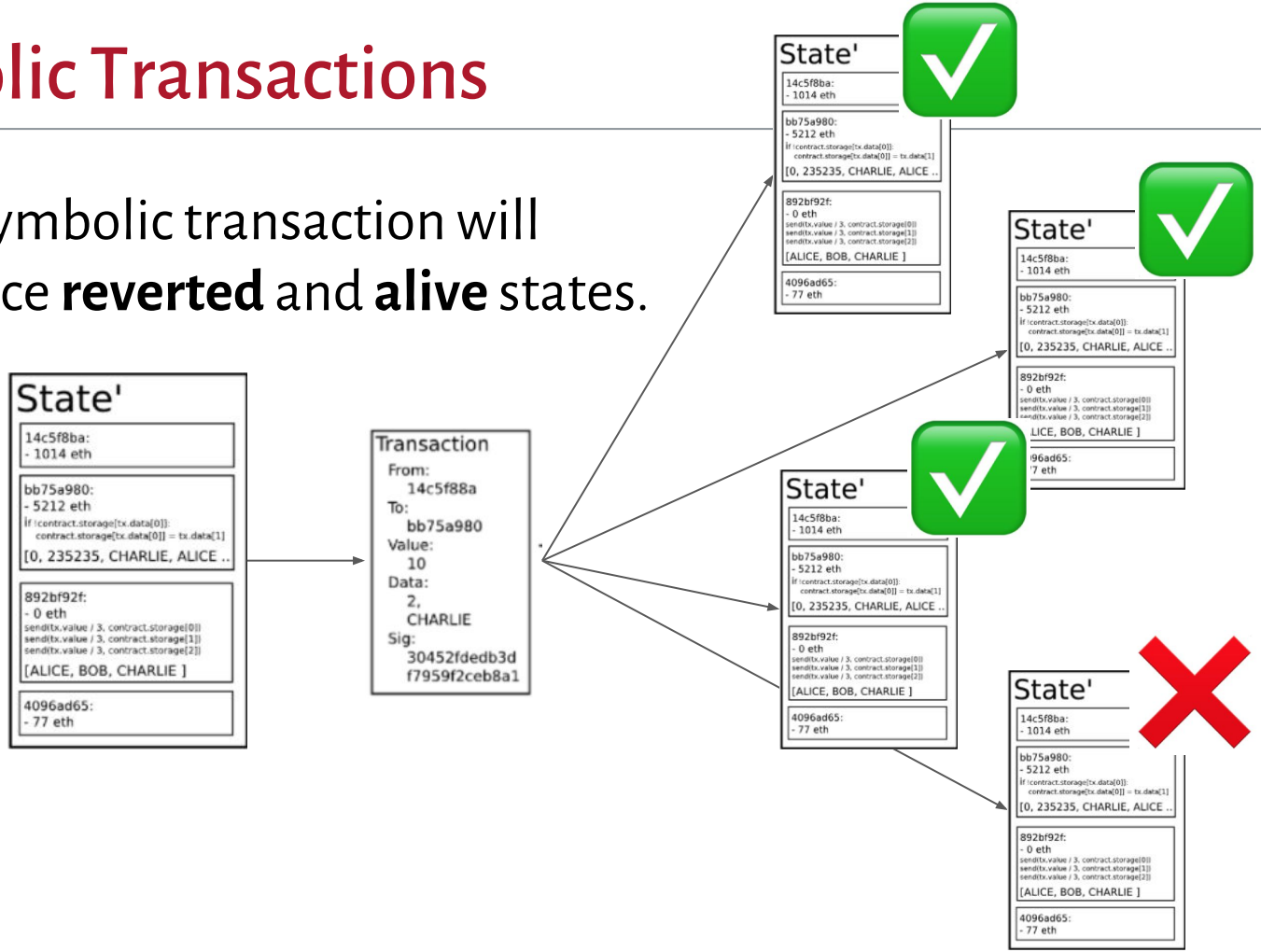


Symbolic Transaction



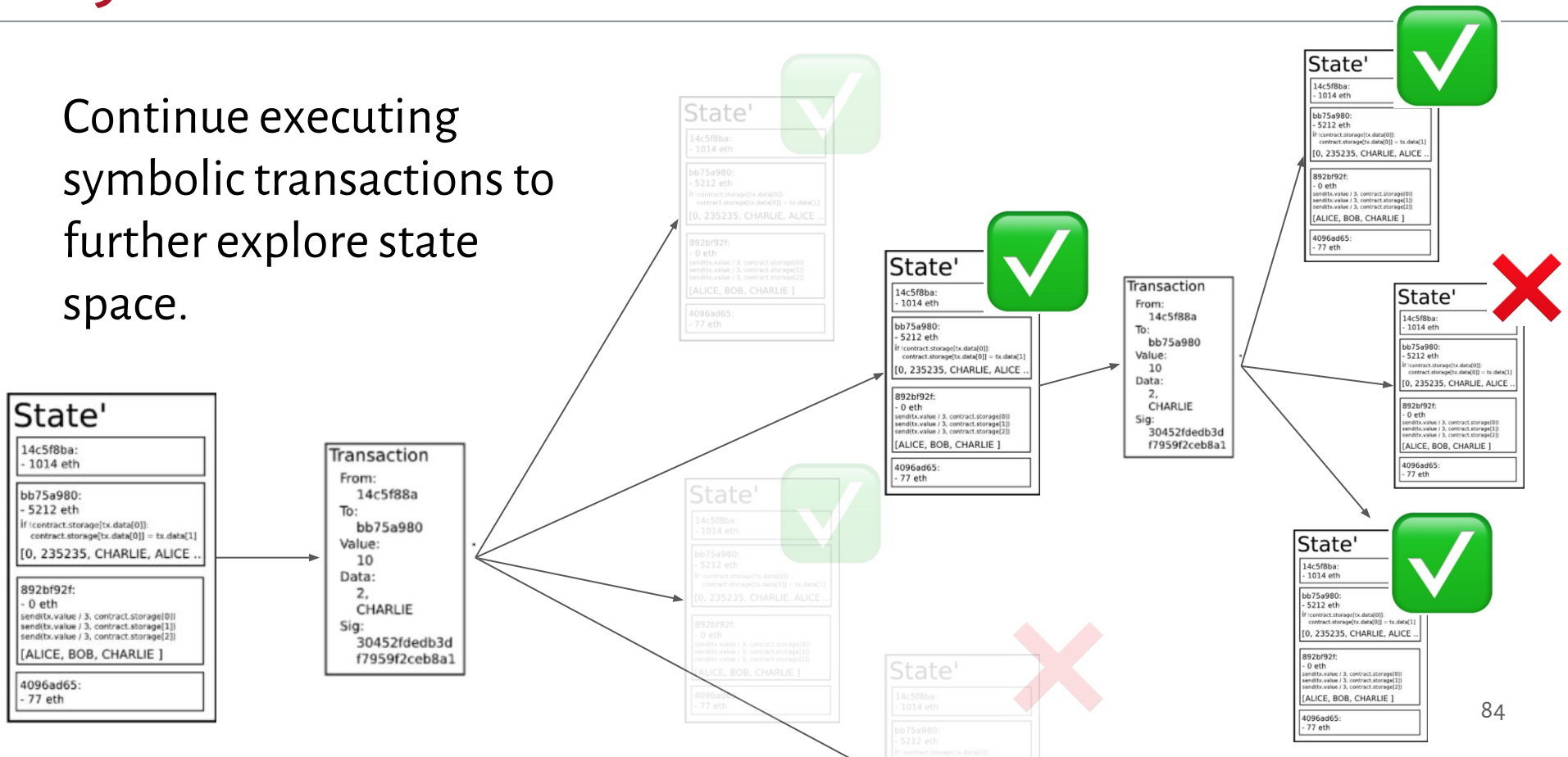
Symbolic Transactions

One symbolic transaction will produce **reverted** and **alive** states.



Symbolic Transactions

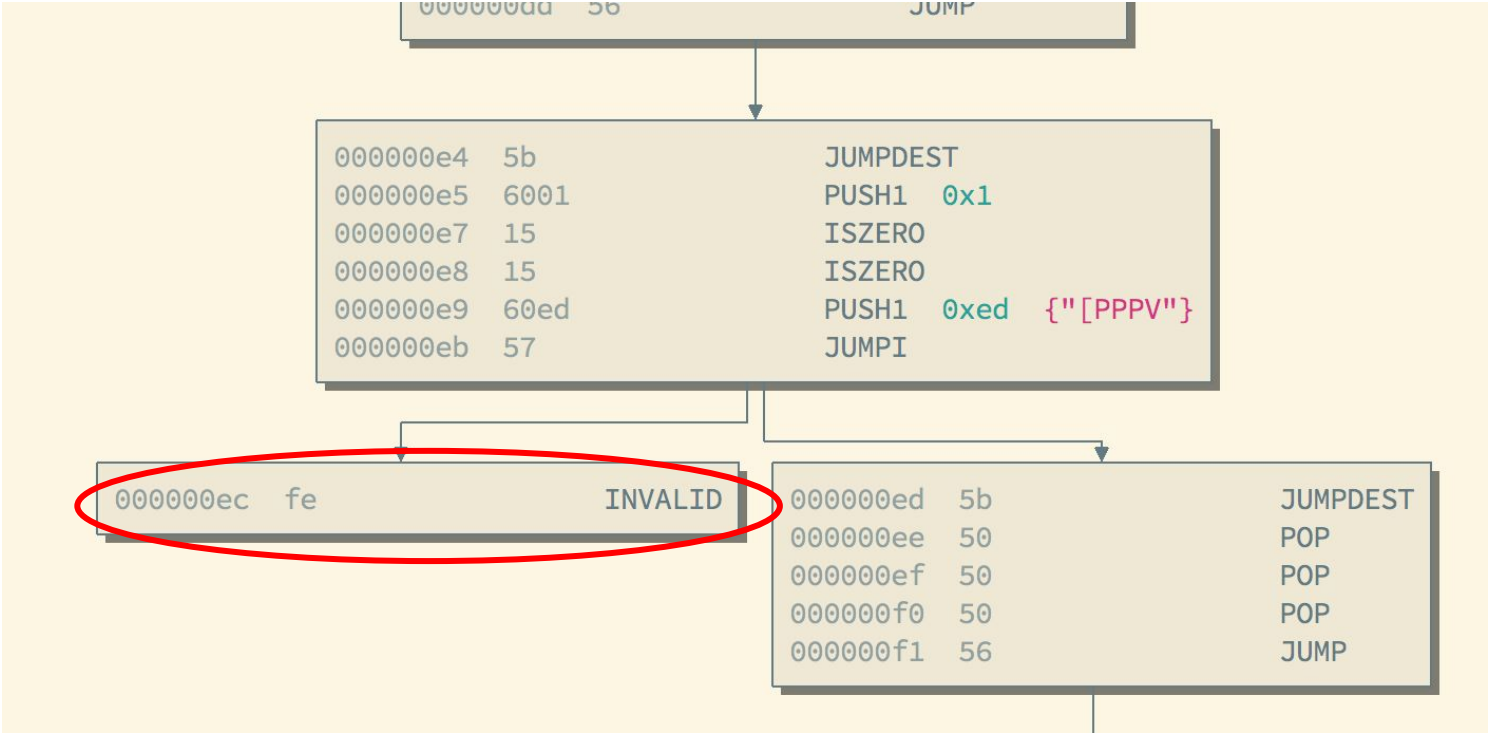
Continue executing symbolic transactions to further explore state space.



Applications of Ethereum symbolic execution

- Automatically check assertions
- Discover all functions in a binary contract
- Generate input transaction sequences

Check assertions



Function Discovery

Dispatch Stub

```
_dispatcher:
00000000  6060                PUSH1  0x60
00000002  6040                PUSH1  0x40
00000004  52                  MSTORE
00000005  6000                PUSH1  0x0
00000007  35                  CALLDATALOAD
00000008  7c0100000000000…    PUSH29 0x1000000000000000000000000000000000000000000000000000000000
00000026  90                  SWAP1
00000027  04                  DIV
00000028  63ffffffff          PUSH4  0xffffffff
0000002d  16                  AND
0000002e  80                  DUP1
0000002f  631822932c          PUSH4  0x1822932c
00000034  14                  EQ
00000035  6046                PUSH1  0x46
00000037  57                  JUMPI
```

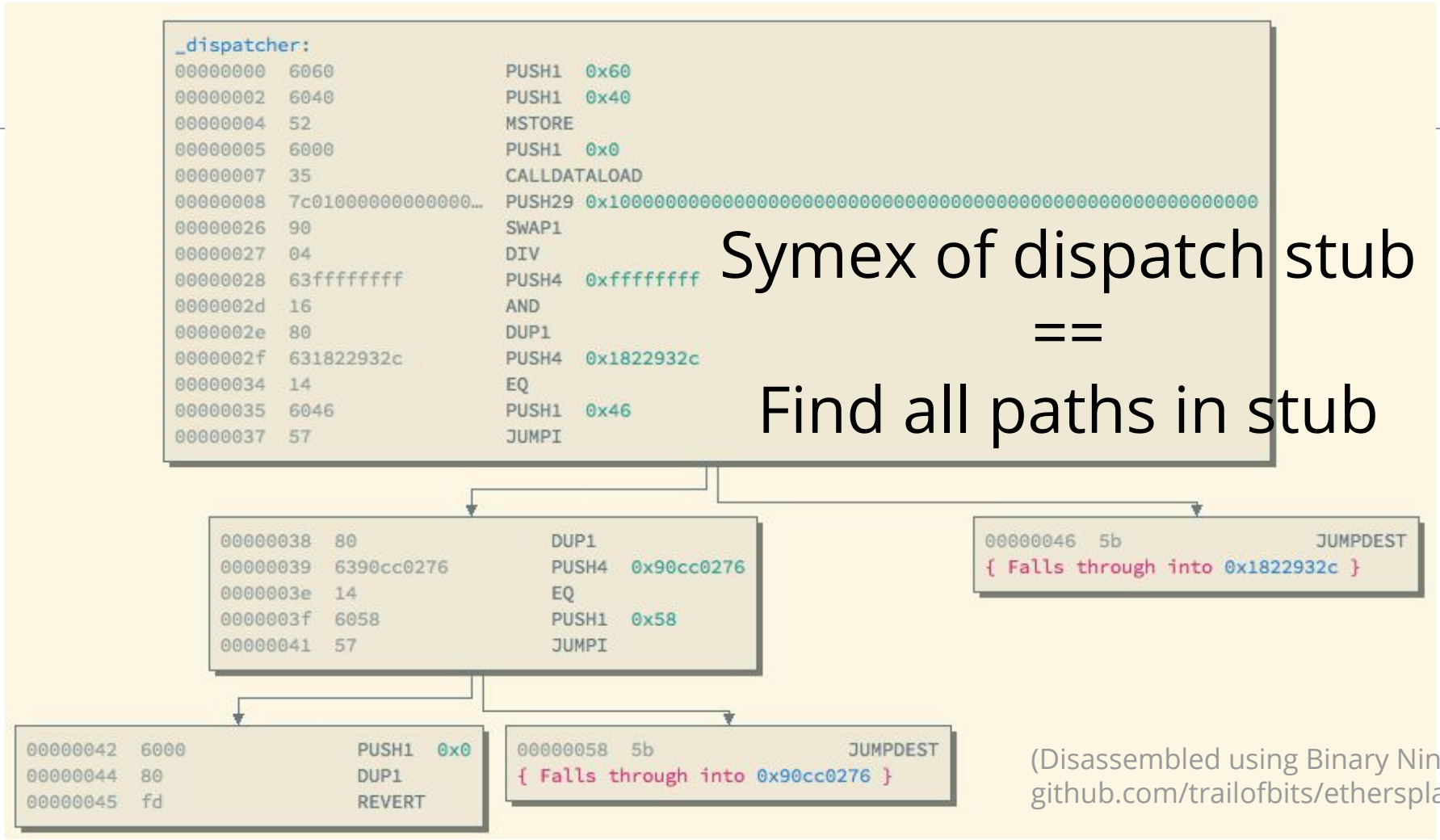
```
00000038  80                  DUP1
00000039  6390cc0276          PUSH4  0x90cc0276
0000003e  14                  EQ
0000003f  6058                PUSH1  0x58
00000041  57                  JUMPI
```

```
00000046  5b                  JUMPDEST
{ Falls through into 0x1822932c }
```

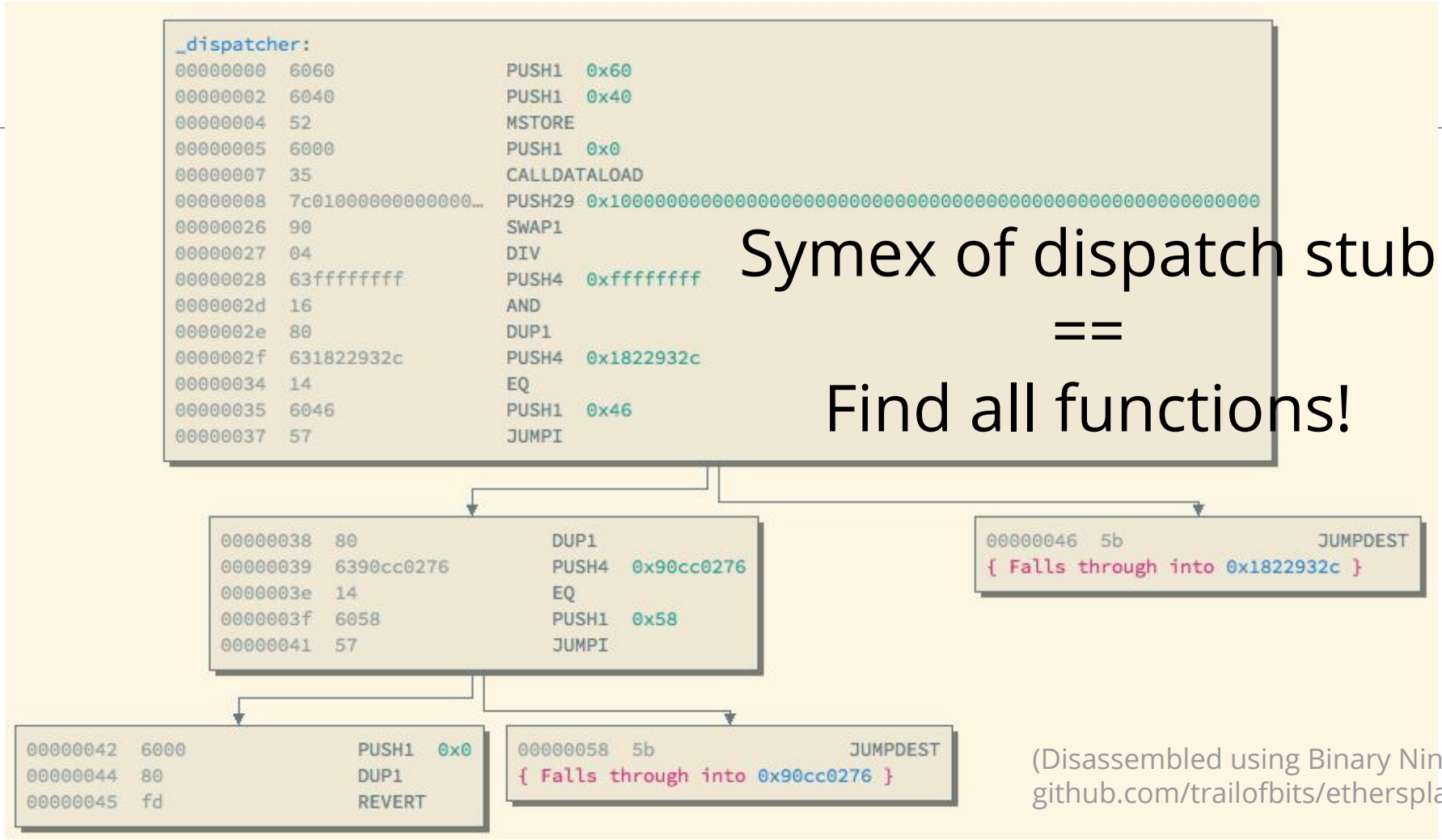
```
00000042  6000                PUSH1  0x0
00000044  80                  DUP1
00000045  fd                  REVERT
```

```
00000058  5b                  JUMPDEST
{ Falls through into 0x90cc0276 }
```

(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)



(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)


_dispatcher:
00000000 6060 PUSH1 0x60
00000002 6040 PUSH1 0x40
00000004 52 MSTORE
00000005 6000 PUSH1 0x0
00000007 35 CALLDATALOAD
00000008 7c0100000000000… PUSH29 0x1000000000000000000000000000000000000000000000000000000000
00000026 90 SWAP1
00000027 04 DIV
00000028 63ffffffff PUSH4 0xffffffff
0000002d 16 AND
0000002e 80 DUP1
0000002f 631822932c PUSH4 0x1822932c
00000034 14 EQ
00000035 6046 PUSH1 0x46
00000037 57 JUMPI
Symex of dispatch stub
==
Find all functions!
00000038 80 DUP1
00000039 6390cc0276 PUSH4 0x90cc0276
0000003e 14 EQ
0000003f 6058 PUSH1 0x58
00000041 57 JUMPI
00000046 5b JUMPDEST
{ Falls through into 0x1822932c }
00000042 6000 PUSH1 0x0
00000044 80 DUP1
00000045 fd REVERT
00000058 5b JUMPDEST
{ Falls through into 0x90cc0276 }
(Disassembled using Binary Ninja +
github.com/trailofbits/ethersplay)

Transaction generation

Will require 2 tx to reach overflow

```
1  pragma solidity ^0.4.15;
2
3  contract SymExExample {
4      uint did_init = 0;
5
6      // function id: 0x13371337
7      function test_me(int input) {
8          if (did_init == 0) {
9              did_init = 1;
10             return;
11         }
12
13         if (input < 42) {
14             // safe
15             return;
16         } else {
17             // overflow possibly!
18             int could_overflow = input + 1;
19         }
20
21     }
22 }
```

Transaction generation

Will require 2 tx to reach overflow

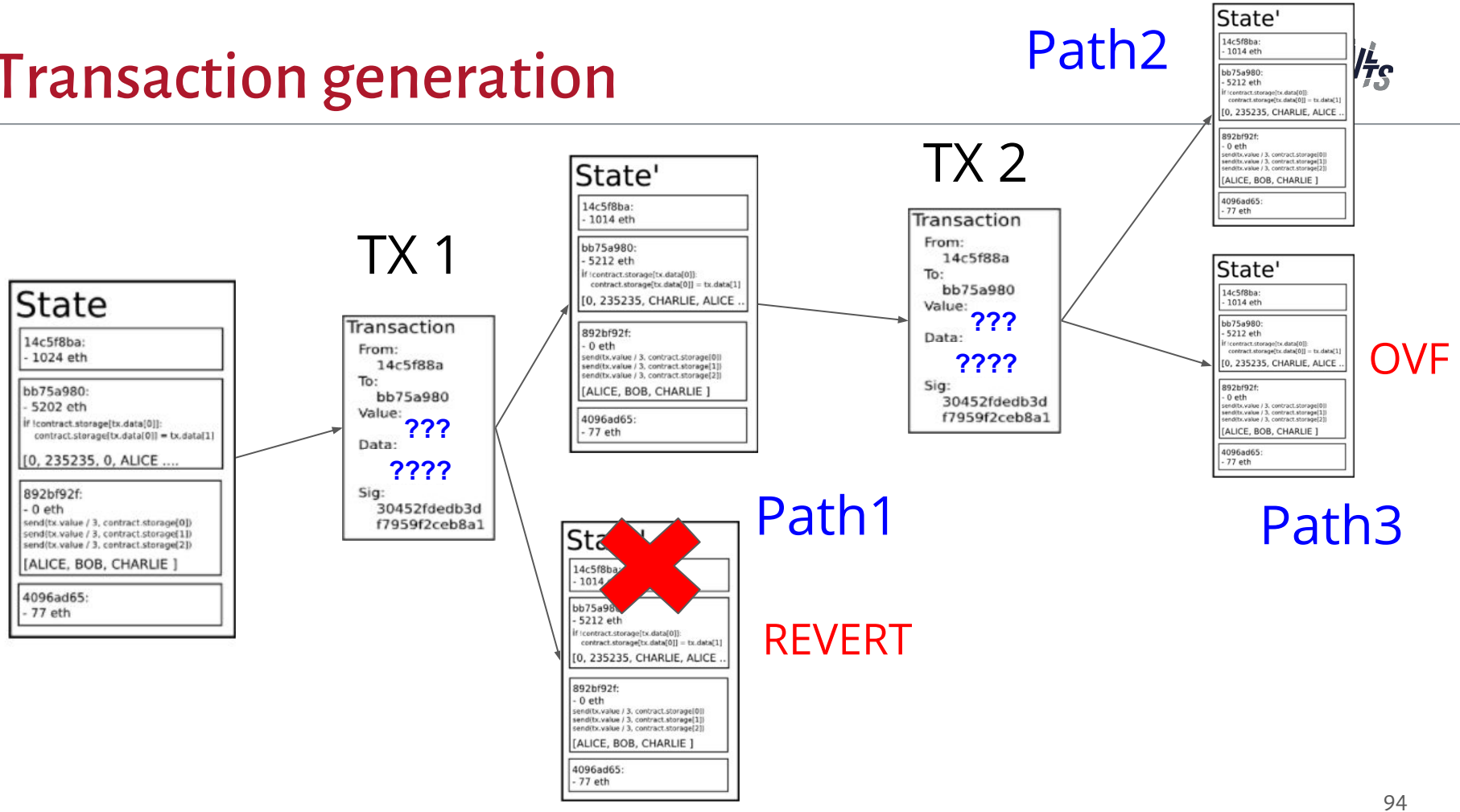
```
1  pragma solidity ^0.4.15;
2
3  contract SymExExample {
4      uint did_init = 0;
5
6      // function id: 0x13371337
7      function test_me(int input) {
8          if (did_init == 0) {
9              did_init = 1;
10             return;
11         }
12
13         if (input < 42) {
14             // safe
15             return;
16         } else {
17             // overflow possibly!
18             int could_overflow = input + 1;
19         }
20
21     }
22 }
```

Transaction generation

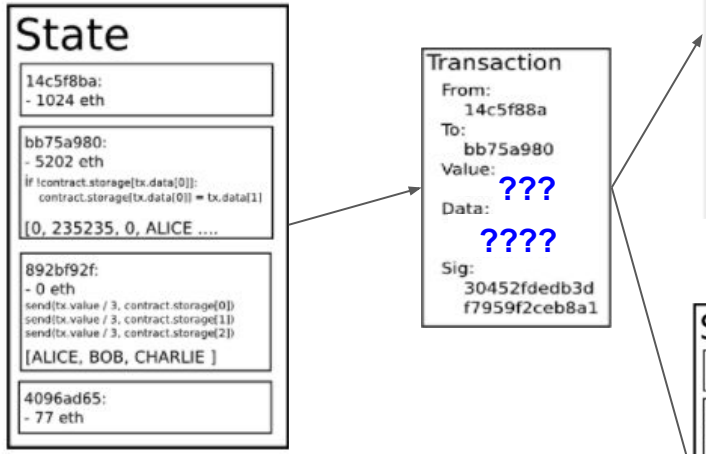
Will require 2 tx to reach overflow

```
1  pragma solidity ^0.4.15;
2
3  contract SymExExample {
4      uint did_init = 0;
5
6      // function id: 0x13371337
7      function test_me(int input) {
8          if (did_init == 0) {
9              did_init = 1;
10             return;
11         }
12
13         if (input < 42) {
14             // safe
15             return;
16         } else {
17             // overflow possibly!
18             int could_overflow = input + 1;
19         }
20
21     }
22 }
```

Transaction generation



Transaction generation



TX 1

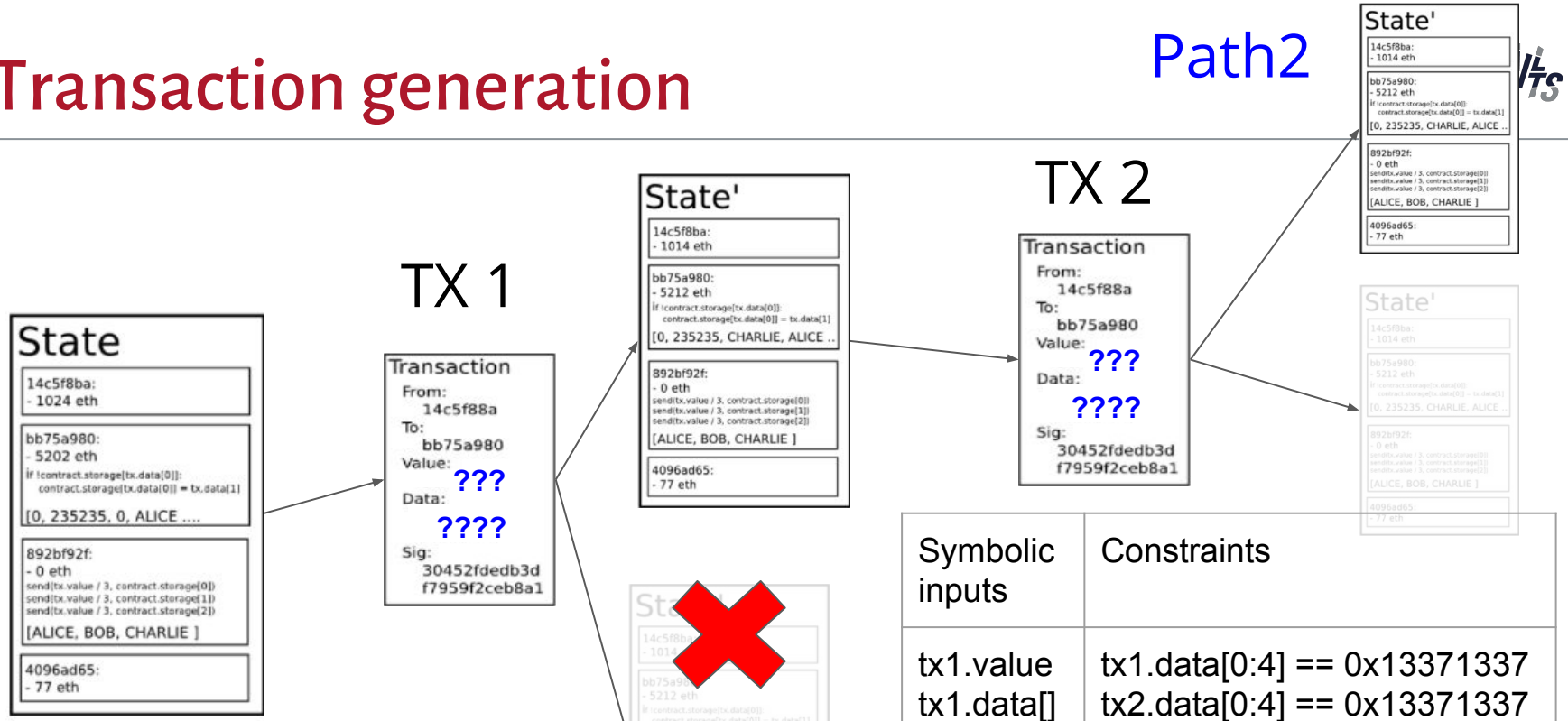
Path1

REVERT

Symbolic inputs	Constraints
tx1.value tx1.data[]	“tx1.data[0:4] != 0x13371337”

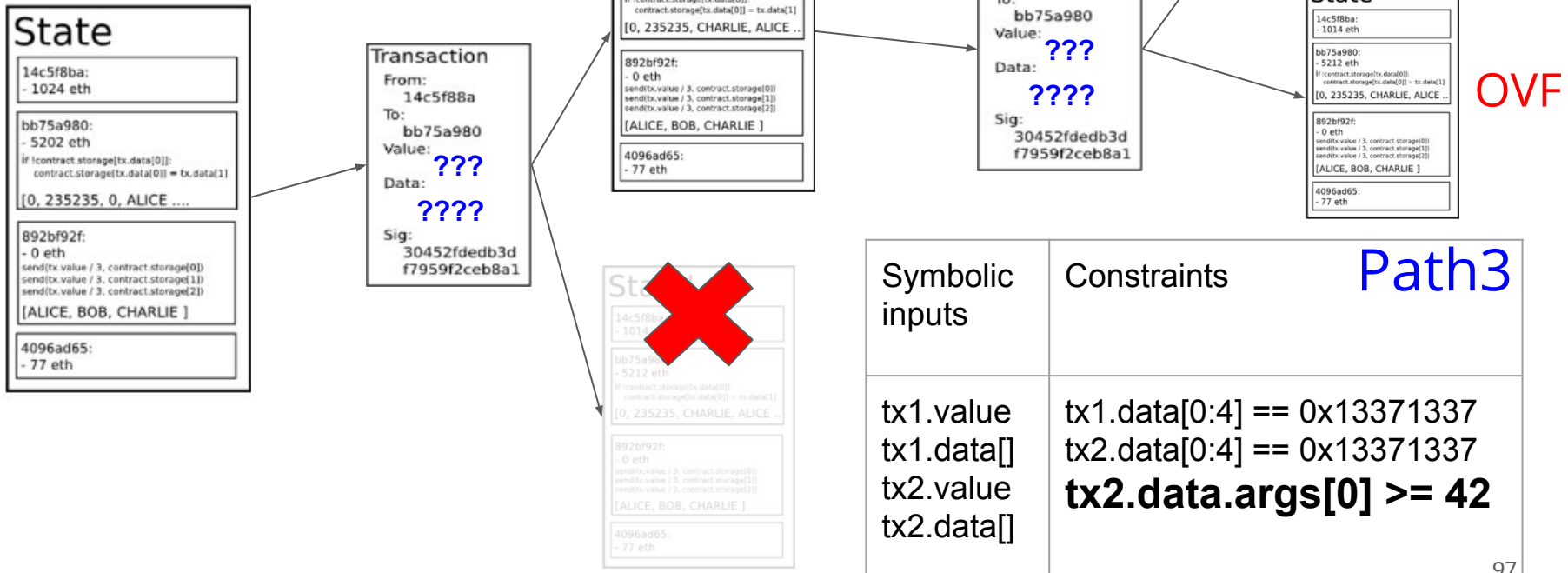
Transaction generation

Path2



Symbolic inputs	Constraints
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] < 42**

Transaction generation



Symbolic inputs	Constraints — Path3
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] >= 42**

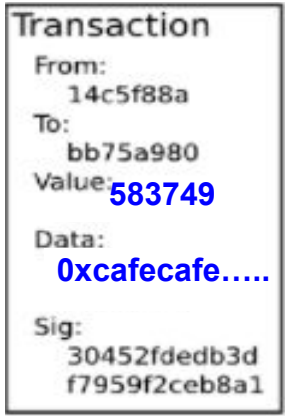
Transaction generation

Path1

Symbolic inputs	Constraints
tx1.value tx1.data[]	“tx1.data[0:4] != 0x13371337”

(constraint solver)

TX 1



Transaction generation

Path 2

Symbolic inputs	Constraints
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] < 42**

(constraint solver)

TX 1

Transaction
From:
14c5f88a
To:
bb75a980
Value:
323423
Data:
0x13371337.....
Sig:
30452fdedb3d
f7959f2ceb8a1

TX 2

Transaction
From:
14c5f88a
To:
bb75a980
Value:
454545
Data:
0x13371337...,
Args[0] = 41
Sig:
30452fdedb3d
f7959f2ceb8a1

Transaction generation

Path 3

Symbolic inputs	Constraints
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] >= 42**

(constraint solver)

TX 1

Transaction
From:
14c5f88a
To:
bb75a980
Value:
323423
Data:
0x13371337...
Sig:
30452fdedb3d
f7959f2ceb8a1

TX 2

Transaction
From:
14c5f88a
To:
bb75a980
Value:
545454
Data:
0x13371337……
Arg[0] = 43
Sig:
30452fdedb3d
f7959f2ceb8a1

Challenges

Challenges

- State explosion
- Symbolic hashing
- Dynamic arguments

State Explosion

```
1  int counter = 0, values = 0;
2  for ( i = 0 ; i < 100 ; i ++ ) {
3      if (input[i] == 'B') {
4          counter ++;
5          values += 2;
6      } }
7  if (counter == 75)      bug ();
```

"Enhancing Symbolic Execution with Veritesting", Avgerinos, et. al ICSE 2014

State Explosion

```
1  int counter = 0, values = 0;
2  for ( i = 0 ; i < 100 ; i ++ ) {  // loop
3      if (input[i] == 'B') {        // branch based on input
4          counter ++;
5          values += 2;
6      } }
7  if (counter == 75)     bug ();
```

"Enhancing Symbolic Execution with Veritesting", Avgerinos, et. al ICSE 2014

State Explosion

```
1  int counter = 0, values = 0;
2  for ( i = 0 ; i < 100 ; i ++ ) {  // loop
3      if (input[i] == 'B') {        // branch based on input
4          counter ++;
5          values += 2;
6      } }
7  if (counter == 75)      bug ();
```

Every loop iteration double # states

"Enhancing Symbolic Execution with Veritesting", Avgerinos, et. al ICSE 2014

State Explosion

```
1  int counter = 0, values = 0;
2  for ( i = 0 ; i < 100 ; i ++ ) {  // loop
3      if (input[i] == 'B') {        // branch based on input
4          counter ++;
5          values += 2;
6      } }
7  if (counter == 75)     bug ();
```

Every loop iteration double # states

Produces 2^{100} states

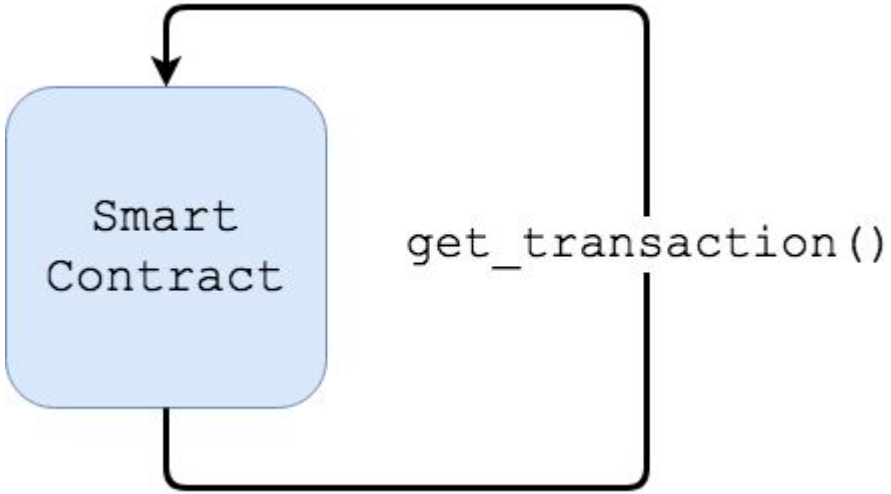
"Enhancing Symbolic Execution with Veritesting", Avgerinos, et. al ICSE 2014

State Explosion?

- Symex struggles to scale to large programs
- Smart contracts are usually very small! (100s LOC)
- State explosion will not be a problem?

Uh oh

Wait: there is an implicit loop for receiving input...



State Explosion?

```
for (;;) {
    tx = get_transaction();
    run_contract(tx);
}
```

State Explosion?

```
for (;;) {                      // loop
    tx = get_transaction();
    run_contract(tx);           // branch based on input
}
```

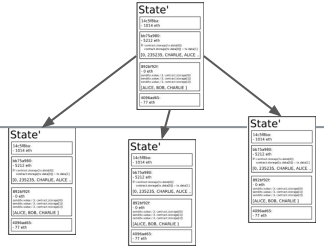
State Explosion?

Look familiar?

```
for (;;) {                      // loop
    tx = get_transaction();
    run_contract(tx);           // branch based on input
}
```

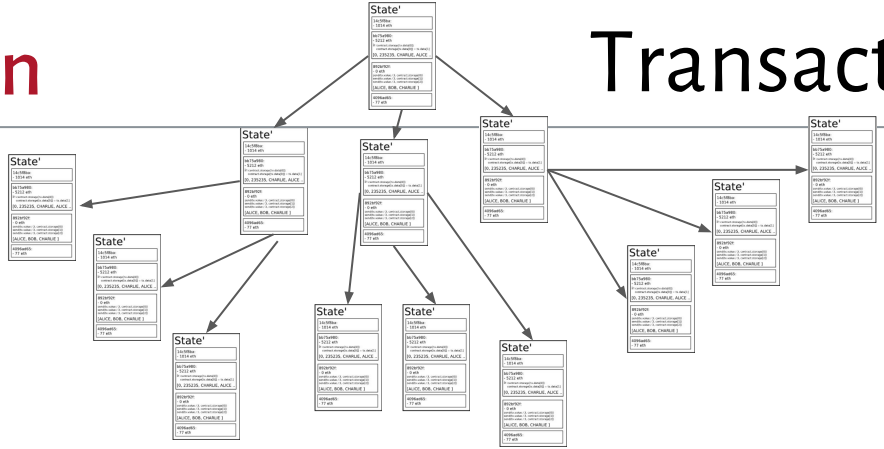
State Explosion

State Explosion



Transaction 1

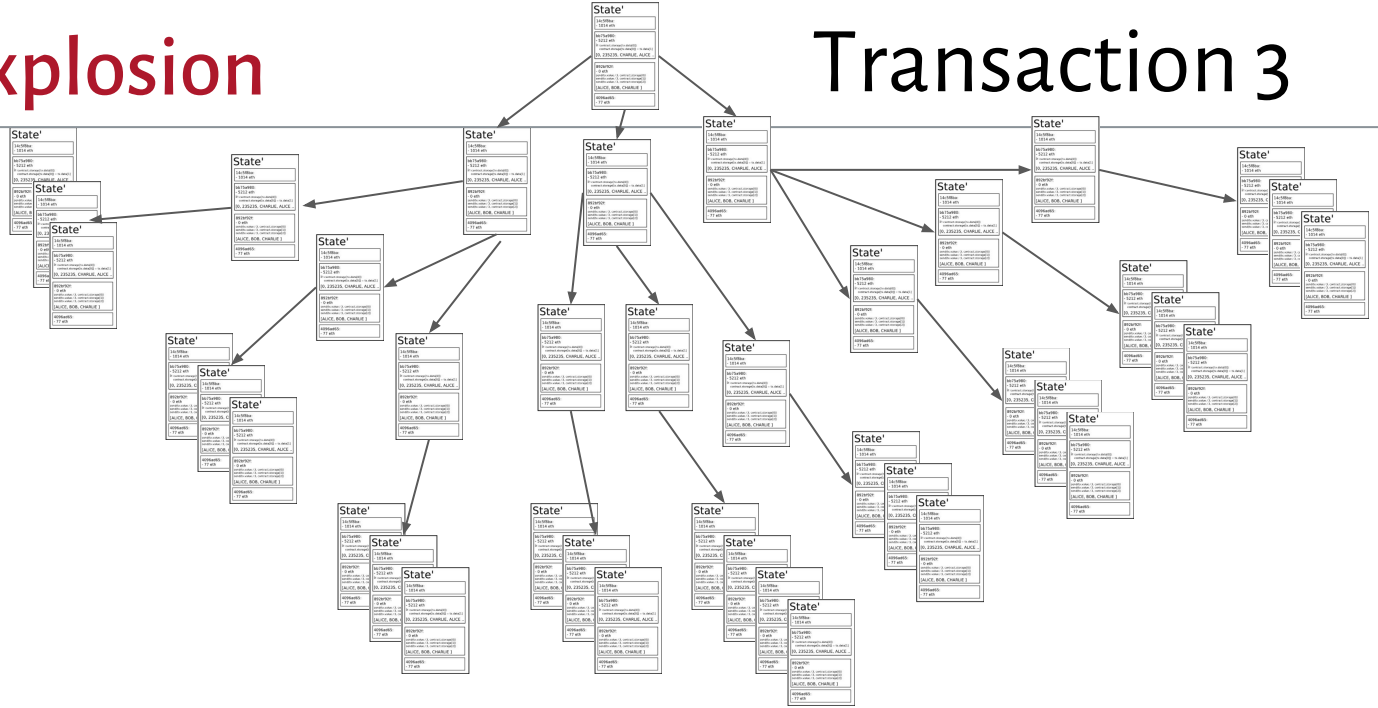
State Explosion



Transaction 2

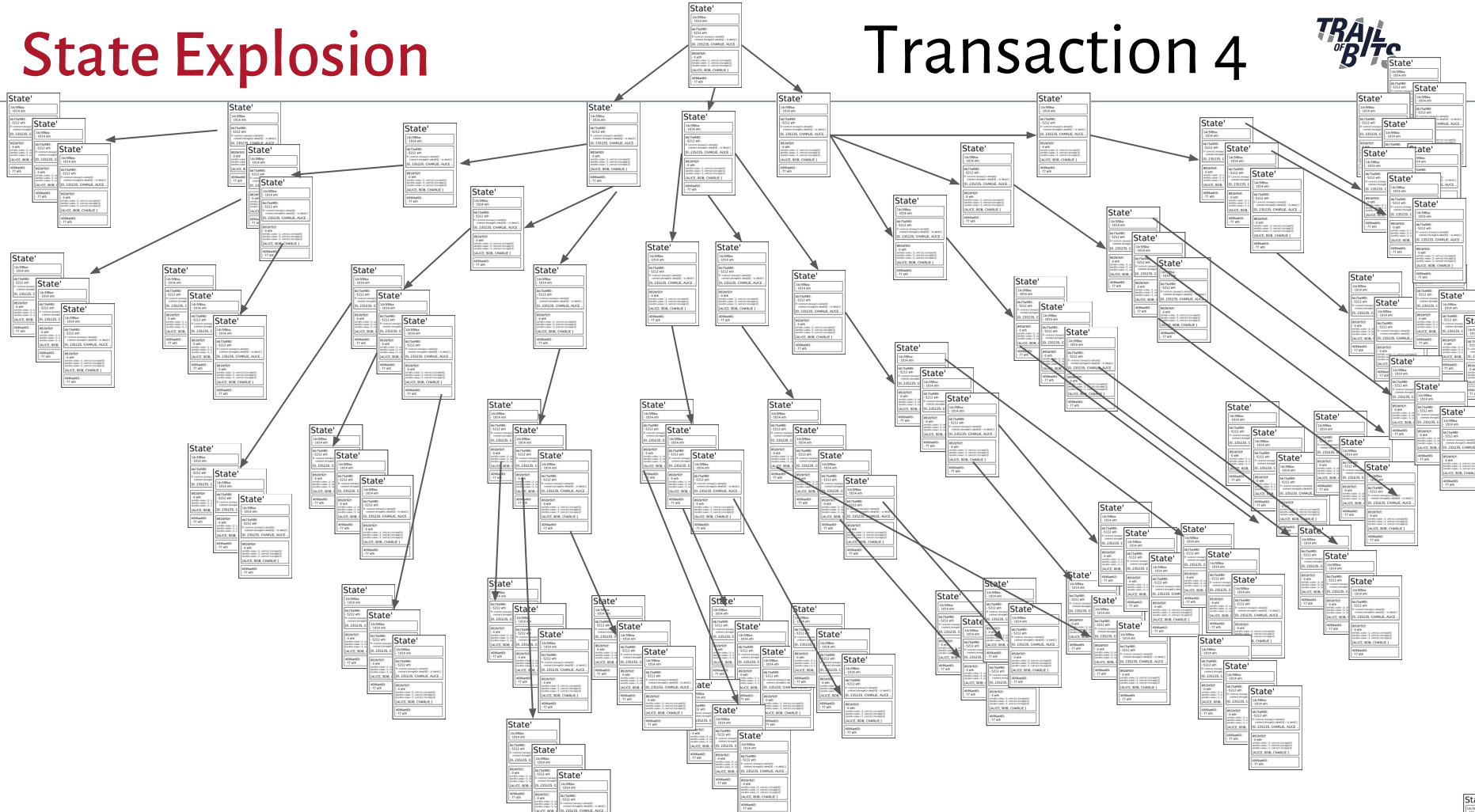
State Explosion

Transaction 3



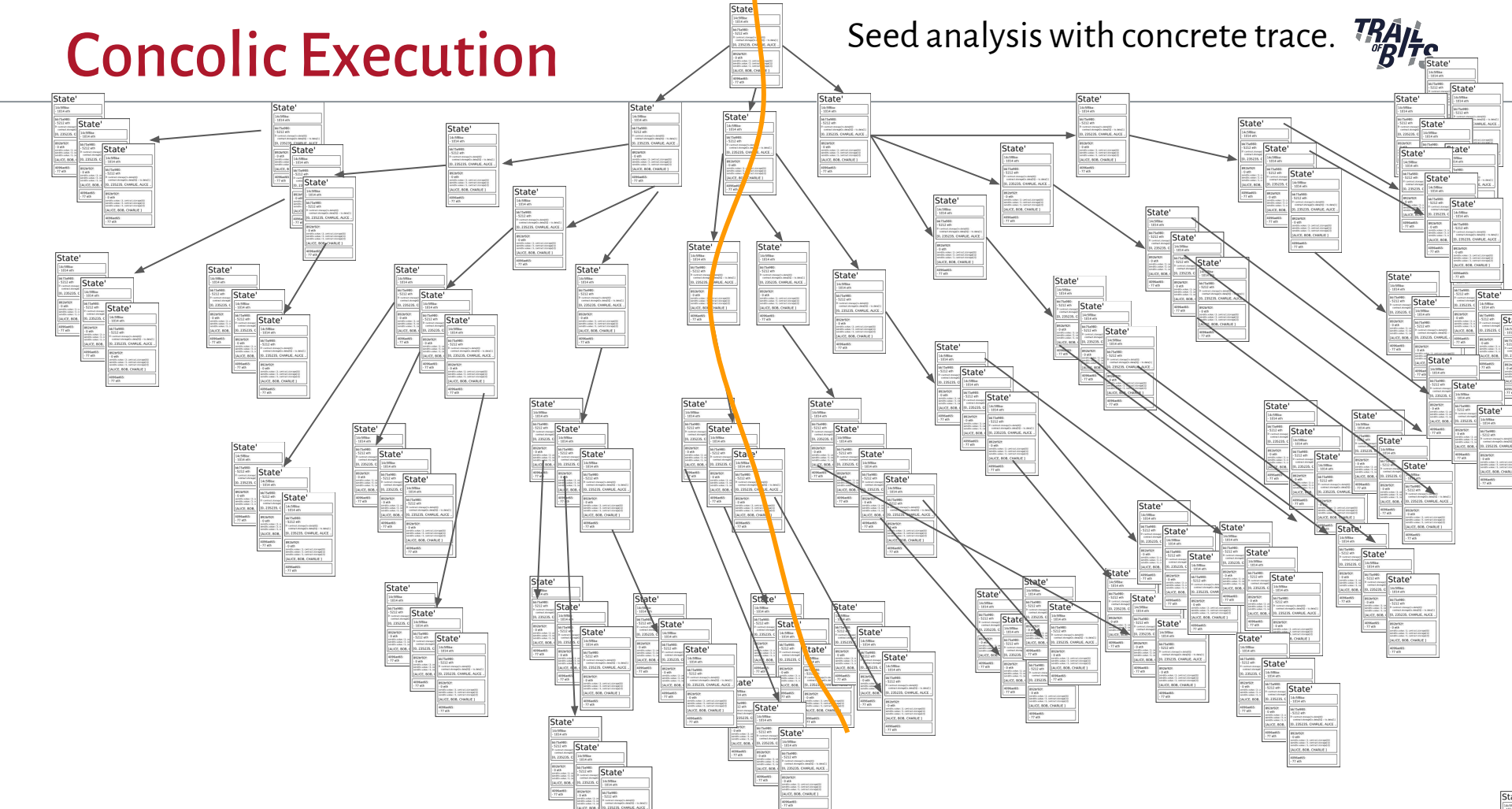
State Explosion

Transaction 4



Concolic Execution

Seed analysis with concrete trace.



Solidity: mappings

```
contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
```

Solidity: mappings

- All possible keys exist, and are default zero initialized
- Not iterable
- Implemented via **direct mapping onto contract storage** (256 bit virtual address space)
- Extensive hash use
- True O(1) access

```
balances[msg.sender] = newBalance;

store(sha3(msg.sender), newBalance);
```

(Simplified)

Solidity: mappings

- Accessing mappings with symbolic keys is common
- **Challenge: Hashing a symbolic value + symbolic storage index**
- Computing hash of symbol produces complex expression that is intentionally impossible to solve

```
balances[msg.sender] = newBalance;

store(sha3(symbolic), newBalance);
```

(Simplified)

Solidity: mappings

```
hash(symbol) == 0xfe67febfe6
```

If a solver could solve this, it would be reversing the hash!

Solidity: mappings

Record concrete hashes..

key	hash
userA	0x34b34b34b..

Solidity: mappings

Record concrete hashes..

key	hash
userA	0x34b34b34b..
userB	0x56c56c56c56c

Solidity: mappings

Rather than computing the symbolic hash...

key	hash
userA	0x34b34b34b..
userB	0x56c56c56c56c

Solidity: mappings

Constrain symbol using known hashes

`sha3(symbolic)` →

```
ITE(symbol=="userA",
    0x34b34b34b..,
    ITE(symbol=="userB",
        0x56c56c56c56c...,
        ITE(symbol=="unknown"
            0x89e89e89e89...,
            0)))
```

key	hash
userA	0x34b34b34b..
userB	0x56c56c56c56c

Solidity: mappings

Allow analysis to continue with solvable constraints

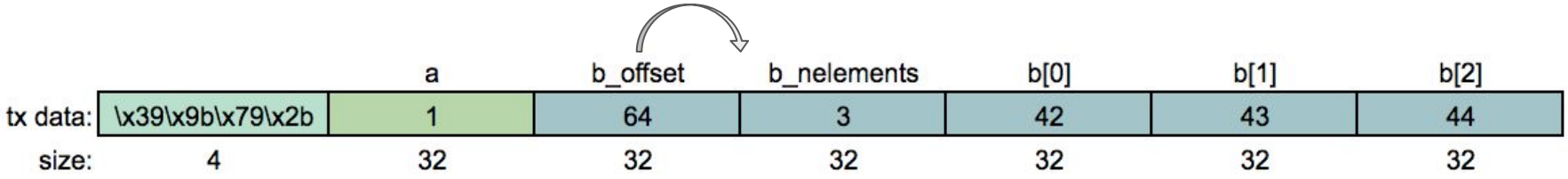
`sha3(symbolic)` →

```
ITE(symbol=="userA",
    0x34b34b34b..,
    ITE(symbol=="userB",
        0x56c56c56c56c...,
        ITE(symbol=="unknown"
            0x89e89e89e89...,
            0)))
```

key	hash
userA	0x34b34b34b..
userB	0x56c56c56c56c

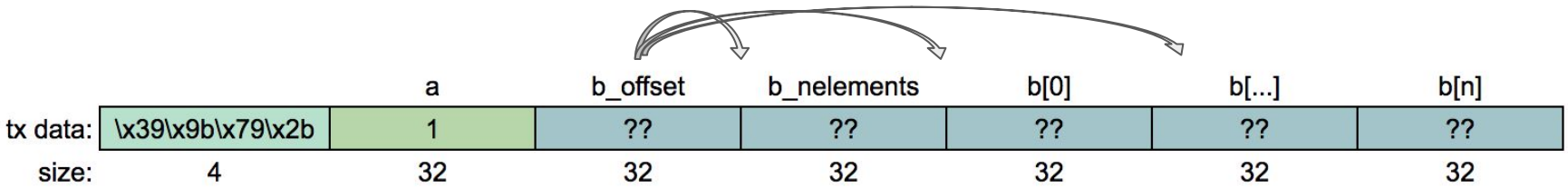
Solidity: dynamic arguments

- Functions can receive variable length data
- Transaction data becomes complex, with various offset and size fields
- Leads to symbolic indexing & memcpy operations



Solidity: dynamic arguments

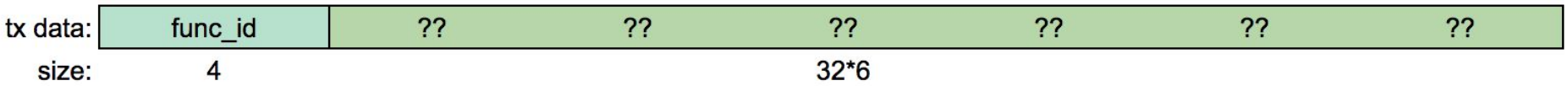
- Functions can receive variable length data
- Transaction data becomes complex, with various offset and size fields
- Leads to **symbolic indexing** & memcpy operations



Solidity: dynamic arguments

- Workaround: **aggressively concretize offset & nelements fields**

```
my_function(uint256[] a, uint256[] b);
```

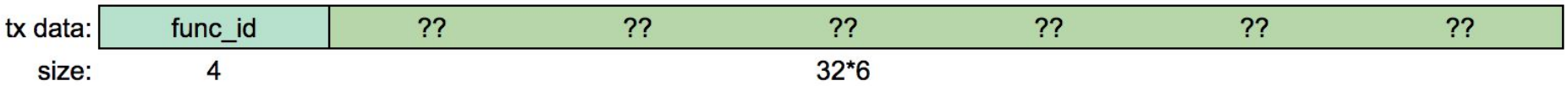


Solidity: dynamic arguments

- Workaround: **aggressively concretize offset & nelements fields**

```
(total_space - metadata_space) / (num_dyn_args) = space_for_each
            ((32 * 6) - ((32 * 2) * 2)) / (2) = 32
```

```
my_function(uint256[] a, uint256[] b);
```

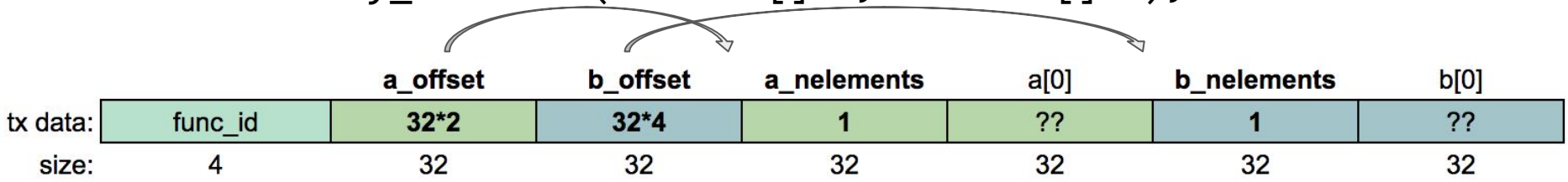


Solidity: dynamic arguments

- Workaround: **aggressively concretize offset & nelements fields**

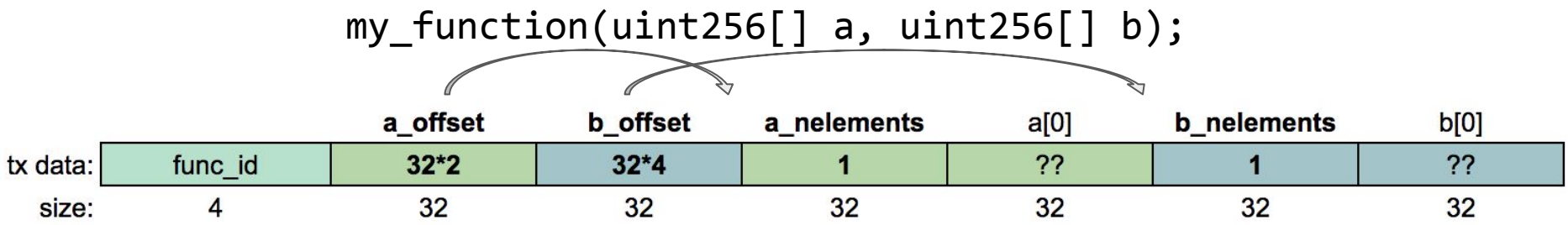
```
(total_space - metadata_space) / (num_dyn_args) = space_for_each
            ((32 * 6) - (32 * 2 * 2)) / (2) = 32
```

```
my_function(uint256[] a, uint256[] b);
```



Solidity: dynamic arguments

- Workaround: **aggressively concretize offset & nelements fields**
- Limitation: State space exploration artificially limited
  - E.g. Will miss branches requiring `a.length > 1`



Other Challenges

- Complete symbolic environment model must support inter contract calls
- Gas/Symbolic Gas

Implementation

- Implemented within Manticore project
- Open source symbolic execution tool
- Ethereum module: ~4k lines of Python
- Python API
  - Customize start execution state
  - Launch symbolic transactions
  - Instrument execution
  - Inspect discovered states
  - Submit solver queries

github.com/trailofbits/manticore

```
pip install manticore
```

Evaluation

- Used by smart contract auditors on 3+ engagements to date
- Also deployed within client test infrastructure
- Develop suite of Manticore scripts for verifying specific sets of functionality
- General pattern:
  - Initialize contract/blockchain state
  - Launch n symbolic transactions
  - Assert certain invariants in all discovered states

Demos

Summary

- Ethereum symbolic execution is possible & useful!
- Many interesting & unique challenges exist
- Significant potential to have a large impact
- Manticore is an available implementation

Special thanks to Felipe Manzano!

Thanks!

Mark Mossberg

Security Engineer @ Trail of Bits

@markmossberg

mark@trailofbits.com

github.com/trailofbits/manticore

```
pip install manticore
```


TRAIL
OF
BITS