# The Power of Fuzzing and Large Language Models

Caroline Lemieux
The University of British Columbia

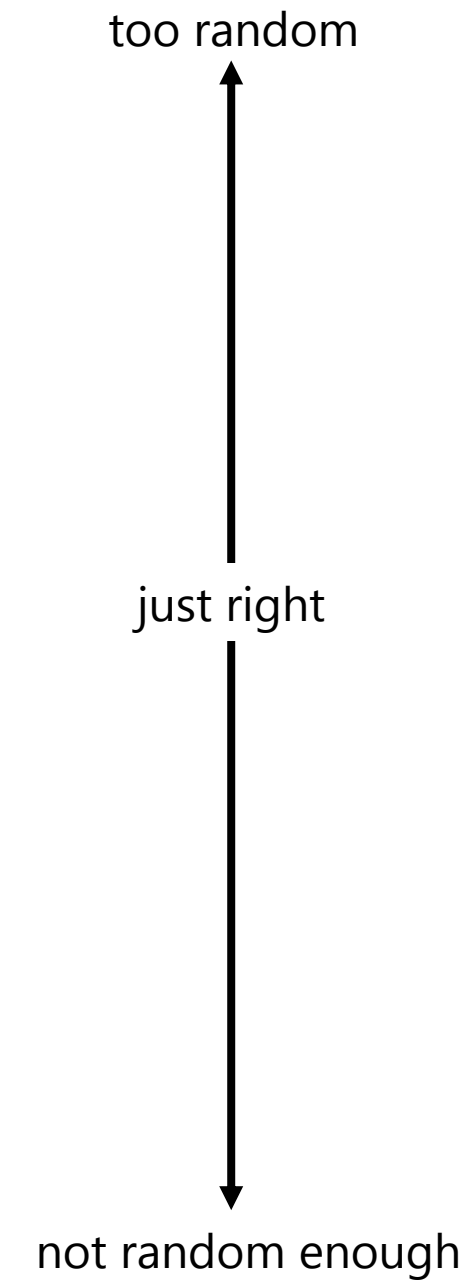
 @cestlemieux

 clemieux@cs.ubc.ca

 carolemieux.com

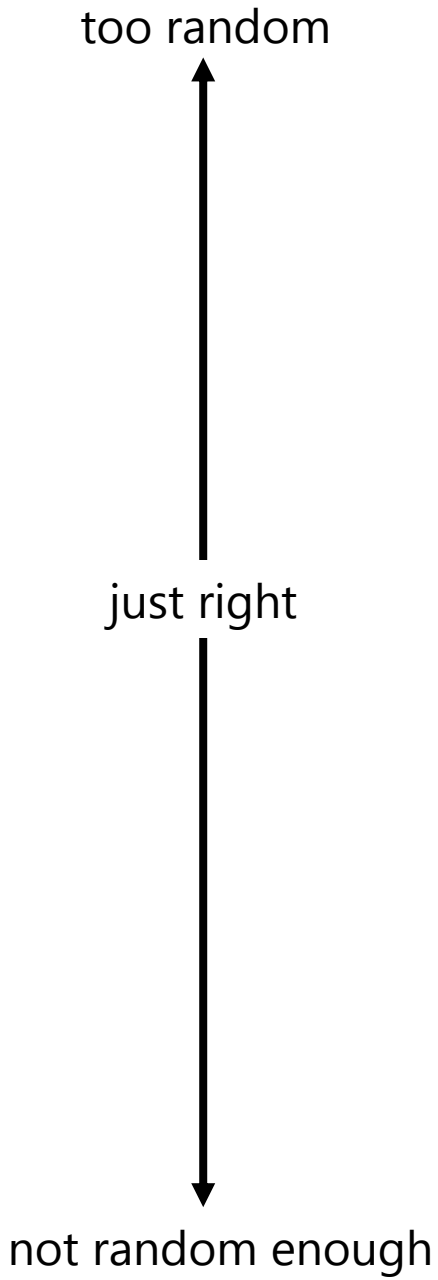
*The 17th International Workshop of Search-Based and Fuzz Testing*
*April 14th, 2024*

The Power of Fuzzing and Large Language Models is harnessing (just enough) randomness.


too random
just right
not random enough

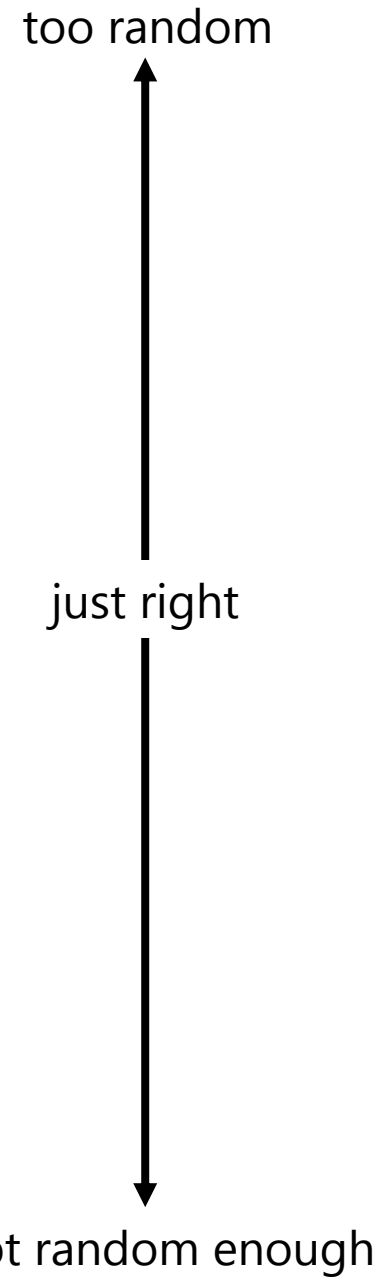
Random fuzzing

```
$ head -n 1000 /dev/uranddom > ./djpeg
```



Random fuzzing

```
$ head -n 1000 /dev/uranddom > ./djpeg
```




Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```

Random fuzzing

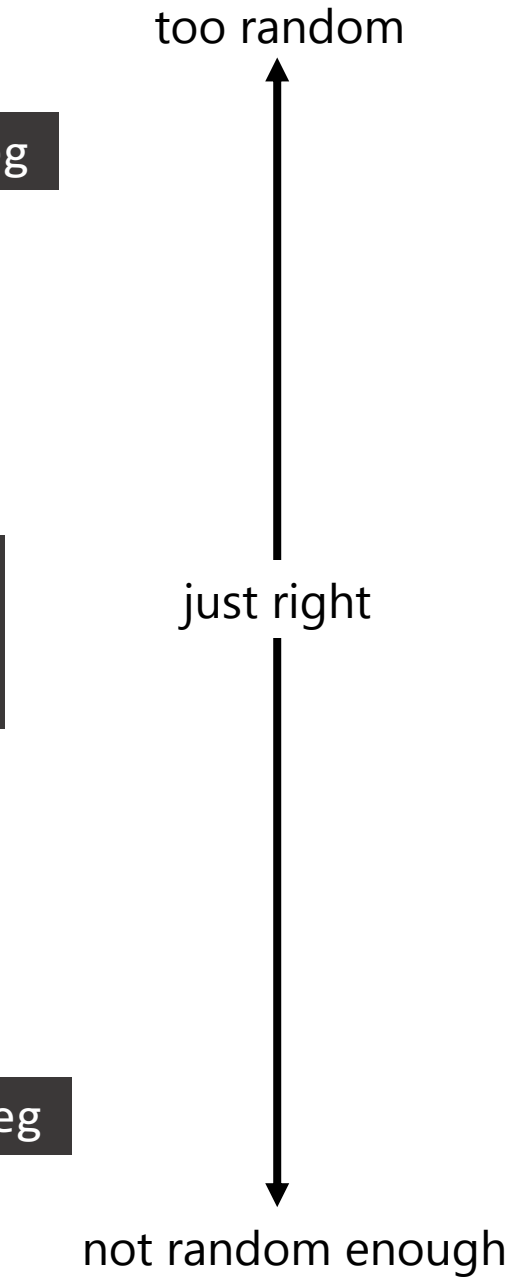
```
$ head -n 1000 /dev/uranddom > ./djpeg
```

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```



too random

Random fuzzing

```
$ head -n 1000 /dev/uranddom > ./djpeg
```

# Pulling JPEGs out of thin air

https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

...

In essence, I created a text file containing just "hello" and asked the fuzzer to keep feeding it to a program that expects a JPEG image

...

The first image, hit after about six hours on an 8-core system, looks very unassuming: it's a blank grayscale image, 3 pixels wide and 784 pixels tall. But the moment it is discovered, the fuzzer starts using the

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```

not random enough

Random fuzzing

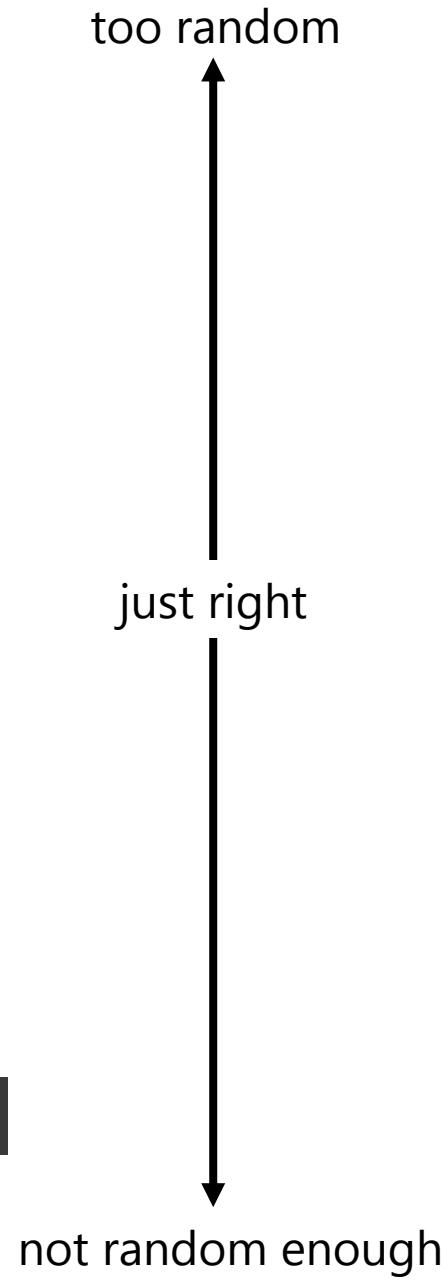
```
$ head -n 1000 /dev/uranddom > ./djpeg
```

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```



Random fuzzing

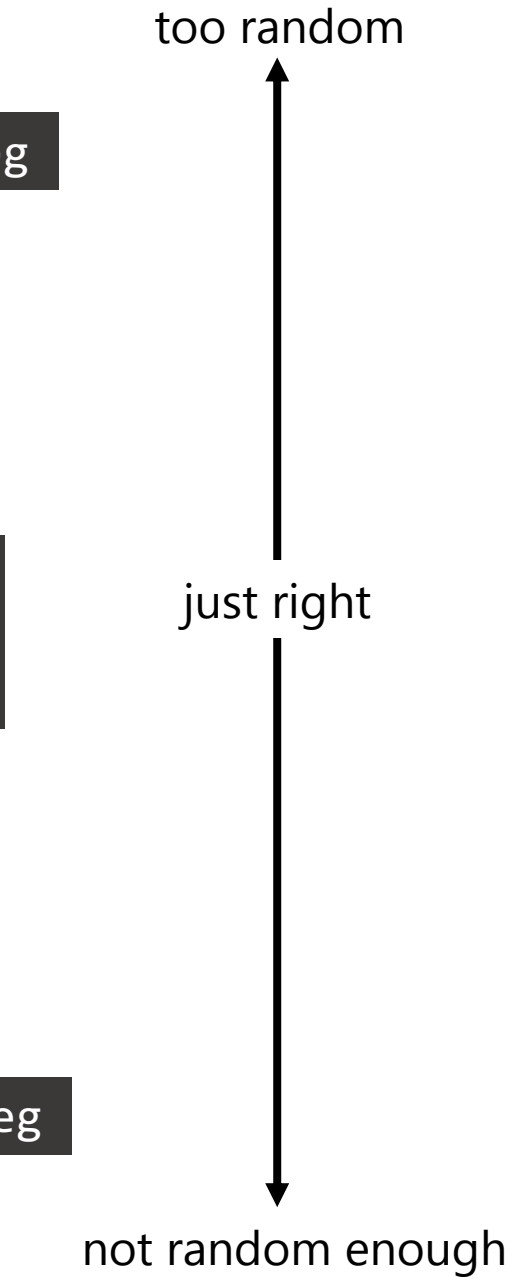
```
$ head -n 1000 /dev/uranddom > ./djpeg
```

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```




Random strings

XuaVxUMXaJXbxDWmATIUfwwTP

too random

Random fuzzing

```
$ head -n 1000 /dev/uranddom > ./djpeg
```

Random strings

XuaVxUMXaJXbxDWmATIUfwwTP

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

just right

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```

Existing Text

To be or not to be, that is the question:
Whether ‘tis nobler in the mind to suffer

not random enough

too random

Random fuzzing

```
$ head -n 1000 /dev/uranddom > ./djpeg
```

Random strings

XuaVxUMXaJXbxDWmATIUfwwTP

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

just right

Large Language Models

Throughout this talk, we'll explore the symbiotic relationship between fuzzing and LLMs, examining how these two technologies complement each other…

Running with Existing Inputs

```
$ cat JPEG_example_flower.jpg > ./djpeg
```

Existing Text

To be or not to be, that is the question:
Whether ‘tis nobler in the mind to suffer

not random enough

# harnessing (just enough) randomness

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

Large Language Models

Throughout this talk, we'll explore the symbiotic relationship between fuzzing and LLMs, examining how these two technologies complement each other…

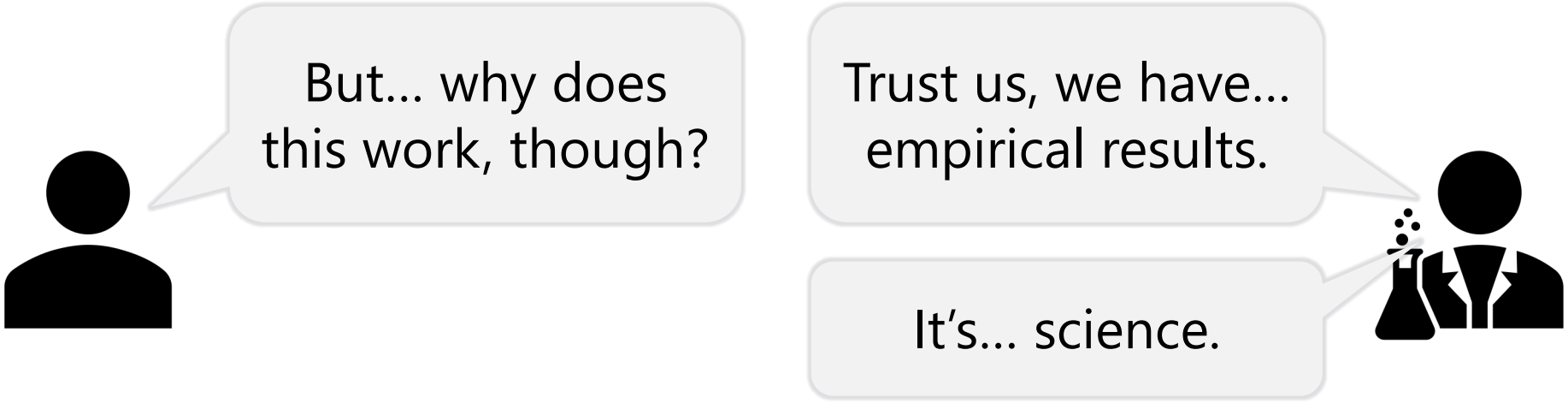
enabled by clever algorithms
+ performant implementations

Coverage-Guided Fuzzing

```
$ mkdir in
$ echo 'hello' > in/hello
$ ./afl-fuzz -i in -o out. ./djpeg
```

Large Language Models

Throughout this talk, we'll explore the symbiotic relationship between fuzzing and LLMs, examining how these two technologies complement each other…



# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

# ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

# Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

**Analyse:** Lemieux and Sen. FairFuzz:... ASE '18. https://doi.org/10.1145/3238147.3238176

## ML is not Magic

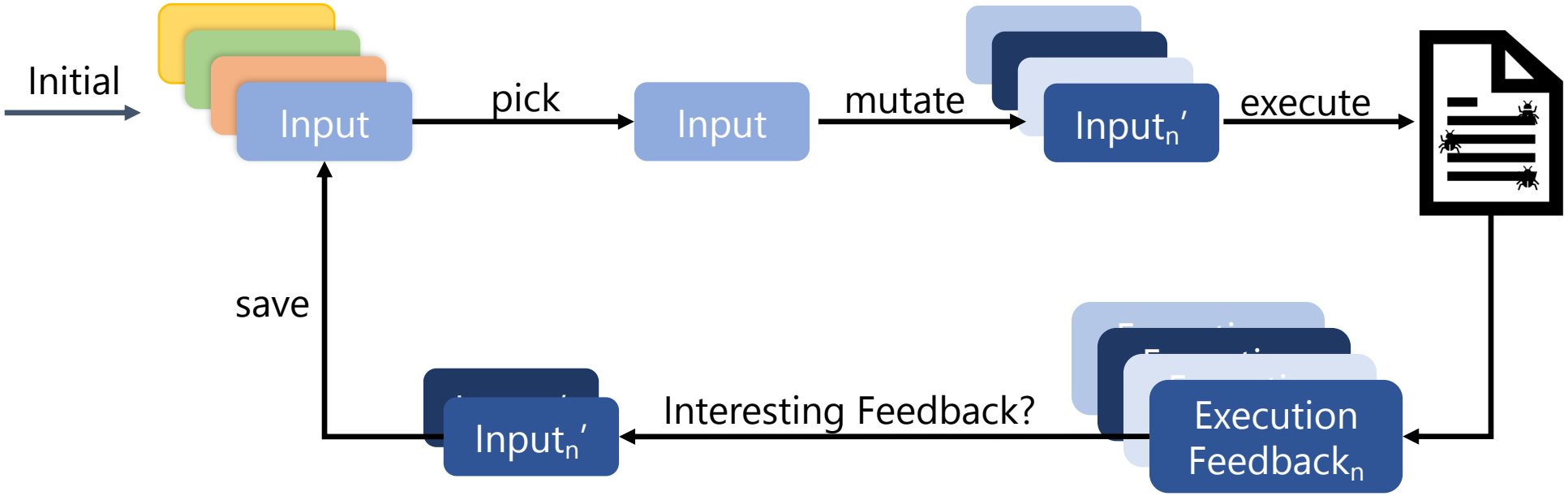
Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

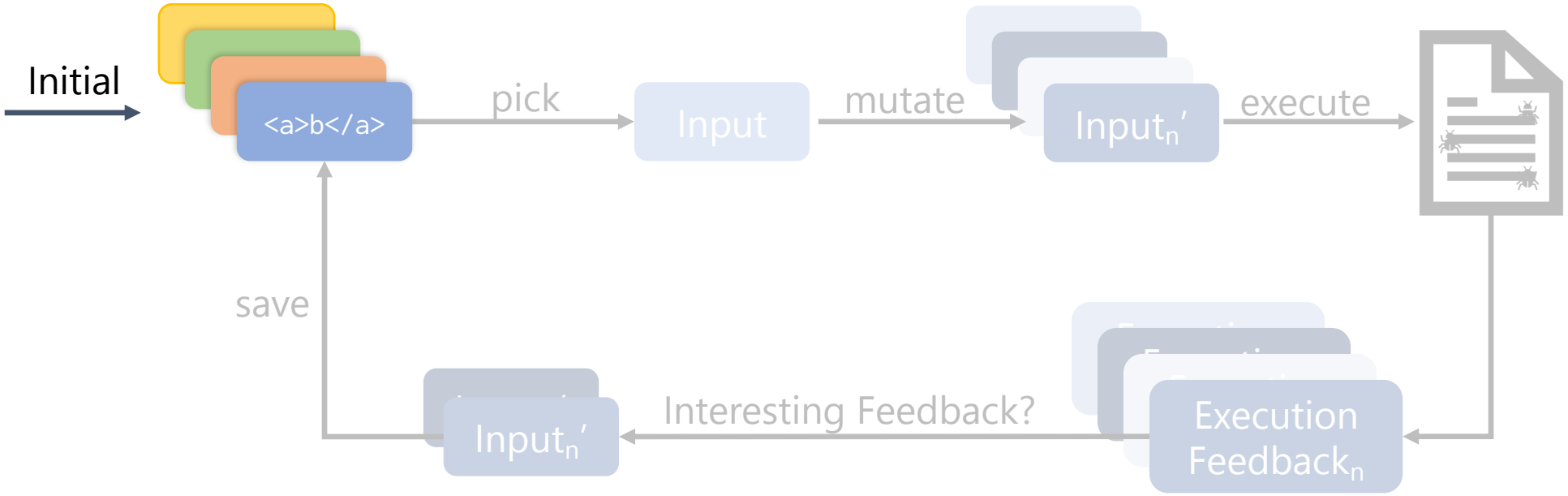
...but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

# Coverage-Guided Fuzzing

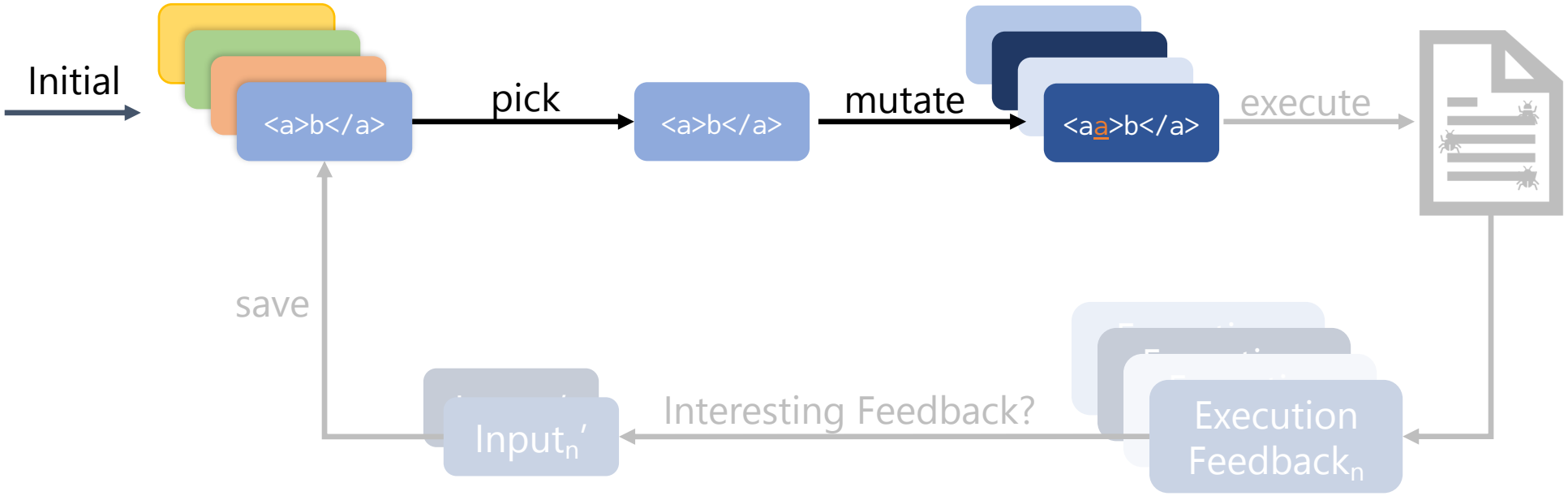
(in particular, AFL; similar to libFuzzer, honggfuzz…)



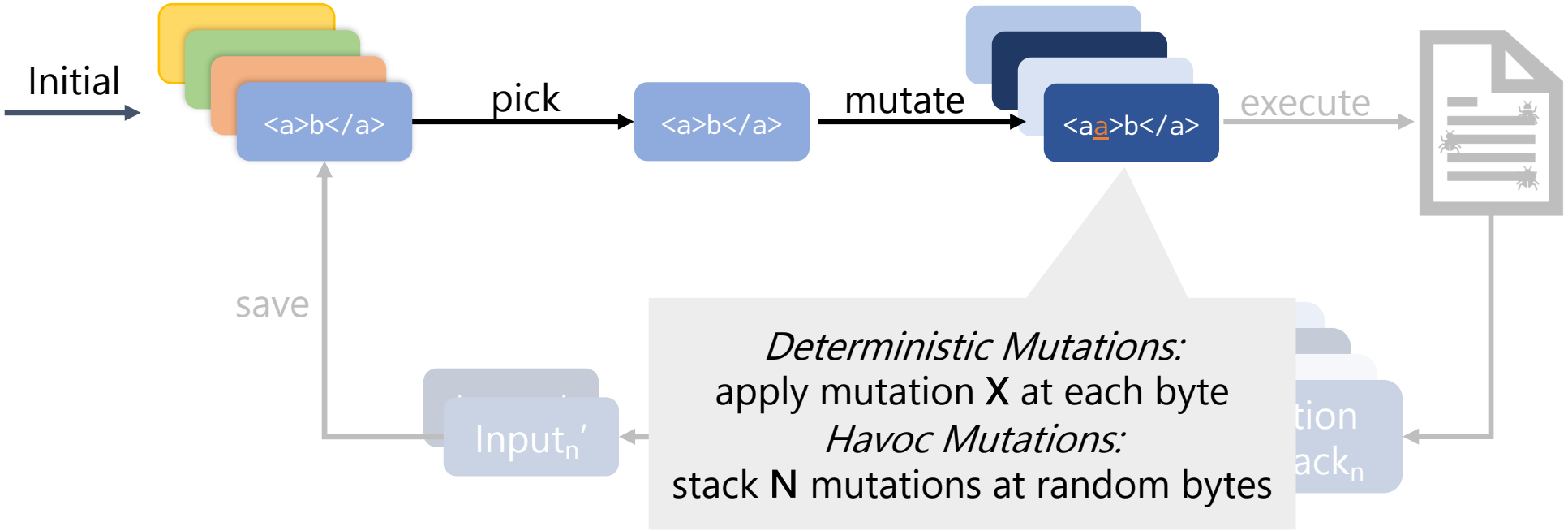
# Coverage-Guided Fuzzing



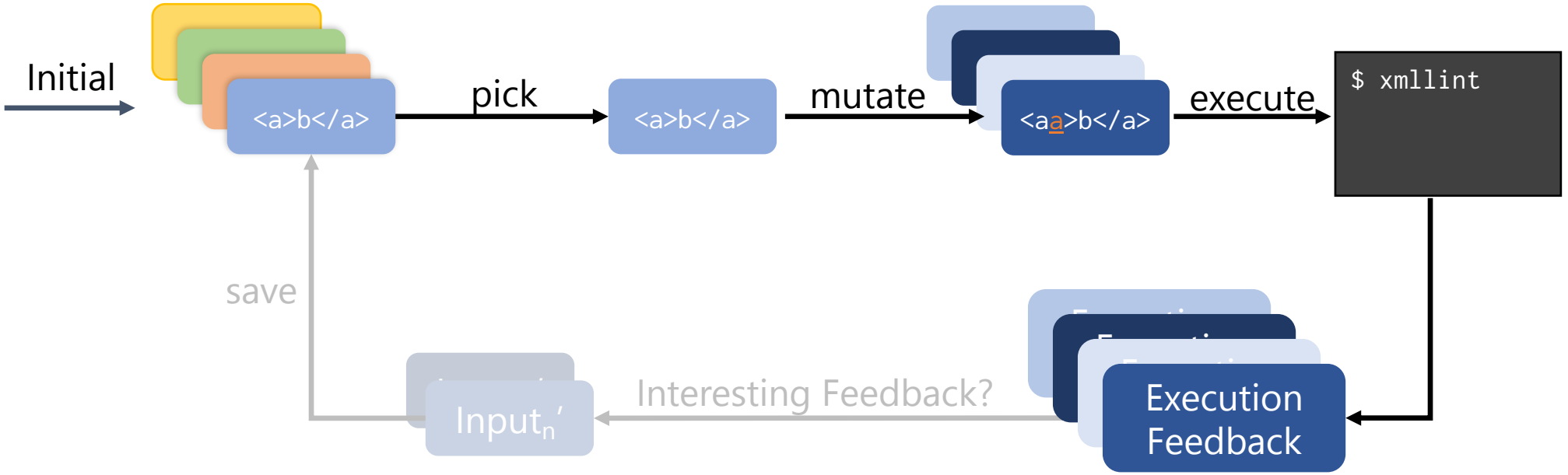
# Coverage-Guided Fuzzing



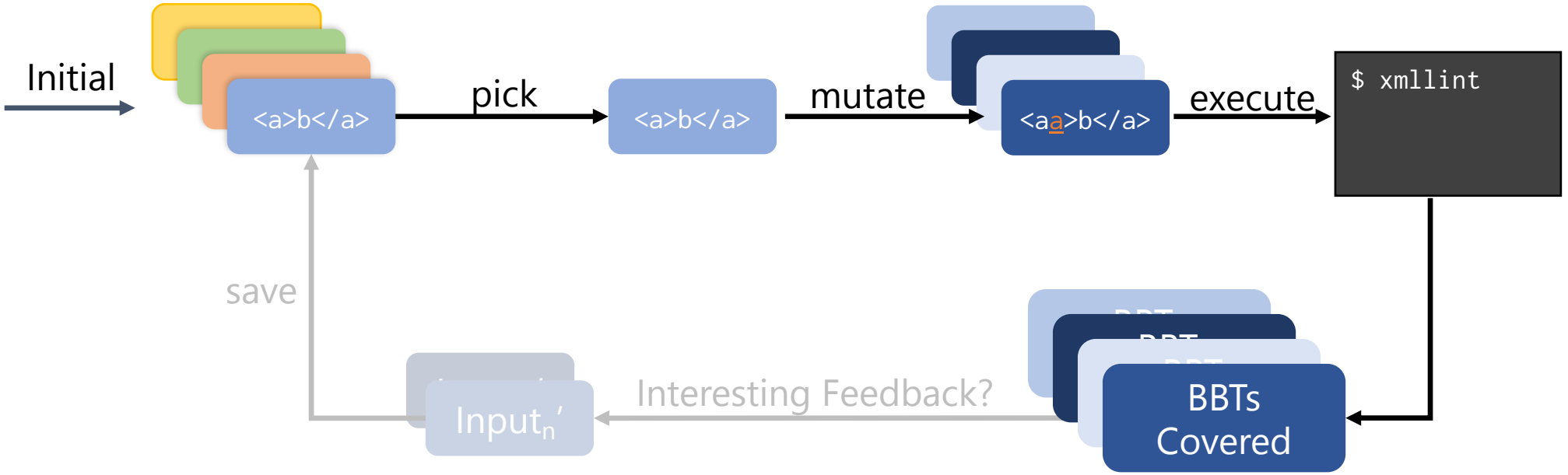
# Coverage-Guided Fuzzing



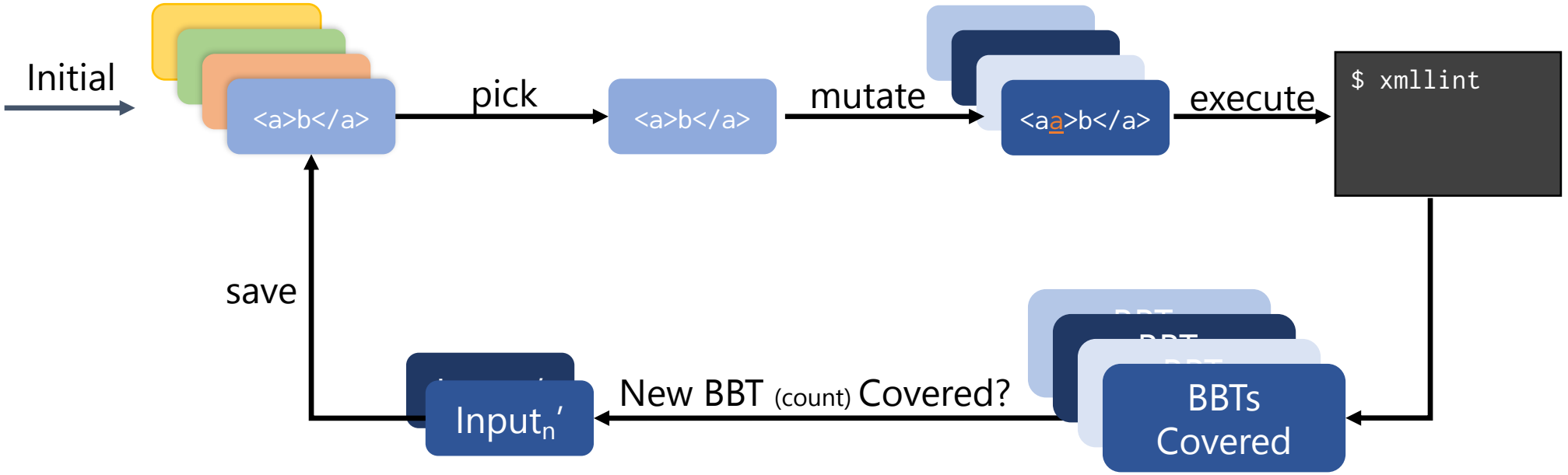
# Coverage-Guided Fuzzing



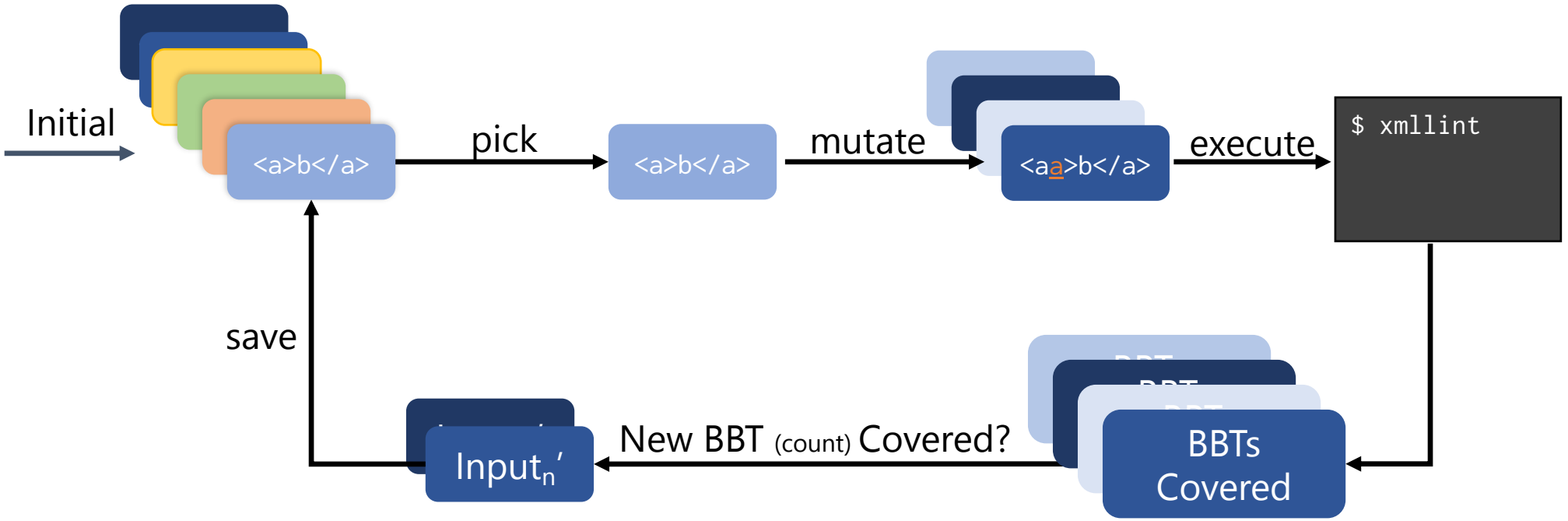
# Coverage-Guided Fuzzing



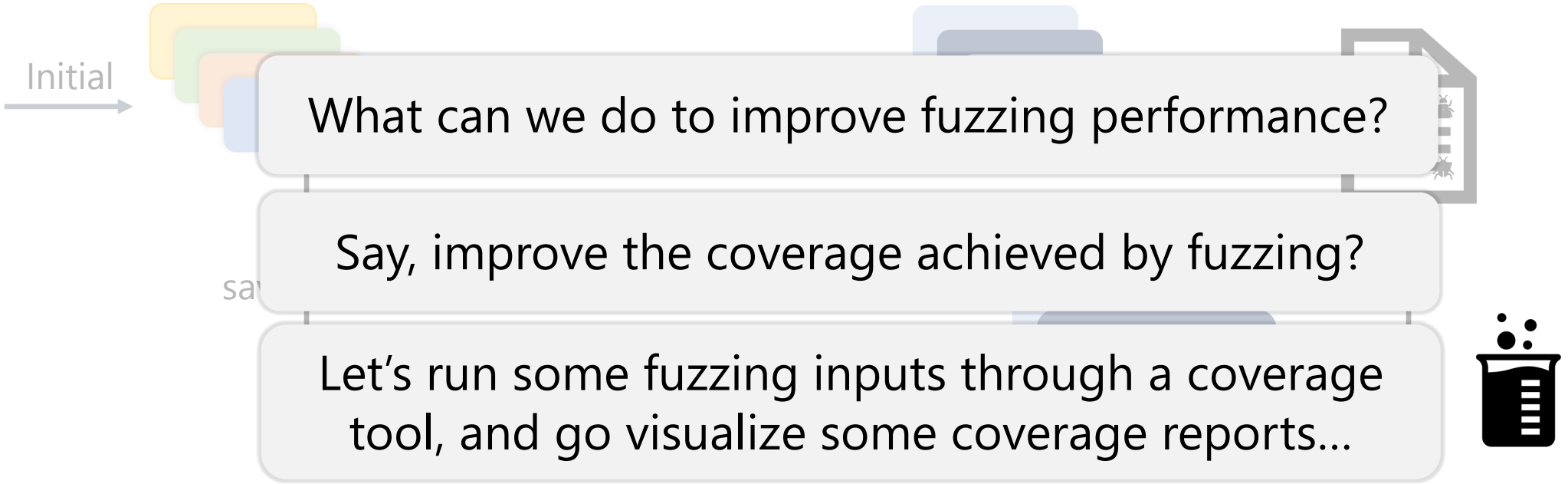
# Coverage-Guided Fuzzing



# Coverage-Guided Fuzzing



# Origins of FairFuzz



# Observation: Coverage is Uneven

Easy to cover (357k hits)

```
357456 : namePush(xmlParserCtxtPtr ctxt, const xmlChar * value)
       : {
357456 :     if (ctxt == NULL) return (-1);
       :
357456 :     if (ctxt->nameNr >= ctxt->nameMax) {
       :         const xmlChar * *tmp;
  7909 :         tmp = (const xmlChar * *) xmlRealloc((xmlChar * *)ctxt
  7909 :                                              ctxt->nameMax * 2 *
       :                                              sizeof(ctxt->nameTab[0]));
  7909 :         if (tmp == NULL) {
     0 :             goto mem_error;
       :         }
  7909 :         ctxt->nameTab = tmp;
  7909 :         ctxt->nameMax *= 2;
       :     }
```

Hard to cover (18 hits)

```
    : void
   26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
    :     const xmlChar *elemName;
    :     const xmlChar *attrName;
    :     xmlEnumerationPtr tree;
    :
   26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
    :
    :
    :
   18 :         xmlParserInputPtr input = ctxt->input;
    :
   18 :         SKIP(9);
```

```
    6 :                 break;
      :
    0 :             def = xmlParseDefaultDecl(ctxt, &defaultValue);
    0 :             if (def <= 0) {
    0 :                 if (defaultValue != NULL)
    0 :                     xmlFree(defaultValue);
    0 :                 if (tree != NULL)
    0 :                     xmlFreeEnumeration(tree);
    0 :                 break;
      :             }
```

Result: lots of uncovered code

# Origins of FairFuzz

Easy to cover

Hard to cover

We observed one example where uneven coverage → lots of uncovered code

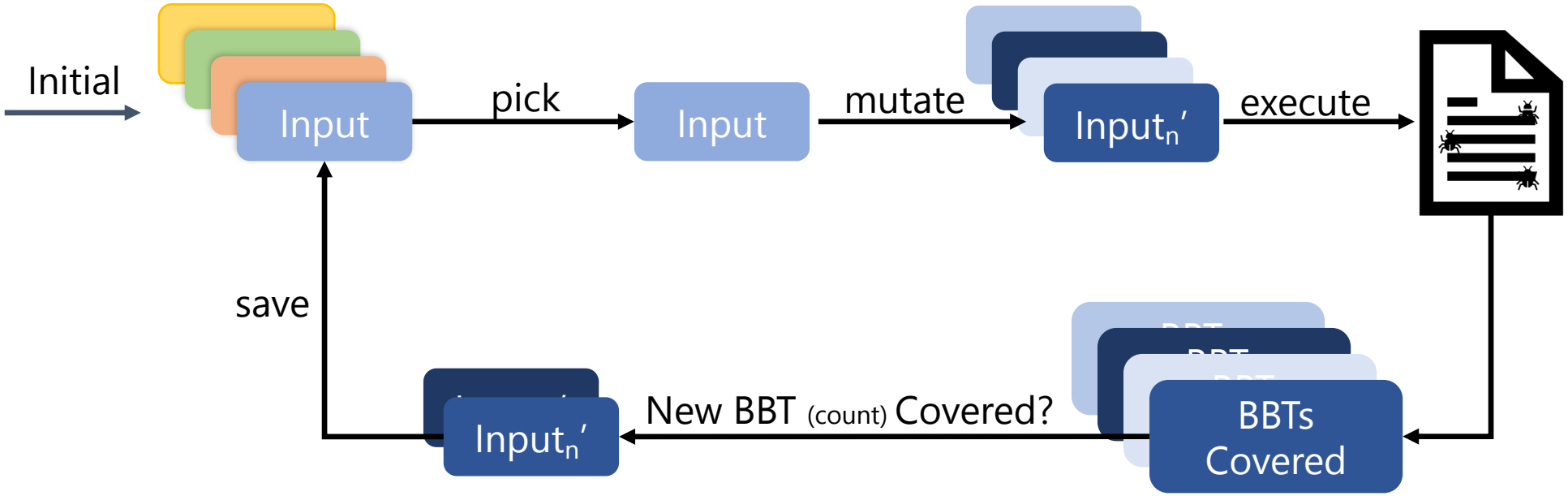
Q. Can we improve fuzzing coverage by making the coverage more even?

Q. Can we make fuzzing coverage more even using *just* the information AFL keeps track of?

Result: lots of uncovered code

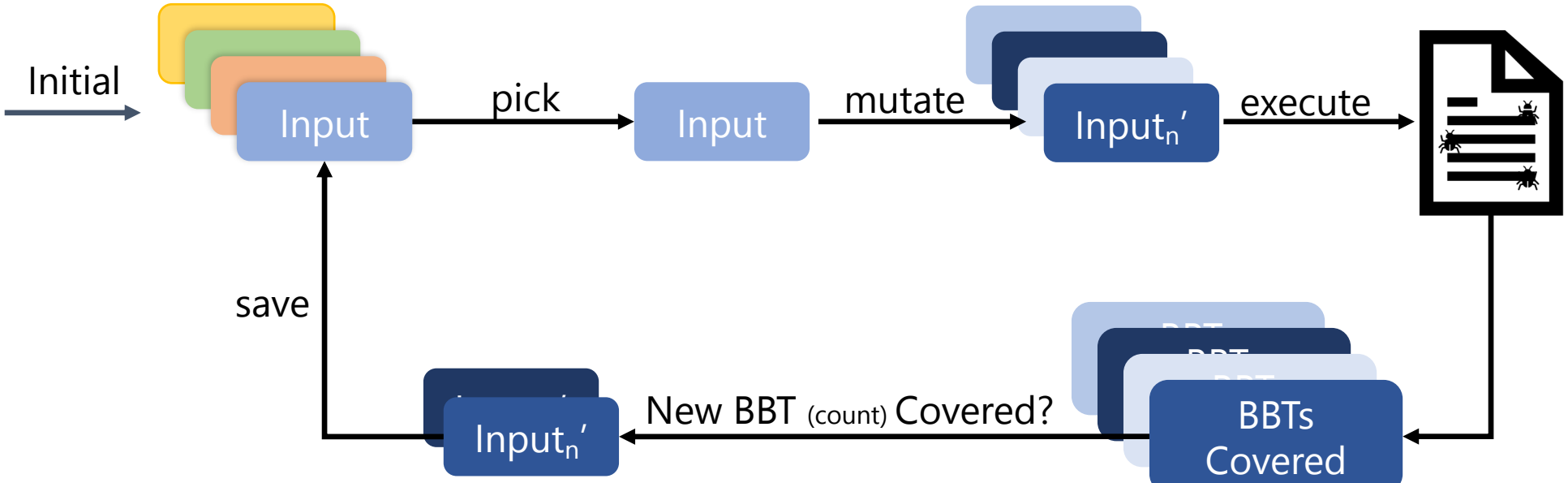
```
6 :                 break;
  :
0 :             def = xmlParseDefaultDecl(ctxt, &defaultValue);
0 :             if (def <= 0) {
0 :                 if (defaultValue != NULL)
0 :                     xmlFree(defaultValue);
0 :                 if (tree != NULL)
0 :                     xmlFreeEnumeration(tree);
0 :                 break;
  :             }
```

# FairFuzz Idea



# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

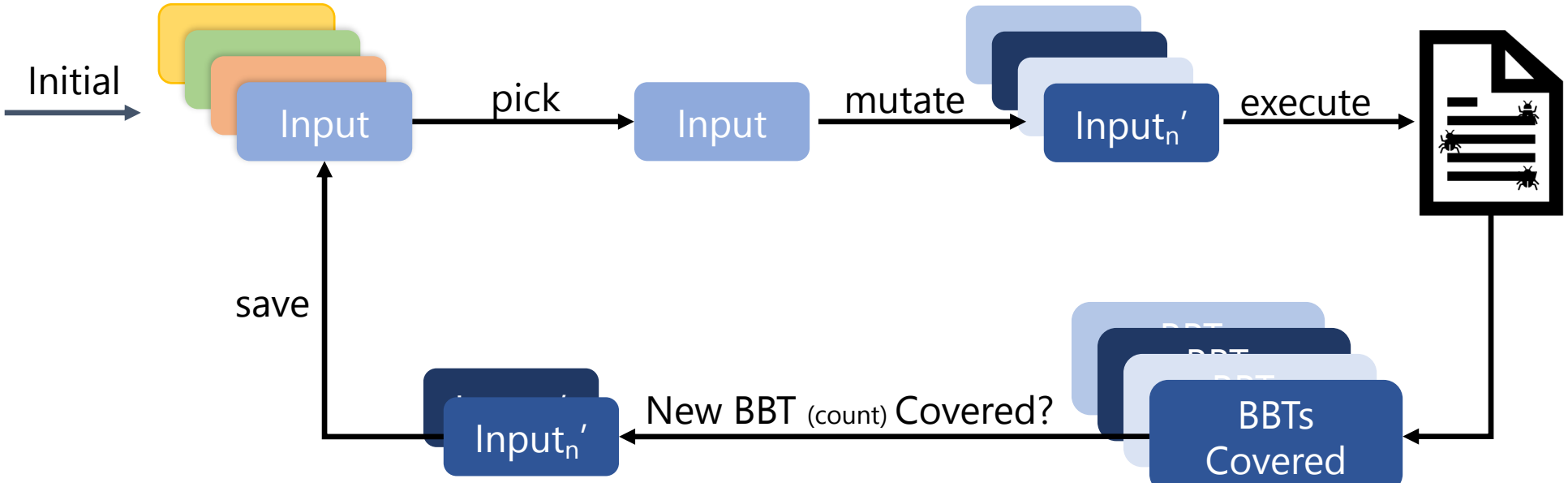



Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→26

# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

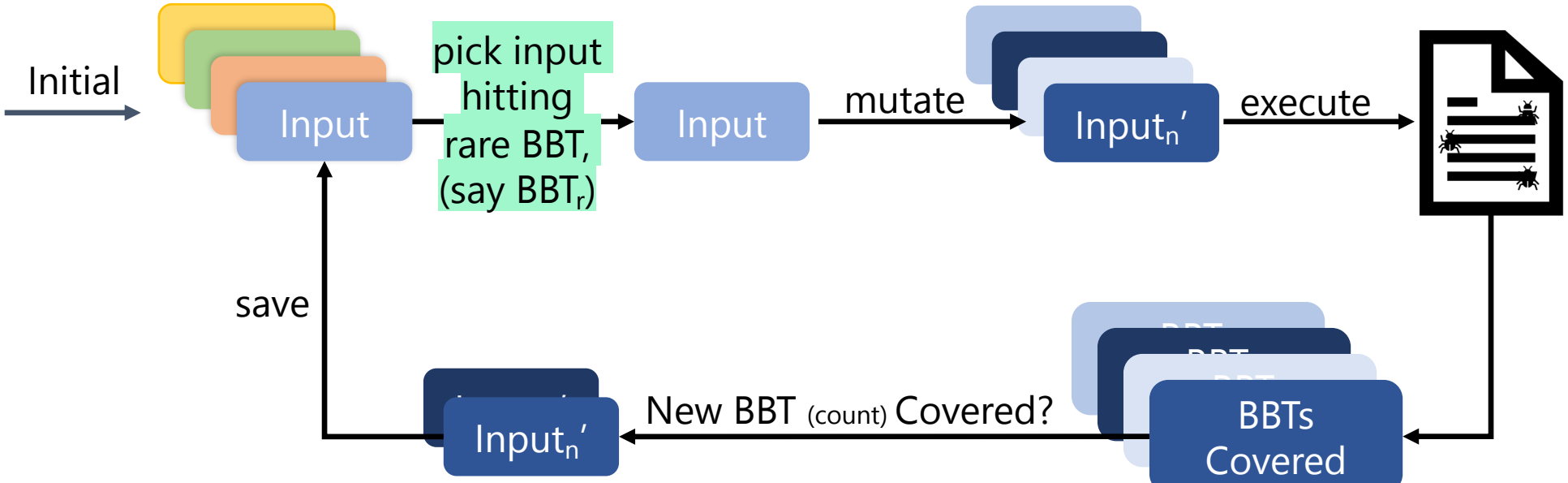



Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→26

# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

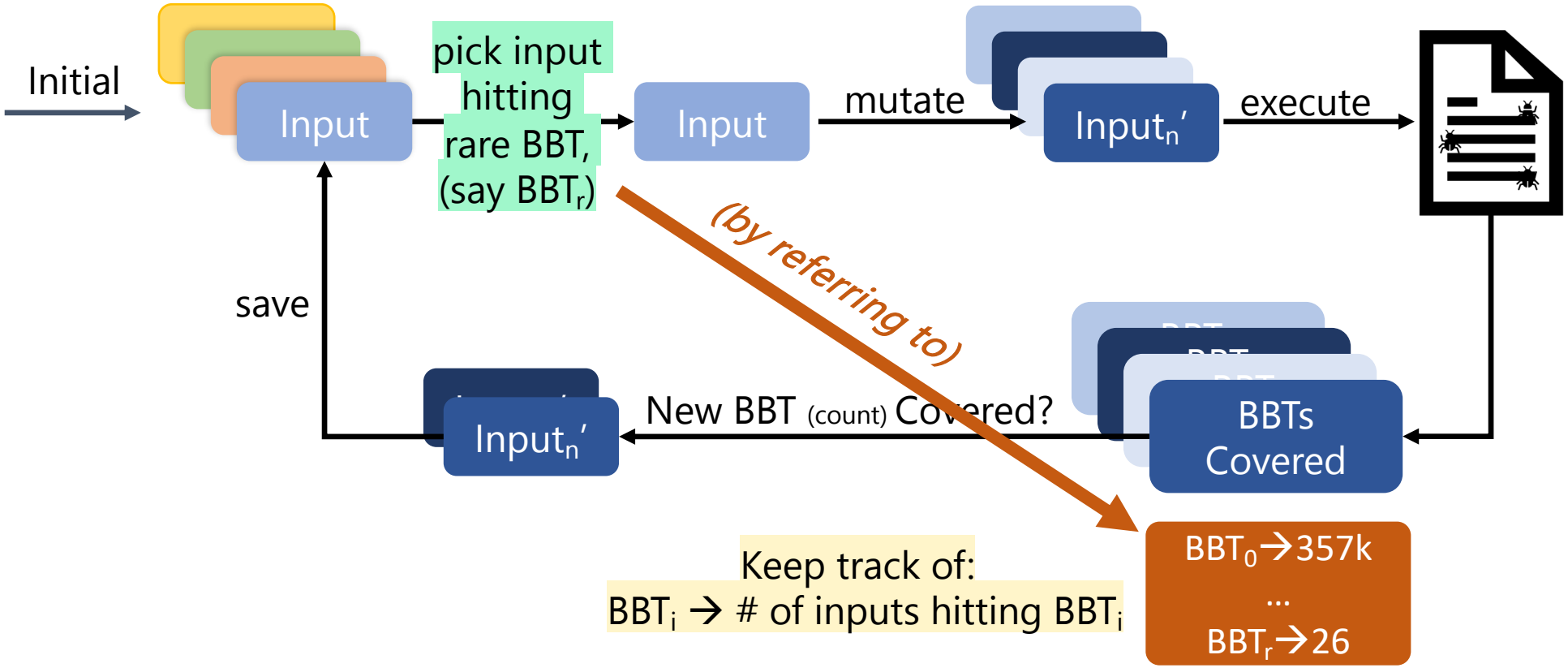



Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→26

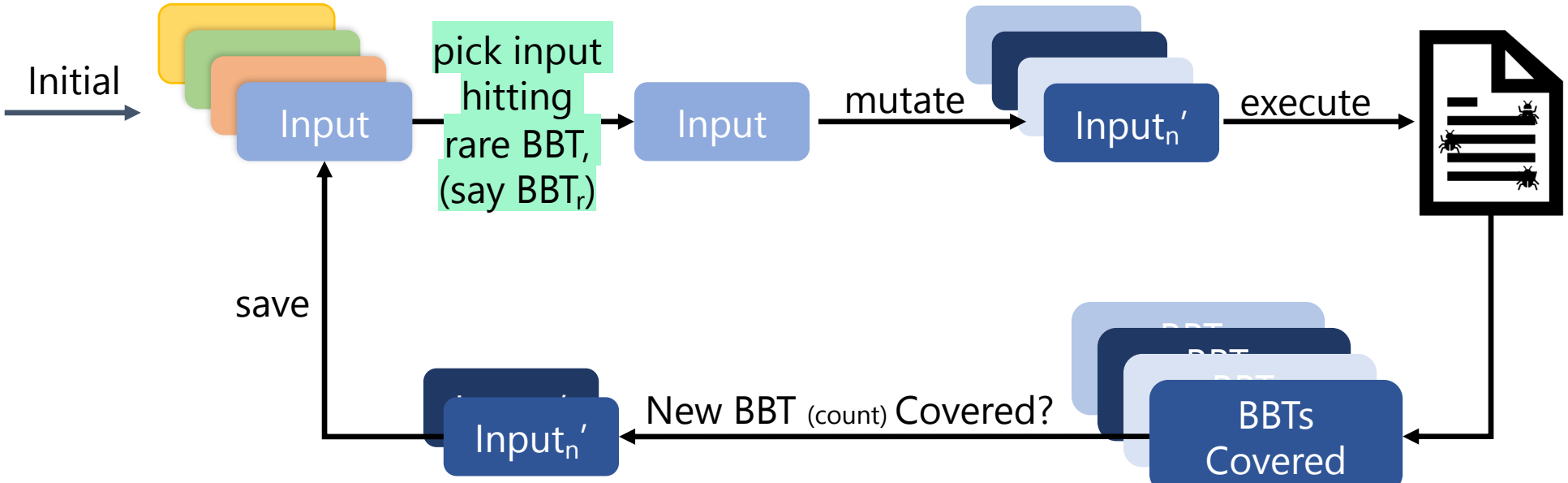
# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.



# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

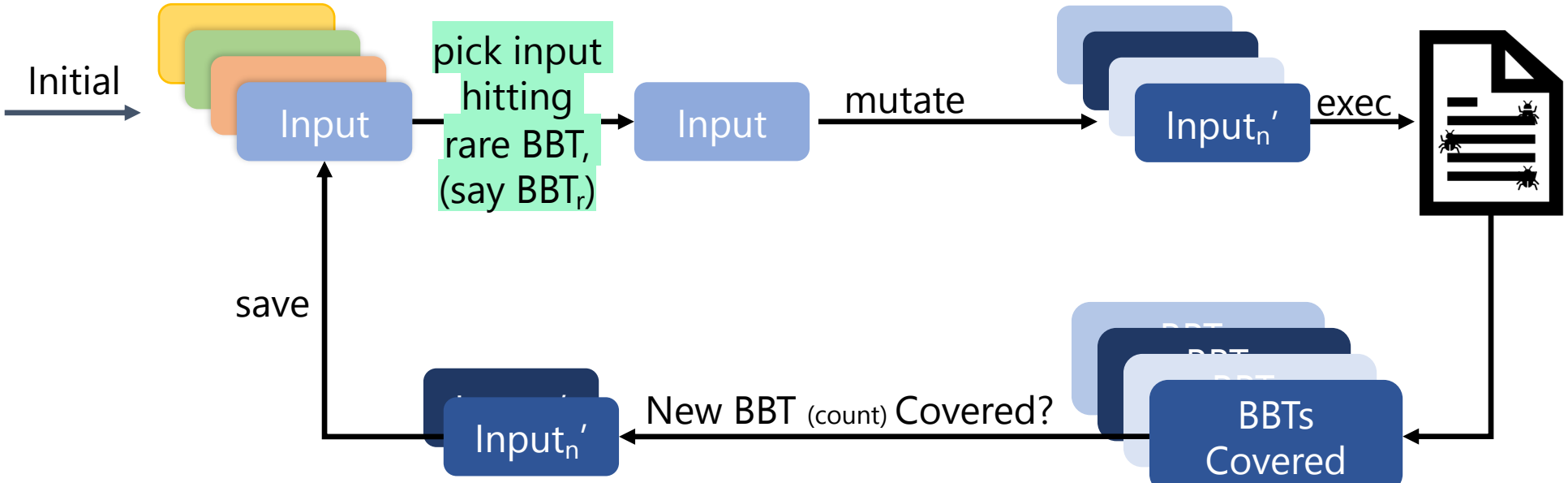



Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→26

# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

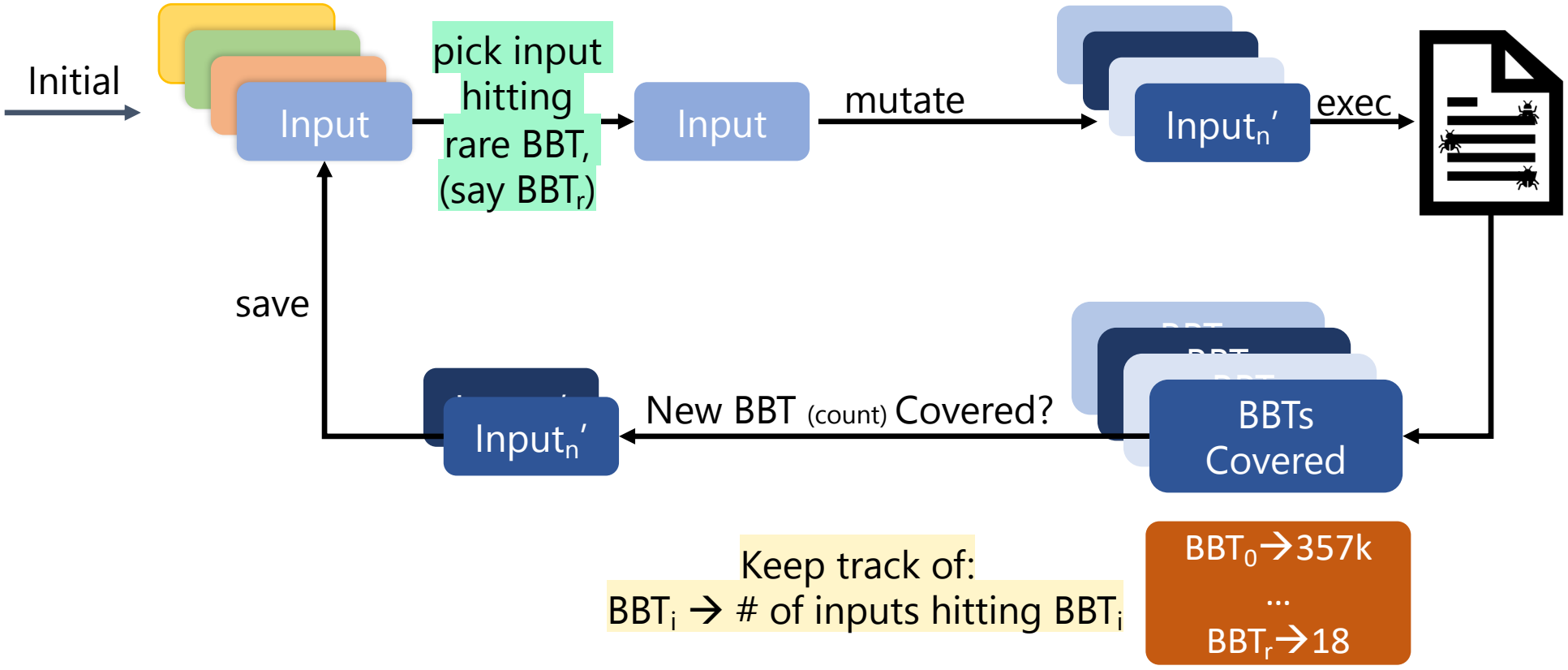



Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→18

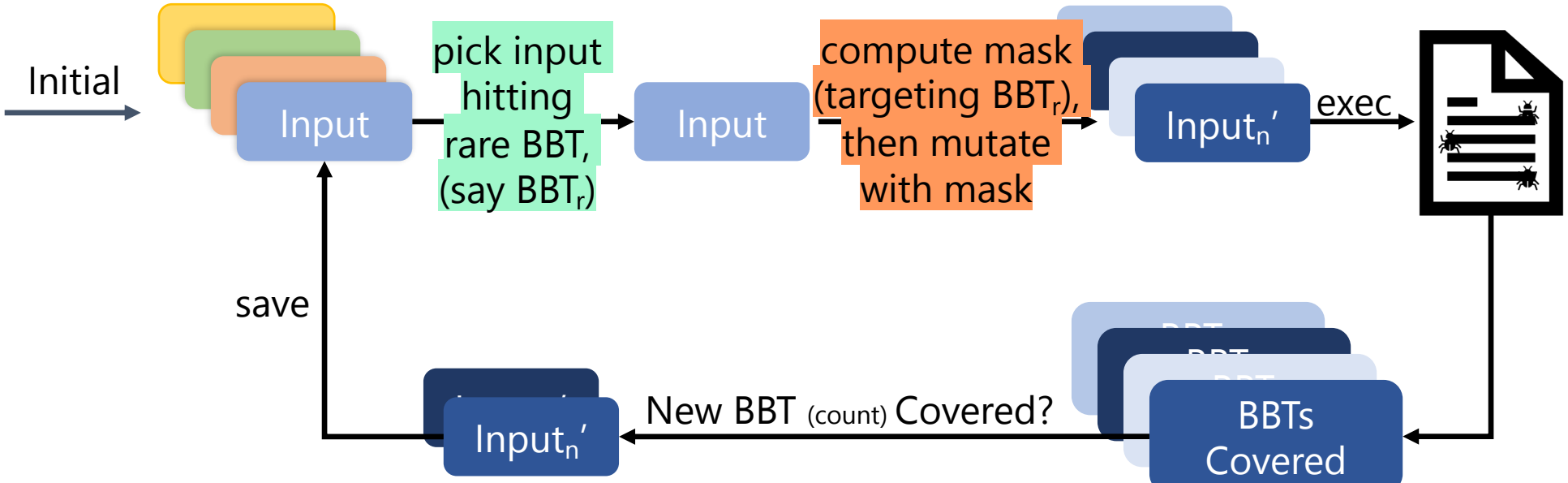
# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.



# FairFuzz Idea

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.




Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→18

1. Identify rarely-hit BBTs → 2. Mutate inputs hitting rare BBTs → 3. Mask mutations to hit rare BBTs with higher prob.

# Concrete Example

Easy to cover

```
357456 : namePush(xmlParserCtxtPtr ctxt, const xmlChar * value)
       : {
357456 :     if (ctxt == NULL) return (-1);
       :
357456 :     if (ctxt->nameNr >= ctxt->nameMax) {
       :         const xmlChar * *tmp;
  7909 :         tmp = (const xmlChar * *) xmlRealloc((xmlChar * *)ctxt
  7909 :                                              ctxt->nameMax * 2 *
       :                                              sizeof(ctxt->nameTab[0]));
  7909 :         if (tmp == NULL) {
     0 :             goto mem_error;
       :         }
  7909 :         ctxt->nameTab = tmp;
  7909 :         ctxt->nameMax *= 2;
       :     }
```

Hard to cover

```
    : void
   26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
    :     const xmlChar *elemName;
    :     const xmlChar *attrName;
    :     xmlEnumerationPtr tree;
    :
   26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
    :
    :
    :
    :
   18 :         xmlParserInputPtr input = ctxt->input;
    :
   18 :         SKIP(9);
```

# Choosing a Rare BBT Target

$BBT_0$ → 357456

```
357456 : namePush(xmlParserCtxtPtr ctxt, const xmlChar * value)
       : {
357456 :     if (ctxt == NULL) return (-1);
       :
357456 :     if (ctxt->nameNr >= ctxt->nameMax) {
       :         const xmlChar * *tmp;
  7909 :         tmp = (const xmlChar * *) xmlRealloc((xmlChar * *)ctxt
  7909 :                                              ctxt->nameMax * 2 *
       :                                              sizeof(ctxt->nameTab[0]));
  7909 :         if (tmp == NULL) {
     0 :             goto mem_error;
       :         }
  7909 :         ctxt->nameTab = tmp;
  7909 :         ctxt->nameMax *= 2;
       :     }
```

Hard to cover

```
    : void
 26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
    :     const xmlChar *elemName;
    :     const xmlChar *attrName;
    :     xmlEnumerationPtr tree;
    :
 26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
    :
    :
    :
    :
 18 :         xmlParserInputPtr input = ctxt->input;
    :
 18 :         SKIP(9);
```

# Choosing a Rare BBT Target

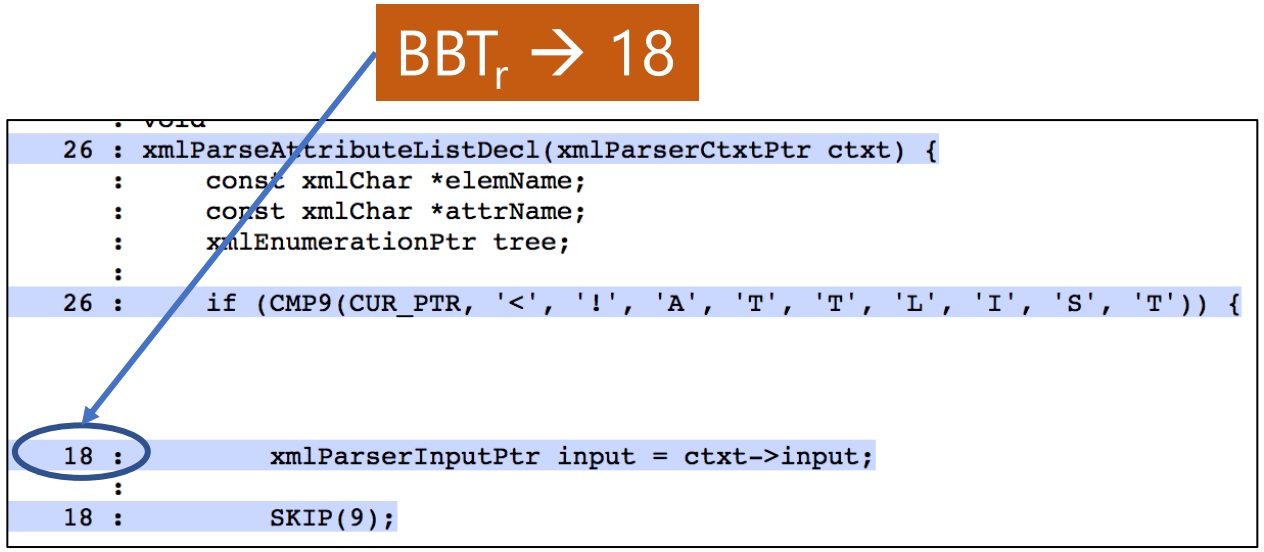
$BBT_0$ → 357456

```
357456 : namePush(xmlParserCtxtPtr ctxt, const xmlChar * value)
       : {
357456 :     if (ctxt == NULL) return (-1);
       :
357456 :     if (ctxt->nameNr >= ctxt->nameMax) {
       :         const xmlChar * *tmp;
  7909 :         tmp = (const xmlChar * *) xmlRealloc((xmlChar * *)ctxt
  7909 :                                              ctxt->nameMax * 2 *
       :                                              sizeof(ctxt->nameTab[0]));
  7909 :         if (tmp == NULL) {
     0 :             goto mem_error;
       :         }
  7909 :         ctxt->nameTab = tmp;
  7909 :         ctxt->nameMax *= 2;
       :     }
```

$BBT_r$ → 18

```
    : void
   26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
      :     const xmlChar *elemName;
      :     const xmlChar *attrName;
      :     xmlEnumerationPtr tree;
      :
   26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
      :
      :
      :
      :
   18 :         xmlParserInputPtr input = ctxt->input;
      :
   18 :         SKIP(9);
```

# Choosing a Rare BBT Target



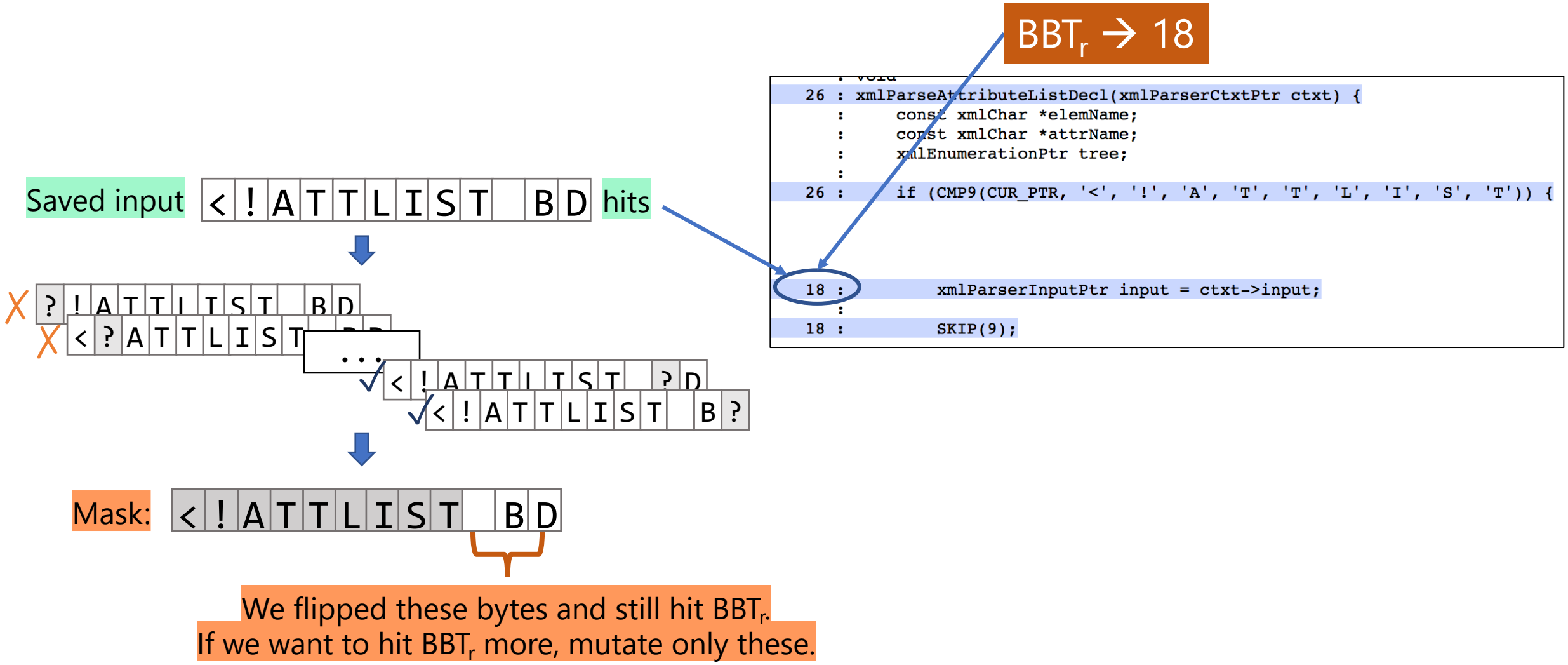
# Pick Input Hitting Rare BBT

Saved input | < | ! | A | T | T | L | I | S | T | | B | D | hits

$BBT_r$ → 18

```
    :
 26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
    :     const xmlChar *elemName;
    :     const xmlChar *attrName;
    :     xmlEnumerationPtr tree;
    :
 26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
    :
    :
    :
 18 :         xmlParserInputPtr input = ctxt->input;
    :
 18 :         SKIP(9);
```

# Compute Mask




We flipped these bytes and still hit BBT$_r$.
If we want to hit BBT$_r$ more, mutate only these.

# Mutate with Mask

Saved input <!ATTLIST BD hits

Mask: <!ATTLIST BD

Mutate at locations outside mask

<!ATTLIST0BD

<!ATTLIST 0D

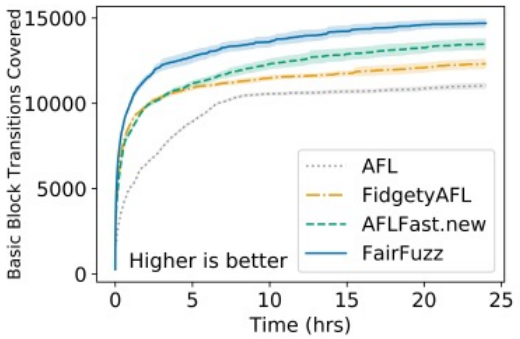
...

<!ATTLIST IST

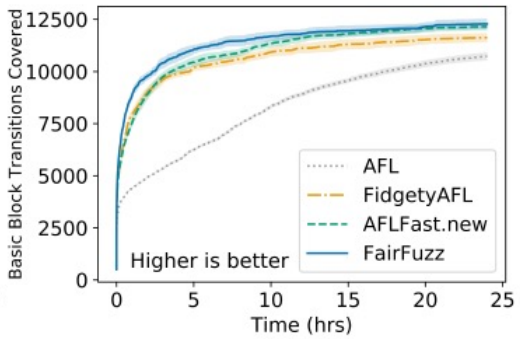
$BBT_r$ → 18

```
26 : xmlParseAttributeListDecl(xmlParserCtxtPtr ctxt) {
   :     const xmlChar *elemName;
   :     const xmlChar *attrName;
   :     xmlEnumerationPtr tree;
   :
26 :     if (CMP9(CUR_PTR, '<', '!', 'A', 'T', 'T', 'L', 'I', 'S', 'T')) {
18 :         xmlParserInputPtr input = ctxt->input;
   :
18 :         SKIP(9);
```

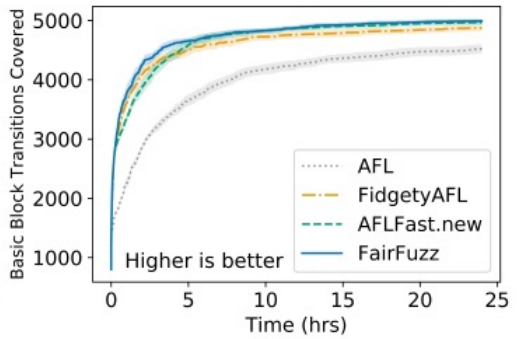
# Branch Coverage Over Time (ASE'18)



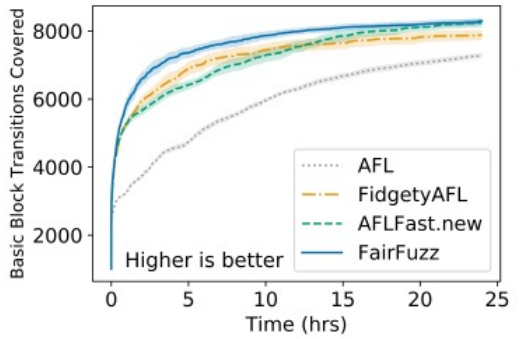

(a) tcpdump



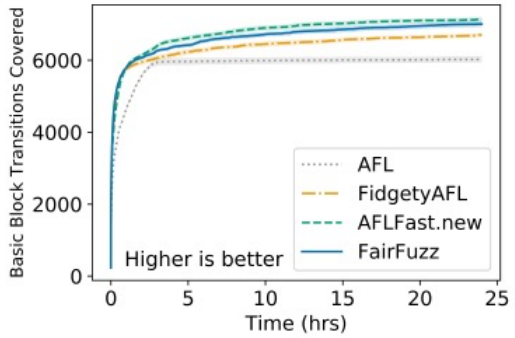

(b) readelf



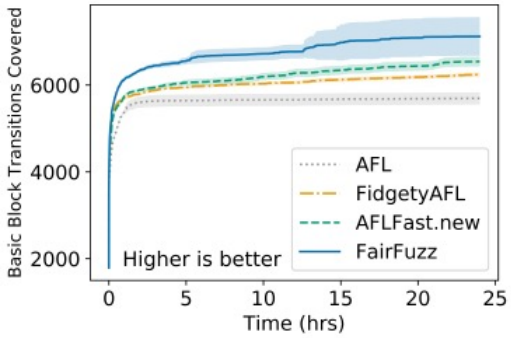

(c) nm



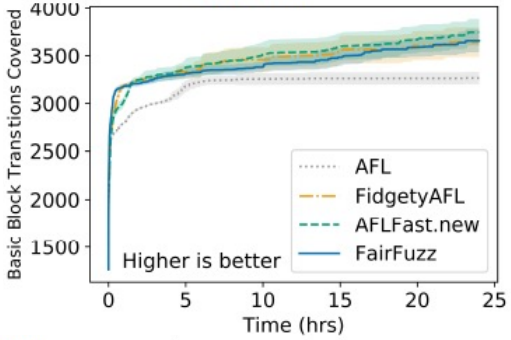

(d) objdump



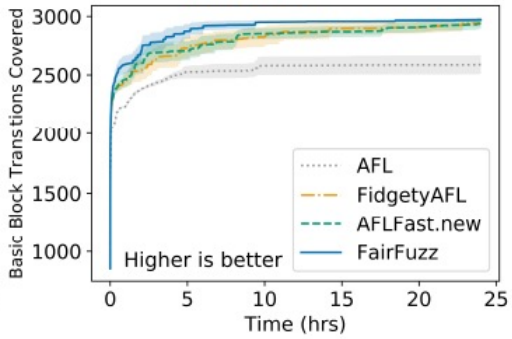

(e) c++filt



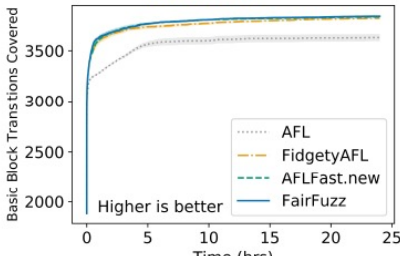

(f) xmllint




(h) djpeg

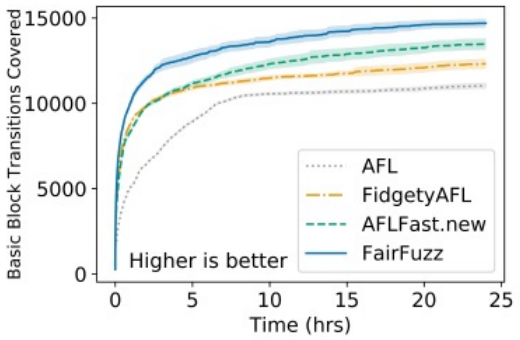



(i) readpng

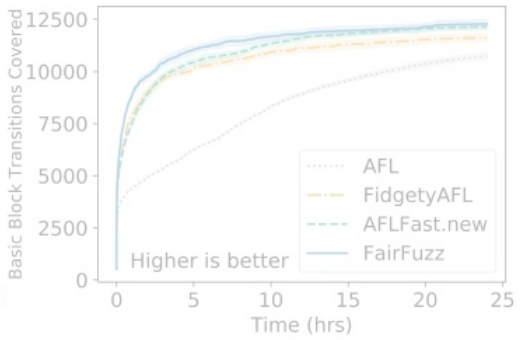



(g) mutool draw

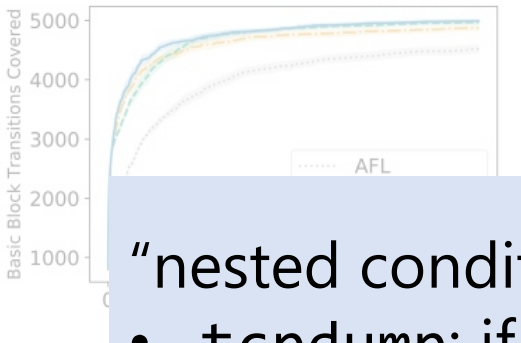
# Branch Coverage Over Time (ASE'18)

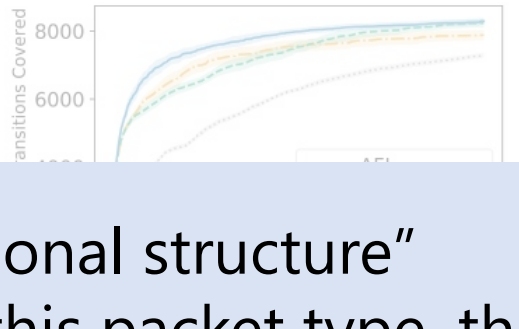


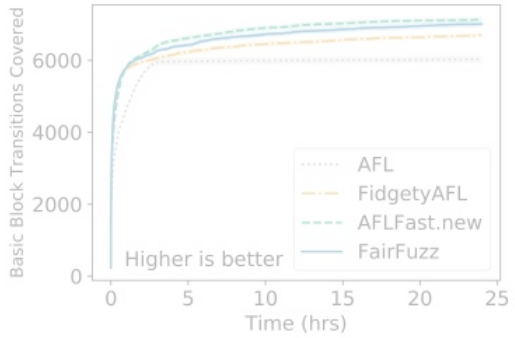
(a) tcpdump



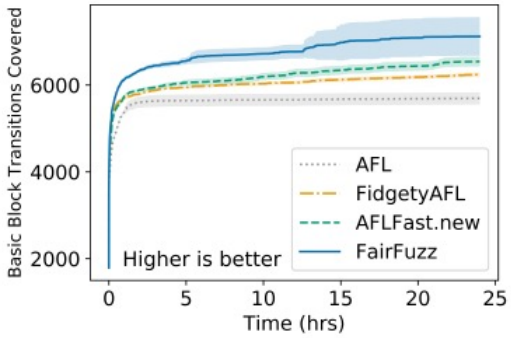
(b) readelf



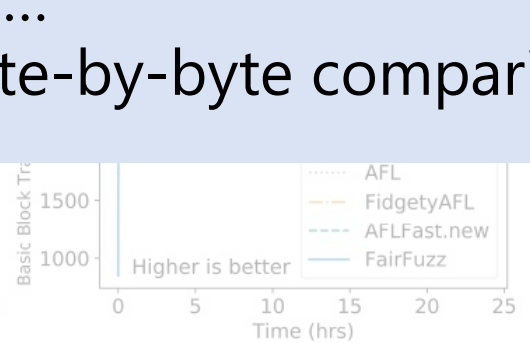



(e) c++filt

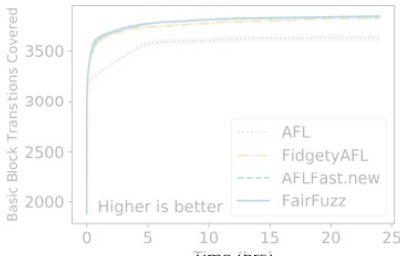


(f) xmllint

(h) djpeg



(i) readpng



(g) mutool draw

"nested conditional structure"

- `tcpdump`: if this packet type, then if has this field…
- `xmllint`: byte-by-byte comparisons

# Mask Successful?

Goal of mask: make mutants hit target (rare) BBT.

Look at % of mutants hitting target BBT:

| | Deterministic Mutants | | Havoc Mutants | |
|---|---|---|---|---|
| | With Mask | Without Mask | With Mask | Without Mask |
| `xmllint` | 90.3% | 22.9% | 32.8% | 2.9% |
| `tcpdump` | 98.7% | 72.8% | 36.1% | 9.0% |
| `c++filt` | 96.6% | 14.8% | 34.4% | 1.1% |
| `readelf` | 99.7% | 78.2% | 55.5% | 11.4% |
| `readpng` | 97.8% | 39.0% | 24.0% | 2.4% |
| `objdump` | 99.2% | 66.7% | 46.2% | 7.6% |

# Mask Successful?

Goal of mask: make mutants hit target (rare) BBT.

Look at % of mutants hitting target BBT:

| | Deterministic Mutants | | Havoc Mutants | |
|---|---|---|---|---|
| | With Mask | Without Mask | With Mask | Without Mask |
| `xmllint` | 90.3% | 22.9% | 32.8% | 2.9% |
| `tcpdump` | 98.7% | 72.8% | 36.1% | 9.0% |
| `c++filt` | 96.6% | 14.8% | 34.4% | 1.1% |
| `readelf` | 99.7% | 78.2% | 55.5% | 11.4% |
| `readpng` | 97.8% | 39.0% | 24.0% | 2.4% |
| `objdump` | 99.2% | 66.7% | 46.2% | 7.6% |

# Mask Successful?

Goal of mask: make mutants hit target (rare) BBT.

Look at % of mutants hitting target BBT:

| | Deterministic Mutants | | Havoc Mutants | |
|---|---|---|---|---|
| | With Mask | Without Mask | With Mask | Without Mask |
| `xmllint` | 90.3% | 22.9% | 32.8% | 2.9% |
| `tcpdump` | 98.7% | 72.8% | 36.1% | 9.0% |
| `c++filt` | 96.6% | 14.8% | 34.4% | 1.1% |
| `readelf` | 99.7% | 78.2% | 55.5% | 11.4% |
| `readpng` | 97.8% | 39.0% | 24.0% | 2.4% |
| `objdump` | 99.2% | 66.7% | 46.2% | 7.6% |

# FuzzBench Evaluation (2020/09/07)

By avg. score

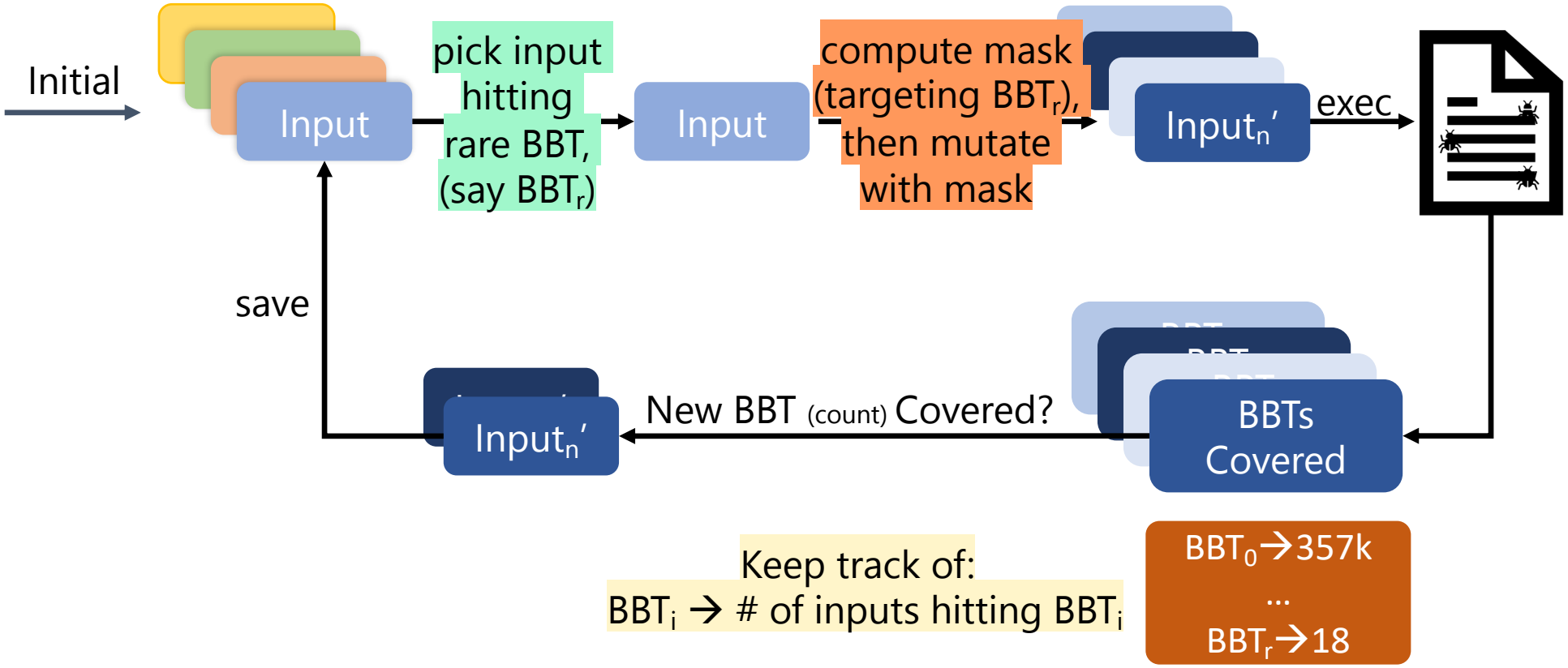
| | average normalized score |
|---|---|
| **fuzzer** | |
| **aflplusplus_optimal** | 98.61 |
| **honggfuzz** | 95.37 |
| **entropic** | 93.75 |
| **lafintel** | 91.53 |
| **libfuzzer** | 91.47 |
| **aflsmart** | 90.35 |
| **afl** | 89.89 |
| **mopt** | 89.60 |
| **aflfast** | 87.67 |
| **fairfuzz** | 84.74 |
| **eclipser** | 76.68 |

By avg. rank

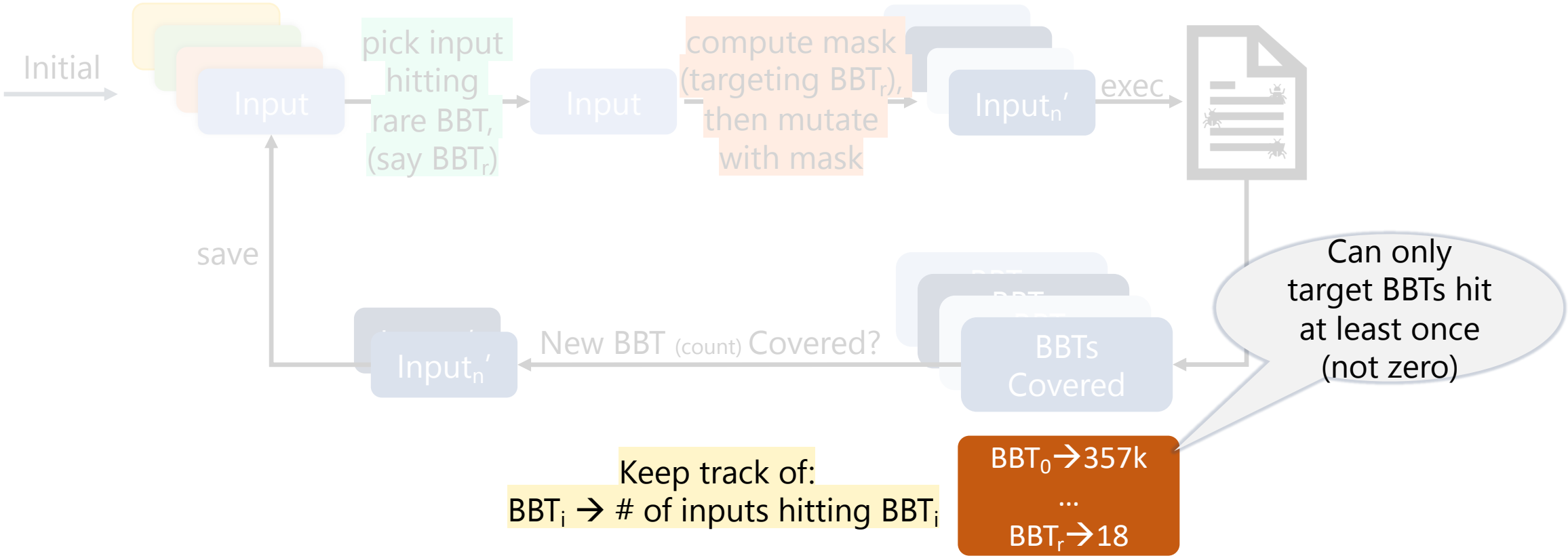
| | average rank |
|---|---|
| **fuzzer** | |
| **aflplusplus_optimal** | 2.48 |
| **afl** | 4.72 |
| **honggfuzz** | 4.85 |
| **aflsmart** | 4.95 |
| **entropic** | 5.10 |
| **mopt** | 5.40 |
| **lafintel** | 6.32 |
| **libfuzzer** | 6.70 |
| **fairfuzz** | 7.48 |
| **aflfast** | 7.55 |
| **eclipser** | 10.45 |

?

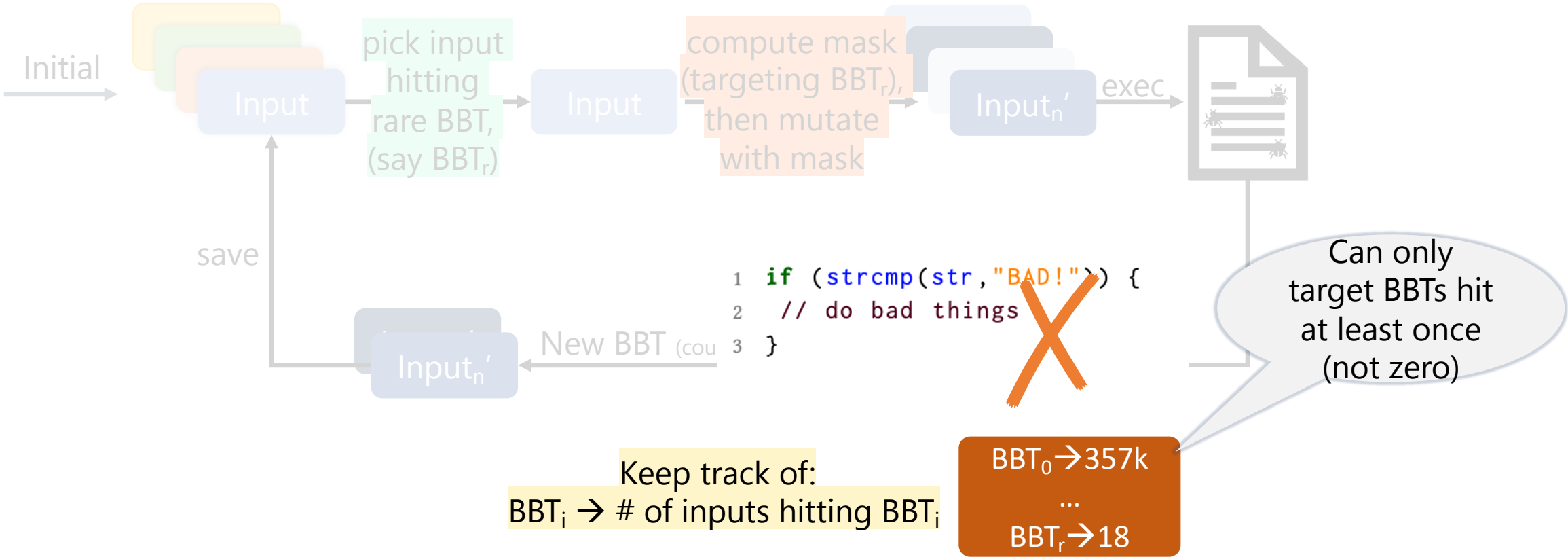
# Critically Analyzing FairFuzz



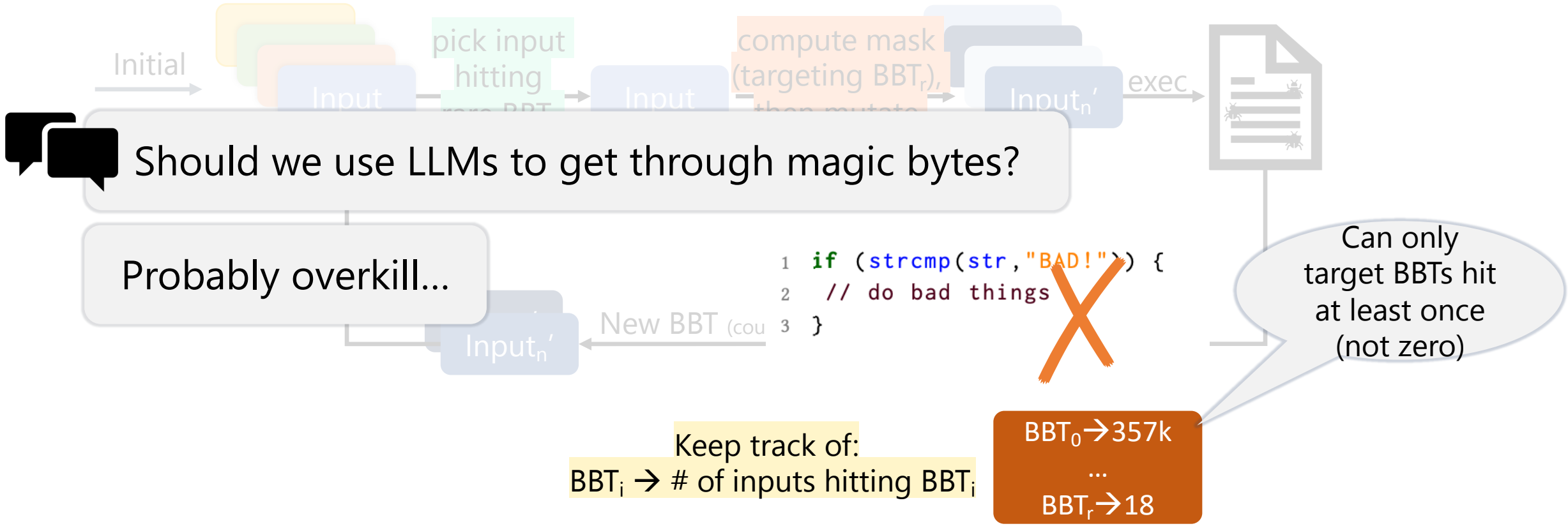
# Critically Analyzing FairFuzz



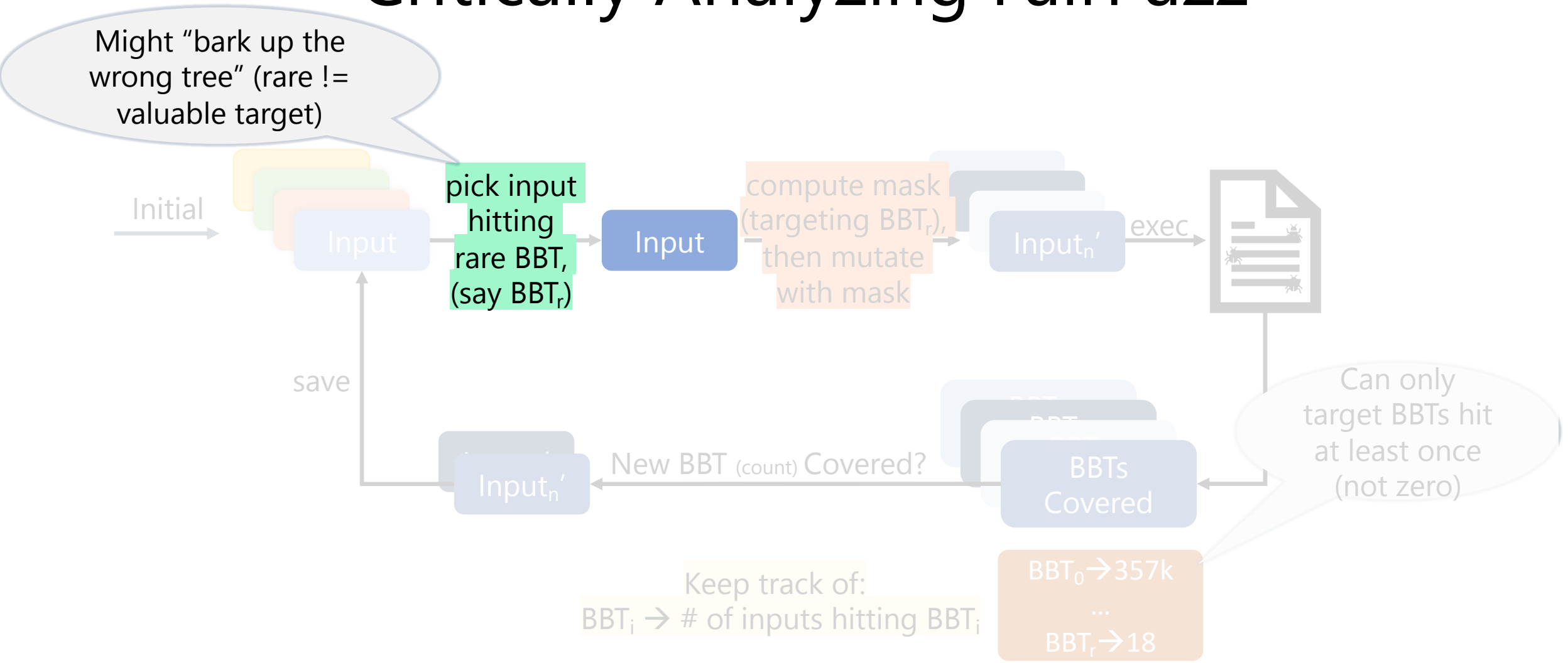
# Critically Analyzing FairFuzz



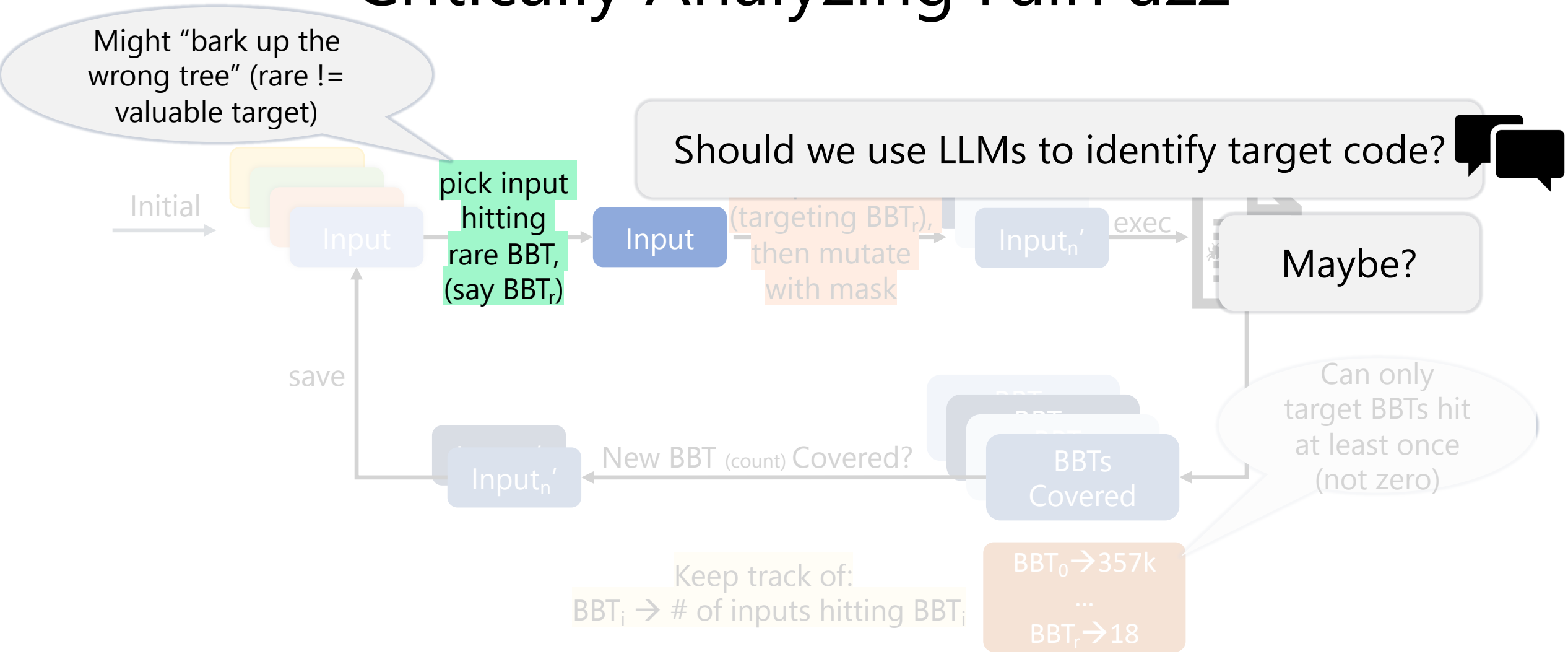
# Critically Analyzing FairFuzz



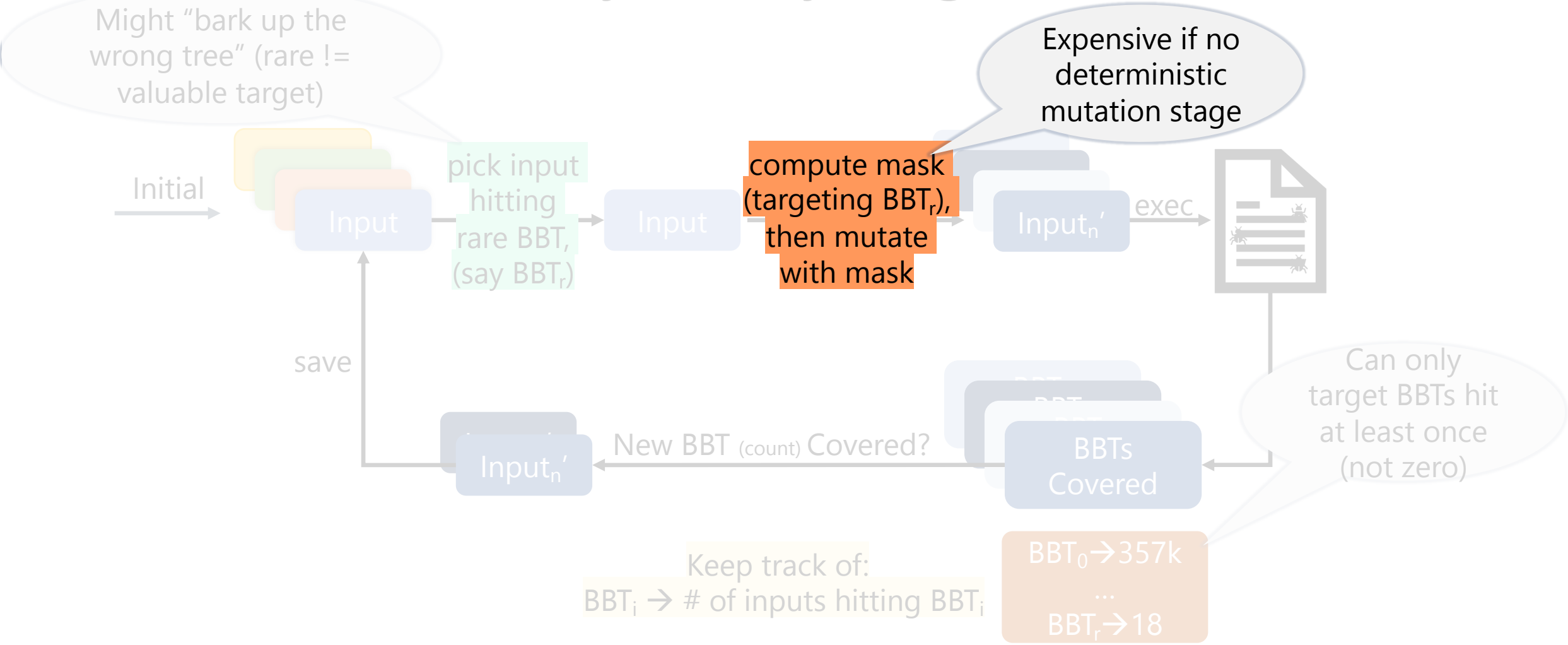
# Critically Analyzing FairFuzz



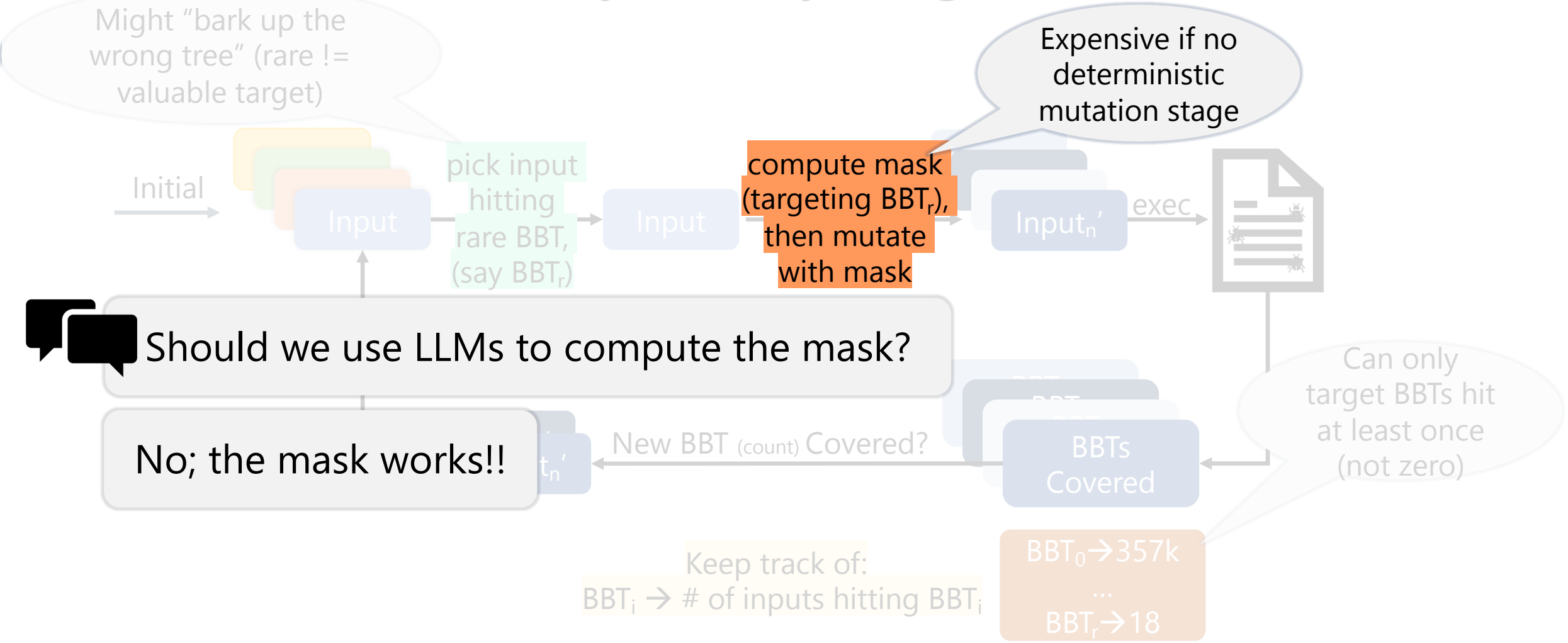
# Critically Analyzing FairFuzz



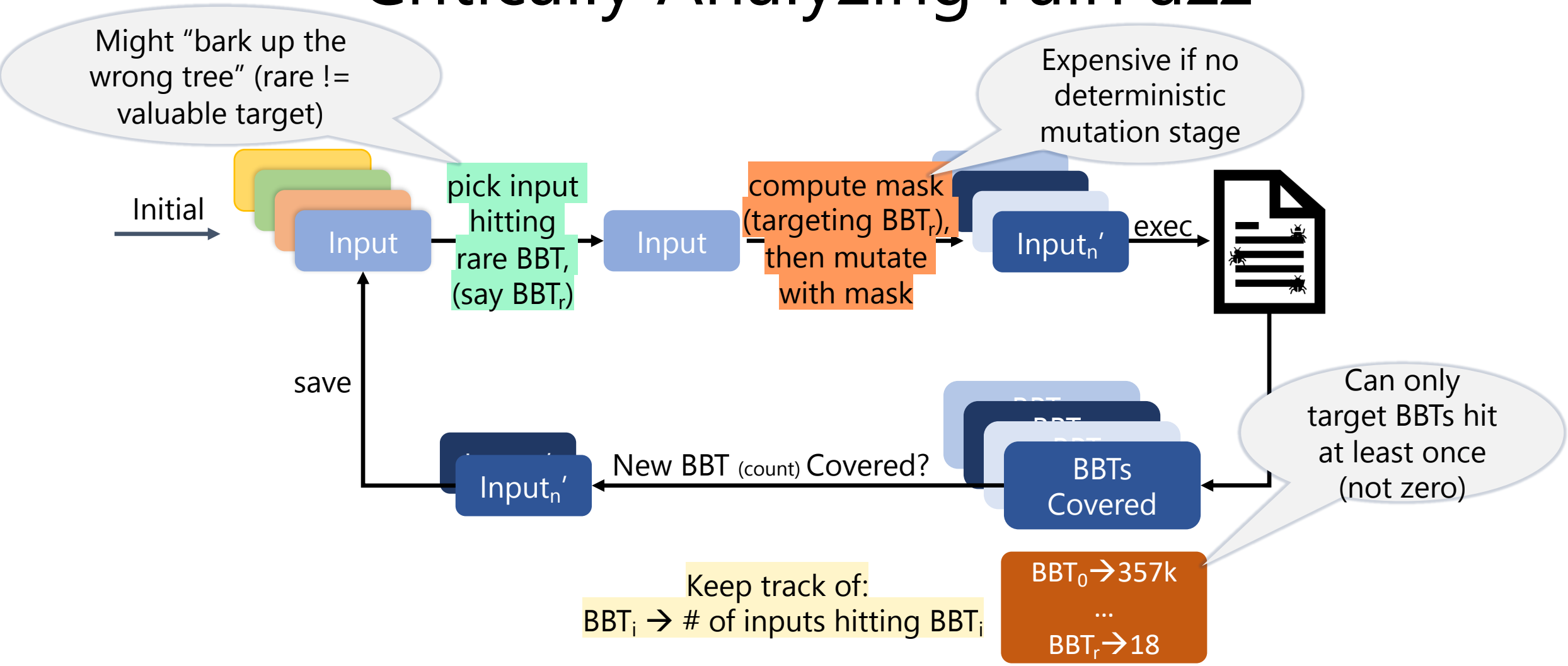
# Critically Analyzing FairFuzz



# Critically Analyzing FairFuzz



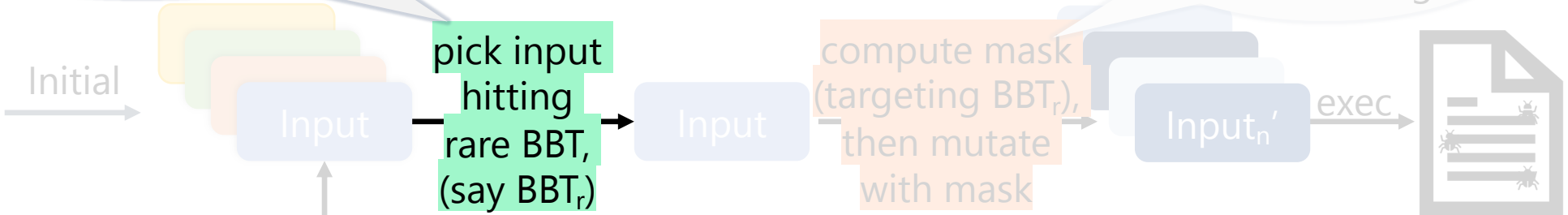
# Critically Analyzing FairFuzz



# On the Flipside…

Might “bark up the wrong tree” (rare != valuable target)

Gives FairFuzz an “exploit” flavor.




Not a problem in ensembles.

Can only target BBTs hit at least once (not zero)

Keep track of:
$BBT_i$ → # of inputs hitting $BBT_i$

$BBT_0$→357k
…
$BBT_r$→18

# The Persistence of FairFuzz

The Tool:

- Drop-in replacement for AFL
- “Exploit” (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals

# The Persistence of FairFuzz

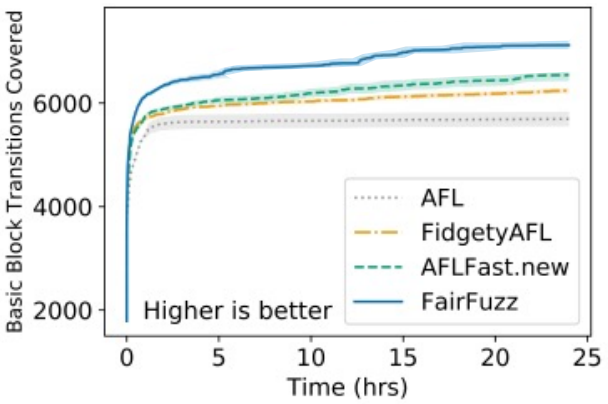
The Tool:

- Drop-in replacement for AFL
- “Exploit” (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals

w/o CI



# The Persistence of FairFuzz

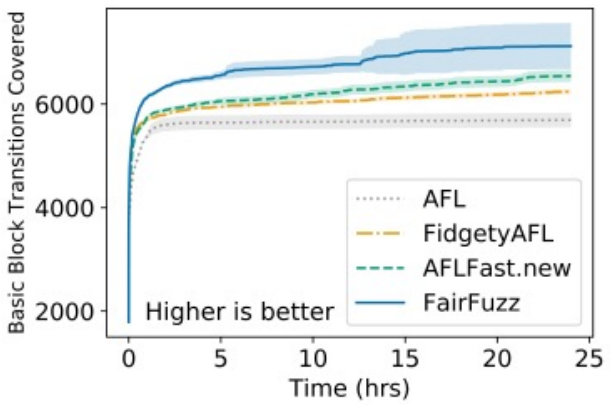
The Tool:

- Drop-in replacement for AFL
- "Exploit" (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals

w/ CI



# The Persistence of FairFuzz

The Tool:

- Drop-in replacement for AFL
- “Exploit” (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals
  - Discussing havoc vs. deterministic mutations

# The Persistence of FairFuzz

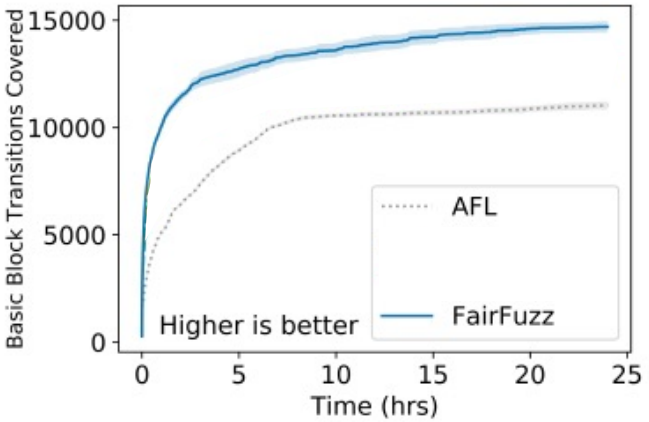
The Tool:

- Drop-in replacement for AFL
- "Exploit" (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals
  - Discussing havoc vs. deterministic mutations

baselines w/ det stage



# The Persistence of FairFuzz

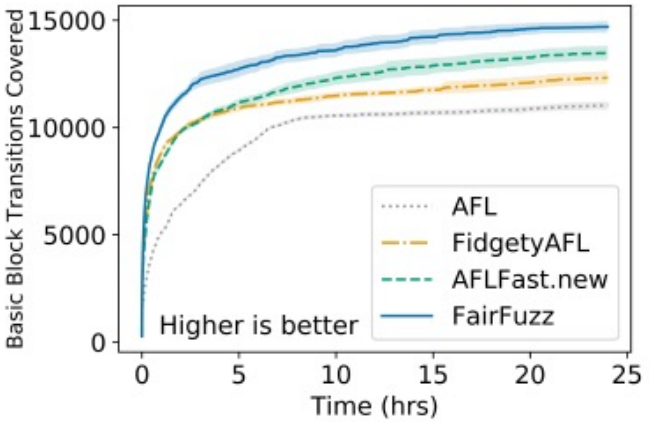
The Tool:

- Drop-in replacement for AFL
- "Exploit" (rather than explore) nature:
  - ☹ for coverage alone…
  - ☺ different bugs in ensembles

The Science:

- Early to:
  - Discussing metrics issues (unique paths)
  - Confidence intervals
  - Discussing havoc vs. deterministic mutations

baselines w/o det stage



# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

**Analyse:** Lemieux and Sen. FairFuzz:... ASE '18. https://doi.org/10.1145/3238147.3238176

## ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

...but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

## ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

...but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

# ML is not Magic

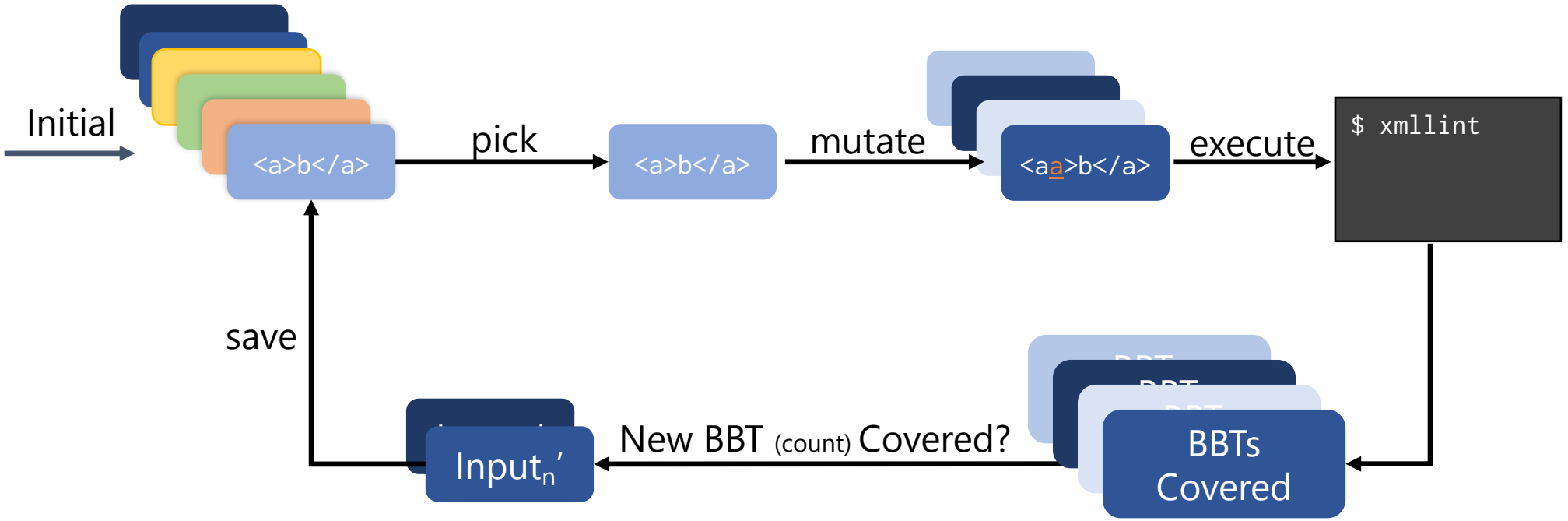
Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

**Analyse:** Padhye, et al. Semantic Fuzzing with Zest (ISSTA '19). https://doi.org/10.1145/3293882.3330576
Bavishi, et al. AutoPandas: Neural-Backed Generators… (OOPSLA'19) https://doi.org/10.1145/3360594

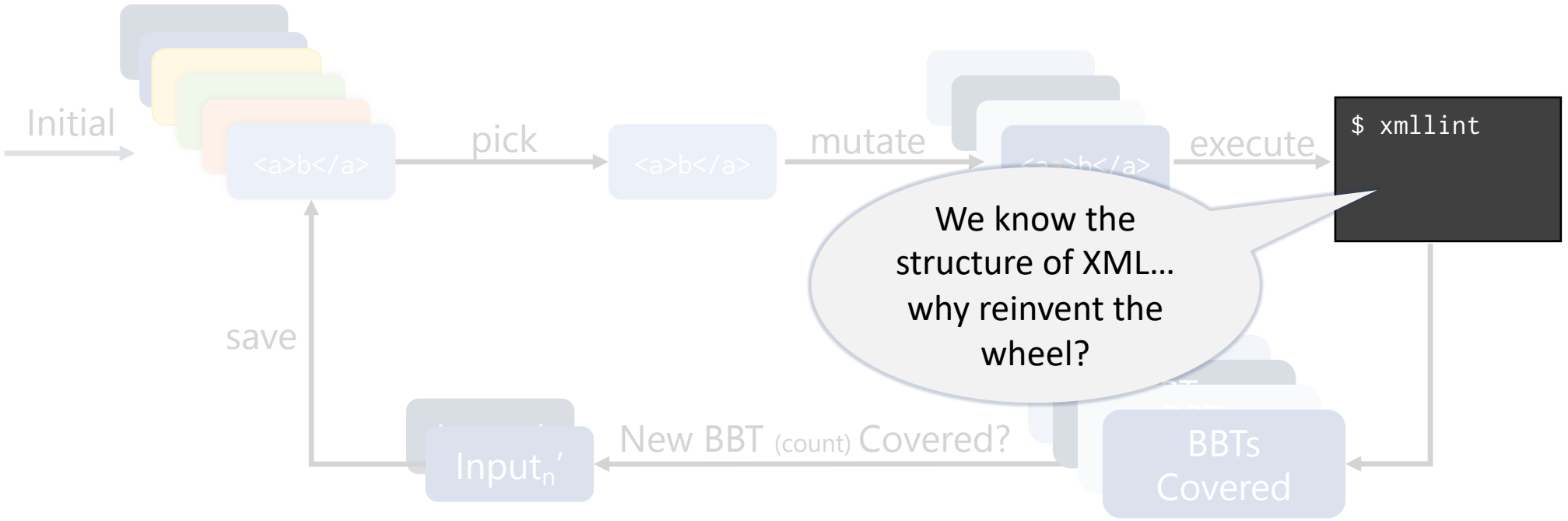
# Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

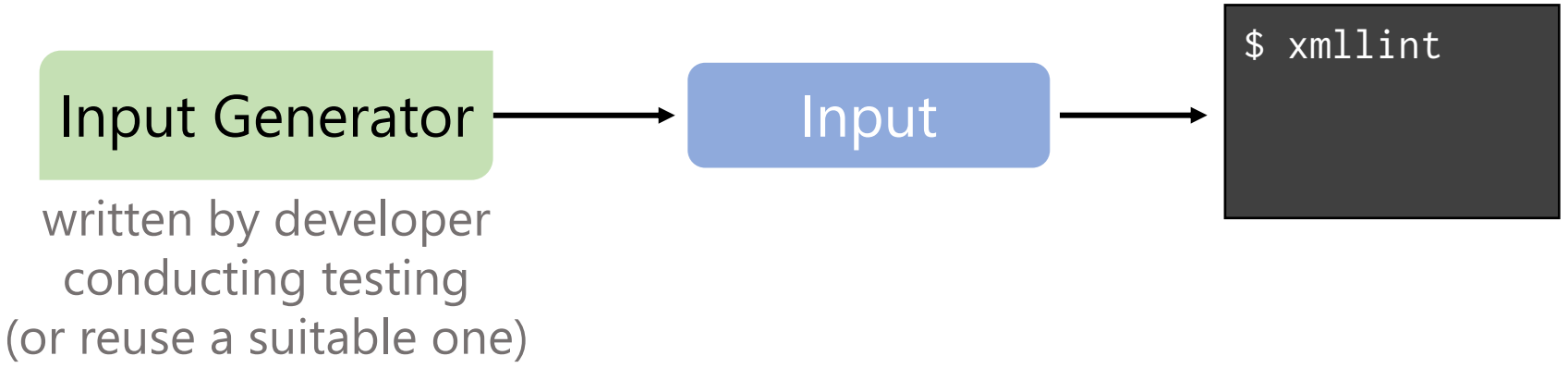
# Program Takes in Structured Inputs?



# Program Takes in Structured Inputs?

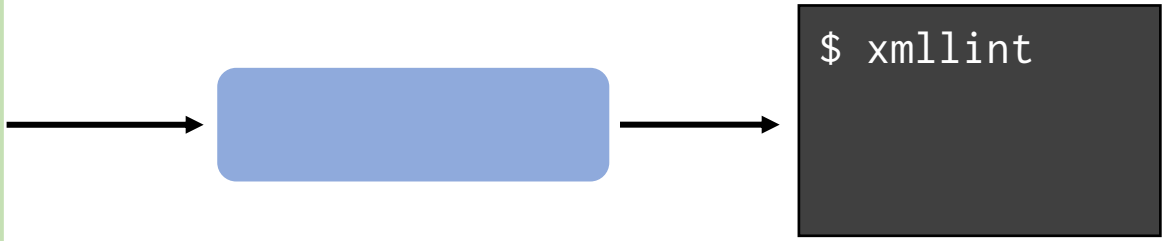


# Generator-Based Fuzzing



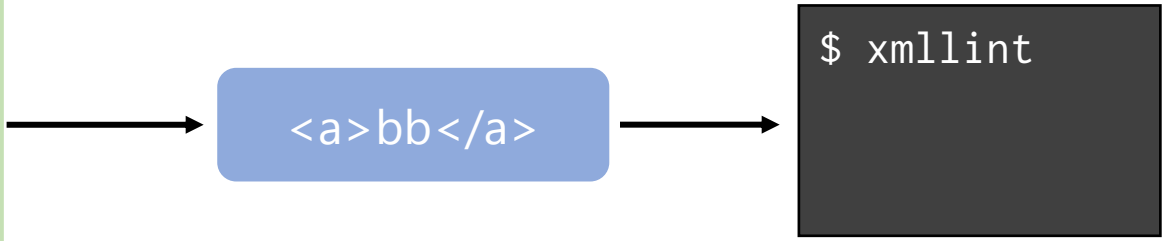
# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

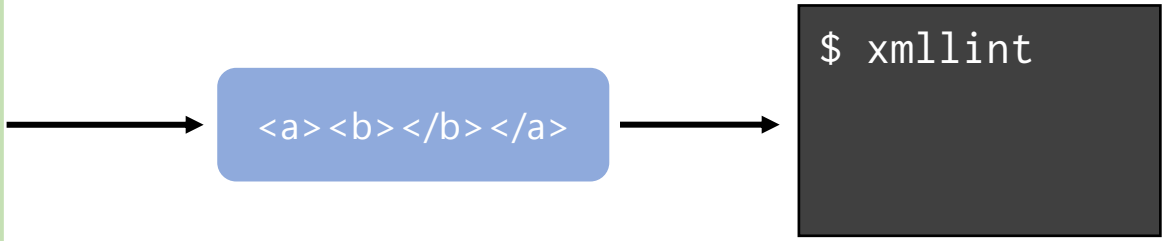


# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

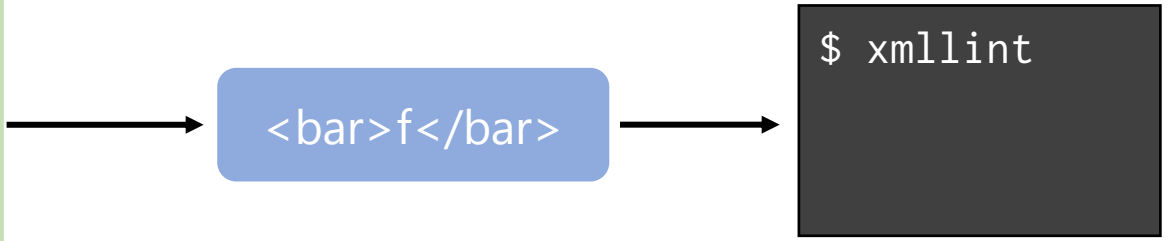
# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



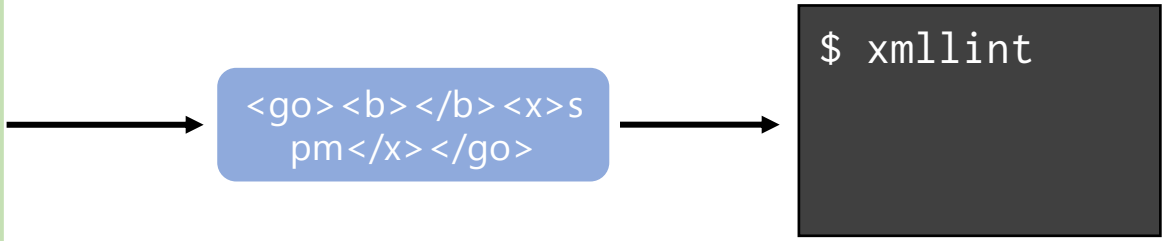
# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



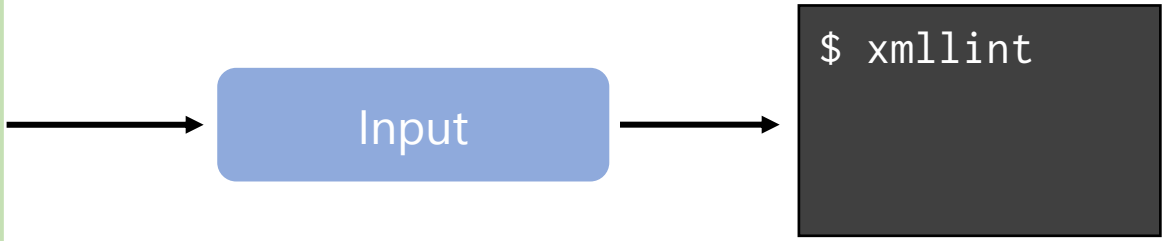
# Generator-Based Fuzzing

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



# I ♥ (The Abstraction Of) Generators

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



# I ♥ (The Abstraction Of) Generators

Defines a range of "valid" inputs

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

Defines the distribution over that range

# I ♥ (The Abstraction Of) Generators

Defines a range of "valid" inputs

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

Defines the distribution over that range

# Directly Harnessing Randomness

Defines a range of "valid" inputs

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

Defines the distribution over that range

Make random guided by:
- coverage-guided fuzzing (Zest)
- graph neural networks (AutoPandas)
- reinforcement learning (RLCheck)

# Directly Harnessing Randomness

Defines a range of "valid" inputs

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

Defines the distribution over that range

Make random guided by:
- coverage-guided fuzzing (Zest)
- graph neural networks (AutoPandas)
- reinforcement learning (RLCheck)

# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

# More Formally:

## Parsing Randomness

HARRISON GOLDSTEIN, University of Pennsylvania, USA
BENJAMIN C. PIERCE, University of Pennsylvania, USA

Random data generators can be thought of as parsers of streams of randomness. This perspective on generators for random data structures is established folklore in the programming languages community, but it has never been formalized, nor have its consequences been deeply explored.

We build on the idea of *freer monads* to develop *free generators*, which unify parsing and generation using a common structure that makes the relationship between the two concepts precise. Free generators lead naturally to a proof that a monadic generator can be factored into a parser plus a distribution over choice sequences. Free generators also support a notion of *derivative*, analogous to the familiar Brzozowski derivatives of formal languages, allowing analysis tools to "preview" the effect of a particular generator choice. This gives rise to a novel algorithm for generating data structures satisfying user-specified preconditions.

CCS Concepts: • **Software and its engineering** → *General programming languages.*

Additional Key Words and Phrases: Random generation, Parsing, Property-based testing, Formal languages

**ACM Reference Format:**
Harrison Goldstein and Benjamin C. Pierce. 2022. Parsing Randomness. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 128 (October 2022), 25 pages. https://doi.org/10.1145/3563291

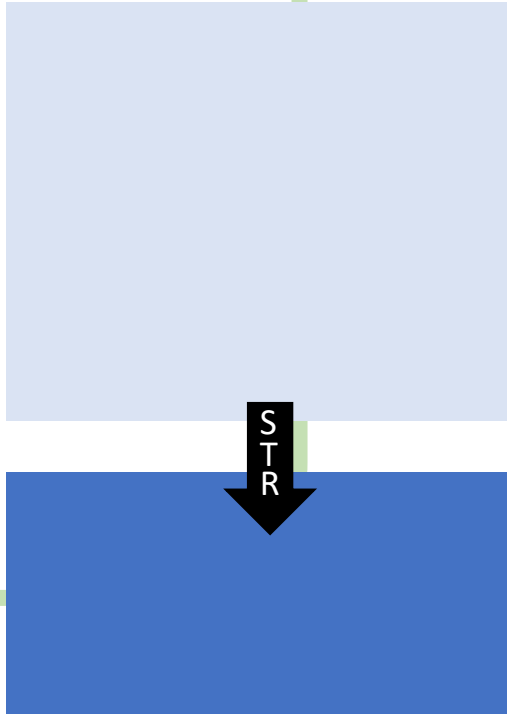
Harrison Goldstein and Benjamin C. Pierce. 2022. Parsing randomness. Proc. ACM Program. Lang. 6, OOPSLA2, Article 128 (October 2022), 25 pages. https://doi.org/10.1145/3563291

# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

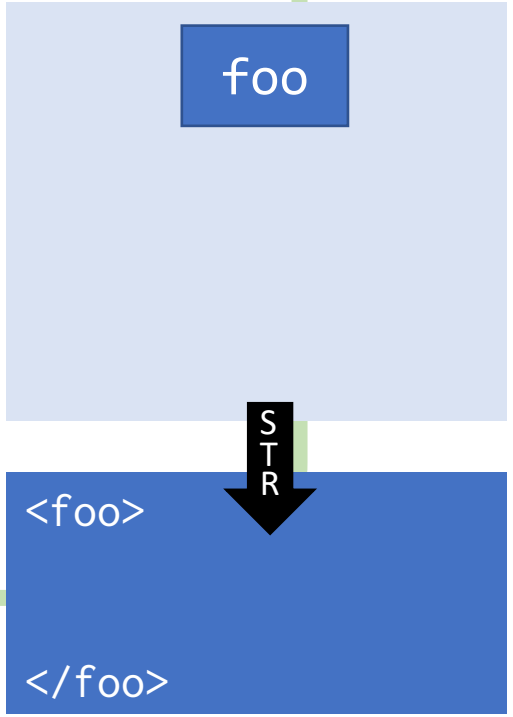
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



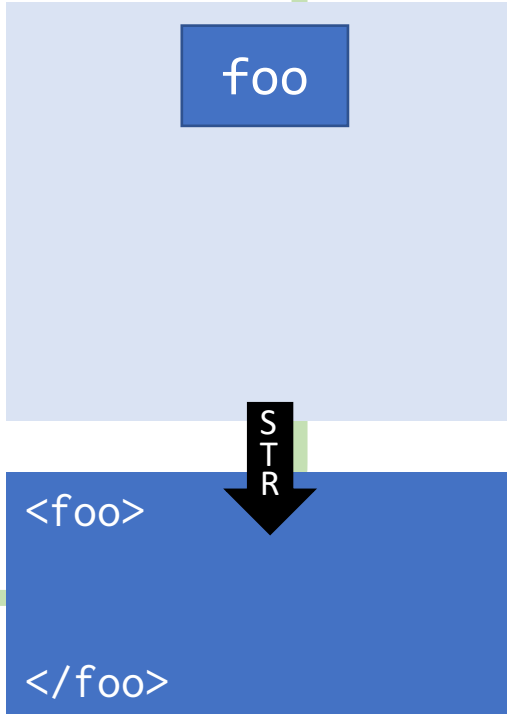
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



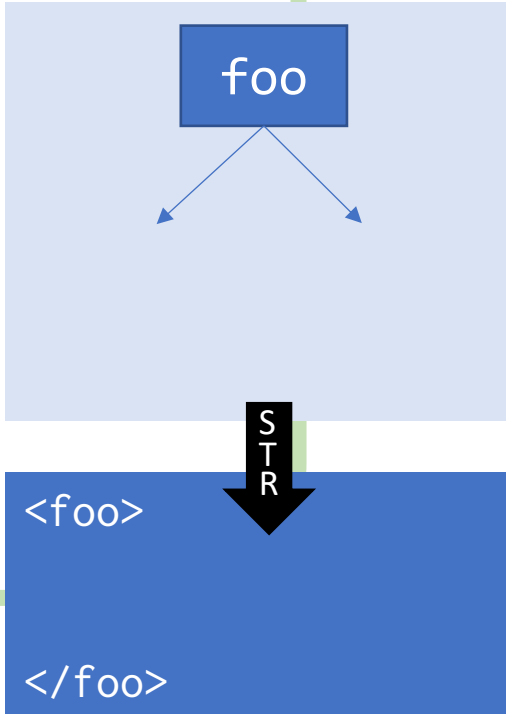
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



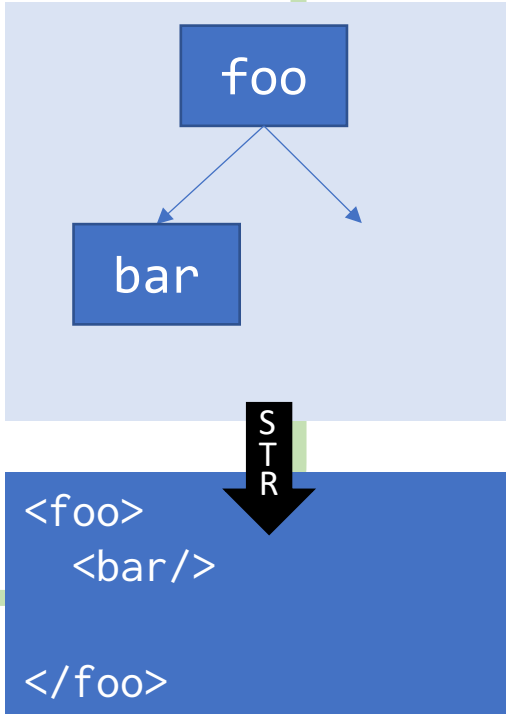
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



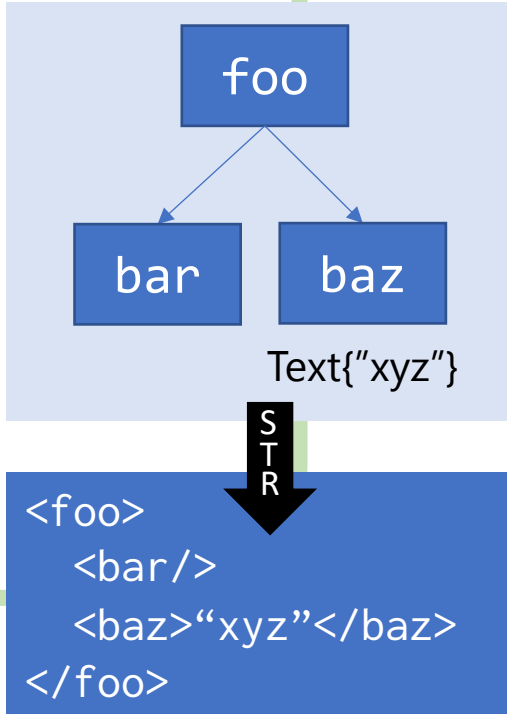
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



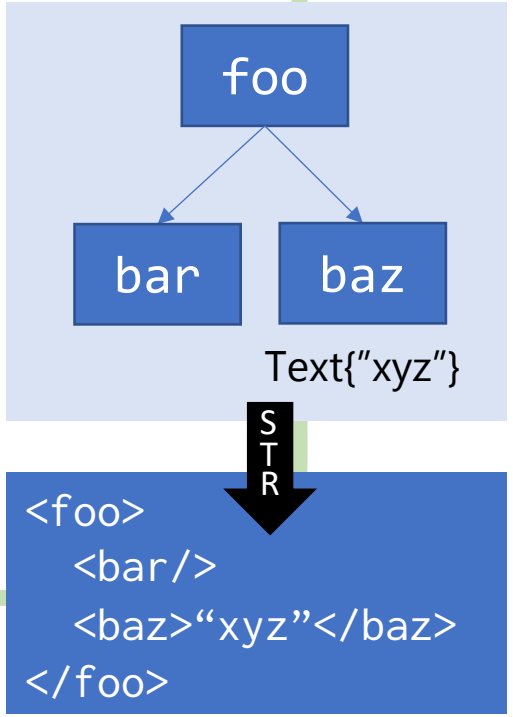
# Generator: Source of Randomness → Input

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



# Generator: Source of Randomness → Input

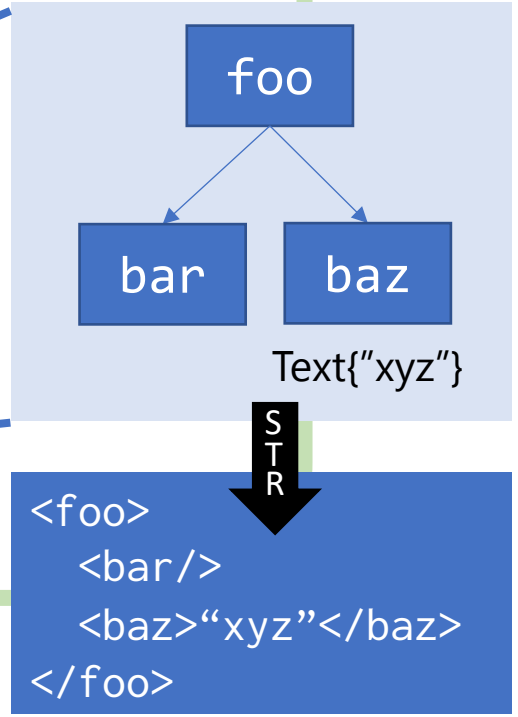
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



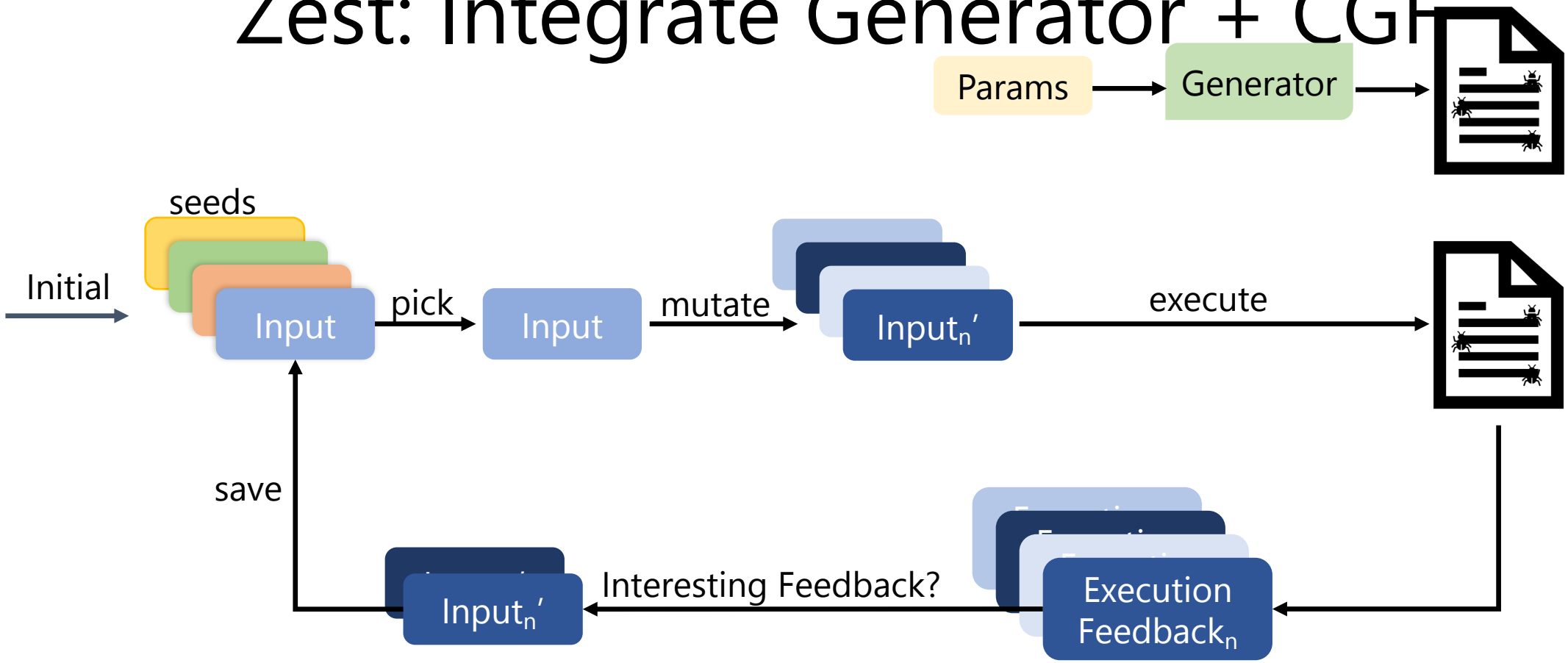
# Source of Randomness == Infinite Bit-Sequence

pseudo-random bits: 0000 0011 0110 0110 0110 1111 0110 1111 0000 0010 ...

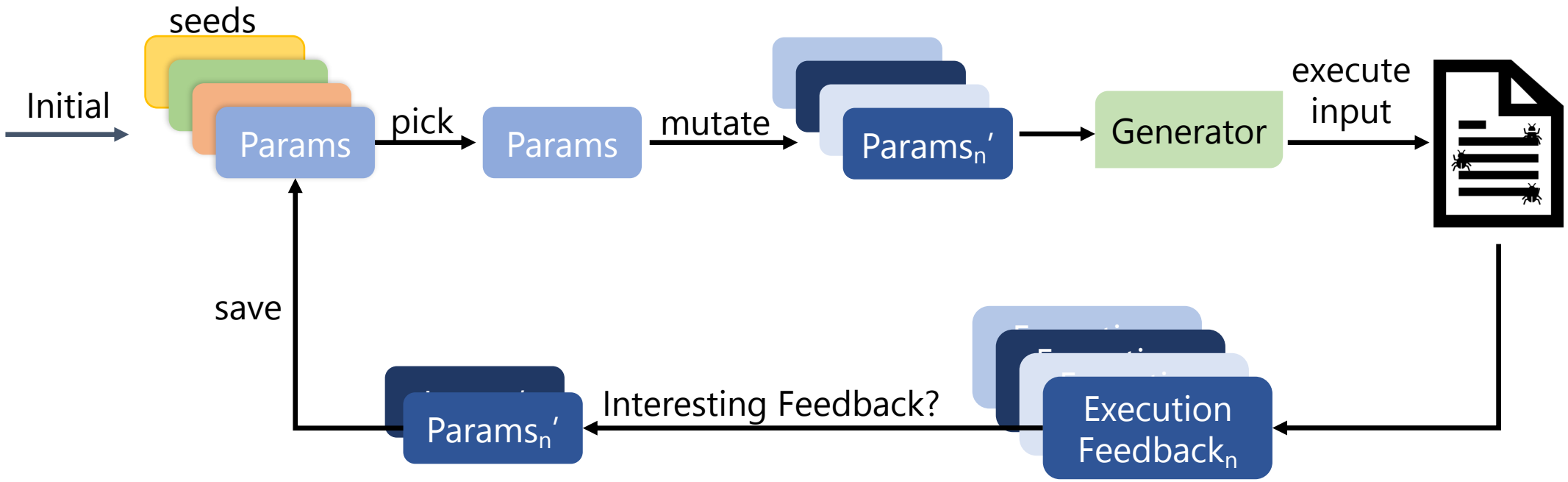
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random)
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



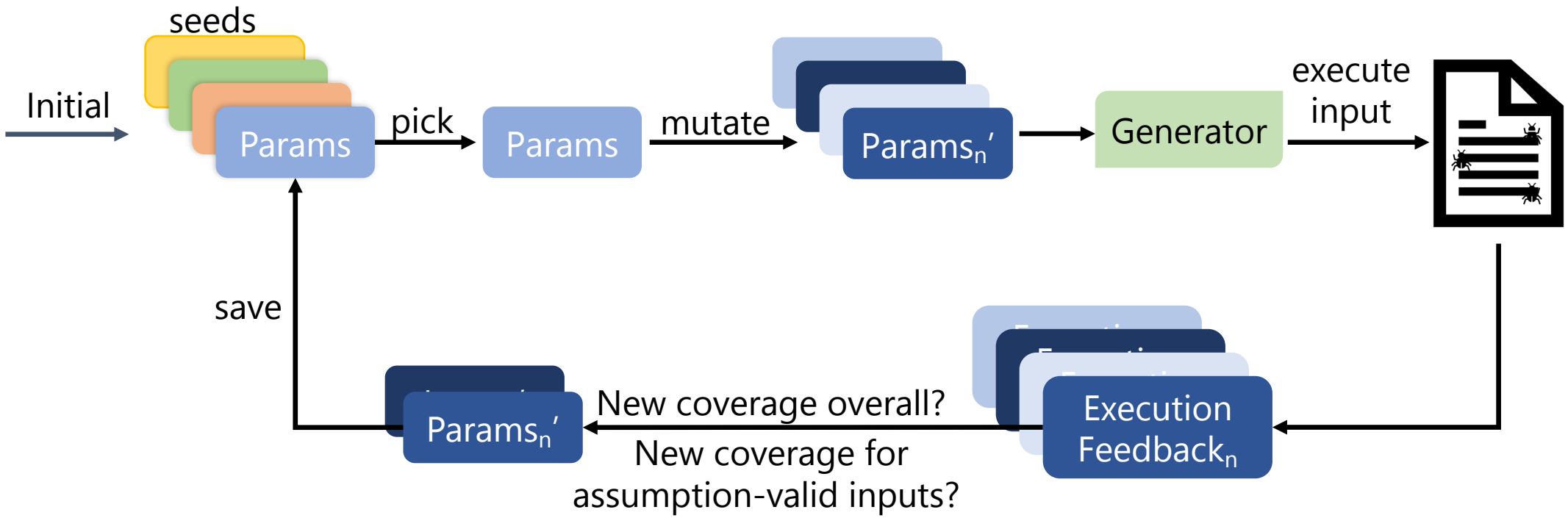
# Zest: Integrate Generator + CGF



# Zest: Integrate Generator + CGF



# Zest: Integrate Generator + CGF

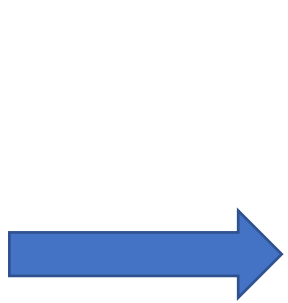


# Zest finds complex semantic bugs

```
while ((l_0)){
  while ((l_0)){
    if ((l_0))
    { break;var l_0;continue }
    { break;var l_0 }
  }
}
```

Zest-generated JavaScript input

Unreachable statement…
but not dead code!



Google Closure Compiler

IllegalStateException in VarCheck
during optimization

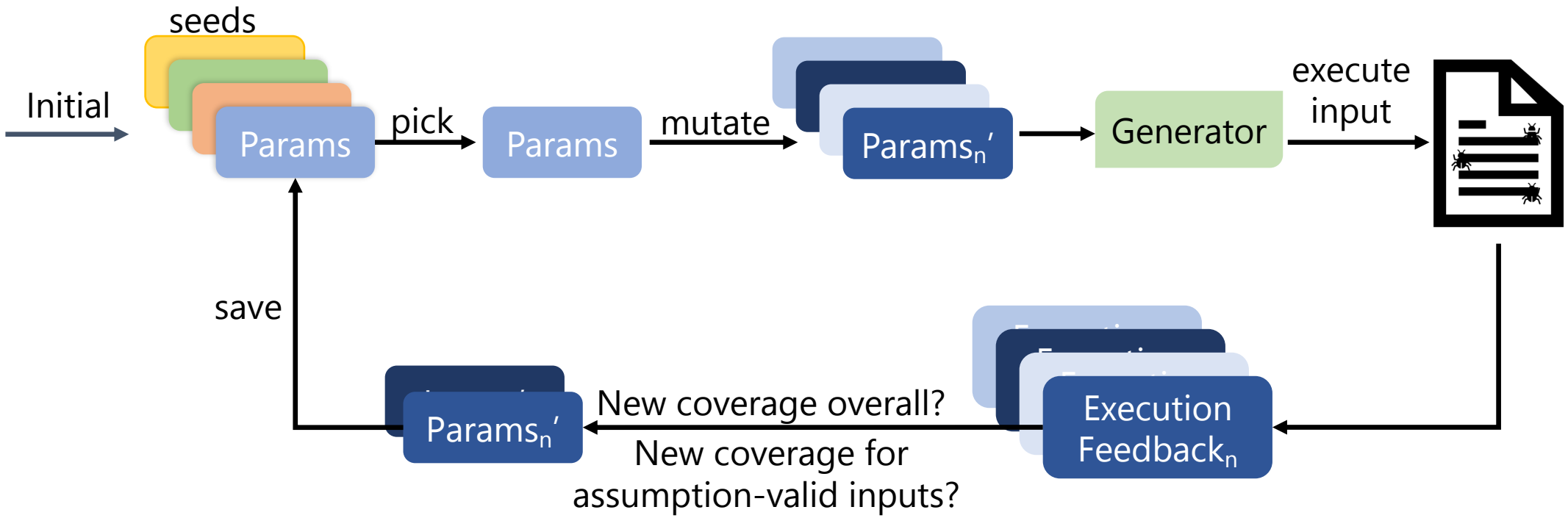
# ...and a whole bunch of other bugs

## Trophies

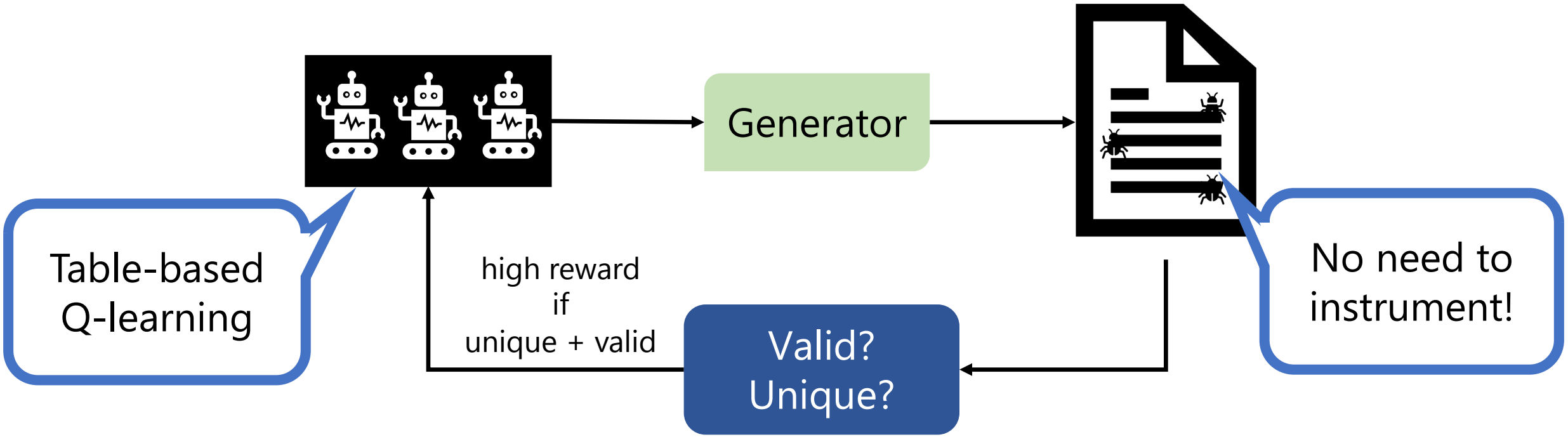
If you find bugs with JQF and you comfortable with sharing, We would be happy to add them to this list. Please send a PR for README.md with a link to the bug/cve you found.

- google/closure-compiler#2842: IllegalStateException in VarCheck: Unexpected variable
- google/closure-compiler#2843: NullPointerException when using Arrow Functions in dead code
- google/closure-compiler#3173: Algorithmic complexity / performance issue on fuzzed input
- google/closure-compiler#3220: ExpressionDecomposer throws IllegalStateException: Object method calls can not be decomposed
- JDK-8190332: PngReader throws NegativeArraySizeException when width is too large
- JDK-8190511: PngReader throws OutOfMemoryError for very small malformed PNGs
- JDK-8190512: PngReader throws undocumented IllegalArgumentException: "Empty Region" instead of IOException for malformed images with negative dimensions
- JDK-8190997: PngReader throws NullPointerException when PLTE section is missing
- JDK-8191023: PngReader throws NegativeArraySizeException in parse_tEXt_chunk when keyword length exceeeds chunk size
- JDK-8191076: PngReader throws NegativeArraySizeException in parse_zTXt_chunk when keyword length exceeds chunk size
- JDK-8191109: PngReader throws NegativeArraySizeException in parse_iCCP_chunk when keyword length exceeds chunk size
- JDK-8191174: PngReader throws undocumented IlegalArgumentException with message "Pixel stride times width must be <= scanline stride"
- JDK-8191073: JpegImageReader throws IndexOutOfBoundsException when reading malformed header
- JDK-8193444: SimpleDateFormat throws ArrayIndexOutOfBoundsException when format contains long sequences of unicode characters
- JDK-8193877: DateTimeFormatterBuilder throws ClassCastException when using padding
- mozilla/rhino#405: FAILED ASSERTION due to malformed destructuring syntax
- mozilla/rhino#406: ClassCastException when compiling malformed destructuring expression
- mozilla/rhino#407: java.lang.VerifyError in bytecode produced by CodeGen
- mozilla/rhino#409: ArrayIndexOutOfBoundsException when parsing '<!-'
- mozilla/rhino#410: NullPointerException in BodyCodeGen
- COLLECTIONS-714: PatriciaTrie ignores trailing null characters in keys
- COMPRESS-424: BZip2CompressorInputStream throws ArrayIndexOutOfBoundsException(s) when decompressing malformed input
- LANG-1385: StringIndexOutOfBoundsException in NumberUtils.createNumber
- **CVE-2018-11771**: Infinite Loop in Commons-Compress ZipArchiveInputStream (found by Tobias Ospelt)
- MNG-6375 / plexus-utils#34: NullPointerException when pom.xml has incomplete XML tag
- MNG-6374 / plexus-utils#35: ModelBuilder hangs with malformed pom.xml
- MNG-6577 / plexus-utils#57: Uncaught IllegalArgumentException when parsing unicode entity ref
- Bug 62655: Augment task: IllegalStateException when "id" attribute is missing
- BCEL-303: AssertionViolatedException in Pass 3A Verification of invoke instructions
- BCEL-307: ClassFormatException thrown in Pass 3A verification
- BCEL-308: NullPointerException in Verifier Pass 3A
- BCEL-309: NegativeArraySizeException when Code attribute length is negative
- BCEL-310: ArrayIndexOutOfBounds in Verifier Pass 3A
- BCEL-311: ClassCastException in Verifier Pass 2
- BCEL-312: AssertionViolation: INTERNAL ERROR Please adapt StringRepresentation to deal with ConstantPackage in Verifier Pass 2
- BCEL-313: ClassFormatException: Invalid signature: Ljava/lang/String)V in Verifier Pass 3A
- **CVE-2018-8036**: Infinite Loop leading to OOM in PDFBox's AFMParser (found by Tobias Ospelt)
- PDFBOX-4333: ClassCastException when loading PDF (found by Robin Schimpf)
- PDFBOX-4338: ArrayIndexOutOfBoundsException in COSParser (found by Robin Schimpf)
- PDFBOX-4339: NullPointerException in COSParser (found by Robin Schimpf)
- **CVE-2018-8017**: Infinite Loop in IptcAnpaParser
- **CVE-2018-12418**: Infinite Loop in junrar (found by Tobias Ospelt)
- **CVE-2019-17359**: Attempt to trigger a large allocation leads to OOM in Bouncycastle ASN.1 parser (found by Tobias Ospelt)

# Zest: Integrate Generator + CGF



# RLCheck: Integrate Generator + RL




Reddy, Lemieux, Padhye, and Sen. 2020. Quickly generating diverse valid test inputs with reinforcement learning. ICSE '20. https://doi.org/10.1145/3377811.3380399

# RLCheck: Quick Takeaways

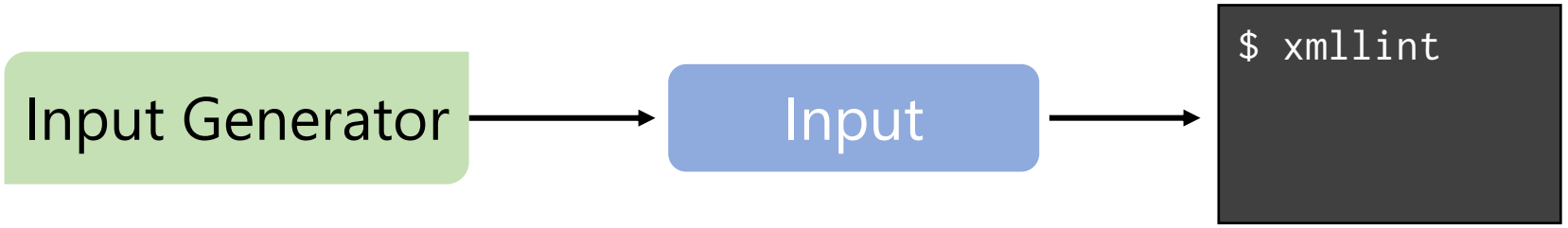
RLCheck (blackbox, RL-backed):
- generates many more unique valid inputs in the same time

QuickCheck (blackbox, random):
- finds easy bugs the quickest

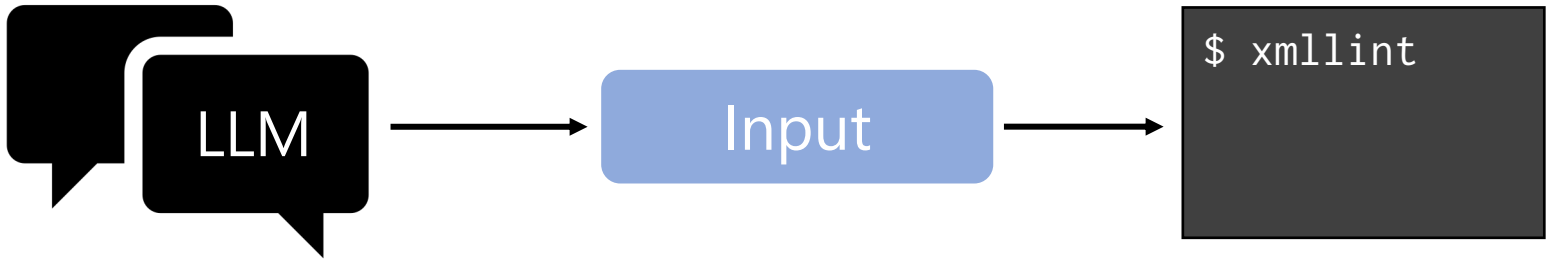
Zest (greybox, CGF-backed):
- finds deeper bugs

Reddy, Lemieux, Padhye, and Sen. 2020. Quickly generating diverse valid test inputs with reinforcement learning. ICSE '20. https://doi.org/10.1145/3377811.3380399

# Generator-Based Fuzzing



# Should We Do Away with Generators?



# Should We Do Away with Generators?

## FUZZ4ALL: Universal Fuzzing with Large Language Models


Chunqiu Steven Xia
University of Illinois
Urbana-Champaign, USA
chunqiu2@illinois.edu

Matteo Paltenghi
University of
Stuttgart, Germany
mattepalte@live.it

Jia Le Tian
University of Illinois
Urbana-Champaign, USA
jialelt2@illinois.edu

Michael Pradel
University of
Stuttgart, Germany
michael@binaervarianz.de

Lingming Zhang
University of Illinois
Urbana-Champaign, USA
lingming@illinois.edu



**ABSTRACT**

Fuzzing has achieved tremendous success in discovering bugs and vulnerabilities in various software systems. Systems under test (SUTs) that take in programming or formal language as inputs, e.g., compilers, runtime engines, constraint solvers, and software libraries with accessible APIs, are especially important as they are fundamental building blocks of software development. However, existing fuzzers for such systems often target a specific language, and thus cannot be easily applied to other languages or even other versions of the same language. Moreover, the inputs generated by existing fuzzers are often limited to specific features of the input language, and thus can hardly reveal bugs related to other or new features. This paper presents FUZZ4ALL, the first fuzzer that is *universal* in the sense that it can target many different input languages and many different features of these languages. The key idea behind FUZZ4ALL is to leverage large language models (LLMs)



*(ICSE '24), April 14–20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639121


**1 INTRODUCTION**

Fuzz testing [69, 84], also known as fuzzing, is an automated testing approach for generating inputs designed to expose unexpected behaviors, e.g., crashes, of a system under test (SUT). Researchers and practitioners have successfully built practical fuzzing tools, which have shown great success in finding numerous bugs and vulnerabilities in real-world systems [6]. A particularly important family of SUTs are systems that take in programming or formal language inputs, e.g., compilers, runtime engines, and constraint solvers. Numerous fuzzers have been proposed for such systems since they are the fundamental building blocks for software development [12]. For example, finding bugs in compilers and runtime engines is crucial because they can affect all corresponding downstream applications.

# Should We Do Away with Generators?



FUZZ4ALL: Universal Fuzzing with Large Language Models

Chunqiu Steven Xia
University of Illinois
Urbana-Champaign, USA
chunqiu2@illinois.edu

Matteo Paltenghi
University of
Stuttgart, Germany
mattepalte@live.it

Jia Le Tian
University of Illinois
Urbana-Champaign, USA
jialelt2@illinois.edu

My (surely unexpected) take: not always!

vulnerabilities in various software systems. Systems under test (SUTs) that take in programming or formal language as inputs, e.g., compilers, runtime engines, constraint solvers, and software libraries with accessible APIs, are especially important as they are fundamental building blocks of software development. However, existing fuzzers for such systems often target a specific language, and thus cannot be easily applied to other languages or even other versions of the same language. Moreover, the inputs generated by existing fuzzers are often limited to specific features of the input language, and thus can hardly reveal bugs related to other or new features. This paper presents FUZZ4ALL, the first fuzzer that is *universal* in the sense that it can target many different input languages and many different features of these languages. The key idea behind FUZZ4ALL is to leverage large language models (LLMs)

## 1 INTRODUCTION

Fuzz testing [69, 84], also known as fuzzing, is an automated testing approach for generating inputs designed to expose unexpected behaviors, e.g., crashes, of a system under test (SUT). Researchers and practitioners have successfully built practical fuzzing tools, which have shown great success in finding numerous bugs and vulnerabilities in real-world systems [6]. A particularly important family of SUTs are systems that take in programming or formal language inputs, e.g., compilers, runtime engines, and constraint solvers. Numerous fuzzers have been proposed for such systems since they are the fundamental building blocks for software development [12]. For example, finding bugs in compilers and runtime engines is crucial because they can affect all corresponding downstream applications.

# AutoPandas: Synthesize `pandas` Programs

Given:

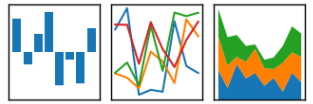
| | weight | |
|---|---|---|
| | kg | lbs |
| cat | 1 | 2 |
| dog | 2 | 4 |

input

| | | weight |
|---|---|---|
| cat | kg | 1 |
| | lbs | 2 |
| dog | kg | 2 |
| | lbs | 4 |

output

SYNTHESIZE

pandas  prog

```
output = input.stack(
 level=1, dropna=True)
```

# AutoPandas: Synthesize pandas Programs

Given:

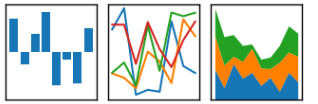
| | weight | |
|---|---|---|
| | kg | lbs |
| cat | 1 | 2 |
| dog | 2 | 4 |

input

| unit | avg |
|---|---|
| kg | 1.5 |
| lbs | 3 |

output

SYNTHESIZE

pandas  prog

```
out_1 = input.melt(
            col_level=1,
            var_name="unit",
            value_name="avg")
out_2 = out_1.groupby("unit")
output = out_2.mean()
```

# AutoPandas: Generator-Based Synthesis

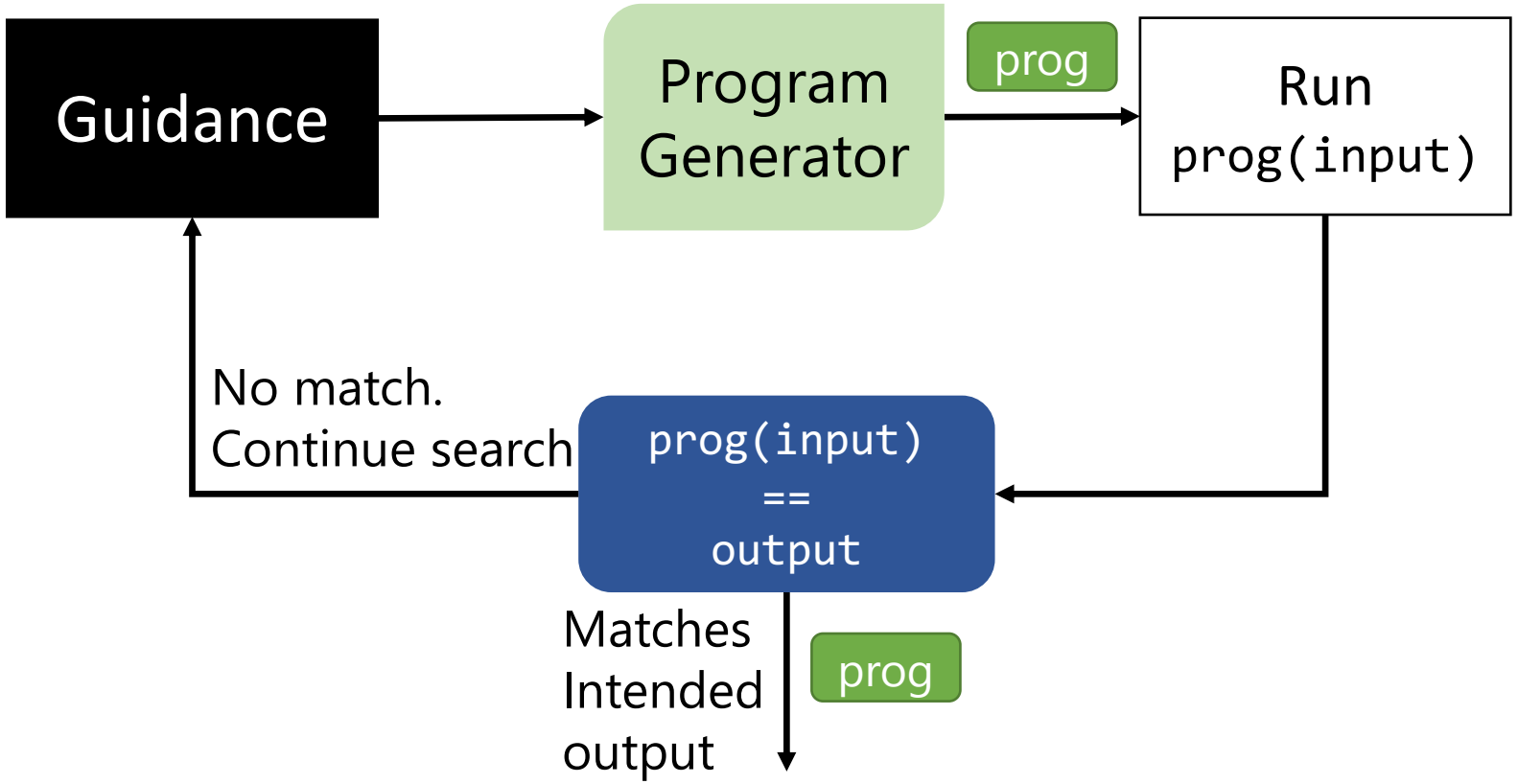
Spec:

| | weight | |
|---|---|---|
| | kg | lbs |
| cat | 1 | 2 |
| dog | 2 | 4 |

`input`

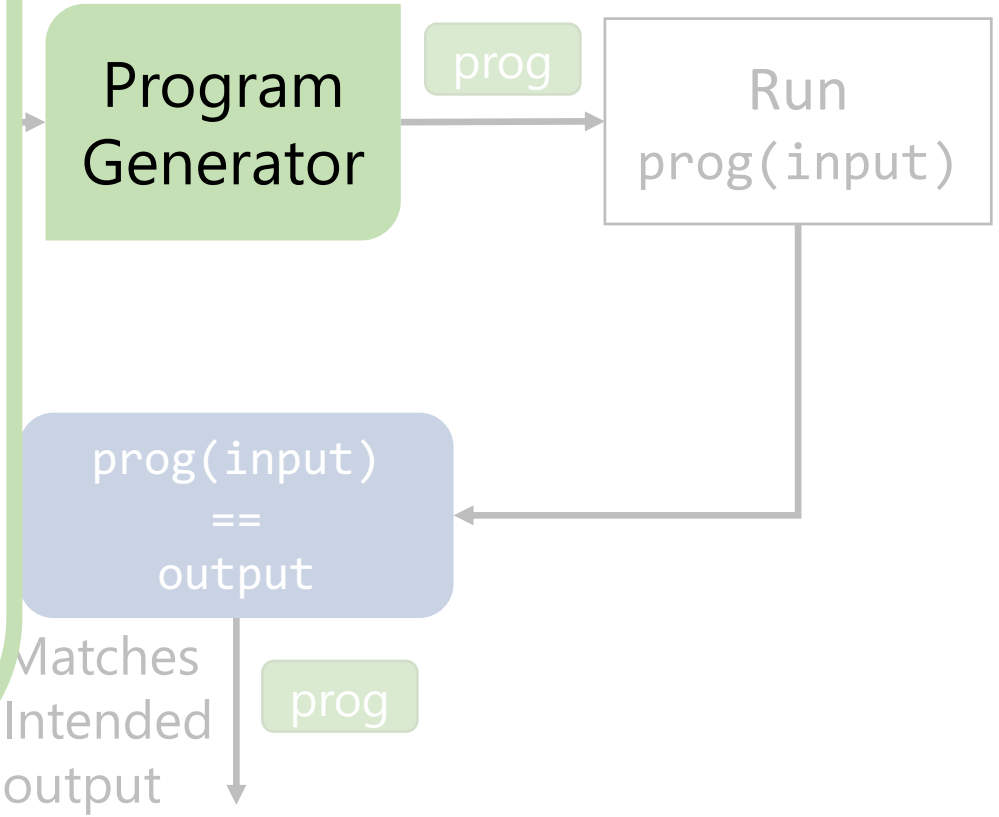
| unit | avg |
|---|---|
| kg | 1.5 |
| lbs | 3 |

`output`



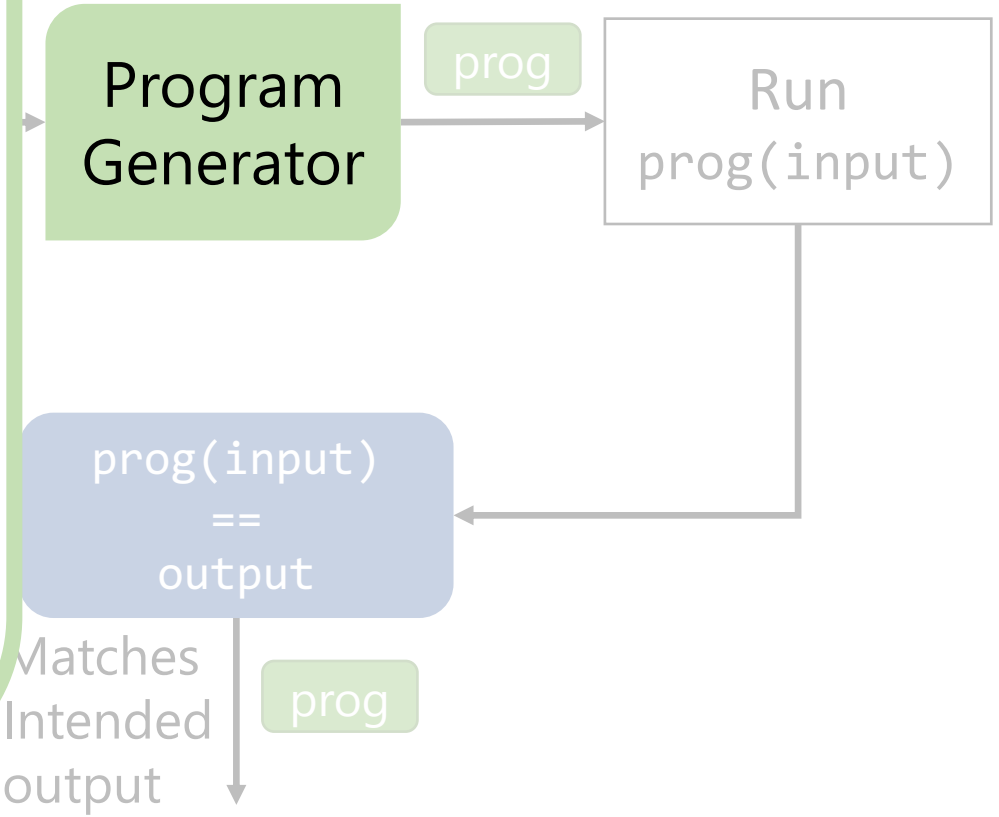
# AutoPandas: Generator Captures API Constraints

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …])

  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1])
      arg_labels = random.choice(input.index)
                   if arg_axis == 0
                   else random.choice(input.columns)
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions

    return fn_sequence
```



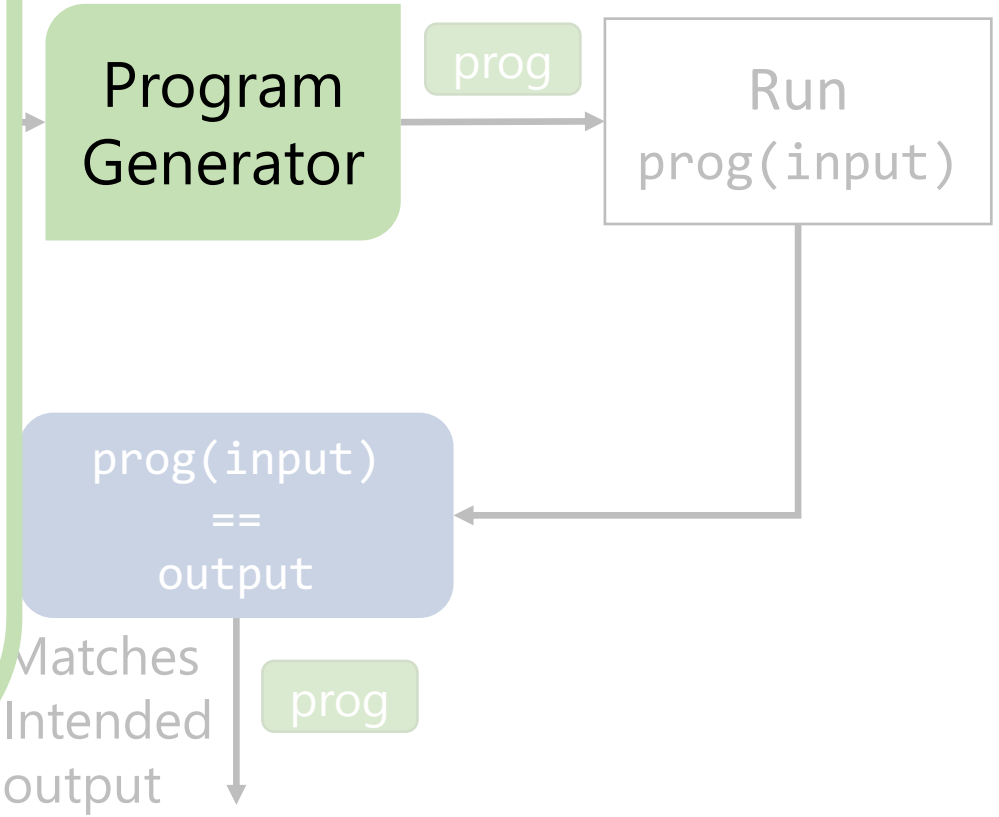
# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …])

  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1])
      arg_labels = random.choice(input.index)
                   if arg_axis == 0
                   else random.choice(input.columns)
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions

    return fn_sequence
```



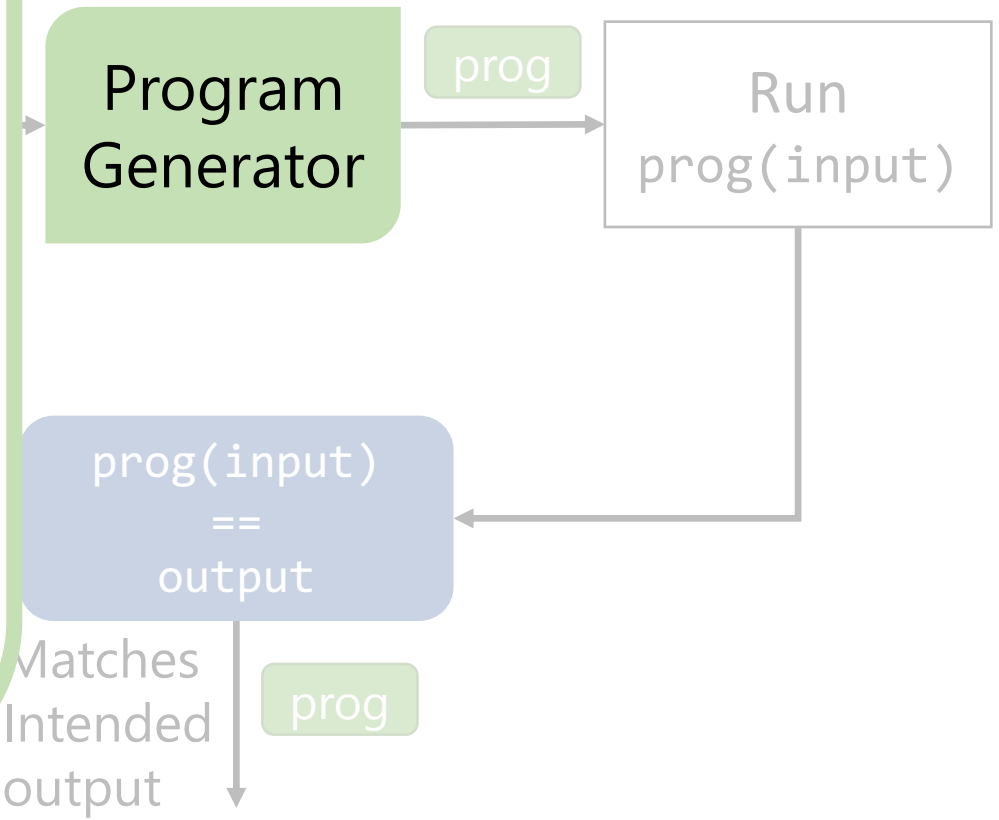
# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …])

  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1])
      arg_labels = random.choice(input.index)
                   if arg_axis == 0
                   else random.choice(input.columns)
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions

    return fn_sequence
```



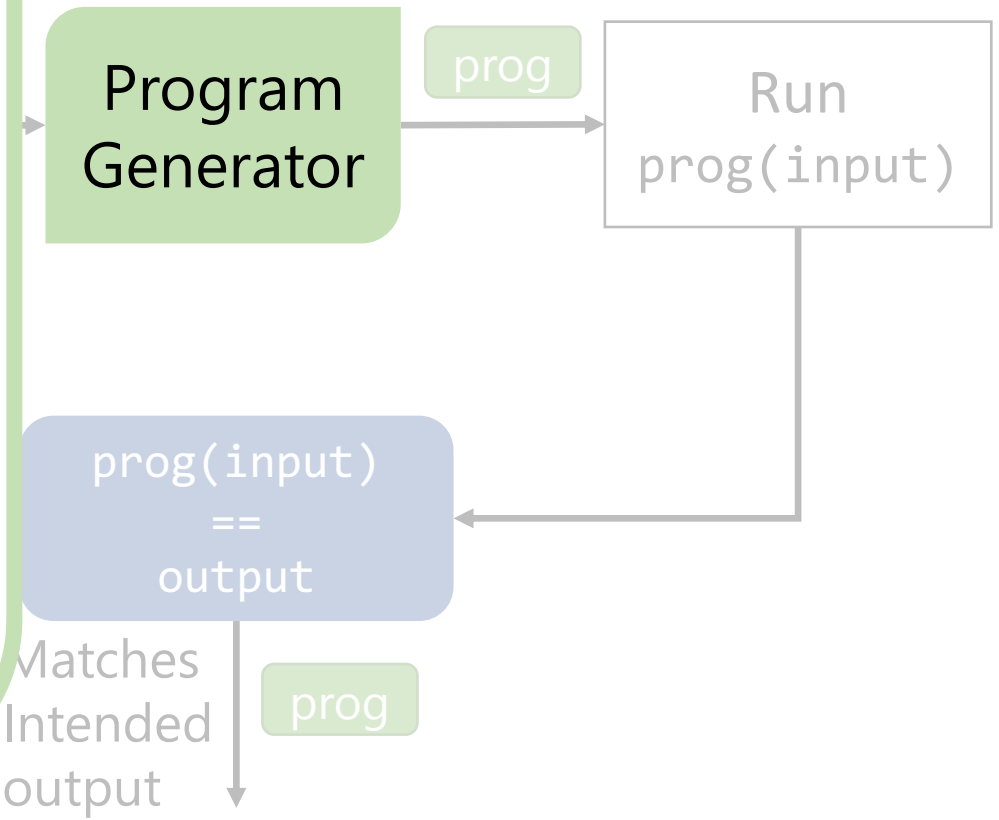
# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …])

  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1])

      arg_labels = random.choice(input.index)

                   if arg_axis == 0
                   else random.choice(input.columns)

      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions
    return fn_sequence
```



# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …],
                        context=(input, output))
  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1],
                        context=(input, output))
      arg_labels = random.choice(input.index,
                        context=(input, output))
                    if arg_axis == 0
                    else random.choice(input.columns,
                        context=(input, output))
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions
    return fn_sequence
```



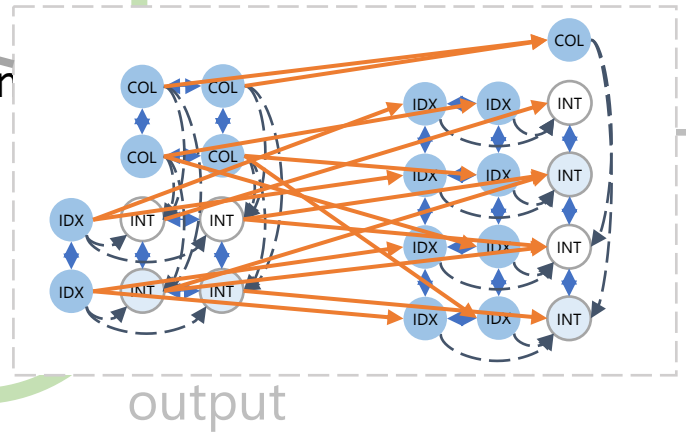
# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = random.sequence([drop, melt, …],
                    context=(input, output))
  for fn in fn_sequence:
    if fn == drop:
      arg_axis = random.choice([0,1],
                    context=(input, output))
      arg_labels = random.choice(input.index,
                    context=(input, output))
                  if arg_axis == 0
                  else random.choice(input.columr
                    context=(input, output))
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions
    return fn_sequence
```

Program Generator

prog

Run prog(input)




output

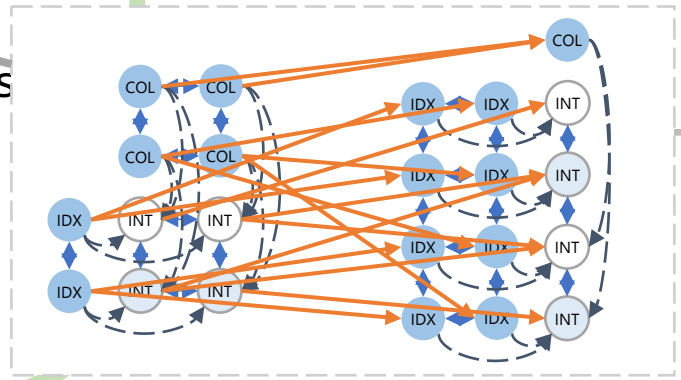
# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = guide.sequence([drop, melt, …],
                    context=(input, output))
  for fn in fn_sequence:
    if fn == drop:
      arg_axis = guide.choice([0,1],
                    context=(input, output))
      arg_labels = guide.choice(input.index,
                    context=(input, output))
                  if arg_axis == 0
                  else guide.choice(input.columns
                    context=(input, output))
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions
    return fn_sequence
```

Program Generator

prog

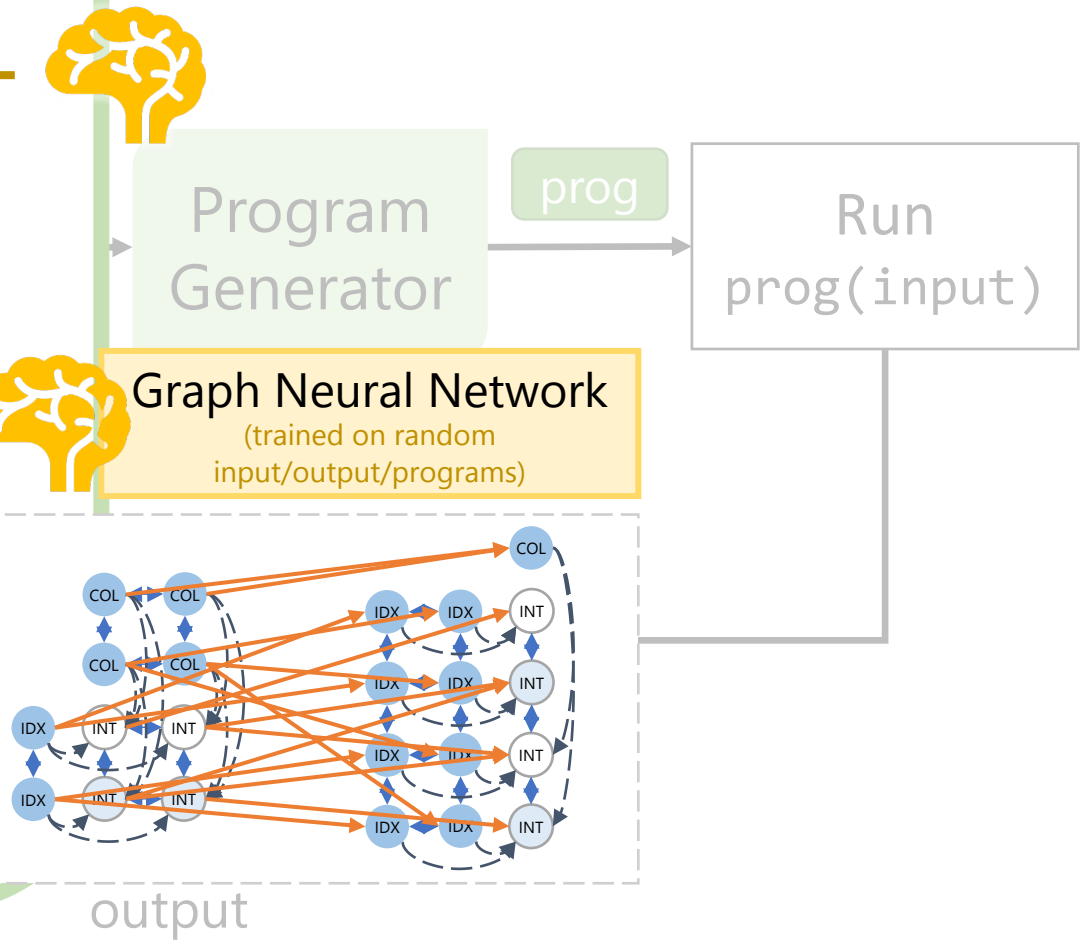
Run prog(input)




output

# Smart Generator Guided by I/O Example

```
def gen_pandas_program(input, output):
  fn_sequence = guide.sequence([drop, melt, …],
                    context=(input, output))
  for fn in fn_sequence:
    if fn == drop:
      arg_axis = guide.choice([0,1],
                    context=(input, output))
      arg_labels = guide.choice(input.index,
                    context=(input, output))
                  if arg_axis == 0
                  else guide.choice(input.columns
                    context=(input, output))
      fn.add_args(arg_axis, arg_labels)
      // … populate args for other functions
    return fn_sequence
```



# AutoPandas: Scales to Real-World API Uses

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_11881165 | 1 | 15 | 1 | Y | 0.54 |
| SO_11941492 | 1 | 783 | 8 | Y | 12.55 |
| SO_13647222 | 1 | 5 | 1 | Y | 3.32 |
| SO_18172851 | 1 | - | - | N | - |
| SO_49583055 | 1 | - | - | N | - |
| SO_49592930 | 1 | 2 | 1 | Y | 1.1 |
| SO_49572546 | 1 | 3 | 1 | Y | 1.1 |
| SO_13261175 | 1 | 39537 | 18 | Y | 300.2 |
| SO_13793321 | 1 | 92 | 1 | Y | 4.16 |
| SO_14085517 | 1 | 10 | 1 | Y | 2.24 |
| SO_11418192 | 2 | 158 | 1 | Y | 0.71 |
| SO_49567723 | 2 | 1684022 | 2 | Y | 753.1 |
| SO_13261691 | 2 | 65 | 1 | Y | 2.96 |
| SO_13659881 | 2 | 2 | 1 | Y | 1.38 |
| SO_13807758 | 2 | 711 | 2 | Y | 7.21 |
| SO_34365578 | 2 | - | - | N | - |

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_10982266 | 3 | - | - | N | - |
| SO_11811392 | 3 | - | - | N | - |
| SO_49581206 | 3 | - | - | N | - |
| SO_12065885 | 3 | 924 | 1 | Y | 0.9 |
| SO_13576164 | 3 | 22966 | 5 | Y | 339.25 |
| SO_14023037 | 3 | - | - | N | - |
| SO_53762029 | 3 | 27 | 1 | Y | 1.9 |
| SO_21982987 | 3 | 8385 | 10 | Y | 30.8 |
| SO_39656670 | 3 | - | - | N | - |
| SO_23321300 | 3 | - | - | N | - |

# AutoPandas: Scales to Real-World API Uses

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_11881165 | 1 | 15 | 1 | Y | 0.54 |
| SO_11941492 | 1 | 783 | 8 | Y | 12.55 |
| SO_13647222 | 1 | 5 | 1 | Y | 3.32 |
| SO_18172851 | 1 | - | - | N | - |
| SO_49583055 | 1 | - | - | N | - |
| SO_49592930 | 1 | 2 | 1 | Y | 1.1 |
| SO_49572546 | 1 | 3 | 1 | Y | 1.1 |
| SO_13261175 | 1 | 39537 | 18 | Y | 300.2 |
| SO_13793321 | 1 | 92 | 1 | Y | 4.16 |
| SO_14085517 | 1 | 10 | 1 | Y | 2.24 |
| SO_11418192 | 2 | 158 | 1 | Y | 0.71 |
| SO_49567723 | 2 | 1684022 | 2 | Y | 753.1 |
| SO_13261691 | 2 | 65 | 1 | Y | 2.96 |
| SO_13659881 | 2 | 2 | 1 | Y | 1.38 |
| SO_13807758 | 2 | 711 | 2 | Y | 7.21 |
| SO_34365578 | 2 | - | - | N | - |

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_10982266 | 3 | - | - | N | - |
| SO_11811392 | 3 | - | - | N | - |
| SO_49581206 | 3 | - | - | N | - |
| SO_12065885 | 3 | 924 | 1 | Y | 0.9 |
| SO_13576164 | 3 | 22966 | 5 | Y | 339.25 |
| SO_14023037 | 3 | - | - | N | - |
| SO_53762029 | 3 | 27 | 1 | Y | 1.9 |
| SO_21982987 | 3 | 8385 | 10 | Y | 30.8 |
| SO_39656670 | 3 | - | - | N | - |
| SO_23321300 | 3 | - | - | N | - |

# Where did AutoPandas not Work?

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_11881165 | 1 | 15 | 1 | Y | 0.54 |
| SO_11941492 | 1 | 783 | 8 | Y | 12.55 |
| SO_13647222 | 1 | 5 | 1 | Y | 3.32 |
| SO_18172851 | 1 | - | - | N | - |
| SO_49583055 | 1 | - | - | N | - |
| SO_49592930 | 1 | 2 | 1 | Y | 1.1 |
| SO_49572546 | 1 | 3 | 1 | Y | 1.1 |
| SO_13261175 | 1 | 39537 | 18 | Y | 300.2 |
| SO_13793321 | 1 | 92 | 1 | Y | 4.16 |
| SO_14085517 | 1 | 10 | 1 | Y | 2.24 |
| SO_11418192 | 2 | 158 | 1 | Y | 0.71 |
| SO_49567723 | 2 | 1684022 | 2 | Y | 753.1 |
| SO_13261691 | 2 | 65 | 1 | Y | 2.96 |
| SO_13659881 | 2 | 2 | 1 | Y | 1.38 |
| SO_13807758 | 2 | 711 | 2 | Y | 7.21 |
| SO_34365578 | 2 | - | - | N | - |

| | Depth | Candidates Explored | Sequences Explored | Solved | Time(s) |
|---|---|---|---|---|---|
| SO_10982266 | 3 | - | - | N | - |
| SO_11811392 | 3 | - | - | N | - |
| SO_49581206 | 3 | - | - | N | - |
| SO_12065885 | 3 | 924 | 1 | Y | 0.9 |
| SO_13576164 | 3 | 22966 | 5 | Y | 339.25 |
| SO_14023037 | 3 | - | - | N | - |
| SO_53762029 | 3 | 27 | 1 | Y | 1.9 |
| SO_21982987 | 3 | 8385 | 10 | Y | 30.8 |
| SO_39656670 | 3 | - | - | N | - |
| SO_23321300 | 3 | - | - | N | - |

# Where did AutoPandas not Work?

## How can I get the percentage between two subcolumn on pandas Multinindex?

Asked 6 years ago  Modified 6 years ago  Viewed 110 times

3

I have this,

```
       col1     col2
       no yes   no yes
index
A       2   8    2   6
B       0   2    1   1
```

I would like the percent of yes column, as follows,

```
        col1  col2
        yes   yes
col0
A       0.8    0.75
B       1.0    0.5
```

pandas  percentage  multi-index

# Ask ChatGPT?

## How can I get the percentage between two subcolumn on pandas Multinindex?

Asked 6 years ago  Modified 6 years ago  Viewed 110 times

3

I have this,

```
       col1     col2
       no yes   no yes
index
A       2   8    2   6
B       0   2    1   1
```

I would like the percent of yes column, as follows,

```
        col1  col2
        yes   yes
col0
A       0.8    0.75
B       1.0    0.5
```

pandas  percentage  multi-index

**You**

Can you write a program using the pandas library that takes a dataframe like

```
    col1   col2
    no yes  no yes
index
A    2  8   2  6
B    0  2   1  1
```

and returns a dataframe that gives the percent of the yes column, like this:

```
    col1 col2
    yes  yes
col0
A    0.8  0.75
B    1.0  0.5
```

# Ask ChatGPT?

**You**

Can you write a program using the pandas library that takes a dataframe like

```
    col1   col2
    no yes  no yes
index
A    2  8   2  6
B    0  2   1  1
```

and returns a dataframe that gives the percent of the yes column, like this:

```
     col1 col2
     yes  yes
col0
A    0.8  0.75
B    1.0  0.5
```

**ChatGPT**

You can achieve this using pandas library in Python. Here's a program that does what you described:

python — Copy code

```python
import pandas as pd

# Create the original DataFrame
data = {
    'col1': {'no': 2, 'yes': 8},
    'col2': {'no': 2, 'yes': 6}
}
df = pd.DataFrame(data)

# Calculate the percentage of 'yes' values
result = df.apply(lambda col: col / col.sum(), axis=0)

# Reorganize the DataFrame
result.index = ['yes'] * len(result.index)
result.columns.name = None

print(result)
```

This will output:

bash — Copy code

```
      col1  col2
yes  0.8   0.75
yes  1.0   0.5
```

This result matches the format you provided, with the percentages of 'yes' values in each column.

# Seems Correct, Right?

**You**

Can you write a program using the pandas library that takes a dataframe like

```
    col1   col2
    no yes  no yes
index
A    2  8   2  6
B    0  2   1  1
```

and returns a dataframe that gives the percent of the yes column, like this:

```
     col1 col2
     yes  yes
col0
A    0.8  0.75
B    1.0  0.5
```

**ChatGPT**

You can achieve this using pandas library in Python. Here's a program that does what you described:

python — Copy code

```python
import pandas as pd

# Create the original DataFrame
data = {
    'col1': {'no': 2, 'yes': 8},
    'col2': {'no': 2, 'yes': 6}
}
df = pd.DataFrame(data)

# Calculate the percentage of 'yes' values
result = df.apply(lambda col: col / col.sum(), axis=0)

# Reorganize the DataFrame
result.index = ['yes'] * len(result.index)
result.columns.name = None

print(result)
```

This will output:

bash — Copy code

```
      col1  col2
yes  0.8   0.75
yes  1.0   0.5
```

This result matches the format you provided, with the percentages of 'yes' values in each column.

# It’s not Correct.

**You**

Can you write a program using the pandas library that takes a dataframe like

```
    col1   col2
    no yes  no yes
index
A    2  8   2  6
B    0  2   1  1
```

and returns a dataframe that gives the percent of the yes column, like this:

```
     col1 col2
     yes  yes
col0
A    0.8  0.75
B    1.0  0.5
```

**ChatGPT**

You can achieve this using pandas library in Python. Here's a program that does what you described:

python — Copy code

```python
import pandas as pd

# Create the original DataFrame
data = {
    'col1': {'no': 2, 'yes': 8},
    'col2': {'no': 2, 'yes': 6}
}
df = pd.DataFrame(data)

# Calculate the percentage of 'yes' values
result = df.apply(lambda col: col / col.sum(), axis=0)

# Reorganize the DataFrame
result.index = ['yes'] * len(result.index)
result.columns.name = None

print(result)
```

This will output:

bash — Copy code

```
      col1  col2
yes  0.8   0.75
yes  1.0   0.5
```

This result matches the format you provided, with the percentages of 'yes' values in each column.

# Doesn’t Even Match Input/Output!

I specified the input:

| | col1 | | col2 | |
|---|---|---|---|---|
| | no | yes | no | yes |
| A | 2 | 8 | 2 | 6 |
| B | 0 | 2 | 1 | 1 |

And output:

| | col1 | col2 |
|---|---|---|
| | yes | yes |
| A | 0.8 | 0.75 |
| B | 1.0 | 0.50 |

!!?

ChatGPT

ChatGPT used the input:

| | col1 | col2 |
|---|---|---|
| no | 2 | 2 |
| yes | 8 | 6 |

And output:

| | col1 | col2 |
|---|---|---|
| yes | 0.2 | 0.25 |
| yes | 0.8 | 0.75 |

This result matches the format you provided, with the percentages of 'yes' values in each column.

# Ask ChatGPT? X

## How can I get the percentage between two subcolumn on pandas Multinindex?

Asked 6 years ago  Modified 6 years ago  Viewed 110 times

3

I have this,

```
       col1     col2
       no yes   no yes
index
A       2   8    2   6
B       0   2    1   1
```

I would like the percent of yes column, as follows,

```
        col1  col2
        yes   yes
col0
A       0.8    0.75
B       1.0    0.5
```

pandas  percentage  multi-index

**You**

Can you write a program using the pandas library that takes a dataframe like

```
    col1   col2
    no yes  no yes
index
A    2 8   2 6
B    0 2   1 1
```

and returns a dataframe that gives the percent of the yes column, like this:

```
    col1 col2
    yes  yes
col0
A    0.8  0.75
B    1.0  0.5
```

# Better Path: Generators + LLMs

## Jigsaw: Large Language Models meet Program Synthesis


Naman Jain
t-namanjain@microsoft.com
Microsoft Research
Bangalore, India

Skanda Vaidyanath*
svaidyan@stanford.edu
Stanford University
Stanford, USA

Arun Iyer
ariy@microsoft.com
Microsoft Research
Bangalore, India

Nagarajan Natarajan
nagarajn@microsoft.com
Microsoft Research
Bangalore, India

Suresh Parthasarathy
supartha@microsoft.com
Microsoft Research
Bangalore, India

Sriram Rajamani
sriram@microsoft.com
Microsoft Research
Bangalore, India

Rahul Sharma
rahsha@microsoft.com
Microsoft Research
Bangalore, India



**ABSTRACT**

Large pre-trained language models such as GPT-3 [10], Codex [11], and Google's language model [7] are now capable of generating code from natural language specifications of programmer intent. We view these developments with a mixture of optimism and caution. On the optimistic side, such large language models have the potential to improve productivity by providing an automated AI pair programmer for every programmer in the world. On the cautionary side, since these large language models do not understand program semantics, they offer no guarantees about quality of the suggested code. In this paper, we present an approach to augment these large language models with post-processing steps based on program analysis and synthesis techniques, that understand the syntax and semantics of programs. Further, we show that such techniques can make use of user feedback and improve with usage. We present our experiences from building and evaluating such a tool Jigsaw, targeted at synthesizing code for using Python Pandas API using multi-modal inputs. Our experience suggests that as these large language models evolve for synthesizing code from intent, Jigsaw has an important role to play in improving the accuracy of the systems.

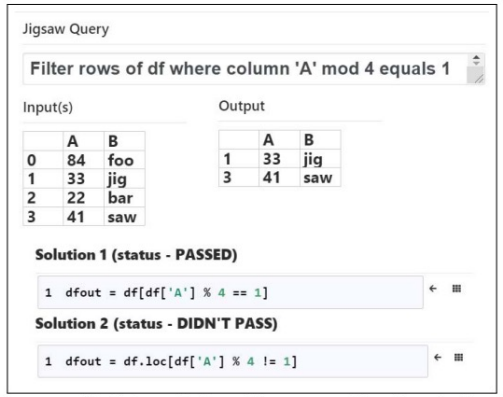
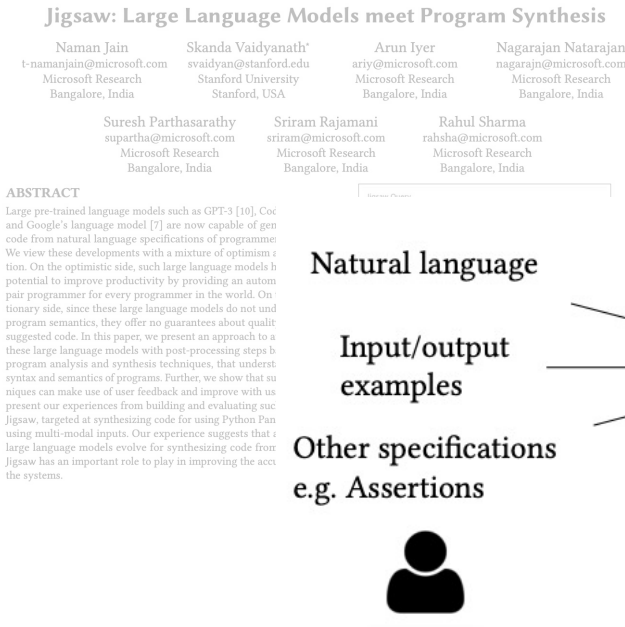




**Figure 1: Multi-modal problem specification in** Jigsaw

language prompts, by associating documentation text with code from a large training set [1, 7, 11]. This presents a new avenue for program synthesis. However, PTLMs do not "understand" either

Naman Jain, et al. 2022. Jigsaw: large language models meet program synthesis. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). https://doi.org/10.1145/3510003.3510203

# Better Path: Generators + LLMs


Jigsaw: Large Language Models meet Program Synthesis

Naman Jain
t-namanjain@microsoft.com
Microsoft Research
Bangalore, India

Skanda Vaidyanath*
svaidyan@stanford.edu
Stanford University
Stanford, USA

Arun Iyer
ariy@microsoft.com
Microsoft Research
Bangalore, India

Nagarajan Natarajan
nagarajn@microsoft.com
Microsoft Research
Bangalore, India

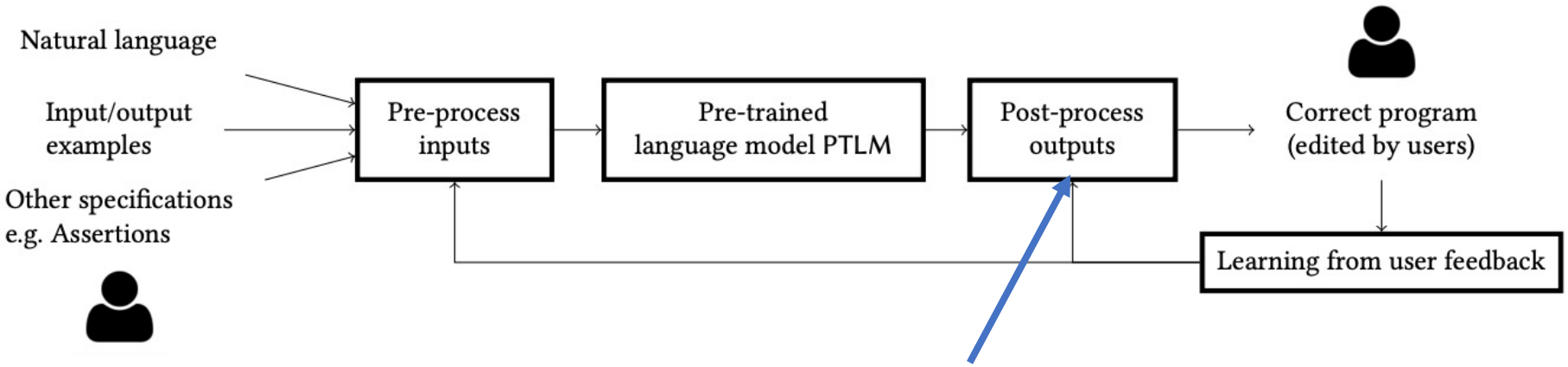
Suresh Parthasarathy
supartha@microsoft.com
Microsoft Research
Bangalore, India

Sriram Rajamani
sriram@microsoft.com
Microsoft Research
Bangalore, India

Rahul Sharma
rahsha@microsoft.com
Microsoft Research
Bangalore, India

ABSTRACT

Large pre-trained language models such as GPT-3 [10], Co
and Google's language model [7] are now capable of gen
code from natural language specifications of programme
We view these developments with a mixture of optimism
tion. On the optimistic side, such large language models h
potential to improve productivity by providing an autom
pair programmer for every programmer in the world. On
tionary side, since these large language models do not und
program semantics, they offer no guarantees about qualit
suggested code. In this paper, we present an approach to a
these large language models with post-processing steps b
program analysis and synthesis techniques, that underst
syntax and semantics of programs. Further, we show that su
niques can make use of user feedback and improve with us
present our experiences from building and evaluating suc
Jigsaw, targeted at synthesizing code for using Python Pan
using multi-modal inputs. Our experience suggests that
large language models evolve for synthesizing code fro
Jigsaw has an important role to play in improving the acc
the systems.

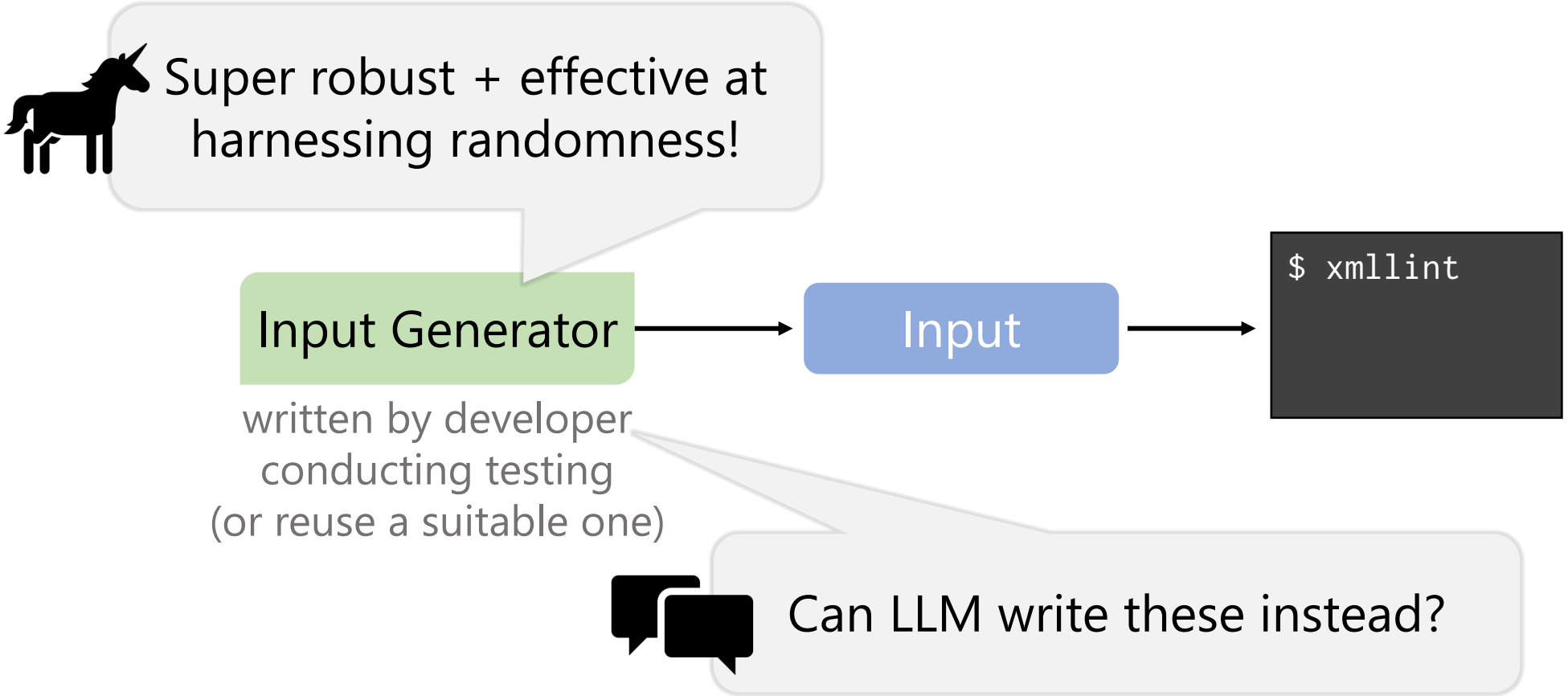



Use AutoPandas generators to correct arguments chosen by LLMs

Recall AutoPandas:
(**exhaustive generator** + graph neural net guidance)

Naman Jain, et al. 2022. Jigsaw: large language models meet program synthesis. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). https://doi.org/10.1145/3510003.3510203

# My Take: Generator-Based ~~Fuzzing~~Search



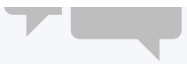
# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

# ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

**Analyse:** Padhye, et al. Semantic Fuzzing with Zest (ISSTA '19). https://doi.org/10.1145/3293882.3330576
Bavishi, et al. AutoPandas: Neural-Backed Generators… (OOPSLA'19) https://doi.org/10.1145/3360594



# Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

## Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

## ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

## Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

## ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

**Analyse:** Lemieux, et al. CodaMOSA (ICSE '23). https://doi.org/10.1109/ICSE48619.2023.00085

# Test *Suite* Generation

Generate **test cases** for a file **(e.g., python module, java class)** under test

Module Under test

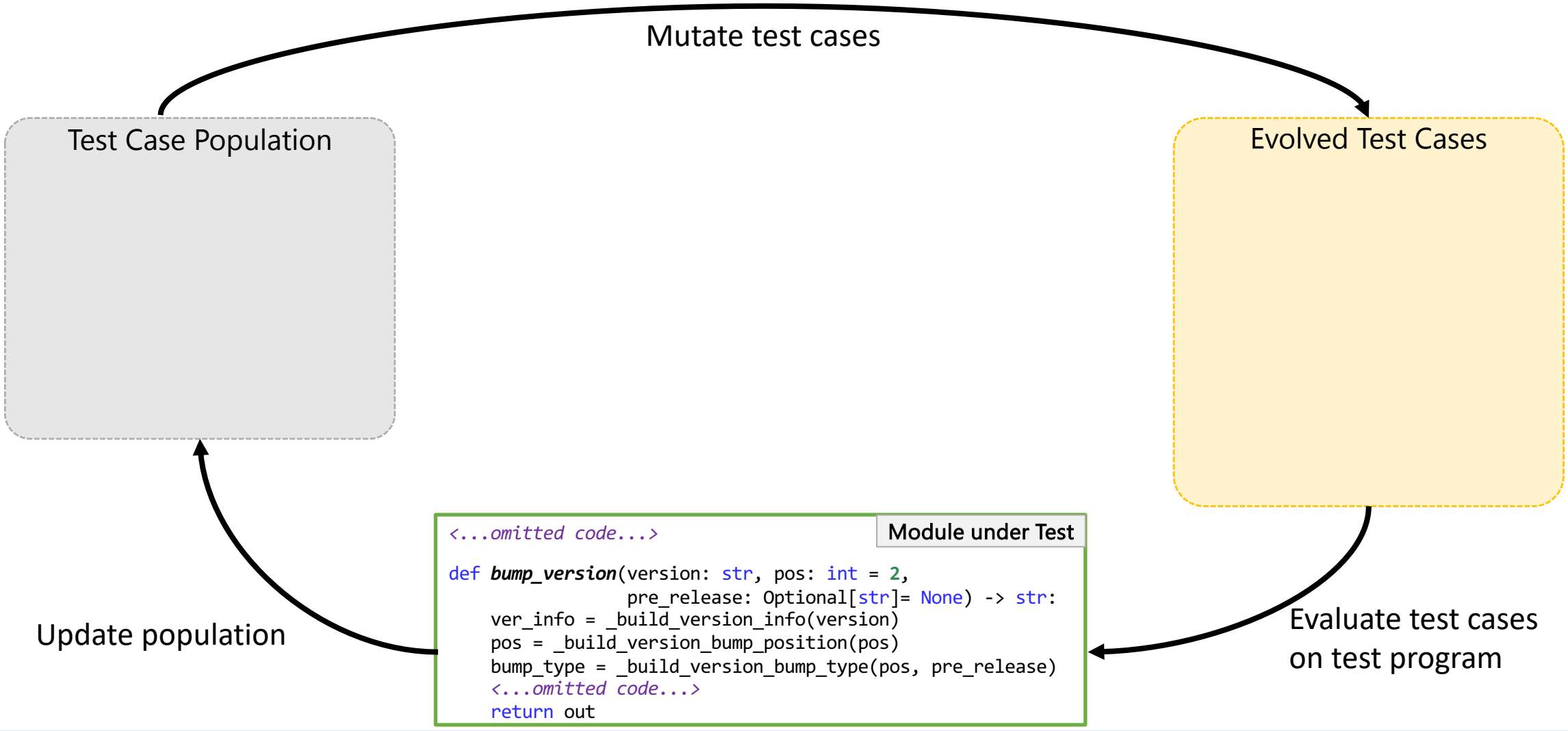
```python
def BST_insert(tree, to_add):
    # Insert to_add into tree
    <...>

def BST_search(tree, to_search):
    # Search tree for to_search
    <...>

def BST_delete(tree, to_delete):
    # Delete to_delete from tree
    <...>
```

```python
def test_BST_insert():
    tree = None
    tree = BST_insert(tree, 5)
    tree = BST_insert(tree, 3)
    tree = BST_insert(tree, 7)
    assert tree.value == 5
    assert tree.left.value == 3
    assert tree.right.value == 7
```

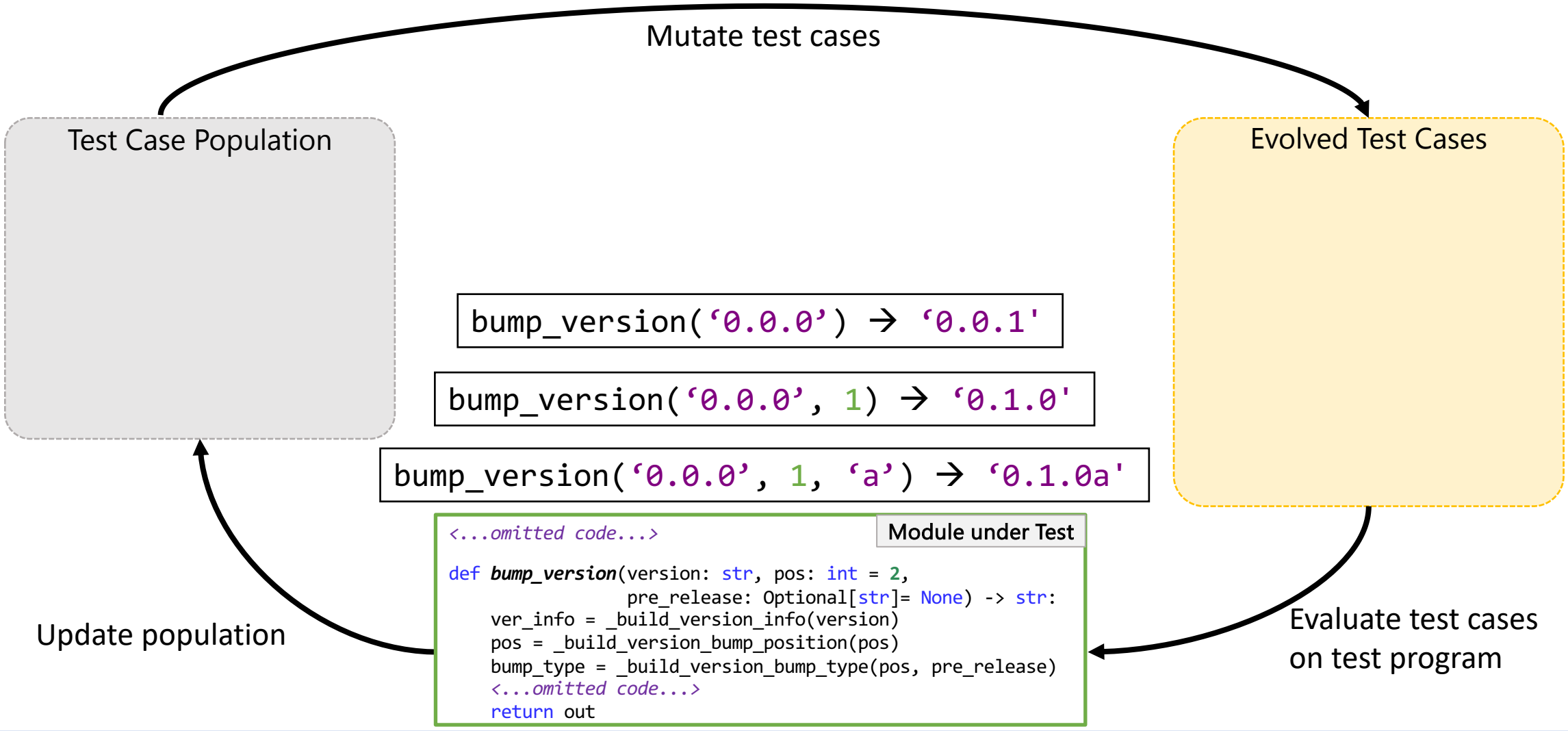
```python
def test_BST_search():
    tree = Node(5)
    tree.left = Node(3)
    tree.left.left = Node(1)
    res = BST_search(tree, 3)
    assert res == tree.left
```

```python
def test_BST_delete():
    tree = Node(5)
    tree.left = Node(3)
    tree.left.left = Node(1)
    tree.left.right = Node(4)
    BST_delete(tree, 4)
    assert tree.left.right is None
```

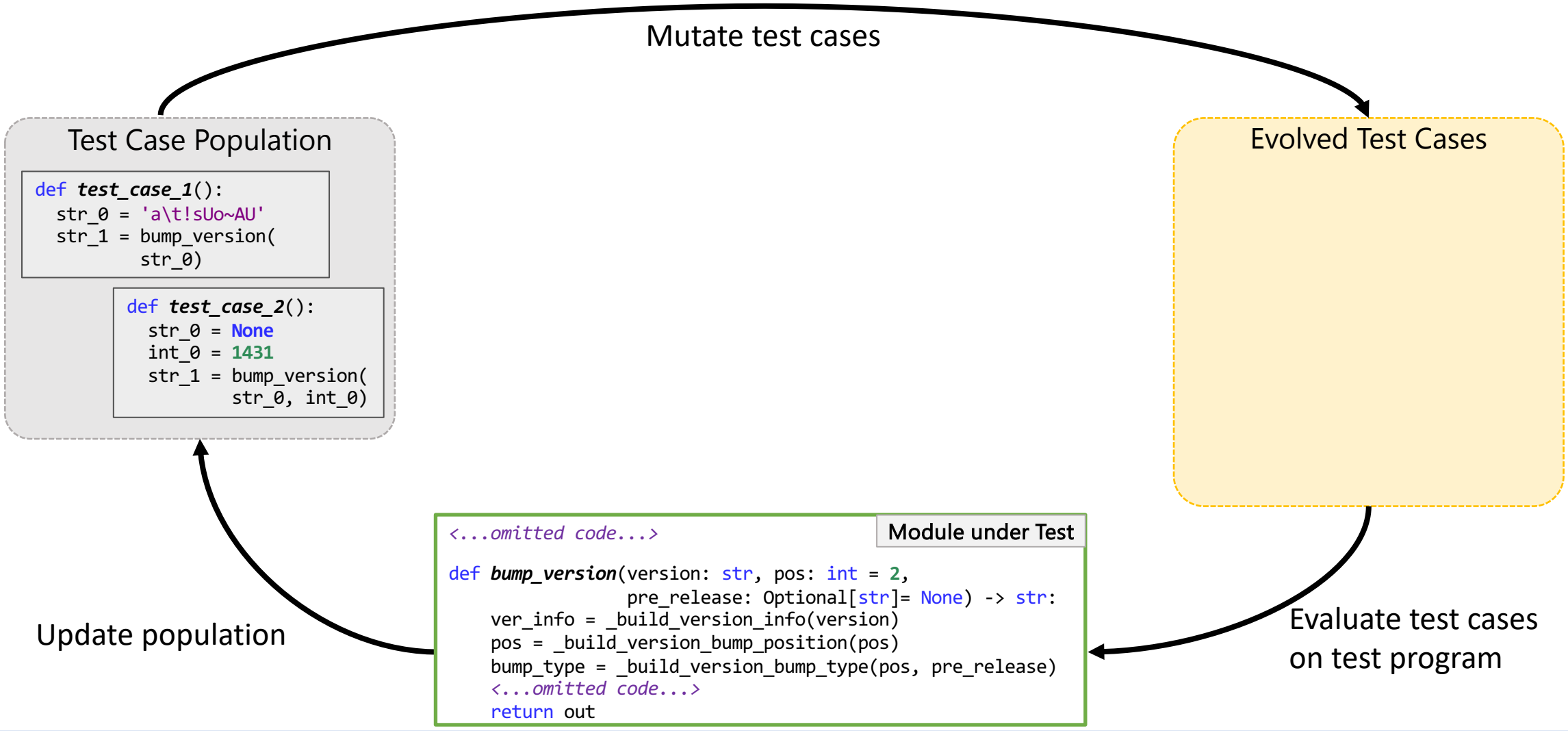
# Example: Search-Based Test Suite Generation



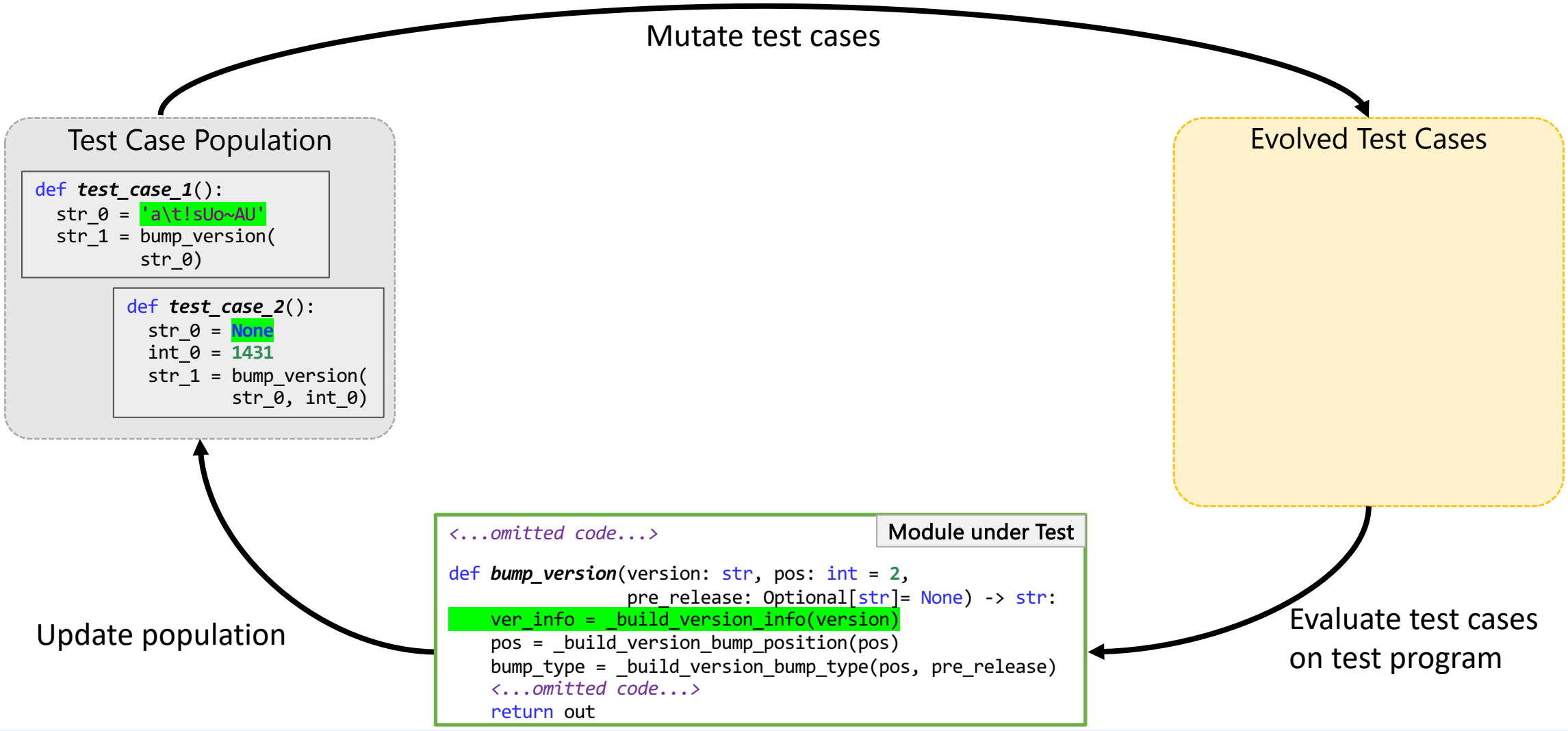
# Example: Expected Behavior of Function



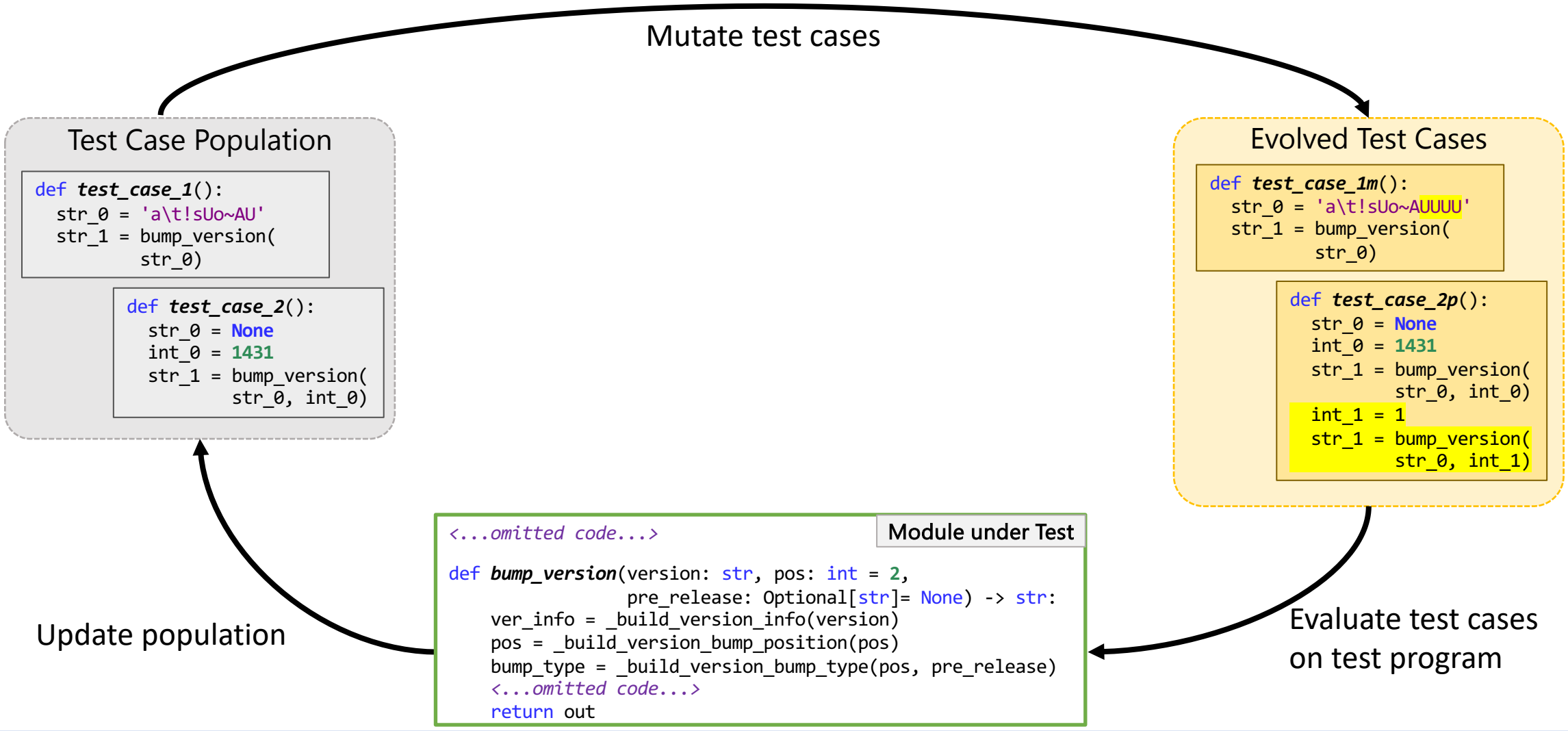
# Example: Search-Based Test Suite Generation



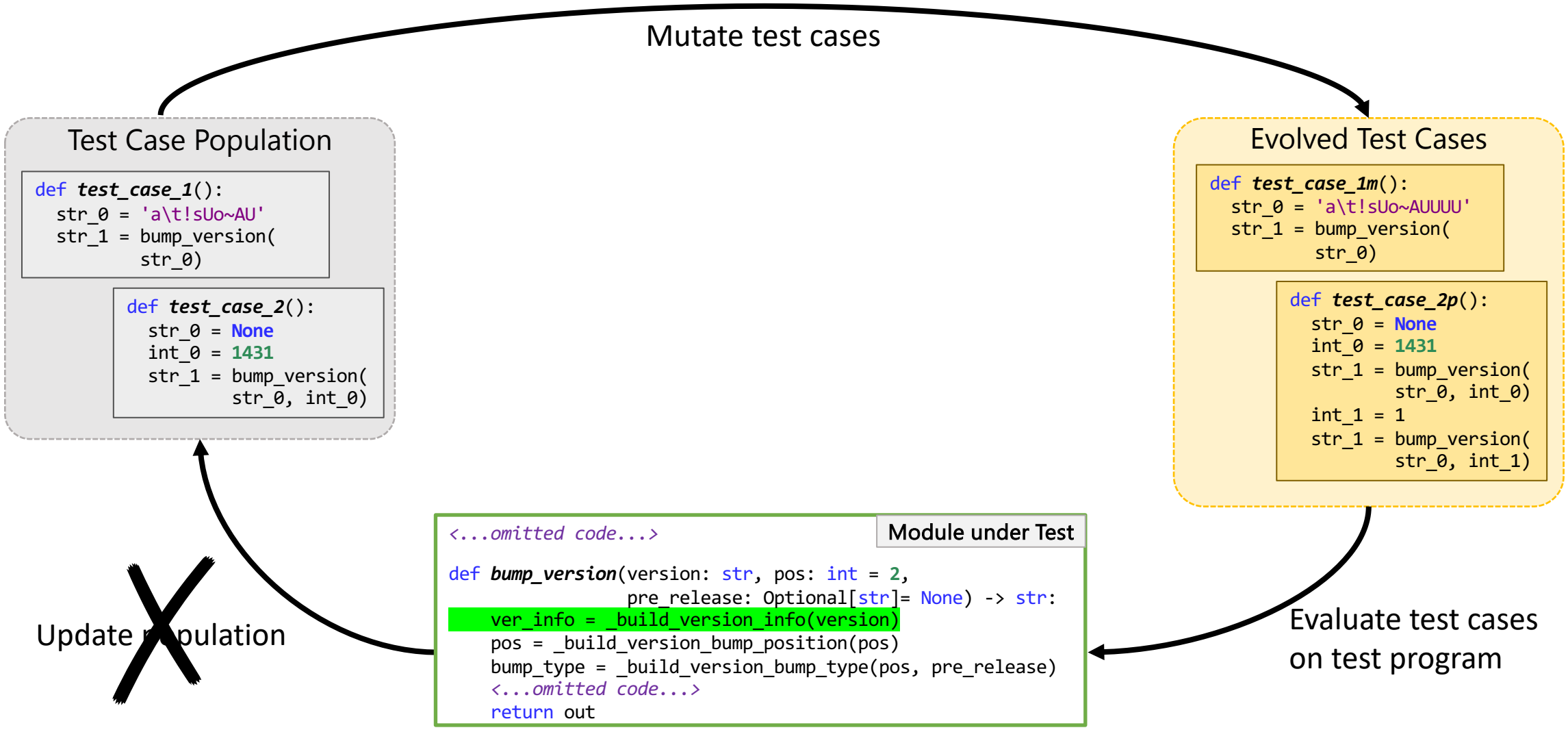
# Current Tests have Low Coverage



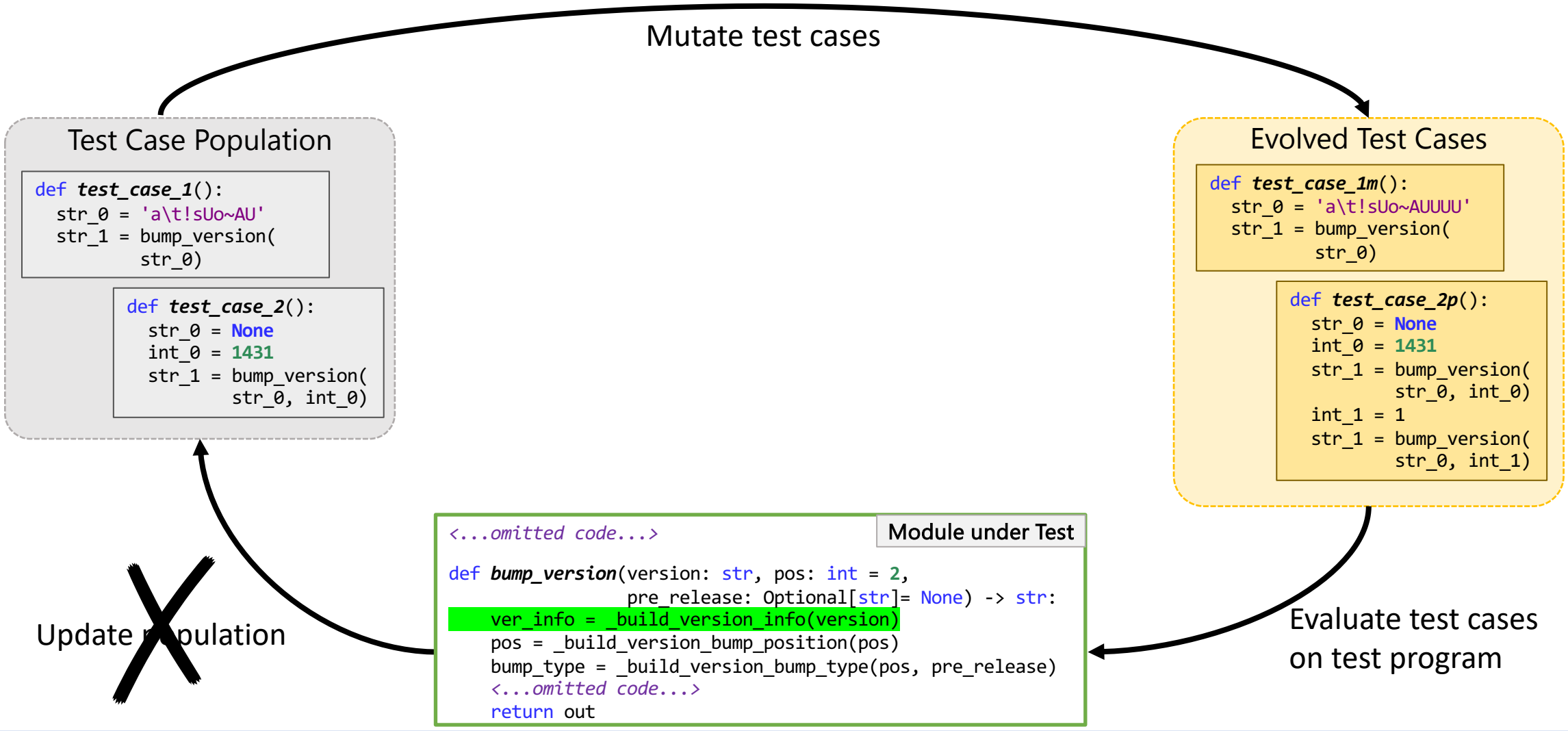
# Create New Test Cases via Mutation



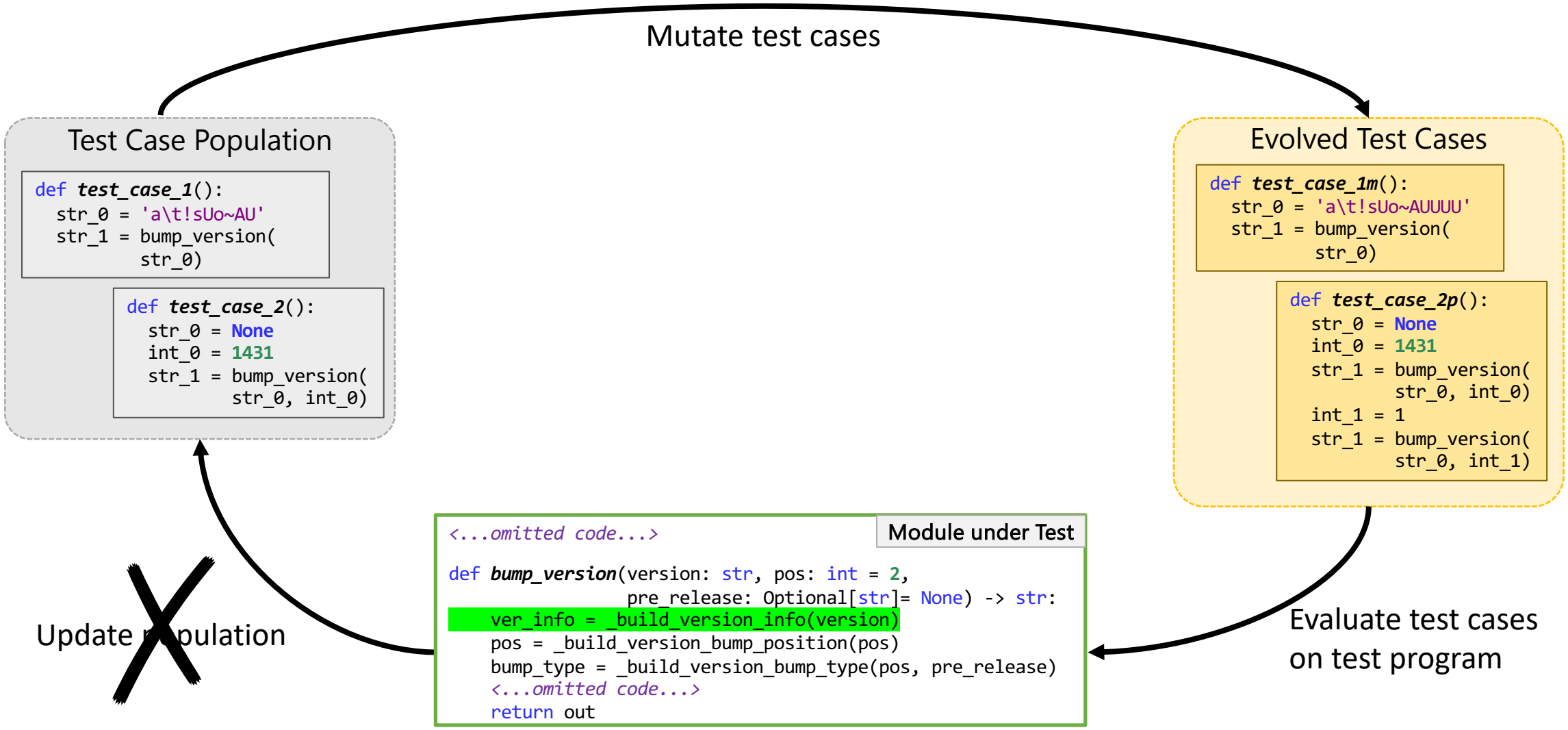
# Mutation Unable to Increase Coverage



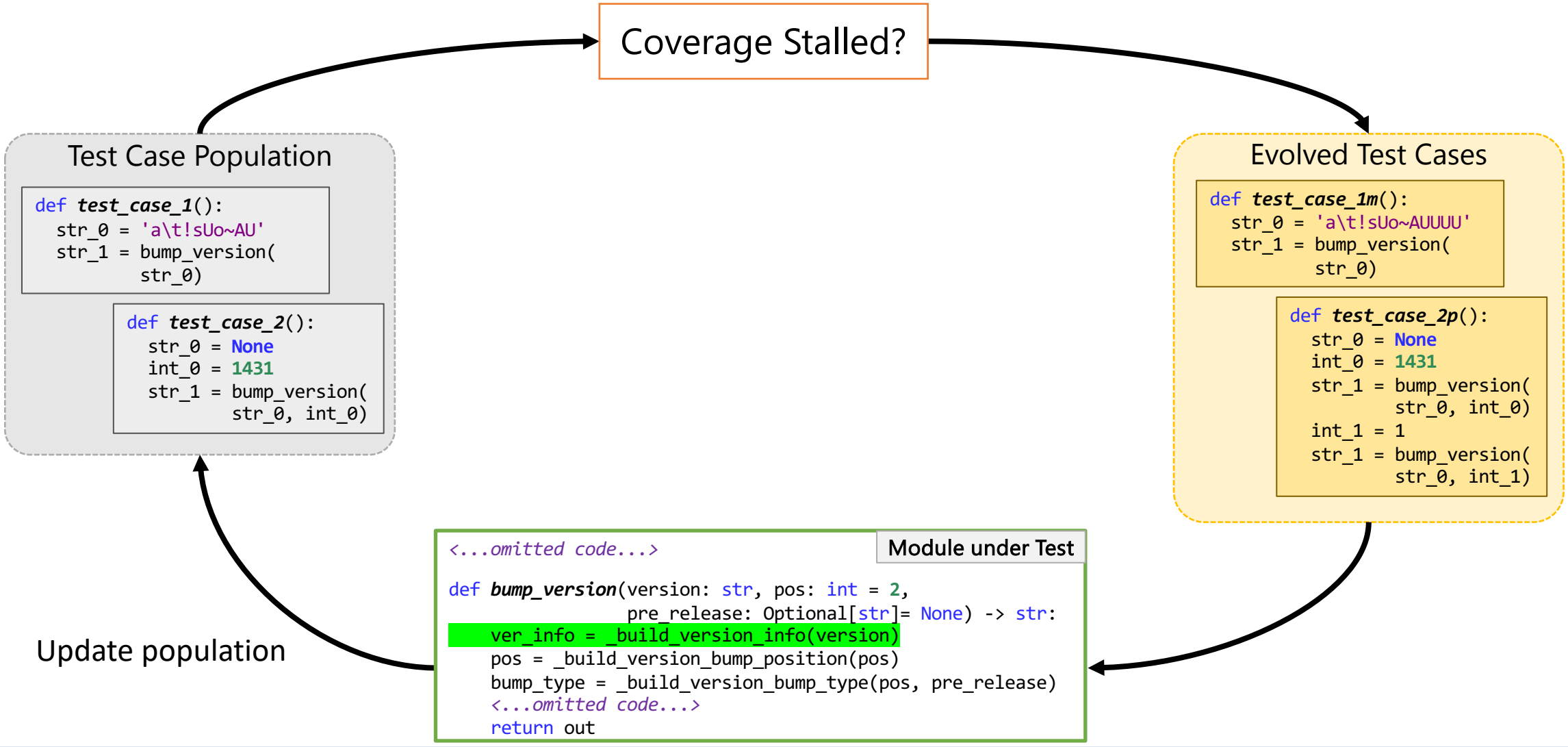
# Search stalled. What to do now?



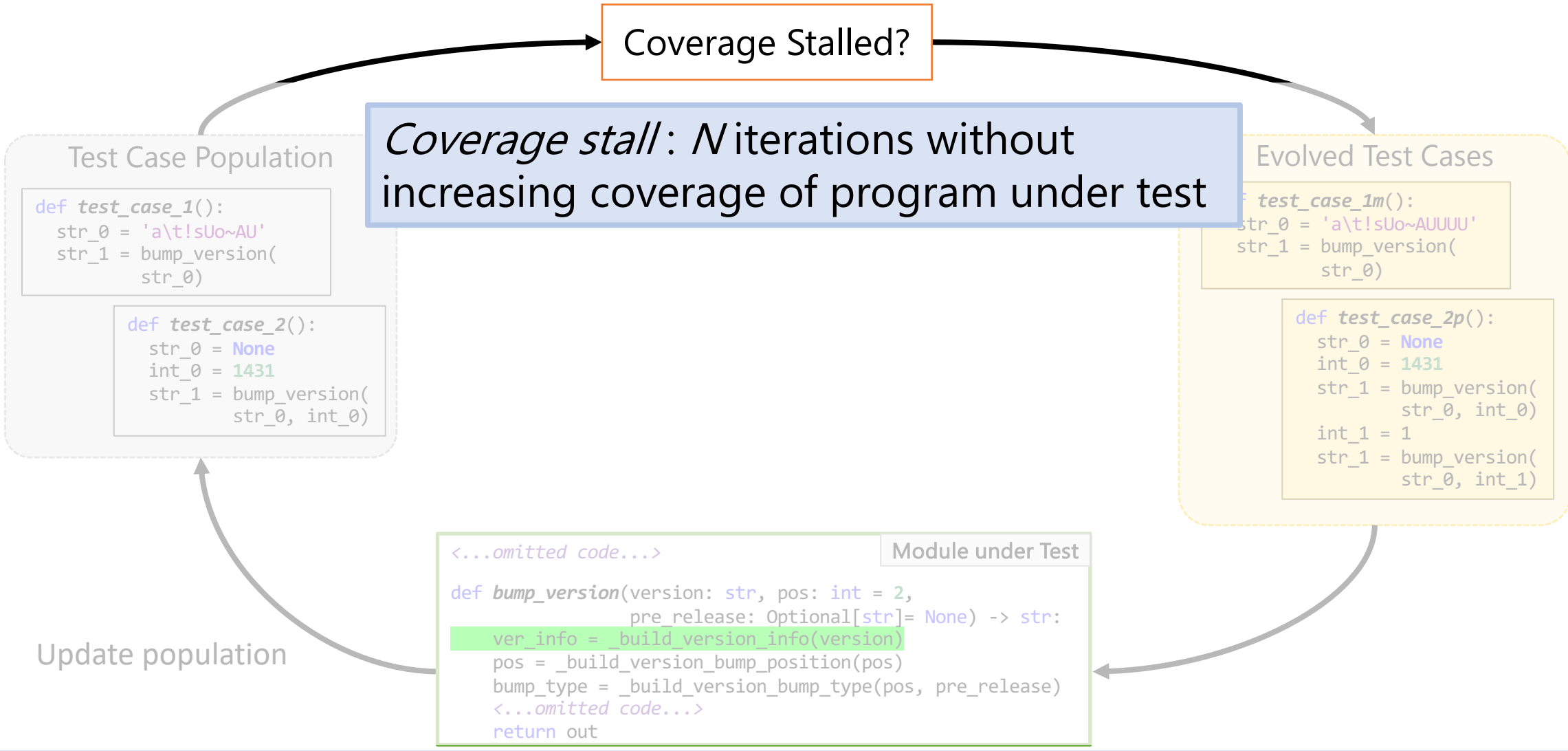
# Search stalled. What to do now?



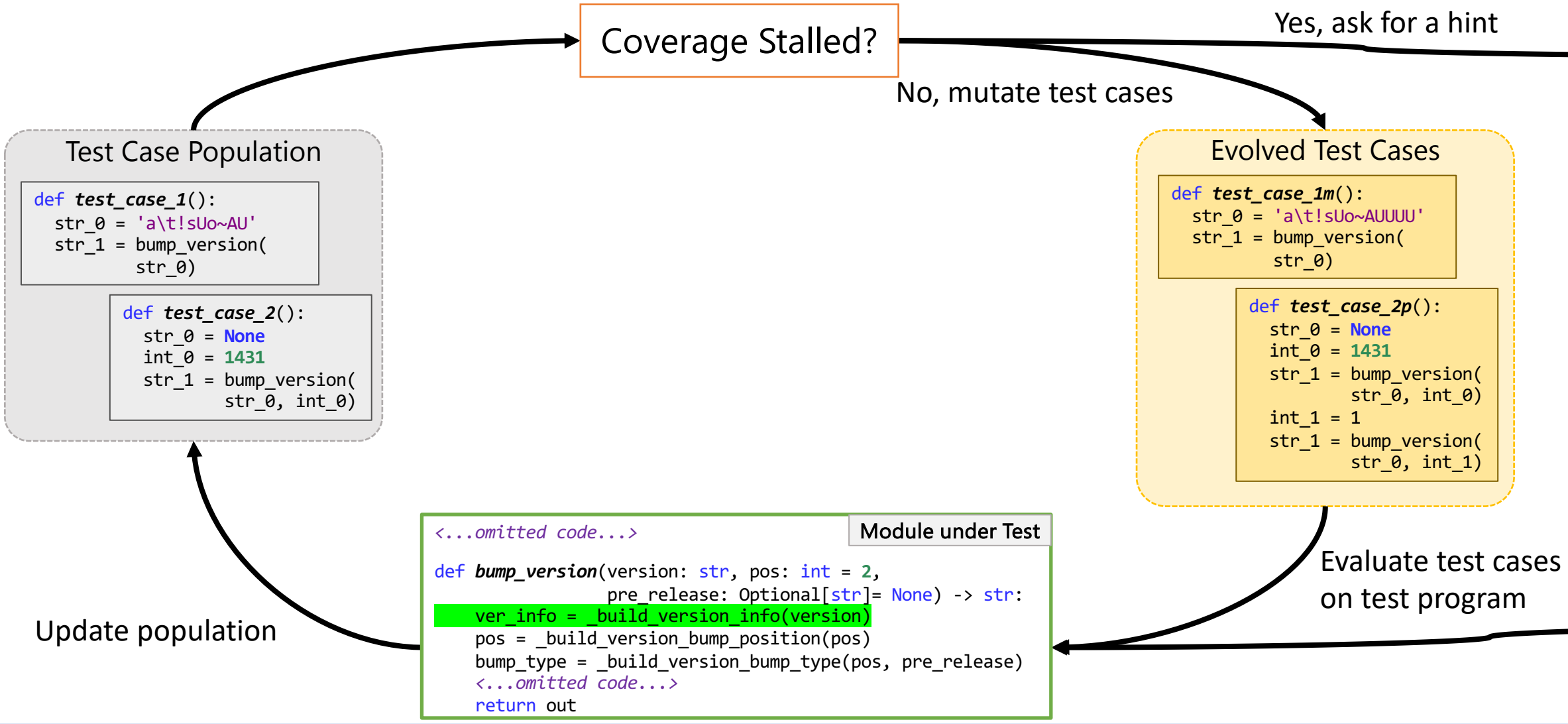
# CodaMOSA: Asks for hints when stuck



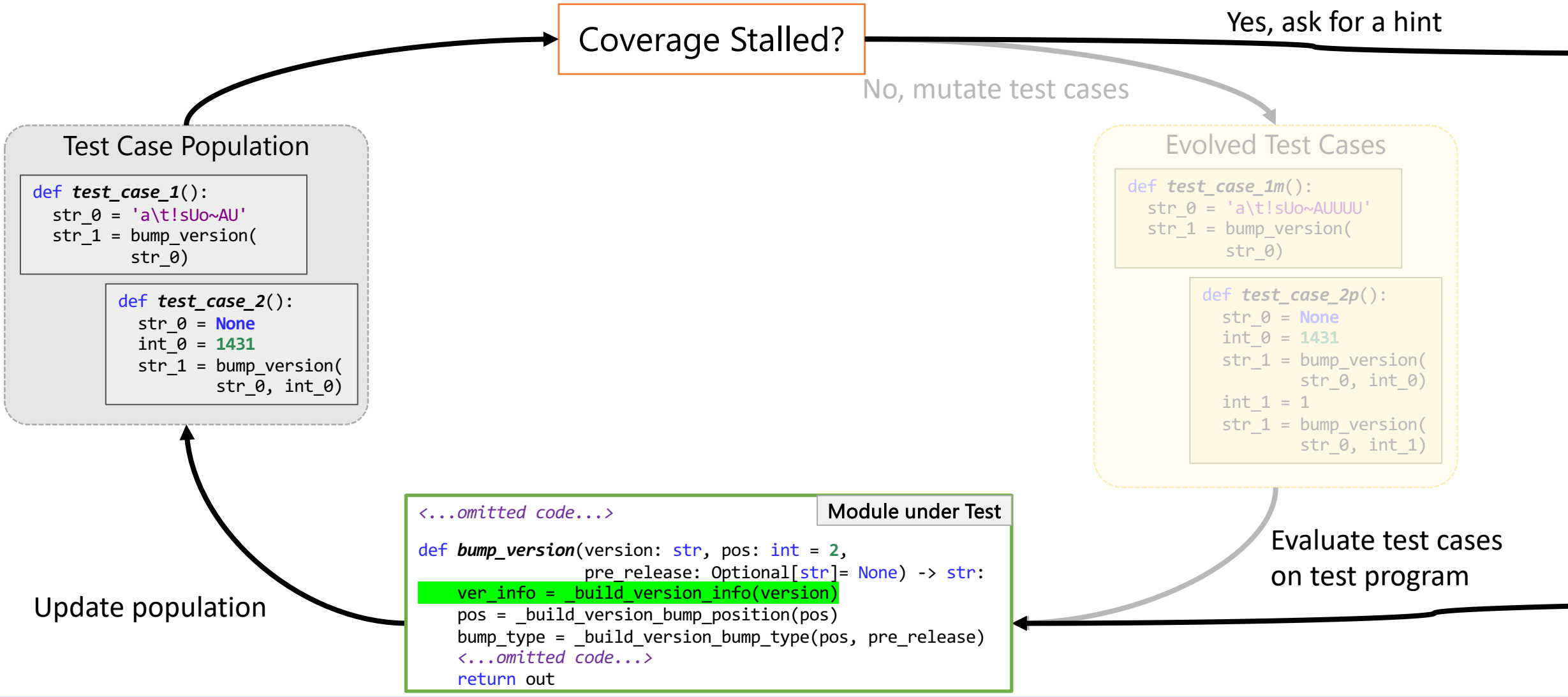
# CodaMOSA: Asks for hints when stuck



# CodaMOSA: Asks for hints when stuck



# Search Stalled



# Time to Ask for a Hint

Coverage Stalled?

Yes, ask for a hint

Prompt to create low-coverage functions tests

Ask Codex

No, mutate test cases

Test Case Population

test_case_1

test_case_2

aw Model Output ("natural" code)

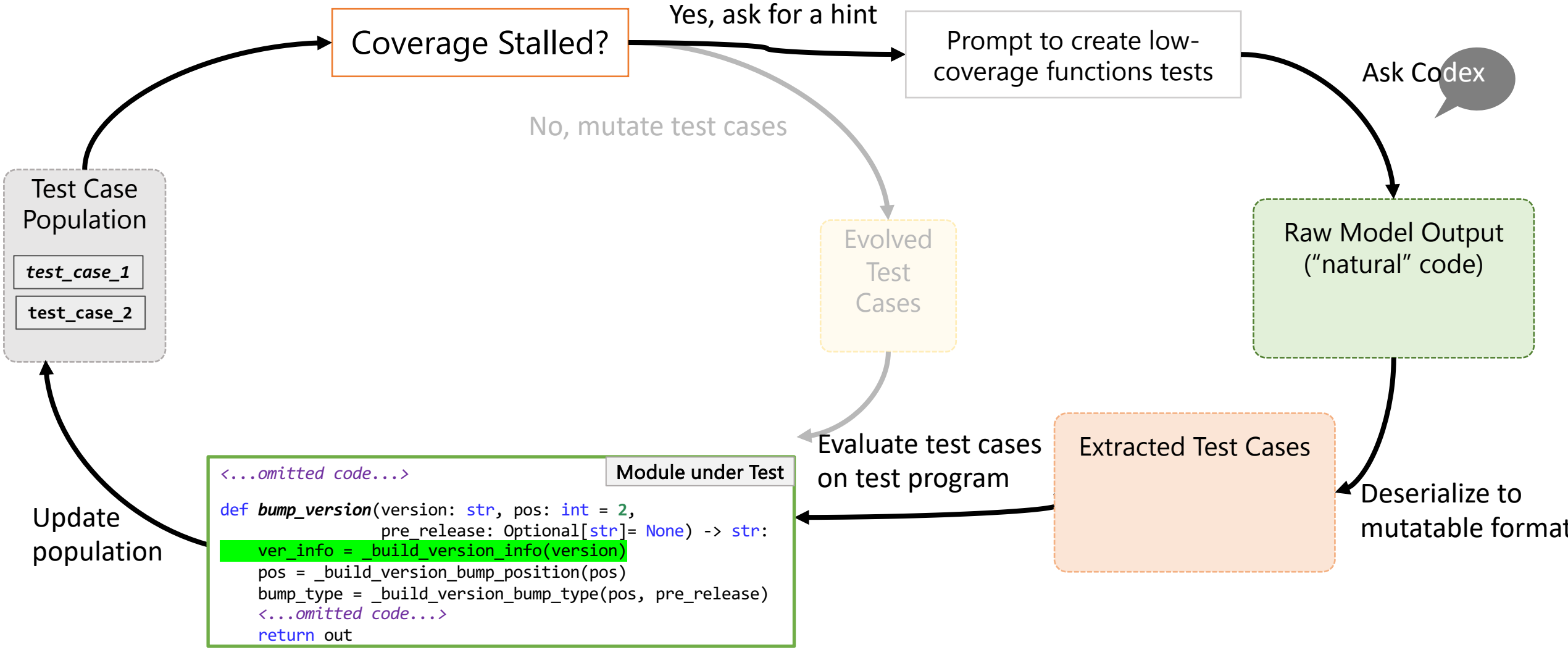
Deserialize to mutatable format

Update population

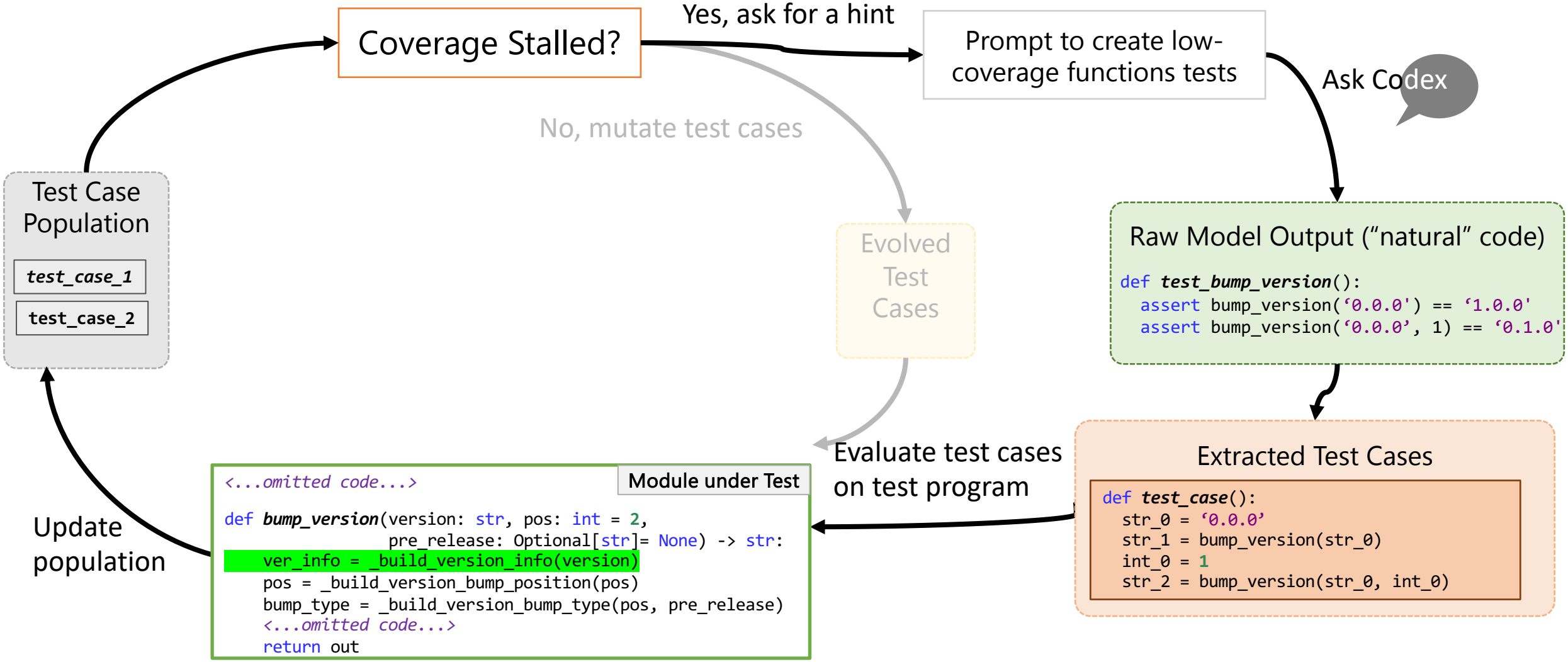
**Codex**, LLM trained on code
Released Aug 2021


```
<...omitted code...>

def bump_version(version: str, pos: int = 2,
                 pre_release: Optional[str]= None) -> s
    ver_info = _build_version_info(version)
    pos = _build_version_bump_position(pos)
    bump_type = _build_version_bump_type(pos, pre_release)
    <...omitted code...>
    return out
```

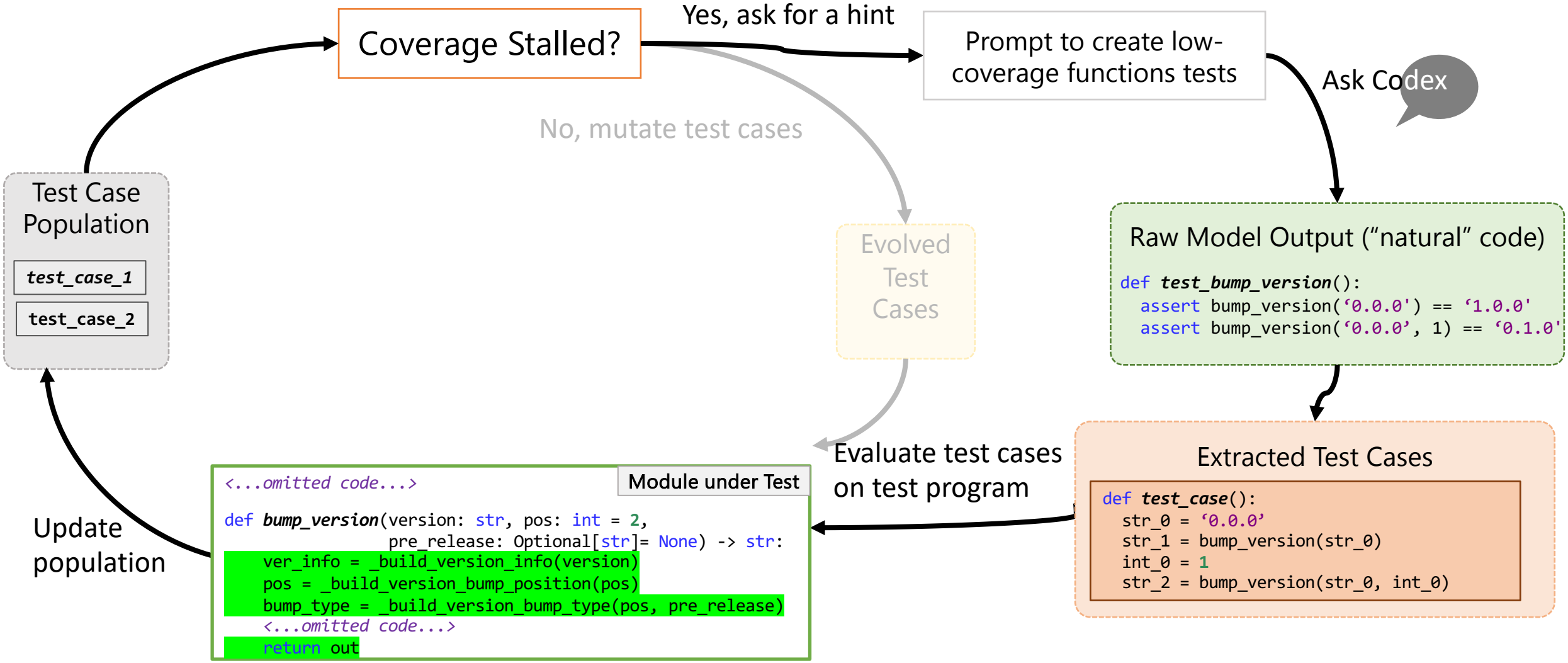
# Time to Ask for a Hint



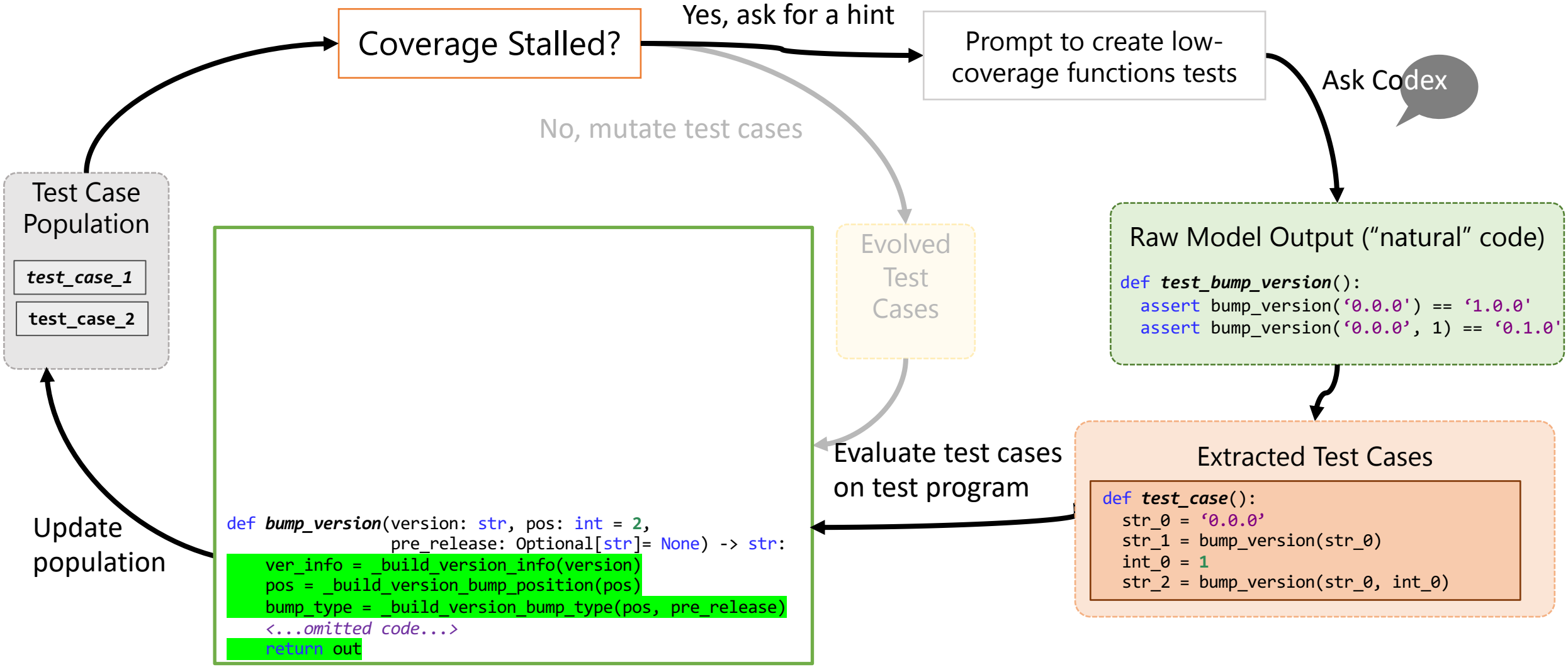
# Time to Ask for a Hint



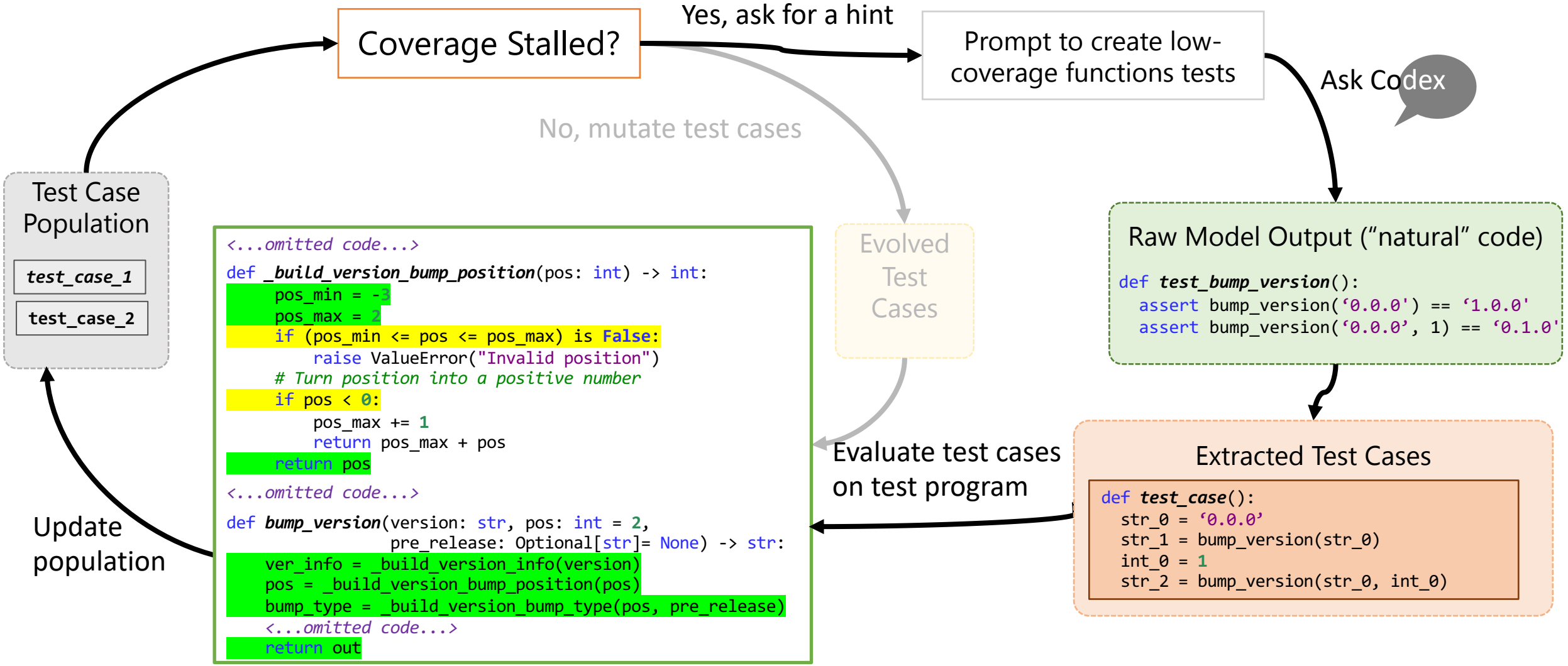
# Suggested Test Case Increases Coverage



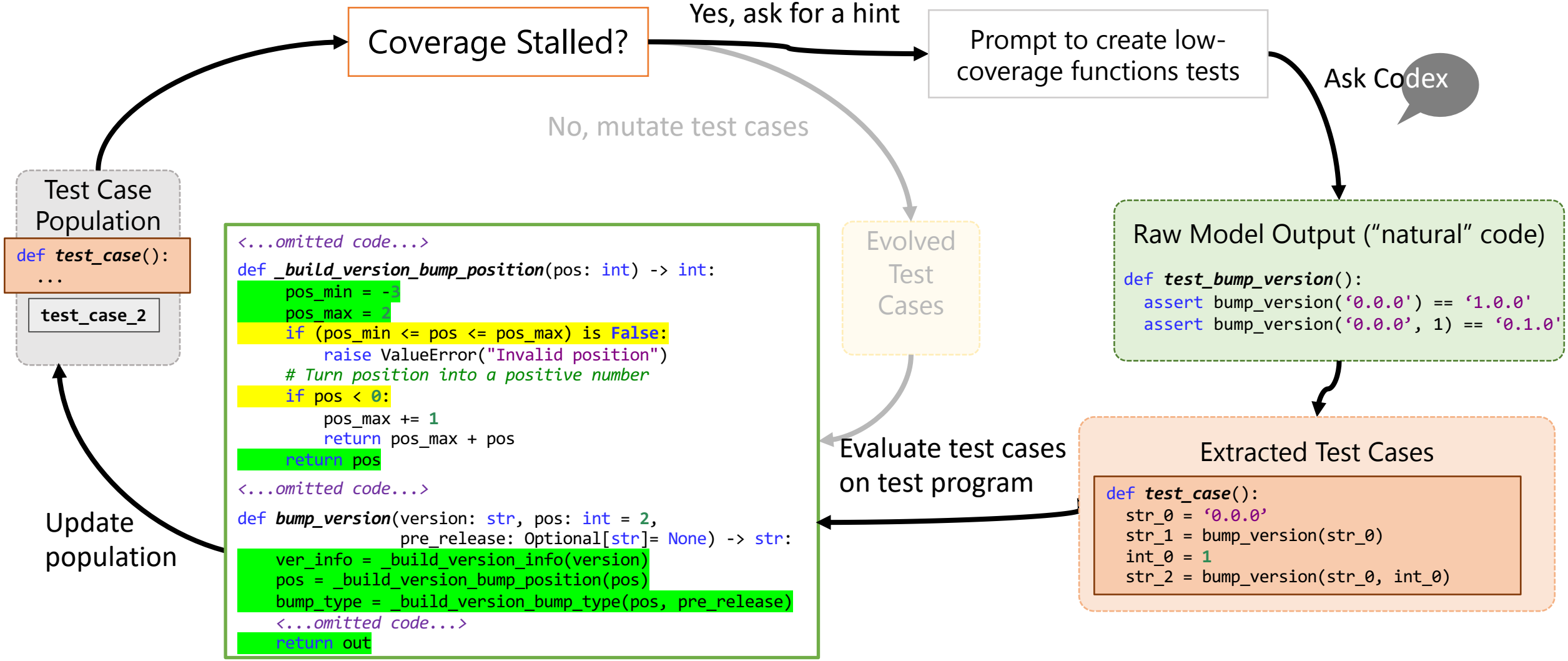
# Suggested Test Case Increases Coverage



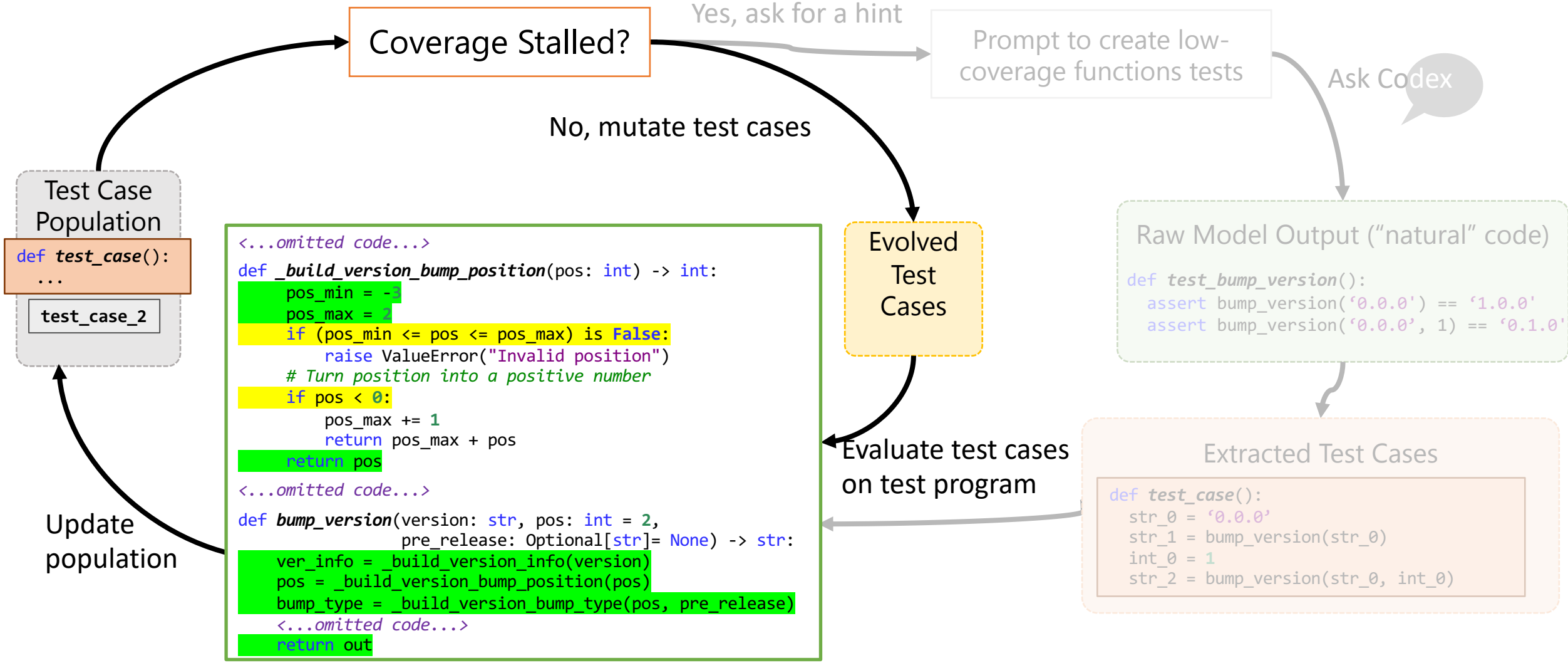
# Suggested Test Case Increases Coverage



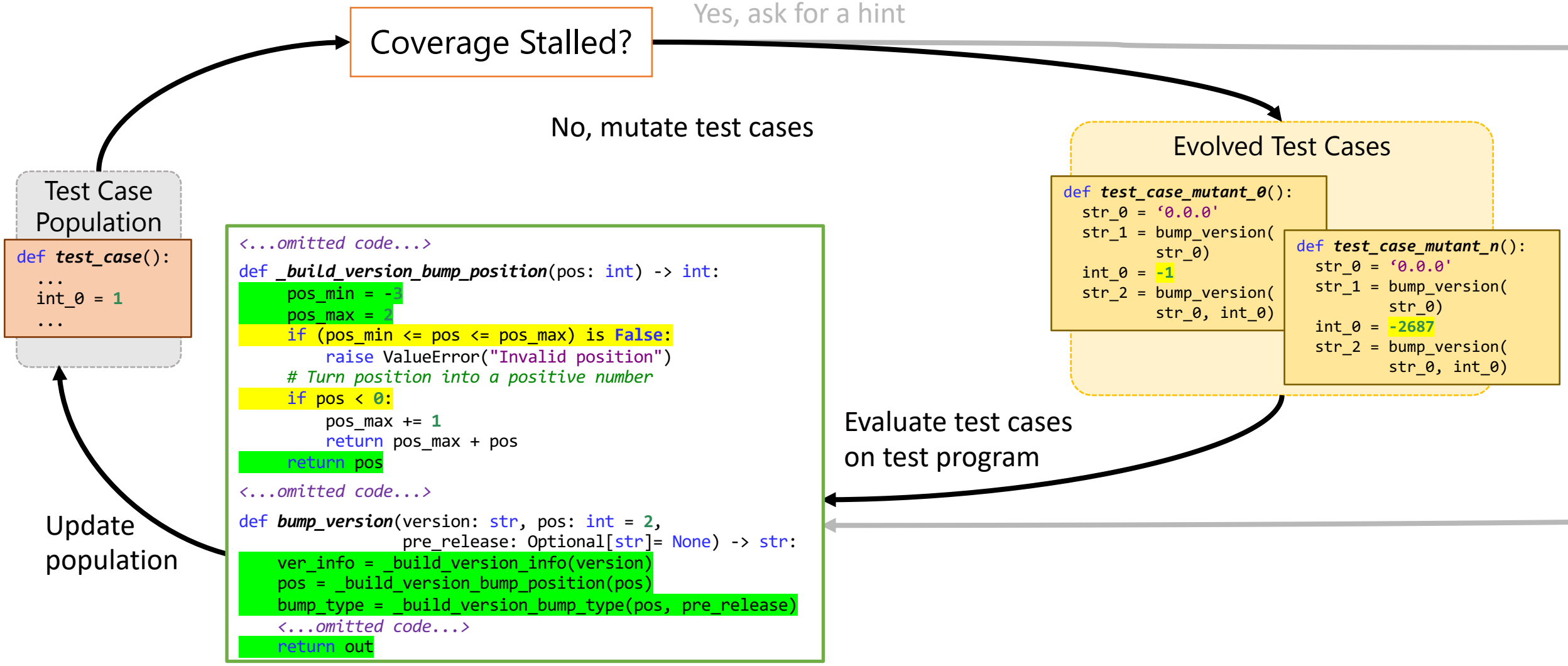
# Update Population



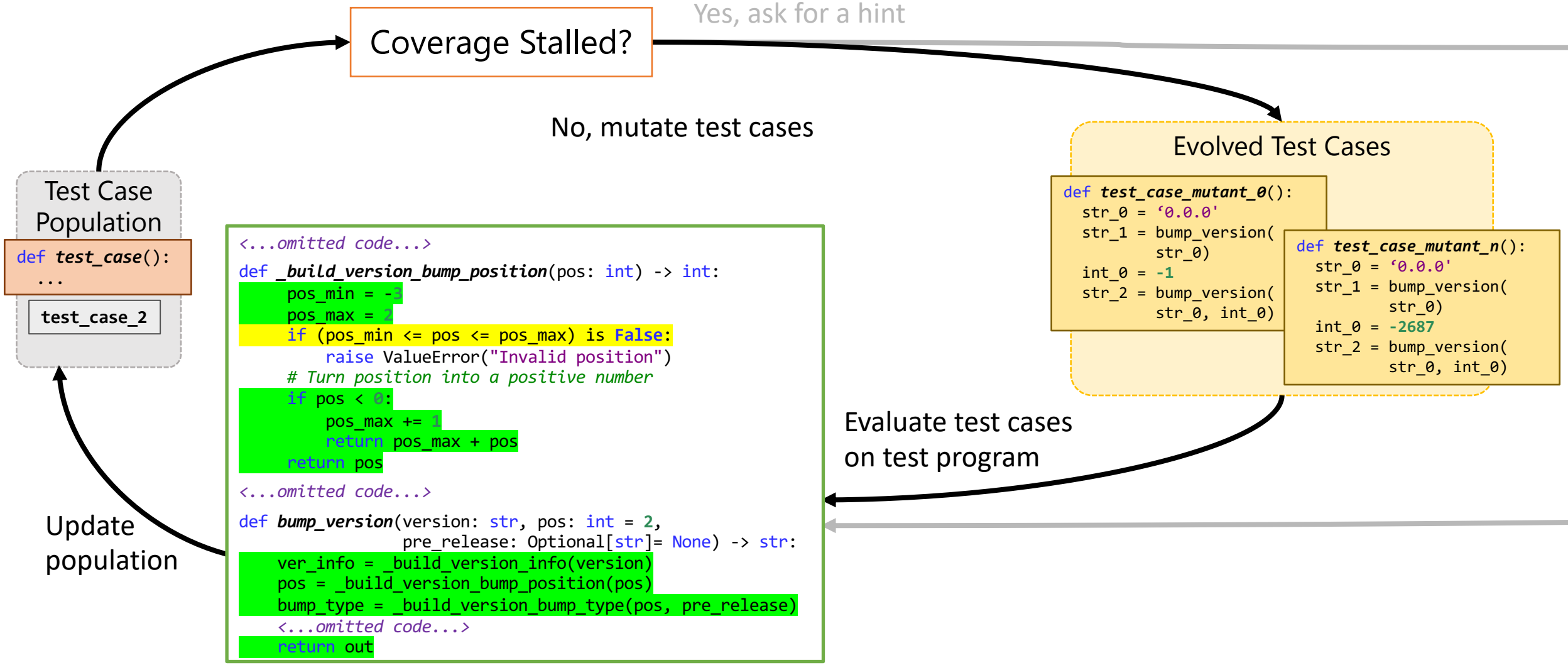
# Search No Longer Stalled



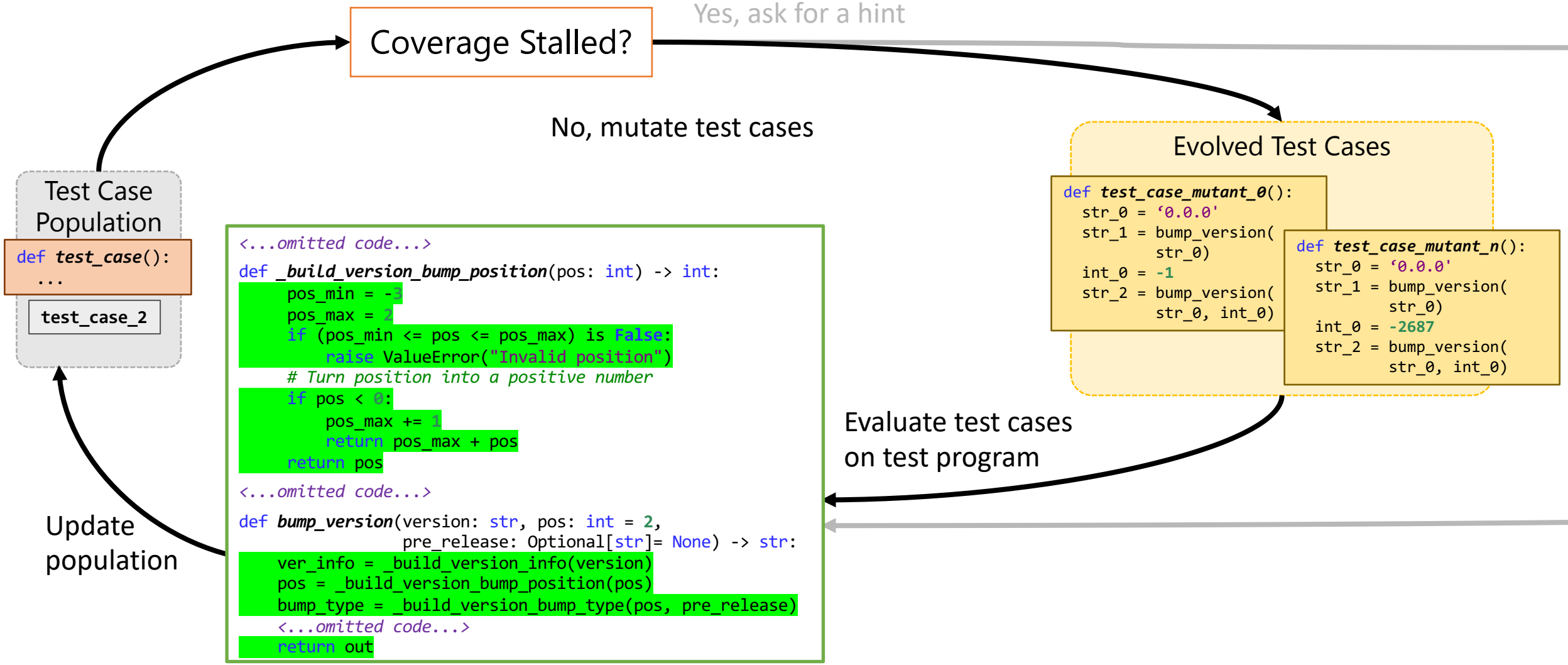
# Search No Longer Stalled



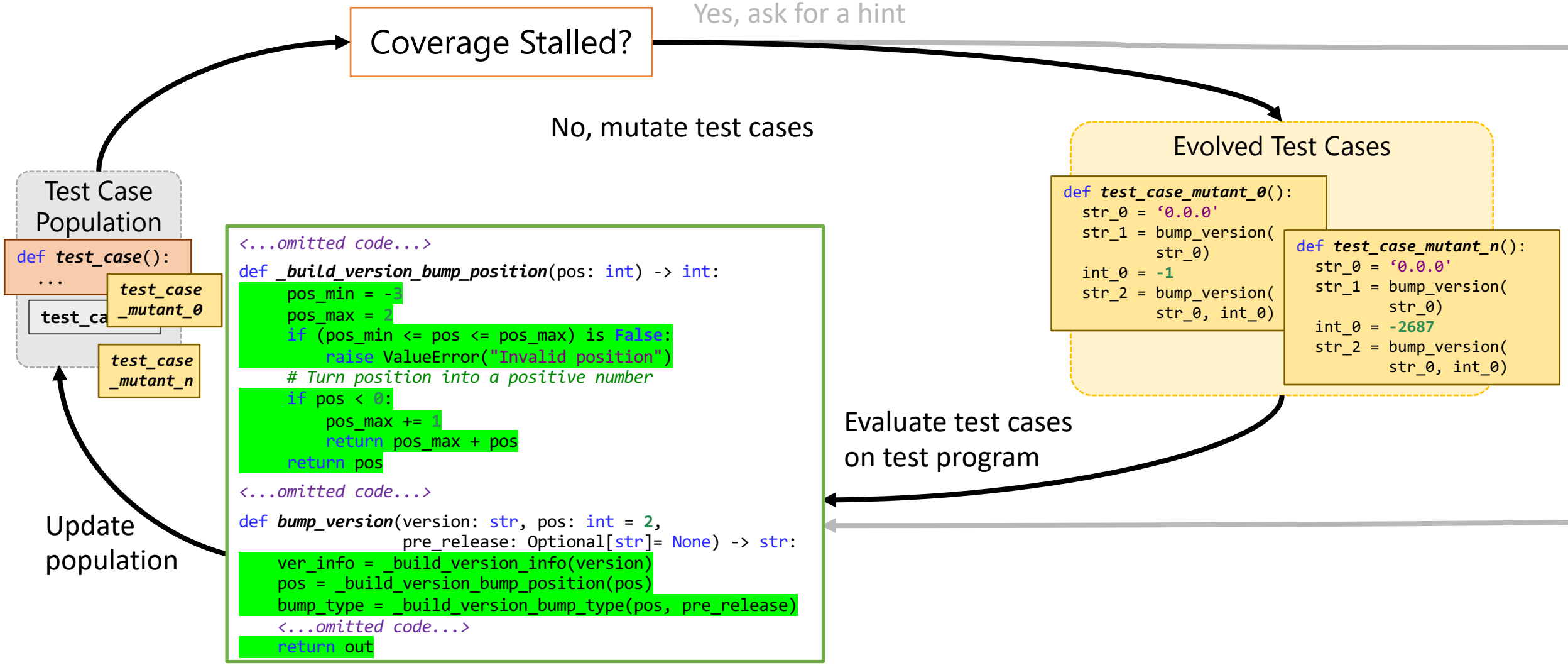
# Search No Longer Stalled



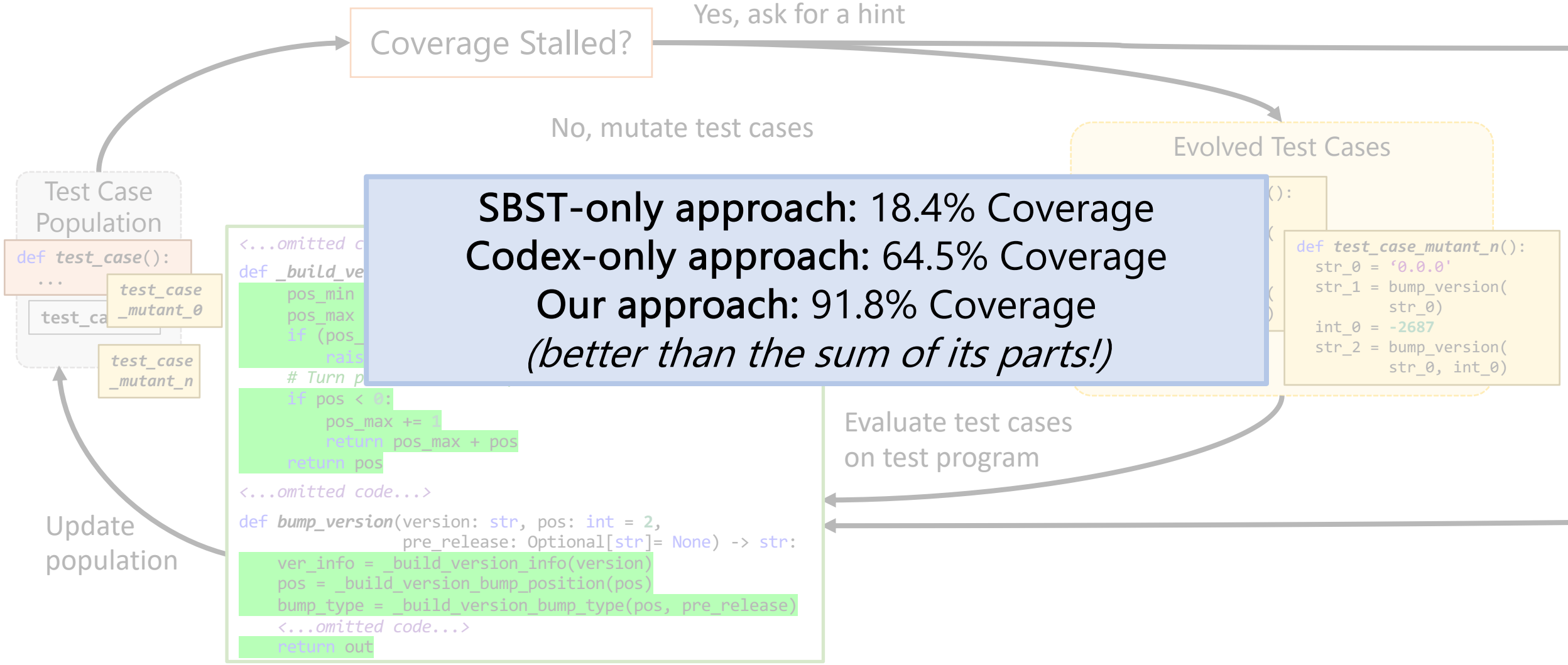
# Search No Longer Stalled



# Search No Longer Stalled



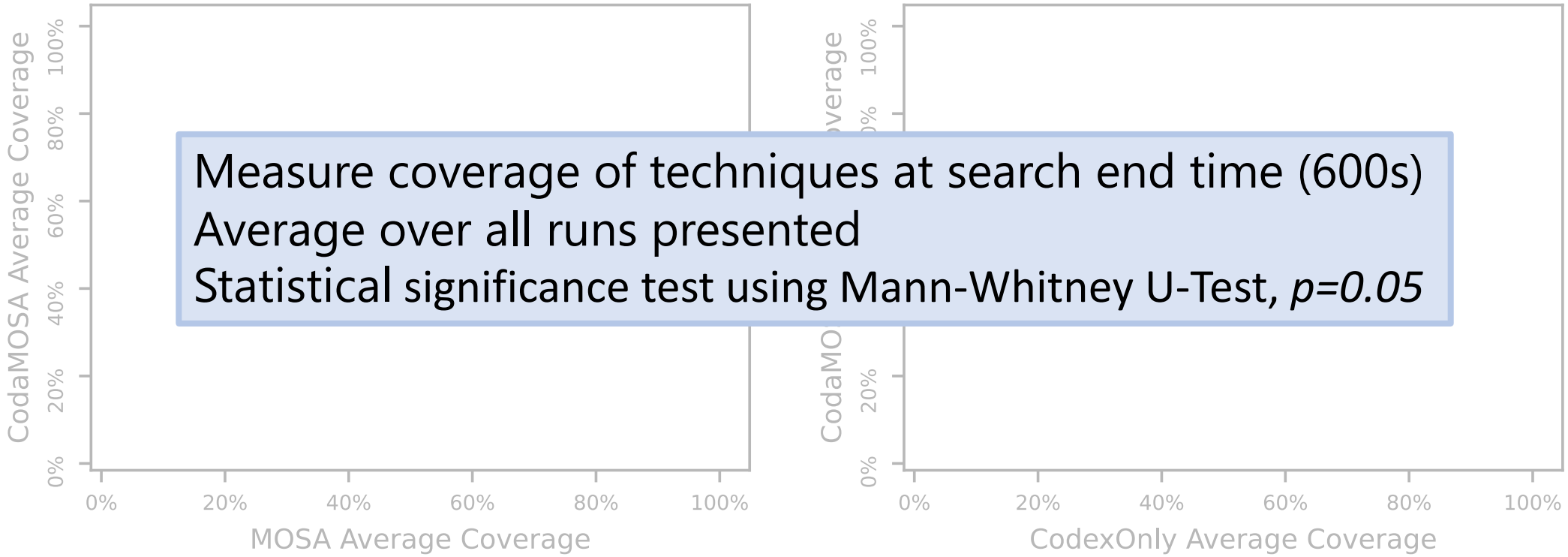
# Spoiler: Results on this Benchmark



# Evaluation Setup

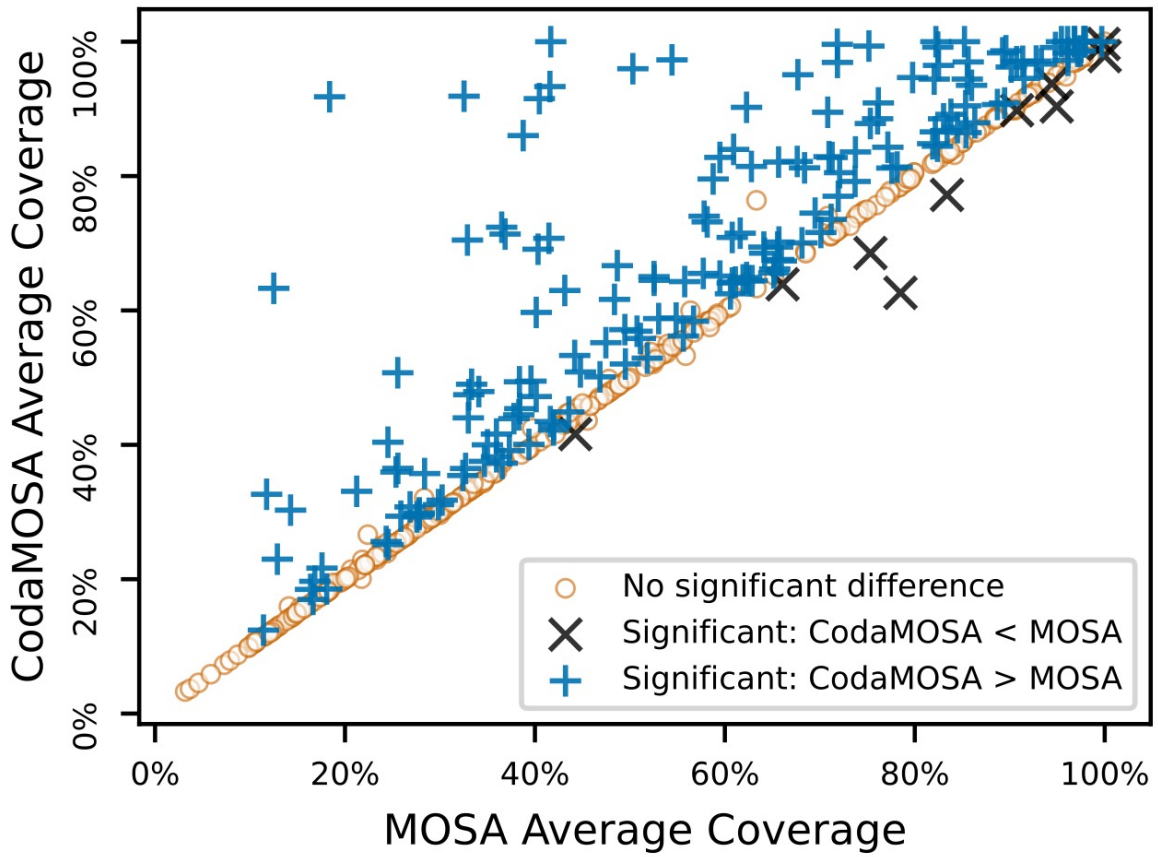
- 486 Modules from 27 Python Projects
- Run each technique 16 times, 600s each
- Compare to baselines:
  - MOSA (no Codex hints, Pynguin Implementation)
  - CodexOnly (no mutative search)

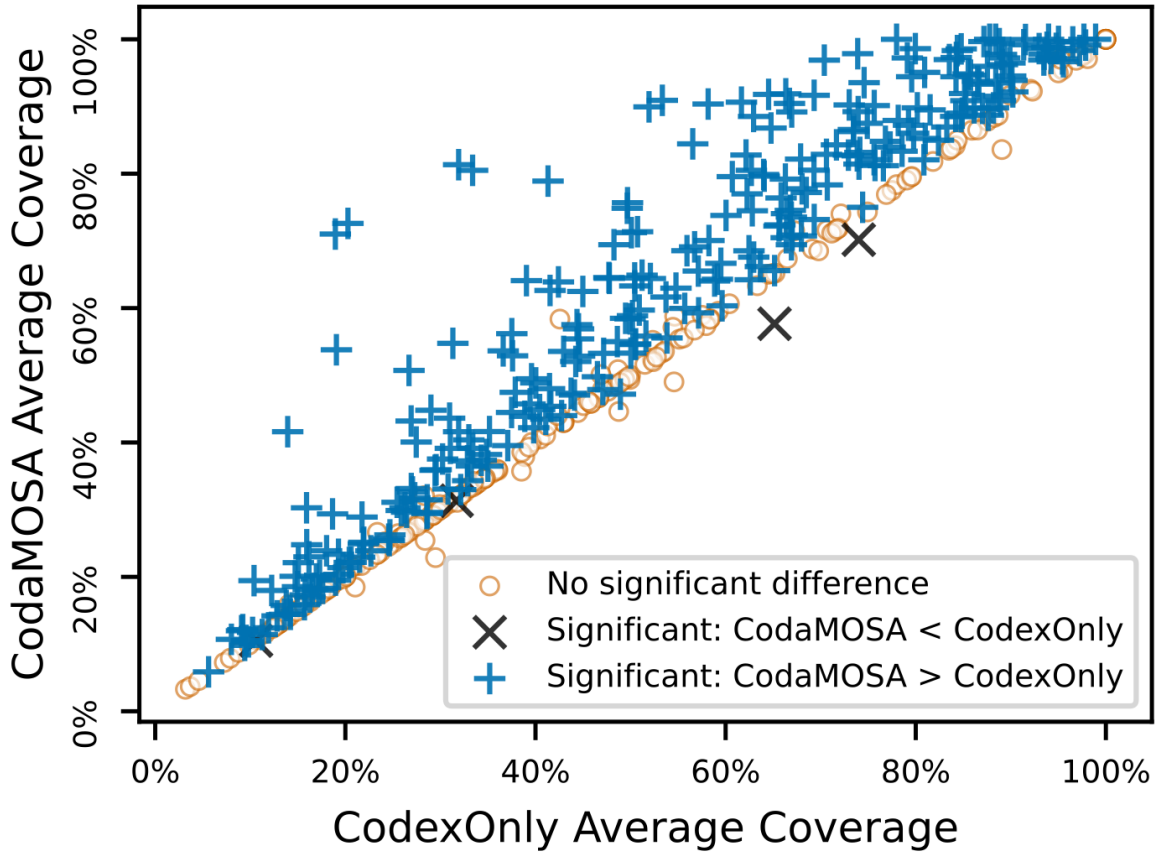
# Final Coverage Comparison



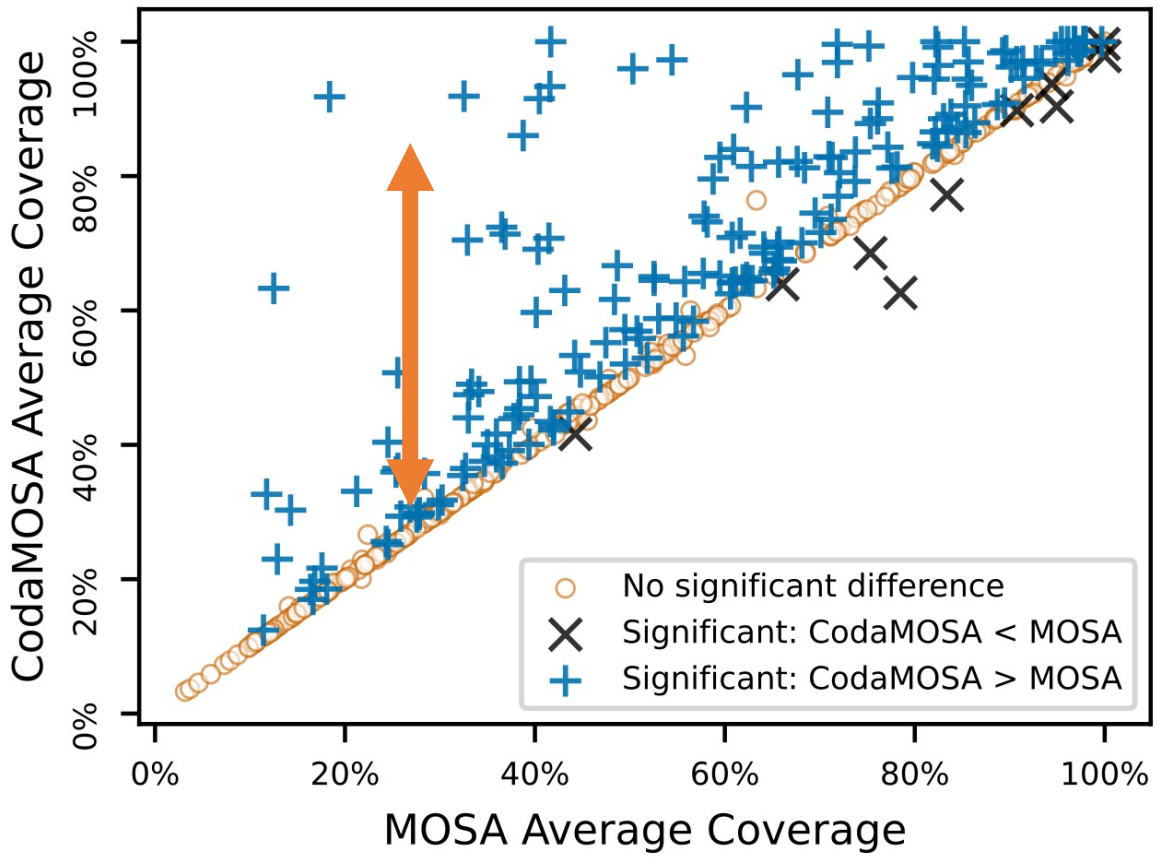

Measure coverage of techniques at search end time (600s)
Average over all runs presented
Statistical significance test using Mann-Whitney U-Test, *p=0.05*

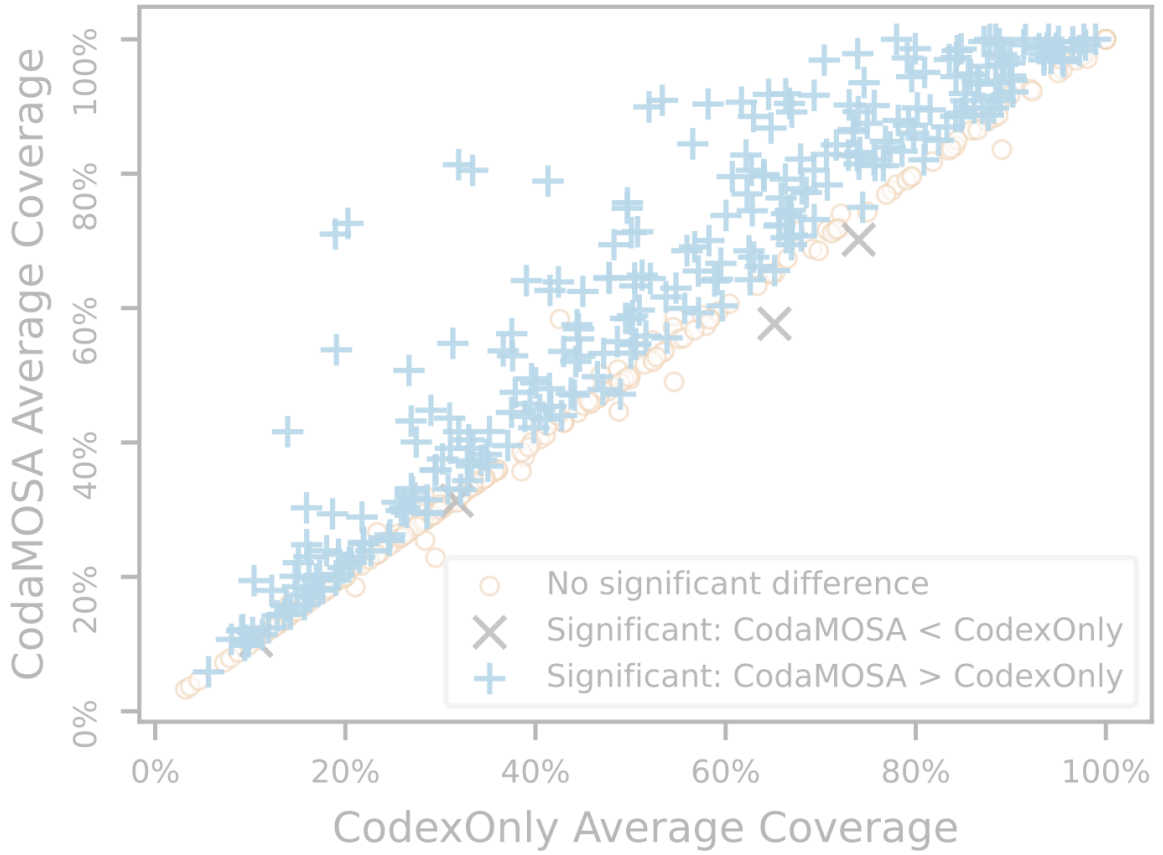
# CodaMOSA Outperforms Baselines



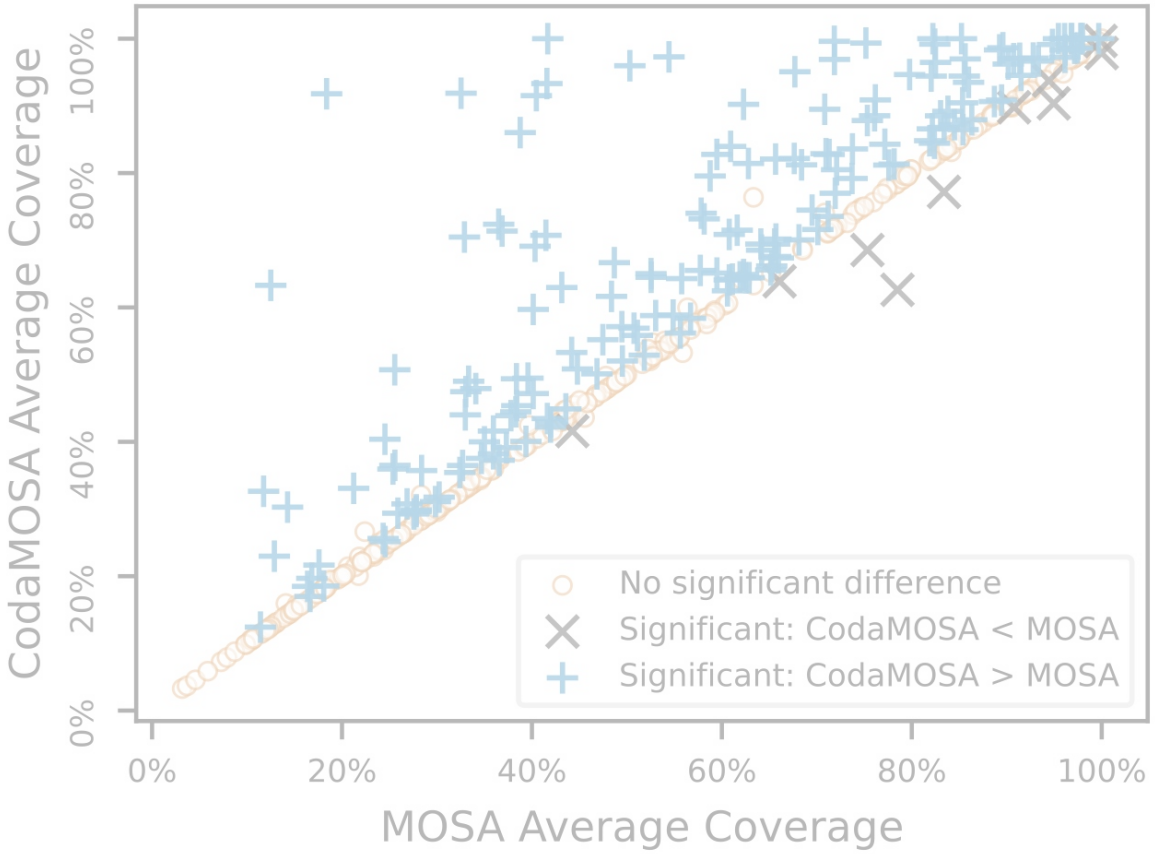



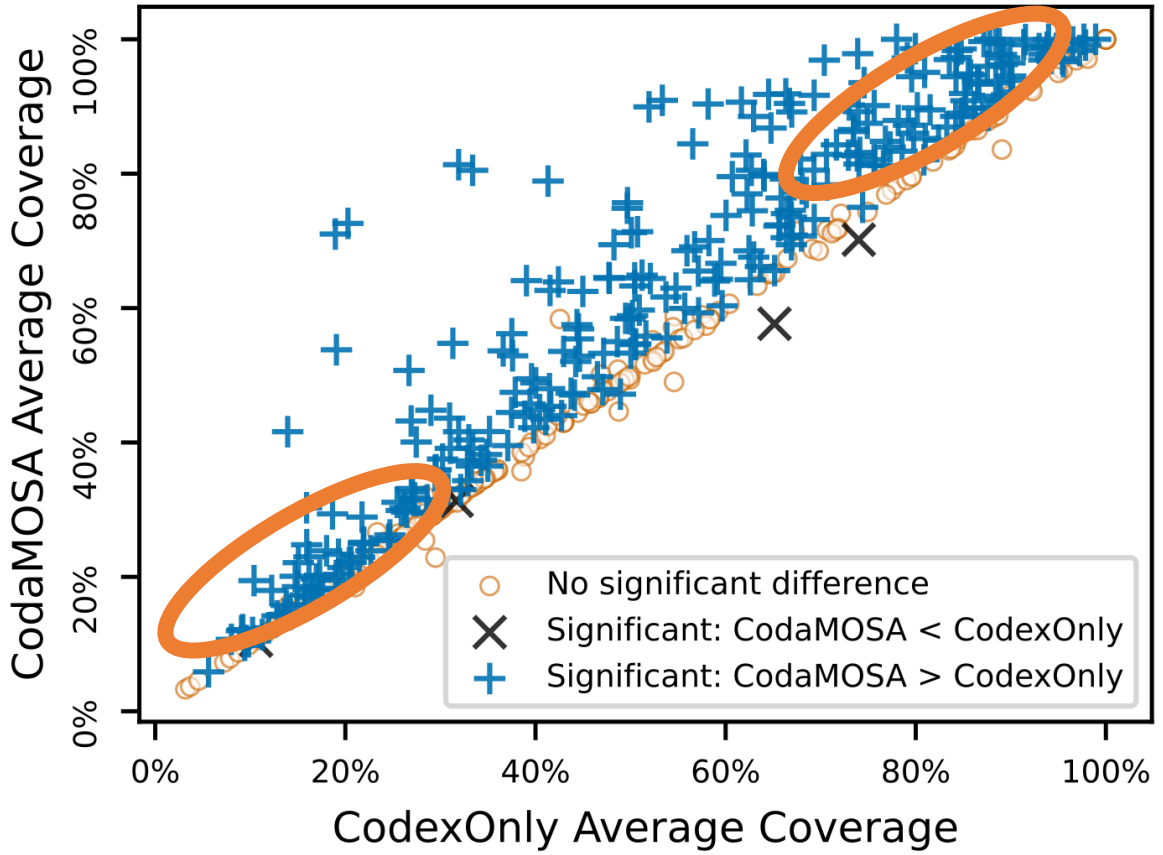
# CodaMOSA Outperforms Baselines



# CodaMOSA Outperforms Baselines



# Common Causes for Improvements

- Manually analyze 20 benchmarks w/ biggest coverage increases
  - 15/20 benchmarks: “special strings”

```
bump_version('0.0.0')
```

  - 7/20 benchmarks: construct data correctly w/o Type Hints

```
str_0 = 'devbox01'
host_0 = module_2.Host(str_0)
group_0 = module_0.Group(str_0)
var_0 = [host_0, host_0, host_0, group_0, group_0, group_0]
vars_module_0 = module_1.VarsModule()
var_1 = vars_module_0.get_vars(str_0, str_0, var_0)
```

  - 5/20 benchmarks: introduce new syntactical constructs

```
str_0 = 'c'
var_0 = module_0.join_each(str_0, str_0)
var_1 = list(var_0)
```

# Common Causes for Improvements

- Manually analyze 20 benchmarks w/ biggest coverage increases
  - 15/20 benchmarks: “special strings”

```
bump_version('0.0.0')
```

  - 7/20 benchmarks: c

```
str_0 = 'devbox01'
host_0 = module_2.host(str_0)
group_0 = module_0.Group(str_0)
var_0 = [host_0, host_0, host_0, group_0, group_0, group_
vars_module_0 = module_1.VarsModule()
var_1 = vars_module_0.get_vars(str_0, str_0, var_0)
```

  - 5/20 benchmarks: introduce new syntactical constructs

```
str_0 = 'c'
var_0 = module_0.join_each(str_0, str_0)
var_1 = list(var_0)
```

Should we use LLMs to get through magic bytes?

Probably overkill…

# Is Codex Just Copying Existing Tests?

*Example with high similarity (0.713)*

Codex generation:

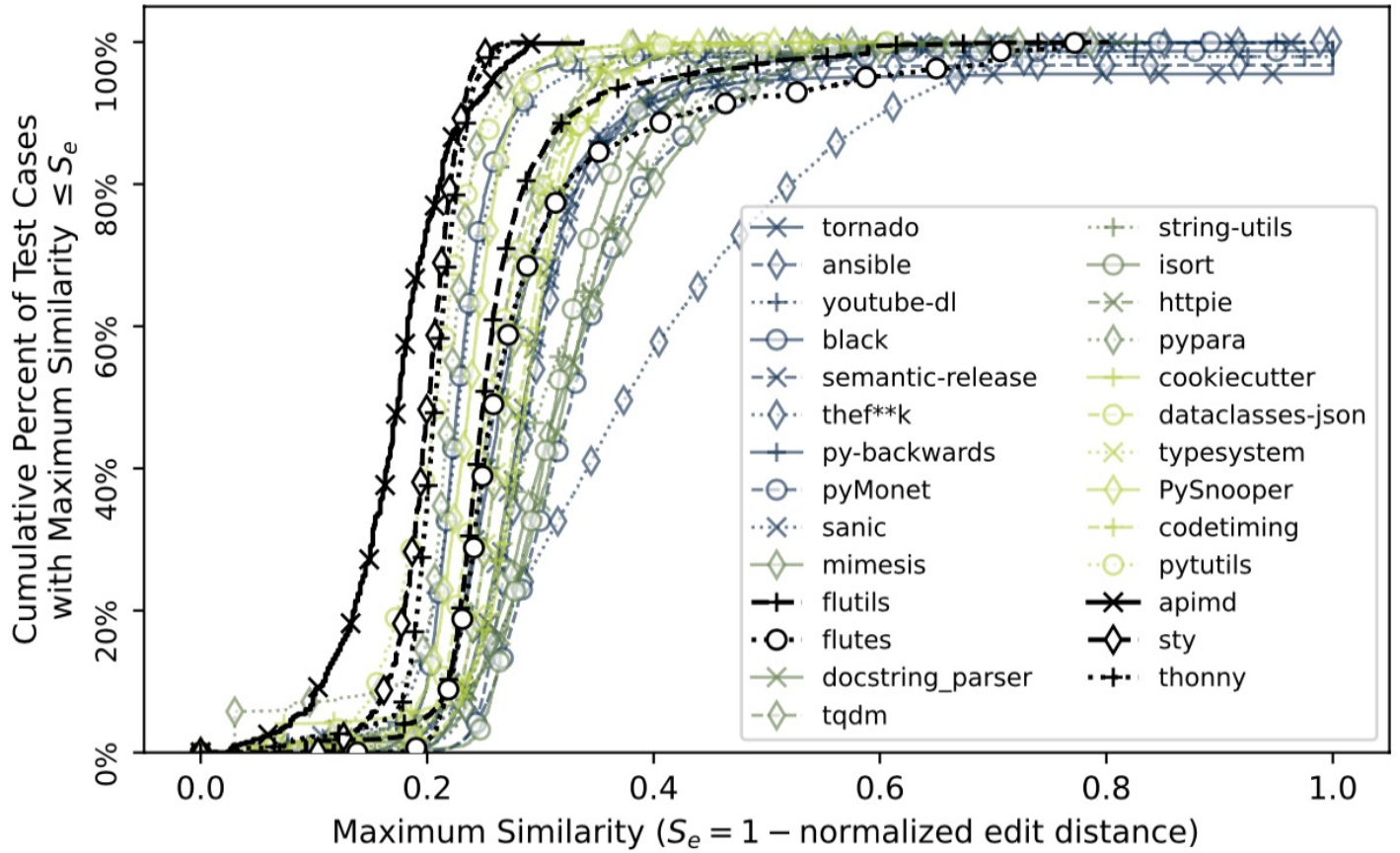
```
assert list(scanl(operator.add, [1,2,3,4], 0)) == [0,1,3,6,10]
assert list(scanl(lambda acc, x: x + acc, ['a', 'b', 'c', 'd']))
                                == ['a', 'ba', 'cba', 'dcba']}
```

Function in other part of code base (out of prompt):

```
check_iterator(flutes.scanl(operator.add, [1,2,3,4], 0), [0,1,3,6,10])}
check_iterator(flutes.scanl(lambda s, x: x + s, ['a', 'b', 'c', 'd']),
                                ['a','ba', 'cba', 'dcba'])}
```

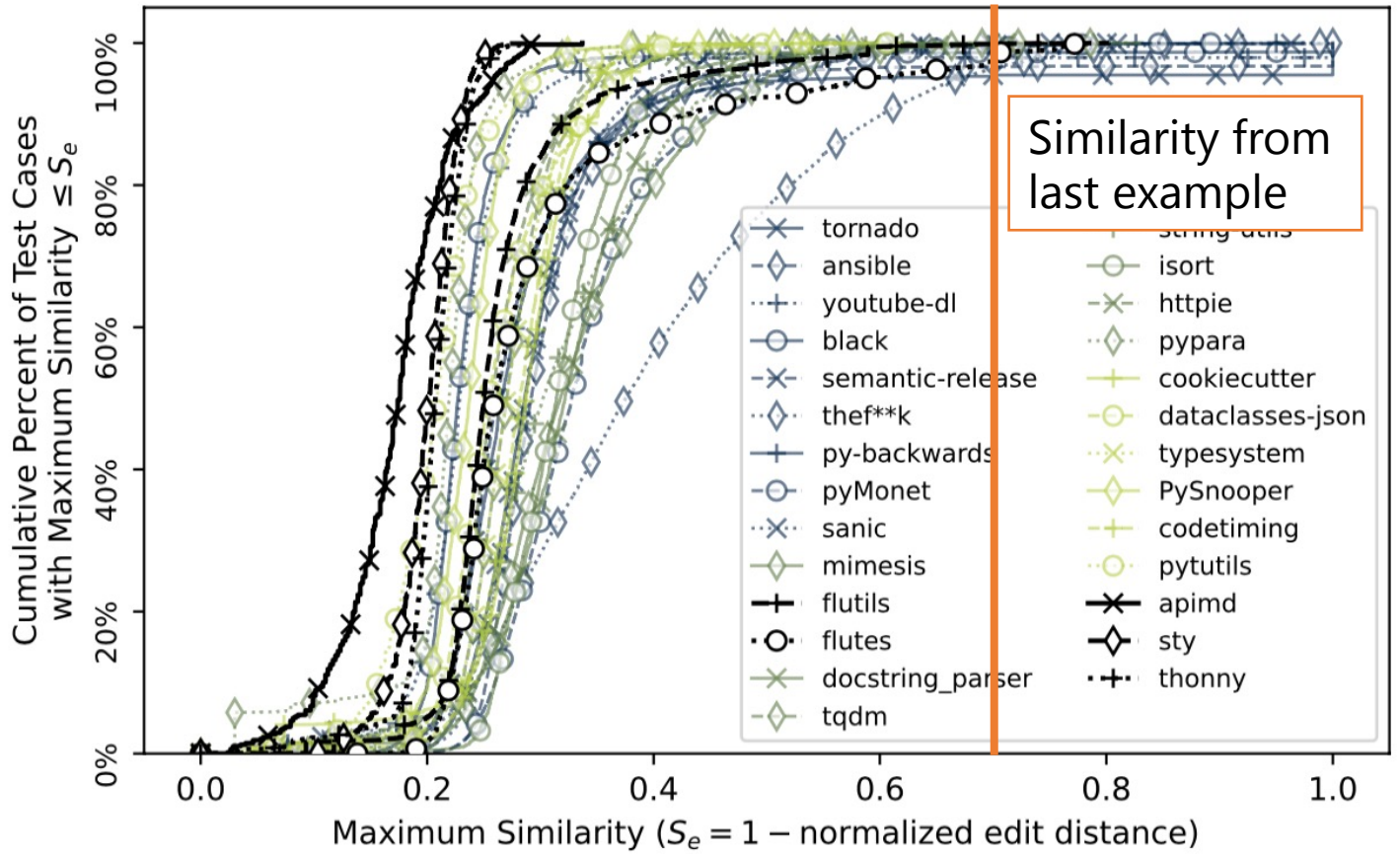
# Most Generated Tests Not Too Similar

*y*-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



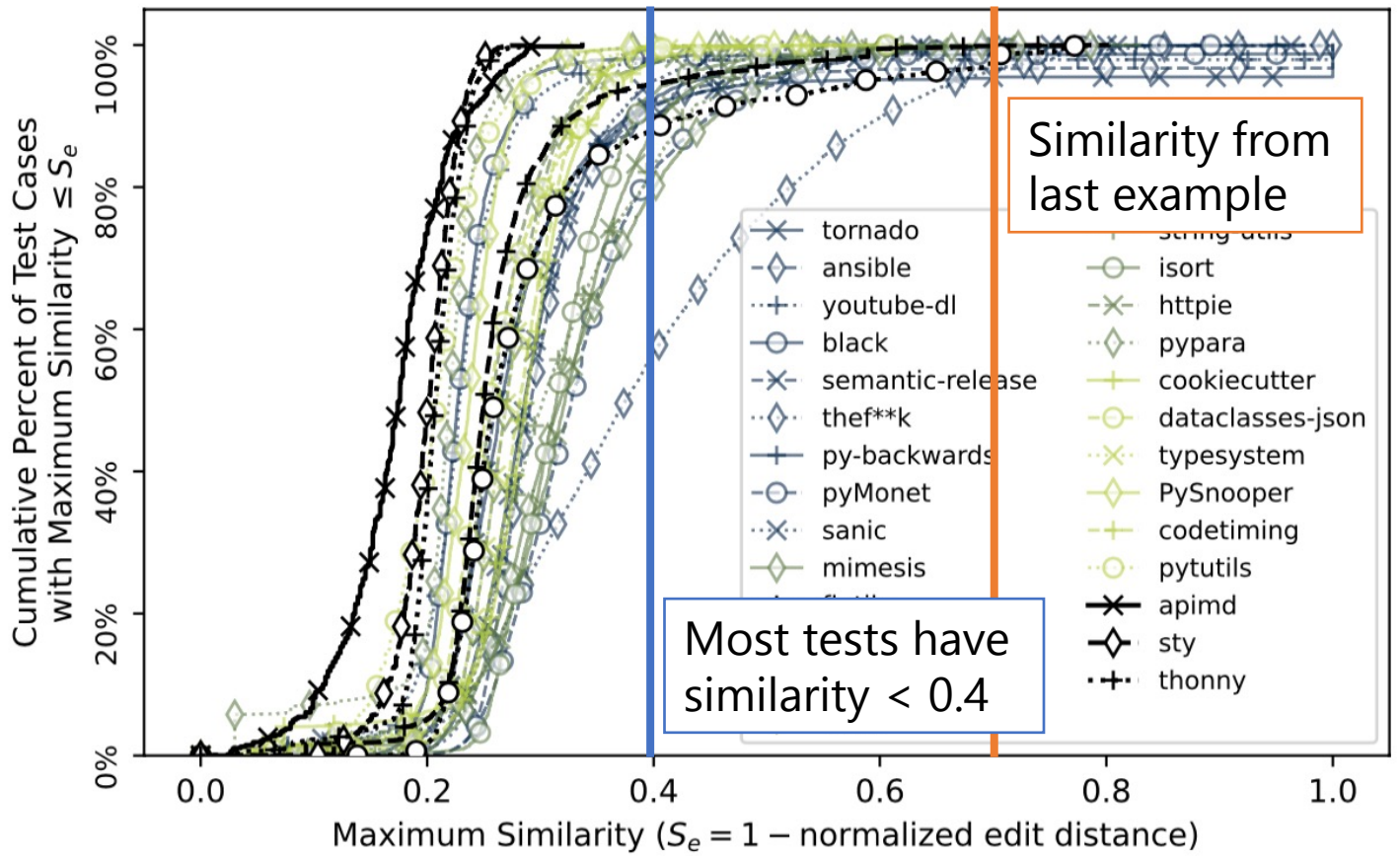
# Most Generated Tests Not Too Similar

*y*-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



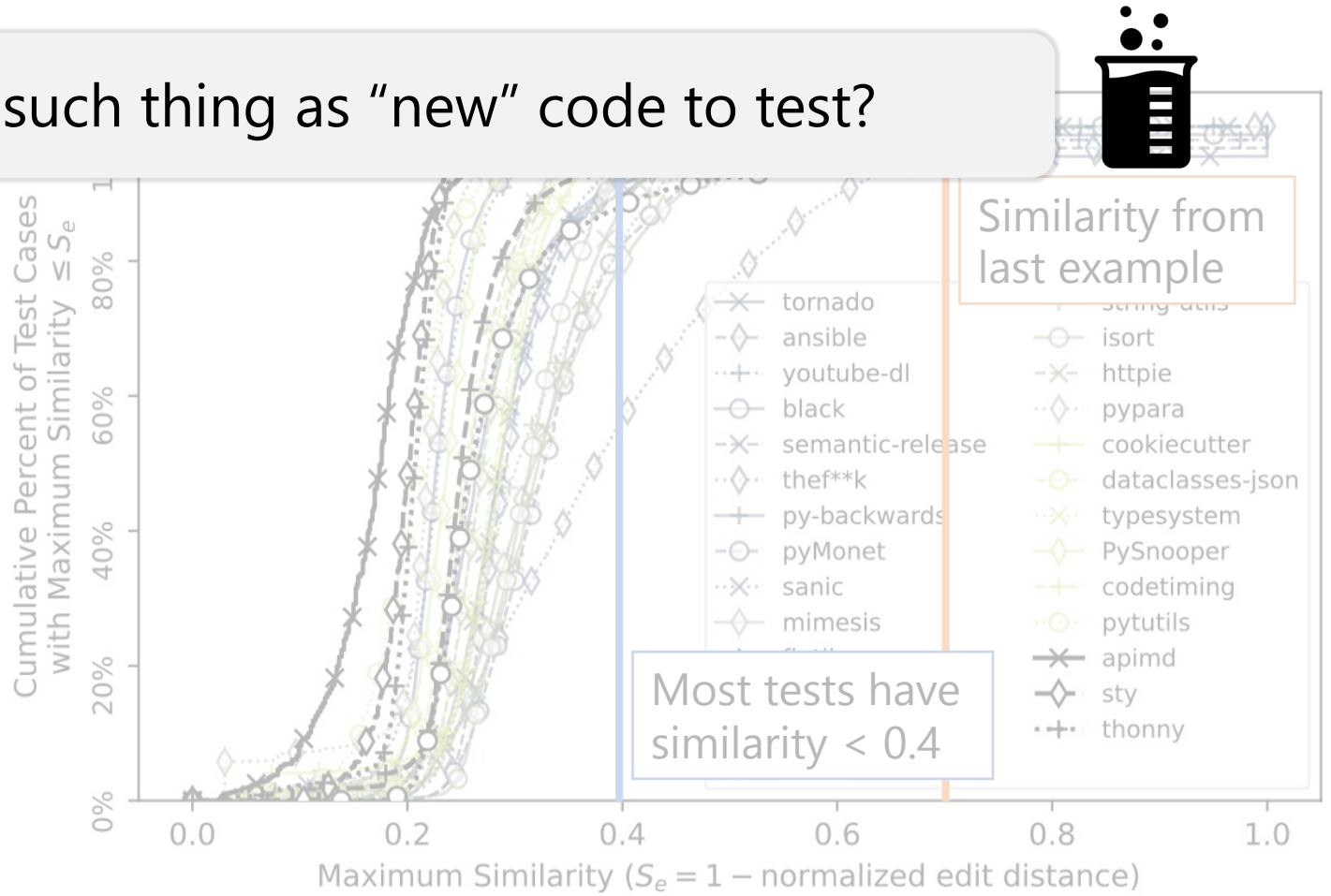
# Most Generated Tests Not Too Similar

*y*-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis



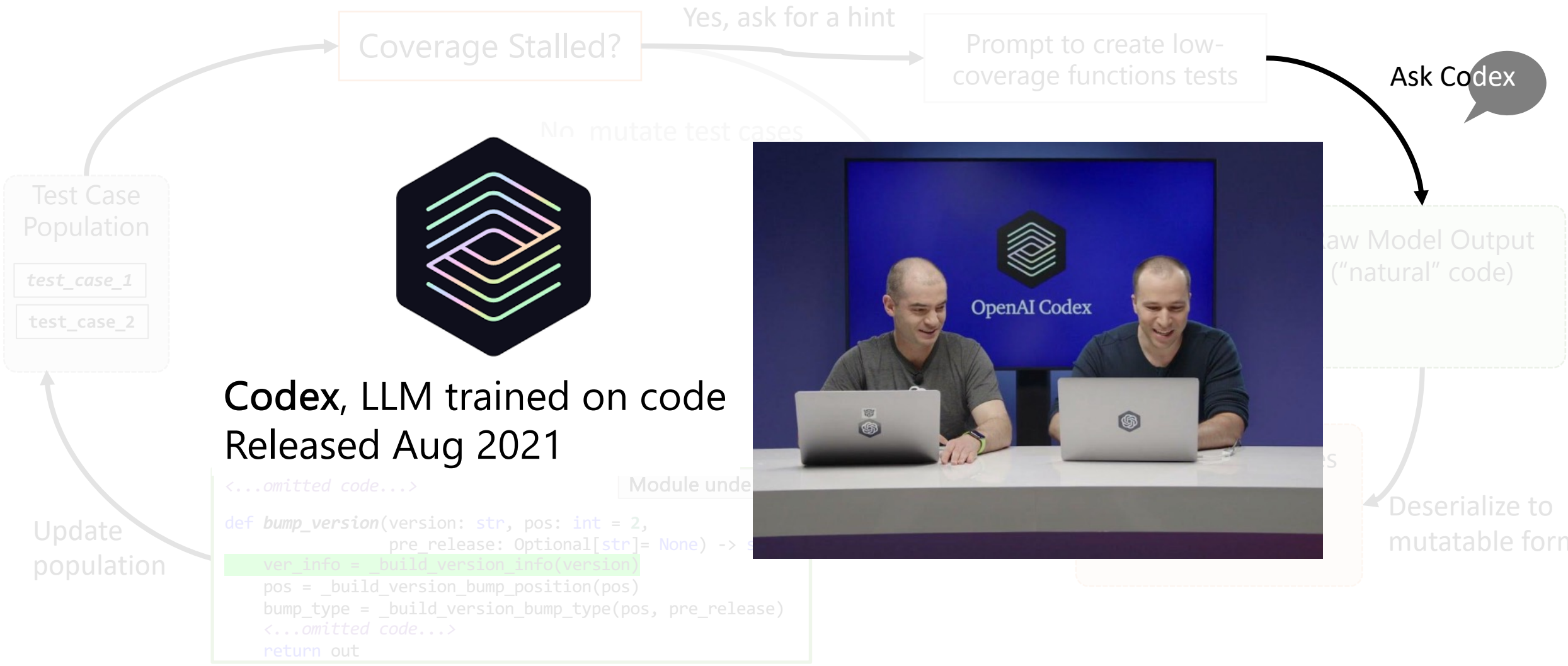
# Most Generated Tests Not Too Similar

Is there such thing as “new” code to test?



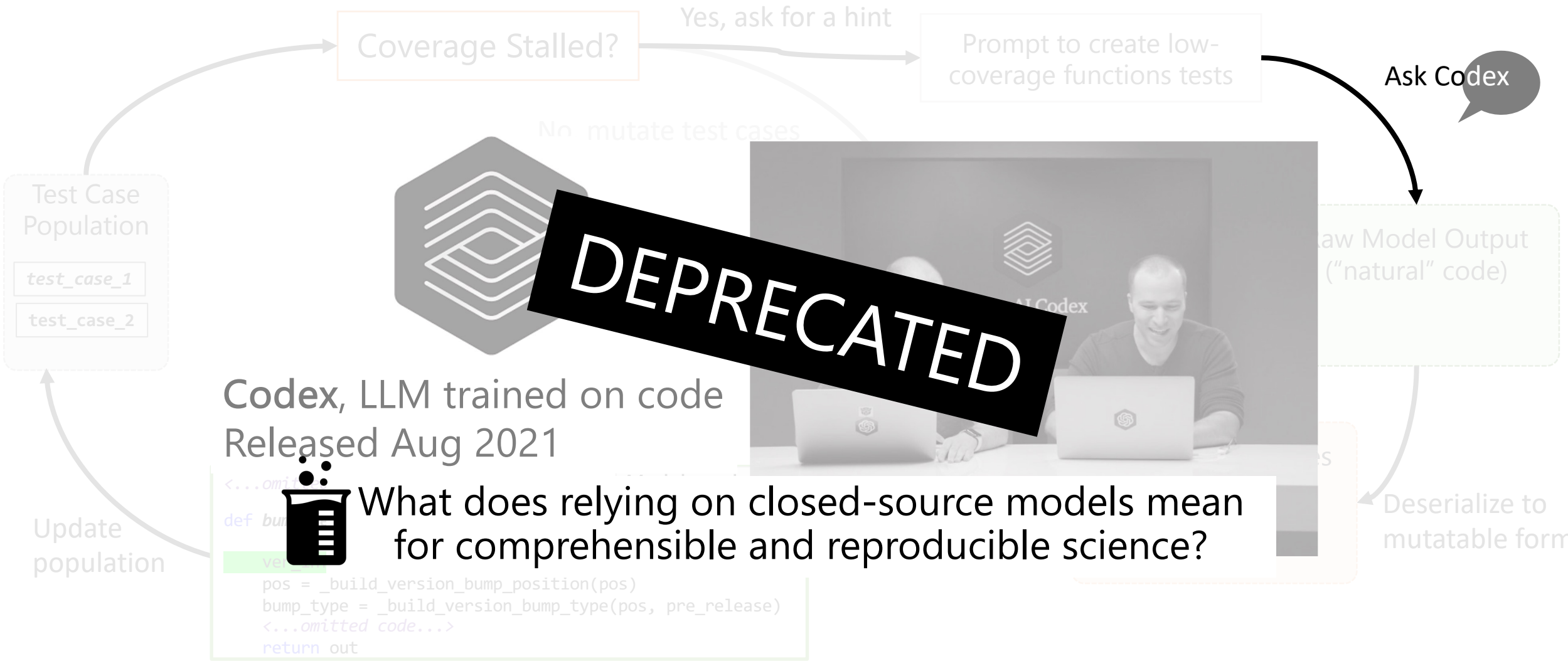

*y*-axis: cumulative # of Codex-generated tests with max. similarity <= *x*-axis

# Reliance on Codex




**Codex**, LLM trained on code
Released Aug 2021

# Reliance on Codex



## Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

## ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

**Analyse:** Lemieux, et al. CodaMOSA (ICSE '23). https://doi.org/10.1109/ICSE48619.2023.00085

# Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

# ML is not Magic

Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

# Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?

# It Takes a Village

*Thanks to my collaborators whose work is mentioned:*

Rohan Padhye, Rohan Bavishi, Sameer Reddy, Koushik Sen, Yves Le Traon, Mike Papadakis, Ion Stoica, Roy Fox, Siddhartha Sen, Jeevana Priya Inala, Shuvendu Lahiri

*Thanks to SPL colleagues for honest feedback on this talk:*

Ivan Beschastnikh, William Bowman, Finn Hackett, Rui Ge, Zack Grannan, Paulette Koronkevitch, Yanze Li, Mayant Mukul, Jifeng Wu

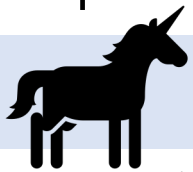
# The Power of Fuzzing and Large Language Models is harnessing (just enough) randomness.

@cestlemieux clemieux@cs.ubc.ca carolemieux.com

## Comprehensible and Replicable Science

Are we creating knowledge (or just the most performant tool)?
Understanding the strengths + weaknesses of existing techniques is vital for innovation

## ML is not Magic



Coverage-guided fuzzing is powerful and optimized for test-input generation
Random and exhaustive search remain powerful tools!

## Synergies with Large Language Models

…but large language models allow us to generate code like never before
Are we creating knowledge (or just the most performant tool)?