CPSC 593L: Topics in Programming Languages

Fuzz Testing

September 14th, 2022

Instructor: Caroline Lemieux

Term: 2022W1

Class website: carolemieux.com/teaching/CPSC539L_2022w1.html

So far…

We've talked about "random" or "blackbox" fuzz testing

- Recall: "blackbox" refers to the fact that we only observe the external reactions of the program under test (black-box == opaque-box)

We have read the paper originating the term "fuzz" testing (1990)

But… fuzz testing did not become a big research area in 1990. Why?

Recall: Bugs in OpenSSL

Heartbleed

Severity: 7.5 HIGH

In

Di

Fixed: 7 Apr 2014

“... can be used to reveal up to 64k of memory to a connected client or server ...”

Costs:

- >$500 million
- 30,000 X.509 certificates compromised
- 4.5 million patient records compromised
- CRA website shutdown, 900 SINs leaked
- ...

by honggfuzz

(modern coverage-guided fuzzer)

?

CVE-2016-6309

Severity: 9.8 CRITICAL

Introduced: 22 Sep 2016

Discovered: 23 Sep 2016

Fixed: 26 Sep 2016

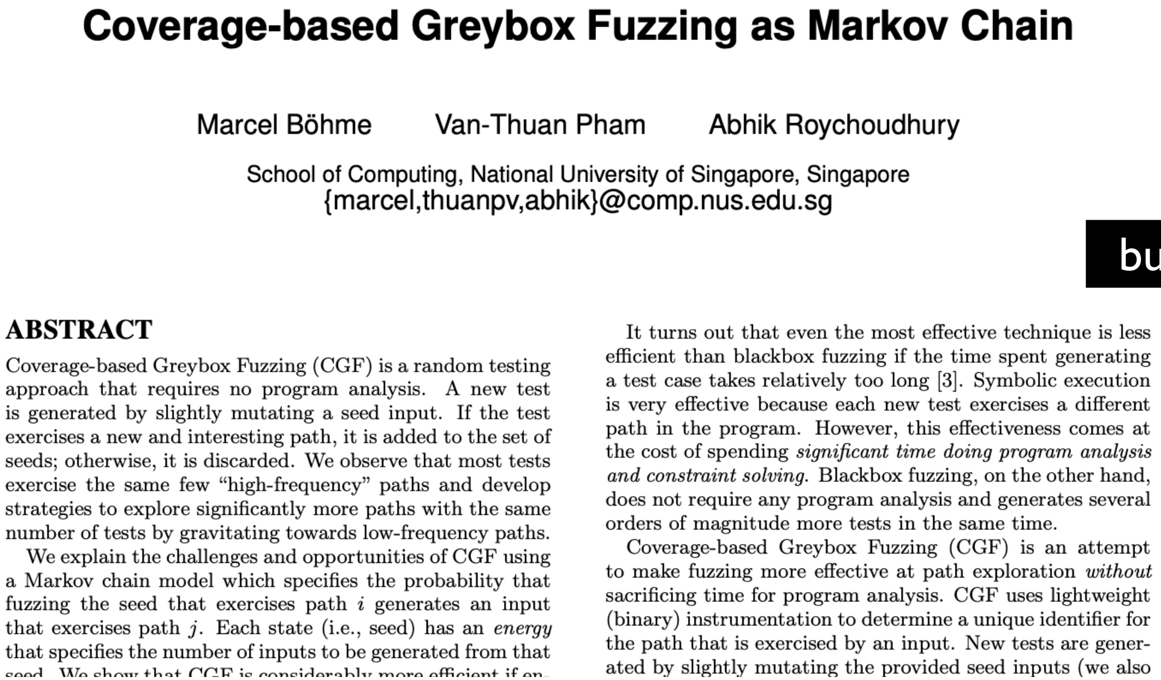
“... likely to result in a crash, however it could potentially lead to execution of arbitrary code ...”

Costs:

- minimal

What really popularized fuzz testing?

For researchers:



Coverage-based Greybox Fuzzing as Markov Chain

Marcel Böhme Van-Thuan Pham Abhik Roychoudhury

School of Computing, National University of Singapore, Singapore
{marcel,thuanpv,abhik}@comp.nus.edu.sg

ABSTRACT

Coverage-based Greybox Fuzzing (CGF) is a random testing approach that requires no program analysis. A new test is generated by slightly mutating a seed input. If the test exercises a new and interesting path, it is added to the set of seeds; otherwise, it is discarded. We observe that most tests exercise the same few "high-frequency" paths and develop strategies to explore significantly more paths with the same number of tests by gravitating towards low-frequency paths.

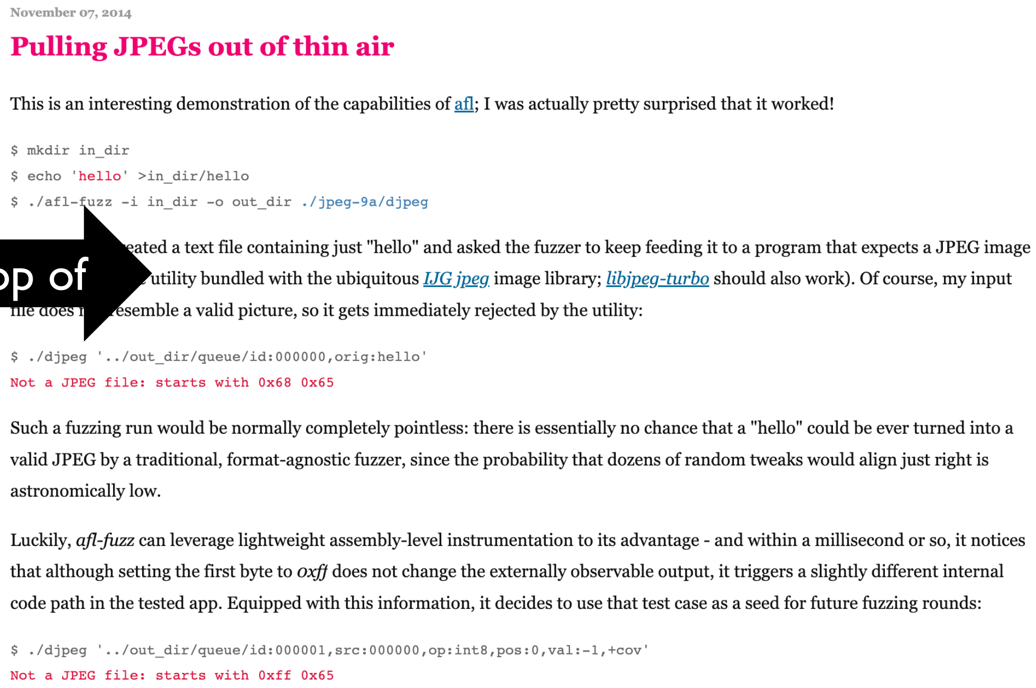
We explain the challenges and opportunities of CGF using a Markov chain model which specifies the probability that fuzzing the seed that exercises path i generates an input that exercises path j. Each state (i.e., seed) has an *energy* that specifies the number of inputs to be generated from that seed. We show that CGF is considerably more efficient if en-

It turns out that even the most effective technique is less efficient than blackbox fuzzing if the time spent generating a test case takes relatively too long [3]. Symbolic execution is very effective because each new test exercises a different path in the program. However, this effectiveness comes at the cost of spending *significant time doing program analysis and constraint solving.* Blackbox fuzzing, on the other hand, does not require any program analysis and generates several orders of magnitude more tests in the same time.

Coverage-based Greybox Fuzzing (CGF) is an attempt to make fuzzing more effective at path exploration *without* sacrificing time for program analysis. CGF uses lightweight (binary) instrumentation to determine a unique identifier for the path that is exercised by an input. New tests are generated by slightly mutating the provided seed inputs (we also

builds on top of

For practitioners:



November 07, 2014

Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

```
$ mkdir in_dir
$ echo 'hello' >in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir ./jpeg-9a/djpeg
```

eated a text file containing just "hello" and asked the fuzzer to keep feeding it to a program that expects a JPEG image utility bundled with the ubiquitous IJG jpeg image library; libjpeg-turbo should also work). Of course, my input file does esemble a valid picture, so it gets immediately rejected by the utility:

```
$ ./djpeg '../out_dir/queue/id:000000,orig:hello'
Not a JPEG file: starts with 0x68 0x65
```

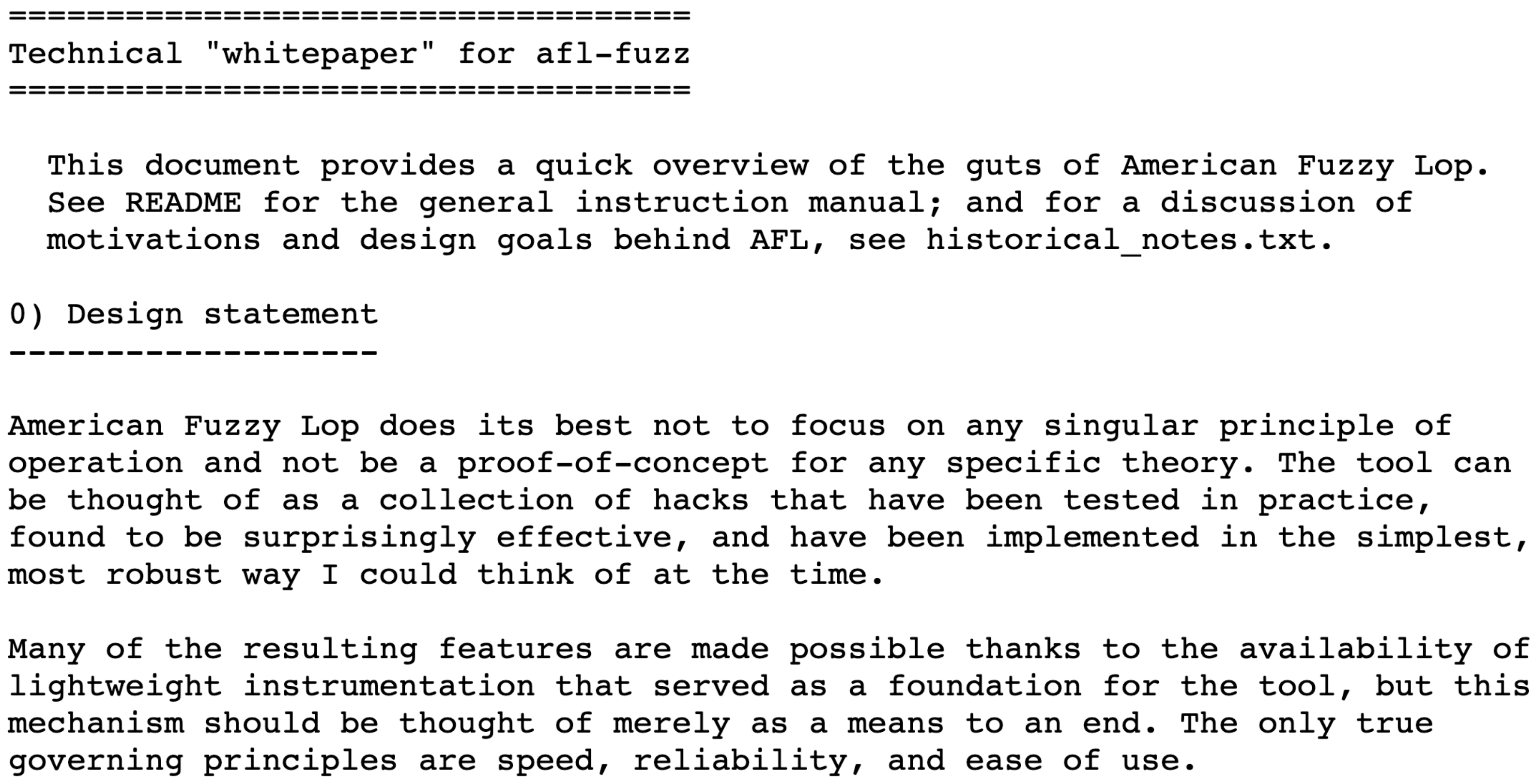
Such a fuzzing run would be normally completely pointless: there is essentially no chance that a "hello" could be ever turned into a valid JPEG by a traditional, format-agnostic fuzzer, since the probability that dozens of random tweaks would align just right is astronomically low.

Luckily, *afl-fuzz* can leverage lightweight assembly-level instrumentation to its advantage - and within a millisecond or so, it notices that although setting the first byte to *0xff* does not change the externally observable output, it triggers a slightly different internal code path in the tested app. Equipped with this information, it decides to use that test case as a seed for future fuzzing rounds:

```
$ ./djpeg '../out_dir/queue/id:000001,src:000000,op:int8,pos:0,val:-1,+cov'
Not a JPEG file: starts with 0xff 0x65
```

Is there a seminal paper of coverage-guided fuzz testing?

No.



```
===================================
Technical "whitepaper" for afl-fuzz
===================================

  This document provides a quick overview of the guts of American Fuzzy Lop.
  See README for the general instruction manual; and for a discussion of
  motivations and design goals behind AFL, see historical_notes.txt.

0) Design statement
-------------------

American Fuzzy Lop does its best not to focus on any singular principle of
operation and not be a proof-of-concept for any specific theory. The tool can
be thought of as a collection of hacks that have been tested in practice,
found to be surprisingly effective, and have been implemented in the simplest,
most robust way I could think of at the time.

Many of the resulting features are made possible thanks to the availability of
lightweight instrumentation that served as a foundation for the tool, but this
mechanism should be thought of merely as a means to an end. The only true
governing principles are speed, reliability, and ease of use.
```

```
===================================
Technical "whitepaper" for afl-fuzz
===================================

  This document provides a quick overview of the guts of American Fuzzy Lop.
  See README for the general instruction manual; and for a discussion of
  motivations and design goals behind AFL, see historical_notes.txt.

0) Design statement
-------------------

American Fuzzy Lop does its best not to focus on any singular principle of
operation and not be a proof-of-concept for any specific theory. The tool can
be thought of as a collection of hacks that have been tested in practice,
found to be surprisingly effective, and have been implemented in the simplest,
most robust way I could think of at the time.

Many of the resulting features are made possible thanks to the availability of
lightweight instrumentation that served as a foundation for the tool, but this
mechanism should be thought of merely as a means to an end. The only true
governing principles are speed, reliability, and ease of use.
```

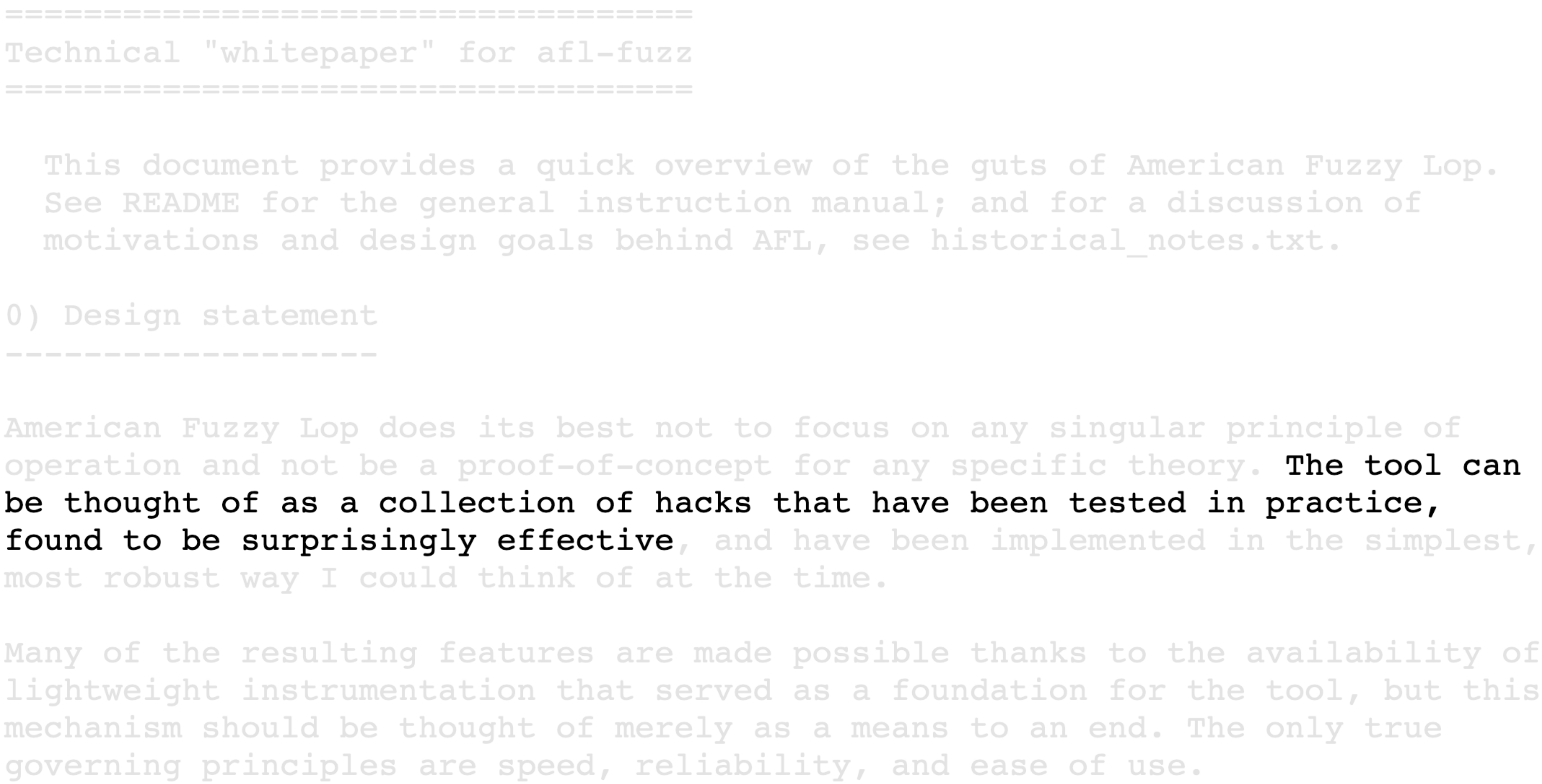
===================================
Technical "whitepaper" for afl-fuzz
===================================

This document provides a quick overview of the guts of American Fuzzy Lop. See README for the general instruction manual; and for a discussion of motivations and design goals behind AFL, see historical_notes.txt.

0) Design statement

American Fuzzy Lop does its best not to focus on any singular principle of operation and not be a proof-of-concept for any specific theory. The tool can be thought of as a collection of hacks that have been tested in practice, found to be surprisingly effective, and have been implemented in the simplest, most robust way I could think of at the time.

Many of the resulting features are made possible thanks to the availability of lightweight instrumentation that served as a foundation for the tool, but this mechanism should be thought of merely as a means to an end. The only true governing principles are speed, reliability, and ease of use.



Technical "whitepaper" for afl-fuzz

This document provides a quick overview of the guts of American Fuzzy Lop. See README for the general instruction manual; and for a discussion of motivations and design goals behind AFL, see historical_notes.txt.

0) Design statement

American Fuzzy Lop does its best not to focus on any singular principle of operation and not be a proof-of-concept for any specific theory. **The tool can be thought of as a collection of hacks that have been tested in practice, found to be surprisingly effective**, and have been implemented in the simplest, most robust way I could think of at the time.

Many of the resulting features are made possible thanks to the availability of lightweight instrumentation that served as a foundation for the tool, but this mechanism should be thought of merely as a means to an end. The only true governing principles are speed, reliability, and ease of use.

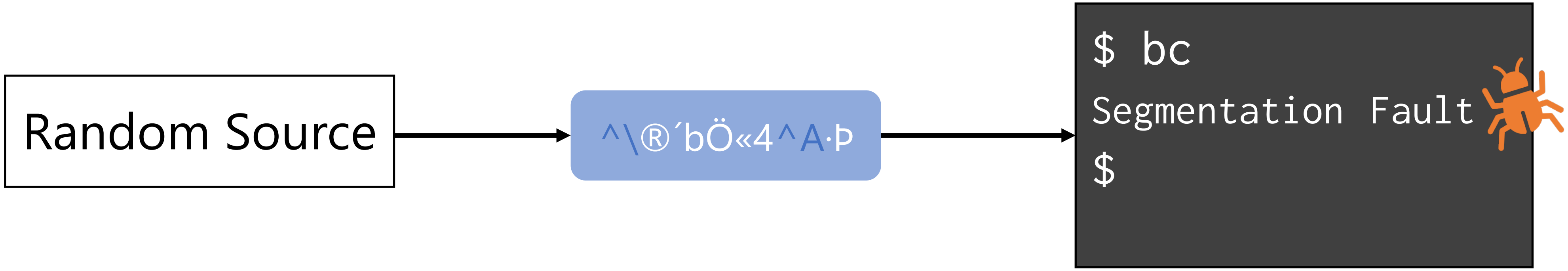
Schedule for Today

- Improving upon pure random fuzzing
- Coverage-guided fuzzing
  - a.k.a. greybox fuzzing, a.k.a. coverage-based greybox fuzzing
- Relation to Evolutionary Algorithms

Schedule for Today

- Improving upon pure random fuzzing
- Coverage-guided fuzzing
  - a.k.a. greybox fuzzing, a.k.a. coverage-based greybox fuzzing
- Relation to Evolutionary Algorithms

What if inputs are too random?

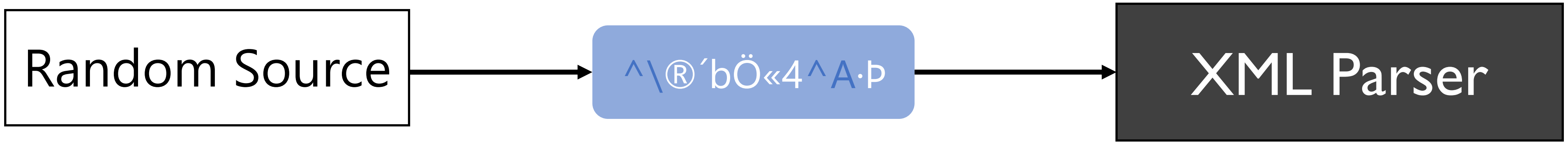


Random Source

^\®´bÖ«4^A·Þ

```
$ bc
Segmentation Fault
$
```

What if inputs are too random?

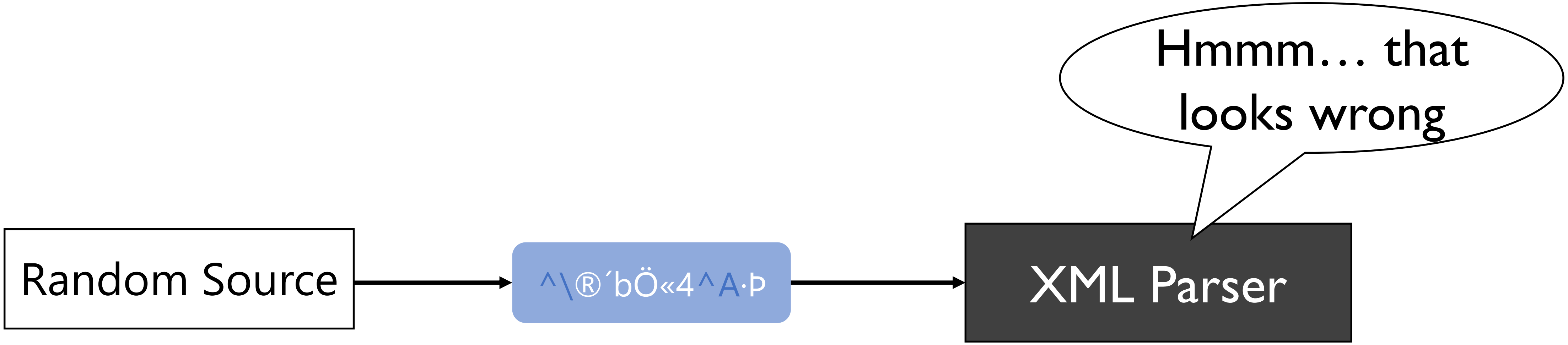


Random Source

^\®´bÖ«4^A·þ

XML Parser

What if inputs are too random?



Random Source

^\®´bÖ«4^A·Þ

XML Parser

Hmmm… that looks wrong

How to have less random inputs?

Write a specification, generate inputs based on that specification

- Generator-based fuzzing
- Property-based testing
- Grammar-based fuzzing
- …

Start from existing inputs and alter them slightly

- mutational fuzzing

How to have less random inputs?

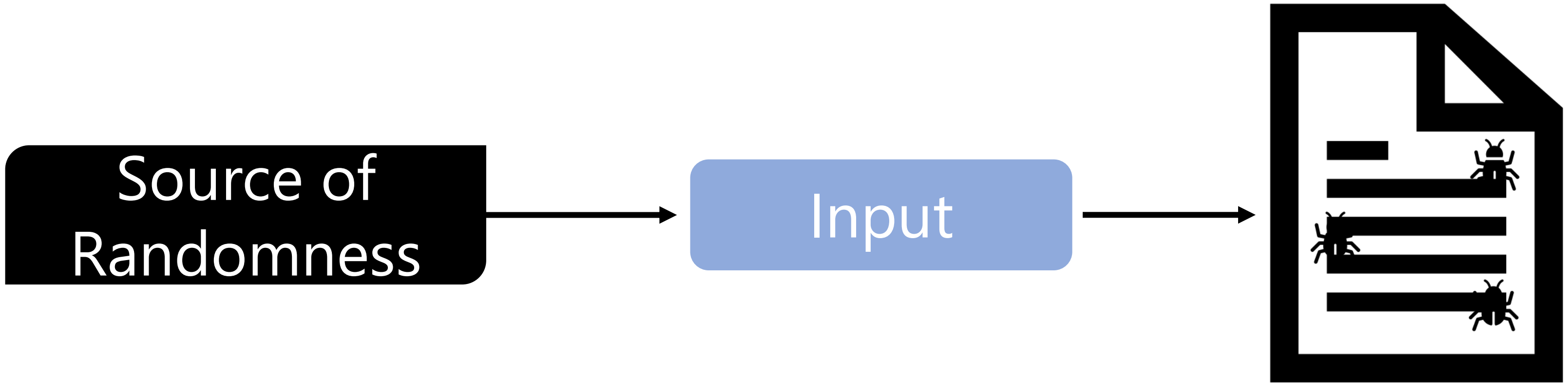
Write a specification, generate inputs based on that specification

- Generator-based fuzzing
- Property-based testing
- Grammar-based fuzzing
- …

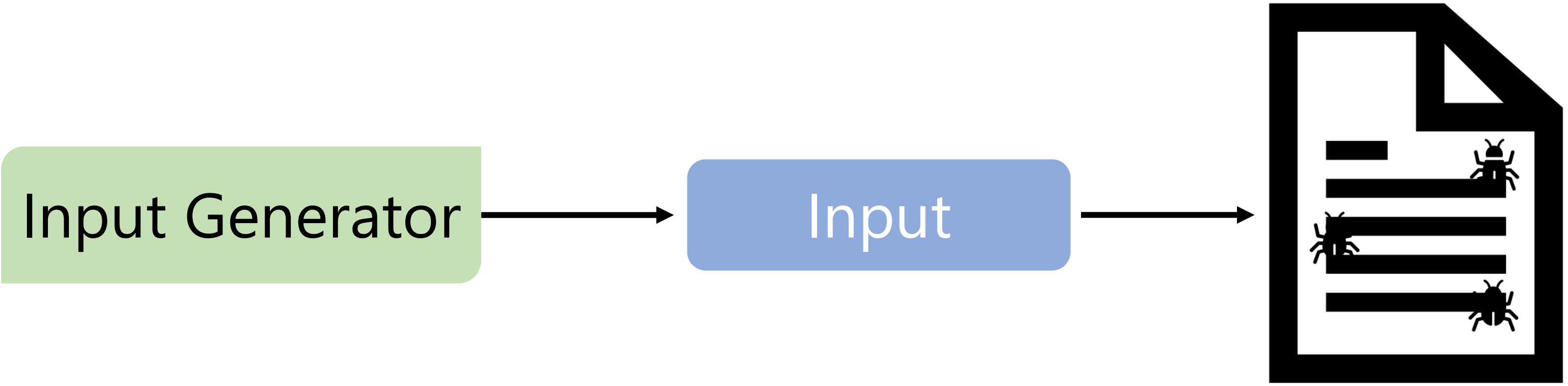
Start from existing inputs and alter them slightly

- mutational fuzzing

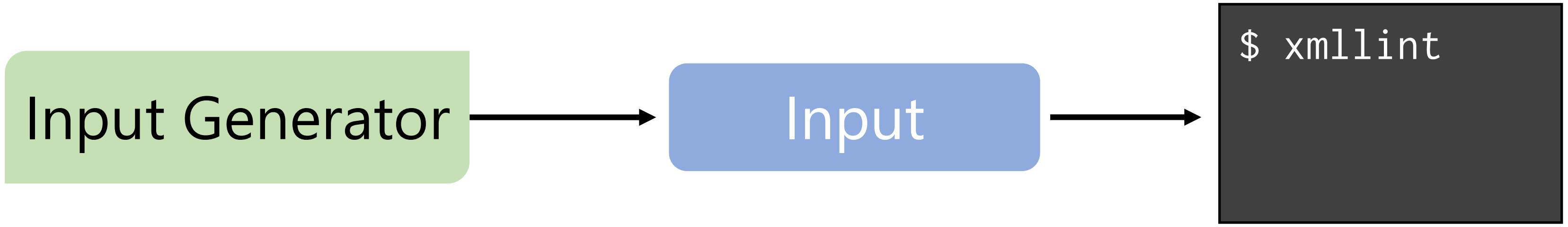
Random Fuzzing



Generator-Based Fuzzing



Generator-Based Fuzzing



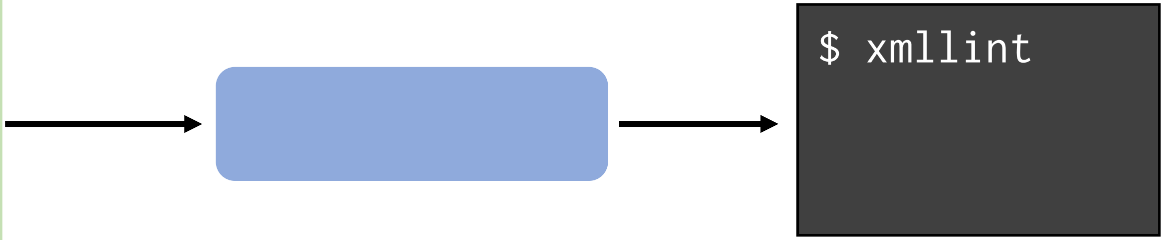
Input Generator

Input

```
$ xmllint
```

Generator-Based Fuzzing

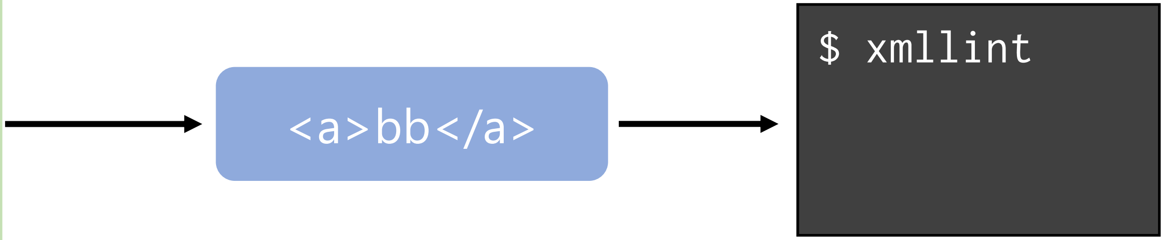
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



```
$ xmllint
```

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
```

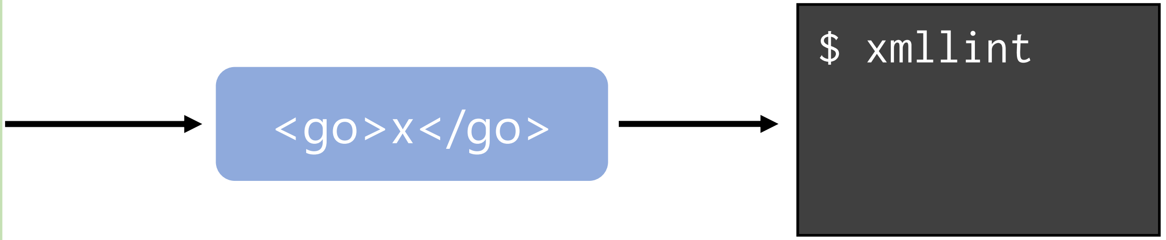


```
<a>bb</a>
```

```
$ xmllint
```

Generator-Based Fuzzing

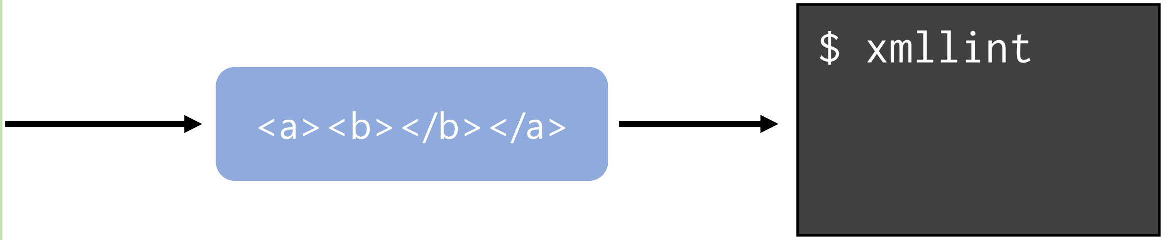
```python
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



```
$ xmllint
```

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
```

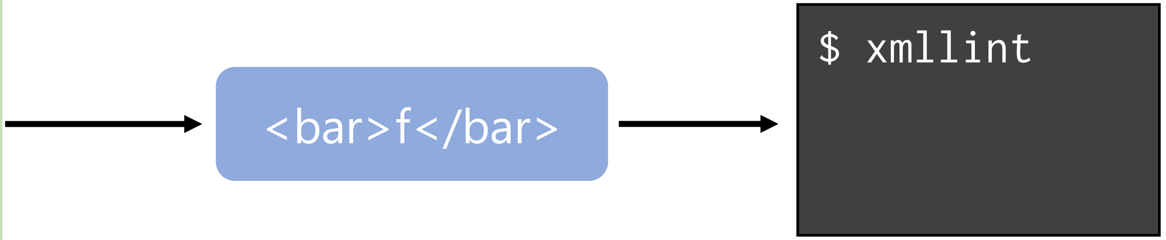


`<a><b></b></a>`

```
$ xmllint
```

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
```

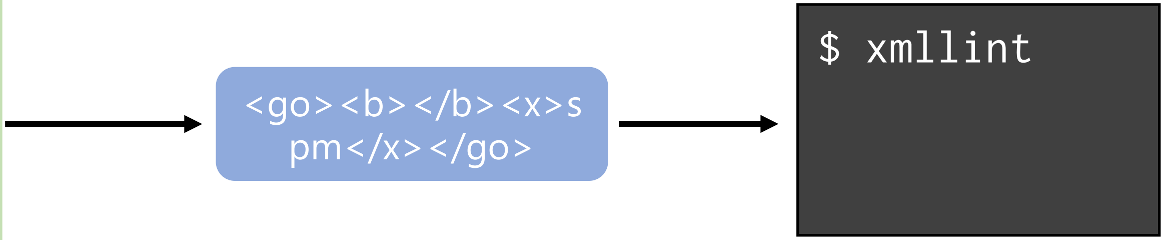


<bar>f</bar>

```
$ xmllint
```

Generator-Based Fuzzing

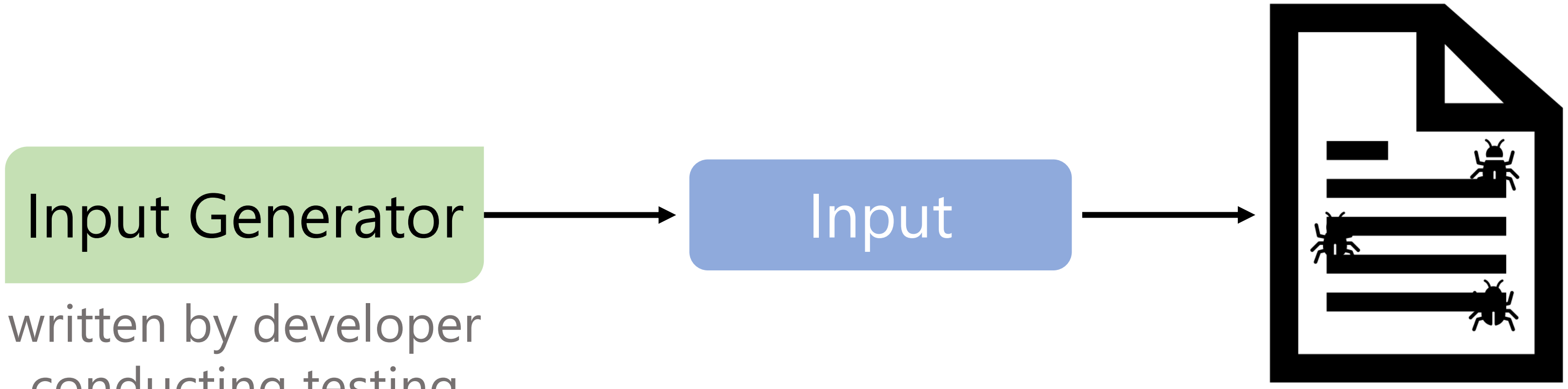
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



<go><b></b><x>spm</x></go>

```
$ xmllint
```

Generator-Based Fuzzing

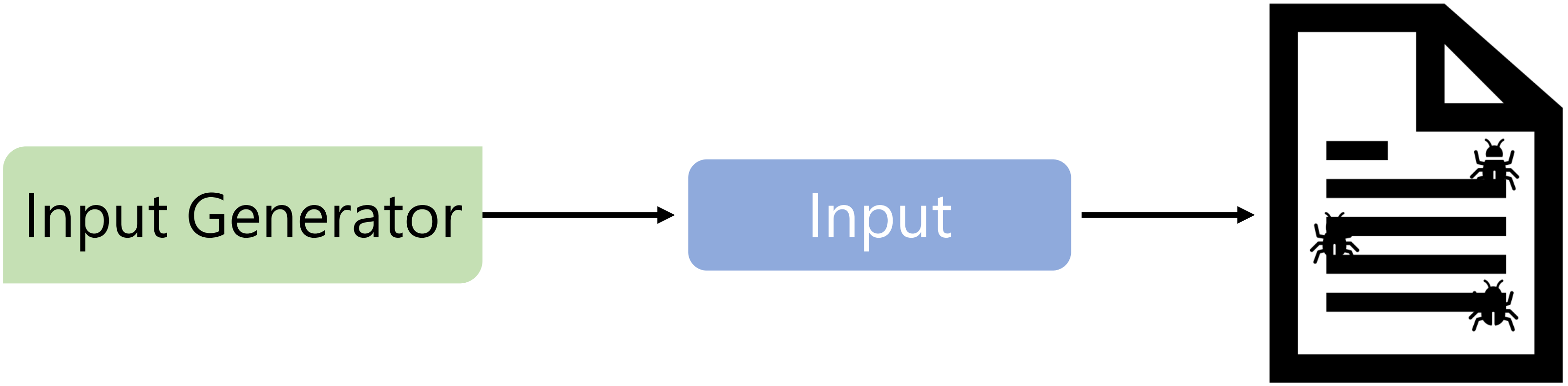


Input Generator

Input

written by developer conducting testing (or reuse a suitable one)

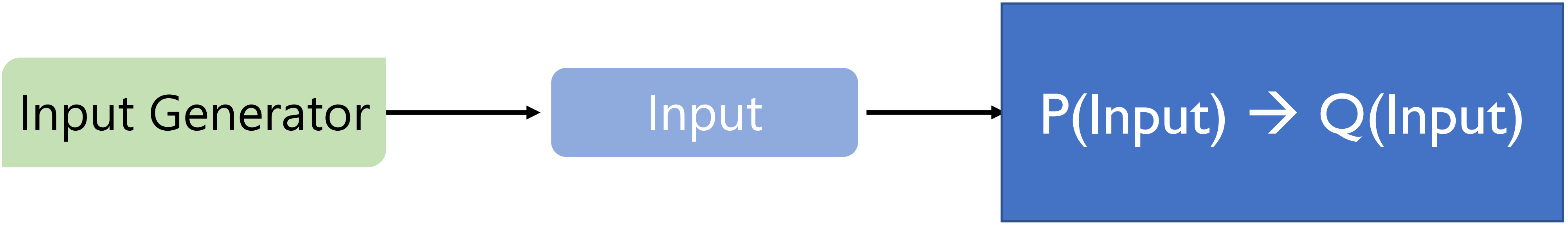
Property-Based Testing?


Input Generator

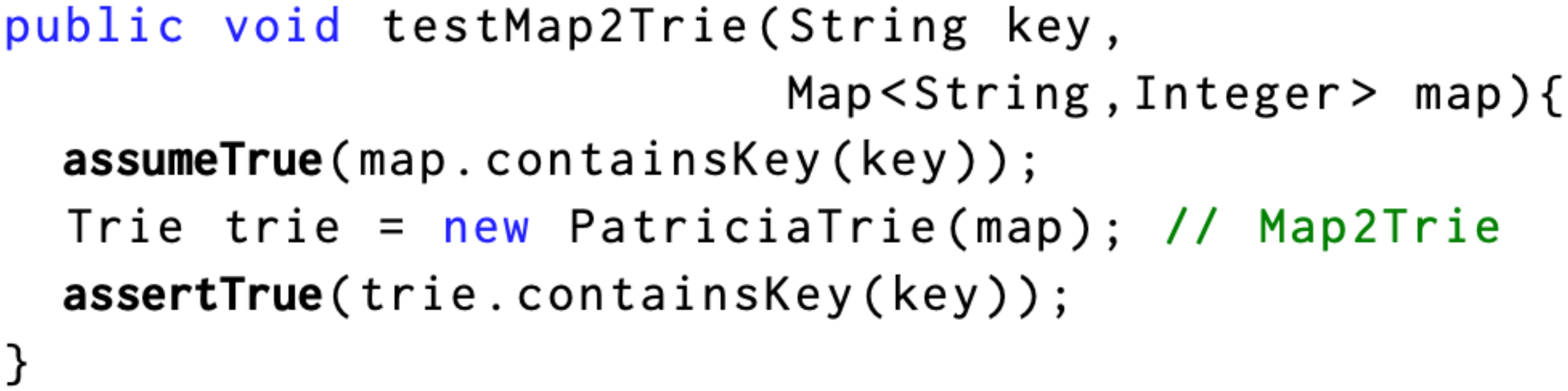
Input

Property-Based Testing

Make pre-conditions/post-conditions explicit in program under test



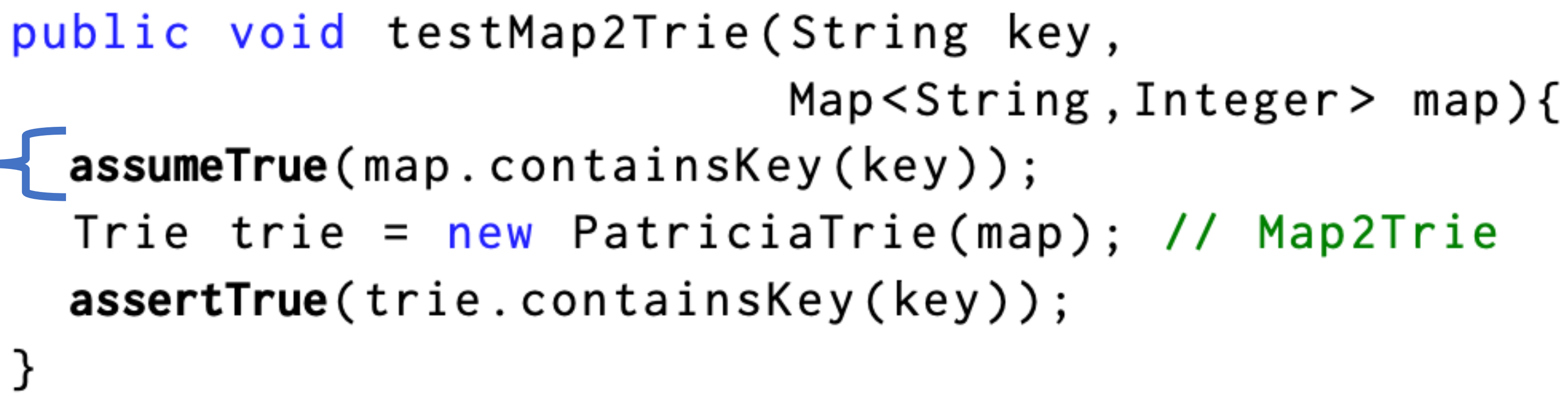
Pre + Post Conditions



```java
public void testMap2Trie(String key,
                         Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

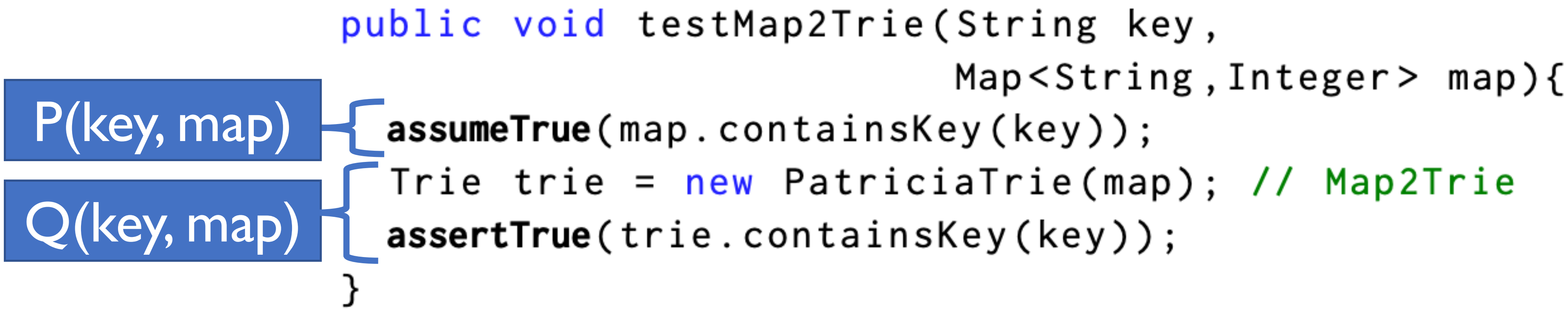
Pre + Post Conditions

P(key, map)



```
public void testMap2Trie(String key,
                           Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

Pre + Post Conditions

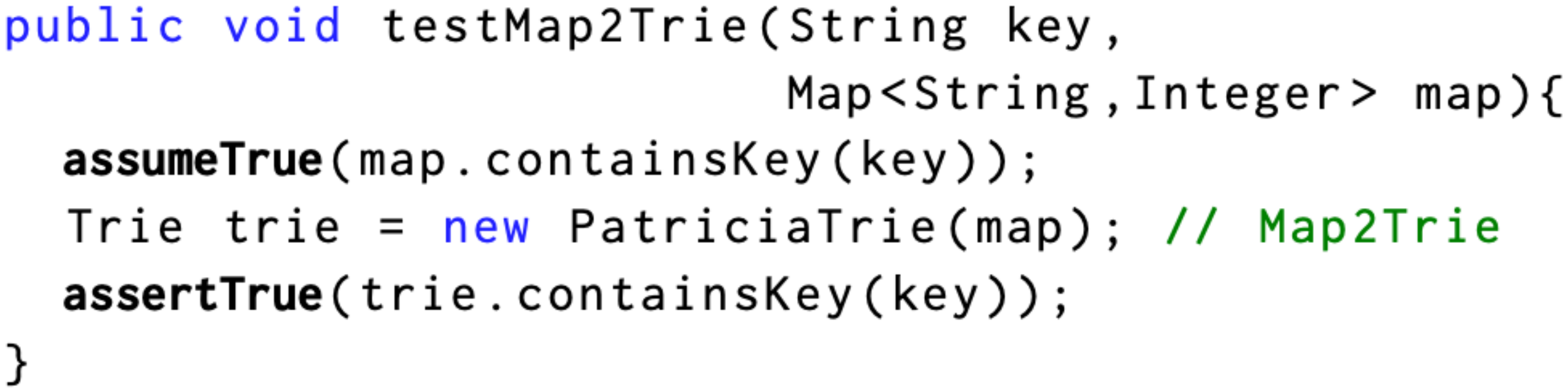


```
public void testMap2Trie(String key,
                         Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

P(key, map)

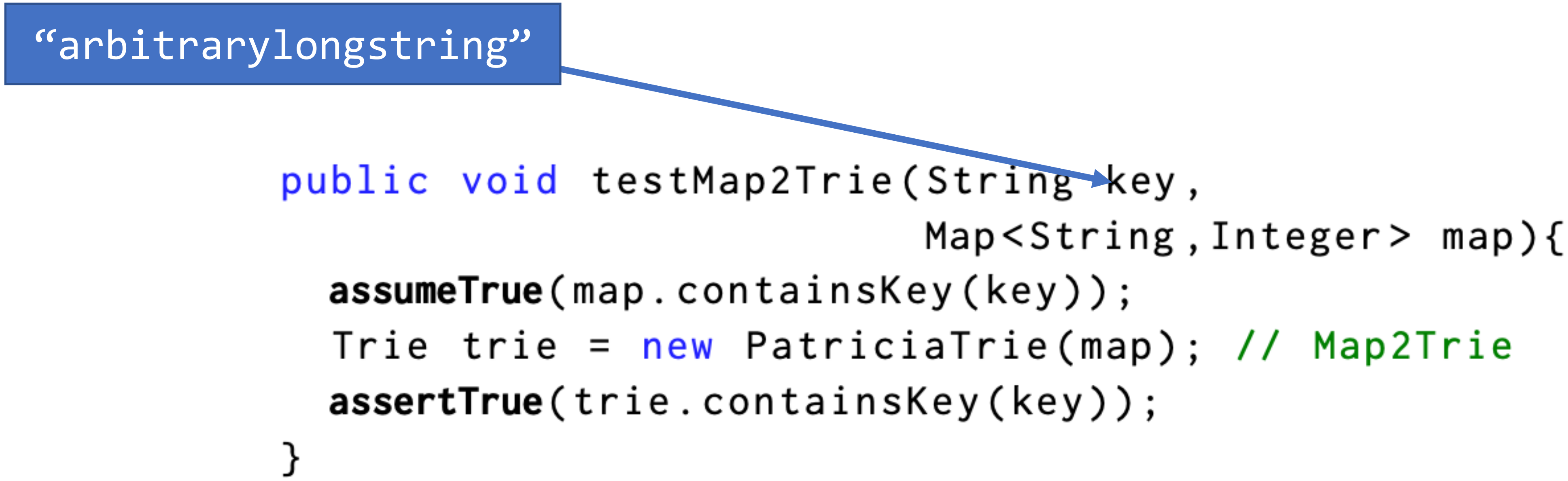
Q(key, map)

Shrinking



```
public void testMap2Trie(String key,
                         Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

Shrinking

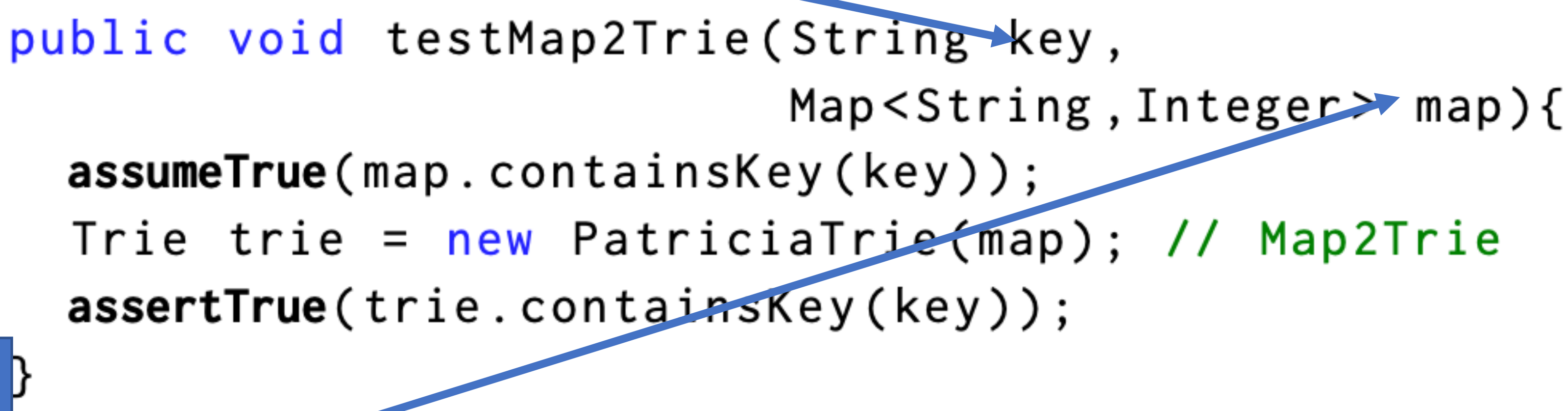


“arbitrarylongstring”

```java
public void testMap2Trie(String key,
                          Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

Shrinking

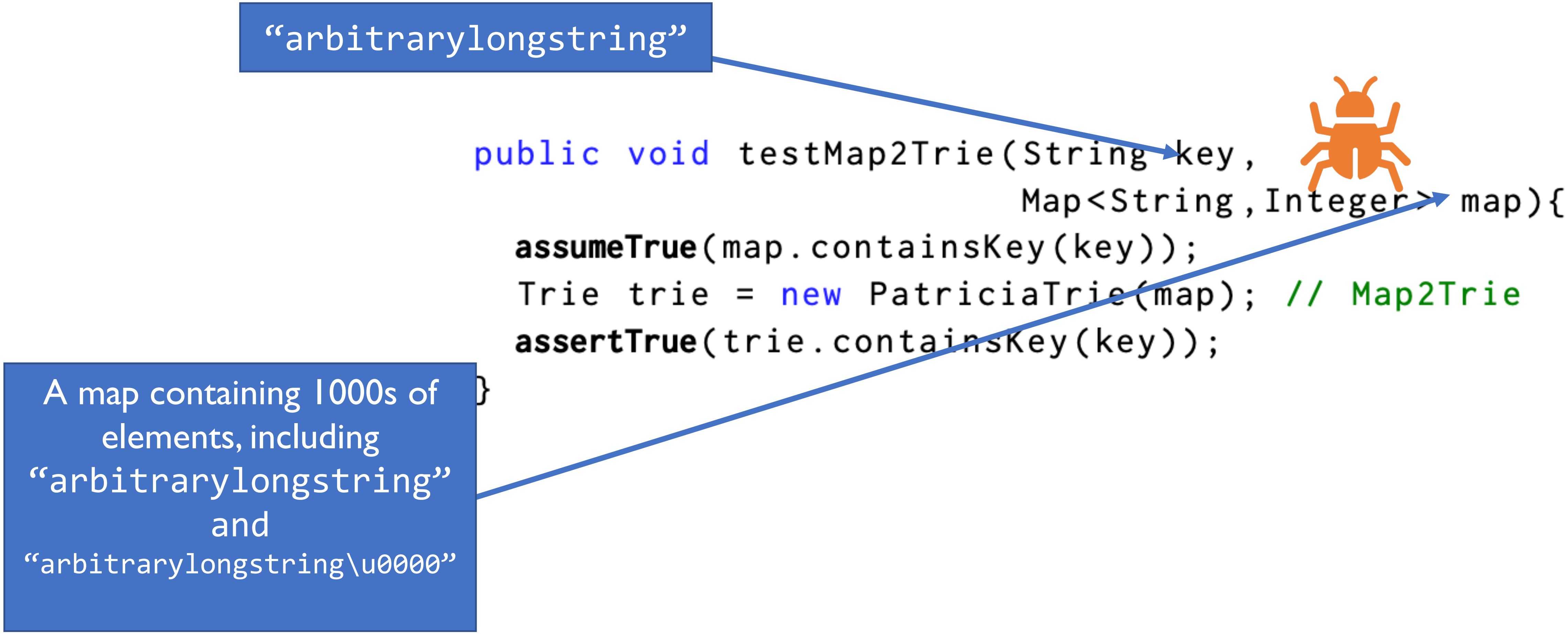
“arbitrarylongstring”



```
public void testMap2Trie(String key,
                         Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

A map containing 1000s of elements, including “arbitrarylongstring” and “arbitrarylongstring\u0000”

Shrinking

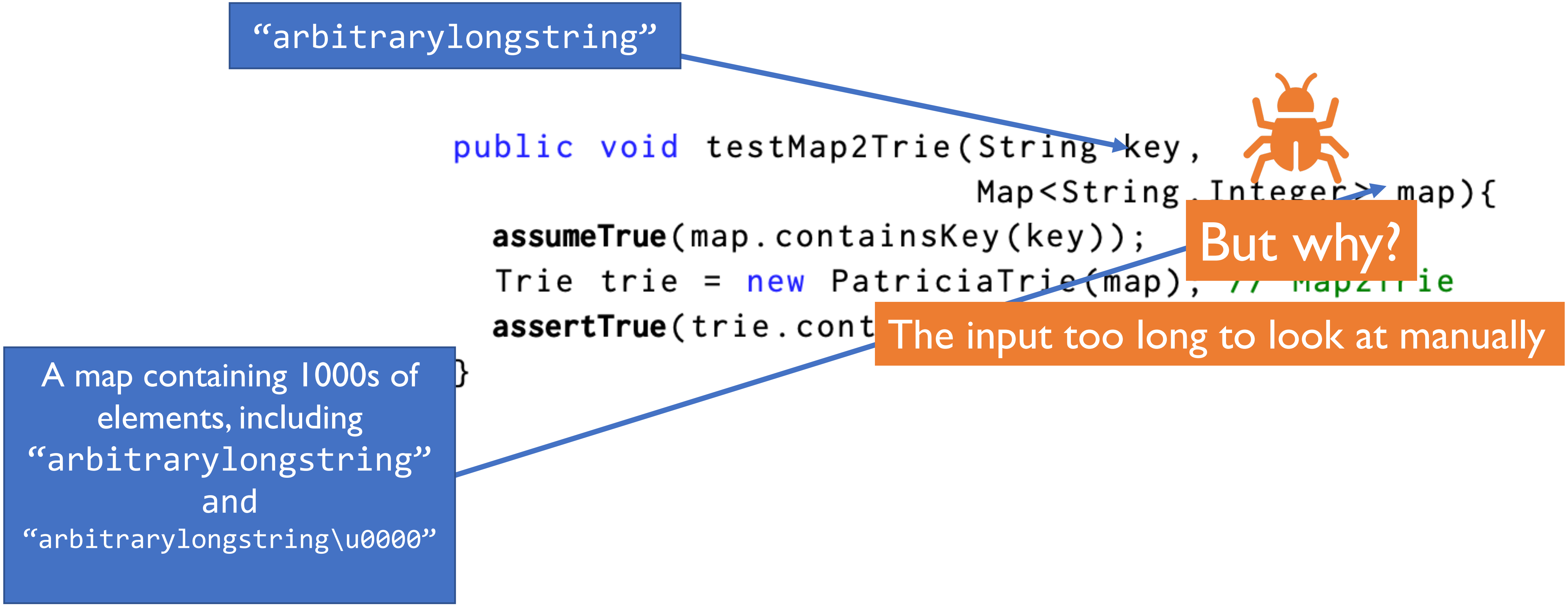


“arbitrarylongstring”

```java
public void testMap2Trie(String key,
                         Map<String,Integer> map){
  assumeTrue(map.containsKey(key));
  Trie trie = new PatriciaTrie(map); // Map2Trie
  assertTrue(trie.containsKey(key));
}
```

A map containing 1000s of elements, including “arbitrarylongstring” and “arbitrarylongstring\u0000”

Shrinking



“arbitrarylongstring”

```
public void testMap2Trie(String key,
                                    Map<String,Integer> map){
   assumeTrue(map.containsKey(key));
   Trie trie = new PatriciaTrie(map); // Map2Trie
   assertTrue(trie.cont
}
```

But why?

The input too long to look at manually

A map containing 1000s of elements, including “arbitrarylongstring” and “arbitrarylongstring\u0000”

Shrinking

shrink_T(input): produce a list of “shrinked” versions of input
- Call recursively on “smaller” values to shrink as much as possible

Shrinking

shrink_T(input): produce a list of “shrinked” versions of input
- Call recursively on “smaller” values to shrink as much as possible

```
shrink_point((x,y)) → [(0,0), (0, y/2), (x/2, 0), (x/2, y/2), (0, y), (0,x)]
```

Shrinking

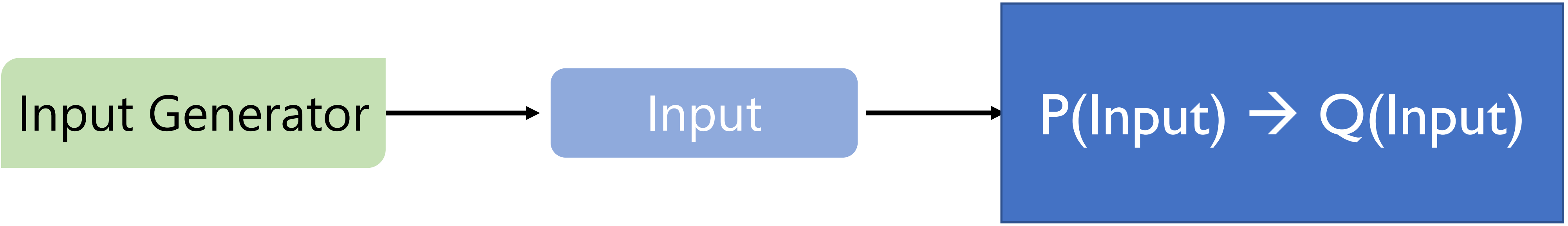
shrink_T(input): produce a list of "shrinked" versions of input
- Call recursively on "smaller" values to shrink as much as possible

```
shrink_point((x,y)) → [(0,0), (0, y/2), (x/2, 0), (x/2, y/2), (0, y), (0,x)]
```

Later in class: we will read the paper on delta-debugging, one way to "shrink" string-type inputs

- Coverage-guided fuzzing uses something like this to "shrink" inputs before mutation

Pros/Cons of Generator-Based Fuzzing?



How to have less random inputs?

Write a specification, generate inputs based on that specification

- Generator-based fuzzing
- Property-based testing
- Grammar-based fuzzing
- …

Start from existing inputs and alter them slightly

- mutational fuzzing

How to have less random inputs?

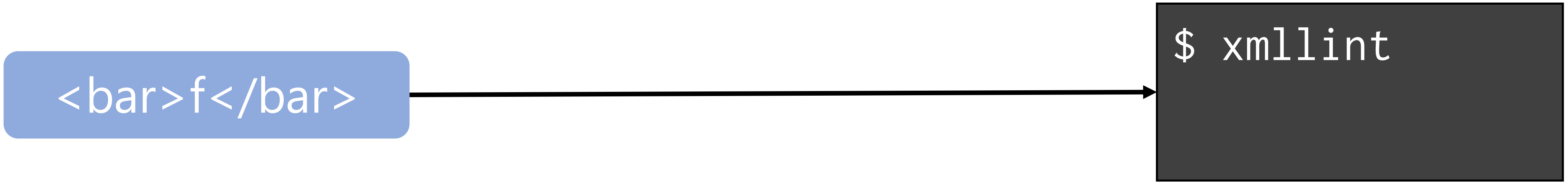
Write a specification, generate inputs based on that specification

- Generator-based fuzzing
- Property-based testing
- Grammar-based fuzzing
- …

Start from existing inputs and alter them slightly

- mutational fuzzing

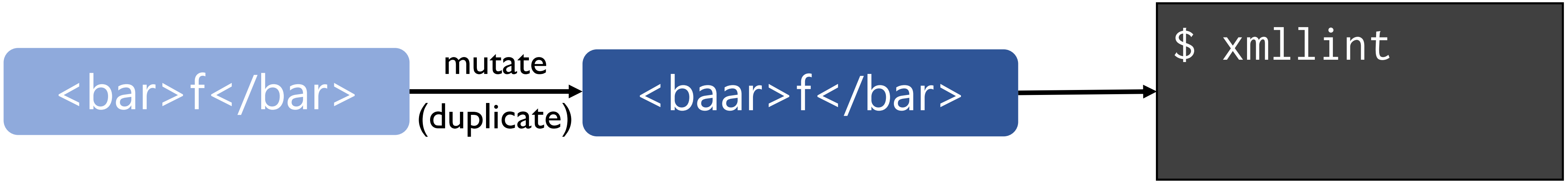
Mutational Fuzzing



<bar>f</bar>

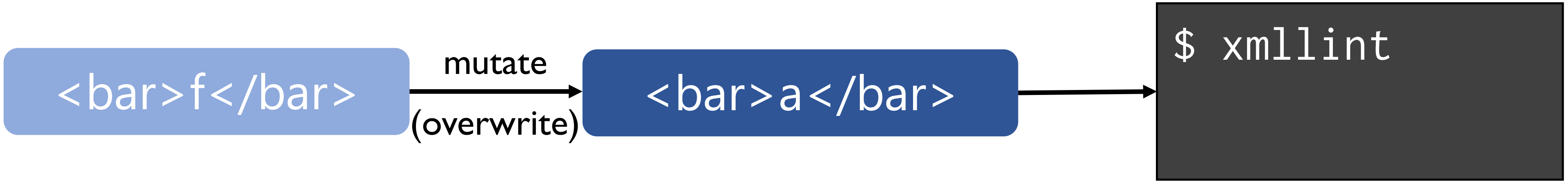
```
$ xmllint
```

Mutational Fuzzing



```
$ xmllint
```

Mutational Fuzzing



<bar>f</bar>

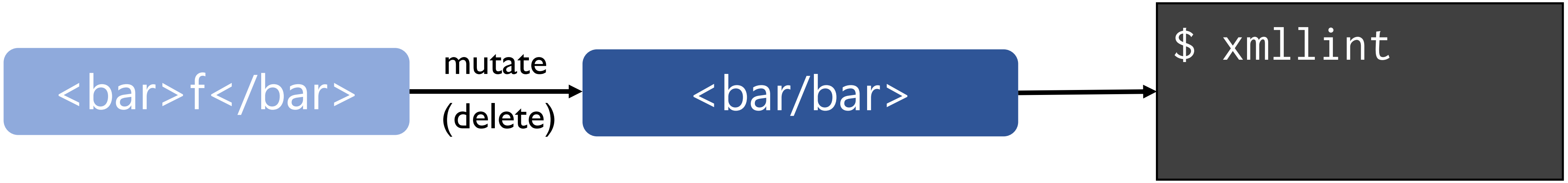
mutate

(overwrite)

<bar>a</bar>

```
$ xmllint
```

Mutational Fuzzing



<bar>f</bar>

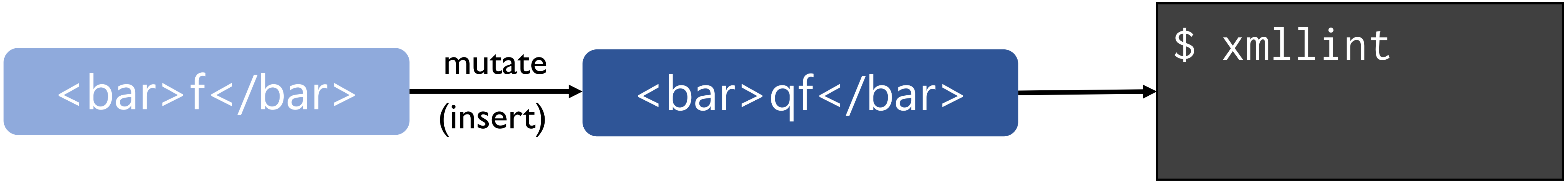
mutate

(delete)

<bar/bar>

```
$ xmllint
```

Mutational Fuzzing



<bar>f</bar>

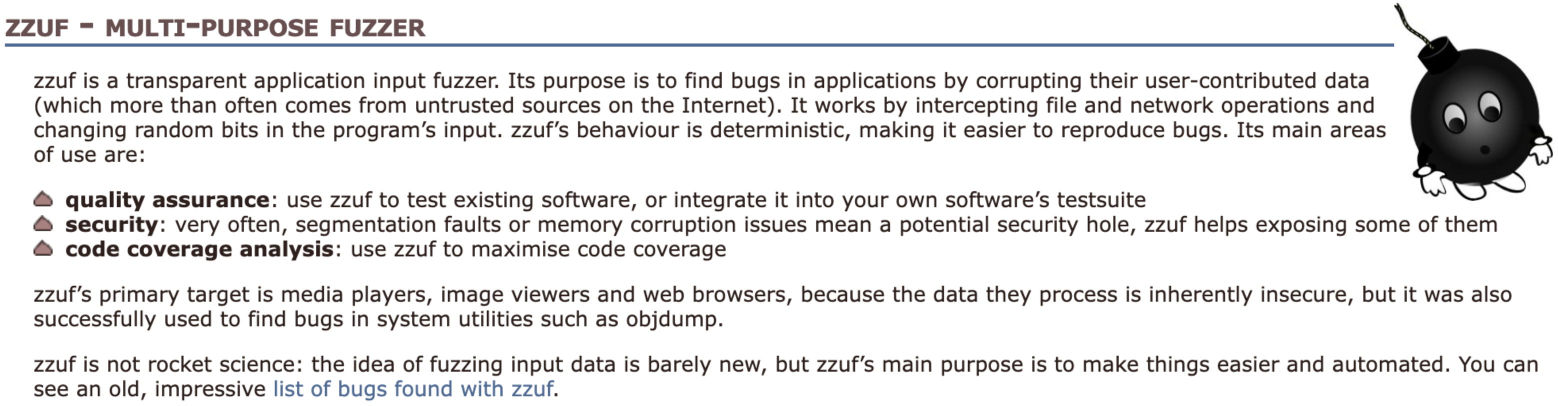
mutate

(insert)

<bar>qf</bar>

```
$ xmllint
```

Examples of Mutational Fuzzers



ZZUF – MULTI–PURPOSE FUZZER

zzuf is a transparent application input fuzzer. Its purpose is to find bugs in applications by corrupting their user-contributed data (which more than often comes from untrusted sources on the Internet). It works by intercepting file and network operations and changing random bits in the program's input. zzuf's behaviour is deterministic, making it easier to reproduce bugs. Its main areas of use are:

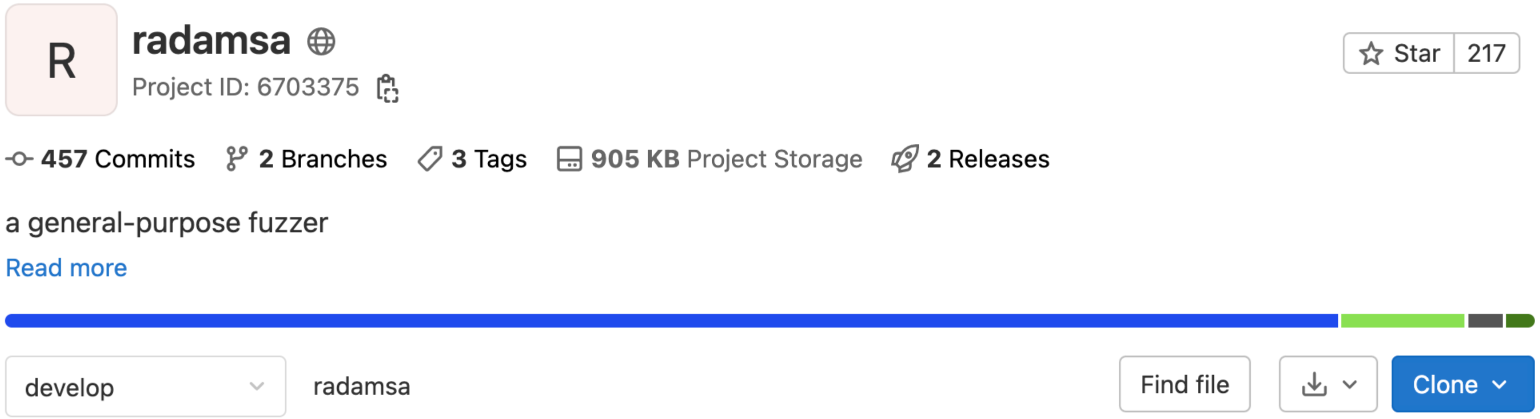
- **quality assurance**: use zzuf to test existing software, or integrate it into your own software's testsuite
- **security**: very often, segmentation faults or memory corruption issues mean a potential security hole, zzuf helps exposing some of them
- **code coverage analysis**: use zzuf to maximise code coverage

zzuf's primary target is media players, image viewers and web browsers, because the data they process is inherently insecure, but it was also successfully used to find bugs in system utilities such as objdump.

zzuf is not rocket science: the idea of fuzzing input data is barely new, but zzuf's main purpose is to make things easier and automated. You can see an old, impressive list of bugs found with zzuf.

`zzuf`

Maintained 2006-2016



radamsa

Project ID: 6703375

Star 217

457 Commits 2 Branches 3 Tags 905 KB Project Storage 2 Releases

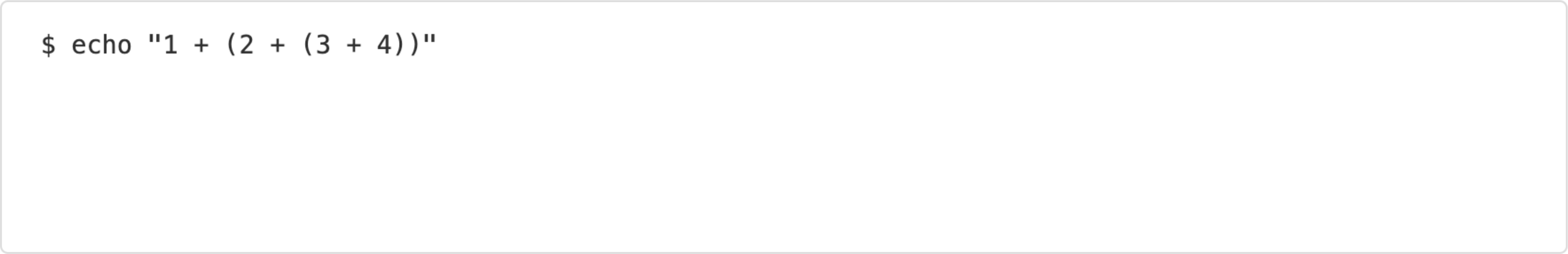
a general-purpose fuzzer

Read more

`radamsa`

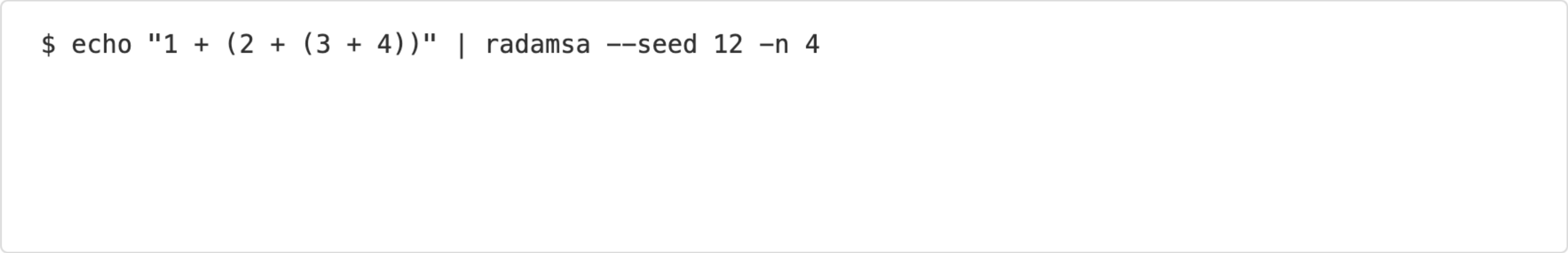
Maintained 2007-now?

Example



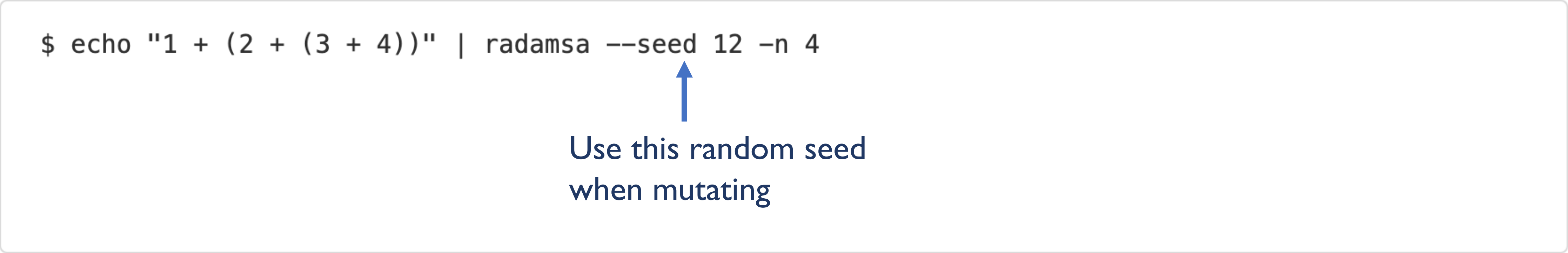
```
$ echo "1 + (2 + (3 + 4))"
```

Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
```

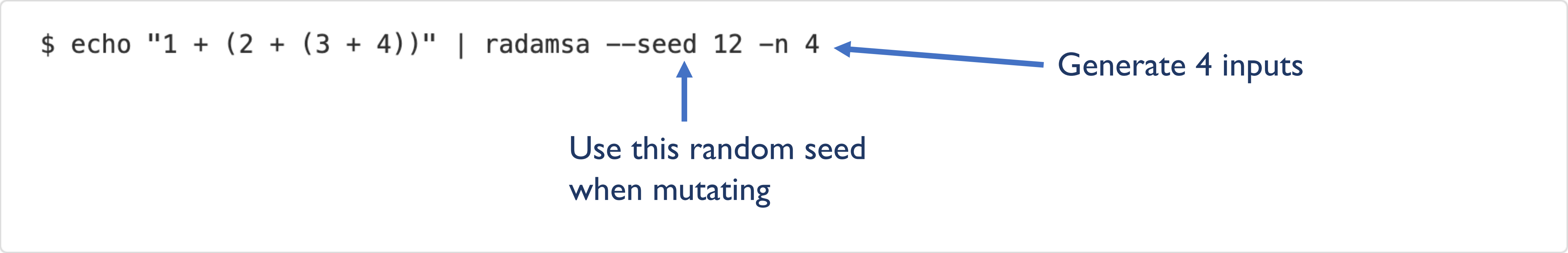
Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
```

Use this random seed when mutating

Example

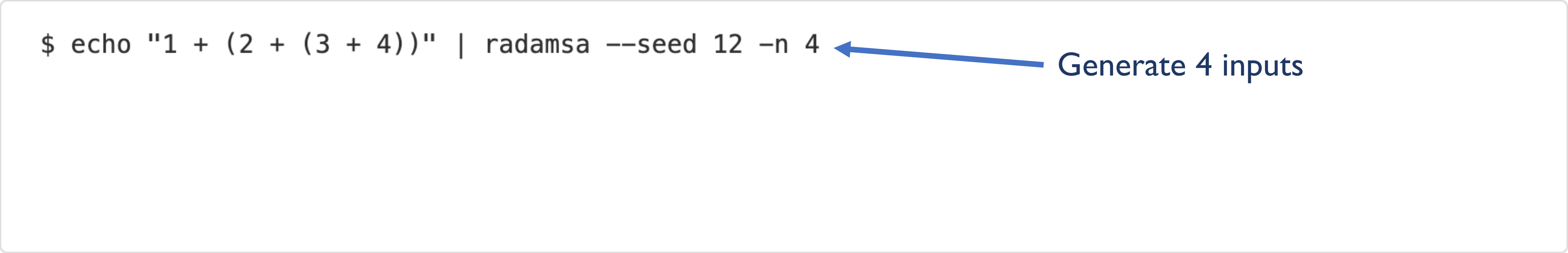


```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
```

Generate 4 inputs

Use this random seed
when mutating

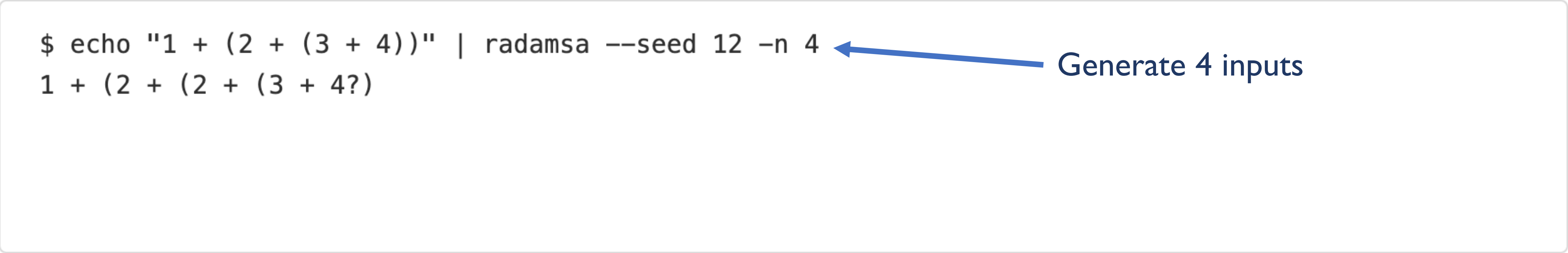
Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
```

Generate 4 inputs

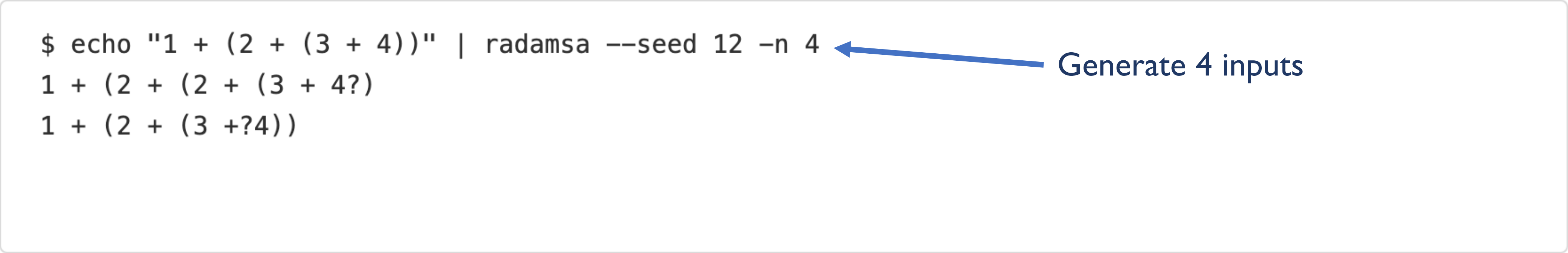
Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
```

Generate 4 inputs

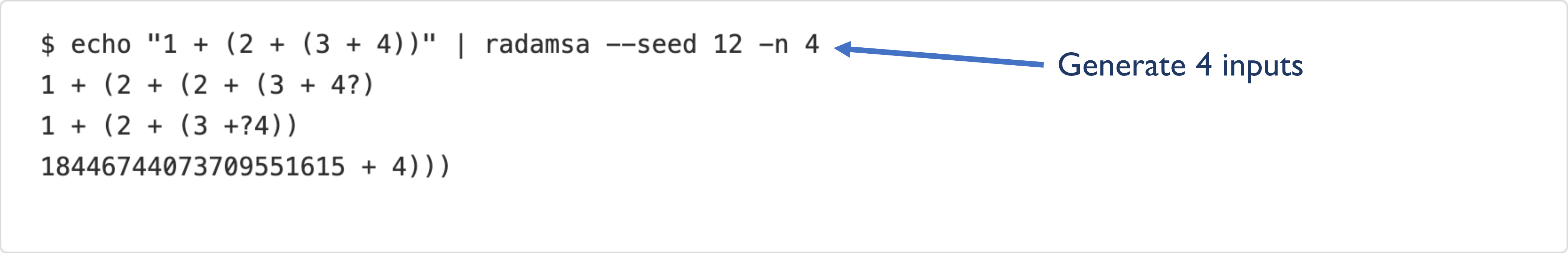
Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
```

Generate 4 inputs

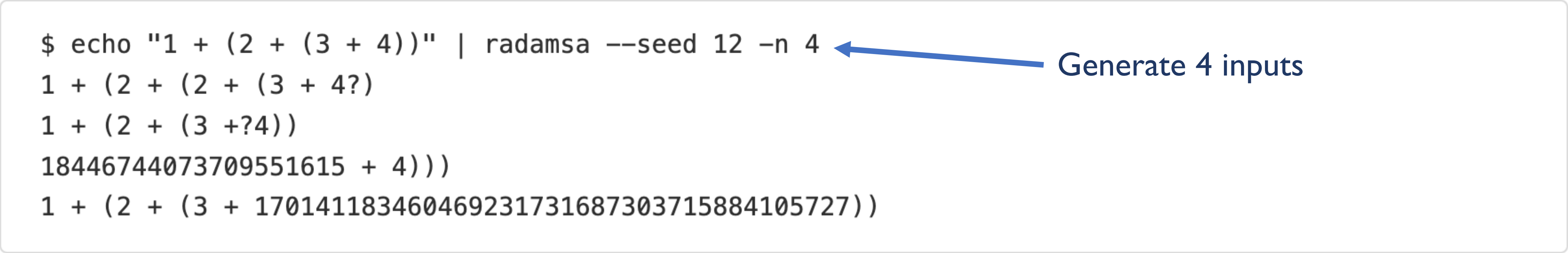
Example



```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
18446744073709551615 + 4)))
```

Generate 4 inputs

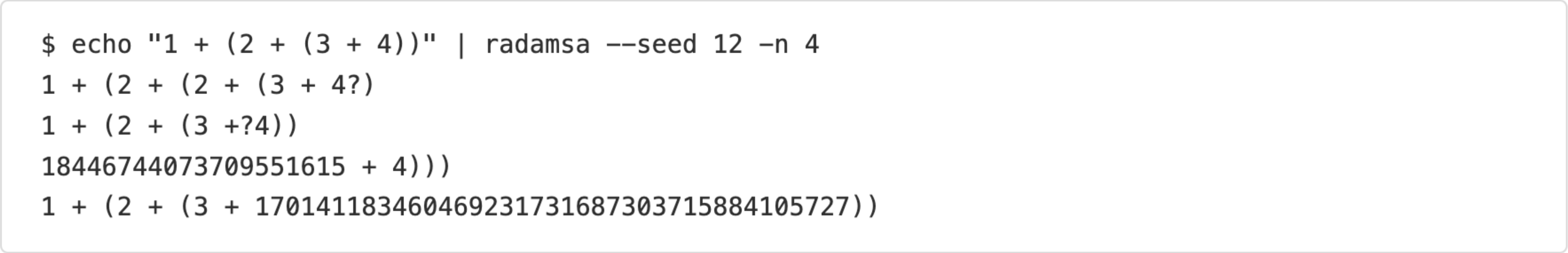
Example



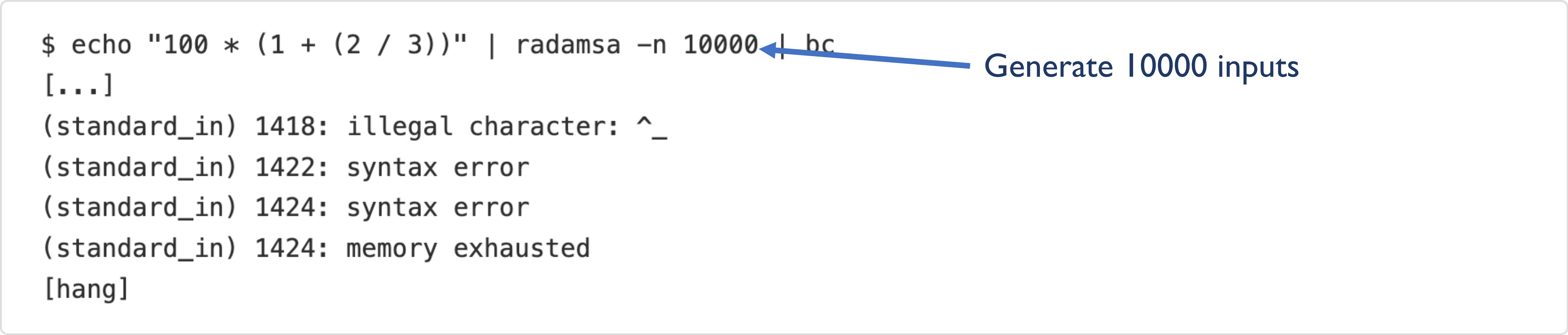
```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
18446744073709551615 + 4)))
1 + (2 + (3 + 170141183460469231731687303715884105727))
```

Generate 4 inputs

Example



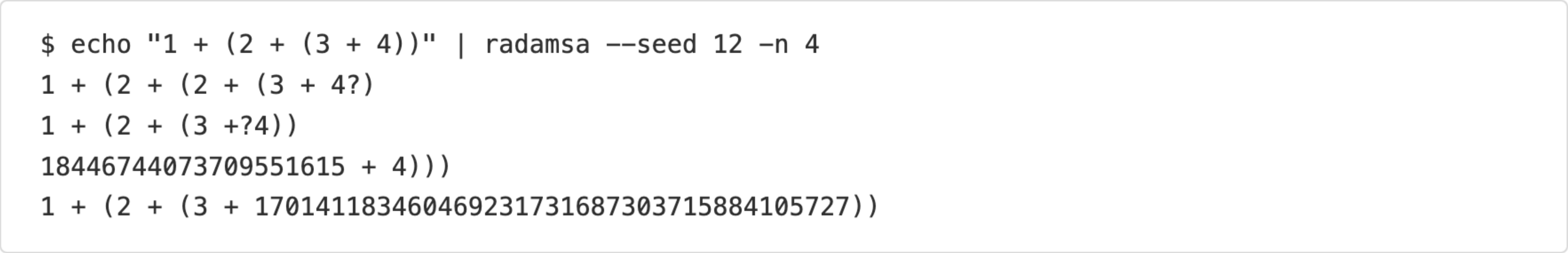
```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
18446744073709551615 + 4)))
1 + (2 + (3 + 170141183460469231731687303715884105727))
```



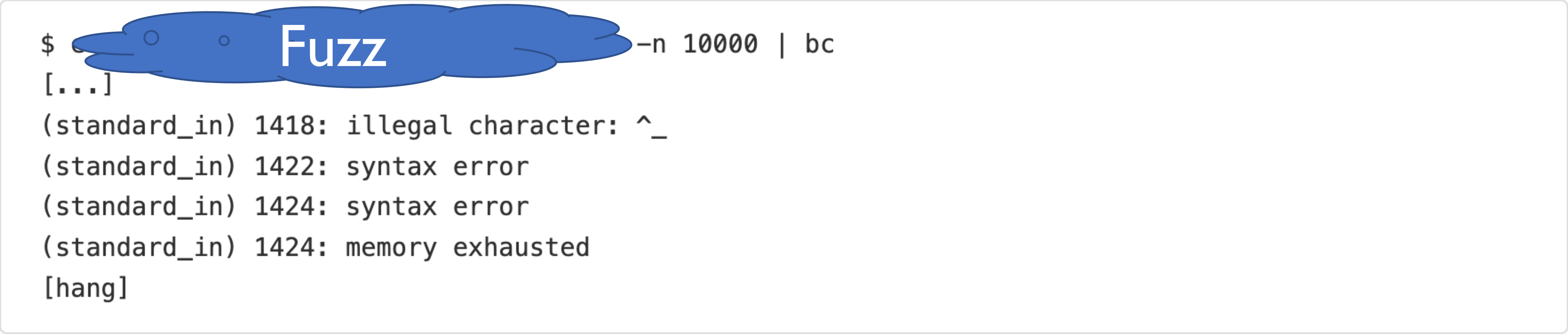
```
$ echo "100 * (1 + (2 / 3))" | radamsa -n 10000 | bc
[...]
(standard_in) 1418: illegal character: ^_
(standard_in) 1422: syntax error
(standard_in) 1424: syntax error
(standard_in) 1424: memory exhausted
[hang]
```

Generate 10000 inputs

Example

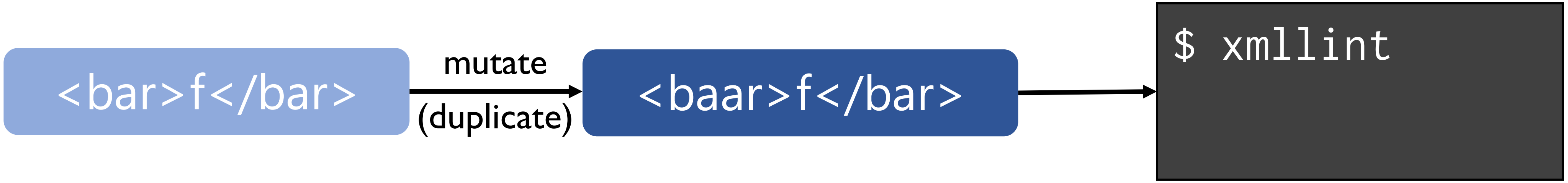


```
$ echo "1 + (2 + (3 + 4))" | radamsa --seed 12 -n 4
1 + (2 + (2 + (3 + 4?)
1 + (2 + (3 +?4))
18446744073709551615 + 4)))
1 + (2 + (3 + 170141183460469231731687303715884105727))
```



```
$ Fuzz -n 10000 | bc
[...]
(standard_in) 1418: illegal character: ^_
(standard_in) 1422: syntax error
(standard_in) 1424: syntax error
(standard_in) 1424: memory exhausted
[hang]
```

Pros/Cons of Mutational Fuzzers?

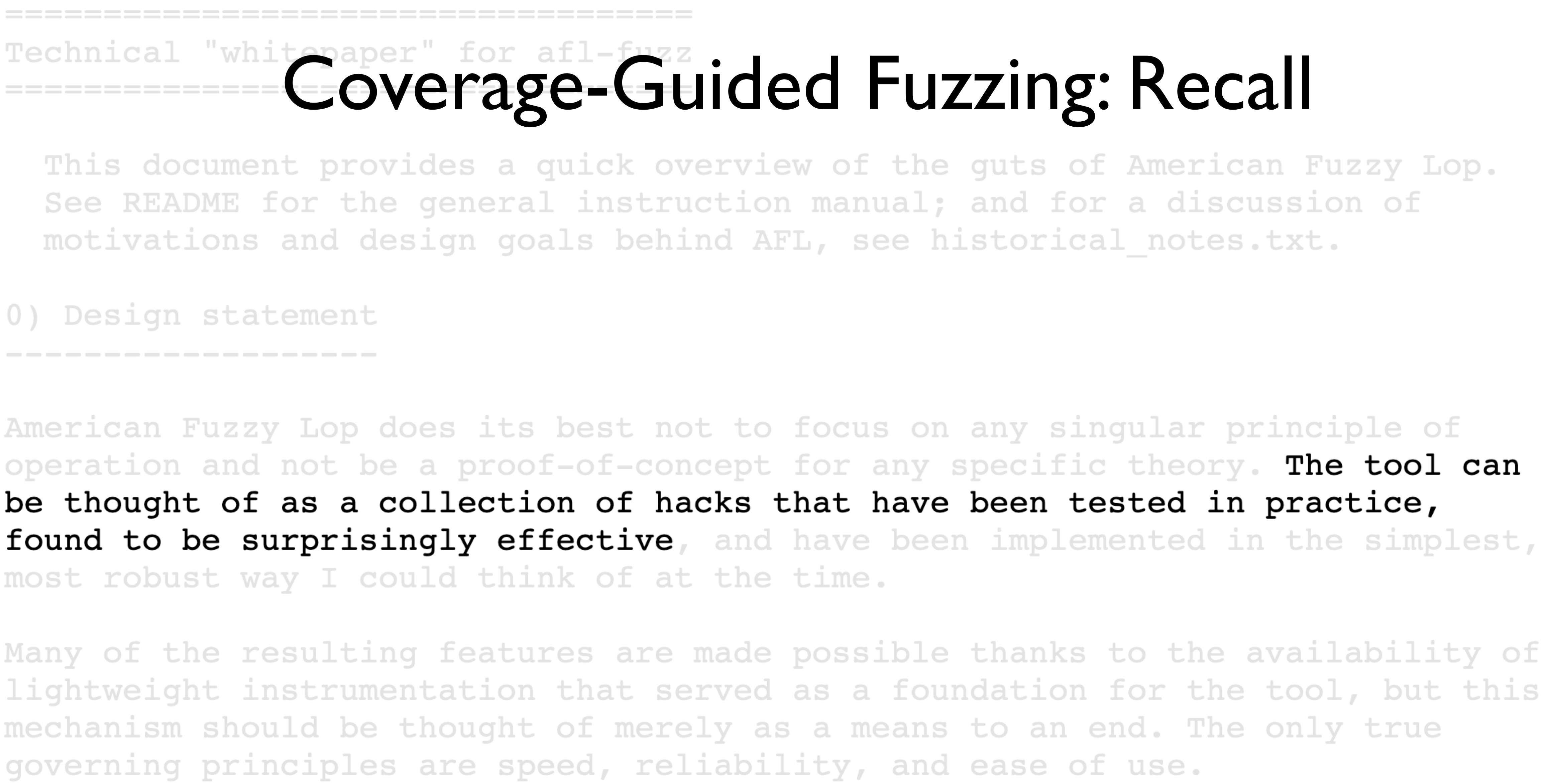


```
$ xmllint
```

Schedule for Today

- Improving upon pure random fuzzing
- Coverage-guided fuzzing
  - a.k.a. greybox fuzzing, a.k.a. coverage-based greybox fuzzing
- Relation to Evolutionary Algorithms

Coverage-Guided Fuzzing: Recall



===================================
Technical "whitepaper" for afl-fuzz
===================================

This document provides a quick overview of the guts of American Fuzzy Lop. See README for the general instruction manual; and for a discussion of motivations and design goals behind AFL, see historical_notes.txt.

0) Design statement

American Fuzzy Lop does its best not to focus on any singular principle of operation and not be a proof-of-concept for any specific theory. **The tool can be thought of as a collection of hacks that have been tested in practice, found to be surprisingly effective**, and have been implemented in the simplest, most robust way I could think of at the time.

Many of the resulting features are made possible thanks to the availability of lightweight instrumentation that served as a foundation for the tool, but this mechanism should be thought of merely as a means to an end. The only true governing principles are speed, reliability, and ease of use.

Coverage

Line coverage: which lines are executed?

```
def foo(x, y):
    z = 2 * x
    if z > y:
        z = y
    return z + y
```

Coverage

Line coverage: which lines are executed?

```
def foo(x, y):
     z = 2 * x
     if z > y:
         z = y
     return z + y
```

```
foo(3,2)
```

Coverage

Line coverage: which lines are executed?

Branch coverage: are both sides of an if() executed?

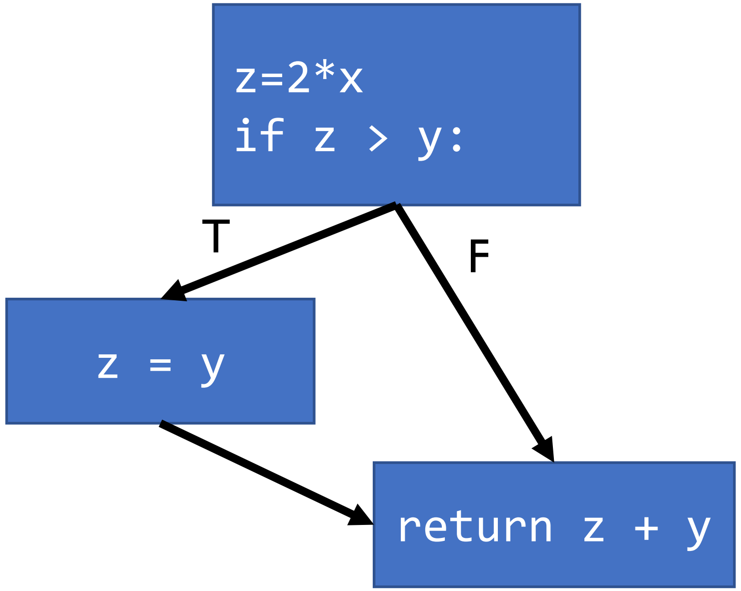
```
def foo(x, y):
    z = 2 * x
    if z > y:
        z = y
    return z + y
```

```
foo(3,7)
```

Coverage

Edge coverage: coverage of edges in control-flow graph of a program

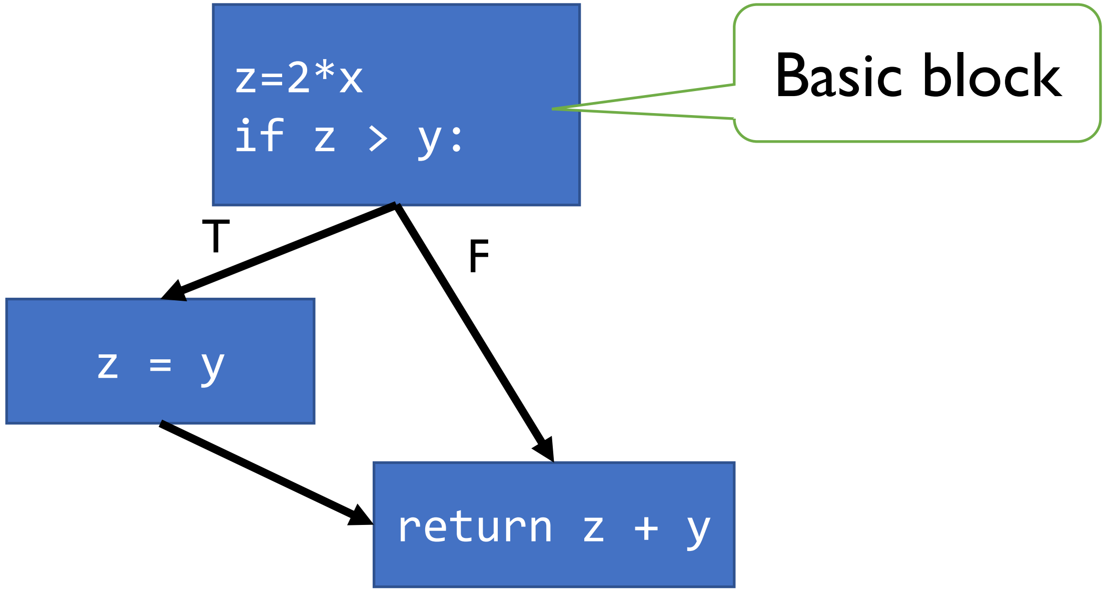
```
def foo(x, y):
    z = 2 * x
    if z > y:
        z = y
    return z + y
```



Coverage

Edge coverage: coverage of edges in control-flow graph of a program

```
def foo(x, y):
        z = 2 * x
        if z > y:
                z = y
        return z + y
```



```
z=2*x
if z > y:
```

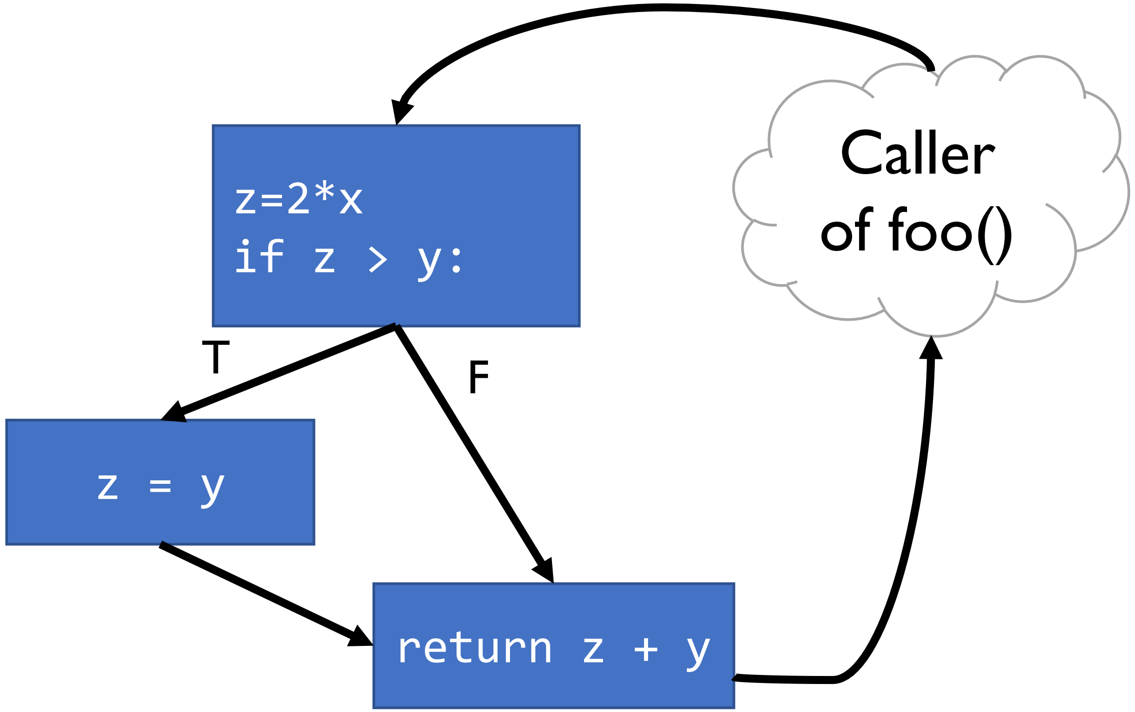
```
z = y
```

```
return z + y
```

Coverage

Edge coverage: coverage of edges in control-flow graph of a program

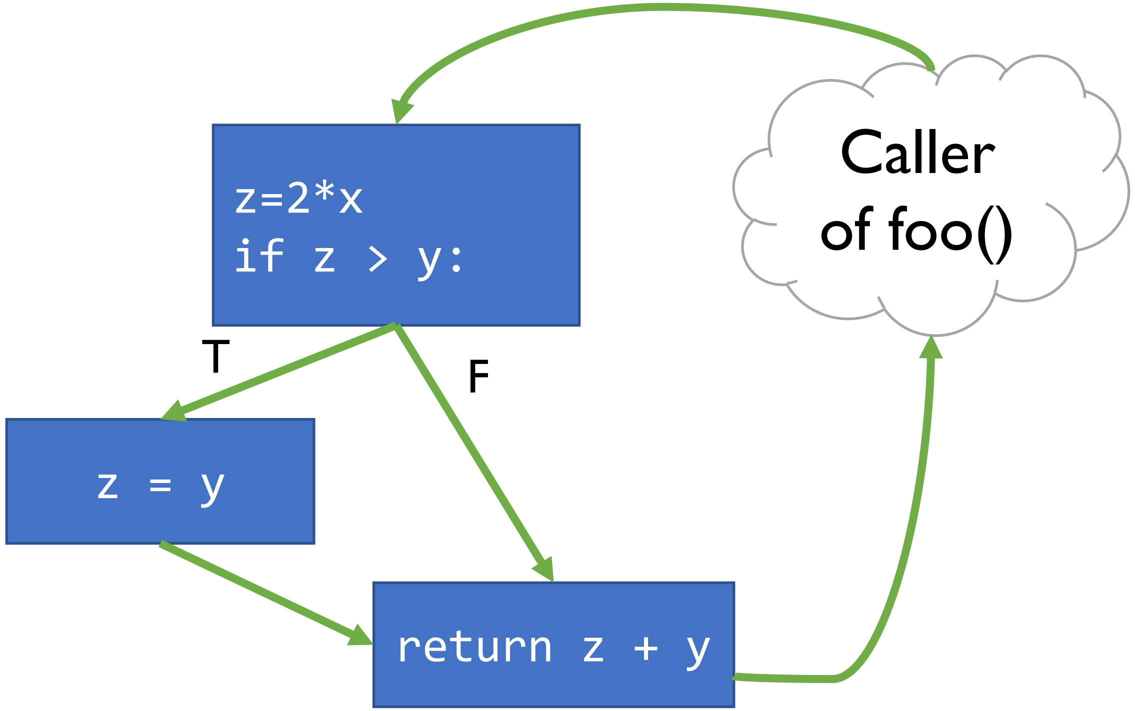
```
def foo(x, y):
    z = 2 * x
    if z > y:
        z = y
    return z + y
```



Coverage

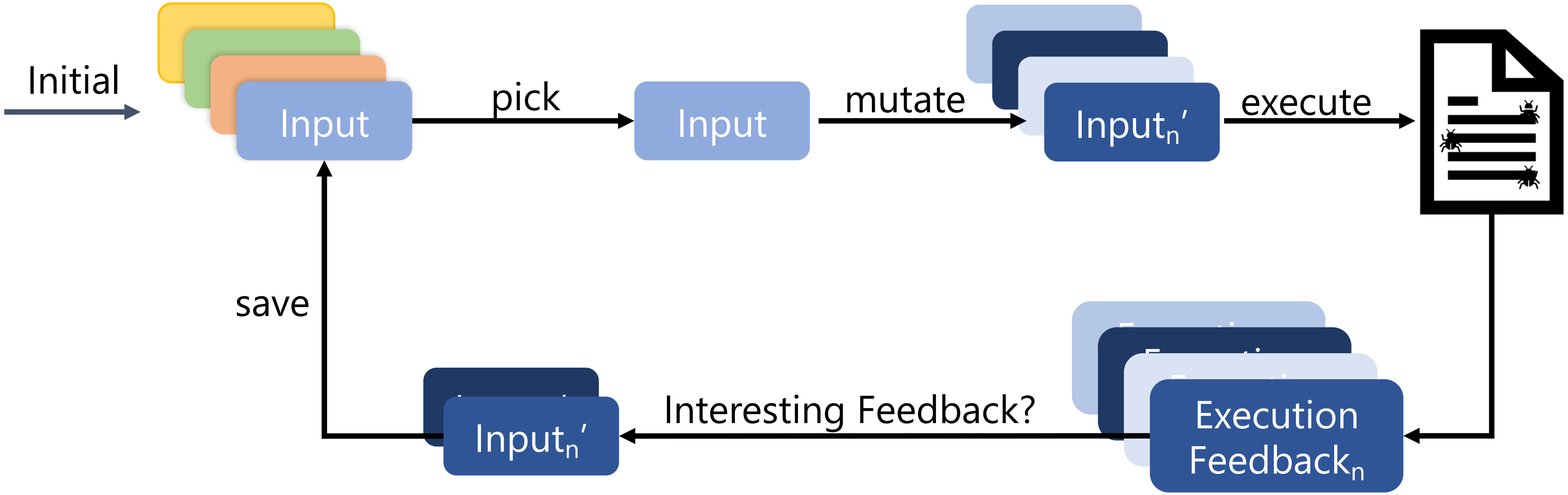
Edge coverage: coverage of edges in control-flow graph of a program

```
def foo(x, y):
    z = 2 * x
    if z > y:
        z = y
    return z + y
```



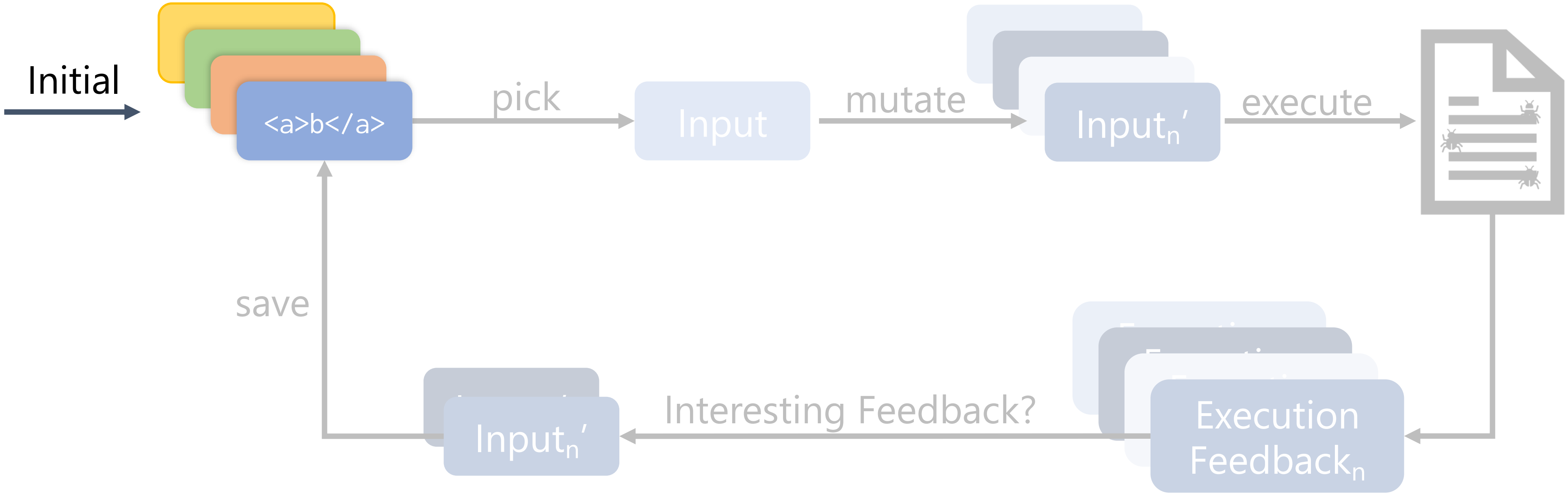
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



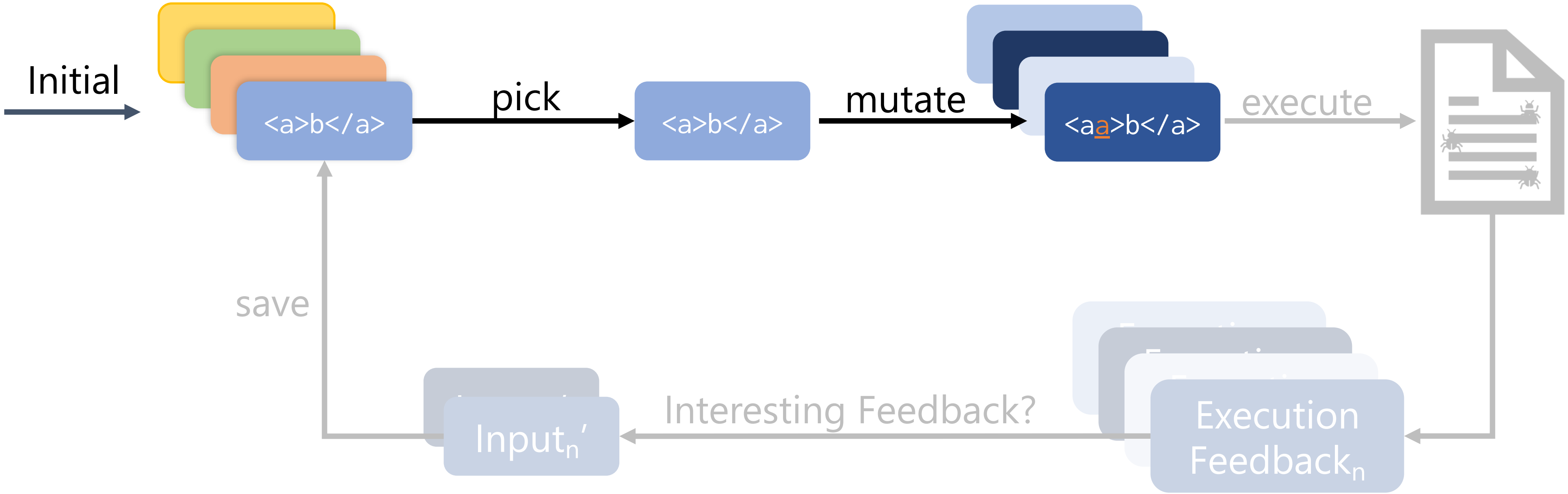
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



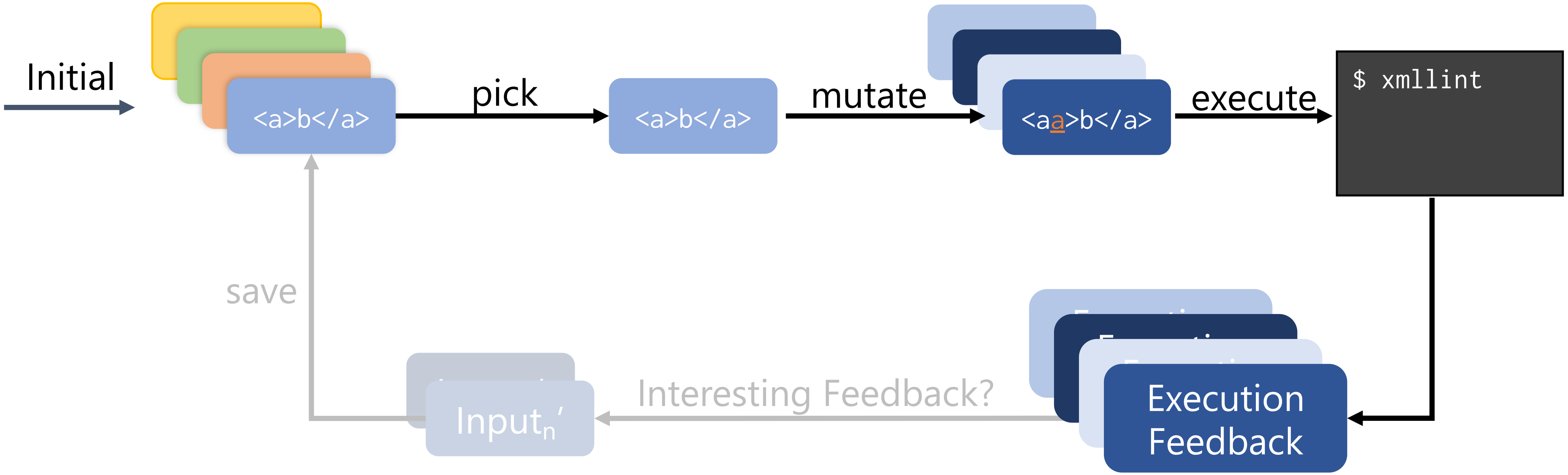
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



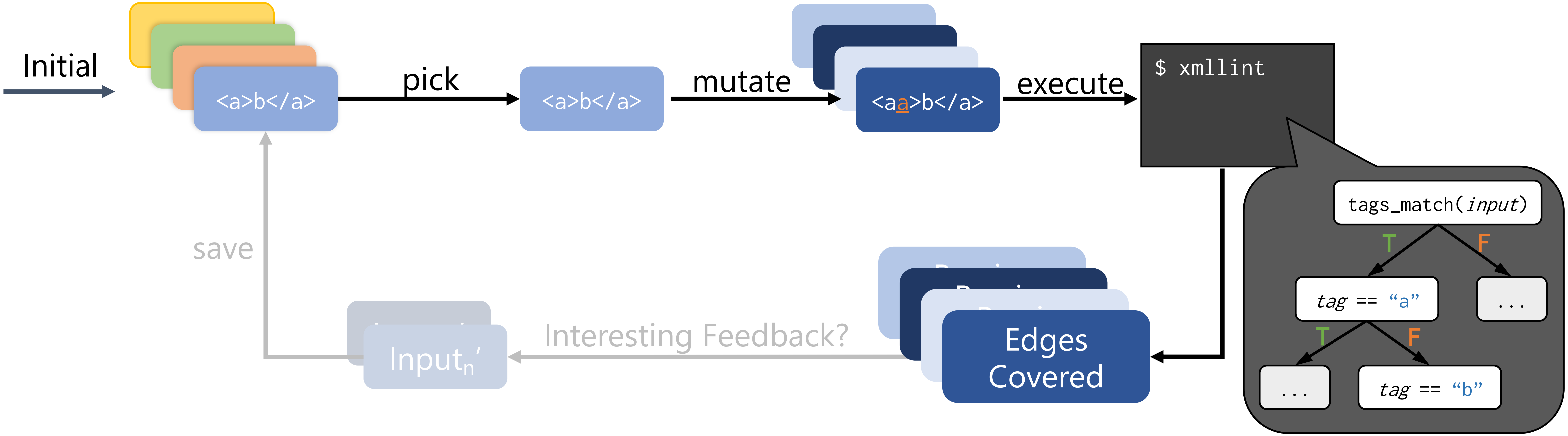
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



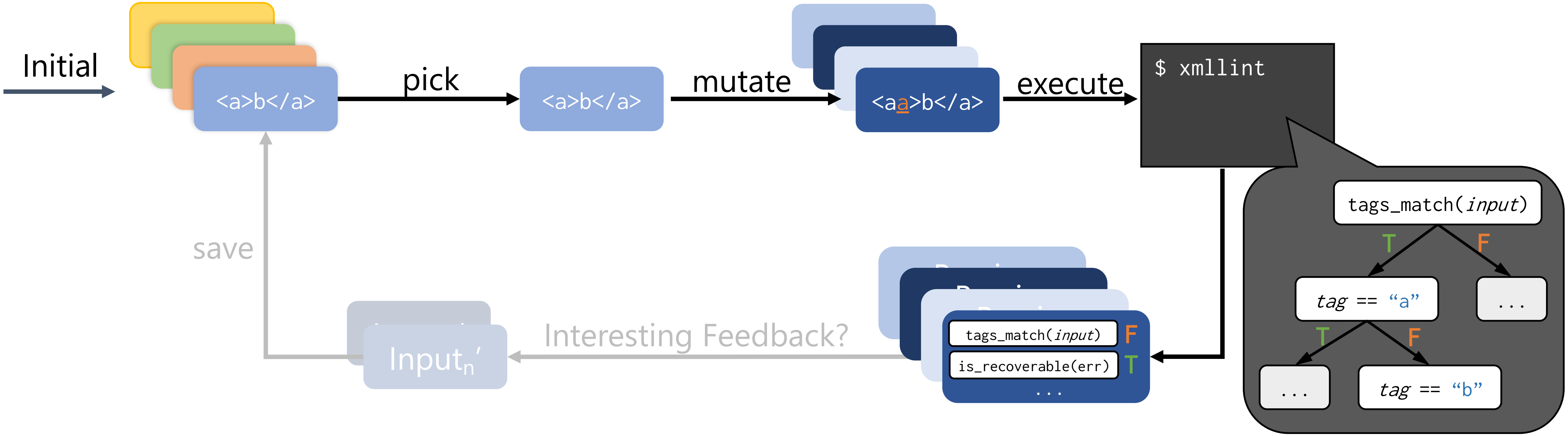
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



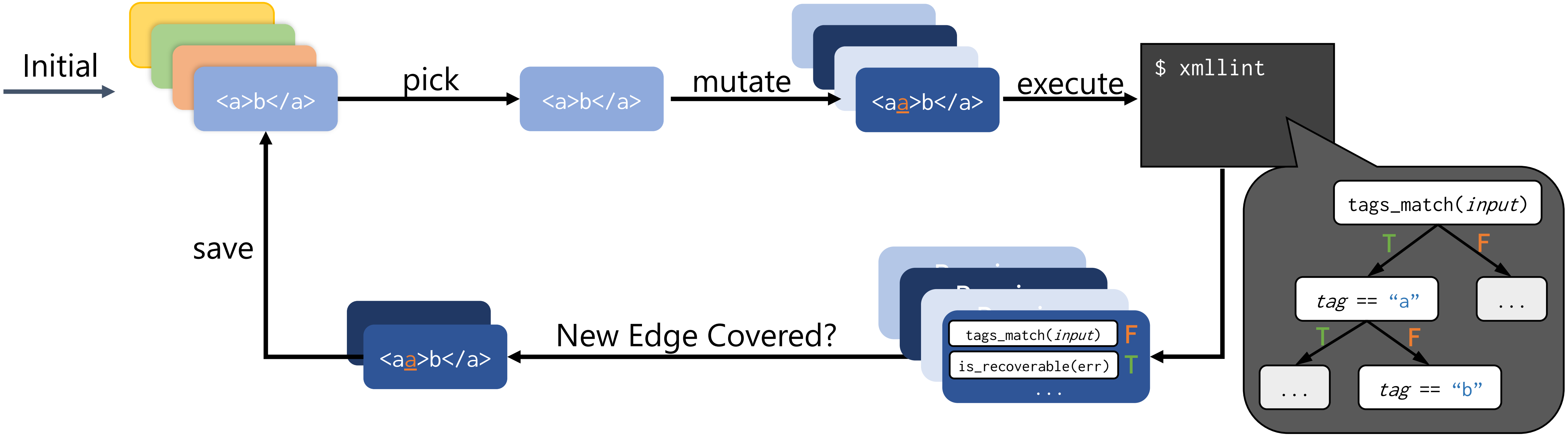
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



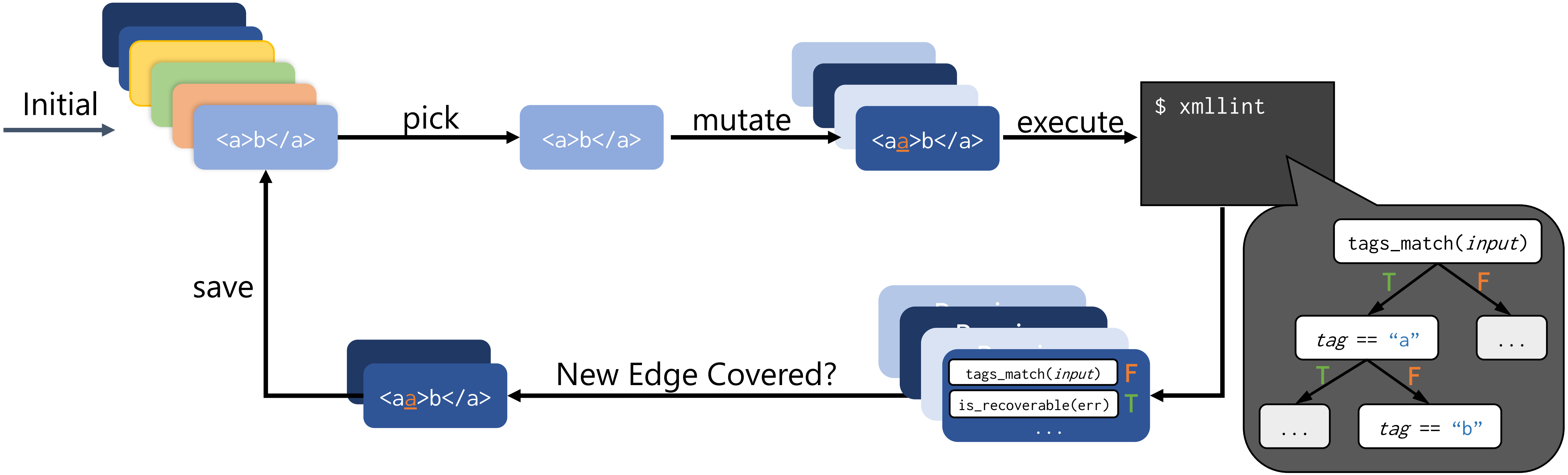
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



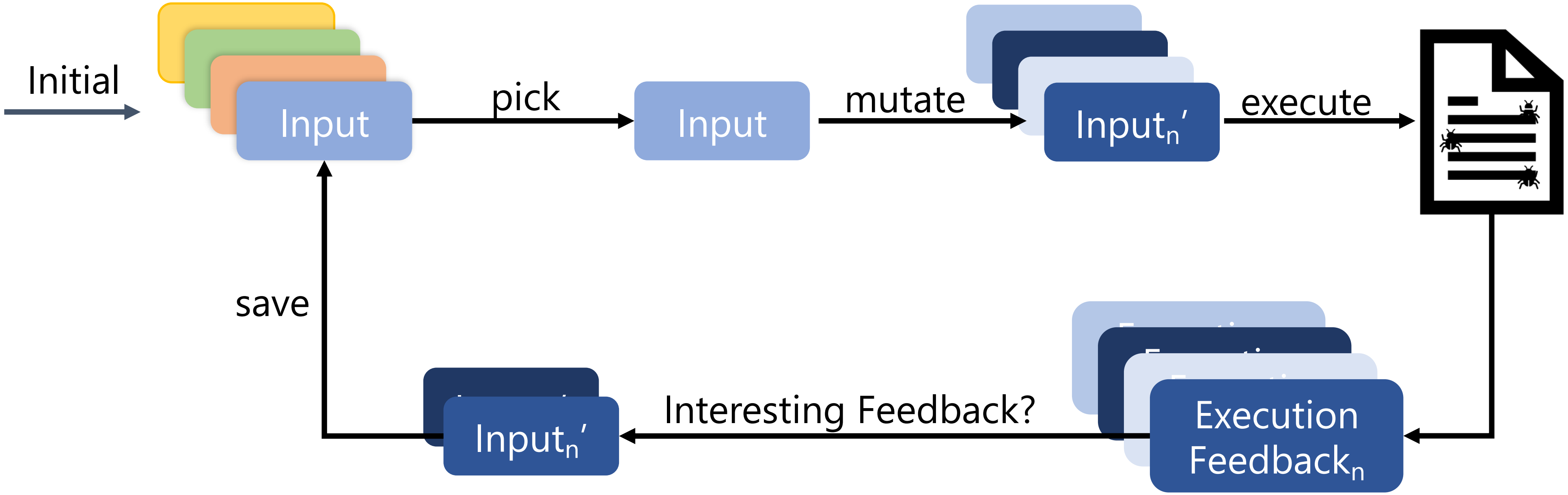
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



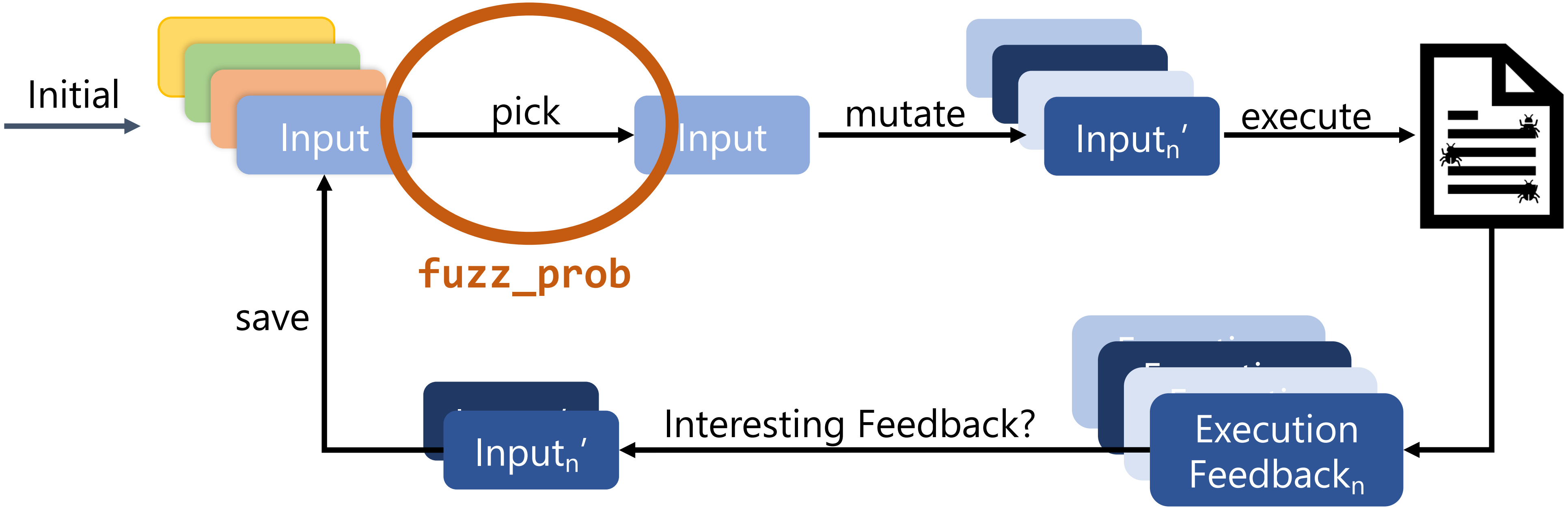
Coverage-Guided Fuzzing

Relation to Assignment



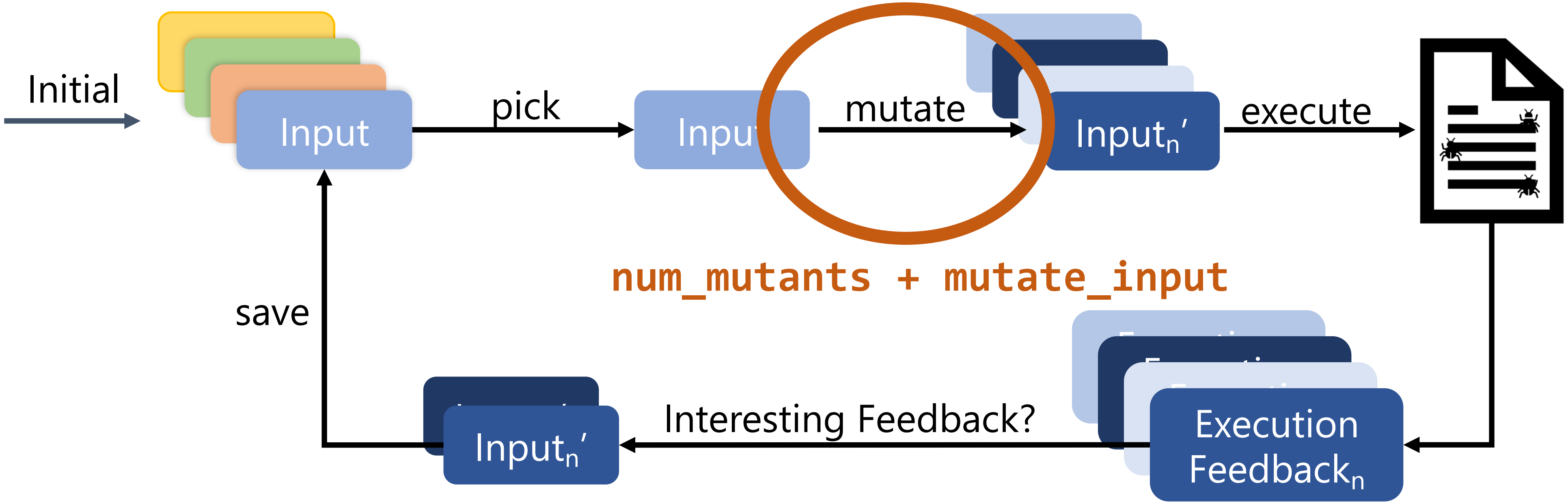
Coverage-Guided Fuzzing

Relation to Assignment



Coverage-Guided Fuzzing

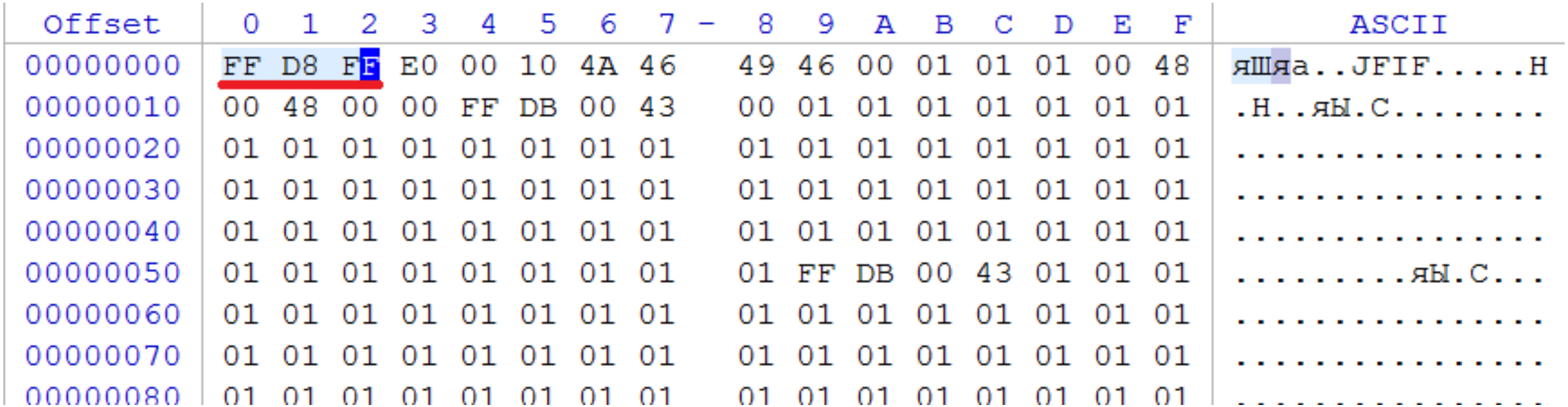
Relation to Assignment



`num_mutants + mutate_input`

“Pulling JPEGs out of Thin Air”

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

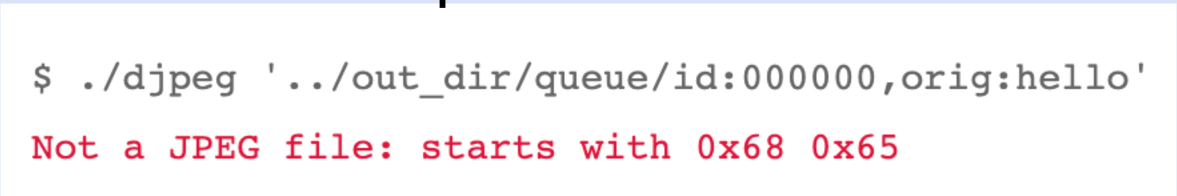


```
 Offset    0  1  2  3  4  5  6  7 -  8  9  A  B  C  D  E  F        ASCII
00000000  FF D8 FF E0 00 10 4A 46    49 46 00 01 01 01 00 48  яШяа..JFIF.....H
00000010  00 48 00 00 FF DB 00 43    00 01 01 01 01 01 01 01  .H..яЫ.C........
00000020  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
00000030  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
00000040  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
00000050  01 01 01 01 01 01 01 01    01 FF DB 00 43 01 01 01  .........яЫ.C...
00000060  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
00000070  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
00000080  01 01 01 01 01 01 01 01    01 01 01 01 01 01 01 01  ................
```

“Pulling JPEGs out of Thin Air”

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

Input: “hello”

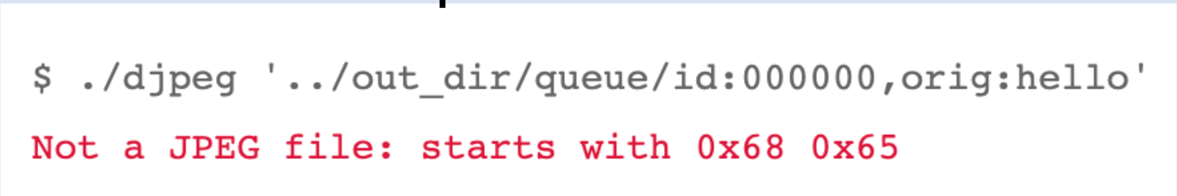


```
$ ./djpeg '../out_dir/queue/id:000000,orig:hello'
Not a JPEG file: starts with 0x68 0x65
```

“Pulling JPEGs out of Thin Air”

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

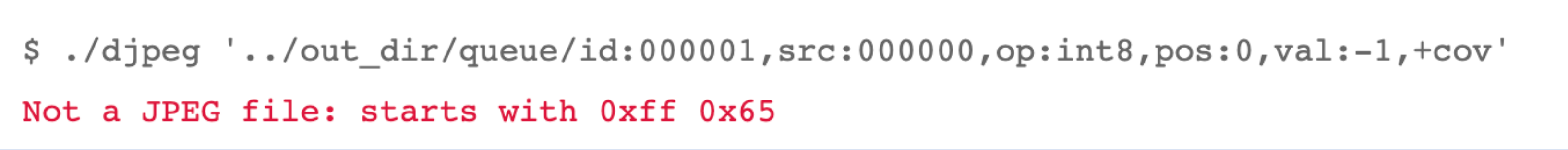
Input: “`hello`”



```
$ ./djpeg '../out_dir/queue/id:000000,orig:hello'
Not a JPEG file: starts with 0x68 0x65
```

Many mutations later

Input: “*`0xff`*`ello`”

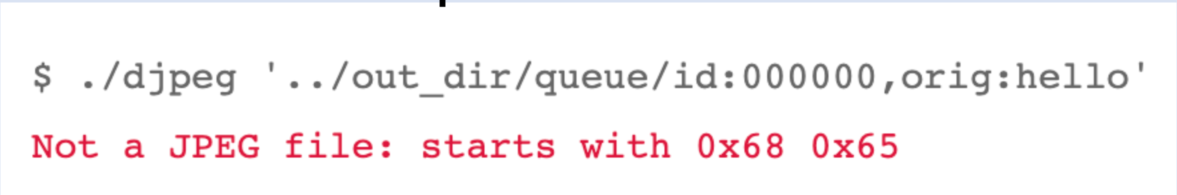


```
$ ./djpeg '../out_dir/queue/id:000001,src:000000,op:int8,pos:0,val:-1,+cov'
Not a JPEG file: starts with 0xff 0x65
```

“Pulling JPEGs out of Thin Air”

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

Input: “hello”

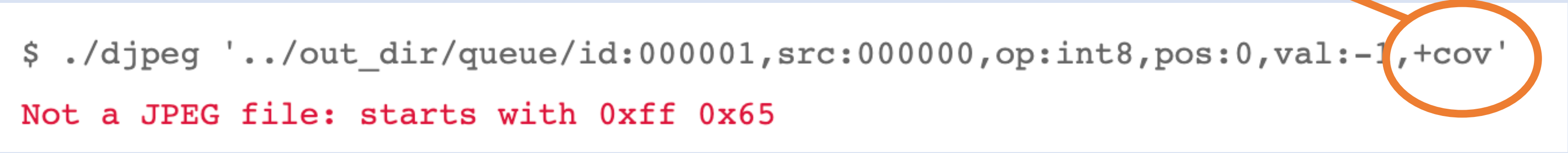


```
$ ./djpeg '../out_dir/queue/id:000000,orig:hello'
Not a JPEG file: starts with 0x68 0x65
```

Many mutations later

Input: “0xffello”

Covers new edge

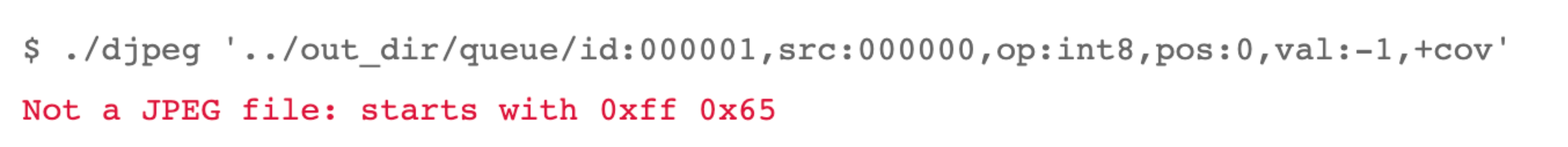


```
$ ./djpeg '../out_dir/queue/id:000001,src:000000,op:int8,pos:0,val:-1,+cov'
Not a JPEG file: starts with 0xff 0x65
```

“Pulling JPEGs out of Thin Air”

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

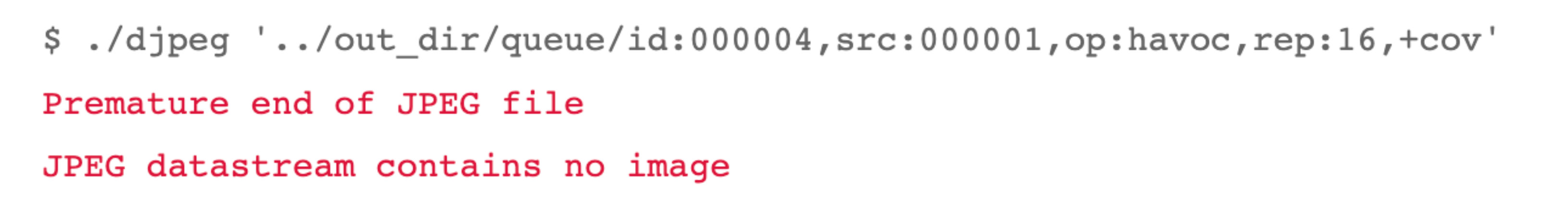
Input: “*0xffello*”



```
$ ./djpeg '../out_dir/queue/id:000001,src:000000,op:int8,pos:0,val:-1,+cov'
Not a JPEG file: starts with 0xff 0x65
```

Many mutations later

Input: “*0xff0xd8llo*”

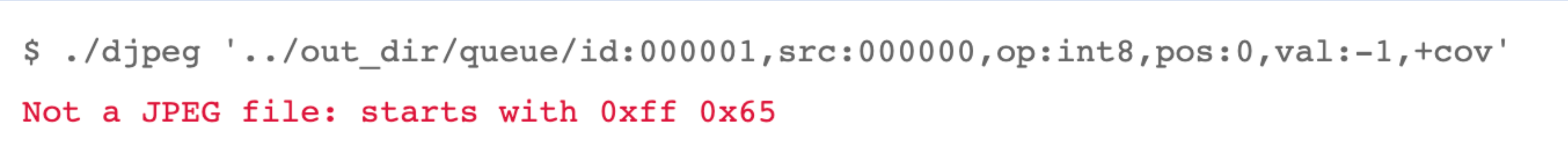


```
$ ./djpeg '../out_dir/queue/id:000004,src:000001,op:havoc,rep:16,+cov'
Premature end of JPEG file
JPEG datastream contains no image
```

"Pulling JPEGs out of Thin Air"

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

Input: "*0xffello*"

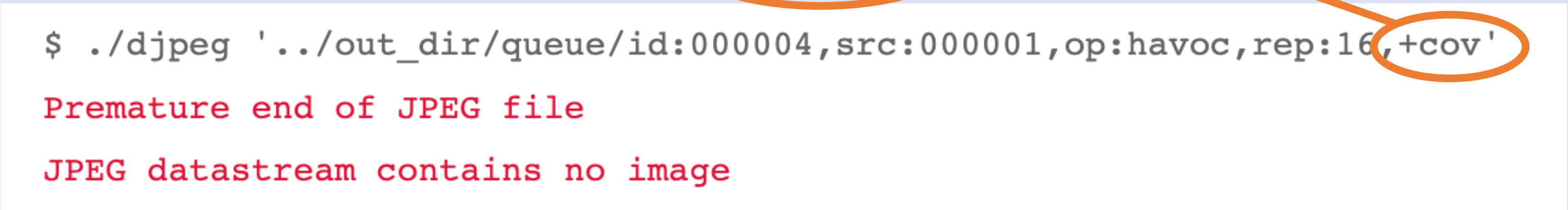


```
$ ./djpeg '../out_dir/queue/id:000001,src:000000,op:int8,pos:0,val:-1,+cov'
Not a JPEG file: starts with 0xff 0x65
```

Many mutations later

Input: "*0xff0xd8llo*"

Covers new edge

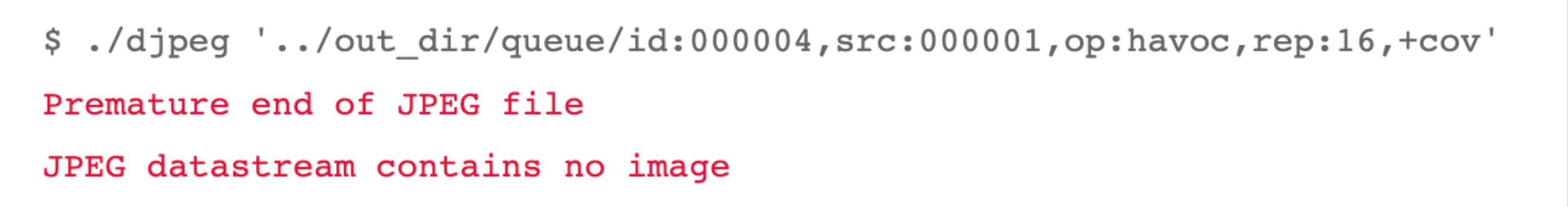


```
$ ./djpeg '../out_dir/queue/id:000004,src:000001,op:havoc,rep:16,+cov'
Premature end of JPEG file
JPEG datastream contains no image
```

“Pulling JPEGs out of Thin Air”

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

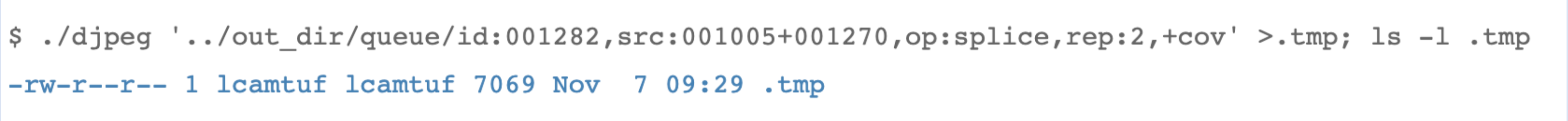
Input: “*0xff0xd8llo*”



```
$ ./djpeg '../out_dir/queue/id:000004,src:000001,op:havoc,rep:16,+cov'
Premature end of JPEG file
JPEG datastream contains no image
```

6 hours of mutations + saving later…

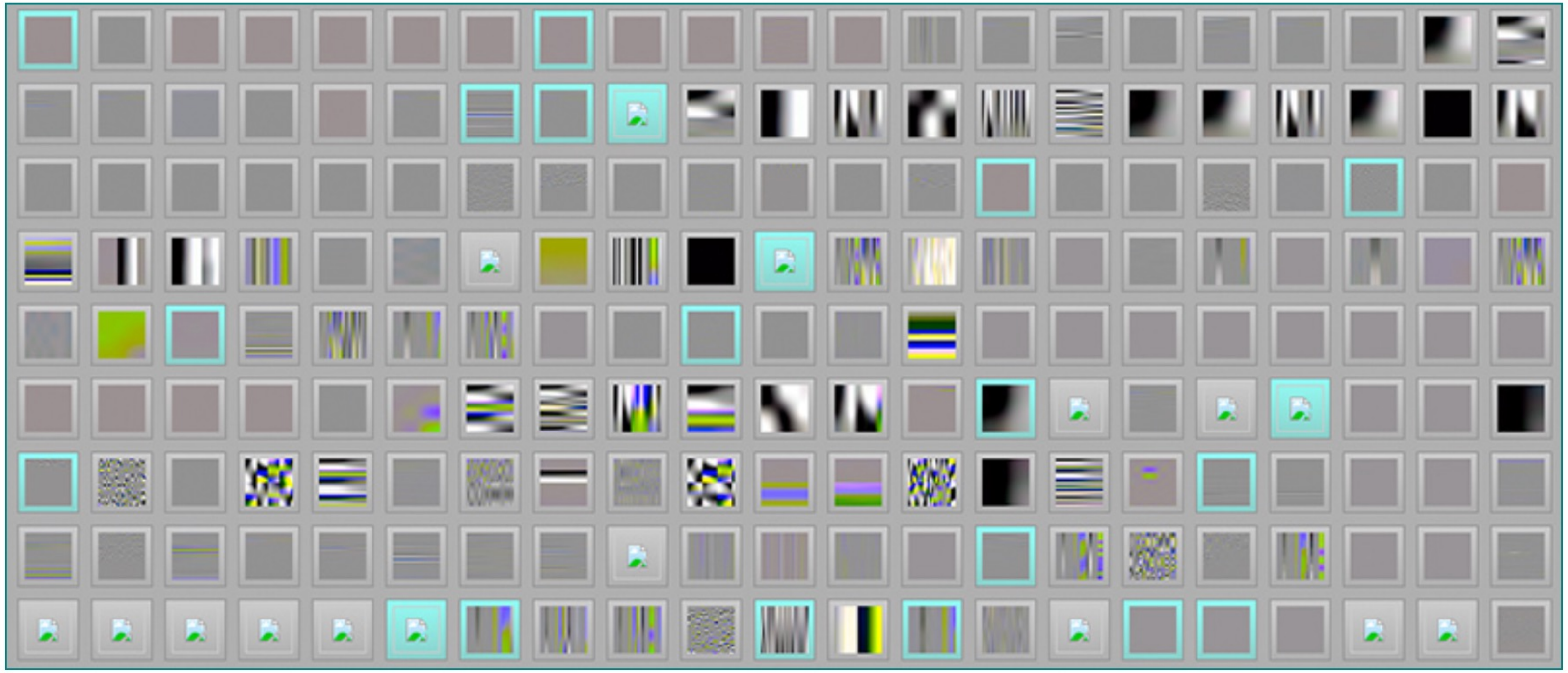
Input: a blank JPEG 3 pixels wide, 786 pixels tall



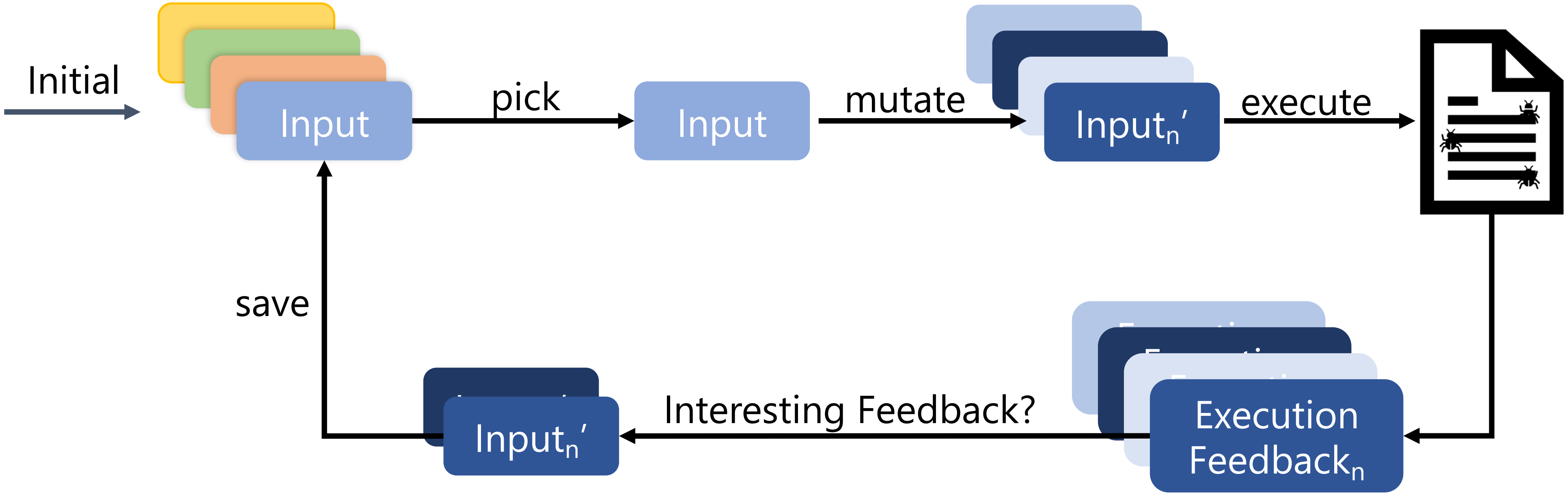
```
$ ./djpeg '../out_dir/queue/id:001282,src:001005+001270,op:splice,rep:2,+cov' >.tmp; ls -l .tmp
-rw-r--r-- 1 lcamtuf lcamtuf 7069 Nov  7 09:29 .tmp
```

“Pulling JPEGs out of Thin Air”

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



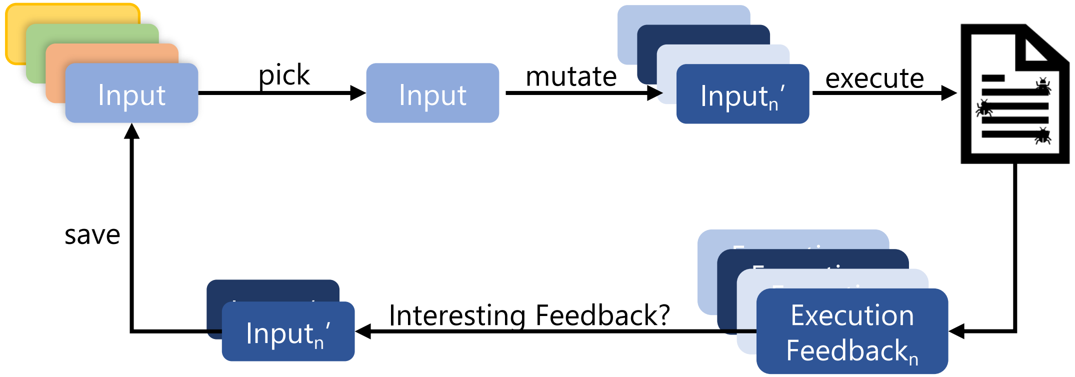
Pros/Cons of Coverage-Guided Fuzzing



Schedule for Today

- Improving upon pure random fuzzing
- Coverage-guided fuzzing
  - a.k.a. greybox fuzzing, a.k.a. coverage-based greybox fuzzing
- Relation to Evolutionary Algorithms

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

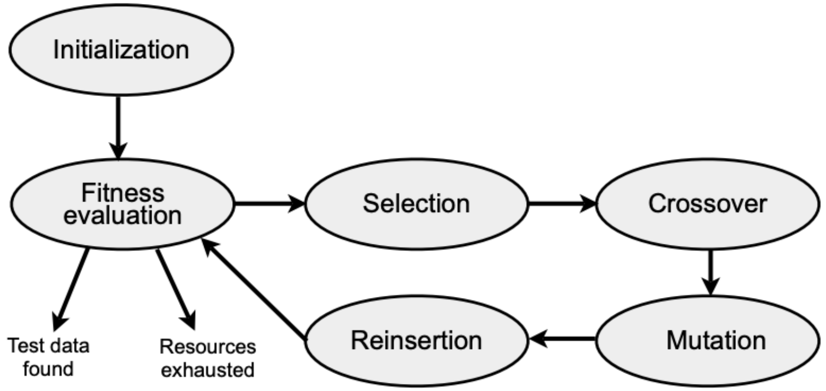
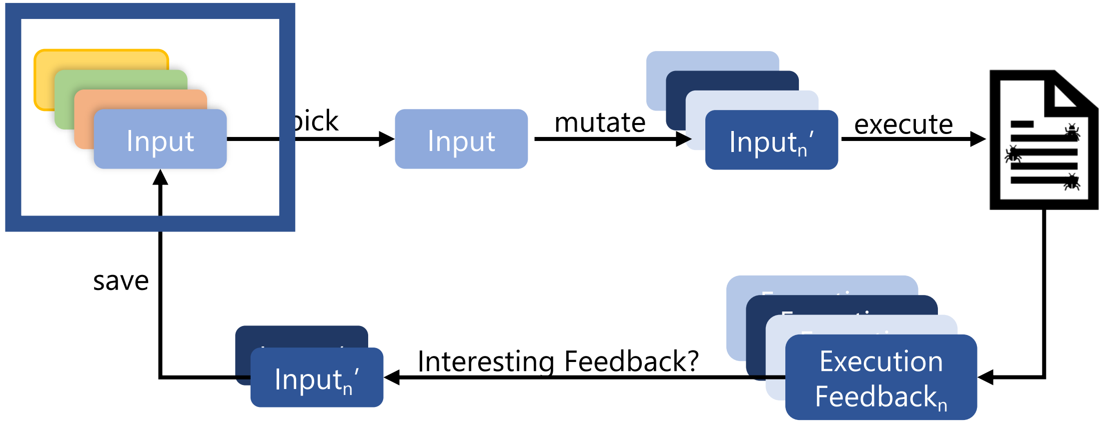


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. *"Search-Based Software Testing: Past, Present and Future"*

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

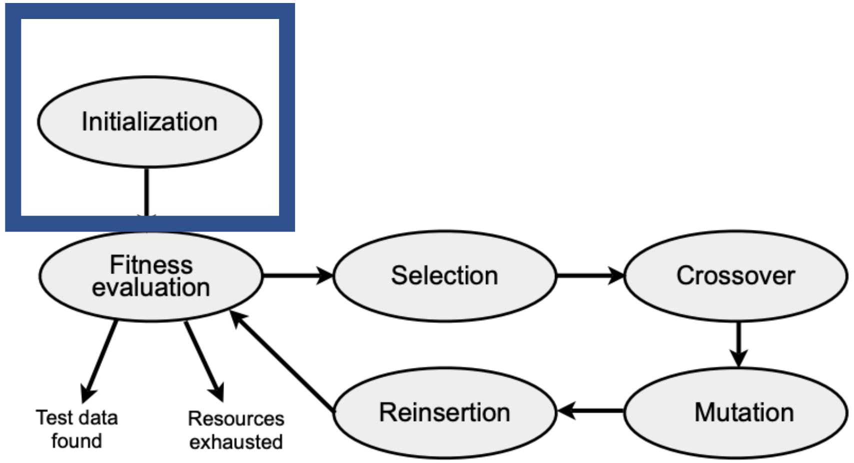
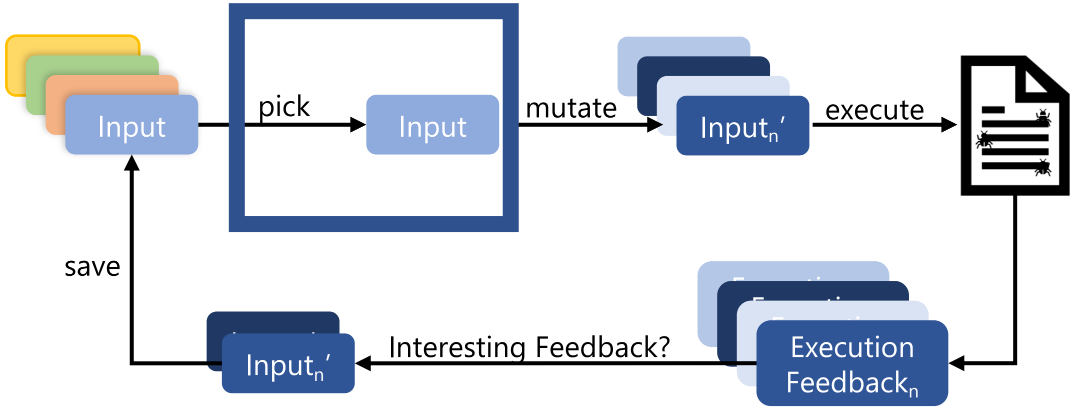


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. "*Search-Based Software Testing: Past, Present and Future*"

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

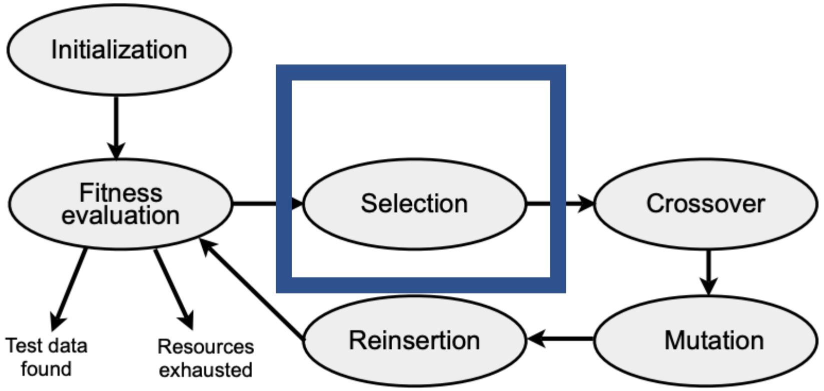
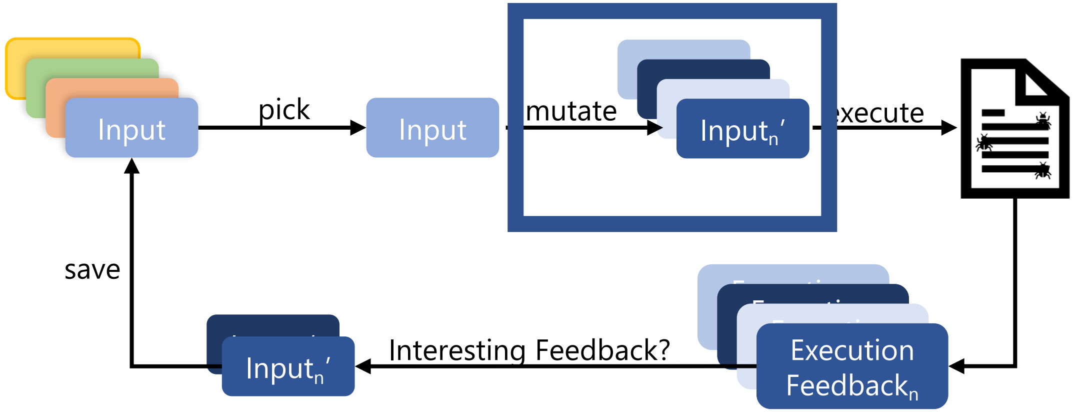


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. "*Search-Based Software Testing: Past, Present and Future*"

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

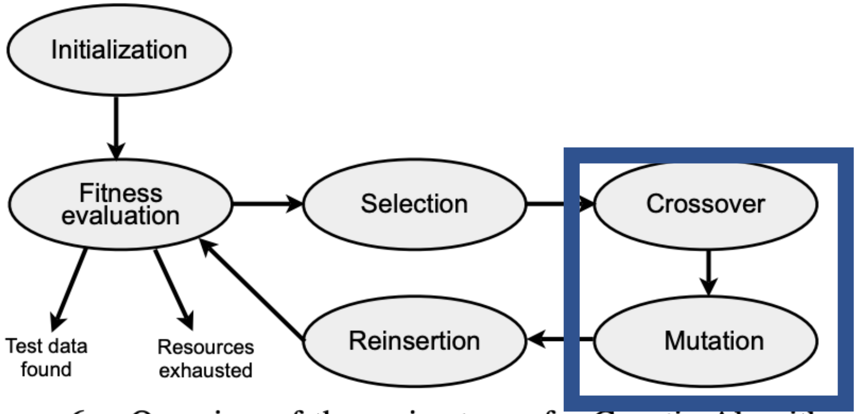
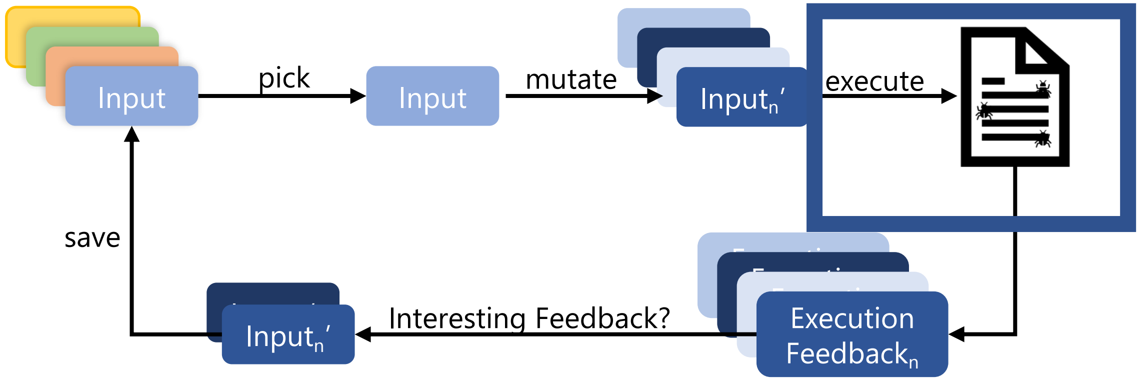


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. "*Search-Based Software Testing: Past, Present and Future*"

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

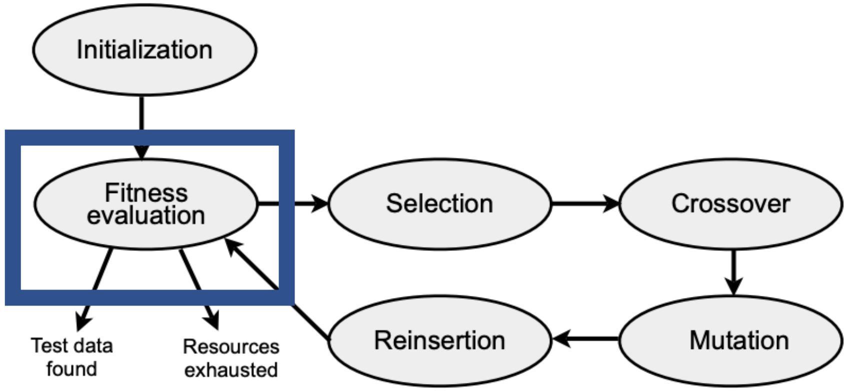
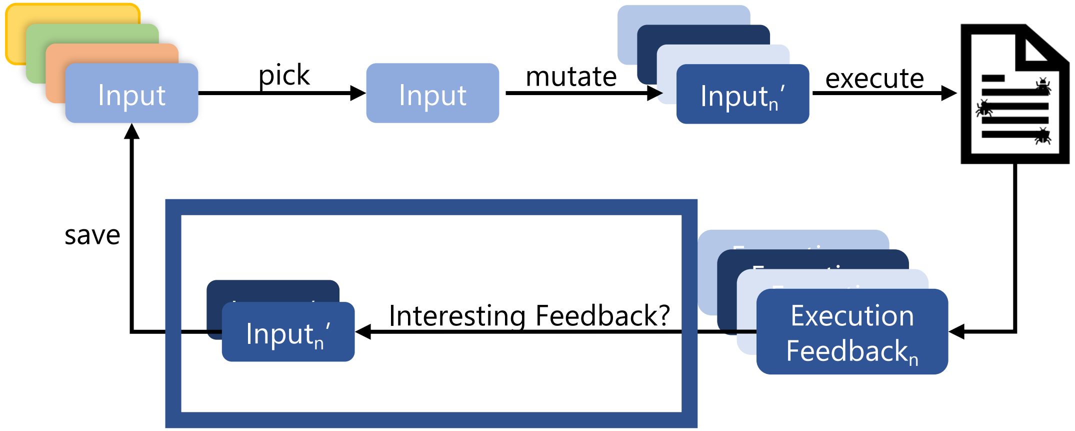


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. "*Search-Based Software Testing: Past, Present and Future*"

Coverage-guided Fuzzing vs. Evolutionary Algorithms



Circa 2014

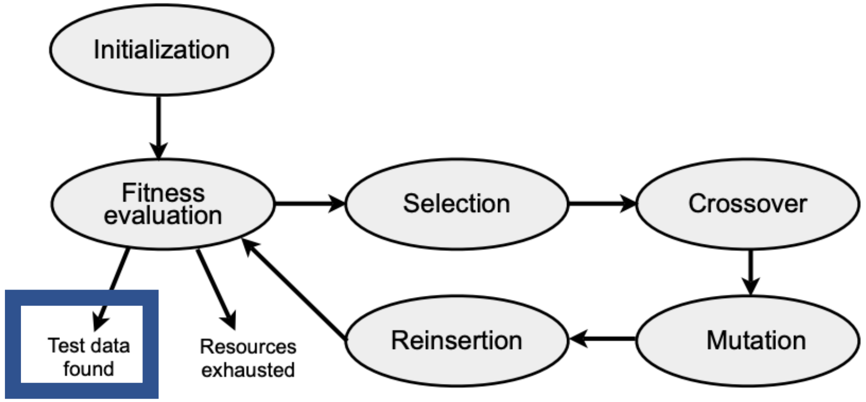


Figure 6. Overview of the main steps of a Genetic Algorithm

Circa 2011:

Phil McMinn. "*Search-Based Software Testing: Past, Present and Future*"

Evolutionary Algorithms

- In traditional genetic algorithms, fitness is a number
- Higher fitness == better
- Fitness of an input does not change over time

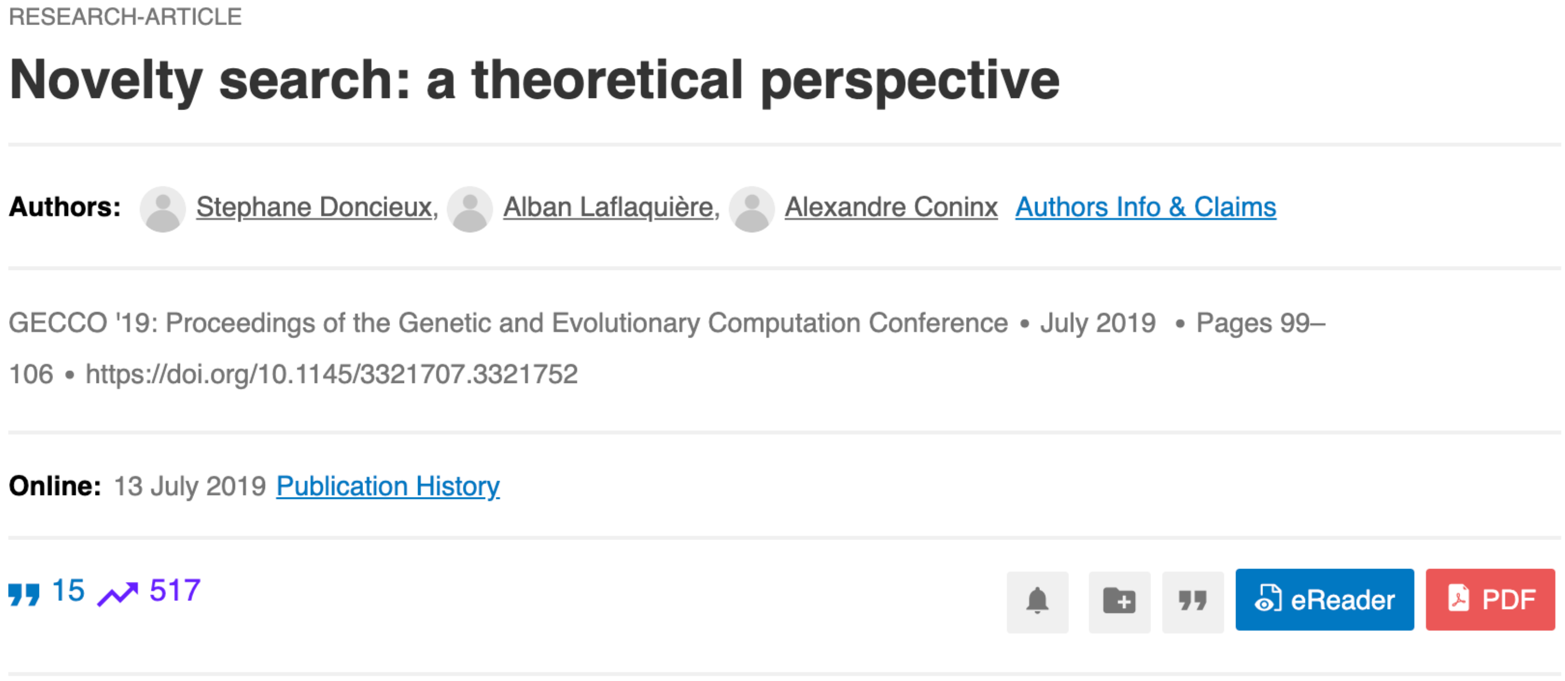
We will study in this class one use of evolutionary algorithms for test suite generation

Coverage-Guided Fuzzing

- Choose inputs to save if they increase coverage
- New coverage == better
- An input is not interesting if it is re-discovered

No constantly increasing fitness… more akin to "novelty search"

Novelty Search



RESEARCH-ARTICLE

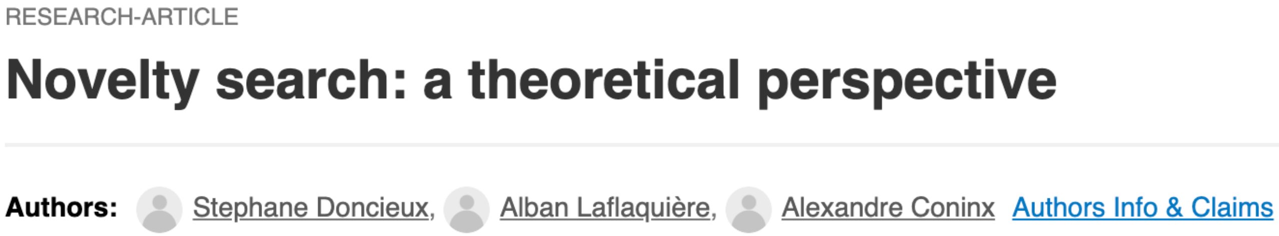
Novelty search: a theoretical perspective

Authors: Stephane Doncieux, Alban Laflaquière, Alexandre Coninx Authors Info & Claims

GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference • July 2019 • Pages 99–106 • https://doi.org/10.1145/3321707.3321752

Online: 13 July 2019 Publication History

Novelty Search



RESEARCH-ARTICLE

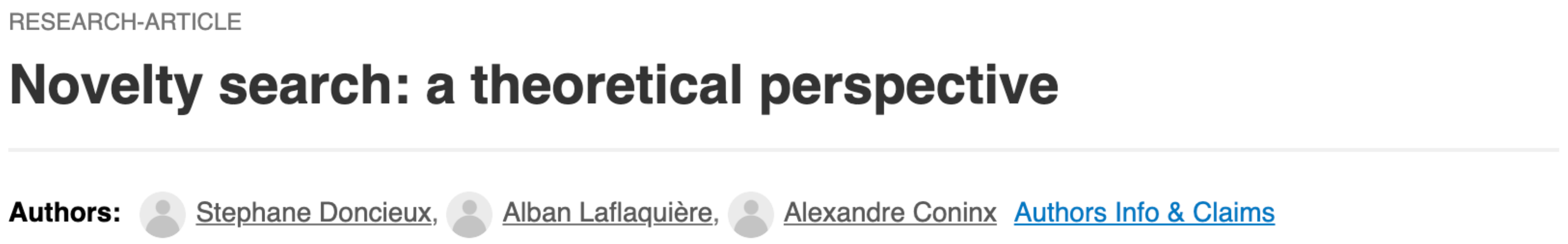
Novelty search: a theoretical perspective

Authors: Stephane Doncieux, Alban Laflaquière, Alexandre Coninx Authors Info & Claims

Abstract

Novelty Search is an exploration algorithm driven by the novelty of a behavior. The same individual evaluated at different generations has different fitness values. […] We assert that Novelty Search asymptotically behaves like a *uniform random search process in the behavior space.* […]

Novelty Search



RESEARCH-ARTICLE

Novelty search: a theoretical perspective

Authors: Stephane Doncieux, Alban Laflaquière, Alexandre Coninx Authors Info & Claims

Abstract

Novelty Search is an exploration algorithm driven by the novelty of a behavior. The same individual evaluated at different generations has different fitness values. […] We assert that Novelty Search asymptotically behaves like a *uniform random search process in the behavior space.* […]

(does this also hold for coverage-guided fuzzing? unknown)