MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY

Magic of Statistics for Software Testing: How to Foresee the Unseen

Seongmin Lee
Max Planck Institute for Security and Privacy (MPI-SP)

25.04.28 | SBFT'25

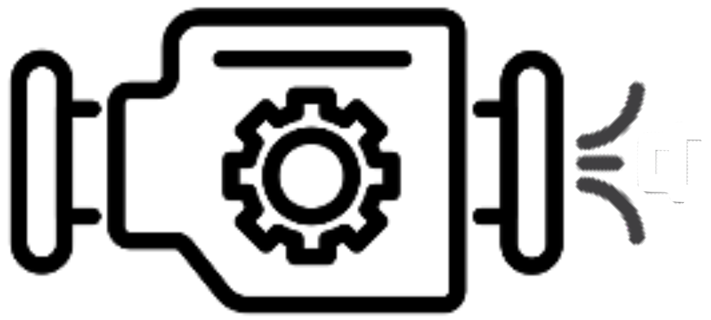
Software Testing — Test by Actual Running

Software Testing — Test by Actual Running

For example, a **Fuzzing**,

Software Testing — Test by Actual Running

For example, a **Fuzzing**,



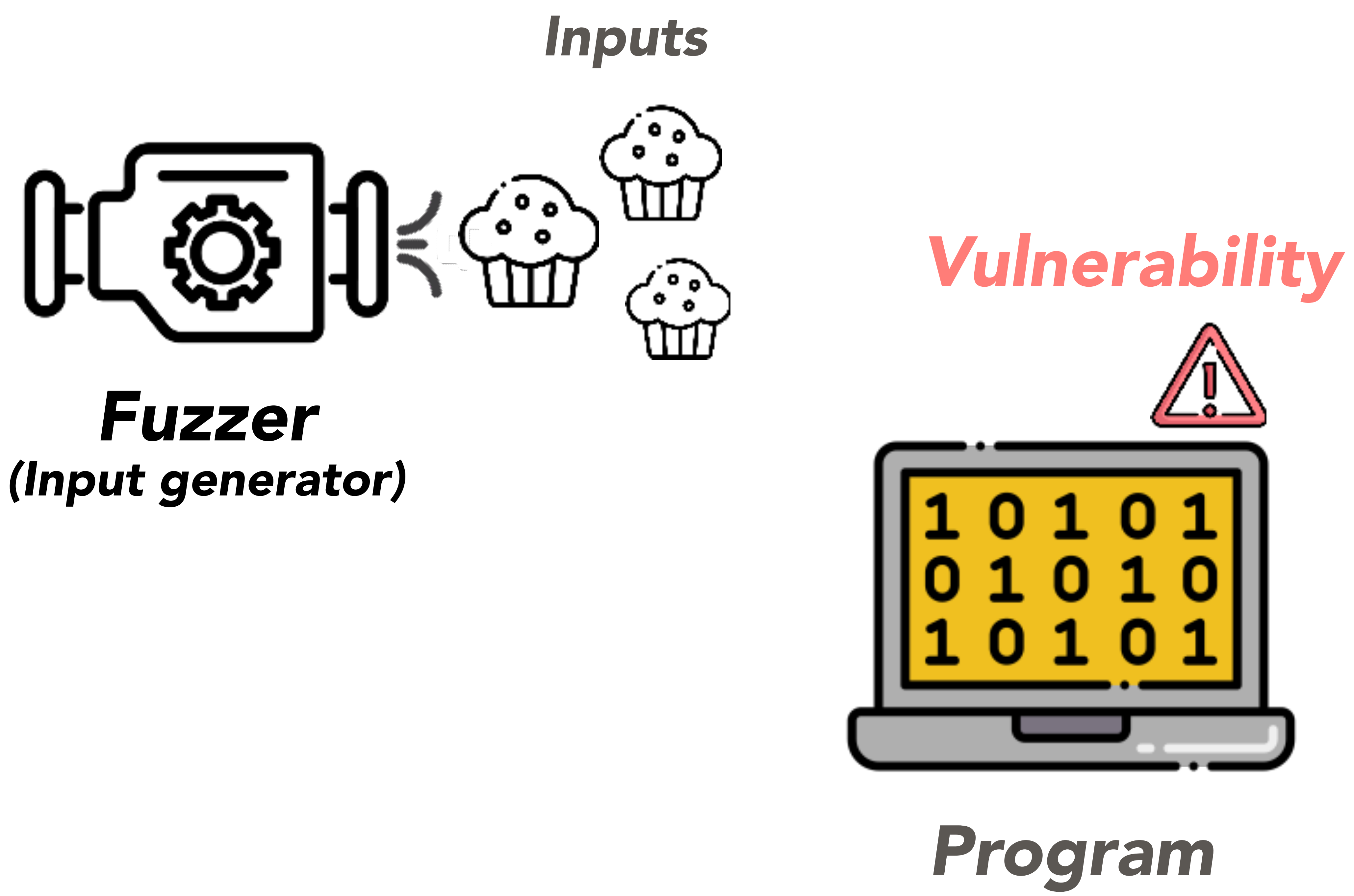
Fuzzer
(Input generator)

Vulnerability

Program

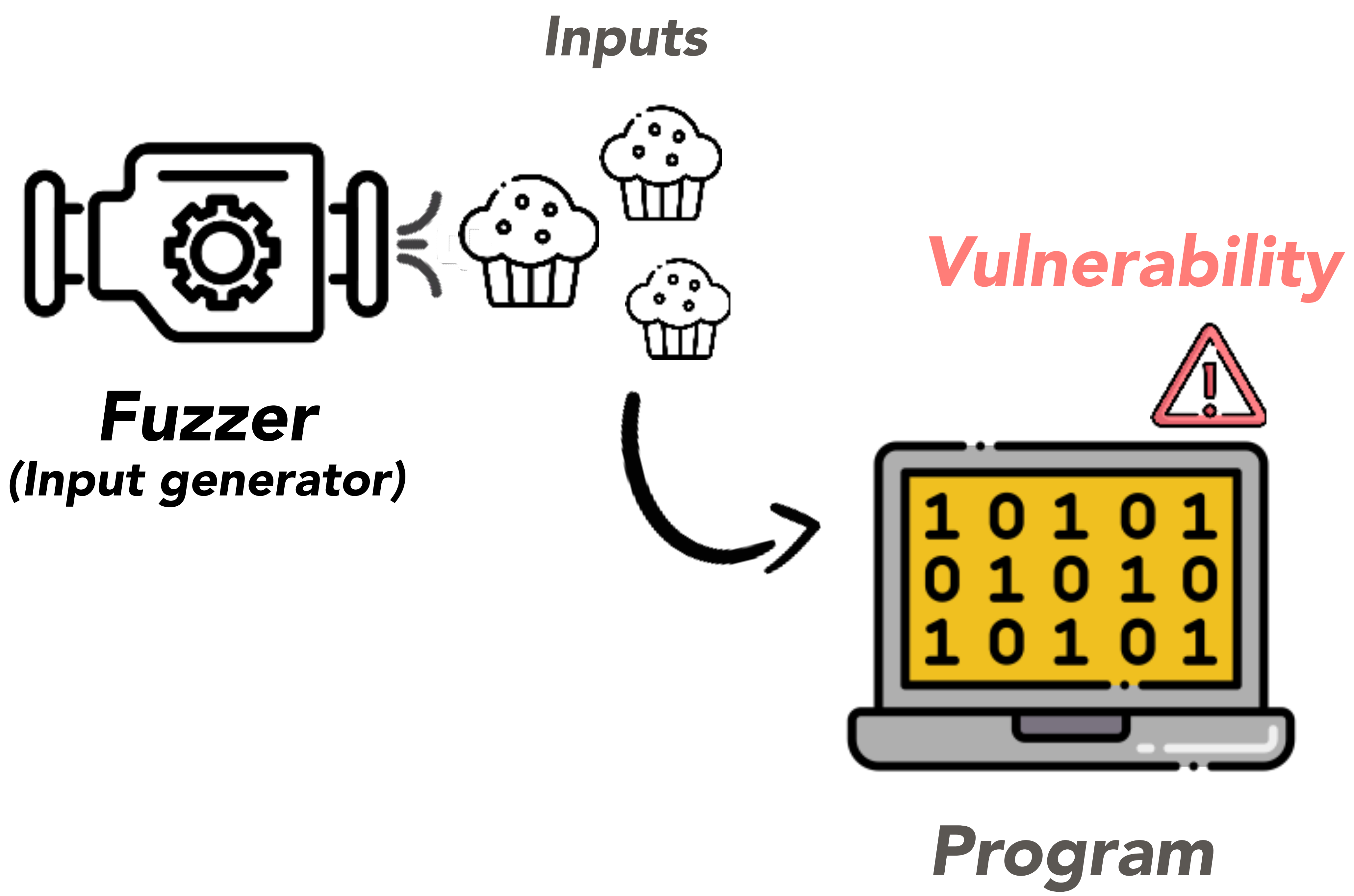
Software Testing — Test by Actual Running

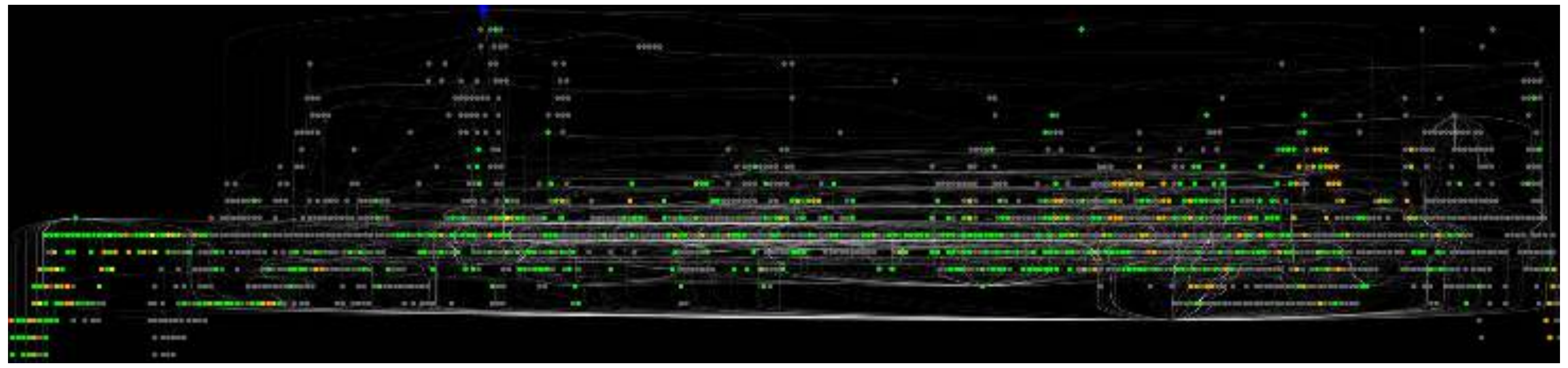
For example, a **Fuzzing**,

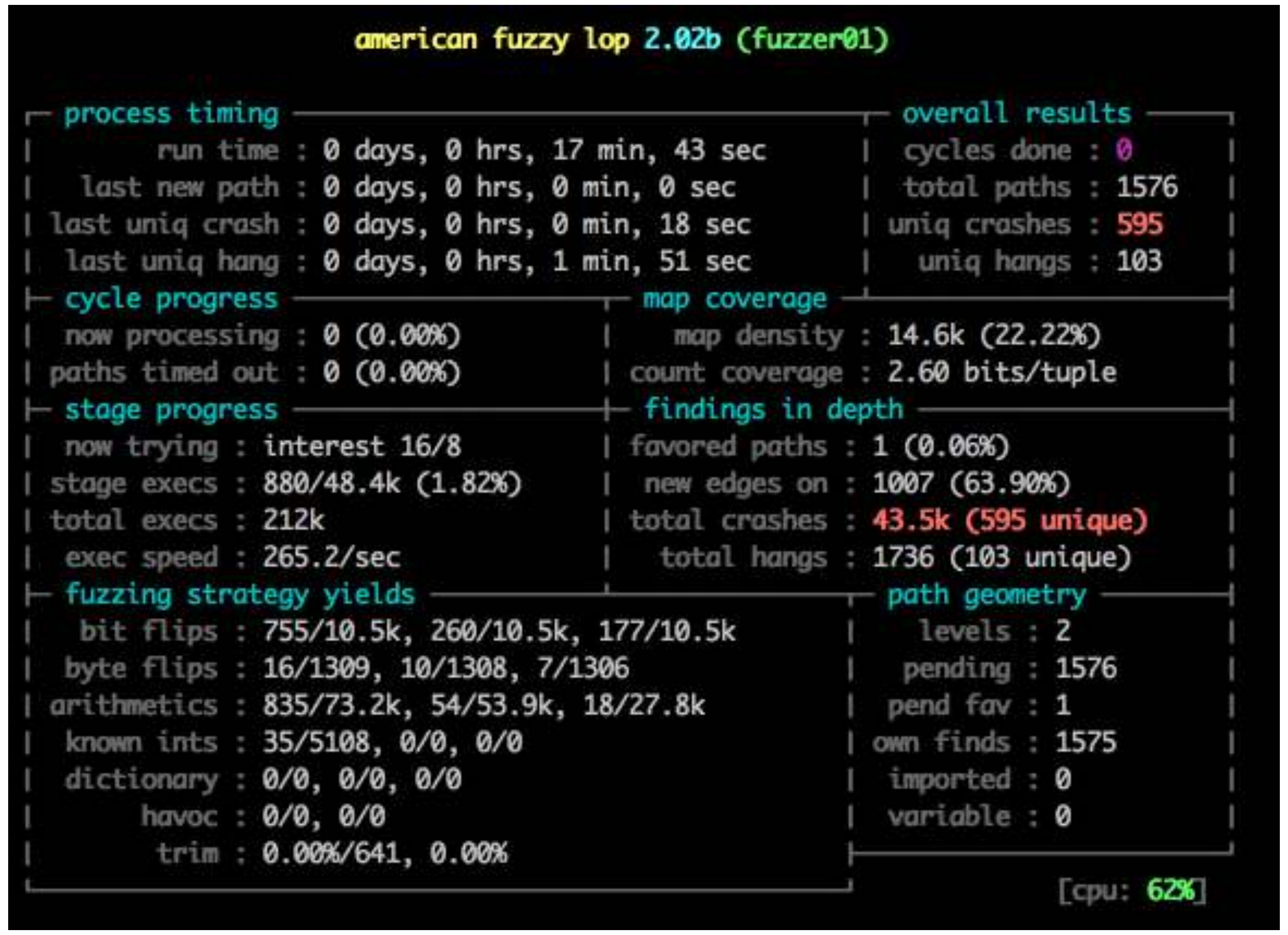


Software Testing — Test by Actual Running

For example, a **Fuzzing**,

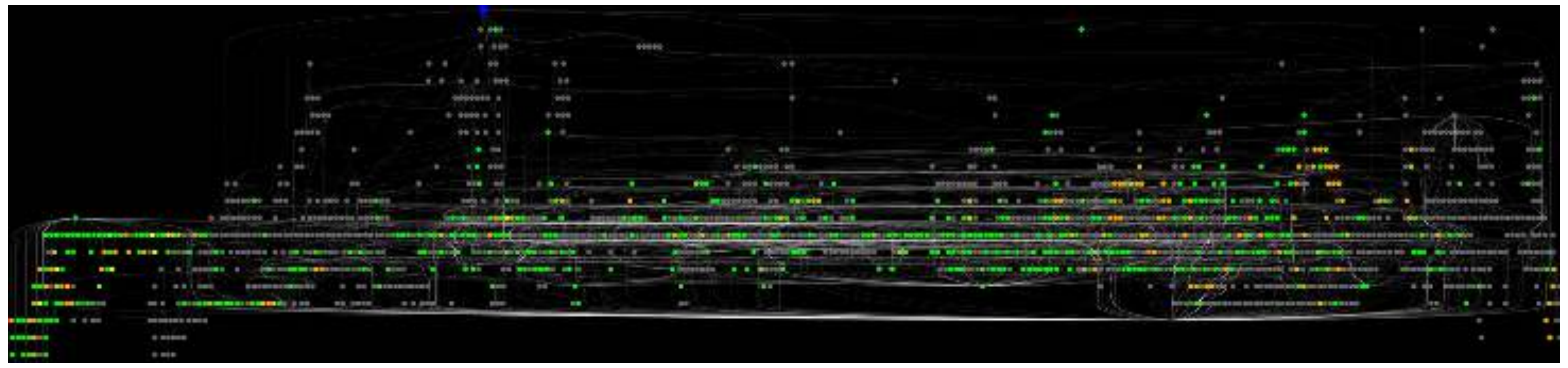


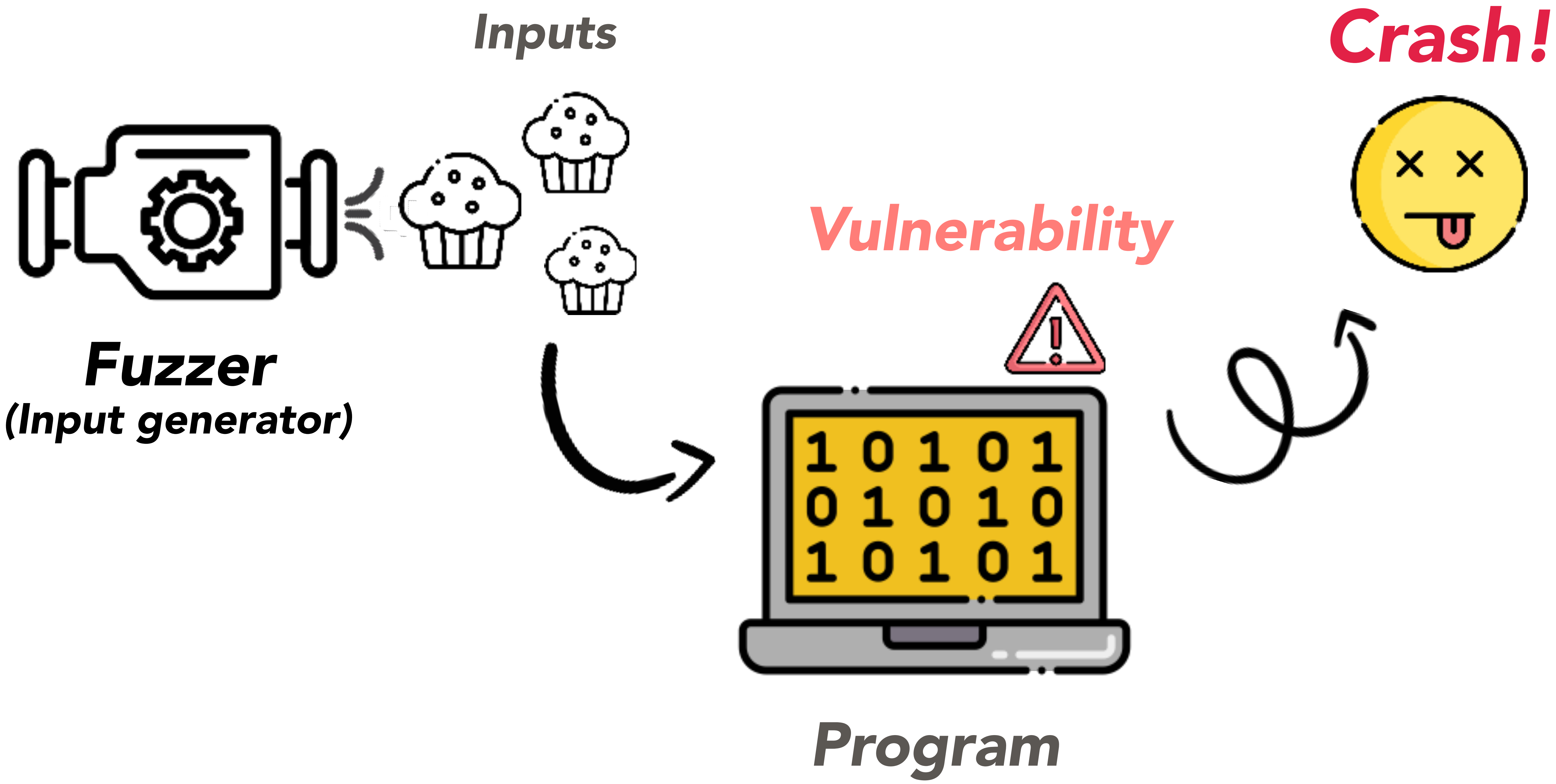


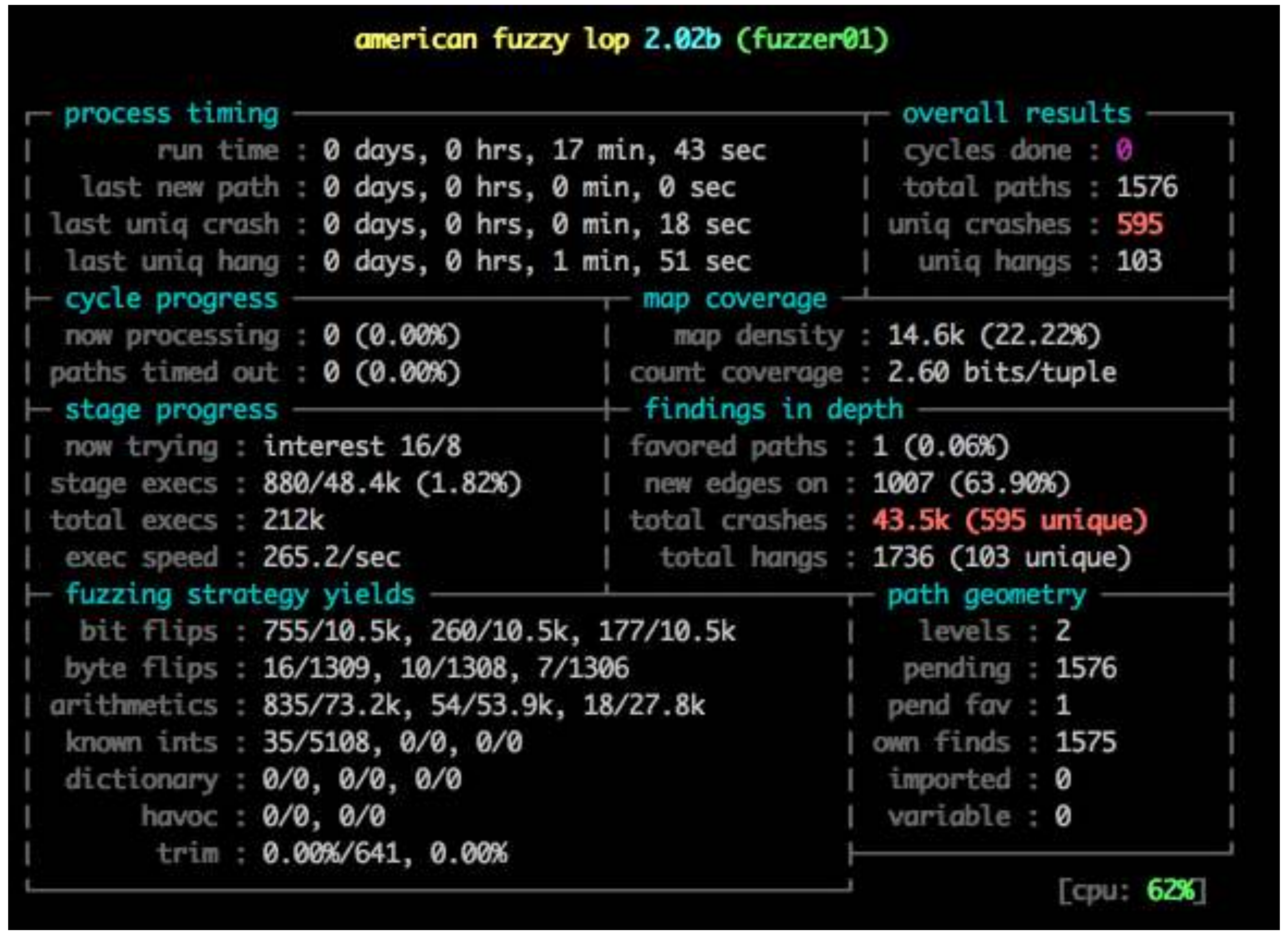


Software Testing — Test by Actual Running

For example, a **Fuzzing**,



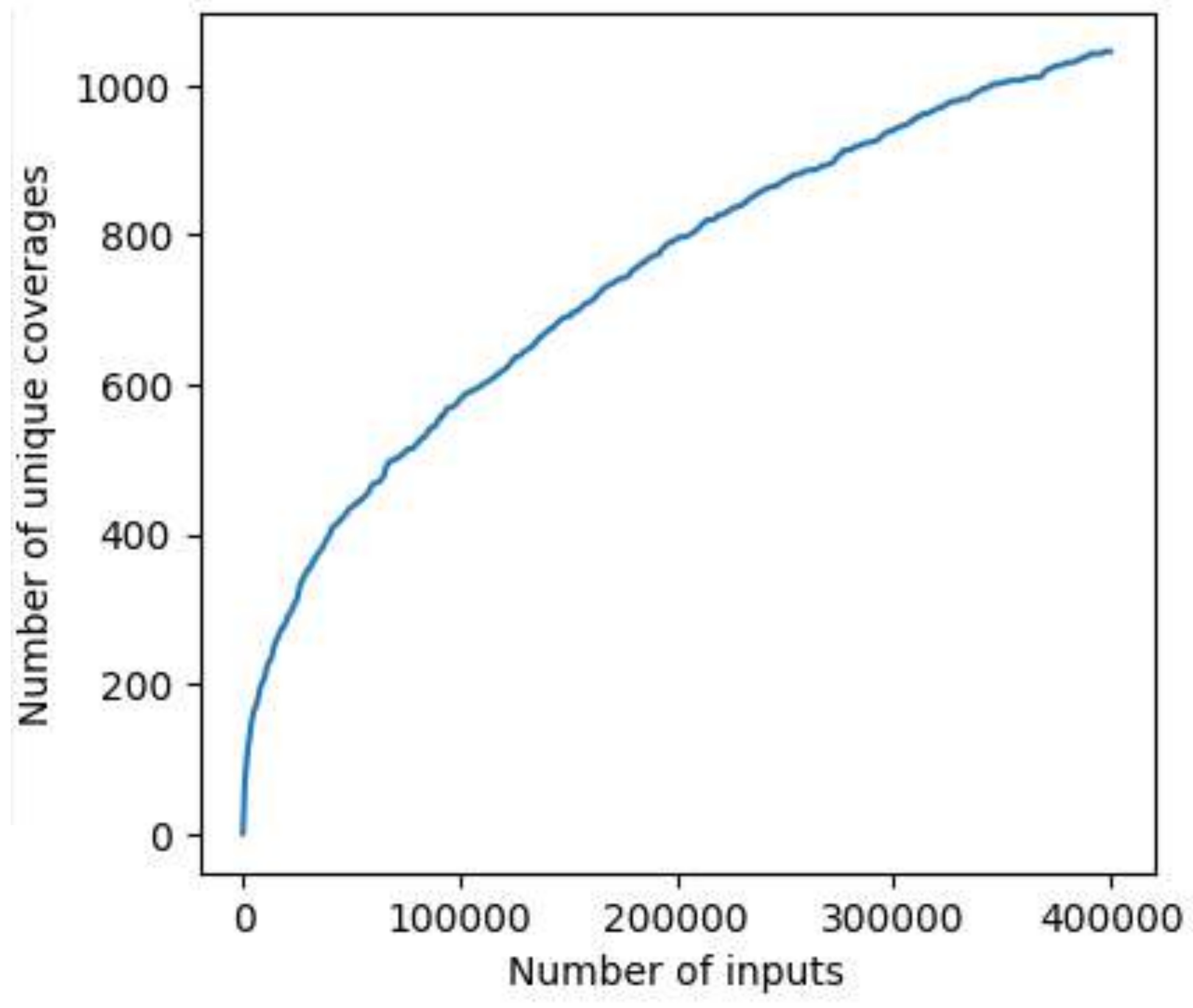




As a fuzzing campaign progresses,

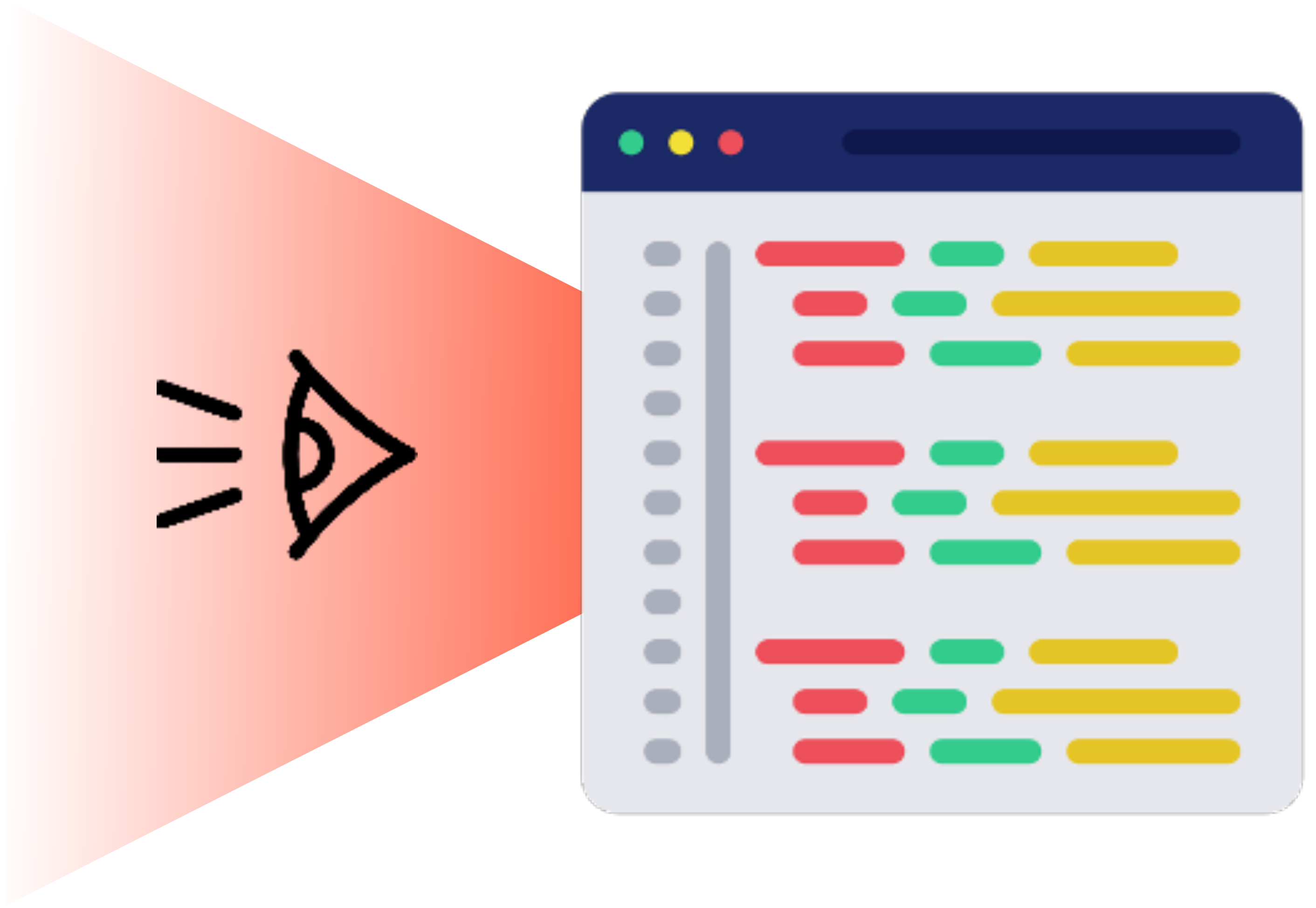
Program

As a fuzzing campaign progresses,



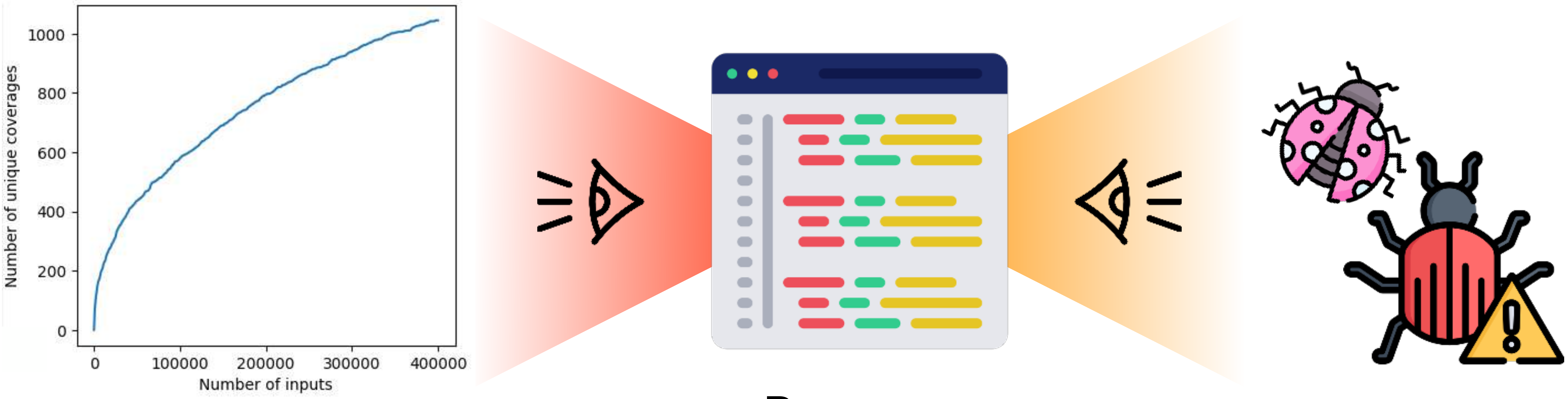
Coverage Increase

(lines or basic blocks)



Program

As a fuzzing campaign progresses,



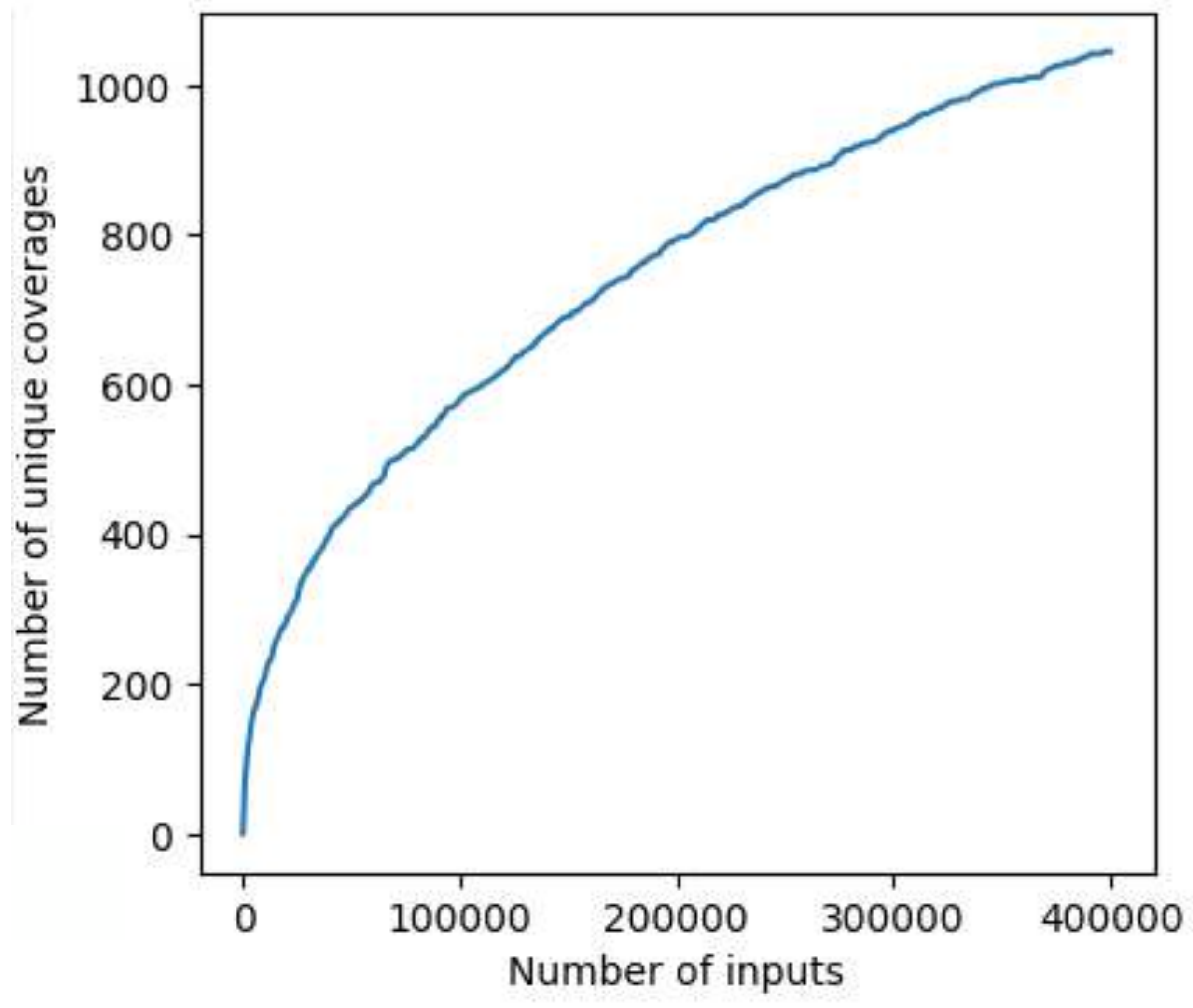
Program

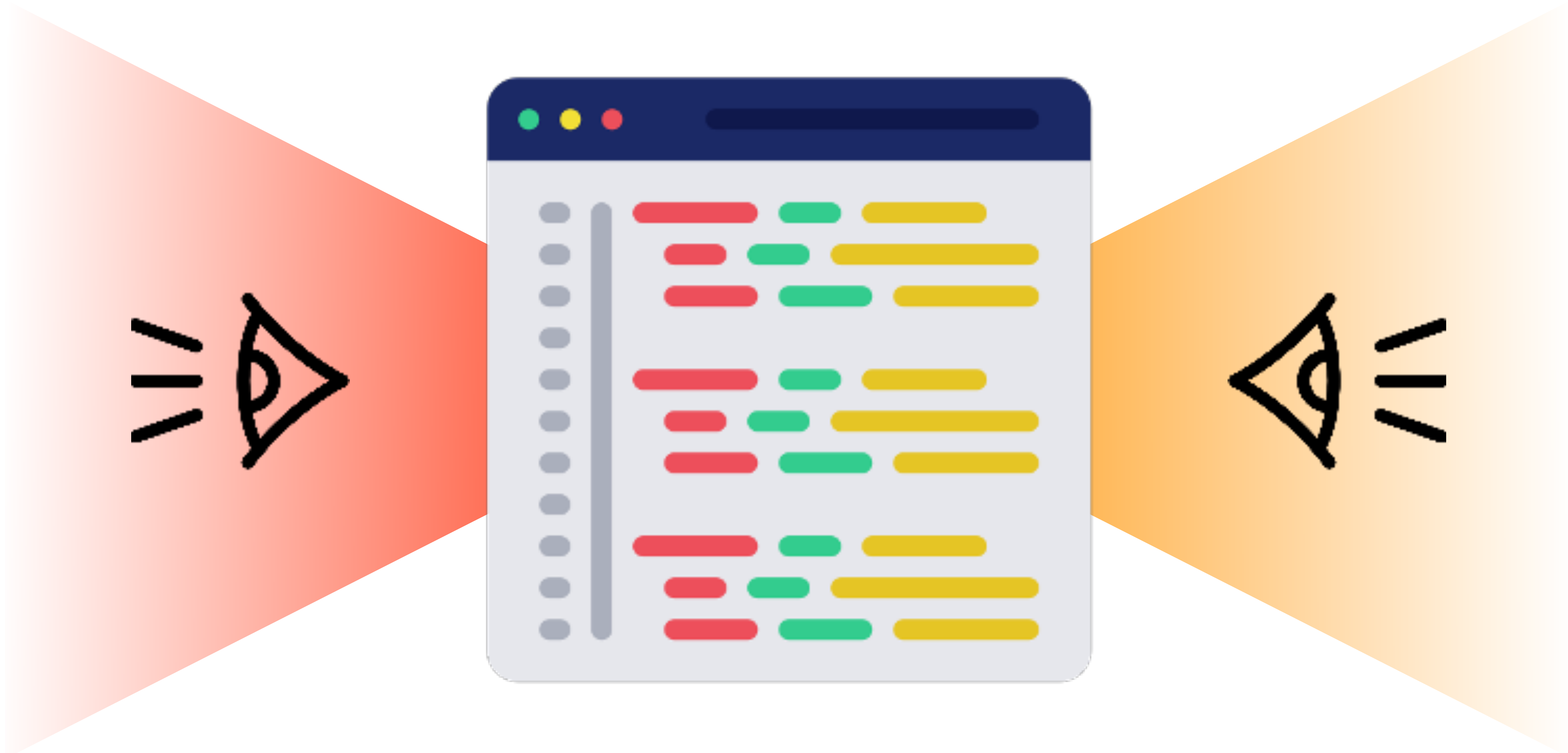
Coverage Increase

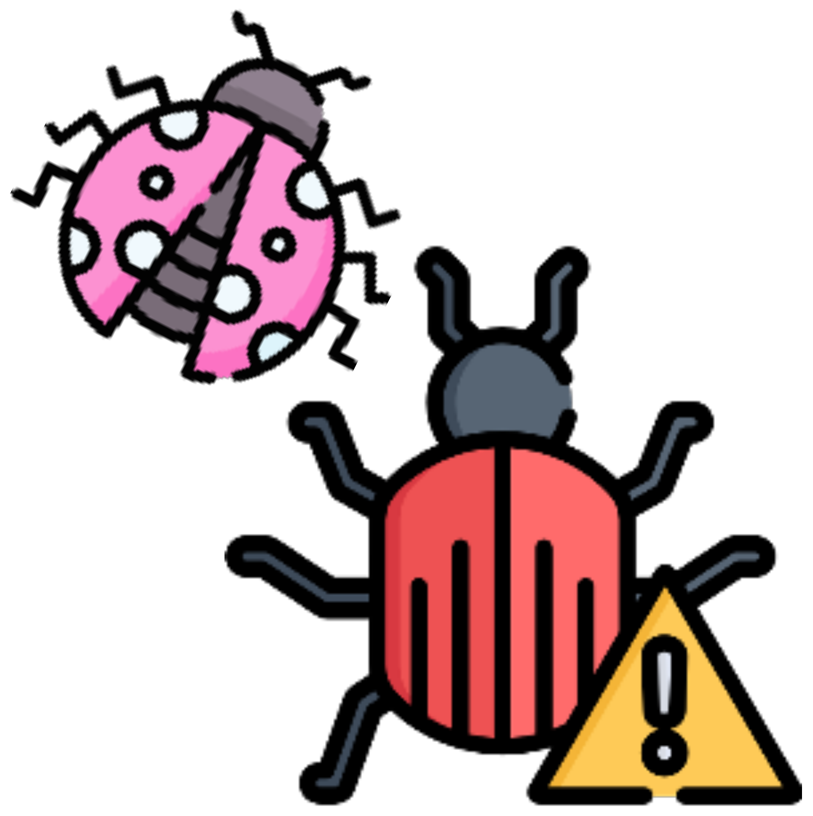
(lines or basic blocks)

Crashes / Anomalies

Q. So, Has this program been completely tested?







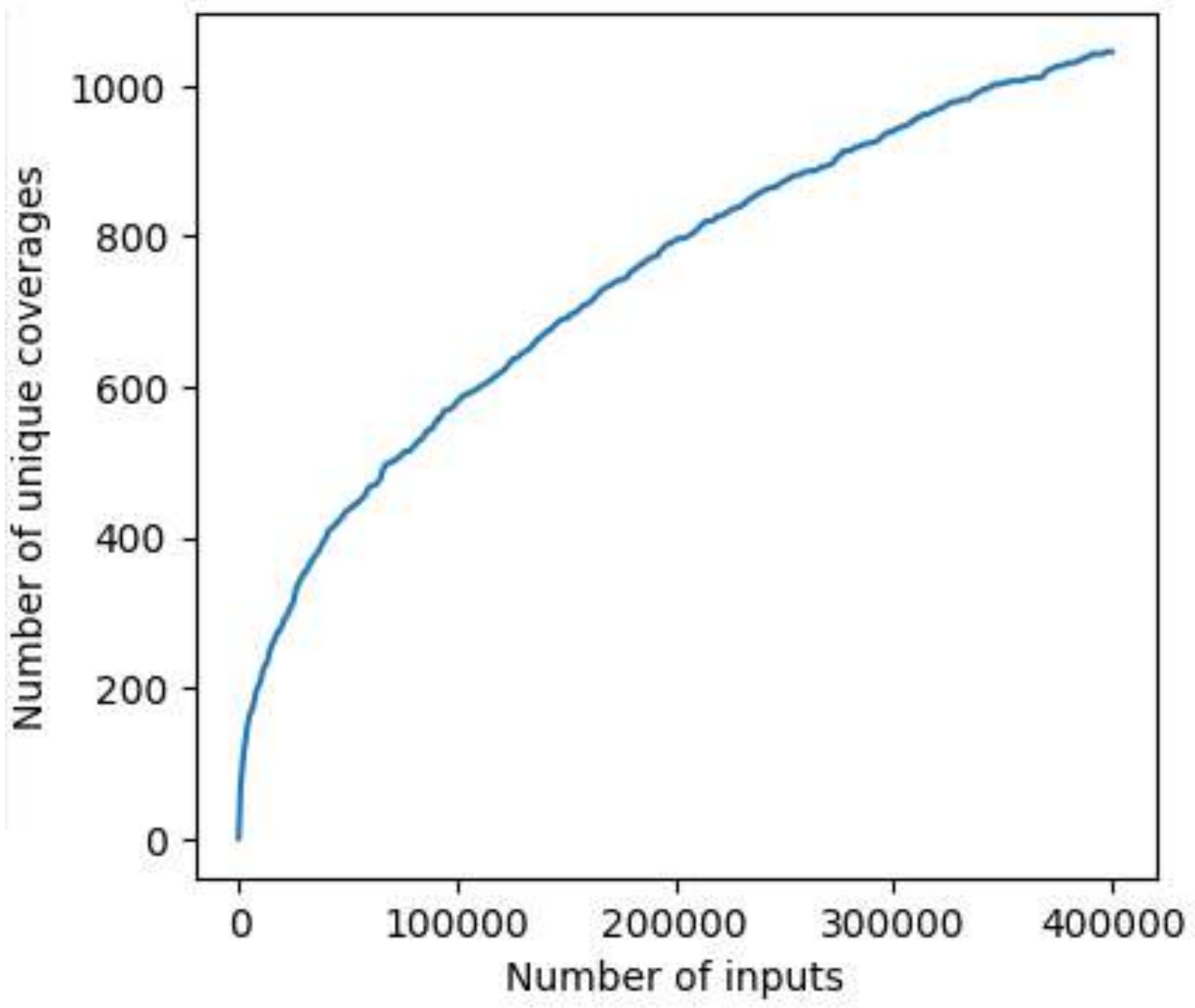
Program

Coverage Increase

(lines or basic blocks)

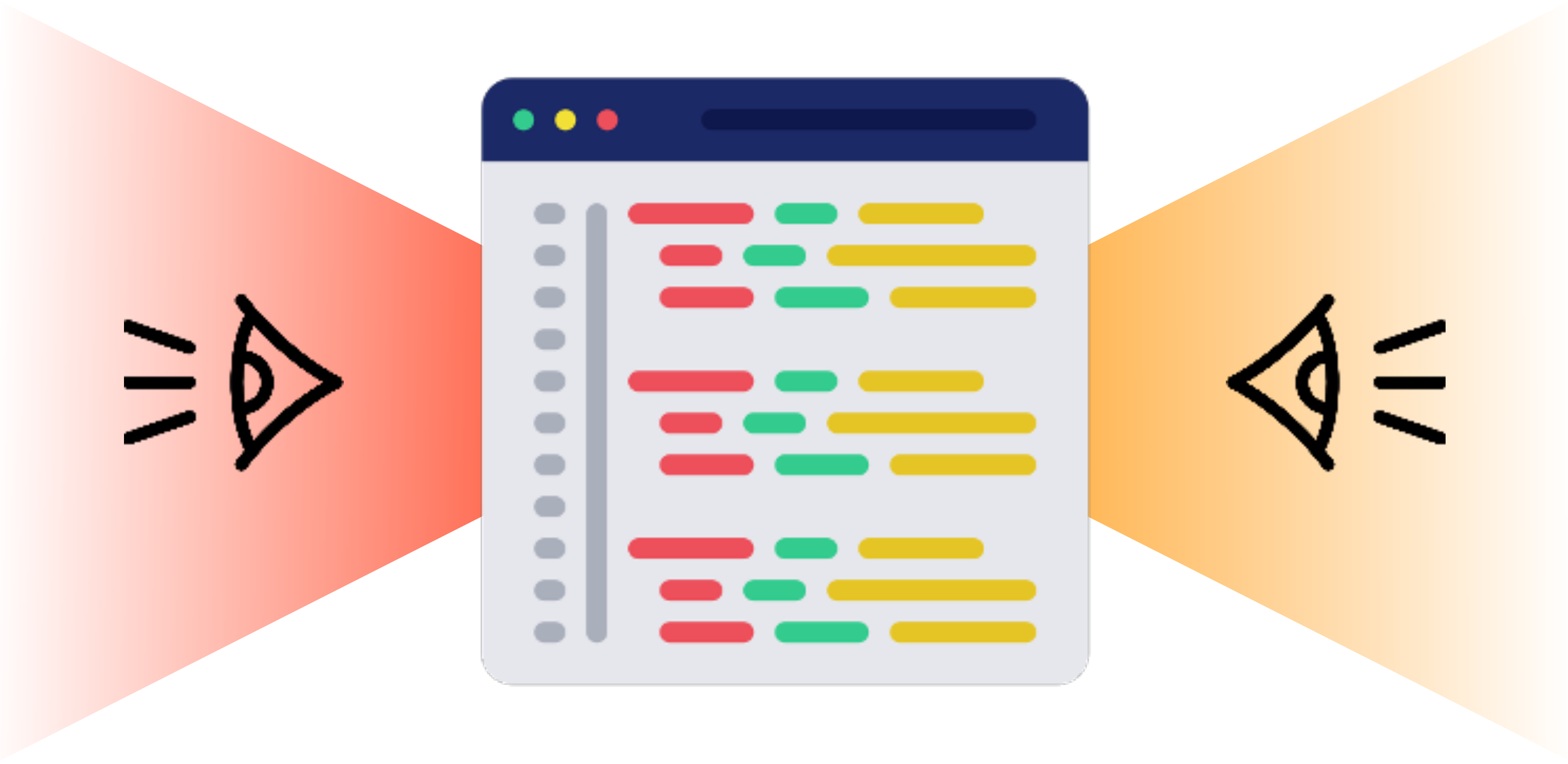
Crashes / Anomalies

Q. So, Has this program been completely tested? A. No

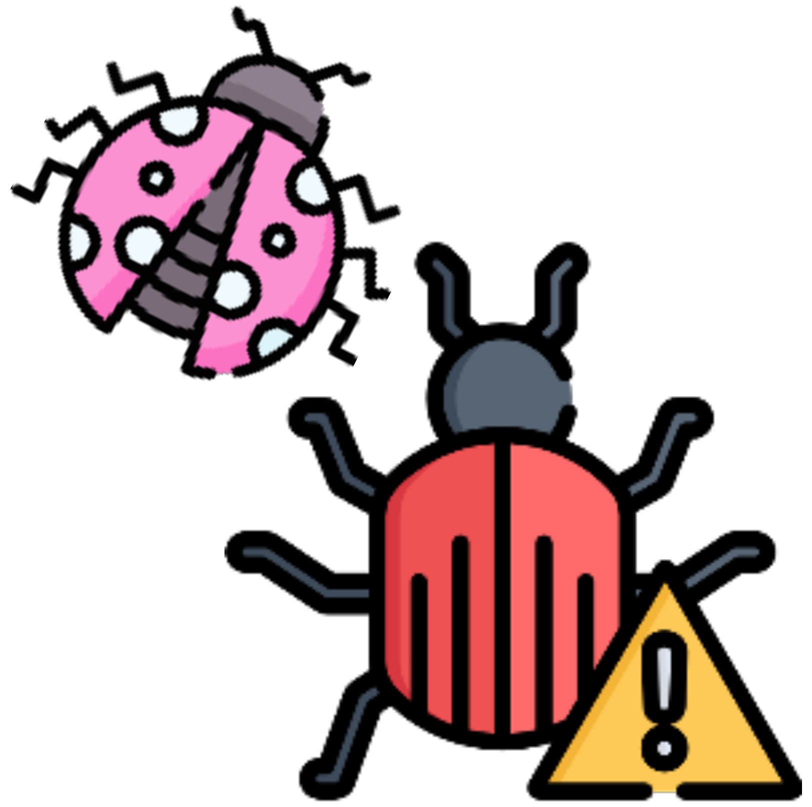


Coverage Increase

(lines or basic blocks)

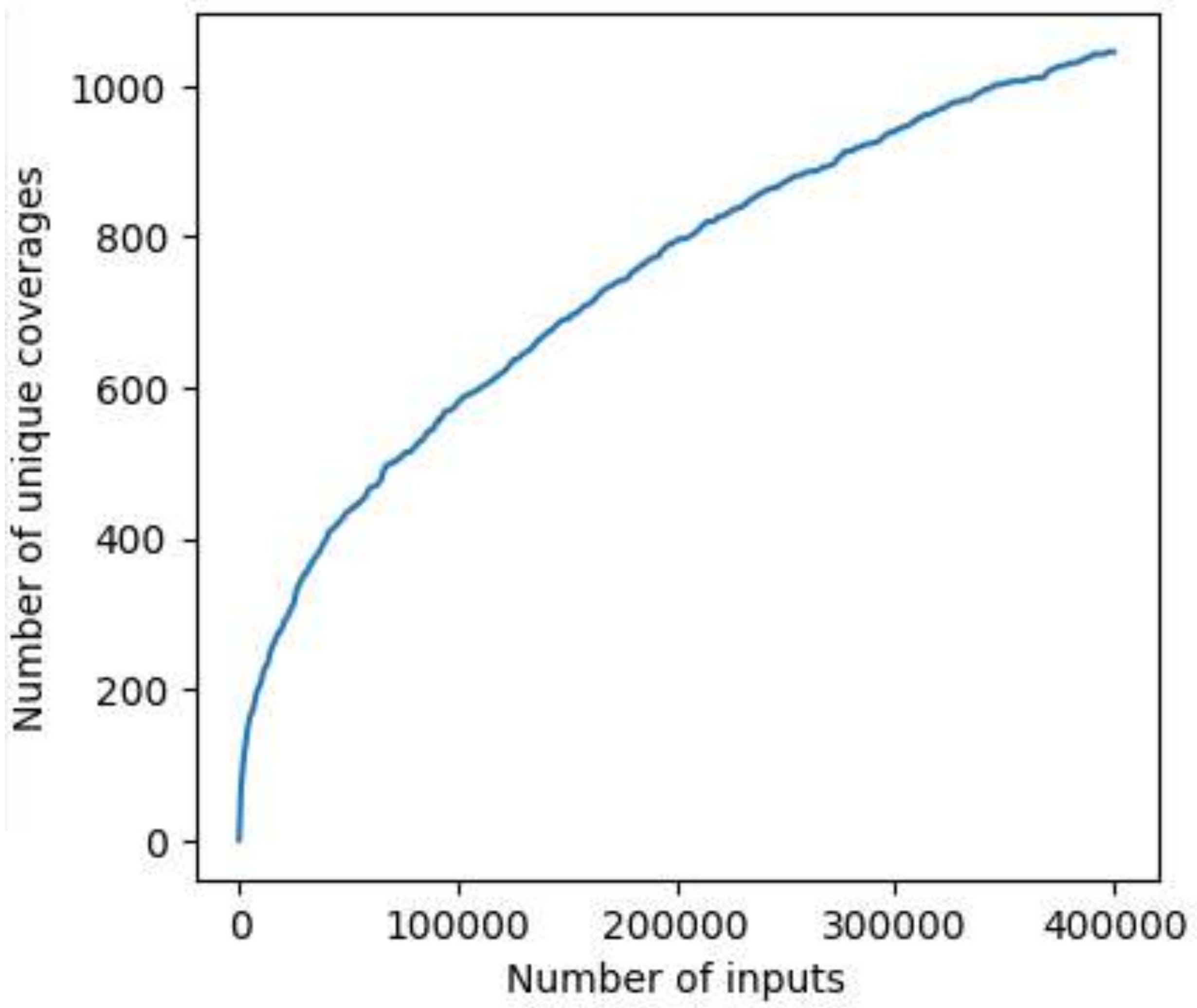


Program



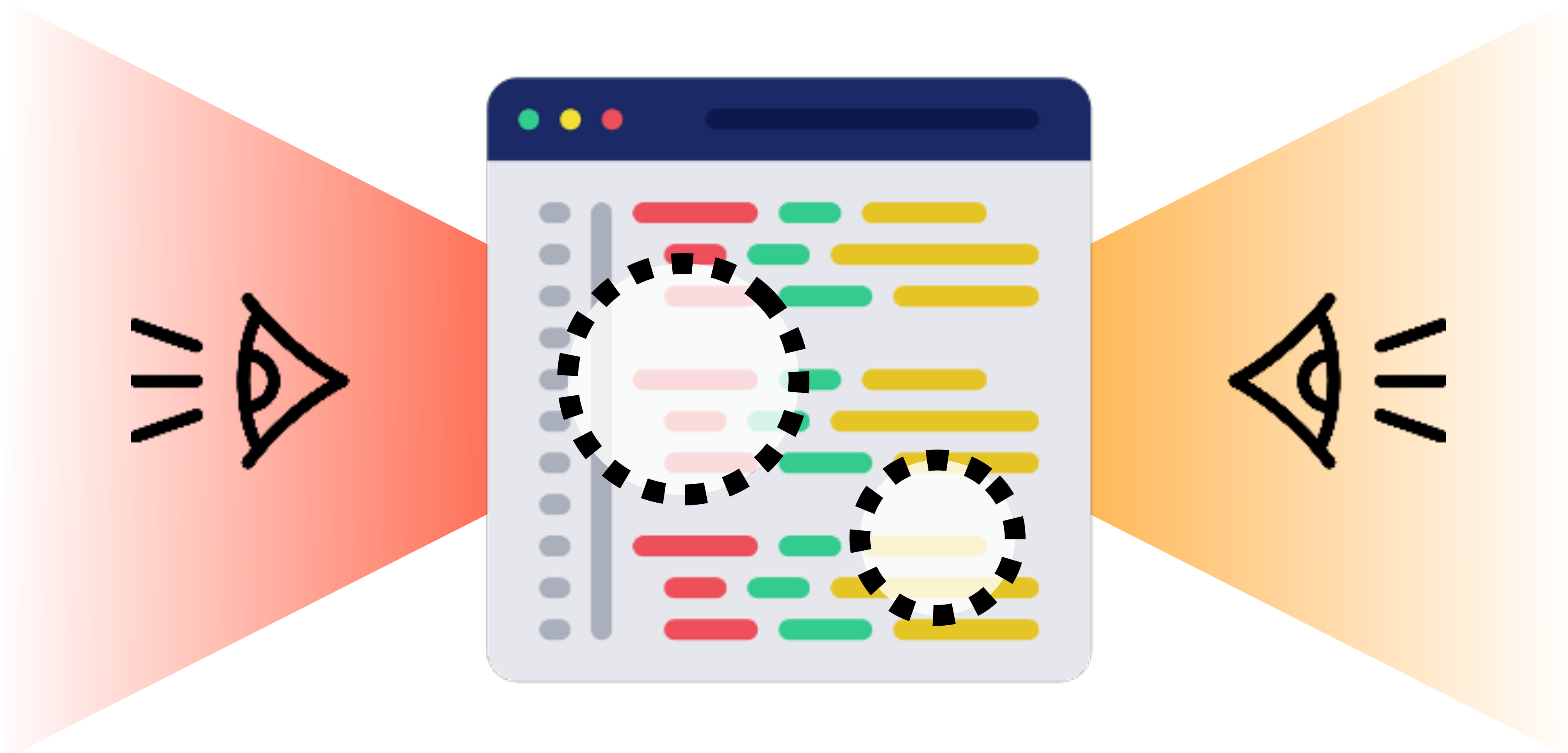
Crashes / Anomalies

Q. So, Has this program been completely tested? A. No

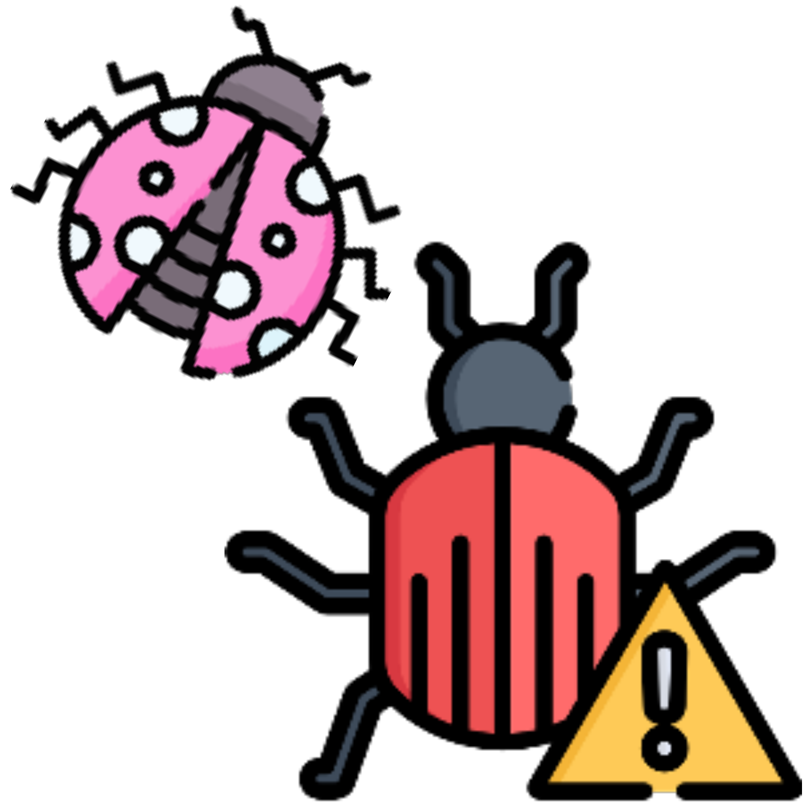


Coverage Increase

(lines or basic blocks)



Unseen Behavior



Crashes / Anomalies

i.e., based on
program executions

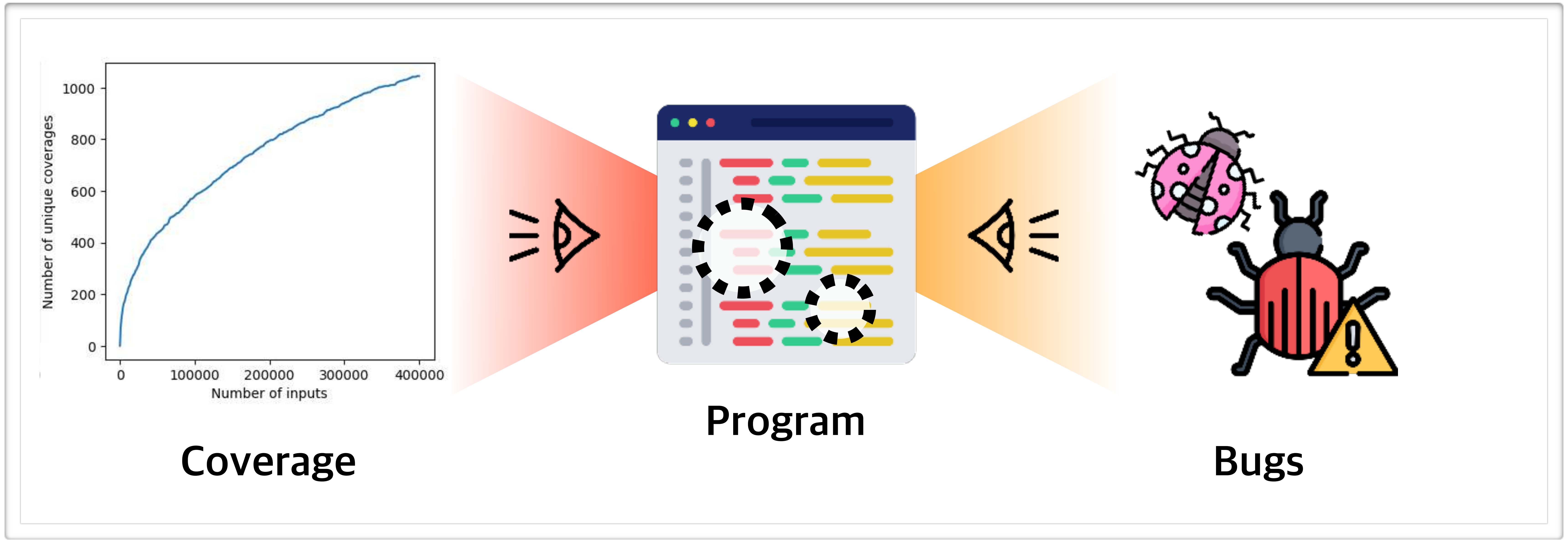
[The Fundamental Problem of Software Testing]

"It is always incomplete."

[The Fundamental Problem of Software Testing]

*"There is always **unseen**."*

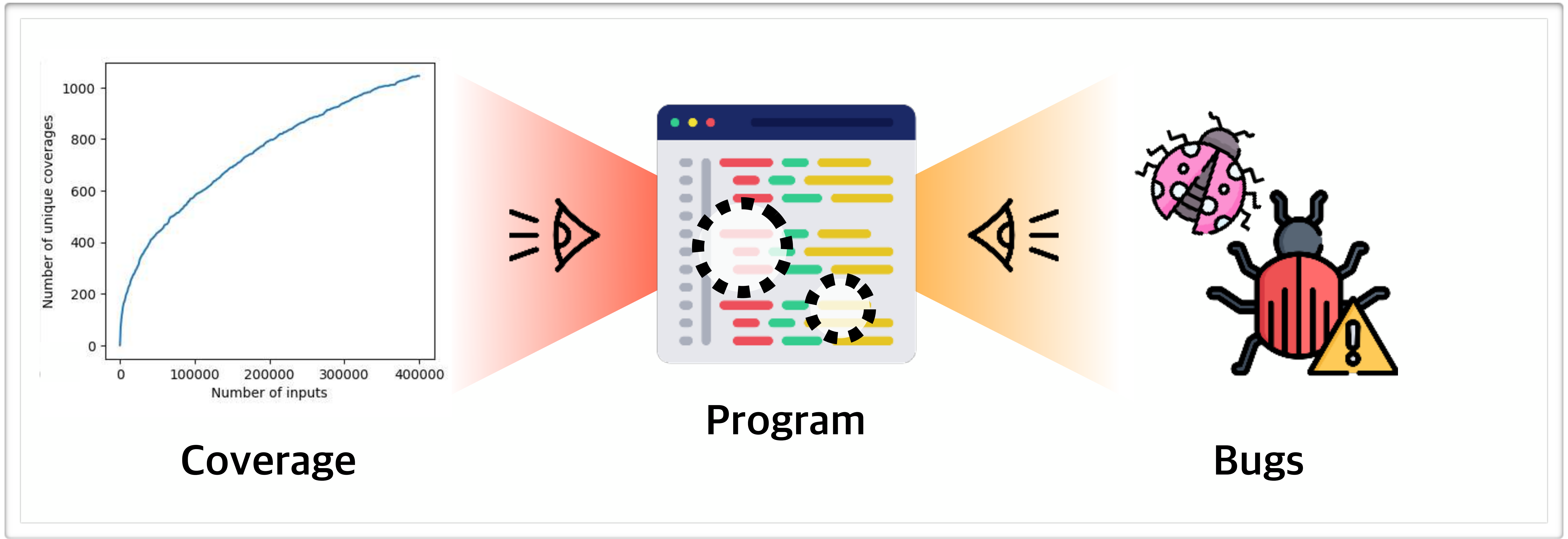
In this talk,



Given only the current status/result of the software testing, we want to know

"How secure is this program?"

Let's think about



Q. **What kind of questions** would help us to know
how much the program has been tested with software testing?

Questions about the unseen

**Red : semantic meaning*

**Black : concrete task*

Questions about the unseen

Red : semantic meaning
Black : concrete task

***How much of the behavior** have we tested/missed in this program?*

*What is the **probability** of observing **a new coverage or a new bug**?*

Questions about the unseen

**Red : semantic meaning*
**Black : concrete task*

***How much of the behavior** have we tested/missed in this program?*

*What is the **probability** of observing **a new coverage or a new bug**?*

How many unobserved vulnerabilities are remaining?

*What is the **maximum coverage** we can achieve?*

Questions about the unseen

**Red : semantic meaning*
**Black : concrete task*

***How much of the behavior** have we tested/missed in this program?*

*What is the **probability** of observing **a new coverage or a new bug**?*

How many unobserved vulnerabilities are remaining?

*What is the **maximum coverage** we can achieve?*

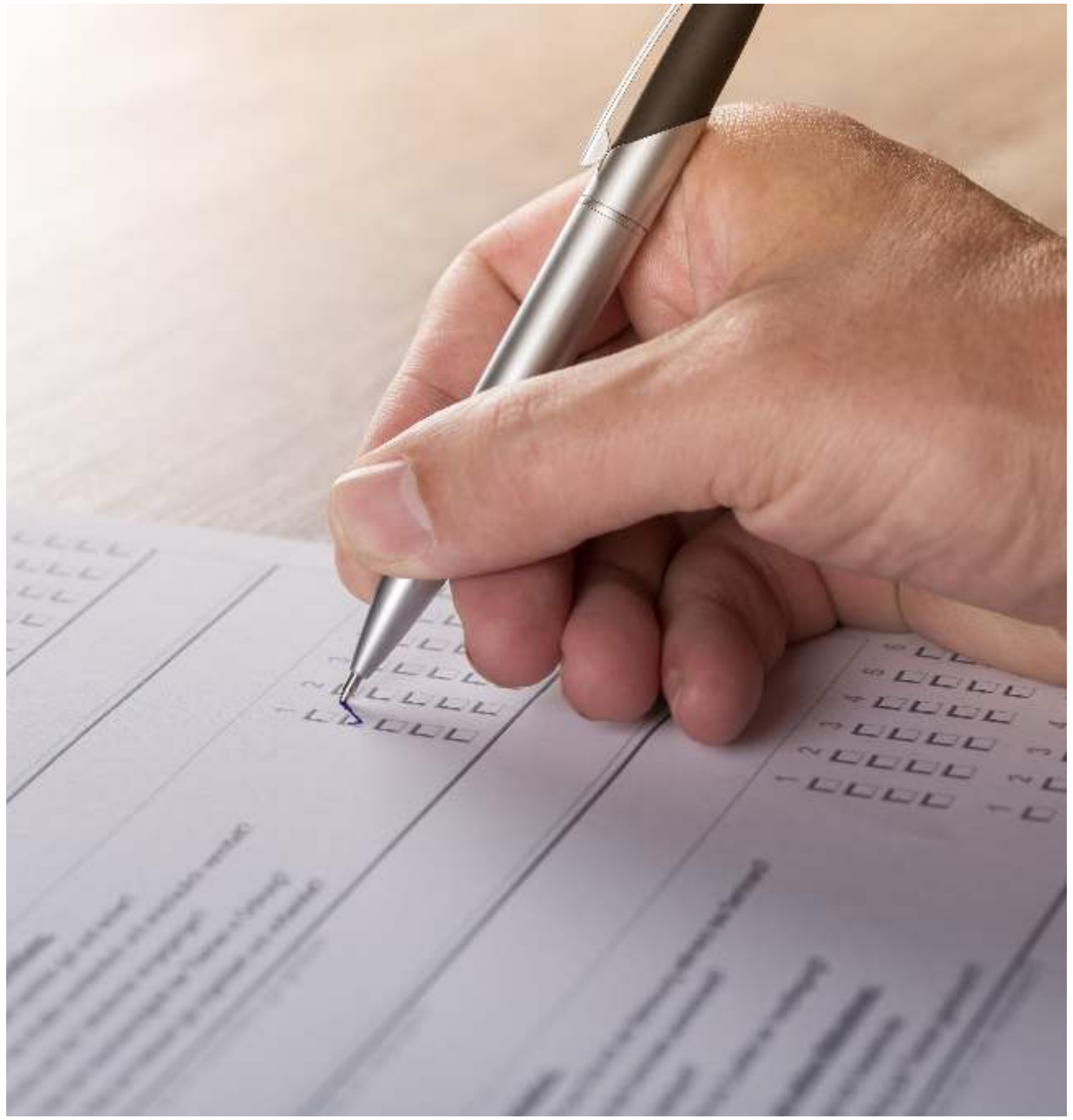
Should we keep running the testing/fuzzing?

*How much more can I achieve **if I spend** X **more time** here?*

How can we answer questions about the unseen?

How can we answer questions about the unseen?

Ecology



Social Science

Free image by d_alexander33 and andibreit from pixabay

Statistics!

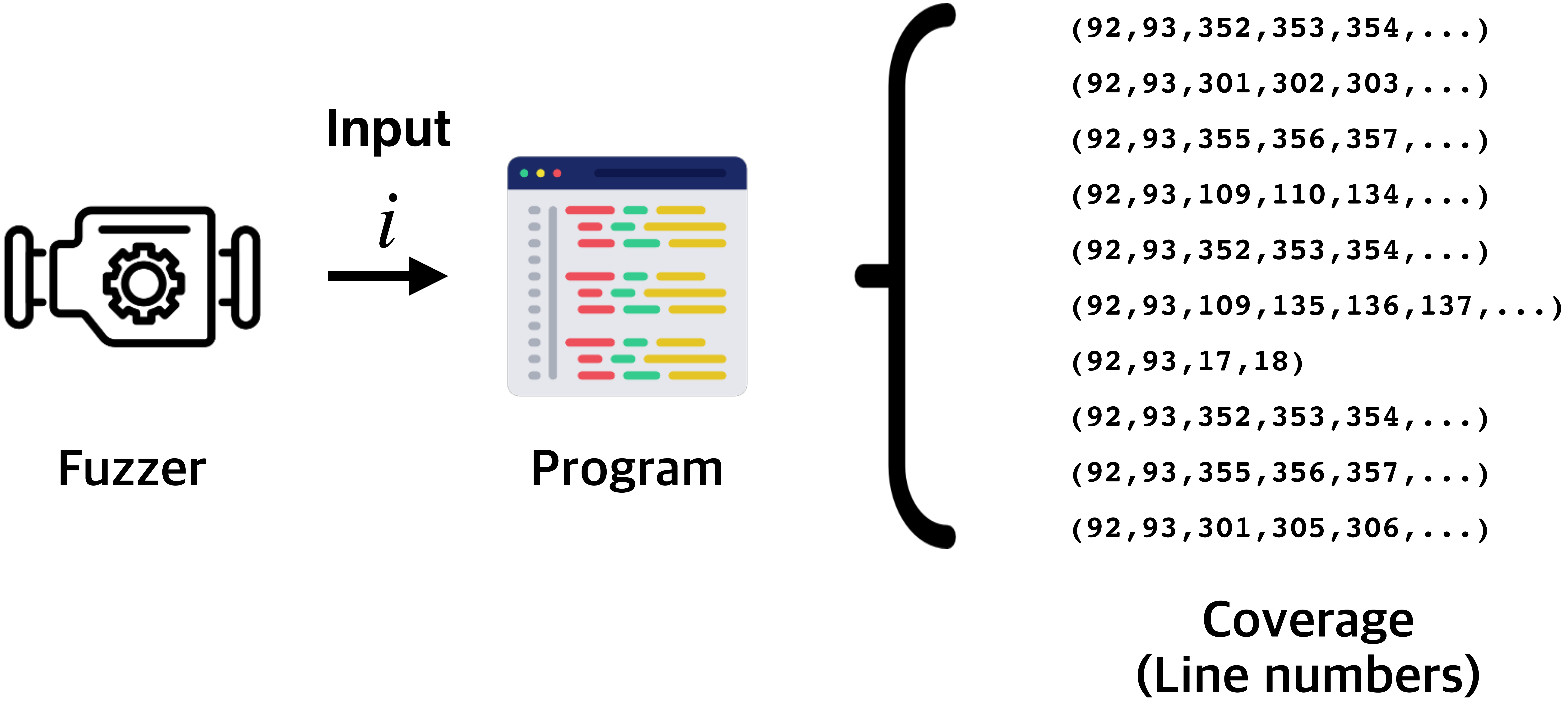
```
def f(x0, x1) {
  if (x0 + 5*x1 - 9 < 0) return;
  if (x0 + x1 -5 > 0) return;
  if (-x0 + 3x1 - 7 > 0) return;
  if (x0 > 0) return;
  assert False
}
f(input() % 5, input() % 5)
```

var phone = document.
var username = document.
var password = document.
var cpassword = document.
var firstname=document.
var firstname=document.
var firstname=document.
){
var firstname=document.
var firstname=document.
var firstname=document.('');if(
var firstname=document.('');if(
var firstname=document.('');if(
var firstname=document.
(!@#$%^&*()*+=~`) Not allowed")){
var firstname=document.
var firstname=document.
Password (!@#$%^&*()*+=~`) Not allowed
var firstname=document.
var firstname=document.
Username(!@#$%^&*()*+=~`) Not allowed


var user_id
var phone
var username
var password
var cpassword
187 var firstname=document.getElement
188 var firstname=document.getElement
189 var firstname=document.
190 var firstname=document.
191 var firstname=document.('');if(isComputer
192 var firstname=document.('');if(isComputer
193 var firstname=document.('');if(isComputer
194 var firstname=document.
(!@#$%^&*()*+=~`) Not allowed
195 var firstname=document.
196 var firstname=document.
Password (!@#$%^&*()*+=~`) Not allowed
197 var firstname=document.
198 var firstname=document.
Username(!@#$%^&*()*+=~`)

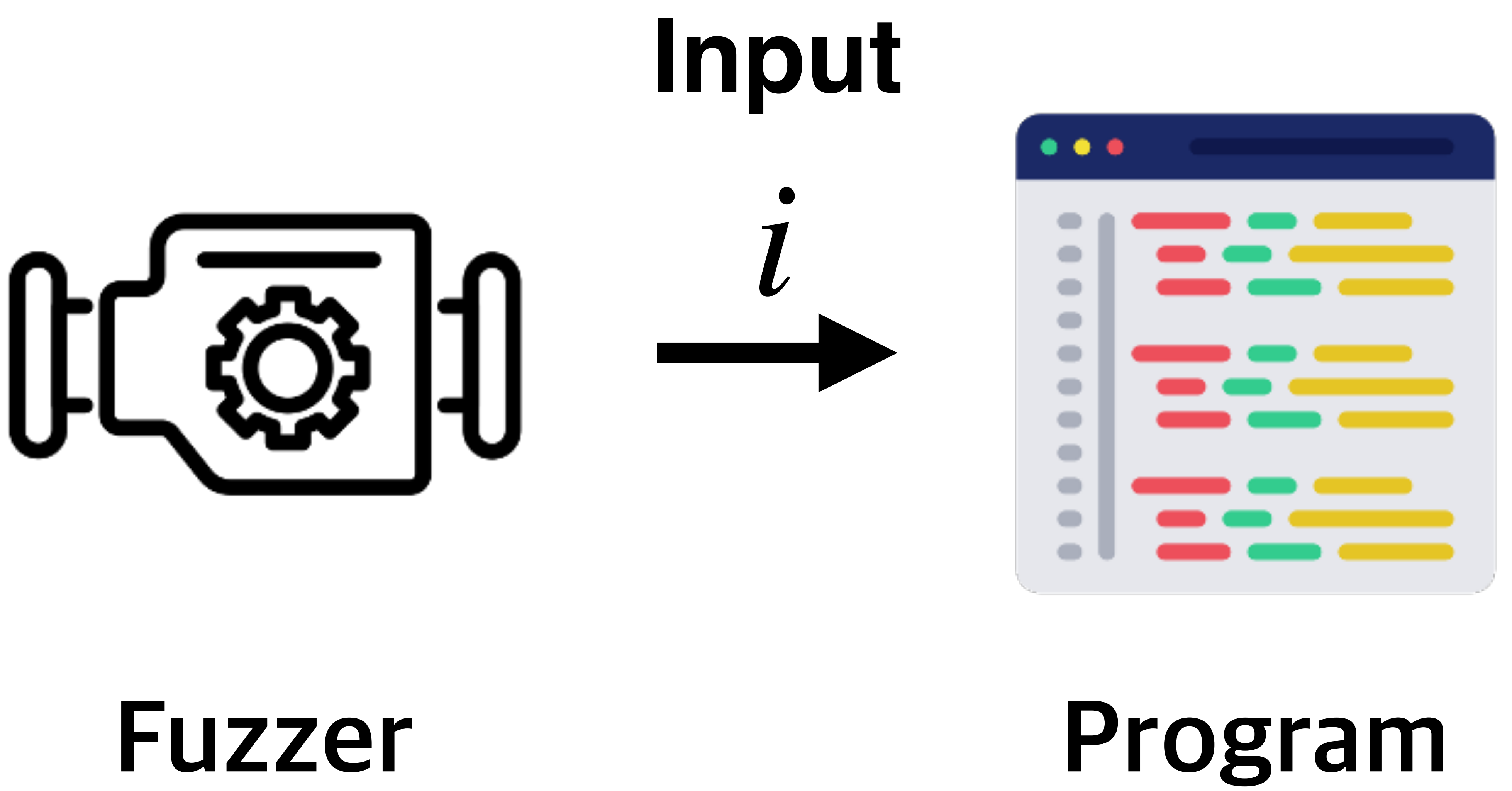
$\approx$

Software Testing $\Longrightarrow$ *Sampling Process*



Software Testing $\Rightarrow$ *Sampling Process*

* Abundantly observed coverage

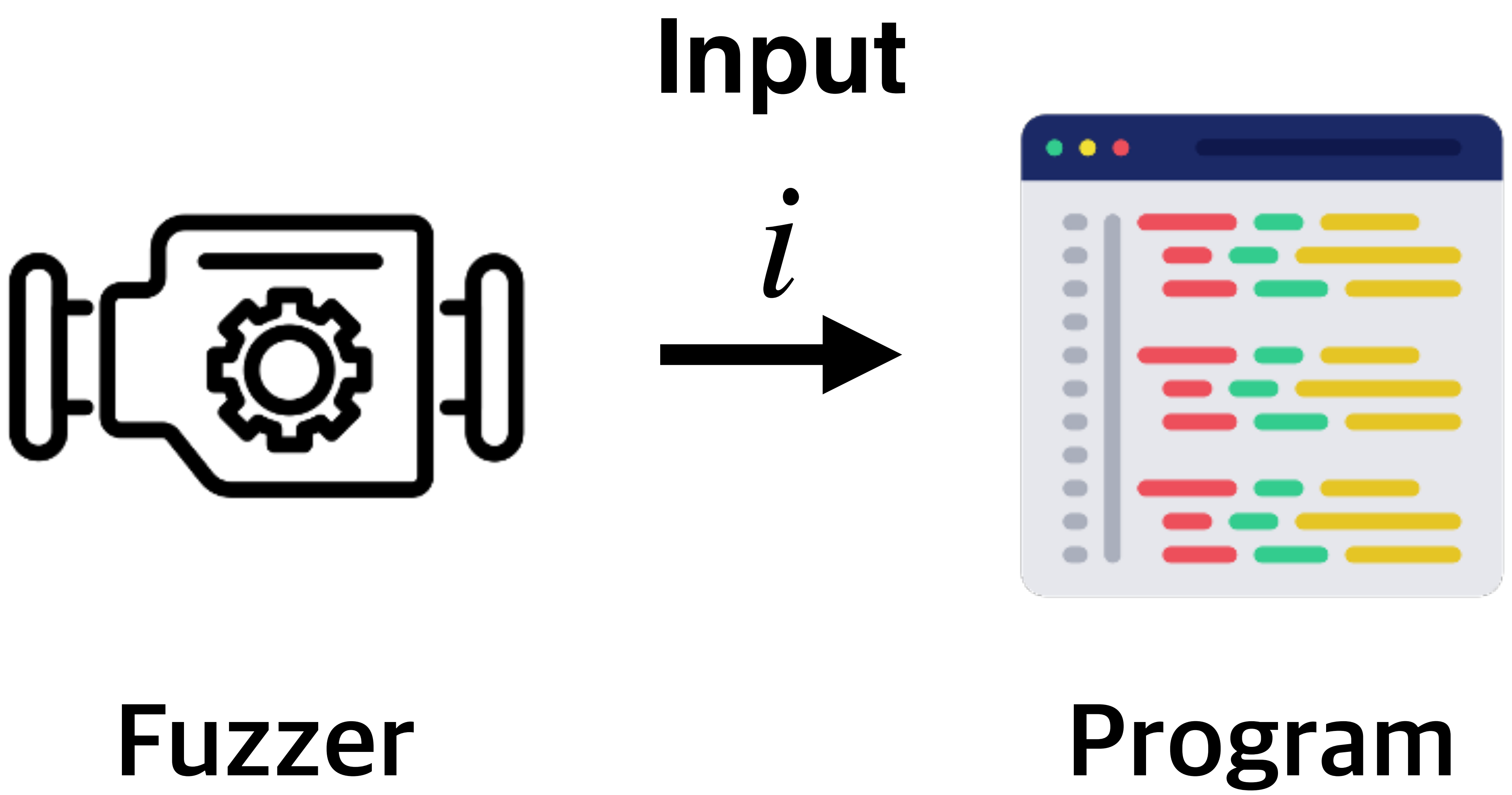


(92,93,352,353,354,...)
(92,93,301,302,303,...)
(92,93,355,356,357,...)
(92,93,109,110,134,...)
(92,93,352,353,354,...)
(92,93,109,135,136,137,...)
(92,93,17,18)
(92,93,352,353,354,...)
(92,93,355,356,357,...)
(92,93,301,305,306,...)

Coverage
(Line numbers)

Software Testing $\Rightarrow$ *Sampling Process*

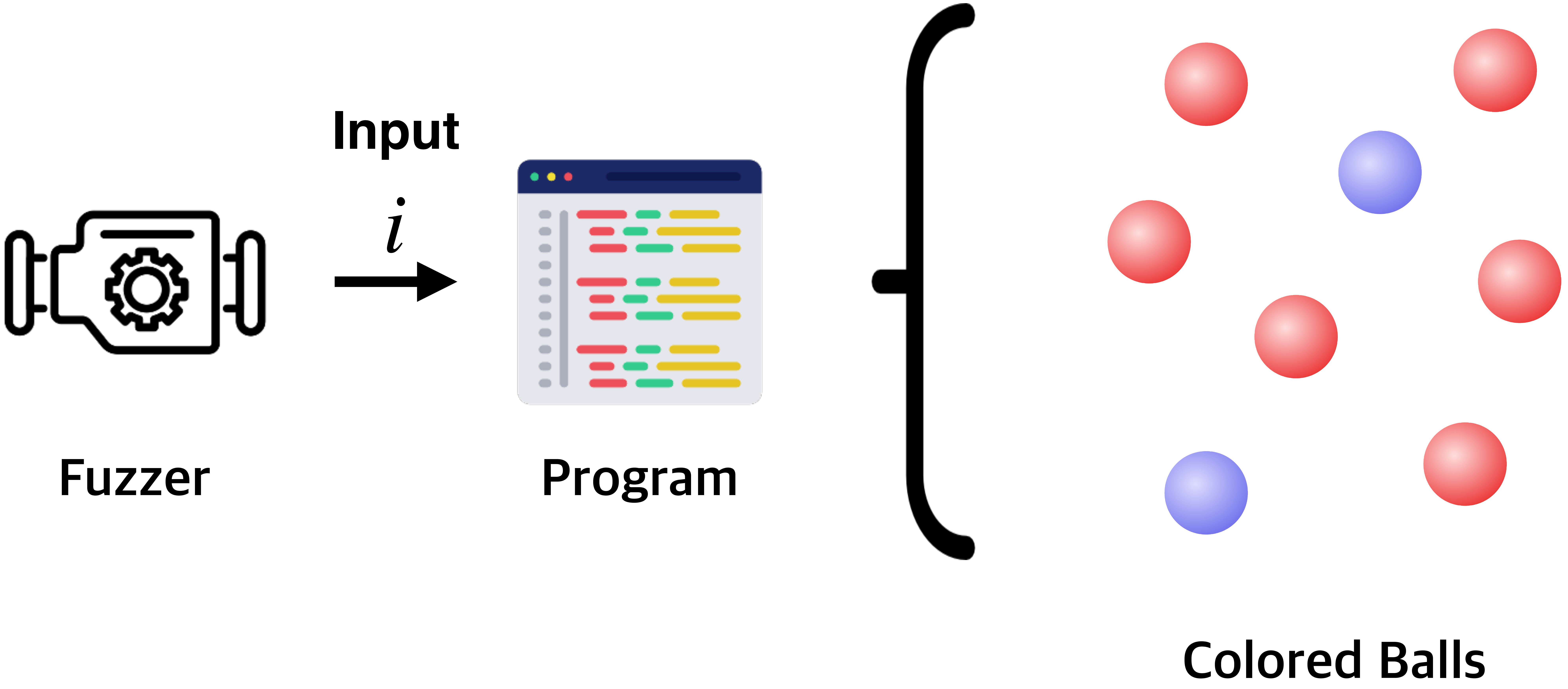
* Abundantly observed coverage
* Rarely observed coverage



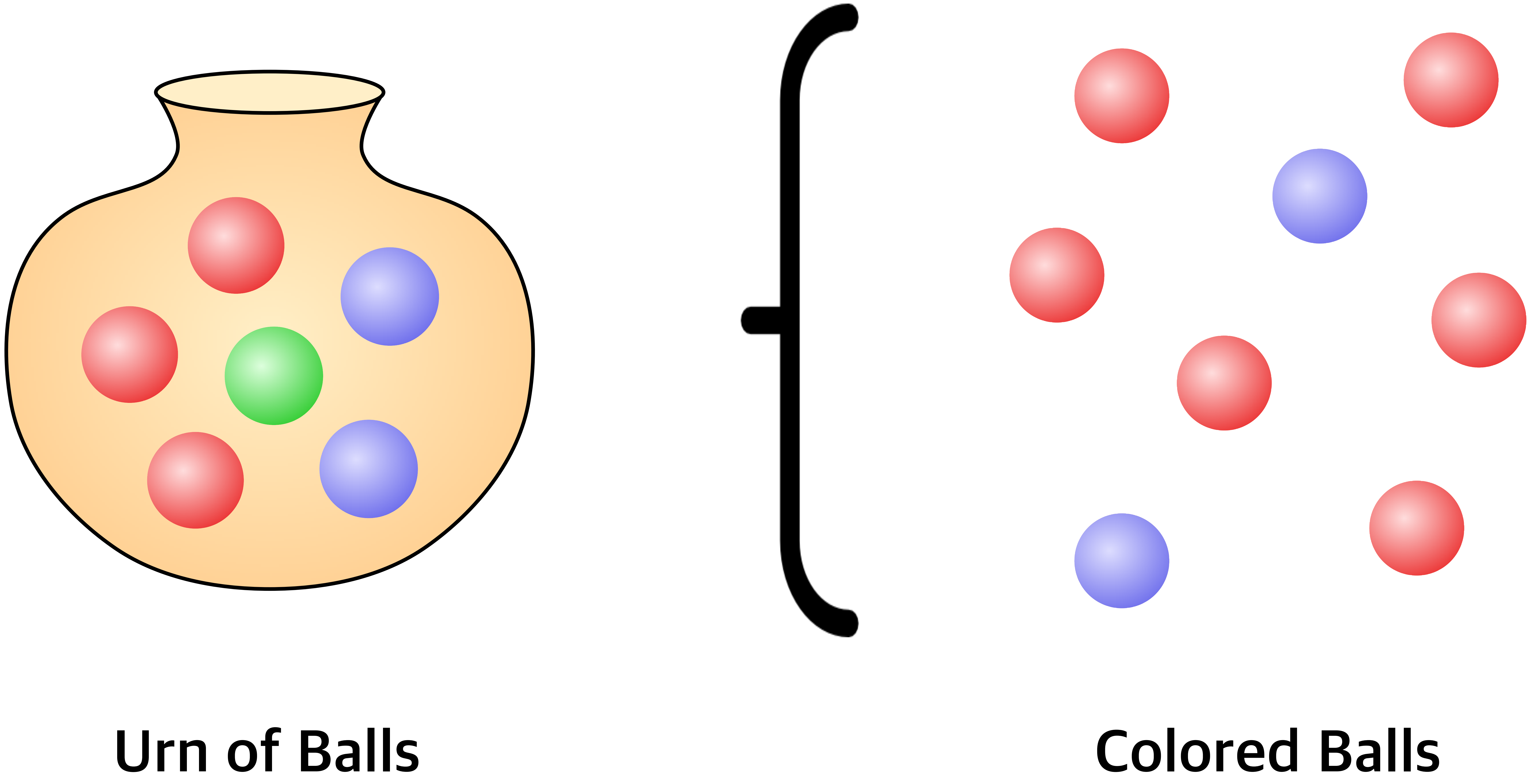
(92,93,352,353,354,...)
(92,93,301,302,303,...)
(92,93,355,356,357,...)
(92,93,109,110,134,...)
(92,93,352,353,354,...)
(92,93,109,135,136,137,...)
(92,93,17,18)
(92,93,352,353,354,...)
(92,93,355,356,357,...)
(92,93,301,305,306,...)

Coverage
(Line numbers)

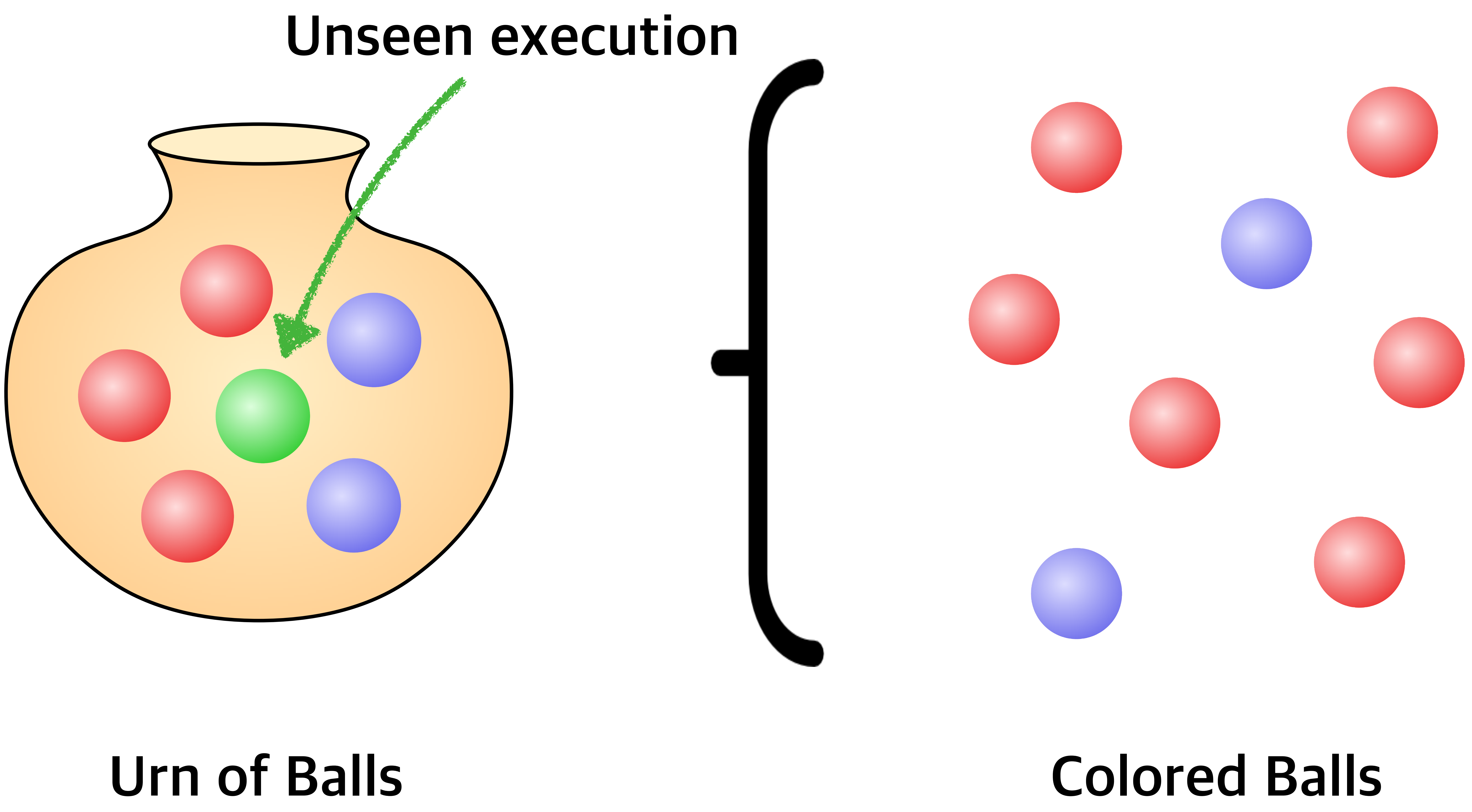
Software Testing $\Longrightarrow$ *Sampling Process*



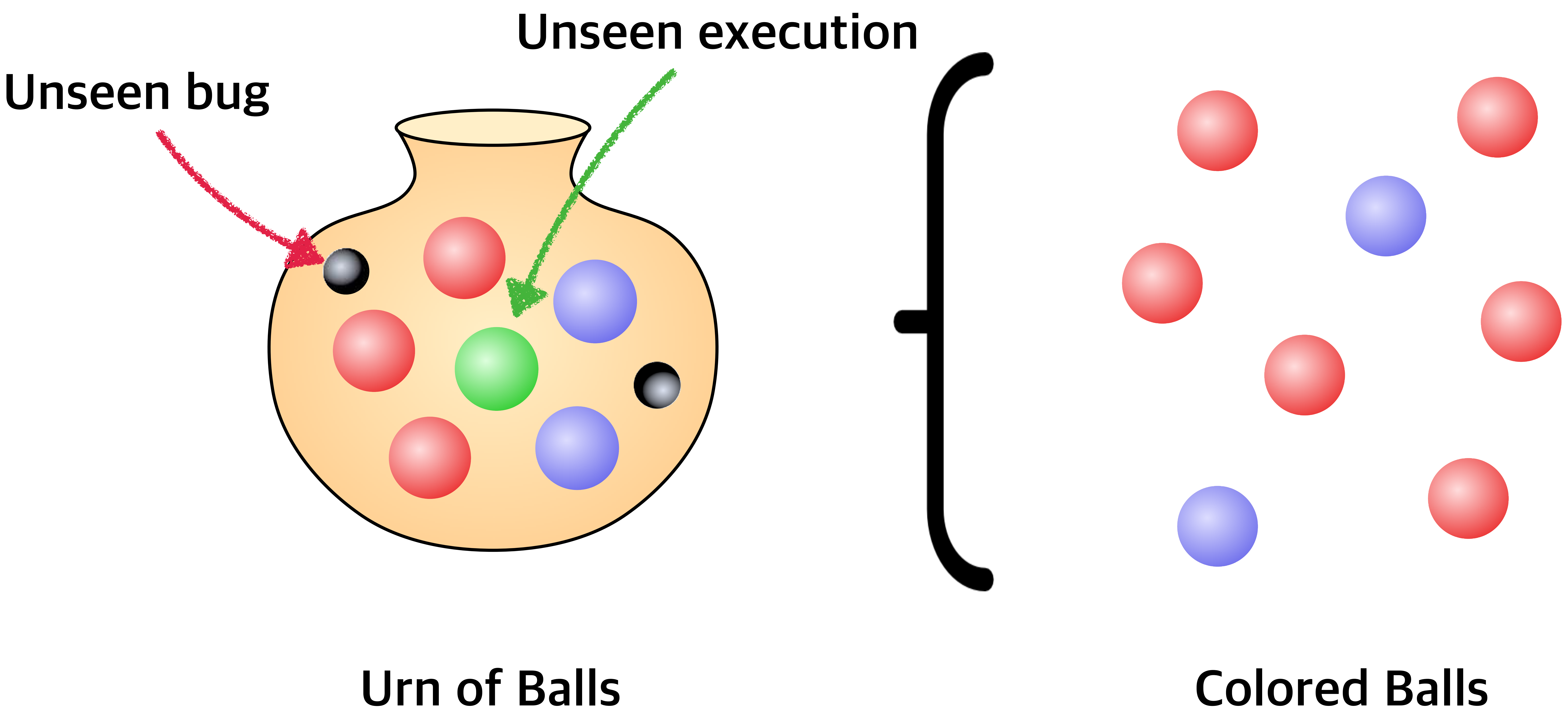
Software Testing ⇒ Sampling Process



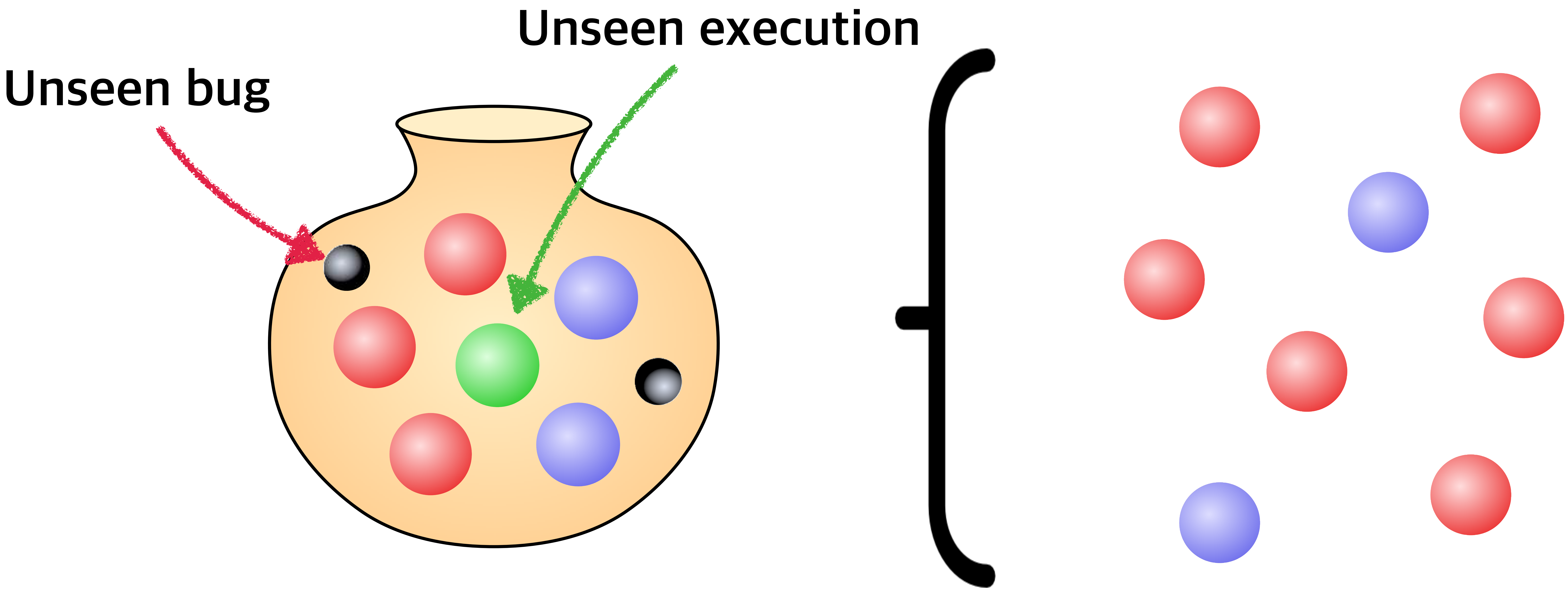
Software Testing $\Rightarrow$ *Sampling Process*



Software Testing ⇒ Sampling Process



Software Testing $\Rightarrow$ *Sampling Process*



Residual Risk of Testing $\approx$ Remaining Unseen Colors

Questions about the unseen in Software Testing

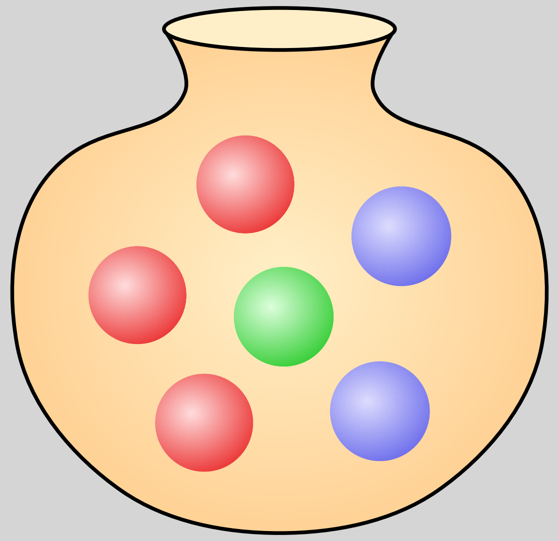
*What is the **probability** of observing **a new coverage or a new bug**?*

*What is the **maximum coverage** we can achieve?*

*How much more can I achieve **if I spend** X **more time** here?*

Questions about the unseen in an Urn filled with Balls

*What is the **probability** of observing **a new color ball** in the next sample?*

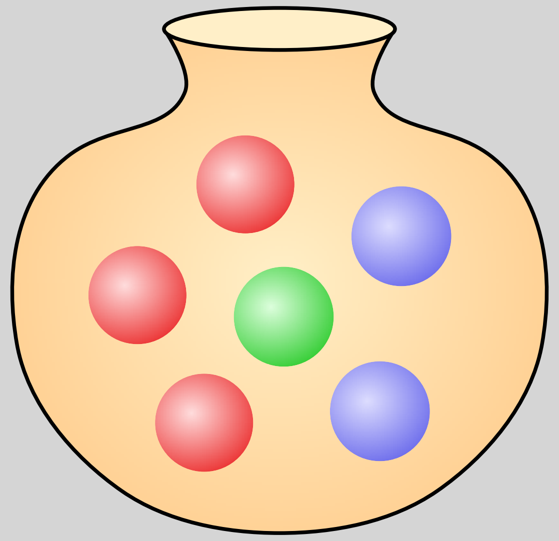


*What is the **maximum coverage** we can achieve?*

*How much more can I achieve **if I spend** X **more time** here?*

Questions about the unseen in an Urn filled with Balls

*What is the **probability** of observing **a new color ball** in the next sample?*

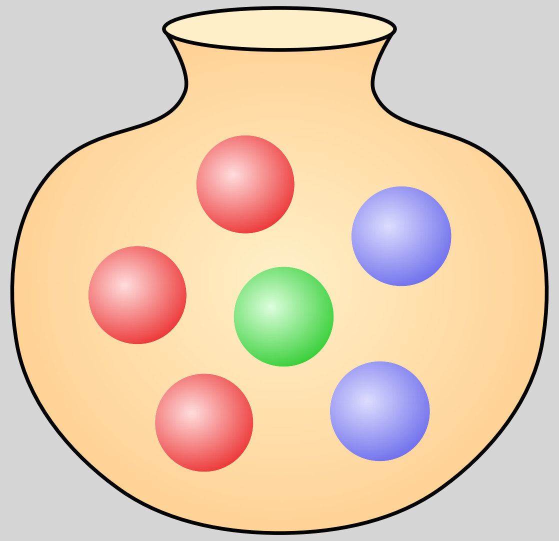


***How many colors are remaining** in the urn?*

*How much more can I achieve **if I spend** X **more time** here?*

Questions about the unseen in an Urn filled with Balls

What is the ***probability*** *of observing* ***a new color ball*** *in the next sample?*



How many colors are remaining *in the urn?*

How many new colors can I see more ***if I sample*** *X* ***more balls*** *from the urn?*

Questions about the unseen in an Urn filled with Balls

*What is the **probability** of observing **a new color ball** in the next sample?*

Statistics!

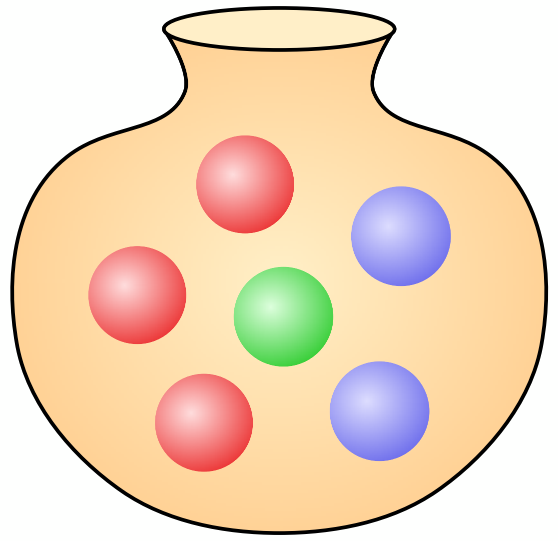
***How many colors are remaining** in the urn?*

*How many new colors can I see more **if I sample** X **more balls** from the urn?*

Statistical notion of the unseen in an Urn filled with Balls

Missing Mass

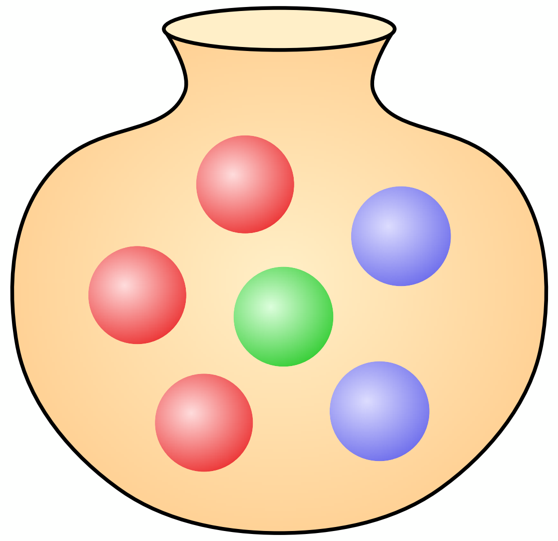
*What is the **probability** of observing **a new color ball** in the next sample?*



***How many colors are remaining** in the urn?*

*How many new colors can I see more **if I sample** X **more balls** from the urn?*

Statistical notion of the unseen in an Urn filled with Balls



Missing Mass

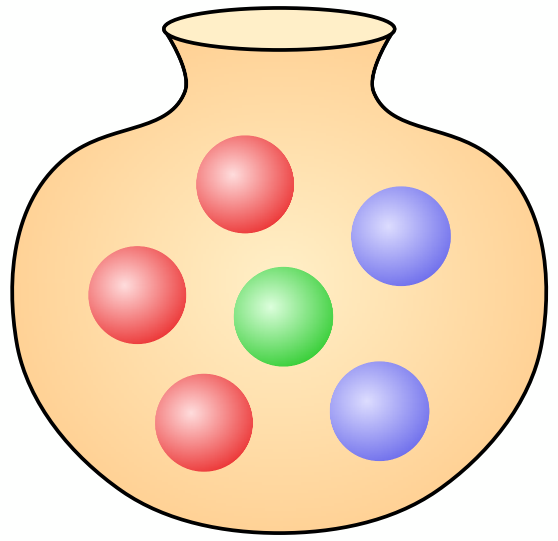
*What is the **probability** of observing **a new color ball** in the next sample?*

Species Richness

***How many colors are remaining** in the urn?*

*How many new colors can I see more **if I sample** X **more balls** from the urn?*

Statistical notion of the unseen in an **Urn filled with Balls**



Missing Mass

What is the ***probability*** *of observing* ***a new color ball*** *in the next sample?*

Species Richness

How many colors are remaining *in the urn?*

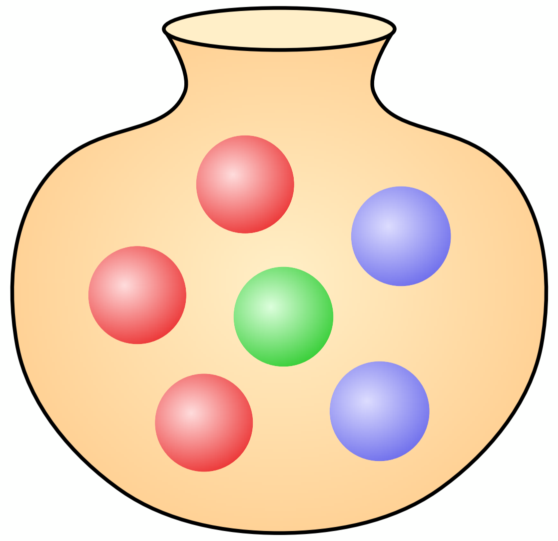
Extrapolation

How many new colors can I see more ***if I sample*** *X* ***more balls*** *from the urn?*

Estimators about the unseen in an Urn filled with Balls

Missing Mass

$$\frac{\Phi_1}{n}$$



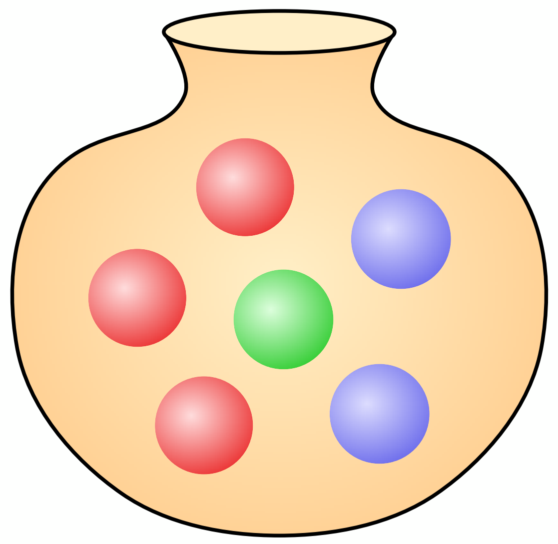
Species Richness

How many colors are remaining *in the urn?*

Extrapolation

How many new colors can I see more ***if I sample*** *X* ***more balls*** *from the urn?*

Estimators about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

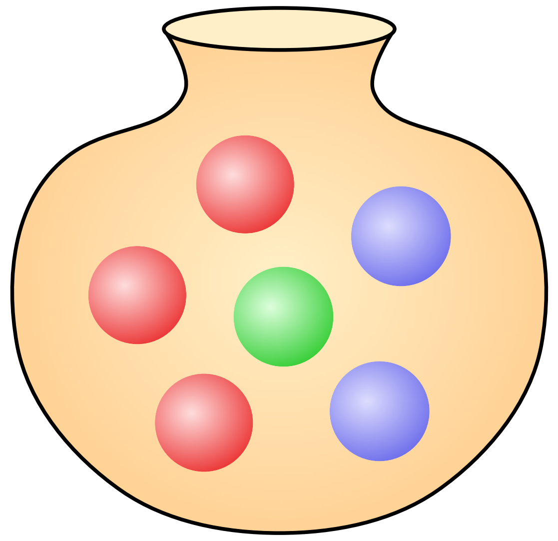
Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

How many new colors can I see more ***if I sample*** *X* ***more balls*** *from the urn?*

Estimators about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

Missing Mass

What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

Extrapolation

How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

Missing Mass

*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

*How much more can I achieve **if I spend** X **more time** here?*

*Present **advanced extensions** to adopt statistical methods for more **realistic testing scenarios**.*

2

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

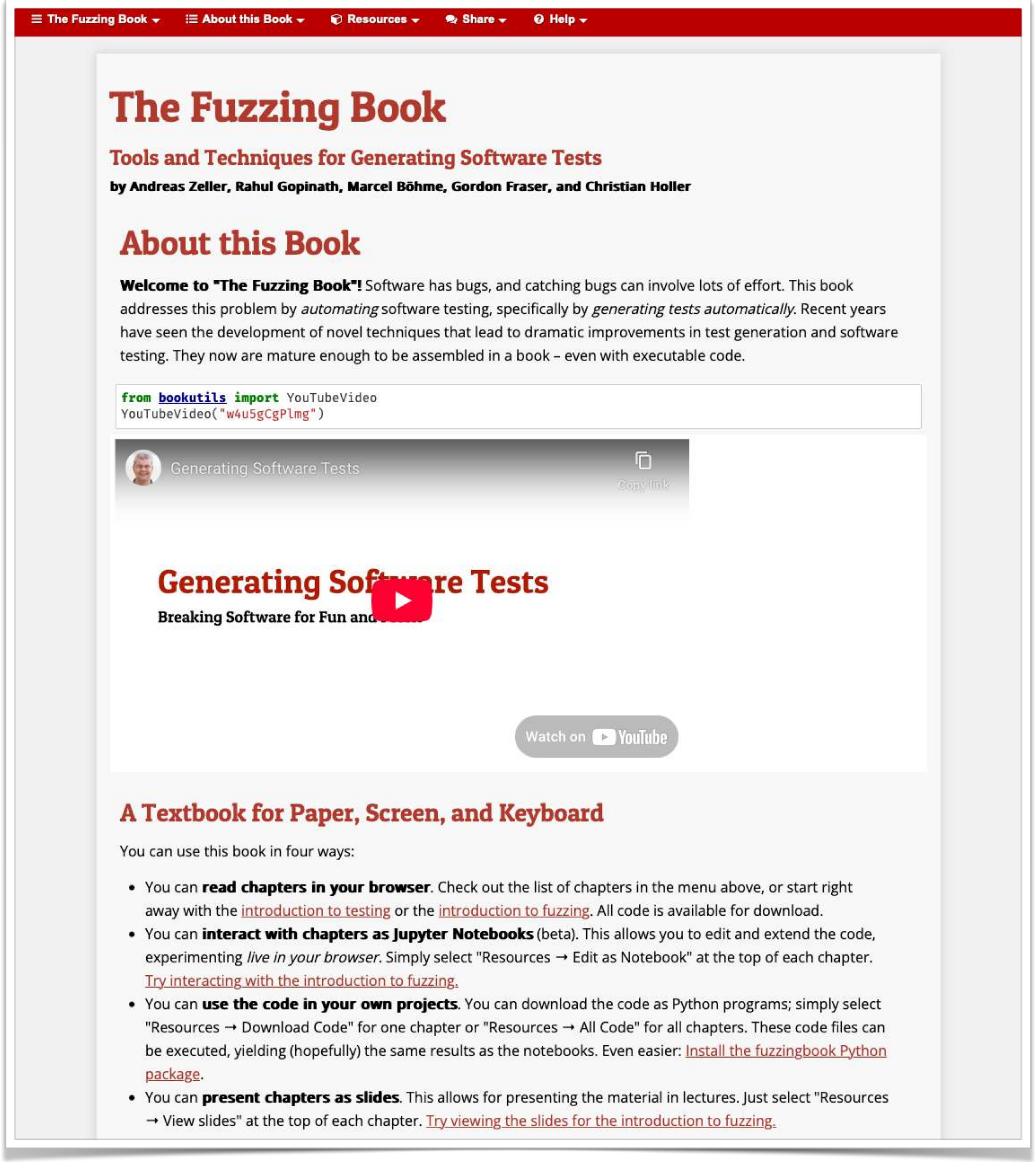
Missing Mass

What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

Extrapolation

How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

Hands-on-exercise with **Fuzzing Book**



☰ The Fuzzing Book ▾ ☷ About this Book ▾ Resources ▾ Share ▾ Help ▾

The Fuzzing Book

Tools and Techniques for Generating Software Tests

by Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler

About this Book

Welcome to "The Fuzzing Book"! Software has bugs, and catching bugs can involve lots of effort. This book addresses this problem by *automating* software testing, specifically by *generating tests automatically*. Recent years have seen the development of novel techniques that lead to dramatic improvements in test generation and software testing. They now are mature enough to be assembled in a book – even with executable code.

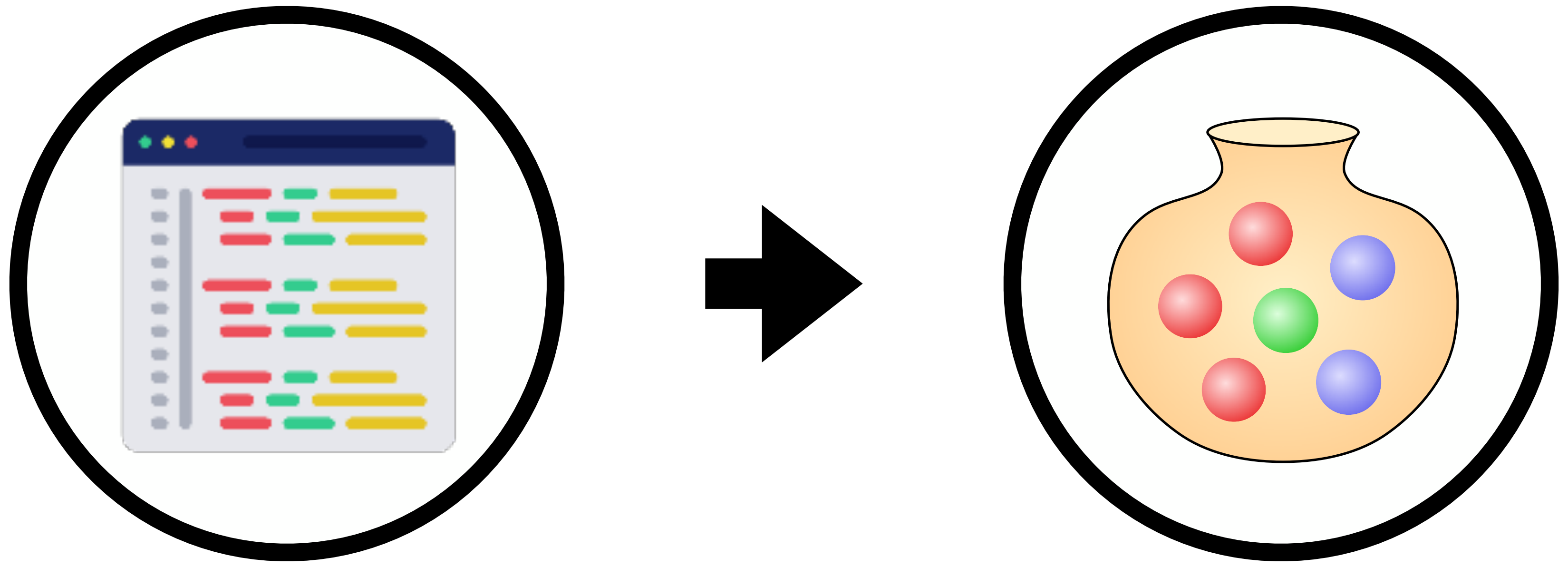
```
from bookutils import YouTubeVideo
YouTubeVideo("w4u5gCgPlmg")
```


A Textbook for Paper, Screen, and Keyboard

You can use this book in four ways:

- You can **read chapters in your browser**. Check out the list of chapters in the menu above, or start right away with the introduction to testing or the introduction to fuzzing. All code is available for download.
- You can **interact with chapters as Jupyter Notebooks** (beta). This allows you to edit and extend the code, experimenting *live in your browser*. Simply select "Resources → Edit as Notebook" at the top of each chapter. Try interacting with the introduction to fuzzing.
- You can **use the code in your own projects**. You can download the code as Python programs; simply select "Resources → Download Code" for one chapter or "Resources → All Code" for all chapters. These code files can be executed, yielding (hopefully) the same results as the notebooks. Even easier: Install the fuzzingbook Python package.
- You can **present chapters as slides**. This allows for presenting the material in lectures. Just select "Resources → View slides" at the top of each chapter. Try viewing the slides for the introduction to fuzzing.

0. Preparation



To the notebook.

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

Missing Mass

What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

Extrapolation

How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

Missing Mass

What is the probability of observing ~~a new coverage or a new bug~~?

a new color ball?

Solution:

Solution:

Good-Turing estimator

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

The estimation of the probability of our following sample is something that has <u>never been seen before</u>.

Solution:

Good-Turing estimator

colors only seen
once in samples

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

of *singleton colors

of samples

The estimation of the probability of our following sample is something that has <u>never been seen before</u>.

Solution:

Good-Turing estimator

Alan Turing

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

colors only seen once in samples

of *singleton colors

of samples

The estimation of the probability of our following sample is something that has <u>never been seen before</u>.

To the notebook.

Good-Turing estimator

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

Good-Turing estimator

is able to estimate

the *missing mass*.

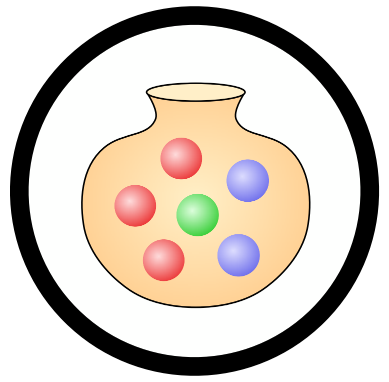
$$\hat{M}_0 = \frac{\Phi_1}{n}$$

Good-Turing estimator

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

is able to estimate

the *missing mass*.

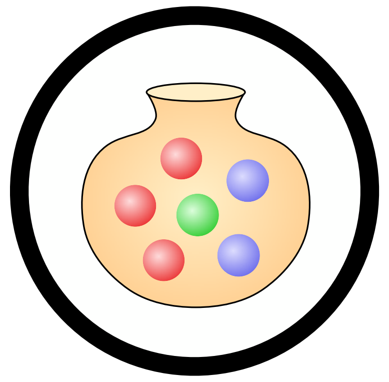
$\Leftrightarrow$ the probability of our next sample being a new color. 

Good-Turing estimator

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

is able to estimate

the *missing mass*.

$\Leftrightarrow$ the probability of our next sample being a new color. 

$\Leftrightarrow$ the probability of the next input generating a new coverage. 

How can the Good-Turing estimator estimate missing mass?

colors only seen
once in samples

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

of *singleton colors

of samples

How can the Good-Turing estimator estimate missing mass?

colors only seen
once in samples

of *singleton colors

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

# of samples

What it implies: "The probability of seeing an unseen event in the next sample is close to the probability of seeing a singleton event."

How can the Good-Turing estimator estimate missing mass?

colors only seen
once in samples

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

of *singleton colors

of samples

What it implies: "The probability of seeing an unseen event in the next sample is close to the probability of seeing a singleton event."

Loose explanation: "Because if we observe the unseen event, it becomes the singleton event."

A little bit more mathematics...

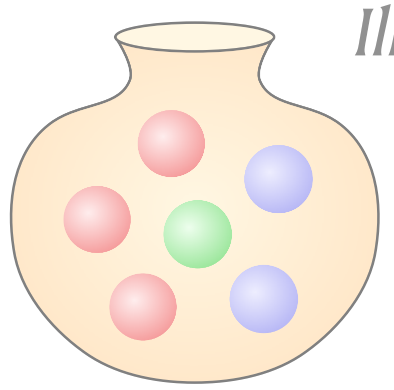


Illustration by Quartl; CC-BY-SA 3.0

- Let's say there is an urn filled with colored balls.
  - The probability of picking the ball of color $i = p_i$, $p_1 + p_2 \cdots + p_S = 1$
- Let's say we picked n balls from the urn.

- [# of **Singleton**] number of colors with only one ball in the sample

$$\Phi_1 = \sum_{i=1}^{S} \begin{cases} 1 & \text{if there's one ball with color } i \\ 0 & \text{otherwise} \end{cases} \Rightarrow_{\textit{on average}} \quad \sum_{i=1}^{S} \binom{n}{1} p_i (1-p_i)^{n-1}$$

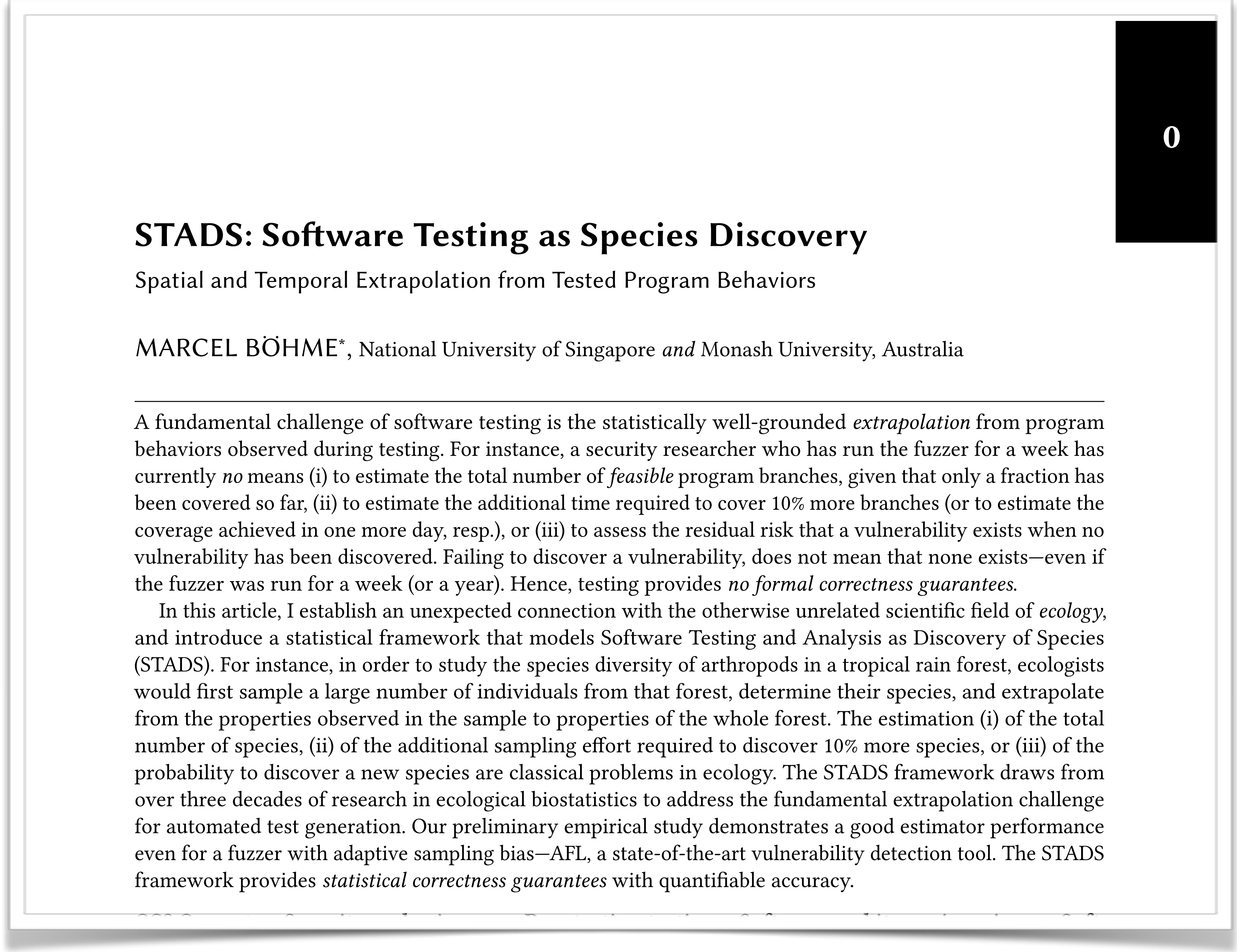
- **[Missing Mass]** The probability of observing one of the unseens

$$\sum_{i=1}^{S} \begin{cases} p_i & \text{if color } i \text{ is unobserved} \\ 0 & \text{otherwise} \end{cases} \Rightarrow_{\textit{on average}} \quad \sum_{i=1}^{S} p_i (1-p_i)^{n}$$

- When n is sufficiently large, $(1-p_i)^{n-1} \approx (1-p_i)^n$. Therefore,

$$\mathbb{E}\left[M_0\right] \approx \frac{\mathbb{E}[\Phi_1]}{n} \quad \Rightarrow \quad \frac{\Phi_1}{n} \text{ as an estimator for the missing mass } M_0 \,.$$

Grounded Mathematical Proof


0

STADS: Software Testing as Species Discovery

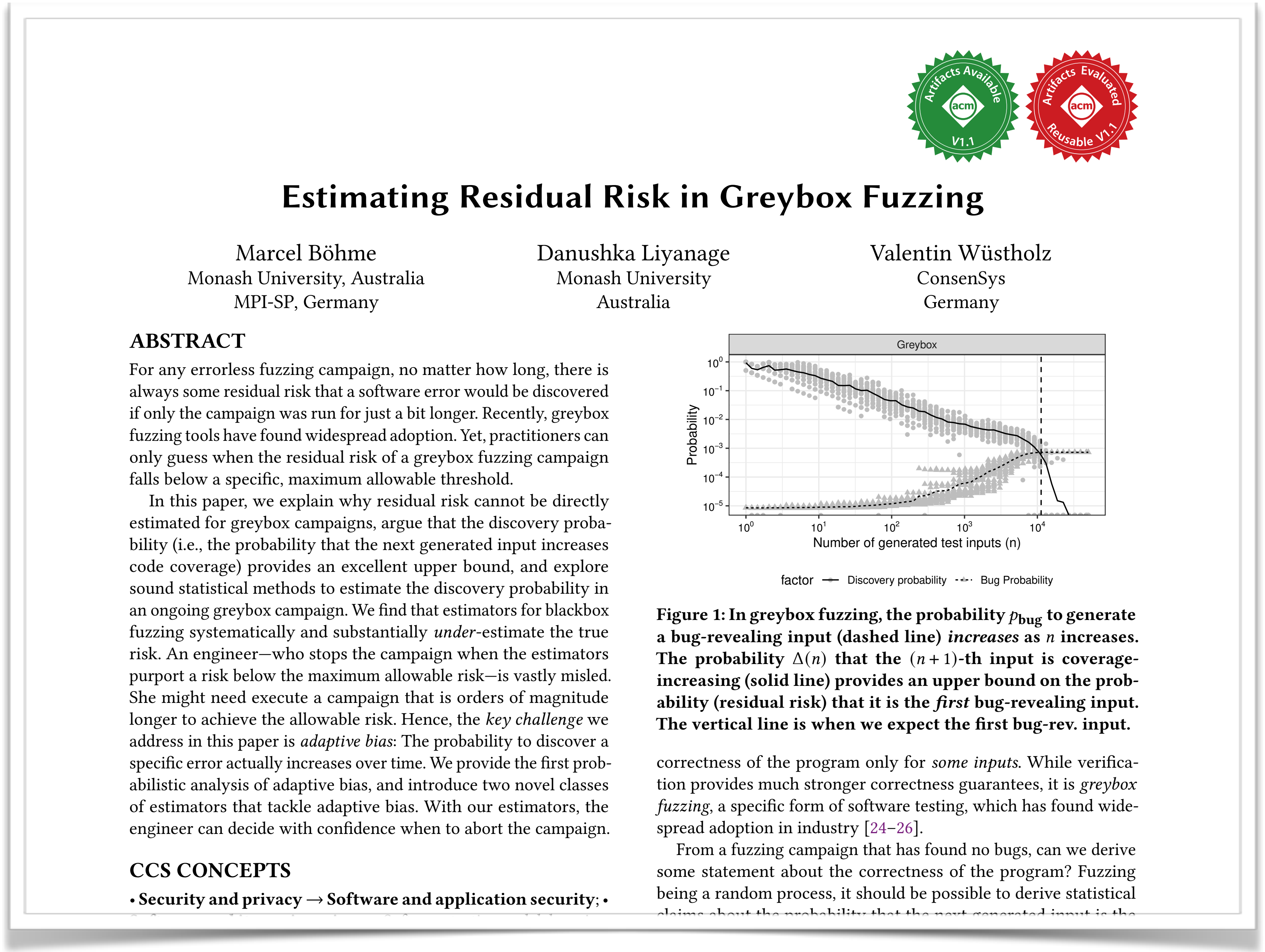
Spatial and Temporal Extrapolation from Tested Program Behaviors

MARCEL BÖHME*, National University of Singapore *and* Monash University, Australia

A fundamental challenge of software testing is the statistically well-grounded *extrapolation* from program behaviors observed during testing. For instance, a security researcher who has run the fuzzer for a week has currently *no* means (i) to estimate the total number of *feasible* program branches, given that only a fraction has been covered so far, (ii) to estimate the additional time required to cover 10% more branches (or to estimate the coverage achieved in one more day, resp.), or (iii) to assess the residual risk that a vulnerability exists when no vulnerability has been discovered. Failing to discover a vulnerability, does not mean that none exists—even if the fuzzer was run for a week (or a year). Hence, testing provides *no formal correctness guarantees.*

In this article, I establish an unexpected connection with the otherwise unrelated scientific field of *ecology*, and introduce a statistical framework that models Software Testing and Analysis as Discovery of Species (STADS). For instance, in order to study the species diversity of arthropods in a tropical rain forest, ecologists would first sample a large number of individuals from that forest, determine their species, and extrapolate from the properties observed in the sample to properties of the whole forest. The estimation (i) of the total number of species, (ii) of the additional sampling effort required to discover 10% more species, or (iii) of the probability to discover a new species are classical problems in ecology. The STADS framework draws from over three decades of research in ecological biostatistics to address the fundamental extrapolation challenge for automated test generation. Our preliminary empirical study demonstrates a good estimator performance even for a fuzzer with adaptive sampling bias—AFL, a state-of-the-art vulnerability detection tool. The STADS framework provides *statistical correctness guarantees* with quantifiable accuracy.

- *STADS: Software Testing as Species Discovery. Marcel Böhme. TOSEM 2018.*
  - Foundational work that interprets the software testing process as a statistical sampling process


Estimating Residual Risk in Greybox Fuzzing

Marcel Böhme, Monash University, Australia, MPI-SP, Germany
Danushka Liyanage, Monash University, Australia
Valentin Wüstholz, ConsenSys, Germany

ABSTRACT

For any errorless fuzzing campaign, no matter how long, there is always some residual risk that a software error would be discovered if only the campaign was run for just a bit longer. Recently, greybox fuzzing tools have found widespread adoption. Yet, practitioners can only guess when the residual risk of a greybox fuzzing campaign falls below a specific, maximum allowable threshold.

In this paper, we explain why residual risk cannot be directly estimated for greybox campaigns, argue that the discovery probability (i.e., the probability that the next generated input increases code coverage) provides an excellent upper bound, and explore sound statistical methods to estimate the discovery probability in an ongoing greybox campaign. We find that estimators for blackbox fuzzing systematically and substantially *under*-estimate the true risk. An engineer—who stops the campaign when the estimators purport a risk below the maximum allowable risk—is vastly misled. She might need execute a campaign that is orders of magnitude longer to achieve the allowable risk. Hence, the *key challenge* we address in this paper is *adaptive bias*: The probability to discover a specific error actually increases over time. We provide the first probabilistic analysis of adaptive bias, and introduce two novel classes of estimators that tackle adaptive bias. With our estimators, the engineer can decide with confidence when to abort the campaign.

CCS CONCEPTS

• Security and privacy → Software and application security; •

Figure 1: In greybox fuzzing, the probability p_{bug} to generate a bug-revealing input (dashed line) *increases* as n increases. The probability $\Delta(n)$ that the $(n+1)$-th input is coverage-increasing (solid line) provides an upper bound on the probability (residual risk) that it is the *first* bug-revealing input. The vertical line is when we expect the first bug-rev. input.

correctness of the program only for *some inputs*. While verification provides much stronger correctness guarantees, it is *greybox fuzzing*, a specific form of software testing, which has found widespread adoption in industry [24–26].

From a fuzzing campaign that has found no bugs, can we derive some statement about the correctness of the program? Fuzzing being a random process, it should be possible to derive statistical

- *Estimating residual risk in greybox fuzzing. Marcel Böhme, Danushka Liyanage, and Valentin Wüstholz. ESEC/FSE 2021*
  - Apply residual risk analysis on Greybox fuzzing

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

Missing Mass

What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

Extrapolation

How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

Extrapolation

~~How much more coverage~~ can I get if I fuzz the program with X more inputs?

How many more colors

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

$\hat{\Phi}_0$: the estimated number of remaining unseen

$$\frac{n-1}{n} \frac{(\Phi_1)^2}{2\Phi_2}$$

Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m)$$

$$= \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

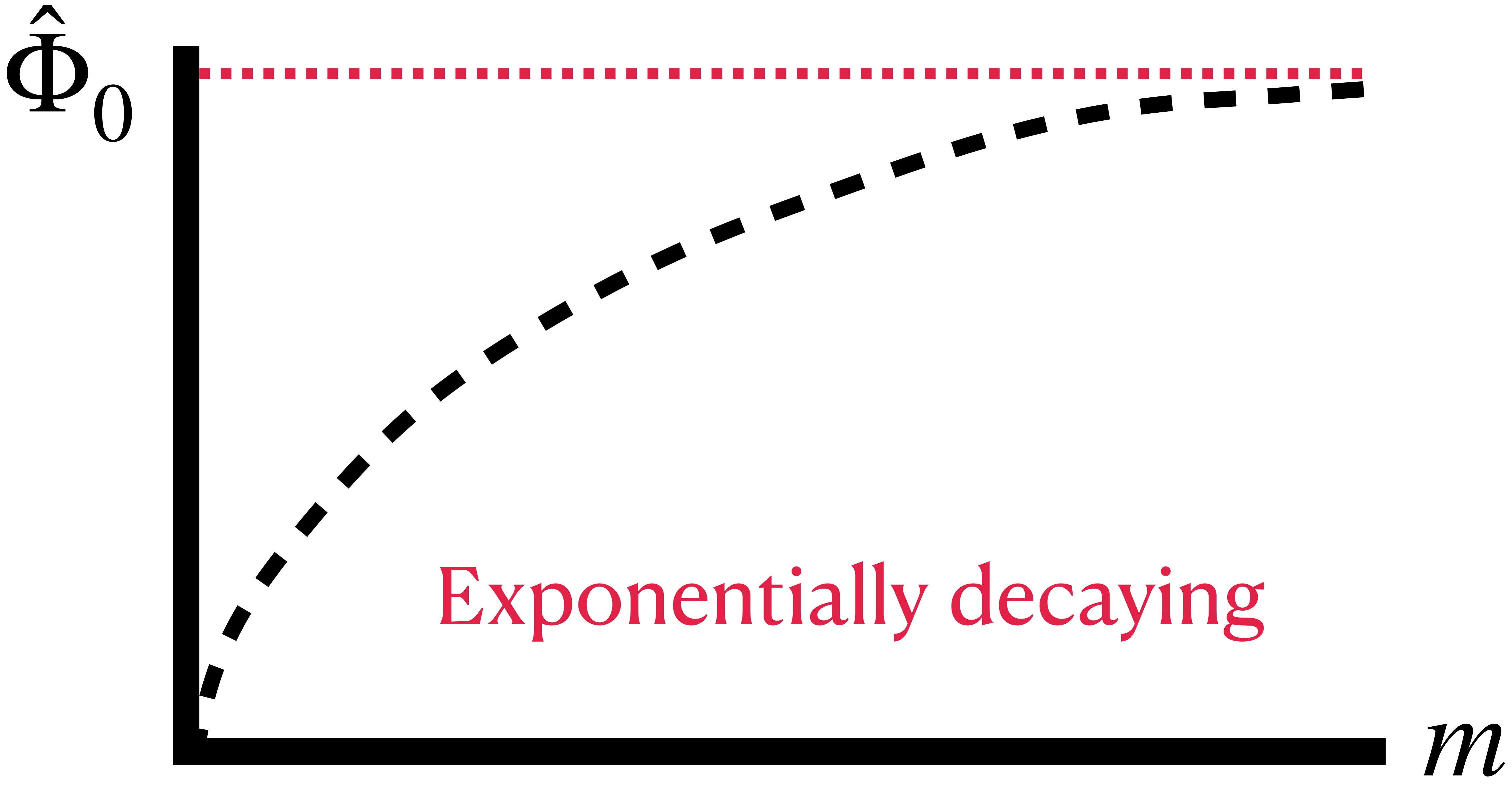
Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^{m} \right]$$



Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

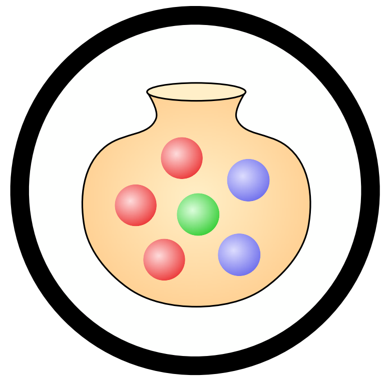
To the notebook.

Extrapolator

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Extrapolator $$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

It is able to

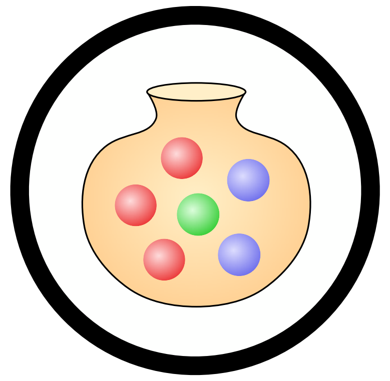
- extrapolate how many new colors we will observe more when we have m more samples. 

Extrapolator

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

It is able to

extrapolate how many new colors we will observe more when we have m more samples.



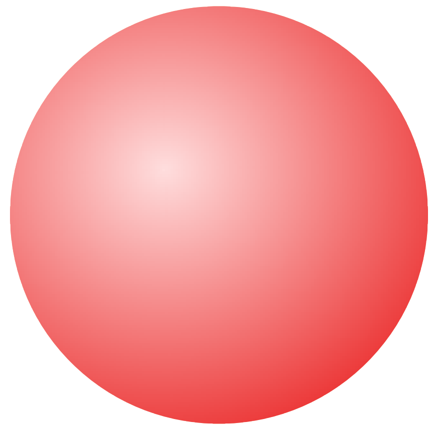
$\Leftrightarrow$ extrapolate the coverage increase when we run the fuzzing longer.

Each line coverage vector

(92,93,109,135,136,137,...)

(92,93,17,18)

(92,93,352,353,354,...)



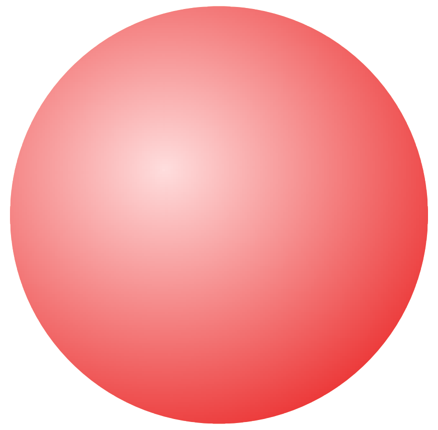
Color of a ball

Each line coverage vector

(92,93,109,135,136,137,...)

(92,93,17,18)

(92,93,352,353,354,...)



Color of a ball

92, 93, 109, ...

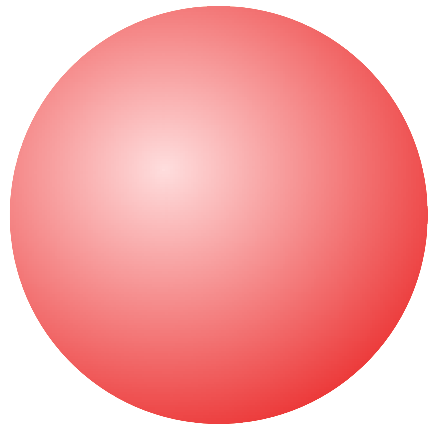
Each line

Each line coverage vector

`(92,93,109,135,136,137,...)`

`(92,93,17,18)`

`(92,93,352,353,354,...)`



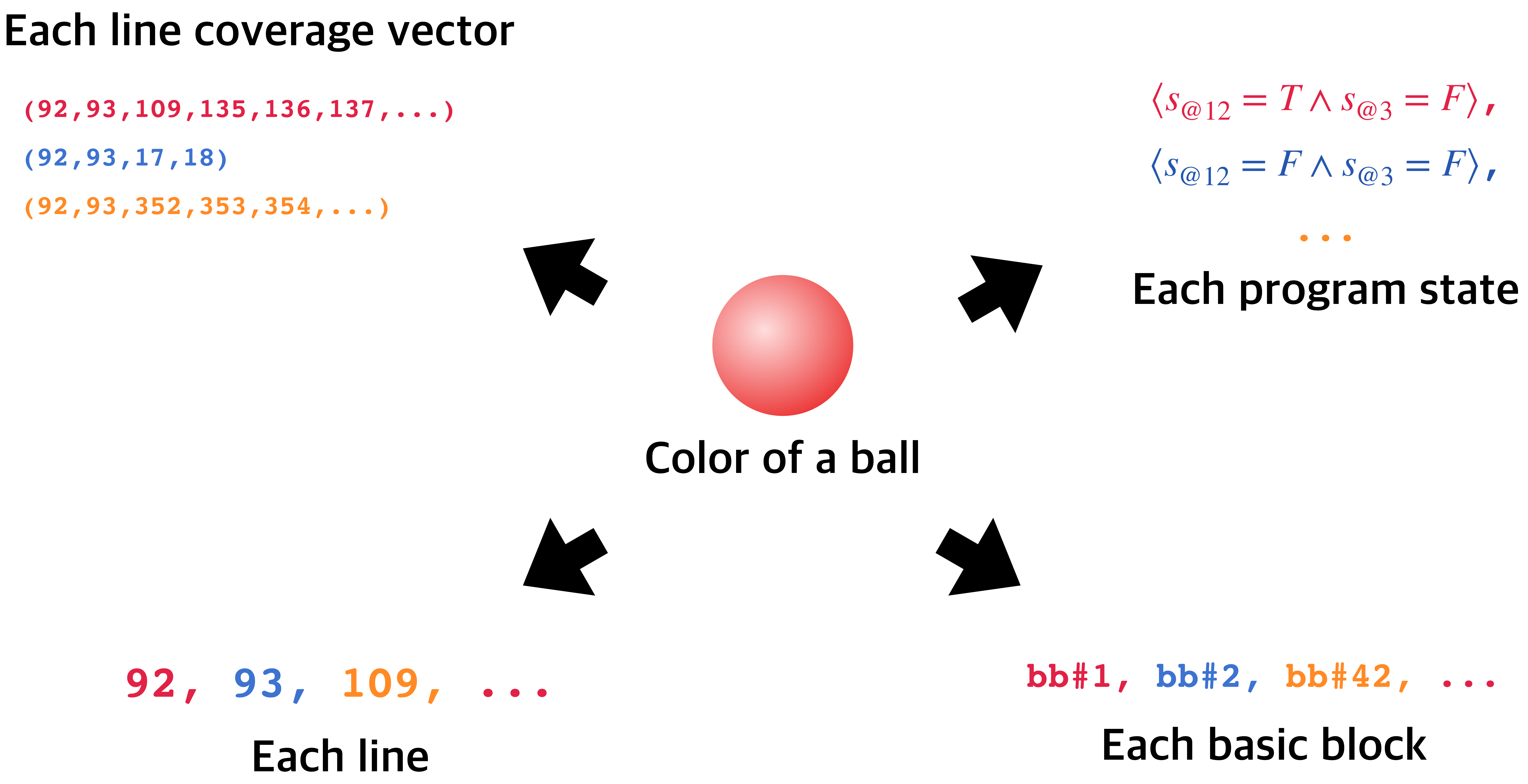
Color of a ball

`92, 93, 109, ...`

Each line

`bb#1, bb#2, bb#42, ...`

Each basic block


Each line coverage vector
(92,93,109,135,136,137,...)
(92,93,17,18)
(92,93,352,353,354,...)
$\langle s_{@12} = T \wedge s_{@3} = F \rangle$,
$\langle s_{@12} = F \wedge s_{@3} = F \rangle$,
...
Each program state
Color of a ball
92, 93, 109, ...
Each line
bb#1, bb#2, bb#42, ...
Each basic block

0

STADS: Software Testing as Species Discovery

Spatial and Temporal Extrapolation from Tested Program Behaviors

MARCEL BÖHME*, National University of Singapore *and* Monash University, Australia

A fundamental challenge of software testing is the statistically well-grounded *extrapolation* from program behaviors observed during testing. For instance, a security researcher who has run the fuzzer for a week has currently *no* means (i) to estimate the total number of *feasible* program branches, given that only a fraction has been covered so far, (ii) to estimate the additional time required to cover 10% more branches (or to estimate the coverage achieved in one more day, resp.), or (iii) to assess the residual risk that a vulnerability exists when no vulnerability has been discovered. Failing to discover a vulnerability, does not mean that none exists—even if the fuzzer was run for a week (or a year). Hence, testing provides *no formal correctness guarantees*.

In this article, I establish an unexpected connection with the otherwise unrelated scientific field of *ecology*, and introduce a statistical framework that models Software Testing and Analysis as Discovery of Species (STADS). For instance, in order to study the species diversity of arthropods in a tropical rain forest, ecologists would first sample a large number of individuals from that forest, determine their species, and extrapolate from the properties observed in the sample to properties of the whole forest. The estimation (i) of the total number of species, (ii) of the additional sampling effort required to discover 10% more species, or (iii) of the probability to discover a new species are classical problems in ecology. The STADS framework draws from over three decades of research in ecological biostatistics to address the fundamental extrapolation challenge for automated test generation. Our preliminary empirical study demonstrates a good estimator performance even for a fuzzer with adaptive sampling bias—AFL, a state-of-the-art vulnerability detection tool. The STADS framework provides *statistical correctness guarantees* with quantifiable accuracy.

CCS Concepts: • **Security and privacy → Penetration testing**; • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Statistical guarantees, extrapolation, fuzzing, stopping rule, code coverage, species coverage, discovery probability, security, reliability, measure of confidence, measure of progress

ACM Reference format:
Marcel Böhme. 2018. STADS: Software Testing as Species Discovery. *ACM Trans. Softw. Eng. Methodol.* 0, 0, Article 0 (April 2018), 52 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

The development of automated and practical approaches to vulnerability detection has never been more important. The recent world-wide WannaCry cyber-epidemic clearly demonstrates the vulnerability of our well-connected software systems. WannaCry exploits a *software vulnerability* on Windows machines to gain root access on a huge number of computers all over the world. The

*Dr. Böhme conducted this research at the National University of Singapore and has since moved to Monash University.

1049-331X/2018/4-ART0
https://doi.org/0000001.0000001

- *STADS: Software Testing as Species Discovery. Marcel Böhme. TOSEM 2018.*
  - Foundational work that interprets the software testing process as a statistical sampling process

Questions?

Missing Mass

*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

*How much more can I achieve **if I spend** X **more time** here?*

*Present **advanced extensions** to adopt statistical methods for more **realistic testing scenarios**.*

2

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

Good-Turing estimator

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^m \right]$$

Extrapolator

$$\cancel{\hat{M}_0 = \frac{\Phi_1}{n}}$$

Good-Turing estimator

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(\cancel{1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1}} \right)^m \right]$$

Extrapolator

Depending on the problem one wants to solve,

the **statistical estimator may not be directly applicable**.

Missing Mass

*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

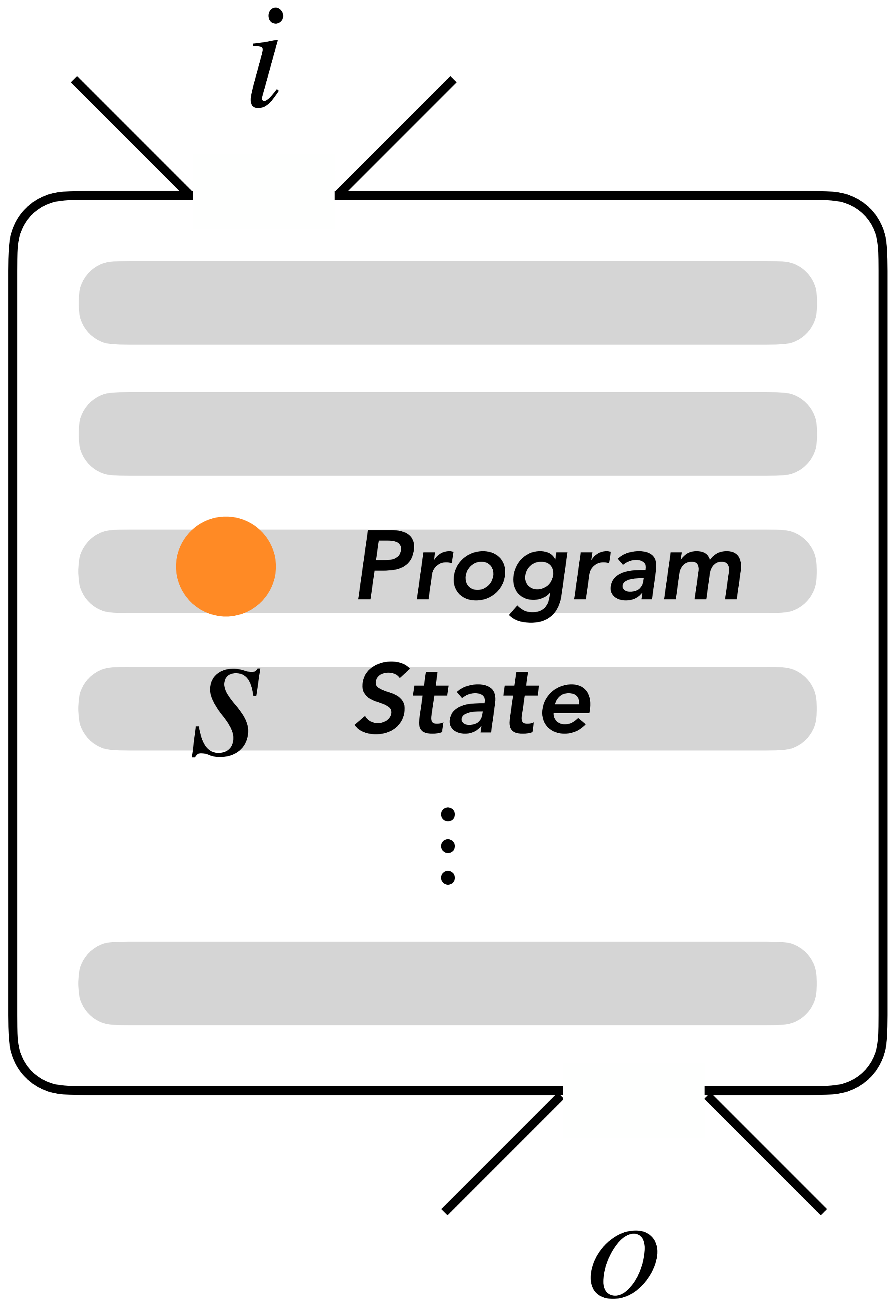
*How much more can I achieve **if I spend** X **more time** here?*

*Present **advanced extensions** to adopt statistical methods for more **realistic testing scenarios**.*

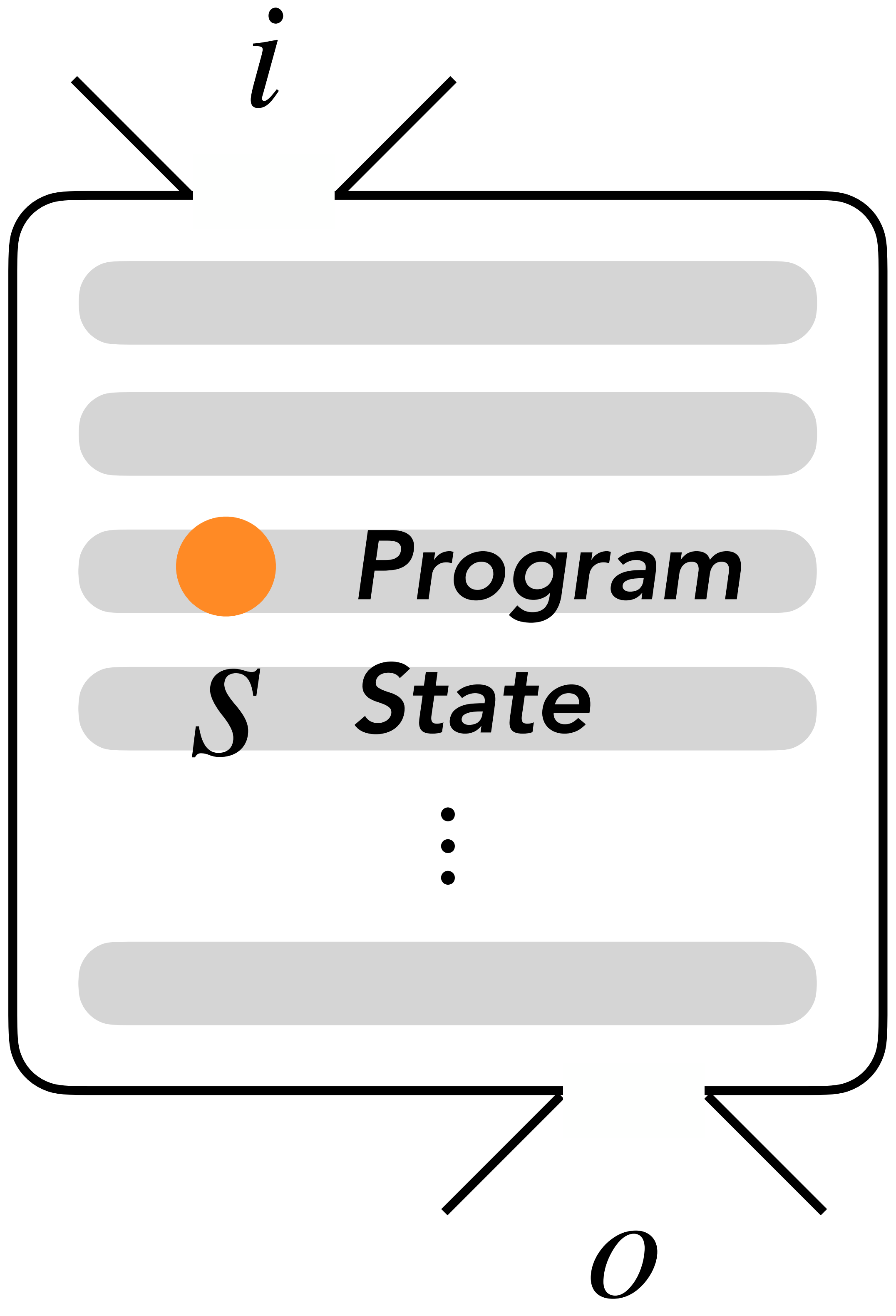
2

Program

P

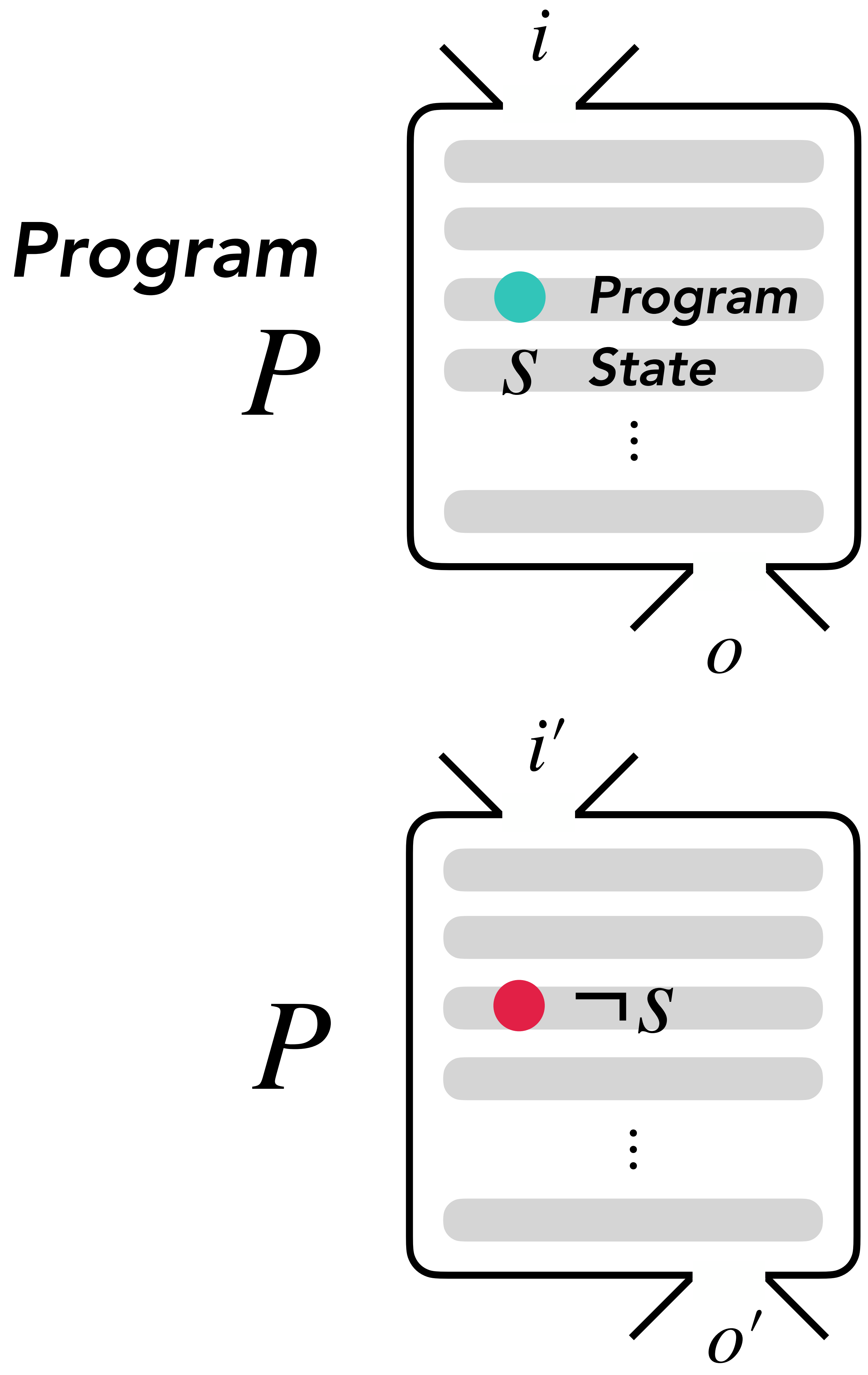


Program
P



“What is the probability of reaching s?”

Quantitative Reachability Analysis (QRA)



A *program state* is a property one is interested in that is either reached or unreached, given the program execution.

Quantitative Reachability Analysis (QRA) measures the probability of how likely a certain program state is reached given the workload of the program.

$$\Pr(s) = \sum_{e \in E} \Pr(e) \cdot \mathbf{1}(s \text{ is reached by } e)$$

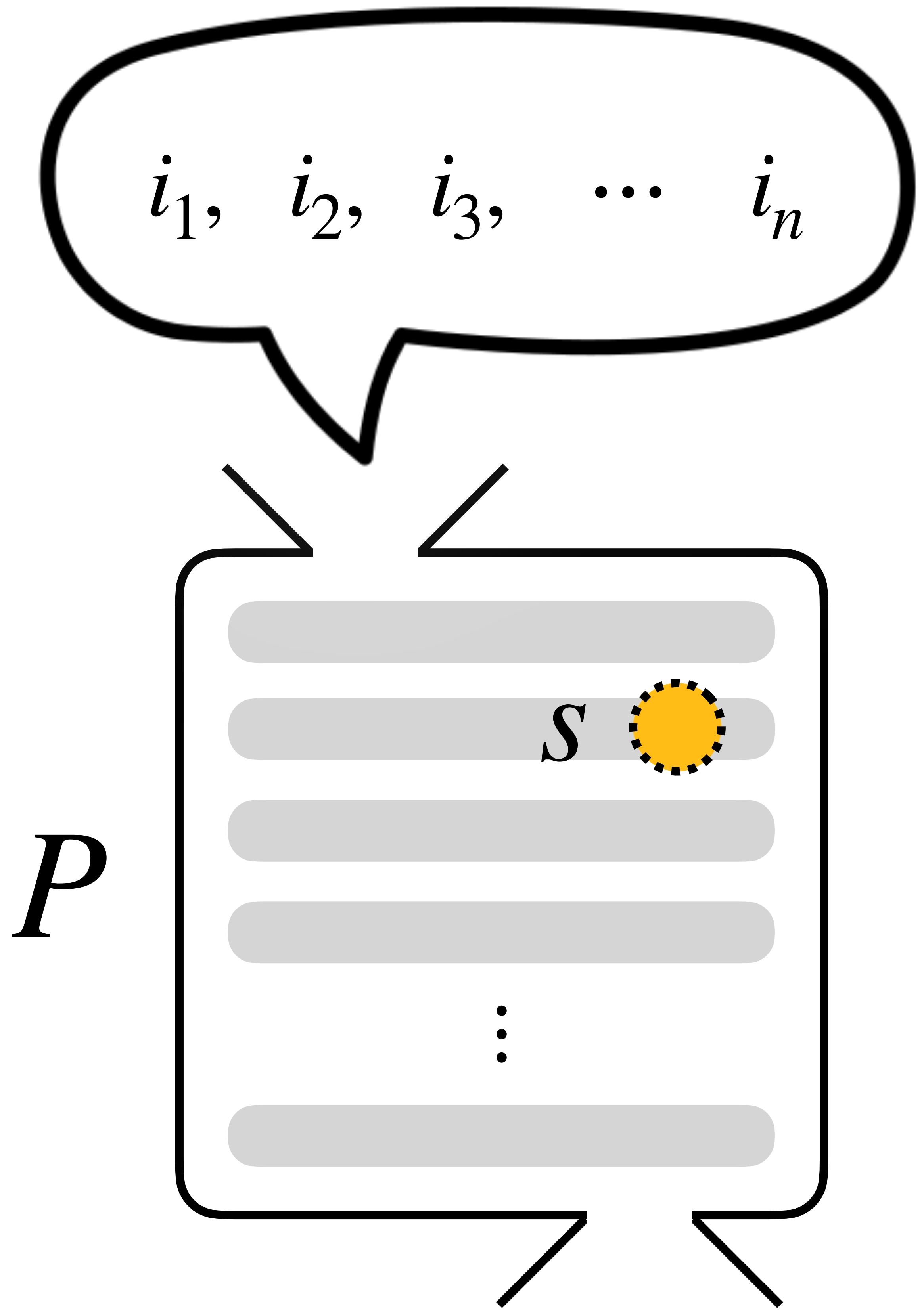
E: workload or execution profile

Statistical Reachability Analysis (SRA)

— For ***seen*** program states, —

Statistical Reachability Analysis (SRA)

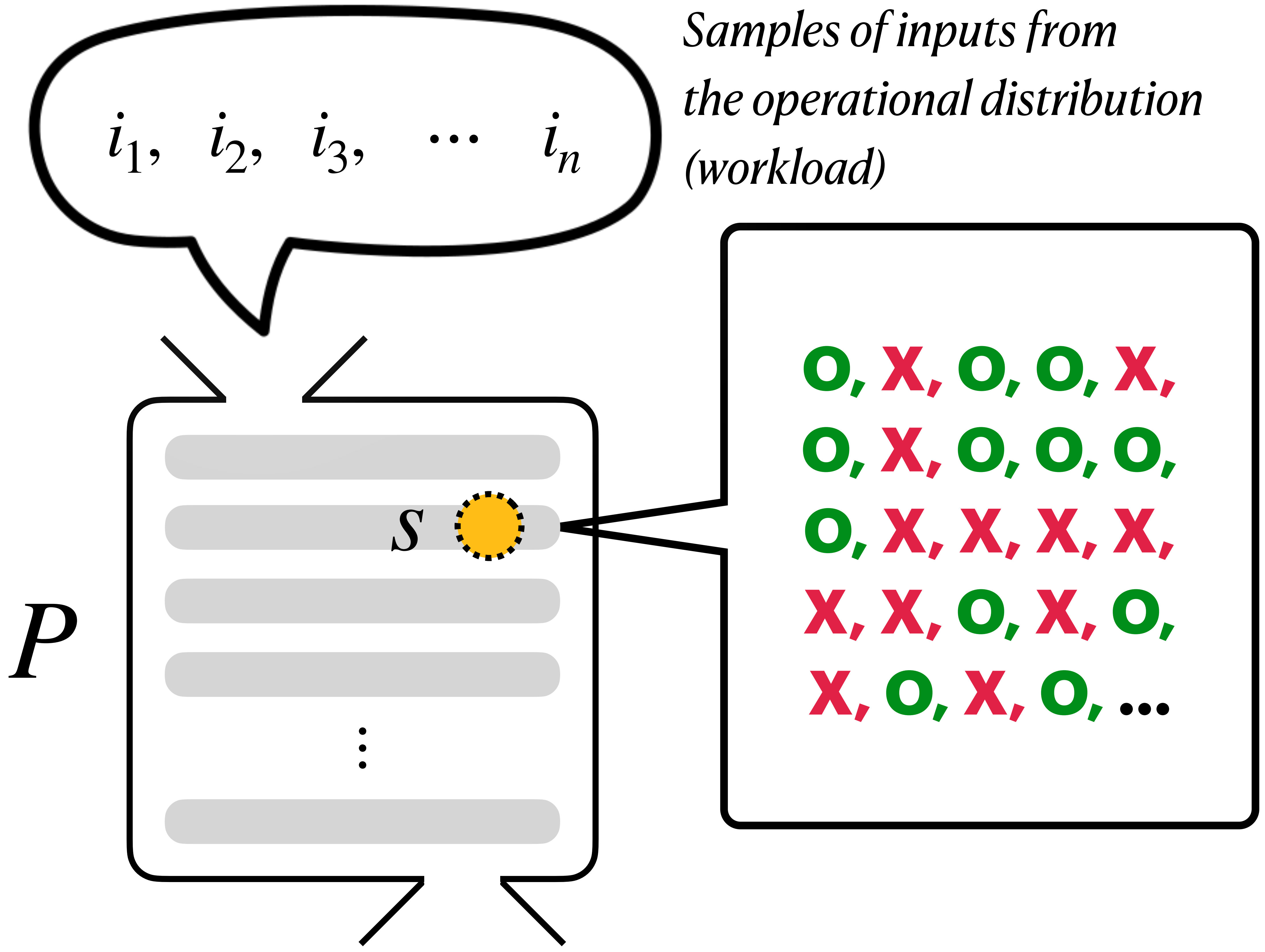
— For ***seen*** program states, —



Samples of inputs from the operational distribution (workload)

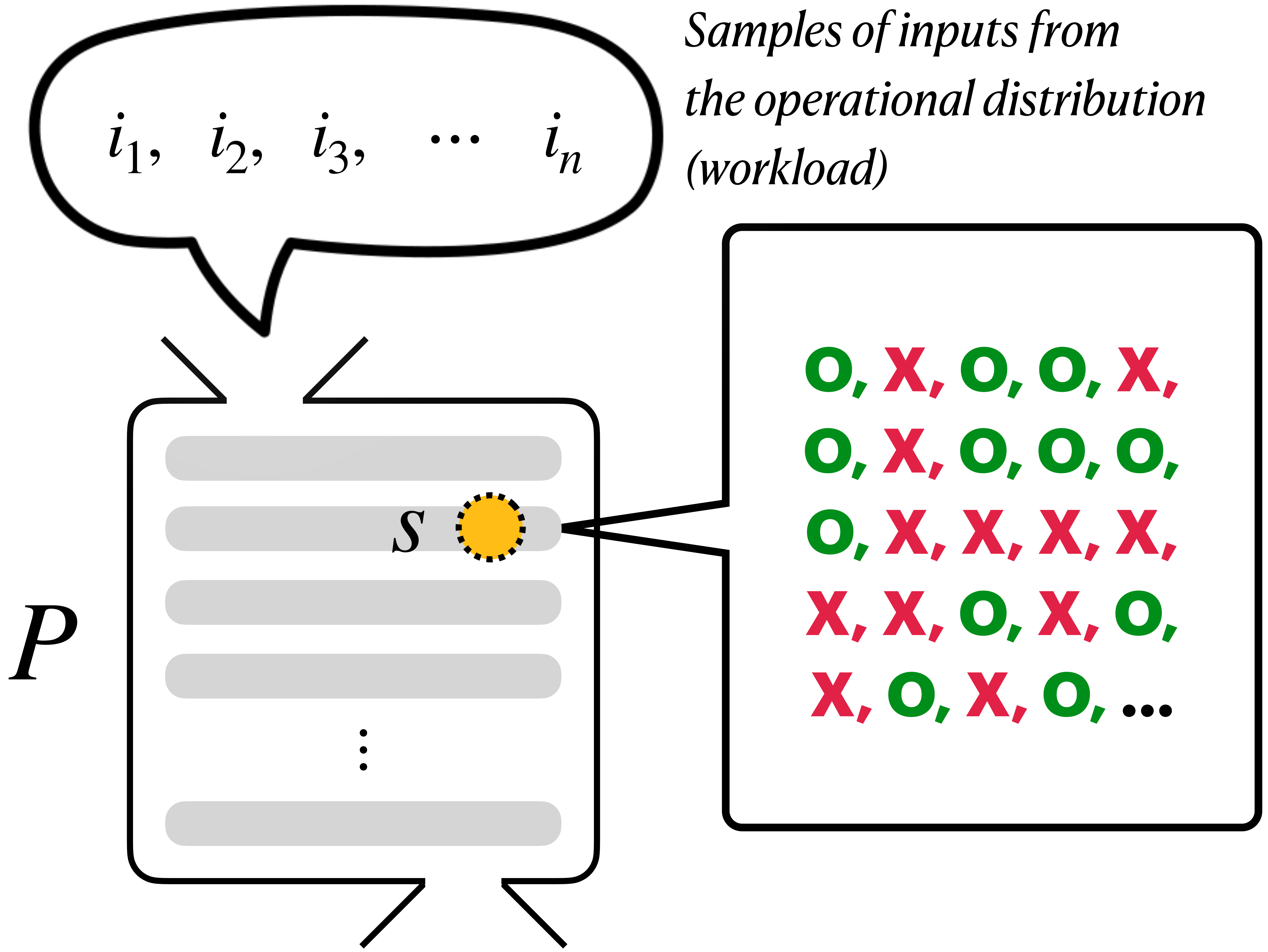
Statistical Reachability Analysis (SRA)

— For ***seen*** program states, —



Statistical Reachability Analysis (SRA)

— For ***seen*** program states, —



X_s := the number of O in n samples

$$\hat{\Pr}(s) = \frac{X_s}{n} \xRightarrow{n \to \infty} \Pr(s)$$

Empirical Probability

Challenge: Missing Rare Program States

If the state s is rarely observable, i.e., $\Pr(s) \approx 0$,

$$\mathbb{E}\left(\frac{X_s}{n} \,\middle|\, X_s = 0\right) = 0$$

If it is unobserved, the empirical probability
underapproximates to zero probability.

Challenge: Missing Rare Program States

If the state s is rarely observable, i.e., $\Pr(s) \approx 0$,

$$\mathbb{E}\left(\frac{X_s}{n} \middle| X_s = 0\right) = 0$$

If it is unobserved, the empirical probability underapproximates to zero probability.

Problem of unseen events / 🌞 Sunrise problem

“Wait... don't we already know how to do that?”

"Wait... don't we already know how to do that?"

Solution:

Good-Turing estimator

Alan Turing

colors only seen once in samples

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

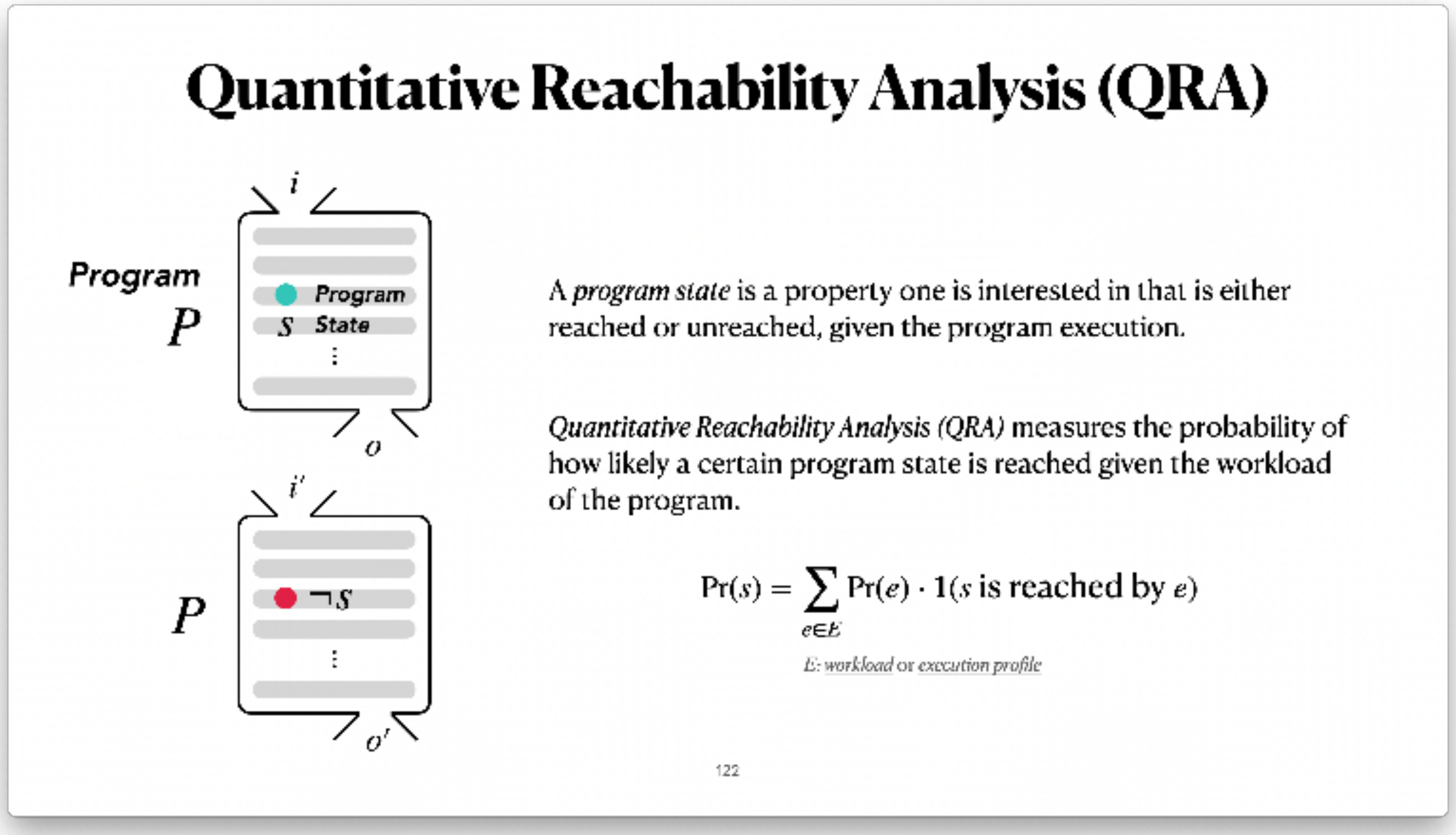
Φ_1: # of *singleton colors

n: # of samples

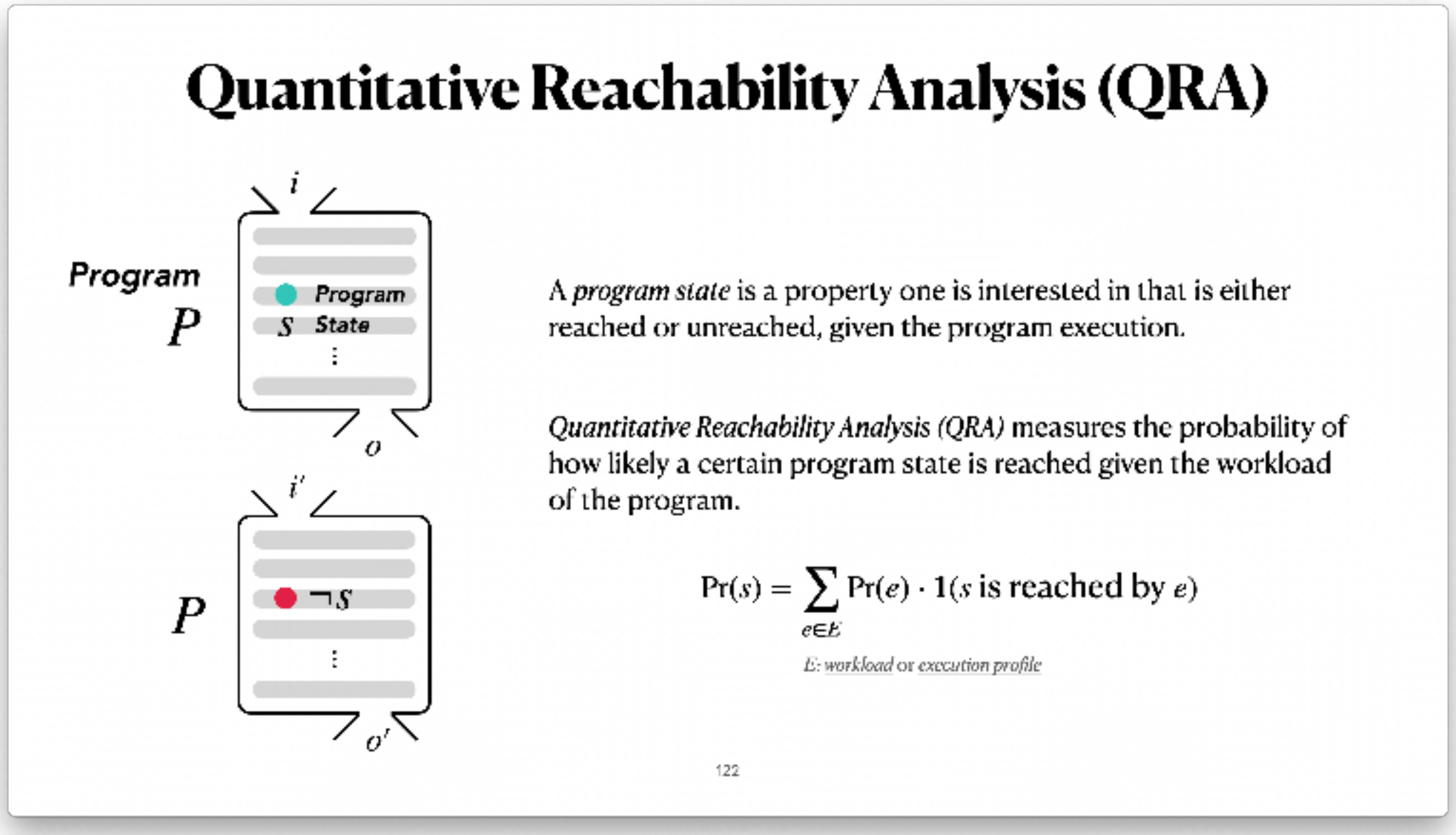
The estimation of the probability of our following sample is something that has <u>never been seen before</u>.

— The estimator for the probability of an unseen event happening —

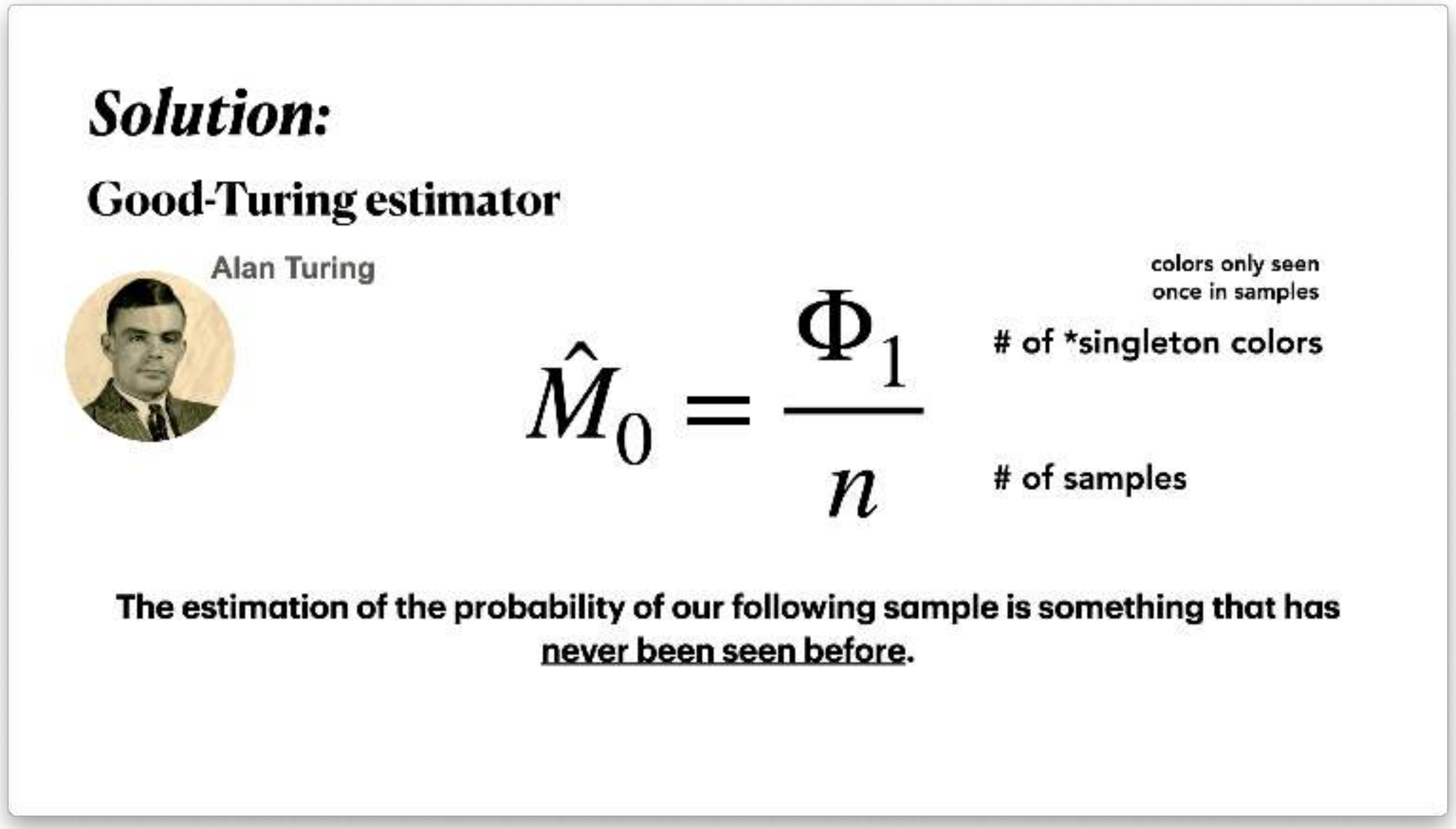
Problem


Quantitative Reachability Analysis (QRA)
i
Program
P
Program
s State
o
i'
P
¬s
o'
A program state is a property one is interested in that is either reached or unreached, given the program execution.
Quantitative Reachability Analysis (QRA) measures the probability of how likely a certain program state is reached given the workload of the program.
$\Pr(s) = \sum_{e \in E} \Pr(e) \cdot \mathbf{1}(s \text{ is reached by } e)$
E: workload or execution profile
122

Problem

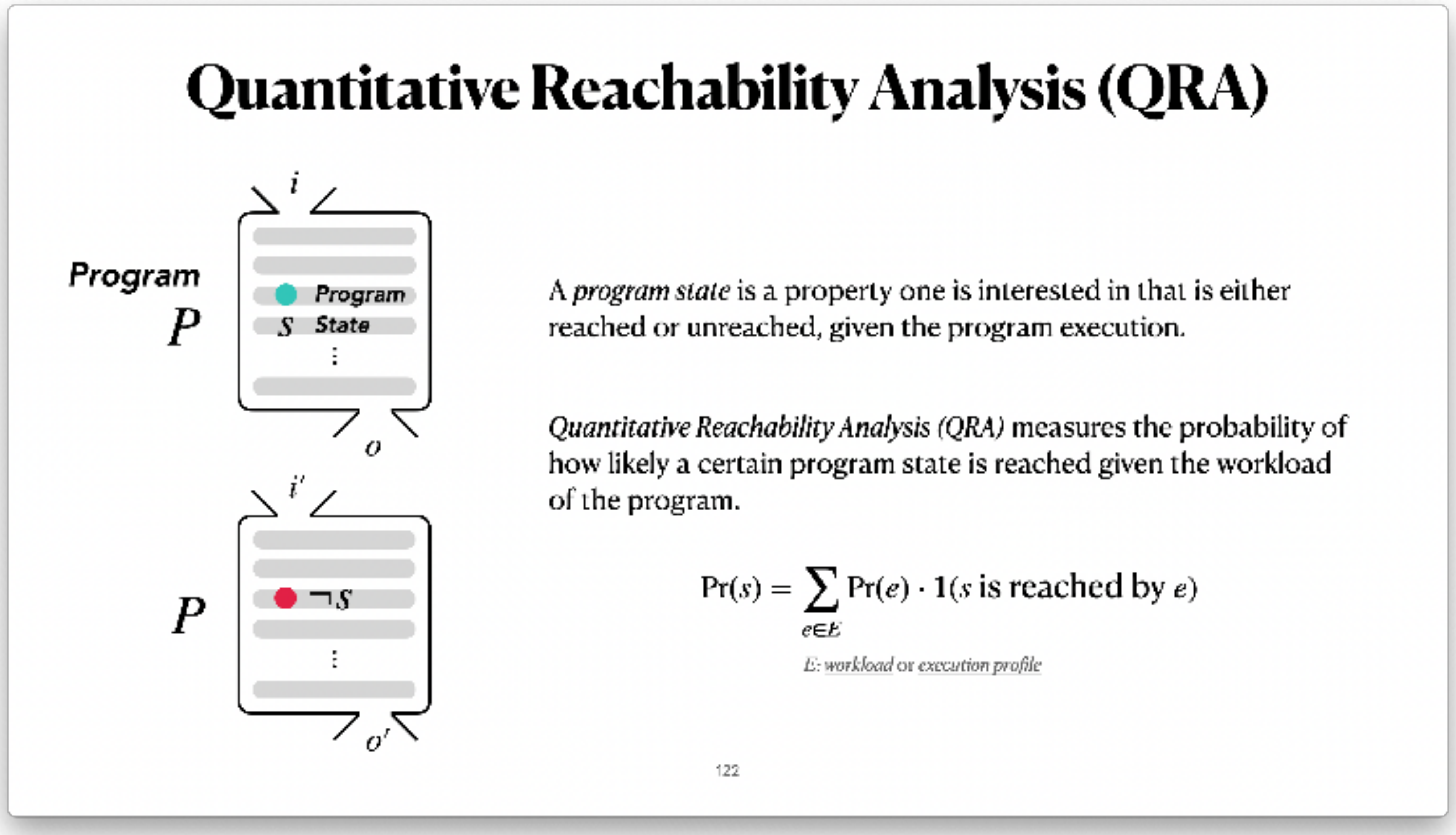


Previous solution

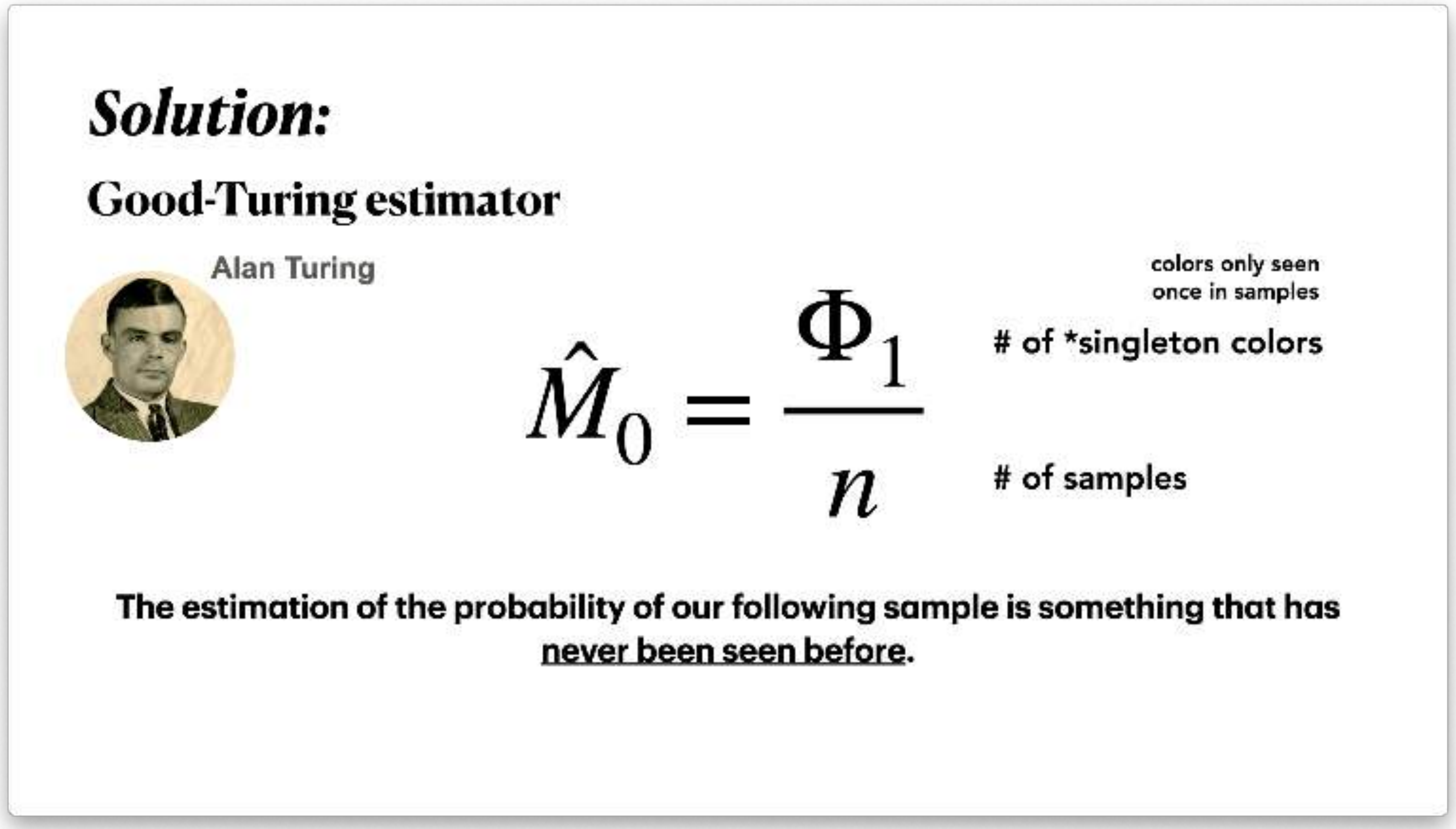


Problem

Previous solution



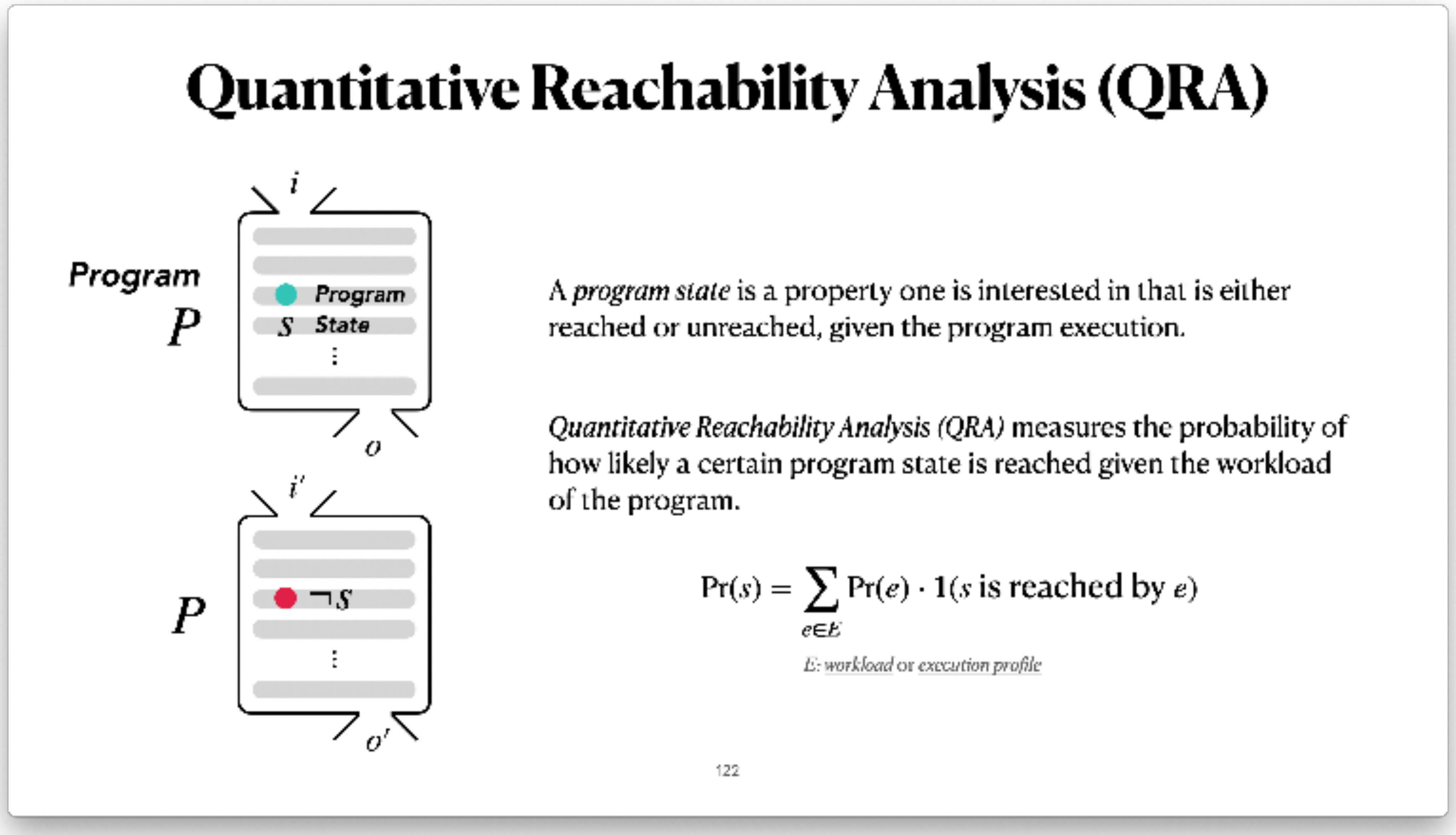
≠



We want the probability of seeing ***THE unseen state,*** not *AN unseen state.*

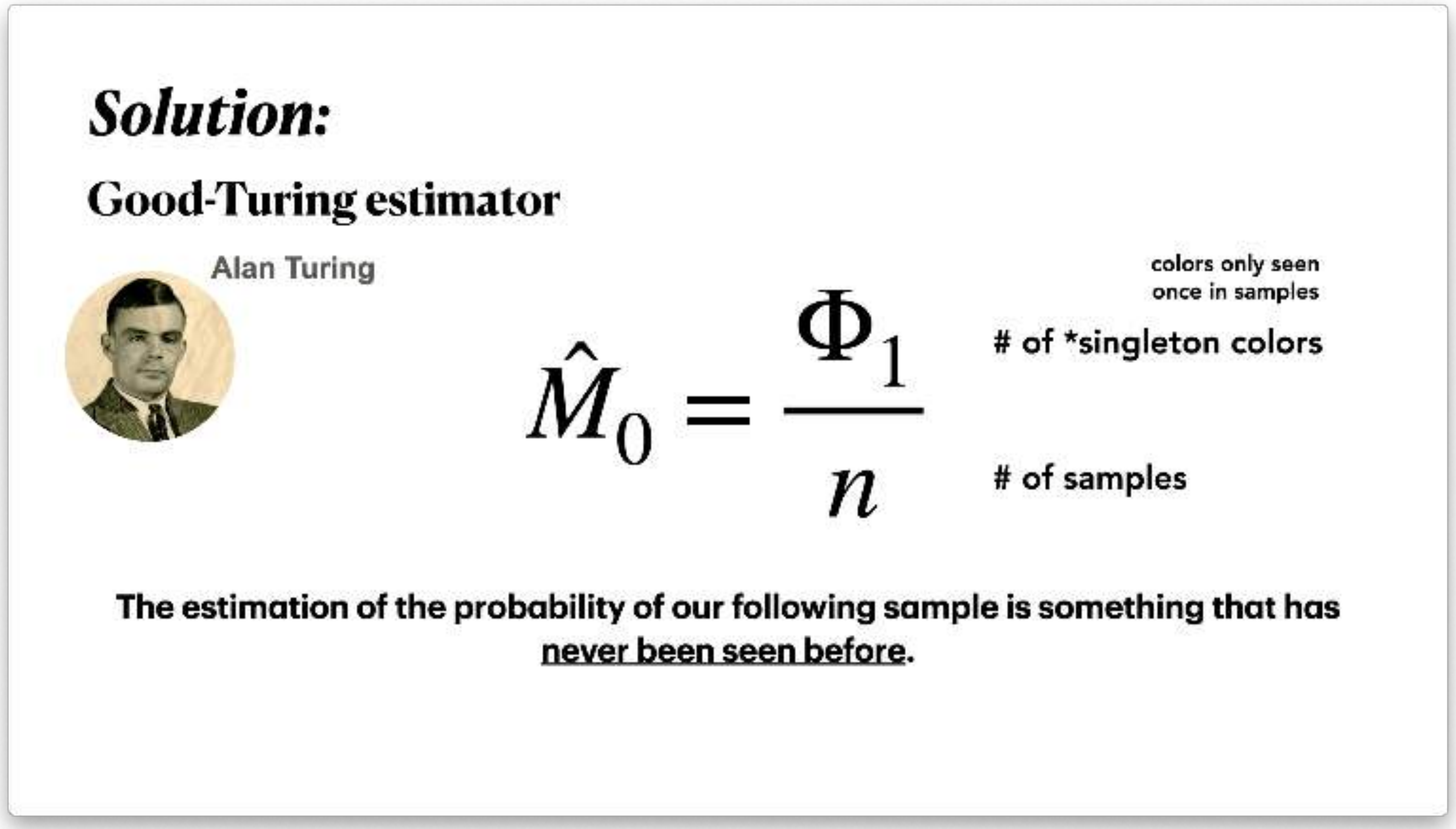
(specific) (any)

Problem



$\neq$

Blackbox Estimator



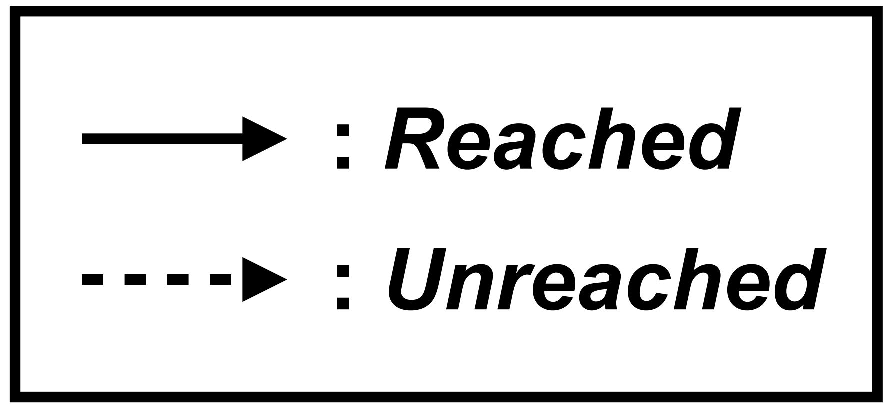
We want the probability of seeing ***THE unseen state,*** not *AN unseen state*.

(specific) (any)

: *Reached*

: *Unreached*

Blackbox estimators cannot distinguish between unseen states.

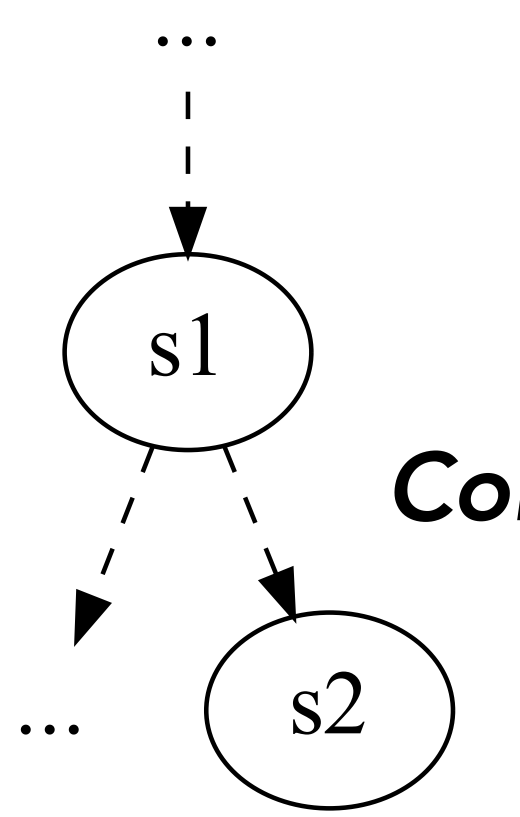


Blackbox estimators **cannot distinguish between unseen states.**

1

Source

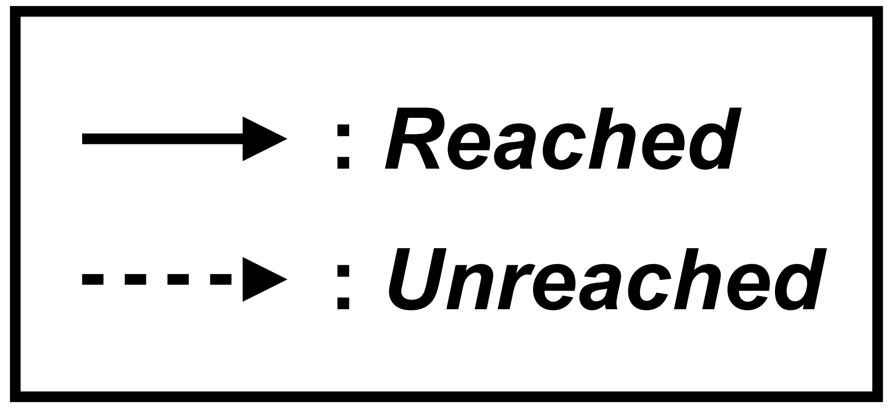
```
    ...
s1: if (pred)
s2:   stmt;
    ...
```



Control-flow

$\Pr(s_1) \geq \Pr(s_2)$;

However $\hat{\Pr}_{BB}(s_1, O) = \hat{\Pr}_{BB}(s_2, O)$, given the sample O.

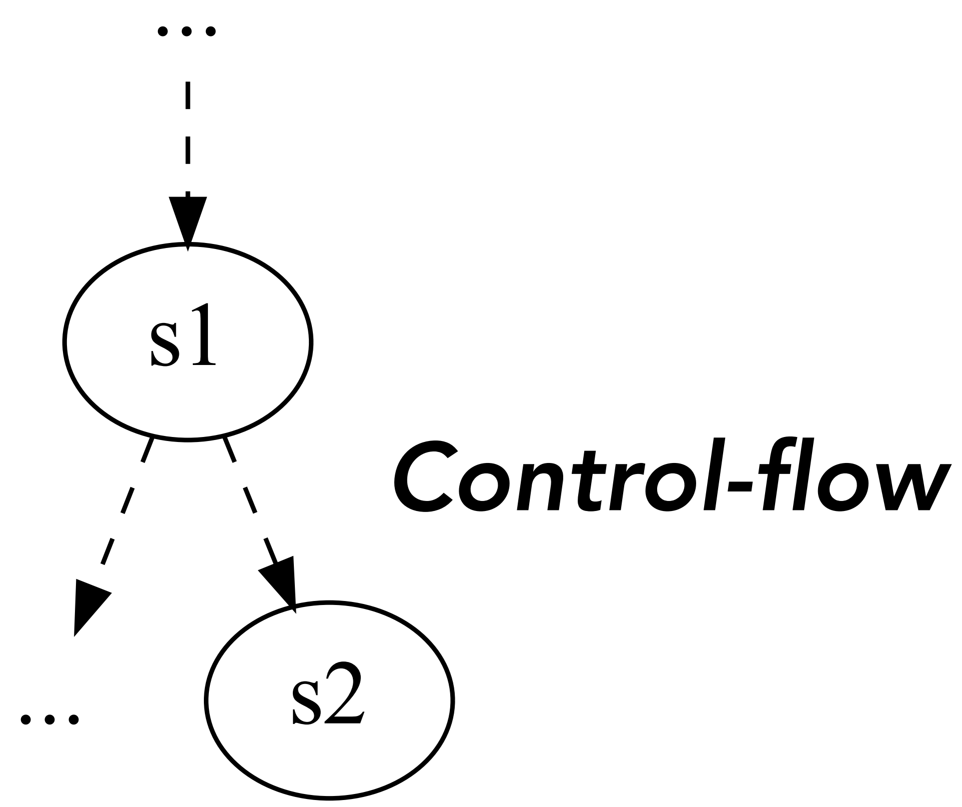


Blackbox estimators *cannot distinguish between unseen states.*

1

Source

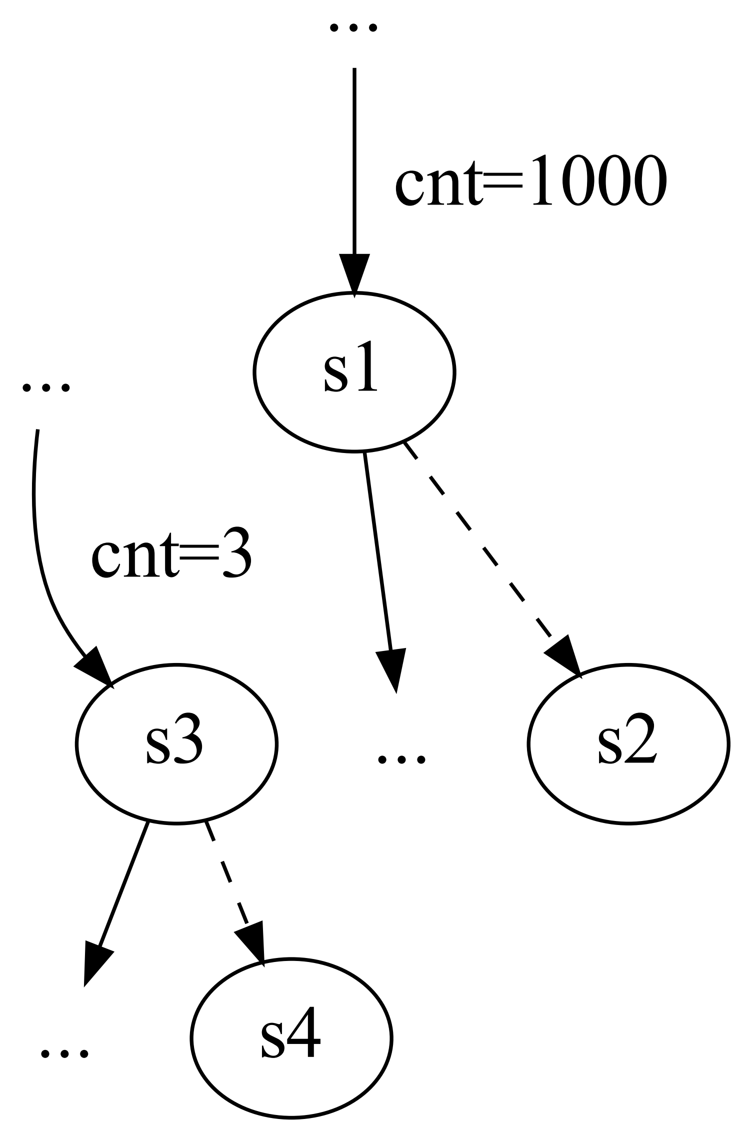
```
     ...
s1: if (pred)
s2:    stmt;
     ...
```



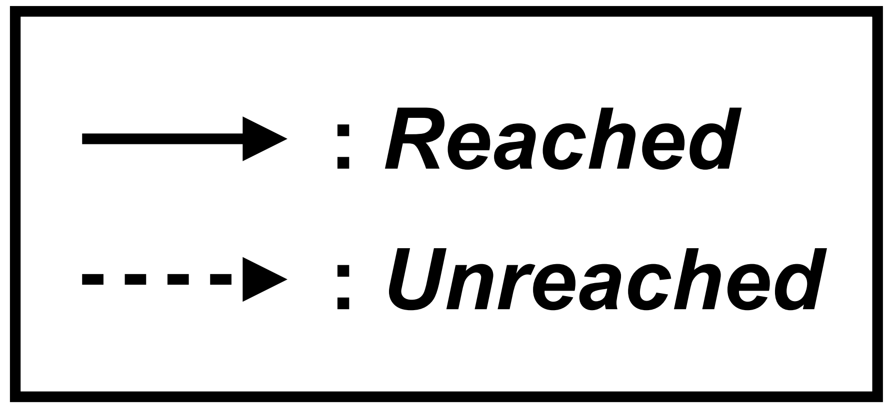
$\Pr(s_1) \geq \Pr(s_2)$;

However $\hat{\Pr}_{BB}(s_1, O) = \hat{\Pr}_{BB}(s_2, O)$, given the sample O.

2



s_2 has larger chances of being reached than s_4

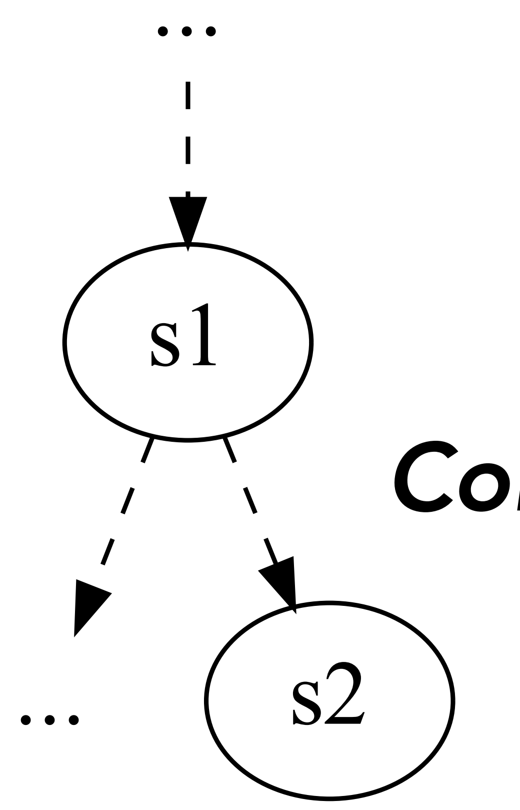


Blackbox estimators cannot distinguish between unseen states.

1

Source

```
    ...
s1: if (pred)
s2:   stmt;
    ...
```

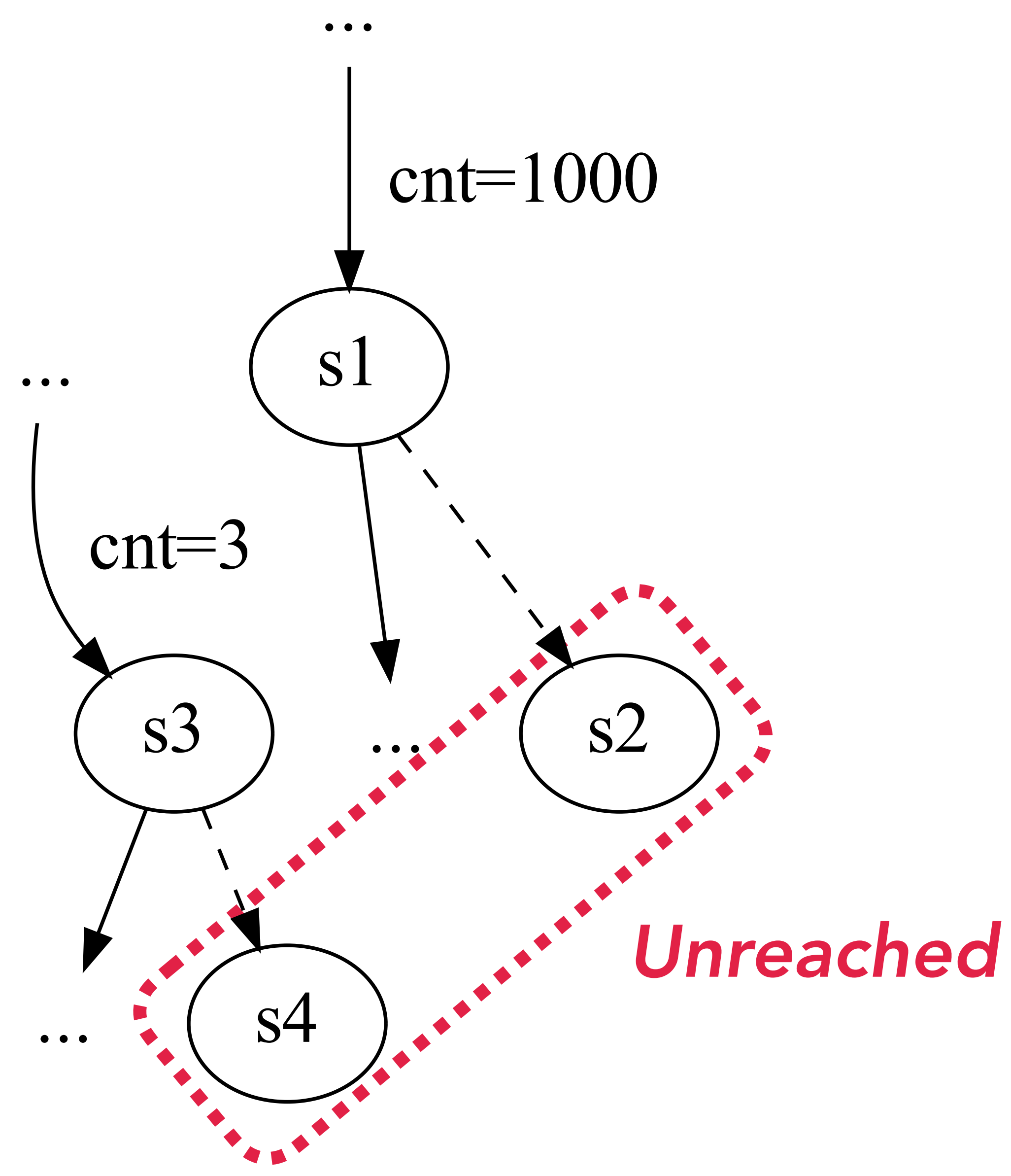


Control-flow

$\Pr(s_1) \geq \Pr(s_2)$;

However $\hat{\Pr}_{BB}(s_1, O) = \hat{\Pr}_{BB}(s_2, O)$, given the sample O.

2



s_2 has larger chances of being reached than s_4

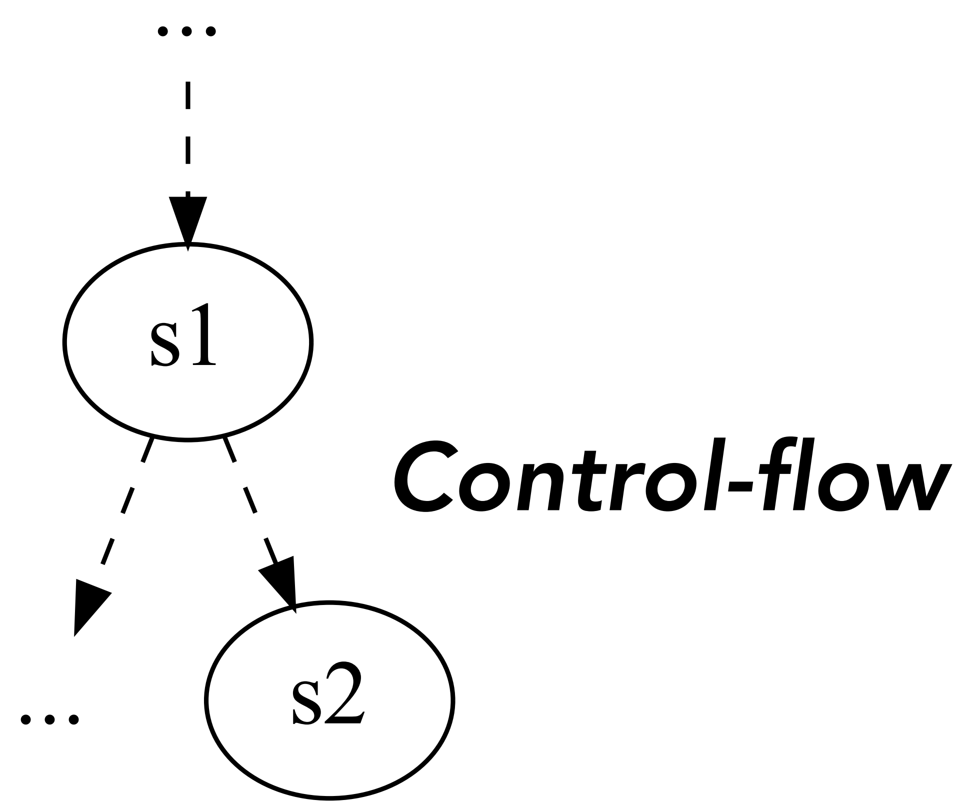
→ : *Reached*
- - -→ : *Unreached*

Blackbox estimators cannot distinguish between unseen states.

1

Source

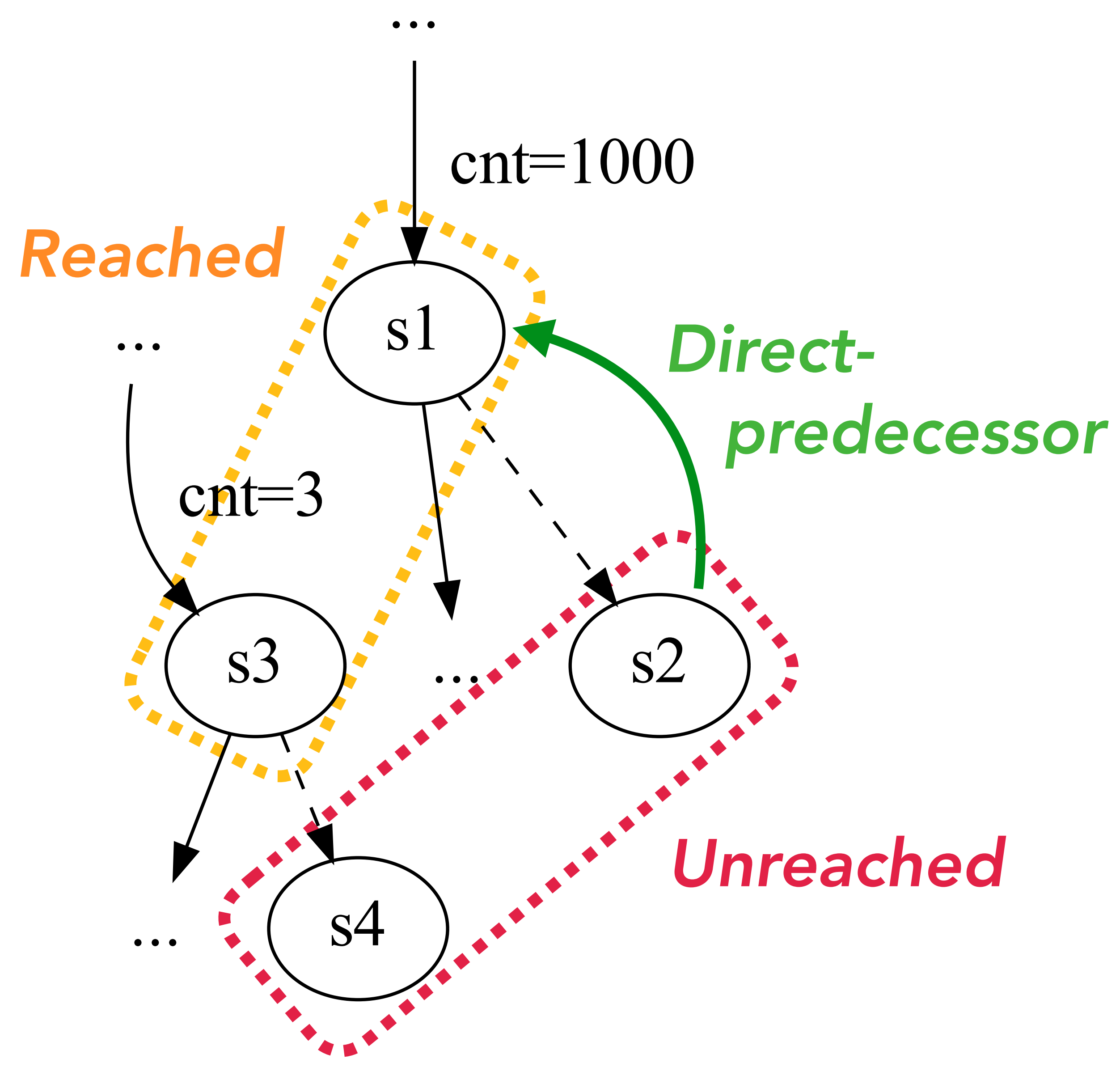
```
    ...
s1: if (pred)
s2:   stmt;
    ...
```



$\Pr(s_1) \geq \Pr(s_2)$;

However $\hat{\Pr}_{BB}(s_1, O) = \hat{\Pr}_{BB}(s_2, O)$, given the sample O.

2



s_2 has larger chances of being reached than s_4

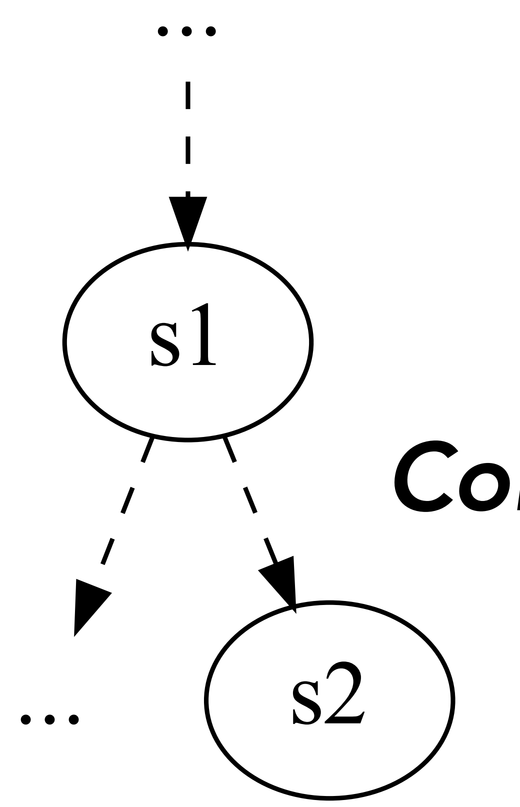
→ : *Reached*
⇢ : *Unreached*

Blackbox estimators cannot distinguish between unseen states.

1

Source

```
   ...
s1: if (pred)
s2:   stmt;
   ...
```

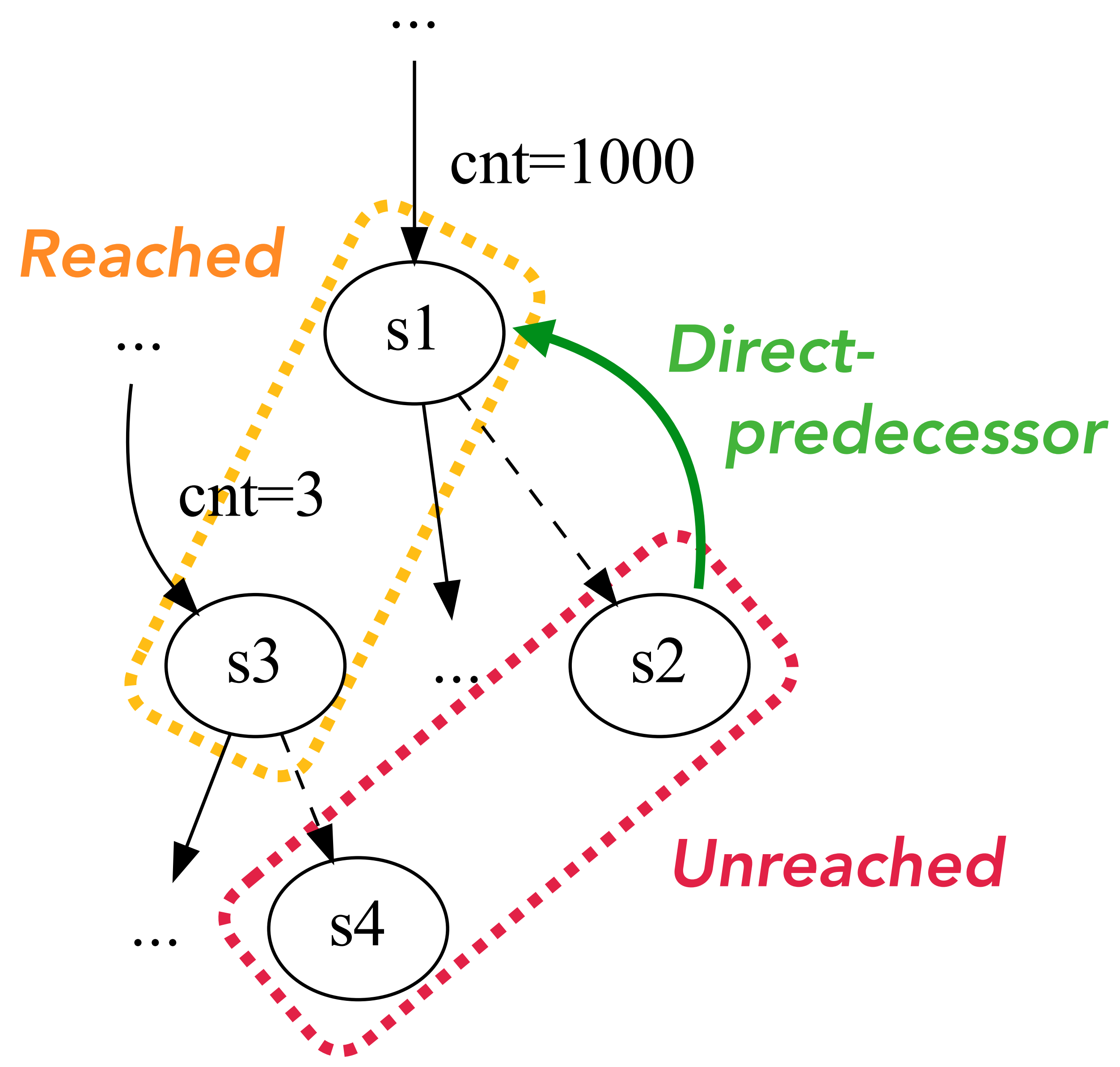


Control-flow

$\Pr(s_1) \geq \Pr(s_2)$;

However $\hat{\Pr}_{BB}(s_1, O) = \hat{\Pr}_{BB}(s_2, O)$, given the sample O.

2

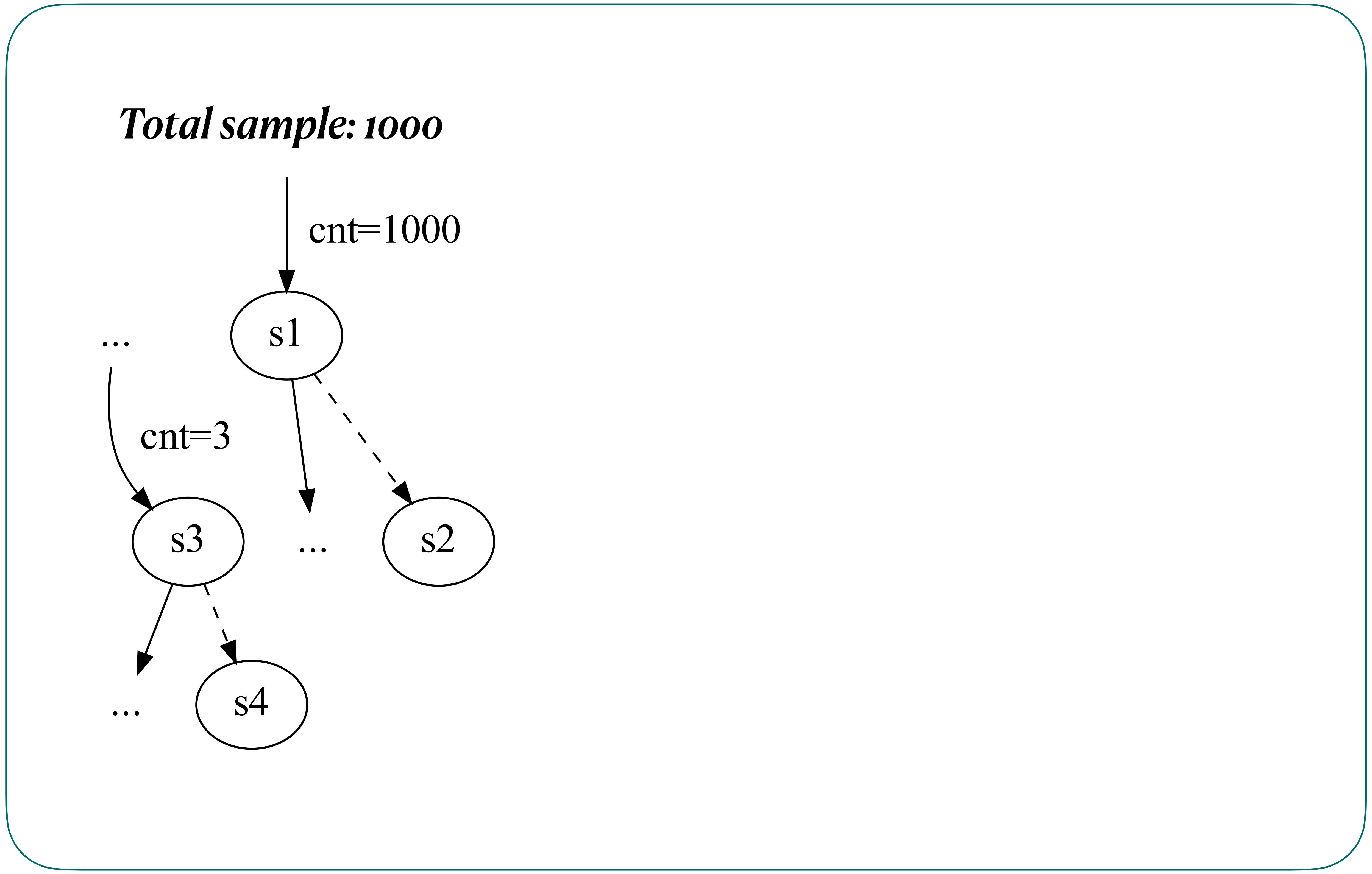


s_2 has larger chances of being reached than s_4

Black-box estimators are entirely unaware of the structural feature of the program.

Our Solution: Structure-aware Reachability Estimator

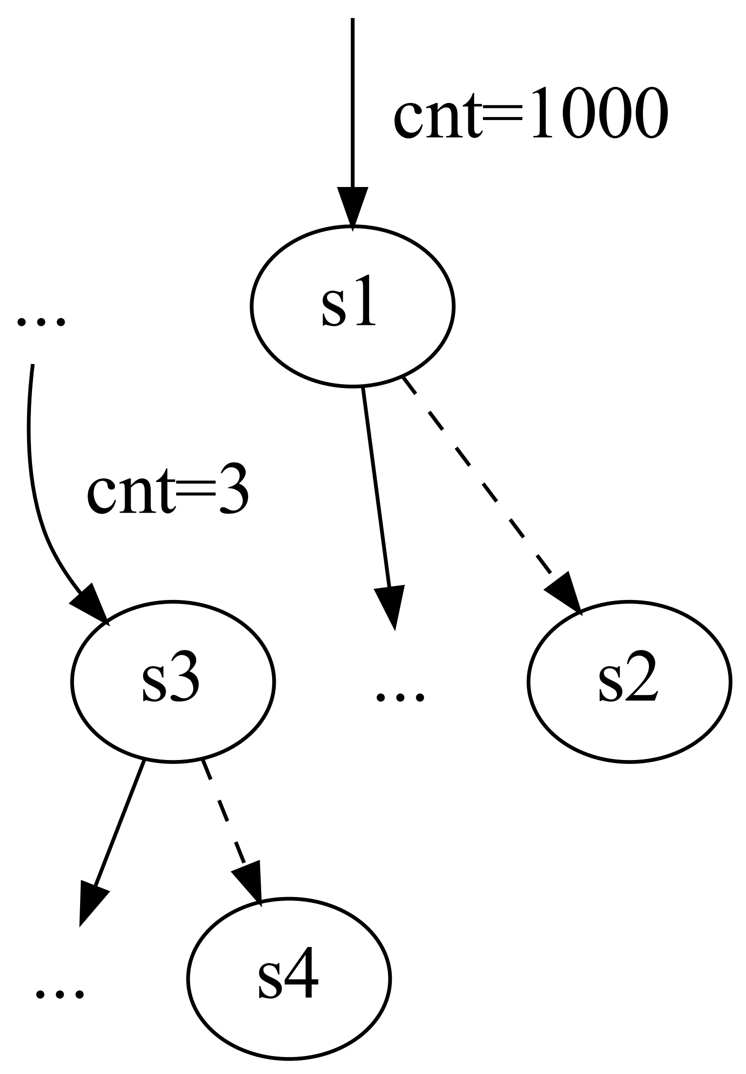
- Approach: reflect the *(control) dependence relation* between the program states.



Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



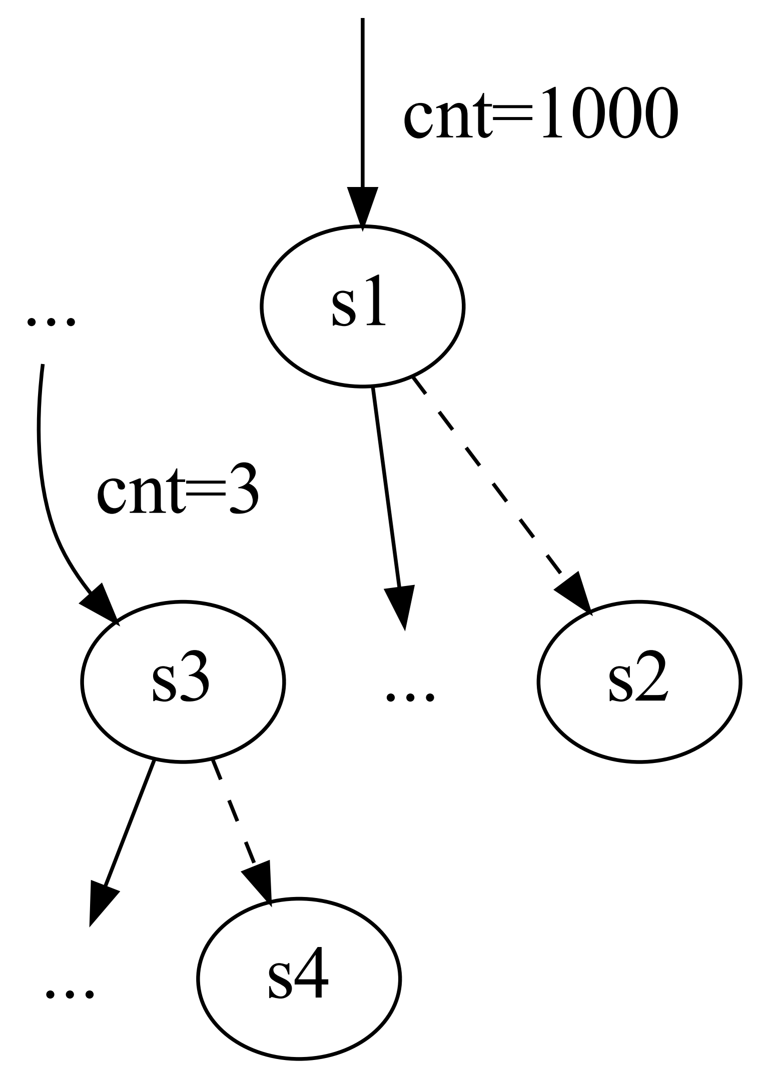
Blackbox estimator:

$$\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



Blackbox estimator:

$$\cancel{\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)}$$

Structure-aware:

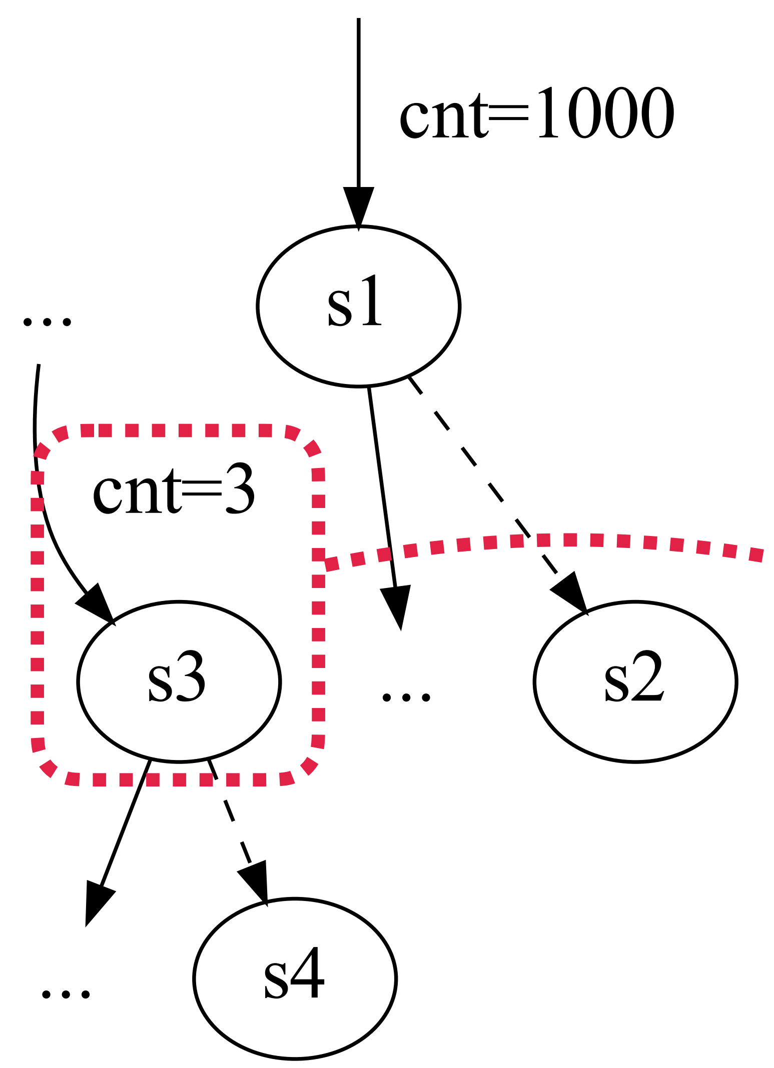
$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$
$$\text{where, } O' = \{o \in O \,|\, \mathrm{Reach}(o, s_1)\}$$

$$\hat{\Pr}_{St}(s_4) = \hat{\Pr}_{Emp}(s_3, O) \times \hat{\Pr}_{BB}(s_4, O'')$$
$$\text{where, } O'' = \{o \in O \,|\, \mathrm{Reach}(o, s_3)\}$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



Blackbox estimator:

$$\cancel{\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)}$$

Structure-aware:

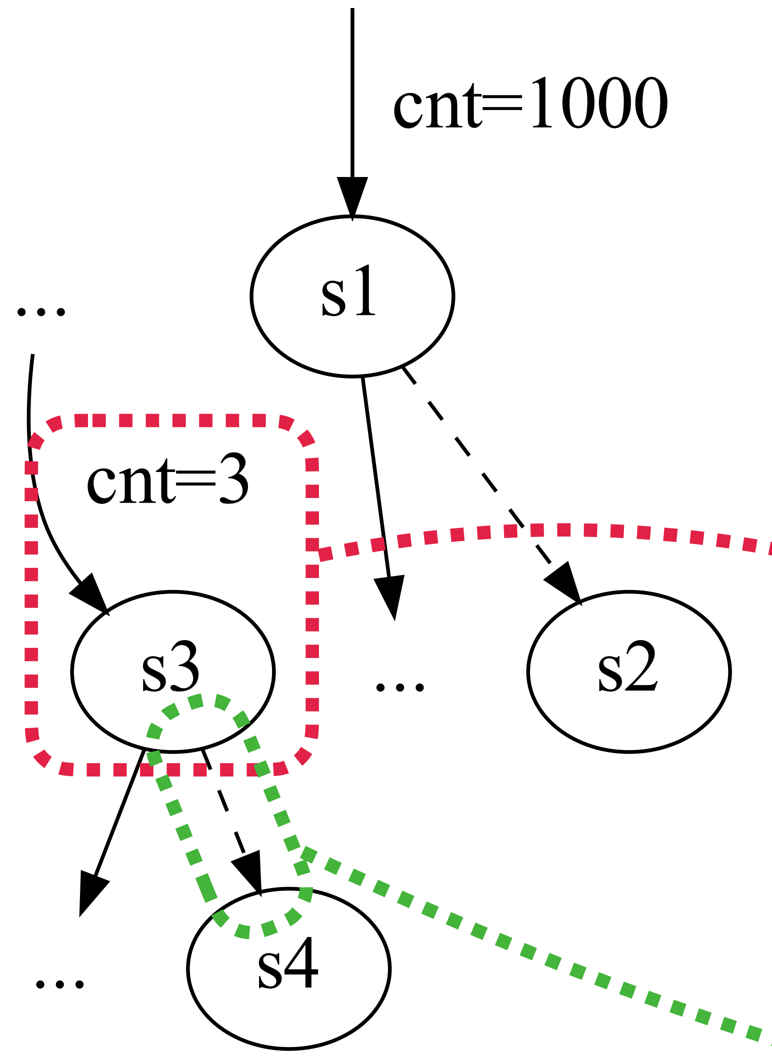
$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$
$$\text{where, } O' = \{o \in O \,|\, \text{Reach}(o, s_1)\}$$

$$\hat{\Pr}_{St}(s_4) = \overset{\text{Pr(Pred)}}{\boxed{\hat{\Pr}_{Emp}(s_3, O)}} \times \hat{\Pr}_{BB}(s_4, O'')$$
$$\text{where, } O'' = \{o \in O \,|\, \text{Reach}(o, s_3)\}$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



Blackbox estimator:

$$\cancel{\hat{\Pr}_{BB}(s_4, O)} = \hat{\Pr}_{BB}(s_2, O)$$

Structure-aware:

$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$

where, $O' = \{o \in O \mid \text{Reach}(o, s_1)\}$

Pr(Pred) Pr(Pred→Next)

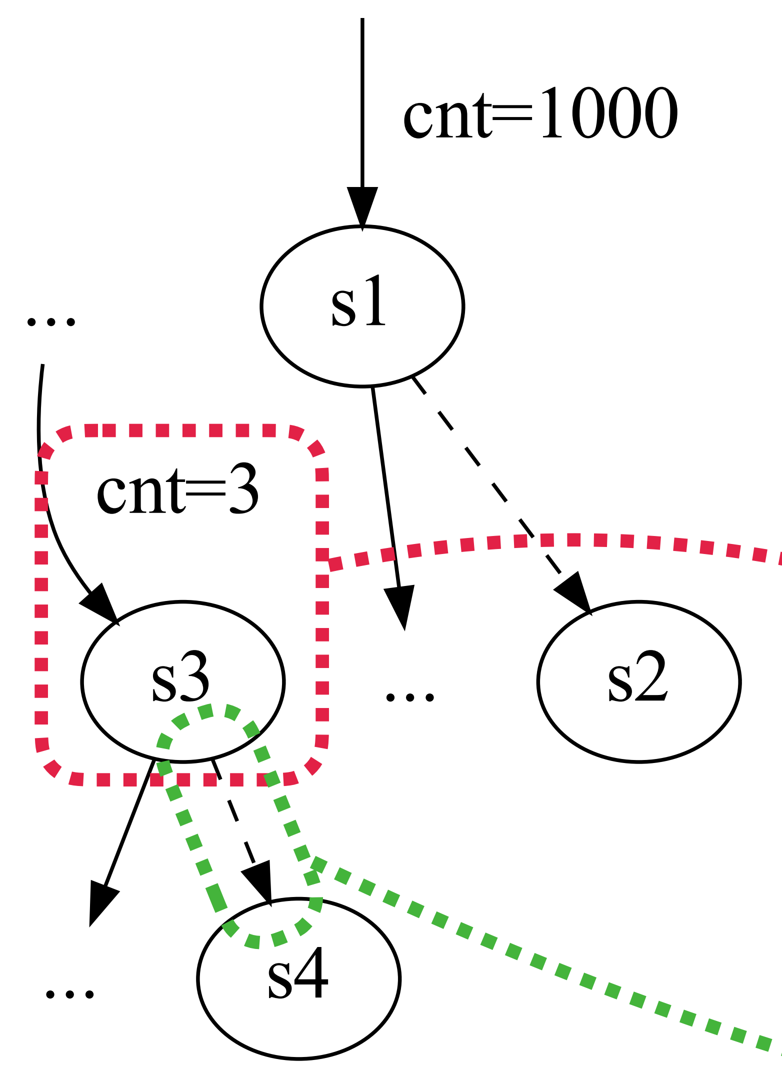
$$\hat{\Pr}_{St}(s_4) = \boxed{\hat{\Pr}_{Emp}(s_3, O)} \times \boxed{\hat{\Pr}_{BB}(s_4, O'')}$$

where, $O'' = \{o \in O \mid \text{Reach}(o, s_3)\}$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



Blackbox estimator:

$$\cancel{\hat{\Pr}_{BB}(s_4, O)} = \hat{\Pr}_{BB}(s_2, O)$$

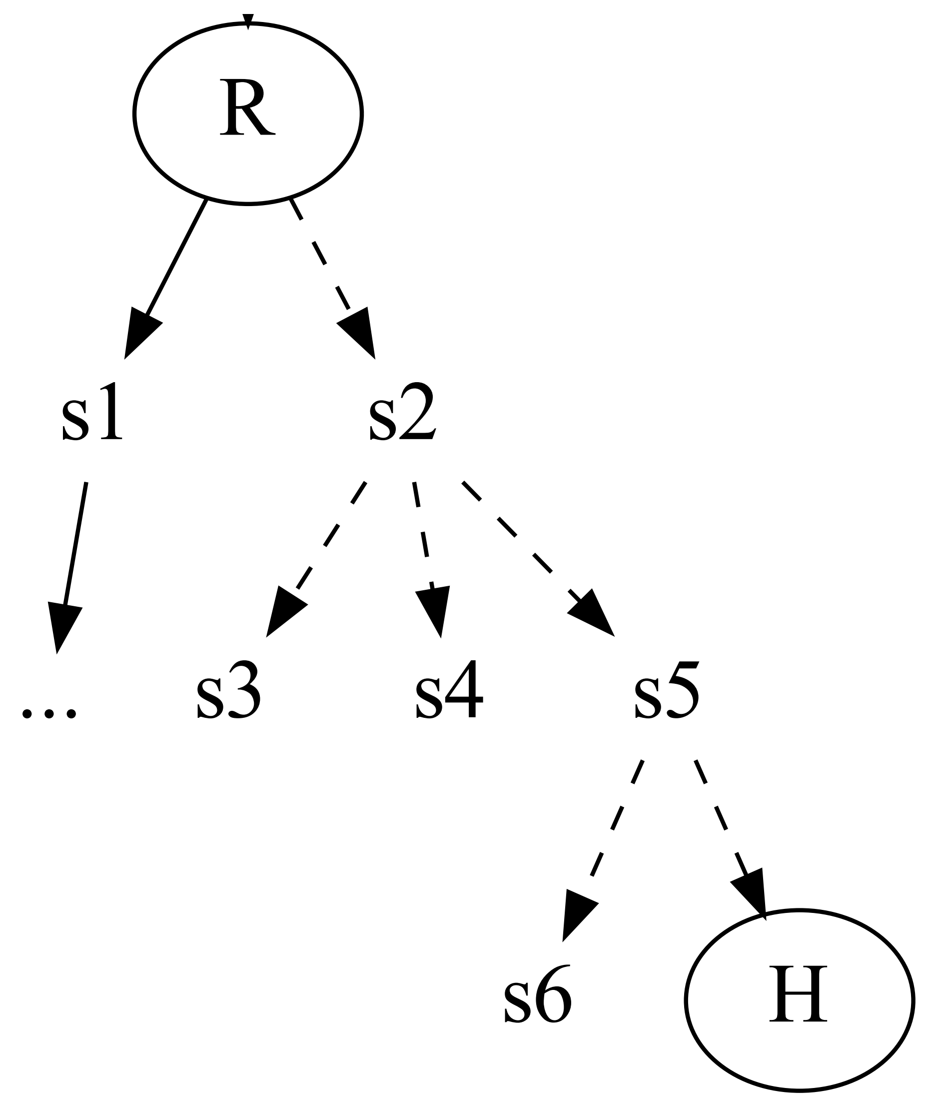
Structure-aware:

$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$

where, $O' = \{o \in O \,|\, \mathrm{Reach}(o, s_1)\}$

$$\hat{\Pr}_{St}(s_4) = \overbrace{\boxed{\hat{\Pr}_{Emp}(s_3, O)}}^{\text{Pr(Pred)}} \times \overbrace{\boxed{\hat{\Pr}_{BB}(s_4, O'')}}^{\text{Pr(Pred→Next)}}$$

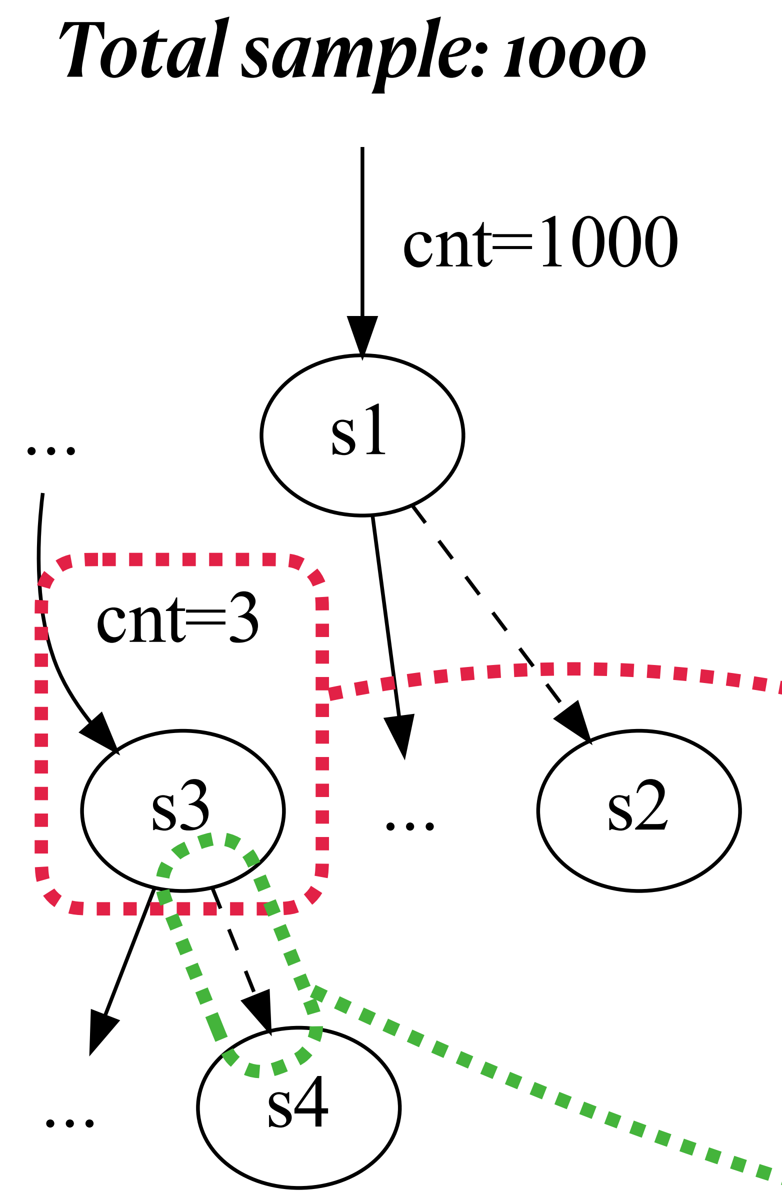
where, $O'' = \{o \in O \,|\, \mathrm{Reach}(o, s_3)\}$



$$\hat{\Pr}_{St}(H) = \hat{\Pr}_{Emp}(R, O) \times \hat{\Pr}_{BB}(s_2, O^R) \times \frac{1}{3} \times \frac{1}{2}$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.



Blackbox estimator:

$$\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)$$

Structure-aware:

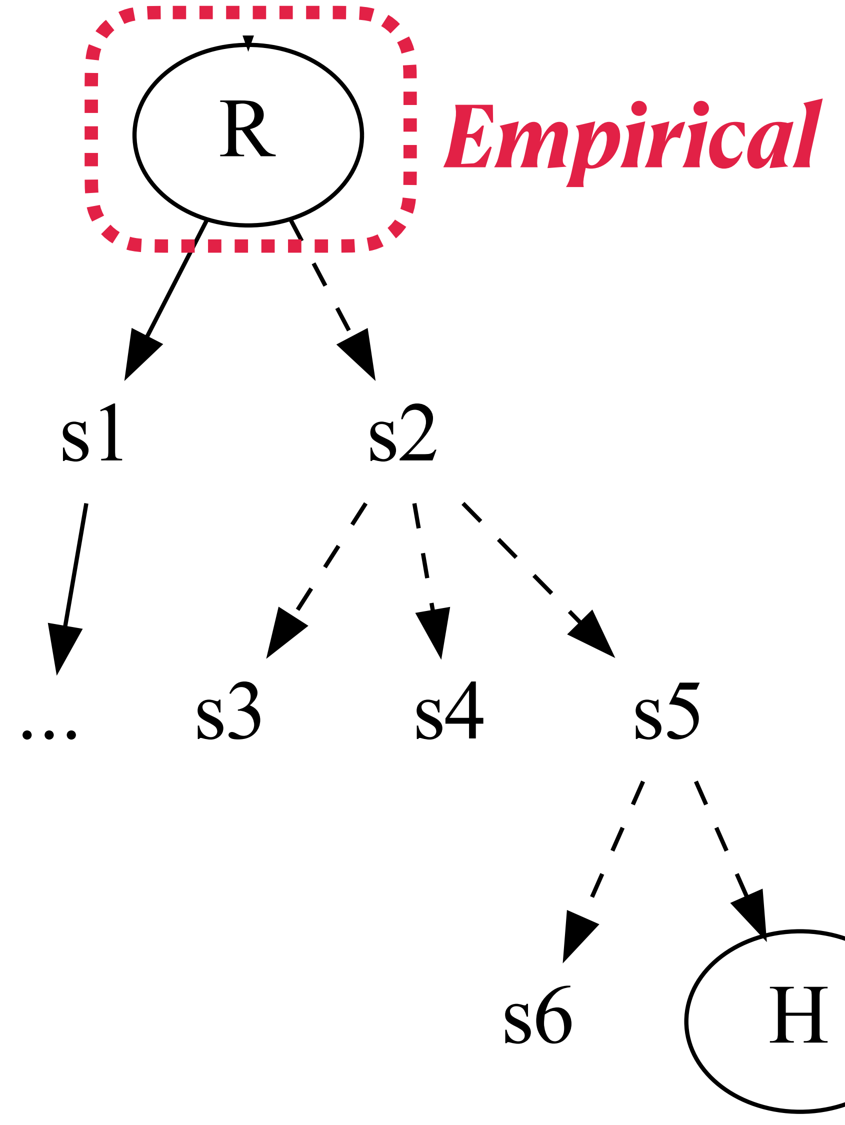
$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$

where, $O' = \{o \in O \mid \text{Reach}(o, s_1)\}$

Pr(Pred) Pr(Pred→Next)

$$\hat{\Pr}_{St}(s_4) = \hat{\Pr}_{Emp}(s_3, O) \times \hat{\Pr}_{BB}(s_4, O'')$$

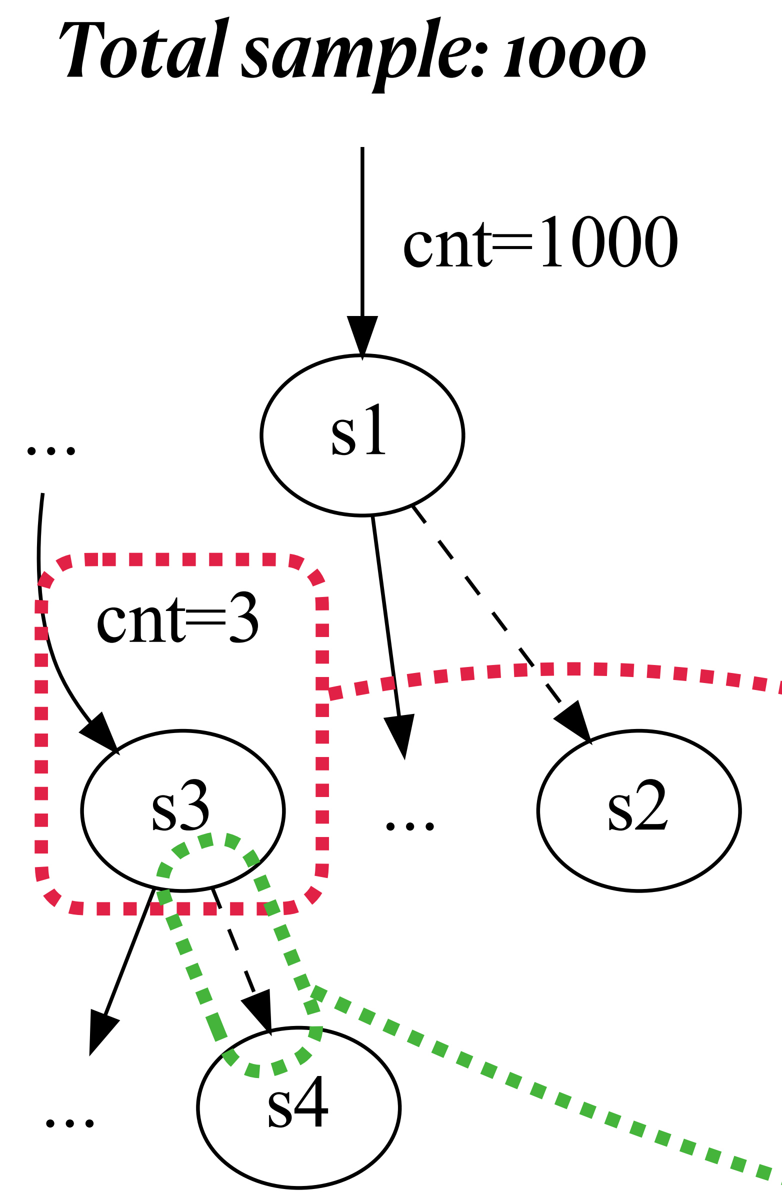
where, $O'' = \{o \in O \mid \text{Reach}(o, s_3)\}$



$$\hat{\Pr}_{St}(H) = \hat{\Pr}_{Emp}(R, O) \times \hat{\Pr}_{BB}(s_2, O^R) \times \frac{1}{3} \times \frac{1}{2}$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.



Blackbox estimator:

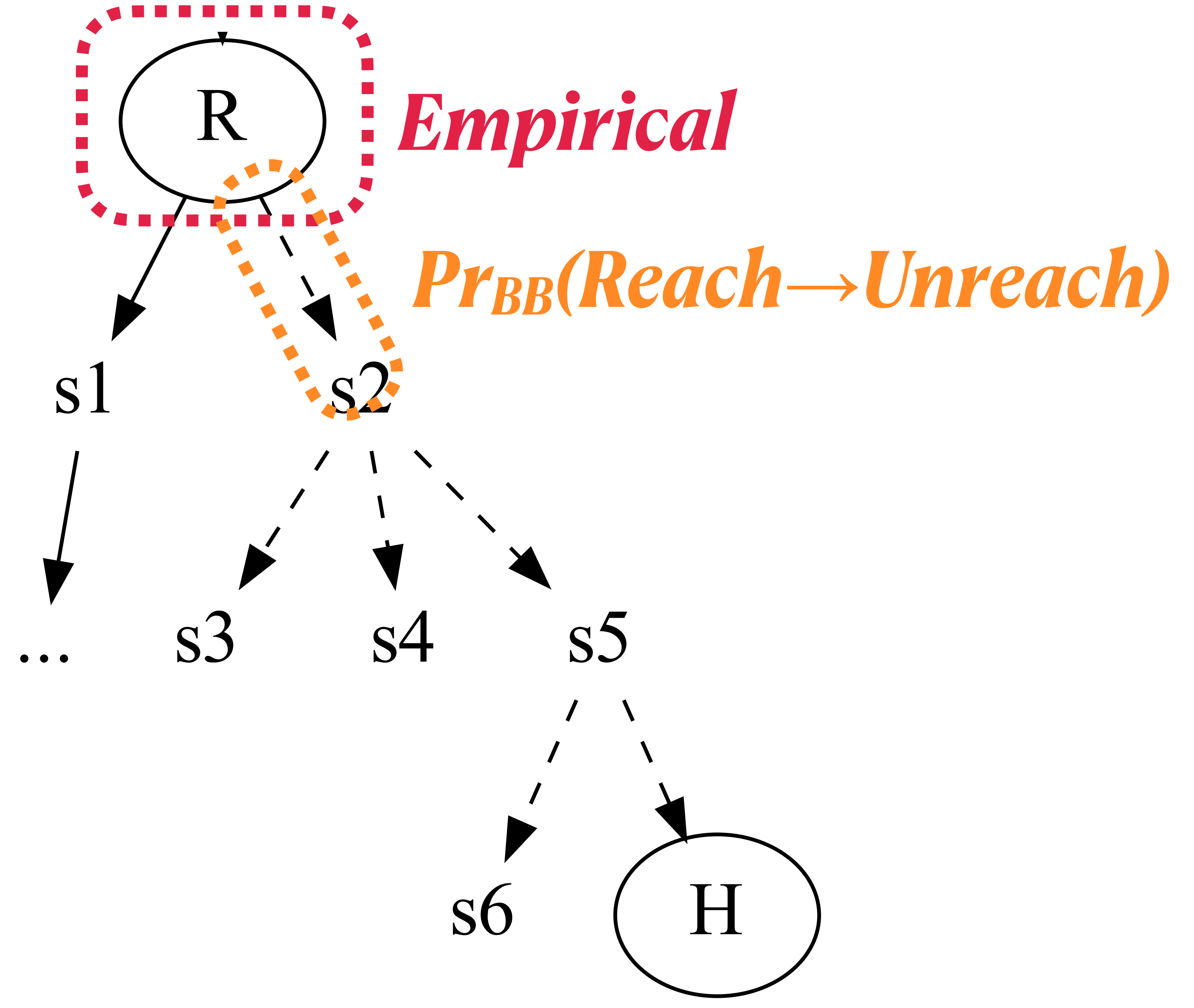
$$\cancel{\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)}$$

Structure-aware:

$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$
where, $O' = \{o \in O \mid \mathrm{Reach}(o, s_1)\}$

Pr(Pred) Pr(Pred→Next)

$$\hat{\Pr}_{St}(s_4) = \boxed{\hat{\Pr}_{Emp}(s_3, O)} \times \boxed{\hat{\Pr}_{BB}(s_4, O'')}$$
where, $O'' = \{o \in O \mid \mathrm{Reach}(o, s_3)\}$

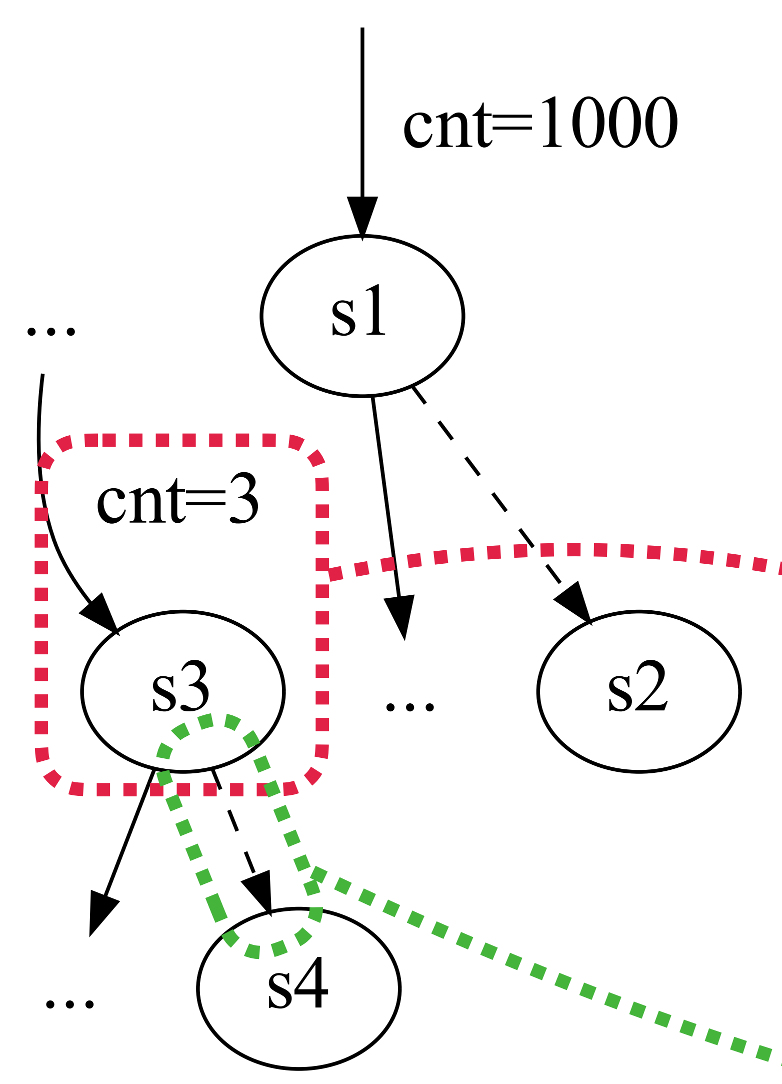


$$\hat{\Pr}_{St}(H) = \boxed{\hat{\Pr}_{Emp}(R, O)} \times \boxed{\hat{\Pr}_{BB}(s_2, O^R)} \times \frac{1}{3} \times \frac{1}{2}$$

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.

Total sample: 1000



Blackbox estimator:

$\hat{\Pr}_{BB}(s_4, O) = \hat{\Pr}_{BB}(s_2, O)$

Structure-aware:

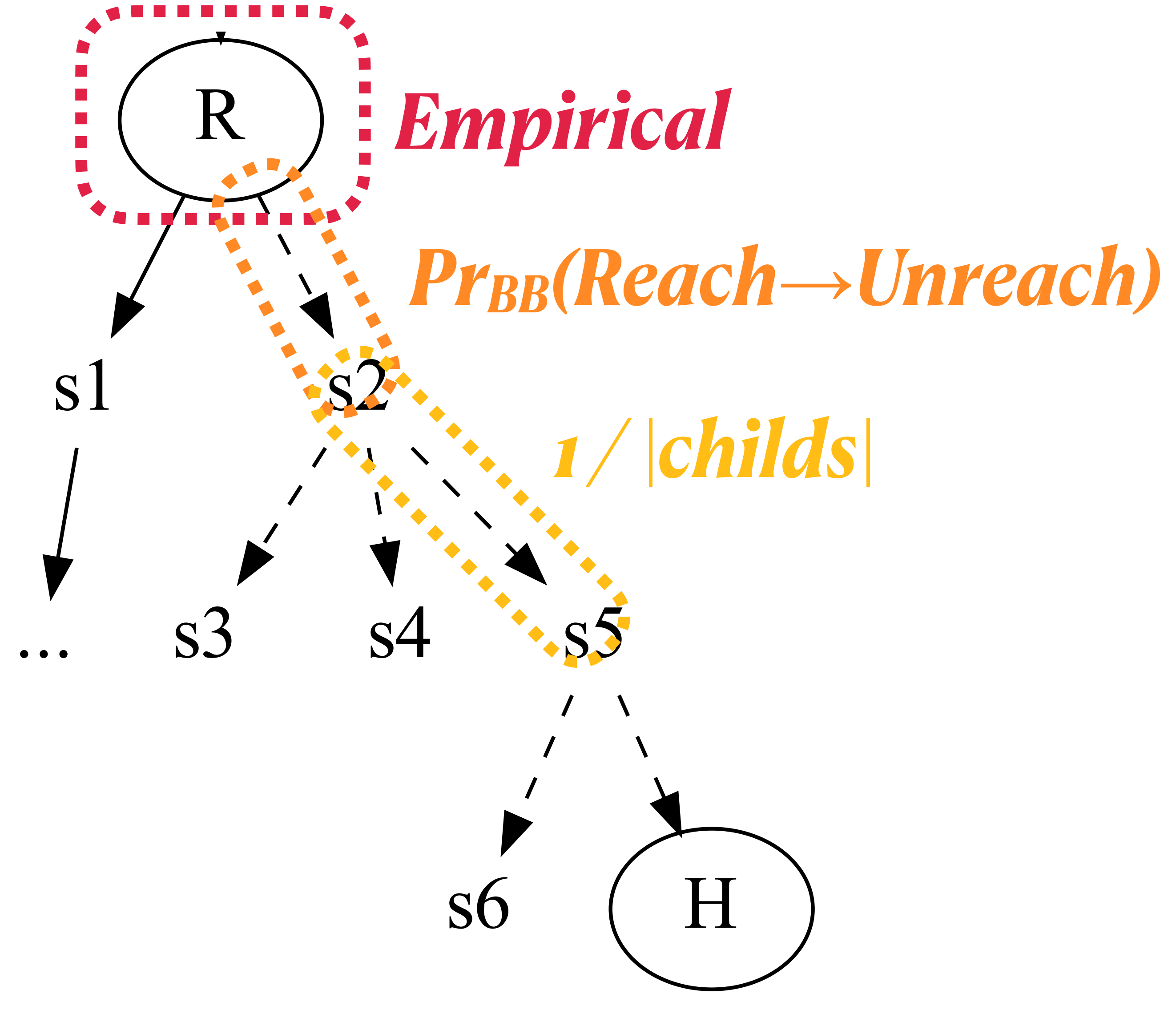
$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$

where, $O' = \{o \in O \mid \mathrm{Reach}(o, s_1)\}$

Pr(Pred) Pr(Pred→Next)

$\hat{\Pr}_{St}(s_4) = \boxed{\hat{\Pr}_{Emp}(s_3, O)} \times \boxed{\hat{\Pr}_{BB}(s_4, O'')}$

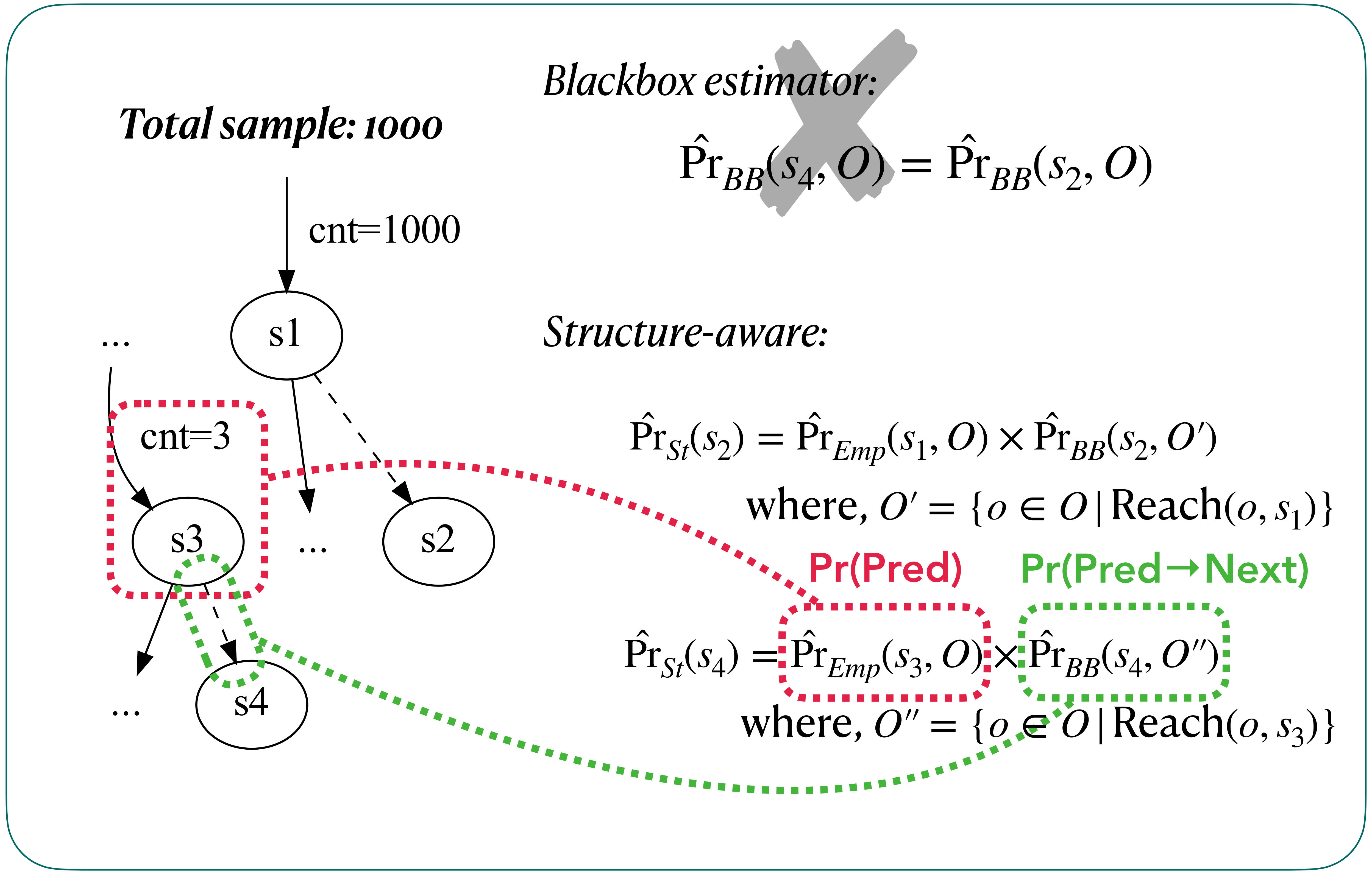
where, $O'' = \{o \in O \mid \mathrm{Reach}(o, s_3)\}$

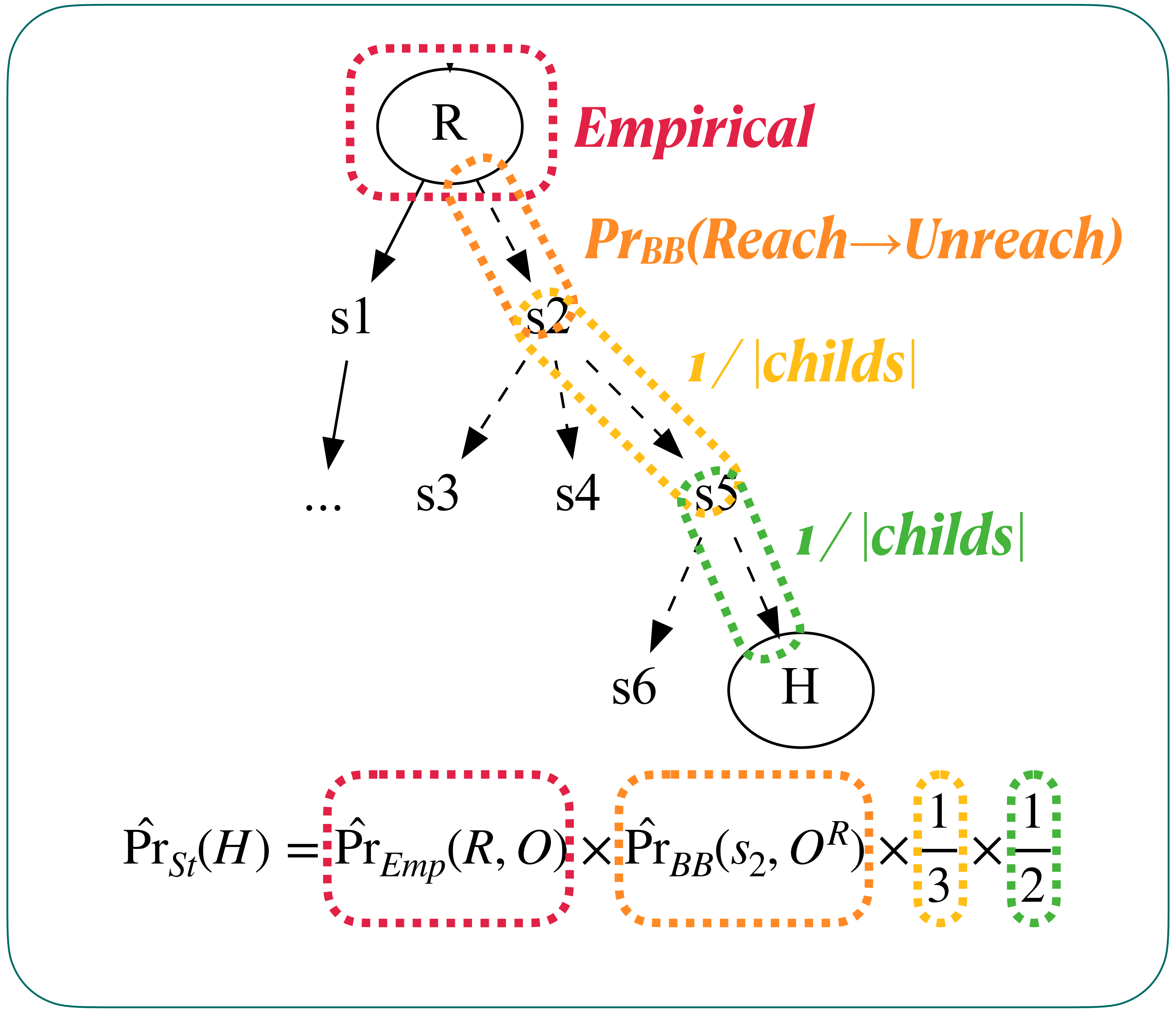


$\hat{\Pr}_{St}(H) = \boxed{\hat{\Pr}_{Emp}(R, O)} \times \boxed{\hat{\Pr}_{BB}(s_2, O^R)} \times \boxed{\frac{1}{3}} \times \frac{1}{2}$

Our Solution: Structure-aware Reachability Estimator

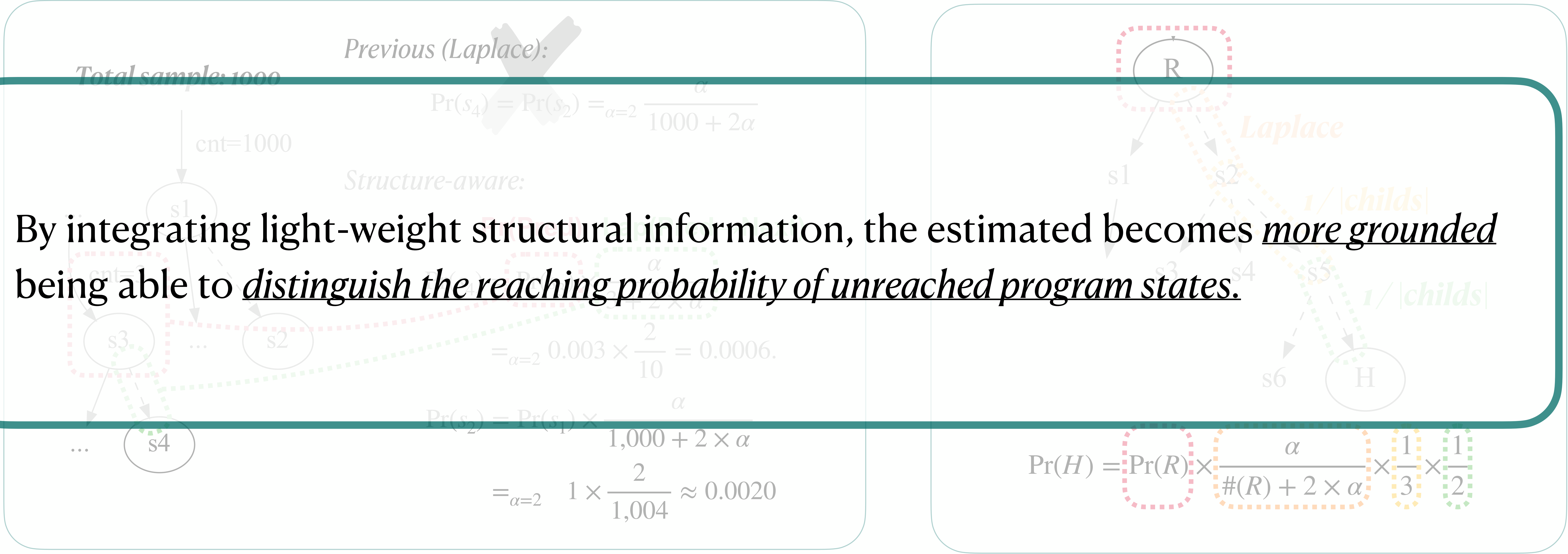
- Approach: reflect the *(control) dependence relation* between the program states.





Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.



By integrating light-weight structural information, the estimated becomes <u>*more grounded*</u> being able to <u>*distinguish the reaching probability of unreached program states.*</u>

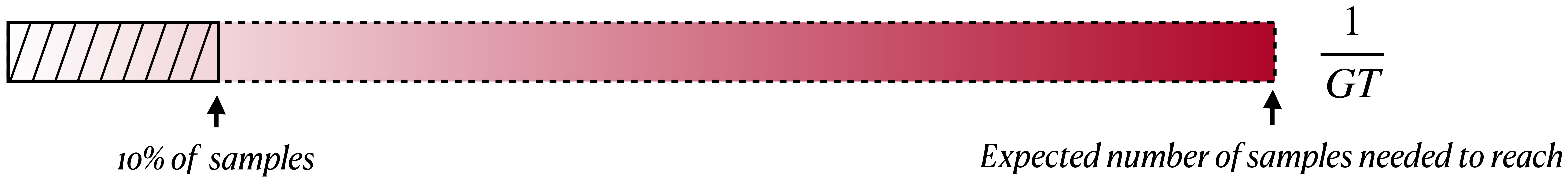
Evaluation

RQ 2. Blackbox estimator vs. Structure-aware estimator

- Subjects: 5 Subjects from Siemens suite
  + 5 Open-source C libraries
- Target state: hard-to-be-covered basic block

Program	NCLOC	# Func	# BB	GT
tcas	146	9	63	5.37E-04
schedule2	332	17	138	3.99E-04
totinfo	349	7	132	9.2E-04
printtokens2	438	19	198	7.82E-03
replace	534	21	228	2.73E-04
gif2png*	988	27	700	2.95E-04
jsoncpp	7,251	1,328	5,938	2.28E-03
jasper*	17,385	720	14,417	2.48E-04
readelf	22,347	477	18,578	1.99E-07
freetype2	44,686	1,635	27,521	8.25E-08

- Evaluation setting:

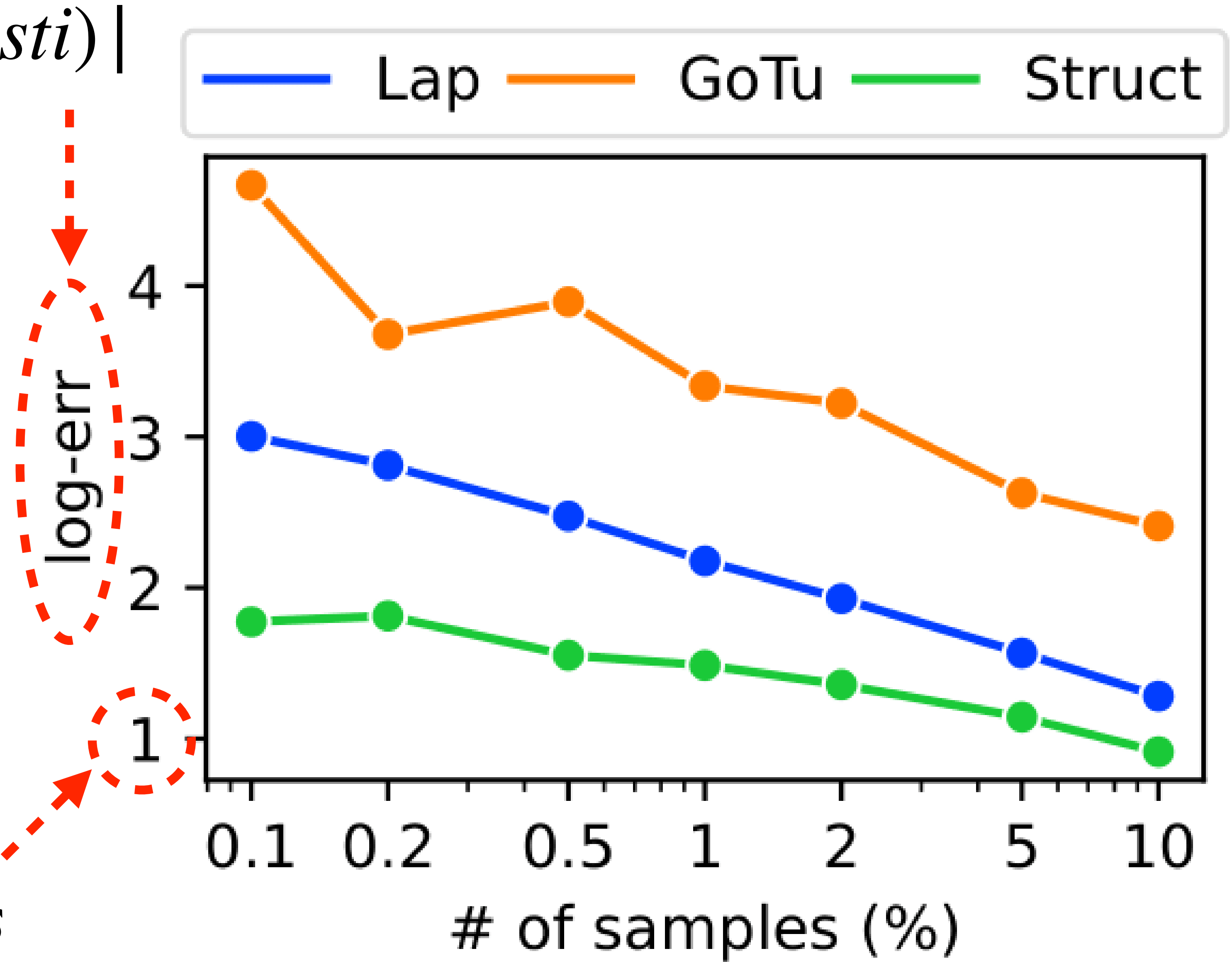


Blackbox Estimator vs Structure-aware Estimator

$|\log(GT) - \log(esti)|$

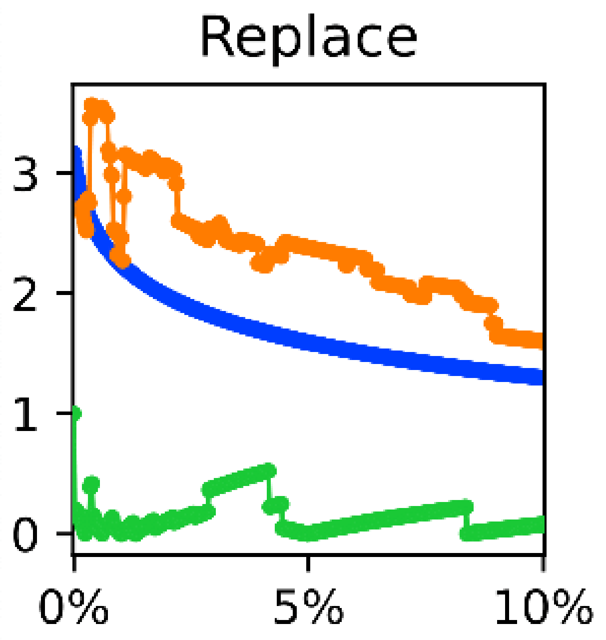
Sample size vs. Bias Curve

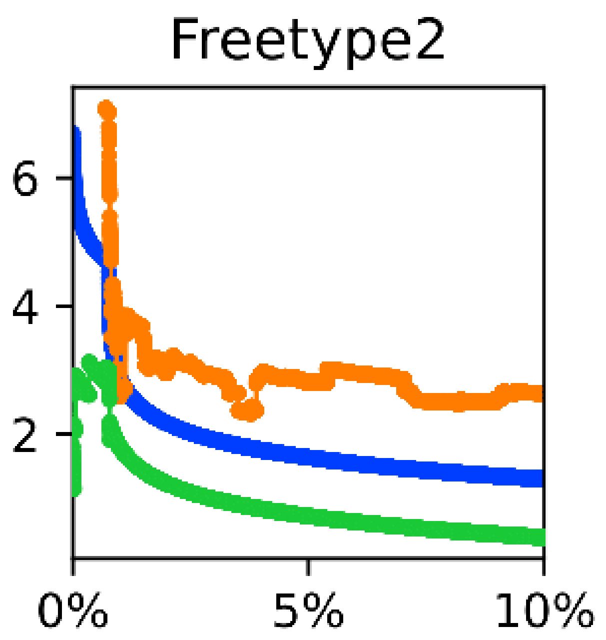
Blackbox



log-bias < 1 means one order of magnitude difference

Individual cases





- The ***structure-aware estimator*** performed ***significantly better*** than the blackbox estimators.

	Blackbox		Structure
Sample size	***Laplace***	***Good-Turing***	***Struct***
10 %	1.28	2.41	**0.91**
0.01 %	3.00	4.67	**1.77**

log-bias

Statistical Reachability Analysis

Seongmin Lee
Max Planck Institute for Security and Privacy
Bochum, Germany
seongmin.lee@mpi-sp.org

Marcel Böhme
Max Planck Institute for Security and Privacy
Bochum, Germany
marcel.boehme@acm.org

ABSTRACT

Given a target program state (or statement) s, what is the probability that an input reaches s? This is the quantitative reachability analysis problem. For instance, quantitative reachability analysis can be used to approximate the reliability of a program (where s is a bad state). Traditionally, quantitative reachability analysis is solved as a model counting problem for a formal constraint that represents the (approximate) reachability of s along paths in the program, i.e., probabilistic reachability analysis. However, in preliminary experiments, we failed to run state-of-the-art probabilistic reachability analysis on reasonably large programs.

In this paper, we explore statistical methods to estimate reachability probability. An advantage of statistical reasoning is that the size and composition of the program are insubstantial as long as the program can be executed. We are particularly interested in the error compared to the state-of-the-art probabilistic reachability analysis. We realize that existing estimators do not exploit the inherent structure of the program and develop structure-aware estimators to further reduce the estimation error given the same number of samples. Our empirical evaluation on previous and new benchmark programs shows that (i) our statistical reachability analysis outperforms state-of-the-art probabilistic reachability analysis tools in terms of accuracy, efficiency, and scalability, and (ii) our structure-aware estimators further outperform (blackbox) estimators that do not exploit the inherent program structure. We also identify multiple program properties that limit the applicability of the existing probabilistic analysis techniques.

CCS CONCEPTS

• **Theory of computation → Program analysis**; • **Mathematics of computing** → *Bayesian computation*.

KEYWORDS

Quantitative reachability analysis, Statistical reachability analysis, Reaching probability, Markov chain

ACM Reference Format:
Seongmin Lee and Marcel Böhme. 2023. Statistical Reachability Analysis. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3611643.3616268

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3616268

1 INTRODUCTION

The traditional assessment of the reachability of a program state provides only a true-false answer: either the state is reachable (e.g., the program may crash for some input) or not (e.g., it never crashes for any input). Due to the undecidability of the analysis problem [16] and the restricted expressiveness of the analysis result, such a binary answer provides only limited information. Instead of a binary answer, *quantitative reachability analysis* provides the probability of how likely a certain program state is reached given the workload of the program. Such a *quantitative* measure of reachability can provide more comprehensive information about the program semantics. For instance, it can estimate how probable is to reach a crashing state under normal workload, which can be critical information for software reliability/security/maintenance.

The typical method considered for quantitative reachability analysis is called *probabilistic reachability analysis* [27], which *analytically computes* the reaching probability directly from the source code. Probabilistic Symbolic Execution (PSE), the pioneering work by Geldenhuys et al. [12], computes the reaching probability of a program state by finding all the path conditions to reach the state using symbolic execution and counting the number of inputs satisfying the path conditions using model counting; the sum of the probabilities becomes the exact reaching probability of the program state. As PSE may suffer from scalability issues for a large and complex program, many follow-up works have been proposed to improve the scalability of probabilistic reachability analysis [11, 13]. Most recently, Saha et al. proposed PReach which computes the reaching probability using branch-level probability information [27].

When facing a problem too complex for the analytical method, especially when it is unmanageable to compute a quantity exactly, a sampling-based statistical method can be used to overcome the limitation [4]. It is well-known that Monte Carlo methods have been successfully applied to numerous problems across various fields, including natural sciences [10] and engineering [23], where the solution is intractable for analytic computation. Recently, in the context of program analysis, Liyanage et al. [21] proposed a statistical method to approximate the number of elements that can be reached by actual program execution, which, previously, can only be upper-bounded by static analysis.

This work explores how the statistical method can be applied to quantitative reachability analysis. We propose a *statistical reachability analysis*, which tackles the quantitative reachability analysis problem with random sampling and statistical modeling. The main issue of statistical reachability analysis is how to estimate the reaching probability of a certain program state that has not yet been observed in the sampling process. To overcome this issue, we first suggest a naive approach of using two well-known estimators, Laplace smoothing and Good-Turing estimator [15], that can estimate the non-zero probability of unseen events from the

- ***Statistical Reachability Analysis.***
  Seongmin Lee and Marcel Böhme. ESEC/FSE 2023.
  - By integrating lightweight structural information, statistical reaching probability estimation becomes ***more grounded,*** being able to ***distinguish the reaching probability of unreached program states.***

Missing Mass

*What is the **probability** of observing*
***a new coverage or a new bug**?*

Extrapolation

How much more can I achieve
***if I spend** X **more time** here?*

Present ***advanced extensions*** *to adopt statistical methods for more* ***realistic testing scenarios**.*

2

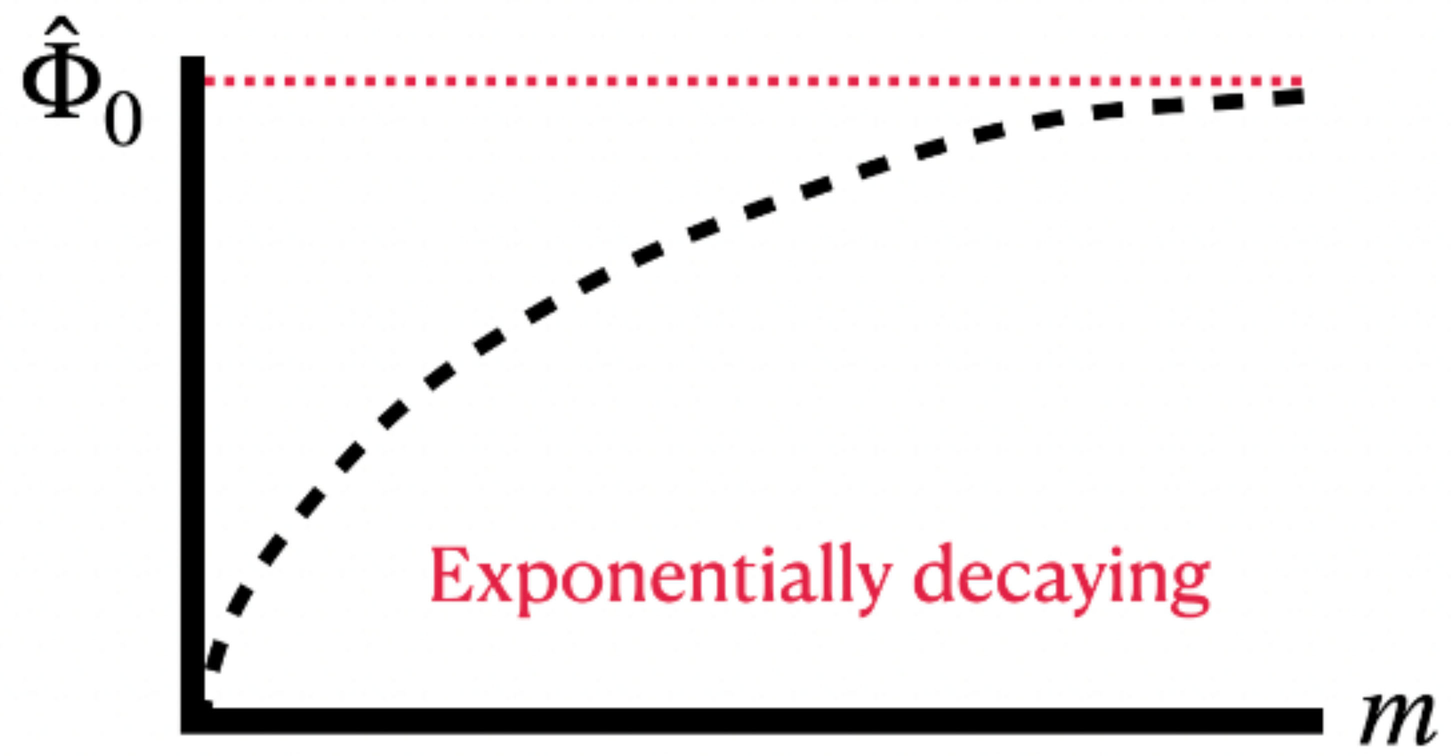
Statistically Extrapolating the Fuzzing Campaign

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^{m} \right]$$



Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

Statistically Extrapolating the Fuzzing Campaign

Without Extrapolation

```
                     american fuzzy lop 2.44b (djpeg)

--------------------------------------------------------------------------
|        run time : 0 days, 12 hrs, 0 min, 5 sec  |   cycles done : 53    |
|   last new path : 0 days, 0 hrs, 17 min, 44 sec | current paths : 4944  |
| last uniq crash : none seen yet                 |  uniq crashes : 0     |
                                    . . .
```

12 hours of running, the last new path was 17 minutes ago.
... should I stop this fuzzing?

VS

With Extrapolation

```
                 extrapolation edition yeah! (djpeg)

--------------------------------------------------------------------------
    residual risk : 7·10^-06                 | total inputs : 63.6M  |
    path coverage : 77.6% paths covered      |   singletons : 447    |
discover new path : 0 hrs, 1 min, 36 sec     |   doubletons : 70     |
                    142k new inputs needed   |                       |
```

A new path will come in 2 minutes? Let's keep going!

```
                 extrapolation edition yeah! (djpeg)

--------------------------------------------------------------------------
    residual risk : 8·10^-07                 | total inputs : 124.8M |
    path coverage : 97.9% paths covered      |   singletons : 95     |
discover new path : 0 hrs, 15 min, 9 sec     |   doubletons : 42     |
                    1.3M new inputs needed   |                       |
```

1/4 hour is needed for the next path? Let's stop!

Extrapolation gives richer information for the ***stopping criteria*** *for the fuzzing campaign*

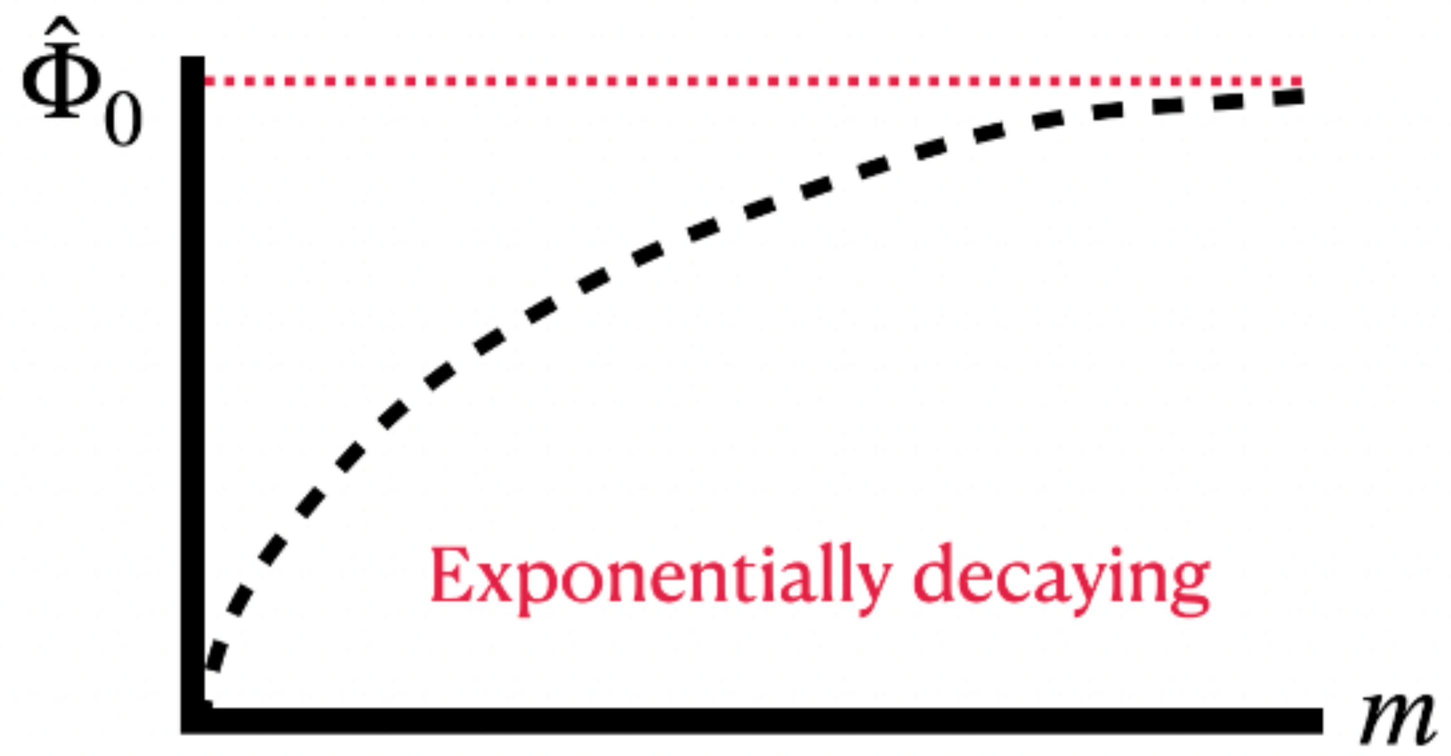
However, there is a hidden assumption:

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

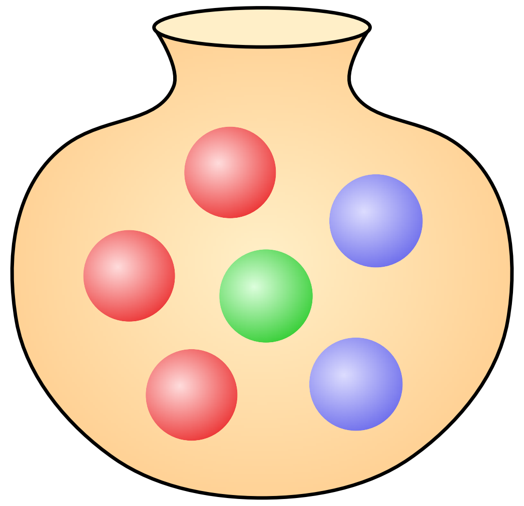
$$\hat{\Delta}(m) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{n\hat{\Phi}_0 + \Phi_1} \right)^{m} \right]$$



Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41
Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

However, there is a hidden assumption:

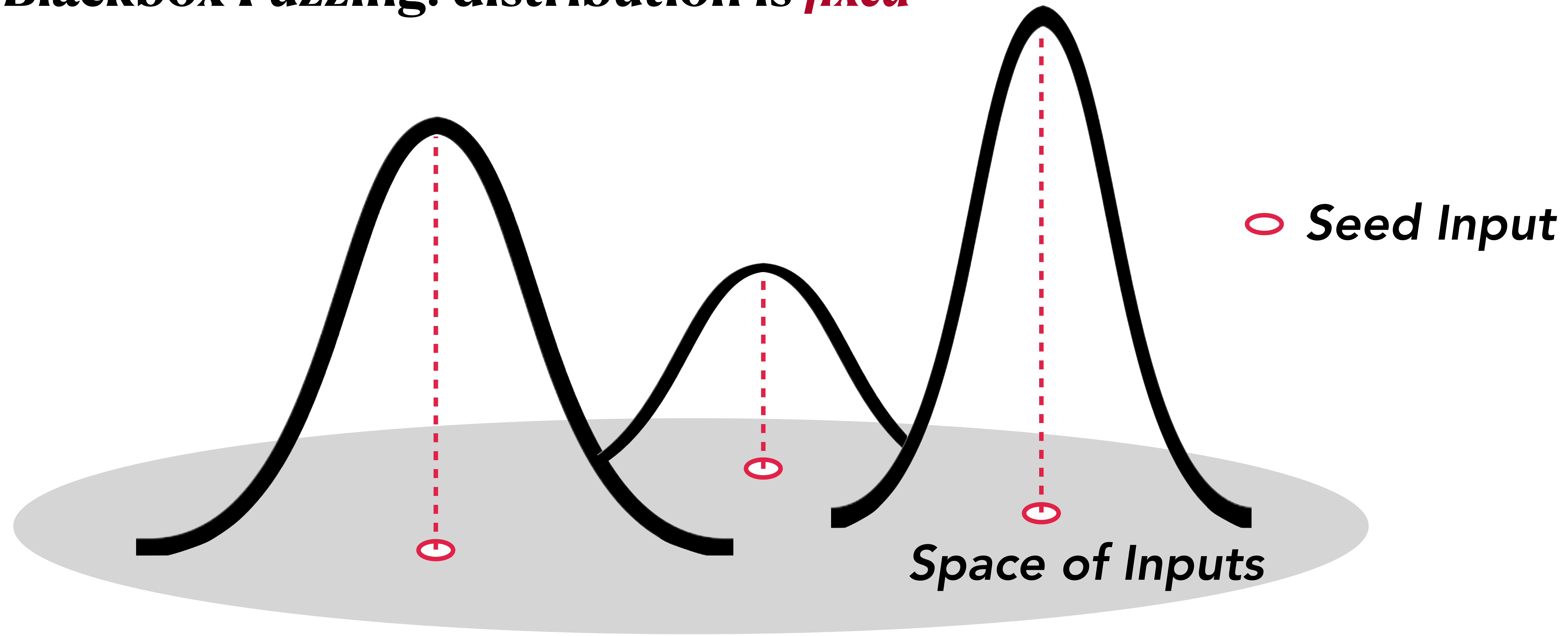
Let's say there is an urn filled with colored balls. The probability of picking the ball of color $i = p_i$. Let's say we picked n balls from the urn.



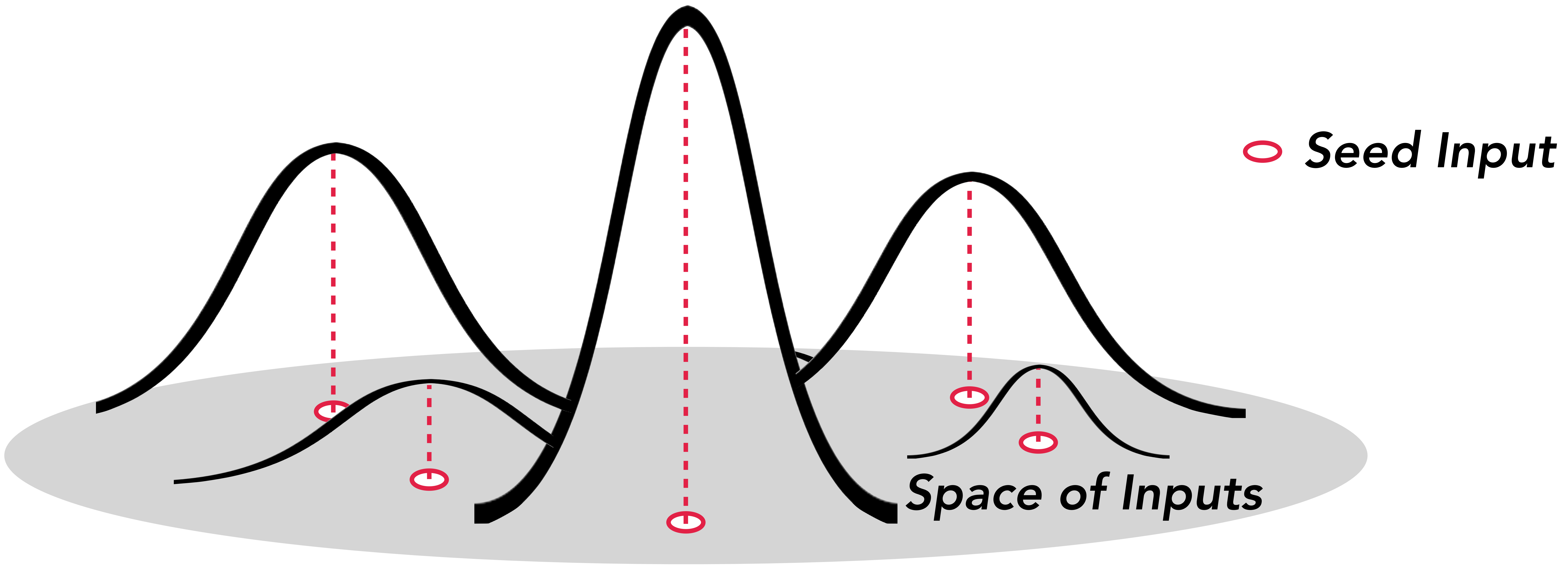
Assumption:

"The sampling distribution does not change."

Blackbox Fuzzing: distribution is *fixed*

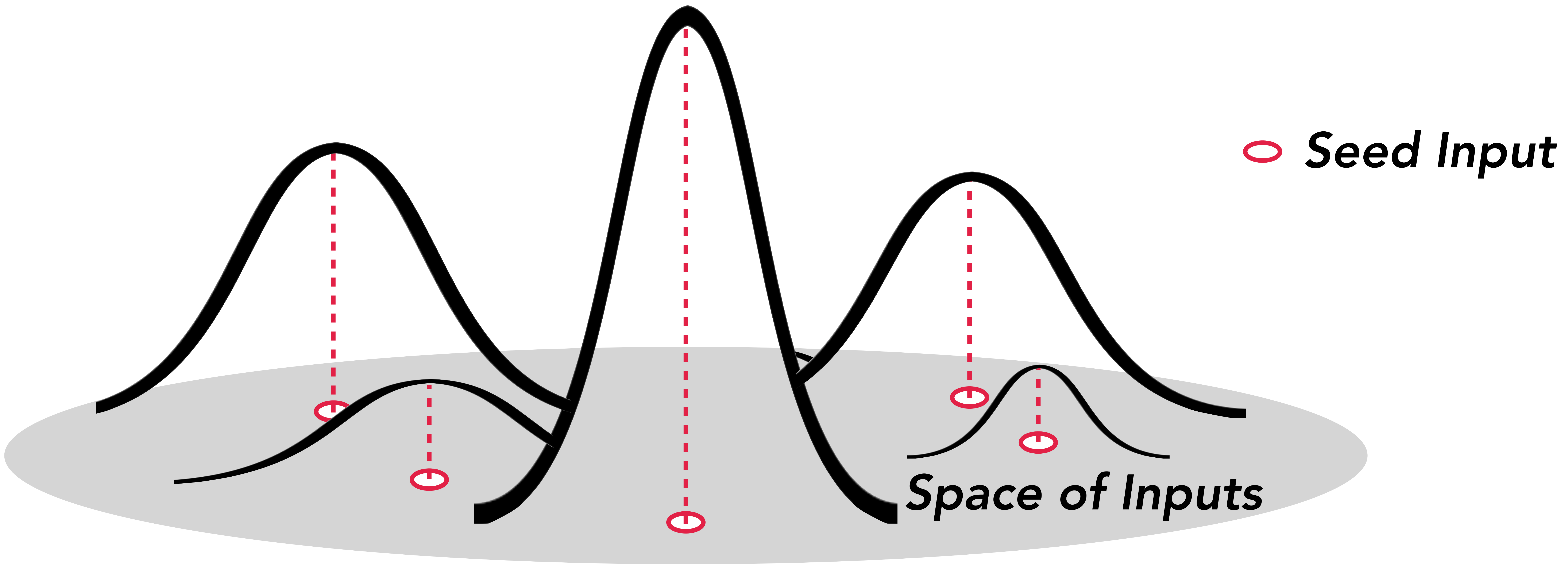


Greybox Fuzzing: distribution *changes* as time goes on

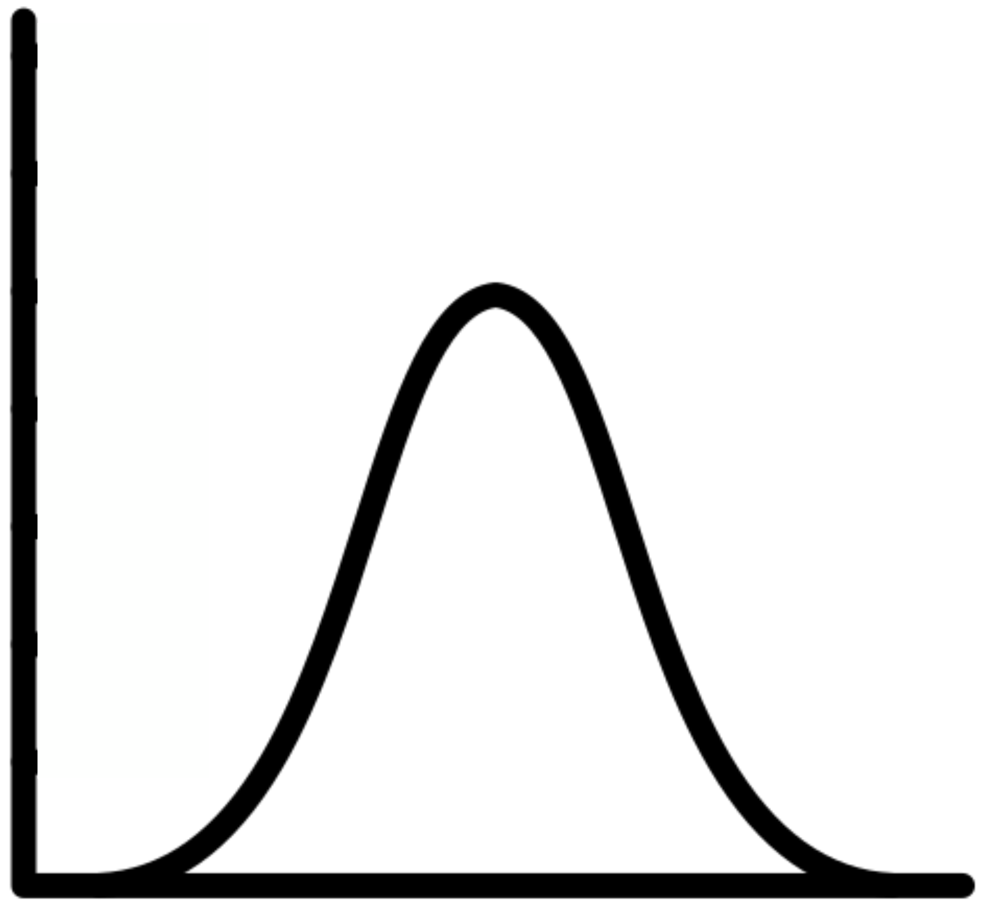


:= *Adaptive bias*

Greybox Fuzzing: distribution *changes* as time goes on

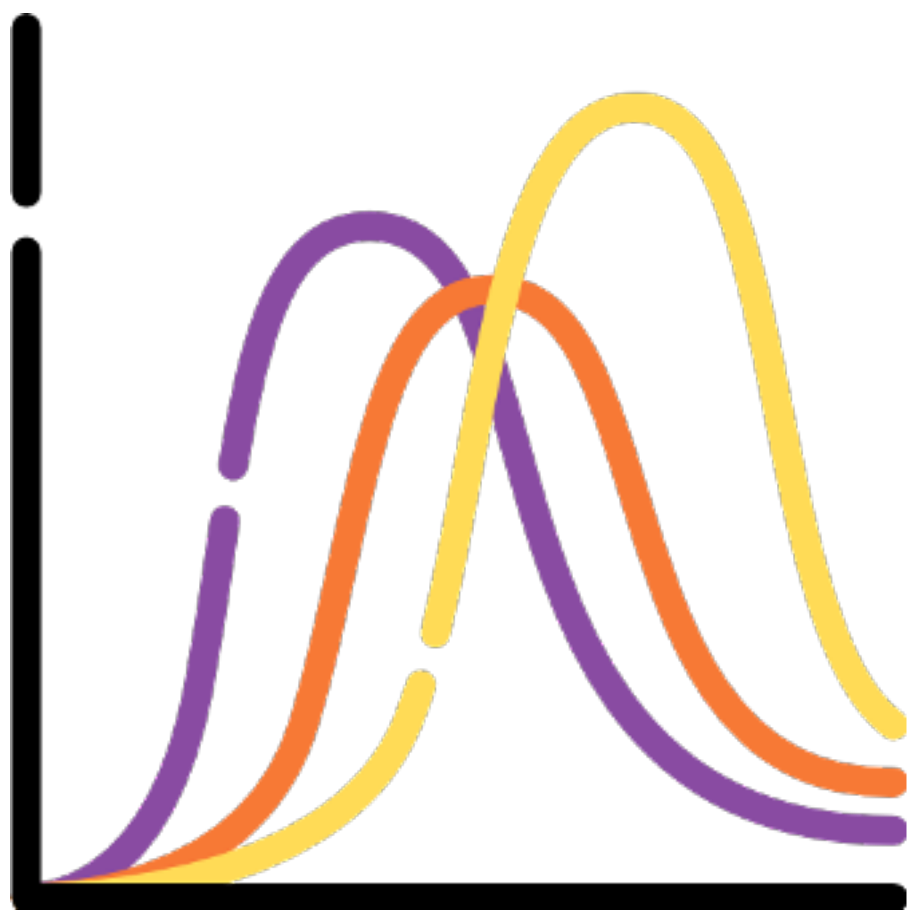


Blackbox

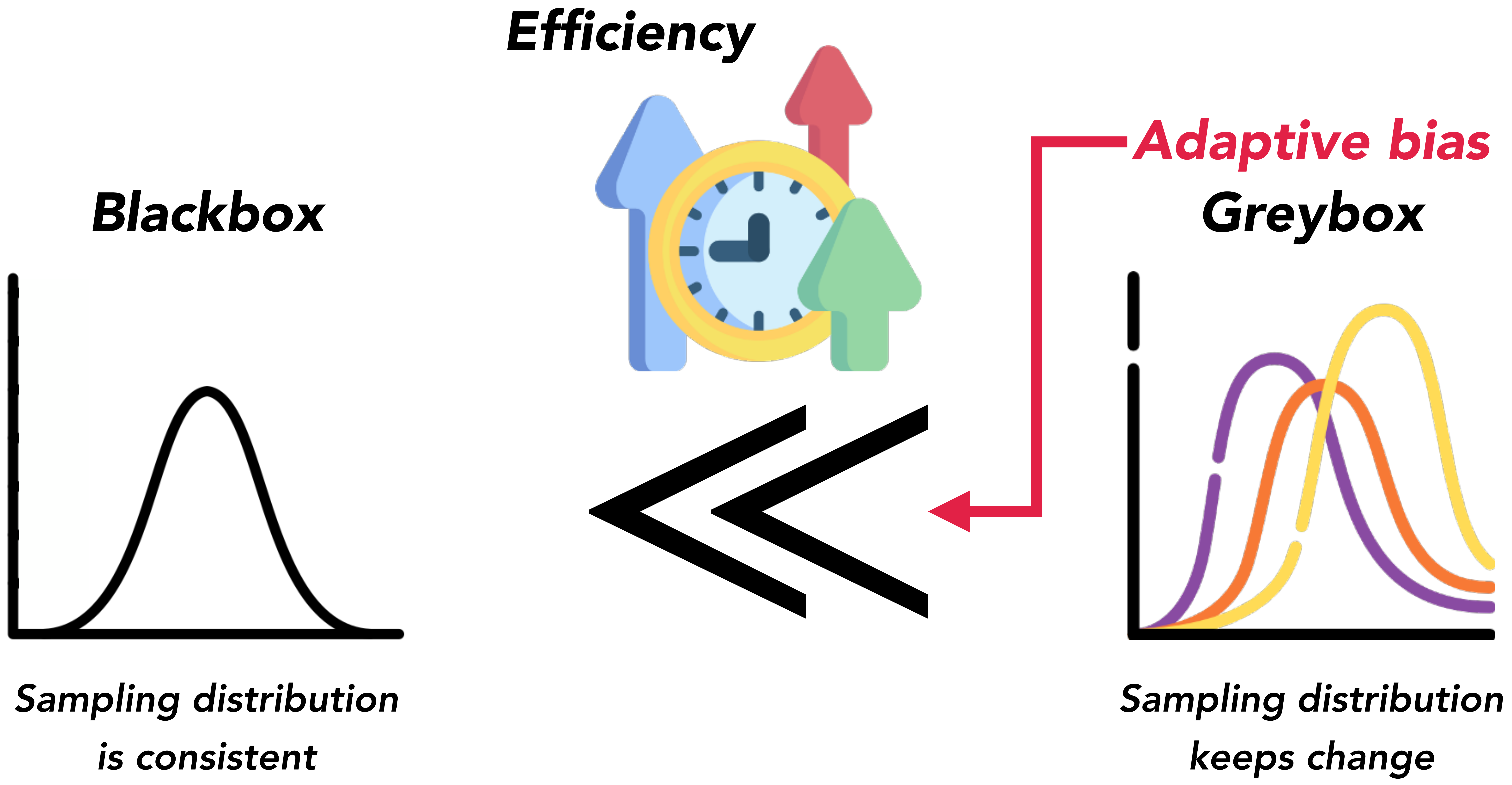


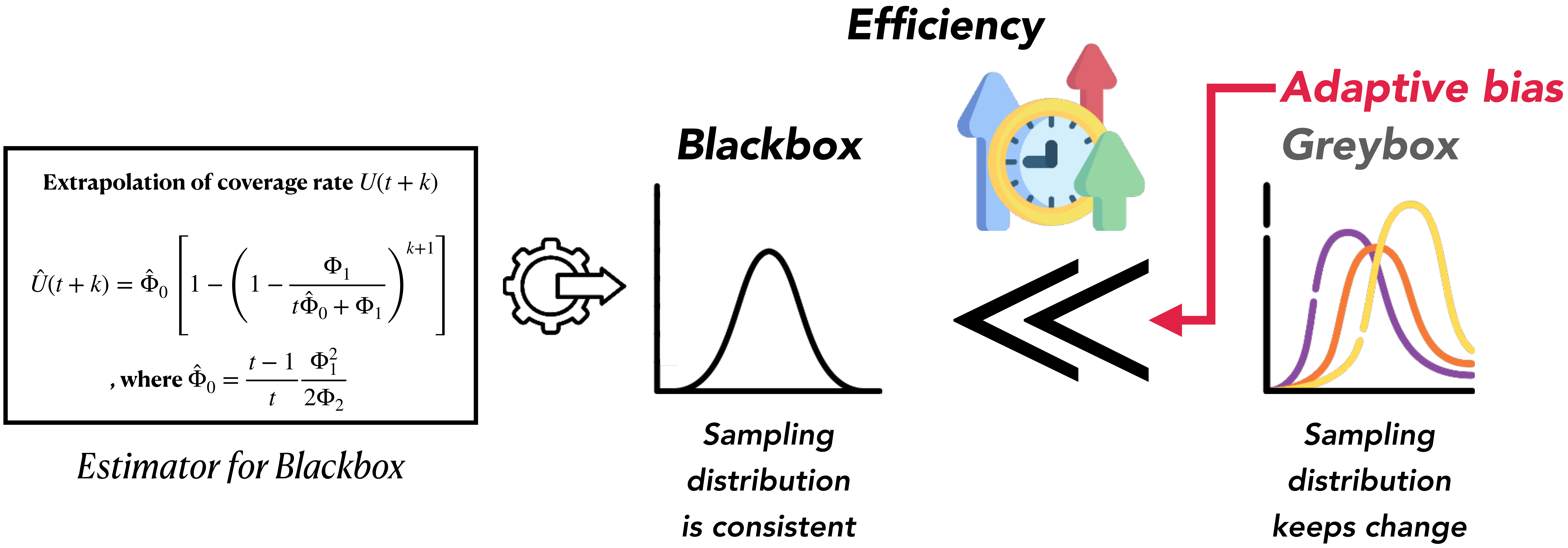
Sampling distribution is consistent

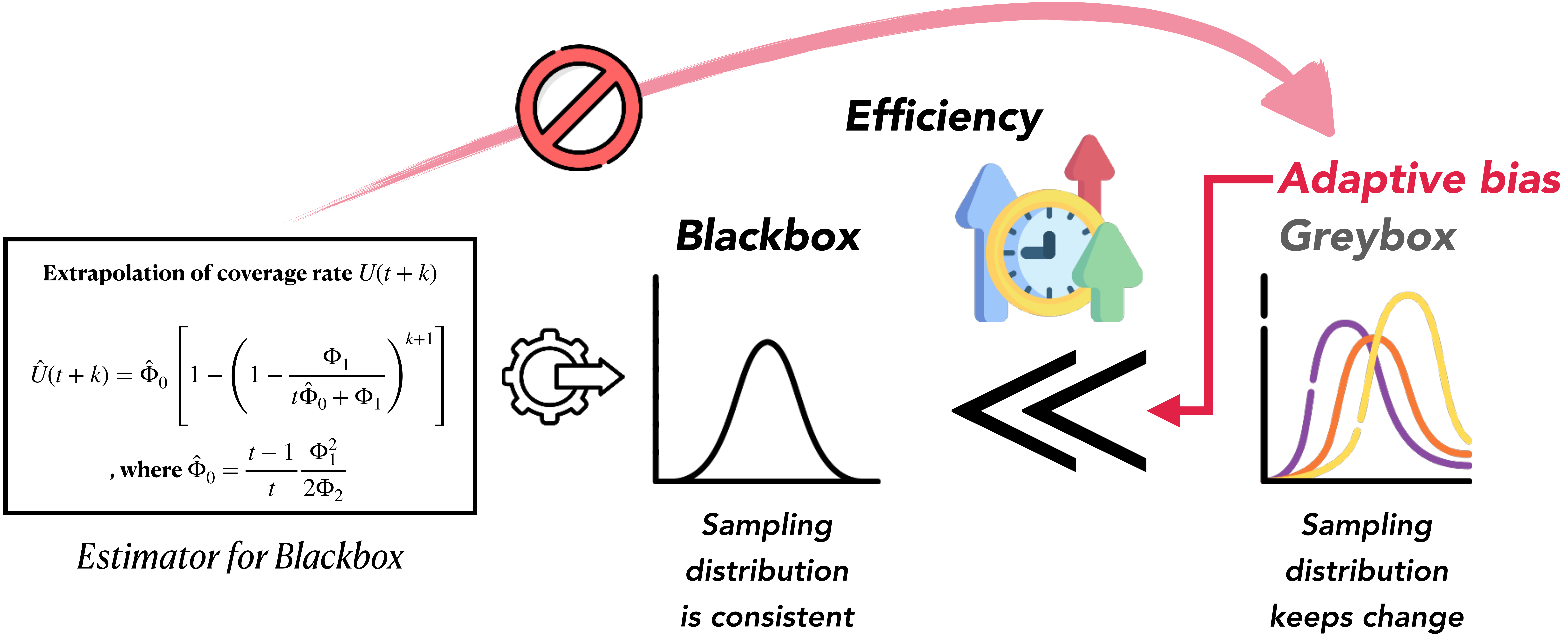
Greybox

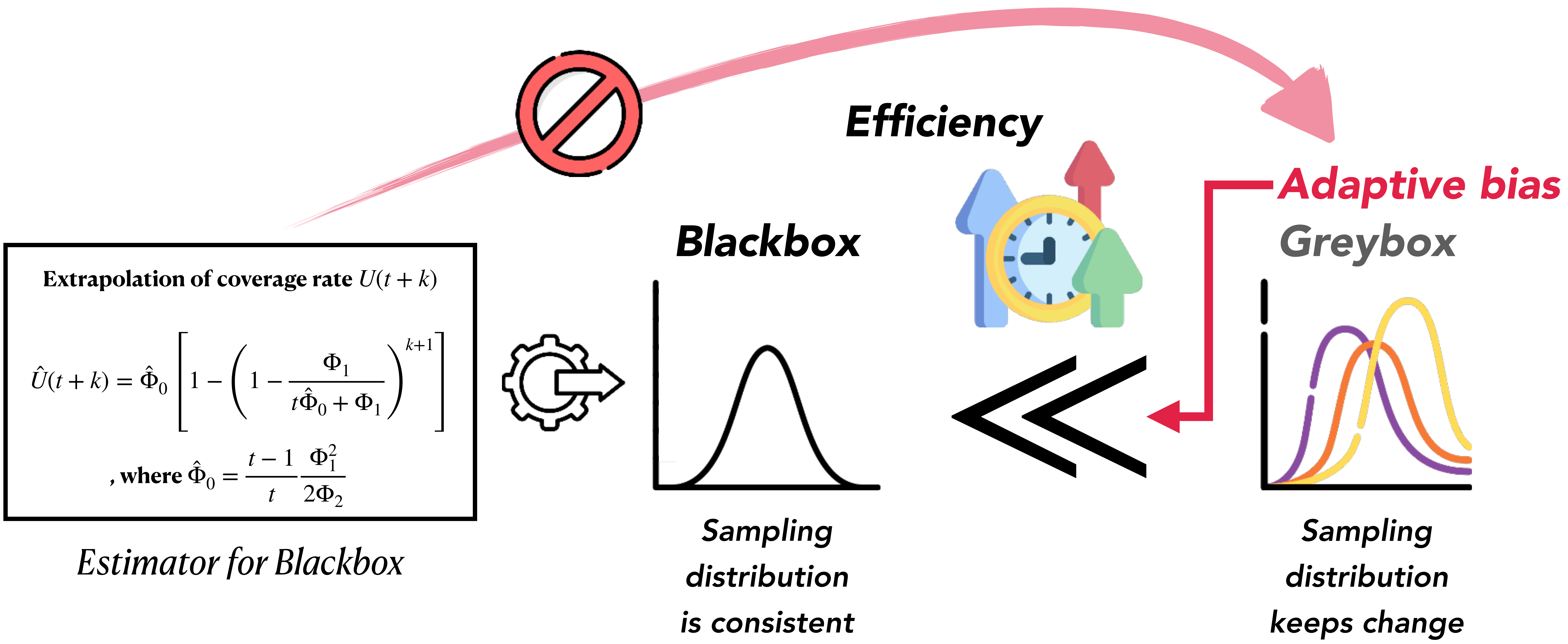


Sampling distribution keeps change


Efficiency
Adaptive bias
Blackbox
Greybox
Sampling distribution
is consistent
Sampling distribution
keeps change


Extrapolation of coverage rate $U(t+k)$
$\hat{U}(t+k) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{t\hat{\Phi}_0 + \Phi_1}\right)^{k+1}\right]$
, where $\hat{\Phi}_0 = \frac{t-1}{t}\frac{\Phi_1^2}{2\Phi_2}$
Estimator for Blackbox
Efficiency
Blackbox
Adaptive bias
Greybox
Sampling distribution is consistent
Sampling distribution keeps change


Extrapolation of coverage rate $U(t+k)$
$\hat{U}(t+k) = \hat{\Phi}_0 \left[1 - \left(1 - \frac{\Phi_1}{t\hat{\Phi}_0 + \Phi_1} \right)^{k+1} \right]$
, where $\hat{\Phi}_0 = \frac{t-1}{t} \frac{\Phi_1^2}{2\Phi_2}$
Estimator for Blackbox
Efficiency
Blackbox
Adaptive bias
Greybox
Sampling distribution is consistent
Sampling distribution keeps change



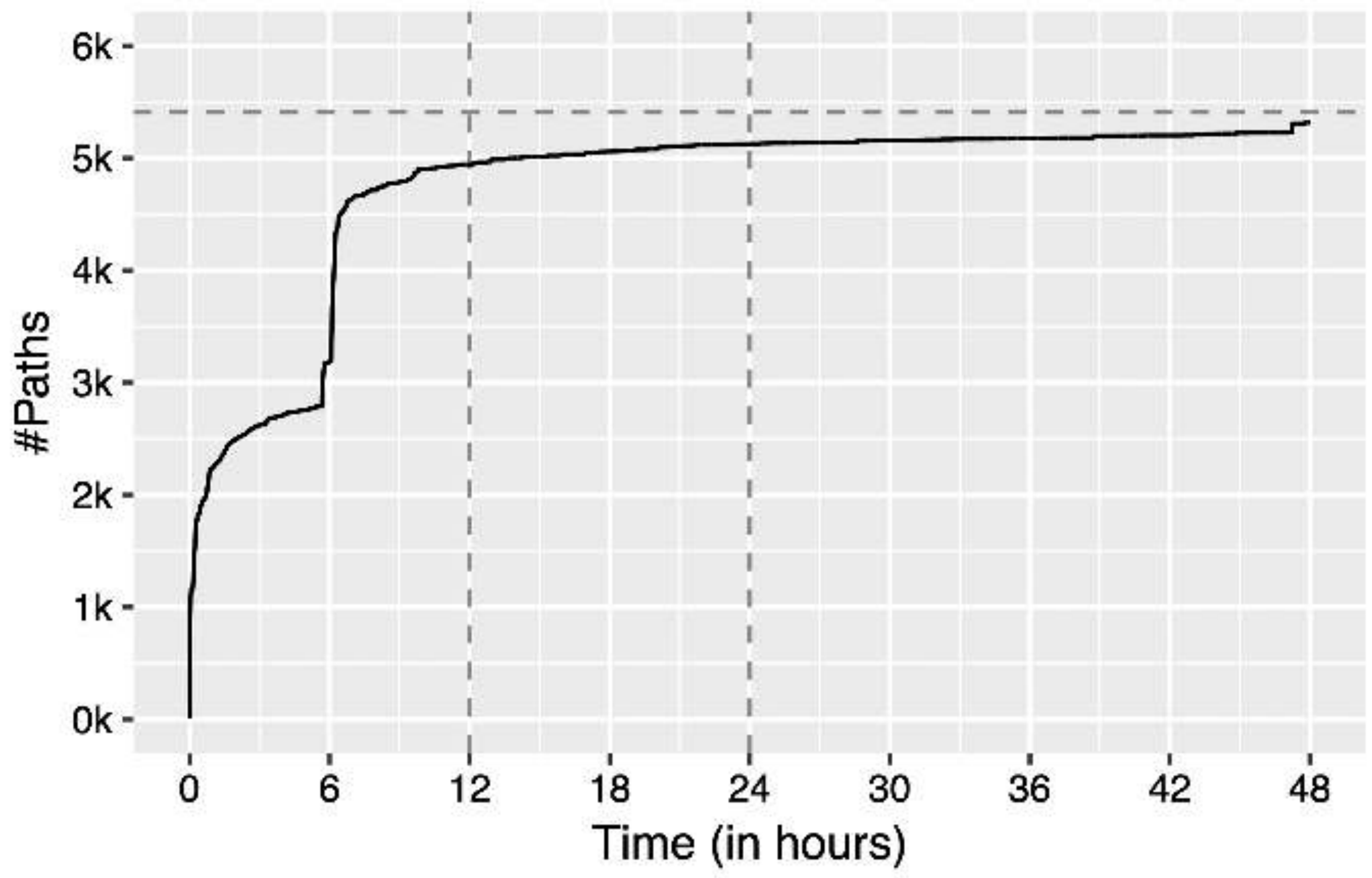
The estimators for the blackbox fuzzing will underestimate the performance of the greybox fuzzing.

Extrapolating the Greybox Fuzzing Campaign

- Aim: Predict the future coverage rate of the greybox fuzzing campaign

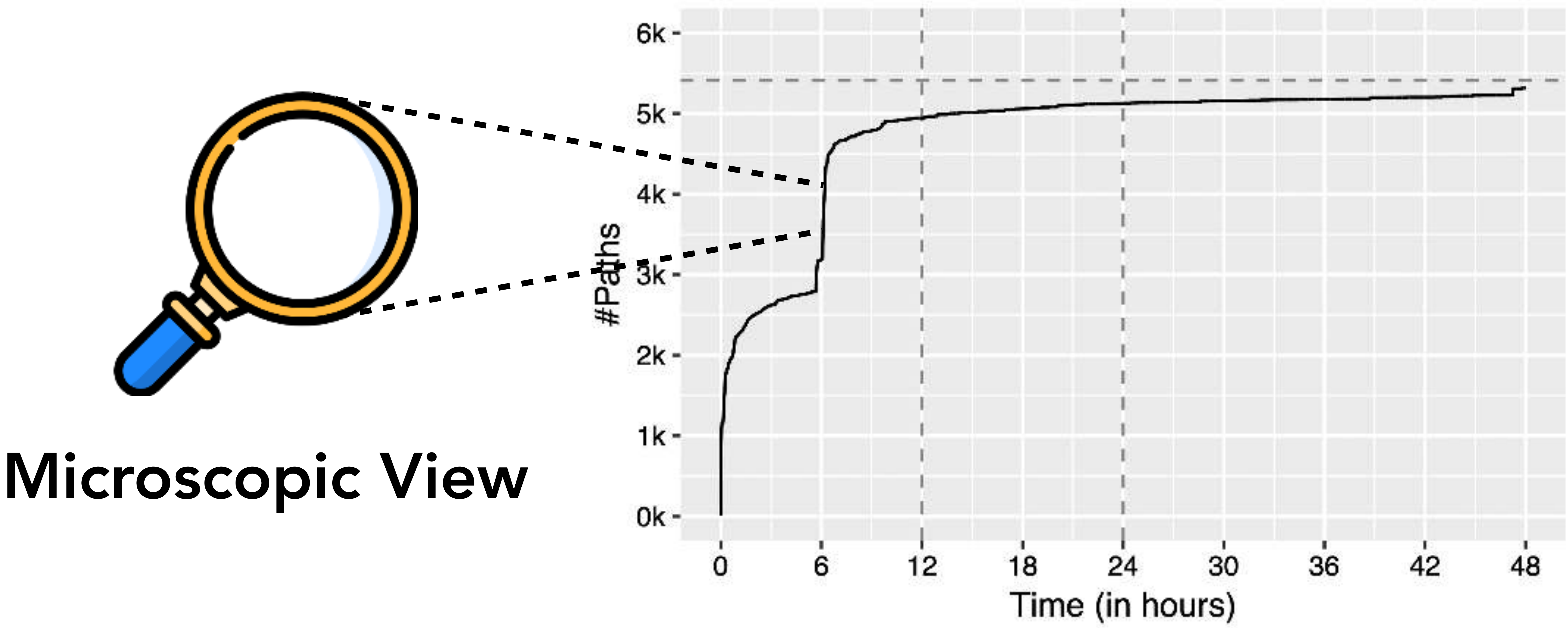
Extrapolating the Greybox Fuzzing Campaign

- Aim: Predict the future coverage rate of the greybox fuzzing campaign
- In other words, how can we solve the ***adaptive bias*** problem?



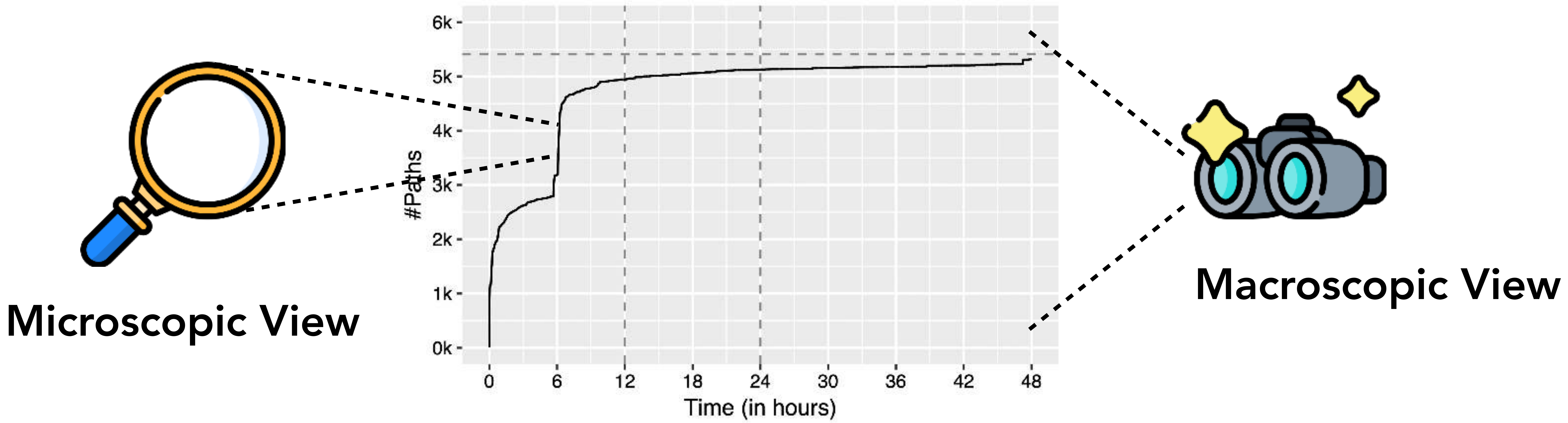
Extrapolating the Greybox Fuzzing Campaign

- Aim: Predict the future coverage rate of the greybox fuzzing campaign
- In other words, how can we solve the ***adaptive bias*** problem?



Extrapolating the Greybox Fuzzing Campaign

- Aim: Predict the future coverage rate of the greybox fuzzing campaign
- In other words, how can we solve the ***adaptive bias*** problem?

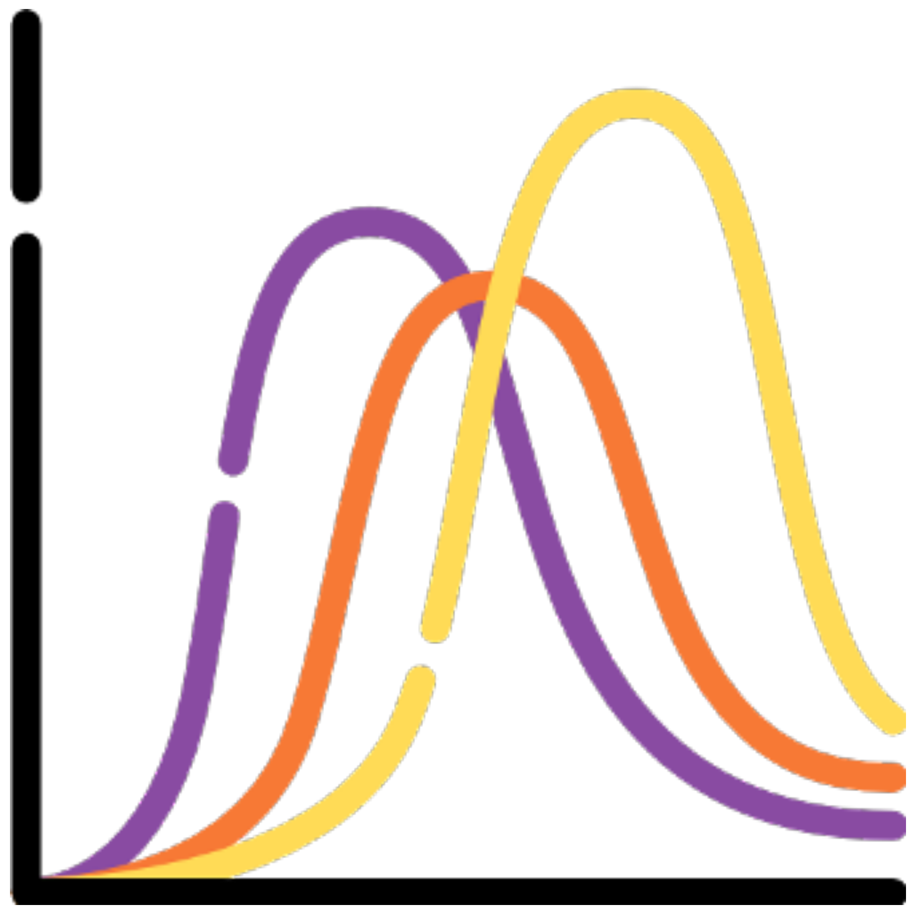


Extrapolating the Greybox Fuzzing Campaign

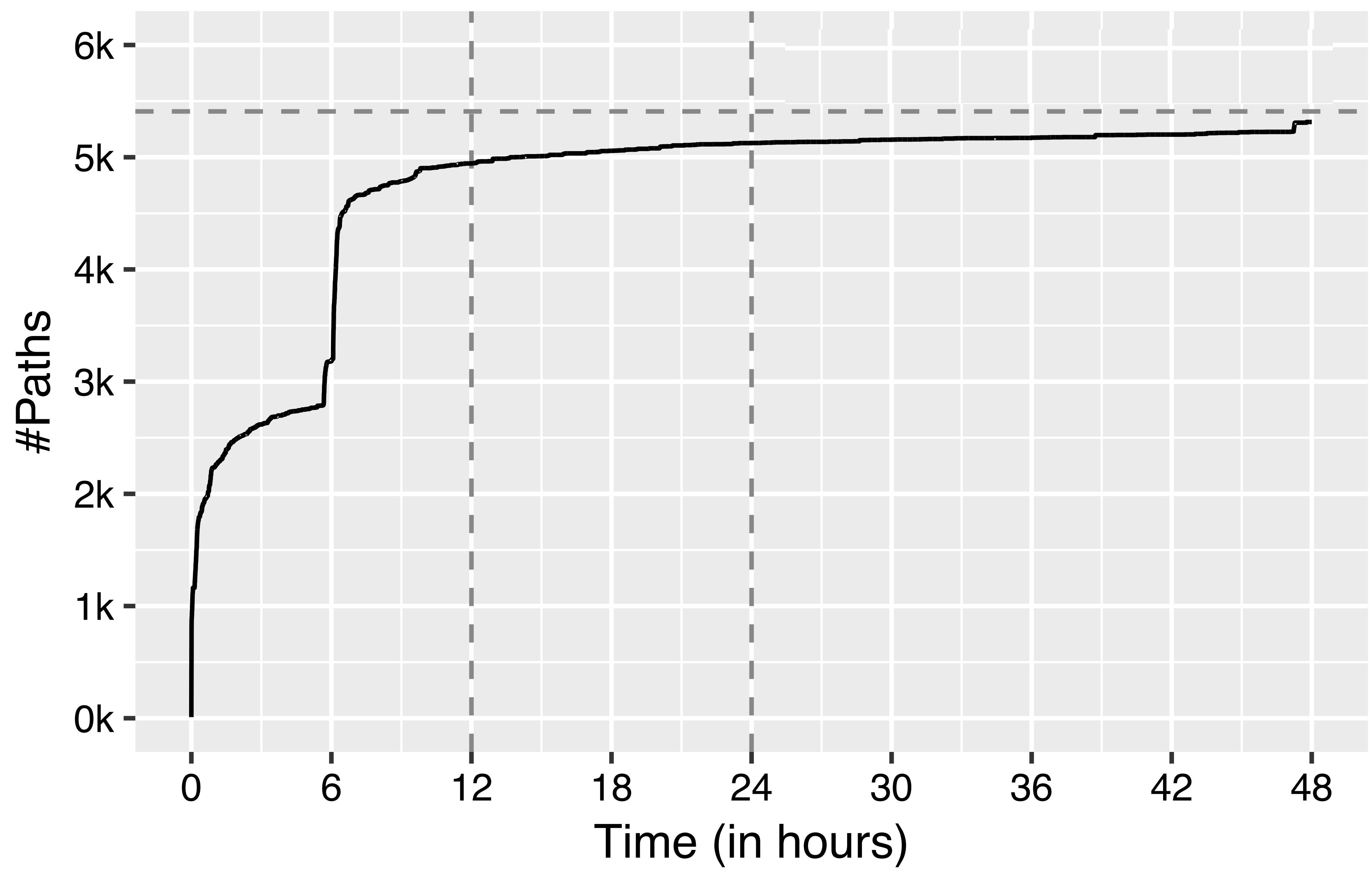
- First key insight — *Microscopic view*

Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*

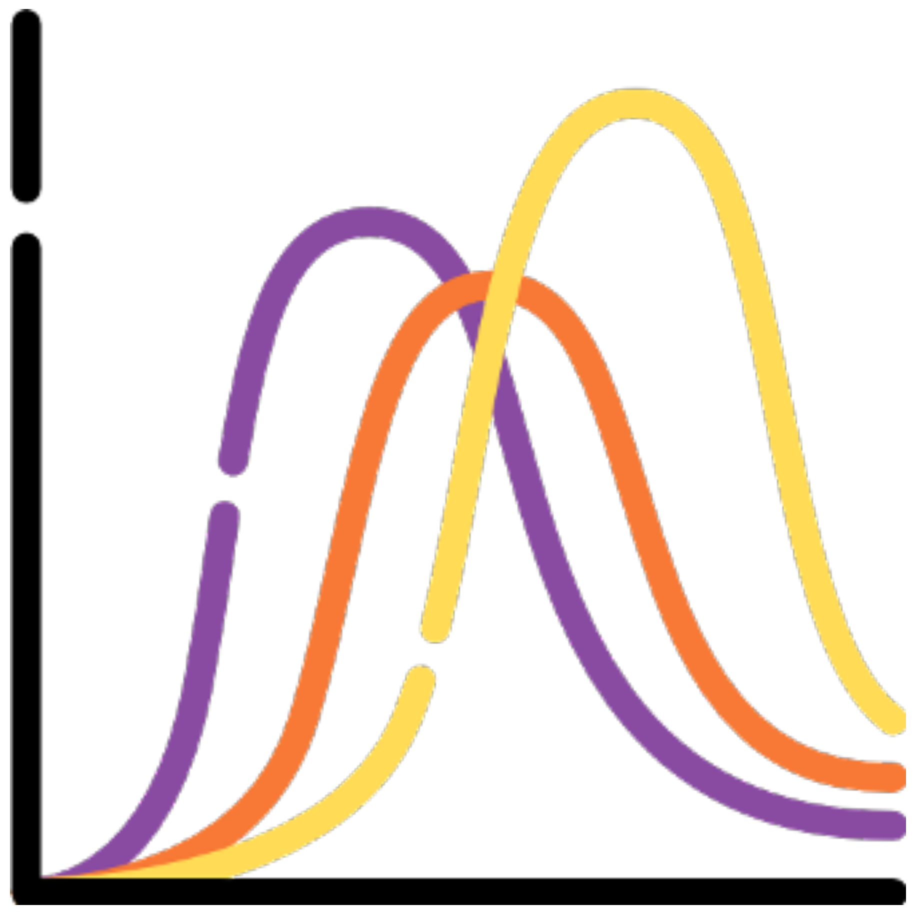


Adaptive bias

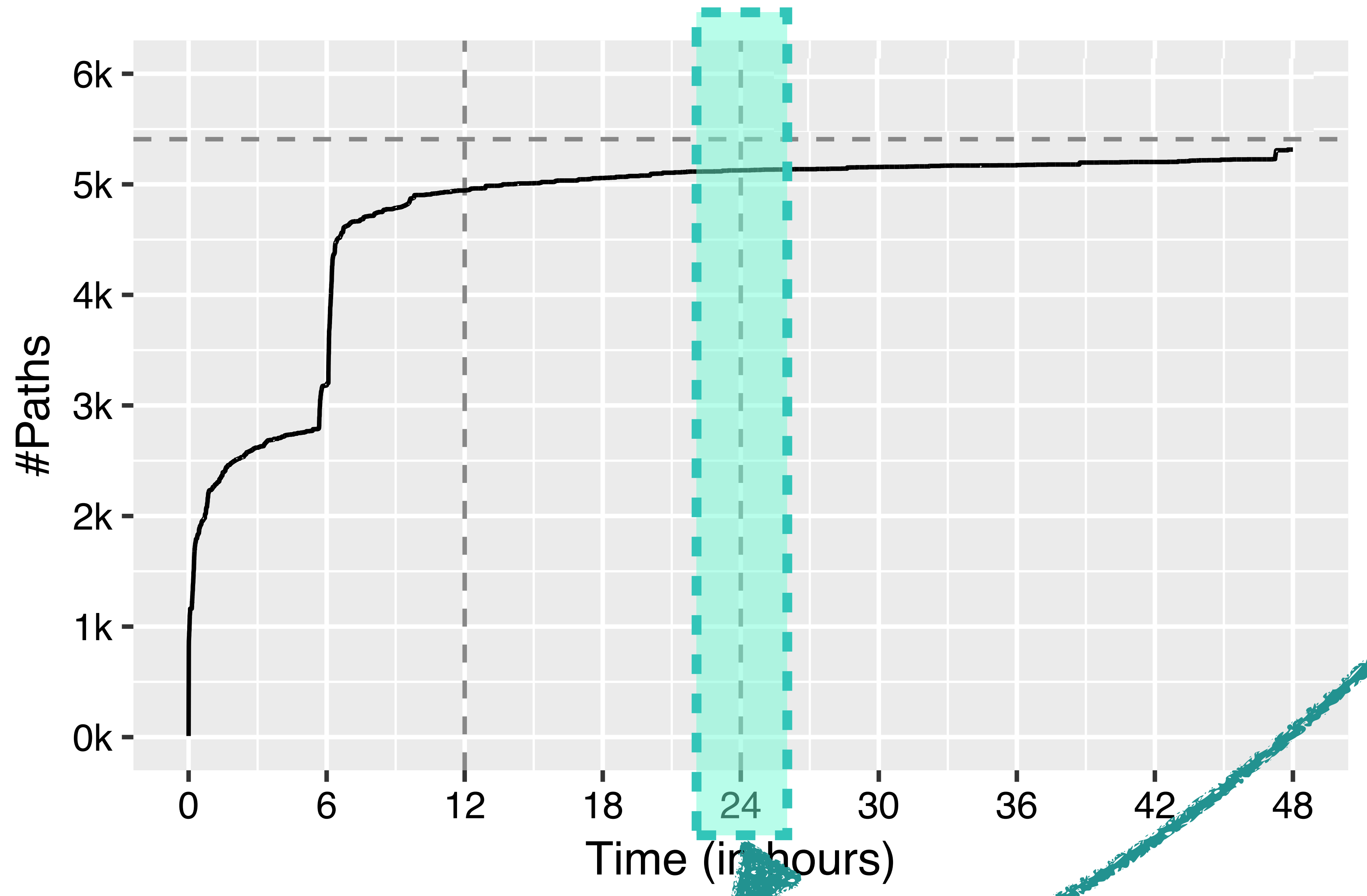


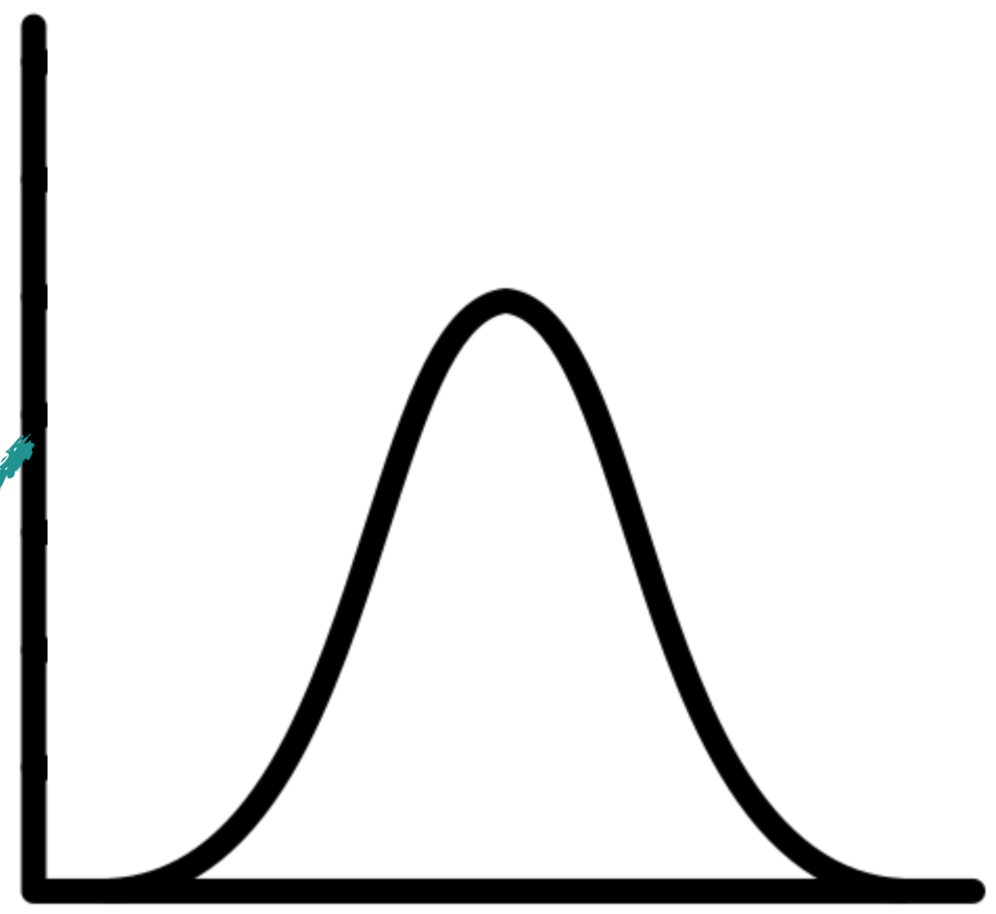
Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*



Adaptive bias

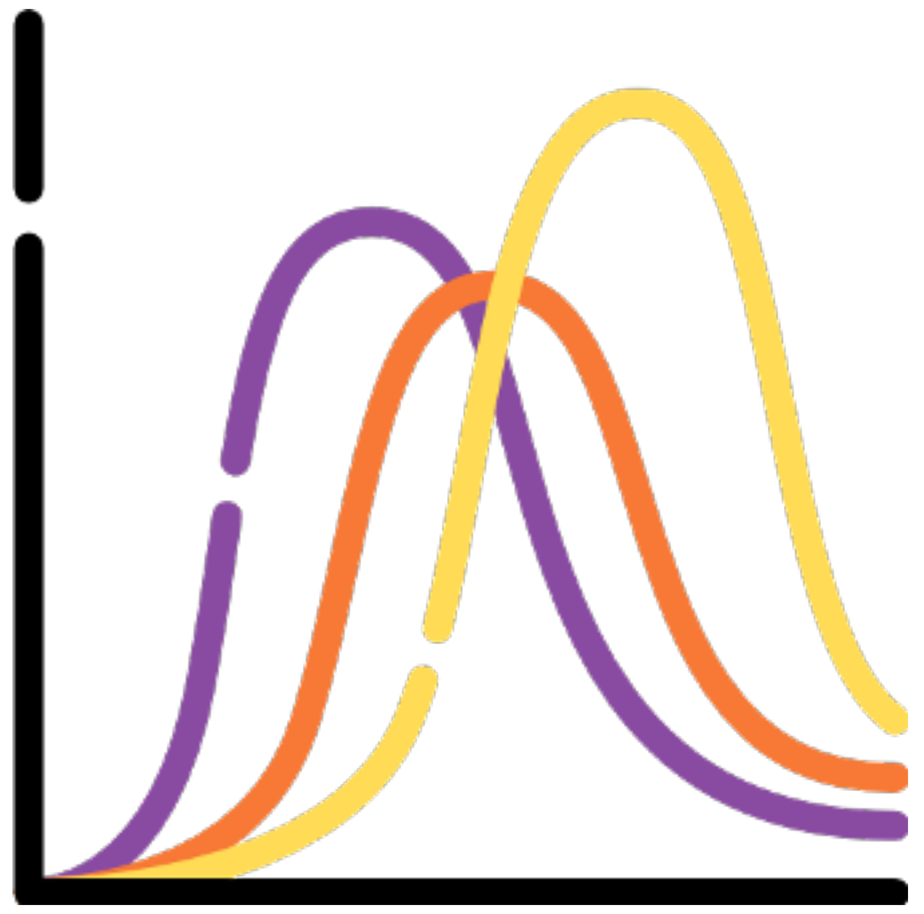




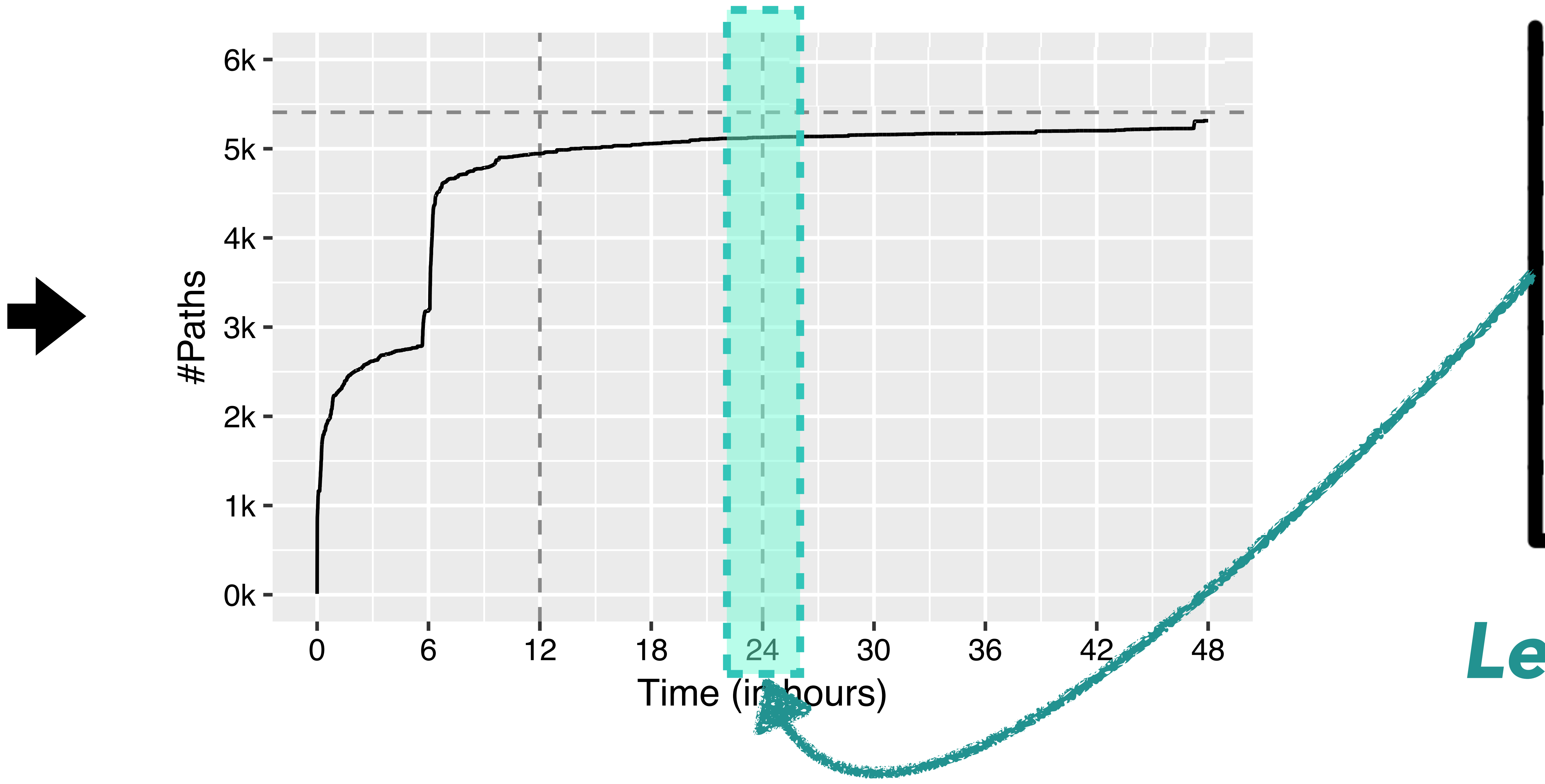
Less adaptive bias

Extrapolating the Greybox Fuzzing Campaign

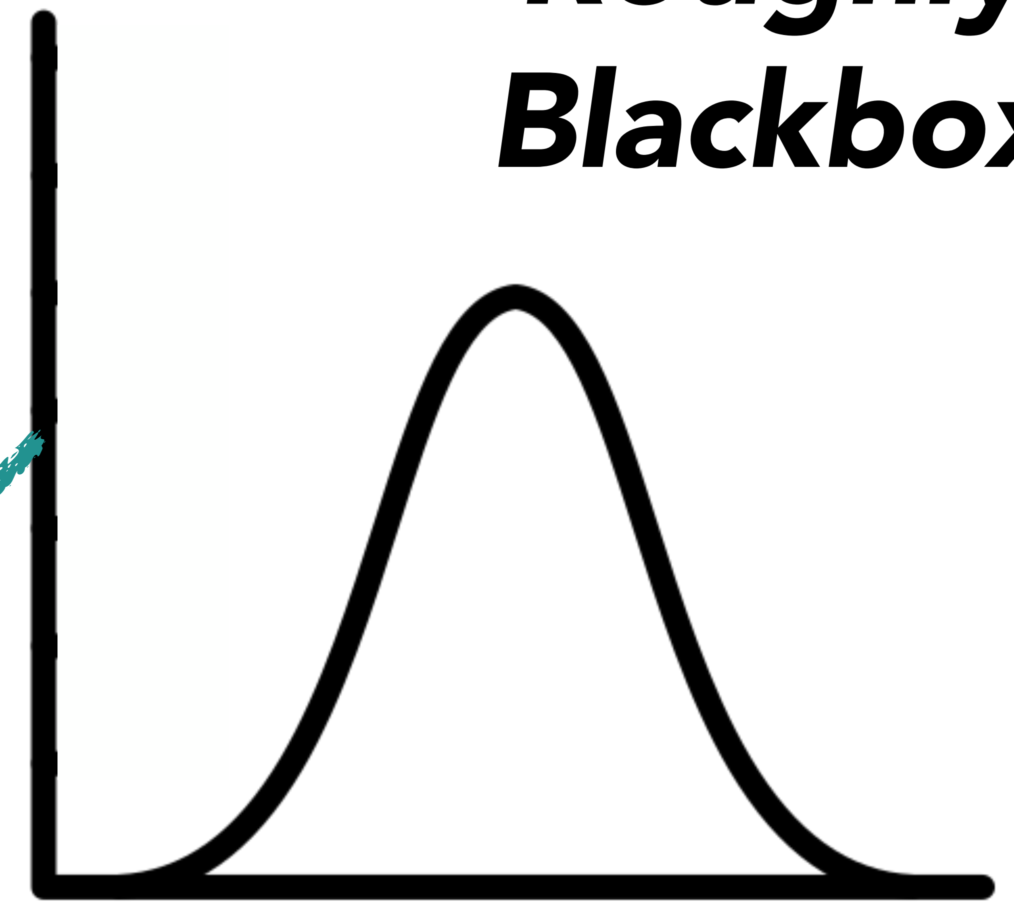
- First key insight — *Microscopic view*



Adaptive bias



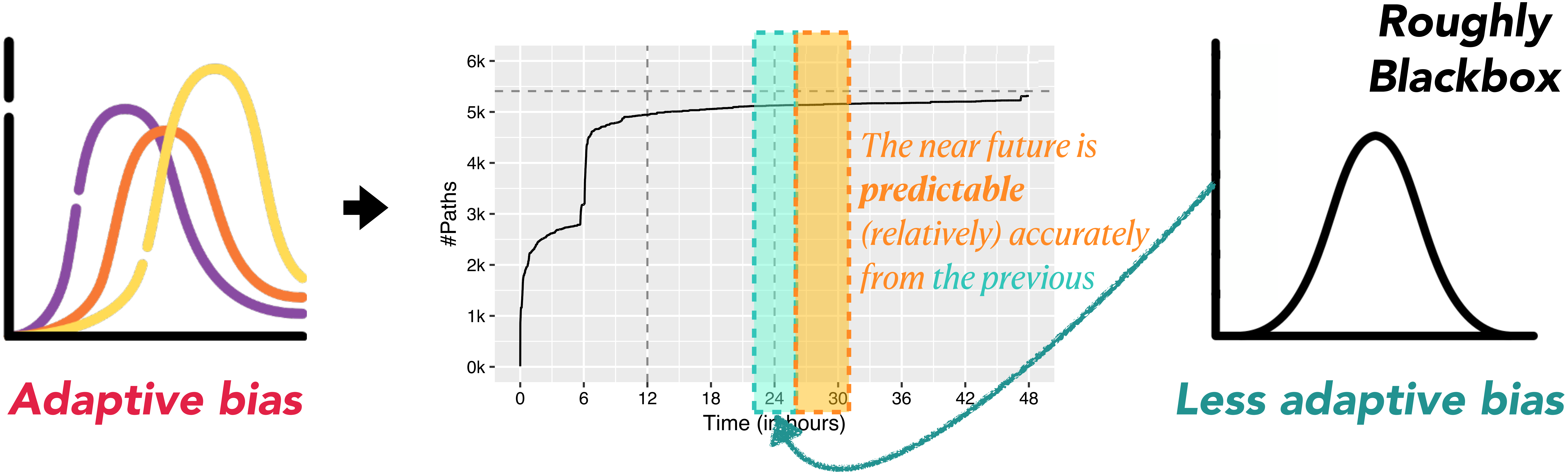
Roughly Blackbox



Less adaptive bias

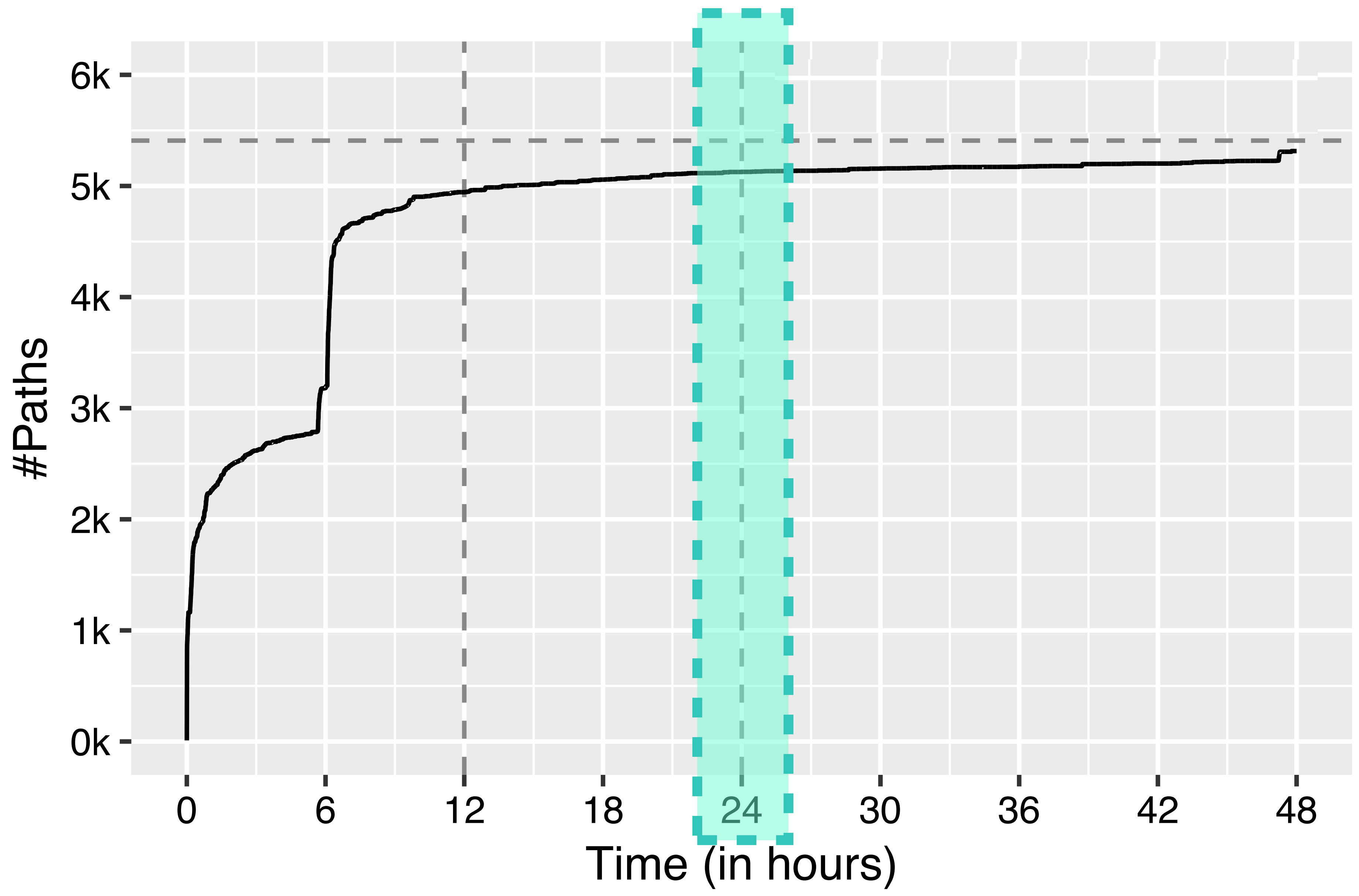
Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*



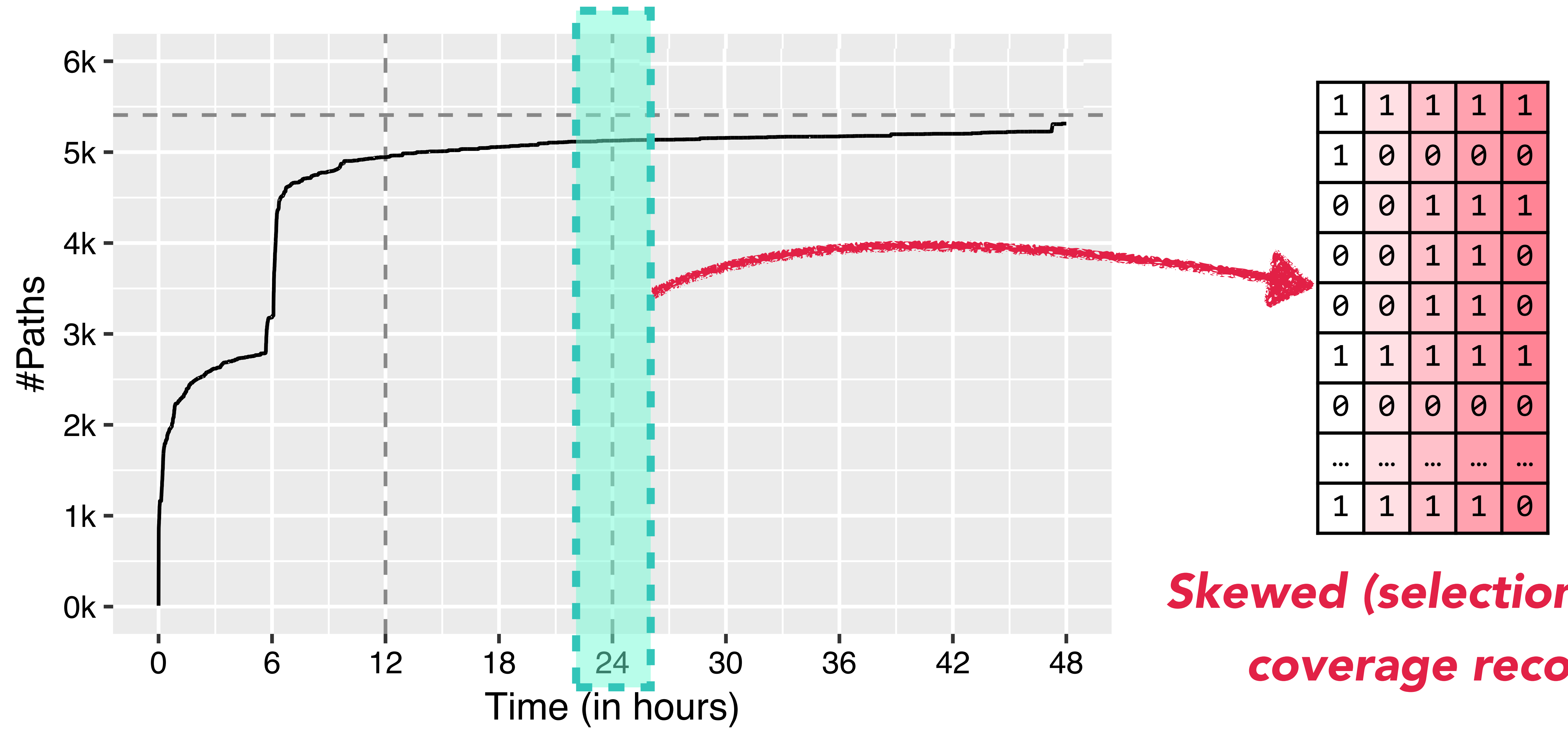
Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*
- Potential drawback: ***Small region has small data to use***



Extrapolating the Greybox Fuzzing Campaign

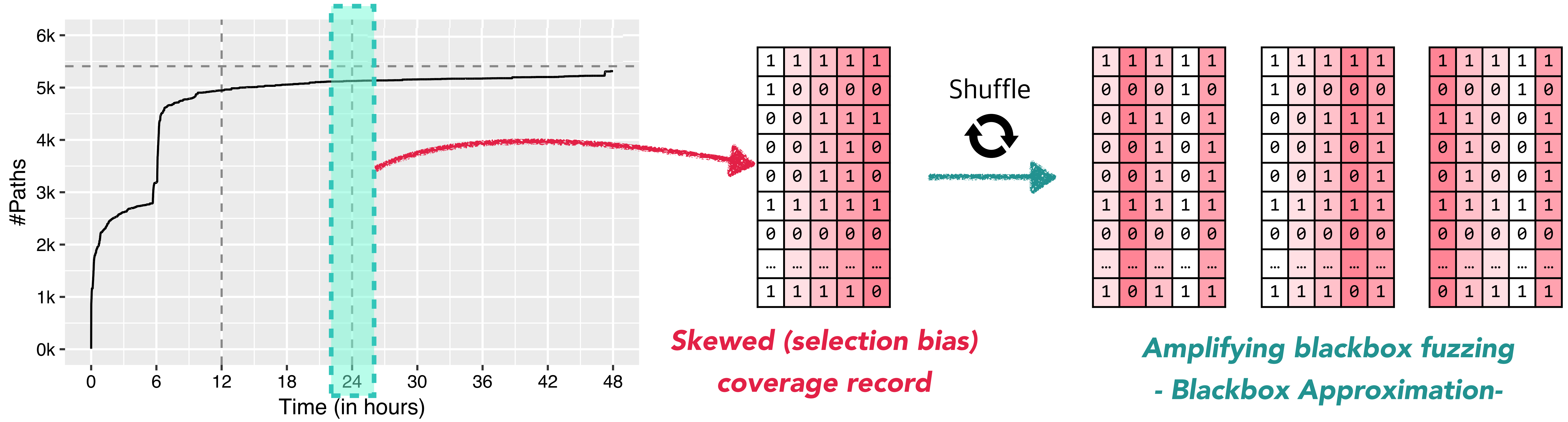
- First key insight — *Microscopic view*
- Potential drawback: ***Small region has small data to use***



Skewed (selection bias) coverage record

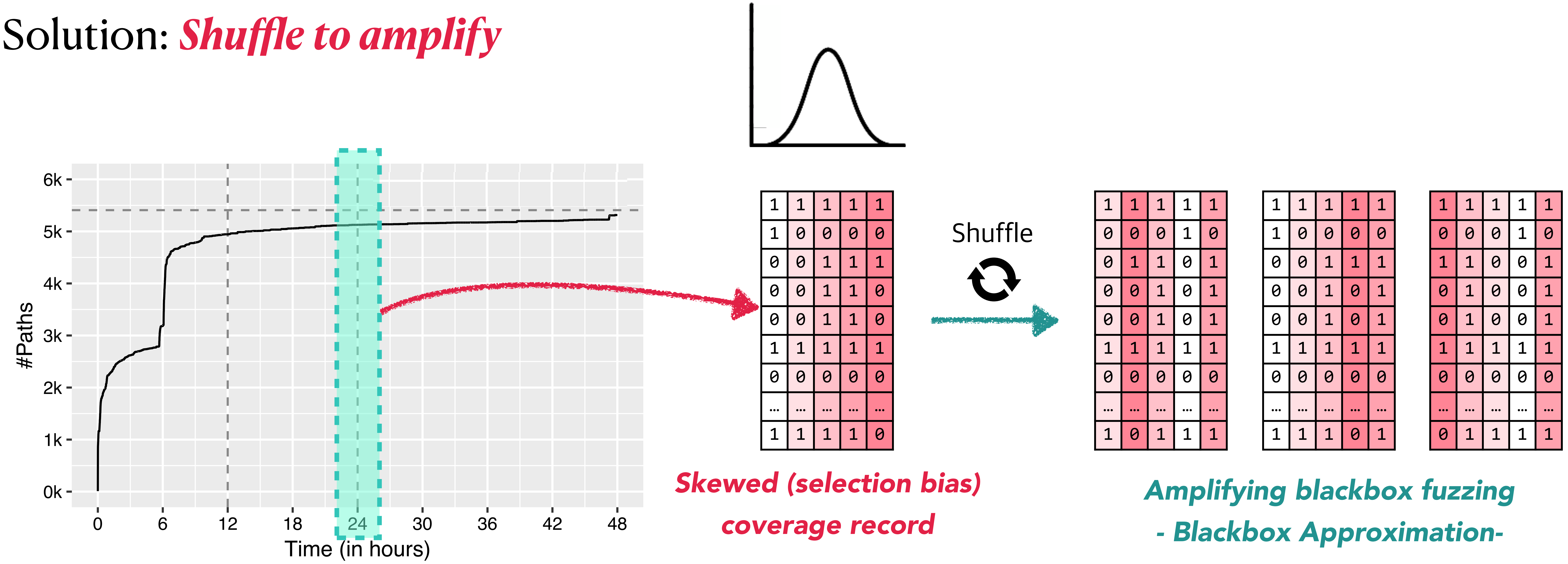
Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*
- Solution: ***Shuffle to amplify***



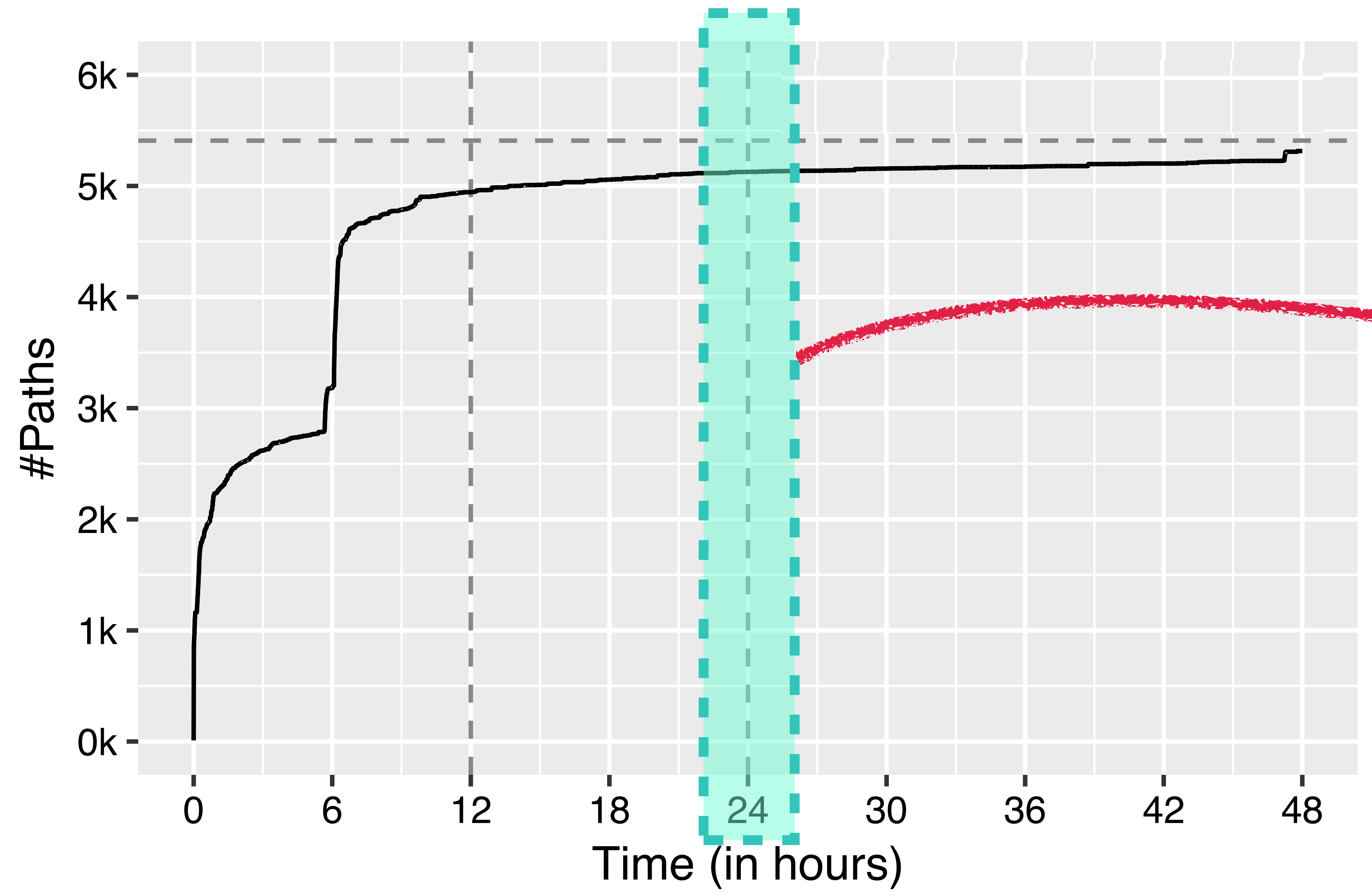
Extrapolating the Greybox Fuzzing Campaign

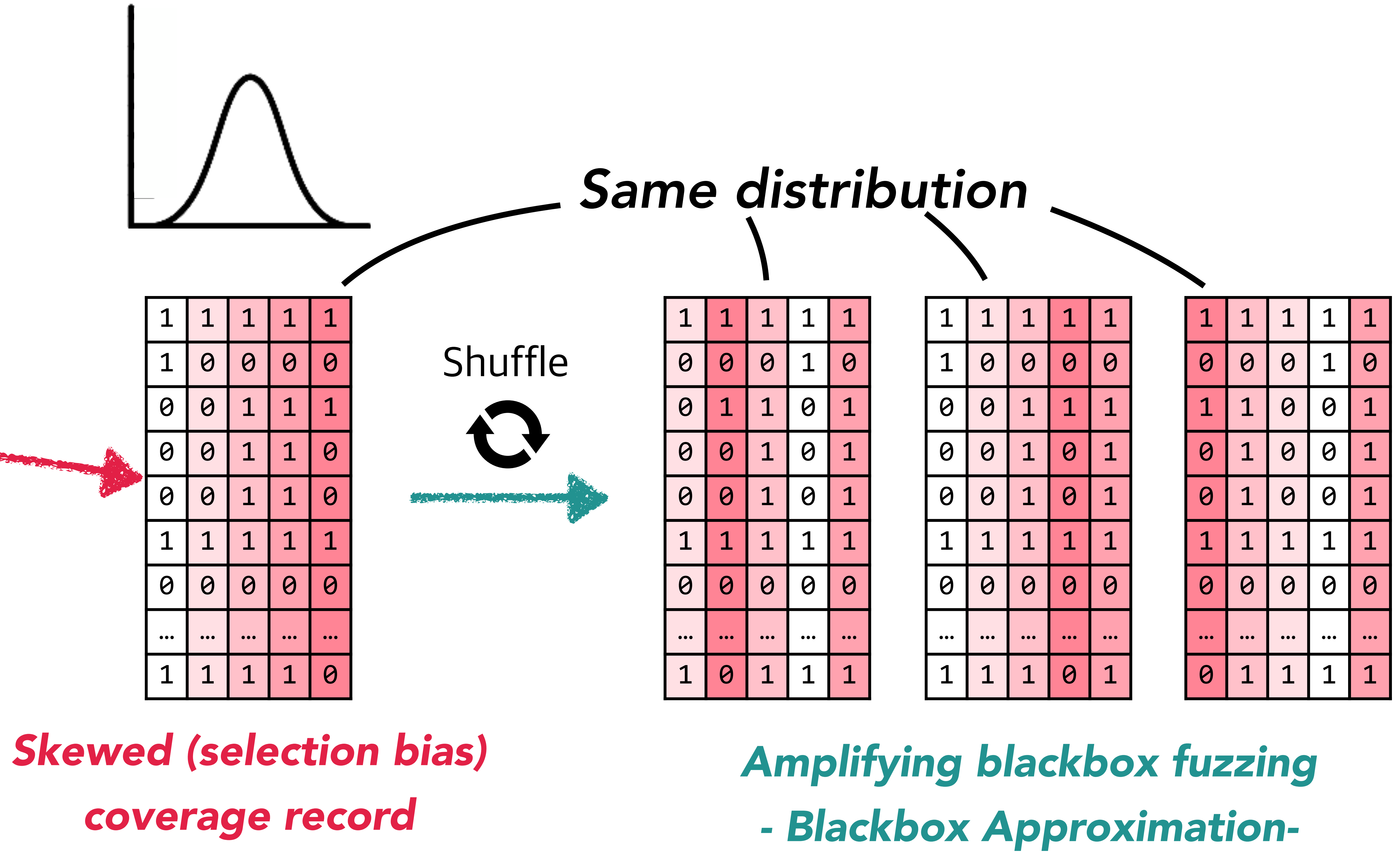
- First key insight — *Microscopic view*
- Solution: ***Shuffle to amplify***



Extrapolating the Greybox Fuzzing Campaign

- First key insight — *Microscopic view*
- Solution: ***Shuffle to amplify***





Extrapolating the Greybox Fuzzing Campaign

- Second key insight — *Macroscopic view*

*"Greybox fuzzing's adaptive bias could be **predictable**."*

Extrapolating the Greybox **Fuzzing Campaign**

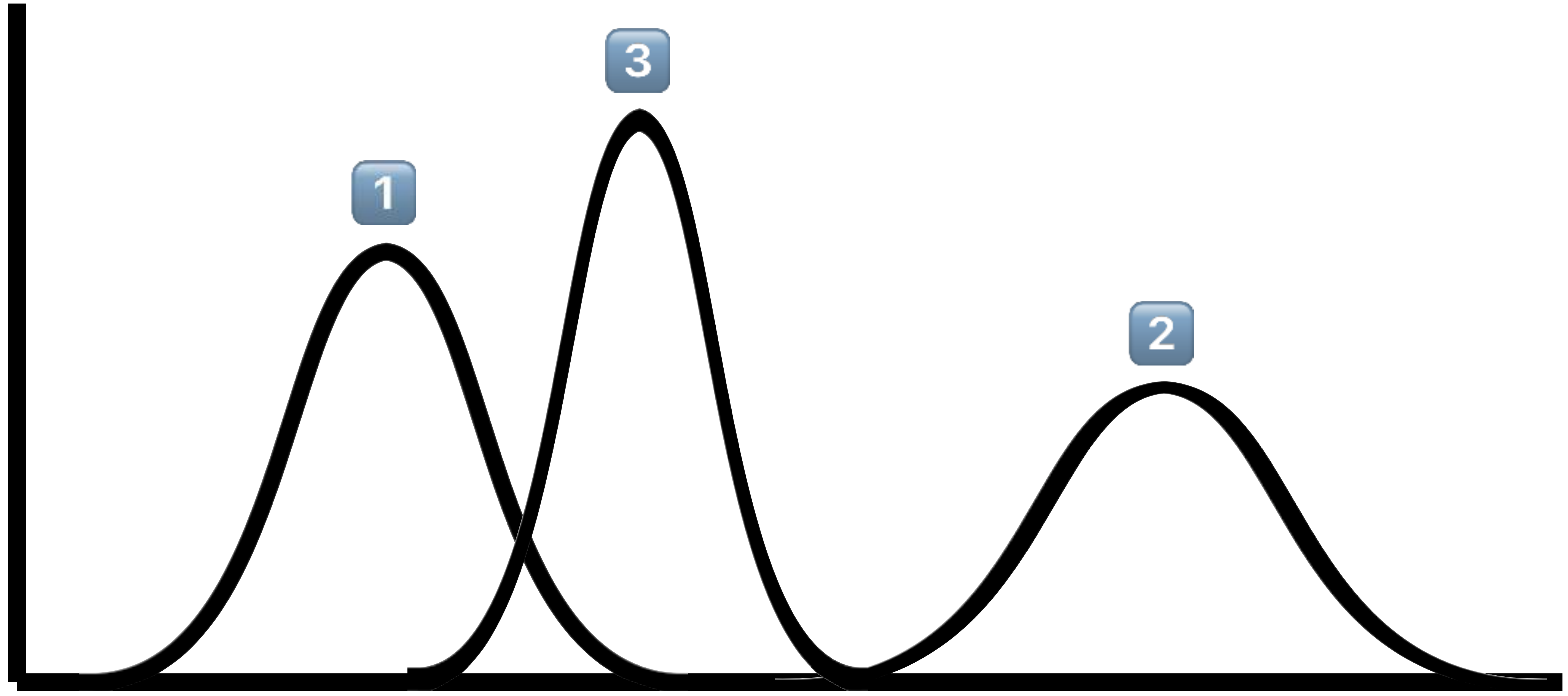
- Second key insight — *Macroscopic view*

*"Greybox fuzzing's adaptive bias could be **predictable**."*

Extrapolating the Greybox Fuzzing Campaign

- Second key insight — *Macroscopic view*

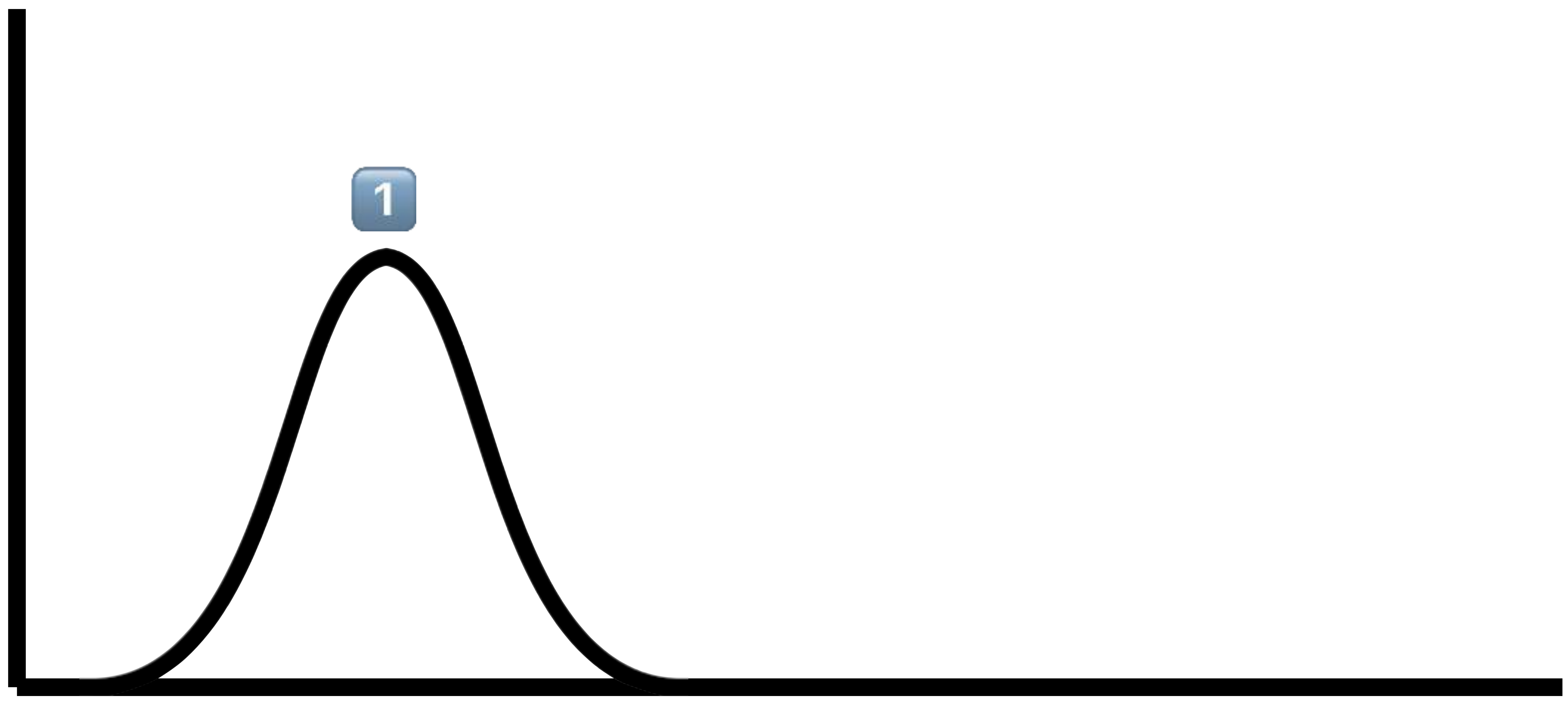
*"Greybox fuzzing's adaptive bias could be **predictable**."*



Extrapolating the Greybox Fuzzing Campaign

- Second key insight — *Macroscopic view*

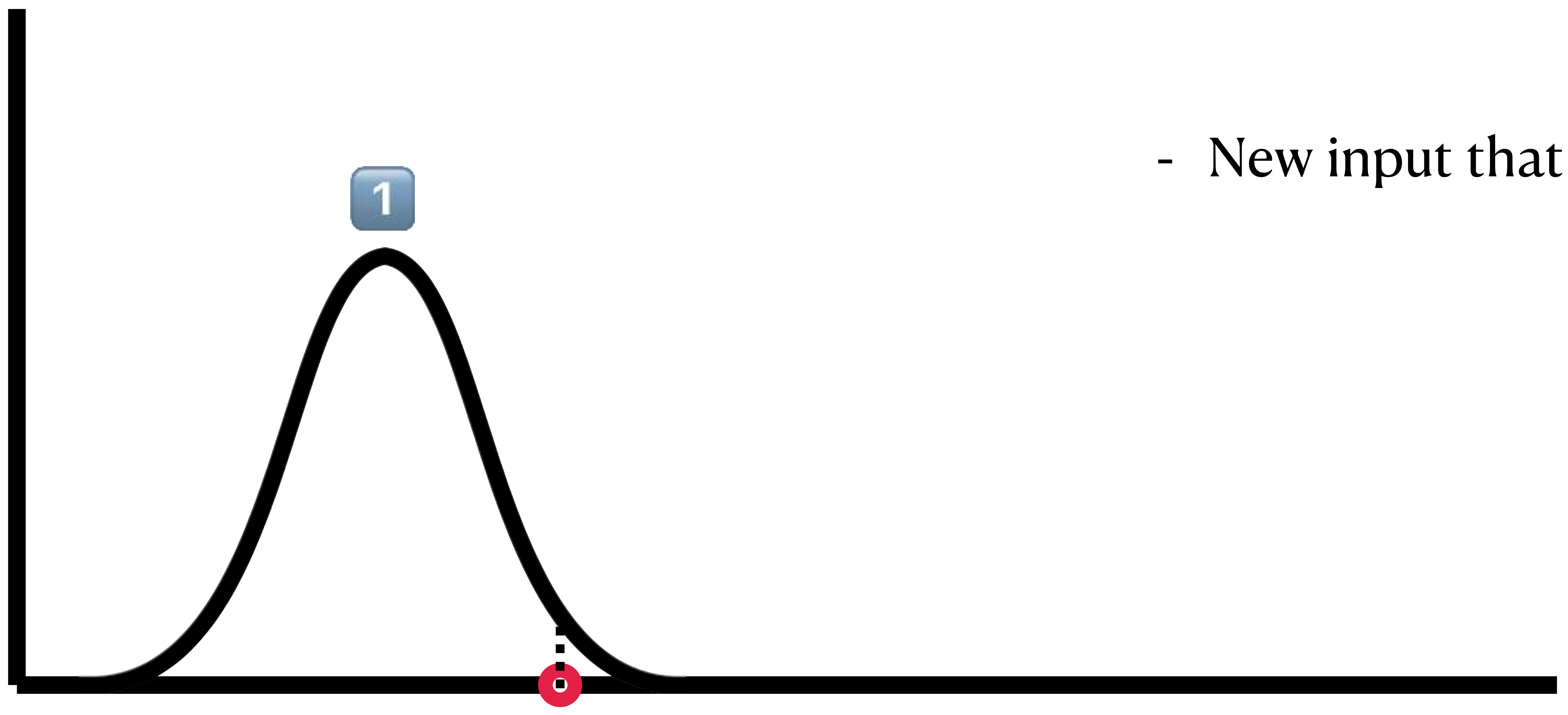
"Greybox fuzzing's adaptive bias could be ***predictable****." — There's a* ***pattern****!*



Extrapolating the Greybox Fuzzing Campaign

- Second key insight — *Macroscopic view*

*"Greybox fuzzing's adaptive bias could be **predictable**." — There's a **pattern**!*

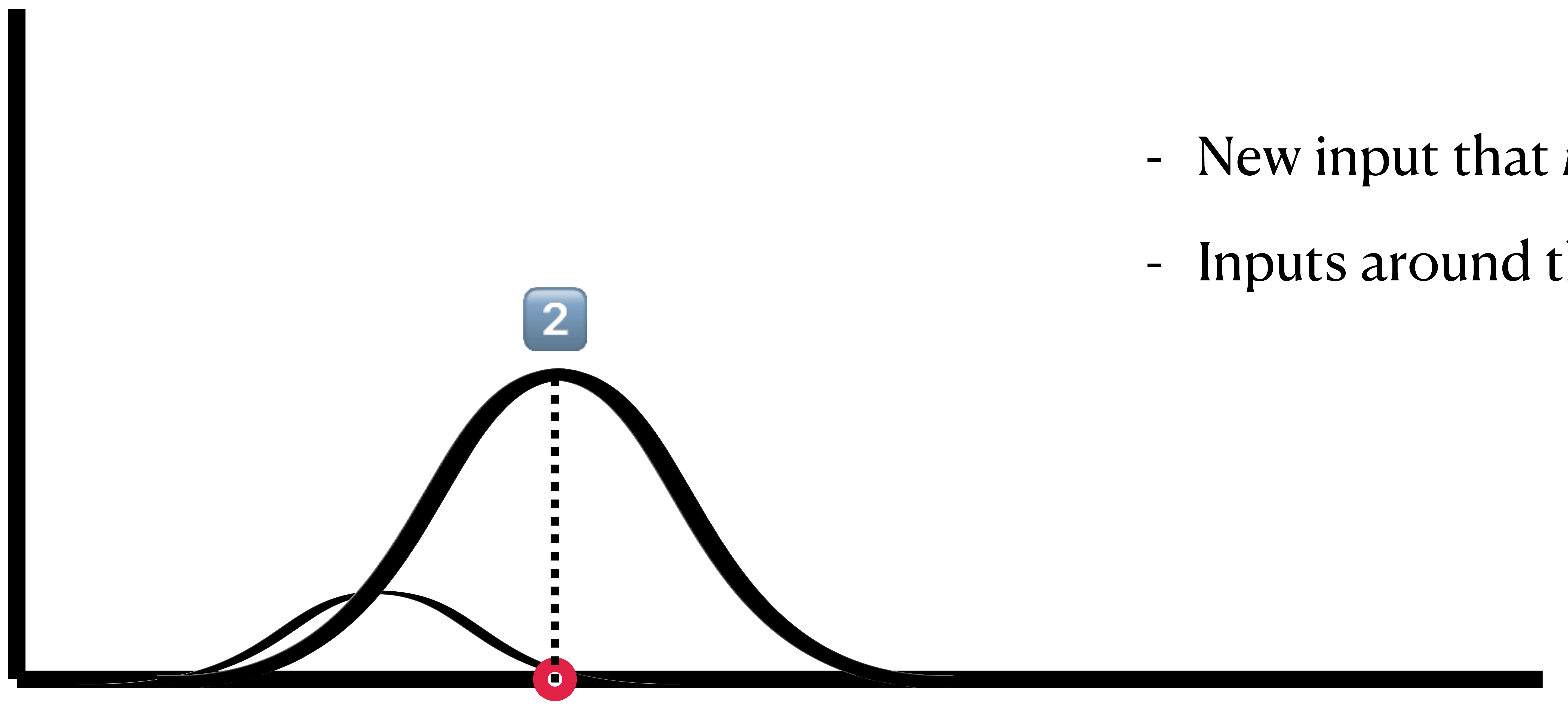


- New input that *increases coverage* is found.

Extrapolating the Greybox **Fuzzing Campaign**

- Second key insight — *Macroscopic view*

"Greybox fuzzing's adaptive bias could be ***predictable****." — There's a* ***pattern****!*

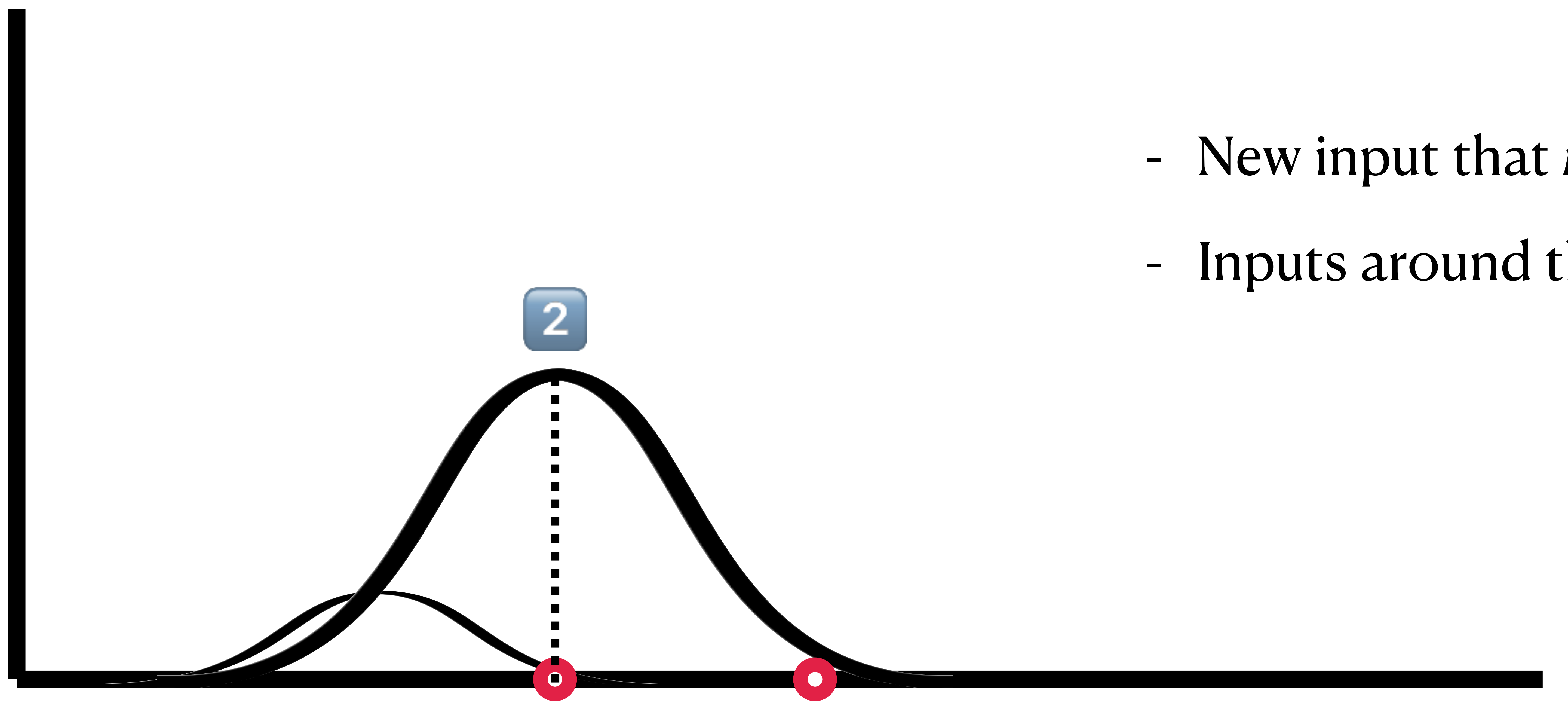


- New input that *increases coverage* is found.
- Inputs around the new input are sampled.

Extrapolating the Greybox **Fuzzing Campaign**

- Second key insight — *Macroscopic view*

"Greybox fuzzing's adaptive bias could be ***predictable****." — There's a* ***pattern****!*

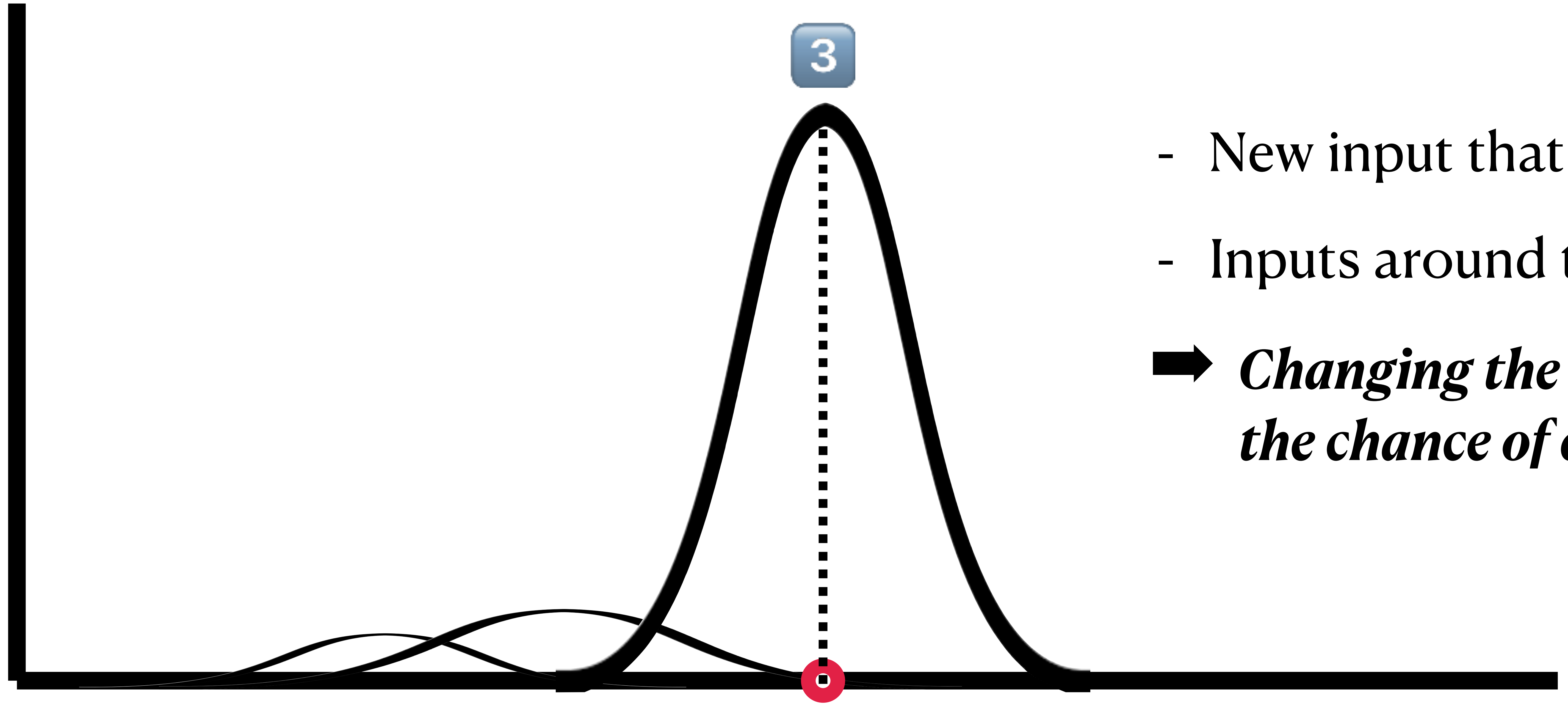


- New input that *increases coverage* is found.
- Inputs around the new input are sampled.

Extrapolating the Greybox Fuzzing Campaign

- Second key insight — *Macroscopic view*

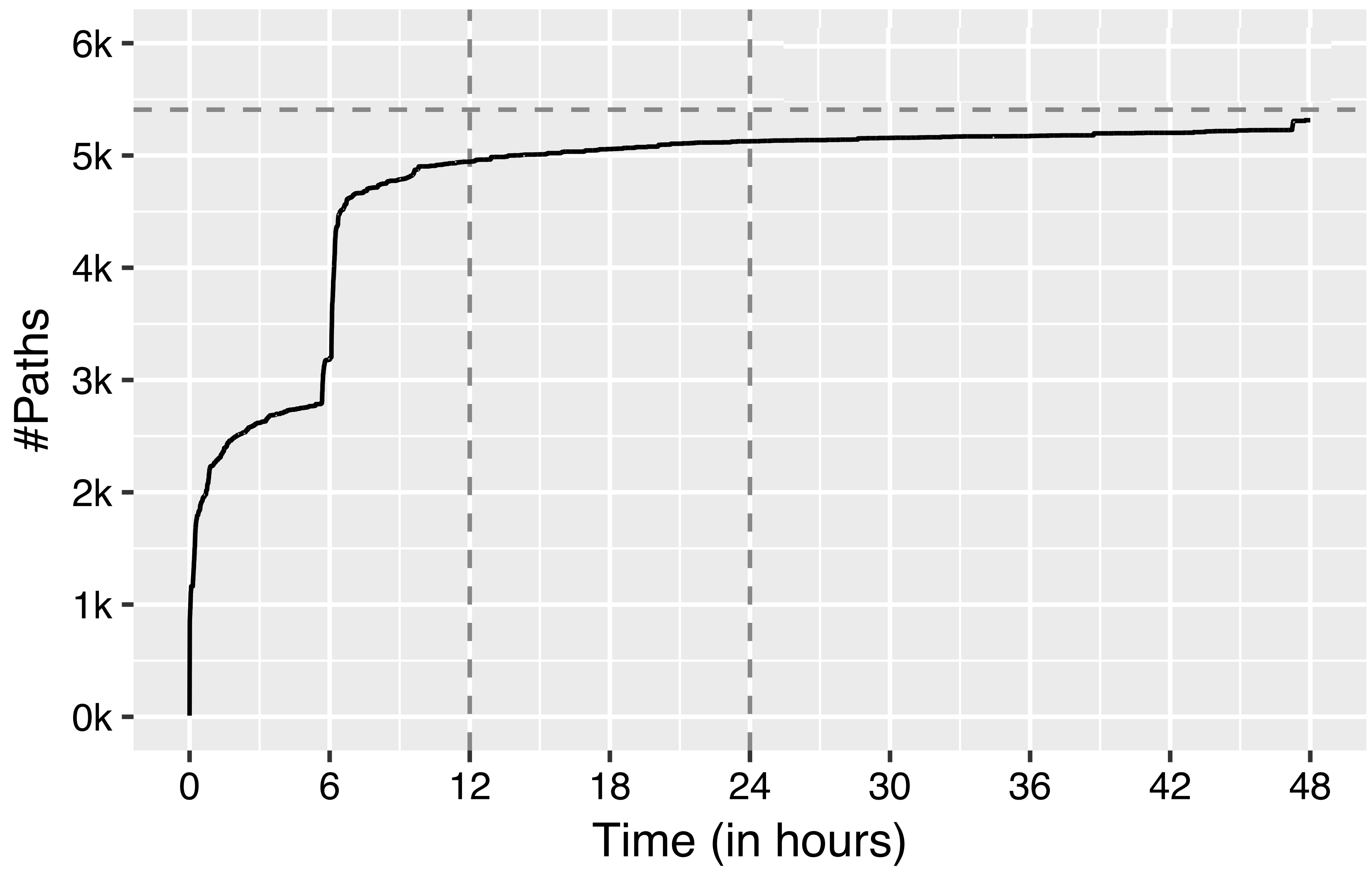
*"Greybox fuzzing's adaptive bias could be **predictable**." — There's a **pattern**!*



- New input that *increases coverage* is found.
- Inputs around the new input are sampled.

➡ ***Changing the focus (distribution) increases the chance of covering a new part of the program.***

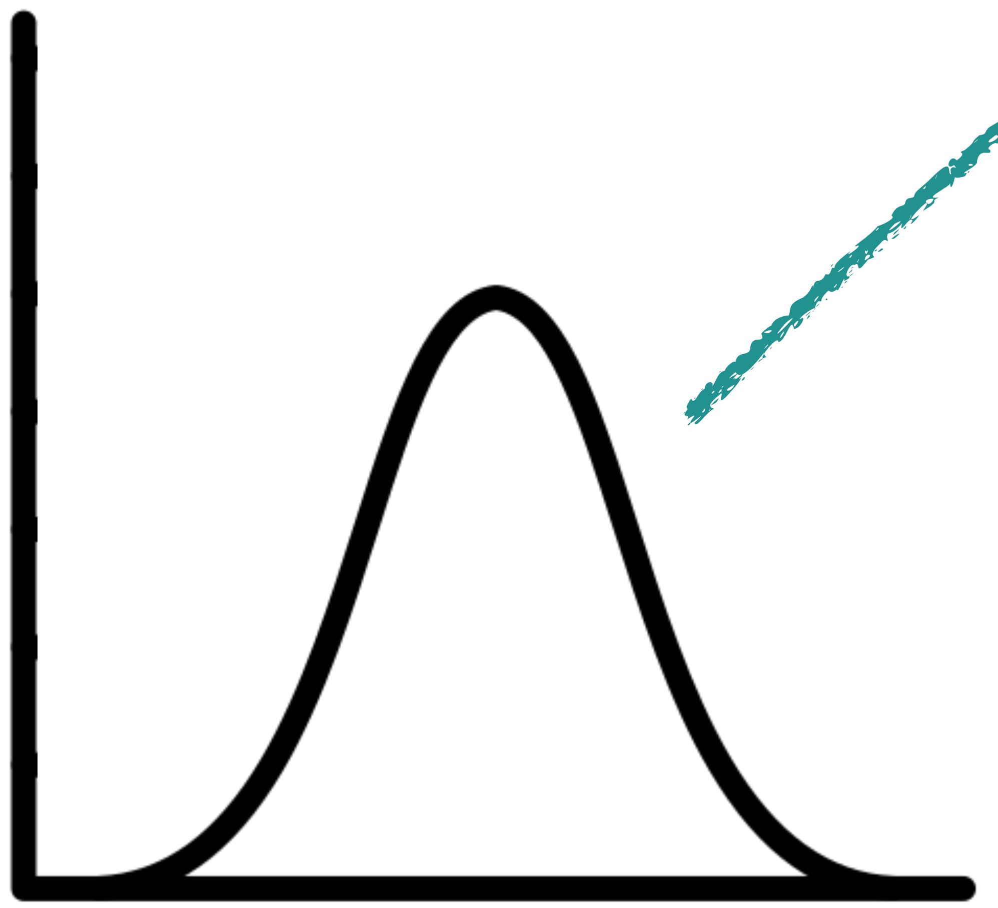
Methodology



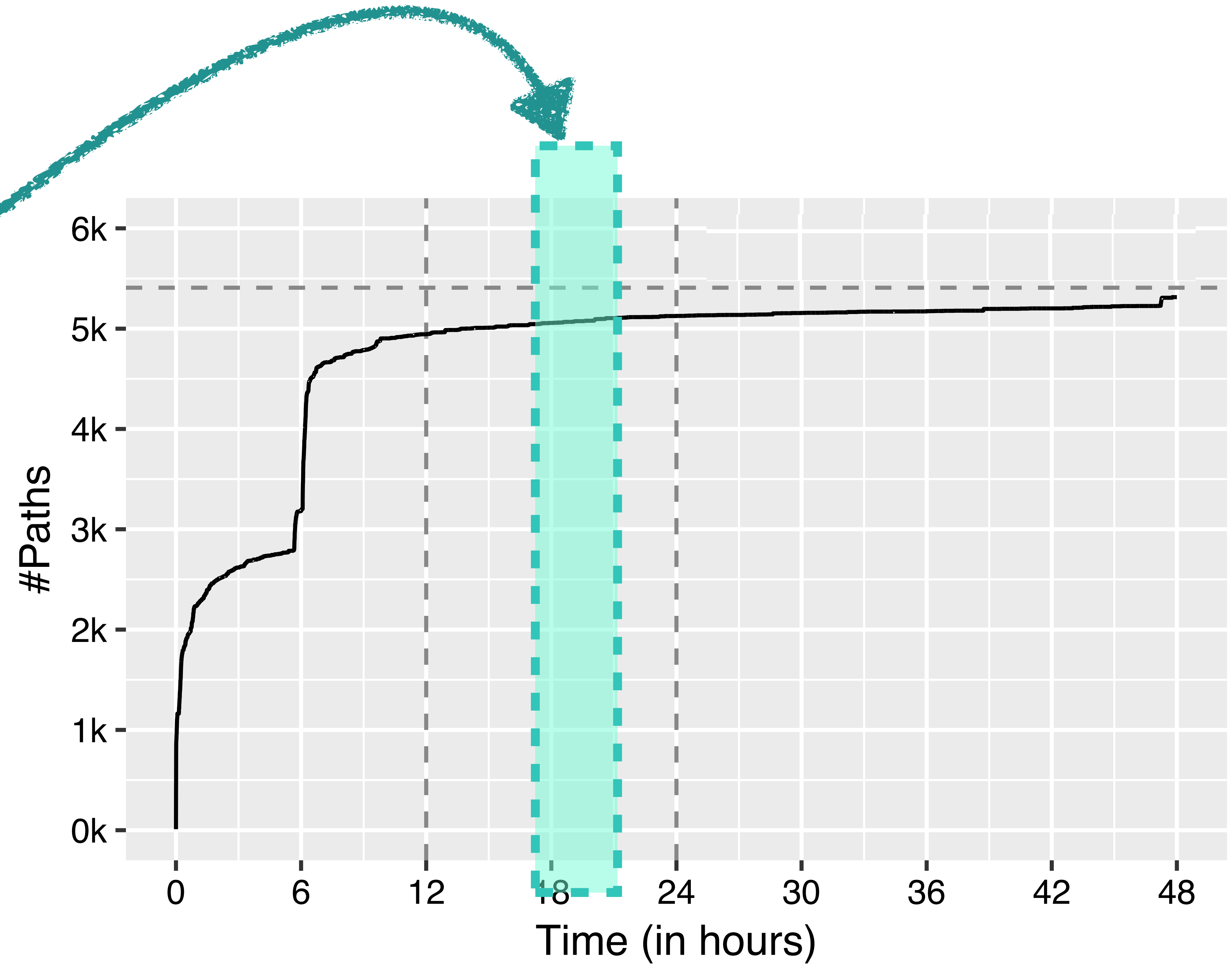
Coverage Increase Plot of the Greybox Fuzzing

Methodology

1️⃣



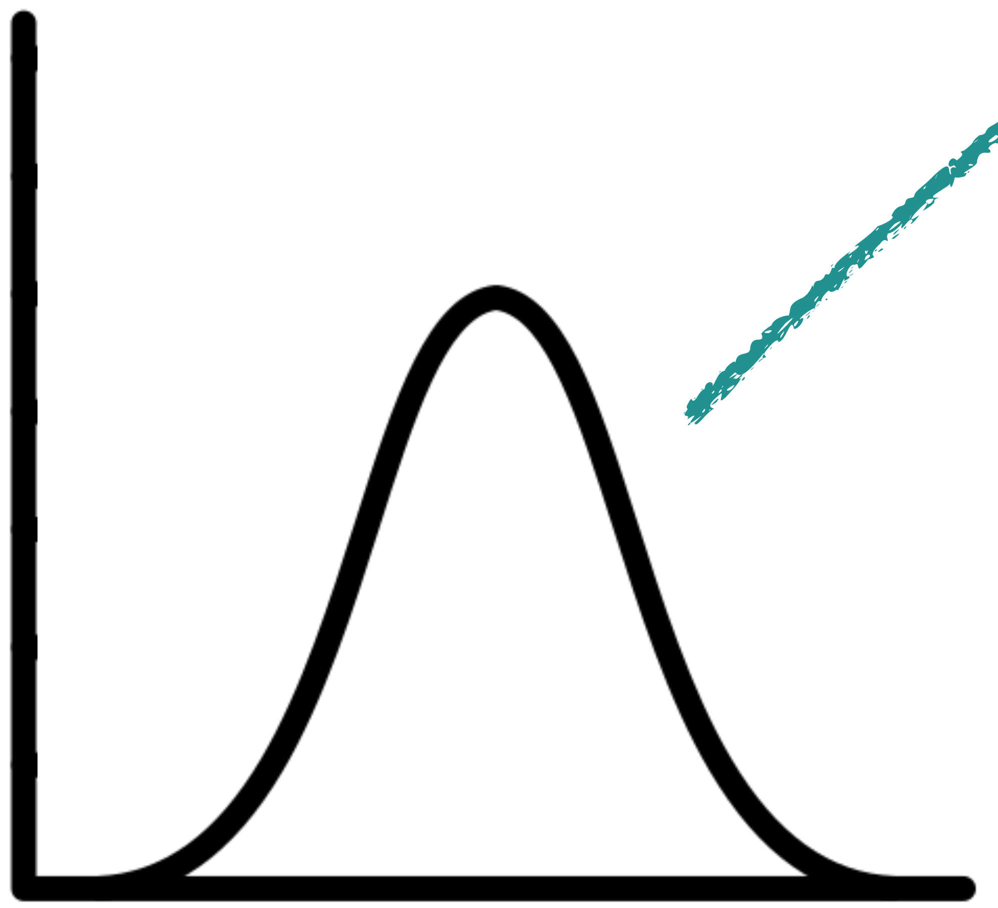
Roughly Blackbox



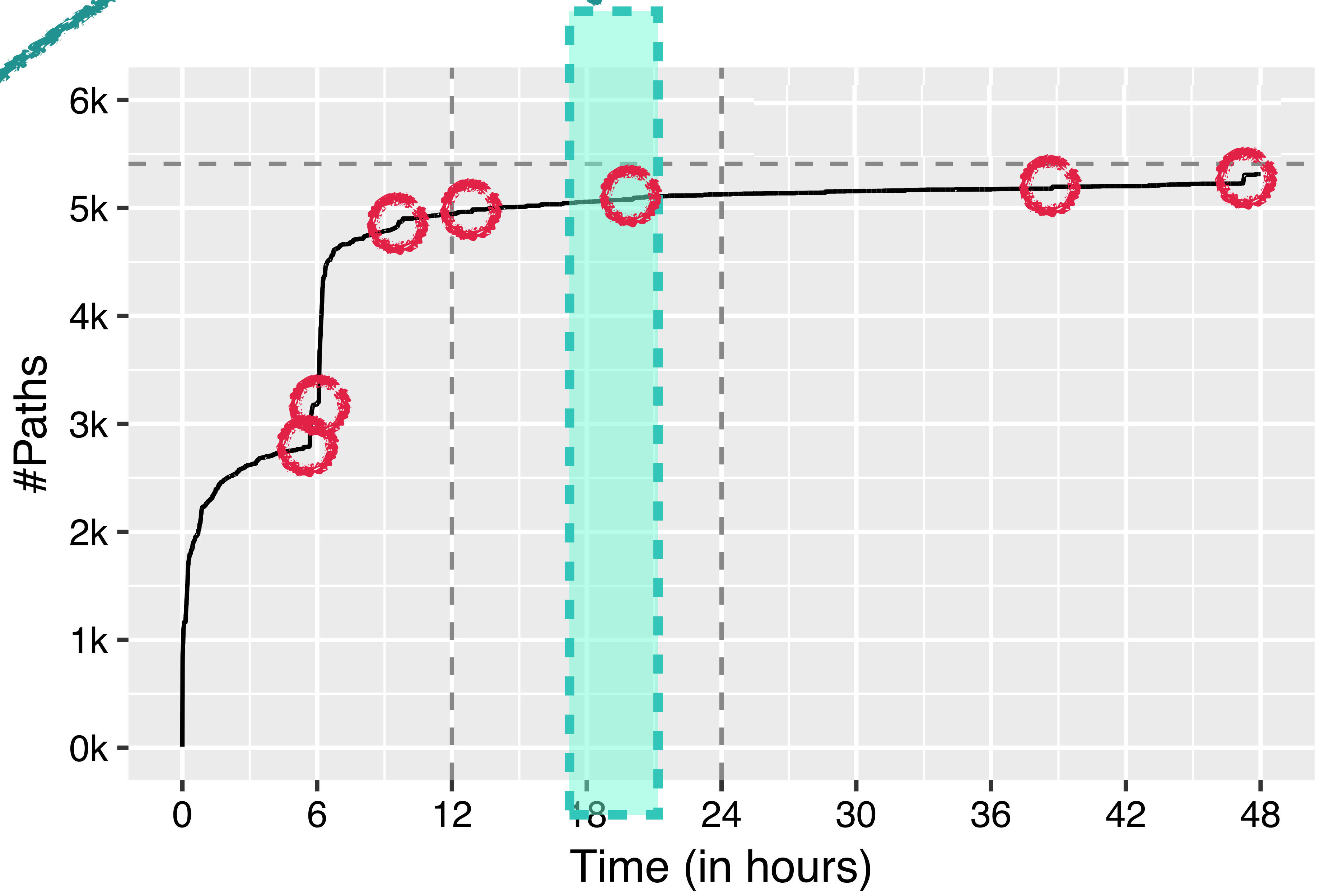
Coverage Increase Plot of the Greybox Fuzzing

Methodology

1



Roughly Blackbox



Coverage Increase Plot of
the Greybox Fuzzing

2

Regression Model

Methodology

Methodology

BB

	1	2	3	4	5	6	7	…	t_0
1	1	1	1	1	1	1	1	…	1
2	1	1	0	0	0	0	1	…	0
3	0	0	0	1	1	1	0	…	0
4	0	0	0	1	1	0	1	…	1
5	0	0	0	1	1	0	1	…	0
6	1	1	1	1	1	1	0	…	0
7	0	0	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…	…	…
S	0	1	1	1	1	0	0	…	0

Greybox coverage record

Methodology

Reset index

1	2	3	4	5
1	1	1	1	1
1	0	0	0	0
0	0	1	1	1
0	0	1	1	0
0	0	1	1	0
1	1	1	1	1
0	0	0	0	0
…	…	…	…	…
1	1	1	1	0

BB

	1	2	3	4	5	6	7	…	t_0
1	1	1	1	1	1	1	1	…	1
2	1	1	0	0	0	0	1	…	0
3	0	0	0	1	1	1	0	…	0
4	0	0	0	1	1	0	1	…	1
5	0	0	0	1	1	0	1	…	0
6	1	1	1	1	1	1	0	…	0
7	0	0	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…	…	…
S	0	1	1	1	1	0	0	…	0

Greybox coverage record **Sub-campaign**

Methodology

BB	1	2	3	4	5	6	7	…	t_0
1	1	1	1	1	1	1	1	…	1
2	1	1	0	0	0	0	1	…	0
3	0	0	0	1	1	1	0	…	0
4	0	0	0	1	1	0	1	…	1
5	0	0	0	1	1	0	1	…	0
6	1	1	1	1	1	1	0	…	0
7	0	0	0	0	0	0	0	…	0
…	…	…	…	…	…	…	…	…	…
S	0	1	1	1	1	0	0	…	0

Greybox coverage record

Reset index

1	2	3	4	5
1	1	1	1	1
1	0	0	0	0
0	0	1	1	1
0	0	1	1	0
0	0	1	1	0
1	1	1	1	1
0	0	0	0	0
…	…	…	…	…
1	1	1	1	0

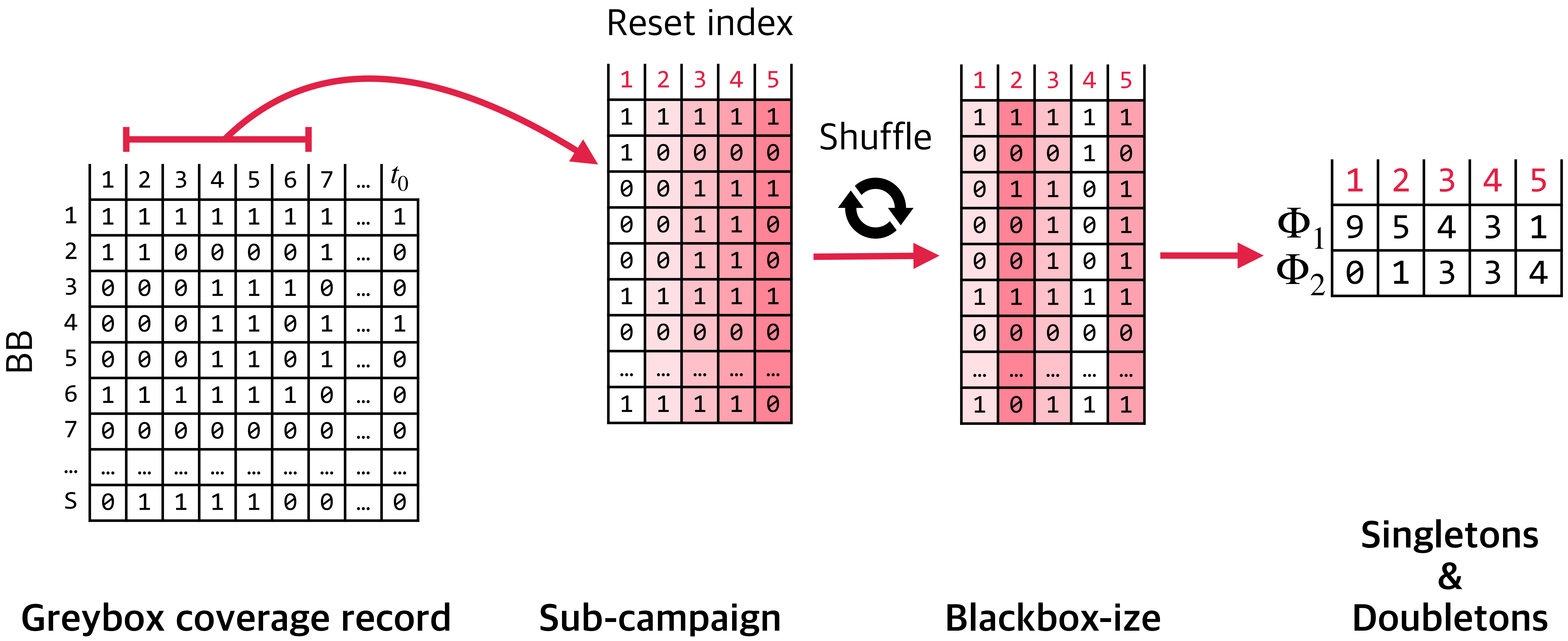
Sub-campaign

Shuffle

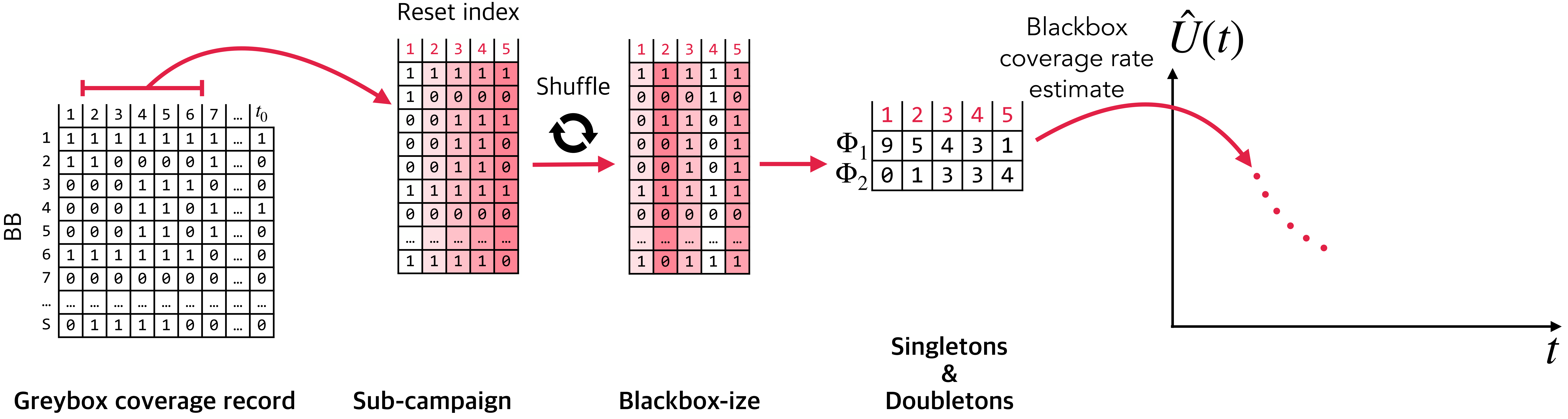
1	2	3	4	5
1	1	1	1	1
0	0	0	1	0
0	1	1	0	1
0	0	1	0	1
0	0	1	0	1
1	1	1	1	1
0	0	0	0	0
…	…	…	…	…
1	0	1	1	1

Blackbox-ize

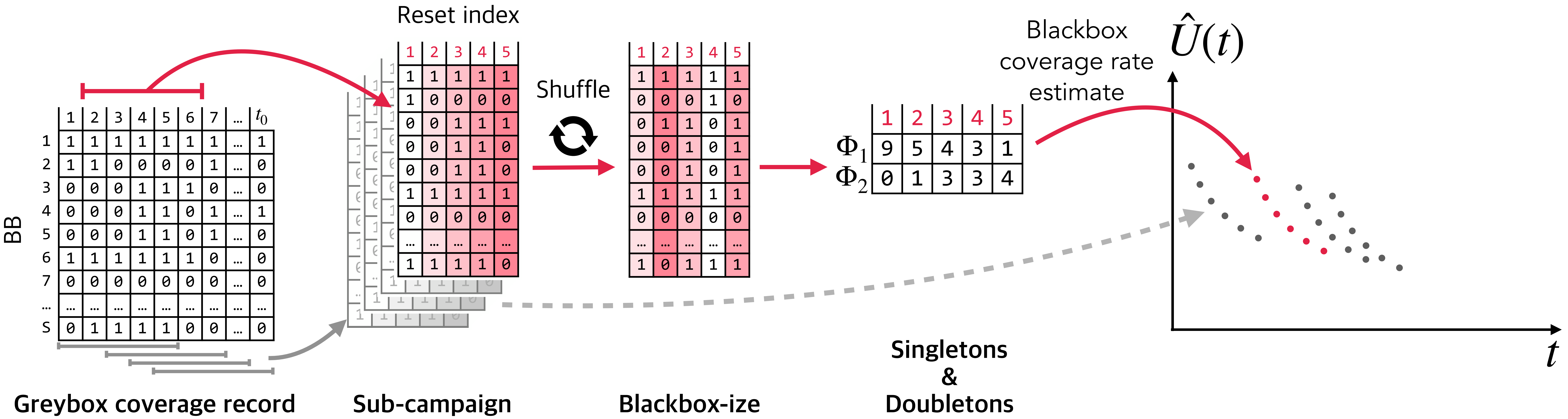
Methodology



Methodology



Methodology



Methodology

Methodology

Evaluation: Coverage Rate Prediction

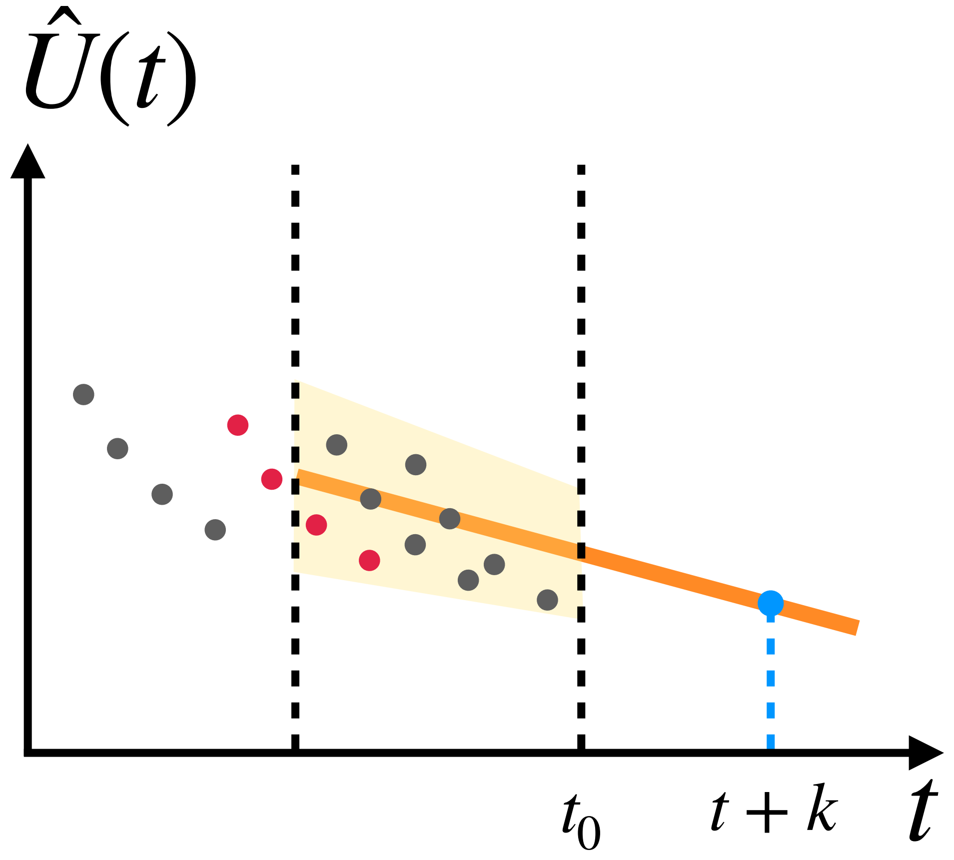
Q. How accurate is our ***regression model considering the adaptive bias*** compared to the ***existing blackbox extrapolation model***?

Existing $\hat{U}(t+k)$ **Extrapolator**

Ignores the adaptive bias

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{t\hat{\Phi}_0+\Phi_1}\right)^k\right]$$

VS



Our $\hat{U}(t+k)$ **Extrapolator**

Consider the adaptive bias

Evaluation: Coverage Rate Prediction

Q. How accurate is our ***regression model considering the adaptive bias*** compared to the ***existing blackbox extrapolation model***?

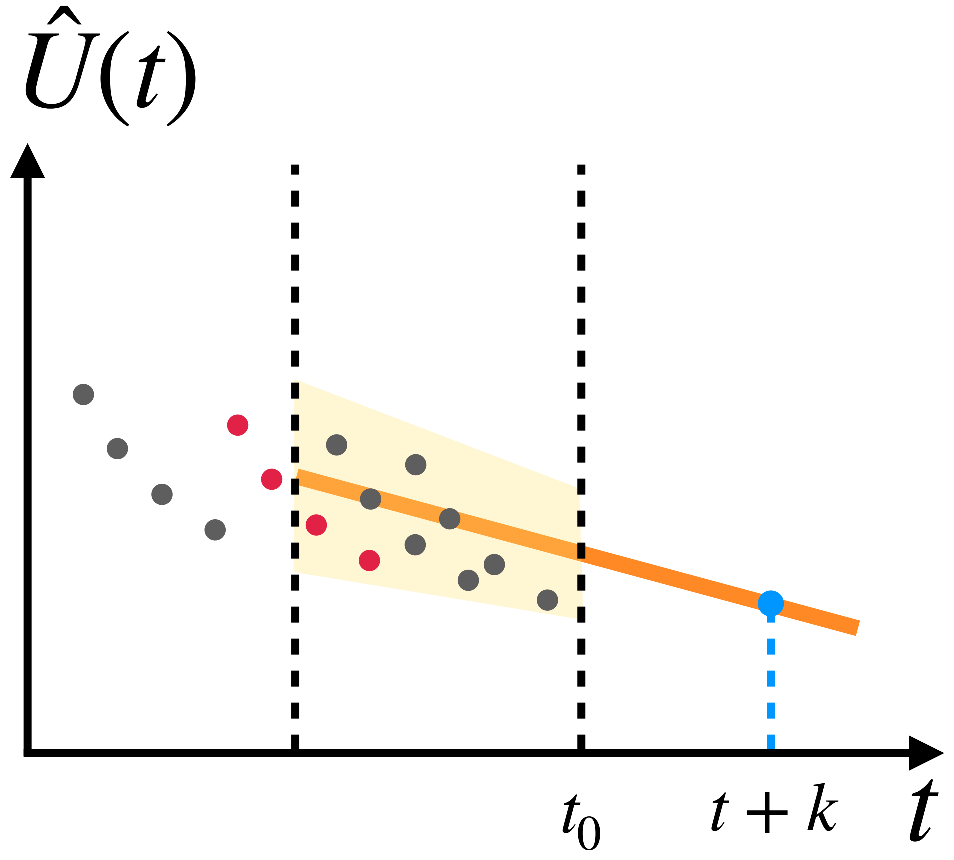
- Subject program: five open-source C libraries
- Evaluation Scenario:

Existing $\hat{U}(t+k)$ **Extrapolator**

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{t\hat{\Phi}_0+\Phi_1}\right)^k\right]$$

Ignores the adaptive bias

VS



Our $\hat{U}(t+k)$ **Extrapolator**

Consider the adaptive bias

Evaluation: Coverage Rate Prediction

Q. How accurate is our ***regression model considering the adaptive bias*** compared to the ***existing blackbox extrapolation model***?

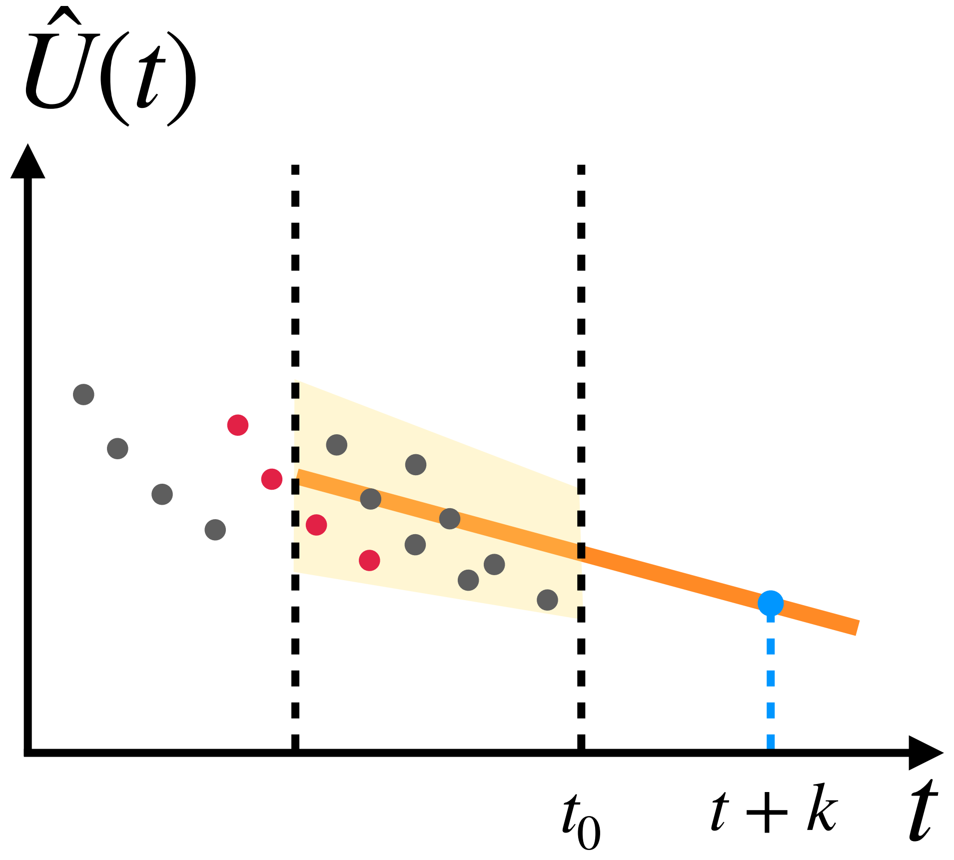
- Subject program: five open-source C libraries
- Evaluation Scenario: 1) run the greybox fuzzer until having t data points
  2) apply each extrapolator to extrapolate $\hat{U}(t+k)$
  3) run the greybox fuzzer for k more data points to get $U(t+k)$.

Existing $\hat{U}(t+k)$ ***Extrapolator***

Ignores the adaptive bias

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{t\hat{\Phi}_0+\Phi_1}\right)^k\right]$$

VS



Our $\hat{U}(t+k)$ ***Extrapolator***

Consider the adaptive bias

Evaluation: Coverage Rate Prediction

Q. How accurate is our ***regression model considering the adaptive bias*** compared to the ***existing blackbox extrapolation model***?

- Subject program: five open-source C libraries
- Evaluation Scenario: 1) run the greybox fuzzer until having t data points
  2) apply each extrapolator to extrapolate $\hat{U}(t+k)$
  3) run the greybox fuzzer for k more data points to get $U(t+k)$.

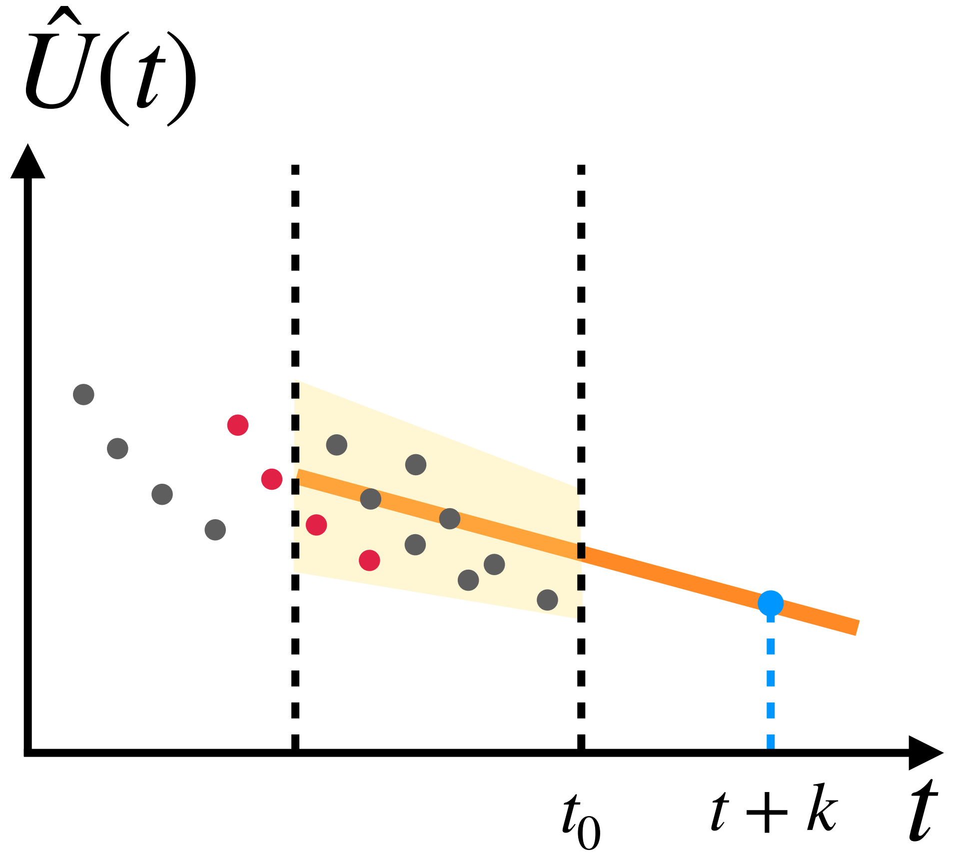
Compare

Existing $\hat{U}(t+k)$ ***Extrapolator***

Ignores the adaptive bias

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{t\hat{\Phi}_0+\Phi_1}\right)^k\right]$$

VS

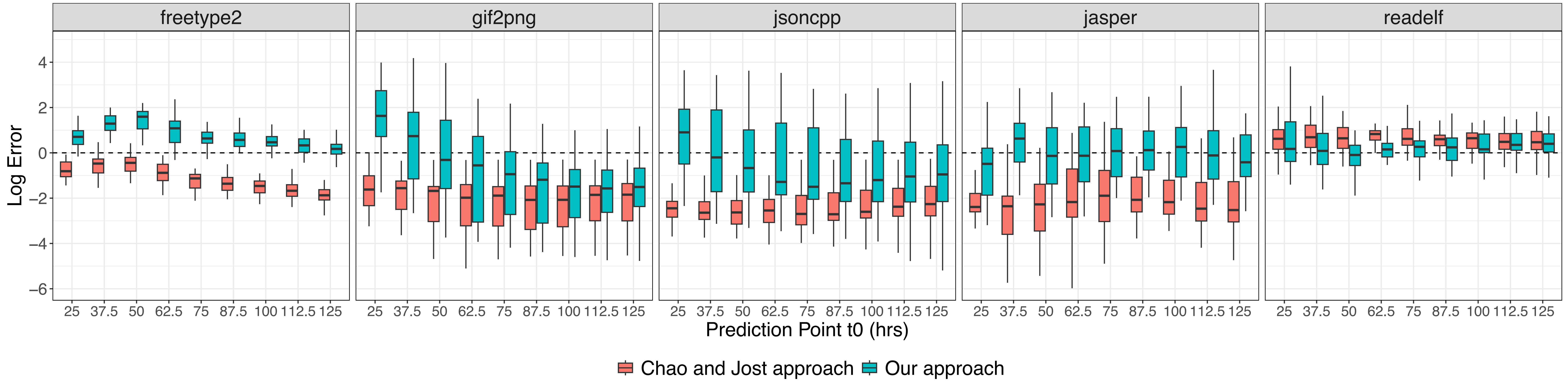


Our $\hat{U}(t+k)$ ***Extrapolator***

Consider the adaptive bias

Evaluation: Coverage Rate Prediction

Difference between $\log(U(t+k))$ ***vs.*** $\log(\hat{U}(t+k))$



"Our extrapolator exhibits ***at least one order of magnitude lower absolute bias*** than the existing extrapolator for *4 out of 5 subjects*, <u>especially for long-term prediction.</u>"

(close to 1 is better)

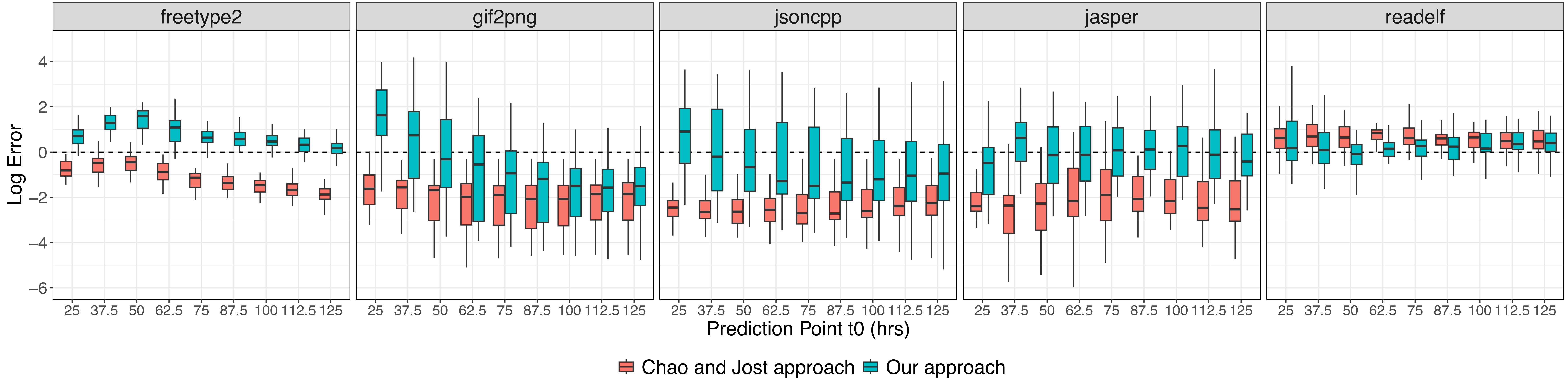
The average ratio $U(t+k)/\hat{U}(t+k)$:

[Existing] **1.6 - 800** [Ours] **1.17 - 7**

across all subjects.

Evaluation: Coverage Rate Prediction

Difference between $\log(U(t+k))$ ***vs.*** $\log(\hat{U}(t+k))$



(close to 1 is better)

"Our extrapolator exhibits
at least one order of magnitude lower absolute bias
than the existing extrapolator for *4 out of 5 subjects*,
<u>especially for long-term prediction.</u>"

⇒ ***Well-handled the adaptive bias***

The average ratio $U(t+k)/\hat{U}(t+k)$:

[Existing] **1.6 - 800** [Ours] **1.17 - 7**

across all subjects.

Extrapolating Coverage Rate in Greybox Fuzzing

Danushka Liyanage*
Monash University
Australia

Seongmin Lee*
MPI-SP
Germany

Chakkrit Tantithamthavorn
Monash University
Australia

Marcel Böhme
MPI-SP
Germany

ABSTRACT

A fuzzer can literally run forever. However, as more resources are spent, the coverage rate continuously drops, and the utility of the fuzzer declines. To tackle this coverage-resource tradeoff, we could introduce a policy to stop a campaign whenever the coverage rate drops below a certain threshold value, say 10 new branches covered per 15 minutes. During the campaign, can we predict the coverage rate at some point in the future? If so, how well can we predict the future coverage rate as the prediction horizon or the current campaign length increases? How can we tackle the statistical challenge of adaptive bias, which is inherent in greybox fuzzing (i.e., samples are not independent and identically distributed)?

In this paper, we i) evaluate existing statistical techniques to predict the coverage rate $U(t_0 + k)$ at any time t_0 in the campaign after a period of k units of time in the future and ii) develop a new extrapolation methodology that tackles the adaptive bias. We propose to efficiently simulate a large number of blackbox campaigns from the collected coverage data, estimate the coverage rate for each of these blackbox campaigns and conduct a simple regression to extrapolate the coverage rate for the greybox campaign.

Our empirical evaluation using the Fuzztastic fuzzer benchmark demonstrates that our extrapolation methodology exhibits at least one order of magnitude lower error compared to the existing benchmark for 4 out of 5 experimental subjects we investigated. Notably, compared to the existing extrapolation methodology, our extrapolator excels in making long-term predictions, such as those extending up to three times the length of the current campaign.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; • **Security and privacy → Software security engineering**.

KEYWORDS

greybox fuzzing, extrapolation, coverage rate, adaptive bias, statistical method

*Both authors contributed equally to this research.

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3639198

1 INTRODUCTION

At the turn of the millennium, the late Mary-Jean Harrold drew a research roadmap for the software testing community of the future [13]. She highlighted the "development of techniques and tools for use in estimating, predicting, and performing testing on evolving software systems" as one of five research pointers. While there has been some recent progress in the estimation of pertinent quantities in the testing process, we have yet to start exploring methodologies for prediction.

The rate at which new coverage is achieved is considered a fundamental measure of the efficiency of a fuzzing campaign. A fuzzer is an automated software testing tool, and with increasing coverage, we mean the generation of inputs that cover new program elements, such as a branch or a statement. If the coverage rate drops below a certain threshold, the tester will abort the ongoing fuzzing campaign for the lack of progress. Terminating a fuzzing campaign early will help release computational resources and reduce the carbon footprint [17, 26]. If, throughout the campaign, the tester could accurately predict the coverage rate at some point in the future, they could conduct a cost-benefit analysis to assess the resources required to achieve the targeted testing progress. Since fuzzing is a preliminary testing technique that constitutes sophisticated testing frameworks (e.g., a hybrid/ensemble fuzzing, an automated test case generation framework, etc.), such a prediction would allow the tester to adequately allocate resources (time and computing power) for the entire testing process in advance [29].

One of the most successful fuzzing techniques is called greybox fuzzing, which takes a mutation-based, coverage-guided approach. A greybox fuzzer is *mutation-based* because it uses a corpus of program inputs that are randomly mutated to slightly corrupt the seed file while preserving much of the unknown but required input format. A greybox fuzzer is *coverage-guided* because it adds generated inputs to the corpus that have been observed to increase coverage. The hope is that an input generated from a coverage-increasing input is itself more likely coverage-increasing. Since the probability of covering a specific program element changes in this process, the underlying distribution over these elements is *not* invariant. However, invariance is a key assumption in most statistical estimation and extrapolation methodologies. Hence, a key *statistical challenge* in the domain of greybox fuzzing is thus to tackle the resulting *adaptive bias*.

In this paper, we introduce a novel extrapolation methodology that allows us to predict the coverage rate $U(t_0 + mt_0)$ in a greybox campaign of length t_0 if the campaign length was extended m more times while accounting for adaptive bias. We systematically select

- ***Extrapolating Coverage Rate in Greybox Fuzzing*** *Danushka Liyanage*, Seongmin Lee*, Chakkrit Tantithamthavorn, and Marcel Böhme. ICSE 2024.*
  - *Extrapolate the future progress of the greybox fuzzing* by handling the adaptive bias through introducing a *regression model* over *predictions on subcampaigns*.

[The Fundamental Problem of **Software Testing**]

"There is always ***unseen****."*

[The Fundamental Problem of **Software Testing**]

*"There is always **unseen**."*

Questions about the **unseen**

Red : semantic meaning
Black : concrete task

***How much of a proportion of the behavior** have we tested in this program?*

*What is the **probability** of observing **a new coverage or a new bug**?*

How many unobserved vulnerabilities are remaining?

*What is the **maximum coverage** we can achieve?*

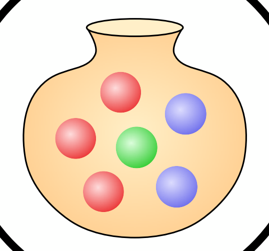
Should we keep running the testing/fuzzing?

*How much more can I achieve **if I spend** X **more time** here?*

[The Fundamental Problem of **Software Testing**]

"There is always ***unseen****."*

Questions about the unseen in an **Urn filled with Balls**



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

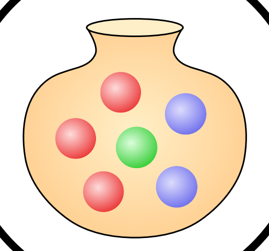
Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

[**The Fundamental Problem of Software Testing**]

“There is always ***unseen****.”*

Questions about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

Missing Mass

What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

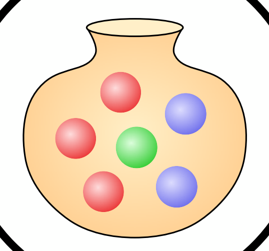
Extrapolation

How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

[**The Fundamental Problem of Software Testing**]

"There is always ***unseen****."*

Questions about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

Solution:

Good-Turing estimator

Alan Turing

$$\hat{M}_0 = \frac{\Phi_1}{n}$$

colors only seen once in samples

of *singleton colors

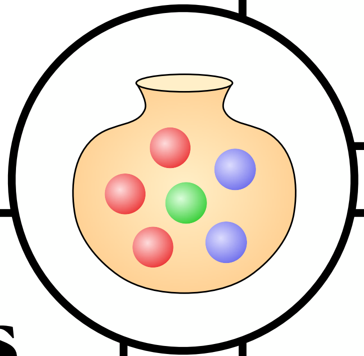
of samples

The estimation of the probability of our following sample is something that has never been seen before.

[The Fundamental Problem of Software Testing]

*“There is always **unseen**.”*

Questions about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

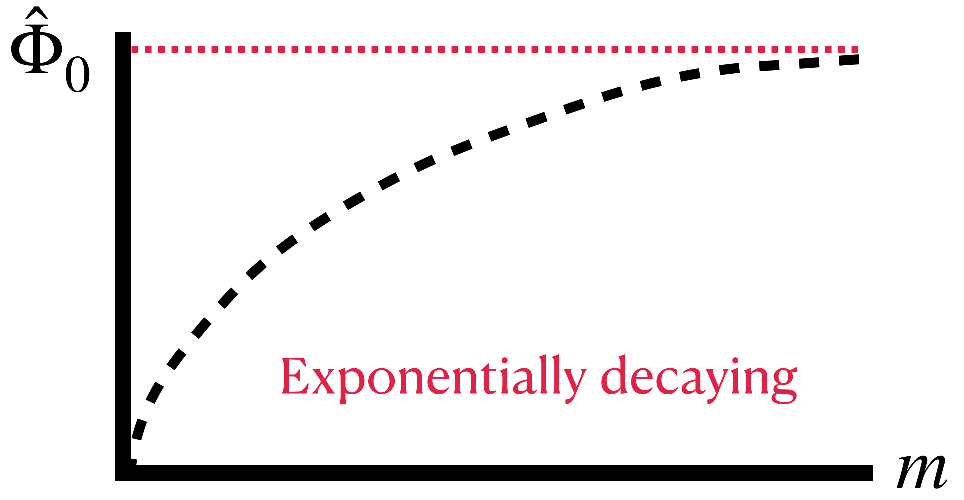
$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

Φ_1: the number of singletons

Φ_2: the number of doubletons

$\Delta(m)$: the number of new discoveries when m more samples are retrieved

$$\hat{\Delta}(m) = \hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$



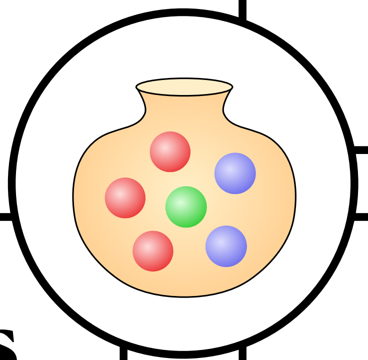
Anne Chao and Robert K Colwell. 2017. Thirty years of progeny from Chao's inequality: Estimating and comparing richness with incidence data and incomplete sampling. SORT 41

Anne Chao and Lou Jost. 2012. Coverage-based rarefaction and extrapolation: standardizing samples by completeness rather than size. Ecology 93

The Fundamental Problem of Software Testing

*"There is always **unseen**."*

Questions about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

Check how the __statistical estimator__ can measure the __unseen in software testing__.

Missing Mass

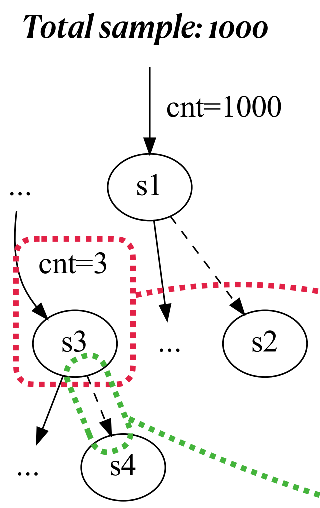
*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

*How much more can I achieve **if I spend** X **more time** here?*

Our Solution: Structure-aware Reachability Estimator

- Approach: reflect the *(control) dependence relation* between the program states.



Blackbox estimator:

$$\cancel{\hat{\Pr}_{BB}(s_4, O)} = \hat{\Pr}_{BB}(s_2, O)$$

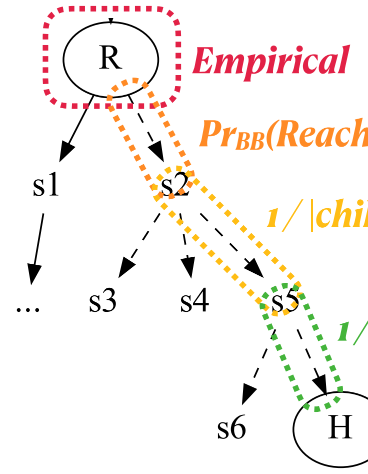
Structure-aware:

$$\hat{\Pr}_{St}(s_2) = \hat{\Pr}_{Emp}(s_1, O) \times \hat{\Pr}_{BB}(s_2, O')$$

where, $O' = \{o \in O \mid \text{Reach}(o, s_1)\}$

$$\hat{\Pr}_{St}(s_4) = \underbrace{\hat{\Pr}_{Emp}(s_3, O)}_{\text{Pr(Pred)}} \times \underbrace{\hat{\Pr}_{BB}(s_4, O'')}_{\text{Pr(Pred→Next)}}$$

where, $O'' = \{o \in O \mid \text{Reach}(o, s_3)\}$

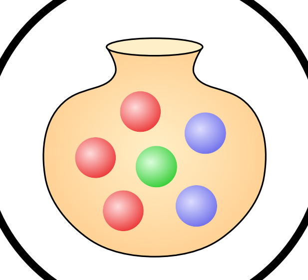


$$\hat{\Pr}_{St}(H) = \boxed{\hat{\Pr}_{Emp}(R, O)} \times \boxed{\hat{\Pr}_{BB}(s_2, O^R)} \times \boxed{\frac{1}{3}} \times \boxed{\frac{1}{2}}$$

[**The Fundamental Problem of Software Testing**]

*"There is always **unseen**."*

Questions about the unseen in an Urn filled with Balls



Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$$

1

Check how the ***statistical estimator*** *can measure the* ***unseen in software testing****.*

Missing Mass

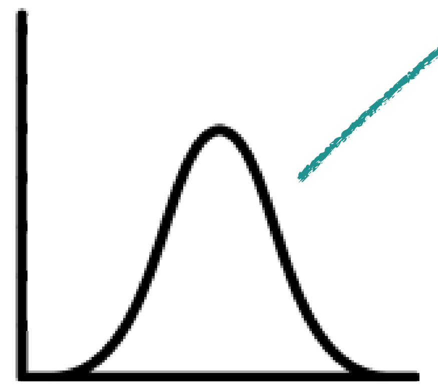
What is the ***probability*** *of observing* ***a new coverage or a new bug****?*

Extrapolation

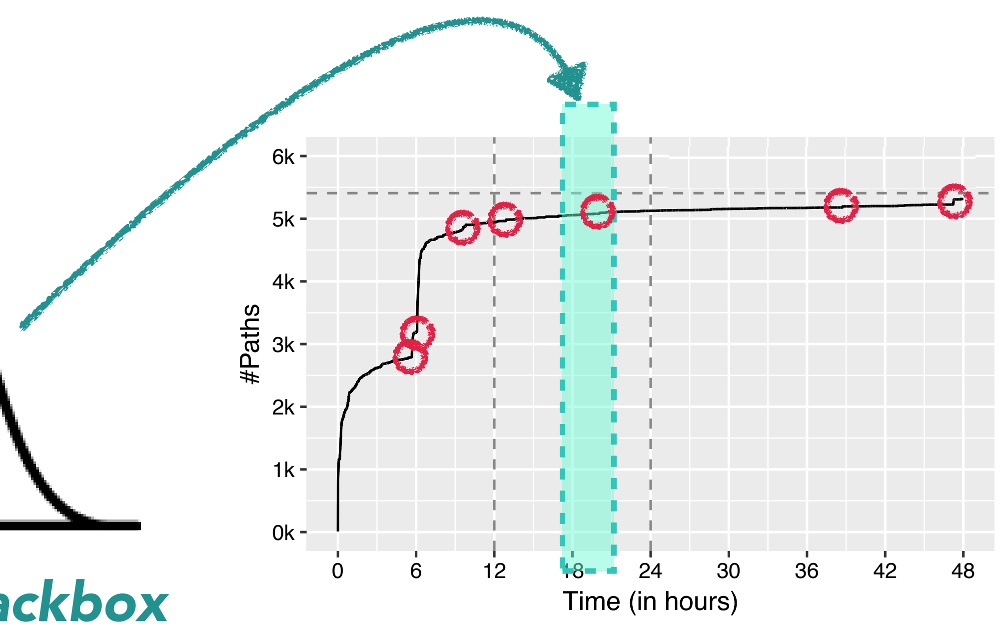
How much more can I achieve ***if I spend*** *X* ***more time*** *here?*

Methodology

1

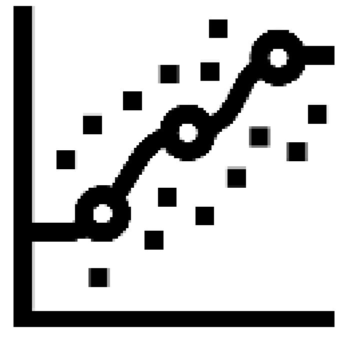


Roughly Blackbox

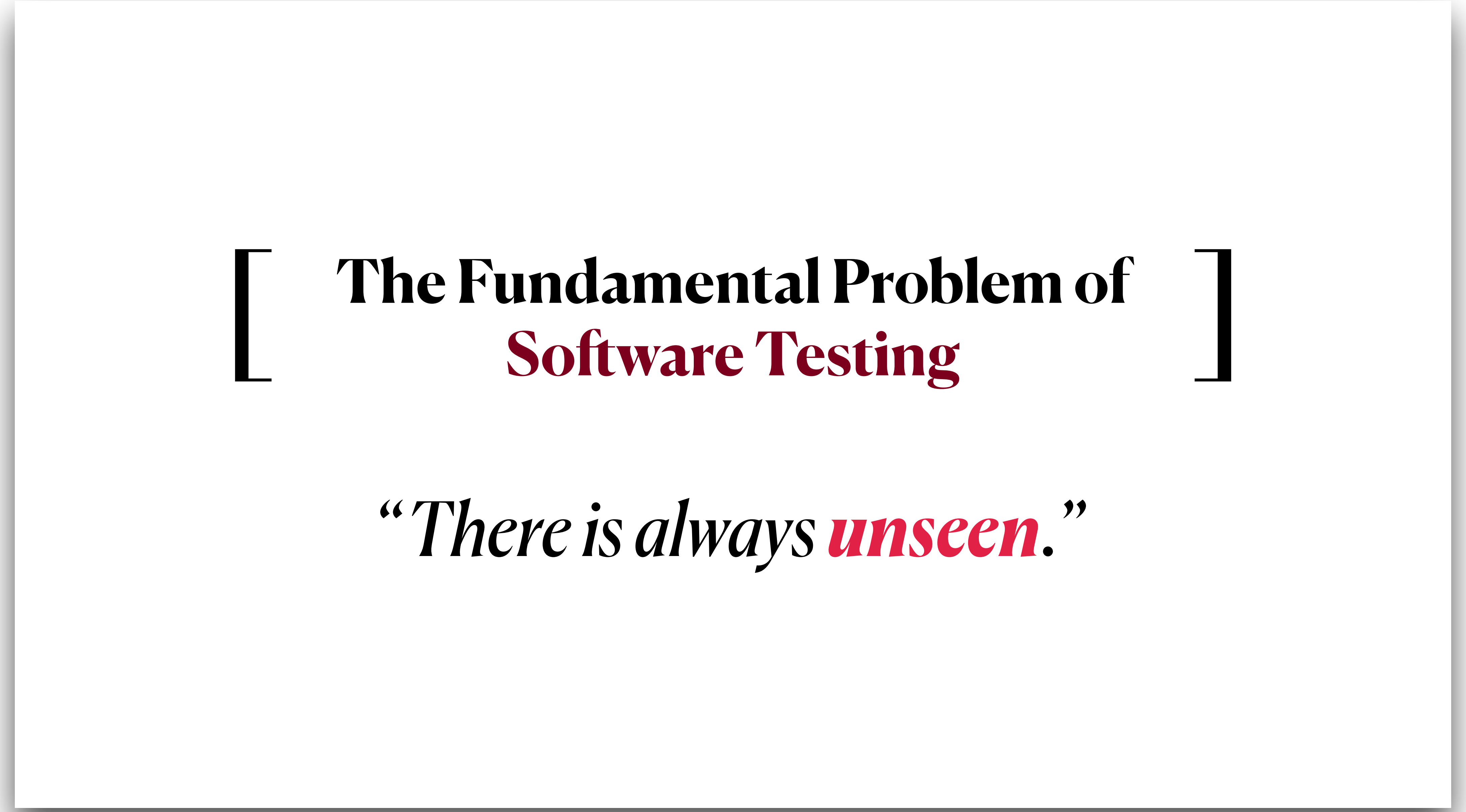


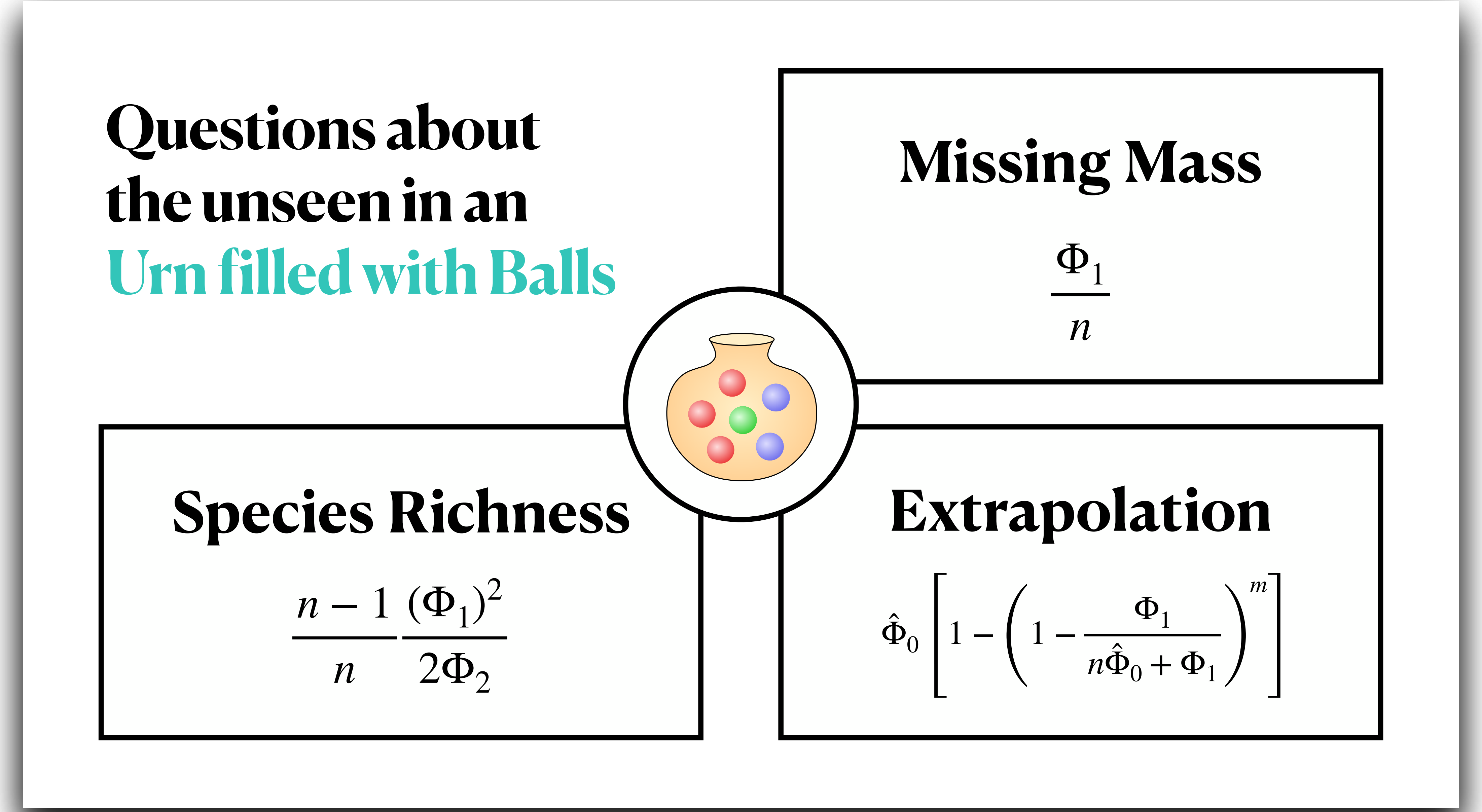
Coverage Increase Plot of the Greybox Fuzzing

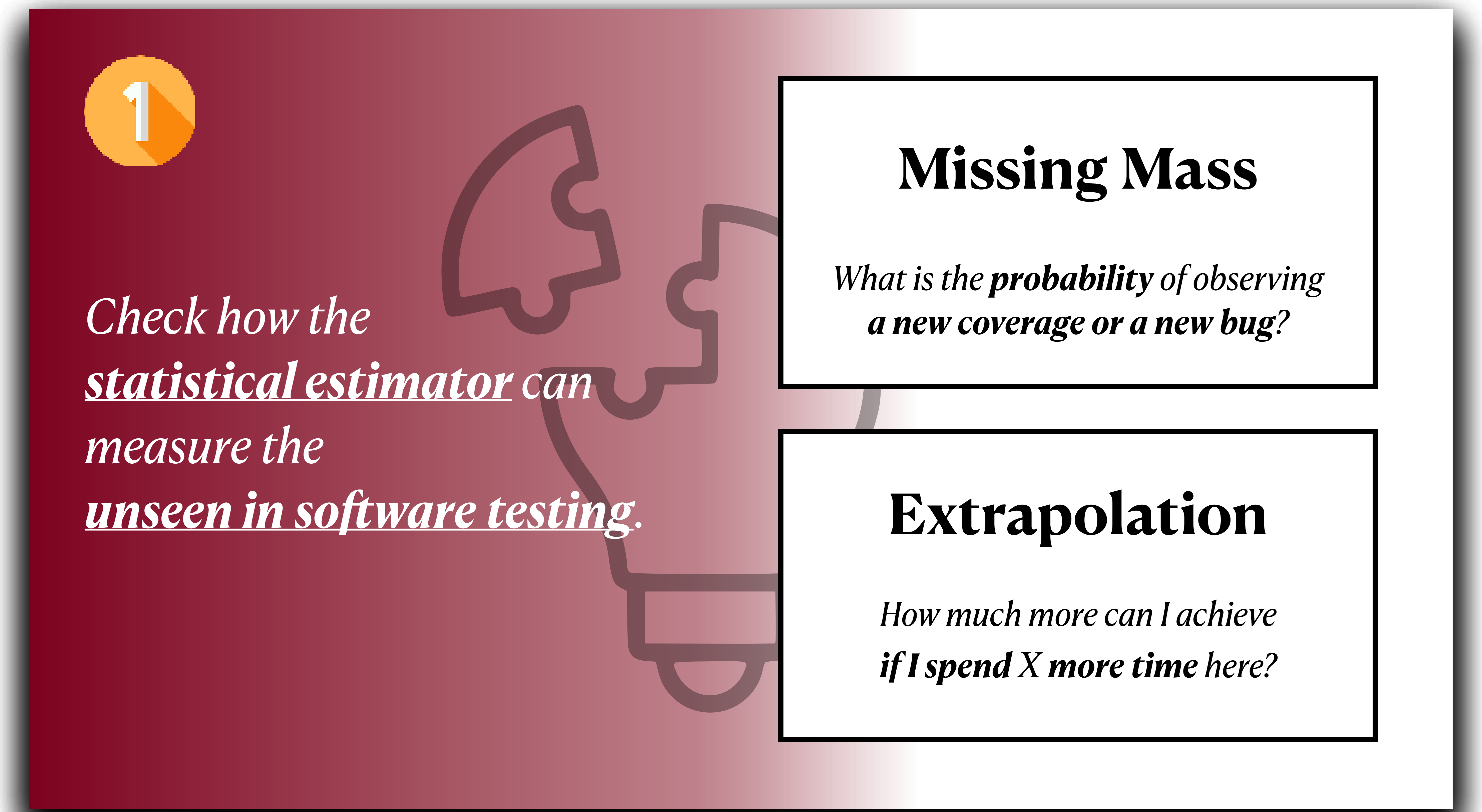
2

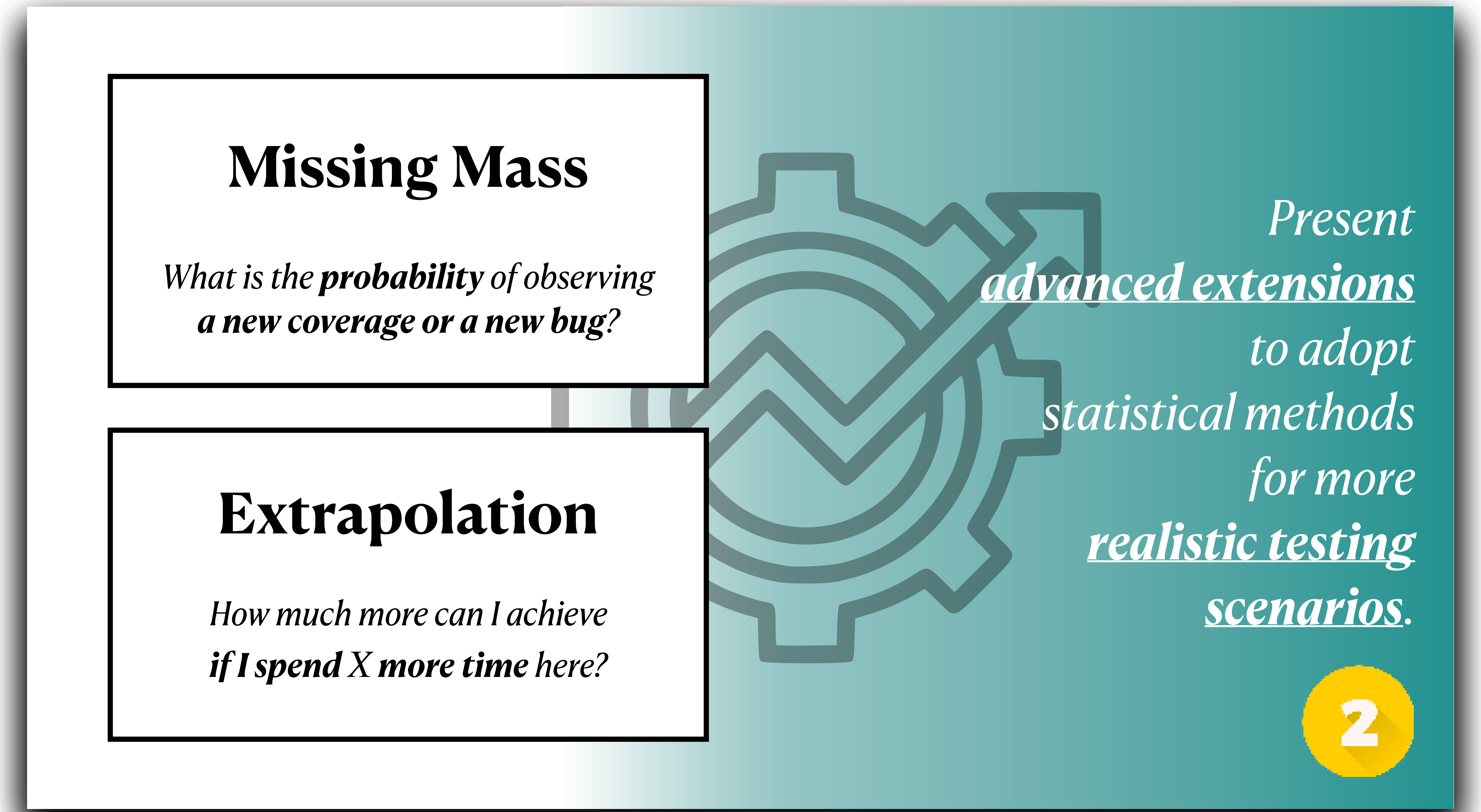


Regression Model


The Fundamental Problem of
Software Testing
"There is always unseen."


Questions about
the unseen in an
Urn filled with Balls
Missing Mass
$\frac{\Phi_1}{n}$
Species Richness
$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$
Extrapolation
$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^m\right]$

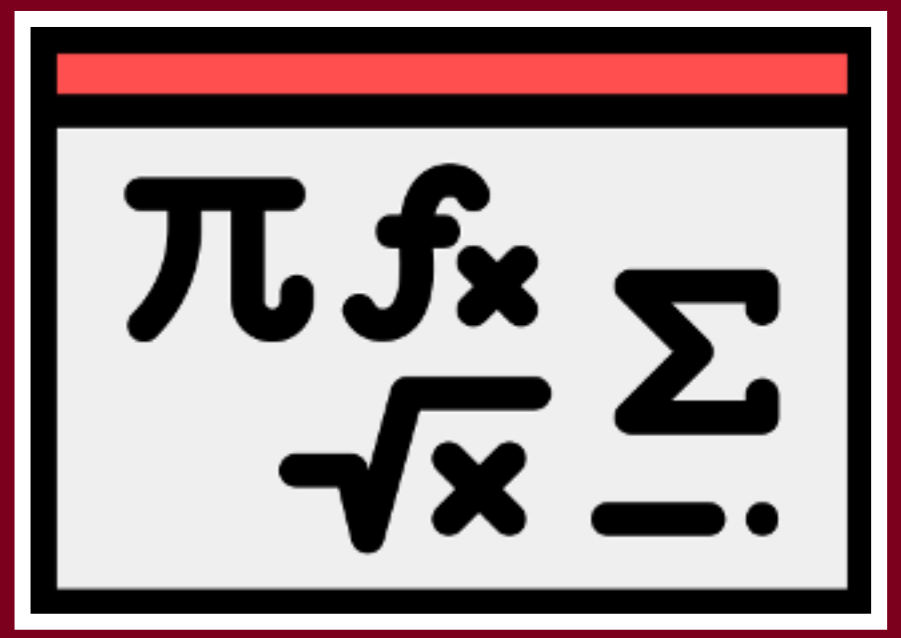

1
Check how the
statistical estimator can
measure the
unseen in software testing.
Missing Mass
What is the probability of observing
a new coverage or a new bug?
Extrapolation
How much more can I achieve
if I spend X more time here?


Missing Mass
What is the probability of observing
a new coverage or a new bug?
Extrapolation
How much more can I achieve
if I spend X more time here?
Present
advanced extensions
to adopt
statistical methods
for more
realistic testing
scenarios.
2

How can we assure the

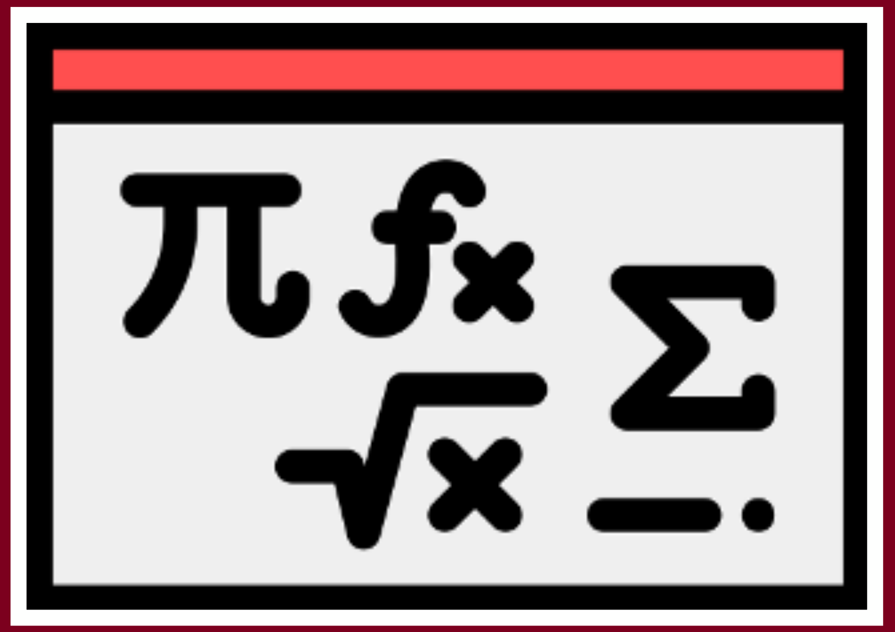
Quality of Software

Analytical Methods



Mathematical proof can provide a ***formal guarantee***

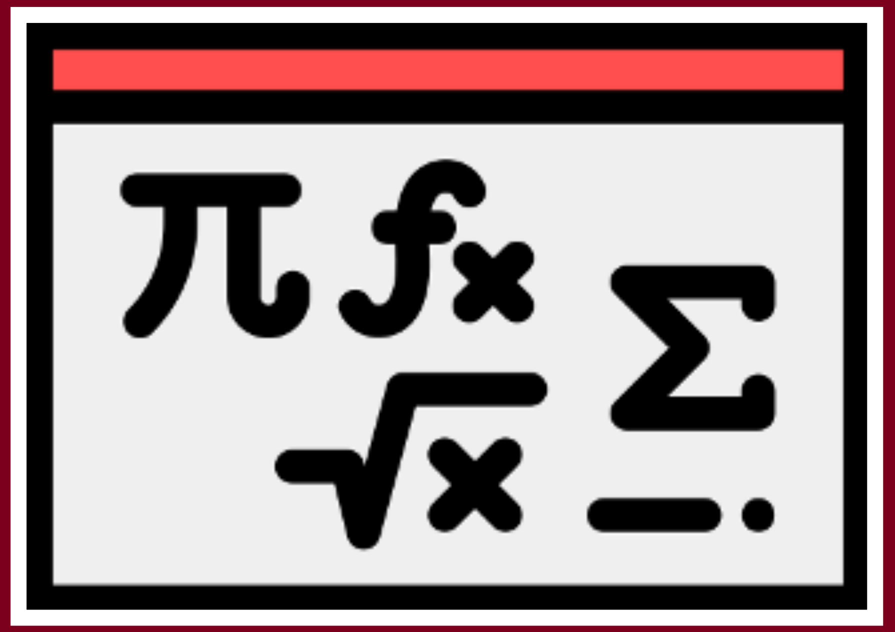
Analytical Methods



Mathematical proof can provide a ***formal guarantee***

Scalability issues on modern software

Analytical Methods



Mathematical proof can provide a ***formal guarantee***

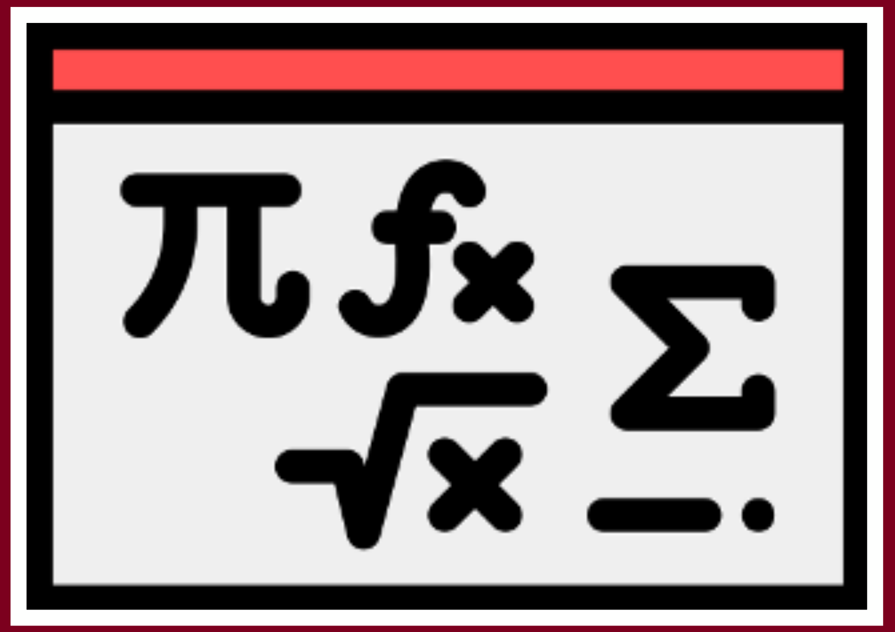
Scalability issues on modern software

Empirical Methods

Test software by running it with various test executions

By actually running the software, it solves the ⚠️ scalability issue

Analytical Methods



Mathematical proof can provide a ***formal guarantee***

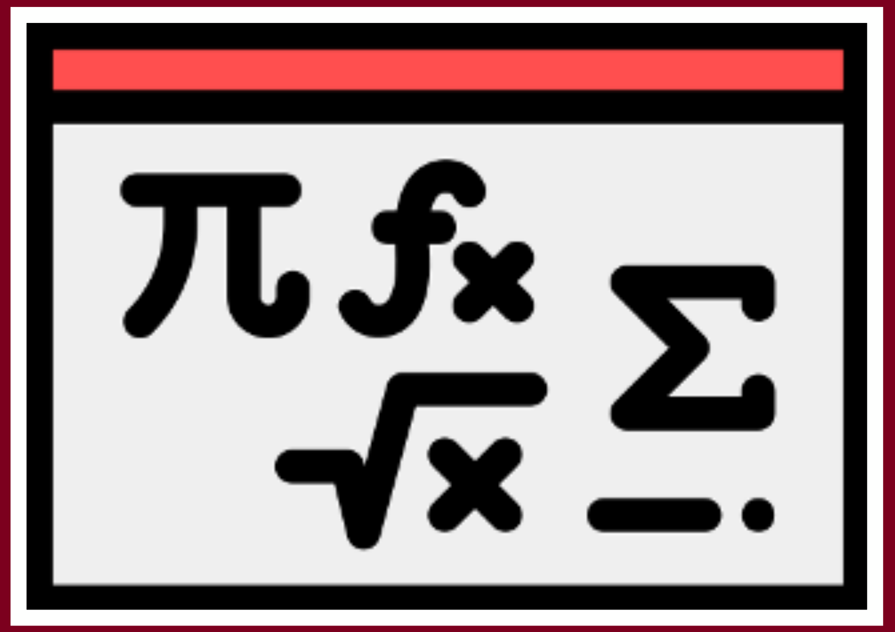
⚠ Scalability issues on modern software

Empirical Methods

Test software by running it with various test executions

By actually running the software, it solves the ⚠ scalability issue

Analytical Methods



Mathematical proof can provide a ***formal guarantee***

Scalability issues on modern software

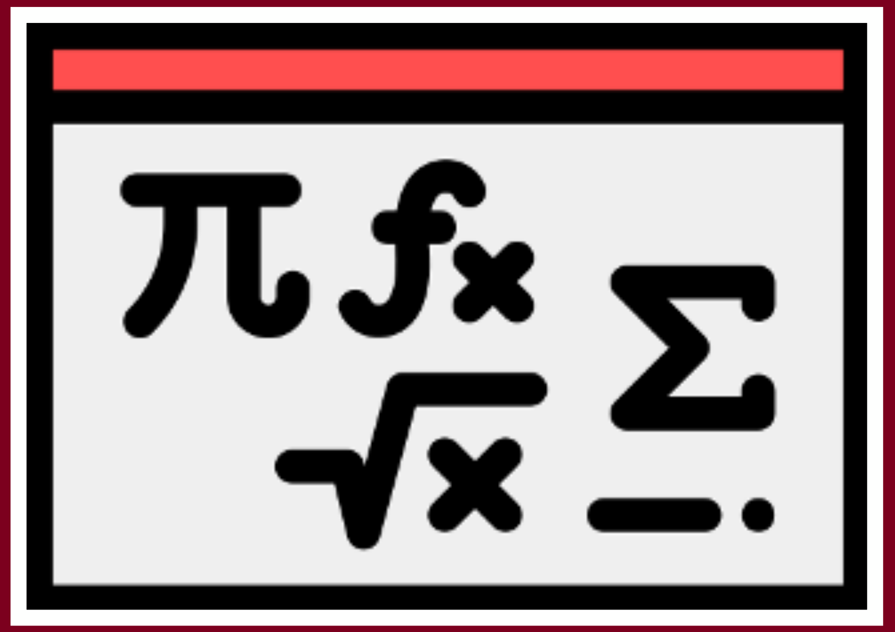
Empirical Methods

Test software by running it with various test executions

By actually running the software, it solves the ⚠️ scalability issue

There is always **unseen**
⇒ ***No guarantee***

Analytical Methods



Mathematical proof can provide a ***formal guarantee***

Scalability issues on modern software

Empirical Methods

Test software by running it with various test executions

By actually running the software, it solves the ⚠ scalability issue

There is always unseen
⇒ ***No guarantee***

Statistics can solve this!

"Data-driven analysis to ensure the correctness of software"

Statistical Software Quality Assurance

“Data-driven analysis to ensure the correctness of software”

Statistical Software Quality Assurance

"Data-driven analysis to ensure the correctness of software"

Statistical Software Quality Assurance

Evolving Software

Guarantee for Rare State

"Data-driven analysis to ensure the correctness of software"

Statistical Software Quality Assurance

Evolving Software

Guarantee for Rare State

Modularization

Adaptation

Analytic + Statistic

“Data-driven analysis to ensure the correctness of software”

Statistical Software Quality Assurance

ChatGPT

Software Engineering

ML models are already widely used in SE research/practice.

"Data-driven analysis to ensure the correctness of software"

Statistical Software Quality Assurance

ChatGPT

Software Engineering

ML models are already widely used in SE research/practice.

Magic of Statistics for Software Testing: How to Foresee the Unseen

[The Fundamental Problem of Software Testing]

*"There is always **unseen**."*

Questions about the unseen in an Urn filled with Balls

Missing Mass

$$\frac{\Phi_1}{n}$$

Species Richness

$$\frac{n-1}{n}\frac{(\Phi_1)^2}{2\Phi_2}$$

Extrapolation

$$\hat{\Phi}_0\left[1-\left(1-\frac{\Phi_1}{n\hat{\Phi}_0+\Phi_1}\right)^{m}\right]$$

1

*Check how the **statistical estimator** can measure the **unseen in software testing**.*

Missing Mass

*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

*How much more can I achieve **if I spend X more time** here?*

Missing Mass

*What is the **probability** of observing **a new coverage or a new bug**?*

Extrapolation

*How much more can I achieve **if I spend X more time** here?*

*Present **advanced extensions** to adopt statistical methods for more **realistic testing scenarios**.*

2

Dr. Seongmin Lee

🏠 https://nimgnoeseel.github.io/

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY