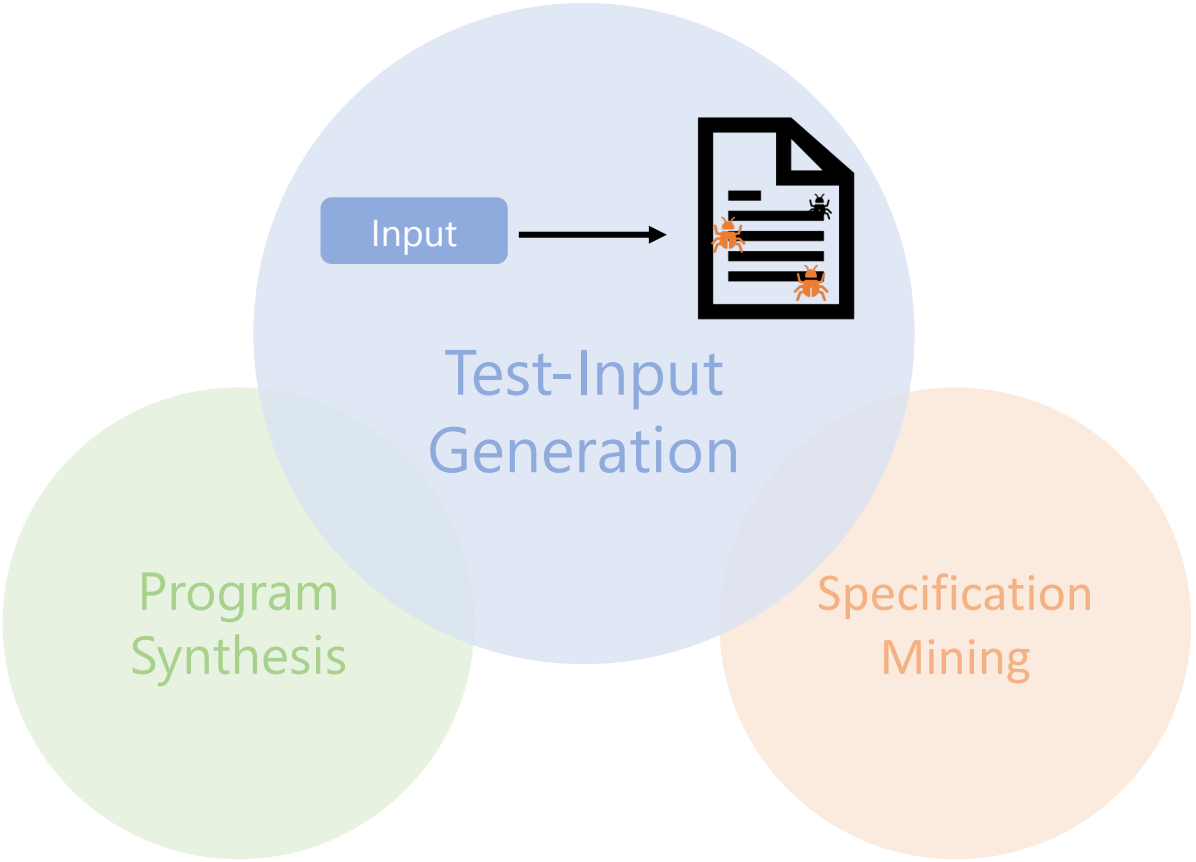
Expanding the Reach of Fuzzing

Caroline Lemieux
University of British Columbia
(currently postdoc @ Microsoft Research NYC)

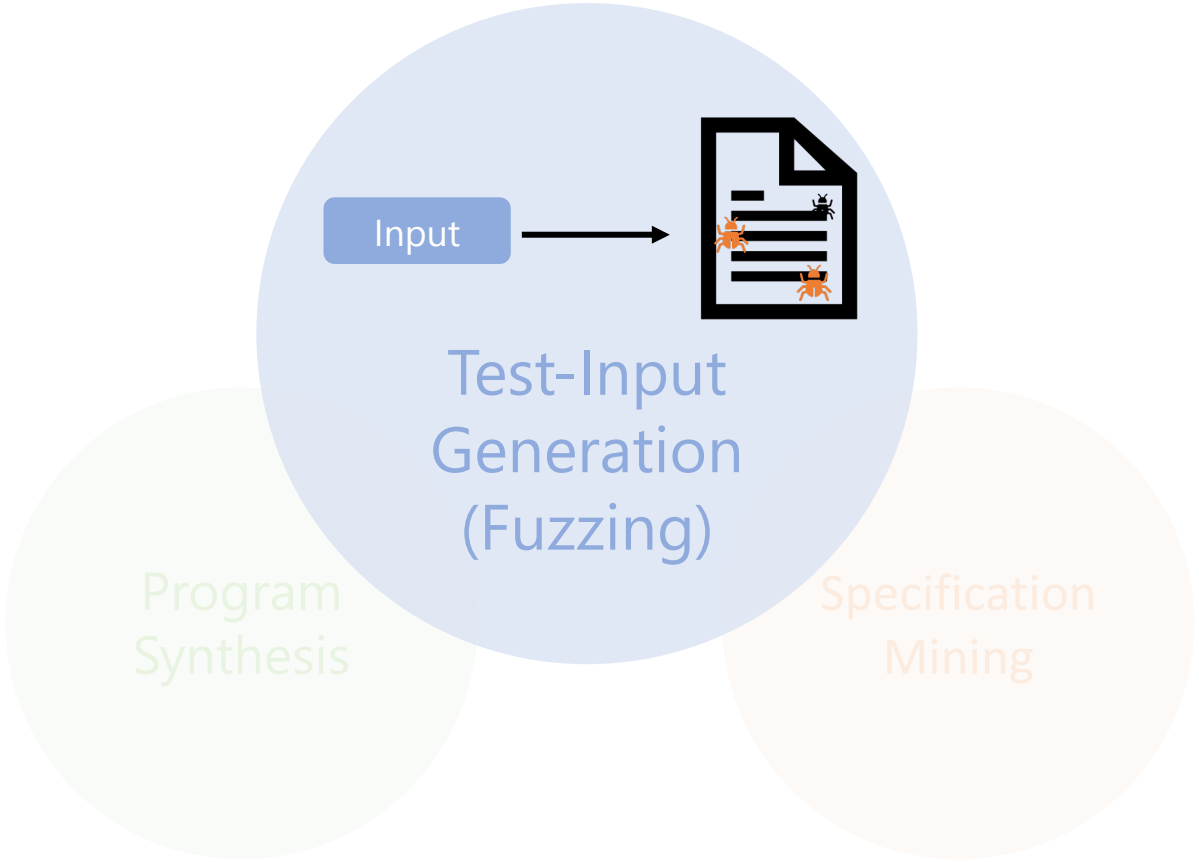
Talk at CSER

Nov 21st, 2021

My Work


Input
Test-Input Generation
Program Synthesis
Specification Mining

My Work


Input
Test-Input Generation (Fuzzing)
Program Synthesis
Specification Mining

Background on Fuzzing

Performance Bugs 

Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions

1990s: Random Fuzzing

When we use basic operating system facilities, such as the kernel and major utility programs, we expect a high degree of reliability. These parts of the system are used frequently and this frequent use implies that the programs are well-tested and working correctly. To make a systematic statement about the correctness of a program, we should probably use some form of formal verification. While the technology for program verification is advancing, it has not yet reached the point where it is easy to apply (or commonly applied) to large systems.

A recent experience led us to believe that, while formal verification of a complete set of operating system utilities was too onerous a task, there was still a need for some form of more complete testing: On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash. These programs included a significant number of basic operating system utilities. It is reasonable to expect that basic utilities should not crash ("core dump"); on receiving unusual input, they might exit with minimal error messages, but they should not crash. This experience led us to believe that there might be serious bugs lurking in the systems that we regularly used.

This scenario motivated a systematic test of the utility programs running on various versions of the Unix operating system. The project proceeded in four steps: (1) programs were constructed to generate random characters, and to help test interactive utilities; (2) these programs were used to test a large number of utilities on random input strings to see if they crashed; (3) the strings (or types of strings) that crash these programs were identified; and (4) the causes of the program crashes were identified and the common mistakes that cause these crashes were categorized. As a result of testing almost 90 different utility programs on seven versions of Unix™, we were able to crash more than 24% of these programs. Our testing included versions of Unix that underwent commercial product testing. A byproduct of this project is a list of bug reports (and fixes) for the crashed programs and a set of tools available to the systems community.

There is a rich body of research on program testing and verification. Our approach is not a substitute for a formal verification or testing procedures, but rather an inexpensive mechanism to identify bugs and increase overall system reliability. We are using a coarse notion of correctness in our study. A program is detected as faulty only if it crashes or hangs (loops indefinitely). Our goal is to complement, not replace, existing test procedures.

This type of study is important for several reasons: First, it contributes to the testing community a large list of real bugs. These bugs can provide test cases against which researchers can evaluate more sophisticated testing and verification strategies. Second, one of the bugs that we found was caused by the same programming practice that provided one of the security holes to the Internet worm (the "gets finger" bug) [2,3] We have found additional bugs that might indicate future security holes. Third, some of the crashes were caused by input that might be carelessly typed—some strange and unexpected errors were uncovered by this method of testing. Fourth, we sometimes inadvertently feed programs noisy input (e.g., trying to edit or view an object module). In these cases, we would like some meaningful and predictable response. Fifth, noisy phone lines are a reality, and major utilities (like shells and editors) should not crash because of them. Last, we were interested in the interactions between our random testing and more traditional industrial software testing.

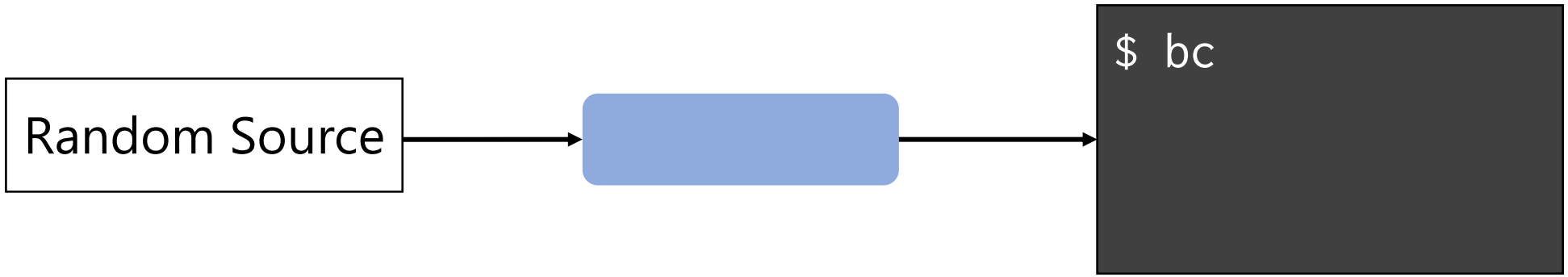
While our testing strategy sounds somewhat naive, its ability to discover fatal program bugs is impressive. If we consider a program to be a complex finite state machine, then our testing strategy can be thought of as a random walk through the state space, searching for undefined states. Similar techniques have been used in areas such as network protocols and CPU cache testing. When testing network protocols, a module can be inserted in the data stream. This module randomly perturbs the packets (either destroying them or modifying them) to test the protocol's error detection and recovery features. Random testing has been used in evaluating complex hardware, such as multiprocessor cache coherence protocols [4]. The state space of the device, when combined with the memory architecture, is large enough that it is difficult to generate systematic tests. In the multiprocessor example, random generation of test cases helped cover a large part of the state space and simplify the generation of cases.

Unix is a trademark of AT&T Bell Laboratories.

Barton P. Miller, Lars Fredriksen and Bryan So

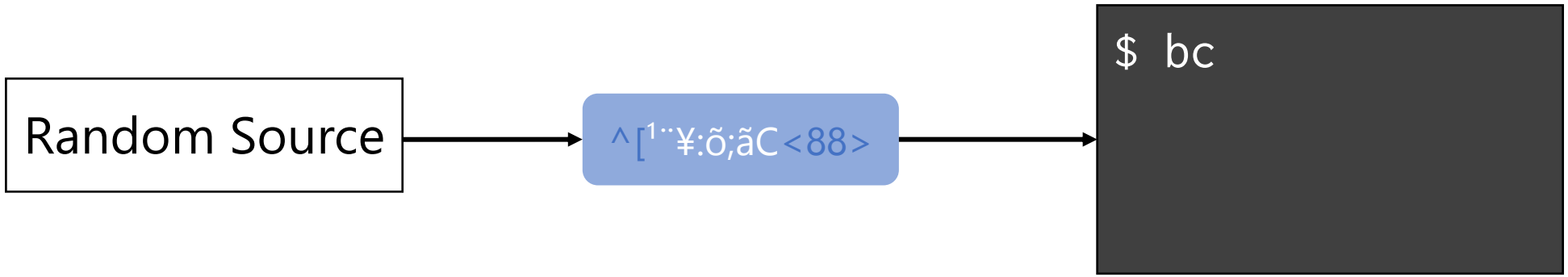
An Empirical Study of the Reliability of UNIX Utilities

1990s: Random Fuzzing



B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

1990s: Random Fuzzing

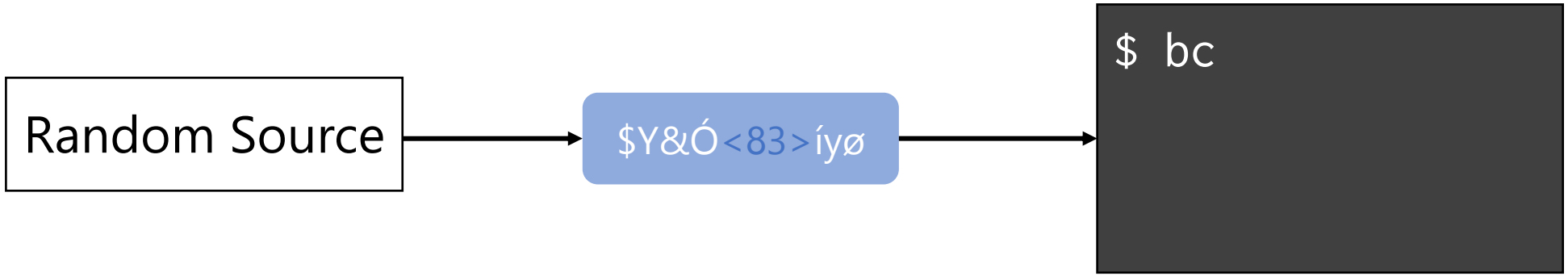


B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

1990s: Random Fuzzing

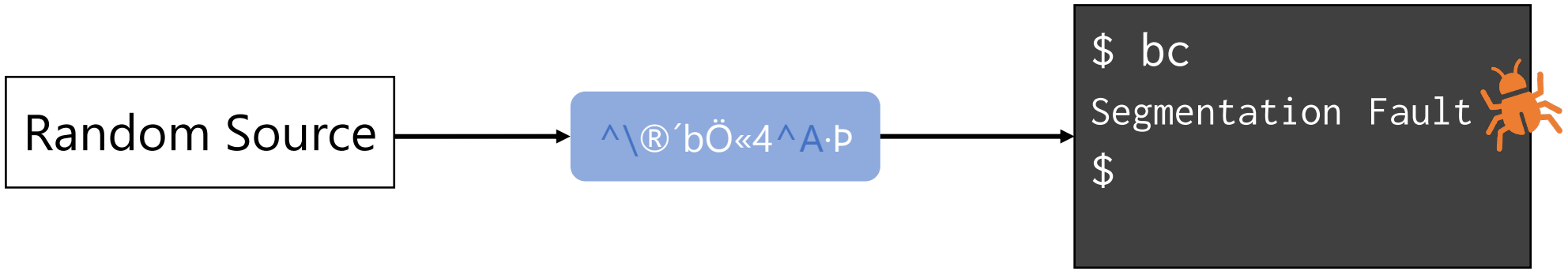
B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

1990s: Random Fuzzing



B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

1990s: Random Fuzzing

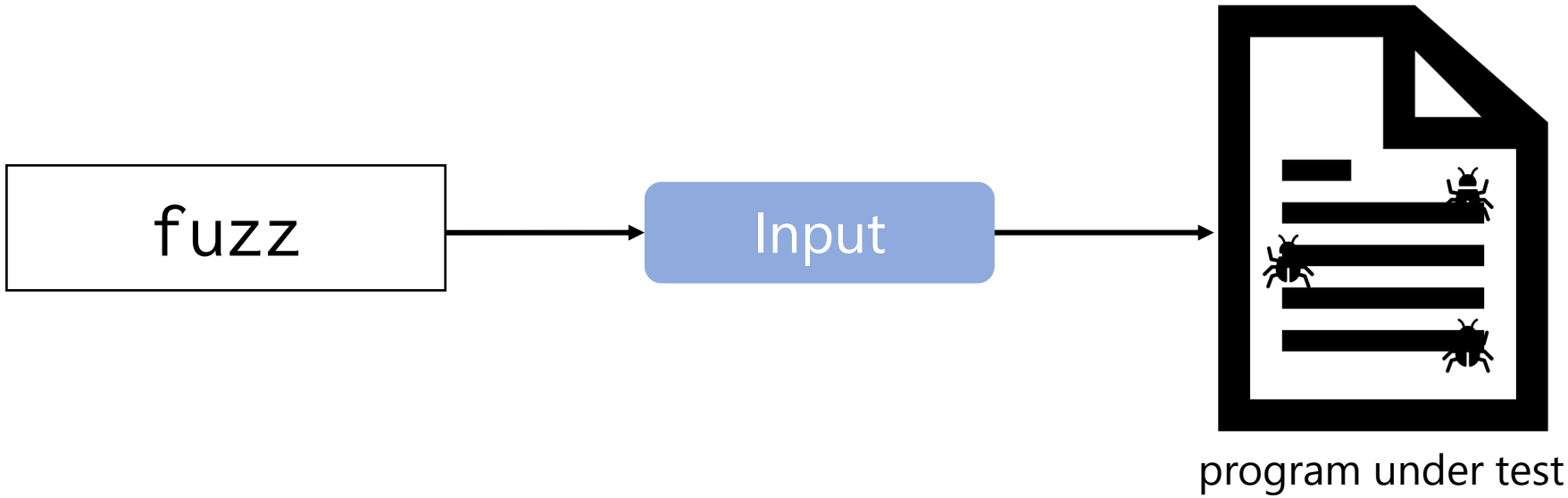


B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

1990s: Random Fuzzing

B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

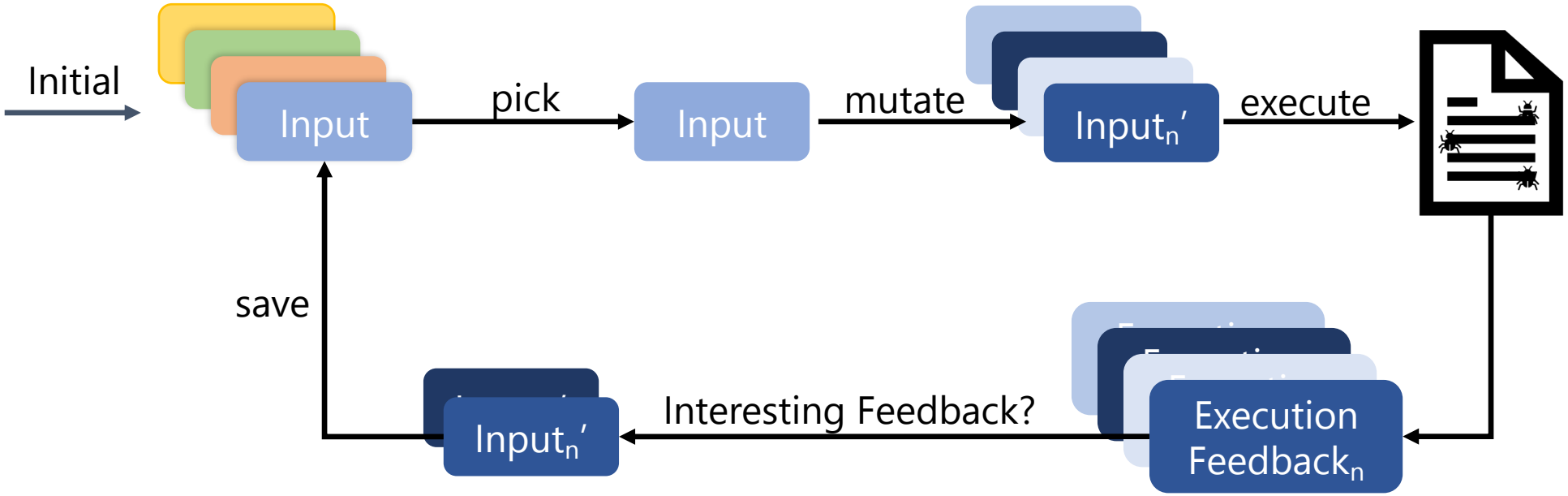
1990s: Random Fuzzing



B. Miller, L. Fredriksen, B. So. *An Empirical Study of the Reliability of Unix Utilities.* Communications of the ACM, 1990.

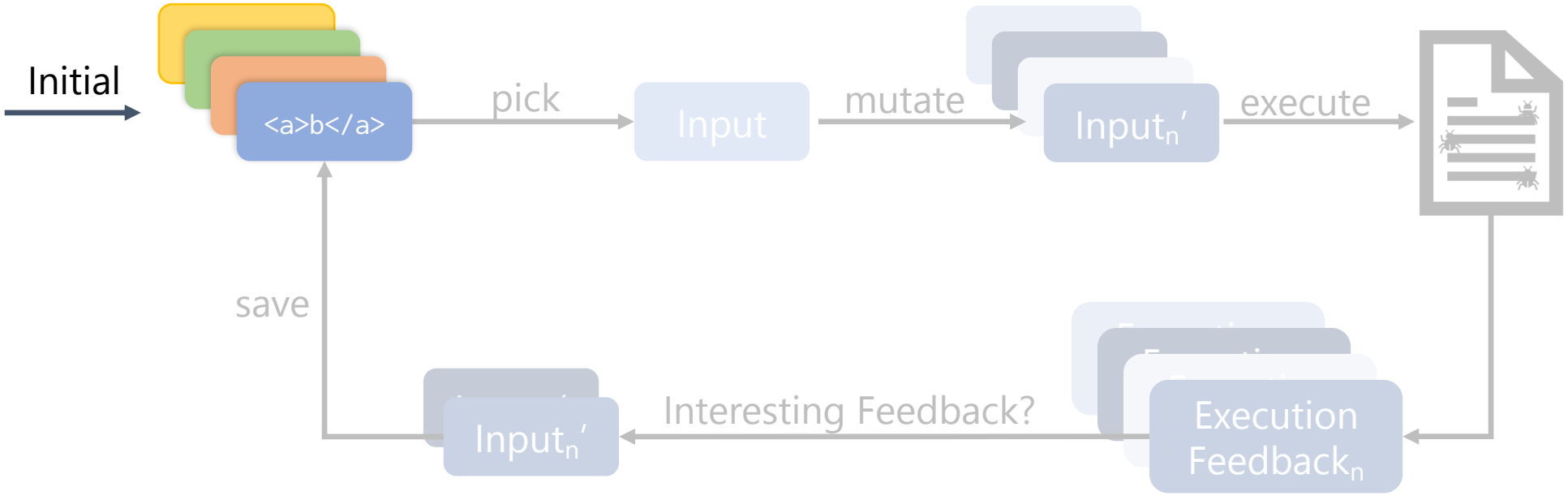
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



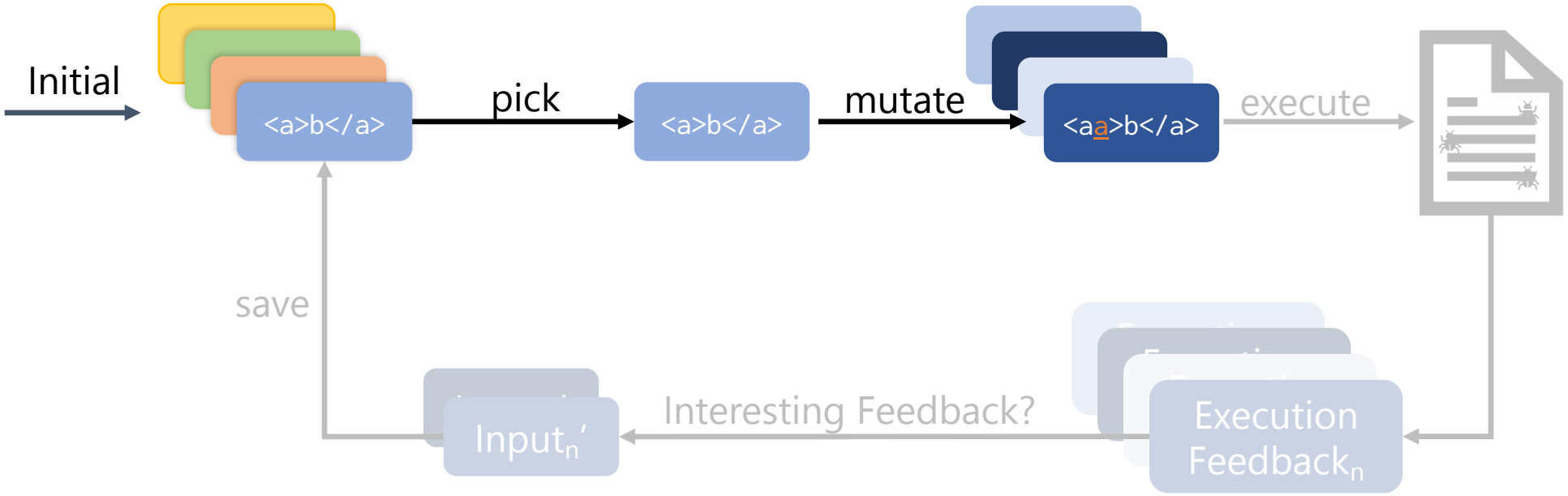
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



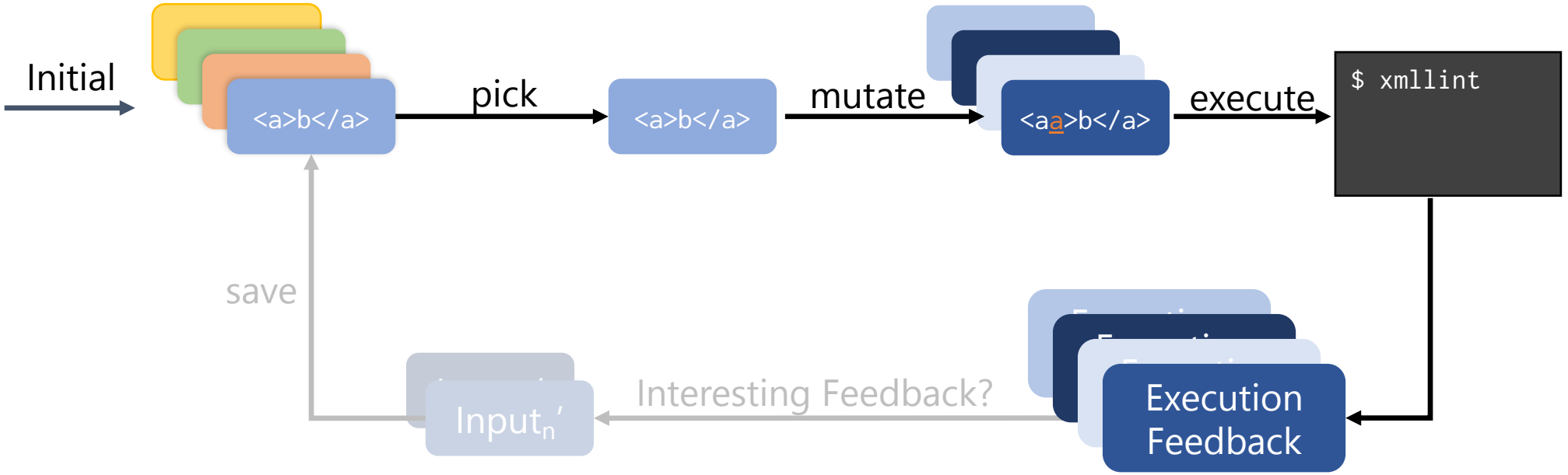
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



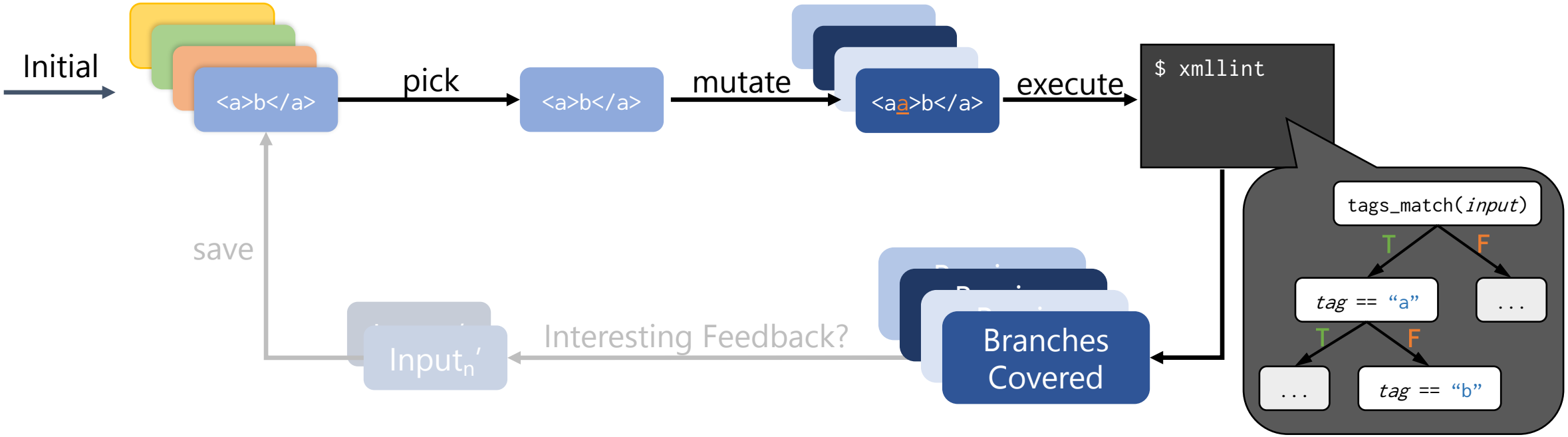
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



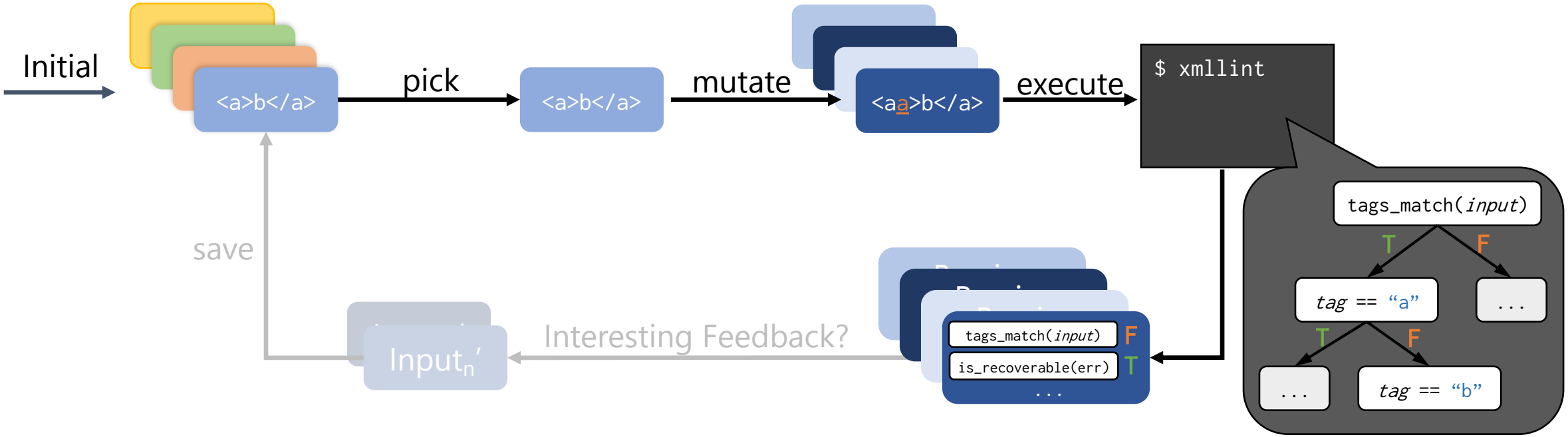
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



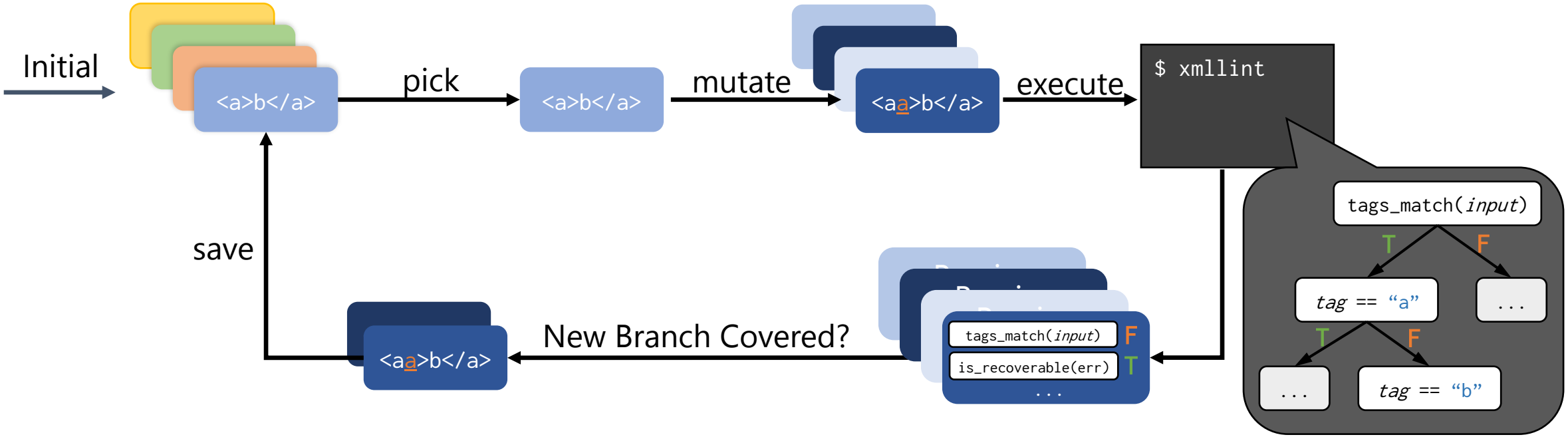
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz



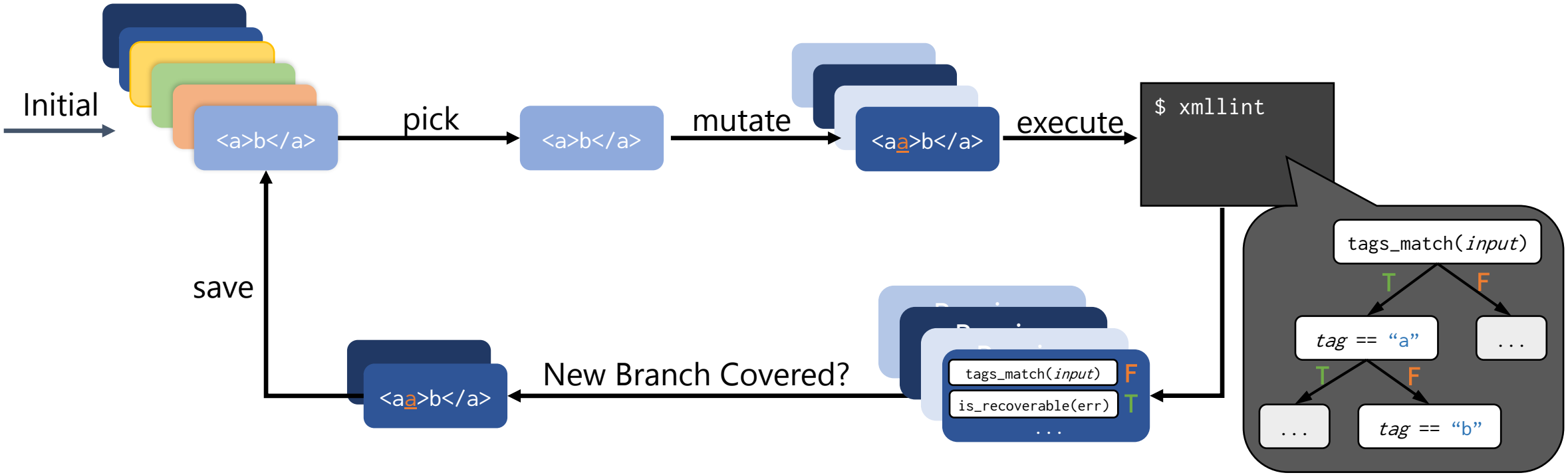
Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz

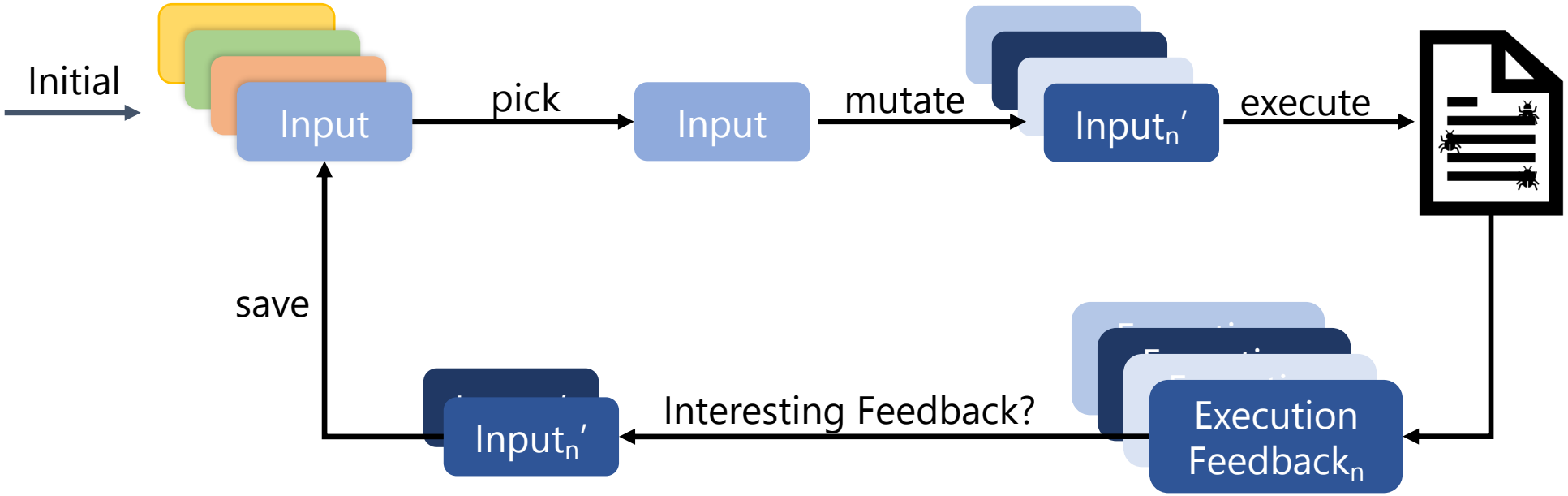


Coverage-Guided Fuzzing

AFL, libFuzzer, honggfuzz

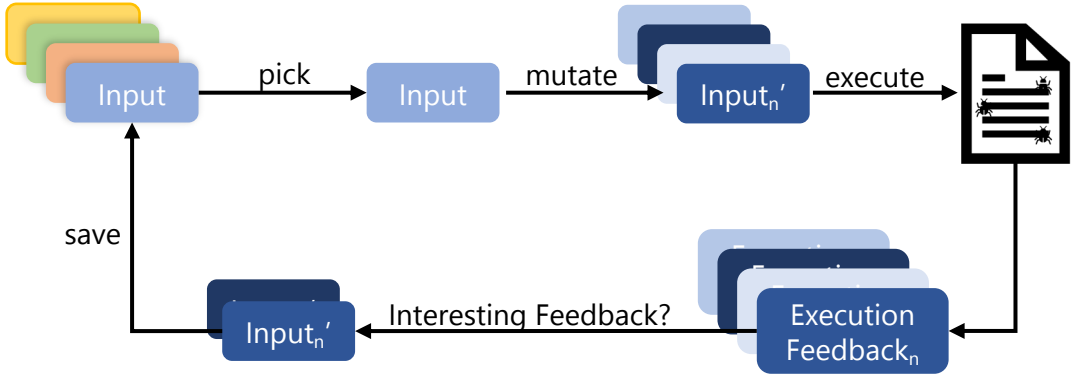


Coverage-Guided Fuzzing



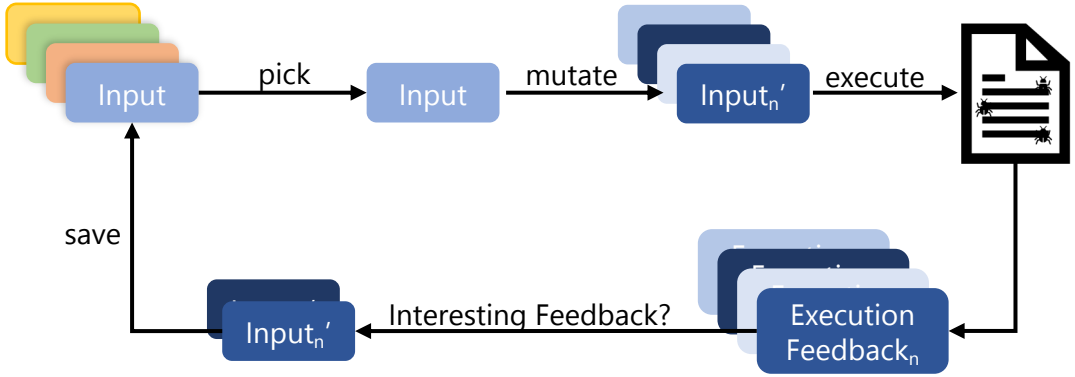
Modern Fuzzing

Coverage-Guided Fuzzing

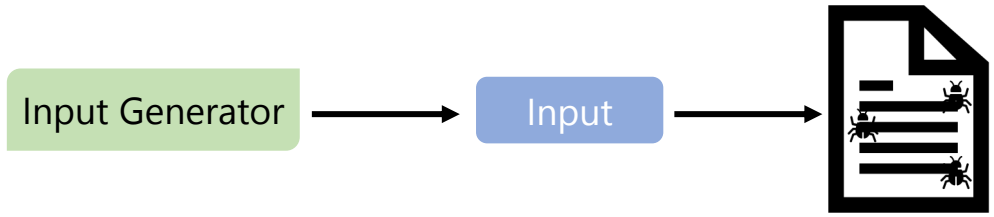


Modern Fuzzing

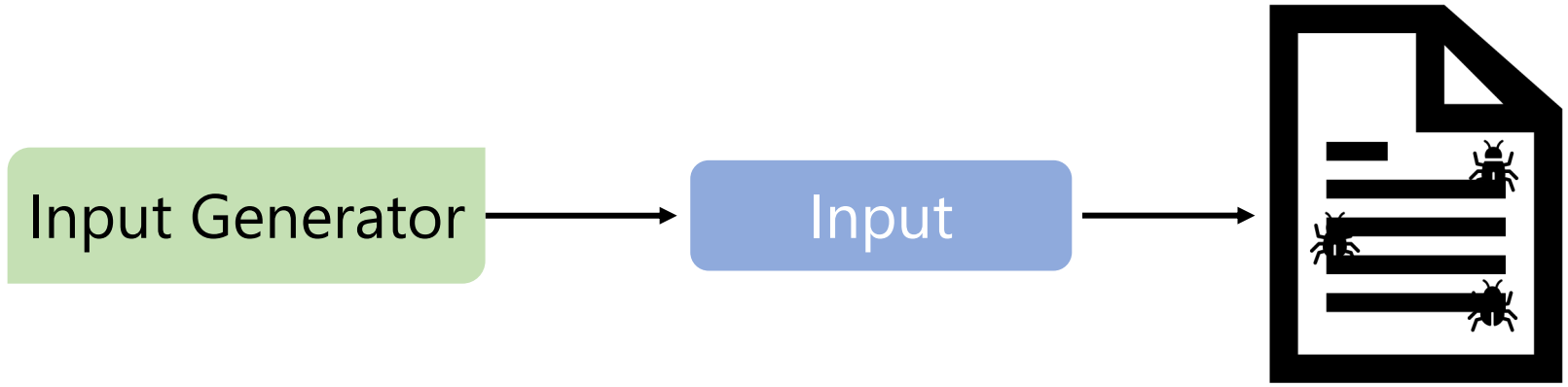
Coverage-Guided Fuzzing



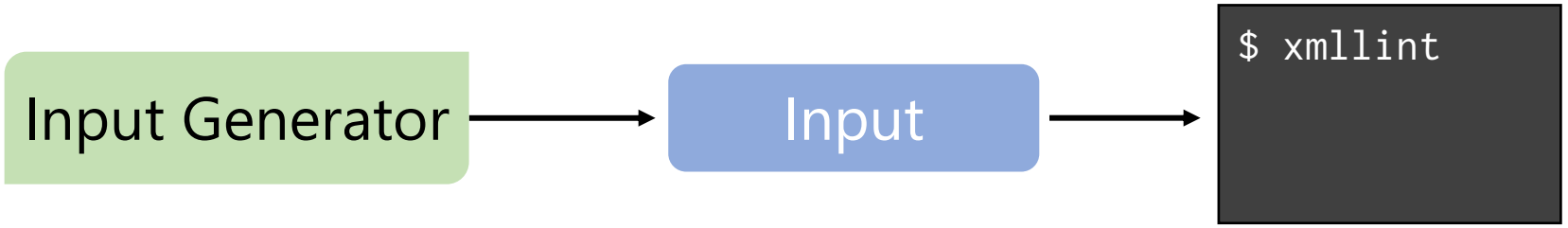
Generator-Based Fuzzing



Generator-Based Fuzzing

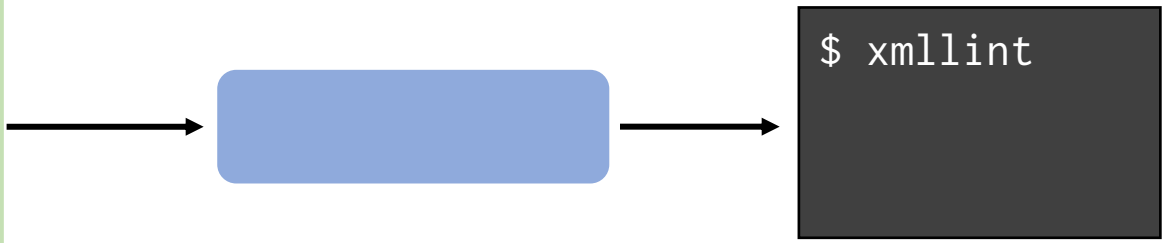


Generator-Based Fuzzing



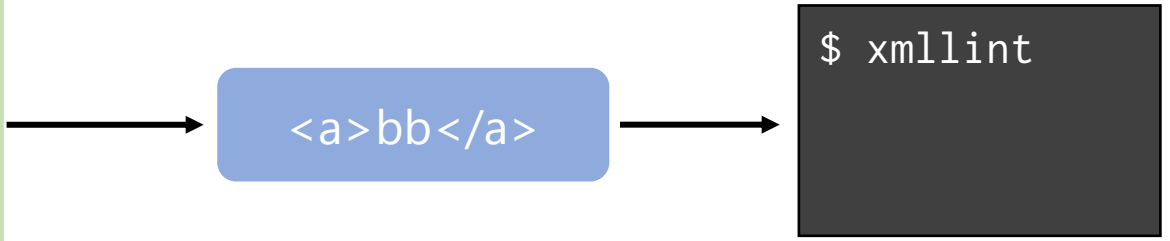
Generator-Based Fuzzing

```
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
```

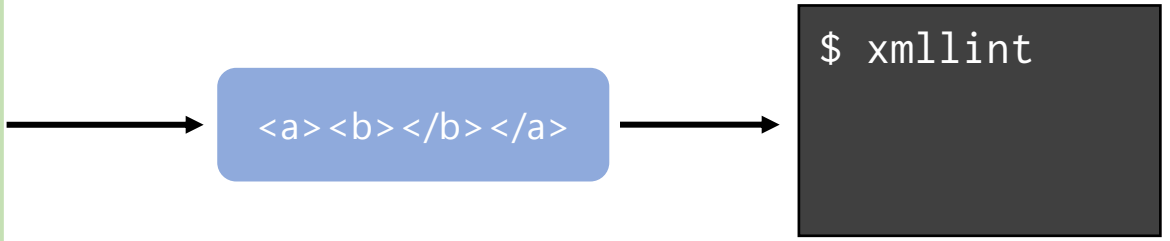


Generator-Based Fuzzing

```
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
```

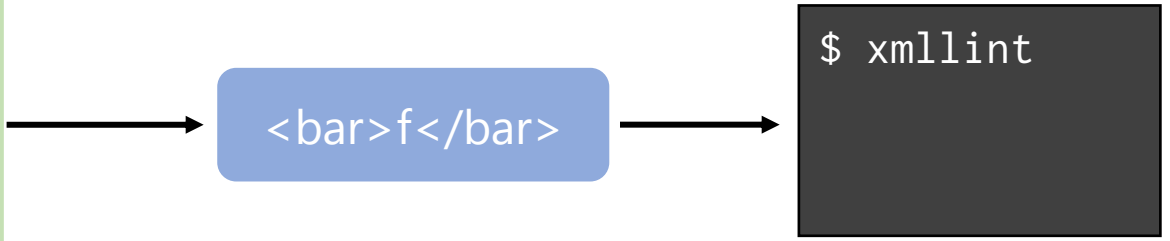
Generator-Based Fuzzing

```
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
```



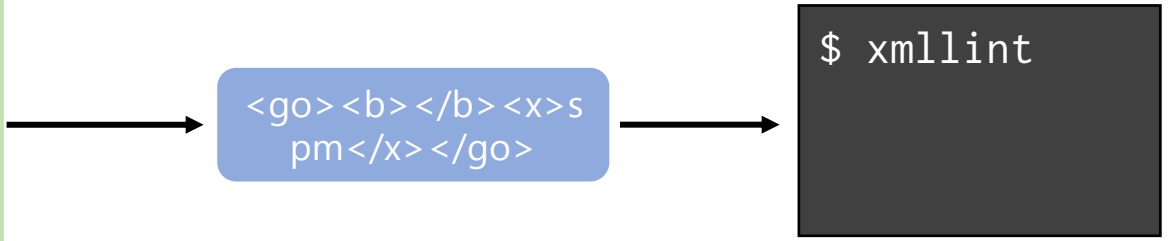
Generator-Based Fuzzing

```
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
```

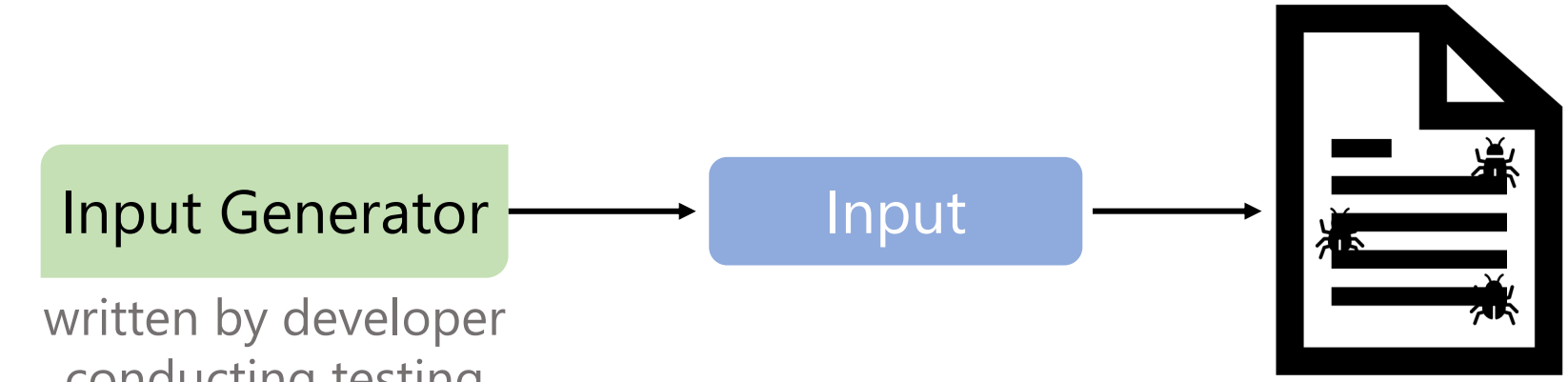


Generator-Based Fuzzing

```
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
```



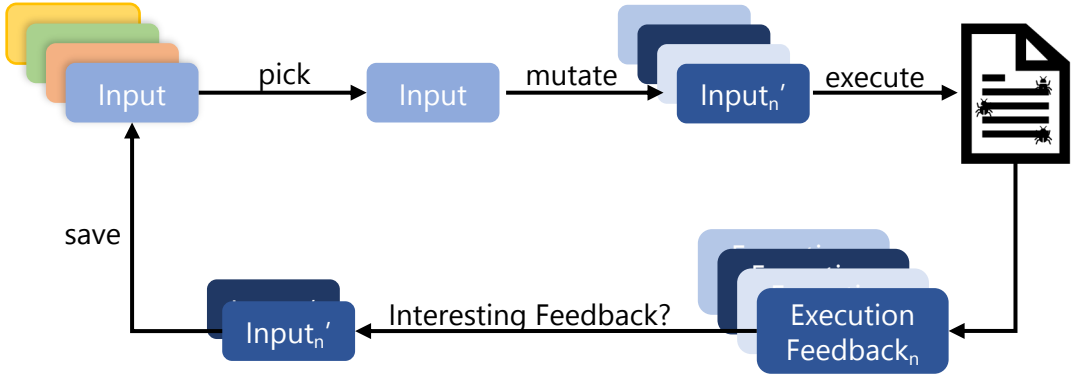
Generator-Based Fuzzing



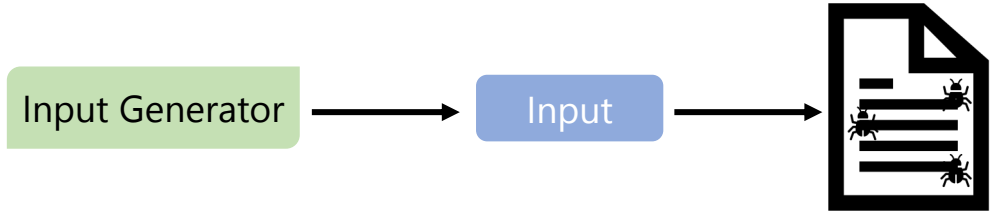
written by developer
conducting testing
(or reuse a suitable one)

Modern Fuzzing

Coverage-Guided Fuzzing

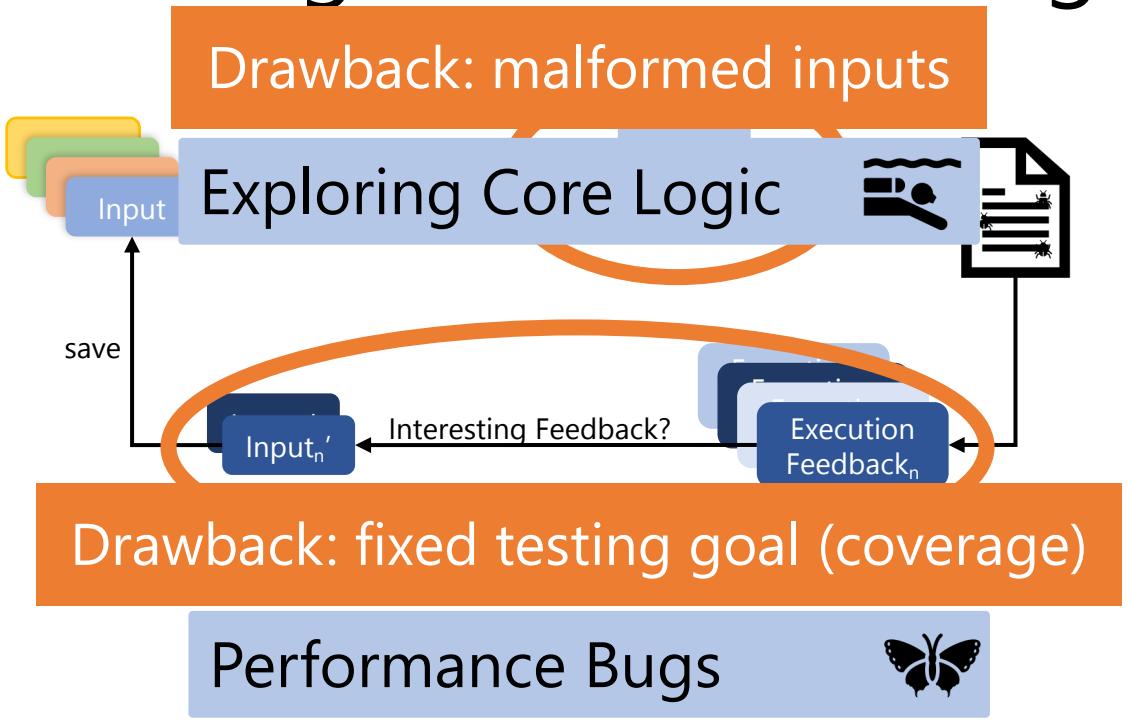


Generator-Based Fuzzing

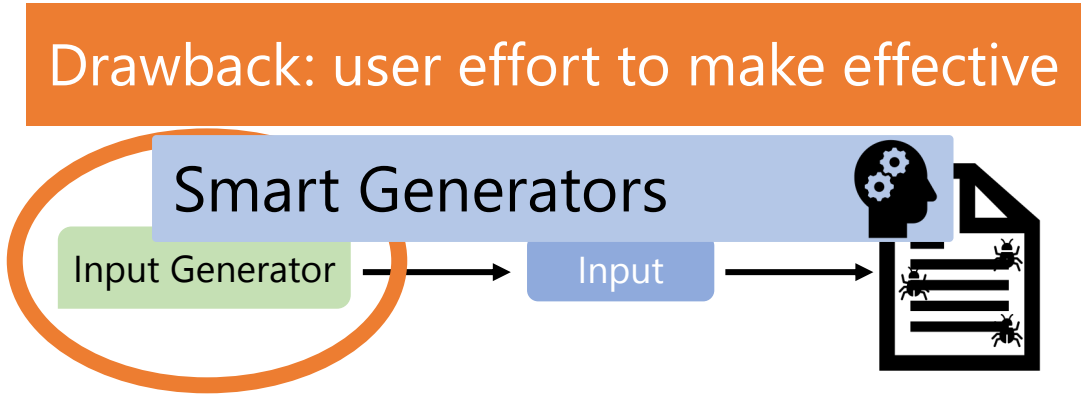


Modern Fuzzing

Coverage-Guided Fuzzing



Generator-Based Fuzzing



Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

Future Directions

Background on Fuzzing

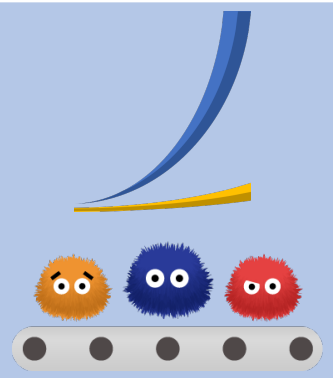
Performance Bugs 

Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing



PerfFuzz

Lemieux, Padhye, Sen & Song. ISSTA '18

FuzzFactory

Padhye, Lemieux, Sen, Laurent & Vijayakumar. OOPSLA '19

Exploring Core Logic

Smart Generators

Future Directions

Performance Bugs

Performance Bugs

Example Program: Word Frequency (`wf`)

- Count frequency of words in string

input:

```
the quick brown the dog
```

output:

```
brown: 1
dog: 1
quick: 1
the: 2
```

Example Program: Word Frequency (wf)

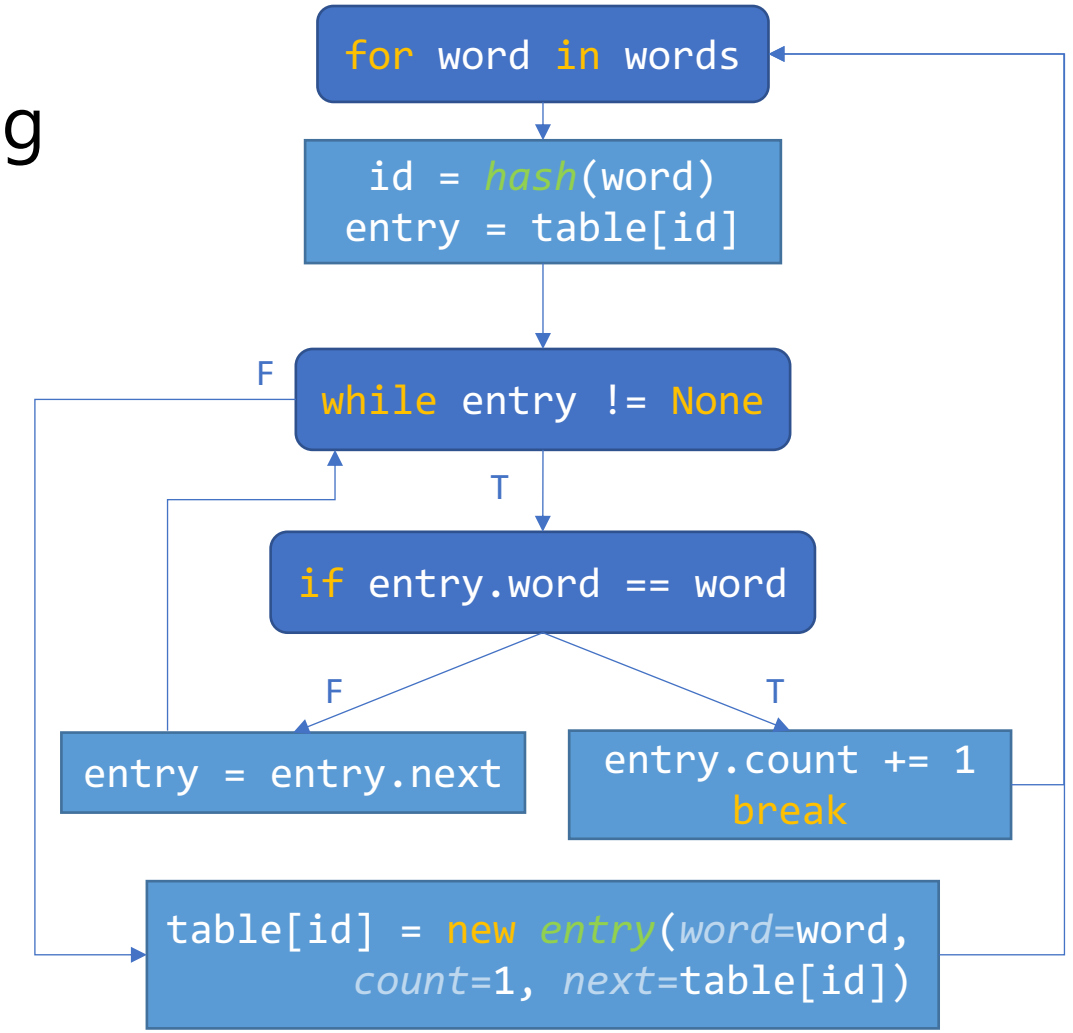
- Count frequency of words in string

input:

```
the quick brown the dog
```

output:

```
brown: 1
dog: 1
quick: 1
the: 2
```

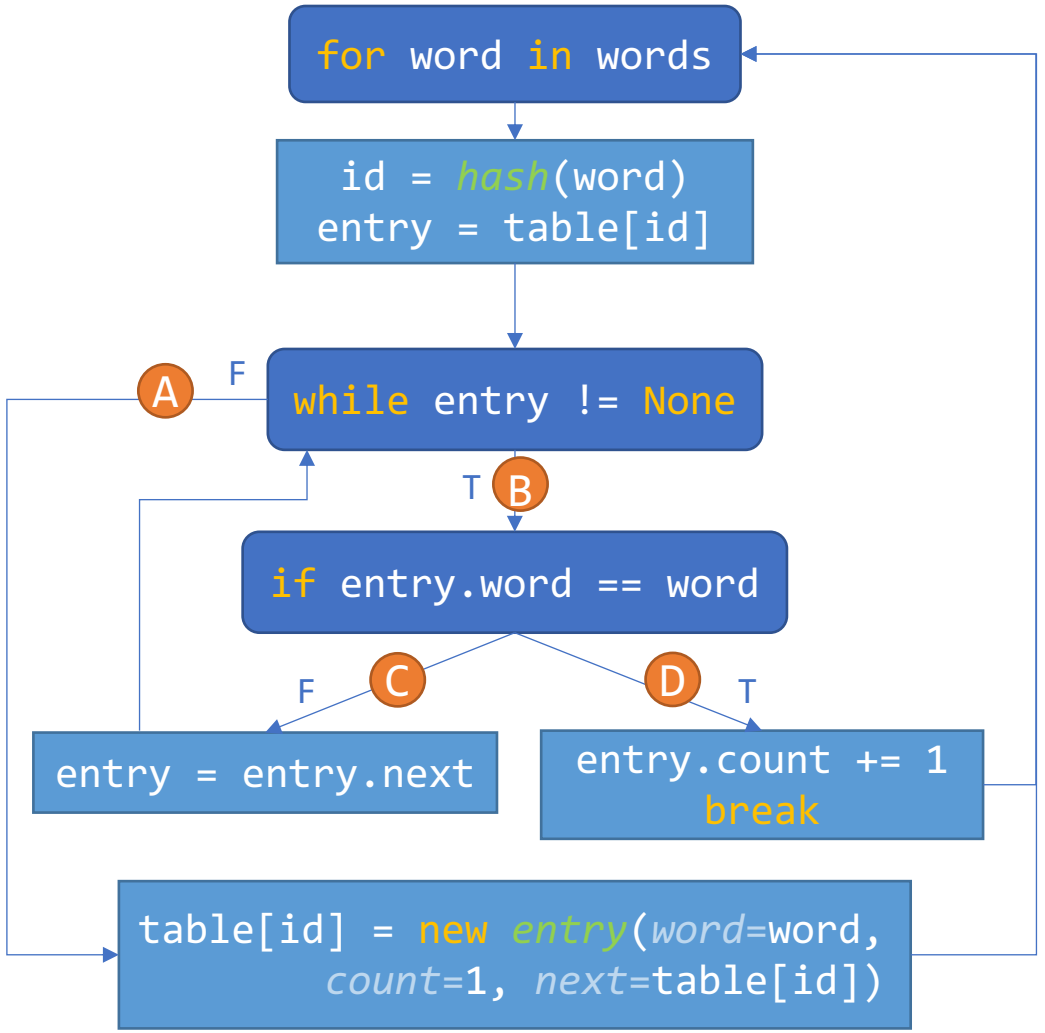


wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	
B	
C	
D	

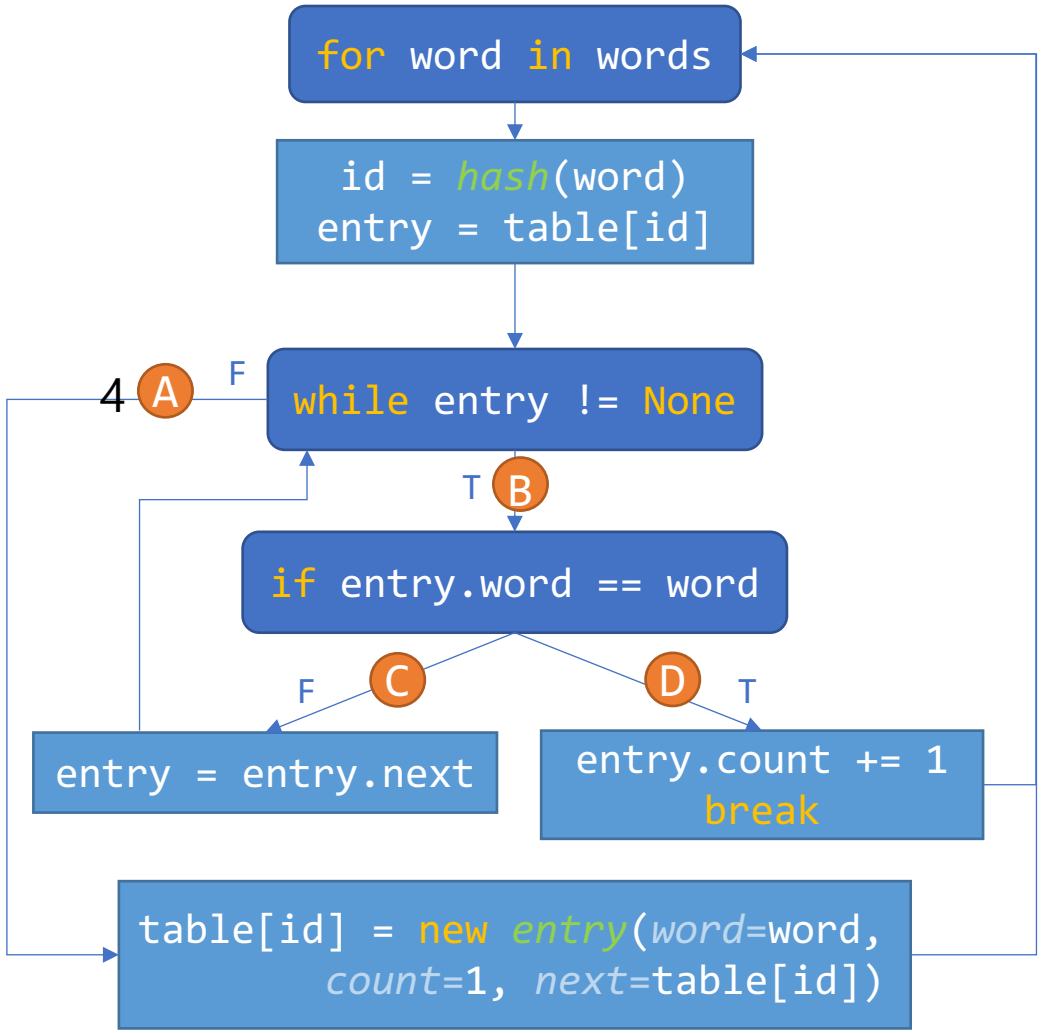


wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	
C	
D	

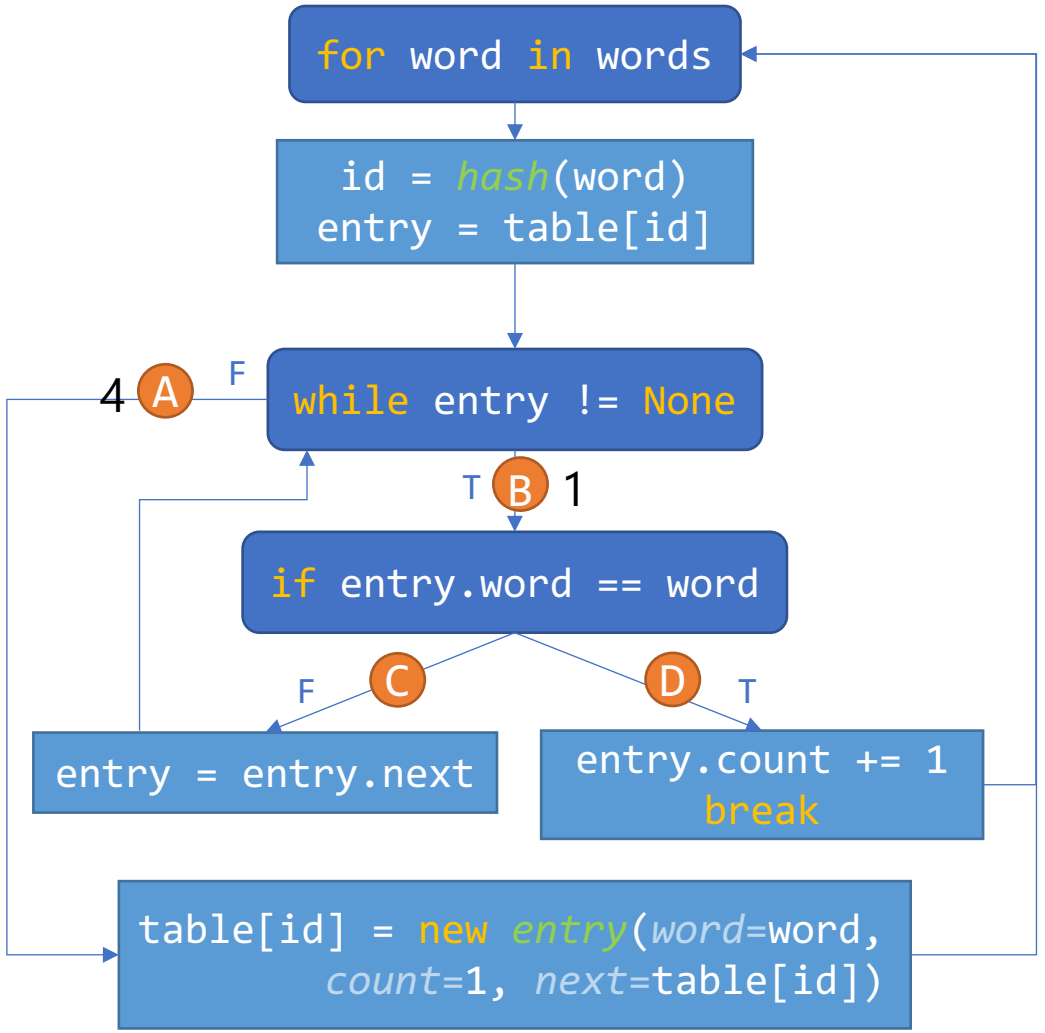


wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	
D	

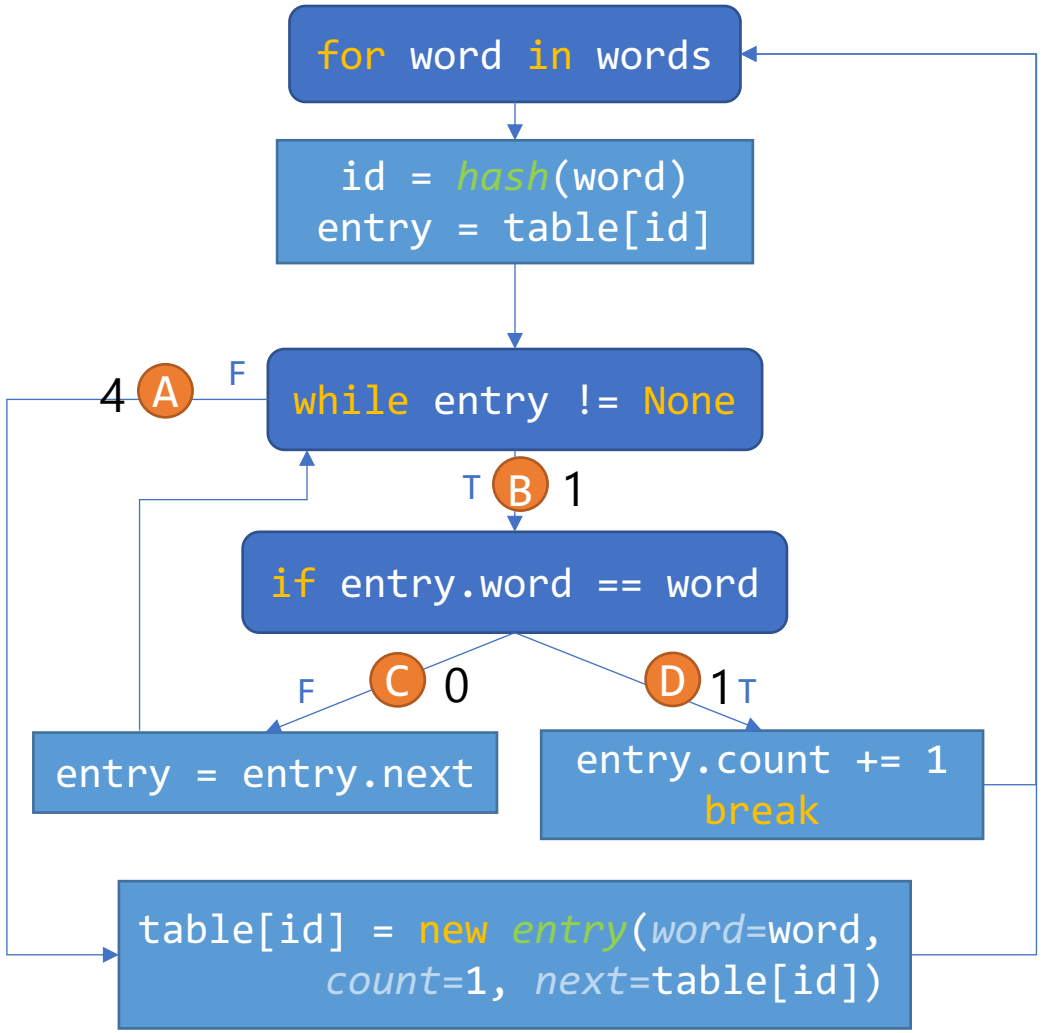


wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	0
D	1



wf Performance Response

- Usual case:

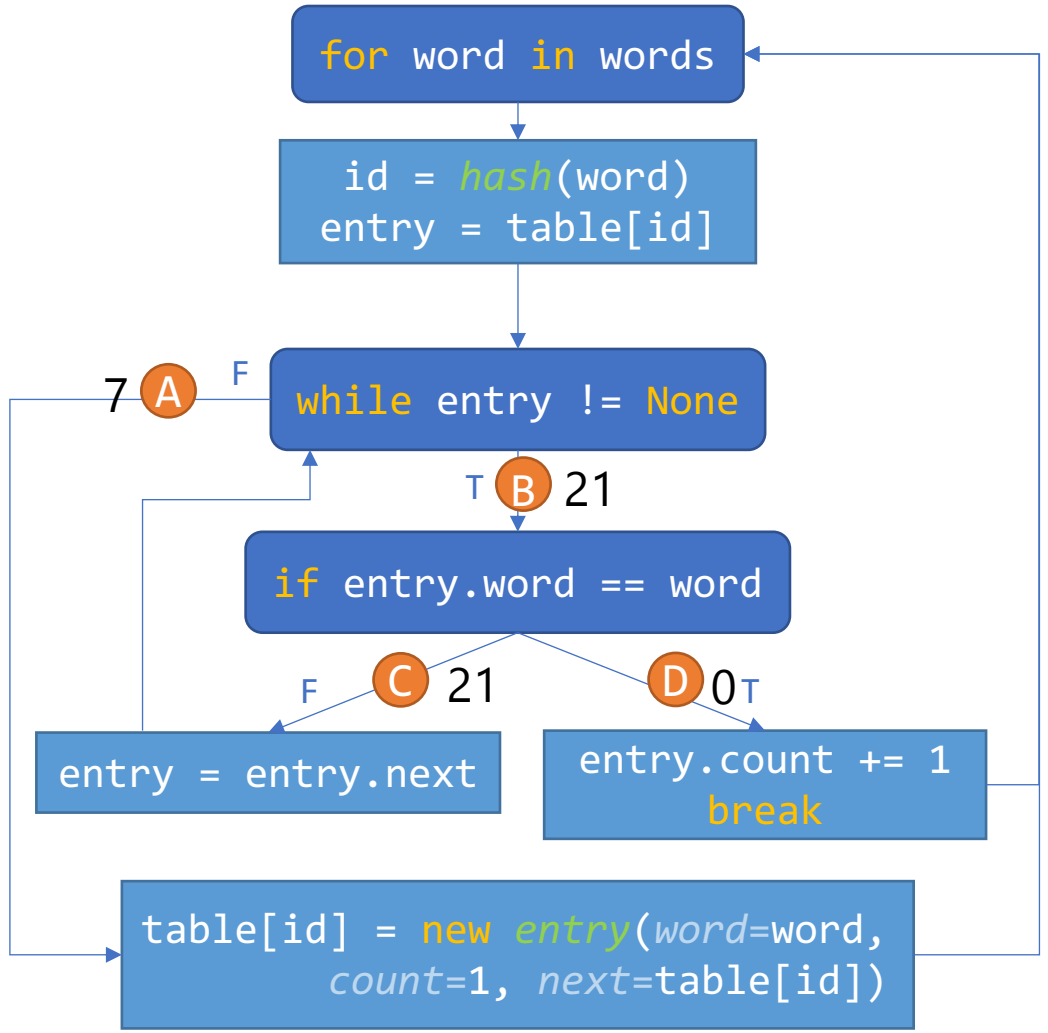
```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	0
D	1

- Hash collisions:

```
t ?t xt at$ #a ))t Qwaa
```

Edge	# Hits
A	7
B	21
C	21
D	0



wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	0
D	1

- Hash collisions:

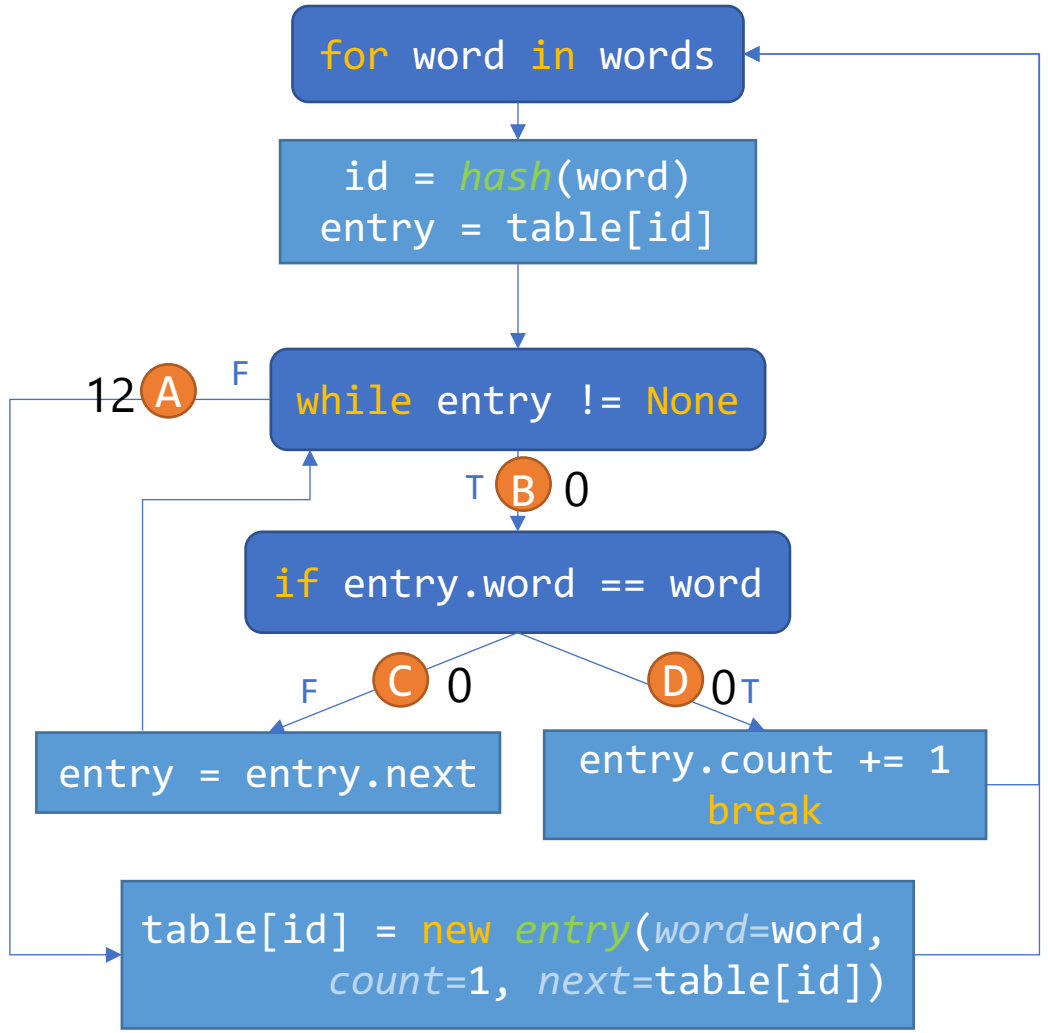
```
t ?t xt at$ #a ))t Qwaa
```

Edge	# Hits
A	7
B	21
C	21
D	0

- Small words:

```
t h e q u i c k b r o w
```

Edge	# Hits
A	12
B	0
C	0
D	0



wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	0
D	1

- Hash collisions:

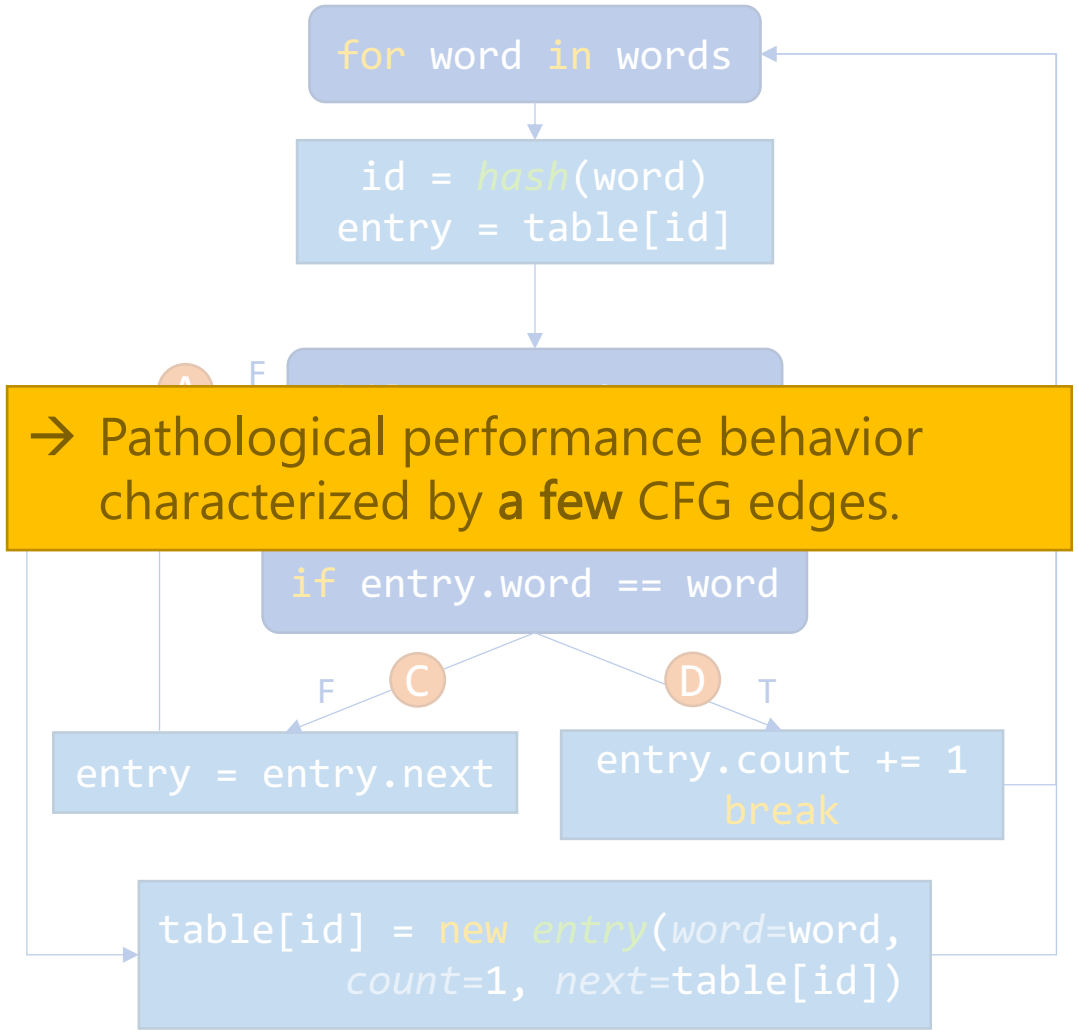
```
t ?t xt at$ #a ))t Qwaa
```

Edge	# Hits
A	7
B	21
C	21
D	0

- Small words:

```
t h e q u i c k b r o w
```

Edge	# Hits
A	12
B	0
C	0
D	0



wf Performance Response

- Usual case:

```
the quick brown the dog
```

Edge	# Hits
A	4
B	1
C	0
D	1

- Hash collisions:

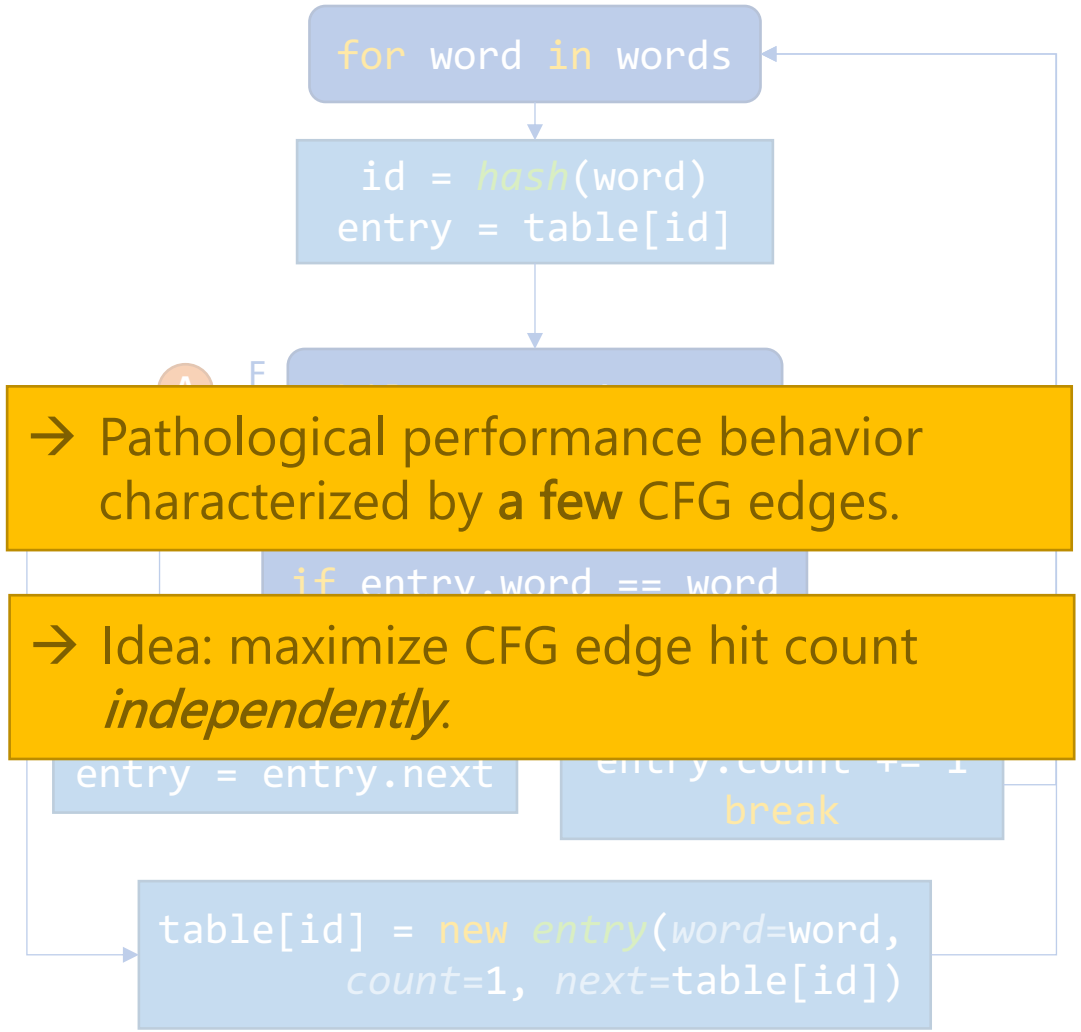
```
t ?t xt at$ #a ))t Qwaa
```

Edge	# Hits
A	7
B	21
C	21
D	0

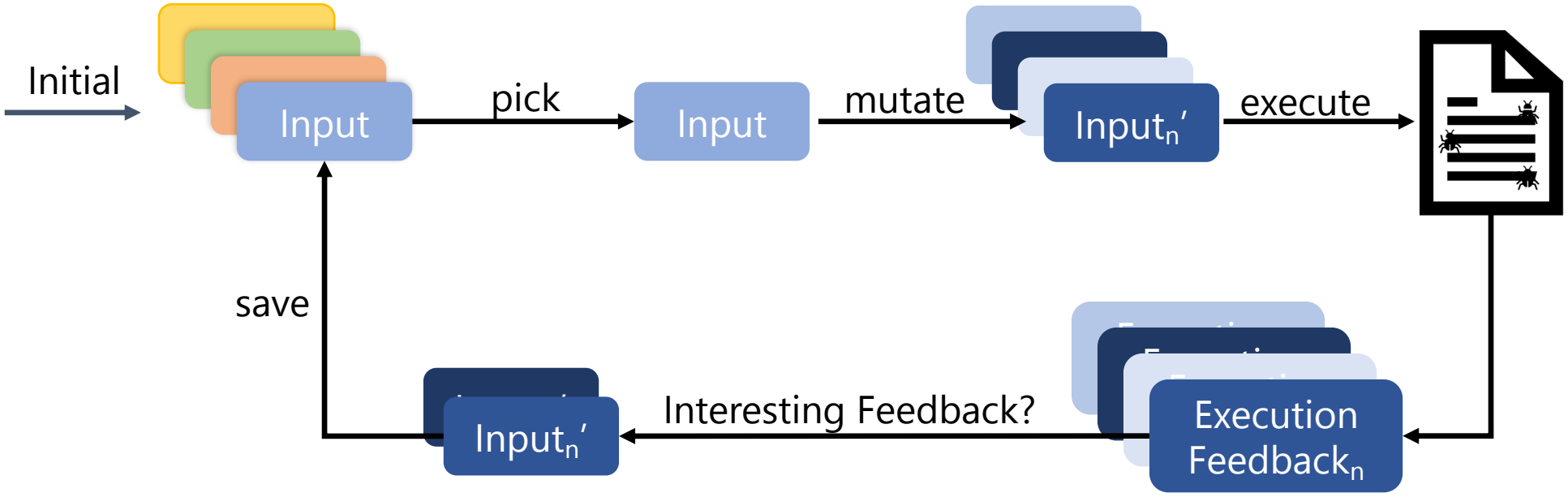
- Small words:

```
t h e q u i c k b r o w
```

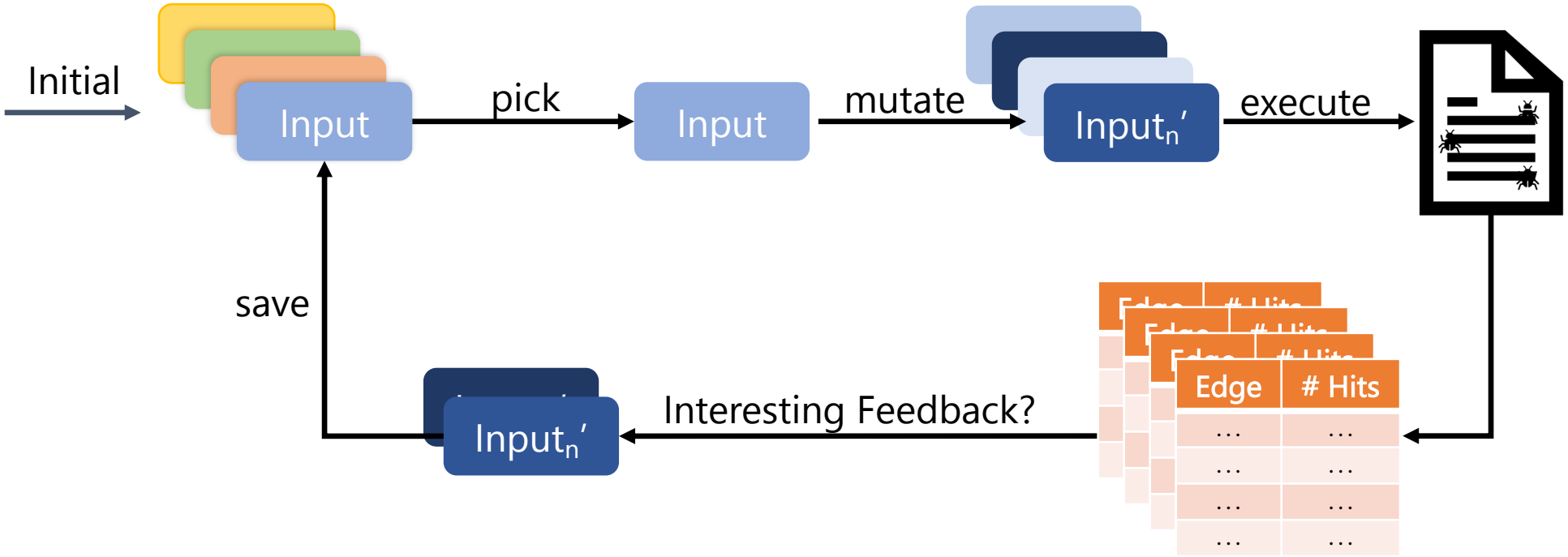
Edge	# Hits
A	12
B	0
C	0
D	0



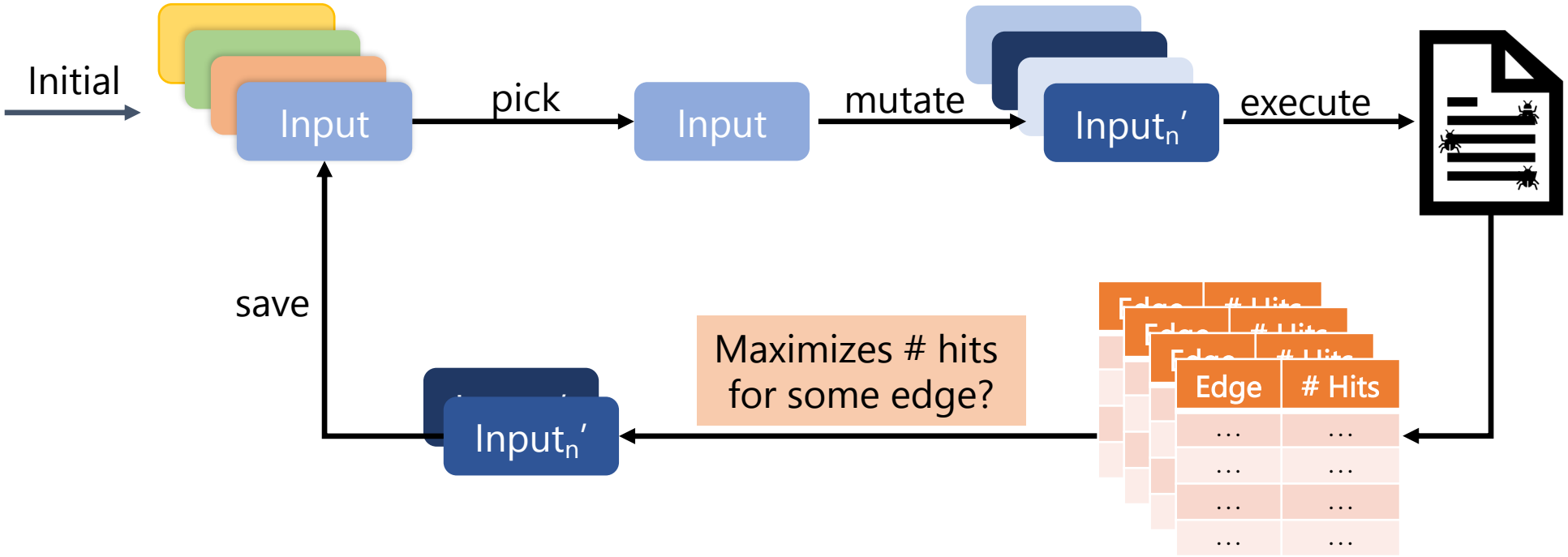
Coverage-Guided Fuzzing



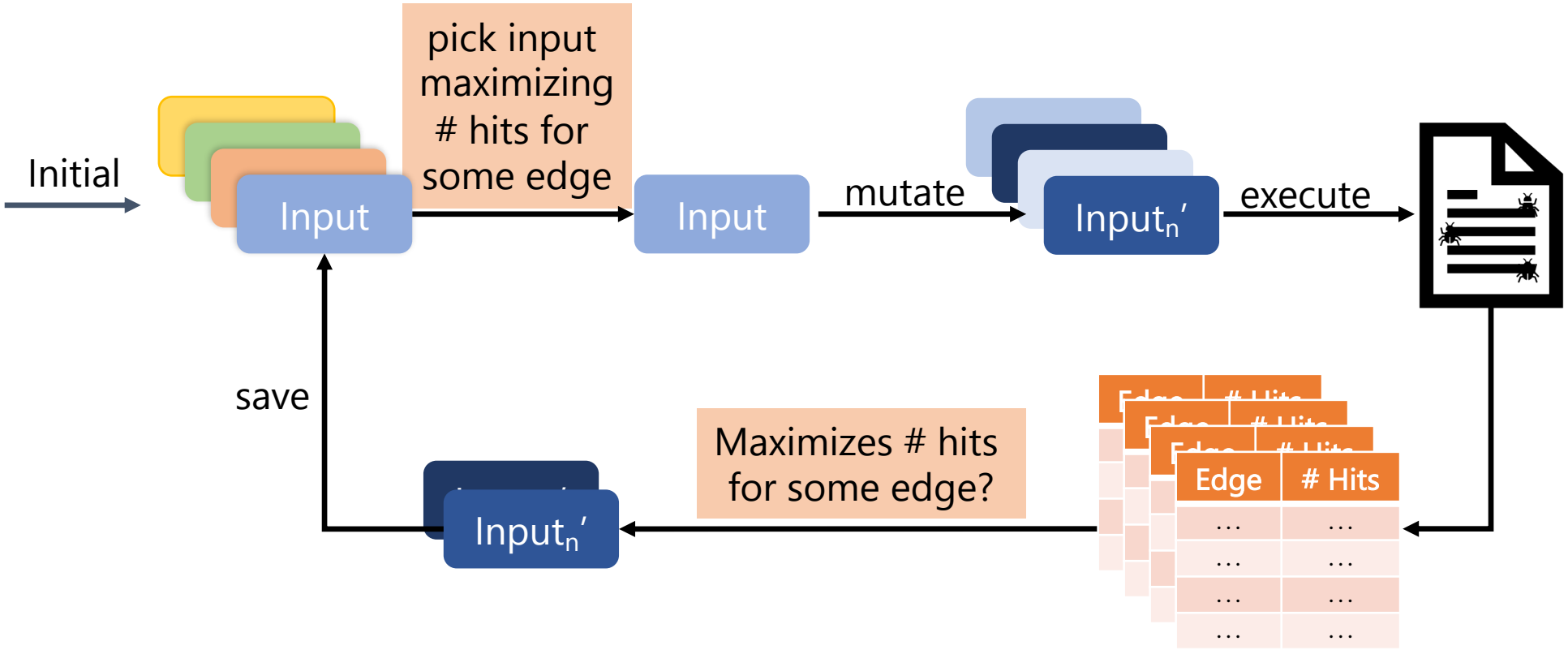
PerfFuzz


Initial
Input
pick
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Interesting Feedback?
Edge
Hits

PerfFuzz


Initial
Input
pick
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Maximizes # hits for some edge?
Edge
Hits

PerfFuzz


Initial
Input
pick input maximizing # hits for some edge
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Maximizes # hits for some edge?
Edge
Hits

`wf` results: PerfFuzz Finds True Worst Cases

`wf` results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

```
t r t t s f o Öe r t s f o r t x x t s f o r t x x
```

`wf` results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

```
t r t t s f o Öe r t s f o r t x x t s f o r t x x
```

PerfFuzz worst case:

```
t <81>v ^?@t <80>!^?@t <80>!t t^Rn t t t t t t t t t
```

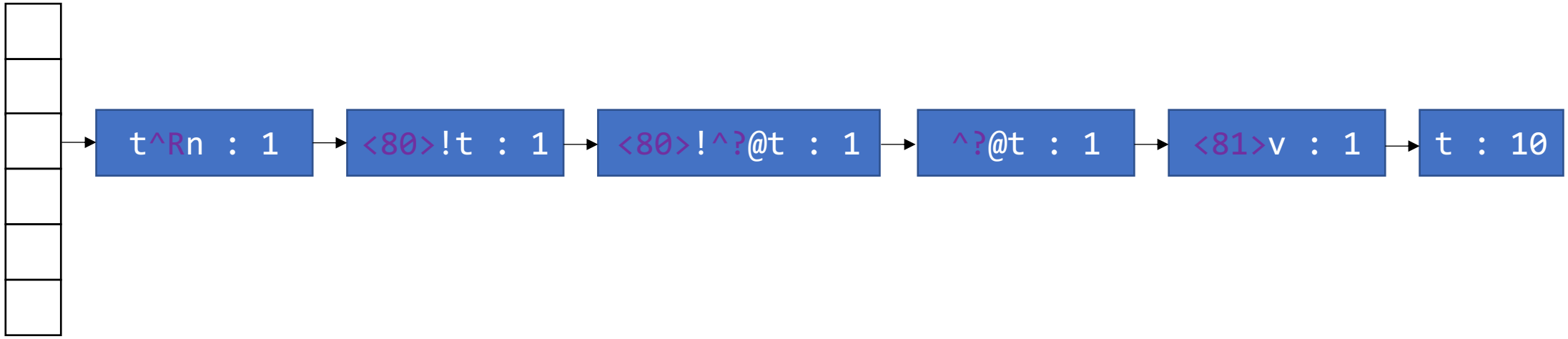
wf results: PerfFuzz Finds True Worst Cases

SlowFuzz (single objective maximization) worst case:

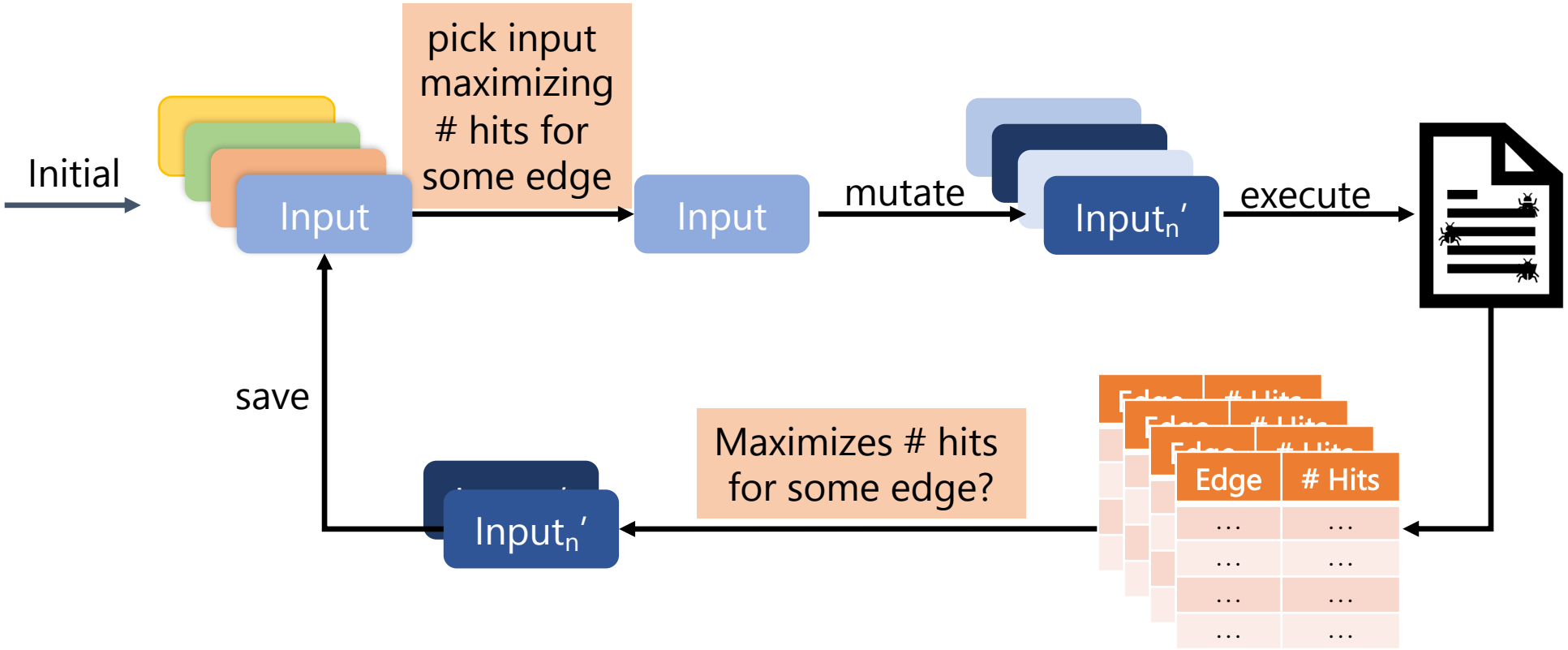
t r t t s f o Öe r t s f o r t x x t s f o r t x x

PerfFuzz worst case:

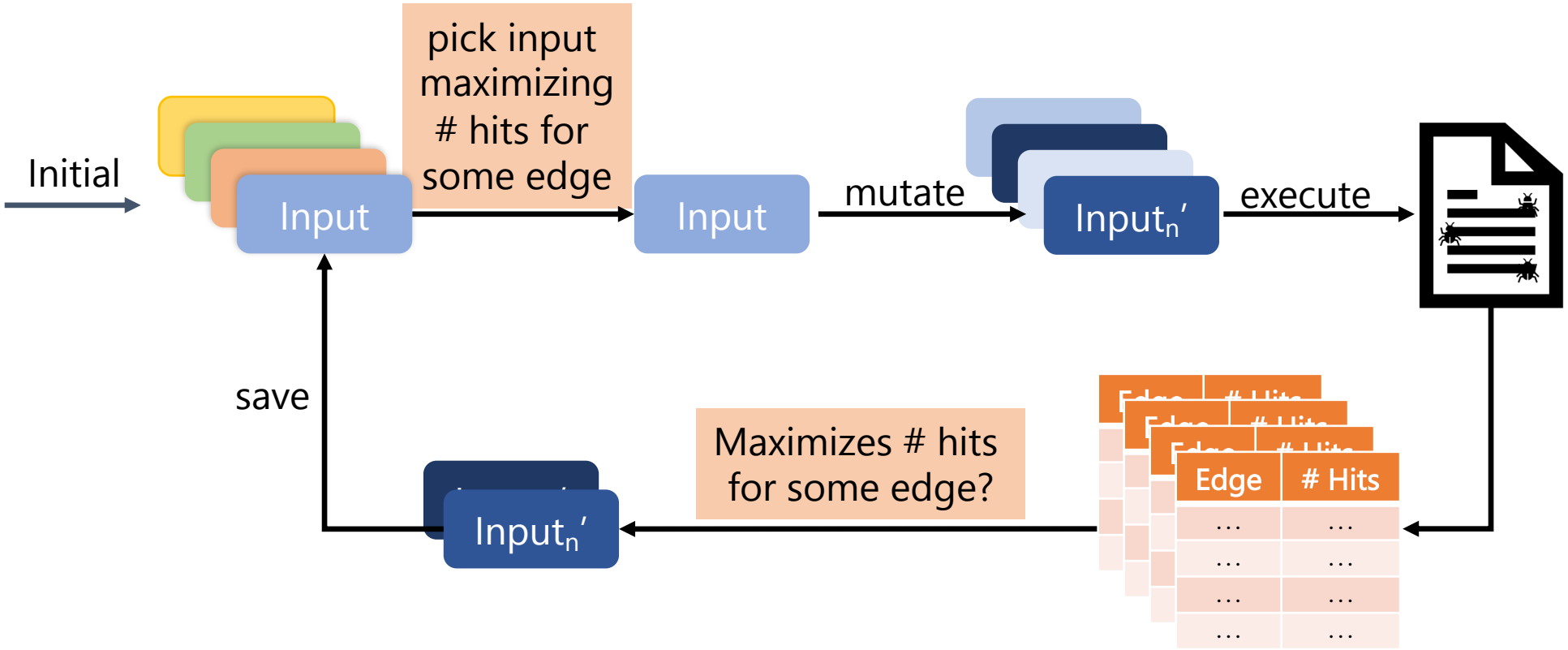
t <81>v ^?@t <80>!^?@t <80>!t t^Rn t t t t t t t t t



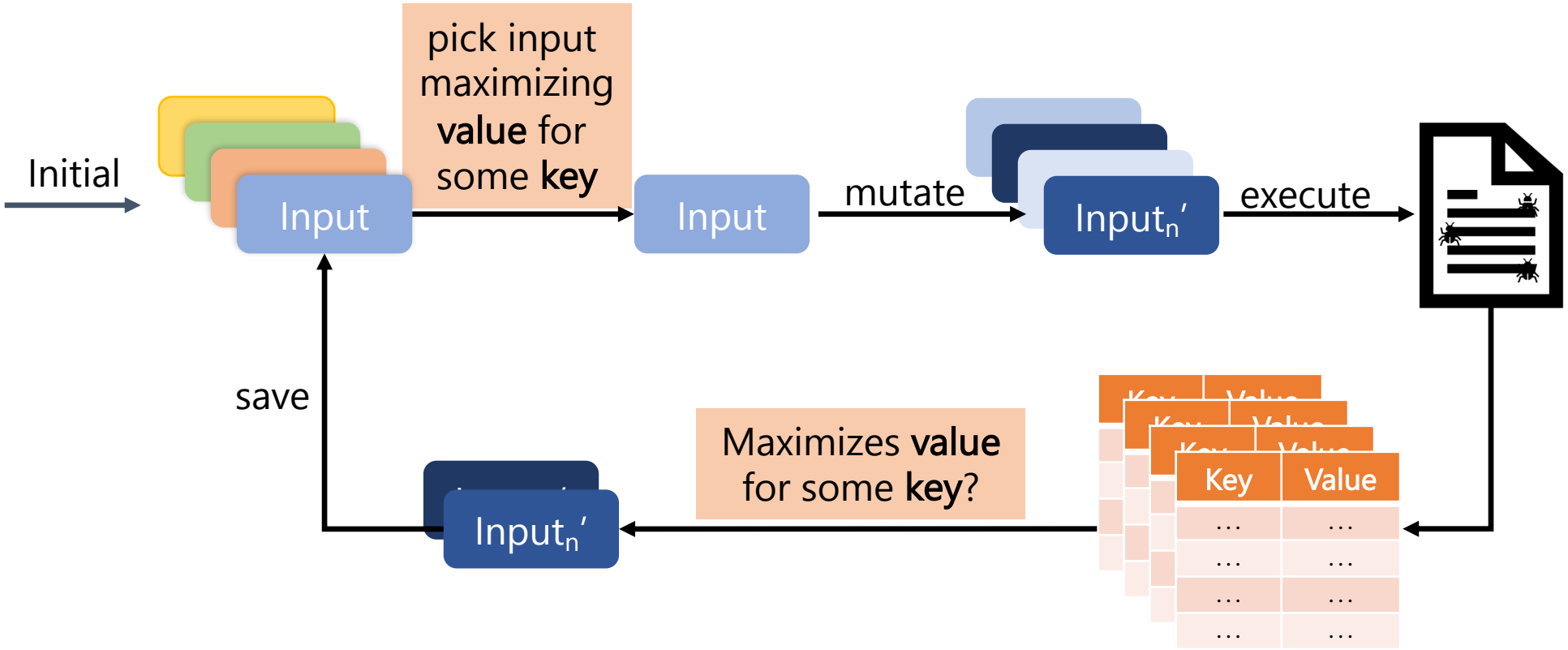
PerfFuzz


Initial
Input
pick input maximizing # hits for some edge
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Maximizes # hits for some edge?
Edge
Hits

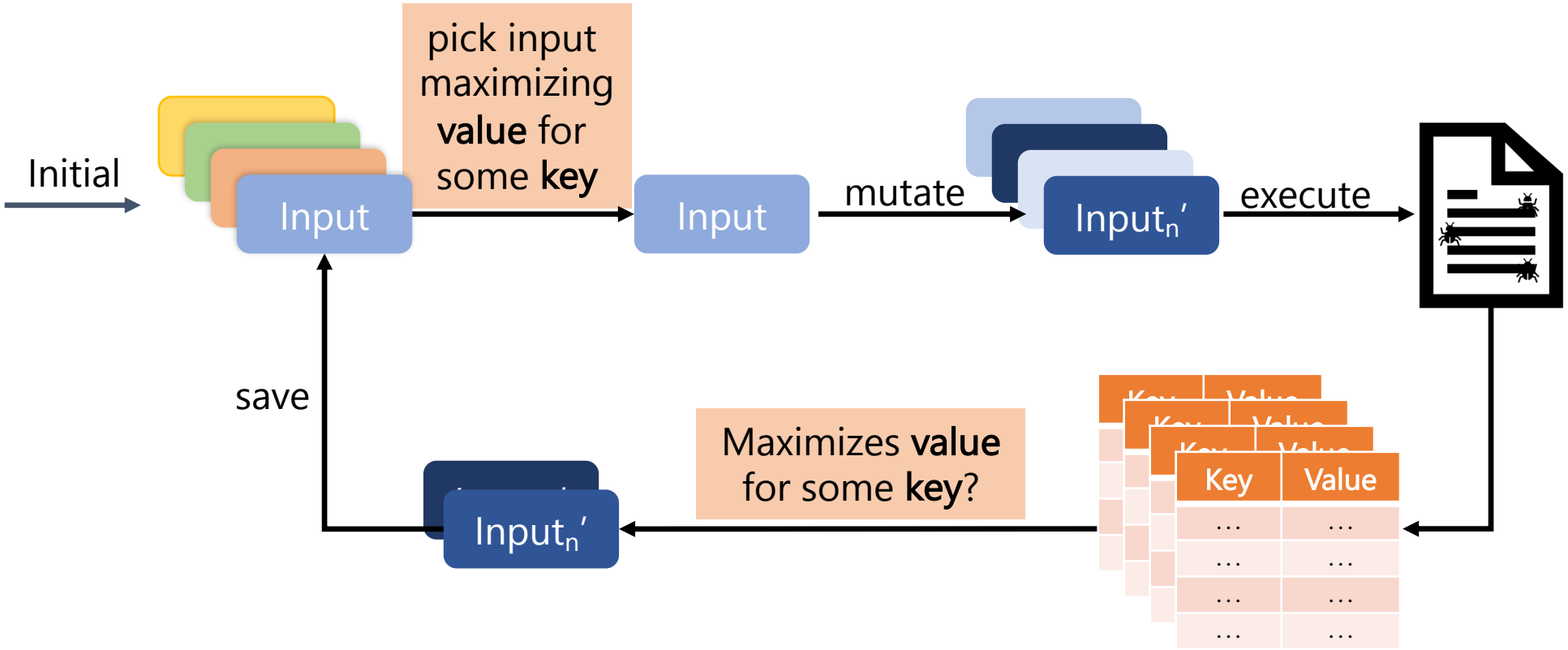
Observation: Algorithm is More General



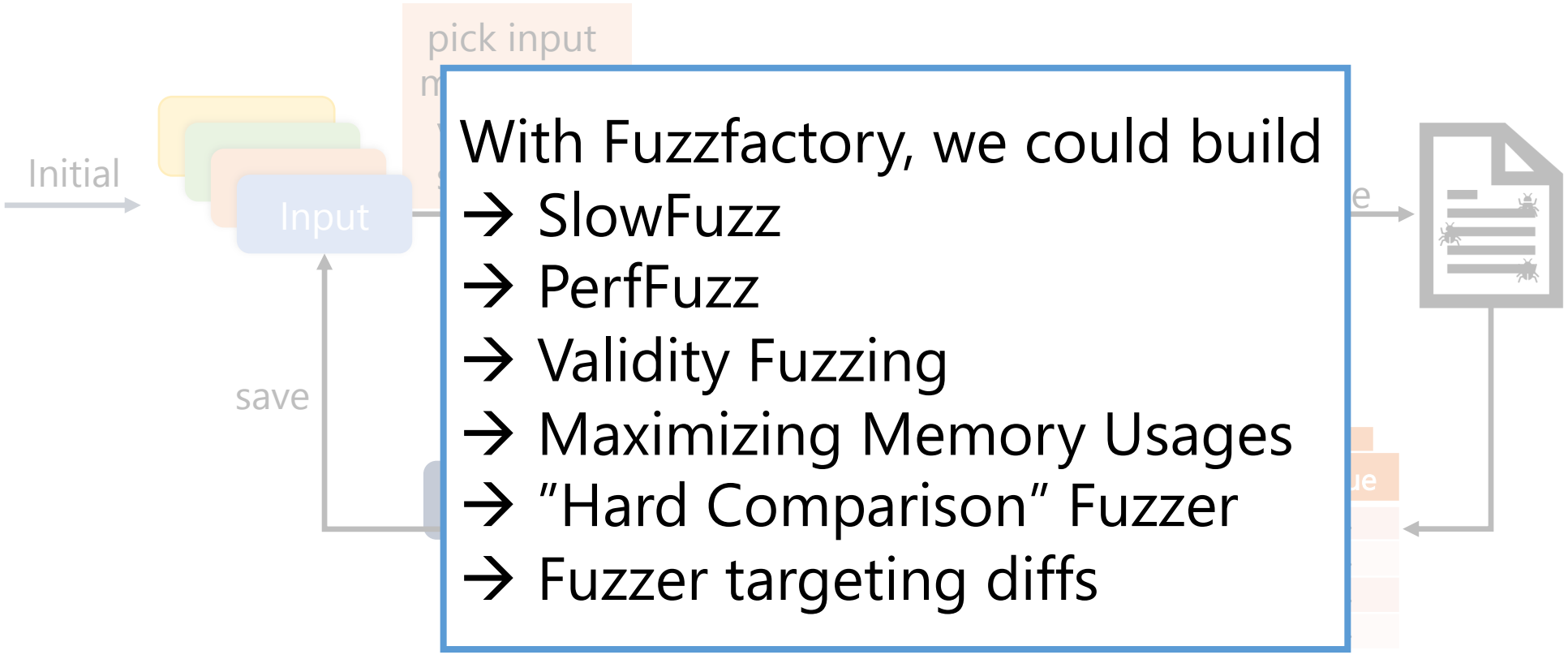
Observation: Algorithm is More General



FuzzFactory

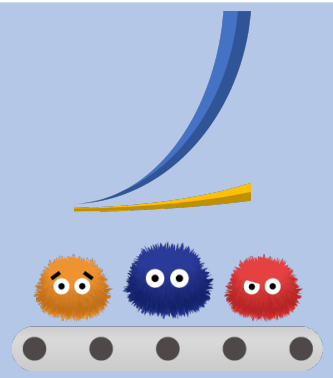


FuzzFactory



With Fuzzfactory, we could build
→ SlowFuzz
→ PerfFuzz
→ Validity Fuzzing
→ Maximizing Memory Usages
→ ”Hard Comparison” Fuzzer
→ Fuzzer targeting diffs

Background on Fuzzing



PerfFuzz

Lemieux, Padhye, Sen & Song. ISSTA '18

FuzzFactory

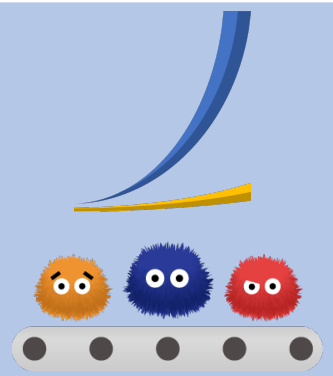
Padhye, Lemieux, Sen, Laurent & Vijayakumar. OOPSLA '19

Exploring Core Logic

Smart Generators

Future Directions

Background on Fuzzing

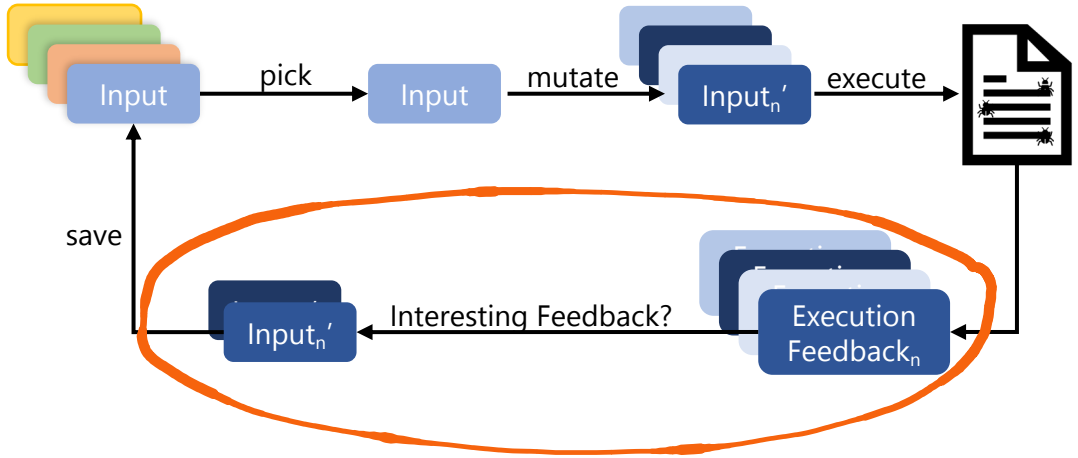


PerfFuzz

Lemieux, Padhye, Sen & Song. ISSTA '18

FuzzFactory

Padhye, Lemieux, Sen, Laurent & Vijayakumar. OOPSLA '19



Generalize "feedback" with map-based abstraction
→ new applications for CGF algorithm

Background on Fuzzing

Performance Bugs 

Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing

Performance Bugs 

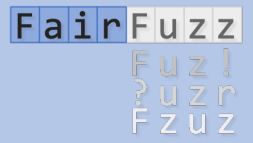
Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing

Performance Bugs



FairFuzz

Lemieux & Sen. ASE ‘18

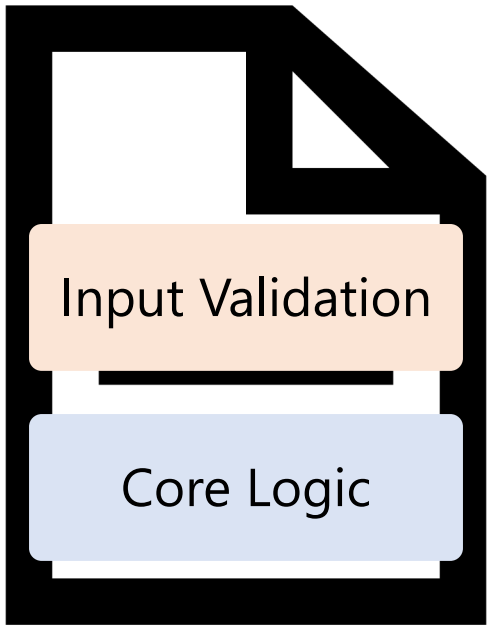
Zest

Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ‘19

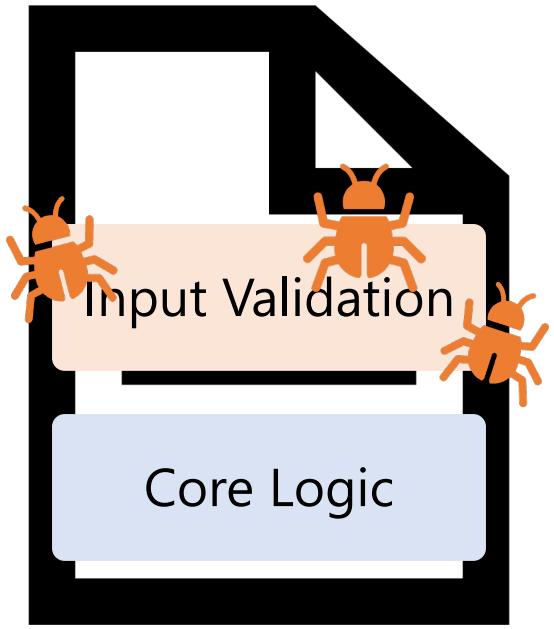
Smart Generators

Future Directions

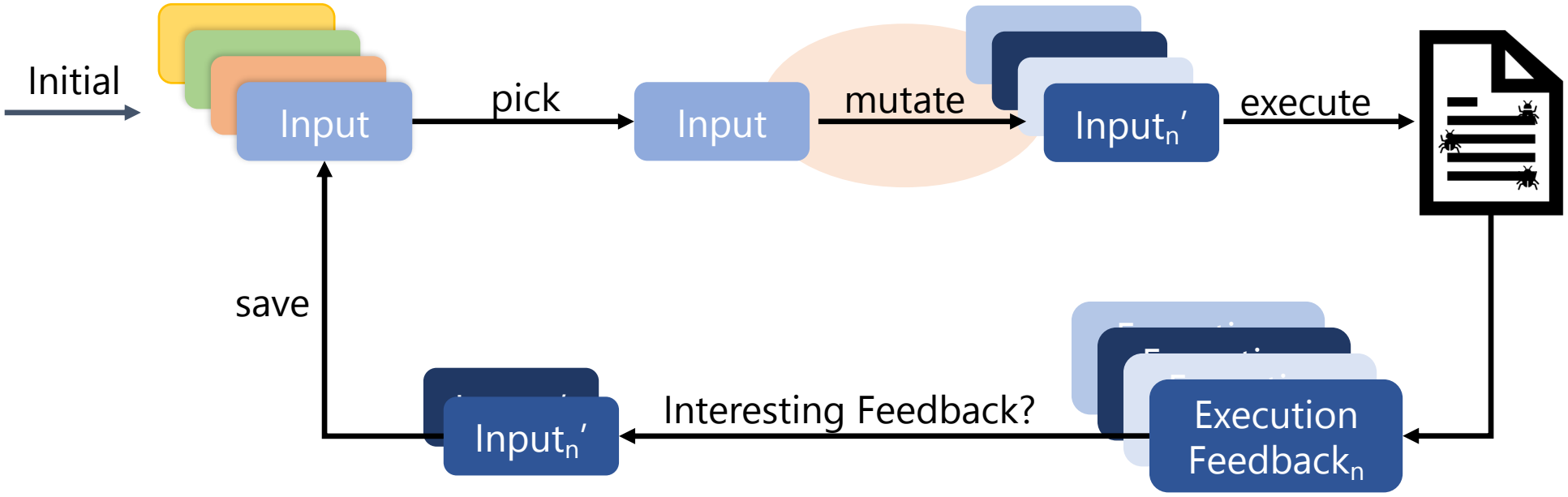
Where Are the Fuzzer-Found Bugs?



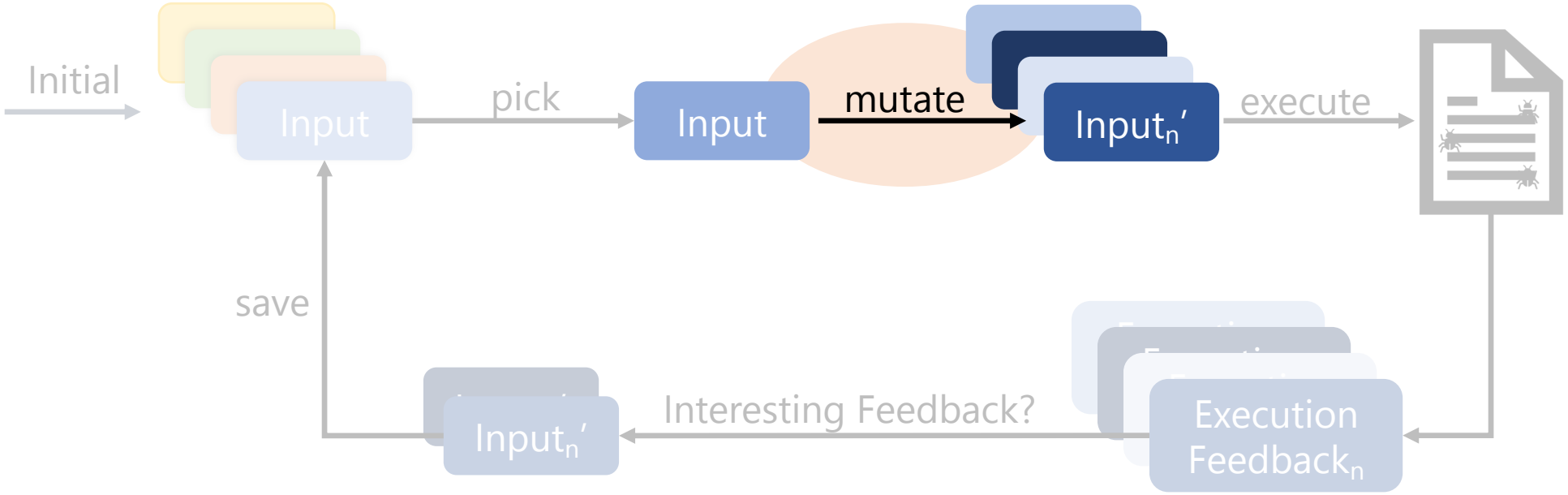
Where Are the Fuzzer-Found Bugs?



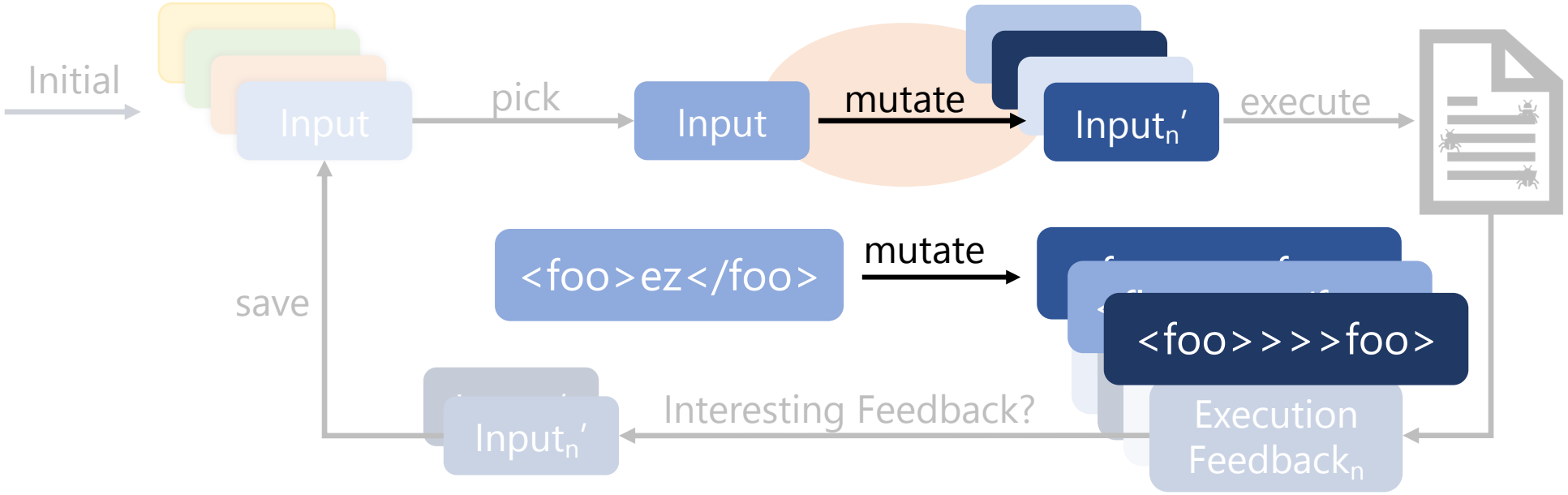
Problem: Random Mutations Ruin Structure



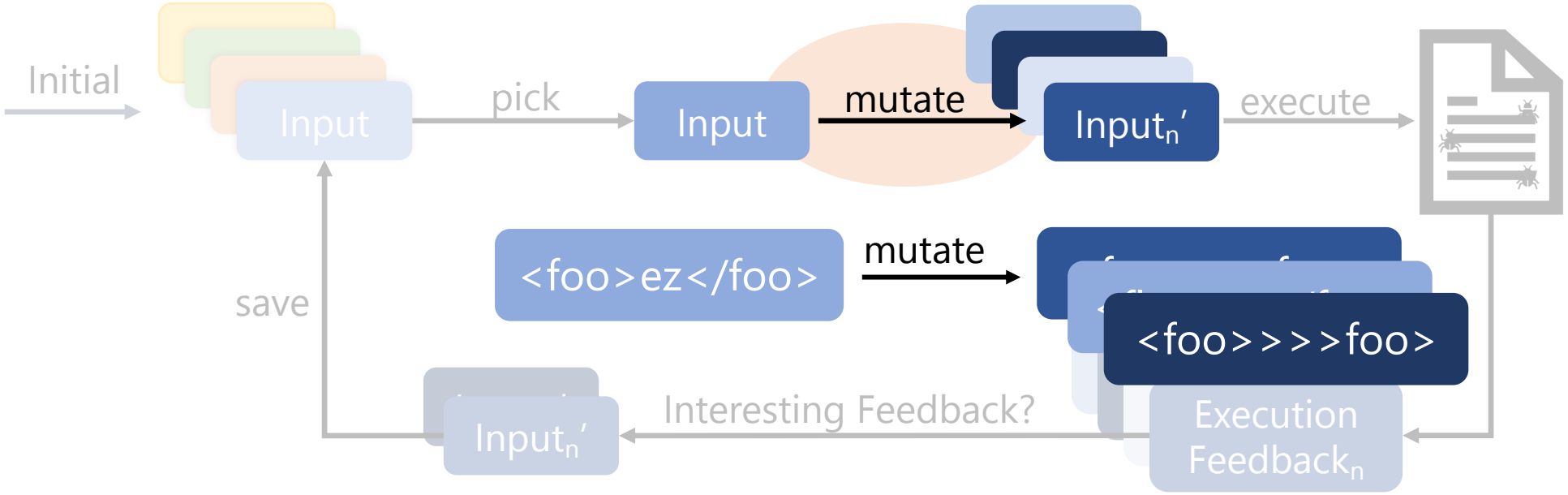
Problem: Random Mutations Ruin Structure



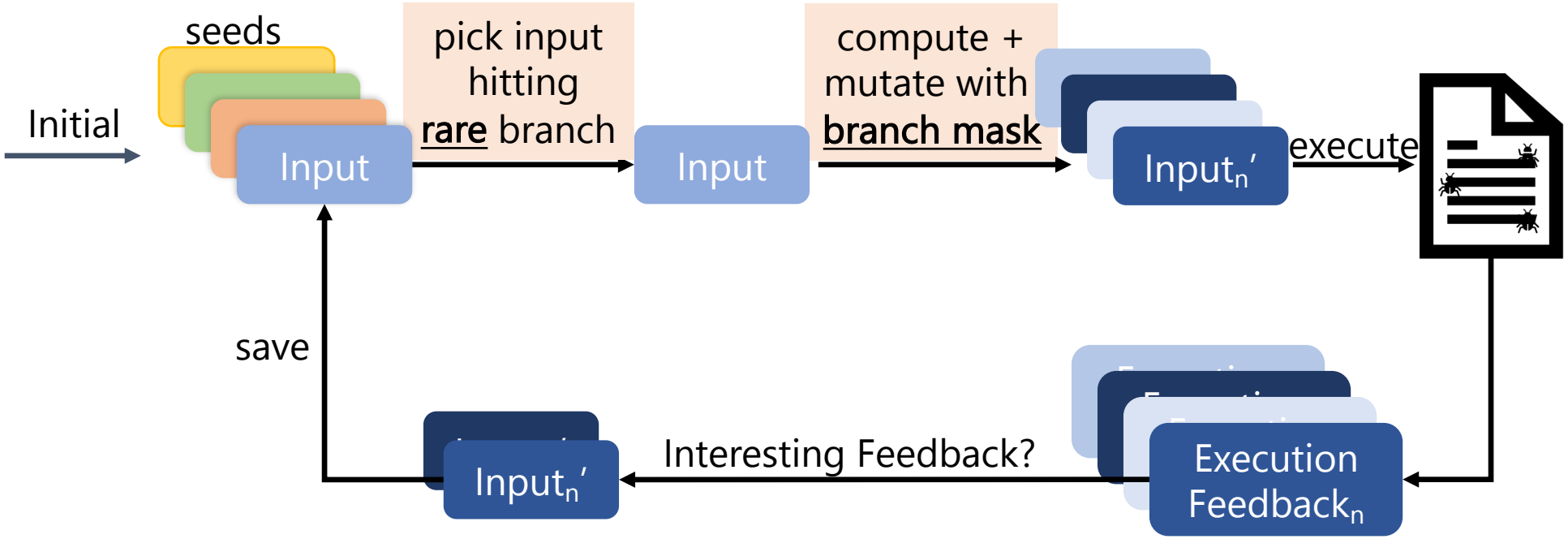
Problem: Random Mutations Ruin Structure



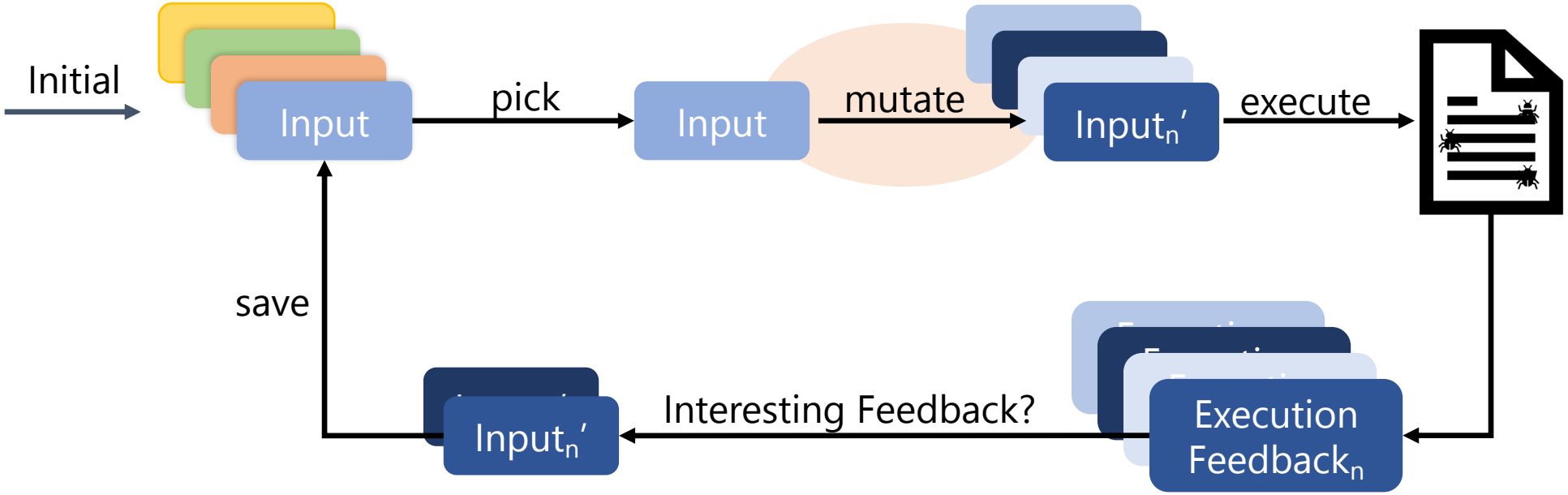
How to Retain Important Structure?



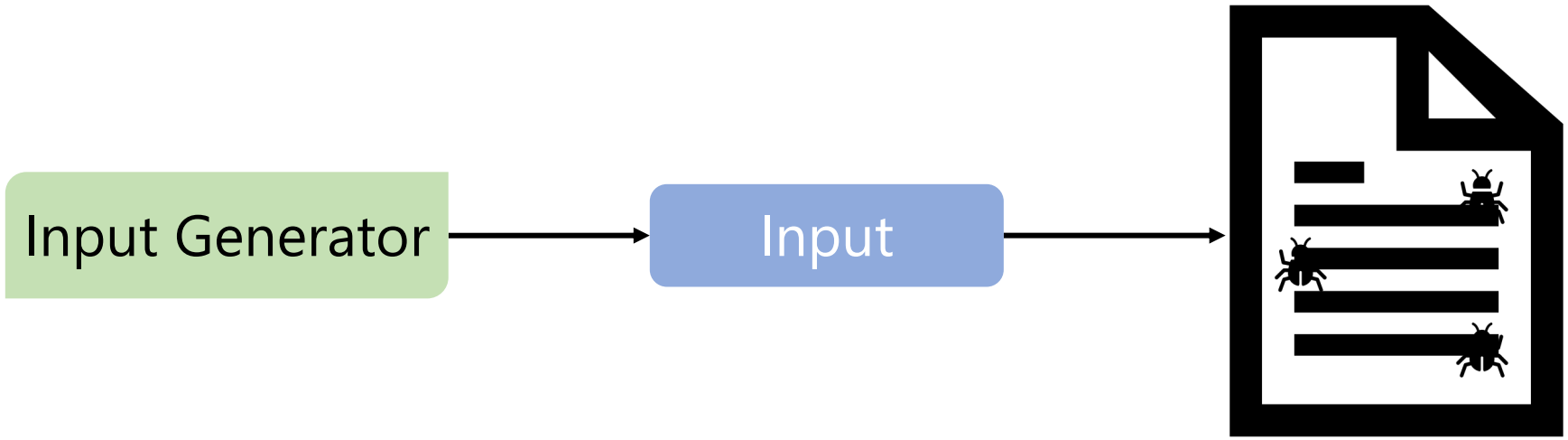
FairFuzz: Filter Mutations Likely to Ruin Structure



Can we get higher-level mutations?
with more information about input structure?



Generators as Input Structure Specification

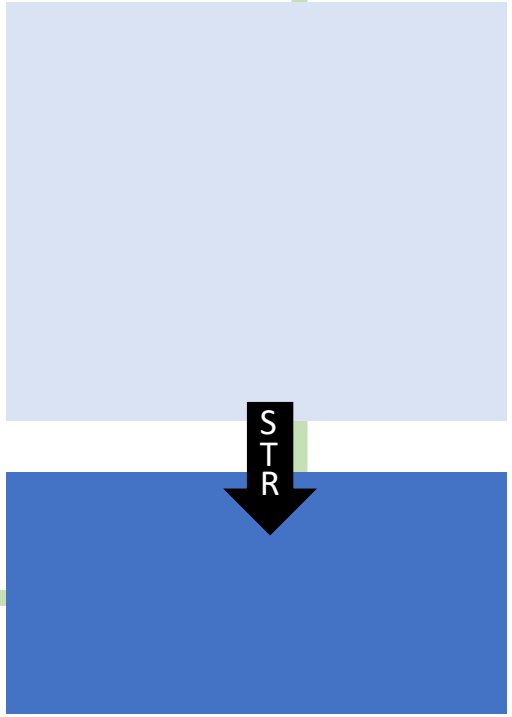


How to Get Mutations?

```
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
```

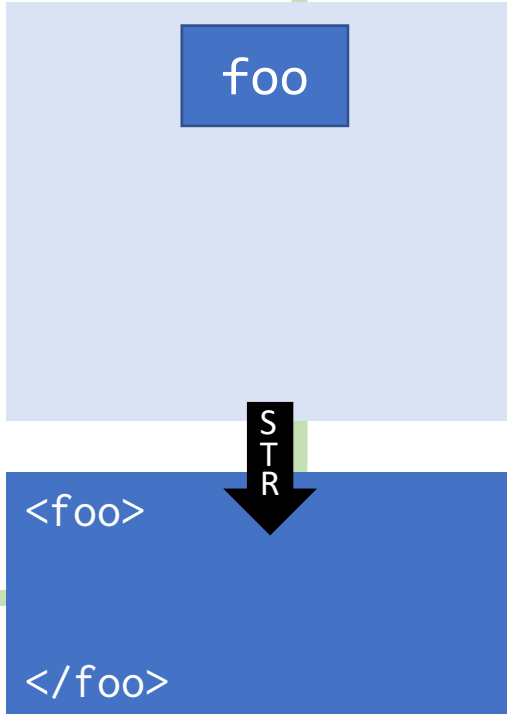
Generator: Source of Randomness → Input

```
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
```



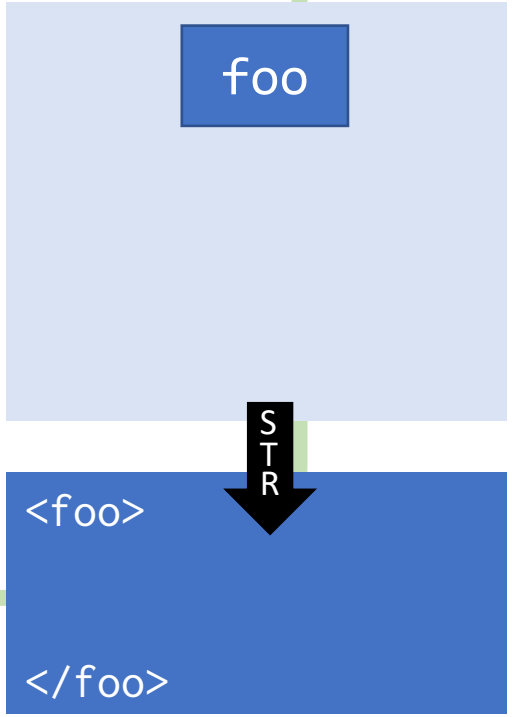
Generator: Source of Randomness → Input

```
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
```



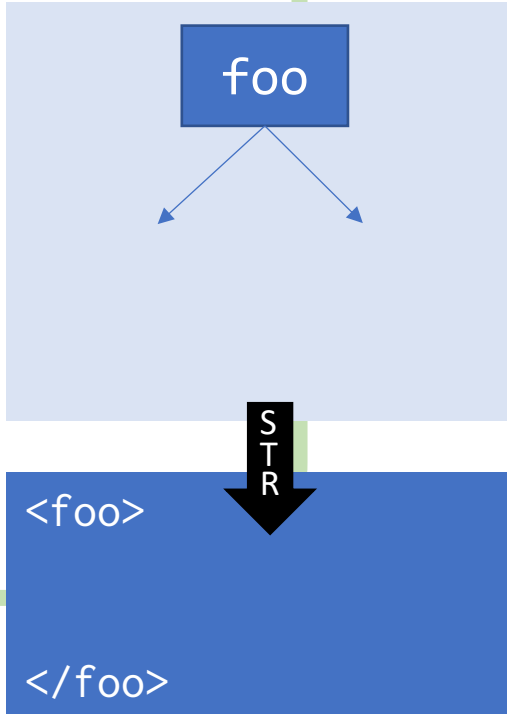
Generator: Source of Randomness → Input

```
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
```



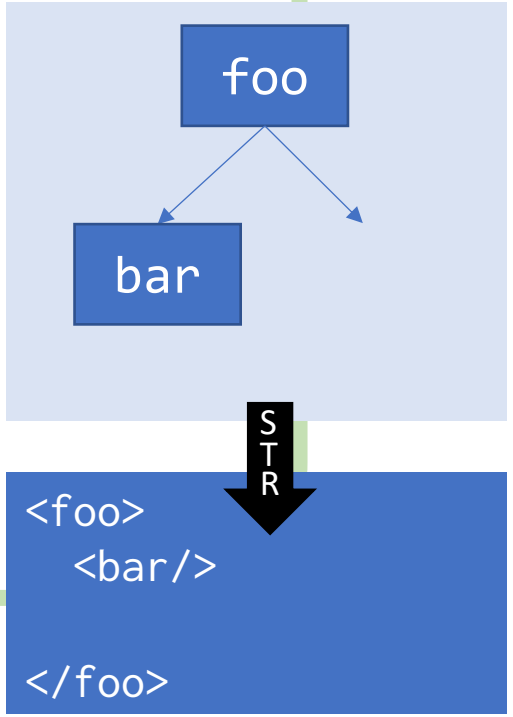
Generator: Source of Randomness → Input

```
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
```



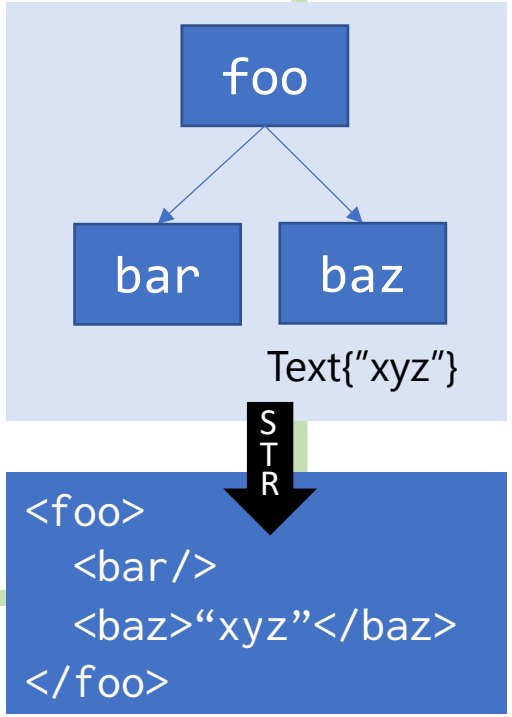
Generator: Source of Randomness → Input

```
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
```



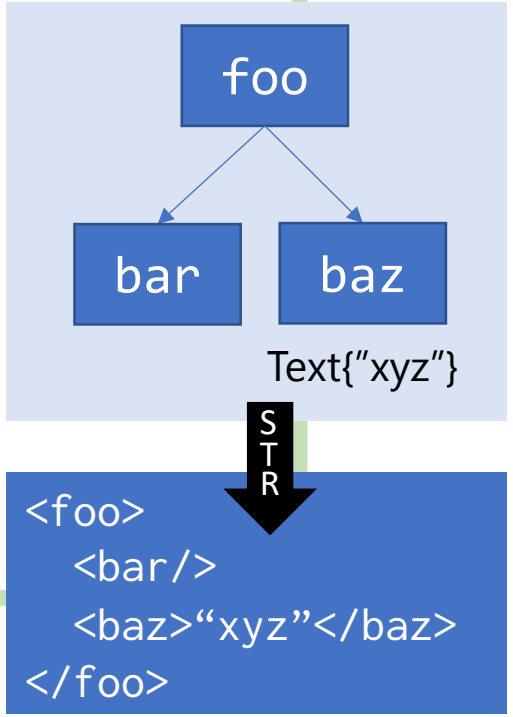
Generator: Source of Randomness → Input

```
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

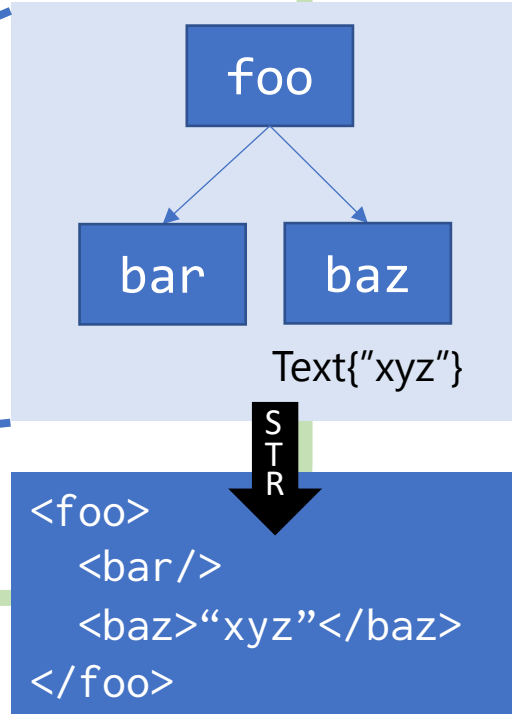
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



Source of Randomness == Infinite Bit-Sequence

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

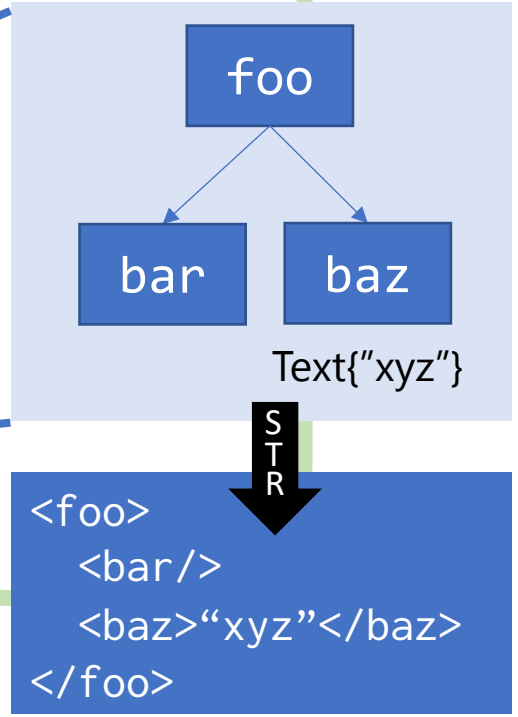
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random)
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



Bit Mutations → Structured Input Mutations

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

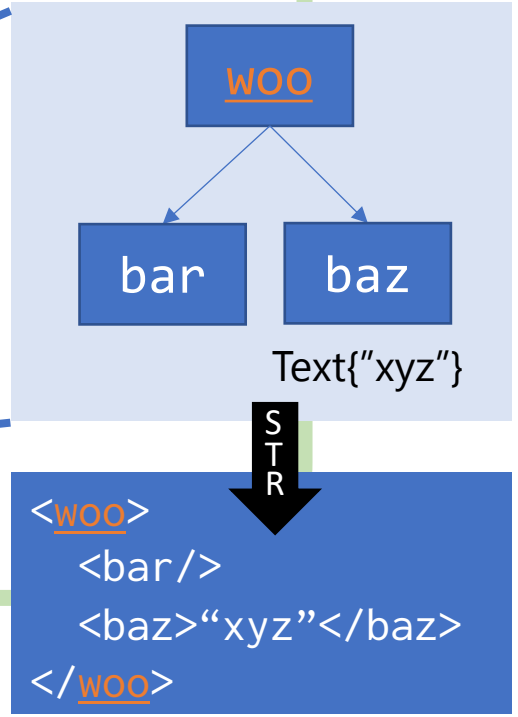
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random)
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



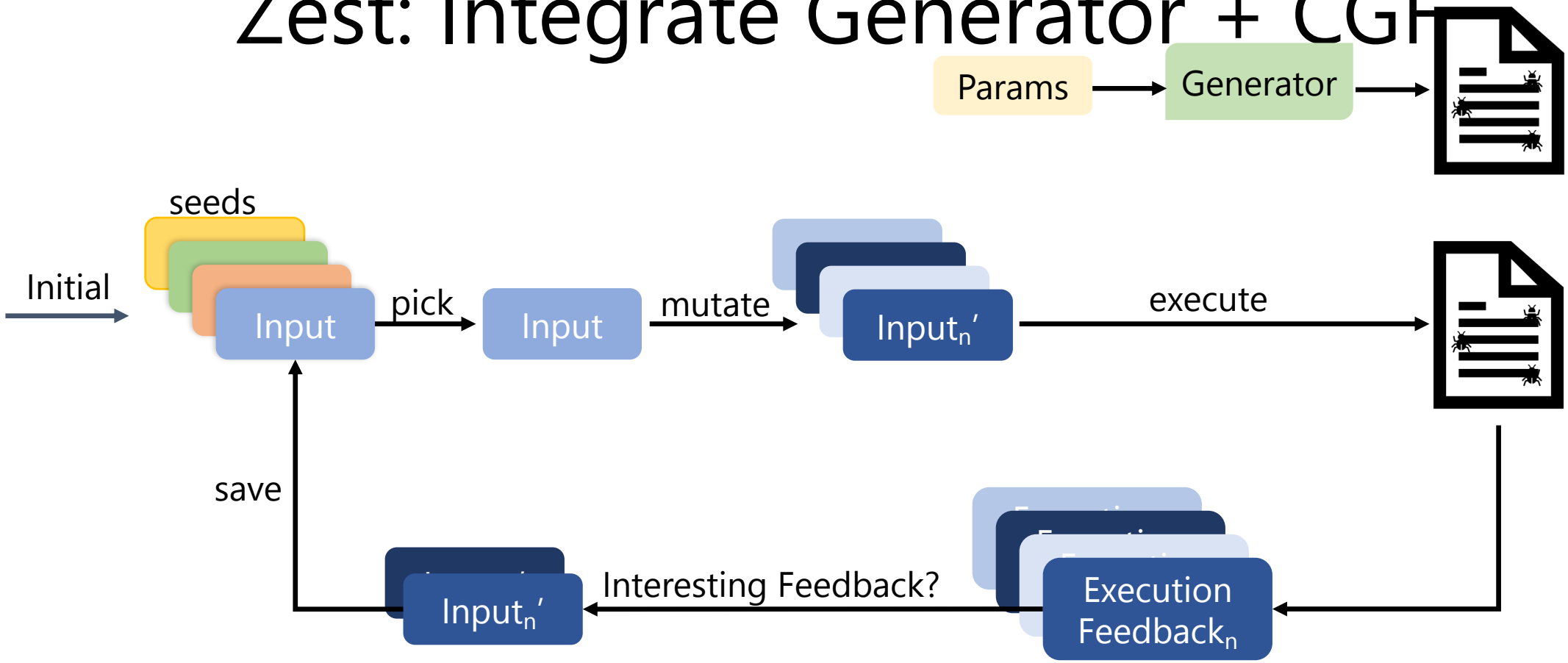
Bit Mutations → Structured Input Mutations

pseudo-random bits: 0000 0011 0101 0111 0110 1111 0110 1111 0000 0010 …

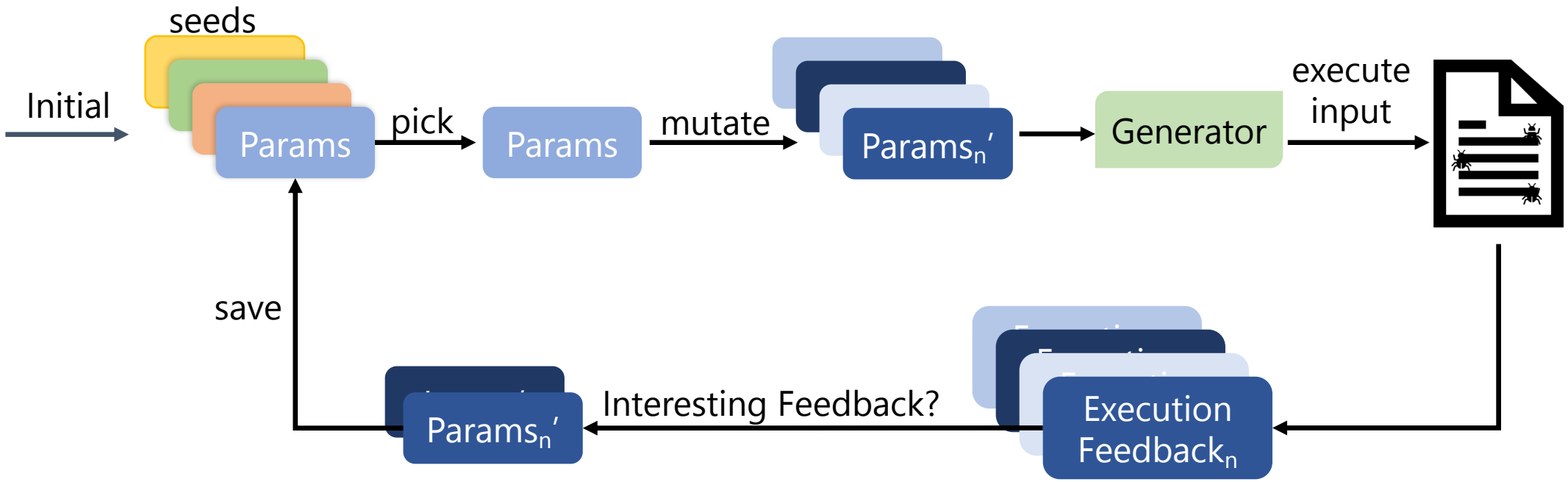
```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```



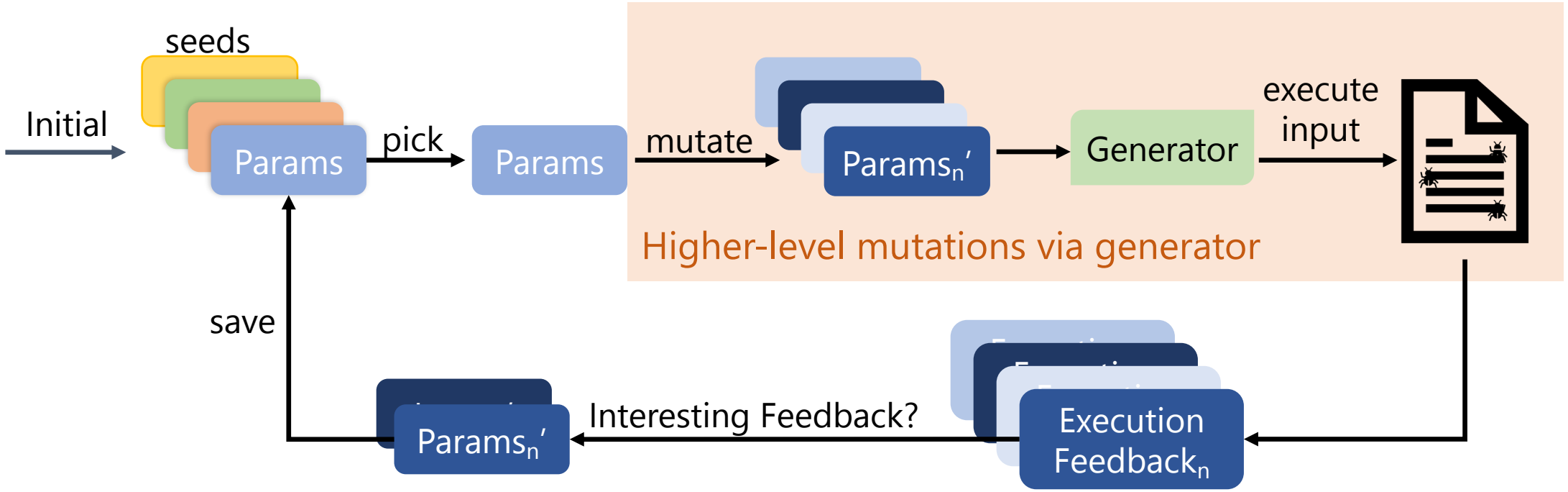
Zest: Integrate Generator + CGF



Zest: Integrate Generator + CGF



Zest: Integrate Generator + CGF

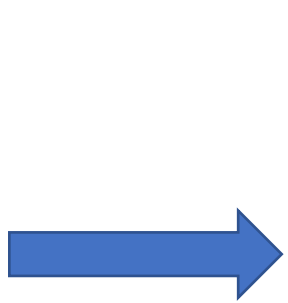


Zest finds complex semantic bugs

```
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!

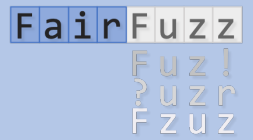


Google Closure Compiler

IllegalStateException in VarCheck
during optimization

Background on Fuzzing

Performance Bugs



FairFuzz

Lemieux & Sen. ASE ‘18

Zest

Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ‘19

Smart Generators

Future Directions

Background on Fuzzing

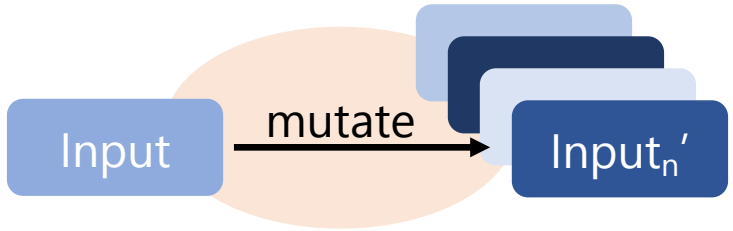
Performance Bugs

FairFuzz

Lemieux & Sen. ASE ‘18

Zest

Padhye, Lemieux, Sen, Papadakis & Le Traon. ISSTA ‘19



Structure-aware mutations
→ New depth of program exploration

Background on Fuzzing

Performance Bugs 

Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing

Performance Bugs

Exploring Core Logic

Smart Generators

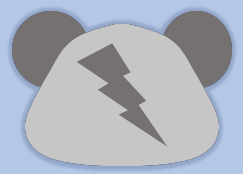
Future Directions

Background on Fuzzing

Performance Bugs

Exploring Core Logic

RLCheck
Reddy, Lemieux, Padhye & Sen. ICSE '20.



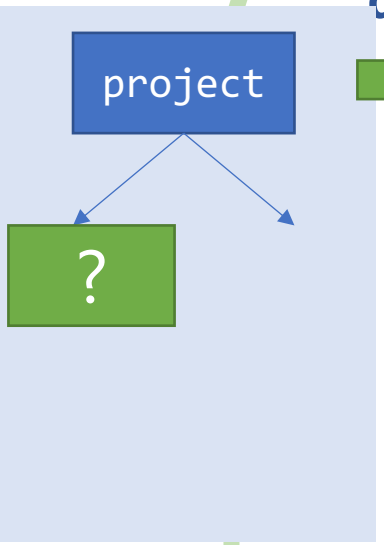
AutoPandas
Bavishi, Lemieux, Sen & Stoica. OOPSLA '19

Future Directions

Likelihood of Generating a Valid Maven File?

```
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
```

Likelihood of Generating a Valid Maven File?

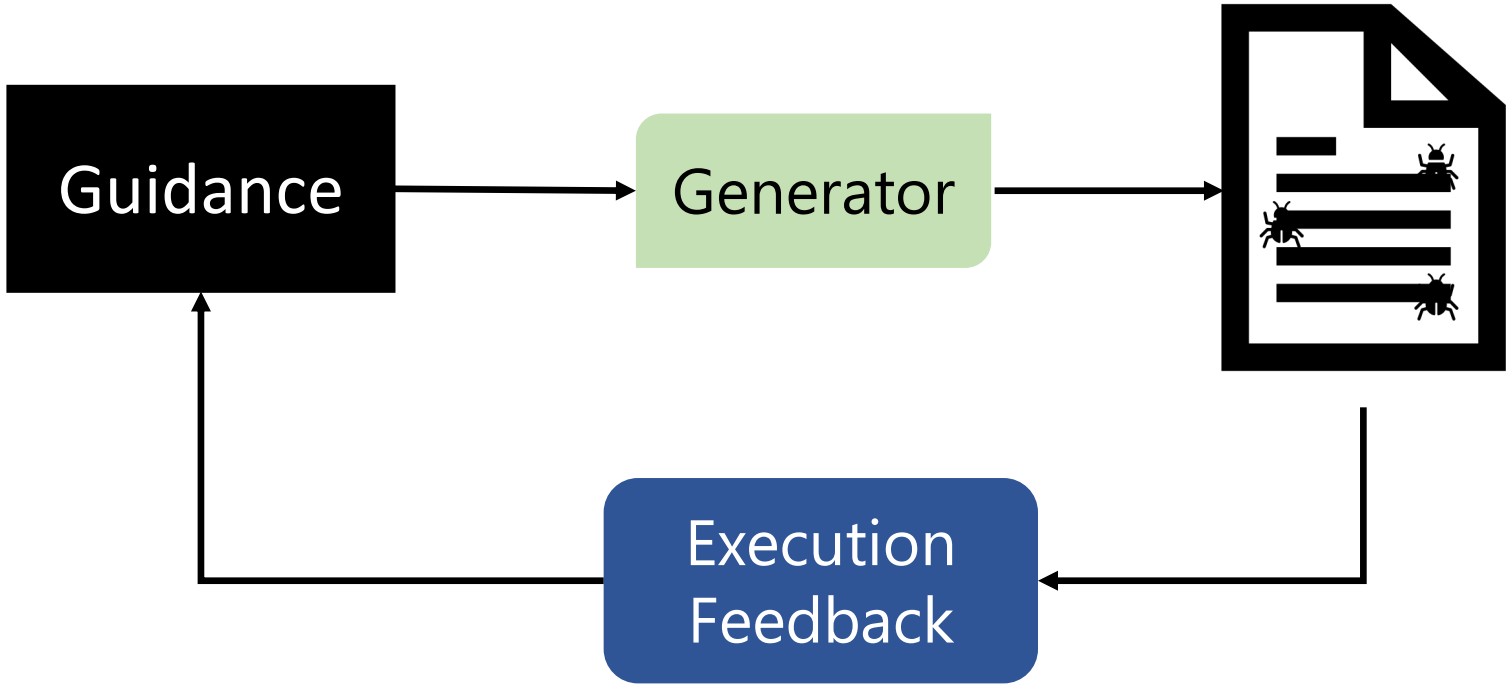



```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean():
        node.addText(random.nextString())
    return node
```

$$P(\text{valid}) \leq \frac{|subset_1| \times |subset_2| \times \cdots \times |subset_n|}{|\text{tags}|^n}$$

☹

RLCheck: Directly Control the Choices



Directly Control the *Choices*

```
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
```

Directly Control the *Choices*

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

What value to return to maximize the chance of generating a valid input?

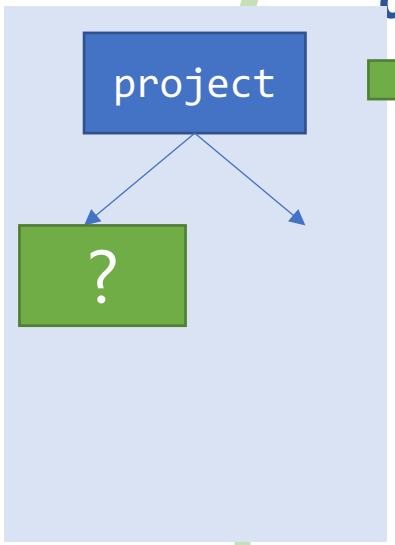
Directly Control the *Choices*

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

What value to return to maximize the chance of generating a valid input?

Depends on *context*

Different Context → Different "Good" Choices



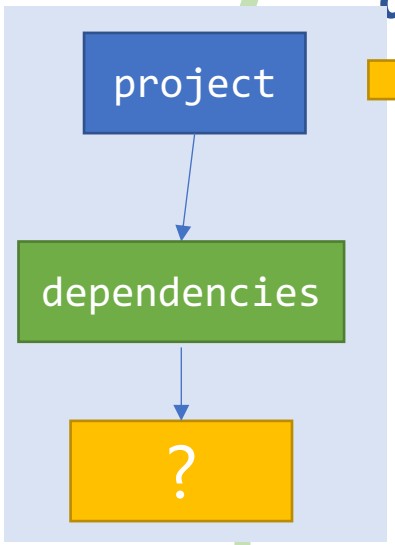

```
def genXML(random):
    tag = random.choice(tags)
    node = XMLElement(tag)
    num_child = random.nextInt(0, MAX_CHILDREN)
    for i in range(0, num_child):
        node.addChild(genXML(random))
    if random.nextBoolean( ):
        node.addText(random.nextString( ))
    return node
```

Different Context → Different "Good" Choices

?

```
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
```

Different Context → Different "Good" Choices




```
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
```

Different Context → Different "Good" Choices

```
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
```

Step 1: Add Context to Generator

```
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
```

Step 1: Add Context to Generator

```
def genXML(random, context):
    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, context))
    if random.nextBoolean( ):

        node.addText(random.nextString( ))
    return node
```

Step 1: Add Context to Generator

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

Step 2: Make Choices Based on Context

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

Step 2: Make Choices Based on Context

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

Our Problem Setting

context `["project", "2", "dependencies", ...]`

choice space

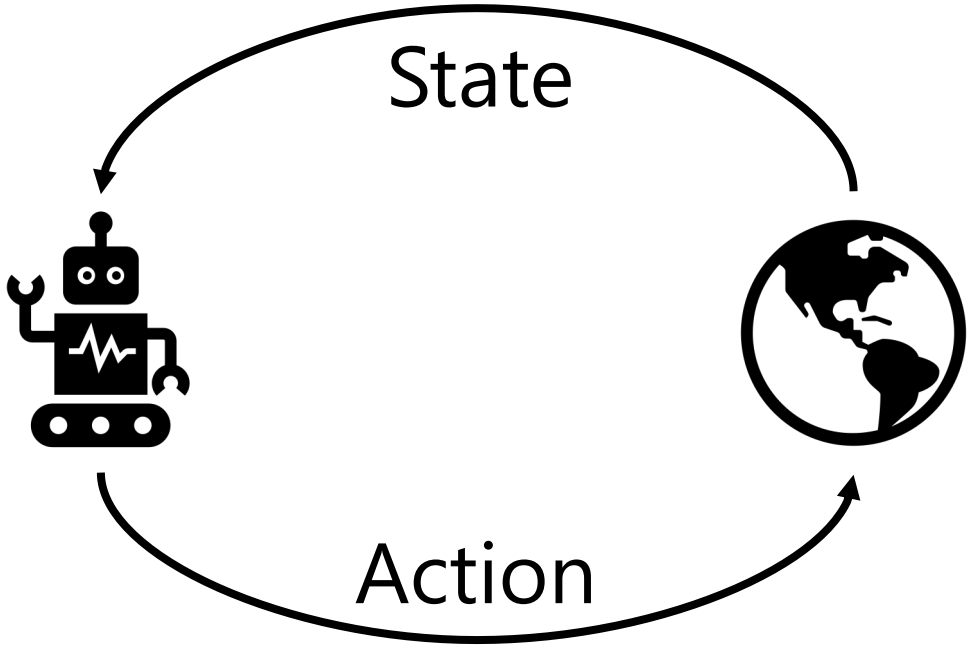
```
tag = random.choice(tags, context)
```

Our Problem Setting

State

Action

Sounds Like Reinforcement Learning

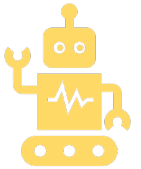
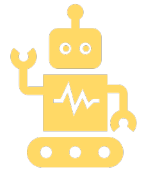


RLCheck Idea: RL Agent At Each Choice Point

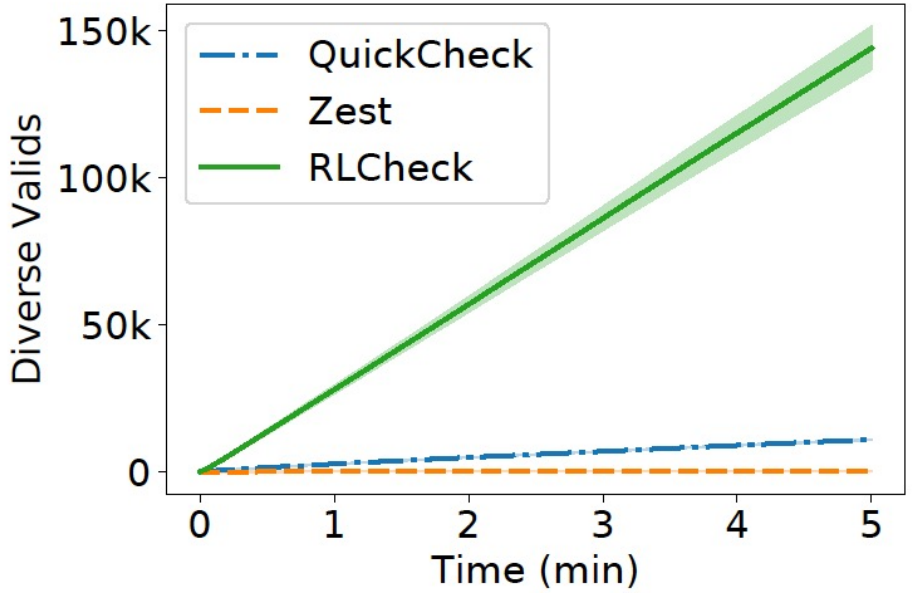
```
def genXML(random, context):
    tag = random.choice(tags, context)
    node = XMLElement(tag)
    context.append(tag)
    num_child = random.nextInt(0, MAX_CHILDREN, context)
    context.append(num_child)
    for i in range(0, num_child):
        node.addChild(genXML(random, context))
    if random.nextBoolean(context):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

RLCheck Idea: RL Agent At Each Choice Point

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

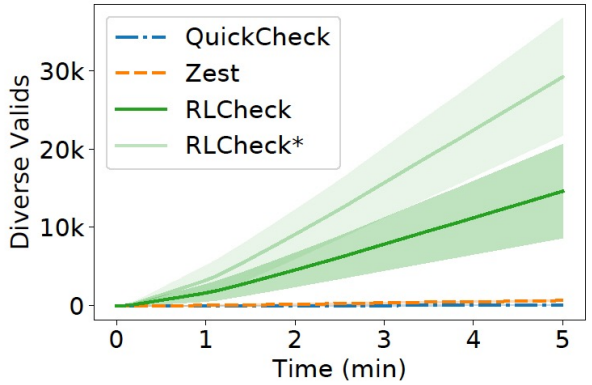



RLCheck: Many More Unique Valid Inputs

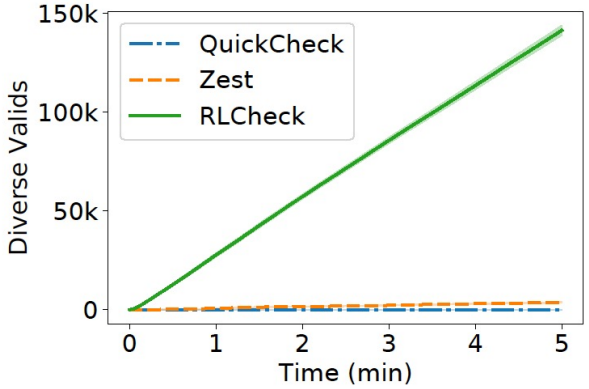


Closure Compiler (JS)

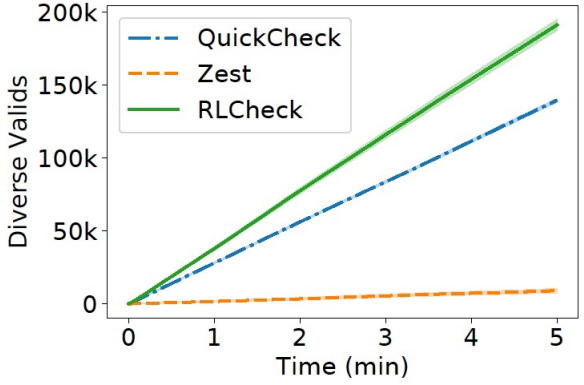
RLCheck: Many More Unique Valid Inputs



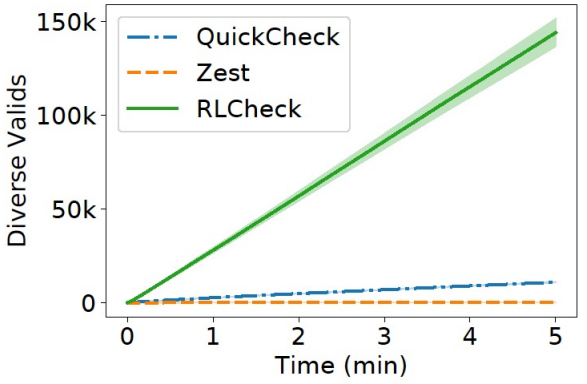
Ant (XML)



Maven (XML)



Rhino Compiler (JS)



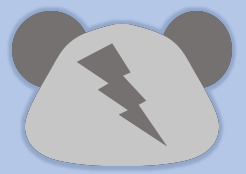
Closure Compiler (JS)

Background on Fuzzing

Performance Bugs 

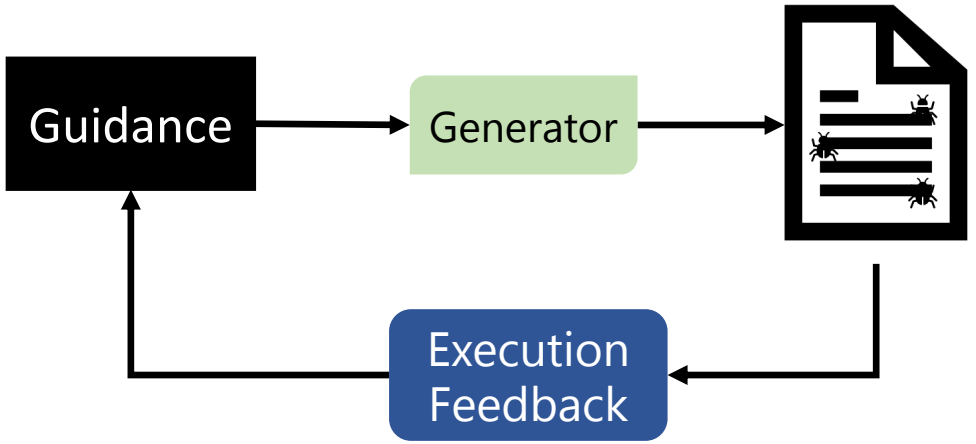
Exploring Core Logic 

RLCheck
Reddy, Lemieux, Padhye & Sen. ICSE '20.



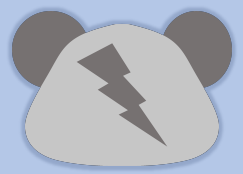
AutoPandas
Bavishi, Lemieux, Sen & Stoica. OOPSLA '19

Future Directions



Separate distribution from user-facing generator
→ faster fuzzing, new synthesis paradigm

RLCheck
Reddy, **Lemieux**, Padhye & Sen. ICSE '20.



AutoPandas
Bavishi, **Lemieux**, Sen & Stoica. OOPSLA '19

Future Directions

Background on Fuzzing

Performance Bugs 

Exploring Core Logic 

Smart Generators 

Future Directions

Background on Fuzzing

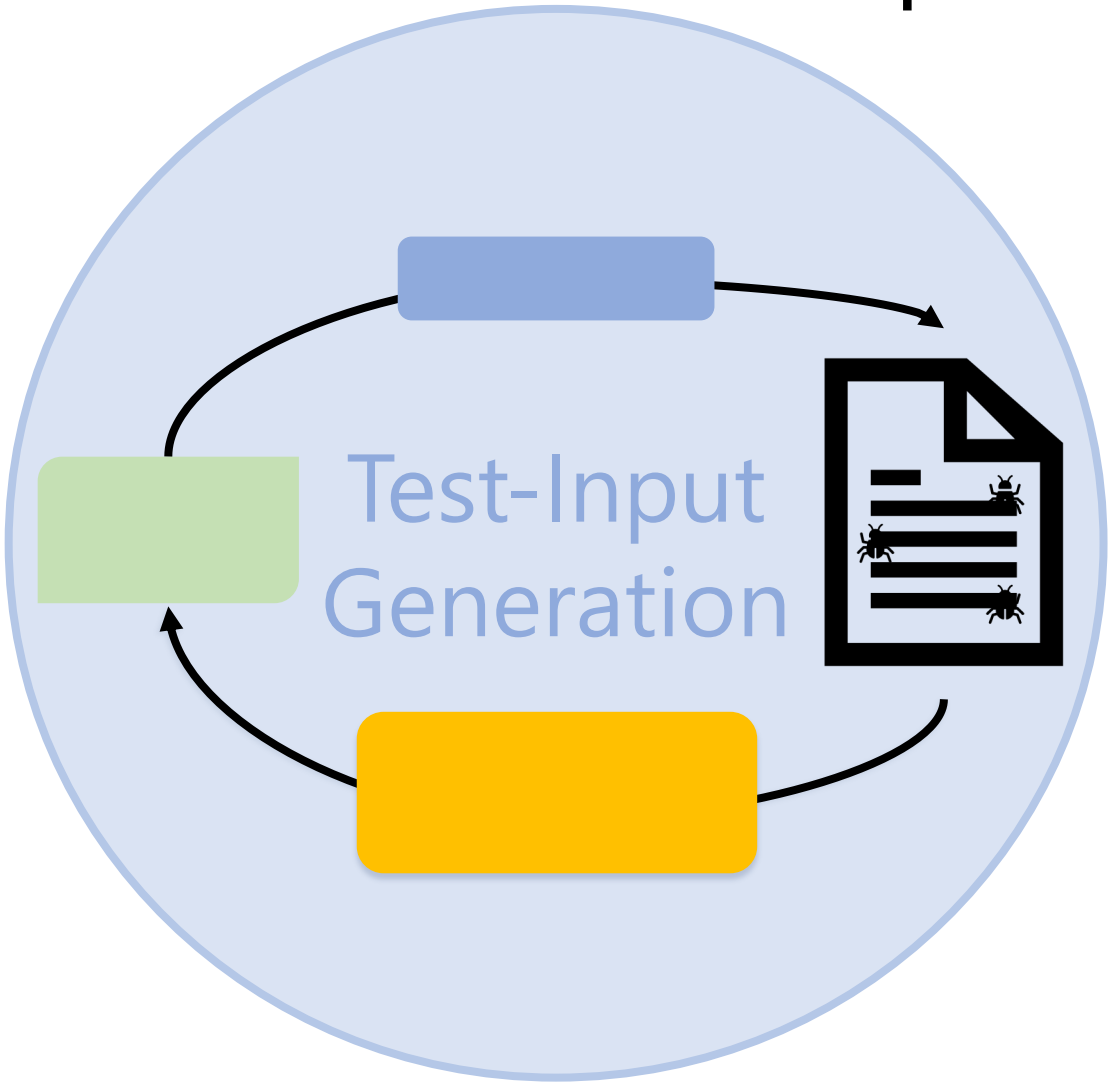
Performance Bugs 

Exploring Core Logic 

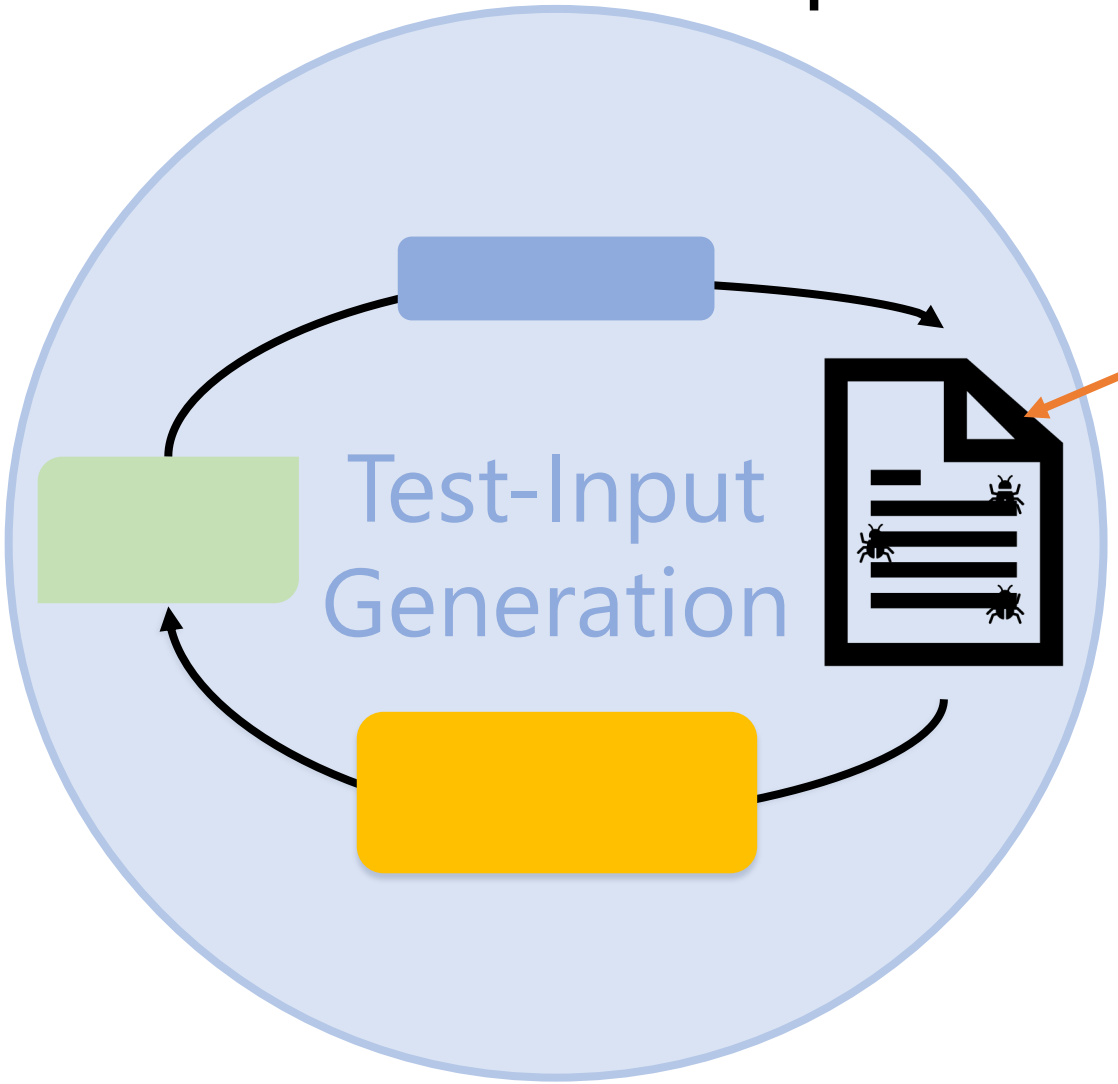
Smart Generators 

Future Directions

So Far: Innovations in Test-Input Generation

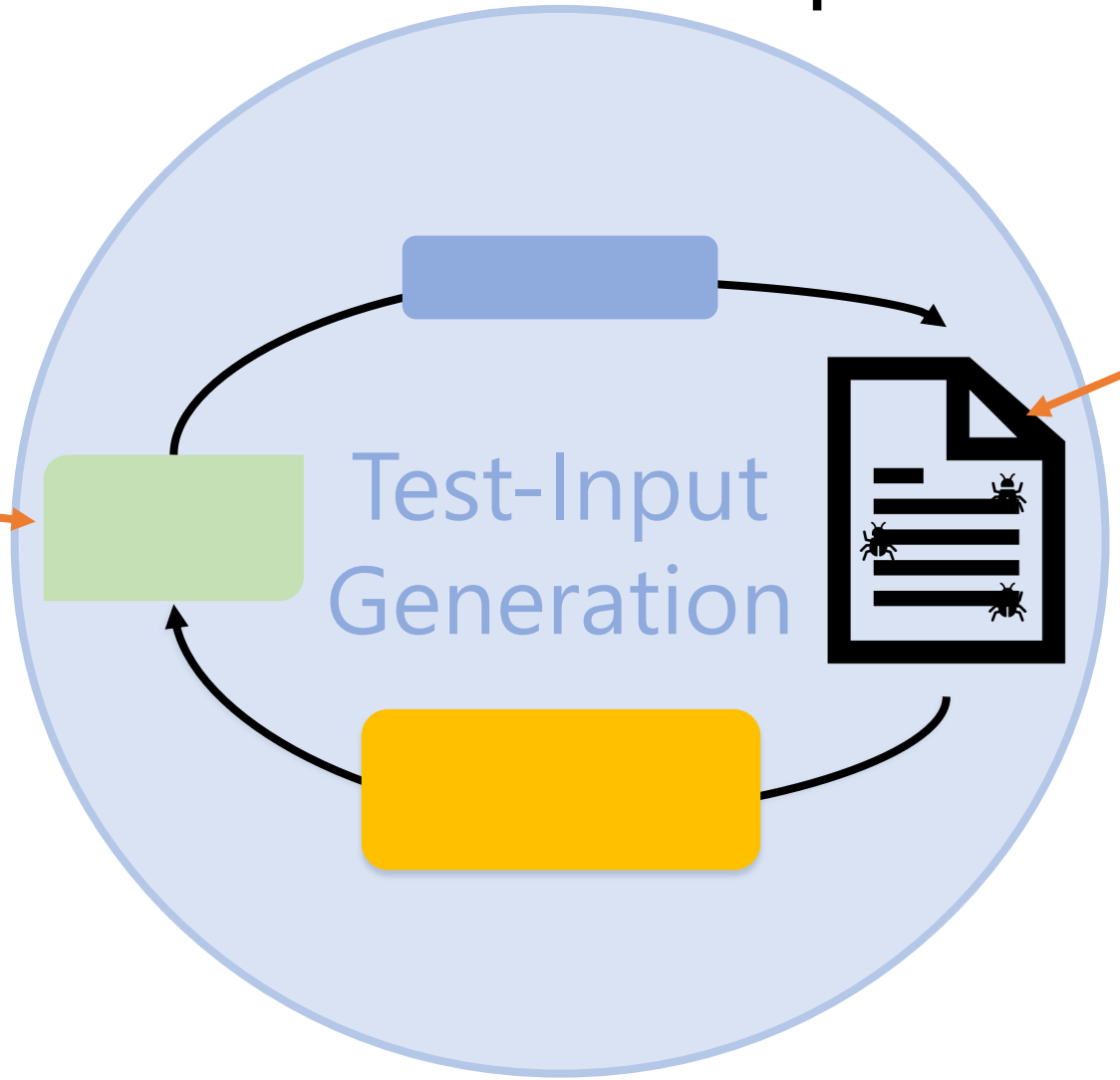


Problems Around Test-Input Generation



How do I write a good test driver?

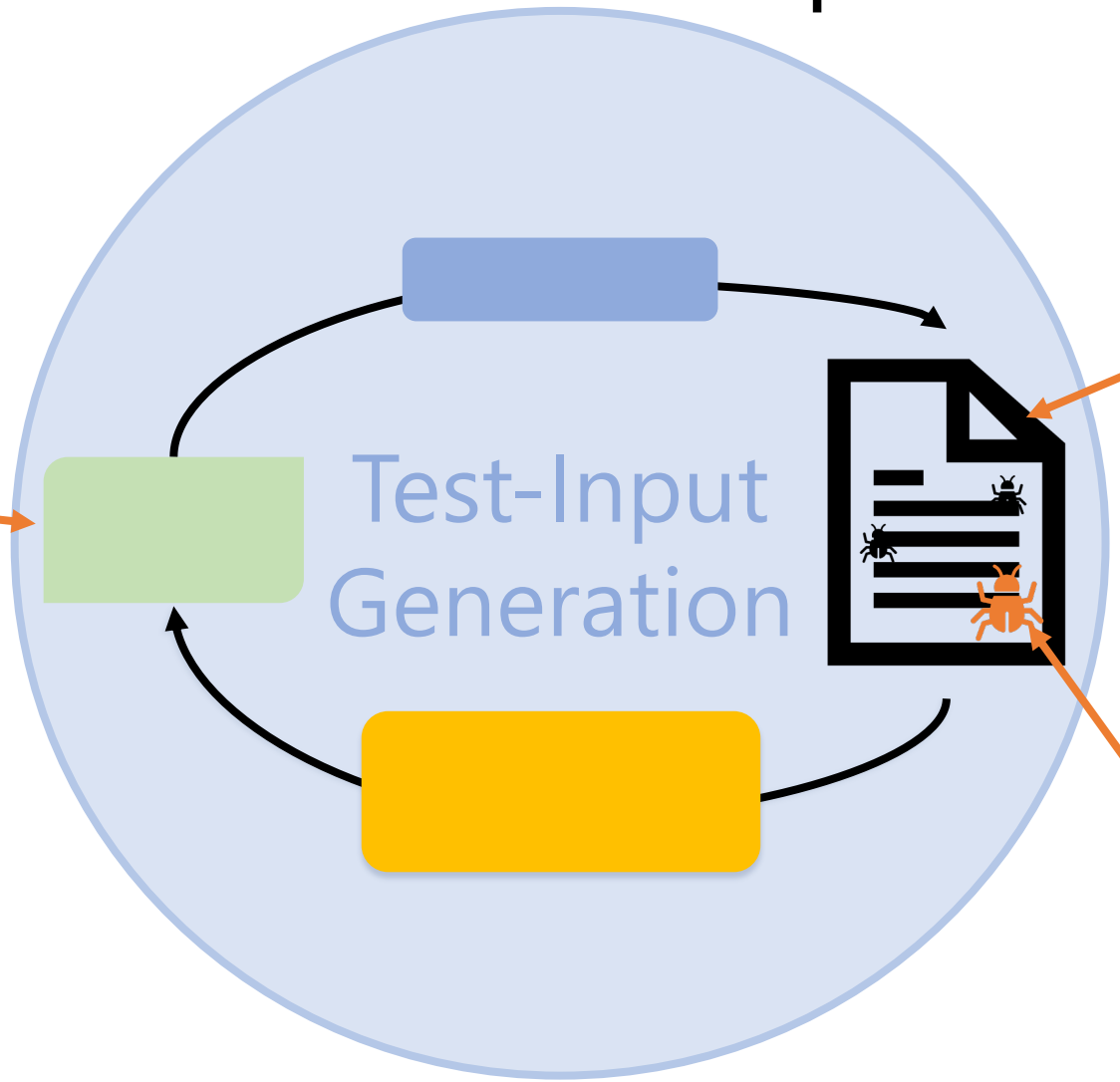
Problems Around Test-Input Generation



How do I specify the structure of my inputs?

How do I write a good test driver?

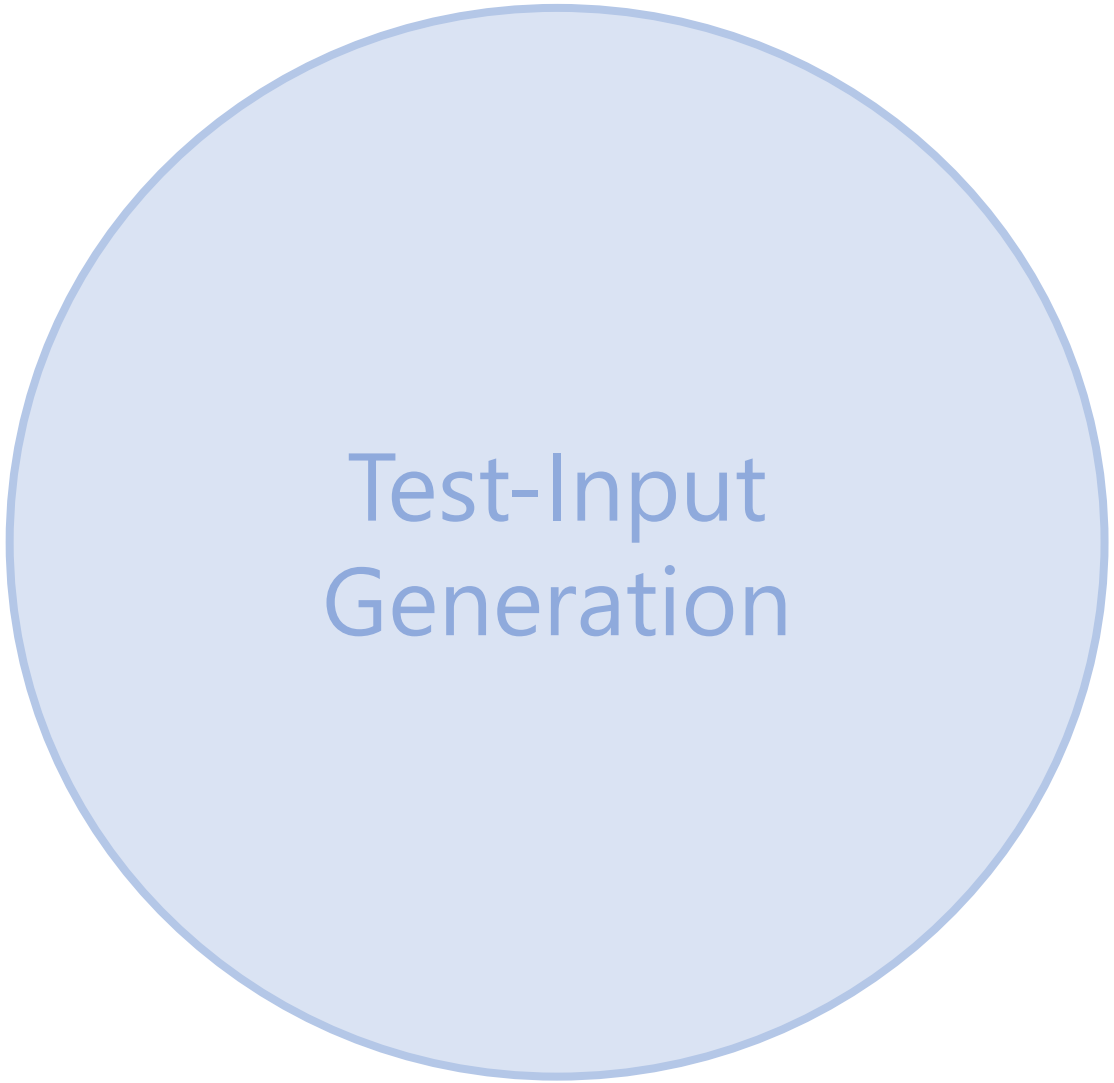
Problems Around Test-Input Generation

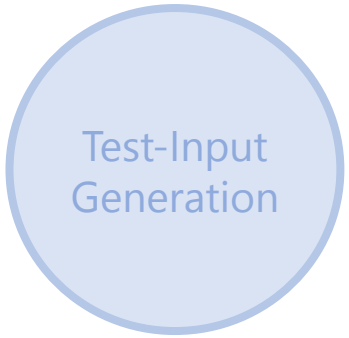


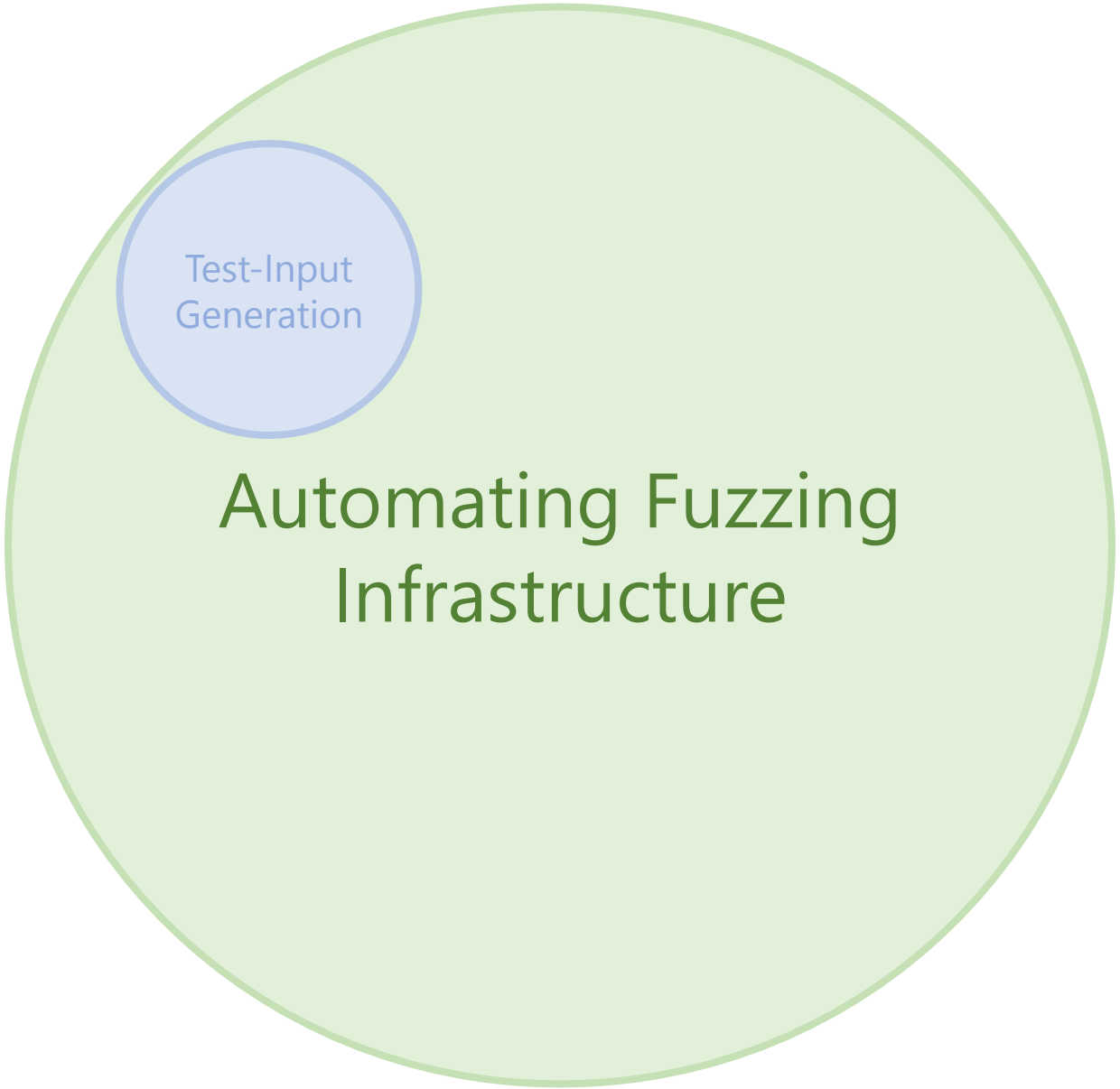
How do I specify the structure of my inputs?

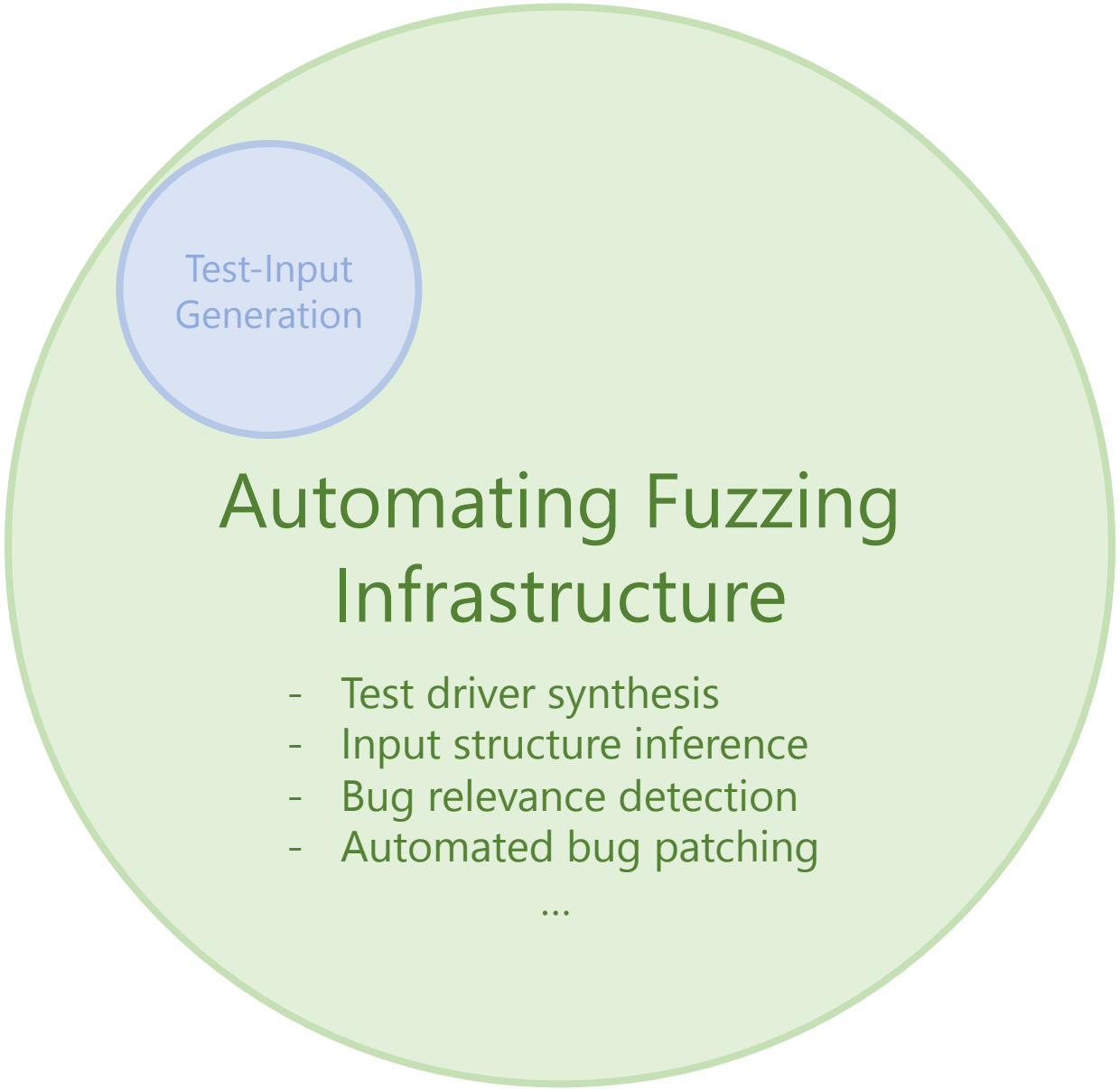
How do I write a good test driver?

Do I really care about this bug?


Test-Input
Generation

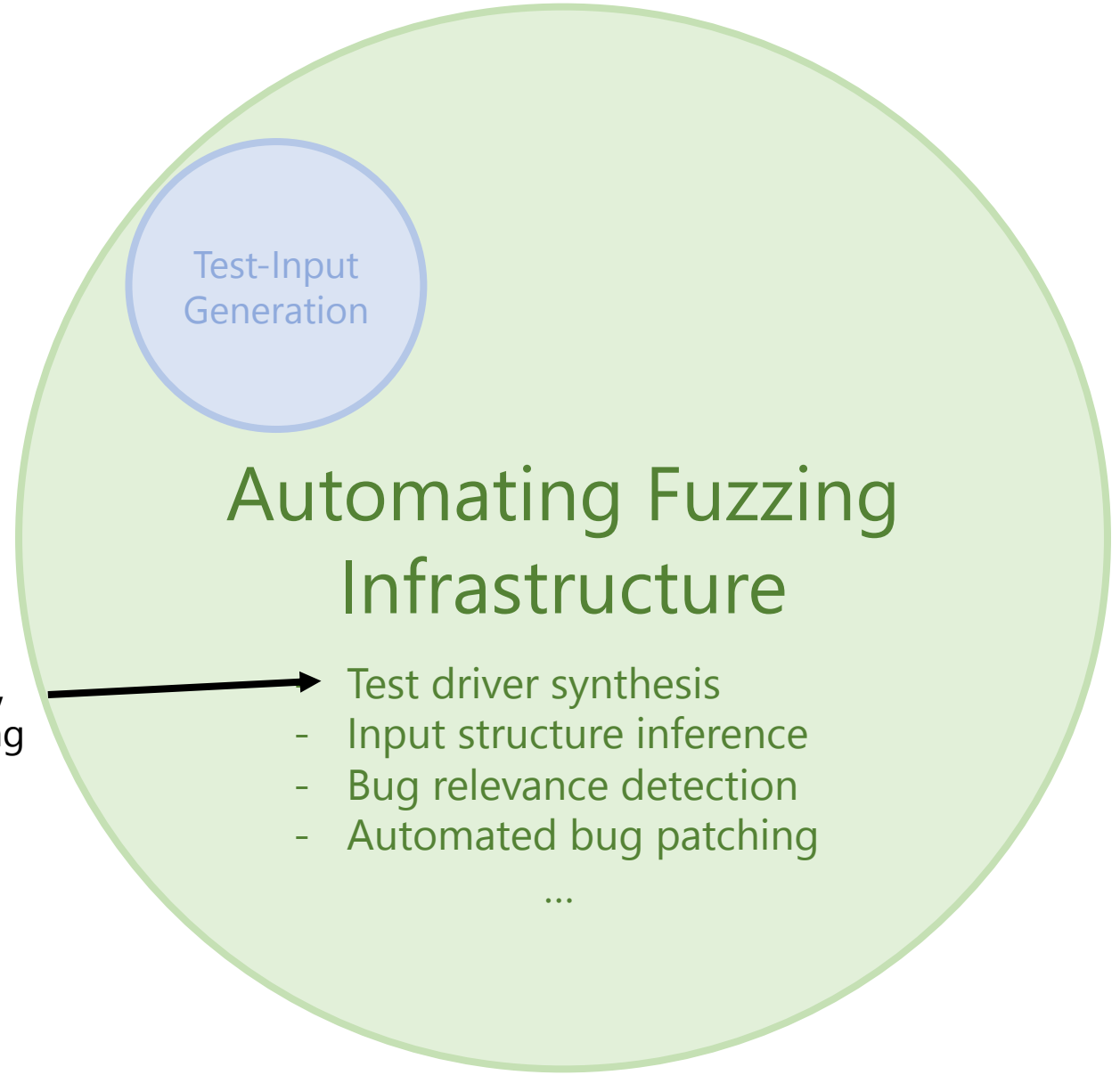

Test-Input
Generation


Test-Input Generation
Automating Fuzzing Infrastructure


Test-Input Generation
Automating Fuzzing Infrastructure
- Test driver synthesis
- Input structure inference
- Bug relevance detection
- Automated bug patching
...

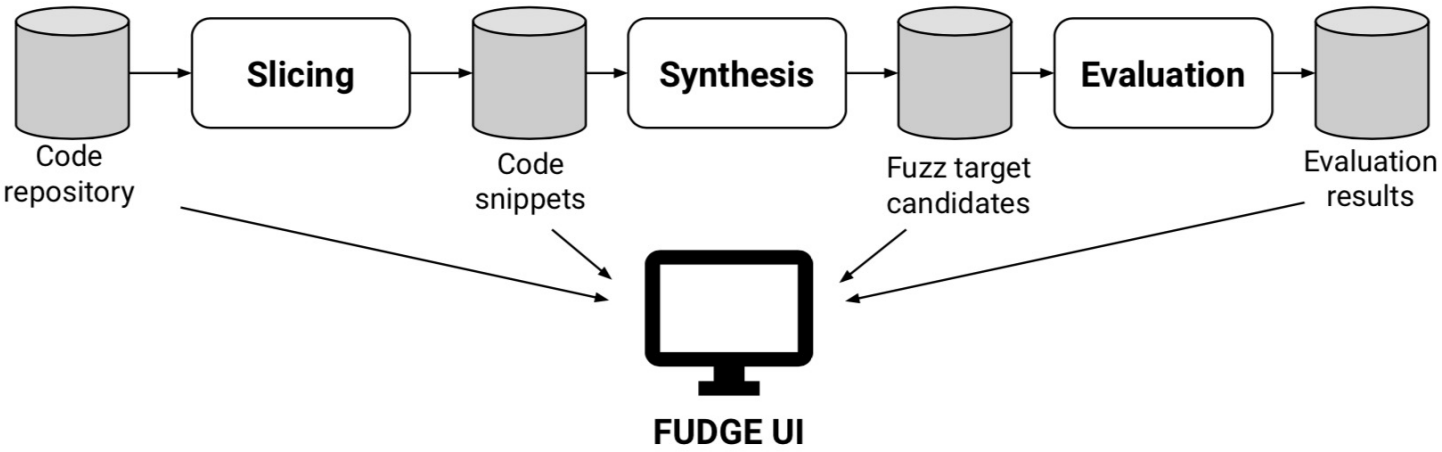
FUDGE

Babic, Bucur, Chen, Ivancic, King, Kusano, **Lemieux**, Szekeres, Wang
ESEC/FSE'19 **(Industry Track)**

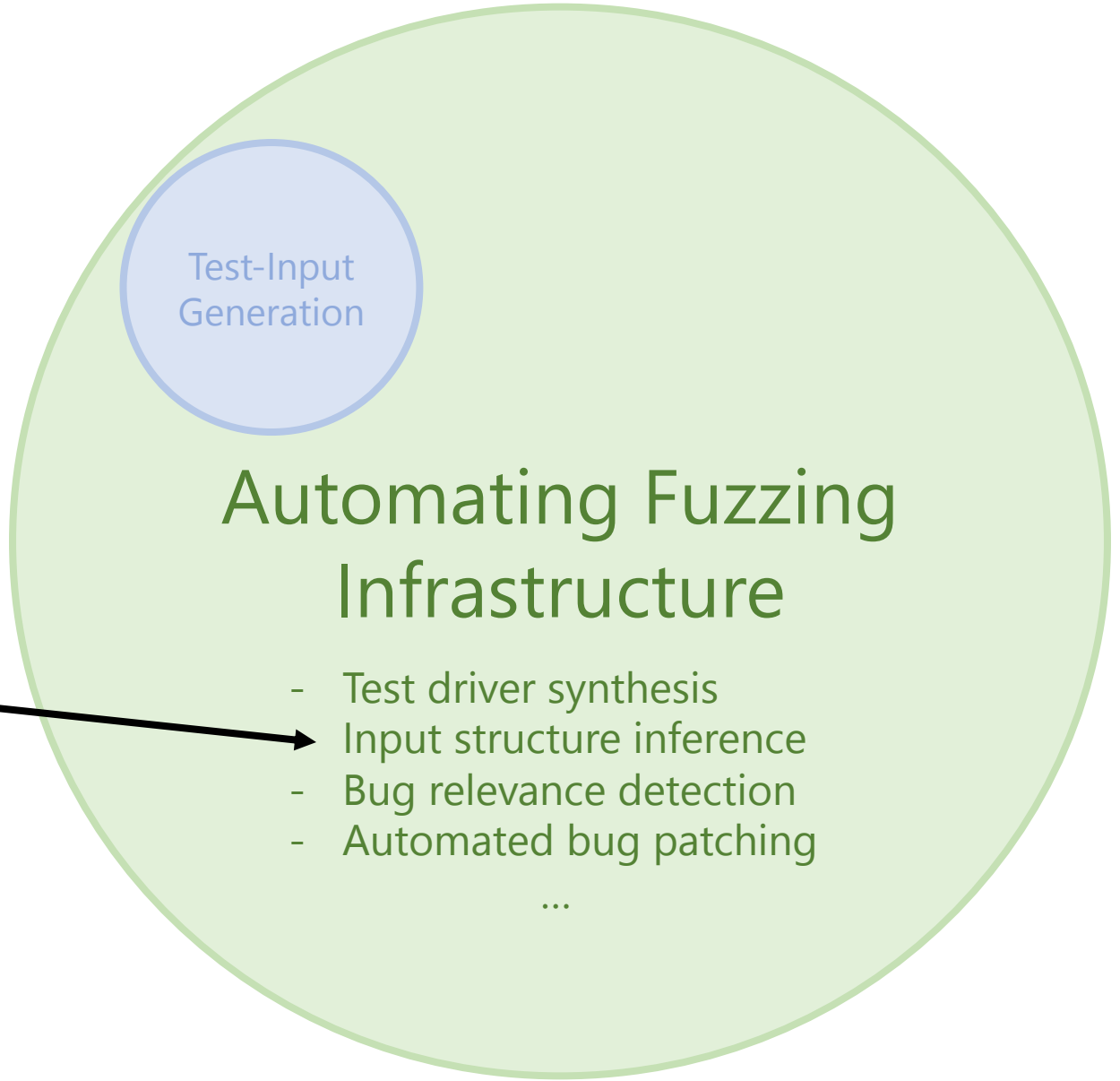


FUDGE

Babic, Bucur, Chen, Ivancic, King, Kusano, **Lemieux**, Szekeres, Wang
ESEC/FSE'19 **(Industry Track)**

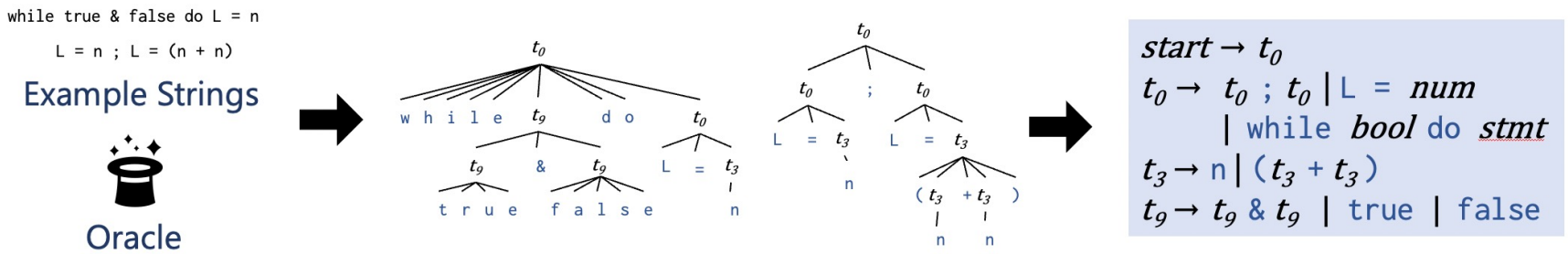


200 drivers integrated into open-source projects
→150 security-improving fixes



Arvada

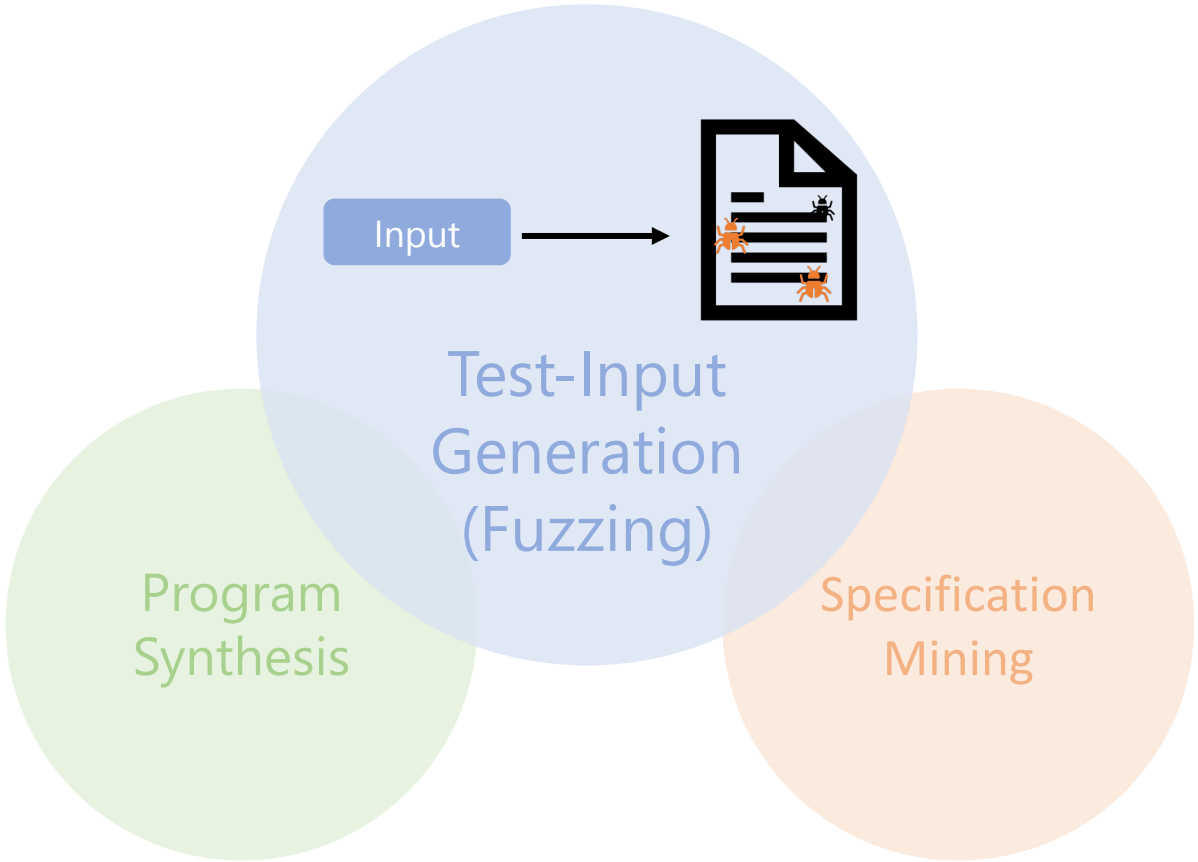
Kulkarni*, **Lemieux***, Sen
ASE'21

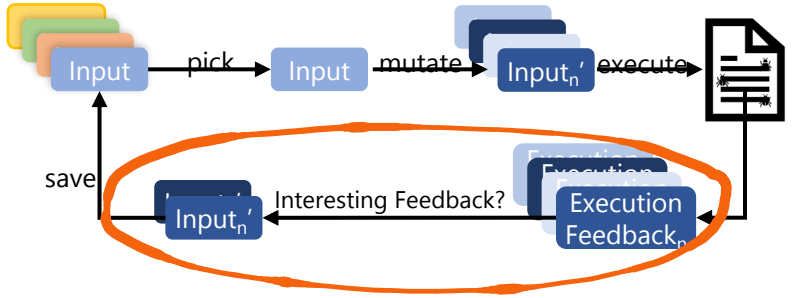


Arvada

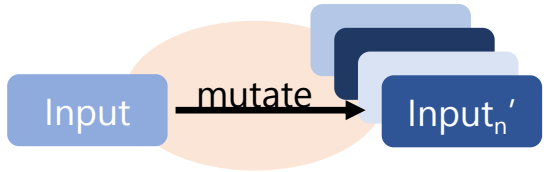
Kulkarni*, **Lemieux***, Sen
ASE'21

5x higher recall that SOTA ☺
1.27x slowdown ☹

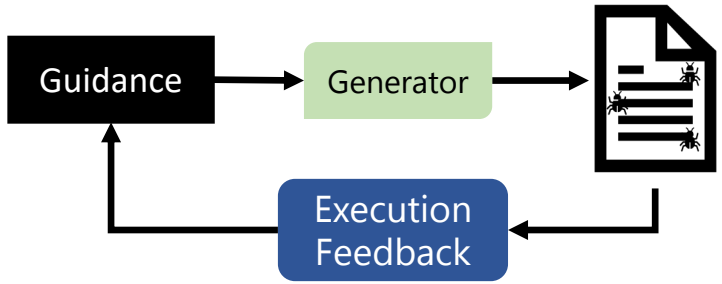




Generalize "feedback" with map-based abstraction
→ new bug domains



Structure-aware mutations
→ New depth of program exploration



Separate distribution from user-facing generator
→ faster fuzzing, new synthesis paradigm

https://www.carolemieux.com
clemieux@cs.ubc.ca
@cestlemieux