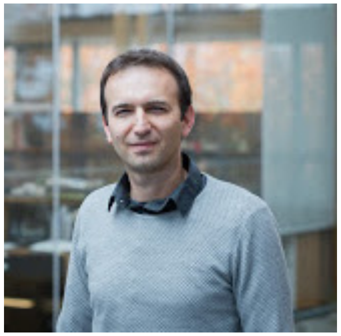
Detecting Unknown Inconsistencies in Web Applications

Frolin Ocariza Jr.

Karthik Pattabiraman

Ali Mesbah



95% of all websites use JavaScript

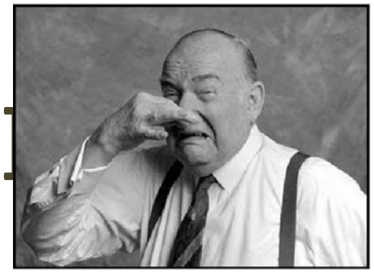
JavaScript

The most popular language on both GitHub and StackOverflow for 4 years

Bugs abound – our prior work
[ISSRE'11][ESEM'13][TSE]

JavaScript

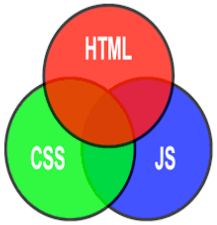
Code smells
[Fard and Mesbah – SCAM'2013]



JavaScript: Challenges

JS has loose semantics

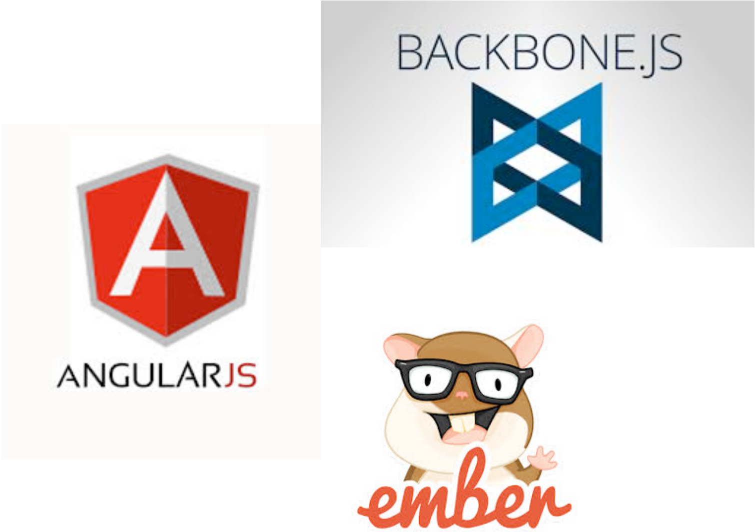
Lack of standard programming style

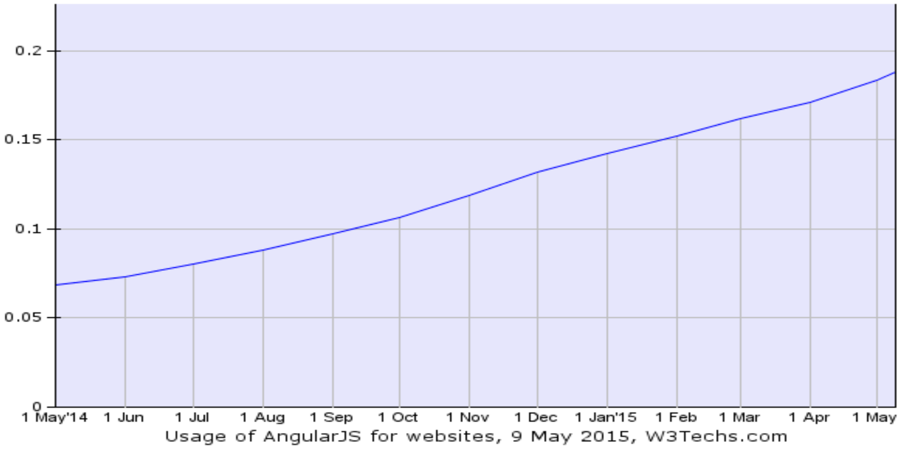


Frequent cross-language interactions

MVC Frameworks

- Model-View-Controller for structuring code
- Amenable to static analysis – less dynamism



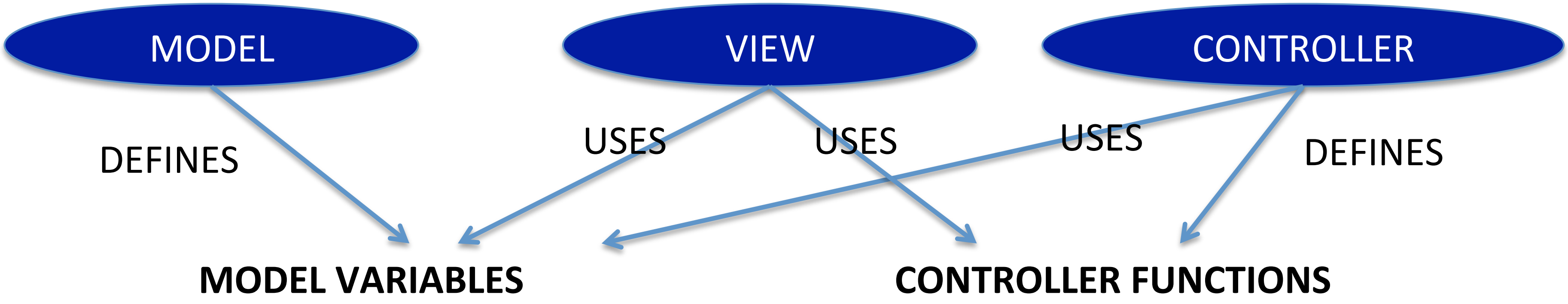


Usage of AngularJS for websites, 9 May 2015, W3Techs.com

300% increase in AngularJS usage in 2015

Our Earlier Work: Aurebesh [ICSE'15]

- **Aurebesh**: Detects mismatches between model, view, controller components in AngularJS code through **Static Analysis**



- **Aurebesh hard-codes rules for bug detection**
  - Name and type inconsistencies

Goal

- **Detect errors in JavaScript-based MVC applications through static analysis**
  - Without hard-coded rules
  - Without programmer annotations or hints
  - Also code smells or bad coding practices

- **Detect inconsistencies across languages**
  - Main difference with existing tools (e.g., Coverity)

Example Inconsistency

JavaScript Code

```
Marionette.LayoutView.extend({
   el: '.some-view',
   ...
});
```

HTML

```
<div class=“some-region”>
   <p>Hello World</p>
</div>
```

View in the JS code incorrectly assumes that the HTML contains an element with class ‘some-view’

Main Insights

How do we infer the consistency rules?

Leverage repeating code patterns

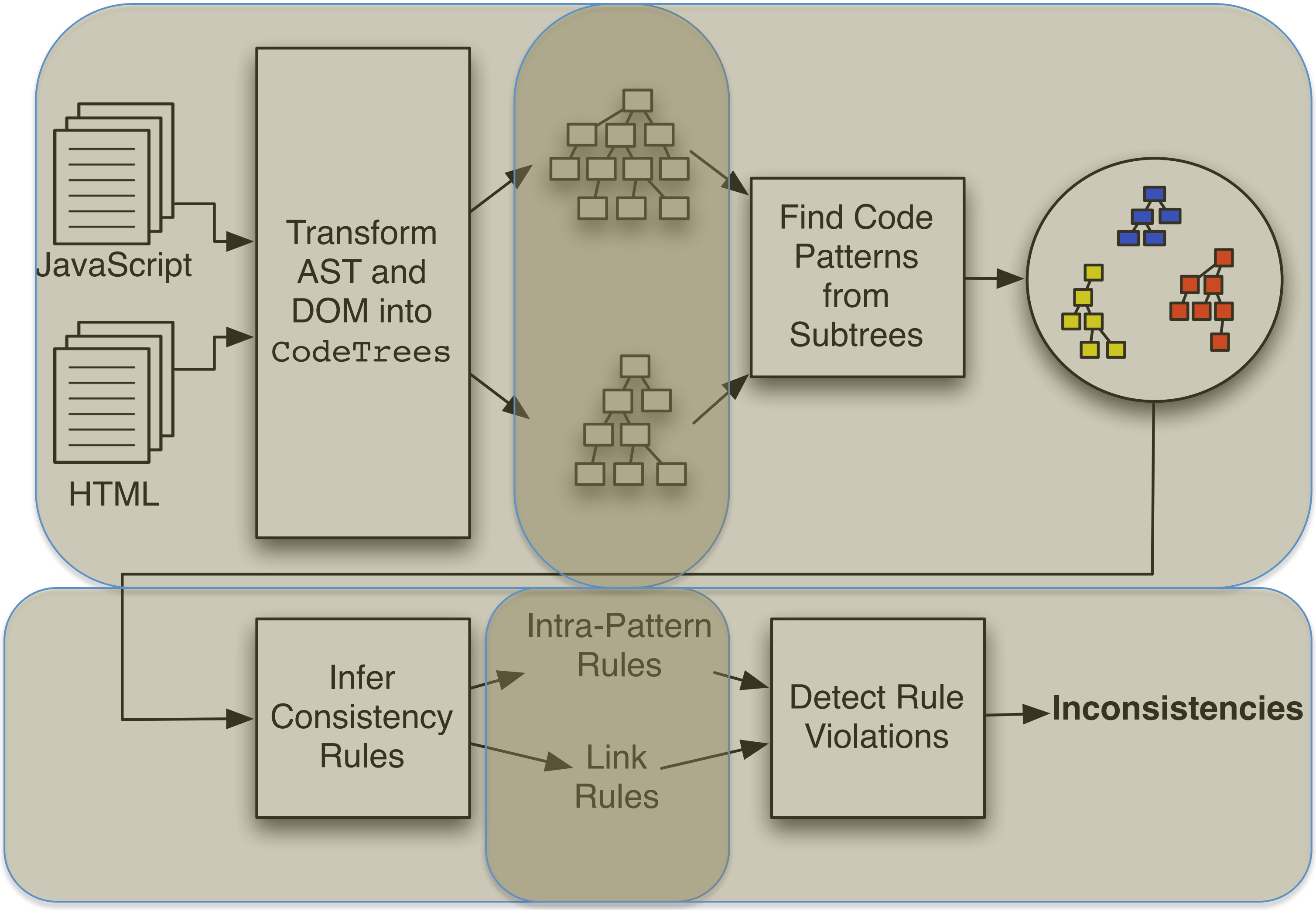
How do we detect cross-language inconsistencies?

LINK RULES

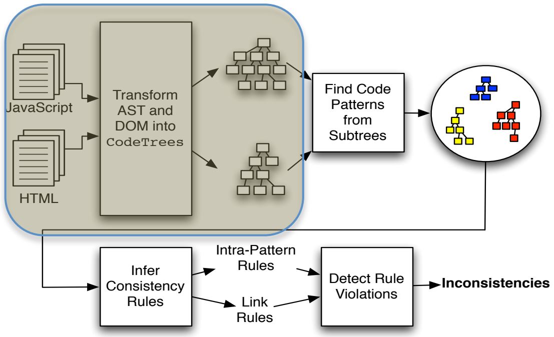
Outline

- Motivation and Goals
- Approach
- Evaluation
- Conclusion

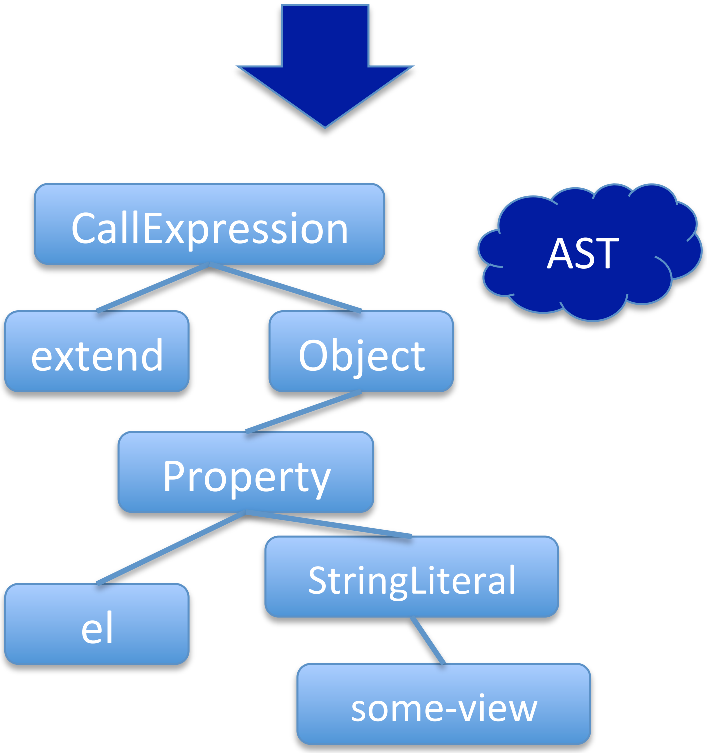
Our Approach



Step 1: Transform AST and DOM into CodeTrees



```
Marionette.LayoutView.extend({
     el: '.some-view',
     ...
});
```



AST

CallExpression

extend

Object

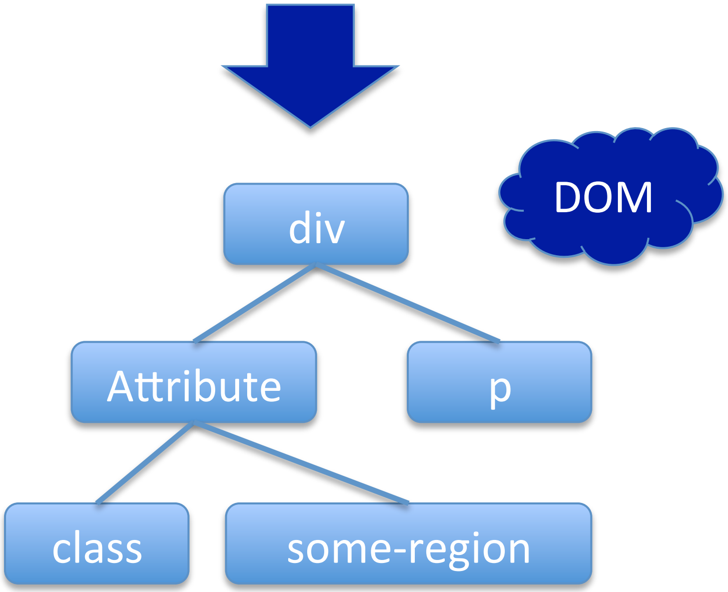
Property

el

StringLiteral

some-view

```
<div class=“some-region”>
    <p>Hello World</p>
</div>
```



DOM

div

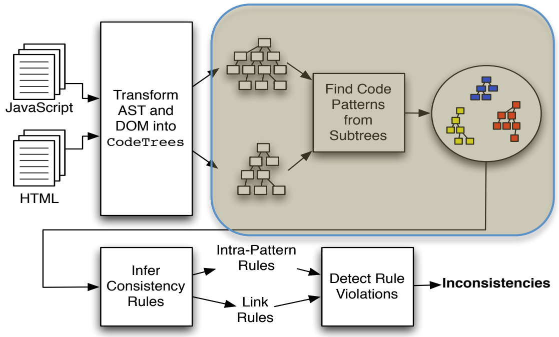
Attribute

p

class

some-region

Step 2: Find Code Patterns from Subtrees



JavaScript

HTML

Transform AST and DOM into CodeTrees

Find Code Patterns from Subtrees

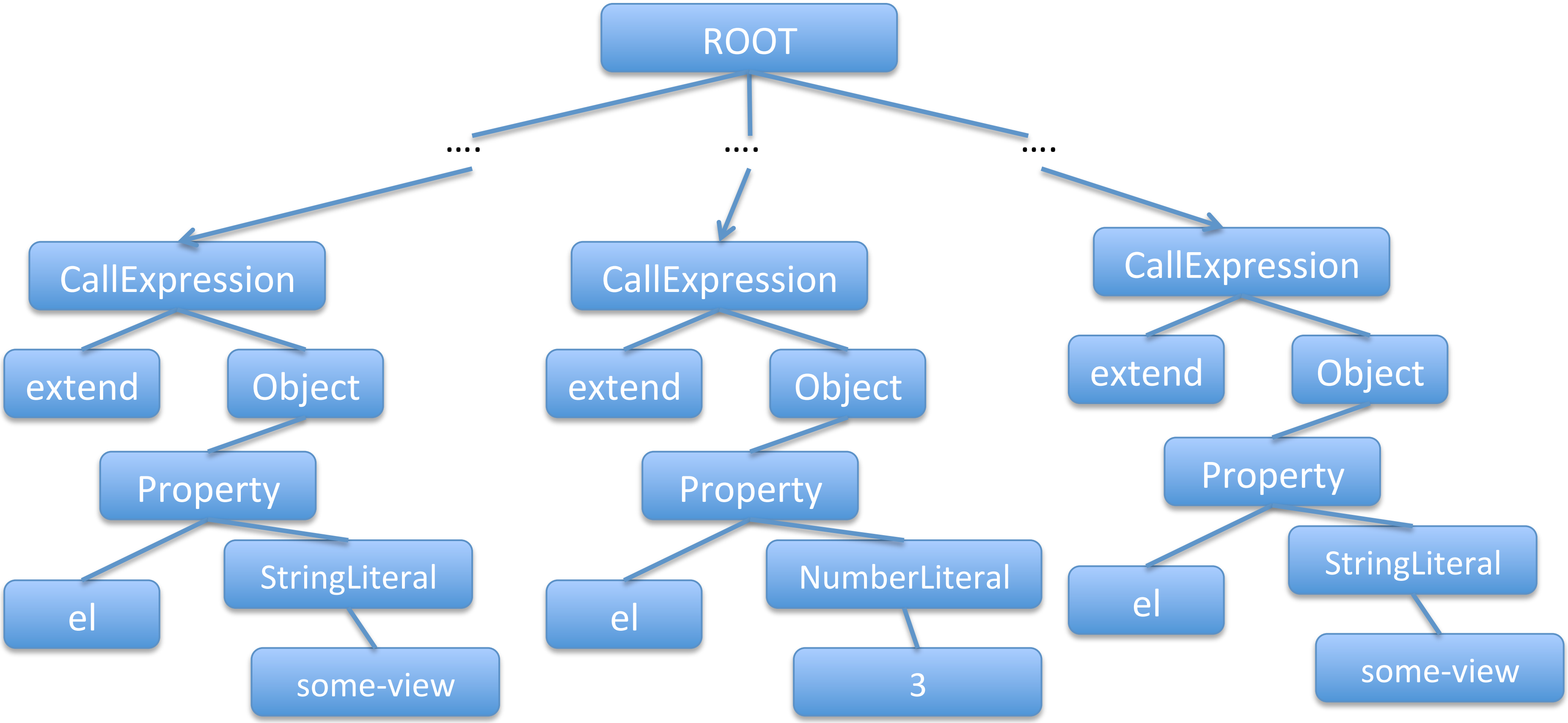
Infer Consistency Rules

Intra-Pattern Rules

Link Rules

Detect Rule Violations

Inconsistencies



ROOT

....

....

....

CallExpression

extend

Object

Property

el

StringLiteral

some-view

CallExpression

extend

Object

Property

el

NumberLiteral

3

CallExpression

extend

Object

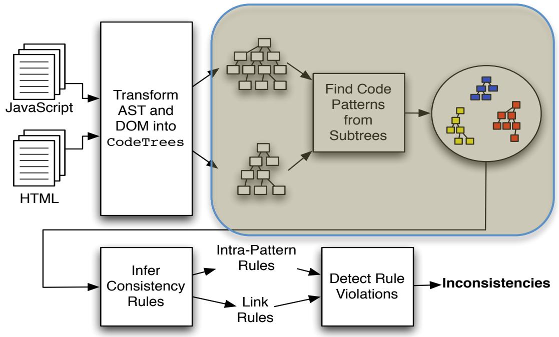
Property

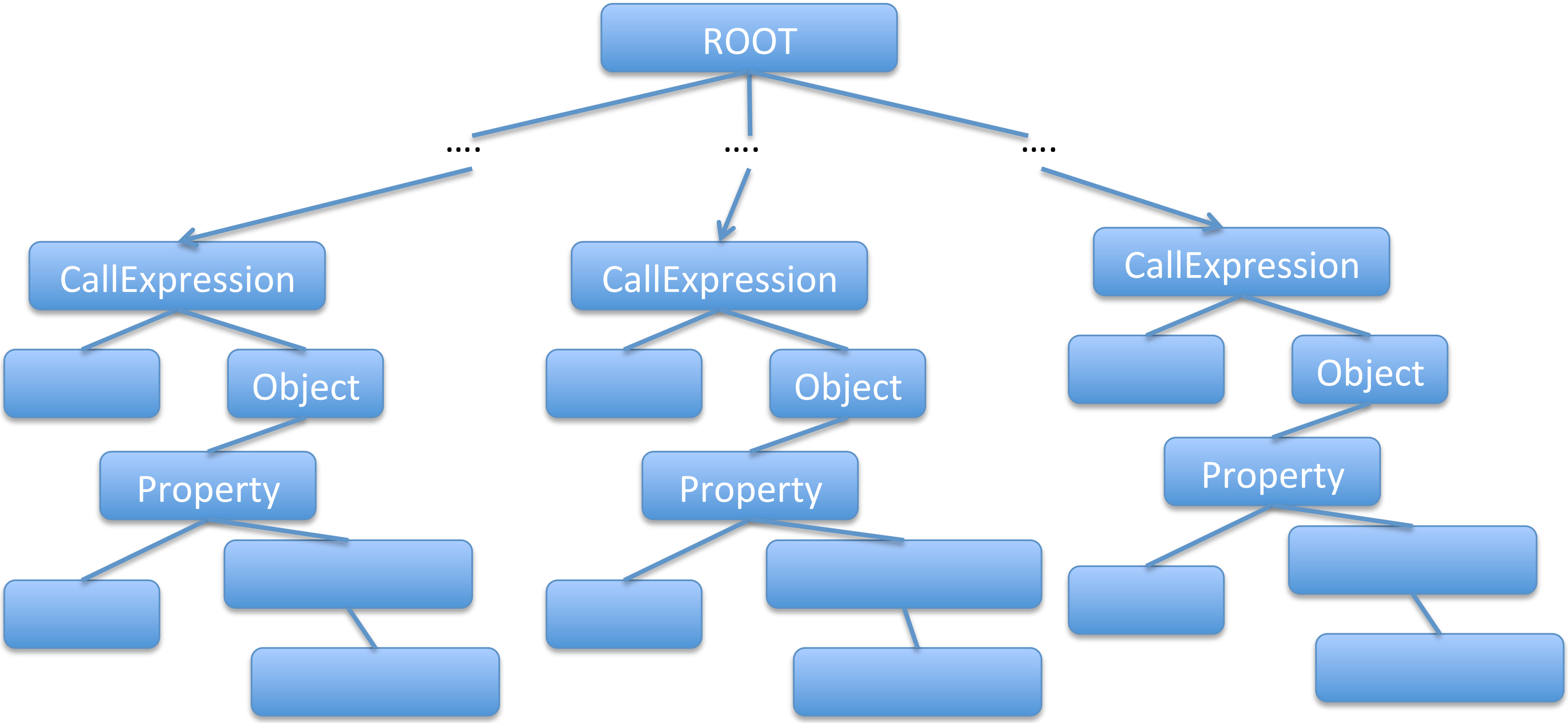
el

StringLiteral

some-view

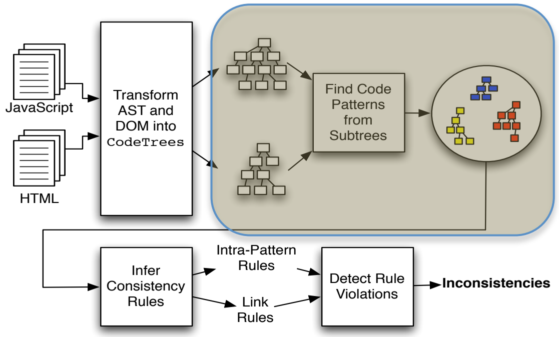
Step 2: Find Code Patterns from Subtrees

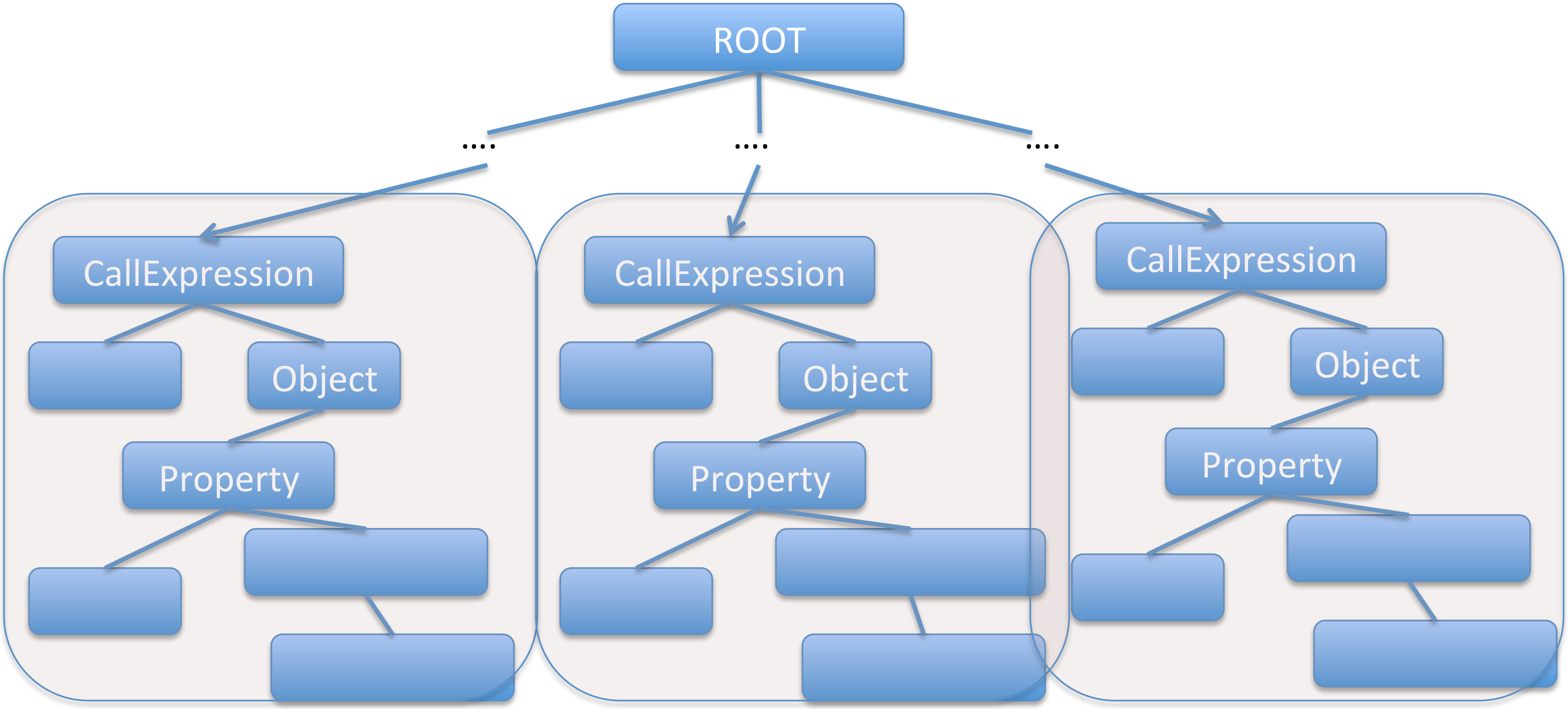




“Abstract out” identifiers, literals, and string types

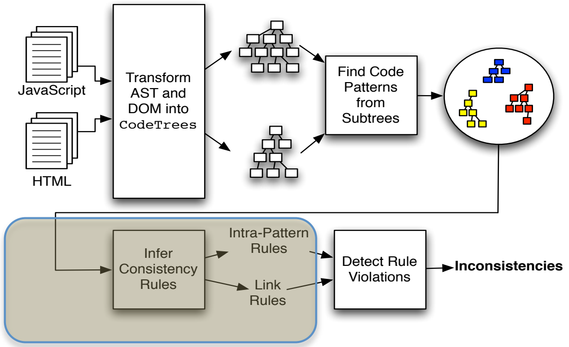
Step 2: Find Code Patterns from Subtrees





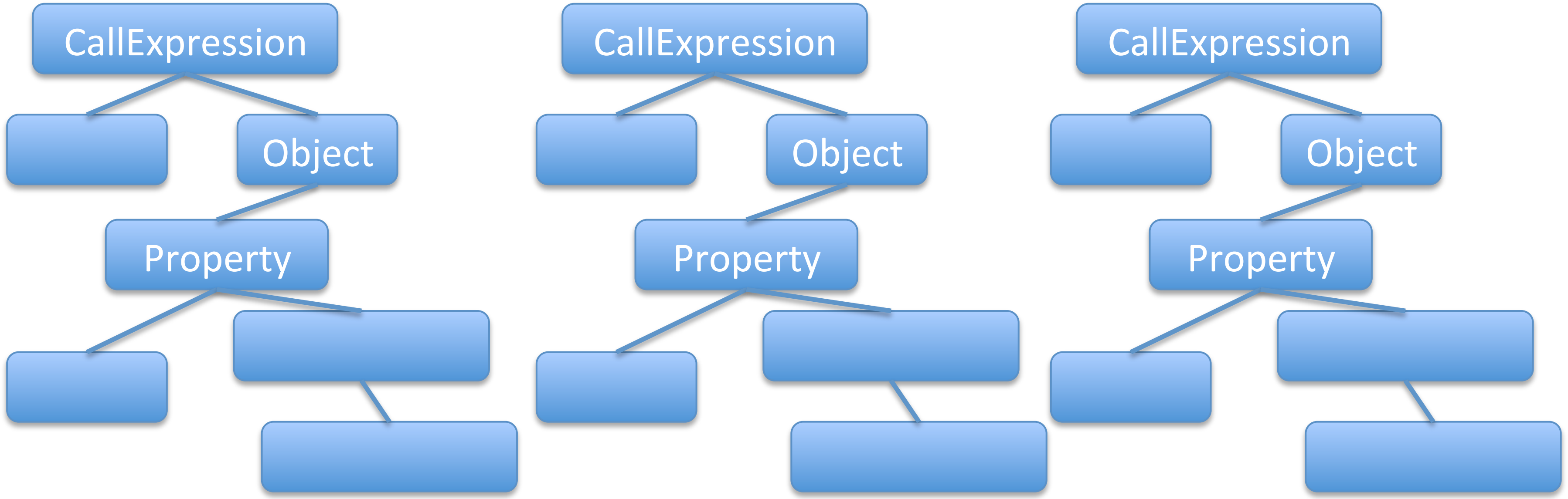
Group together isomorphic subtrees

Step 3: Infer Consistency Rules



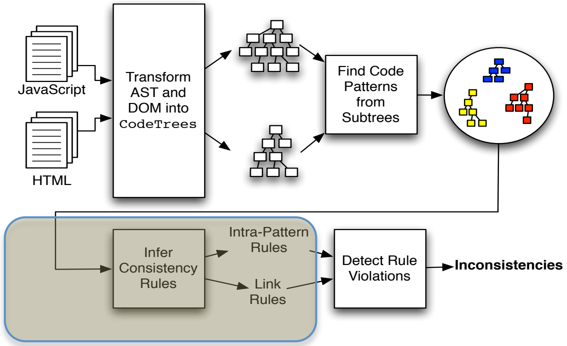
INTRA-PATTERN CONSISTENCY RULE

*Inconsistencies **within** pattern groups*



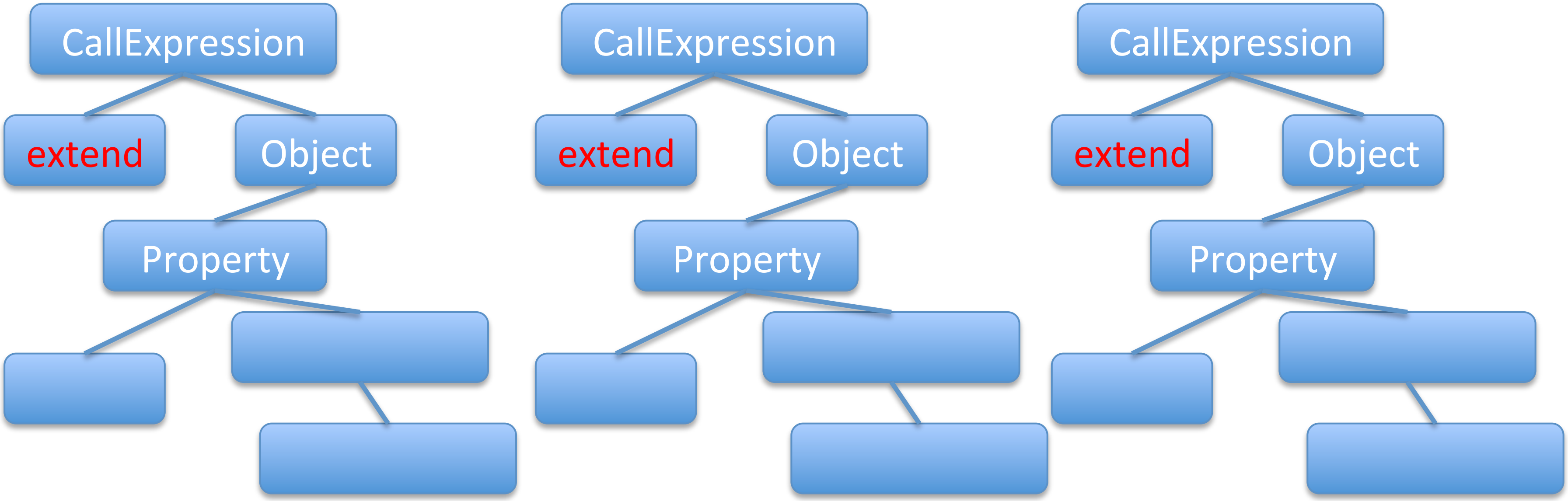
Nodes are "concretized" one by one

Step 3: Infer Consistency Rules



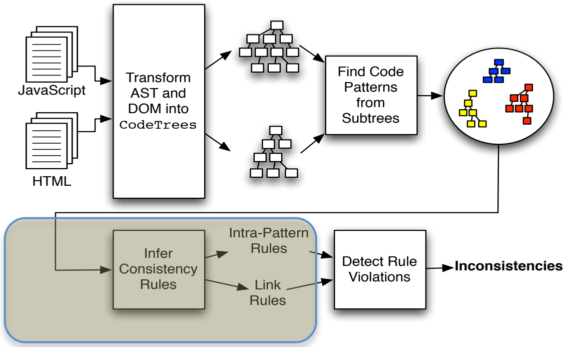
INTRA-PATTERN CONSISTENCY RULE

*Inconsistencies **within** pattern groups*



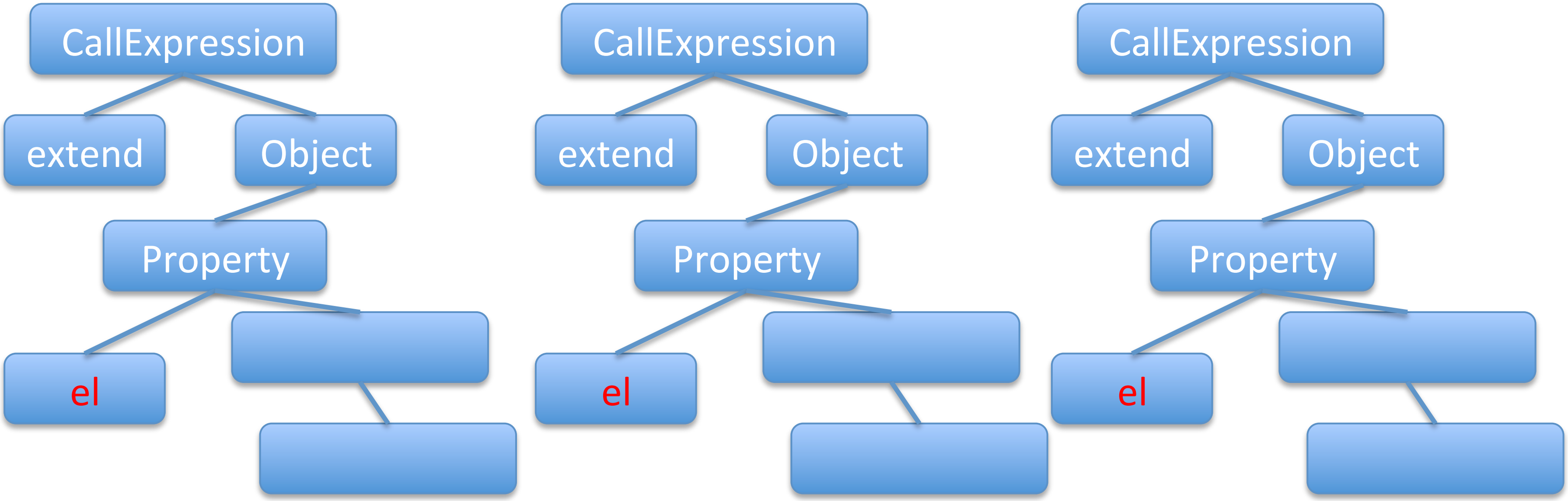
Nodes are “concretized” one by one

Step 3: Infer Consistency Rules



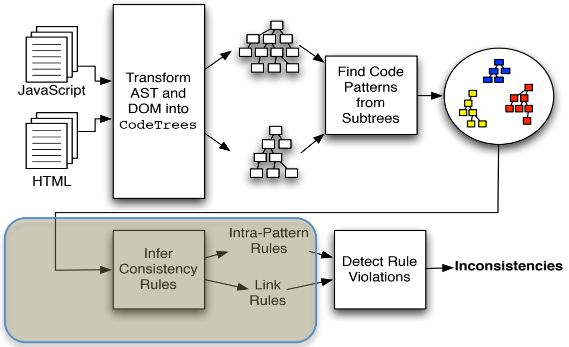
INTRA-PATTERN CONSISTENCY RULE

*Inconsistencies **within** pattern groups*



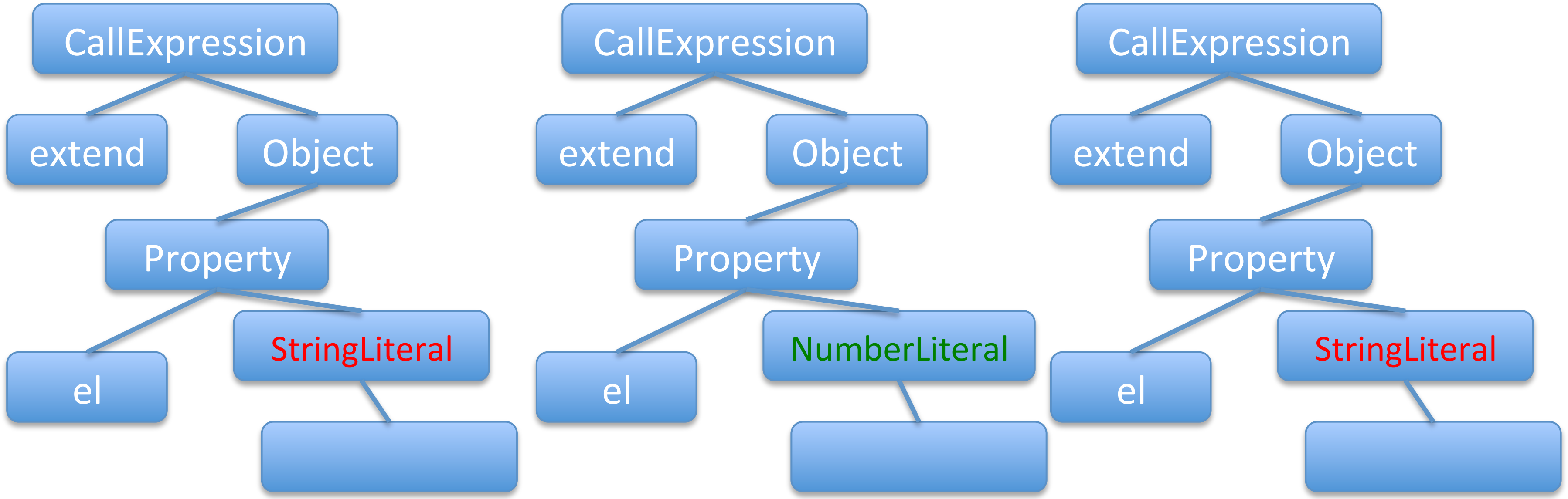
Nodes are "concretized" one by one

Step 3: Infer Consistency Rules



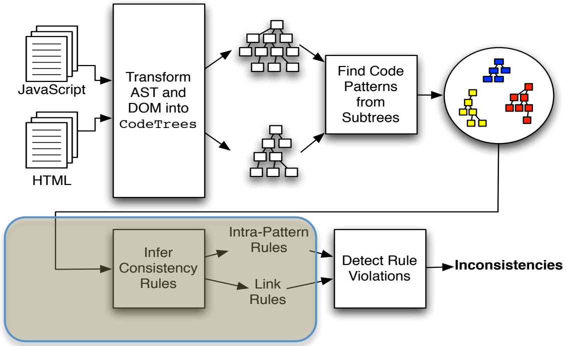
INTRA-PATTERN CONSISTENCY RULE

*Inconsistencies **within** pattern groups*



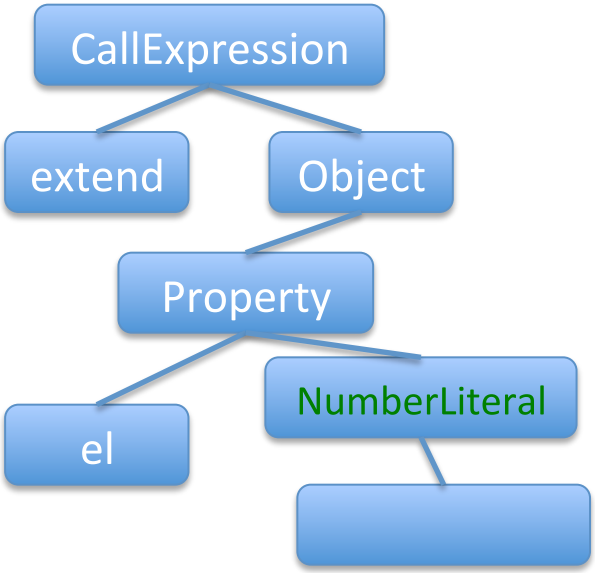
Subtrees are partitioned according to differences found while concretizing

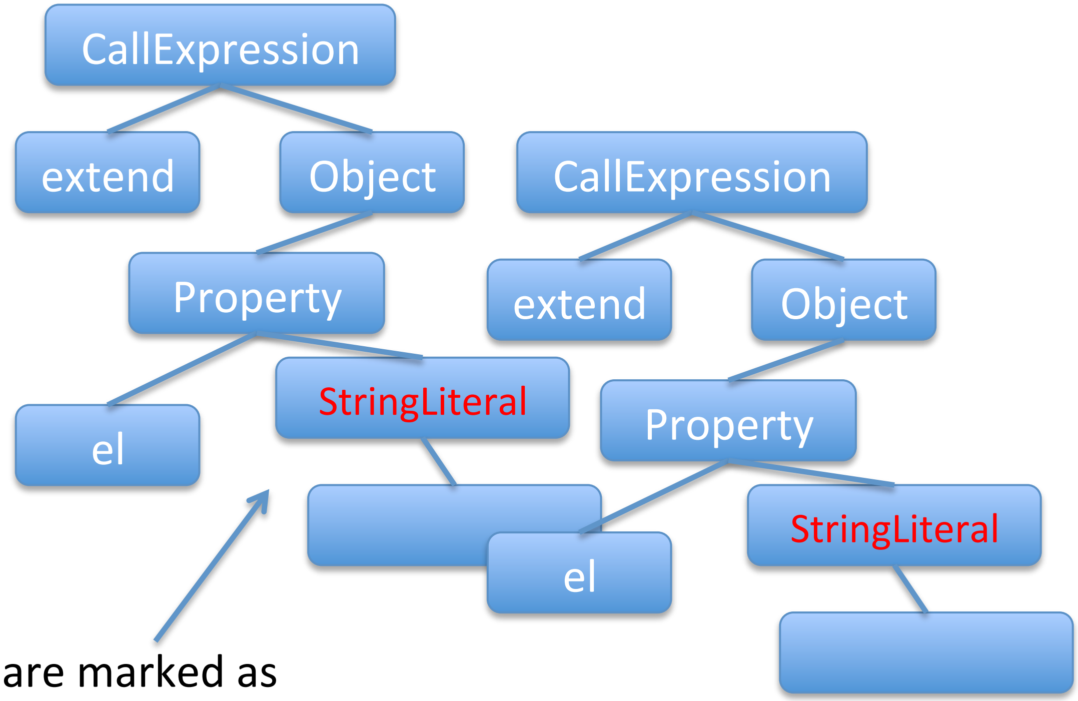
Step 3: Infer Consistency Rules



INTRA-PATTERN CONSISTENCY RULE

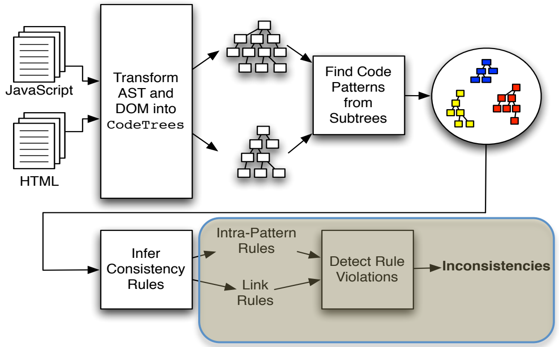
*Inconsistencies **within** pattern groups*





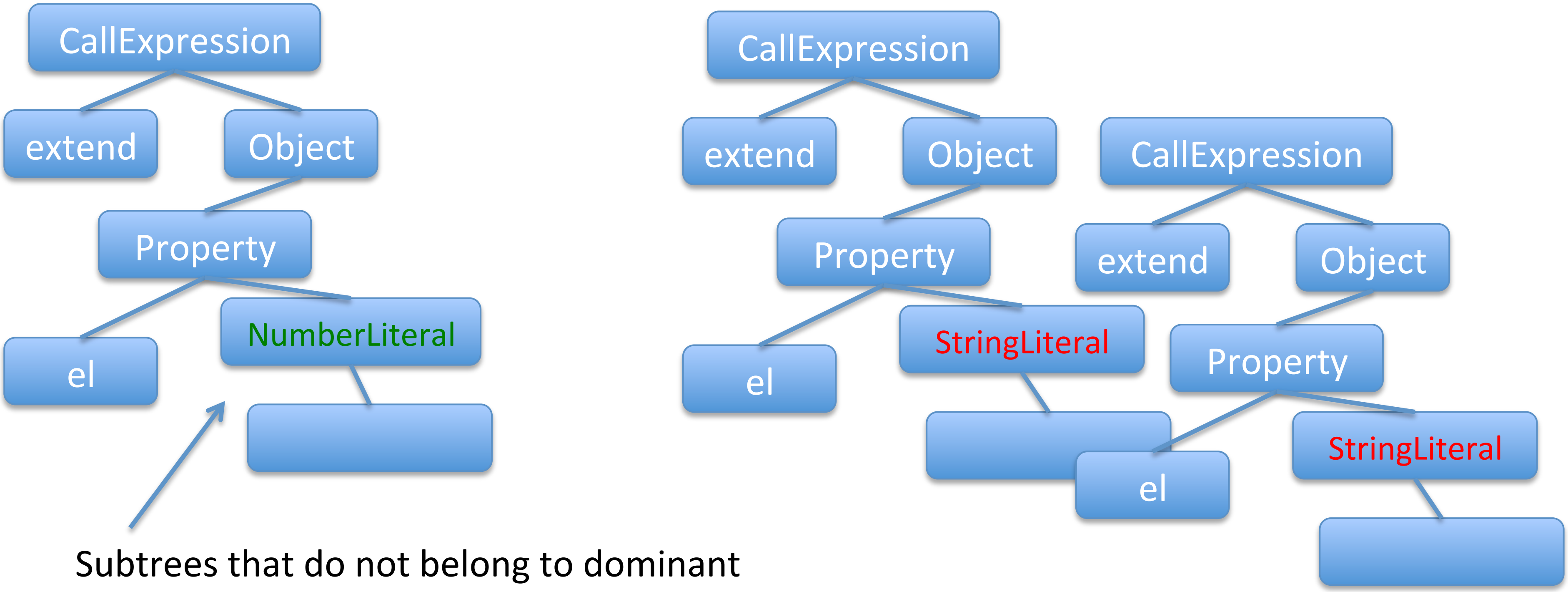
Dominant patterns are marked as intra-pattern consistency rules

Step 4: Detect Rule Violations



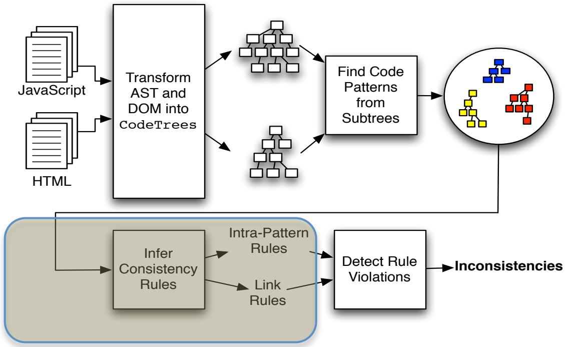
INTRA-PATTERN CONSISTENCY RULE

*Inconsistencies **within** pattern groups*



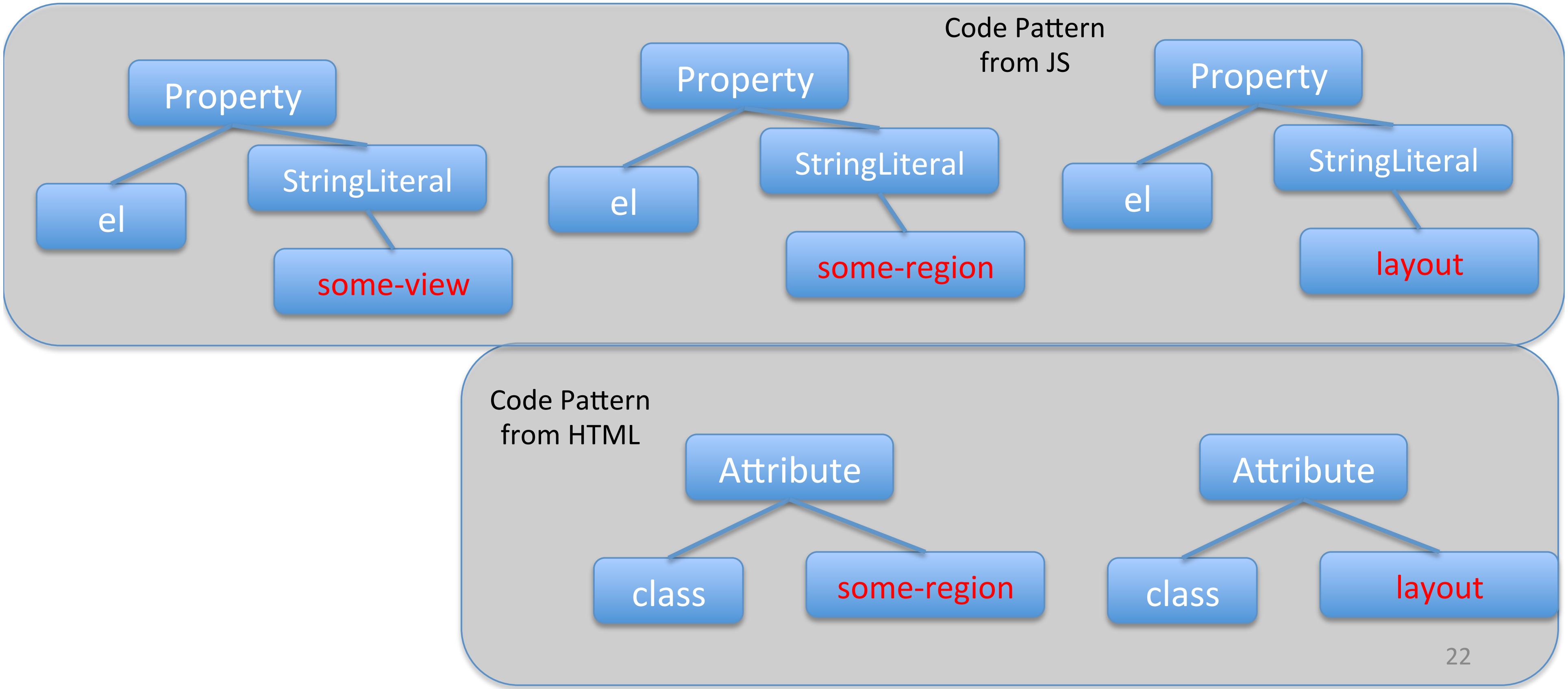
Subtrees that do not belong to dominant patterns are marked as inconsistencies

Step 3: Infer Consistency Rules


JavaScript
HTML
Transform AST and DOM into CodeTrees
Find Code Patterns from Subtrees
Infer Consistency Rules
Intra-Pattern Rules
Link Rules
Detect Rule Violations
Inconsistencies

LINK RULE

Inconsistencies ***between*** *pattern groups*


Code Pattern from JS

Property
el
StringLiteral
some-view

Property
el
StringLiteral
some-region

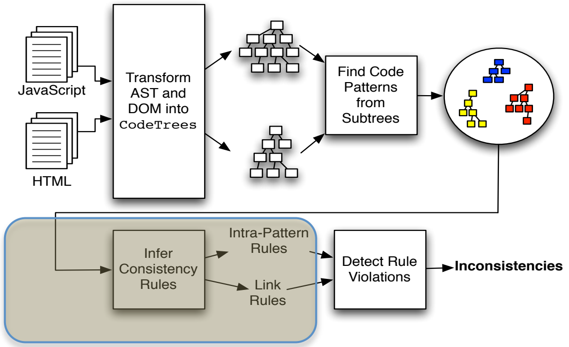
Property
el
StringLiteral
layout

Code Pattern from HTML

Attribute
class
some-region

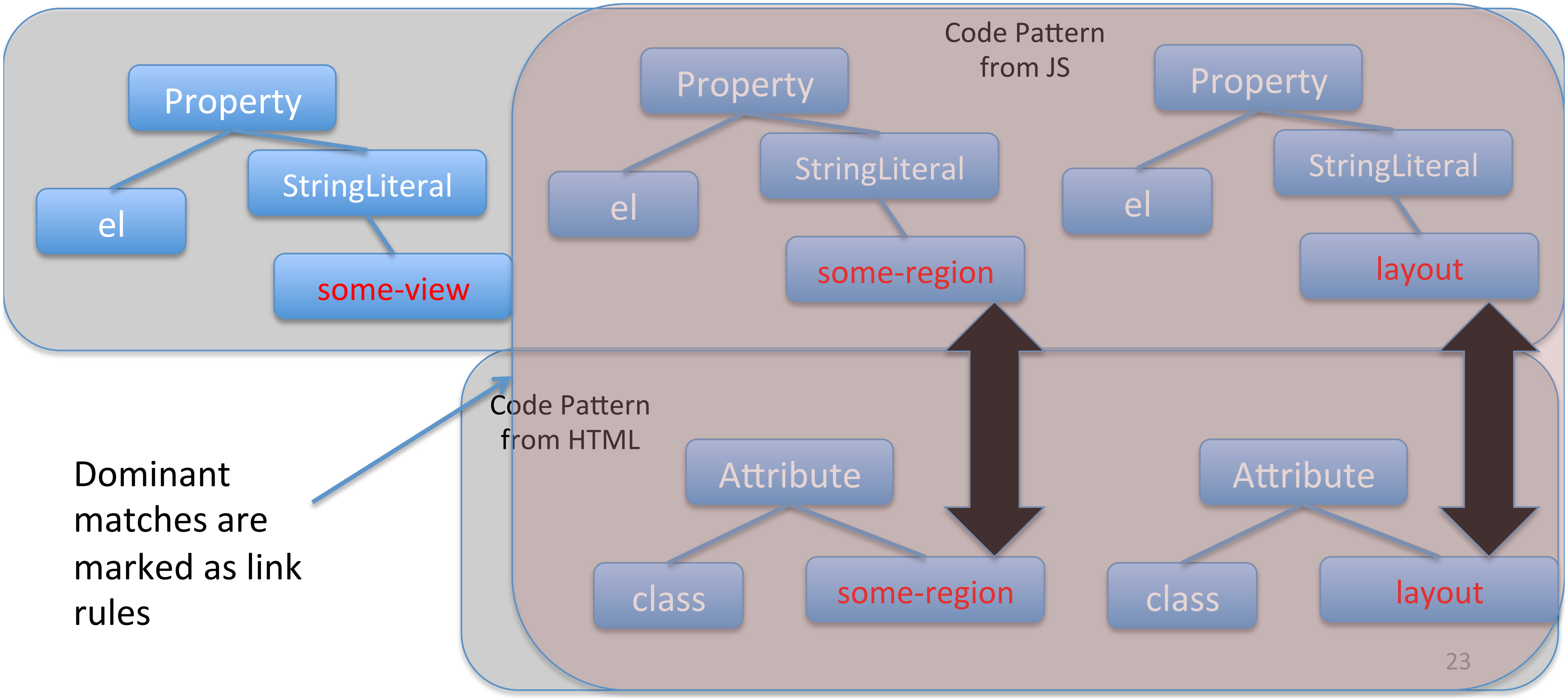
Attribute
class
layout

Step 3: Infer Consistency Rules



LINK RULE

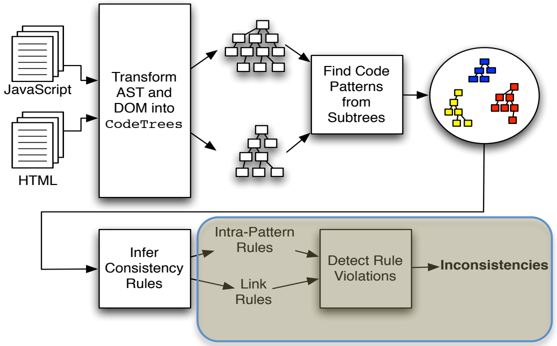
Inconsistencies ***between*** *pattern groups*



Dominant matches are marked as link rules

Step 4: Detect Rule Violations

LINK RULE



JavaScript

HTML

Transform AST and DOM into `CodeTrees`

Find Code Patterns from Subtrees

Infer Consistency Rules

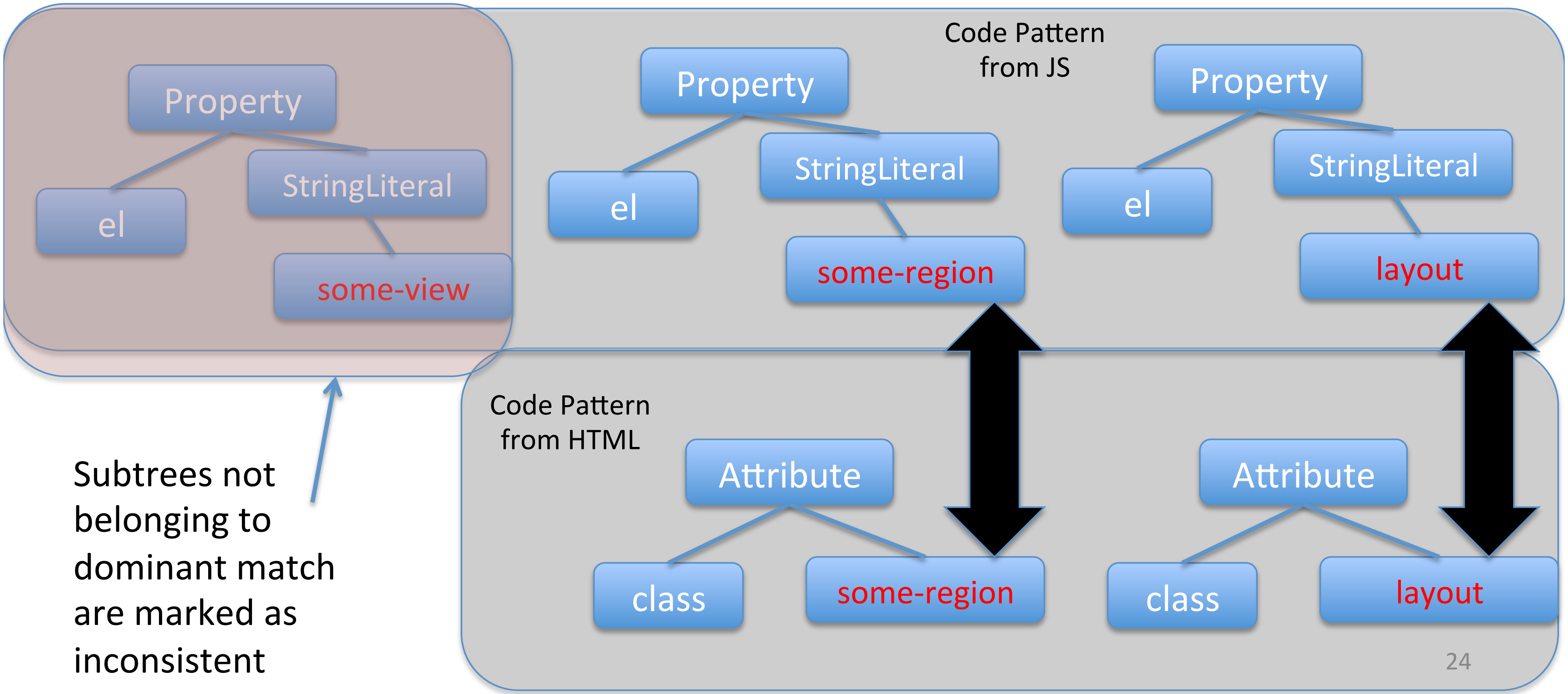
Intra-Pattern Rules

Link Rules

Detect Rule Violations

Inconsistencies

*Inconsistencies **between** pattern groups*



Property

el

StringLiteral

some-view

Code Pattern from JS

Property

el

StringLiteral

some-region

Property

el

StringLiteral

layout

Code Pattern from HTML

Attribute

class

some-region

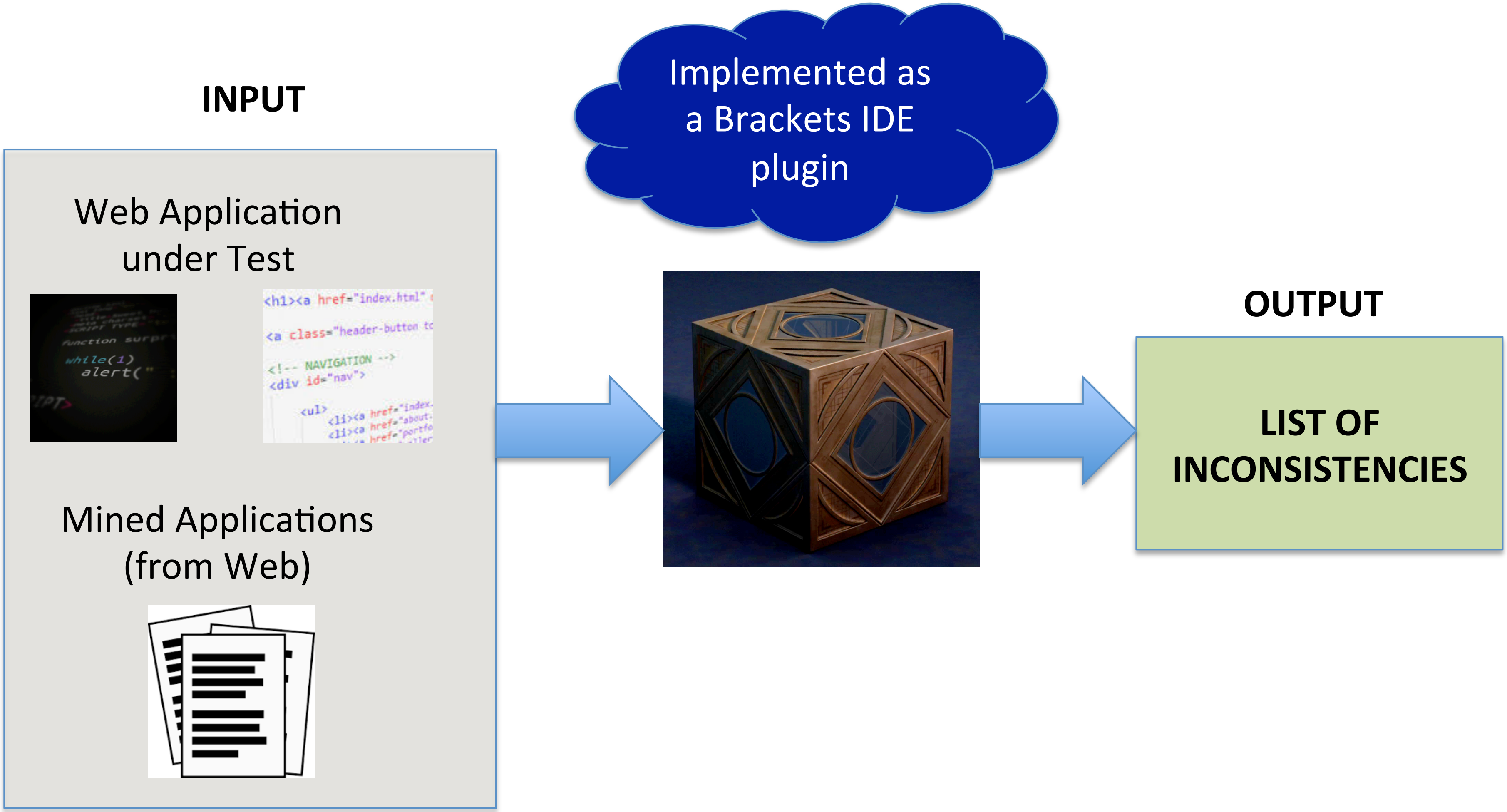
Attribute

class

layout

Subtrees not belonging to dominant match are marked as inconsistent

Holocron



INPUT

Web Application under Test

Mined Applications (from Web)

Implemented as a Brackets IDE plugin

OUTPUT

LIST OF INCONSISTENCIES

Outline

- Motivation and Goals
- Approach
- Evaluation
- Conclusion

Research Questions

RQ1 (Prevalence of Inconsistencies):

Do inconsistencies occur in MVC applications, and if so, what are their characteristics?

RQ2 (Real Bugs and Code Smells):

Can Holocron be used to detect bugs and code smells in real-world MVC applications?

RQ1: Prevalence of Inconsistencies

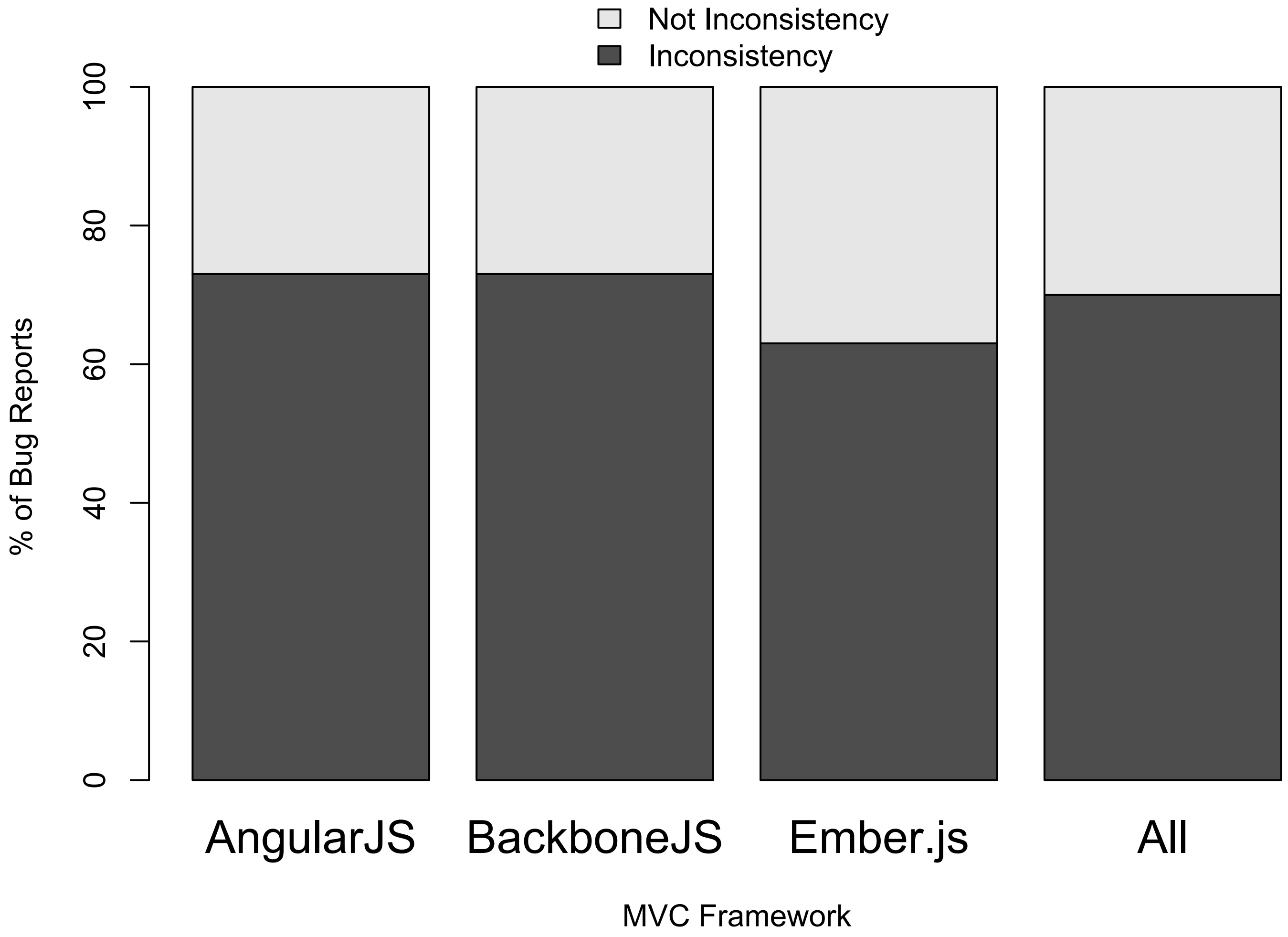
BACKBONE.JS

ANGULARJS

ember

Analyzed **90** GitHub bug reports (30 for each of three main MVC frameworks)

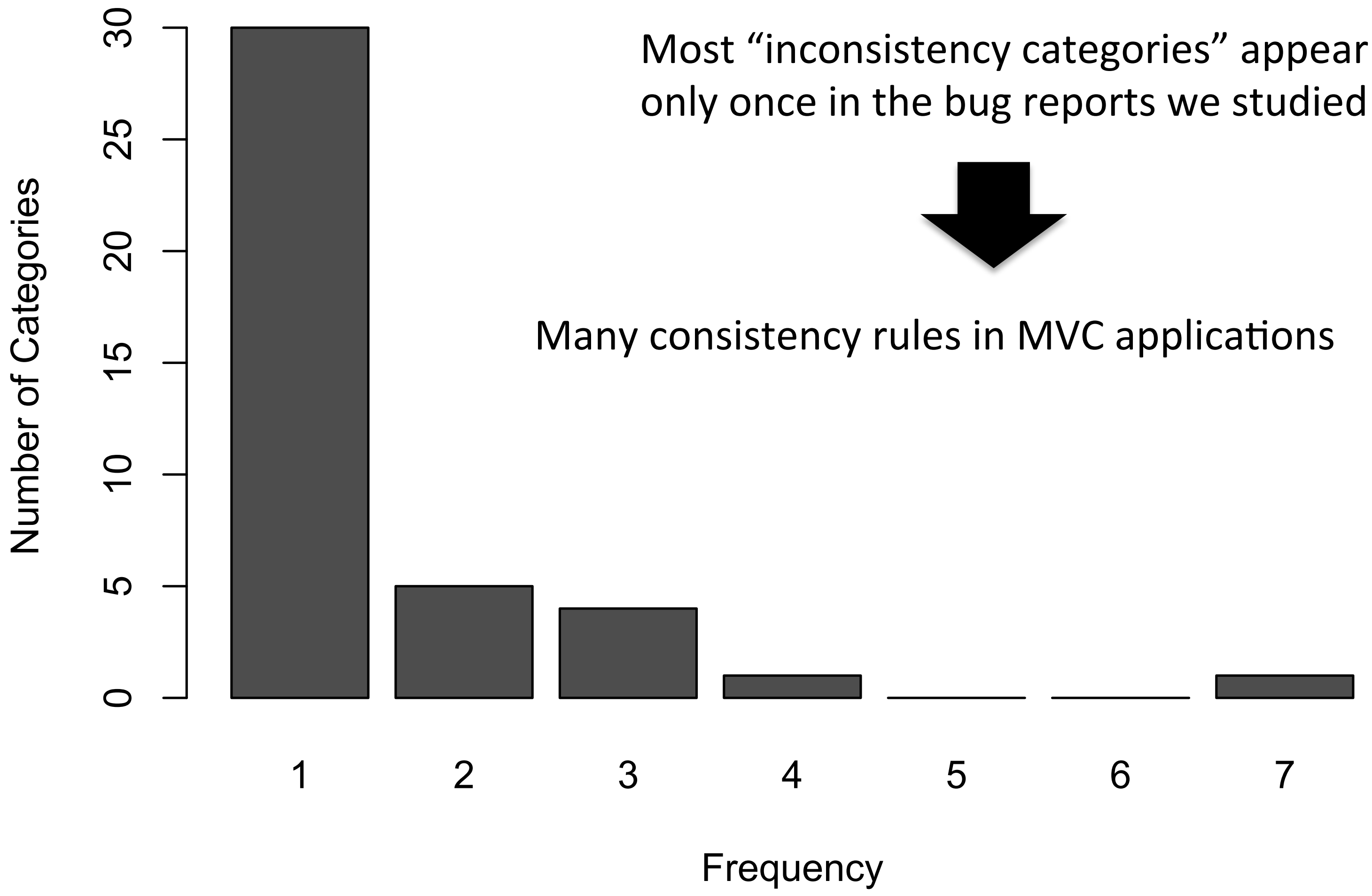
RQ1: Prevalence of Inconsistencies



70% of the bug reports correspond to inconsistencies

35% of the inconsistencies are cross-language - 25% of bug reports

RQ1: Prevalence of Inconsistencies



Most "inconsistency categories" appear only once in the bug reports we studied

Many consistency rules in MVC applications

RQ2: Real Bugs and Code Smells

Ran Holocron on **12 real-world MVC applications** (4 for each framework) – **different** from those used in RQ1

Inspected each output by Holocron, and classified each as a **bug**, a **code smell**, or a **false positive**

RQ2: Real Bugs and Code Smells

18 unreported bugs (95 inconsistencies)

5 cross-language inconsistencies

33 code smells

Around 54% of the inconsistencies are either real bugs or code smells

Took 1.14 minutes on average for each application

Outline

- Motivation and Goals
- Approach
- Evaluation
- Conclusion

Future Work

Smarter way of choosing code examples - fewer false positives

```
function surpr
while(1)
alert("
```

```
<h1><a href="index.html"
<a class="header-button t
<!-- NAVIGATION -->
<div id="nav">
    <ul>
        <li><a href="index.
        <li><a href="about.
        <li><a href="portfo
```

Extending Holocron to non-MVC web applications

Conclusion

- **Bugs and code smells often manifest as inconsistencies in MVC web applications**
  - Many are cross-language inconsistencies
- **Holocron can find bugs and code smells without hardcoded consistency rules**
  - Found 18 unreported bugs (5 cross-language)
  - About 1 in 2 inconsistencies are bugs or smells

https://github.com/karthikp-ubc/Holocron