SAILFISH

Vetting Smart Contract State-Inconsistency Bugs in Seconds

Priyanka Bose, Dipanjan Das, Yanju Chen, Yu Feng, Christopher Kruegel, Giovanni Vigna

UNIVERSITY OF CALIFORNIA SANTA BARBARA

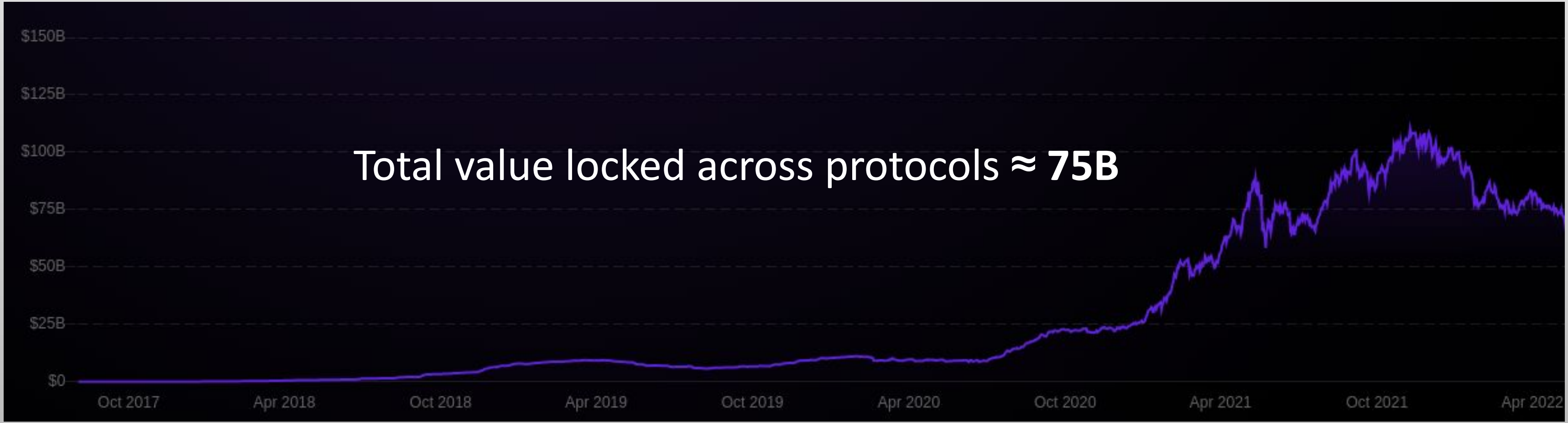
SECLAB

Smart contracts are popular

- Smart contracts are computer programs run on a Ethereum Virtual Machine (EVM)
- They process high value money transactions

Smart contracts are popular

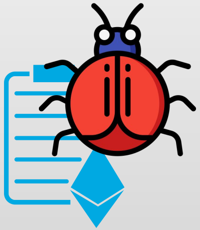
- Smart contracts are computer programs Run on a Ethereum Virtual Machine (EVM)
- They process high value money transactions



Smart contract attacks

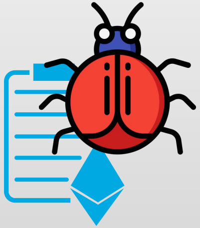
SECLAB

Vulnerabilities in smart contracts result in a loss of millions …



Smart contract attacks

Vulnerabilities in smart contracts result in a loss of millions …

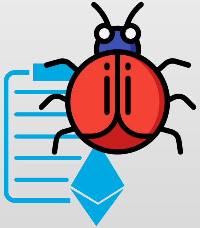


DAO	2016	$70M

Smart contract attacks

SECLAB

Vulnerabilities in smart contracts result in a loss of millions …



DAO	2016	$70M
BurgerSwap	2021	$7.2M
Cream Finance	2021	$17M
Fei Protocol	2022	$80M

Many more …

Existing techniques

Static analysis

Securify [Tsankov et. al., CCS 18]
Vandal [Brent et. al]
Slither [Feist et. al., WEBSTEB 19]
SmartCheck [Tikhomirov et. al., WEBSEB 18]

Symbolic execution

Oyente [Luu et. al., CCS 16]
Manticore [Trail of bits]
Mytril [Consensys]

Runtime analysis

Sereum [Rodler et. el., NDSS 19]
ECFChecker [Grossman et. al., POPL 18]

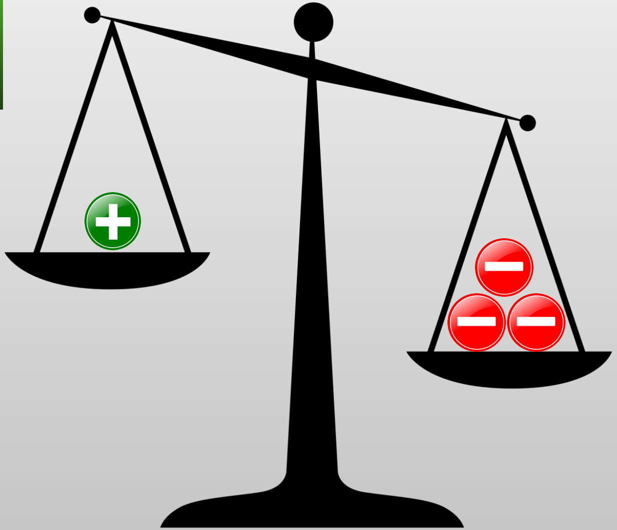
Verification

ZEUS [Kalra et. al, NDSS 18],
SeRIF [Cecchetti et. al, S&P 21]

Existing techniques

SECLAB

Variety of techniques to detect reentrancy and TOD



Symbolic techniques not scalable for large contracts

Static techniques incurs false positives

Relies on some bug signatures, misses others

Contributions

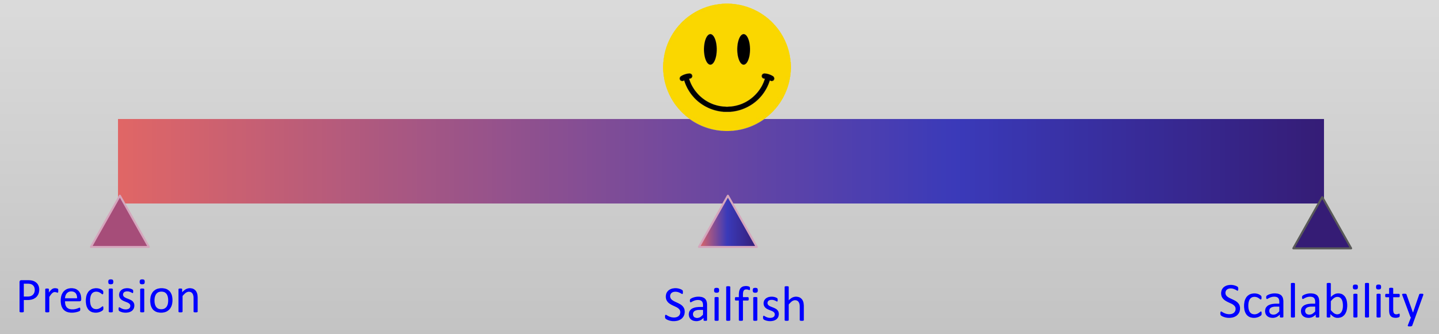
- A general technique to detect reentrancy and transaction order dependence (TOD)
- Scales well for large contracts and achieves precision

Contributions

- A general technique to detect reentrancy and transaction order dependence (TOD)
- Scales well for large contracts and achieves precision

Contributions

- A general technique to detect reentrancy and transaction order dependence (TOD)
- Scales well for large contracts and achieves precision



Sailfish

Introduces *State Inconsistency* (SI), a general definition of reentrancy and transaction order dependence (TOD)

Defines read-write dependencies of a storage variables as Hazardous access, a root cause of SI

SECLAB

Sailfish

Introduces *State Inconsistency* (SI), a general definition of reentrancy and transaction order dependence (TOD)

Defines read-write dependencies of a storage variables as Hazardous access, a root cause of SI

✓ Detects reentrancy and TOD including the ones missed by prior tools

Sailfish

Introduces *State Inconsistency* (SI), a general definition of reentrancy and transaction order dependence (TOD)

Defines read-write dependencies of a storage variables as Hazardous access, a root cause of SI

Detects reentrancy and TOD including the ones missed by prior tools

Combines static analysis and symbolic execution

Summarizes the contract storage variable using scalable value-summary analysis

Sailfish

Introduces *State Inconsistency* (SI), a general definition of reentrancy and transaction order dependence (TOD)

Defines read-write dependencies of a storage variables as Hazardous access, a root cause of SI

Detects reentrancy and TOD including the ones missed by prior tools

Combines static analysis and symbolic execution

Summarizes the contract storage variables using scalable value-summary analysis

Achieves scalability and precision.

State Inconsistency (SI)

Smart contract: C

Methods: $(f_1, f_2, \ldots, f_n)$

Schedule: H contains ordered external/public function invocations of C

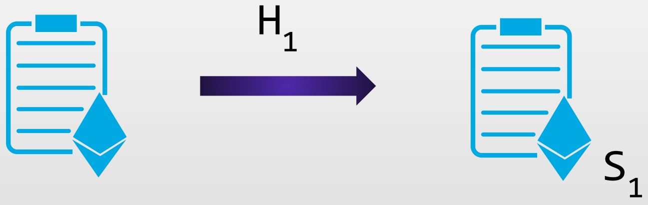
State Inconsistency (SI)

Smart contract: C

Methods: $(f_1, f_2, \ldots, f_n)$

Schedule: H contains ordered external/public function invocations of C

Initial state: S_0

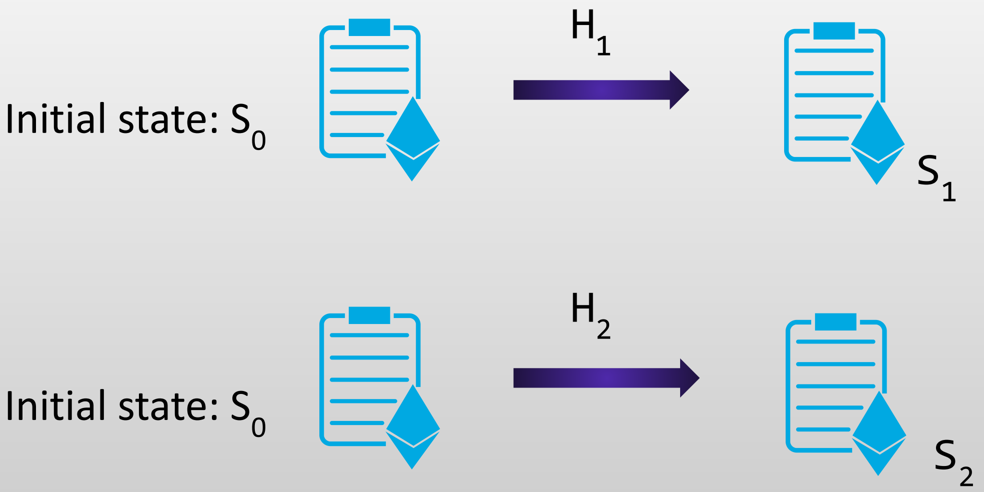


State Inconsistency (SI)

Smart contract: C

Methods: $(f_1, f_2, \ldots, f_n)$

Schedule: H contains ordered external/public function invocations of C

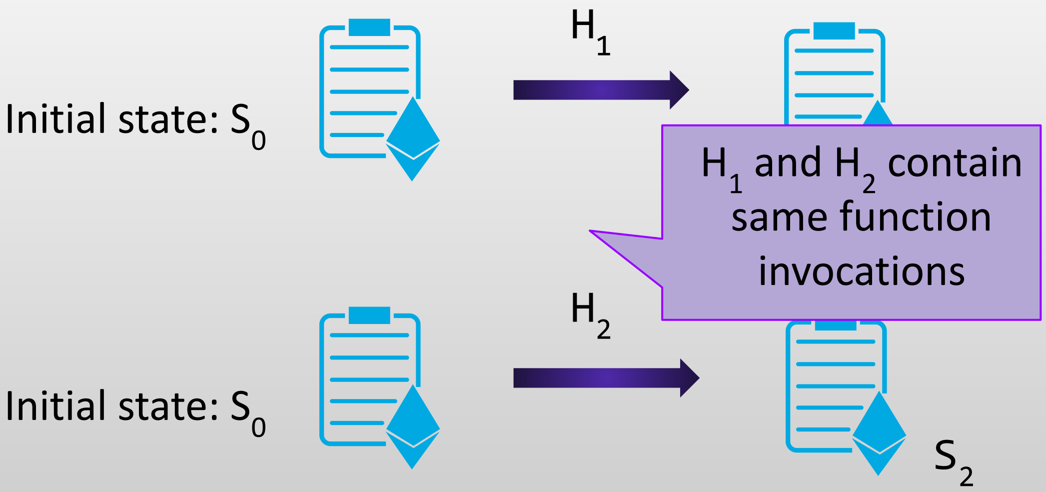


State Inconsistency (SI)

Smart contract: C

Methods: $(f_1, f_2, \ldots, f_n)$

Schedule: H contains ordered external/public function invocations of C

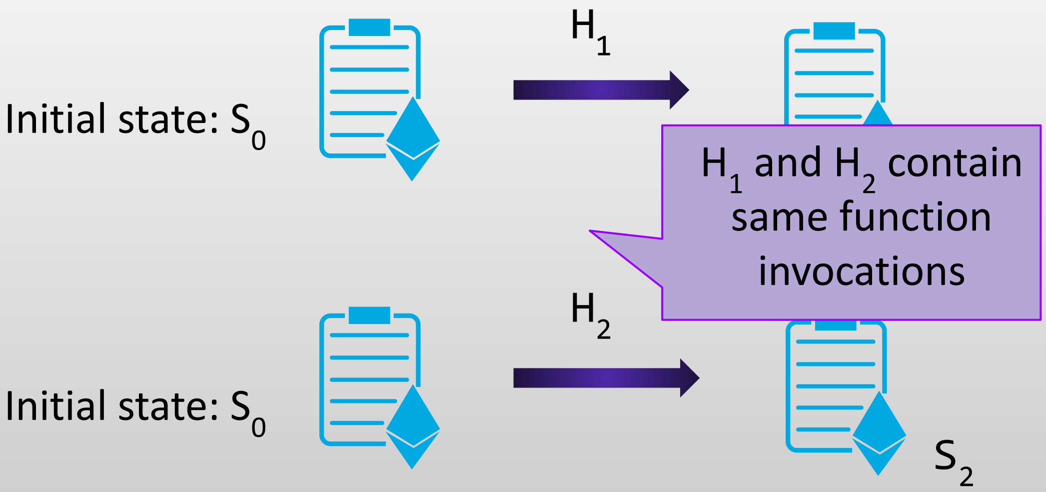


State Inconsistency (SI)

Smart contract: C

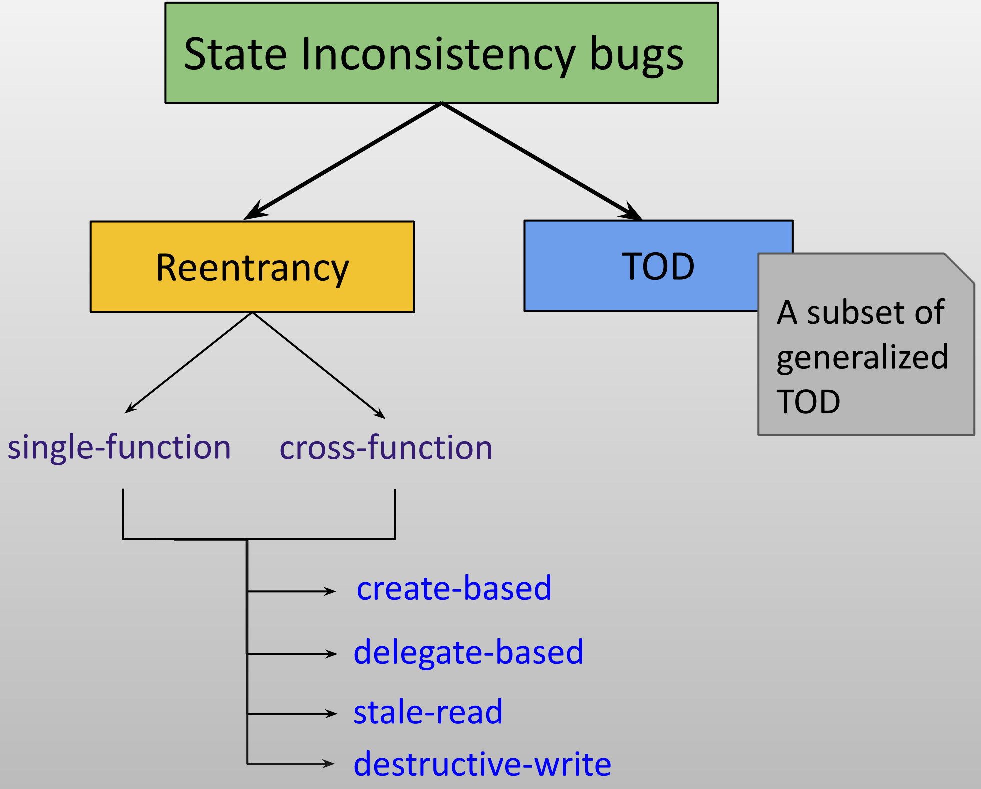
Methods: $(f_1, f_2, \ldots, f_n)$

Schedule: H contains ordered external/public function invocations of C

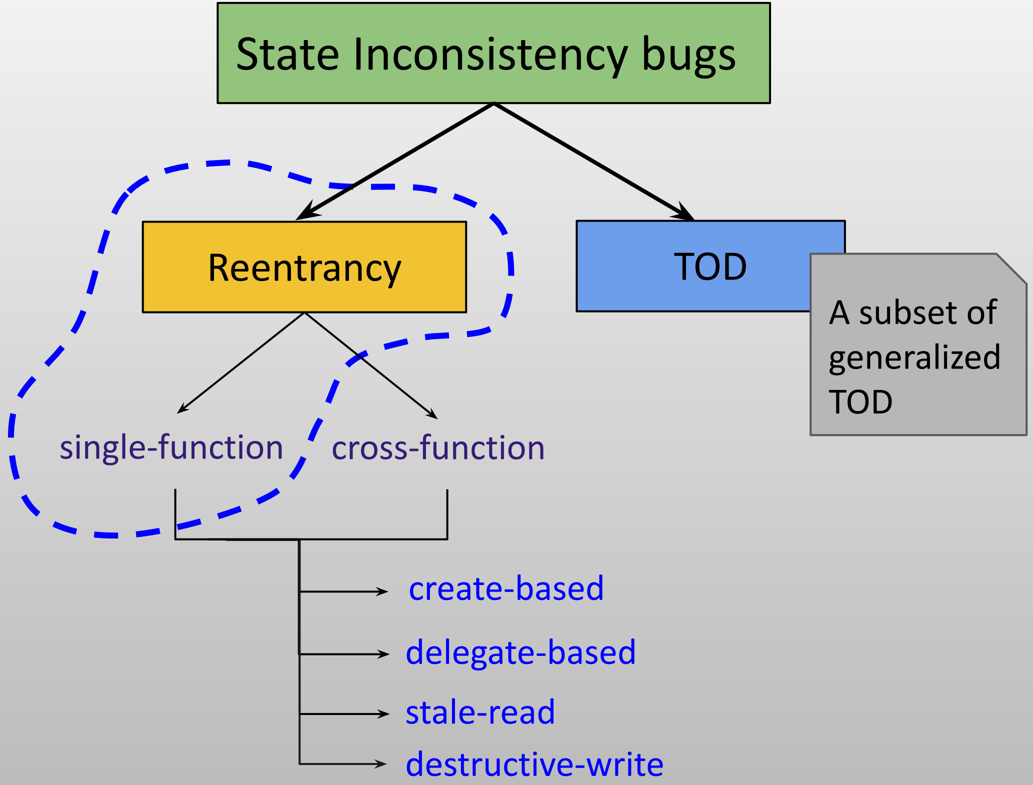


If $H_1 \neq H_2$ and $S_1 \neq S_2$, contract C is said to have a **State Inconsistency** bug

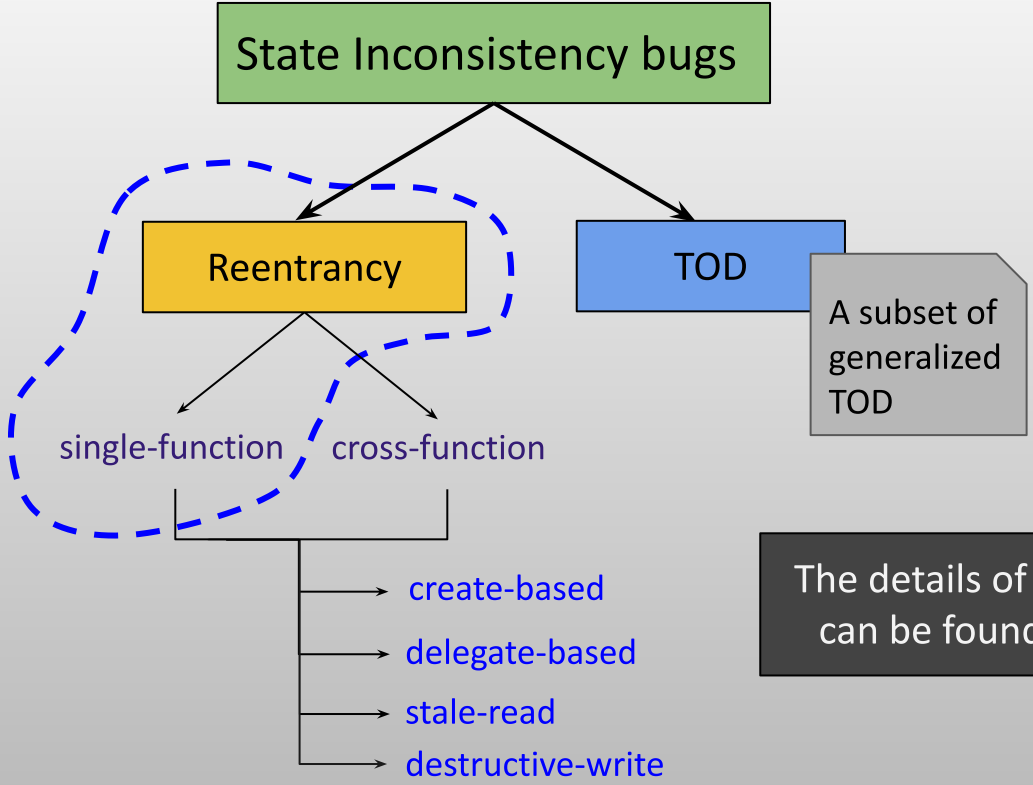
State Inconsistency bugs



State Inconsistency bugs



State Inconsistency bugs



The details of the other bugs can be found in our paper

The Reentrancy problem

User state

User’s balance: 0

User

```
bank.withdraw(100)
```

Bank

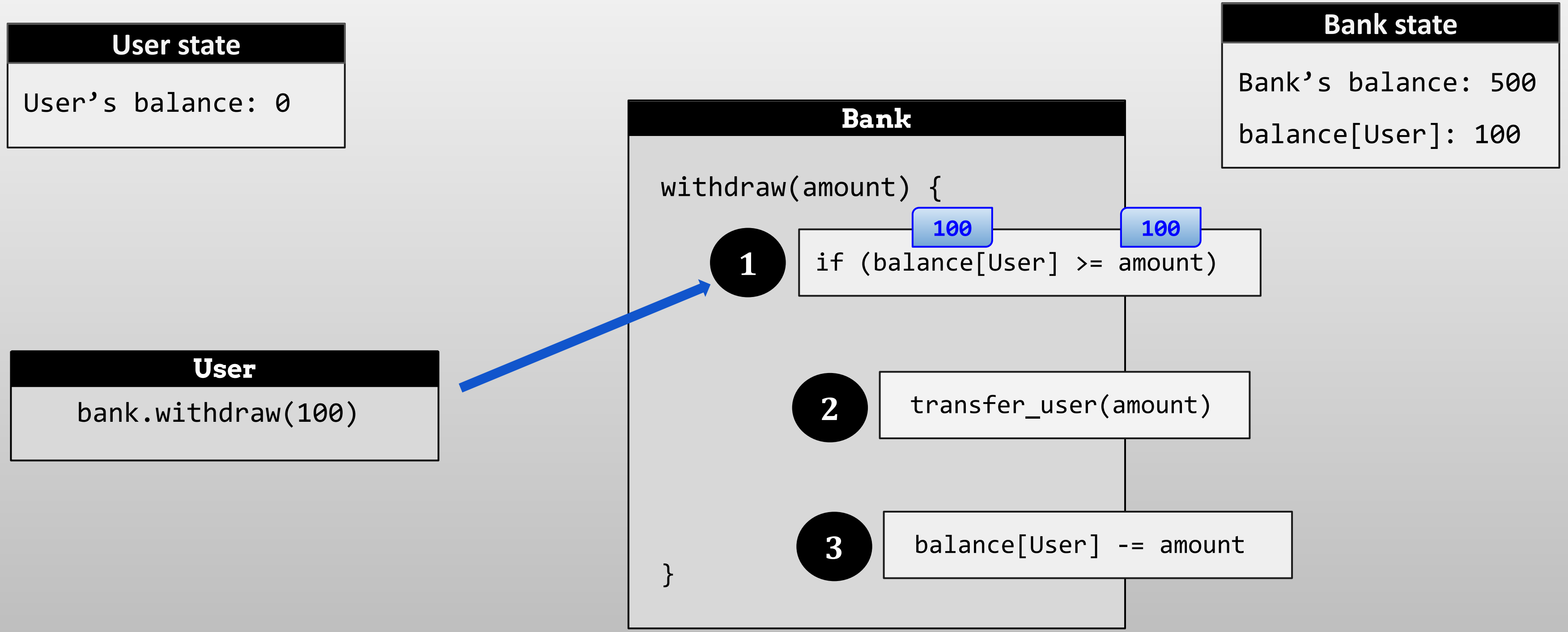
```
withdraw(amount) {
    1  if (balance[User] >= amount)
    2  transfer_user(amount)
    3  balance[User] -= amount
}
```

Bank state

Bank’s balance: 500

balance[User]: 100

The Reentrancy problem



The Reentrancy problem

User state

User's balance: 0

User

```
bank.withdraw(100)
```

Bank

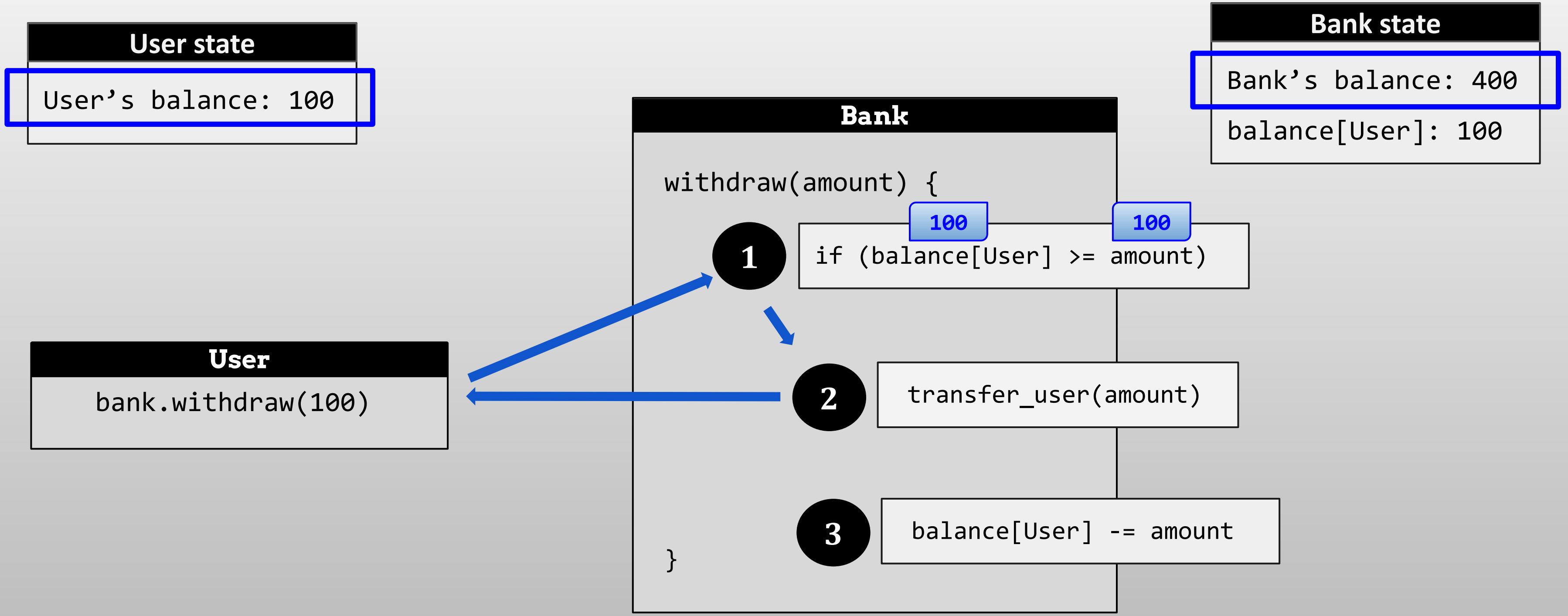
```
withdraw(amount) {
   1  if (balance[User] >= amount)      [balance[User] = 100, amount = 100]
   2  transfer_user(amount)
   3  balance[User] -= amount
}
```

Bank state

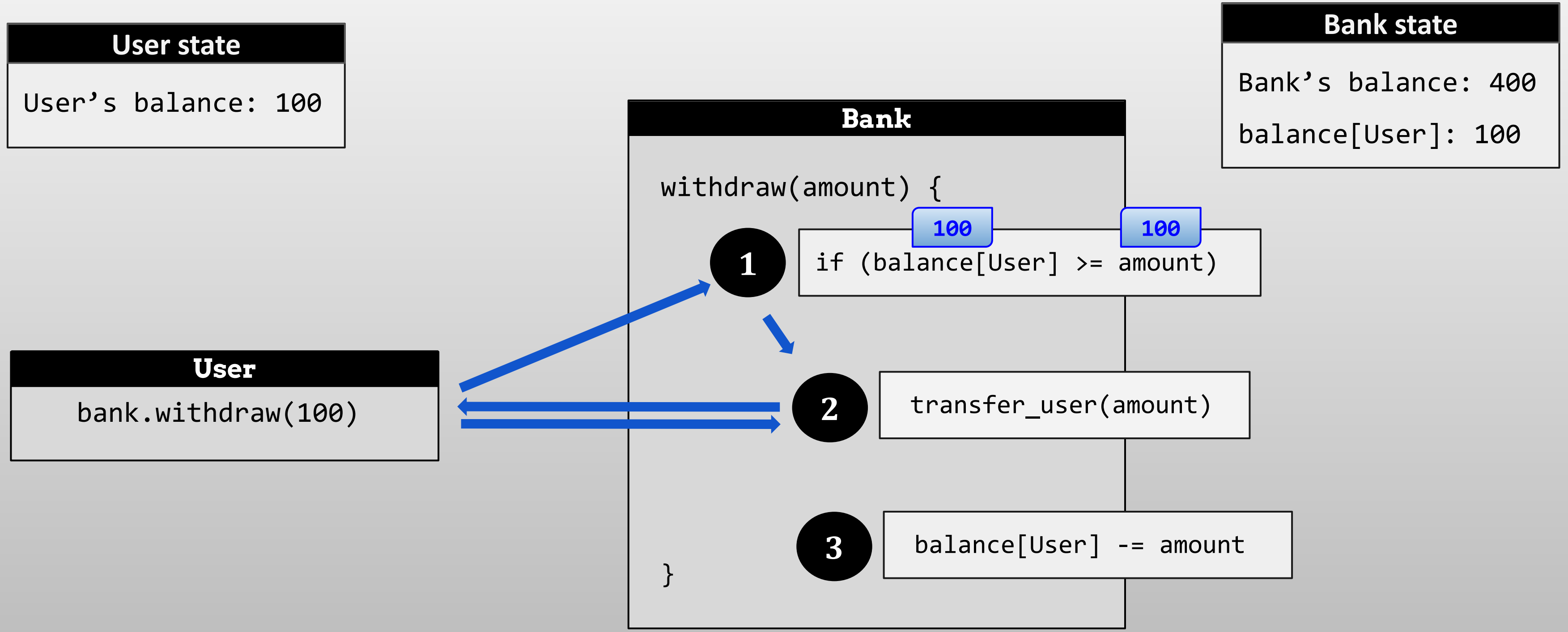
Bank's balance: 500

balance[User]: 100

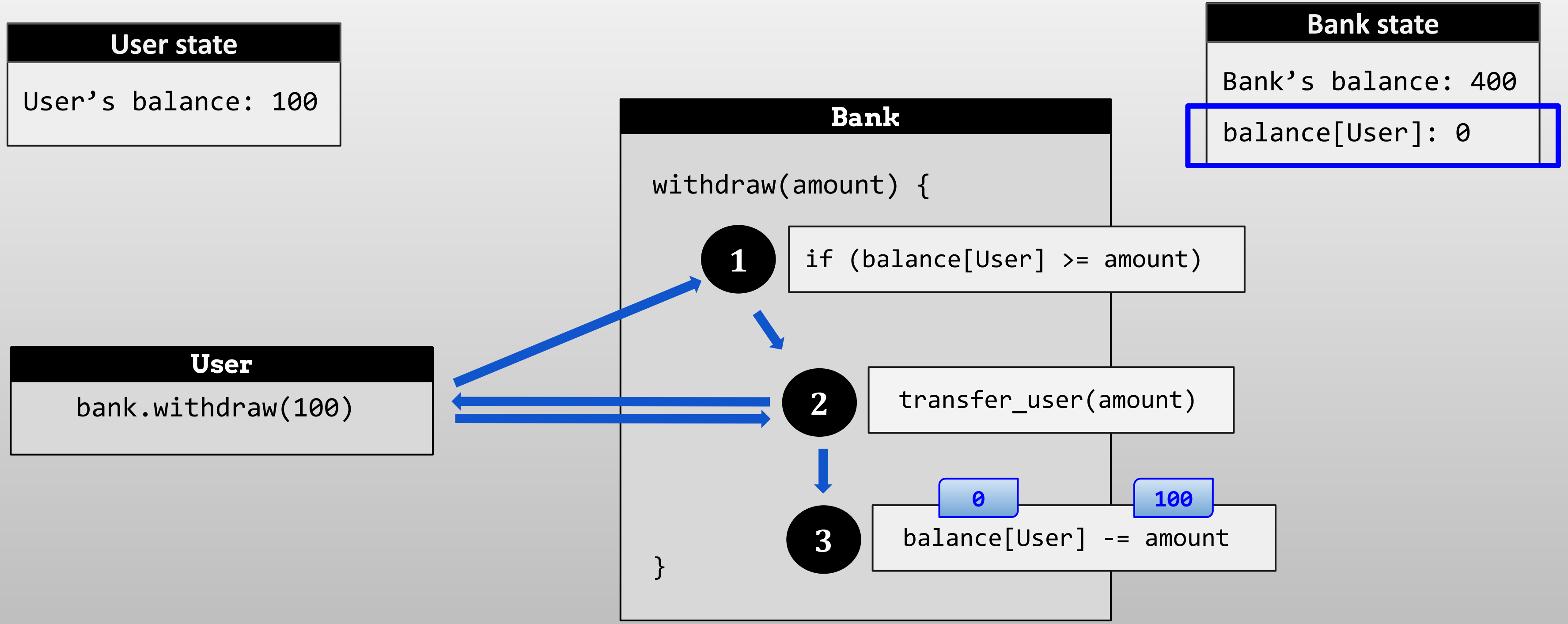
The Reentrancy problem



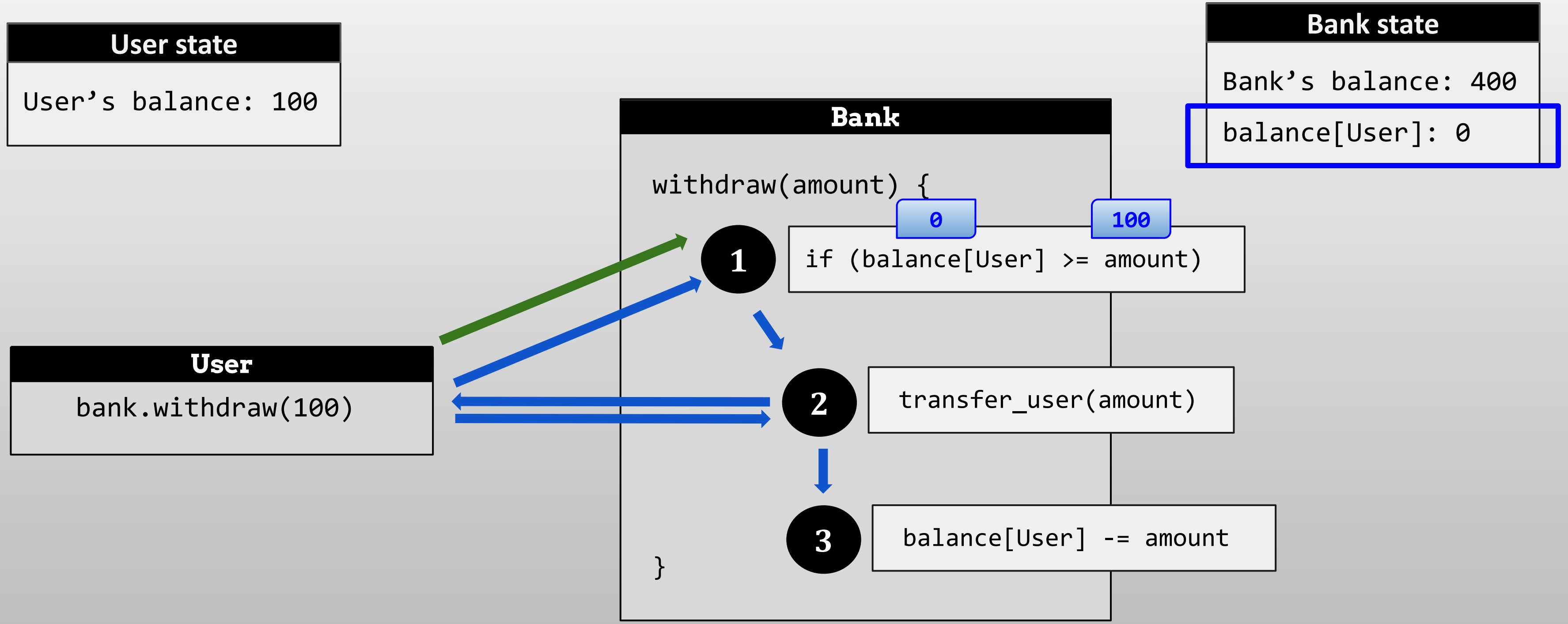
The Reentrancy problem



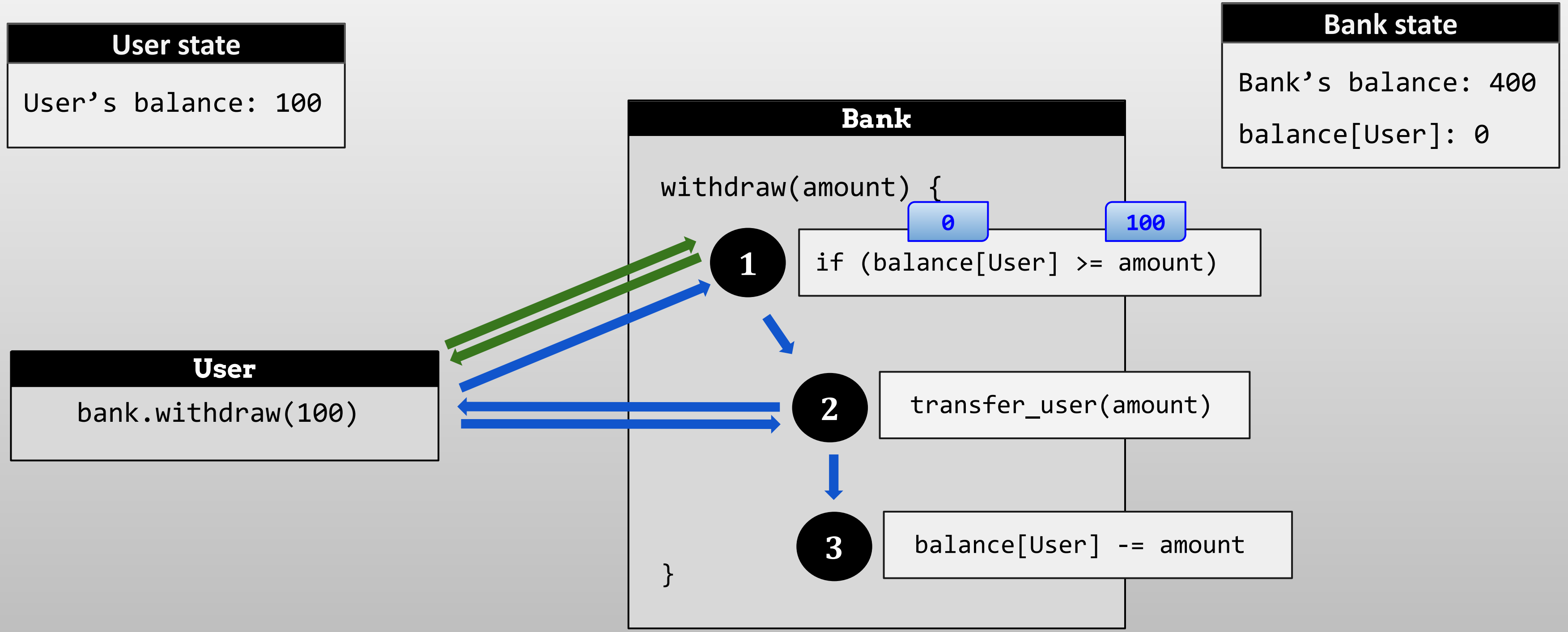
The Reentrancy problem



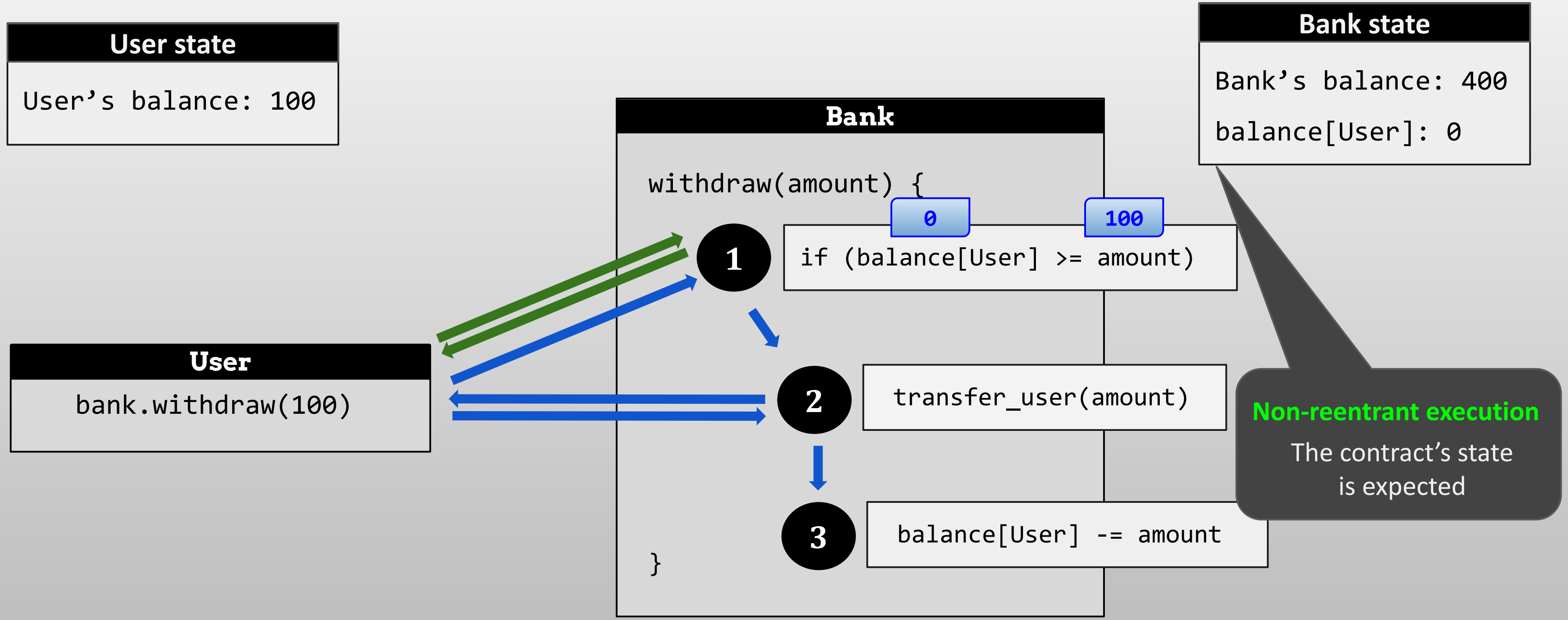
The Reentrancy problem



The Reentrancy problem



The Reentrancy problem



The Reentrancy problem

User state

User's balance: 100

Bank state

Bank's balance: 400

balance[User]: 100

Attacker

```
bank.withdraw(100)
```

Bank

```
withdraw(amount) {
  1  if (balance[User] >= amount)
  2  transfer_user(amount)
  3  balance[User] -= amount
}
```

100 100

The Reentrancy problem

User state

User's balance: 100

Bank state

Bank's balance: 400

balance[User]: 100

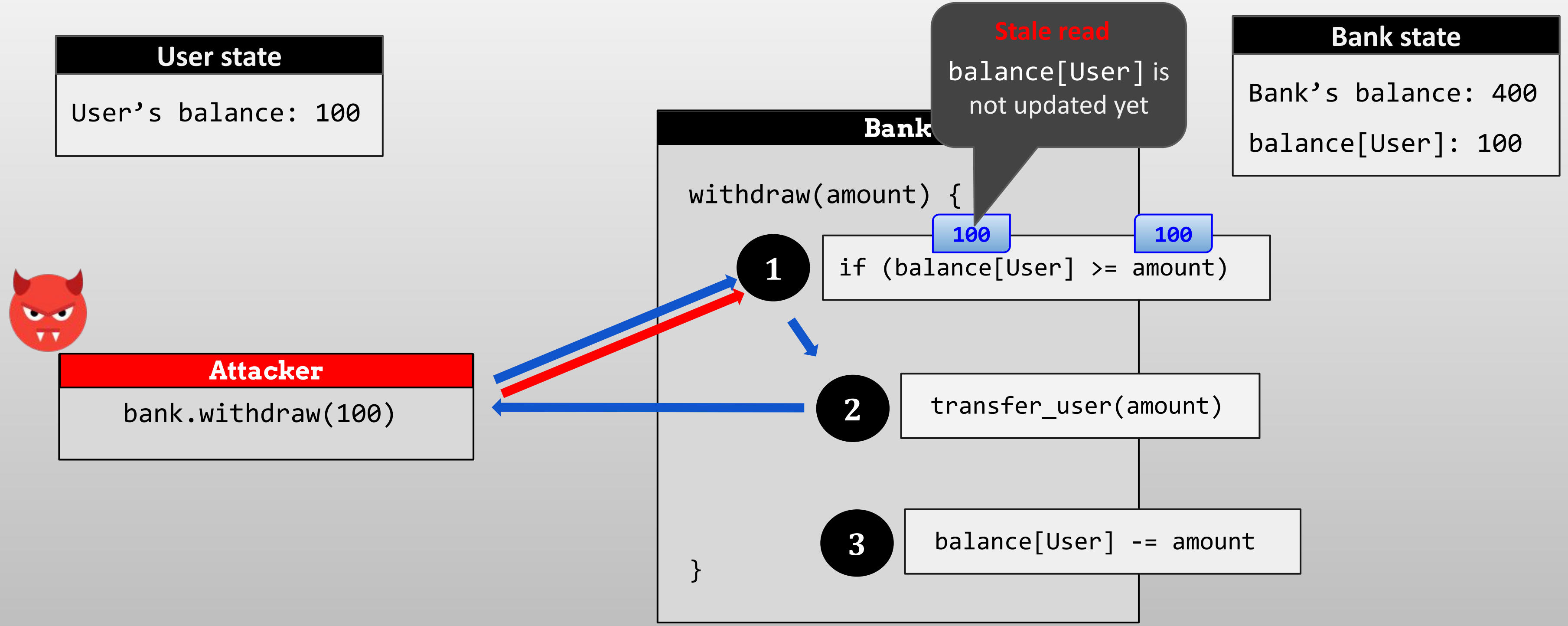
Attacker

```
bank.withdraw(100)
```

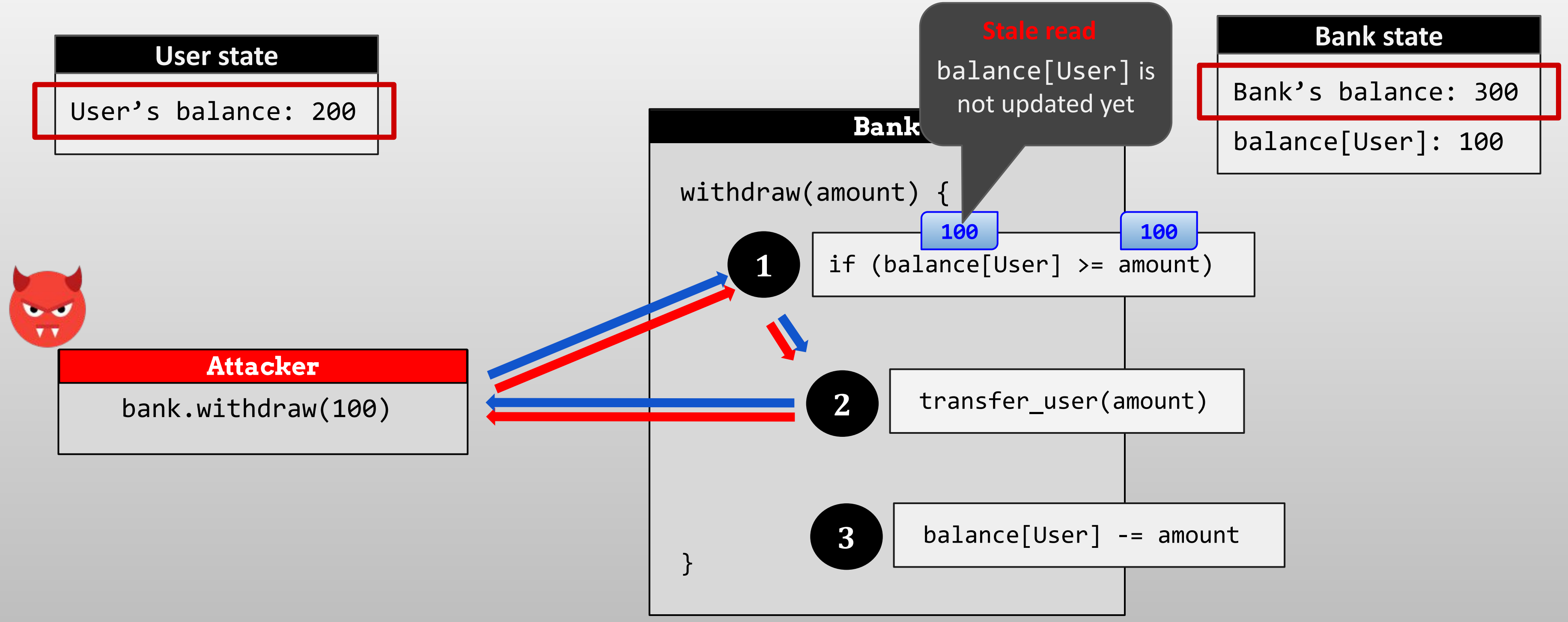
Bank

```
withdraw(amount) {
  1  if (balance[User] >= amount)      [100]  [100]
  2  transfer_user(amount)
  3  balance[User] -= amount
}
```

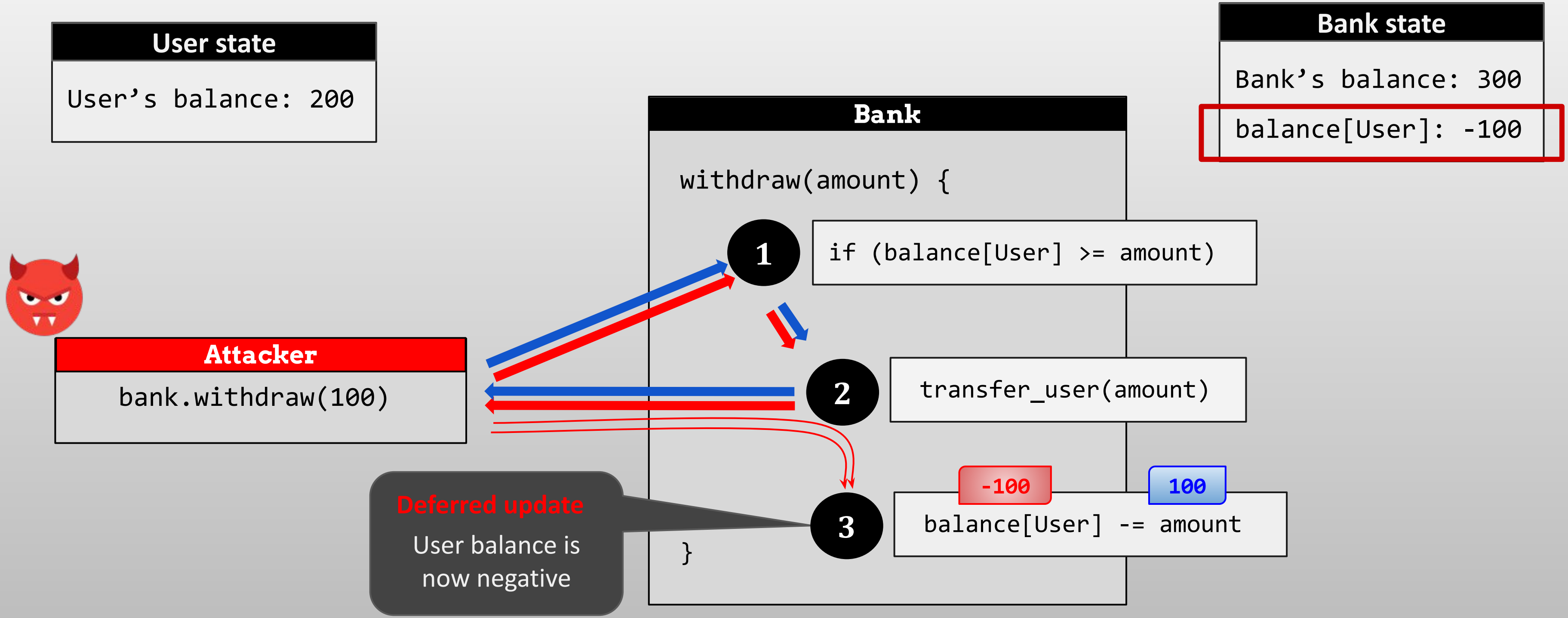
The Reentrancy problem



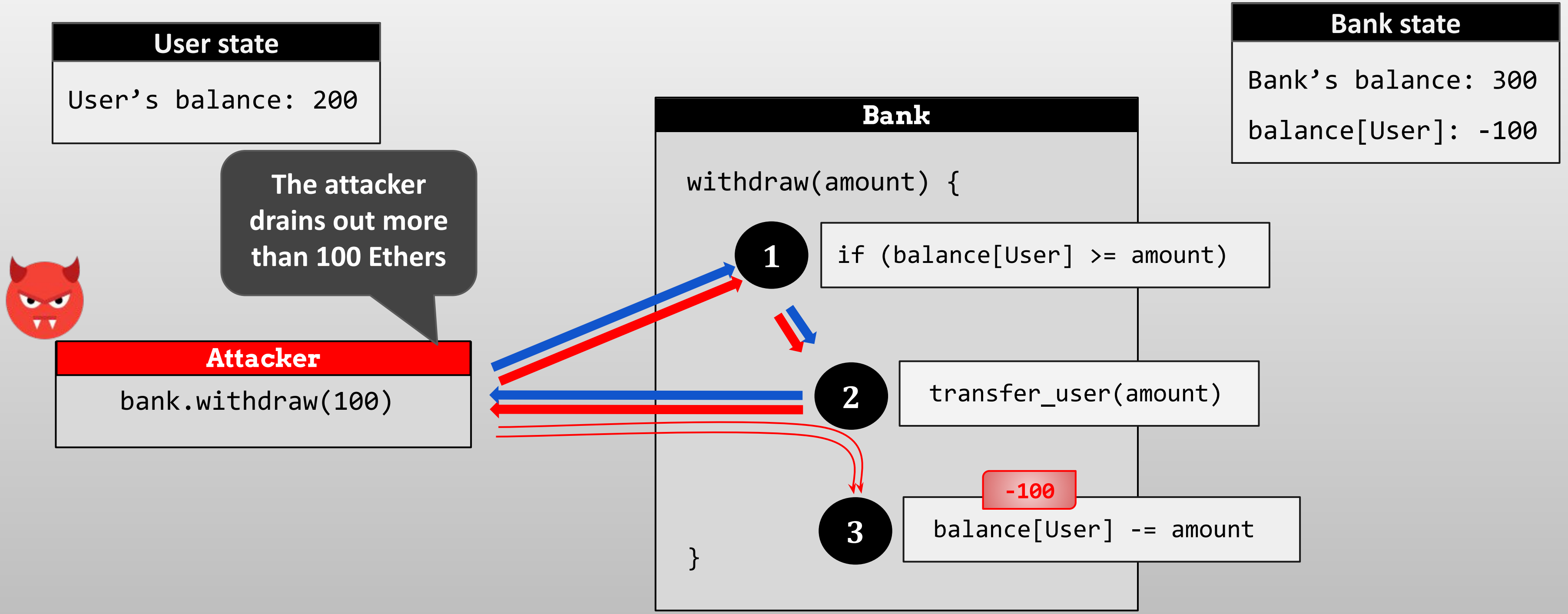
The Reentrancy problem



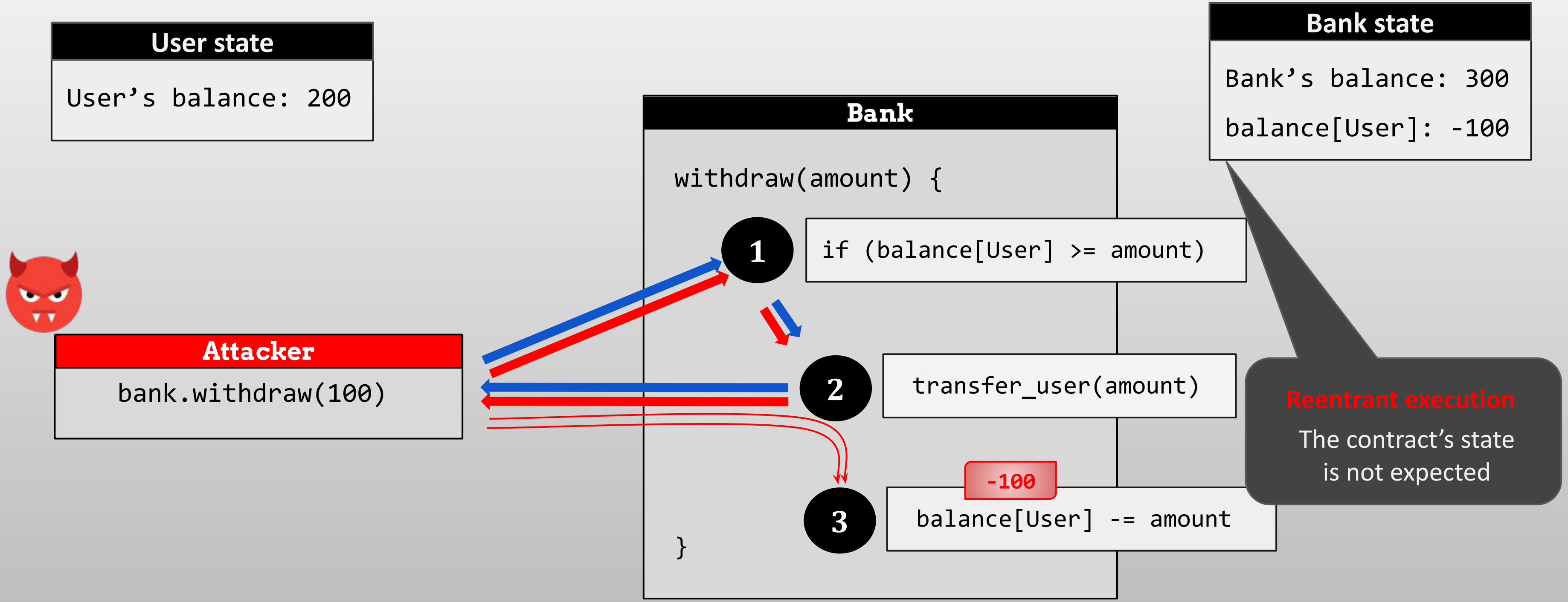
The Reentrancy problem



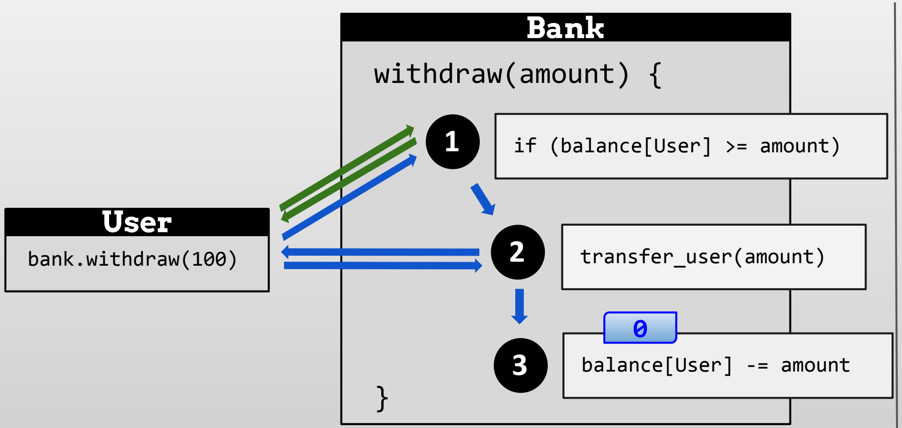
The Reentrancy problem



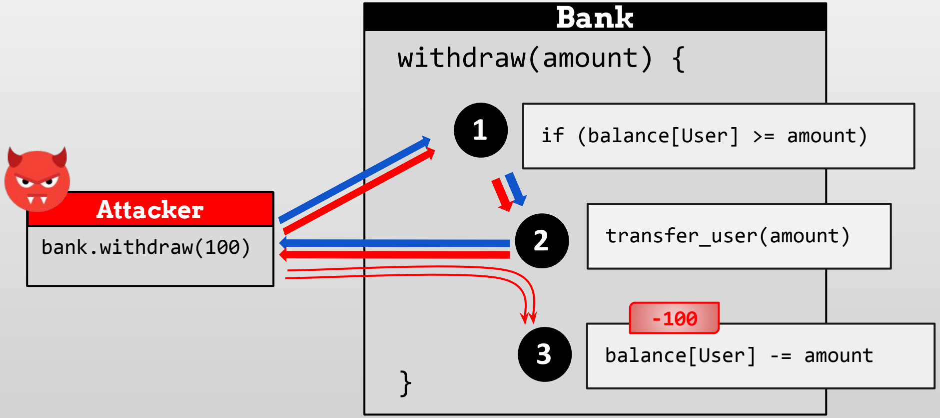
The Reentrancy problem



SI and Reentrancy

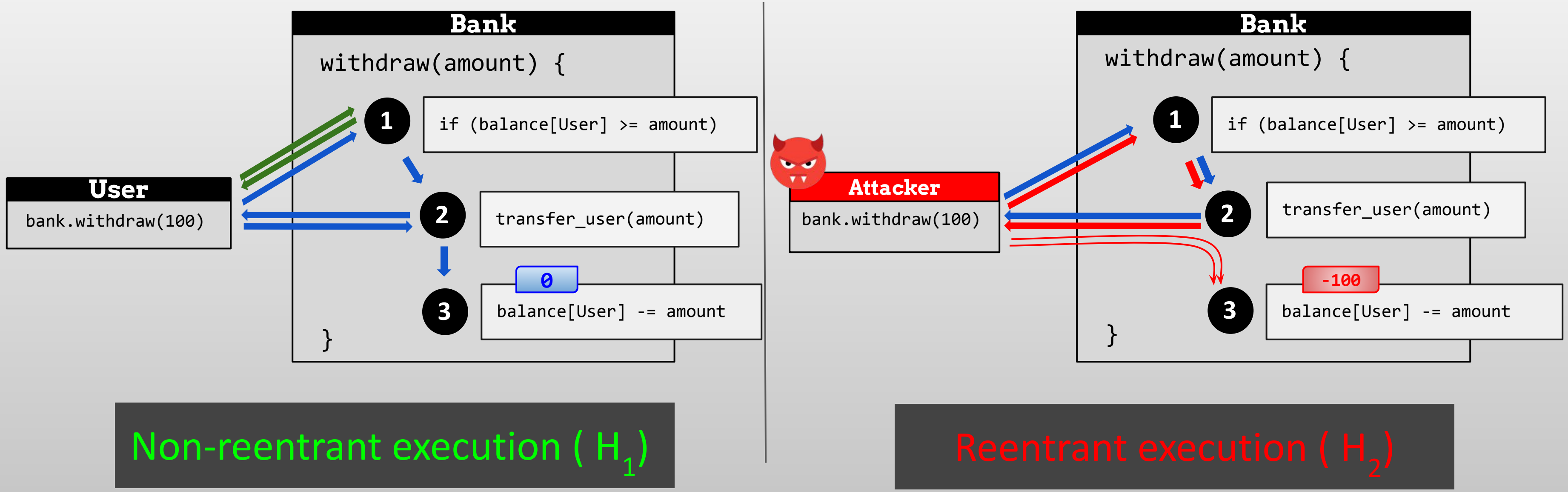


Non-reentrant execution (H_1)



Reentrant execution (H_2)

SI and Reentrancy



$H_1 \neq H_2$ and Final state@$H_1 \neq$ Final state@H_2

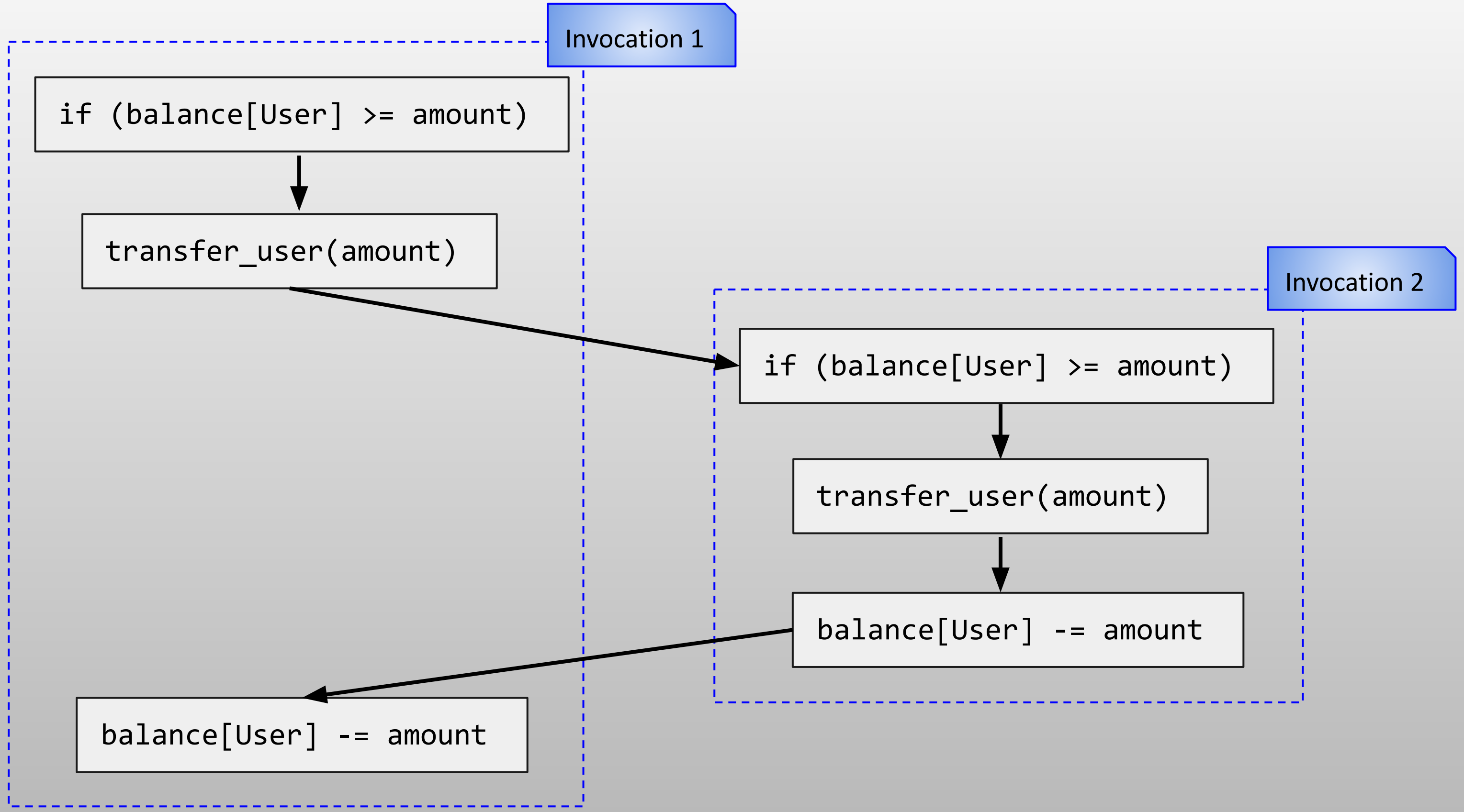
SI bugs: Intuition

- SI bugs occur because different schedules result in different final state, i.e., different values of storage variables.

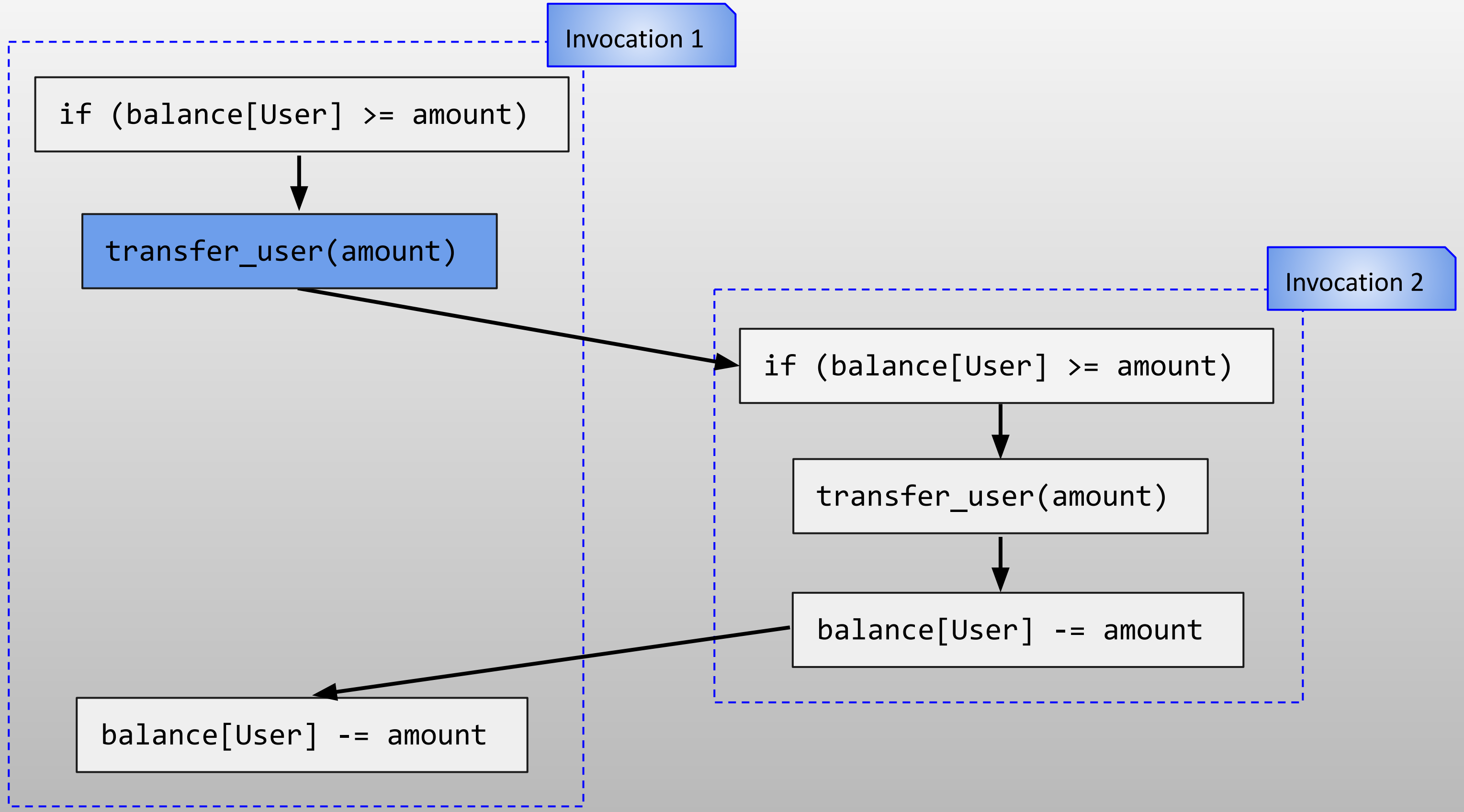
- Two such schedules can result in different contract states if:
  - There exist two operations, at least one is a write access, on a common storage variable
  - The relative order of such operations differ in these two schedules.

We define such a read-write hazard as ***Hazardous access***

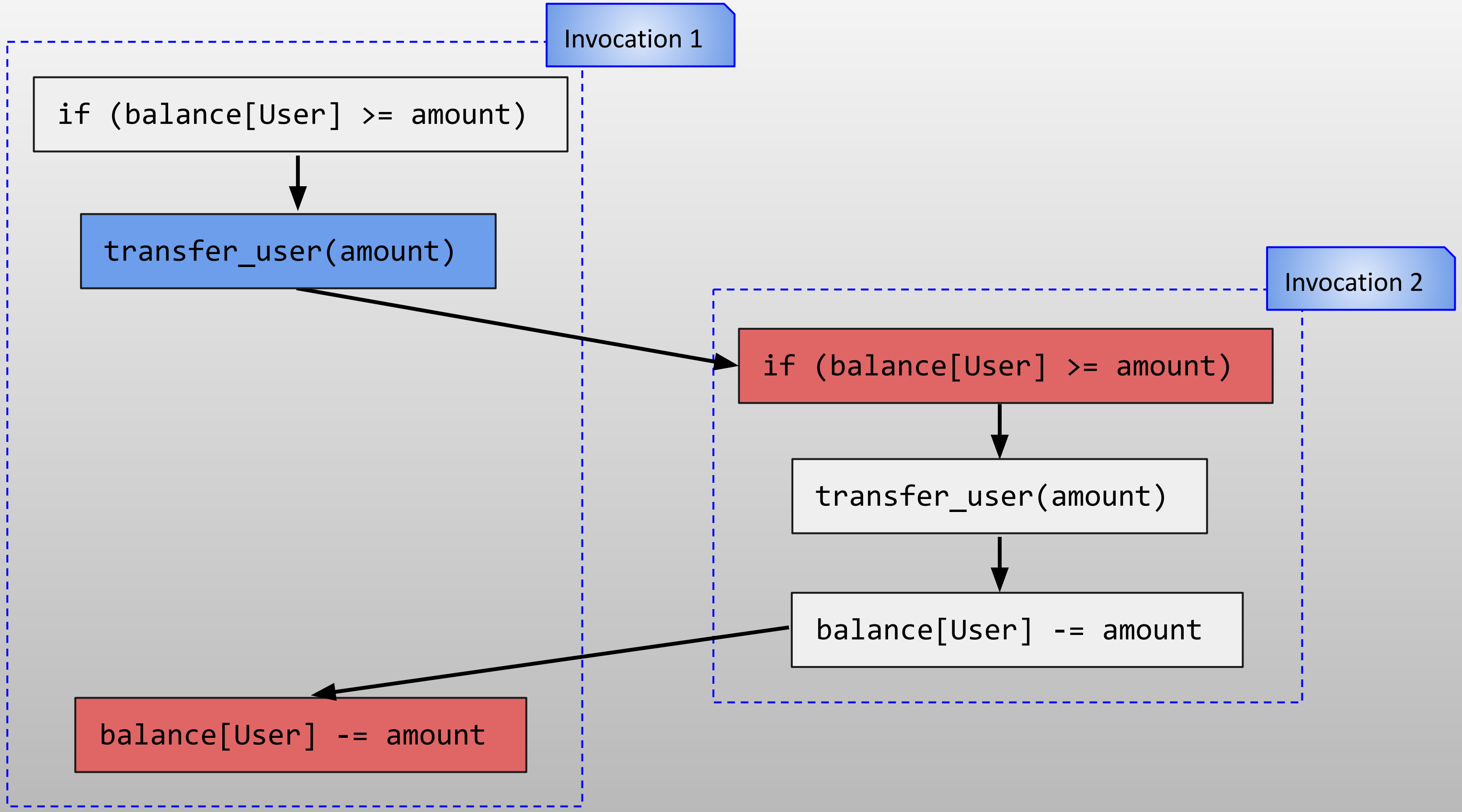
Hazardous access



Hazardous access

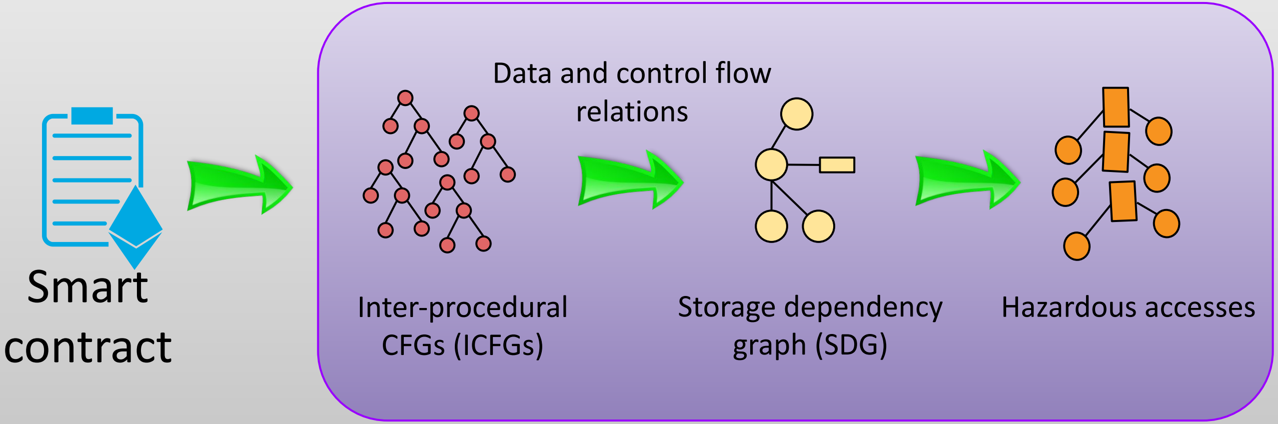


Hazardous access



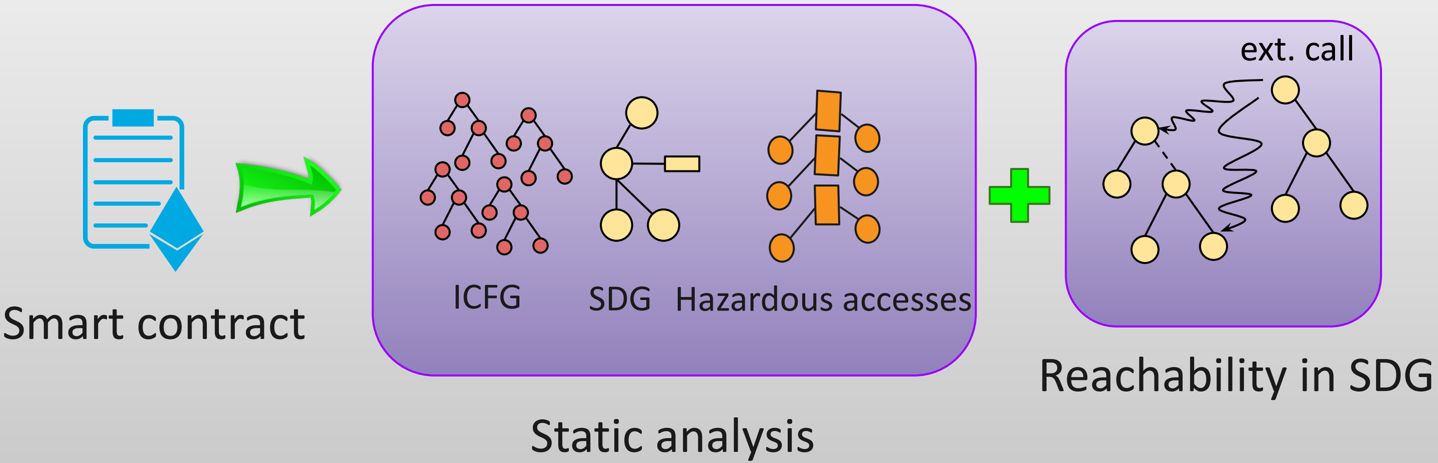
Explorer: Hazardous access detection

Explorer: Hazardous access detection

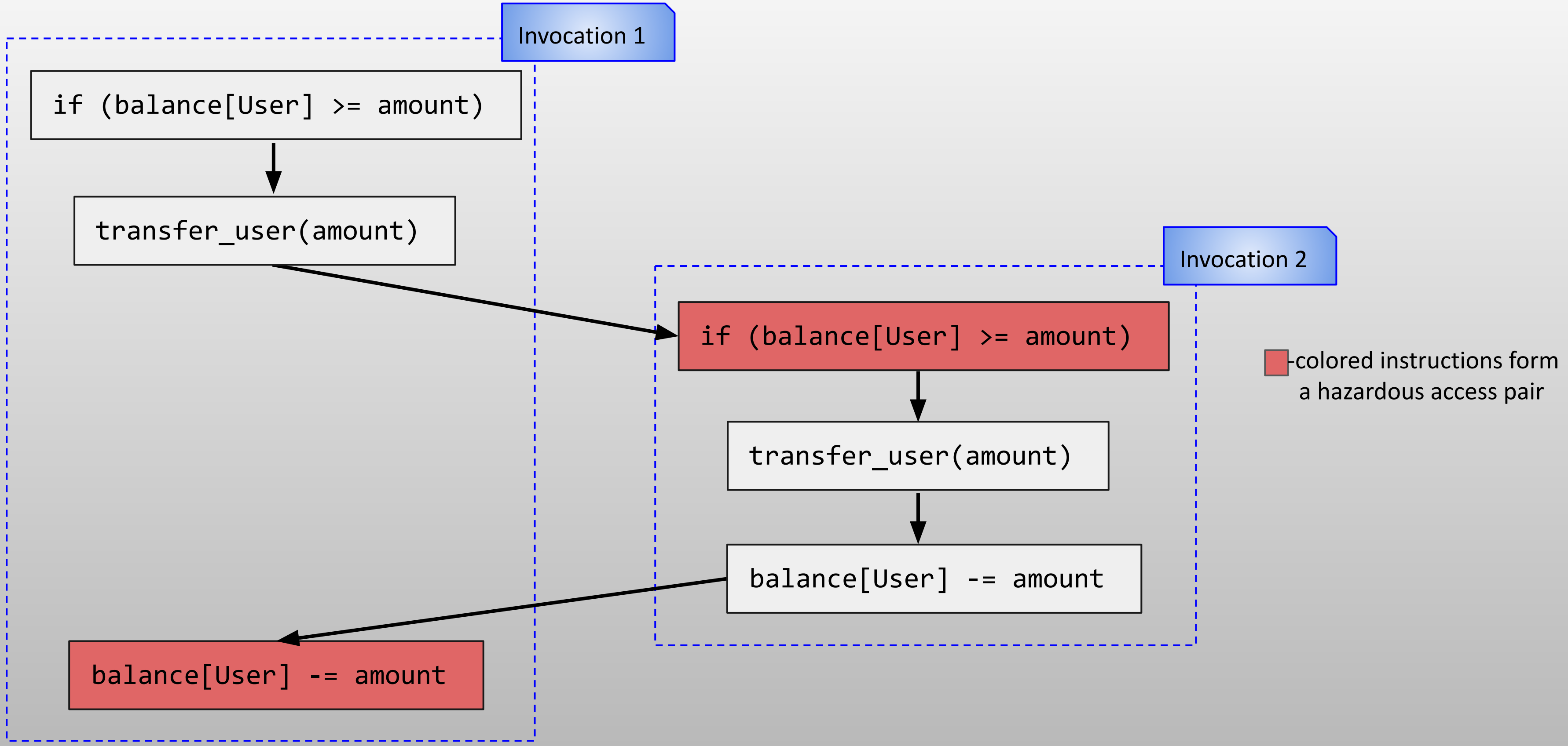


Static analysis

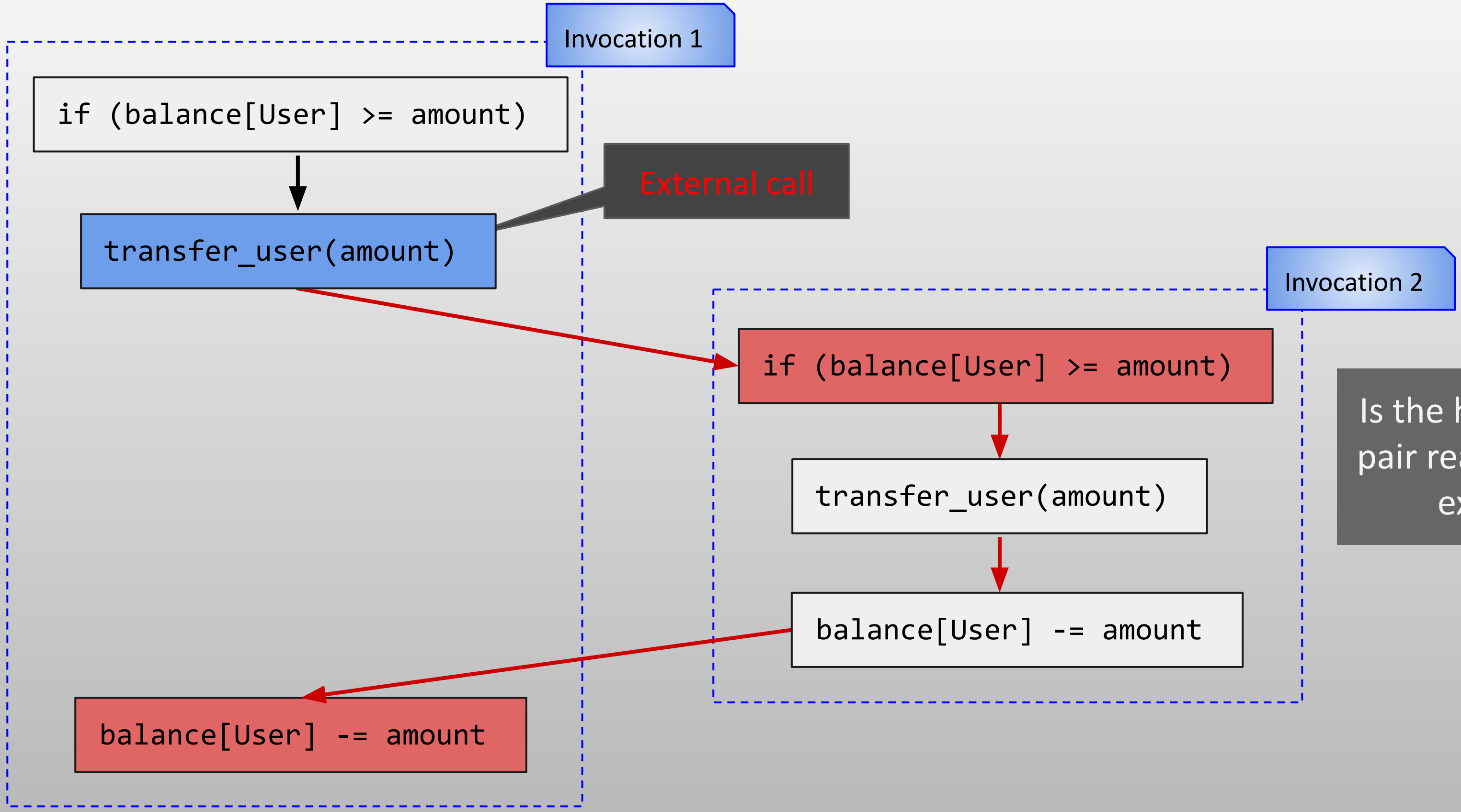
Explorer: Reachability in SDG



Explorer: Reachability in SDG

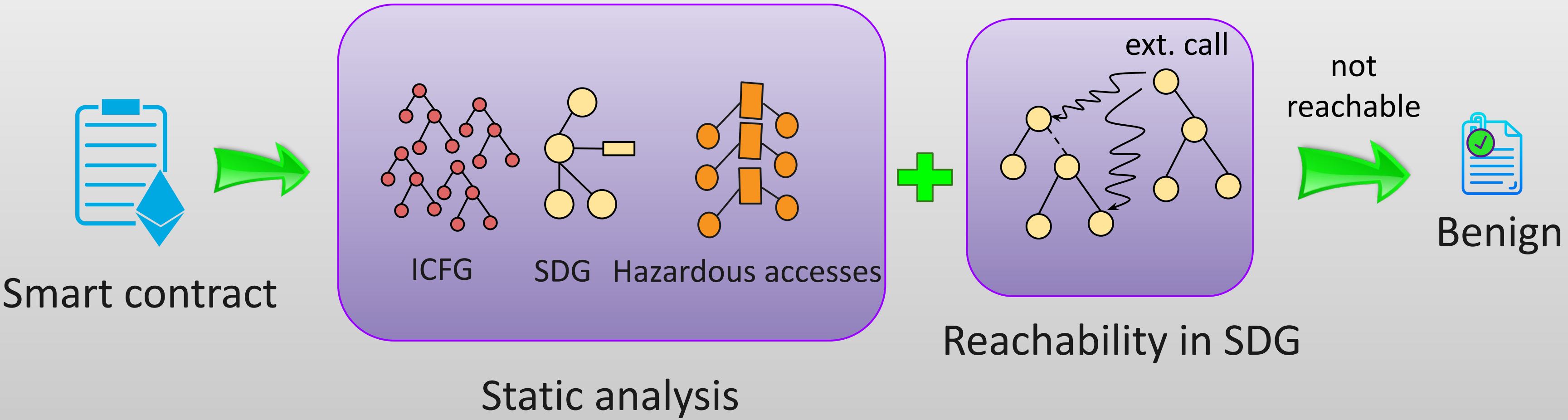


Explorer: Reachability in SDG

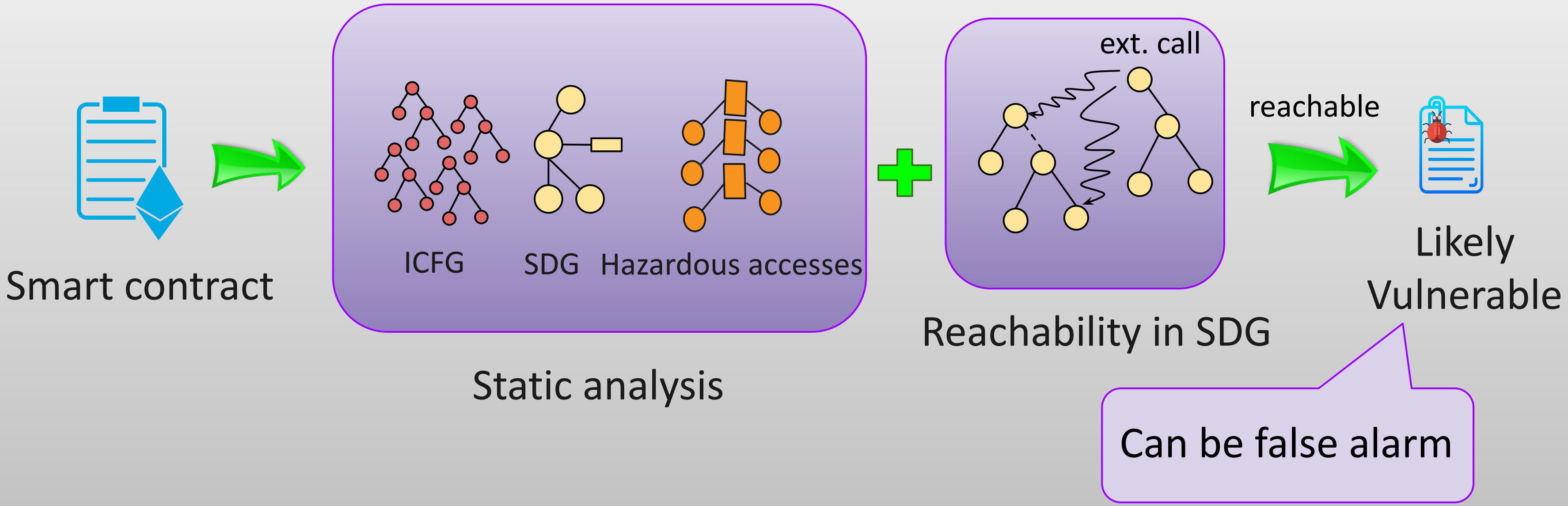


Is the hazardous access pair reachable from the external call?

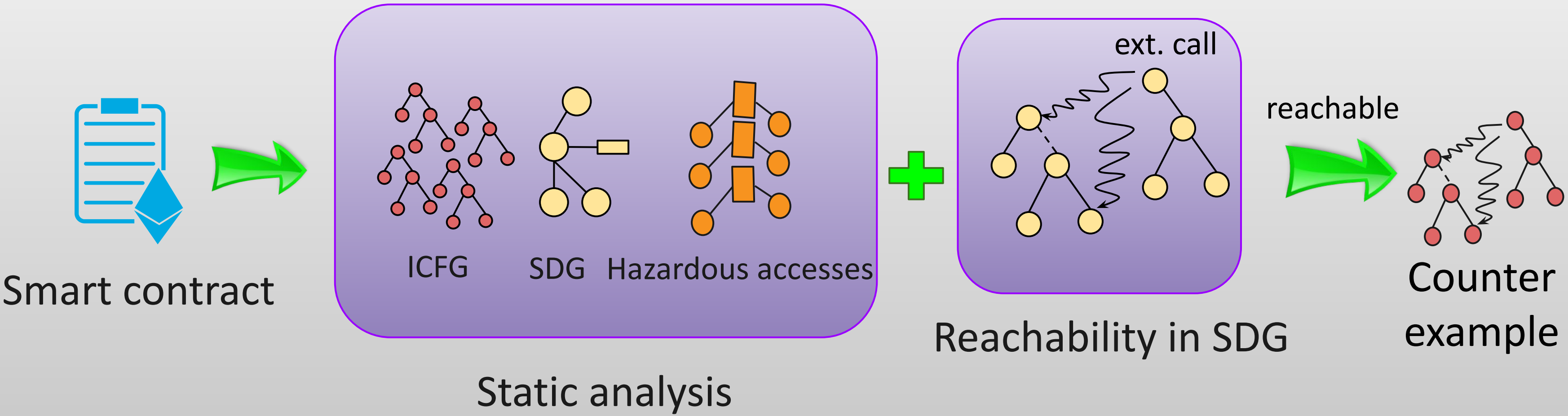
Explorer: Reachability in SDG



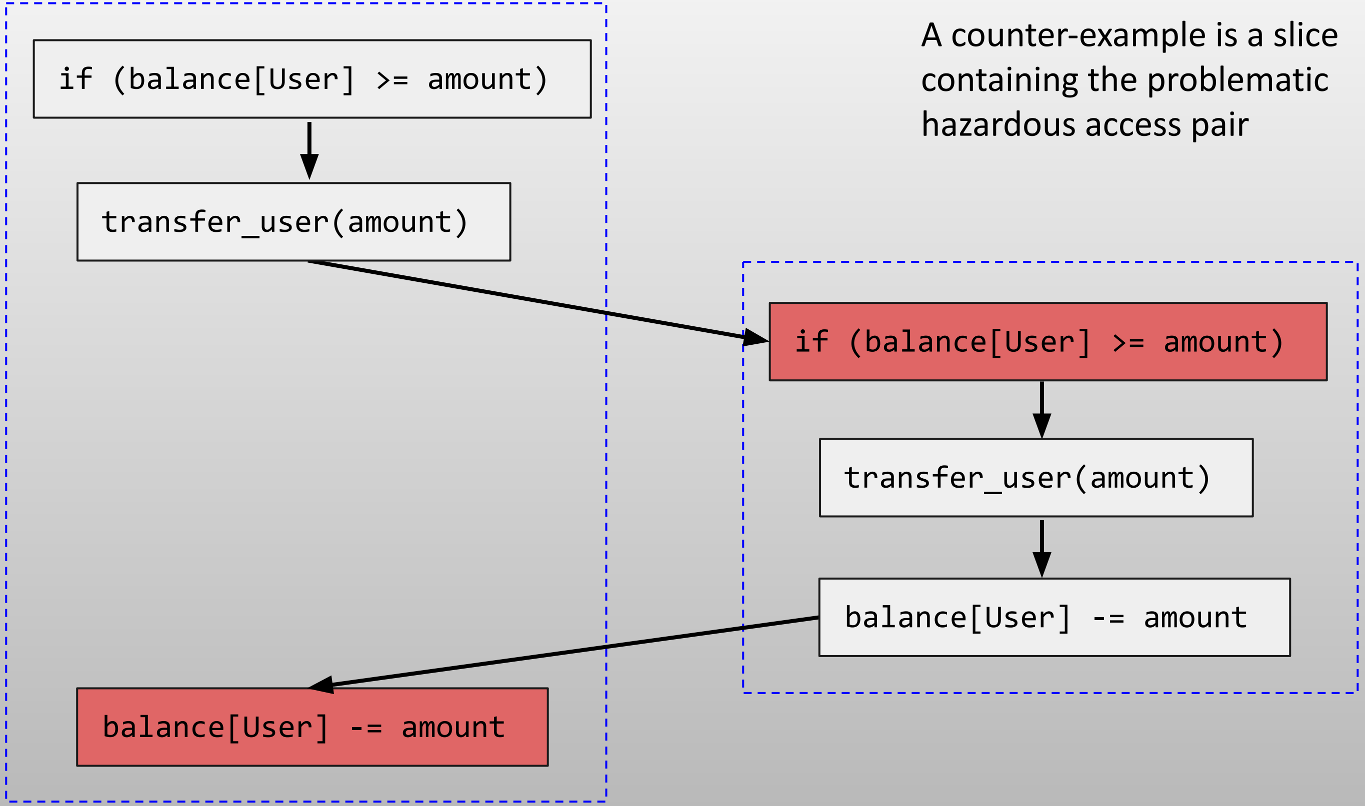
Explorer: Reachability in SDG



Explorer: Counter-example generation



Explorer: Counter-example generation



A counter-example is a slice containing the problematic hazardous access pair

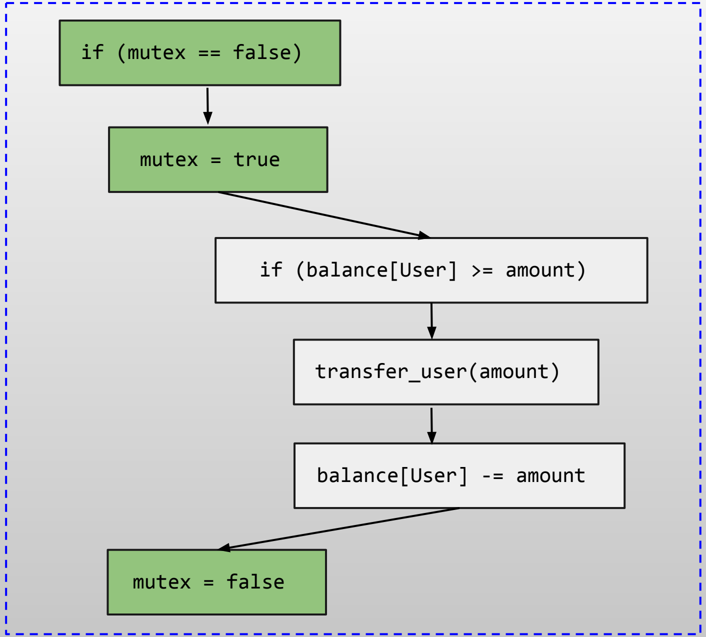
Refiner: Value-summary analysis + Symbolic execution

SECLAB

- Value-summary analysis (VSA) outputs the potential symbolic values of each storage variable under different constraints

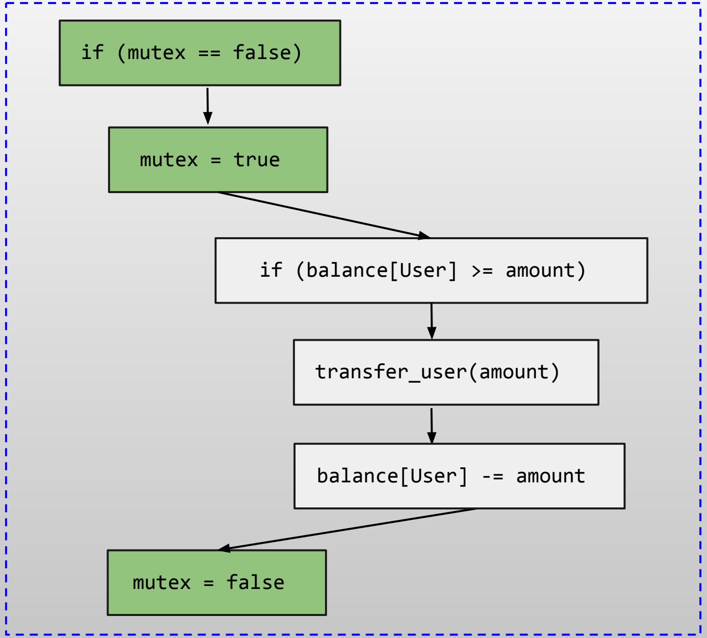
- This will be used as the precondition of the symbolic evaluation

Refiner: Value-summary analysis + Symbolic execution



withdraw

Refiner: Value-summary analysis + Symbolic execution



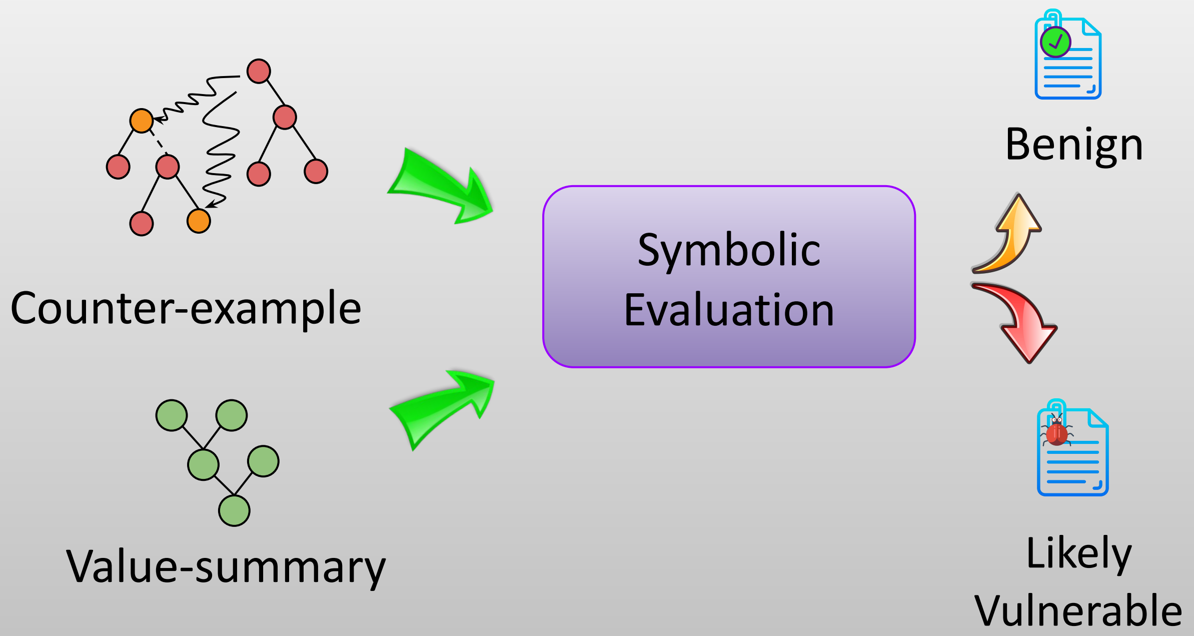
withdraw

Value-summary analysis outputs the following for `mutex`:

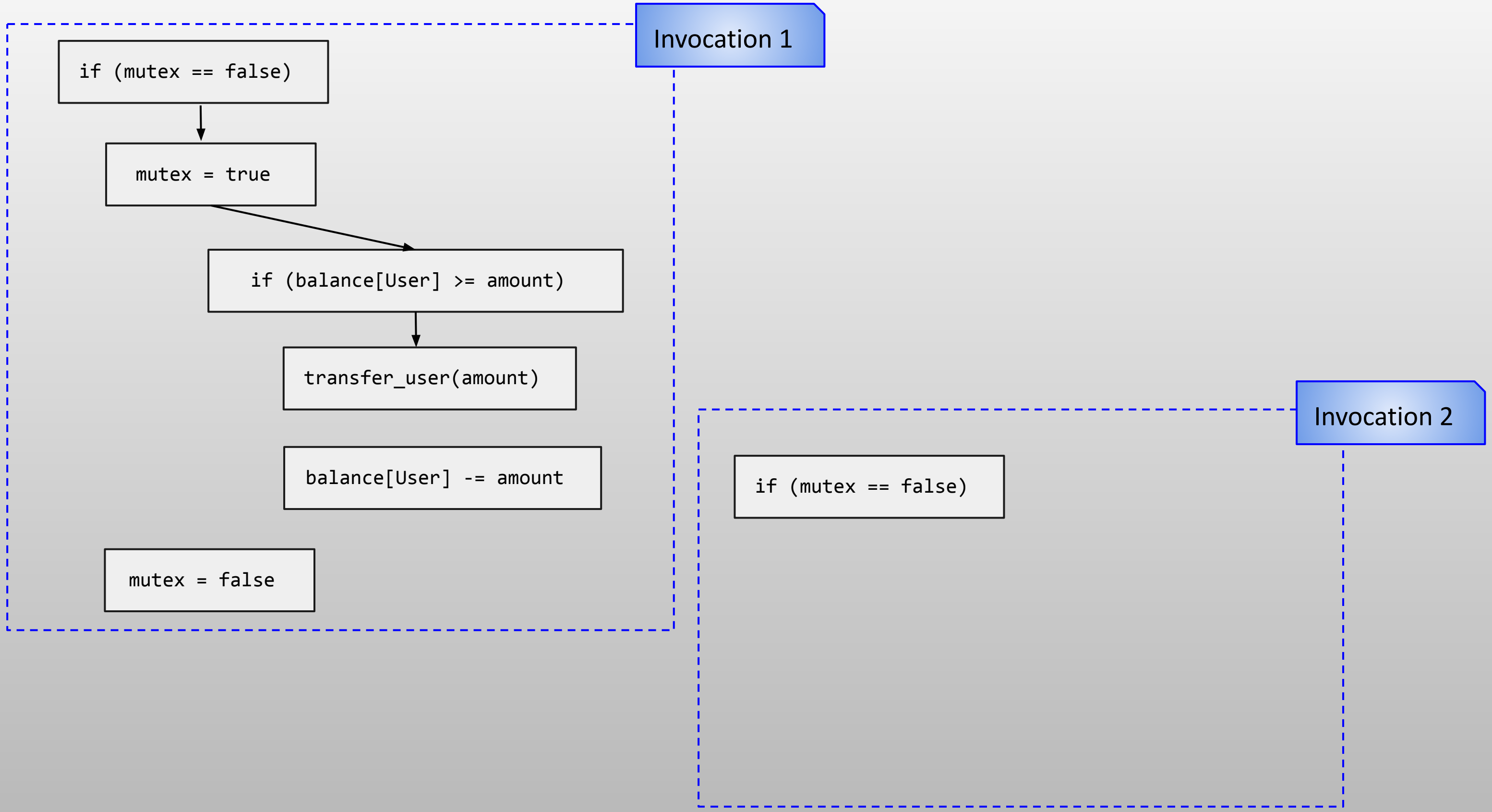
```
mutex: {<mutex=false, false>,
        <mutex=false, true>}
```

pre-condition   value

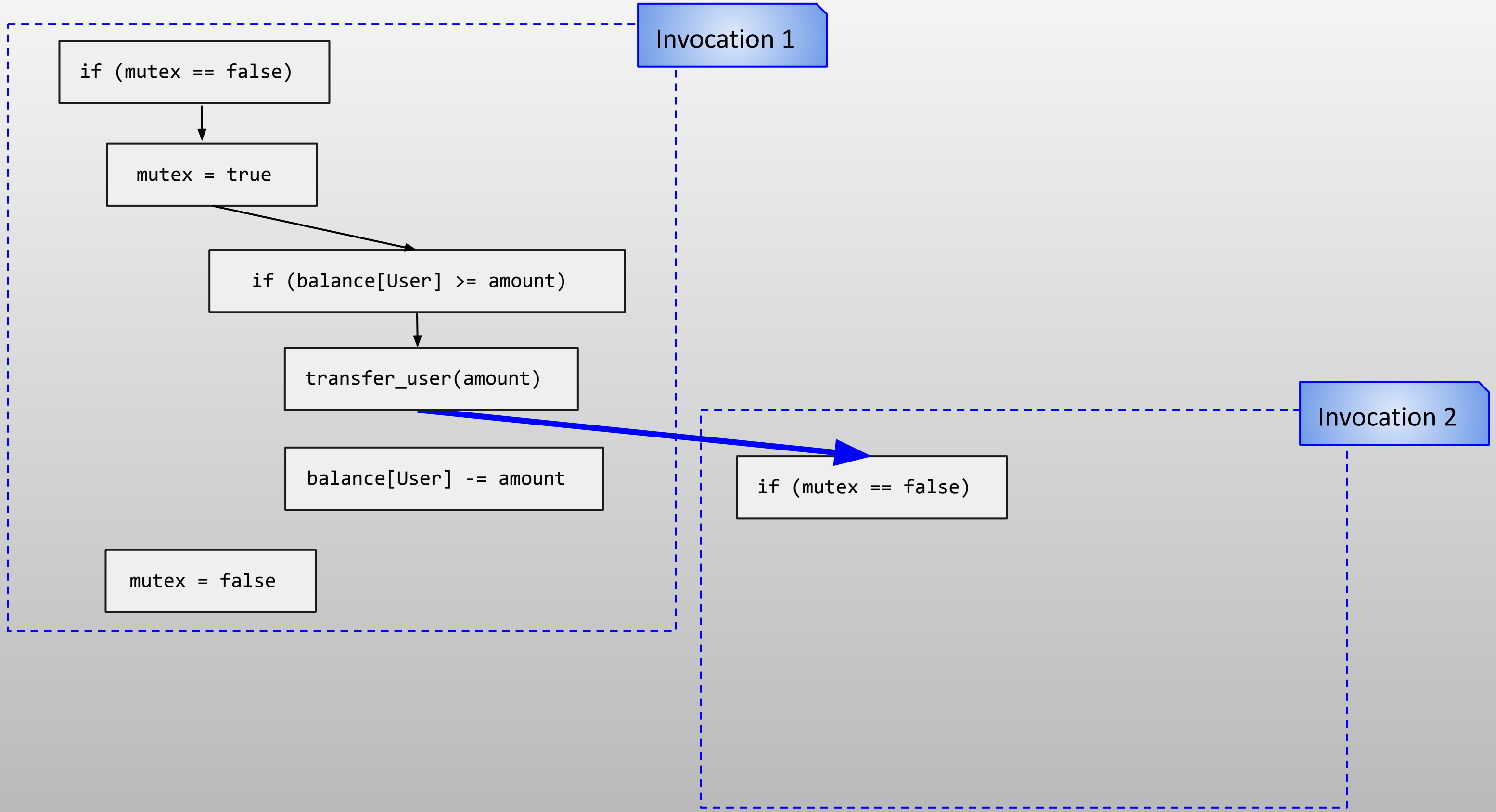
Refiner: Value-summary analysis + Symbolic execution



Refiner: Value-summary analysis + Symbolic execution



Refiner: Value-summary analysis + Symbolic execution

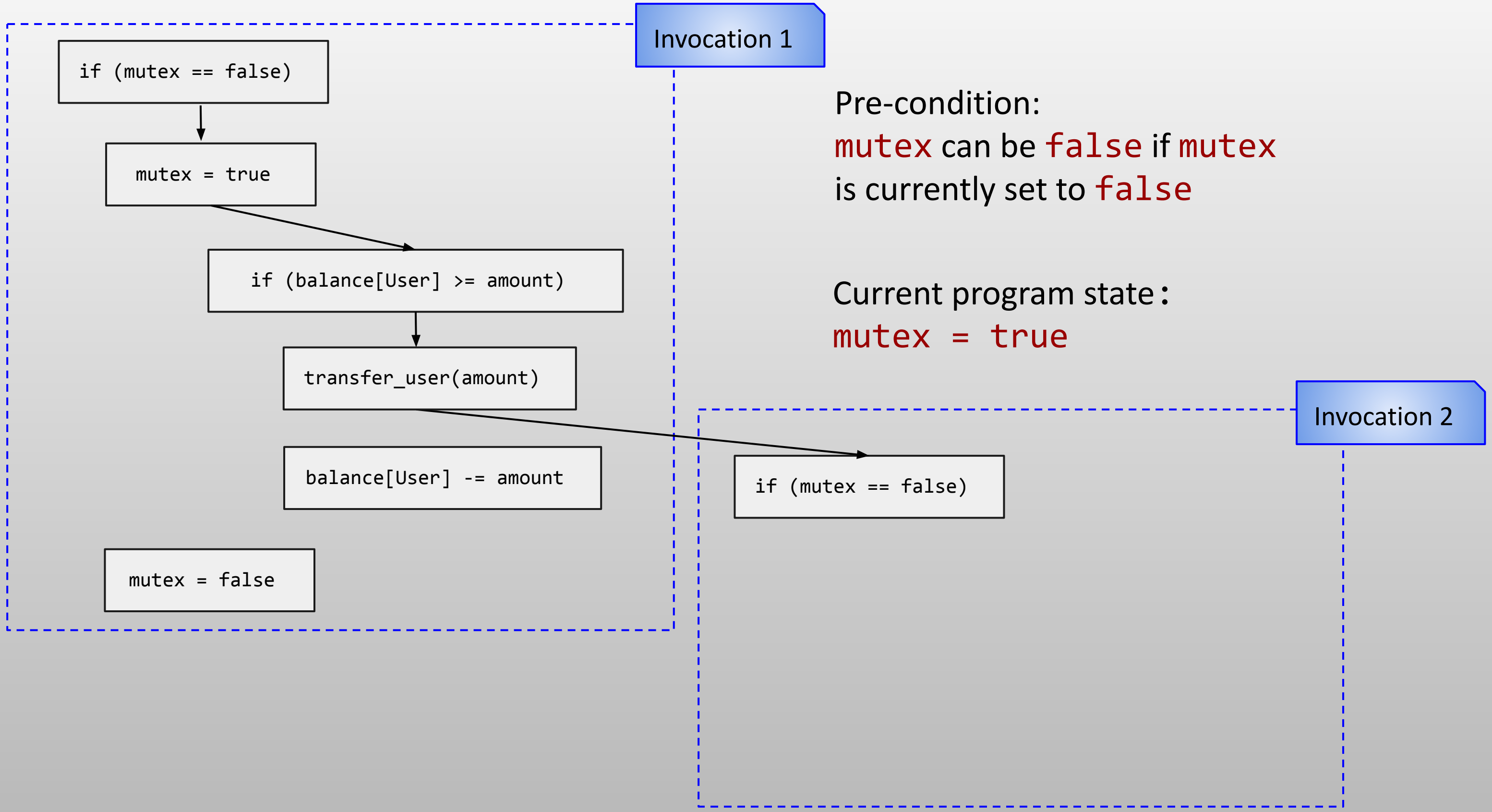


Refiner: Value-summary analysis + Symbolic execution

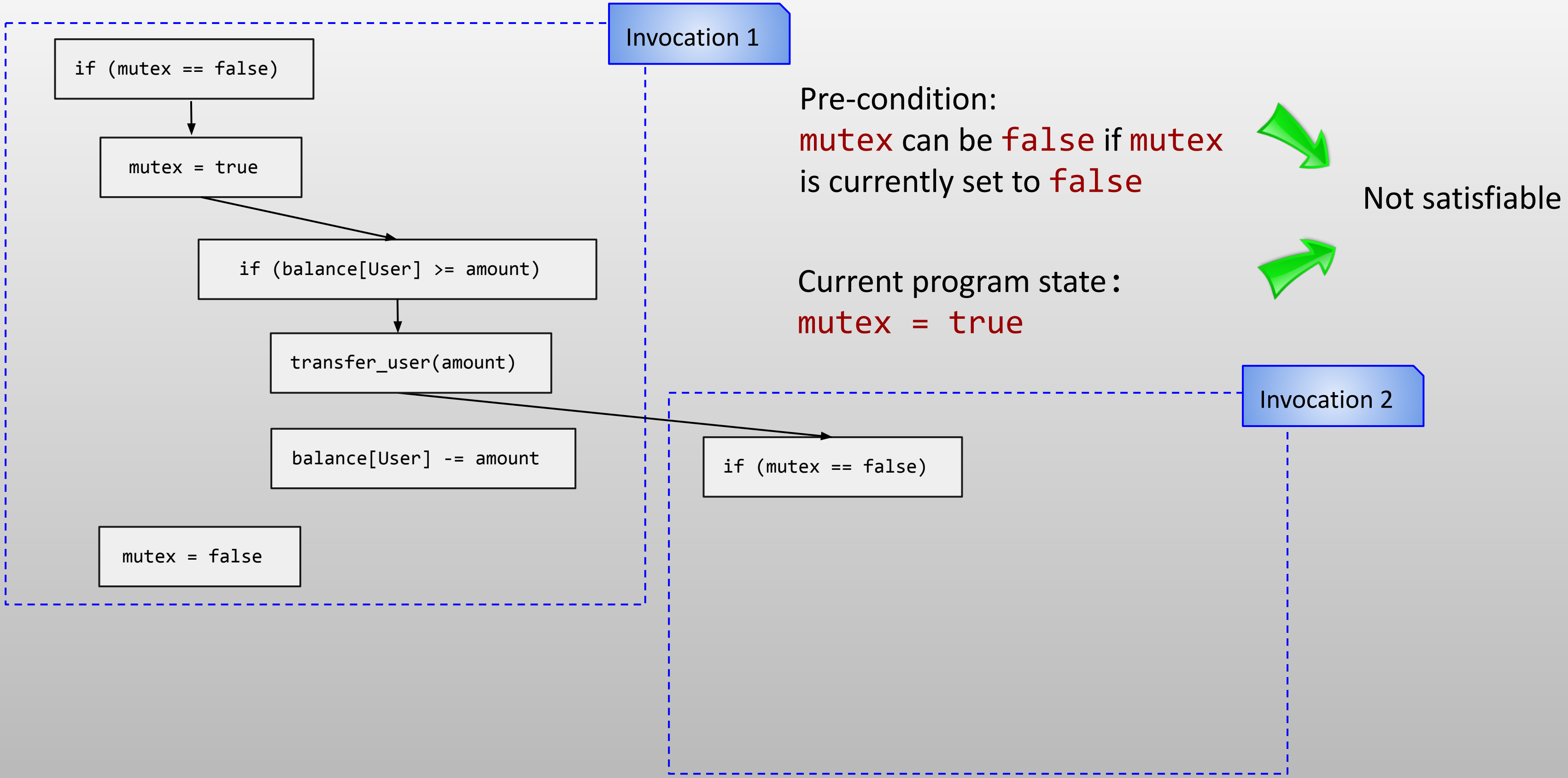
SECLAB

Pre-condition:
`mutex` can be `false` if `mutex` is currently set to `false`

Current program state:
`mutex = true`

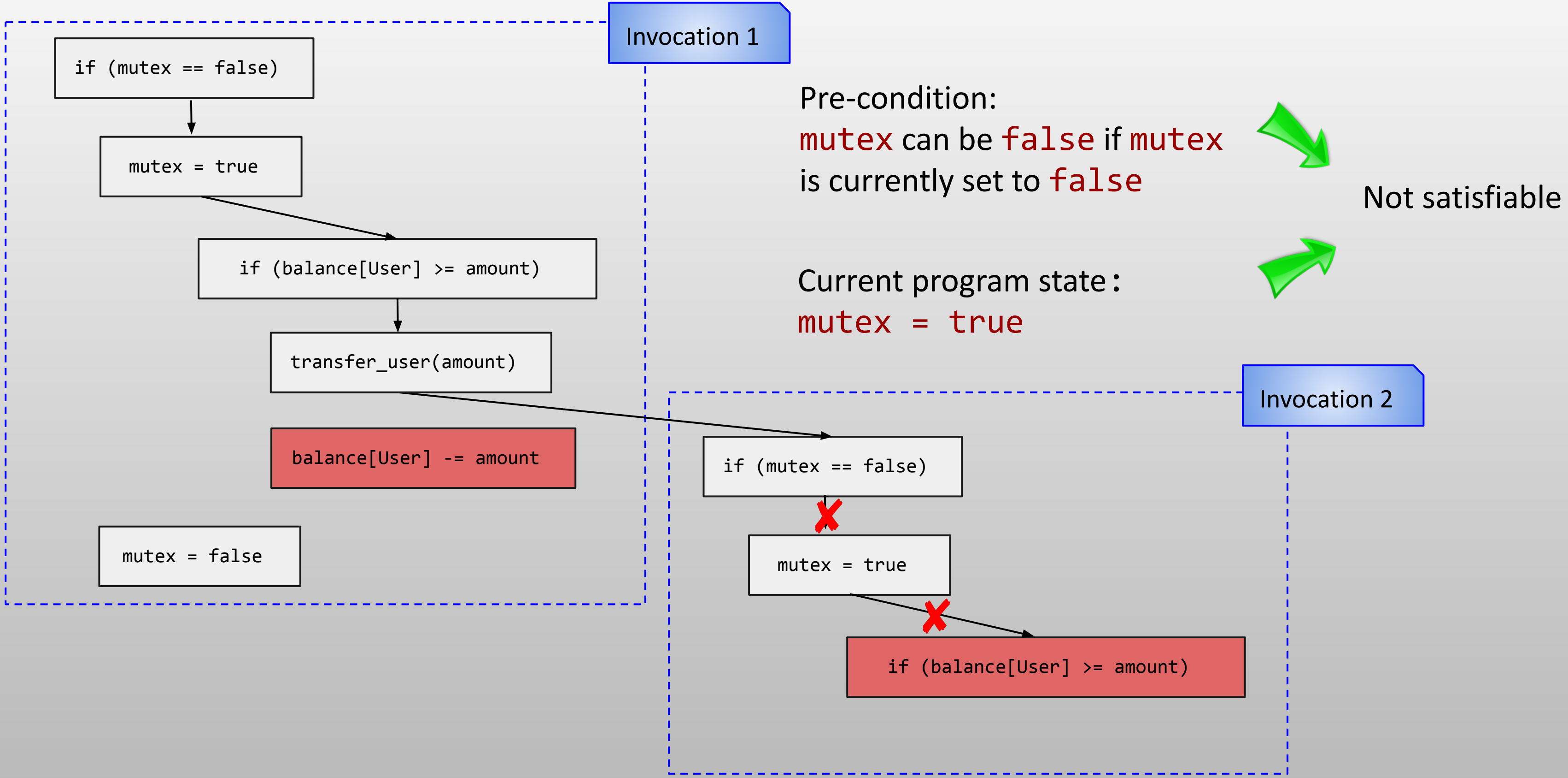


Refiner: Value-summary analysis + Symbolic execution



Refiner: Value-summary analysis + Symbolic execution

SECLAB



Results

Sailfish is evaluated against **Securify, Vandal, Oyente, Mythril**

+

The dataset consists of **89,853** Solidity open-sourced smart contracts

- ➔ Crawled from Etherscan
- ➔ All contracts till October, 2020
- ➔ Deduplicated

➔

(i) Vulnerability detection

(ii) Performance

Vulnerability detection

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Vulnerability detection

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Mythril timed out for more than 50% of the contracts

Oyente errored out for around 60% of the contracts

Vulnerability detection

SECLAB

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Manually verified randomly-chosen 750 contracts from the our dataset

- ➔ [0, 3000] lines of code
- ➔ No tool errored out/timed out
- ➔ Found 26 reentrancy bugs

Vulnerability detection

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Manually verified randomly-chosen 750 contracts from the our dataset

- ➔ [0, 3000] lines of code
- ➔ No tool errored out/timed out
- ➔ Found 26 reentrancy bugs

Reentrancy

Tool	True positive	False positive	False negative
Securify	9 (35%)	163 (22%)	17 (65%)
Vandal	26 (100%)	626 (86%)	0 (0%)
Mythril	7 (27%)	334 (46%)	19 (73%)
Oyente	8 (31%)	16 (2%)	18 (69%)
Sailfish	26 (100%)	11 (1.5%)	0 (0%)

Vulnerability detection

SECLAB

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Manually verified randomly-chosen 750 contracts from the our dataset

- ➔ [0, 3000] lines of code
- ➔ No tool errored out/timed out
- ➔ Found 26 reentrancy bugs

Reentrancy

Tool	True positive	False positive	False negative
Securify	9 (35%)	163 (22%)	17 (65%)
Vandal	26 (100%)	626 (86%)	0 (0%)
Mythril	7 (27%)	334 (46%)	19 (73%)
Oyente	8 (31%)	16 (2%)	18 (69%)
Sailfish	26 (100%)	11 (1.5%)	0 (0%)

Vulnerability detection

SECLAB

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

Manually verified randomly-chosen 750 contracts from the our dataset

- ➔ [0, 3000] lines of code
- ➔ No tool errored out/timed out
- ➔ Found 26 reentrancy bugs

Reentrancy

Tool	True positive	False positive	False negative
Securify	9 (35%)	163 (22%)	17 (65%)
Vandal	26 (100%)	626 (86%)	0 (0%)
Mythril	7 (27%)	334 (46%)	19 (73%)
Oyente	8 (31%)	16 (2%)	18 (69%)
Sailfish	26 (100%)	11 (1.5%)	0 (0%)

- ➢ Least false positives
- ➢ No false negatives
- ➢ Finds all true bugs

Vulnerability detection

Tool	Reentrancy	TOD
Securify	6,321	19,031
Vandal	45,971	-
Mythril	3,708	-
Oyente	269	3,472
Sailfish	2,076	7,555

- Sailfish found **47** bugs not found by any other tool

Manually verified randomly-chosen 750 contracts from the our dataset

➔ [0, 3000] lines of code

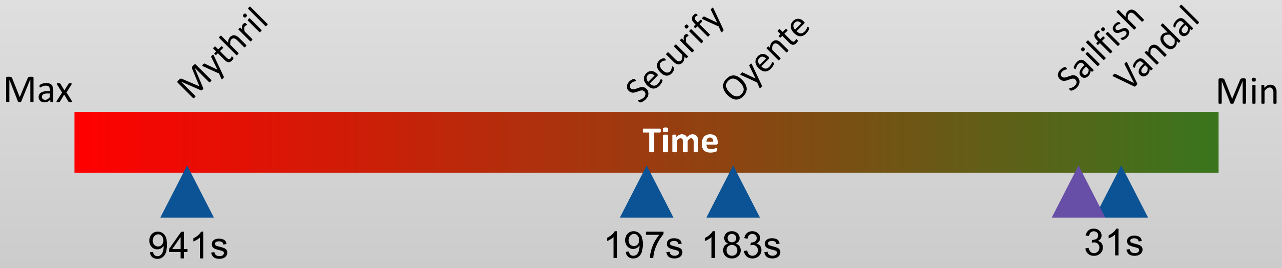
➔ No tool errored out/timed out

➔ Found 26 reentrancy bugs

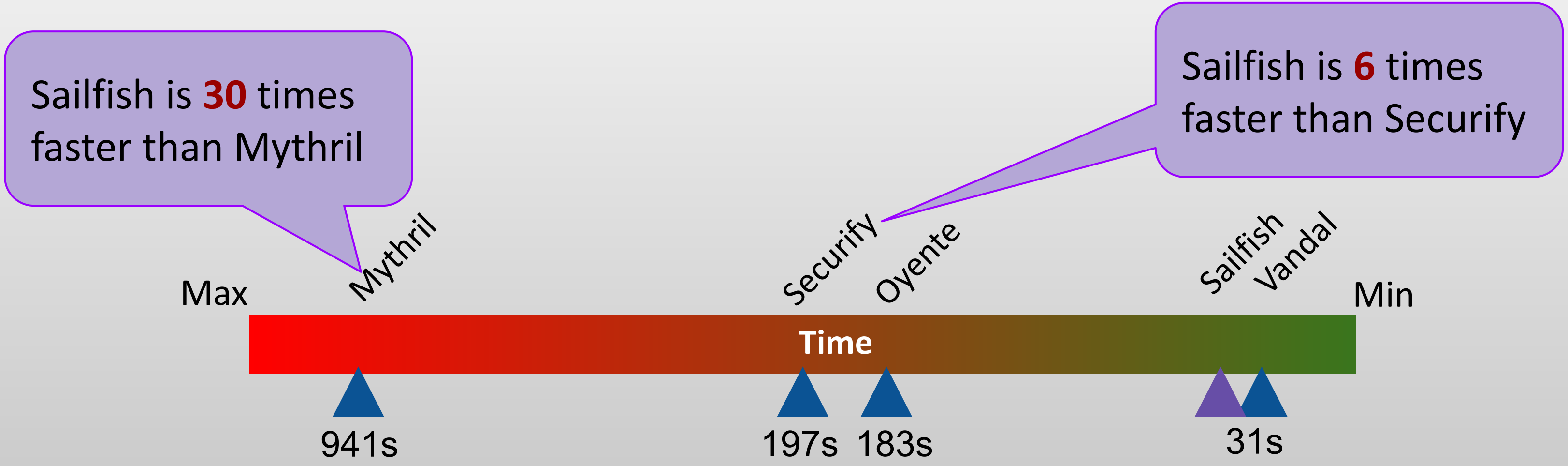
Reentrancy

Tool	True positive	False positive	False negative
Securify	9 (35%)	163 (22%)	17 (65%)
Vandal	26 (100%)	626 (86%)	0 (0%)
Mythril	7 (27%)	334 (46%)	19 (73%)
Oyente	8 (31%)	16 (2%)	18 (69%)
Sailfish	26 (100%)	11 (1.5%)	0 (0%)

Performance



Performance



Conclusion

- We presented Sailfish, a bug finding tool for reentrancy and TOD

Conclusion

- We presented Sailfish, a bug finding tool for reentrancy and TOD
- Sailfish generalized reentrancy and TOD in terms of State Inconsistency

Conclusion

- We presented Sailfish, a bug finding tool for reentrancy and TOD
- Sailfish generalized reentrancy and TOD in terms of State Inconsistency
- Sailfish combines static analysis and VSA-enabled symbolic execution to achieve both precision and scalability

Conclusion

SECLAB

- We presented Sailfish, a bug finding tool for reentrancy and TOD
- Sailfish generalized reentrancy and TOD in terms of State Inconsistency
- Sailfish combines static analysis and VSA-enabled symbolic execution to achieve both precision and scalability
- Sailfish outperforms state-of-the-art techniques in both performance and bug finding ability

THANKS!

@cinderella0x80

priyanka@cs.ucsb.edu

https://github.com/ucsb-seclab/sailfish