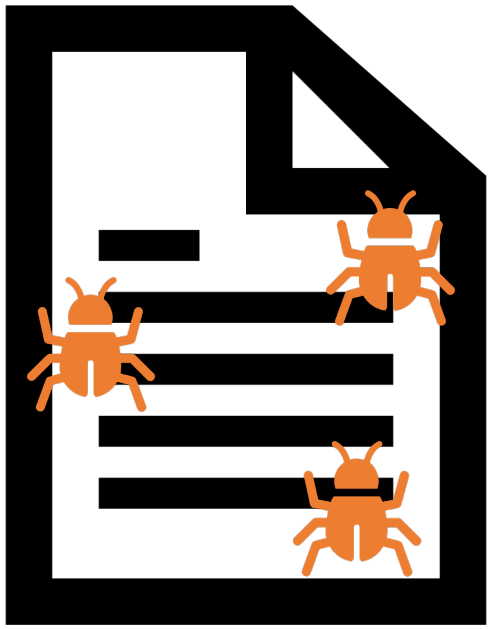
RLCheck: Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning

Sameer Reddy, **Caroline Lemieux**, Rohan Padhye, Koushik Sen

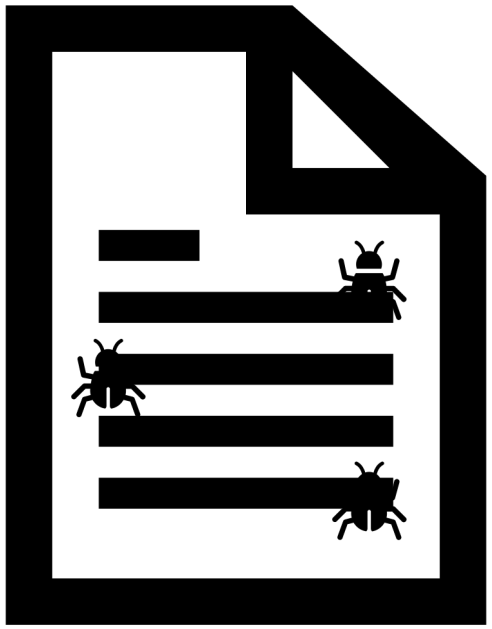
UC Berkeley

Presentation on July 8th, 2020 at ICSE 2020

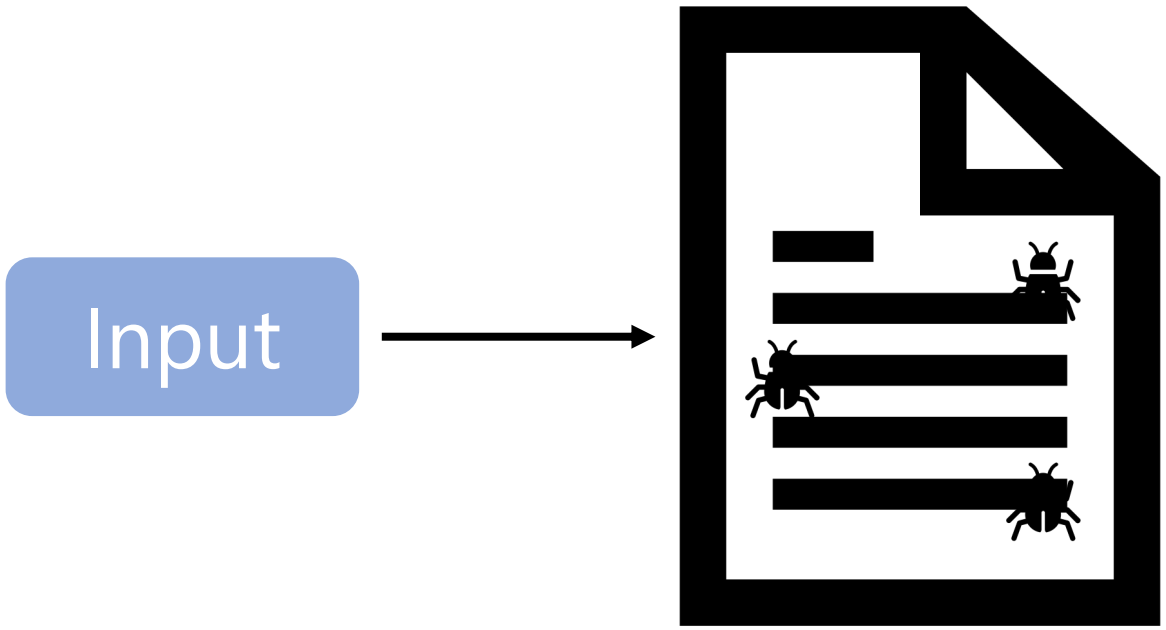
Programs Have Bugs



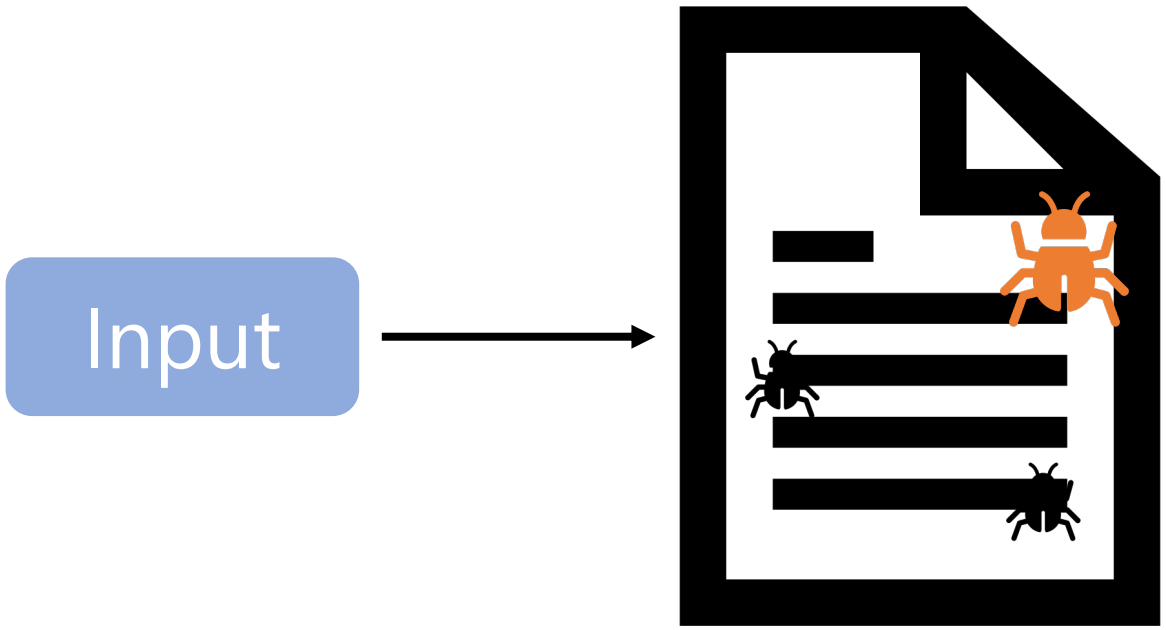
Bugs Can Be Hard to Find



Which Inputs Reveal Bugs?



Which Inputs Reveal Bugs?



Reveal Bugs?

Releasing jsfunfuzz and DOMFuzz

Tuesday, July 28th, 2015

Today I'm releasing two fuzzers: jsfunfuzz, which tests JavaScript engines, and DOMFuzz, which tests layout and DOM APIs.

Over the last 11 years, these fuzzers have found 6450 Firefox bugs, including 790 bugs that were rated as security-critical.

What is Microsoft Security Risk Detection?

Security Risk Detection is Microsoft's unique fuzz testing service for finding security critical bugs in software. Security Risk Detection helps customers quickly adopt practices and technology battle-tested over the last 15 years at Microsoft.

Google Testing Blog

Announcing OSS-Fuzz: Continuous Fuzzing for Open Source Software

Thursday, December 01, 2016

Linux 4.14-rc5

The other thing perhaps worth mentioning is how much random fuzzing people are doing, and it's finding things. We've always done fuzzing (who remembers the old "crashme" program that just generated random code and jumped to it? We used to do that quite actively very early on), but people have been doing some nice targeted fuzzing of driver subsystems etc, and there's been various fixes (not just this last week either) coming out of those efforts. Very nice to see.

Background

Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion

Background

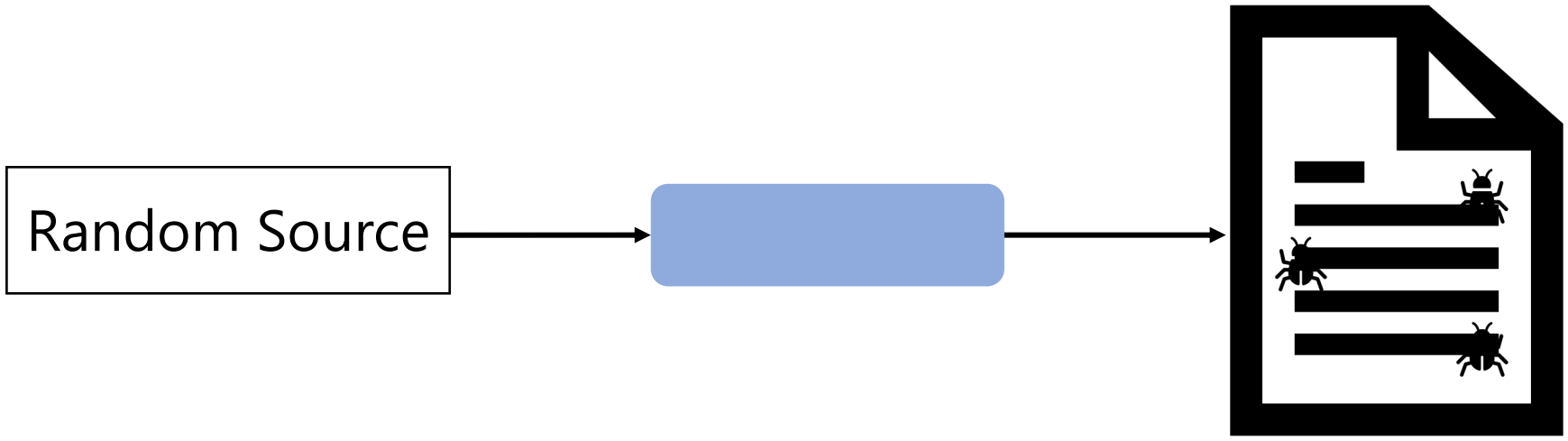
Generator-Based Fuzzing

The Fundamental Tradeoff

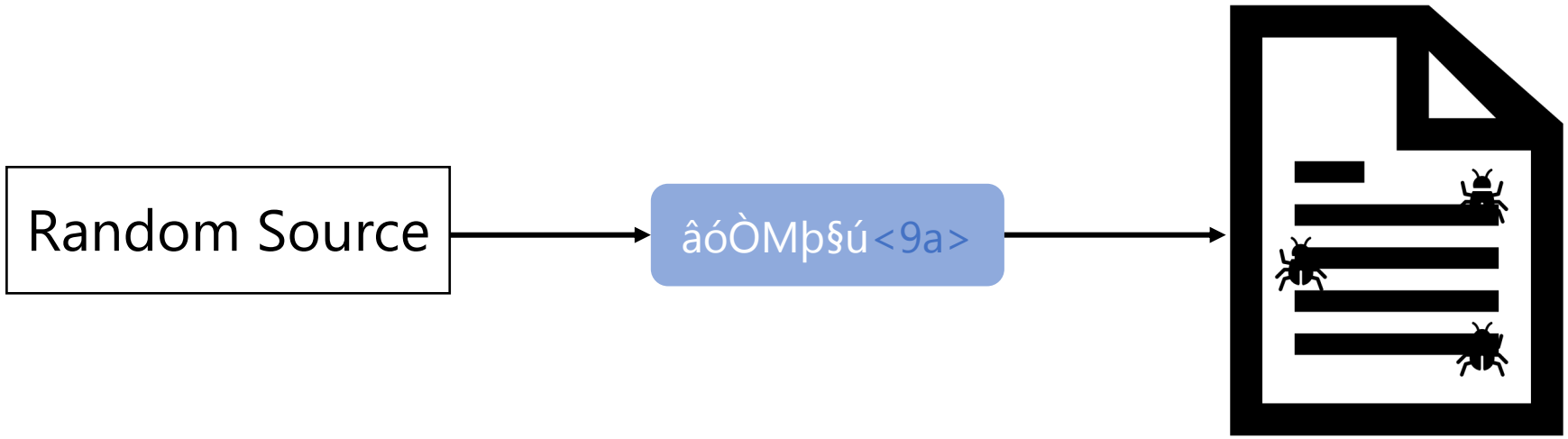
RLCheck: A Solution

Conclusion

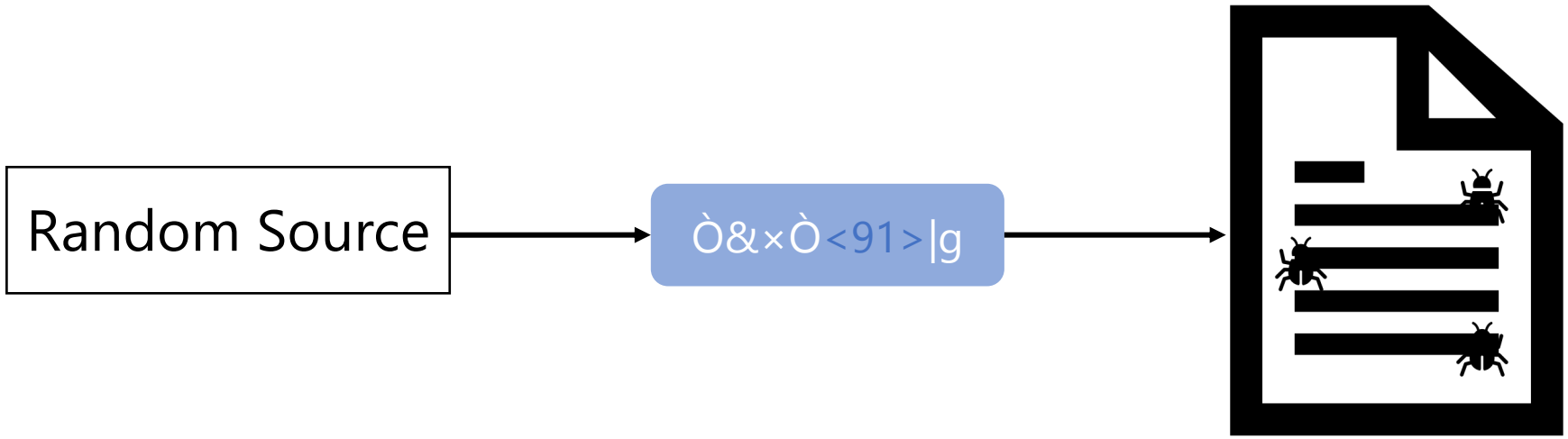
The Birth of Fuzzing



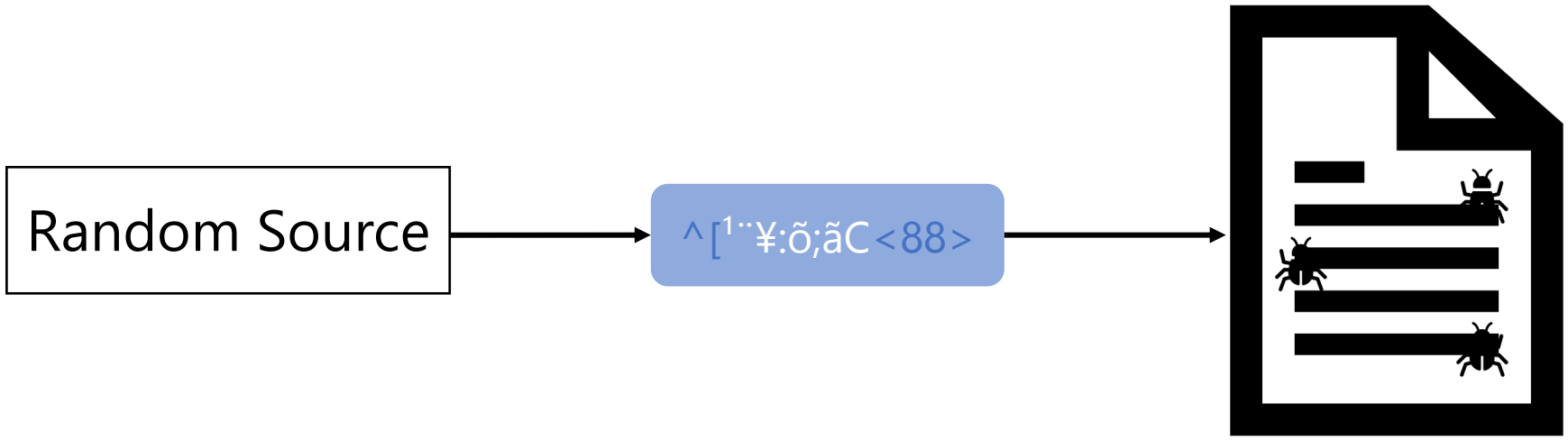
The Birth of Fuzzing



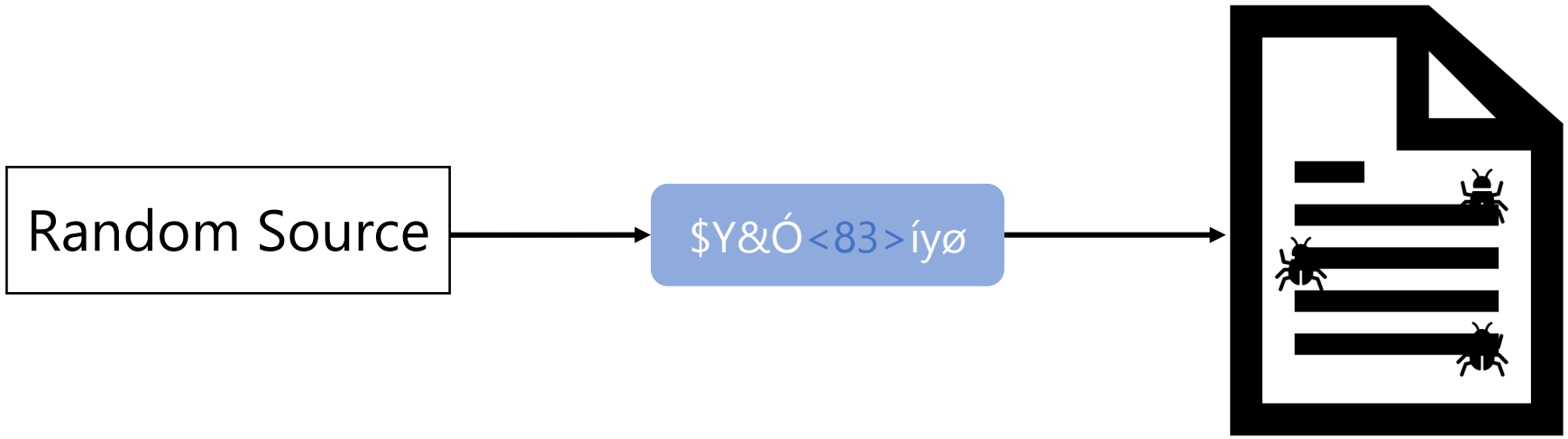
The Birth of Fuzzing



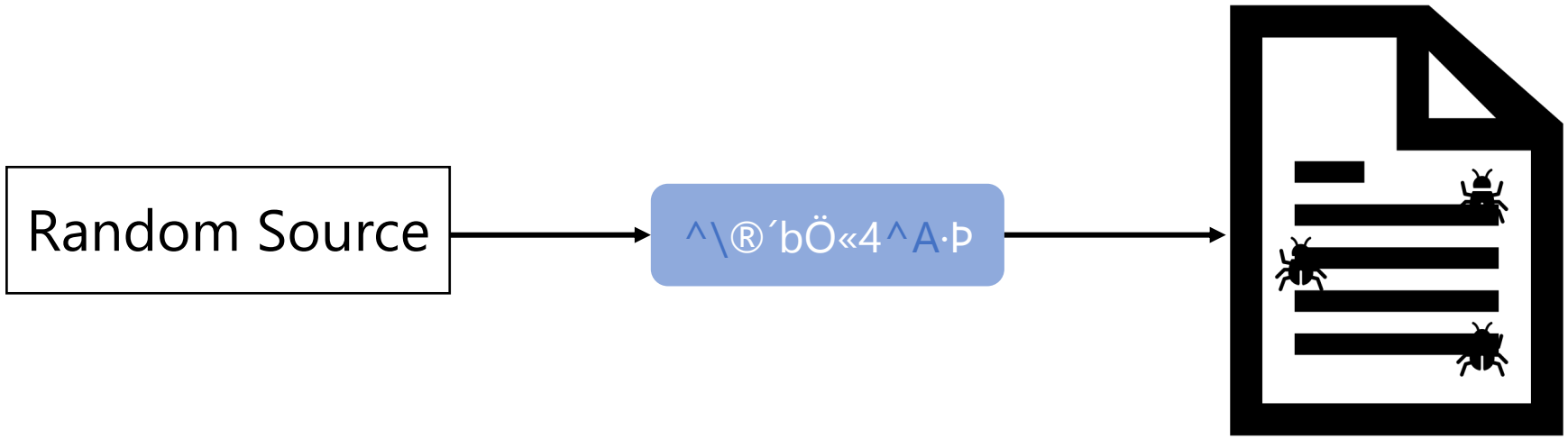
The Birth of Fuzzing



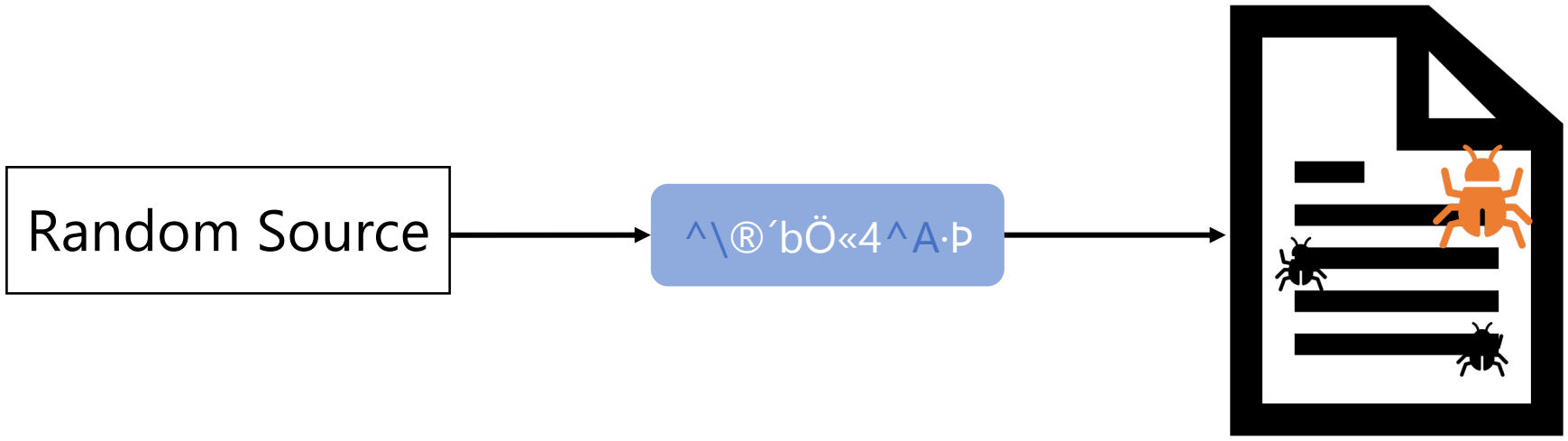
The Birth of Fuzzing



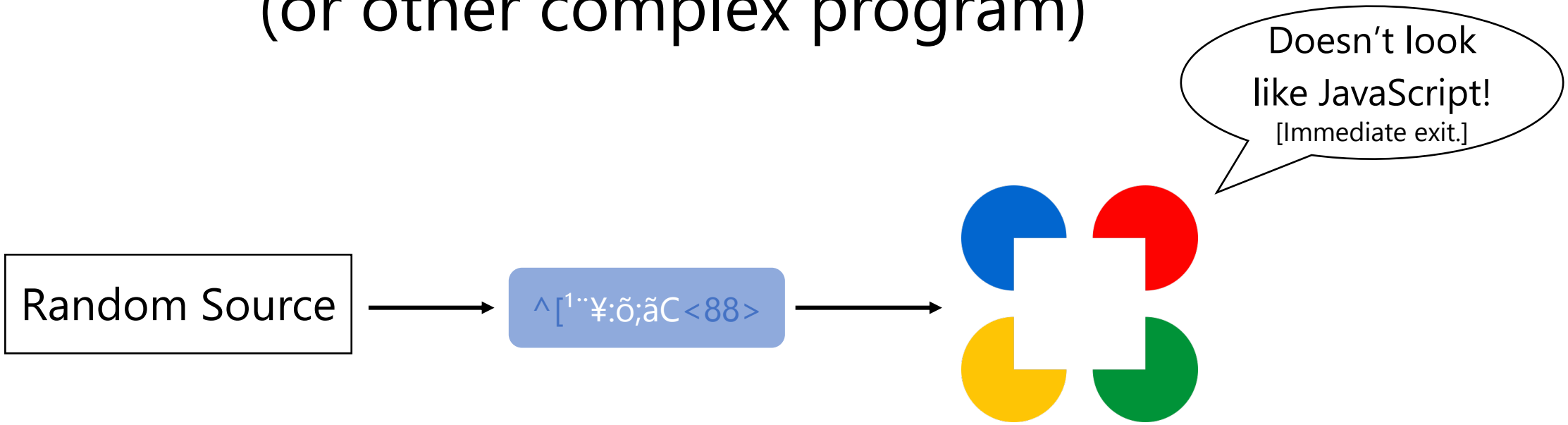
The Birth of Fuzzing



The Birth of Fuzzing

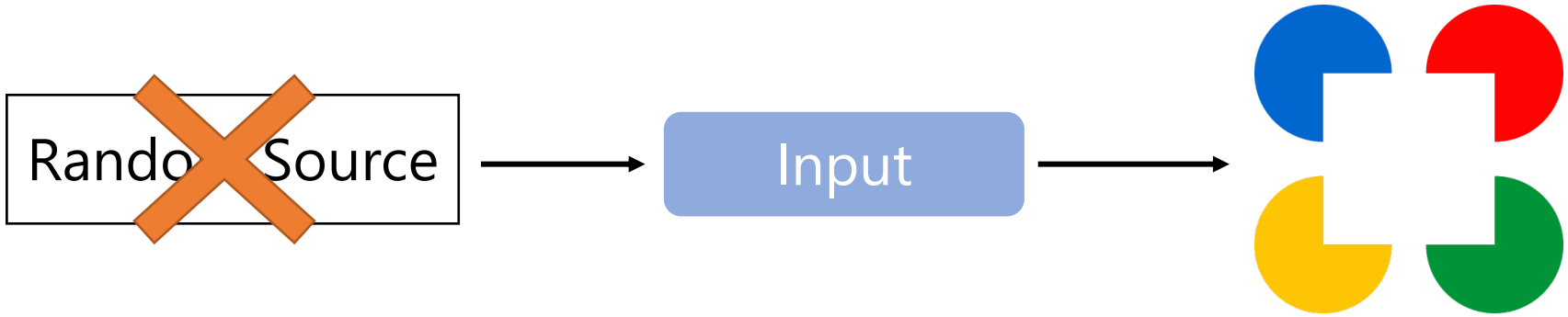


I Want to Fuzz a Compiler!
(or other complex program)



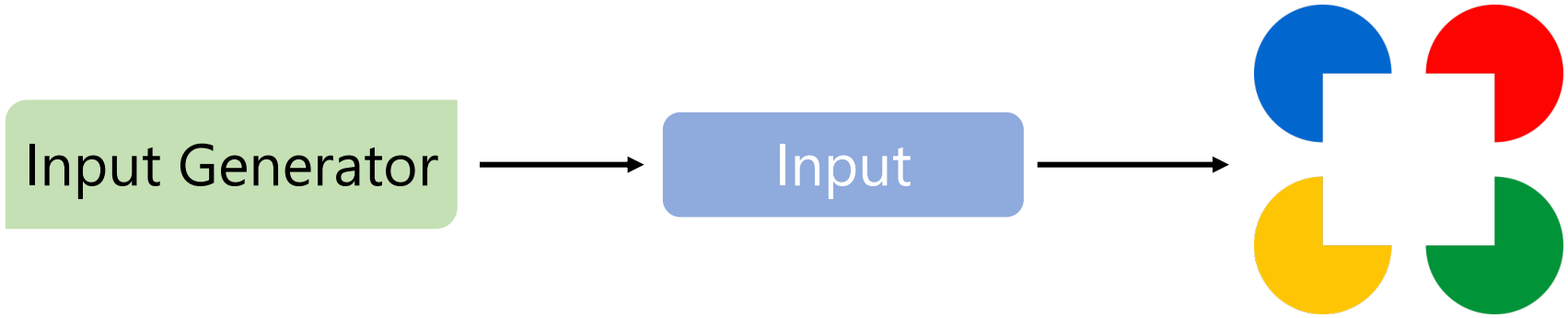
Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing



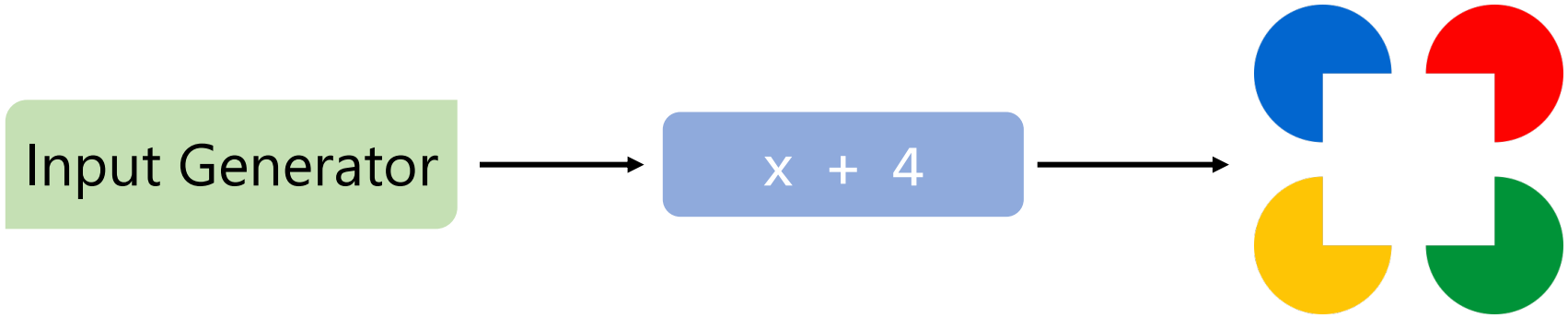
Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing



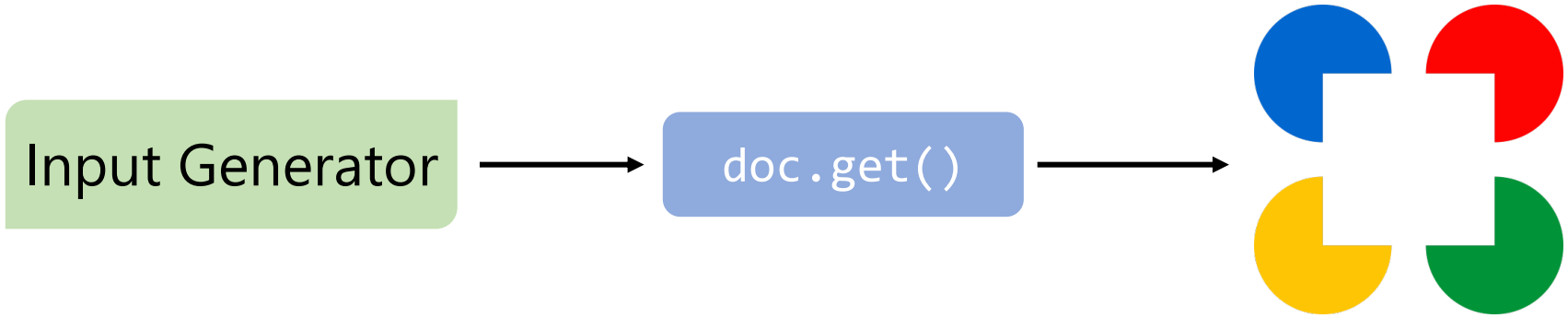
Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing



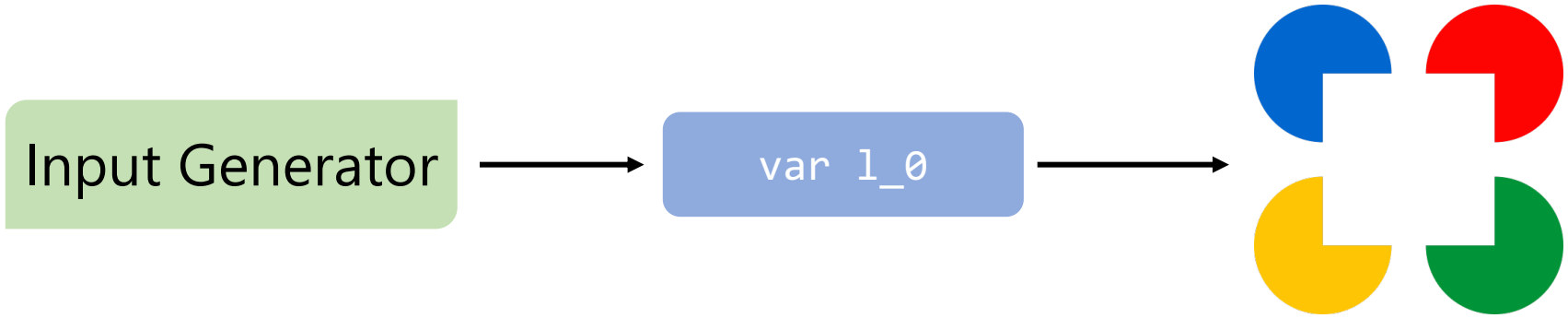
Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing



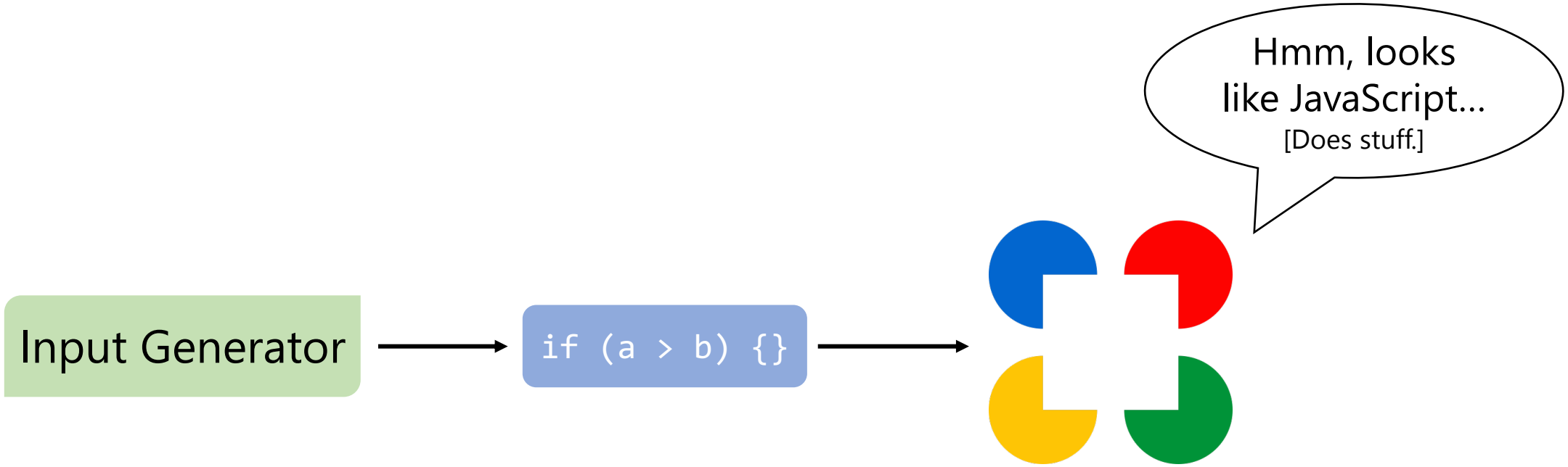
Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing



Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

Generator-Based Fuzzing

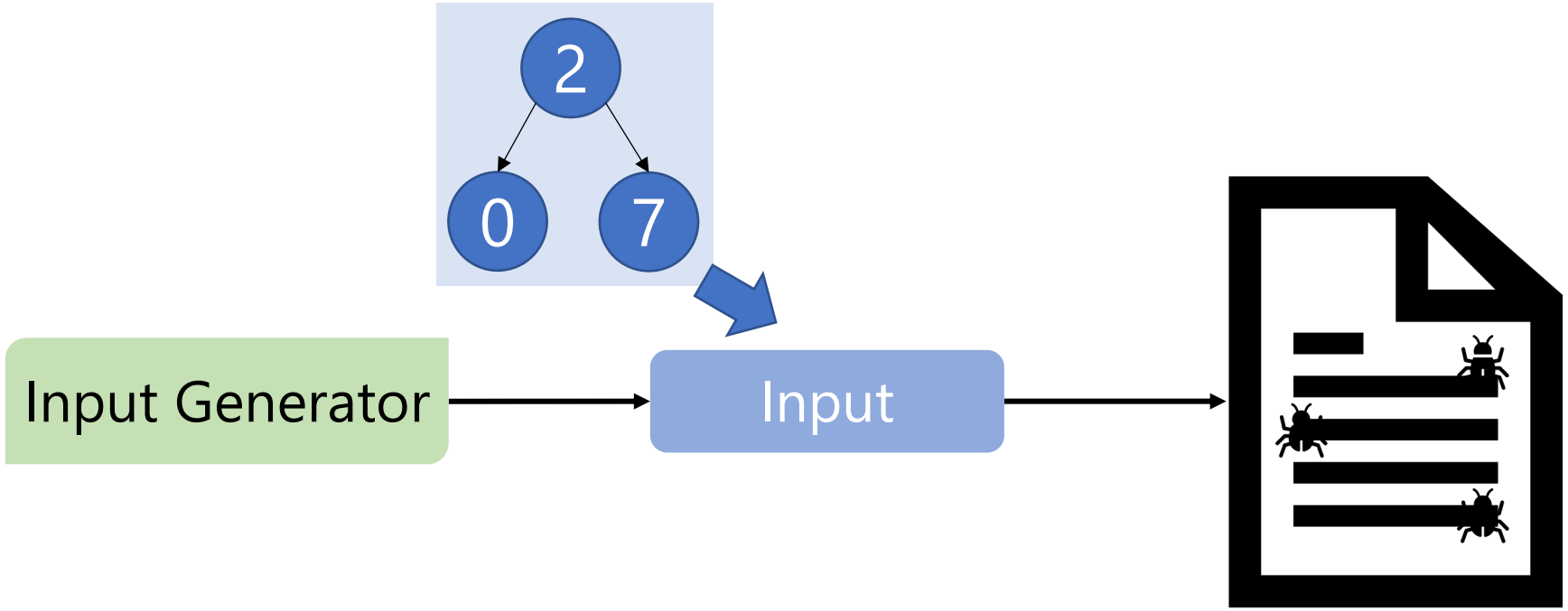


Logo CC-by-SA 4.0: https://en.wikipedia.org/wiki/File:Closure_logo.svg

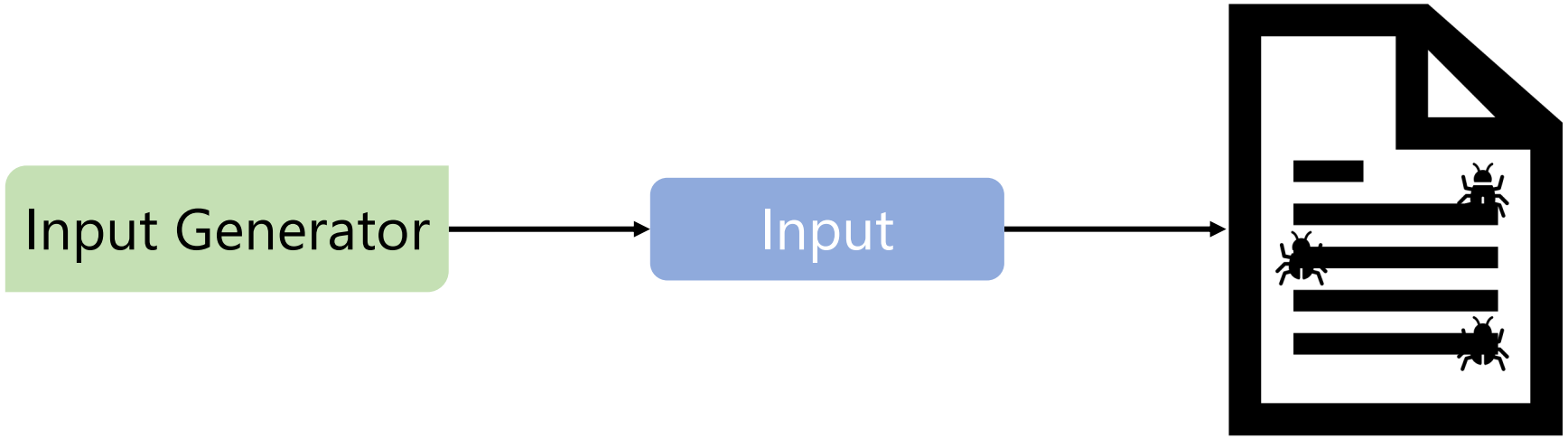
In-Depth Example

For ease of explanation, let's consider a simple program

Binary Tree Example



Binary Tree Example



Binary Tree Example: Generator

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Binary Tree Example: Generator

?

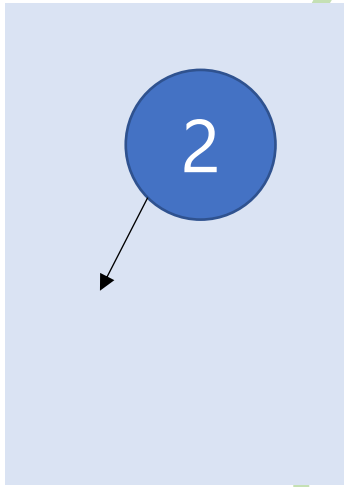
```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Binary Tree Example: Generator

2

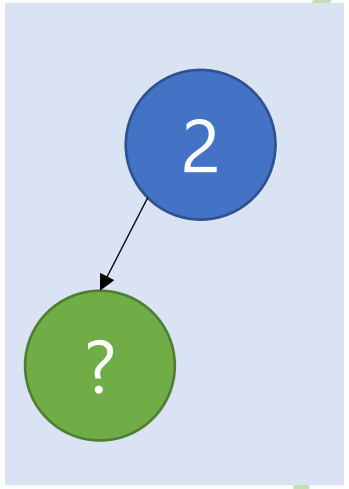
```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Binary Tree Example: Generator




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
      node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Binary Tree Example: Generator

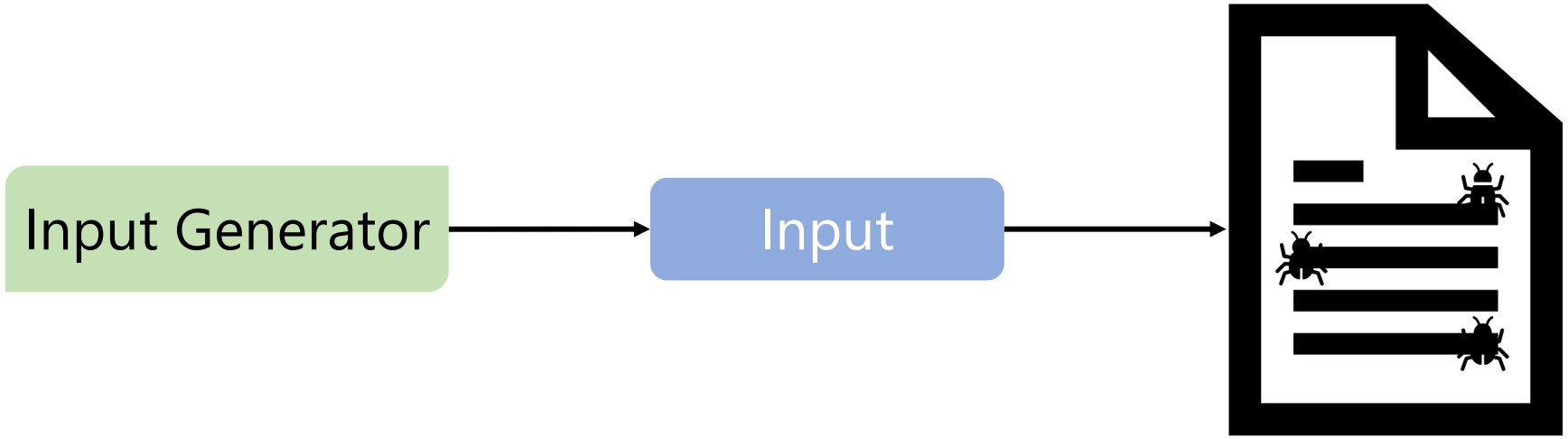



```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

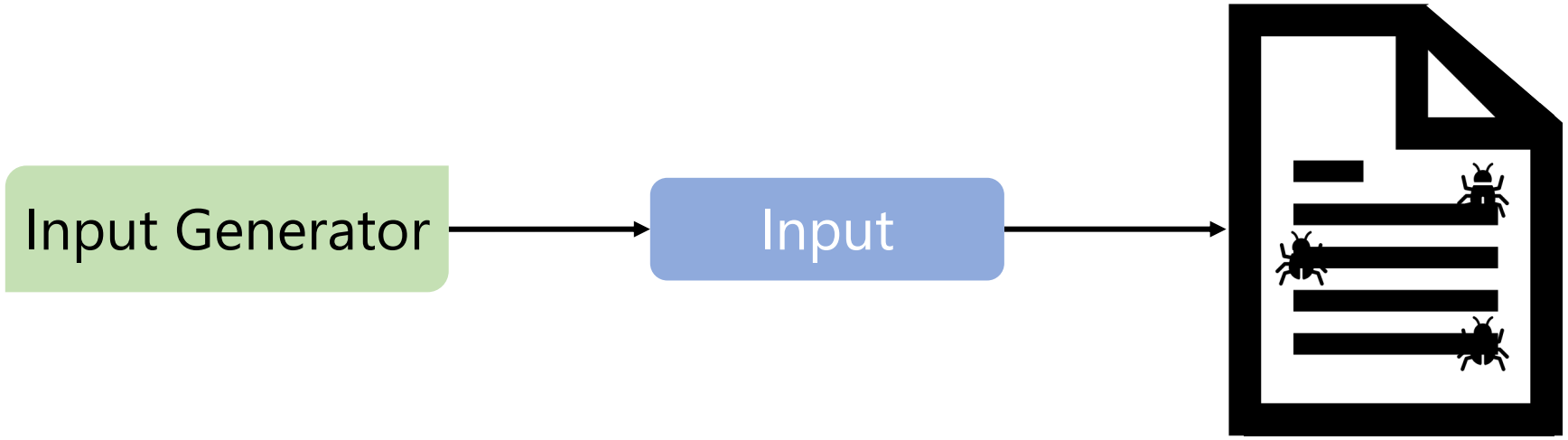
Binary Tree Example: Generator

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Binary Tree Example



Binary Tree Example



Binary Tree Example: Test Program

```
@given(tree = genBinaryTree, to_add = genInt)
def test_insert(tree, to_add):
    assume(is_BST(tree))
    BST_insert(tree, to_add)
    assert(is_BST(tree))
```

Binary Tree Example: Test Program

```
@given(tree = genBinaryTree, to_add = genInt)
def test_insert(tree, to_add):
    assume(is_BST(tree))
    BST_insert(tree, to_add)
    assert(is_BST(tree))
```

Returns if tree is not BST

Binary Tree Example: Test Program

```
@given(tree = genBinaryTree, to_add = genInt)
def test_insert(tree, to_add):
    assume(is_BST(tree))
    BST_insert(tree, to_add)
    assert(is_BST(tree))
```

Returns if tree is not BST

I.e., rejection sampling

Binary Tree Example: Test Program

```
@given(tree = genBinaryTree, to_add = genInt)
def test_insert(tree, to_add):
    assume(is_BST(tree))
    BST_insert(tree, to_add)
    assert(is_BST(tree))
```

Returns if tree is not BST

I.e., rejection sampling

How many rejections?

Background

Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion

Background

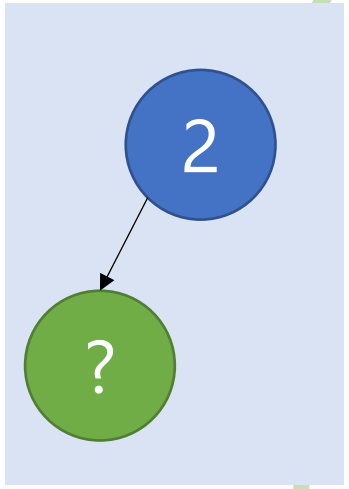
Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion

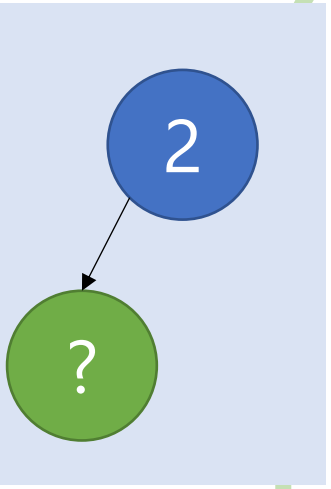
Generating A Binary Search Tree




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Generating A Binary **Search** Tree

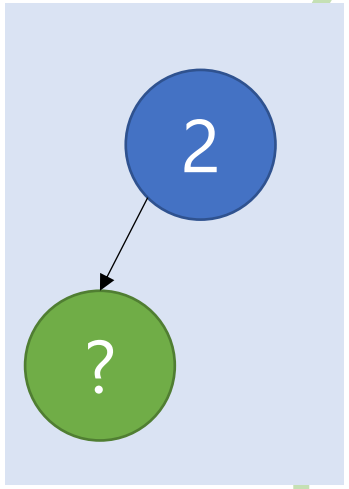
```
is_bst(node) = node.value > node.left.value & is_bst(node.left) & ...
```




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Generating A Binary Search Tree

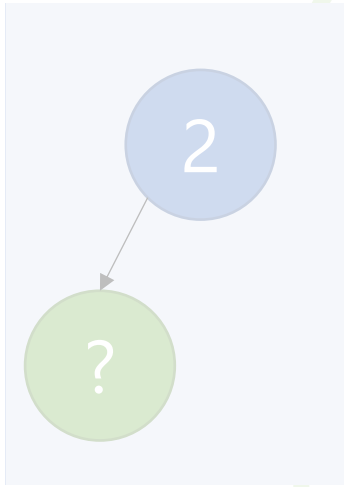
```
is_bst(node) = node.value > node.left.value & is_bst(node.left) & ...
```




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

The Fundamental Tradeoff

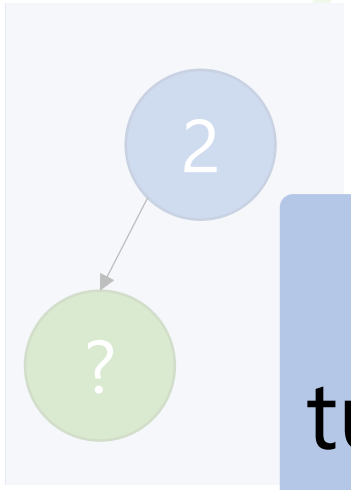
```
is_bst(node) = node.value > node.left.value & is_bst(node.left) & ...
```




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

The Fundamental Tradeoff

```
is_bst(node) = node.value > node.left.value & is_bst(node.left) & ...
```




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Time spent tuning generator

Effectiveness of fuzz testing

Background

Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion

Background

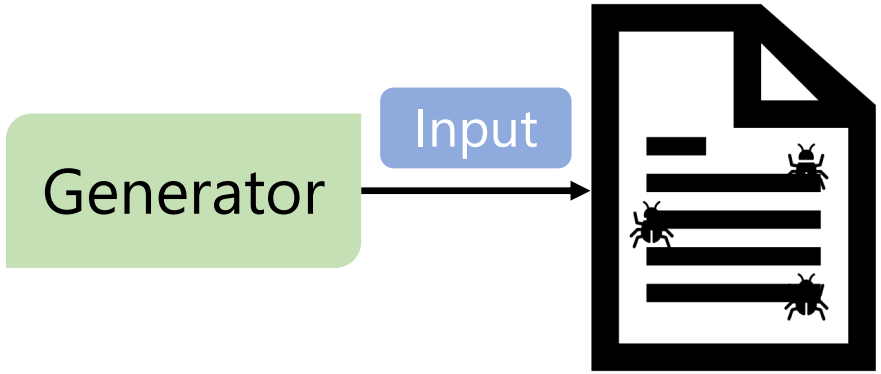
Generator-Based Fuzzing

The Fundamental Tradeoff

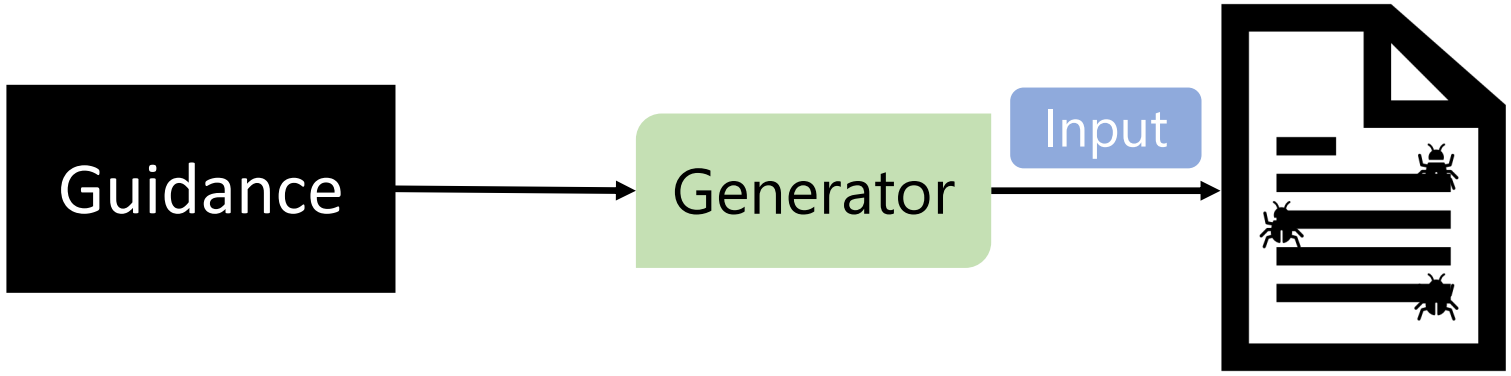
RLCheck: A Solution

Conclusion

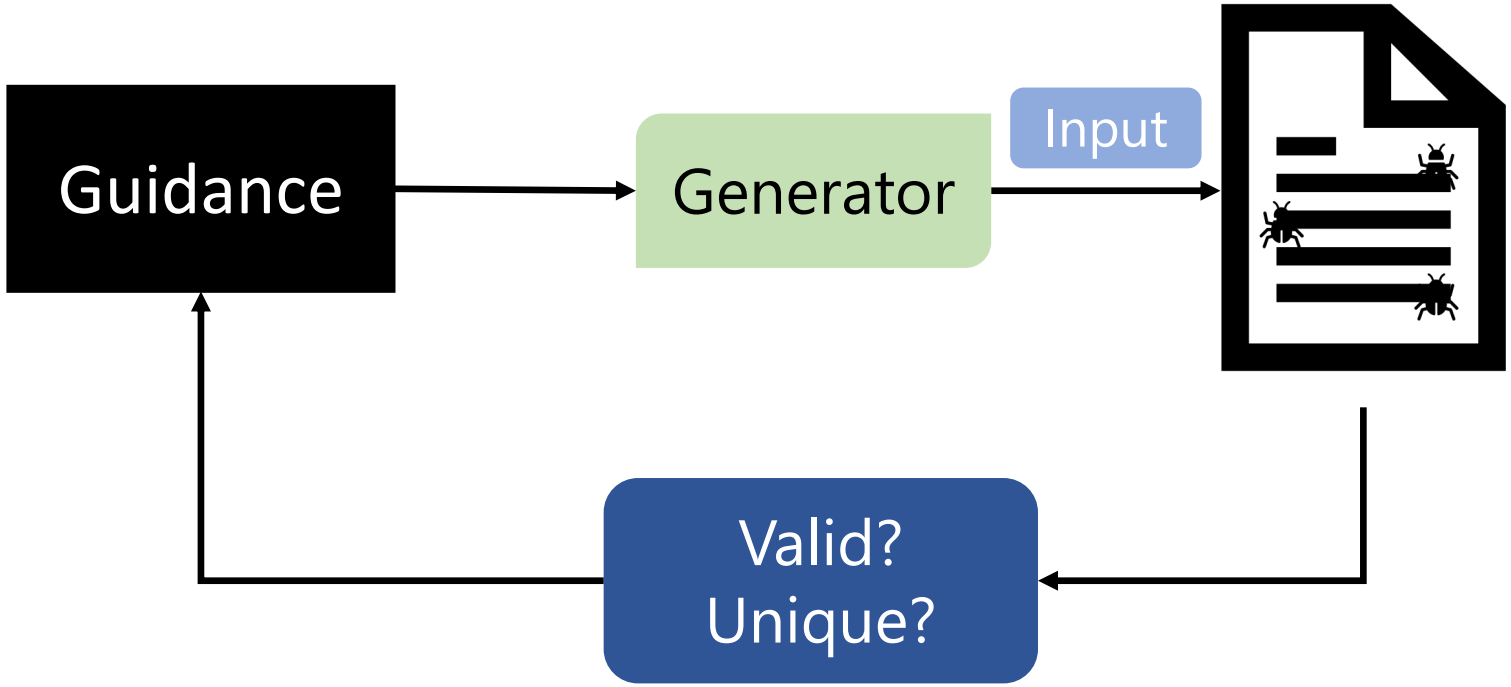
RLCheck: High-Level Idea



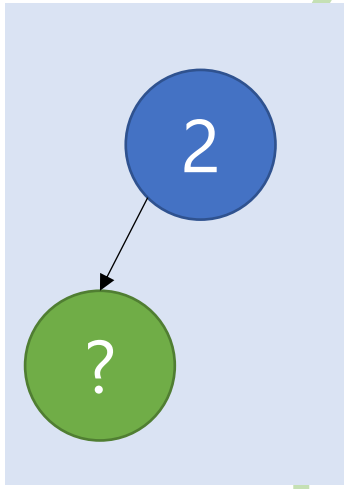
RLCheck: High-Level Idea



RLCheck: High-Level Idea



Recall: Generating A Binary Search Tree




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (dep                          bool():
    node                              + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Can we make the generator choose from these? Without modifying the code?

Can We Guide the *Choices*?

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

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

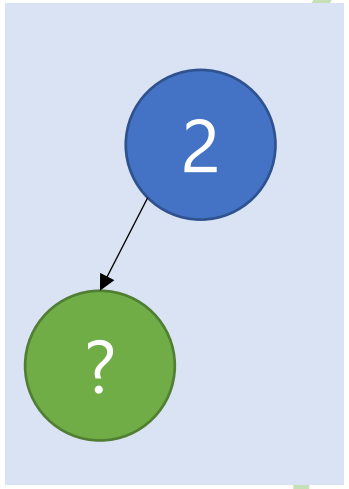
Can We Guide the *Choices*?

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    AX_DEPTH
    = genBin

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

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

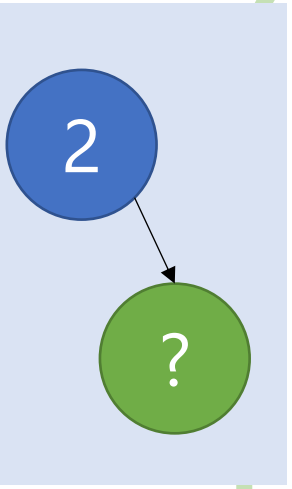
Depends on *context*

Different Context → Different "Good" Choices



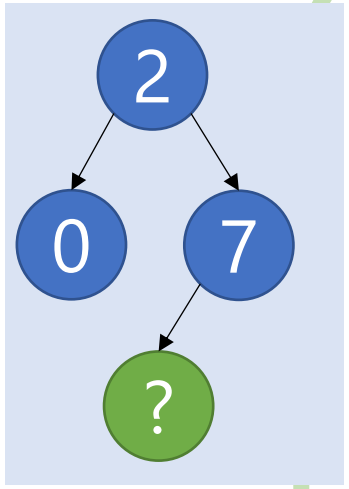

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Different Context → Different "Good" Choices




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Different Context → Different "Good" Choices




```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

Different Context → Different "Good" Choices

```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10])
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool():
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool():
        node.right = genBinaryTree(depth + 1)

    return node
```

RLCheck: Make Best Choices Given Context

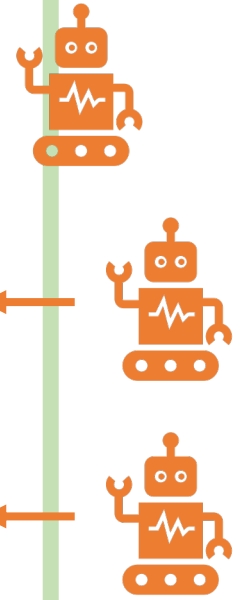
```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10] )
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and random.bool( ):
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and random.bool( ):
        node.right = genBinaryTree(depth + 1)

    return node
```

RLCheck: Make Best Choices Given Context

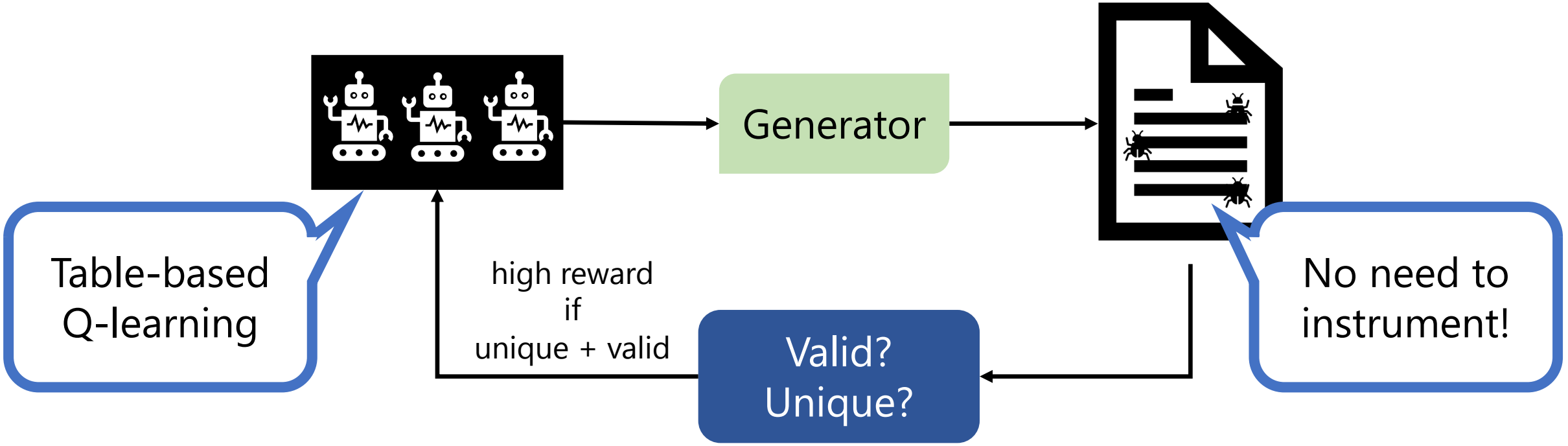
```
def genBinaryTree(depth = 0):
    value = guide.choice([0, 1, …, 10], context)
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and guide.bool(context):
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and guide.bool(context):
        node.right = genBinaryTree(depth + 1)

    return node
```

RLCheck Idea: RL Agent at Each Choice Point

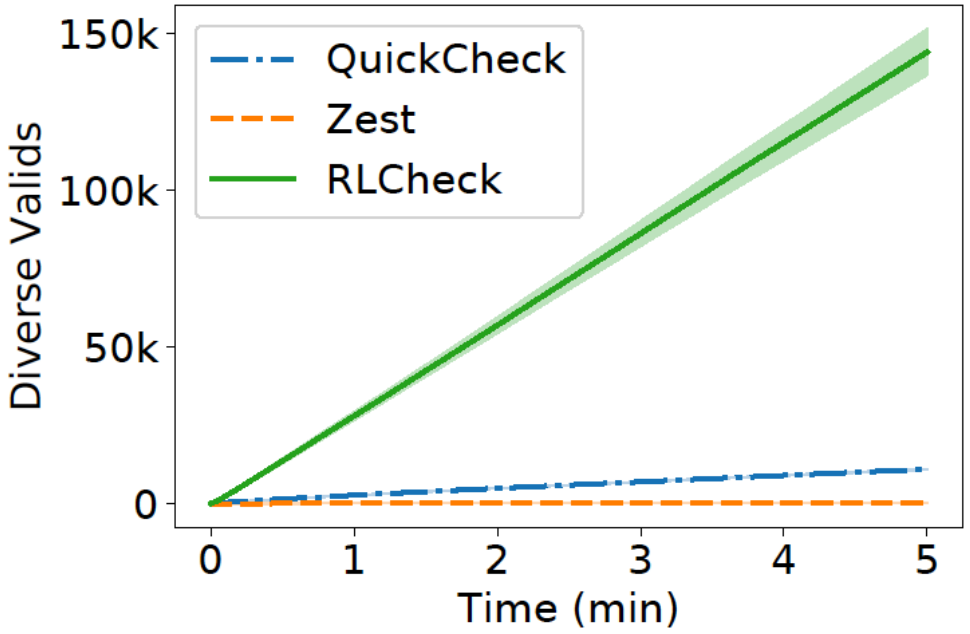
```
def genBinaryTree(depth = 0):
    value = guide.choice([0, 1, …, 10], context)
    node = BinaryTree(value);

    if (depth < MAX_DEPTH) and guide.bool(context):
        node.left = genBinaryTree(depth + 1)

    if (depth < MAX_DEPTH) and guide.bool(context):
        node.right = genBinaryTree(depth + 1)

    return node
```



RLCheck

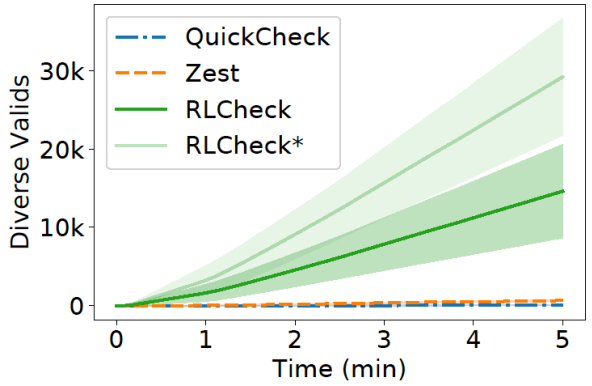

Generator
Table-based
Q-learning
high reward
if
unique + valid
Valid?
Unique?
No need to
instrument!

RLCheck: Many More Unique Valid Inputs

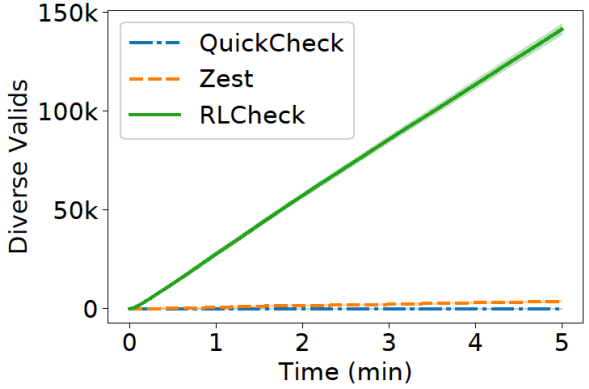


(d) Closure

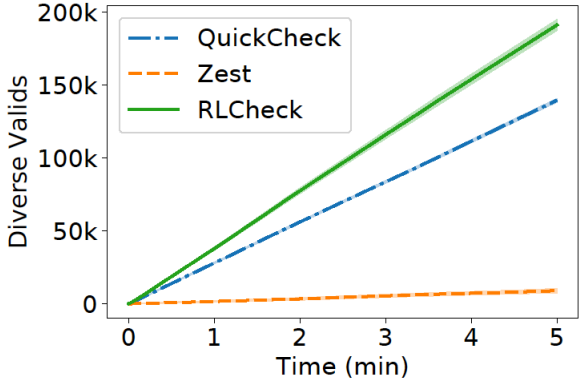
RLCheck: Many More Unique Valid Inputs



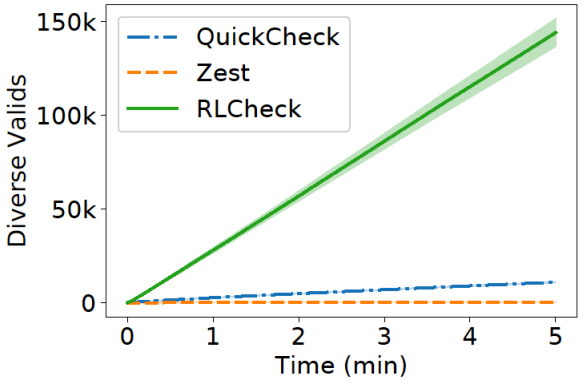
(a) Ant (*: at least 1 valid)



(b) Maven



(c) Rhino



(d) Closure

Background

Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion

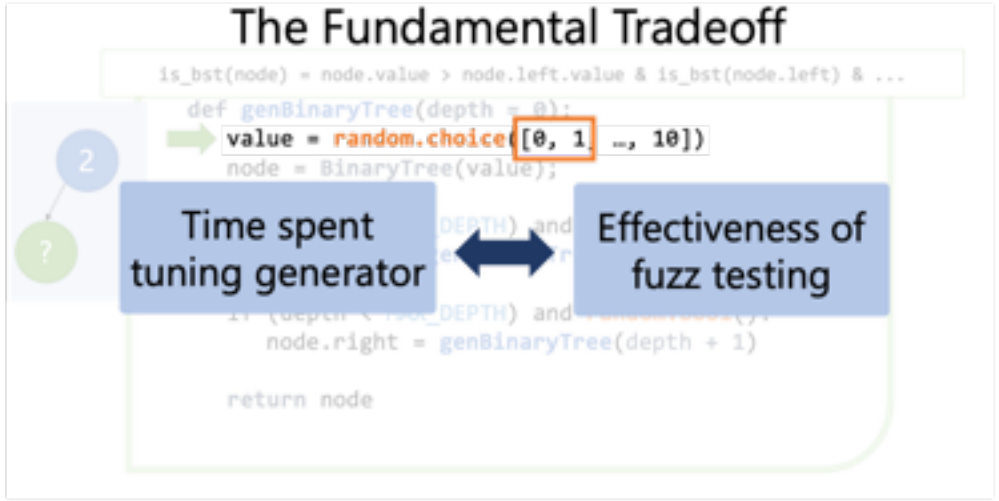
Background

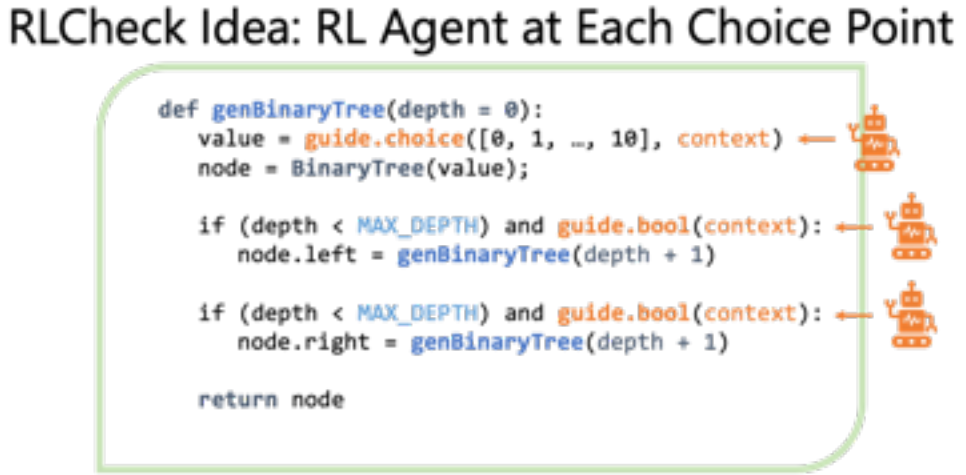
Generator-Based Fuzzing

The Fundamental Tradeoff

RLCheck: A Solution

Conclusion





Quickly Generating Diverse Valid Test Inputs with Reinforcement Learning. Reddy, Lemieux, Padhye, and Sen.

Preprint: https://www.carolemieux.com/rlcheck_preprint.pdf

Code: https://github.com/sameerreddy13/rlcheck

Artifact: `docker pull carolemieux/rlcheck-artifact`