Erays: Reverse Engineering Ethereum's Opaque Smart Contracts

Yi Zhou, Deepak Kumar, Surya Bakshi,
Joshua Mason, Andrew Miller, Michael Bailey
University of Illinois Urbana-Champaign

Introduction:

Ethereum

Introduction:

Ethereum Smart Contracts

- Computer programs on the blockchain
- Written in high level language (Solidity)
- Executed in the Ethereum Virtual Machine (EVM)

Solidity Code

```
contract dummy {
    uint s;

    function foo(uint a) public returns (uint) {
        while (a < s) {
            if (a > 10) {
                a += 1;
            } else {
                a += 2;
            }
        }
        return a;
    }
}
```

Compiled Contract

```
608060405260043610603e5763ffffffff7c0100000000000000000000000
0000000000000000000000000000000006000350416632fbebd3881146043
575b600080fd5b348015604e57600080fd5b5060586004356064565b60408
051918252519081900360200190f35b60005b600054821015609357600a82
111560885760018201915060608f565b6002820191505b606d565b50905600a
165627a7a7230582095826fc9f61669f3d0fe36966d60c64042dec36a23ac
89e6b4ebe1752f2c7ca00029
```

EVM Bytecode

```
PUSH1 0x80
PUSH1 0x40
MSTORE
PUSH1 0x04
CALLDATASIZE
LT
PUSH1 0x3e
JUMPI
PUSH4 0xffffffff
PUSH29
0x0100000000000000000000000000000000000000000000000000000000
PUSH1 0x00
CALLDATALOAD
...
```

Problem:

Opaque/proprietary contracts

- EVM bytecode is not easily understandable
- High level source code is not always available
- Contract functionality remains opaque/proprietary

Ecosystem:

How many contracts are there?

- Total Count: 1,024,886
- Unique Count: 34,328

Ecosystem:

How many contracts are opaque/proprietary?

- 10,387 Solidity Source Files Collected (from Etherscan)
- 35 Versions (v0.1.3 to v0.4.19) of Solidity Compilers Used
- 88,426 Unique Binaries Compiled

Ecosystem: Measuring Opacity

	Contracts
Total	1,024,886
Unique	34,328 (100.0%)
Unique Transparent	7,734 (22.5%)
Unique Opaque	26,594 (77.5%)

Ecosystem: Measuring Opacity

	Contracts
Total	1,024,886
Unique	34,328 (100.0%)
Unique Transparent	7,734 (22.5%)
Unique Opaque	$\boxed{26,594 \ (77.5\%)}$

Erays

Erays: System Design

1. Control Flow Graph Recovery
2. Lifting
3. Optimization
4. Aggregation
5. Control Flow Structure Recovery

Control Flow Graph Recovery

- Identify basic block boundaries

```
...
JUMPDEST
PUSH1   0x0
JUMPDEST
PUSH1   0x0
SLOAD
DUP3
LT
ISZERO
PUSH1   0x93
JUMPI
...
```

Control Flow Graph Recovery

- Identify basic block boundaries

```
...
JUMPDEST
PUSH1   0x0
JUMPDEST
PUSH1   0x0
SLOAD
DUP3
LT
ISZERO
PUSH1   0x93
JUMPI
...
```

Control Flow Graph Recovery

- Identify basic block boundaries
- Organize basic blocks into a CFG
  - Emulate the contract using a stack model
  - Explore the contract in a manner similar to Depth First Search
  - Record stack images at each block entrance

```
JUMPDEST
...
```

```
...
PUSH1 0x93
JUMPI
```

```
...
PUSH1 0x88
JUMPI
```

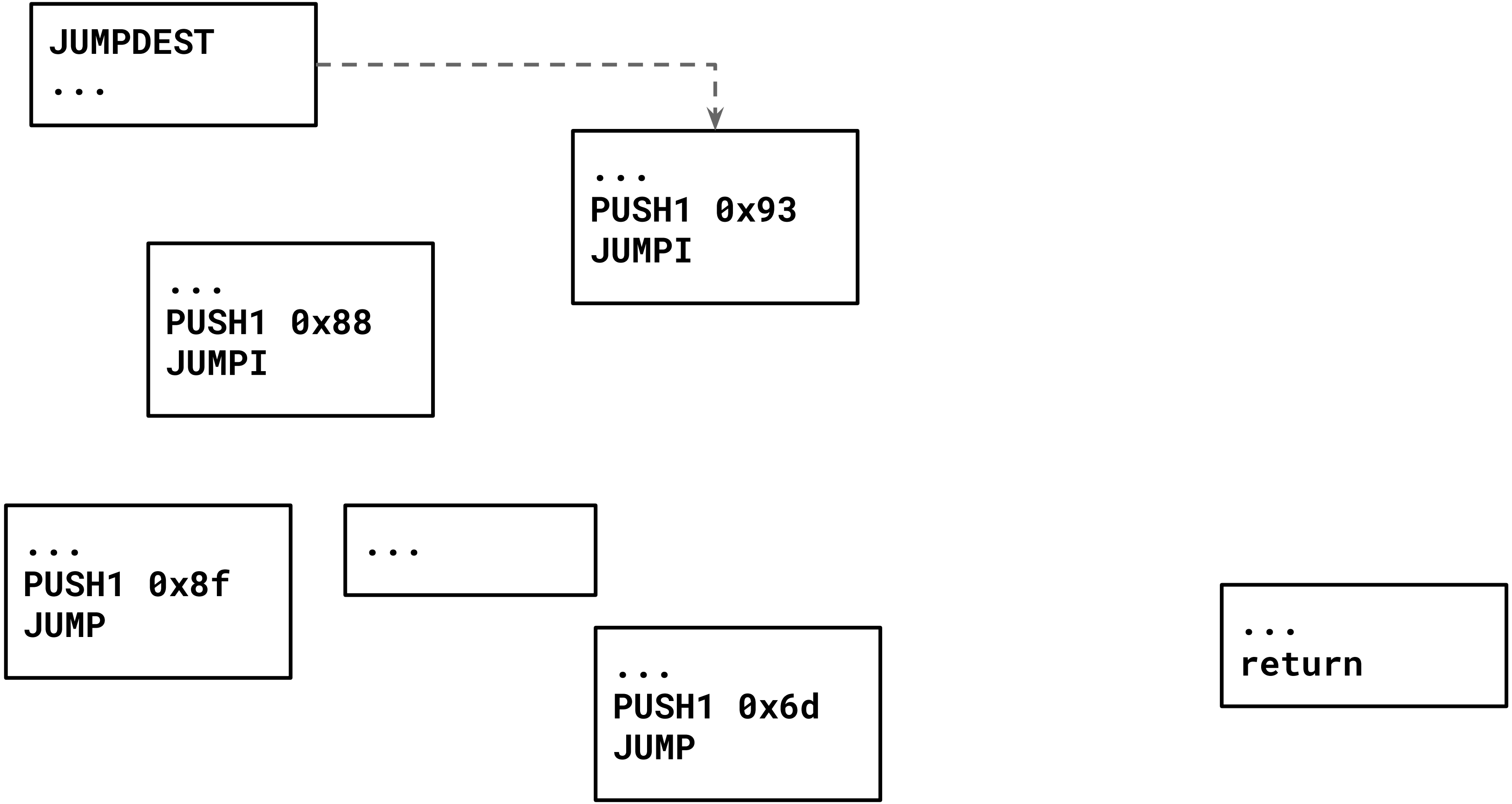
```
...
PUSH1 0x8f
JUMP
```

```
...
```

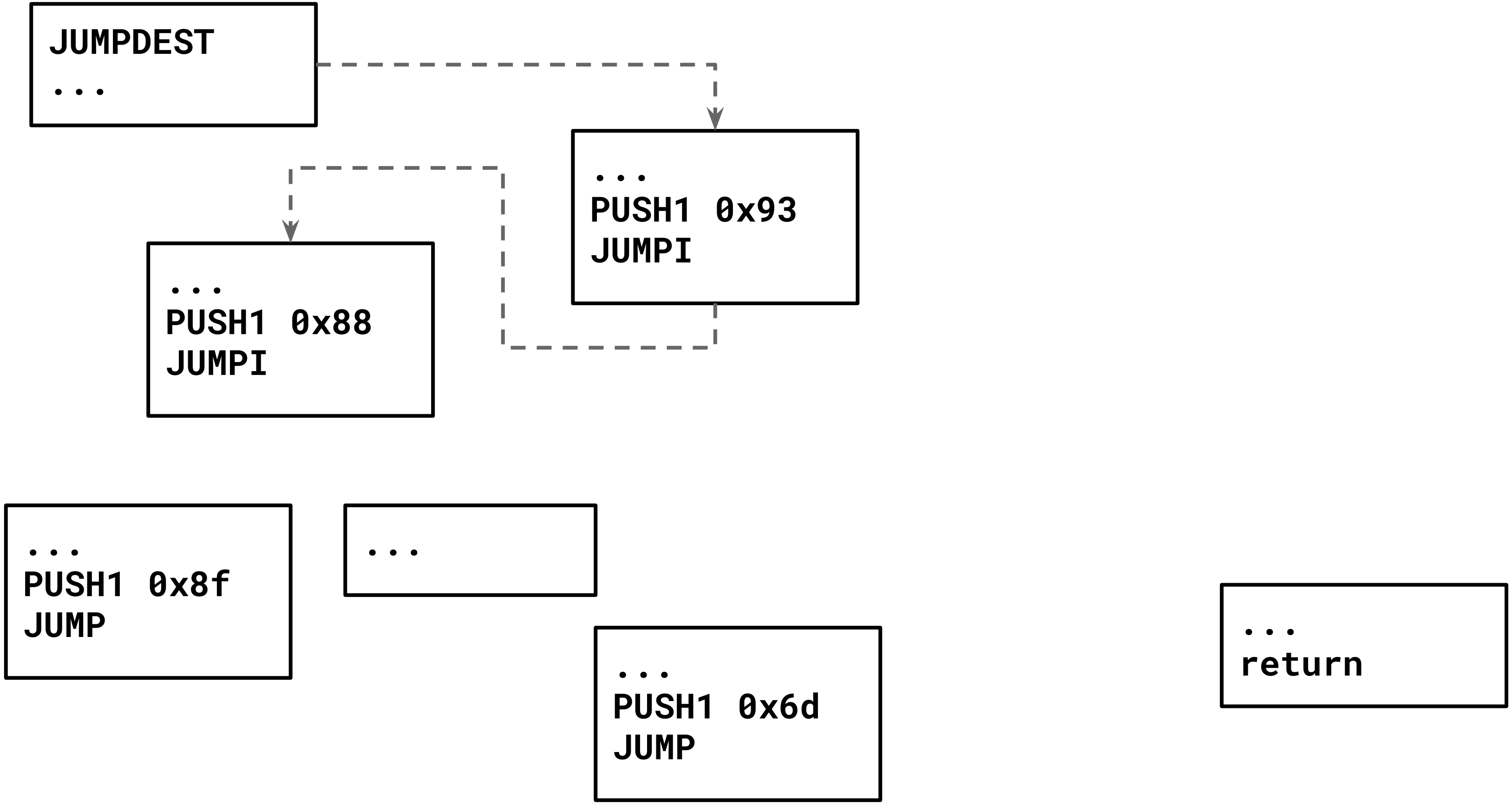
```
...
return
```

```
...
PUSH1 0x6d
JUMP
```

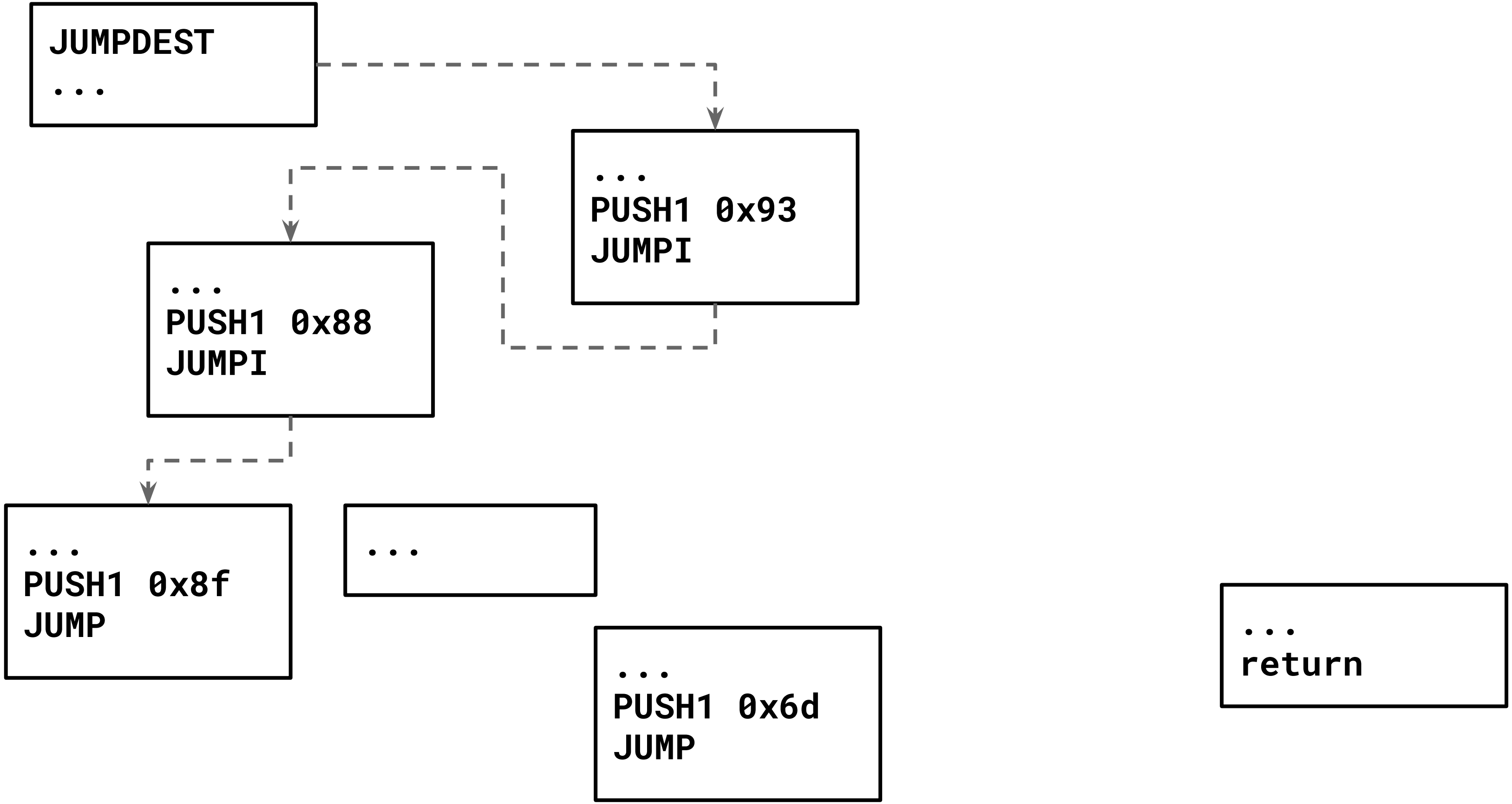
Control Flow Graph Recovery



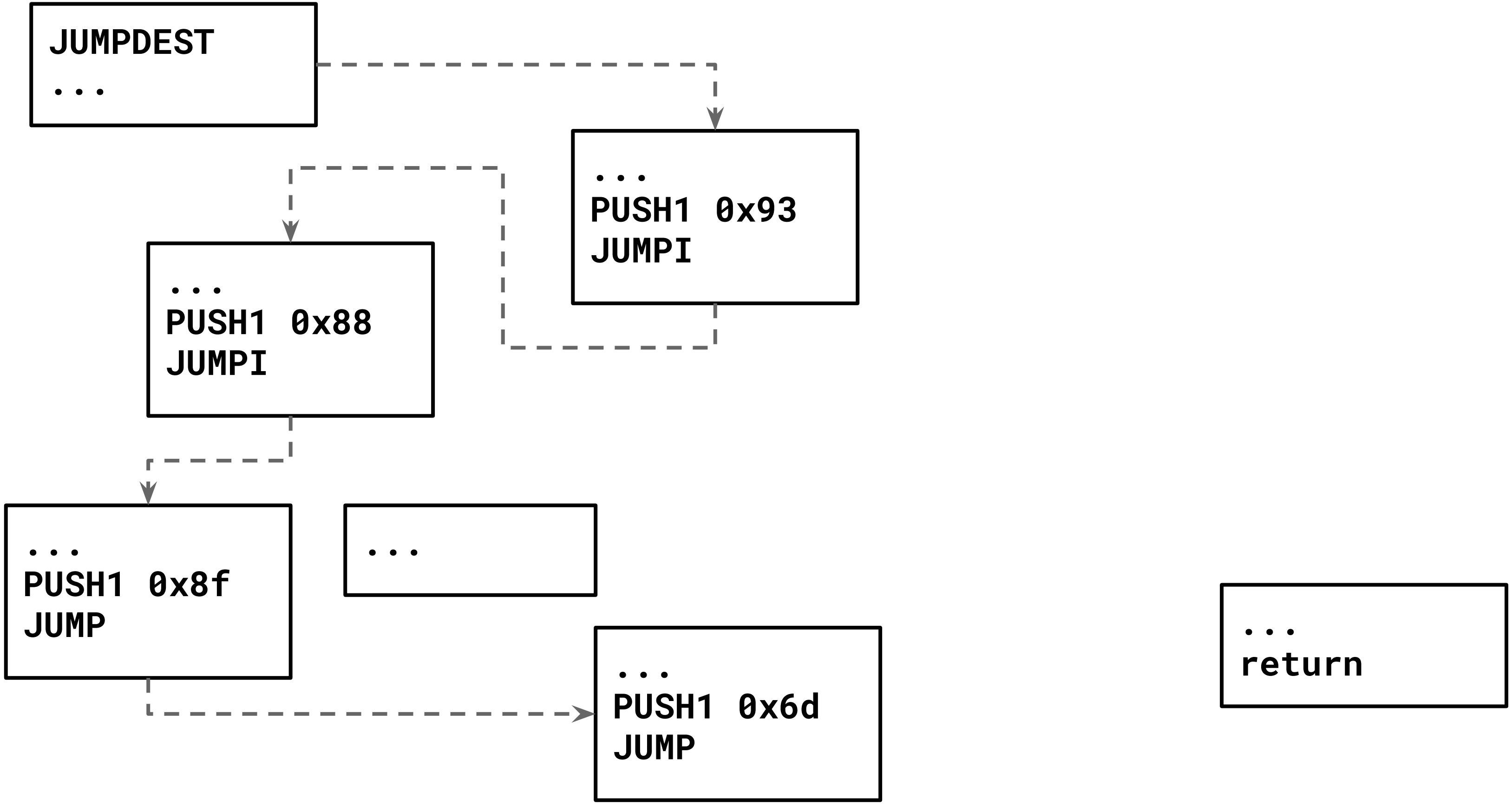
Control Flow Graph Recovery



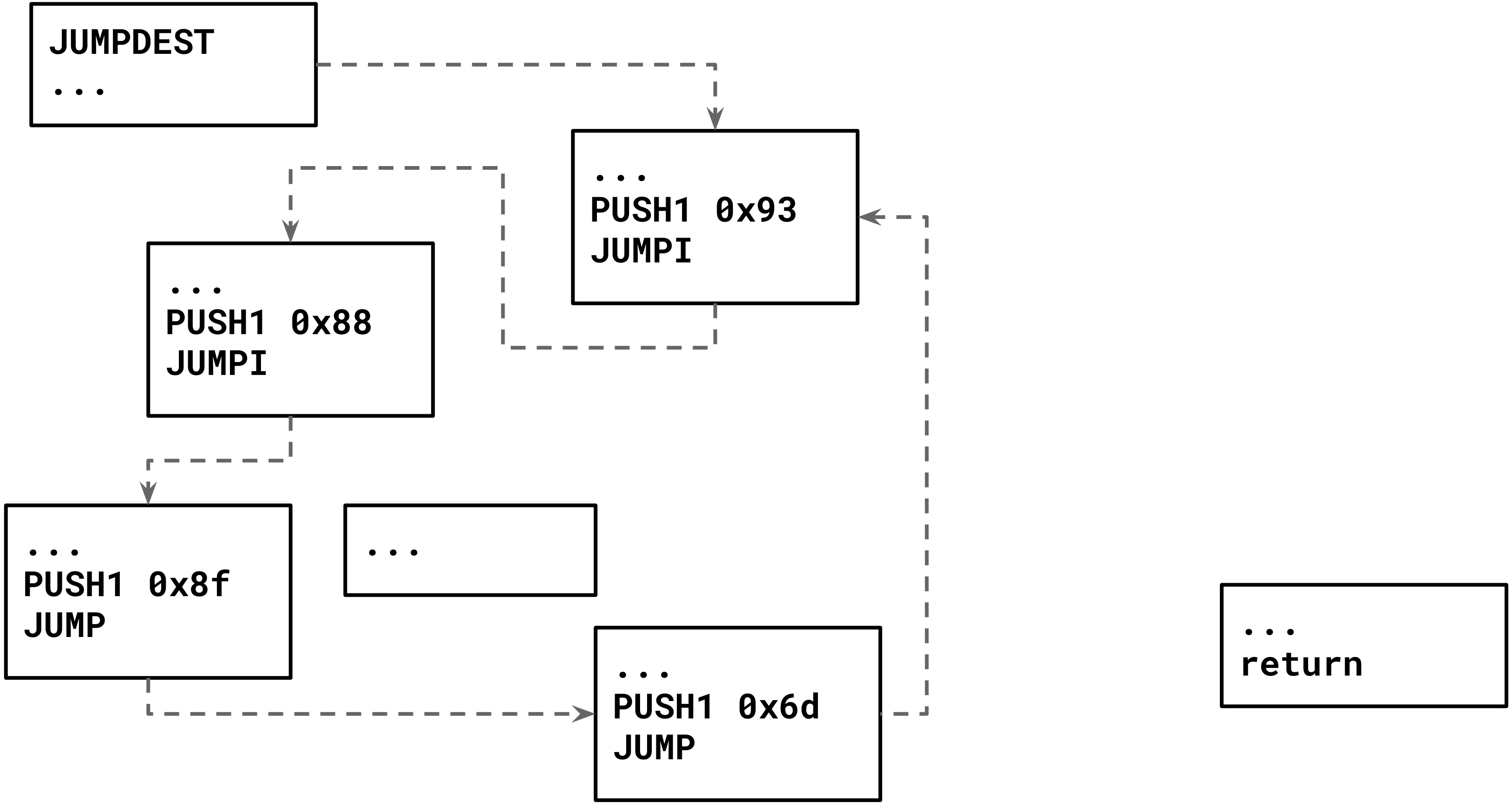
Control Flow Graph Recovery



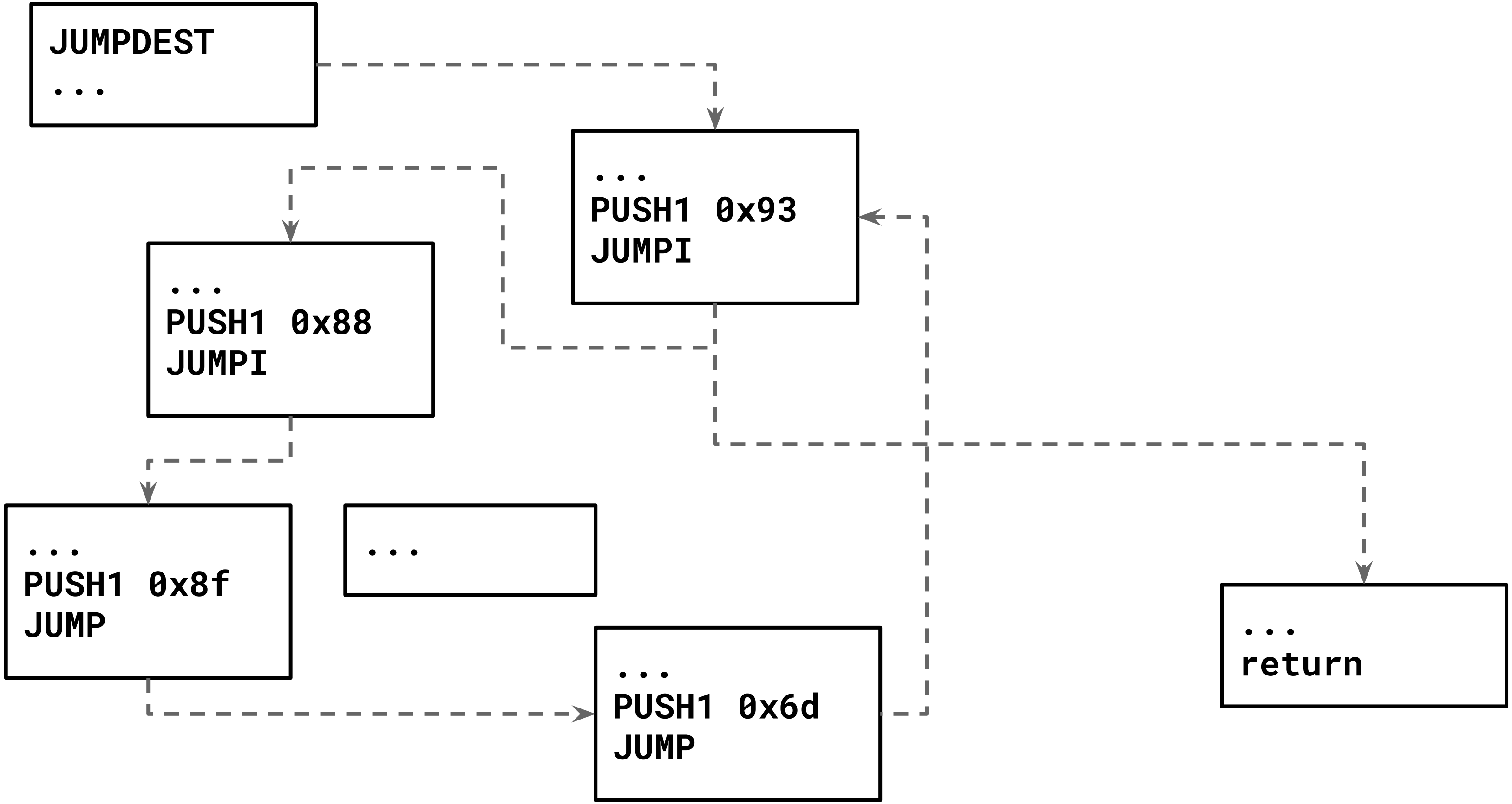
Control Flow Graph Recovery



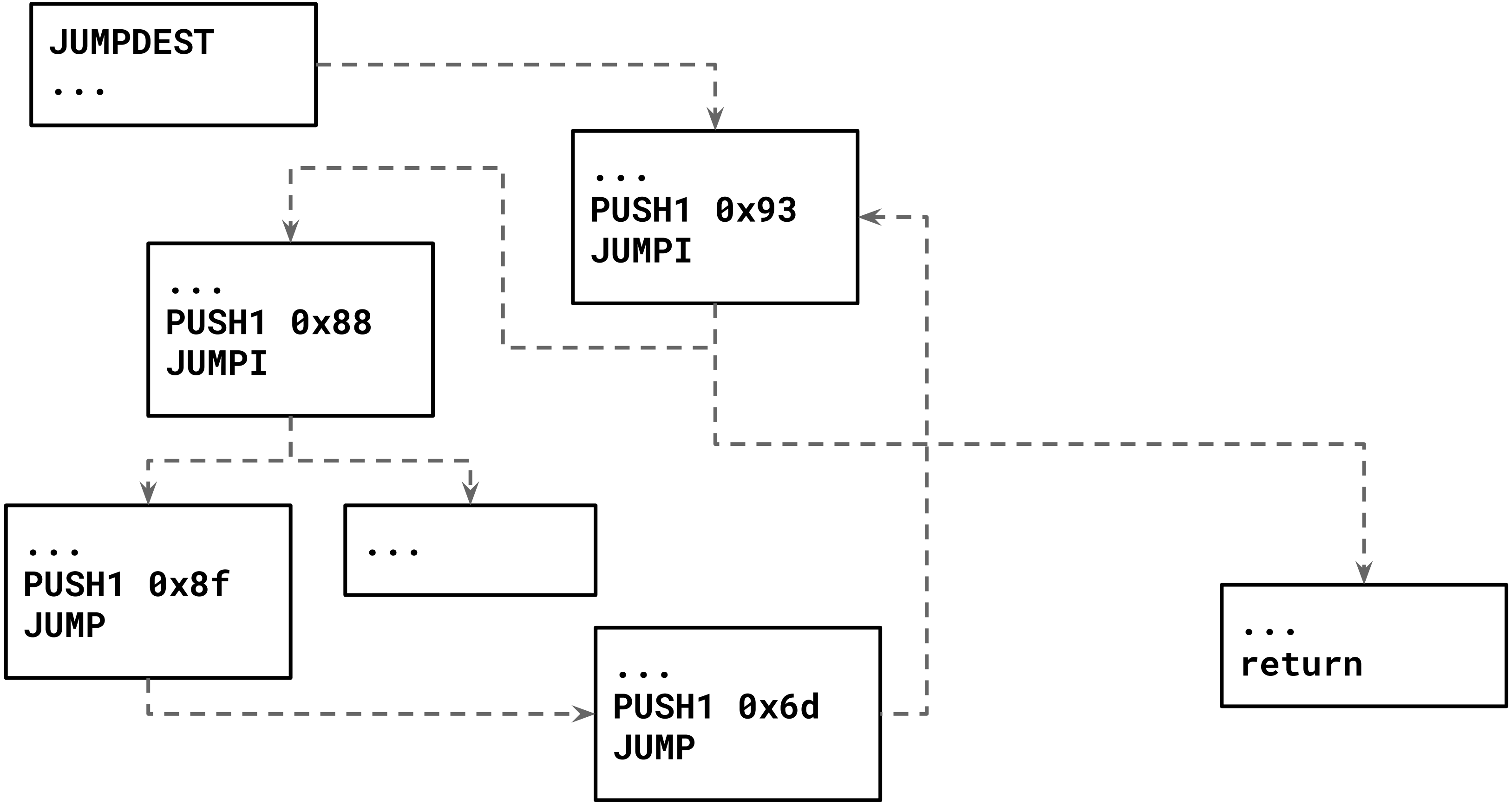
Control Flow Graph Recovery



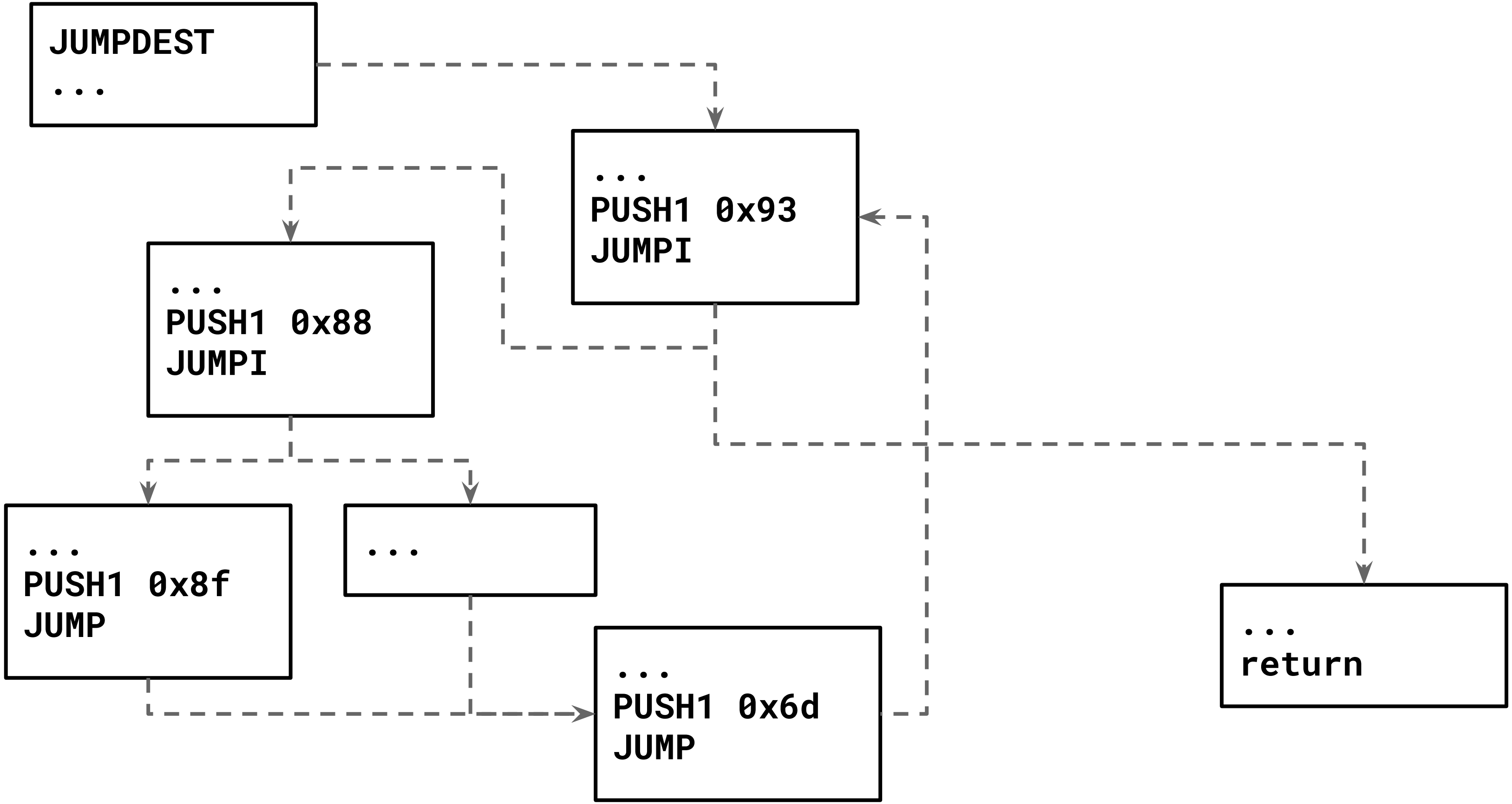
Control Flow Graph Recovery



Control Flow Graph Recovery



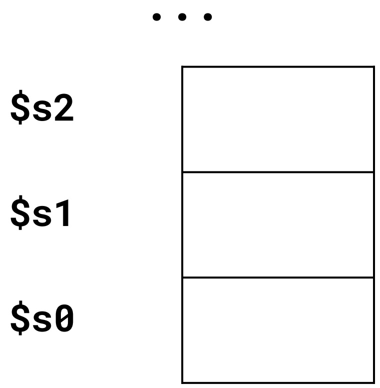
Control Flow Graph Recovery



Control Flow Graph Recovery

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers



Lifting: Stack-based to Register-based

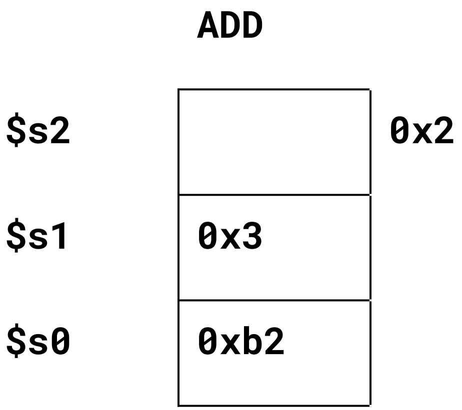
- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
    - Map stack slots to registers
    - Assign registers to each bytecode (using stack height)

ADD

$s2	0x2
$s1	0x3
$s0	0xb2

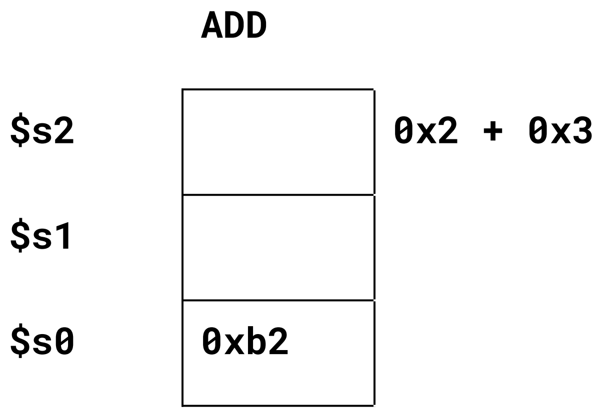
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
    - Map stack slots to registers
    - Assign registers to each bytecode (using stack height)



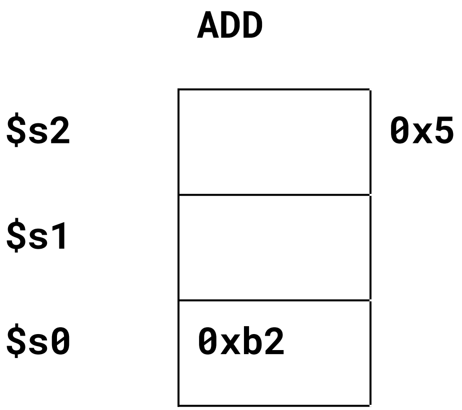
Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)



Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
    - Map stack slots to registers
    - Assign registers to each bytecode (using stack height)



Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
    - Map stack slots to registers
    - Assign registers to each bytecode (using stack height)

ADD

`$s2`	
`$s1`	`0x5`
`$s0`	`0xb2`

1 2

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
ADD $s1, $s2, $s1
```

$s2	
$s1	0x5
$s0	0xb2

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
PUSH1       0x0
SLOAD
DUP3
LT
ISZERO
PUSH1       0x93
JUMPI
```

1 → 2 → → →

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
  - Map stack slots to registers
  - Assign registers to each bytecode (using stack height)

```
MOVE          $s4, 0x0
SLOAD         $s4, [$s4]
MOVE          $s5, $s2
LT            $s4, $s5, $s4
ISZERO        $s4, $s4
MOVE          $s5, 0x93
JUMPI         $s5, $s4
```

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
- Introduce new instructions

Lifting: Stack-based to Register-based

- Convert stack-based operations into register-based representation (R. Vallee-Rai 1999)
- Introduce new instructions
  - **INTCALL, INTRET**
  - **MOVE**
  - **ASSERT**
  - **NEQ, GEQ, LEQ, SL, SR**

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)

```
MOVE        $s4, 0x0
SLOAD       $s4, [$s4]
MOVE        $s5, $s2
LT          $s4, $s5, $s4
ISZERO      $s4, $s4
MOVE        $s5, 0x93
JUMPI       $s5, $s4
```

1 → 2 → 3 → →

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation

```
MOVE        $s4, 0x0
SLOAD       $s4, [0x0]
MOVE        $s5, $s2
LT          $s4, $s5, $s4
ISZERO      $s4, $s4
MOVE        $s5, 0x93
JUMPI       0x93, $s4
```

1 → 2 → 3 → →

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation

```
MOVE        $s4, 0x0
SLOAD       $s4, [0x0]
MOVE        $s5, $s2
LT          $s4, $s2, $s4
ISZERO      $s4, $s4
MOVE        $s5, 0x93
JUMPI       0x93, $s4
```

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination

```
--
SLOAD       $s4, [0x0]
--
LT          $s4, $s2, $s4
ISZERO      $s4, $s4
--
JUMPI       0x93, $s4
```

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination
- Local optimizations

```
--
SLOAD       $s4, [0x0]
--
LT          $s4, $s2, $s4
ISZERO      $s4, $s4
--
JUMPI       0x93, $s4
```

1 2 3

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination
- Local optimizations

```
--
SLOAD        $s4, [0x0]
--
--
GEQ          $s4, $s2, $s4
--
JUMPI        0x93, $s4
```

Optimization: Removing Redundancy

- Global optimizations (1973 G. Kildall)
  - Constant propagation
  - Copy propagation
  - Dead code elimination
- Local optimizations

```
SLOAD        $s4, [0x0]
GEQ          $s4, $s2, $s4
JUMPI        0x93, $s4
```

1 → 2 → 3 → →

Aggregation: Condensing the Output

- Convert register-based instructions into three address form

```
SLOAD       $s4, [0x0]
GEQ         $s4, $s2, $s4
JUMPI       0x93, $s4
```

Aggregation: Condensing the Output

- Convert register-based instructions into three address form

```
$s4 = S[0x0]
$s4 = $s2 ≥ $s4
if ($s4) goto 0x93
```

1 → 2 → 3 → 4 →

Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
$s4 = S[0x0]
$s4 = $s2 ≥ $s4
if ($s4) goto 0x93
```

1 → 2 → 3 → 4 →

Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
--
$s4 = $s2 ≥ S[0x0]
if ($s4) goto 0x93
```

1 → 2 → 3 → 4 →

Aggregation: Condensing the Output

- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
--
--
if ($s2 ≥ S[0x0]) goto 0x93
```

Aggregation: Condensing the Output

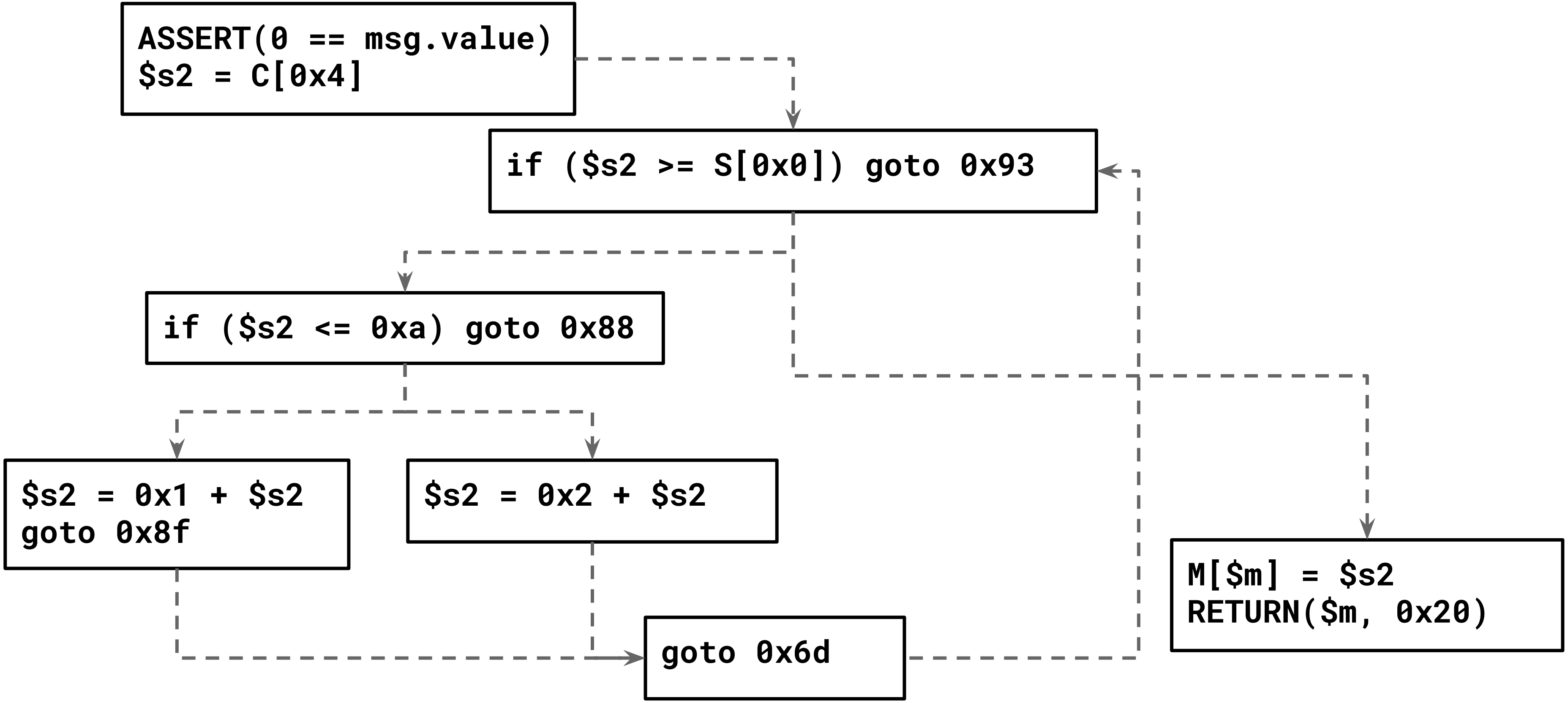
- Convert register-based instructions into three address form
- Aggregate instructions into nested expressions (R. Vallee-Rai 1999)

```
if ($s2 ≥ S[0x0]) goto 0x93
```

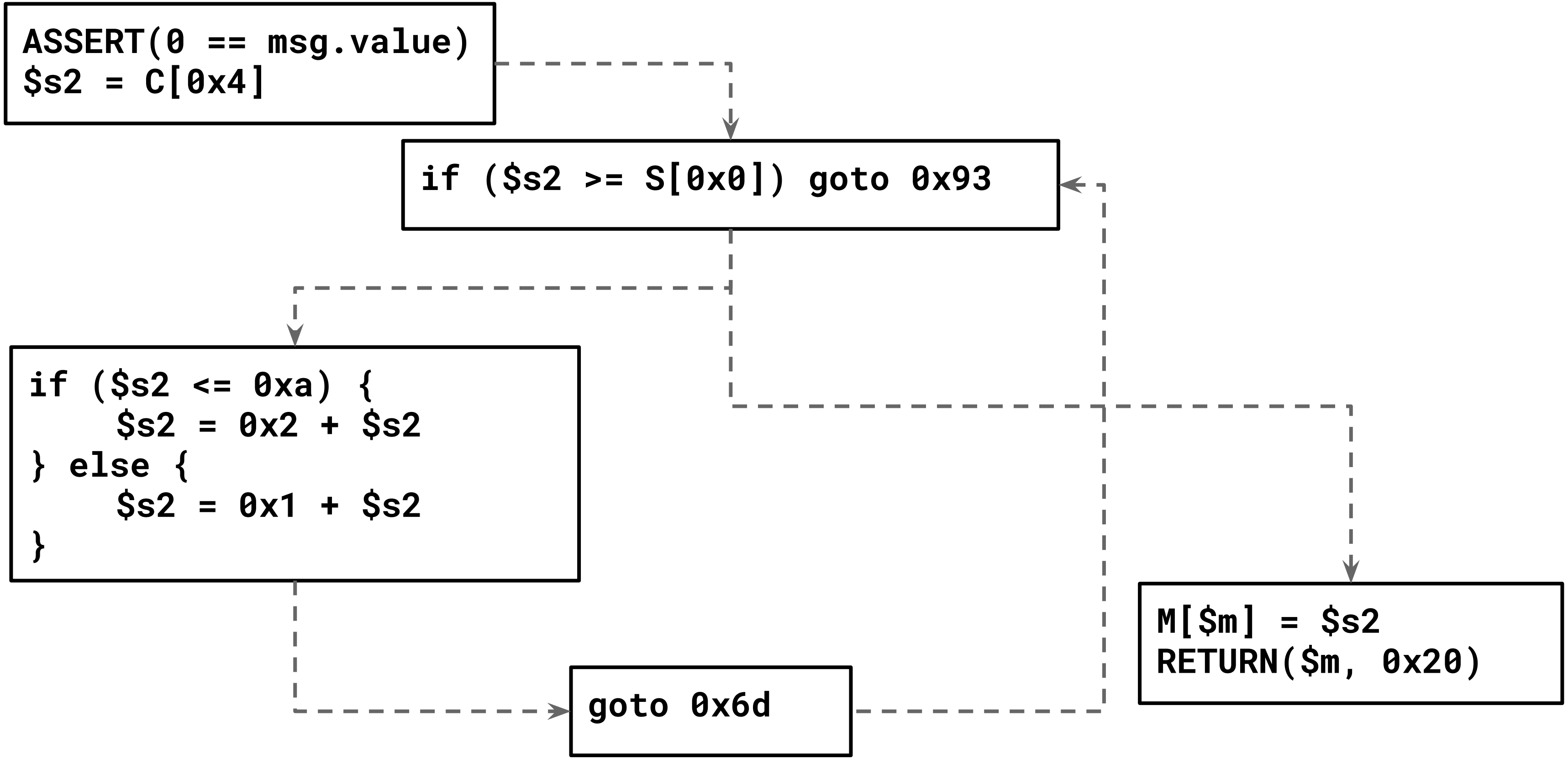
1 → 2 → 3 → 4 →

Control Flow Structure Recovery

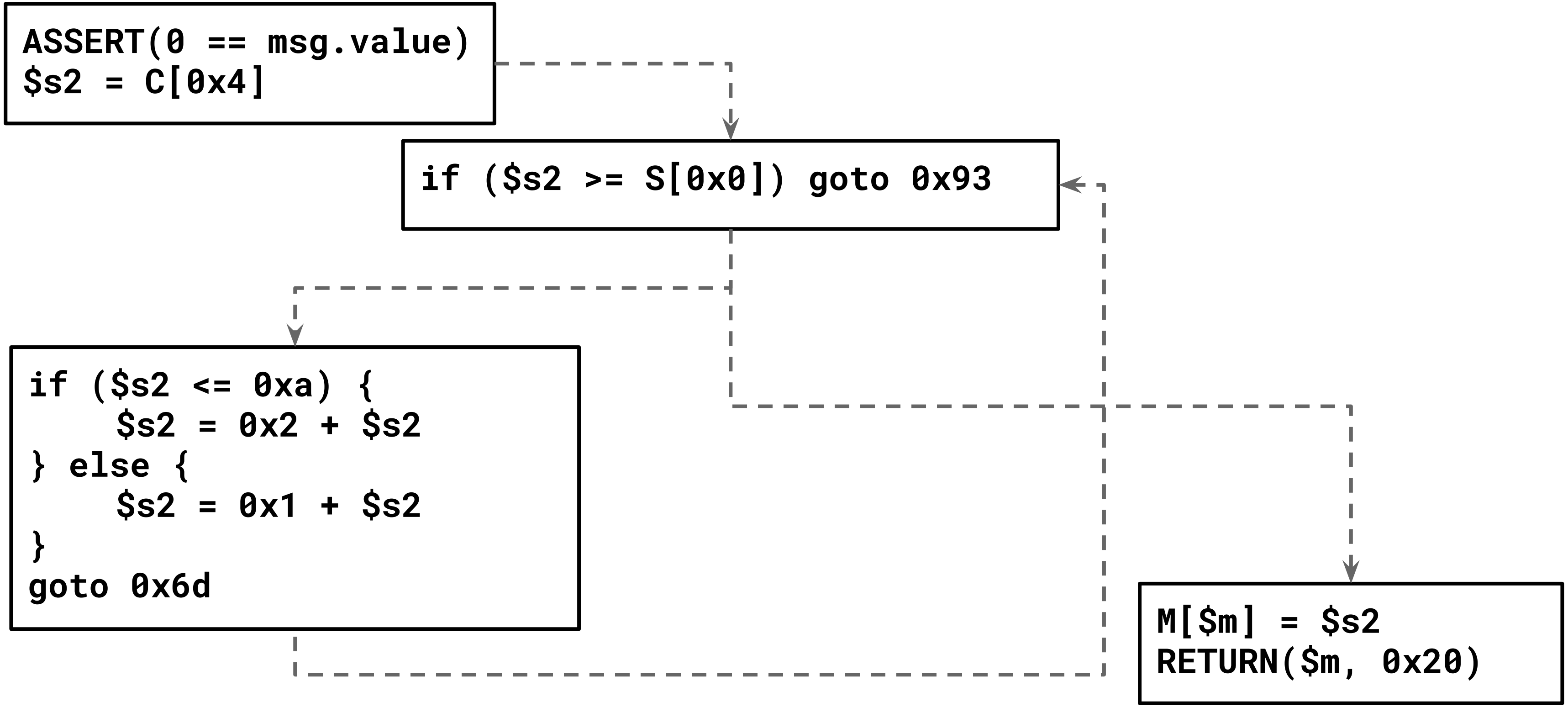
- Separate each public function subgraph
- Use structural analysis (M. Sharir 1980)
  - Match subgraphs to control constructs (while, if then else)
  - Collapse matched subgraphs



Control Flow Structure Recovery



Control Flow Structure Recovery



Control Flow Structure Recovery

```
ASSERT(0 == msg.value)
$s2 = C[0x4]
```

```
while (0x1) {
    if ($s2 >= S[0x0])
        break
    if ($s2 <= 0xa) {
        $s2 = 0x2 + $s2
    } else {
        $s2 = 0x1 + $s2
    }
}
```

```
M[$m] = $s2
RETURN($m, 0x20)
```

Control Flow Structure Recovery

```
ASSERT(0 == msg.value)
$s2 = C[0x4]
while (0x1) {
    if ($s2 >= S[0x0])
        break
    if ($s2 <= 0xa) {
        $s2 = 0x2 + $s2
    } else {
        $s2 = 0x1 + $s2
    }
}
M[$m] = $s2
RETURN($m, 0x20)
```

Control Flow Structure Recovery

Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

Validation

- Construct test cases using historical transactions

- Leverage Geth to generate the expected transaction output

- “Execute” our representation and compare the output

	Transactions
Total	15,855 (100.0 %)

Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

	Transactions
Total	15,855 (100.0 %)
Success	15,345 (96.8%)

Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

	Transactions
Total	15,855 (100.0 %)
Success	15,345 (96.8%)
Failures	510 (3.2%)

Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

	Transactions
Total	15,855 (100.0 %)
Success	15,345 (96.8%)
Failures	510 (3.2%)
Construction Failures	196 (1.2%)

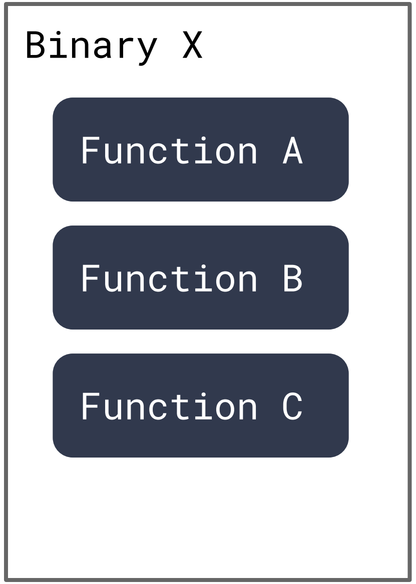
Validation

- Construct test cases using historical transactions
- Leverage Geth to generate the expected transaction output
- “Execute” our representation and compare the output

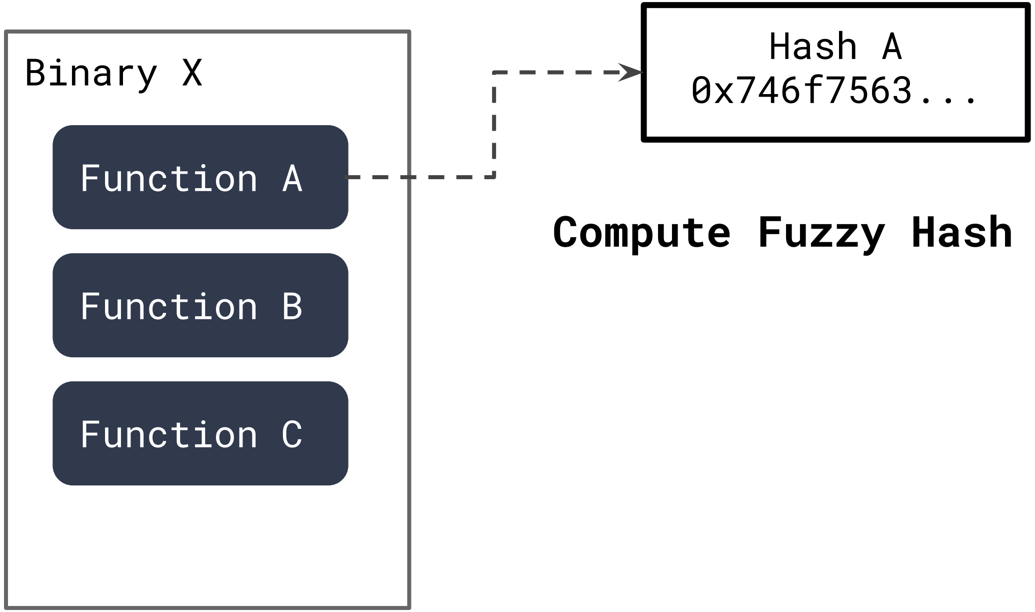
	Transactions
Total	15,855 (100.0 %)
Success	15,345 (96.8%)
Failures	510 (3.2%)
Construction Failures	196 (1.2%)
Comparison Failures	314 (2.0%)

Use Case

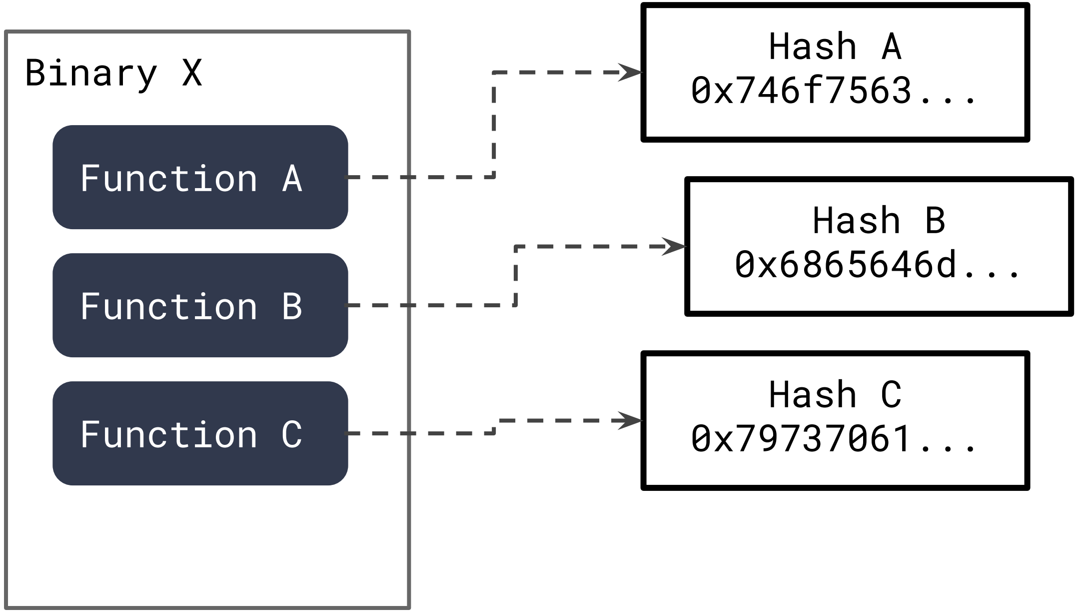
Erays: Function Fuzzy Hash



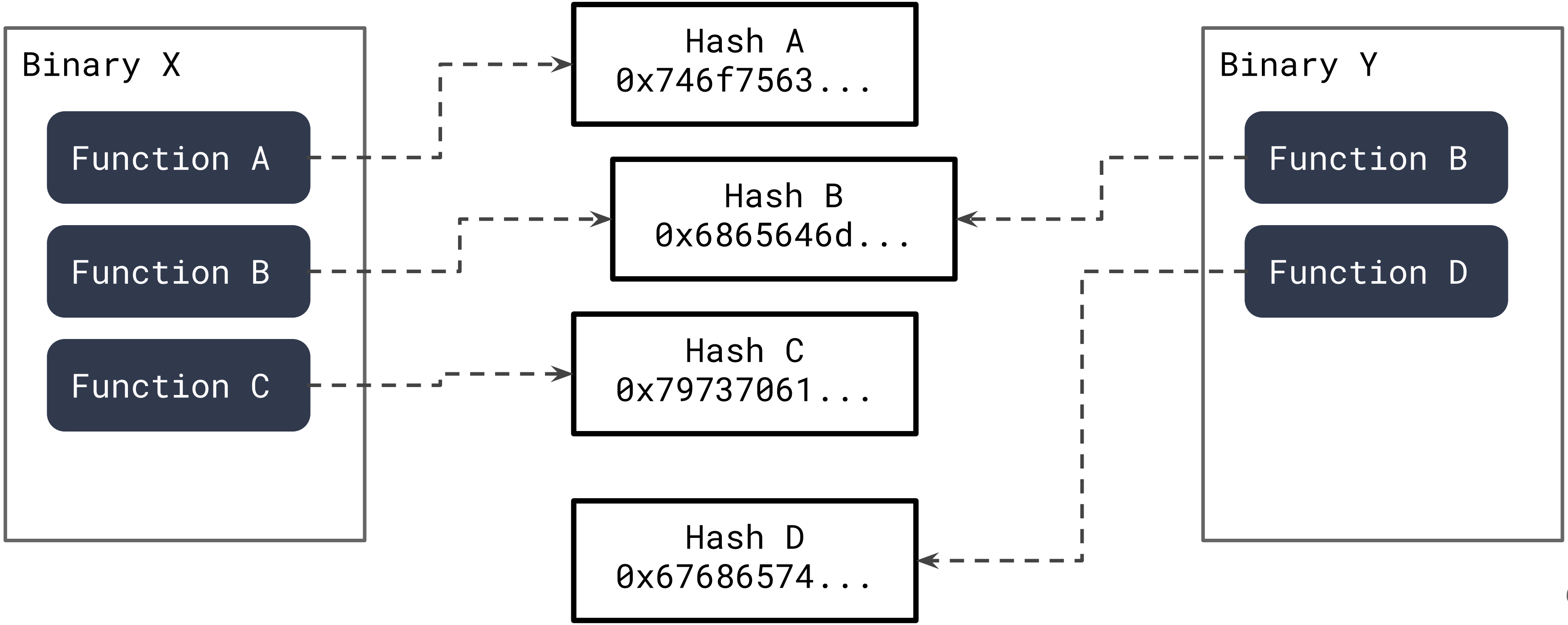
Erays: Function Fuzzy Hash



Erays: Function Fuzzy Hash



Erays: Code Sharing



Case Studies

Case Study: High Value Contracts

- Look for opaque contracts with large Ether balance ~ $590M
- **Multi-signature** wallets likely used by the **Gemini** exchange

Multi-Signature Wallet: signature scheme requiring k-of-N signatures.

- Security best practice for large sums of money

Case Study: High Value Contracts

- Look for opaque contracts with large Ether balance ~ $590M / 3 contracts
- **Multi-signature** wallets likely used by the **Gemini** exchange
- Interesting, time-dependent withdrawal policies

Multi-Signature Wallet: signature scheme requiring k-of-N signatures.

- Security best practice for large sums of money

Time Dependency Hazard

- Found **block.timestamp** used in contract
- Erays reveals it is used to control the delay of withdrawal requests
- Useful auditing tool, even for opaque contracts

```
$s10 = sha3(0x0, 0x40)
$s8 = $s10
s[$s10] = (ad_mask & $s3) | (0xffffff
s[0x1 + $s10] = $s4
s[0x2 + $s10] = $s7
$s9 = block.timestamp
s[0x3 + $s10] = $s9
if (msg.sender == ad_mask & s[0x0]){
  $s9 = s[0x1] + $s9
```

Case Study: Duplicate Contracts

- Look for opaque contracts with the **most instances**

- Exchange **user** wallets
  - **Poloniex**: ~350,000 contracts
  - **Yunbi**: ~90,000 contracts

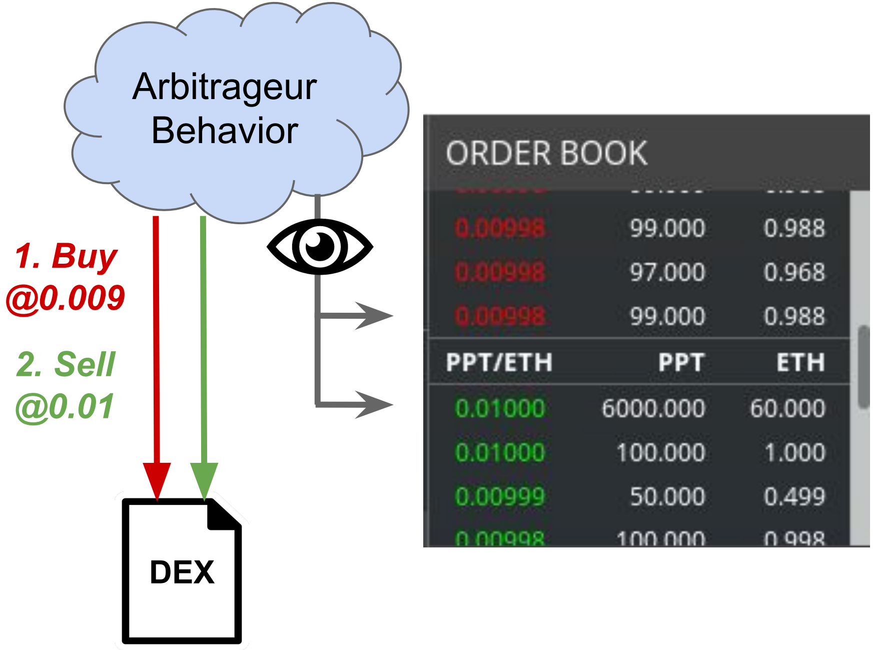
- A different approach to handling user funds

Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - Etherdelta

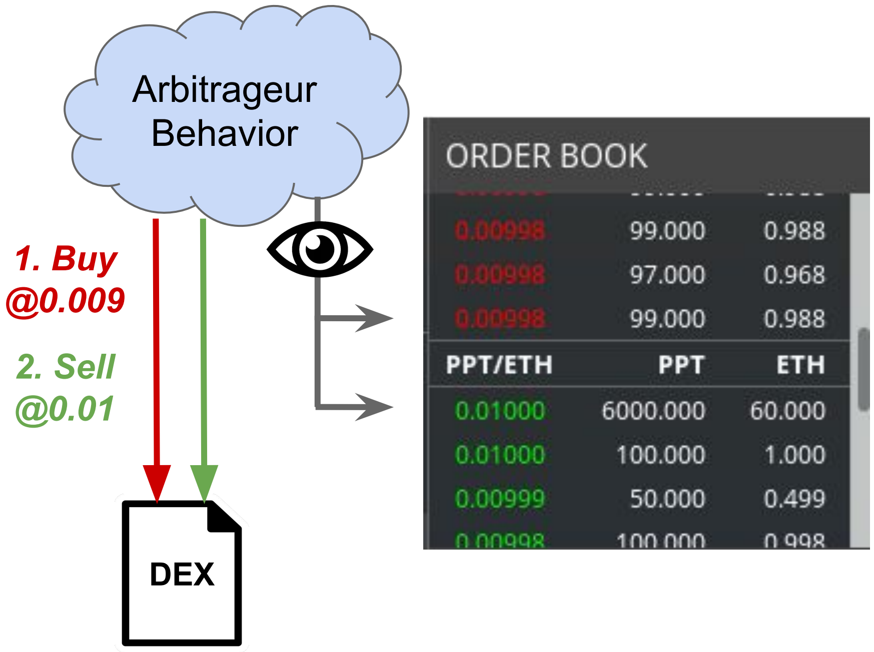
Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - Etherdelta
- Evidence of arbitrageurs



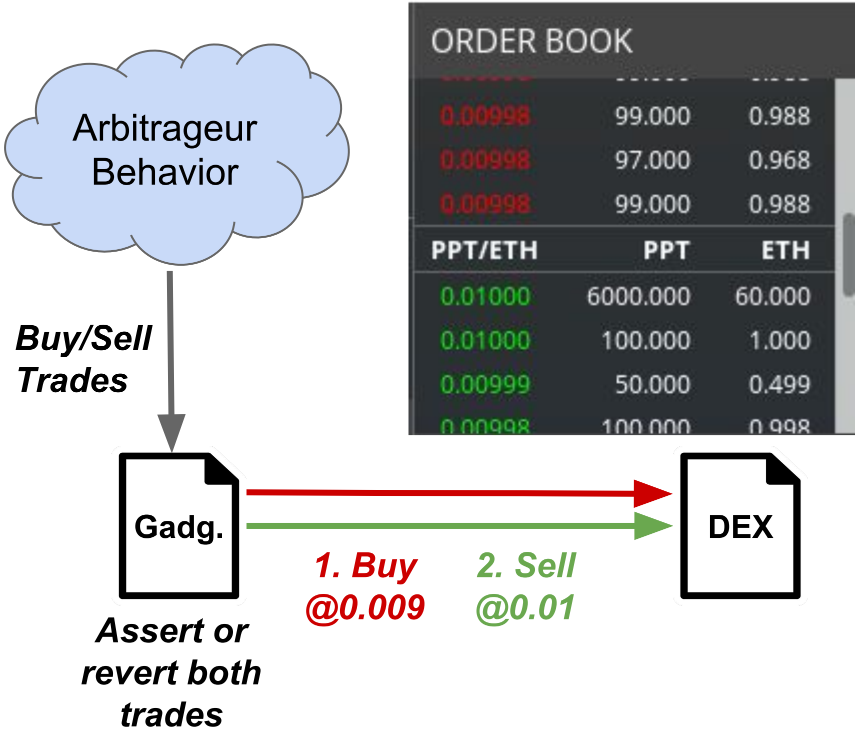
Case Study: EtherDelta Arbitrage

- Decentralized token exchanges (DEX) operate entirely on-chain
  - Etherdelta
- Evidence of arbitrageurs
- Executing a buy/sell mismatch for a profit

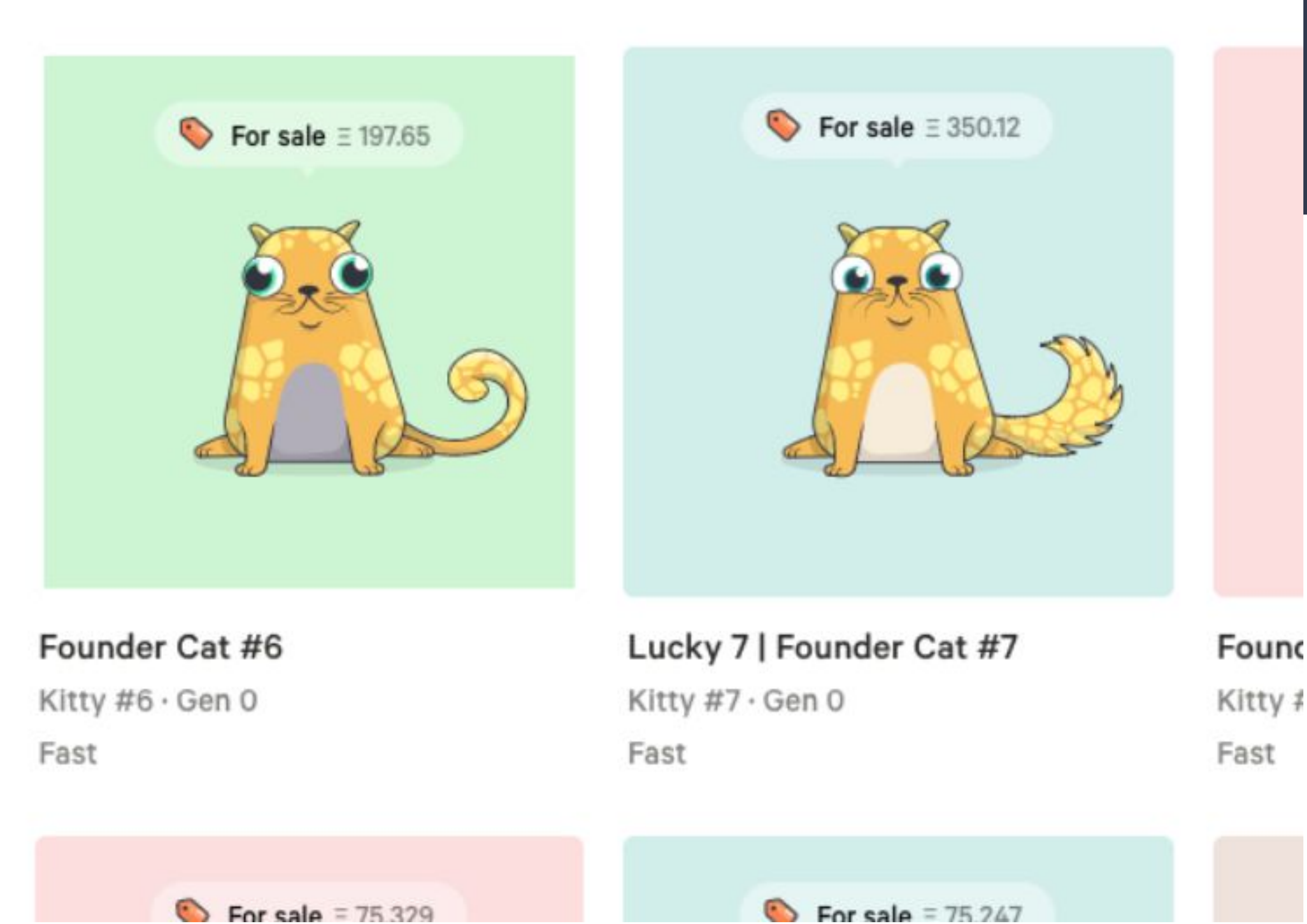


Case Study: EtherDelta Arbitrage Bots

- Arbitrageurs must publish *gadgets* to facilitate arbitrage
- Create functions to validate the order and new trade
- Implement atomic batch trades (or fail)



Case Study: CryptoKitties


For sale Ξ 197.65
For sale Ξ 350.12
Founder Cat #6
Kitty #6 · Gen 0
Fast
Lucky 7 | Founder Cat #7
Kitty #7 · Gen 0
Fast
Fast
For sale Ξ 75.329
For sale Ξ 75.247

Case Study: CryptoKitties

- On-chain game code is published with source code

- Game mechanism well understood

```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```

Case Study: CryptoKitties

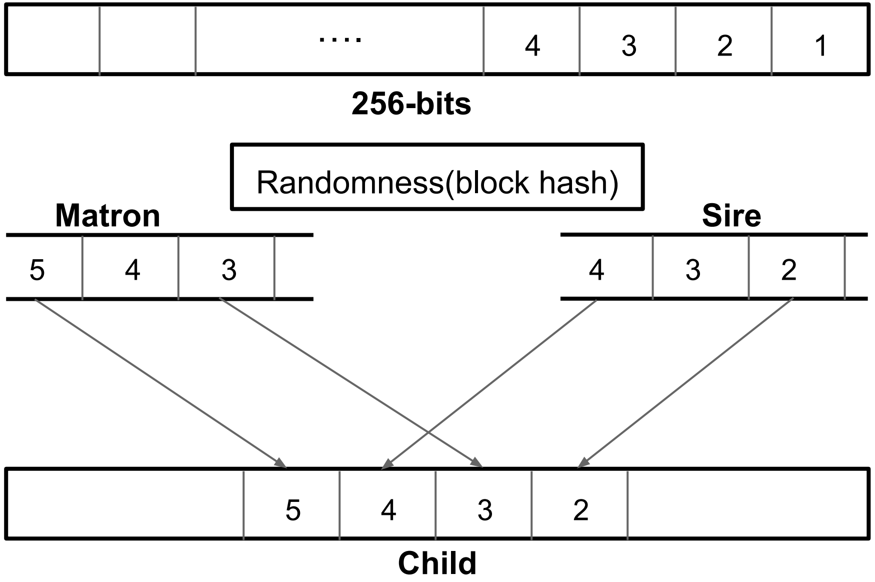
- Developers who know the algorithm aren't allowed to play the game!

Case Study: CryptoKitties

- Developers who know the algorithm aren't allowed to play the game!
- So obviously we had to target this function

Case Study: CryptoKitties

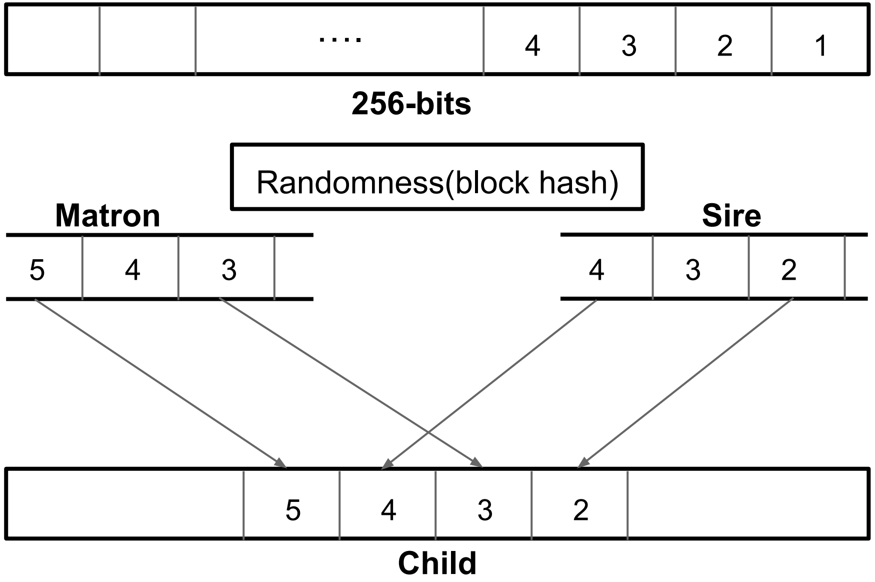
```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```



- The block hash is used to inject random mutations into genes and to select a parent for a gene

Case Study: CryptoKitties

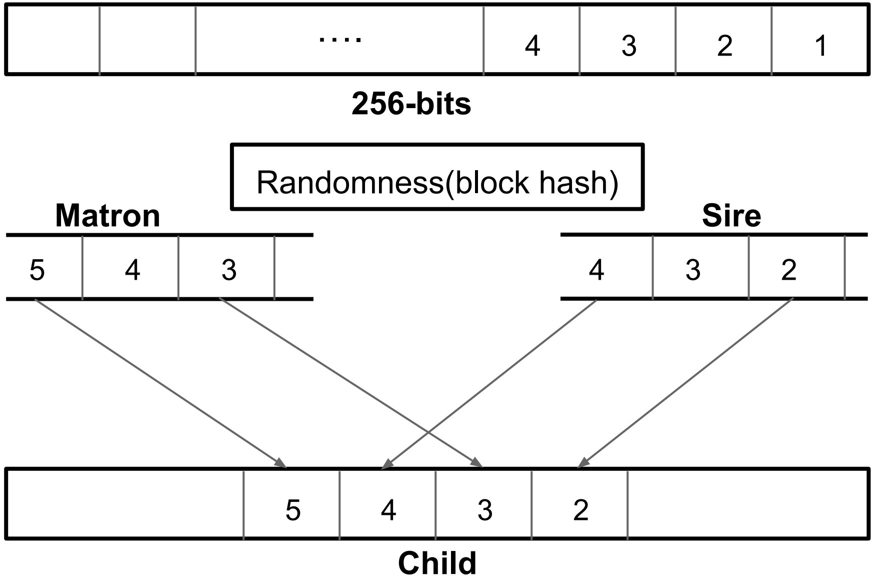
```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```



- The block hash is used to inject random mutations into genes and to select a parent for a gene
- Found a more effective breeding strategy

Case Study: CryptoKitties

```
// Call the sooper-sekret gene mixing operation.
uint256 childGenes = geneScience.mixGenes(matron.genes, sire.genes, matron.cooldownEndBlock - 1);
```



- The block hash is used to inject random mutations into genes and to select a parent for a gene
- Found a more effective breeding strategy
- Don't rely on security through obscurity!

Conclusion

- Ethereum smart contract ecosystem is largely opaque
  - ~ 1M contracts, 34K unique, 77.5% unique opaque

Conclusion

- Ethereum smart contract ecosystem is largely opaque
    - ~ 1M contracts, 34K unique, 77.5% unique opaque
- Erays converts EVM bytecode into higher level representations
    - https://github.com/teamnsrg/erays
    - yizhou7@illinois.edu

Conclusion

- Ethereum smart contract ecosystem is largely opaque
  - ~ 1M contracts, 34K unique, 77.5% unique opaque

- Erays converts EVM bytecode into higher level representations
  - https://github.com/teamnsrg/erays
  - yizhou7@illinois.edu

- The utility of Erays is demonstrated in several case studies
  - High value wallets, exchange user wallets, arbitrage bots, CryptoKitties secret algorithm