Rattle

reCON Montreal 2018

Agenda

1. Ethereum Smart Contracts
2. Ethereum VM Internals
3. SSA Construction
4. Analysis

Who I am

Ryan Stortz (@withzombies)

- Principal Security Researcher at Trail of Bits in NYC
- Previously at Raytheon SI in Melbourne, FL
- In the industry for ~10 years
- Used to play CTF: VedaGodz, HatesIrony, Marauders, Hacking4Danbi
- Used to host CTF: GhostInTheShellcode, CSAW CTF
- Past Presentations on Swift Reversing, Cyber Grand Challenge, Binary Ninja, Blackhat Ethereum

Blackhat Ethereum

Blackhat Ethereum

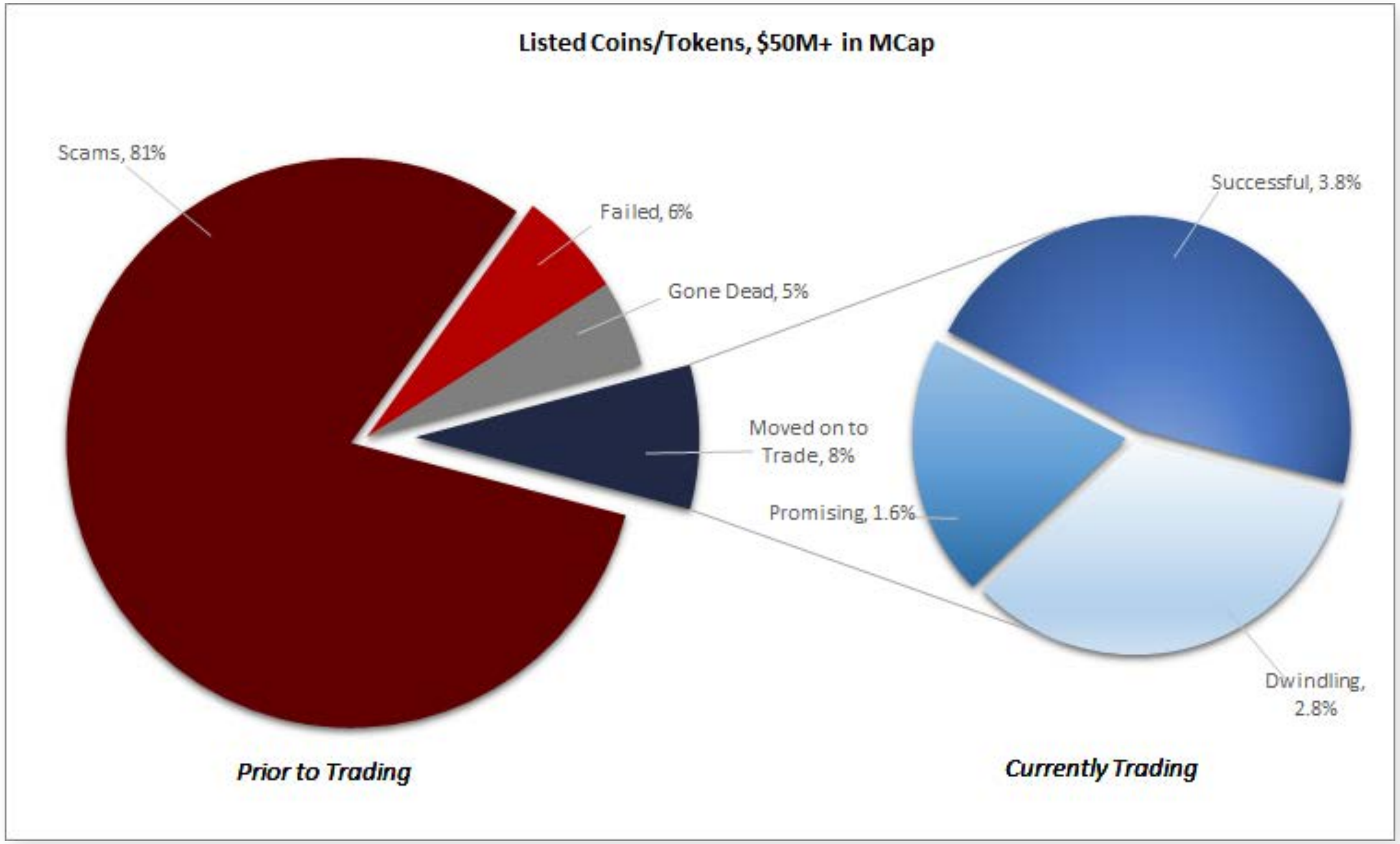
CanSecWest 2018

Ethereum Smart Contracts

Initial Coin Offerings

```
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

ICOs are scams



https://medium.com/satis-group/ico-quality-development-trading-e4fef28df04f

Ethereum VM Internals

- Stack machine
- EVM is a Harvard architecture!
- There are ~6 address spaces
  - Code, Stack, Call data, Storage, Memory, Return Data
- Native data width is 256 bits / 32 bytes
- ~181 opcodes, many are duplicates
  - PUSH1 – PUSH32, DUP1 – DUP16, SWAP1 – SWAP16
- All execution enters at PC 0x0 and functions are dispatched based on call data

Stack Machine

Code	Stack
PUSH1 0x2	**0x2**
PUSH1 0x3	
ADD	
PUSH1 0x8	
MUL	

Code	Stack
PUSH1 0x2	0x2
PUSH1 0x3	**0x3**
ADD	
PUSH1 0x8	
MUL	

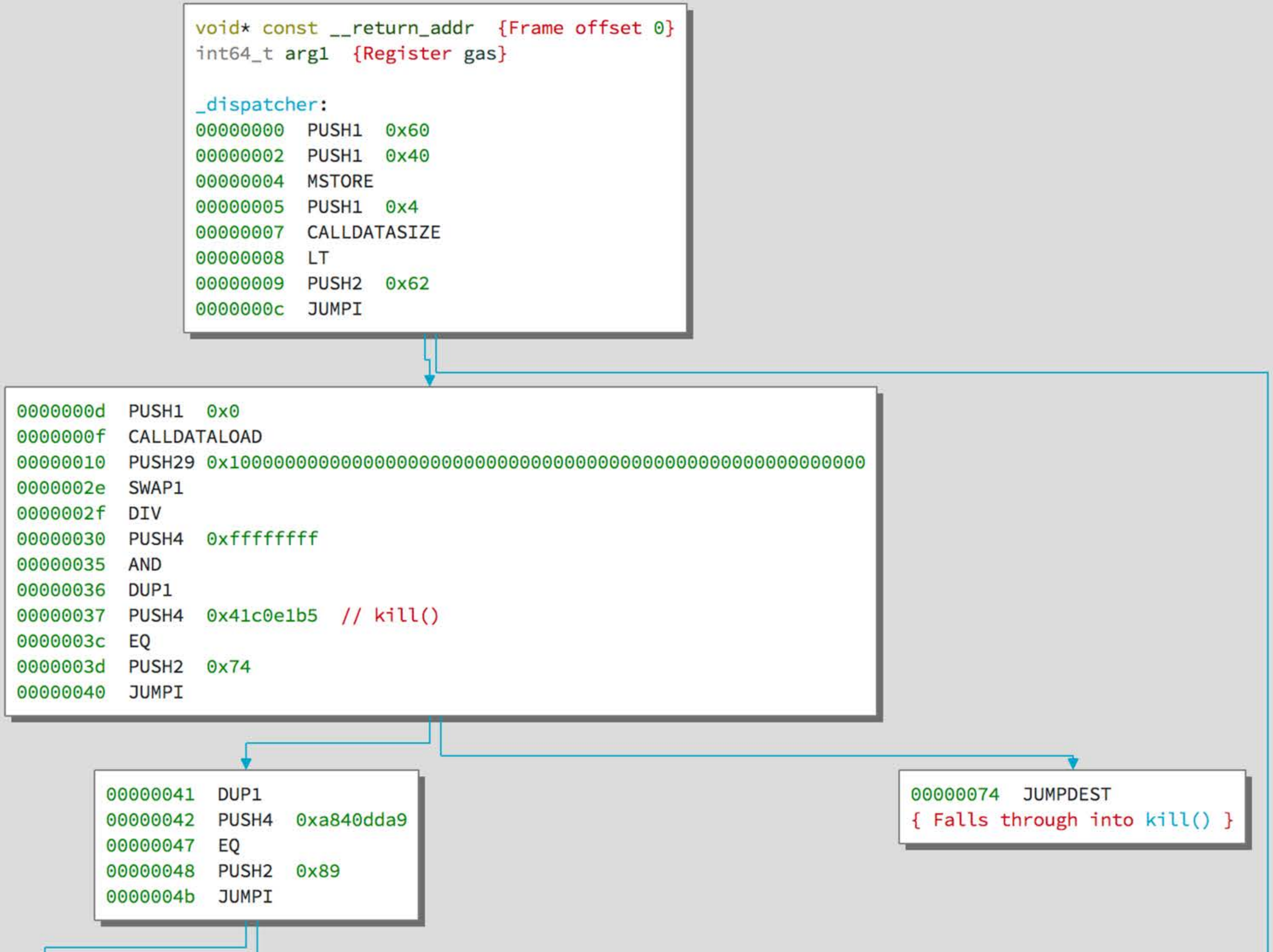
Code	Stack
PUSH1 0x2	**0x5**
PUSH1 0x3	
ADD	
PUSH1 0x8	
MUL	

Code	Stack
PUSH1 0x2	0x5
PUSH1 0x3	**0x8**
ADD	
PUSH1 0x8	
MUL	

Code	Stack
PUSH1 0x2	**0x28**
PUSH1 0x3	
ADD	
PUSH1 0x8	
MUL	

EVM Opcodes (the good ones)

Opcode	Purpose
JUMPI, JUMP, RETURN	These instructions define control flow
REVERT, INVALID	Exception causing opcodes
CALLVALUE	Transaction Ether (in Wei)
CALLDATASIZE, CALLDATALOAD	Transaction Arguments
SSTORE, SLOAD	Load and store to persistent storage
CALL, CALLCODE, DELEGATECALL	External Calls, can send Ether
SELFDESTRUCT	Destroy the contract and return its value



```
void* const __return_addr  {Frame offset 0}
int64_t arg1  {Register gas}

_dispatcher:
00000000  PUSH1  0x60
00000002  PUSH1  0x40
00000004  MSTORE
00000005  PUSH1  0x4
00000007  CALLDATASIZE
00000008  LT
00000009  PUSH2  0x62
0000000c  JUMPI
```

```
0000000d  PUSH1  0x0
0000000f  CALLDATALOAD
00000010  PUSH29 0x100000000000000000000000000000000000000000000000000000000
0000002e  SWAP1
0000002f  DIV
00000030  PUSH4  0xffffffff
00000035  AND
00000036  DUP1
00000037  PUSH4  0x41c0e1b5  // kill()
0000003c  EQ
0000003d  PUSH2  0x74
00000040  JUMPI
```

```
00000041  DUP1
00000042  PUSH4  0xa840dda9
00000047  EQ
00000048  PUSH2  0x89
0000004b  JUMPI
```

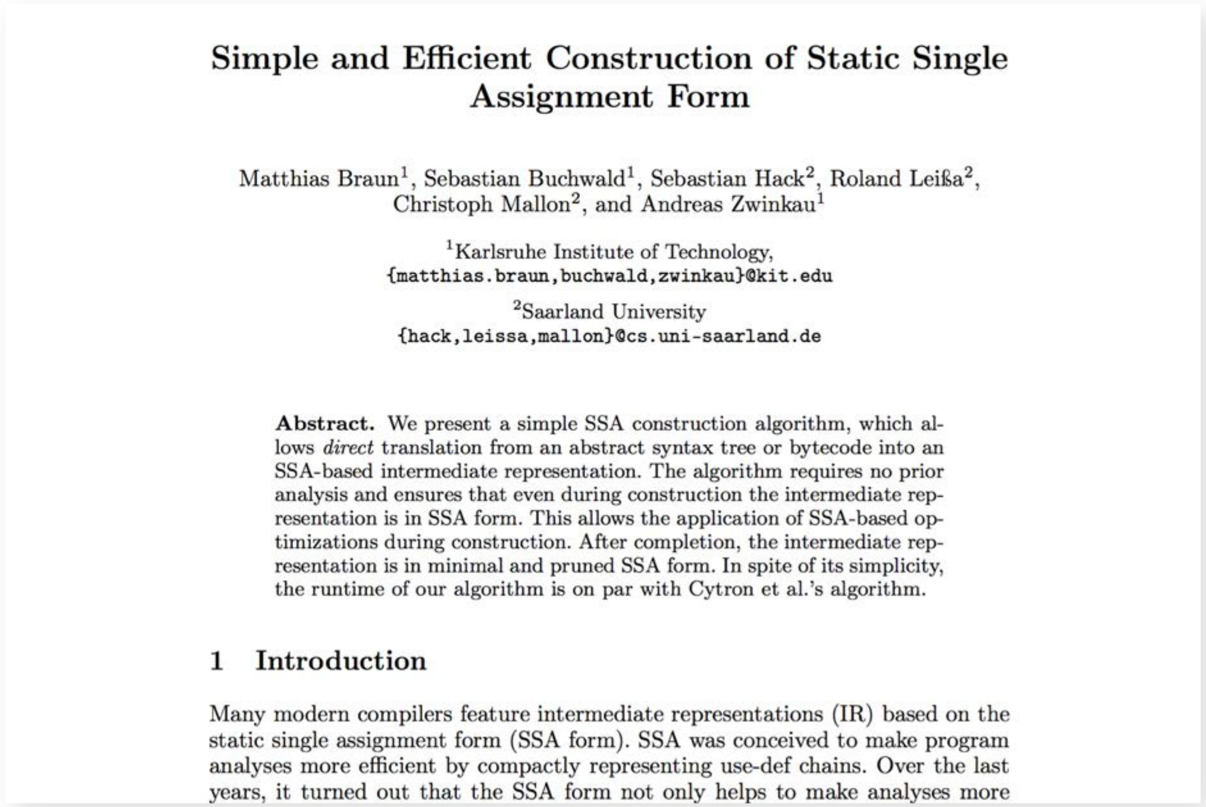
```
00000074  JUMPDEST
{ Falls through into kill() }
```

Rattle

Single Static Assignment form

- Values are assigned once
- Values get implicitly "versioned"
- Values can be used multiple times

```
%1 = ADD(%0, #1)
%2 = SUB(%0, #1)
%3 = CMP(%1, %2)
```



Simple and Efficient Construction of Static Single Assignment Form

Matthias Braun[1], Sebastian Buchwald[1], Sebastian Hack[2], Roland Leißa[2], Christoph Mallon[2], and Andreas Zwinkau[1]

[1]Karlsruhe Institute of Technology, {matthias.braun,buchwald,zwinkau}@kit.edu

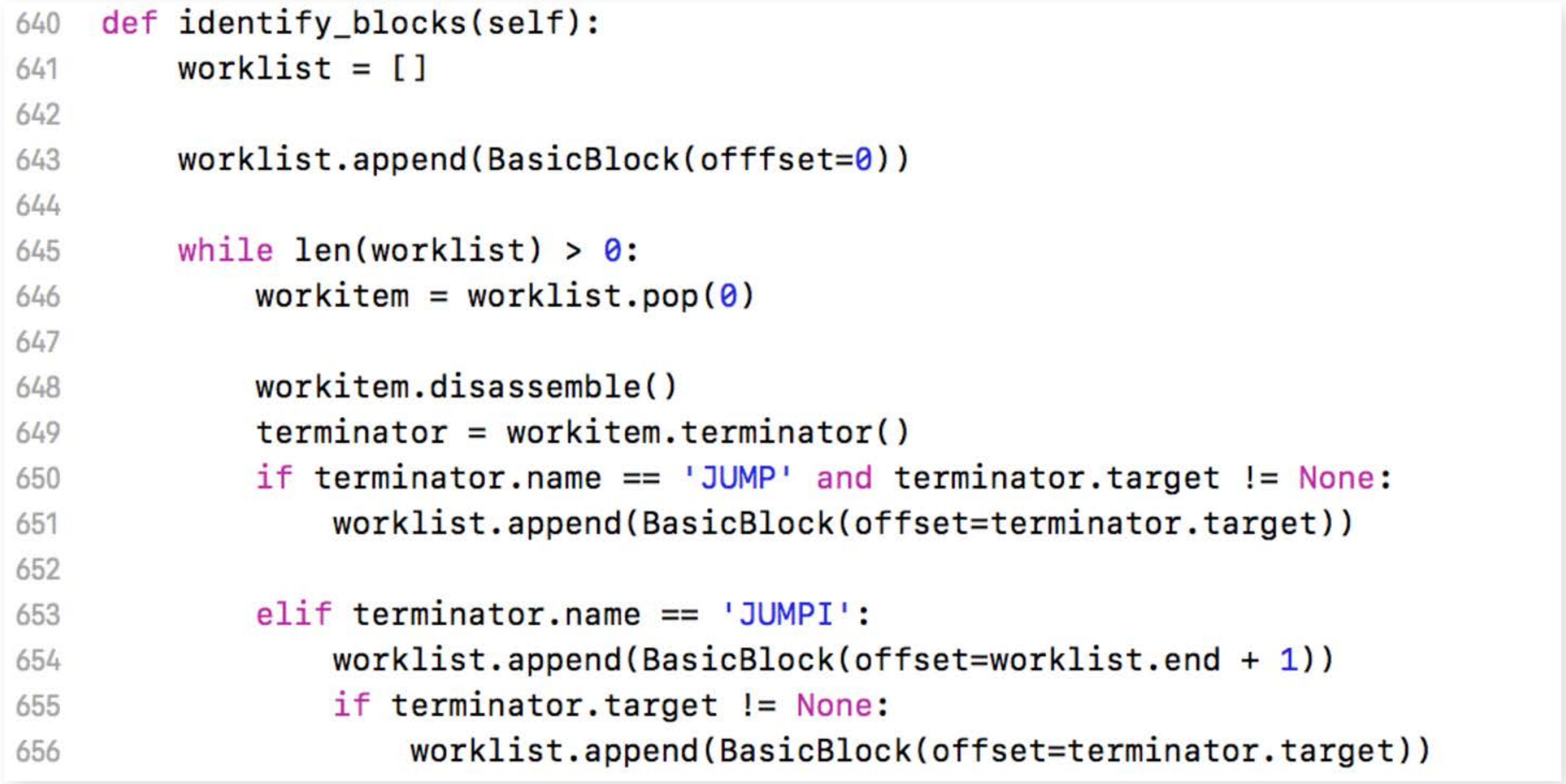
[2]Saarland University {hack,leissa,mallon}@cs.uni-saarland.de

Abstract. We present a simple SSA construction algorithm, which allows *direct* translation from an abstract syntax tree or bytecode into an SSA-based intermediate representation. The algorithm requires no prior analysis and ensures that even during construction the intermediate representation is in SSA form. This allows the application of SSA-based optimizations during construction. After completion, the intermediate representation is in minimal and pruned SSA form. In spite of its simplicity, the runtime of our algorithm is on par with Cytron et al.'s algorithm.

1 Introduction

Many modern compilers feature intermediate representations (IR) based on the static single assignment form (SSA form). SSA was conceived to make program analyses more efficient by compactly representing use-def chains. Over the last years, it turned out that the SSA form not only helps to make analyses more

Block Identification

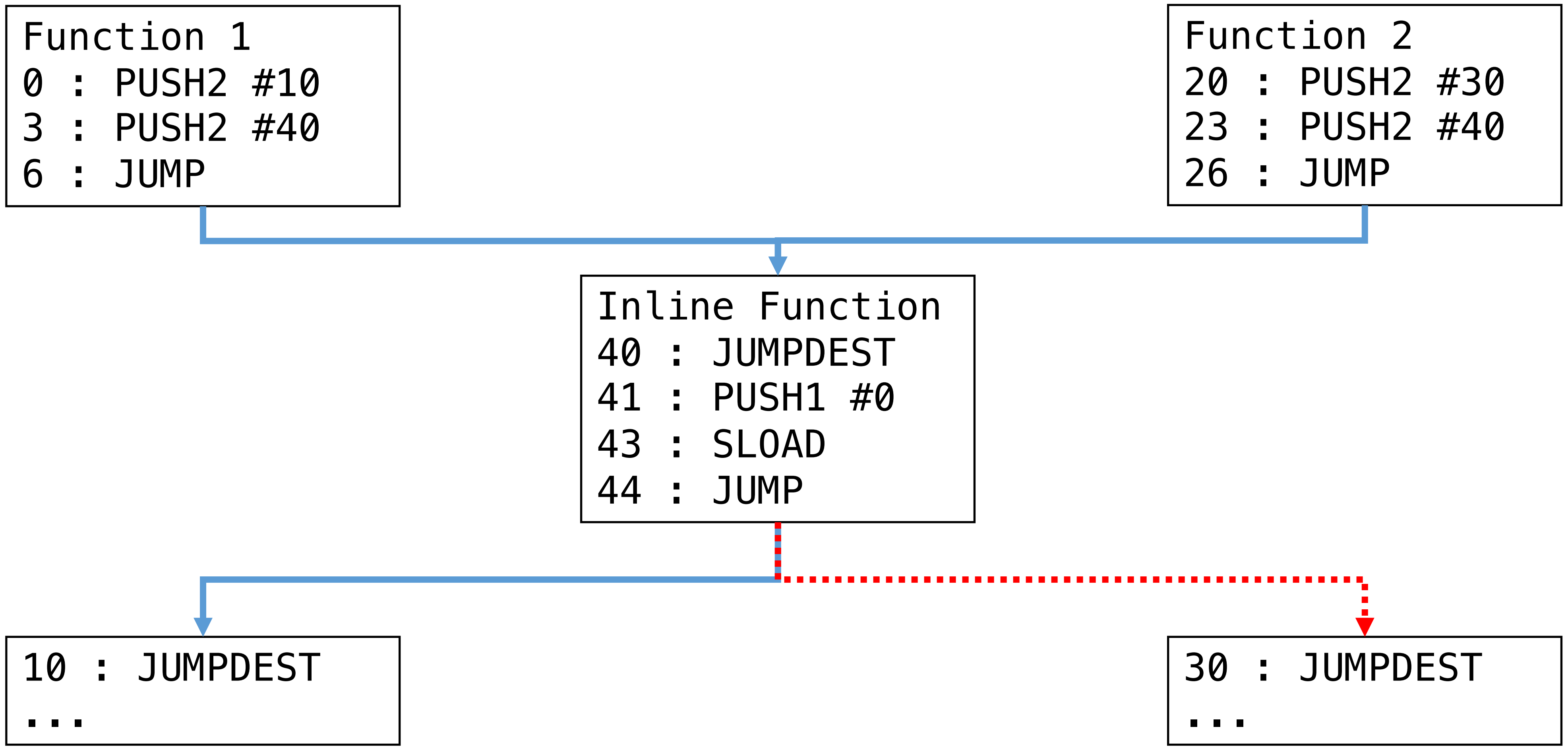


```python
def identify_blocks(self):
    worklist = []

    worklist.append(BasicBlock(offfset=0))

    while len(worklist) > 0:
        workitem = worklist.pop(0)

        workitem.disassemble()
        terminator = workitem.terminator()
        if terminator.name == 'JUMP' and terminator.target != None:
            worklist.append(BasicBlock(offset=terminator.target))

        elif terminator.name == 'JUMPI':
            worklist.append(BasicBlock(offset=worklist.end + 1))
            if terminator.target != None:
                worklist.append(BasicBlock(offset=terminator.target))
```

Control Flow Graph Recovery

1. Identify blocks
2. Link blocks
3. Trace stack
4. Replace EVM Instructions with SSA Instructions
5. Skip DUP, SWAP, POP, PUSH -- but perform their stack actions
6. Success!

Internal Calls



```
Function 1
0 : PUSH2 #10
3 : PUSH2 #40
6 : JUMP
```

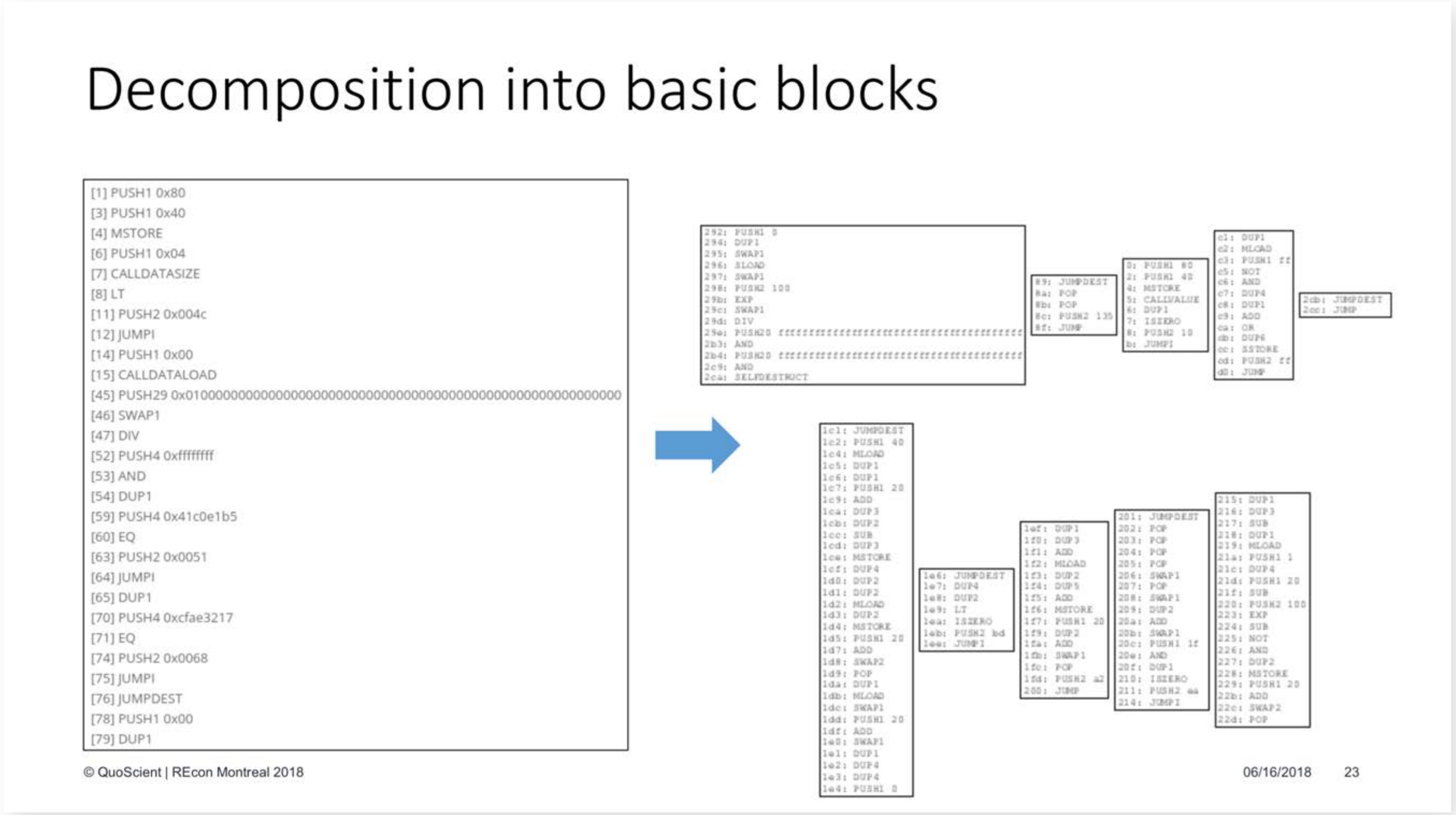
```
Function 2
20 : PUSH2 #30
23 : PUSH2 #40
26 : JUMP
```

```
Inline Function
40 : JUMPDEST
41 : PUSH1 #0
43 : SLOAD
44 : JUMP
```

```
10 : JUMPDEST
...
```

```
30 : JUMPDEST
...
```

Take 2 (or really 5)



Decomposition into basic blocks

```
[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[6] PUSH1 0x04
[7] CALLDATASIZE
[8] LT
[11] PUSH2 0x004c
[12] JUMPI
[14] PUSH1 0x00
[15] CALLDATALOAD
[45] PUSH29 0x0100000000000000000000000000000000000000000000000000000000
[46] SWAP1
[47] DIV
[52] PUSH4 0xffffffff
[53] AND
[54] DUP1
[59] PUSH4 0x41c0e1b5
[60] EQ
[63] PUSH2 0x0051
[64] JUMPI
[65] DUP1
[70] PUSH4 0xcfae3217
[71] EQ
[74] PUSH2 0x0068
[75] JUMPI
[76] JUMPDEST
[78] PUSH1 0x00
[79] DUP1
```

```
292: PUSH1 0
294: DUP1
295: SWAP1
296: SLOAD
297: SWAP1
298: PUSH2 100
29b: EXP
29c: SWAP1
29d: DIV
29e: PUSH20 ffffffffffffffffffffffffffffffffffffffff
2b3: AND
2b4: PUSH20 ffffffffffffffffffffffffffffffffffffffff
2c9: AND
2ca: SELFDESTRUCT
```

```
89: JUMPDEST
8a: POP
8b: POP
8c: PUSH2 135
8f: JUMP
```

```
0: PUSH1 80
2: PUSH1 40
4: MSTORE
5: CALLVALUE
6: DUP1
7: ISZERO
8: PUSH2 10
b: JUMPI
```

```
c1: DUP1
c2: MLOAD
c3: PUSH1 ff
c5: NOT
c6: AND
c7: DUP4
c8: DUP1
c9: ADD
ca: OR
cb: DUP6
cc: SSTORE
cd: PUSH2 ff
d0: JUMP
```

```
2db: JUMPDEST
2cc: JUMP
```

```
1c1: JUMPDEST
1c2: PUSH1 40
1c4: MLOAD
1c5: DUP1
1c6: DUP1
1c7: PUSH1 20
1c9: ADD
1ca: DUP3
1cb: DUP2
1cc: SUB
1cd: DUP3
1ce: MSTORE
1cf: DUP4
1d0: DUP2
1d1: DUP2
1d2: MLOAD
1d3: DUP2
1d4: MSTORE
1d5: PUSH1 20
1d7: ADD
1d8: SWAP2
1d9: POP
1da: DUP1
1db: MLOAD
1dc: SWAP1
1dd: PUSH1 20
1df: ADD
1e0: SWAP1
1e1: DUP1
1e2: DUP4
1e3: DUP4
1e4: PUSH1 0
```

```
1e6: JUMPDEST
1e7: DUP4
1e8: DUP2
1e9: LT
1ea: ISZERO
1eb: PUSH2 bd
1ee: JUMPI
```

```
1ef: DUP1
1f0: DUP3
1f1: ADD
1f2: MLOAD
1f3: DUP2
1f4: DUP5
1f5: ADD
1f6: MSTORE
1f7: PUSH1 20
1f9: DUP2
1fa: ADD
1fb: SWAP1
1fc: POP
1fd: PUSH2 a2
200: JUMP
```

```
201: JUMPDEST
202: POP
203: POP
204: POP
205: POP
206: SWAP1
207: POP
208: SWAP1
209: DUP2
20a: ADD
20b: SWAP1
20c: PUSH1 1f
20e: AND
20f: DUP1
210: ISZERO
211: PUSH2 ea
214: JUMPI
```

```
215: DUP1
216: DUP3
217: SUB
218: DUP1
219: MLOAD
21a: PUSH1 1
21c: DUP4
21d: PUSH1 20
21f: SUB
220: PUSH2 100
223: EXP
224: SUB
225: NOT
226: AND
227: DUP2
228: MSTORE
229: PUSH1 20
22b: ADD
22c: SWAP2
22d: POP
```

© QuoScient | REcon Montreal 2018

Third (9th) and final attempt

- Use Linear Sweep
- Pre-fill the stack
- Process blocks in isolation
- Keep discovered edges out of band, restart when new edges are discovered

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
0: <Unresolved SP: -1>
1: <Unresolved SP: -2>
2: <Unresolved SP: -3>
3: <Unresolved SP: -4>
4: <Unresolved SP: -5>
5: <Unresolved SP: -6>
6: <Unresolved SP: -7>
7: <Unresolved SP: -8>
8: <Unresolved SP: -9>
```

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
%0 = CALLER()
```

```
0: %0
1: <Unresolved SP: -1>
2: <Unresolved SP: -2>
3: <Unresolved SP: -3>
4: <Unresolved SP: -4>
5: <Unresolved SP: -5>
6: <Unresolved SP: -6>
7: <Unresolved SP: -7>
8: <Unresolved SP: -8>
9: <Unresolved SP: -9>
```

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
%0 = CALLER()
```

```
0: <Unresolved SP: -1>
1: %0
2: <Unresolved SP: -1>
3: <Unresolved SP: -2>
4: <Unresolved SP: -3>
5: <Unresolved SP: -4>
6: <Unresolved SP: -5>
7: <Unresolved SP: -6>
8: <Unresolved SP: -7>
9: <Unresolved SP: -8>
```

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
%0 = CALLER()
%1 = PUSH(#0)
```

```
0: %1
1: <Unresolved SP: -1>
2: %0
3: <Unresolved SP: -1>
4: <Unresolved SP: -2>
5: <Unresolved SP: -3>
6: <Unresolved SP: -4>
7: <Unresolved SP: -5>
8: <Unresolved SP: -6>
9: <Unresolved SP: -7>
```

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
%0 = CALLER()
%1 = PUSH(#0)
%2 = ADD(<Unresolved SP: -1>, %1)
```

```
0: %5
1: %0
2: <Unresolved SP: -1>
3: <Unresolved SP: -2>
4: <Unresolved SP: -3>
5: <Unresolved SP: -4>
6: <Unresolved SP: -5>
7: <Unresolved SP: -6>
8: <Unresolved SP: -7>
9: <Unresolved SP: -8>
```

SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

```
%0 = CALLER()
%1 = PUSH(#0)
%2 = ADD(<Unresolved SP: -1>, %1)
%3 = PUSH(#0)
%4 = PUSH(#100)
%5 = EXP(%4, %3)
```

```
0: %5
1: %0
2: <Unresolved SP: -1>
3: <Unresolved SP: -2>
4: <Unresolved SP: -3>
5: <Unresolved SP: -4>
6: <Unresolved SP: -5>
7: <Unresolved SP: -6>
8: <Unresolved SP: -7>
9: <Unresolved SP: -8>
```

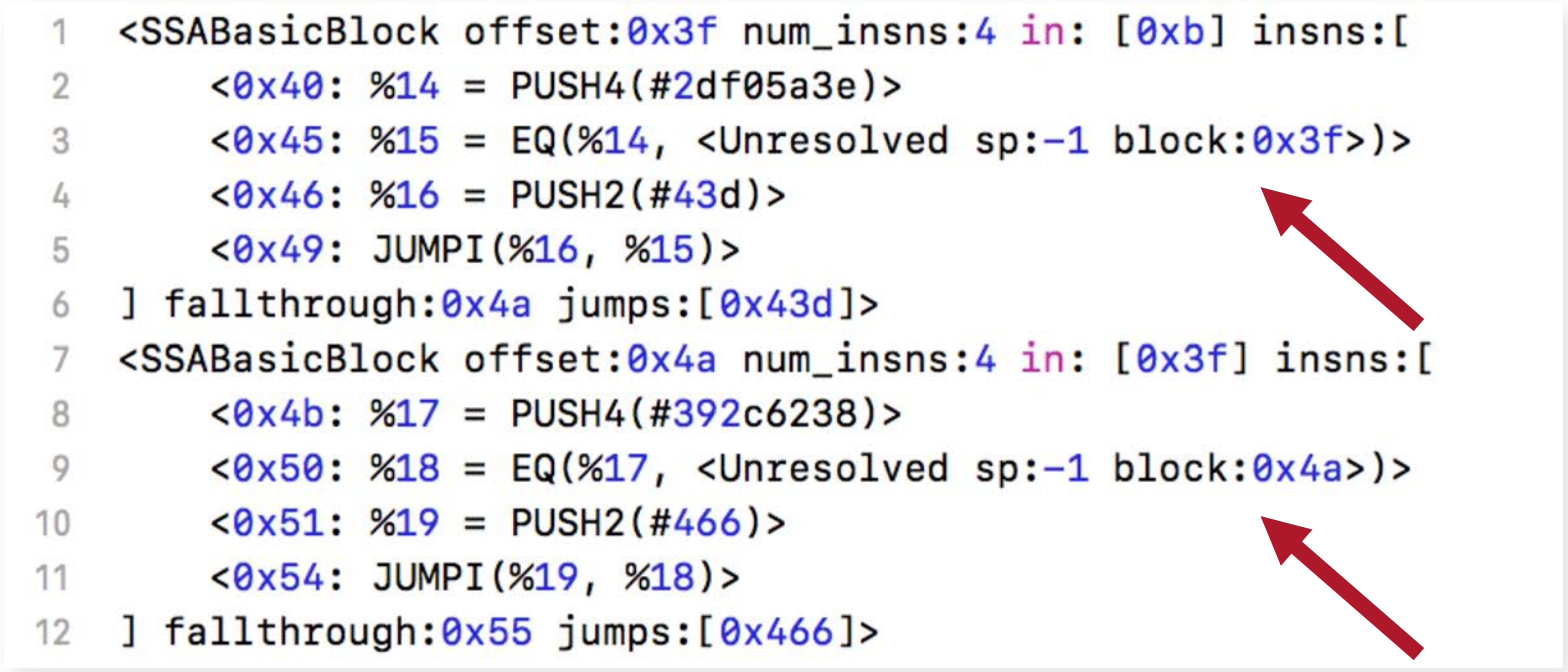
SSA Recovery

```
JUMPDEST
CALLER
DUP2
PUSH1 #0
ADD
PUSH1 #0
PUSH2 #100
EXP
DUP2
SLOAD
DUP2
PUSH20 #ffffffff..ffffffff
MUL
NOT
AND
SWAP1
DUP4
```

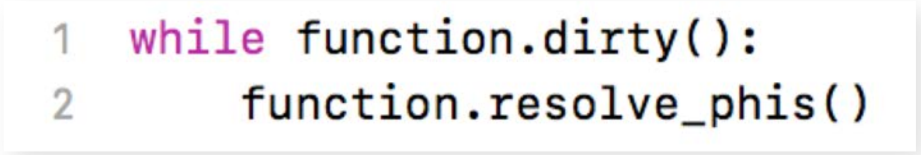
```
%0 = CALLER()
%1 = PUSH(#0)
%2 = ADD(<Unresolved SP: -1>, %1)
%3 = PUSH(#0)
%4 = PUSH(#100)
%5 = EXP(%4, %3)
%6 = SLOAD(%2)
%7 = PUSH(#ffffffff..ffffffff)
%8 = MUL(%7, %5)
%9 = NOT(%8)
%10 = AND(%9, %8)
```

```
0: %10
1: %0
2: <Unresolved SP: -1>
3: <Unresolved SP: -2>
4: <Unresolved SP: -3>
5: <Unresolved SP: -4>
6: <Unresolved SP: -5>
7: <Unresolved SP: -6>
8: <Unresolved SP: -7>
9: <Unresolved SP: -8>
```

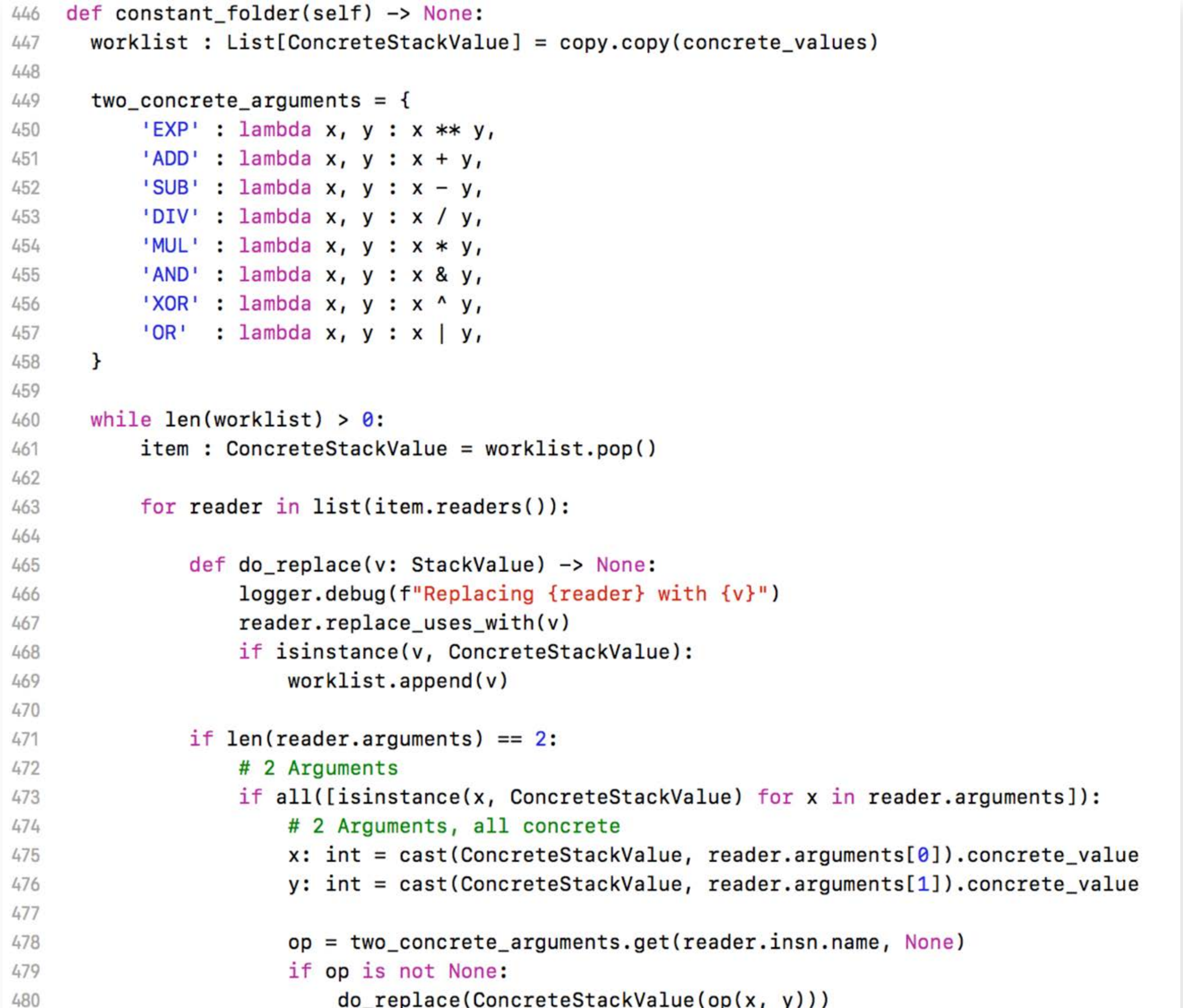
Lift to SSA



```
<SSABasicBlock offset:0x3f num_insns:4 in: [0xb] insns:[
    <0x40: %14 = PUSH4(#2df05a3e)>
    <0x45: %15 = EQ(%14, <Unresolved sp:-1 block:0x3f>)>
    <0x46: %16 = PUSH2(#43d)>
    <0x49: JUMPI(%16, %15)>
] fallthrough:0x4a jumps:[0x43d]>
<SSABasicBlock offset:0x4a num_insns:4 in: [0x3f] insns:[
    <0x4b: %17 = PUSH4(#392c6238)>
    <0x50: %18 = EQ(%17, <Unresolved sp:-1 block:0x4a>)>
    <0x51: %19 = PUSH2(#466)>
    <0x54: JUMPI(%19, %18)>
] fallthrough:0x55 jumps:[0x466]>
```

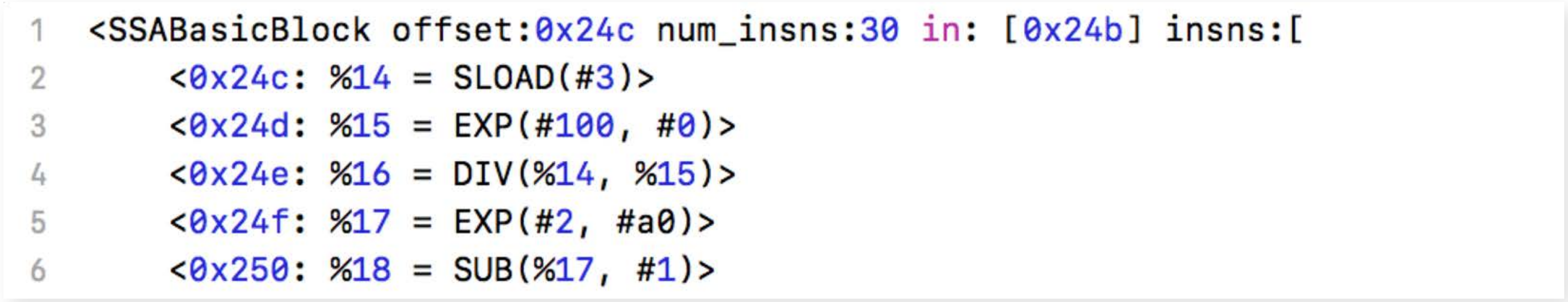


```
while function.dirty():
    function.resolve_phis()
```

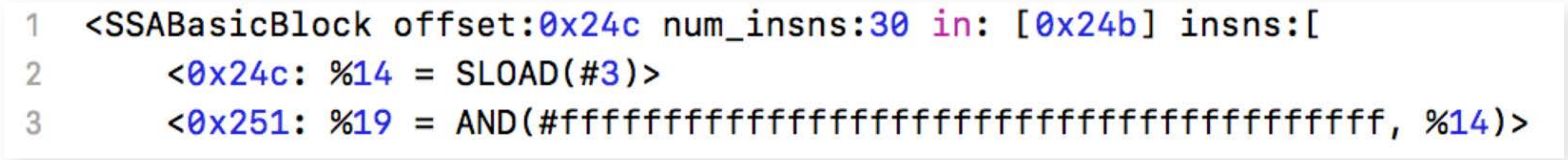


```python
def constant_folder(self) -> None:
  worklist : List[ConcreteStackValue] = copy.copy(concrete_values)

  two_concrete_arguments = {
      'EXP' : lambda x, y : x ** y,
      'ADD' : lambda x, y : x + y,
      'SUB' : lambda x, y : x - y,
      'DIV' : lambda x, y : x / y,
      'MUL' : lambda x, y : x * y,
      'AND' : lambda x, y : x & y,
      'XOR' : lambda x, y : x ^ y,
      'OR'  : lambda x, y : x | y,
  }

  while len(worklist) > 0:
      item : ConcreteStackValue = worklist.pop()

      for reader in list(item.readers()):

          def do_replace(v: StackValue) -> None:
              logger.debug(f"Replacing {reader} with {v}")
              reader.replace_uses_with(v)
              if isinstance(v, ConcreteStackValue):
                  worklist.append(v)

          if len(reader.arguments) == 2:
              # 2 Arguments
              if all([isinstance(x, ConcreteStackValue) for x in reader.arguments]):
                  # 2 Arguments, all concrete
                  x: int = cast(ConcreteStackValue, reader.arguments[0]).concrete_value
                  y: int = cast(ConcreteStackValue, reader.arguments[1]).concrete_value

                  op = two_concrete_arguments.get(reader.insn.name, None)
                  if op is not None:
                      do_replace(ConcreteStackValue(op(x, y)))
```

SSA Optimized

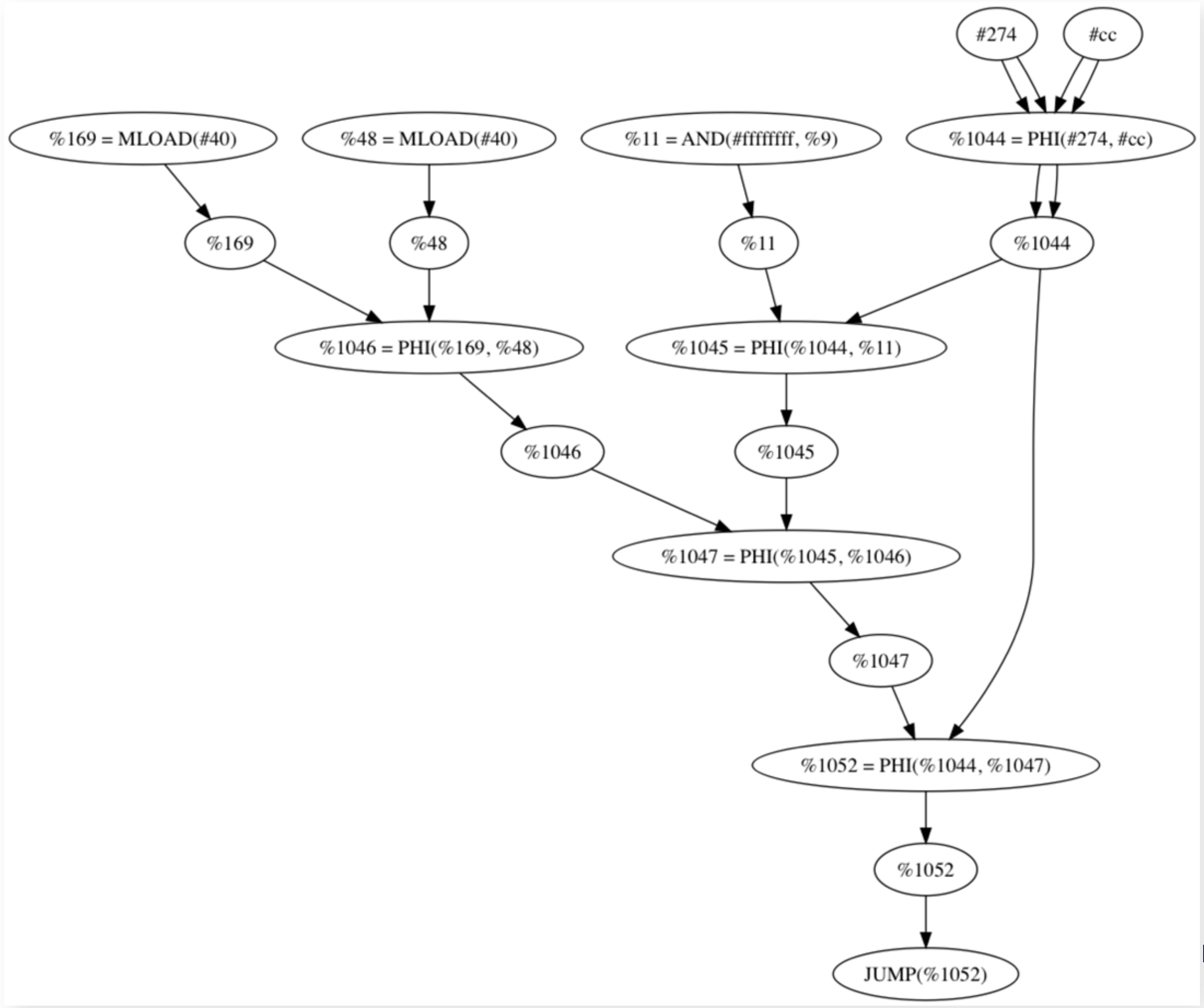


```
<SSABasicBlock offset:0x24c num_insns:30 in: [0x24b] insns:[
    <0x24c: %14 = SLOAD(#3)>
    <0x24d: %15 = EXP(#100, #0)>
    <0x24e: %16 = DIV(%14, %15)>
    <0x24f: %17 = EXP(#2, #a0)>
    <0x250: %18 = SUB(%17, #1)>
```



```
<SSABasicBlock offset:0x24c num_insns:30 in: [0x24b] insns:[
    <0x24c: %14 = SLOAD(#3)>
    <0x251: %19 = AND(#ffffffffffffffffffffffffffffffffffffffff, %14)>
```

Def Use and Use Def Graphs



Memory and Storage Recovery

```
$ python3 rattle-cli.py -O --input inputs/kingofether/KingOfTheEtherThrone.bin

Storage Locations: [0, 1, 2, 3, 4, 5, 6]
Memory Locations: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 64]

...

Function pastMonarchs(uint256) storage:
        Analyzing Storage Location: 6
                0x54f: %365 = SLOAD(#6)
                0x553: SSTORE(#6, %366)
                0xa42: %695 = SLOAD(#6)
                0xb3a: %790 = SLOAD(#6)
```

Call Analysis

```
[+] Contract can send ether from following functions:

        - _dispatch

                %53 = CALL(%46, %42, %44, %51, %52, %51, #0)
                        To:     %42 = AND(#ffffffffffffffffffffffffffffffffffffffff, %40)
                        Value:  %44 = CALLVALUE()

                %75 = CALL(%68, %64, #de0b6b3a7640000, %73, %74, %73, #0)
                        To:     %64 = AND(#ffffffffffffffffffffffffffffffffffffffff, %62)
                        Value:  #de0b6b3a7640000 1.0ETH

                %262 = CALL(%255, %251, #8ac7230489e80000, %260, %261, %260, #0)
                        To:     %251 = AND(#ffffffffffffffffffffffffffffffffffffffff, %249)
                        Value:  #8ac7230489e80000 10.0ETH
```

Function Identification

```
Identified Functions:
        _dispatch
                argument offsets:[(0, 32)]

        balance()
                argument offsets:[]

        _unknown_0xf8626af8()
                argument offsets:[(4, 36)]

        kill()
                argument offsets:[]

        _unknown_0xa840dda9()
                argument offsets:[]

        _fallthrough
                argument offsets:[]
```

Demo

Releasing Rattle

- If you do smart-contract work and would like early access, we have a form: https://trailofbits.wufoo.com/forms/m1qfujq31qyj9ee/

- Watch our **GitHub** (github.com/trailofbits) or our **Twitter** (@trailofbits) for its release!

Questions?

Ryan Stortz

Principal Security Researcher

ryan@trailofbits.com
@withzombies