Applying Symbolic Execution to Blockchain Applications

Mark Mossberg

High Confidence Software & Systems (HCSS) 2018

Blockchain

Blockchain

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

(Wikipedia)

Blockchains have useful properties

- Decentralized
- Resilient
- Verifiable
- Transparent
- Immutable

Ethereum

Ethereum

Blockchain-based, decentralized
computation platform

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

Parity Team Publishes Postmortem on $160 Million Ether Freeze

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

(1975)

James C. King
IBM Thomas J. Watson Research Center

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	Michael Y. Levin	David Molnar*
Microsoft (Research)	Microsoft (CSE)	UC Berkeley
pg@microsoft.com	mlevin@microsoft.com	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: A Binary Analysis Platform

David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA, USA

(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, dbrumley*}@*cmu.edu*

(MAYHEM, 2012)

CYBER GRAND_CHALLENGE

(2016)

Concrete Execution

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

Symbolic Execution (SE)

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

Symbolic Execution (SE)

Symbolic

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

Concrete

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

Execution could go either way!

```
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

Path *Constraints*

```
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] ∪ [43, INT_MAX]
}
```

Symbolic execution example

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

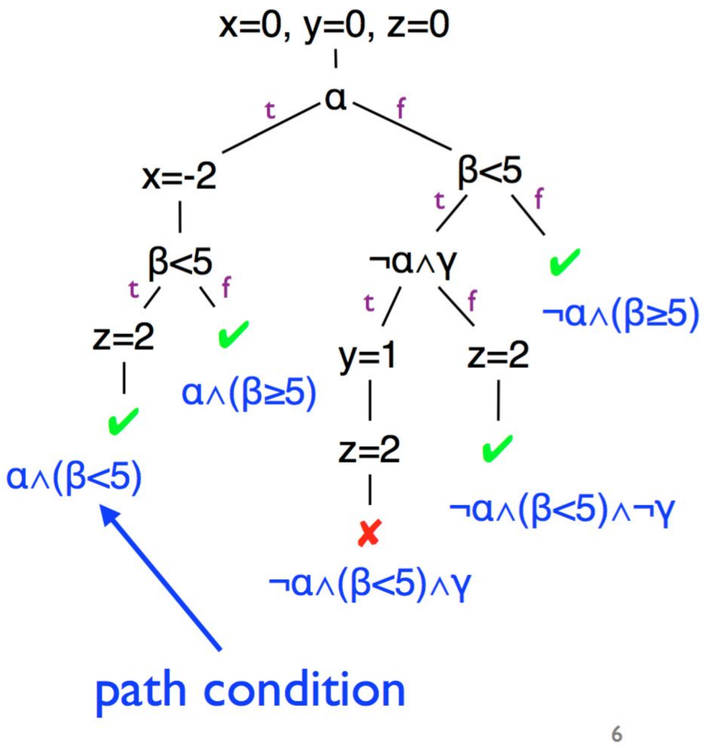


Image from Michael Hicks, University of Maryland

Constraint Solvers

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

SAT, α=false, β=3, γ=true

Constraint Solvers

$\neg a \land a$

z3

Microsoft Research

UNSAT

Using Symbolic Execution, we can:

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

Ethereum

Ethereum

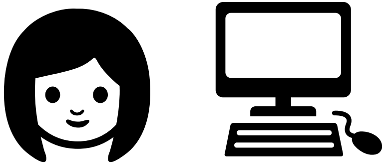
- Decentralized computation platform
- Includes a cryptocurrency implemented using a blockchain (ether)

"Most cryptocurrency barely has anything to do with serious cryptography," Matthew Green, a renowned computer scientist who studies cryptography, told me via email. "Aside from the trivial use of digital signatures and hash functions, it's a stupid name."

Ethereum

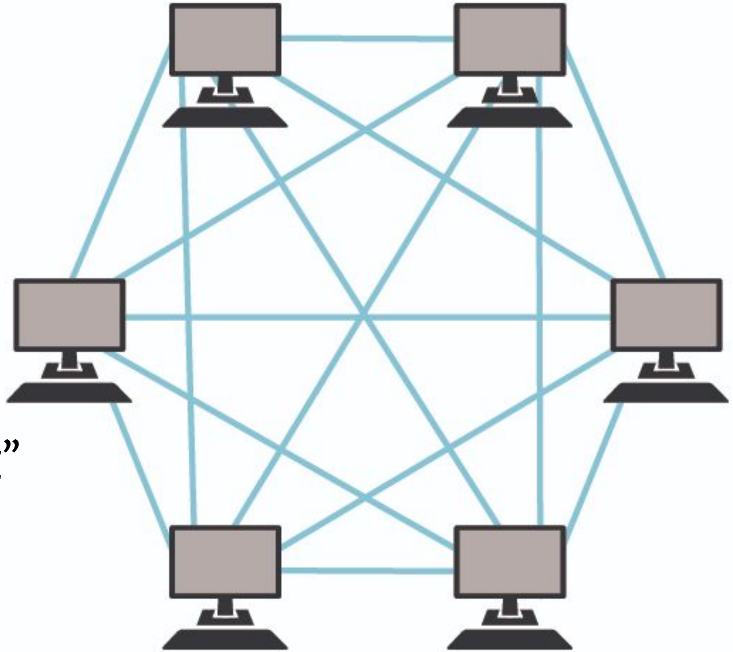
- Decentralized computation platform
- Includes a “cryptocurrency” implemented using a blockchain (ether)
- Includes a **virtual machine**

Ethereum entities



“External Account”

- balance



“Contract Account”

- balance
- code

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

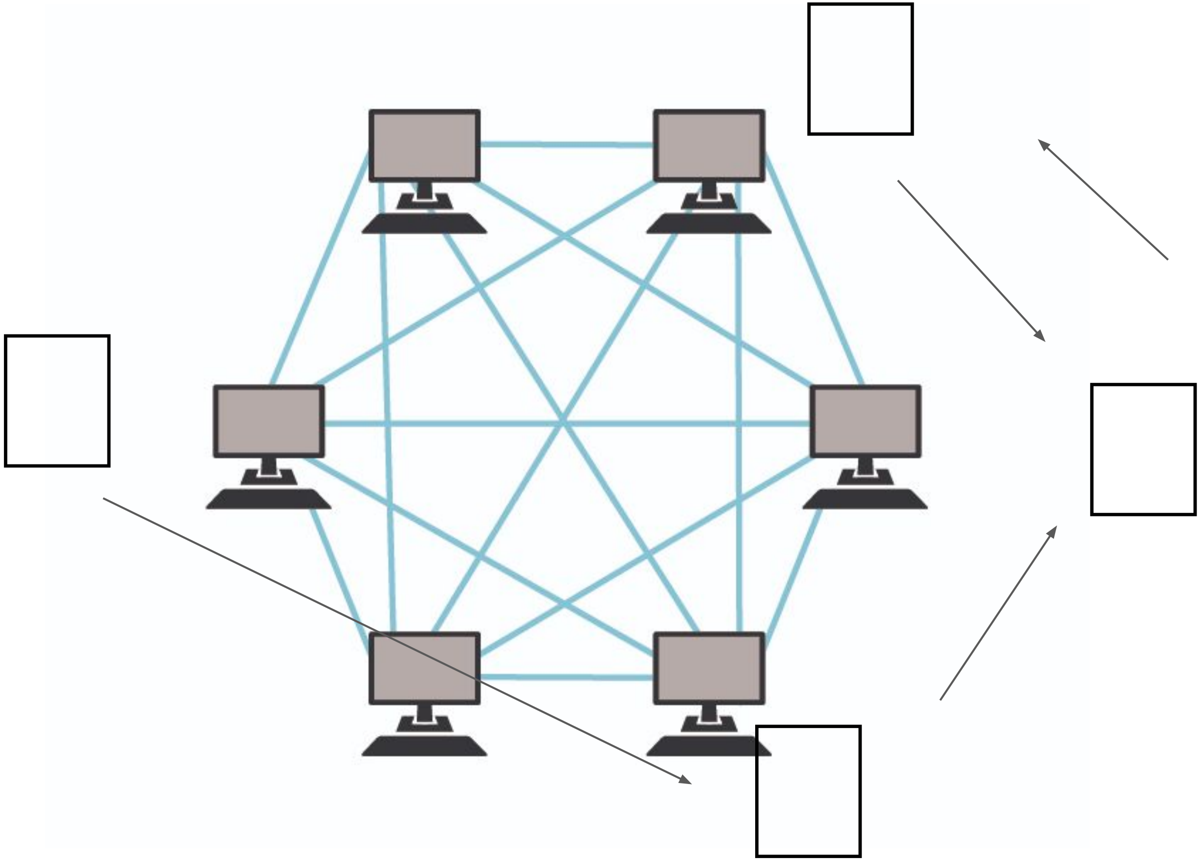
```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* 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

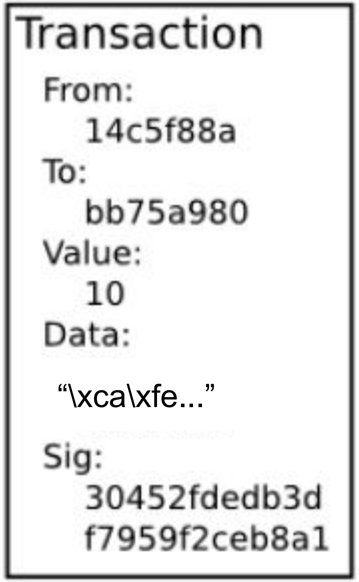
```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* 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;
    }
}
```

Contracts can call other contracts



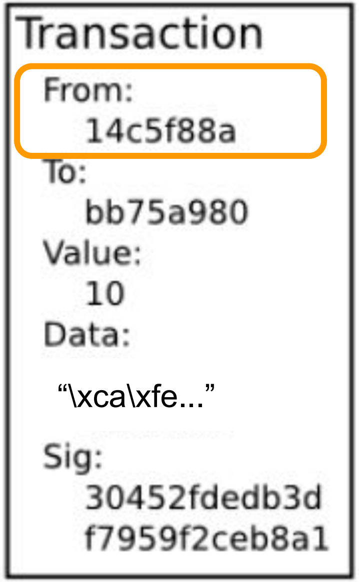
Ethereum transactions

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



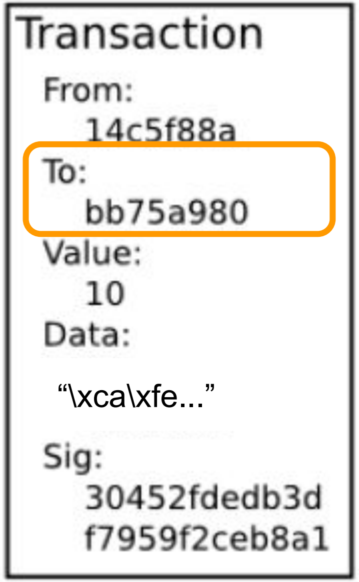
Ethereum transactions

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



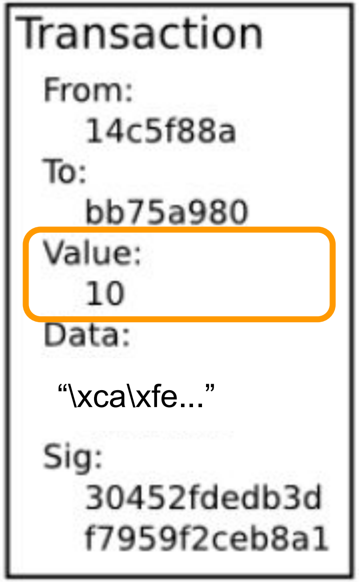
Ethereum transactions

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



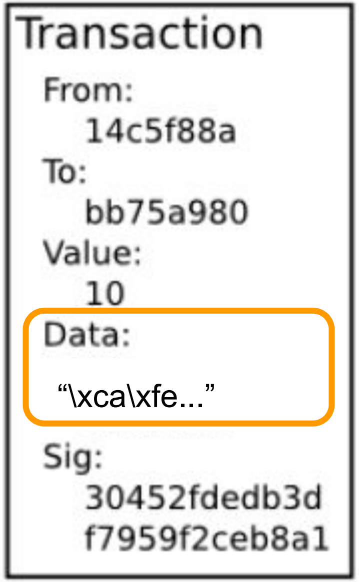
Ethereum transactions

- 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 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

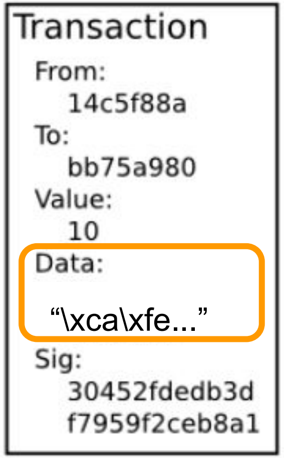
EVM: Memory Regions

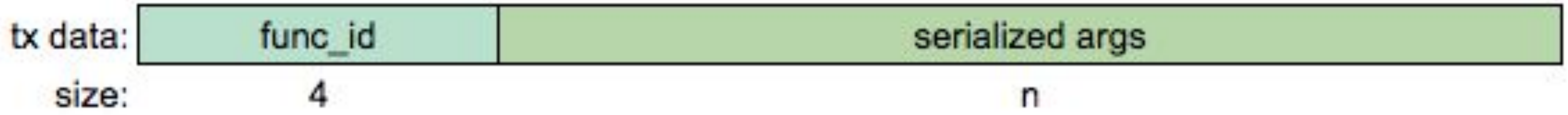
Several types of address spaces accessible:

- **Storage**: Persistent data store, 256 bit addressable space
- **Memory**: Volatile data store for intermediate storage, expands
- **Calldata**: Region containing transaction data buffer
- **Stack**: 1024 words capacity

Ethereum ABI

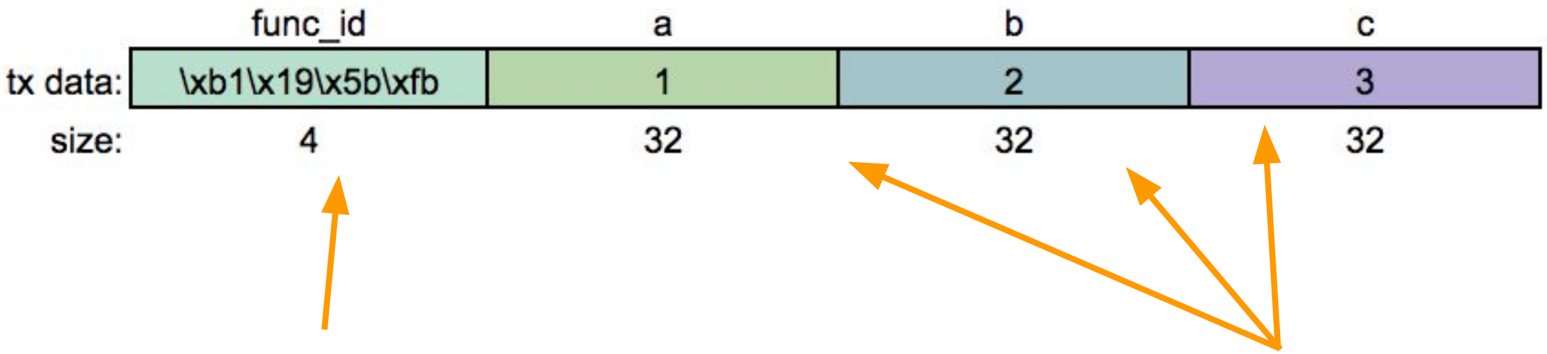
- Specifies how function call information is serialized
  - Function id being called
  - Arguments passed to function
- Serialized call info passed via transaction data field
- ABI spec required to call contract





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]`

Big Endian

Ethereum ABI Example: Dynamic Types

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

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

		a	b_offset	b_nelements	b[0]	b[1]	b[2]
tx data:	\x39\x9b\x79\x2b	1	64	3	42	43	44
size:	4	32	32	32	32	32	32

EVM Bytecode

EVM Runtime Bytecode Format

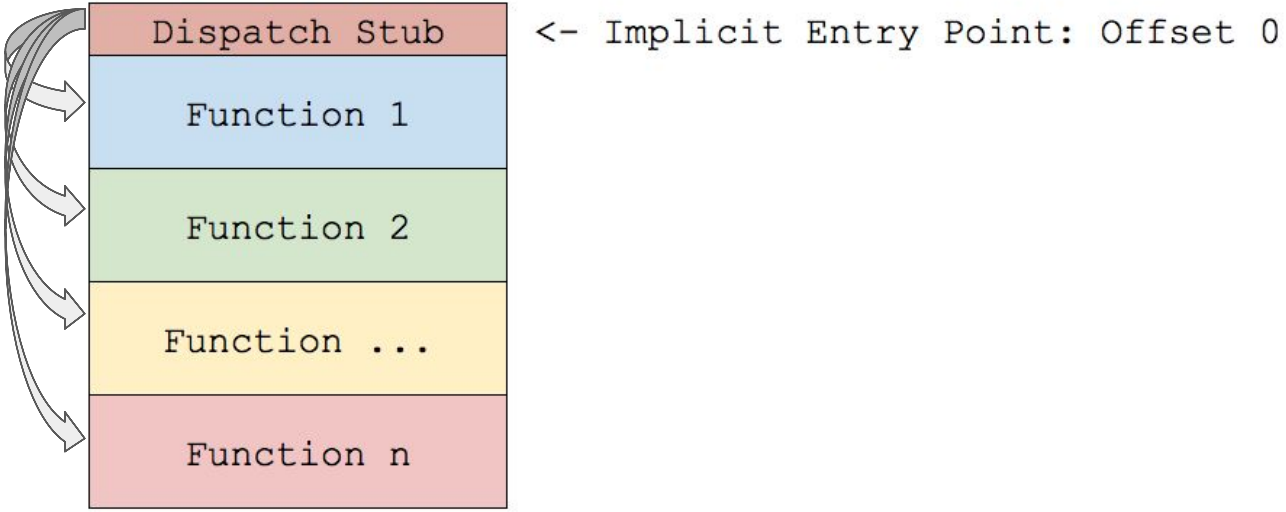
```
6060 6040 5260 0035
0000 0000 0000 0000
0000 0000 0000 9004
1822 932c 1460 4657
5857 6000 80fd 5b34
5b60 5660 6a56 5b00
80fd 5b60 6860 7356
5050 565b 6000 6001
7a7a 7230 5820 b407
638a 855f 6c0a 7117
9634 fd27 159a 0029
```

Entry point?

*as of solc 0.4.17

EVM Bytecode

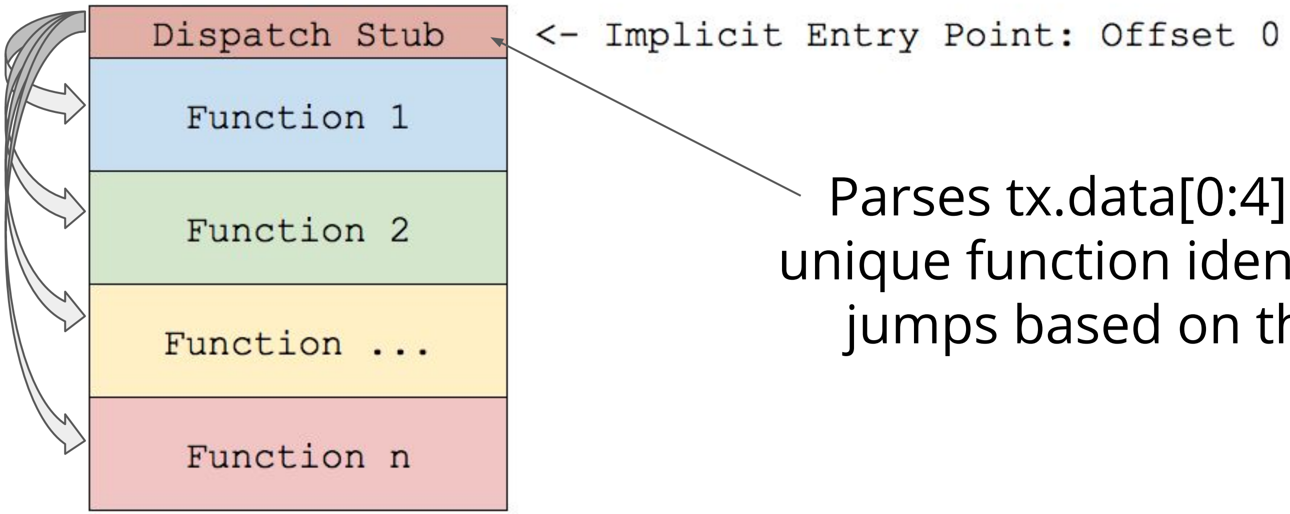
EVM Runtime Bytecode Format



*as of solc 0.4.17

EVM Bytecode

EVM Runtime Bytecode Format

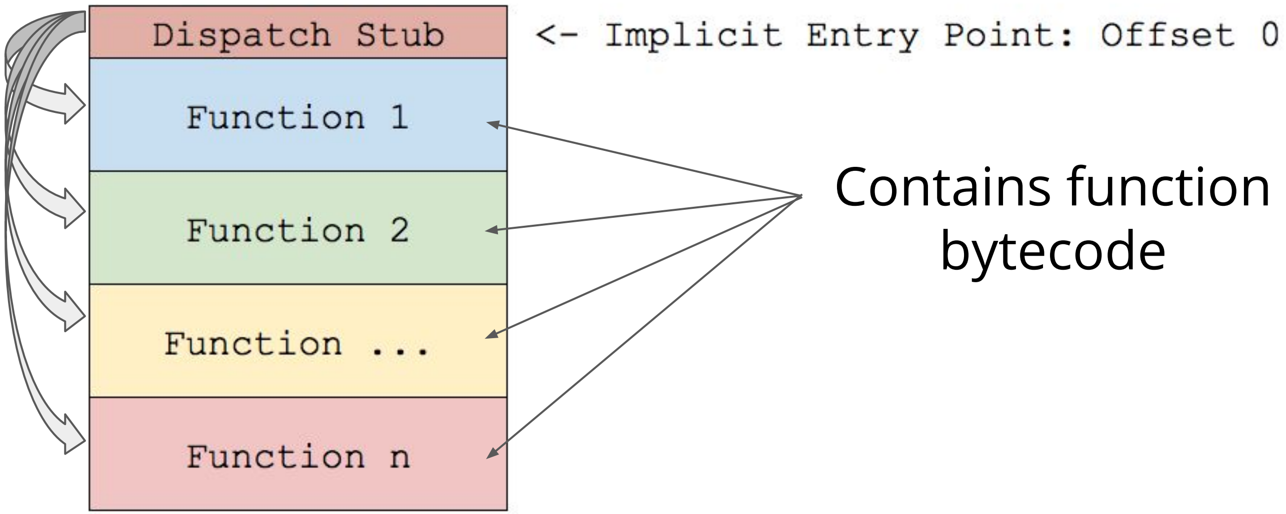


Parses tx.data[0:4] for unique function identifier, jumps based on that

*as of solc 0.4.17

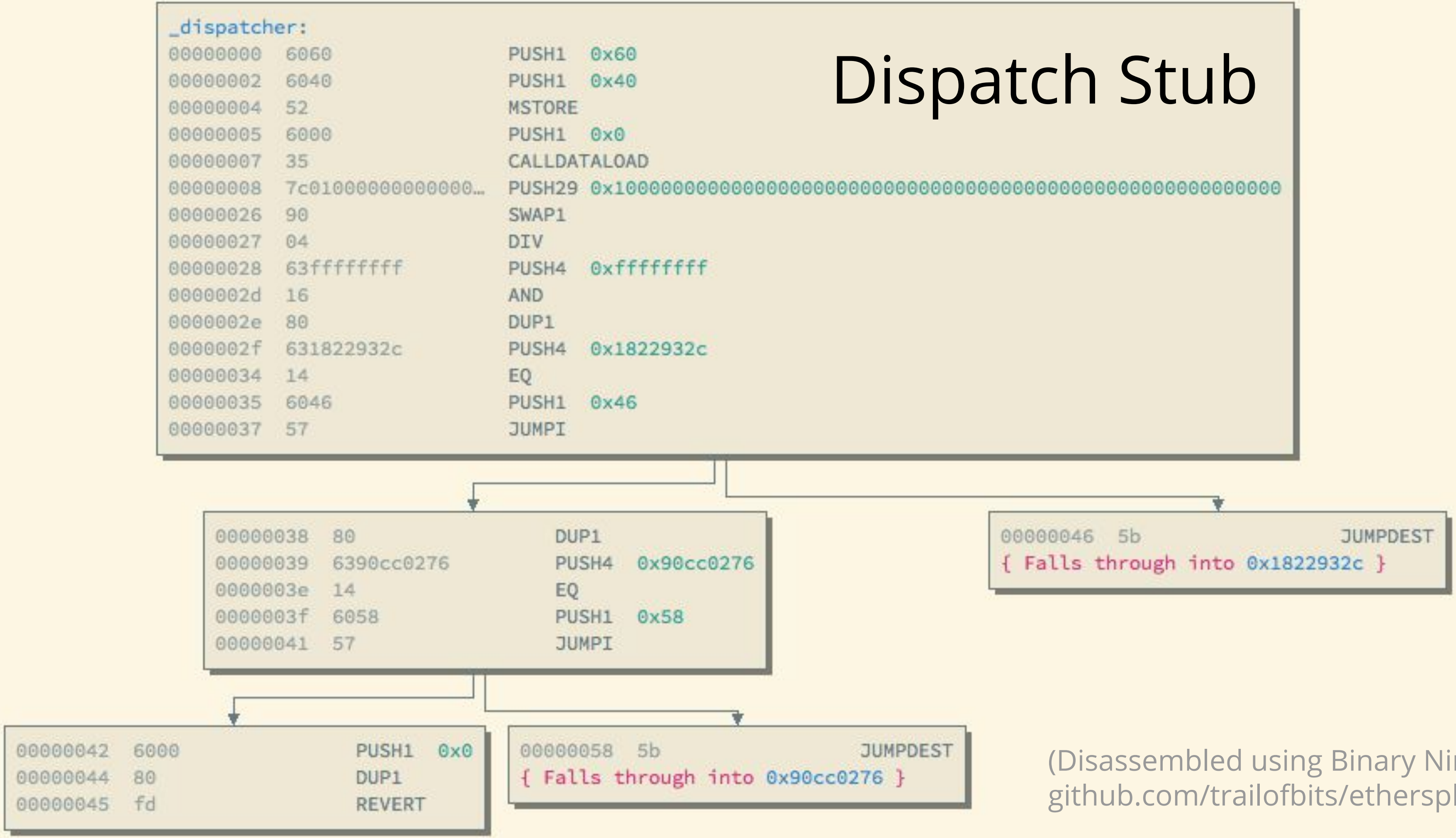
EVM Bytecode

EVM Runtime Bytecode Format



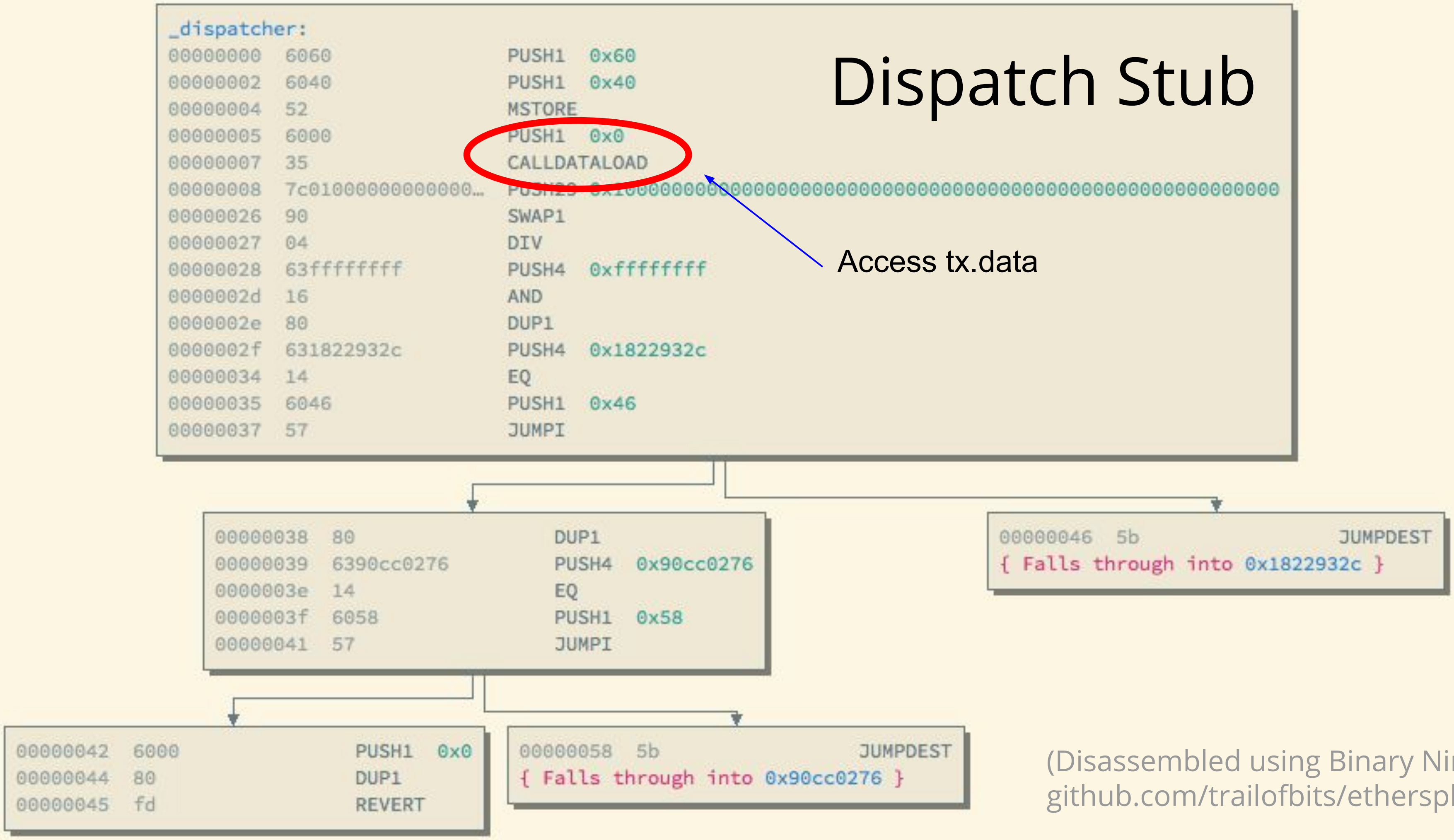
*as of solc 0.4.17

Dispatch Stub



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

Dispatch Stub



(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  7c01000000000000…  PUSH29 0x100000000000000000000000000000000000000000000000000000000
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
```

Function 1 identifier

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

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

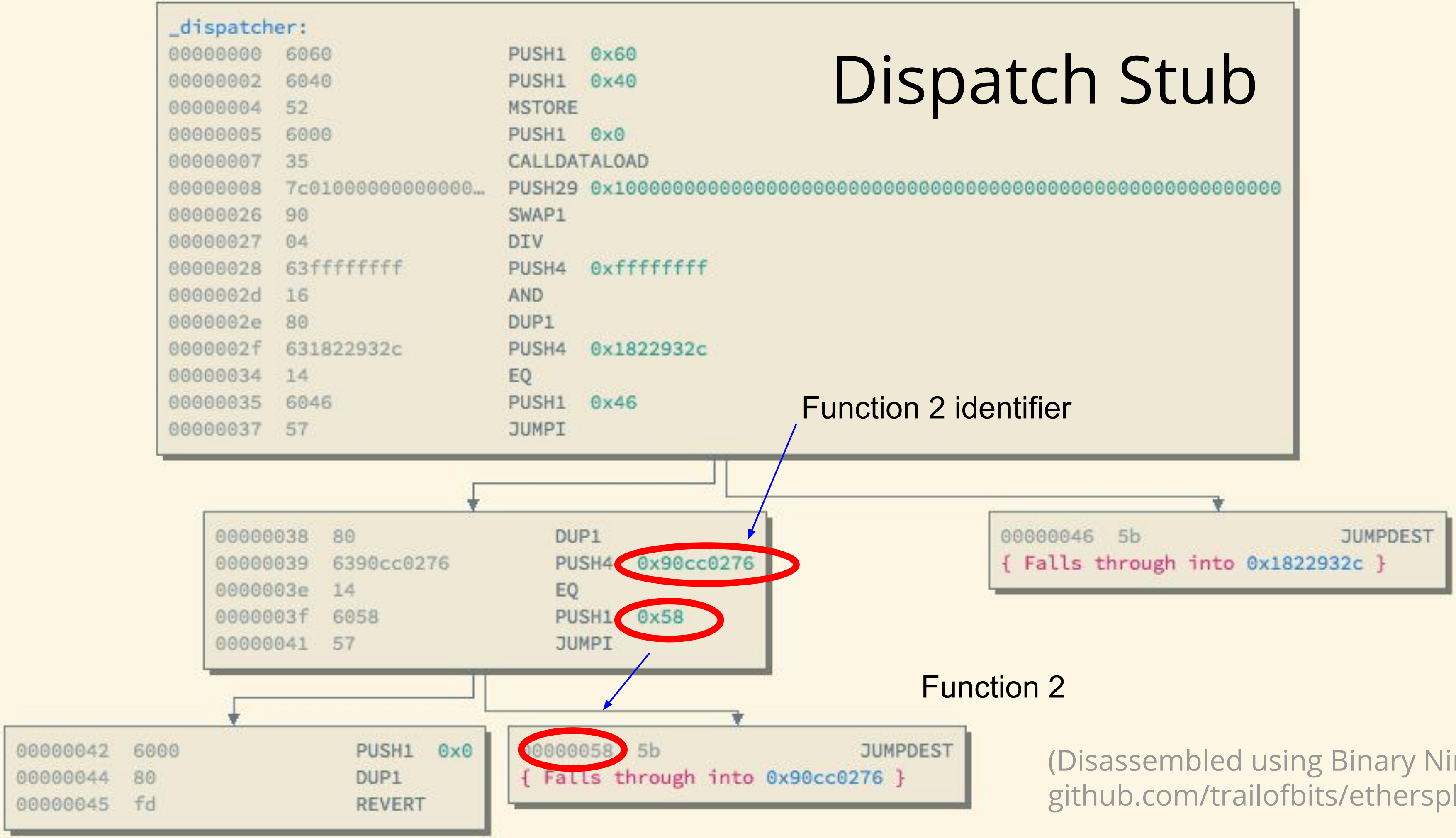
Function 1

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

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

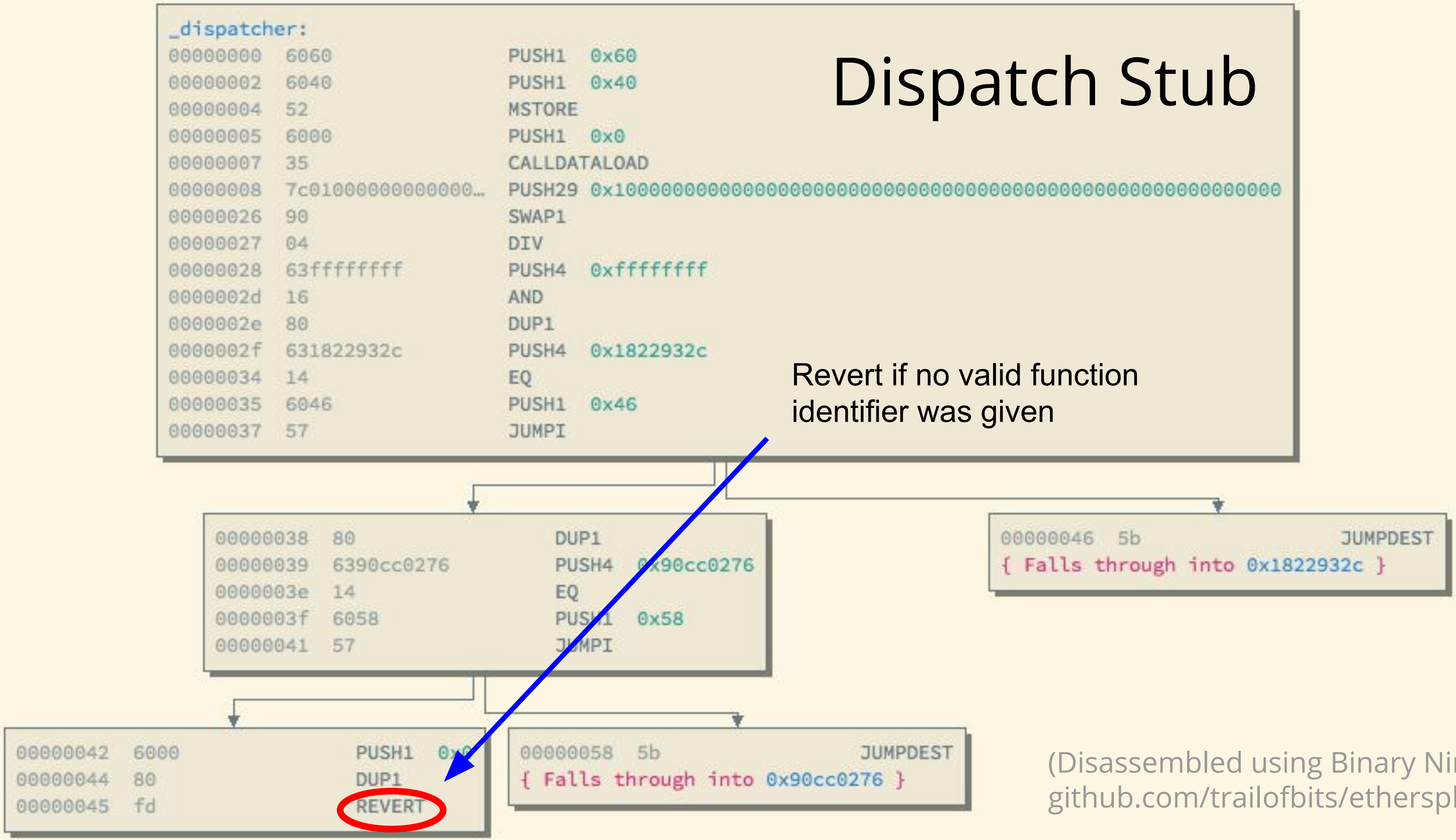
(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)

Dispatch Stub



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

Dispatch Stub



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

Ethereum

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

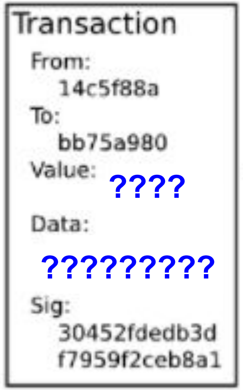
Symbolic Execution + Ethereum

Goals

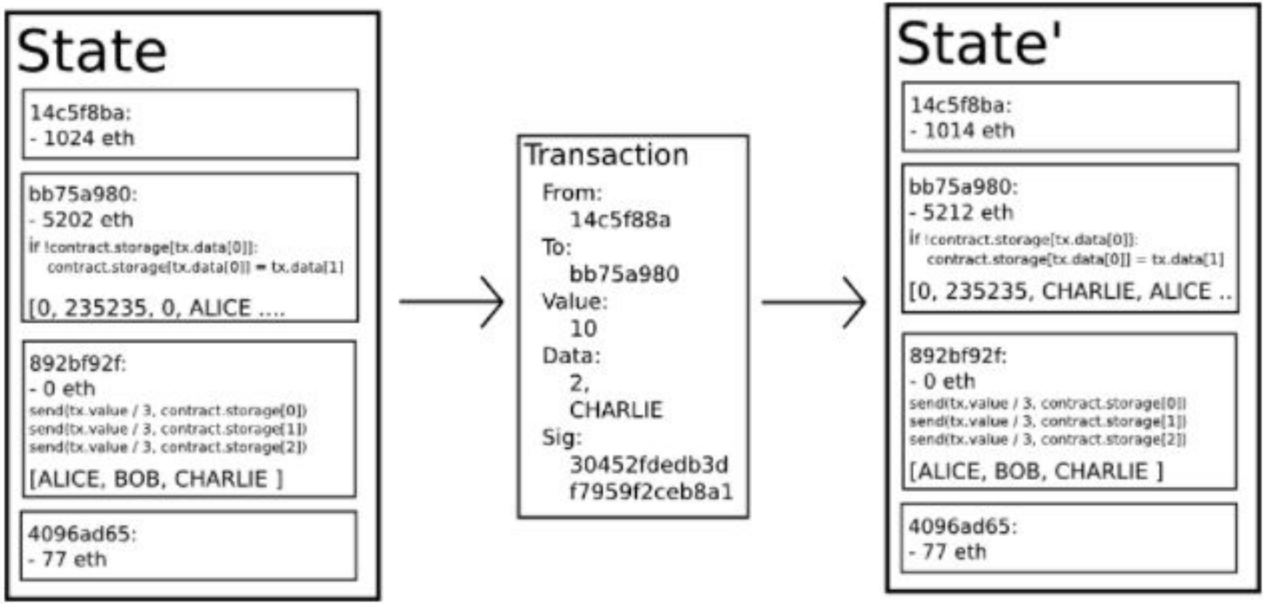
- Generate inputs that exercise contract functionality
- Enumerate state space & discover failure states
- Allow humans to prove properties about contract

Methodology

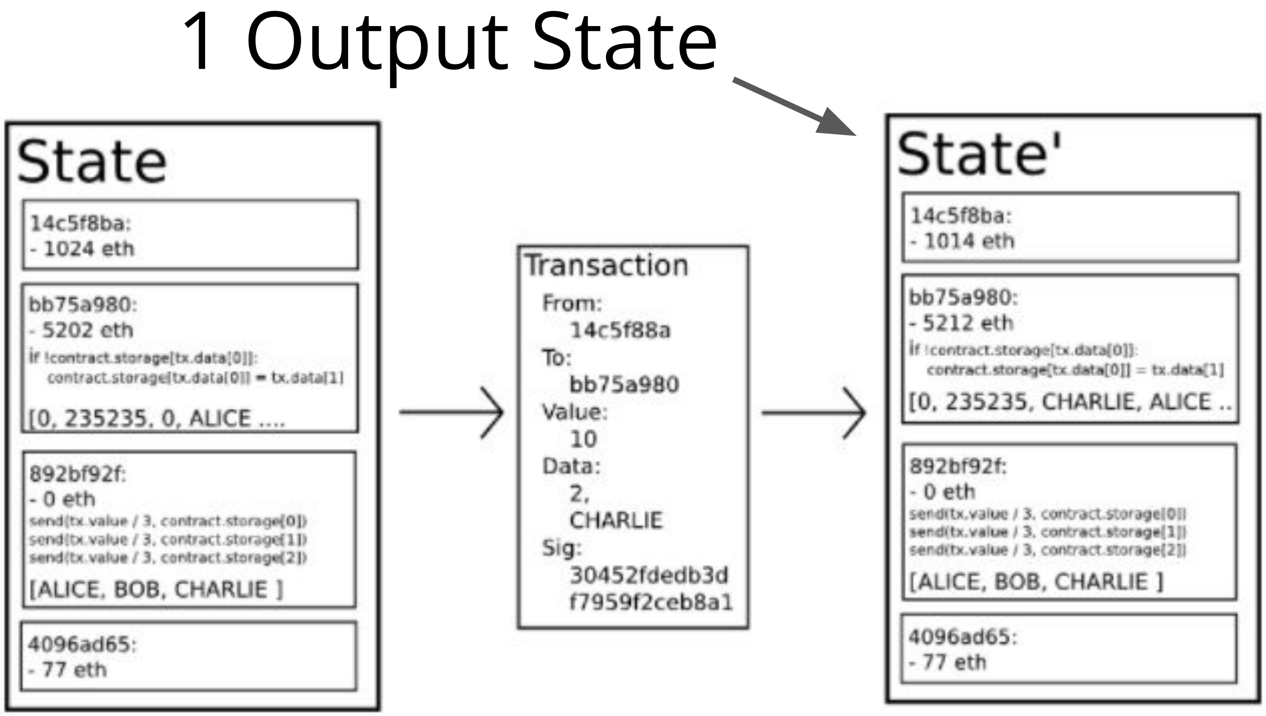
- Implement symbolic EVM interpreter
- Execute contracts with symbolic input
    - Symbolic transaction value
    - **Symbolic transaction data buffer**



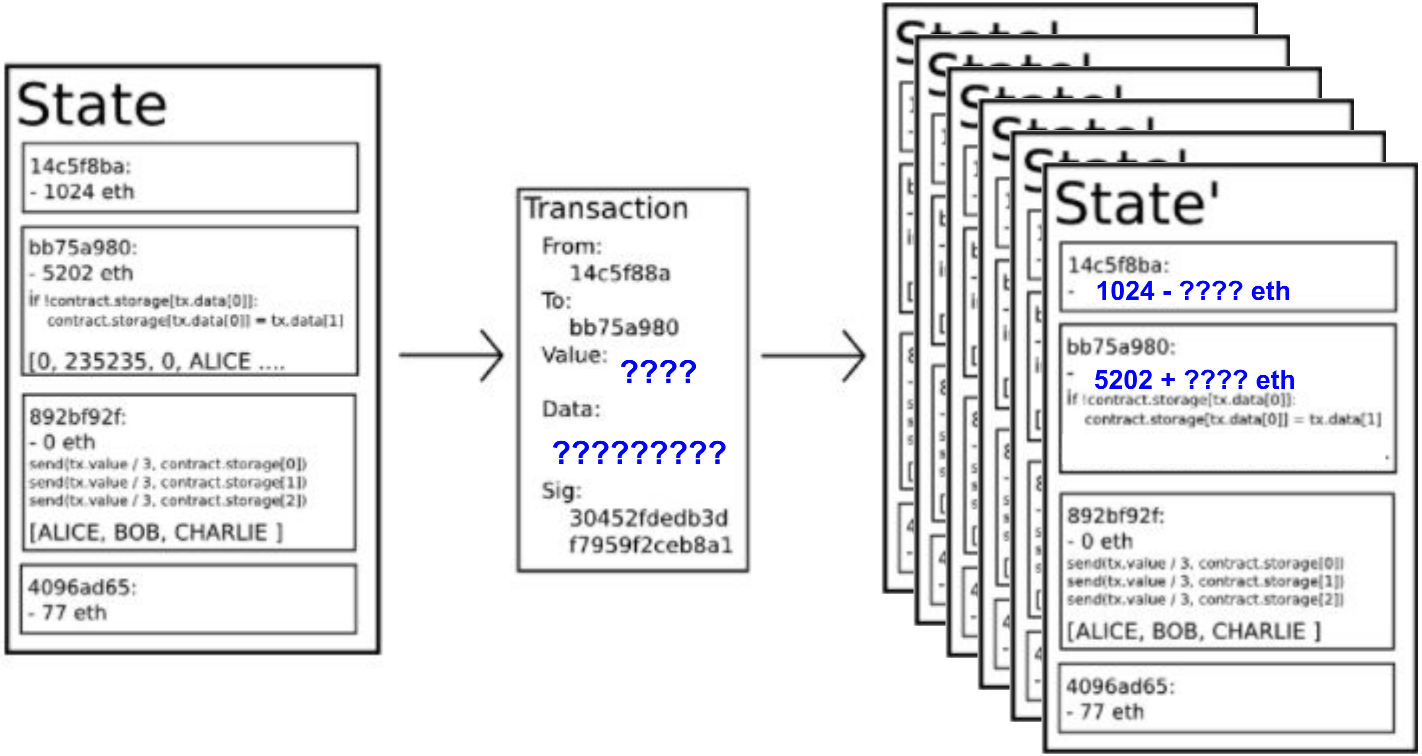
Concrete Transaction



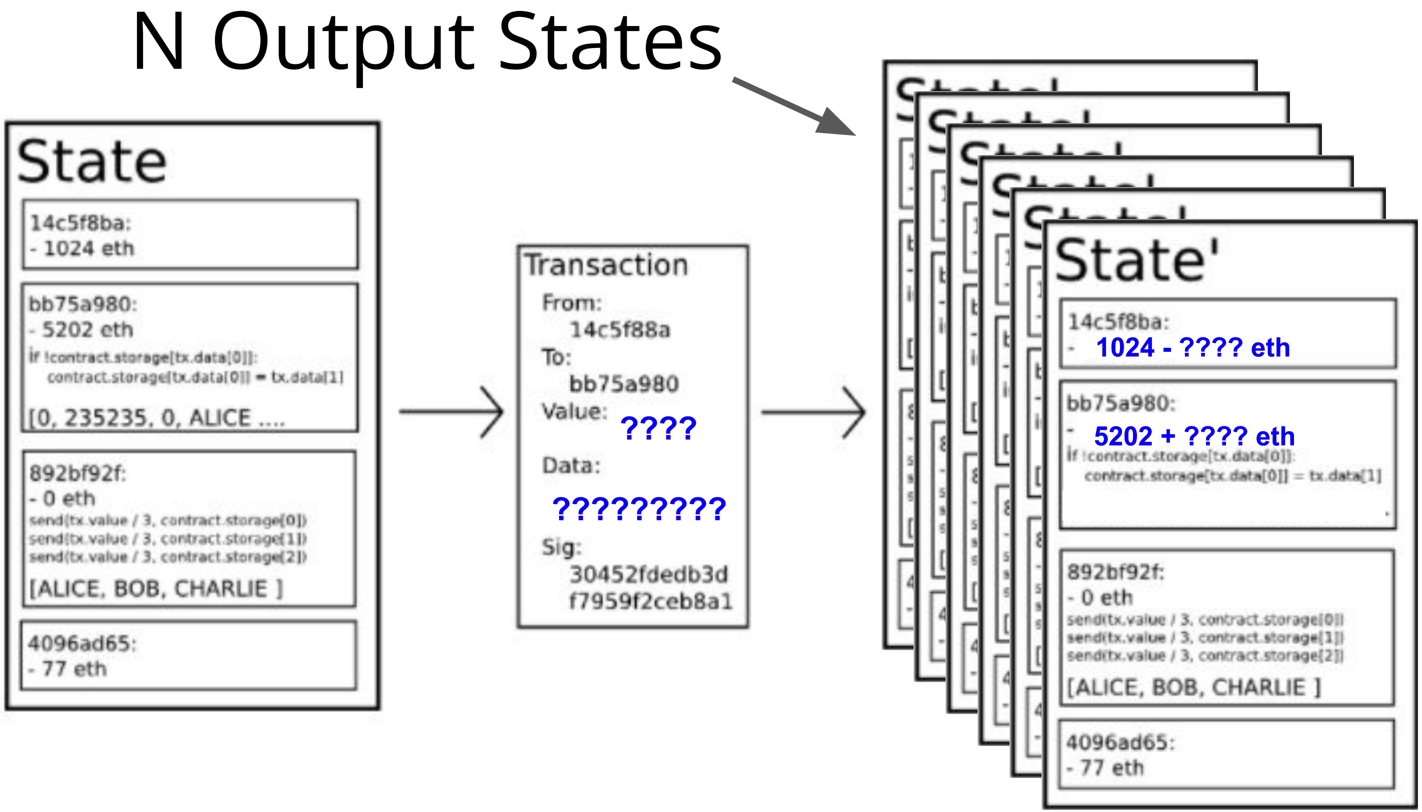
Concrete Transaction



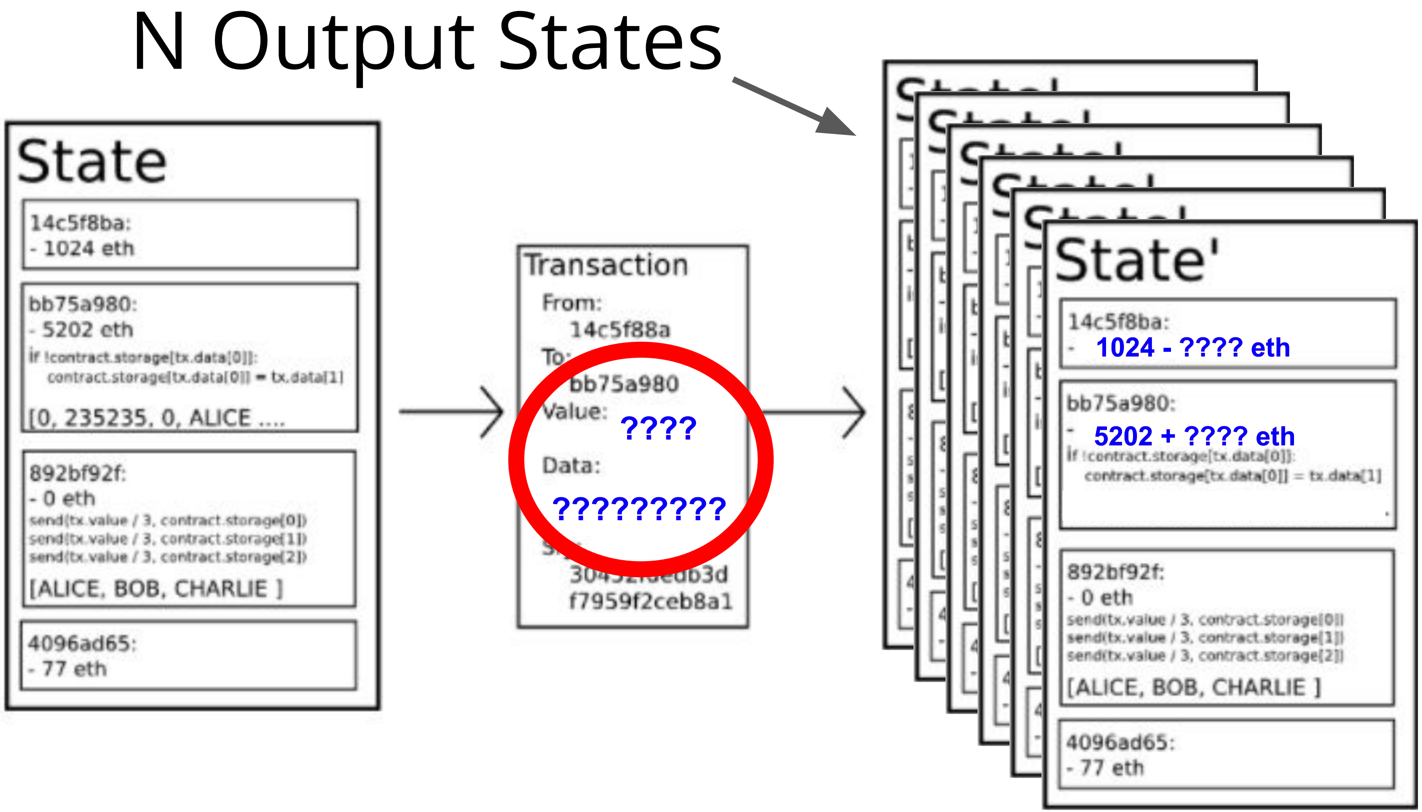
Symbolic Transaction



Symbolic Transaction

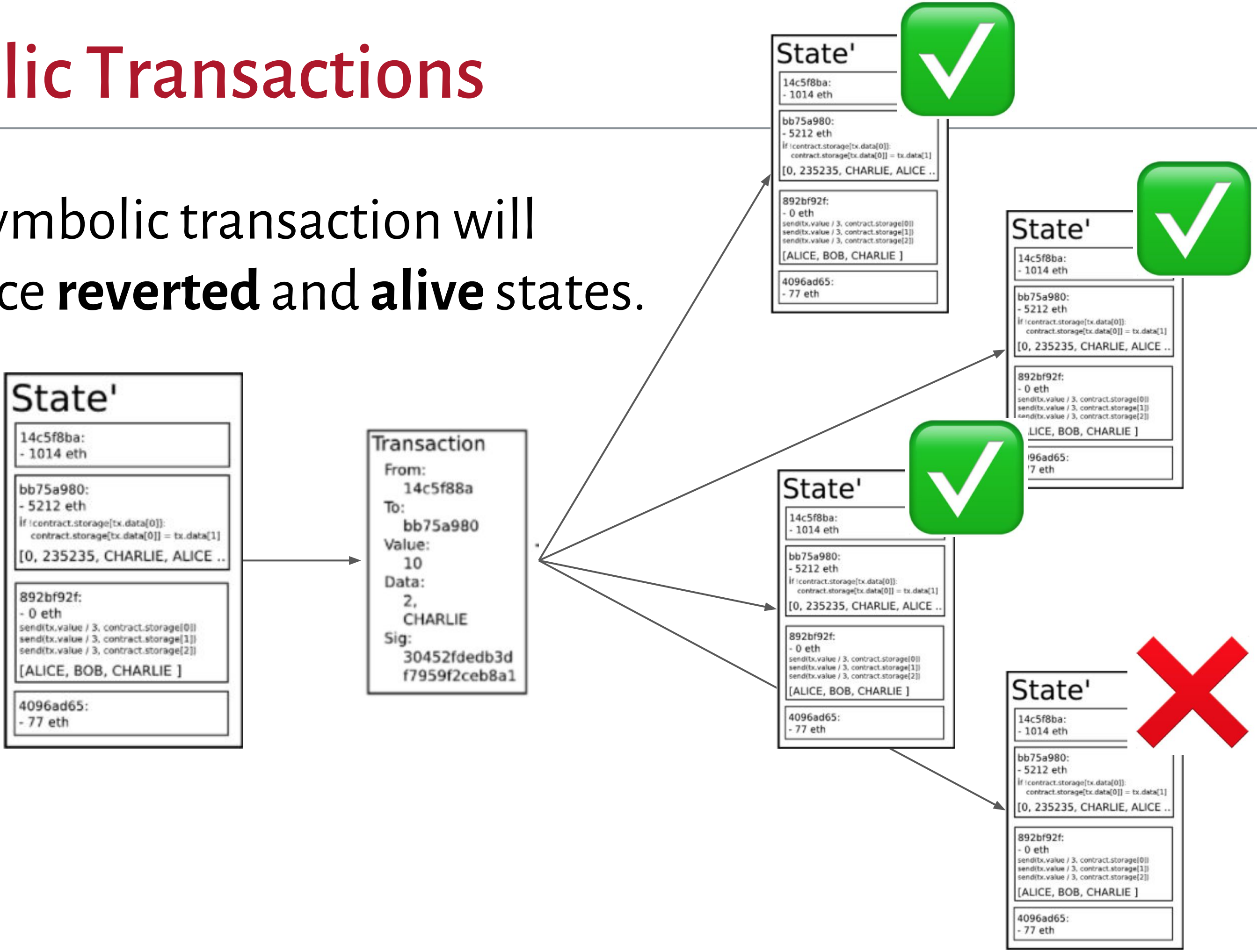


Symbolic Transaction



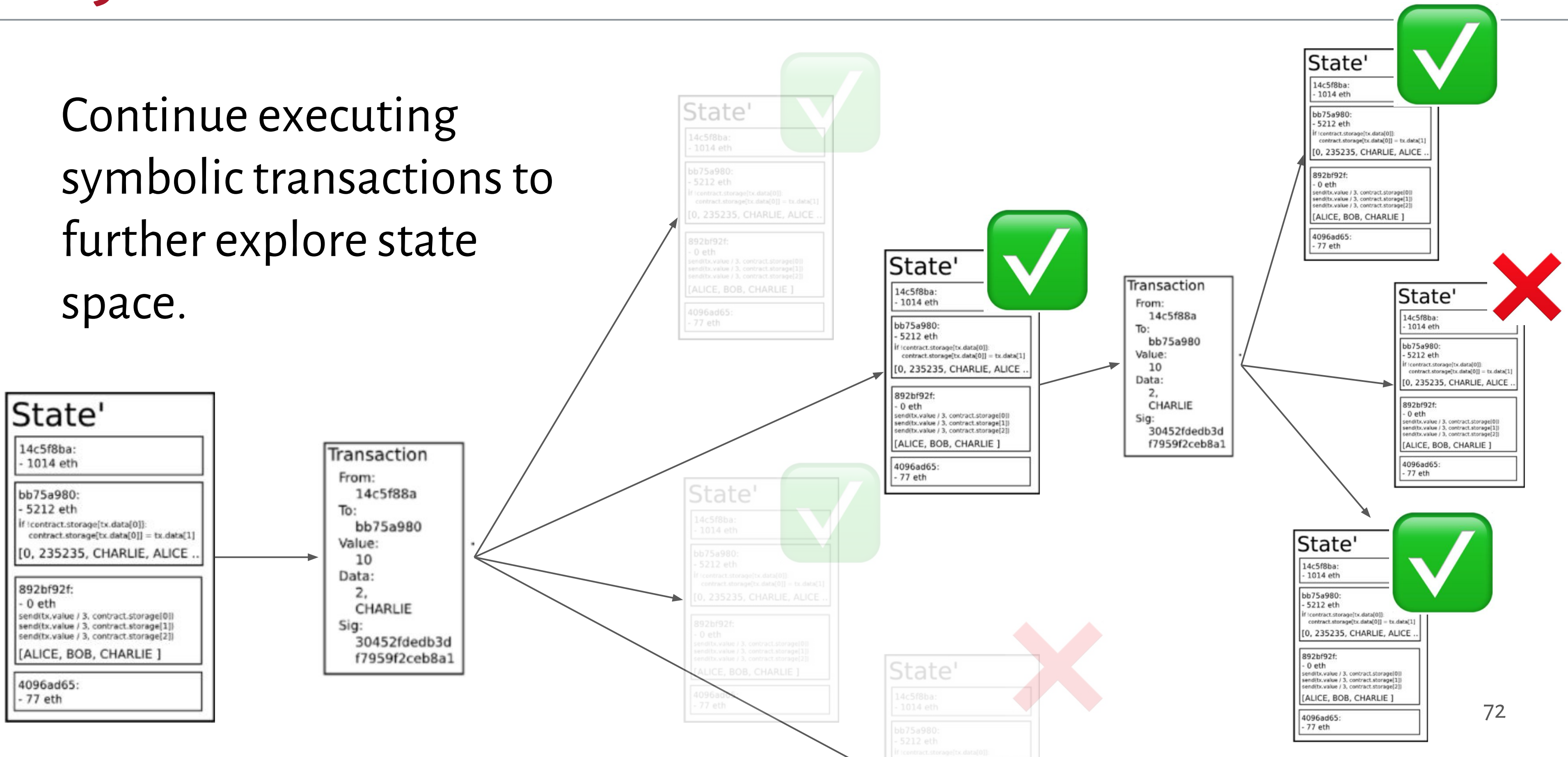
Symbolic Transactions

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



Symbolic Transactions

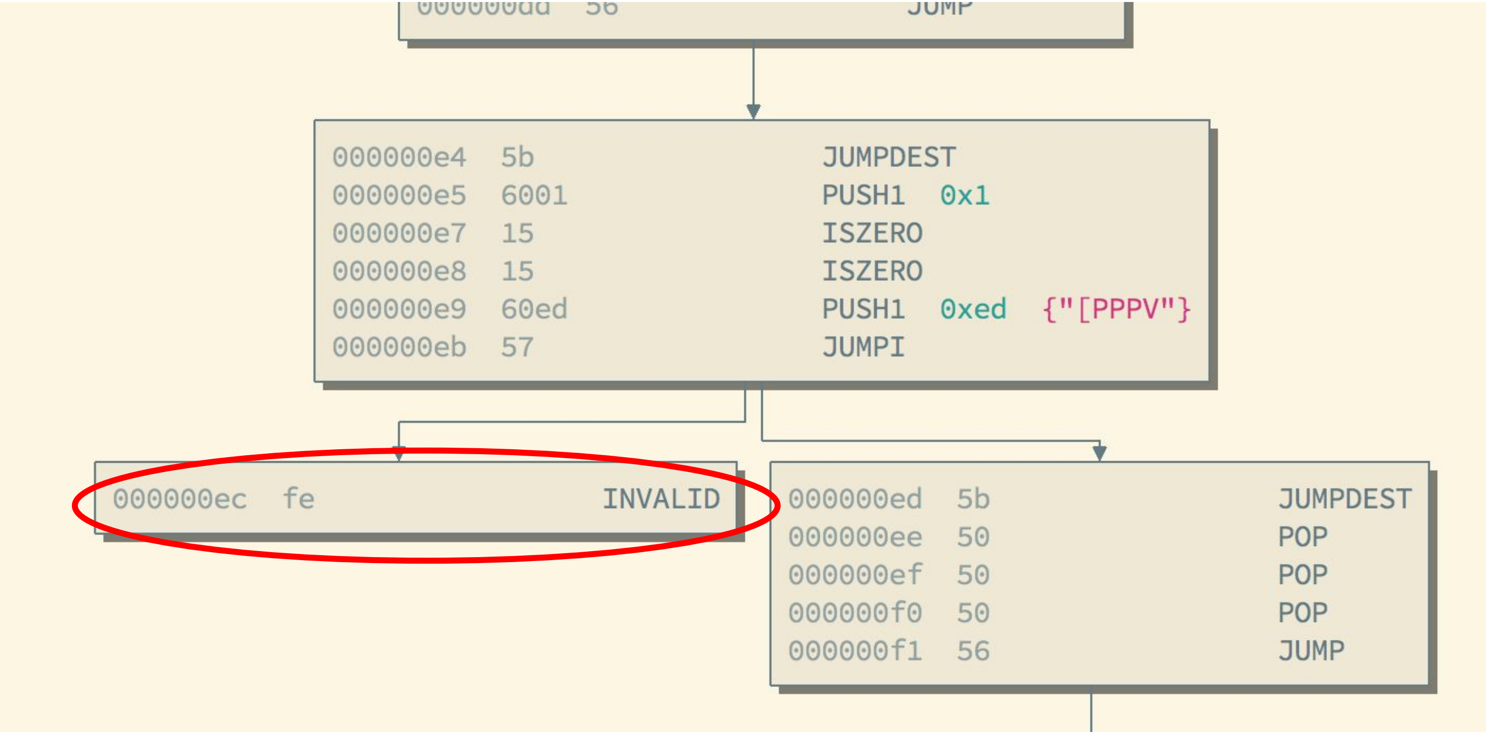
Continue executing symbolic transactions to further explore state space.



Applications of Ethereum symbolic execution

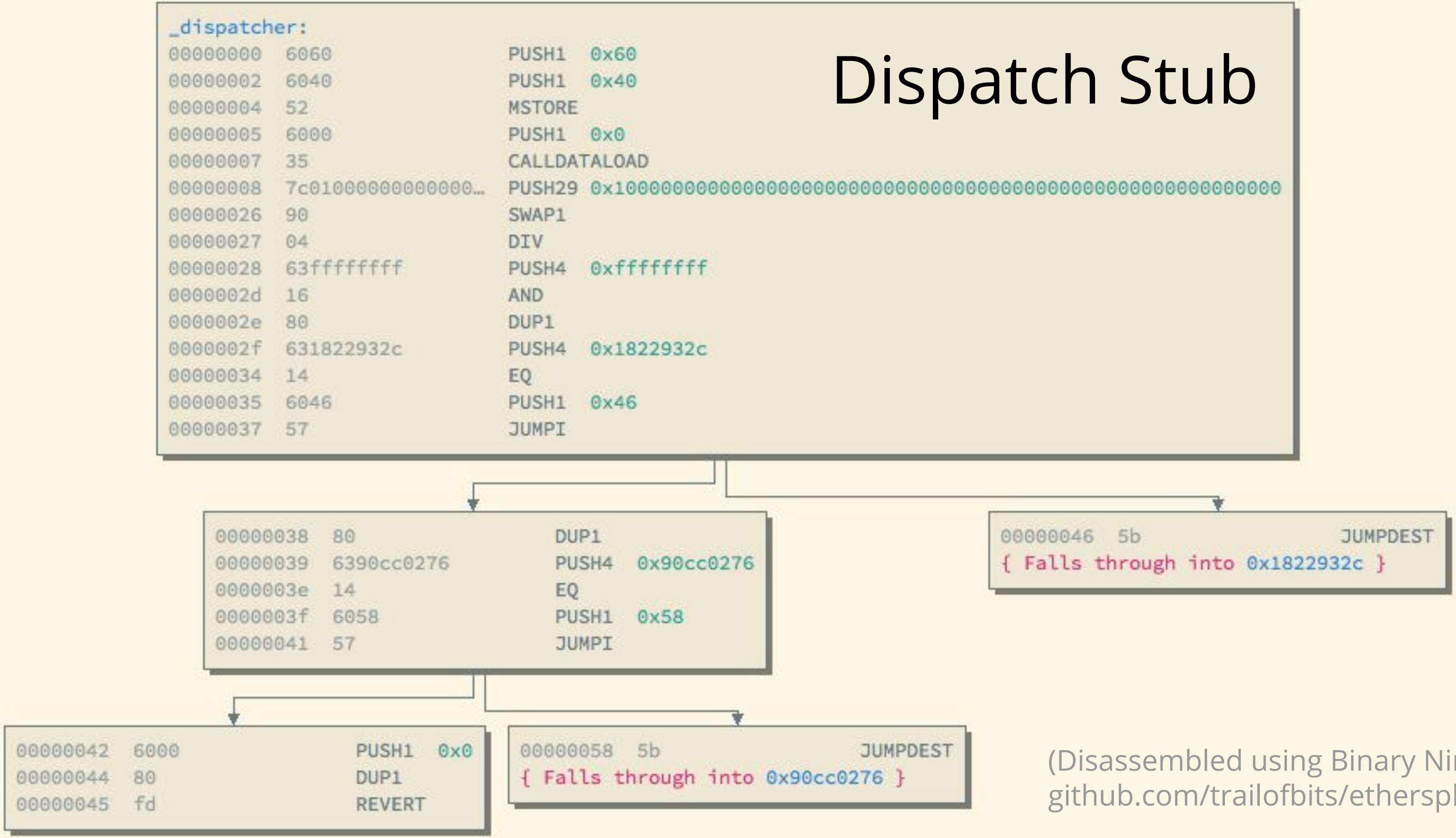
- Automatically check assertions
- Discover all functions in a binary contract
- Generate transaction sets that enumerate paths in those functions

Check assertions

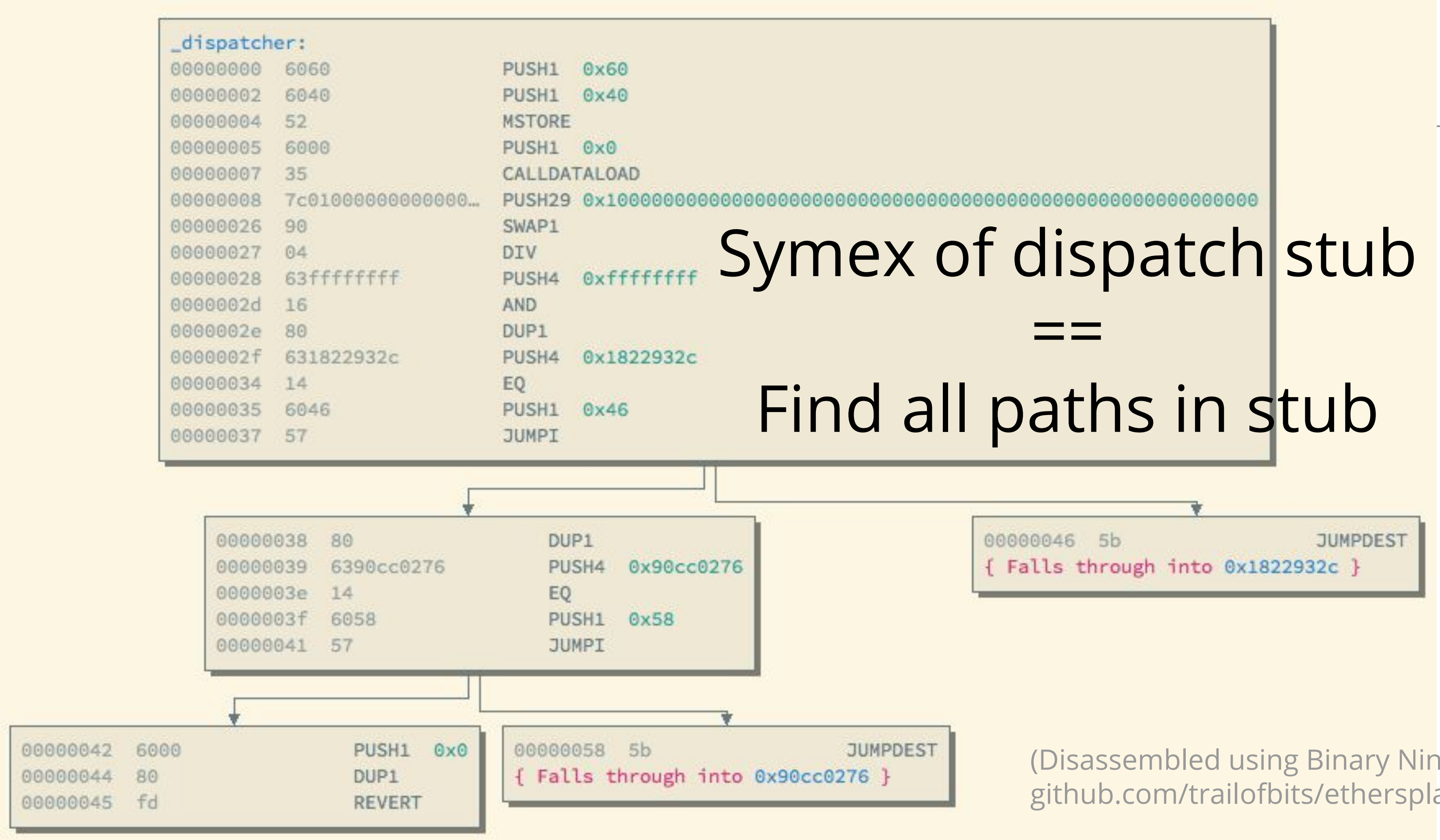


Function Discovery

Dispatch Stub

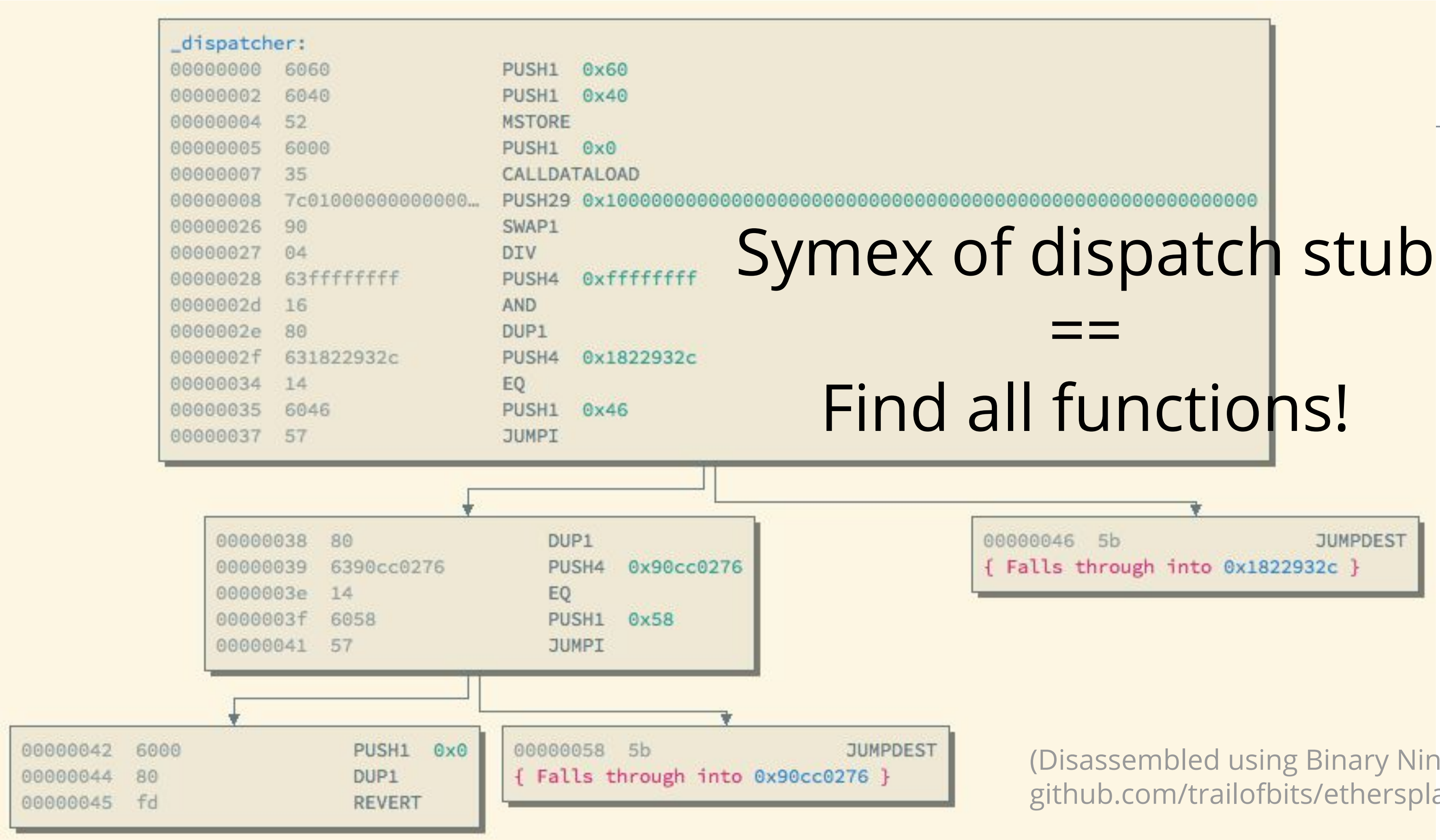


(Disassembled using Binary Ninja + github.com/trailofbits/ethersplay)



Symex of dispatch stub
==
Find all paths in stub

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



Symex of dispatch stub
==
Find all functions!

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

Transaction generation

Will require 2 tx to reach overflow

```
pragma solidity ^0.4.15;

contract SymExExample {
    uint did_init = 0;

    // function id: 0x13371337
    function test_me(int input) {
        if (did_init == 0) {
            did_init = 1;
            return;
        }

        if (input < 42) {
            // safe
            return;
        } else {
            // overflow possibly!
            int could_overflow = input + 1;
        }

    }
}
```

Transaction generation

Will require 2 tx to reach overflow

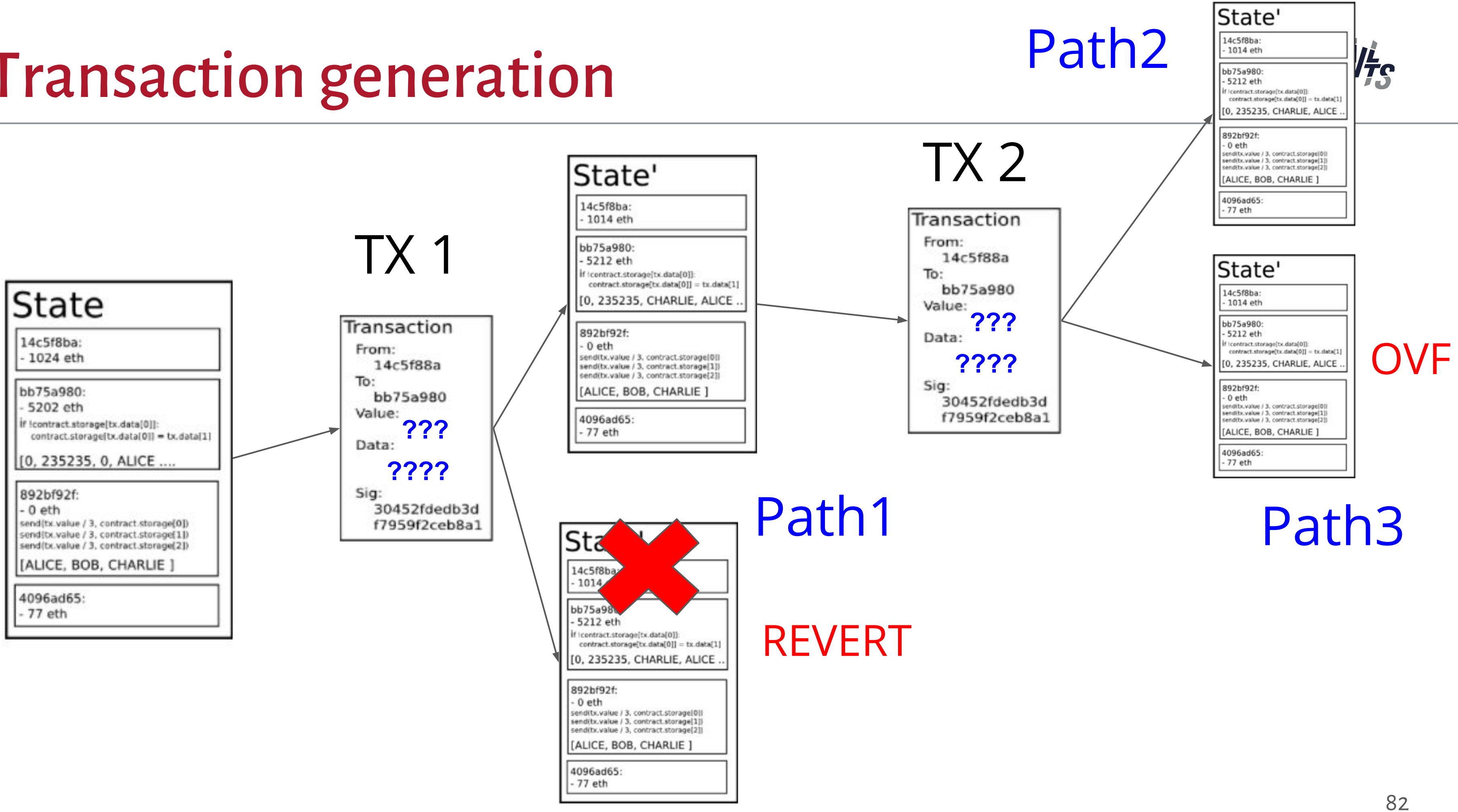
```
pragma solidity ^0.4.15;

contract SymExExample {
    uint did_init = 0;

    // function id: 0x13371337
    function test_me(int input) {
        if (did_init == 0) {
            did_init = 1;
            return;
        }

        if (input < 42) {
            // safe
            return;
        } else {
            // overflow possibly!
            int could_overflow = input + 1;
        }

    }
}
```

Transaction generation

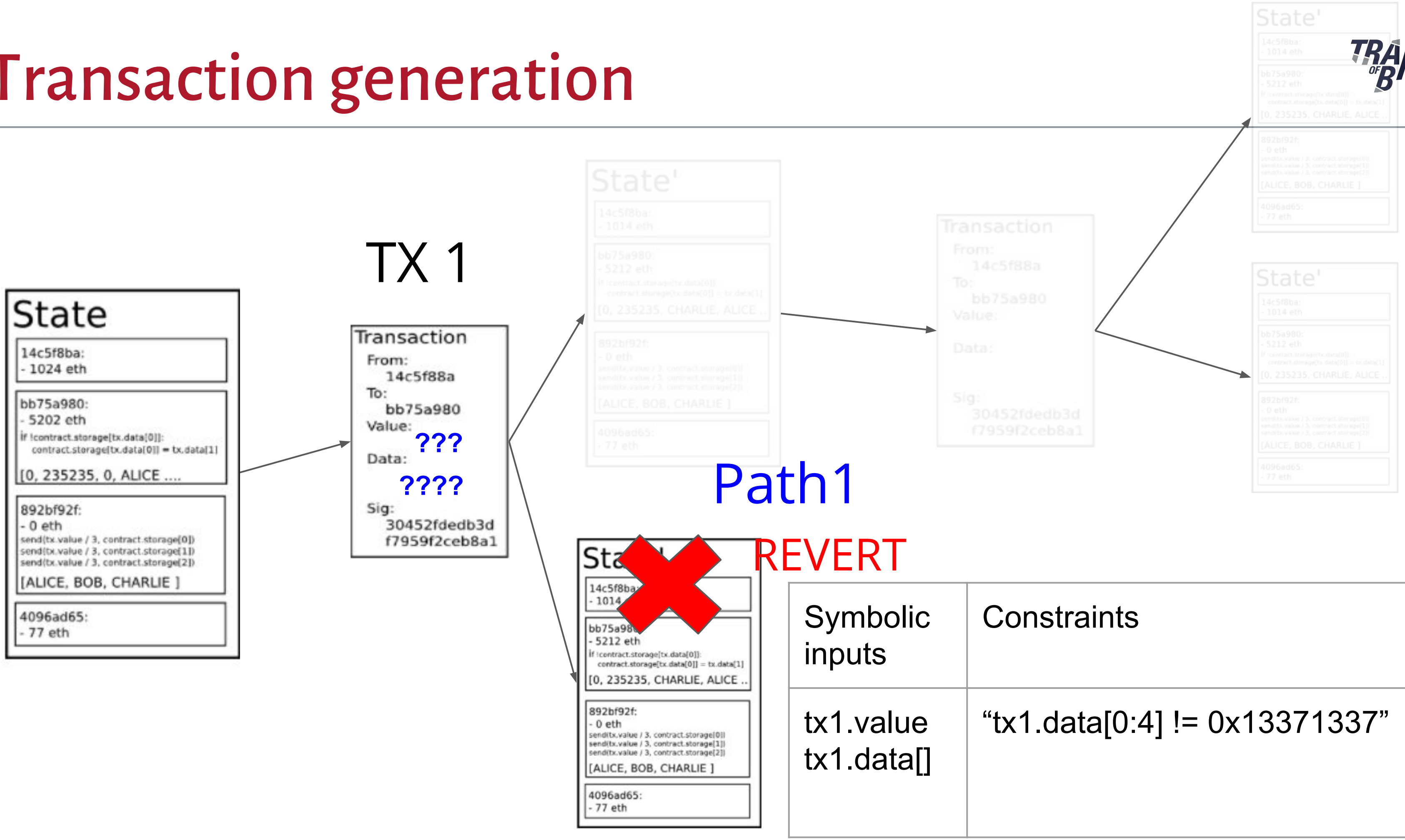
Will require 2 tx to reach overflow

```
pragma solidity ^0.4.15;

contract SymExExample {
    uint did_init = 0;

    // function id: 0x13371337
    function test_me(int input) {
        if (did_init == 0) {
            did_init = 1;
            return;
        }

        if (input < 42) {
            // safe
            return;
        } else {
            // overflow possibly!
            int could_overflow = input + 1;
        }

    }
}
```

Transaction generation

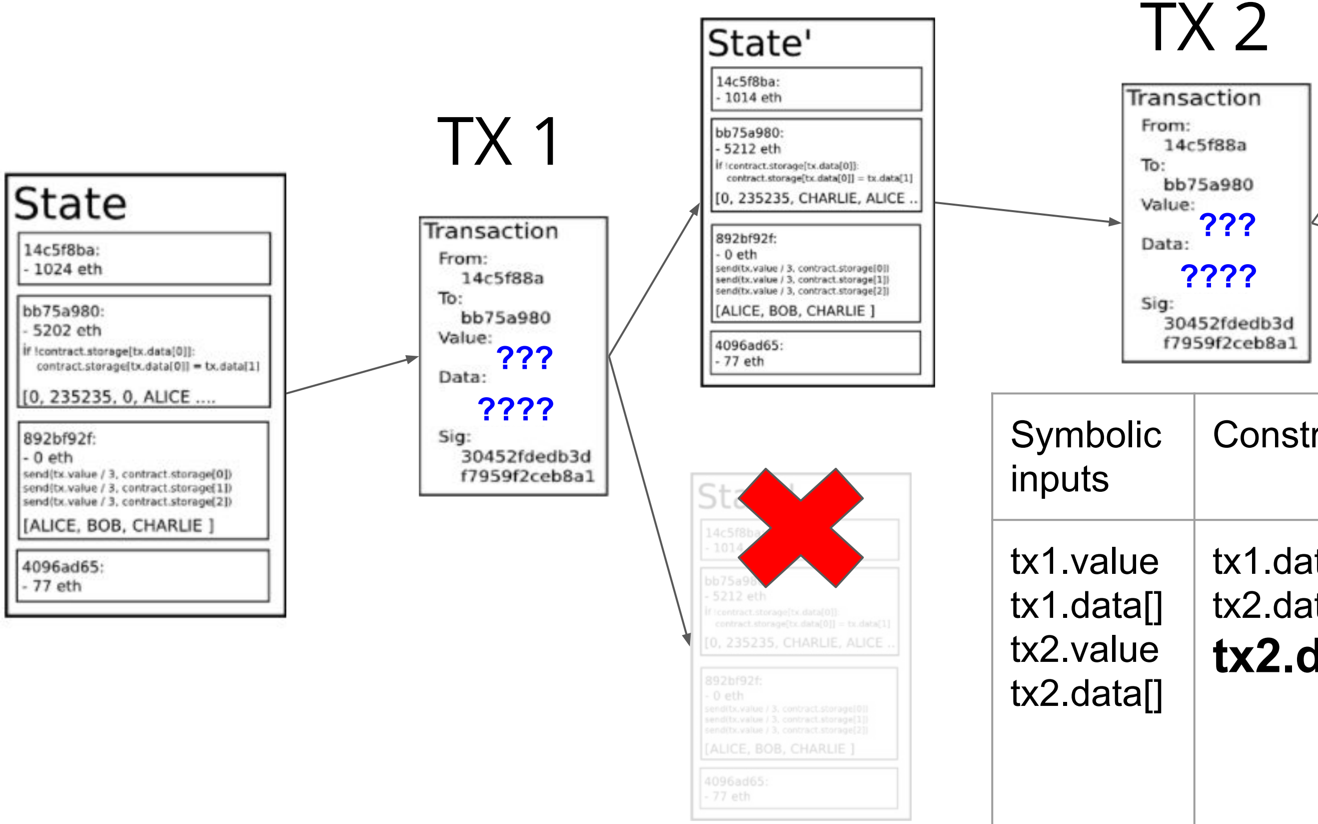


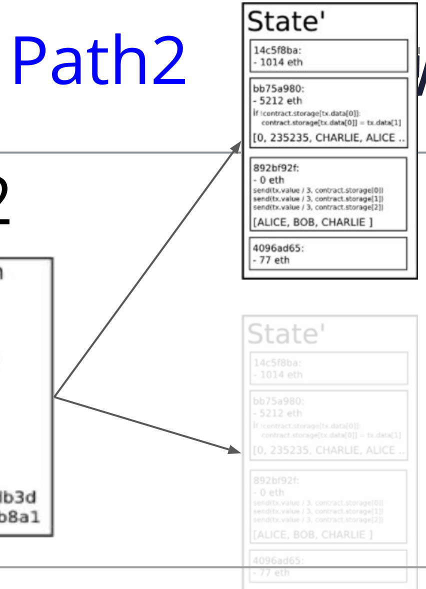
Transaction generation



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

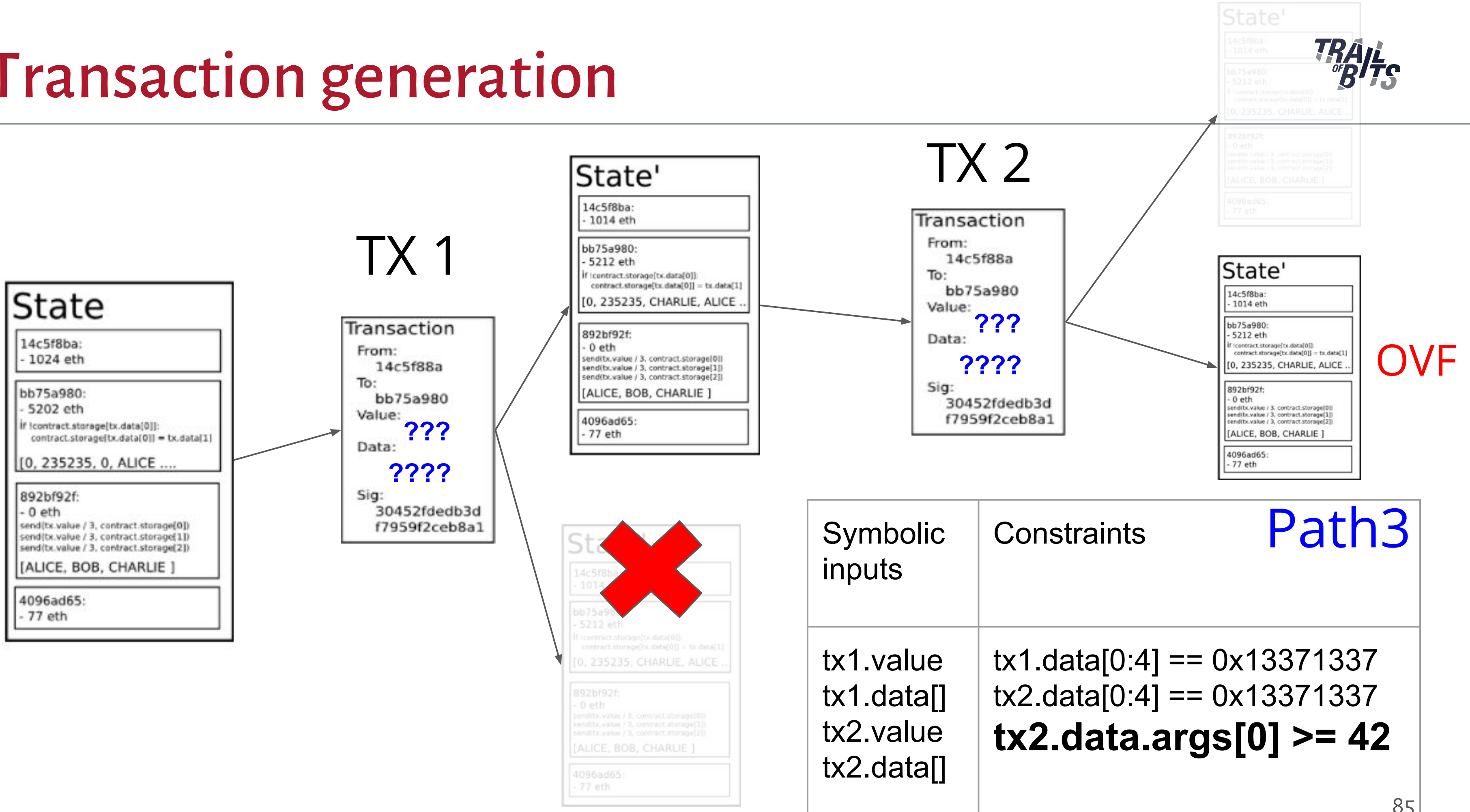
Transaction generation





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

TX 1

Path1

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

(constraint solver)

Z3

Transaction
From:
14c5f88a
To:
bb75a980
Value: 583749
Data:
0xcafecafe…..
Sig:
30452fdedb3d
f7959f2ceb8a1

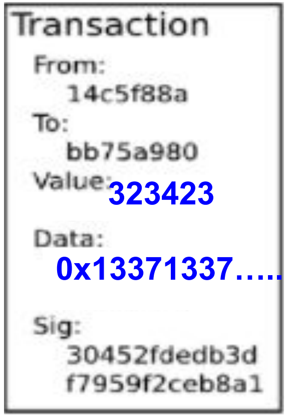
Transaction generation

Path 2

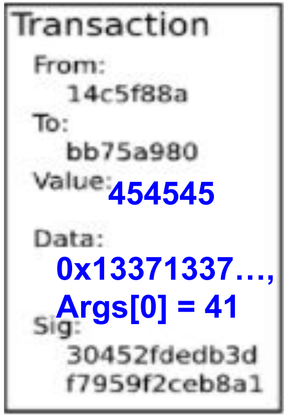
Symbolic inputs	Constraints
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] < 42**

(constraint solver)

TX 1



TX 2



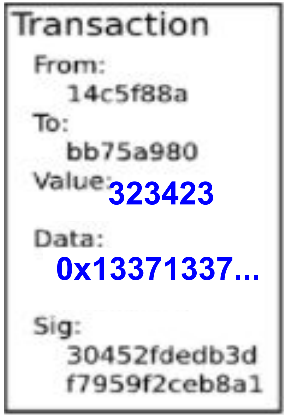
Transaction generation

Path 3

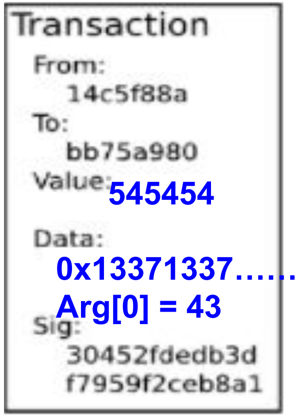
Symbolic inputs	Constraints
tx1.value tx1.data[] tx2.value tx2.data[]	tx1.data[0:4] == 0x13371337 tx2.data[0:4] == 0x13371337 **tx2.data.args[0] >= 42**

(constraint solver)

TX 1



TX 2



Challenges

Challenges

- State explosion
- Symbolic hashing
- Dynamic arguments

State Explosion

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

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

State Explosion

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

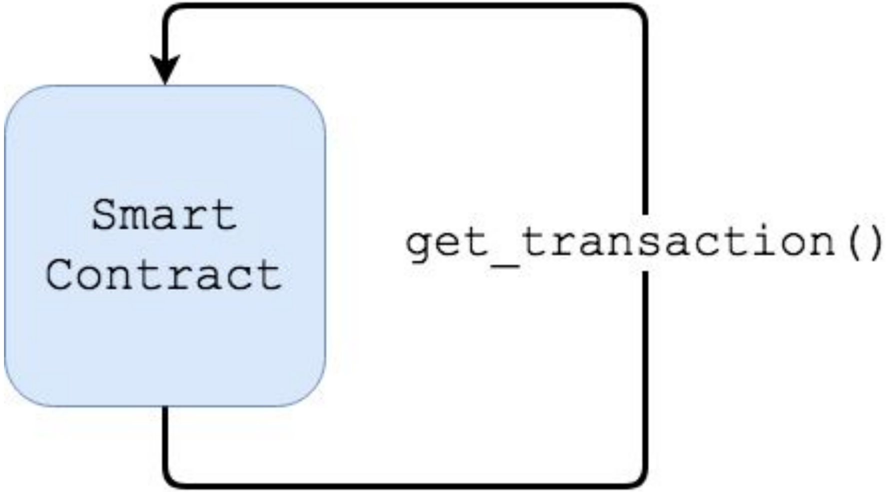
"Enhancing Symbolic Execution with Veritesting", Avgerinos, et. al ICSE 2014

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?

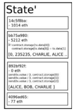
```
for (;;) {                      // loop
    tx = get_transaction();
    run_contract(tx);           // branch based on input
}
```

State Explosion?

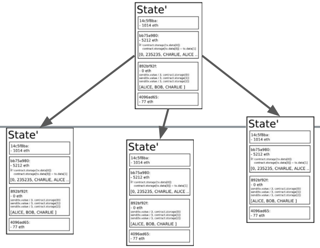
Look familiar?

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

State Explosion



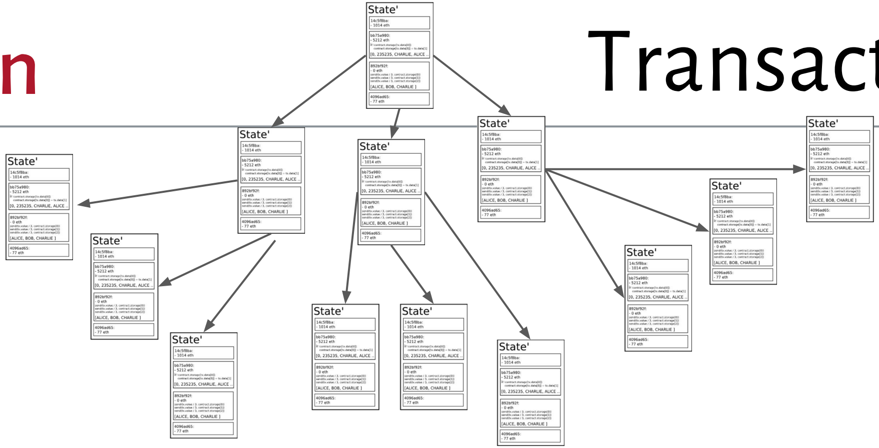
State Explosion



Transaction 1

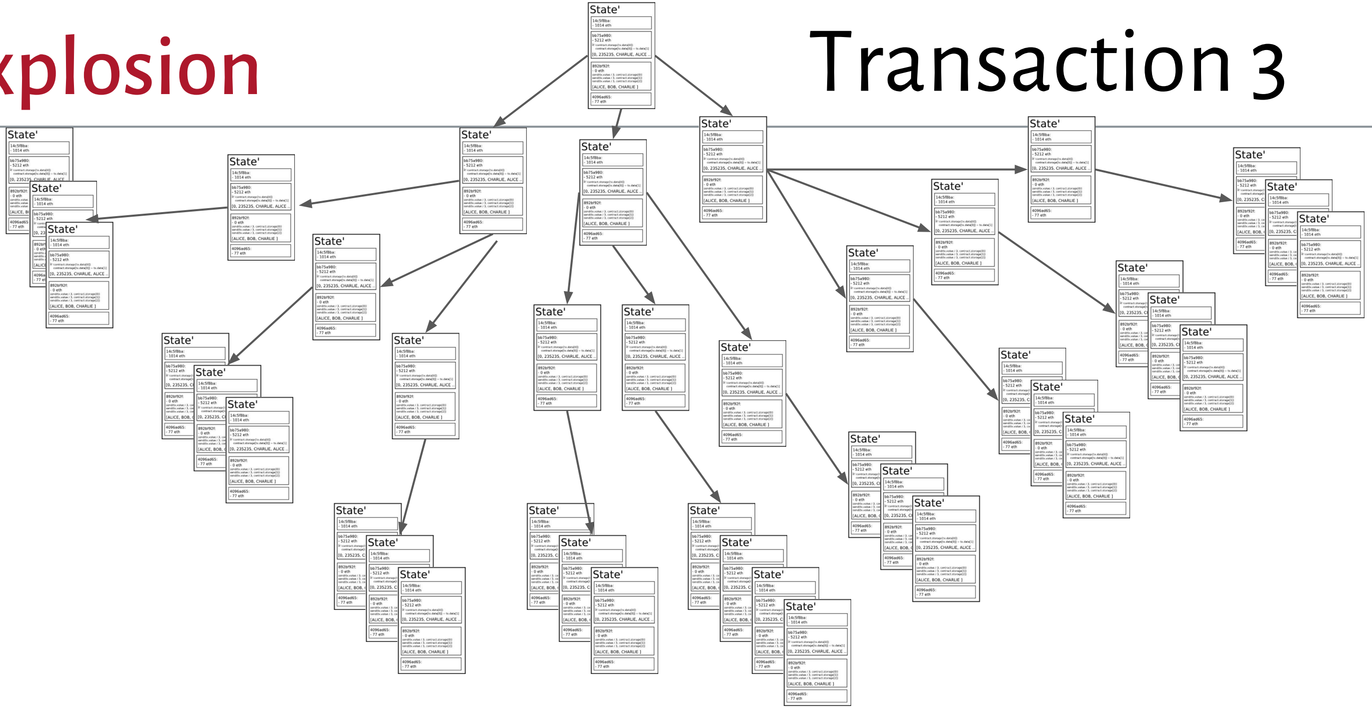
State Explosion

Transaction 2



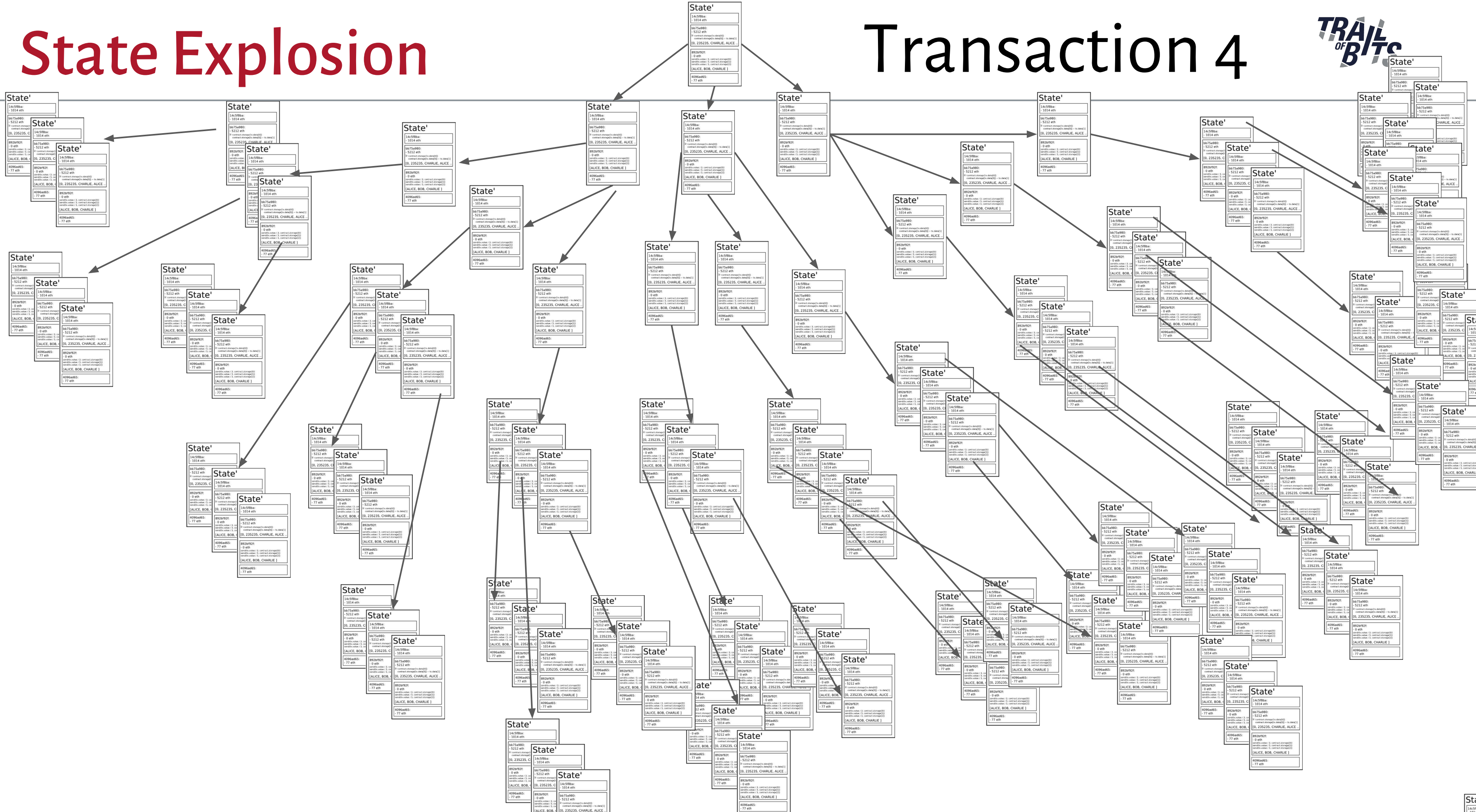
State Explosion

Transaction 3



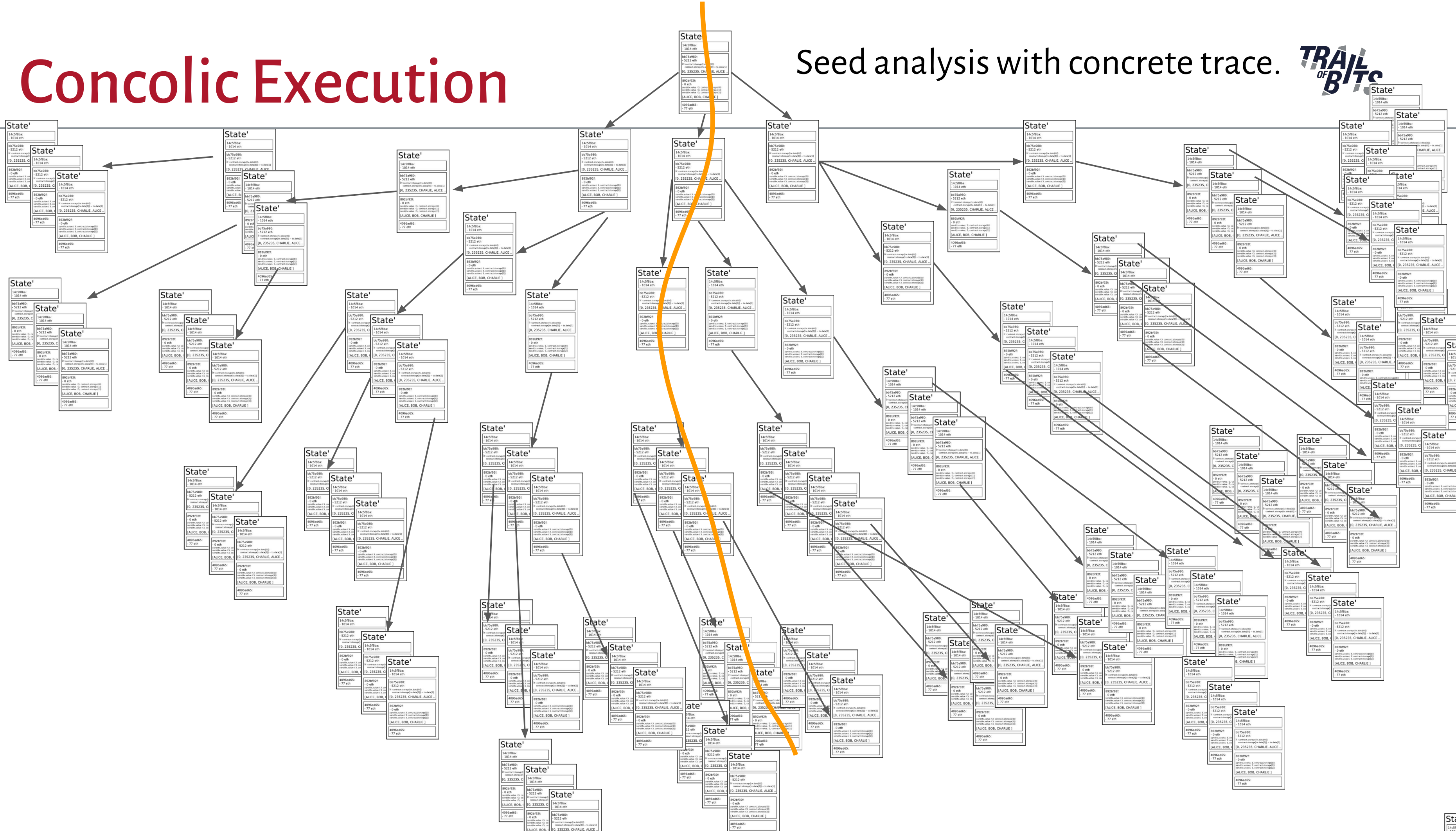
State Explosion

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
- 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..

```
sha3("userA")  ──────────►  0x34b34b34b..
```

key	hash
userA	0x34b34b34b..

Solidity: mappings

Record concrete hashes..

```
sha3("userB") → 0x56c56c56c56c..
```

key	hash
userA	0x34b34b34b..
userB	0x56c56c56c56c

Solidity: mappings

Rather than computing the symbolic hash...

`sha3(`**`symbol`**`)` → `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

		a	b_offset	b_nelements	b[0]	b[1]	b[2]
tx data:	\x39\x9b\x79\x2b	1	64	3	42	43	44
size:	4	32	32	32	32	32	32

Solidity: dynamic arguments

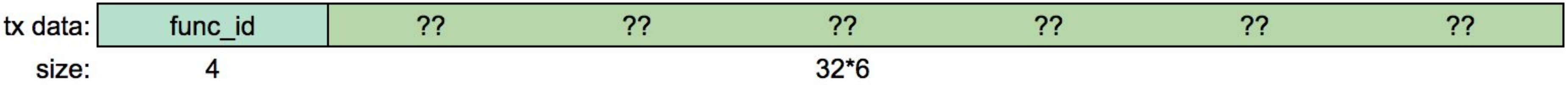
- Functions can receive variable length data
- Transaction data becomes complex, with various offset and size fields
- Leads to symbolic indexing & memcpy operations

		a	b_offset	b_nelements	b[0]	b[...]	b[n]
tx data:	\x39\x9b\x79\x2b	1	??	??	??	??	??
size:	4	32	32	32	32	32	32

Solidity: dynamic arguments

- Solution: **aggressively concretize offset & nelements fields**
- Length of symbolic transaction data is concrete
- Statically compute even allocation of data space for all dynamic arguments, concretize the lengths of those arrays and the offsets
- Requires function prototypes

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



Solidity: dynamic arguments

- Solution: **aggressively concretize offset & nelements fields**
- Length of symbolic transaction data is concrete
- Statically compute even allocation of data space for all dynamic arguments, concretize the lengths of those arrays and the offsets
- Requires function prototypes

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

		a_offset	b_offset	a_nelements	a[0]	b_nelements	b[0]
tx data:	func_id	**32*2**	**32*4**	**1**	??	**1**	??
size:	4	32	32	32	32	32	32

Other Challenges

- Complete symbolic environment model must support inter contract calls
- Gas/Symbolic Gas
- Symbolic indexing is common due to direct mapping

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

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`