

Vejovis: Suggesting Fixes for JavaScript Faults

Frolin S. Ocariza, Jr.

Karthik Pattabiraman, Ali Mesbah



University of British Columbia

Problem and Motivation

**JavaScript in web applications
has plenty of reliability issues**

**JS faults are not trivially
fixed [ESEM'13]
issues [ESEM'13]**

Problem and Motivation

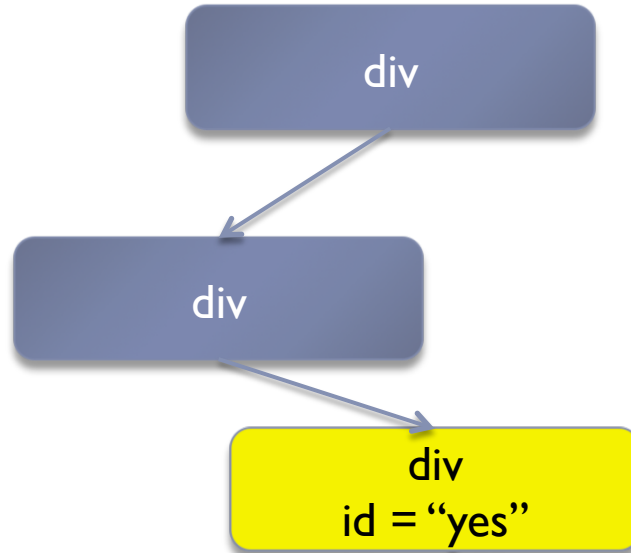
**JavaScript in web applications
has plenty of reliability issues,
these JavaScript faults matter
and these JavaScript faults are
non-trivial to fix**

Faults in JavaScript Code

- ▶ Study of JS bug reports [ESEM'13]
- ▶ **Key Insight:** Most (65%) mistakes programmers make in JS propagate to ***parameters of DOM API method calls***
 - ▶ DOM API methods: getElementById, getElementsByTagName, jQuery's \$(), etc.
 - ▶ We also found that such faults are the most impactful, and take the longest to fix

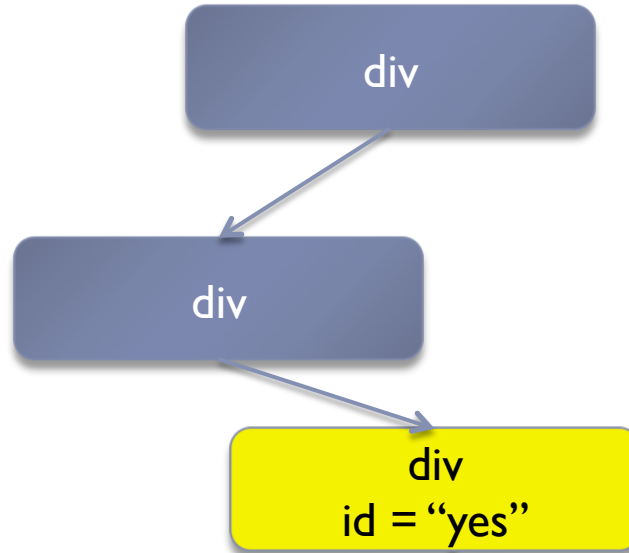
DOM-RELATED FAULTS

DOM-Related Fault Example



```
var x = "yes";  
var elem = document.getElementById(x);
```

DOM-Related Fault Example

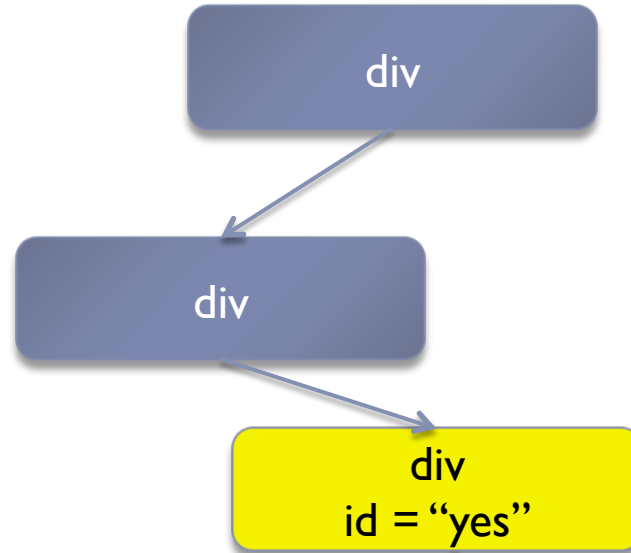


`var x = "no";`
`var elem = document.getElementById(x);`

MISTAKE!

An arrow points from the word 'MISTAKE!' to the variable 'x' in the first line of code.

DOM-Related Fault Example



`var x = "no";`
`var elem = document.getElementById(x);`

MISTAKE!

ID parameter evaluates to "no", which is not in the DOM

Goal

Facilitate the process of fixing DOM-related faults

Fault Model

- ▶ Suggest repairs for ***DOM-related faults***
- ▶ Only one mistake made

Common Developer Fixes

- ▶ Study of 190 *fixed* bug reports from 12 web apps


```
elem = getElementById(param)  
elem.innerHTML = "..."
```

Common Developer Fixes

- ▶ Study of 190 *fixed* bug reports from 12 web apps

Modify the parameter

```
elem = getElementById(new_param)  
elem.innerHTML = "..."
```



Ways Programmers Fix Faults

- Parameter Modification

Common Developer Fixes

- ▶ Study of 190 *fixed* bug reports from 12 web apps

```
elem = getElementById(param)
```

```
if (elem) ←————— Check if null
```

```
elem.innerHTML = "..."
```

Ways Programmers Fix Faults

- Parameter Modification
- DOM Element Validation

Common Developer Fixes

- ▶ Study of 190 *fixed* bug reports from 12 web apps

```
elem = querySelector(param)  
elem.innerHTML = "..."
```



Modify the method

Ways Programmers Fix Faults

- Parameter Modification
- DOM Element Validation
- Method Modification

Common Developer Fixes

- ▶ Study of 190 *fixed* bug reports from 12 web apps

```
elem = getElementById(param)  
elem.innerHTML = "..."
```

Ways Programmers Fix Faults

- Parameter Modification 27.2%
- DOM Element Validation 25.7%
- Method Modification 24.6%

Structure in DOM Method Parameters

WRONG

`getElementById("no")`

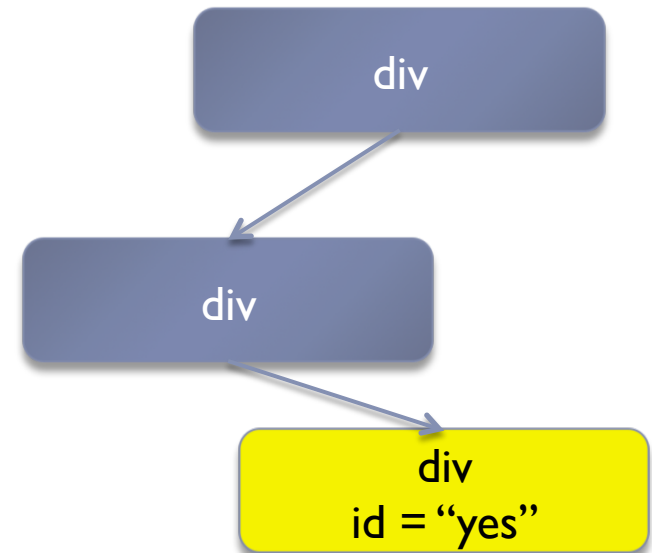
RIGHT

???

Question: How do we know that we should replace “no” with “yes”

Answer: We need to infer programmer *intent*

- Very difficult to do in general, but...
- We have the DOM!



Structure in DOM Method Parameters

WRONG

`getElementById("no")`

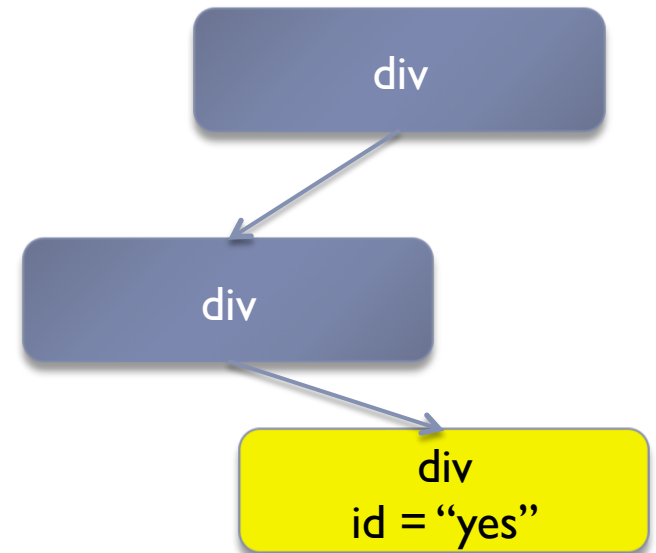
RIGHT

`getElementById("yes")`

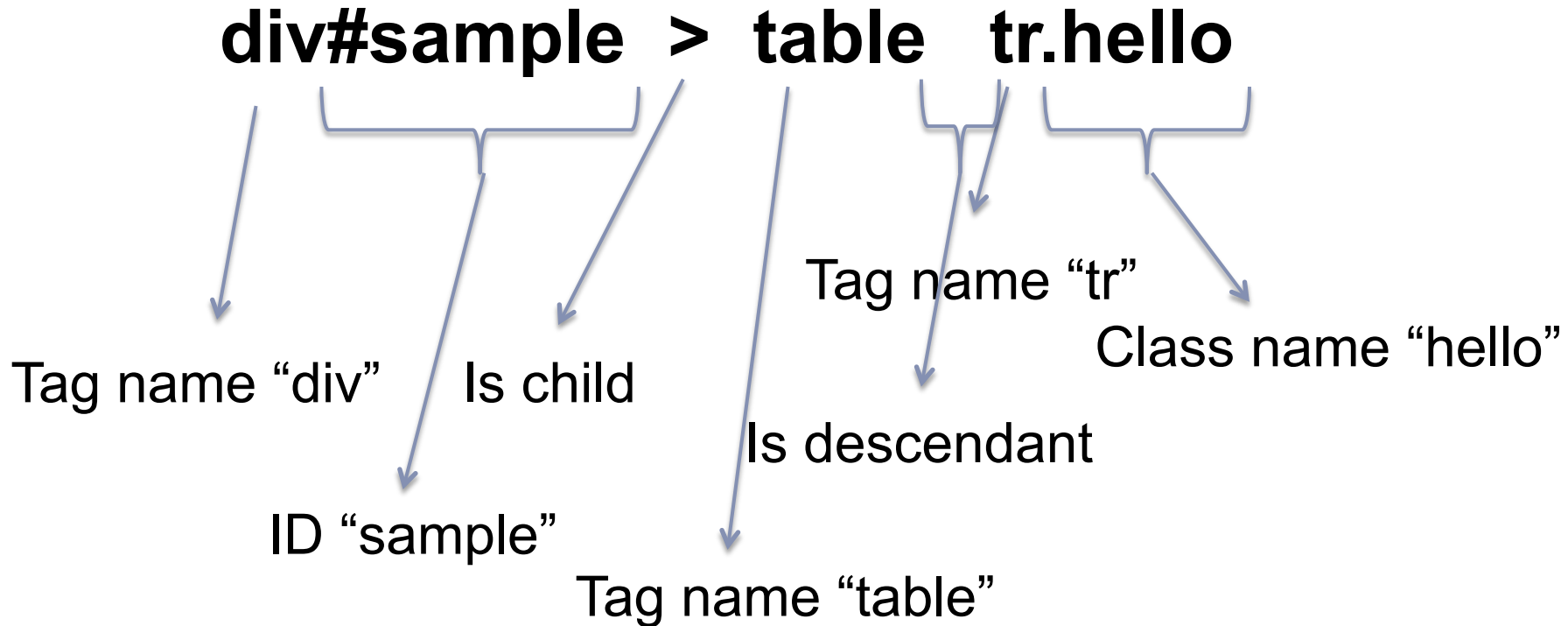
Question: How do we know that we should replace “no” with “yes”

Answer: We need to infer programmer *intent*

- Very difficult to do in general, but...
- We have the DOM!

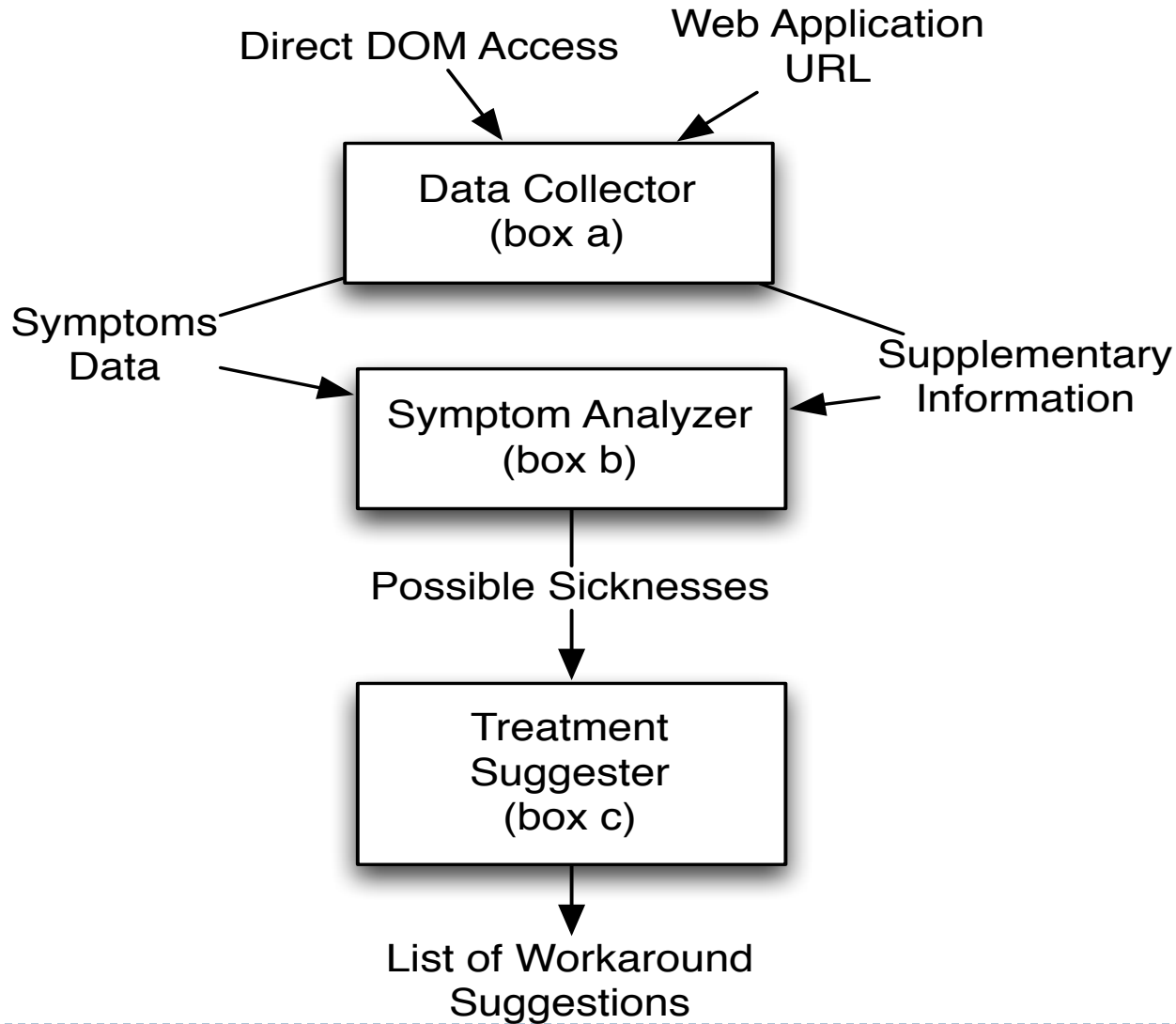


CSS Selectors



Input to `querySelector()`, `$()`, etc. to retrieve list of elements


Design



Running Example

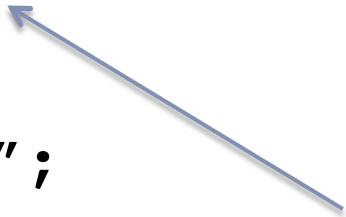
```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2);
7 e[0].innerHTML = "new content";
```

Access
DOM
element
using CSS
selector



Running Example

```
1 firstTag = "div";  
2 prefix = "pain-";  
3 suffix = "elem";  
4 level1 = firstTag + "#" + prefix + suffix;  
5 level2 = "span.cls";  
6 e = $(level1 + " " + level2);  
7 e[0].innerHTML = "new content";
```



Lines to set
up the
CSS selector
passed to \$()

Running Example

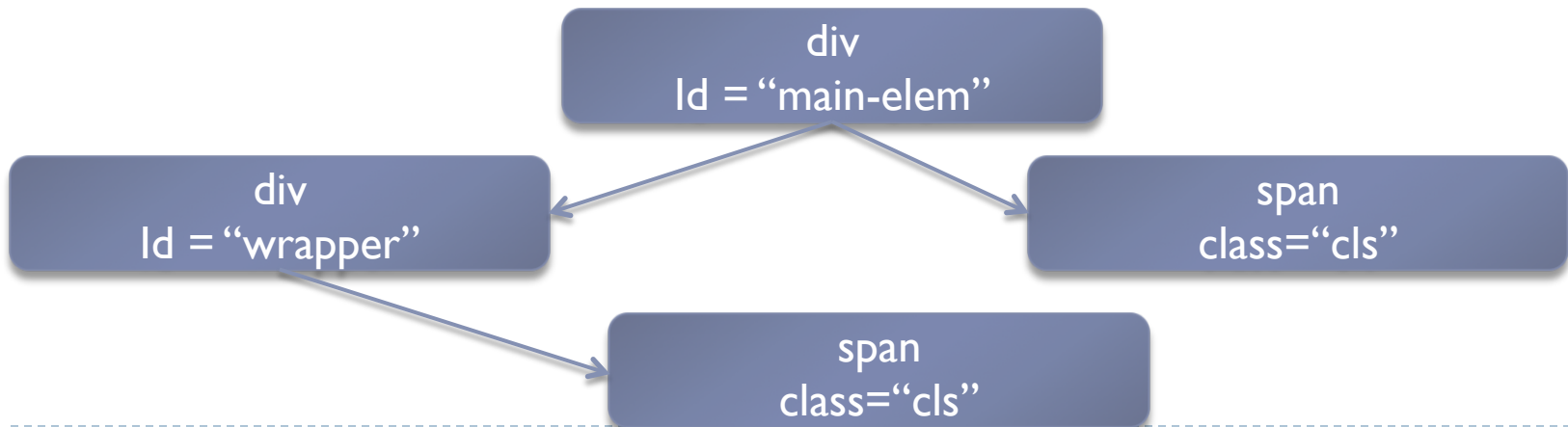
```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2);
7 e[0].innerHTML = "new content";
```

Constructed selector: **div#pain-elem span.cls**

Running Example

```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2);
7 e[0].innerHTML = "new content";
```

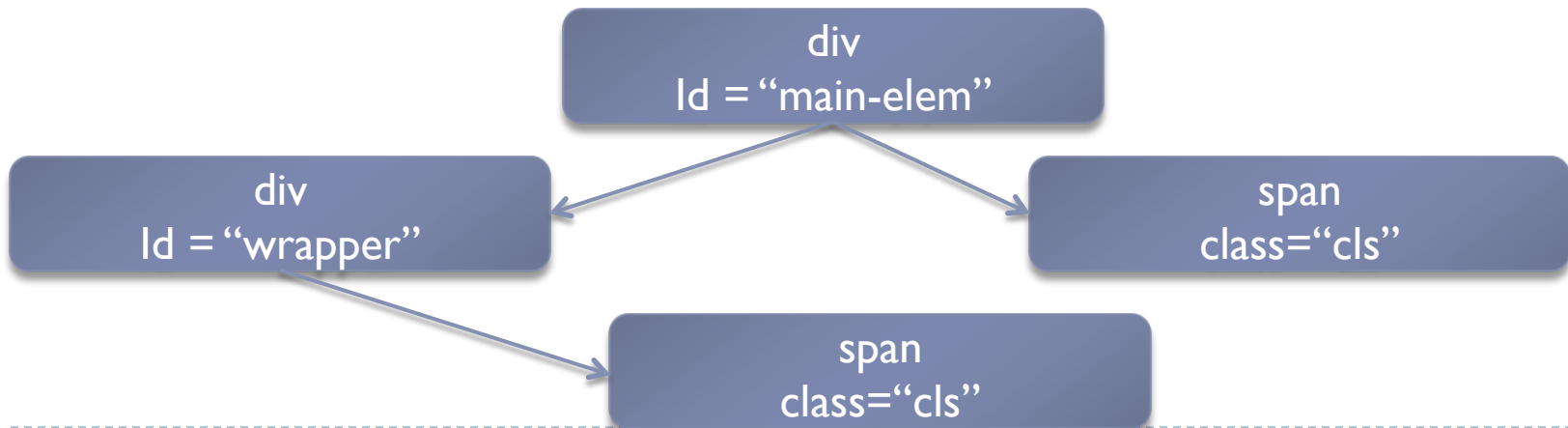
Constructed selector: **div#pain-elem span.cls**



Running Example

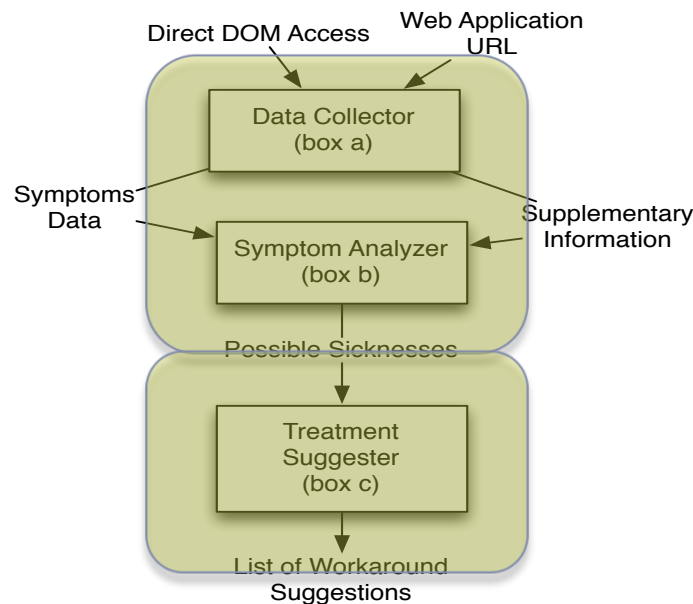
```
1 firstTag = "div";
2 prefix = "pain-";
3 suffix = "elem";
4 level1 = firstTag + "#" + prefix + suffix;
5 level2 = "span.cls";
6 e = $(level1 + " " + level2); ← Would return empty set!
7 e[0].innerHTML = "new content";
```

Constructed selector: **div#pain-elem span.cls**



Main Idea

- ▶ **Parameter Analysis:** What portion of the parameter do we replace?
- ▶ **Context Analysis:** How do we perform the replacement in the code?

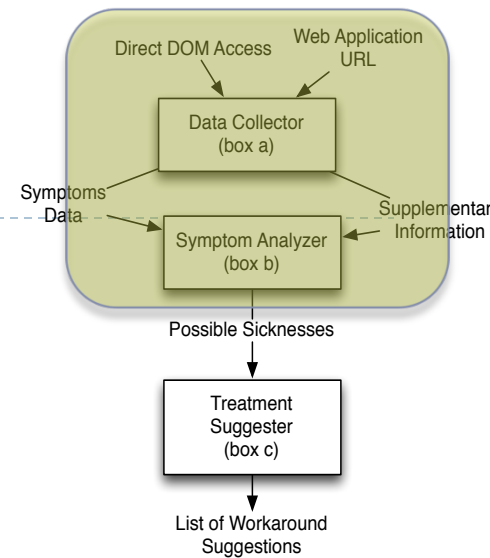


Parameter Analysis: Dividing Components

Invalid selector: `div#pain-elem span.cls`

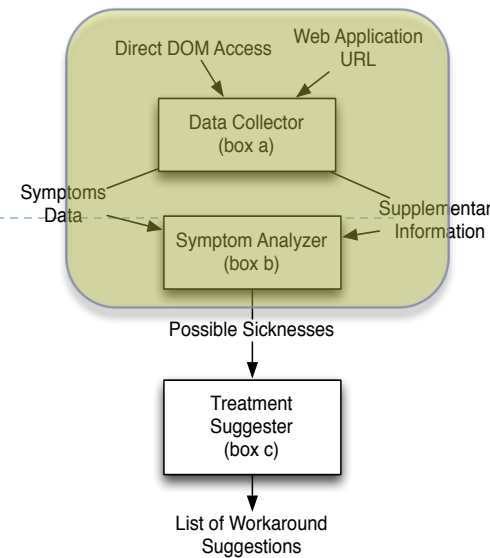
Divide into components

div | # | pain-elem | | span | . | cls

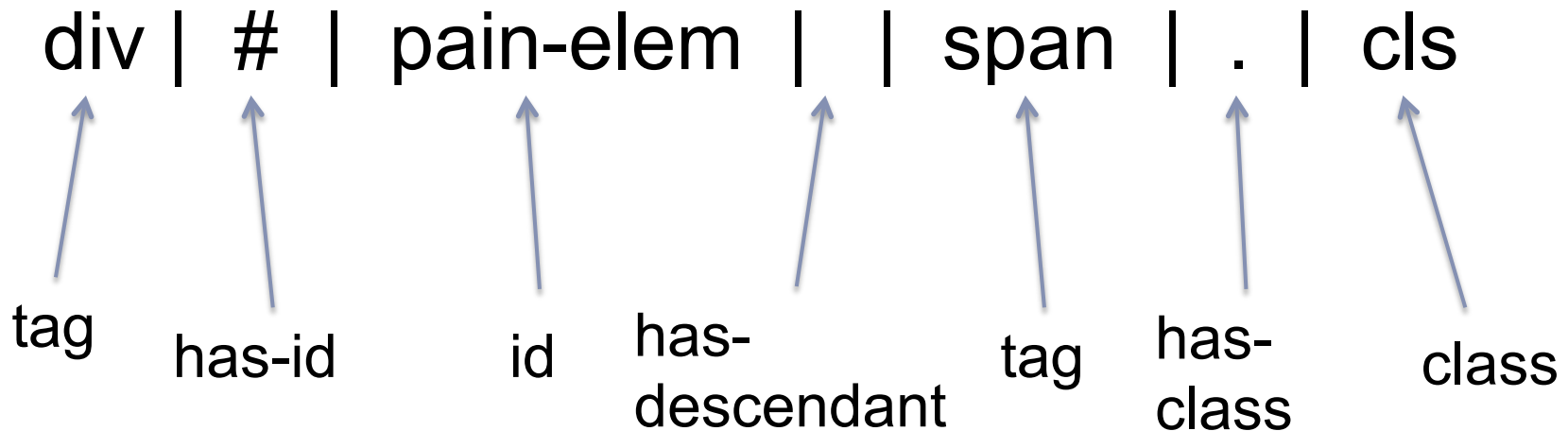


Parameter Analysis: Dividing Components

Invalid selector: `div#pain-elem span.cls`



Divide into components



Parameter Analysis: Dividing Components

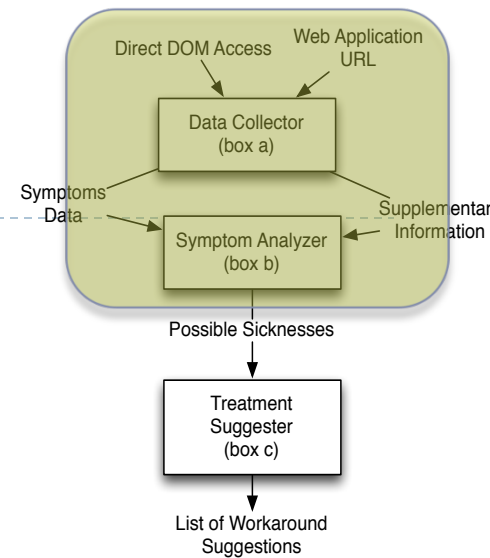
Invalid selector: `div#pain-elem span.cls`

Subdivide each component according to dynamic backward slice

div | # | pain- | elem | | span | . | cls

Line 2

Line 3

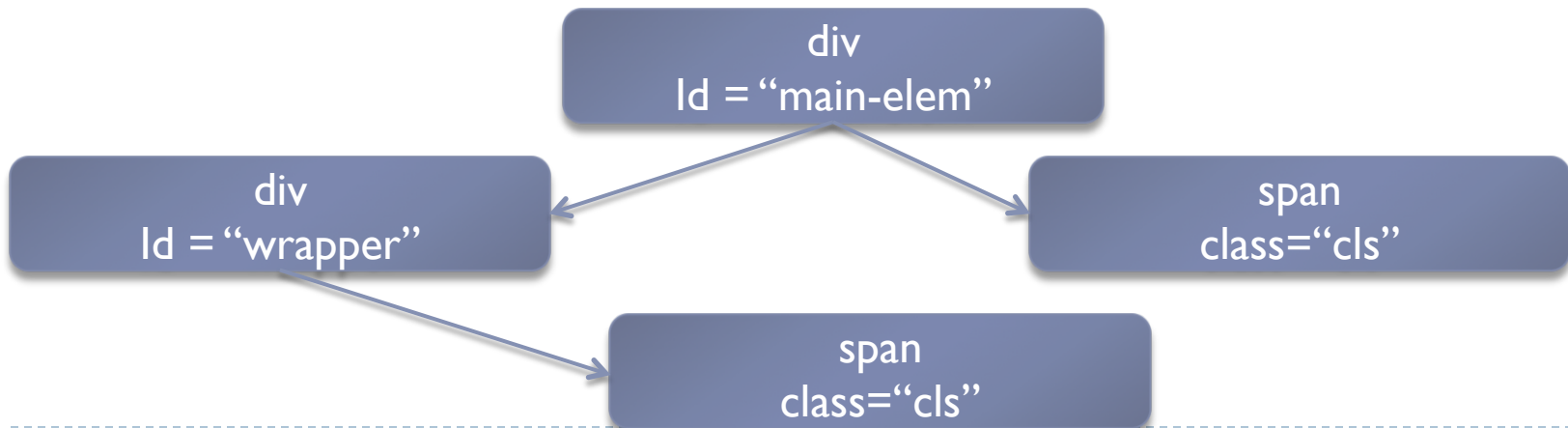


Parameter Analysis: Dividing Components

```
1 firstTag = "div";  
2 prefix = "pain-";  
3 suffix = "elem";  
4 level1 = firstTag + "#" + prefix + suffix;  
5 level2 = "span.cls";  
6 e = $(level1 + " " + level2);  
7 e[0].innerHTML = "new content";
```

Backward slice
of "pain-elem"

Invalid selector: **div#pain-elem span.cls**

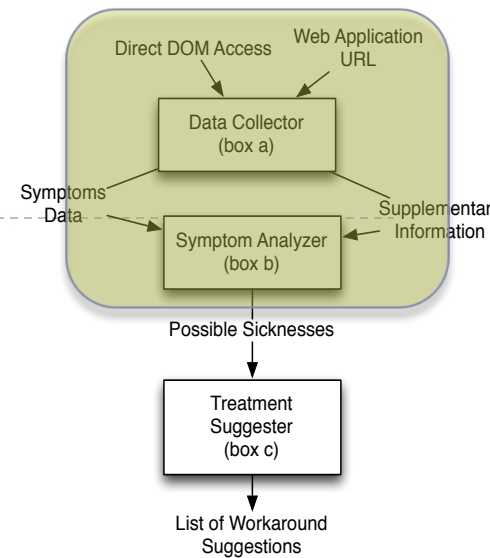


Parameter Analysis: Dividing Components

Invalid selector: `div#pain-elem span.cls`

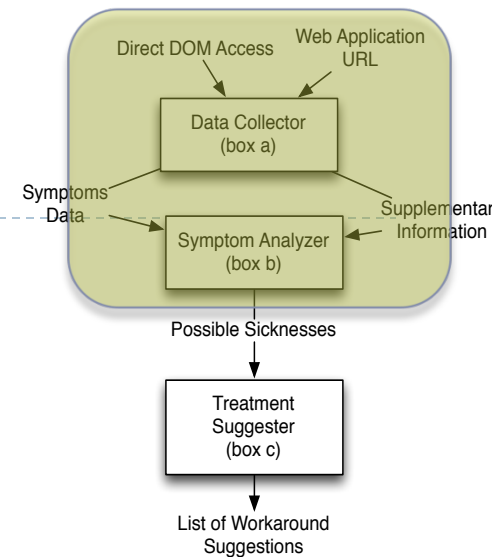
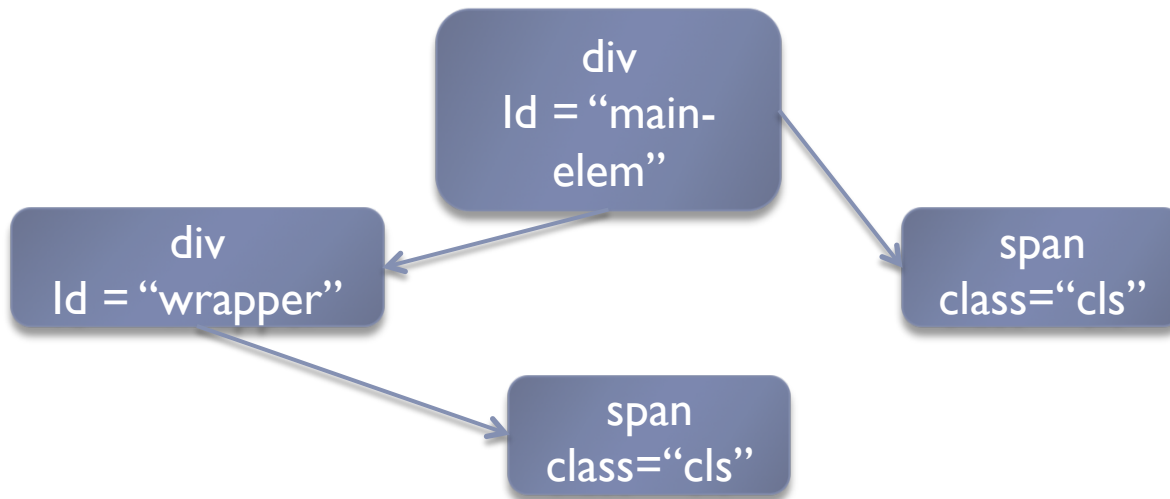
Subdivide each component according to dynamic backward slice

div | # | pain- | elem | | span | . | cls



Parameter Analysis: Finding Valid Selectors

Invalid selector: `div#pain-elem span.cls`



Construct VALID selectors from current DOM that are “sufficiently close” to the erroneous one

Parameter Analysis: Finding Valid Selectors

Invalid selector: `div#pain-elem span.cls`

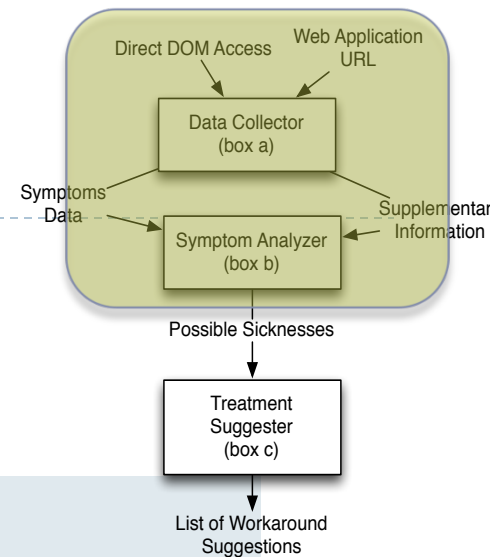
List of valid selectors:

`div#main-elem span.cls`

`div#wrapper span.cls`

`span`
`class="cls"`

Construct VALID selectors from current DOM that are “sufficiently close” to the erroneous one



Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: `div#pain-elem span.cls`

div | # | pain- | elem | | span | . | cls

Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: `div#pain-elem span.cls`

div | # | **pain-** | elem | | span | . | cls

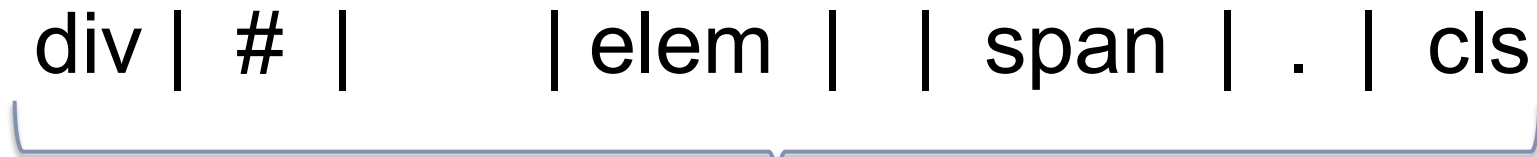
Assumed
incorrect



Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: `div#pain-elem span.cls`

div | # | | elem | | span | . | cls



List of valid selectors:

div#main-elem span.cls
div#wrapper span.cls

Use as pattern

Parameter Analysis: Inferring Possible Replacements [Example]

Invalid selector: `div#pain-elem span.cls`

div | # | | elem | | span | . | cls

List of valid selectors:

`div#main-elem span.cls` – **MATCHES PATTERN!**

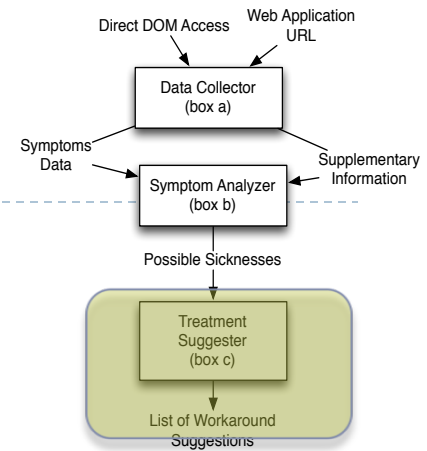
`div#wrapper span.cls`

Context Analysis

```
1 firstTag = "div";  
2 prefix = "pain-";  
3 suffix = "elem";  
4 level1 = firstTag + "#" + prefix + suffix;  
5 level2 = "span.cls";  
6 e = $(level1 + " " + level2);  
7 e[0].innerHTML = "new content";
```

Invalid selector: **div#pain-elem span.cls**

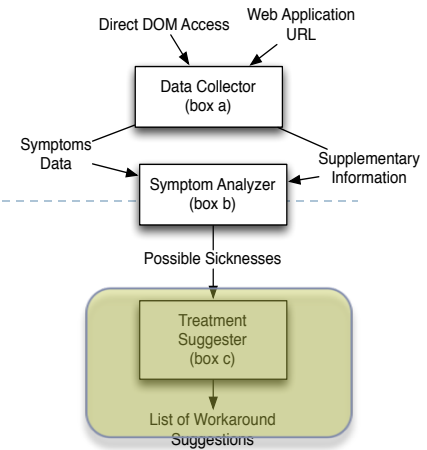
Replacement selector: **div#main-elem span.cls**



Context Analysis

```
1 firstTag = "div";  
2 prefix = "main-";  
3 suffix = "elem";  
4 level1 = firstTag + "#" + prefix + suffix;  
5 level2 = "span.cls";  
6 e = $(level1 + " " + level2);  
7 e[0].innerHTML = "new content";
```

String literal replaced



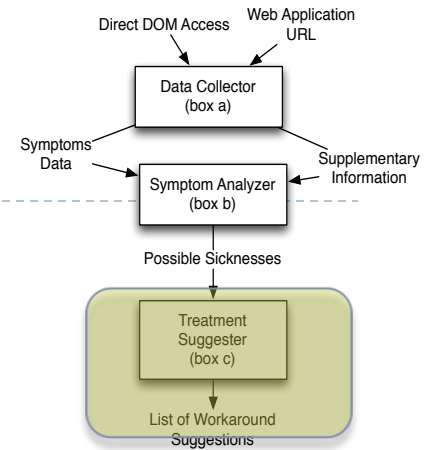
Invalid selector: **div#pain-elem span.cls**

Replacement selector: **div#main-elem span.cls**

Context Analysis

```
1 firstTag = "div";  
2 prefix = "main-";  
3 suffix = "elem";  
4 level1 = firstTag + "#" + prefix + suffix;  
5 level2 = "span.cls";  
6 e = $(level1 + " " + level2);  
7 e[0].innerHTML = "new content";
```

String literal replaced



Invalid selector: **div#pain-elem span.cls**

Replacement selector: **div#main-elem span.cls**

Message:

REPLACE STRING LITERAL “pain-” in line 2 with string literal “main-”

Context Analysis: Non-“Replace” Messages

- ▶ Loops – “replace” may be unsafe
- ▶ String value doesn’t originate from string literal
- ▶ **Analyze the context!**

MESSAGE TYPES
REPLACE
REPLACE AT ITERATION
OFF BY ONE AT BEGINNING
OFF BY ONE AT END
MODIFY UPPER BOUND
EXCLUDE ITERATION
ENSURE

Implementation

▶ **Vejovis**

<http://ece.ubc.ca/~frolino/projects/vejovis>

- ▶ Data collection: Rhino and Crawljax
- ▶ Pattern matching: Hampi

Usage Model

INPUTS

URL

AUTOFLOX

DOM METHOD LOCATION

VEJOVIS

OUTPUT

LIST OF ACTIONABLE REPAIR MESSAGES

Evaluation: Research Questions

RQ1: What is the accuracy of Vejovis in suggesting a correct repair?

RQ2: How quickly can Vejovis determine possible replacements? What is its performance overhead?

RQ1: Accuracy of VejoVis

Subjects	JS Code Size (KB)
Drupal	213
Ember.js	745
Joomla	434
jQuery	94
Moodle	352
MooTools	101
Prototype	164
Roundcube	729
TYPO3	2252
WikiMedia	160
WordPress	197

- 22 bug reports (2 per app, and randomly chosen)
- Replicated bug and ran with VejoVis
- ***Recall*** and ***Precision***

RECALL: 100% if correct fix appears; 0% otherwise

PRECISION: Measure of extraneous suggestions

RQ1: Recall

Subject	Bug Report #1	Bug Report #2
Drupal	✓	✓
Ember.js	✓	✓
Joomla	✓	✓
jQuery	✓	✗
Moodle	✓	✓
MooTools	✓	✓
Prototype	✓	✓
Roundcube	✓	✗
TYPO3	✓	✓
WikiMedia	✓	✓
WordPress	✓	✓

**Overall
Recall: 91%**

RQ1: Precision

Subject	Bug Report #1	Bug Report #2
Drupal	3%	25%
Ember.js	50%	33%
Joomla	1%	1%
jQuery	1%	0%
Moodle	3%	3%
MooTools	50%	50%
Prototype	17%	50%
Roundcube	1%	0%
TYPO3	1%	100%
WikiMedia	4%	1%
WordPress	3%	1%

Avg. Precision: 2%

49 suggestions per bug on average!

Improvements

1. Edit distance bound
2. Ranked suggestions

Alternative: Ranking

Subject	Bug Report #1	Bug Report #2
Drupal	31 / 40	1 / 4
Ember.js	1 / 2	1 / 3
Joomla	1 / 88	1 / 88
jQuery	2 / 108	-
Moodle	2 / 37	1 / 37
MooTools	2 / 2	1 / 2
Prototype	1 / 6	1 / 2
Roundcube	4 / 79	-
TYPO3	1 / 187	1 / 1
WikiMedia	6 / 24	1 / 71
WordPress	13 / 30	1 / 170

#1 Ranking in 13 out of 20 bugs

Conservative ranking

Ranking seems to be beneficial

RQ2: Performance

- ▶ Takes average of **44 seconds** to find correct fix
- ▶ Worst case: 91.1 seconds (Joomla)

Threats to Validity

- ▶ **External:** Evaluated on 11 web apps
- ▶ **Internal:** Took bugs from earlier empirical study

Conclusion

- ▶ Vejovis: replacement suggestor for DOM-related faults
 - ▶ Project Link: <http://ece.ubc.ca/~frolino/projects/vejovis>
- ▶ Evaluated on 22 real-world bugs
 - ▶ Good recall – 91%
 - ▶ Correct fix ranked #1 in 13/20 cases
 - ▶ Average 44 s to complete