Language Design and Implementation using JavaScript Mechanized Specification

Jihyeok Park and Sukyoung Ryu

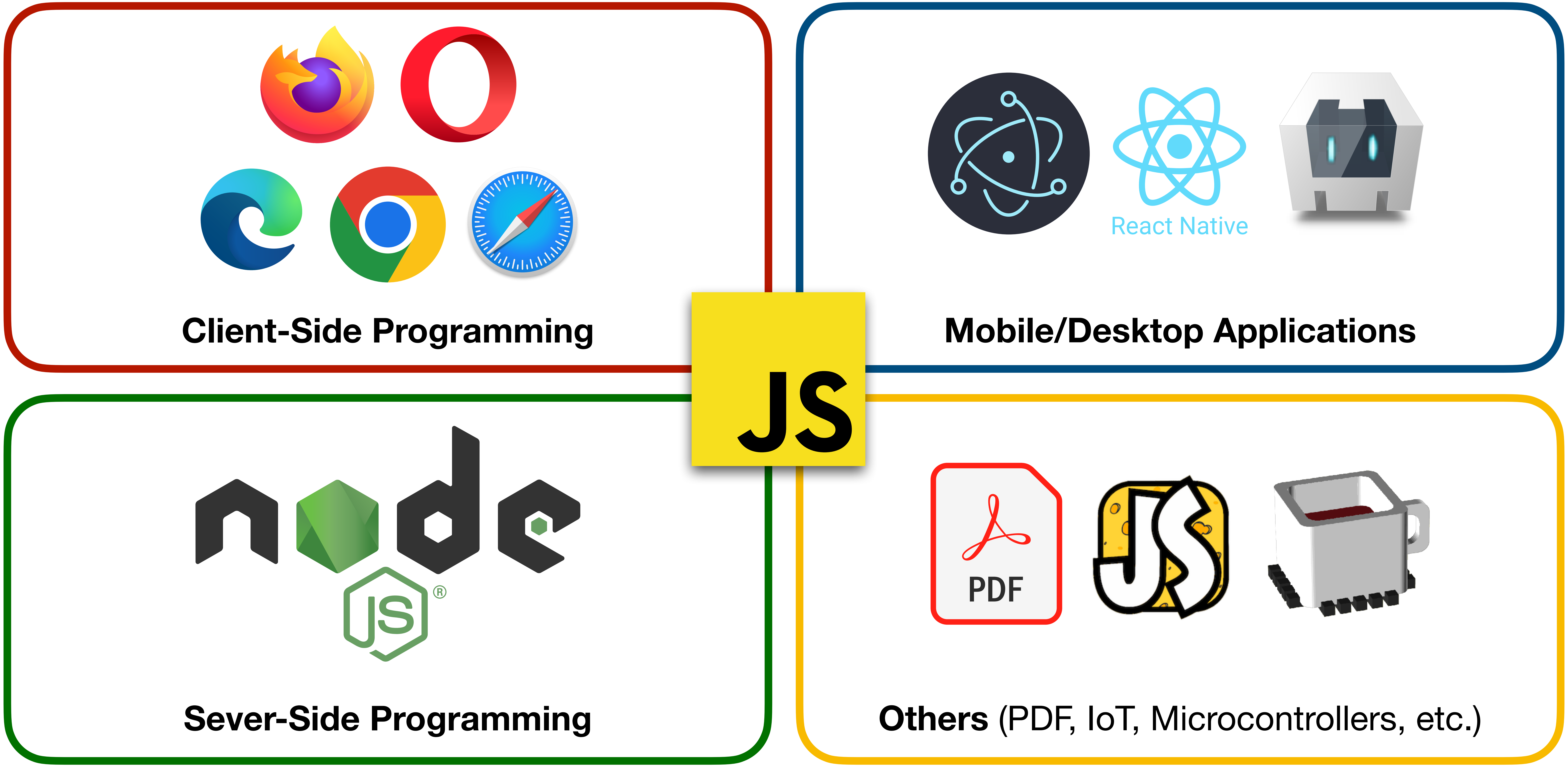
KOREA UNIVERSITY

KAIST

Instituto Superior Técnico of University of Lisbon

2024.06.24

JavaScript is Everywhere



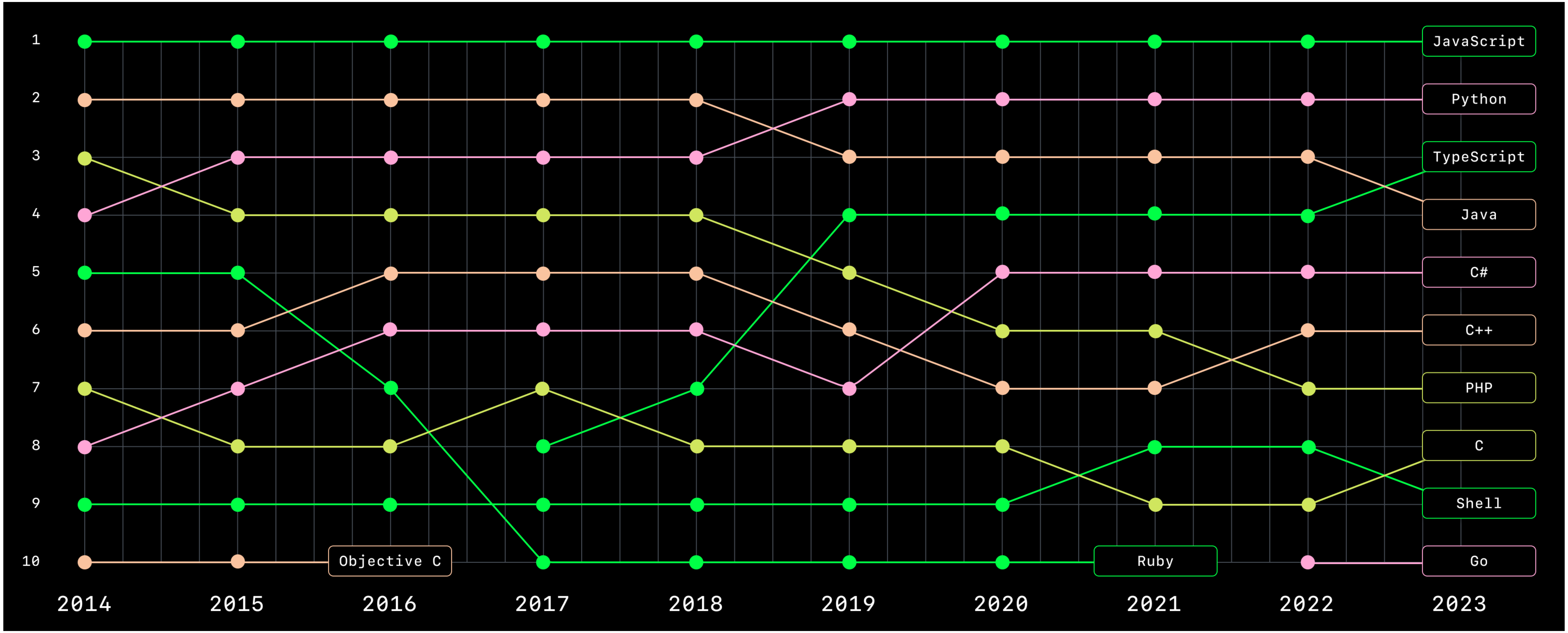
Client-Side Programming

Mobile/Desktop Applications

Sever-Side Programming

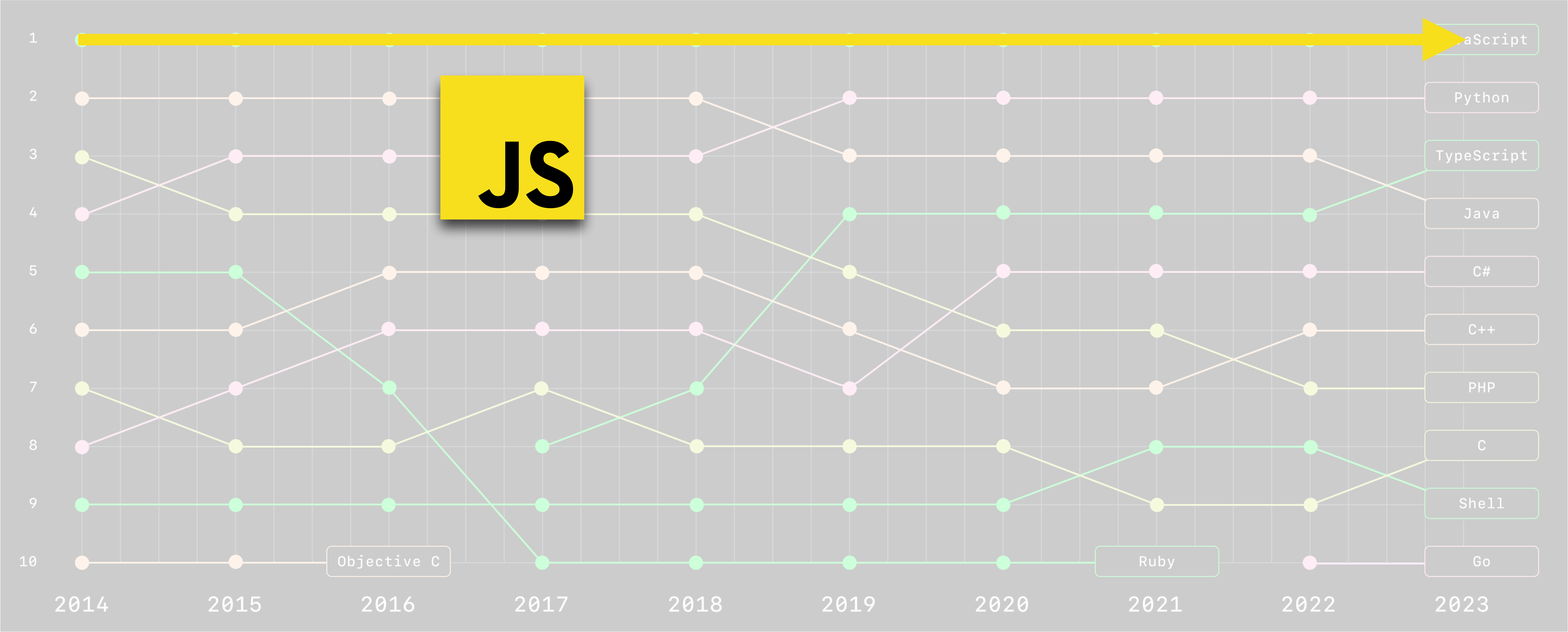
Others (PDF, IoT, Microcontrollers, etc.)

JavaScript is Everywhere



https://octoverse.github.com/

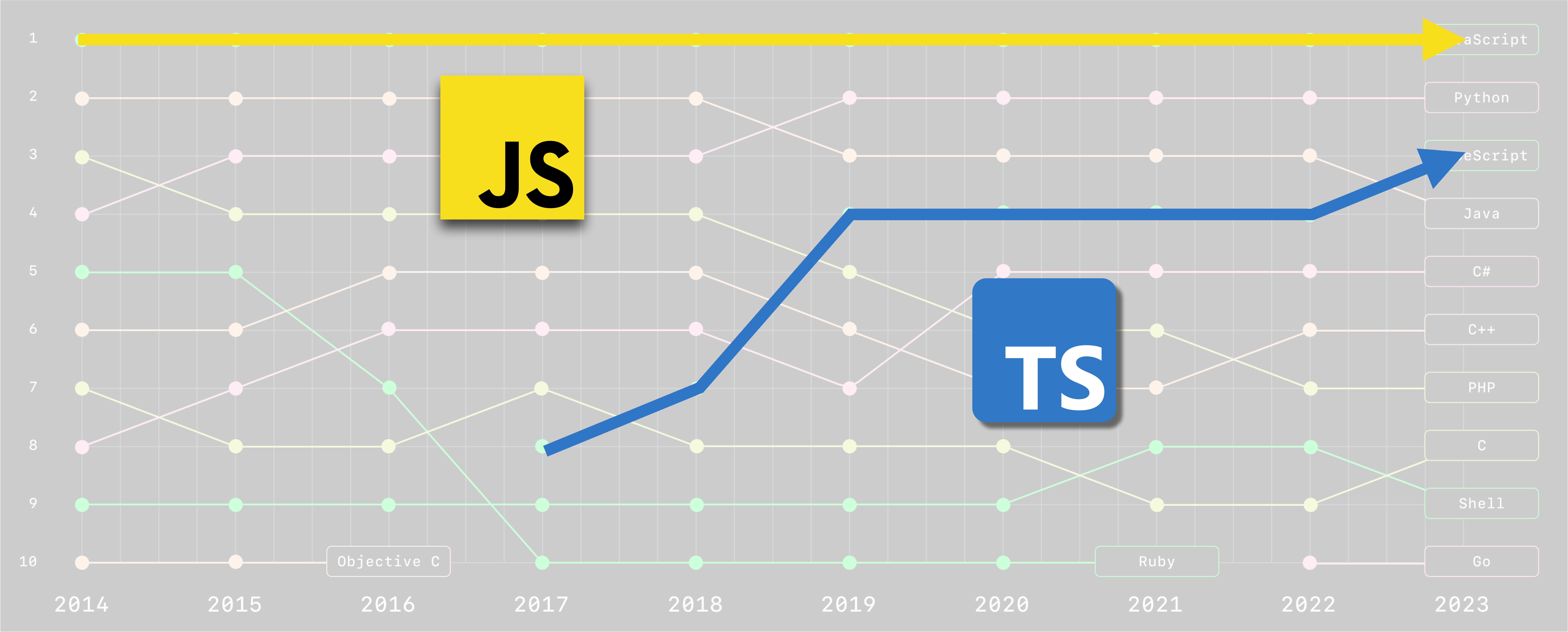
JavaScript is Everywhere



https://octoverse.github.com/

JavaScript is Everywhere

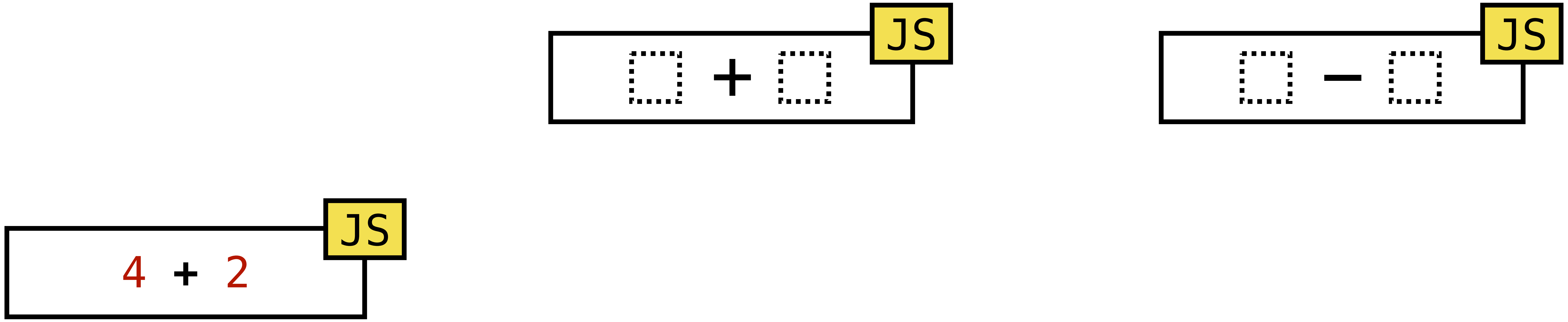
GitHub



https://octoverse.github.com/

But, JavaScript is Complicated

But, **JavaScript** is Complicated



JS

```
⬚ + ⬚
```

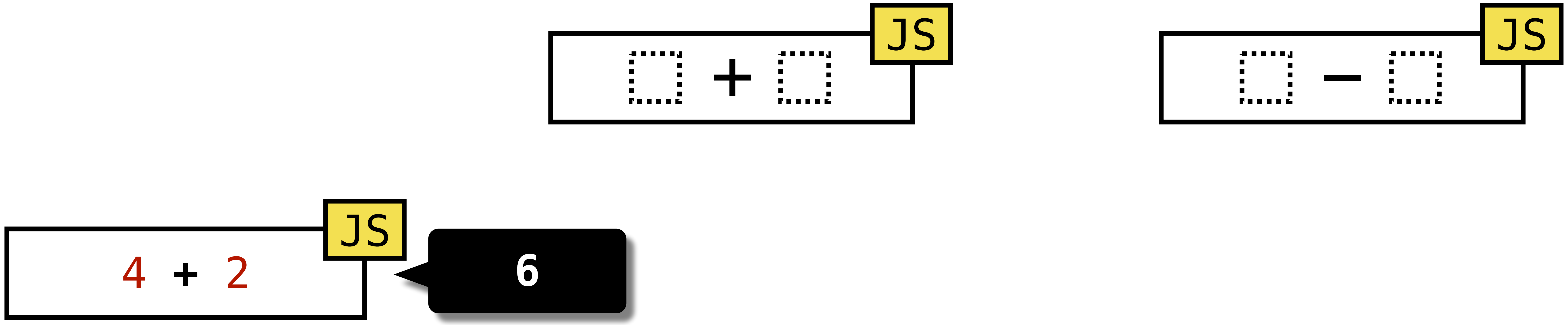
JS

```
⬚ − ⬚
```

JS

```
4 + 2
```

But, JavaScript is Complicated



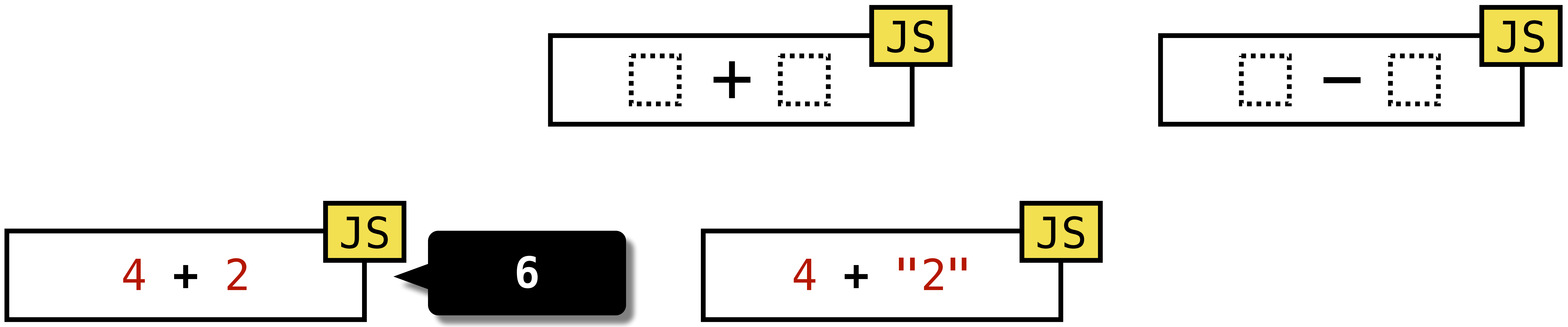
□ + □ JS

□ − □ JS

4 + 2 JS

6

But, JavaScript is Complicated



JS

```
□ + □
```

JS

```
□ − □
```

JS

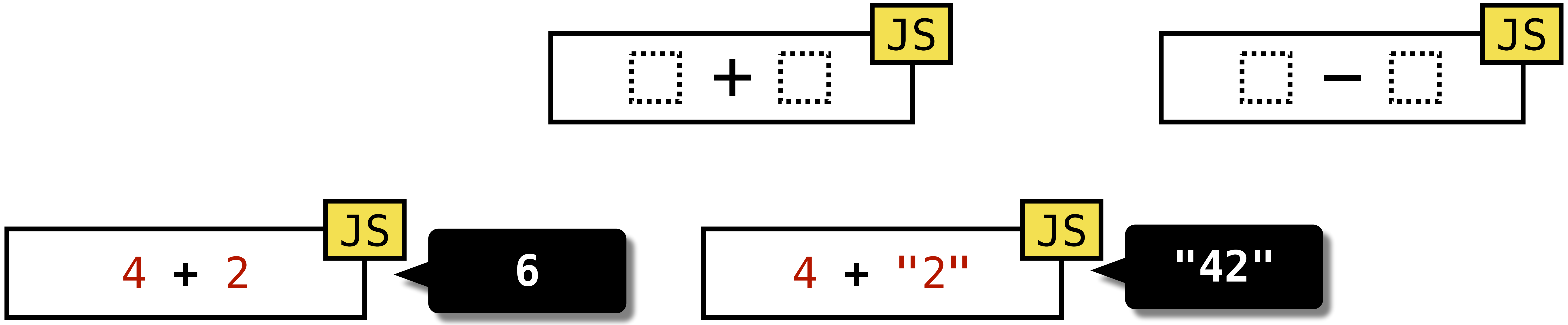
```
4 + 2
```

6

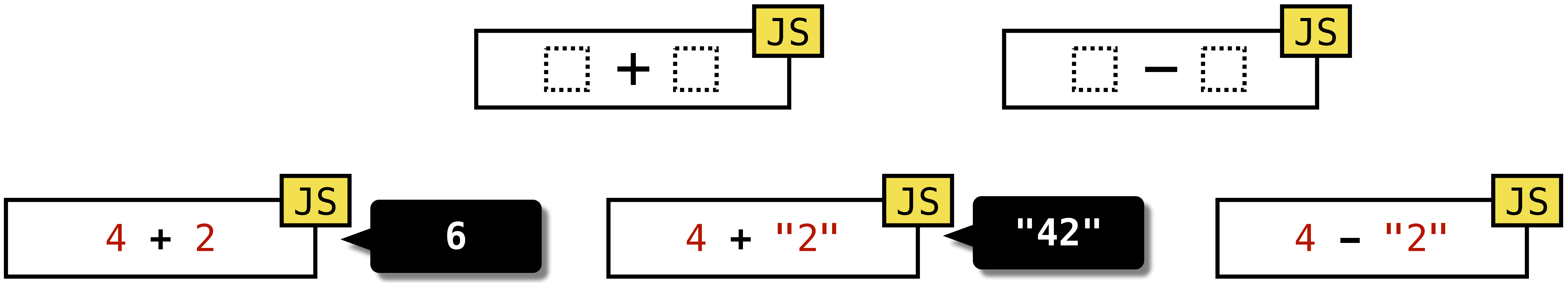
JS

```
4 + "2"
```

But, JavaScript is Complicated



But, JavaScript is Complicated



```
⬚ + ⬚
```

```
⬚ - ⬚
```

```
4 + 2
```

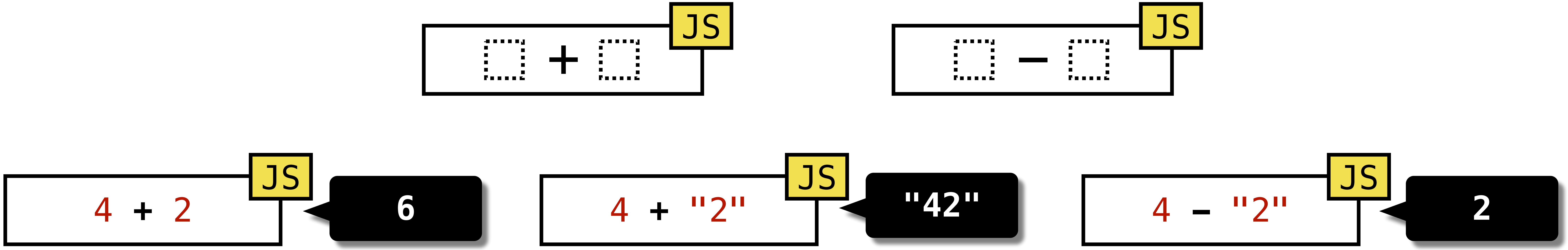
```
6
```

```
4 + "2"
```

```
"42"
```

```
4 - "2"
```

But, JavaScript is Complicated



```
□ + □
```

```
□ − □
```

```
4 + 2
```

```
6
```

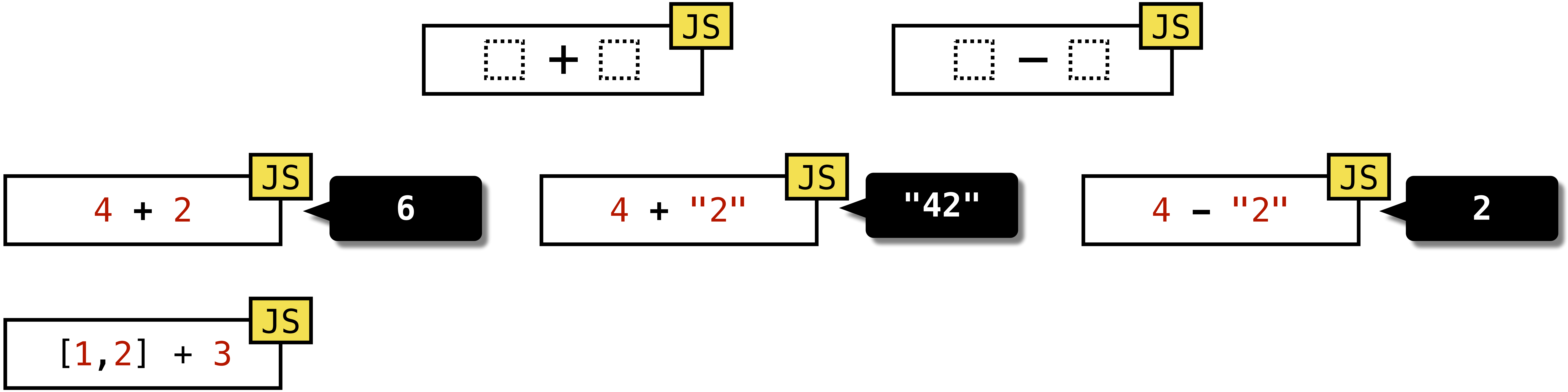
```
4 + "2"
```

```
"42"
```

```
4 - "2"
```

```
2
```

But, JavaScript is Complicated



JS

□ + □

JS

□ − □

JS

`4 + 2`

6

JS

`4 + "2"`

"42"

JS

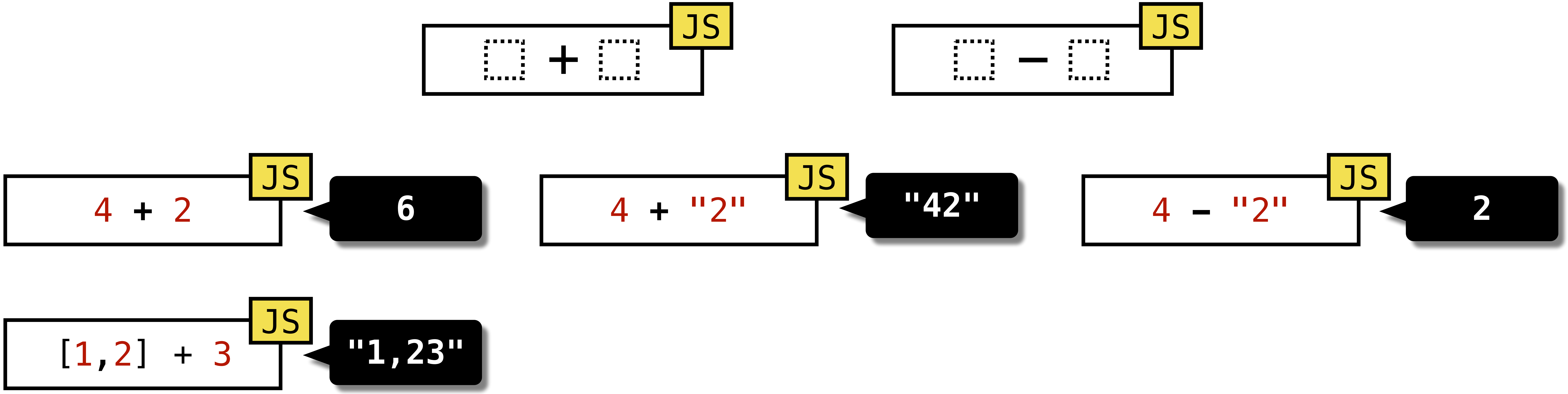
`4 - "2"`

2

JS

`[1,2] + 3`

But, JavaScript is Complicated



JS

```
□ + □
```

JS

```
□ - □
```

JS

```
4 + 2
```

```
6
```

JS

```
4 + "2"
```

```
"42"
```

JS

```
4 - "2"
```

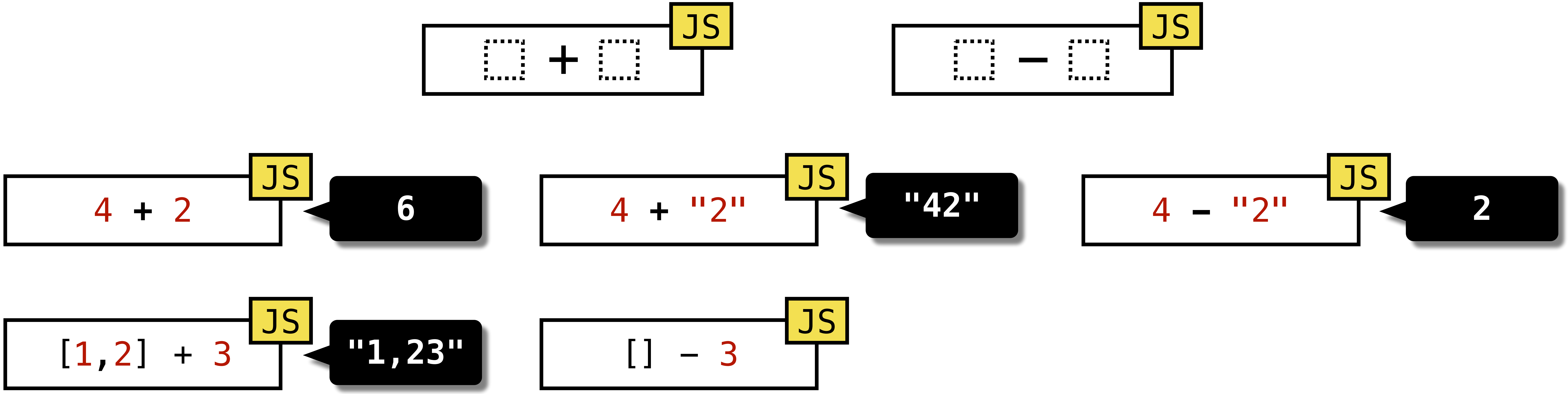
```
2
```

JS

```
[1,2] + 3
```

```
"1,23"
```

But, JavaScript is Complicated



JS `□ + □`

JS `□ - □`

JS `4 + 2` 6

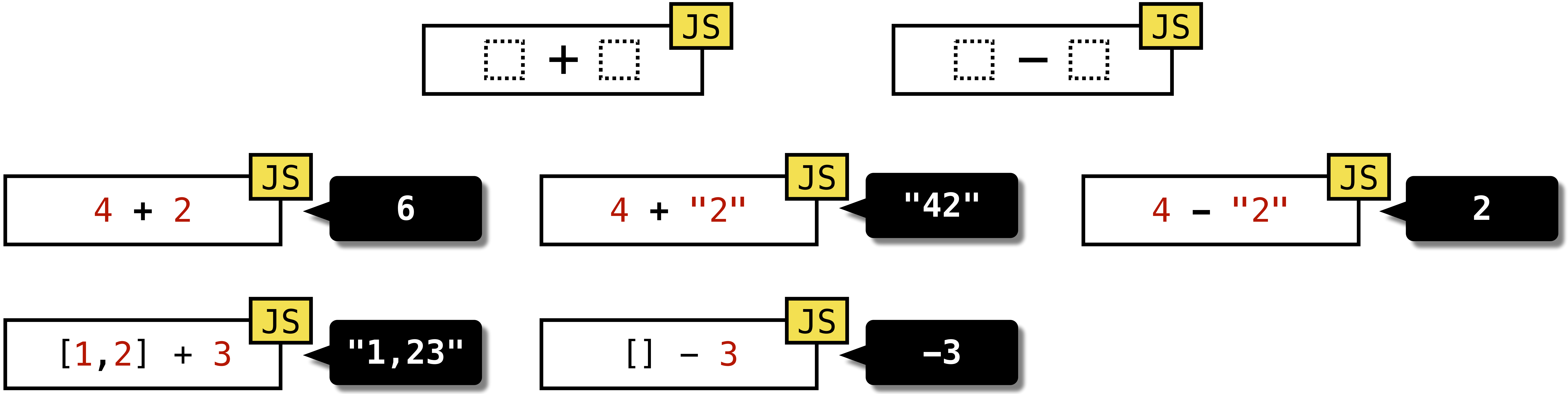
JS `4 + "2"` "42"

JS `4 - "2"` 2

JS `[1,2] + 3` "1,23"

JS `[] - 3`

But, JavaScript is Complicated



JS

□ + □

JS

□ − □

JS

```
4 + 2
```

6

JS

```
4 + "2"
```

"42"

JS

```
4 - "2"
```

2

JS

```
[1,2] + 3
```

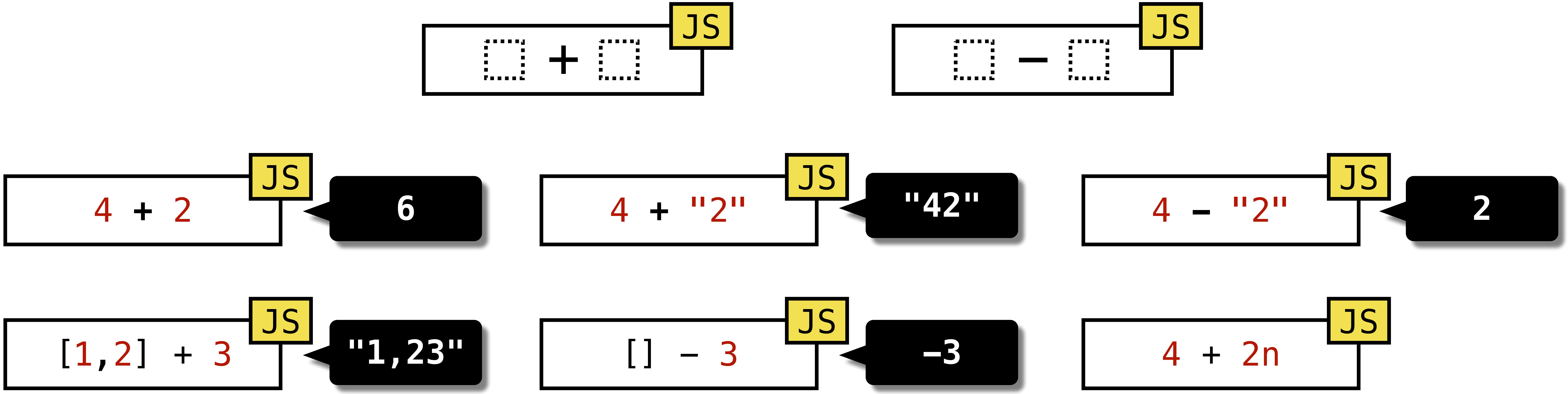
"1,23"

JS

```
[] - 3
```

-3

But, JavaScript is Complicated



```
□ + □
```

```
□ - □
```

```
4 + 2
```

```
6
```

```
4 + "2"
```

```
"42"
```

```
4 - "2"
```

```
2
```

```
[1,2] + 3
```

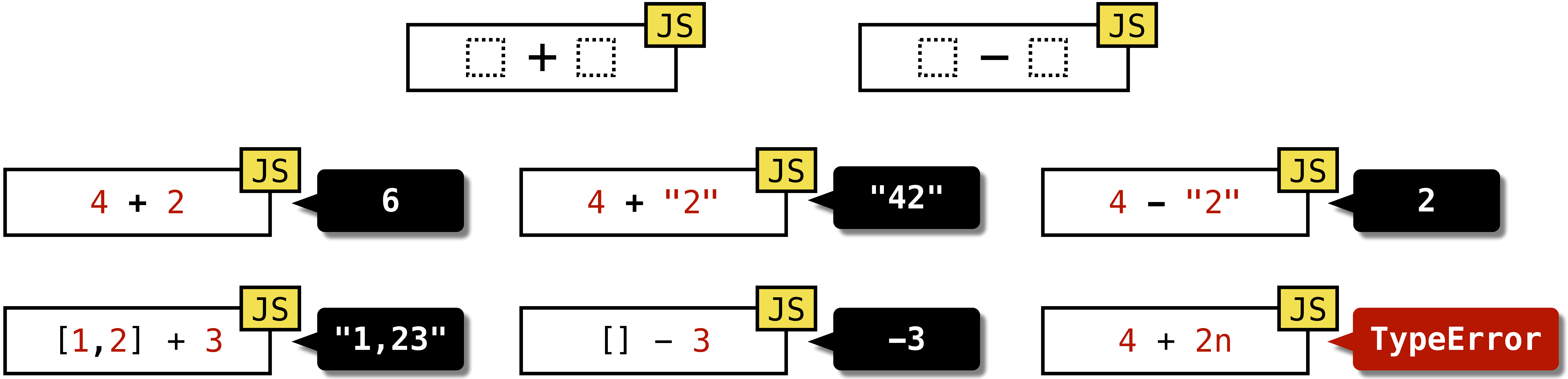
```
"1,23"
```

```
[] - 3
```

```
-3
```

```
4 + 2n
```

But, JavaScript is Complicated



JS □ + □

JS □ − □

JS 4 + 2 → 6

JS 4 + "2" → "42"

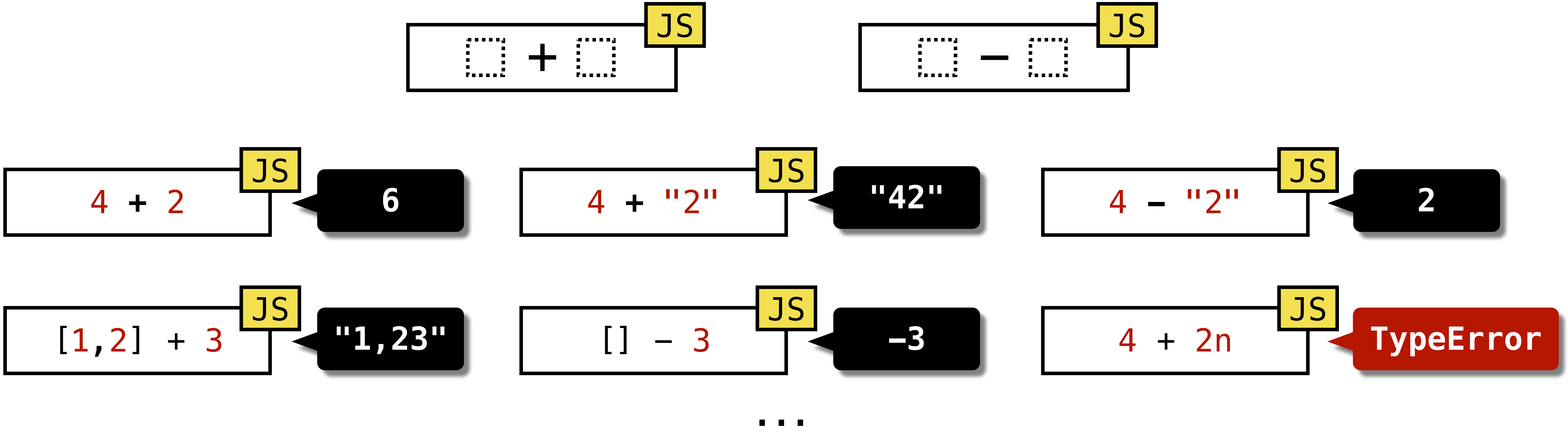
JS 4 − "2" → 2

JS [1,2] + 3 → "1,23"

JS [] − 3 → −3

JS 4 + 2n → TypeError

But, JavaScript is Complicated



```
□ + □
□ − □
```

```
4 + 2
6
```

```
4 + "2"
"42"
```

```
4 − "2"
2
```

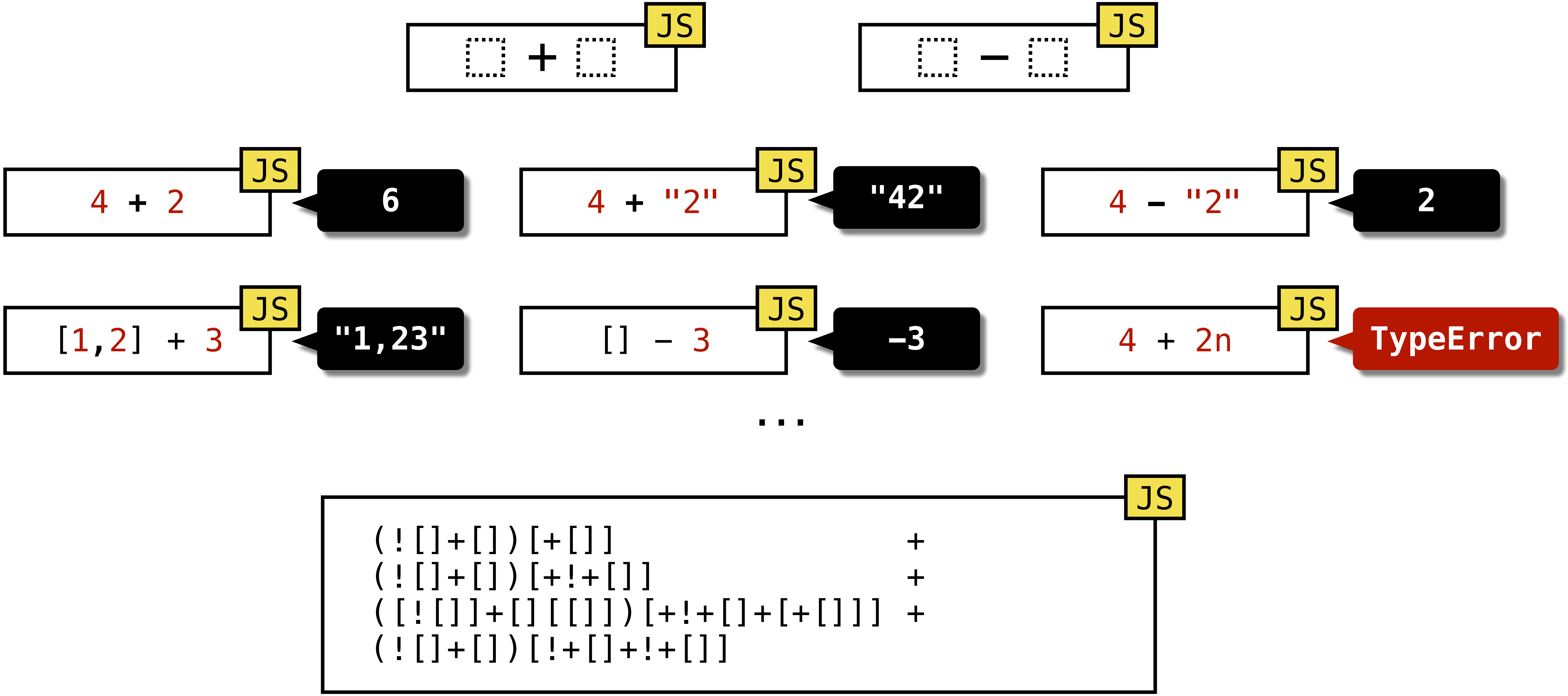
```
[1,2] + 3
"1,23"
```

```
[] − 3
−3
```

```
4 + 2n
TypeError
```

...

But, JavaScript is Complicated



JS `□ + □`

JS `□ − □`

JS `4 + 2` → 6

JS `4 + "2"` → "42"

JS `4 - "2"` → 2

JS `[1,2] + 3` → "1,23"

JS `[] - 3` → -3

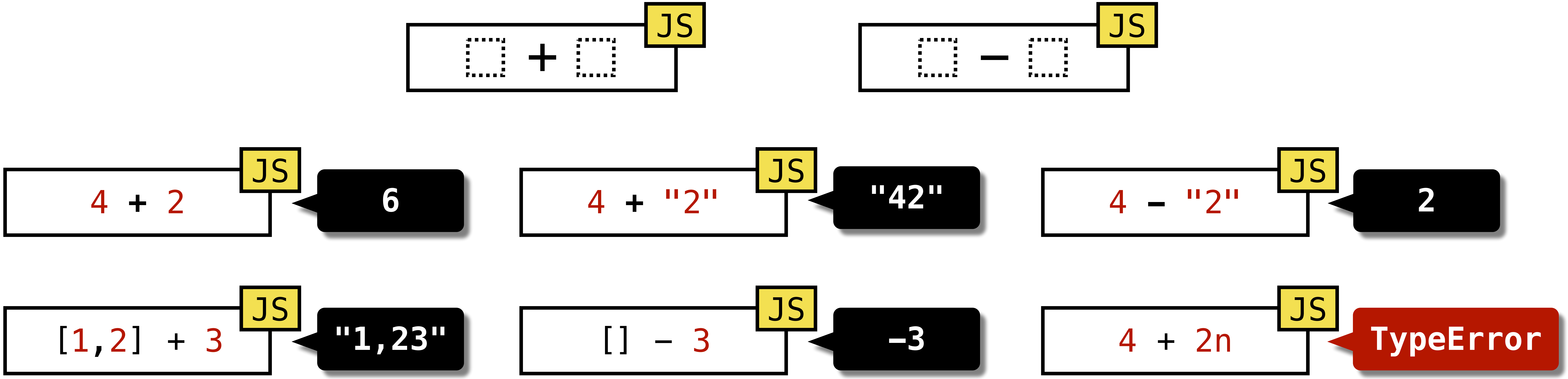
JS `4 + 2n` → TypeError

...

JS

```
(![]+[])[+[]]                +
(![]+[])[+!+[]]              +
([![]]+[][[]])[+!+[]+[+[]]] +
(![]+[])[!+[]+!+[]]
```

But, JavaScript is Complicated



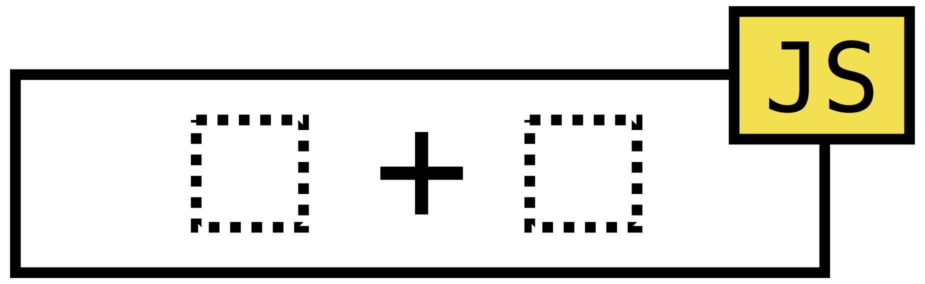
...

JS

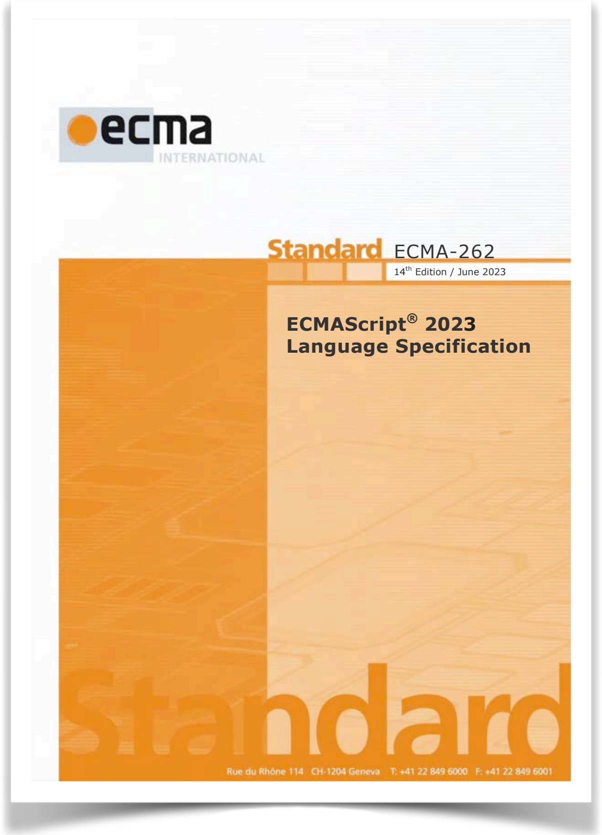
```
(![]+[])[+[]]                    +   // "f"
(![]+[])[+!+[]]                  +   // "a"
([![]]+[][[]])[+!+[]+[+[]]]      +   // "i"
(![]+[])[!+[]+!+[]]                  // "l"
```

"fail"

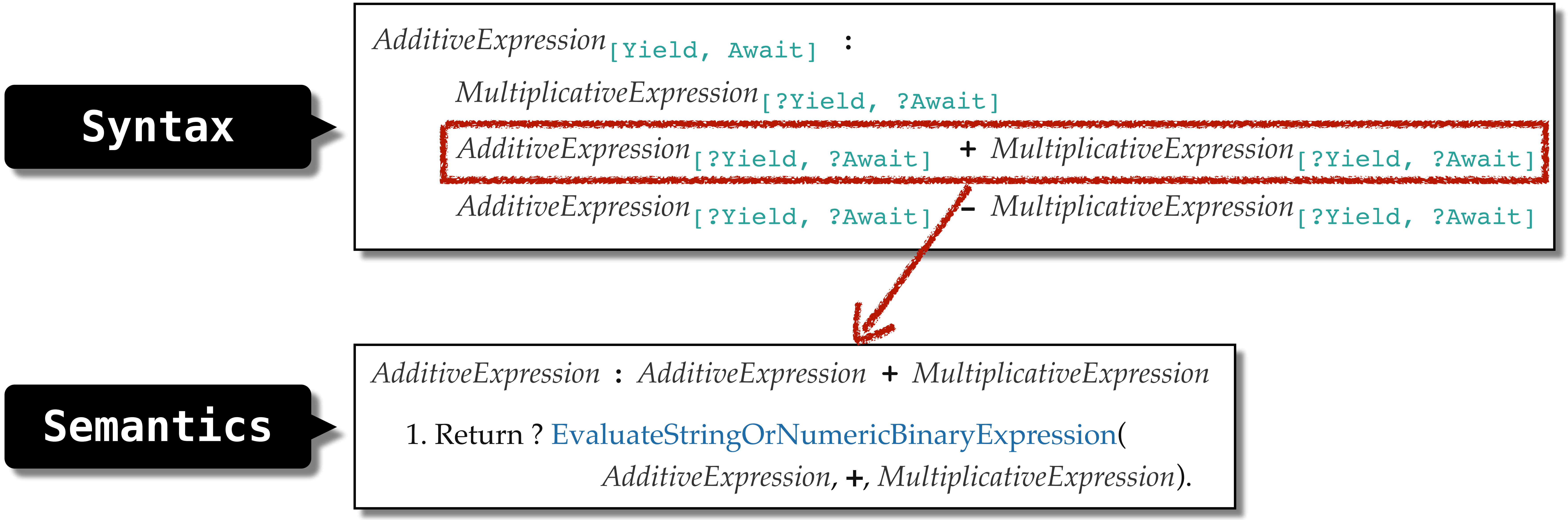
Language Specification (ECMA-262) of JavaScript



TC 39



ECMA-262
(JavaScript Spec.)


Syntax

AdditiveExpression$_{[Yield, Await]}$:
MultiplicativeExpression$_{[?Yield, ?Await]}$
AdditiveExpression$_{[?Yield, ?Await]}$ **+** *MultiplicativeExpression*$_{[?Yield, ?Await]}$
AdditiveExpression$_{[?Yield, ?Await]}$ **-** *MultiplicativeExpression*$_{[?Yield, ?Await]}$

Semantics

AdditiveExpression : *AdditiveExpression* **+** *MultiplicativeExpression*

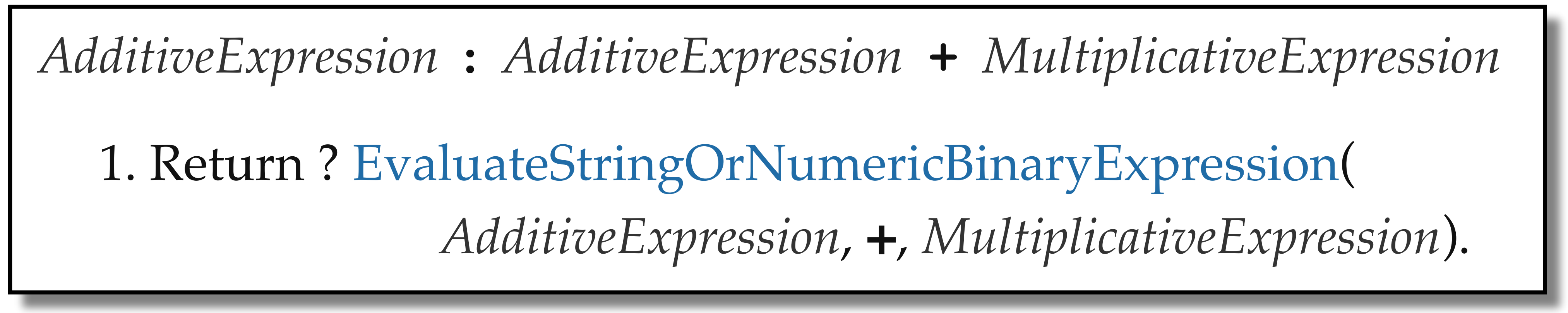
1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

Language Specification (ECMA-262) of JavaScript

JS

```
4 + 2n
```

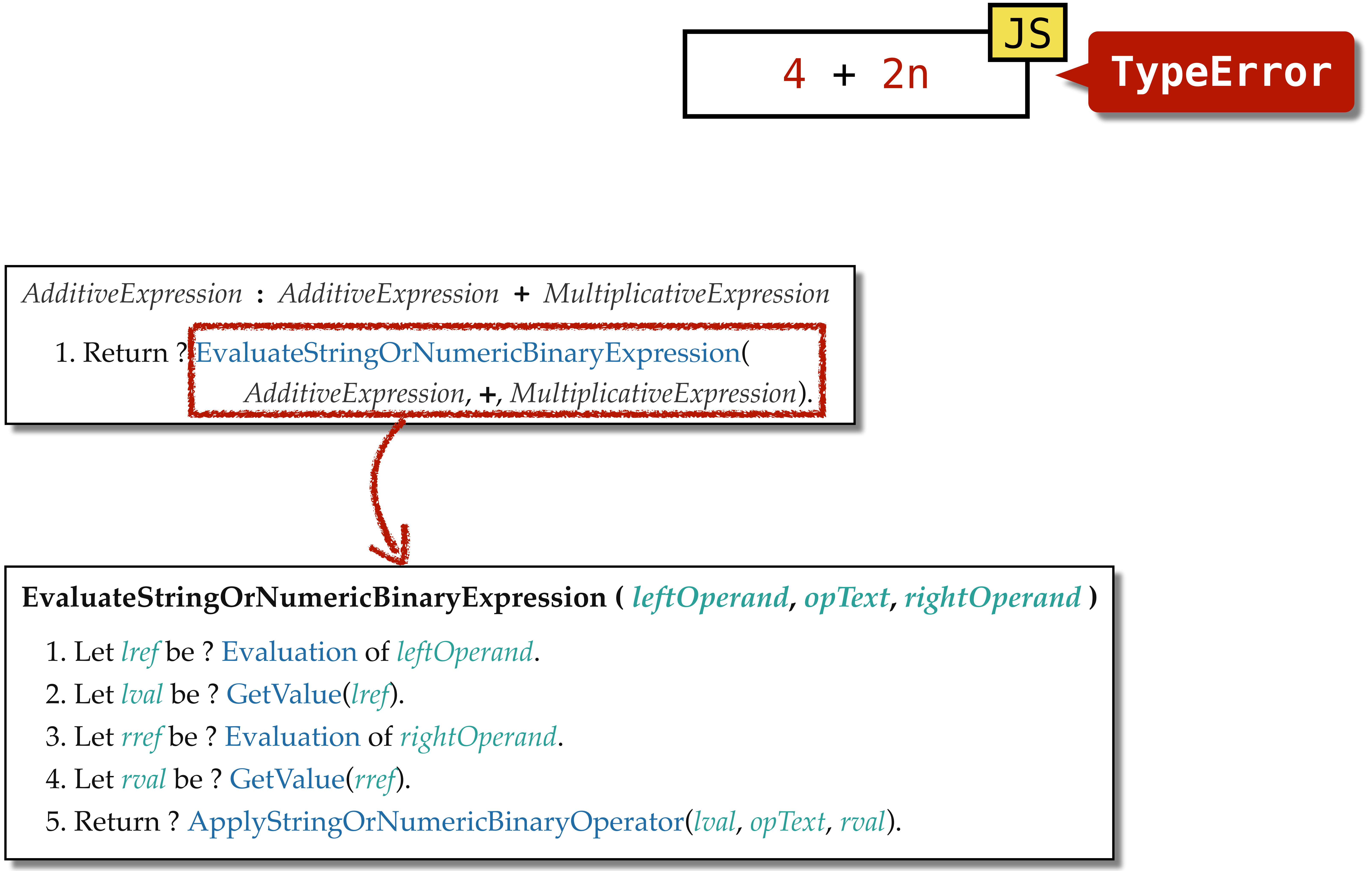
TypeError



AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

AdditiveExpression : *AdditiveExpression* **+** *MultiplicativeExpression*

```
1. Return ? EvaluateStringOrNumericBinaryExpression(
       AdditiveExpression, +, MultiplicativeExpression).
```

EvaluateStringOrNumericBinaryExpression (***leftOperand***, ***opText***, ***rightOperand*** **)**

```
1. Let lref be ? Evaluation of leftOperand.
2. Let lval be ? GetValue(lref).
3. Let rref be ? Evaluation of rightOperand.
4. Let rval be ? GetValue(rref).
5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

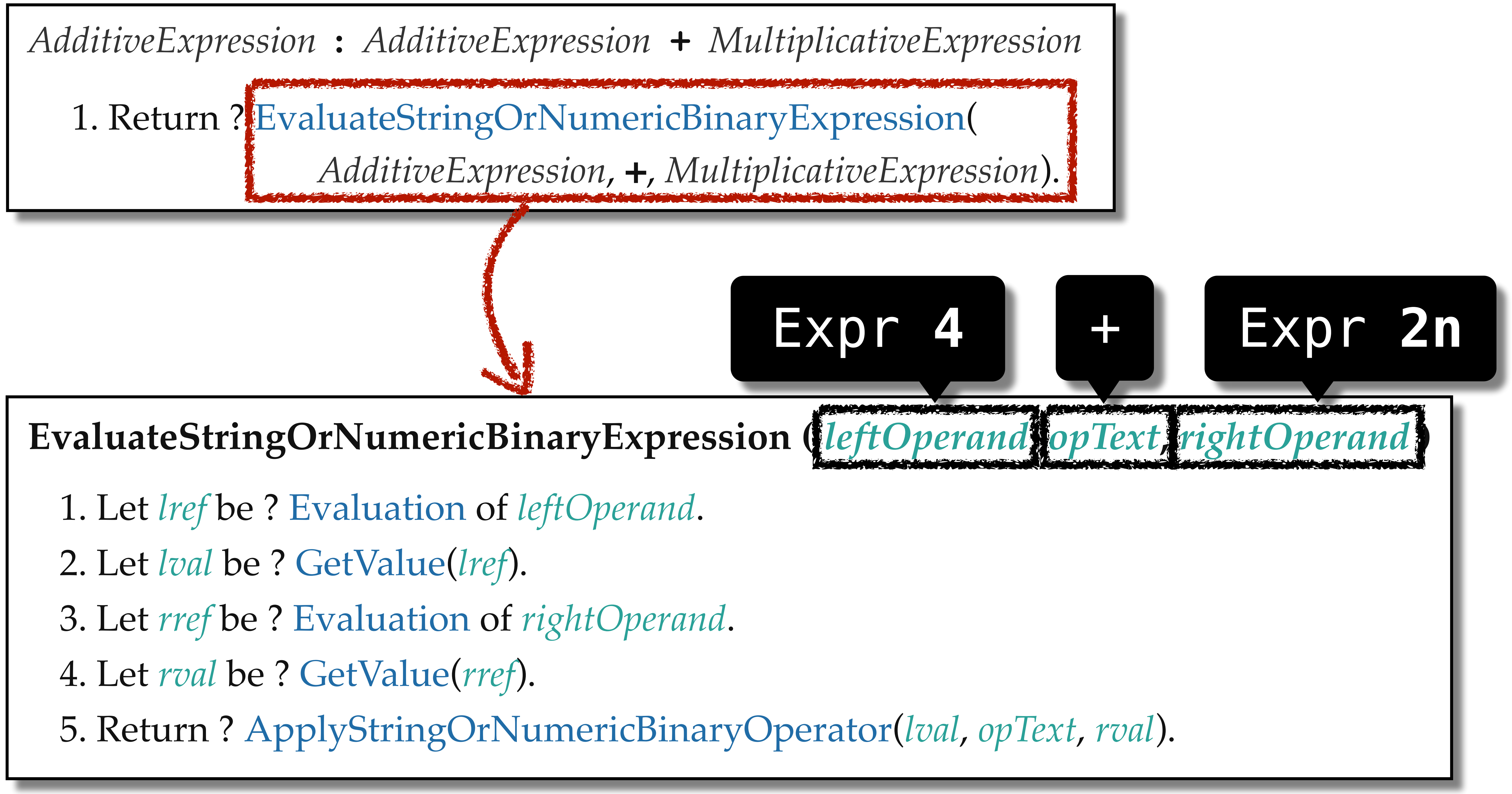
PLRG

Language Specification (ECMA-262) of JavaScript

JS

```
4 + 2n
```

TypeError



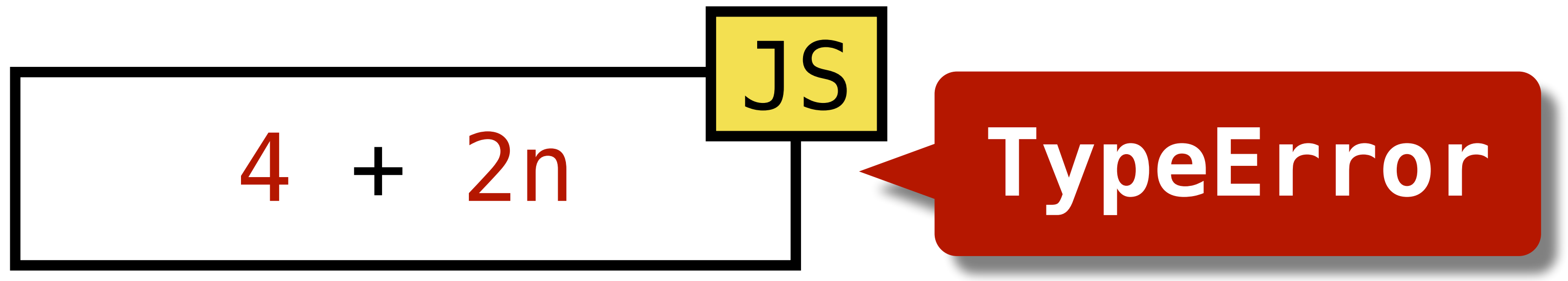
```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

  1. Return ? EvaluateStringOrNumericBinaryExpression(
         AdditiveExpression, +, MultiplicativeExpression).
```

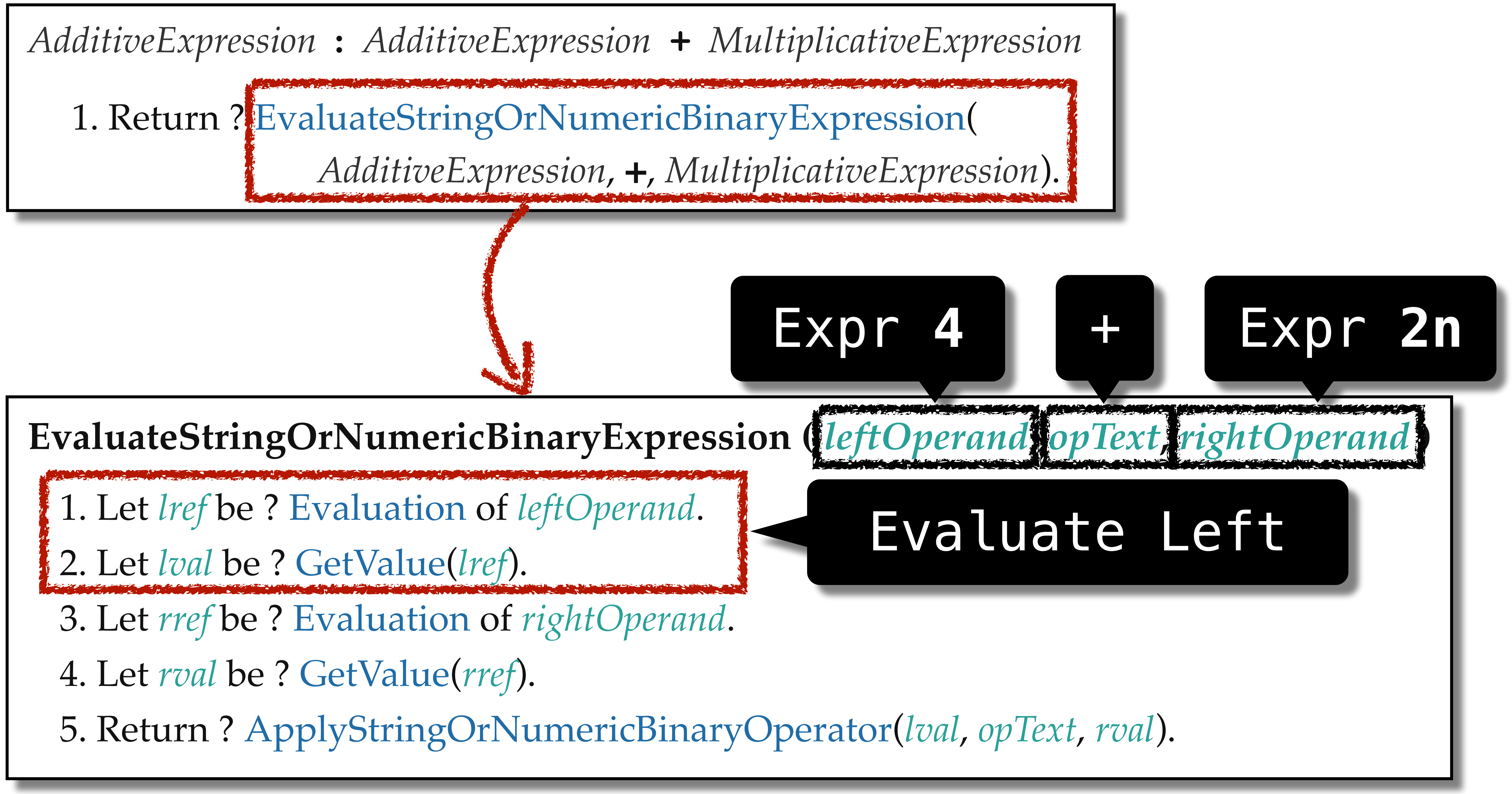
```
EvaluateStringOrNumericBinaryExpression ( leftOperand, opText, rightOperand )

  1. Let lref be ? Evaluation of leftOperand.
  2. Let lval be ? GetValue(lref).
  3. Let rref be ? Evaluation of rightOperand.
  4. Let rval be ? GetValue(rref).
  5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```



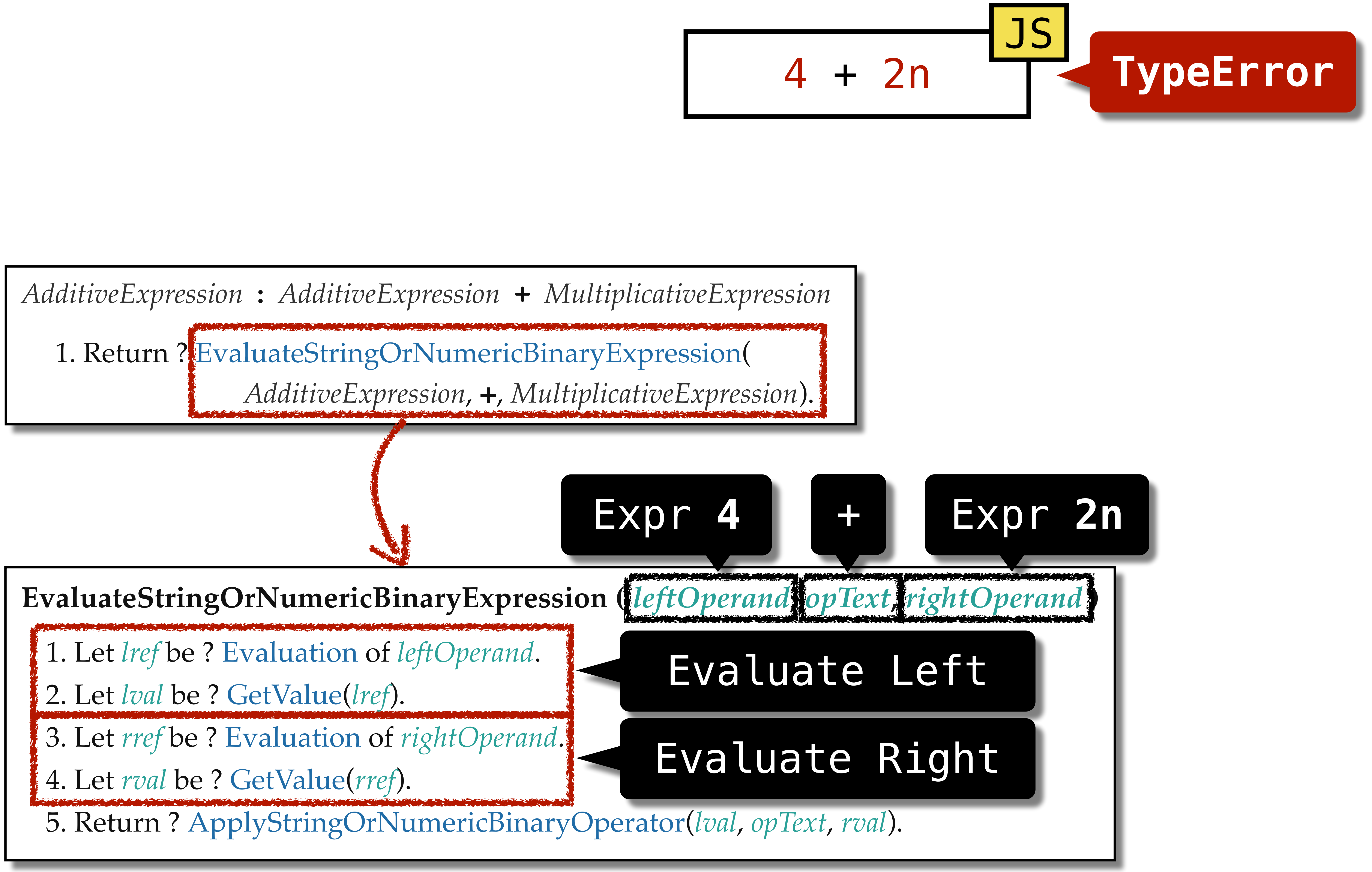
AdditiveExpression : *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Language Specification (ECMA-262) of JavaScript



JS

```
4 + 2n
```

TypeError

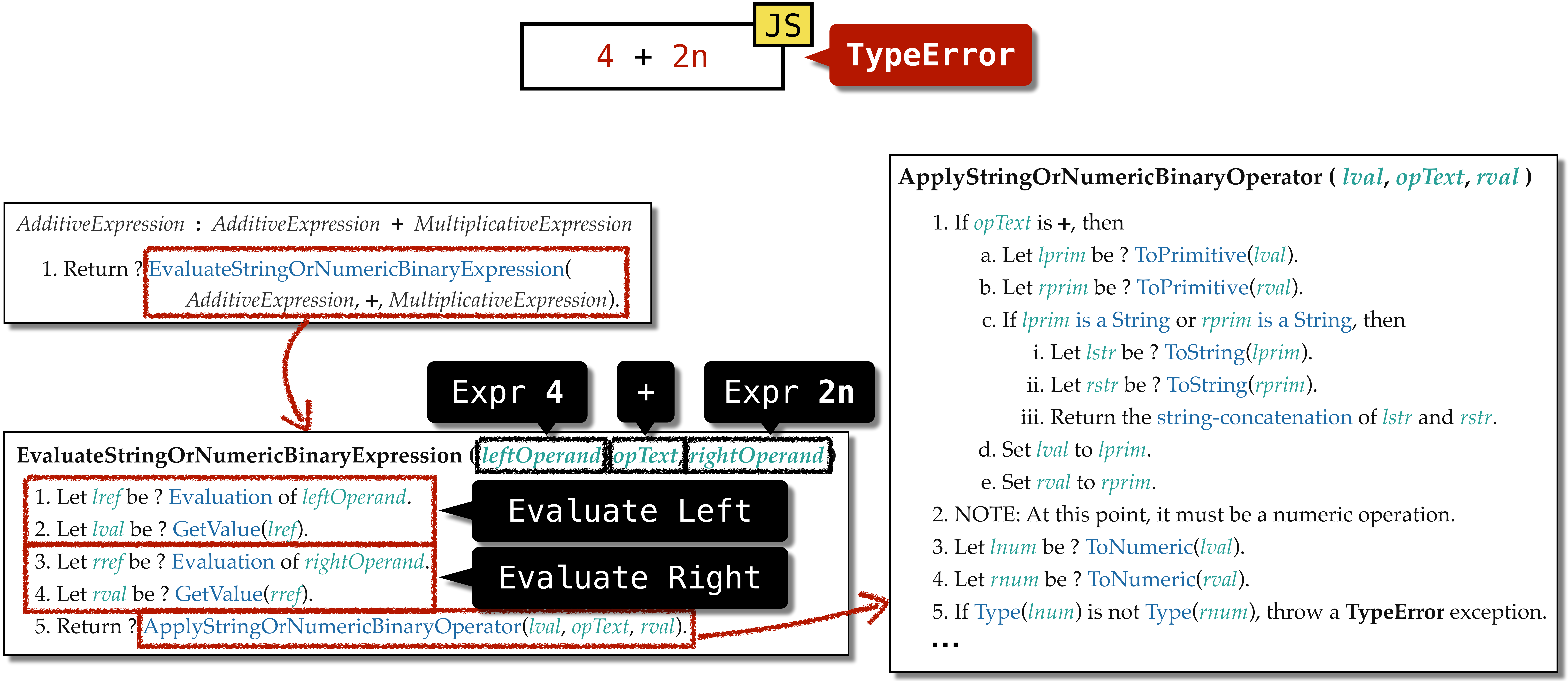
AdditiveExpression : *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

AdditiveExpression : *AdditiveExpression* **+** *MultiplicativeExpression*

```
1. Return ? EvaluateStringOrNumericBinaryExpression(
       AdditiveExpression, +, MultiplicativeExpression).
```

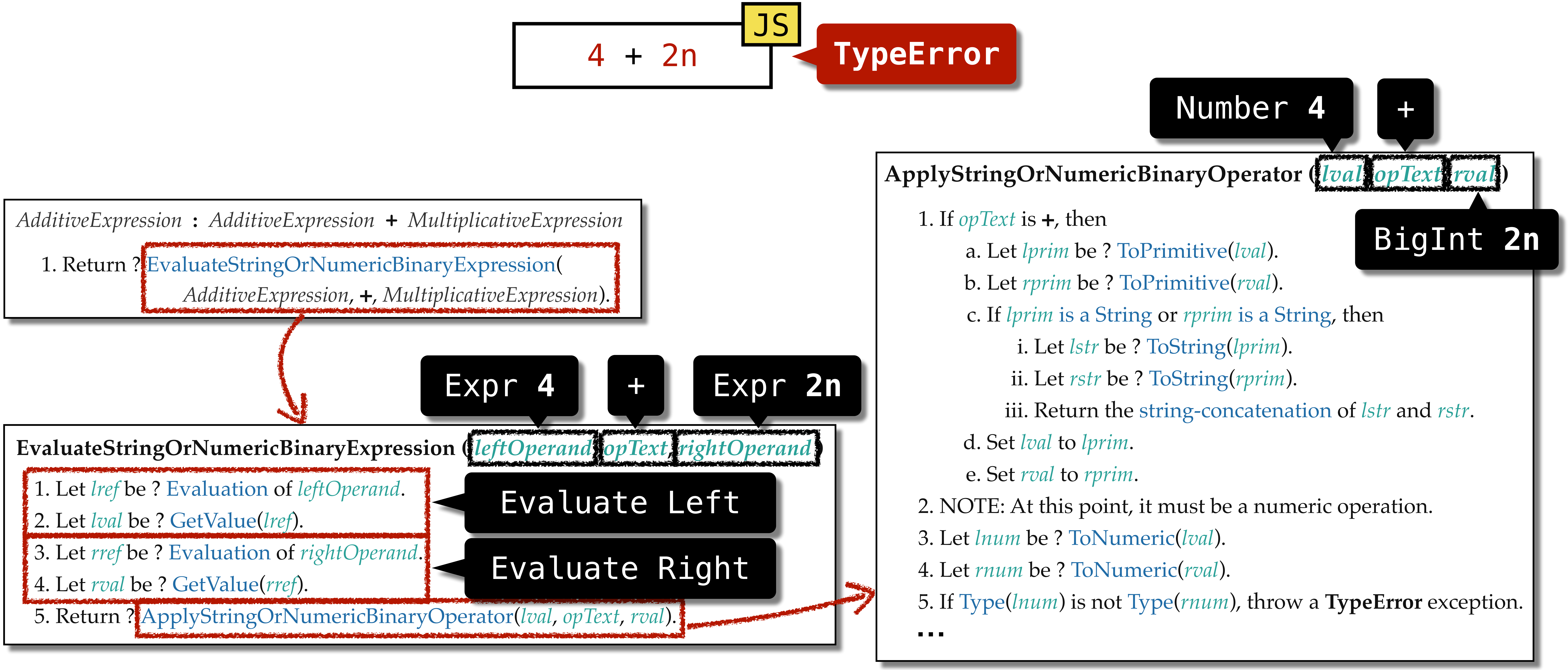
EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

```
1. Let lref be ? Evaluation of leftOperand.
2. Let lval be ? GetValue(lref).
3. Let rref be ? Evaluation of rightOperand.
4. Let rval be ? GetValue(rref).
5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

```
1. If opText is +, then
    a. Let lprim be ? ToPrimitive(lval).
    b. Let rprim be ? ToPrimitive(rval).
    c. If lprim is a String or rprim is a String, then
        i. Let lstr be ? ToString(lprim).
        ii. Let rstr be ? ToString(rprim).
        iii. Return the string-concatenation of lstr and rstr.
    d. Set lval to lprim.
    e. Set rval to rprim.
2. NOTE: At this point, it must be a numeric operation.
3. Let lnum be ? ToNumeric(lval).
4. Let rnum be ? ToNumeric(rval).
5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
...
```

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

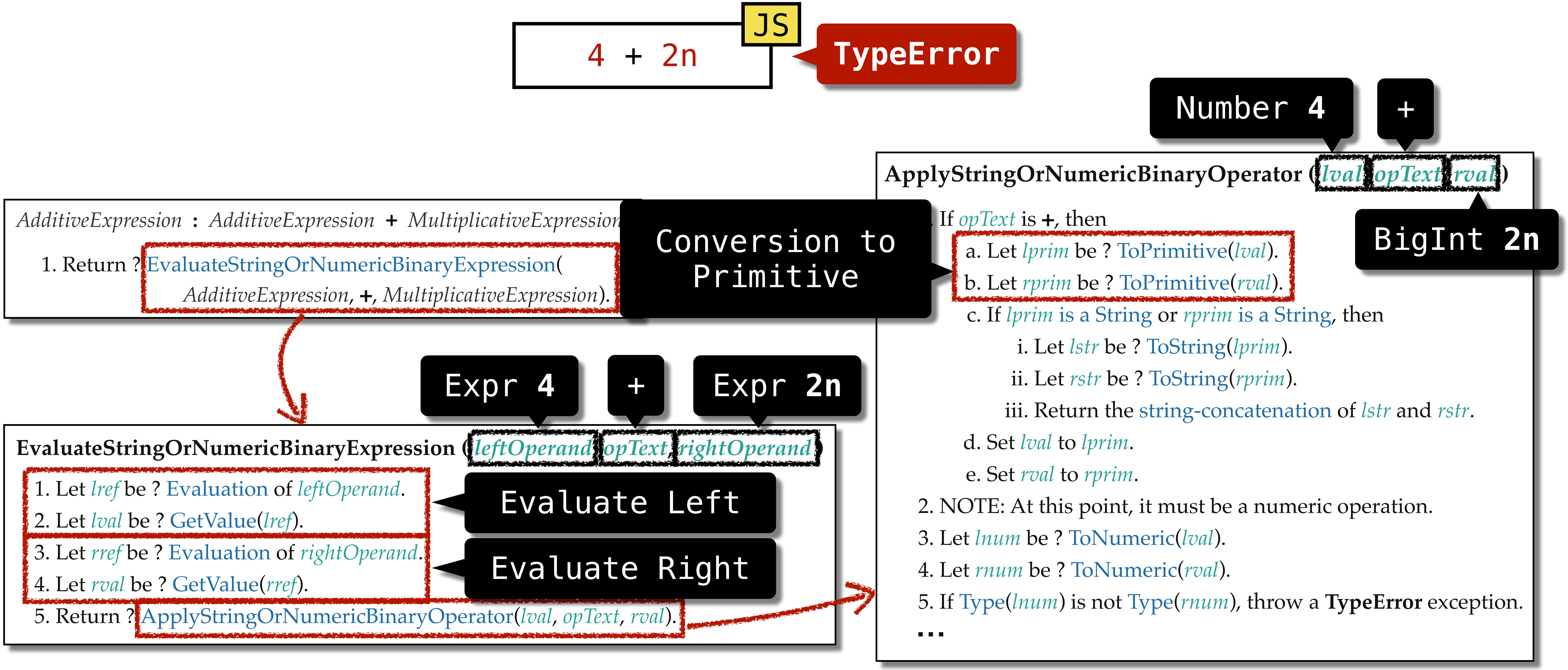
```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression

  1. Return ? EvaluateStringOrNumericBinaryExpression(
         AdditiveExpression, +, MultiplicativeExpression).
```

```
EvaluateStringOrNumericBinaryExpression (leftOperand, opText, rightOperand)
  1. Let lref be ? Evaluation of leftOperand.
  2. Let lval be ? GetValue(lref).
  3. Let rref be ? Evaluation of rightOperand.
  4. Let rval be ? GetValue(rref).
  5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

```
ApplyStringOrNumericBinaryOperator (lval, opText, rval)
  1. If opText is +, then
     a. Let lprim be ? ToPrimitive(lval).
     b. Let rprim be ? ToPrimitive(rval).
     c. If lprim is a String or rprim is a String, then
        i. Let lstr be ? ToString(lprim).
        ii. Let rstr be ? ToString(rprim).
        iii. Return the string-concatenation of lstr and rstr.
     d. Set lval to lprim.
     e. Set rval to rprim.
  2. NOTE: At this point, it must be a numeric operation.
  3. Let lnum be ? ToNumeric(lval).
  4. Let rnum be ? ToNumeric(rval).
  5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
  ...
```

Language Specification (ECMA-262) of JavaScript

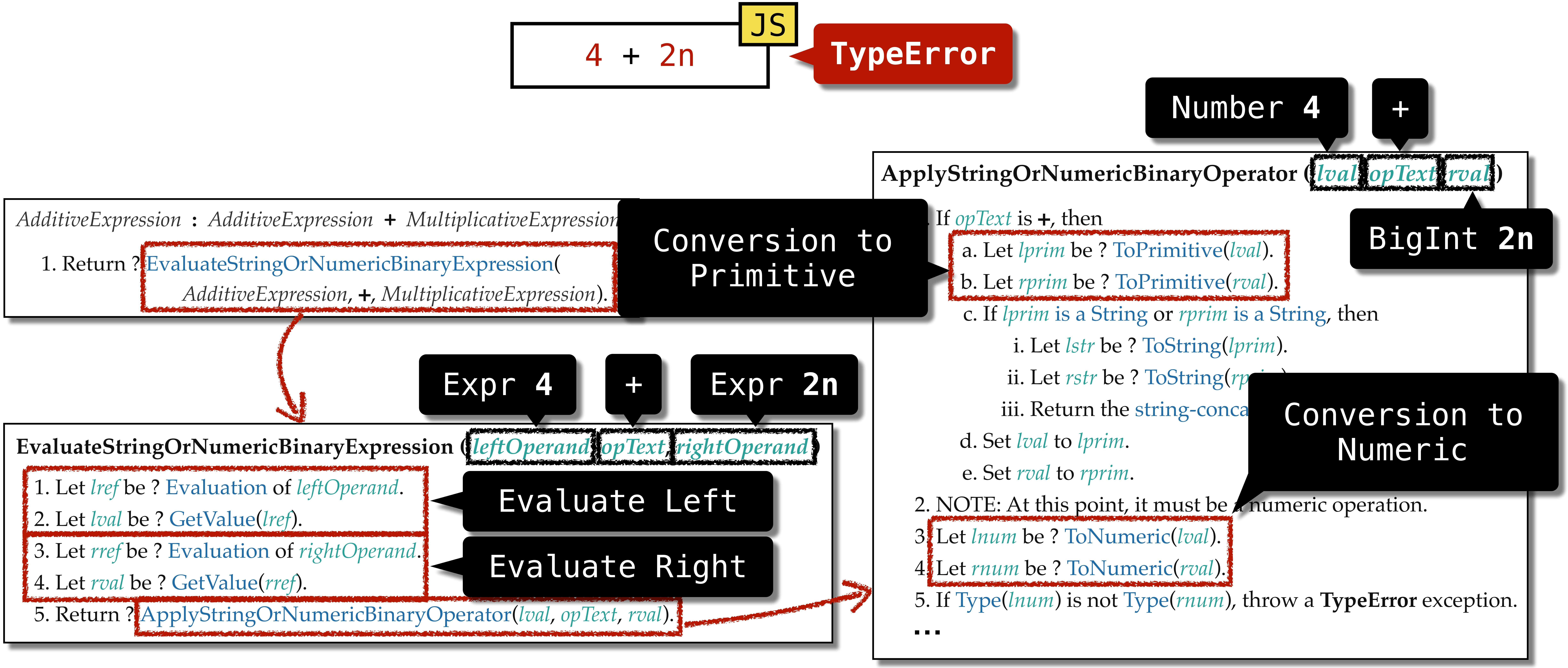


```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression
    1. Return ? EvaluateStringOrNumericBinaryExpression(
           AdditiveExpression, +, MultiplicativeExpression).
```

```
EvaluateStringOrNumericBinaryExpression (leftOperand, opText, rightOperand)
    1. Let lref be ? Evaluation of leftOperand.
    2. Let lval be ? GetValue(lref).
    3. Let rref be ? Evaluation of rightOperand.
    4. Let rval be ? GetValue(rref).
    5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

```
ApplyStringOrNumericBinaryOperator (lval, opText, rval)
    If opText is +, then
        a. Let lprim be ? ToPrimitive(lval).
        b. Let rprim be ? ToPrimitive(rval).
        c. If lprim is a String or rprim is a String, then
            i. Let lstr be ? ToString(lprim).
            ii. Let rstr be ? ToString(rprim).
            iii. Return the string-concatenation of lstr and rstr.
        d. Set lval to lprim.
        e. Set rval to rprim.
    2. NOTE: At this point, it must be a numeric operation.
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    ...
```

Language Specification (ECMA-262) of JavaScript



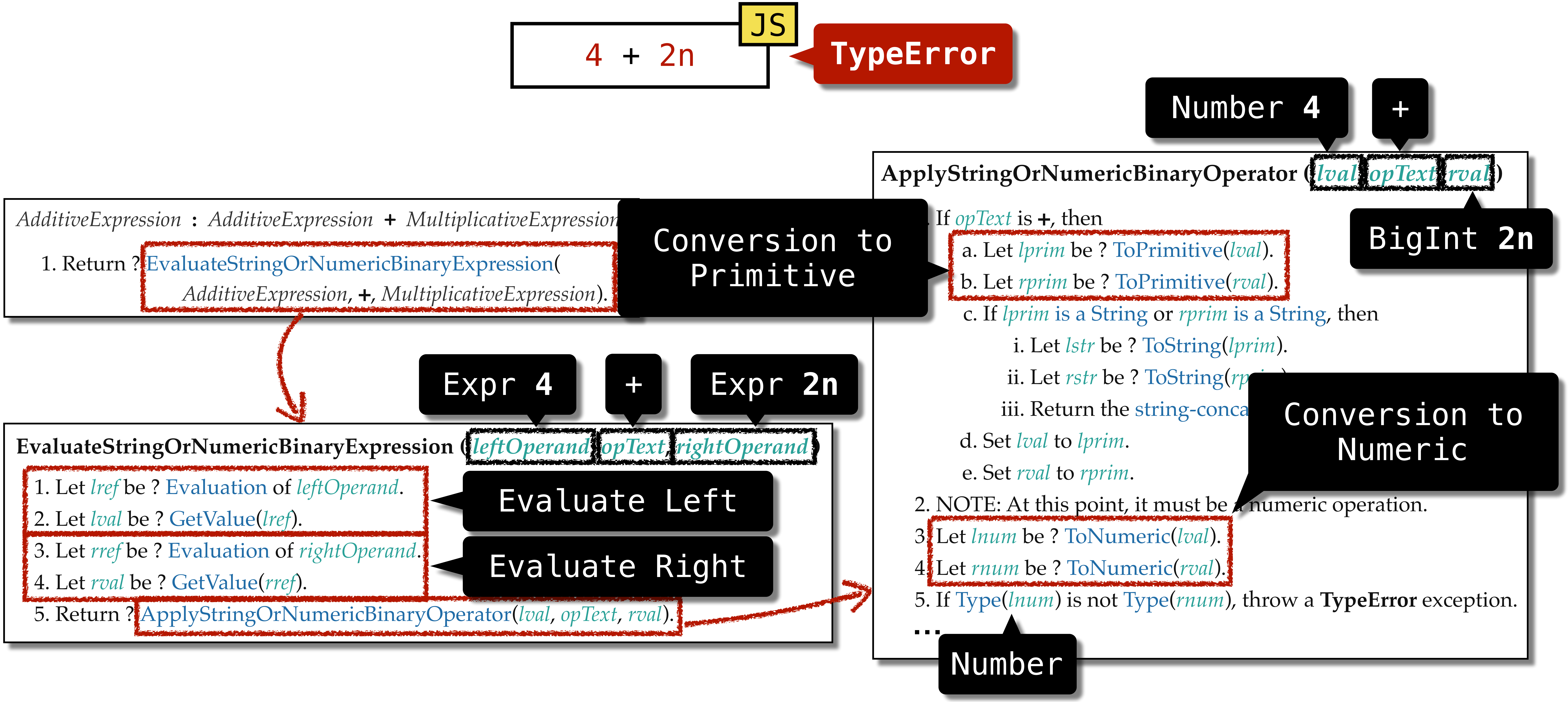
```
4 + 2n
```

```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression
  1. Return ? EvaluateStringOrNumericBinaryExpression(
         AdditiveExpression, +, MultiplicativeExpression).
```

```
EvaluateStringOrNumericBinaryExpression (leftOperand, opText, rightOperand)
  1. Let lref be ? Evaluation of leftOperand.
  2. Let lval be ? GetValue(lref).
  3. Let rref be ? Evaluation of rightOperand.
  4. Let rval be ? GetValue(rref).
  5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

```
ApplyStringOrNumericBinaryOperator (lval, opText, rval)
  If opText is +, then
    a. Let lprim be ? ToPrimitive(lval).
    b. Let rprim be ? ToPrimitive(rval).
    c. If lprim is a String or rprim is a String, then
         i. Let lstr be ? ToString(lprim).
        ii. Let rstr be ? ToString(rp
       iii. Return the string-conca
    d. Set lval to lprim.
    e. Set rval to rprim.
  2. NOTE: At this point, it must be a numeric operation.
  3. Let lnum be ? ToNumeric(lval).
  4. Let rnum be ? ToNumeric(rval).
  5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
  ...
```

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

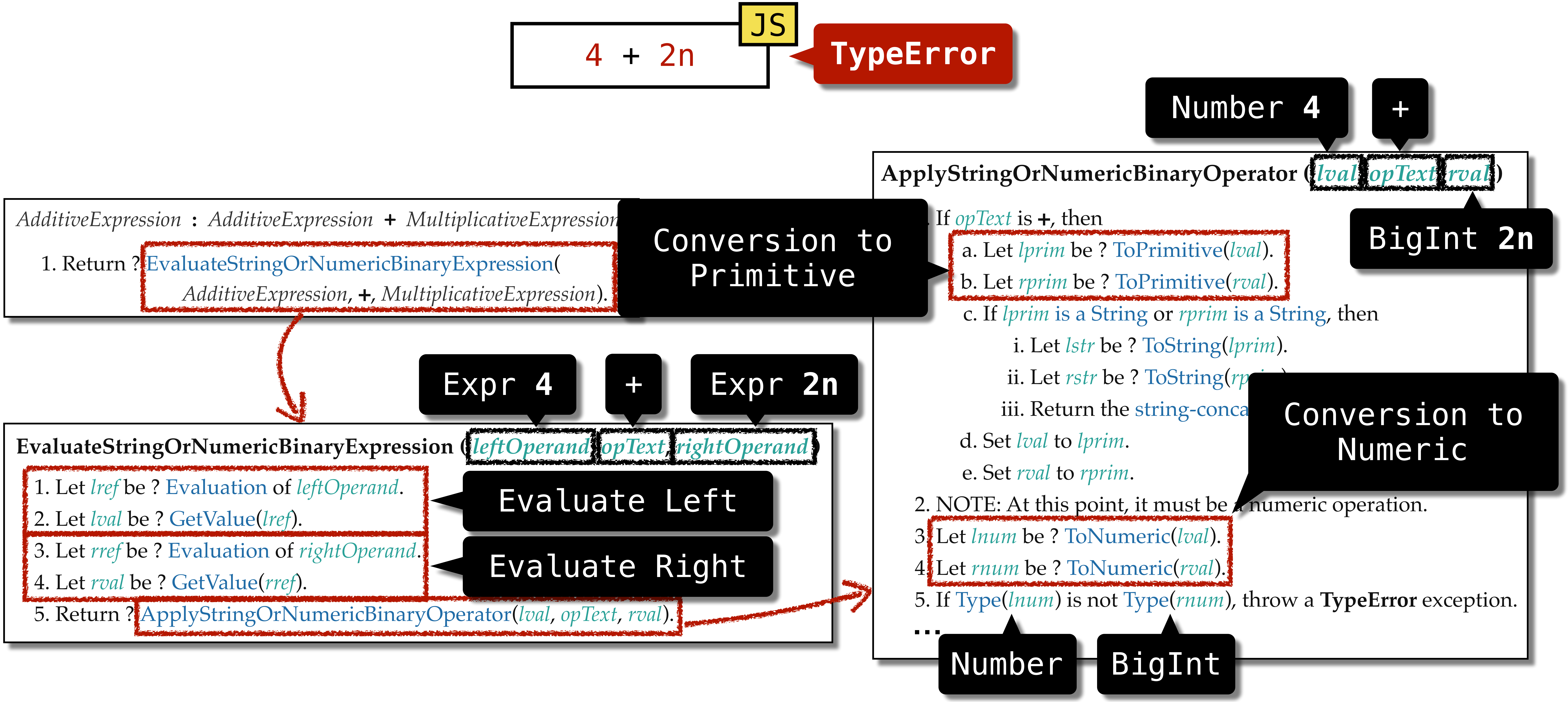
```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression
    1. Return ? EvaluateStringOrNumericBinaryExpression(
           AdditiveExpression, +, MultiplicativeExpression).
```

```
EvaluateStringOrNumericBinaryExpression (leftOperand, opText, rightOperand)
    1. Let lref be ? Evaluation of leftOperand.
    2. Let lval be ? GetValue(lref).
    3. Let rref be ? Evaluation of rightOperand.
    4. Let rval be ? GetValue(rref).
    5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

```
ApplyStringOrNumericBinaryOperator (lval, opText, rval)
    If opText is +, then
        a. Let lprim be ? ToPrimitive(lval).
        b. Let rprim be ? ToPrimitive(rval).
        c. If lprim is a String or rprim is a String, then
            i. Let lstr be ? ToString(lprim).
            ii. Let rstr be ? ToString(rp
            iii. Return the string-conca
        d. Set lval to lprim.
        e. Set rval to rprim.
    2. NOTE: At this point, it must be a numeric operation.
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    ...
```

PLRG

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

AdditiveExpression **:** *AdditiveExpression* **+** *MultiplicativeExpression*

1. Return ? EvaluateStringOrNumericBinaryExpression(*AdditiveExpression*, **+**, *MultiplicativeExpression*).

EvaluateStringOrNumericBinaryExpression (*leftOperand*, *opText*, *rightOperand*)

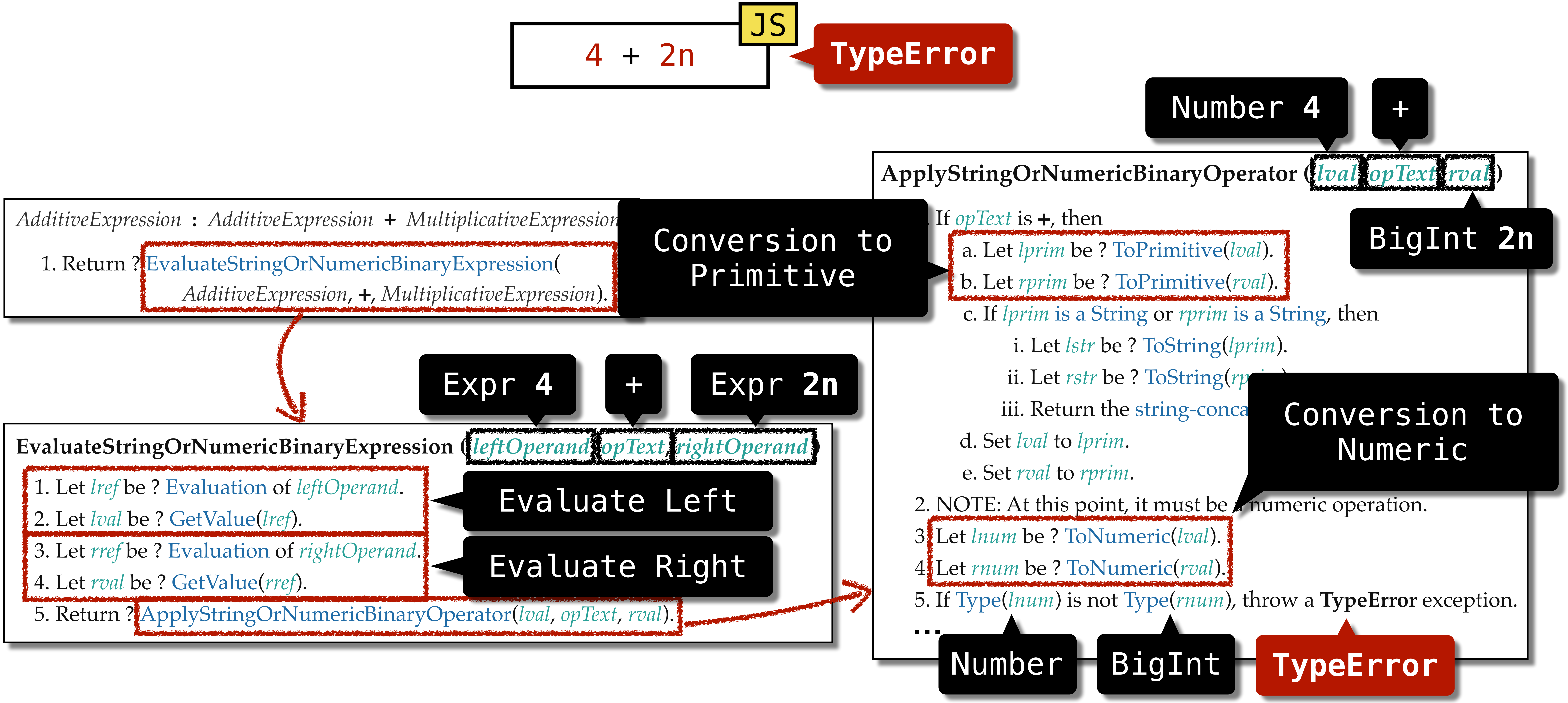
1. Let *lref* be ? Evaluation of *leftOperand*.
2. Let *lval* be ? GetValue(*lref*).
3. Let *rref* be ? Evaluation of *rightOperand*.
4. Let *rval* be ? GetValue(*rref*).
5. Return ? ApplyStringOrNumericBinaryOperator(*lval*, *opText*, *rval*).

ApplyStringOrNumericBinaryOperator (*lval*, *opText*, *rval*)

```
1. If opText is +, then
   a. Let lprim be ? ToPrimitive(lval).
   b. Let rprim be ? ToPrimitive(rval).
   c. If lprim is a String or rprim is a String, then
      i. Let lstr be ? ToString(lprim).
      ii. Let rstr be ? ToString(rp
      iii. Return the string-conca
   d. Set lval to lprim.
   e. Set rval to rprim.
2. NOTE: At this point, it must be a numeric operation.
3. Let lnum be ? ToNumeric(lval).
4. Let rnum be ? ToNumeric(rval).
5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
...
```

PLRG

Language Specification (ECMA-262) of JavaScript



```
4 + 2n
```

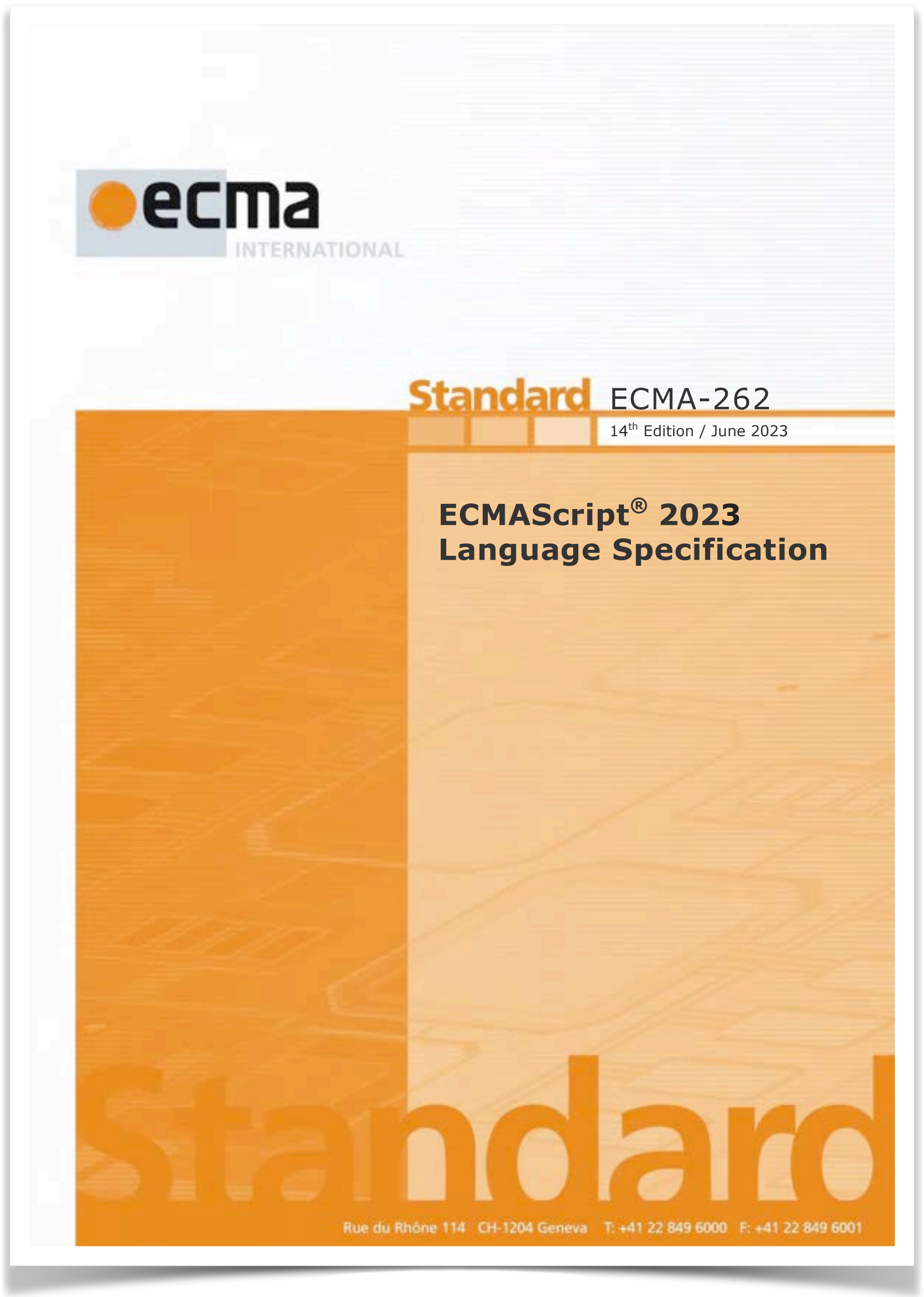
```
AdditiveExpression : AdditiveExpression + MultiplicativeExpression
    1. Return ? EvaluateStringOrNumericBinaryExpression(
           AdditiveExpression, +, MultiplicativeExpression).
```

```
EvaluateStringOrNumericBinaryExpression (leftOperand, opText, rightOperand)
    1. Let lref be ? Evaluation of leftOperand.
    2. Let lval be ? GetValue(lref).
    3. Let rref be ? Evaluation of rightOperand.
    4. Let rval be ? GetValue(rref).
    5. Return ? ApplyStringOrNumericBinaryOperator(lval, opText, rval).
```

```
ApplyStringOrNumericBinaryOperator (lval, opText, rval)
    If opText is +, then
        a. Let lprim be ? ToPrimitive(lval).
        b. Let rprim be ? ToPrimitive(rval).
        c. If lprim is a String or rprim is a String, then
            i. Let lstr be ? ToString(lprim).
            ii. Let rstr be ? ToString(rp
            iii. Return the string-conca
        d. Set lval to lprim.
        e. Set rval to rprim.
    2. NOTE: At this point, it must be a numeric operation.
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    ...
```

PLRG

Design and Implementation of JavaScript



ecma INTERNATIONAL

Standard ECMA-262

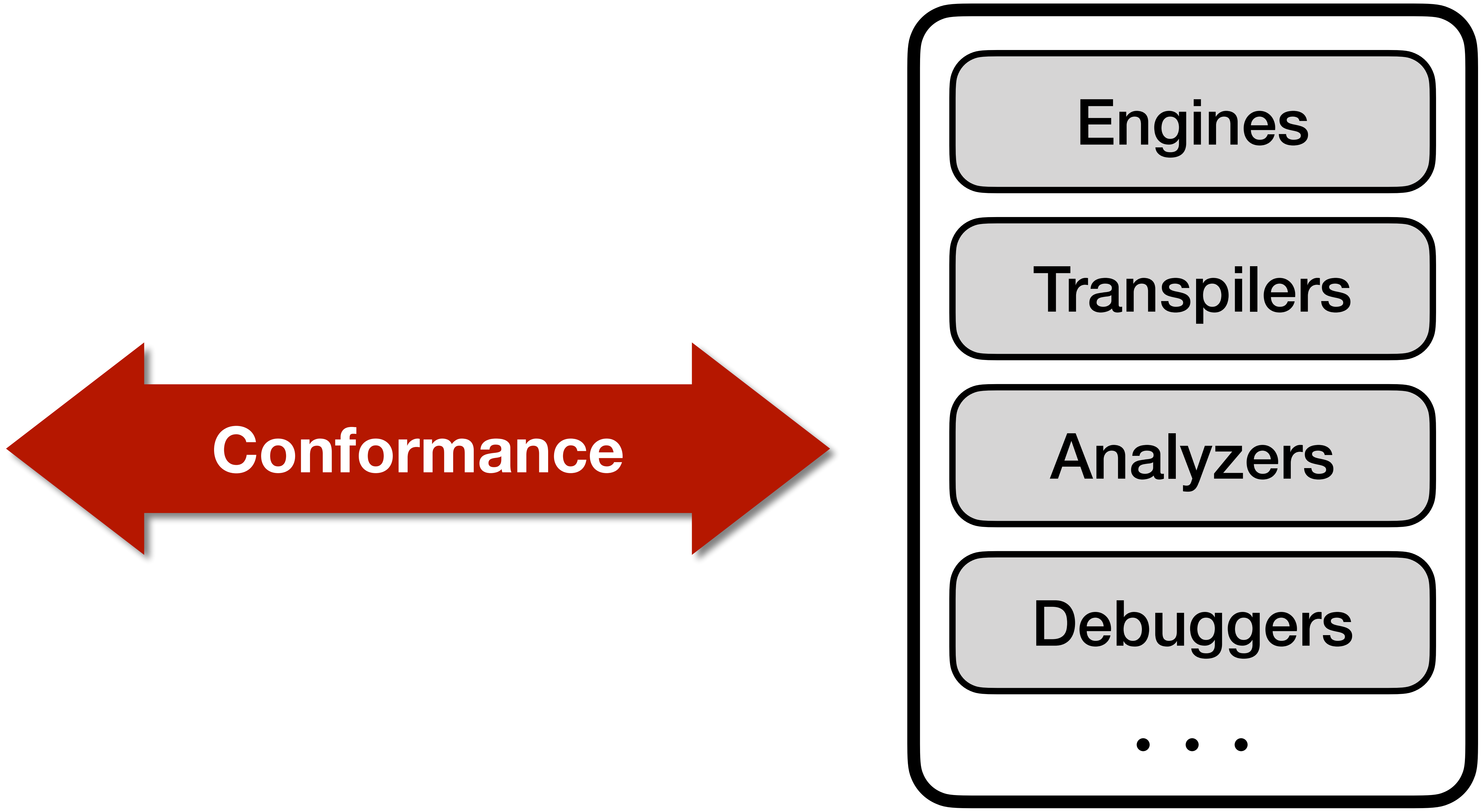
14th Edition / June 2023

ECMAScript® 2023
Language Specification

Standard

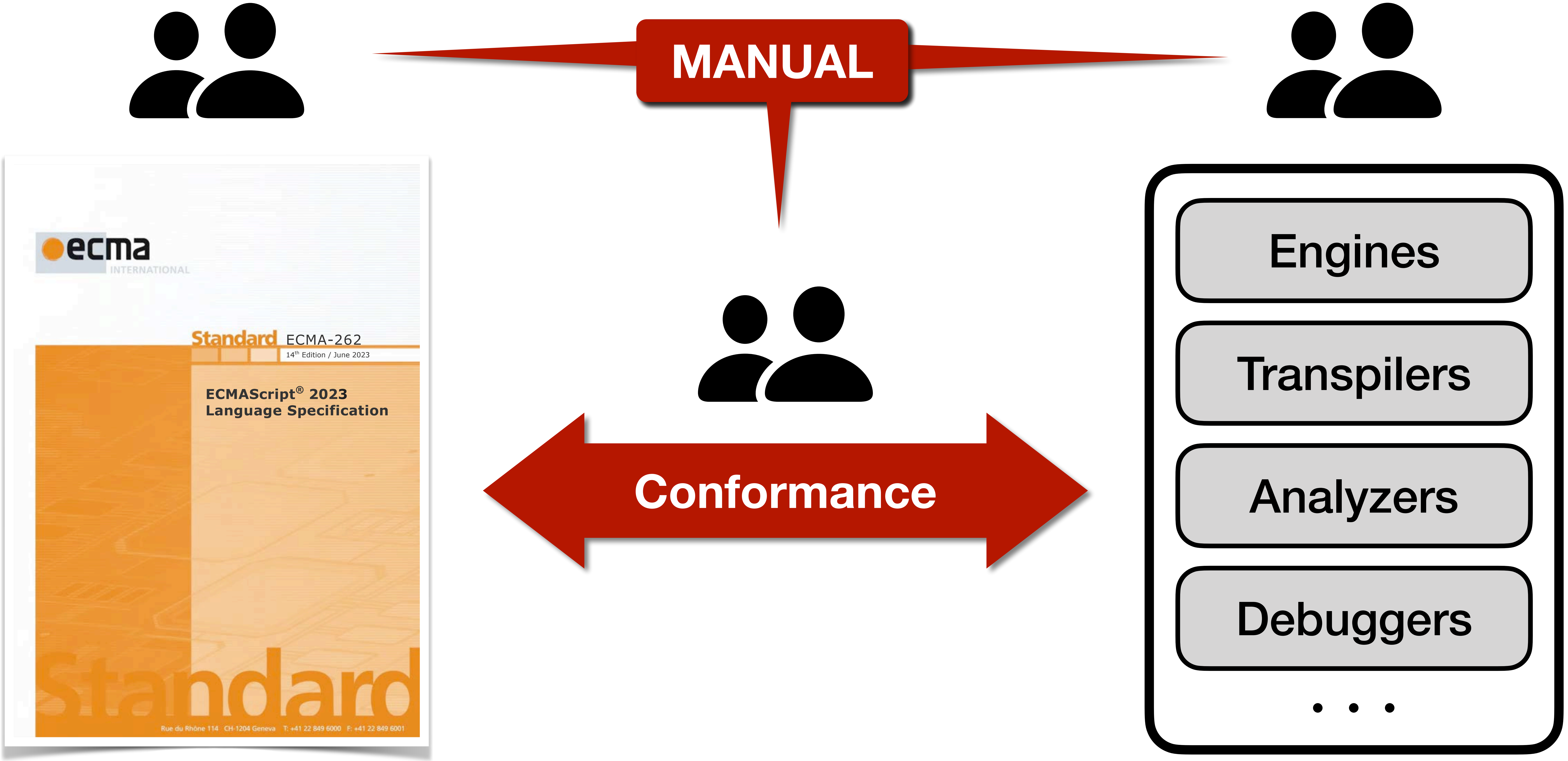
Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

ECMA-262
(JavaScript Spec.)



**JavaScript
Implementations**

Design and Implementation of JavaScript



MANUAL

ecma INTERNATIONAL

Standard ECMA-262

14th Edition / June 2023

ECMAScript® 2023 Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

Conformance

Engines

Transpilers

Analyzers

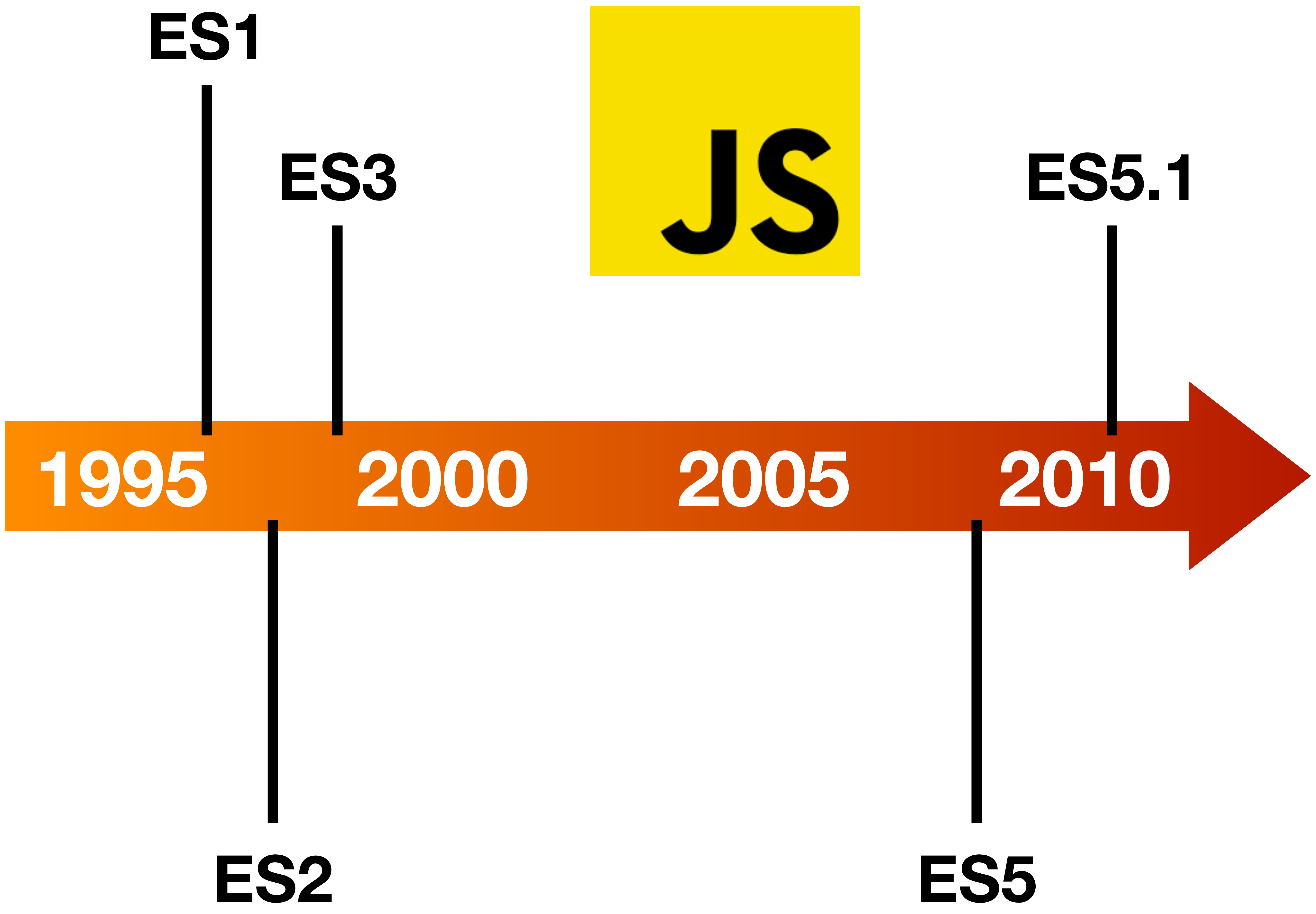
Debuggers

. . .

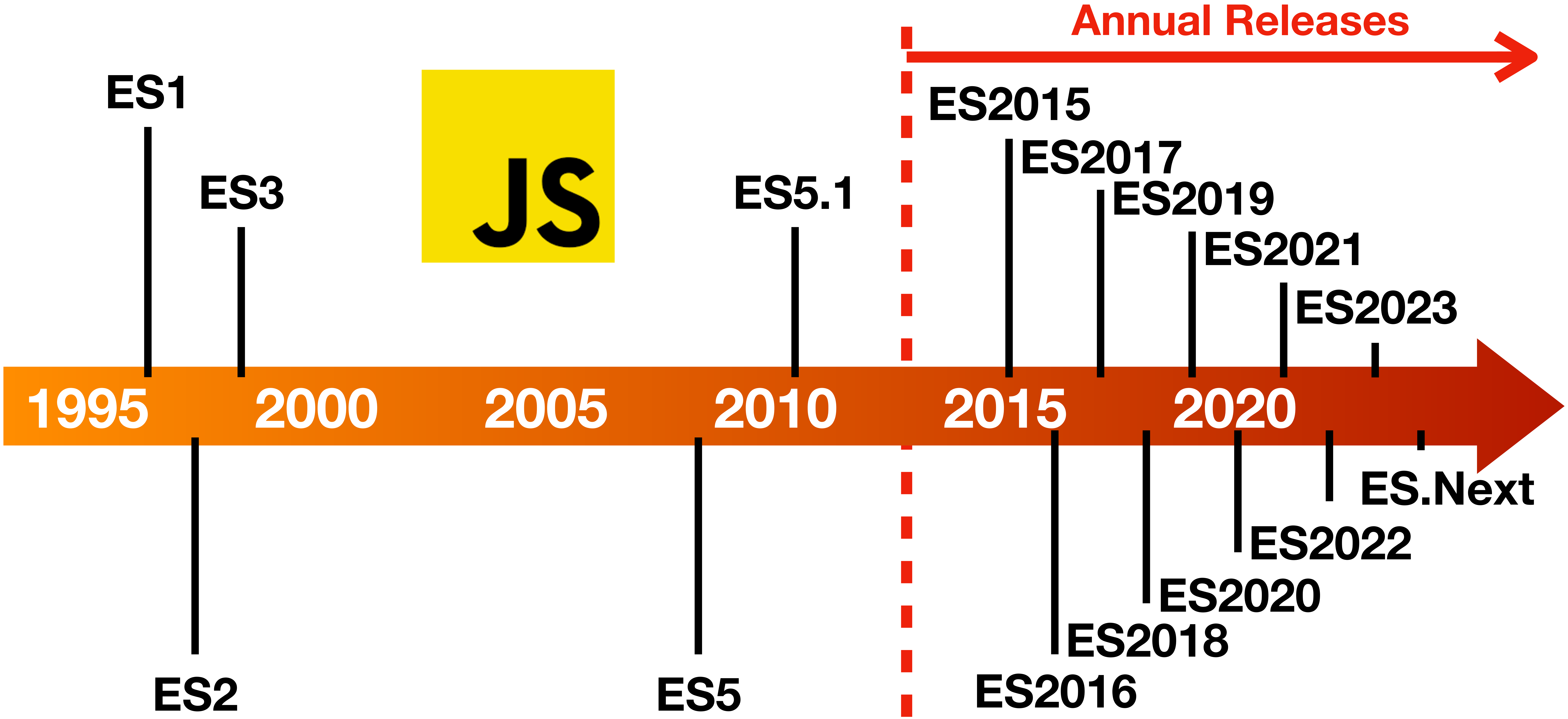
ECMA-262
(JavaScript Spec.)

JavaScript Implementations

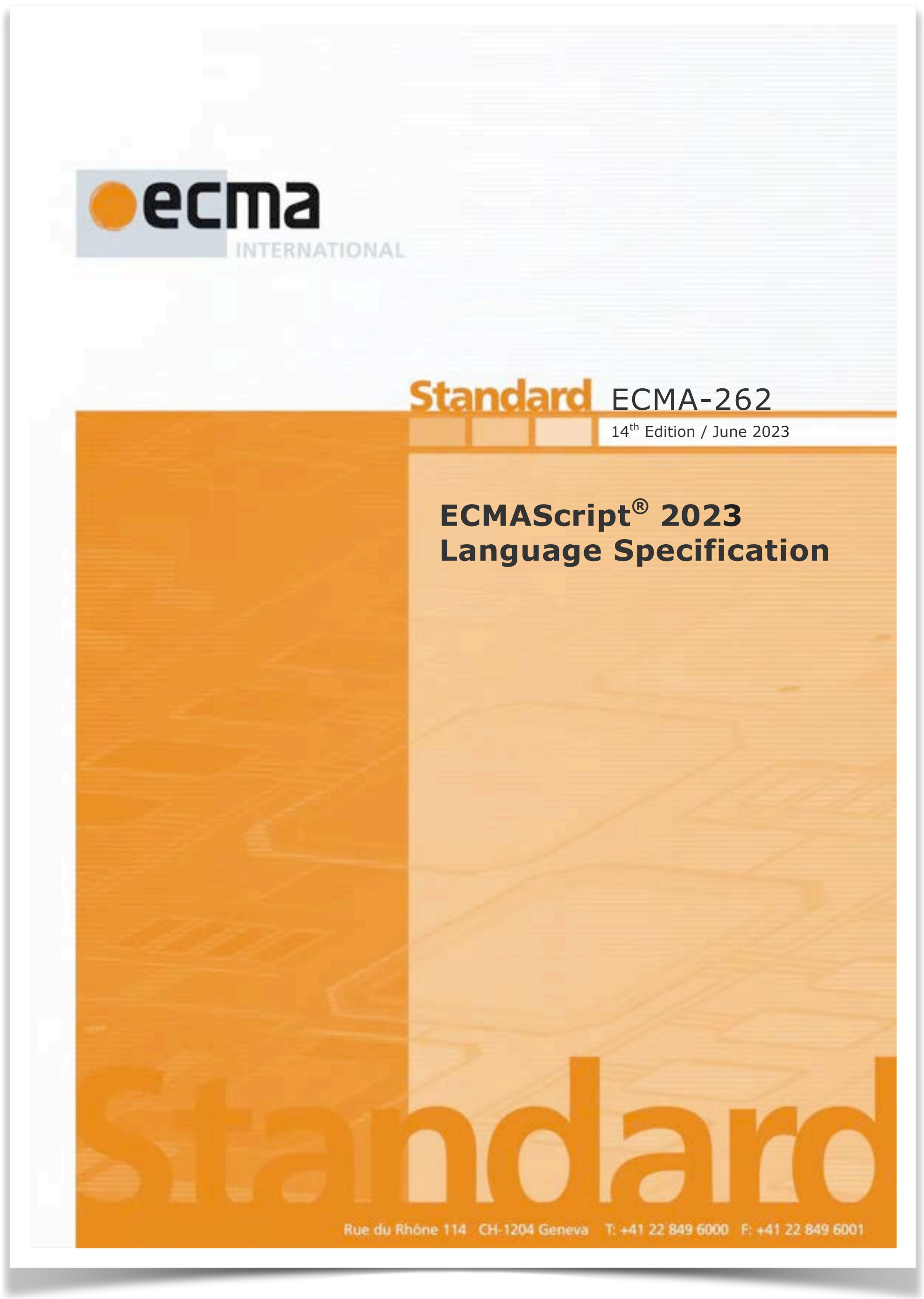
Problem - Fast Evolving JavaScript



Problem - Fast Evolving JavaScript



Solution - Mechanized Language Specification



ecma INTERNATIONAL

Standard ECMA-262

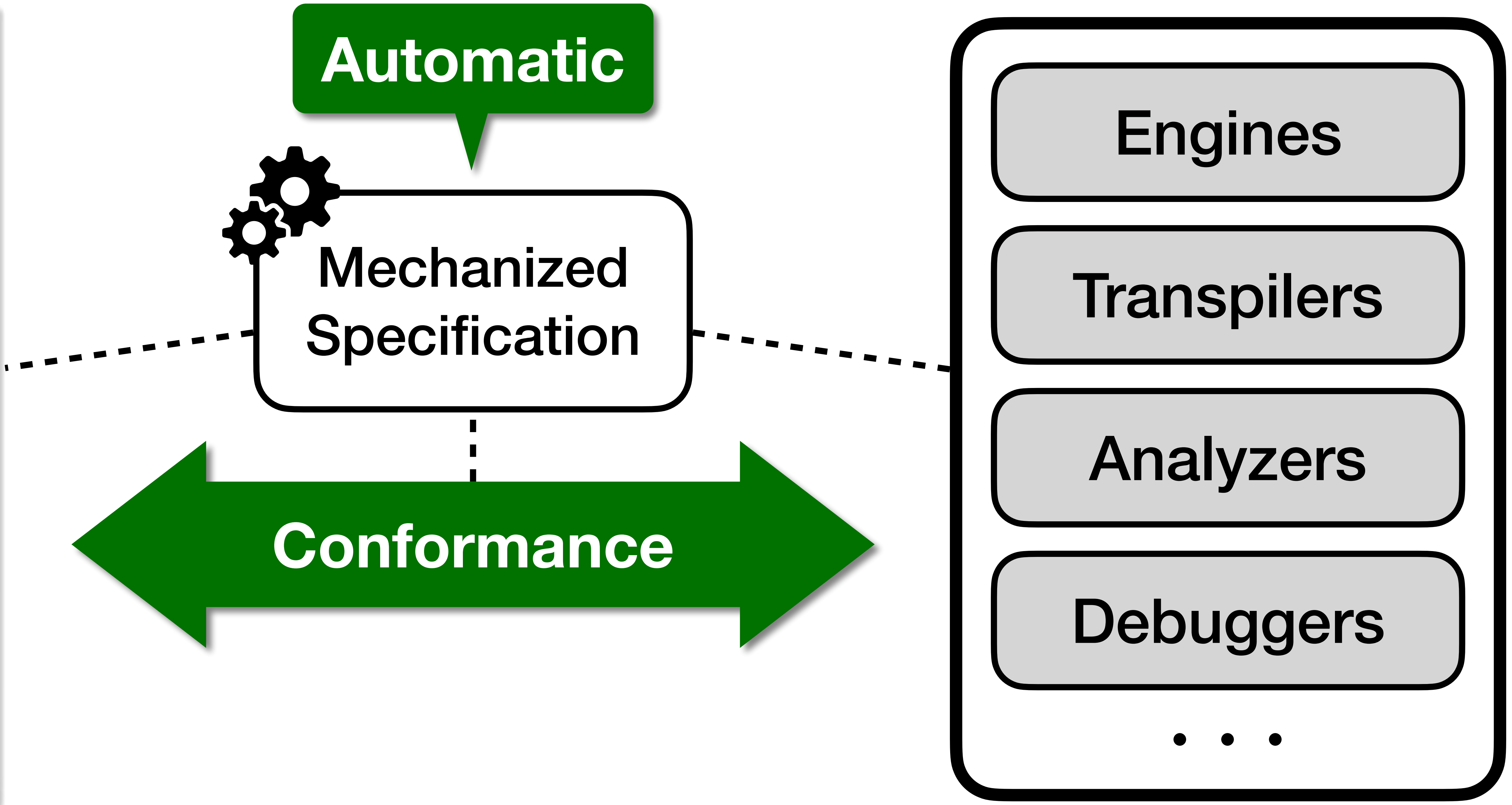
14th Edition / June 2023

ECMAScript® 2023 Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

ECMA-262
(JavaScript Spec.)



Automatic

Mechanized Specification

Conformance

Engines

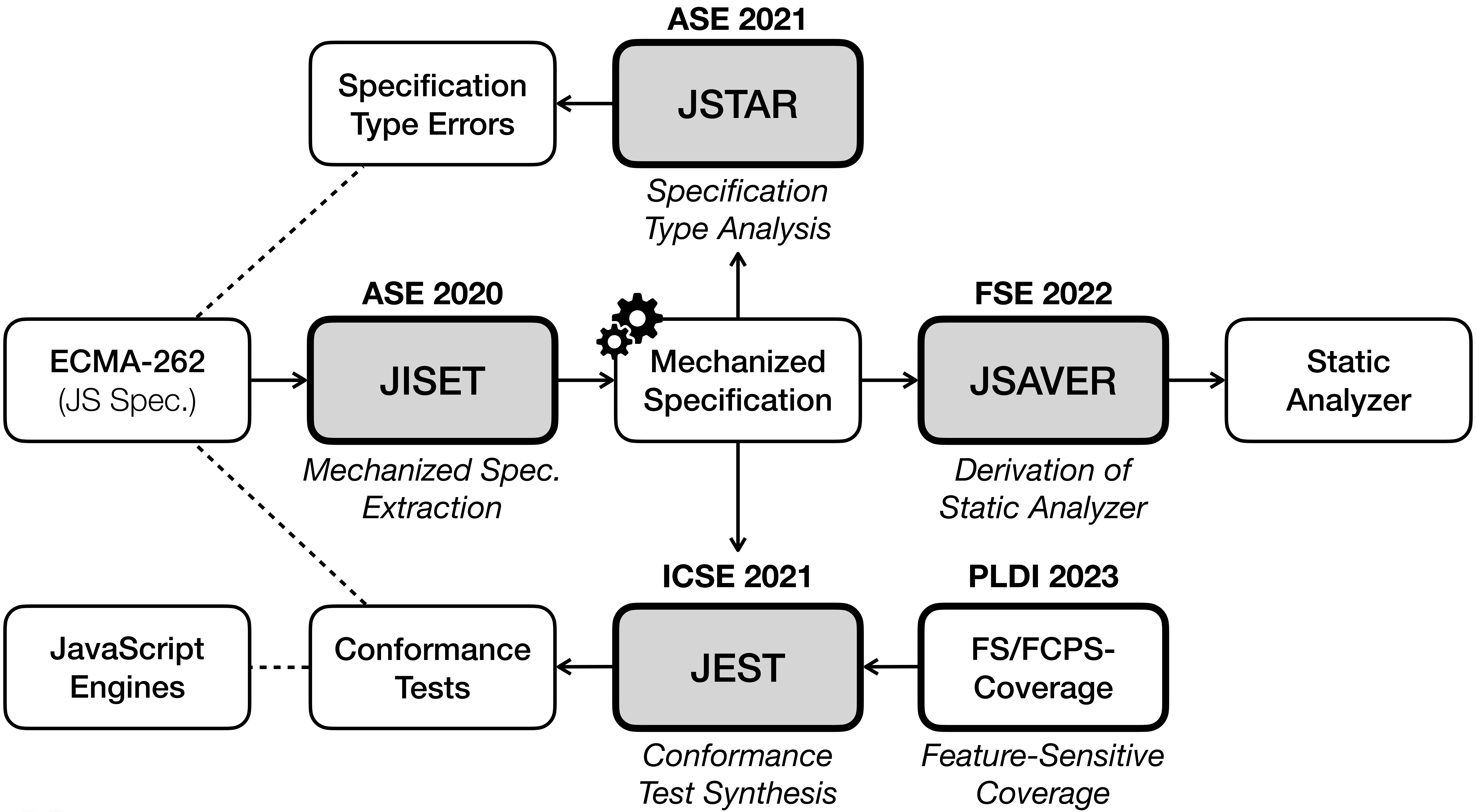
Transpilers

Analyzers

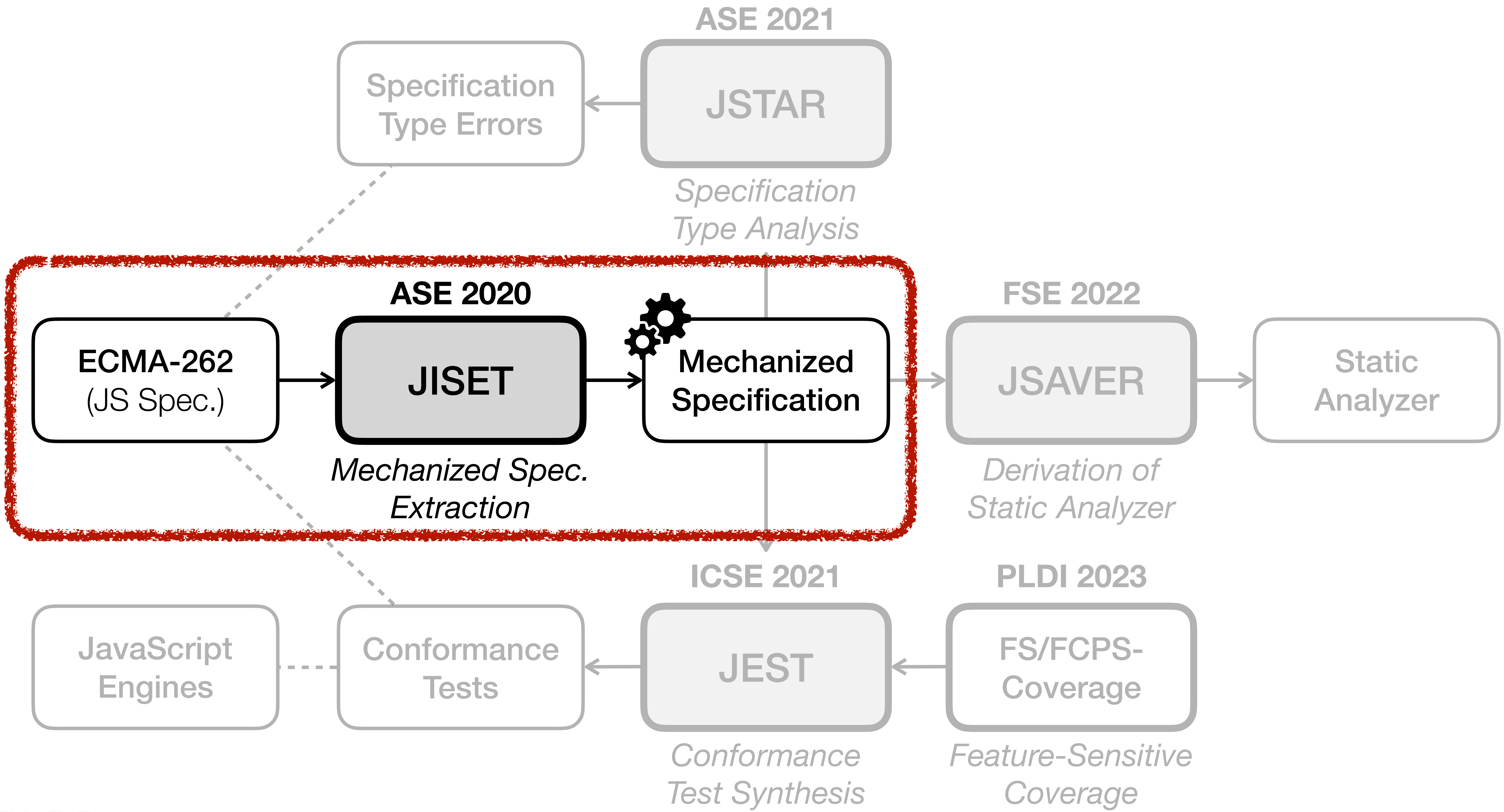
Debuggers

. . .

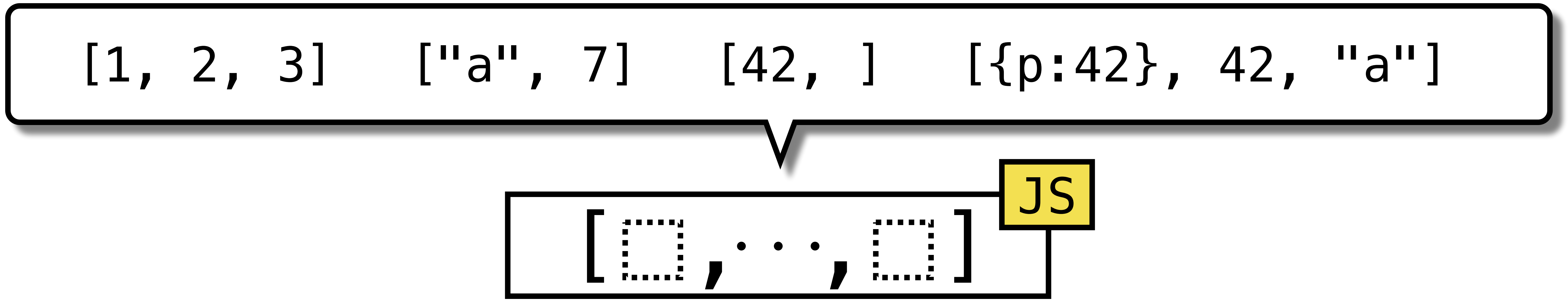
JavaScript
Implementations


ASE 2021
Specification Type Errors
JSTAR
Specification Type Analysis
ASE 2020
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
FSE 2022
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
ICSE 2021
PLDI 2023
JavaScript Engines
Conformance Tests
JEST
FS/FCPS-Coverage
Conformance Test Synthesis
Feature-Sensitive Coverage

[ASE 2020] J. Park et al., "JISET: JavaScript IR-based Semantics Extraction Toolchain"



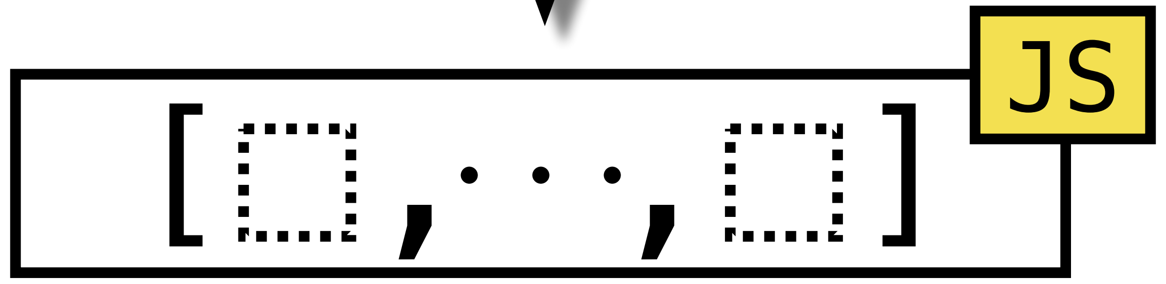
Language Specification (ECMA-262) of JavaScript



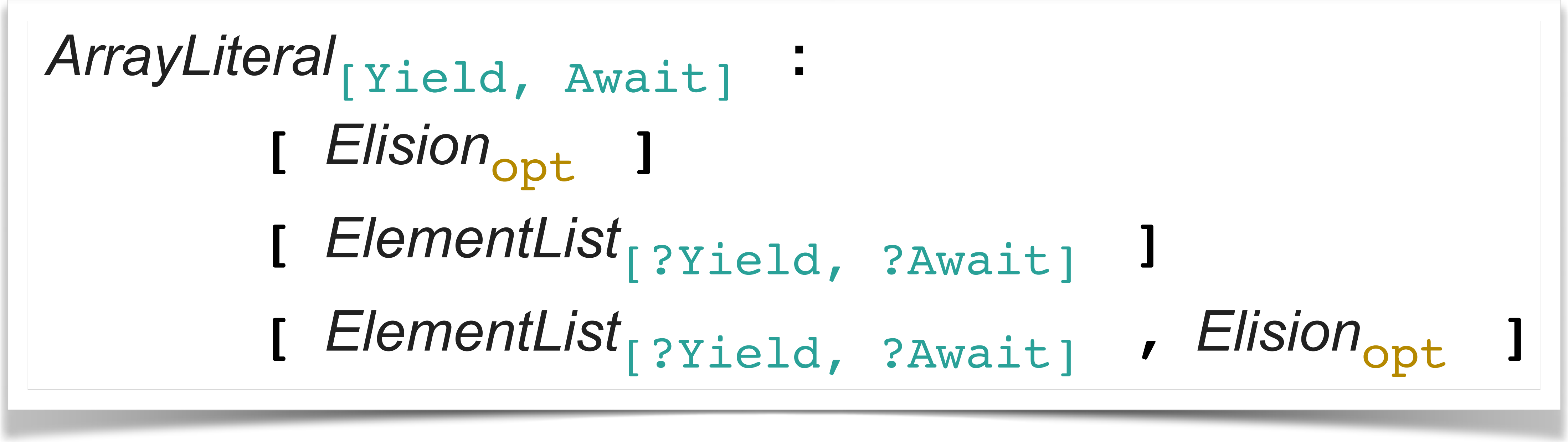
```
[1, 2, 3]   ["a", 7]   [42, ]   [{p:42}, 42, "a"]
```

Language Specification (ECMA-262) of JavaScript

```
[1, 2, 3]   ["a", 7]   [42, ]   [{p:42}, 42, "a"]
```



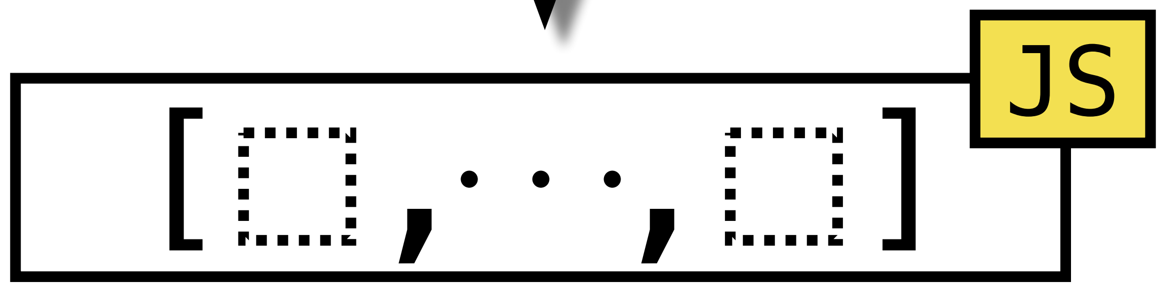
Syntax



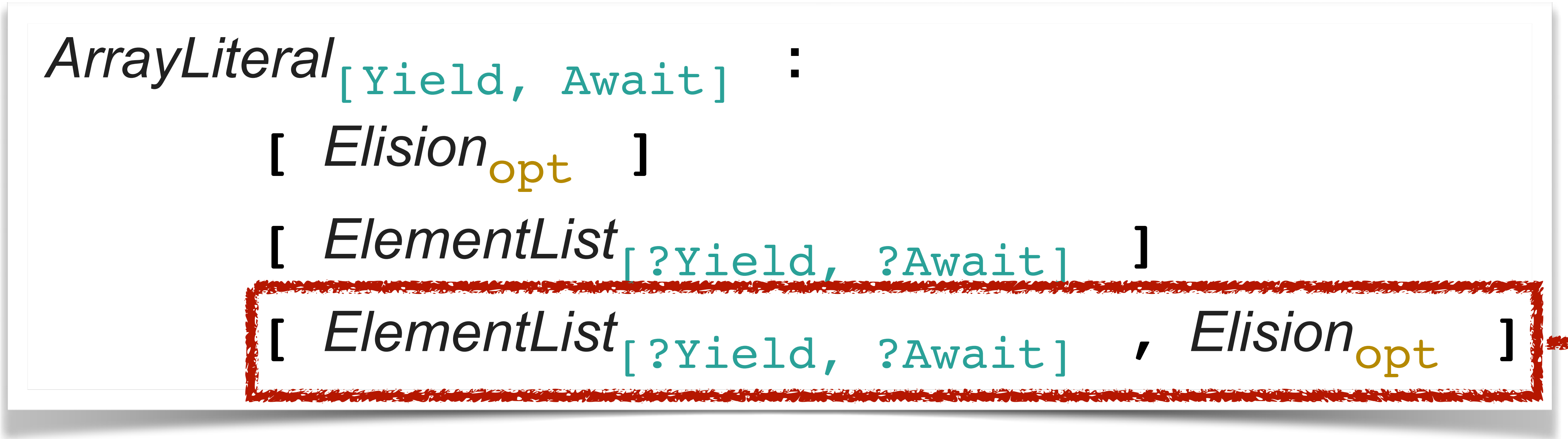
```
ArrayLiteral[Yield, Await] :
        [ Elision_opt ]
        [ ElementList[?Yield, ?Await] ]
        [ ElementList[?Yield, ?Await] , Elision_opt ]
```

Language Specification (ECMA-262) of JavaScript

```
[1, 2, 3]   ["a", 7]   [42, ]   [{p:42}, 42, "a"]
```

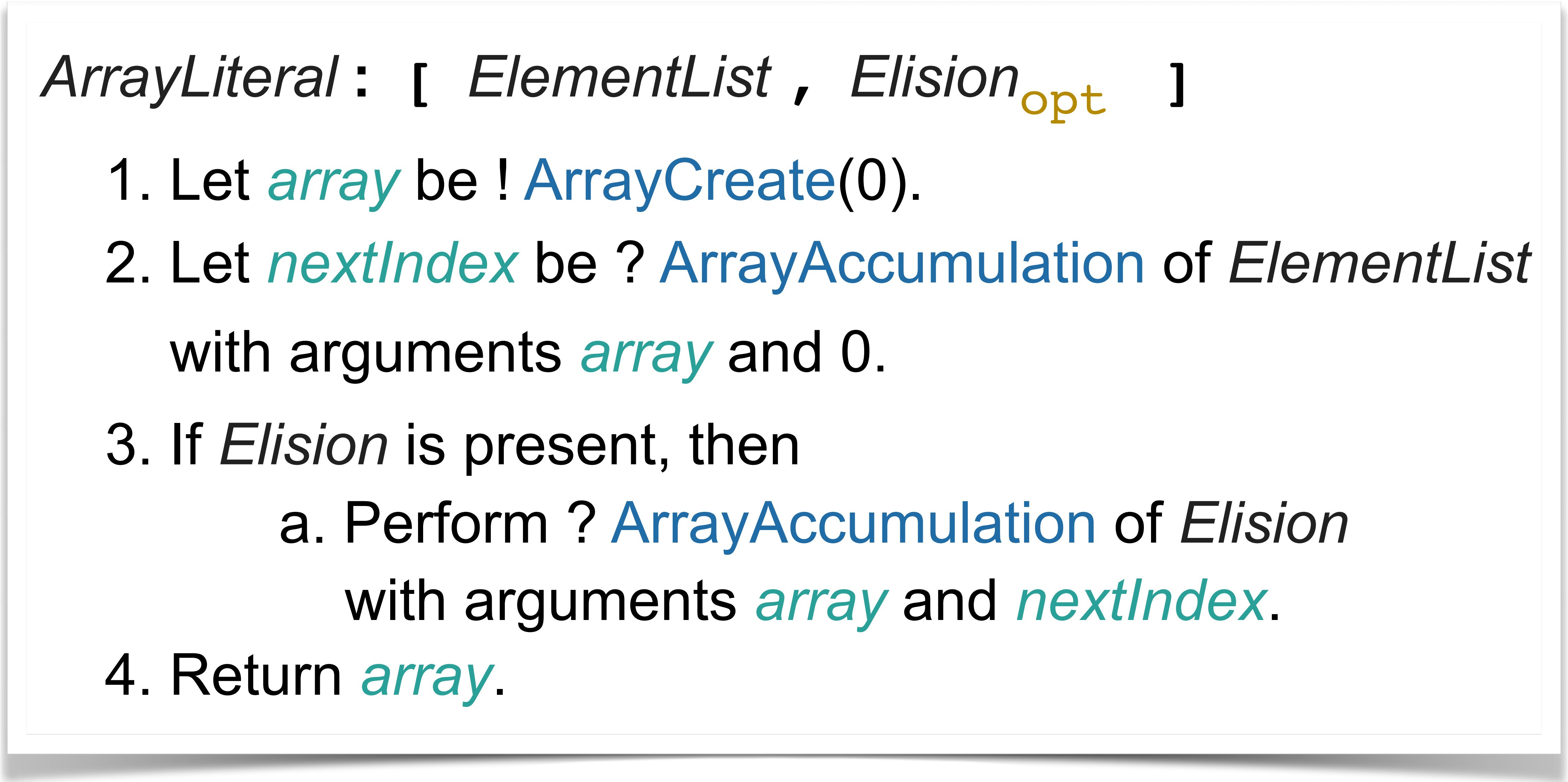


Syntax



```
ArrayLiteral[Yield, Await] :
        [ Elision_opt ]
        [ ElementList[?Yield, ?Await] ]
        [ ElementList[?Yield, ?Await] , Elision_opt ]
```

Semantics



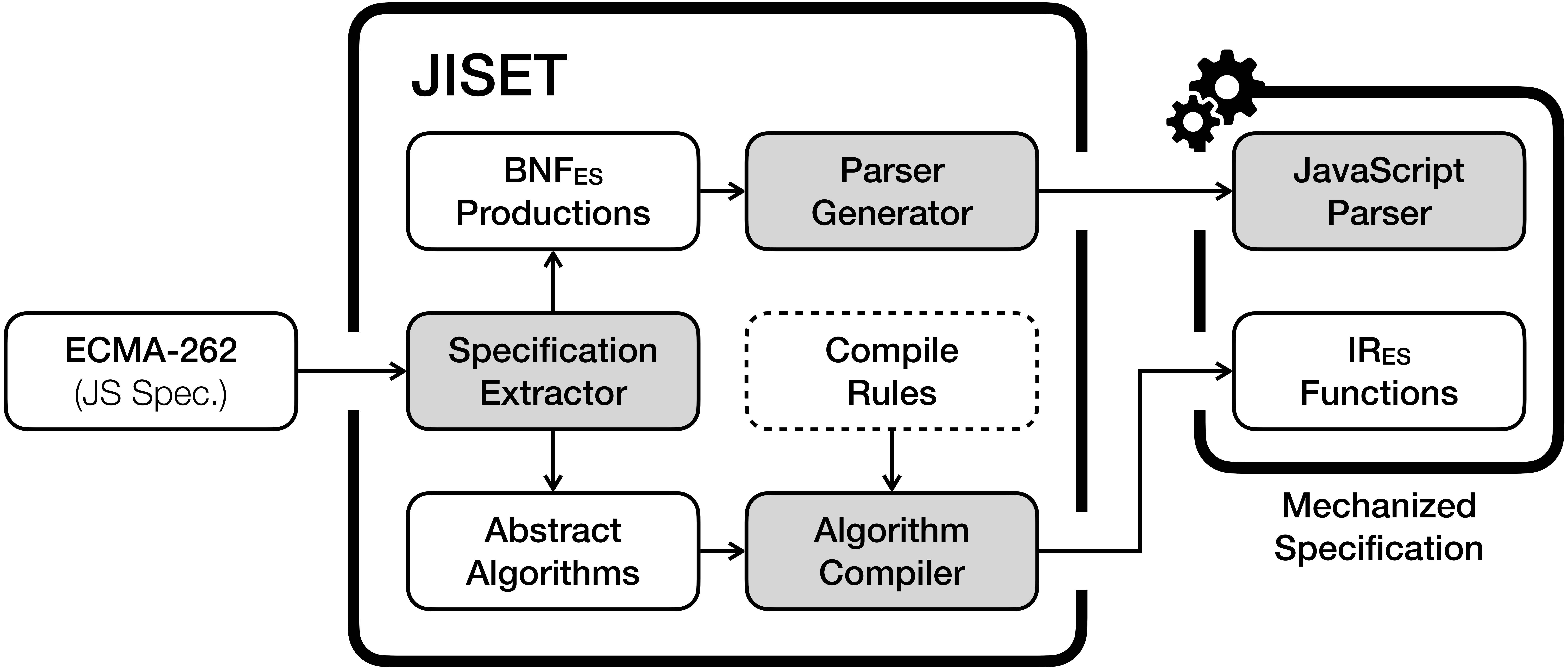
ArrayLiteral : `[` *ElementList* `,` *Elision*$_{opt}$ `]`

1. Let *array* be ! ArrayCreate(0).
2. Let *nextIndex* be ? ArrayAccumulation of *ElementList* with arguments *array* and 0.
3. If *Elision* is present, then
    a. Perform ? ArrayAccumulation of *Elision* with arguments *array* and *nextIndex*.
4. Return *array*.

PLRG

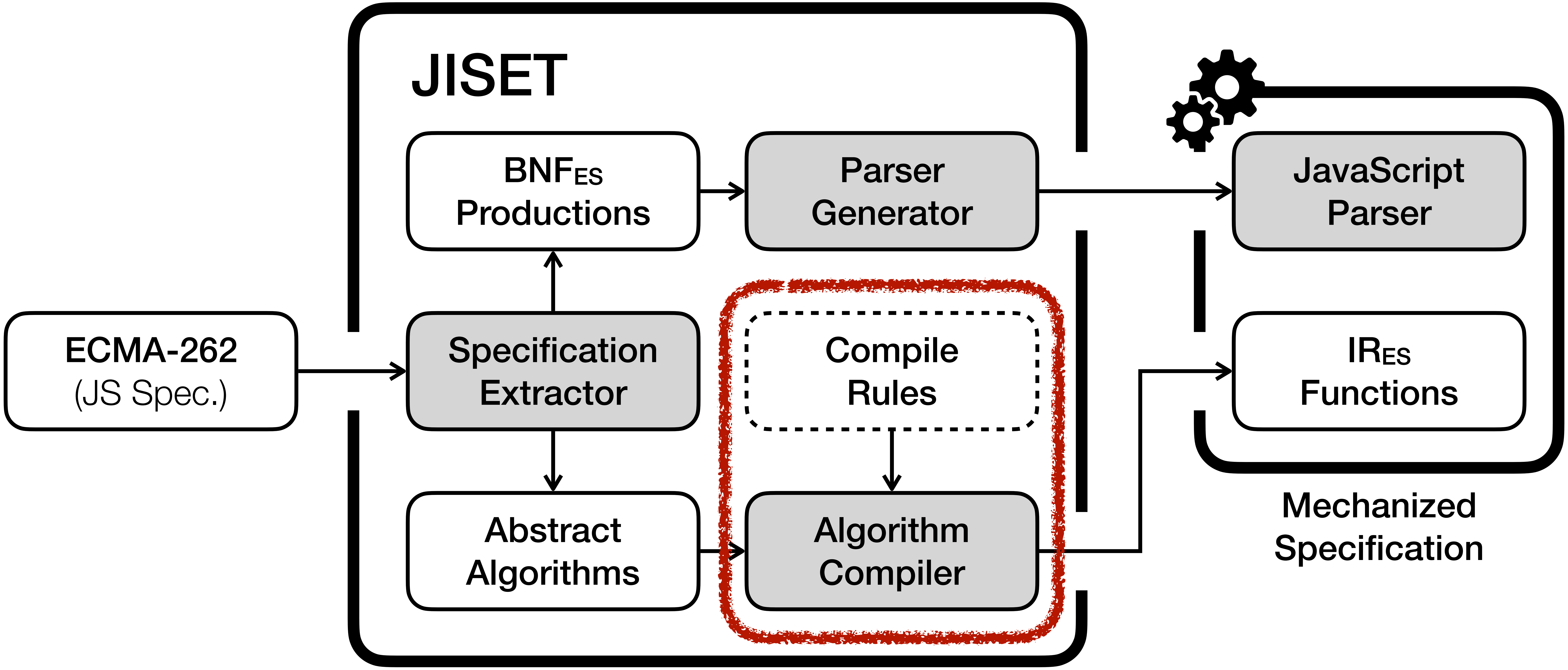
JISET

(**J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain)

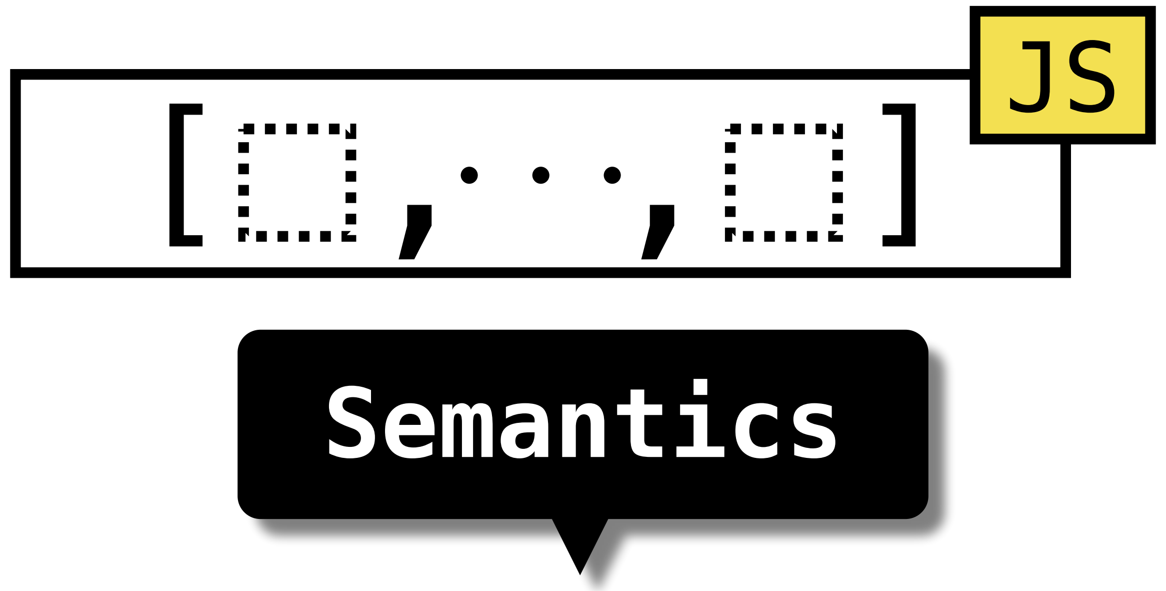


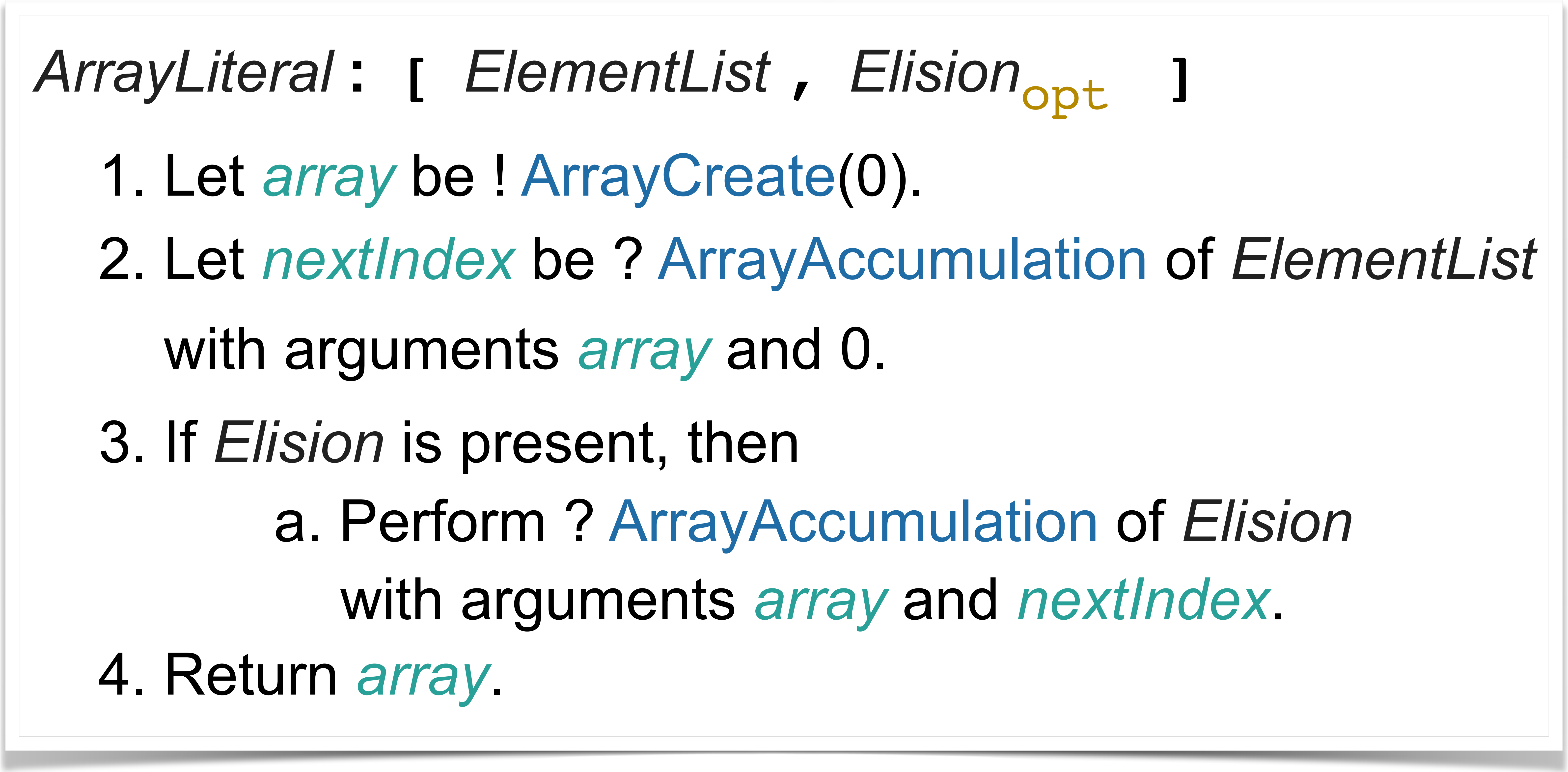
JISET

(**J**avaScript **I**R-based **S**emantics **E**xtraction **T**oolchain)



JISET - Patterns in Abstract Algorithms



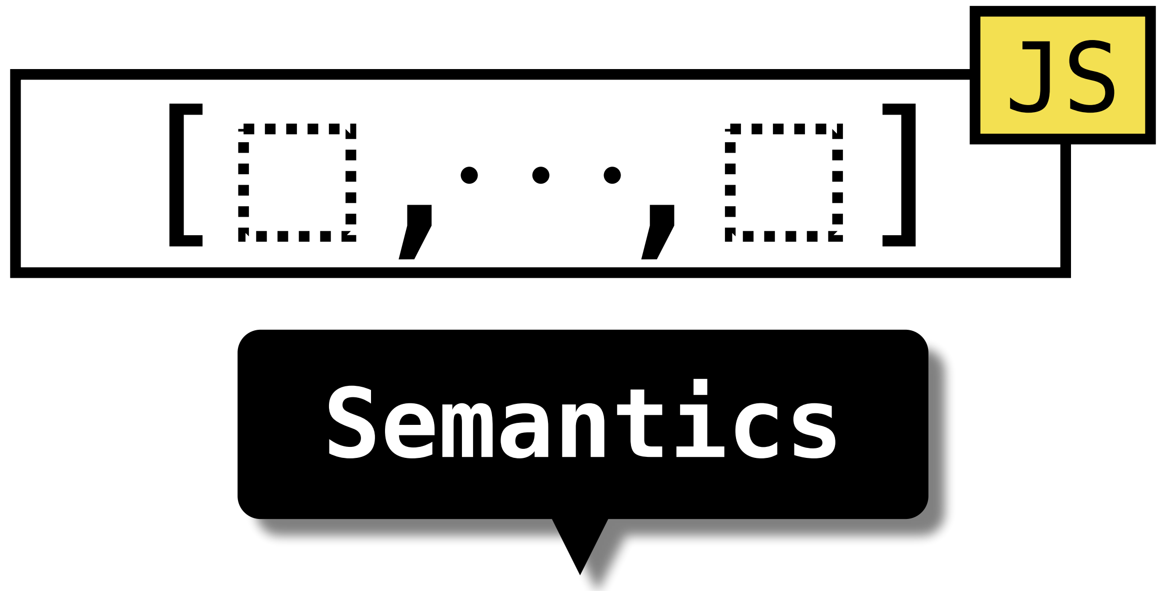


ArrayLiteral **:** **[** *ElementList* **,** *Elision*$_{opt}$ **]**

```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
      a. Perform ? ArrayAccumulation of Elision
         with arguments array and nextIndex.
4. Return array.
```

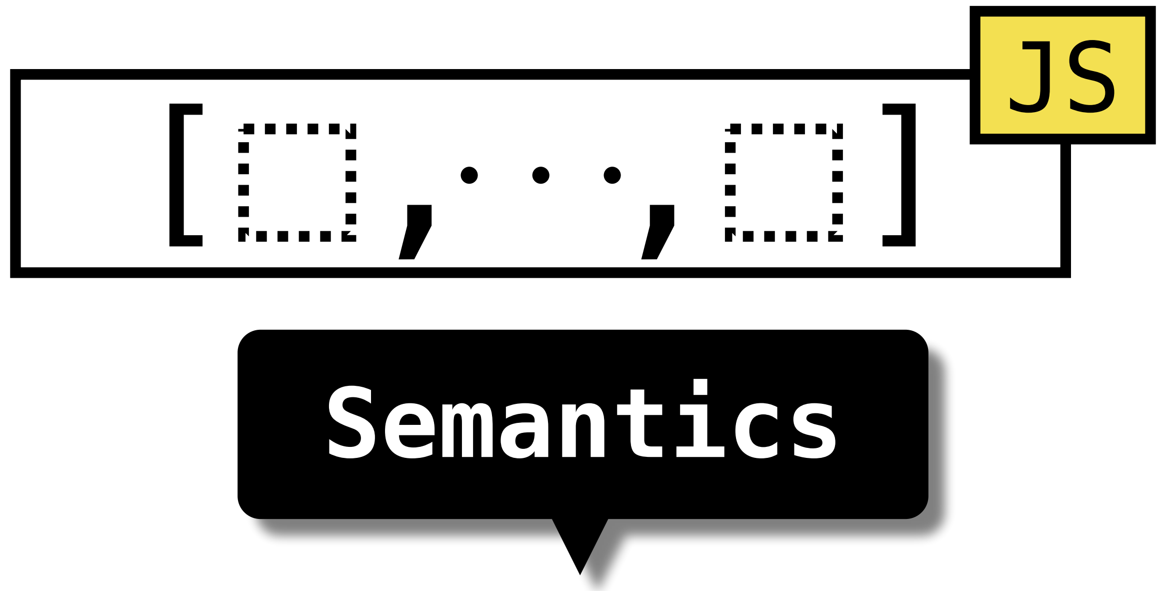
PLRG

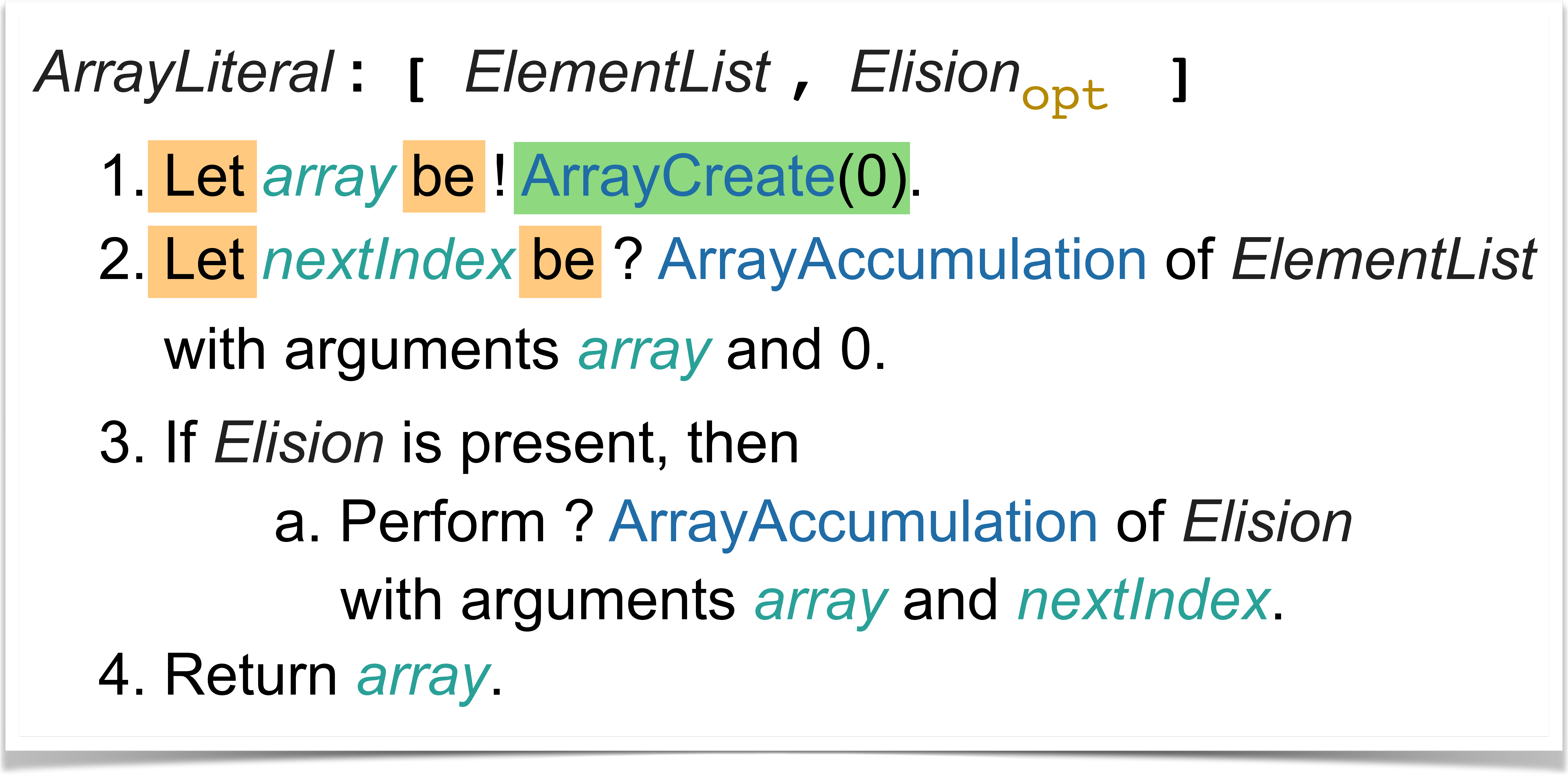
JISET - Patterns in Abstract Algorithms



```
ArrayLiteral : [ ElementList , Elision_opt ]
    1. Let array be ! ArrayCreate(0).
    2. Let nextIndex be ? ArrayAccumulation of ElementList
       with arguments array and 0.
    3. If Elision is present, then
         a. Perform ? ArrayAccumulation of Elision
            with arguments array and nextIndex.
    4. Return array.
```

JISET - Patterns in Abstract Algorithms

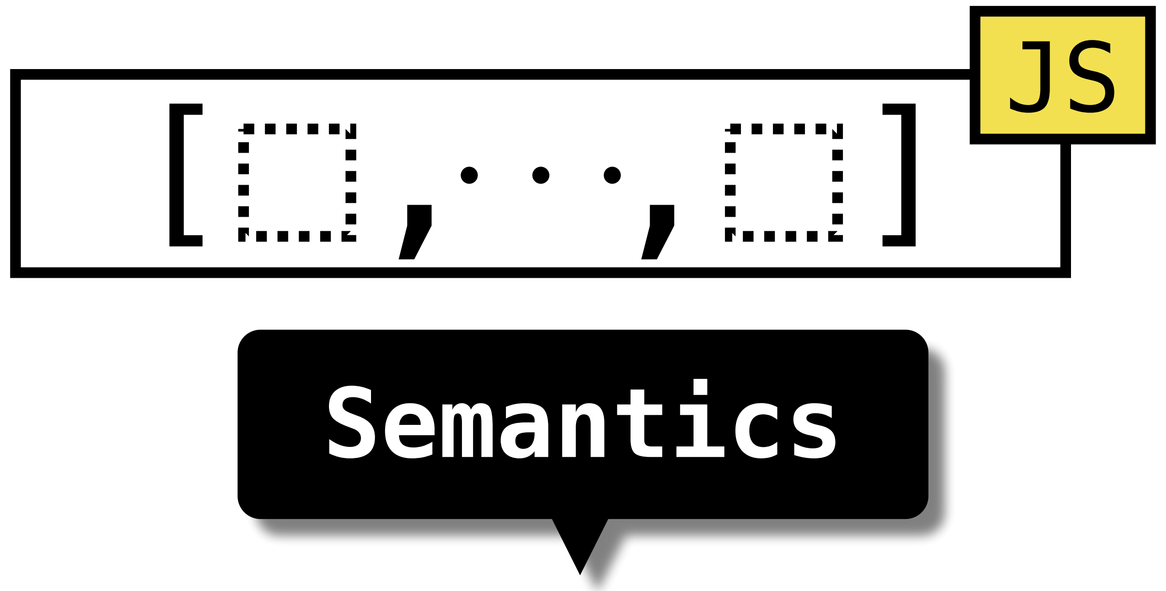


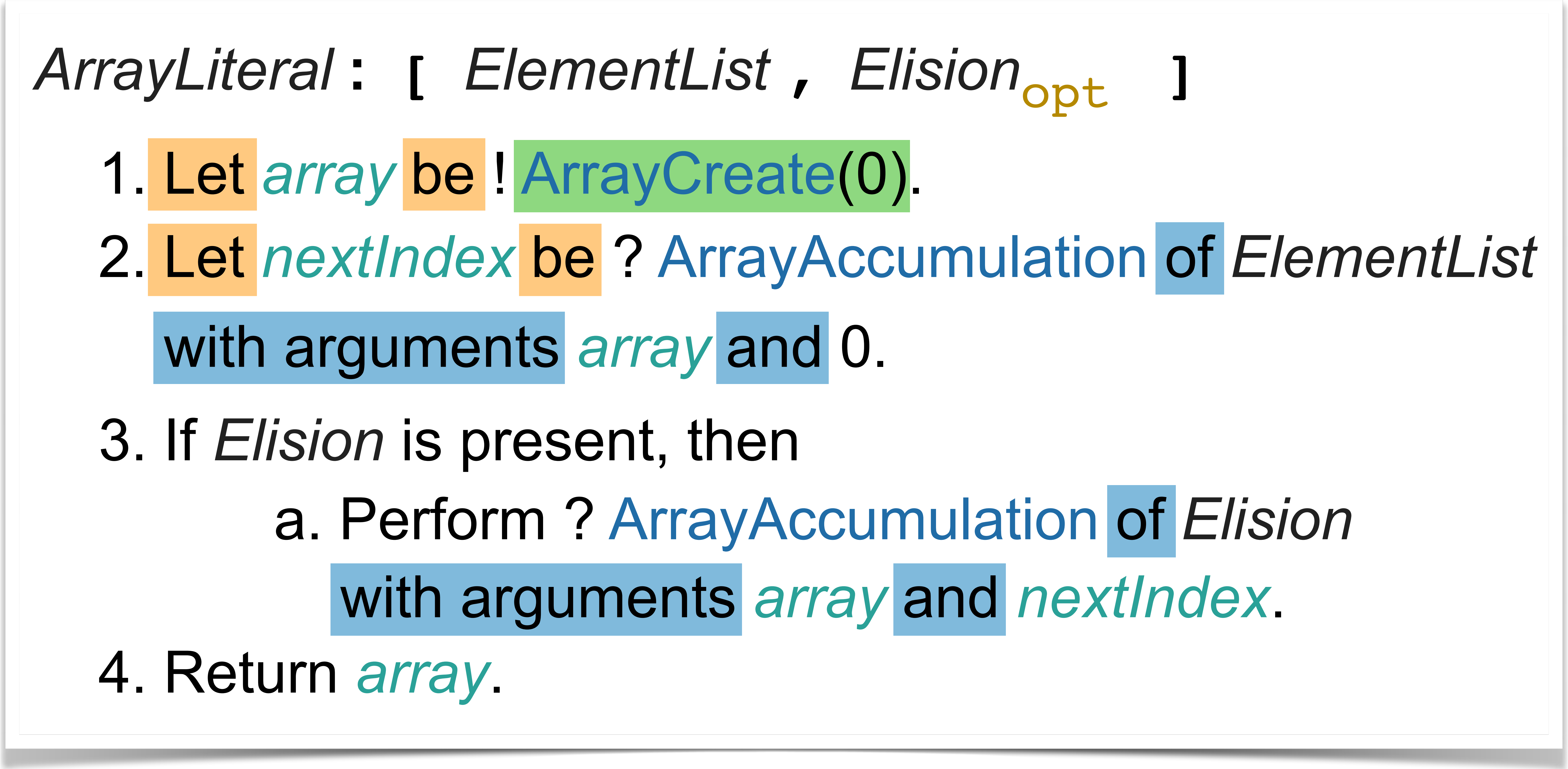


ArrayLiteral **:** **[** *ElementList* **,** *Elision*$_{opt}$ **]**

```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
    a. Perform ? ArrayAccumulation of Elision
       with arguments array and nextIndex.
4. Return array.
```

JISET - Patterns in Abstract Algorithms





ArrayLiteral : `[` *ElementList* `,` *Elision*$_{opt}$ `]`

```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
     a. Perform ? ArrayAccumulation of Elision
        with arguments array and nextIndex.
4. Return array.
```

JISET - Metalanguage for **ECMA-262**

(**IR$_{\mathrm{ES}}$** - **Intermediate Representation** for ECMA-262)

Programs	$\mathfrak{P} \ni P ::= f^*$
Functions	$\mathcal{F} \ni f ::= \texttt{syntax}^{?}\ \texttt{def}\ \texttt{x}(\texttt{x}^*)\ \{[\ell : i]^*\}$
Variables	$\mathcal{X} \ni \texttt{x}$
Labels	$\mathcal{L} \ni \ell$
Instructions	$\mathcal{I} \ni i ::= r := e \mid \texttt{x} := \{\} \mid \texttt{x} := e(e^*)$
	$\mid\ \texttt{if}\ e\ \ell\ \ell \mid \texttt{return}\ e$
Expressions	$\mathcal{E} \ni e ::= v^{\mathrm{p}} \mid \texttt{op}(e^*) \mid r$
References	$\mathcal{R} \ni r ::= \texttt{x} \mid e[e] \mid e[e]_{\texttt{js}}$
	$\vdots$
Values	$v \in \mathbb{V} = \mathbb{A} \uplus \mathbb{V}^{\mathrm{p}} \uplus \mathbb{T} \uplus \mathcal{F}$
Primitive Values	$v^{\mathrm{p}} \in \mathbb{V}^{\mathrm{p}} = \mathbb{V}_{\mathrm{bool}} \uplus \mathbb{V}_{\mathrm{int}} \uplus \mathbb{V}_{\mathrm{str}} \uplus \cdots$
JS ASTs	$t \in \mathbb{T}$

JISET - Metalanguage for ECMA-262

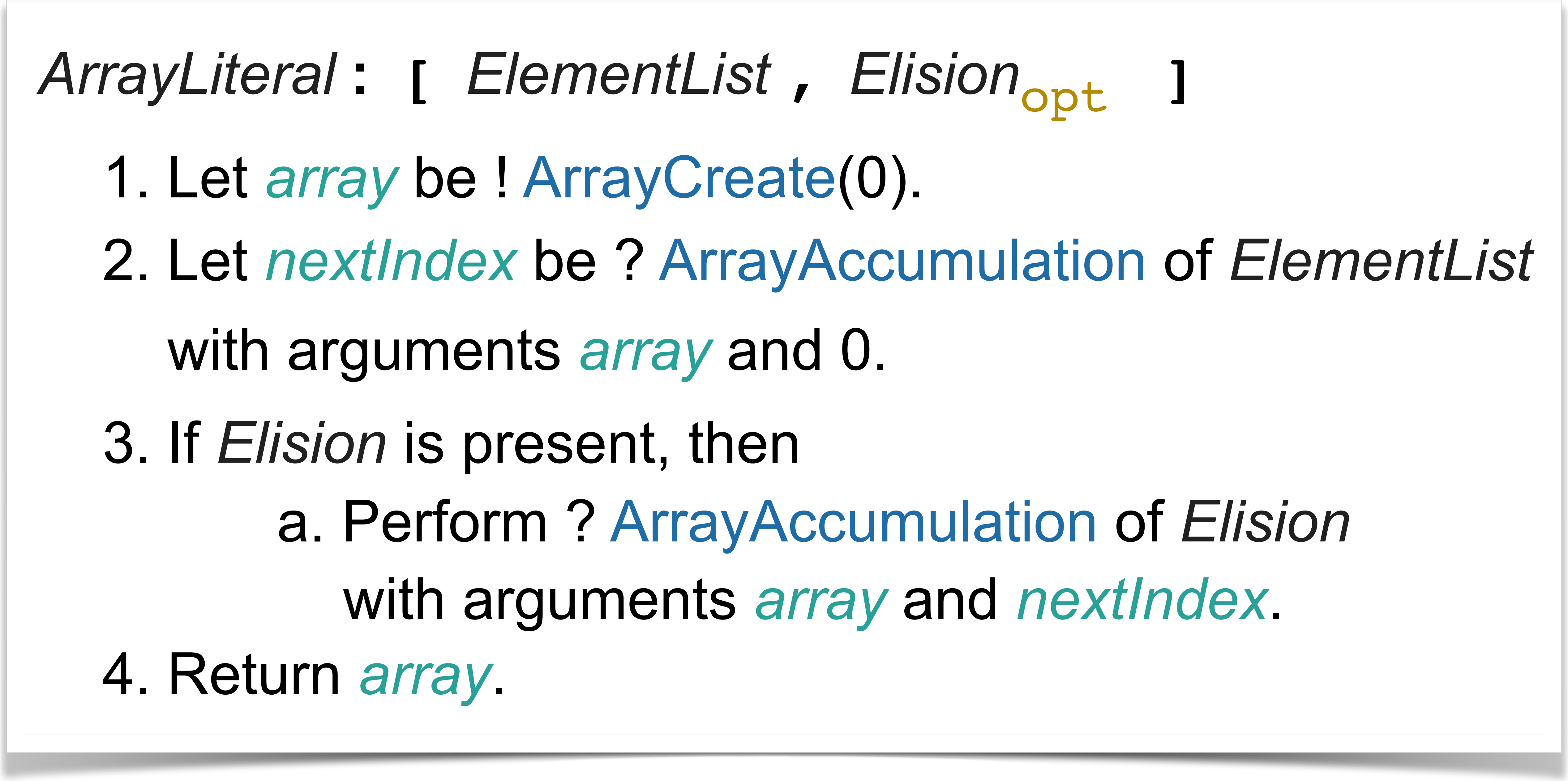
(**IR$_{\text{ES}}$ - Intermediate Representation** for ECMA-262)

$$
\begin{array}{ll}
\text{Programs} & \mathfrak{P} \ni P ::= f^* \\
\text{Functions} & \mathcal{F} \ni f ::= \texttt{syntax}^? \ \texttt{def} \ \texttt{x}(\texttt{x}^*) \ \{[\ell : i]^*\} \\
\text{Variables} & \mathcal{X} \ni \texttt{x} \\
\text{Labels} & \mathcal{L} \ni \ell \\
\text{Instructions} & \mathcal{I} \ni i ::= r := e \mid \texttt{x} := \{\} \mid \texttt{x} := e(e^*) \\
 & \qquad\qquad \mid \ \texttt{if} \ e \ \ell \ \ell \mid \texttt{return} \ e \\
\text{Expressions} & \mathcal{E} \ni e ::= v^{\text{p}} \mid \texttt{op}(e^*) \mid r \\
\text{References} & \mathcal{R} \ni r ::= \texttt{x} \mid e[e] \mid e[e]_{\text{js}} \\
 & \qquad\qquad \vdots
\end{array}
$$

$$
\begin{array}{lrl}
\text{Values} & v \in \mathbb{V} & = \mathbb{A} \uplus \mathbb{V}^{\text{p}} \uplus \boxed{\mathbb{T}} \uplus \mathcal{F} \\
\text{Primitive Values} & v^{\text{p}} \in \mathbb{V}^{\text{p}} & = \mathbb{V}_{\text{bool}} \uplus \mathbb{V}_{\text{int}} \uplus \mathbb{V}_{\text{str}} \uplus \cdots \\
\boxed{\text{JS ASTs} \qquad\qquad t \in \mathbb{T}} & &
\end{array}
$$

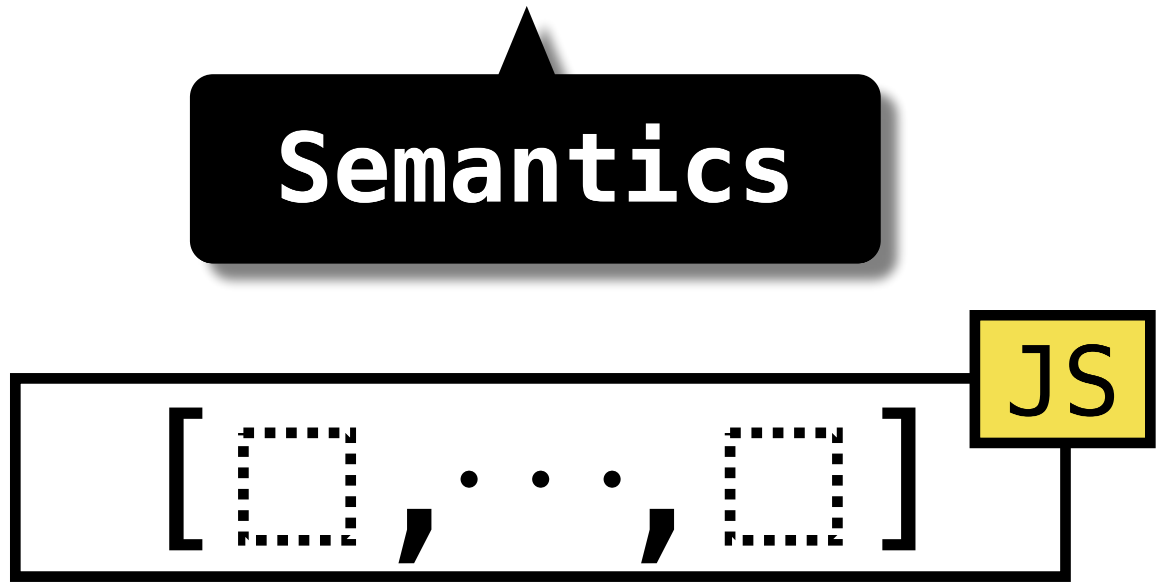
JISET - Algorithm Compiler

Abstract algorithm for *ArrayLiteral* in ES13



ArrayLiteral : `[` *ElementList* `,` *Elision*$_{\texttt{opt}}$ `]`

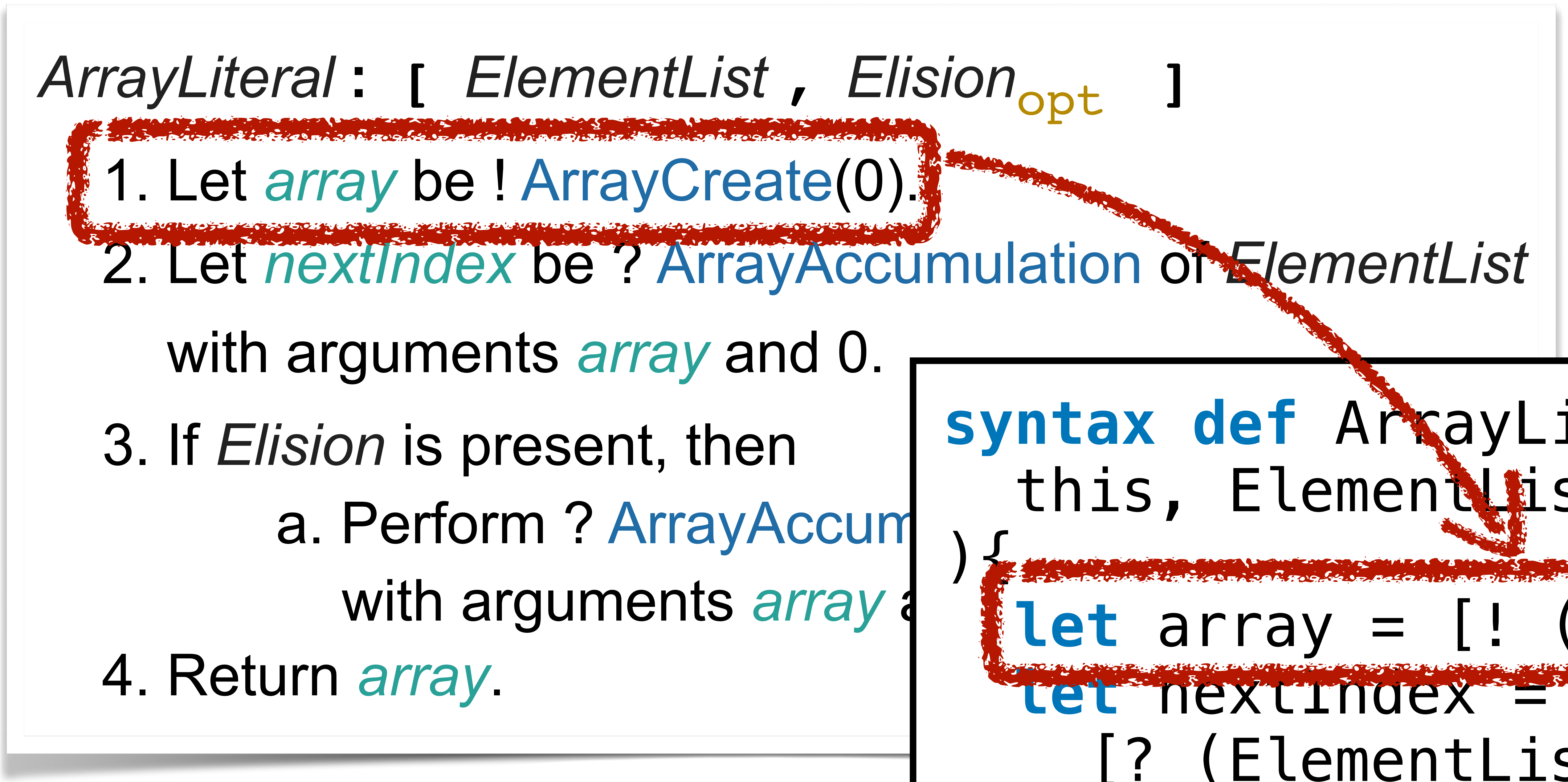
```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
    a. Perform ? ArrayAccumulation of Elision
       with arguments array and nextIndex.
4. Return array.
```



Semantics

JISET - Algorithm Compiler

Abstract algorithm for *ArrayLiteral* in ES13



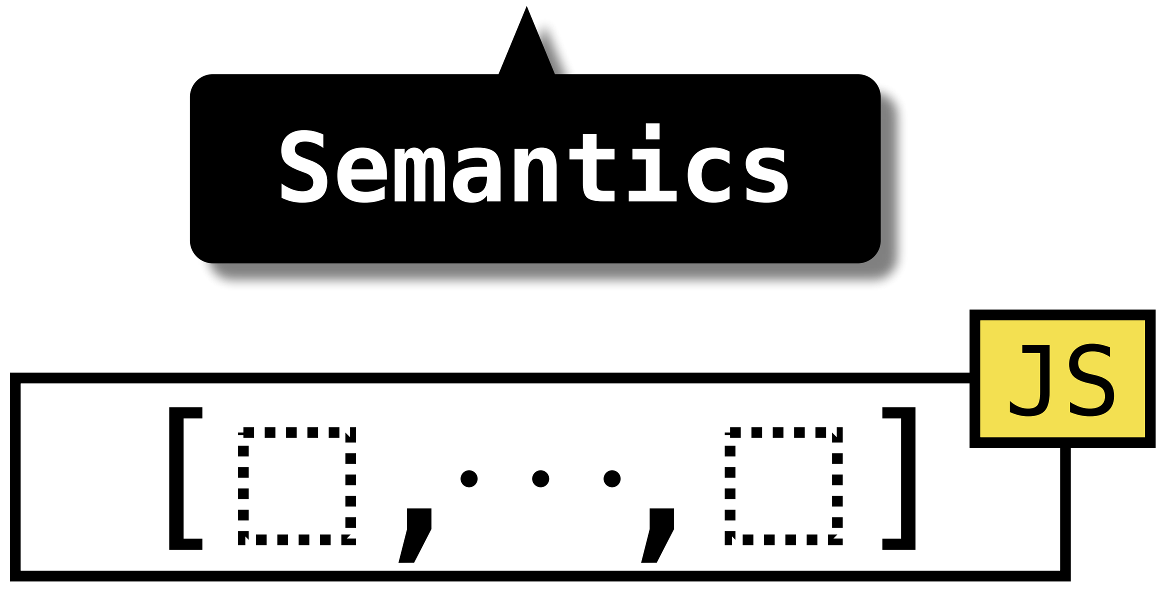
ArrayLiteral : `[` *ElementList* `,` *Elision*$_{\text{opt}}$ `]`

```
1. Let array be ! ArrayCreate(0).
2. Let nextIndex be ? ArrayAccumulation of ElementList
   with arguments array and 0.
3. If Elision is present, then
     a. Perform ? ArrayAccum
        with arguments array a
4. Return array.
```

118 compile rules for
steps in abstract algorithms

```
syntax def ArrayLiteral[2].Evaluation(
  this, ElementList, Elision
){
  let array = [! (ArrayCreate 0)]
  let nextIndex =
    [? (ElementList.ArrayAccumulation array 0)]
  if (! (= Elision absent))
    [? (Elision.ArrayAccumulation array nextIndex)]
  return array
}
```

IR_{ES} function for *ArrayLiteral* in ES13



Parsing rules **Conversion Rules**

```
S = // statements
  Let ~ V ~ be ~ E ~ .   ^^   ILet

E = // expressions
  ! E                    ^^   EAbruptCheck |
  str ~ ( ~ E ~ )        ^^   ECall        |
  num                    ^^ _.toDoub le
```

Simplified compile rules

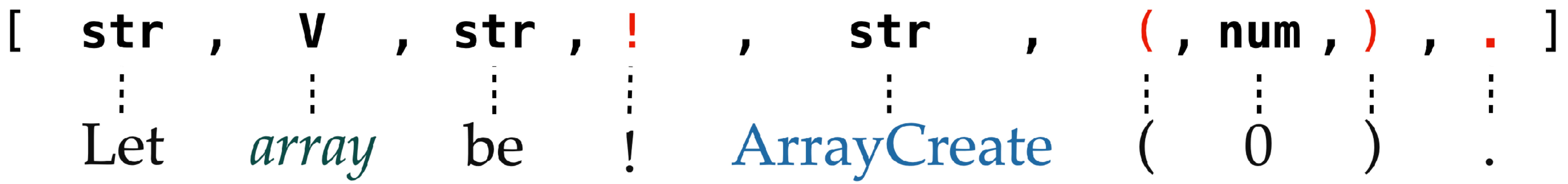
Let *array* be ! ArrayCreate (0) .

Parsing rules **Conversion Rules**

```
S = // statements
  Let ~ V ~ be ~ E ~ .   ^^  ILet

E = // expressions
  ! E                    ^^  EAbruptCheck |
  str ~ ( ~ E ~ )        ^^  ECall        |
  num                    ^^ _.toDoub le
```

Simplified compile rules



[str , V , str , ! , str , (, num ,) , .]

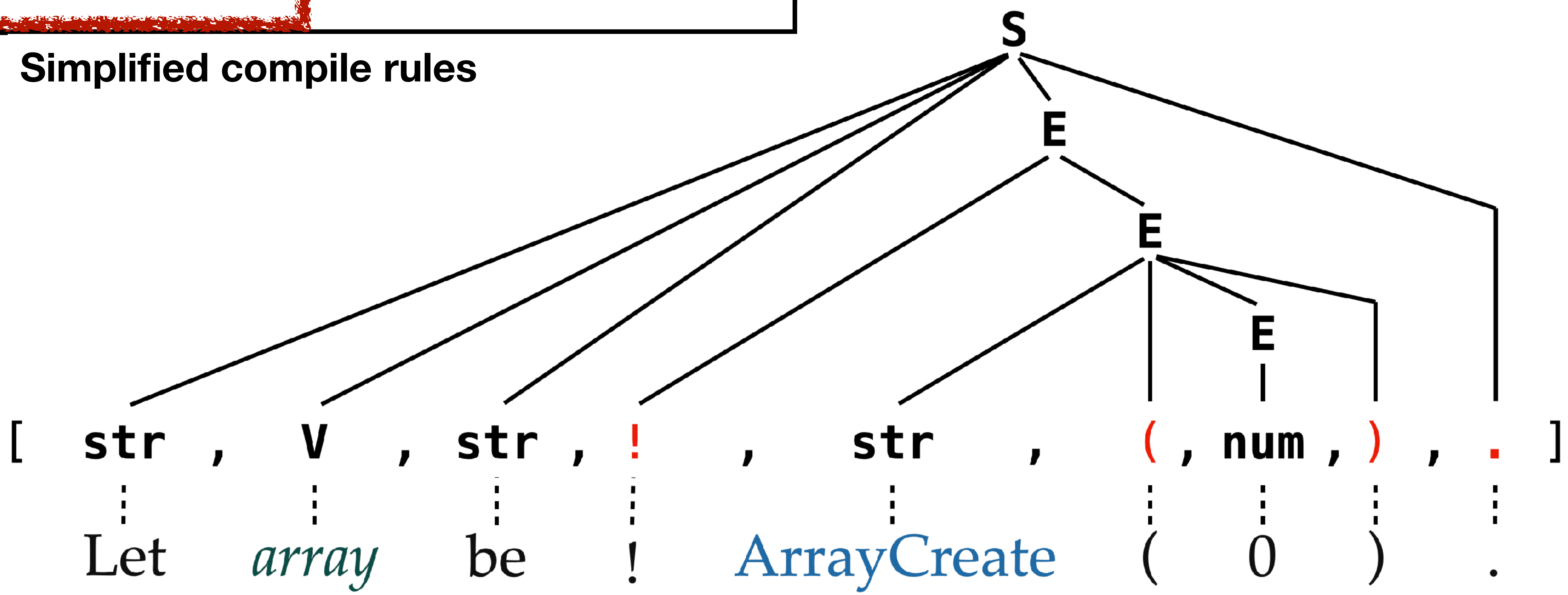
Let *array* be ! ArrayCreate (0) .

Parsing rules **Conversion Rules**

```
S = // statements
  Let ~ V ~ be ~ E ~ .   ^^  ILet

E = // expressions
  ! E                    ^^  EAbruptCheck |
  str ~ ( ~ E ~ )        ^^  ECall        |
  num                    ^^ _.toDoub le
```

Simplified compile rules



Parsing rules **Conversion Rules**

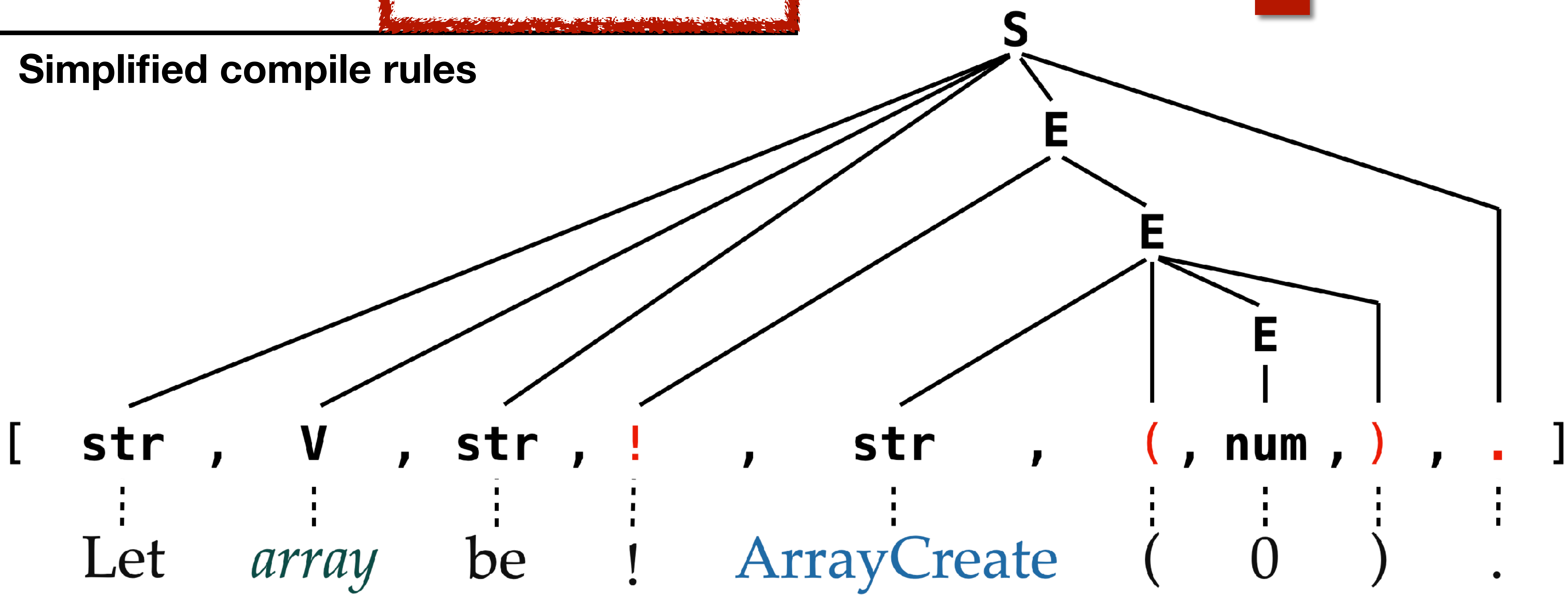
```
S = // statements
  Let ~ V ~ be ~ E ~ .     ^^  ILet

E = // expressions
  ! E                      ^^  EAbruptCheck |
  str ~ ( ~ E ~ )          ^^  ECall        |
  num                      ^^ _.toDouble
```

Simplified compile rules

```
let array = [! (ArrayCreate 0)]
```

```
ILet(array, EAbruptCheck(
  ECall("ArrayCreate", 0)))
```



JISET - Evaluation

≈ 96% Compiled

🟩 auto 🟥 manual

T: Total **L**: Core Language Semantics **B**: Built-in Libraries

Version	# Algo.		
ES7	2,105	T	**10,471 / 10,982** (95.35%)
		L	**8,041 / 8,415** (95.56%)
		B	**2,430 / 2,567** (94.66%)
ES8	2,238	T	**11,181 / 11,732** (95.30%)
		L	**8,453 / 8,811** (95.94%)
		B	**2,728 / 2,921** (93.39%)
ES9	2,370	T	**11,849 / 12,393** (95.61%)
		L	**8,932 / 9,311** (95.93%)
		B	**2,917 / 3,082** (94.65%)
ES10	2,396	T	**12,022 / 12,569** (95.65%)
		L	**9,073 / 9,456** (94.95%)
		B	**2,949 / 3,113** (94.73%)
ES11	2,521	T	**12,505 / 13,047** (94.85%)
		L	**9,495 / 9,881** (96.09%)
		B	**3,010 / 3,166** (95.07%)
ES12	2,640	T	**12,975 / 13,544** (95.80%)
		L	**9,717 / 10,136** (95.87%)
		B	**3,258 / 3,408** (95.60%)
Average	2,378	T	**11,834 / 12,378** (95.61%)
		L	**8,952 / 9,335** (95.90%)
		B	**2,882 / 3,043** (94.71%)

JISET - Evaluation

≈ 96% Compiled

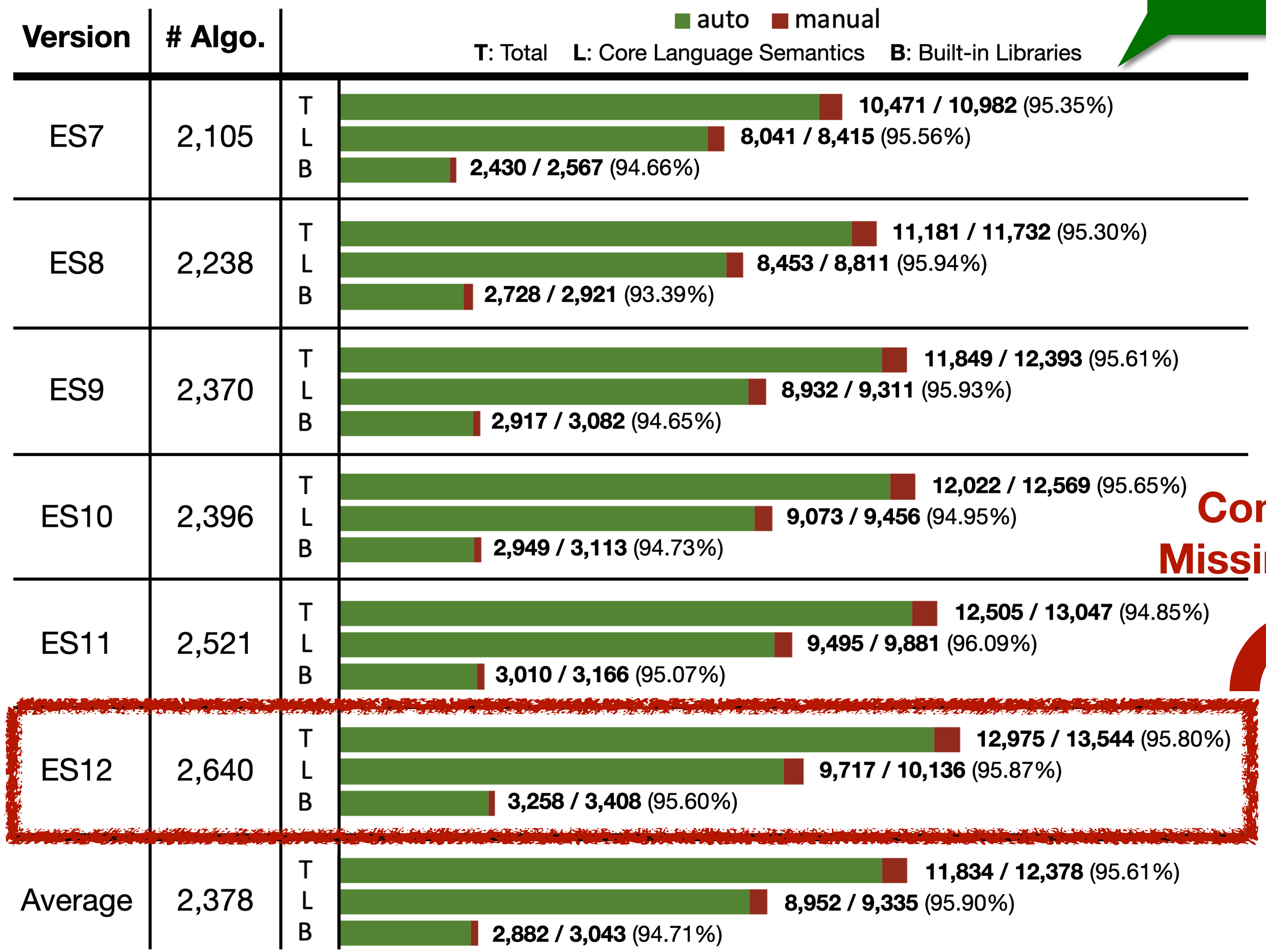
🟩 auto 🟥 manual

T: Total **L**: Core Language Semantics **B**: Built-in Libraries

Version	# Algo.		
ES7	2,105	T	**10,471 / 10,982** (95.35%)
		L	**8,041 / 8,415** (95.56%)
		B	**2,430 / 2,567** (94.66%)
ES8	2,238	T	**11,181 / 11,732** (95.30%)
		L	**8,453 / 8,811** (95.94%)
		B	**2,728 / 2,921** (93.39%)
ES9	2,370	T	**11,849 / 12,393** (95.61%)
		L	**8,932 / 9,311** (95.93%)
		B	**2,917 / 3,082** (94.65%)
ES10	2,396	T	**12,022 / 12,569** (95.65%)
		L	**9,073 / 9,456** (94.95%)
		B	**2,949 / 3,113** (94.73%)
ES11	2,521	T	**12,505 / 13,047** (94.85%)
		L	**9,495 / 9,881** (96.09%)
		B	**3,010 / 3,166** (95.07%)
ES12	2,640	T	**12,975 / 13,544** (95.80%)
		L	**9,717 / 10,136** (95.87%)
		B	**3,258 / 3,408** (95.60%)
Average	2,378	T	**11,834 / 12,378** (95.61%)
		L	**8,952 / 9,335** (95.90%)
		B	**2,882 / 3,043** (94.71%)

Complete Missing Parts

JISET - Evaluation



🟩 auto 🟥 manual

T: Total **L**: Core Language Semantics **B**: Built-in Libraries

Version	# Algo.		
ES7	2,105	T	**10,471 / 10,982** (95.35%)
		L	**8,041 / 8,415** (95.56%)
		B	**2,430 / 2,567** (94.66%)
ES8	2,238	T	**11,181 / 11,732** (95.30%)
		L	**8,453 / 8,811** (95.94%)
		B	**2,728 / 2,921** (93.39%)
ES9	2,370	T	**11,849 / 12,393** (95.61%)
		L	**8,932 / 9,311** (95.93%)
		B	**2,917 / 3,082** (94.65%)
ES10	2,396	T	**12,022 / 12,569** (95.65%)
		L	**9,073 / 9,456** (94.95%)
		B	**2,949 / 3,113** (94.73%)
ES11	2,521	T	**12,505 / 13,047** (94.85%)
		L	**9,495 / 9,881** (96.09%)
		B	**3,010 / 3,166** (95.07%)
ES12	2,640	T	**12,975 / 13,544** (95.80%)
		L	**9,717 / 10,136** (95.87%)
		B	**3,258 / 3,408** (95.60%)
Average	2,378	T	**11,834 / 12,378** (95.61%)
		L	**8,952 / 9,335** (95.90%)
		B	**2,882 / 3,043** (94.71%)

≈ 96% Compiled

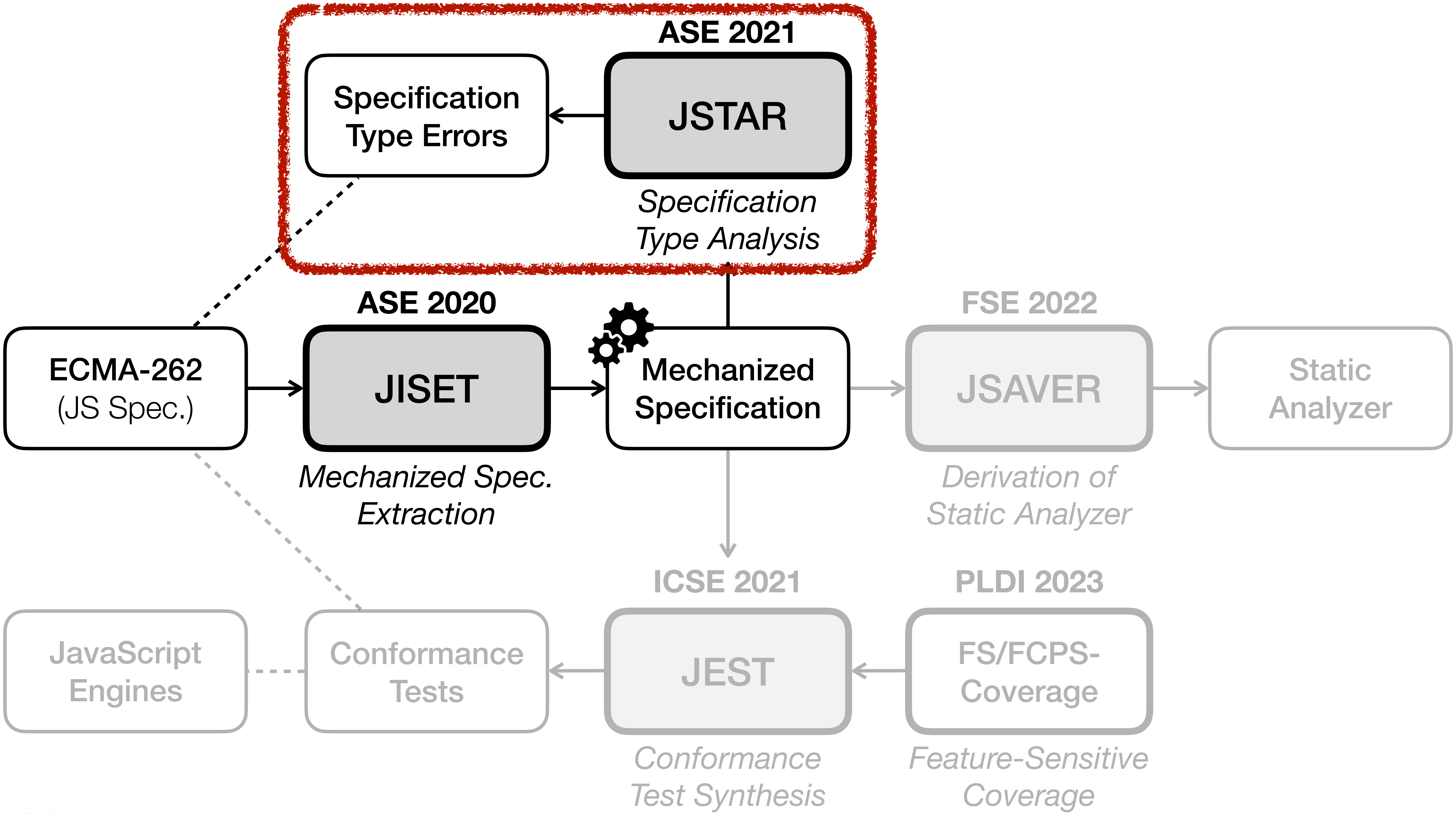
Complete Missing Parts

Passed All Tests

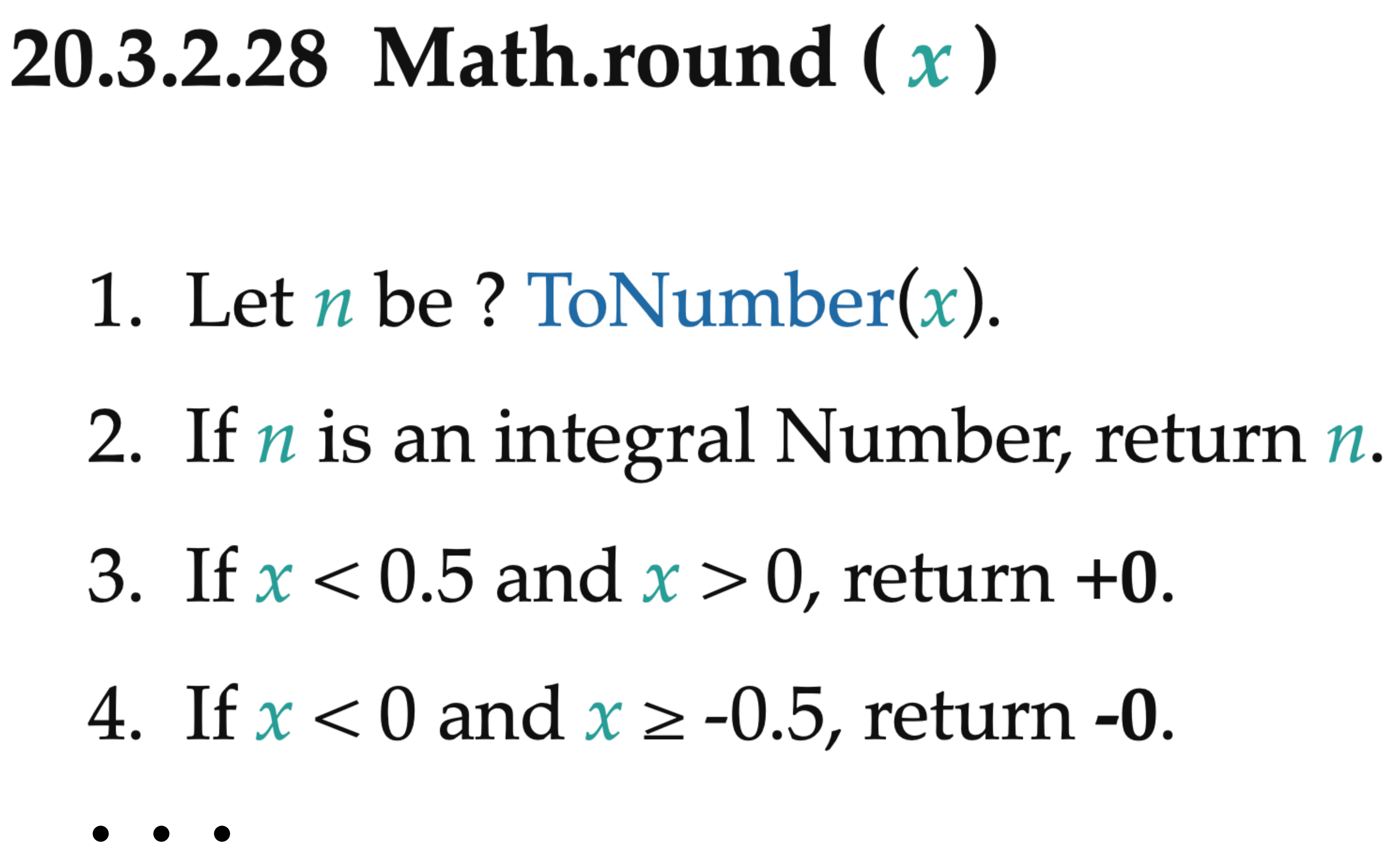
- **Test262**
  (Official Conformance Tests)
  - 18,556 applicable tests
- **Parsing tests**
  - Passed all 18,556 tests
- **Evaluation Tests**
  - Passed all 18,556 tests

PLRG

[ASE 2021] J. Park et al., “JSTAR: JavaScript Specification Type Analyzer using Refinement”



JSTAR - Specification Type Analysis

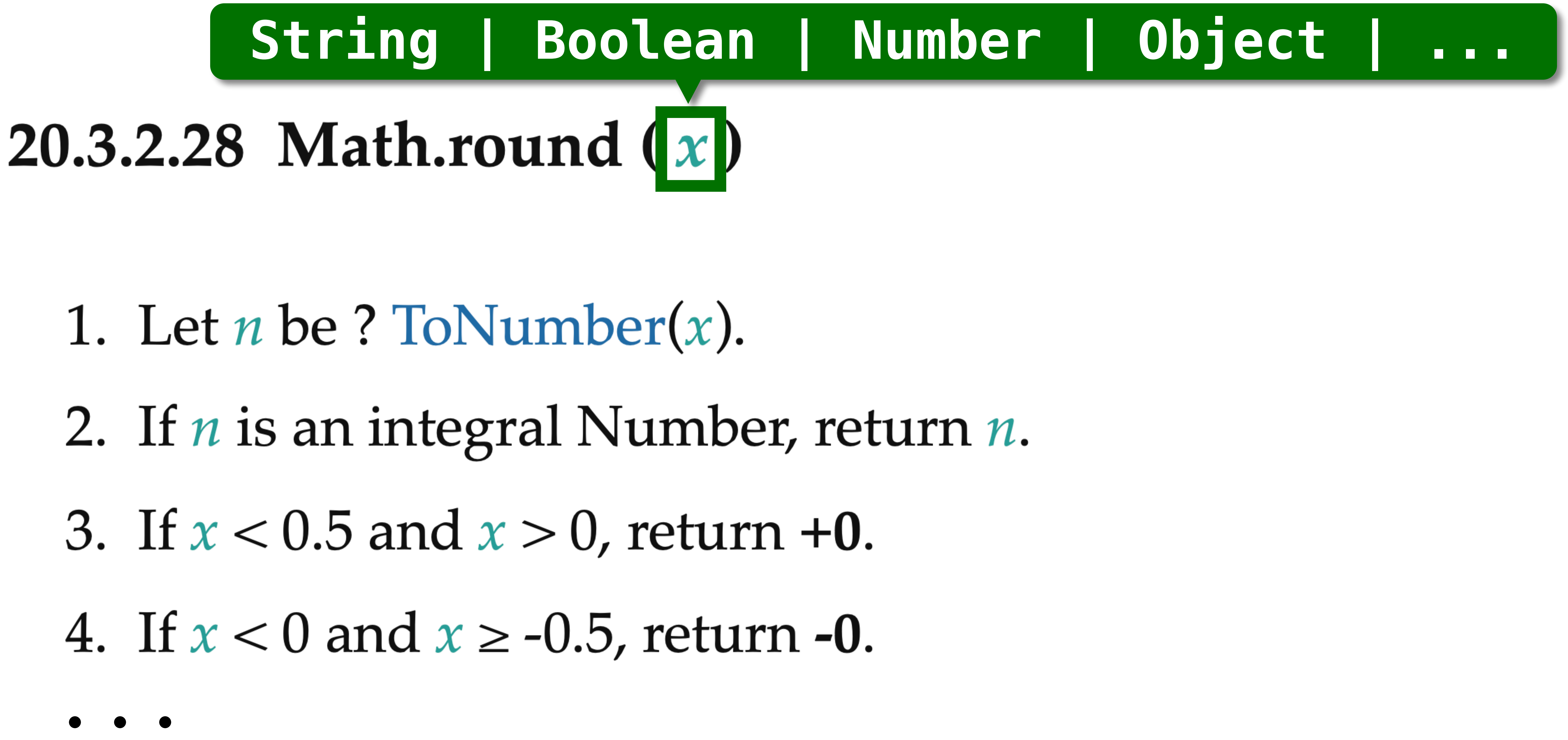


20.3.2.28 Math.round (*x*)

```
1. Let $n$ be ? ToNumber($x$).
2. If $n$ is an integral Number, return $n$.
3. If $x < 0.5$ and $x > 0$, return +0.
4. If $x < 0$ and $x \geq -0.5$, return -0.
…
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



`String | Boolean | Number | Object | ...`

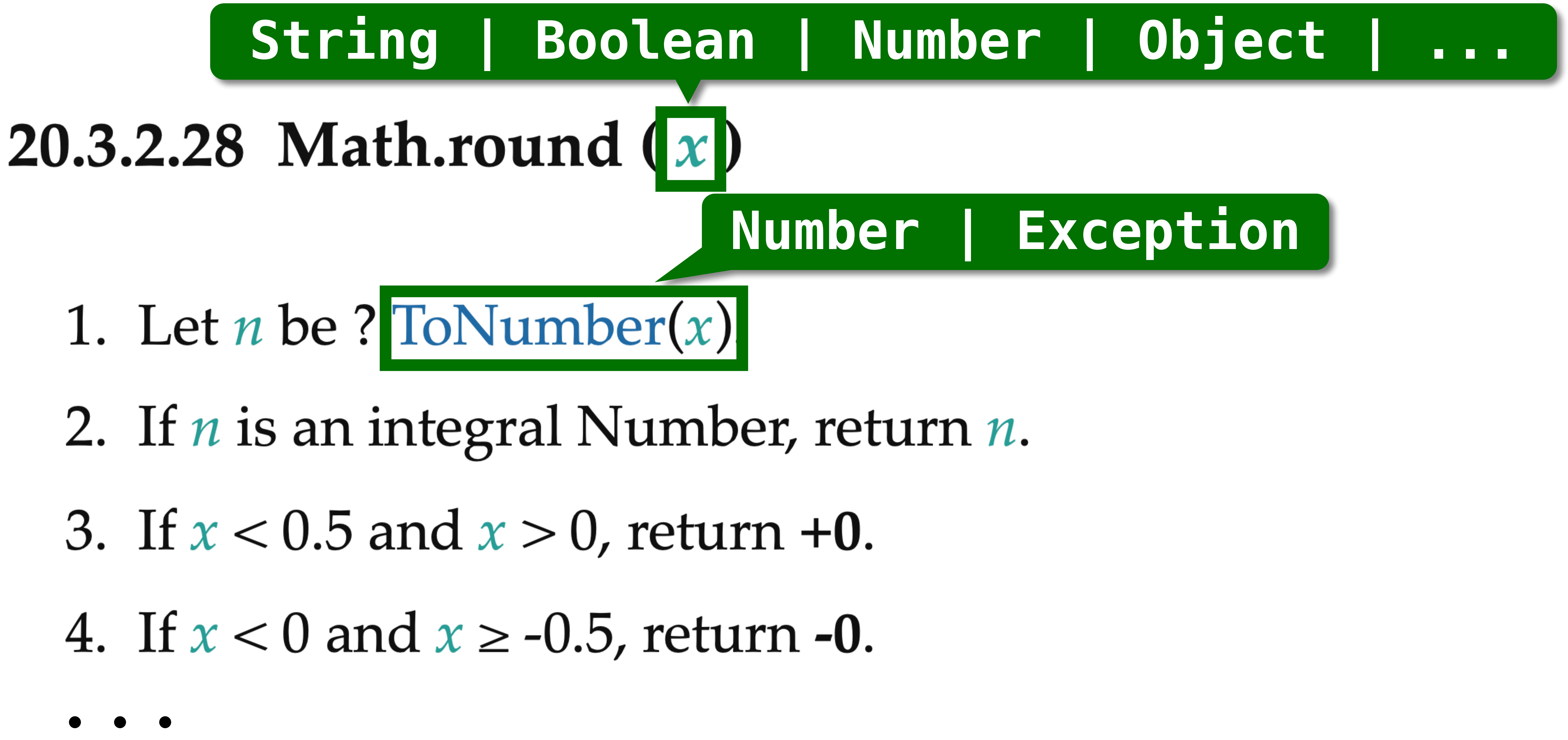
20.3.2.28 Math.round (x)

1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

. . .

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



`String | Boolean | Number | Object | ...`

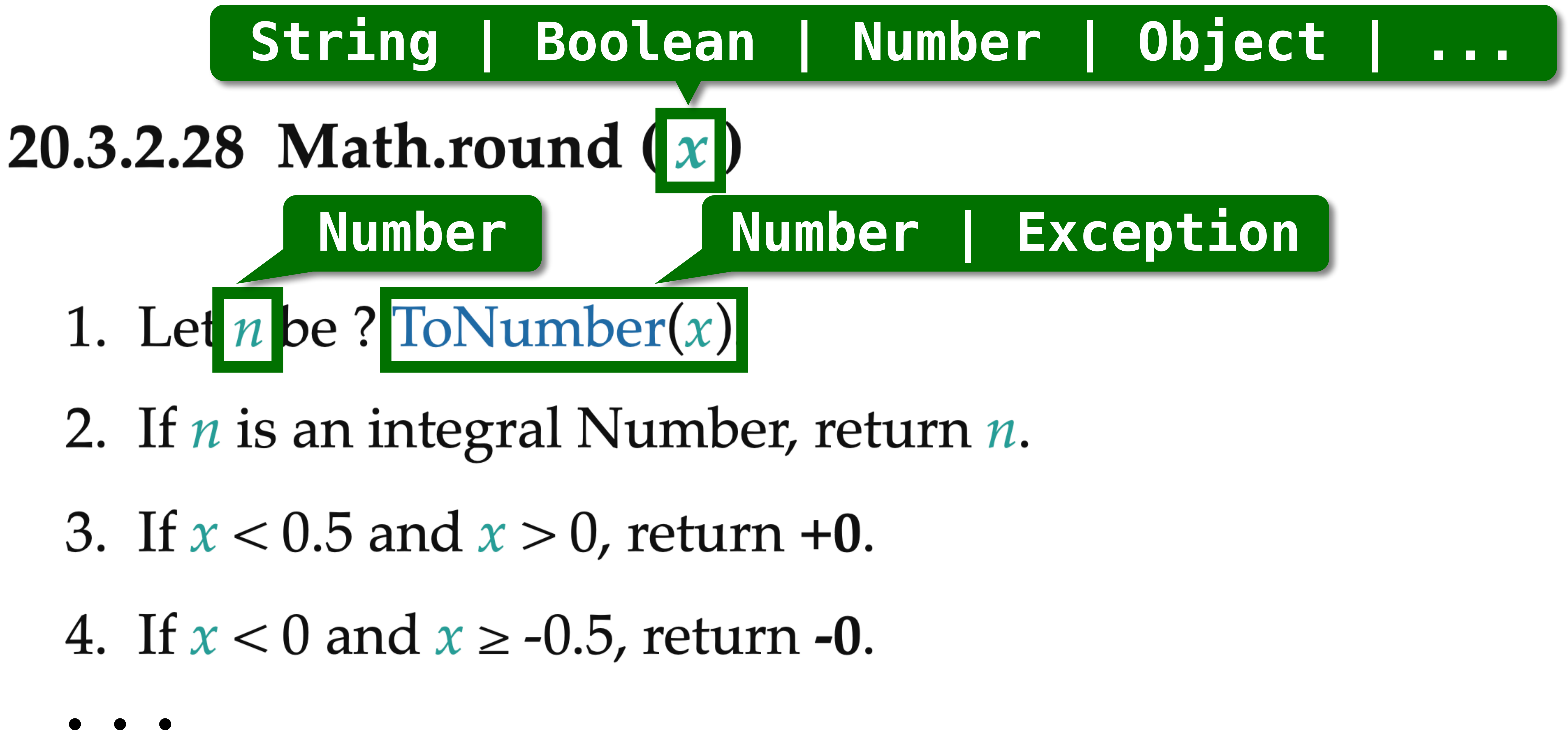
20.3.2.28 Math.round (*x*)

`Number | Exception`

```
1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return +0.
4. If $x < 0$ and $x \geq -0.5$, return -0.
. . .
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



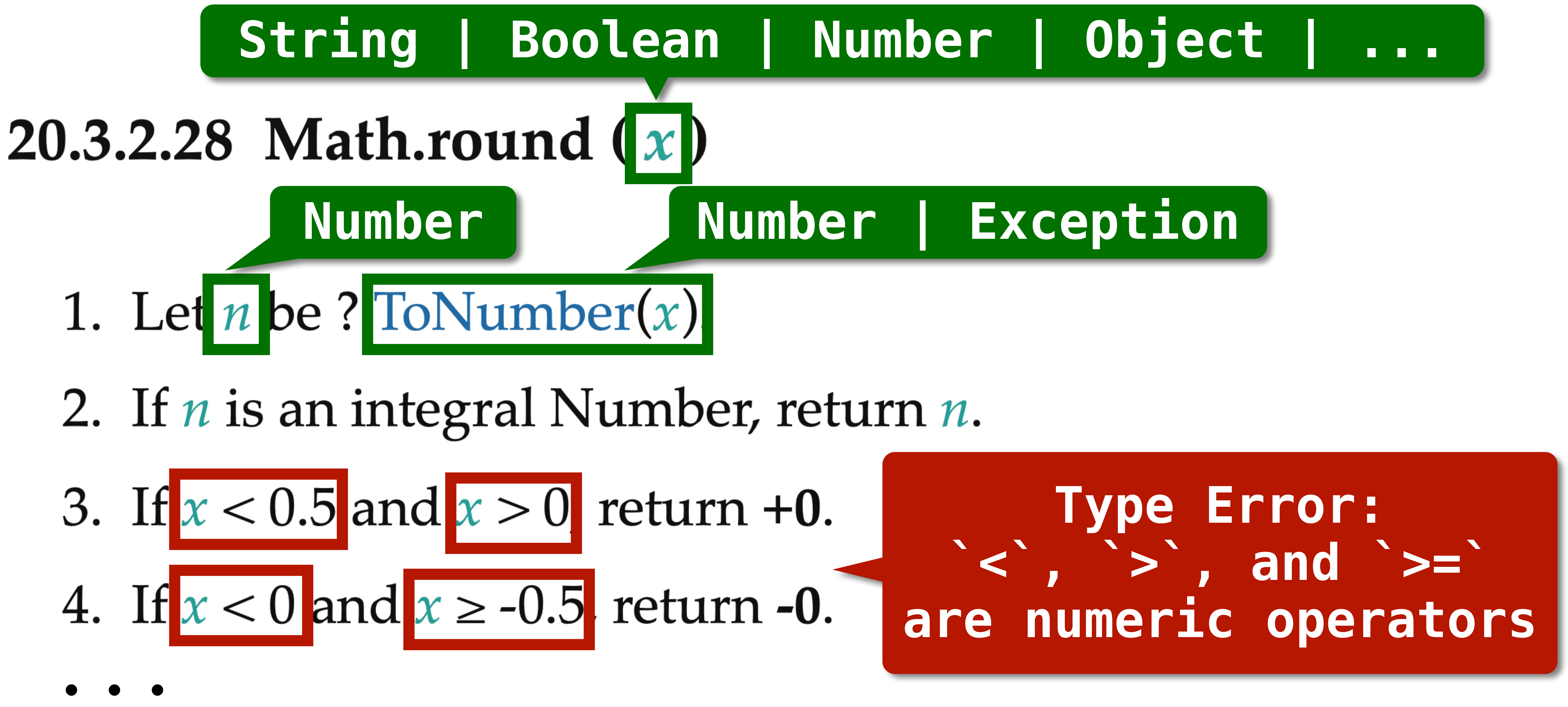
20.3.2.28 Math.round (*x*)

1. Let *n* be ? ToNumber(*x*).
2. If *n* is an integral Number, return *n*.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

• • •

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



String | Boolean | Number | Object | ...

20.3.2.28 Math.round (x)

Number

Number | Exception

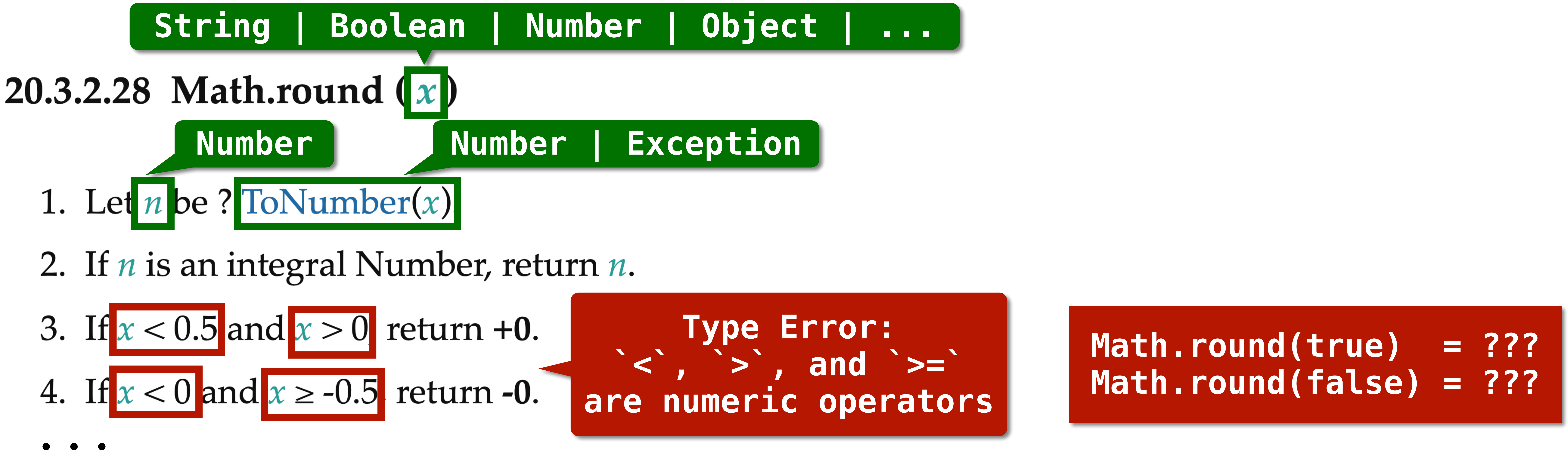
1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq$ -0.5, return **-0**.

. . .

Type Error:
\`<\`, \`>\`, and \`>=\`
are numeric operators

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



20.3.2.28 Math.round (*x*)

1. Let *n* be ? ToNumber(*x*).
2. If *n* is an integral Number, return *n*.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

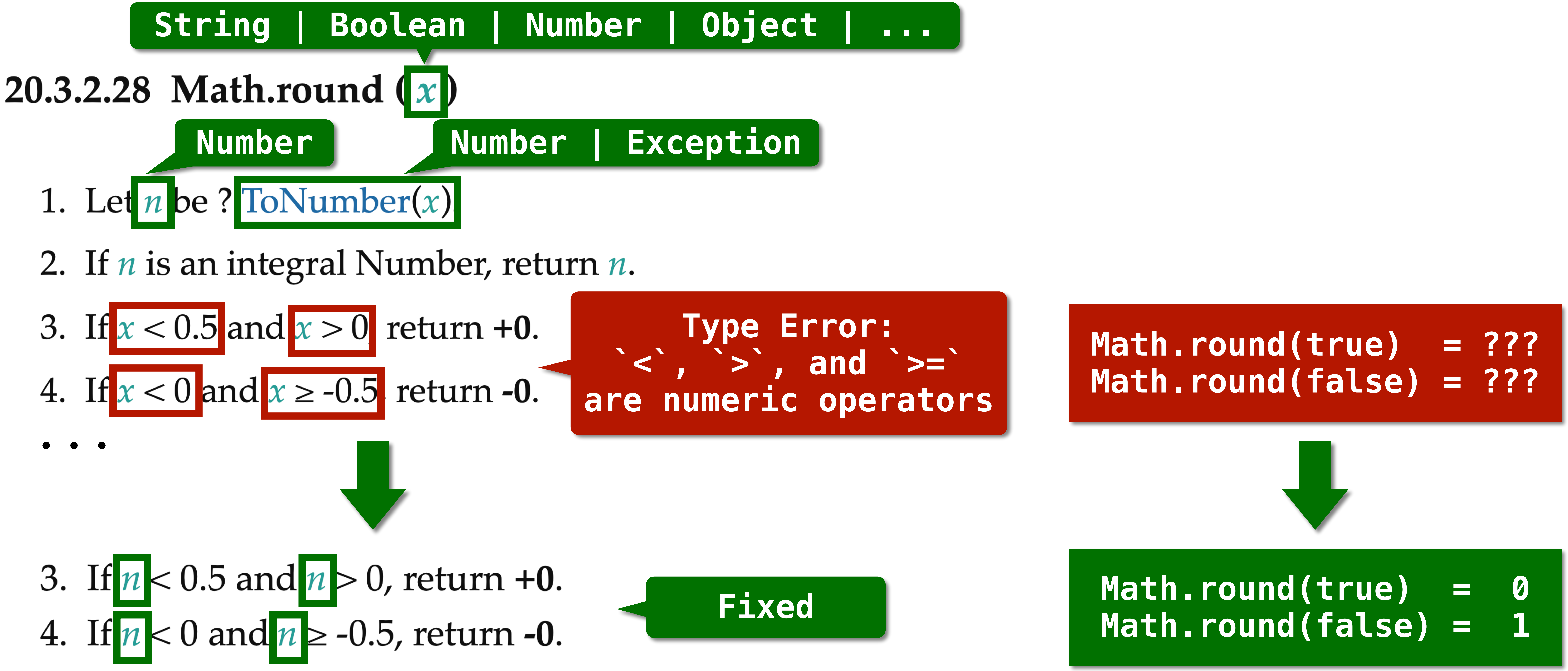
. . .

```
Type Error:
`<`, `>`, and `>=`
are numeric operators
```

```
Math.round(true)  = ???
Math.round(false) = ???
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

JSTAR - Specification Type Analysis



```
String | Boolean | Number | Object | ...
```

20.3.2.28 Math.round (x)

```
Number
```

```
Number | Exception
```

1. Let n be ? ToNumber(x).
2. If n is an integral Number, return n.
3. If $x < 0.5$ and $x > 0$, return **+0**.
4. If $x < 0$ and $x \geq -0.5$, return **-0**.

. . .

```
Type Error:
`<`, `>`, and `>=`
are numeric operators
```

```
Math.round(true)  = ???
Math.round(false) = ???
```

3. If $n < 0.5$ and $n > 0$, return **+0**.
4. If $n < 0$ and $n \geq -0.5$, return **-0**.

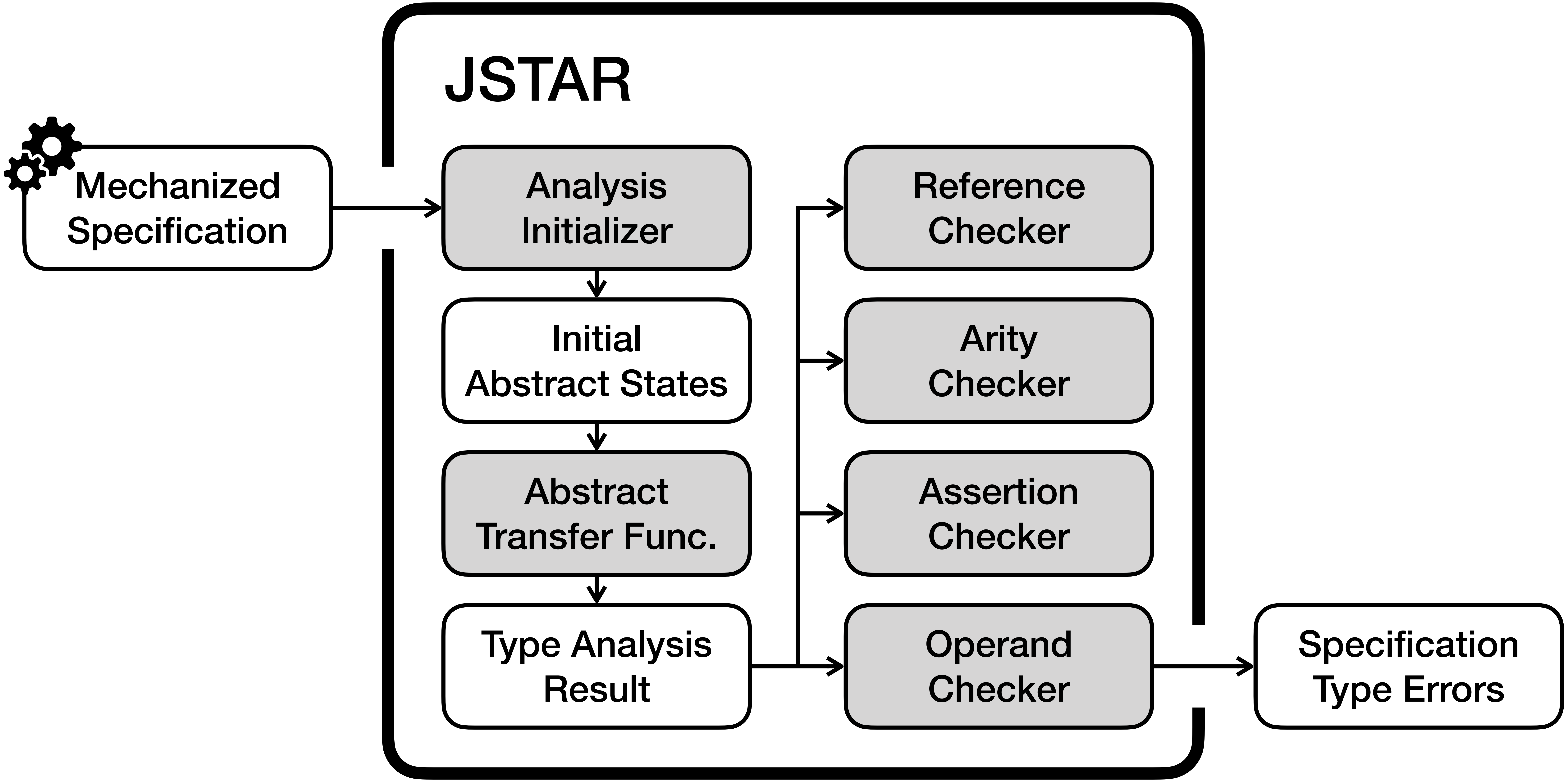
```
Fixed
```

```
Math.round(true)  =  0
Math.round(false) =  1
```

https://github.com/tc39/ecma262/tree/575149cfd77aebcf3a129e165bd89e14caafc31c

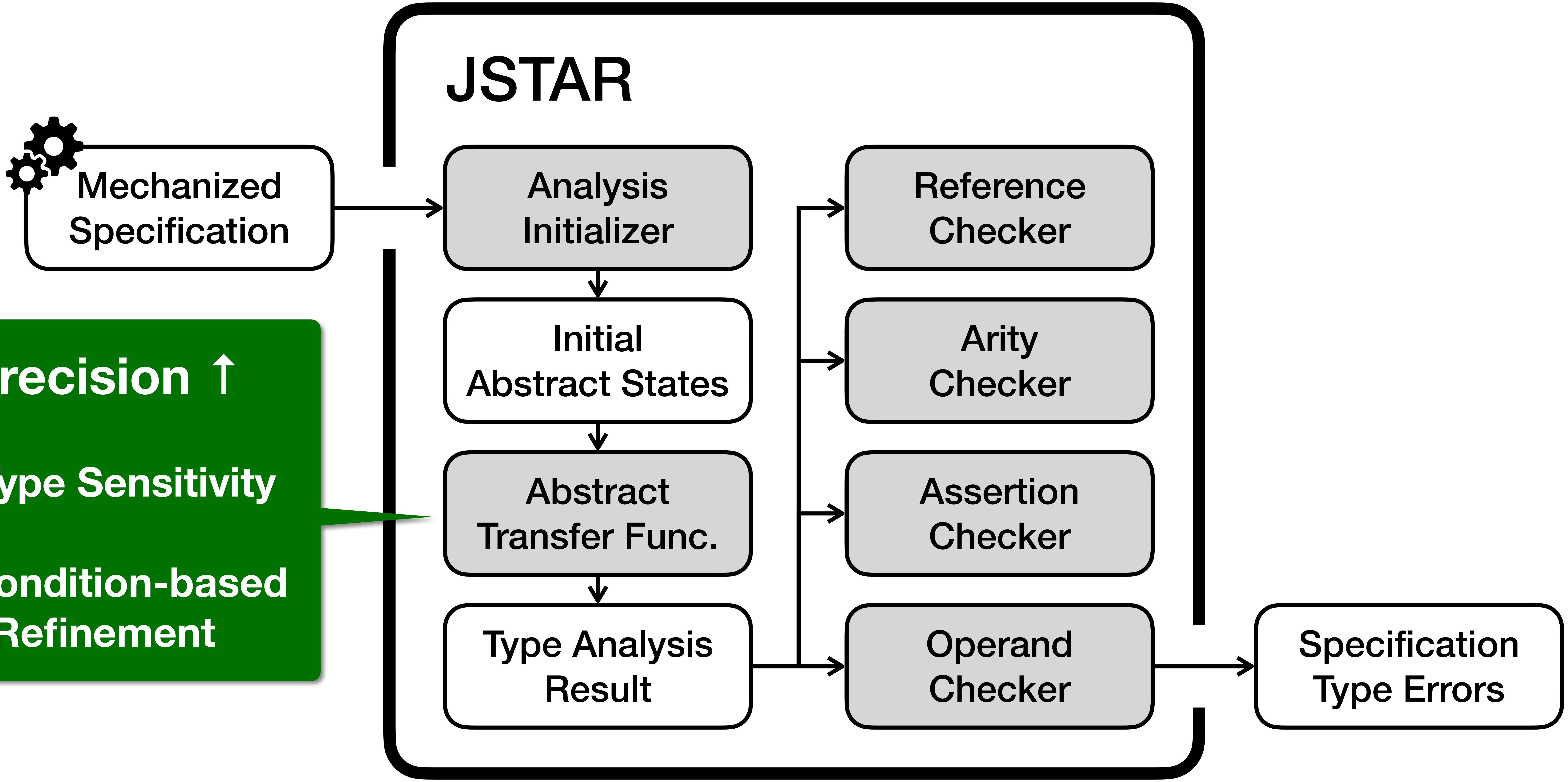
JSTAR

(**J**avaScript **S**pecification **T**ype **A**nalyzer using **R**efinement)



JSTAR

(**J**avaScript **S**pecification **T**ype **A**nalyzer using **R**efinement)

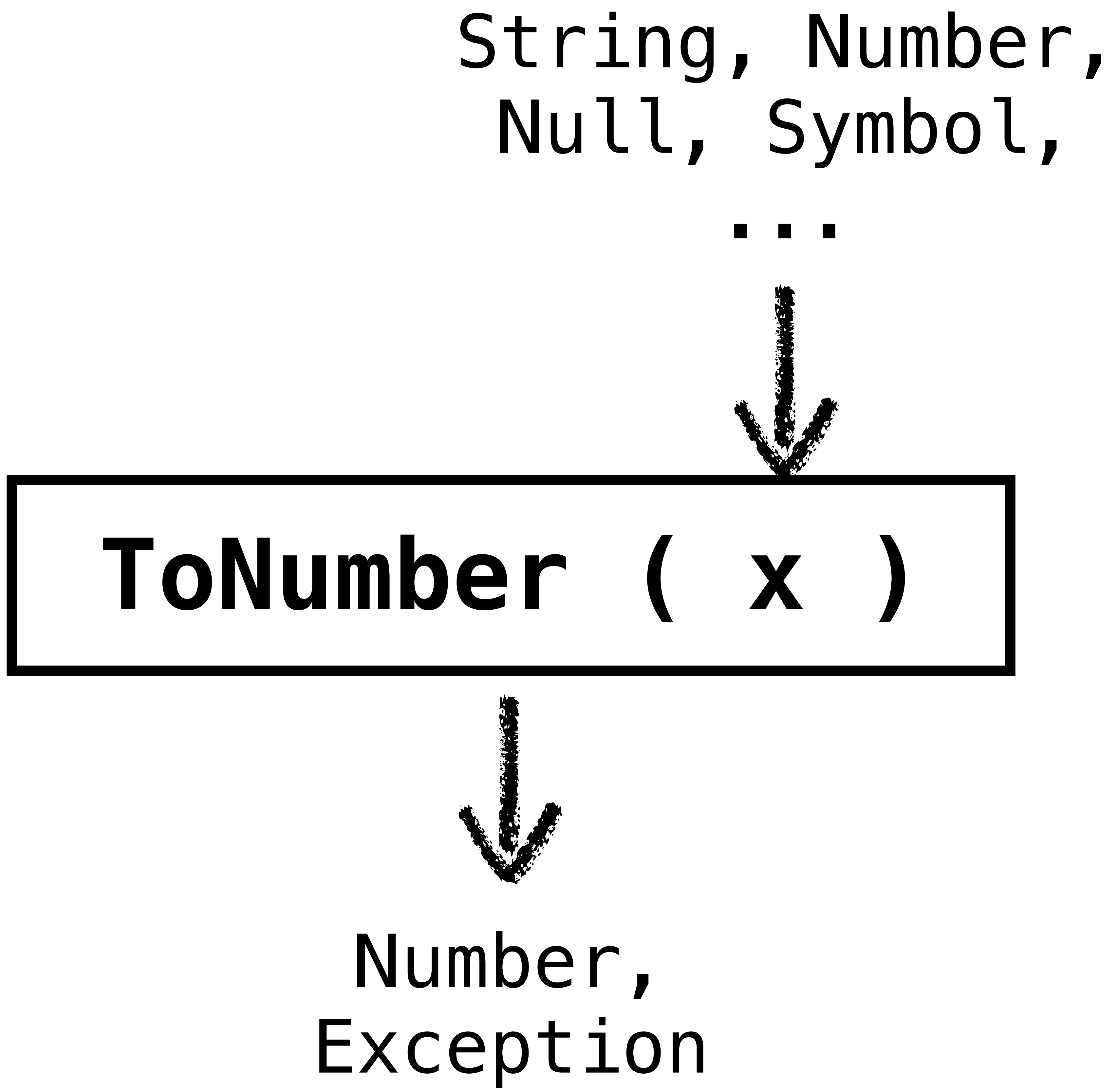


Precision ↑

1. Type Sensitivity

2. Condition-based Refinement

JSTAR - Type Sensitivity

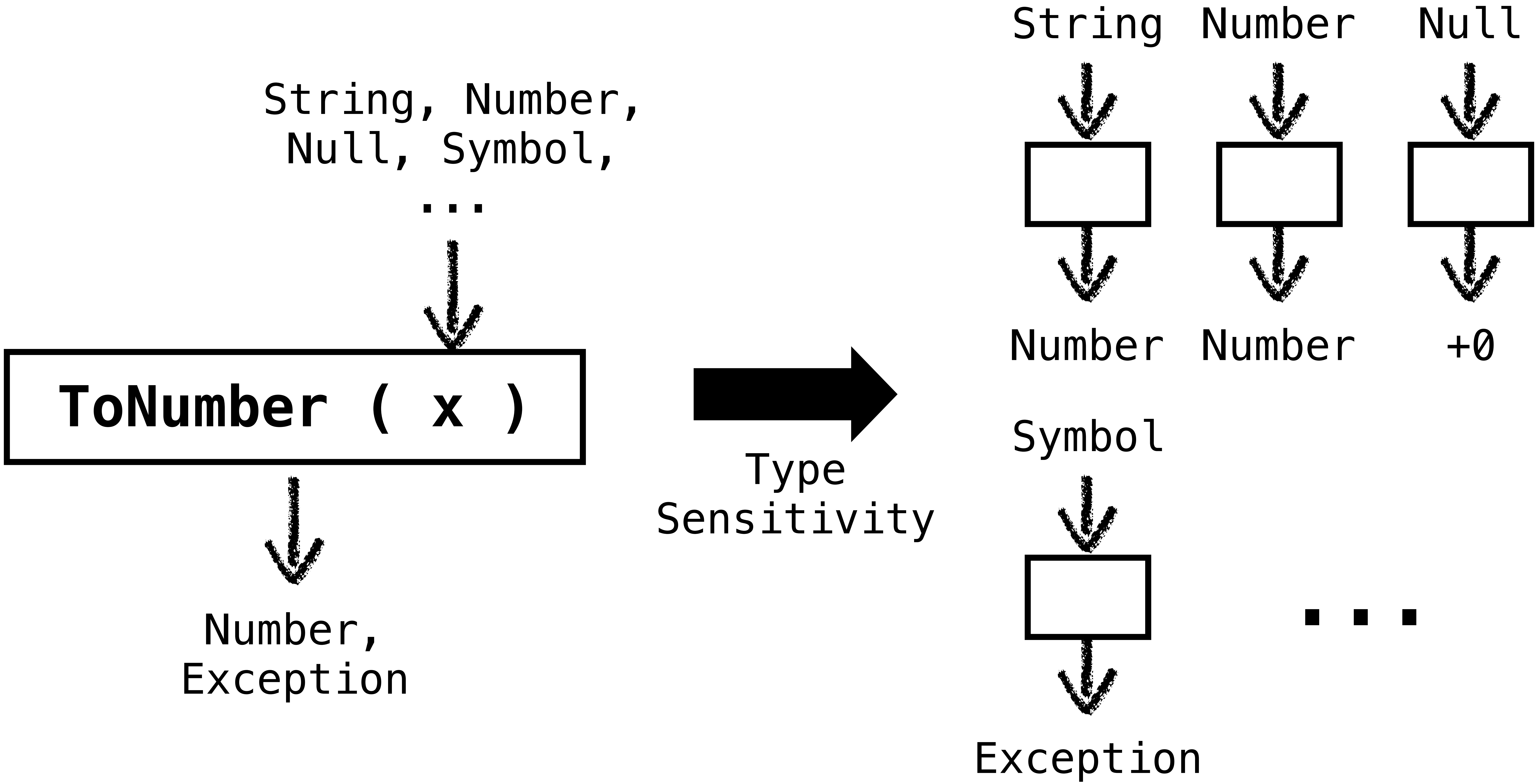


String, Number,
Null, Symbol,
...

ToNumber (x)

Number,
Exception

JSTAR - Type Sensitivity



String, Number,
Null, Symbol,
...

ToNumber (x)

Number,
Exception

Type
Sensitivity

String → Number

Number → Number

Null → +0

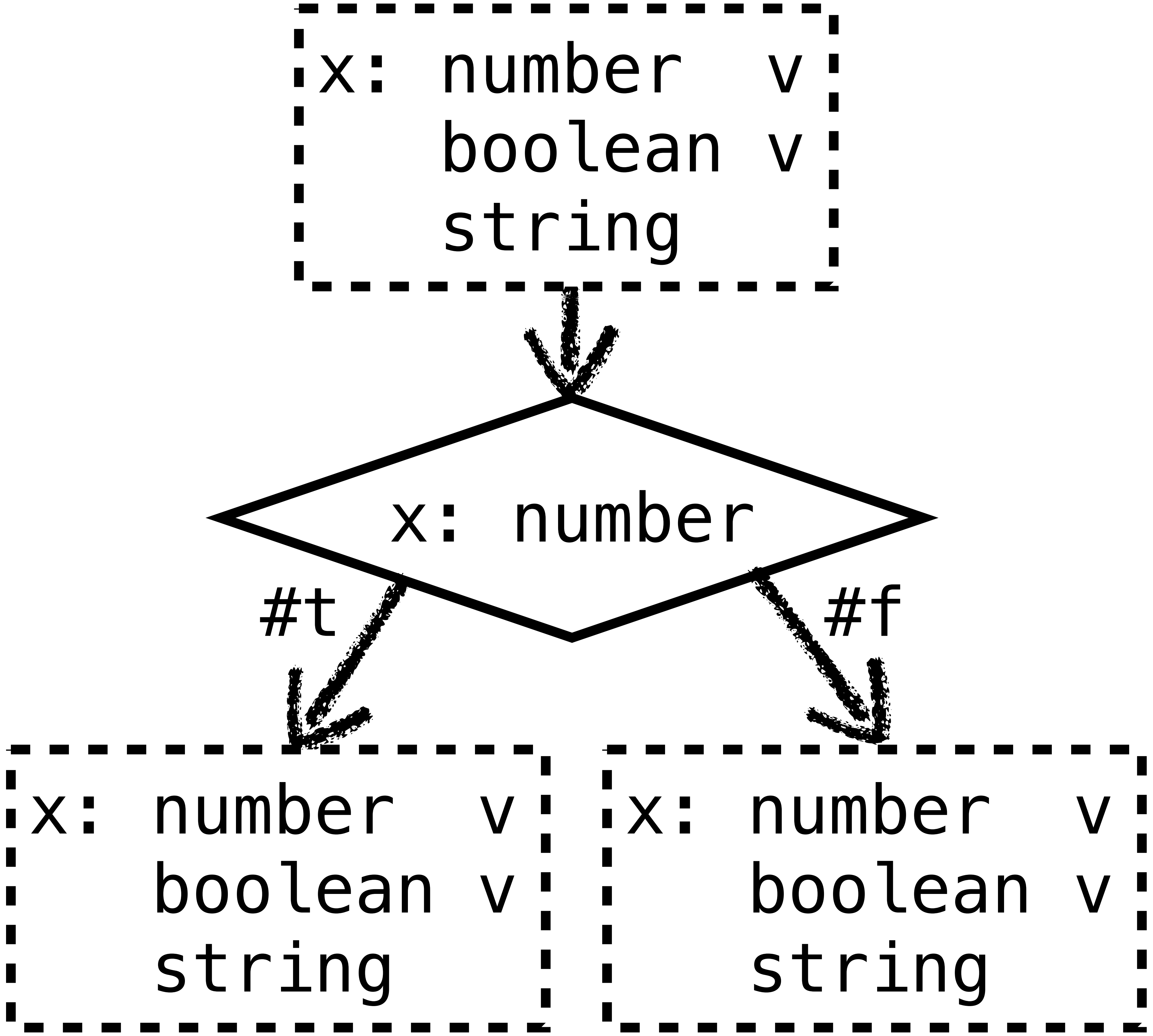
Symbol → Exception

...

JSTAR - Condition-based Refinement

$$
\begin{aligned}
\texttt{refine}(!e, b)(\sigma^\sharp) &= \texttt{refine}(e, \neg b)(\sigma^\sharp) \\
\texttt{refine}(e_0 \texttt{ || } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma^\sharp_0 \sqcup \sigma^\sharp_1 & \text{if } b \\ \sigma^\sharp_0 \sqcap \sigma^\sharp_1 & \text{if } \neg b \end{cases} \\
\texttt{refine}(e_0 \texttt{ \&\& } e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma^\sharp_0 \sqcap \sigma^\sharp_1 & \text{if } b \\ \sigma^\sharp_0 \sqcup \sigma^\sharp_1 & \text{if } \neg b \end{cases} \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \sqcap \texttt{normal}(\mathbb{T})] \\
\texttt{refine}(\texttt{x.Type} == c_{\texttt{normal}}, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \sqcap \{\texttt{abrupt}\}] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \sqcap \tau^\sharp_e] \\
\texttt{refine}(\texttt{x} == e, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \setminus \lfloor \tau^\sharp_e \rfloor] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \sqcap \{\tau\}] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau^\sharp_{\texttt{x}} \setminus \{\tau' \mid \tau' <: \tau\}] \\
\texttt{refine}(e, b)(\sigma^\sharp) &= \sigma^\sharp
\end{aligned}
$$

where $\sigma^\sharp_j = \texttt{refine}(e_j, b)(\sigma^\sharp)$ for $j = 0, 1$, $\tau^\sharp_e = [\![e]\!]^\sharp_e(\sigma^\sharp)$, and $\lfloor \tau^\sharp \rfloor$ returns $\{\tau\}$ if $\tau^\sharp$ denotes a singleton type τ, or returns $\varnothing$, otherwise.



x: number v boolean v string

x: number

#t

#f

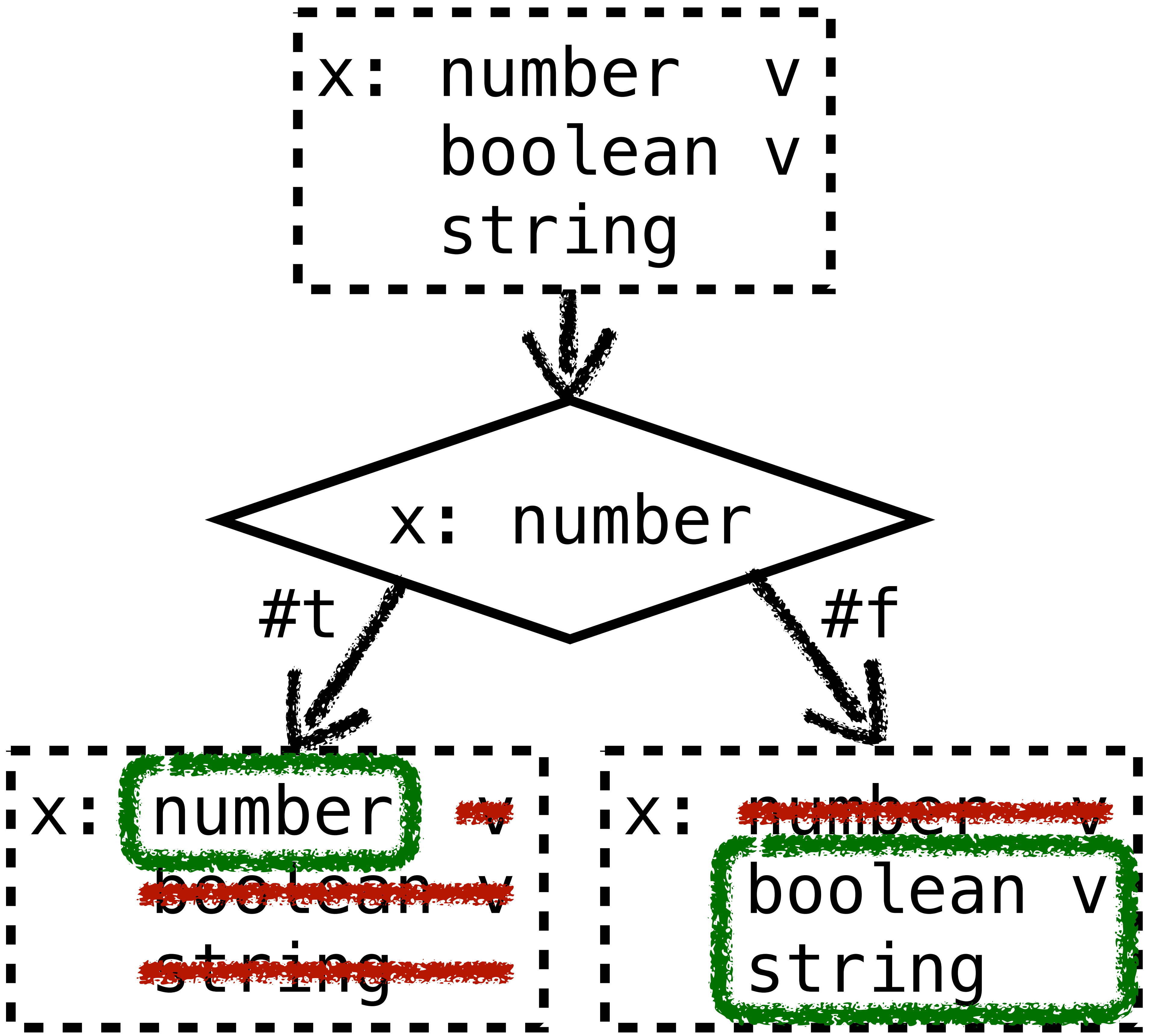
x: number v boolean v string

x: number v boolean v string

JSTAR - Condition-based Refinement

$$
\begin{aligned}
\texttt{refine}(!e, b)(\sigma^\sharp) &= \texttt{refine}(e, \neg b)(\sigma^\sharp) \\
\texttt{refine}(e_0 \;\texttt{||}\; e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(e_0 \;\texttt{\&\&}\; e_1, b)(\sigma^\sharp) &= \begin{cases} \sigma_0^\sharp \sqcap \sigma_1^\sharp & \text{if } b \\ \sigma_0^\sharp \sqcup \sigma_1^\sharp & \text{if } \neg b \end{cases} \\
\texttt{refine}(\texttt{x.Type == } c_{\texttt{normal}}, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \texttt{normal}(\mathbb{T})] \\
\texttt{refine}(\texttt{x.Type == } c_{\texttt{normal}}, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\texttt{abrupt}\}] \\
\texttt{refine}(\texttt{x == } e, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \tau_e^\sharp] \\
\texttt{refine}(\texttt{x == } e, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \lfloor \tau_e^\sharp \rfloor] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#t})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \sqcap \{\tau\}] \\
\texttt{refine}(\texttt{x} : \tau, \texttt{\#f})(\sigma^\sharp) &= \sigma^\sharp[\texttt{x} \mapsto \tau_{\texttt{x}}^\sharp \setminus \{\tau' \mid \tau' <: \tau\}] \\
\texttt{refine}(e, b)(\sigma^\sharp) &= \sigma^\sharp
\end{aligned}
$$

where $\sigma_j^\sharp = \texttt{refine}(e_j, b)(\sigma^\sharp)$ for $j = 0, 1$, $\tau_e^\sharp = [\![e]\!]_e^\sharp(\sigma^\sharp)$, and $\lfloor \tau^\sharp \rfloor$ returns $\{\tau\}$ if $\tau^\sharp$ denotes a singleton type τ, or returns $\varnothing$, otherwise.

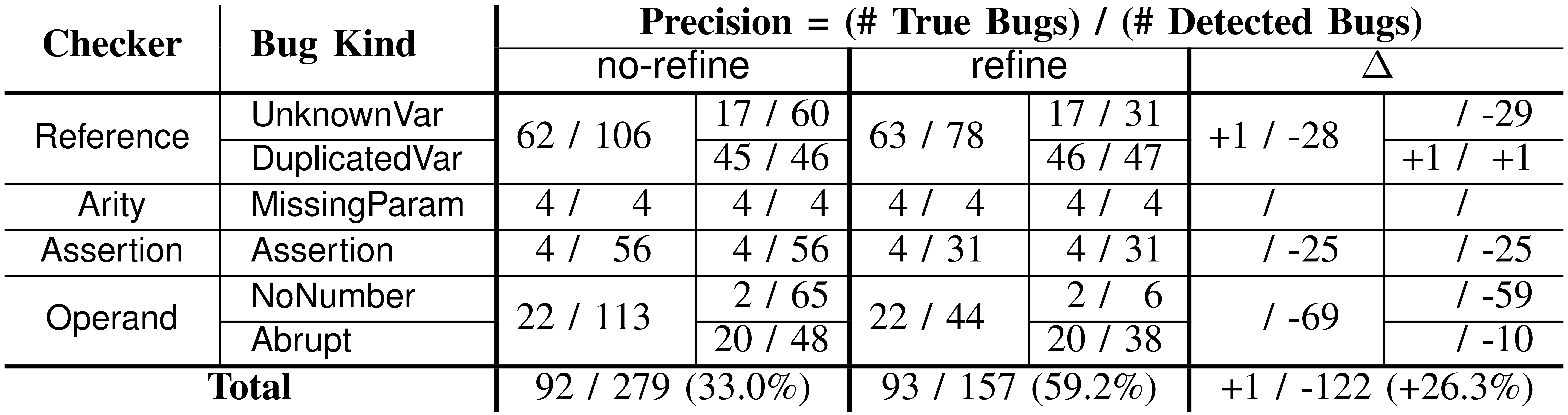


JSTAR - Evaluation

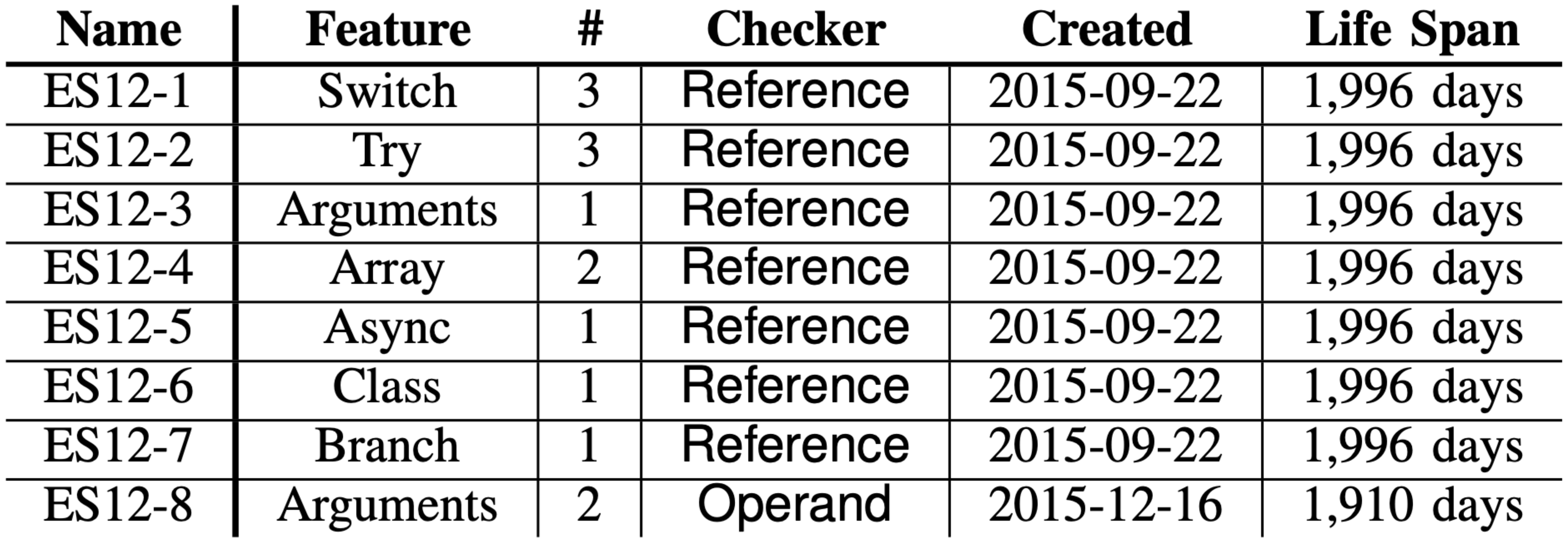
59.2% Precision

93 Errors Detected

- Type analysis on **864 versions** of ECMA-262 in 3 years



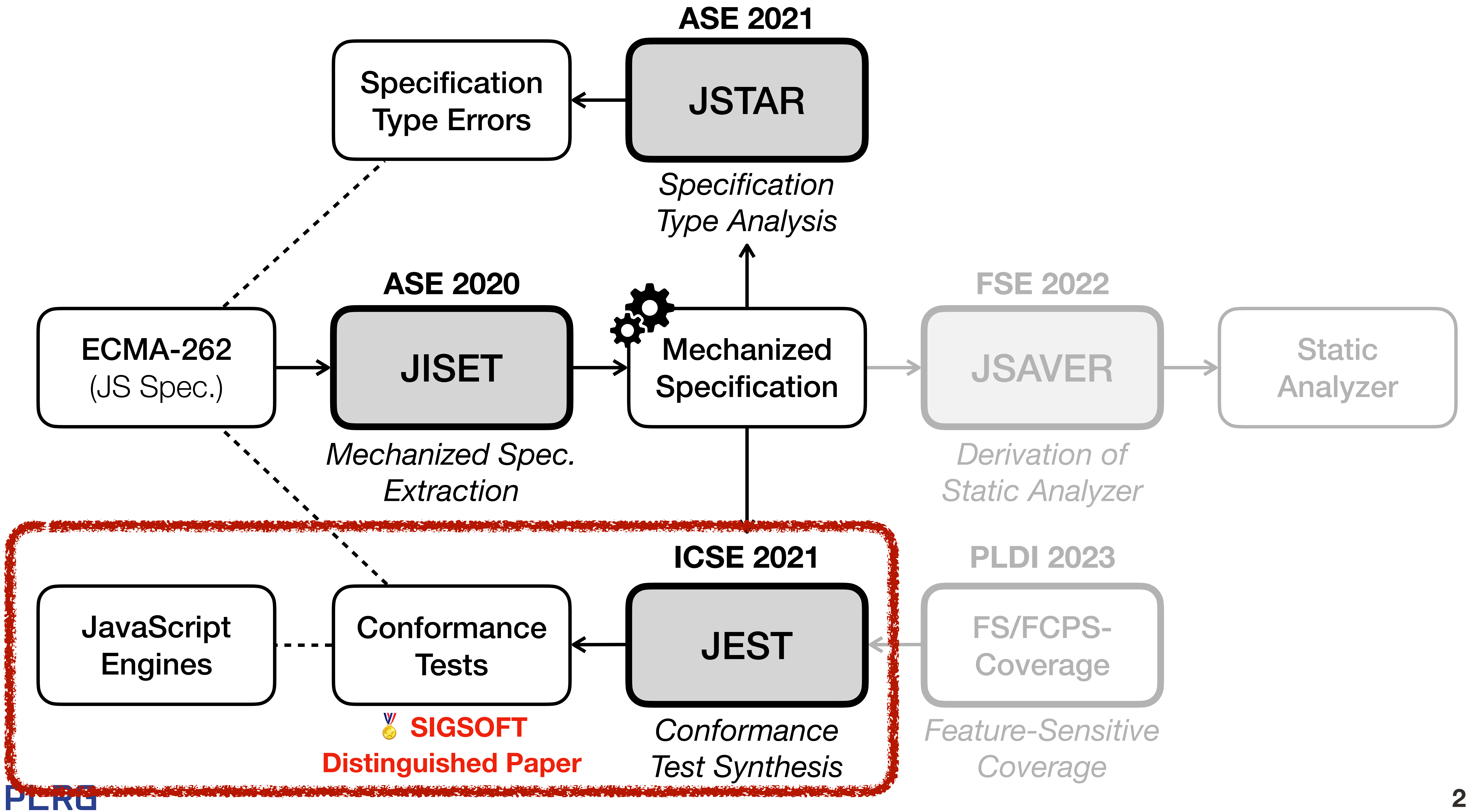
Checker	**Bug Kind**	**Precision = (# True Bugs) / (# Detected Bugs)**					
		no-refine		refine		Δ	
Reference	UnknownVar	62 / 106	17 / 60	63 / 78	17 / 31	+1 / -28	/ -29
	DuplicatedVar		45 / 46		46 / 47		+1 / +1
Arity	MissingParam	4 / 4	4 / 4	4 / 4	4 / 4	/	/
Assertion	Assertion	4 / 56	4 / 56	4 / 31	4 / 31	/ -25	/ -25
Operand	NoNumber	22 / 113	2 / 65	22 / 44	2 / 6	/ -69	/ -59
	Abrupt		20 / 48		20 / 38		/ -10
Total		92 / 279 (33.0%)		93 / 157 (59.2%)		+1 / -122 (+26.3%)	



Name	**Feature**	**#**	**Checker**	**Created**	**Life Span**
ES12-1	Switch	3	Reference	2015-09-22	1,996 days
ES12-2	Try	3	Reference	2015-09-22	1,996 days
ES12-3	Arguments	1	Reference	2015-09-22	1,996 days
ES12-4	Array	2	Reference	2015-09-22	1,996 days
ES12-5	Async	1	Reference	2015-09-22	1,996 days
ES12-6	Class	1	Reference	2015-09-22	1,996 days
ES12-7	Branch	1	Reference	2015-09-22	1,996 days
ES12-8	Arguments	2	Operand	2015-12-16	1,910 days

14 New Bugs In ES2021

[ICSE 2021] J. Park et al., “JEST: N +1-version Differential Testing of Both JavaScript Engines and Specification”


ASE 2021
Specification Type Errors
JSTAR
Specification Type Analysis
ASE 2020
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
Mechanized Spec. Extraction
FSE 2022
JSAVER
Static Analyzer
Derivation of Static Analyzer
ICSE 2021
JavaScript Engines
Conformance Tests
JEST
PLDI 2023
FS/FCPS-Coverage
SIGSOFT Distinguished Paper
Conformance Test Synthesis
Feature-Sensitive Coverage

Conformance of JavaScript Engines

ecma INTERNATIONAL

Standard ECMA-262

11th Edition / June 2020

ECMAScript® 2020 Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

ECMA-262

(JavaScript Spec.)

Conformance

GraalVM™

QuickJS

moddable

JavaScript Engines

Conformance of JavaScript Engines

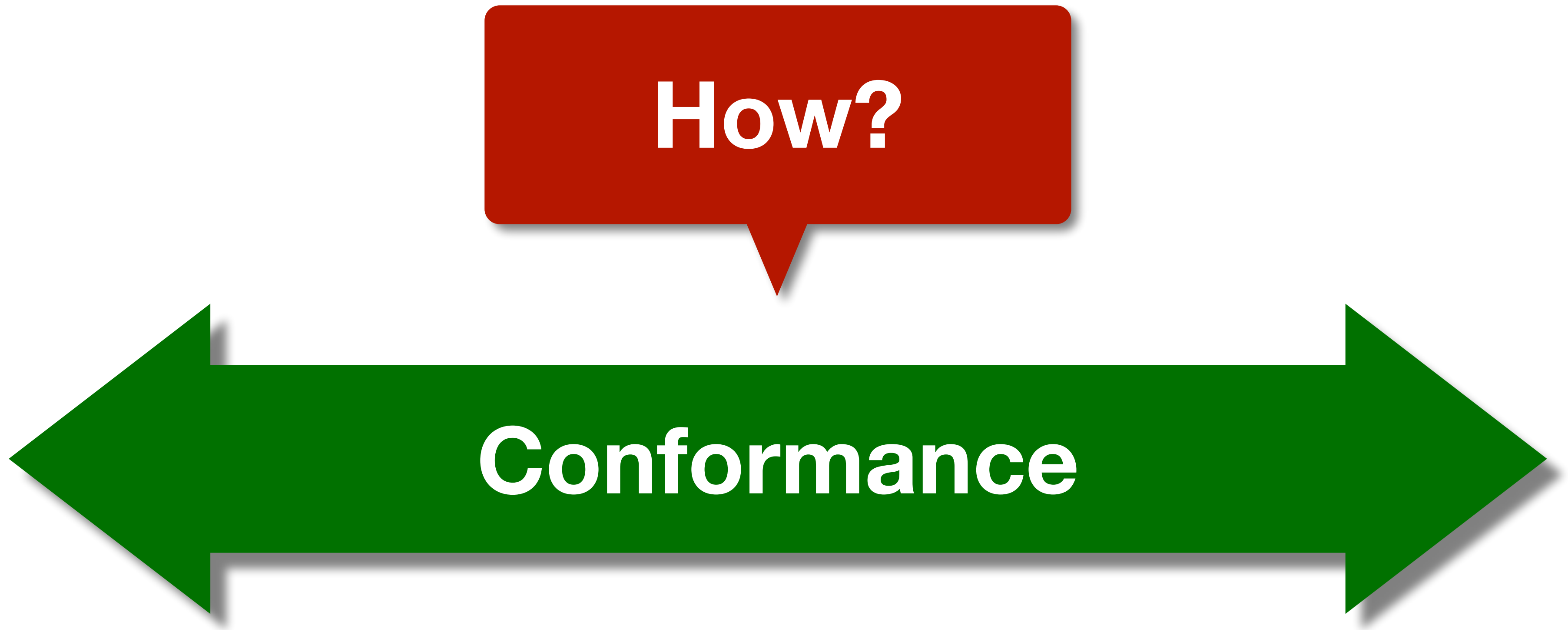
ecma INTERNATIONAL

Standard ECMA-262
11th Edition / June 2020

ECMAScript® 2020
Language Specification

Standard

ECMA-262
(JavaScript Spec.)


How?

Conformance

GraalVM™

QuickJS

moddable

JavaScript Engines

Conformance of JavaScript Engines

ecma INTERNATIONAL

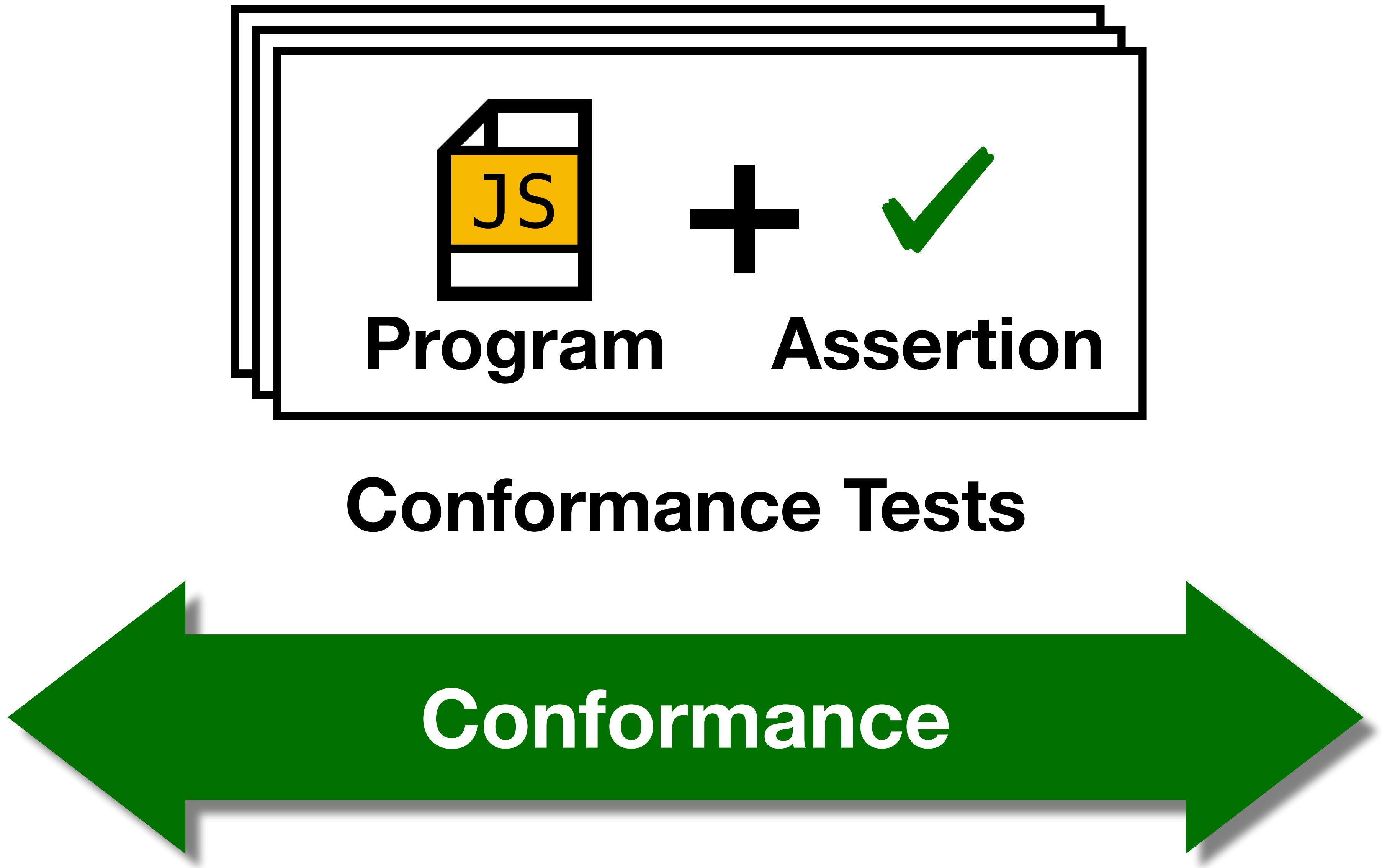
Standard ECMA-262

11th Edition / June 2020

ECMAScript® 2020 Language Specification

Standard

ECMA-262
(JavaScript Spec.)



Conformance Tests

Test262
(Official Test Suite)

GraalVM™

QuickJS

moddable

JavaScript Engines

Conformance of JavaScript Engines

ecma INTERNATIONAL

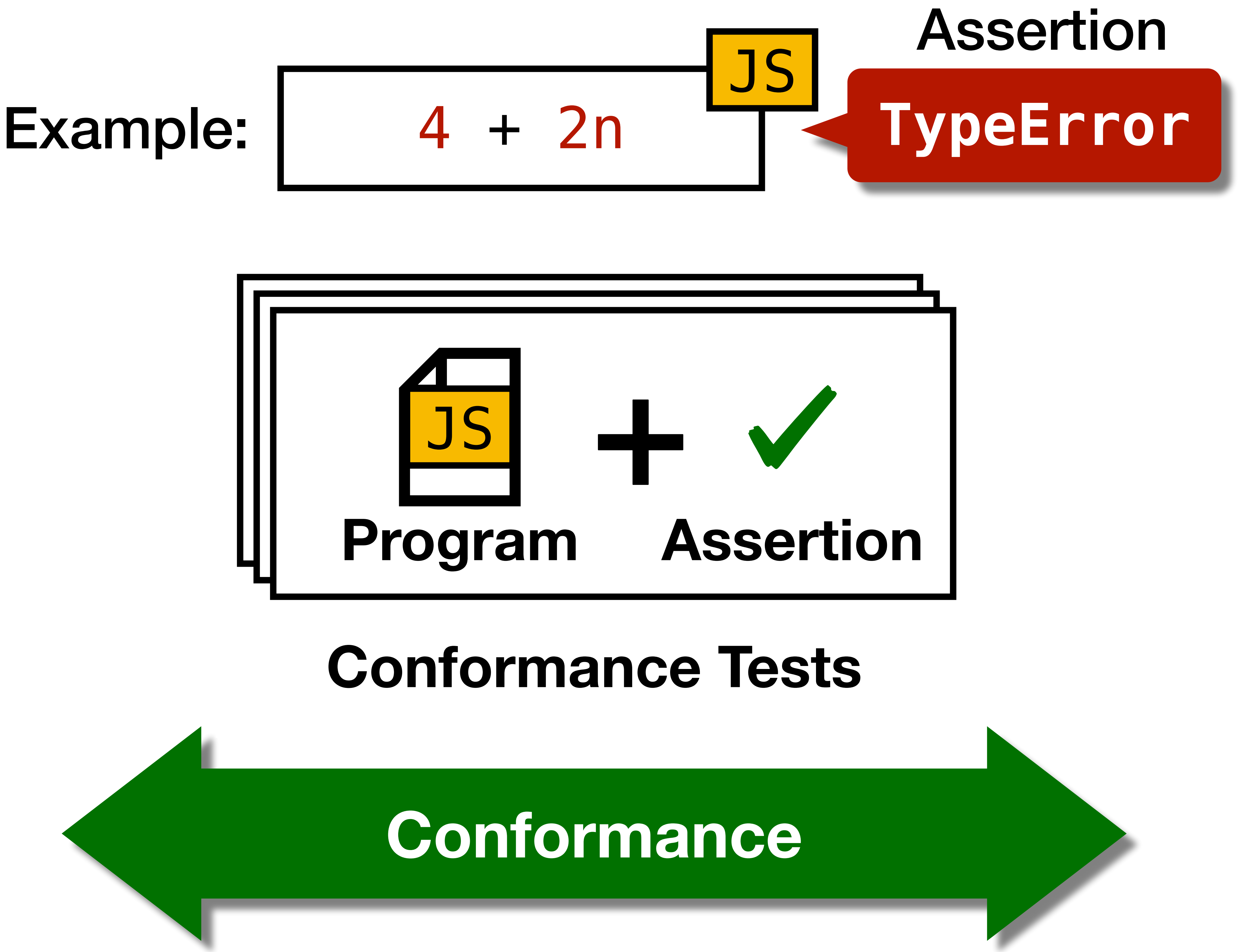
Standard ECMA-262

11th Edition / June 2020

ECMAScript® 2020 Language Specification

Standard

ECMA-262
(JavaScript Spec.)



Example:

```
4 + 2n
```

Assertion

```
TypeError
```

Conformance Tests

Test262
(Official Test Suite)

GraalVM

QuickJS

moddable

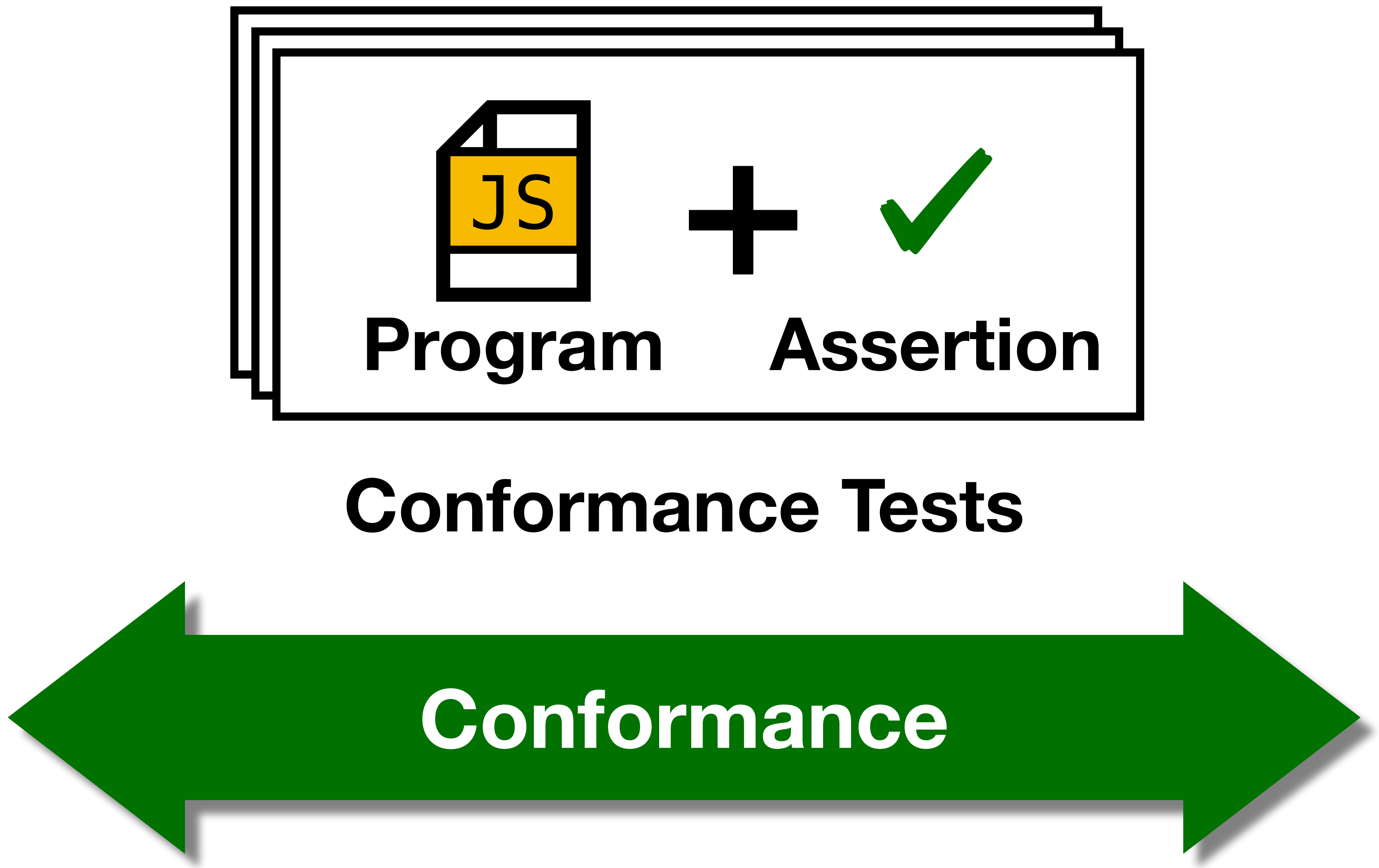
JavaScript Engines

Problem - Manual Approach

Example: `4 + 2n` JS

Assertion `TypeError`

ECMA-262
(JavaScript Spec.)



Test262
(Official Test Suite)

`Manual`

GraalVM

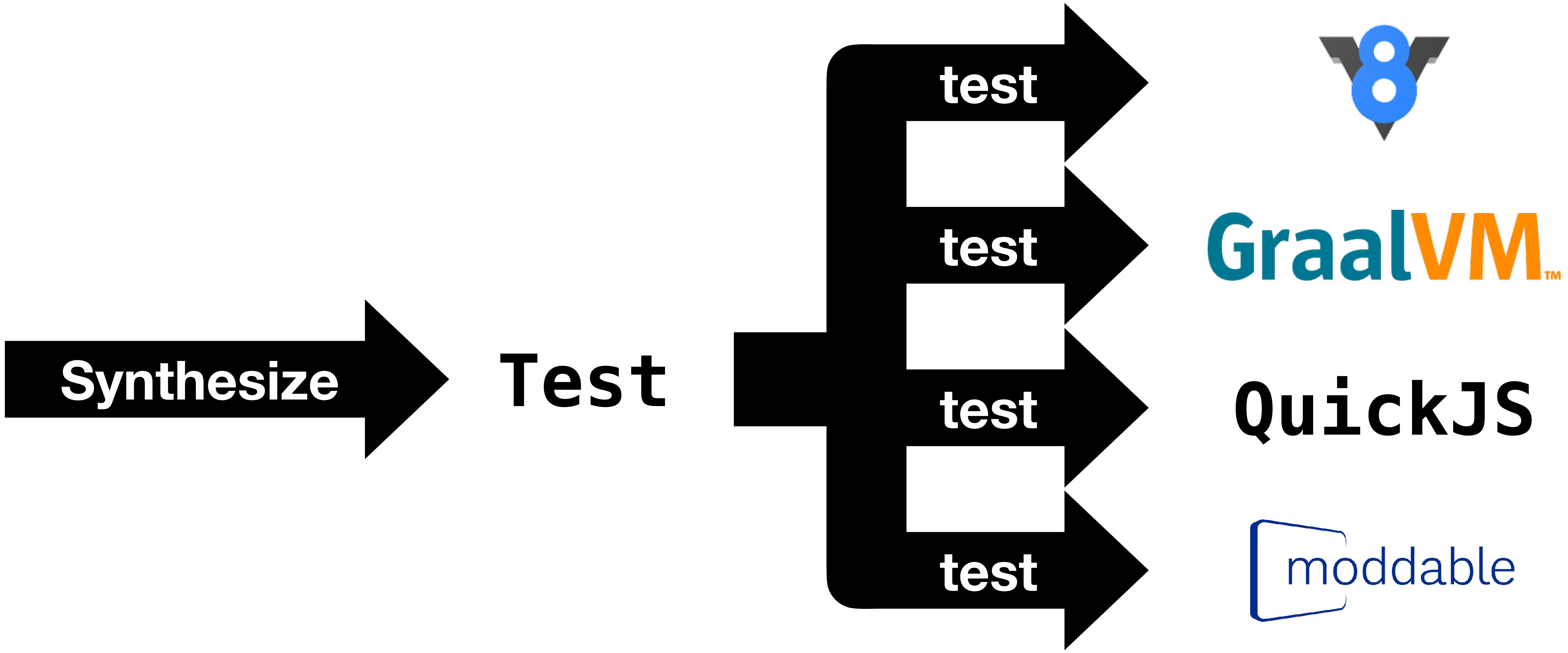
QuickJS

moddable

JavaScript Engines

N+1-version Differential Testing

ECMA-262
(JavaScript Spec.)



JavaScript Engines

N+1-version Differential Testing

ecma INTERNATIONAL

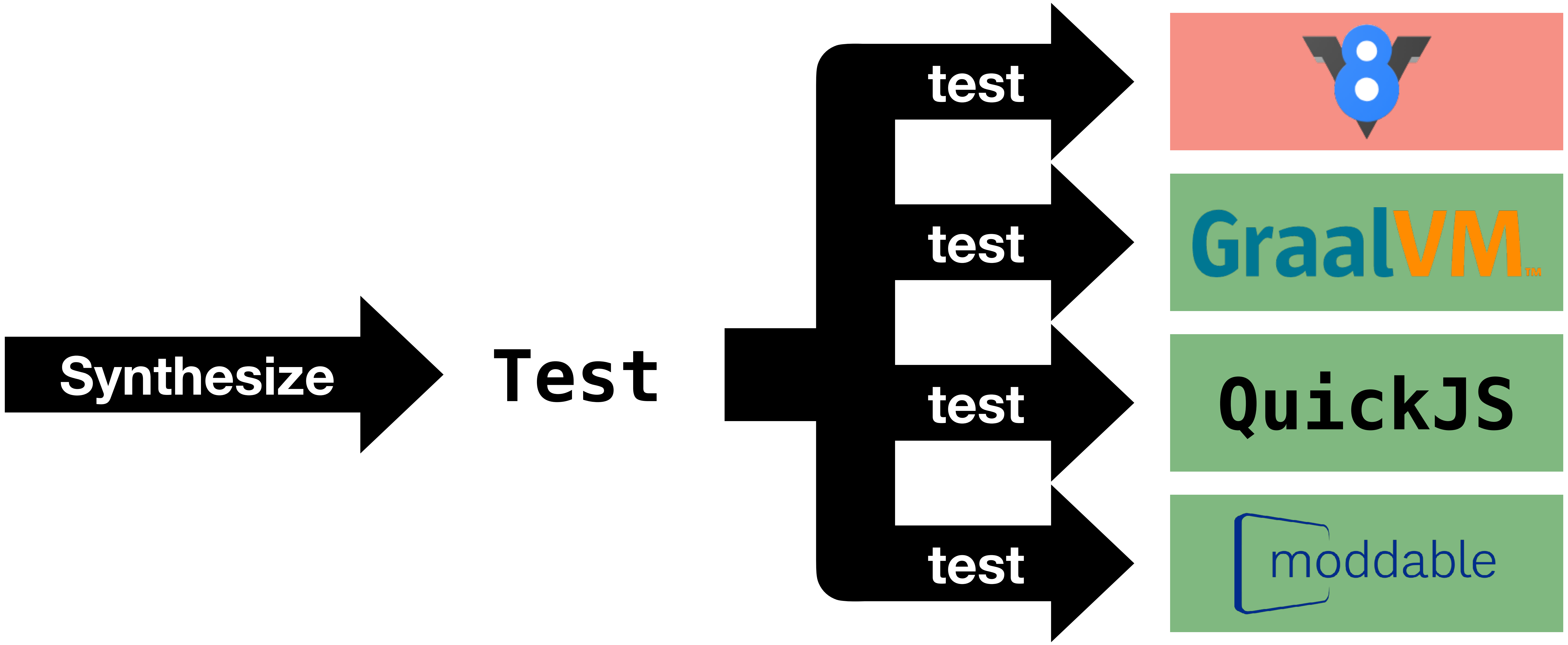
Standard ECMA-262

11th Edition / June 2020

ECMAScript® 2020 Language Specification

Standard

ECMA-262
(JavaScript Spec.)



Synthesize

Test

test

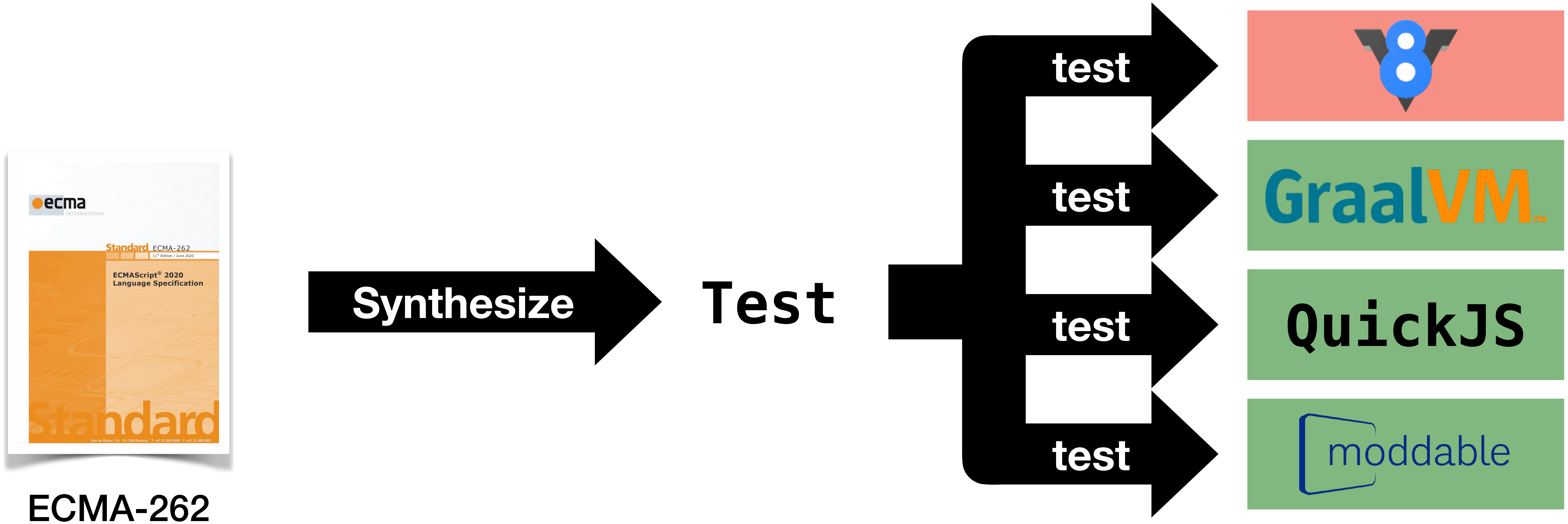
test

test

test

JavaScript
Engines

N+1-version Differential Testing



Synthesize

Test

test

test

test

test

QuickJS

ECMA-262
(JavaScript Spec.)

JavaScript
Engines

An engine bug in

N+1-version Differential Testing

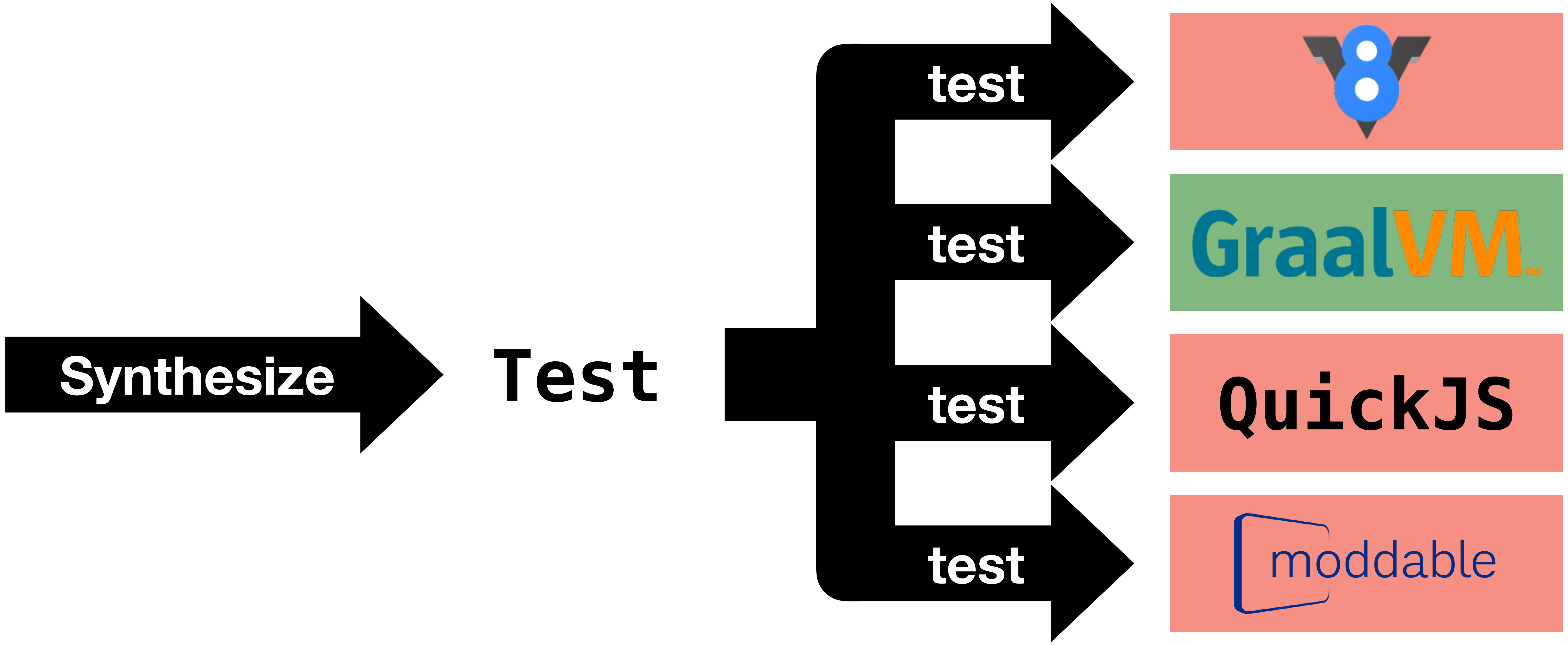
ecma

Standard ECMA-262

ECMAScript® 2020 Language Specification

Standard

ECMA-262
(JavaScript Spec.)



Synthesize

Test

test

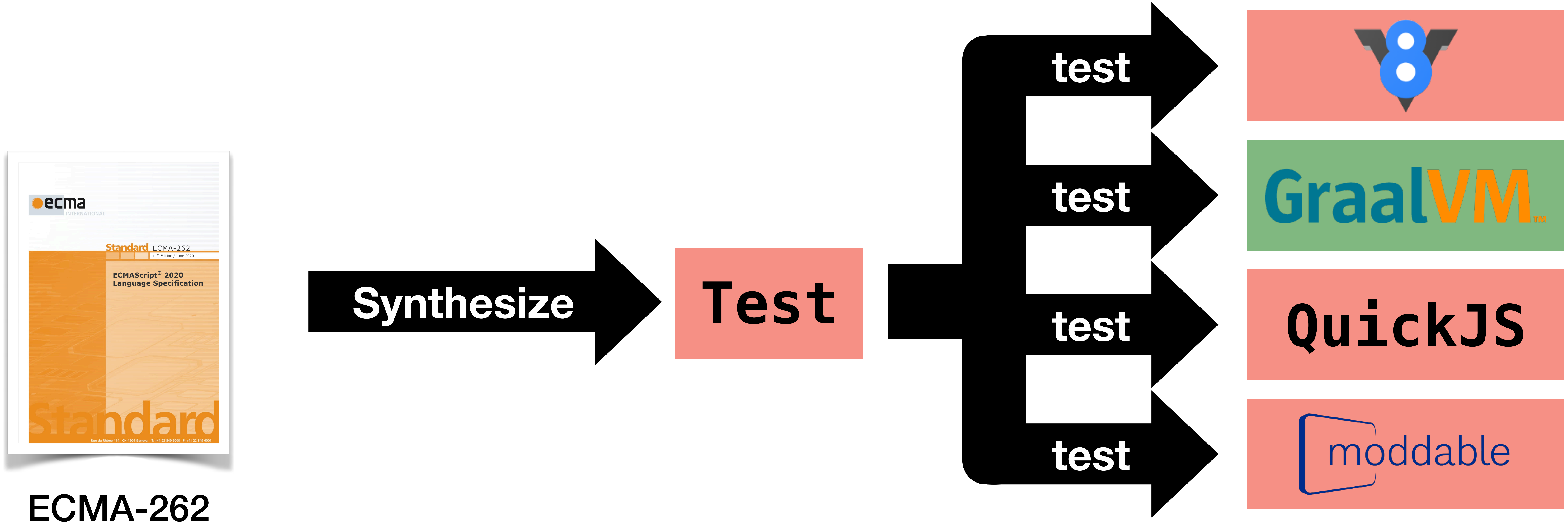
test

test

test

JavaScript Engines

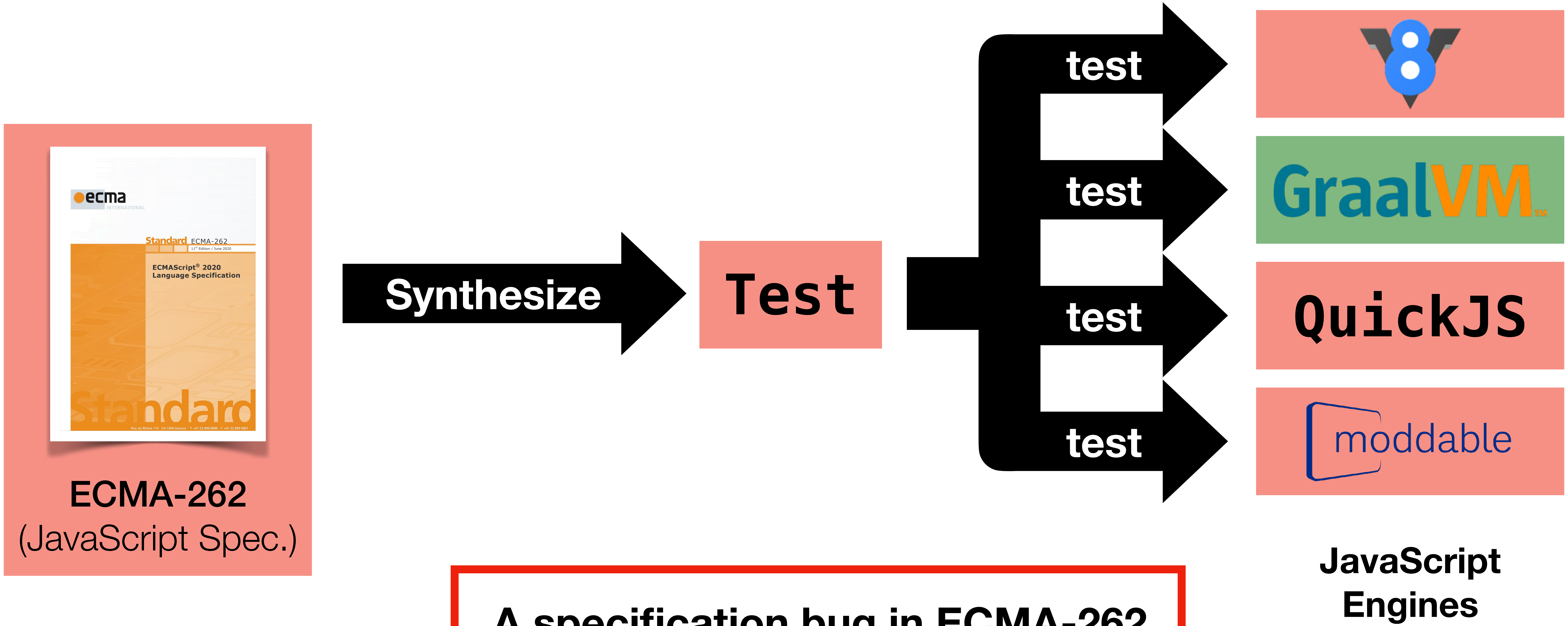
N+1-version Differential Testing



ECMA-262
(JavaScript Spec.)

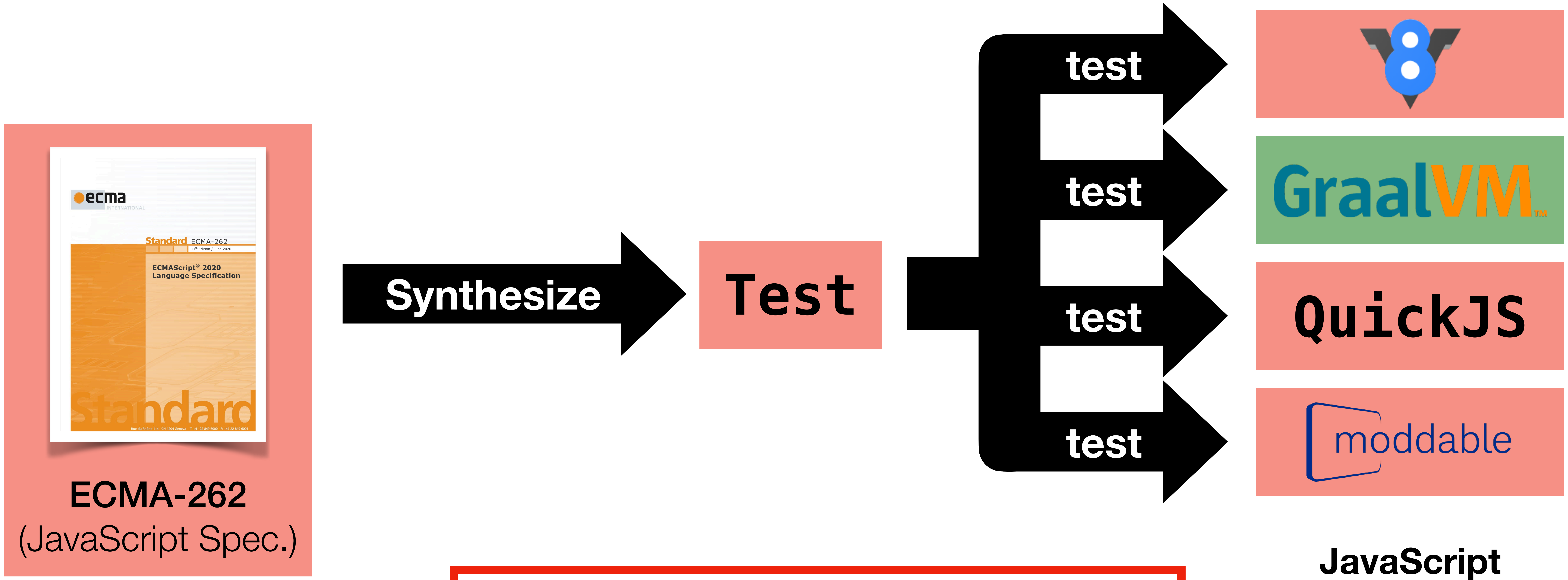
JavaScript Engines

N+1-version Differential Testing



A specification bug in ECMA-262

N+1-version Differential Testing



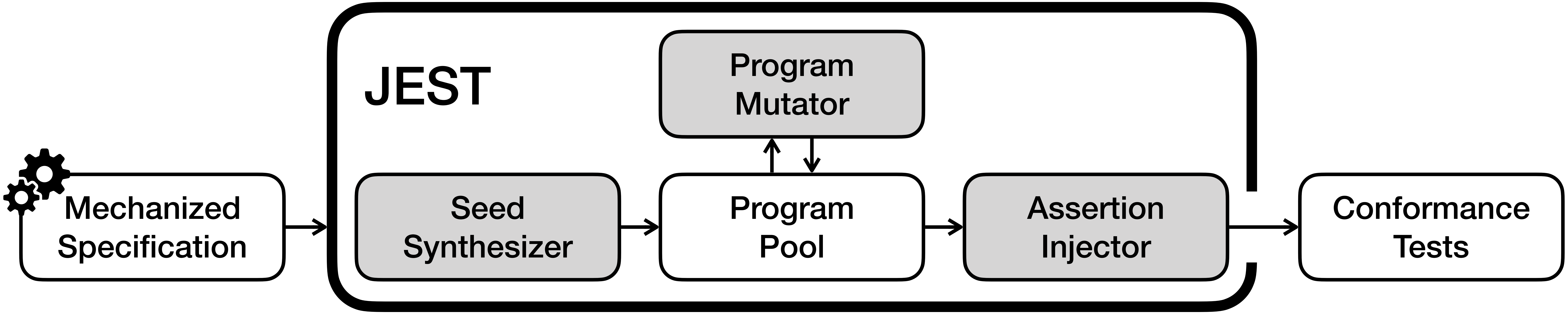
JavaScript Engines

A specification bug in ECMA-262

An engine bug in GraalVM

JEST

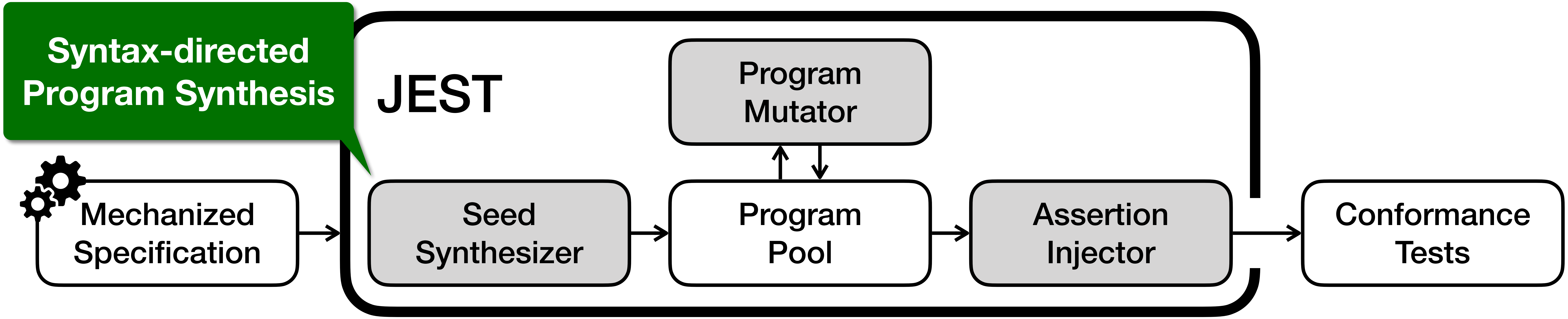
(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

. . .

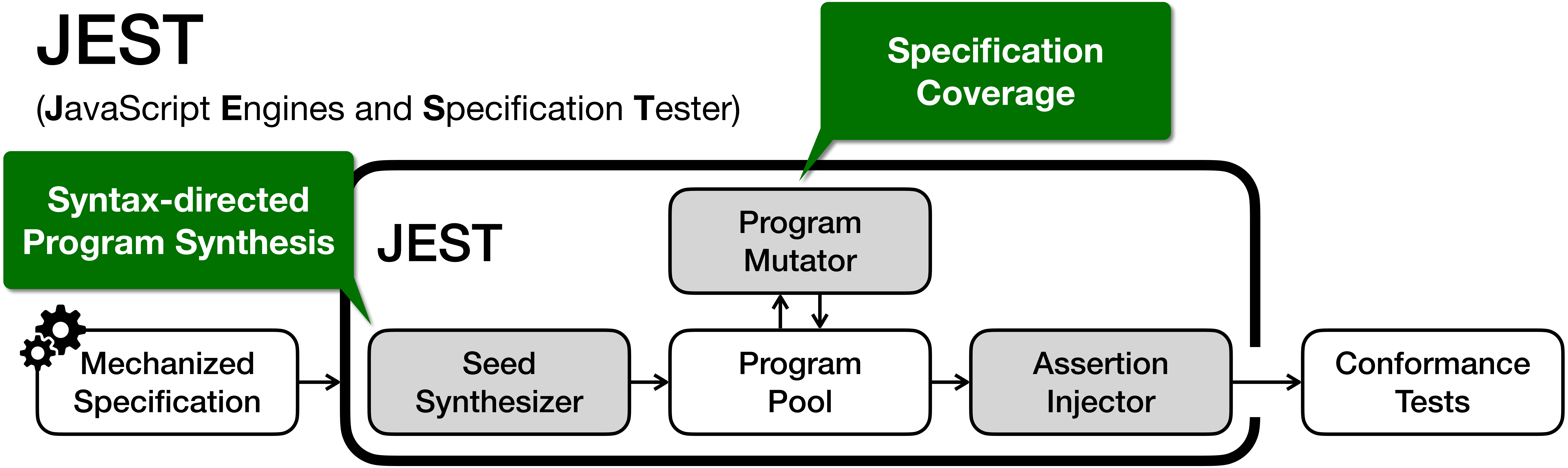
```
let x = 42;
```

. . .

. . .

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

. . .

```
let x = 1 + 2;
```

. . .

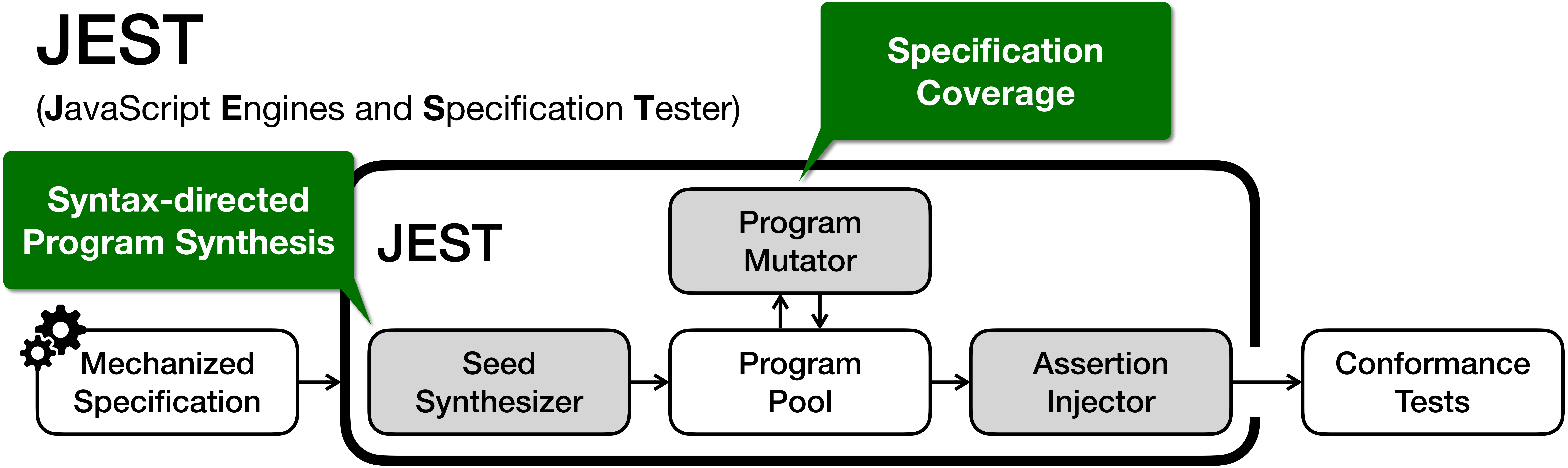
```
let x = 42;
```

. . .

. . .

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

. . .

`let x = 1 + 2;`

. . .

`let x = 42;`

. . .

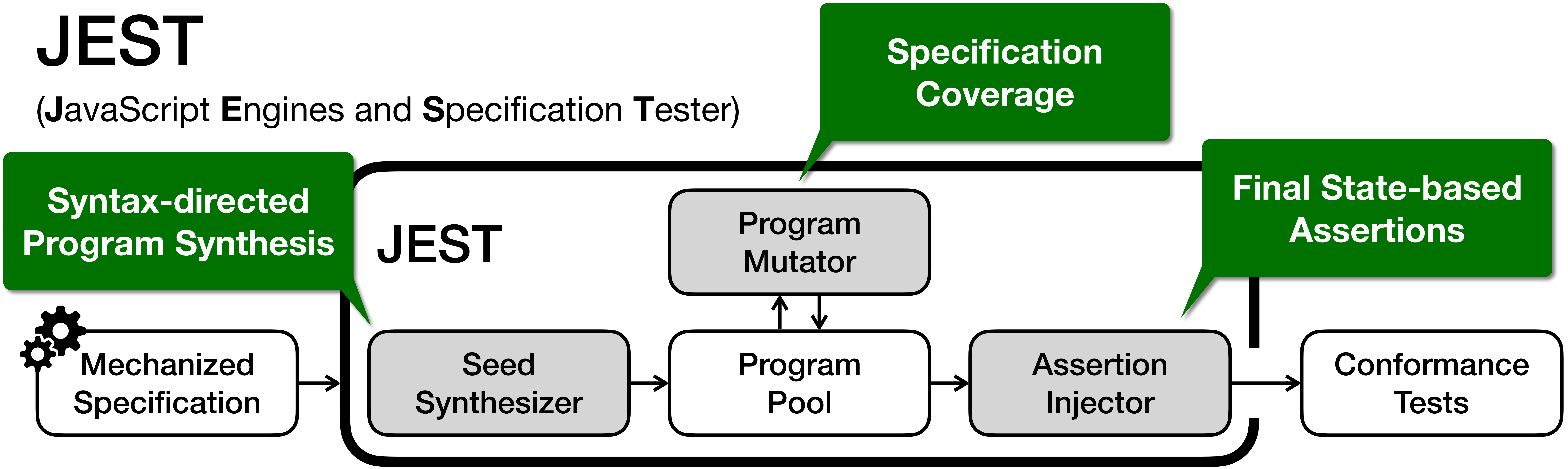
`let x = ![];`

. . .

. . .

JEST

(**J**avaScript **E**ngines and **S**pecification **T**ester)



Program Pool

• • •

```
let x = 1 + 2;
assert(x == 3);
```

• • •

```
let x = 42;
assert(x == 42);
```

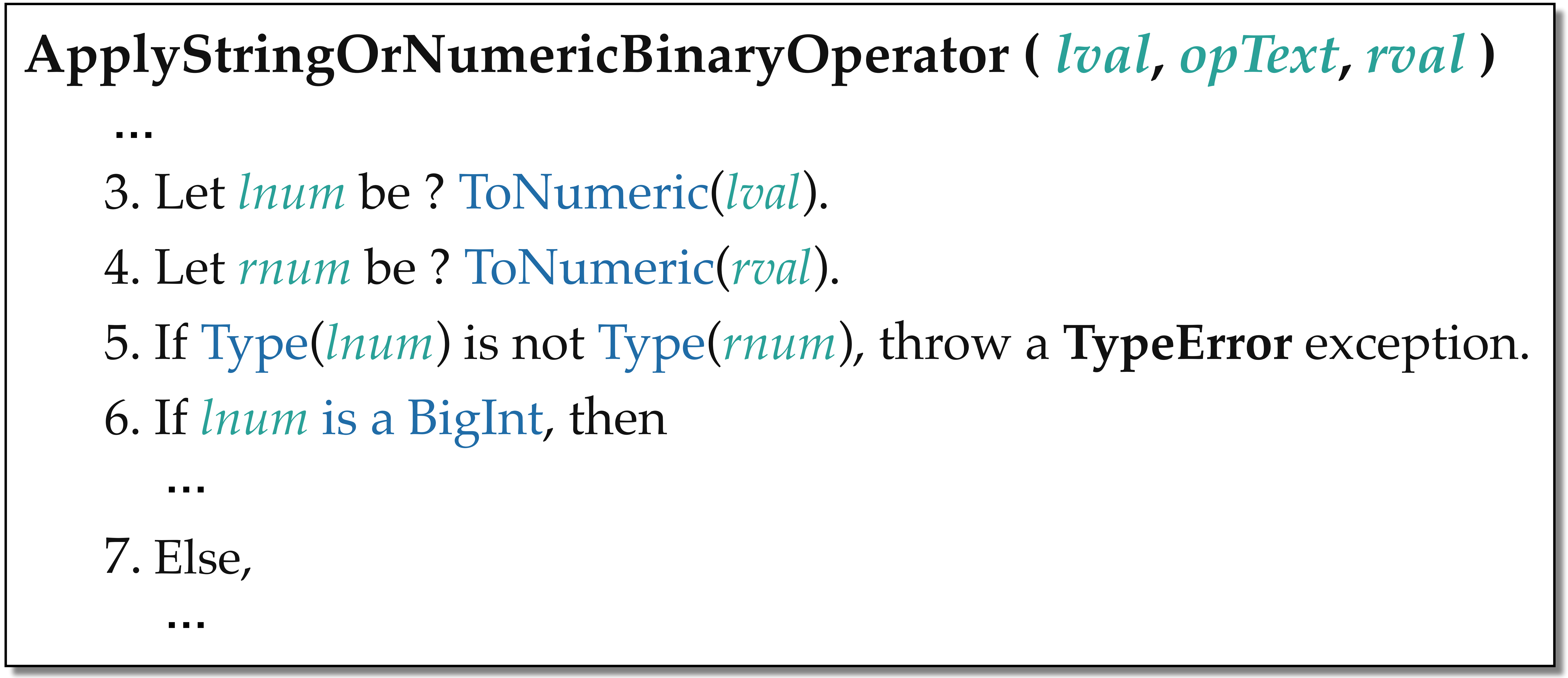
• • •

```
let x = ![];
assert(x == false);
```

• • •

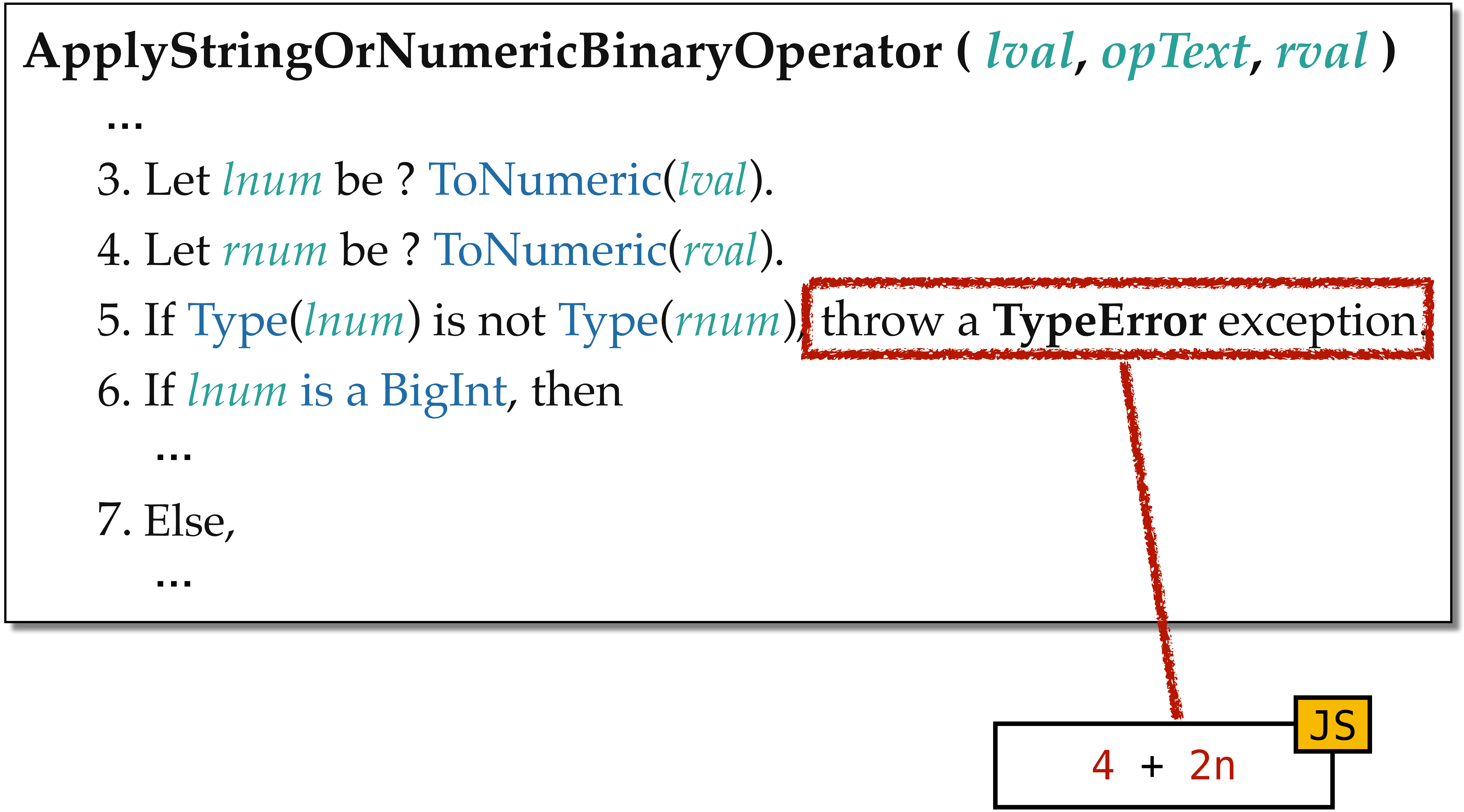
• • •

JEST - Specification Coverage



```
ApplyStringOrNumericBinaryOperator ( lval, opText, rval )
    ...
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    6. If lnum is a BigInt, then
        ...
    7. Else,
        ...
```

JEST - Specification Coverage

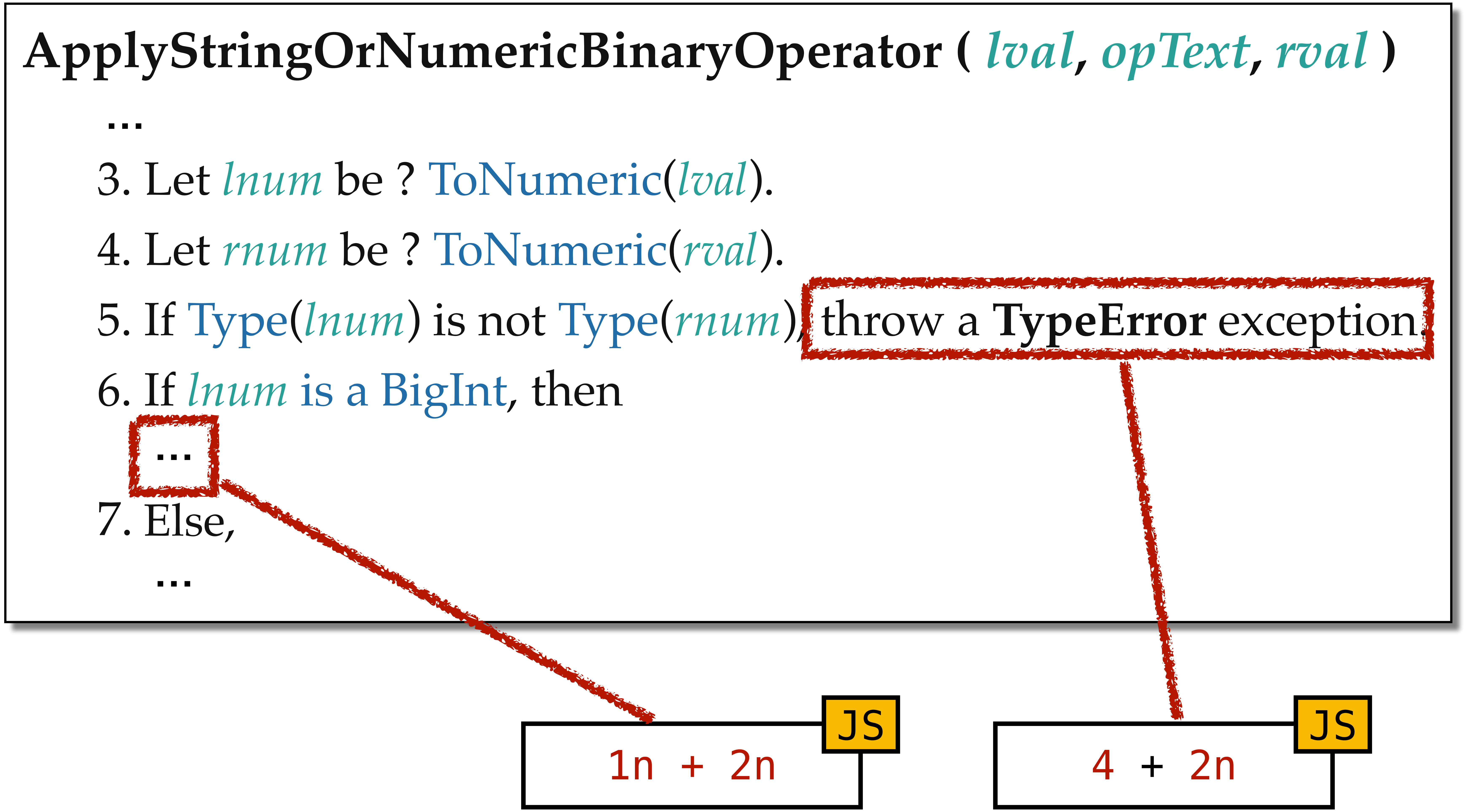


```
ApplyStringOrNumericBinaryOperator ( lval, opText, rval )
    …
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    6. If lnum is a BigInt, then
        …
    7. Else,
        …
```

JS

```
4 + 2n
```

JEST - Specification Coverage

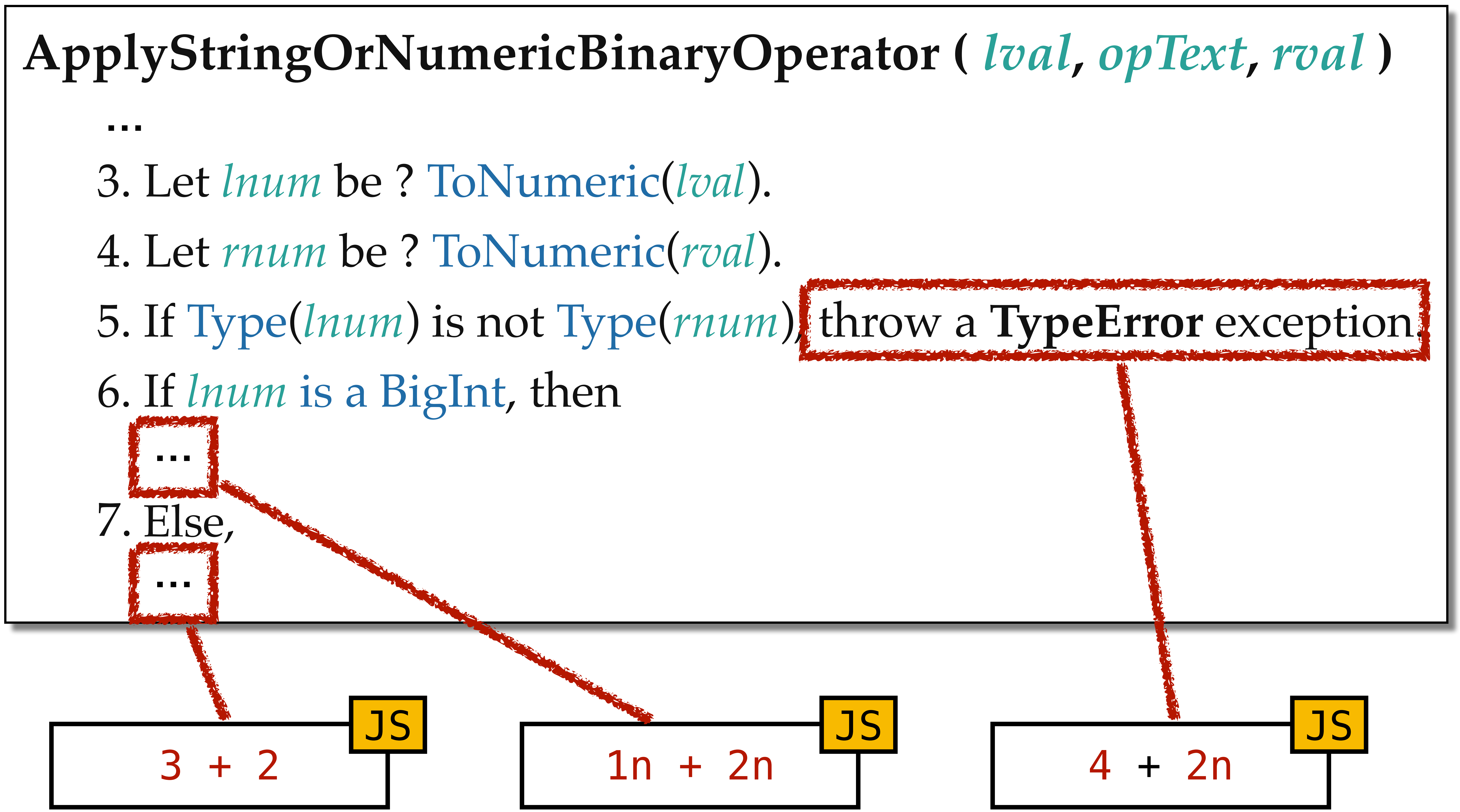


```
ApplyStringOrNumericBinaryOperator ( lval, opText, rval )
   ...
   3. Let lnum be ? ToNumeric(lval).
   4. Let rnum be ? ToNumeric(rval).
   5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
   6. If lnum is a BigInt, then
      ...
   7. Else,
      ...
```

```
1n + 2n
```

```
4 + 2n
```

JEST - Specification Coverage



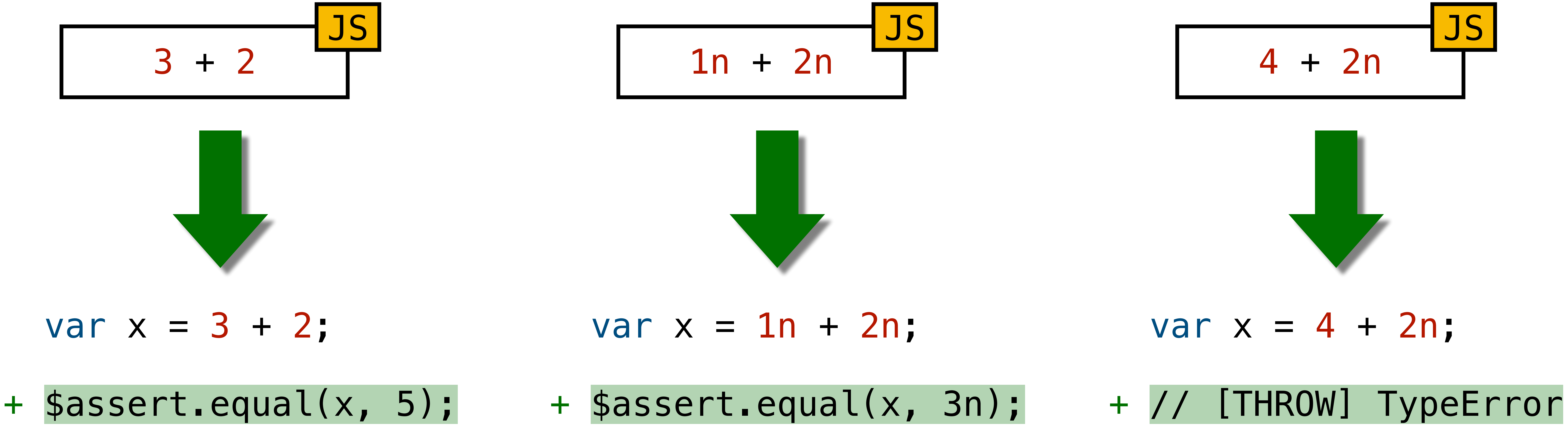
```
ApplyStringOrNumericBinaryOperator ( lval, opText, rval )
    ...
    3. Let lnum be ? ToNumeric(lval).
    4. Let rnum be ? ToNumeric(rval).
    5. If Type(lnum) is not Type(rnum), throw a TypeError exception.
    6. If lnum is a BigInt, then
        ...
    7. Else,
        ...
```

```
3 + 2
```

```
1n + 2n
```

```
4 + 2n
```

JEST - Final State-based Assertion Injection

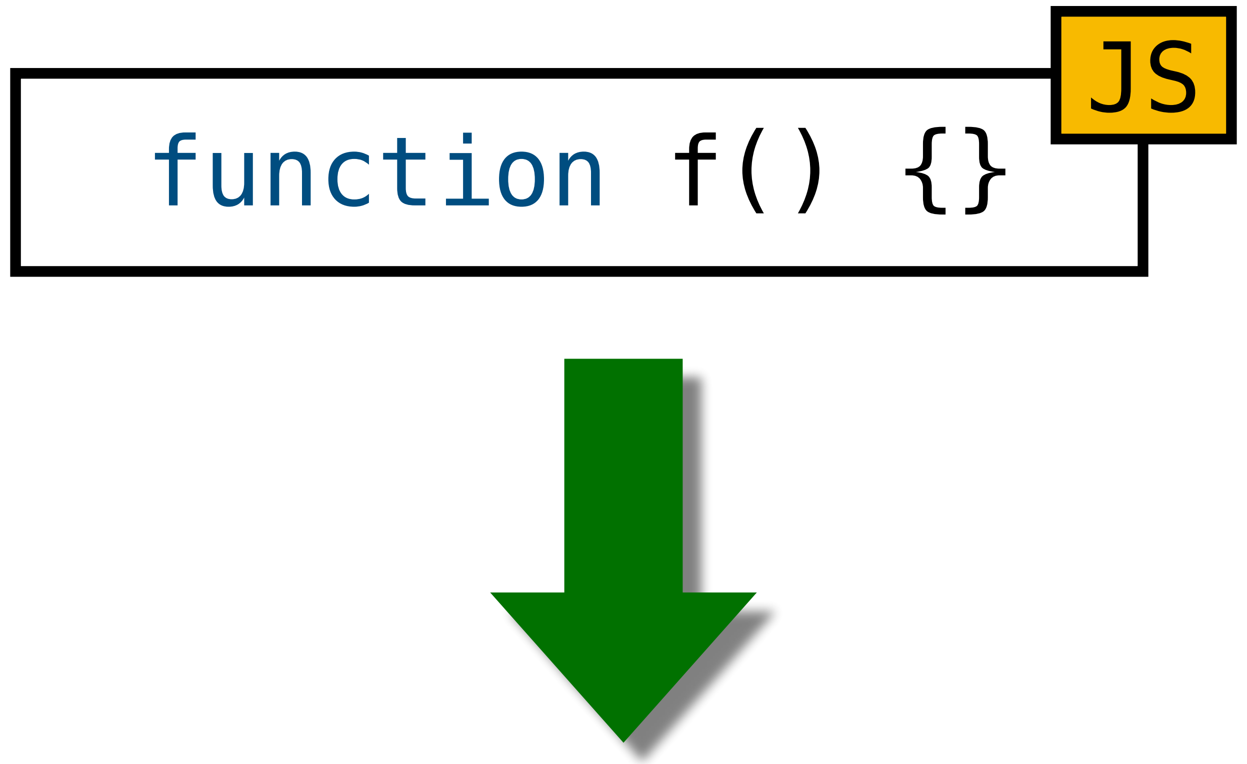


```
  var x = 3 + 2;
+ $assert.equal(x, 5);
```

```
  var x = 1n + 2n;
+ $assert.equal(x, 3n);
```

```
  var x = 4 + 2n;
+ // [THROW] TypeError
```

JEST - Final State-based Assertion Injection



```
function f() {}
```

```
  function f() {}

+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);

+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

JEST - Final State-based Assertion Injection

JS

```
function f() {}
```

Prototype Chain

```
  function f() {}

+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);

+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

JEST - Final State-based Assertion Injection

JS

```
function f() {}
```

Prototype Chain

Property Descriptor

```
  function f() {}

+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);

+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

JEST - Final State-based Assertion Injection

JS

```
function f() {}
```

```
  function f() {}

+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);

+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

Prototype Chain

Property Descriptor

Property Order

JEST - Final State-based Assertion Injection

JS

```
function f() {}
```

```
  function f() {}

+ $assert.equal(Object.getPrototypeOf(f), Function.prototype);

+ $assert.verifyProperty(f, "prototype", {
+   writable: true,
+   enumerable: false,
+   configurable: false,
+ });

+ $assert.compare(Reflect.ownKeys(f), ['length', 'name', 'prototype'], f);

+ ...
```

Prototype Chain

Property Descriptor

Property Order

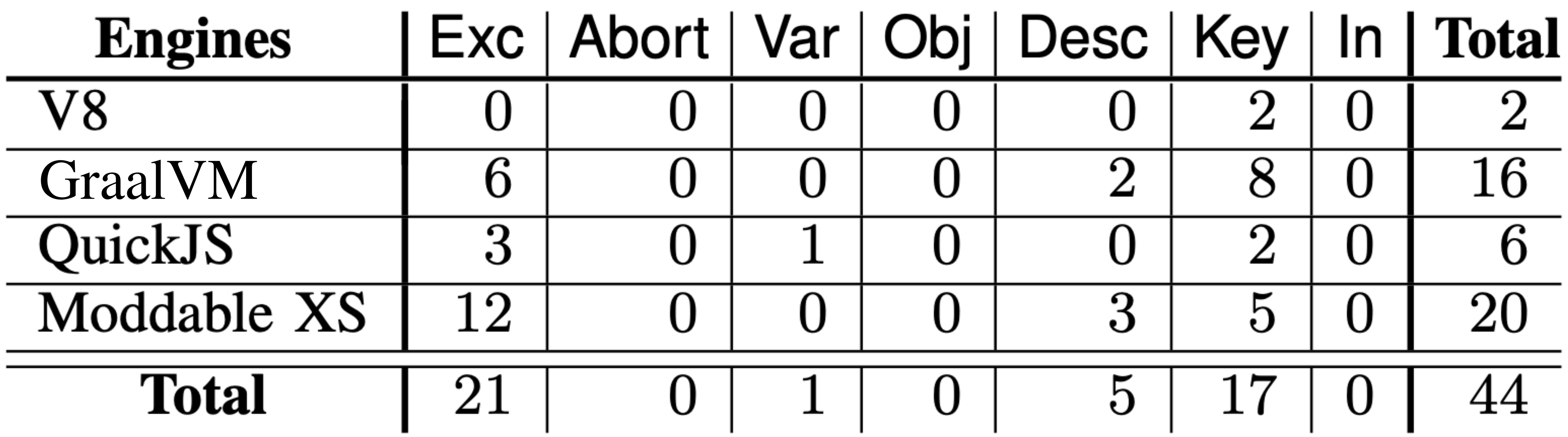
Etc.

JEST - Evaluation

- JEST synthesized **1,700 conformance tests** from ES2020

44 Bugs In Engines

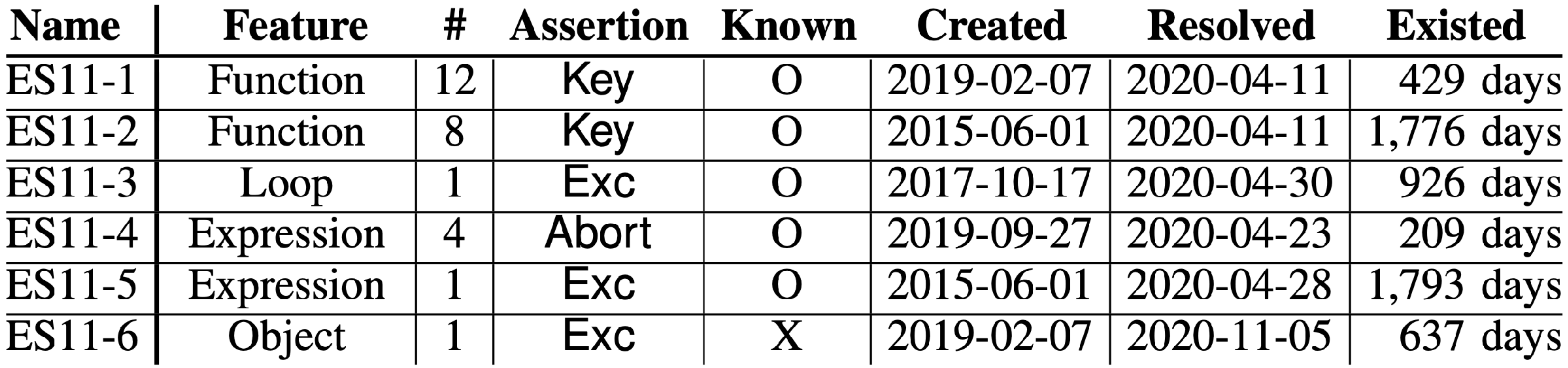
TABLE II: The number of engine bugs detected by JEST



Engines	Exc	Abort	Var	Obj	Desc	Key	In	Total
V8	0	0	0	0	0	2	0	2
GraalVM	6	0	0	0	2	8	0	16
QuickJS	3	0	1	0	0	2	0	6
Moddable XS	12	0	0	0	3	5	0	20
Total	21	0	1	0	5	17	0	44

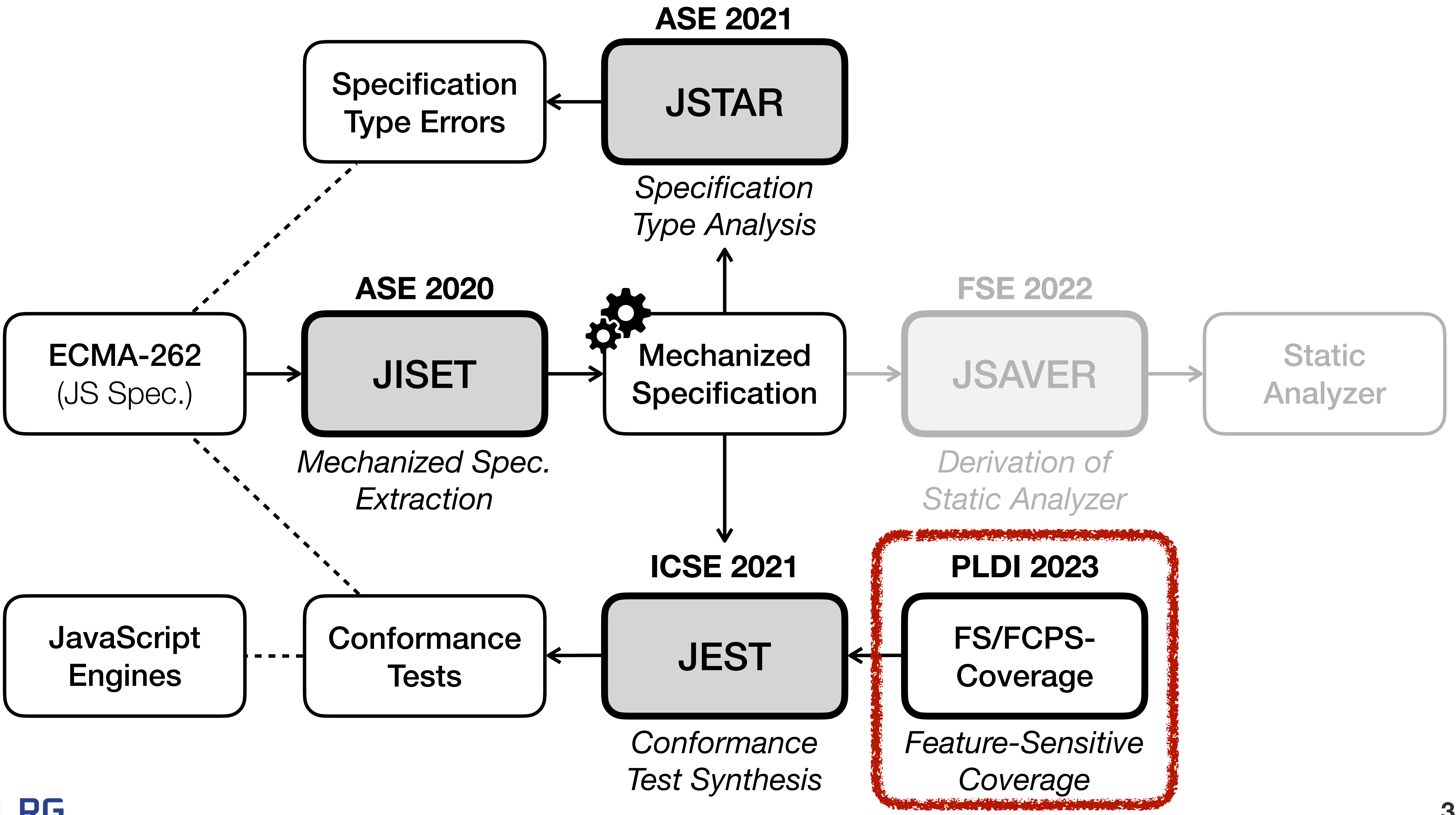
27 Bugs In Spec.

TABLE III: Specification bugs in ECMAScript 2020 (ES11) detected by JEST

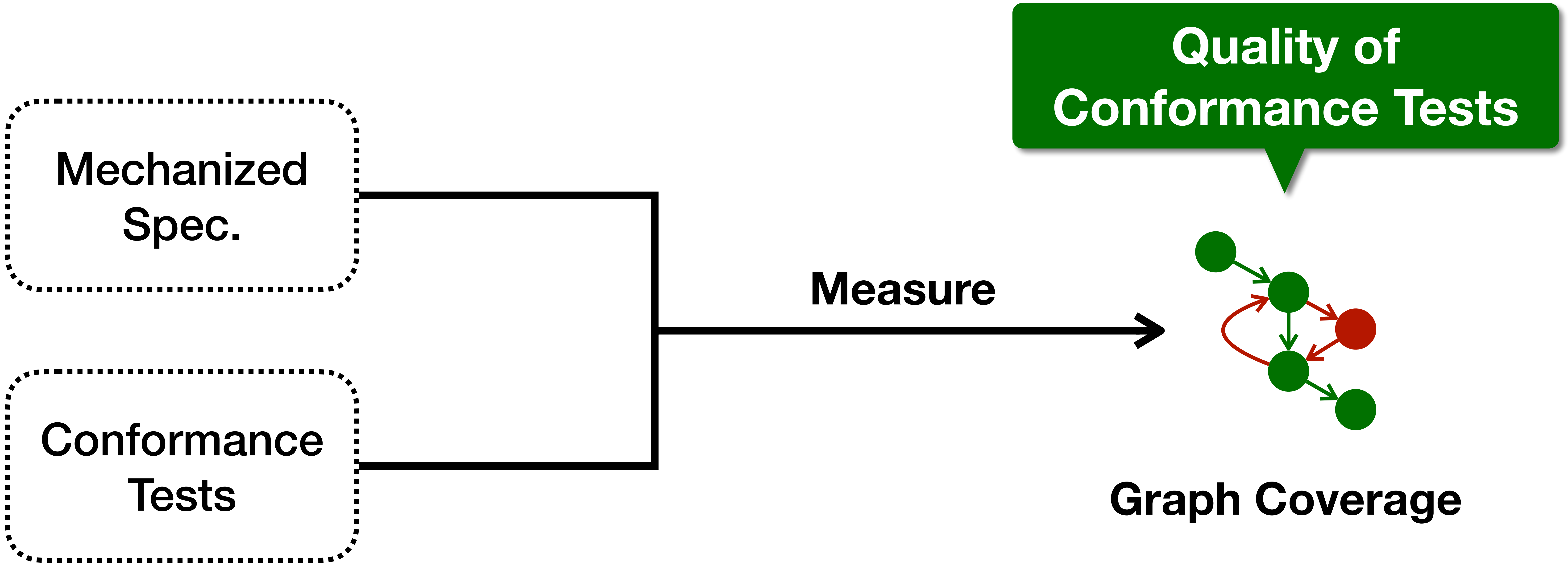


Name	Feature	#	Assertion	Known	Created	Resolved	Existed
ES11-1	Function	12	Key	O	2019-02-07	2020-04-11	429 days
ES11-2	Function	8	Key	O	2015-06-01	2020-04-11	1,776 days
ES11-3	Loop	1	Exc	O	2017-10-17	2020-04-30	926 days
ES11-4	Expression	4	Abort	O	2019-09-27	2020-04-23	209 days
ES11-5	Expression	1	Exc	O	2015-06-01	2020-04-28	1,793 days
ES11-6	Object	1	Exc	X	2019-02-07	2020-11-05	637 days

[PLDI 2023] J. Park et al., “Feature-Sensitive Coverage for Conformance Testing of Programming Language Implementations”



Graph Coverage for Language Specification



Mechanized Spec.

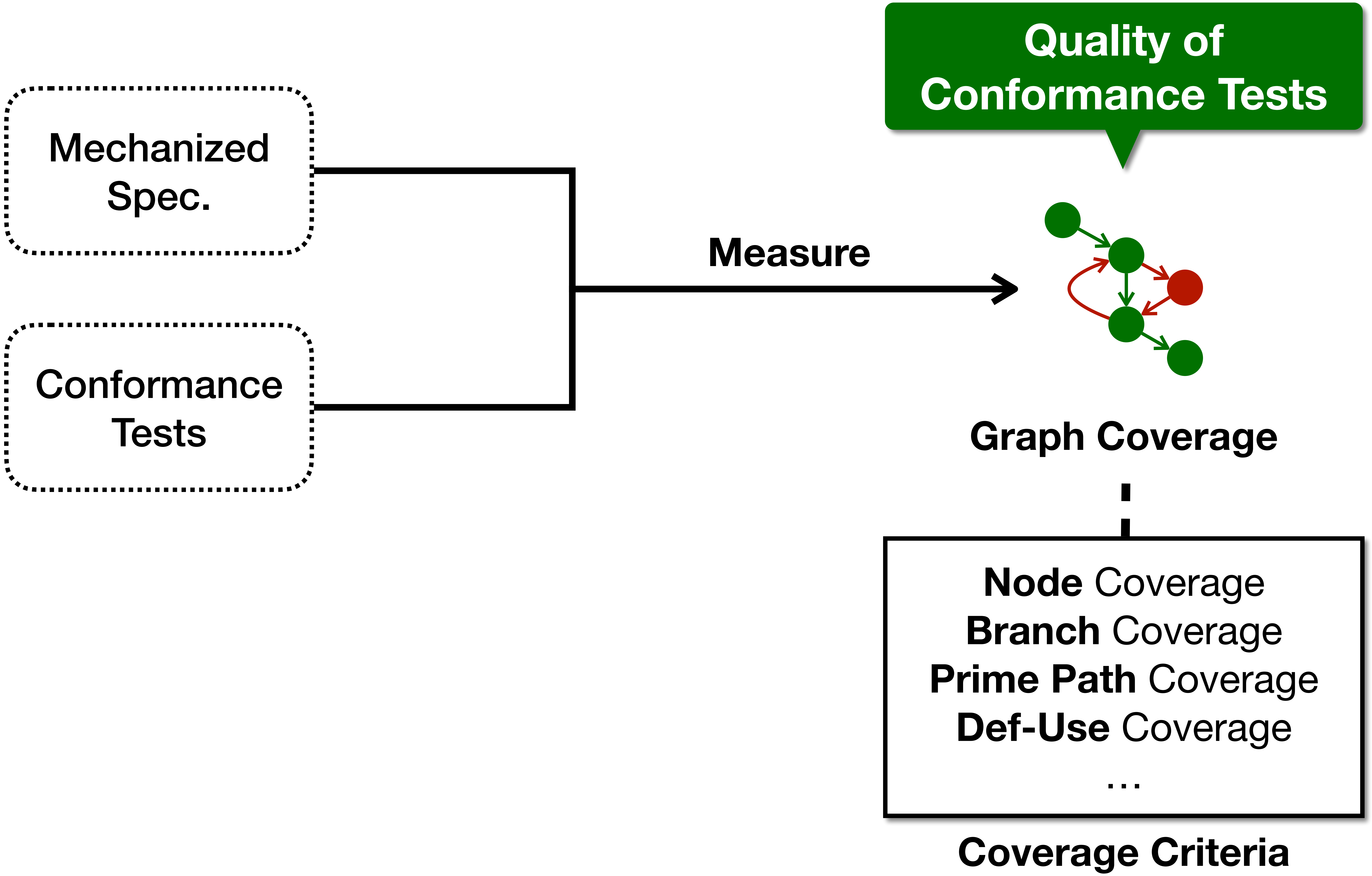
Conformance Tests

Measure

Quality of Conformance Tests

Graph Coverage

Graph Coverage for Language Specification



Quality of Conformance Tests

Mechanized Spec.

Conformance Tests

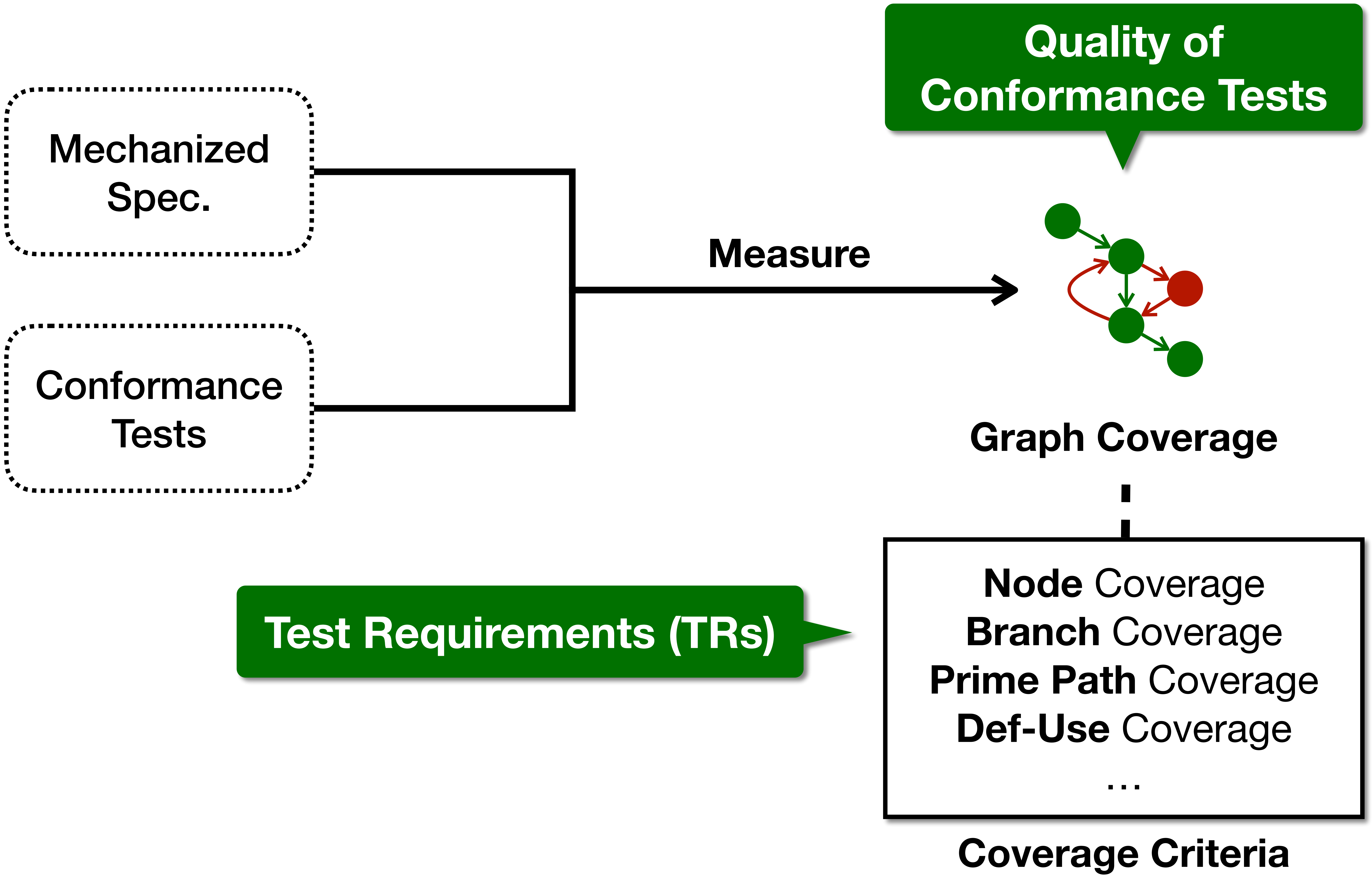
Measure

Graph Coverage

Node Coverage
Branch Coverage
Prime Path Coverage
Def-Use Coverage
…

Coverage Criteria

Graph Coverage for Language Specification



Quality of Conformance Tests

Mechanized Spec.

Conformance Tests

Measure

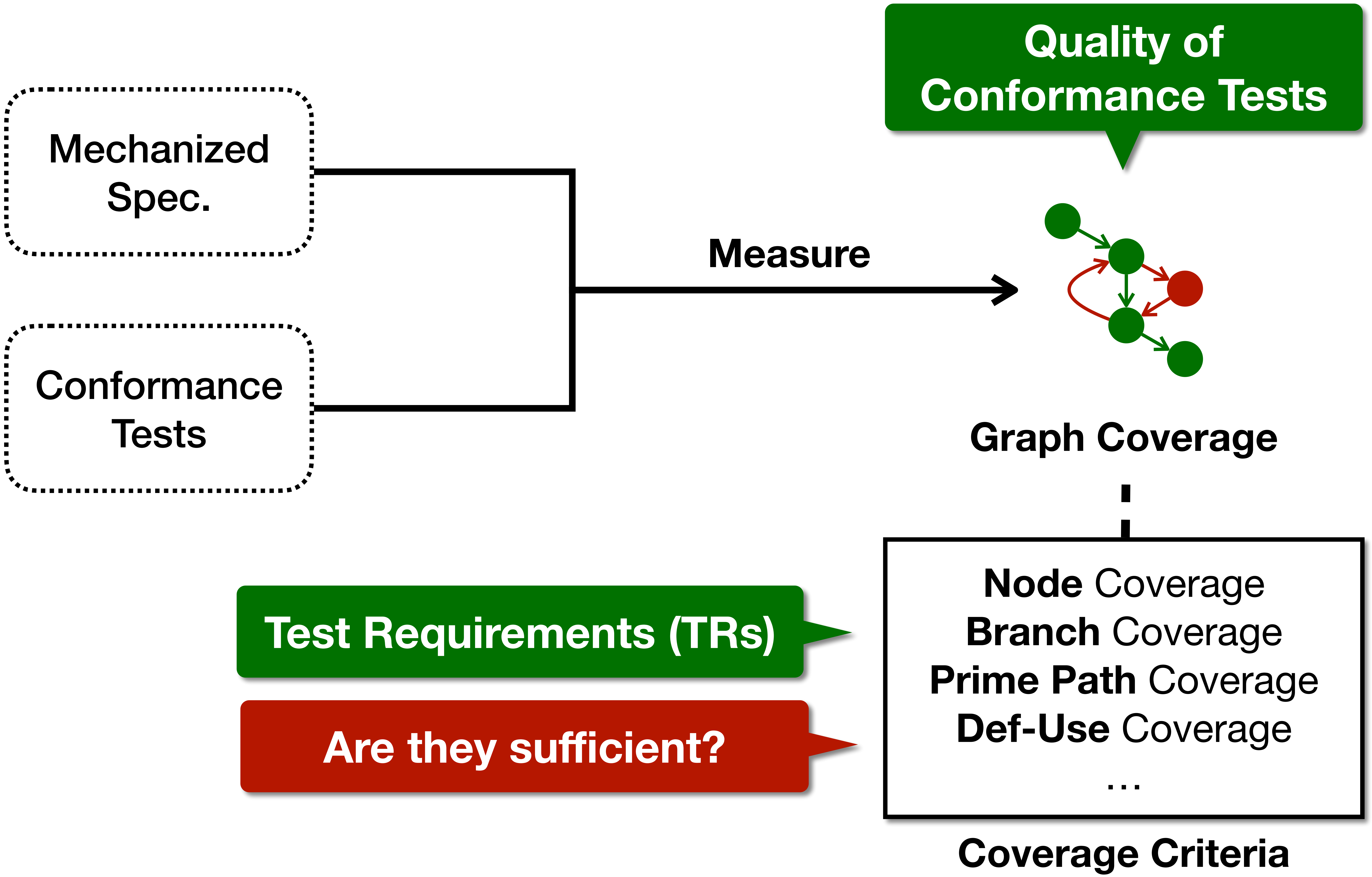
Graph Coverage

Test Requirements (TRs)

Node Coverage
Branch Coverage
Prime Path Coverage
Def-Use Coverage
…

Coverage Criteria

Graph Coverage for Language Specification



Mechanized Spec.

Conformance Tests

Measure

Quality of Conformance Tests

Graph Coverage

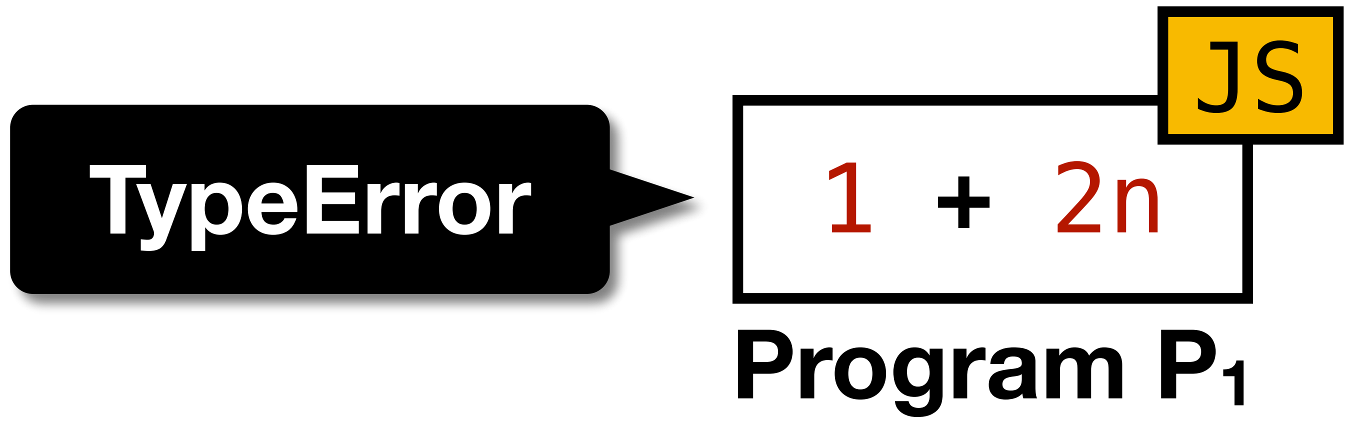
Node Coverage
Branch Coverage
Prime Path Coverage
Def-Use Coverage
…

Coverage Criteria

Test Requirements (TRs)

Are they sufficient?

Motivating Example 1 with Node Coverage



TypeError

JS

```
1 + 2n
```

Program P_1

Motivating Example 1 with **Node Coverage**

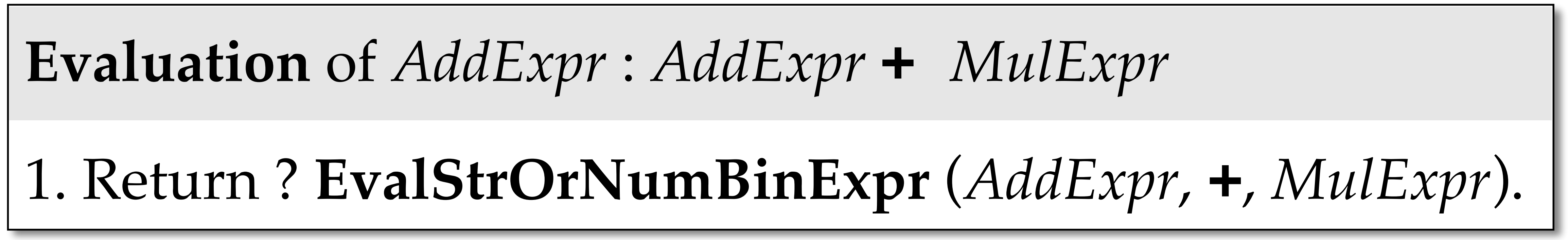
TypeError

JS

```
1 + 2n
```

Program P₁

feat **ADD**

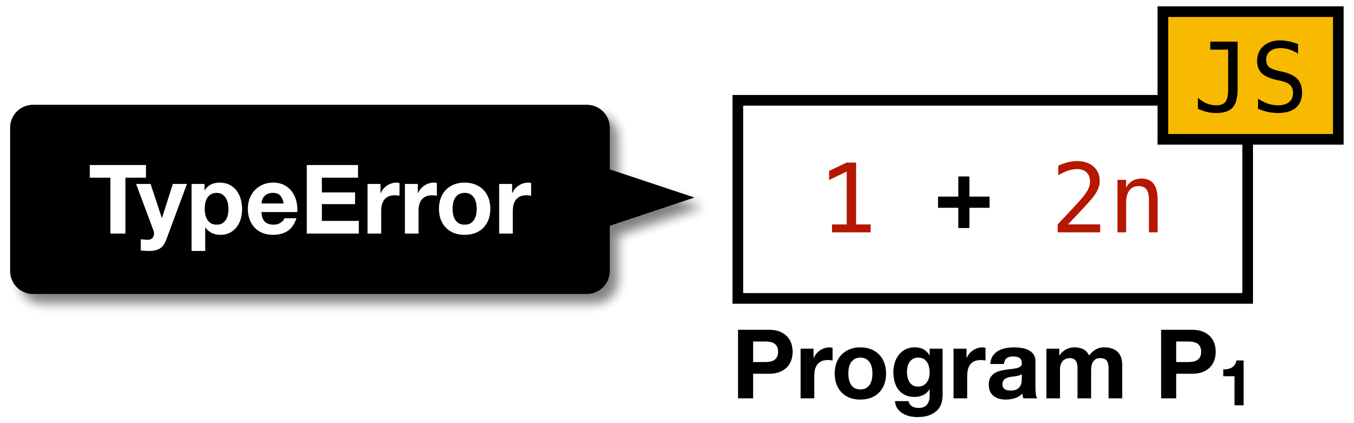


Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with Node Coverage

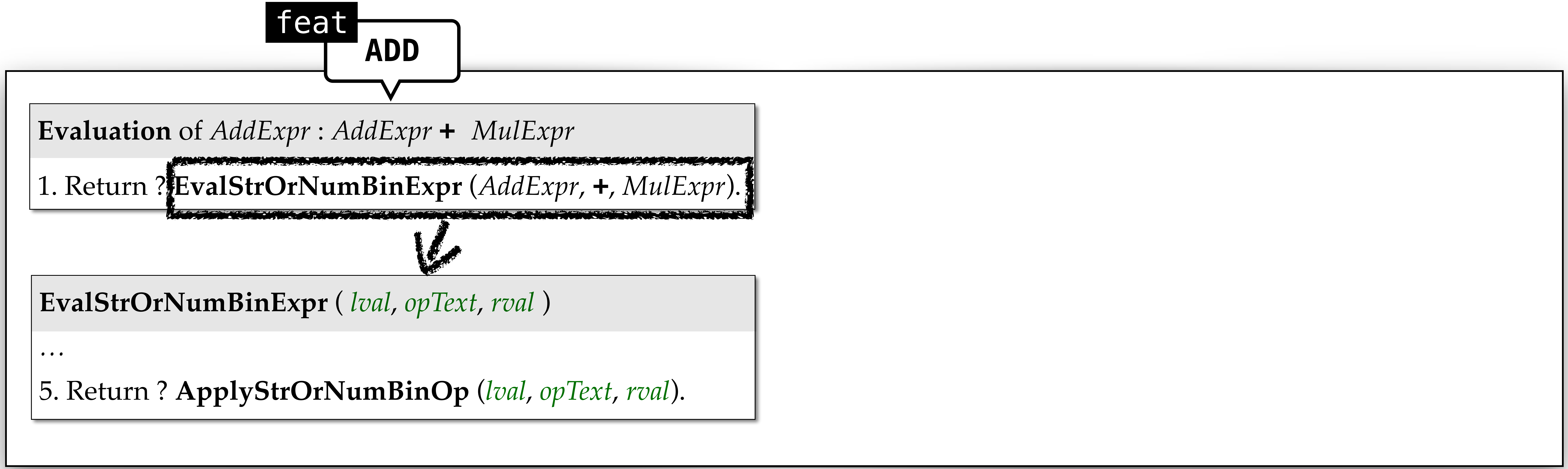


TypeError

JS

`1 + 2n`

Program P₁



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

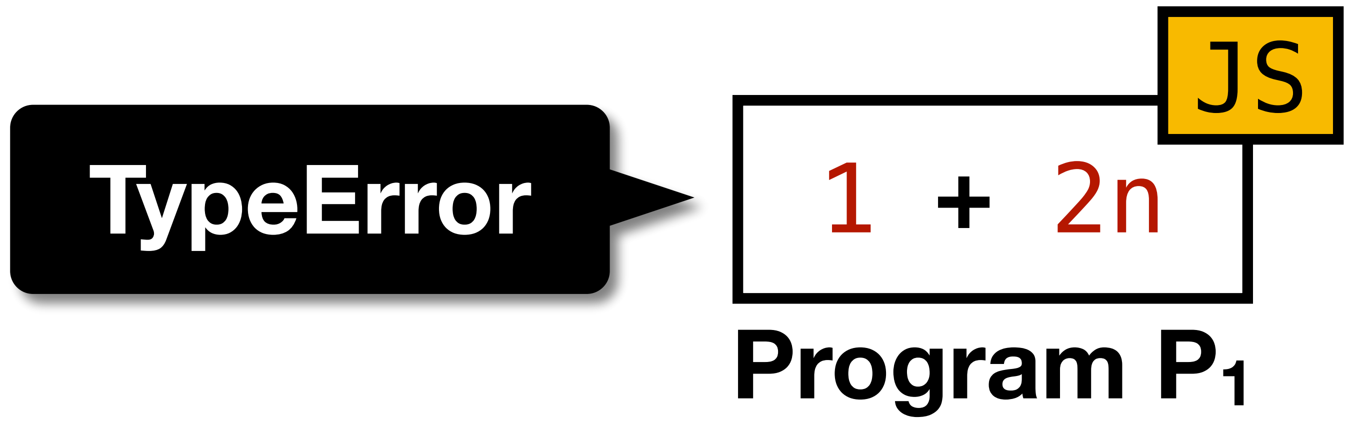
EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

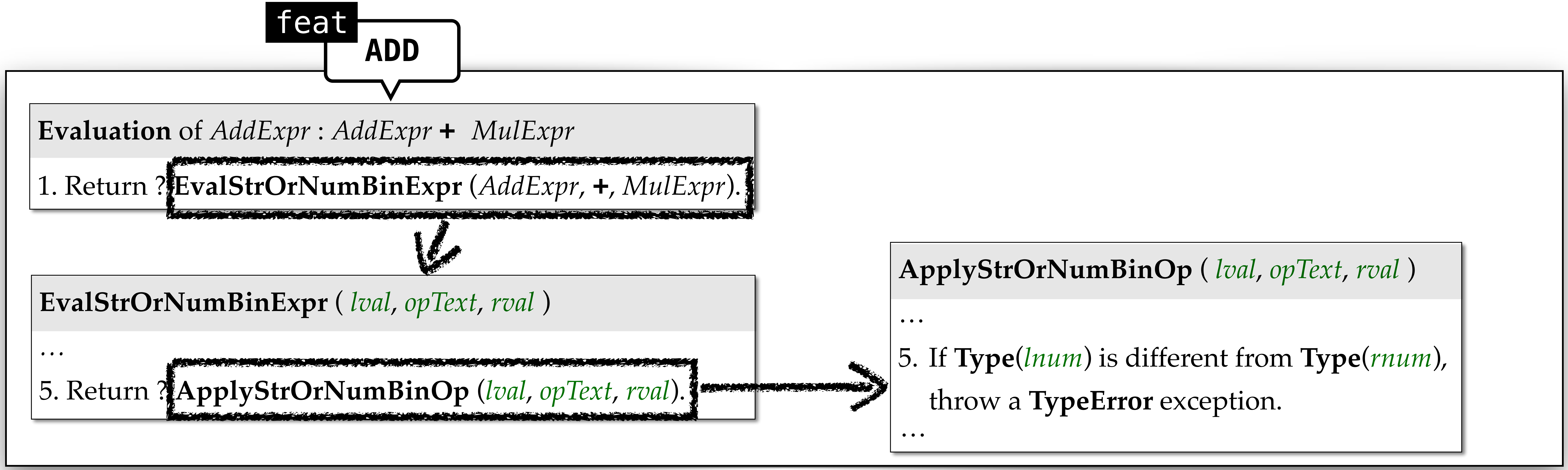
…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**





Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

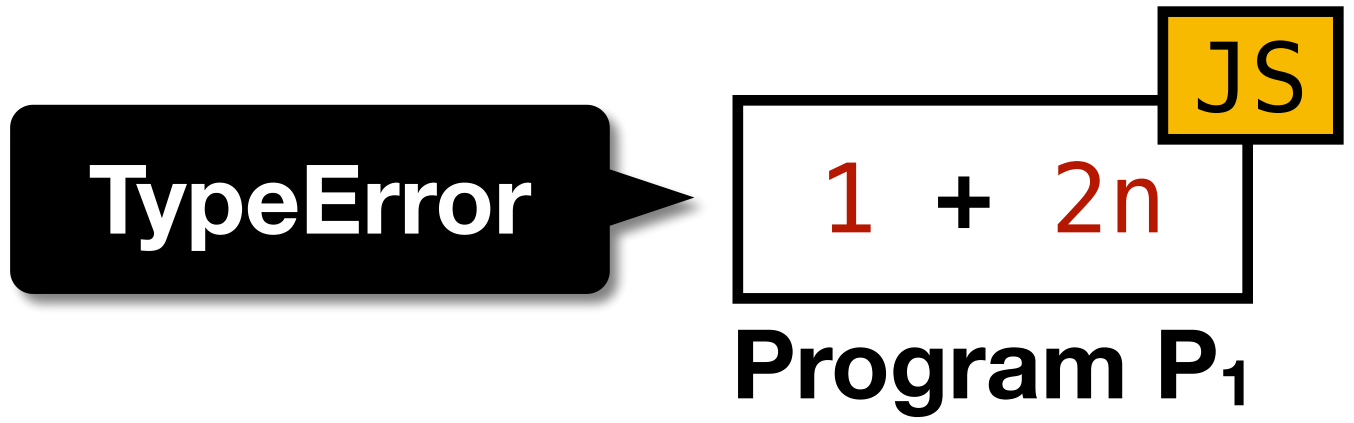
…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**

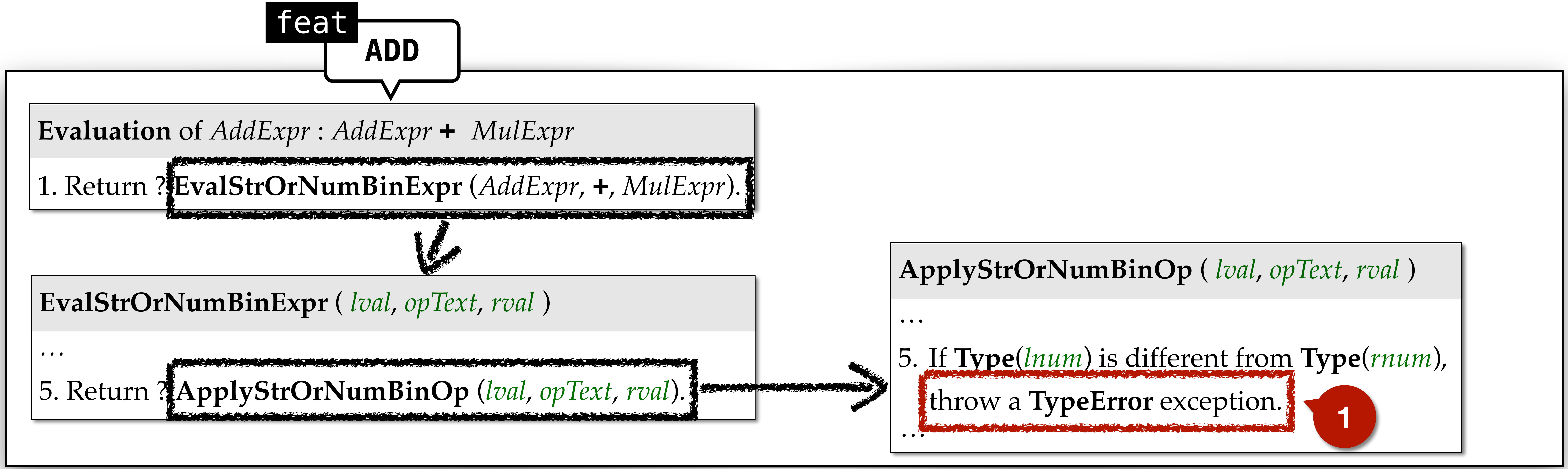


TypeError

JS

```
1 + 2n
```

Program P_1



feat

ADD

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**

Node Coverage

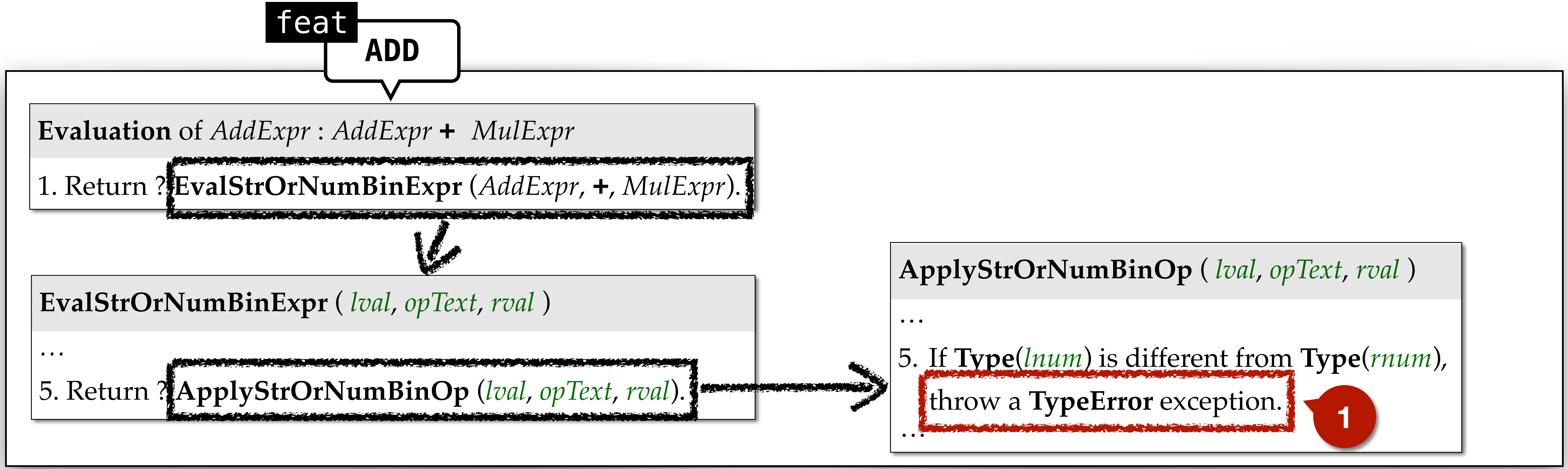
TR = Node

TypeError

JS

```
1 + 2n
```

Program P₁



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

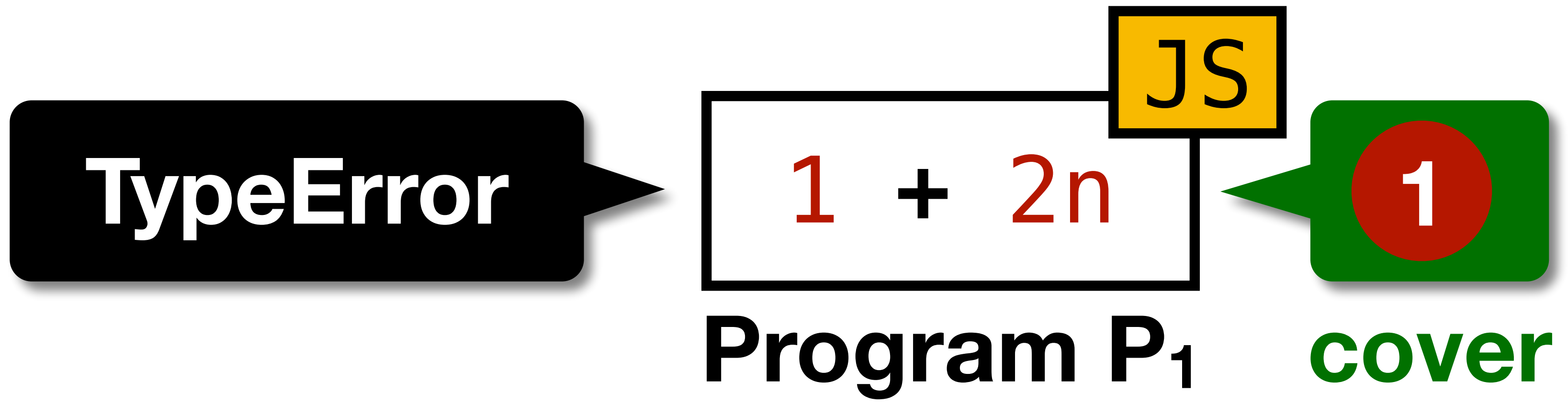
…

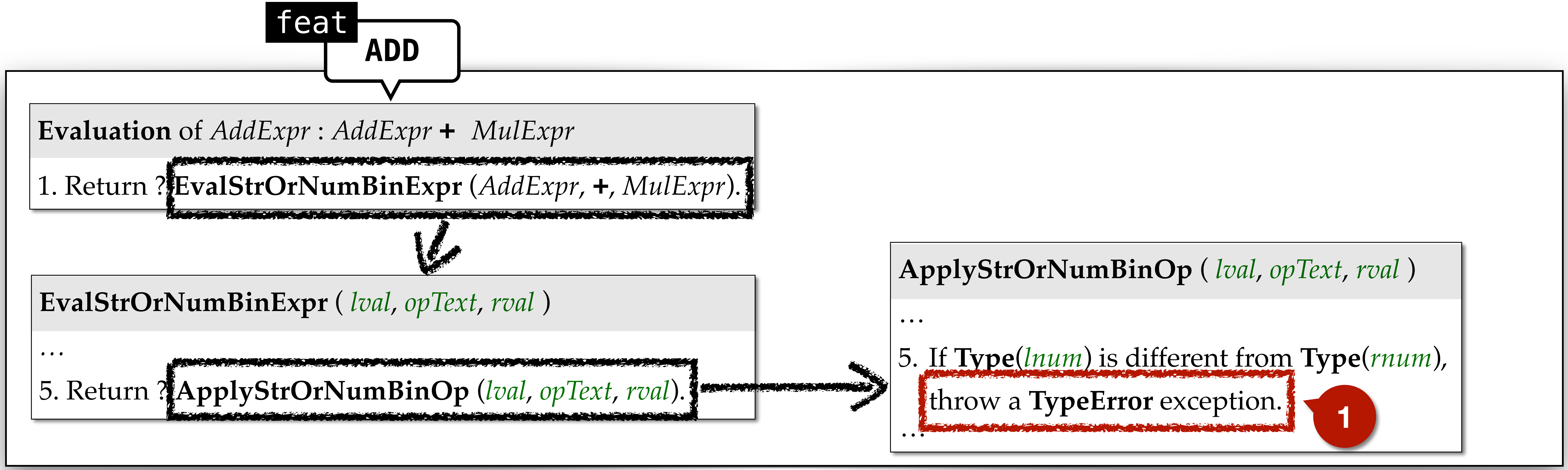
Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**

Node Coverage

TR = Node





Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

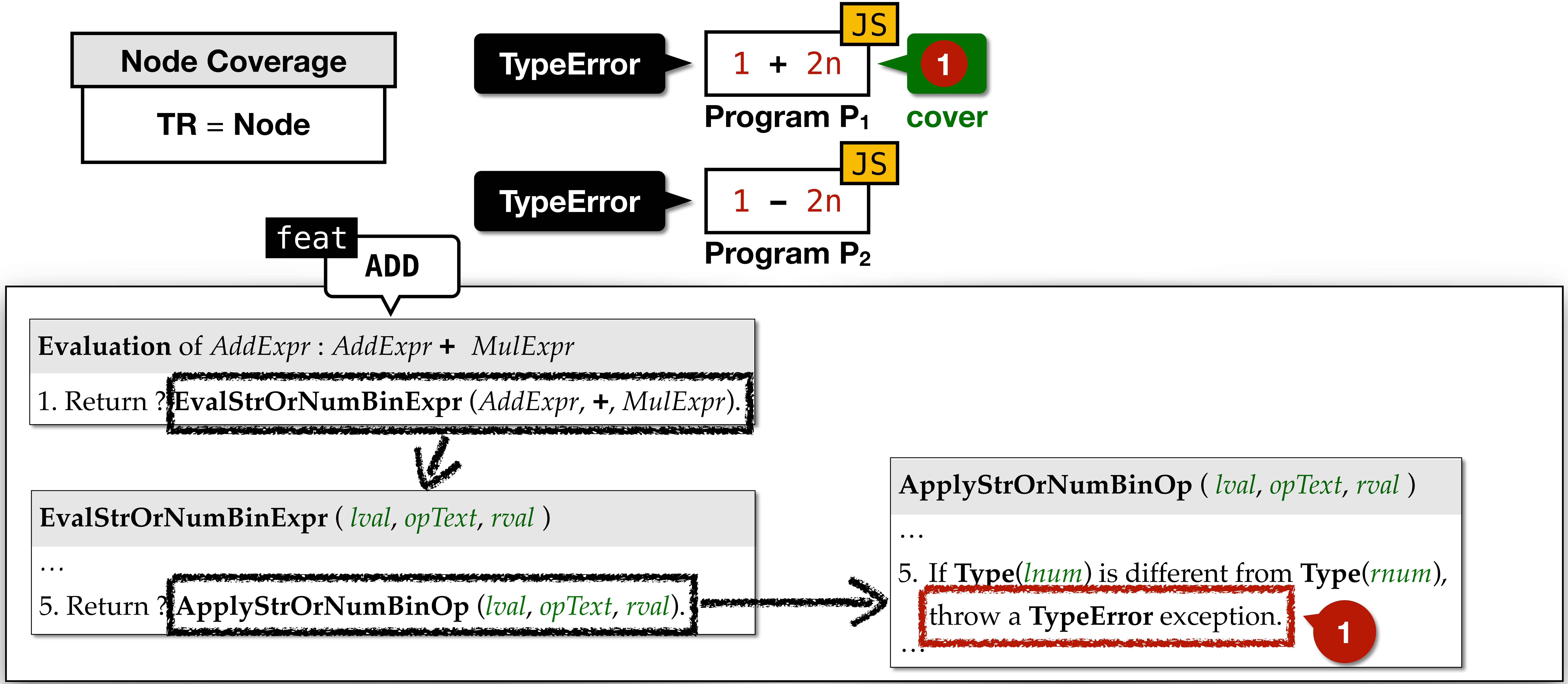
…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**



Node Coverage

TR = Node

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

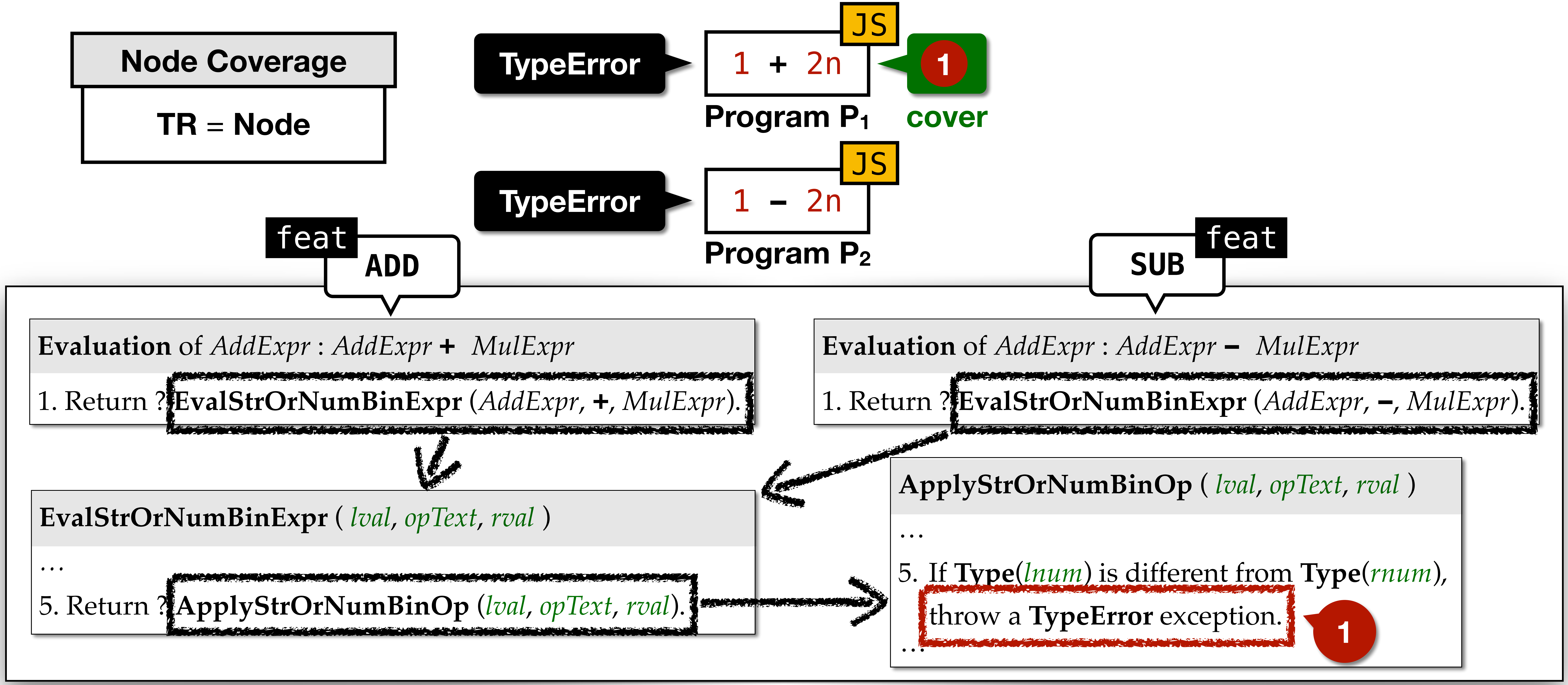
…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

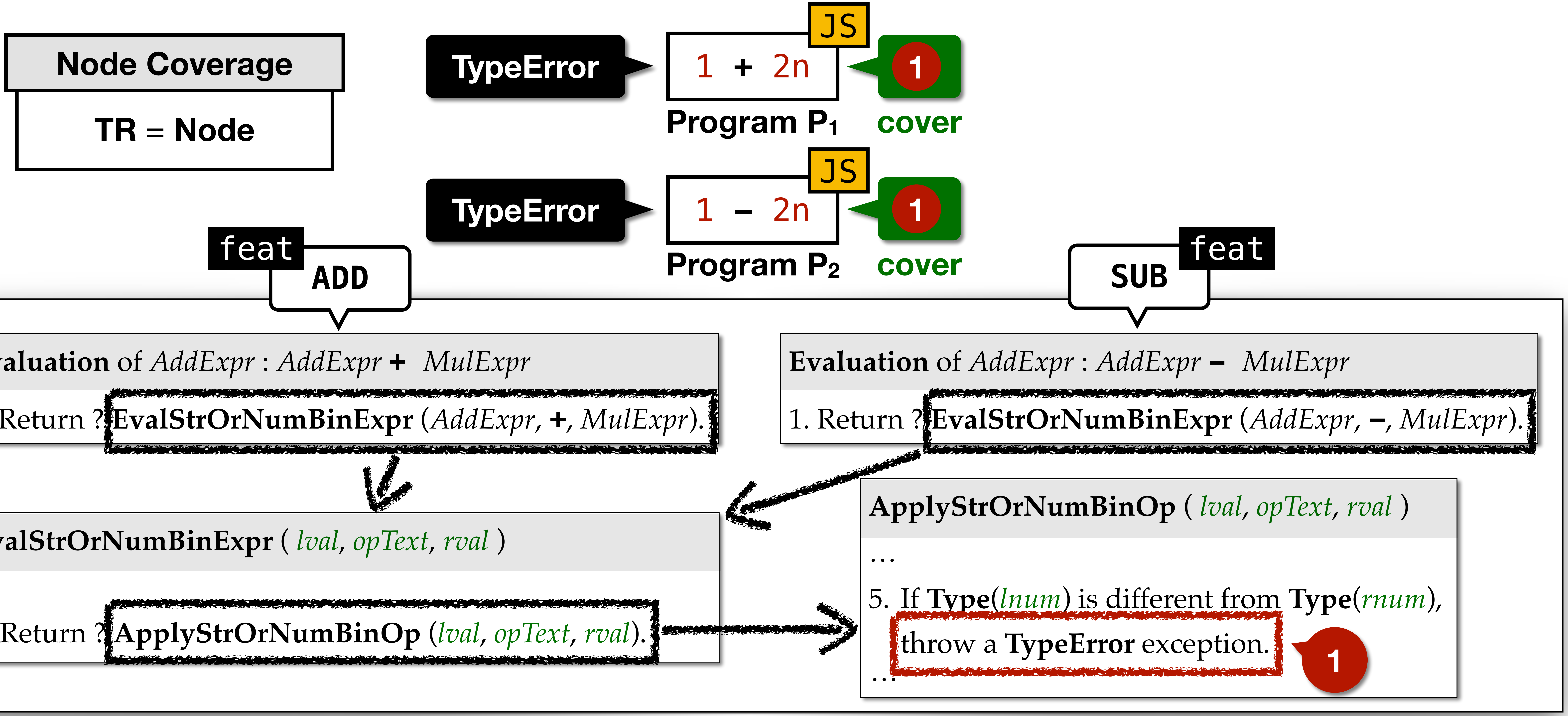
…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**



Node Coverage

TR = Node

TypeError

```
1 + 2n
```

JS

Program P_1

1

cover

TypeError

```
1 - 2n
```

JS

Program P_2

1

cover

feat ADD

feat SUB

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

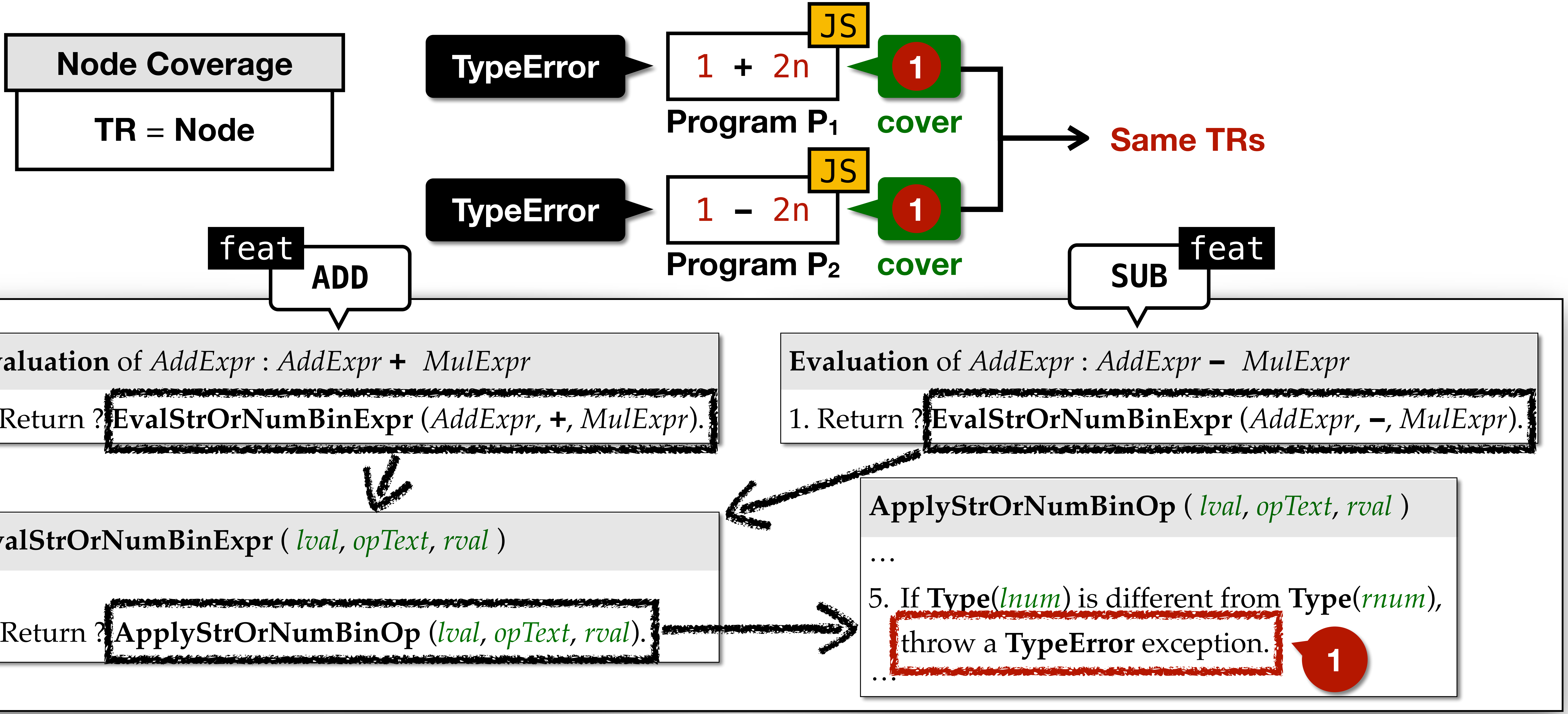
5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

1

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

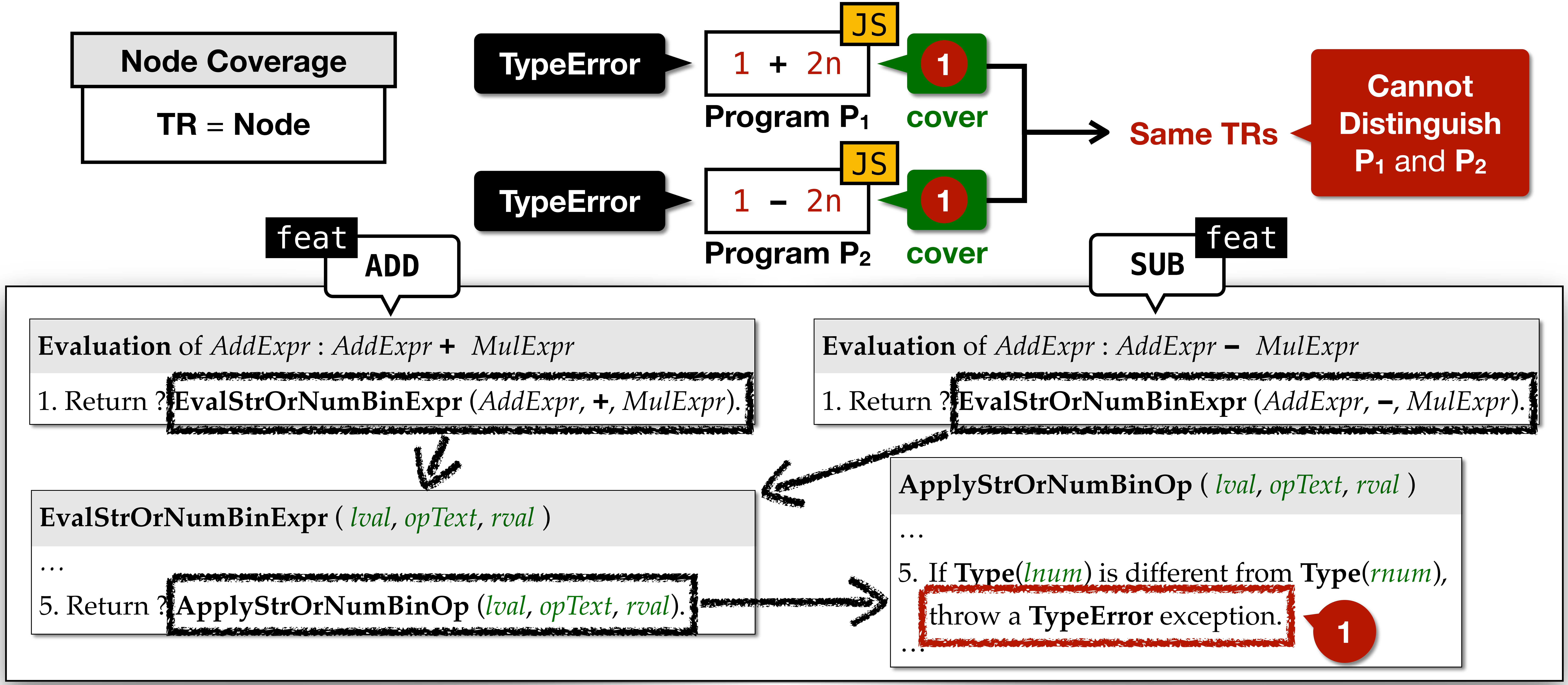
5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 1 with **Node Coverage**

Node Coverage
TR = Node



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

…

5. Return ? **ApplyStrOrNumBinOp** (*lval*, *opText*, *rval*).

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

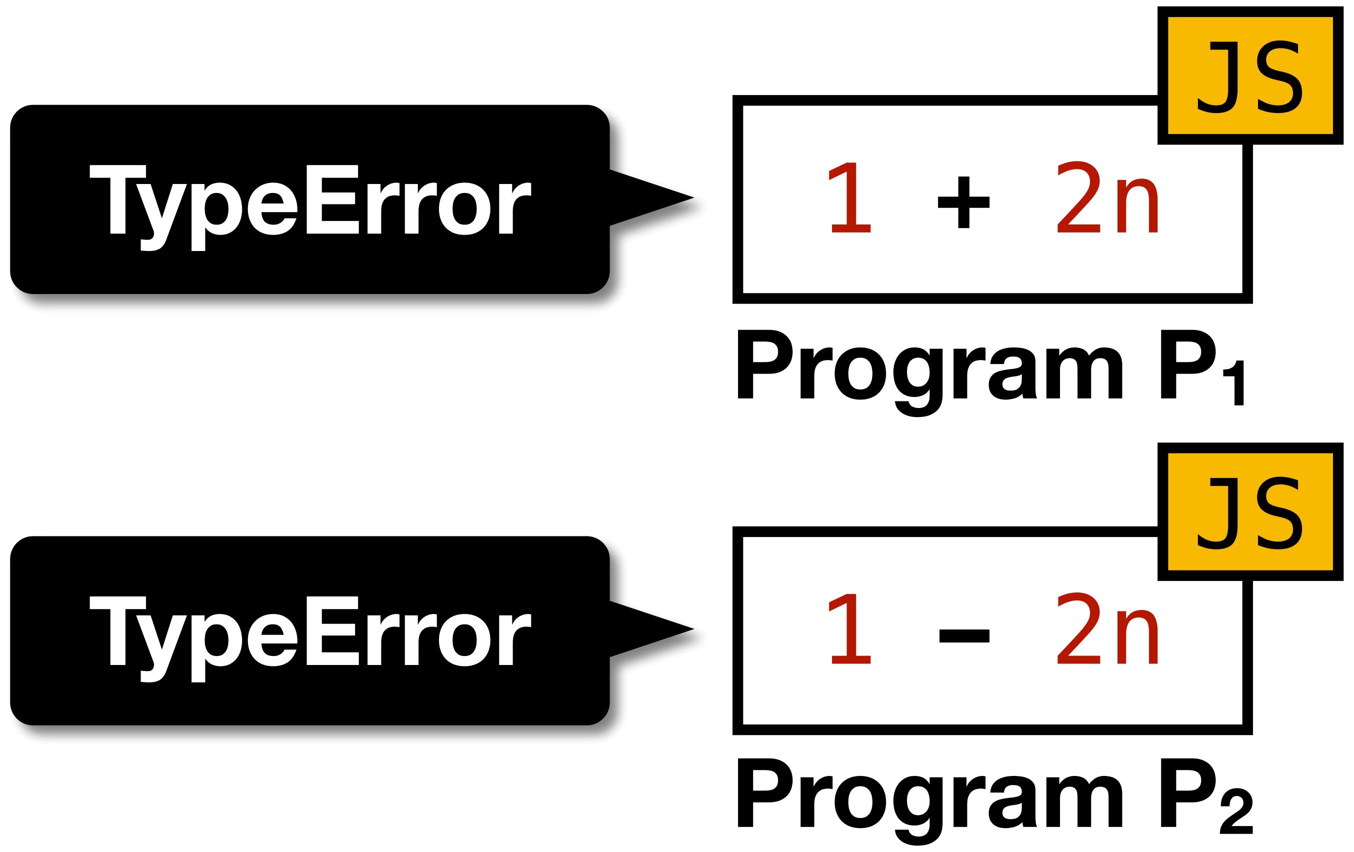
…

5. If **Type**(*lnum*) is different from **Type**(*rnum*), throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Feature-Sensitive (FS) Coverage



TypeError

```
1 + 2n
```

JS

Program P_1

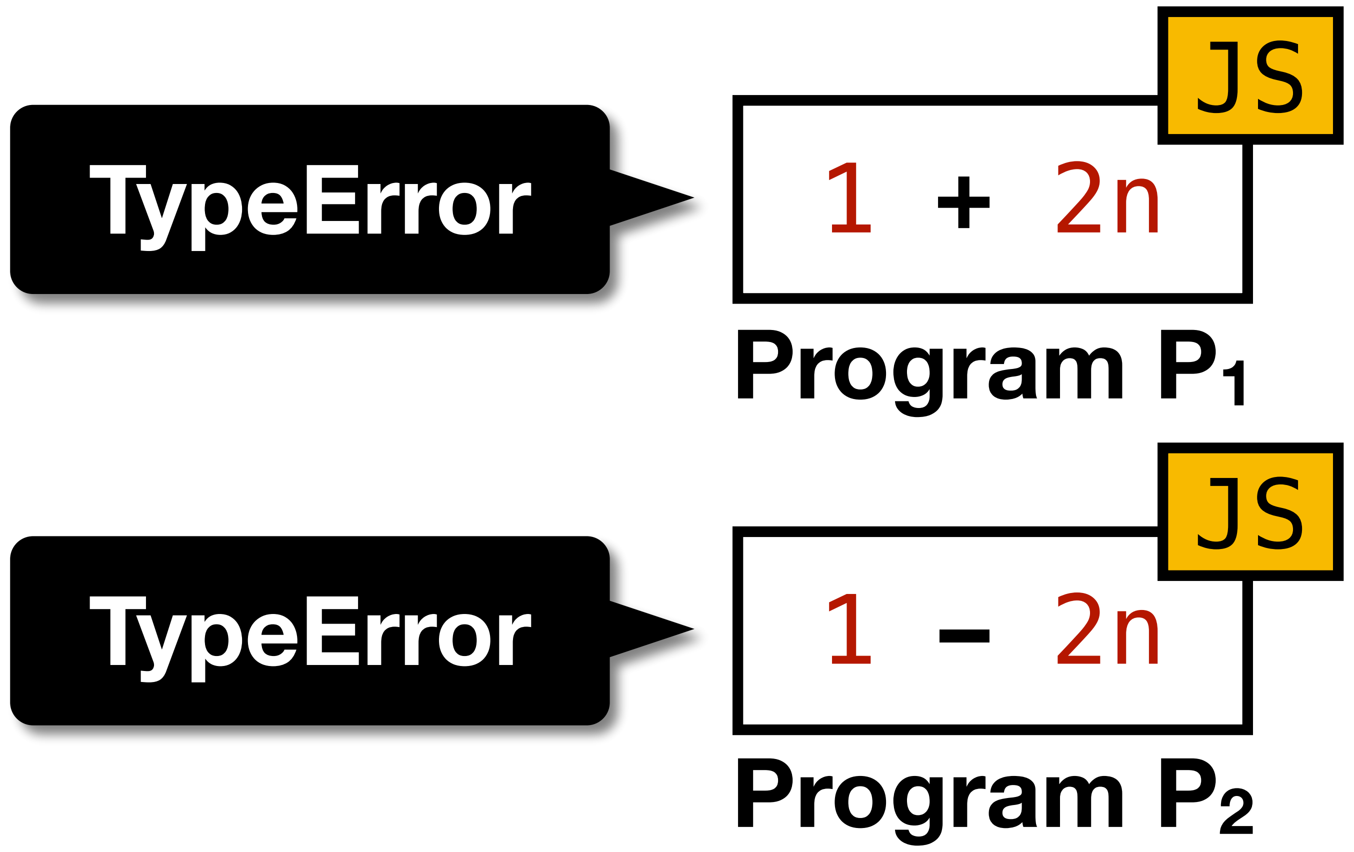
TypeError

```
1 - 2n
```

JS

Program P_2

Feature-Sensitive (FS) Coverage



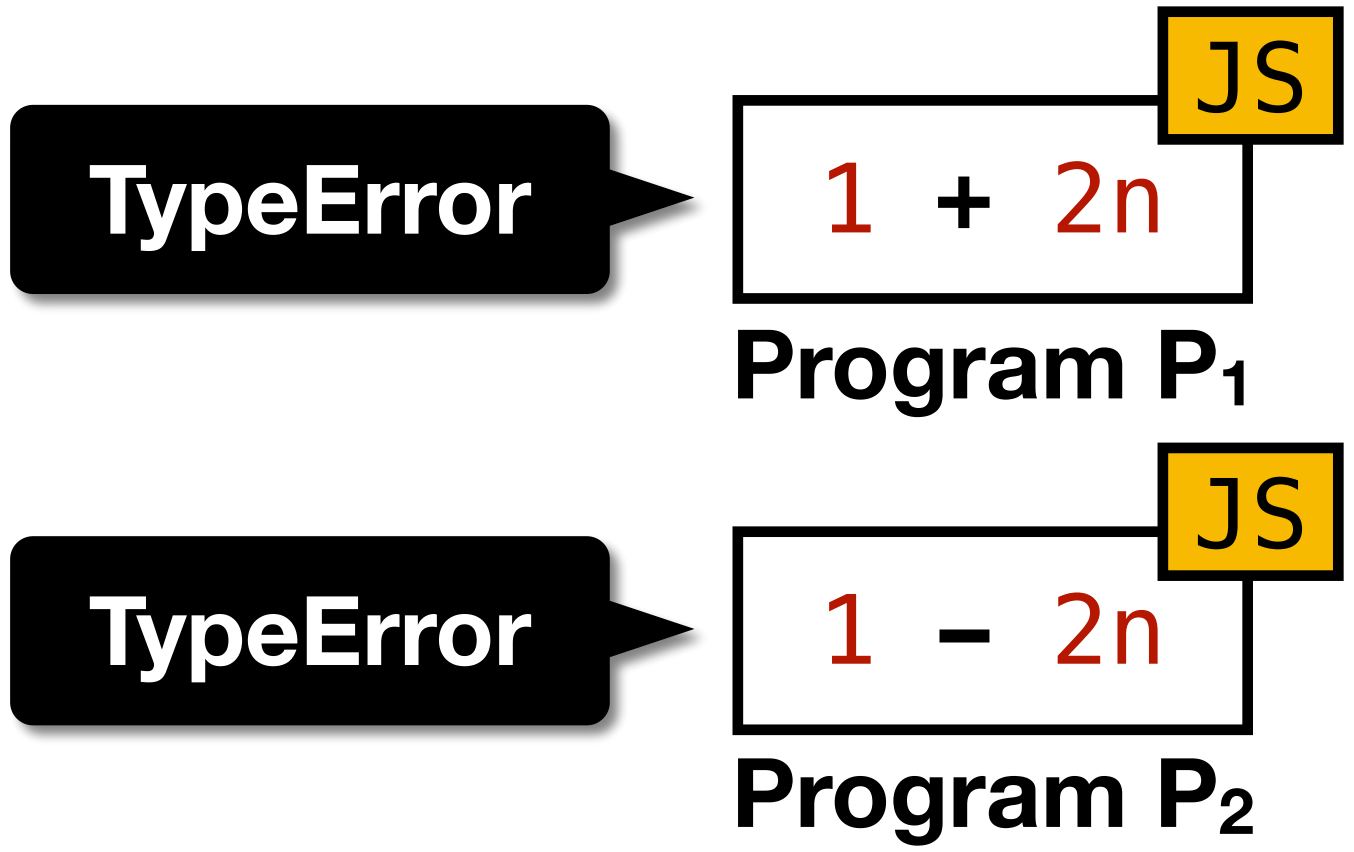
```
1 + 2n
```

```
1 - 2n
```

- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage
TR = (Feature, given **TR)**

Feature-Sensitive (FS) Coverage



TypeError

JS

`1 + 2n`

Program P_1

TypeError

JS

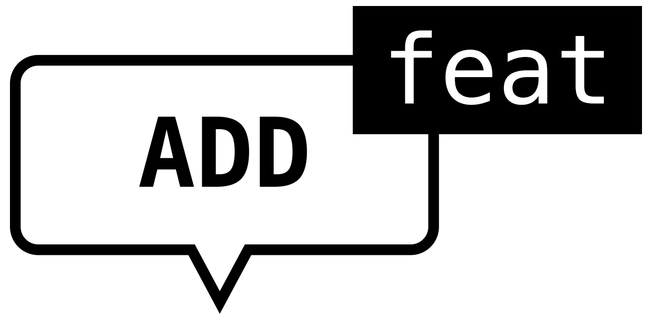
`1 - 2n`

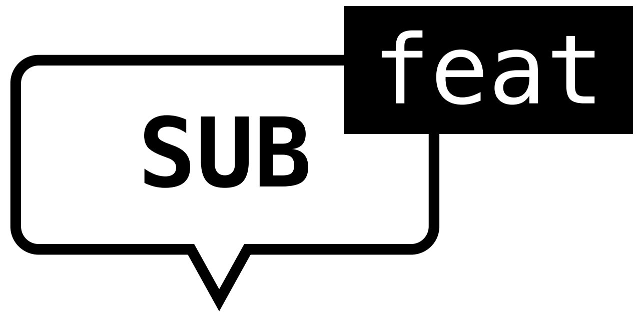
Program P_2

- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given **TR)**





Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

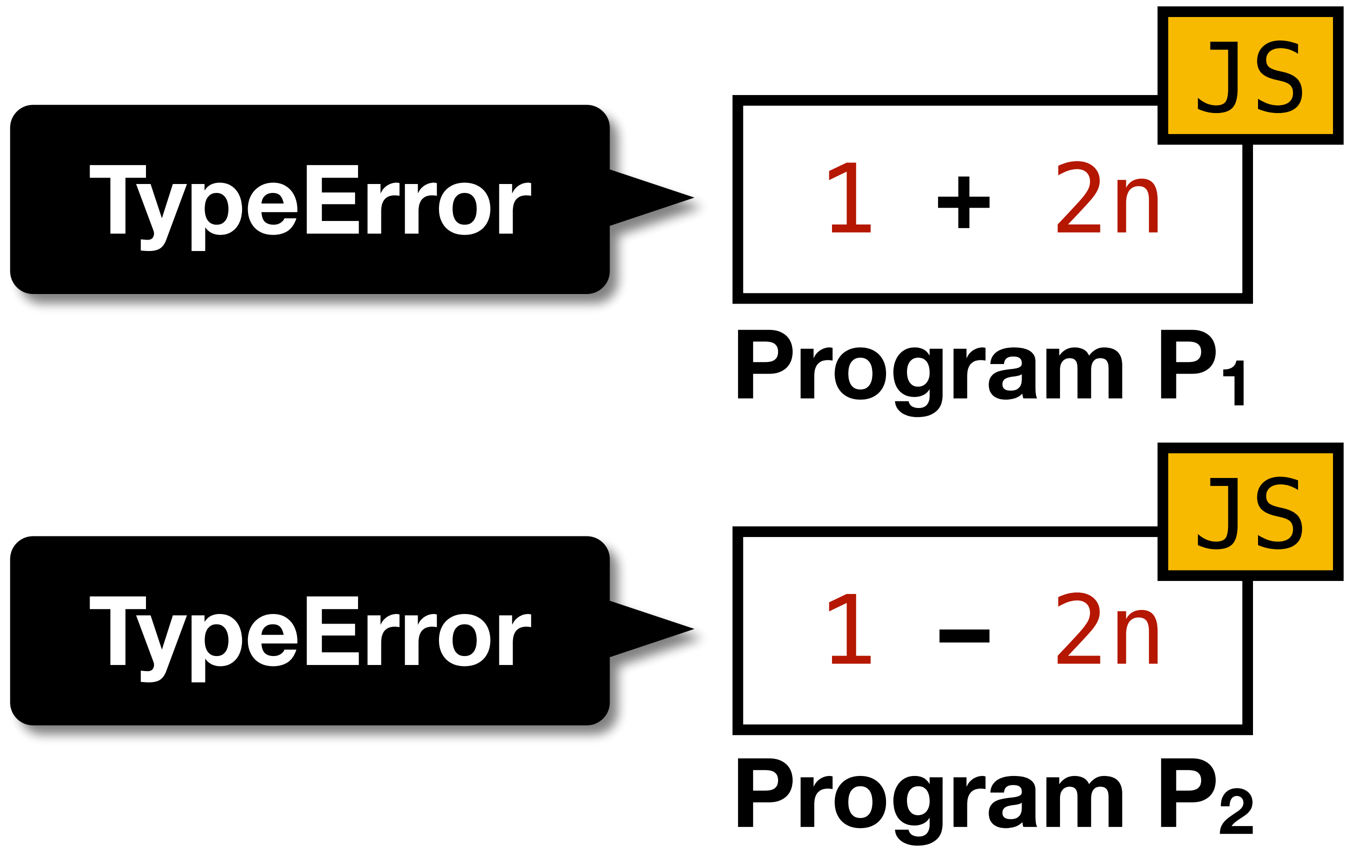
Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Feature-Sensitive (FS) Coverage

FS Node Coverage

TR = **(Feature, Node)**



TypeError

JS

`1 + 2n`

Program P_1

TypeError

JS

`1 - 2n`

Program P_2

- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given **TR)**

feat **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

feat **SUB**

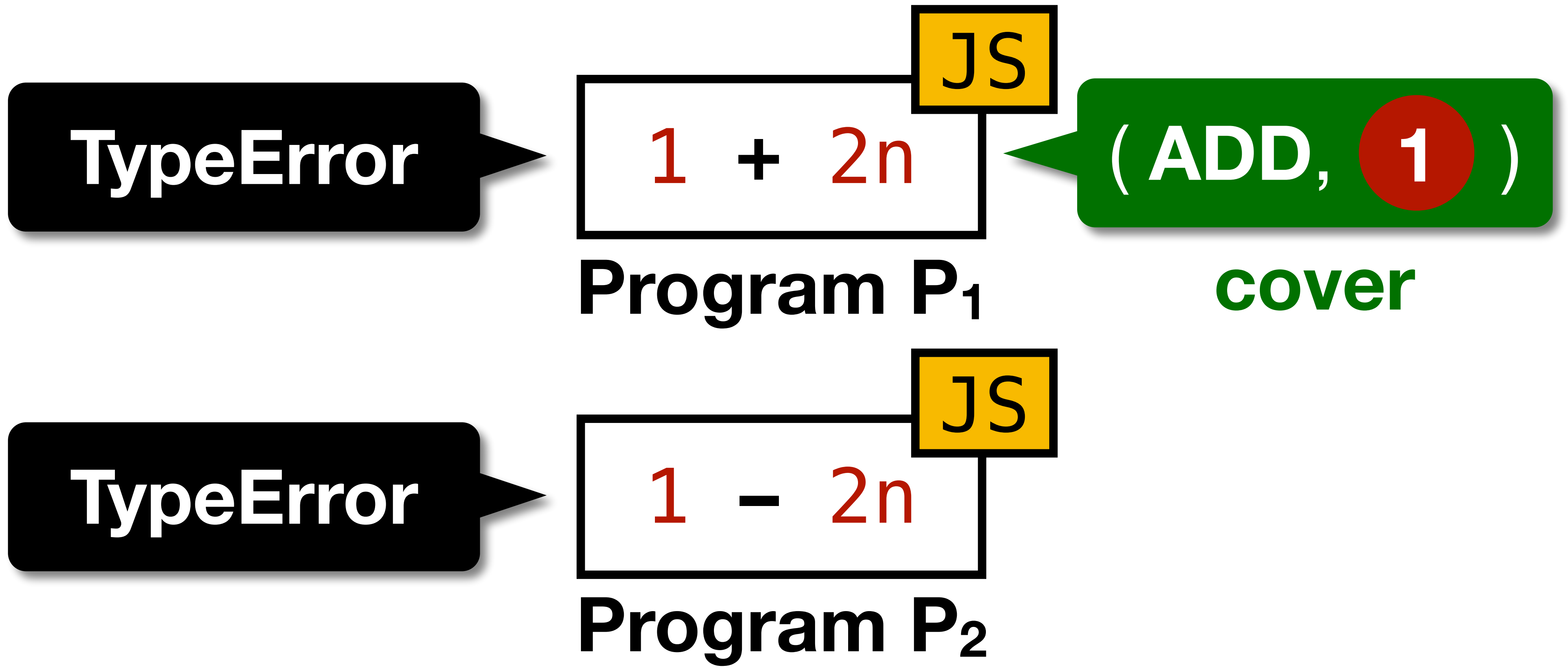
Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Feature-Sensitive (FS) Coverage

FS Node Coverage

TR = (Feature, Node)



- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given TR)

ADD feat

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

SUB feat

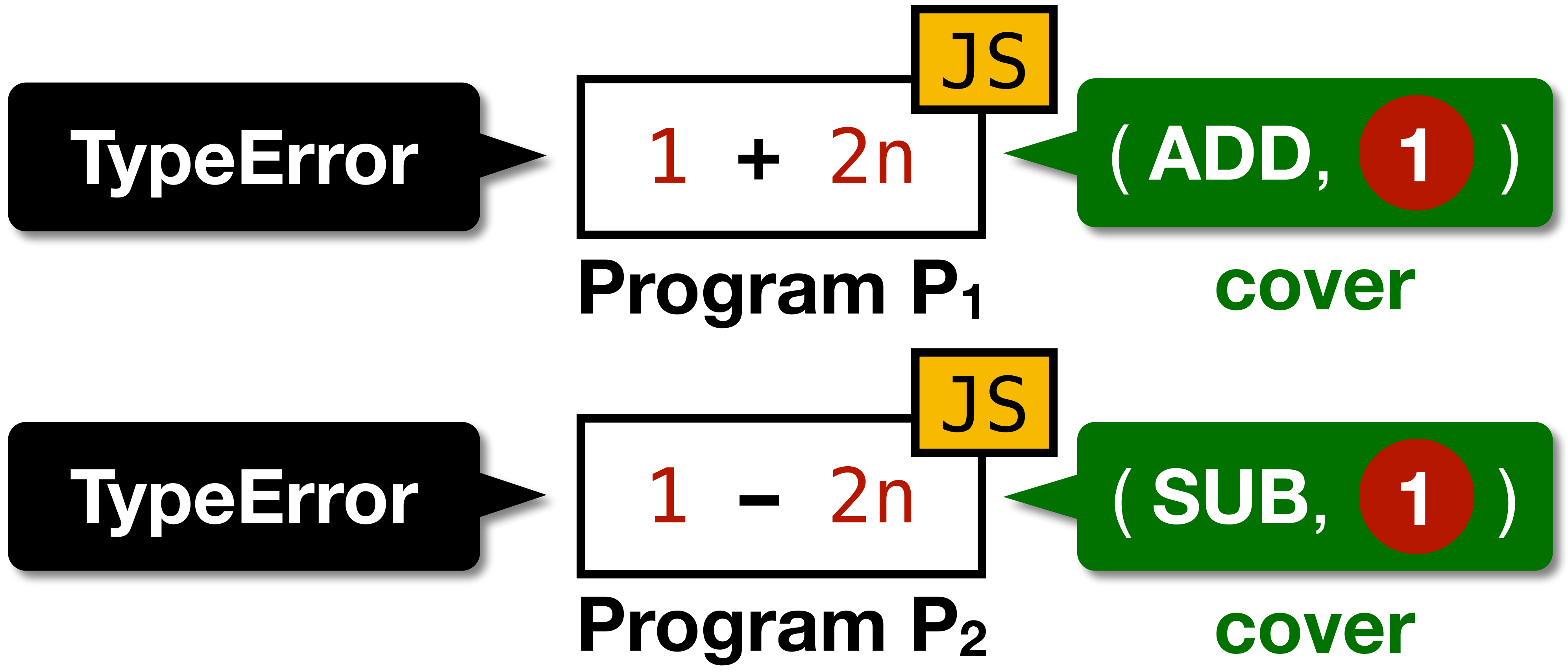
Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Feature-Sensitive (FS) Coverage

FS Node Coverage

TR = (Feature, Node)


TypeError

JS

`1 + 2n`

Program P₁

(ADD, 1)

cover

TypeError

JS

`1 - 2n`

Program P₂

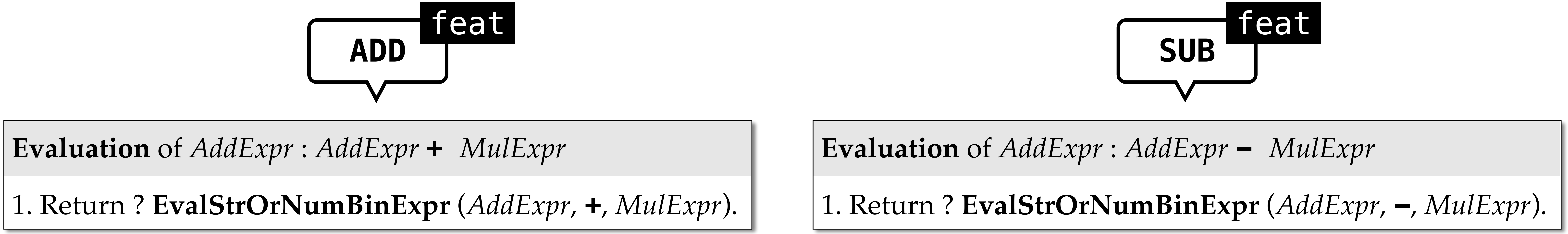
(SUB, 1)

cover

- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given TR)


`feat` **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

`feat` **SUB**

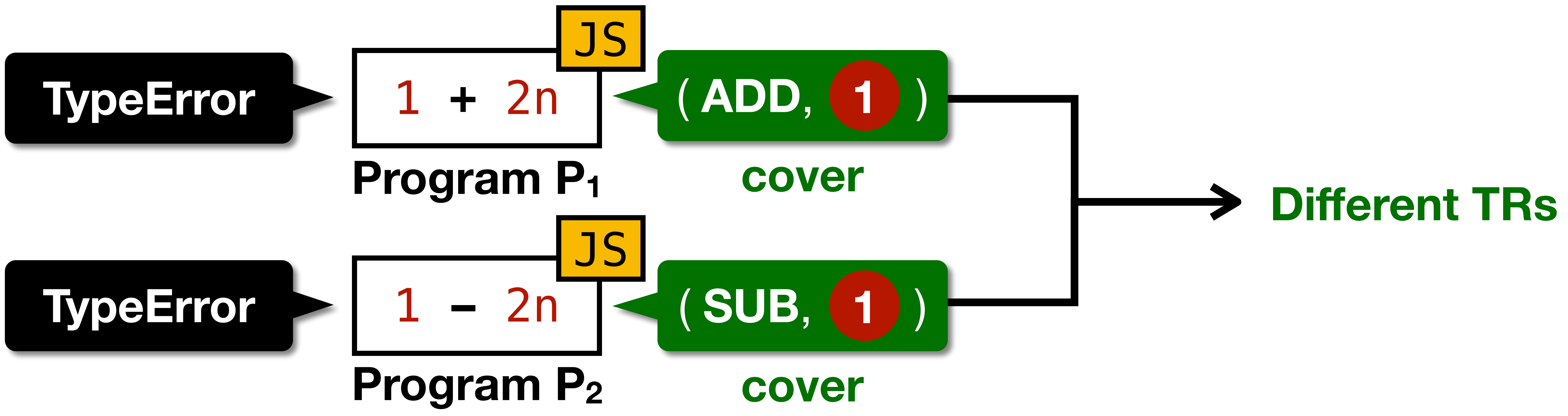
Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Feature-Sensitive (FS) Coverage

FS Node Coverage

TR = (Feature, Node)



- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given TR)

feat **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

feat **SUB**

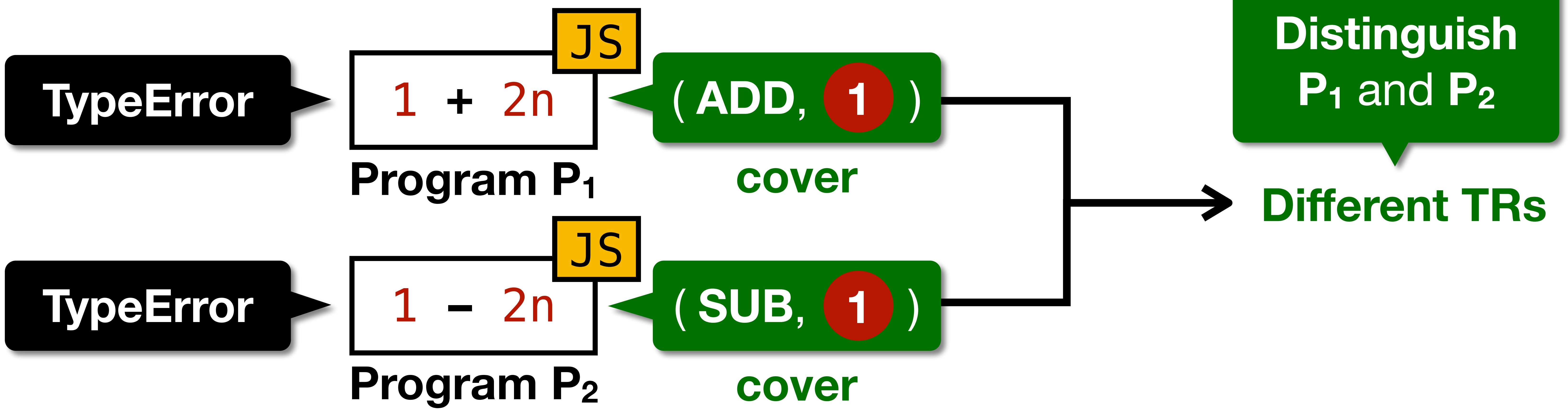
Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Feature-Sensitive (FS) Coverage

FS Node Coverage

TR = (Feature, Node)



- **Feature-Sensitive (FS)** coverage criterion **divides** the given TRs with the **innermost enclosing** language **features**

FS Coverage

TR = (Feature, given TR)

feat
ADD

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

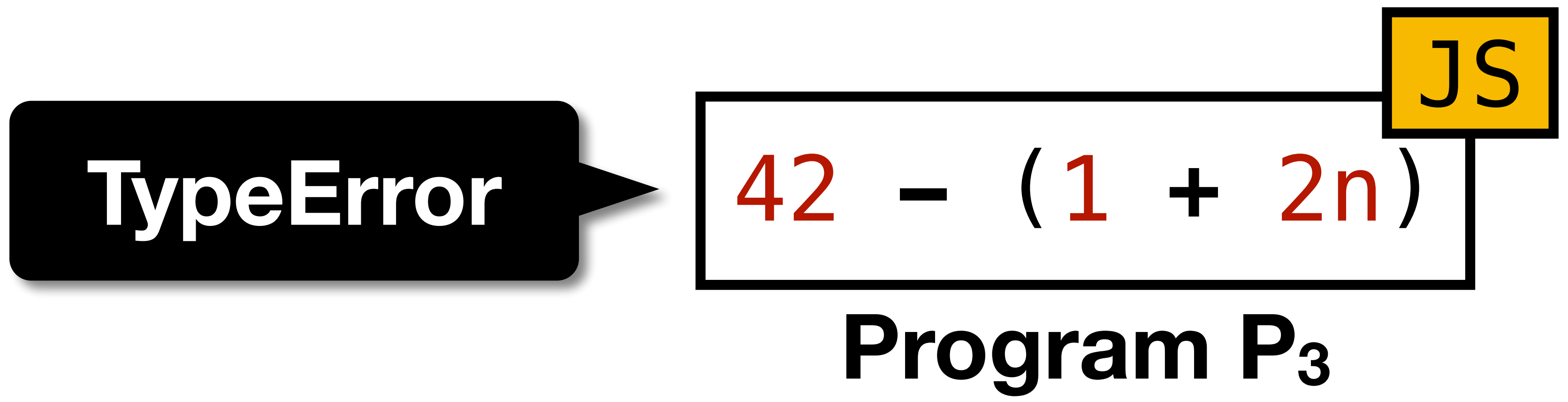
1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

feat
SUB

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

k-Feature-Sensitive (k-FS) Coverage



```
42 - (1 + 2n)
```

- k-**F**eature-**S**ensitive (**k-FS**) coverage criterion **divides** the given TRs with at most k**-innermost enclosing** language **features**

k-FS Coverage

TR = (Feature$^{\leq k}$, given TR)

feat **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

feat **SUB**

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

k-Feature-Sensitive (k-FS) Coverage

2-FS Node Coverage

TR = **(Feature$^{\le 2}$, Node)**

TypeError

JS

```
42 - (1 + 2n)
```

Program P$_3$

- k-**F**eature-**S**ensitive (**k-FS**) coverage criterion **divides** the given TRs with at most **k-innermost enclosing** language **features**

k-FS Coverage

TR = (Feature$^{\le k}$, given **TR)**

feat **ADD**

feat **SUB**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

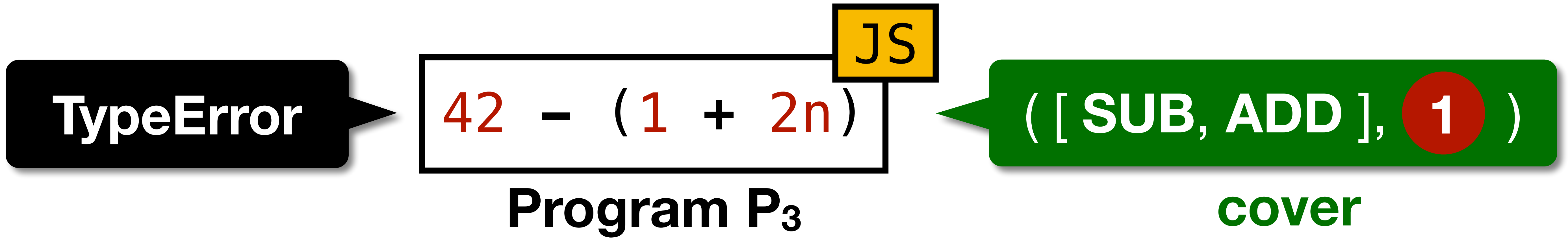
Evaluation of *AddExpr* : *AddExpr* **−** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **−**, *MulExpr*).

k-Feature-Sensitive (k-FS) Coverage

2-FS Node Coverage

TR = **(Feature$^{\le 2}$, Node)**



- ***k*-Feature-Sensitive (*k*-FS)** coverage criterion **divides** the given TRs with at most ***k*-innermost enclosing** language **features**

***k*-FS Coverage**

TR = (Feature$^{\le k}$, given **TR)**

ADD `feat`

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **+**, *MulExpr*).

SUB `feat`

Evaluation of *AddExpr* : *AddExpr* **–** *MulExpr*

1. Return ? **EvalStrOrNumBinExpr** (*AddExpr*, **–**, *MulExpr*).

Motivating Example 2

TypeError

JS

```
Symbol() + 1
```

Program P_4

Motivating Example 2

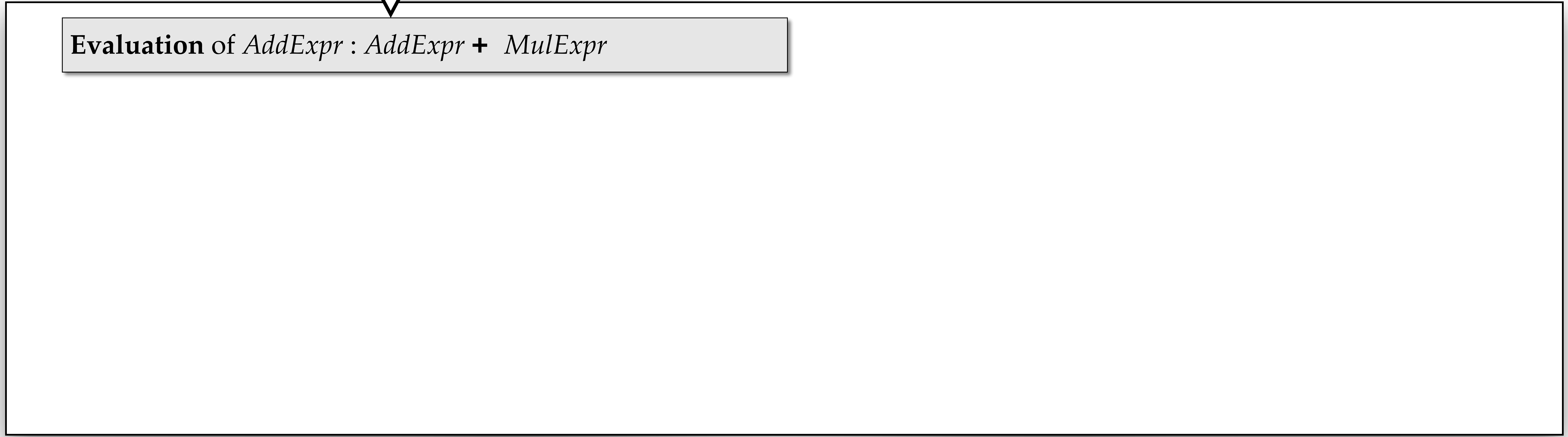
TypeError

JS

```
Symbol() + 1
```

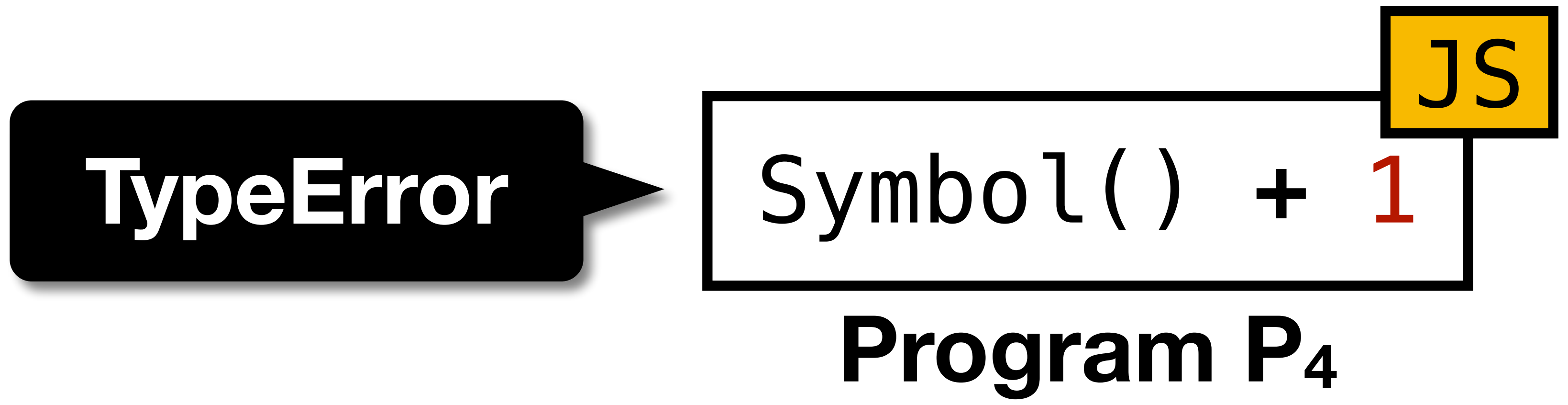
Program P_4

feat **ADD**



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

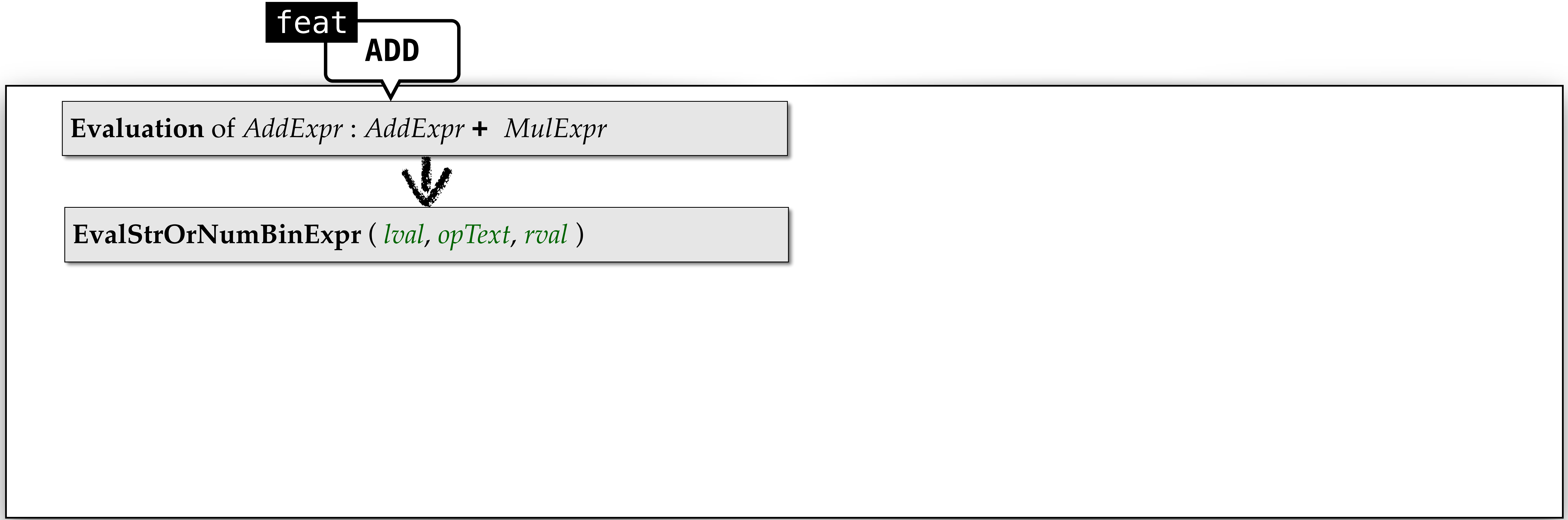
Motivating Example 2



TypeError

```
Symbol() + 1
```

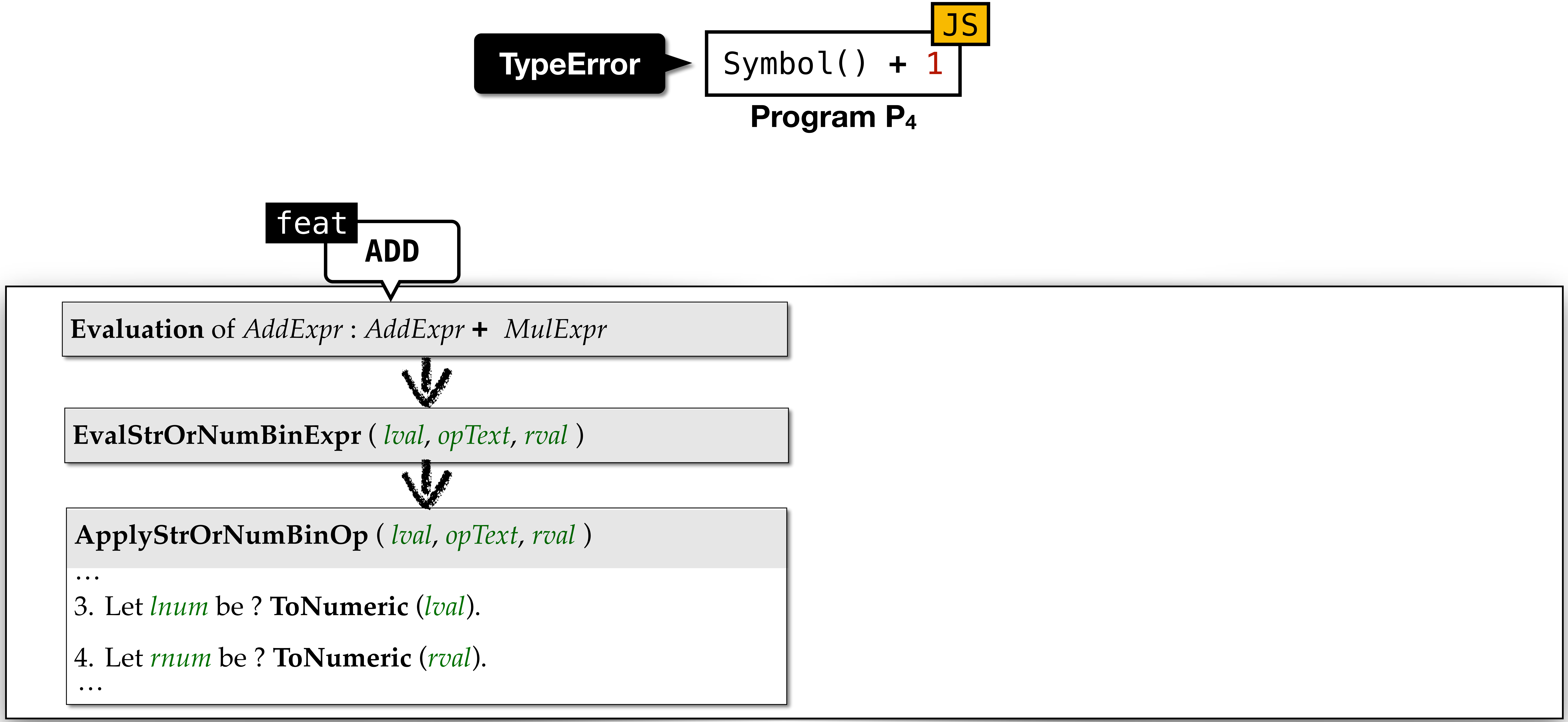
Program P_4



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

Motivating Example 2



```
Symbol() + 1
```

Program P_4

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

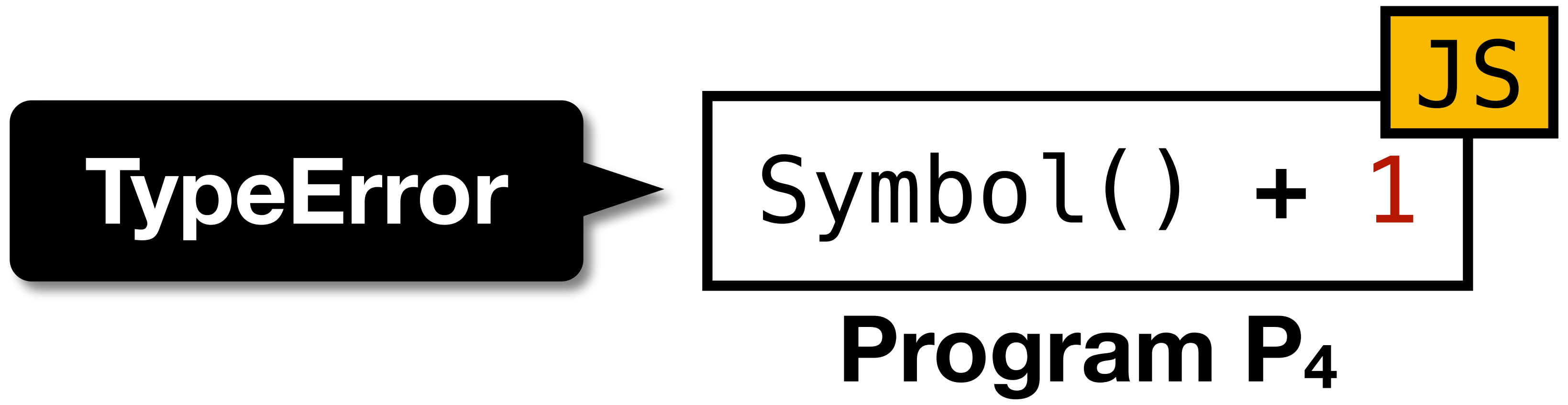
…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

Motivating Example 2

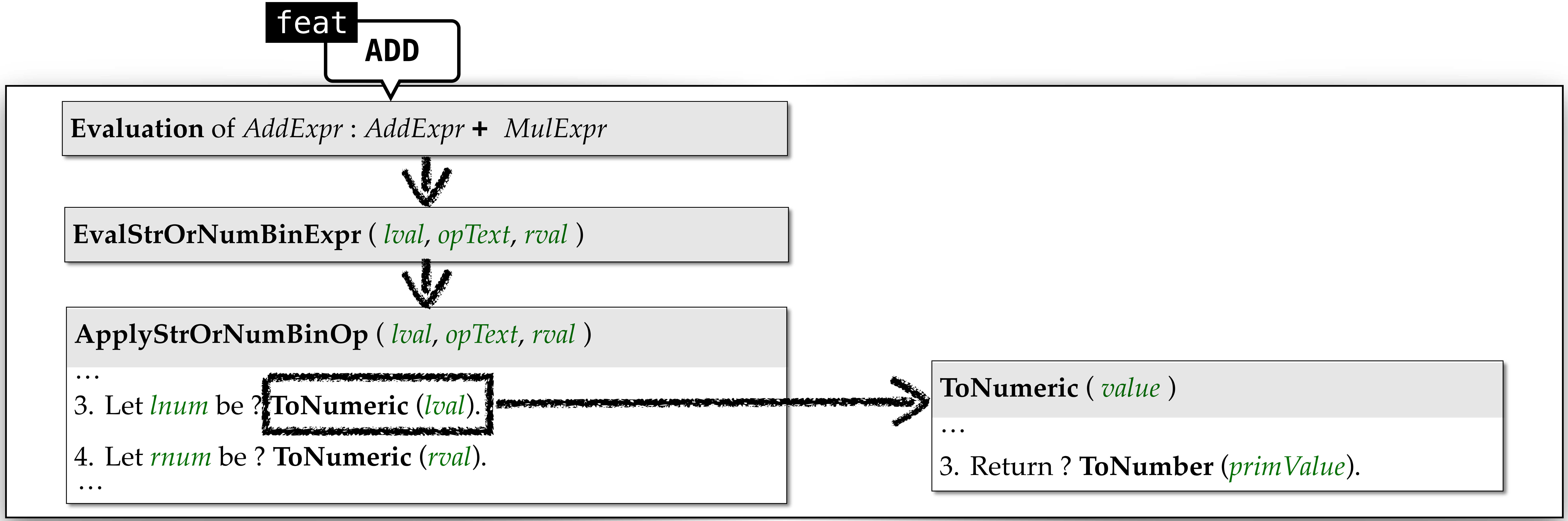


TypeError

JS

```
Symbol() + 1
```

Program P_4



feat

ADD

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

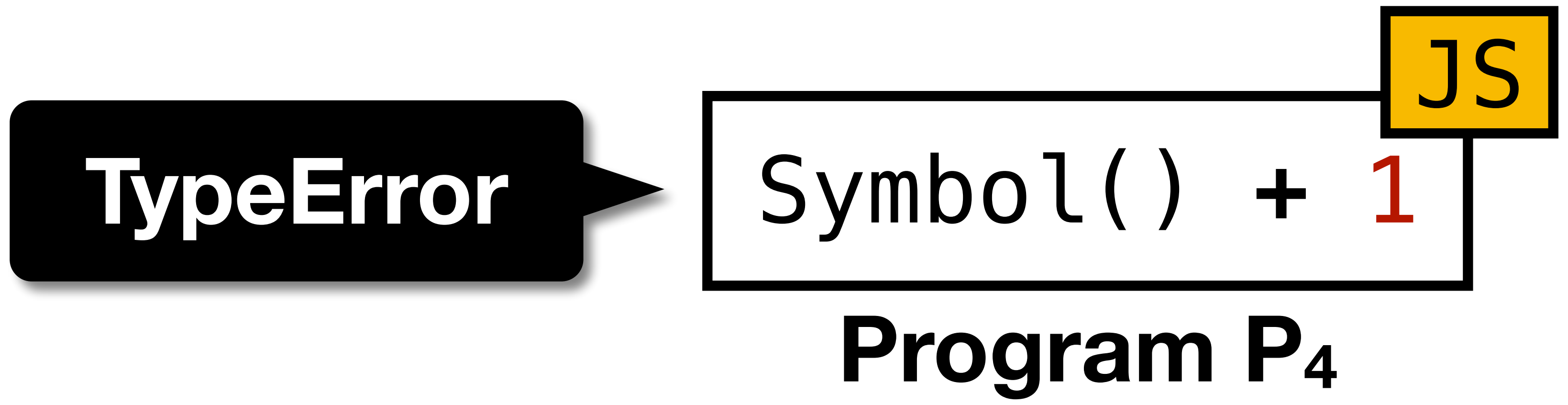
ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

```
…
3. Let lnum be ? ToNumeric (lval).
4. Let rnum be ? ToNumeric (rval).
…
```

ToNumeric (*value*)

```
…
3. Return ? ToNumber (primValue).
```

Motivating Example 2

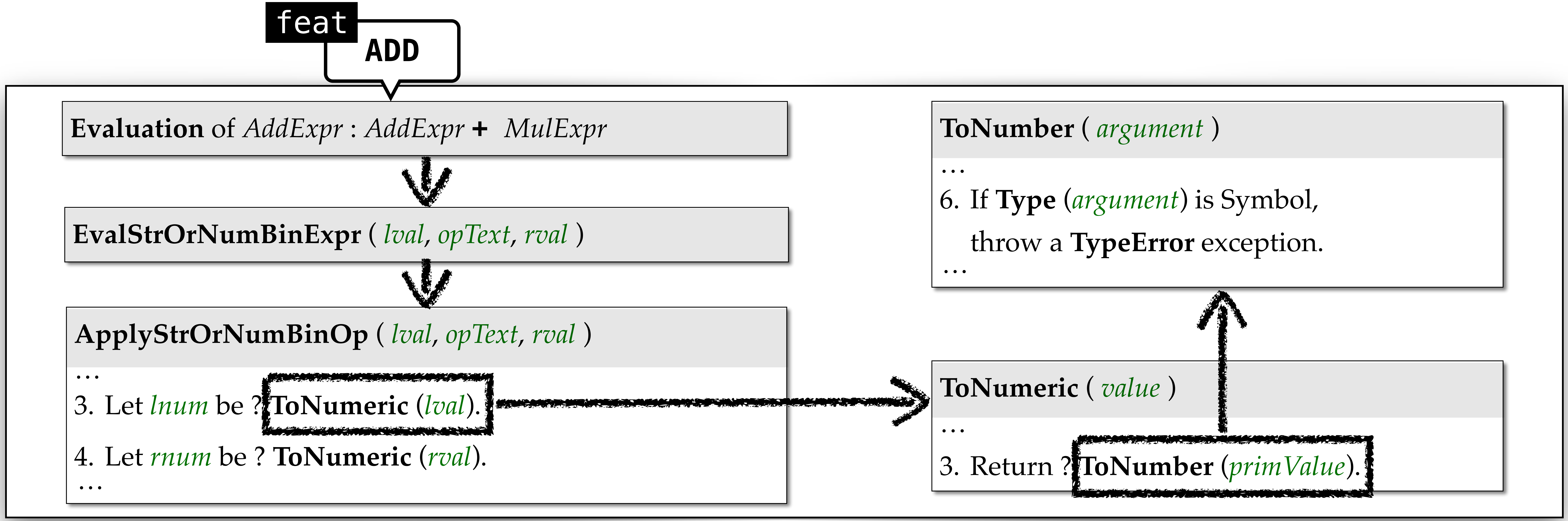


TypeError

JS

```
Symbol() + 1
```

Program P_4



feat

ADD

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

```
…
3. Let lnum be ? ToNumeric (lval).
4. Let rnum be ? ToNumeric (rval).
…
```

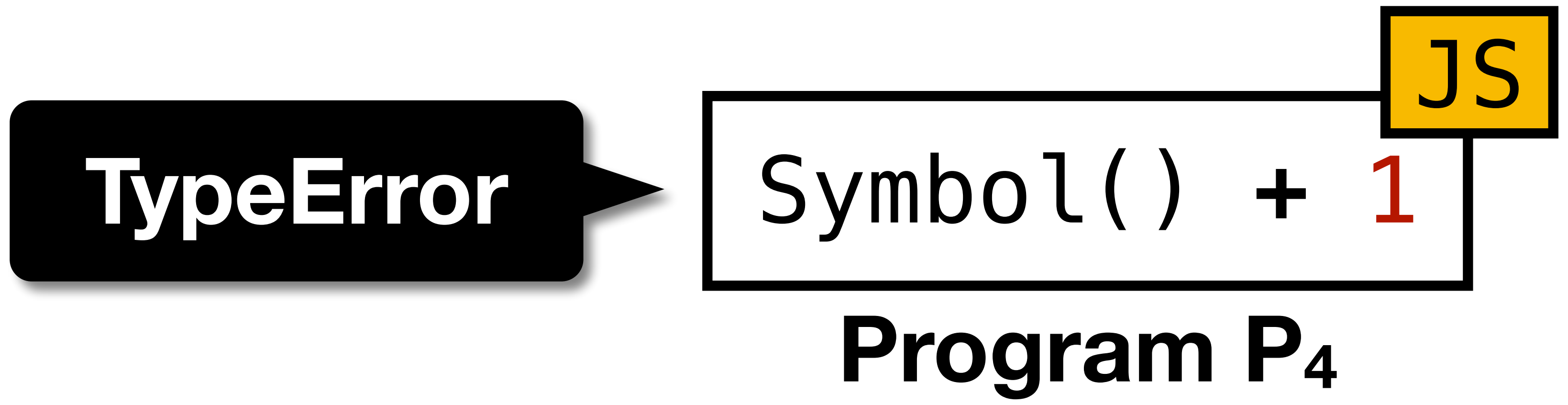
ToNumeric (*value*)

```
…
3. Return ? ToNumber (primValue).
```

ToNumber (*argument*)

```
…
6. If Type (argument) is Symbol,
   throw a TypeError exception.
…
```

Motivating Example 2

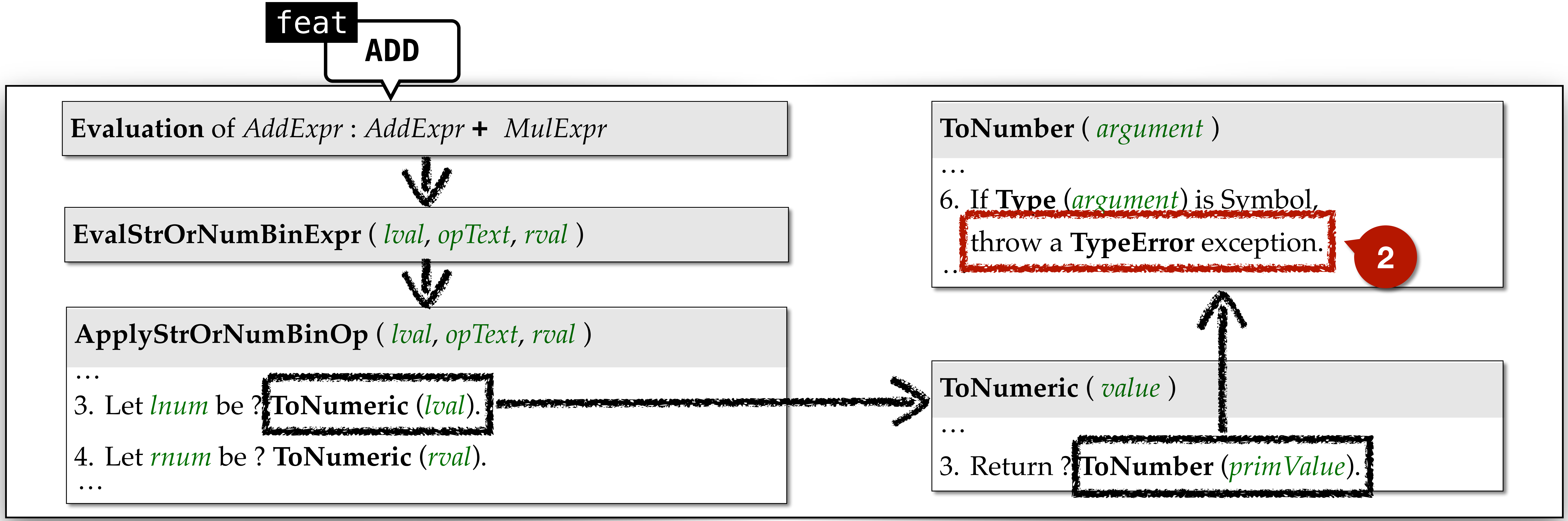


TypeError

JS

```
Symbol() + 1
```

Program P_4



feat ADD

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

```
…
3. Let lnum be ? ToNumeric (lval).
4. Let rnum be ? ToNumeric (rval).
…
```

ToNumeric (*value*)

```
…
3. Return ? ToNumber (primValue).
```

ToNumber (*argument*)

```
…
6. If Type (argument) is Symbol,
   throw a TypeError exception.
…
```

Motivating Example 2

1-FS Node Coverage

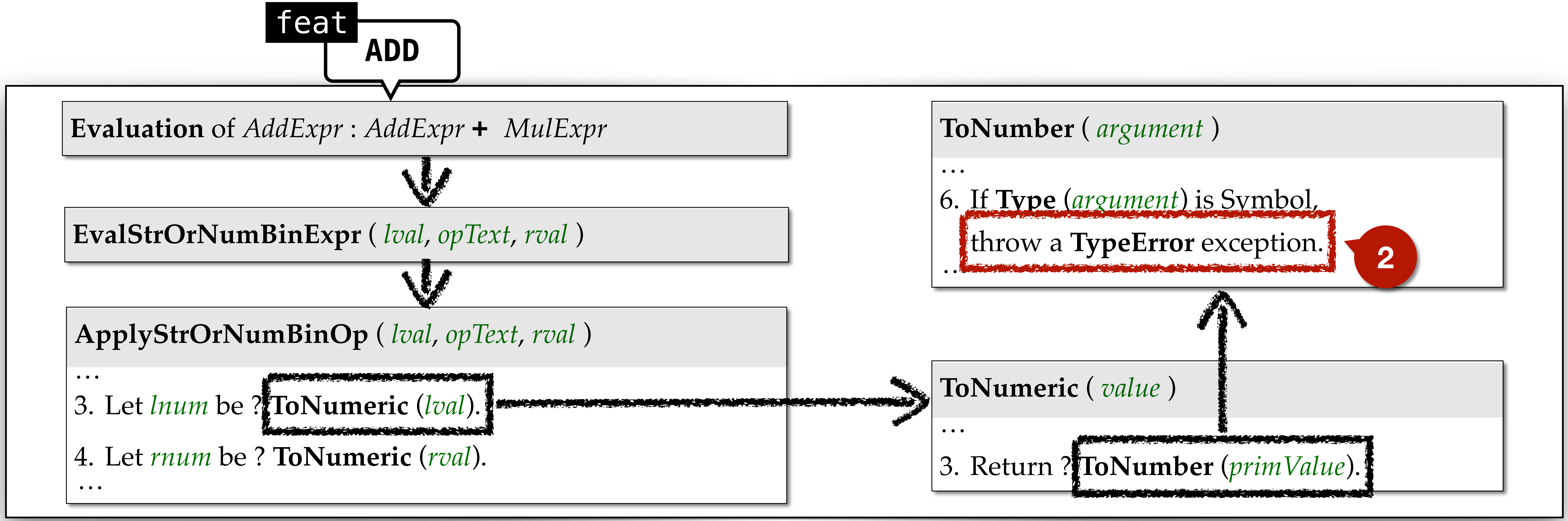
TR = (Feature, Node)

TypeError

```
Symbol() + 1
```

JS

Program P$_4$



feat **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

ToNumber (*argument*)

…

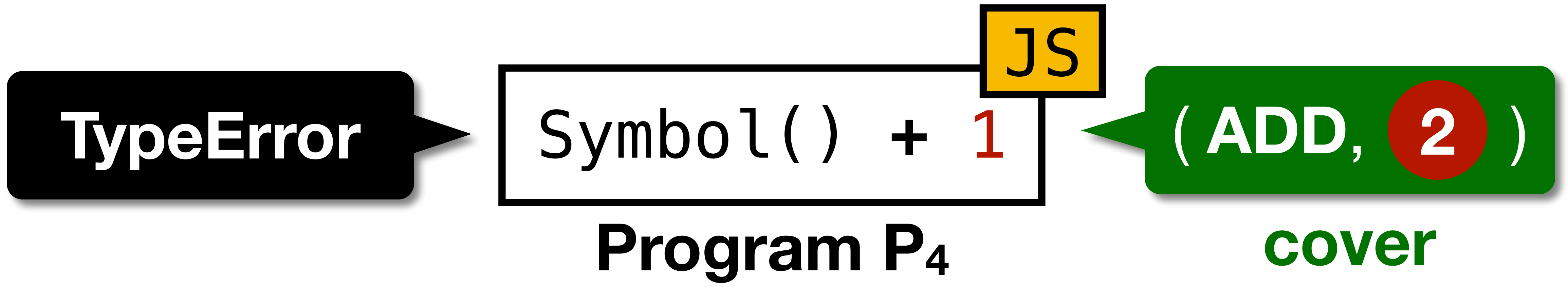
6. If **Type** (*argument*) is Symbol, throw a **TypeError** exception.

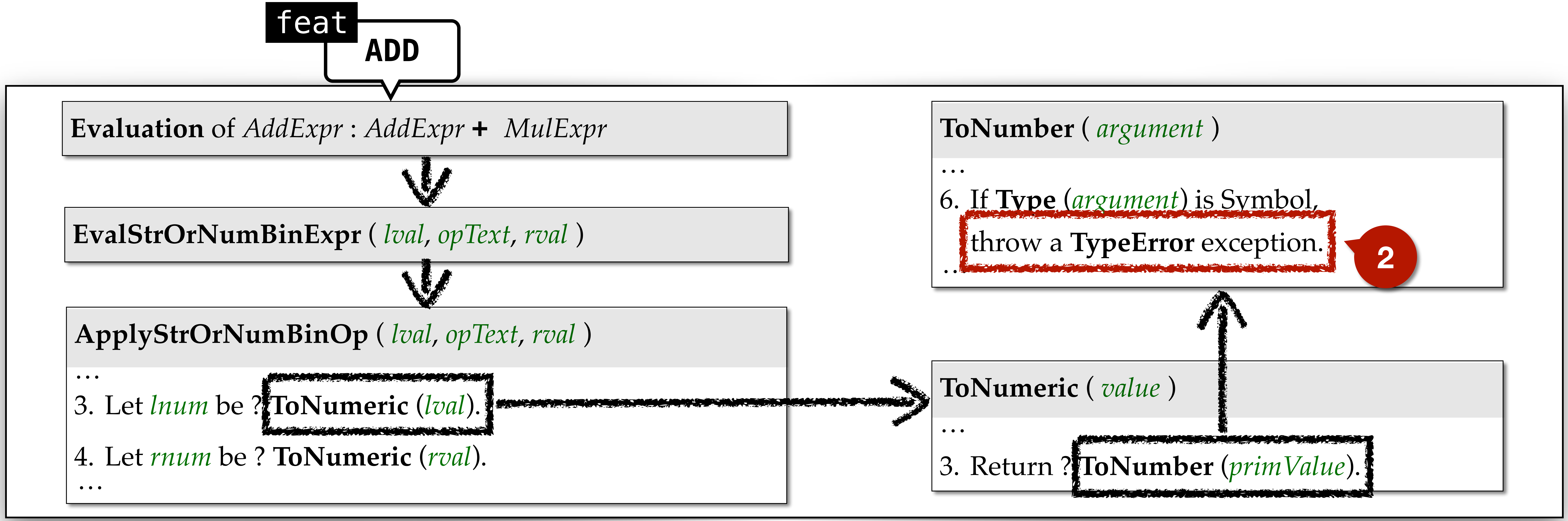
…

Motivating Example 2

1-FS Node Coverage

TR = (Feature, Node)





Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

```
…
3. Let lnum be ? ToNumeric (lval).
4. Let rnum be ? ToNumeric (rval).
…
```

ToNumeric (*value*)

```
…
3. Return ? ToNumber (primValue).
```

ToNumber (*argument*)

```
…
6. If Type (argument) is Symbol,
   throw a TypeError exception.
…
```

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 2

1-FS Node Coverage

TR = (Feature, Node)

TypeError ▶
```
Symbol() + 1
```
JS

Program P4

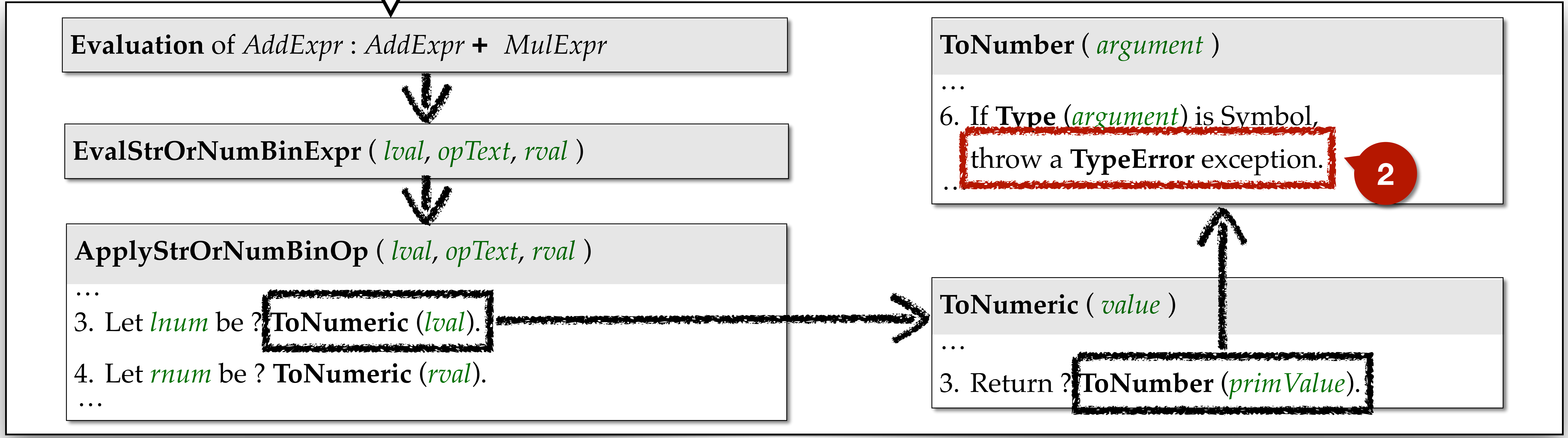
◀ **(ADD, 2)**

cover

TypeError ▶
```
1 + Symbol()
```
JS

Program P5

feat **ADD**



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

ToNumber (*argument*)

…

6. If **Type** (*argument*) is Symbol,
   throw a **TypeError** exception.

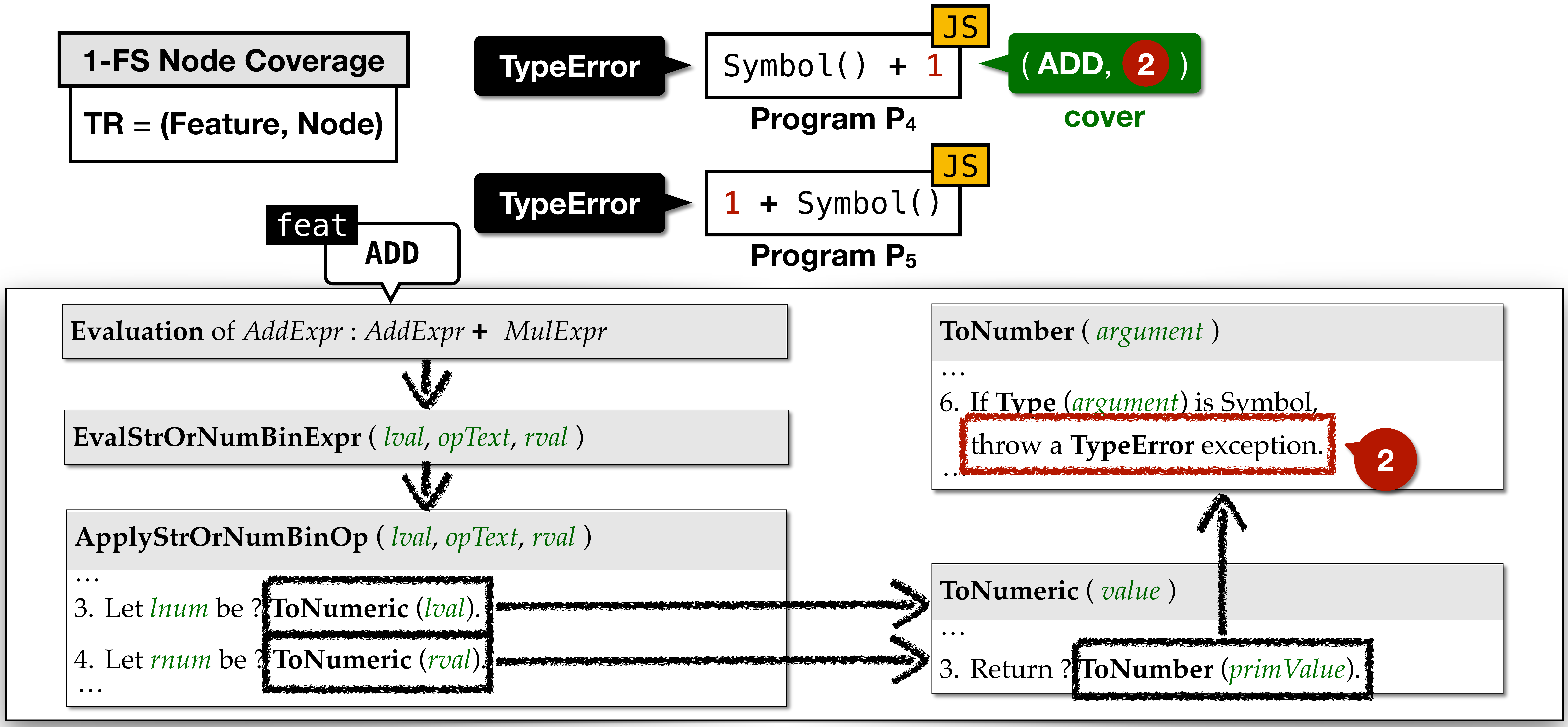
…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 2

1-FS Node Coverage

TR = (Feature, Node)



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

```
…
3. Let lnum be ? ToNumeric (lval).
4. Let rnum be ? ToNumeric (rval).
…
```

ToNumeric (*value*)

```
…
3. Return ? ToNumber (primValue).
```

ToNumber (*argument*)

```
…
6. If Type (argument) is Symbol,
   throw a TypeError exception.
…
```

PLRG   Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 2

1-FS Node Coverage

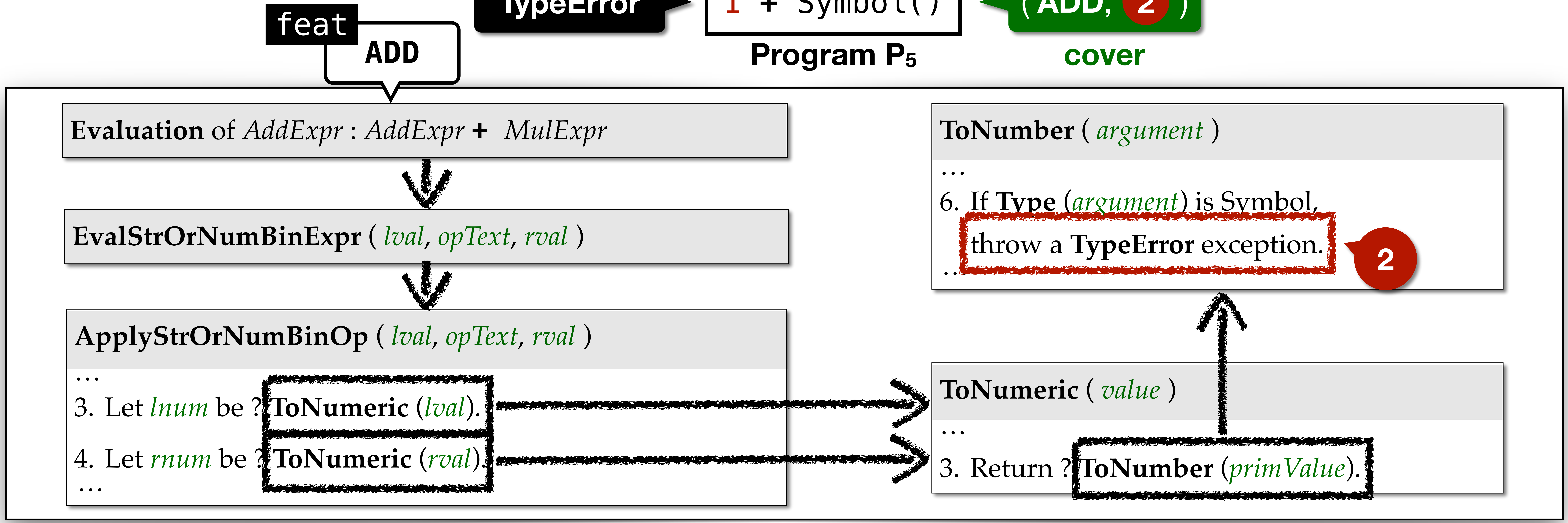
TR = (Feature, Node)

TypeError ▶ `Symbol() + 1` JS ◀ **(ADD, 2)**

Program P_4 **cover**

TypeError ▶ `1 + Symbol()` JS ◀ **(ADD, 2)**

Program P_5 **cover**



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).
4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

ToNumber (*argument*)

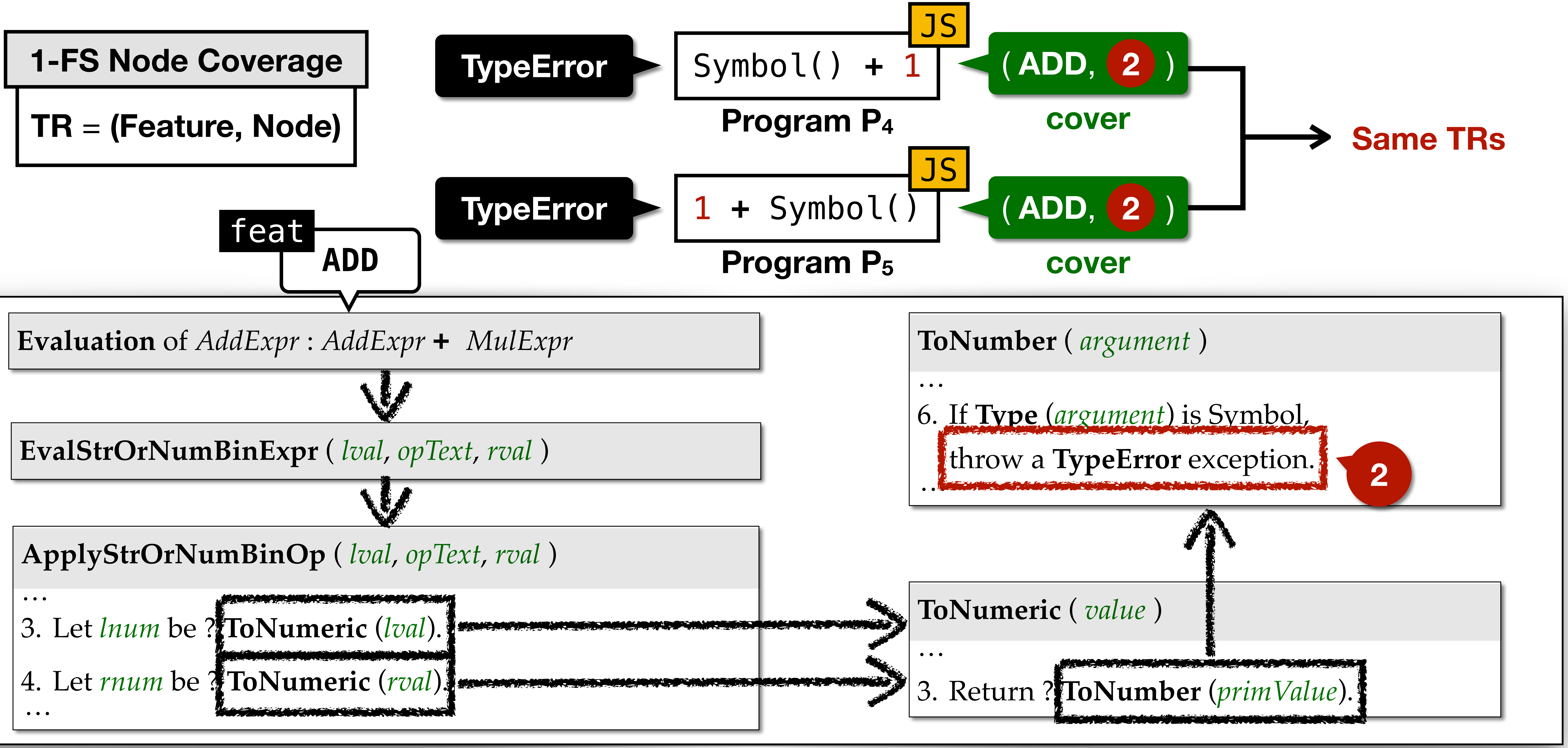
…

6. If **Type** (*argument*) is Symbol, throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 2



1-FS Node Coverage

TR = (Feature, Node)

TypeError → `Symbol() + 1` (JS) — Program P_4 — (ADD, 2) cover

TypeError → `1 + Symbol()` (JS) — Program P_5 — (ADD, 2) cover

→ Same TRs

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

ToNumber (*argument*)

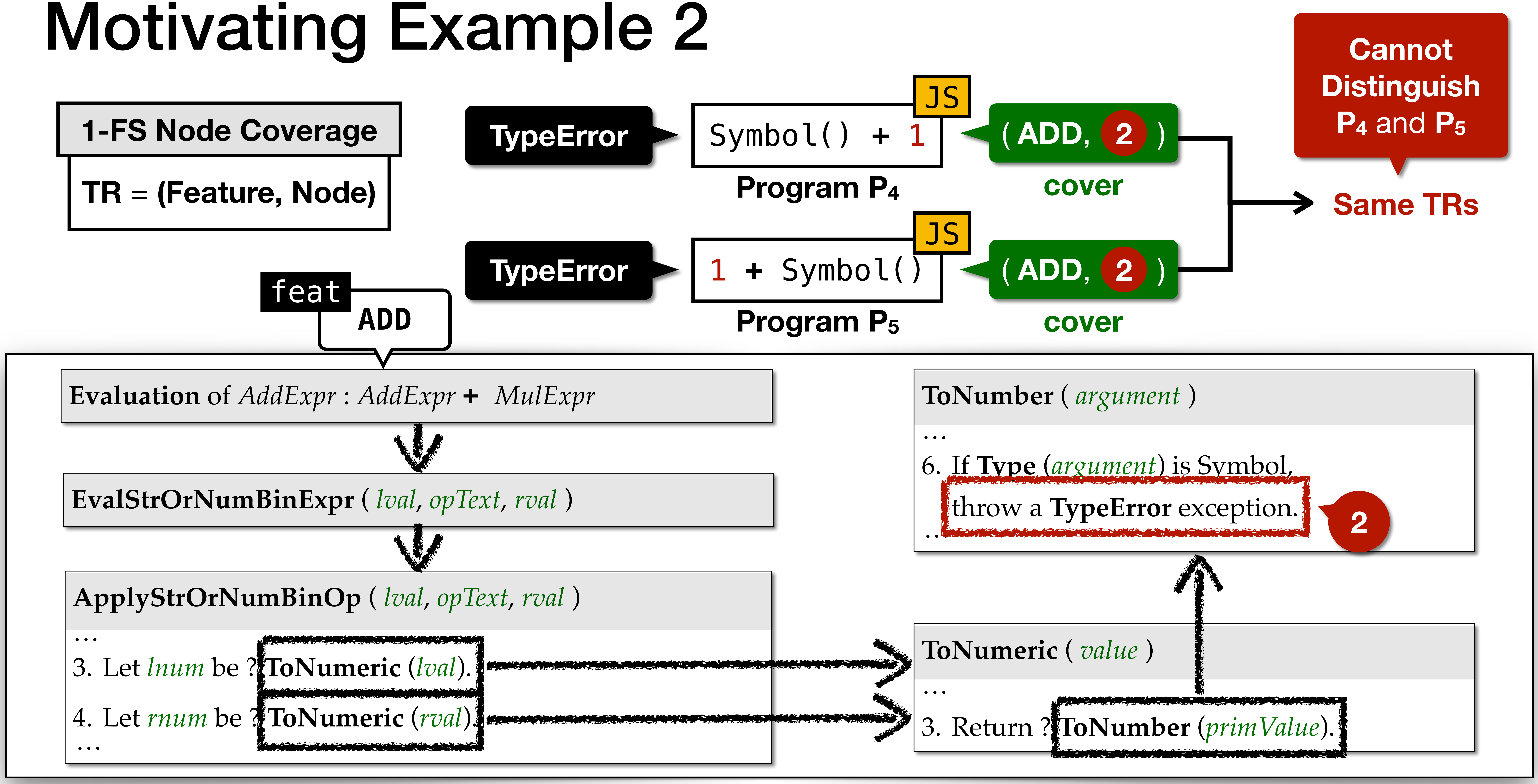
…

6. If **Type** (*argument*) is Symbol, throw a **TypeError** exception.

…

PLRG Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

Motivating Example 2



1-FS Node Coverage

TR = (Feature, Node)

TypeError → `Symbol() + 1` JS ← **(ADD, 2)** **cover**

Program P_4

TypeError → `1 + Symbol()` JS ← **(ADD, 2)** **cover**

Program P_5

→ **Same TRs**

Cannot Distinguish P_4 and P_5

feat **ADD**

Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

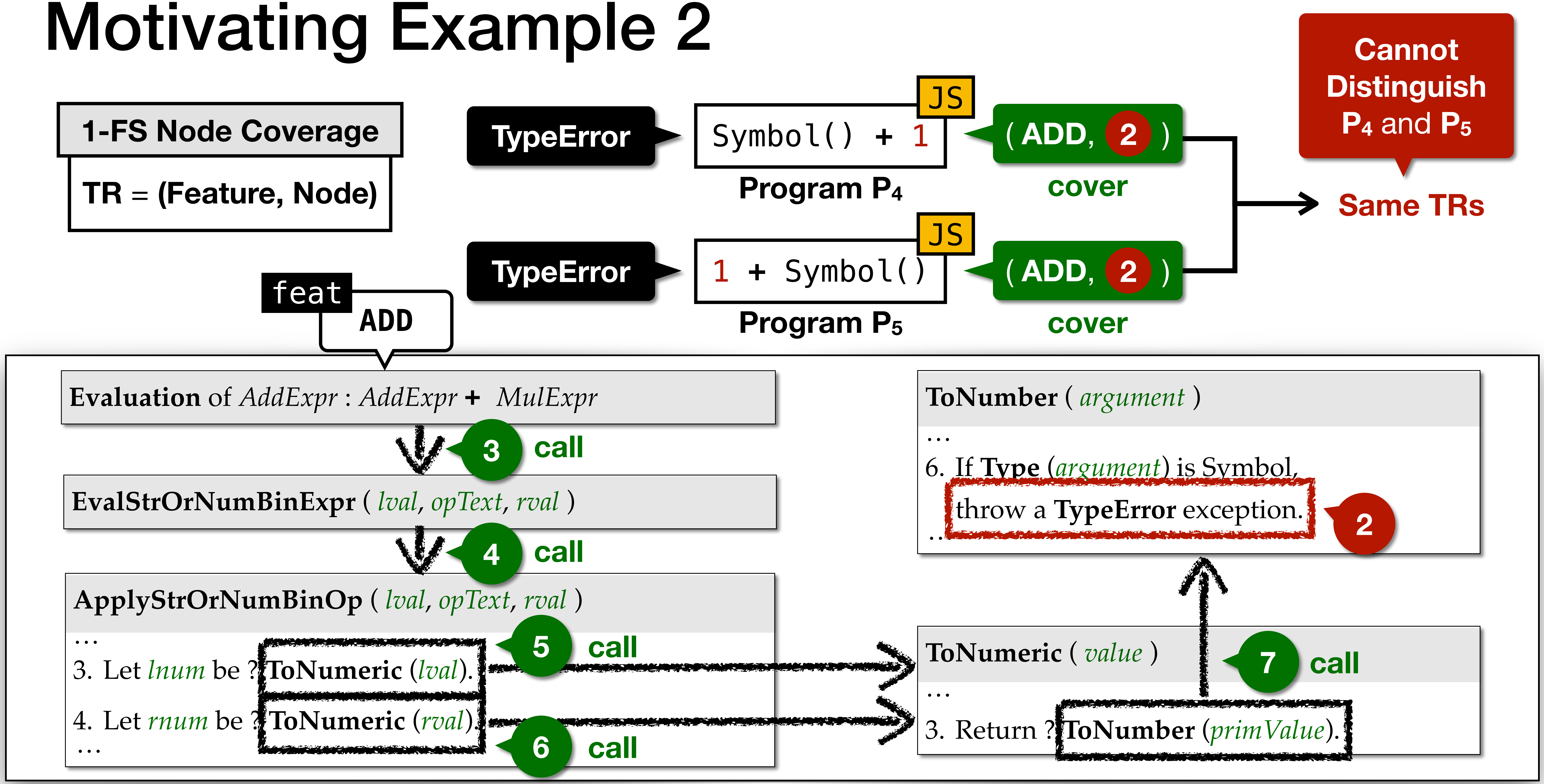
ToNumber (*argument*)

…

6. If **Type** (*argument*) is Symbol, throw a **TypeError** exception.

…

Motivating Example 2



Evaluation of *AddExpr* : *AddExpr* **+** *MulExpr*

EvalStrOrNumBinExpr (*lval*, *opText*, *rval*)

ApplyStrOrNumBinOp (*lval*, *opText*, *rval*)

…

3. Let *lnum* be ? **ToNumeric** (*lval*).

4. Let *rnum* be ? **ToNumeric** (*rval*).

…

ToNumeric (*value*)

…

3. Return ? **ToNumber** (*primValue*).

ToNumber (*argument*)

…

6. If **Type** (*argument*) is Symbol, throw a **TypeError** exception.

…

Abstract Algorithms in **ECMA-262** (ES13, 2022), **JavaScript Language Specification**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage

TypeError

JS

```
Symbol() + 1
```

Program P_4

TypeError

JS

```
1 + Symbol()
```

Program P_5

- k-Feature-Call-Path-Sensitive (**k-FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given TR)

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage

TypeError

JS

```
Symbol() + 1
```

Program P_4

TypeError

JS

```
1 + Symbol()
```

Program P_5

1-FCPS Node Coverage

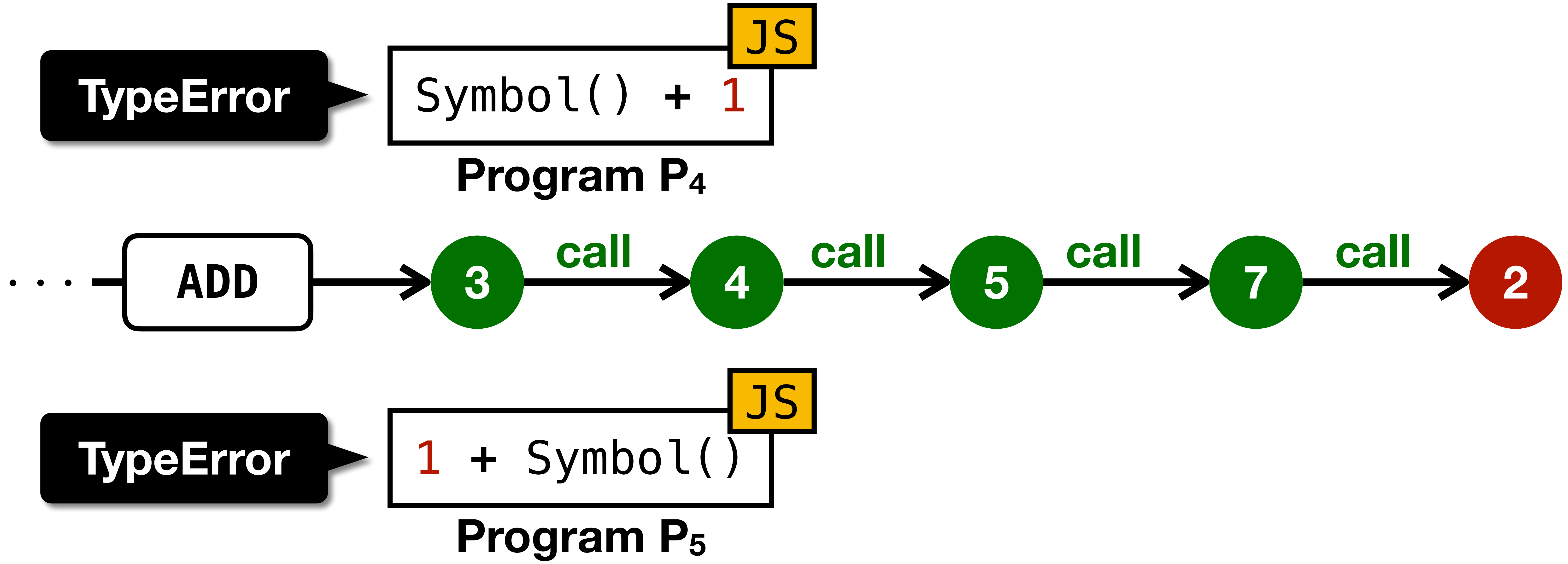
TR = (Feature, Call-Path, Node)

- k-Feature-Call-Path-Sensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



1-FCPS Node Coverage

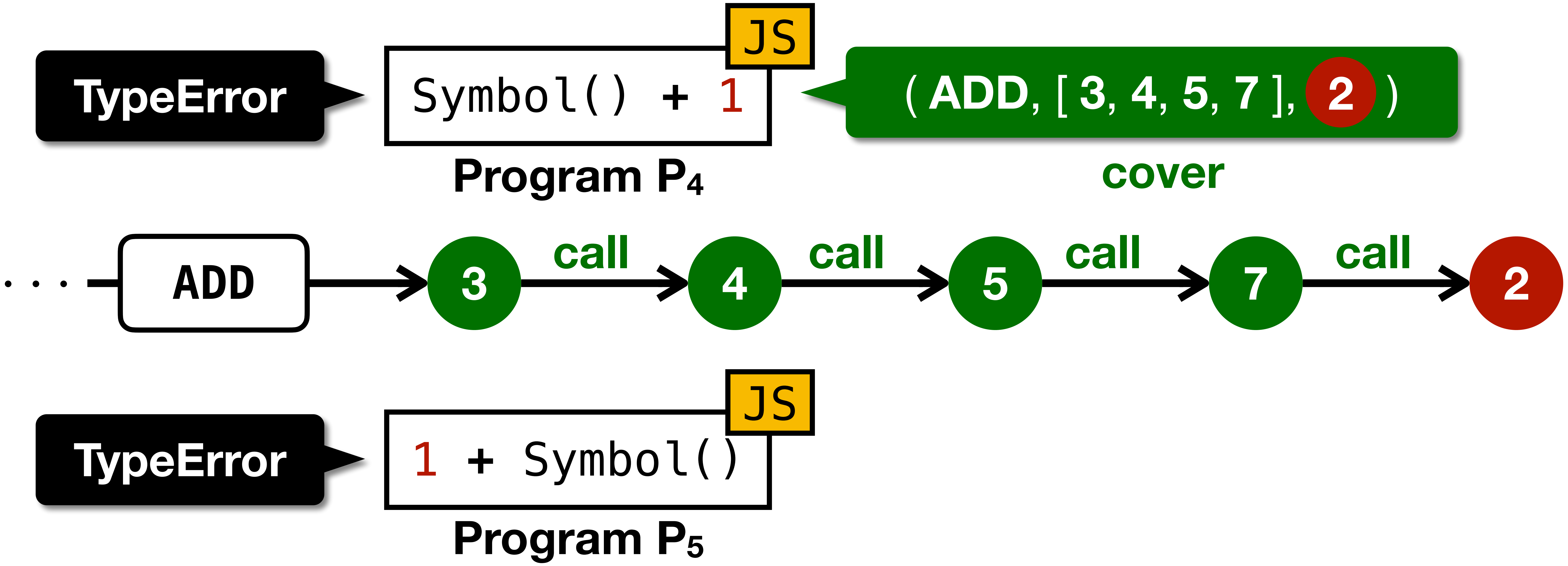
TR = (Feature, Call-Path, Node)

- k-Feature-Call-Path-Sensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



```
Symbol() + 1
```

Program P_4

```
1 + Symbol()
```

Program P_5

1-FCPS Node Coverage

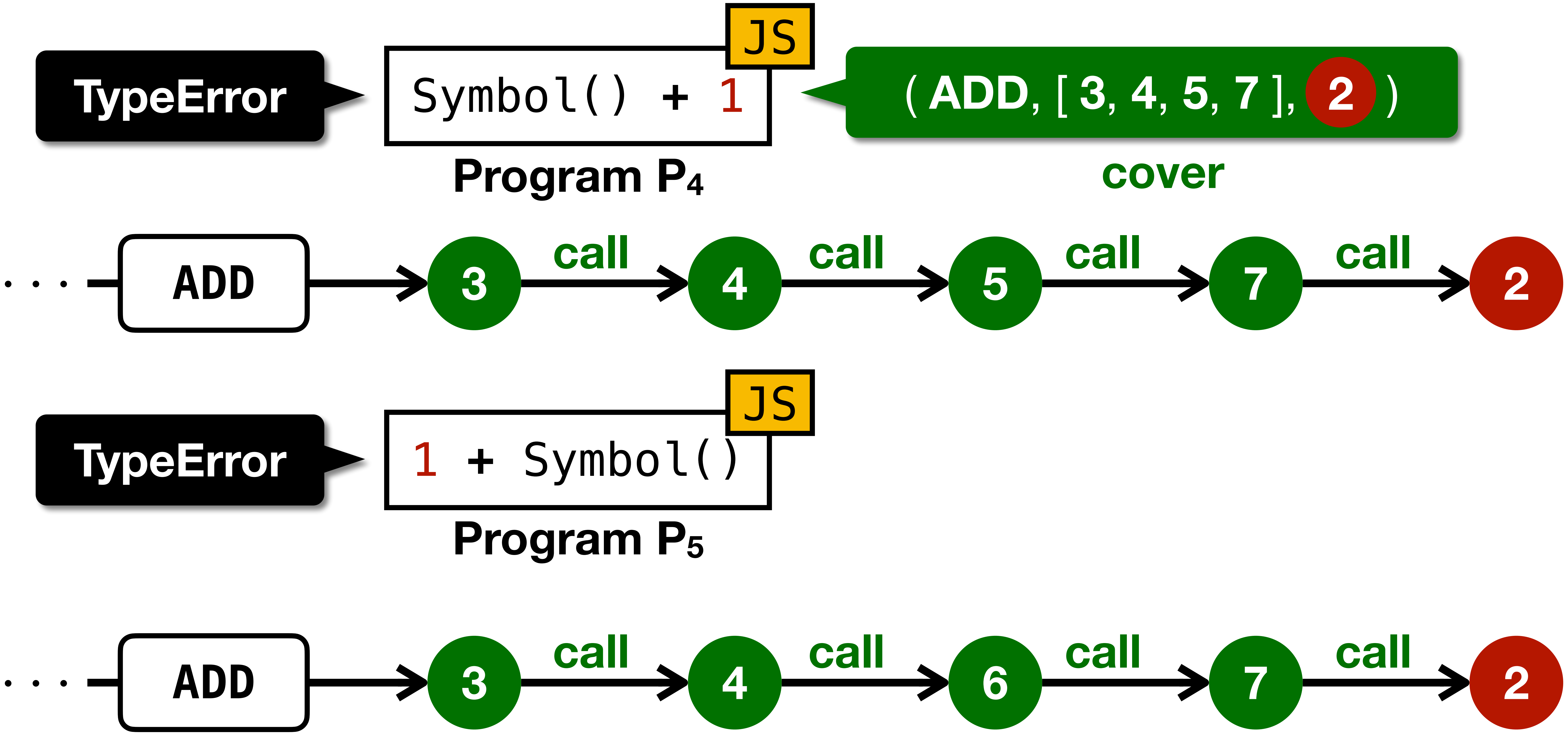
TR = (Feature, Call-Path, Node)

- k-**F**eature-**C**all-**P**ath-**S**ensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\le k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



TypeError

```
Symbol() + 1
```

JS

Program P_4

(ADD, [3, 4, 5, 7], 2)

cover

· · · ADD → 3 —call→ 4 —call→ 5 —call→ 7 —call→ 2

TypeError

```
1 + Symbol()
```

JS

Program P_5

· · · ADD → 3 —call→ 4 —call→ 6 —call→ 7 —call→ 2

1-FCPS Node Coverage

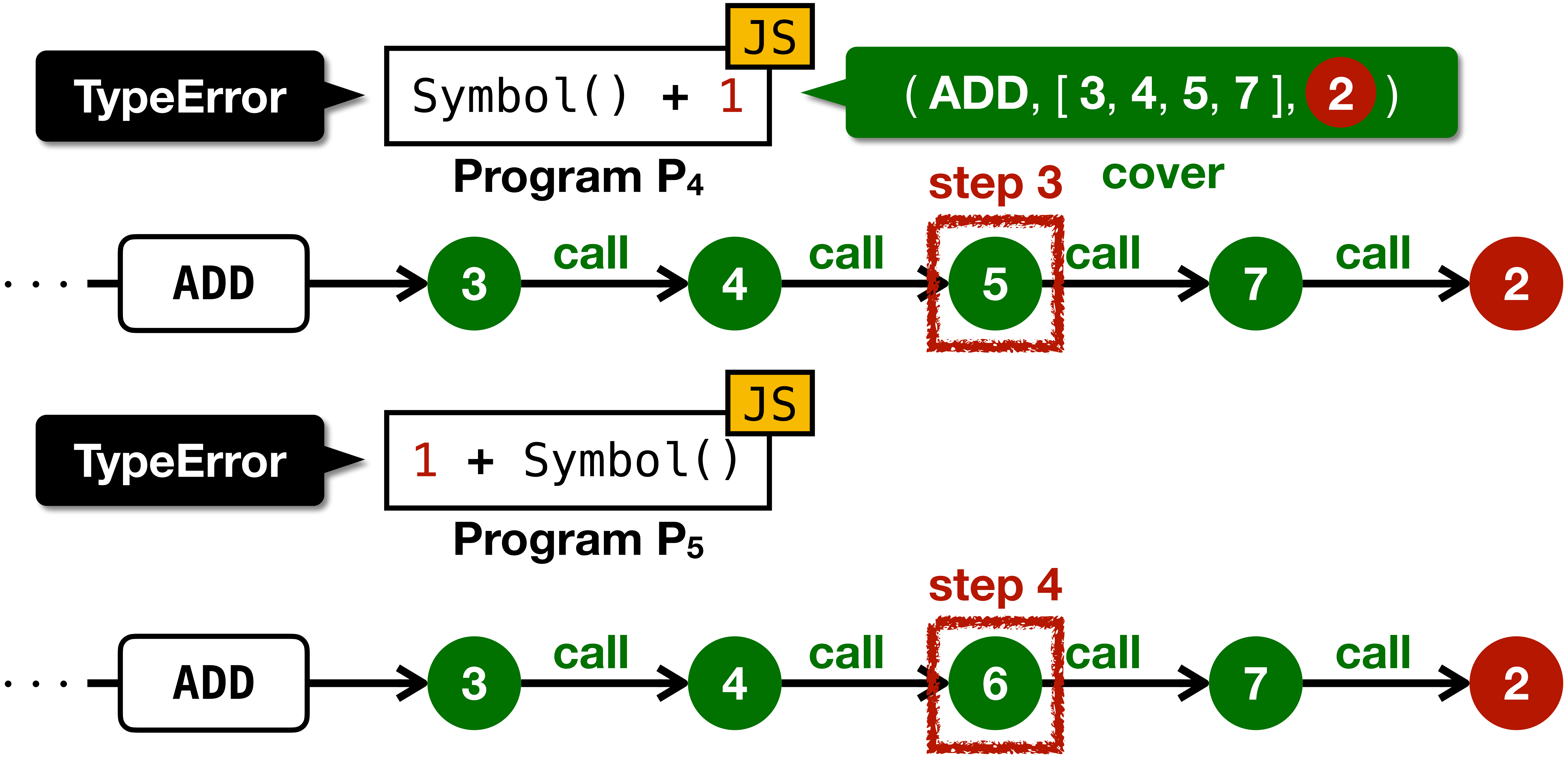
TR = (Feature, Call-Path, Node)

- k-**F**eature-**C**all-**P**ath-**S**ensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



```
Symbol() + 1
```

```
1 + Symbol()
```

1-FCPS Node Coverage

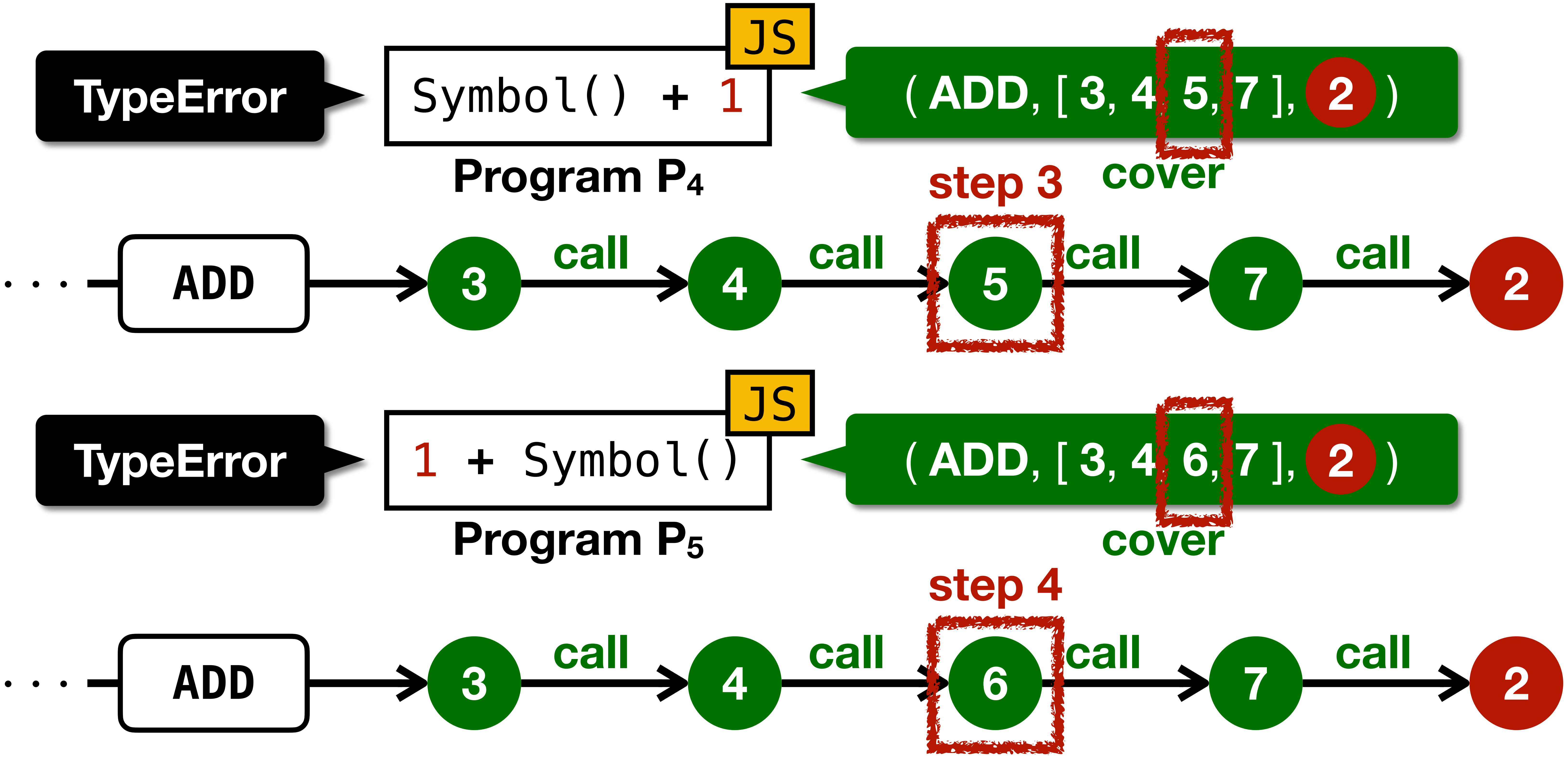
TR = (Feature, Call-Path, Node)

- k-**F**eature-**C**all-**P**ath-**S**ensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



1-FCPS Node Coverage

TR = (Feature, Call-Path, Node)

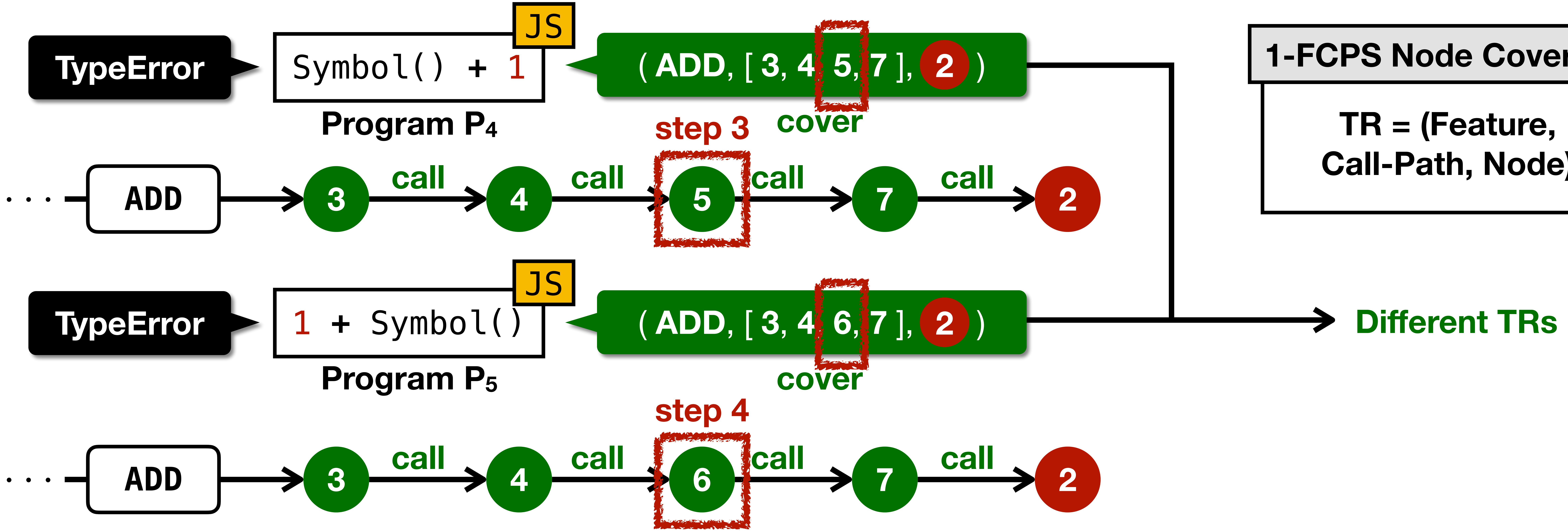
- k-**F**eature-**C**all-**P**ath-**S**ensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

PLRG

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



1-FCPS Node Coverage

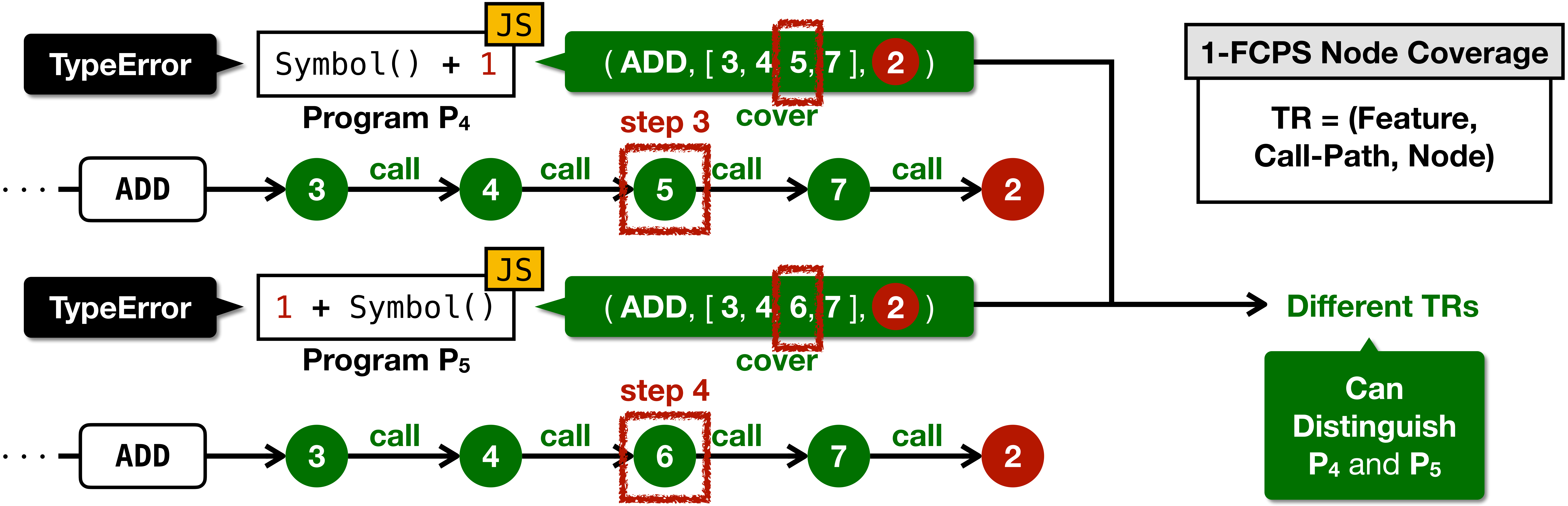
TR = (Feature, Call-Path, Node)

- k-**F**eature-**C**all-**P**ath-**S**ensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

k-Feature-Call-Path-Sensitive (k-FCPS) Coverage



- k-Feature-Call-Path-Sensitive (k-**FCPS**) coverage criterion divides the k-FS TRs with the **call-paths from** the innermost enclosing language feature

k-FCPS Coverage

TR = (Feature$^{\leq k}$, Call-Path, given **TR)**

PLRG

Evaluation

- **Evaluation** with **ES2022** in 50 hours with **0-FS** / **1-FS** / **2-FS** / **1-FCPS** / **2-FCPS**

Kind	Name	Version	Release	# Detected Unique Bugs		
				# New	# Confirmed	# Reported
Engine	V8	v10.8.121	2022.10.06	0	0	4
	JSC	v615.1.10	2022.10.26	15	15	24
	GraalJS	v22.2.0	2022.07.26	9	9	10
	SpiderMonkey	v107.0b4	2022.10.24	1	3	4
	Total			**25**	**27**	**42**
Transpiler	Babel	v7.19.1	2022.09.15	30	30	35
	SWC	v1.3.10	2022.10.21	27	27	41
	Terser	v5.15.1	2022.10.05	1	1	18
	Obfuscator	v4.0.0	2022.02.15	0	0	7
	Total			**58**	**58**	**101**
Total				**83**	**85**	**143**

Evaluation

- **Evaluation** with **ES2022** in 50 hours with **0-FS** / **1-FS** / **2-FS** / **1-FCPS** / **2-FCPS**

Kind	Name	Version	Release	# Detected Unique Bugs		
				# New	# Confirmed	# Reported
Engine	V8	v10.8.121	2022.10.06	0	0	4
	JSC	v615.1.10	2022.10.26	15	15	24
	GraalJS	v22.2.0	2022.07.26	9	9	10
	SpiderMonkey	v107.0b4	2022.10.24	1	3	4
	Total			**25**	**27**	**42**
Transpiler	Babel	v7.19.1	2022.09.15	30	30	35
	SWC	v1.3.10	2022.10.21	27	27	41
	Terser	v5.15.1	2022.10.05	1	1	18
	Obfuscator	v4.0.0	2022.02.15	0	0	7
	Total			**58**	**58**	**101**
Total				**83**	**85**	**143**

Evaluation

- **Evaluation** with **ES2022** in 50 hours with **0-FS** / **1-FS** / **2-FS** / **1-FCPS** / **2-FCPS**

Kind	**Name**	**Version**	**Release**	**# Detected Unique Bugs**		
				# New	**# Confirmed**	**# Reported**
Engine	V8	v10.8.121	2022.10.06	0	0	4
	JSC	v615.1.10	2022.10.26	15	15	24
	GraalJS	v22.2.0	2022.07.26	9	9	10
	SpiderMonkey	v107.0b4	2022.10.24	1	3	4
	Total			**25**	**27**	**42**
Transpiler	Babel	v7.19.1	2022.09.15	30	30	35
	SWC	v1.3.10	2022.10.21	27	27	41
	Terser	v5.15.1	2022.10.05	1	1	18
	Obfuscator	v4.0.0	2022.02.15	0	0	7
	Total			**58**	**58**	**101**
Total				**83**	**85**	**143**

Evaluation

- **Evaluation** with **ES2022** in 50 hours with **0-FS** / **1-FS** / **2-FS** / **1-FCPS** / **2-FCPS**

Kind	Name	Version	Release	# Detected Unique Bugs		
				# New	# Confirmed	# Reported
Engine	V8	v10.8.121	2022.10.06	0	0	4
	JSC	v615.1.10	2022.10.26	15	15	24
	GraalJS	v22.2.0	2022.07.26	9	9	10
	SpiderMonkey	v107.0b4	2022.10.24	1	3	4
	Total			**25**	**27**	**42**
Transpiler	Babel	v7.19.1	2022.09.15	30	30	35
	SWC	v1.3.10	2022.10.21	27	27	41
	Terser	v5.15.1	2022.10.05	1	1	18
	Obfuscator	v4.0.0	2022.02.15	0	0	7
	Total			**58**	**58**	**101**
Total				**83**	**85**	**143**

Effectiveness of k-FS / k-FCPS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	**# Syn. Test**	**# Bug**
0-FS node-or-branch (0-fs)	2,111	55
1-FS node-or-branch (1-fs)	6,766	83
1-FCPS node-or-branch (1-fcps)	9,092	87
2-FS node-or-branch (2-fs)	97,423	102
2-FCPS node-or-branch (2-fcps)	122,589	111

Effectiveness of k-FS / k-FCPS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	**# Syn. Test**	**# Bug**
0-FS node-or-branch (0-fs)	2,111	55
1-FS node-or-branch (1-fs)	6,766	83
1-FCPS node-or-branch (1-fcps)	9,092	87
2-FS node-or-branch (2-fs)	97,423	102
2-FCPS node-or-branch (2-fcps)	122,589	111

+28

Effectiveness of k-FS / k-FCPS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	**# Syn. Test**	**# Bug**
0-FS node-or-branch (0-fs)	2,111	55
1-FS node-or-branch (1-fs)	6,766	83
1-FCPS node-or-branch (1-fcps)	9,092	87
2-FS node-or-branch (2-fs)	97,423	102
2-FCPS node-or-branch (2-fcps)	122,589	111

+28

+19

Effectiveness of *k*-FS / *k*-FCPS Coverage Criteria

Coverage Criteria $C_{\mathbb{G}}$	**# Syn. Test**	**# Bug**
0-FS node-or-branch (0-fs)	2,111	55
1-FS node-or-branch (1-fs)	6,766	83
1-FCPS node-or-branch (1-fcps)	9,092	87
2-FS node-or-branch (2-fs)	97,423	102
2-FCPS node-or-branch (2-fcps)	122,589	111

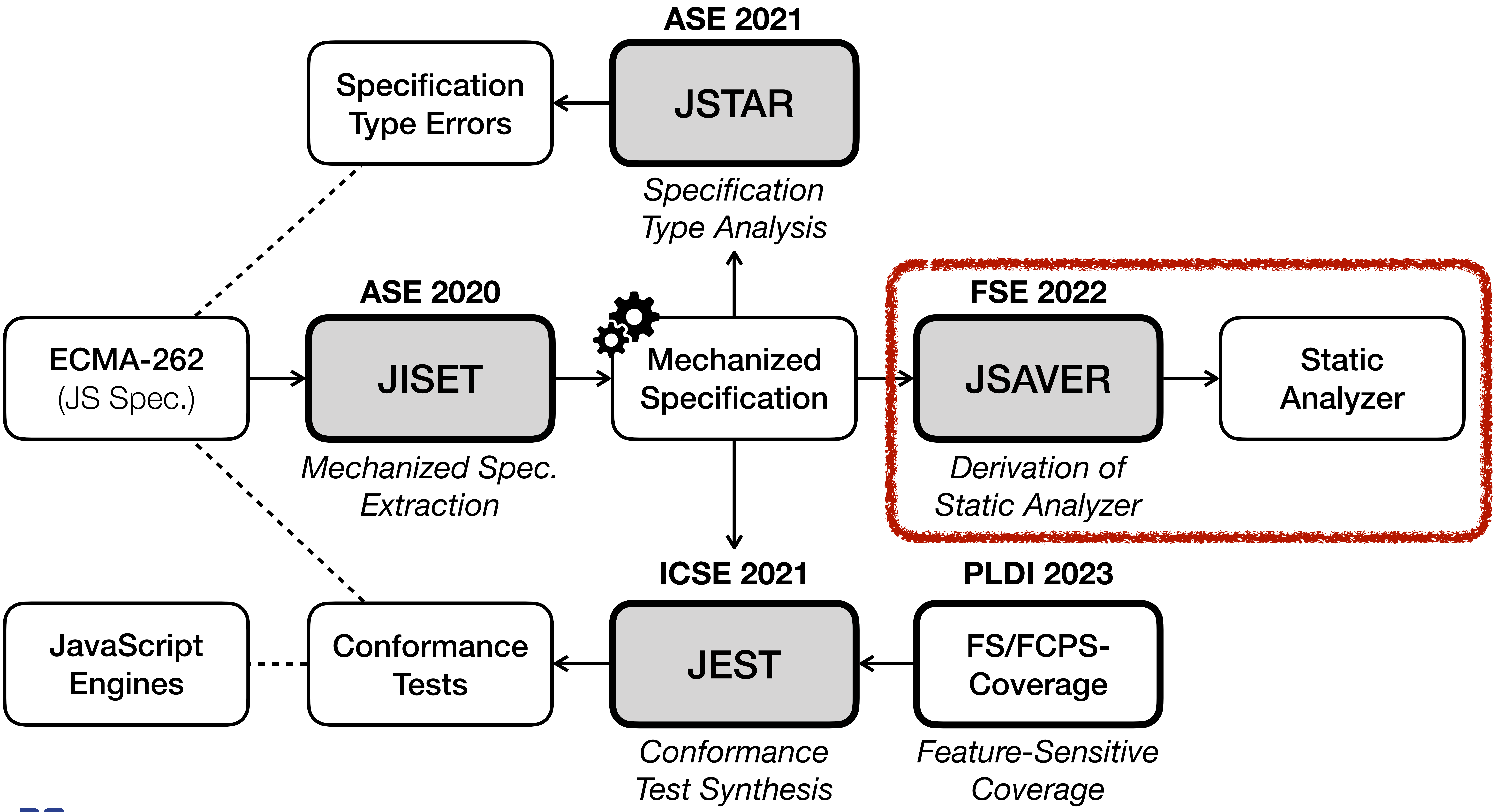
+28

+4

+19

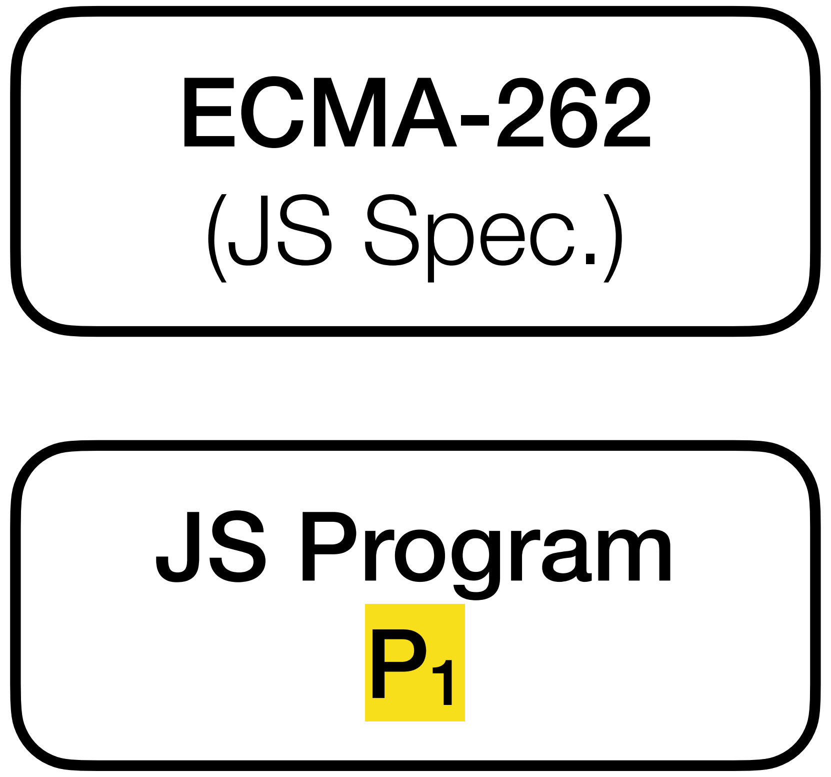
+9

[FSE 2022] J. Park et al., “Automatically Deriving JavaScript Static Analyzers from Specifications using Meta-level Static Analysis”



Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?

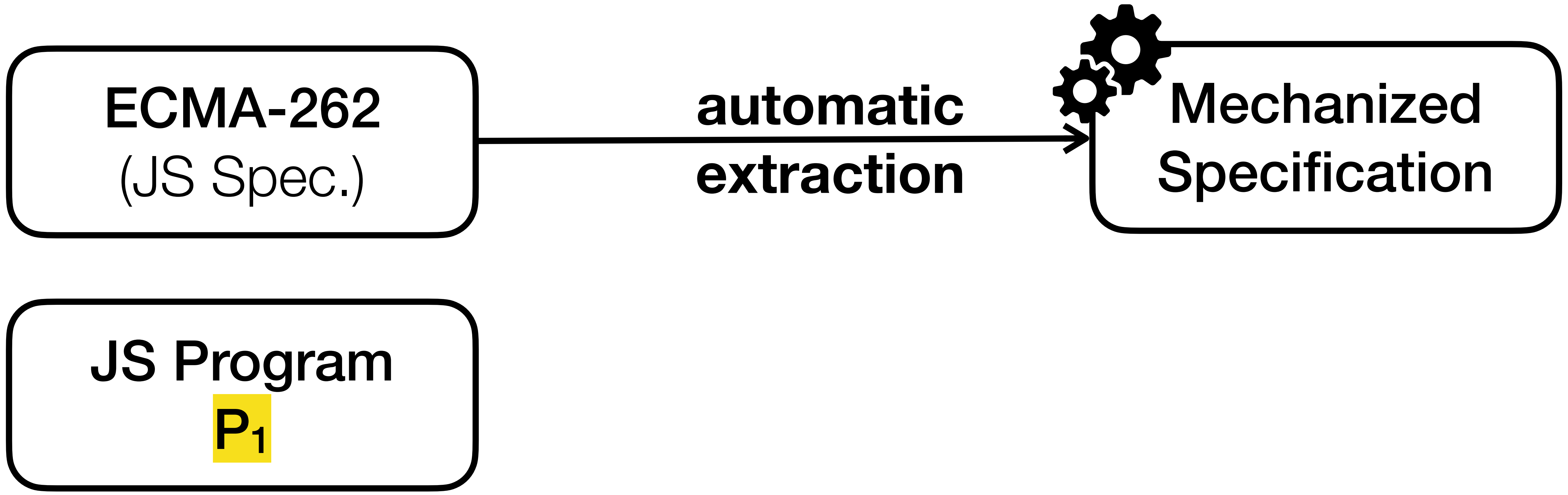


ECMA-262
(JS Spec.)

JS Program
P_1

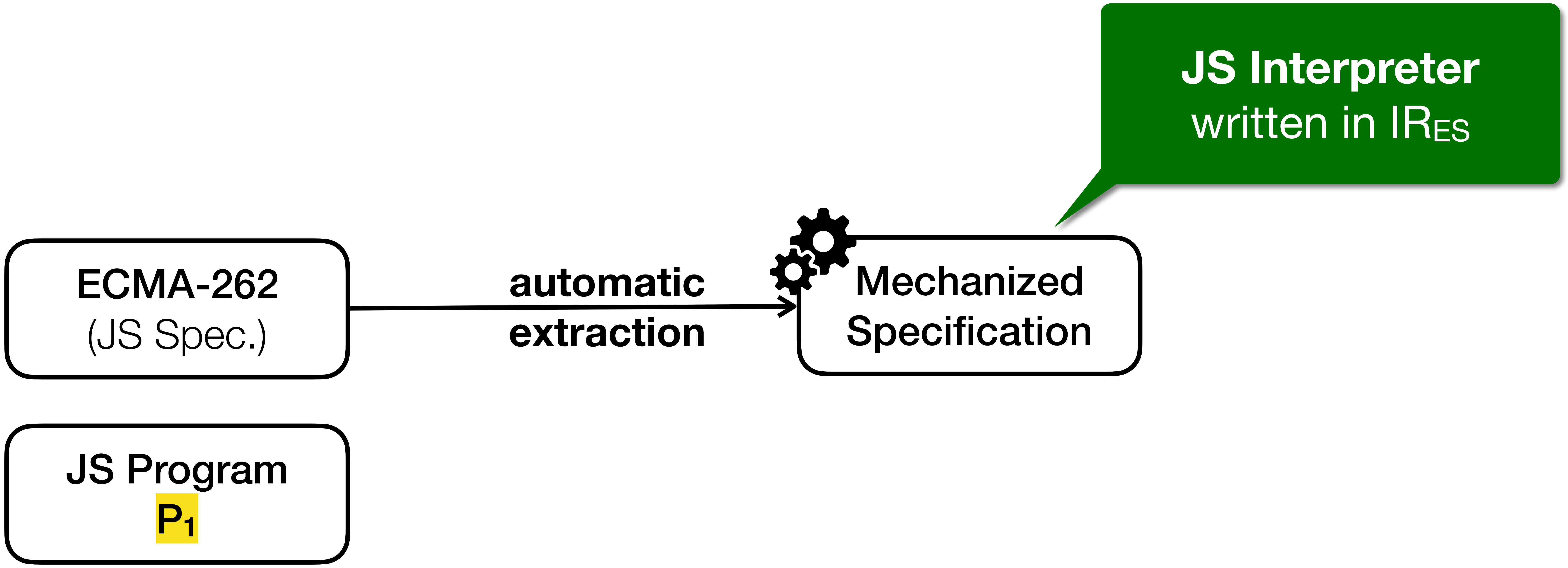
Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



JS Interpreter written in IR_{ES}

ECMA-262 (JS Spec.)

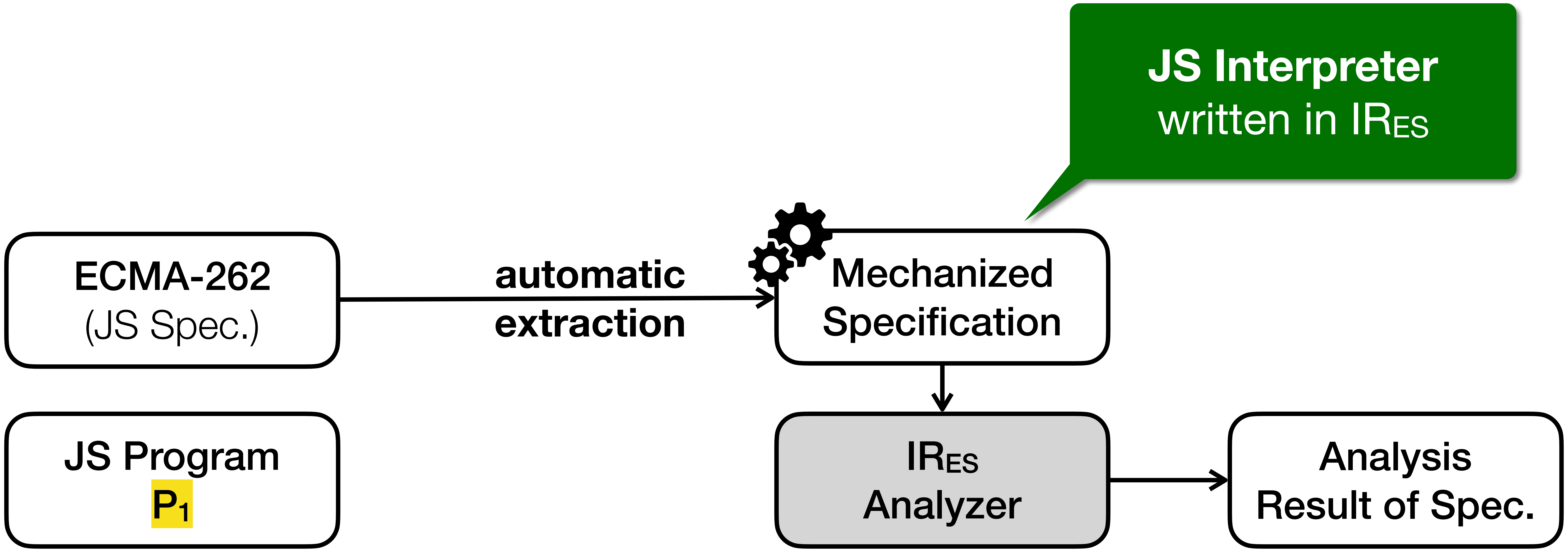
automatic extraction

Mechanized Specification

JS Program P_1

Meta-Level Static Analysis

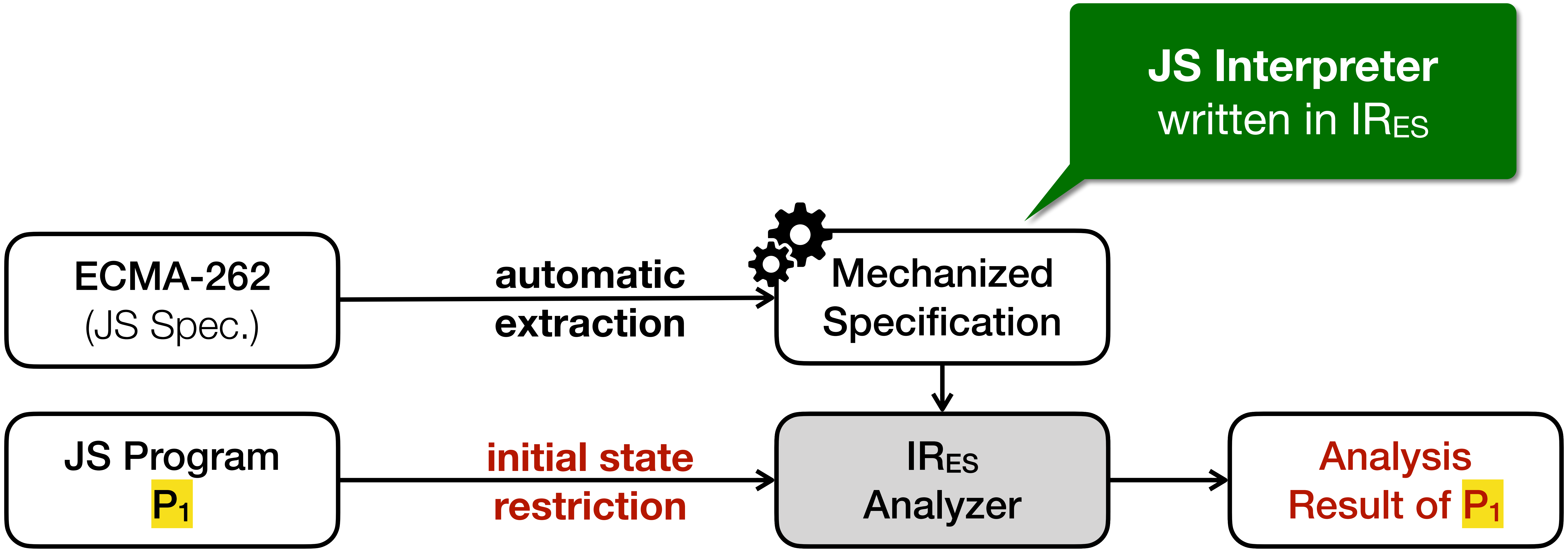
How to perform **static analysis** on **JavaScript programs** using **language specification**?



PLRG

Meta-Level Static Analysis

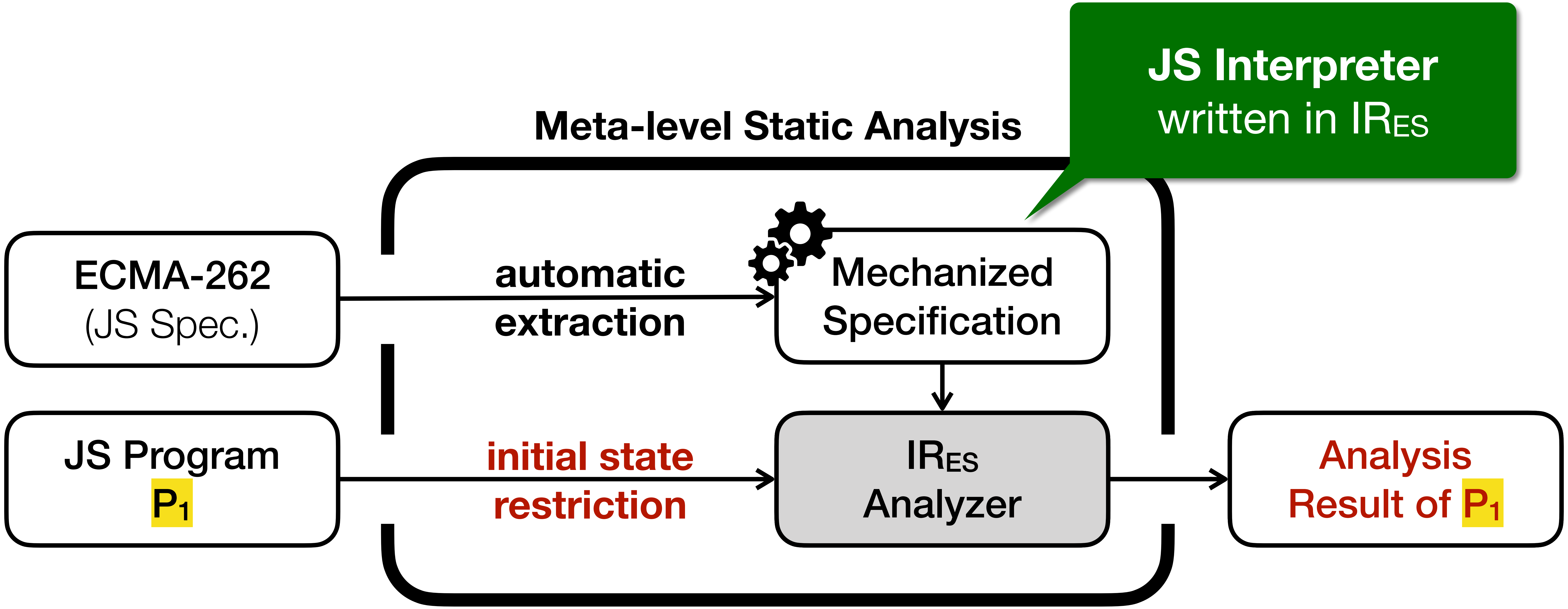
How to perform **static analysis** on **JavaScript programs** using **language specification**?



JS Interpreter written in IR_{ES}

Meta-Level Static Analysis

How to perform **static analysis** on **JavaScript programs** using **language specification**?



PLRG

Meta-Level Static Analysis

JS Program P_1

```
x ||= y
```

JavaScript

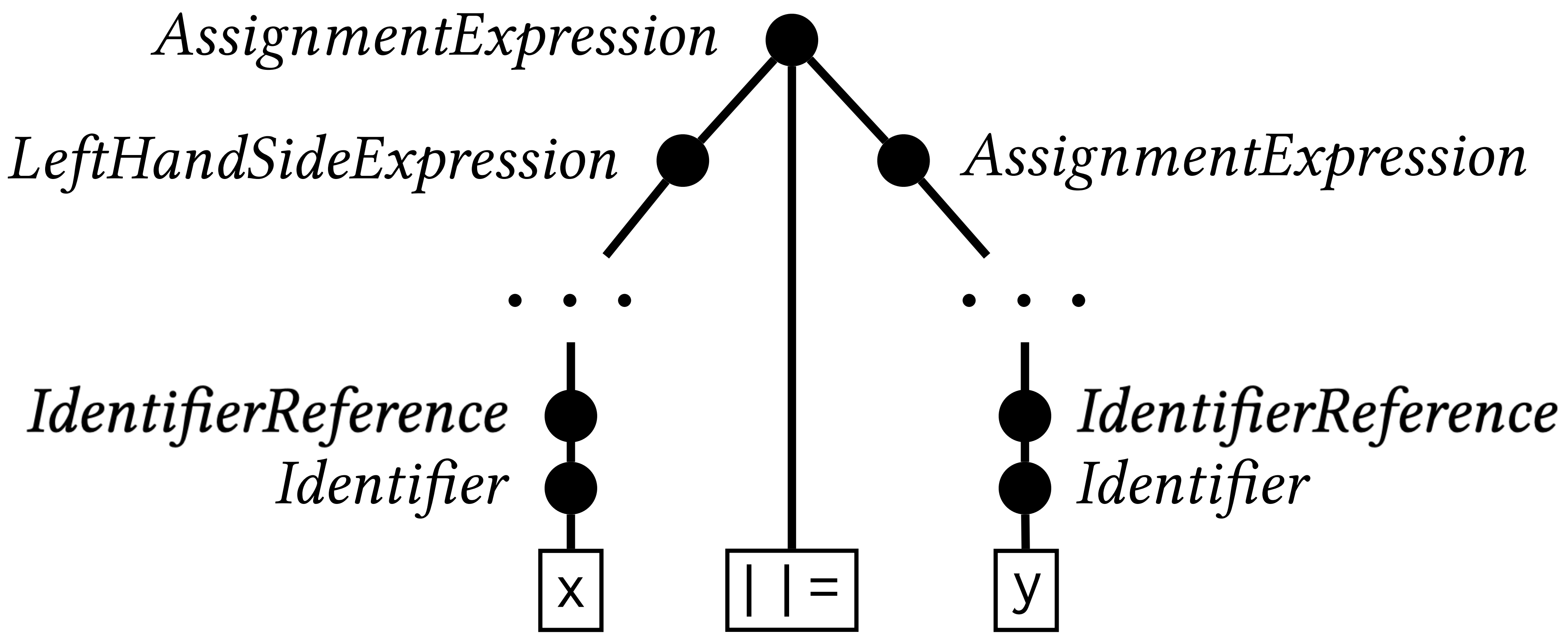
IR$_{ES}$

Meta-Level Static Analysis

JS Program P_1

```
x ||= y
```

parse



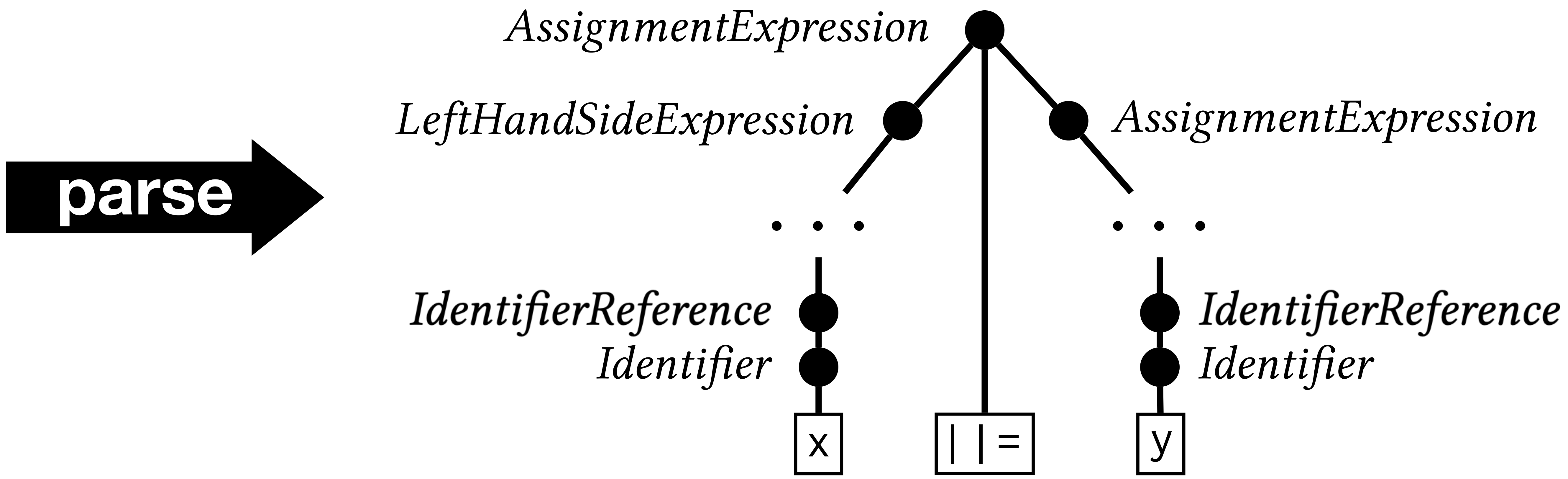
JavaScript

IRES

Meta-Level Static Analysis

JS Program P1

```
x ||= y
```



JavaScript

IR

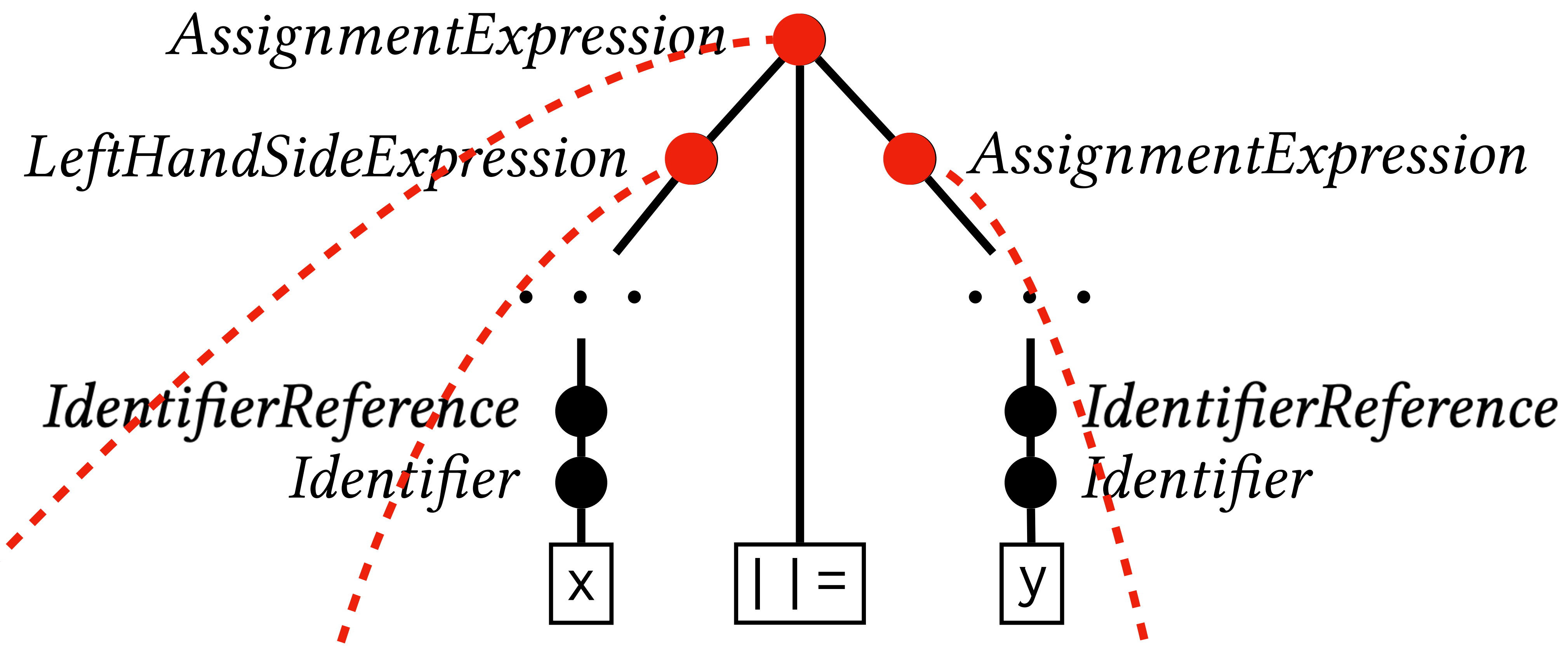
```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

Meta-Level Static Analysis

JS Program P₁

```
x ||= y
```

parse



JavaScript

IR_{ES}

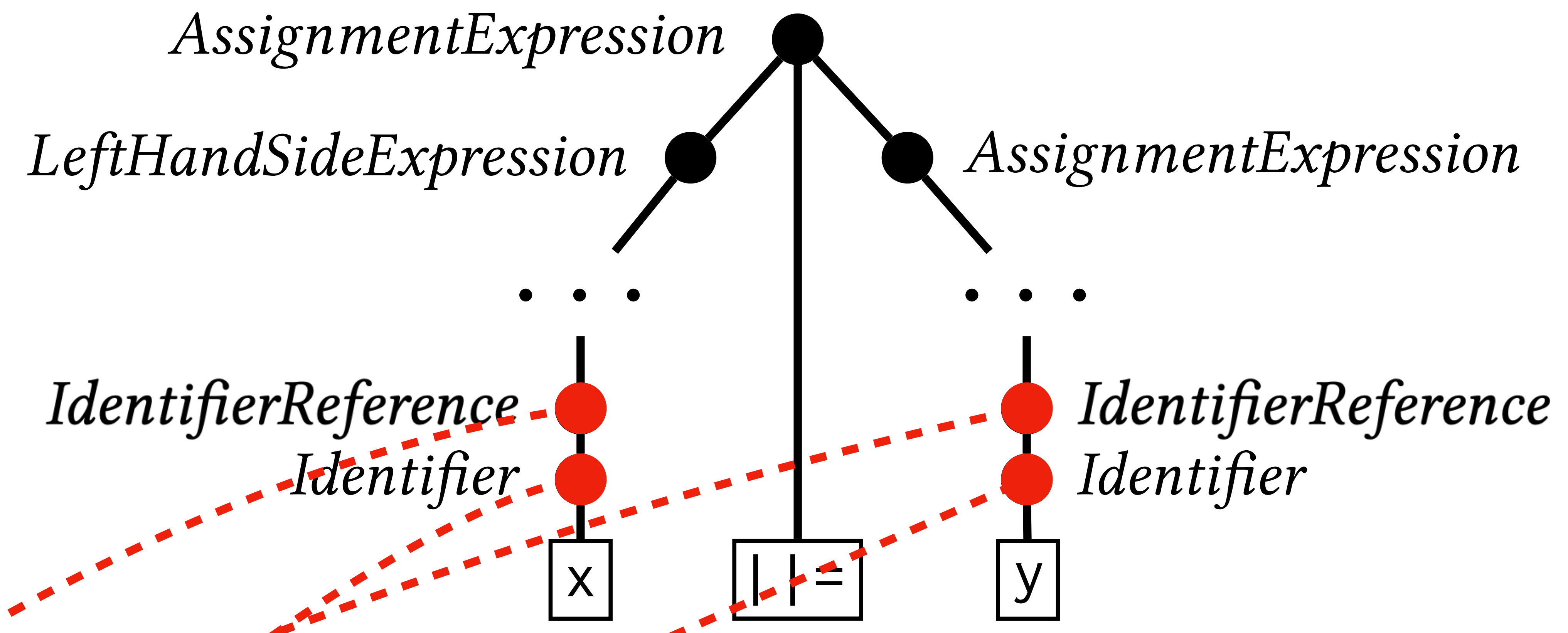
```
syntax def AssignmentExpression[8].Evaluation(
  this, LeftHandSideExpression, AssignmentExpression
) {
  let lref = (LeftHandSideExpression.Evaluation)
  let lval = [? (GetValue lref)]
  let lbool = [! (ToBoolean lval)]
  if (= lbool true) return lval
  ...
}
```

AST Sensitivity

JS Program P_1

```
x ||= y
```

parse



JavaScript

IR_{ES}

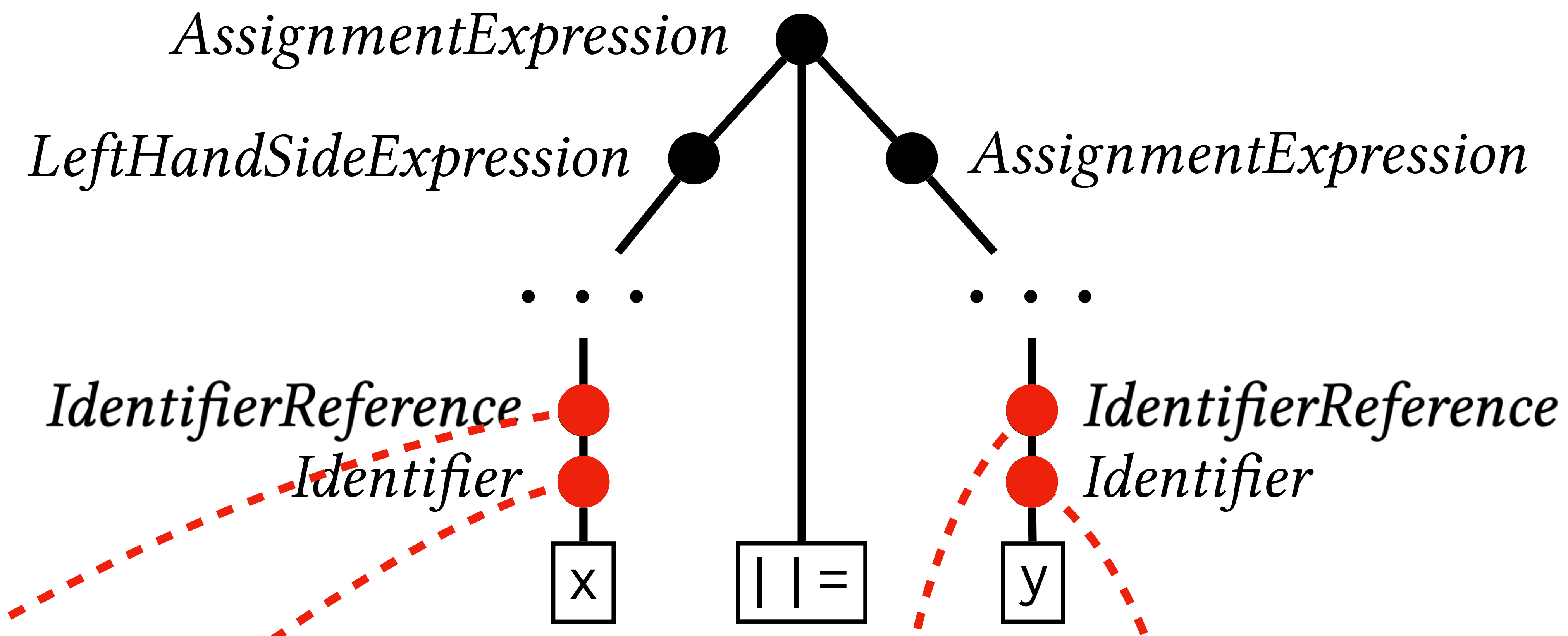
```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

AST Sensitivity

JS Program P₁

```
x ||= y
```

parse



JavaScript

IR$_{ES}$

```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

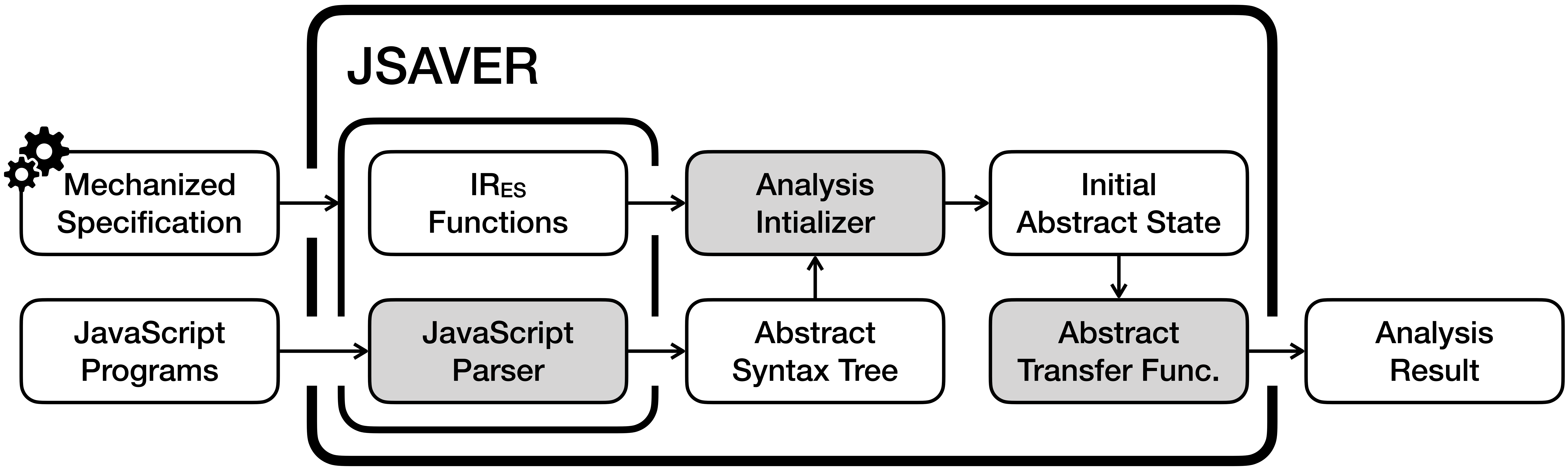
```
syntax def IdentifierReference[0]
  .Evaluation(
  this, Identifier
) {
  return [?
    (ResolveBinding
      (Identifier.StringValue))]
}
```

AST Sensitivity

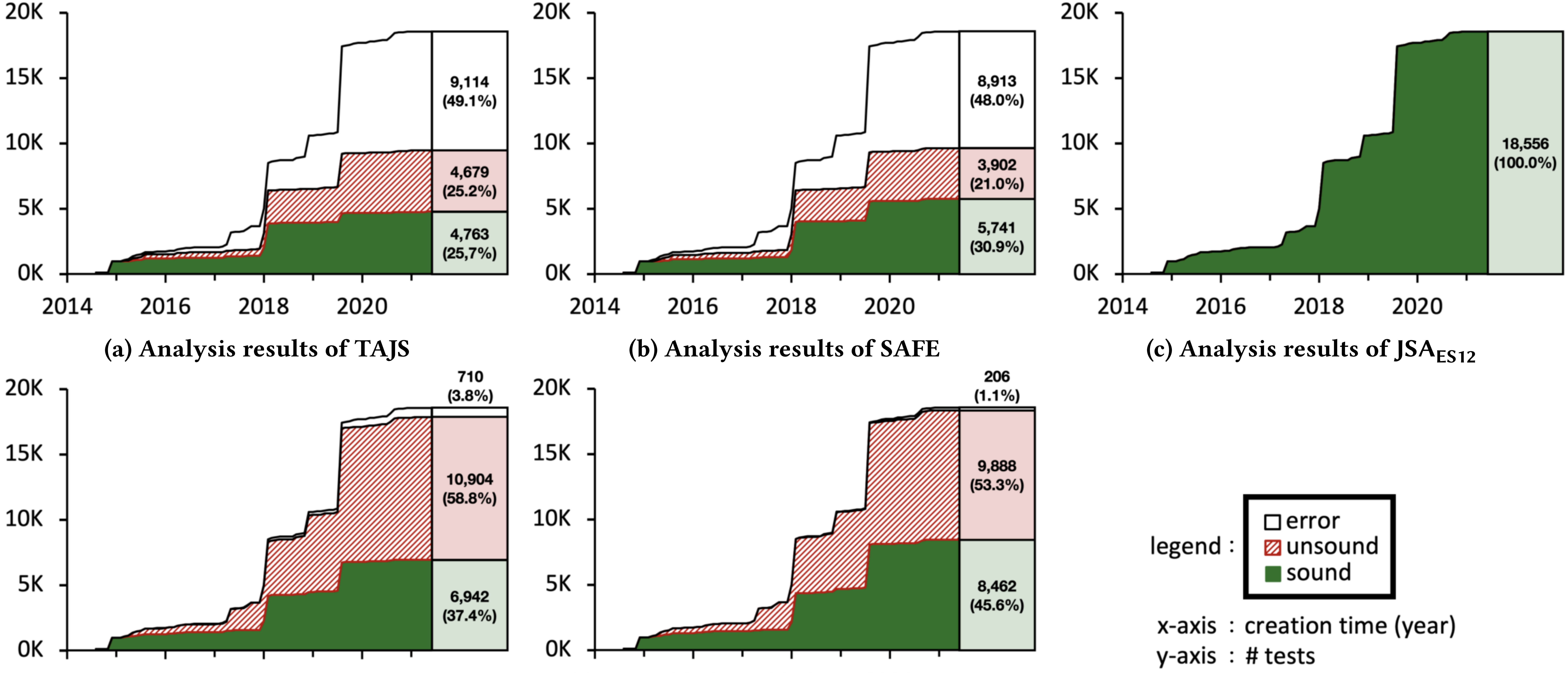
JavaScript	AST Sensitivity in IR$_{\text{ES}}$
Flow-Sensitivity	$\delta^{\texttt{js-flow}}(t_\bot) = \{\sigma = (\_, \_, \bar{c}, \_) \in \mathbb{S} \mid \texttt{ast}(\bar{c}) = t_\bot\}$
k-Callsite-Sensitivity	$\delta^{\texttt{js-}k\texttt{-cfa}}([t_1, \cdots, t_n]) = \{\sigma = (\_, \_, \bar{c}, \_) \in \mathbb{S} \mid n \leq k \wedge (n = k \vee \texttt{js-ctxt}^{n+1}(\bar{c}) = \bot) \wedge \forall 1 \leq i \leq n.\ \texttt{ast} \circ \texttt{js-ctxt}^{i}(\bar{c}) = t_i\}$

JSAVER

(**J**avaScript **S**tatic **A**nalyzer **v**ia **E**CMAScript **R**epresentation)



JSAVER - Soundness



(a) Analysis results of TAJS

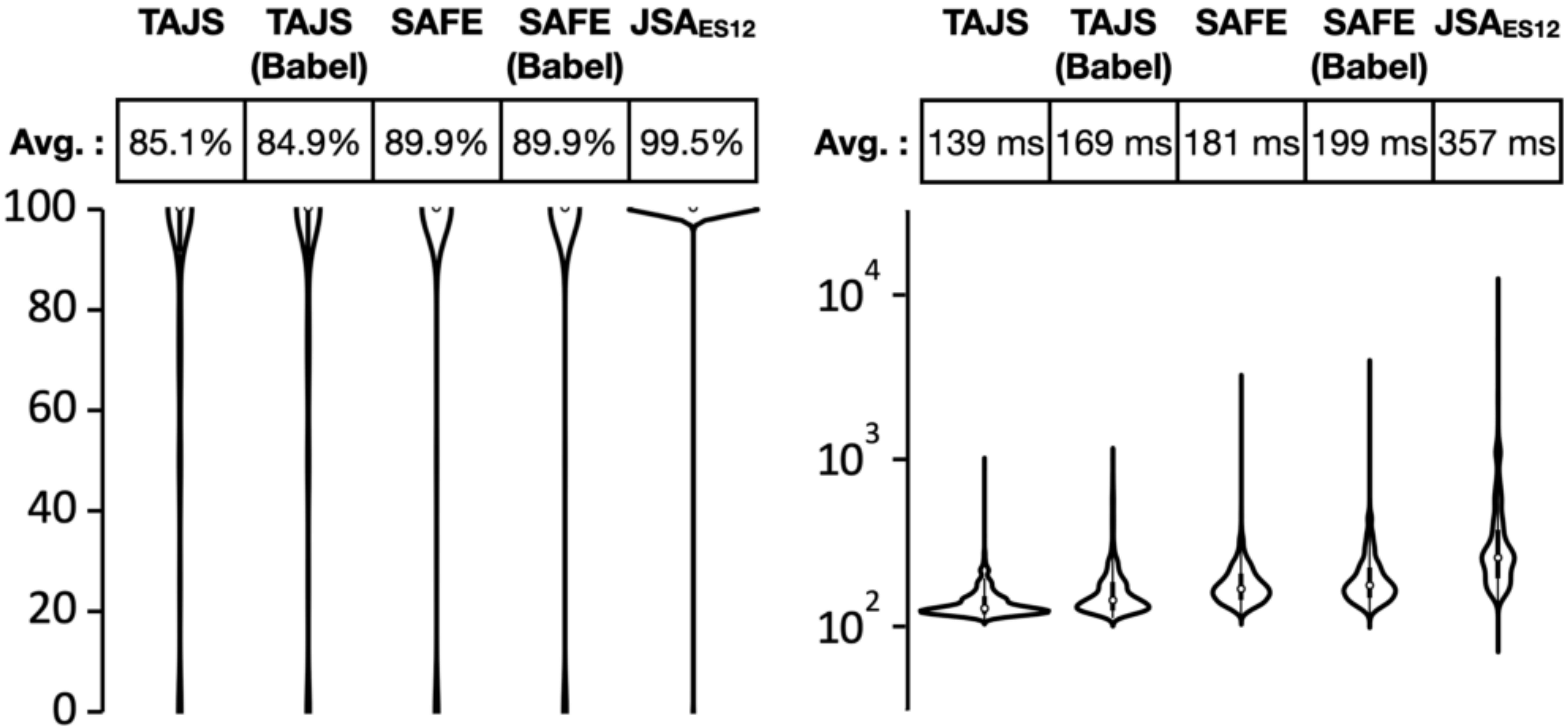
(b) Analysis results of SAFE

(c) Analysis results of JSA$_{\text{ES12}}$

(d) Analysis results of TAJS with Babel

(e) Analysis results of SAFE with Babel

JSAVER - Precision vs Performance

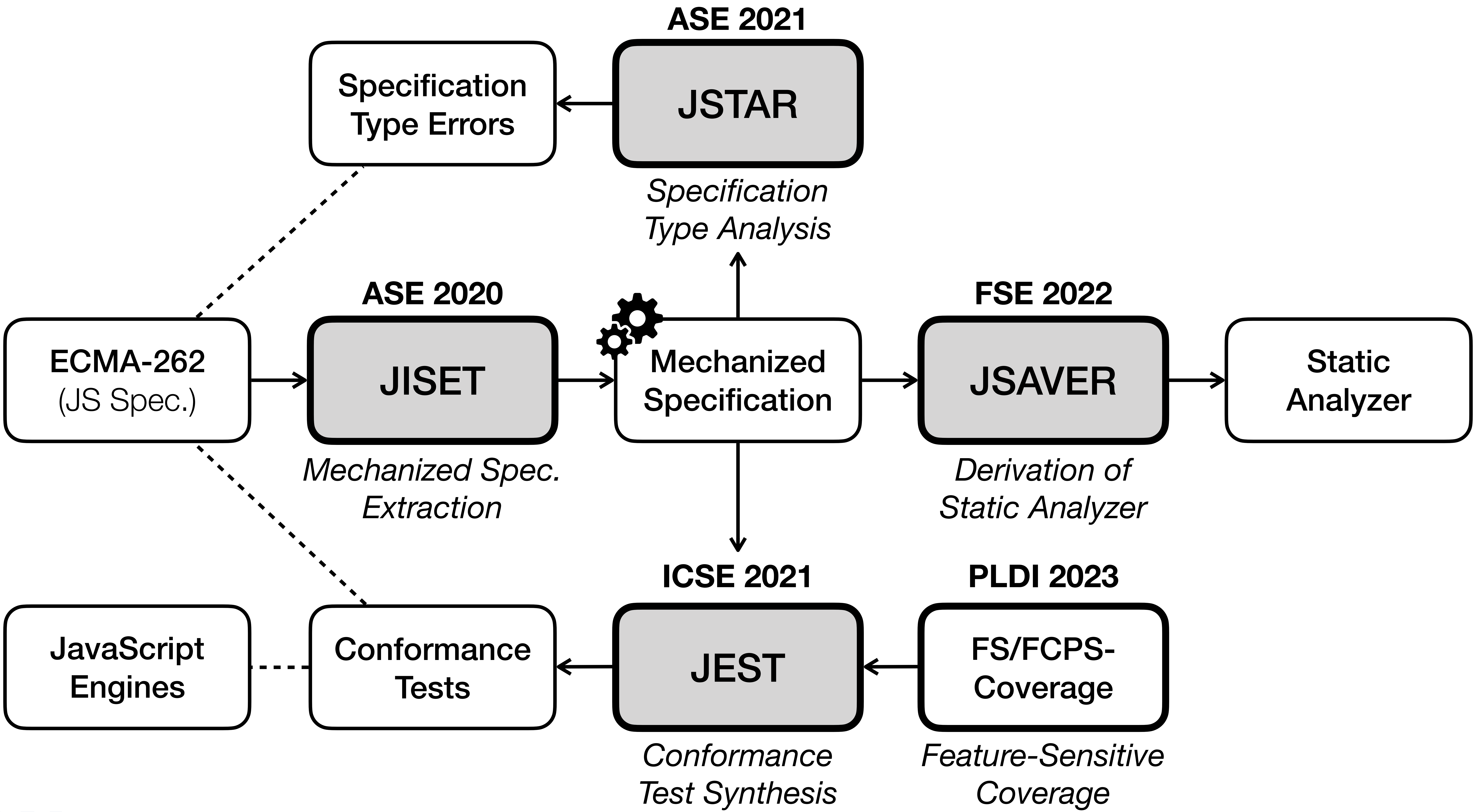


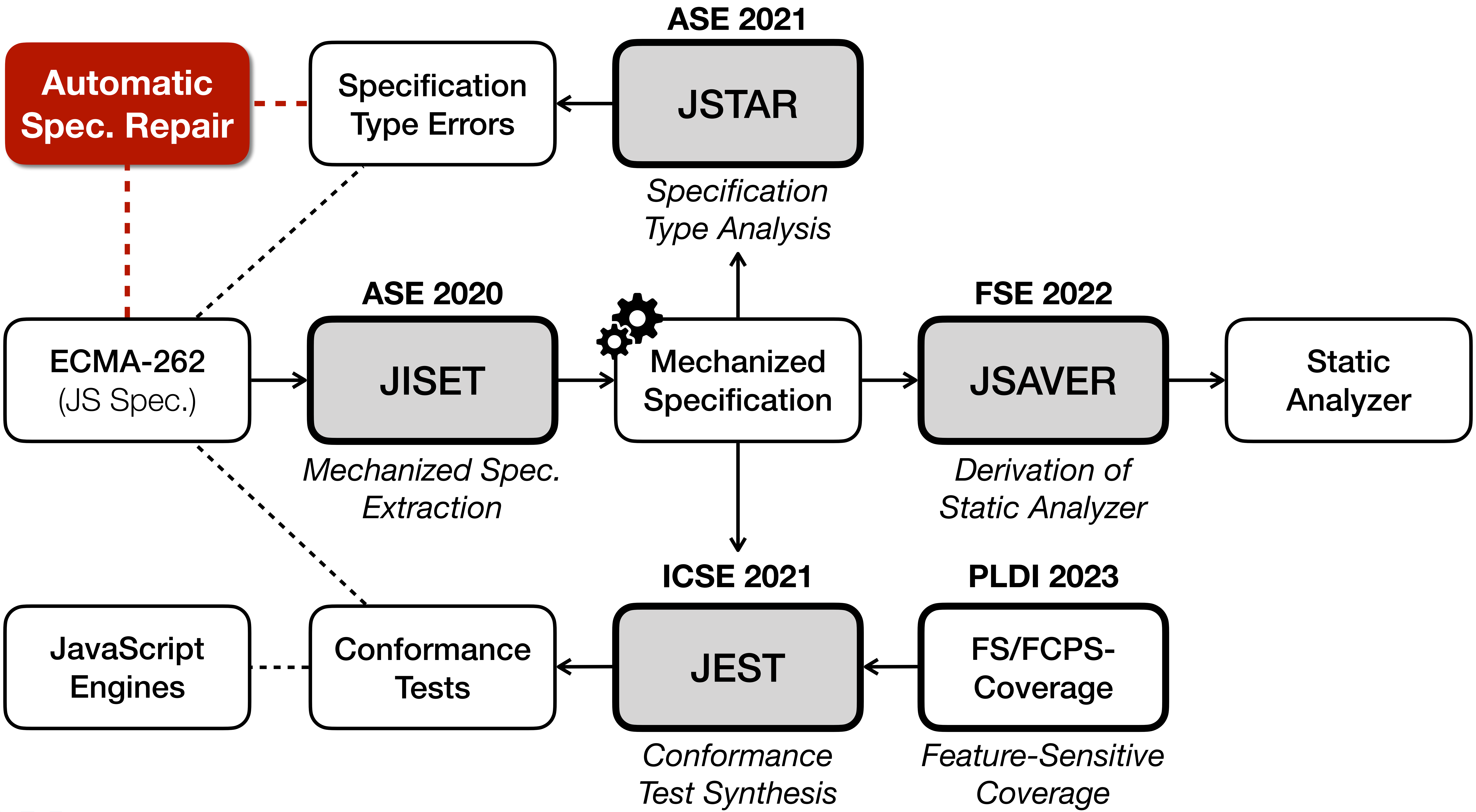
	TAJS	TAJS (Babel)	SAFE	SAFE (Babel)	JSA$_{ES12}$
Avg. :	85.1%	84.9%	89.9%	89.9%	99.5%

	TAJS	TAJS (Babel)	SAFE	SAFE (Babel)	JSA$_{ES12}$
Avg. :	139 ms	169 ms	181 ms	199 ms	357 ms

(a) The analysis precision

(b) The analysis performance


ASE 2021
JSTAR
Specification Type Analysis
Specification Type Errors
ASE 2020
JISET
Mechanized Spec. Extraction
ECMA-262
(JS Spec.)
Mechanized Specification
FSE 2022
JSAVER
Derivation of Static Analyzer
Static Analyzer
ICSE 2021
JEST
Conformance Test Synthesis
PLDI 2023
FS/FCPS-Coverage
Feature-Sensitive Coverage
JavaScript Engines
Conformance Tests



ASE 2021

Automatic Spec. Repair

Specification Type Errors

JSTAR

Specification Type Analysis

ASE 2020

FSE 2022

ECMA-262 (JS Spec.)

JISET

Mechanized Specification

JSAVER

Static Analyzer

Mechanized Spec. Extraction

Derivation of Static Analyzer

ICSE 2021

PLDI 2023

JavaScript Engines

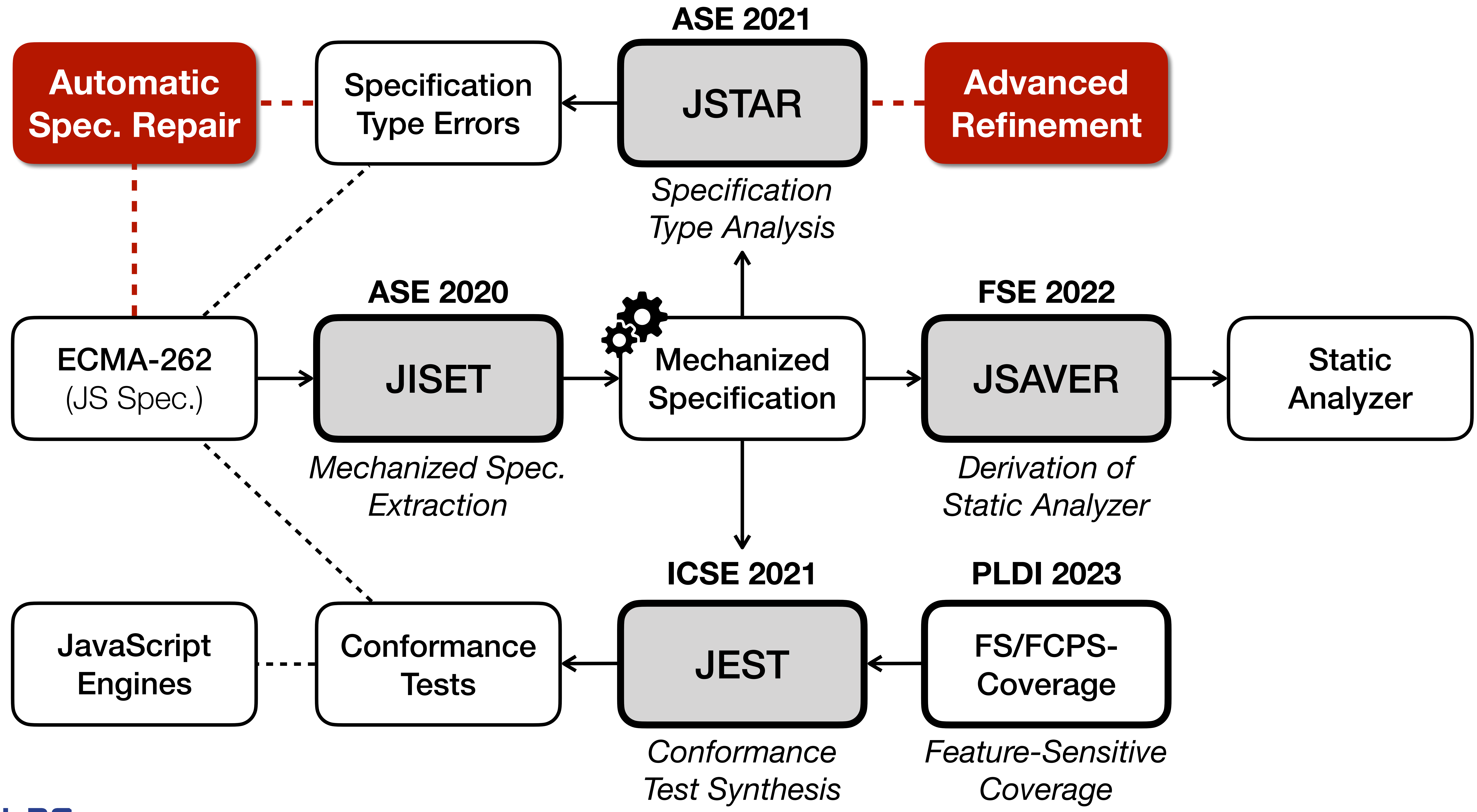
Conformance Tests

JEST

FS/FCPS-Coverage

Conformance Test Synthesis

Feature-Sensitive Coverage



ASE 2021

JSTAR

Specification Type Analysis

Automatic Spec. Repair

Specification Type Errors

Advanced Refinement

ASE 2020

JISET

Mechanized Spec. Extraction

ECMA-262 (JS Spec.)

Mechanized Specification

FSE 2022

JSAVER

Derivation of Static Analyzer

Static Analyzer

ICSE 2021

JEST

Conformance Test Synthesis

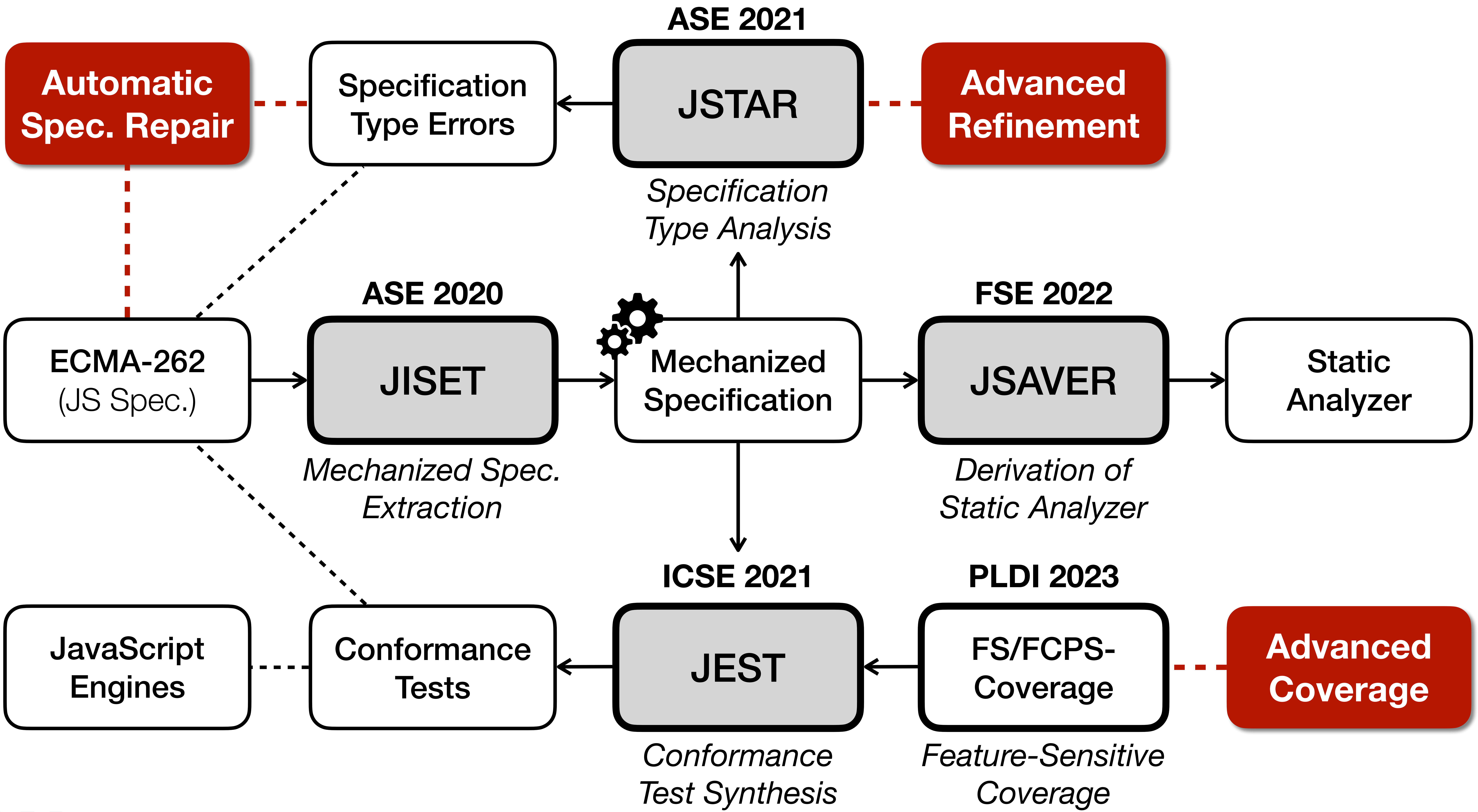
PLDI 2023

FS/FCPS-Coverage

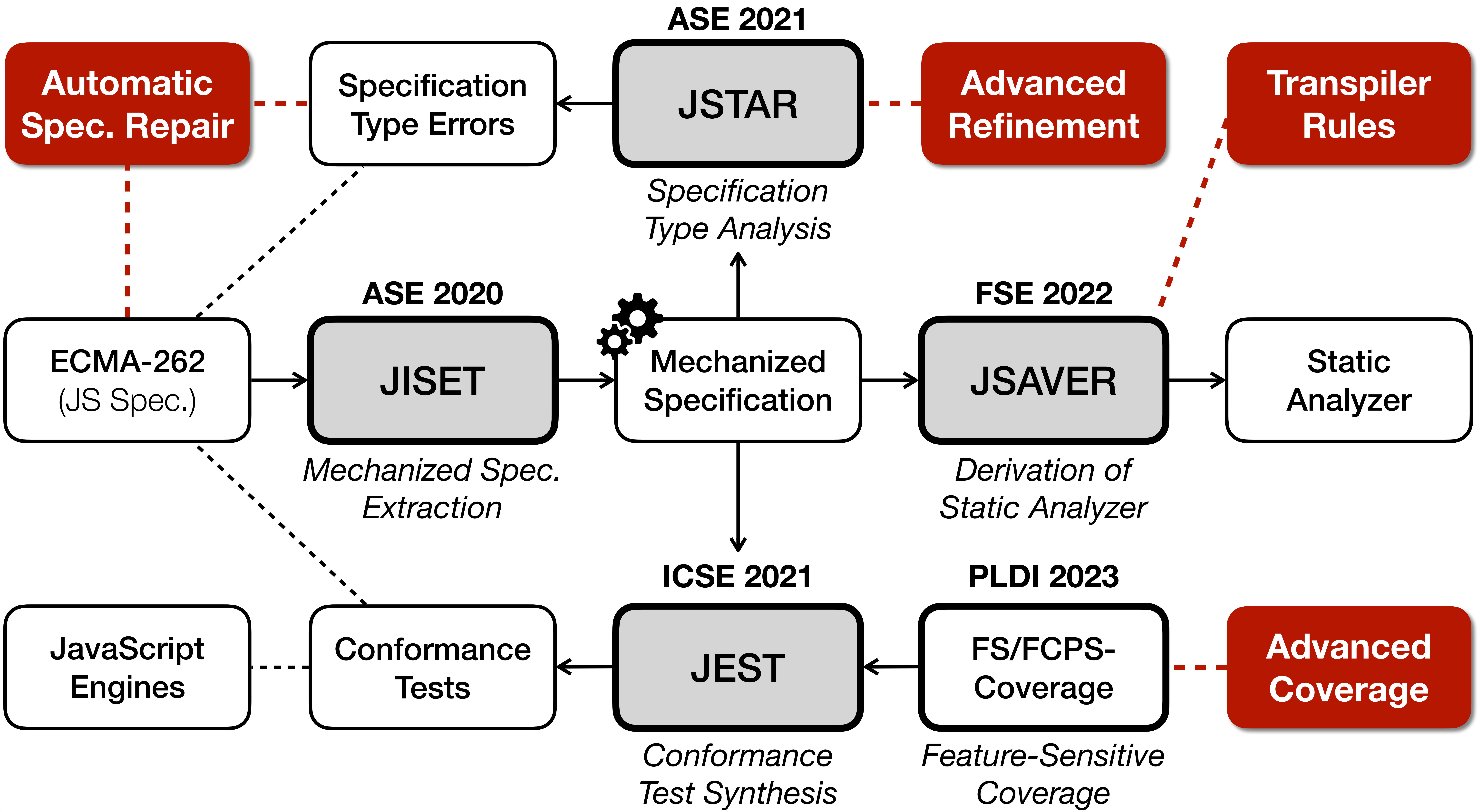
Feature-Sensitive Coverage

JavaScript Engines

Conformance Tests


ASE 2021
Automatic Spec. Repair
Specification Type Errors
JSTAR
Advanced Refinement
Specification Type Analysis
ASE 2020
FSE 2022
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
ICSE 2021
PLDI 2023
JavaScript Engines
Conformance Tests
JEST
FS/FCPS-Coverage
Advanced Coverage
Conformance Test Synthesis
Feature-Sensitive Coverage

PLRG



ASE 2021

JSTAR

Specification Type Analysis

Specification Type Errors

Automatic Spec. Repair

Advanced Refinement

Transpiler Rules

ASE 2020

JISET

Mechanized Spec. Extraction

ECMA-262 (JS Spec.)

Mechanized Specification

FSE 2022

JSAVER

Derivation of Static Analyzer

Static Analyzer

ICSE 2021

JEST

Conformance Test Synthesis

PLDI 2023

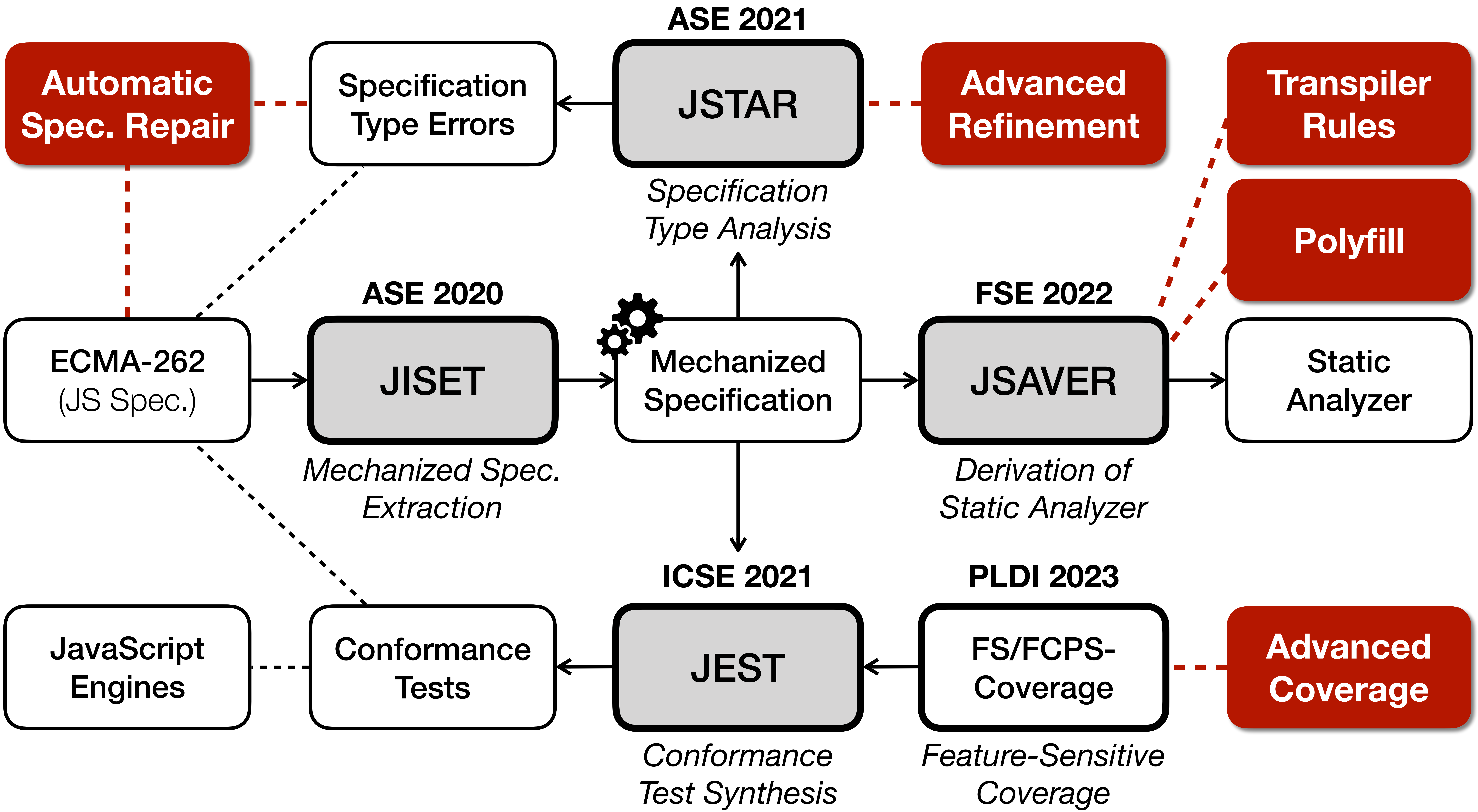
FS/FCPS-Coverage

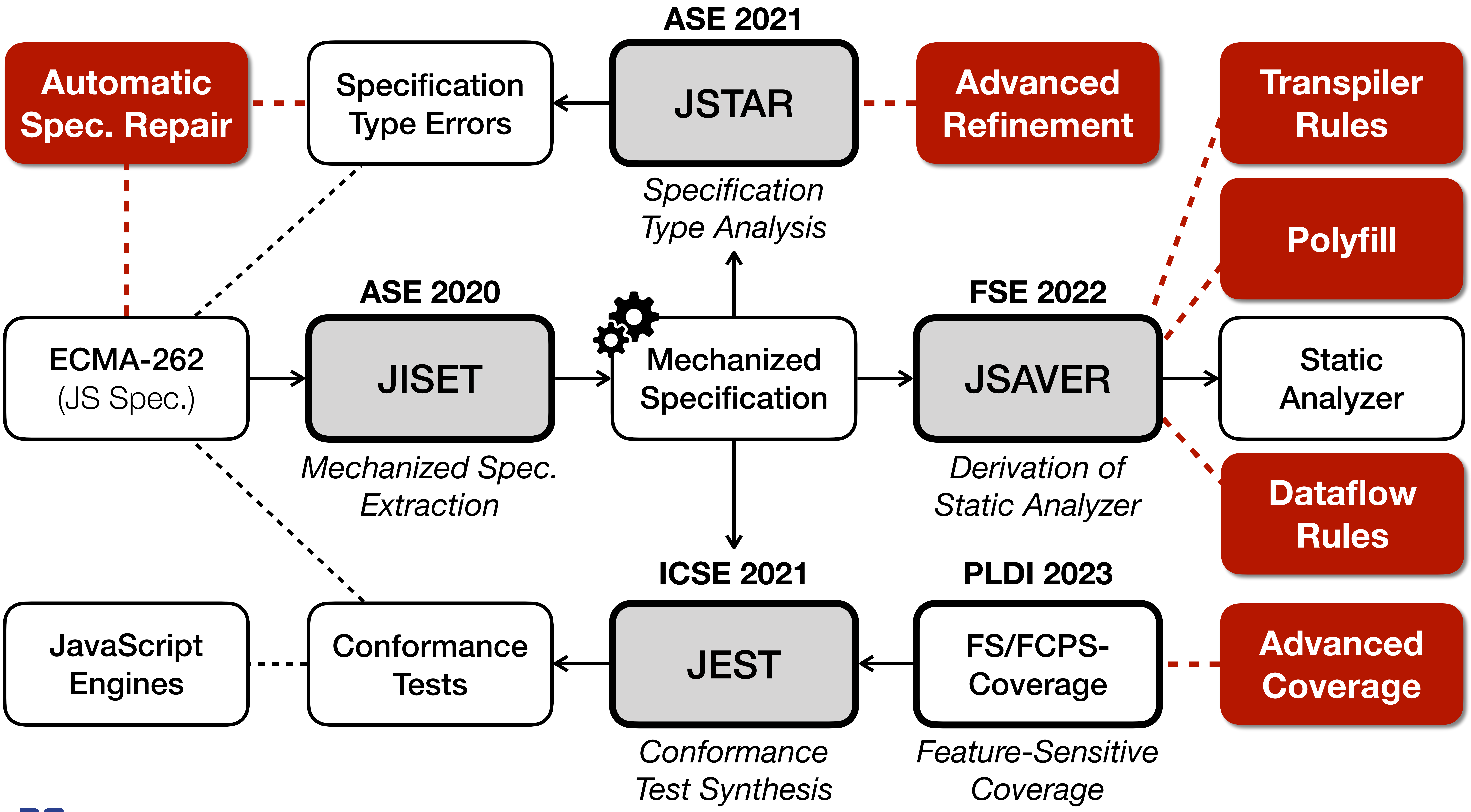
Feature-Sensitive Coverage

JavaScript Engines

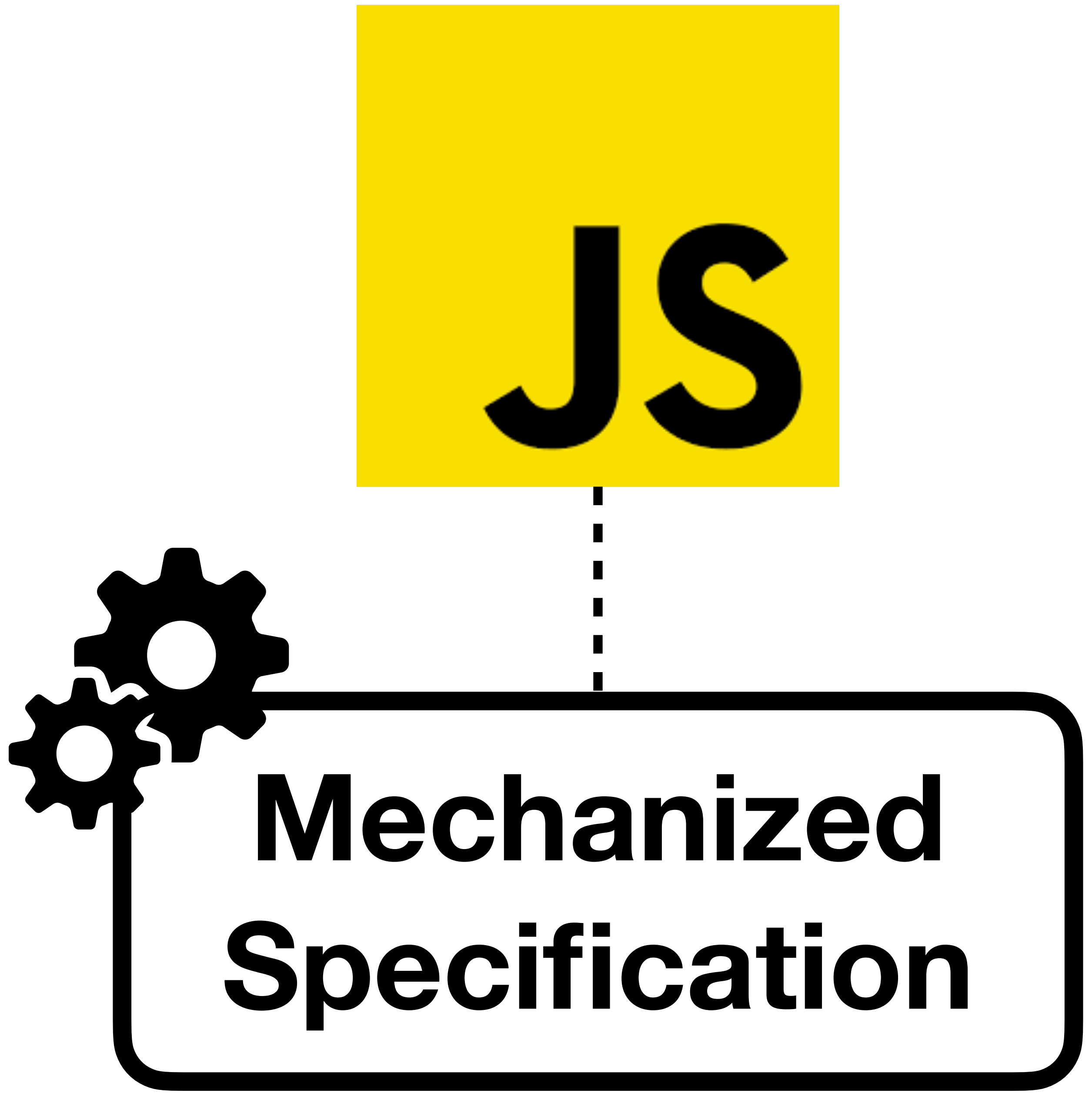
Conformance Tests

Advanced Coverage

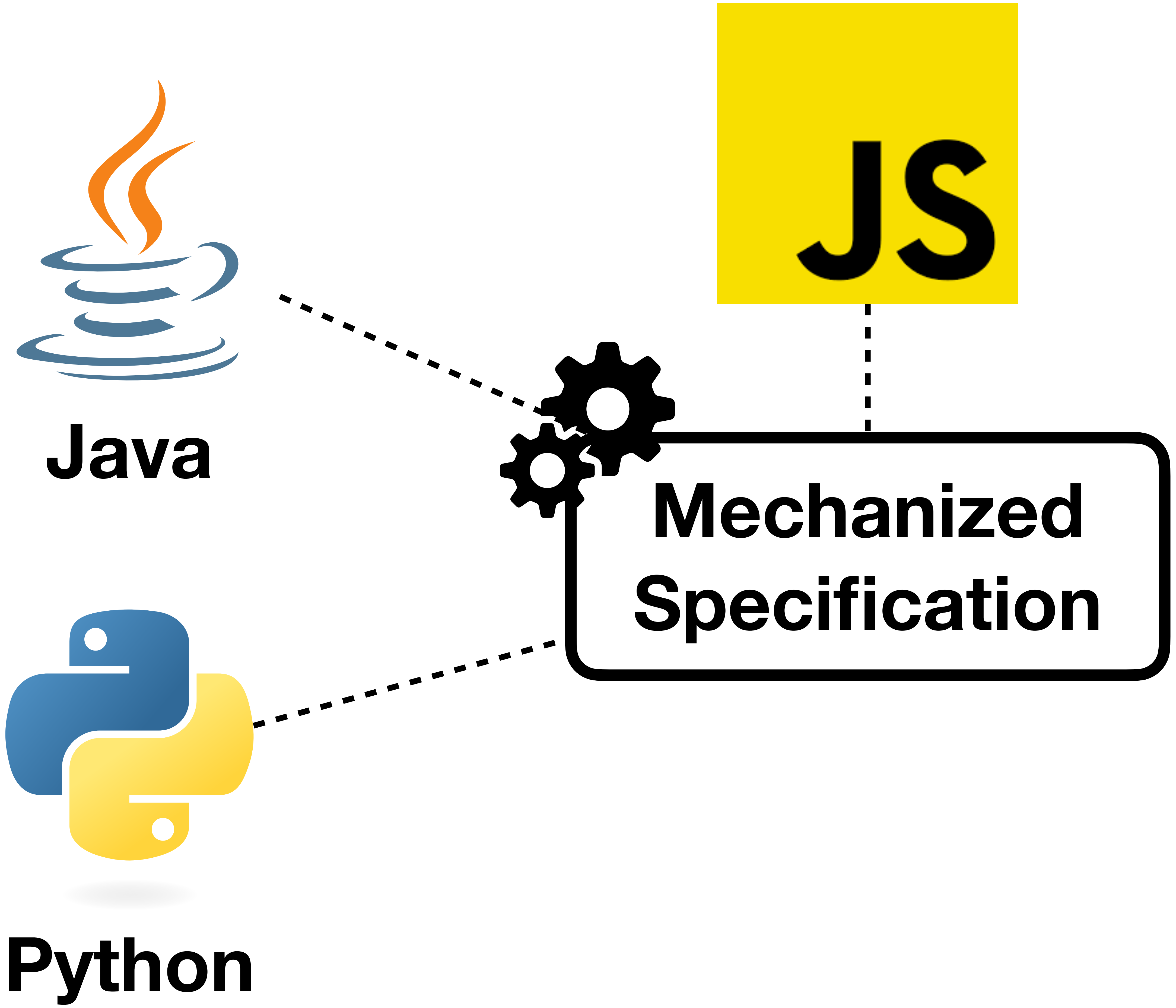

ASE 2021
Automatic Spec. Repair
Specification Type Errors
JSTAR
Advanced Refinement
Transpiler Rules
Specification Type Analysis
Polyfill
ASE 2020
FSE 2022
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
ICSE 2021
PLDI 2023
JavaScript Engines
Conformance Tests
JEST
FS/FCPS-Coverage
Advanced Coverage
Conformance Test Synthesis
Feature-Sensitive Coverage


ASE 2021
Automatic Spec. Repair
Specification Type Errors
JSTAR
Advanced Refinement
Transpiler Rules
Specification Type Analysis
Polyfill
ASE 2020
FSE 2022
ECMA-262 (JS Spec.)
JISET
Mechanized Specification
JSAVER
Static Analyzer
Mechanized Spec. Extraction
Derivation of Static Analyzer
Dataflow Rules
ICSE 2021
PLDI 2023
JavaScript Engines
Conformance Tests
JEST
FS/FCPS-Coverage
Advanced Coverage
Conformance Test Synthesis
Feature-Sensitive Coverage

PLRG



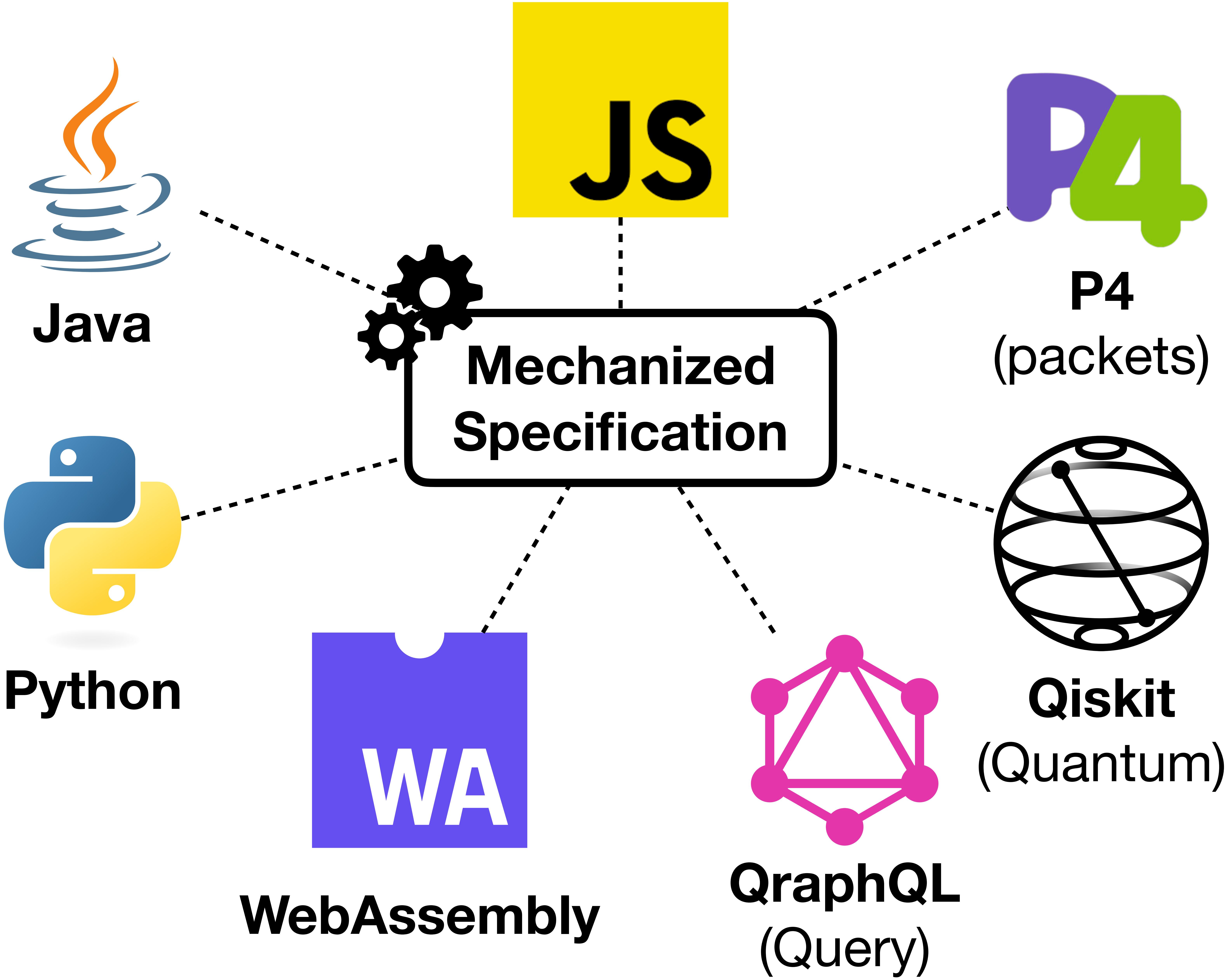
Mechanized Specification



Java

Mechanized Specification

Python



Mechanized Specification

Java

JS

P4
(packets)

Python

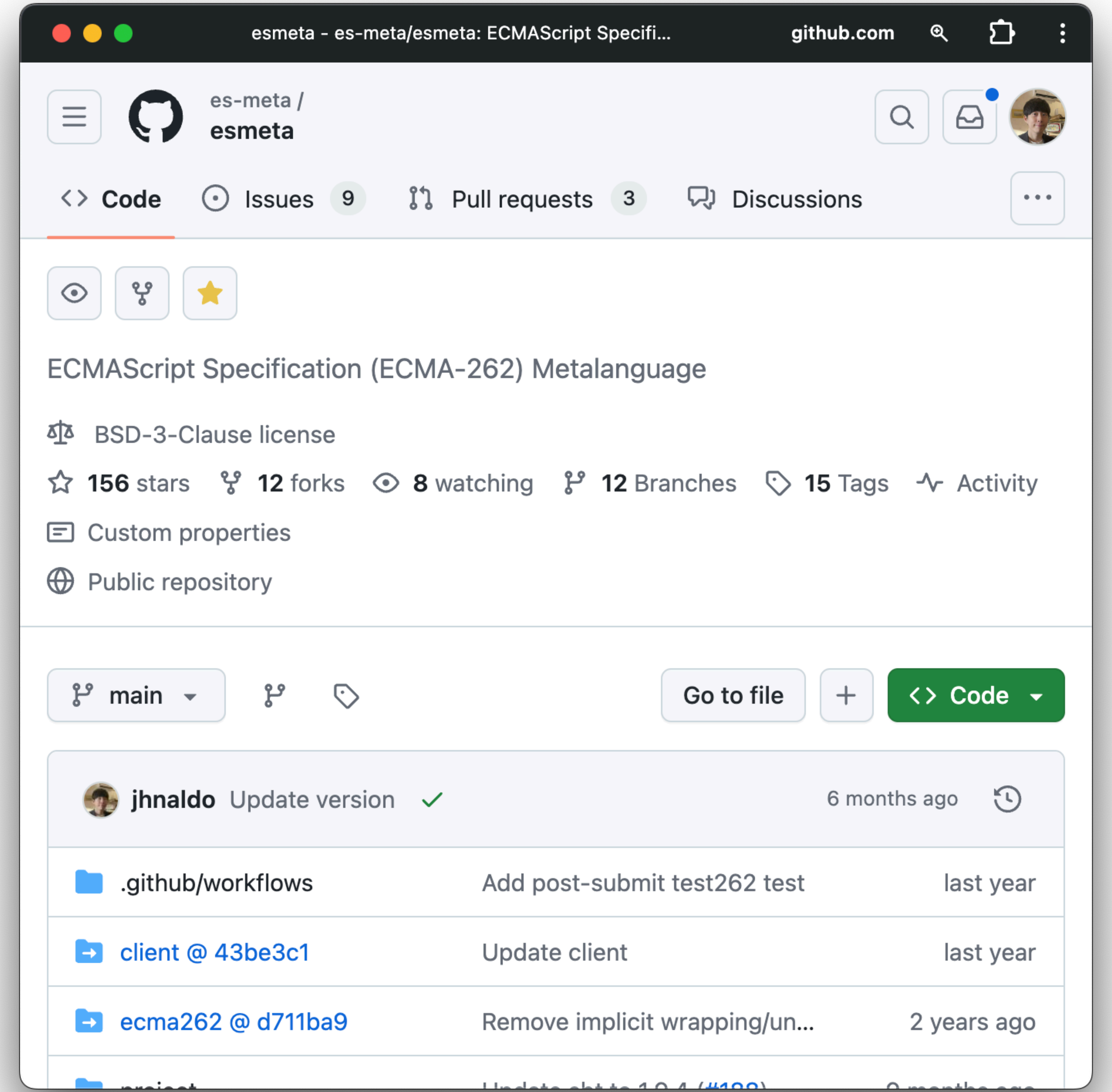
Qiskit
(Quantum)

WebAssembly

QraphQL
(Query)

ESMETA

https://github.com/es-meta/esmeta



ECMAScript Specification (ECMA-262) Metalanguage

BSD-3-Clause license

156 stars · 12 forks · 8 watching · 12 Branches · 15 Tags · Activity

Custom properties

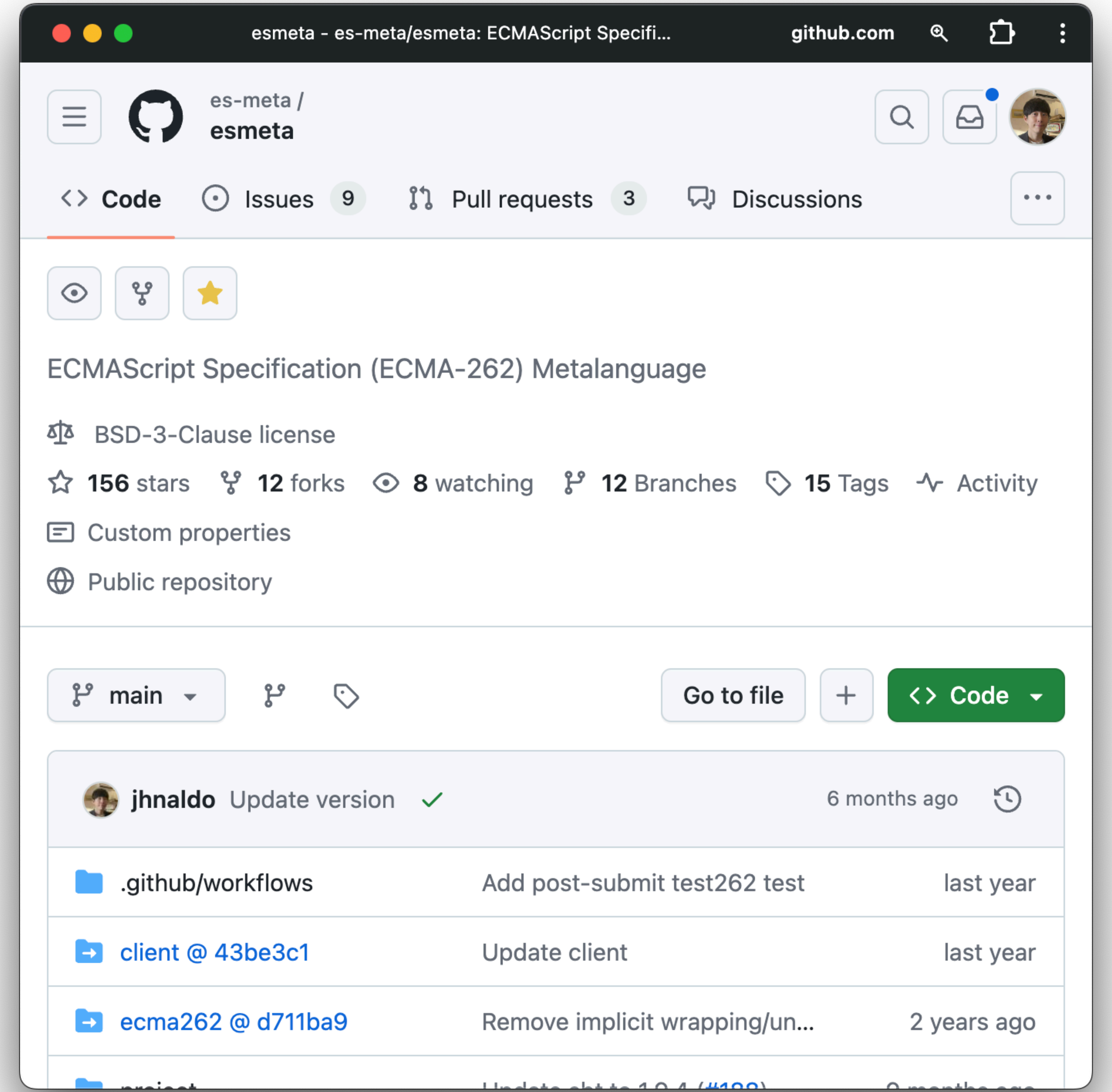
Public repository

jhnaldo Update version ✓ 6 months ago

.github/workflows	Add post-submit test262 test	last year
client @ 43be3c1	Update client	last year
ecma262 @ d711ba9	Remove implicit wrapping/un...	2 years ago

Official tool used in CI system of ECMA-262 and Test262

https://github.com/es-meta/esmeta



ECMAScript Specification (ECMA-262) Metalanguage

BSD-3-Clause license

156 stars 12 forks 8 watching 12 Branches 15 Tags Activity

Custom properties

Public repository

jhnaldo Update version ✓ 6 months ago

.github/workflows	Add post-submit test262 test	last year
client @ 43be3c1	Update client	last year
ecma262 @ d711ba9	Remove implicit wrapping/un...	2 years ago
project	Update sbt to 1.9.4 (#188)	9 months ago

PLRG