



# Grammar-Based Fuzzing + Bug Explanations

Software Engineering II

04.05.2025

Martin Eberlein  
Humboldt-Universität zu Berlin, Germany



# Agenda

## Fuzzing 101

Automatic Input Generation

## Grammar based Fuzzing

Advanced Input Generation

## Automatic Debugging

Automatically Explaining the  
Circumstances of Program Faults

Martin Eberlein, Doctoral Researcher, SE 2 (2025)

# Grammar-Based Fuzzing

Quality Assurance



# Basic Fuzzing

## Generating Program Input 101

- **Software Testing Technique**  
Many different flavors: Black-, Grey, and White-box fuzzer
- **Basic Fuzzer *randomly* generate Input**  
Most popular fuzzers: AFL (AFL++), libfuzzer, ...
- **Fuzzing can be guided by properties**  
Most common strategies: code coverage, diverse inputs, ...
- **Simple Fuzzers can easily be written by yourself!**  
Jupyter-Notebook





# Basic Fuzzing

## Generating Program Input 101

- Simple Fuzzers can easily be written by yourself!

Jupyter-Notebook



```
localhost
jupyter GrammarBasedFuzzing (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel) O
Part 1: Simple Fuzzing
First, let's start with a simple fuzzing function that generates random strings.

In [2]: import string
import random

def simple_fuzzer(max_length=50, char_set=string.ascii_letters + string.punctuation):
    """A simple fuzzer that creates a string of random characters."""
    length = random.randint(0, max_length)
    return ''.join(random.choice(char_set) for _ in range(length))

Now, we simply need to run our fuzzing function simple_fuzzer() multiple times and use the output to test the program or service we're interested in.

In [3]: for _ in range(10):
    print(simple_fuzzer())
t\z|HTD^jq&_6E;=fuP~\eZw
,7UAY(Zb)7]s^4DWbU!~i_/0+:Q
dG1S(XQI@G|8T
|!SZ9Mg7ciQD4,DV,ux%k
fd1@T@o:P+u^Z@:U-T1[#]nLfo@IE(T,1ENO]m
```



# Grammar-Based Fuzzing

Like *Fuzzing* with extra steps

- **Most applications require structured input**

Web applications, interpreters, and parser require syntactically valid input

- **Guided by Grammars**

Input generation can be guided by context free grammars to produce syntactically valid inputs

- **More Efficient!**

Input generation can be guided by context free grammars to produce syntactically valid inputs

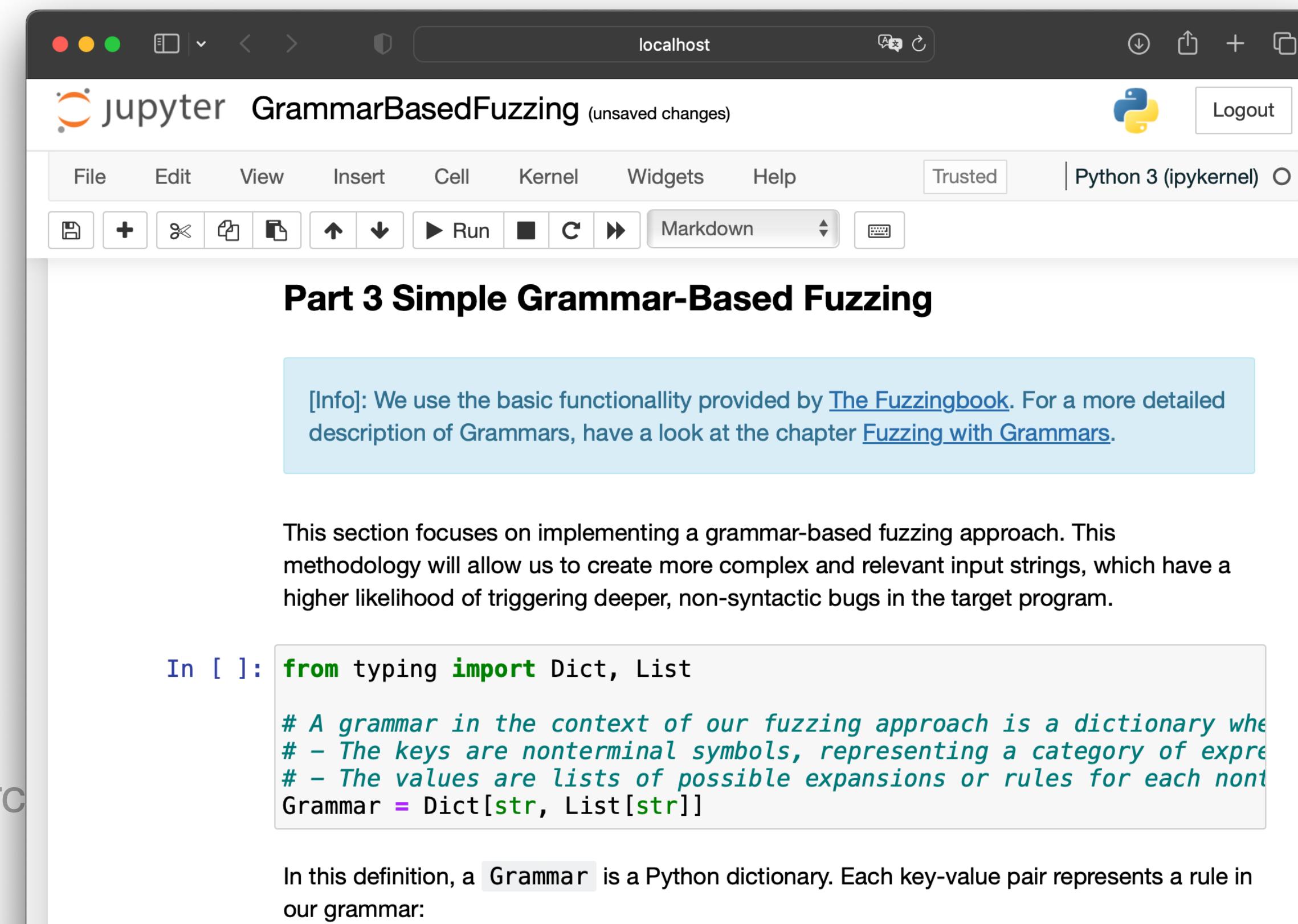
# Grammar-Based Fuzzing

Like *Fuzzing* with extra steps

- More Efficient!



Input generation can be guided by context free grammars to produce syntactically valid inputs



The screenshot shows a Jupyter Notebook interface with the title bar "localhost" and "jupyter GrammarBasedFuzzing (unsaved changes)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3 (ipykernel), and a Logout button. Below the toolbar is a toolbar with icons for file operations, cell creation, and execution. The main content area is titled "Part 3 Simple Grammar-Based Fuzzing". A blue info box contains the text: "[Info]: We use the basic functionality provided by [The Fuzzingbook](#). For a more detailed description of Grammars, have a look at the chapter [Fuzzing with Grammars](#)." The text below explains the methodology: "This section focuses on implementing a grammar-based fuzzing approach. This methodology will allow us to create more complex and relevant input strings, which have a higher likelihood of triggering deeper, non-syntactic bugs in the target program." A code cell labeled "In [ ]:" contains the following Python code:

```
In [ ]: from typing import Dict, List

# A grammar in the context of our fuzzing approach is a dictionary where
# - The keys are nonterminal symbols, representing a category of expressions
# - The values are lists of possible expansions or rules for each non-terminal symbol
Grammar = Dict[str, List[str]]
```

At the bottom, a note states: "In this definition, a `Grammar` is a Python dictionary. Each key-value pair represents a rule in our grammar."



# Probabilistic Grammar-Based Fuzzing

## Guided by Probabilities

### - More Efficient!

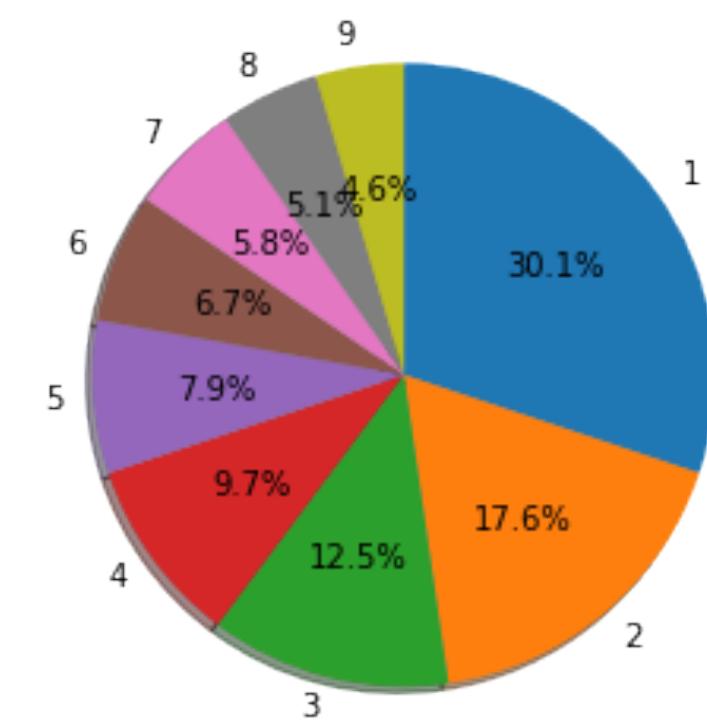
Input generation can be guided by context free grammars to produce syntactically valid inputs

### - Observation:

Some Features are more important than others

### - Example: Law of the Leading Digits

- In real-world data sets of numerical data, the leading digit is likely to be small
- [Newcomb 1881, Benford 1938]



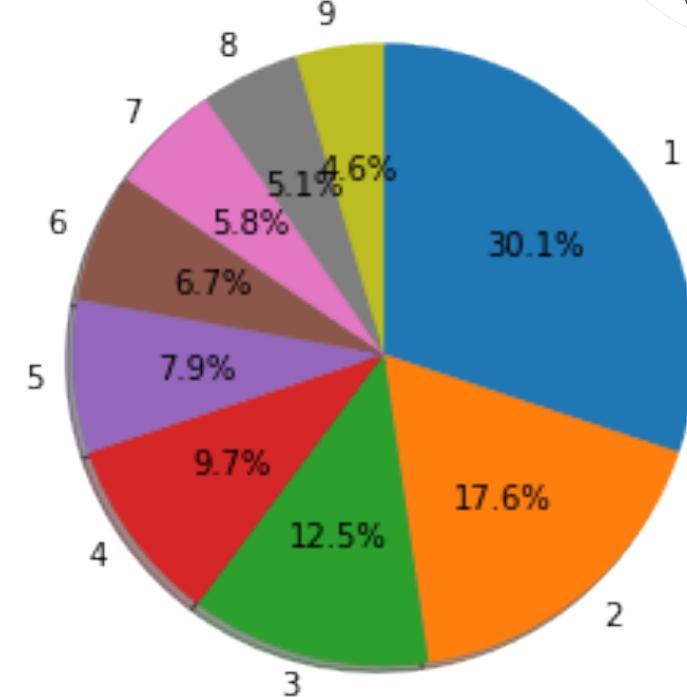


# Probabilistic Grammar-Based Fuzzing

## Guided by Probabilities

### - Example: Law of the Leading Digits

- In real-world data sets of numerical data, the leading digit is likely to be small
- [Newcomb 1881, Benford 1938]



The screenshot shows a Jupyter Notebook interface with the following content:

### Part 4: Probabilistic Grammar-Based Fuzzing

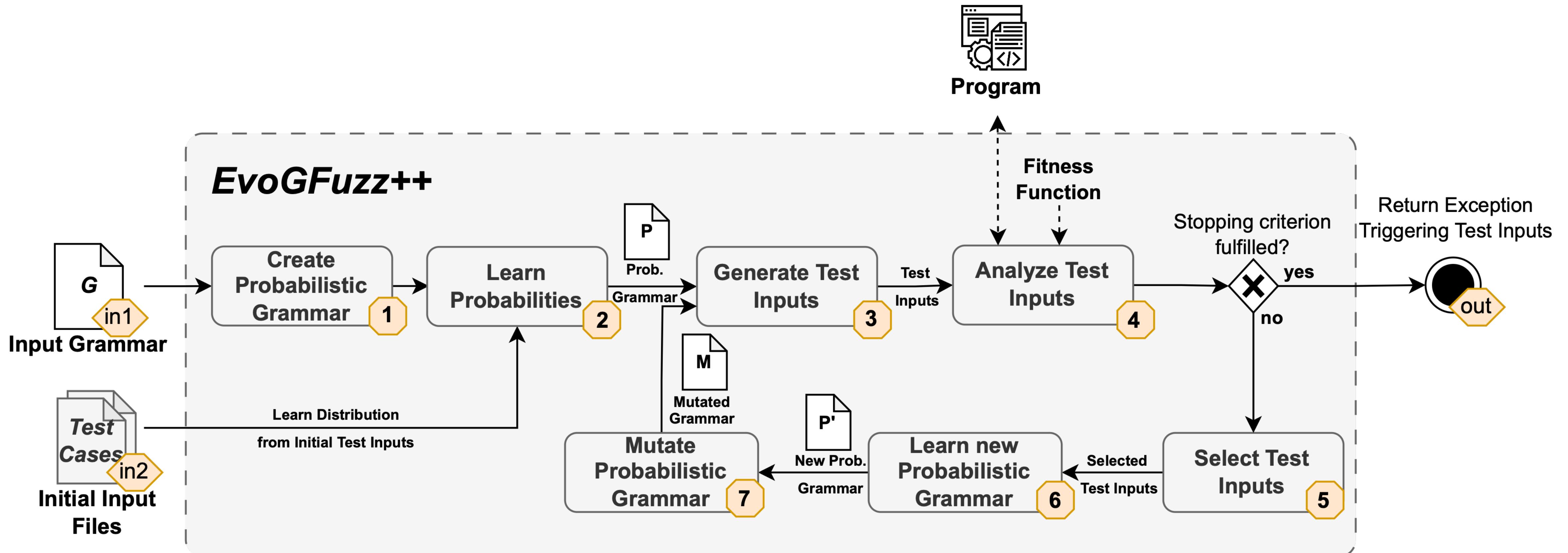
[Info]: For this chapter on probabilistic fuzzing, we use the functions provided by [The Fuzzingbook](#). For a more detailed description of the ProbabilisticGrammarFuzzer, have a look at the chapter [Probabilistic Grammar Fuzzing](#).

In the next section, we delve into "Probabilistic Grammar-Based Fuzzing". An essential aspect of this approach is harnessing the concept of probability distribution. To illustrate this, we consider the Law of Leading Digits, also known as Benford's Law.

#### Law of the leading digits

Benford's Law reveals a surprising phenomenon about leading digits in many sets of numerical data: smaller numbers tend to occur as the leading digits more frequently. Specifically, '1' appears as the leading digit about 30% of the time, while '9' appears just

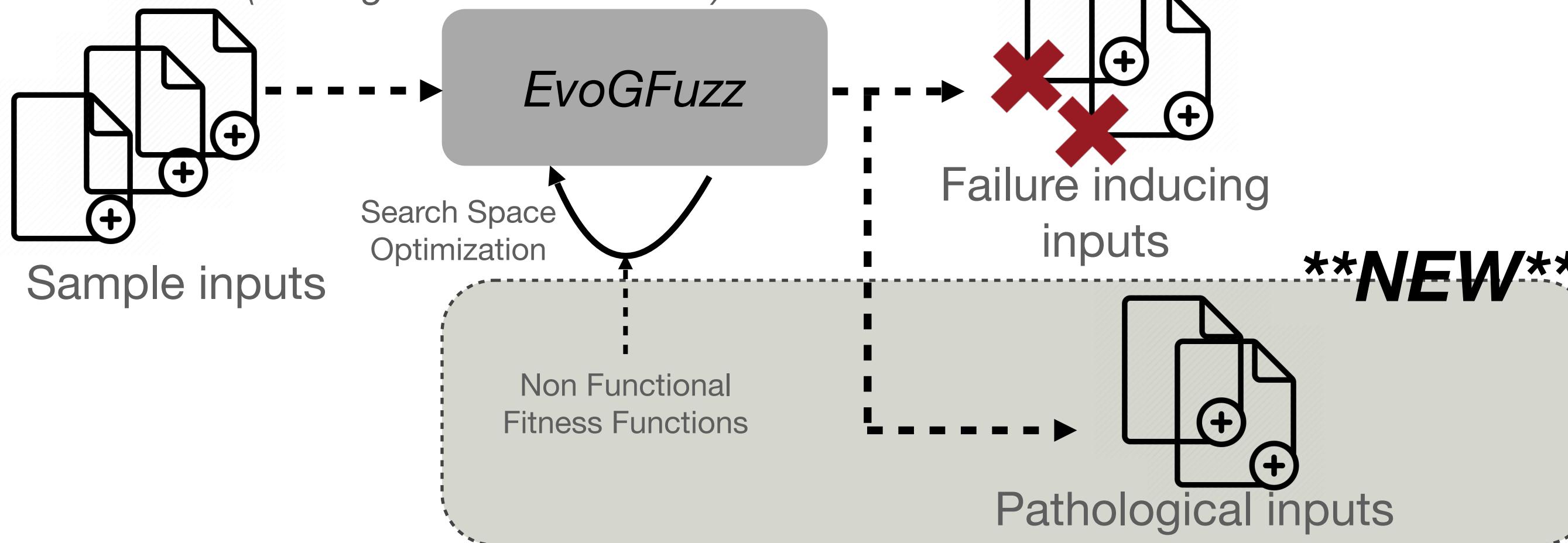
# Evolutionary Grammar-Based Fuzzing





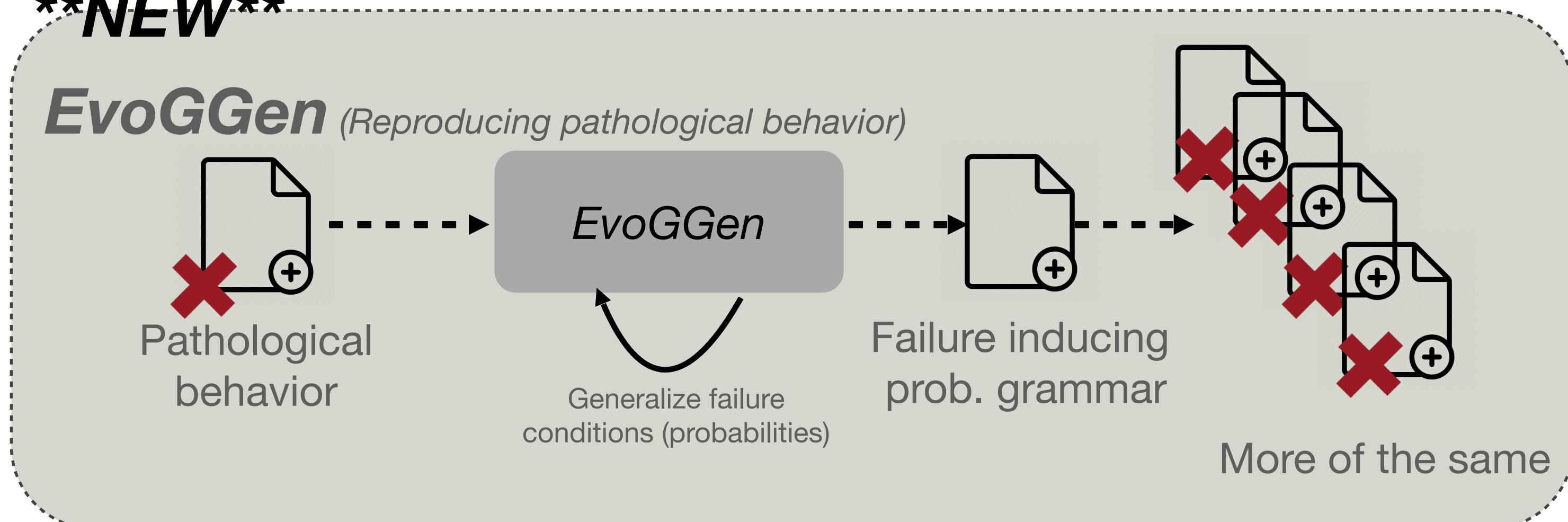
# Evolutionary Grammar-Based Fuzzing

**EvoGFuzz** (*Finding new vulnerabilities*)



**\*\*NEW\*\***

**EvoGGen** (*Reproducing pathological behavior*)



A screenshot of a GitHub repository page for "EvoGFuzz". The repository has 57.8% Python code and 42.2% Jupyter Notebook code. The README.md file contains the following text:

**EvoGFuzz**

This repo contains the code to execute, develop and test our grammar-based fuzzing tool **EvoGFuzz**, which was first described in our paper *Evolutionary Grammar-Based Fuzzing* accepted at [SSBSE'2020](#).

**Example**

To illustrate **EvoGFuzz**'s capabilities, we start with a quick motivating example. First, let us introduce our program under test: **The Calculator**.

```

import math

def arith_eval(inp) -> float:
    return eval(
        str(inp), {"sqrt": math.sqrt, "sin": math.sin, "cos": math.cos}
    )
    
```

This infamous program accepts arithmetic equations, trigonometric functions and allows us to calculate the square root. To help us determine faulty behavior, i.e., a crash, we implement an evaluation function

```

def prop(inp: str) -> bool:
    try:
        arith_eval(inp)
    except:
        return True
    return False
    
```

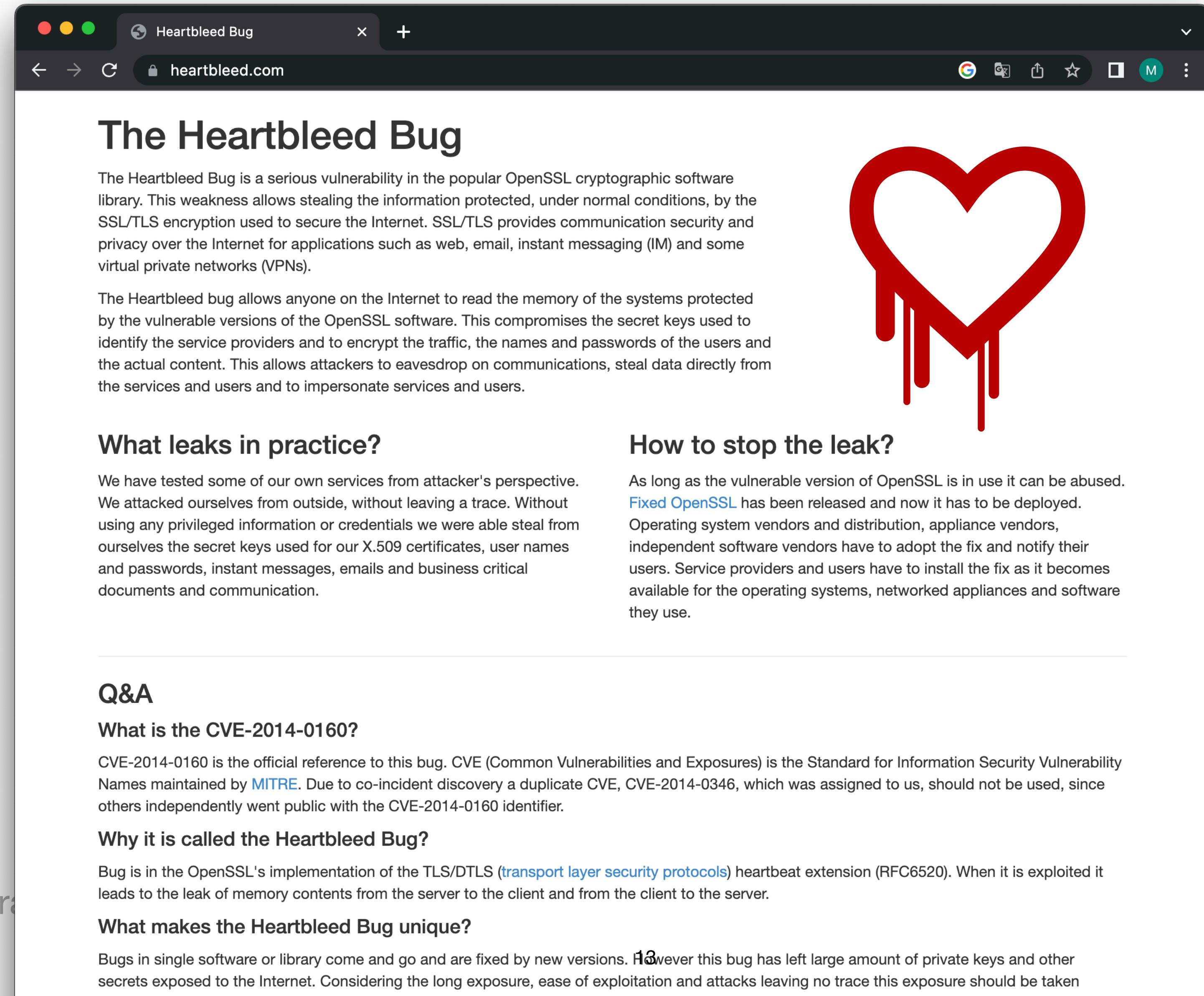
# Explaining Pathological Program Behavior

The collage consists of three screenshots:

- Semantic Debugging Paper:** A screenshot of a PDF document titled "Semantic Debugging..." by Martin Eberlein, Lars Grunske, Dominic Steinköfel, and Andreas Zeller. The paper discusses a technique for automatically determining failure causes and conditions using logical properties over input elements. It includes abstract, introduction, index terms, and references sections.
- GitHub Repository (alhazen-py):** A screenshot of a GitHub repository page for "alhazen-py". The repository contains code for a Python implementation of Alhazen, a tool for explaining failing programs. It includes setup files, a README, and a test script. The repository has 2 weeks ago activity.
- GitHub Repository (avicenna):** A screenshot of a GitHub repository page for "avicenna". This repository contains the code for the Avicenna prototype. It includes setup files, a README, and a test script. The repository has 2 weeks ago activity.



# Heartbeat Protocol

A screenshot of a web browser window. The title bar says "Heartbleed Bug" and the address bar shows "heartbleed.com". The main content area features a large red heart outline with red liquid dripping down from the bottom, symbolizing the bug. The text discusses the Heartbleed Bug, what it leaks in practice, how to stop it, and provides Q&A sections about CVE-2014-0160 and the bug's name.

**The Heartbleed Bug**

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

**What leaks in practice?**

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

**How to stop the leak?**

As long as the vulnerable version of OpenSSL is in use it can be abused. [Fixed OpenSSL](#) has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

---

**Q&A**

**What is the CVE-2014-0160?**

CVE-2014-0160 is the official reference to this bug. CVE (Common Vulnerabilities and Exposures) is the Standard for Information Security Vulnerability Names maintained by [MITRE](#). Due to co-incident discovery a duplicate CVE, CVE-2014-0346, which was assigned to us, should not be used, since others independently went public with the CVE-2014-0160 identifier.

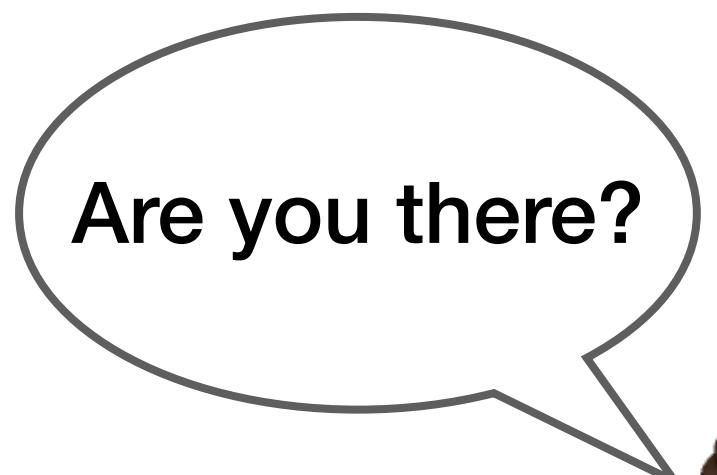
**Why it is called the Heartbleed Bug?**

Bug is in the OpenSSL's implementation of the TLS/DTLS ([transport layer security protocols](#)) heartbeat extension (RFC6520). When it is exploited it leads to the leak of memory contents from the server to the client and from the client to the server.

**What makes the Heartbleed Bug unique?**

Bugs in single software or library come and go and are fixed by new versions. However this bug has left large amount of private keys and other secrets exposed to the Internet. Considering the long exposure, ease of exploitation and attacks leaving no trace this exposure should be taken

# Heartbeat Protocol



Client

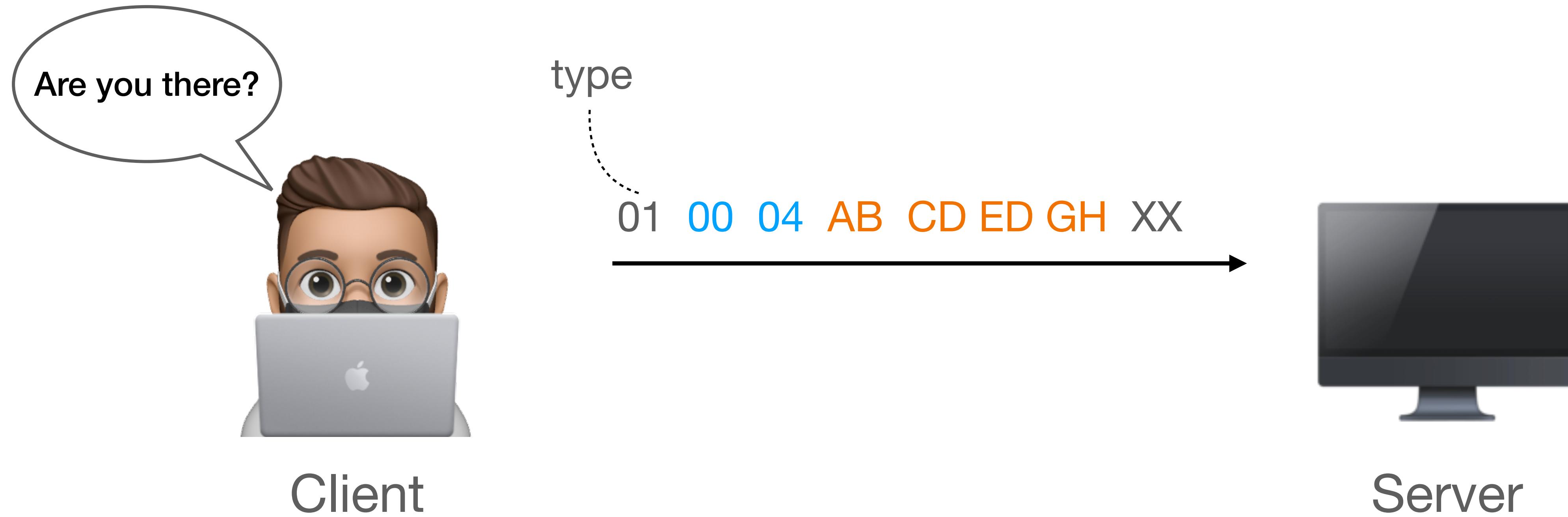


Server

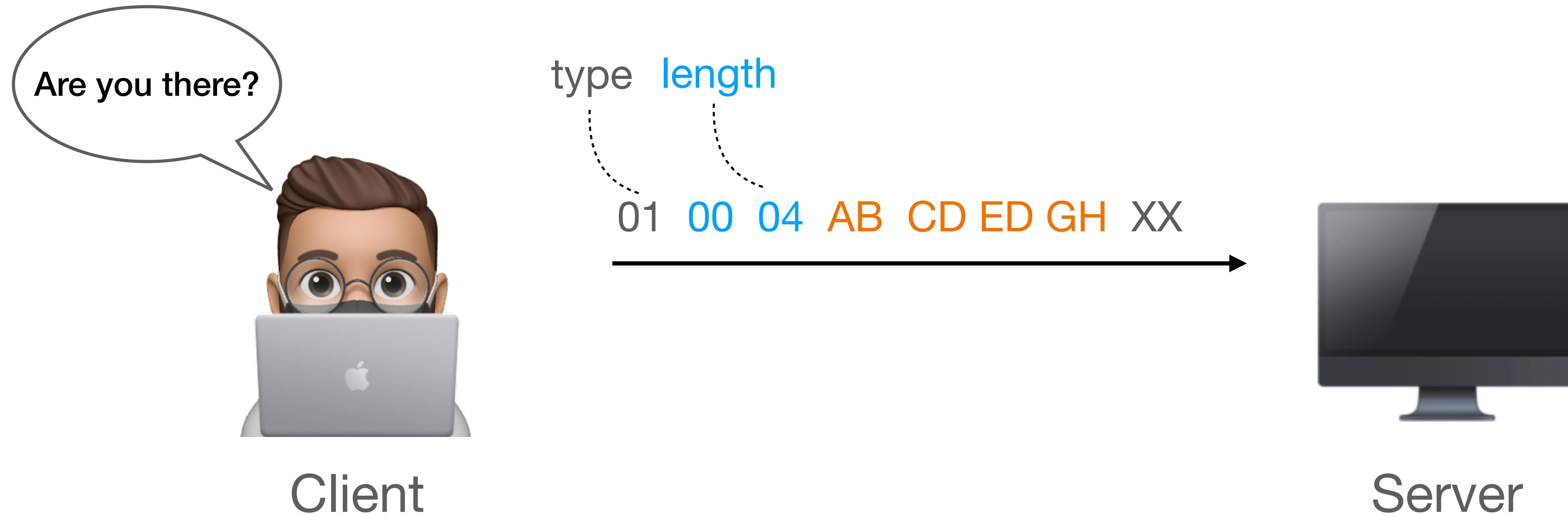
# Heartbeat Protocol



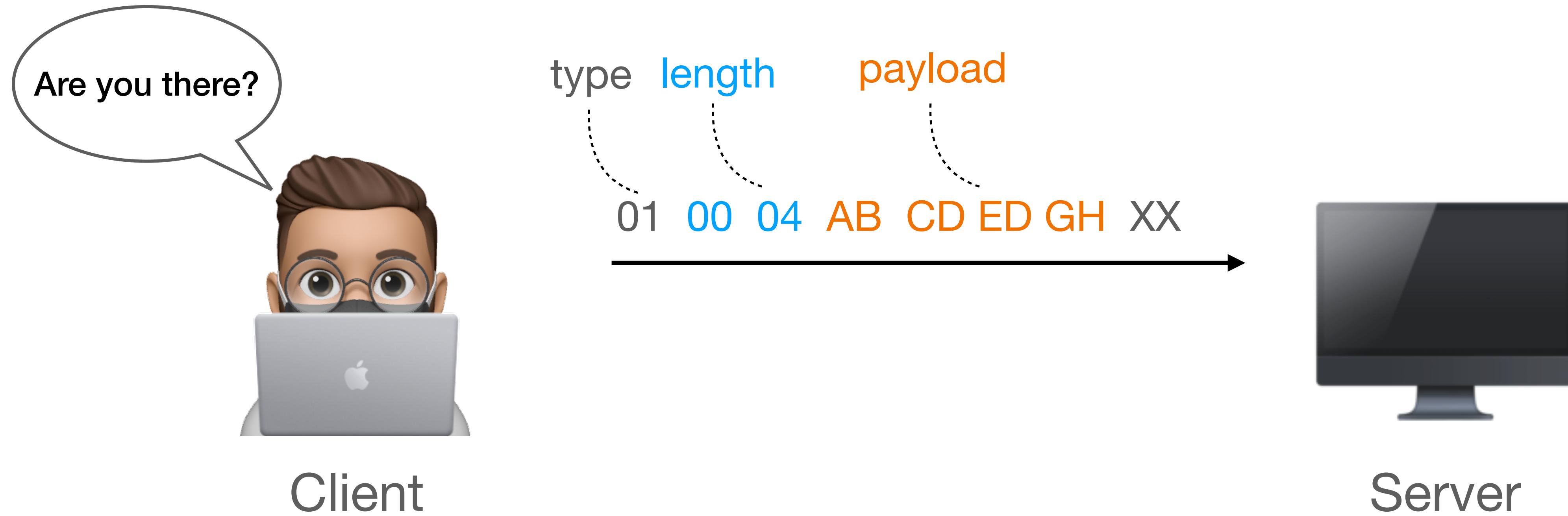
# Heartbeat Protocol



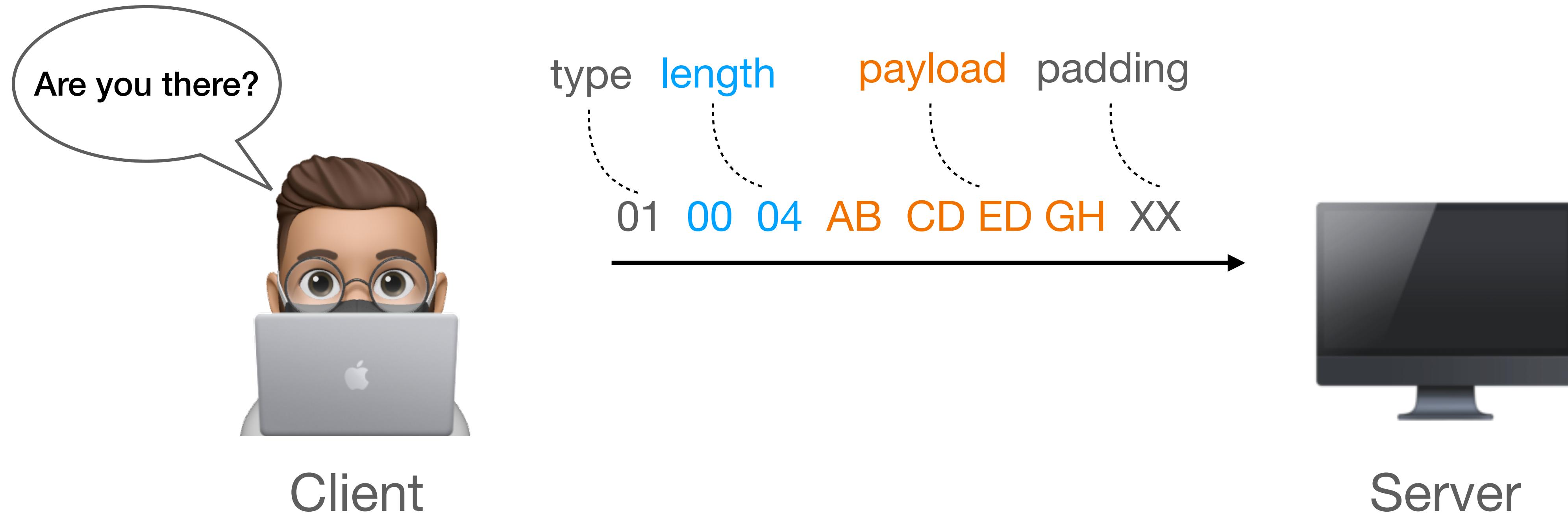
# Heartbeat Protocol



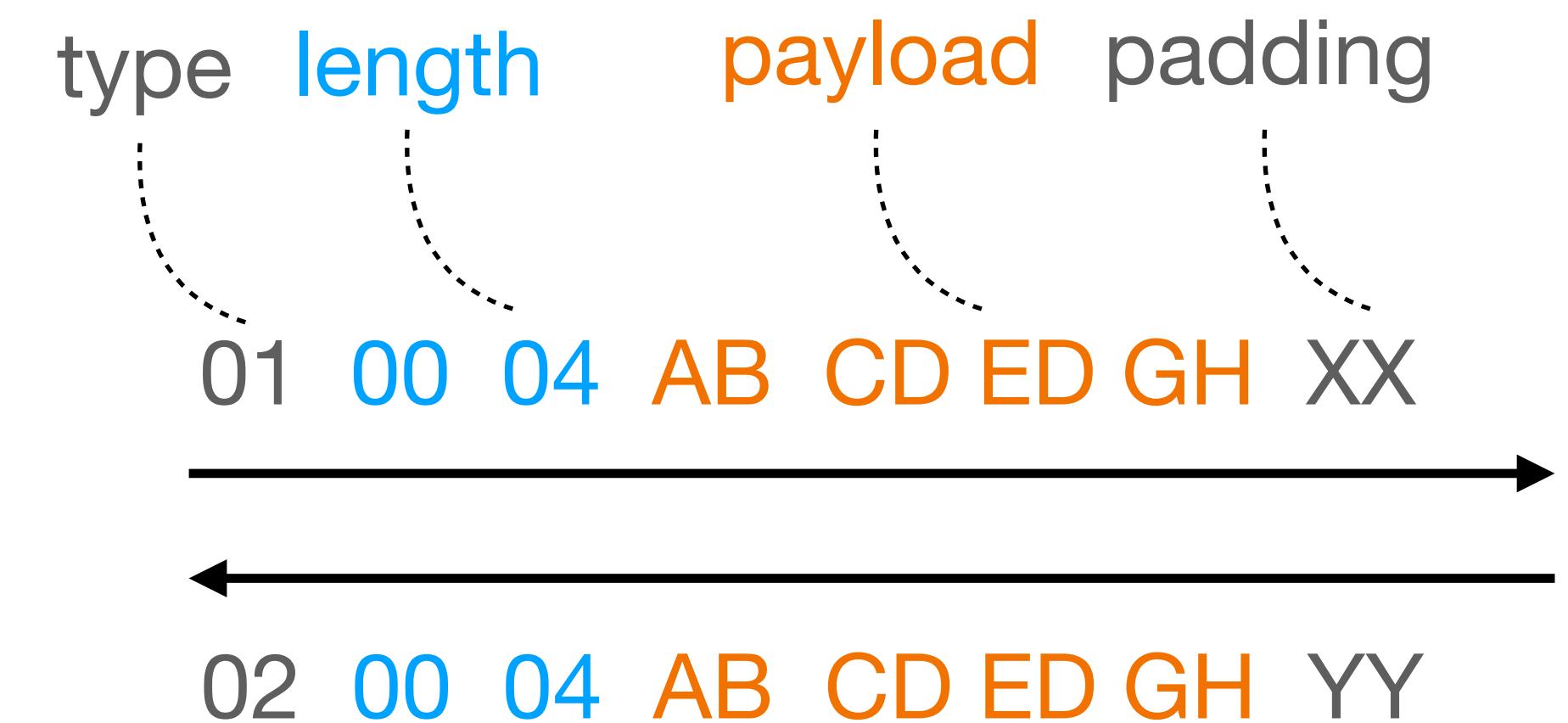
# Heartbeat Protocol



# Heartbeat Protocol



# Heartbeat Protocol

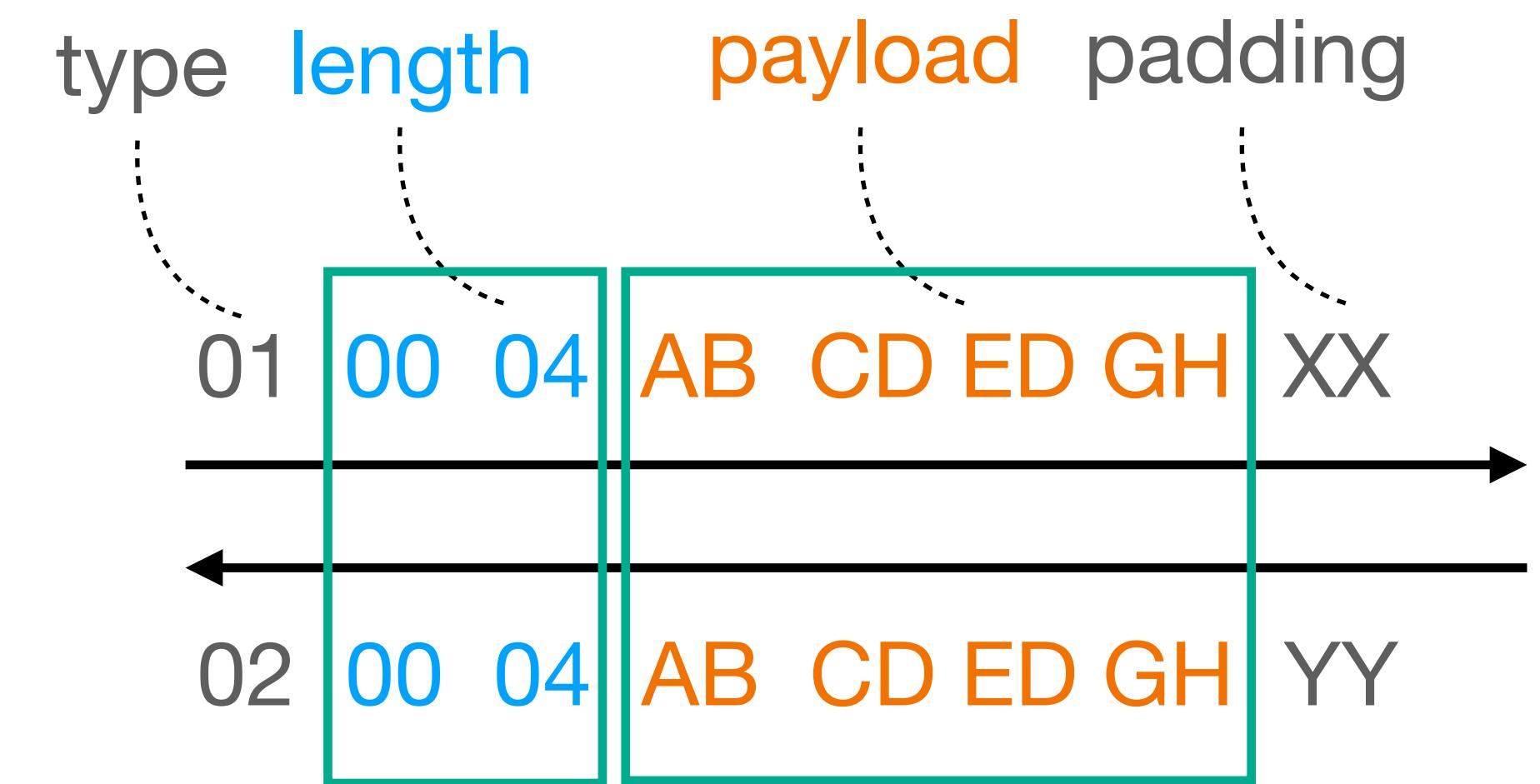


Client



Server

# Heartbeat Protocol



Client



Server

# Heartbeat Protocol

## Exploit

Are you there?



**Attacker**



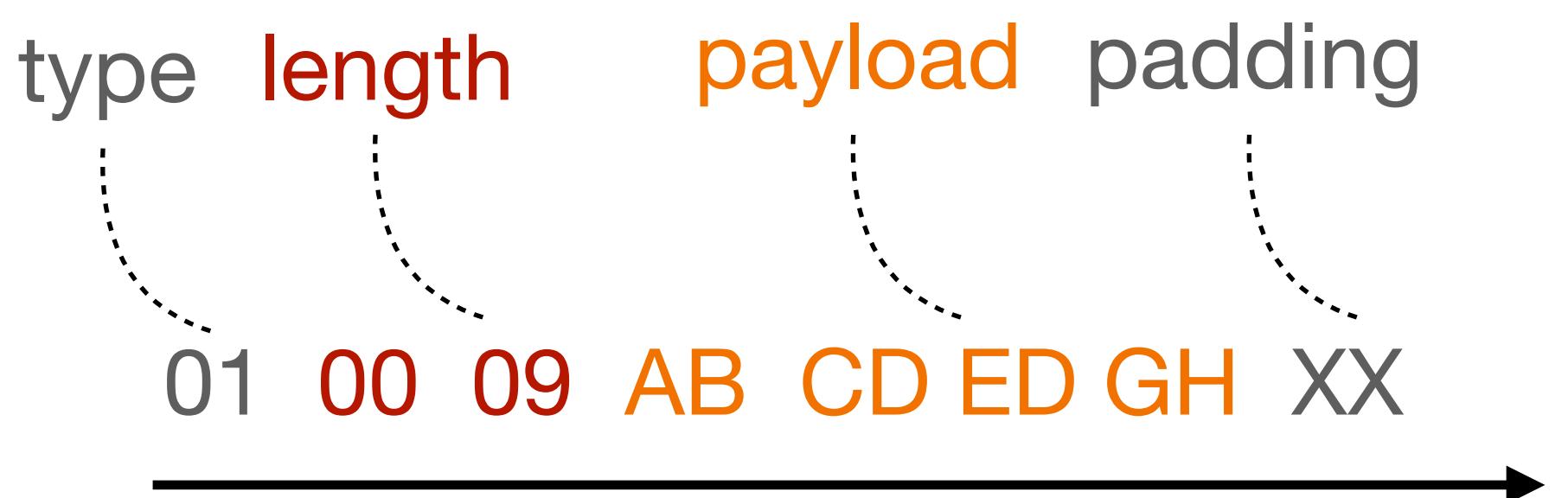
**Server**

# Heartbeat Protocol

## Exploit



**Attacker**



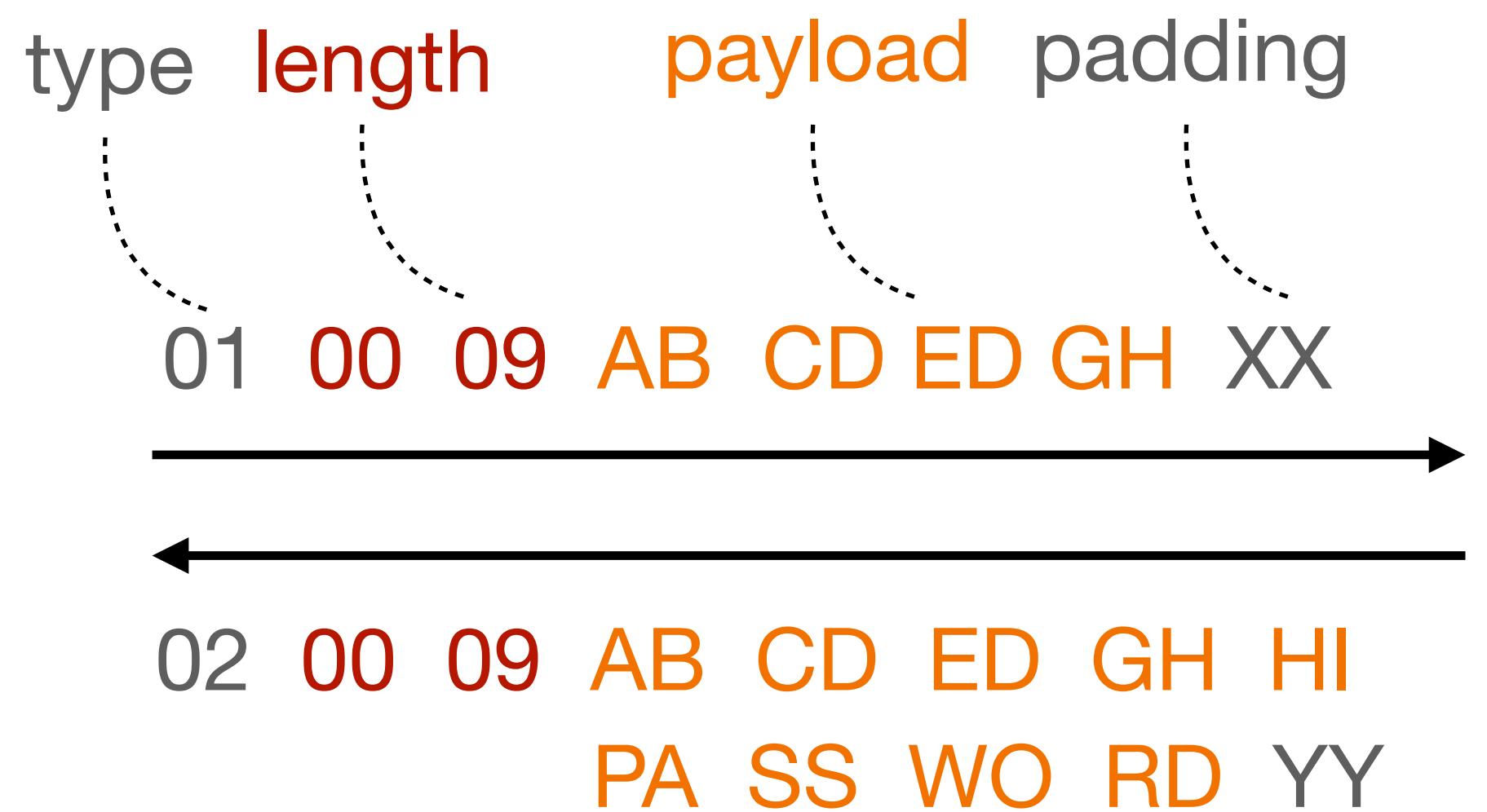
**Server**

# Heartbeat Protocol

## Exploit



Attacker



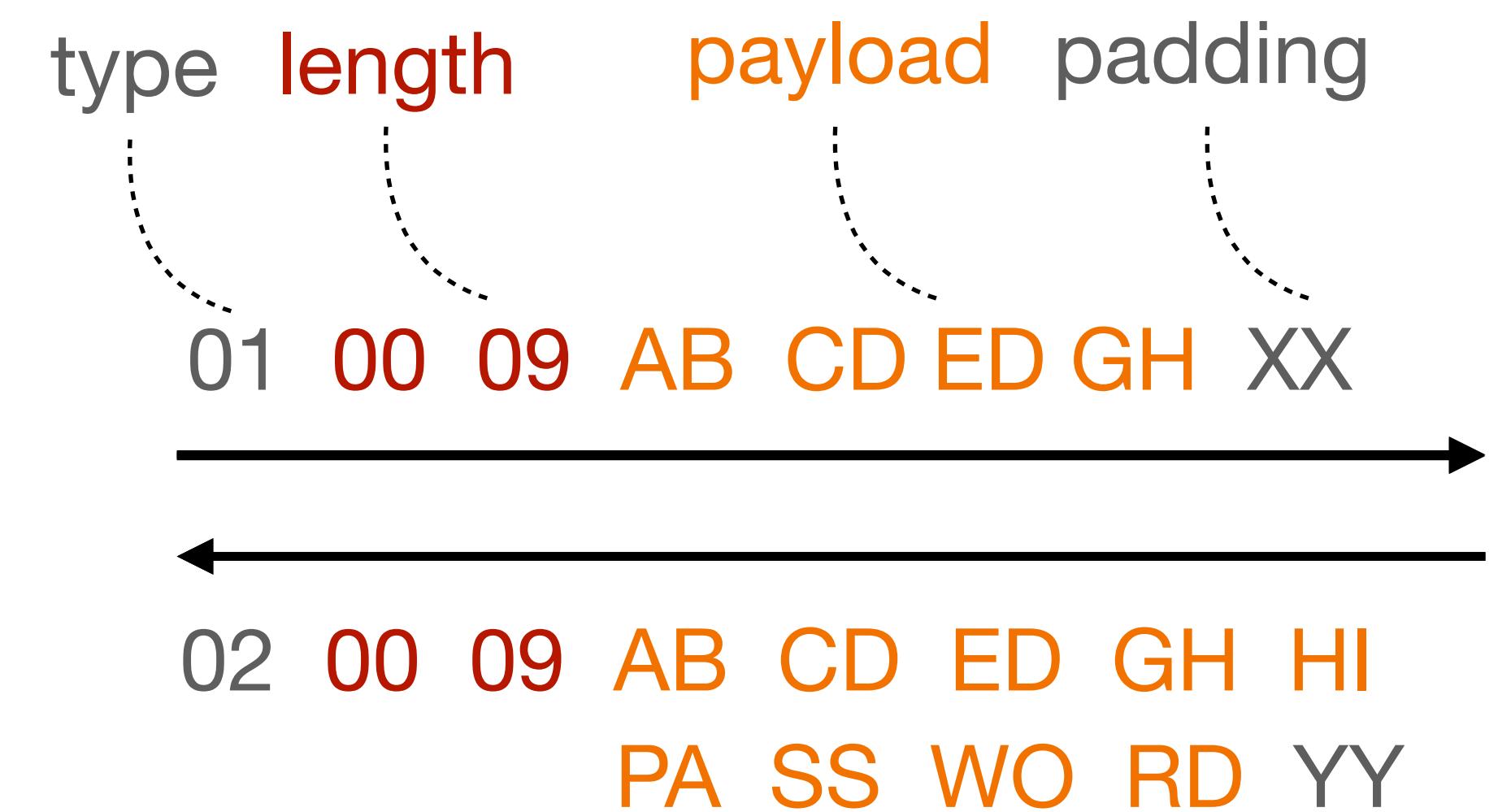
Server

# Heartbeat Protocol

## Exploit



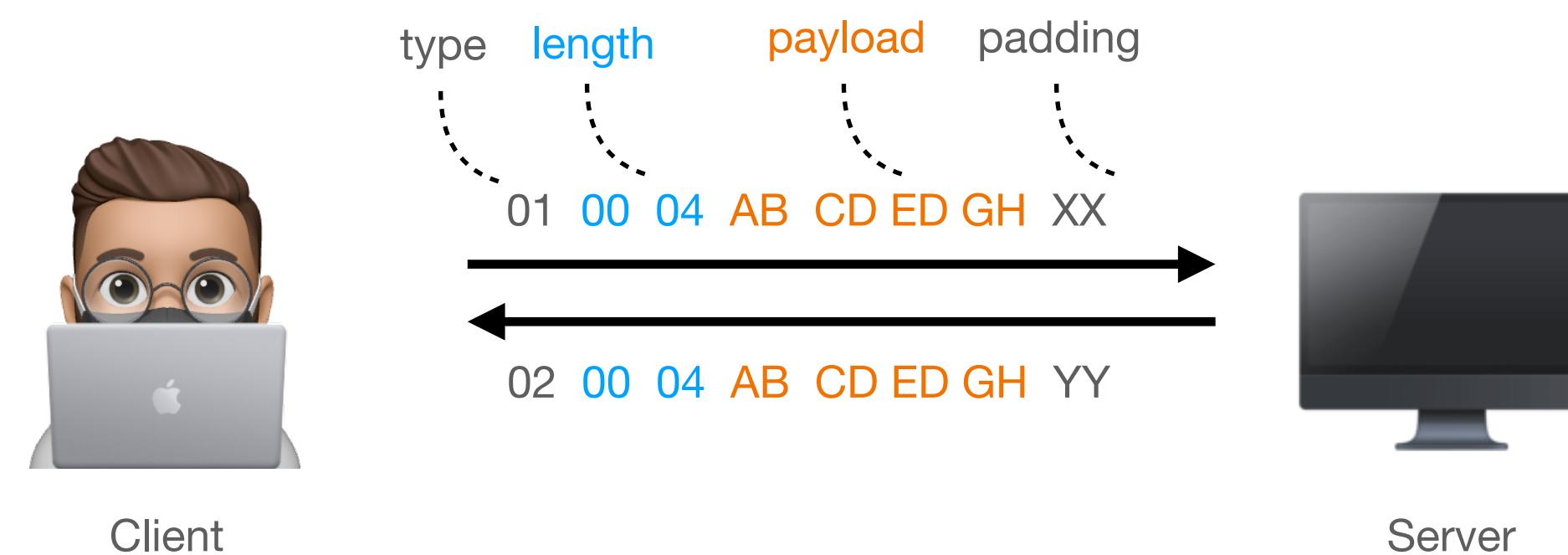
Attacker



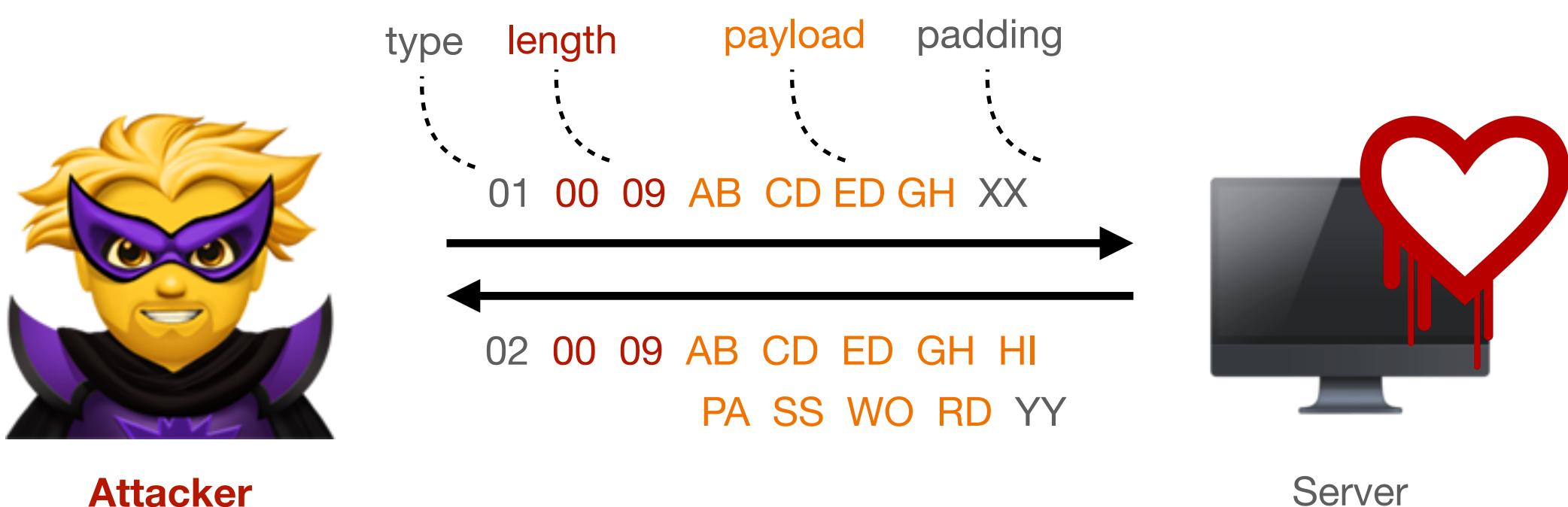
Server

The *heartbleed bug* occurs if  
 $\text{int}(\text{length}) > \text{length}(\text{payload})$

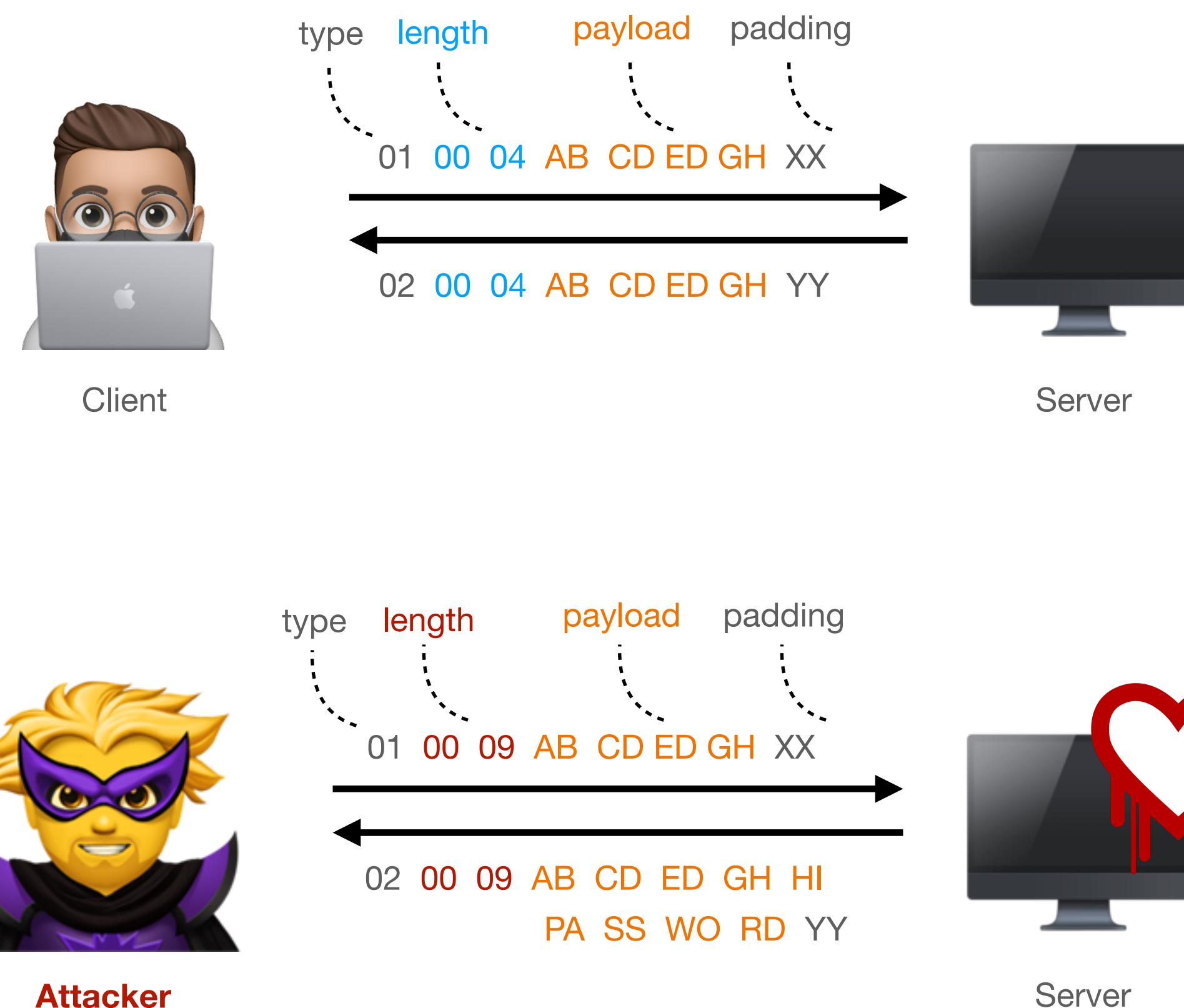
# Explaining Program Behavior



Can we **automatically** generate a **theory** explaining why a given input triggers the bug?



# Explaining Program Behavior



Can we **automatically** generate a **theory** explaining why a given input triggers the bug?

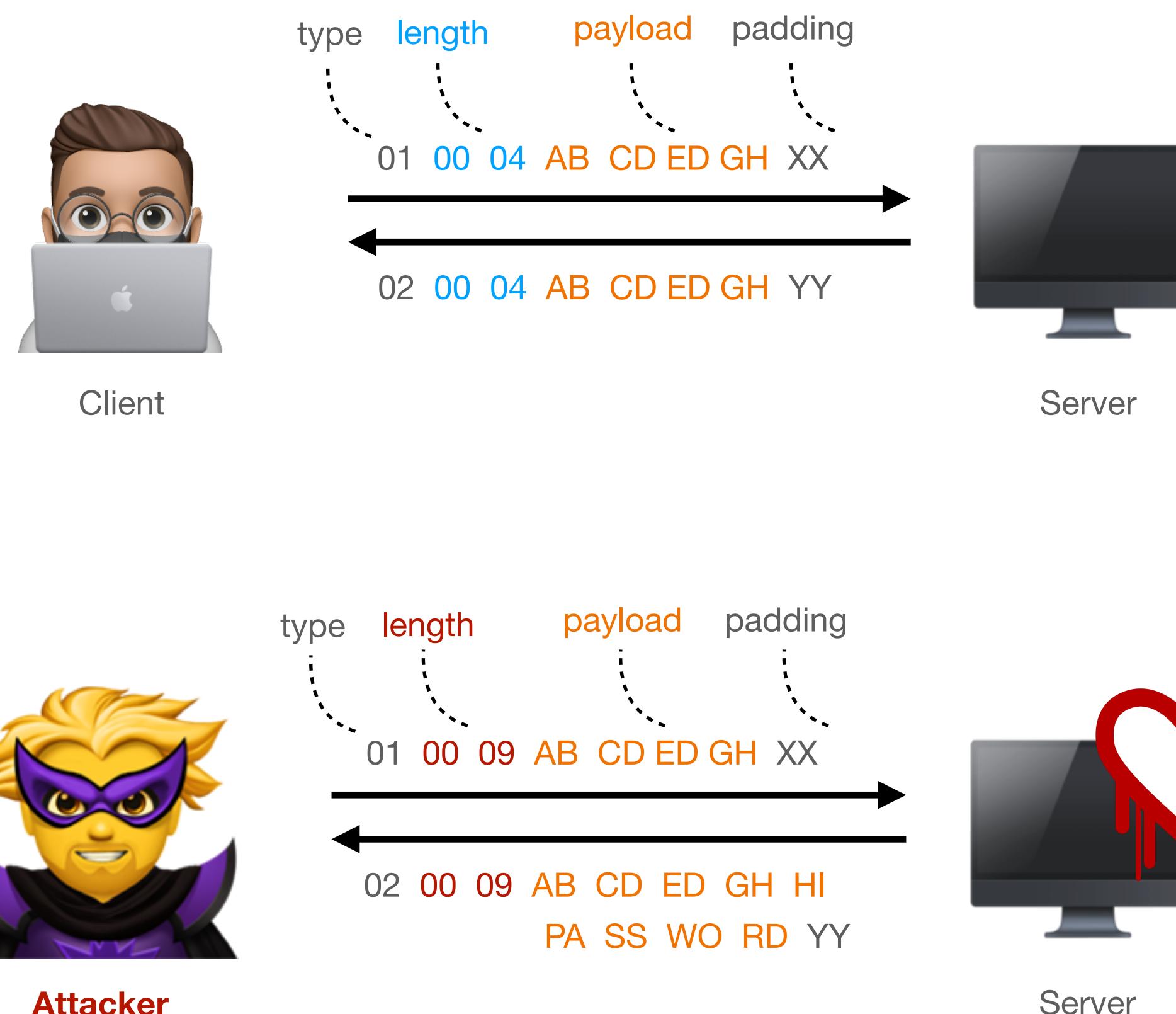
Possible Theories:

$\text{payload} \geq 9$

$\text{length} > 7$

$\text{length} > \text{payload}$

# Explaining Program Behavior



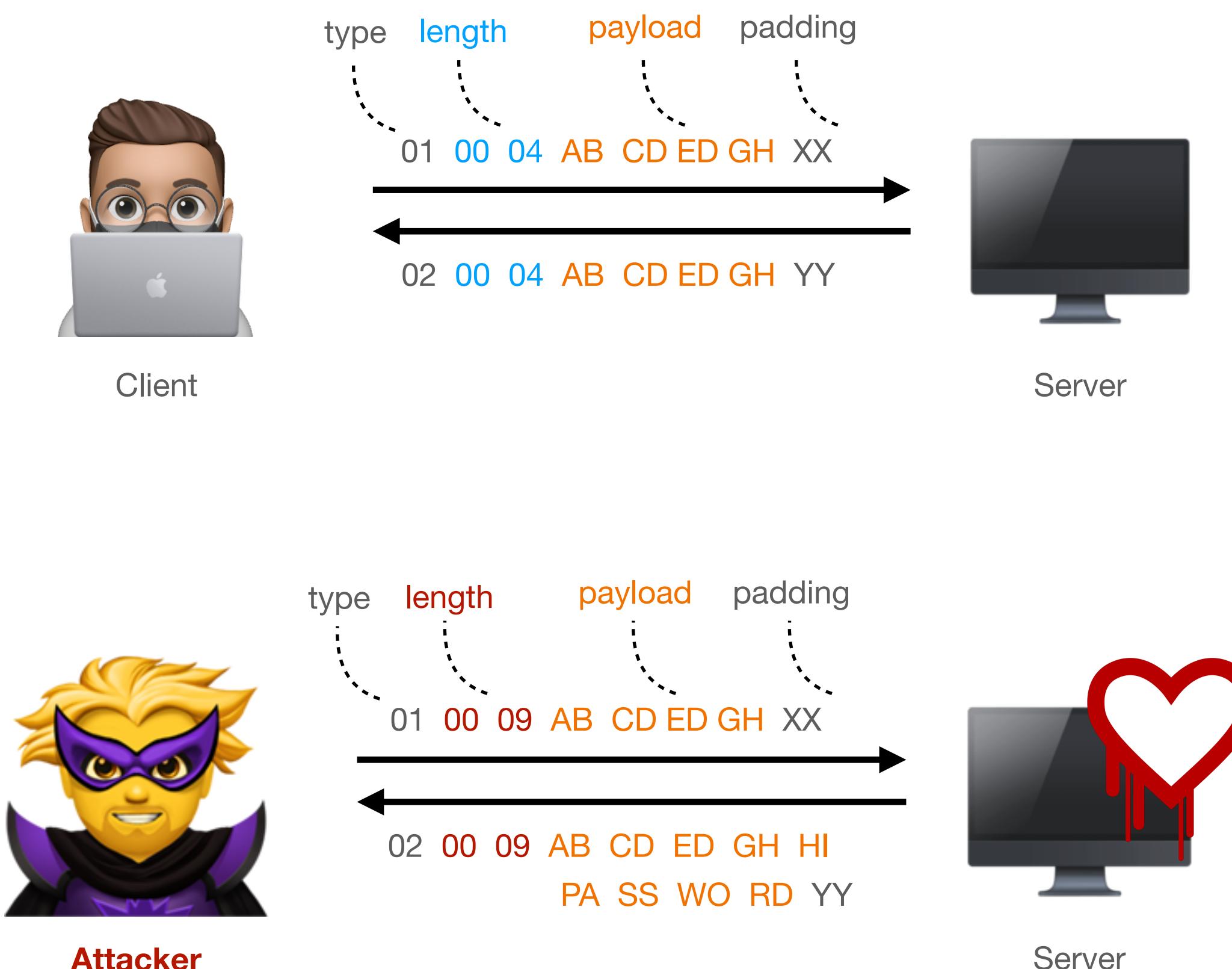
Can we **automatically** generate a **theory** explaining why a given input triggers the bug?

Observe  
Interactions

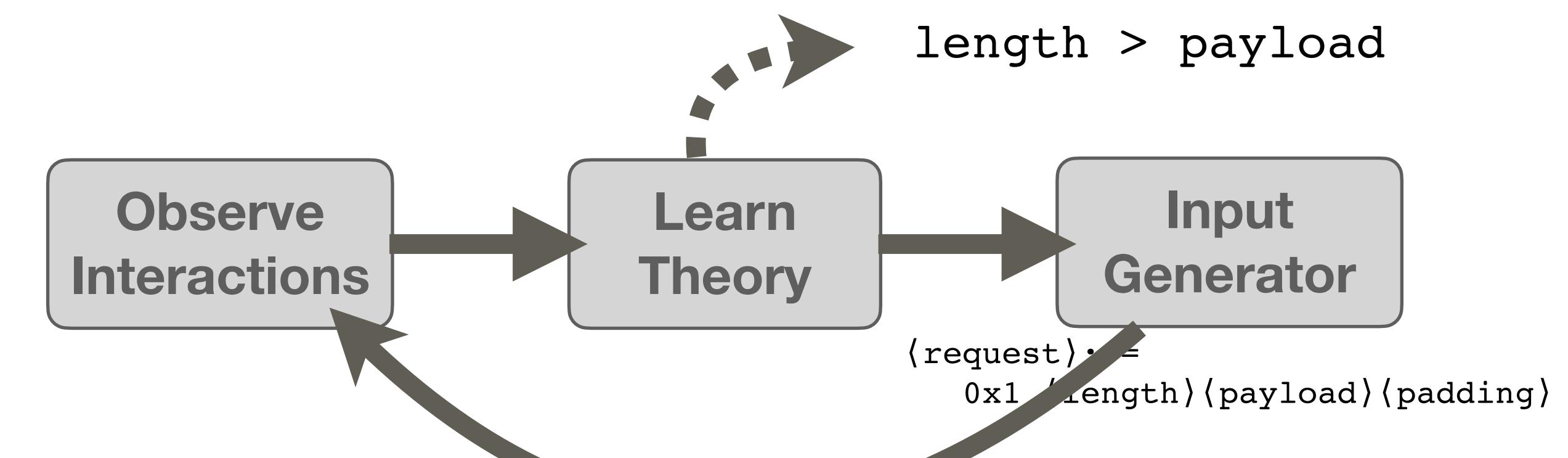
Input  
Generator

```
(request)::=
 0x1 {length}{payload}{padding}
```

# Explaining Program Behavior



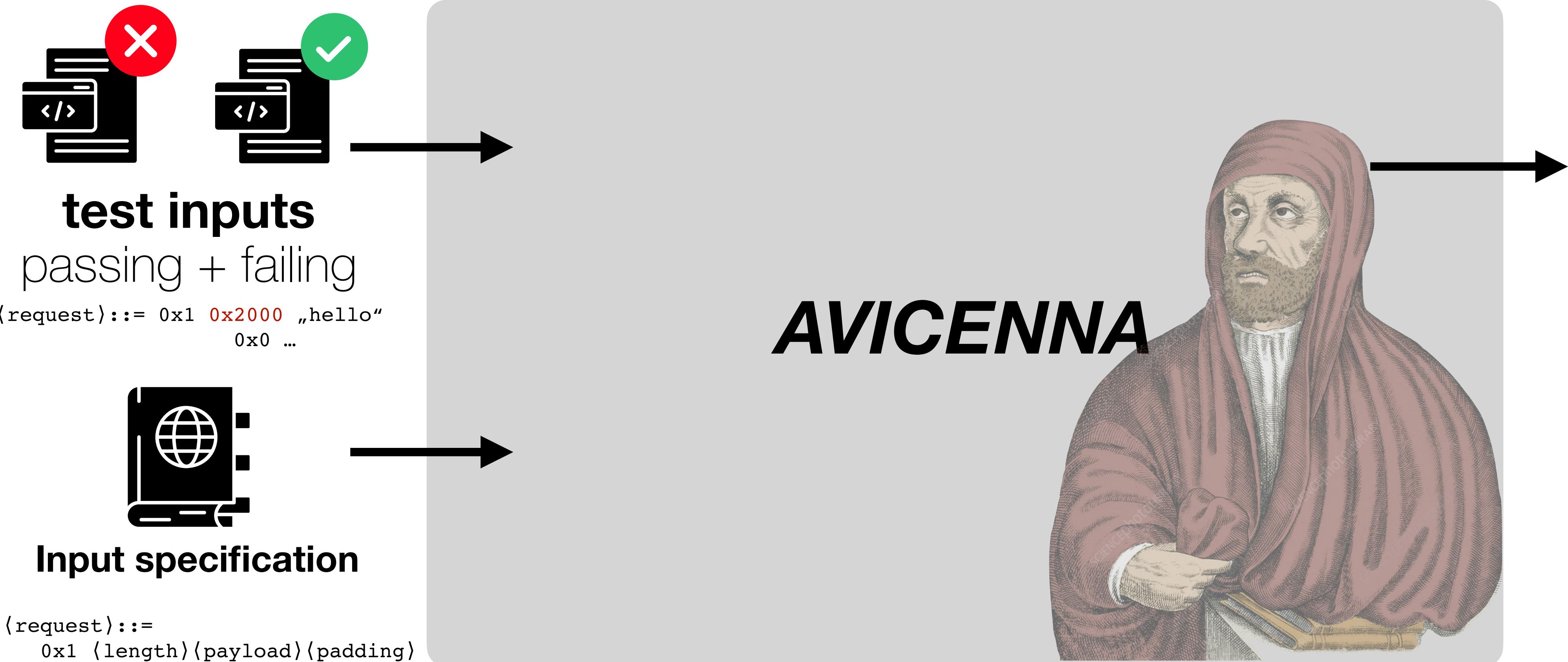
Can we **automatically** generate a **theory** explaining why a given input triggers the bug?





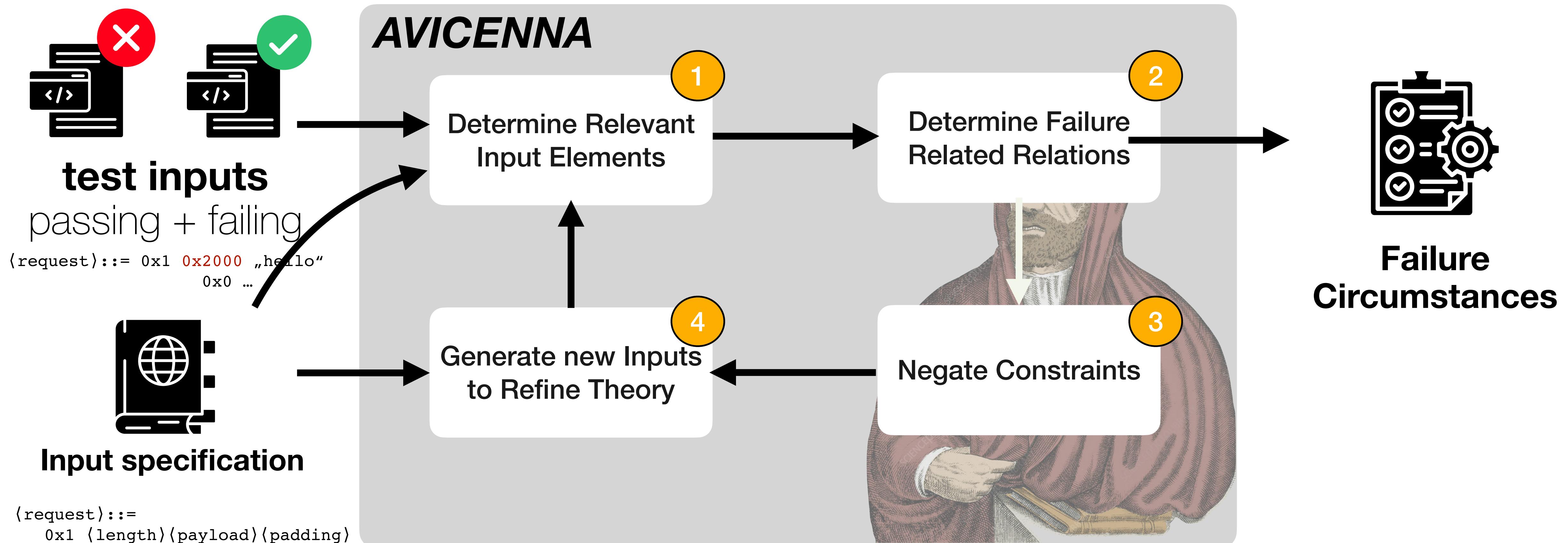
# Semantic Debugging

## Learning Failure Circumstances



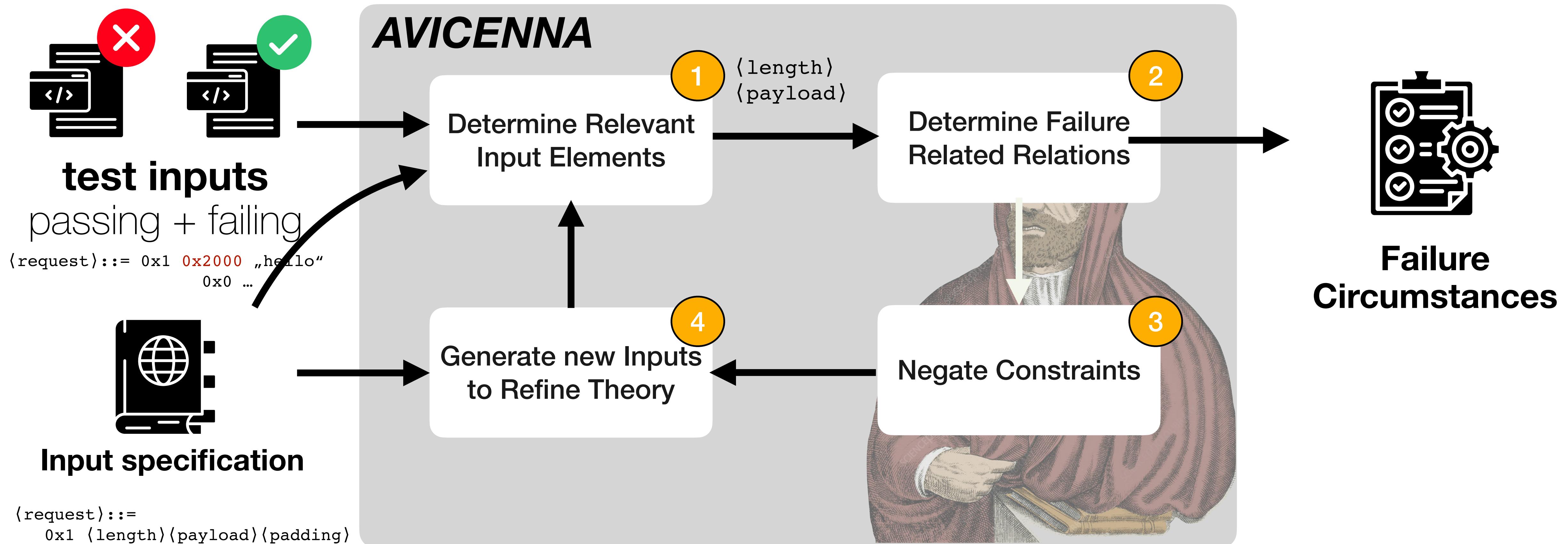
# Semantic Debugging

## Learning Failure Circumstances



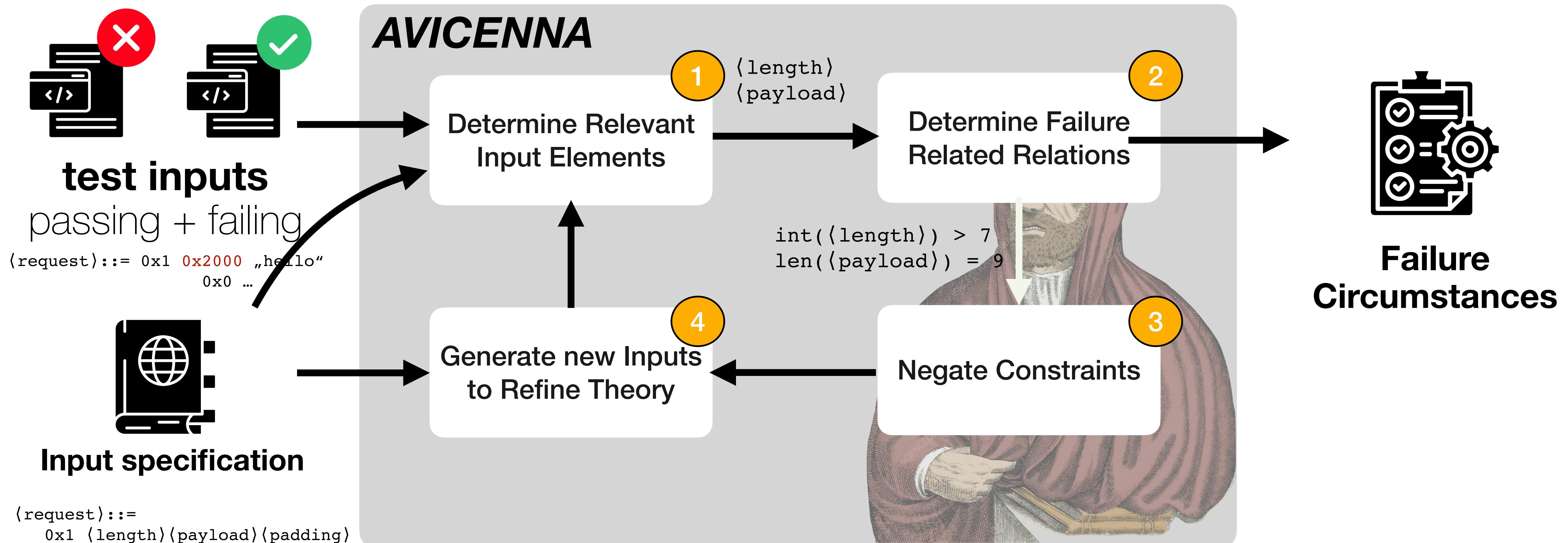
# Semantic Debugging

## Learning Failure Circumstances



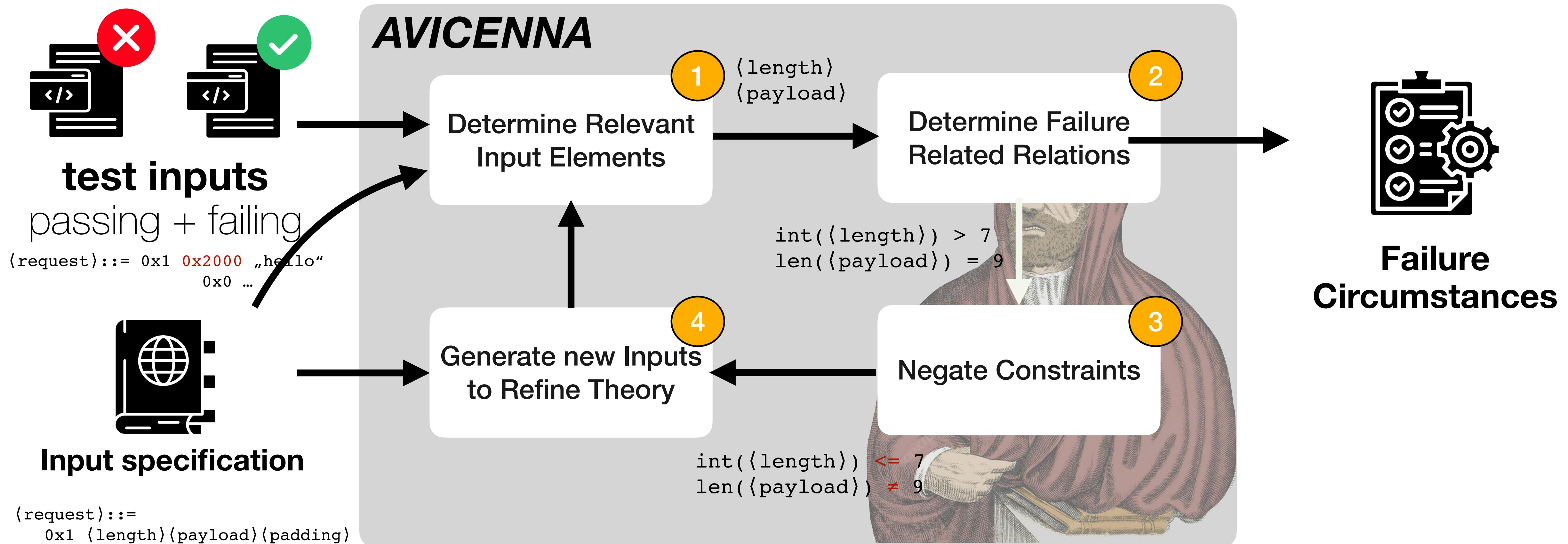
# Semantic Debugging

## Learning Failure Circumstances



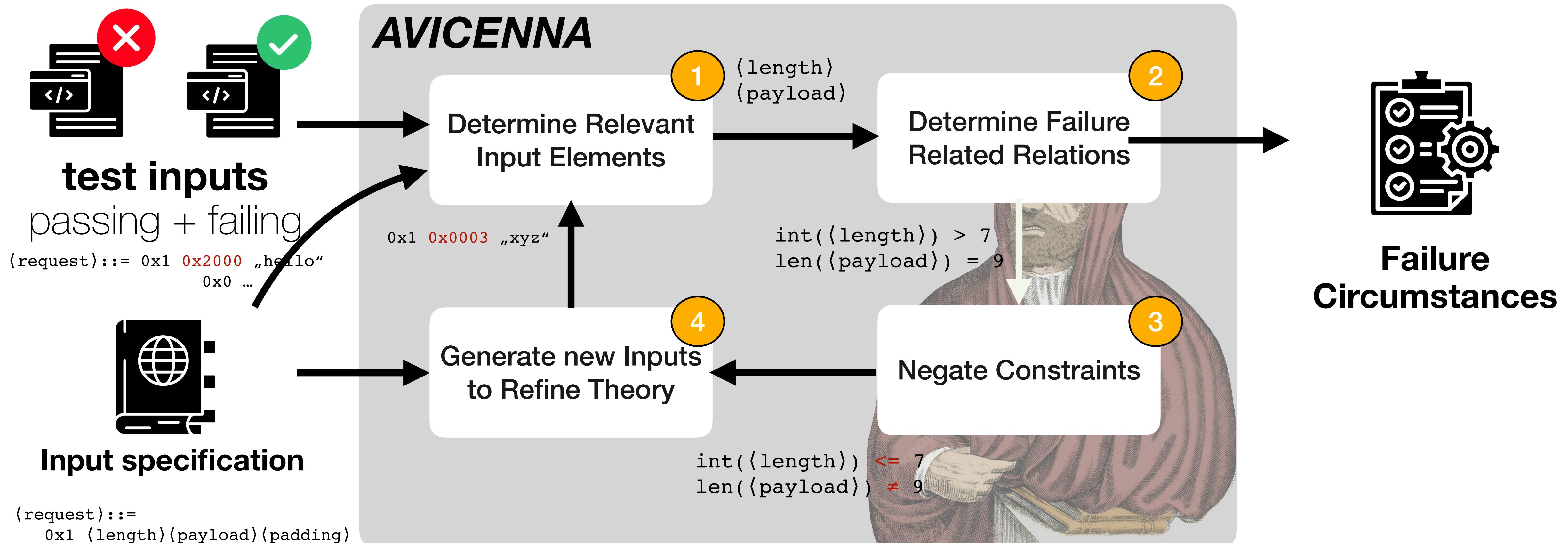
# Semantic Debugging

## Learning Failure Circumstances



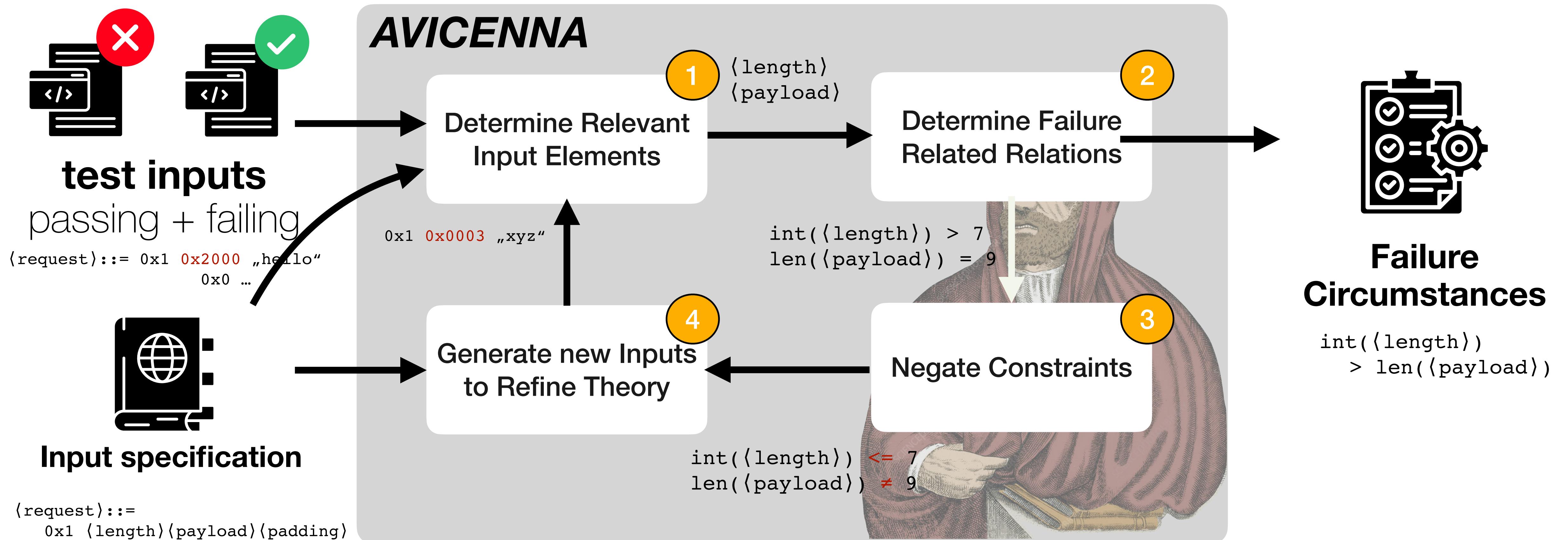
# Semantic Debugging

## Learning Failure Circumstances



# Semantic Debugging

## Learning Failure Circumstances

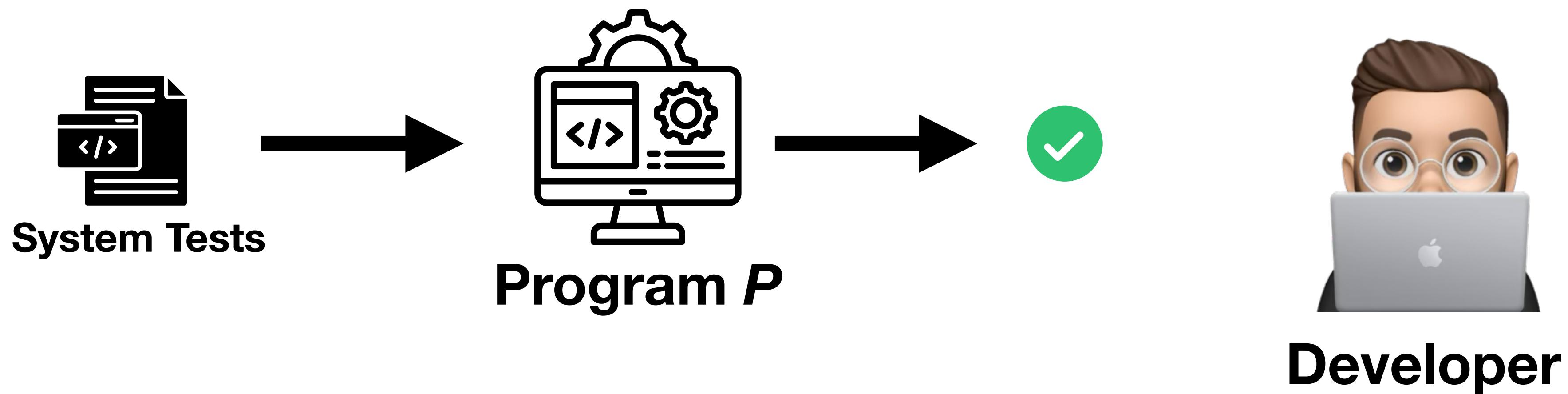


# Which Inputs Trigger my Patch?

Generating explanations to describe the input properties affected by a patch

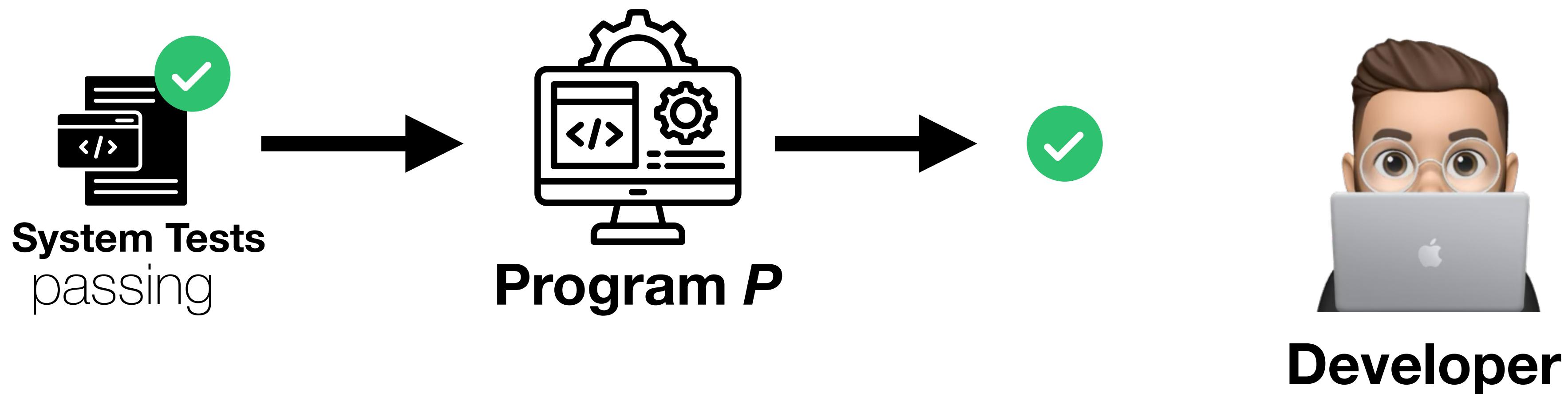
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



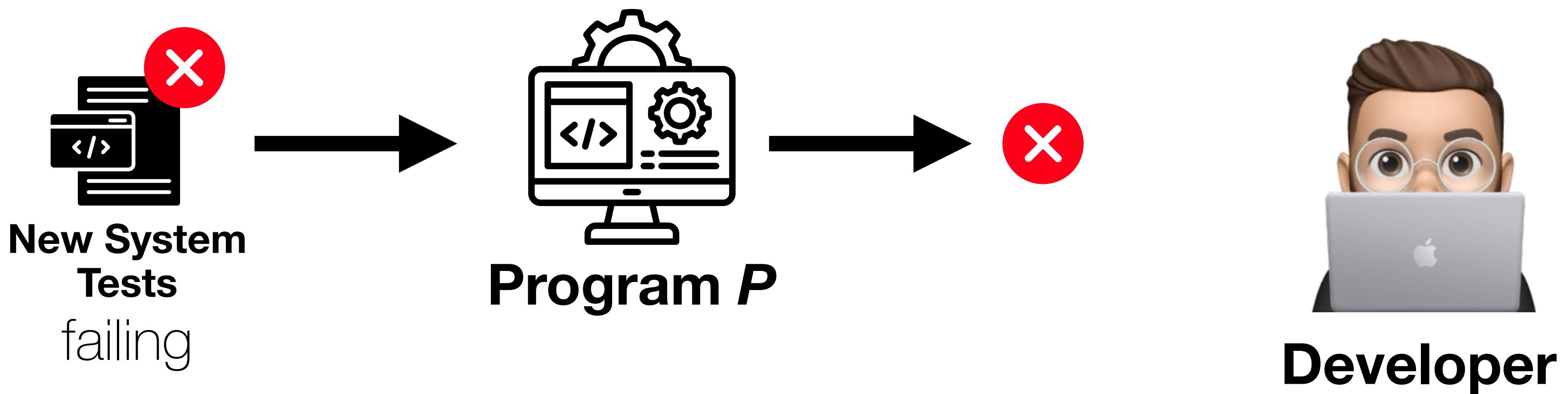
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



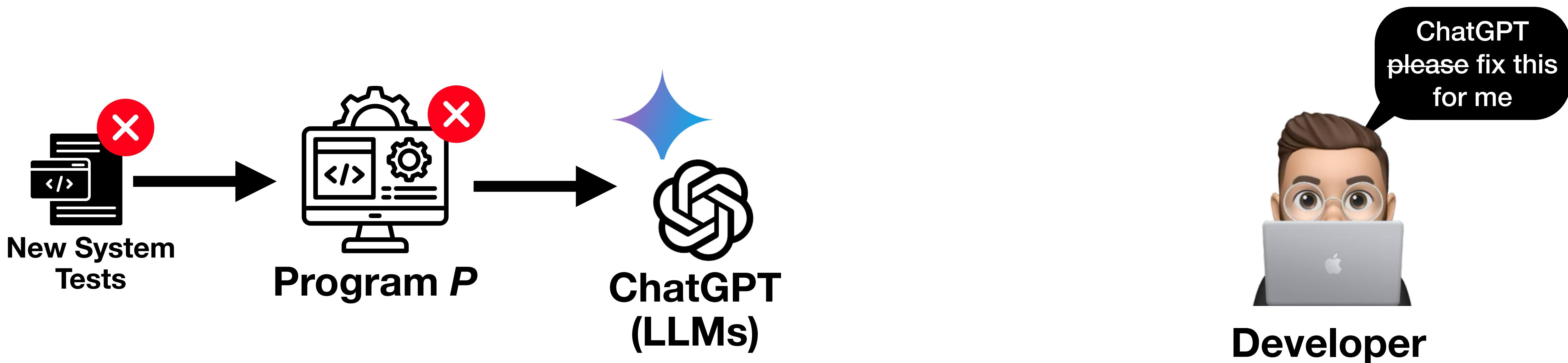
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



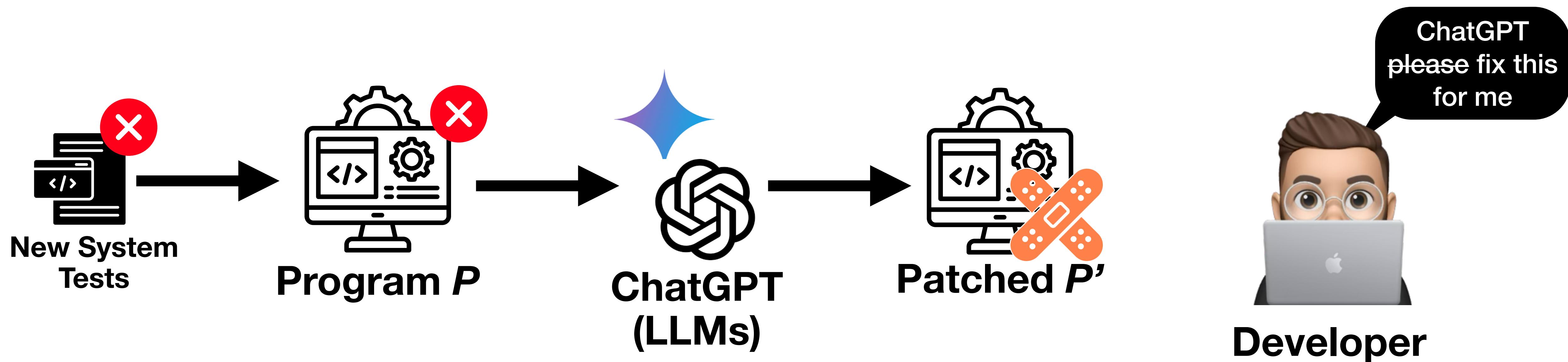
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



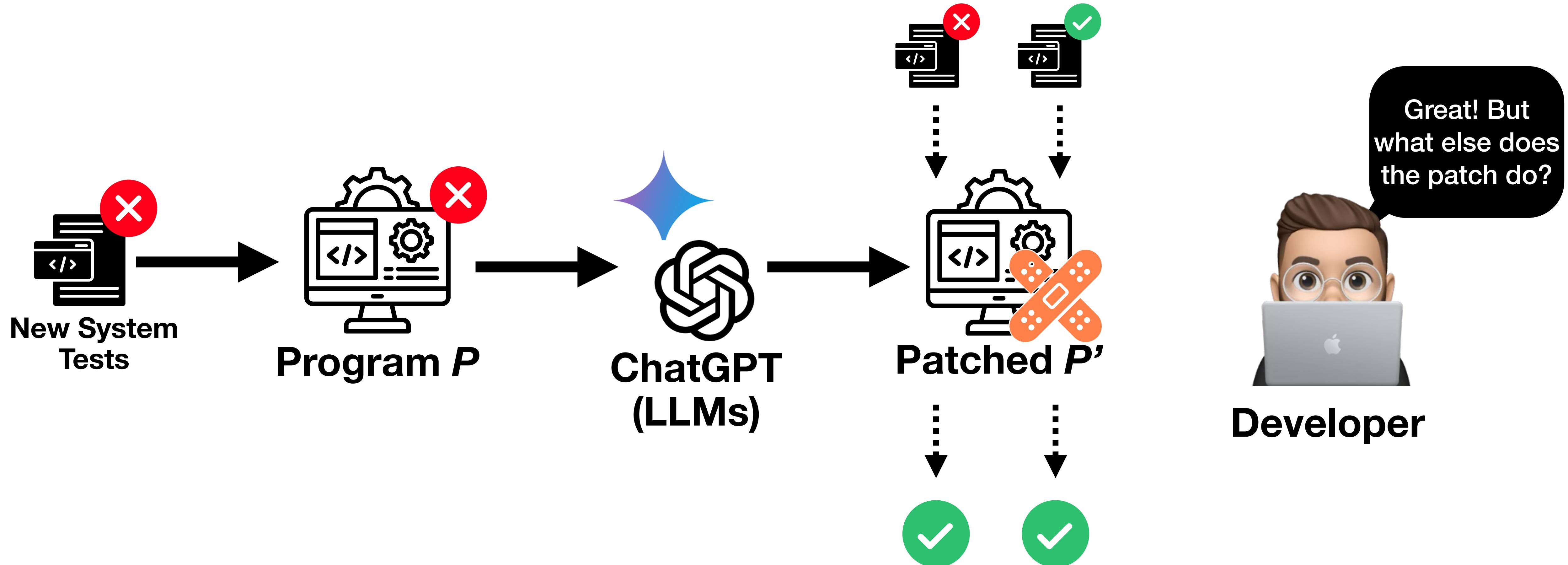
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



# Trust Issues?

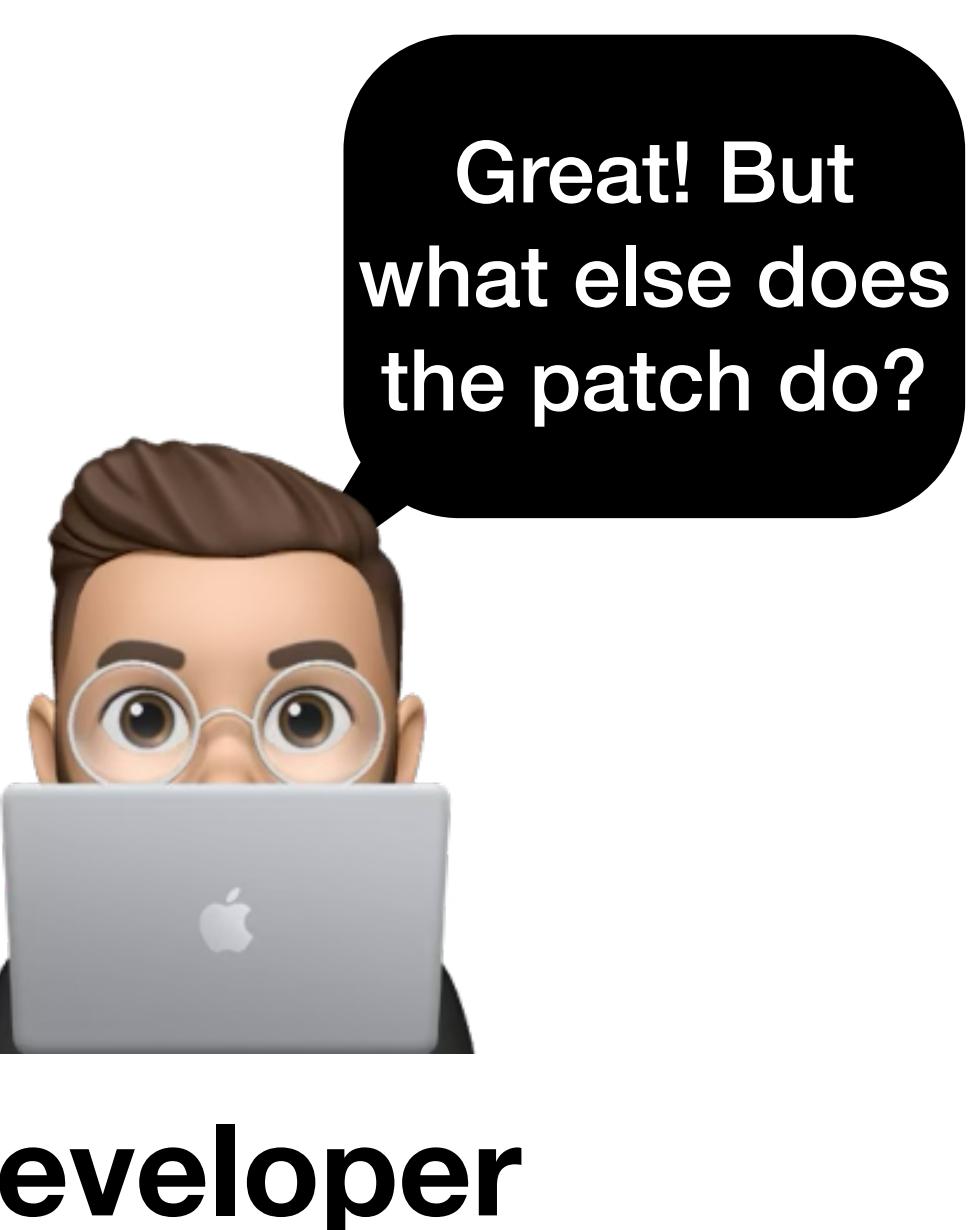
Would you trust a patch generated by ChatGPT to fix your critical code?



# Trust Issues?

## The Core Issue

- APR tools—including those powered by LLMs—are powerful but often opaque.
- It's unclear exactly which inputs are affected by these automatically generated patches.
- This lack of clarity creates significant trust issues for developers.



**Developer**

# Trust Issues?

## Why is this crucial?

- Developers must be certain about patch boundaries to ensure reliability.
- Blind trust in APR tools or LLM-generated fixes could lead to unforeseen consequences and software failures.

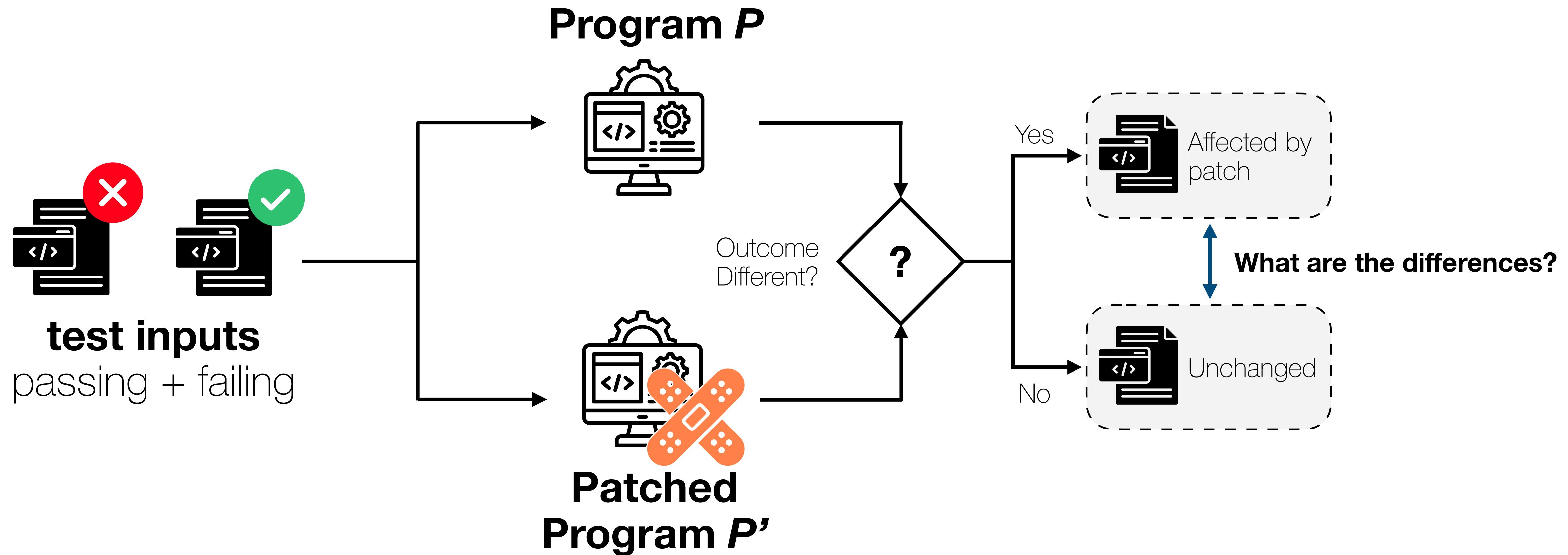
Great! But  
what else does  
the patch do?



**Developer**

# Automatic Explanations

## Differential Behavior



# Example

```
1 import math  
2  
3 def logarithm(x: float):  
4     return math.log(x)
```

Fig. 2. A simple function computing the natural logarithm of a number  $x$ . While straightforward, it does not handle edge cases such as negative values, leading to errors.

output	
	P
<code>logarithm(10)</code>	✓
<code>logarithm(-34)</code>	✗
<code>logarithm(-1521)</code>	✗



# Example

```
1 import math  
2  
3 def logarithm(x: float):  
4     return math.log(x)
```

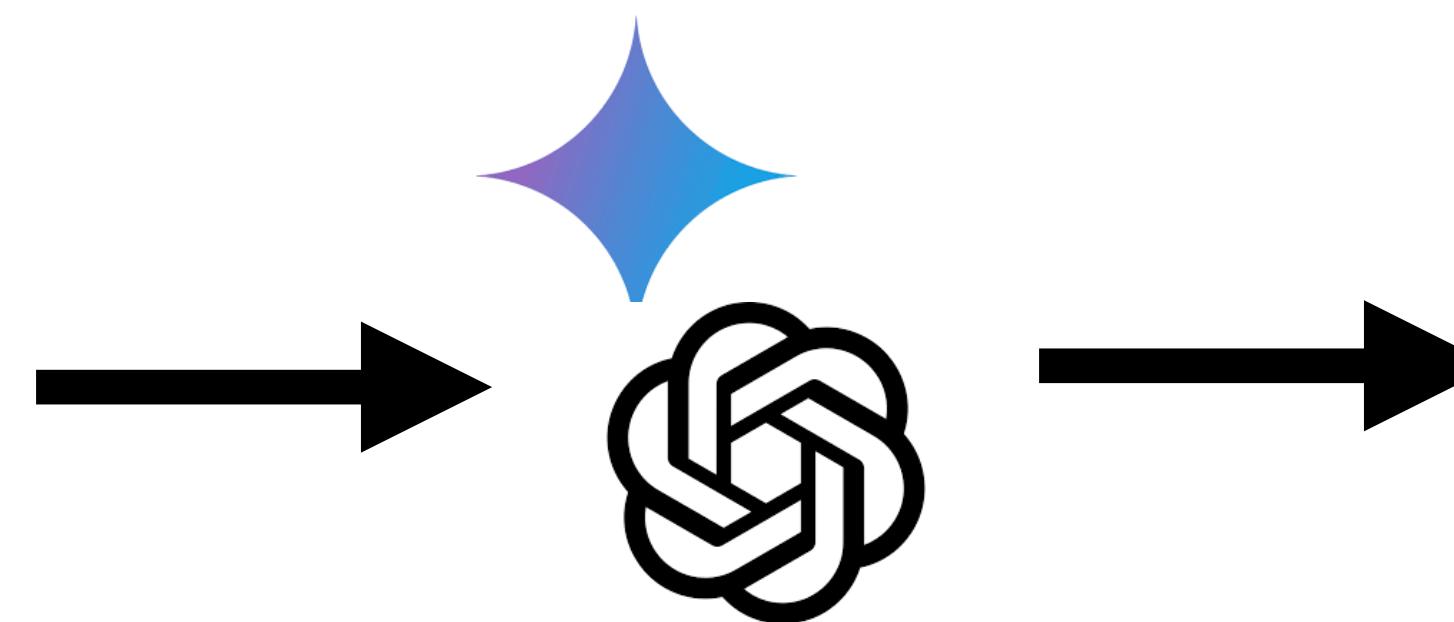


Fig. 2. A simple function computing the natural logarithm of a number  $x$ . While straightforward, it does not handle edge cases such as negative values, leading to errors.

```
1 import math  
2  
3 def logarithm(x: float):  
4     # Patch applied to cover edge cases  
5     if x == 0:  
6         return float("-inf")  
7     elif x > 0:  
8         return math.log(x)  
9     else:  
10        raise TypeError("Input must be a" \  
11                           "positive value")
```

Fig. 3. Patched logarithmic function. It explicitly checks for zero and negative numbers. These changes ensure robust handling of edge cases.

	output		changed?
	$P$	$P'$	
$\logarithm(10)$			
$\logarithm(-34)$			
$\logarithm(-1521)$			

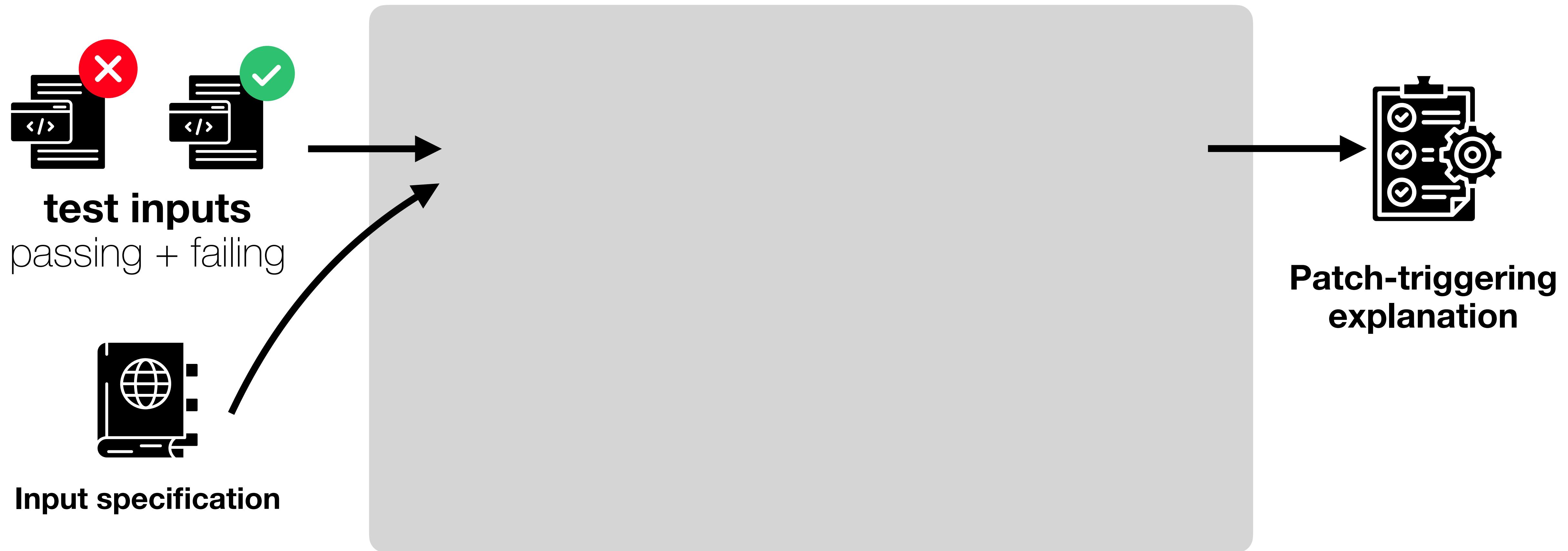
**<number>  $\leq 0$**



What are the properties of the inputs that are affected by the patch

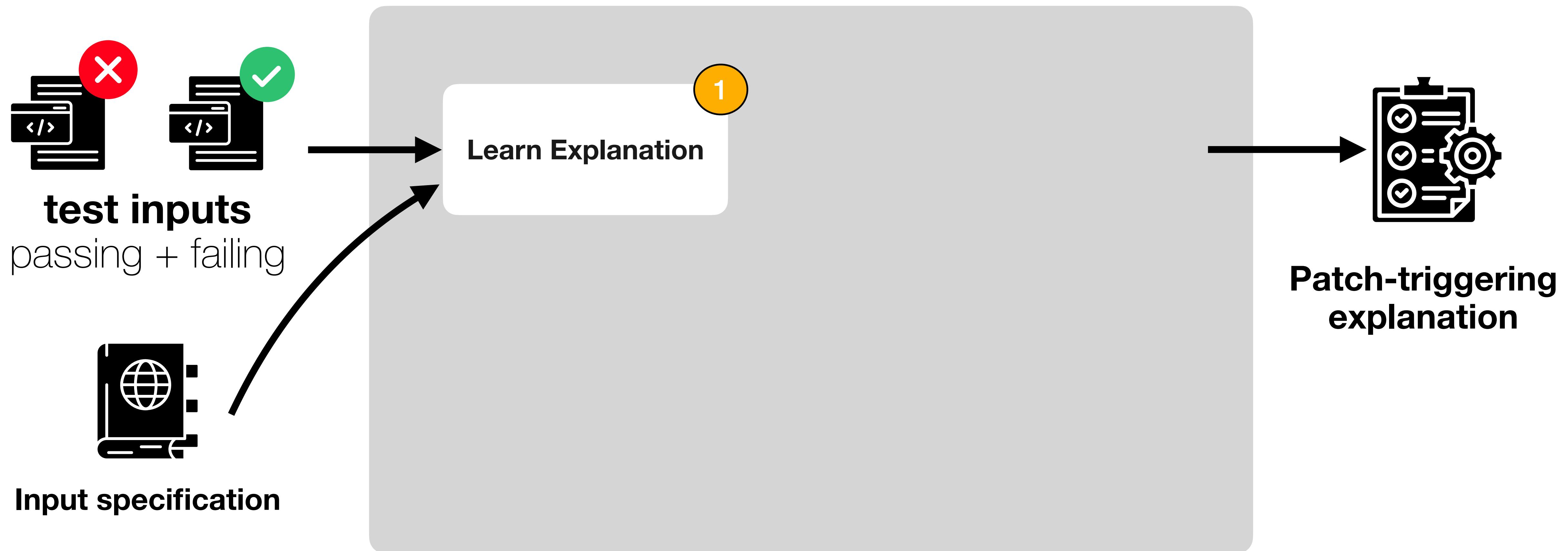
# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



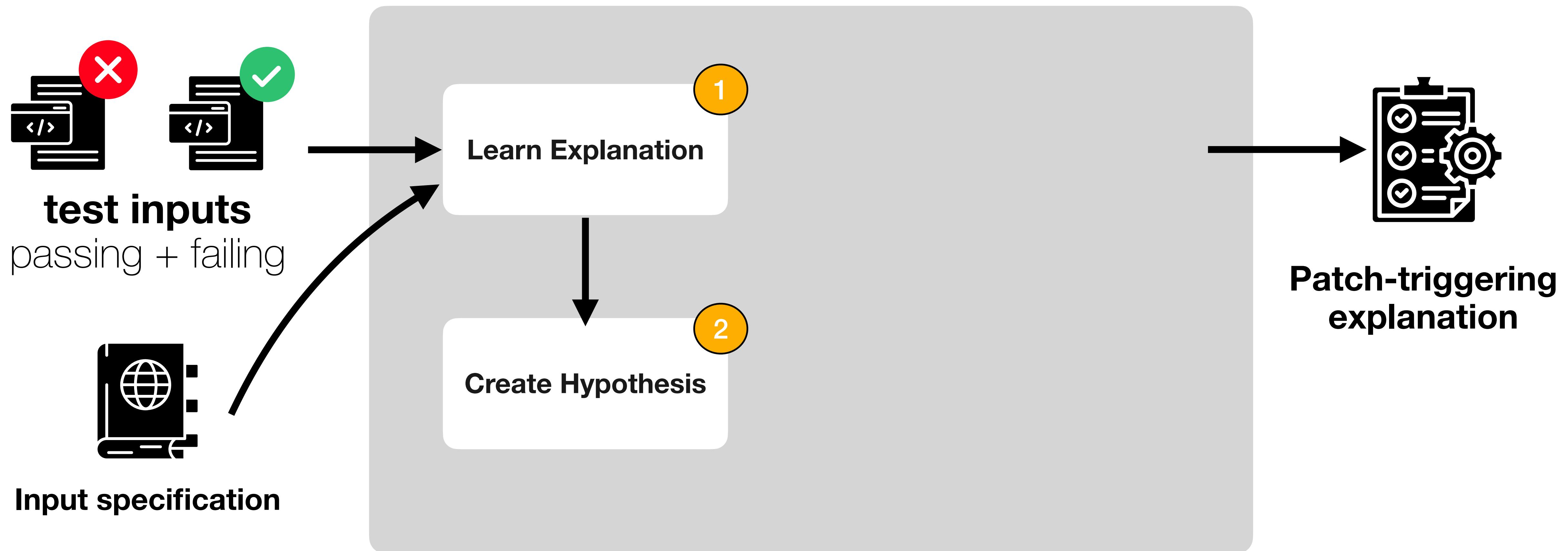
# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



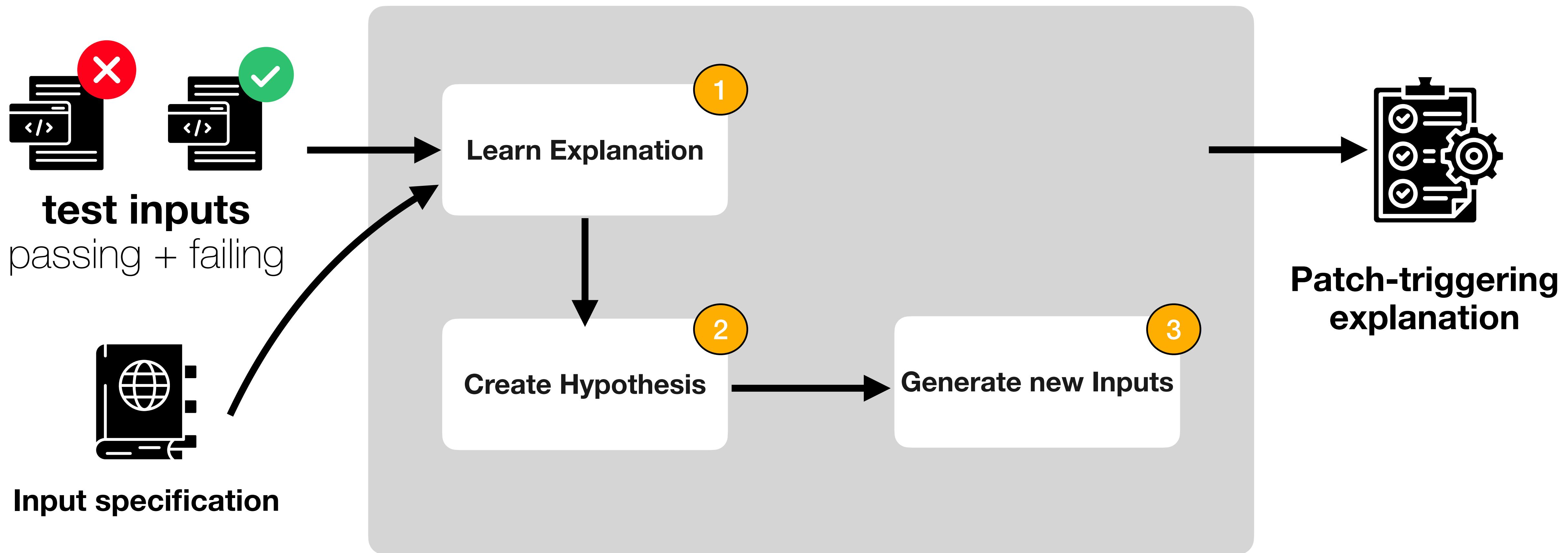
# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



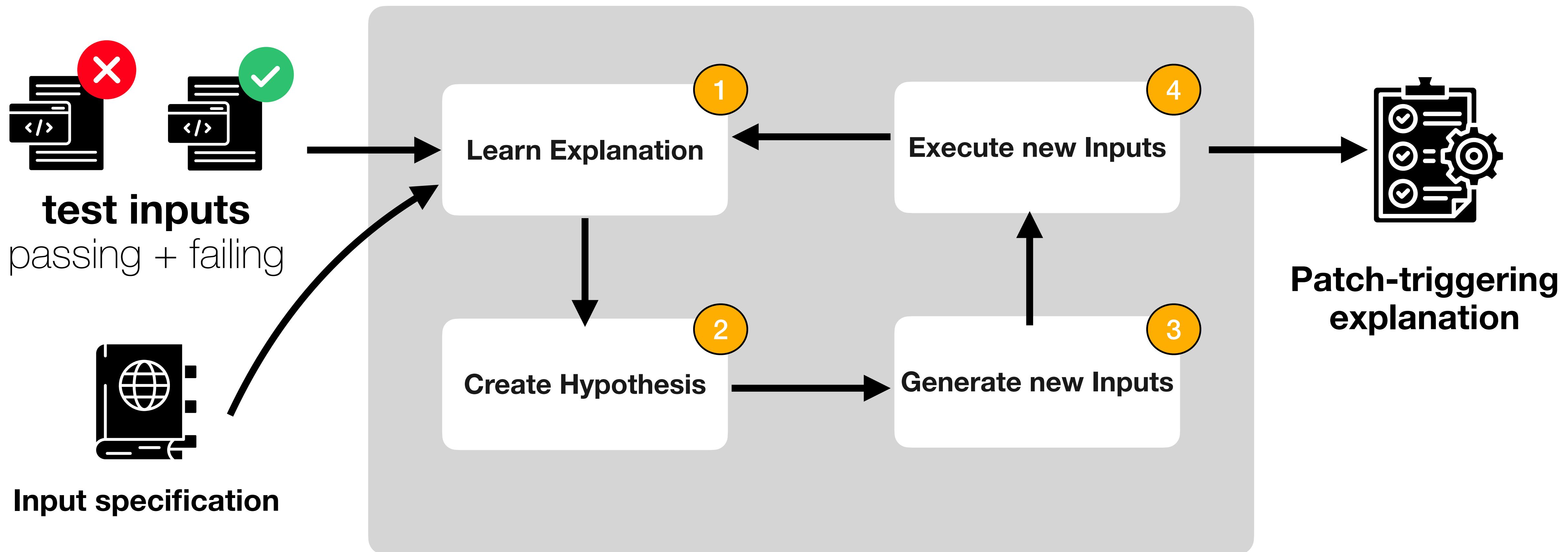
# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



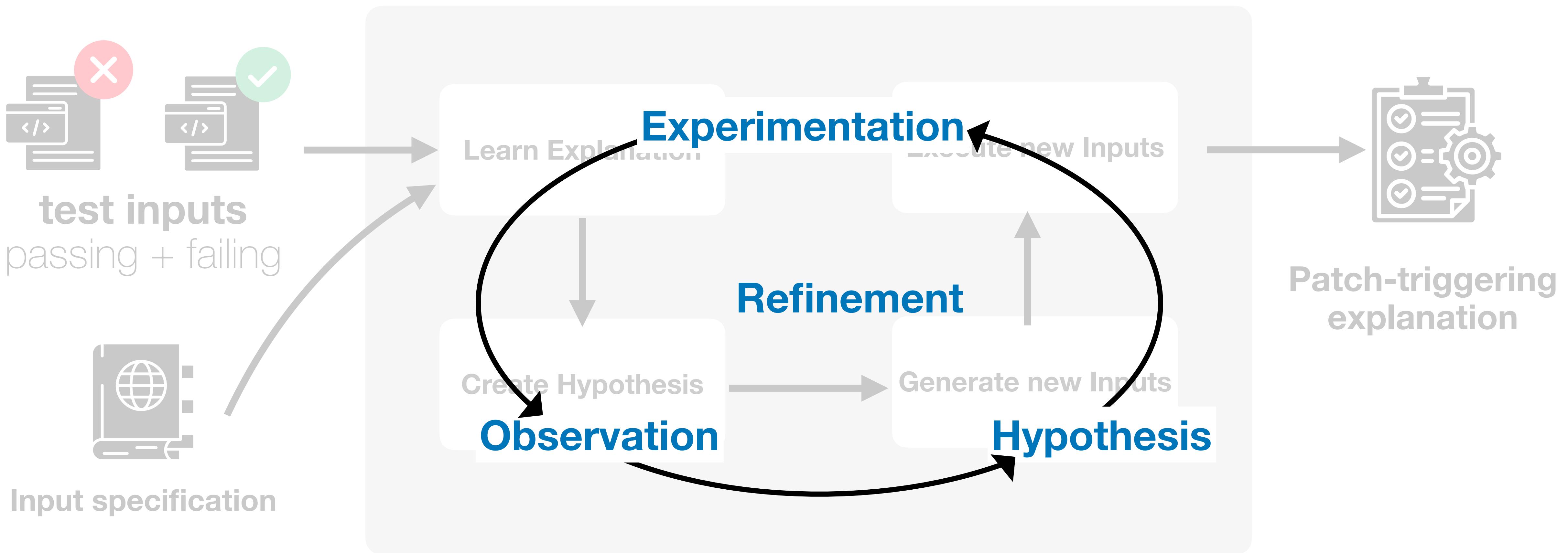
# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



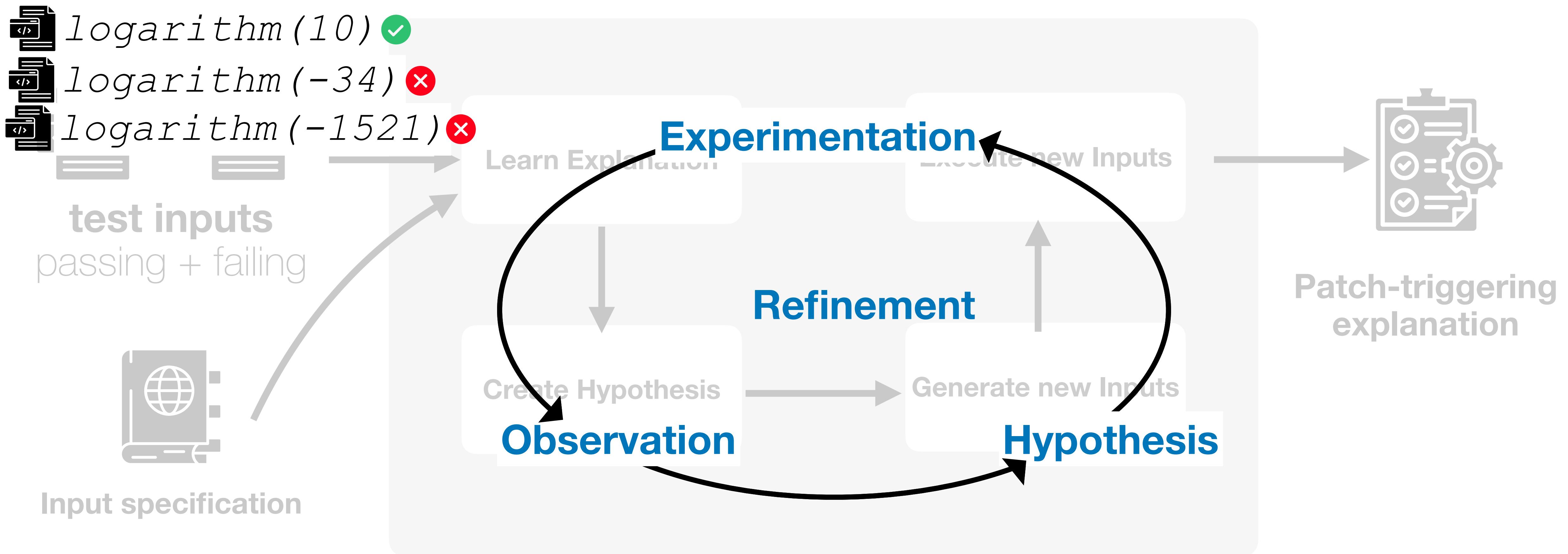
# Automatic Explanations

## Scientific Debugging



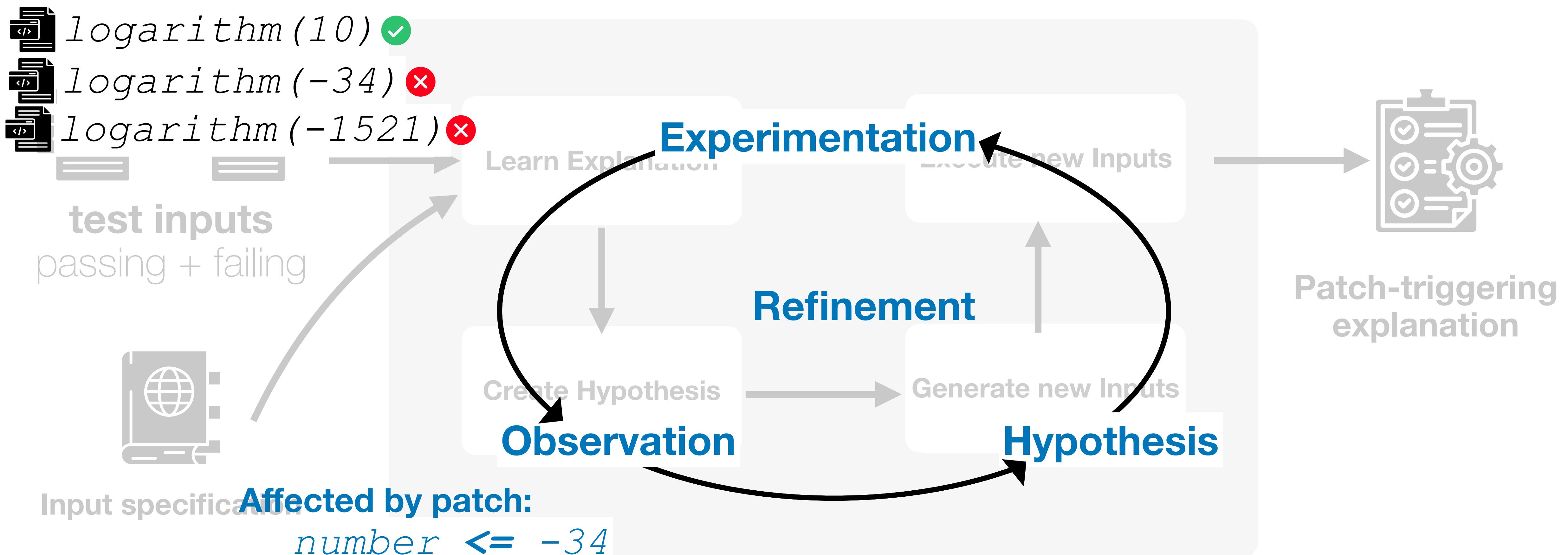
# Automatic Explanations

## Scientific Debugging



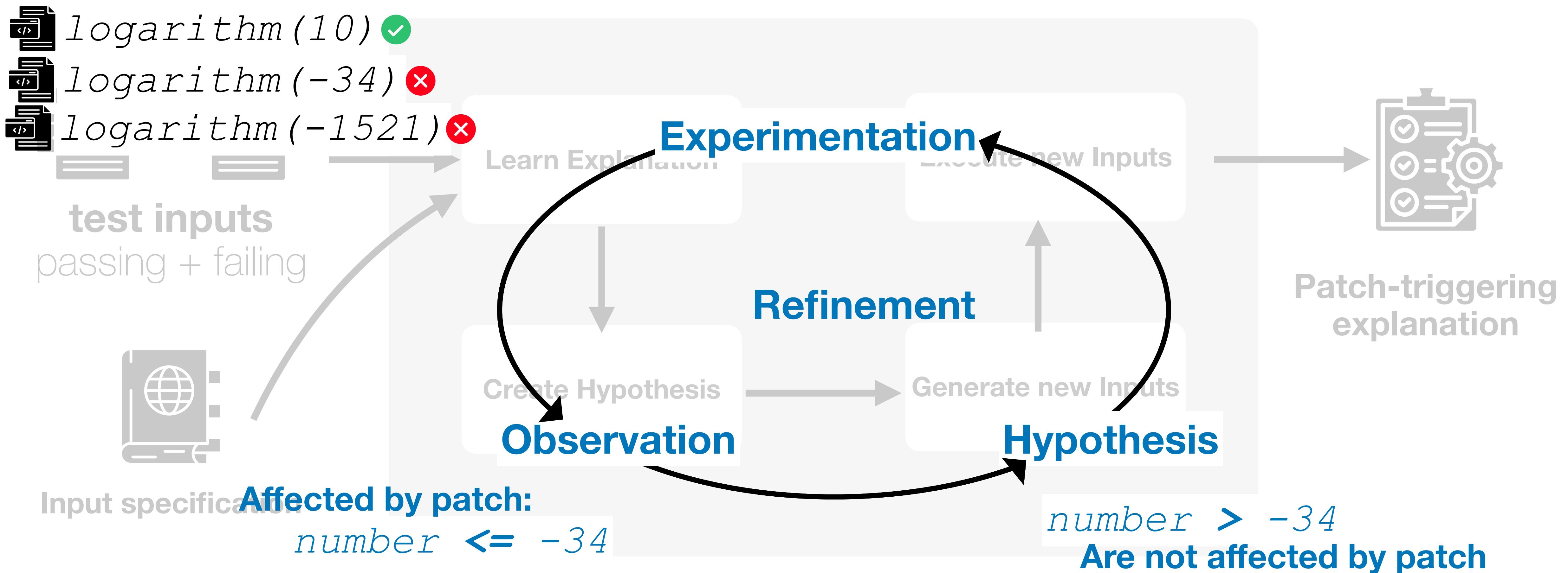
# Automatic Explanations

## Scientific Debugging



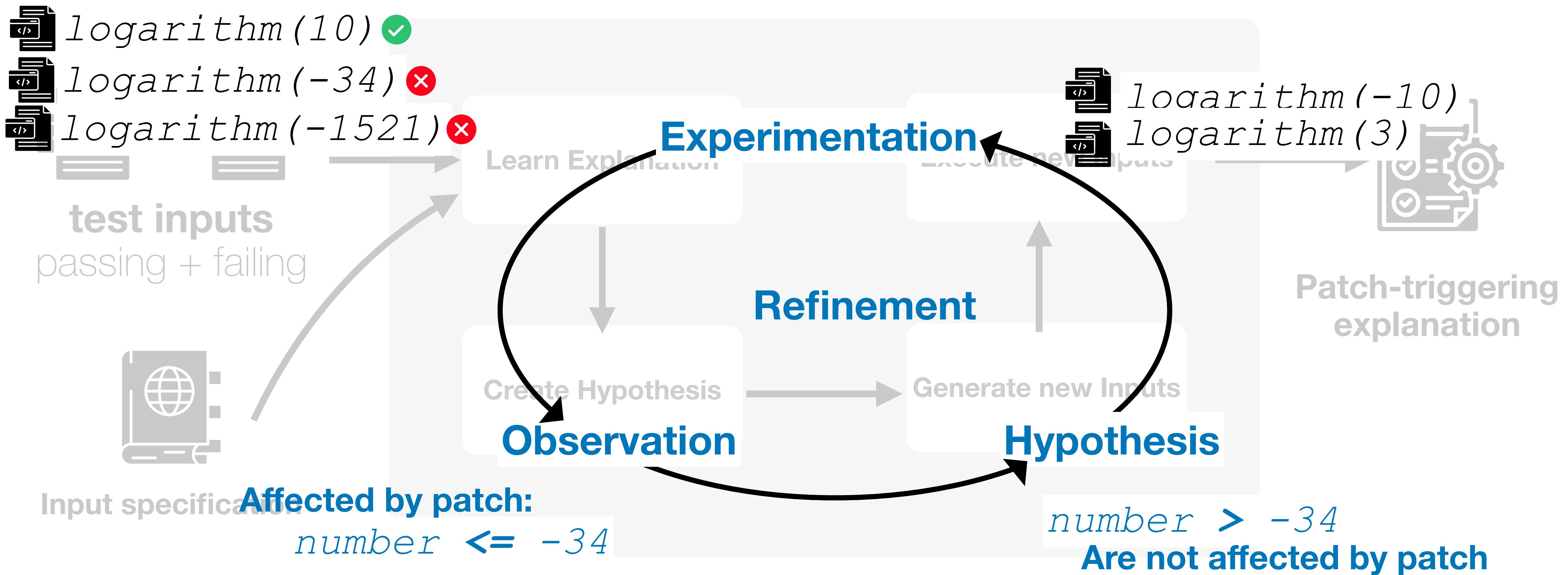
# Automatic Explanations

## Scientific Debugging



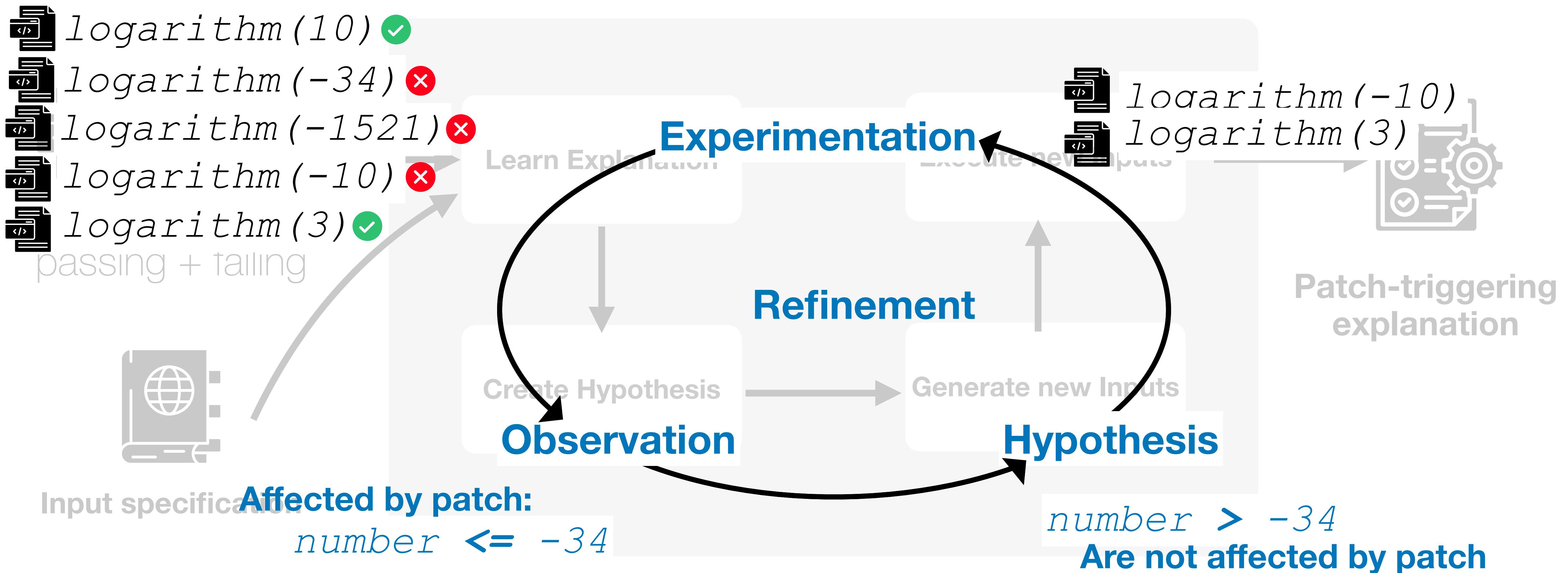
# Automatic Explanations

## Scientific Debugging



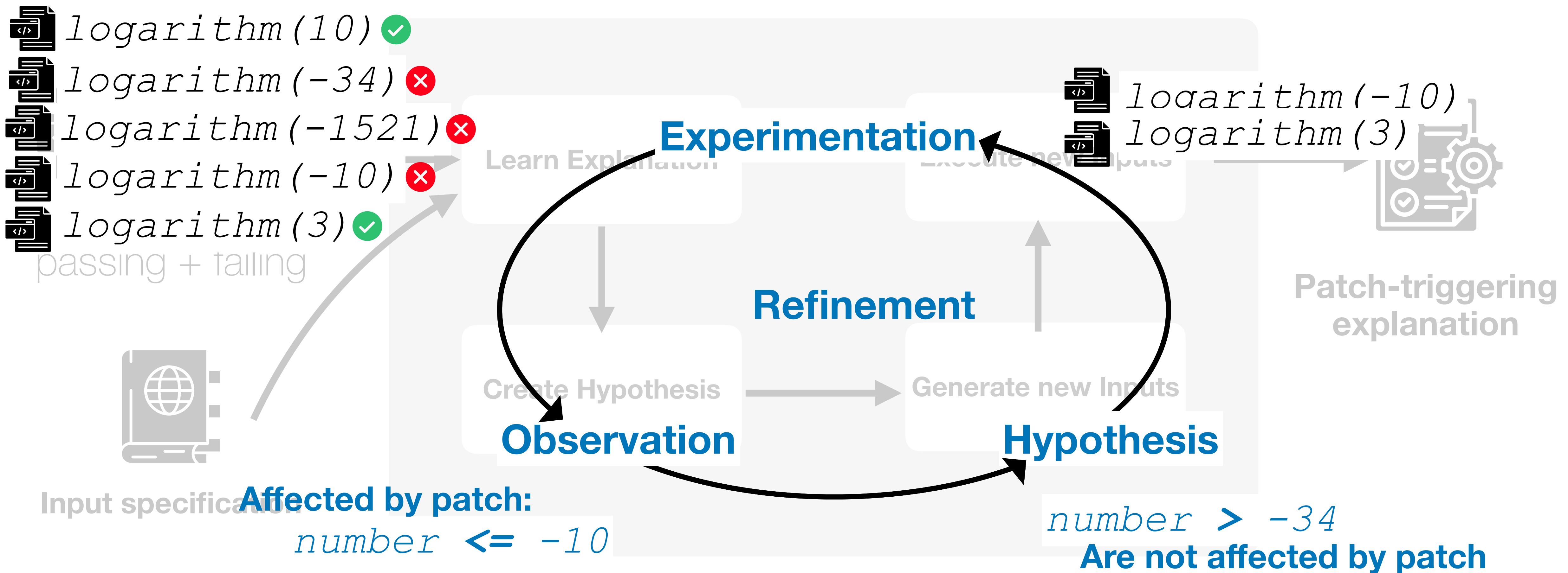
# Automatic Explanations

## Scientific Debugging



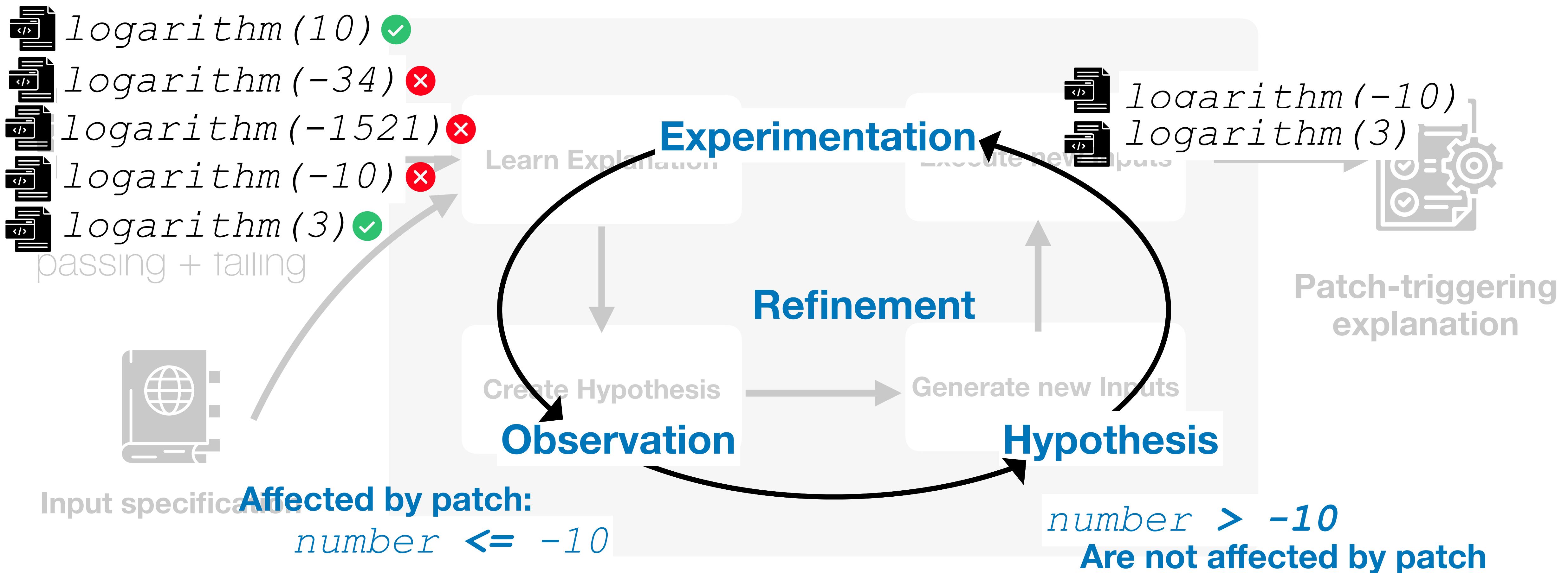
# Automatic Explanations

## Scientific Debugging



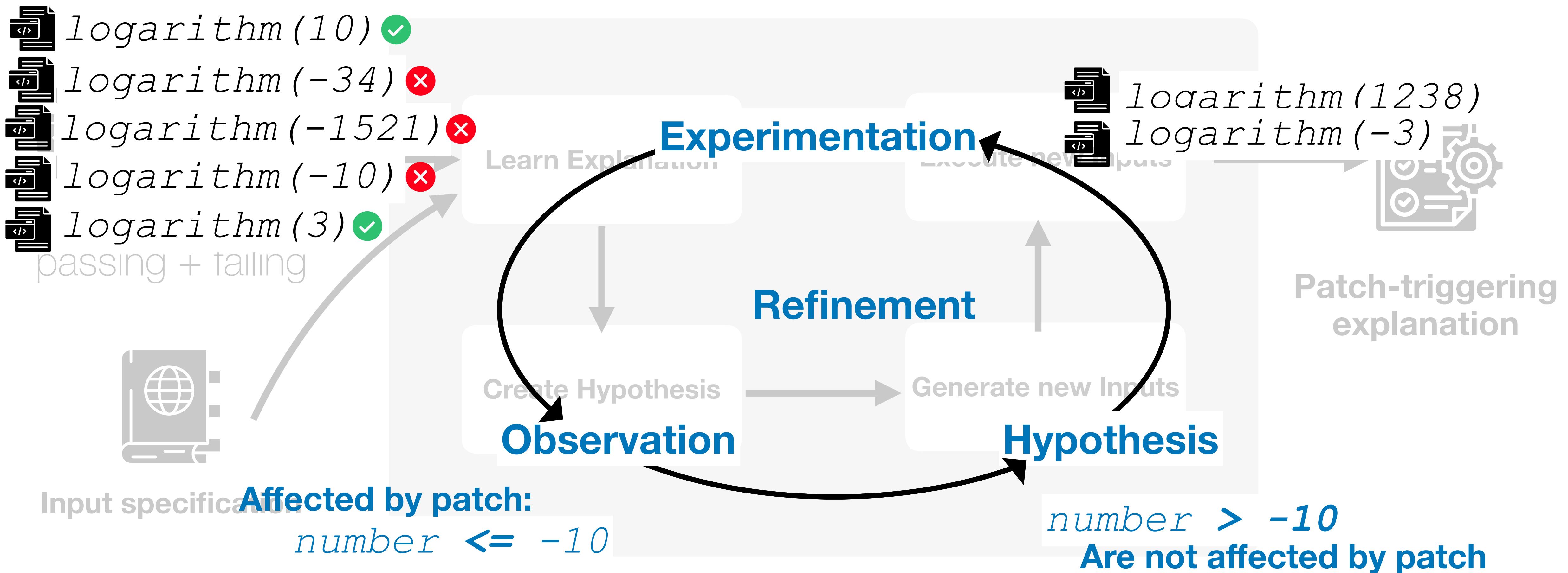
# Automatic Explanations

## Scientific Debugging



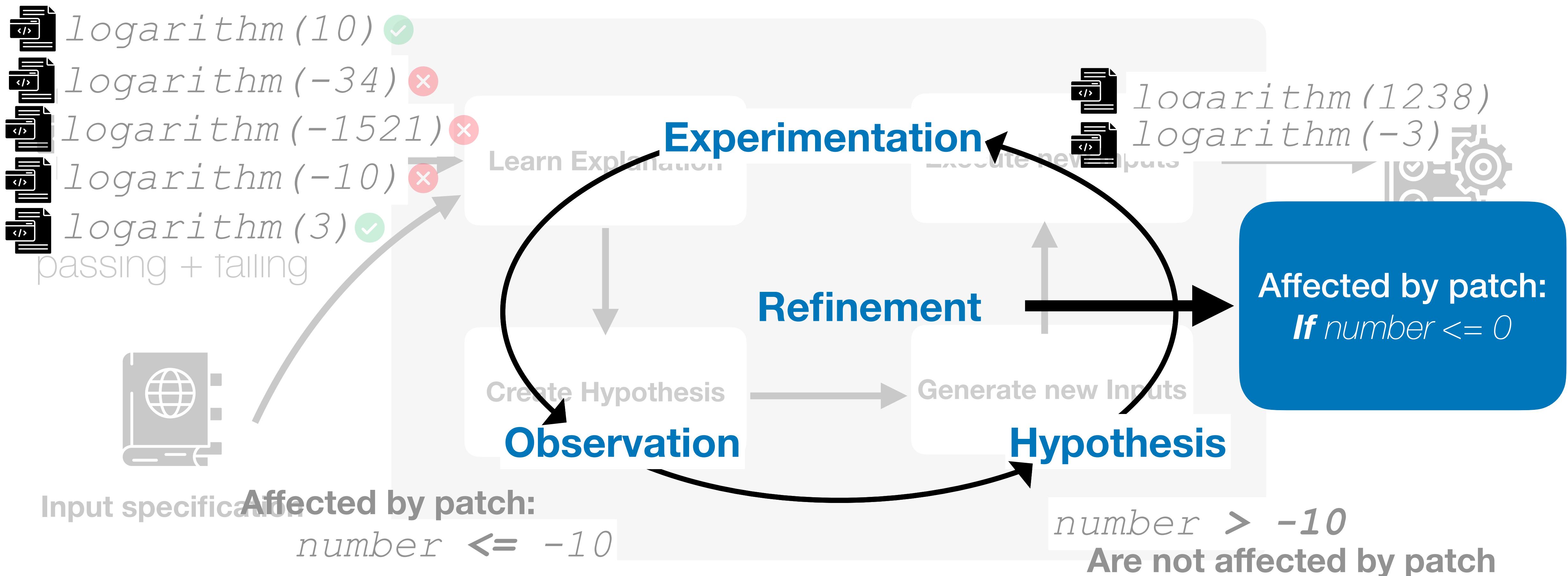
# Automatic Explanations

## Scientific Debugging



# Automatic Explanations

## Scientific Debugging



# Benchmark Results

## RQ1: Predictor

*How accurately can the explanations predict whether an input belongs to the input space affected by the patch?*

## RQ2: Producer

*How effectively can our explanations generate new inputs that belong to the input space affected by the patch?*

Subject	RQ1 Predictor		RQ2 Producer	
	Precision	Recall	Precision	Recall
Logarithm	100%	100%	100%	100%
Calculator	100%	100%	100%	100%
Middle.1	100%	100%	100%	100%
Middle.2	100%	100%	100%	100%
Expression	71%	93%	100%	66%
Markup.1	65%	100%	99%	100%
Markup.2	72%	100%	85%	100%
Pysnooper.1	100%	100%	100%	100%
Pysnooper.2	100%	100%	100%	100%
Cookiecutter.1	60%	92%	98%	76%
Cookiecutter.2	81%	93%	75%	89%
Cookiecutter.3	59%	100%	100%	100%
<b>Total Average</b>	<b>84%</b>	<b>98%</b>	<b>96%</b>	<b>94%</b>

# Limitations

## Vocabulary & Pattern Catalogue

*Reliance on an appropriate vocabulary to express the conditions under which a patch is triggered.*

## Requires Grammar/Specifaction

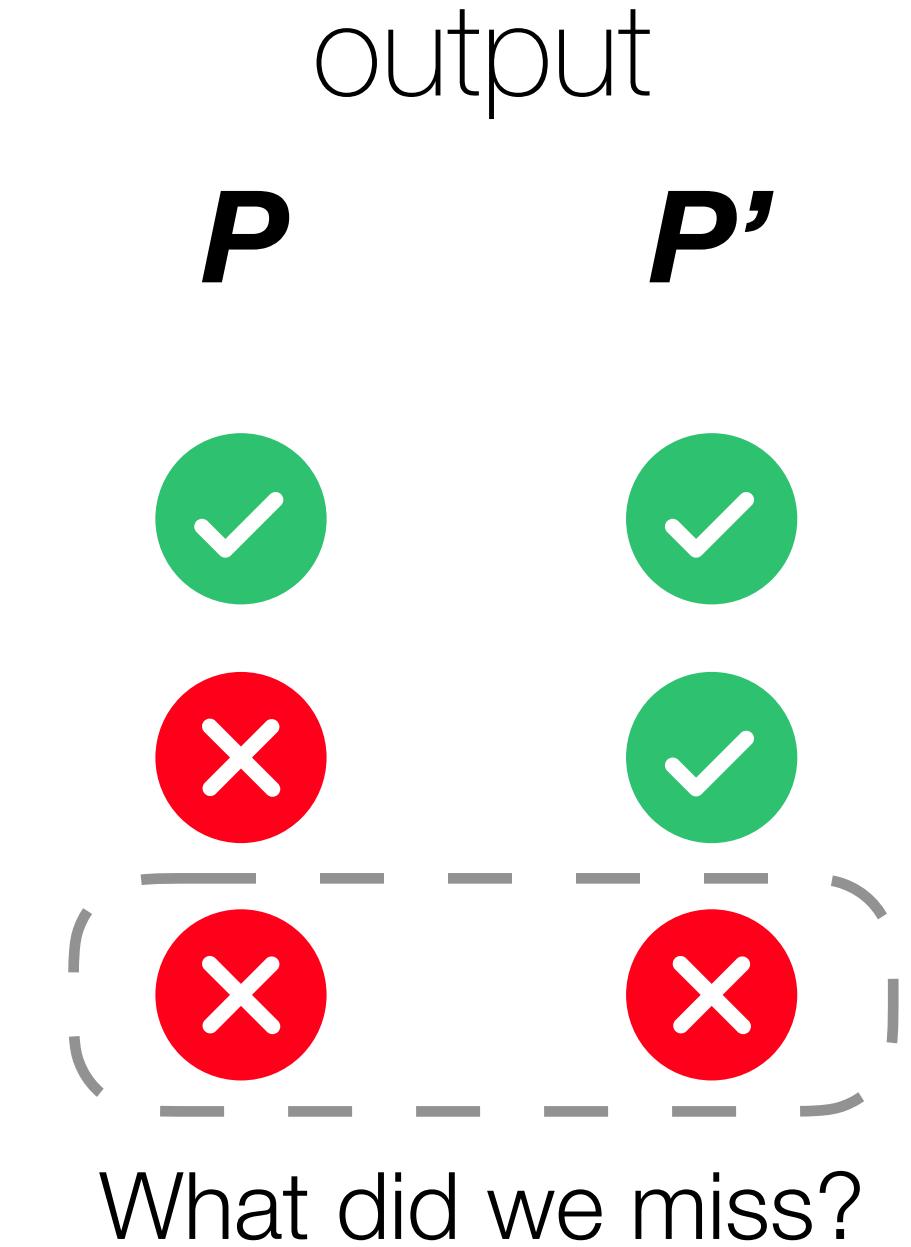
*Explanations are constructed based on the provided grammar. Grammars may not always be available.*

# Next Steps?

## Partial and Incomplete Patches

*Extended to identify partial or incomplete patches by analyzing the properties of failing inputs unaffected by the patch*

-   $\logarithm(0)$
-   $\logarithm(-3)$
-   $\logarithm(-3i)$



# Next Steps?

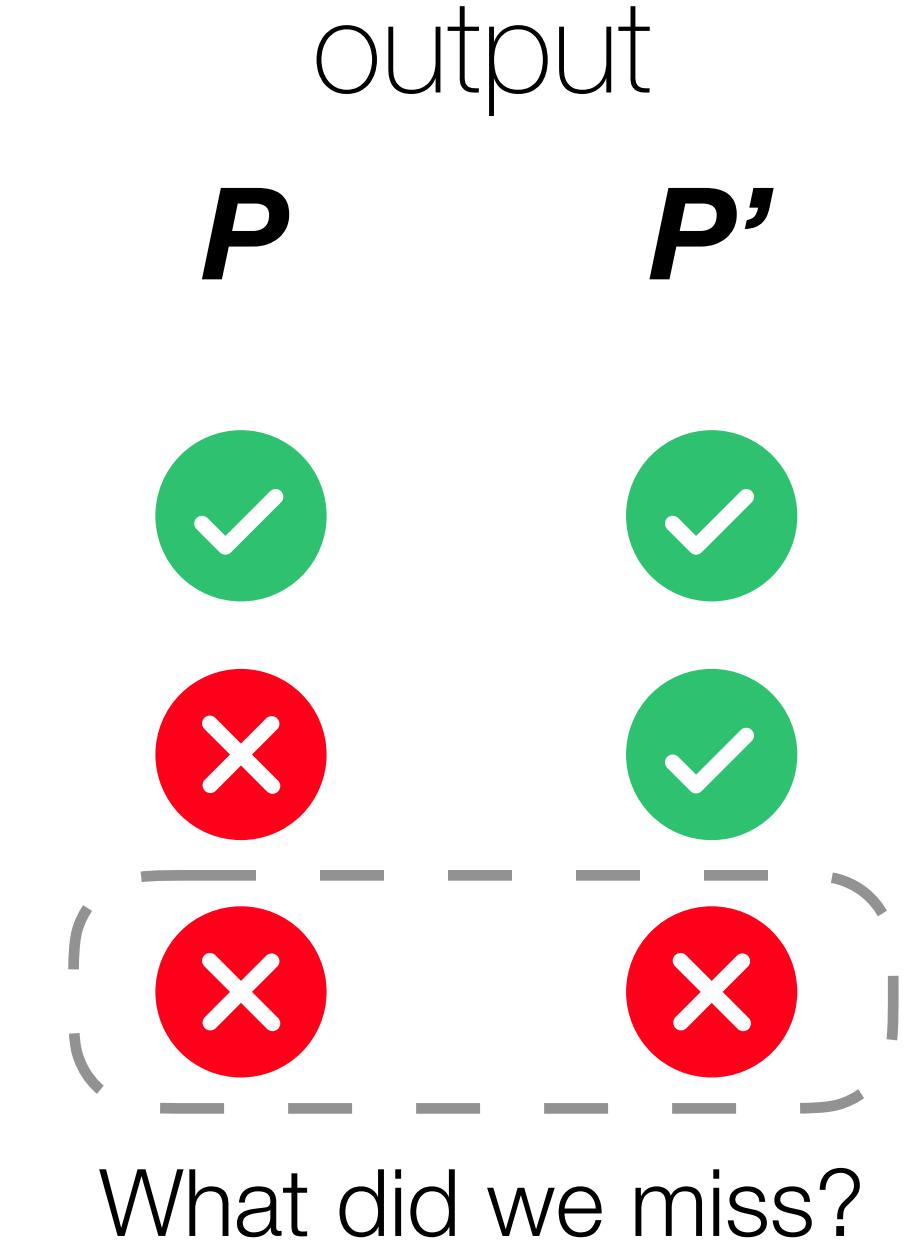
## Partial and Incomplete Patches

*Extended to identify partial or incomplete patches by analyzing the properties of failing inputs unaffected by the patch*

## Integration with Symbolic Execution

*Can the explanations be used to guide symbolic/concolic execution techniques?*

-   $\logarithm(0)$
-   $\logarithm(-3)$
-   $\logarithm(-3i)$



# Next Steps?

## Partial and Incomplete Patches

*Extended to identify partial or incomplete patches by analyzing the properties of failing inputs unaffected by the patch*

	output $P$	output $P'$
 $\logarithm(0)$	✓	✓
 $\logarithm(-3)$	✗	✓
 $\logarithm(-3i)$	✗	✗

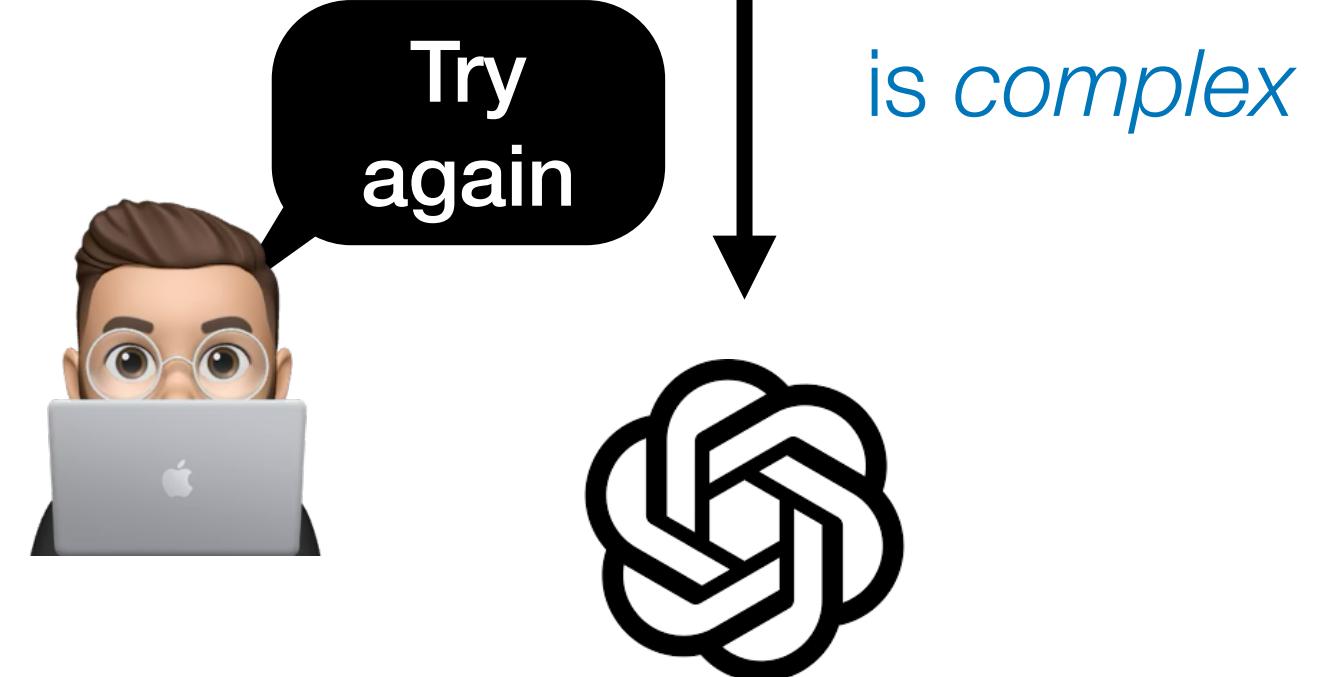
What did we miss?

## Integration with Symbolic Execution

*Can the explanations be used to guide symbolic/concolic execution techniques?*

## Additional Applications Scenarios

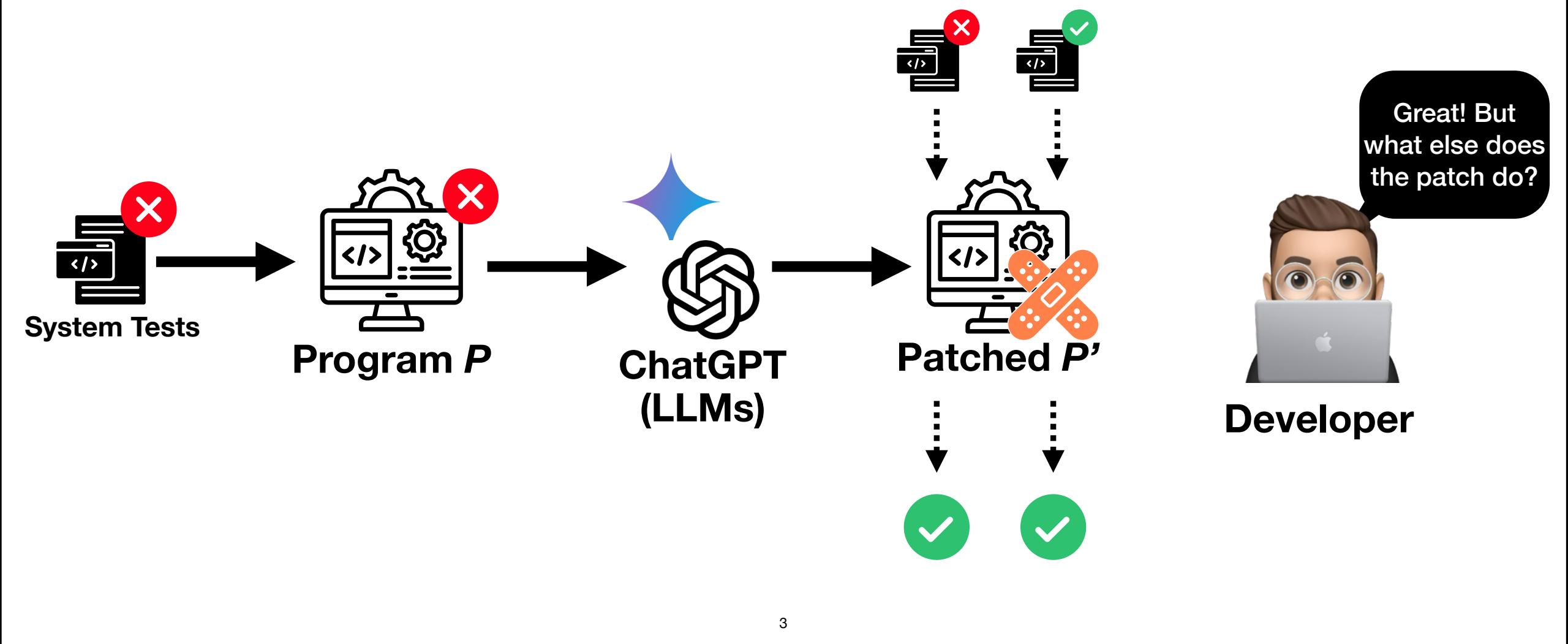
*Automated Program Repair & Guide LLMs*





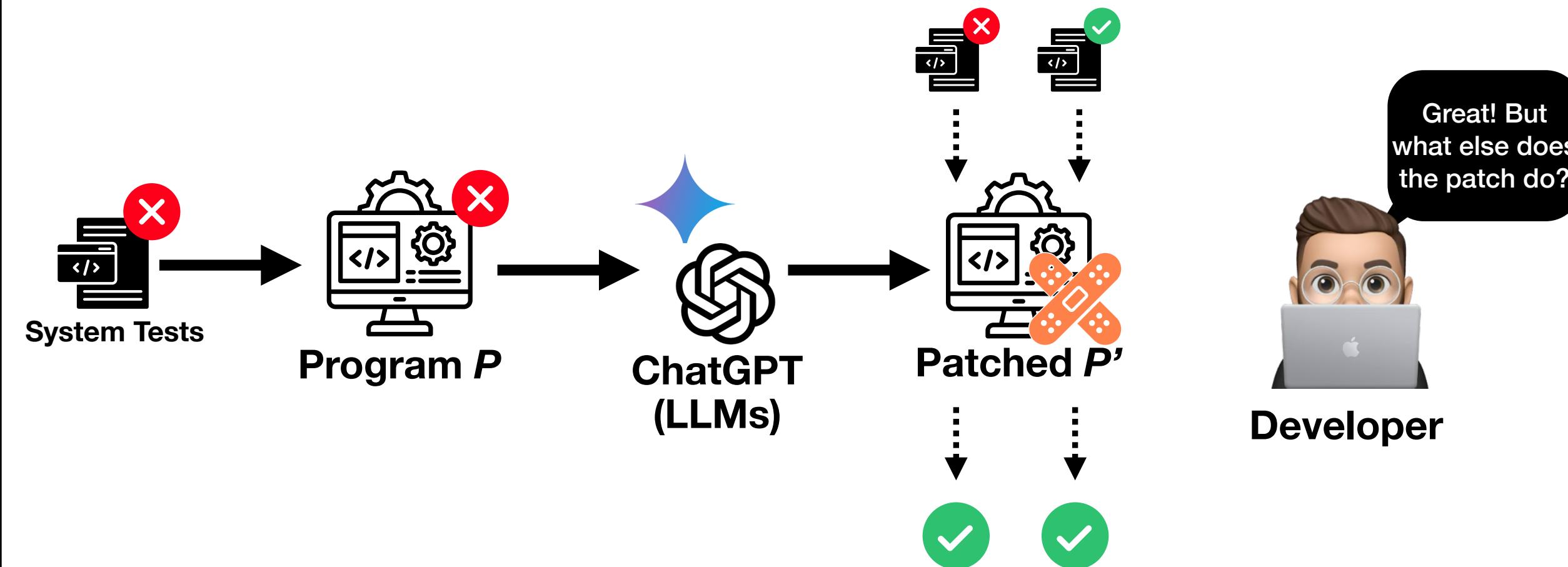
# Trust Issues?

Would you trust a patch generated by ChatGPT to fix your critical code?



# Trust Issues?

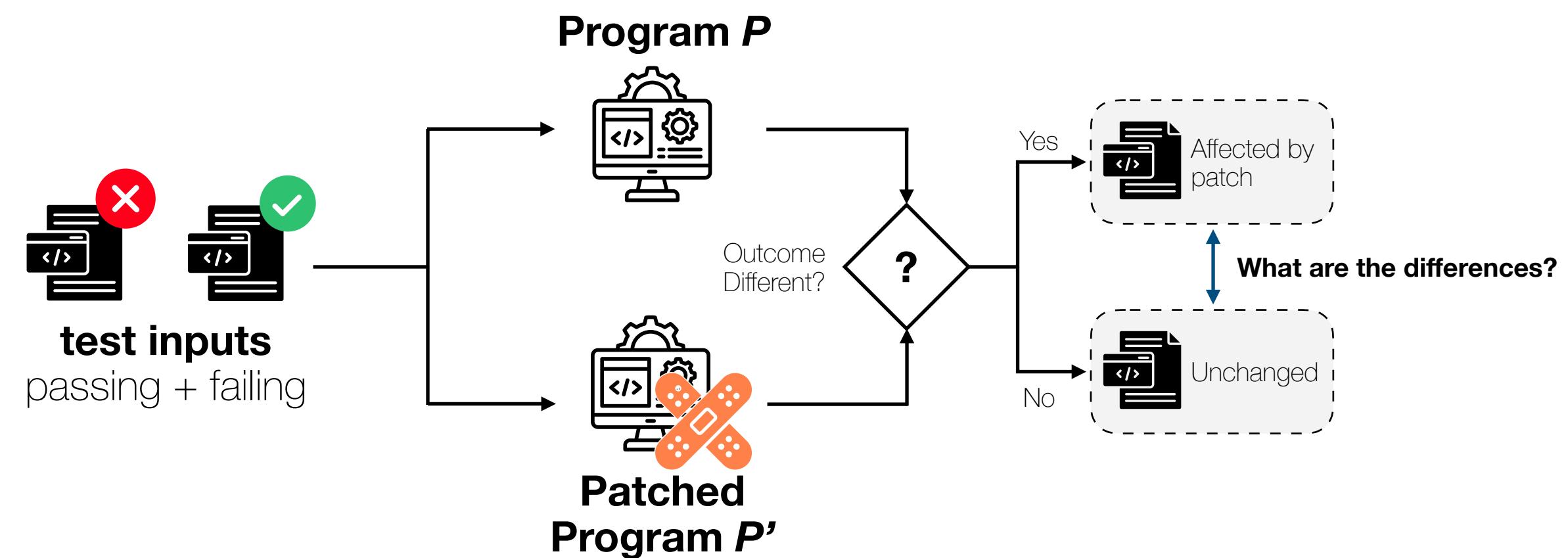
Would you trust a patch generated by ChatGPT to fix your critical code?



3

# Automatic Explanations

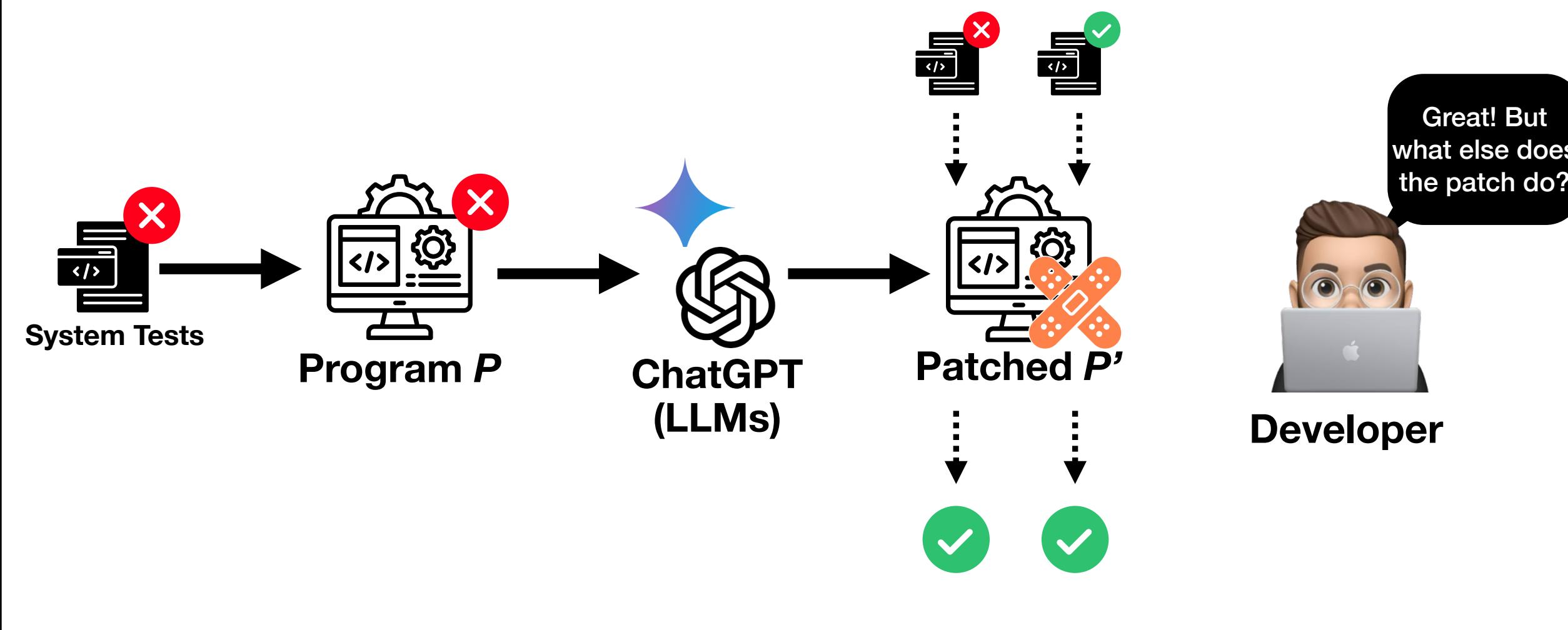
Differential Behavior



x

# Trust Issues?

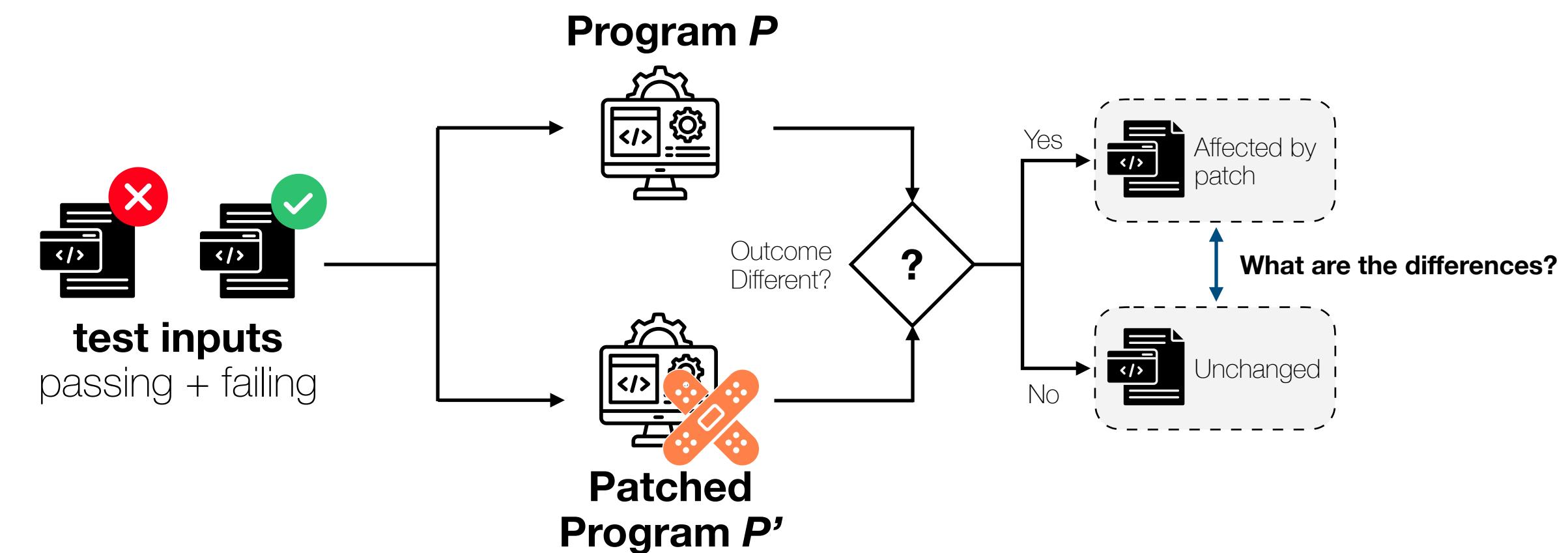
Would you trust a patch generated by ChatGPT to fix your critical code?



3

# Automatic Explanations

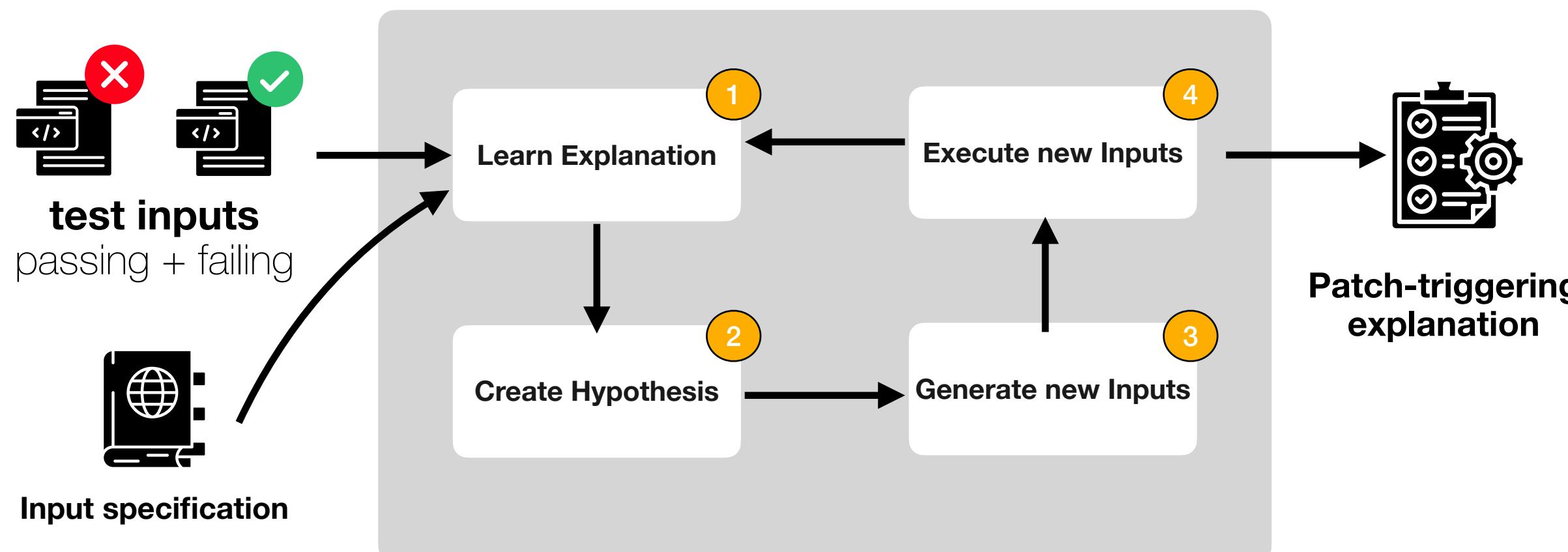
Differential Behavior



x

# Automatic Explanations

Generating explanations to describe the input properties affected by a patch

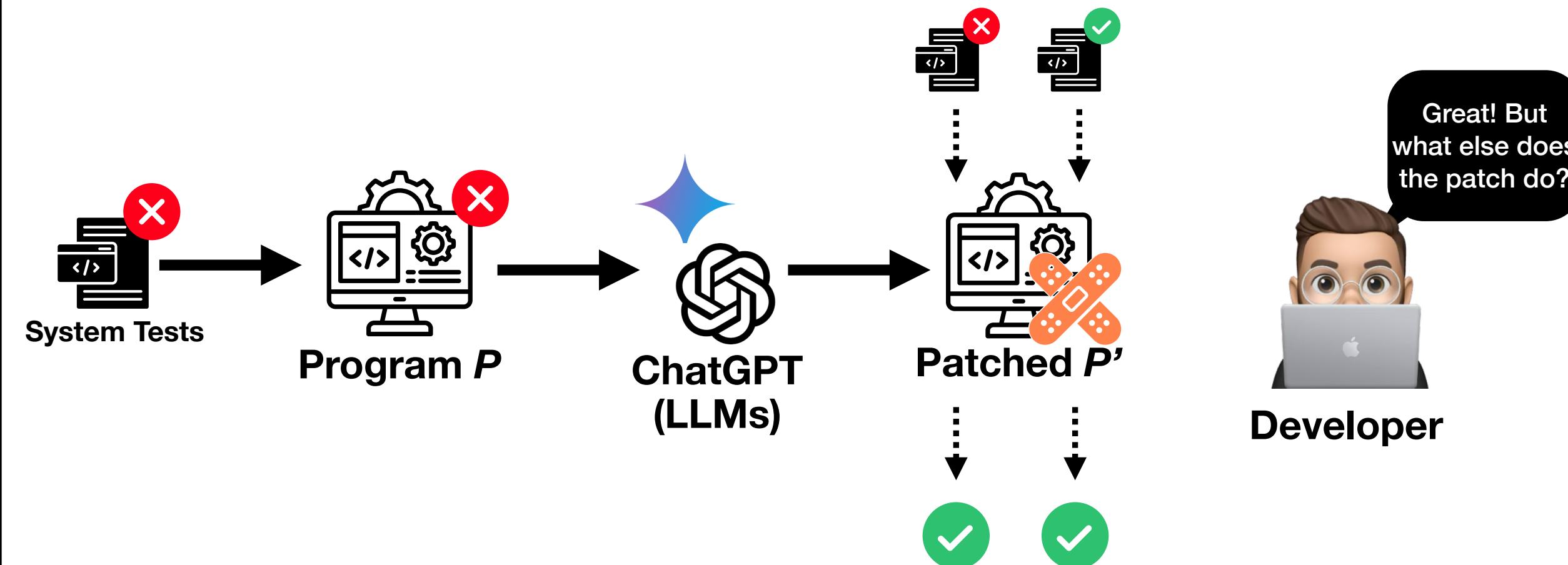


x

2

# Trust Issues?

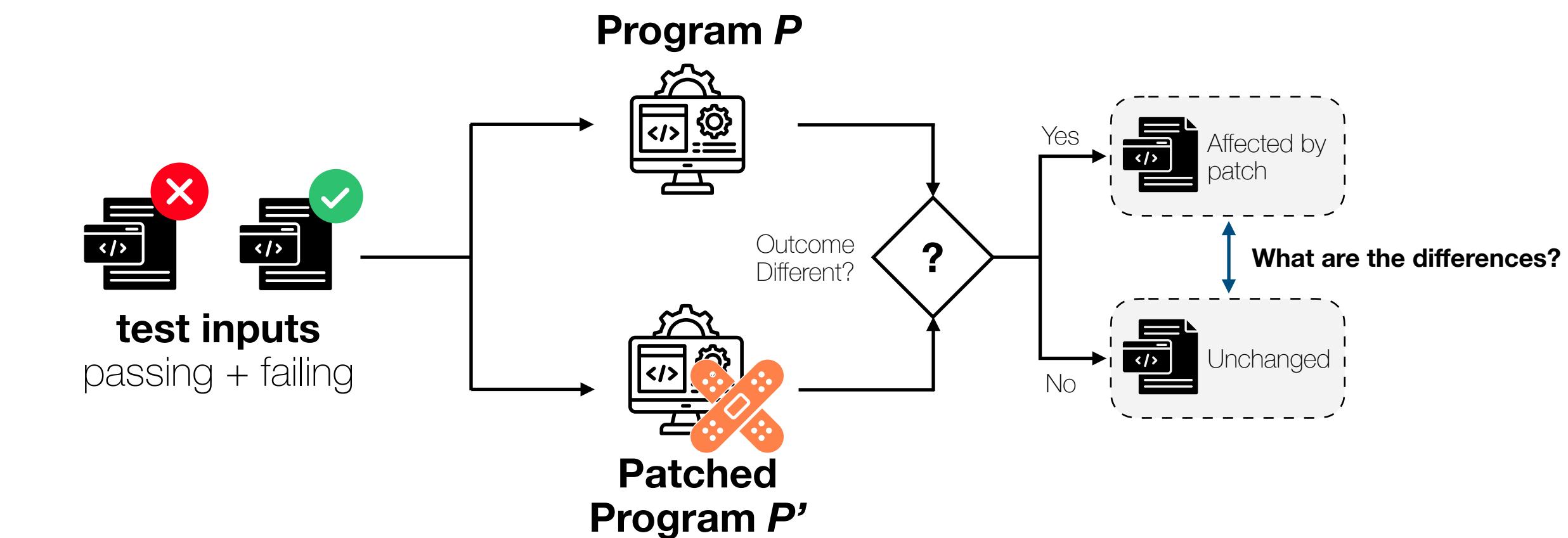
Would you trust a patch generated by ChatGPT to fix your critical code?



3

# Automatic Explanations

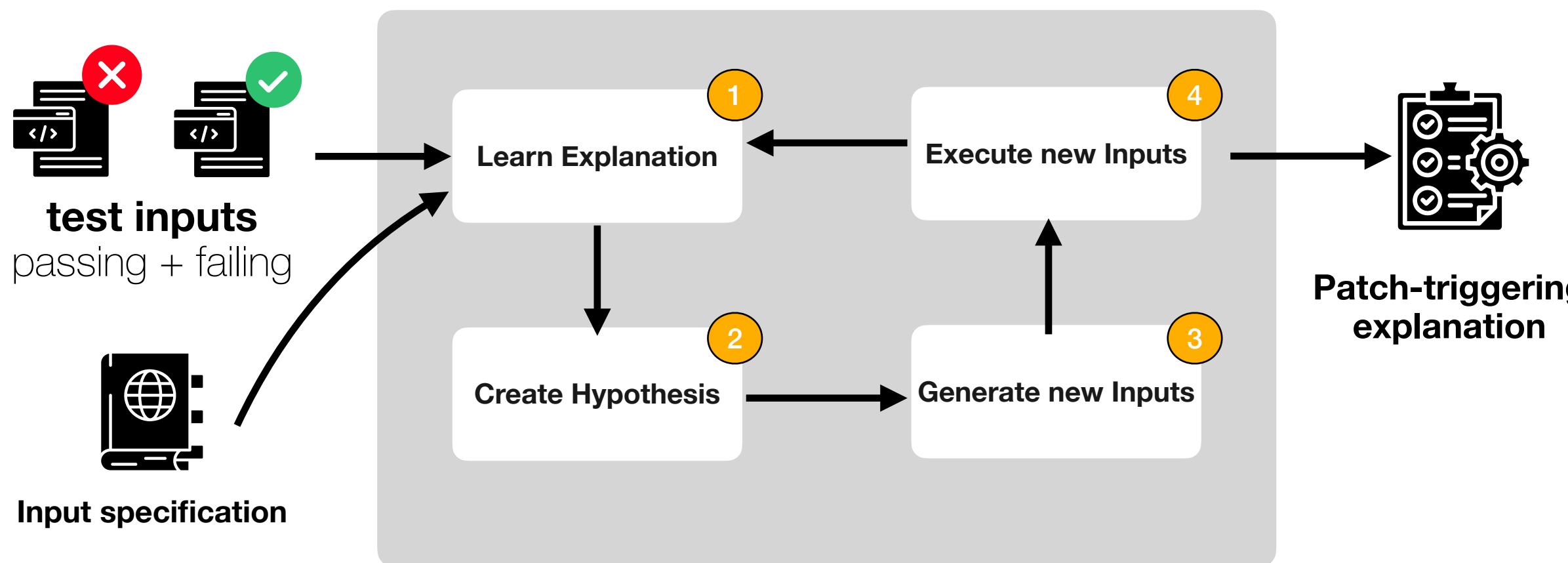
Differential Behavior



x

# Automatic Explanations

Generating explanations to describe the input properties affected by a patch



x

# Benchmark Results

## RQ1: Predictor

How accurately can the explanations predict whether an input belongs to the input space affected by the patch?

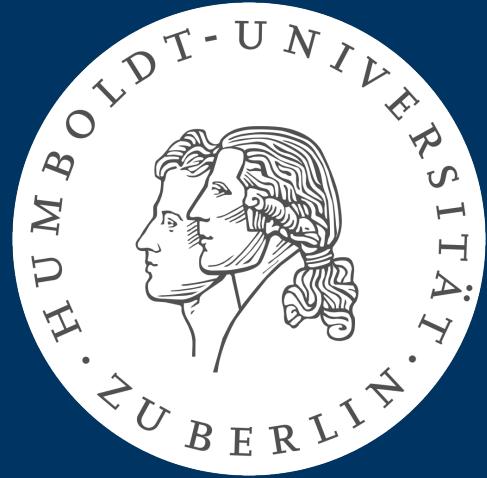
## RQ2: Producer

How effectively can our explanations generate new inputs that belong to the input space affected by the patch?

TABLE I  
PRECISION AND RECALL WHEN USING AVICENNAPATCH AS A PREDICTOR AND PRODUCER. PRECISION AND RECALL ARE AVERAGES OVER 10 RUNS.

Subject	RQ1 Predictor		RQ2 Producer	
	Precision	Recall	Precision	Recall
Logarithm	100%	100%	100%	100%
Calculator	100%	100%	100%	100%
Middle.1	100%	100%	100%	100%
Middle.2	100%	100%	100%	100%
Expression	71%	93%	100%	66%
Markup.1	65%	100%	99%	100%
Markup.2	72%	100%	85%	100%
Pysnooper.1	100%	100%	100%	100%
Pysnooper.2	100%	100%	100%	100%
Cookiecutter.1	60%	92%	98%	76%
Cookiecutter.2	81%	93%	75%	89%
Cookiecutter.3	59%	100%	100%	100%
Total Average	84%	98%	96%	94%

# Contact



## Website

<https://martin-eberlein.com>

## Mail

[martin.eberlein@hu-berlin.de](mailto:martin.eberlein@hu-berlin.de)



[martineberlein](#)

## LinkedIn

Martin Eberlein

The screenshot shows a website page titled "Martin Eberlein". At the top right is a circular profile picture of a man with a beard and sunglasses, wearing a dark jacket and a beanie, standing outdoors with mountains in the background. Below the photo is a brief bio: "Martin Eberlein is a doctoral researcher at Humboldt-Universität zu Berlin." To the left of the bio is a small paragraph: "Photo taken on an expedition to antarctica." On the right side of the bio is a section titled "About Me" with a small plant icon. It contains text about the researcher's background and current work. Below the bio is a "Quick Links" sidebar with links to publications, current theses, CV, and academic service. At the bottom of the page is a "Latest News" section with a list of recent updates.

**Martin Eberlein**



Martin Eberlein is a doctoral researcher at Humboldt-Universität zu Berlin.

Photo taken on an expedition to antarctica.

Hosted on GitHub Pages — Theme by [orderedlist](#)

**About Me**

I am a doctoral researcher and Ph.D. candidate in the Software Engineering group of [Prof. Lars Grunske](#) at [Humboldt-Universität zu Berlin](#), Germany. Previously, I collaborated with [Prof. Andreas Zeller](#)'s research group at the [CISPA Helmholtz Center for Information Security](#) in Saarbrücken.

My research focuses on automated software engineering, particularly in software testing, vulnerability detection, and *fuzzing*. Currently, I am part of the **EMPEROR** project, which aims to automatically generate *explanations for program behaviors*, particularly program failures. By leveraging predictive and generative models, we seek to uncover and refine relationships among input features, enhancing our understanding of how and why software behaves as it does.

**Quick Links**

- Publications
- Current Theses ([New Topics!](#))
- CV
- [Academic Service](#)

**Latest News**

- I'll be presenting our paper "[Which Inputs Trigger my Patch](#)" at [APR'25](#) — co-located with ICSE 2025 — in Ottawa, Canada! 
- I've been invited to serve on the [FSE 2025 Artifact Track](#) Program Committee! Looking forward to reviewing outstanding artifact submissions. See you in Norway! 
- Our paper demonstrating [JAST](#) has been accepted at the [FSE'25 Demonstrations Track](#)! Preprint will follow soon!
- I have been invited to serve on the [ISSTA 2025 Tools Demonstrations](#) track Program Committee! Looking forward to many great submissions! 
- I co-founded [InputLab](#), a startup that generates test data for various formats, from electronic invoices to communication between government authorities, covering all input features. We secured [€900k in initial funding](#) from the German Federal Ministry of Education and Research!

# Describing Input Properties



## Input Specification Language

„Input Invariants“: Dominic Steinhöfel & Andreas Zeller, ESEC/FSE 2022

### Syntax

```
<heartbeat-request> ::= 0x1 <length> <payload> <padding>
```

### Input Grammar

```
<heartbeat-response> ::= 0x2 <length> <payload> <padding>
```

```
<length> ::= <uint16>
```

```
<payload> ::= ε | <byte> <payload>
```

```
<padding> ::= ε | <byte> <padding>
```

### Semantics

### ISLa Constraints

```
int(<length>) <= 7
```

```
len(<payload>) ≠ 9
```

```
int(<length>) > len(<payload>)
```