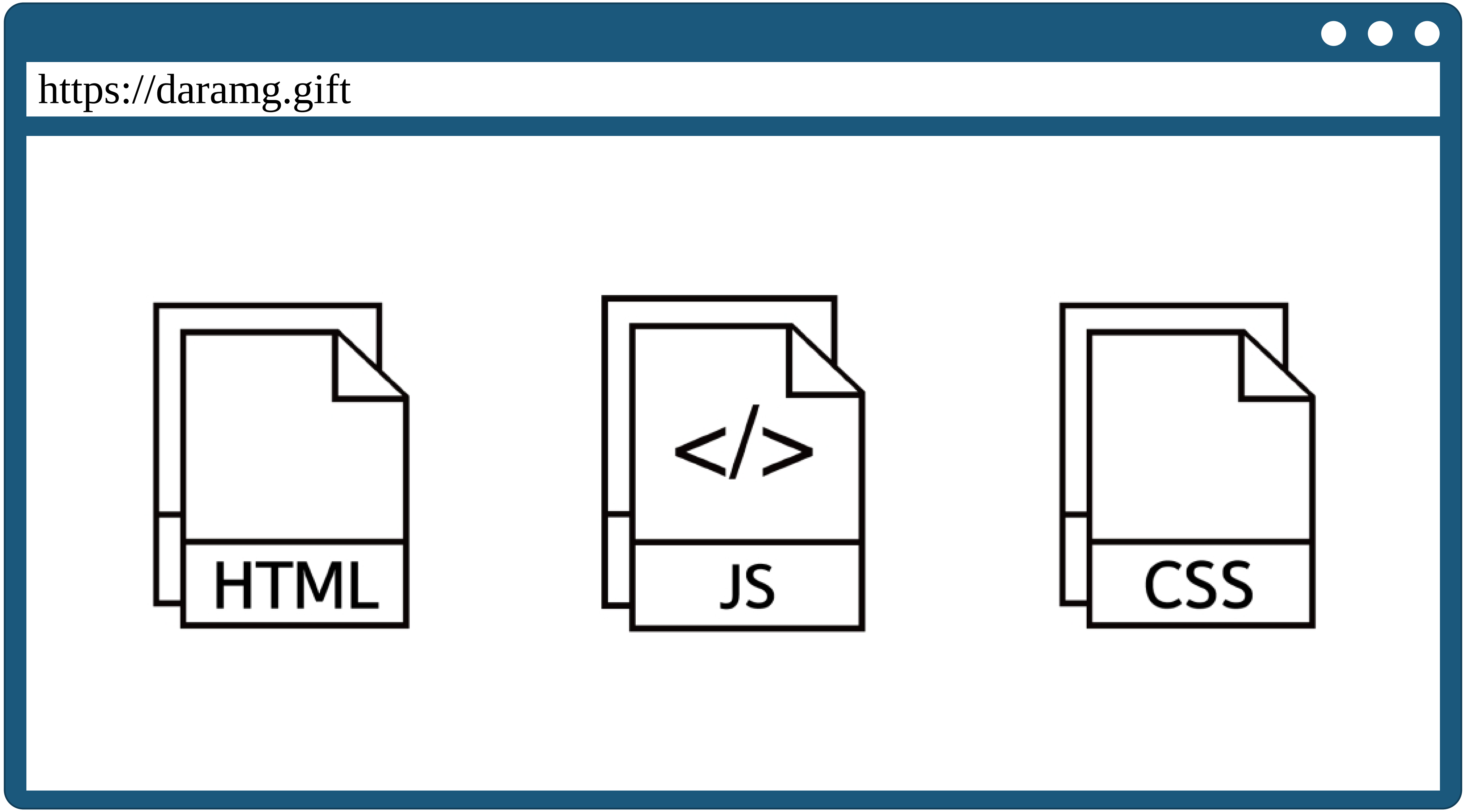
CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines

HyungSeok Han, DongHyeon Oh, Sang Kil Cha

KAIST

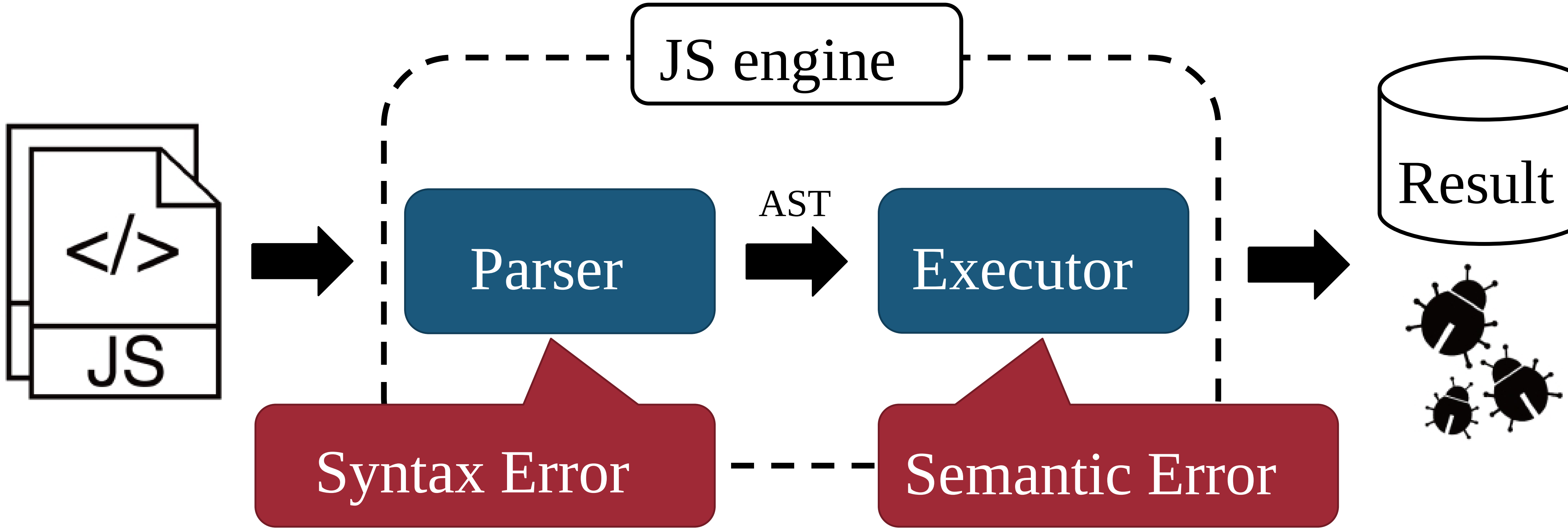

https://daramg.gift
HTML
</>
JS
CSS

How to Find JS Bugs?

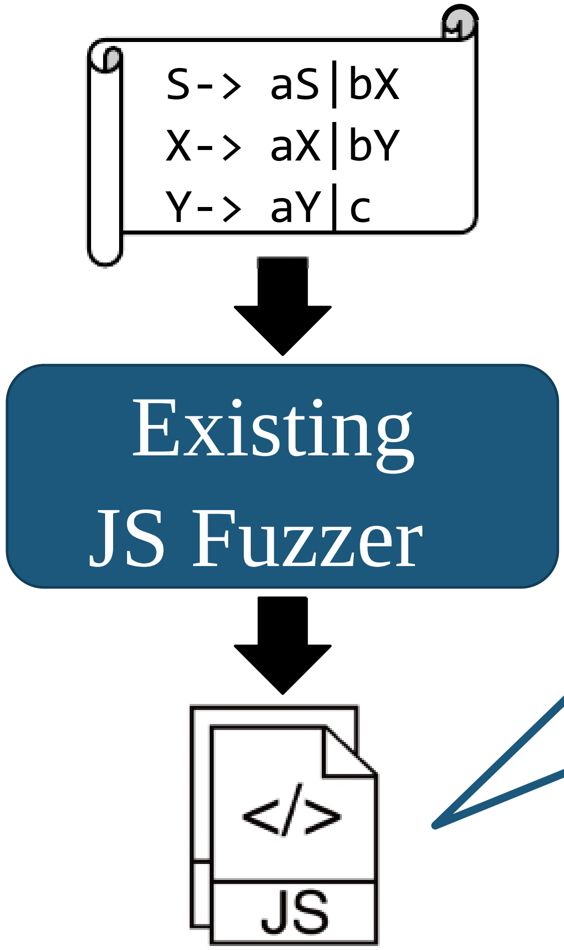
1. Analyzing JS Engine Code

2. Fuzzing

Structure of JS Engine



Semantics-Unawareness



```
// ReferenceError
var arr = new Array (n);

// TypeError
var num = 10; num ();
```

Previous Work: Grammar-based Fuzzer

- jsfunfuzz
  - A state-of-the-art **generation-based** fuzzer developed by ***Mozilla***
  - Found **2,800** bugs since 2006

- LangFuzz
  - A state-of-the-art **mutation-based** fuzzer appeared at ***USENIX'12***
  - A parent of IFuzzer and TreeFuzz
  - Found **2,300** bugs since 2011

Semantics-Unawareness of jsfunfuzz

jsfunfuzz

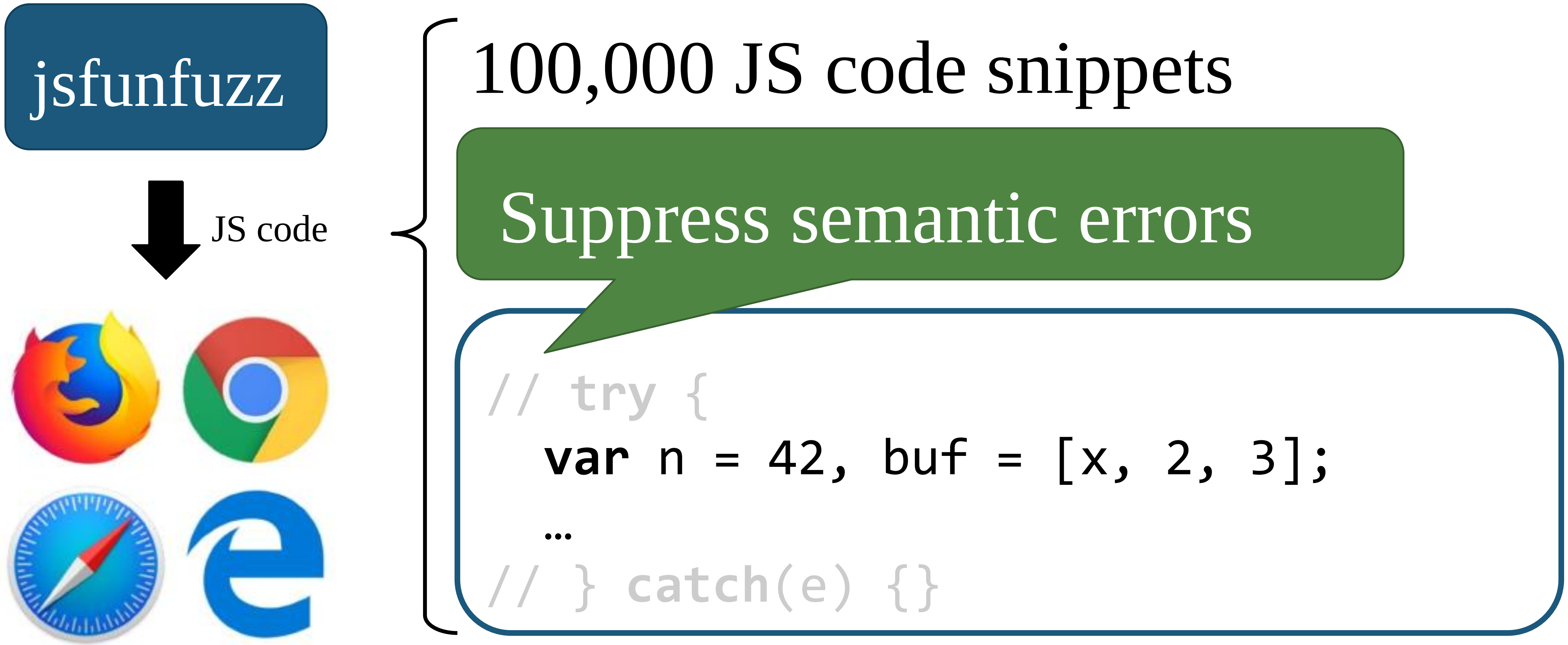
JS code

100,000 JS code snippets

Suppress semantic errors

```
try {
  var n = 42, buf = [x, 2, 3];
  …
} catch(e) {}
```

Semantics-Unawareness of jsfunfuzz



```
// try {
  var n = 42, buf = [x, 2, 3];
  …
// } catch(e) {}
```

Previous Work: Grammar-based Fuzzer

- jsfunfuzz
  - A state-of-the-art **generation-based** fuzzer developed by ***Mozilla***
  - Found **2,800** bugs since 2006

- LangFuzz
  - A state-of-the-art **mutation-based** fuzzer appeared in ***USENIX'12***
  - A parent of IFuzzer and TreeFuzz
  - Found **2,300** bugs since 2011

Semantics-Unawareness of LangFuzz

```
var arr = new Array (0x100);
for(let i = 0; i < 0x100; i++){
  // i is only available here
  arr[i] = i;
}
```

Semantics-Unawareness of LangFuzz

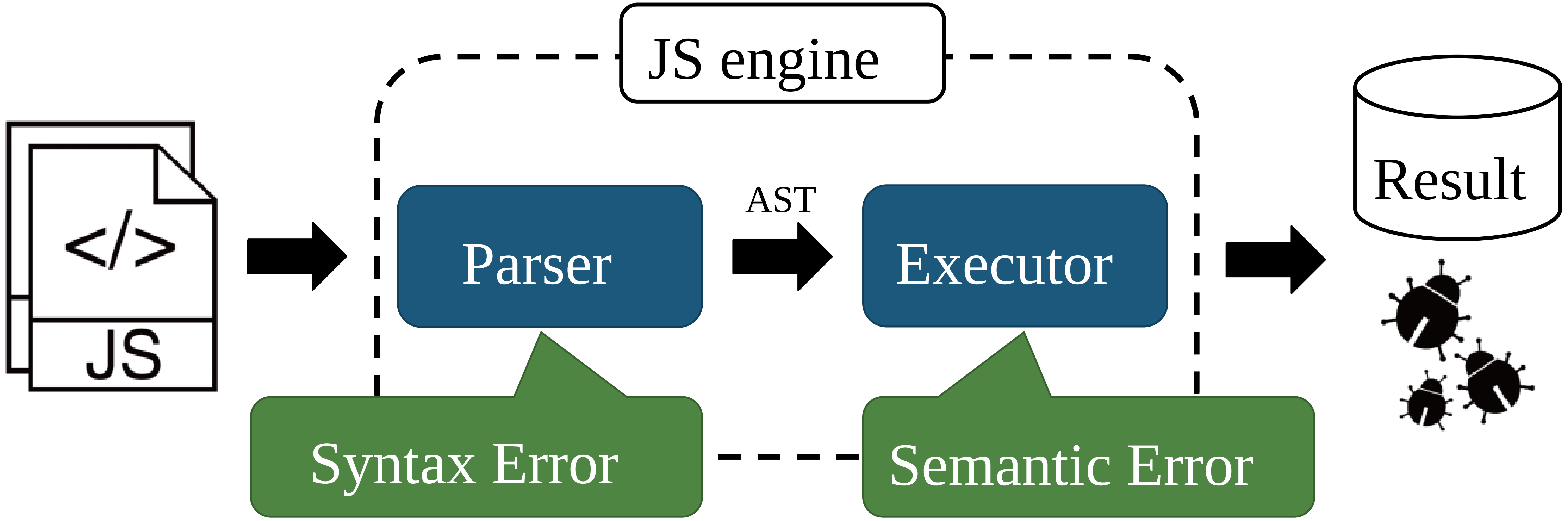
```
var x = new String (y);
for(let i = 0; i < 0x100; i++){
  // i is only available here
  arr[i] = i;
}
```

Semantics-Unawareness of LangFuzz

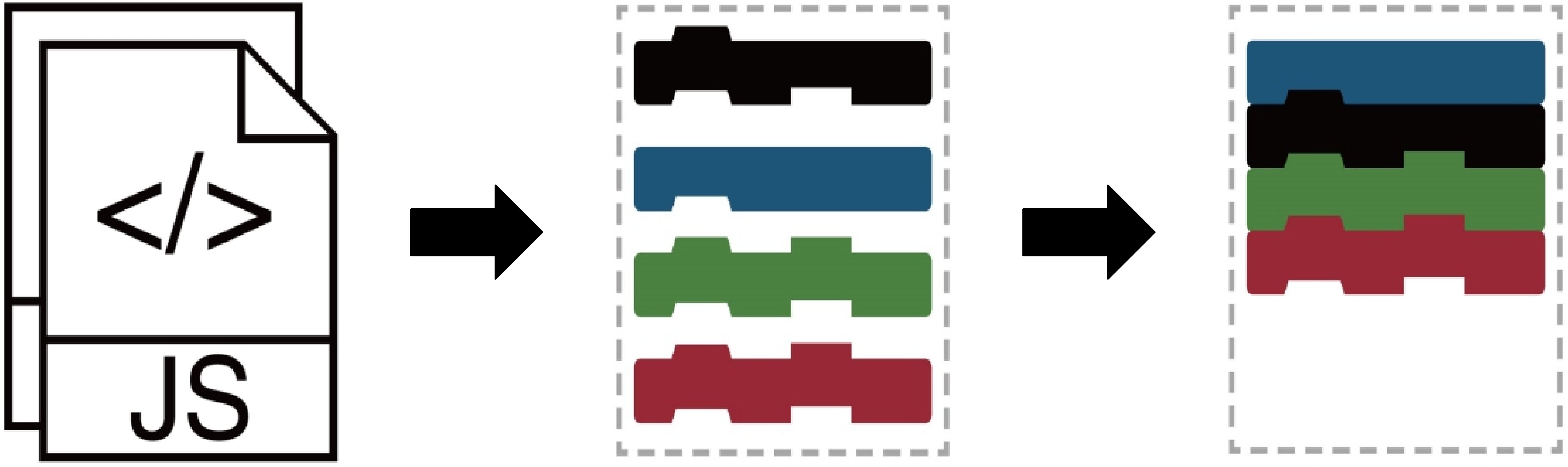
```
var arr = new String (i);
for(let i = 0; i < 0x100; i++){
  // i is only available here
  arr[i] = i;
}
```

ReferenceError: i is not DefineD

Our Goal: Be Semantics-Aware



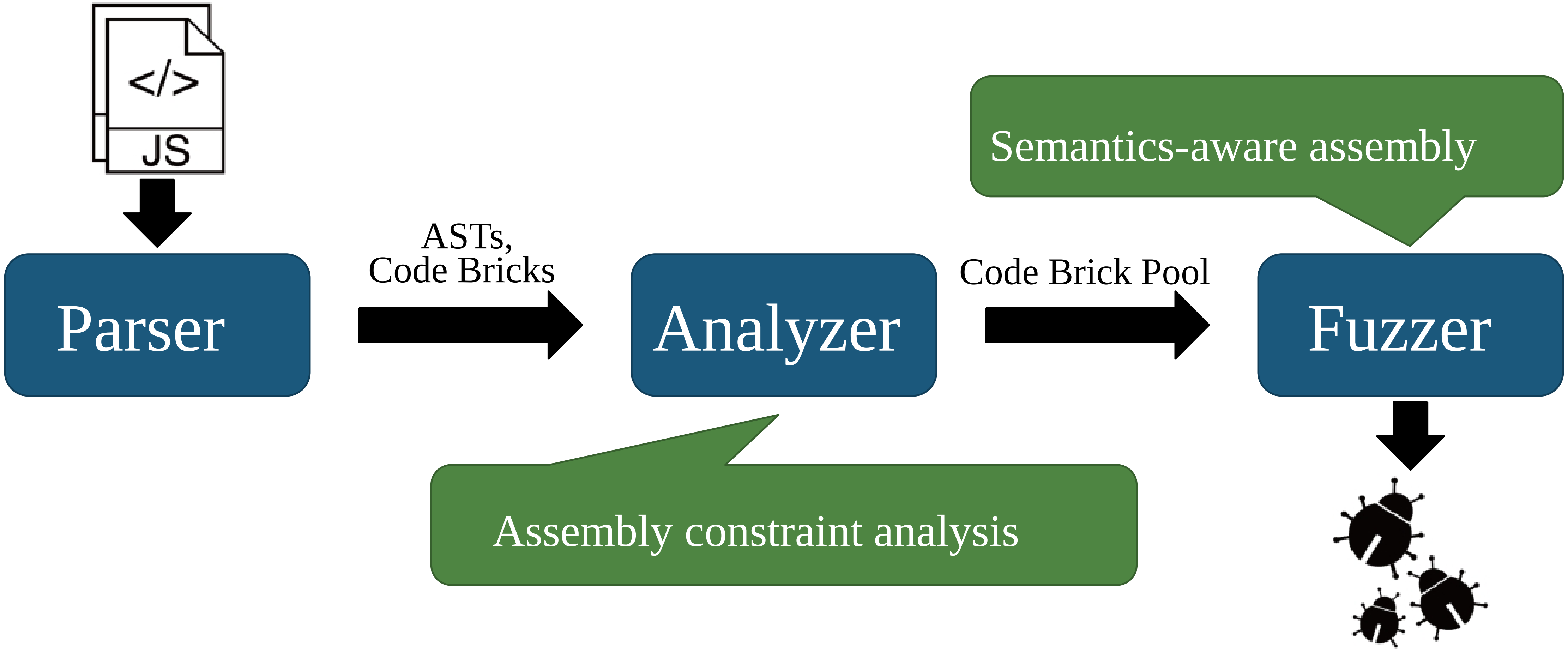
Intuition: Assemble Code Bricks by Assembly Constraints

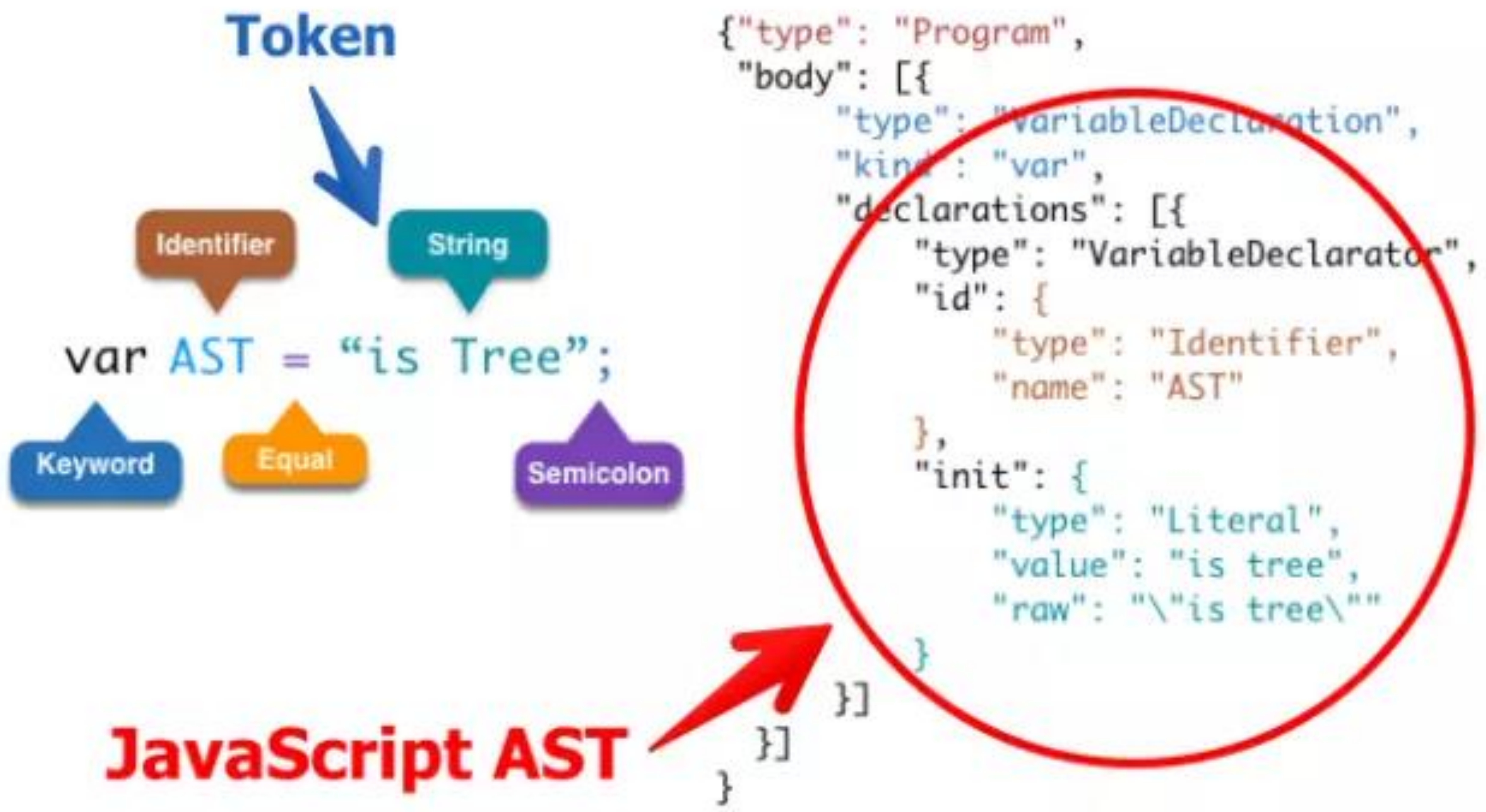


Code bricks
w/ constraints

Semantically
valid code

CodeAlchemist: Semantics-Aware Code Generation for Fuzzing

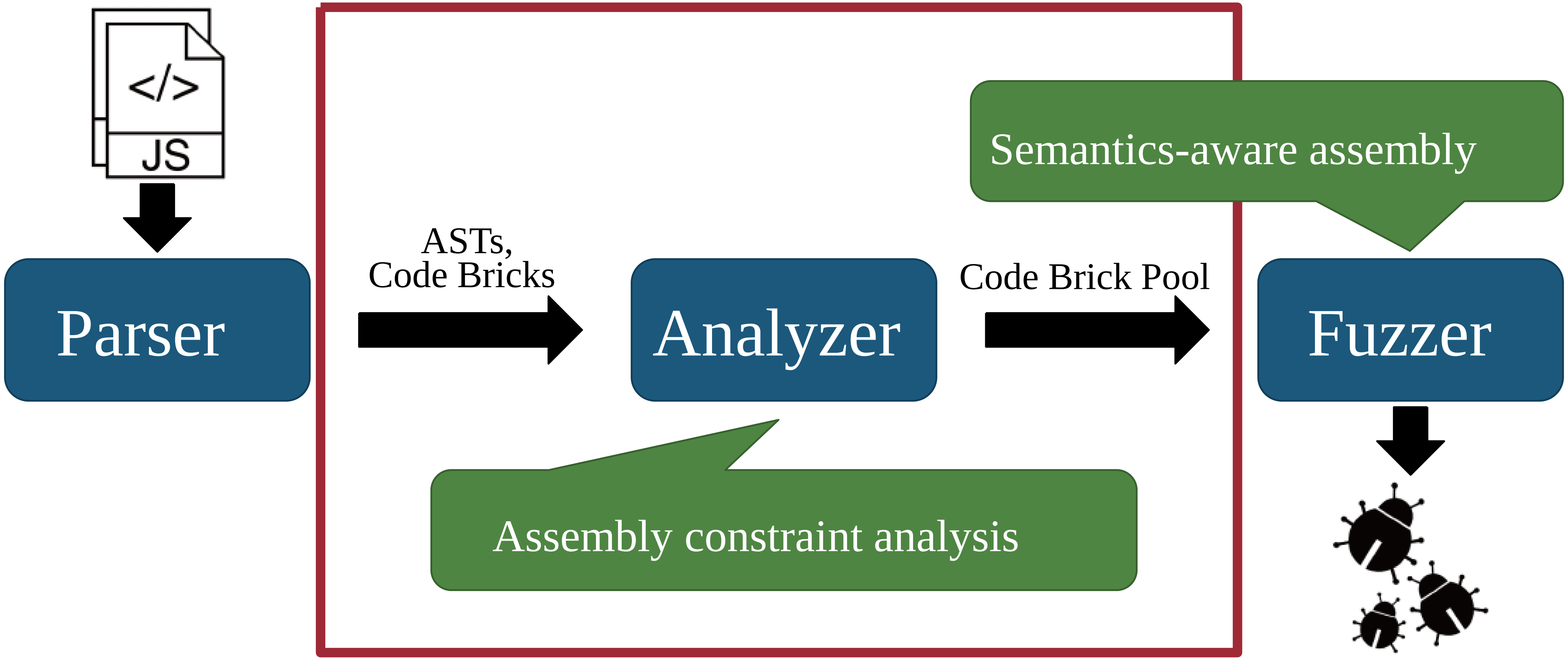




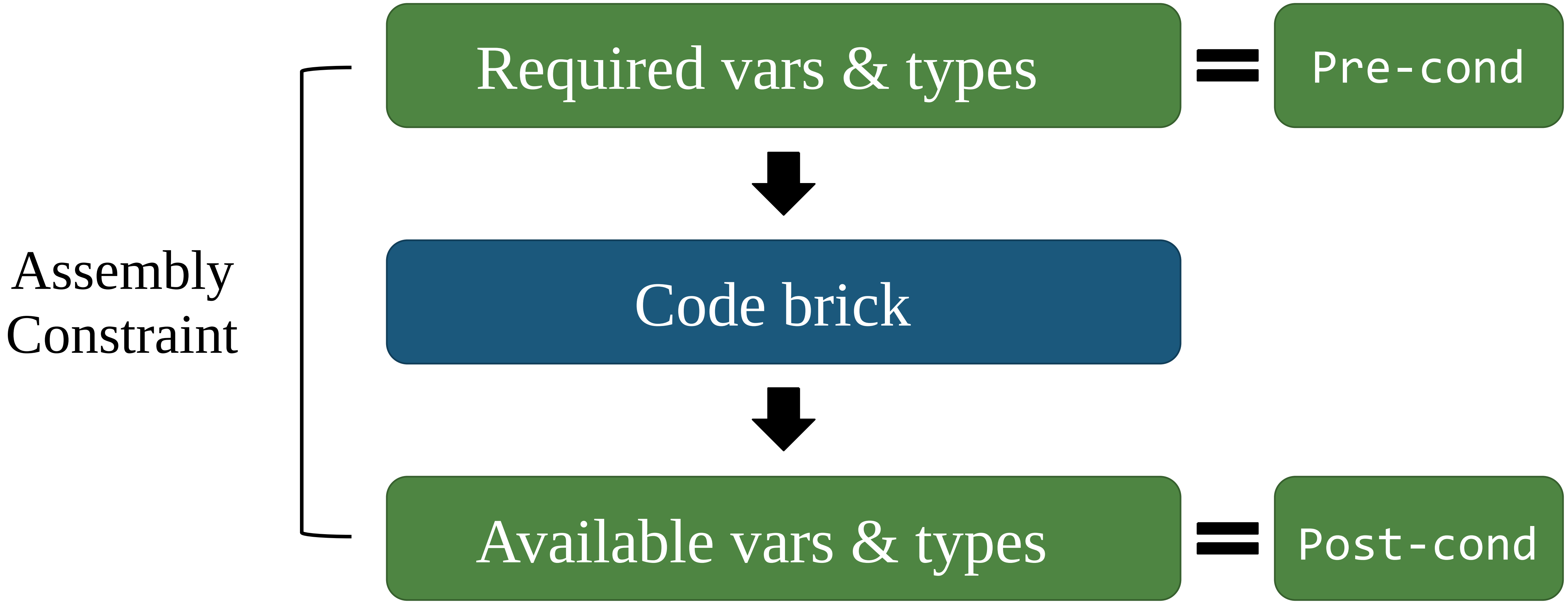
```
var AST = “is Tree”;
```

```
{"type": "Program",
 "body": [{
        "type": "VariableDeclaration",
        "kind": "var",
        "declarations": [{
             "type": "VariableDeclarator",
             "id": {
                  "type": "Identifier",
                  "name": "AST"
             },
             "init": {
                  "type": "Literal",
                  "value": "is tree",
                  "raw": "\"is tree\""
             }
        }]
 }]
}
```

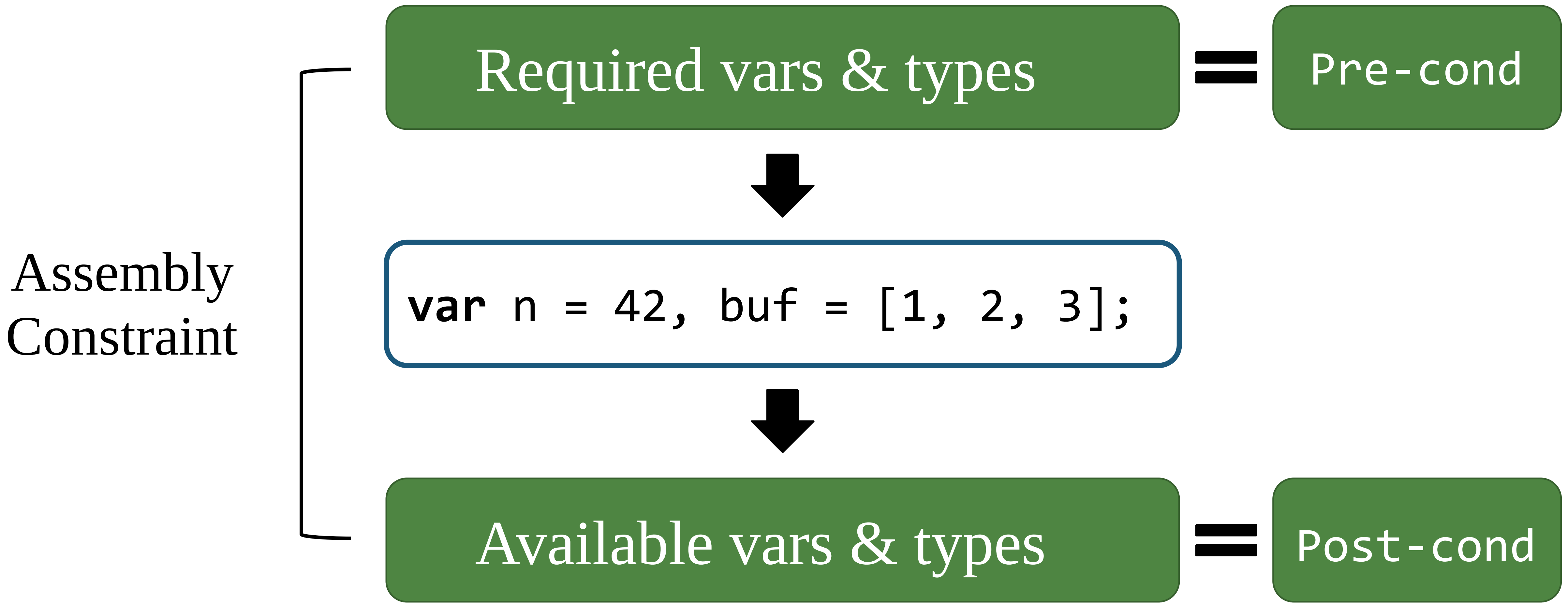
How to Analyze Assembly Constraints?



Assembly Constraint

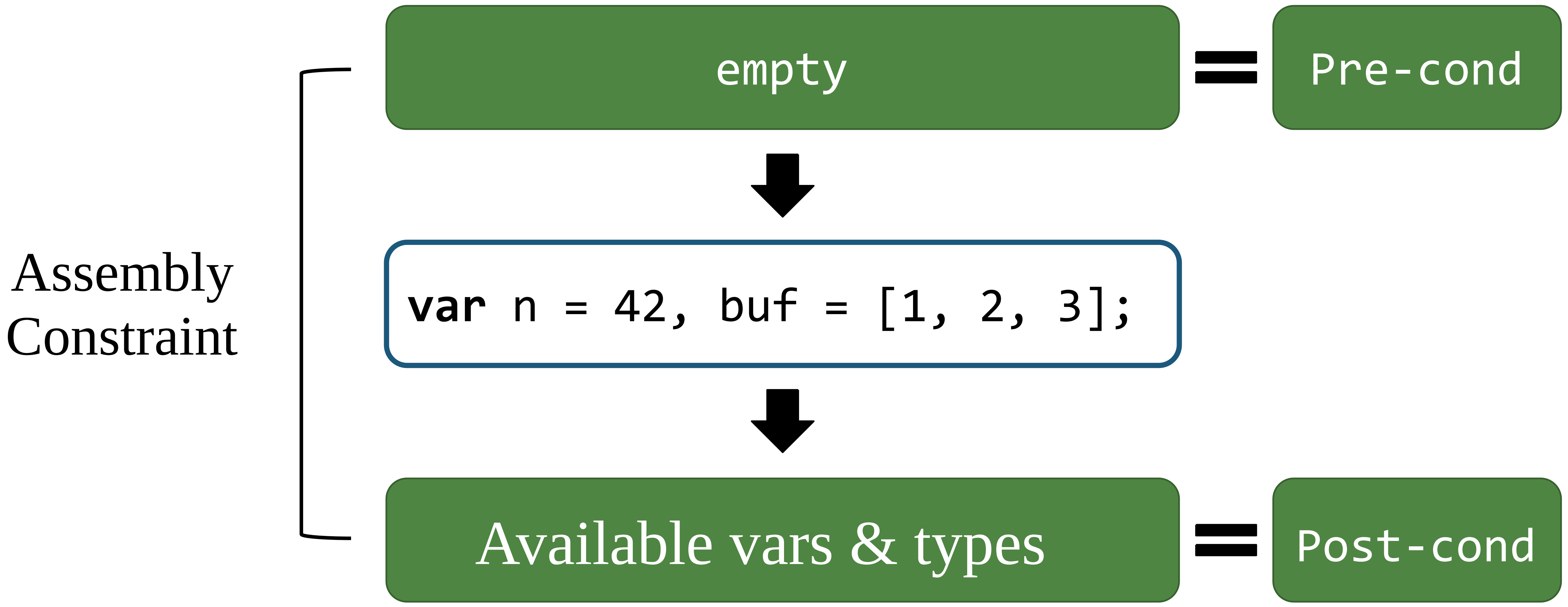


Assembly Constraint



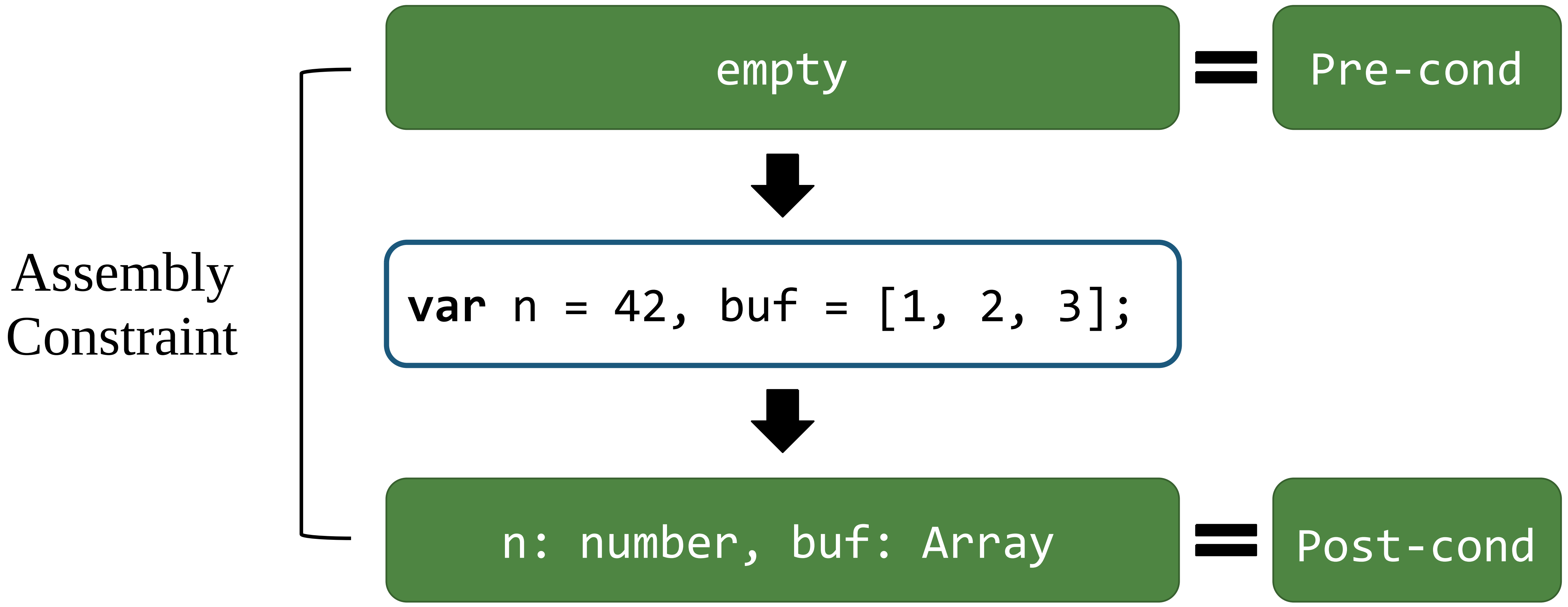
```
var n = 42, buf = [1, 2, 3];
```

Assembly Constraint



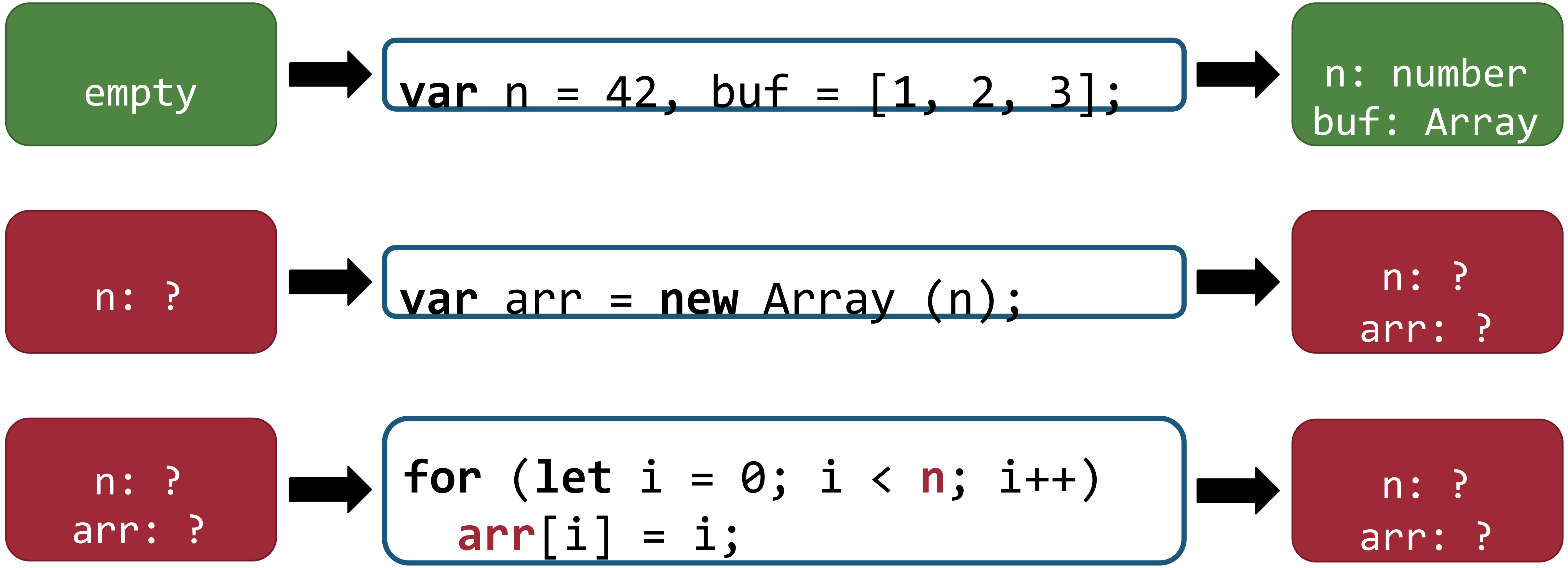
```
var n = 42, buf = [1, 2, 3];
```

Assembly Constraint



```
var n = 42, buf = [1, 2, 3];
```

Data-flow Analysis

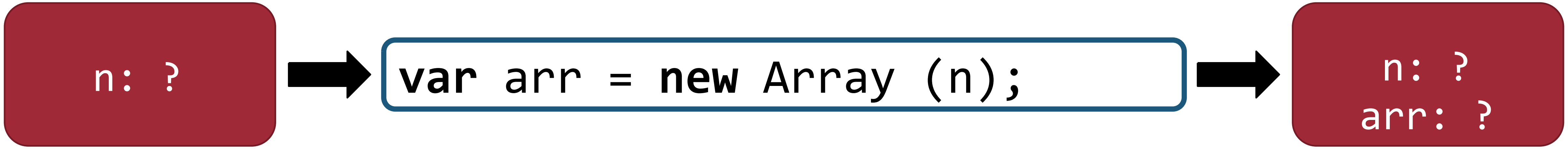


```
var n = 42, buf = [1, 2, 3];
```

```
var arr = new Array (n);
```

```
for (let i = 0; i < n; i++)
   arr[i] = i;
```

Dynamic Type Analysis



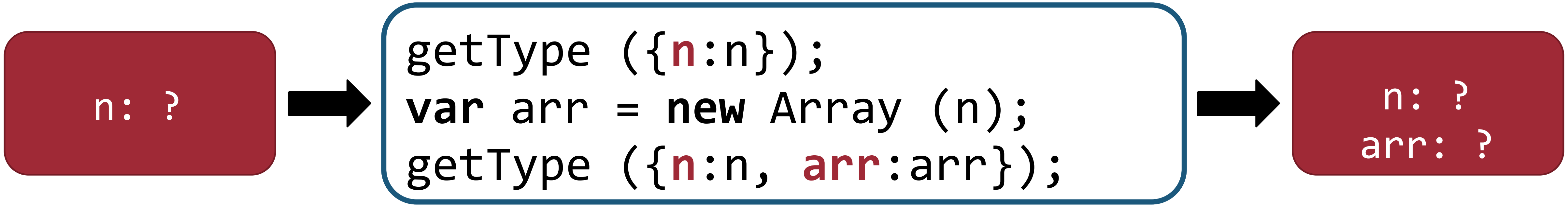
```
var arr = new Array (n);
```

JS Type CodeAlchemist handles

- Primitive types
  - Undefined, Null, String, Boolean, Symbol, Number, Object

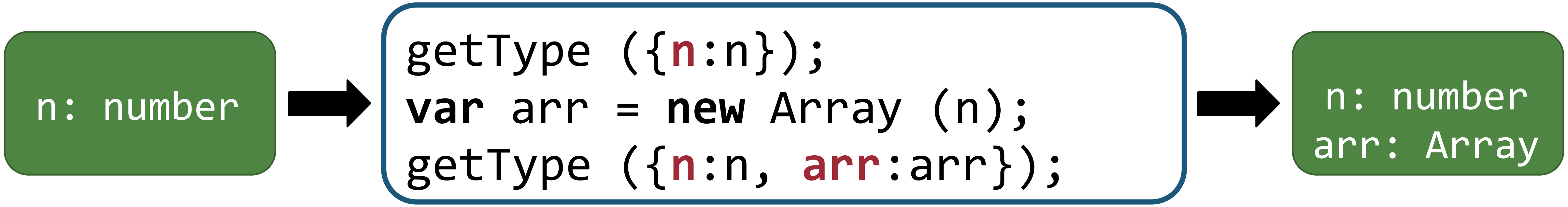
- Built-in types
  - Array, ArrayBuffer, Function, …
  - Depend on JS engine

Dynamic Type Analysis



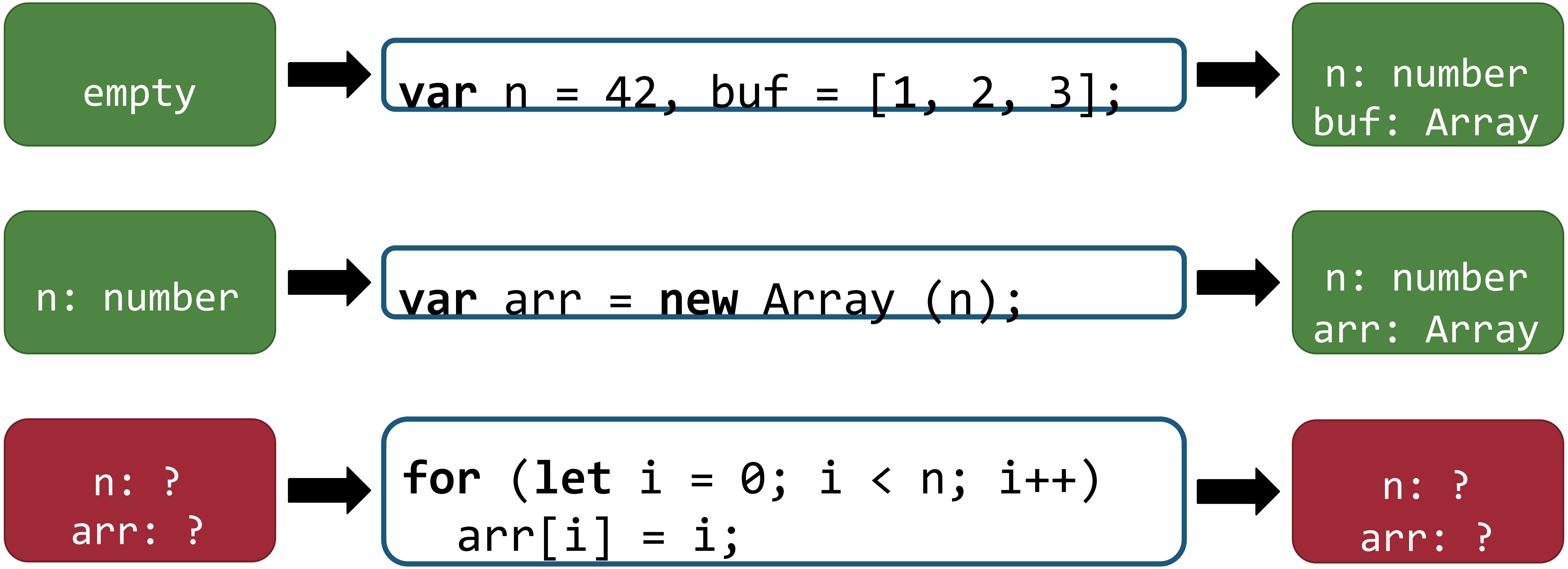
```
getType ({n:n});
var arr = new Array (n);
getType ({n:n, arr:arr});
```

Dynamic Type Analysis



```
getType ({n:n});
var arr = new Array (n);
getType ({n:n, arr:arr});
```

Dynamic Type Analysis

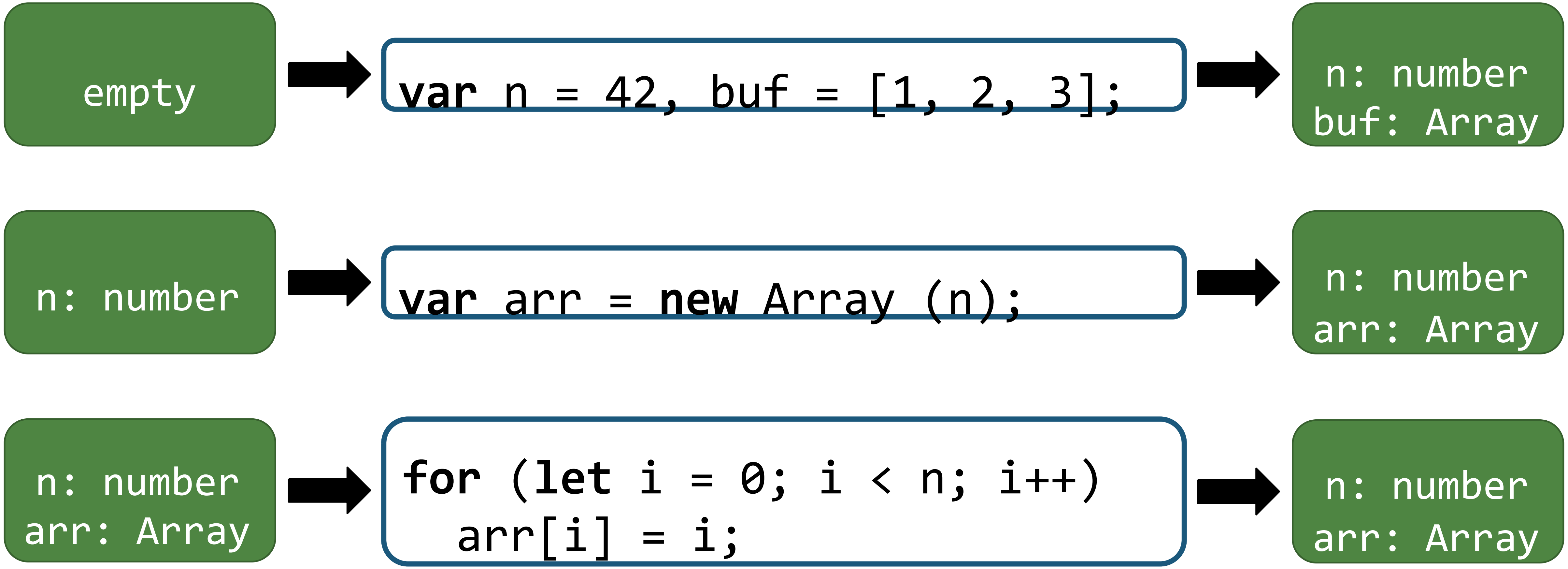


```
var n = 42, buf = [1, 2, 3];
```

```
var arr = new Array (n);
```

```
for (let i = 0; i < n; i++)
   arr[i] = i;
```

Assembly Constraint Analysis

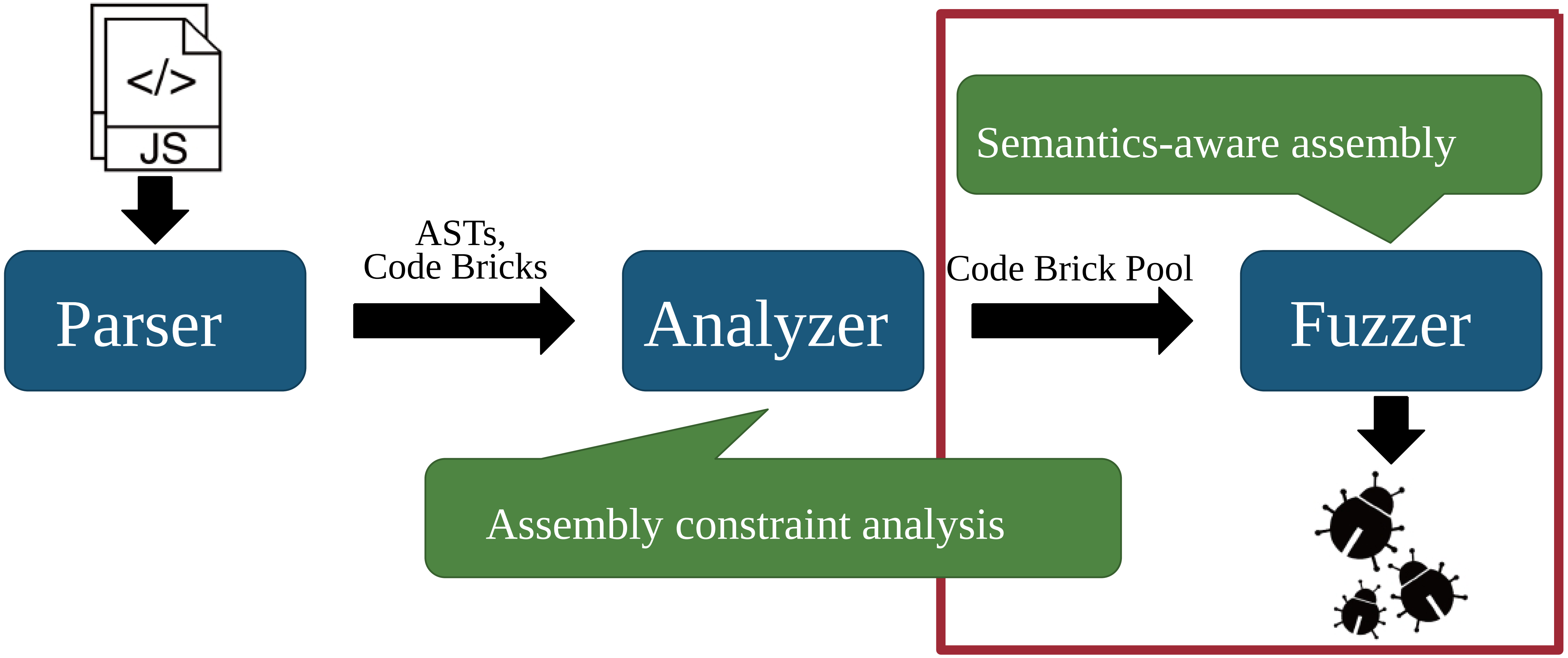


```
var n = 42, buf = [1, 2, 3];
```

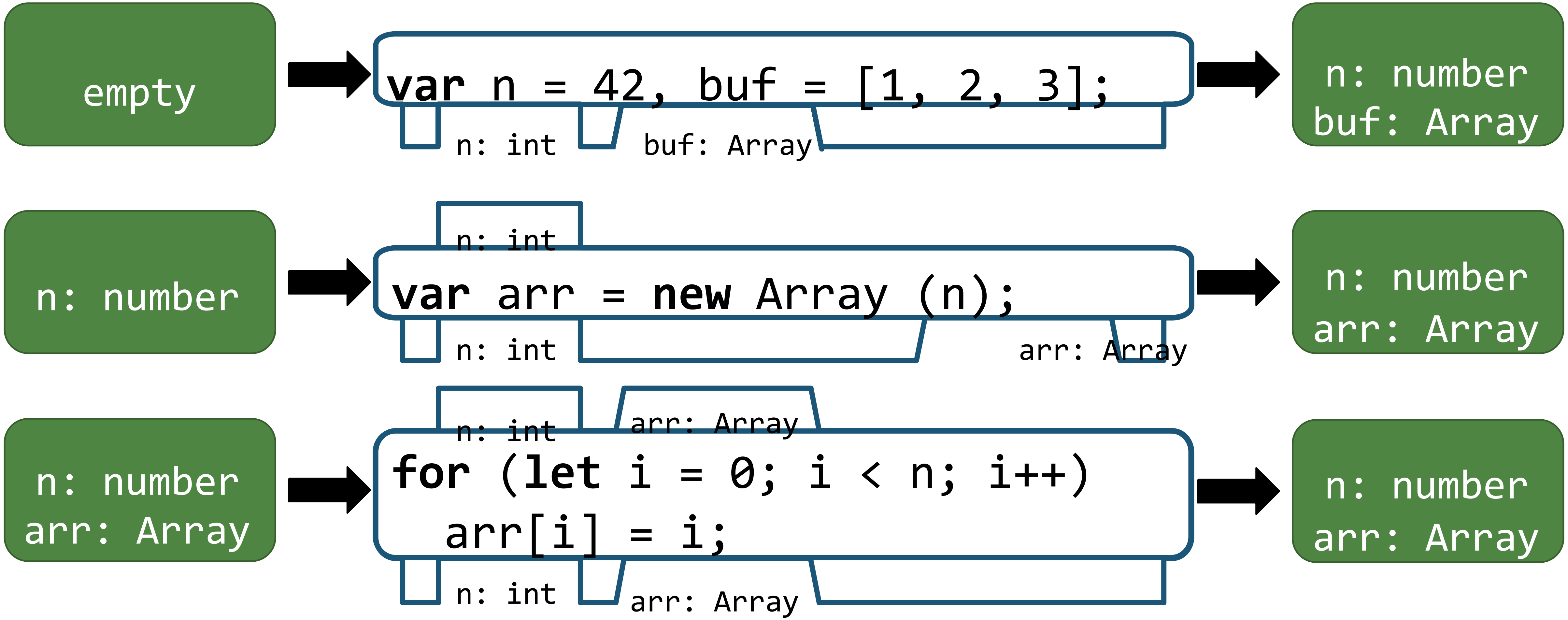
```
var arr = new Array (n);
```

```
for (let i = 0; i < n; i++)
   arr[i] = i;
```

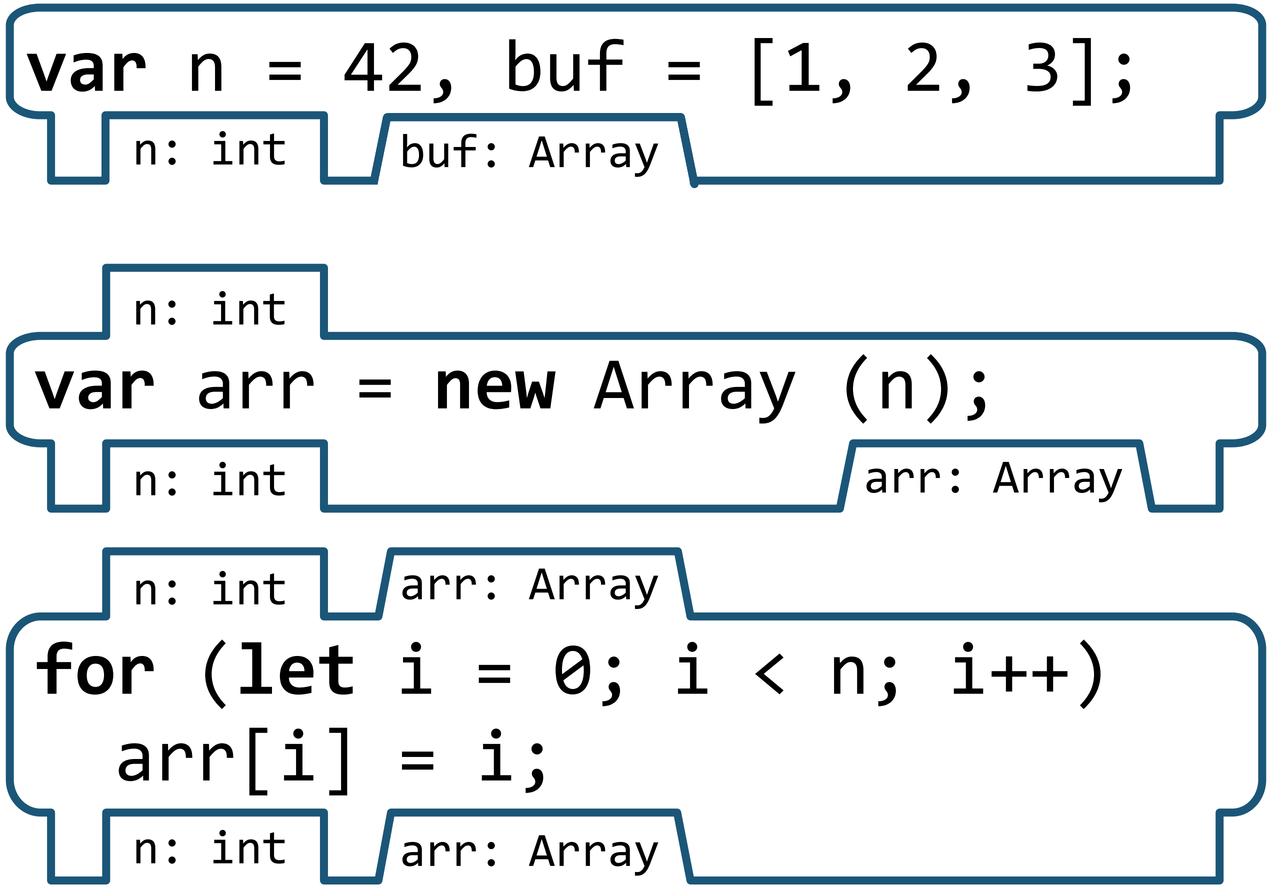
How to Assemble Code Bricks?



Code Bricks with Teeth & Holes



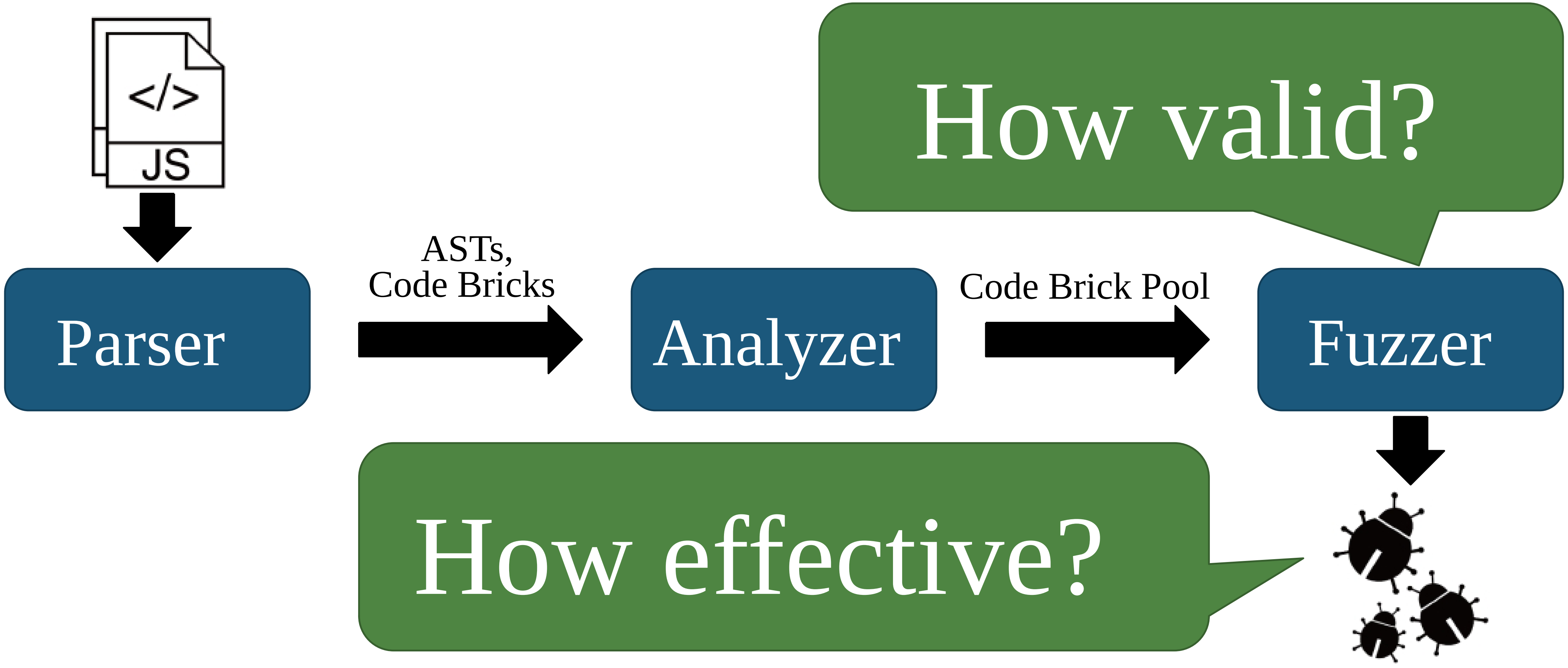
Semantics-Aware Assembly


```
var n = 42, buf = [1, 2, 3];
```

```
var arr = new Array (n);
```

```
for (let i = 0; i < n; i++)
  arr[i] = i;
```

Evaluation

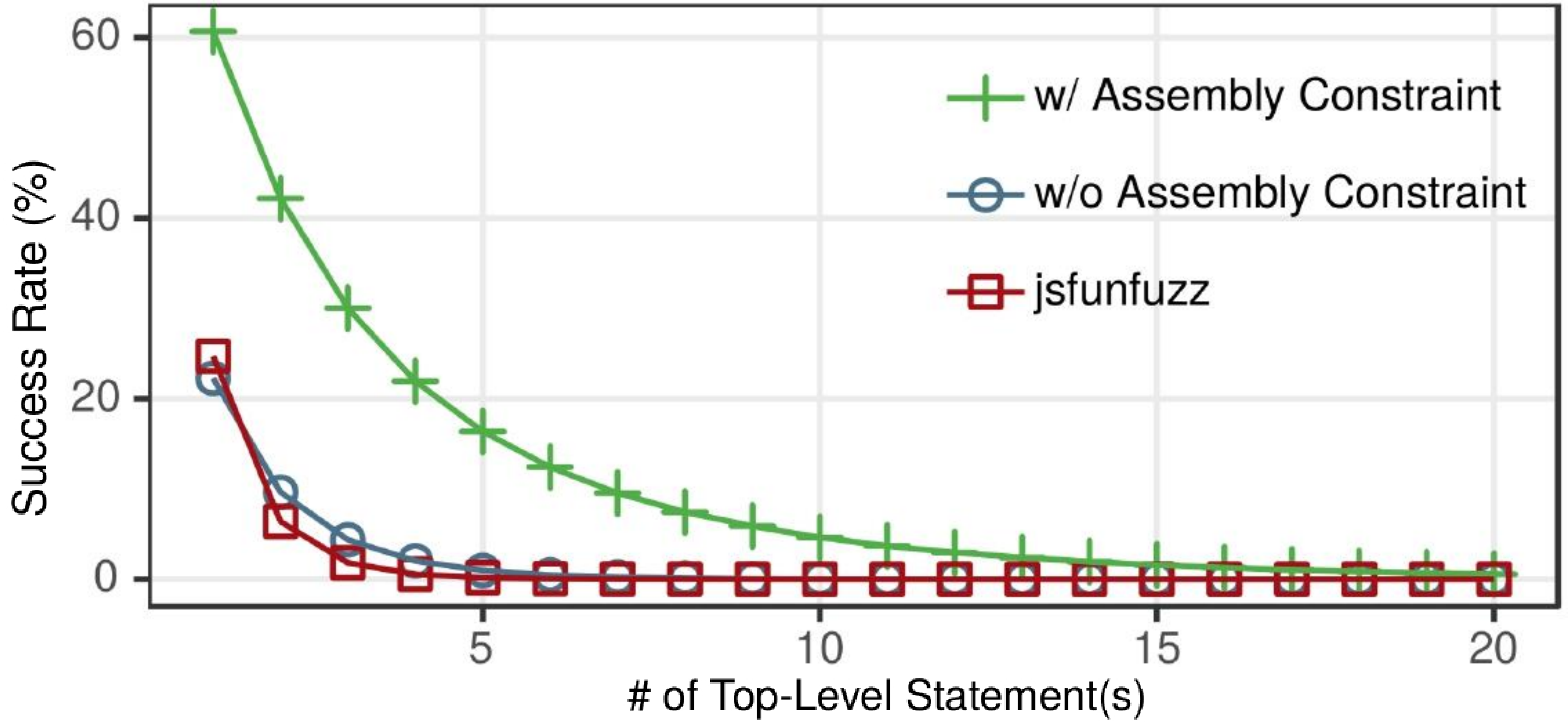

JS
Parser
ASTs,
Code Bricks
Analyzer
Code Brick Pool
Fuzzer
How valid?
How effective?

Experiment Setup

- Collect about **63,000** JS code snippets
  - Regression tests in four major JS engines
  - Test code snippets in Test262
  - PoC exploits for previous security bugs

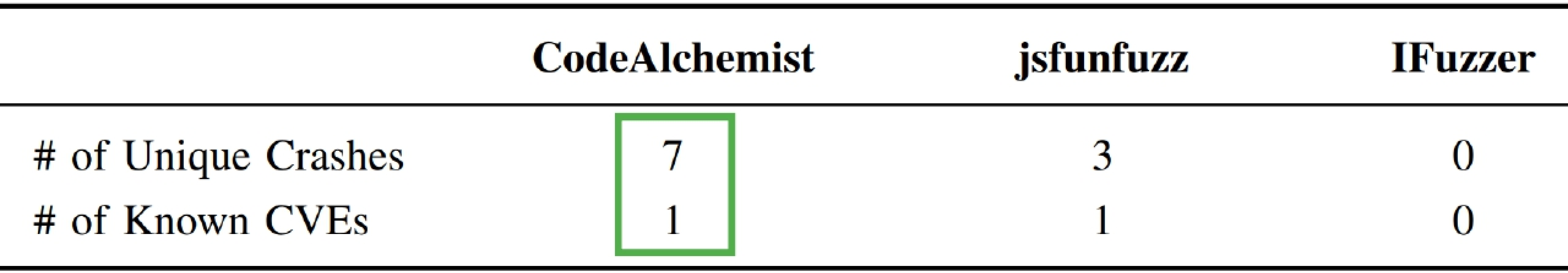
- The latest JS engines as of July 10th, 2018
  - ChakraCore 1.10.1
  - V8 6.7.288.46
  - JavaScriptCore 2.20.3
  - SpiderMonkey 61.0.1

Validity of Generated JS



vs. State-of-the-Arts (in Previous Ver.)

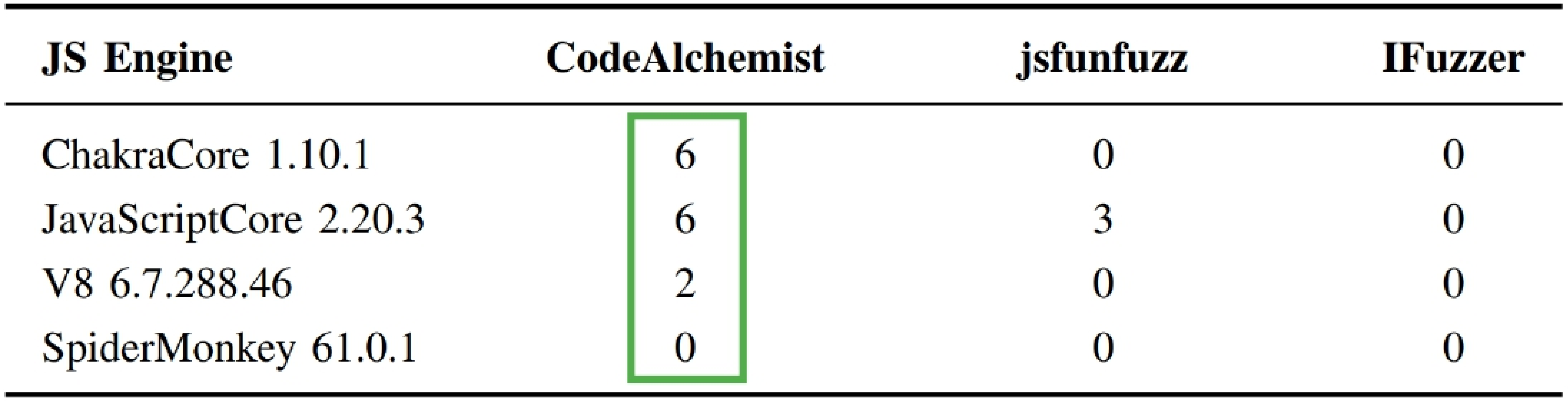
- Ran 24 hours for ChakraCore 1.7.6 (Jan. 9th, 2018)
- jsfunfuzz: the latest version before Jan. 9th, 2018
- Seeds: JS snippets before Jan. 9th, 2018



	CodeAlchemist	jsfunfuzz	IFuzzer
# of Unique Crashes	7	3	0
# of Known CVEs	1	1	0

vs. State-of-the-Arts

- jsfunfuzz: A **state-of-the-art** JS fuzzer developed by **Mozilla**
- IFuzzer: A variant of LangFuzz, ***ESORICS'16***
- Running time: 24 hours x 4 engines = 96 hours



JS Engine	**CodeAlchemist**	**jsfunfuzz**	**IFuzzer**
ChakraCore 1.10.1	6	0	0
JavaScriptCore 2.20.3	6	3	0
V8 6.7.288.46	2	0	0
SpiderMonkey 61.0.1	0	0	0

Real-World Bug Finding

- Ran a week for the latest JS engines
  - JavaScriptCore: 2.20.3, 2.21.4 (Beta)
  - V8: 6.7.288.46
  - SpiderMonkey: 61.0.1
  - ChakraCore: 1.10.0, 1.10.1

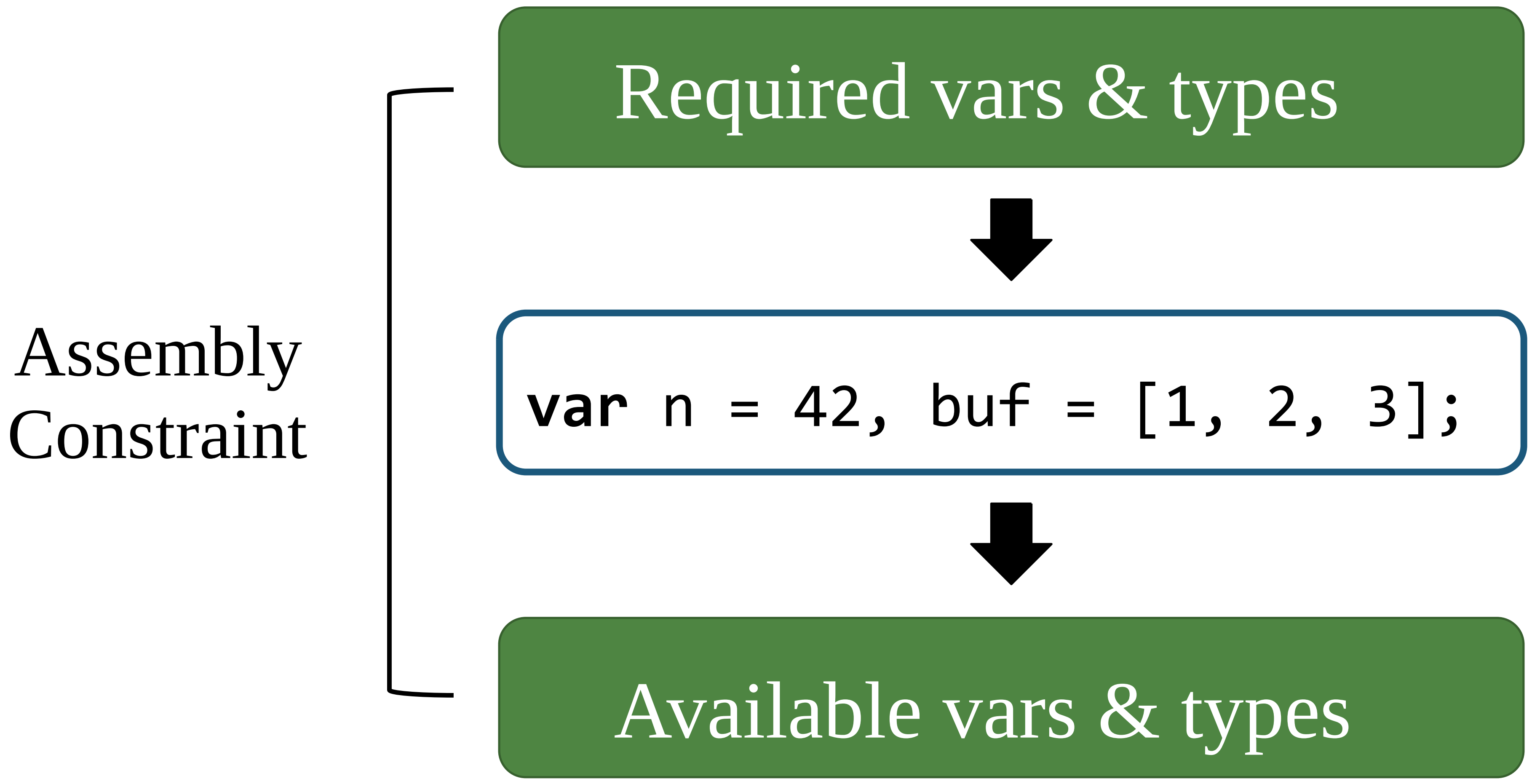
- Found 19 unique Bugs
  - 11 exploitaBle Bugs
  - 3 CVEs for us

Idx	JS Engine	Browser	Description	Impact	Status
1	JSC 2.20.3	Safari 11.1.1	Uninitialized memory access due to incorrect scoping	Exploitable	CVE-2018-4264
2	JSC 2.20.3	Safari 11.1.1	Use after free due to incorrect garbage collection	Exploitable	Confirmed
3	JSC 2.20.3	Safari 11.1.2	Memory corruption due to incorrect scoping	Exploitable	Confirmed
4	JSC 2.20.3	Safari 11.1.2	Memory corruption due to incorrect async function handling	Exploitable	Confirmed
5	JSC 2.20.3	Safari 11.1.2	Memory corruption due to incorrect regex parsing	Exploitable	Confirmed
6	JSC 2.20.3	Safari 11.1.2	Memory corruption due to incorrect date parsing	Exploitable	Confirmed
7	JSC 2.21.4 (beta)	Safari 11.1.2	Heap overflow due to incorrect string handling	Exploitable	CVE-2018-4437
8	JSC 2.21.4 (beta)	Safari 11.1.2	Memory corruption due to incorrect stack overflow handling	Exploitable	CVE-2018-4372
9	JSC 2.21.4 (beta)	Safari 12.0.0	Memory corruption due to incorrect JIT compilation	Exploitable	CVE-2018-4378
10	JSC 2.21.4 (beta)	Safari 11.1.2	Memory corruption due to incorrect string handling	Not Exploitable	Confirmed
11	V8 6.7.288.46	Chrome 67.0.3396.99	Out of bound access due to side effect in Float64Array	Exploitable	Confirmed
12	V8 6.7.288.46	Chrome 67.0.3396.99	Stack overflow due to incorrect recursively defined class handling	Not Exploitable	Confirmed
13	ChakraCore 1.10.0	-	Type confusion due to incorrect duplicated property handling	Exploitable	CVE-2018-8283
14	ChakraCore 1.10.1	-	Memory corruption due to incorrect yield handling in async function	Likely Exploitable	Reported
15	ChakraCore 1.10.1	-	Memory corruption due to incorrect JIT compilation	Likely Exploitable	Reported
16	ChakraCore 1.10.1	-	Use after free due to incorrect JIT compilation	Likely Exploitable	Reported
17	ChakraCore 1.10.1	Edge 43.17713.1000.0	Use after free due to incorrect JIT compilation	Not Exploitable	Confirmed
18	ChakraCore 1.10.1	Edge 43.17713.1000.0	Memory corruption due to incorrect JIT compilation	Not Exploitable	Confirmed
19	ChakraCore 1.10.1	Edge 43.17713.1000.0	Null dereference due to incorrect JIT compilation	Not Exploitable	Confirmed

Future Research

- Seed selection
- Simple random code brick selection
- Supporting other language interpreters or compilers

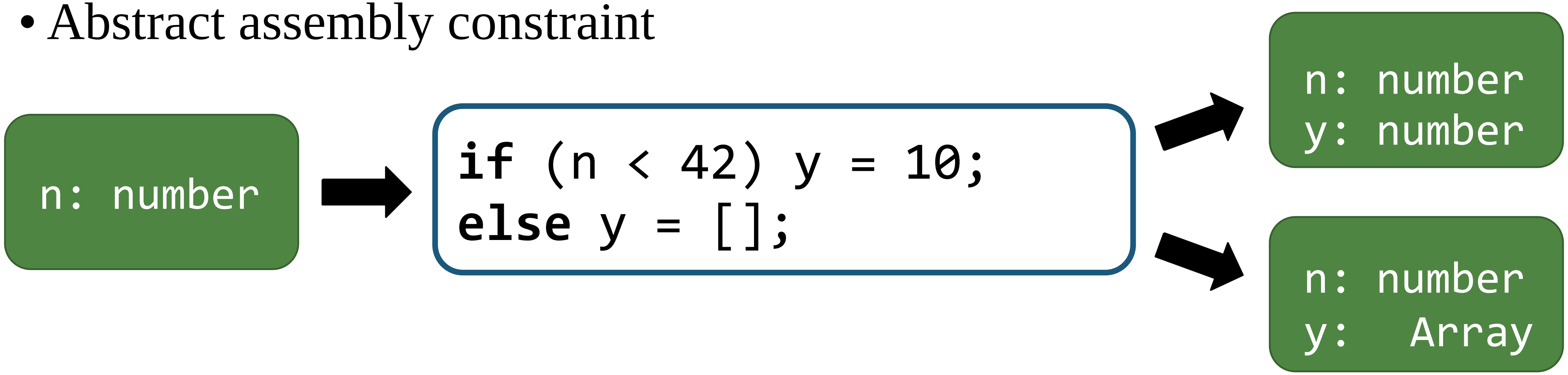
Assembly Constraint



```
var n = 42, buf = [1, 2, 3];
```

Why not 100% Success?

- Dynamic nature of JS
- Complex and large top-level statement
- Abstract assembly constraint



```
n: number
```

```
if (n < 42) y = 10;
else y = [];
```

```
n: number
y: number
```

```
n: number
y:   Array
```