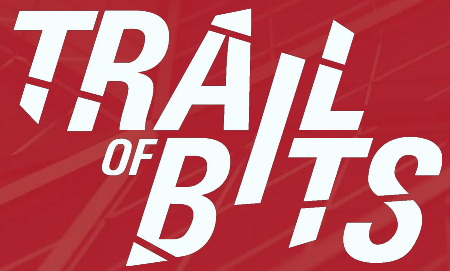


Detecting transaction displacement attacks with Manticore

Sam Moelius

Please **do** interrupt me with questions.

Transaction displacement attack[1]

An attacker replaces a legitimate transaction with their own, to steal something of value intended the legitimate transaction's sender.

[1] Shayan Eskandari, Seyedehmahsa Moosavi, Jeremy Clark. "SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain." In Proceedings of Financial Cryptography and Data Security, 2019.

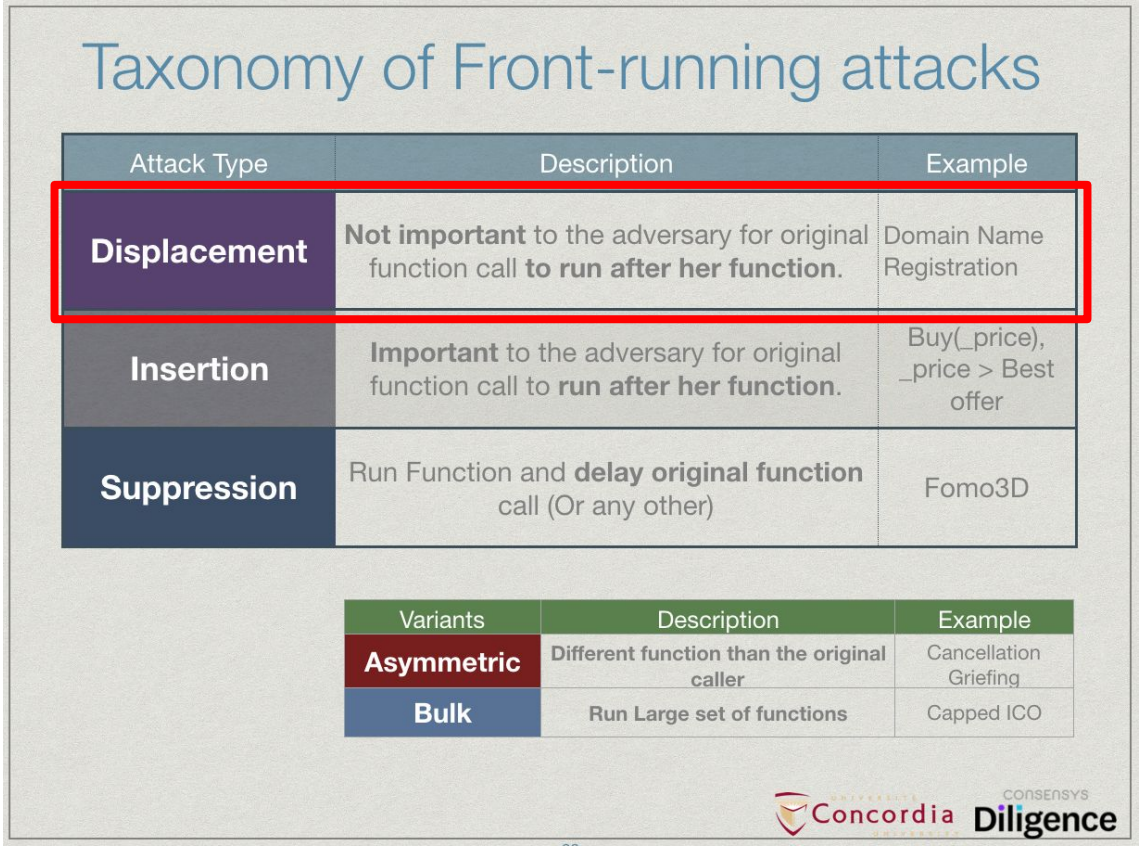
Eskandari, et al. 2019, “Transparent Dishonesty”

Taxonomy of Front-running attacks

Attack Type	Description	Example
Displacement	**Not important** to the adversary for original function call **to run after her function.**	Domain Name Registration
Insertion	**Important** to the adversary for original function call to **run after her function.**	Buy(_price), _price > Best offer
Suppression	Run Function and **delay original function** call (Or any other)	Fomo3D

Variants	Description	Example
Asymmetric	**Different function than the original caller**	Cancellation Griefing
Bulk	**Run Large set of functions**	Capped ICO

Eskandari, et al. 2019, “Transparent Dishonesty”



“Examples of displacement include:

- Alice trying to register a domain name and Mallory registering it first…;
- Alice trying to submit a bug to receive a bounty and Mallory stealing it and submitting it first…; and
- Alice trying to submit a bid in an auction and Mallory copying it.”[2]

[2] Eskandari, et al. 2019, page 173

“Examples of displacement include:

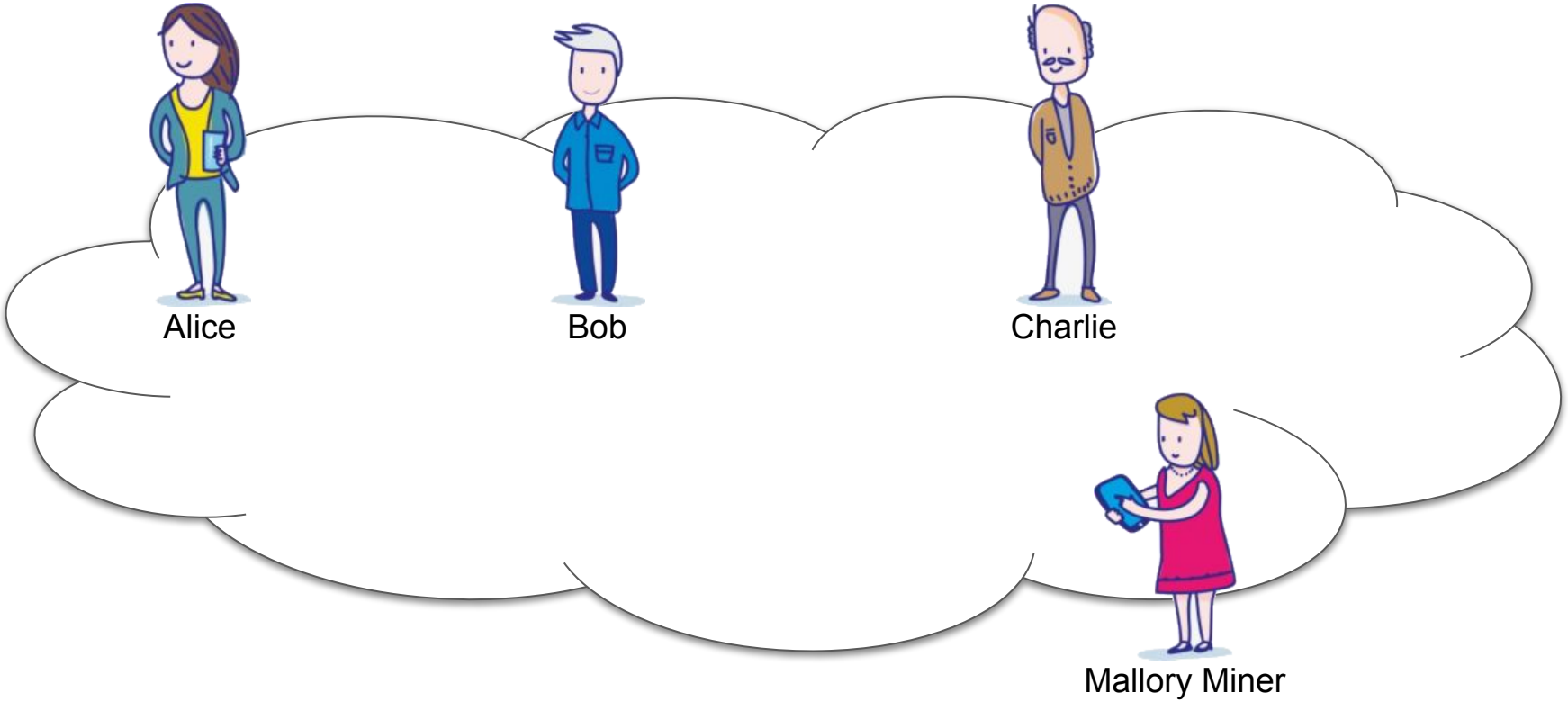
- Alice trying to register a domain name and Mallory registering it first…;
- Alice trying to submit a bug to receive a bounty and Mallory stealing it and submitting it first…; and
- Alice trying to submit a bid in an auction and Mallory copying it.”[2]

[2] Eskandari, et al. 2019, page 173

Detailed example

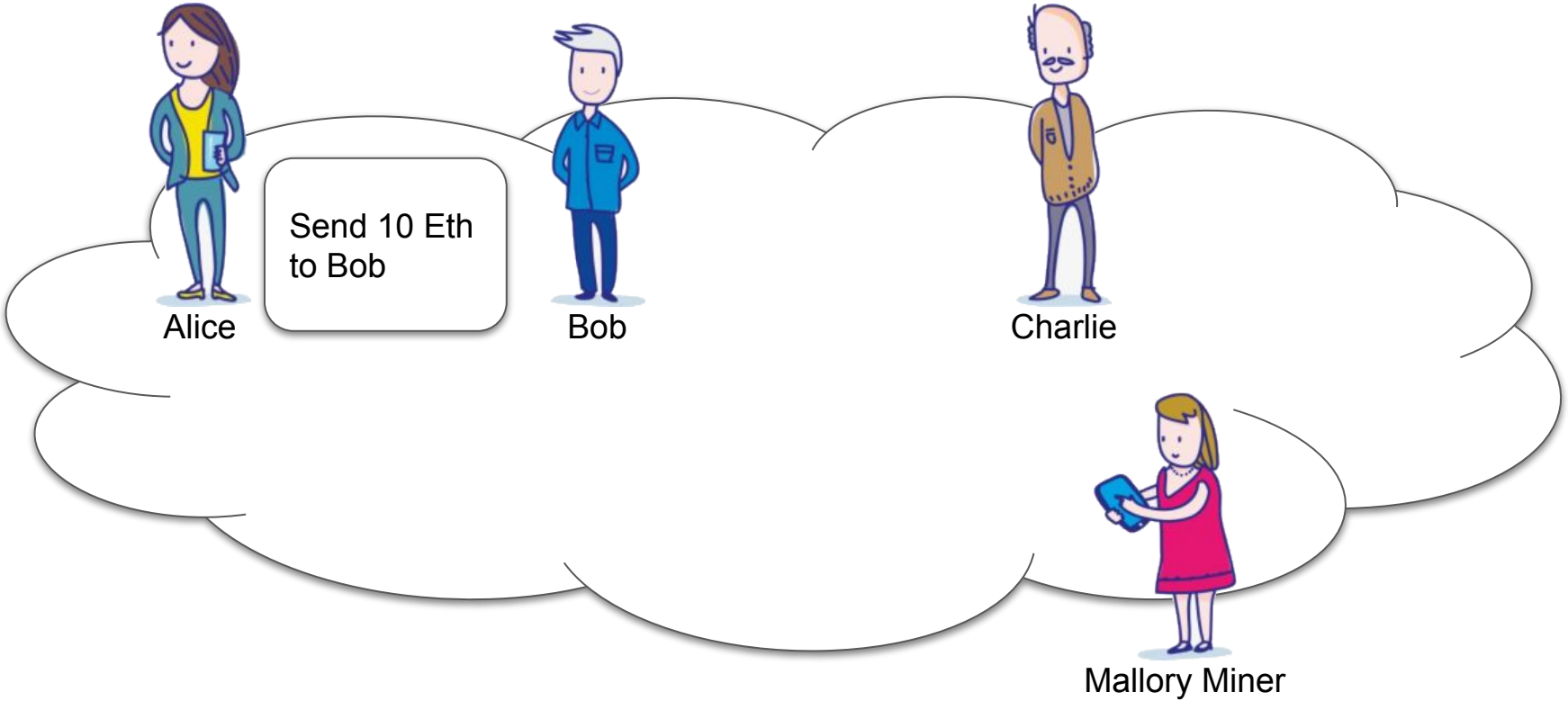
```
contract AddressBounty {
    bool redeemed;
    address target;

    constructor(address _target) public payable {
        target = _target;
    }

    function submit(uint256 _x, uint256 _y) public {
        require(!redeemed, "already redeemed");
        require(target == address(uint256(keccak256(abi.encodePacked(_x, _y)))), "invalid");
        redeemed = true;
        msg.sender.transfer(address(this).balance);
    }
}
```

Ethereum



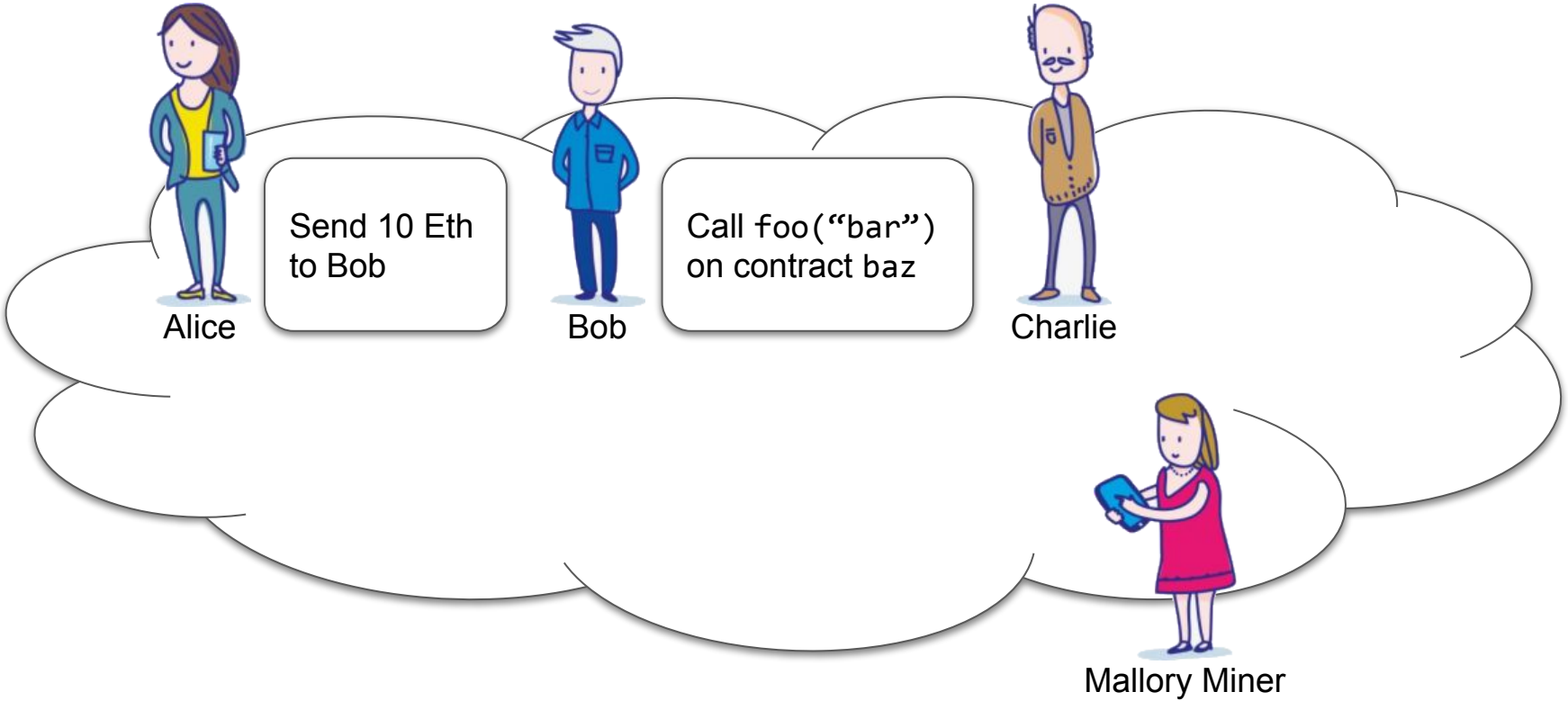
✔ transaction succeeds ✘ transaction reverts

Ethereum



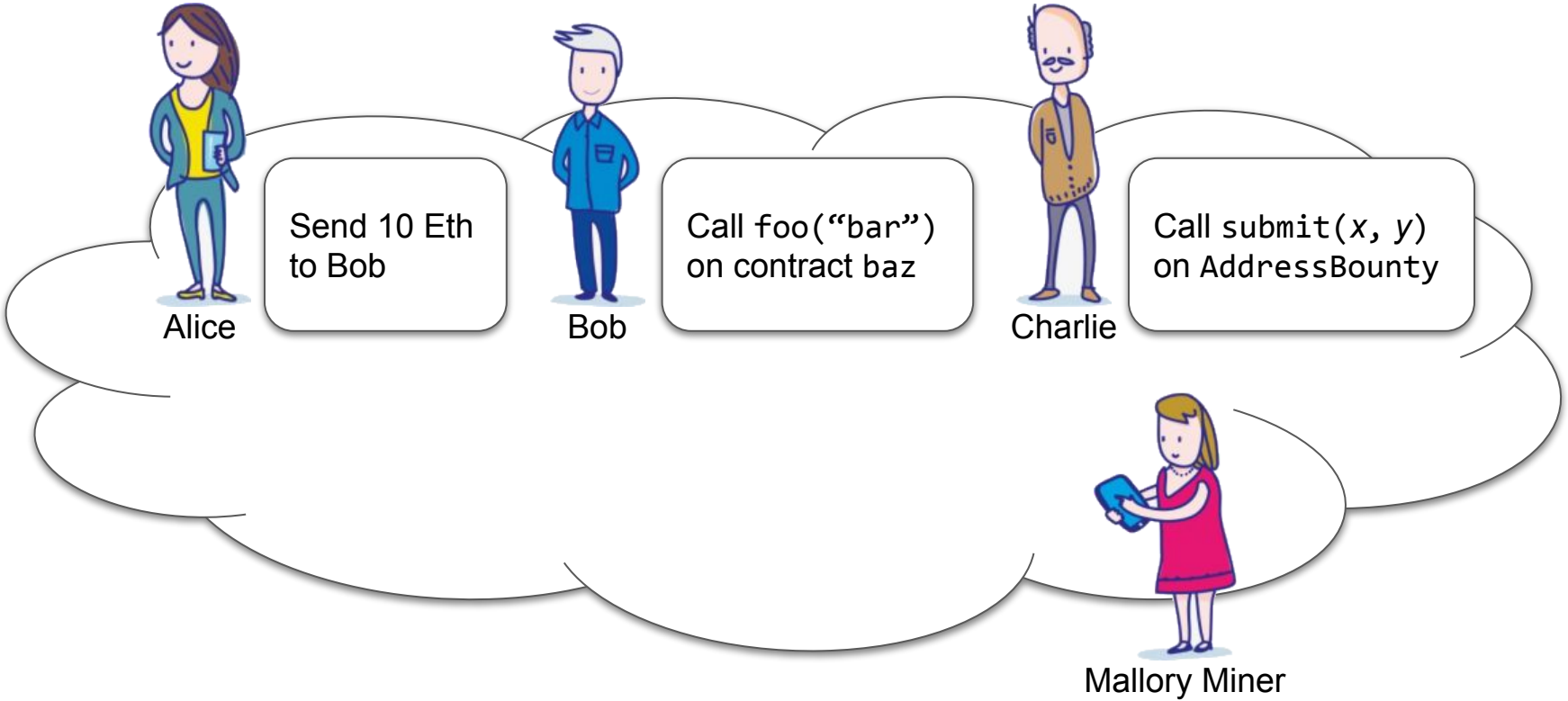
✔ transaction succeeds ✘ transaction reverts

Ethereum



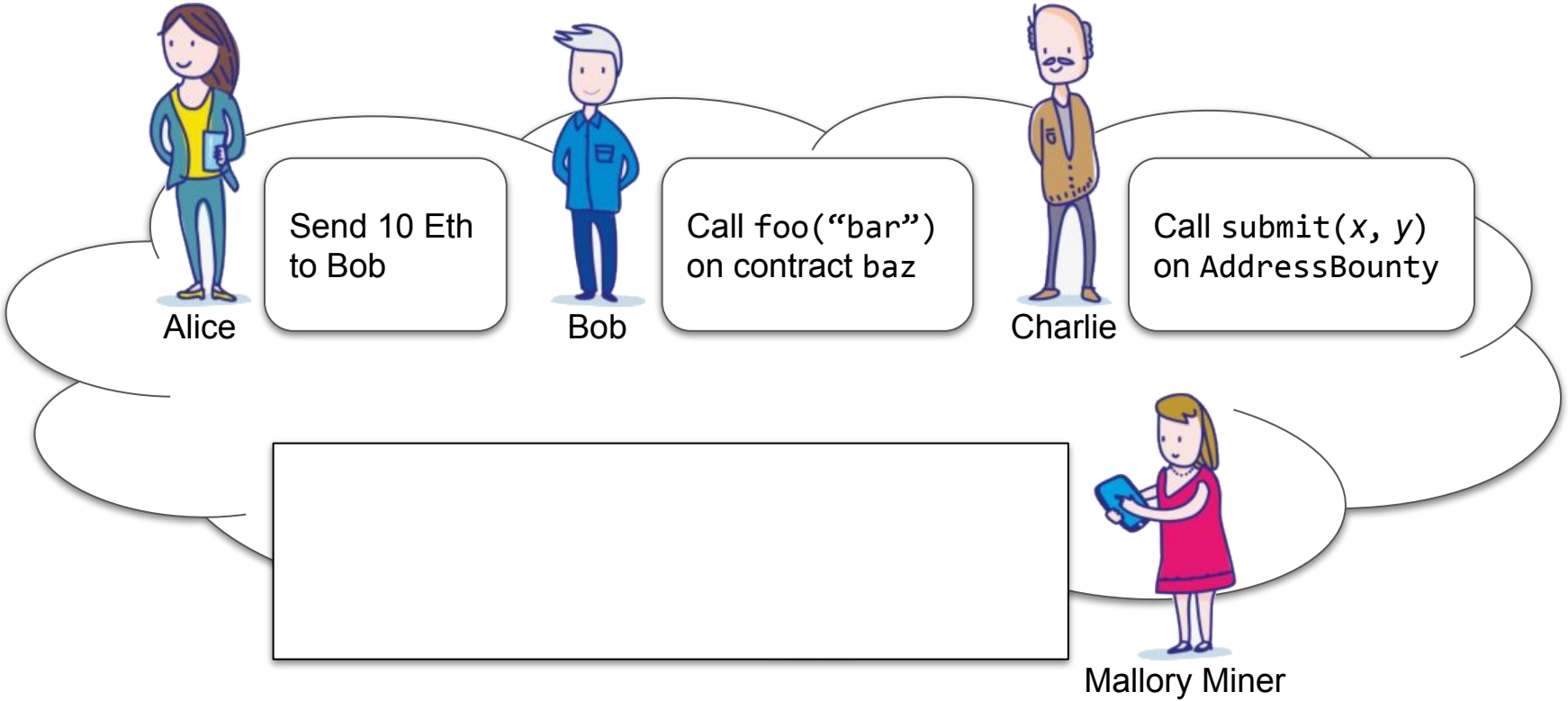
✔ transaction succeeds ✘ transaction reverts

Ethereum



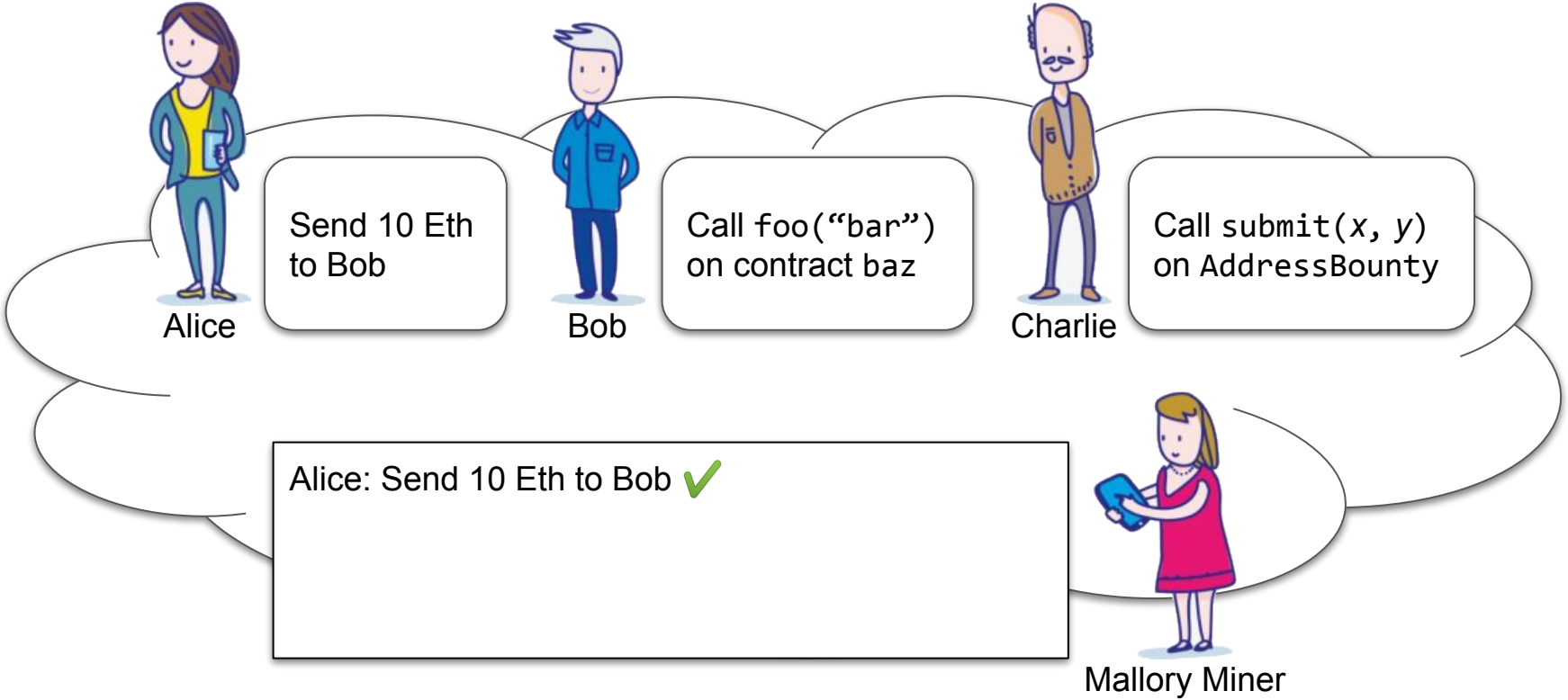
✔ transaction succeeds ✘ transaction reverts

Ethereum



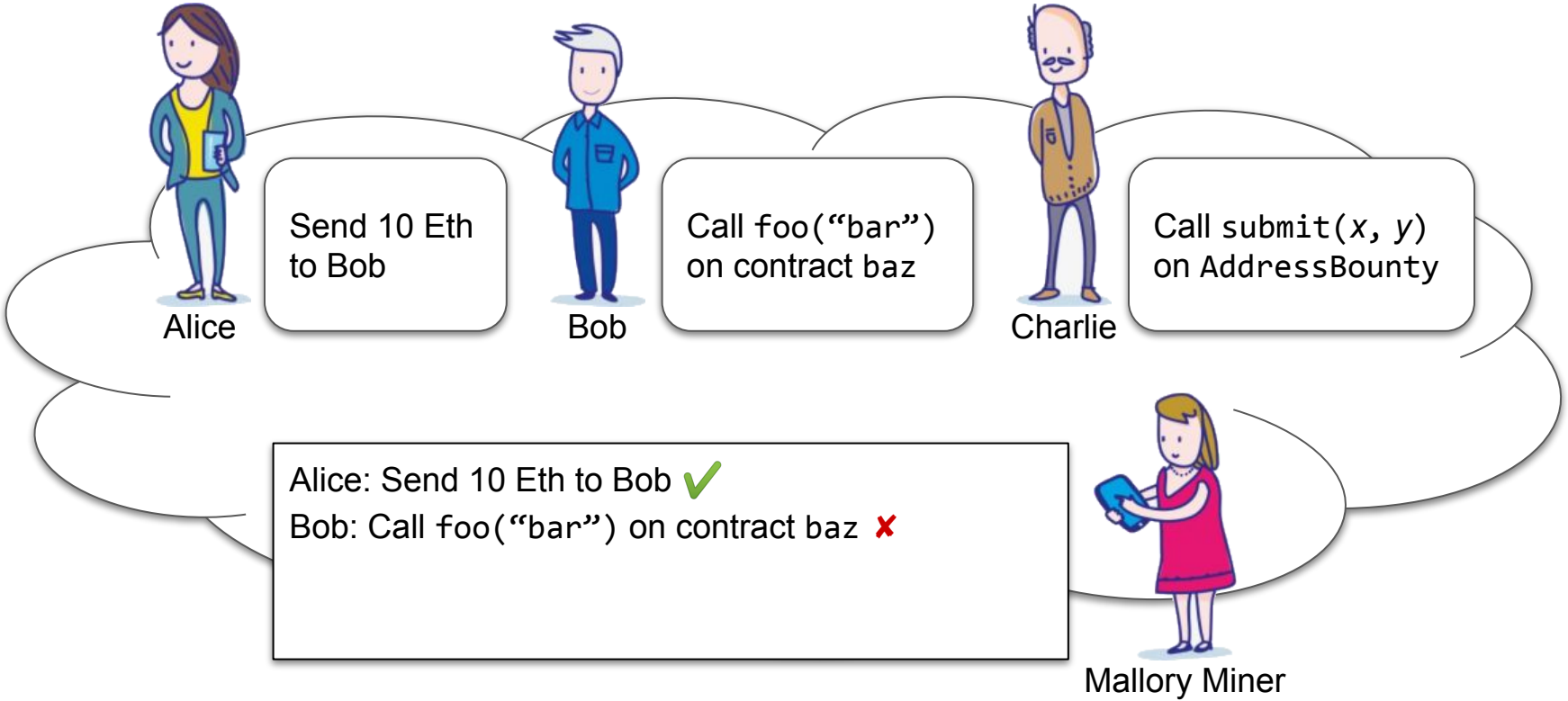
✔ transaction succeeds ✘ transaction reverts

Ethereum



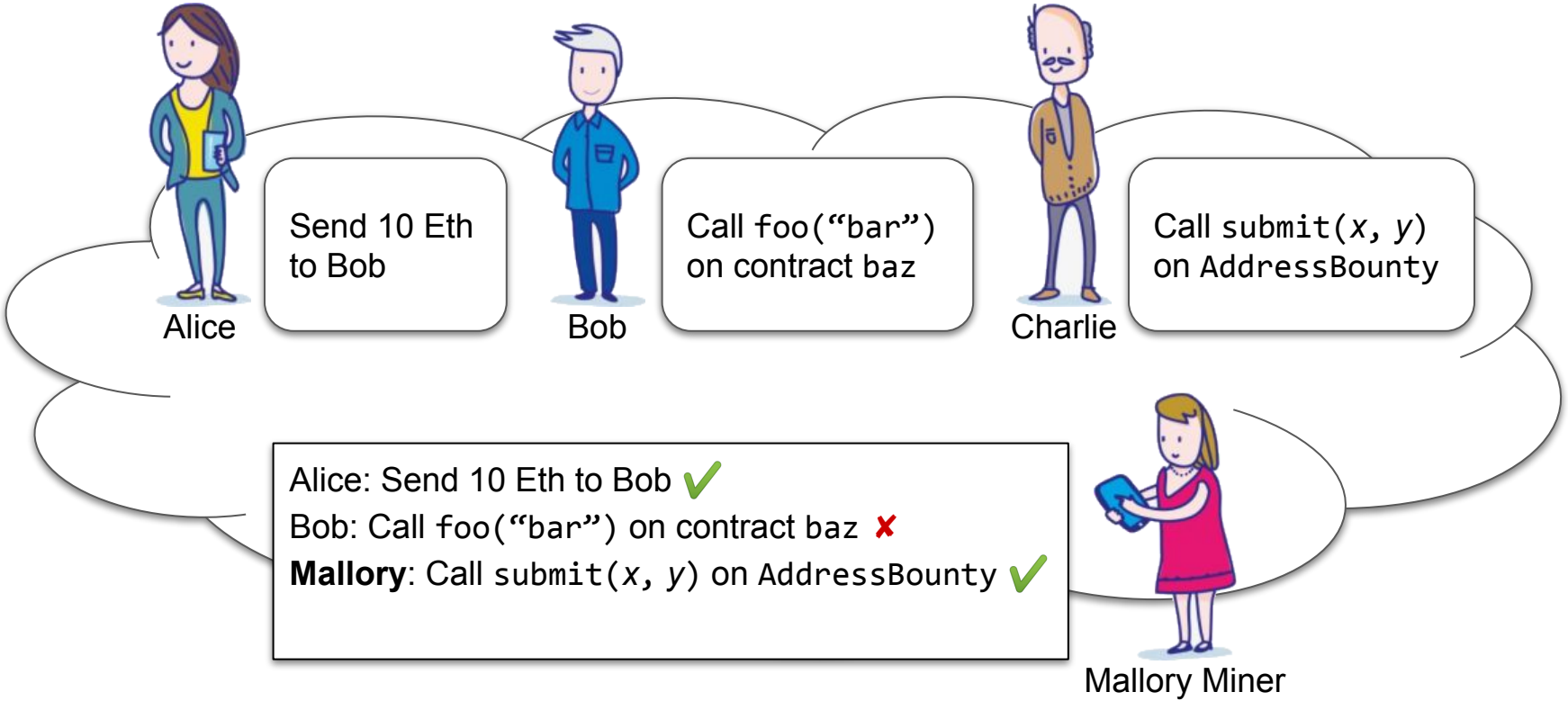
✔ transaction succeeds ✘ transaction reverts

Ethereum



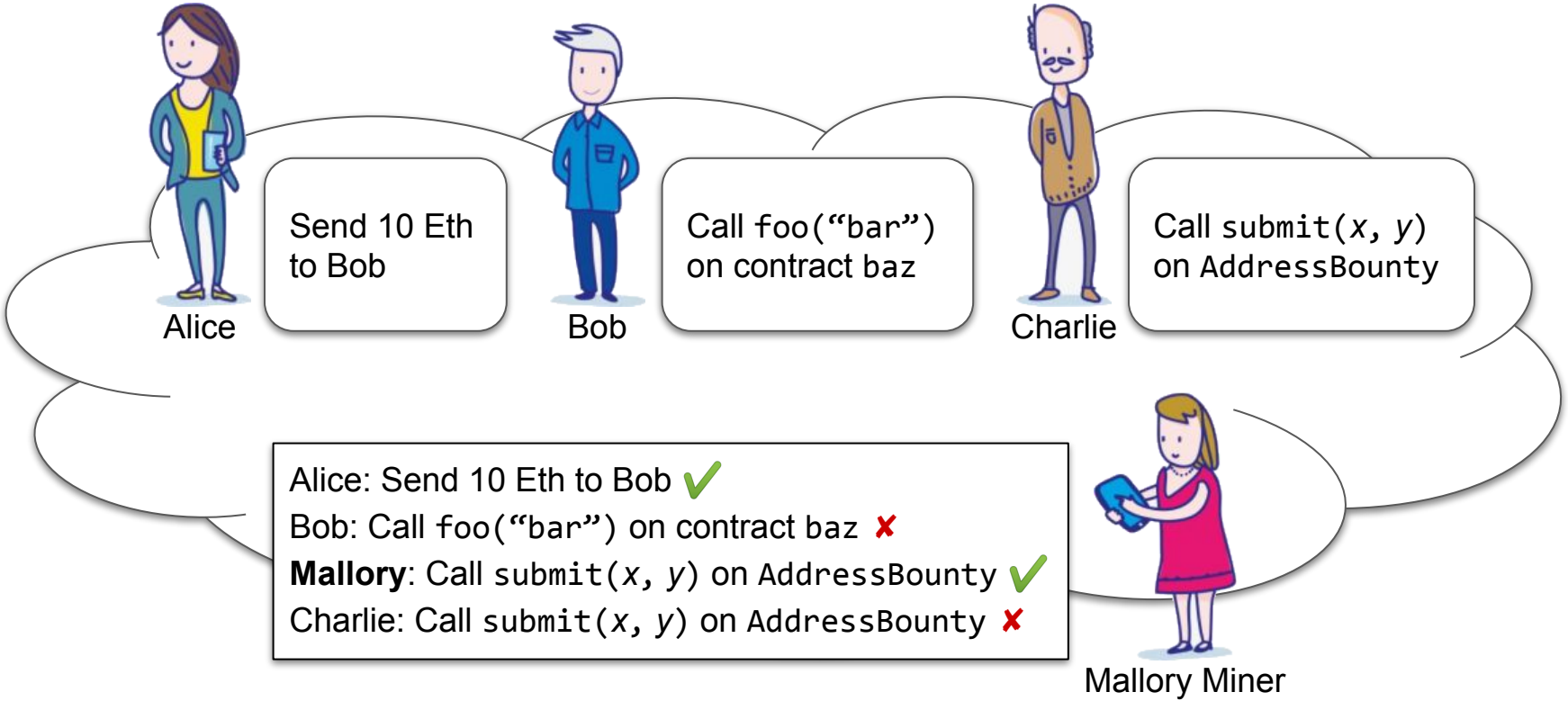
✔ transaction succeeds ✘ transaction reverts

Ethereum



✔ transaction succeeds ✘ transaction reverts

Ethereum



✔ transaction succeeds ✘ transaction reverts

How to fix?

- Instead of calling just `submit(`*x*, *y*`)`, the sender calls `commit(`*z*`)` followed by `reveal(`*x*, *y*`)`.

- The contract verifies:
  - the calls to `commit` and `reveal` are separated by at least *n* blocks
  - *z* == `hash(`*x*, *y, sender's address*`)`
  - `submit(`*x*, *y*`)` would have succeeded

How to detect?

In each of these examples, you would expect the legitimate transaction to revert following the fraudulent one:

- "Alice trying to register a domain name and Mallory registering it first…;
- Alice trying to submit a bug to receive a bounty and Mallory stealing it and submitting it first…; and
- Alice trying to submit a bid in an auction and Mallory copying it."[3]

[3] Eskandari, et al. 2019, page 173 (repeated)

Example revisited

```
contract AddressBounty {
    bool redeemed;
    address target;

    constructor(address _target) public payable {
        target = _target;
    }

    function submit(uint256 _x, uint256 _y) public {
        require(!redeemed, "already redeemed");
        require(target == address(uint256(keccak256(abi.encodePacked(_x, _y)))), "invalid");
        redeemed = true;
        msg.sender.transfer(address(this).balance);
    }
}
```

Example revisited

```
contract AddressBounty {
    bool redeemed;
    address target;

    constructor(address _target) public payable {
        target = _target;
    }

    function submit(uint256 _x, uint256 _y) public {
        require(!redeemed, "already redeemed");
        require(target == address(uint256(keccak256(abi.encodePacked(_x, _y)))), "invalid");
        redeemed = true;
        msg.sender.transfer(address(this).balance);
    }
}
```

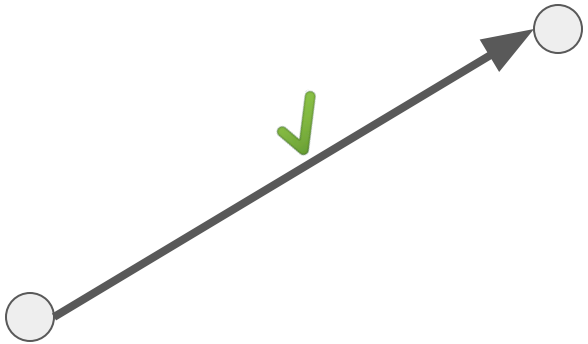
Manticore[4]

- Trail of Bits' symbolic execution tool

- Manticore can analyze the following types of programs:
  - Ethereum smart contracts (EVM bytecode)
  - Linux ELF binaries (x86, x86_64, aarch64, and ARMv7)
  - WASM modules

[4] https://github.com/trailofbits/manticore

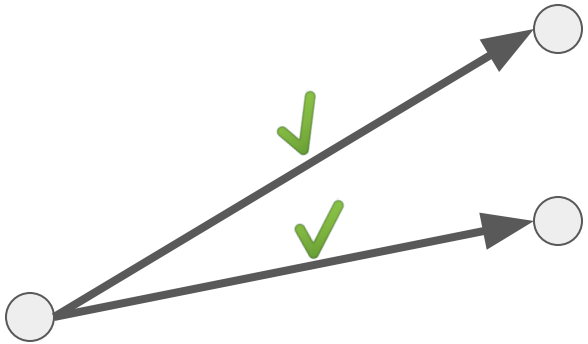
Manticore



- Each circle in these diagrams represents a state of the blockchain.
- Each arrow represents a transaction.

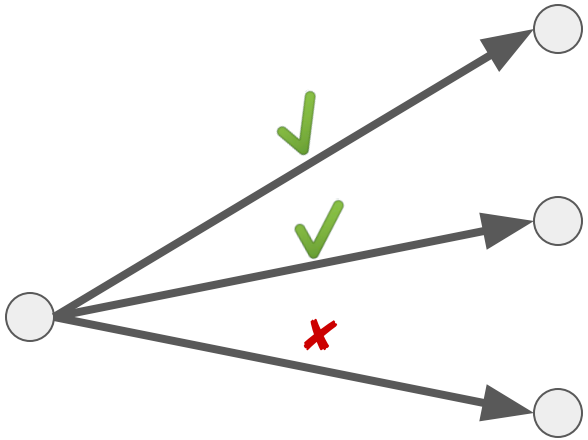
✔ transaction succeeds ✘ transaction reverts

Manticore



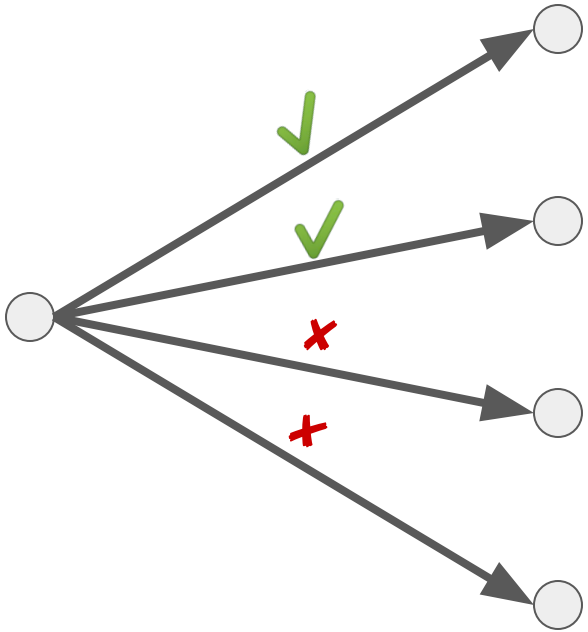
✔ transaction succeeds ✘ transaction reverts

Manticore



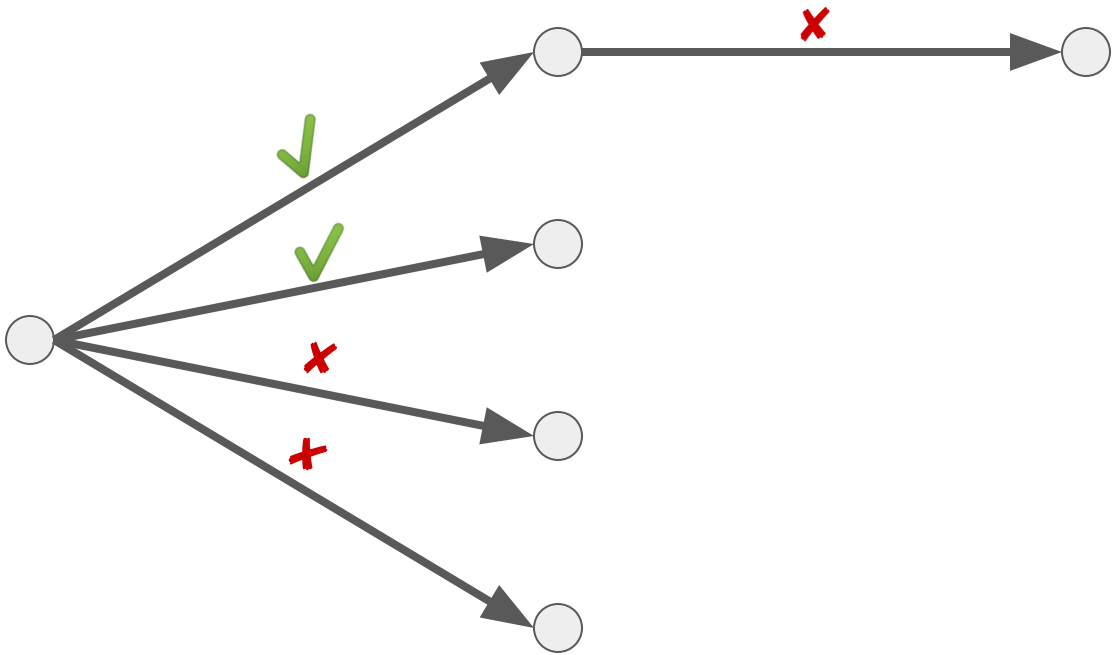
✔ transaction succeeds ✘ transaction reverts

Manticore



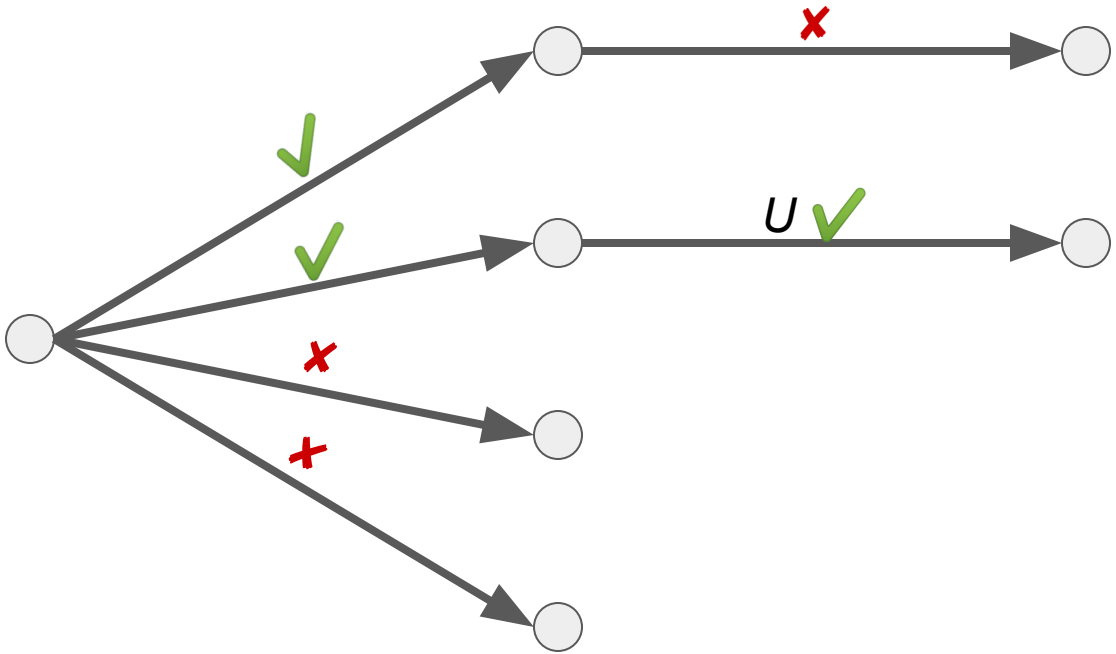
✔ transaction succeeds ✘ transaction reverts

Manticore



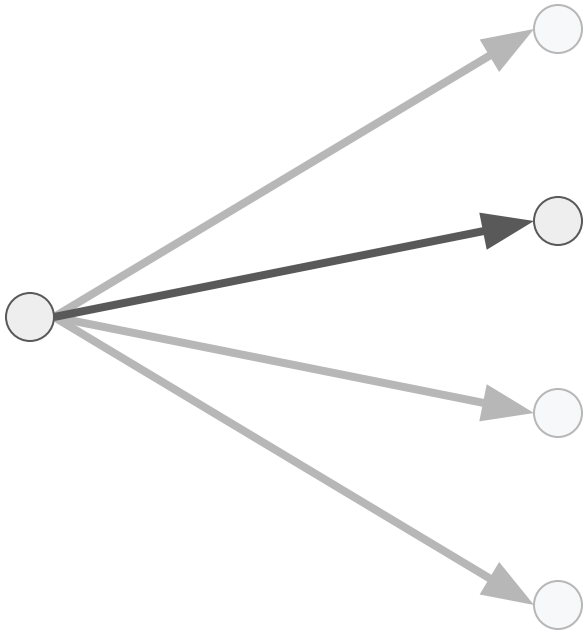
✔ transaction succeeds ✘ transaction reverts

Manticore



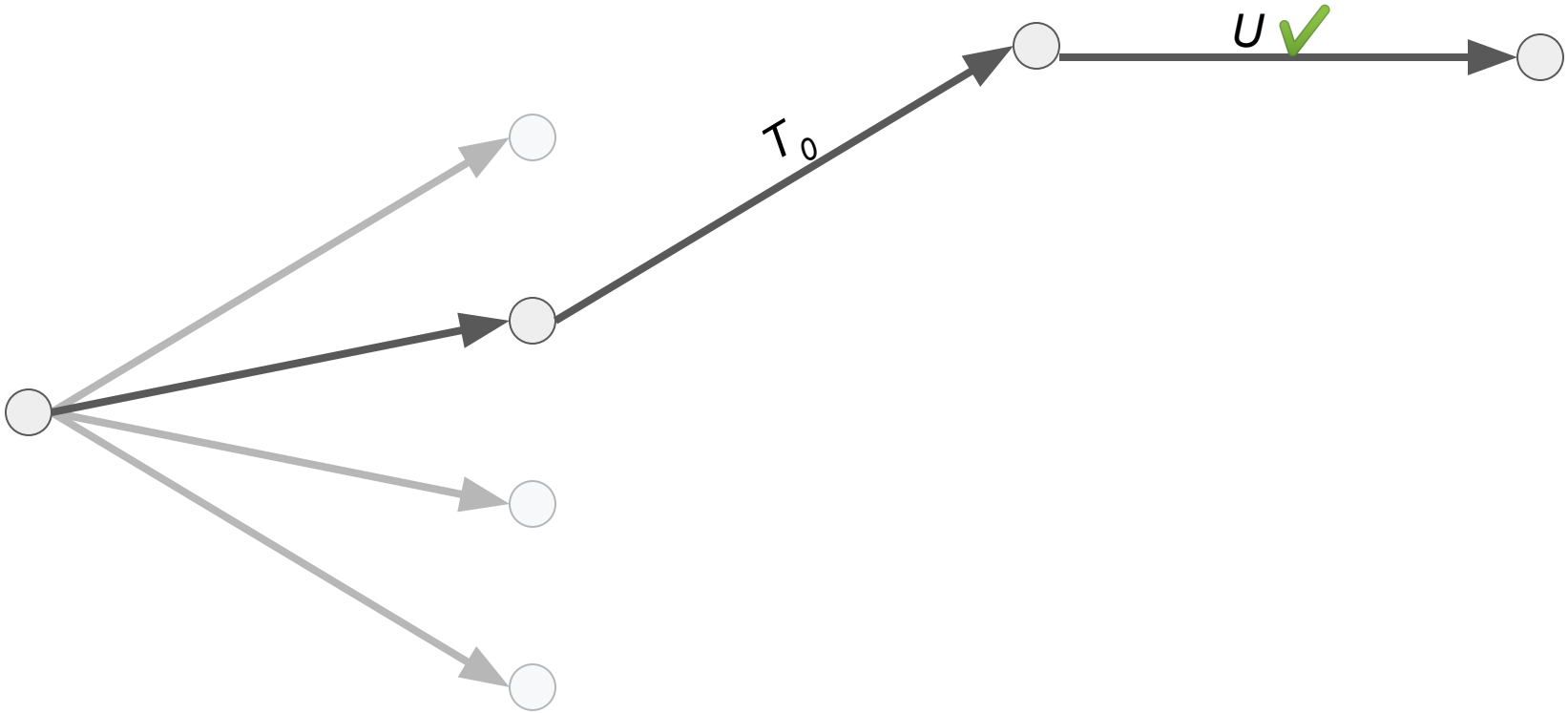
✓ transaction succeeds ✘ transaction reverts

Manticore



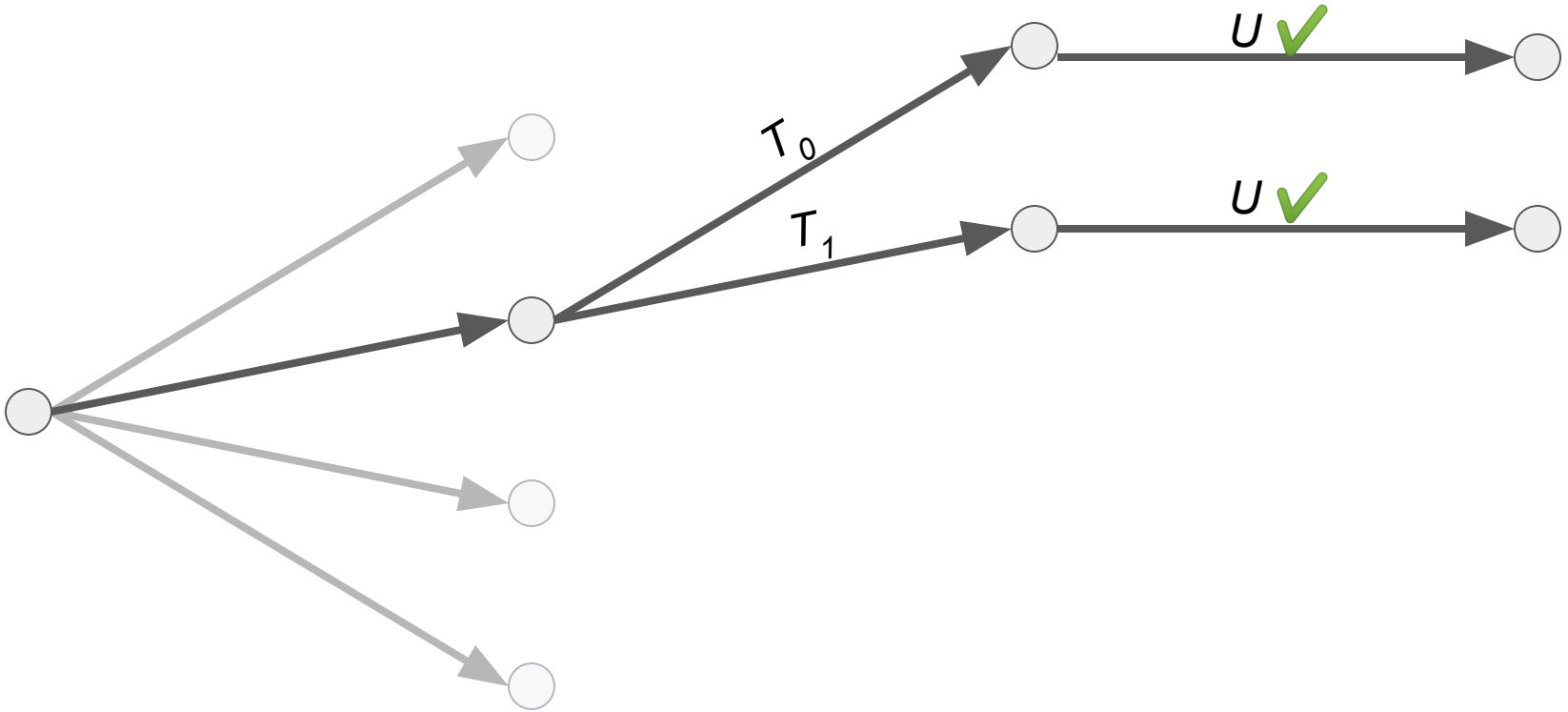
✔ transaction succeeds ✘ transaction reverts

Manticore



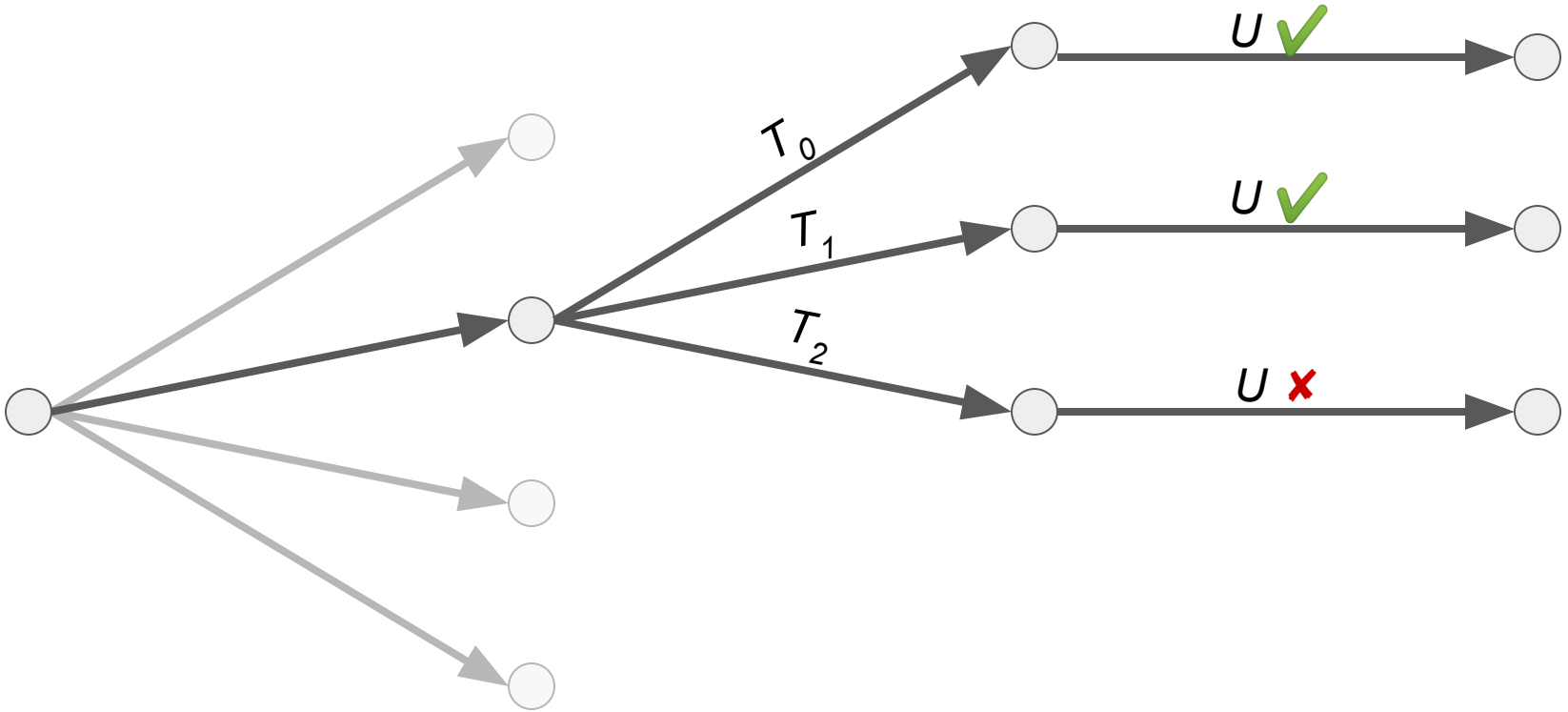
✓ transaction succeeds ✘ transaction reverts

Manticore



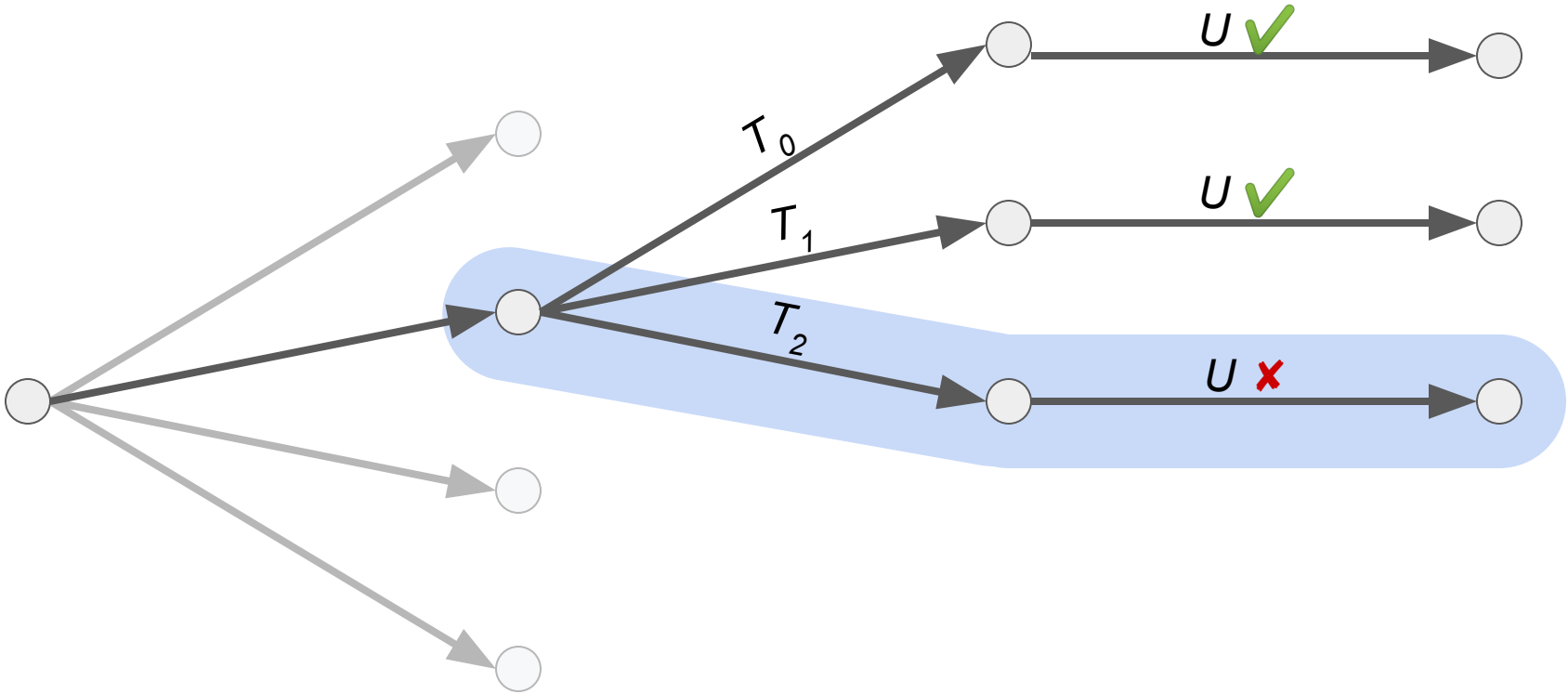
✓ transaction succeeds ✘ transaction reverts

Manticore



✓ transaction succeeds ✘ transaction reverts

Manticore



✓ transaction succeeds ✘ transaction reverts

Example re-revisited

```
#! /usr/bin/env python3

from manticore.ethereum import ManticoreEVM, DetectTransactionDisplacement

# Example due to Vincent Kobel:
#   https://kobl.one/blog/create-full-ethereum-keypair-and-address/
target = 0x0BED7ABD61247635C1973EB38474A2516ED1D884
x = 0x836B35A026743E823A90A0EE3B91BF615C6A757E2B60B9E1DC1826FD0DD16106
y = 0xF7BC1E8179F665015F43C6C81F39062FC2086ED849625C06E04697698B21855E

mevm = ManticoreEVM()
mevm.register_detector(DetectTransactionDisplacement())
owner = mevm.create_account()
contract = mevm.solidity_create_contract("addressbounty.sol", owner=owner, args=(target,))
contract.submit(x, y)
mevm.finalize()
```

Example re-revisited

```
#! /usr/bin/env python3

from manticore.ethereum import ManticoreEVM, DetectTransactionDisplacement

# Example due to Vincent Kobel:
#   https://kobl.one/blog/create-full-ethereum-keypair-and-address/
target = 0x0BED7ABD61247635C1973EB38474A2516ED1D884
x = 0x836B35A026743E823A90A0EE3B91BF615C6A757E2B60B9E1DC1826FD0DD16106
y = 0xF7BC1E8179F665015F43C6C81F39062FC2086ED849625C06E04697698B21855E

mevm = ManticoreEVM()
mevm.register_detector(DetectTransactionDisplacement())
owner = mevm.create_account()
contract = mevm.solidity_create_contract("addressbounty.sol", owner=owner, args=(target,))
contract.submit(x, y)
mevm.finalize()
```

This is where the magic happens.

user_00000005.tx (abbreviated)

```
...
Transactions No. 2
...
Data: 0x68284601836b35a026743e823a90a0ee3b91bf615c6a757e2b60b9e1dc1826fd0dd16106f7bc1e8179...06 (*)
Return_data: 0x ()
Function call: submit(594424059105367178073394452667619576459793512869338951059943371690025943943 74,
112053651391312806093045014627548927131116995152518320482663926909859426043230) -> STOP (*)

Transactions No. 3
...
Data: 0x68284601836b35a026743e823a90a0ee3b91bf615c6a757e2b60b9e1dc1826fd0dd16106f7bc1e8179...06 (*)
Return_data:
0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000
0000000000000000000000000000000000010616c726561647920726564656d656400000000000000000000000000000000
00 (y already redeemed) (*)
Function call: submit(594424059105367178073394452667619576459793512869338951059943371690025943943 74,
112053651391312806093045014627548927131116995152518320482663926909859426043230) -> REVERT
```

user_00000005.tx (abbreviated)

```
...
Transactions No. 2
...
Data: 0x68284601836b35a026743e823a90a0ee3b91bf615c6a757e2b60b9e1dc1826fd0dd16106f7bc1e8179...06 (*)
Return_data: 0x ()
Function call: submit(594424059105367178073394452667619576459793512869338951059943371690025943943 74,
112053651391312806093045014627548927131116995152518320482663926909859426043230) -> STOP (*)

Transactions No. 3
...
Data: 0x68284601836b35a026743e823a90a0ee3b91bf615c6a757e2b60b9e1dc1826fd0dd16106f7bc1e8179...06 (*)
Return_data:
0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000
0000000000000000000000000000000000010616c726561647920726564656d656400000000000000000000000000000000
00 (y already redeemed) (*)
Function call: submit(594424059105367178073394452667619576459793512869338951059943371690025943943 74,
112053651391312806093045014627548927131116995152518320482663926909859426043230) -> REVERT
```

Limitations (1 of 3): Expensive

- Adding a symbolic transaction to a run is expensive.

Limitations (2 of 3): False negatives

- Not every transaction displacement attack will cause a legitimate transaction to revert following a fraudulent one.
- Manticore's search is not guaranteed to be exhaustive.
- For contracts involving hard computational problems, an example, successful transaction is a must.
- If transaction *U* is a CREATE, we ignore it.

Limitations (3 of 3): False positives

- “Unsound symbolication,”[5] which turns the output of certain hash functions into free variables, causes false positives.

- Not every situation where a legitimate transaction would revert following a fraudulent one is worth fixing.

[5] “Unsound symbolication - A general approach to handle symbolic imprecisions” https://github.com/trailofbits/manticore/pull/1526

Real world “false positive”

```
/// @title Golem Network Token (GNT) - crowdfunding code for Golem Project
contract GolemNetworkToken {
    ...
    function finalize() external {
        // Abort if not in Funding Success state.
        if (!funding) throw;
        if ((block.number <= fundingEndBlock ||
             totalTokens < tokenCreationMin) &&
            totalTokens < tokenCreationCap) throw;

        // Switch to Operational state. This is the only place this can happen.
        funding = false;
        ...
    }
    ...
}
```

Real world “false positive”

```
/// @title Golem Network Token (GNT) - crowdfunding code for Golem Project
contract GolemNetworkToken {
    ...
    function finalize() external {
        // Abort if not in Funding Success state.
        if (!funding) throw;                          // if (funding) ...
        if ((block.number <= fundingEndBlock ||       // if ((block.number > fundingEndBlock &&
             totalTokens < tokenCreationMin) &&       //     totalTokens >= tokenCreationMin) ||
            totalTokens < tokenCreationCap) throw; //     totalTokens >= tokenCreationCap) ...

        // Switch to Operational state. This is the only place this can happen.
        funding = false;
        ...
    }
    ...
}
```

[WIP] Transaction displacement attack detector

https://github.com/trailofbits/manticore/pull/1698

sam.moelius@trailofbits.com

Manticore team

Eric Hennenfent Eric Kilmer Brad Larsen

Felipe Manzano Sonya Schriner

An example

```
contract Basic {
    uint256 n;

    constructor(uint256 _n) public {
        n = _n;
    }

    function submit(uint256 _x) external {
        require(n == _x, "wrong value");
        n++;
    }
}
```