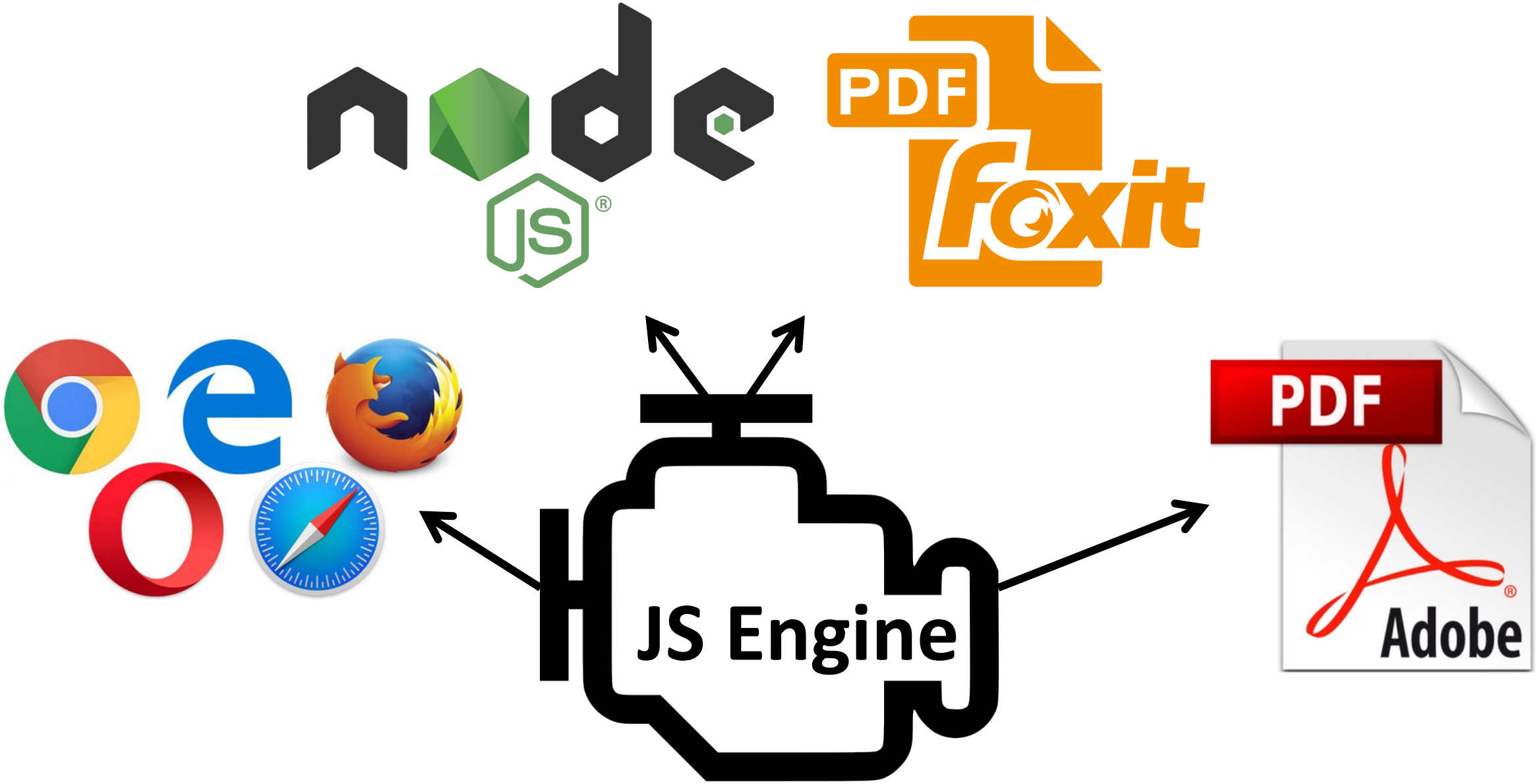
Favocado: Fuzzing the Binding Code of JavaScript Engines Using Semantically Correct Test Cases

Sung Ta Dinh, **Haehyun Cho**, Kyle Martin, Adam Oest, Kyle Zeng,
Alexandros Kapravelos, Gail-Joon Ahn, Tiffany Bao, Ruoyu Wang,
Adam Doupé, and Yan Shoshitaishvili

The use of JavaScript

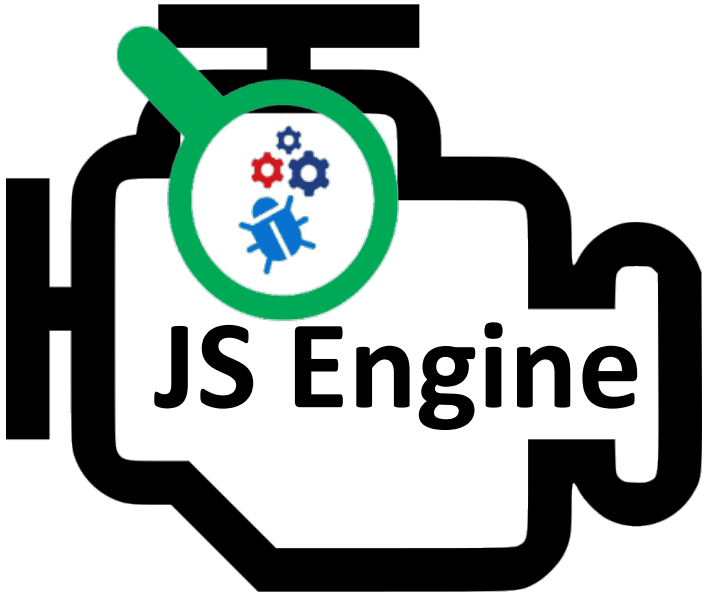
https://d2h0cx97tjks2p.cloudfront.net/blogs/wp-content/uploads/sites/2/2019/02/JavaScript-Applications.jpg


node
JS
PDF
Foxit
JS Engine
PDF
Adobe

Vulnerabilities in JS Engines

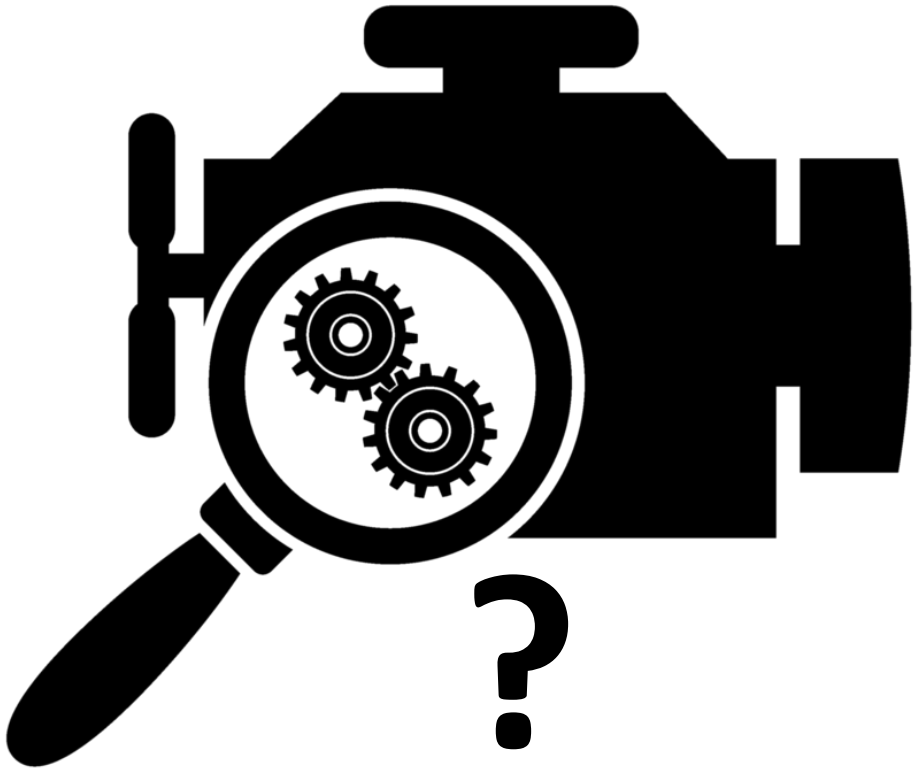
- Discovering vulnerabilities in JS Engines is a critical task
  → Various studies have been conducted !

  e.g., CodeAlchemist (NDSS `19), DIE (Oakland `20), Montage (SEC `20)



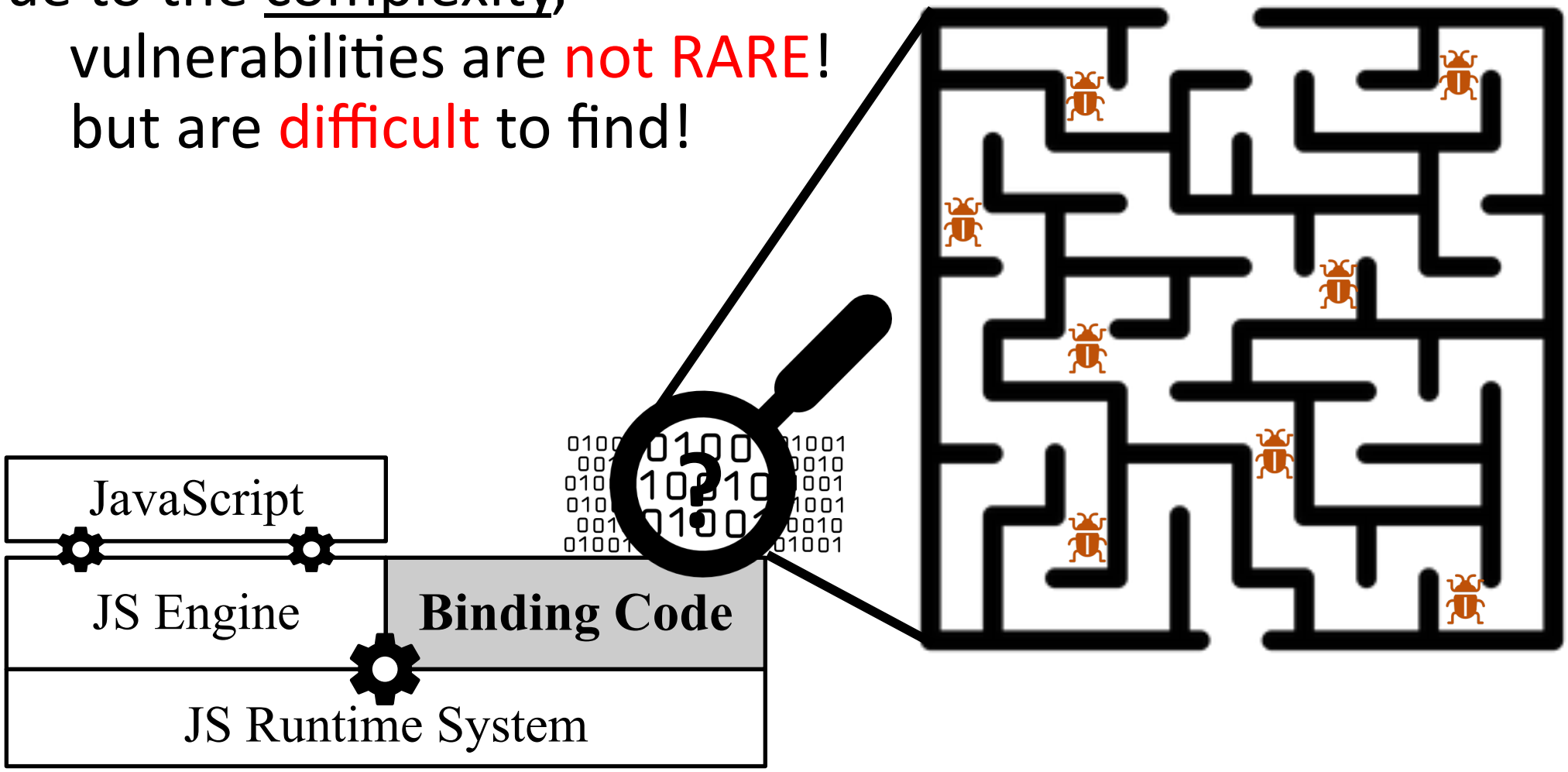
Binding Code

- Binding Code:
  Native components in JS engines for extending functionalities of JS



Vulnerabilities in Binding Code

- Due to the complexity,
  vulnerabilities are not RARE!
  but are difficult to find!



Fuzzing Binding Code

- A couple of fuzzers for fuzzing DOM objects in browsers
  - DOMFuzz 

  - Domato 🍅
    (https: //github.com/googleprojectzero/domato)
    It relies on manual development of a grammar and
    lacks the ability to generate semantically correct test cases

For Fuzzing Binding Code

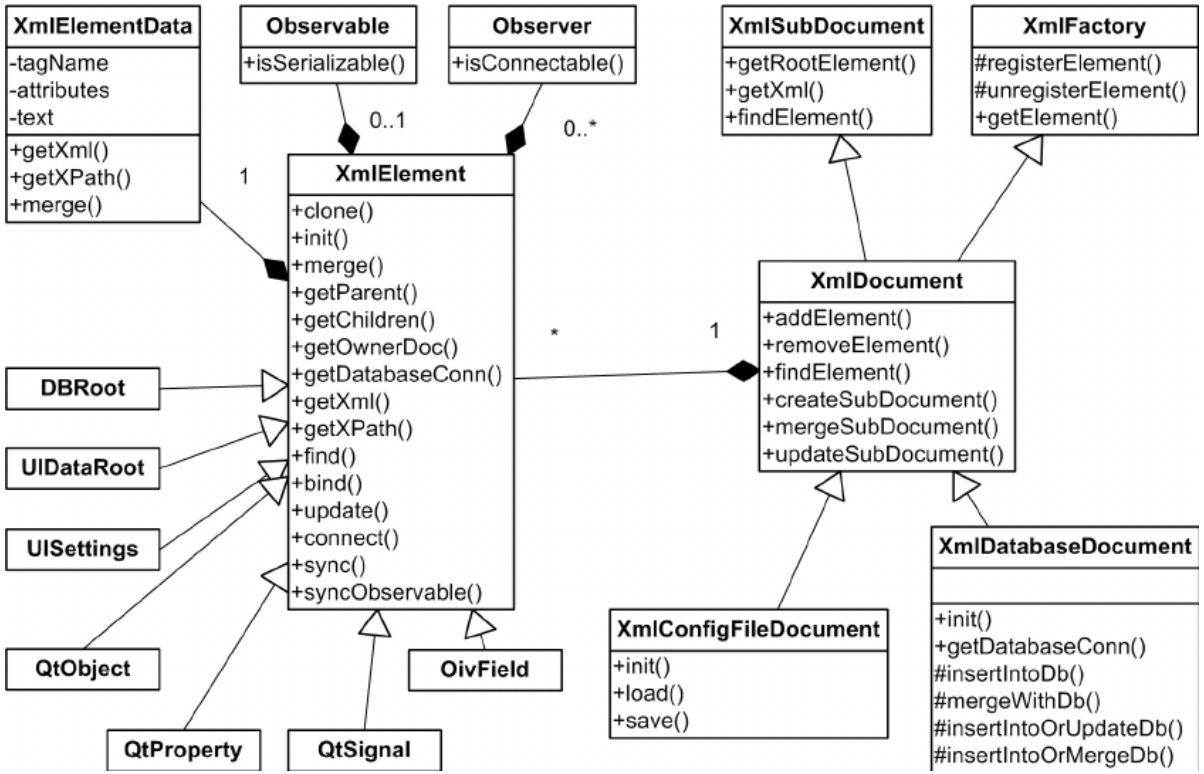
- Need to generate *syntactically* and *semantically* correct test cases

Test Case Example

```
1 var cb = this.getField("CheckBox");
2 cb.checkThisBox(0, true);
```

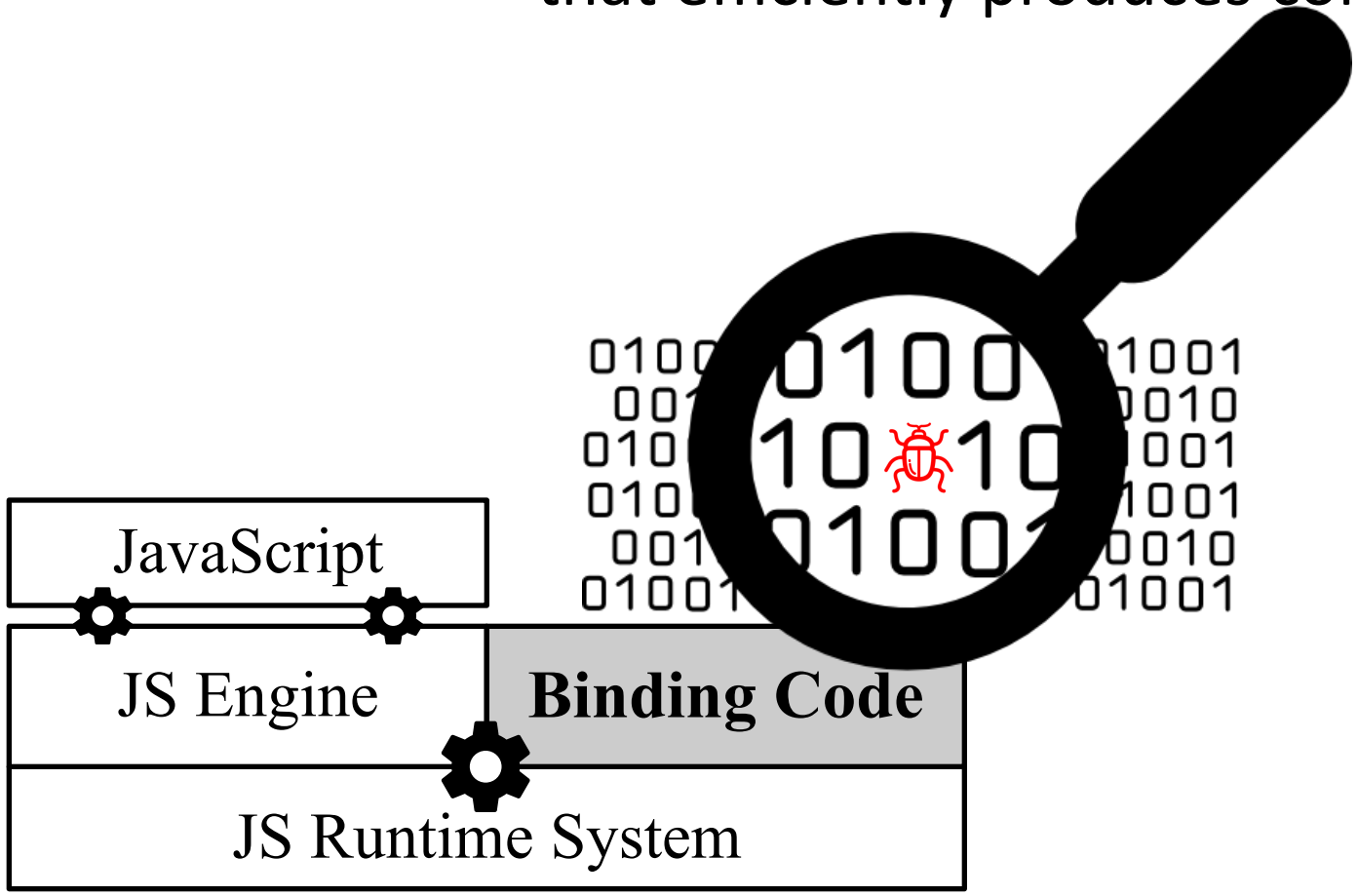
For Fuzzing Binding Code

- Need to generate *syntactically* and *semantically* correct test cases
- Need to deal with the huge search space

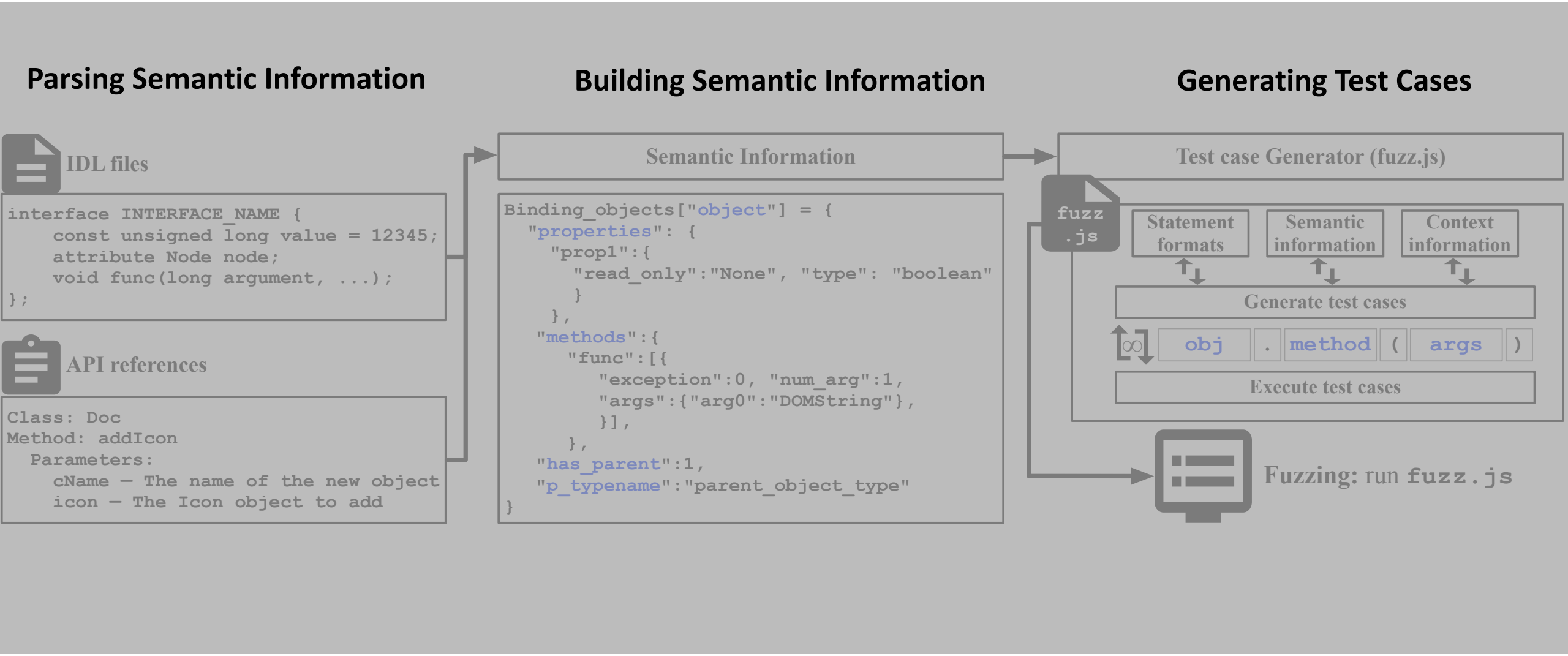


Our Goal

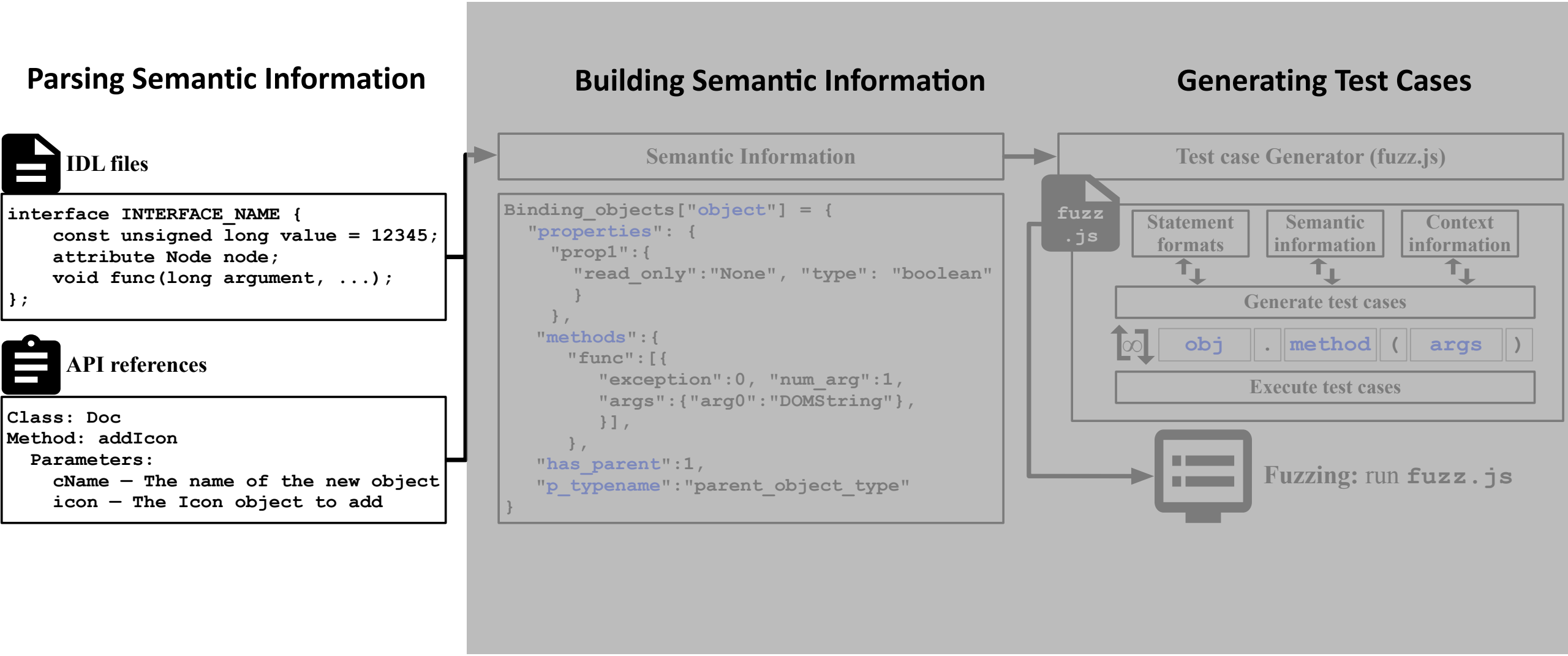
- Aim to design a general binding code fuzzer
  that efficiently produces correct test cases



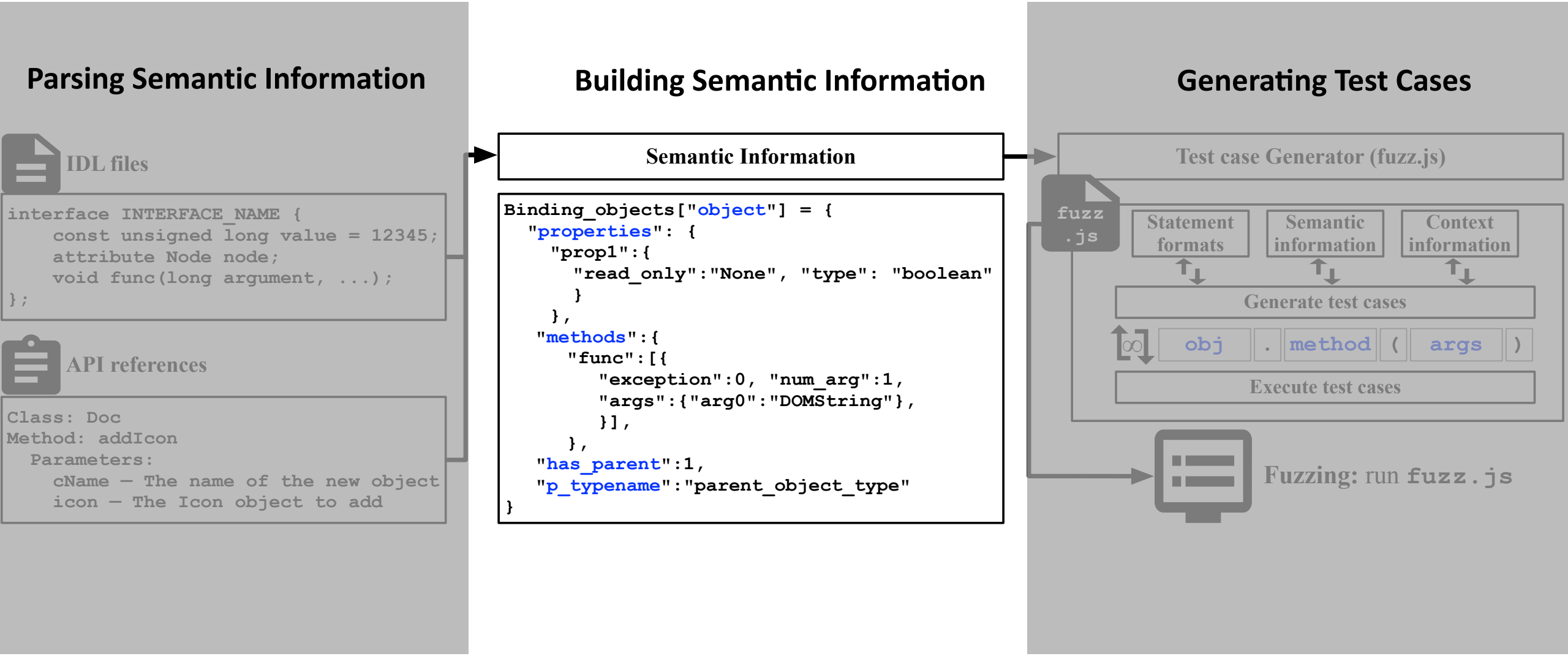
Overview of Favocado



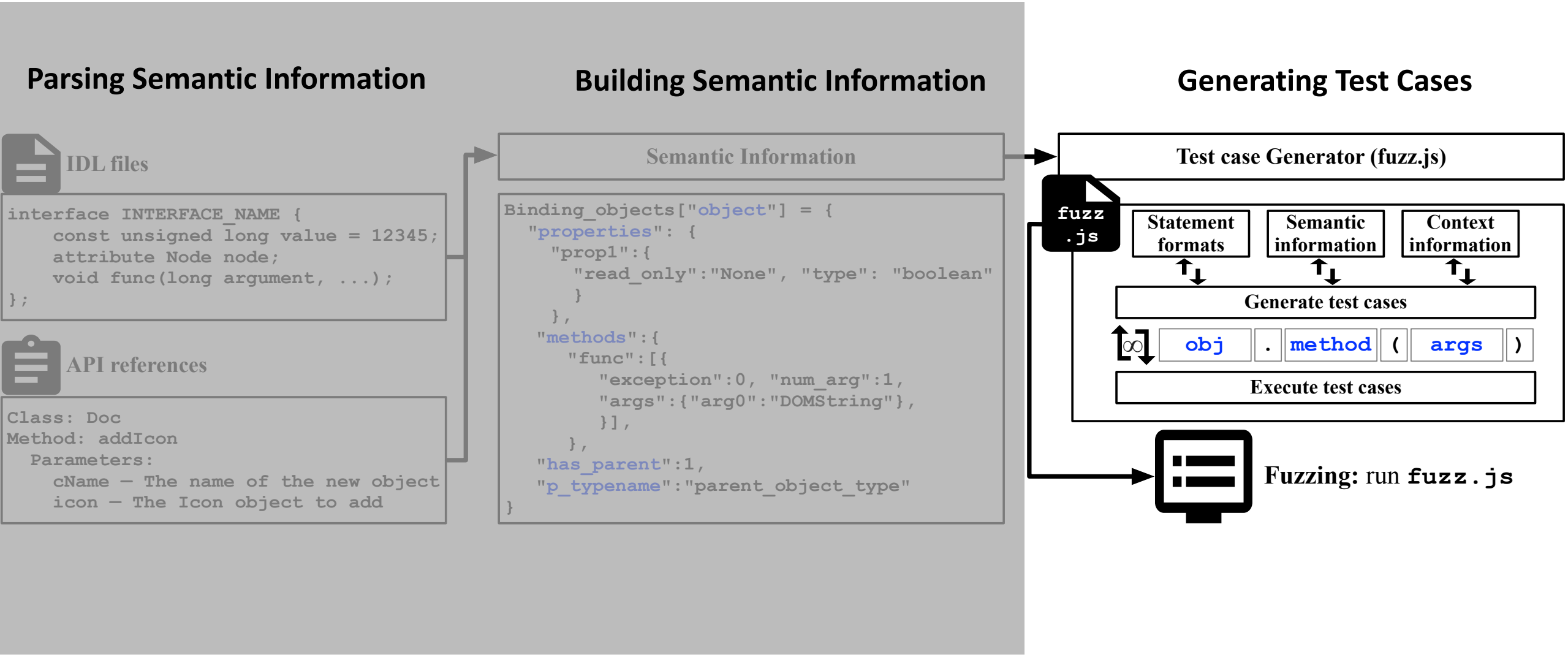
Overview of Favocado



Overview of Favocado



Overview of Favocado



Semantic Information Construction

- Favocado parses
  - Names of binding objects
  - Methods (and arguments) of binding objects
  - Properties of binding objects

  and type information of them

Semantic

- Favocado pars
  - Names of bi
  - Methods of
  - Properties o

```
Binding_objects["HTMLDialogElement"] = {
  "properties":
  {
    "open":
    {
      "read_only":"None", "type":"boolean"
    },
    "returnValue":
    {
      "read_only":"None", "type":"DOMString"
    }
  },
  "methods":
  {
    "close":
    {
      "exception":0, "numarg":1,
      "args":{"arg0":"DOMString"},
    },
    "showModal":
    {
      "exception":1, "numarg":0,
      "args":{},
    },
    "show":
    {
      "exception":0, "numarg":0,
      "args":{},
    }
  },
  "has_parent":1,
  "p_typename":"HTMLElement"
}
```

Semantic Information Construction

- Favocado parses …
  - Names of binding objects
  - Methods of binding objects
  - Properties of binding objects

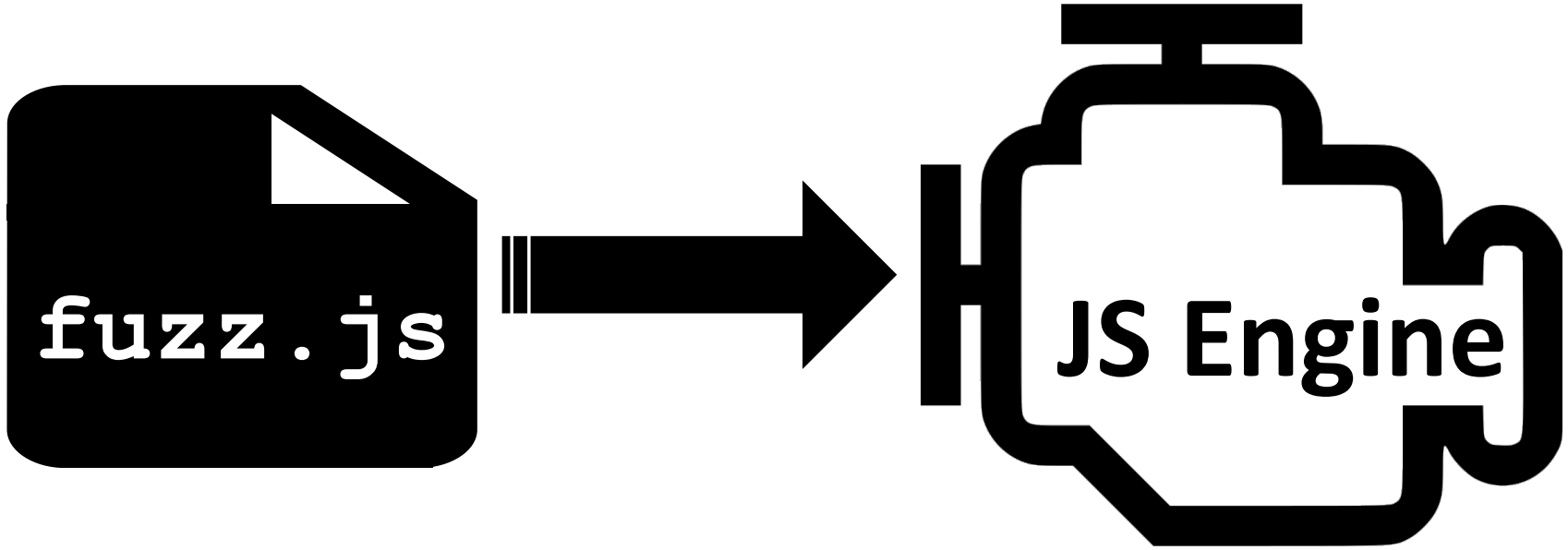
  and type information of them

- Also, Favocado analyzes …
  - Related binding objects

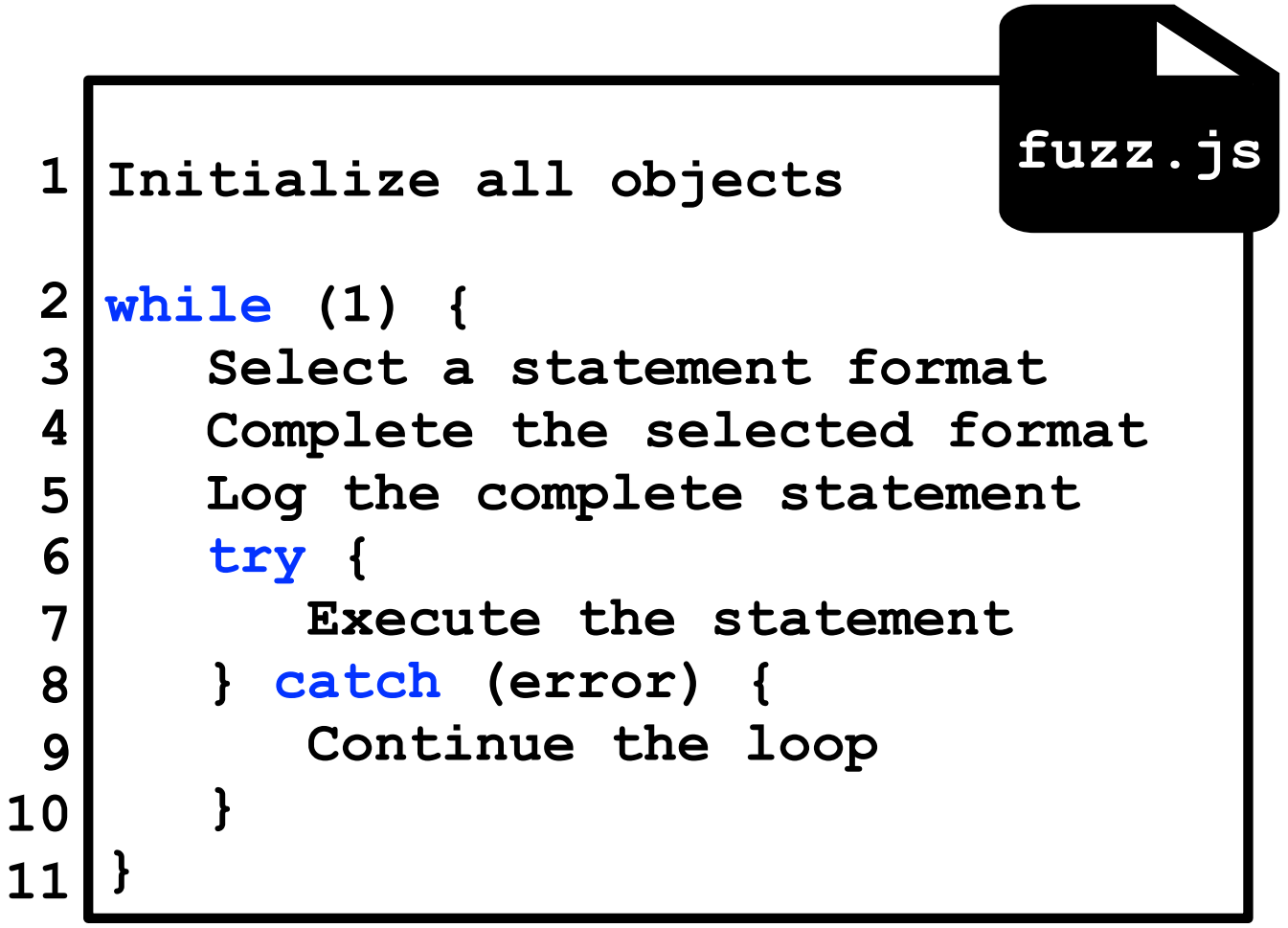
```
1 "ImageCapture":
2 [{
3   "Blob", "ImageBitmap", "MediaStreamTrack", "
        PhotoCapabilities"
4 }]
```

Dynamic Test Case Generator and Fuzzing

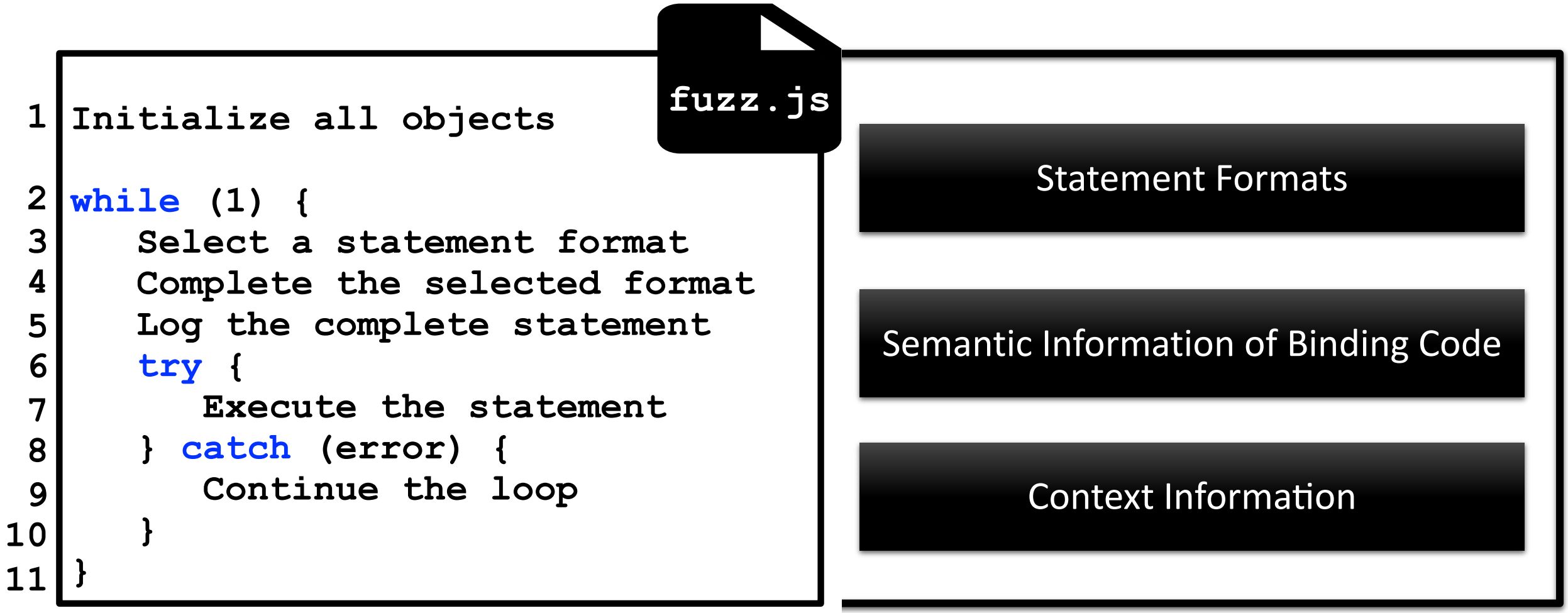
1. Selecting binding objects
2. Generating a test case generator (fuzz.js)
   with semantic information of the selected binding objects
3. Executing the text case generator



Test Case Generator



Test Case Generator



Statement Formats

Statement formats

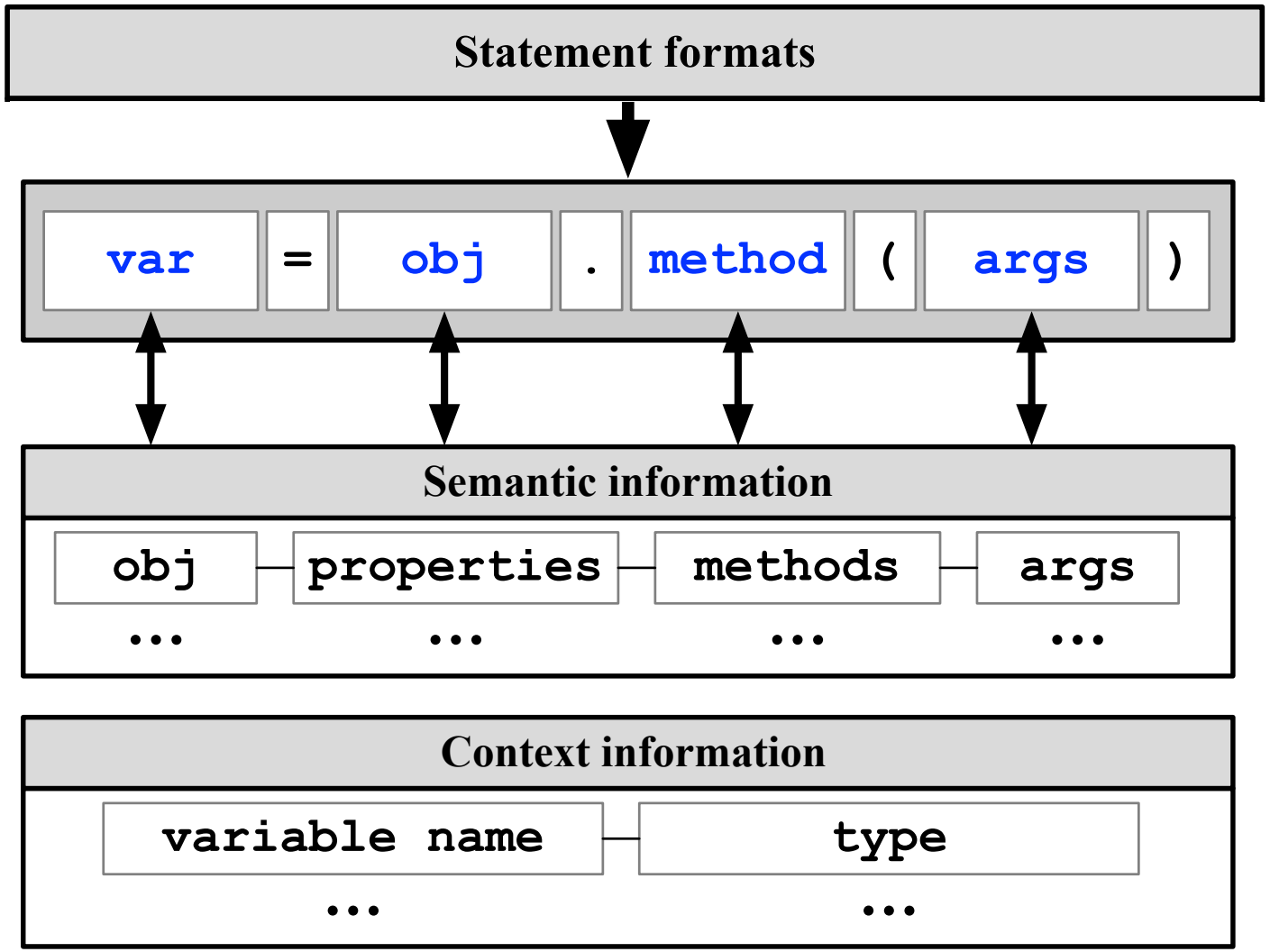
```
1  var obj = new obj(args)
2  obj.prop = value
3  var variable = obj.method_with_return(args)
4  obj.method_without_return(args)
5  for(var i=1; i++; i<n) { statements }
6  array[index] = value
7  obj.__proto__ = obj;
8  obj.__defineSetter__(prop, func)
9  obj.__defineGetter__(prop, func)
10 obj.prototype.method()
11 function(args) { statements }
```

Statement Formats

Statement formats

```
1  var obj = new obj(args)
2  obj.prop = value
3  var variable = obj.method_with_return(args)
4  obj.method_without_return(args)
5  for(var i=1; i++; i<n) { statements }
6  array[index] = value
7  obj.__proto__ = obj;
8  obj.__defineSetter__(prop, func)
9  obj.__defineGetter__(prop, func)
10 obj.prototype.method()
11 function(args) { statements }
```

Generating JS statements for fuzzing



Evaluation

- Targeted JS Runtime Systems and Binding Objects

PDF

PDF

Mojo and DOM

DOM

- Counting distinct bugs
  - Used the most recent version of target systems, all bugs found by Favocado were previously unknown ones

    → We manually analyzed all crashes to prevent overcounting

Evaluation Result

Target System	Binding Object	# of VMs used	Fuzzing Duration	# of Bugs	# of Vulnerabilities
Adobe Acrobat Reader	PDF	8	2 weeks	45	24
Foxit Reader	PDF	8	3 days	3	3
Chromium	Mojo	8	1 week	2	1
Chromium	DOM	8	2 weeks	6	2
WebKit	DOM	8	4 days	5	3
			Total	**62**	**34**

- 2 cores and 4GB of memory for each VM (1 fuzzing process executes on each VM)
- 13 vulnerabilities have become CVE entries as of today

A Case Study

- Use-after-free (CVE-2019-8211, Adobe Acrobat Reader v2019.012.20035)

Minimized JavaScript snippet

```
1  X.toString = function(){
2  this.flattenPages(0); //Deallocate the textfield object
3  return "center";
4  }
5
6  textfield = this.addField("Field", "text", 0, [0,0,800,800]);
7  textfield.alignment = X //Use-after-free occurs!
```

Comparison with Domato

- Run Favocado and Domato for Adobe Acrobat Reader v2020.009.20067
  - Manually developed a grammar file for Domato
  - Used 8 VMs for 1 week
    - Domato found 1 bug
    - Favocado found 6 distinct bugs including the one discovered by Domato

- The error rate of test cases
  - Domato – around 34% of test cases caused runtime errors
  - Favocado – around 9% of test cases caused runtime errors
    - Note: It randomly triggers runtime errors by intentionally using different types of objects, values out of range, etc. for finding vulnerabilities.

Conclusion

- Proposed Favocado, a JavaScript binding code fuzzer, that can generate semantically correct test cases.
- Demonstrated the importance of semantically correct test cases and the effectiveness of Favocado.
- We open the source code !
  - https://github.com/favocado/Favocado


Thanks!

Limitations

- Implementation detail to enable fully automated fuzzing
- Feedback-driven fuzzing
- Minimizing test cases for analyzing crashes
- Fuzzing binding code in other scripting languages