Expanding the Reach of Fuzzing

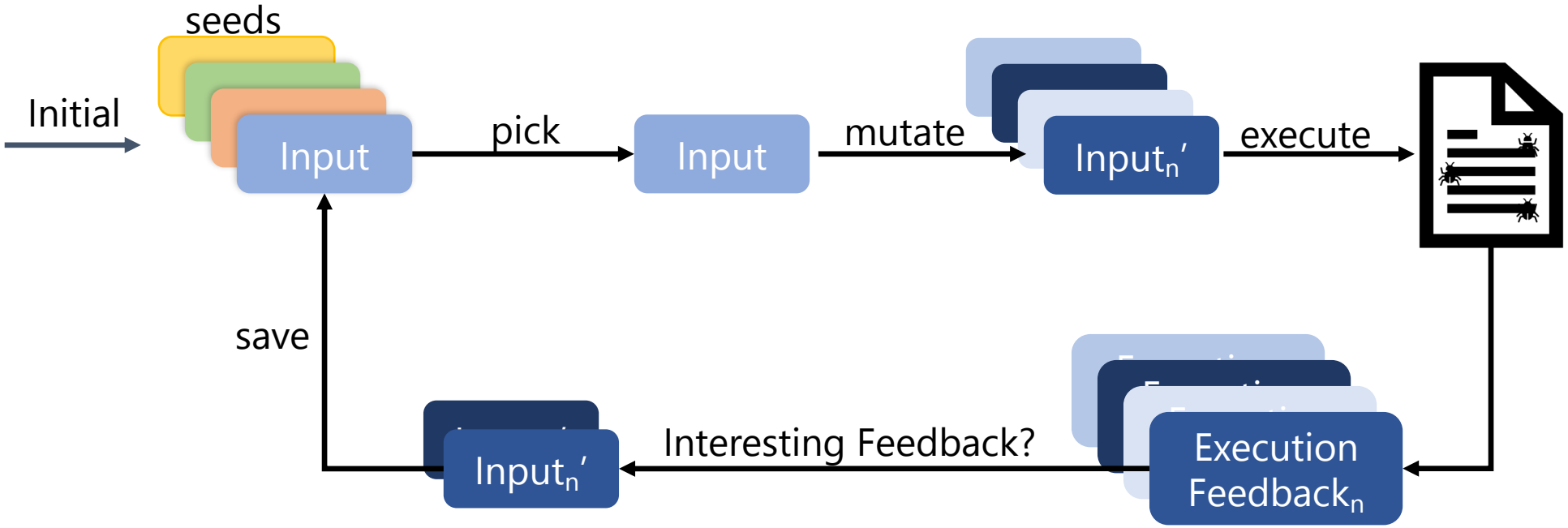
Caroline Lemieux

September 8th, 2020

Fuzzcon Europe

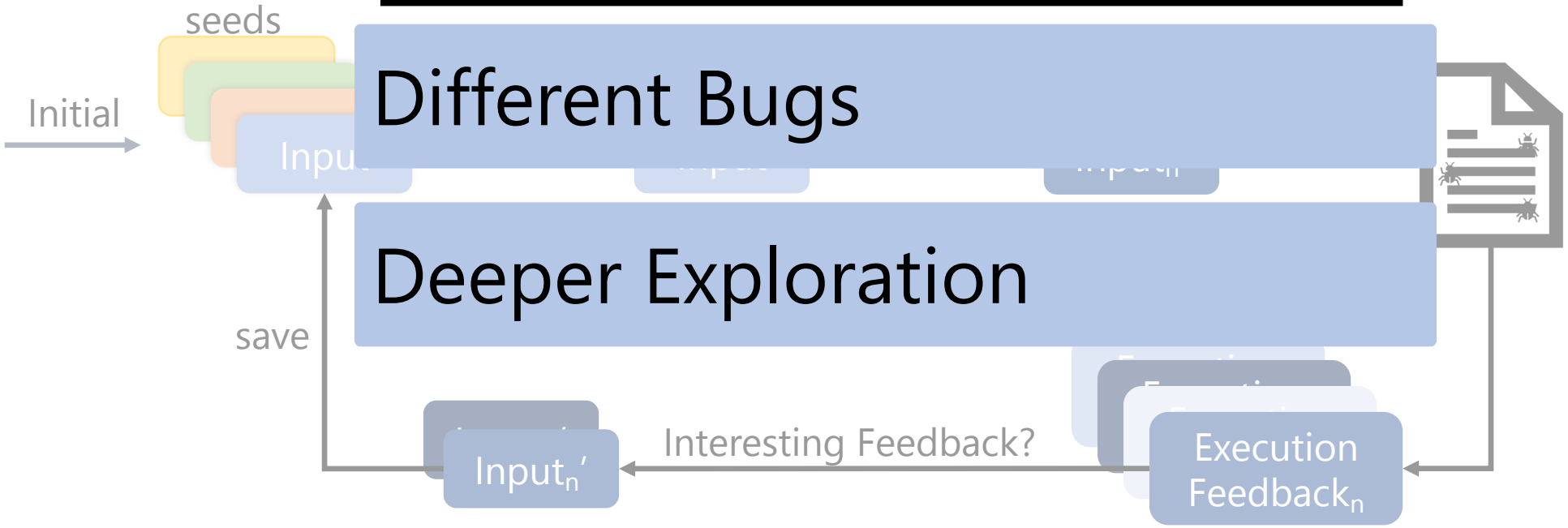
Coverage-Guided Fuzzing

Greybox, Mutational



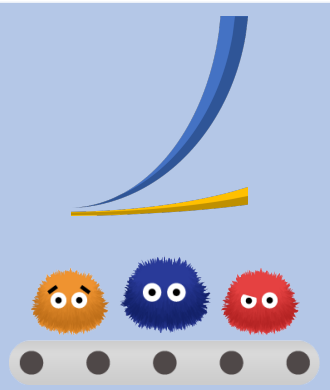
Coverage-Guided Fuzzing

Can we modify CGF tools for:



Different Bugs 

Deeper Exploration 



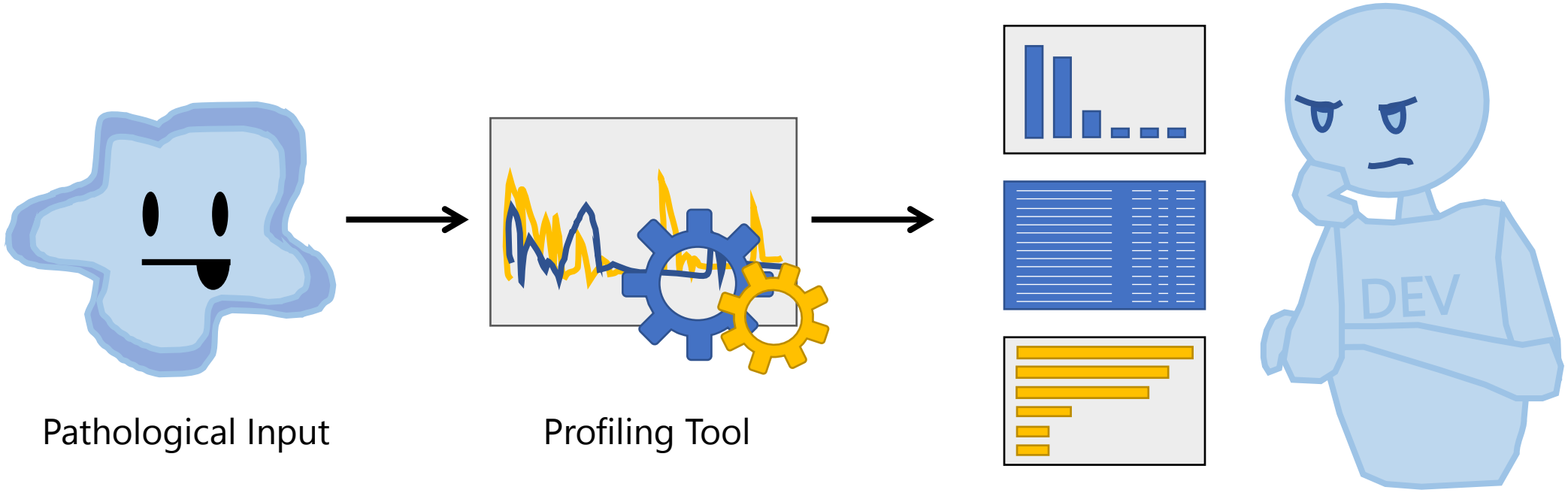
PerfFuzz
https://github.com/carolemieux/perffuzz

FuzzFactory
https://github.com/rohanpadhye/FuzzFactory

Deeper Exploration

Nobody Expects Performance Problems

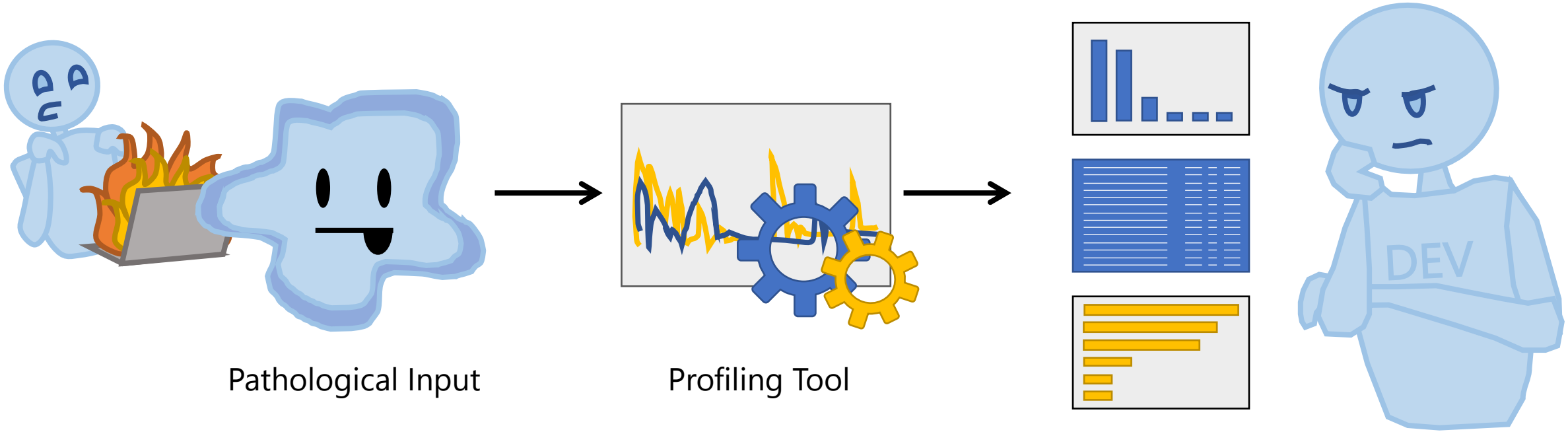
Alleviating Performance Problems



Pathological Input

Profiling Tool

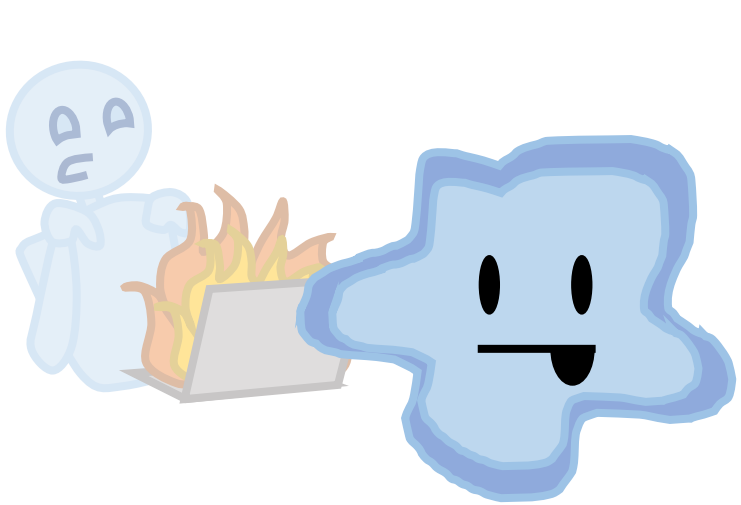
Alleviating Performance Problems



Pathological Input

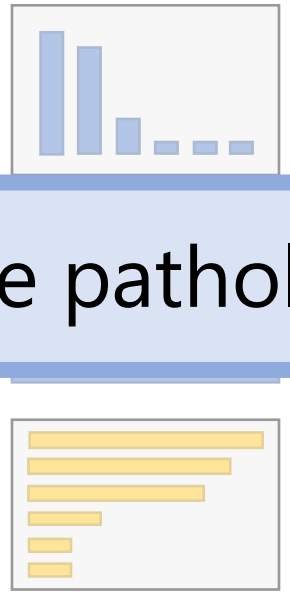
Profiling Tool

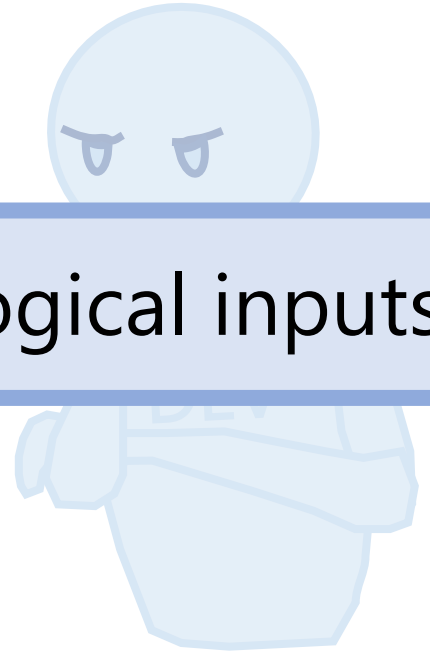
PerfFuzz Goal



Pathological Input

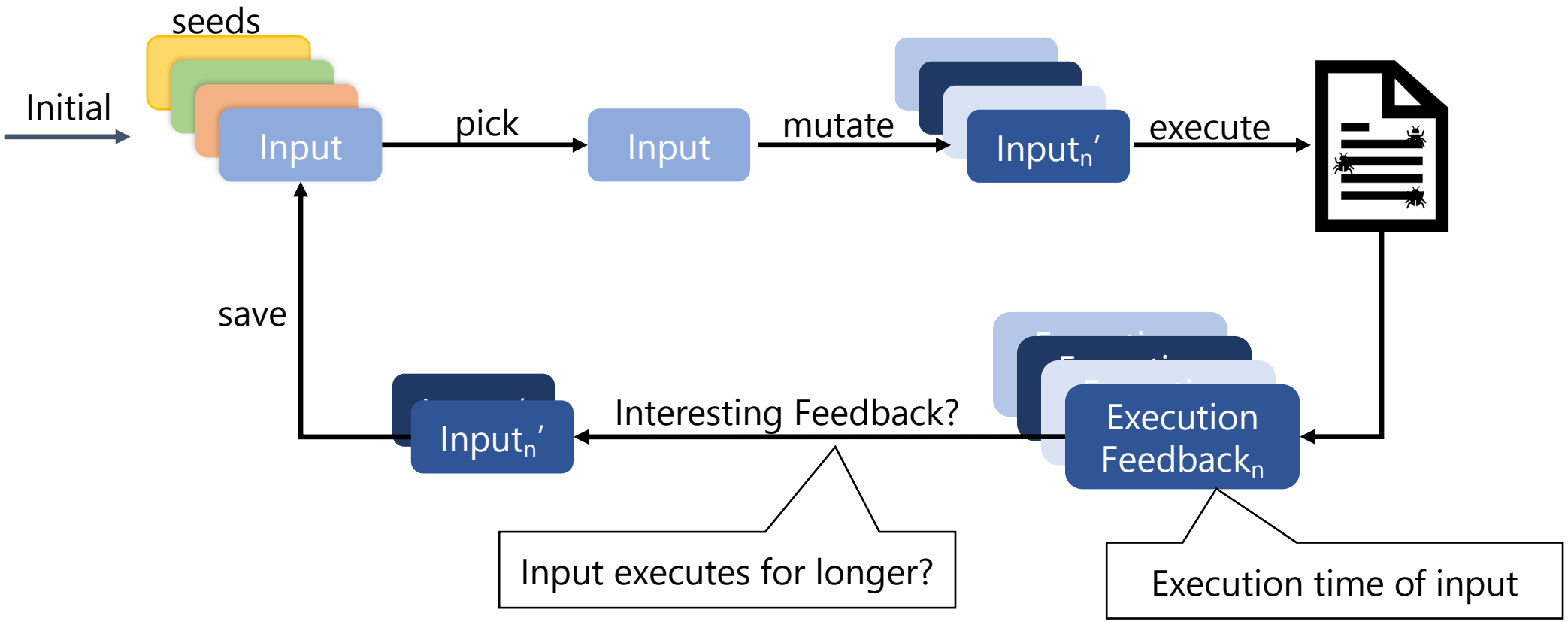
Profiling Tool



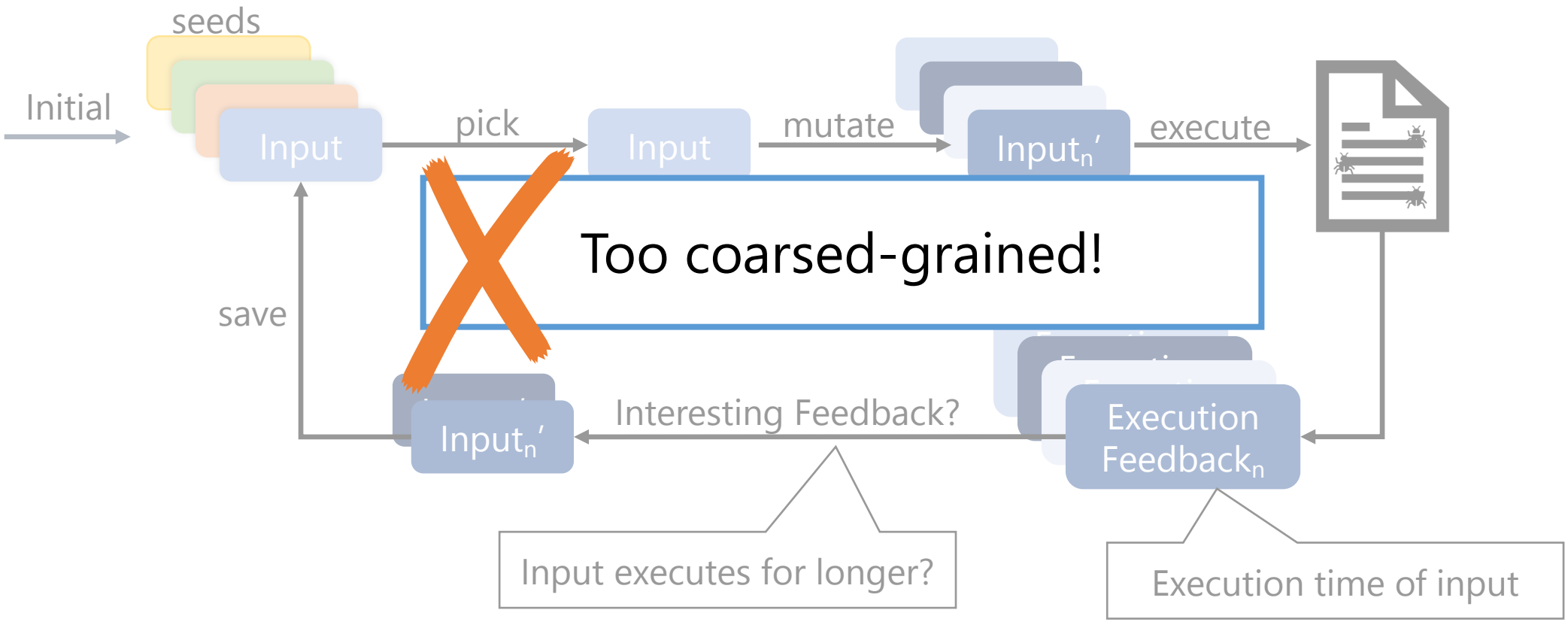


Automatically generate pathological inputs

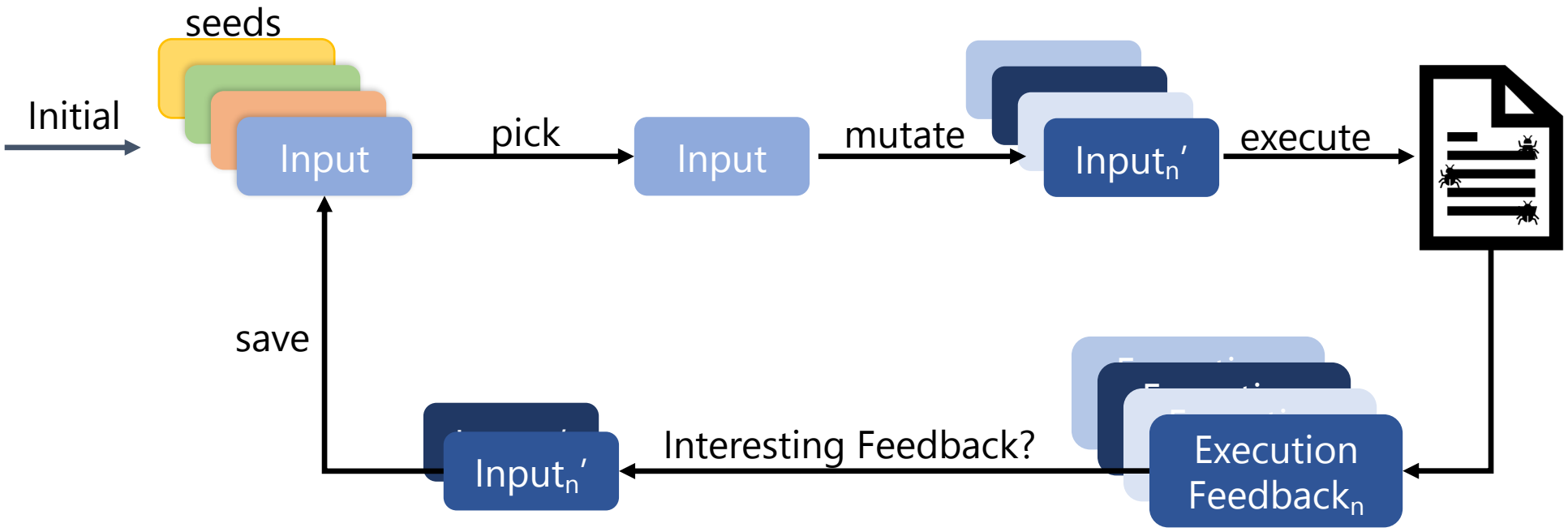
Can We Use Coverage-Guided Fuzzing?



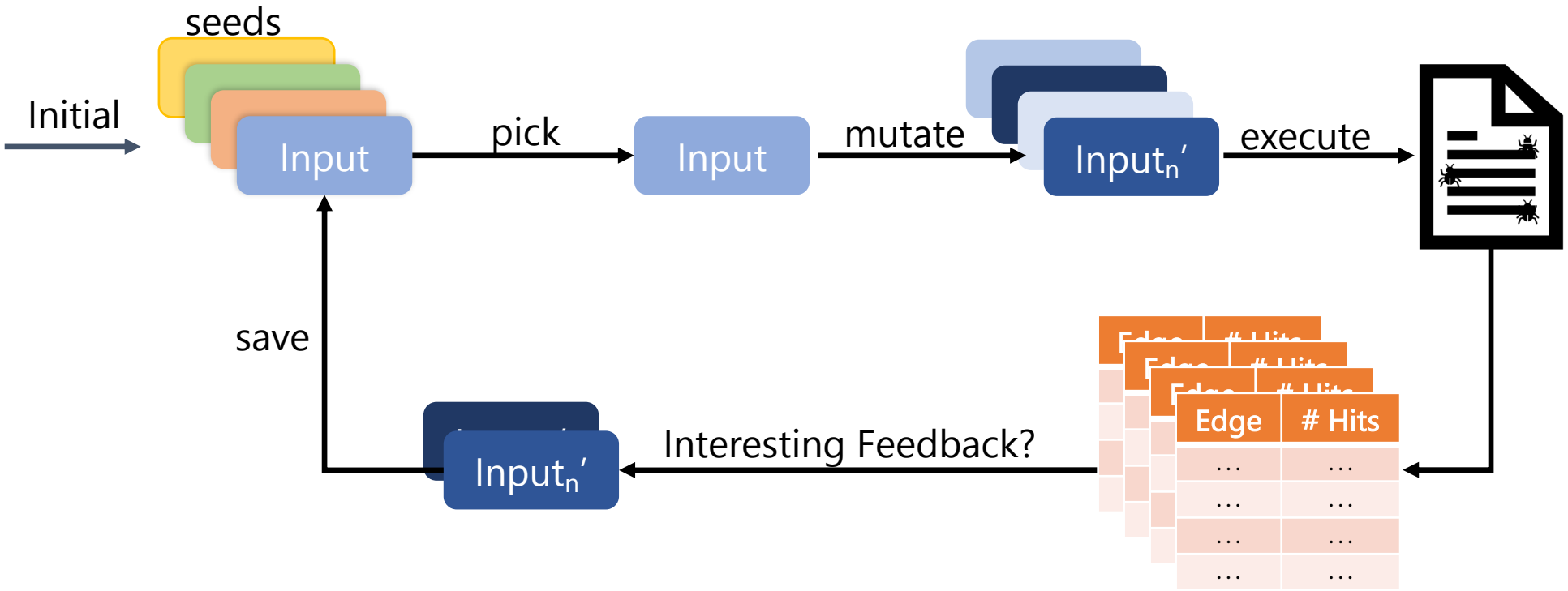
Can We Use Coverage-Guided Fuzzing?



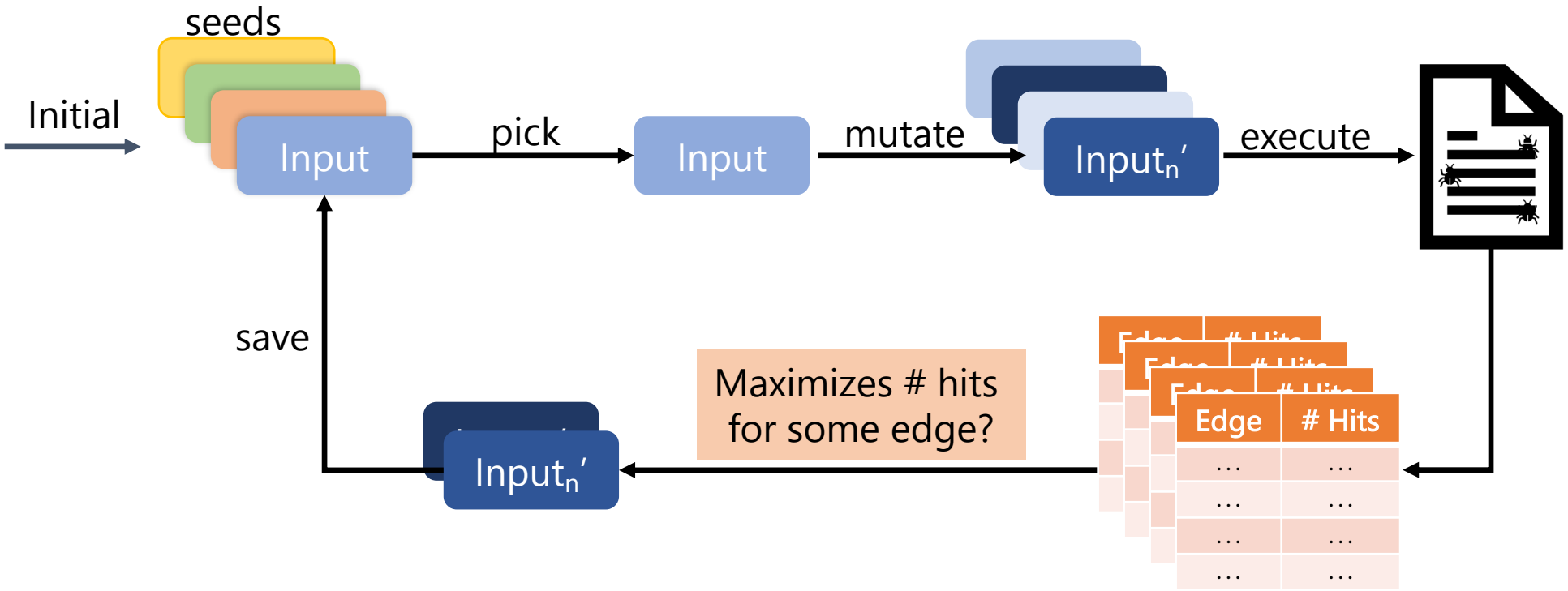
Can We Use Coverage-Guided Fuzzing?



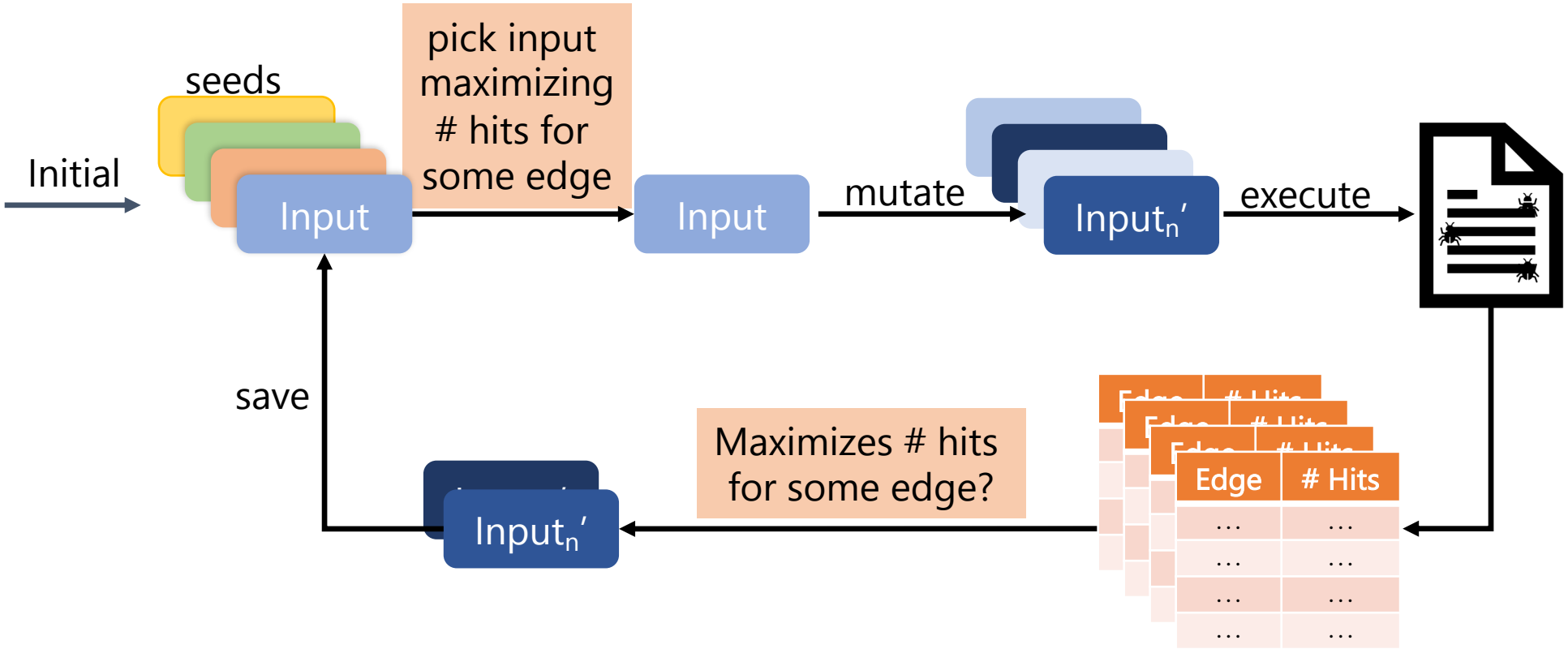
PerfFuzz


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

PerfFuzz


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

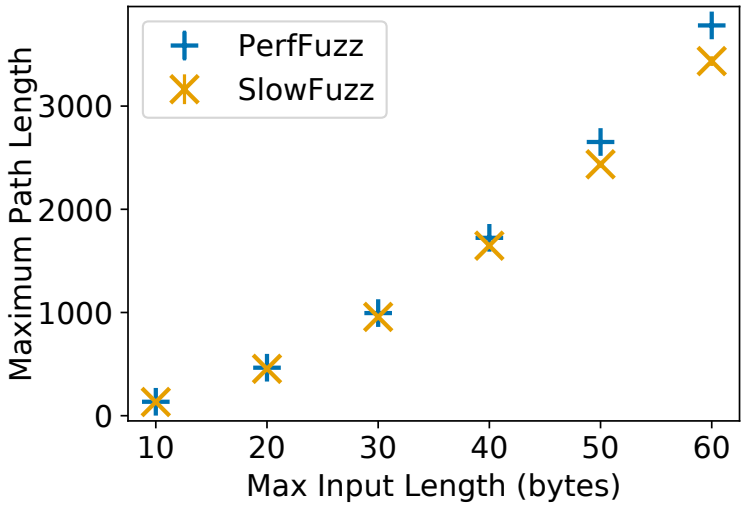
PerfFuzz


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

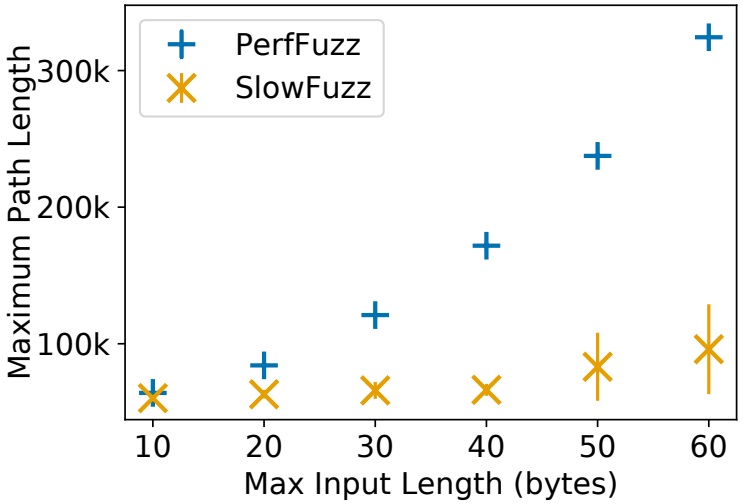
PerfFuzz: Algorithmic Complexity

- Maximum path length for varying input sizes

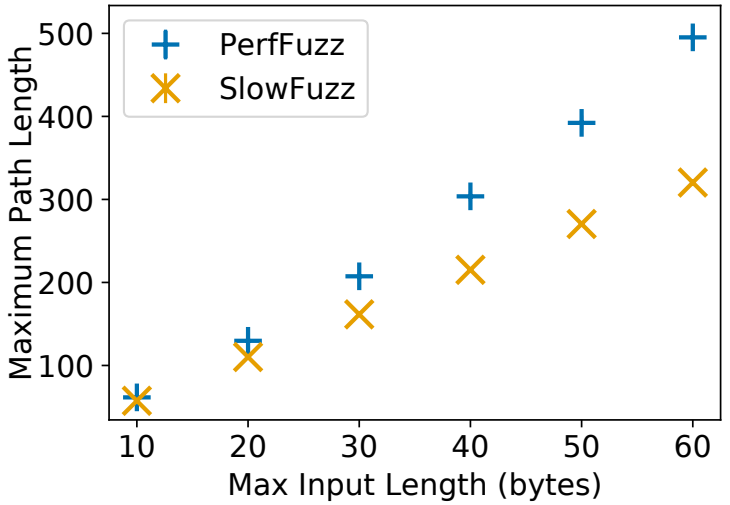
Insertion Sort



PCRE URL regex

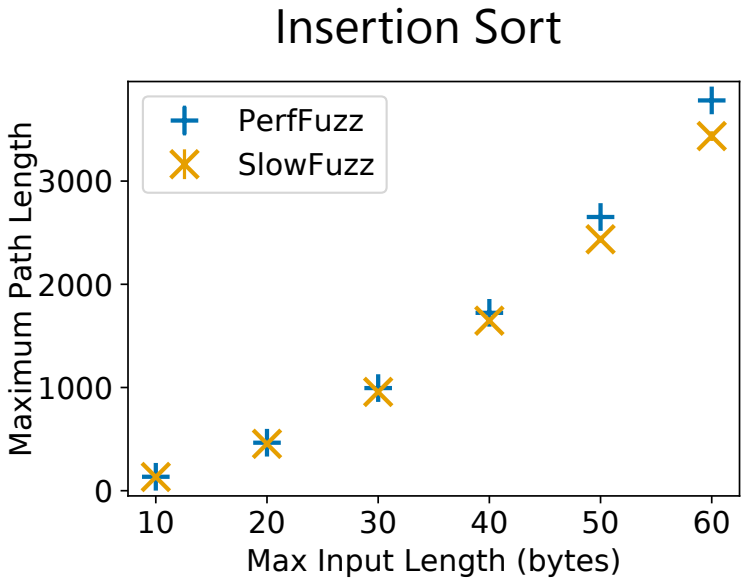


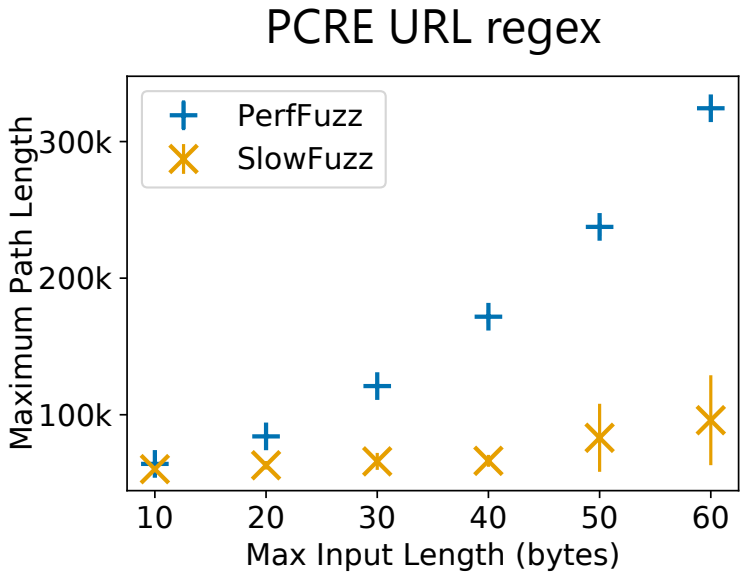
Word Frequency

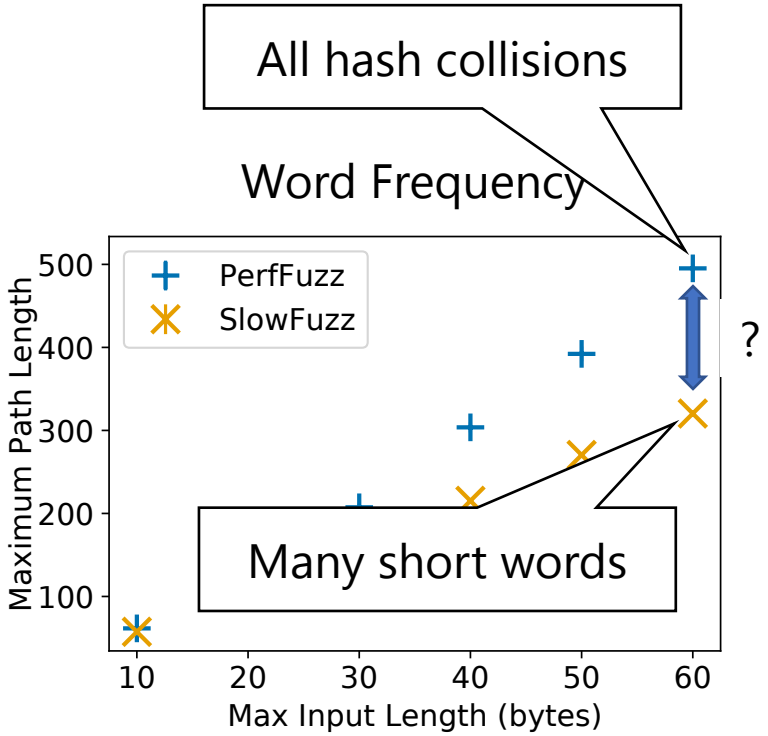


PerfFuzz: Algorithmic Complexity

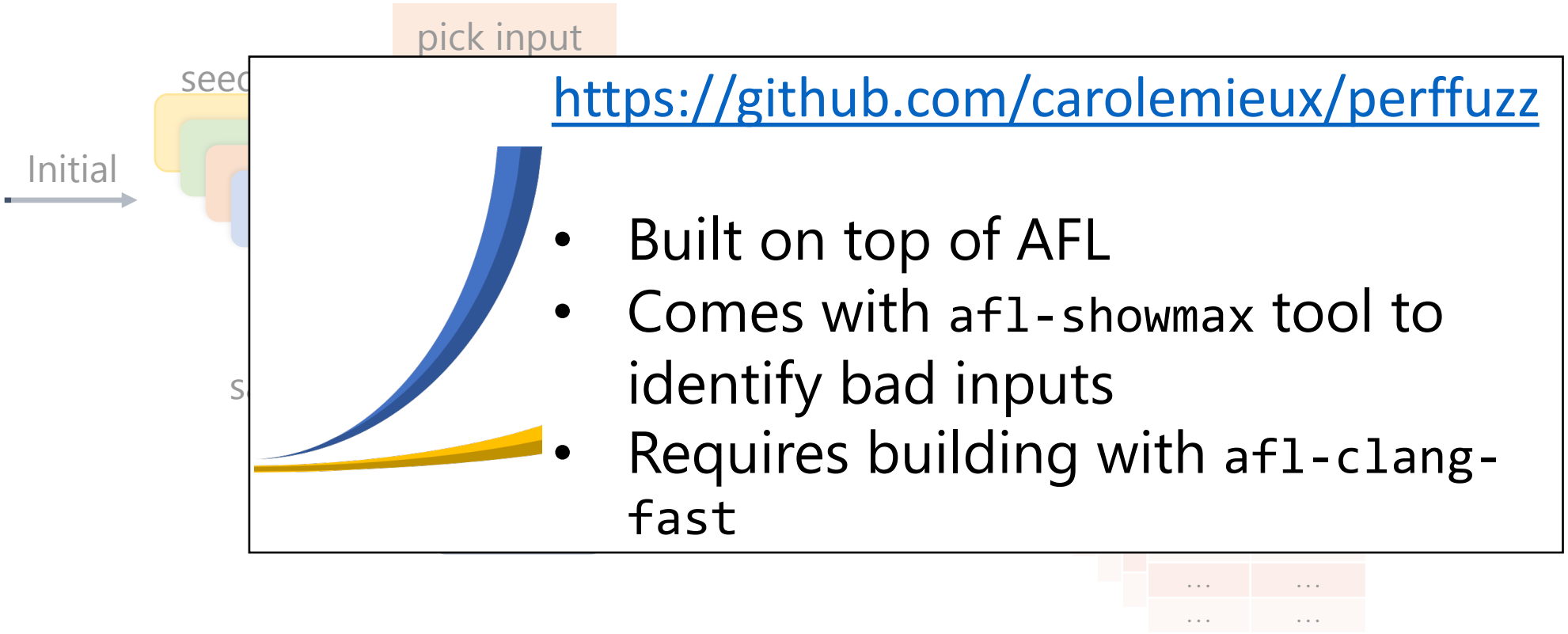
- Maximum path length for varying input sizes







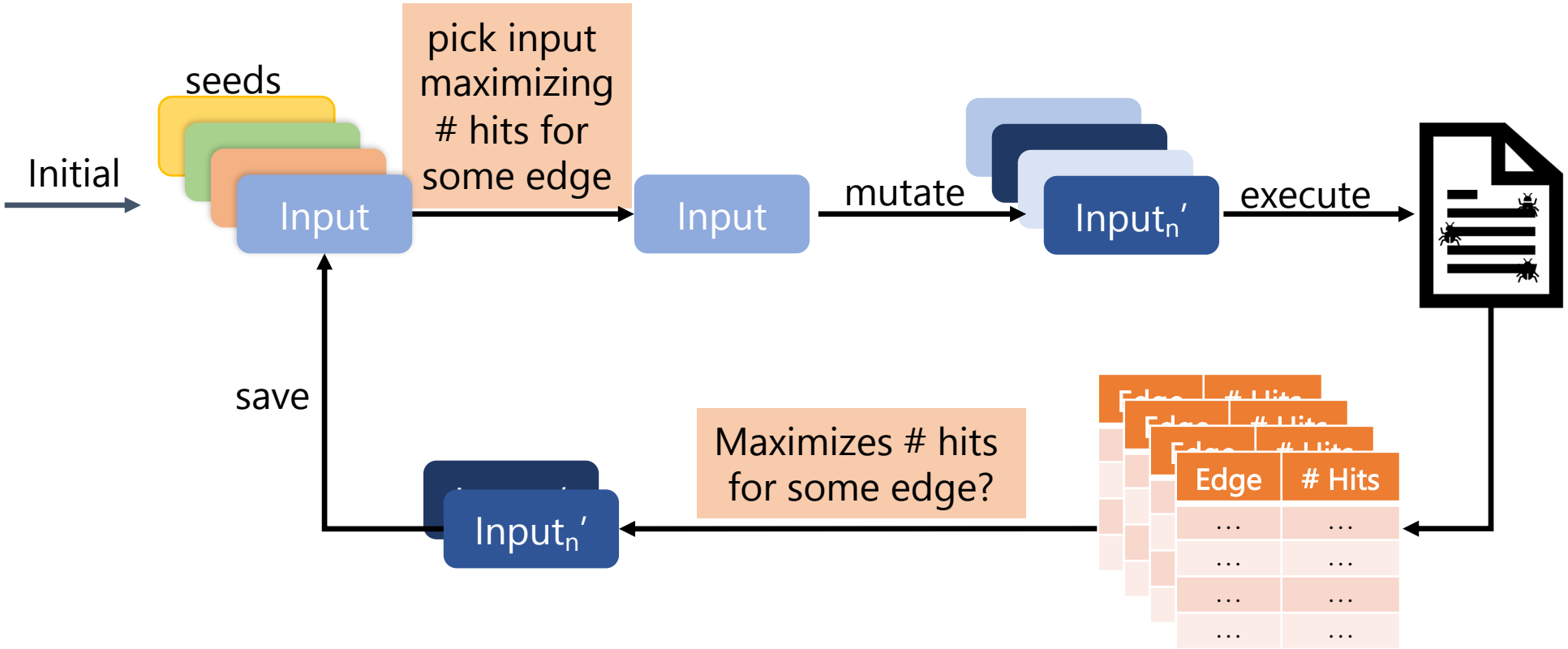
PerfFuzz



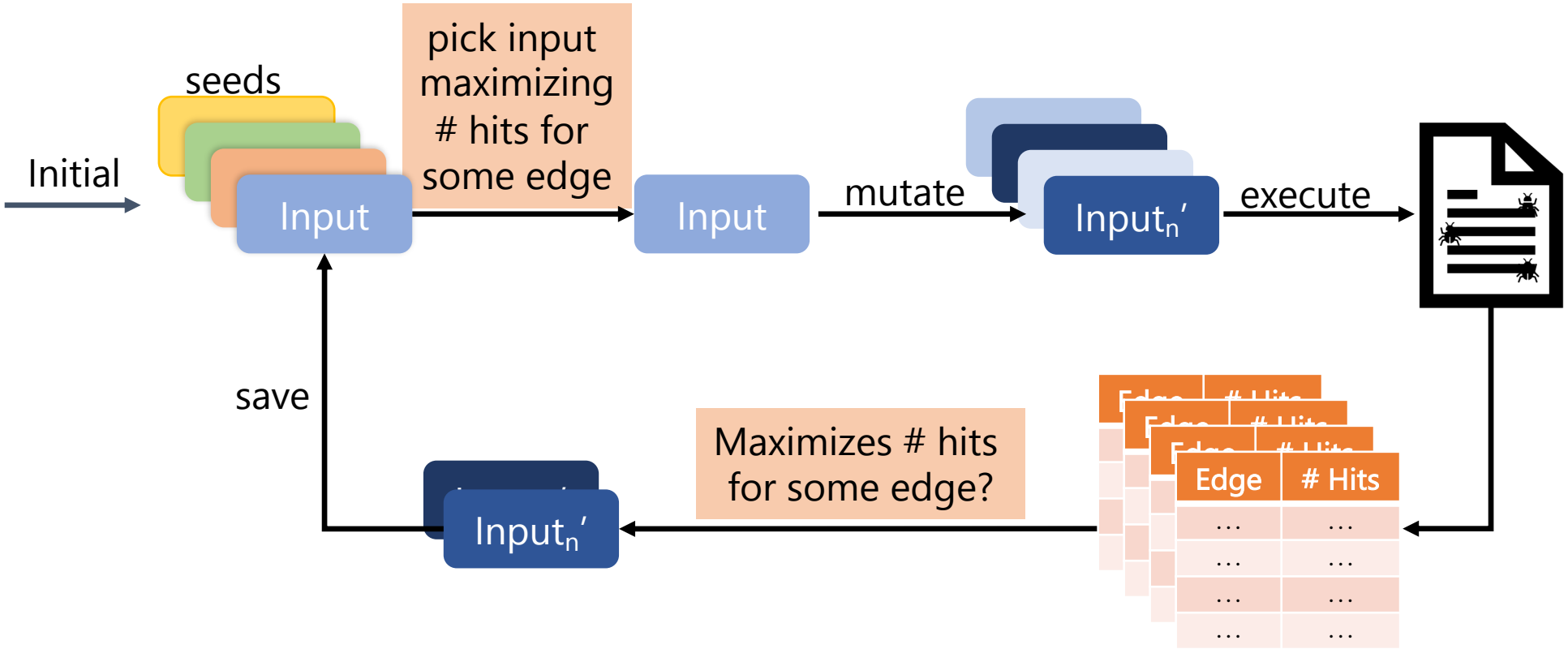
https://github.com/carolemieux/perffuzz

- Built on top of AFL
- Comes with `afl-showmax` tool to identify bad inputs
- Requires building with `afl-clang-fast`

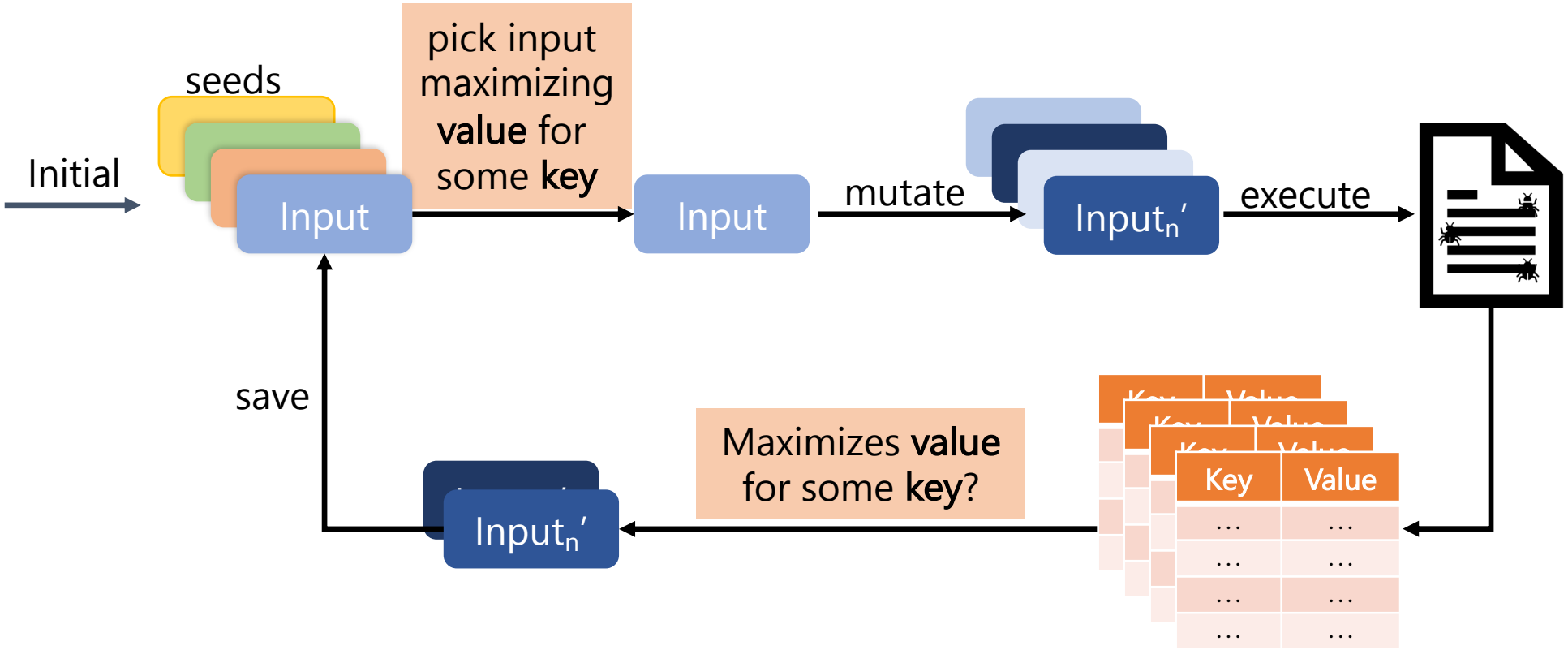
PerfFuzz


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

Observation: Algorithm is More General



Observation: Algorithm is More General

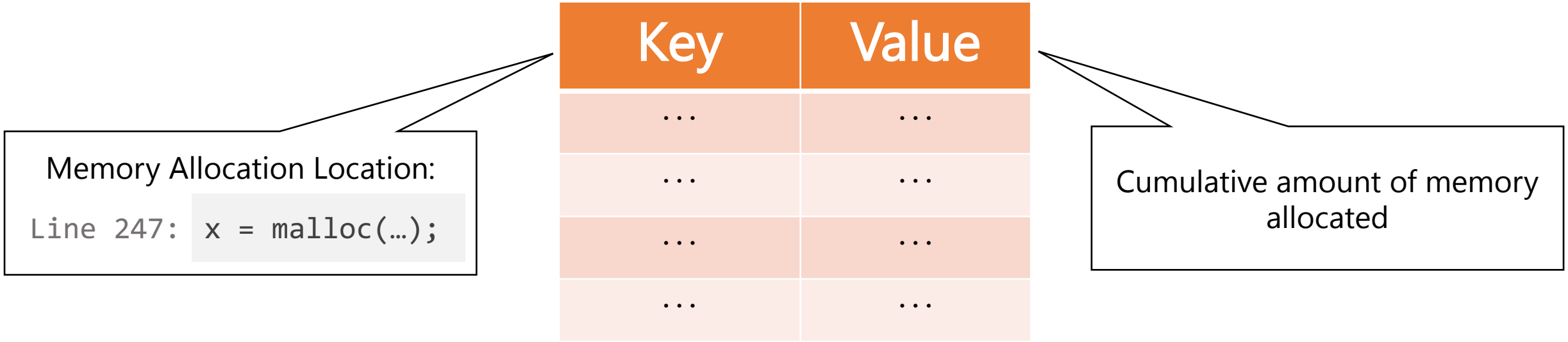


What Other Problems Can We Solve?

Key	Value
…	…
…	…
…	…
…	…

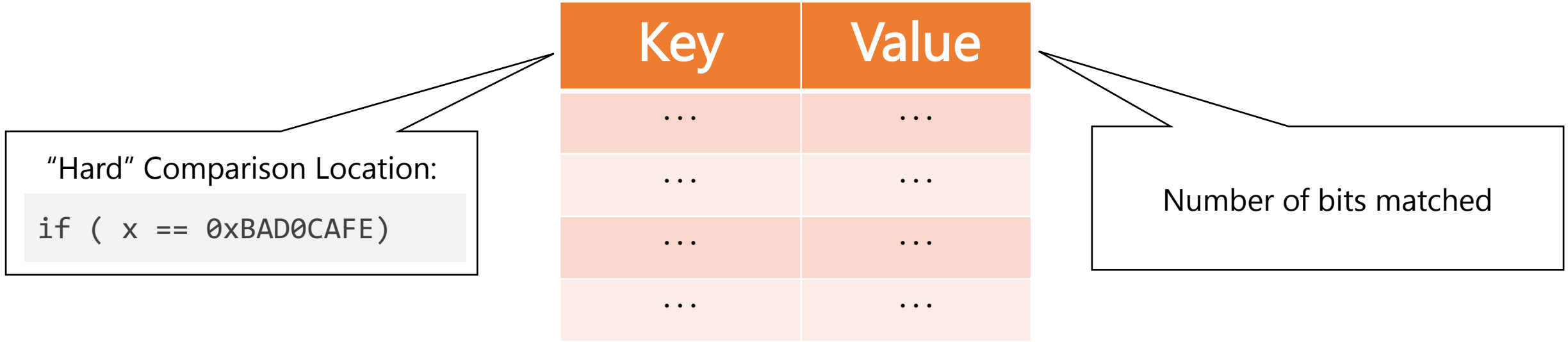
What Other Problems Can We Solve?

e.g. finding memory-allocation maximizing inputs

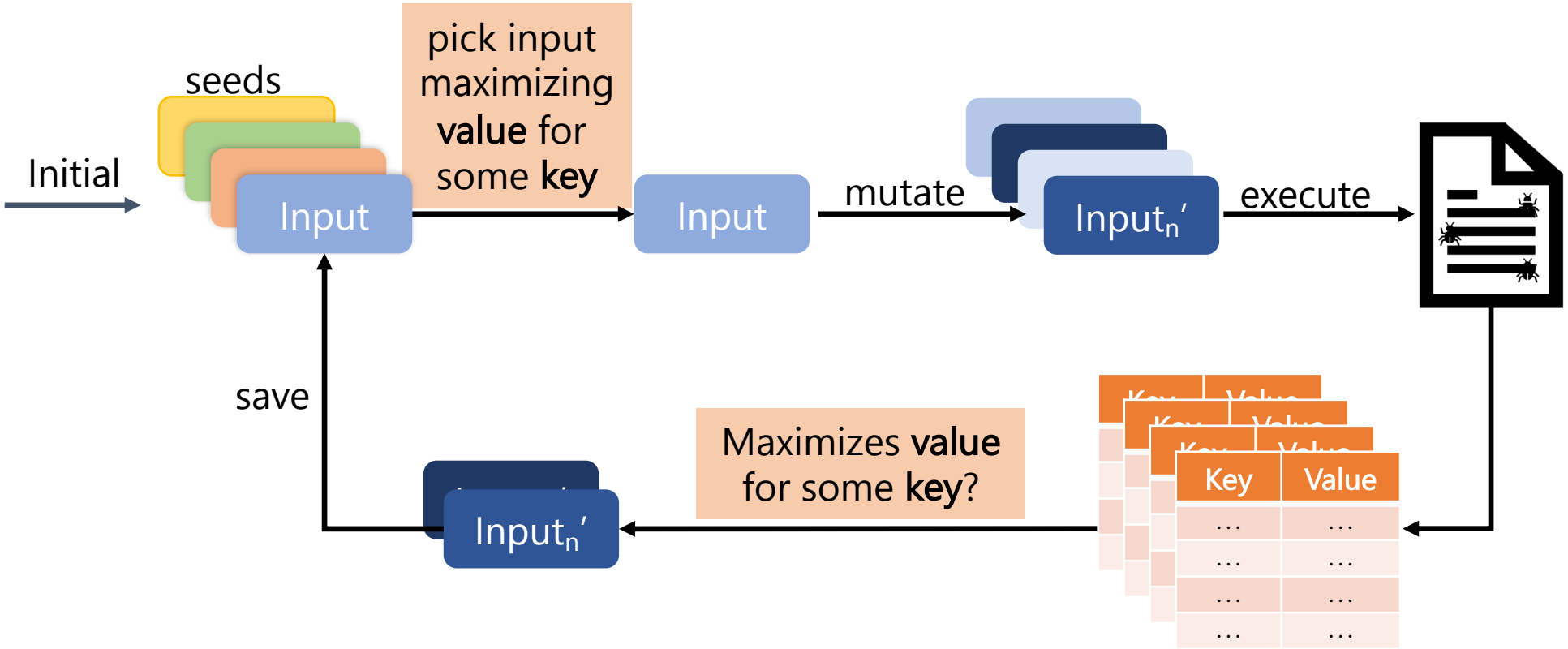


What Other Problems Can We Solve?

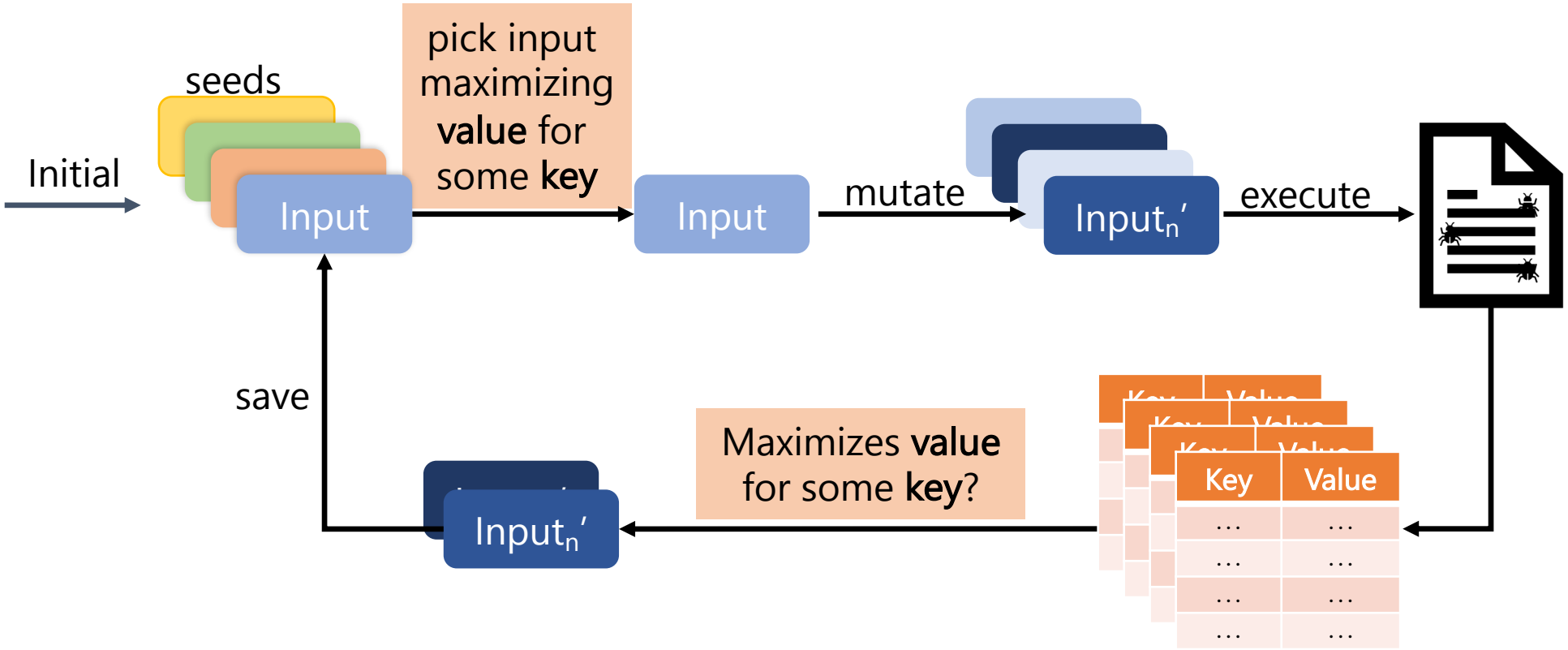
e.g. going through “hard” comparisons



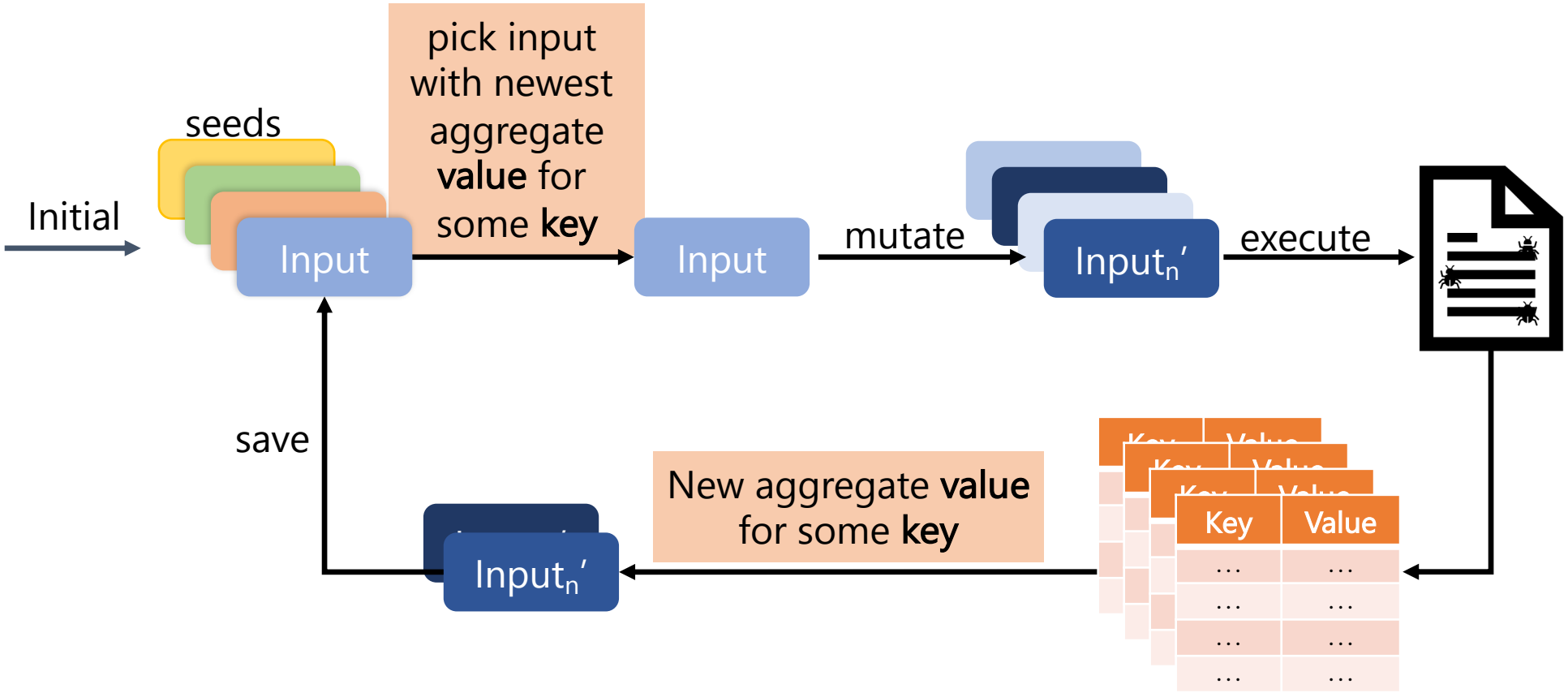
Observation: Algorithm is More General



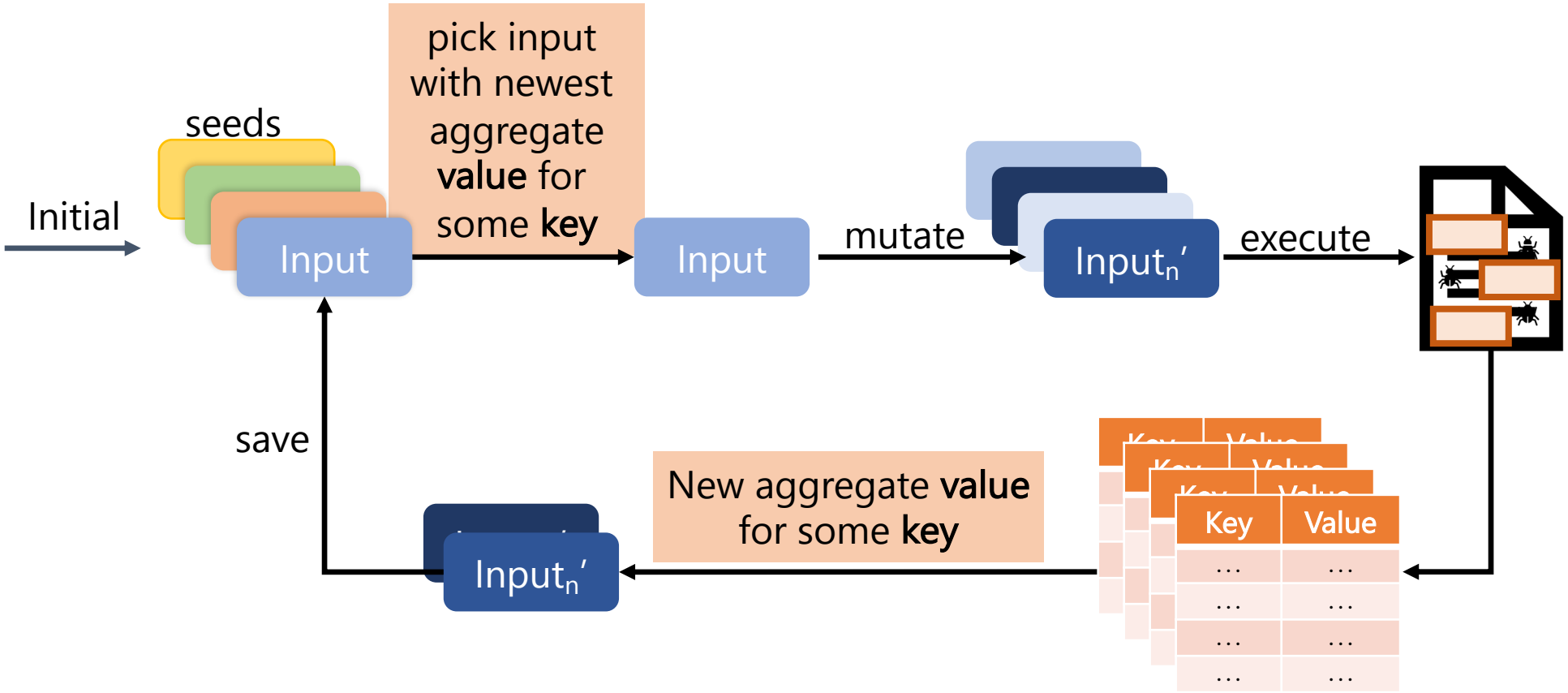
FuzzFactory



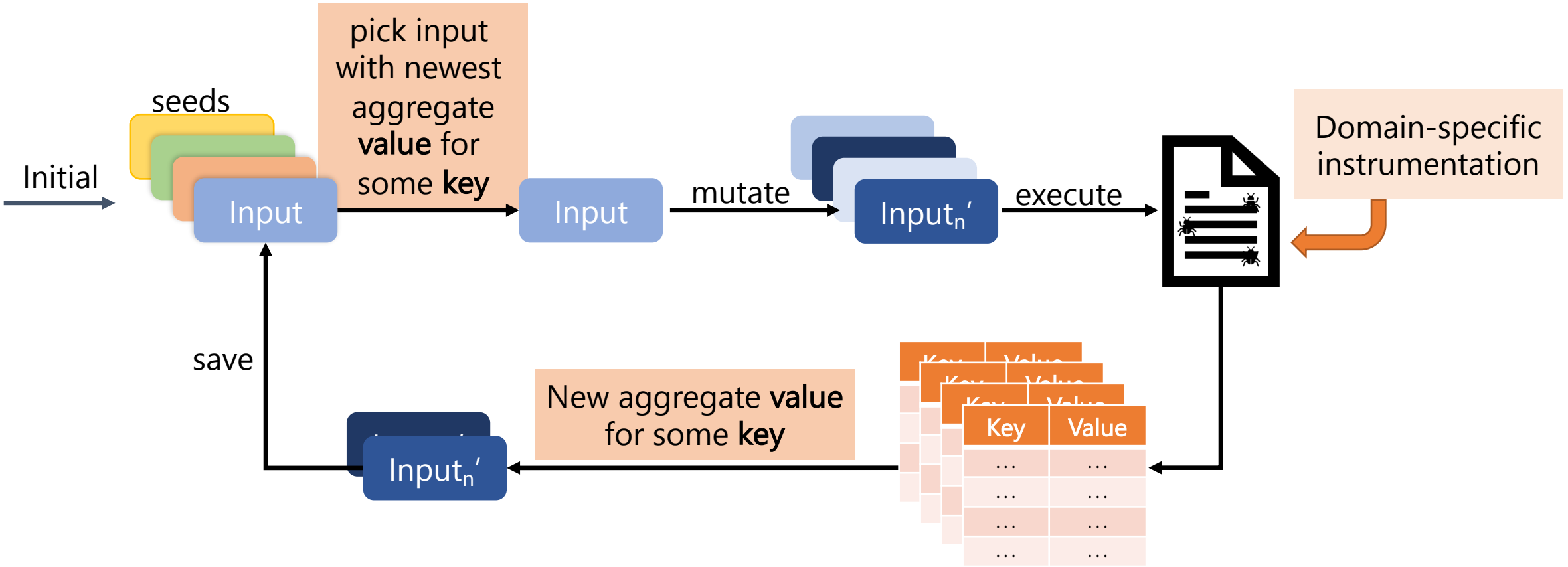
FuzzFactory, Step 1: Generalize Algorithm



FuzzFactory, Step 2: Separate Algo & Feedback

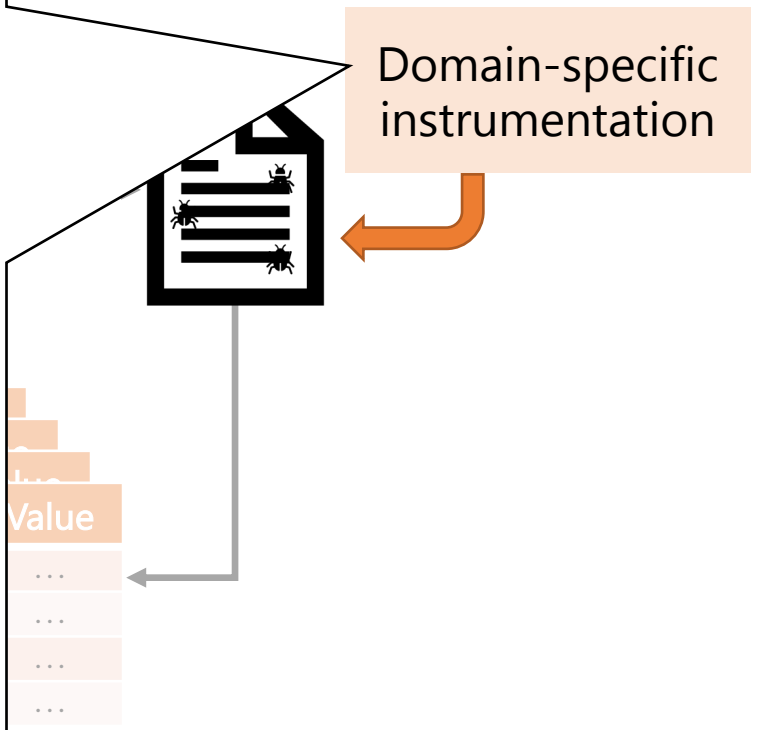


FuzzFactory, Step 2: Separate Algo & Feedback



FuzzFactory, Step 2: Separate Algo & Feedback

```
/*
* Creates a new DSF map `name` with `size` keys,
* `reducer` function, and `initial` aggregate value.
*
* To be called at the top-level global scope.
*/
FUZZFACTORY_DSF_NEW(name, size, reducer, initial)

/* Set dsf[k] = max(dsf[k], v); */
FUZZFACTORY_DSF_MAX(dsf, k, v)
/* Set dsf[k] = dsf[k] | v; */
FUZZFACTORY_DSF_BIT(dsf, k, v)
/* Set dsf[k] = v; */
FUZZFACTORY_DSF_SET(dsf, k, v)
/* Set dsf[k] = dsf[k] + v; */
FUZZFACTORY_DSF_INC(dsf, k, v)
```



Six LLVM-based Domains in FuzzFactory

Fuzzer	Keys	Values	Aggregation	LoC (C++)
Port of **SlowFuzz** [Petsios et al. ‘17]				
Port of **PerfFuzz** [Lemieux et al. ‘18]				
Validity Fuzzing [Padhye et al. ’19]				
Mem **Alloc** Fuzzing				
Cmp Fuzzing				
Incremental Fuzzing				

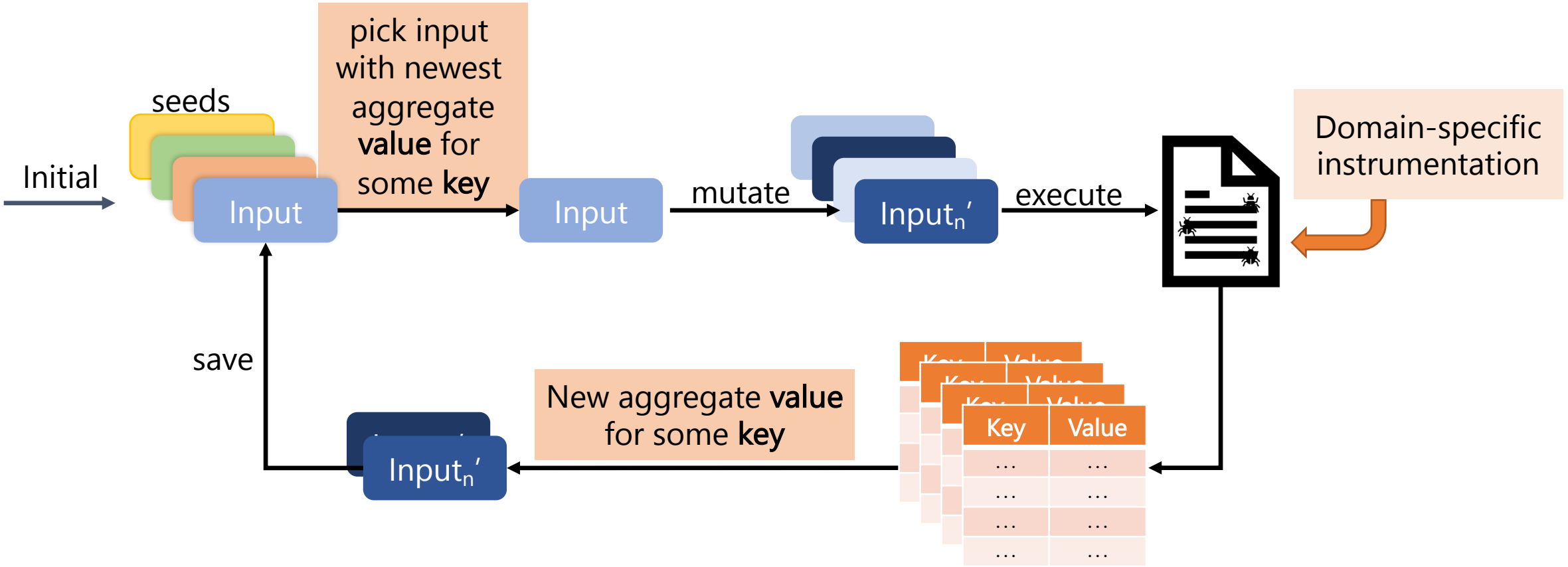
Six LLVM-based Domains in FuzzFactory

Fuzzer	Keys	Values	Aggregation	LoC (C++)
Port of **SlowFuzz** [Petsios et al. '17]	Singleton	Path length	max	
Port of **PerfFuzz** [Lemieux et al. '18]	Basic Blocks	Exec Counts	max	
Validity Fuzzing [Padhye et al. '19]	Basic Blocks	Exec Counts if Valid else 0	log-union (AFL-style bucketing)	
Mem **Alloc** Fuzzing	Locations invoking malloc()/calloc()	# of bytes allocated	max	
Cmp Fuzzing	==, strcmp, memcmp, switch, etc.	# of bits common between operands	max	
Incremental Fuzzing	Basic Block Transitions	Exec Counts	log-union (AFL-style bucketing)	

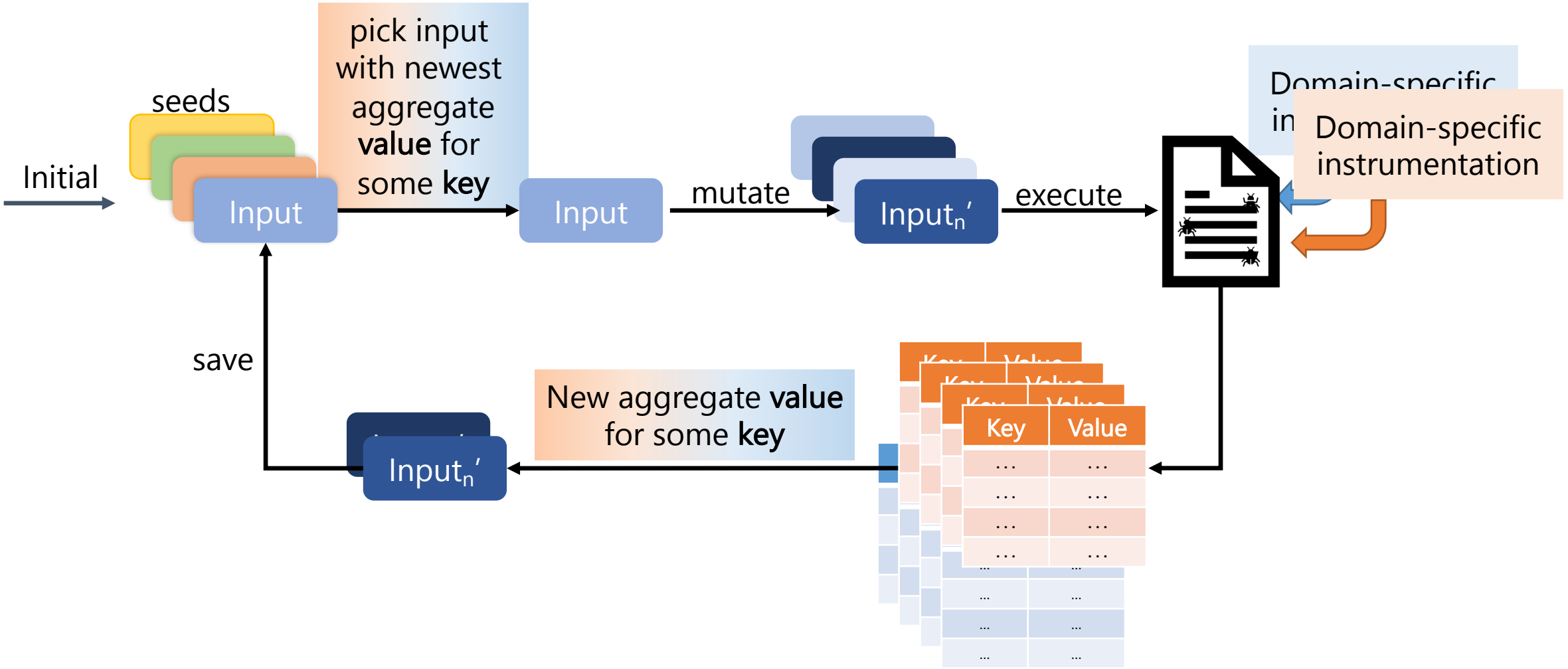
Six LLVM-based Domains in FuzzFactory

Fuzzer	Keys	Values	Aggregation	LoC (C++)
Port of **SlowFuzz** [Petsios et al. '17]	Singleton	Path length	max	**18**
Port of **PerfFuzz** [Lemieux et al. '18]	Basic Blocks	Exec Counts	max	**19**
Validity Fuzzing [Padhye et al. '19]	Basic Blocks	Exec Counts if Valid else 0	log-union (AFL-style bucketing)	**24**
Mem **Alloc** Fuzzing	Locations invoking malloc()/calloc()	# of bytes allocated	max	**29**
Cmp Fuzzing	==, strcmp, memcmp, switch, etc.	# of bits common between operands	max	**355**
Incremental Fuzzing	Basic Block Transitions	Exec Counts	log-union (AFL-style bucketing)	**146**

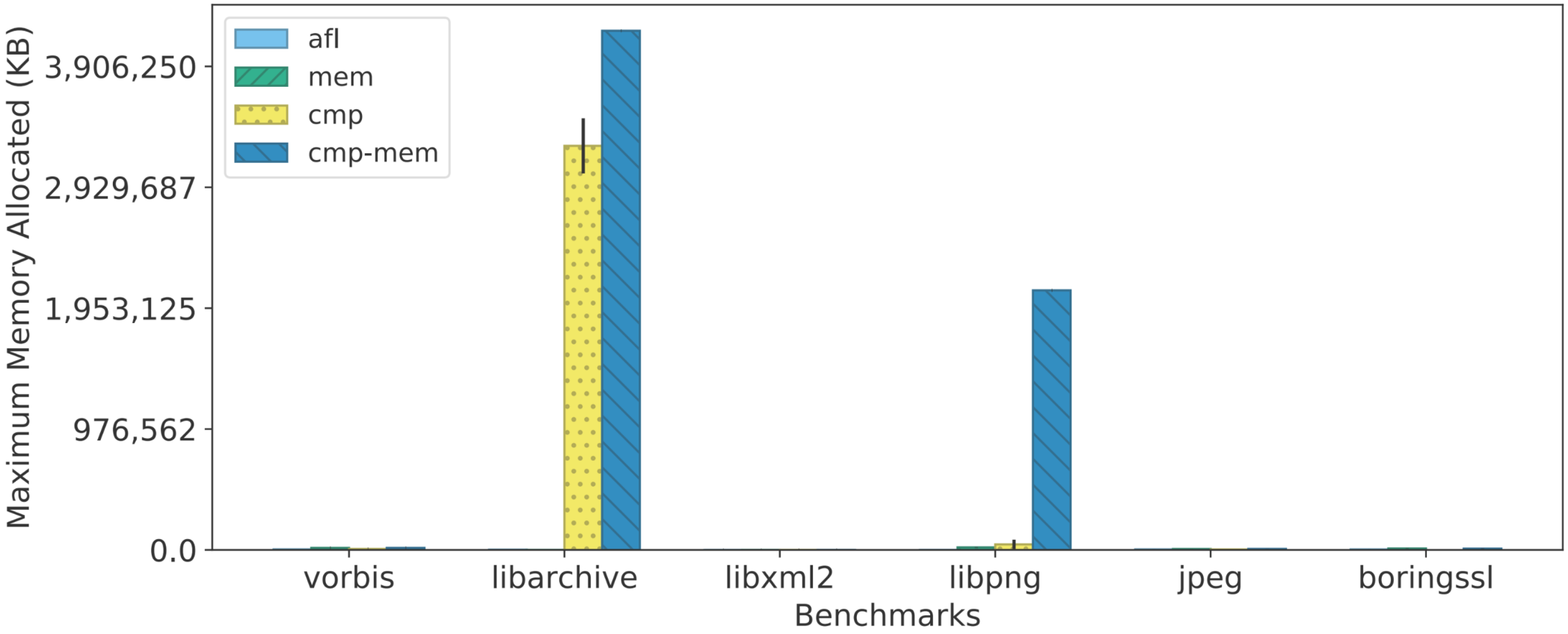
FuzzFactory, Step 2: Separate Algo & Feedback



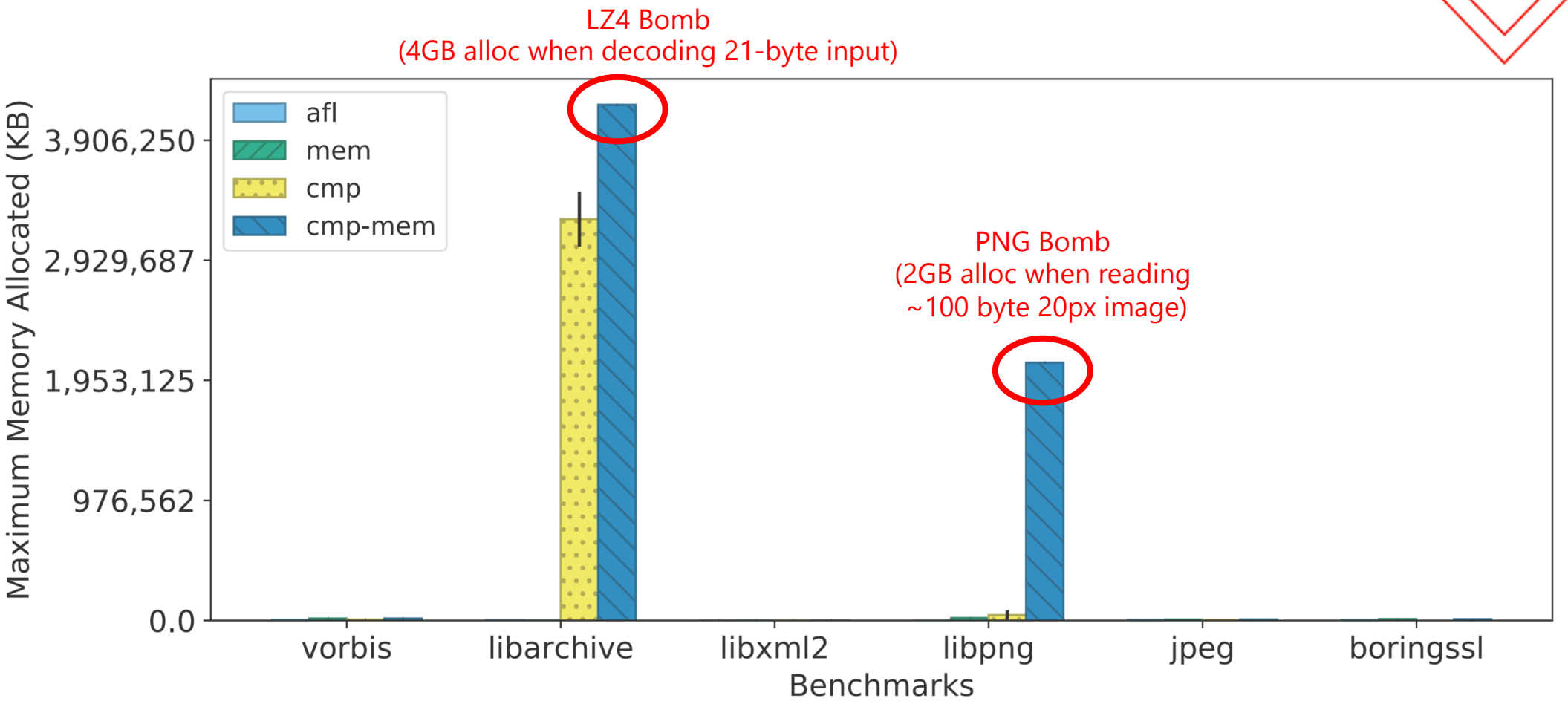
FuzzFactory, Step 2: Allows Easy *Composition*



Super-Fuzzer: CMP ∘ MEM

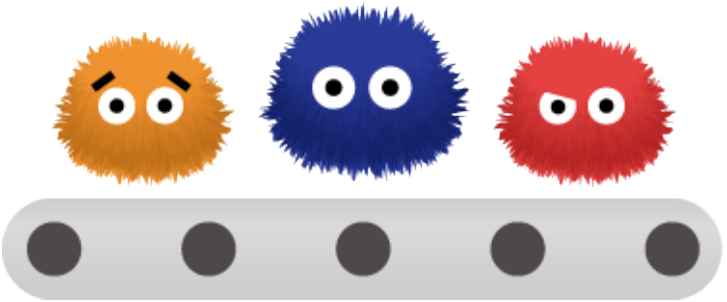


Super-Fuzzer: CMP ∘ MEM

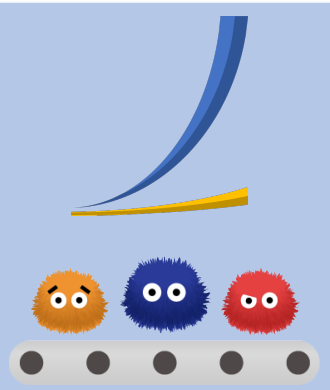


FuzzFactory

https://github.com/rohanpadhye/FuzzFactory



- Built on top of AFL
- Comes with `afl-showdsf` tool
- Use our domain-specific fuzzers or build your own



PerfFuzz
https://github.com/carolemieux/perffuzz

FuzzFactory
https://github.com/rohanpadhye/FuzzFactory

Deeper Exploration

Different Bugs 

Deeper Exploration 

Different Bugs 

Deeper Exploration 

Different Bugs

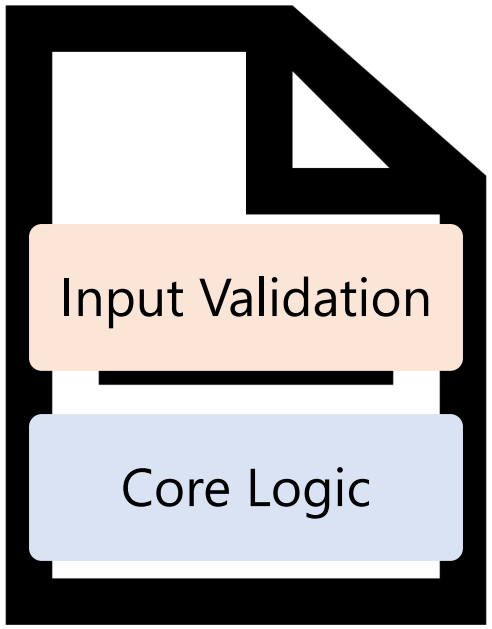
FairFuzz

https://github.com/carolemieux/afl-rb

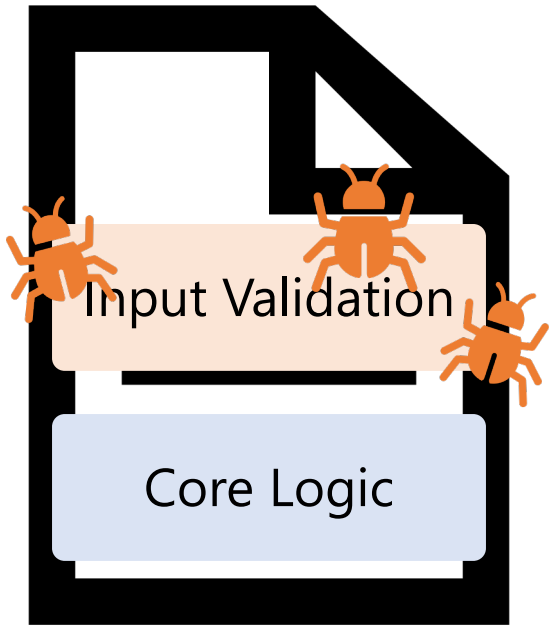
JQF/Zest

https://github.com/rohanpadhye/jqf

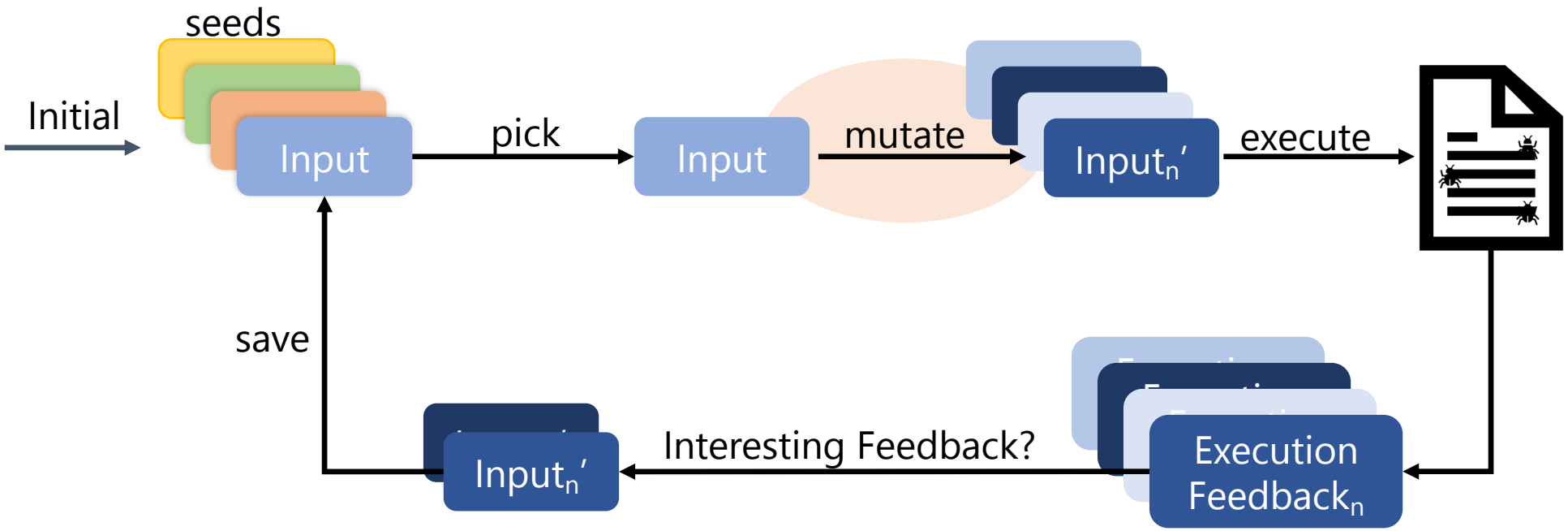
Where Are the Fuzzer-Found Bugs?



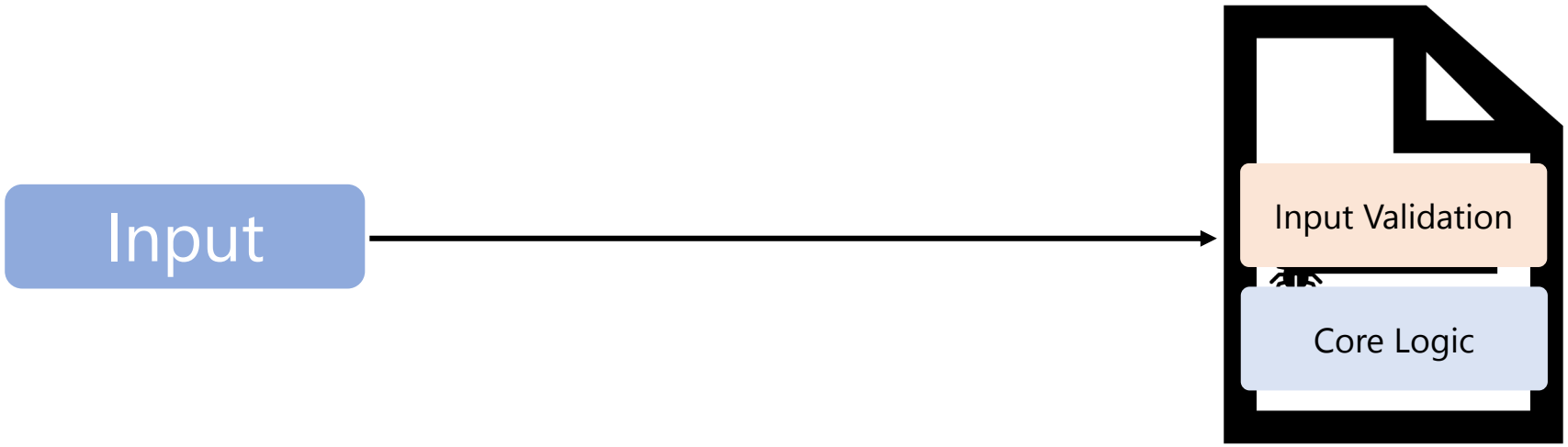
Where Are the Fuzzer-Found Bugs?



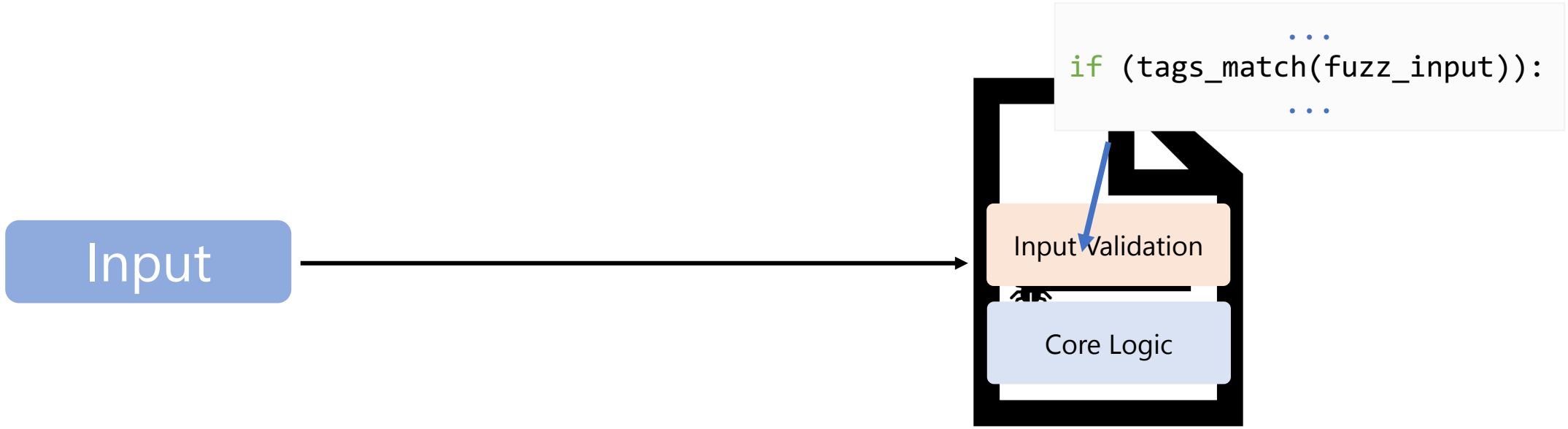
Coverage-Guided Fuzzing



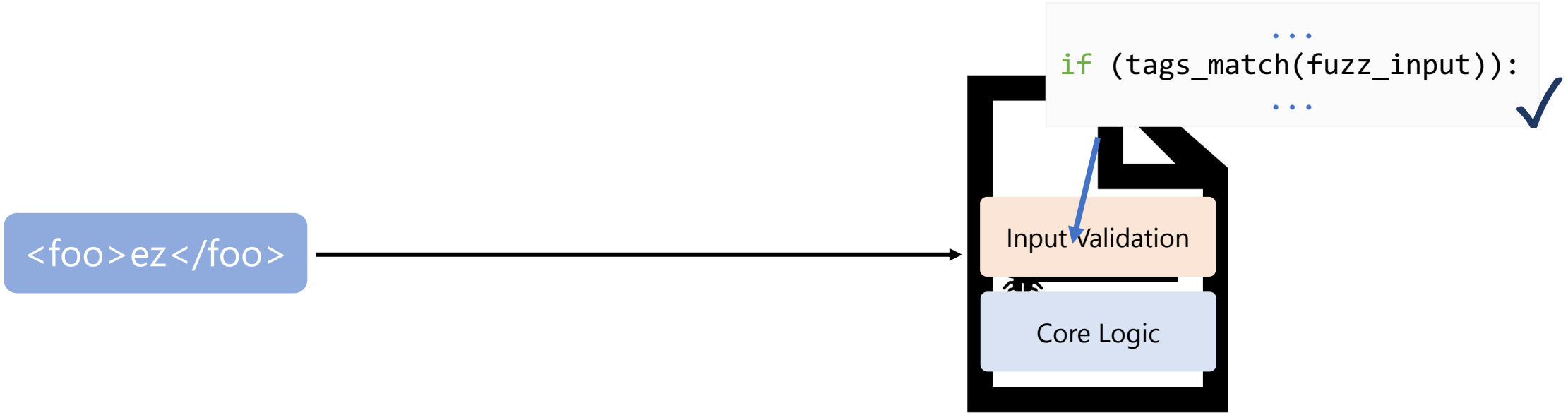
Branches Guard Core Logic



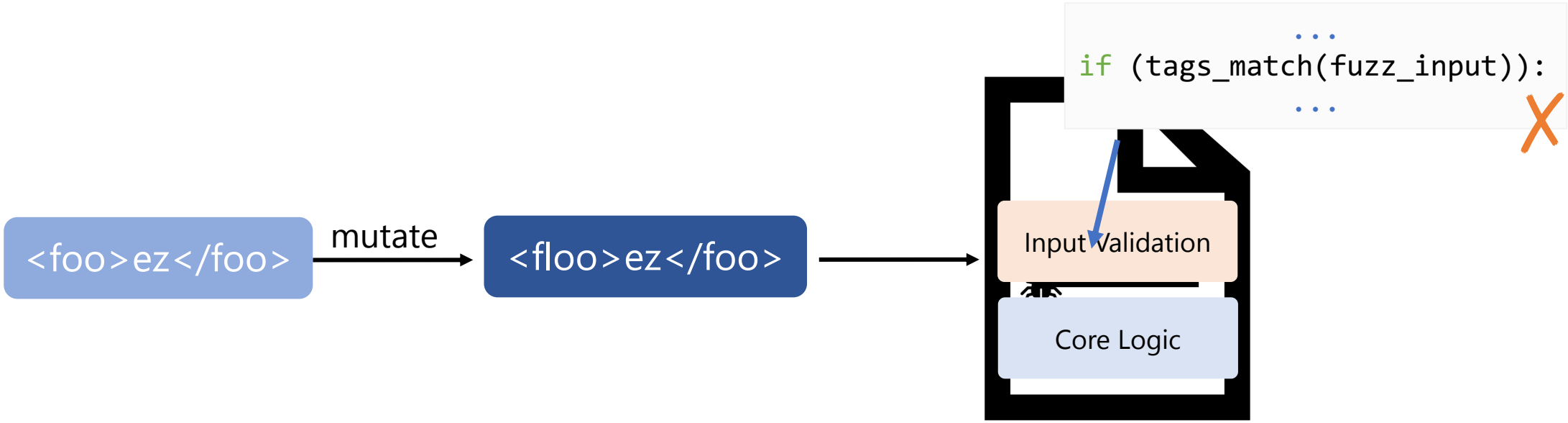
Branches Guard Core Logic



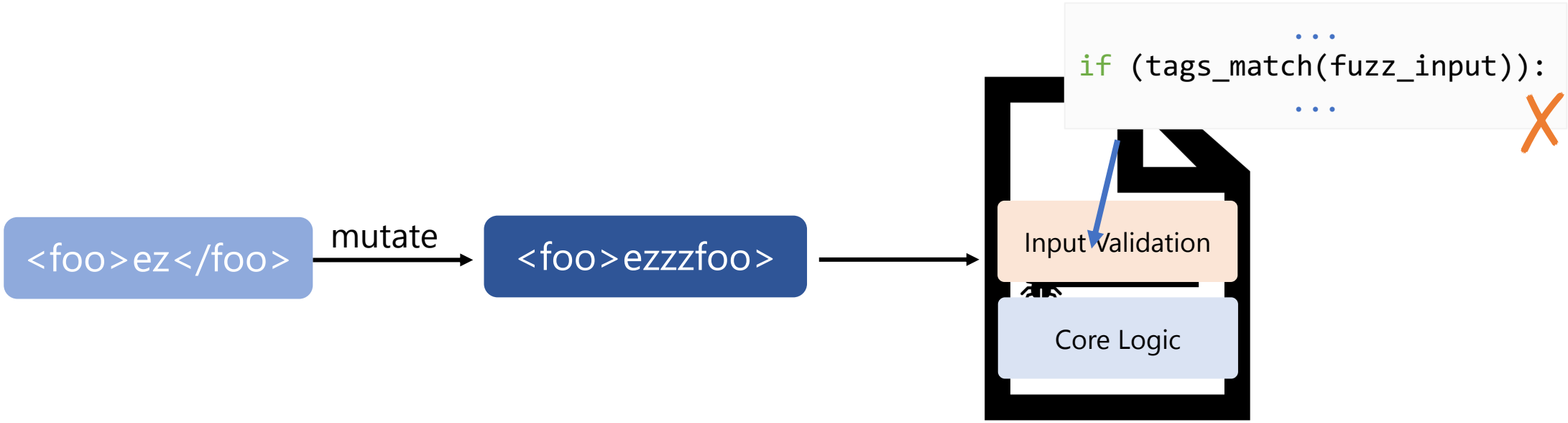
Branches Guard Core Logic



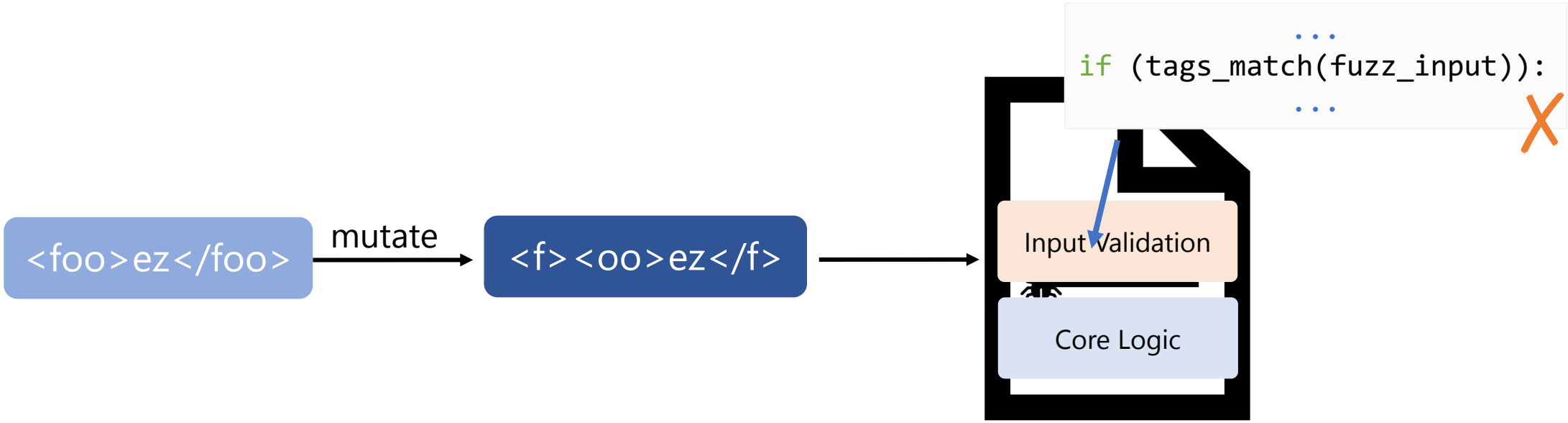
Some branches hard-to-hit with mutants



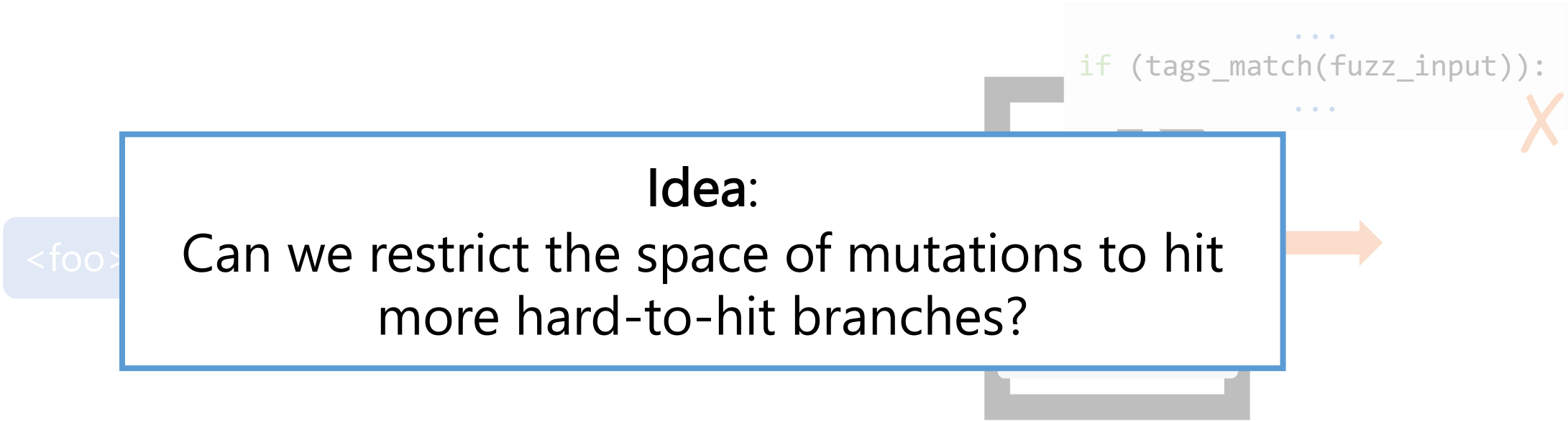
Some branches hard-to-hit with mutants



Some branches hard-to-hit with mutants



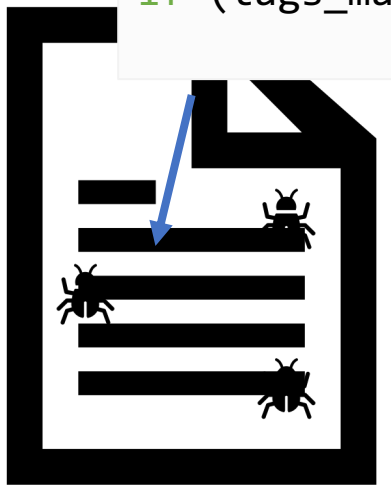
Some branches hard-to-hit with mutants



Idea:
Can we restrict the space of mutations to hit more hard-to-hit branches?

FairFuzz: Branch Mask Idea

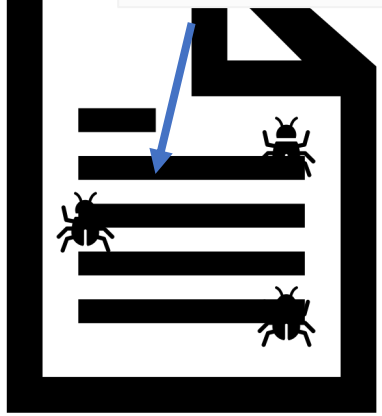
```
...
if (tags_match(fuzz_input)):
...
```



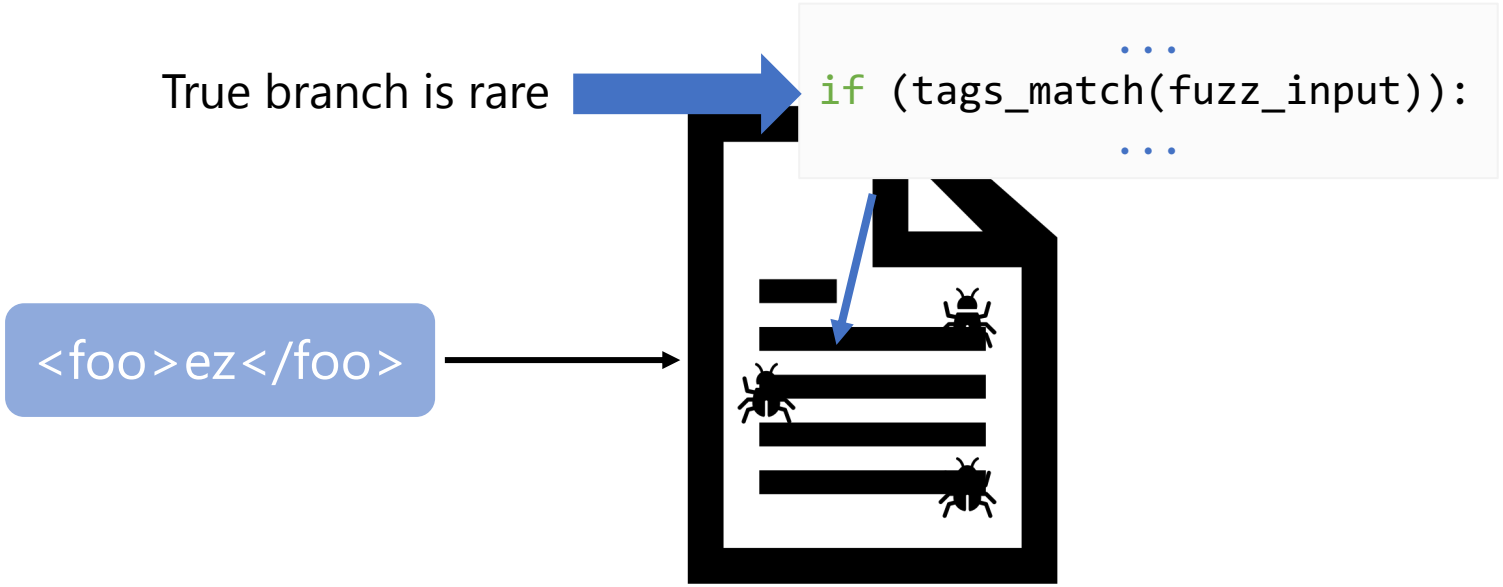
FairFuzz: Branch Mask Idea

True branch is rare

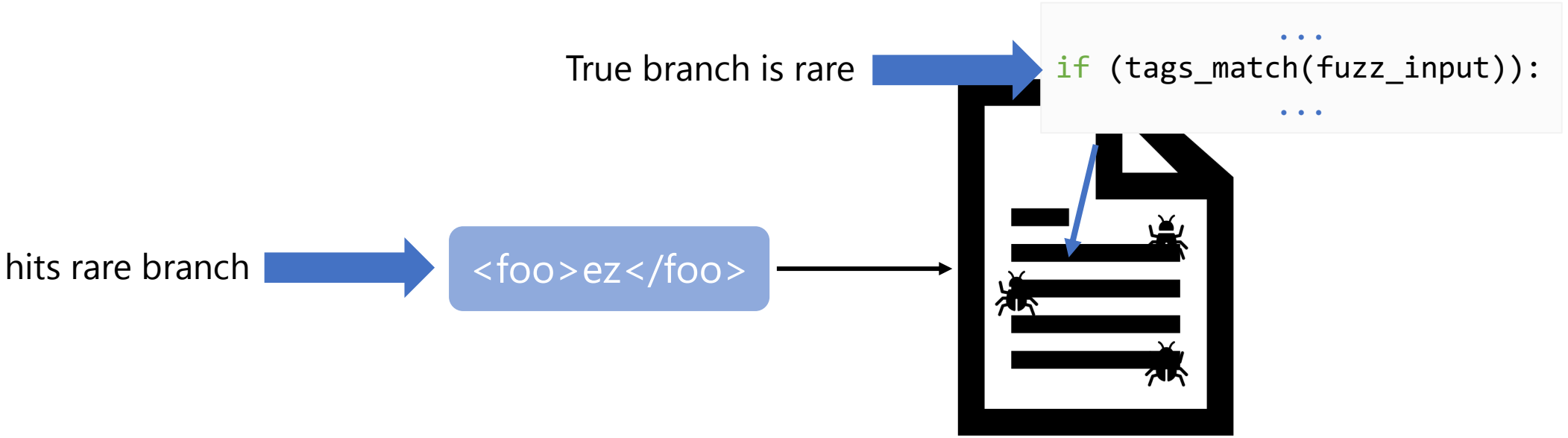
```
...
if (tags_match(fuzz_input)):
...
```



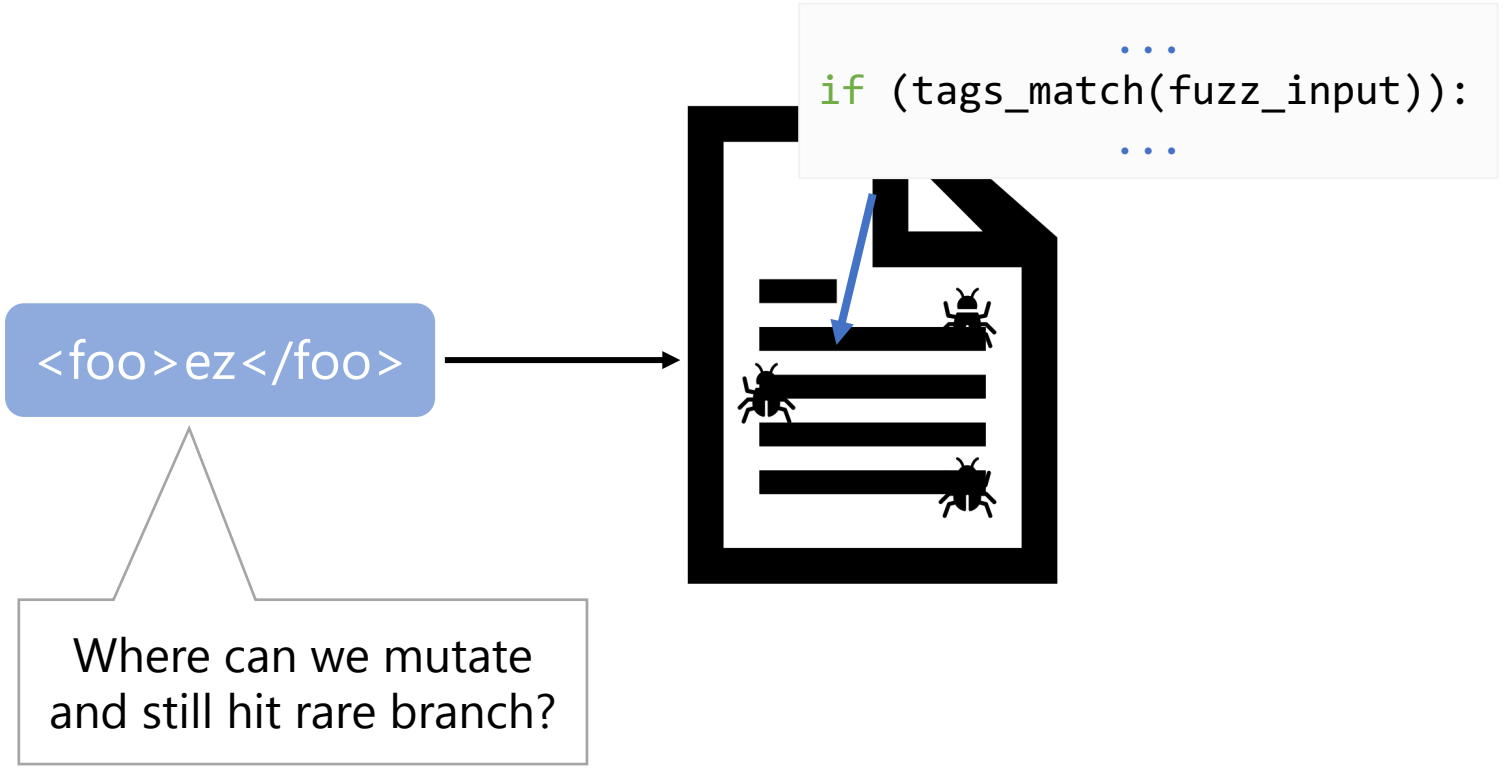
FairFuzz: Branch Mask Idea



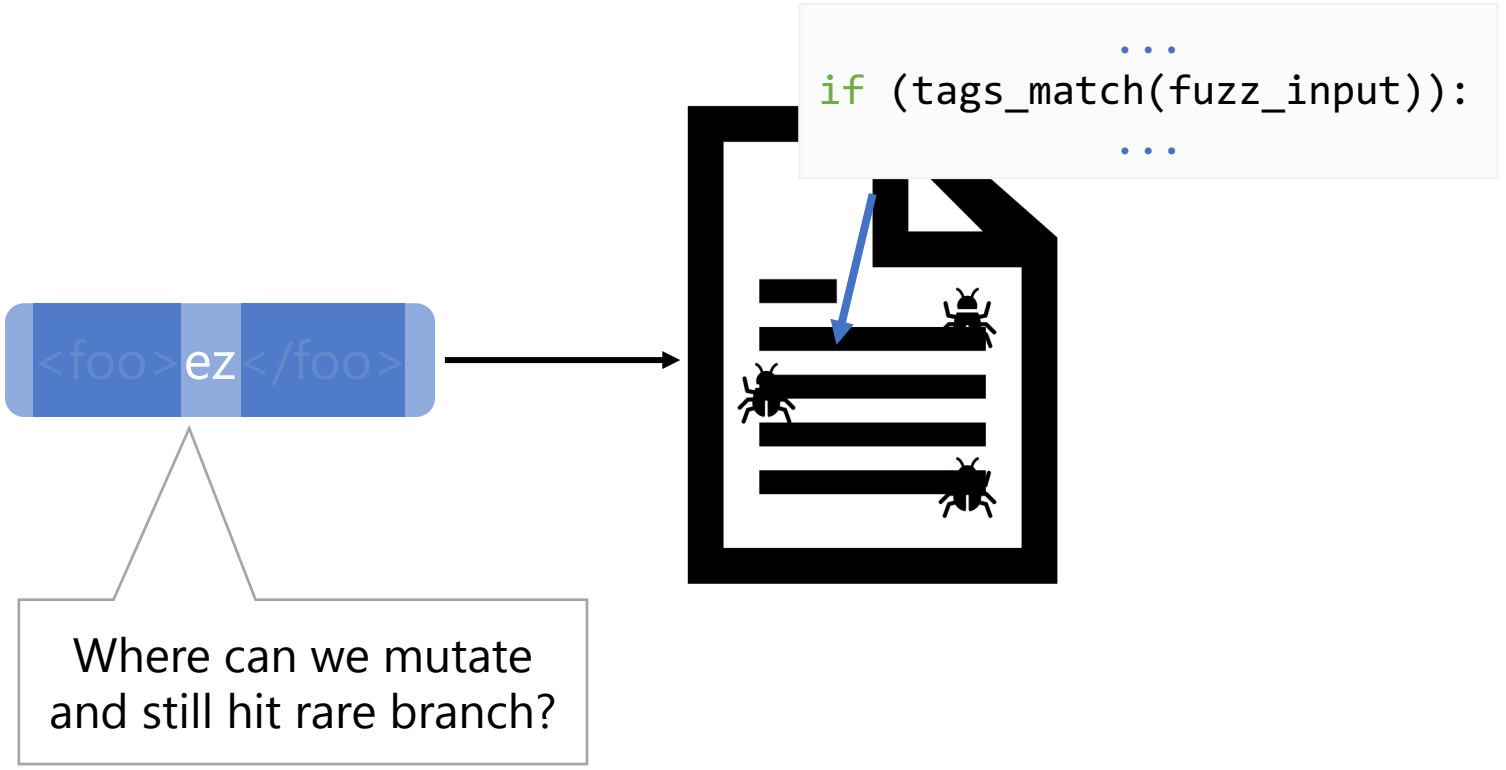
FairFuzz: Branch Mask Idea



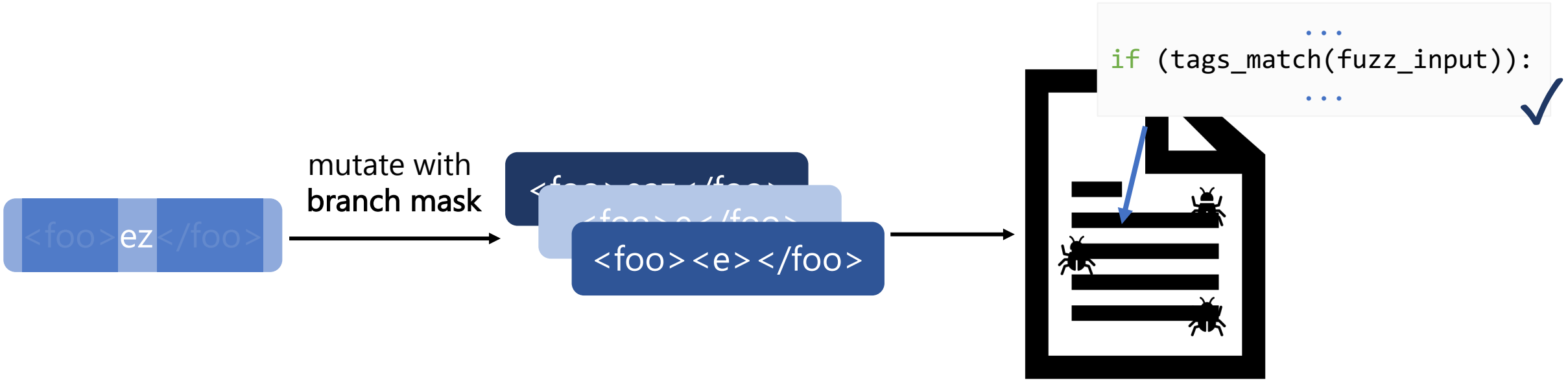
FairFuzz: Branch Mask Idea



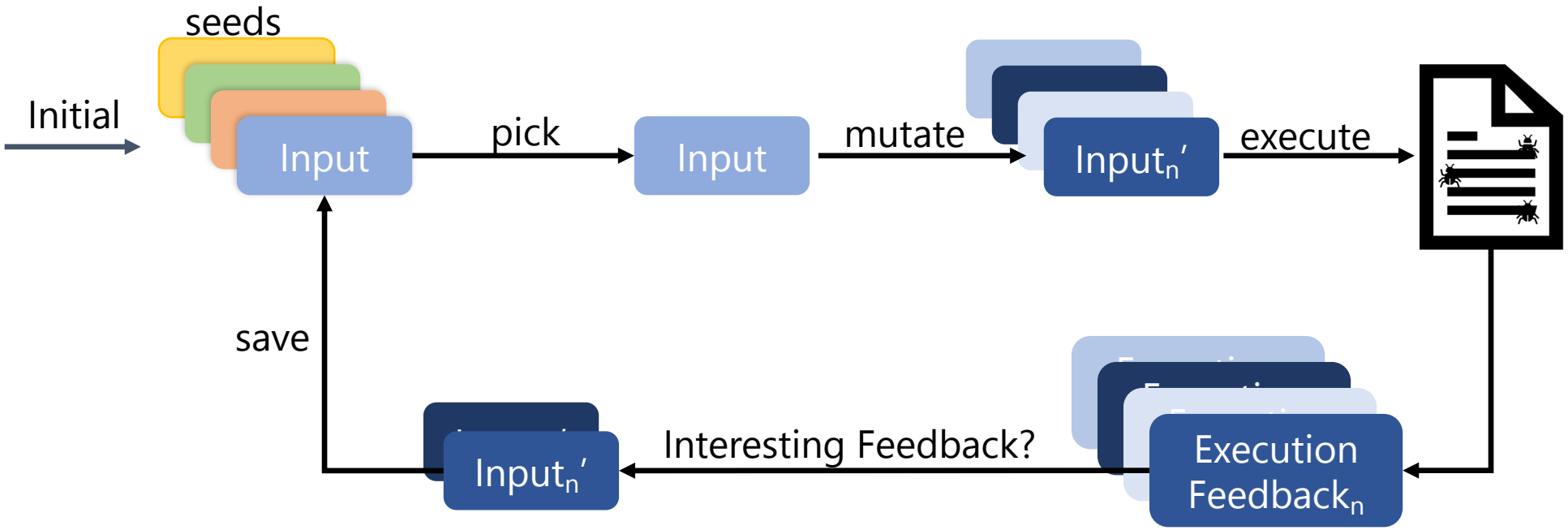
FairFuzz: Branch Mask Idea



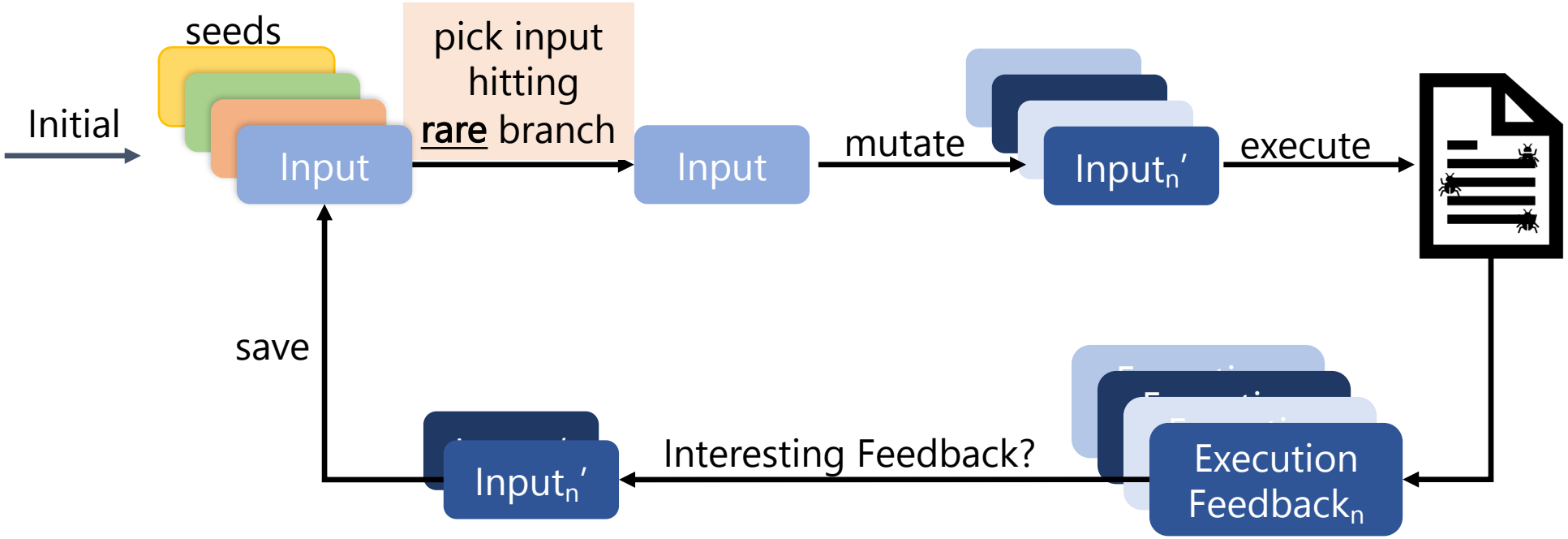
FairFuzz: Branch Mask Idea



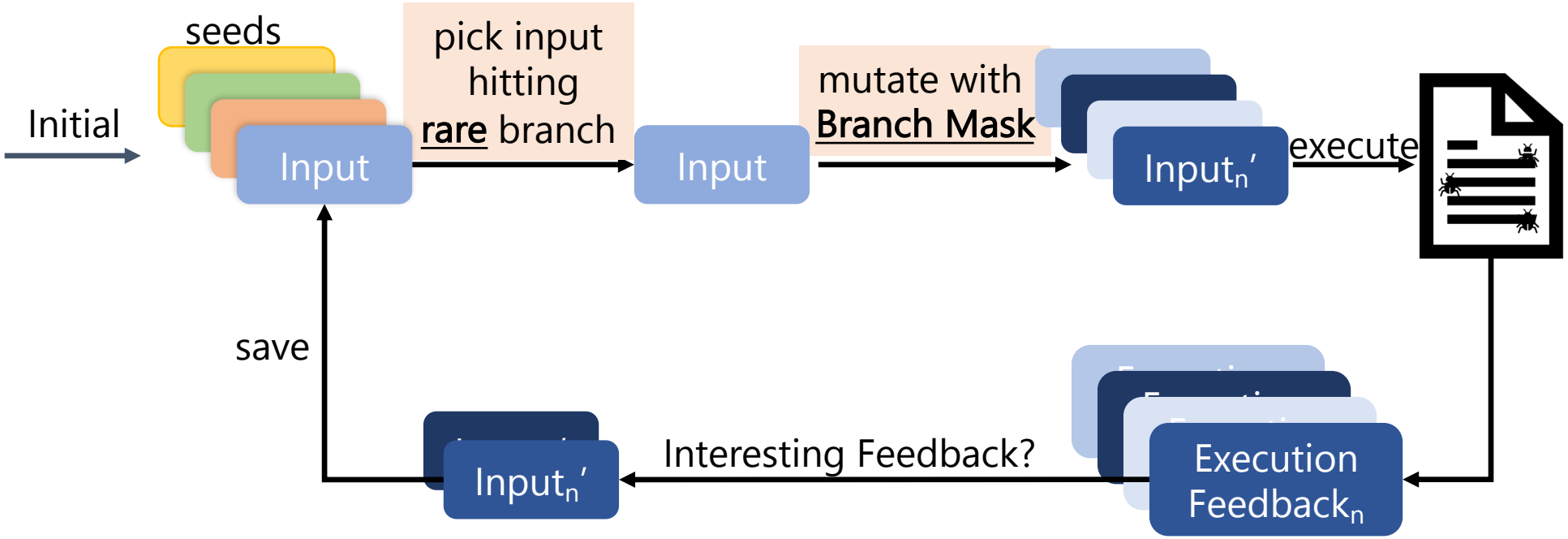
FairFuzz


seeds
Initial
Input
pick
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Interesting Feedback?
Execution Feedback$_n$

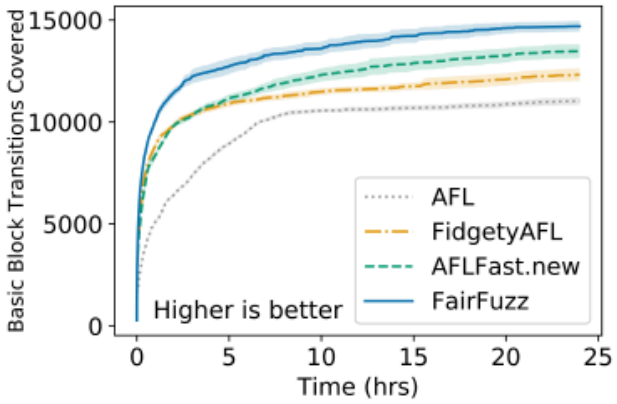
FairFuzz


seeds
Initial
Input
pick input hitting **rare** branch
Input
mutate
Input$_n$'
execute
save
Input$_n$'
Interesting Feedback?
Execution Feedback$_n$

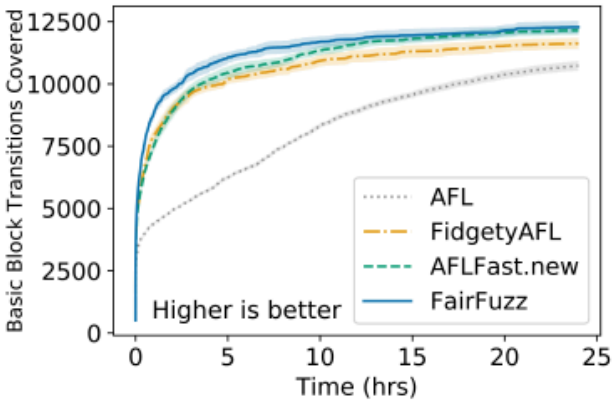
FairFuzz


seeds
Initial
Input
pick input hitting **rare** branch
Input
mutate with **Branch Mask**
Input$_n$'
execute
save
Input$_n$'
Interesting Feedback?
Execution Feedback$_n$

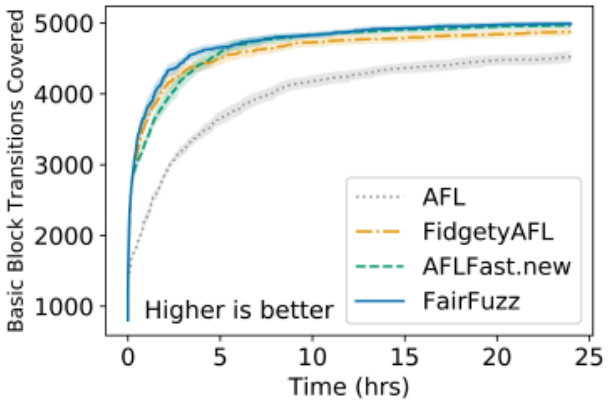
FairFuzz Eval: Branch Coverage



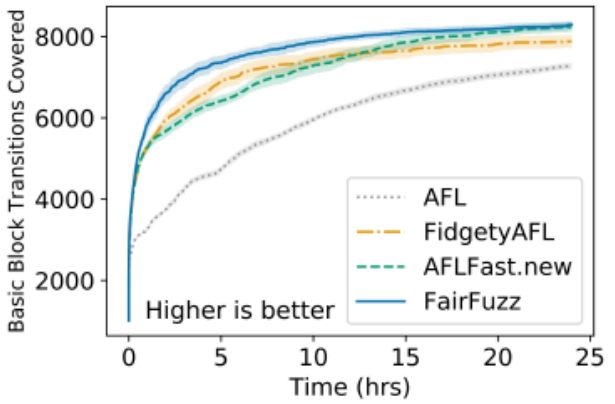
(a) tcpdump



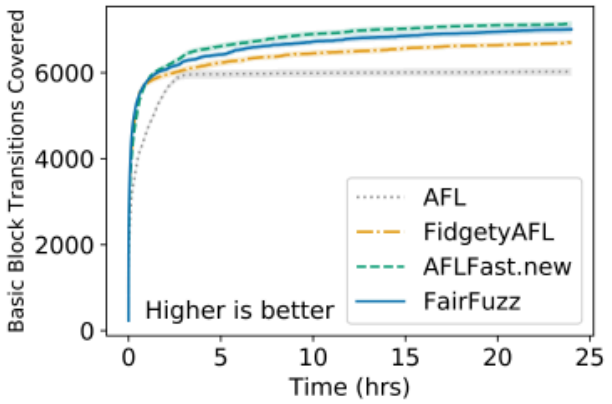
(b) readelf



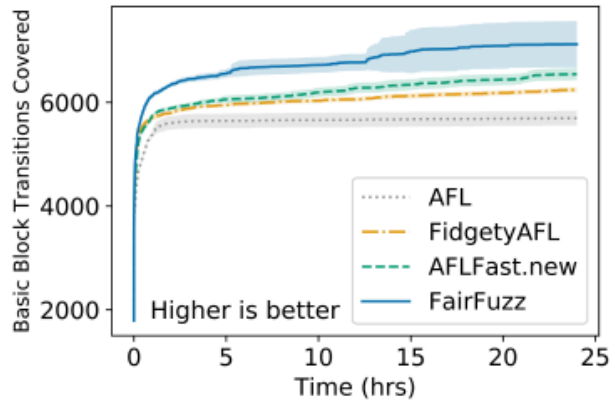
(c) nm



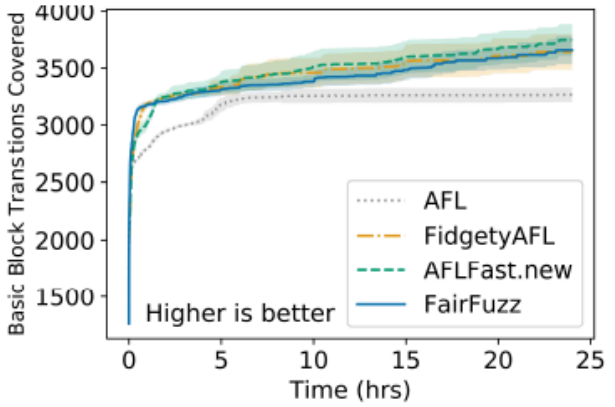
(d) objdump



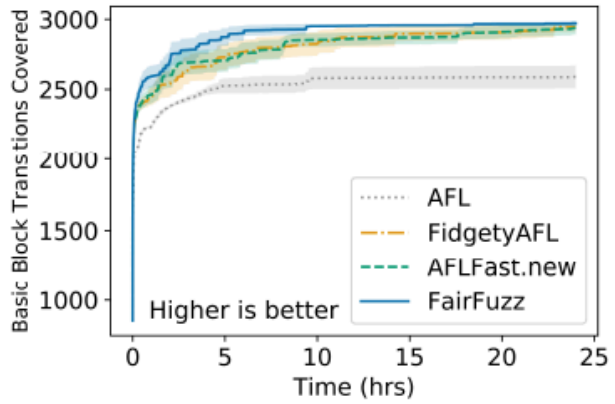
(e) c++filt



(f) xmllint

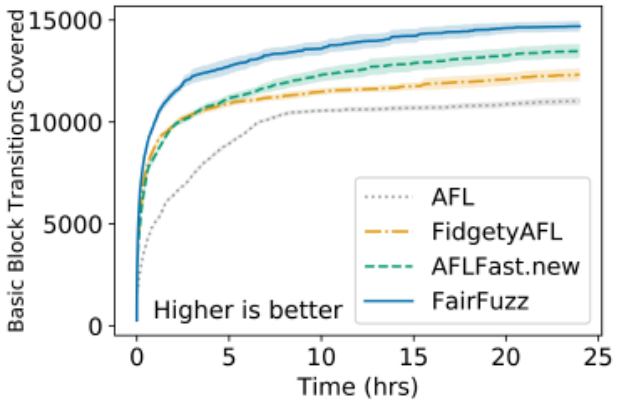


(h) djpeg

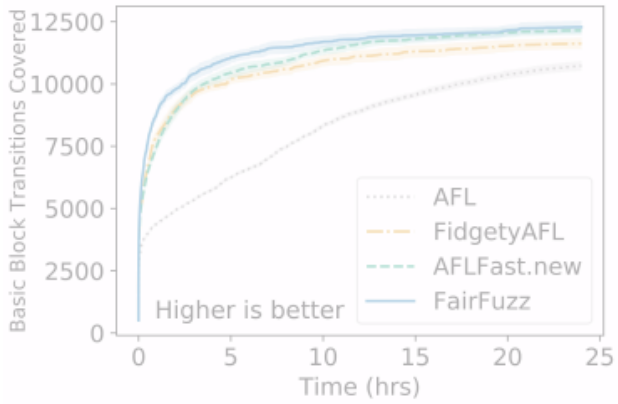


(i) readpng

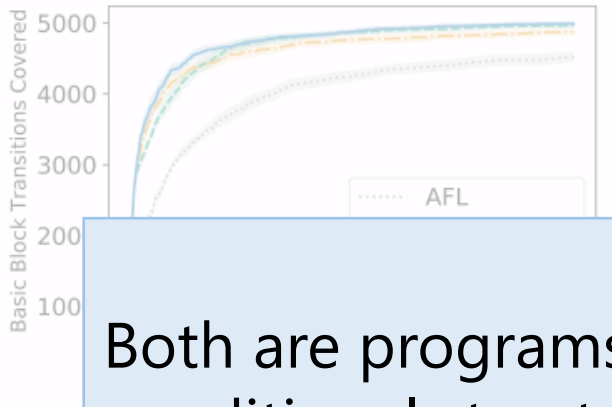
Where Does FairFuzz Perform Much Better?

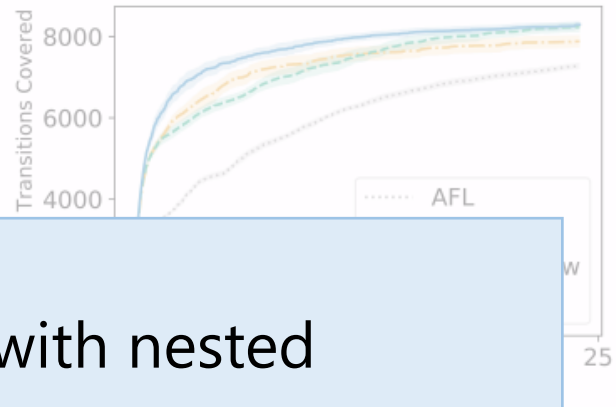


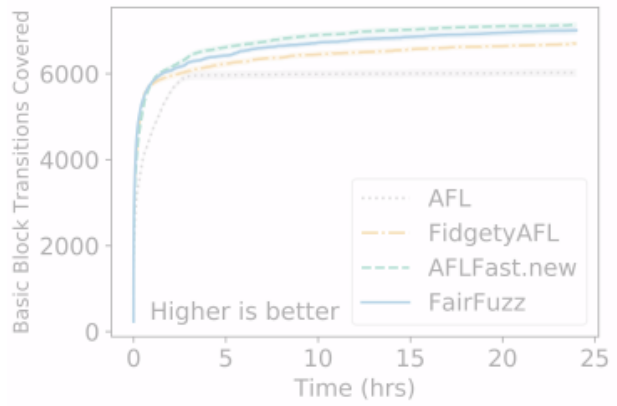
(a) tcpdump



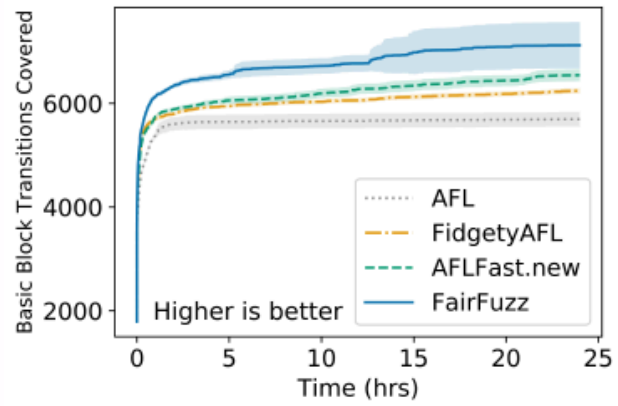
(b) readelf



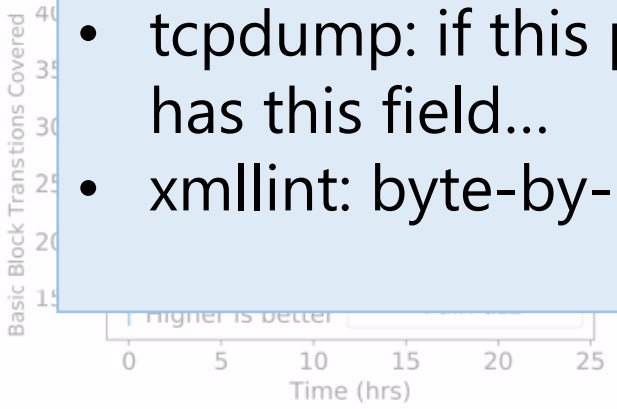




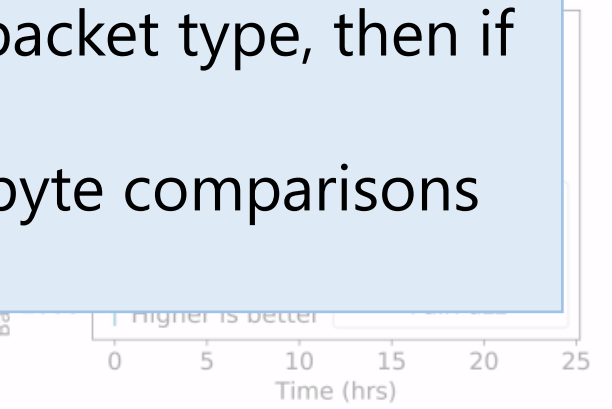
(e) c++filt



(f) xmllint



(h) djpeg



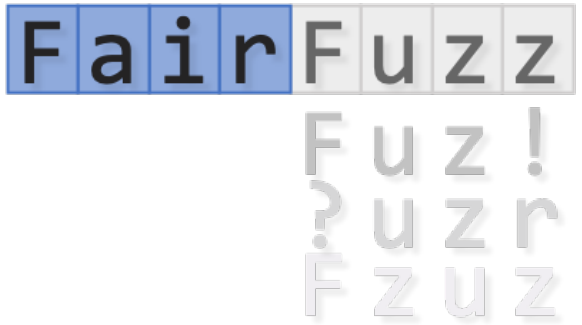
(i) readpng

Both are programs with nested conditional structure

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

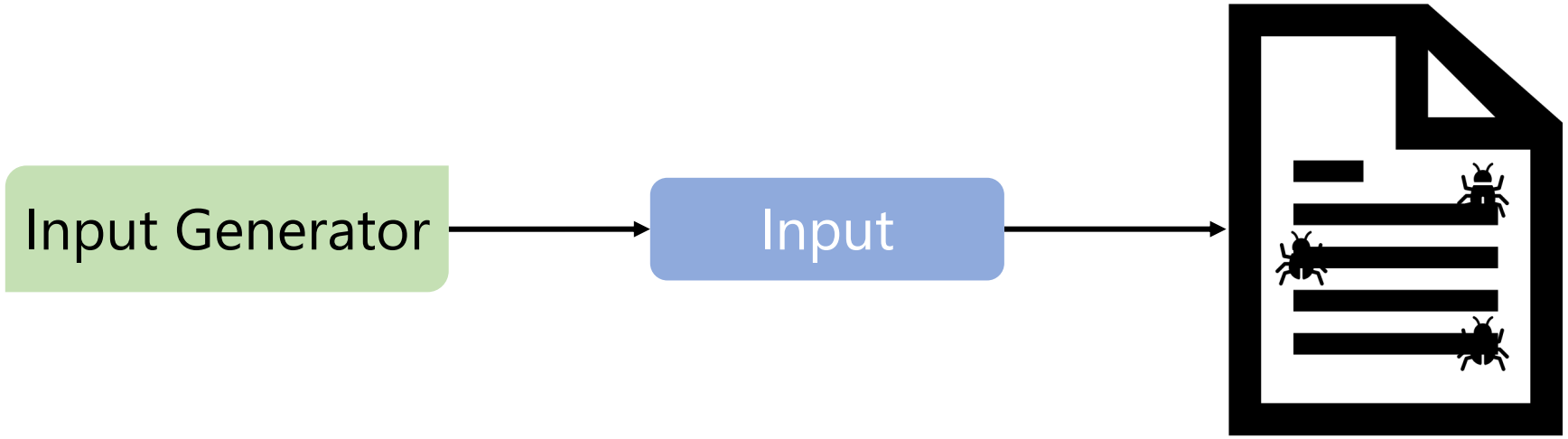
FairFuzz

https://github.com/carolemieux/afl-rb

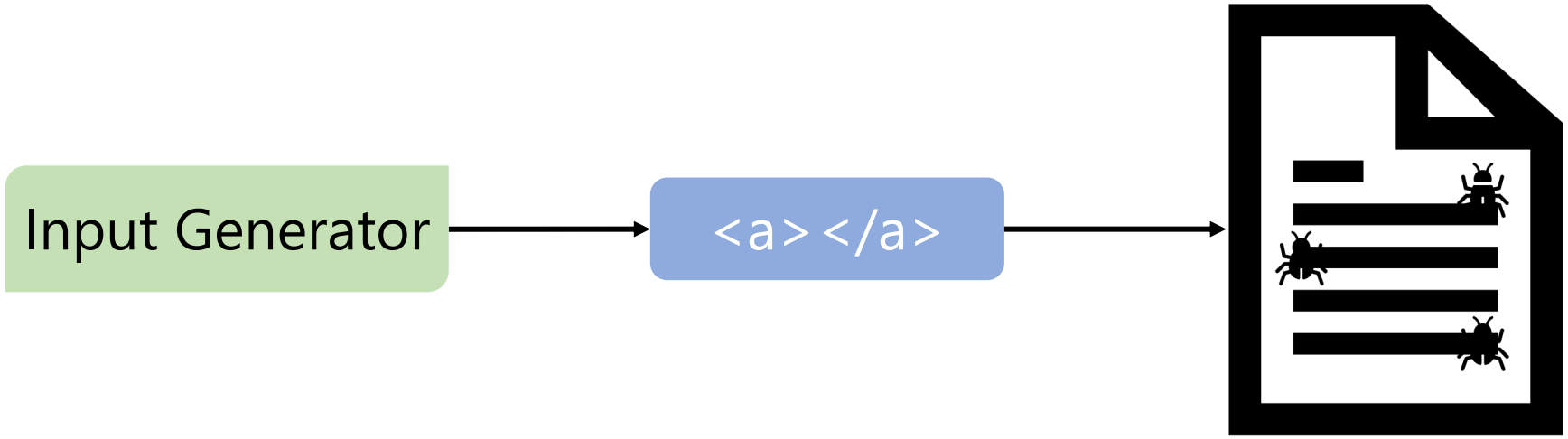


- Built on top of AFL
- Works with any AFL instrumentation
- Especially powerful in ensemble fuzzing

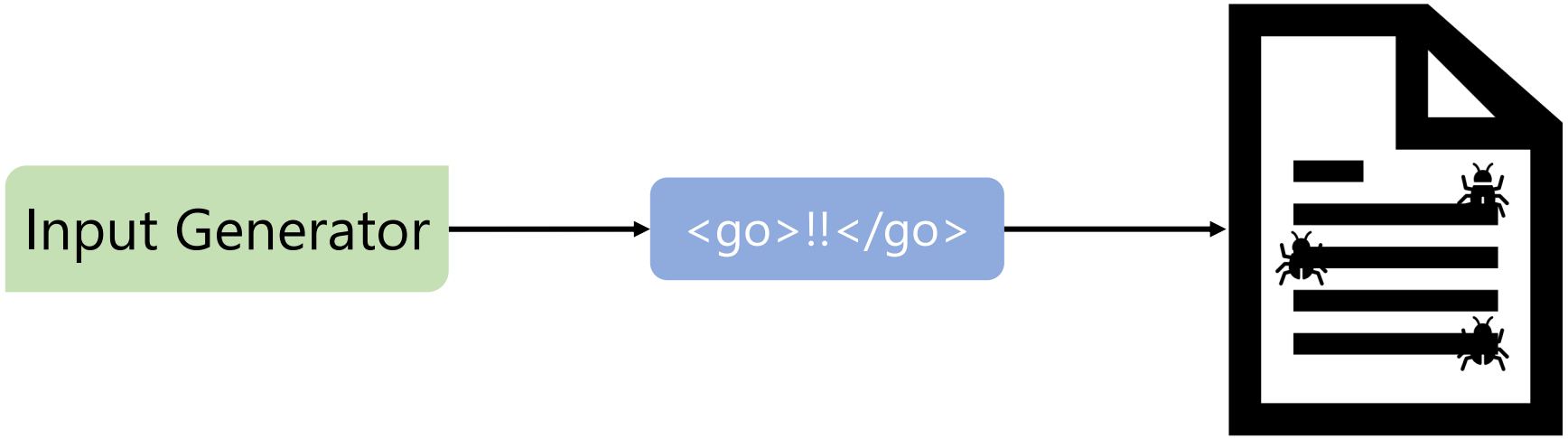
Generator-Based Fuzzing



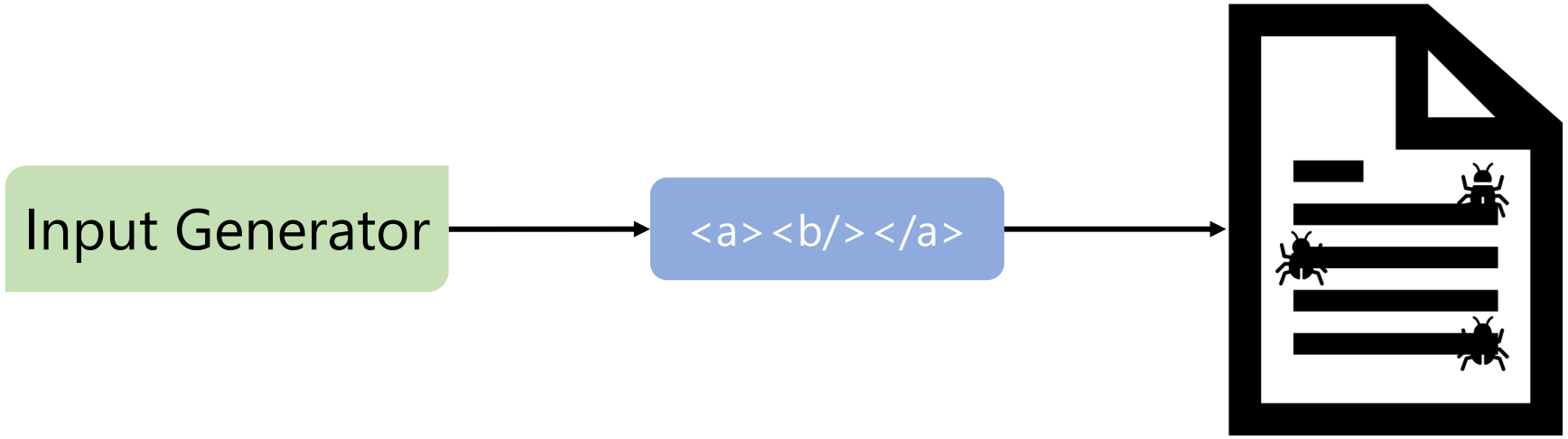
Generator-Based Fuzzing



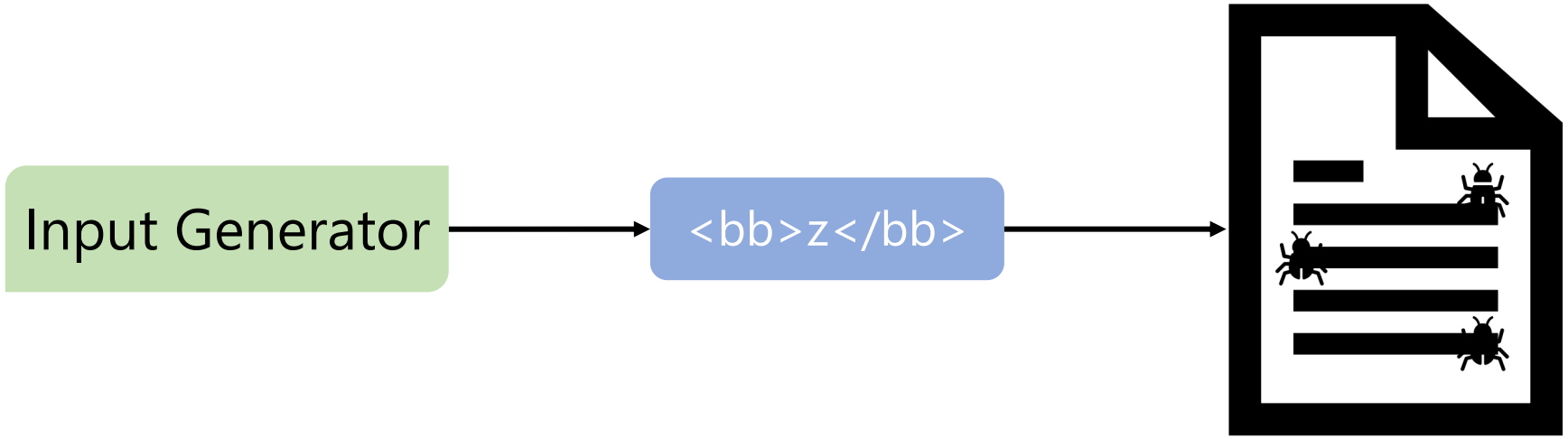
Generator-Based Fuzzing



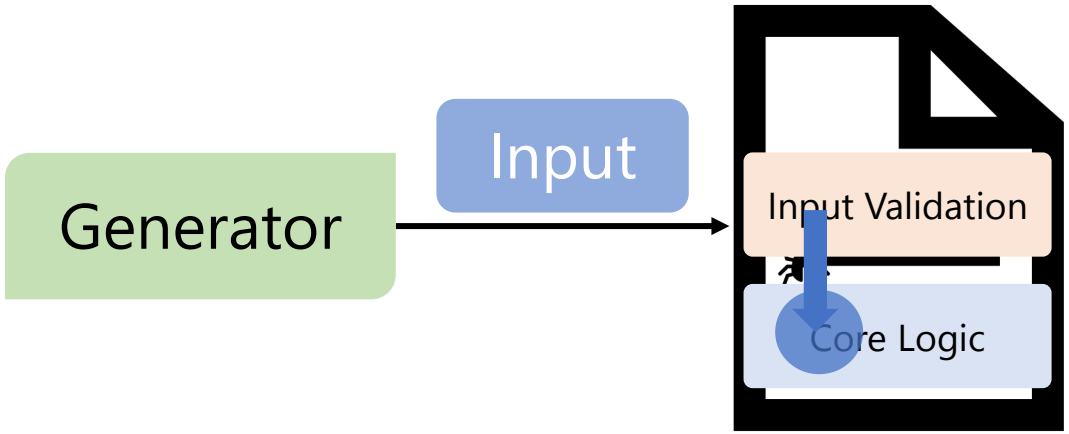
Generator-Based Fuzzing



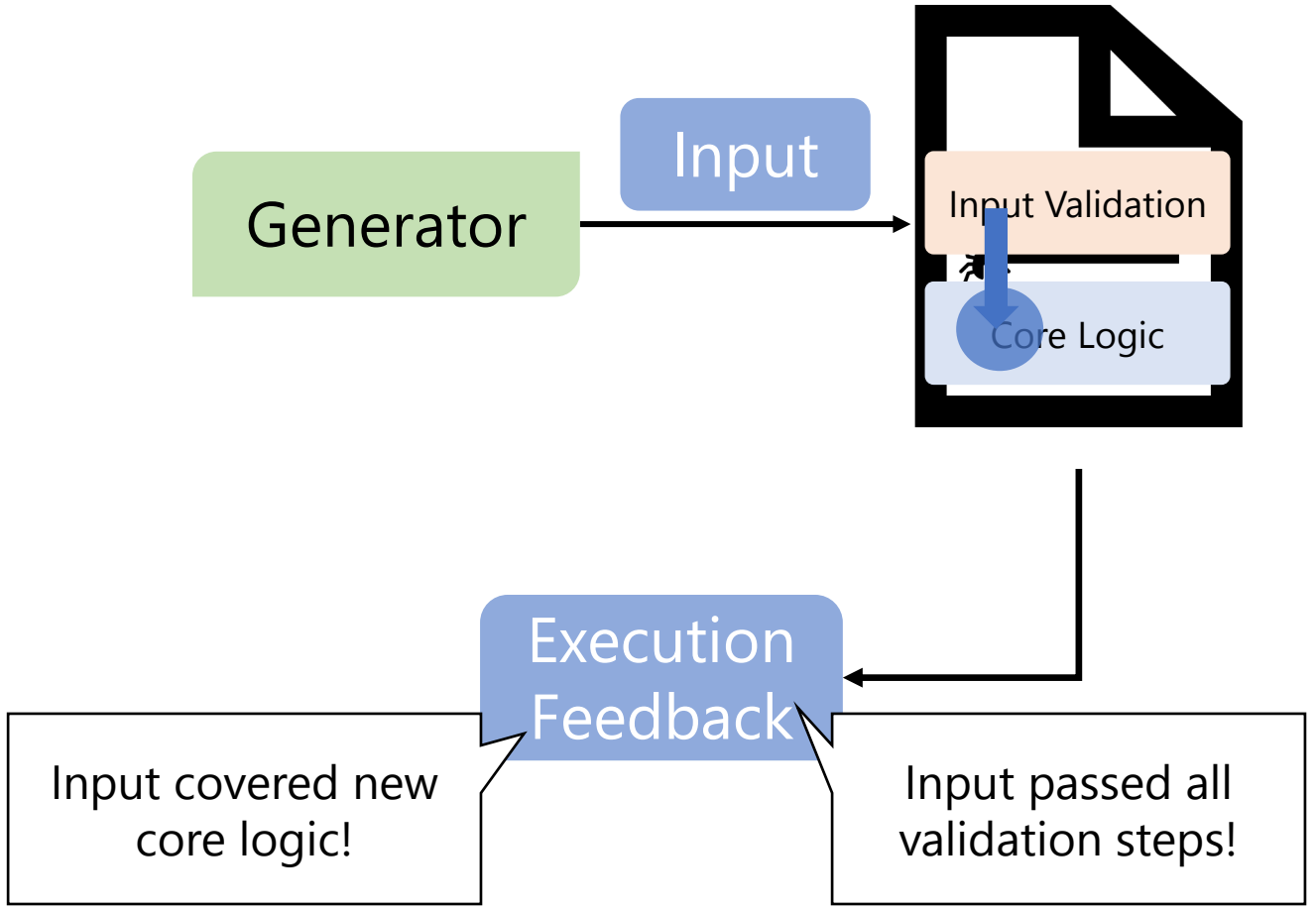
Generator-Based Fuzzing



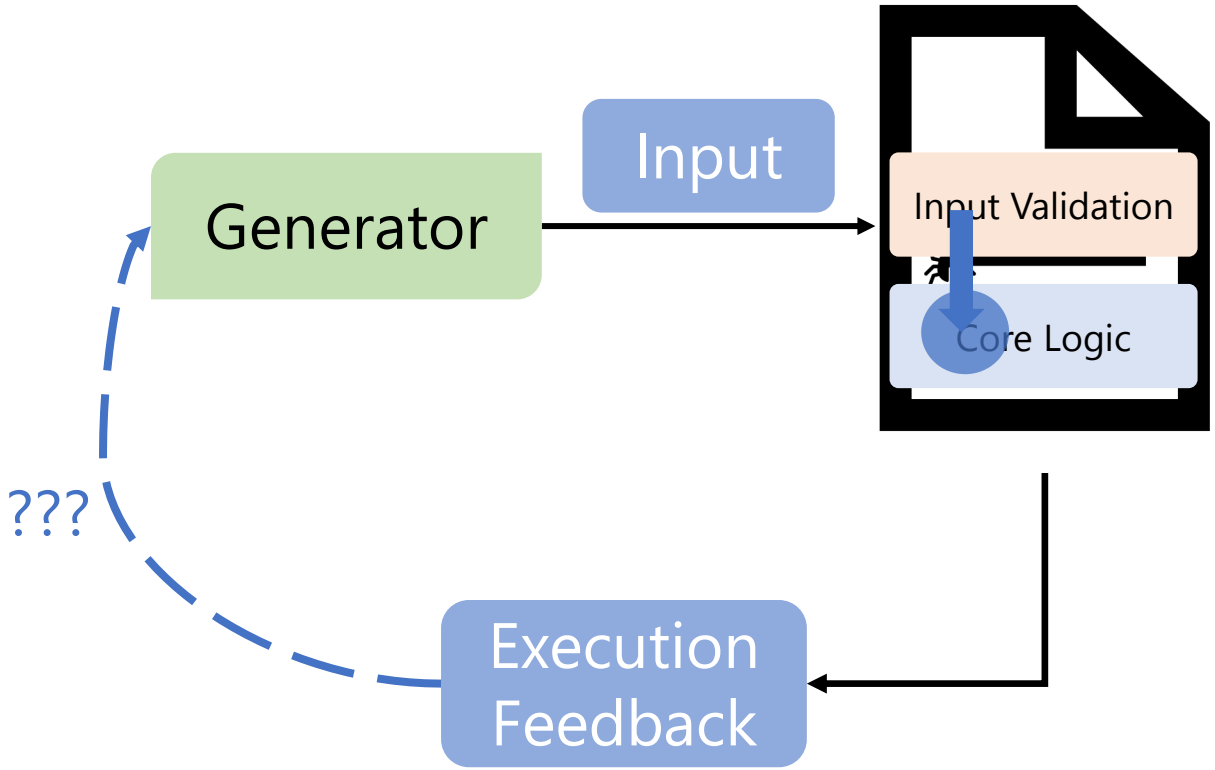
Generator-Based Fuzzing: Get "Deeper"



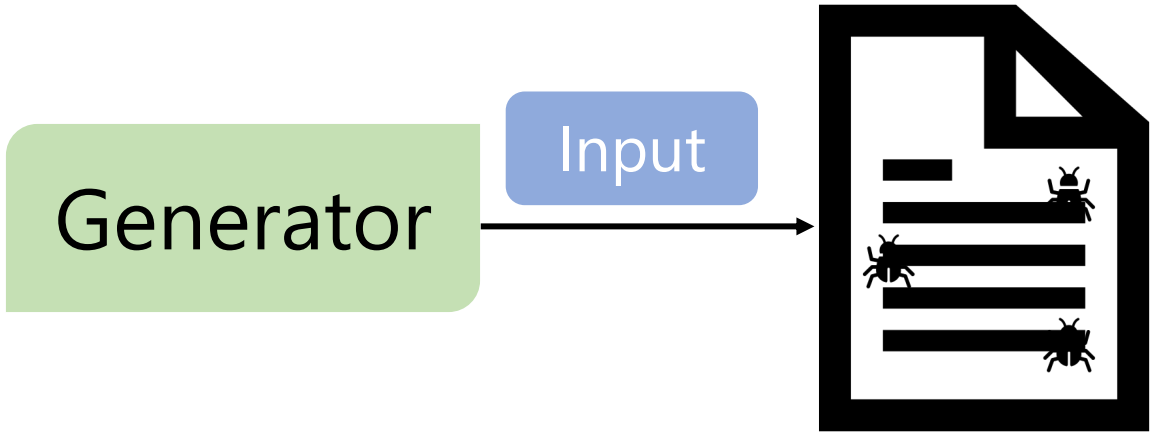
Generator-Based Fuzzing: Drawbacks



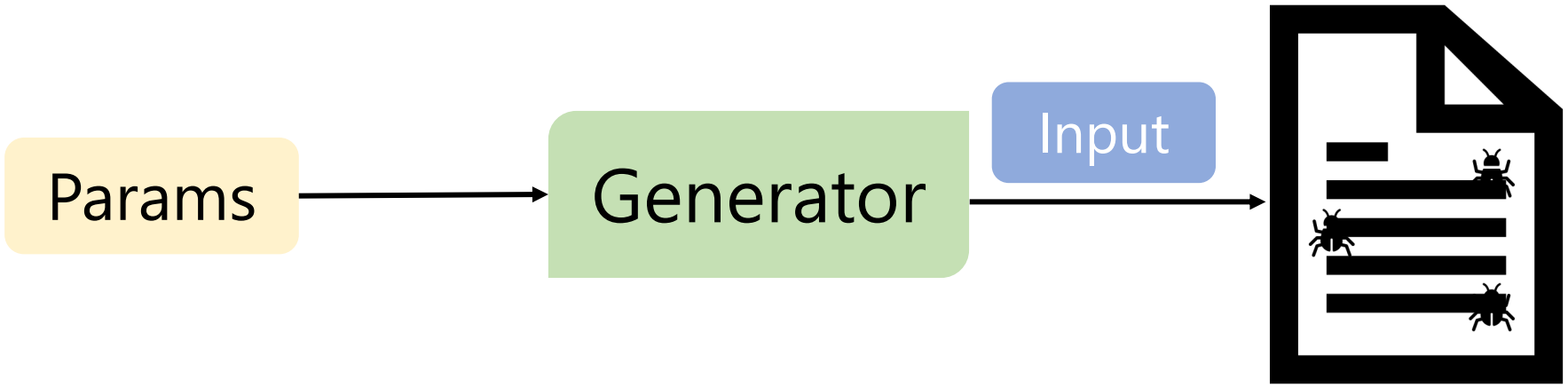
Generator-Based Fuzzing: Drawbacks



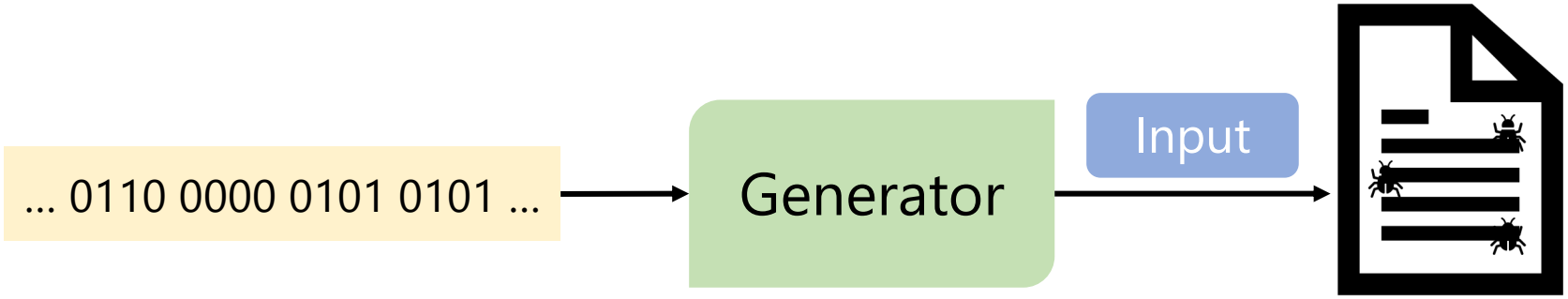
Parametric Generators: Explicitly Pass in Stream of Bit "Parameters"



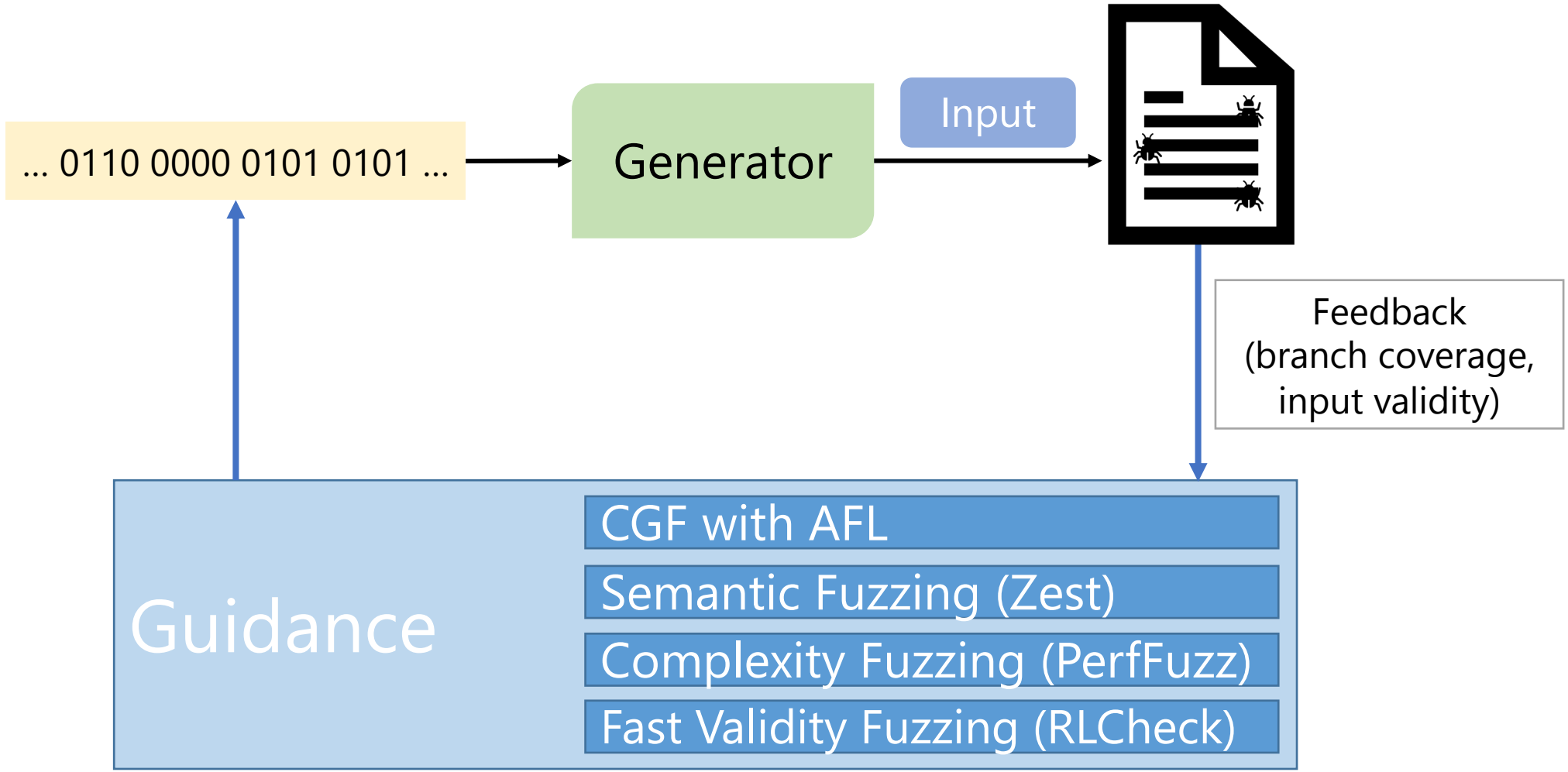
Parametric Generators: Explicitly Pass in Stream of Bit "Parameters"



JQF: Framework for Guided Generator-Based Fuzzing



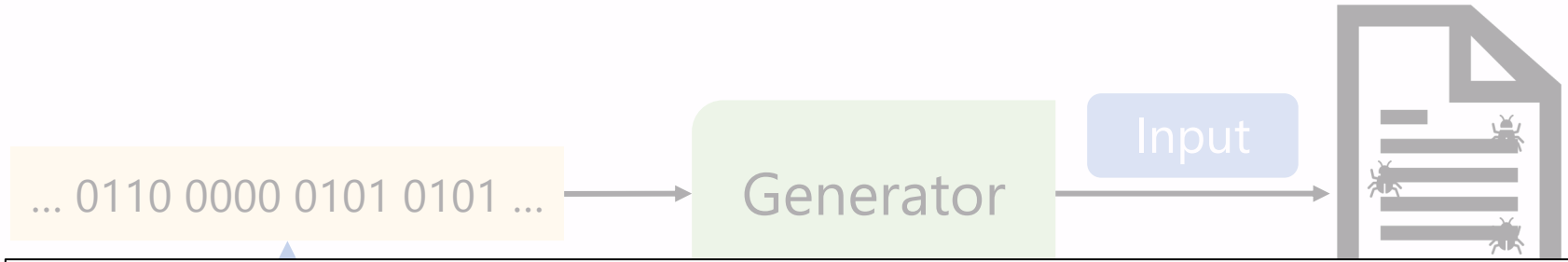
JQF: Framework for Guided Generator-Based Fuzzing



JQF: Input Stream Fuzzing with AFL

```
@Fuzz /* JQF will generate inputs to this method */
public void testRead(InputStream input) {
        // Create parser
        ImageReader reader = ImageIO.getImageReadersByFormatName("png").next();
        // Decode image from input stream
        try {
                reader.setInput(ImageIO.createImageInputStream(input));
                // Bound dimensions to avoid OOM
                Assume.assumeTrue(reader.getHeight(0) <= 256);
                Assume.assumeTrue(reader.getWidth(0) <= 256);
                // Decode first image in the input stream
                reader.read(0);
        } catch (IOException e) {
        // This exception signals invalid input and not a test failure
                Assume.assumeNoException(e);
        }
}
```

JQF: Generator-Based Validity Fuzzing with Zest




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

Guidance

- CGF with AFL
- Semantic Fuzzing (Zest)
- Complexity Fuzzing (PerfFuzz)
- Fast Validity Fuzzing (RLCheck)

JQF: Framework for Guided Generator-Based Fuzzing

https://github.com/rohanpadhye/jqf

- Robust Java Fuzzer w/ JUnit interface
- Can fuzz functions that take in objects!
- Our best trophy case

eedback
ch coverage,
ut validity)

Guidance

Semantic Fuzzing (Zest)

Complexity Fuzzing (PerfFuzz)

Fast Validity Fuzzing (RLCheck)

JQF: Framework for Guided Generator-Based Fuzzing

https://githu

- Robust Ja
- Can fuzz
- Our best

Guidance

🏆 Bug Trophy Case (Sep 2020) 🏆

Google Closure Compiler: #2842, #2843, #3220, #3173

OpenJDK: JDK-8190332, JDK-8190511, JDK-8190512, JDK-8190997, JDK-8191023, JDK-8191076, JDK-8191109, JDK-8191174, JDK-8191073, JDK-8193444, JDK-8193877

Apache Ant: #62655

Apache Maven: MNG-6374, MNG-6375, MNG-6577

Apache Commons: LANG-1385, COMPRESS-424, COLLECTIONS-714, **CVE-2018-11771**

Apache PDFBox: PDFBOX-4333, PDFBOX-4338, PDFBOX-4339, **CVE-2018-8036**

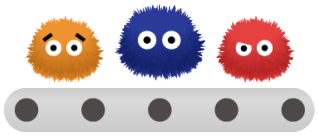
Apache TIKA: **CVE-2018-8017**, **CVE-2018-12418**

Apache BCEL: BCEL-303, BCEL-307, BCEL-308, BCEL-309, BCEL-310, BCEL-311, BCEL-312, BCEL-313

Mozilla Rhino: #405, #406, #407, #409, #410

Thanks for listening!

PerfFuzz
https://github.com/carolemieux/perffuzz

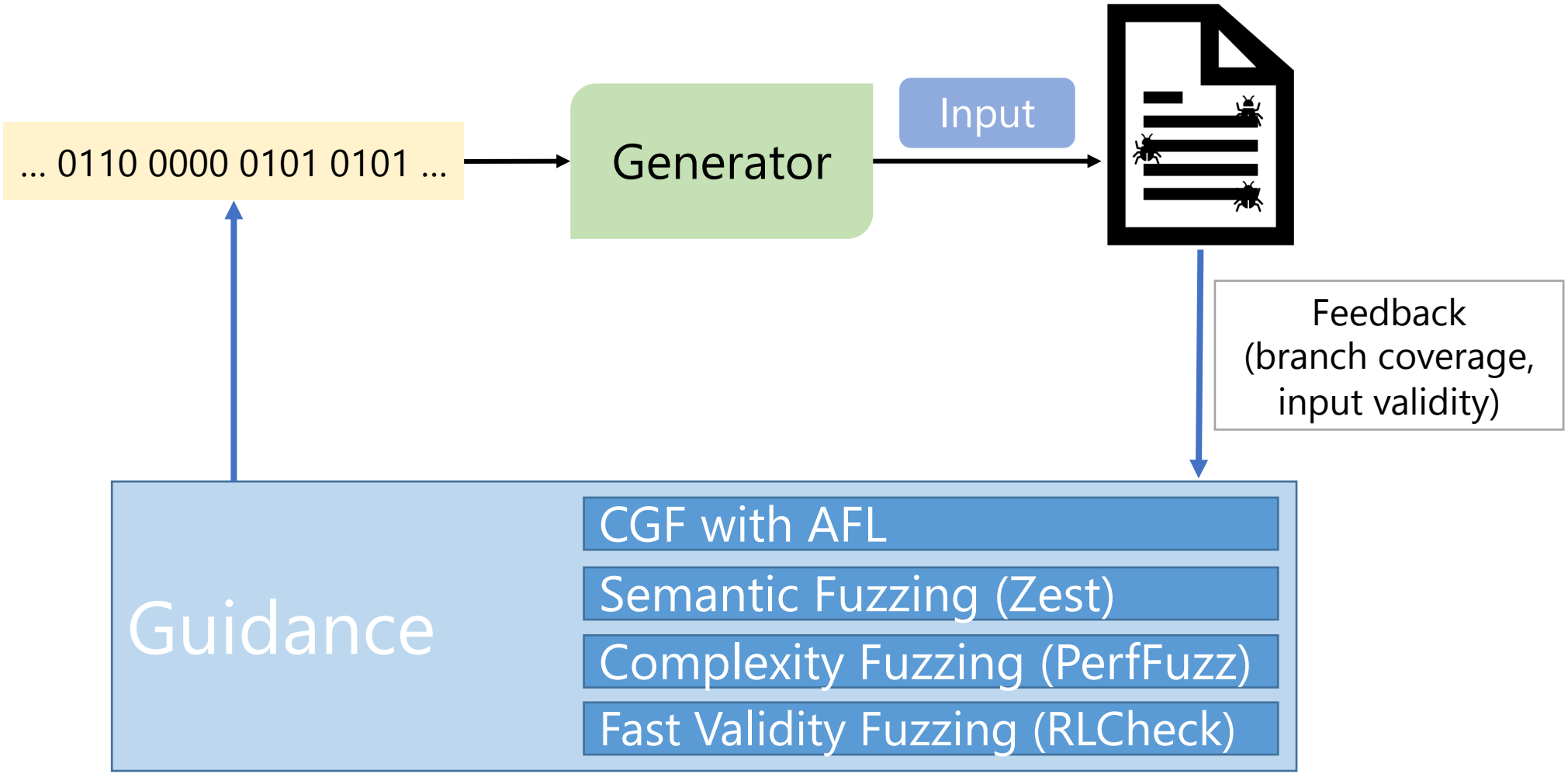


FuzzFactory
https://github.com/rohanpadhye/FuzzFactory

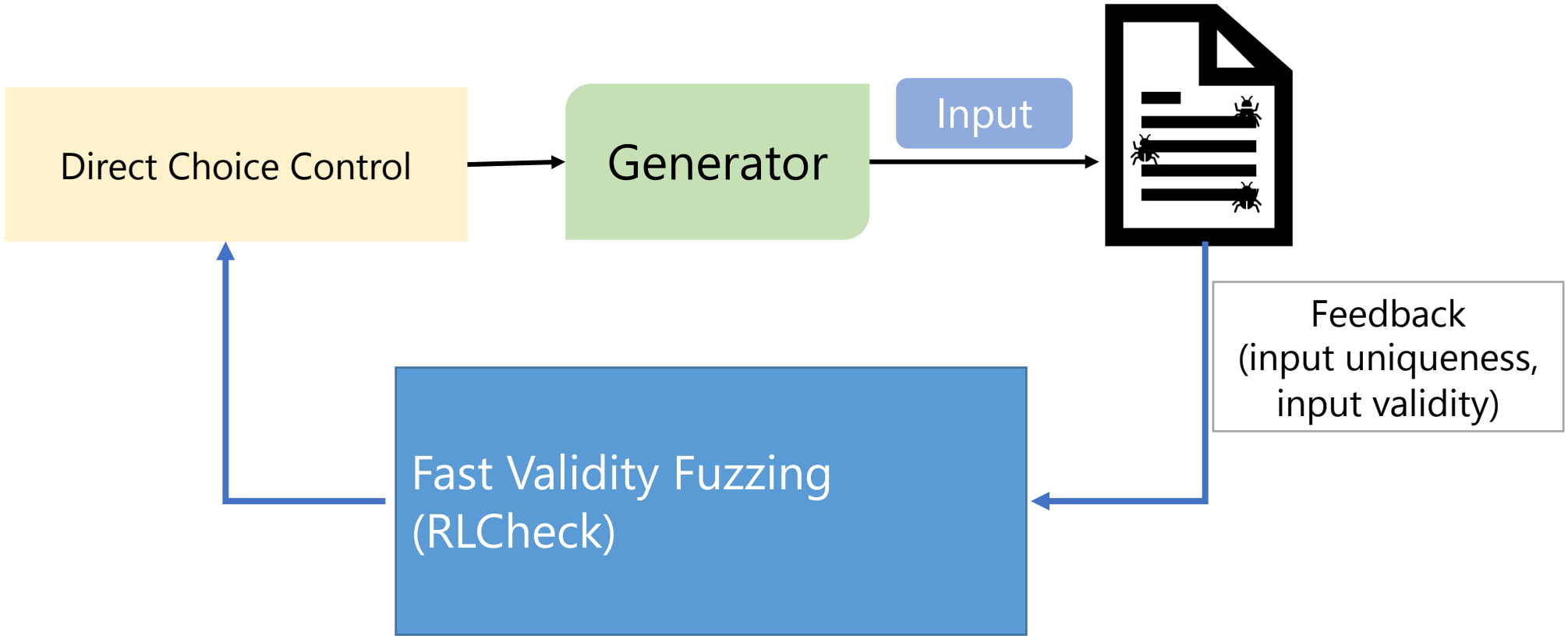
FairFuzz
https://github.com/carolemieux/afl-rb

JQF/Zest
https://github.com/rohanpadhye/jqf

RLCheck: Fast Validity Fuzzing



RLCheck: Fast Validity Fuzzing



RLCheck: Make Best Choices Given Context

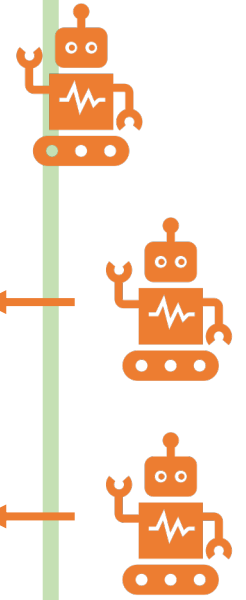
```
def genBinaryTree(depth = 0):
    value = random.choice([0, 1, …, 10] )
    node = BinaryTree(value);

    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

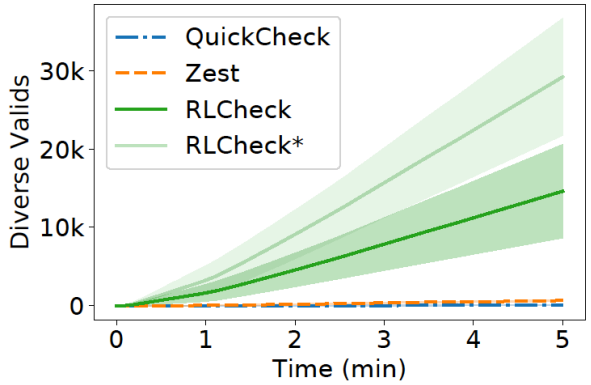
```
def genBinaryTree(depth = 0):
    value = guide.choice([0, 1, …, 10], context)
    node = BinaryTree(value);

    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

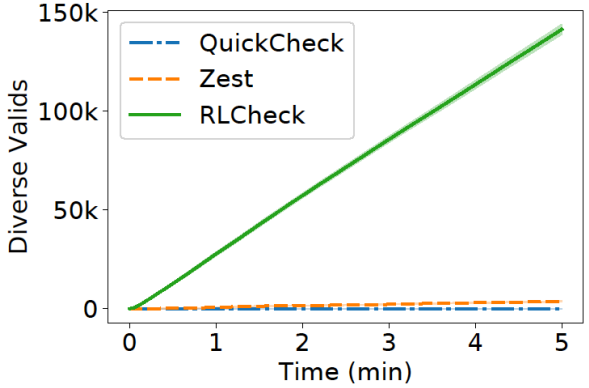
```
def genBinaryTree(depth = 0):
    value = guide.choice([0, 1, …, 10], context)
    node = BinaryTree(value);

    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
```



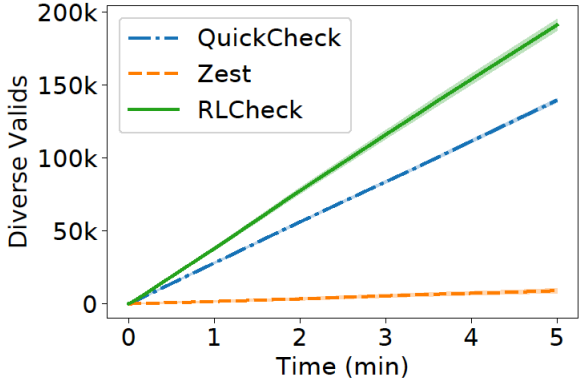
RLCheck: Many More Unique Valid Inputs



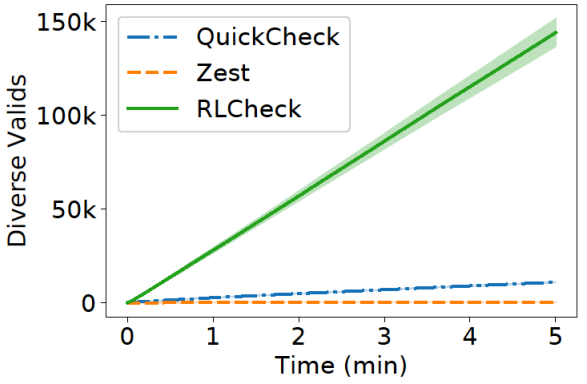
(a) Ant (*: at least 1 valid)



(b) Maven

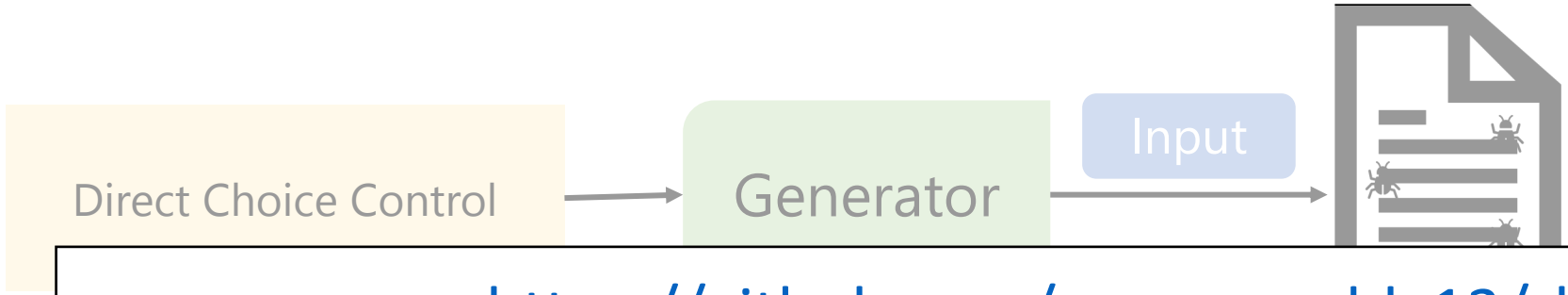


(c) Rhino



(d) Closure

RLCheck: Fast Validity Fuzzing



https://github.com/sameerreddy13/rlcheck

RLCheck

- Research prototype
- Promising direction for “smarter” blackbox fuzzing