CityU
香港城市大學
City University of Hong Kong

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

EPFL

Everything is Good for Something: Counterexample-Guided Directed Fuzzing

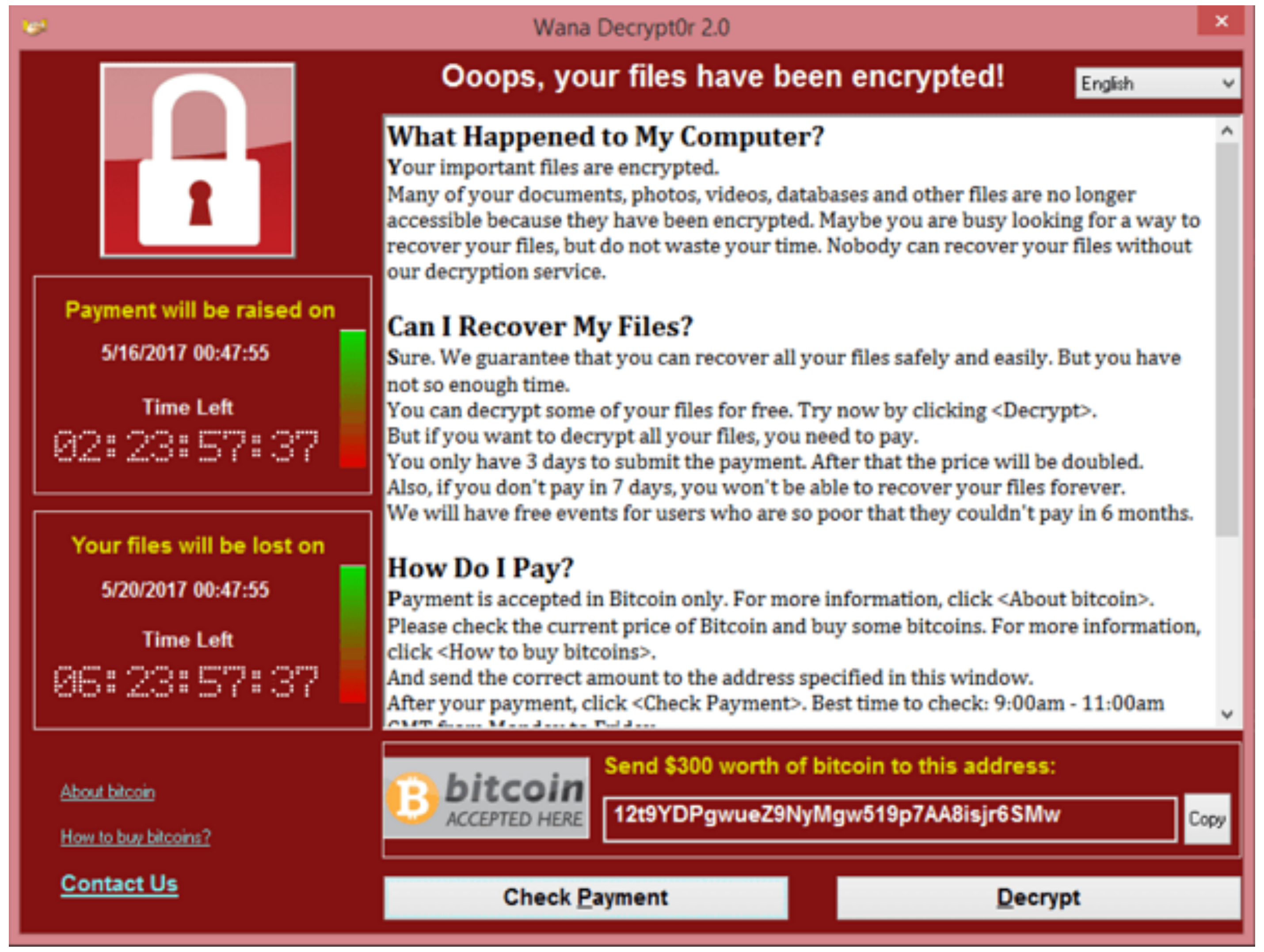
Heqing Huang, Anshunkang Zhou, Mathias Payer, Charles Zhang

{heqhuang}@cityu.edu.hk, {azhouad, charlesz}@cse.ust.hk, {mathias.payer}@nebelwelt.net

45th IEEE Symposium on Security and Privacy

May, 21st, 2024

Software bugs are prevalent and can cause severe consequences



Ooops, your files have been encrypted!

Payment will be raised on
5/16/2017 00:47:55
Time Left
02:23:57:37

Your files will be lost on
5/20/2017 00:47:55
Time Left
06:23:57:37

What Happened to My Computer?

Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?

Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?

Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

bitcoin ACCEPTED HERE

Send $300 worth of bitcoin to this address:

12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

Electronics

Killer software: 4 lessons from the deadly 737 MAX crashes

by Matt Hamblen | Mar 2, 2020 1:23pm

The rocket exploded seconds after launching

1.7 TRILLION: FINANCIAL LOSSES CAUSED BY SOFTWARE FAILURES IN 2017

Published March 25 2020

Explosive software code size

Linux Kernel: 17 Million Lines

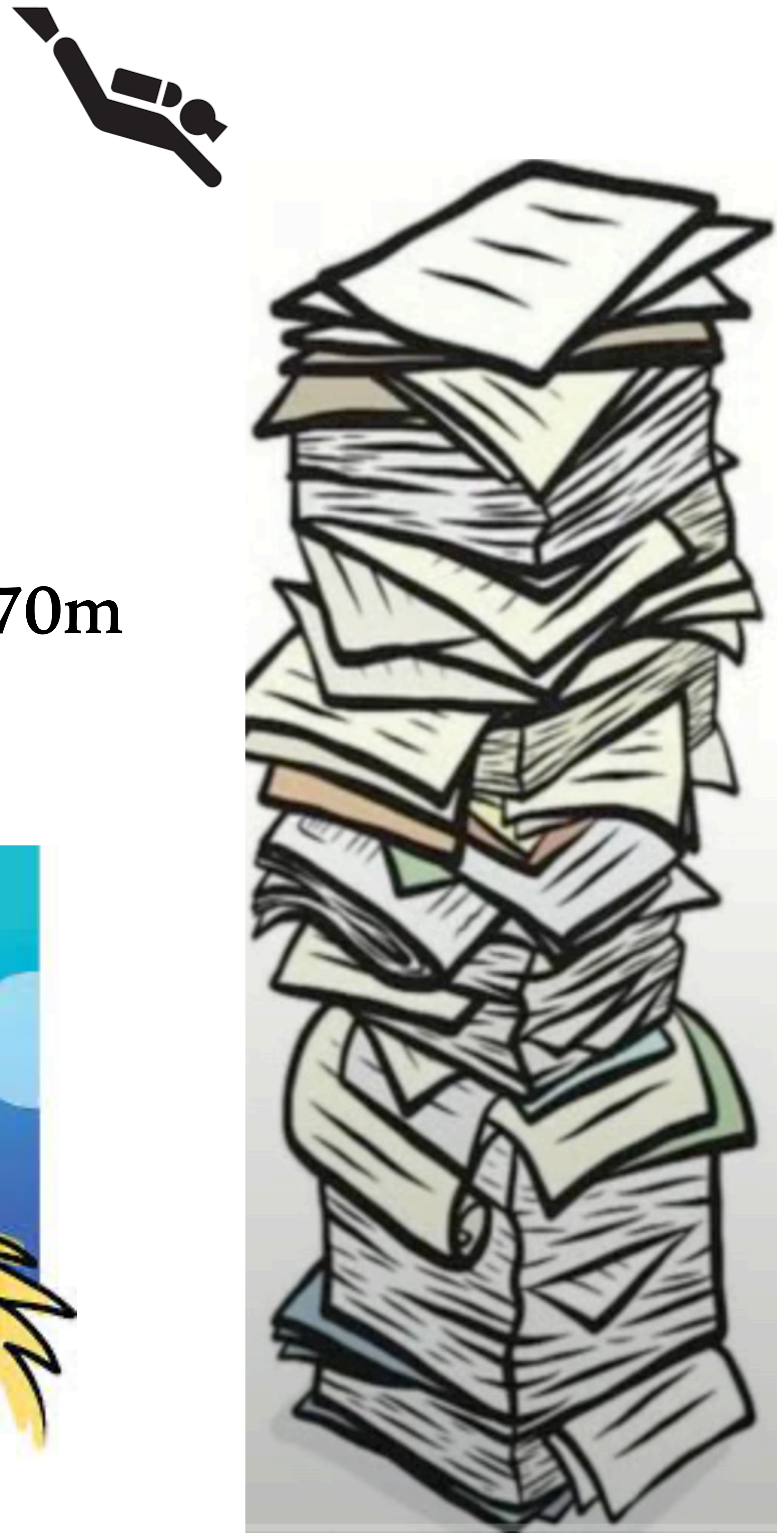
Google Chrome: 76 Million Lines

TESLA

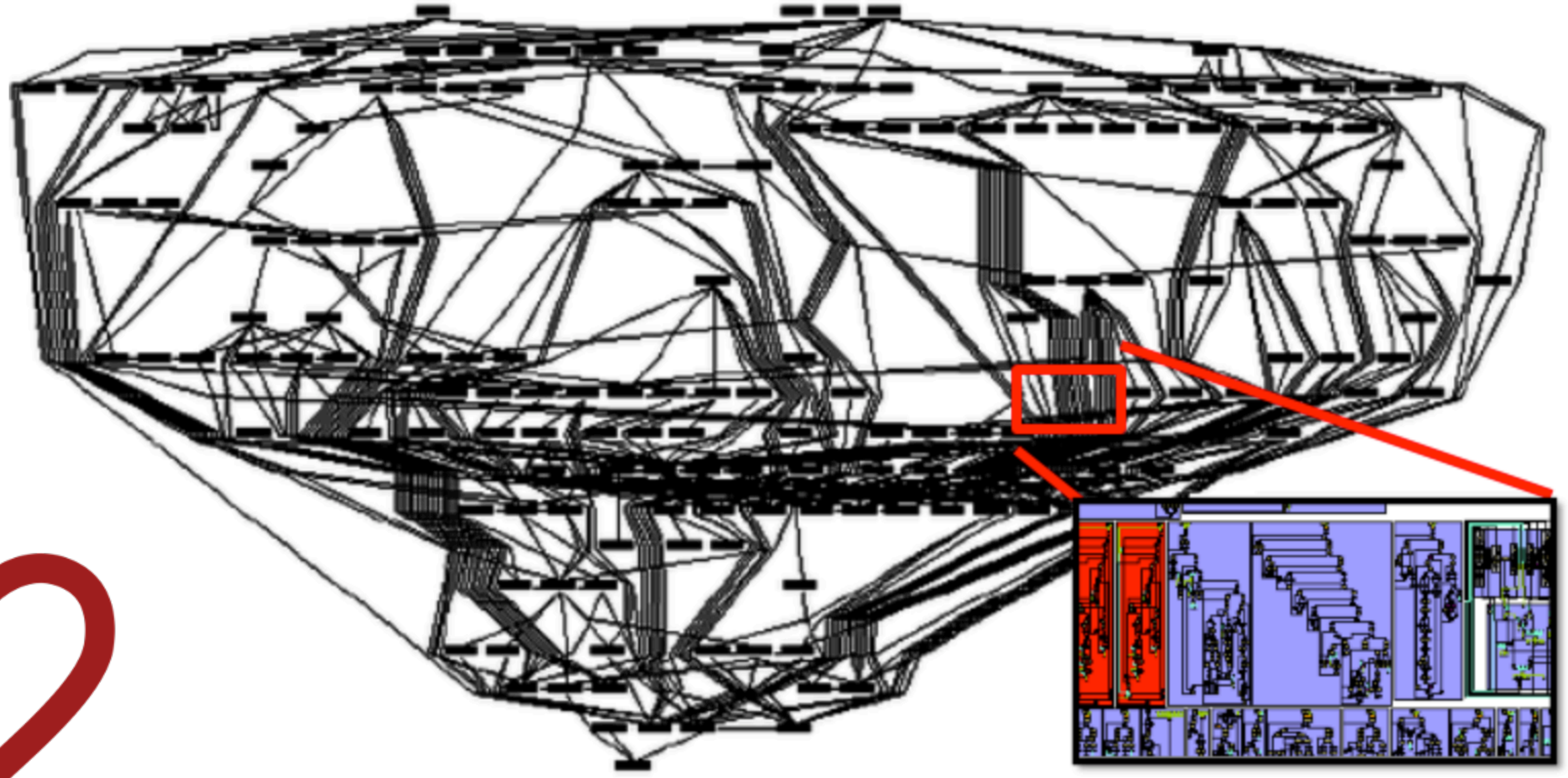
Autonomous vehicles: > 100 Million Lines

27 lines/page

0.01mm/page≈370m



Obstacles: Explosive paths and their complex conditions



7 files, over 13,000 lines

24348 files, over 1.5M lines

Generating inputs to satisfy the complex path condition is an NP-HARD problem!

Solution: Directed fuzzing!

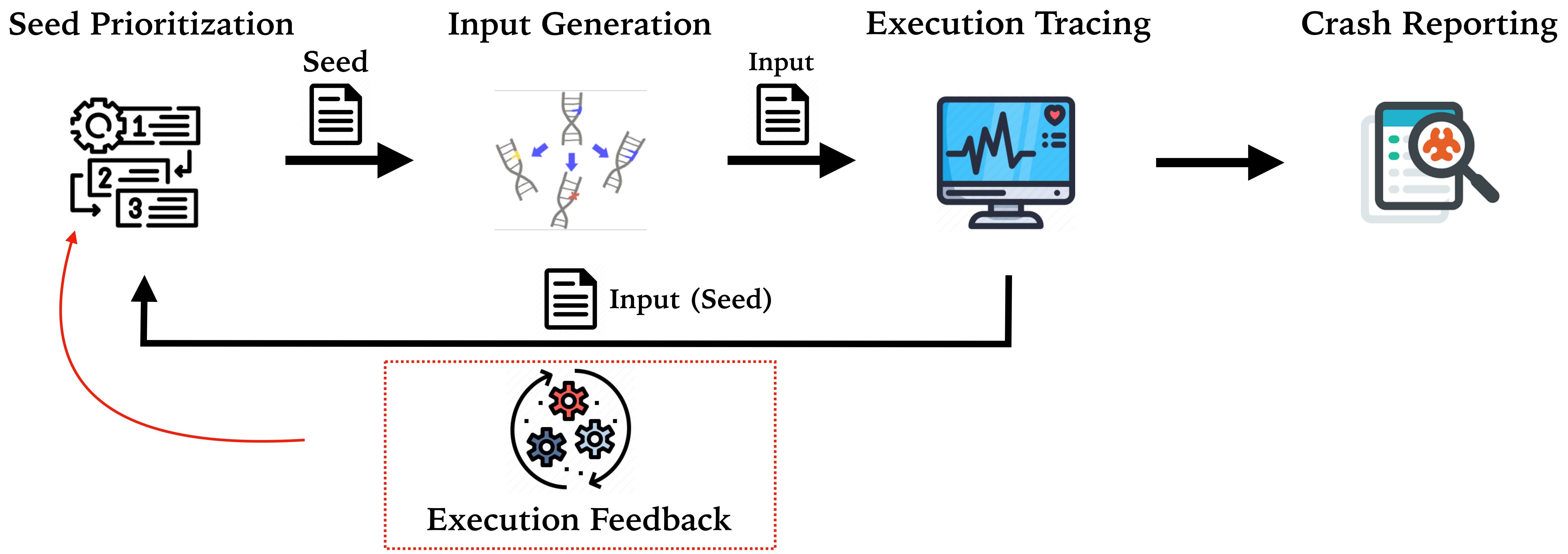
Patch Verification

1-day POC Generation

Debugging

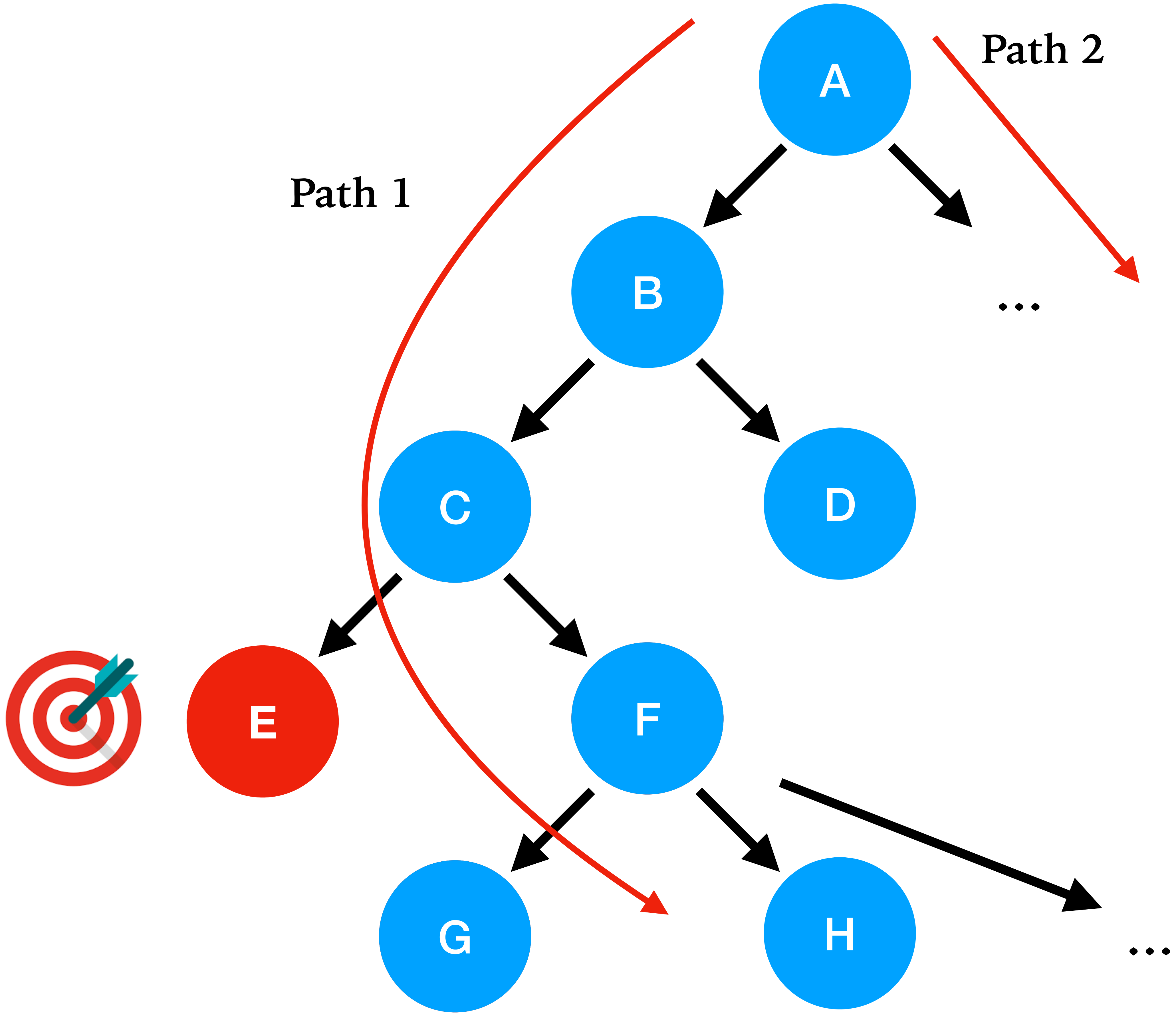
Aim to detect specific bugs automatically

Directed fuzzing in a nutshell



Directed fuzzers use additional execution feedback to adjust the priority of the preserved input uniquely for the target

Key Intuition: Prioritize paths “closer” to the target



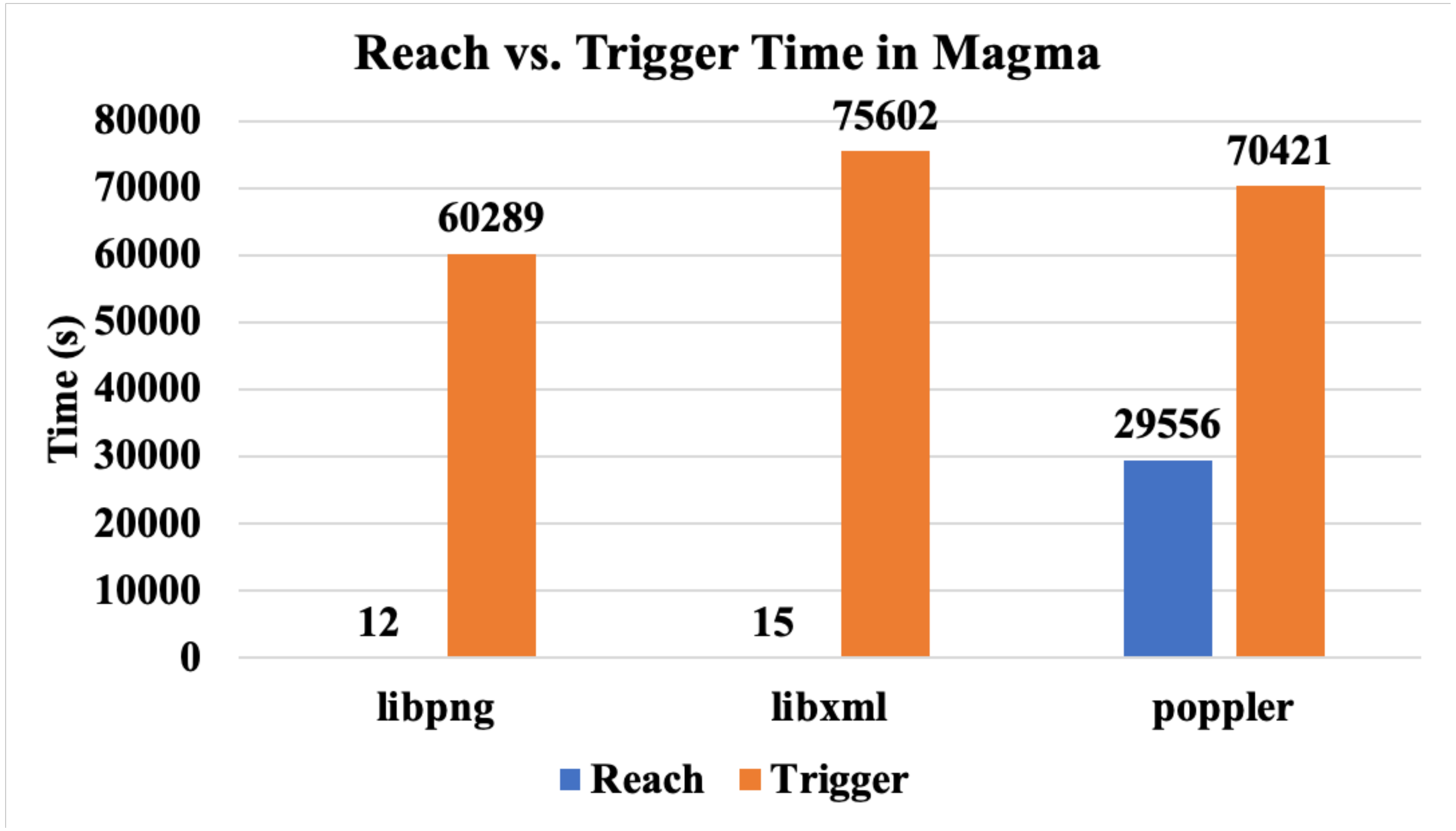
Closeness represents the possibility of reaching the target

Path 1 > Path 2

The majority of the inputs are still randomly generated

Problem observation: Deficient bug triggering

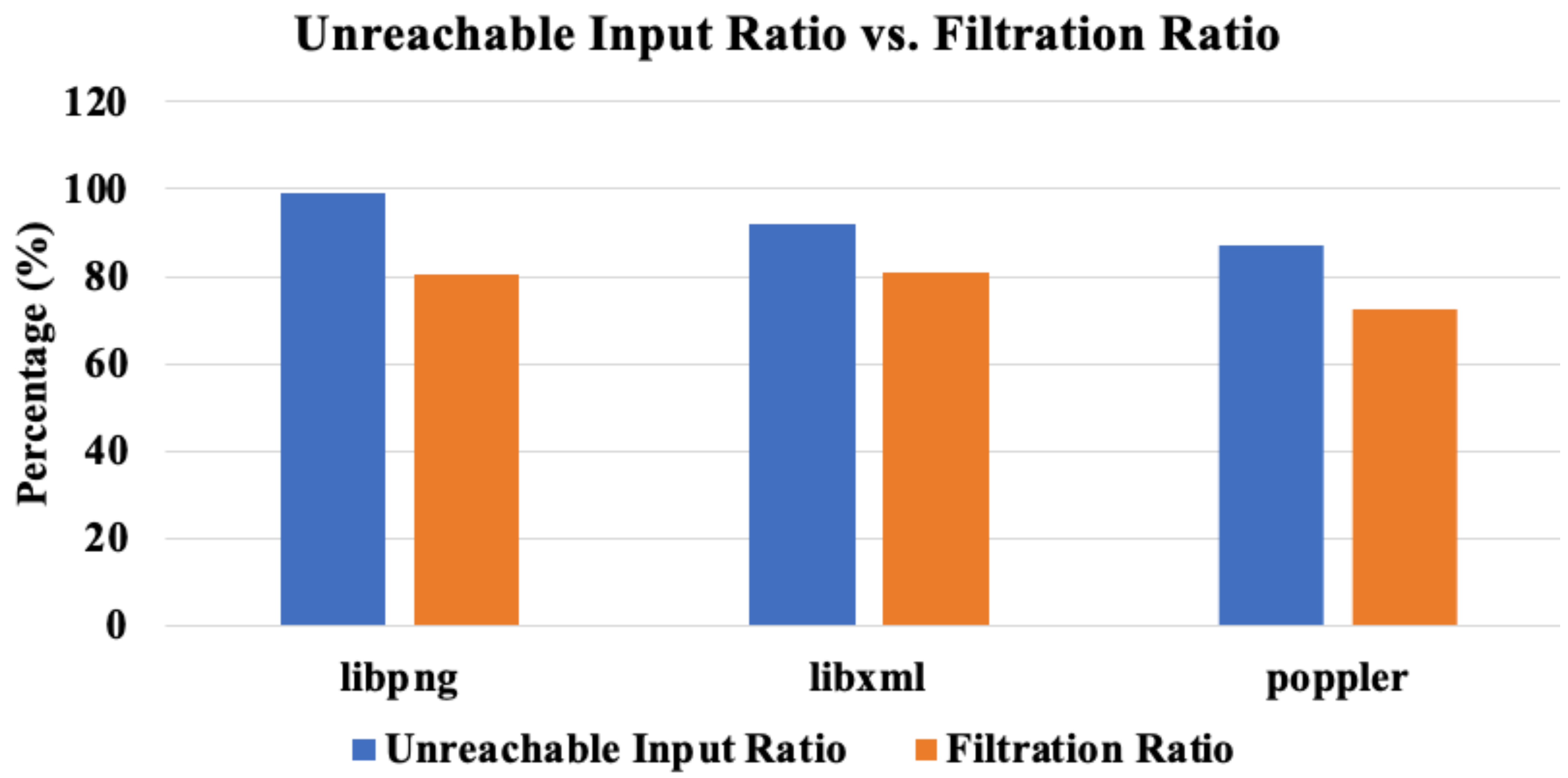
Directed fuzzing could spend much more time to trigger specific bugs after reaching it



24 hours experiments using AFLGo

Triggering the targets can be 1000 times longer than reaching it!

Root Cause: Majority of the generated inputs cannot even reach the bug



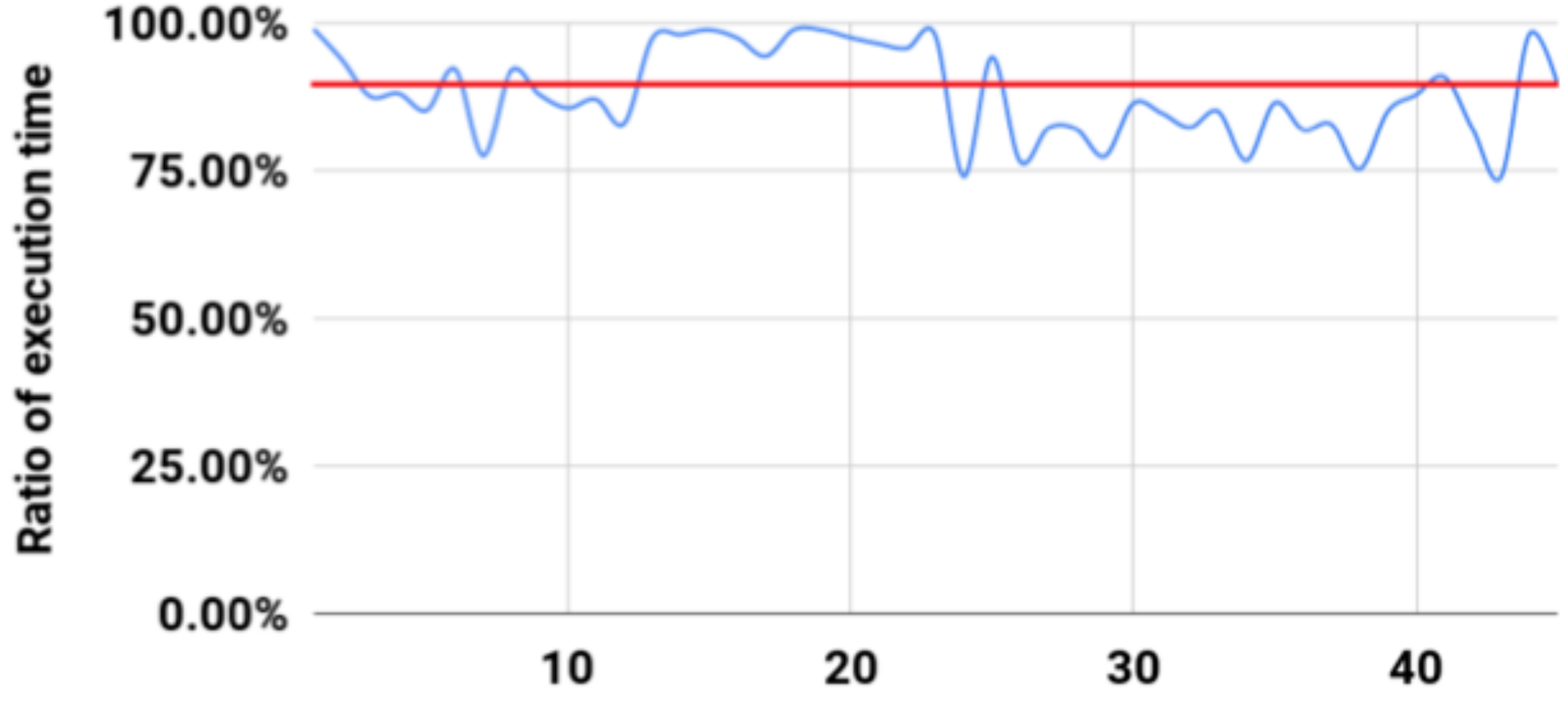
The unreachable input accounts for almost 100% of the generated inputs

The filtration techniques by early termination can prune 80% of the generated inputs

Root Cause: Majority of the generated inputs cannot even reach the bug

Majority of the times are wasted on executing infeasible inputs

24 hours experiment in 45 open-source bugs with AFLGO

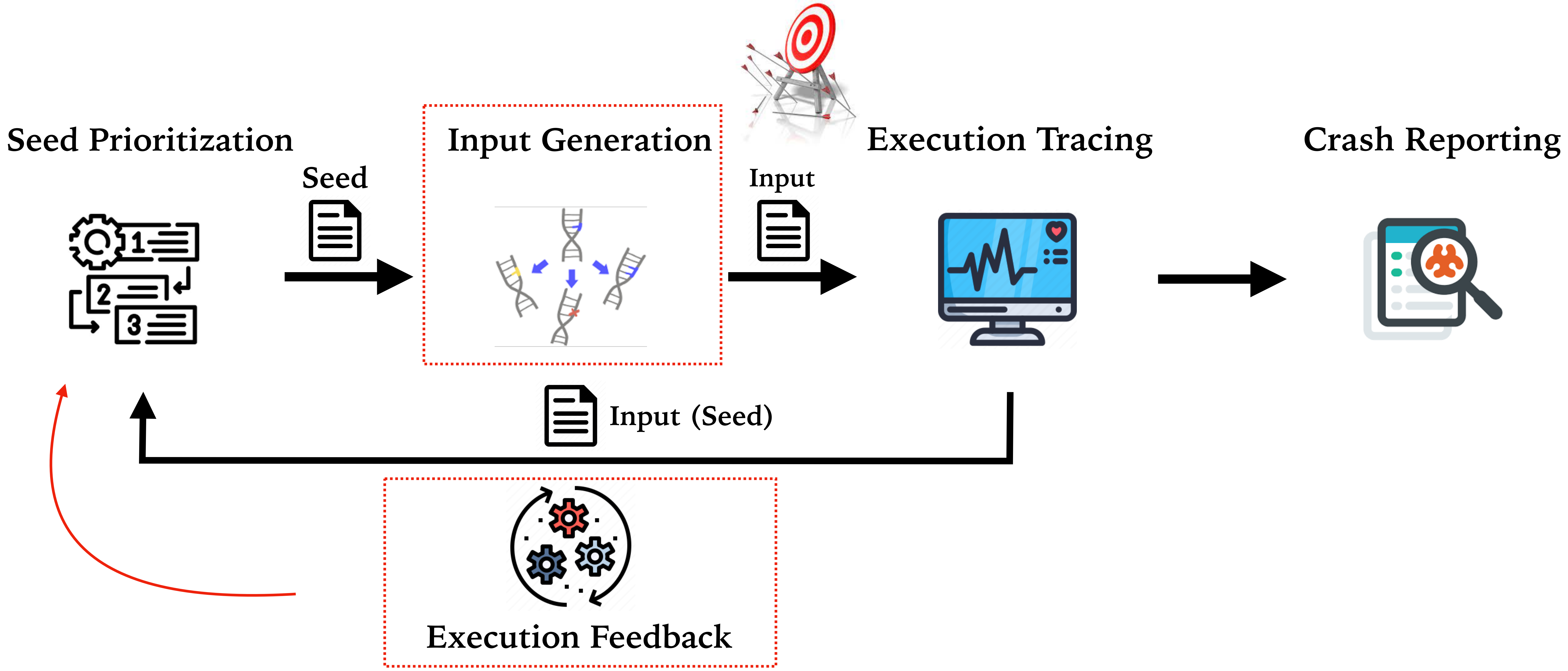


Project(ASAN version)	Execution times(24h)
libjpeg	25,238,863
nm	10,926,018
objdump	5,119,023
readelf	9,294,909
strip	8,090,507
tcpdump	5,828,969
tiff2ps	12,232,064
libpng	27,032,654
bento	10,102,720

Executing these infeasible inputs accounts for 88% of the time in fuzzing process

Indirect Input Generation Problem:

Existing directed fuzzing does not directly generate inputs toward the targets



Directed fuzzing proposes additional execution feedback to adjust the priority of the preserved input

Our Improvement

Halo

Counterexample-guided directed fuzzing

- Effective input generation

  6.2x more test cases reaching the targets

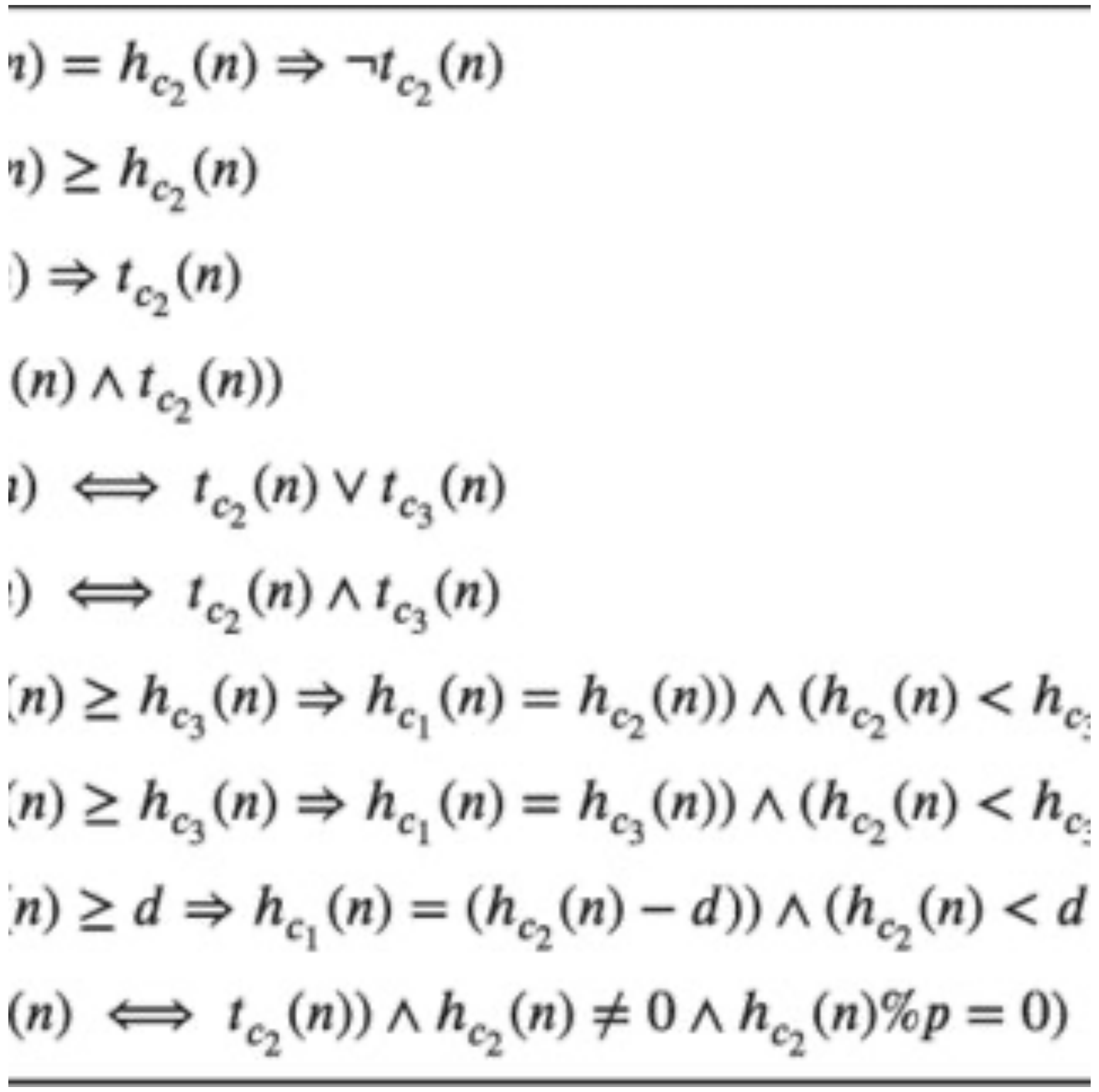
- Efficiency contribution

  15.3x speedup to detect the same bug

- Real-world practicalness

  10 incomplete fixes of previous CVEs/bugs

Challenges: Directly generate inputs towards the targets is time-consuming



$) = h_{c_2}(n) \Rightarrow \neg t_{c_2}(n)$

$) \geq h_{c_2}(n)$

$) \Rightarrow t_{c_2}(n)$

$(n) \wedge t_{c_2}(n))$

$) \iff t_{c_2}(n) \vee t_{c_3}(n)$

$) \iff t_{c_2}(n) \wedge t_{c_3}(n)$

$n) \geq h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_2}(n)) \wedge (h_{c_2}(n) < h_{c}$

$n) \geq h_{c_3}(n) \Rightarrow h_{c_1}(n) = h_{c_3}(n)) \wedge (h_{c_2}(n) < h_{c}$

$n) \geq d \Rightarrow h_{c_1}(n) = (h_{c_2}(n) - d)) \wedge (h_{c_2}(n) < d$

$(n) \iff t_{c_2}(n)) \wedge h_{c_2}(n) \neq 0 \wedge h_{c_2}(n)\%p = 0)$

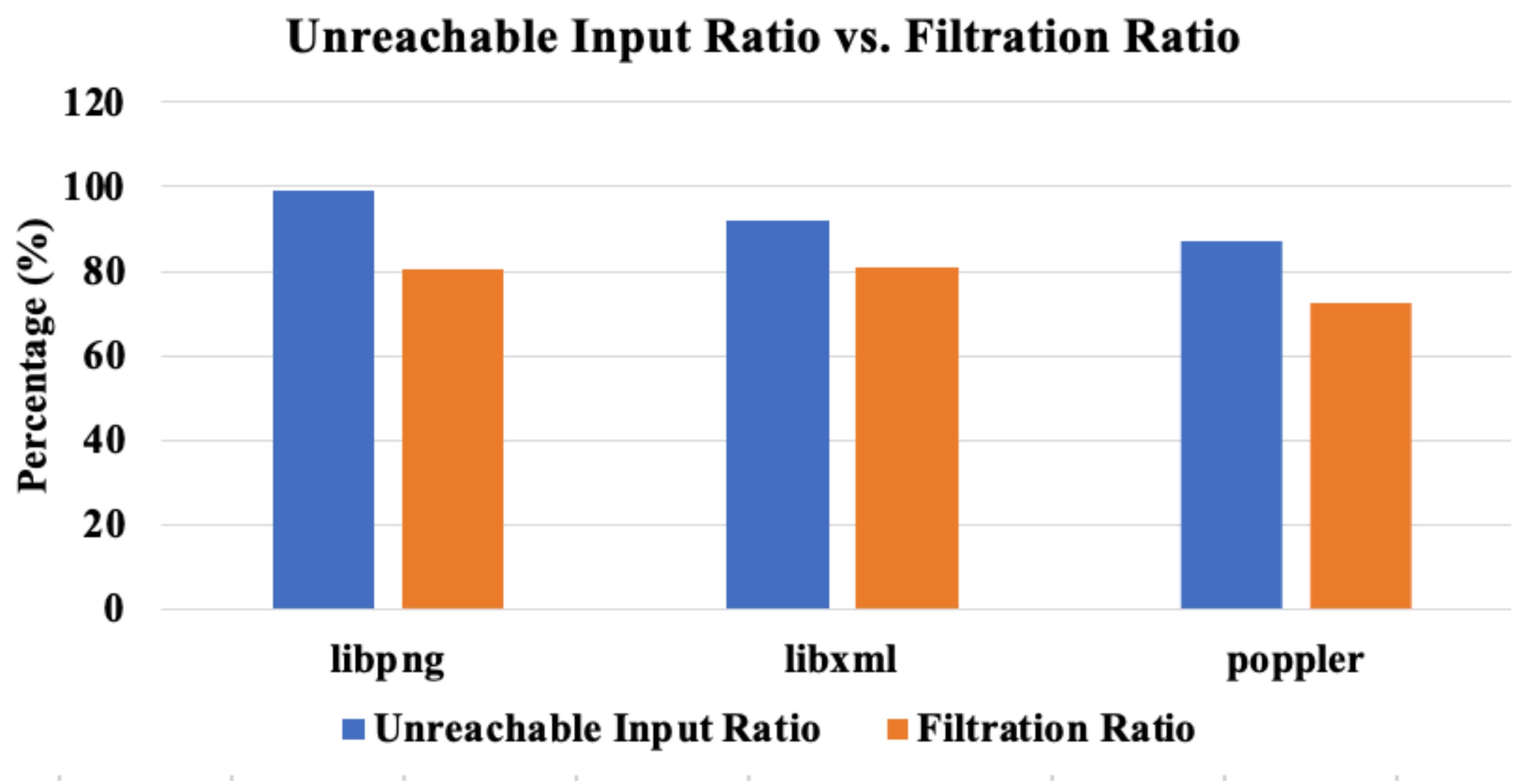
- Fuzzing needs to solve the path conditions

Path explosion + Expensive constraint solving

NP-HARD!

Intuition:

Can we leverage such large proportions of unreachable inputs to guide input generation?



Project(ASAN version)	Execution times(24h)
libjpeg	25,238,863
nm	10,926,018
objdump	5,119,023
readelf	9,294,909
strip	8,090,507
tcpdump	5,828,969
tiff2ps	12,232,064
libpng	27,032,654
bento	10,102,720

The unreachable input accounts for almost 100% of the generated inputs

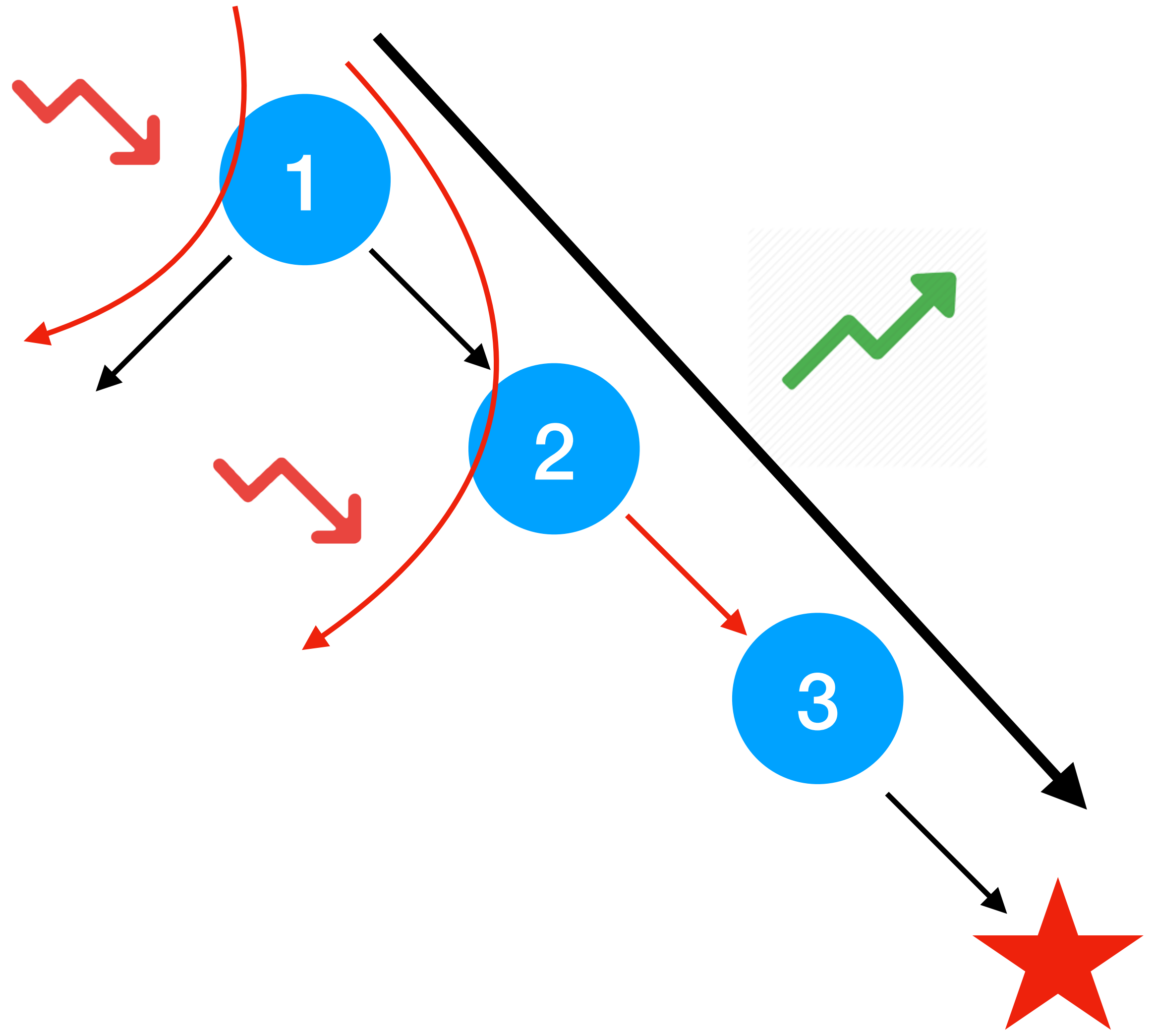
The filtration can over prune 80% of the generated inputs

Problem Summarization:

With given input satisfying certain patterns (path condition),

can we generate more similar/contradict inputs following the same pattern?

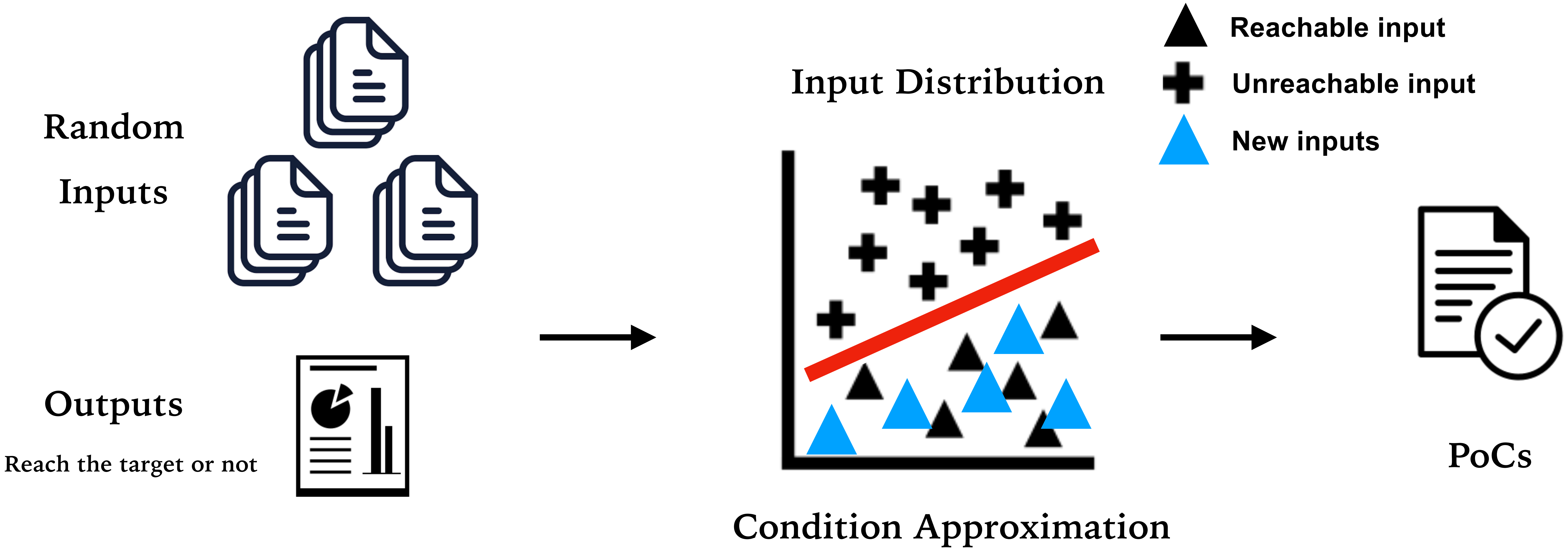
```
void fun() {
  int x, y, z = input();
  if (x == 10) {  ①
    if (lib_hash(y) > 30) { ②
      if (x + y <= 40) { ③
        //crash
      }
    }
  }
}
```



The frequency of inputs towards target increases along with fuzzing

Key insight

Approximate the conditions using existing fuzzing I/O to improve further input generation



Random Inputs

Outputs

Reach the target or not

Input Distribution

▲ Reachable input

✚ Unreachable input

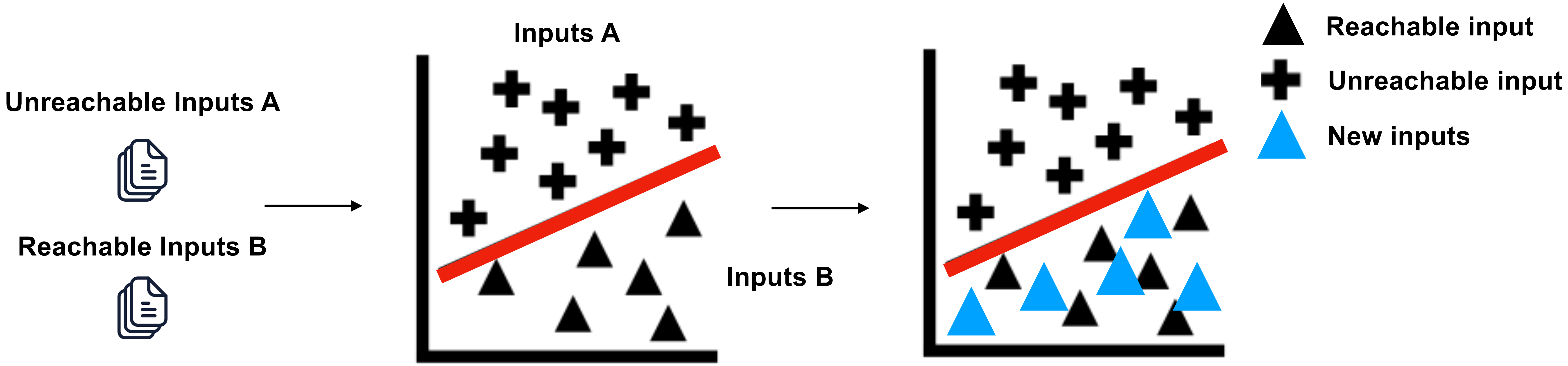
▲ New inputs

Condition Approximation

PoCs

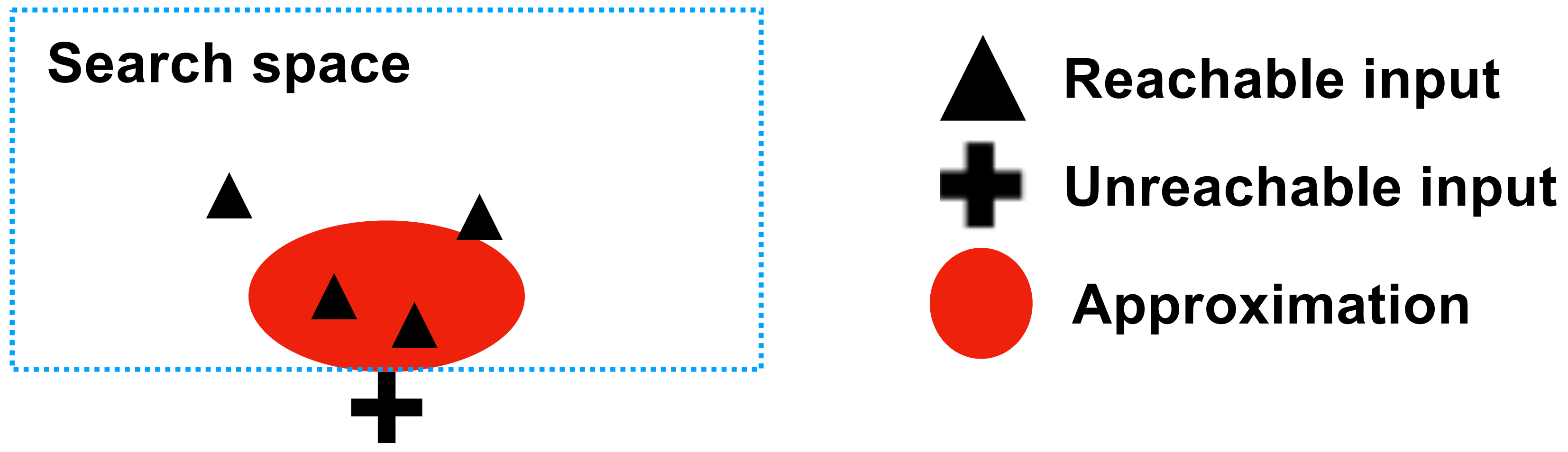
Fuzzer can adaptively optimize its input generation during the fuzzing process

Condition Approximation



Solutions:

Dynamic likely invariant inference, Daikon[1], Dig[2]



Approximation of the exact search space based on the given inputs

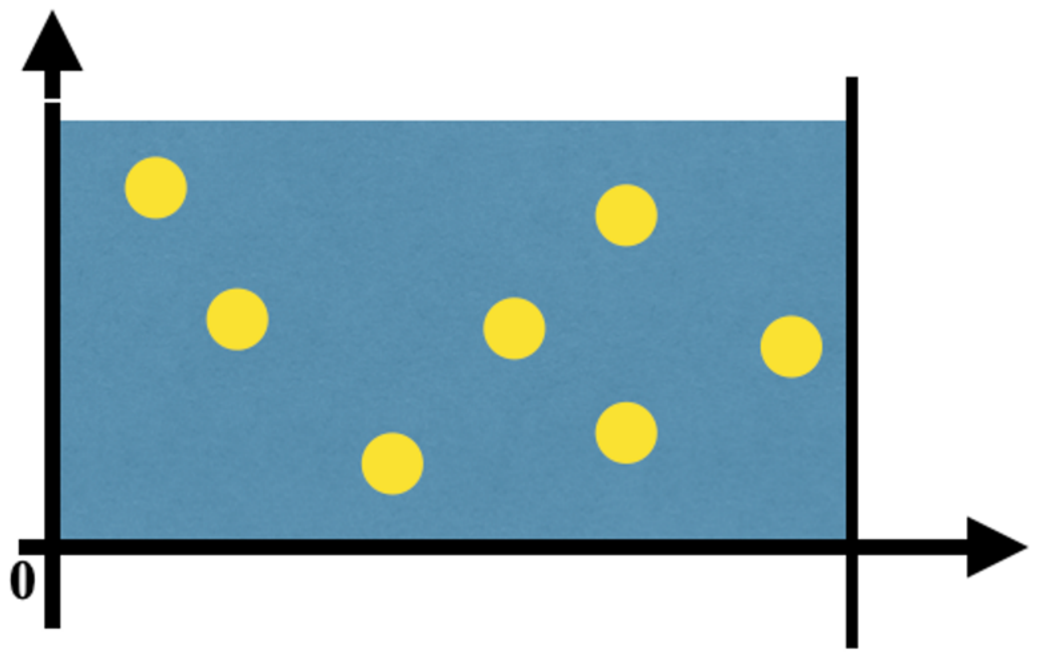
[1] Ernst, Michael D., et al. "The Daikon system for dynamic detection of likely invariants." *Science of computer programming* 69.1-3 (2007): 35-45.

[2] Clarke, Edmund, et al. "Counterexample-guided abstraction refinement." *International Conference on Computer Aided Verification*. Springer, Berlin, Heidelberg, 2000.

Condition Approximation

```
void fun() {
  int x, y, z = input();
  if (x == 10) {
    if (lib_hash(y) > 30) {
      if (x + y <= 40) {
        //crash
      }
    }
  }
}
```

Sample the inputs from the constrained search space described by the invariant

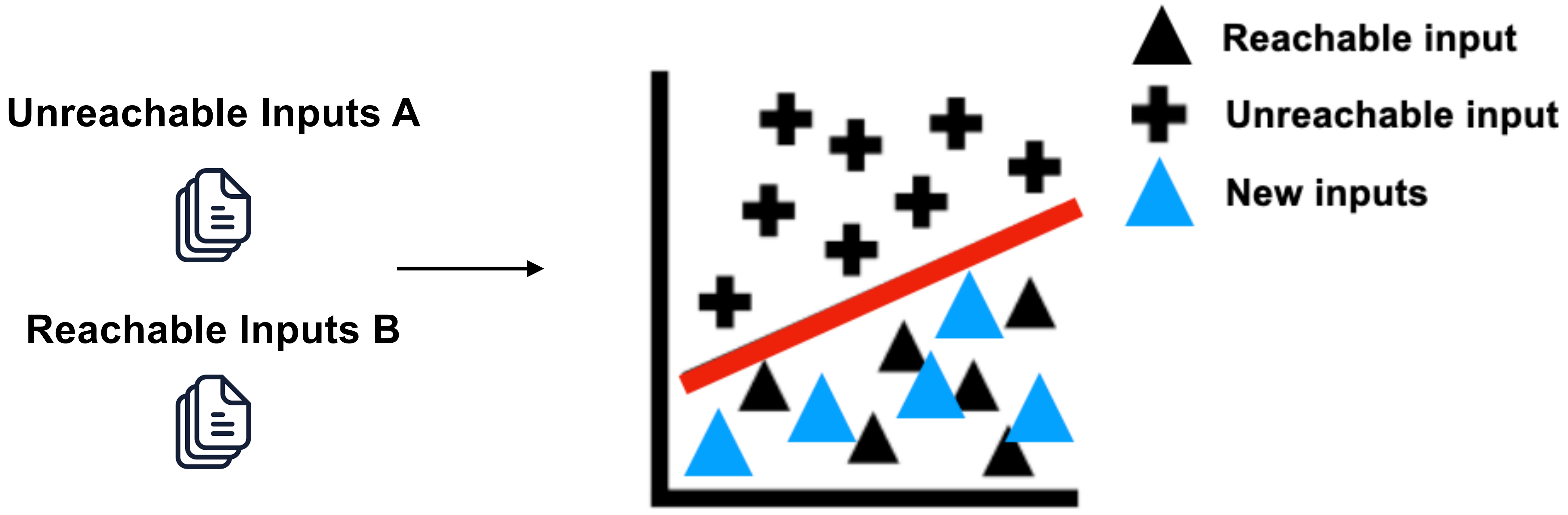


x==0	x==0	x==0	x==10	
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5

Key obstacle: Dimensional Curse

Challenge 1: How to infer conditions from executed inputs efficiently?

Challenge 2: How to generate inputs constrained by conditions efficiently?

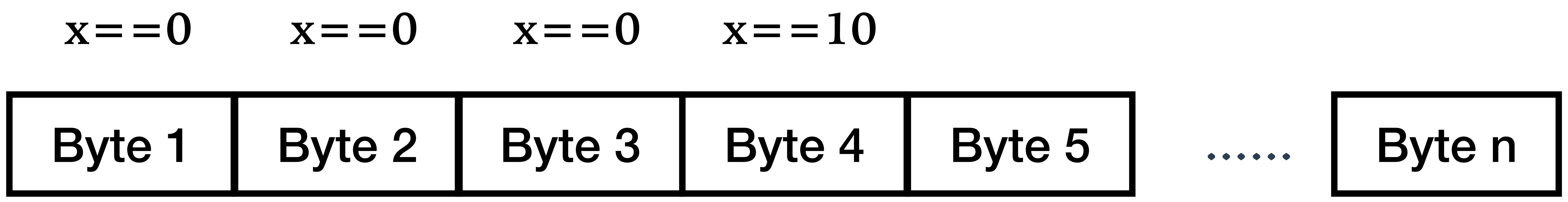


The restriction for the inputs consists of three dimensions

Input Bytes X Values X Relations

Reduced Dimension (Input Bytes): Taint Inference

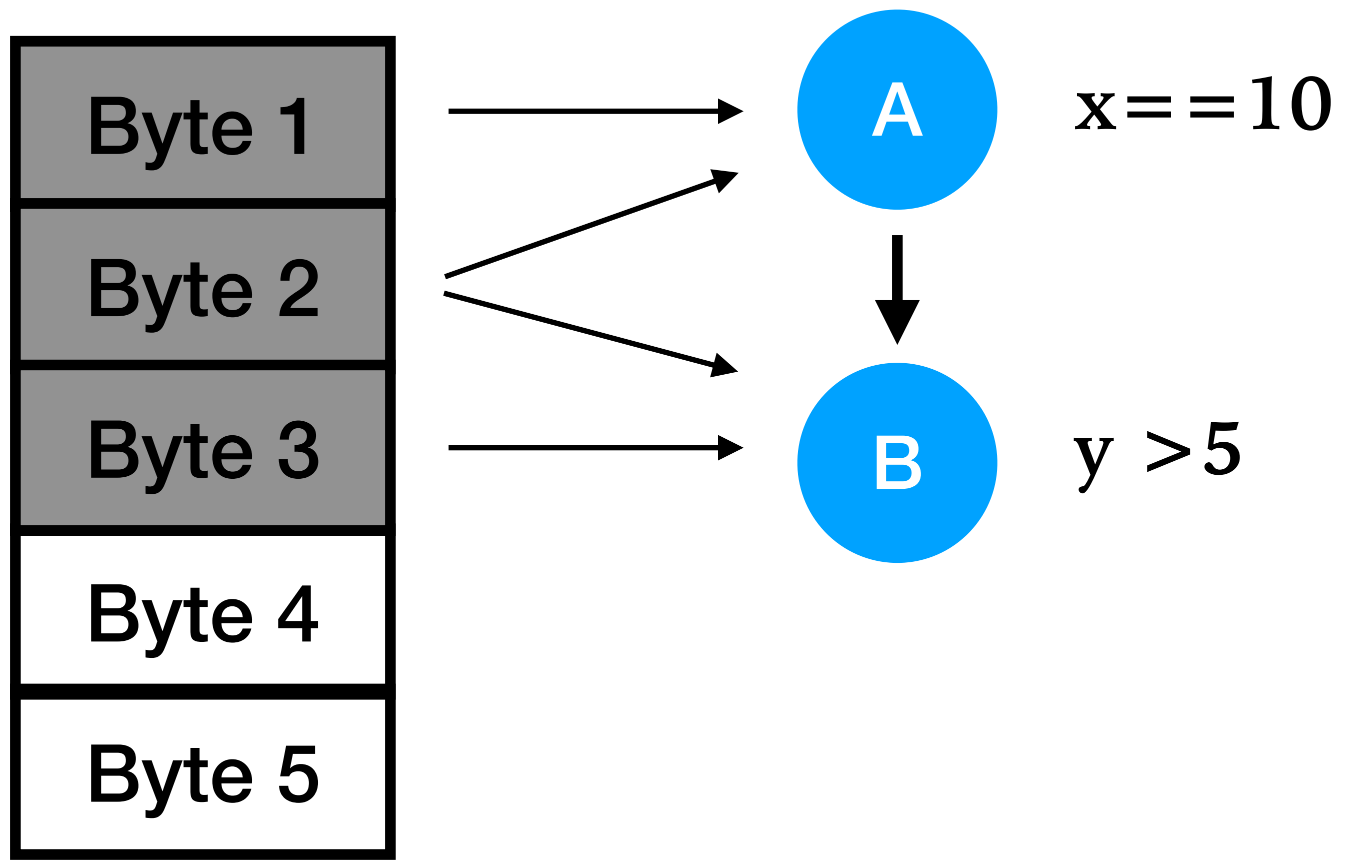
Invariant inference is not scale for large input size, e.g., few Kb



x==0 x==0 x==0 x==10

Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 …… Byte n

Taint inference through execution to filter the irrelevant bytes



Byte 1
Byte 2
Byte 3
Byte 4
Byte 5

A x==10

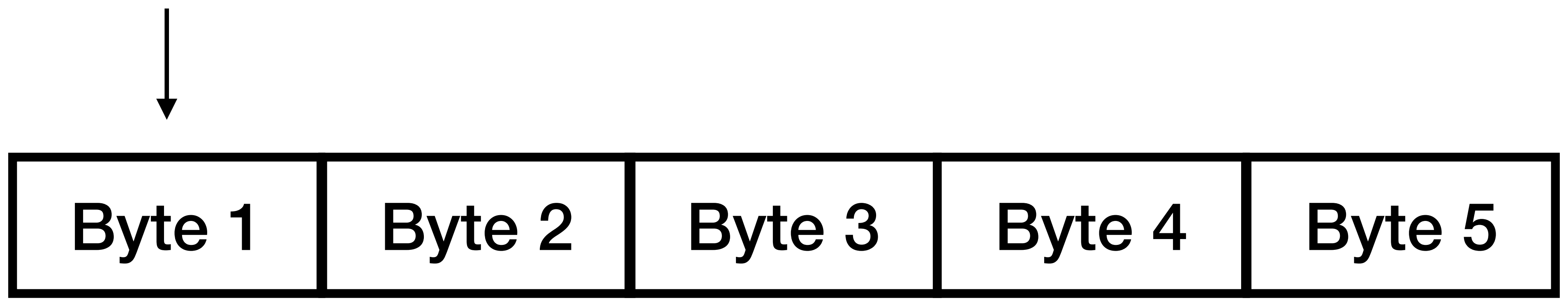
B y >5

The byte is relevant if it influences the variable values in the branch conditions reachable to the target

Reduced Dimension (Input Bytes): Taint Inference

The byte is relevant if it influences the variable values in the reachable branch conditions

```
void fun() {
  char x, y, z = input();
  if (x == 10) {
    if (lib_hash(y) > 30) {
      if (x + y <= 40) {
        //crash
      }
    }
  }
}
```



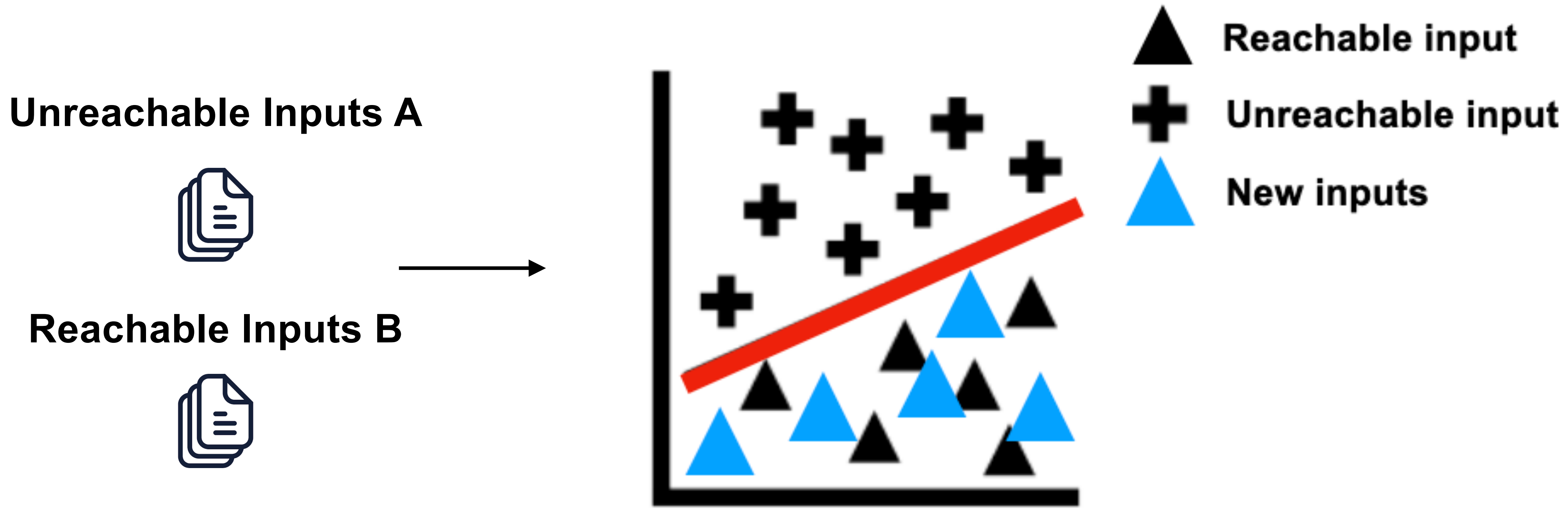
Byte 1 Byte 2 Byte 3 Byte 4 Byte 5

Byte 1 is relevant since mutating byte 1 influences the value of x

Key obstacle: Dimensional Curse

Challenge 1: How to infer invariant from executed inputs efficiently?

Challenge 2: How to generate inputs constrained by conditions efficiently?



Unreachable Inputs A

Reachable Inputs B

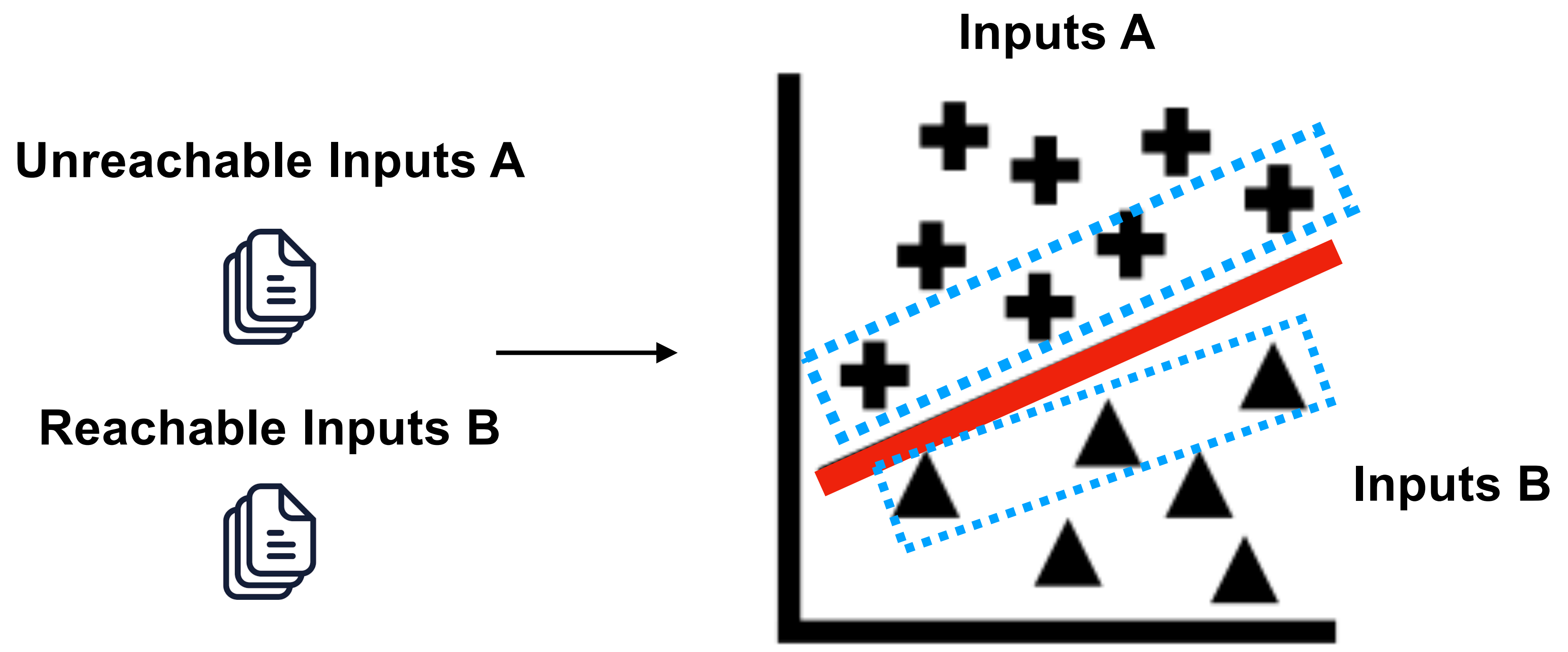
The restriction for the inputs consists of three dimensions

Input Bytes X Values X Relations

Too many input (values) for approximating the conditions

Challenges: How many input is needed?

Intuition: Not all input contribute equally for the approximation



Unreachable Inputs A

Reachable Inputs B

Inputs A

Inputs B

Only the input close the boundary that helps

To approximate the condition x > 10:

$\text{Inputs}_{\text{feasible}} = 11, 12$ $\text{Inputs}_{\text{infeasible}} = 8, 9$ ⟶ $x > 10$

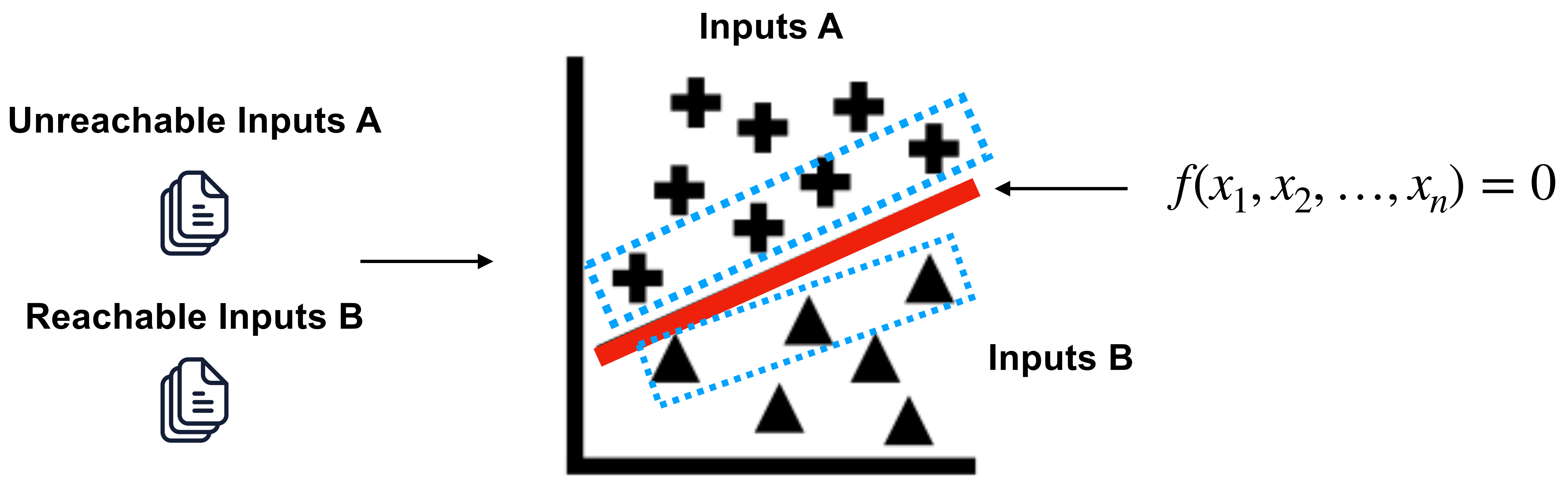
$\text{Inputs}_{\text{feasible}} = 1000, 2000$ $\text{Inputs}_{\text{infeasible}} = -100, -200$ ⟶ $x > 500$

Reduced Dimension (Values): Distance towards the boundary

A path condition can be transformed into:

$$f(x_1, x_2, \ldots, x_n) \geq 0$$

Choose the input prioritized by the closeness toward the boundary



Distance: $|f(x_1, x_2, \ldots, x_n)|$

To approximate the condition x > 10: ⟶ $f(x)$: x - 10 > 0

✓ Inputs$_{\text{feasible}}$ = 11, 12   Inputs$_{\text{infeasible}}$ = 8, 9 ⟶ Distance: 1 and 2

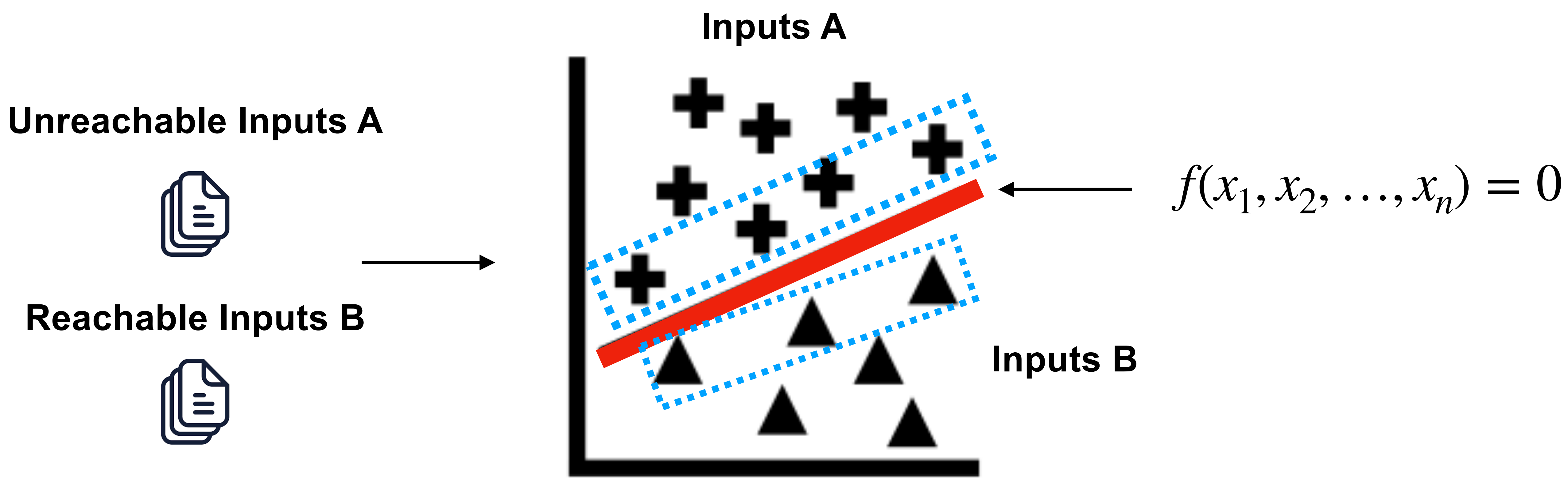
Inputs$_{\text{feasible}}$ = 1000, 2000   Inputs$_{\text{infeasible}}$ = -100, -200 ⟶ Distance: >100

Reduced Dimension (Values): Distance towards the boundary

A path condition can be transformed into:

$$f(x_1, x_2, \ldots, x_n) \geq 0$$

Choose the input prioritized by the closeness toward the boundary



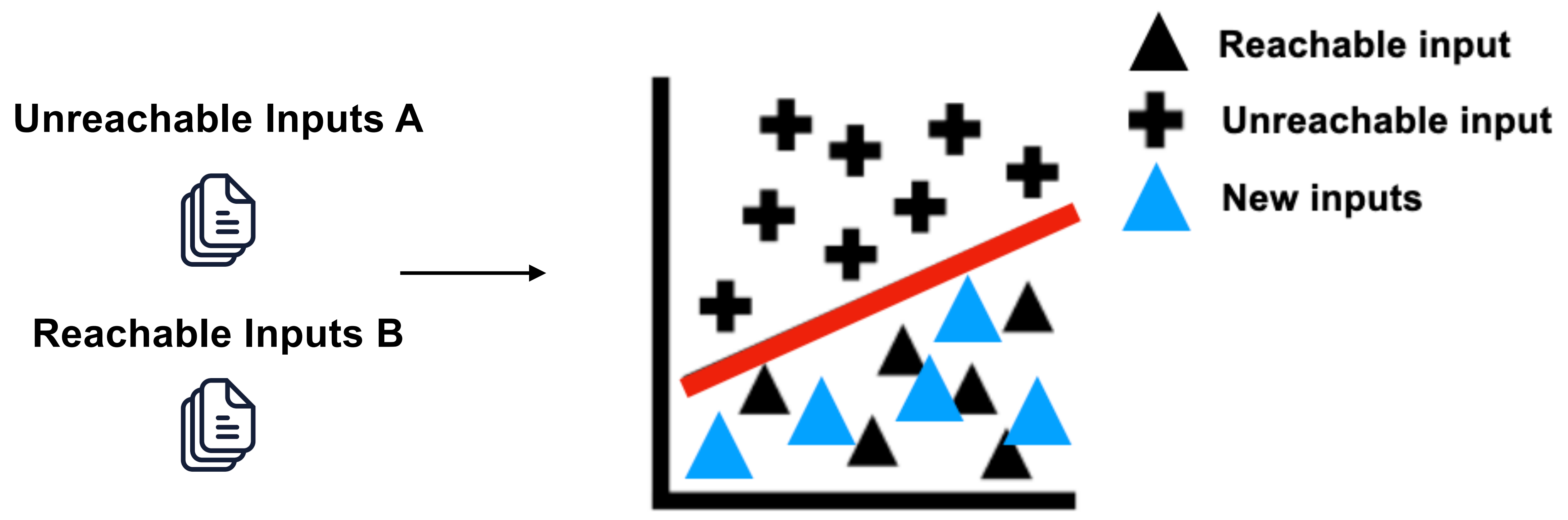
Distance: $|f(x_1, x_2, \ldots, x_n)|$

We then calculate the sample size based on the statistic to satisfy the confident interval where $\alpha > 0.95$

Key obstacle: Dimensional Curse

Challenge 1: How to infer conditions from executed inputs efficiently?

Challenge 2: How to generate inputs constrained by conditions efficiently?



Unreachable Inputs A

Reachable Inputs B

▲ Reachable input

✚ Unreachable input

▲ New inputs

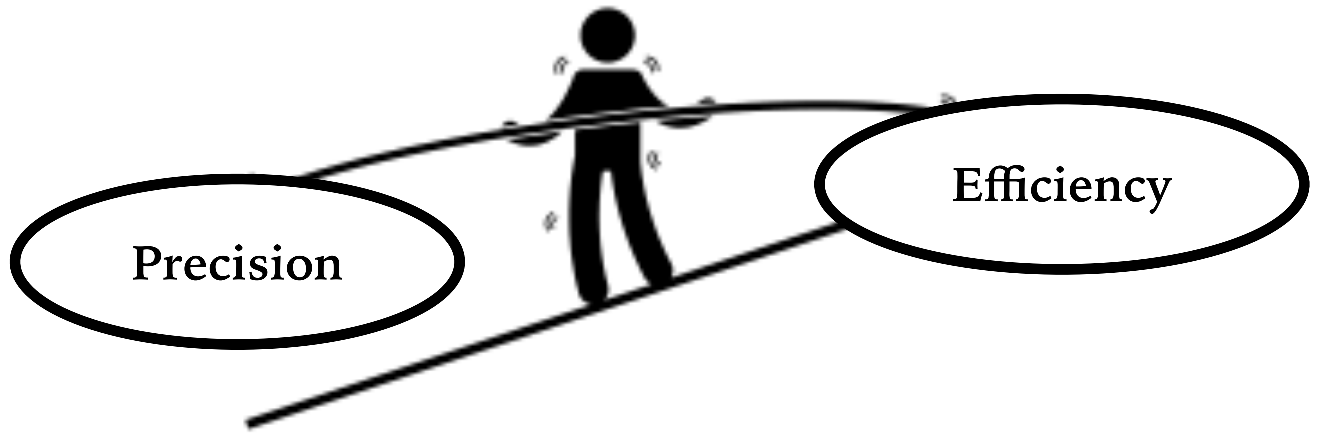
The restriction for the inputs consists of three dimensions

Input Bytes X Values X Relations

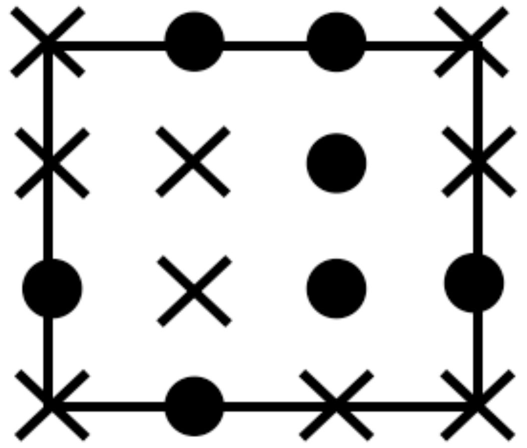
Too many relations could exists for input generation

Challenge: efficiency tradeoff between approximation refining and input generation

With more relations, sampling is less efficient. O(n)

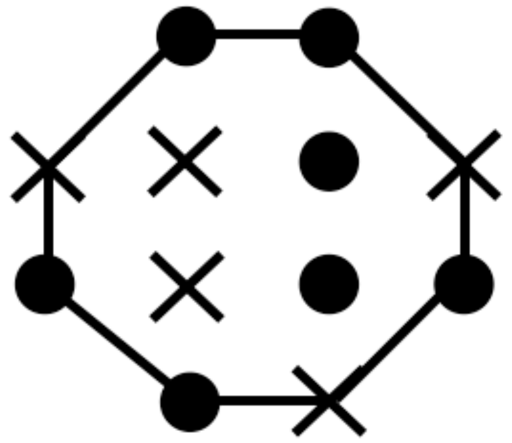


✖ Infeasible instance ● feasible instance



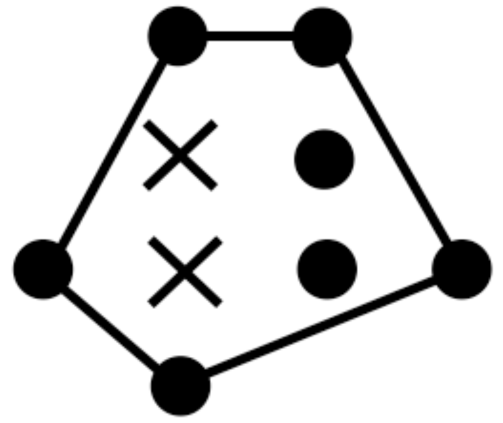
$$x \in [a, b]$$
$$y \in [c, d]$$

Interval



$$k_1 x + k_2 y \leq b$$
$$k_i \in \{0, \pm 1\}$$

Octagon



$$k_1 x + k_2 y \leq b$$
$$k_i \in \mathbb{Z}$$

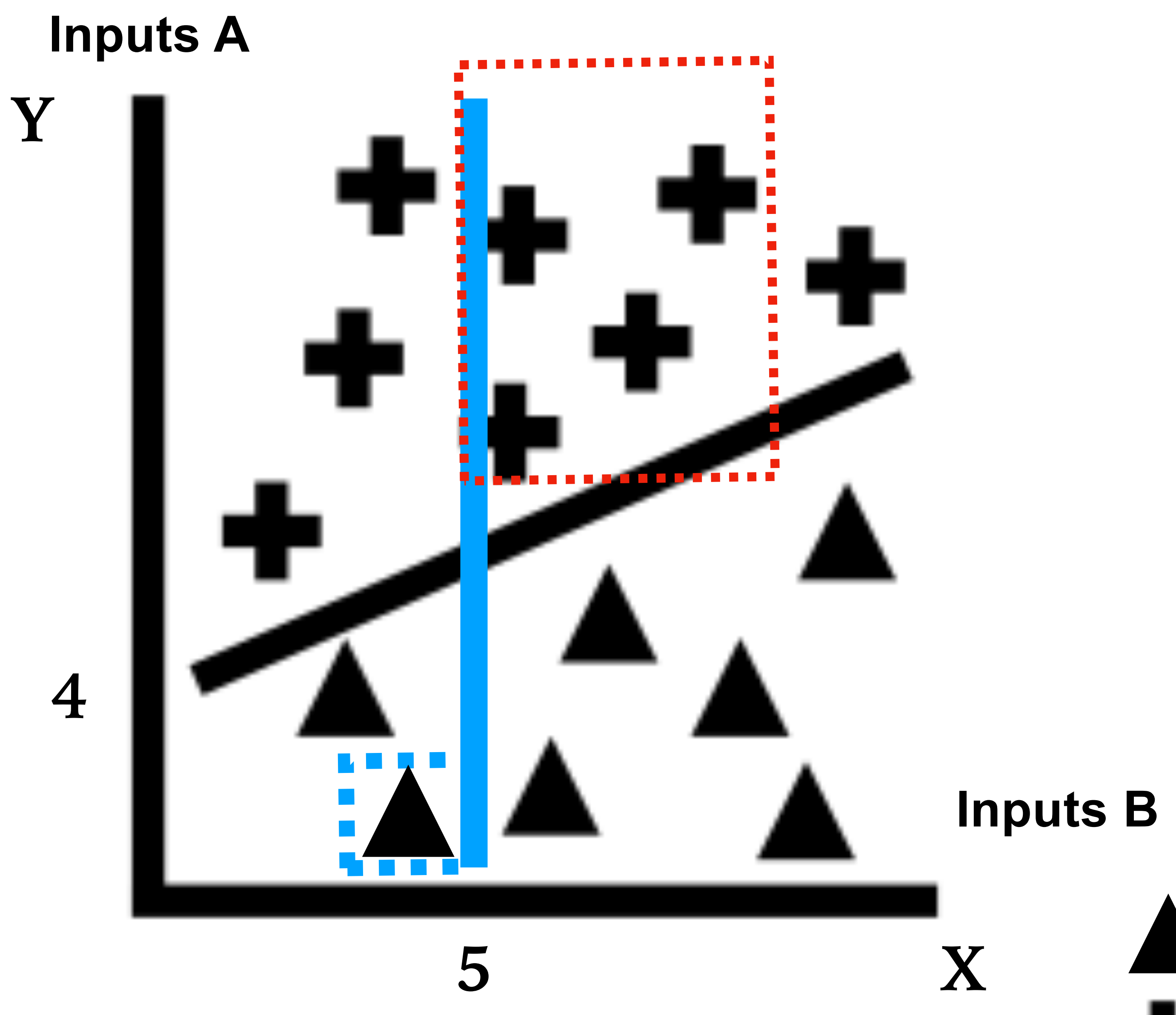
Polyhedra

Efficiency ←→ Precision

Reduced Dimension (Relations): Importance Sampling

For each relation, we assign an initial importance

Importance represents the likelihood of the feasible inputs containing in the regions



▲ Reachable input

✚ Unreachable input

x > 5

For counterexample, the confidence interval decreases

(x,y): (6, 100) ↘

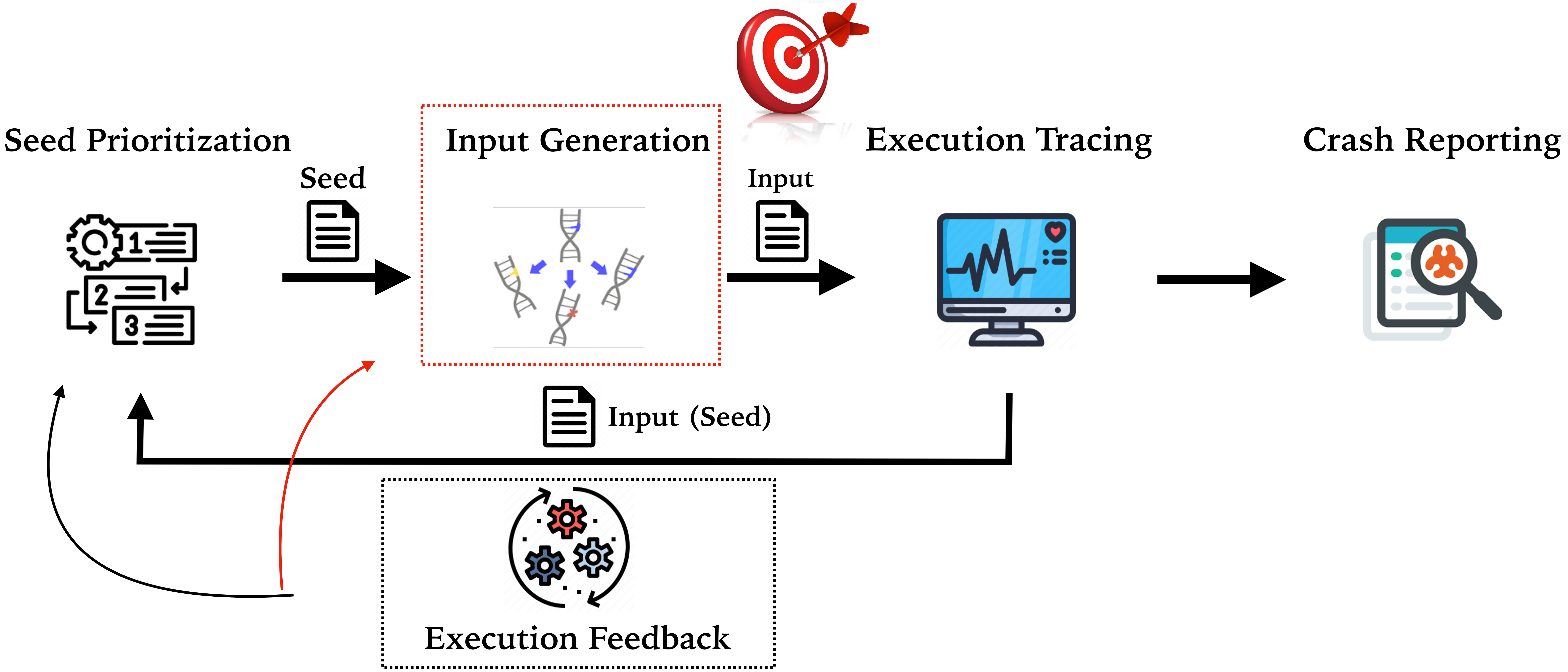
For correct sampling, the confidence interval increases

(x,y): (4, 1) ↗

Fuzzer can adaptively use more reliable relations

Indirect Input Generation Problem:

Existing directed fuzzing does not directly generate inputs

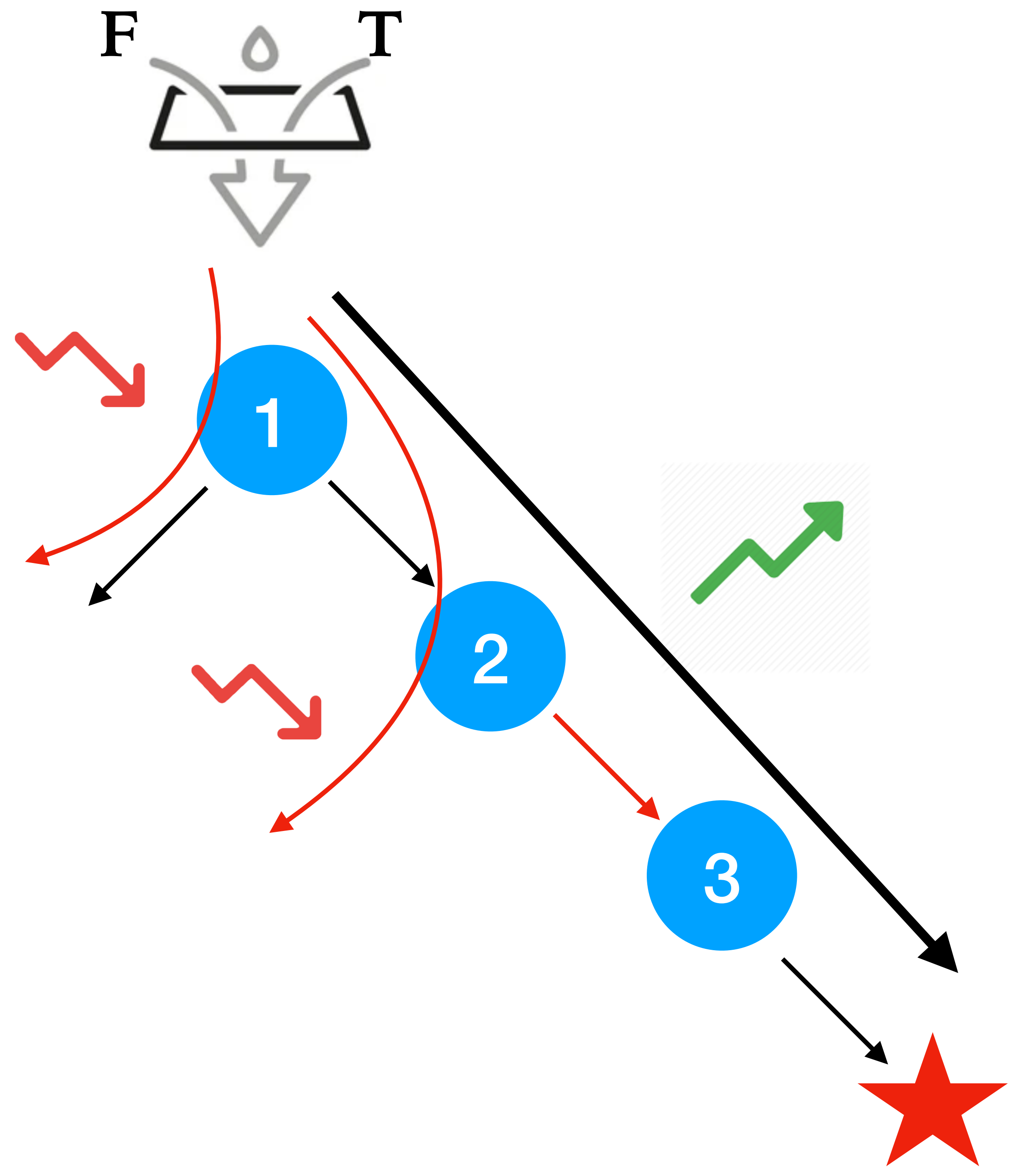


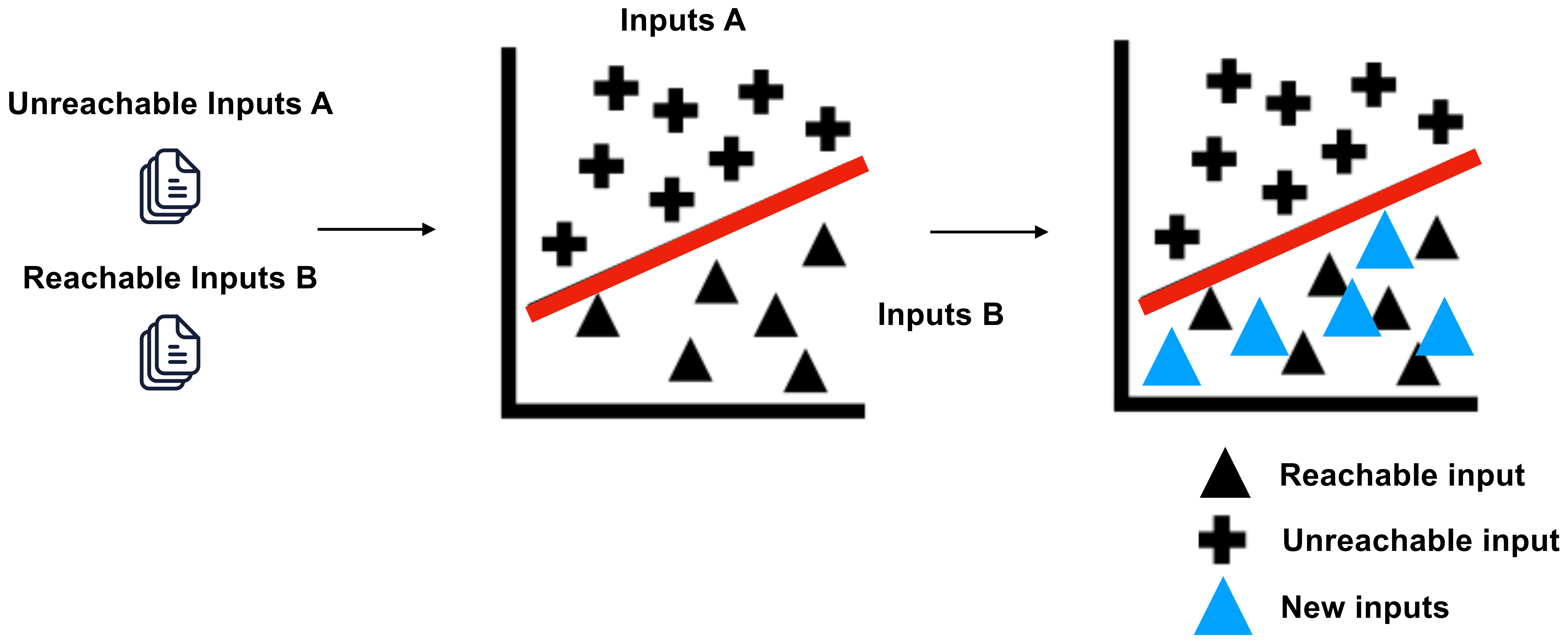
Directed fuzzing proposes additional execution feedback to adjust the priority of the preserved input

Conclusion: Everything is good for something

Halo

Make the input generation directed toward the target via likely invariant generation





The frequency of inputs towards target increases along with fuzzing

香港城市大學
City University of Hong Kong

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

EPFL