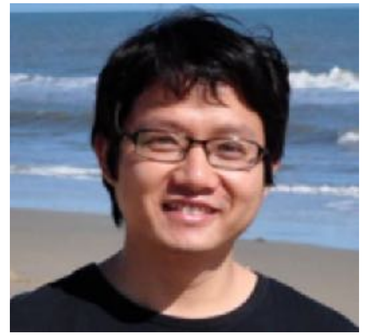
Secure ZK Circuits with Formal Methods

Guest Lecturer: **Yu Feng (UCSB & Veridise)**



Zero Knowledge Proofs

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang

Stanford University

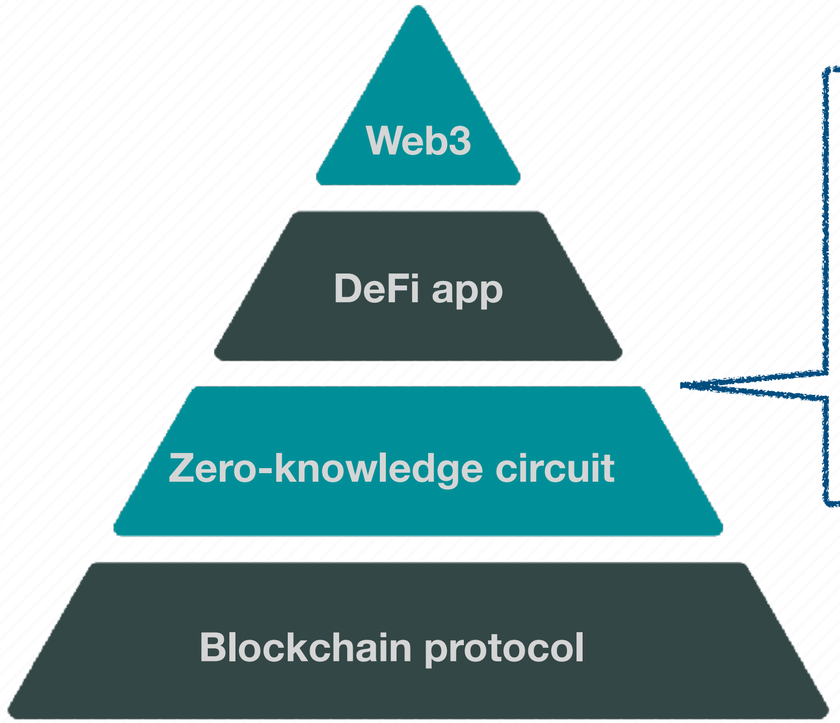
Berkeley UNIVERSITY OF CALIFORNIA

GEORGETOWN UNIVERSITY

TEXAS A&M UNIVERSITY

Motivation

Bugs in blockchain software are **extremely** costly



Bugs in any of these layers can be catastrophic when exploited!

Smart Contract Bugs

Ethereum DeFi Protocol Beanstalk Hacked for $182 Million—What You Need to Know

Beanstalk got jacked by a giant flash attack.

By Jeff Benson

Apr 18, 2022

2 min read

Beanstalk. Image: Shutterstock

Flash loan vulnerability in smart contract

Blockchain Protocol Bugs

CRYPTO WORLD

Solana suffered its second outage in a month, sending price plunging

PUBLISHED WED, JUN 1 2022·9:27 PM EDT

MacKenzie Sigalos
@KENZIESIGALOS

SHARE

KEY POINTS

- Solana fell more than 12% on Wednesday as the blockchain suffered its second outage in the last month.
- Investors who had been focused largely on ethereum began diversifying into Solana and other alternative blockchains during last year's crypto run-up.
- But the last year and a half has laid bare the trade-off as the blockchain network has suffered multiple outages.

The logo of cryptocurrency platform Solana.
Jakub Porzycki | NurPhoto via | Getty Images

RELATED


Crypto hedge fund Three Arrows Capital plunges into liquidation as market crash takes toll


One of the most prominent crypto hedge funds just defaulted on a $670 million loan


Snoop Dogg on the current crypto winter and future of NFTs

El Salvador's $425 million bitcoin experiment isn't saving the country's finances

DoS vulnerability in consensus protocol

ZK Bugs are Coming

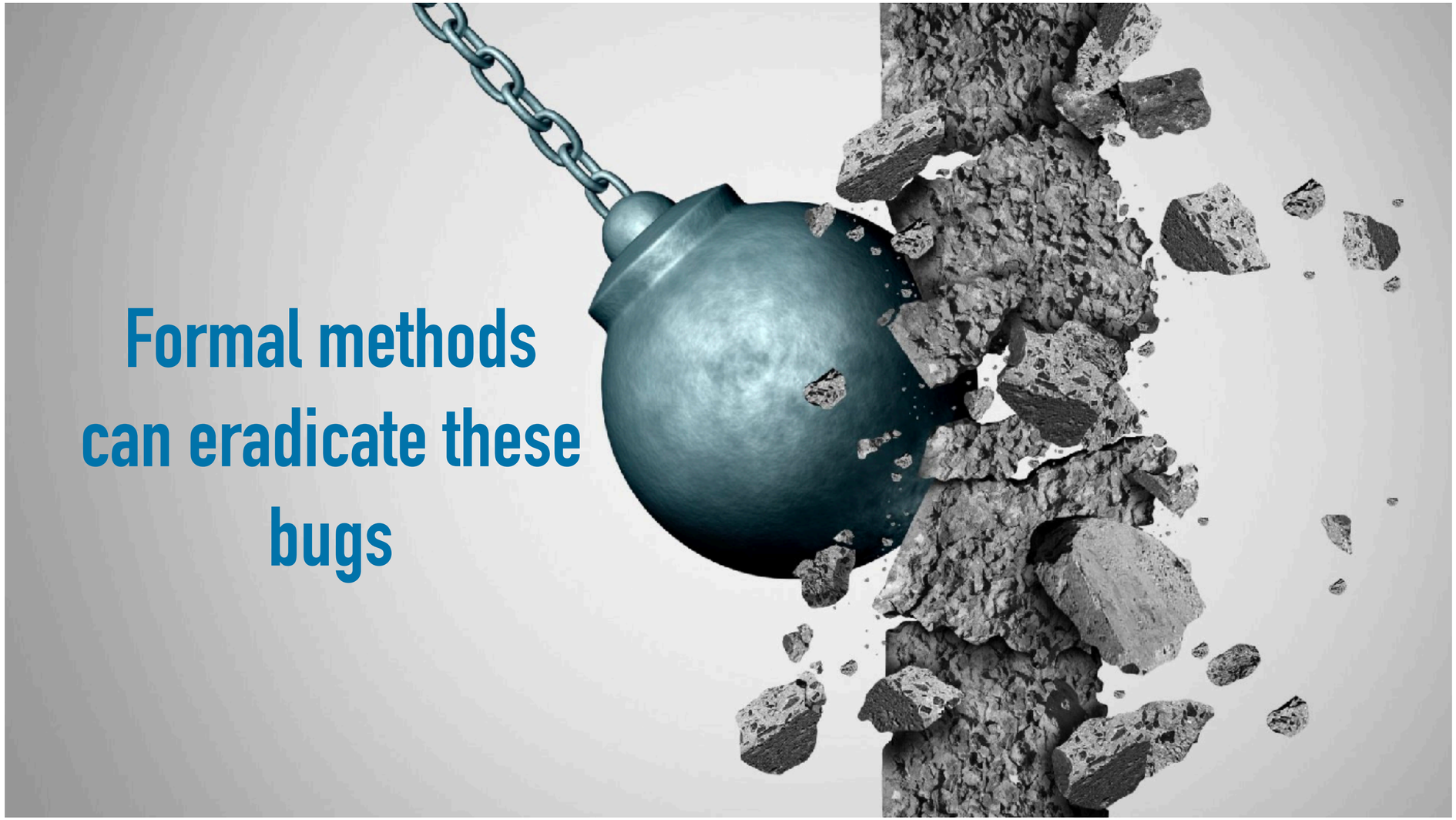
Zcash team fixes serious vulnerability that allowed counterfeiting

Malware and Vulnerabilities • February 07, 2019 • Cyware Hacker News

- The vulnerability was discovered by a cryptographer from Zcash Company in March 2018.
- Attackers could create fake Zcash coins in large numbers by exploiting this vulnerability.

Bug in arithmetic circuit implementing zkSNARK!

Formal Methods to Rescue

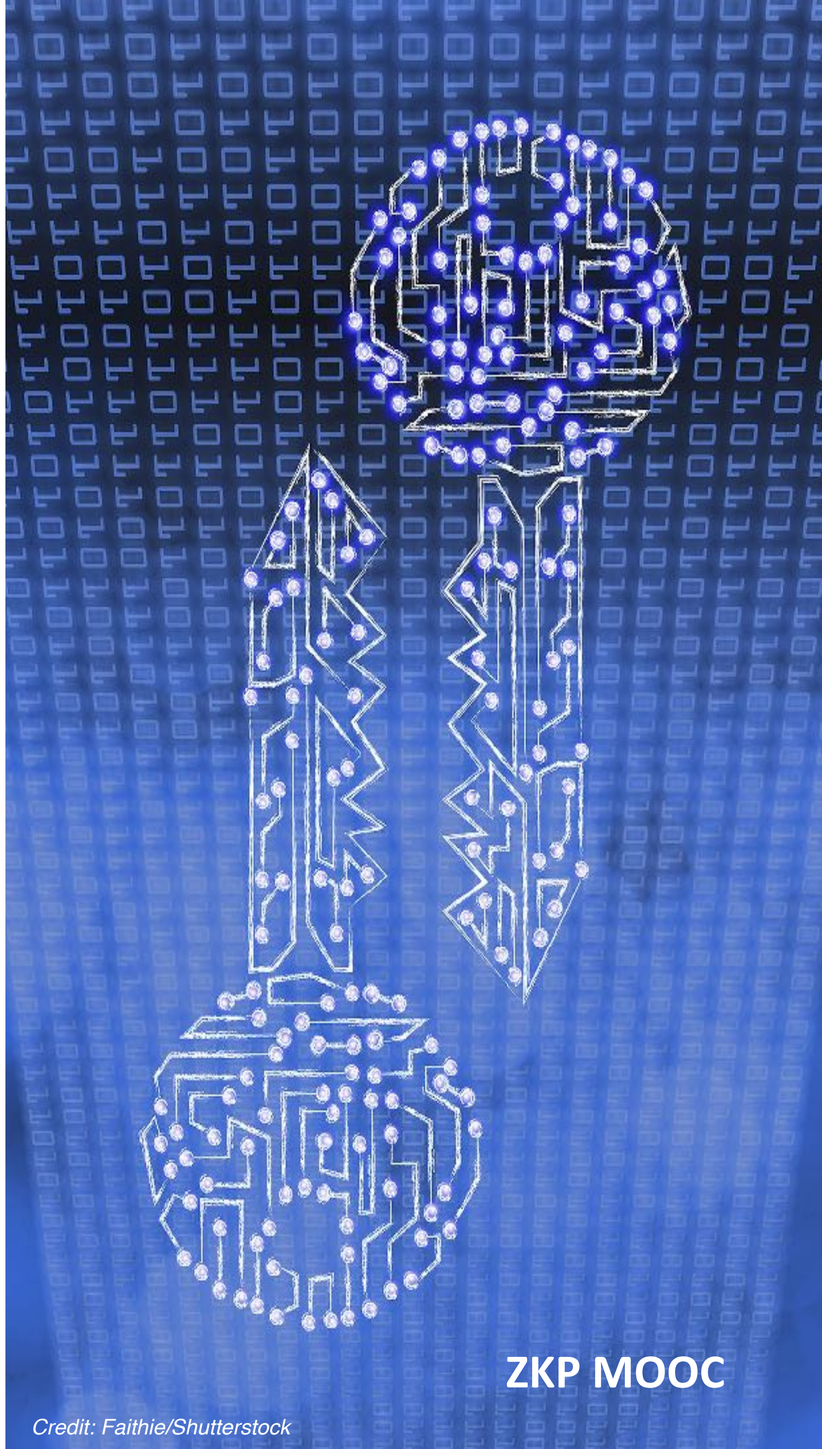


Outline

- Formal methods in a nutshell
- Formal methods for ZK I (static analysis)
- Formal methods for ZK II (SMT solver)
- Future work & Conclusion

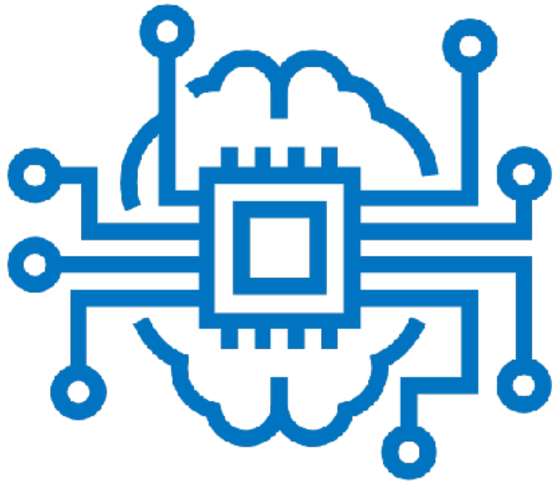
Section 1

Formal Methods in a Nutshell



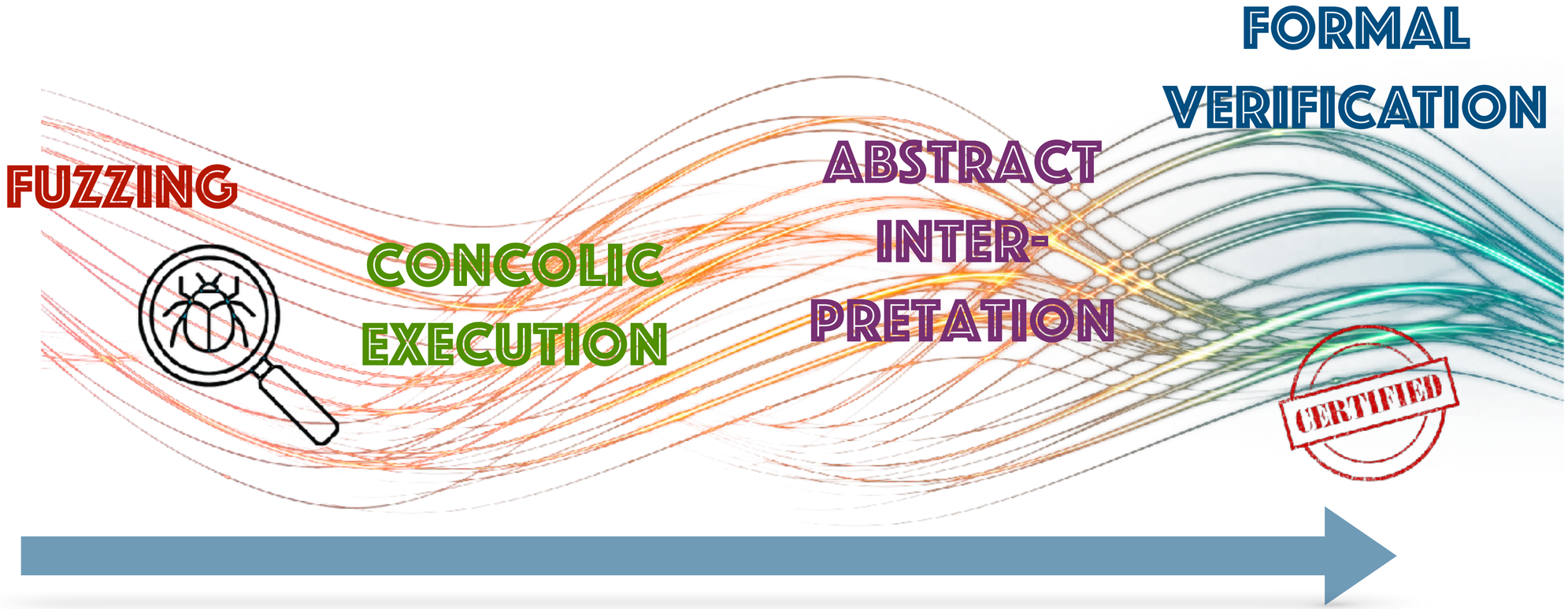
Credit: Faithie/Shutterstock

What is Formal Methods



Set of mathematically rigorous techniques for finding bugs and constructing proofs about software

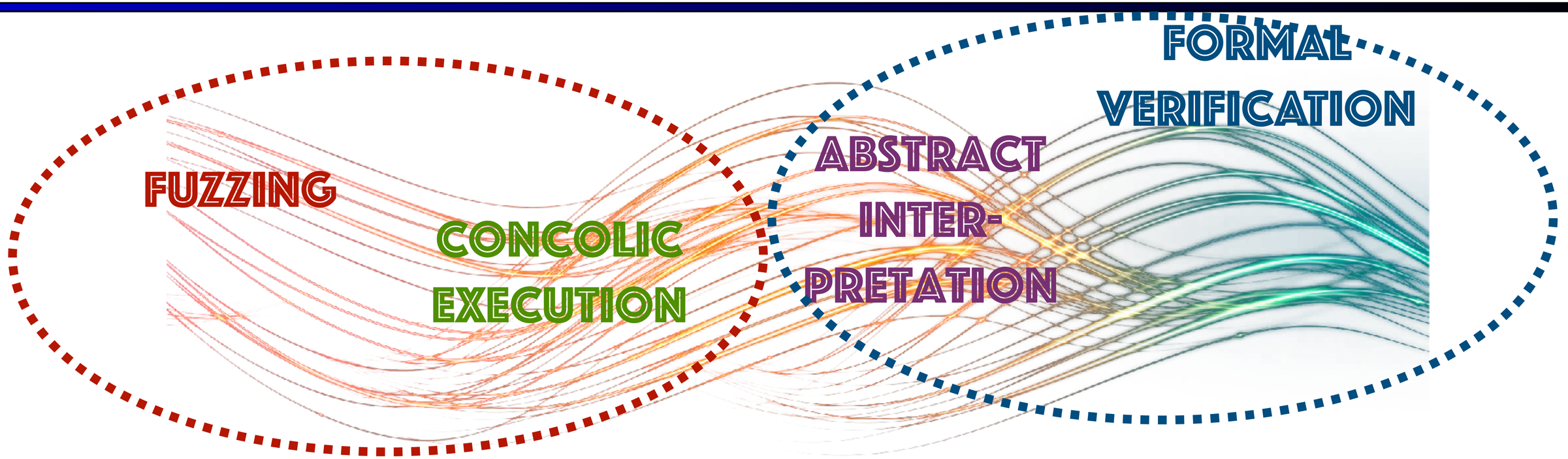
Formal Methods Techniques on Spectrum



Stronger guarantees

More human effort

Classification of FM Techniques



DYNAMIC

Execute the program on interesting inputs & monitor what happens

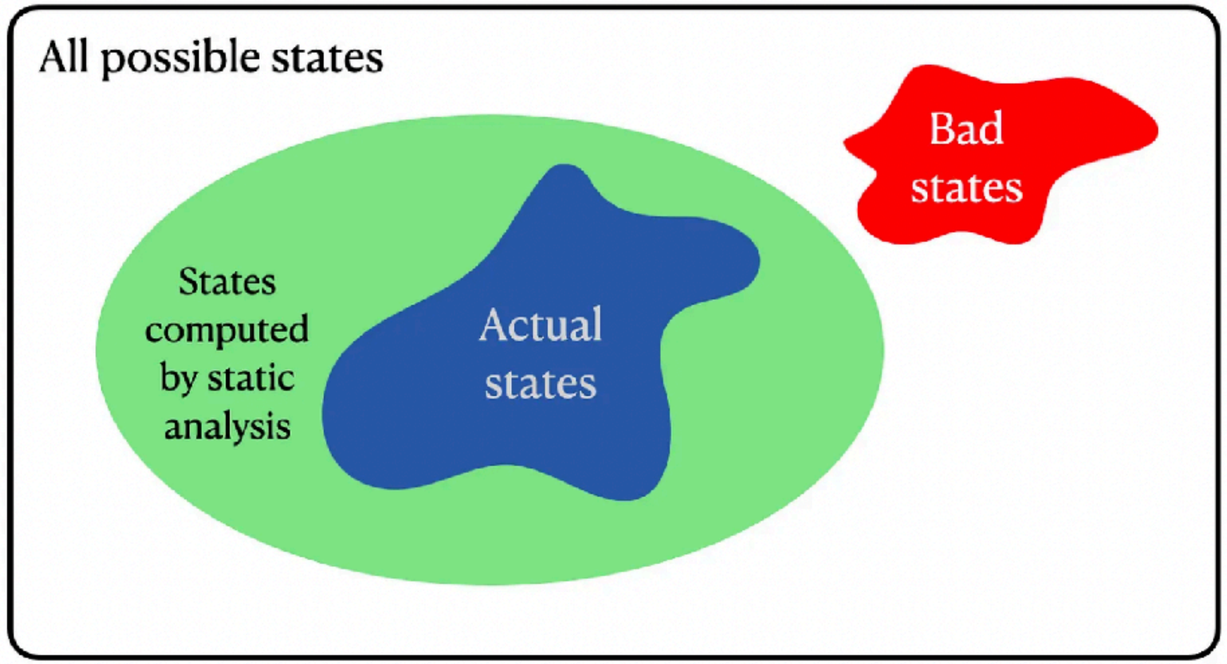
STATIC

Analyze source code and reason about all executions

Static Analysis via Abstract Interpretation

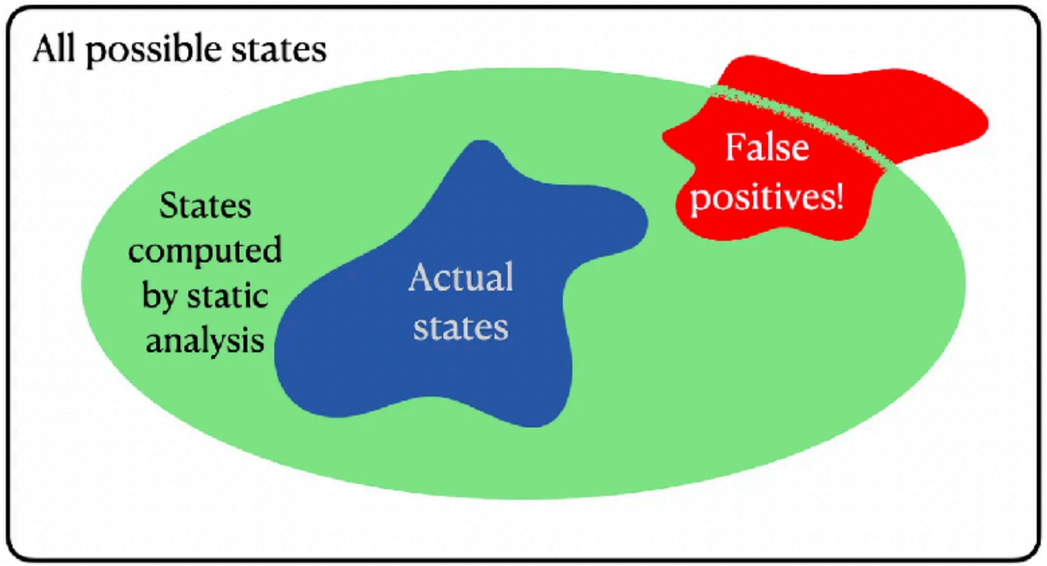
- Cannot reason about the exact program behavior due to undecidability
- Obtain a conservative over-approximation and this can be enough to prove program correctness
- Abstract interpretation is a framework for computing over-approximations of program states

Static Analysis via Abstract Interpretation

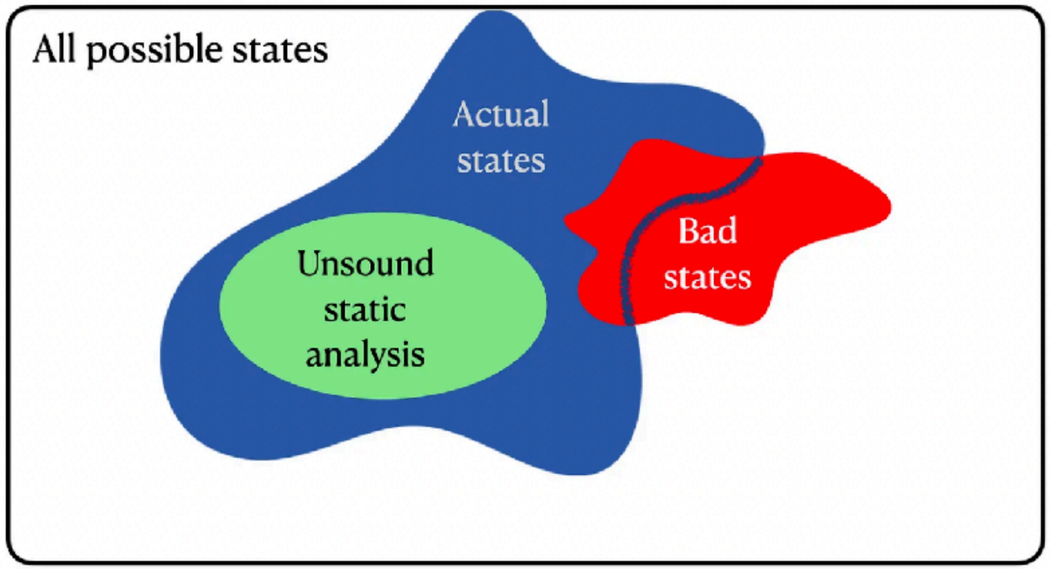


Program is safe: intersection between the green and red region is empty

Static Analysis via Abstract Interpretation



False Positives



False Negatives

Concrete Interpretation

- Concrete interpreter operates over **concrete values**
- x=y+z. x evaluates to 3 if y=1 and z=2

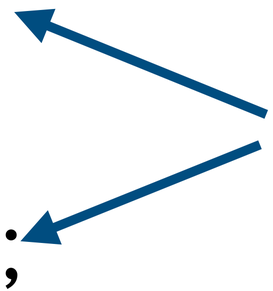
Static Analysis via Abstract Interpretation

- Abstract interpreter operates over **abstract values**
- Integer x can have an abstract value of interval [a,b]
- x=y+z $y \in [a,b] \wedge z \in [c,d] \implies x \in [a+c, b+d]$

Static Analysis via Abstract Interpretation

<u>Idea:</u> Emulate all possible program paths

```
if(flag)
  x = 1;
else
  x = -1;
```



When in doubt, conservatively assume either path could be taken and merge information for different paths

$$x \in [-1,1]$$

Detect Reentrancy via Abstract Interpretation

```
1  contract Attacker {
2    Victim v;
3
4    function exploit() {
5      v = Victim(0x123);
6      v.withdraw(10);
7    }
8
9    function() payable {
10     v.withdraw(10);
11   }
12 }
```

(2) attack program

```
1 contract Victim {
2   ...
3   function withdraw(uint a) {
4     msg.sender.call.value(a);
5     balances[msg.sender] -= a;
6 }
```

(3) victim program

External function call followed by a storage update

Other Vulnerabilities & Tools

- Integer overflow
- Transaction order dependency
- Flashloan attack
- Related tools:
  - Slither (TrailOfBits)
  - Vanguard (Veridise)
  - Sailfish (Oakland’22), Securify (CCS’19)

From Abstract Interpretation to Formal Verification

- Identify **specific** types of bugs & security vulnerabilities
- Can’t guarantee program is free from **logical errors**
- Formal verification to rescue
- Need a **specification** to describe how the program is supposed to behave

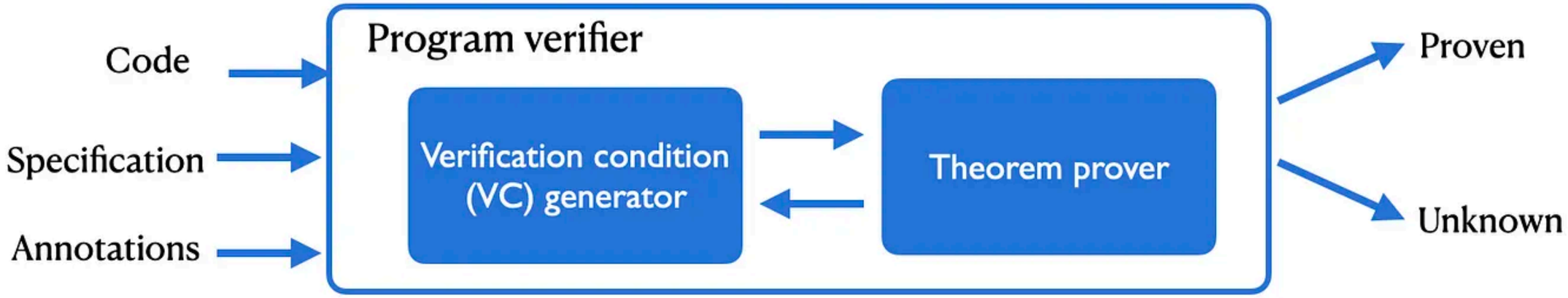
Formal Specifications

<u>Formal specification:</u> Precise mathematical description of intended program behavior, typically in some formal logic

$$\square\ ((finish(bid, msg\,.\,value = X \wedge msg\,.\,sender = L))$$
$$\wedge \Diamond\ finish(close, L \neq winner)$$
$$\rightarrow \Diamond\ send(to = L \wedge amt = X))$$

If auction closes with me not being the winner, I should eventually get back my bid

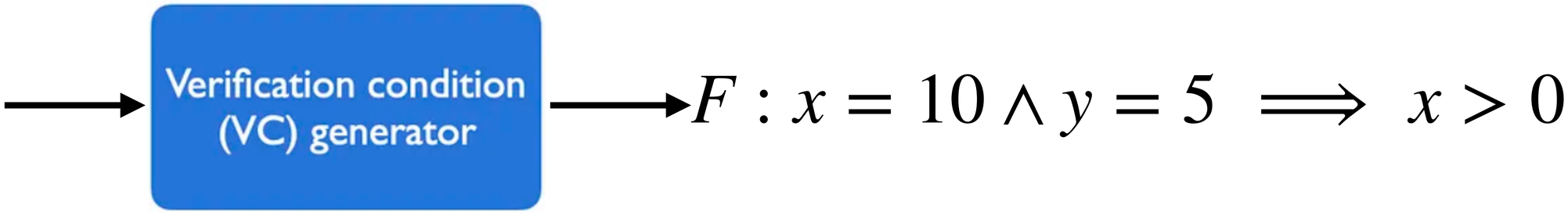
Overview of Formal Verification



- Code: Source code or byte code
- Specification: A formal description of the property to be verified
- Human annotations (optional): Loop invariants, Contract invariants

VC Generation

```
foo () {
 x = 10;
 y = 5;
 assert x>0;
}
```



Assertion always hold iff F is valid!

When Formal Verification Fails…

- Logical error (true positive)
- Insufficient human input, loop iterations, missing lemmas
- Incompleteness of theorem prover
  - Linear integer arithmetic ✓
  - Finite fields over large prime, non-linear arithmetic ✗

Bounded vs Unbounded Verification

- Unbounded verification so far
- Bounded verification with weaker guarantees
  - Restrict input size/space
  - Bounded loop iterations
- Tools: Certora prover, K-framework, Mythril, Sailfish, etc.

Different Flavors of Static Analysis

Formal verification checks program against provided specification

Abstract Interpretation

- Looks for known types of bugs
- Doesn't require specifications

Formal Verification

- Can find (prove absence of) any bug
- Requires specifications