# FUZZING 
# WITH CODE FRAGMENTS

**Christian Holler**
*Mozilla Corporation*
*choller@mozilla.com*

**Kim Herzig**
*Saarland University*
*herzig@cs.uni-saarland.de*

**Andreas Zeller**
*Saarland University*
*zeller@cs.uni-saarland.de*

# Fuzz Testing




Very common in security industry

# Typical Targets



**File formats** *(e.g. documents, pictures, ...)*



**Network protocols** *(servers and clients)*



**Specific interfaces** *(e.g. DOM)*

# Typical Targets



**File formats** *(e.g. documents, pictures, ...)*



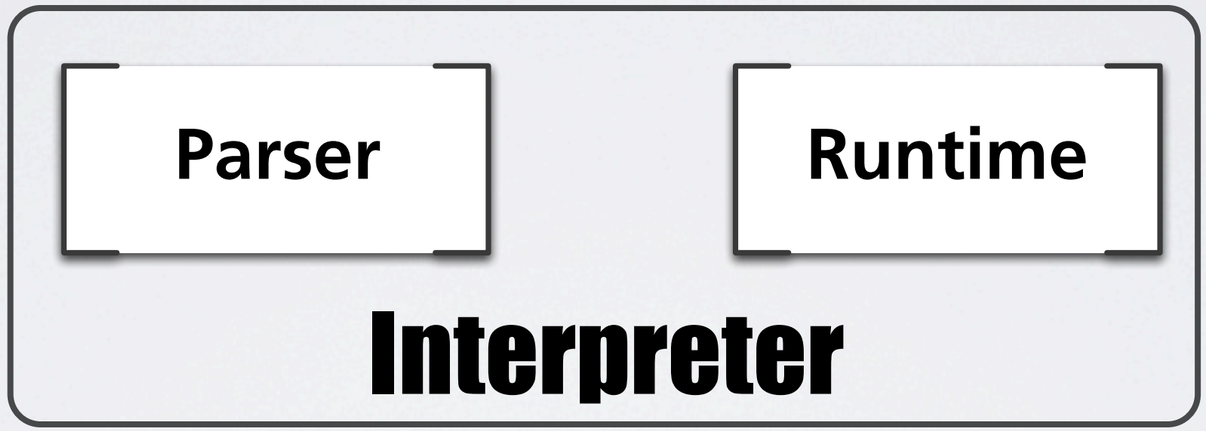
**Network protocols** *(servers and clients)*



**Specific interfaces** *(e.g. DOM)*

Lots of tools available for these types of fuzzing

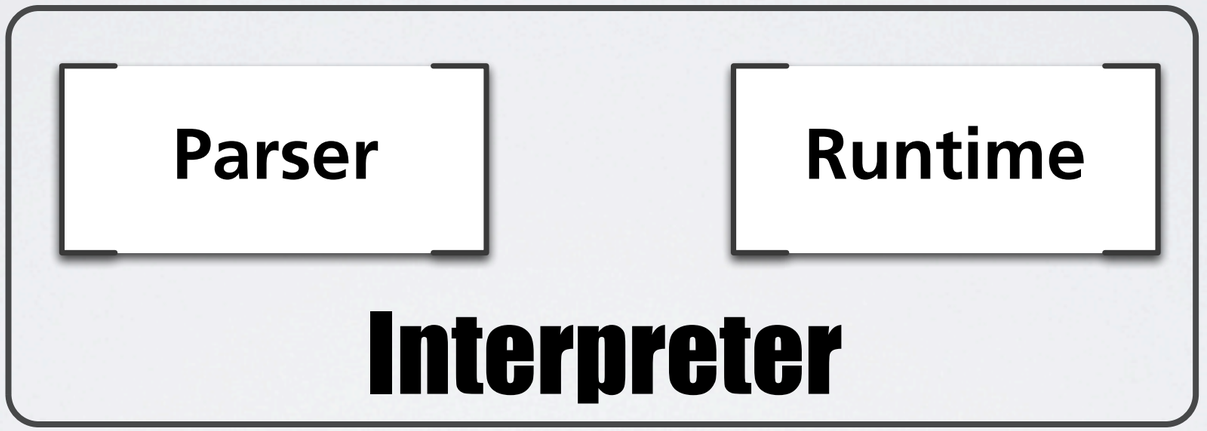
# Random Input on Interpreter



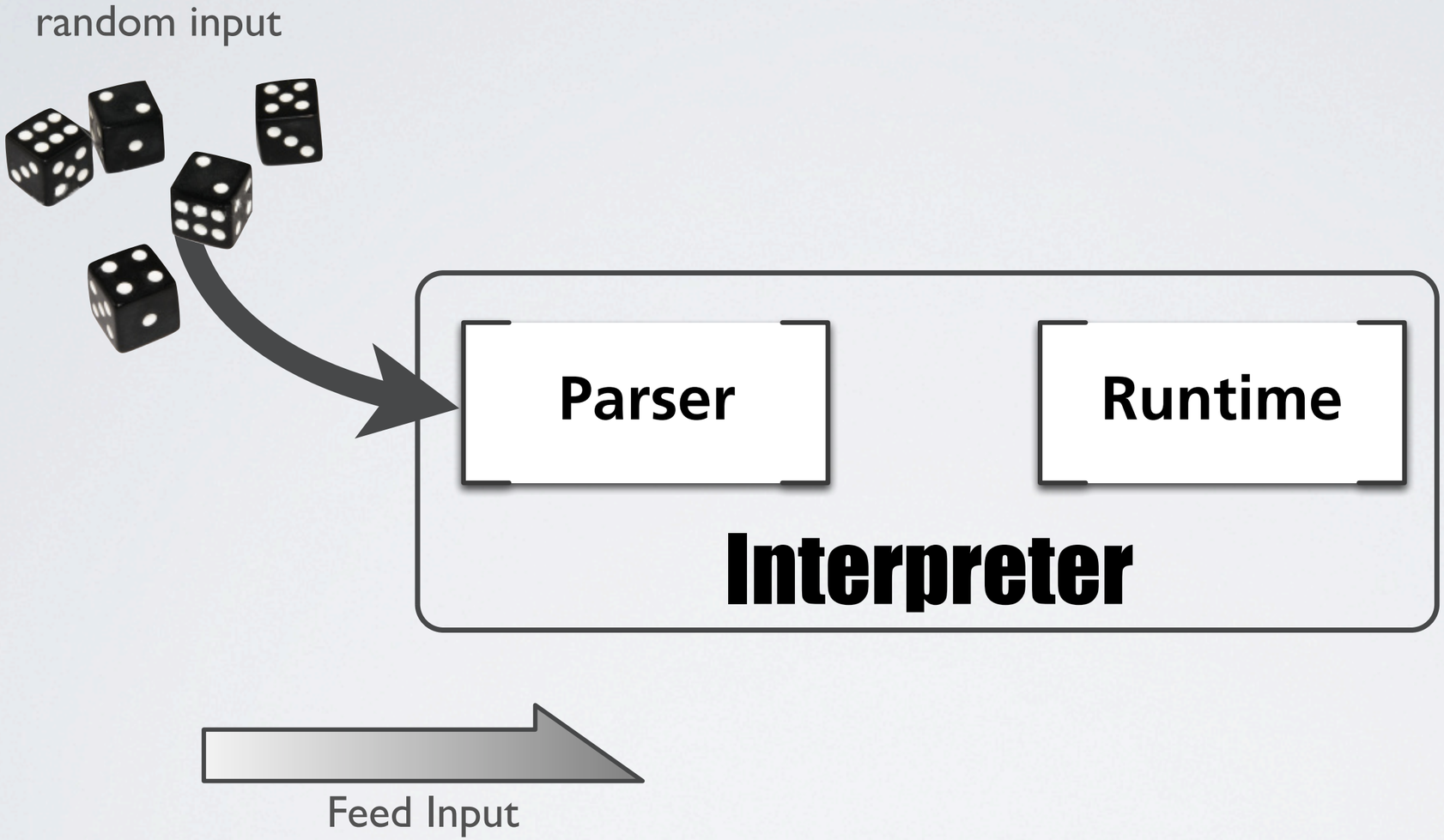
# Random Input on Interpreter

random input

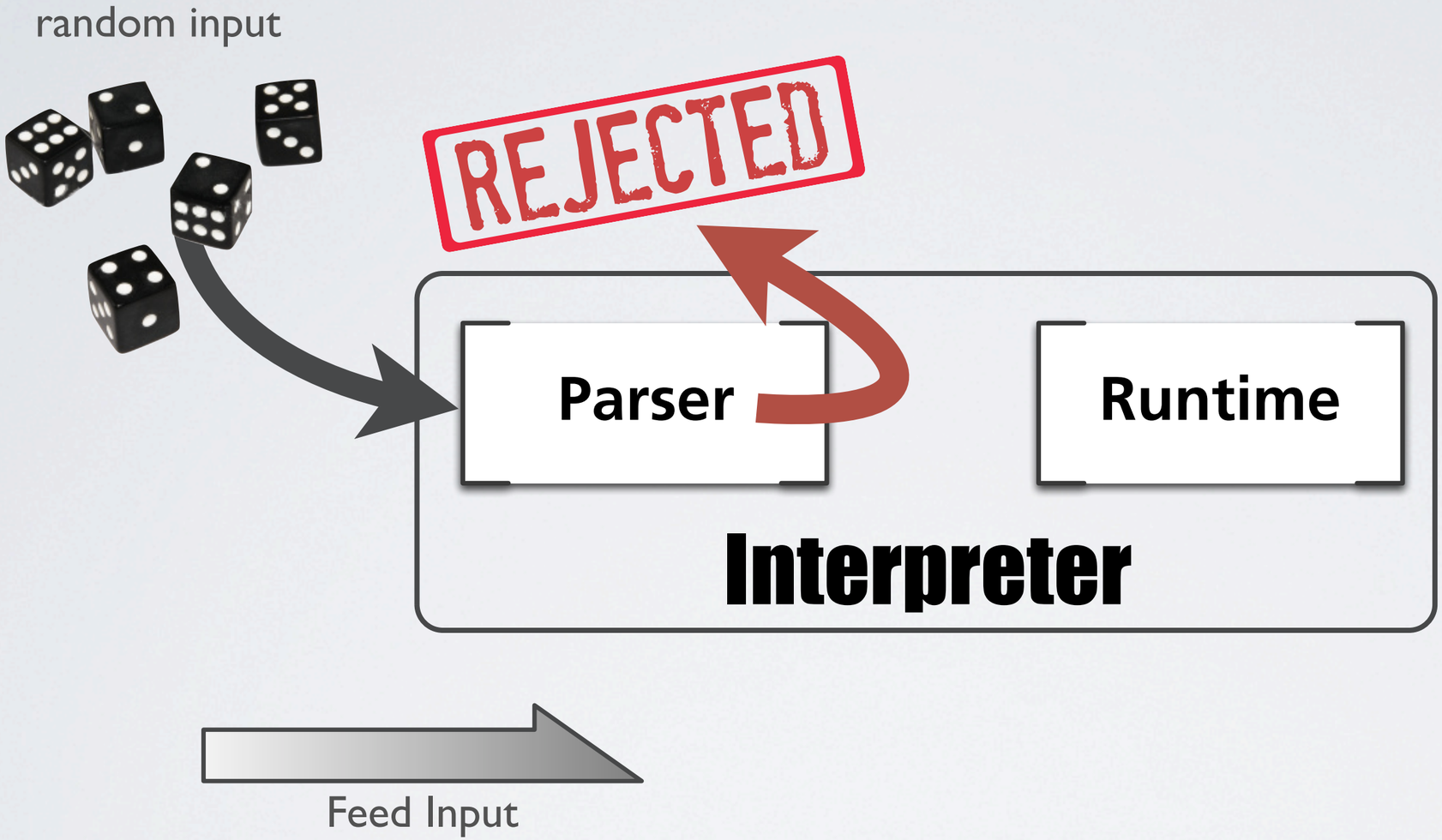




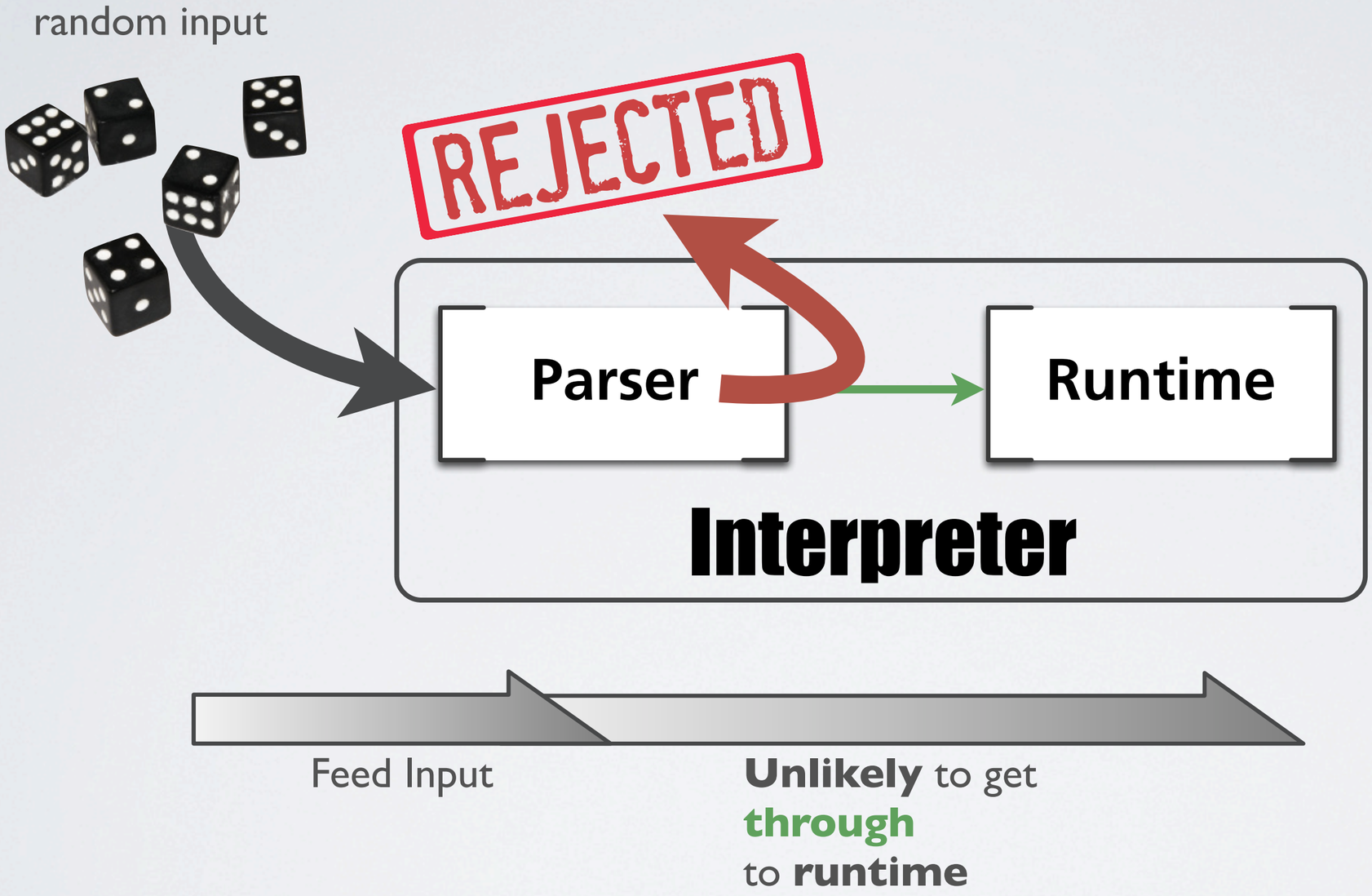
# Random Input on Interpreter



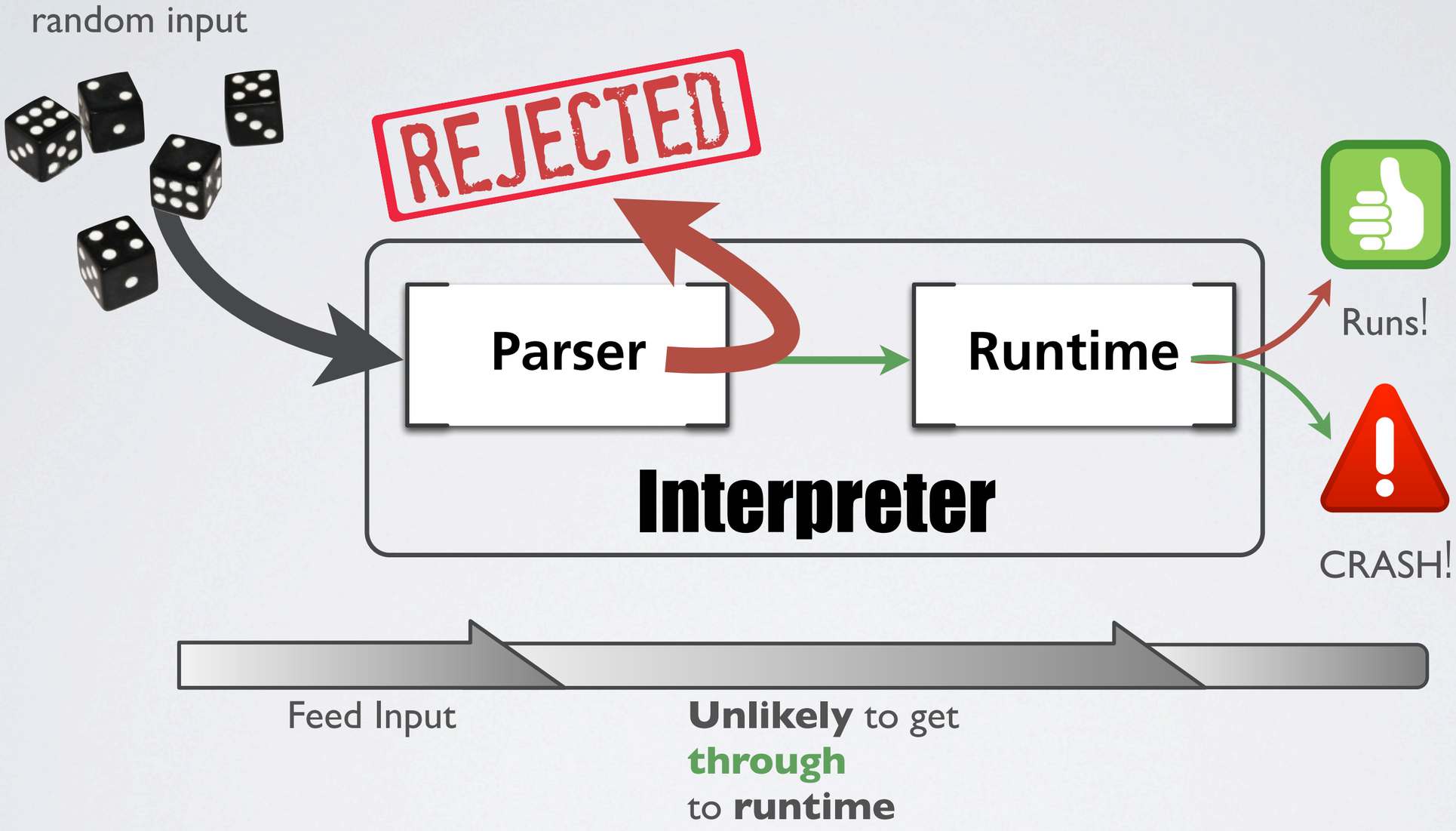
# Random Input on Interpreter



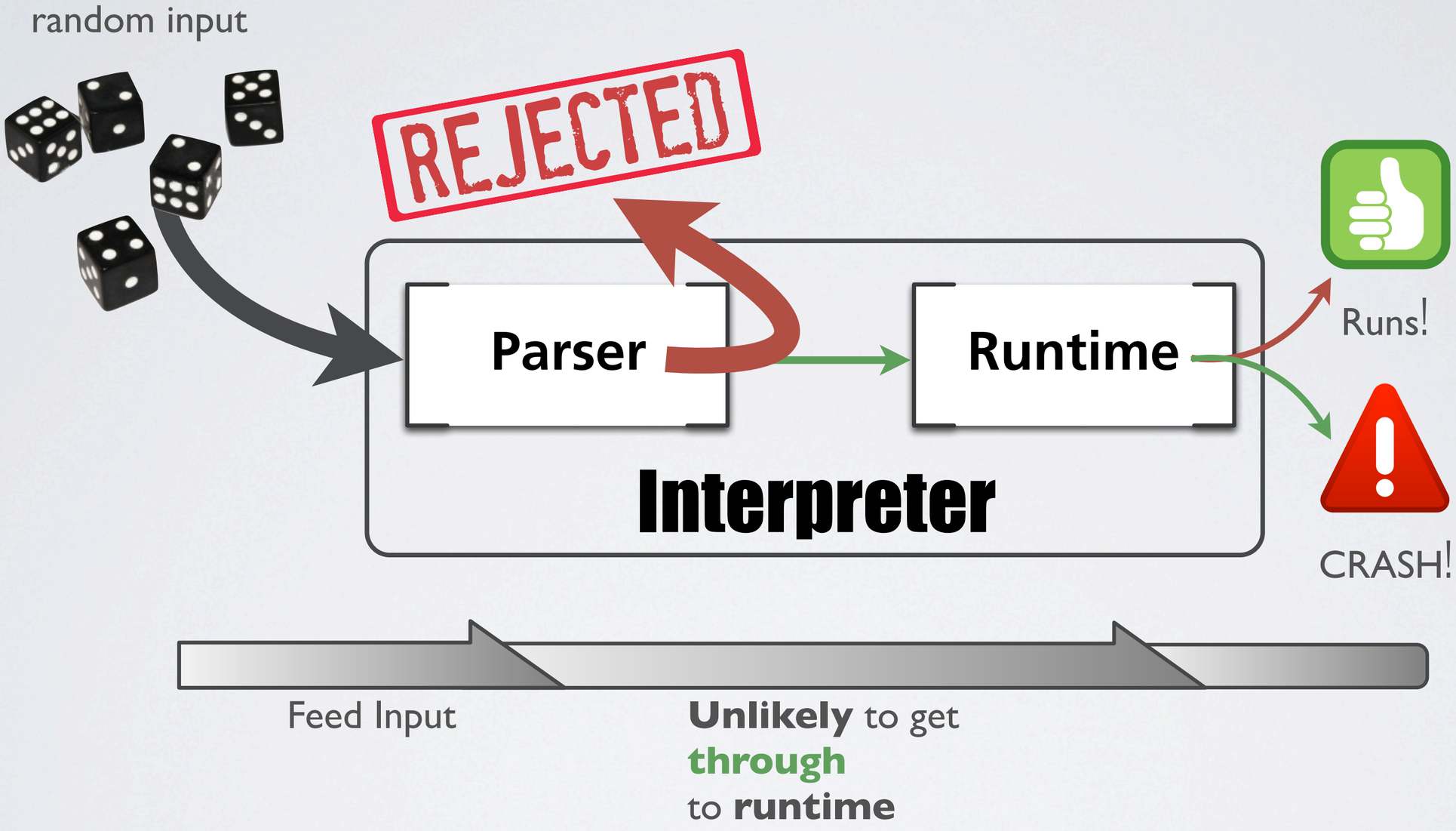
# Random Input on Interpreter



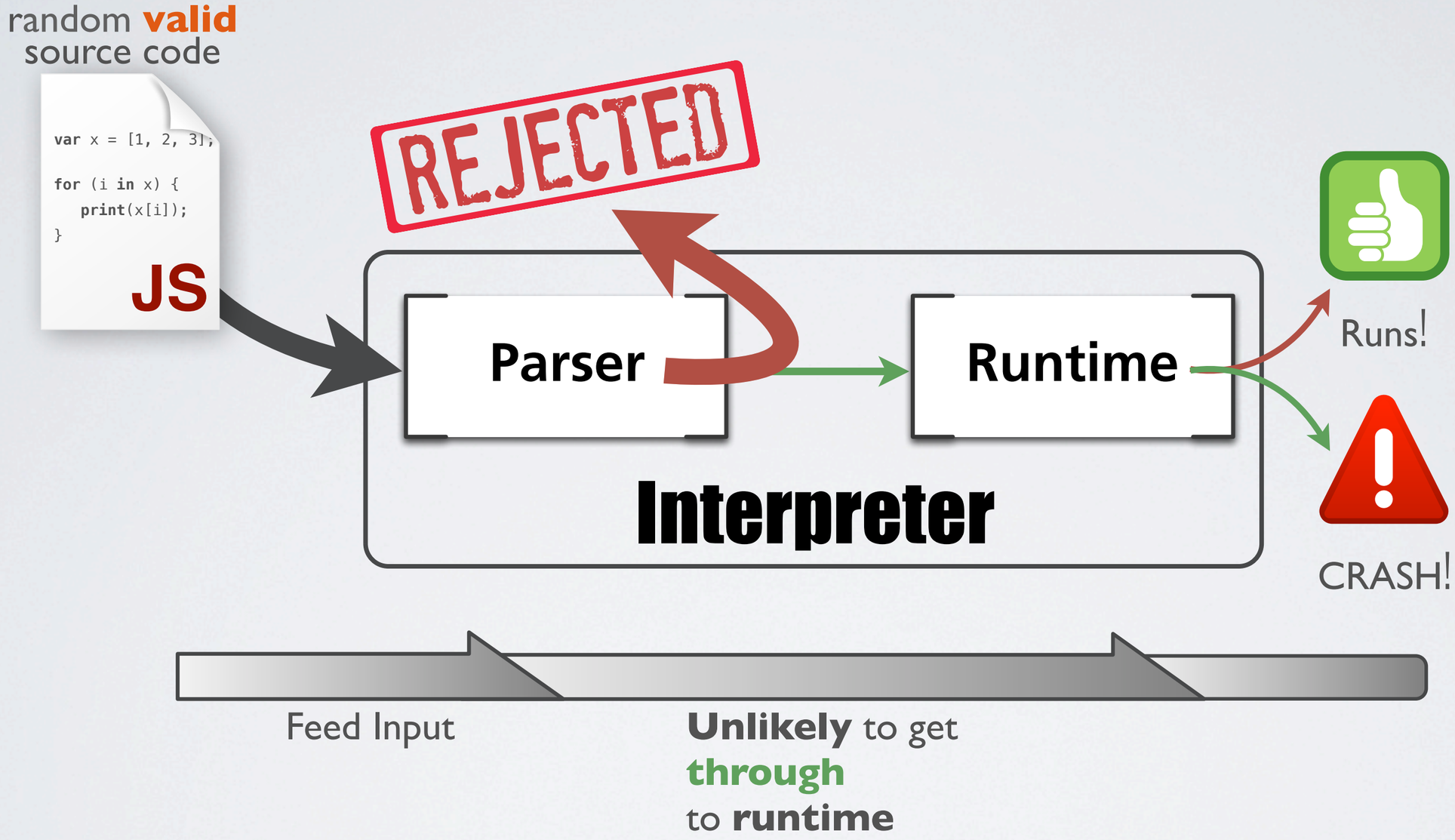
# Random Input on Interpreter



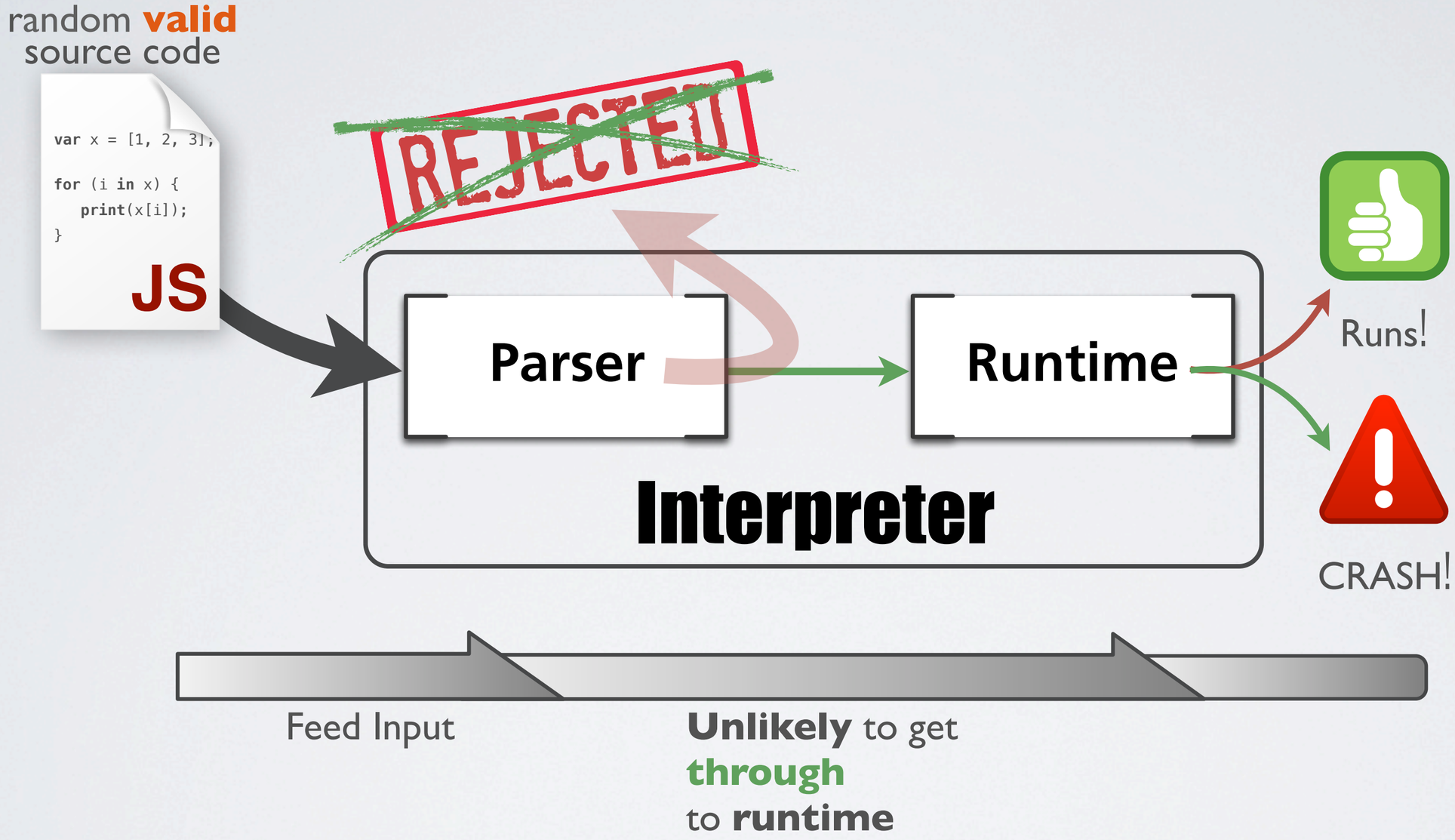
# Random Input on Interpreter



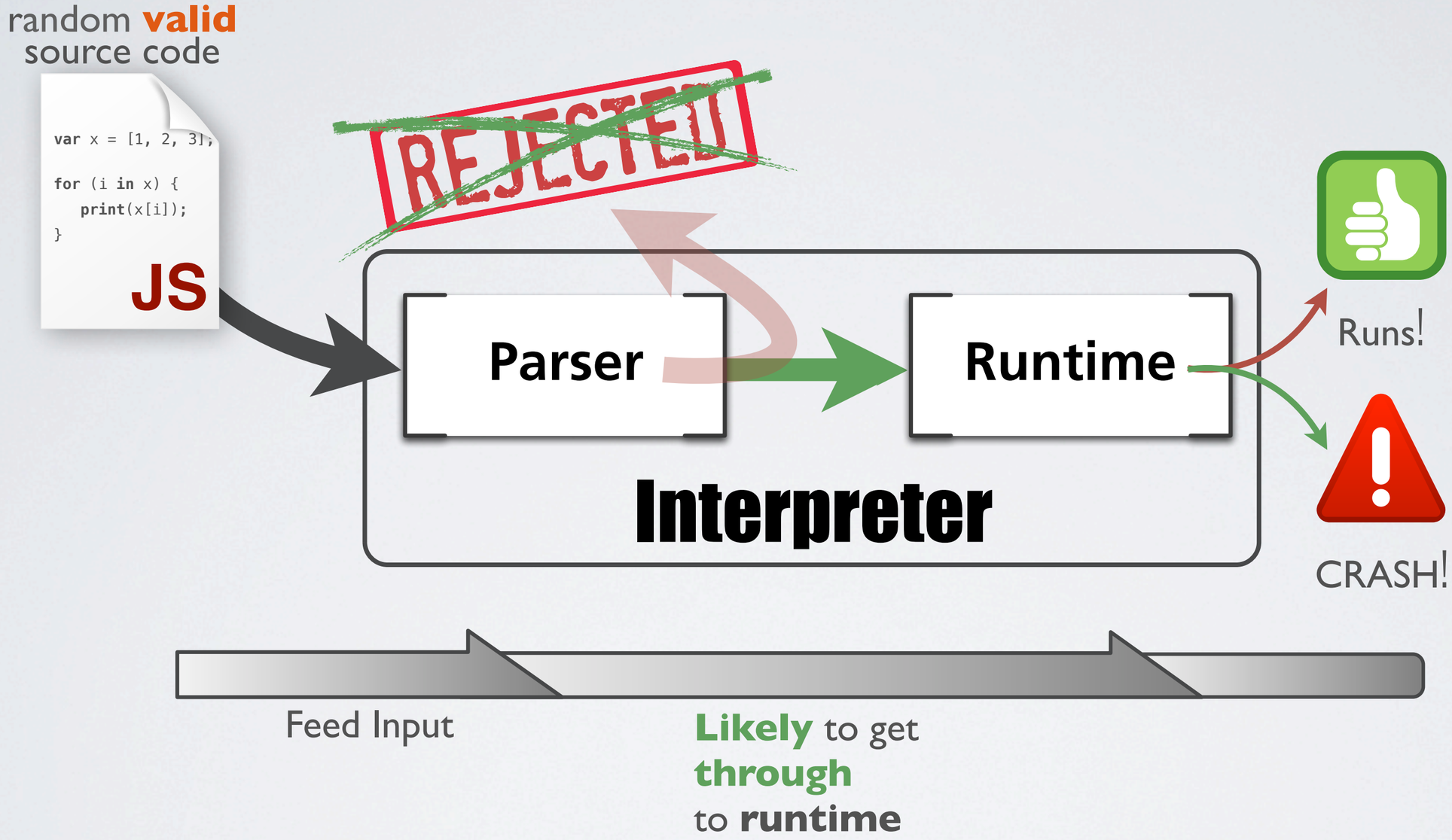
# Random Input on Interpreter



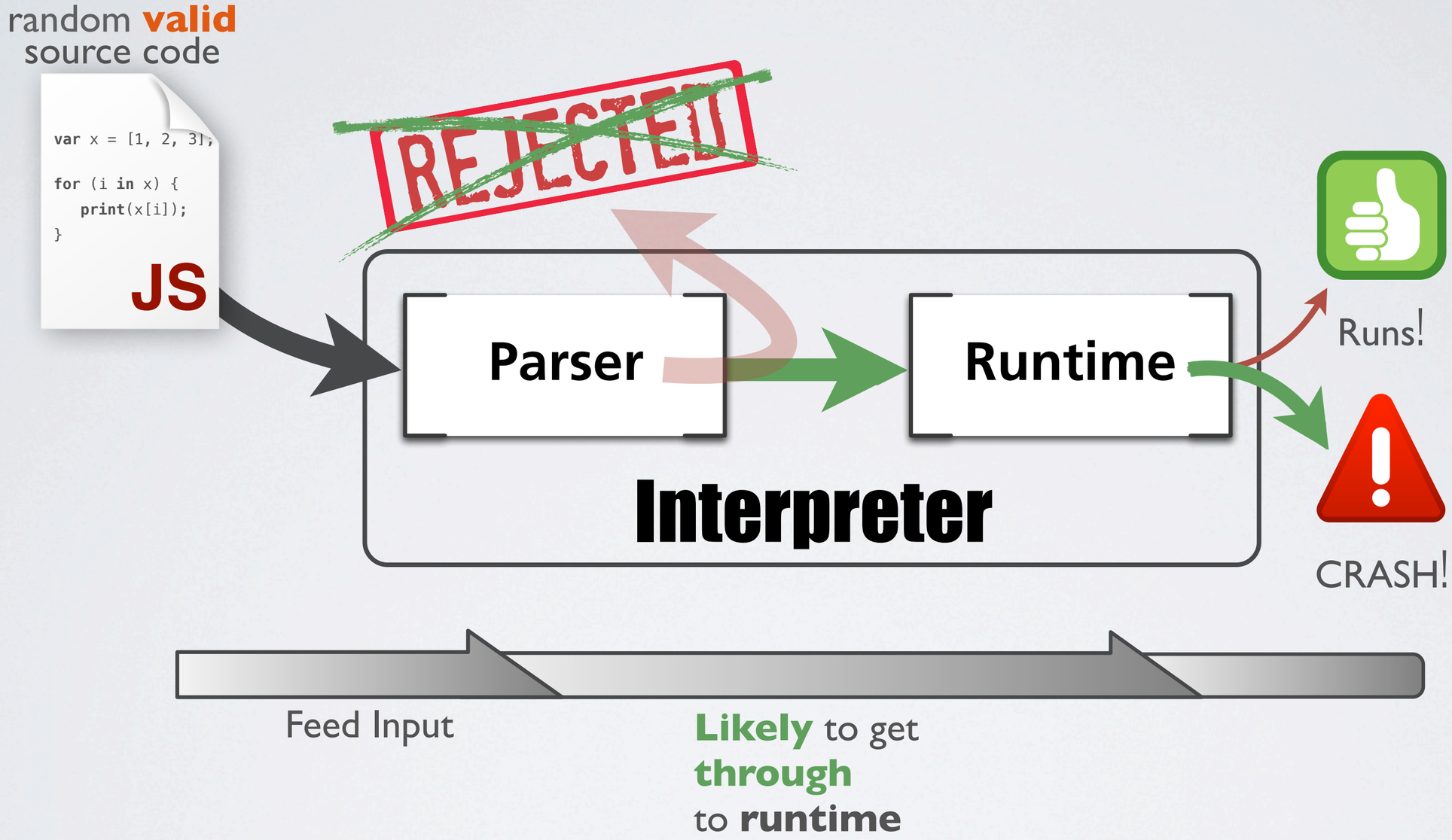
# Random Input on Interpreter



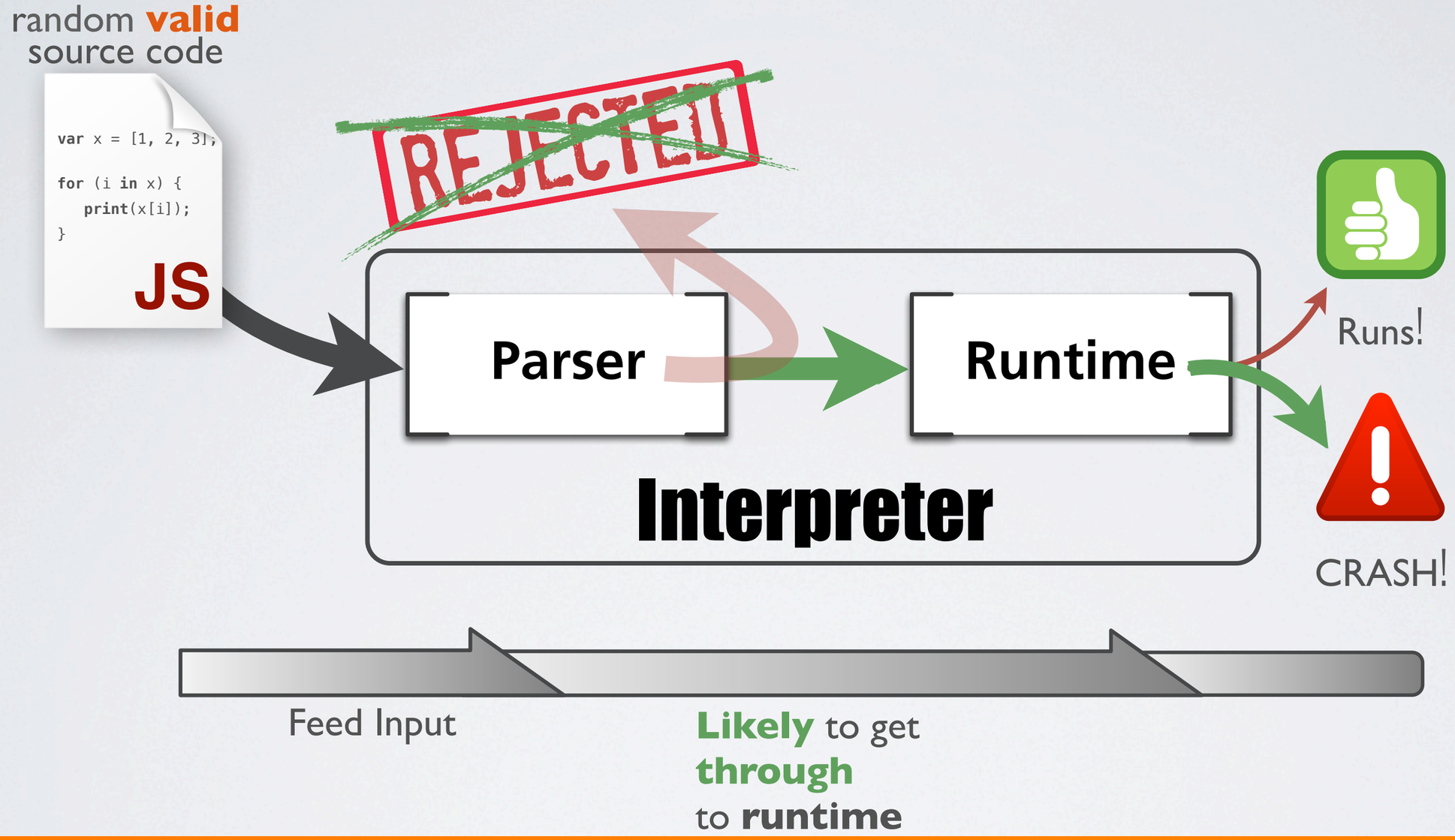
# Random Input on Interpreter



# Random Input on Interpreter



# Random Input on Interpreter




**Goal:** Find crashes during runtime.

# Random Input on Interpreter









random **valid** source code

```
var x = [1, 2, 3];

for (i in x) {
   print(x[i]);
}
```

JS

REJECTED

Parser

Runtime

Interpreter

Runs!

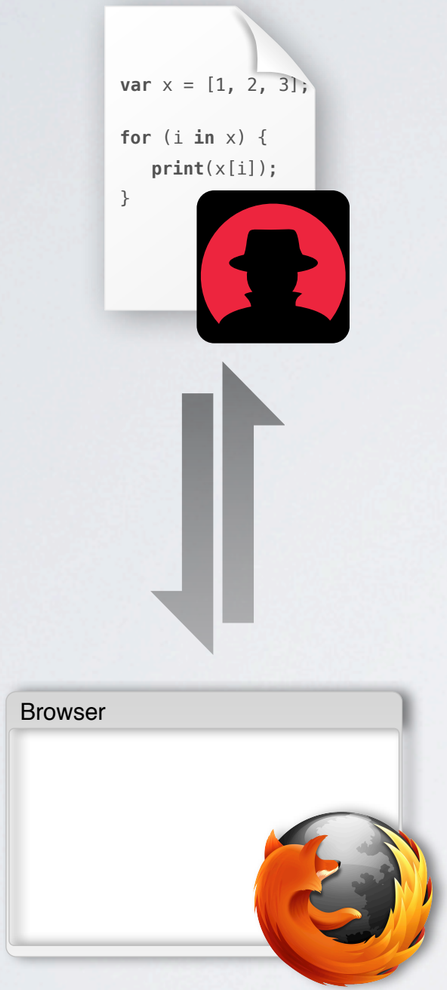
CRASH!

Feed Input

**Likely** to get **through** to **runtime**

**Goal:** Find crashes during runtime.

# Is this **security** relevant?



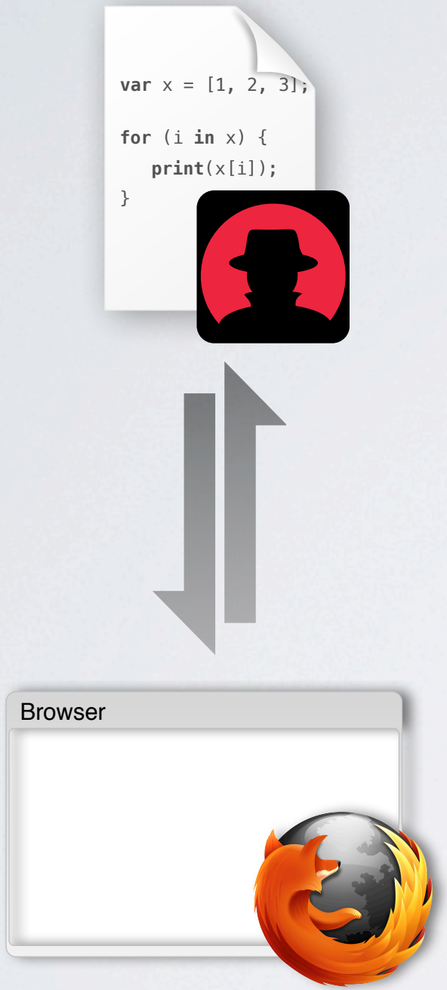

**Browsers execute arbitrary JavaScript found on websites**

*(assume attacker controls website the user visits)*

**Attacker can exploit the problem**

*(code execution, information disclosure, etc.)*

# Is this **security** relevant?




**Browsers execute arbitrary JavaScript found on websites**

*(assume attacker controls website the user visits)*

**Attacker can exploit the problem**

*(code execution, information disclosure, etc.)*

# Is this **security** relevant?




**Browsers execute arbitrary JavaScript found on websites**

*(assume attacker controls website the user visits)*

**Attacker can exploit the problem**

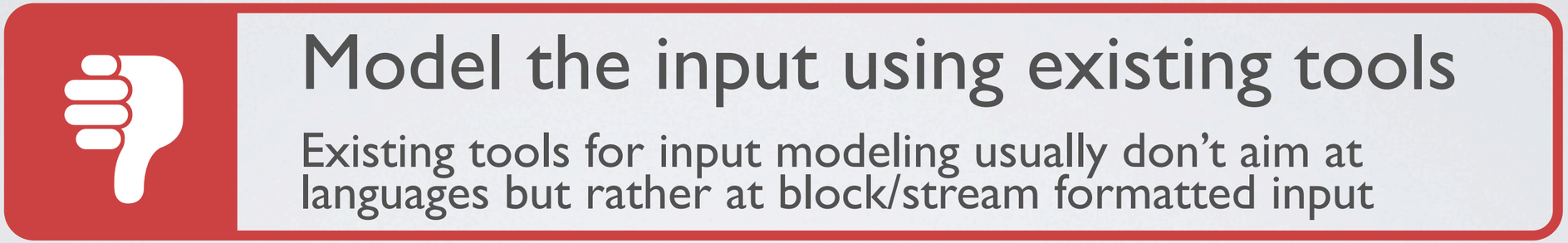
*(code execution, information disclosure, etc.)*

JavaScript as the primary language for evaluation

# Possible Approaches

Model the input using existing tools

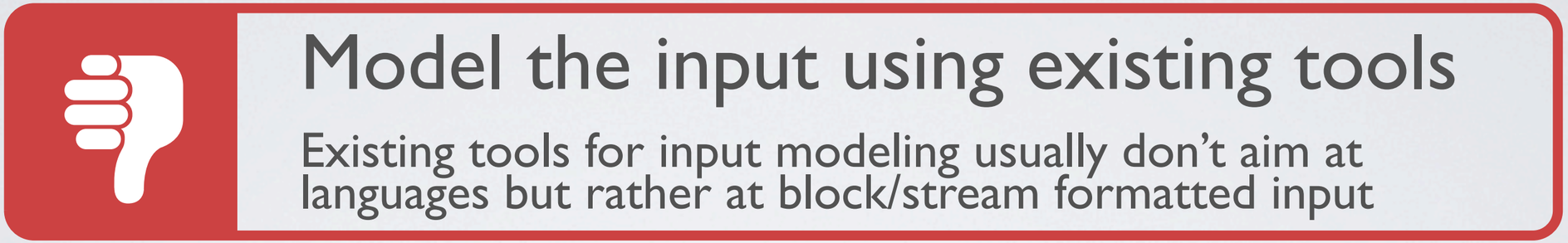
# Possible Approaches



## Model the input using existing tools

Existing tools for input modeling usually don't aim at languages but rather at block/stream formatted input

# Possible Approaches

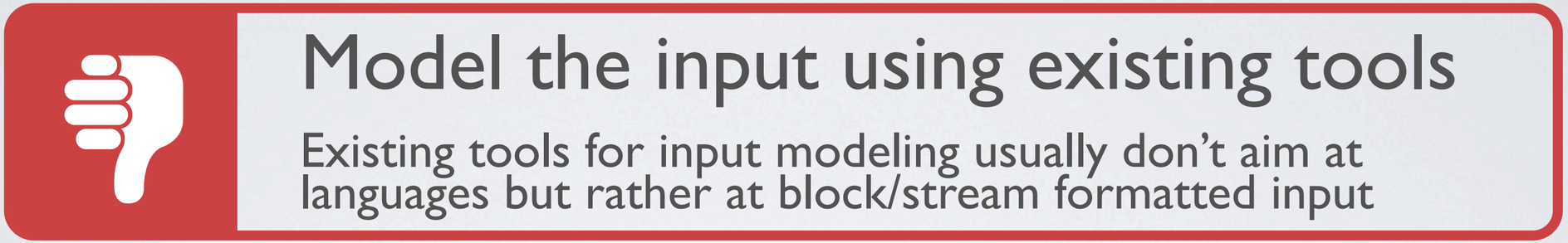


## Model the input using existing tools

Existing tools for input modeling usually don't aim at languages but rather at block/stream formatted input

## Write a custom fuzzer for language

# Possible Approaches



## Model the input using existing tools

Existing tools for input modeling usually don't aim at languages but rather at block/stream formatted input



## Write a custom fuzzer for language

Not generic, high maintenance

# Possible Approaches



## Model the input using existing tools

Existing tools for input modeling usually don't aim at languages but rather at block/stream formatted input



## Write a custom fuzzer for language

Not generic, high maintenance



## Generic fuzzing framework for languages with low maintenance

# Our Tool



# Our Tool

- as independent as possible from the target interpreter language



# Our Tool

- as independent as possible from the target interpreter language
- only makes general assumptions about languages (e.g. identifiers and builtin functions)



# Our Tool

- as independent as possible from the target interpreter language
- only makes general assumptions about languages (e.g. identifiers and builtin functions)
- does not rely on source code of target program

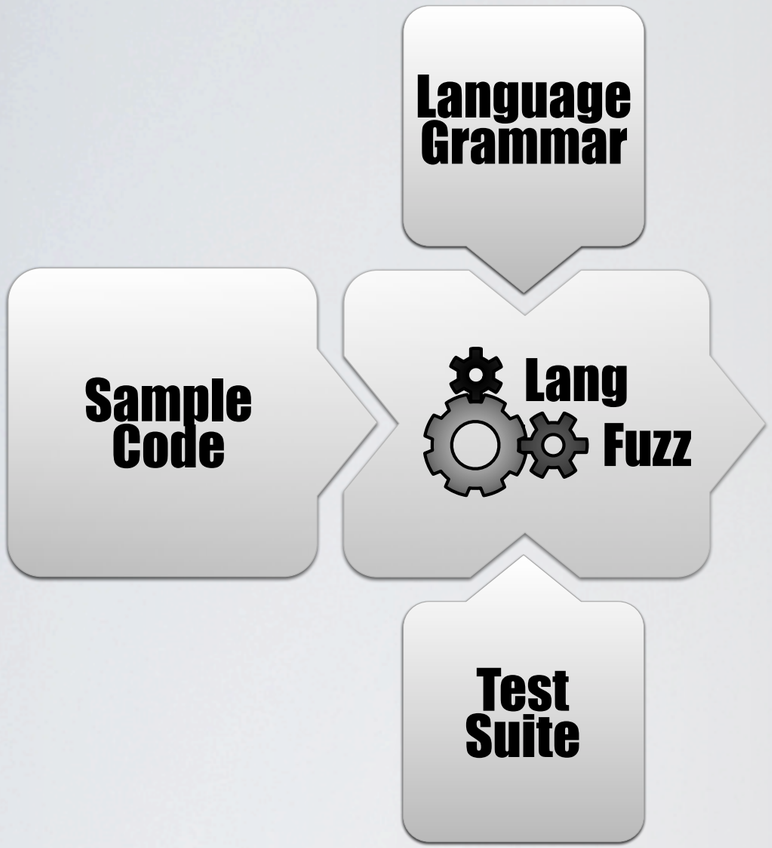


# Our Tool

- as independent as possible from the target interpreter language
- only makes general assumptions about languages (e.g. identifiers and builtin functions)
- does not rely on source code of target program
- finds defects in real-world programs



# Workflow




## Phase 1

Learning code fragments from sample code

# Workflow



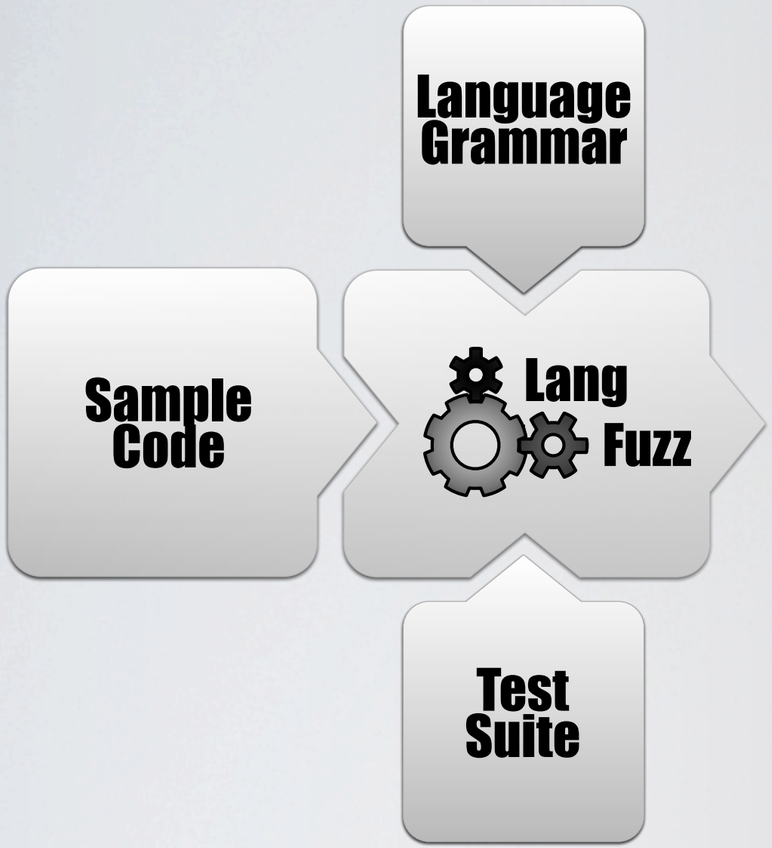

## Phase 1

Learning code fragments from sample code

## Phase 2

LangFuzz generates (mutated) test cases

# Workflow



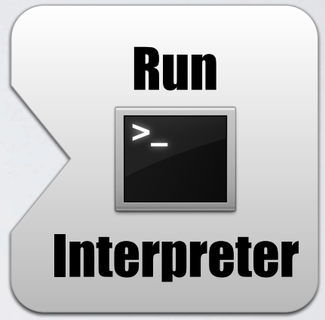

## Phase 1

Learning code fragments from sample code




## Phase 2

LangFuzz generates (mutated) test cases




## Phase 3

Feed test case into interpreter, check for crashes and assertions

# Learning Code Fragments






# Learning Code Fragments






# Code Mutation



# Code Mutation



# Code Mutation



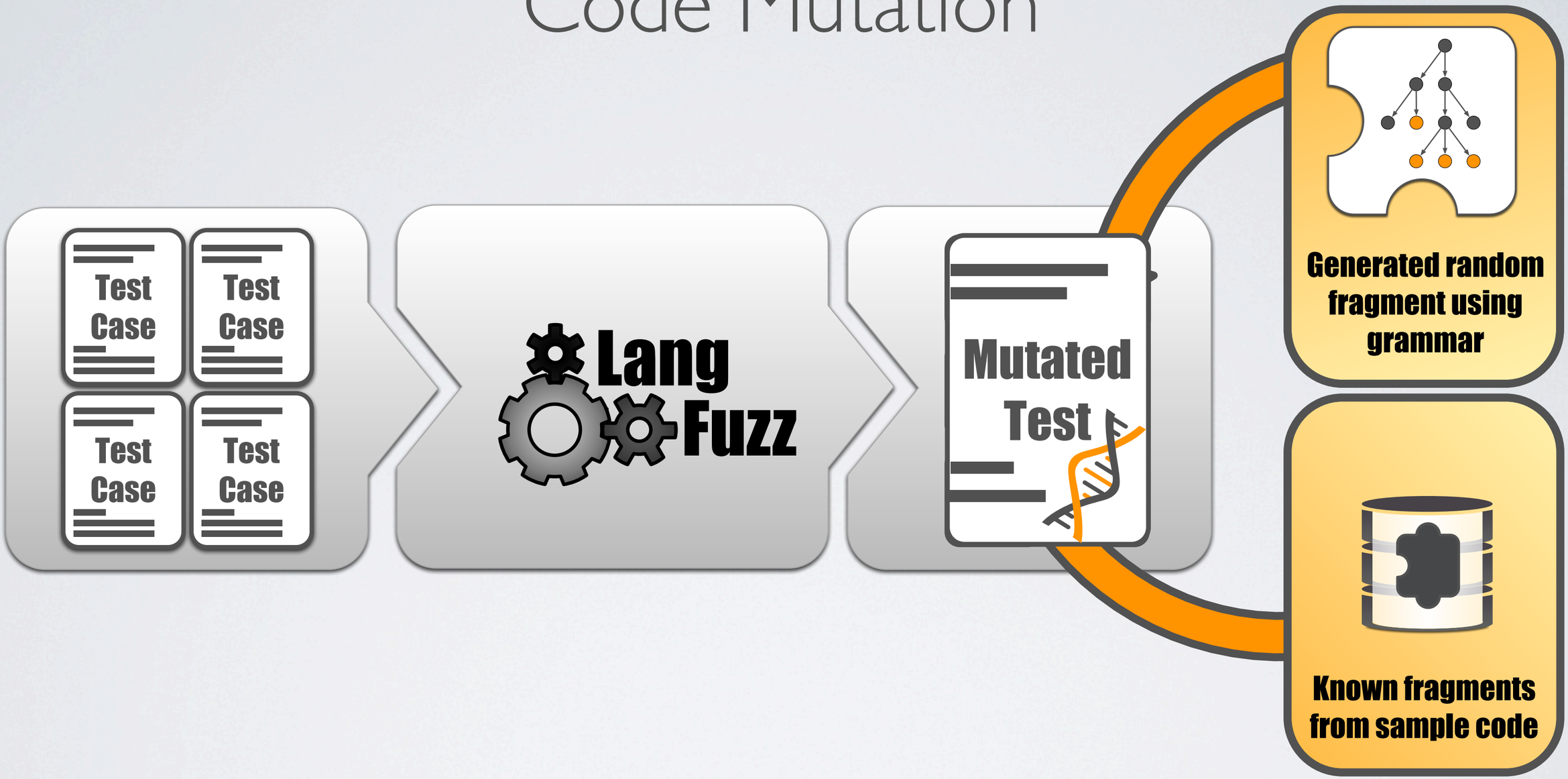
# Code Mutation



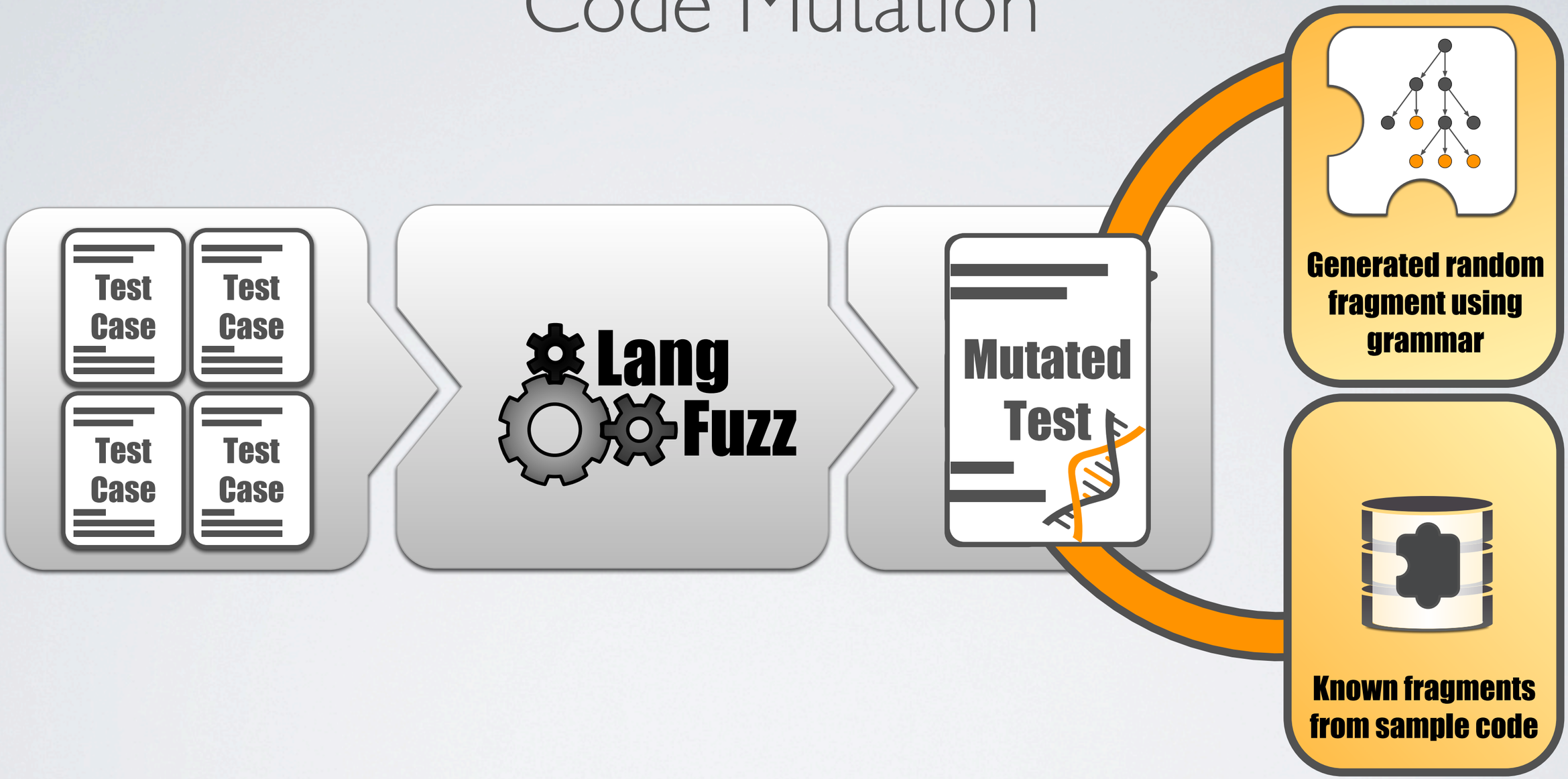
# Code Mutation



# Code Mutation



# Code Mutation




Locate **incomplete fixes** and **new defects**.

# Evaluation Strategies

Comparison with
**State of the Art**




**Internal Comparison**

Field Tests with
**Mozilla** and **Google**

Proof-of-Concept with
**Different Language**



Indistinguishable from Jesse

Jesse Ruderman on Firefox, security, and more

« Black Hat

Fuzzing for correctness »

### Introducing jsfunfuzz

I wrote a fuzzer called jsfunfuzz for testing the JavaScript engine in Firefox. Window, Shaver, and I announced it at Black Hat earlier today, as part of Mozilla's presentation, "Building and Breaking the Browser".

It tests the JavaScript language engine itself, not the DOM. (That means that it works with language features such as functions, objects, operators, garbage collection rather than DOM objects accessed through "window" or "document".)

It has found about 280 bugs in Firefox's JavaScript engine, over two-thirds of which have already been fixed (go Brendan!). About two dozen were memory safety bugs that we believe were likely to be exploitable to run arbitrary code.

In the presentation, I speculated as why it has been able to find so many bugs:

- **It knows the rules** of the JavaScript language, allowing it to get decent coverage of combinations of language features.
- **It breaks the rules**, allowing it to find errors in syntax error handling such as bug 350415 and more generally helping the fuzzer avoid having "blind spots".
- **It isn't afraid to nest** JavaScript constructs in fairly complicated ways, like when it found bug 353079.
- **It allows state to accumulate** by creating and running functions in a loop. (See bug 361346 for an example of a bug that would be hard to find otherwise.)
- **It tests for correctness**, not just crashes and assertions. (Since I didn't talk about this aspect much during the security-focused Black Hat presentation, I've made it a separate blog post.)

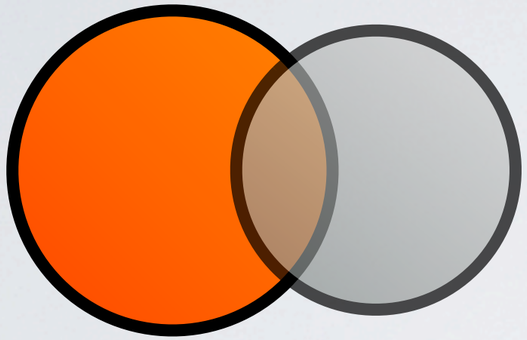
If you want to test it out, grab jsfunfuzz.js and multi_timed_run.py from jsfunfuzz and build the standalone JavaScript Shell. (It can be tested inside a web browser, but it works better in a standalone shell where output is simple and restarting isn't painful.)

# State of the Art

## Jesse Ruderman (Mozilla) released **jsfunfuzz** in 2007

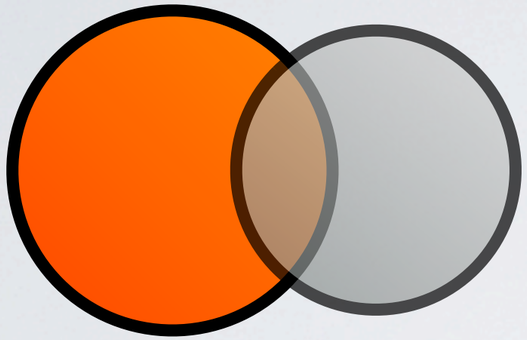
- found over 1000 bugs in Mozilla's engine
- highly specialized on JavaScript features
- requires changes for different languages or new language features
- produces only what it was programmed for

# Comparison with State of the Art

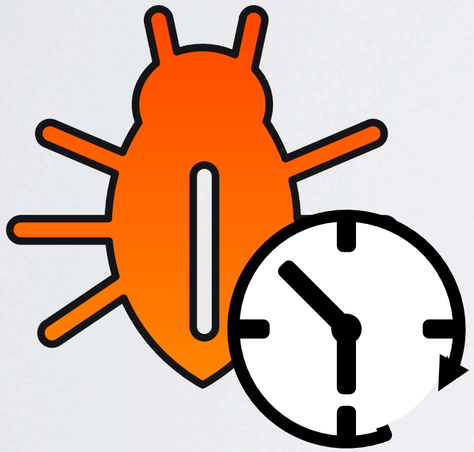


Overlap: What amount of defects are found by both tools?

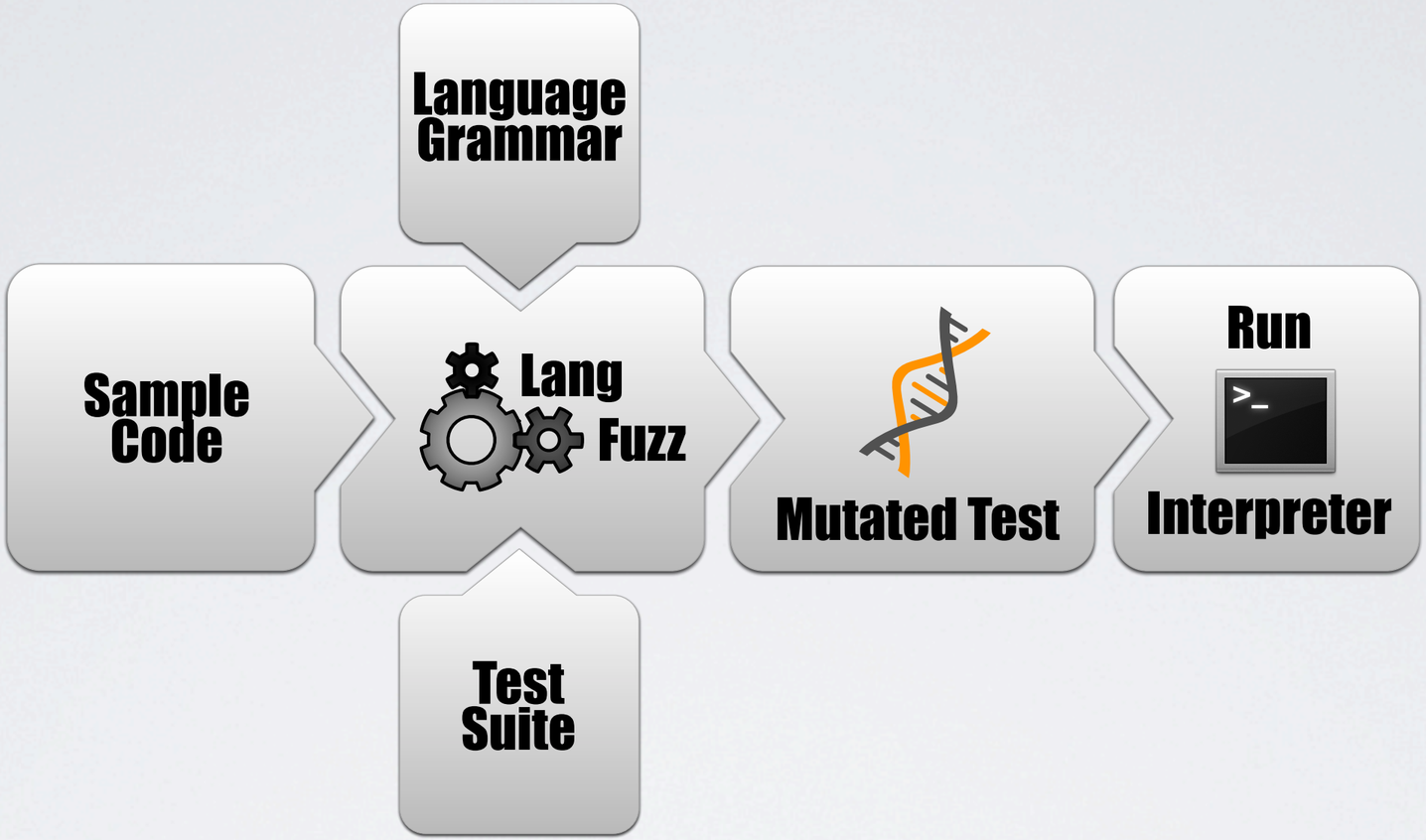
# Comparison with State of the Art

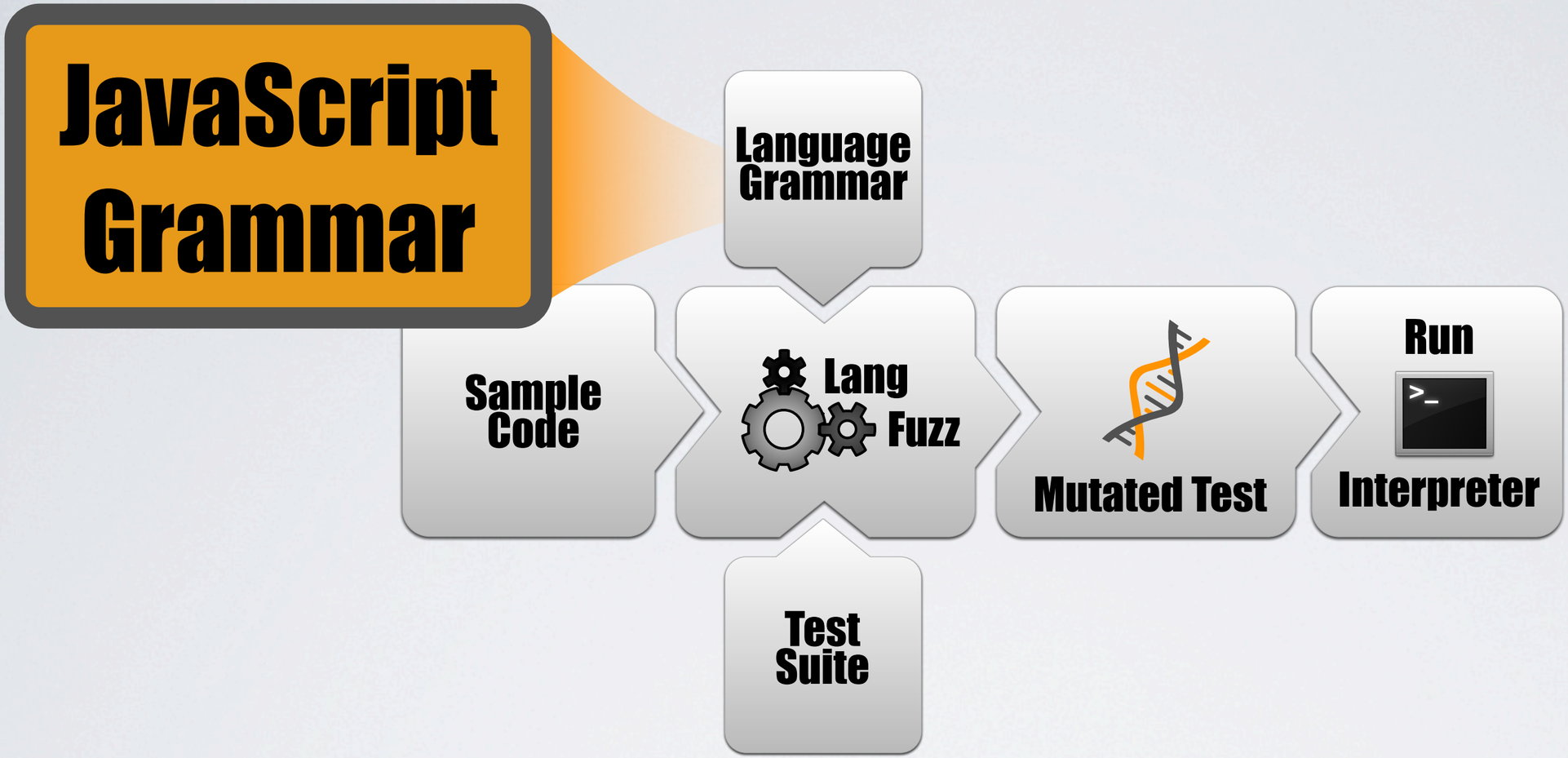


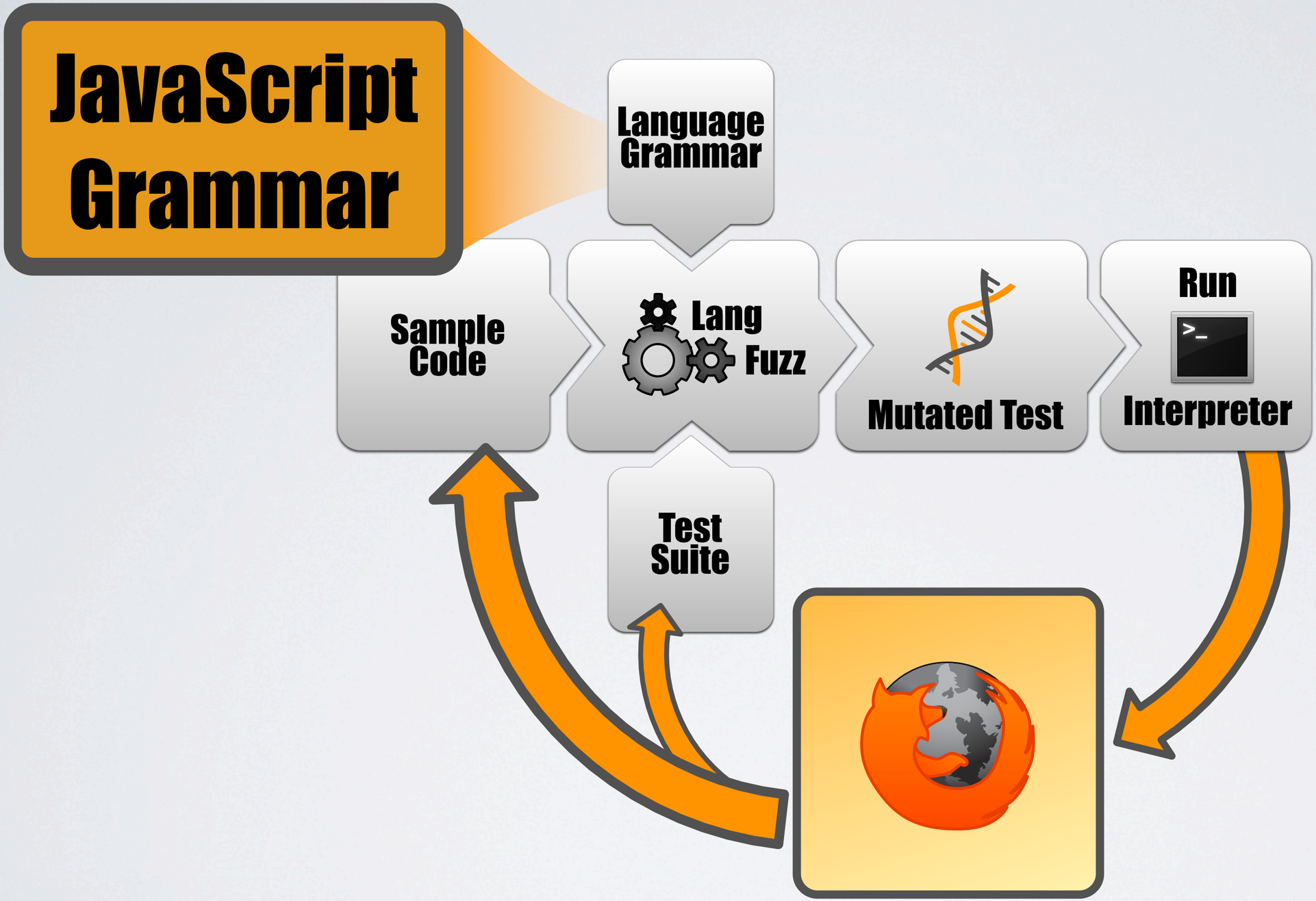
Overlap: What amount of defects are found by both tools?



Effectiveness: How many defects per time unit do the tools find?


Language
Grammar
Sample
Code
Lang
Fuzz
Mutated Test
Run
Interpreter
Test
Suite

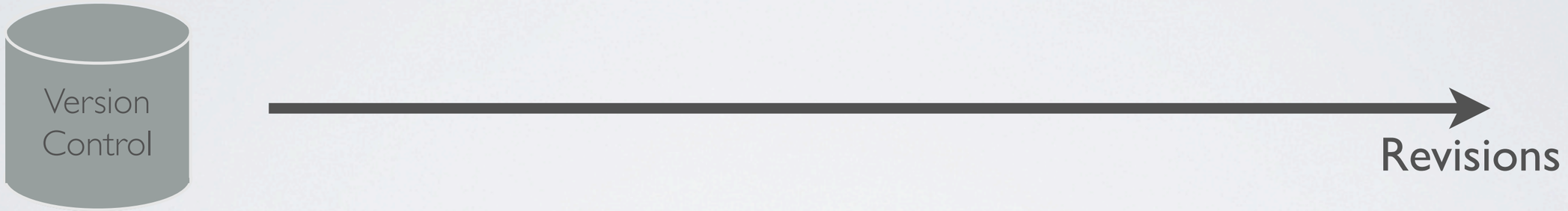

JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Test Suite
Mutated Test
Run
Interpreter


JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Run
Interpreter
Test Suite

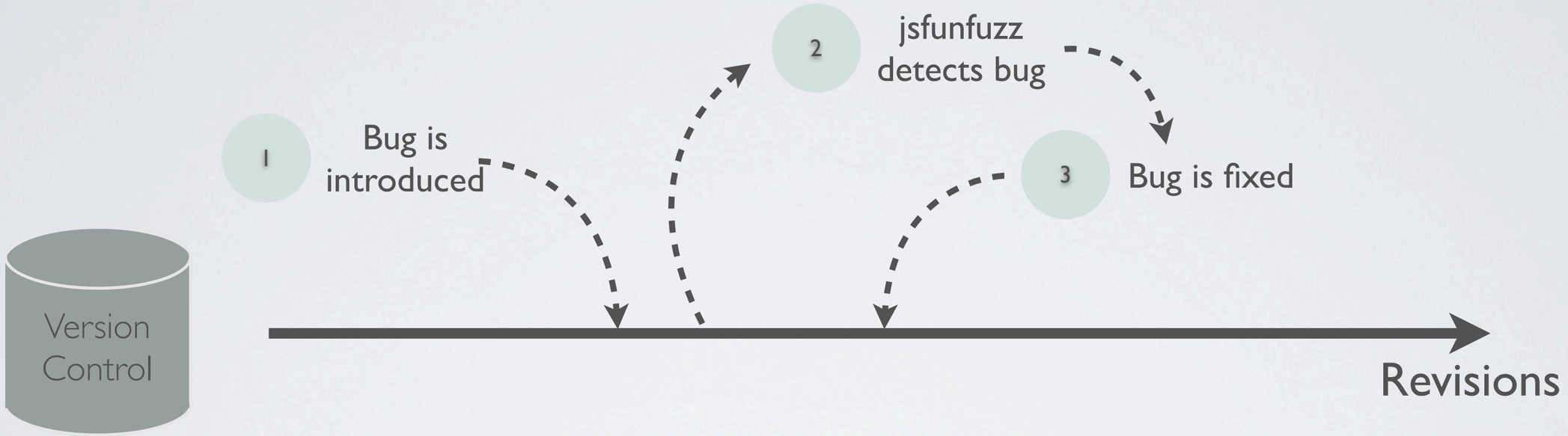
# Comparison with State of the Art



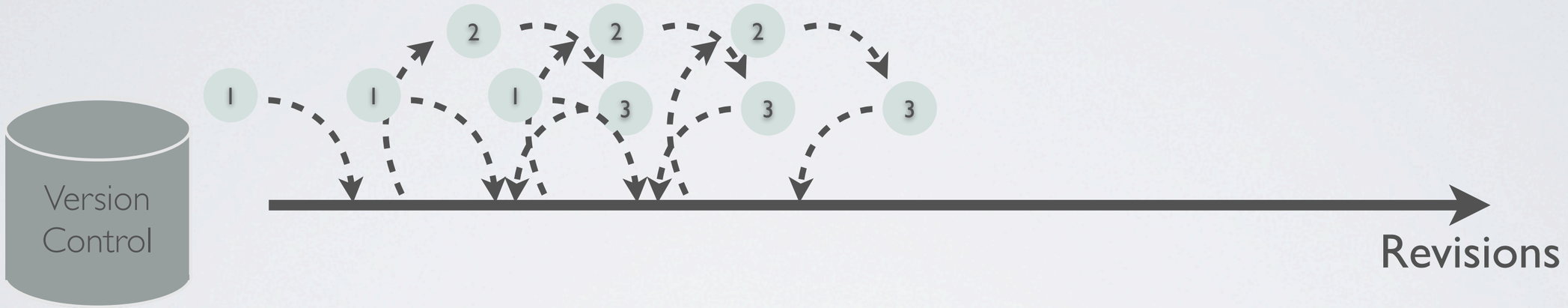
# Comparison with State of the Art



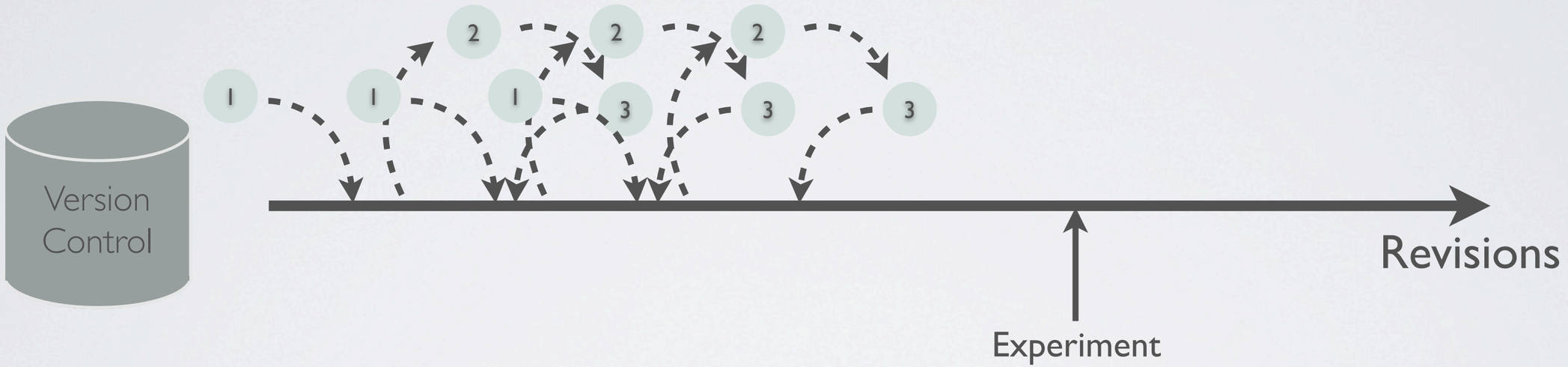
# Comparison with State of the Art



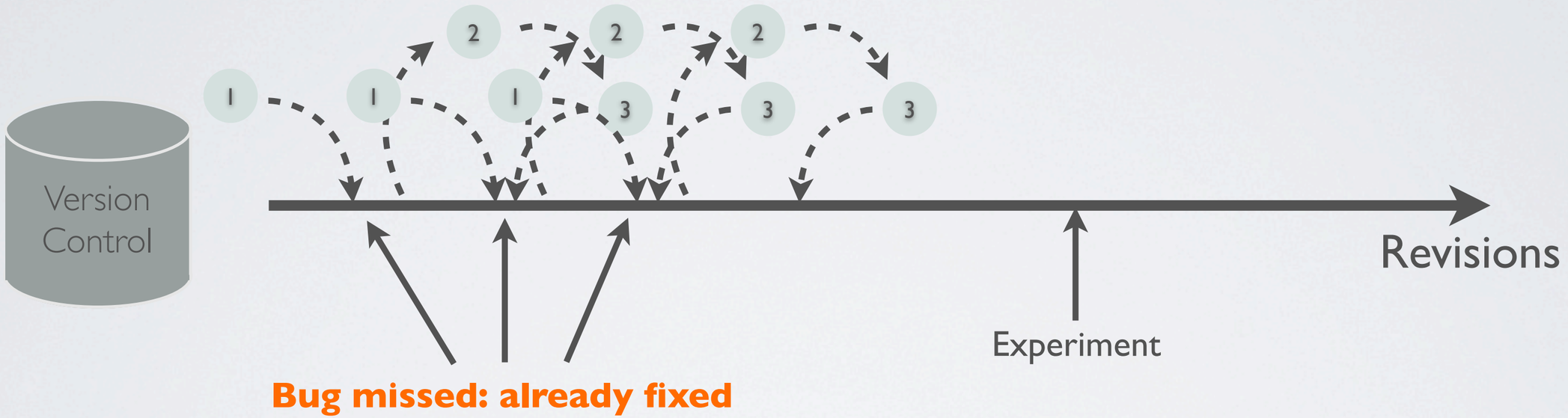
# Comparison with State of the Art



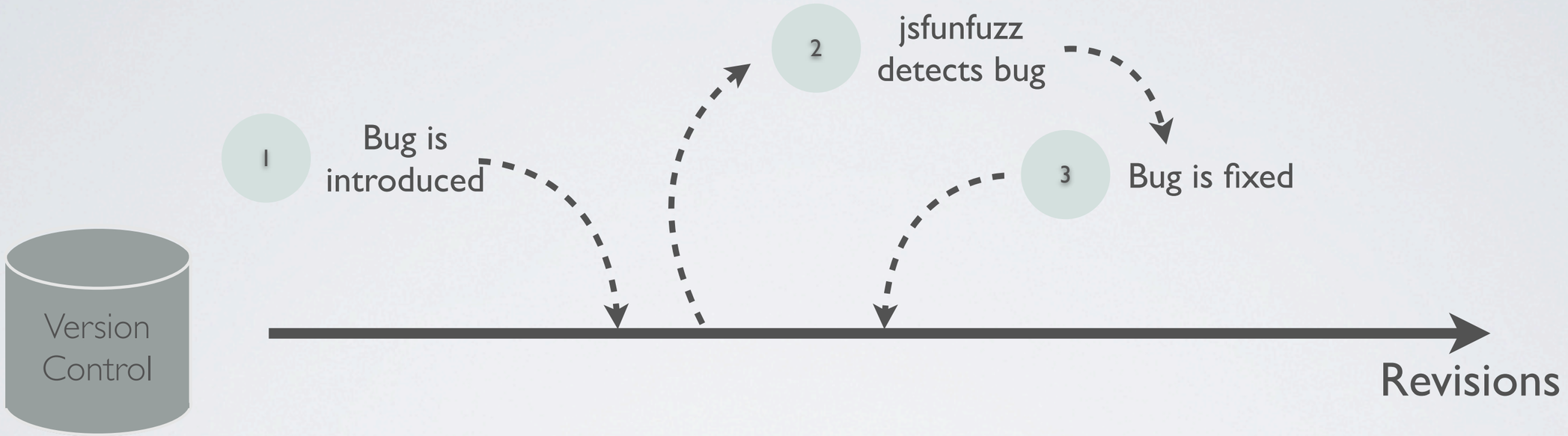
# Comparison with State of the Art



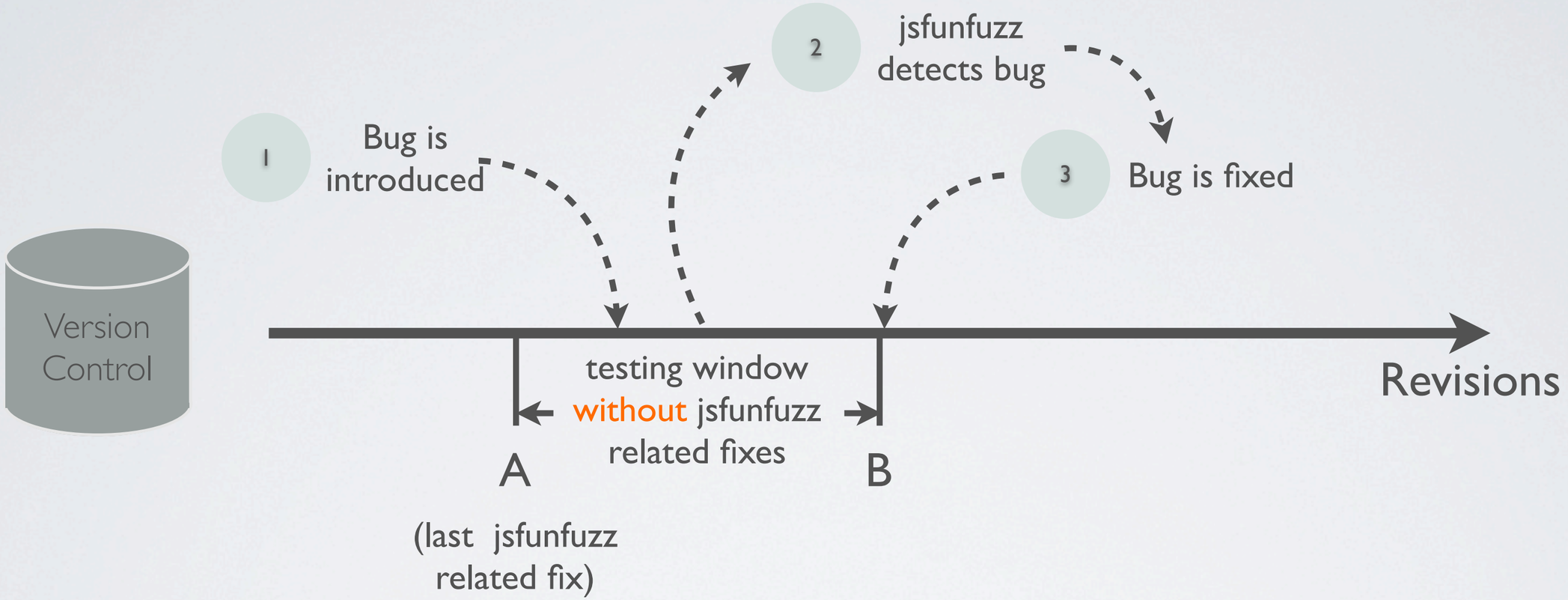
# Comparison with State of the Art



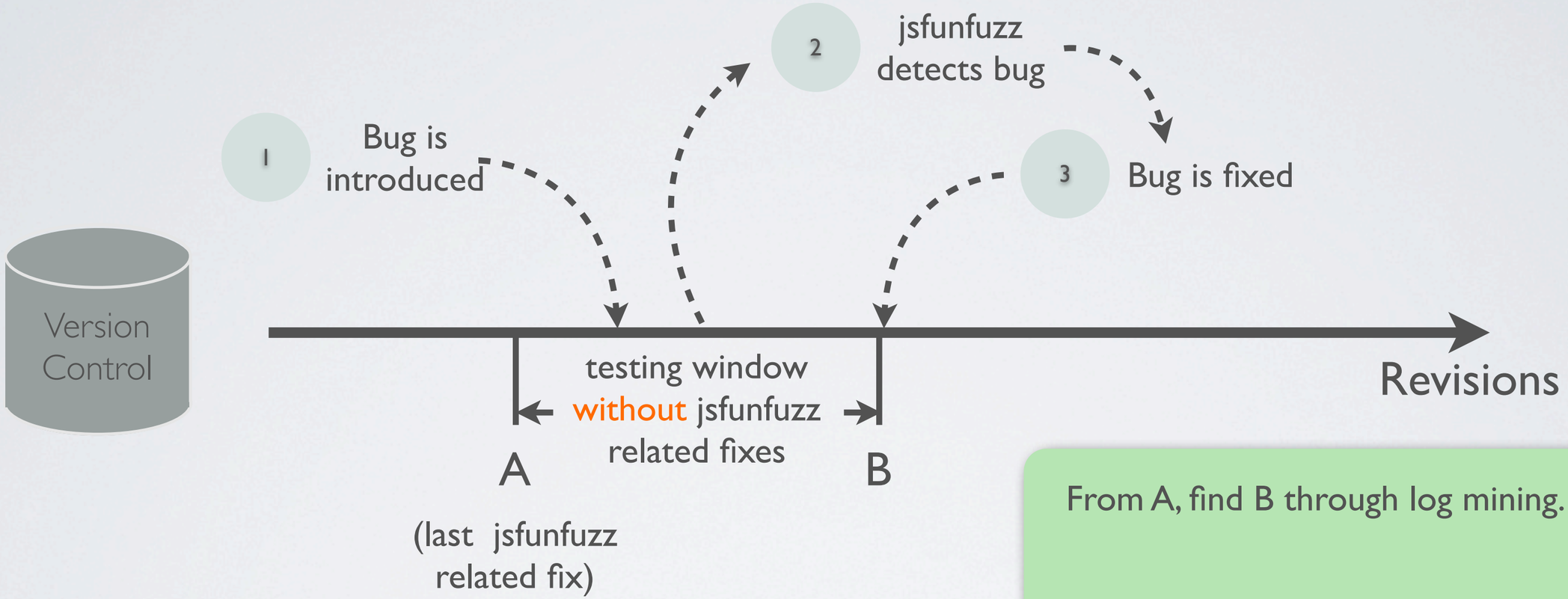
# Comparison with State of the Art



# Comparison with State of the Art

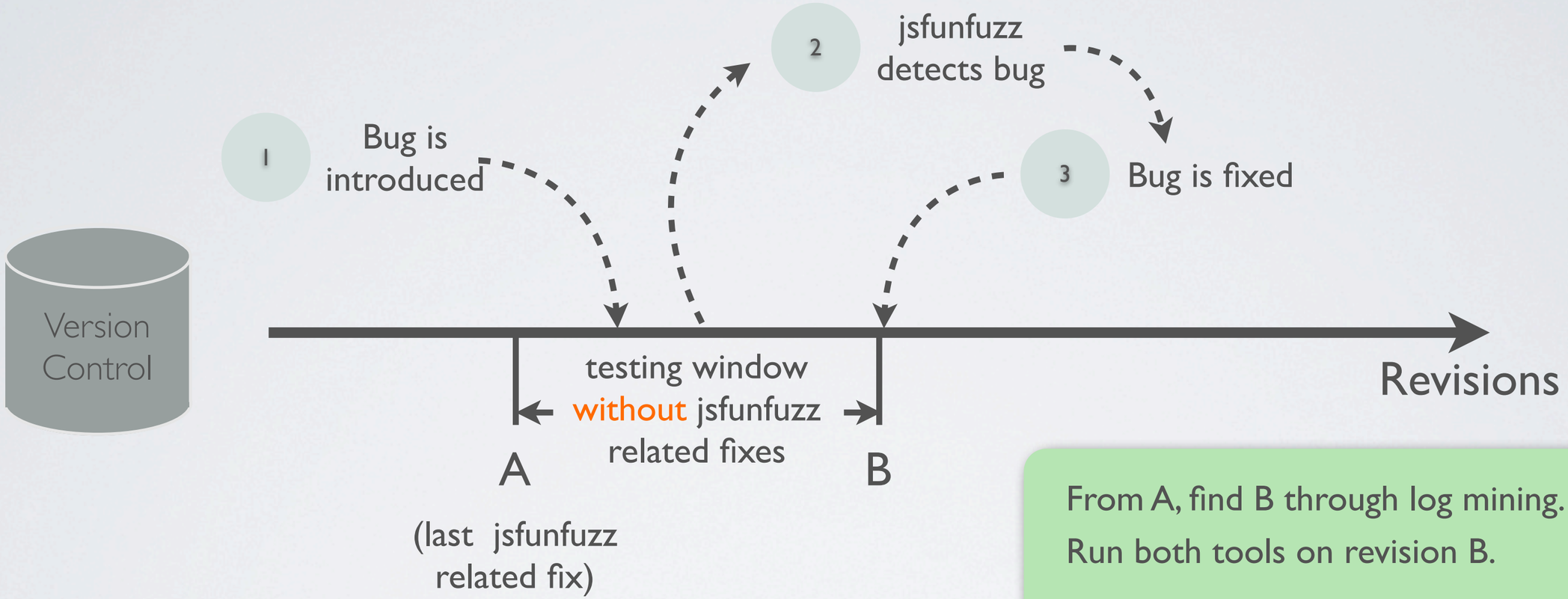


# Comparison with State of the Art




From A, find B through log mining.

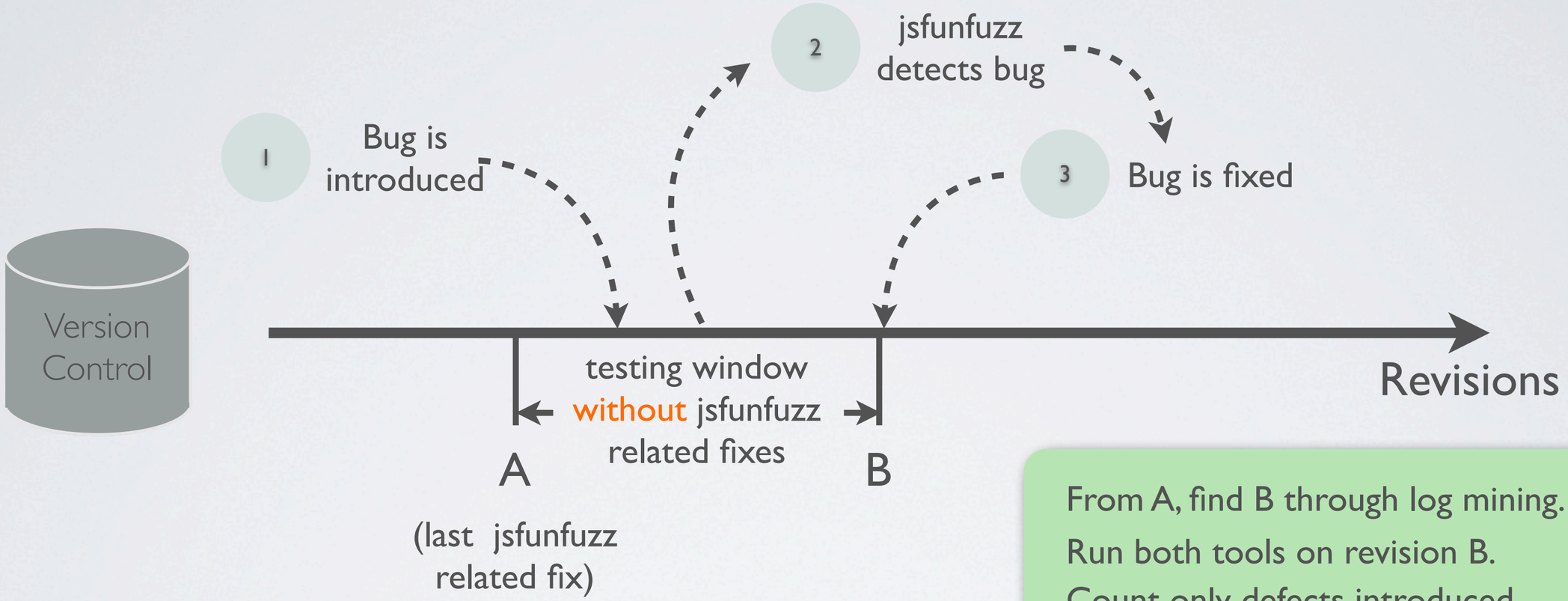
# Comparison with State of the Art




From A, find B through log mining.

Run both tools on revision B.

# Comparison with State of the Art

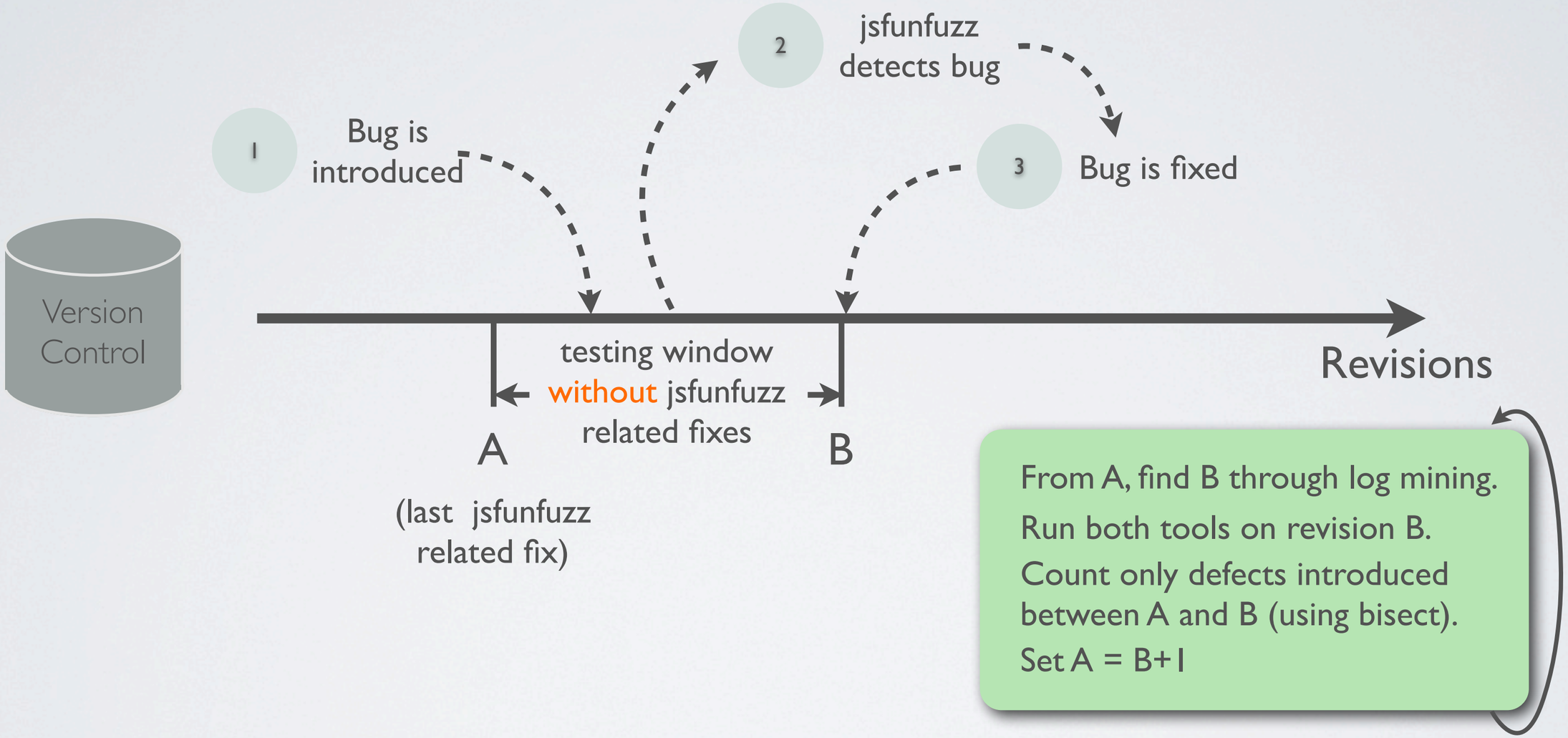



From A, find B through log mining.

Run both tools on revision B.

Count only defects introduced between A and B (using bisect).

# Comparison with State of the Art

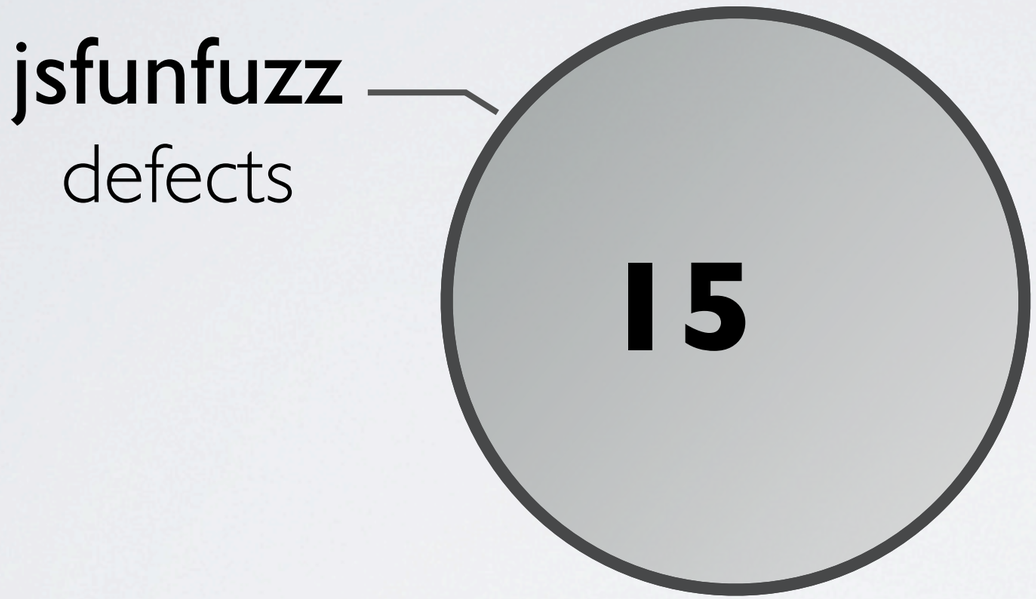


# Comparison with State of the Art

Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development

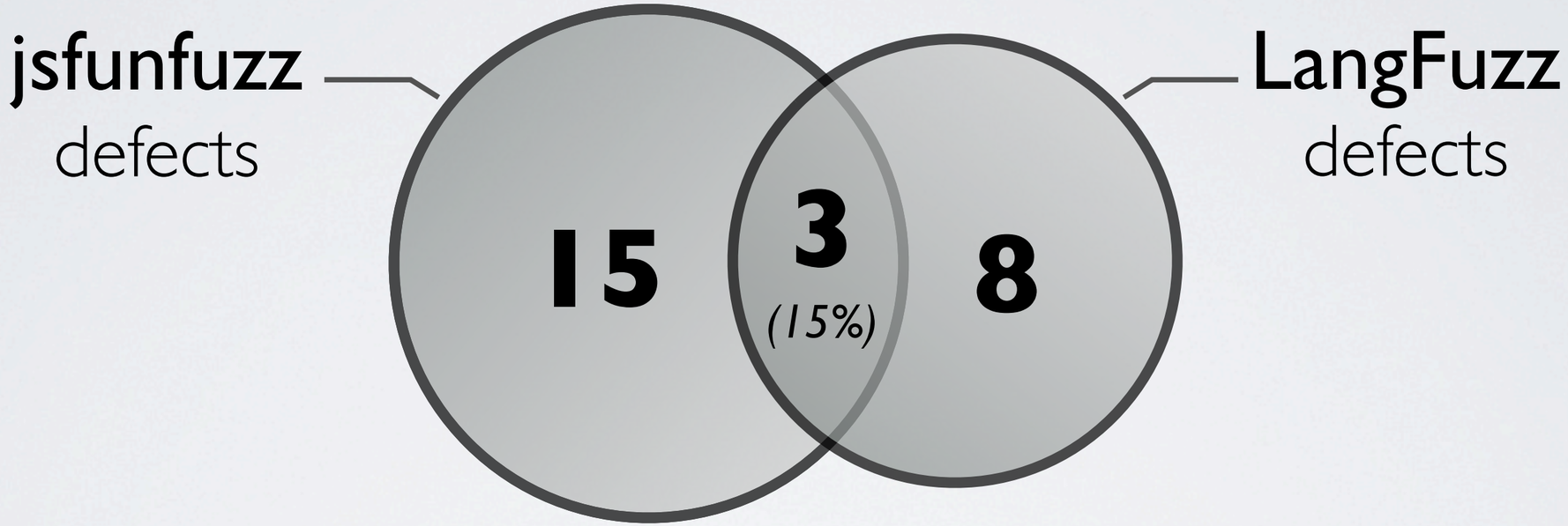
# Comparison with State of the Art

Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development



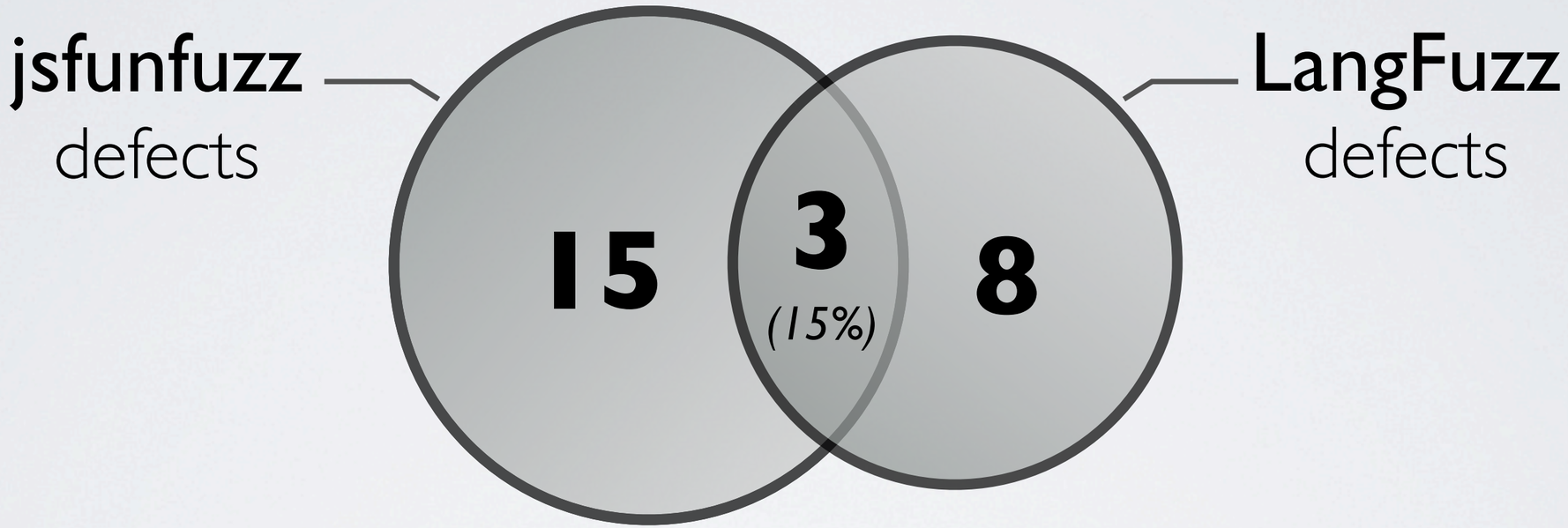
# Comparison with State of the Art

Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development



# Comparison with State of the Art

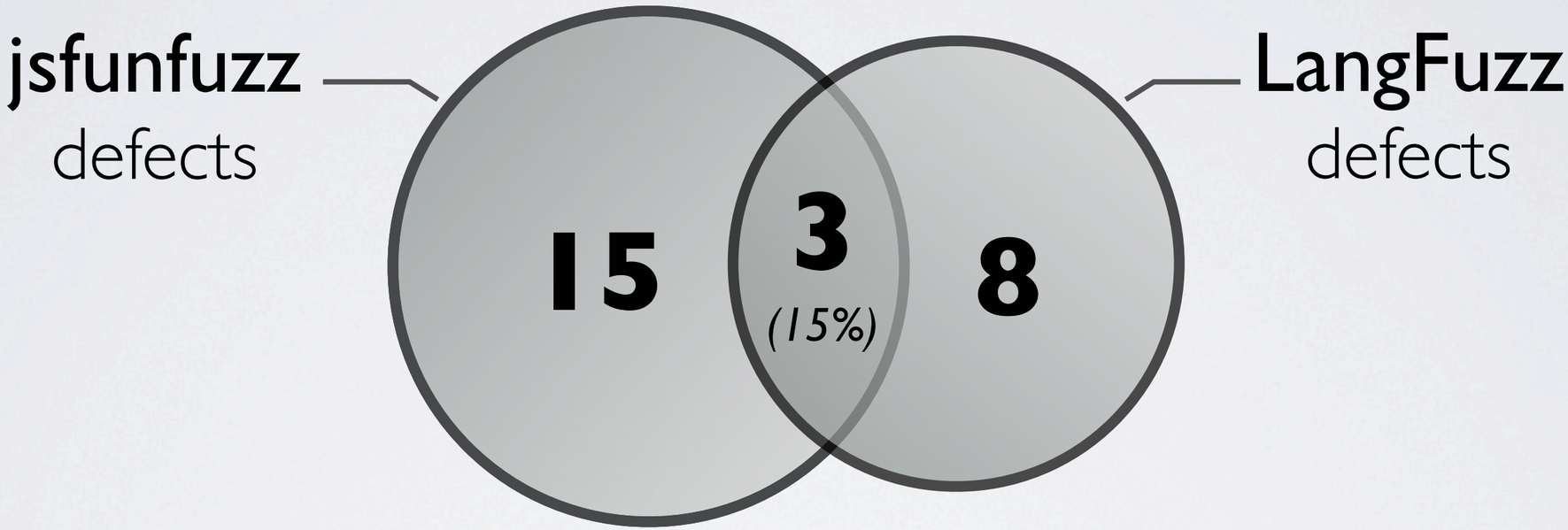
Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development




$$\text{Overlap} = \frac{\#\text{ both}}{\#\text{ all defects}} = \mathbf{15\%}$$

# Comparison with State of the Art

Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development

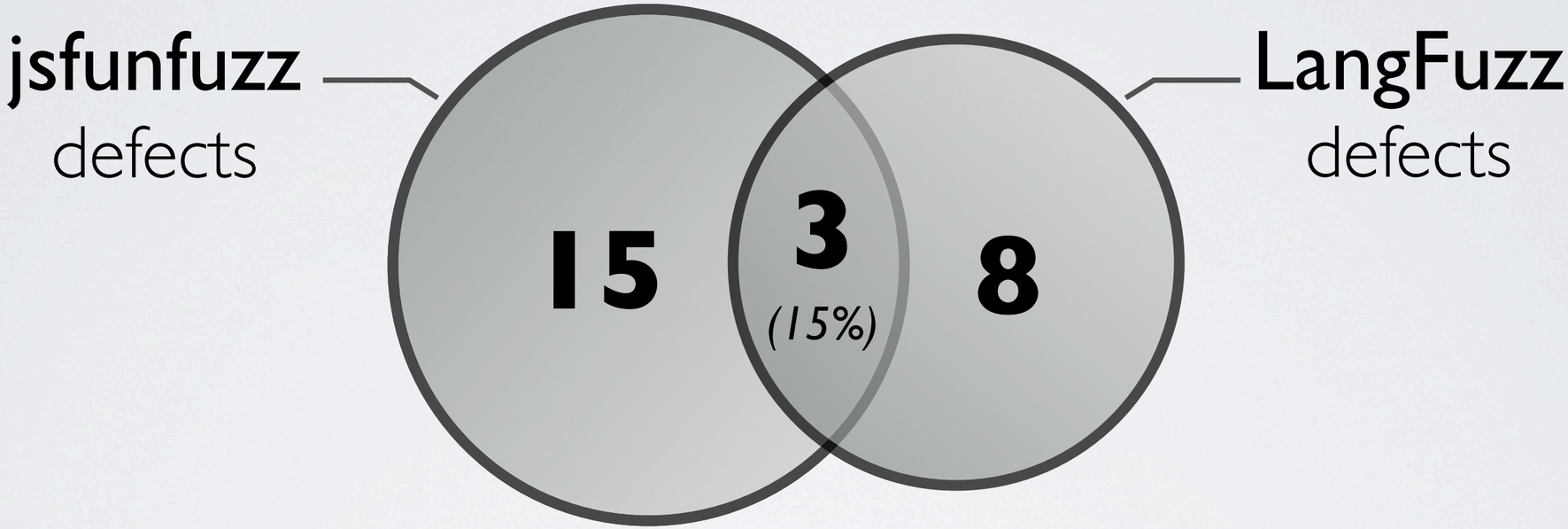



**Overlap** $= \frac{\text{\# both}}{\text{\# all defects}} =$ **15%**

**Effectiveness** $= \frac{\text{\# both + \# LangFuzz}}{\text{\# both + \# jsfunfuzz}} =$ **53%**

# Comparison with State of the Art

Based on five testing windows (5 x 24 h experiment) that cover roughly 2 months of development



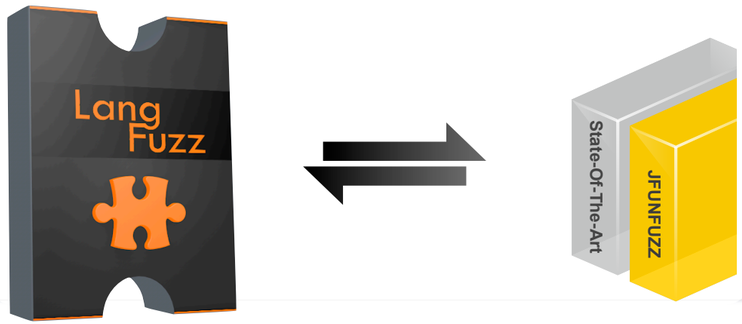

**Overlap** $= \frac{\text{\# both}}{\text{\# all defects}} =$ **15%**

**Effectiveness** $= \frac{\text{\# both + \# LangFuzz}}{\text{\# both + \# jsfunfuzz}} =$ **53%**

Low overlap with affordable performance

# Evaluation Strategies

Comparison with
**State of the Art**



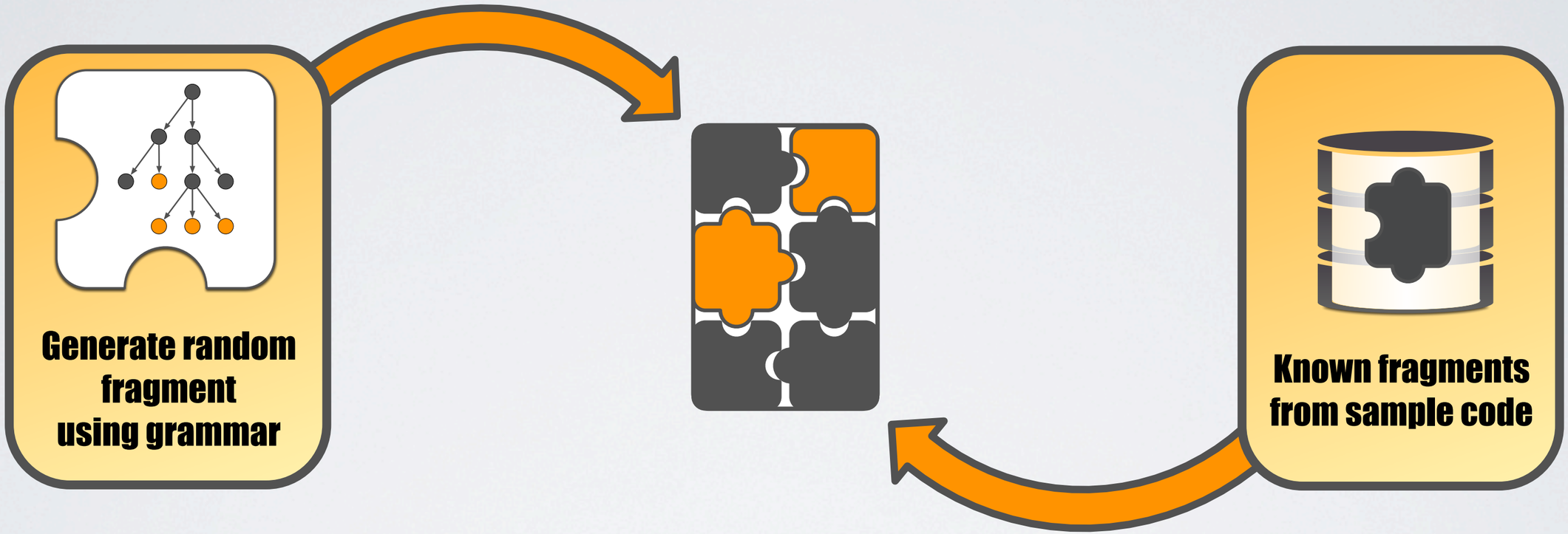

**Internal Comparison**




Field Tests with
**Mozilla** and **Google**

Proof-of-Concept with
**Different Language**

# Internal Comparison



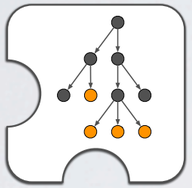

Is code generation even necessary?

# Internal Comparison

Tested the two most defect-prone revisions from previous experiments (2 x 72 hours experiment)

# Internal Comparison

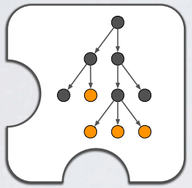
Tested the two most defect-prone revisions from previous experiments (2 x 72 hours experiment)



**Run with 100% generated code.**

# Internal Comparison

Tested the two most defect-prone revisions from previous experiments (2 x 72 hours experiment)



**Run with 100% generated code.**



**Run with 0% generated code.**

# Internal Comparison

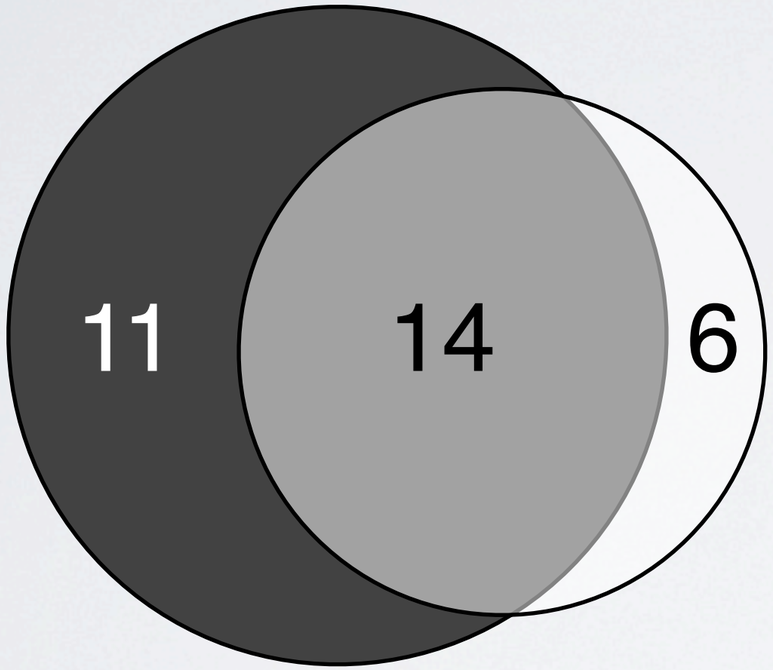


# Internal Comparison

100% generation

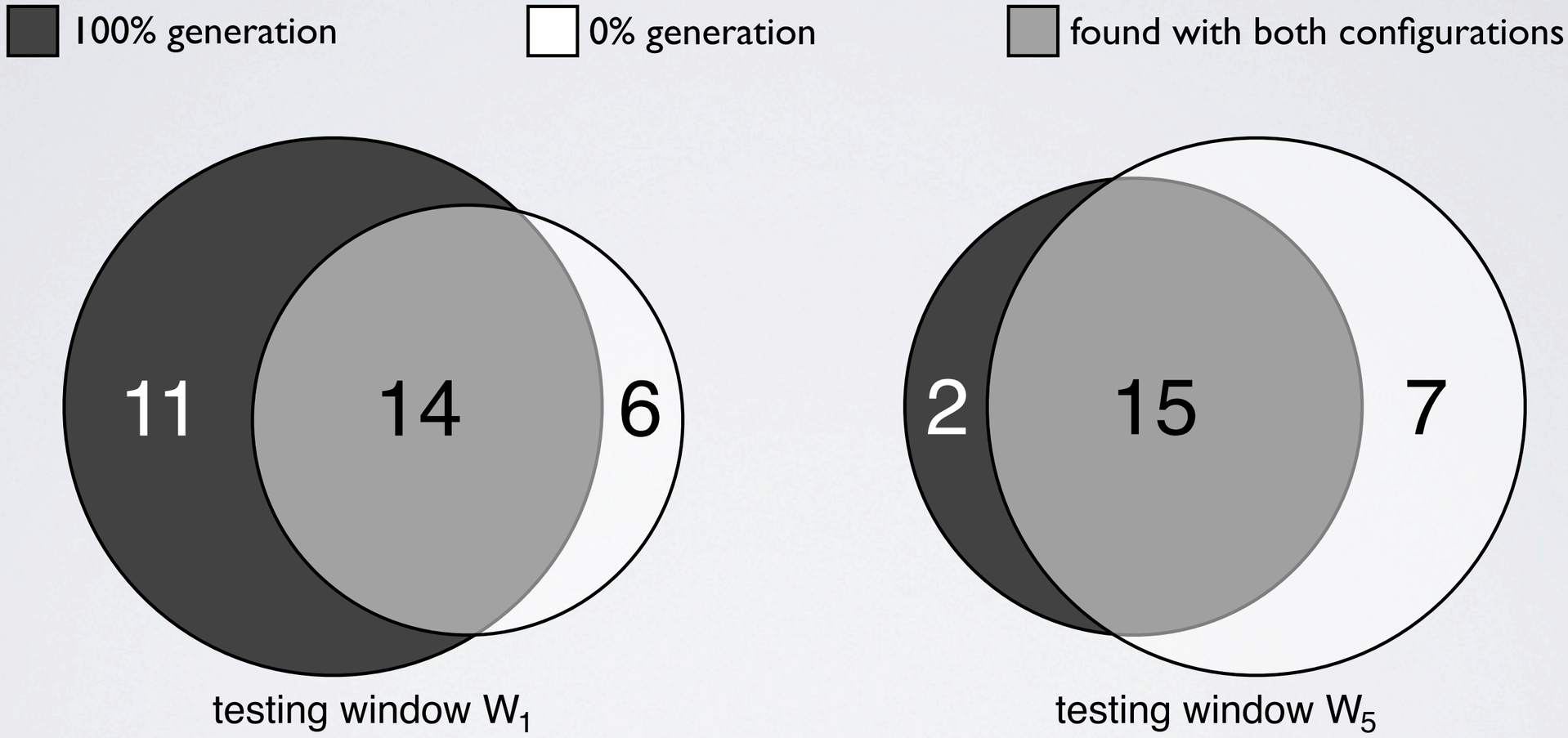
0% generation

found with both configurations

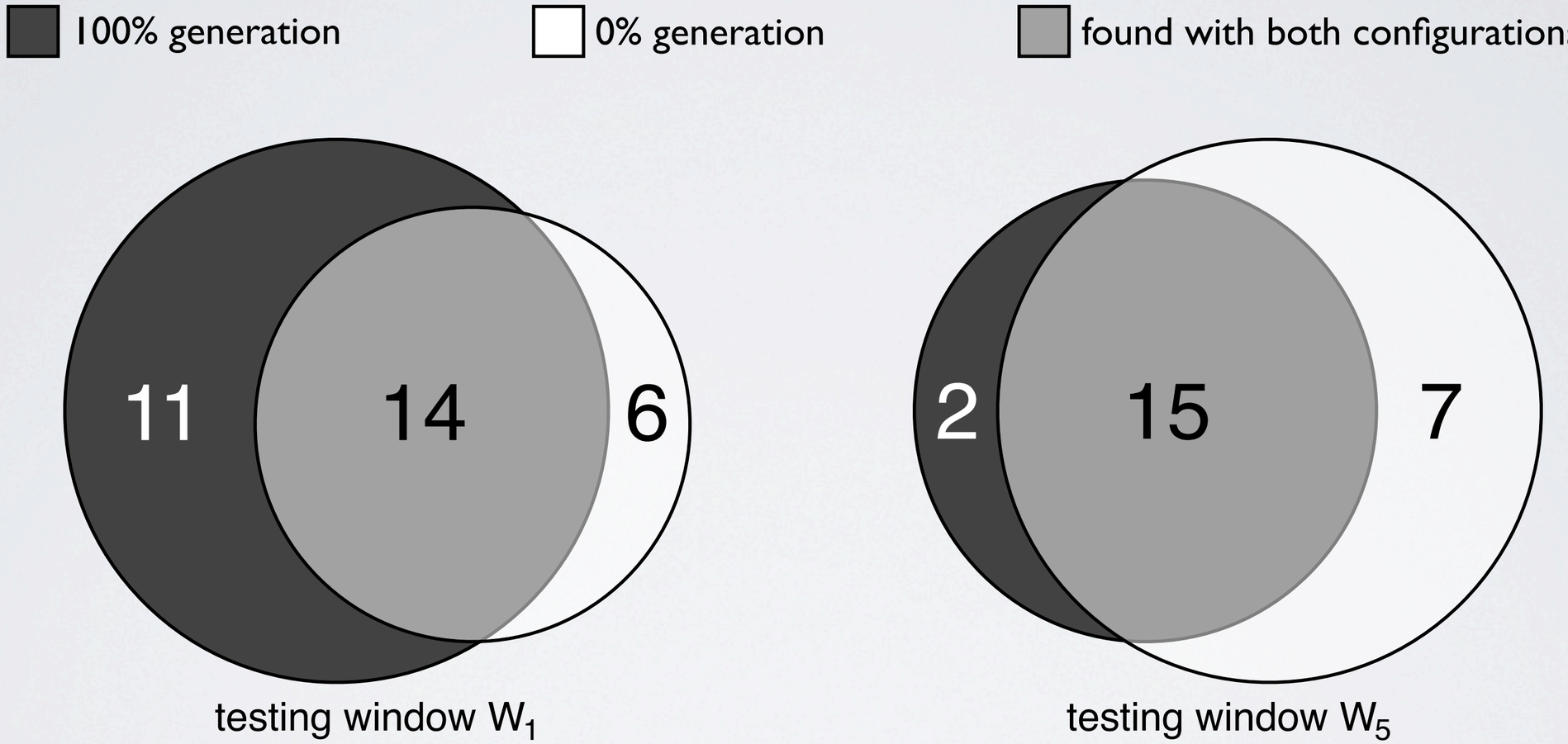



testing window $W_1$

# Internal Comparison



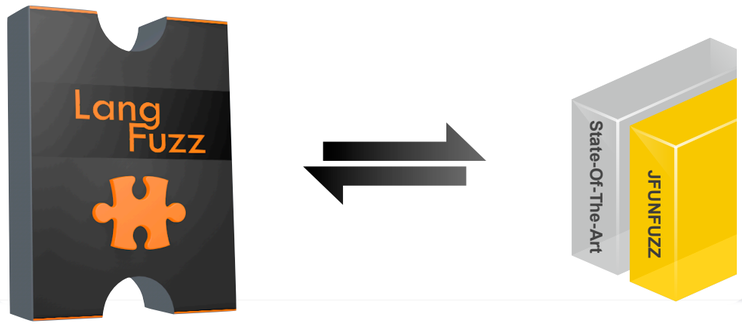
# Internal Comparison




**Both** techniques are required.

# Evaluation Strategies

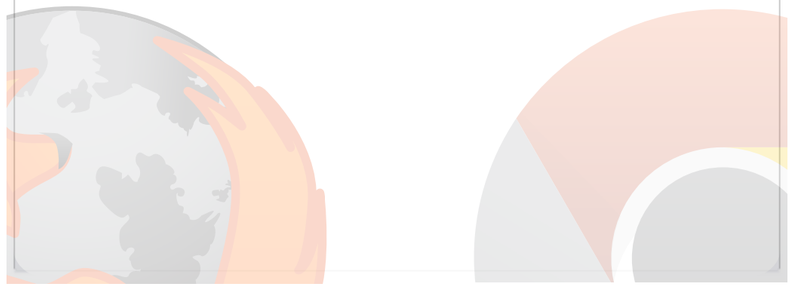
Comparison with
**State of the Art**



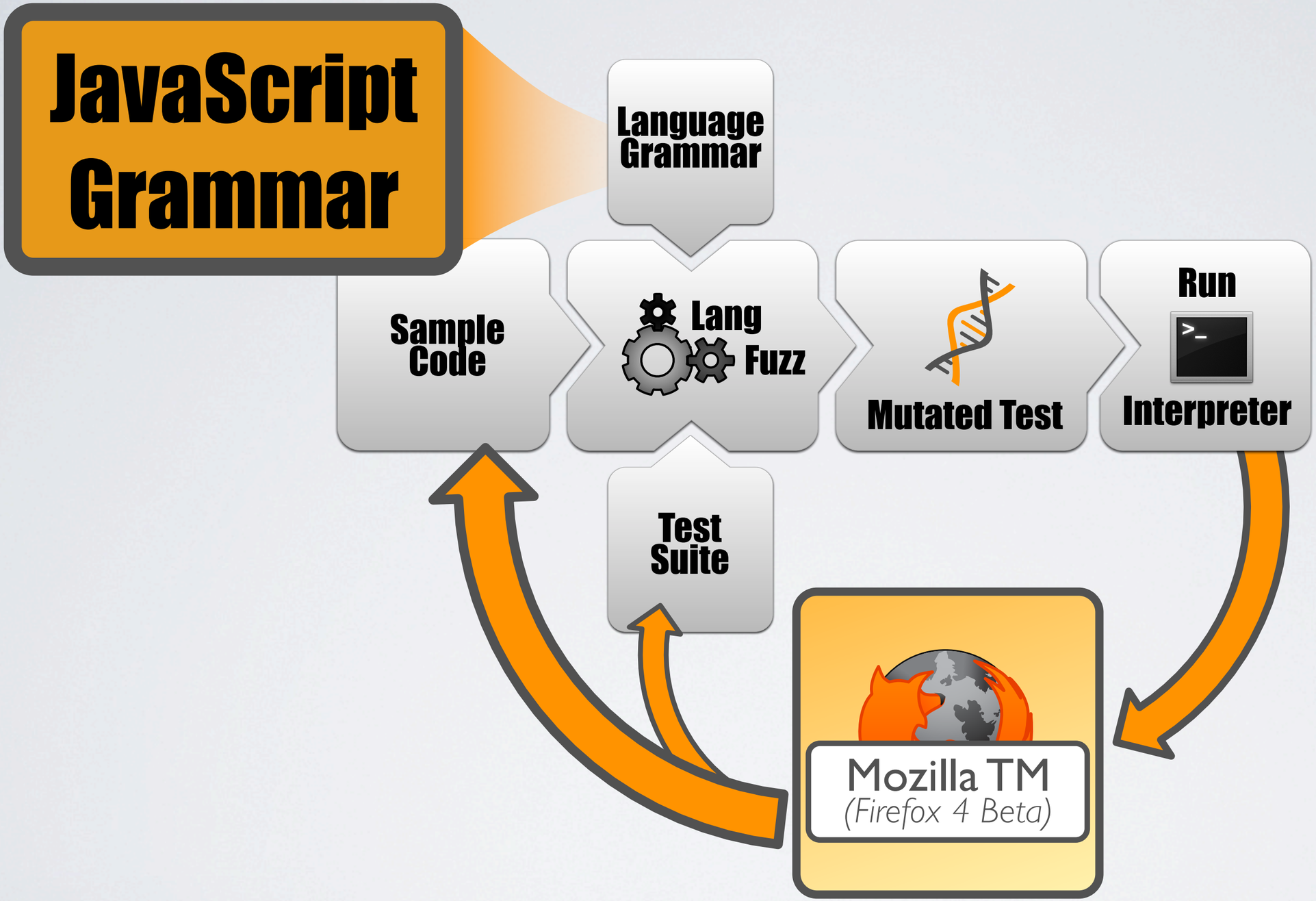

**Internal Comparison**

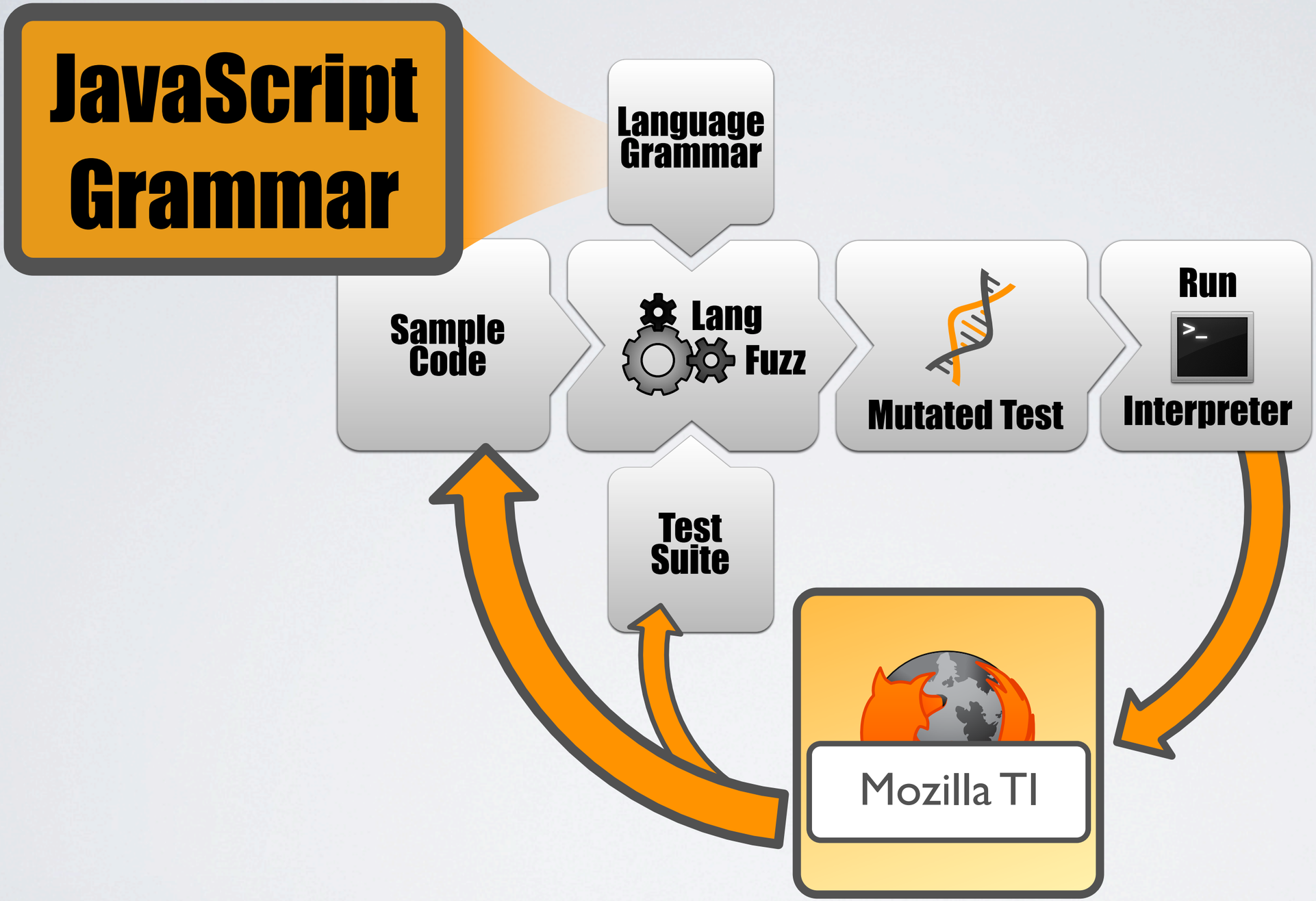


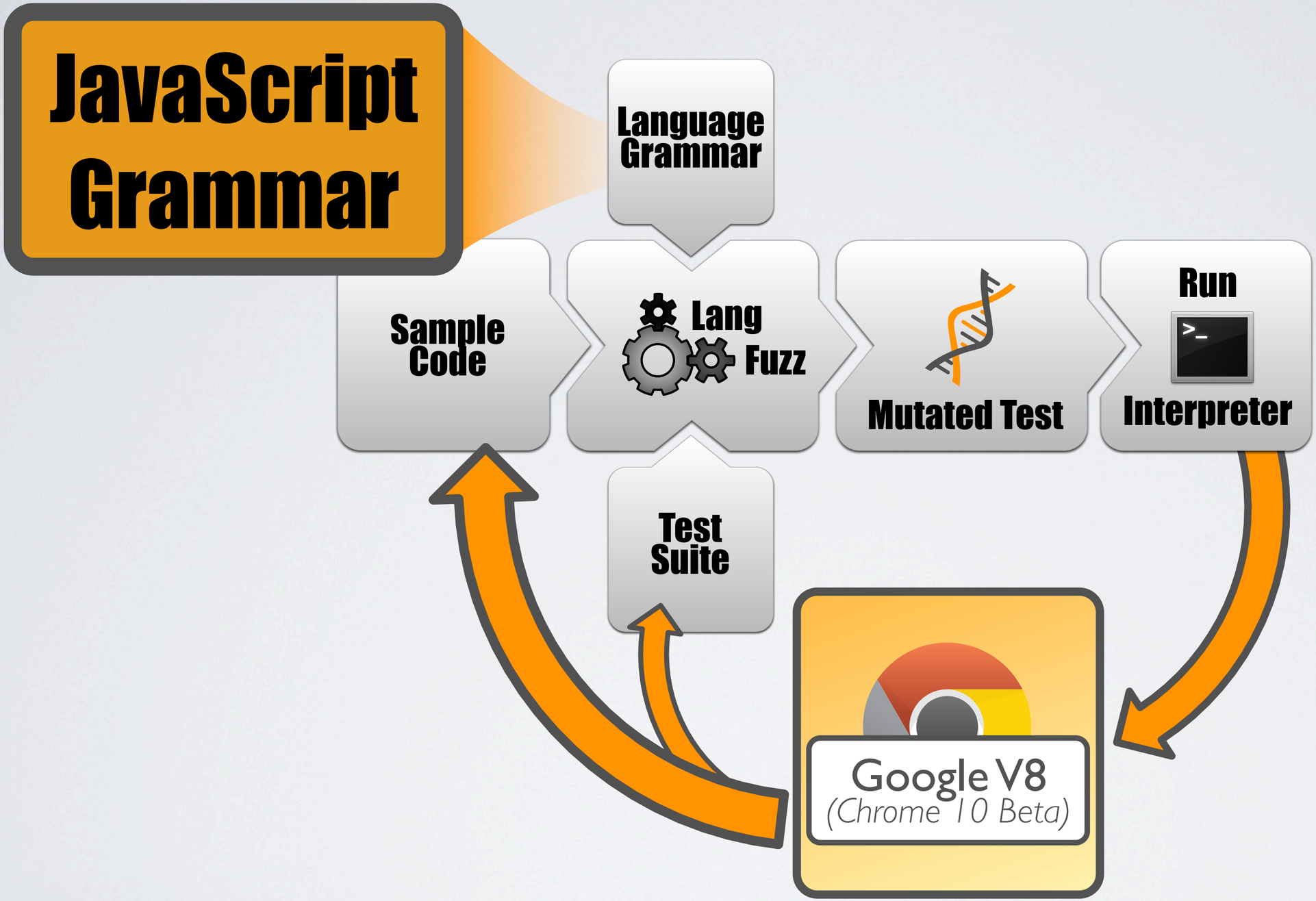

Field Tests with
**Mozilla** and **Google**



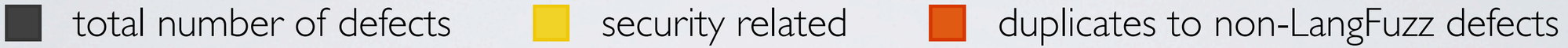
Proof-of-Concept with
**Different Language**


JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Run
Interpreter
Test Suite
Mozilla TM
(Firefox 4 Beta)

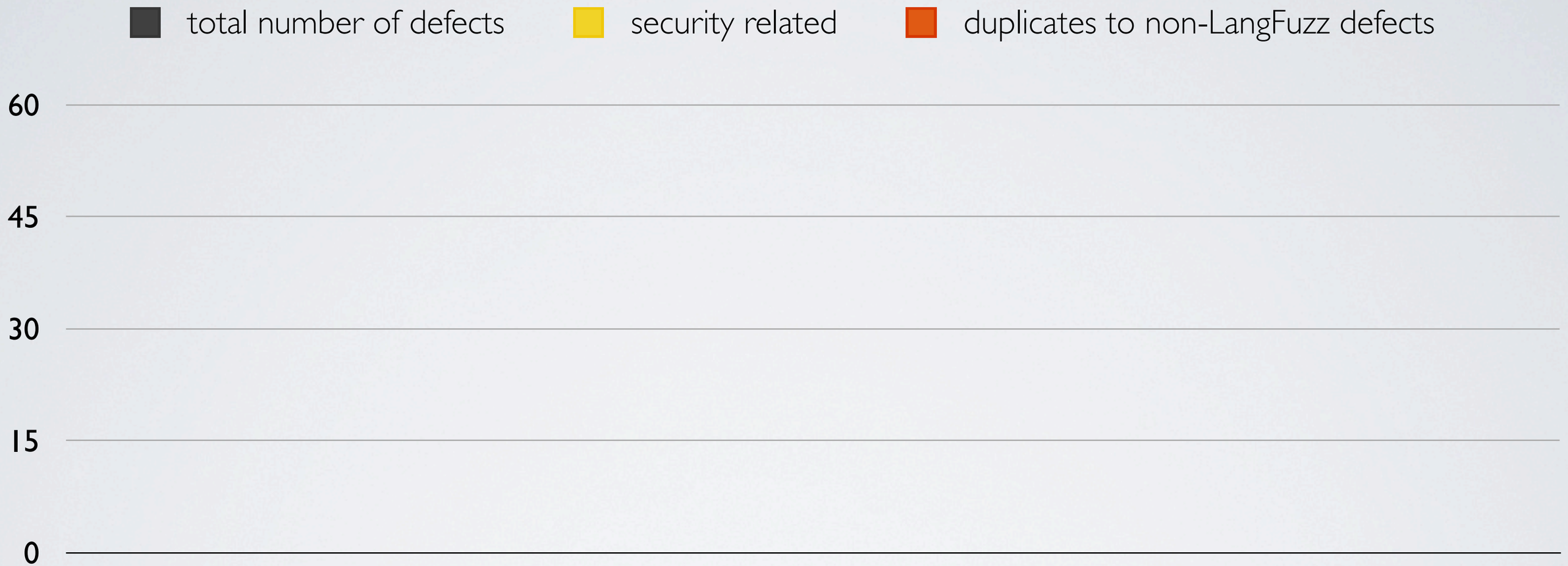

JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Run
Interpreter
Test Suite
Mozilla TI


JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Run
Interpreter
Test Suite
Google V8
(Chrome 10 Beta)

# Field Tests with Mozilla and Google



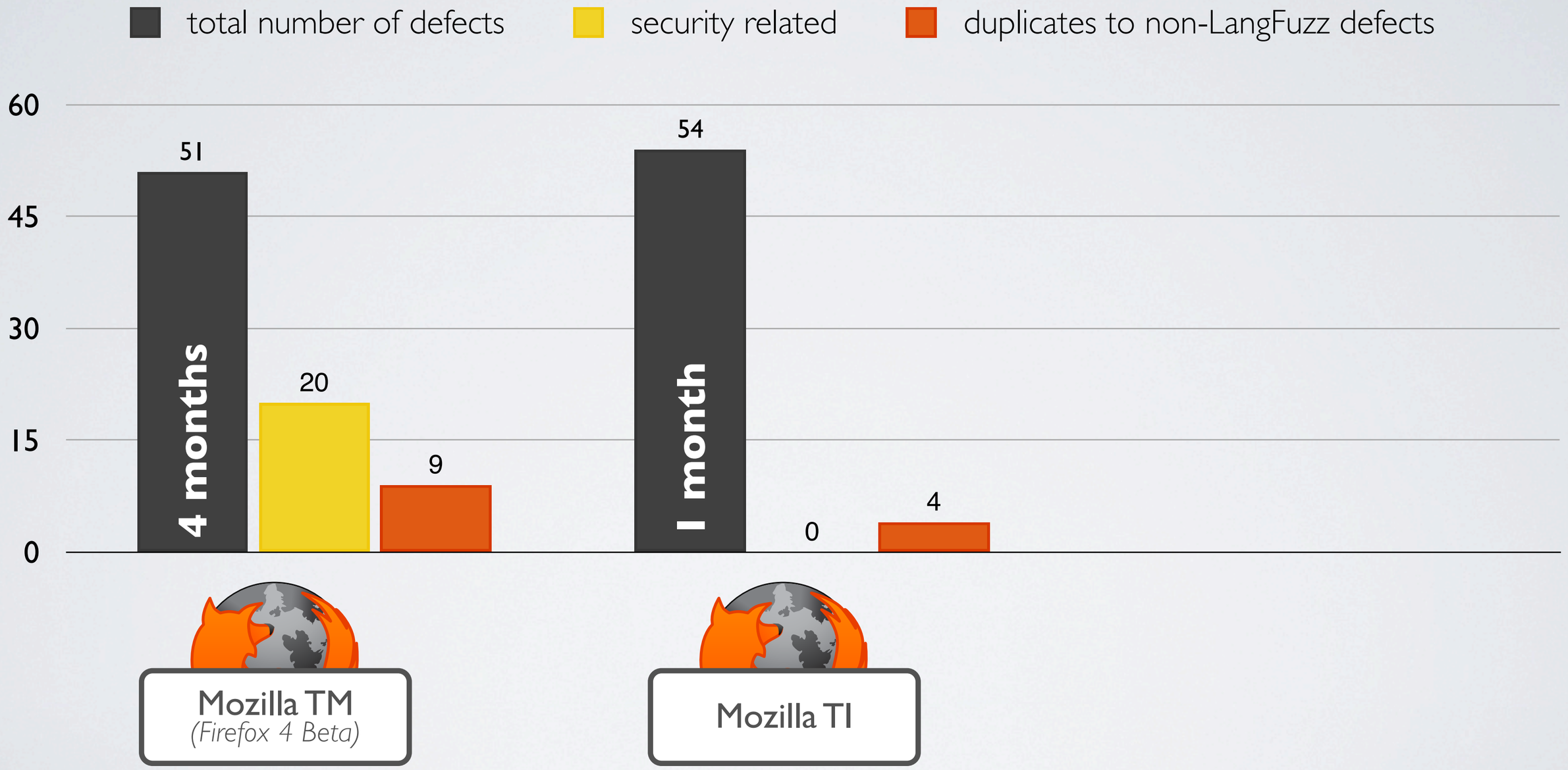
# Field Tests with Mozilla and Google



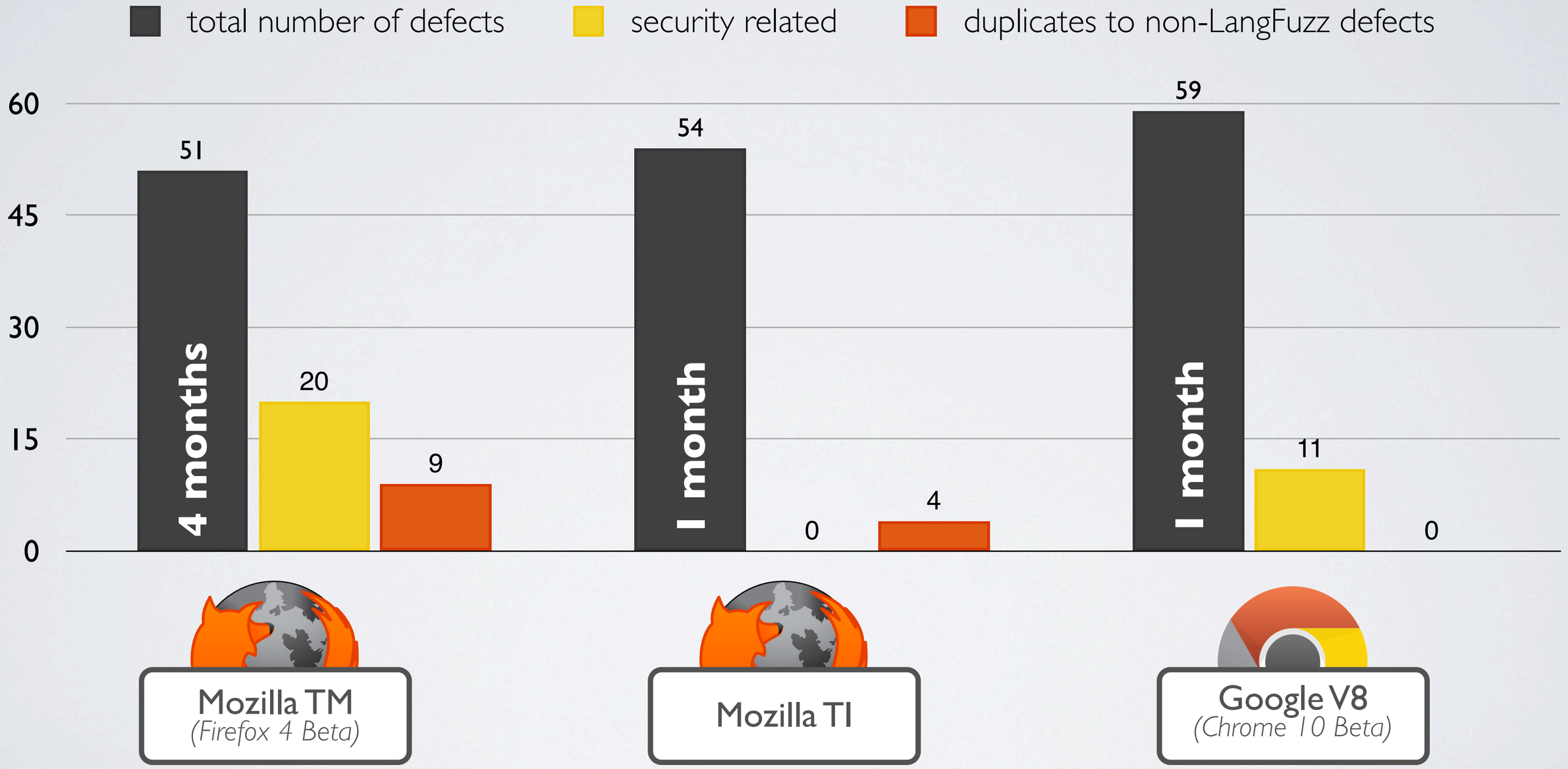
# Field Tests with Mozilla and Google



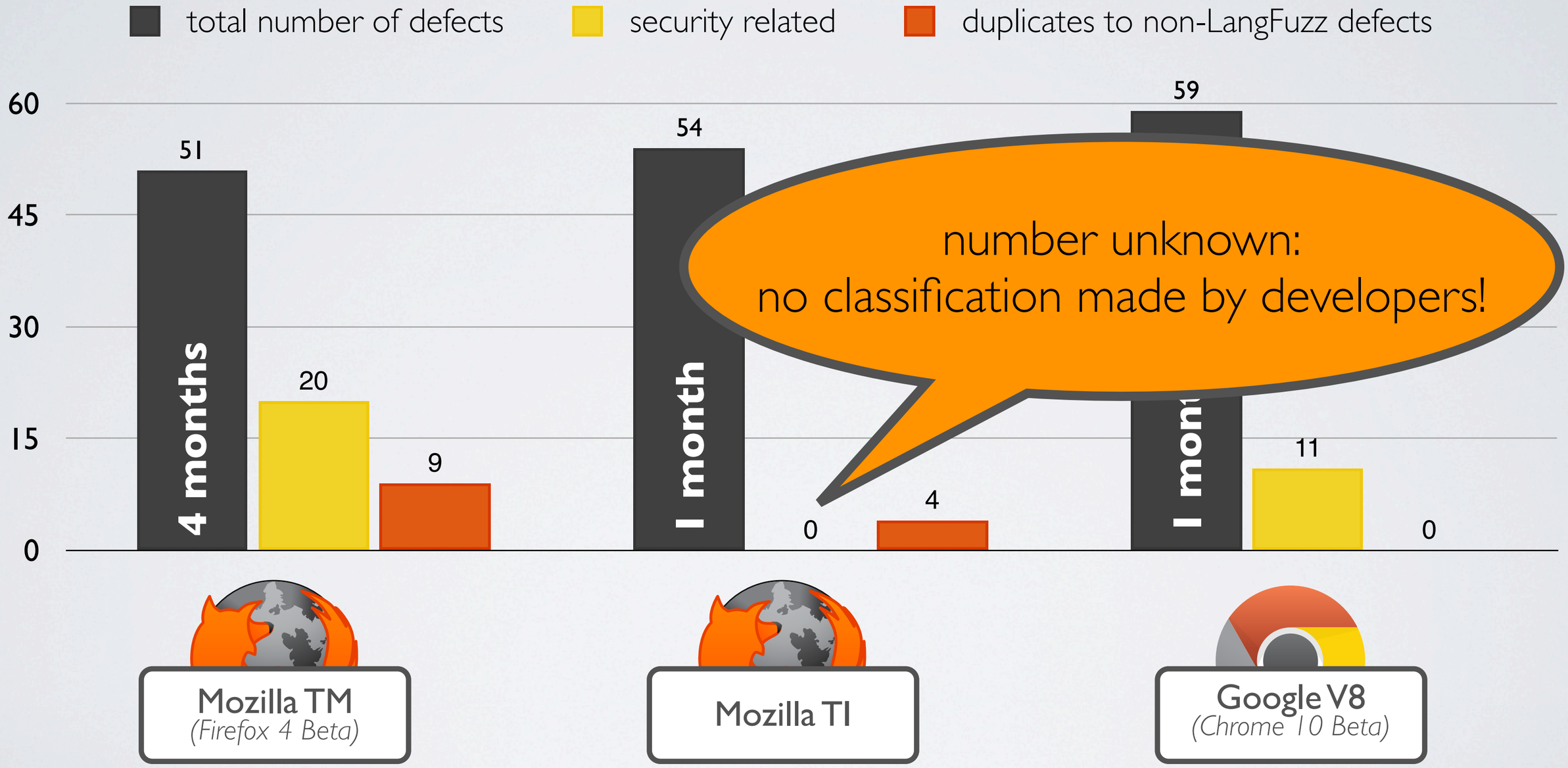
# Field Tests with Mozilla and Google



# Field Tests with Mozilla and Google

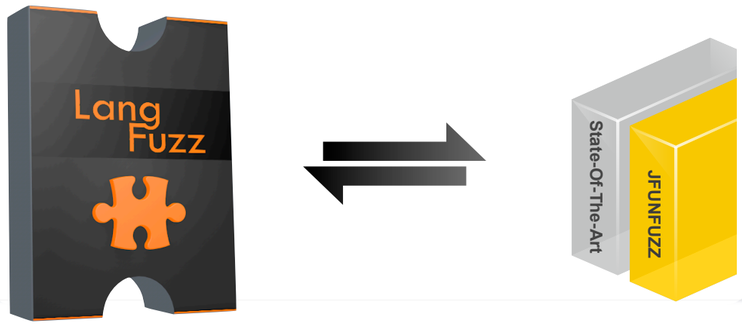


# Field Tests with Mozilla and Google



# Evaluation Strategies

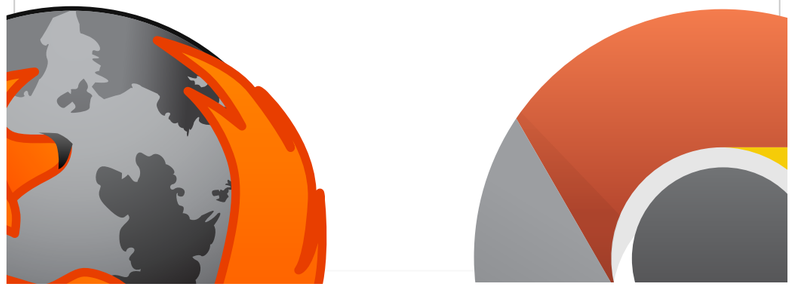
Comparison with
**State of the Art**




**Internal Comparison**

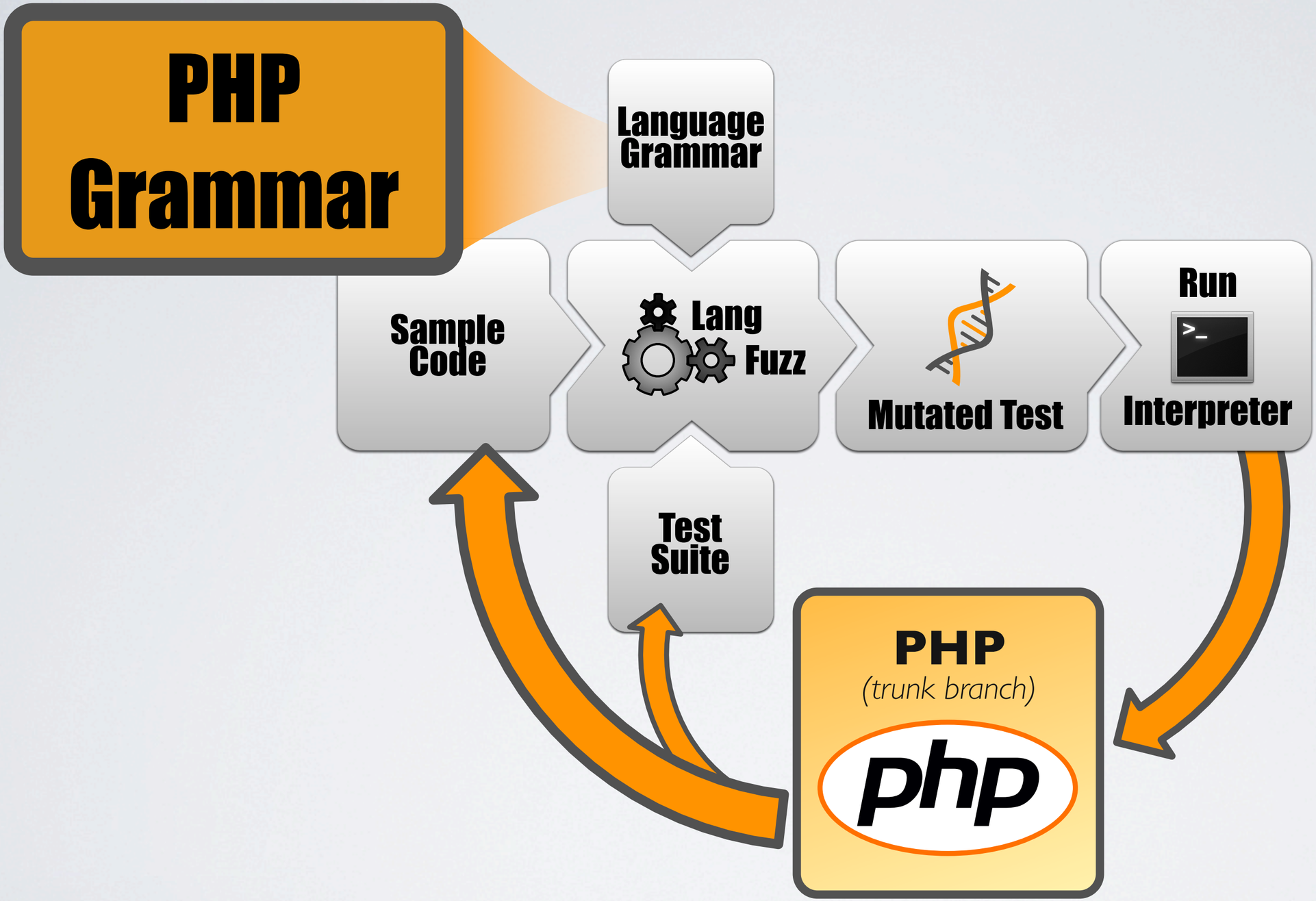



Field Tests with
**Mozilla** and **Google**



Proof-of-Concept with
**Different Language**




PHP Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Run
Interpreter
Test Suite
PHP
(trunk branch)
php

# Proof-of-Concept with Different Language

found **18 bugs** in **two** weeks

***54280***: *Crash with substr_replace and array*
***54281***: *Crash in spl_recursive_it_rewind.ex*
***54282***: *Crash in zend-mm-check-ptr*
***54283***: *Crash in zen_object_store_get_object*
***54284***: *Crash in zen_object_store_get_object*
***54285***: *Crash in _zval_ptr_dtor*
***54291***: *Crash in spl_filesystem_object_get_path*
***54292***: *Wrong parameter causes crash in SplFileObject::__construct()*
***54296***: *Crash in SQLite3Stmt internal object destructor*
***54304***: *Crash in php_pcre_replace_impl*
***54305***: *Crash in gc_remove_zval_from_buffer*
***54322***: *Crash (null pointer) in zif_get_html_translation_table*
***54323***: *Accessing unset()'ed ArrayObject's property causes crash*
***54324***: *Crash in date_object_compare_date*
***54332***: *Crash in zend_mm_check_ptr Heap corruption*
***54348***: *Crash (Call stack overflow) in ExchangeArray*
***54349***: *Crash in zend_std_write_property*
***54350***: *Memory corruption with user_filter*

# Proof-of-Concept with Different Language

**54280**: *Crash with substr_replace and array*
**54281**: *Crash in spl_recursive_it_rewind.ex*
**54282**: *Crash in zend-mm-check-ptr*
**54283**: *Crash in zen_object_store_get_object*
**54284**: *Crash in zen_object_store_get_object*
**54285**: *Crash in _zval_ptr_dtor*
**54291**: *Crash in spl_filesystem_object_get_path*
**54292**: *Wrong parameter causes crash in SplFileObject::__construct()*
**54296**: *Crash in SQLite3Stmt internal object destructor*
**54304**: *Crash in php_pcre_replace_impl*
**54305**: *Crash in gc_remove_zval_from_buffer*
**54322**: *Crash (null pointer) in zif_get_html_translation_table*
**54323**: *Accessing unset()'ed ArrayObject's property causes crash*
**54324**: *Crash in date_object_compare_date*
**54332**: *Crash in zend_mm_check_ptr Heap corruption*
**54348**: *Crash (Call stack overflow) in ExchangeArray*
**54349**: *Crash in zend_std_write_property*
**54350**: *Memory corruption with user_filter*

found **18 bugs** in **two** weeks

Technique works with **PHP** as well

# Can I use LangFuzz on any language?



Languages using static typing (Java, C++, ...) will most likely give poor results



Requires a good amount of tests/examples

# Can I use LangFuzz on any language?



Languages using static typing (Java, C++, ...) will most likely give poor results

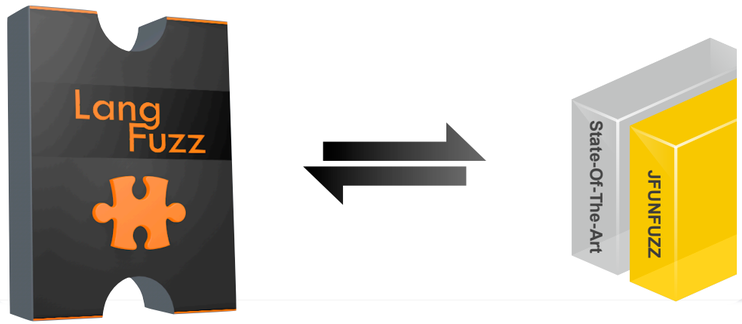


Requires a good amount of tests/examples

Works best on **dynamically, weakly typed** languages with a lot of tests

# Evaluation Strategies

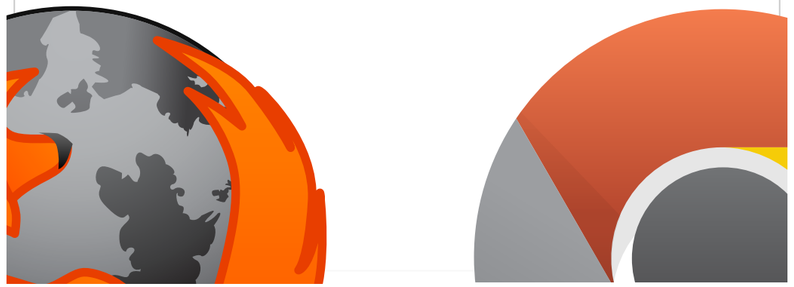
Comparison with
**State of the Art**




**Internal Comparison**



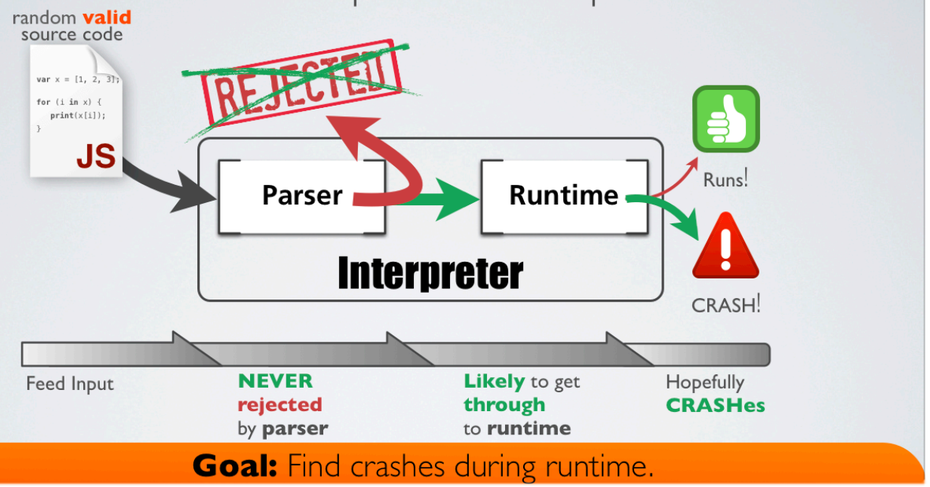

Field Tests with
**Mozilla** and **Google**



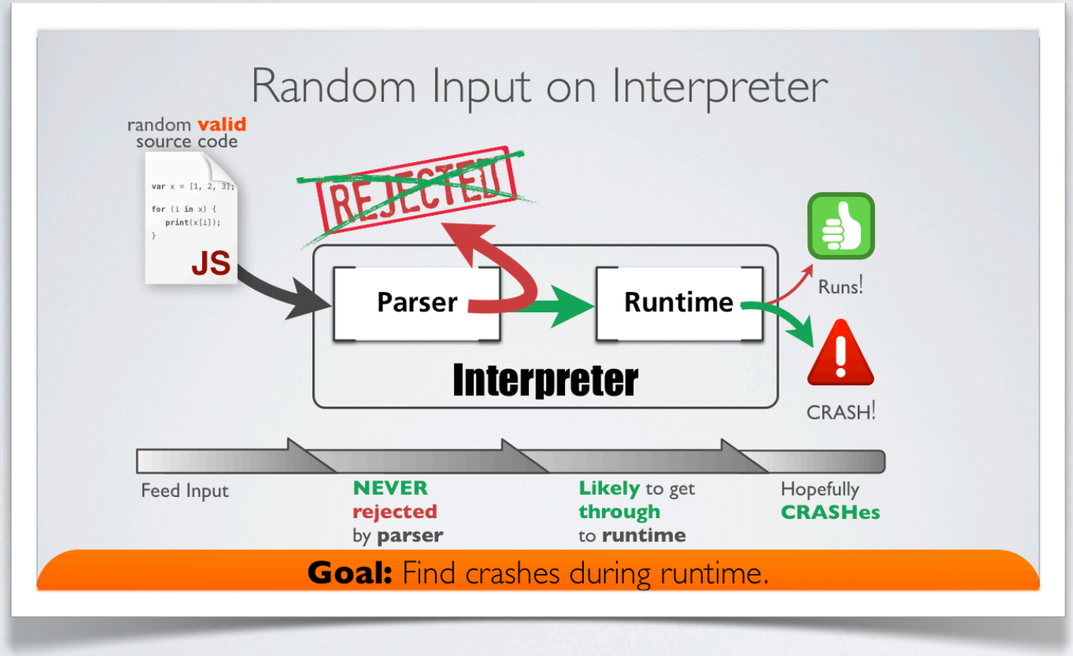
Proof-of-Concept with
**Different Language**



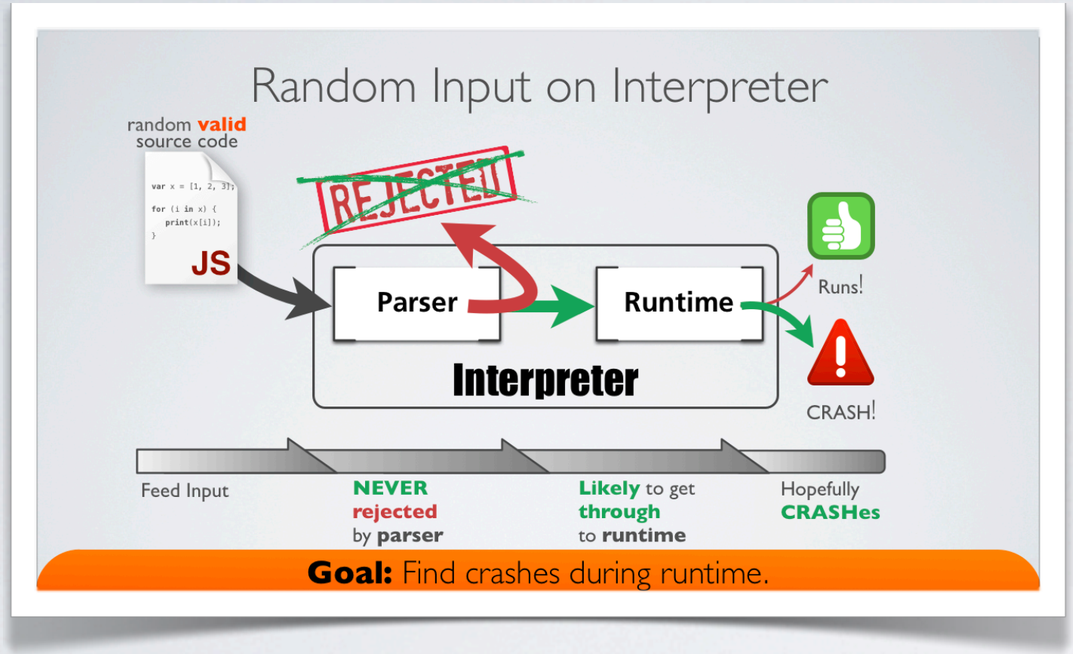
# Random Input on Interpreter



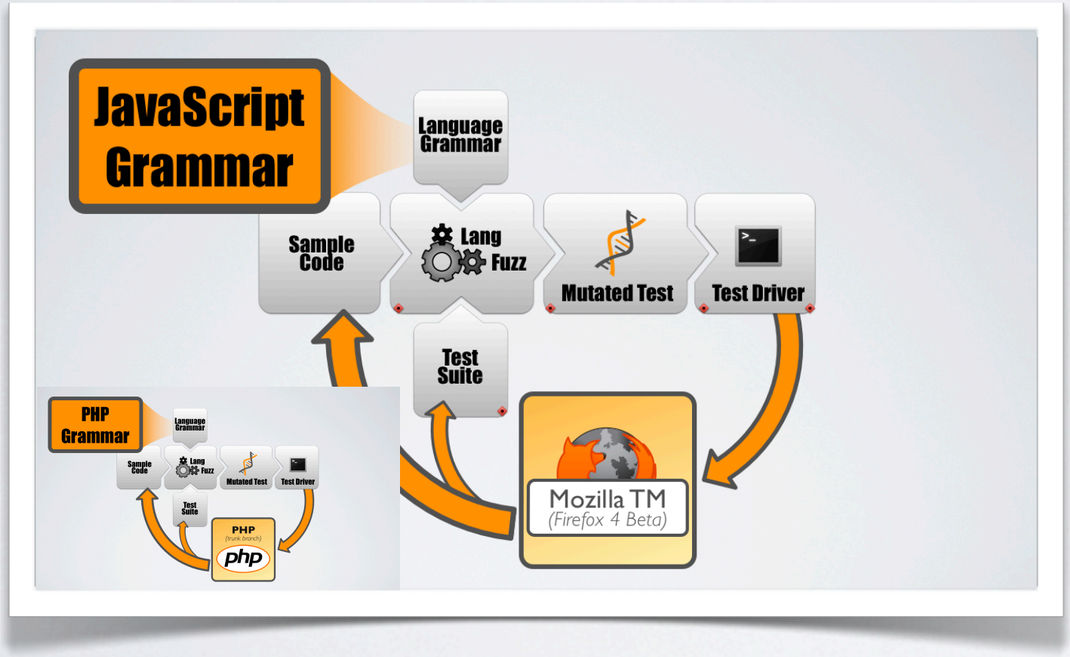

**Goal:** Find crashes during runtime.


Random Input on Interpreter
random valid source code
var x = [1, 2, 3];
for (i in x) {
print(x[i]);
}
JS
REJECTED
Parser
Runtime
Interpreter
Runs!
CRASH!
Feed Input
NEVER rejected by parser
Likely to get through to runtime
Hopefully CRASHes
Goal: Find crashes during runtime.

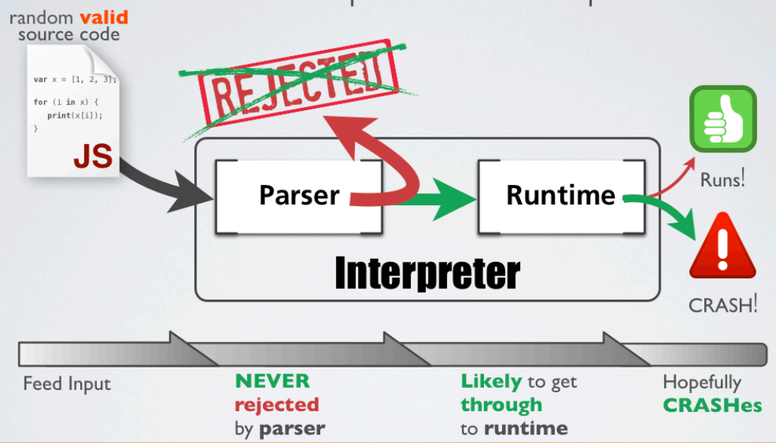

Lang Fuzz
Workflow
Language Grammar
Sample Code
Lang Fuzz
Test Suite
Phase 1
Learning code fragments from sample code and test suite
Mutated Test
Phase 2
LangFuzz generated (mutated) test cases
Test Driver
Phase 3
Feed test case into interpreter, check for crashes and assertions


Random Input on Interpreter
random valid source code
var x = [1, 2, 3];
for (i in x) {
print(x[i]);
}
JS
REJECTED
Parser
Runtime
Interpreter
Runs!
CRASH!
Feed Input
NEVER rejected by parser
Likely to get through to runtime
Hopefully CRASHes
Goal: Find crashes during runtime.


Lang Fuzz
Workflow
Language Grammar
Sample Code
Lang Fuzz
Test Suite
Mutated Test
Test Driver
Phase 1
Learning code fragments from sample code and test suite
Phase 2
LangFuzz generated (mutated) test cases
Phase 3
Feed test case into interpreter, check for crashes and assertions


JavaScript Grammar
Language Grammar
Sample Code
Lang Fuzz
Mutated Test
Test Driver
Test Suite
PHP Grammar
Mozilla TM
(Firefox 4 Beta)

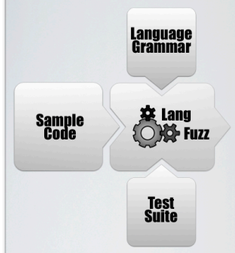
## Random Input on Interpreter




**Goal:** Find crashes during runtime.




## Workflow




**Phase 1**

Learning code fragments from sample code and test suite



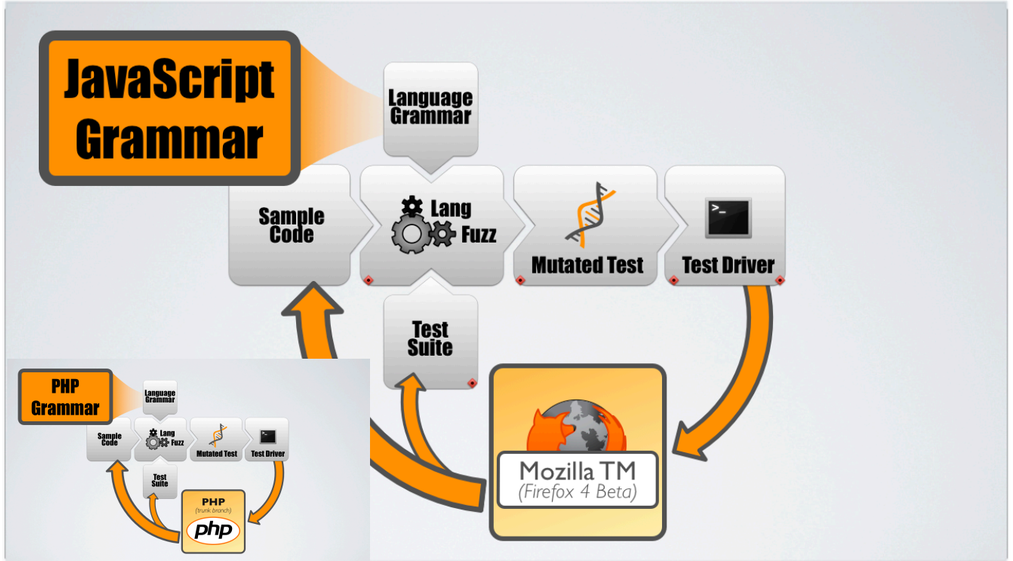

**Phase 2**

LangFuzz generated (mutated) test cases



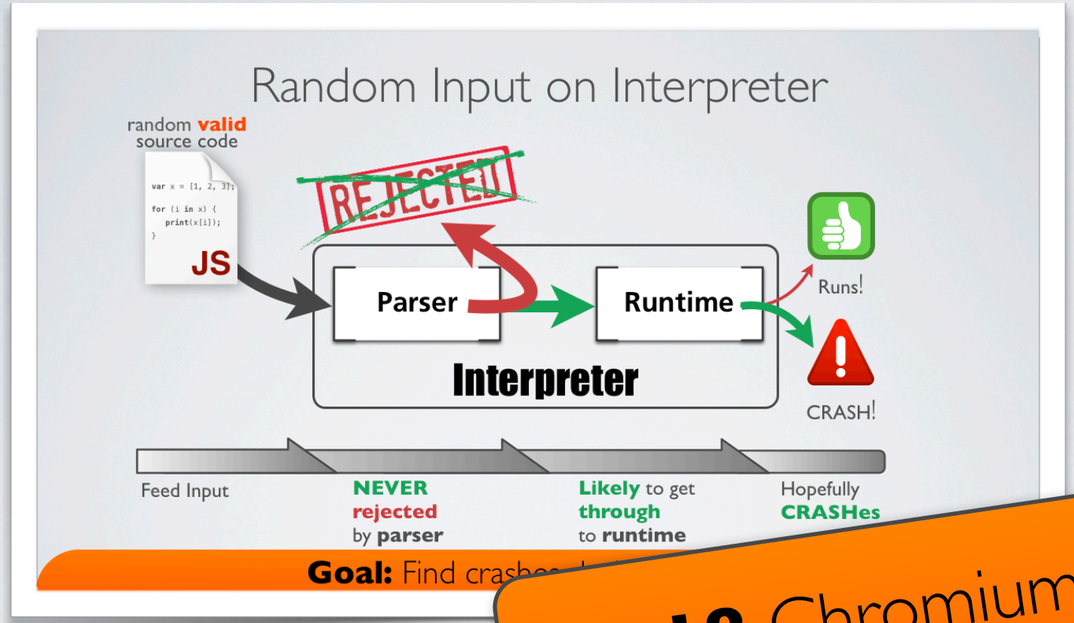

**Phase 3**

Feed test case into interpreter, check for crashes and assertions




## Field Tests with Mozilla and Google












**18** Chromium Security Rewards
**12** Mozilla Security Bug Bounty Awards
in **9 months**

