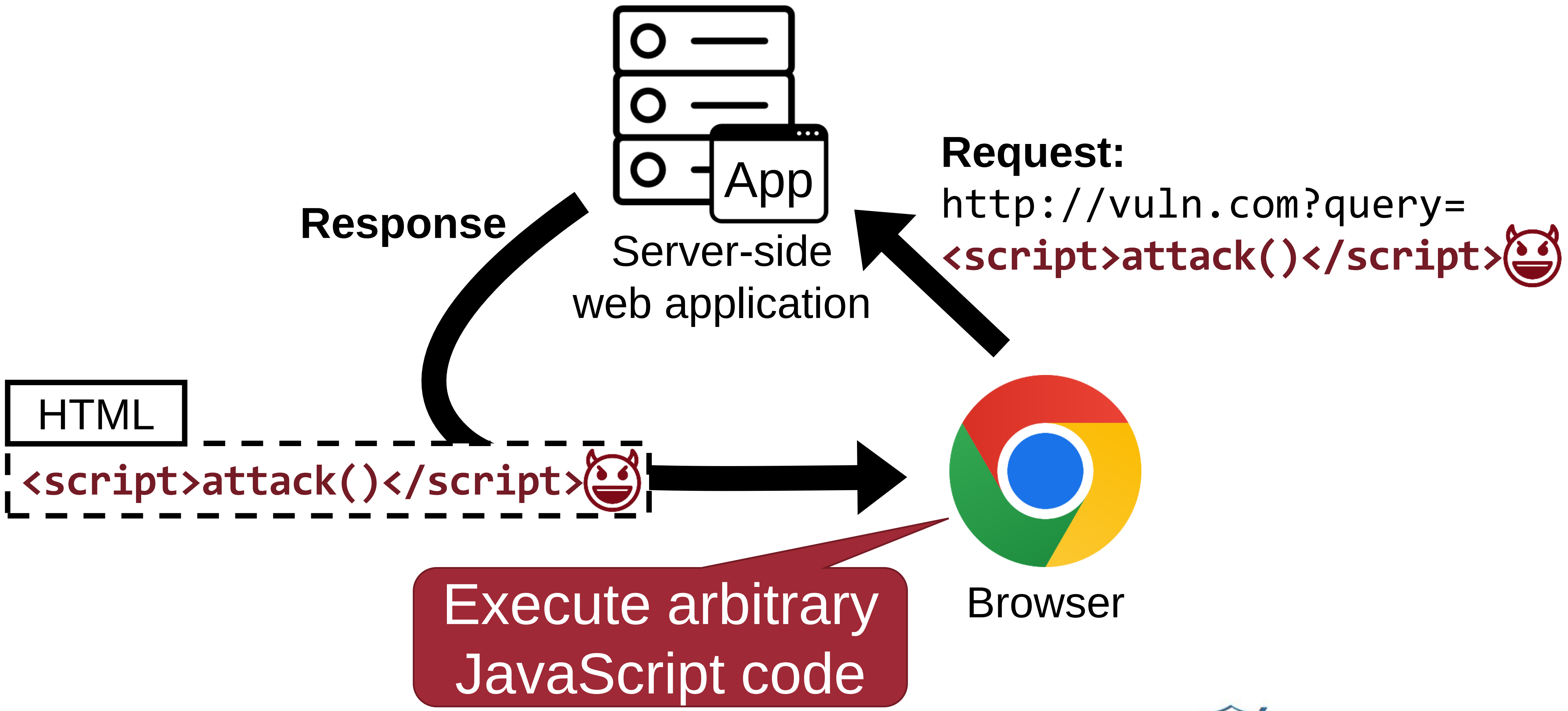
DiffCSP: Finding Browser Bugs in Content Security Policy Enforcement through Differential Testing

Seongil Wi[*], Trung Tin Nguyen[†], Jihwan Kim[*], Ben Stock[†], Sooel Son[*]

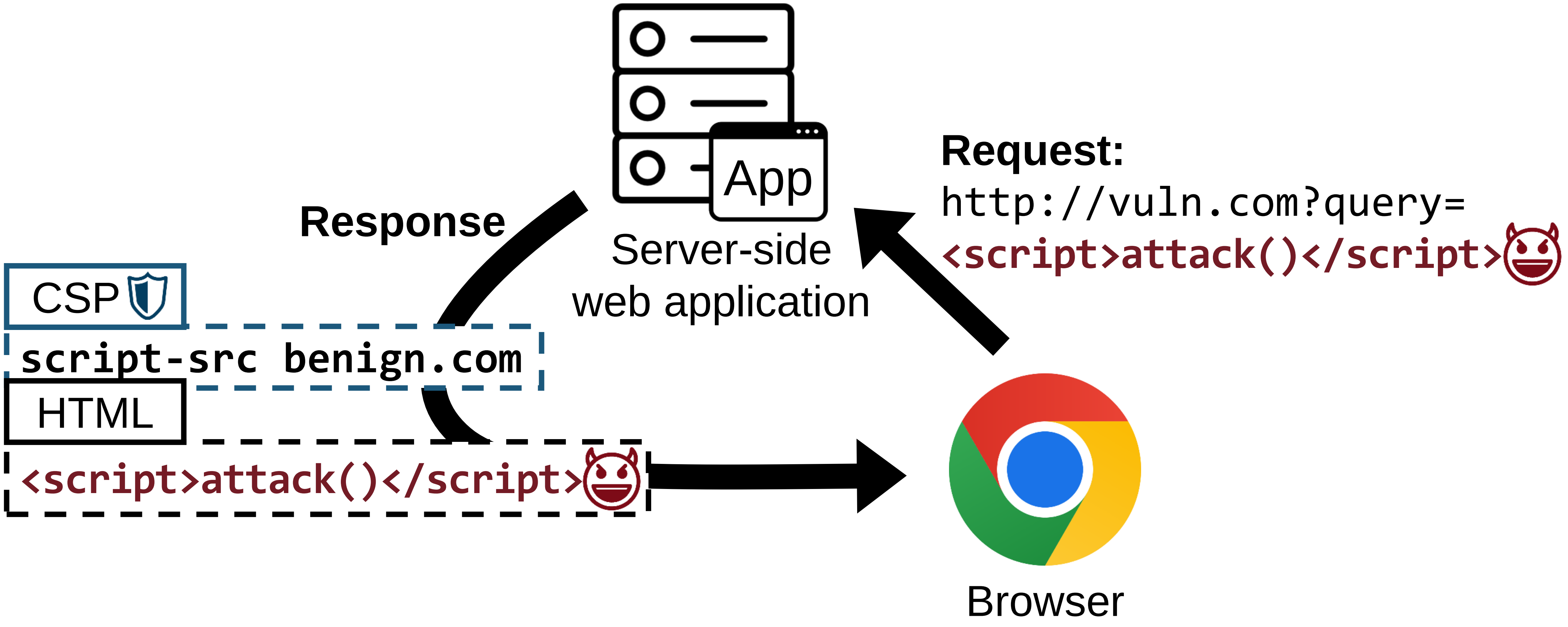
[*]KAIST [†]CISPA Helmholtz Center for Information Security

NDSS 2023

Cross-Site Scripting (XSS) Attacks

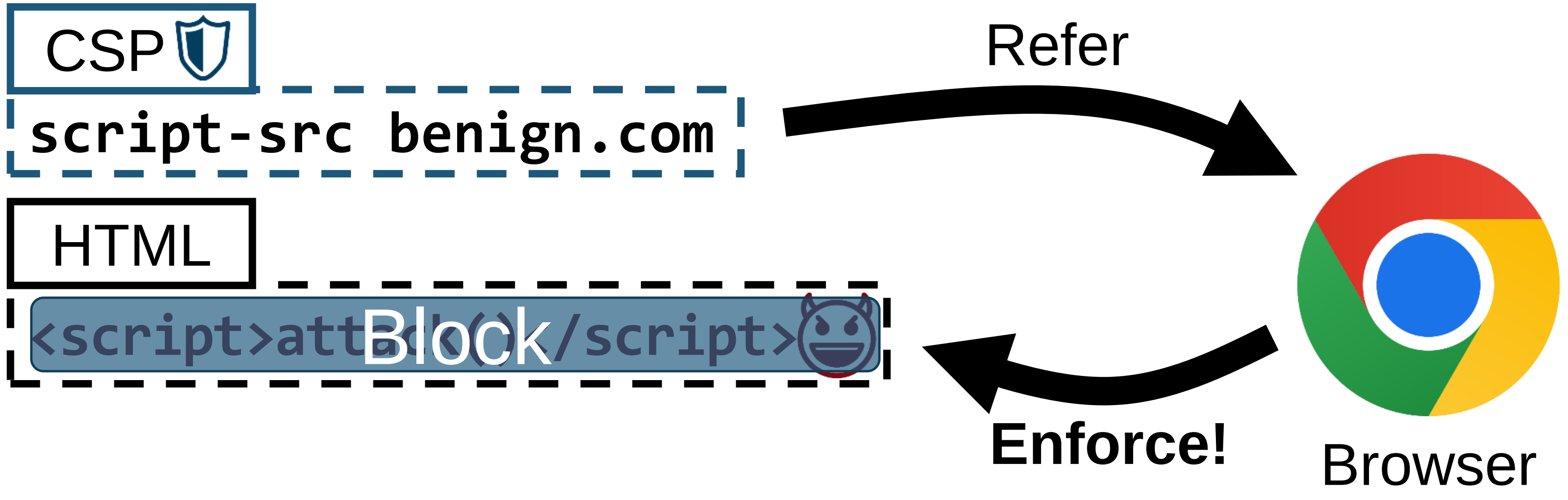

App
Server-side web application
Response
Request:
http://vuln.com?query=<script>attack()</script>😈
HTML
<script>attack()</script>😈
Browser
Execute arbitrary JavaScript code

Content Security Policy (CSP)


App
Server-side web application
Request:
http://vuln.com?query=
<script>attack()</script>😈
Response
CSP
script-src benign.com
HTML
<script>attack()</script>😈
Browser

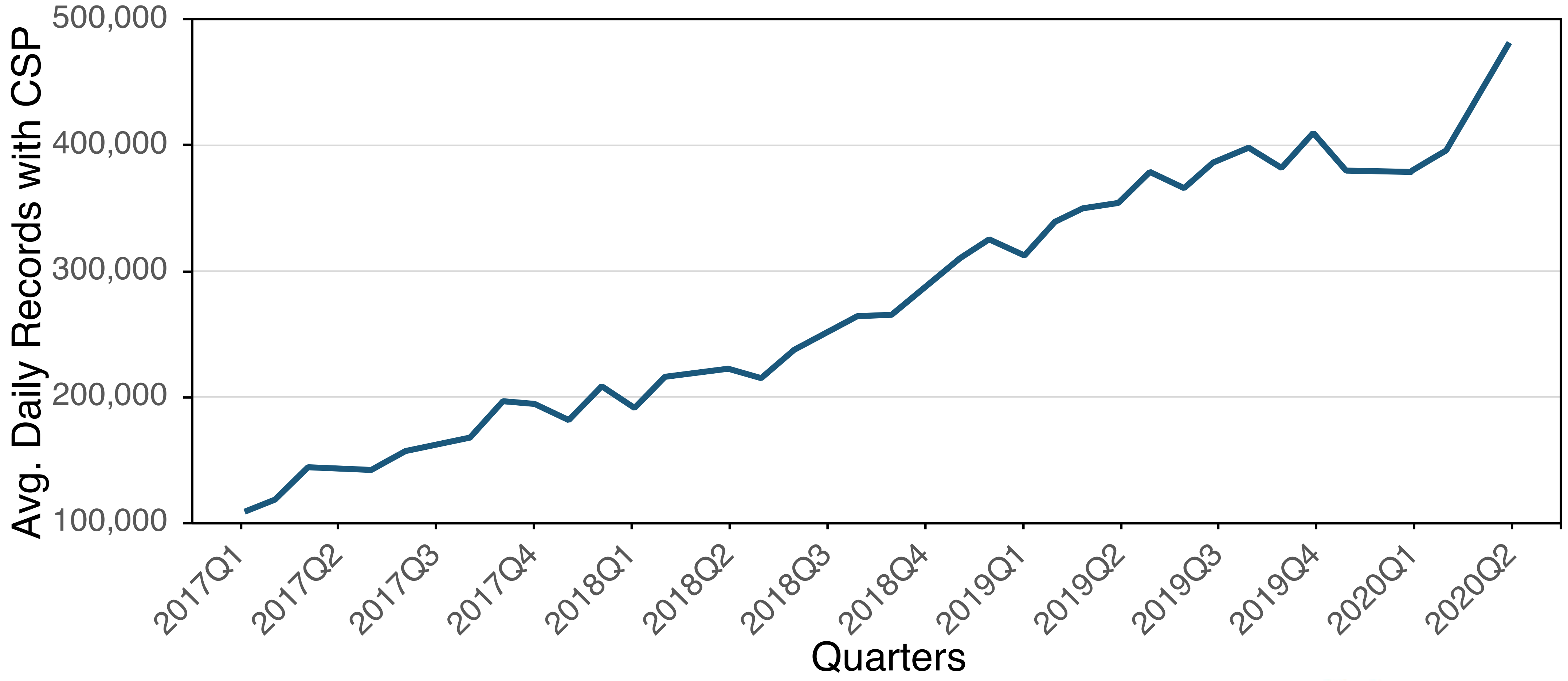
Content Security Policy (CSP)

- A browser-enforced security mechanism



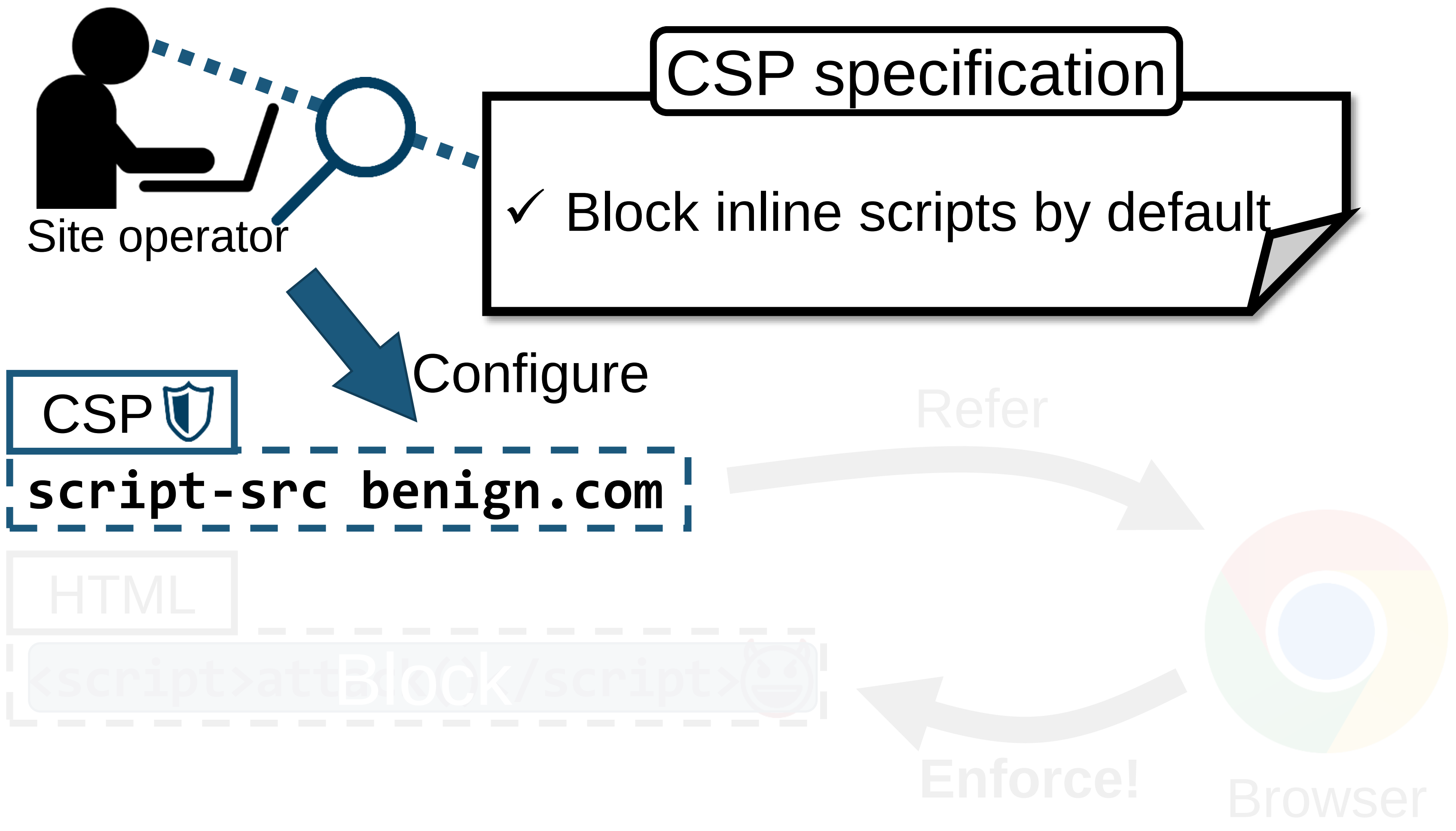
Content Security Policy (CSP)

- A browser-enforced security mechanism



https://www.bitsight.com/blog/content-security-policy-limits-dangerous-activity-so-why-isnt-everyone-doing-it

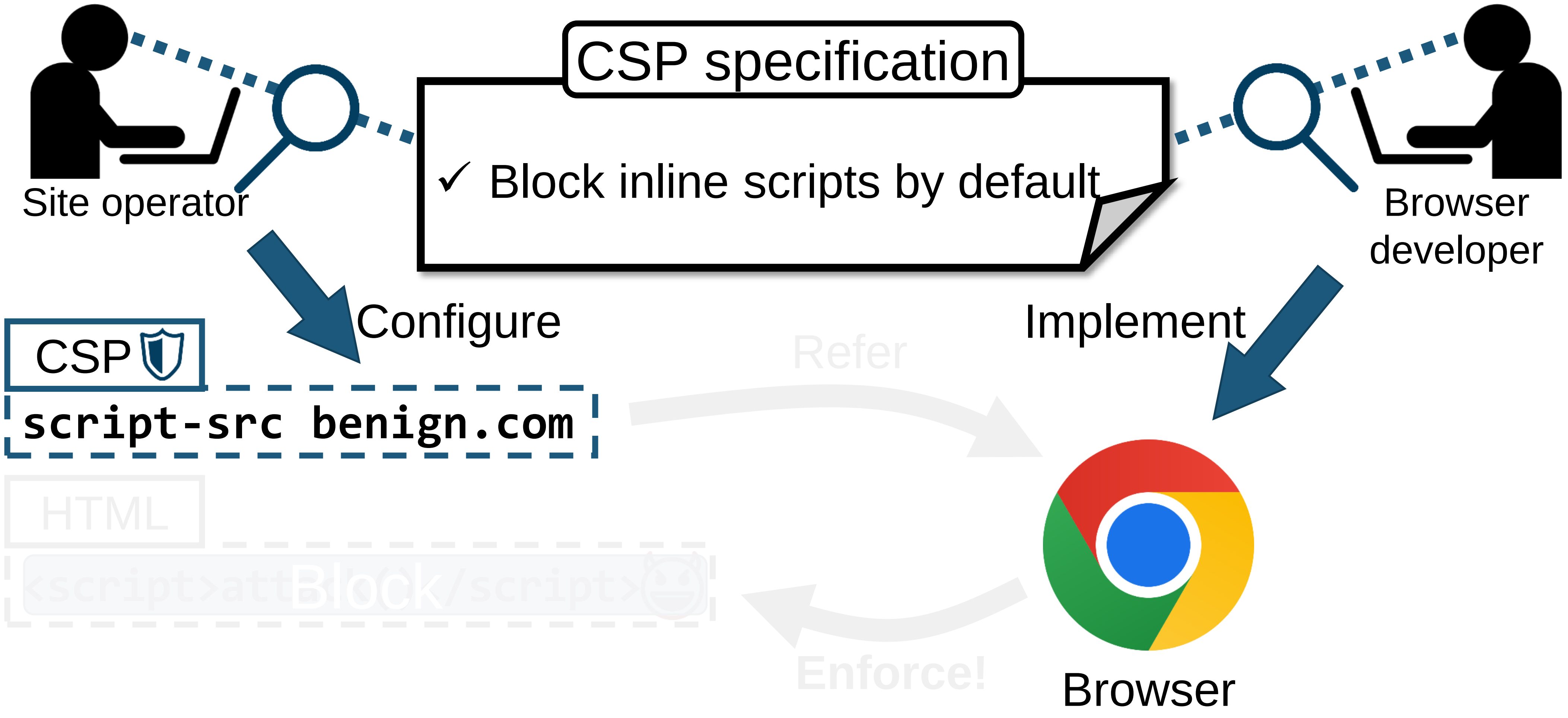
CSP Ecosystem



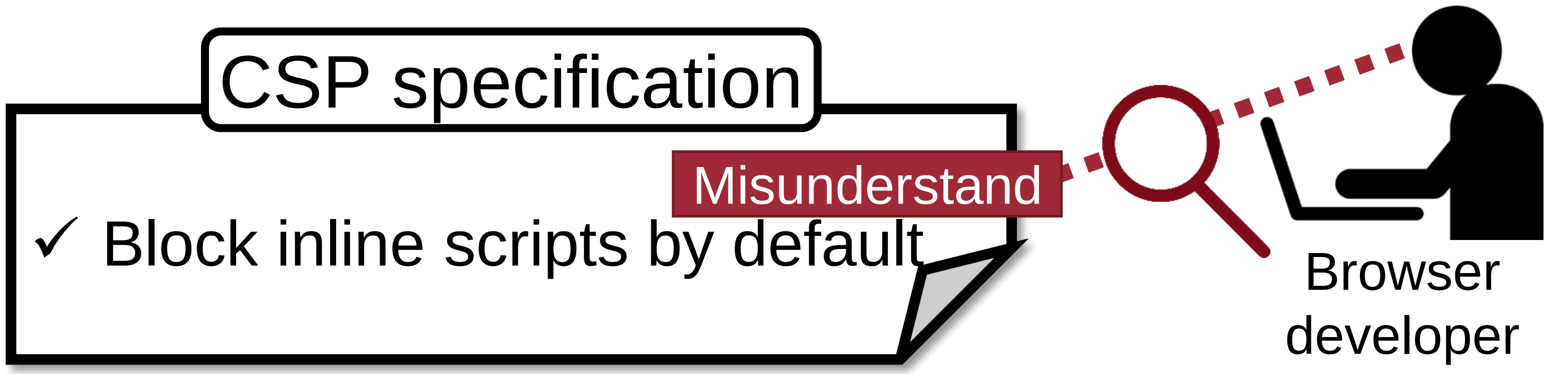
```
script-src benign.com
```

```
<script>attack</script>
```

CSP Ecosystem

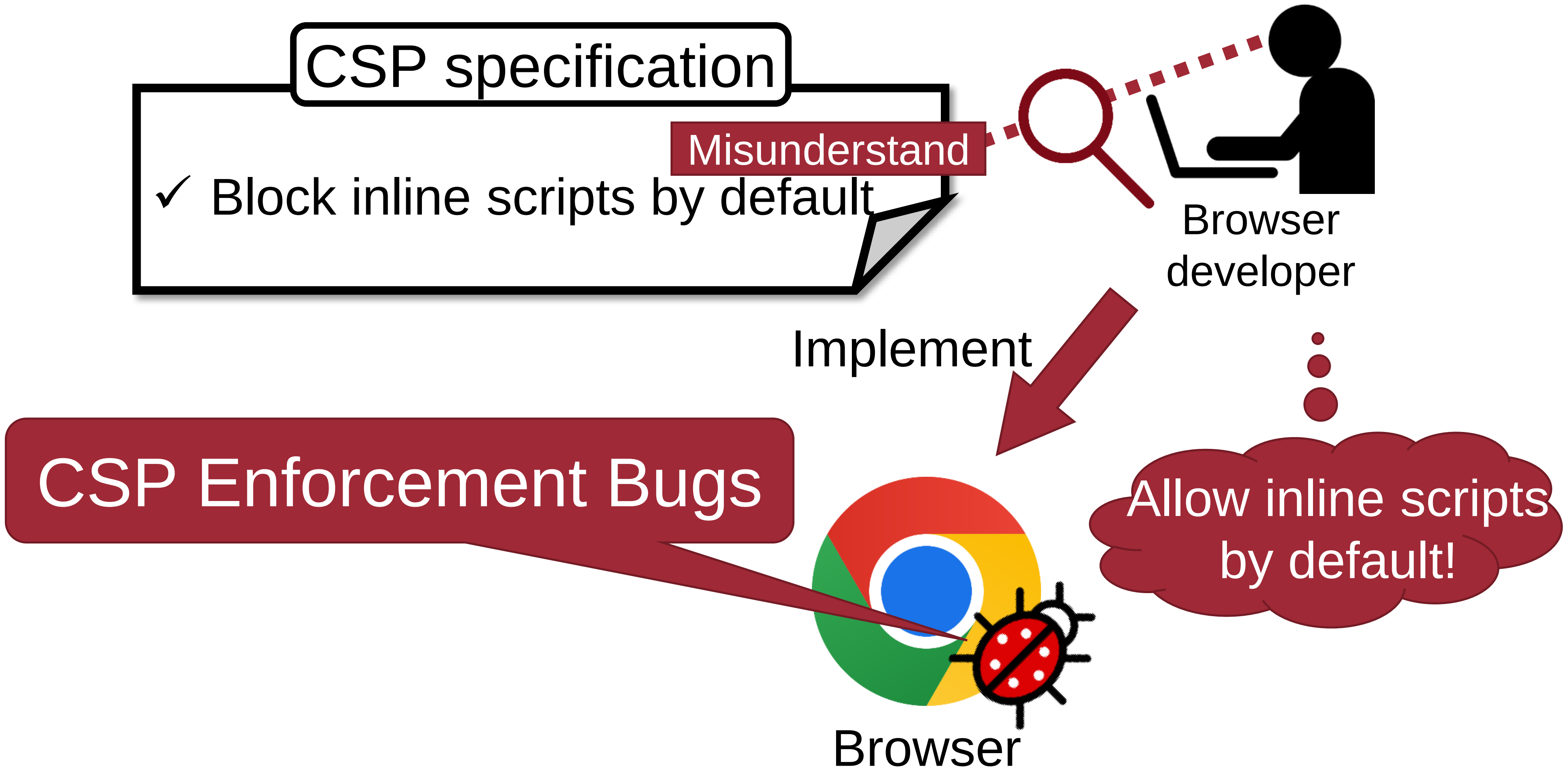


Our Research Question

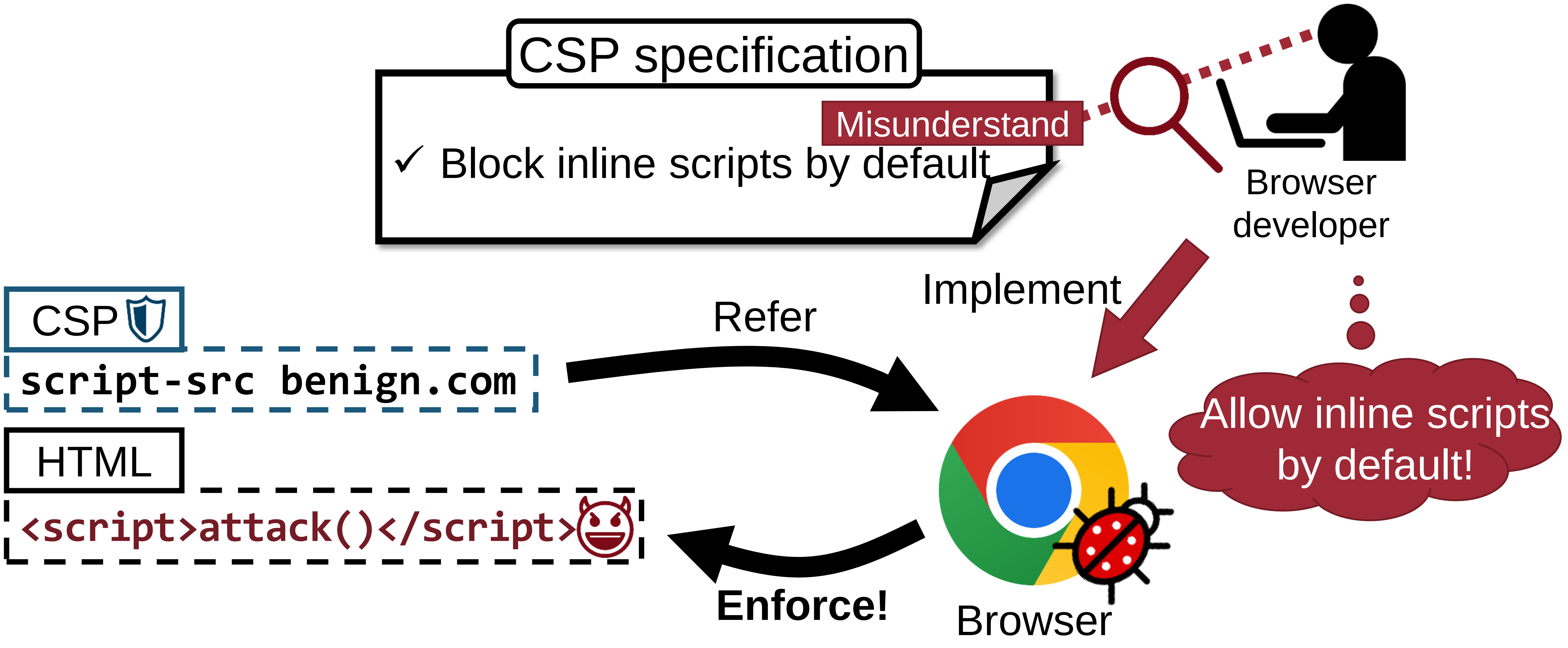


What if developers **misunderstand or misimplement the CSP specification**?

CSP Enforcement Bugs

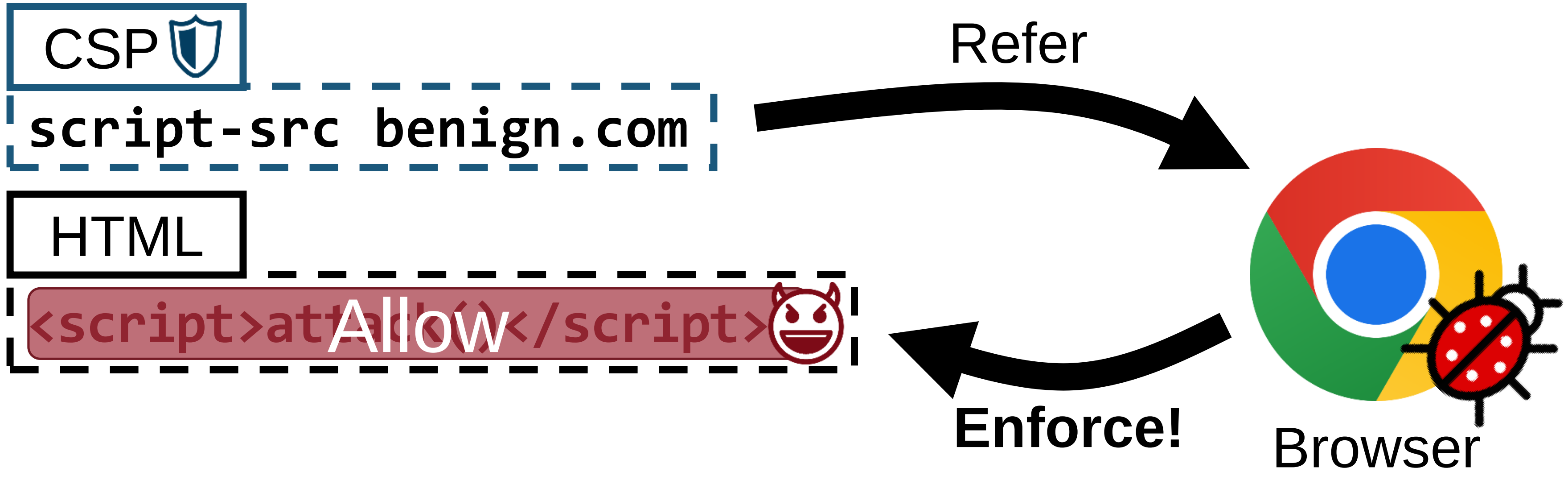


CSP Enforcement Bugs

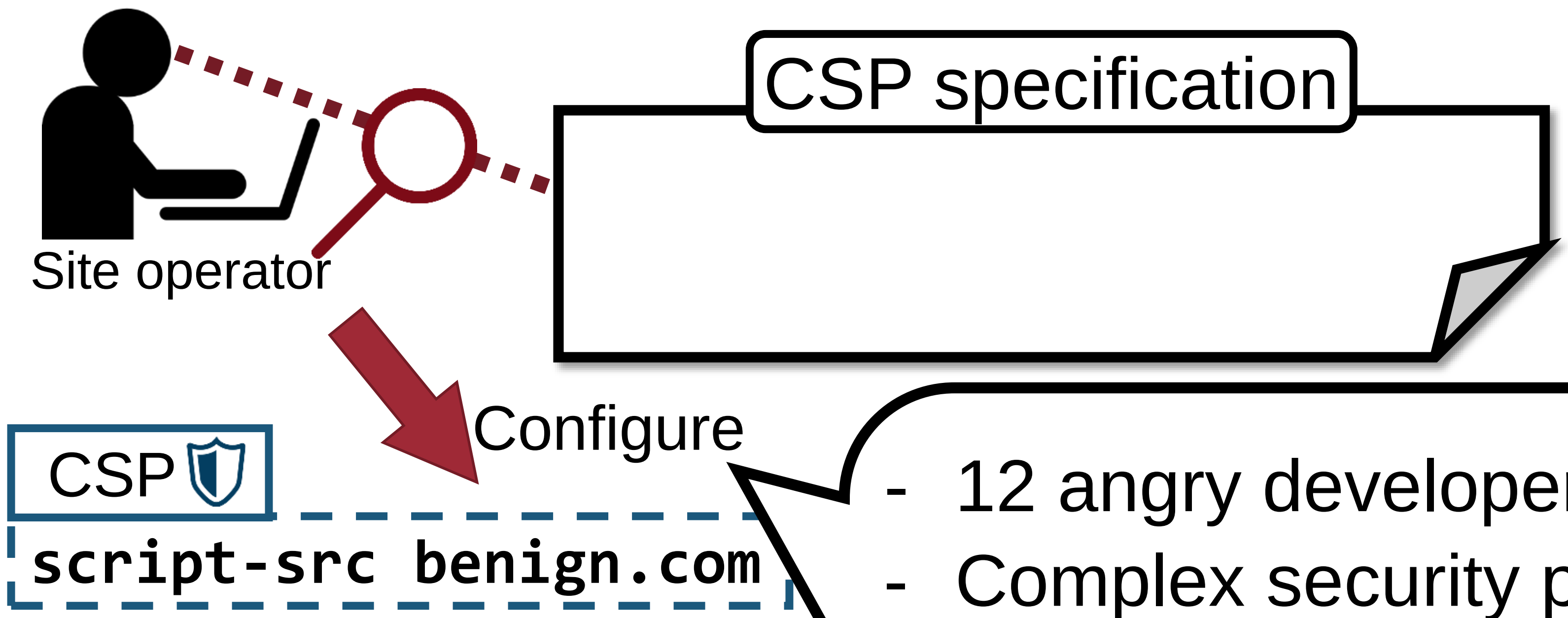


CSP Enforcement Bugs

Allow adversaries to
bypass CSPs and **execute adversarial JS snippets**

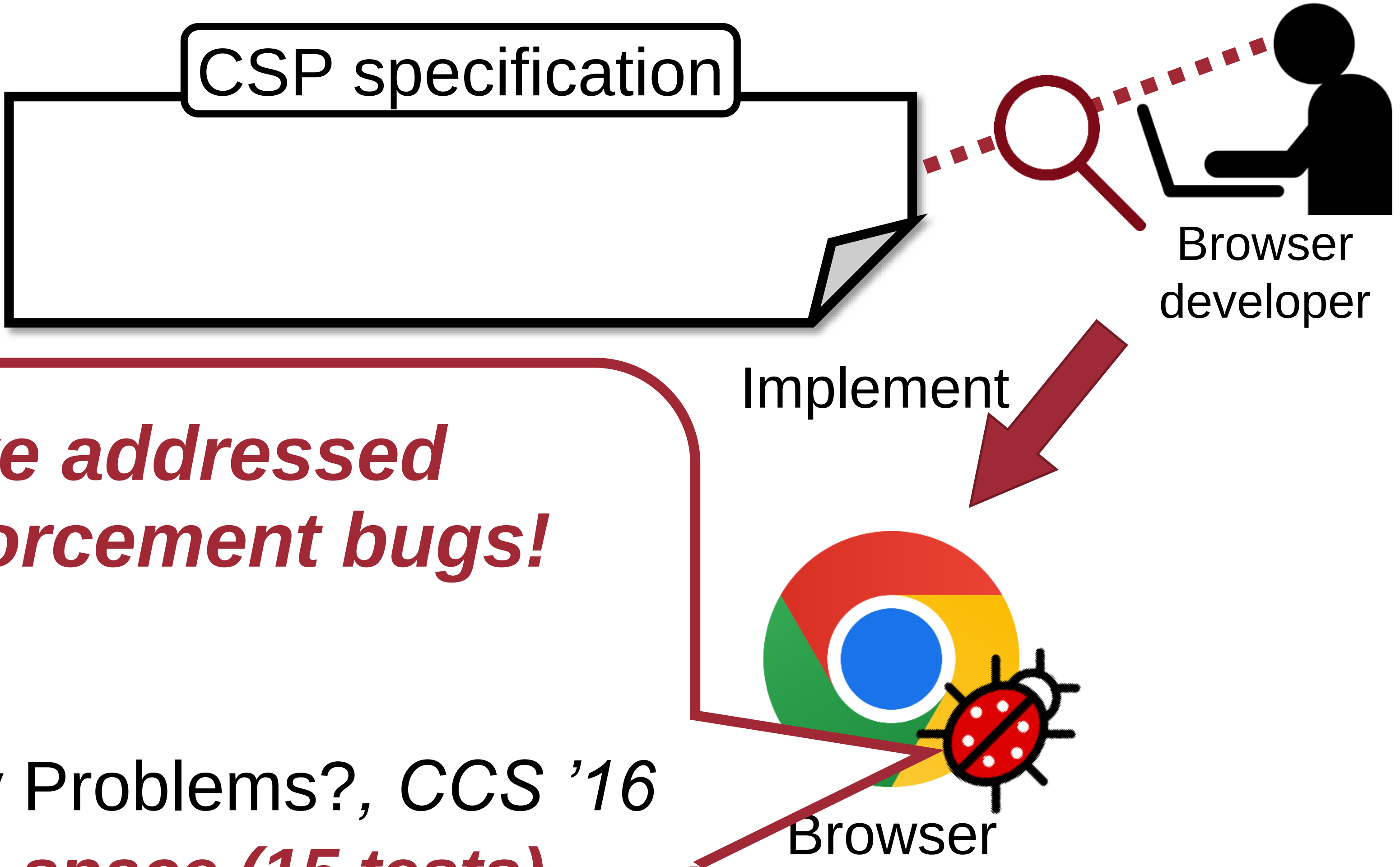


Recent Studies – Insecure CSP Deployment



- 12 angry developers, *CCS ’21*
- Complex security policy?, *NDSS ’20*
- CSP is dead, long live CSP!, *CCS ’16*
- Reining in the web with CSP, *WWW ’10*
- CCSP, *USENIX Security ’17*
- CSPAutoGen, *CCS ’16*

Recent Studies – CSP Enforcement Bugs



Few studies have addressed finding CSP enforcement bugs!

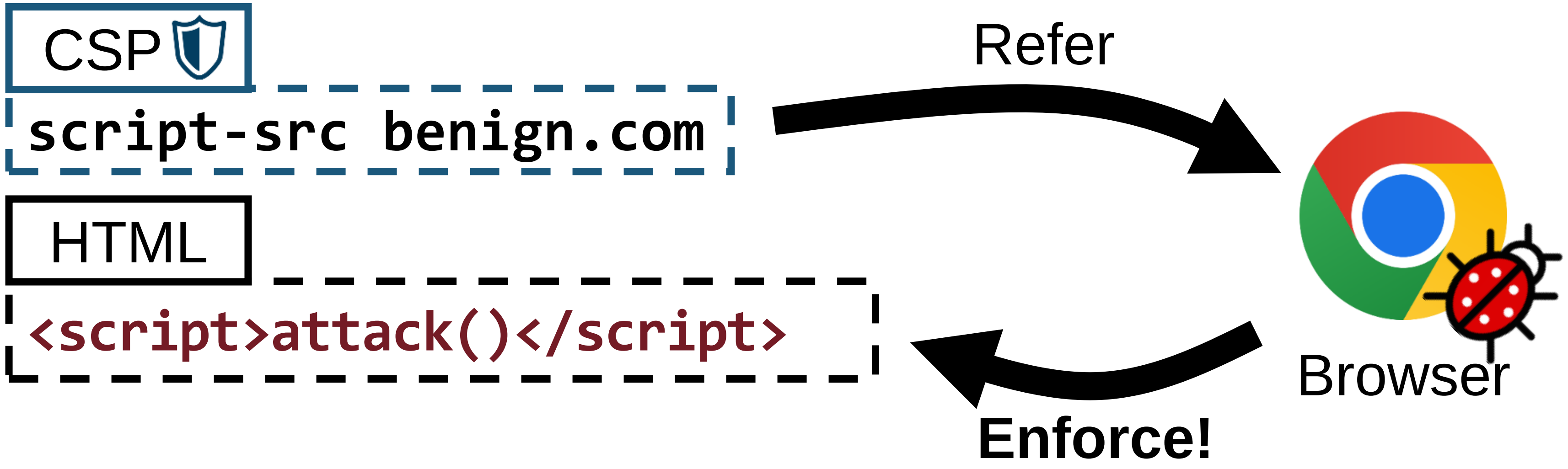
- Content Security Problems?, *CCS '16*

*→ **Limited search space (15 tests)***

Our Goal:

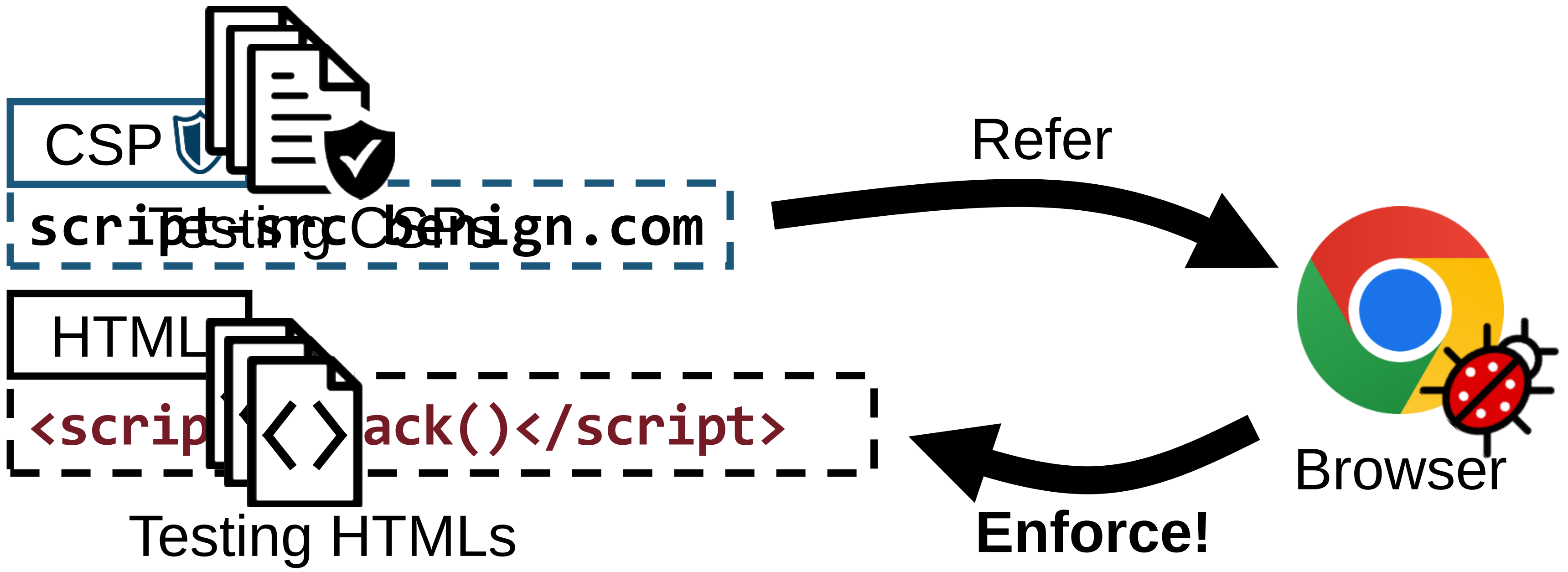
Finding CSP enforcement bugs <u>regarding JS execution</u>

Challenges in Finding Bugs



Challenges in Finding Bugs

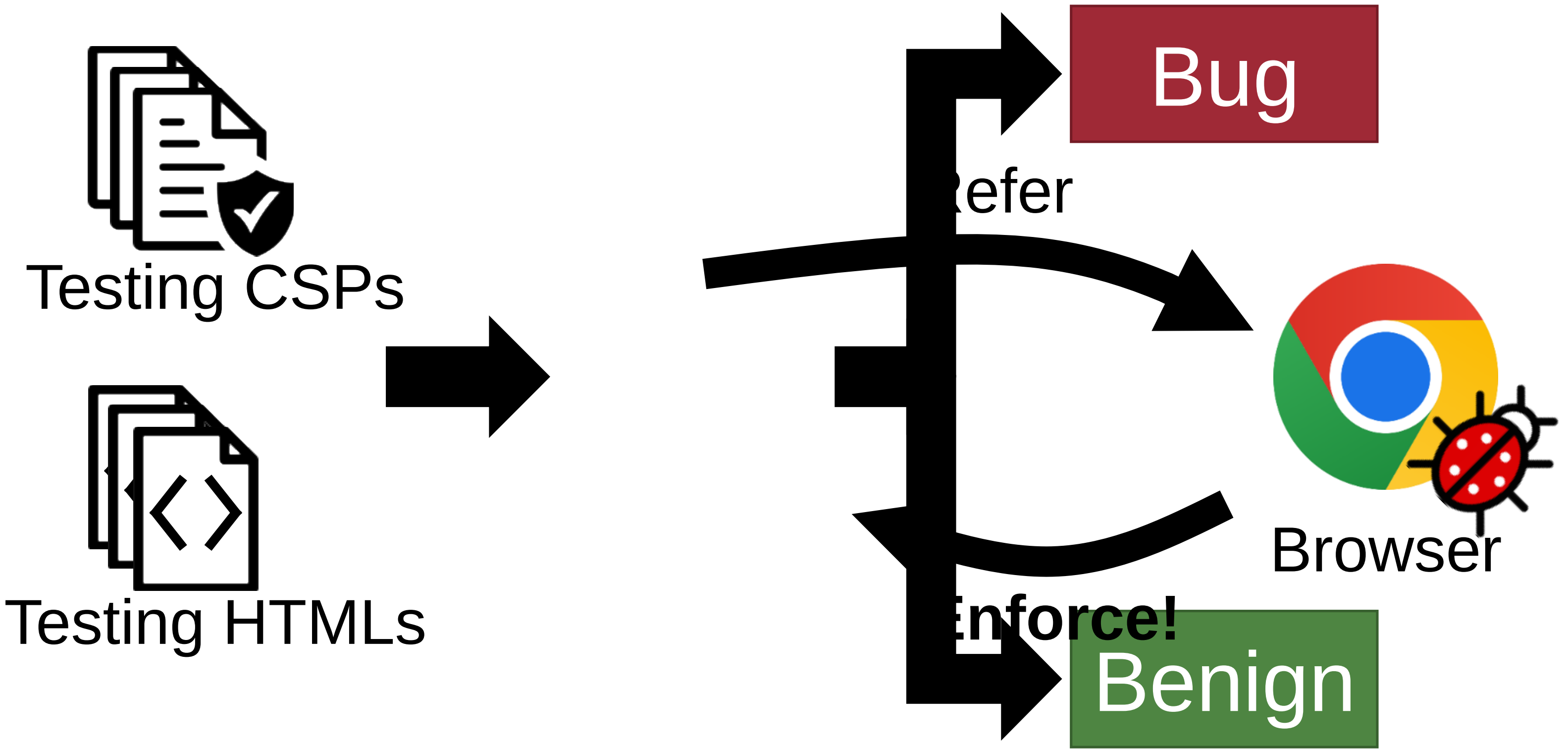
Generating diverse inputs



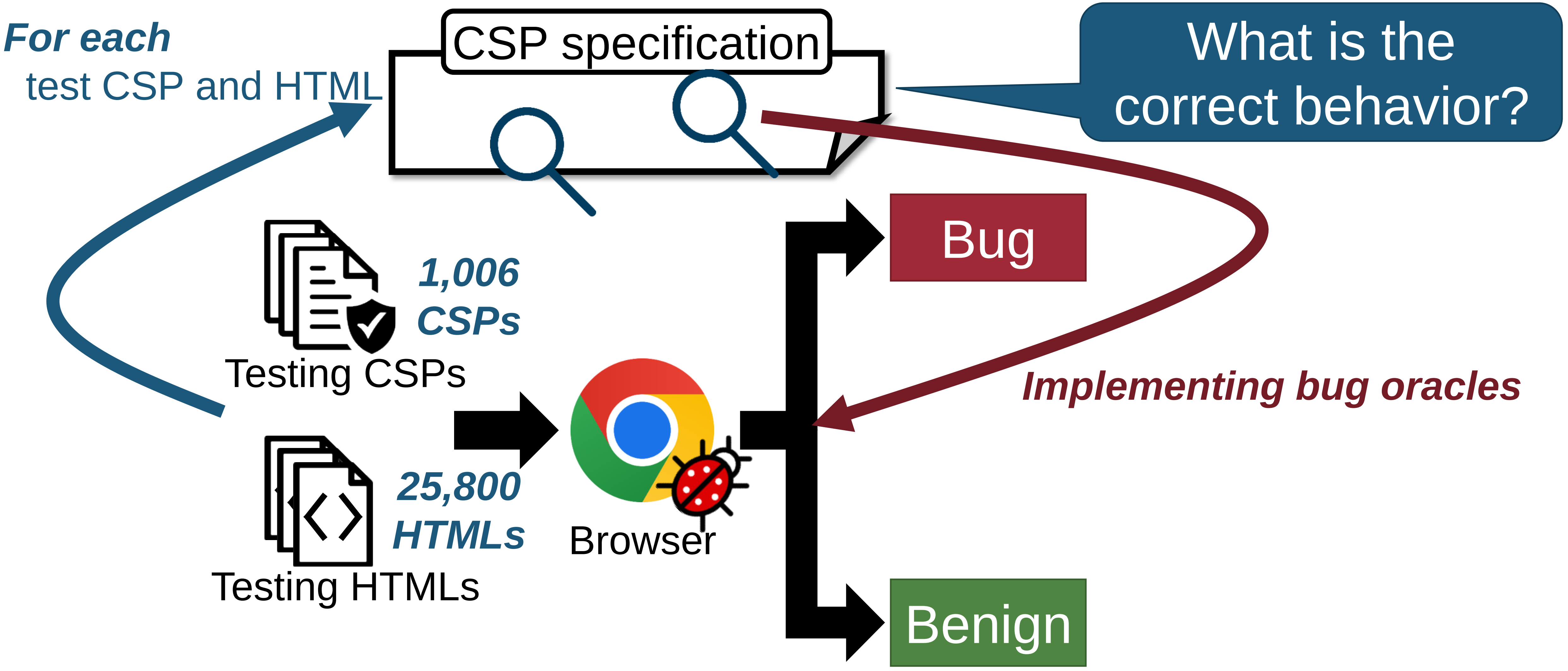
Challenges in Finding Bugs

Generating diverse inputs

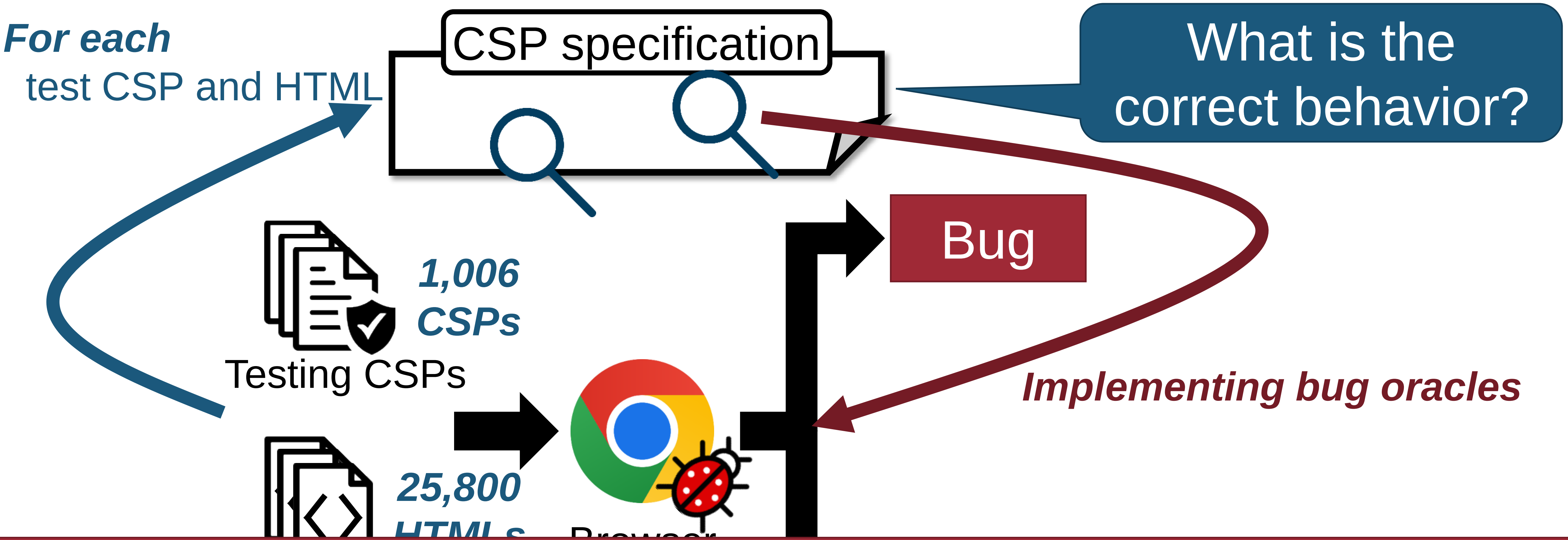
Implementing bug oracles



Challenge: Implementing Bug Oracles



Challenge: Implementing Bug Oracles



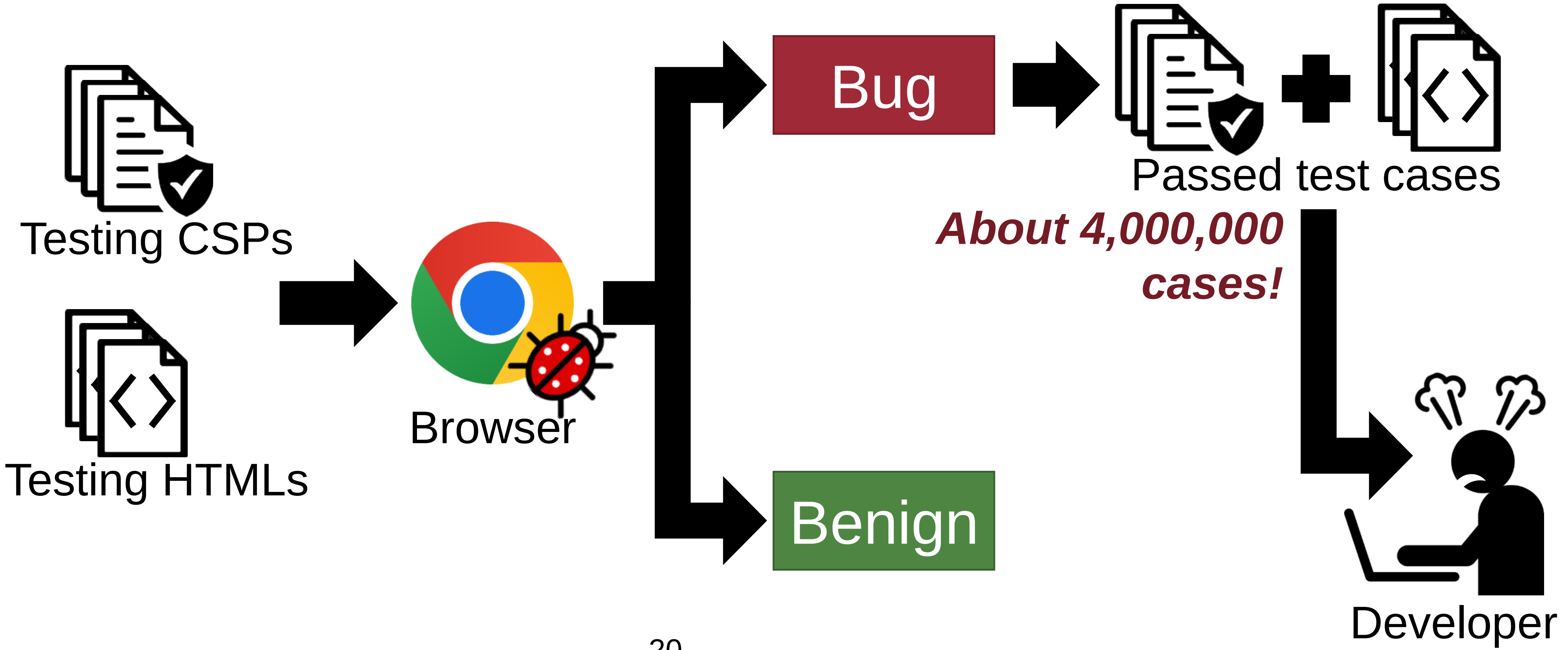
The **manual identification of correct behaviors** is not scalable

Challenges in Finding Bugs

Generating diverse inputs

Implementing bug oracles

Identifying root causes



How do we address all the challenges?

We propose

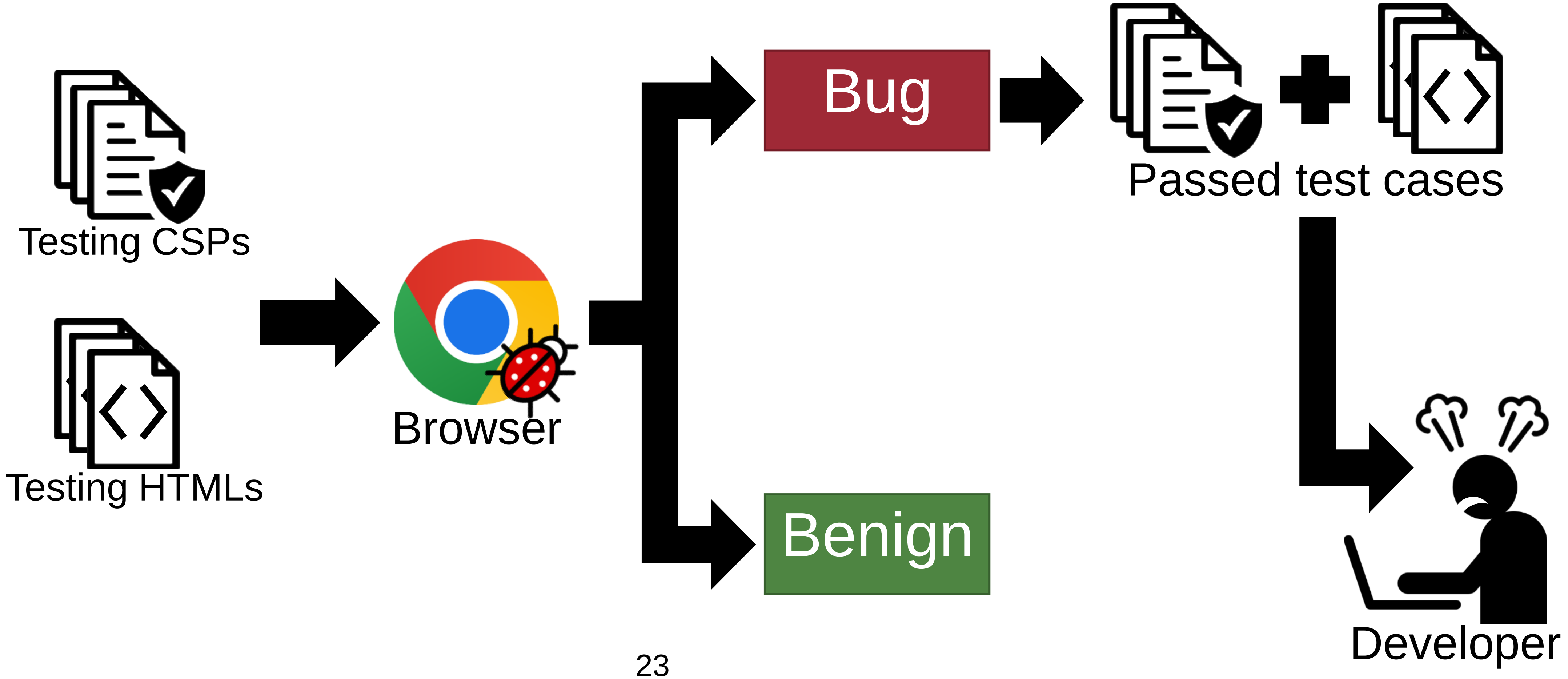
DiffCSP

Our Goal: Finding CSP Enforcement Bugs

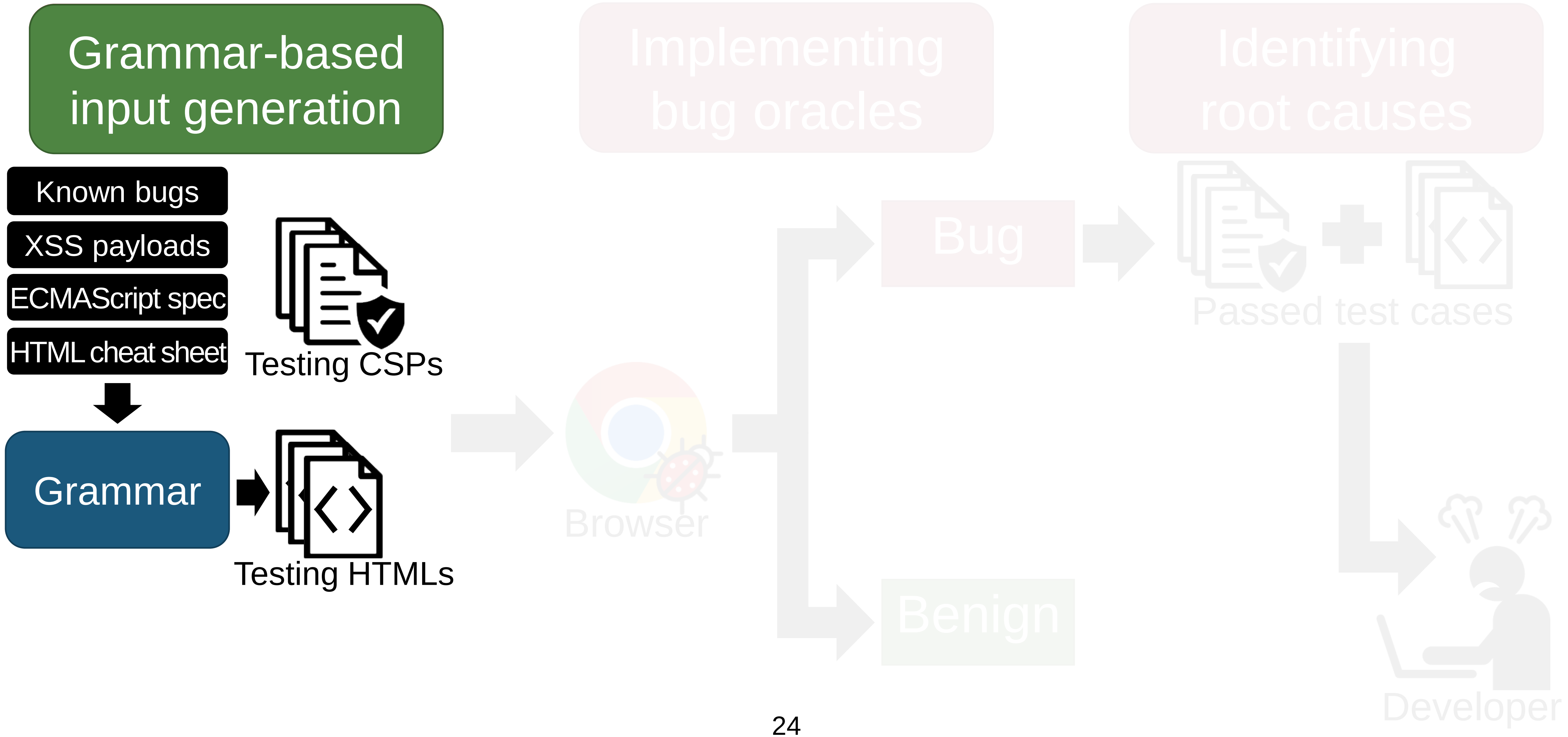
Generating diverse inputs

Implementing bug oracles

Identifying root causes



Generating Diverse Inputs in DiffCSP



Grammar-based Input Generation

Known CSP bugs

XSS payloads

ECMAScript spec

HTML cheat sheet

Diverse ways of executing JS snippets

Grammar-based Input Generation

Known CSP bugs

```
document.body.innerHTML+=
  “<script>
     eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

Diverse ways of executing JS snippets

Derive all known forms of executing JS codes

Testing HTMLs

Grammar-based Input Generation

Known CSP bugs

```
document.body.innerHTML+=
  "<script>
     eval('attack()')
   </script>"
```

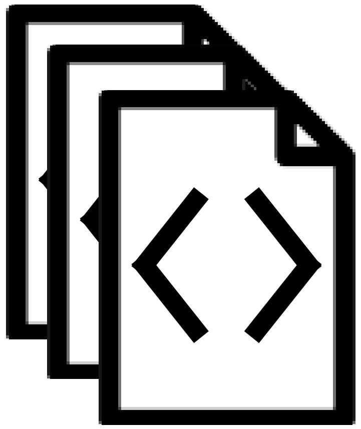
HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

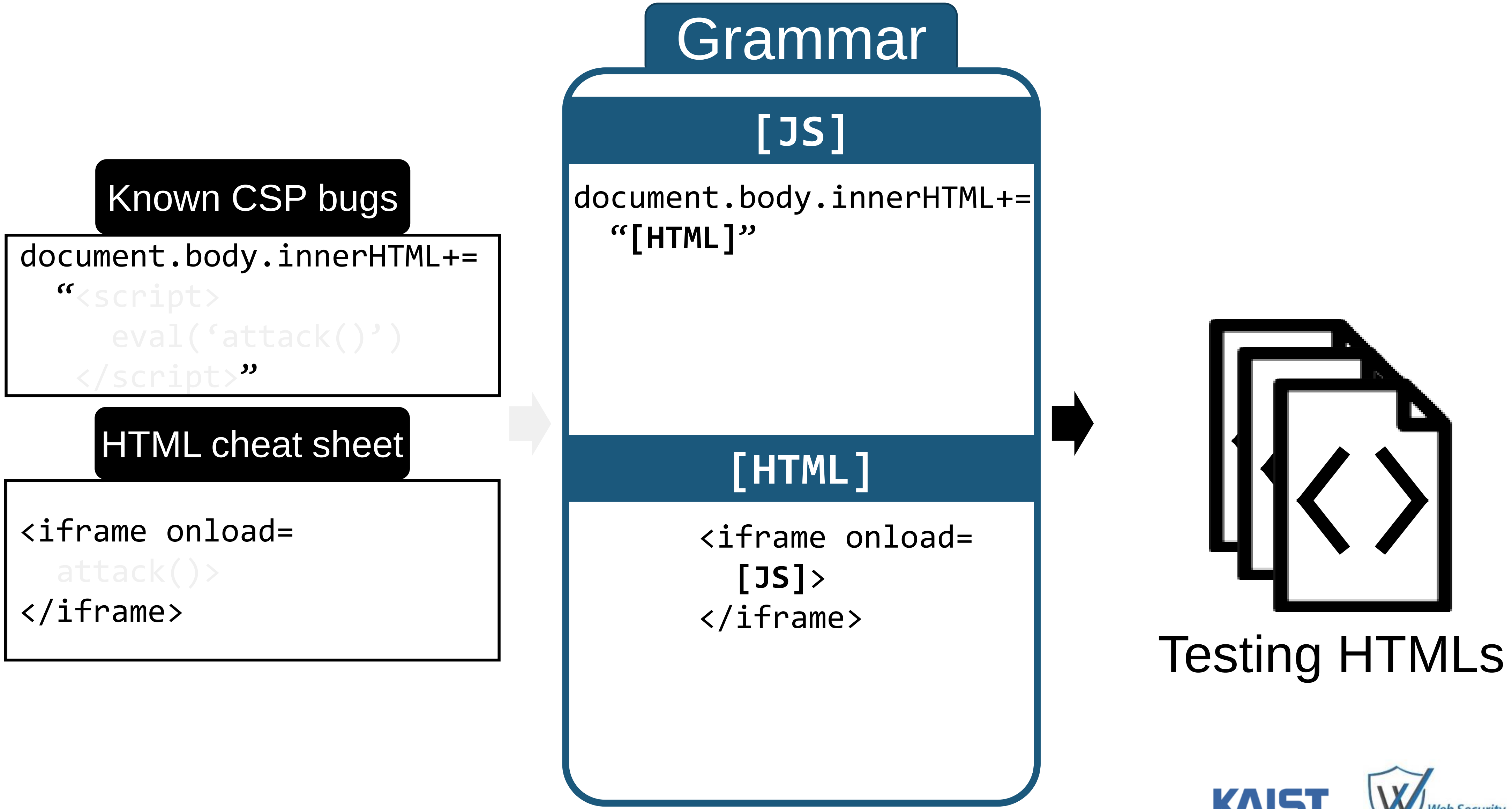
```
document.body.innerHTML+=
  "[HTML]"
```



Testing HTMLs

KAIST Web Security & Privacy Lab CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Grammar-based Input Generation



Known CSP bugs

```
document.body.innerHTML+=
  “<script>
     eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

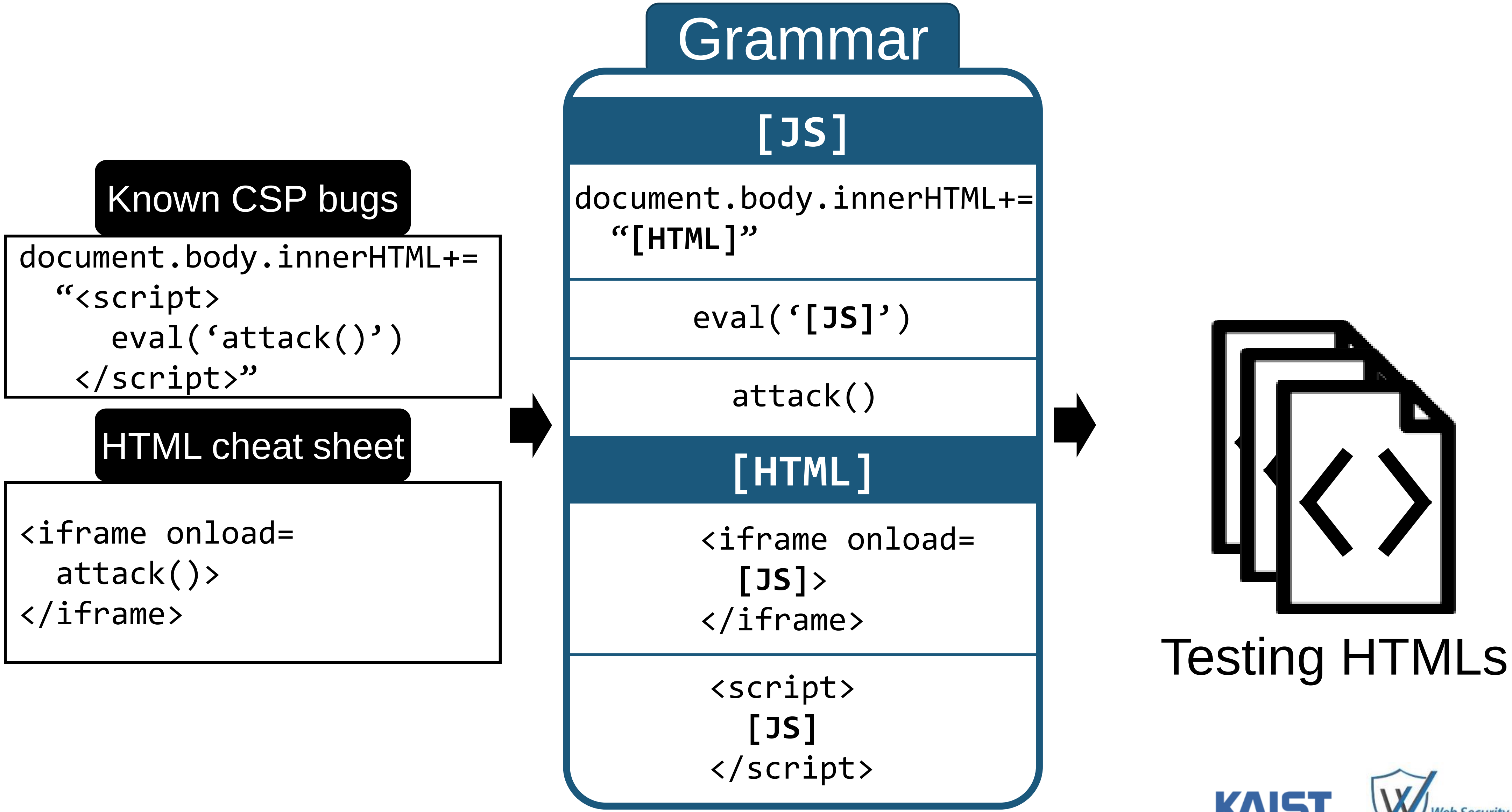
```
document.body.innerHTML+=
  “[HTML]”
```

[HTML]

```
<iframe onload=
  [JS]>
</iframe>
```

Testing HTMLs

Grammar-based Input Generation



Known CSP bugs

```
document.body.innerHTML+=
  “<script>
      eval(‘attack()’)
    </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

```
document.body.innerHTML+=
  “[HTML]”
```

```
eval(‘[JS]’)
```

```
attack()
```

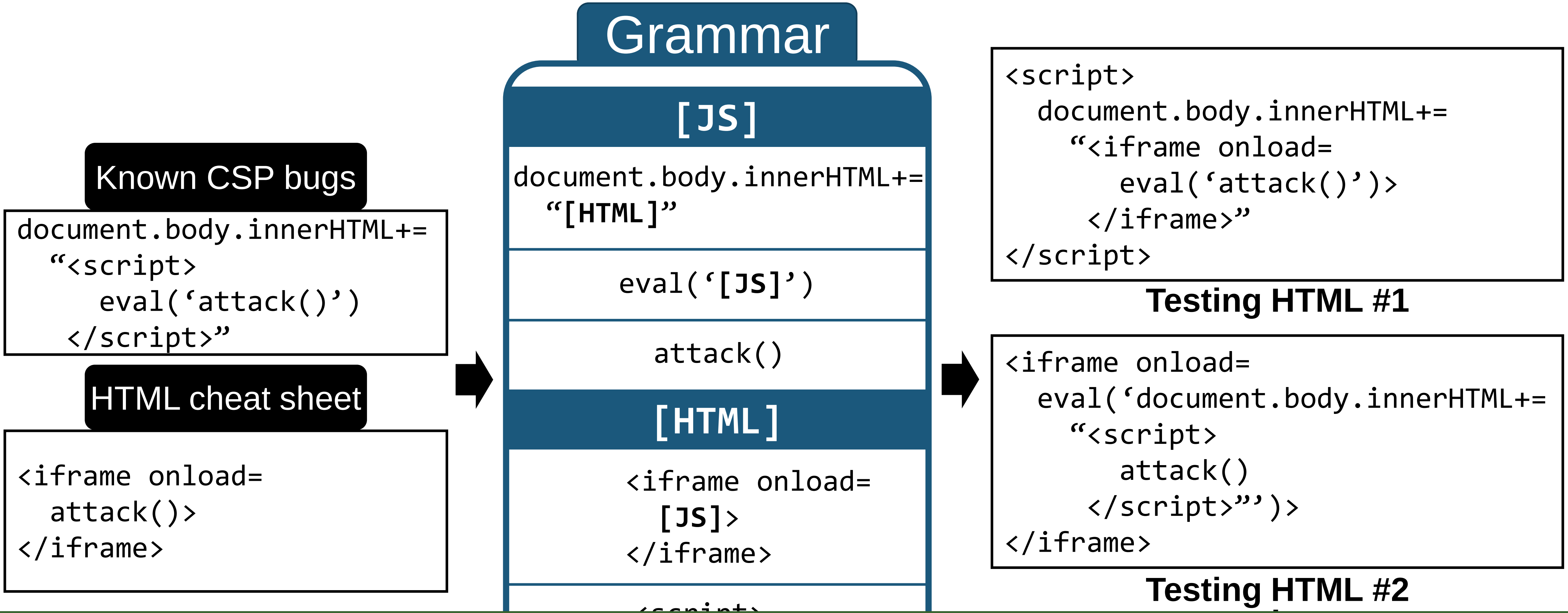
[HTML]

```
<iframe onload=
  [JS]>
</iframe>
```

```
<script>
  [JS]
</script>
```

Testing HTMLs

Grammar-based Input Generation



Known CSP bugs

```
document.body.innerHTML+=
  “<script>
     eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

```
document.body.innerHTML+=
  “[HTML]”
```

```
eval(‘[JS]’)
```

```
attack()
```

[HTML]

```
<iframe onload=
  [JS]>
</iframe>
```

```
<script>
```

```
<script>
  document.body.innerHTML+=
    “<iframe onload=
       eval(‘attack()’)>
     </iframe>”
</script>
```

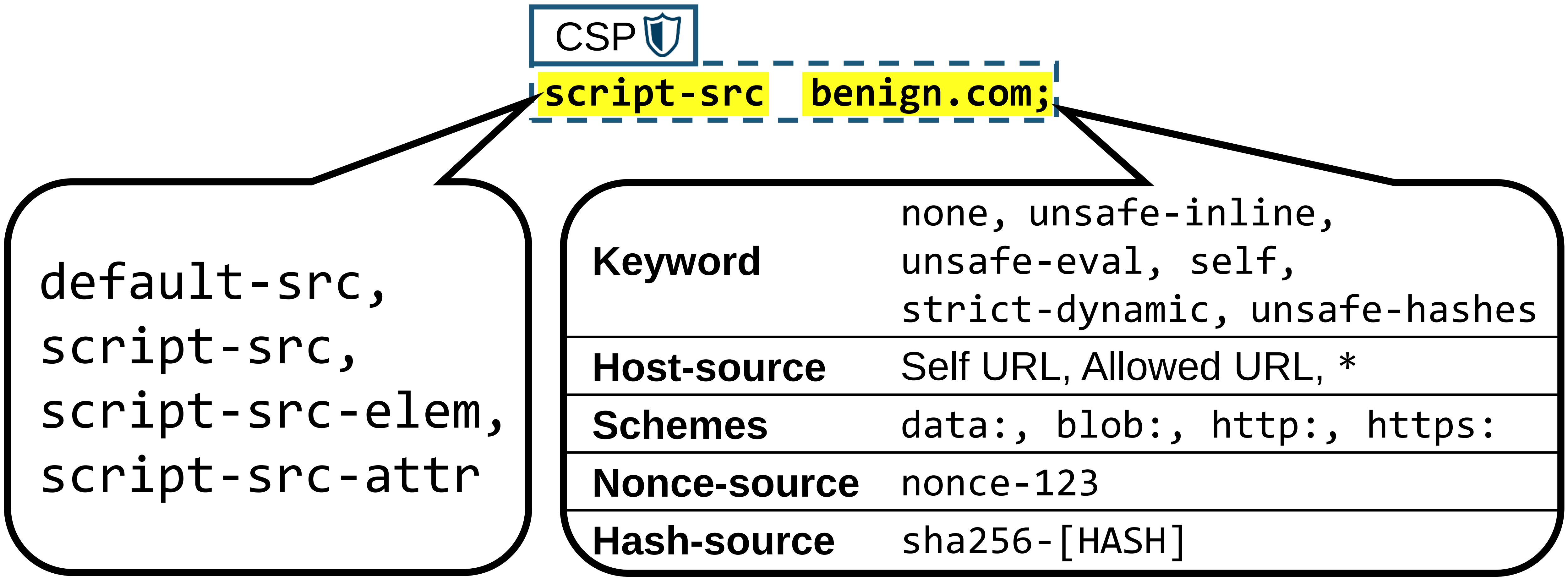
Testing HTML #1

```
<iframe onload=
  eval(‘document.body.innerHTML+=
    “<script>
       attack()
     </script>”’)>
</iframe>
```

Testing HTML #2

Generate **25,880 HTML instances**

CSP Generation

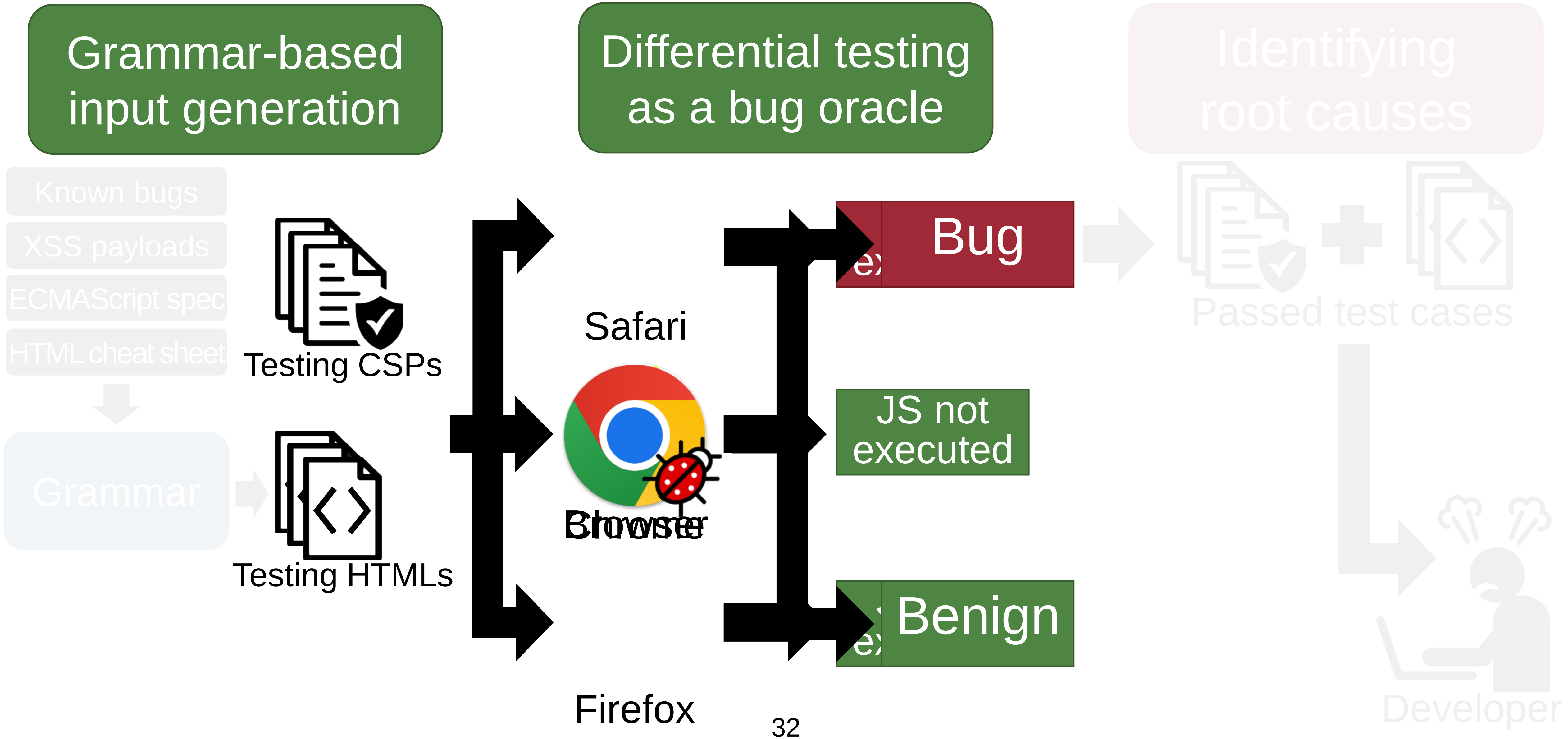


```
default-src,
script-src,
script-src-elem,
script-src-attr
```

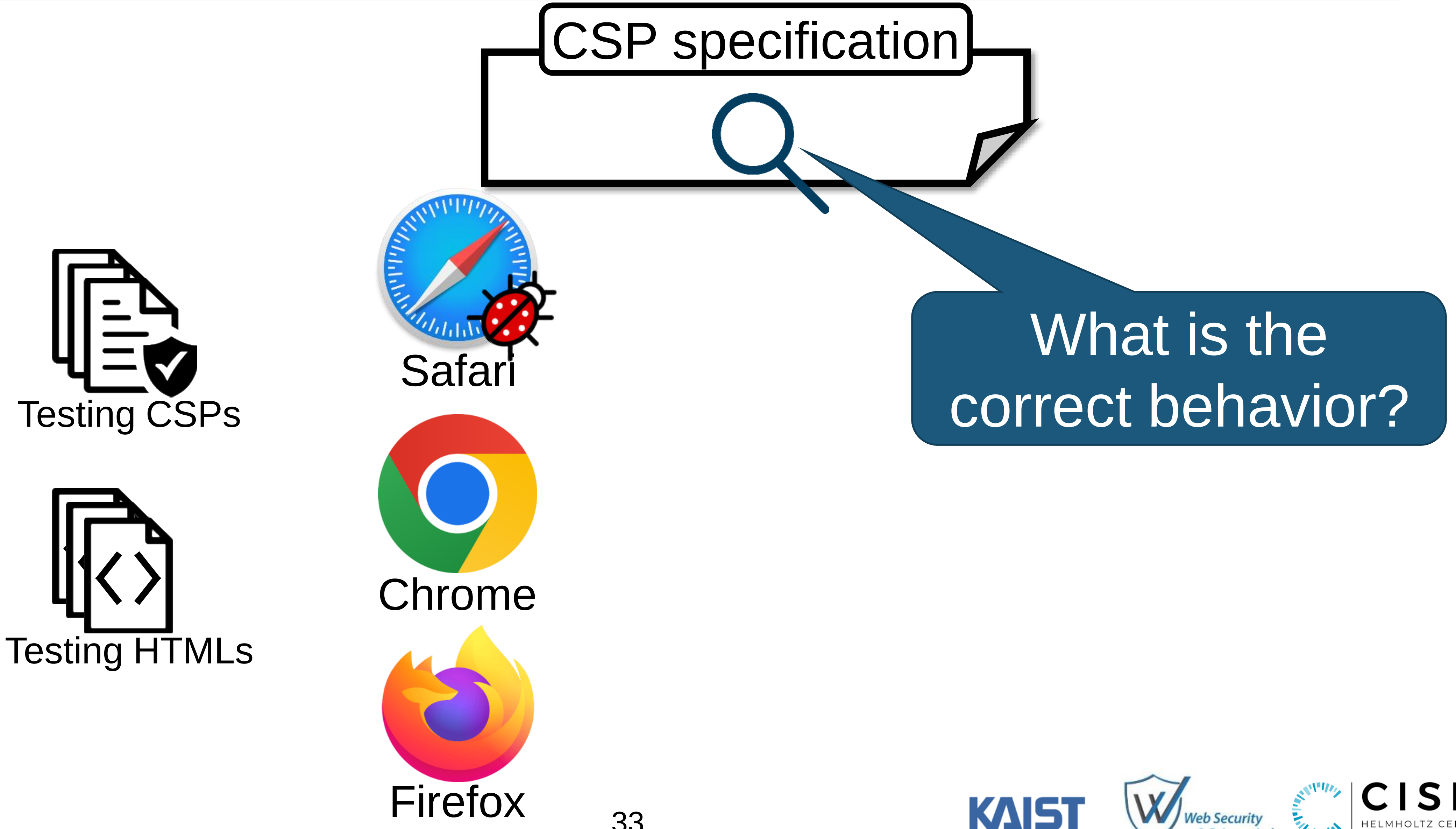
Keyword	`none, unsafe-inline, unsafe-eval, self, strict-dynamic, unsafe-hashes`
Host-source	Self URL, Allowed URL, *
Schemes	`data:, blob:, http:, https:`
Nonce-source	`nonce-123`
Hash-source	`sha256-[HASH]`

Generate **1,006 policies**

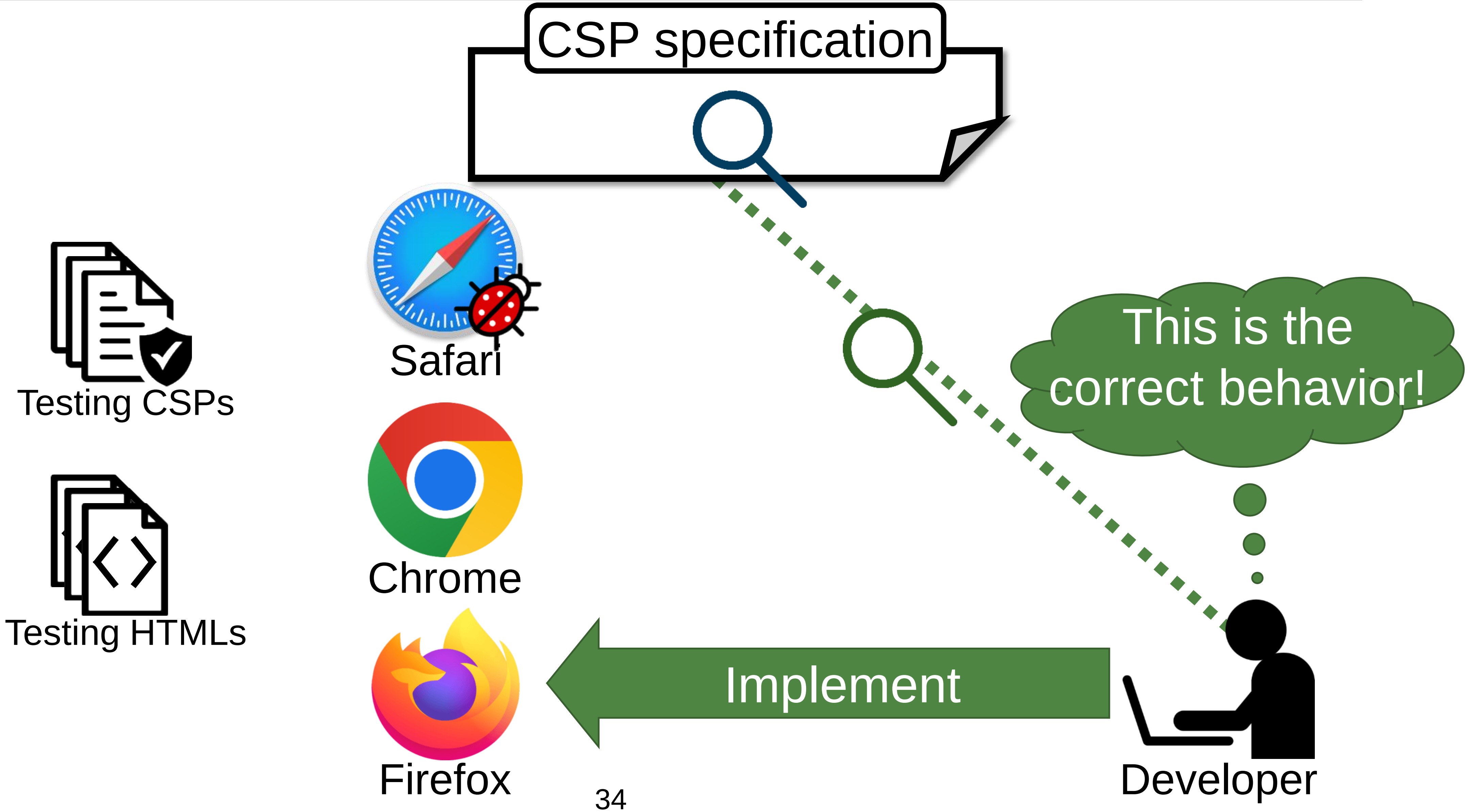
Implementing Bug Oracles in DiffCSP



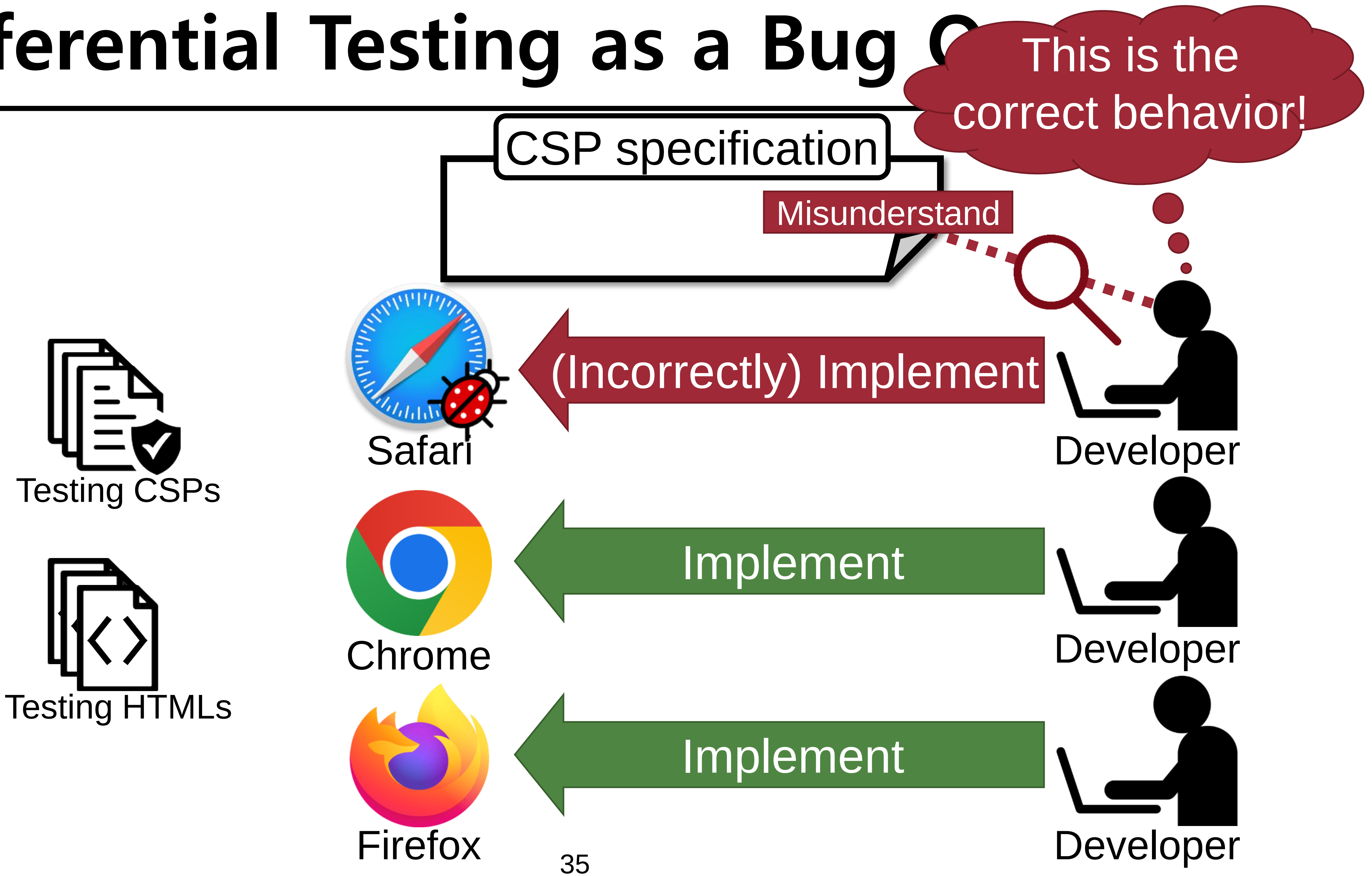
Differential Testing as a Bug Oracle



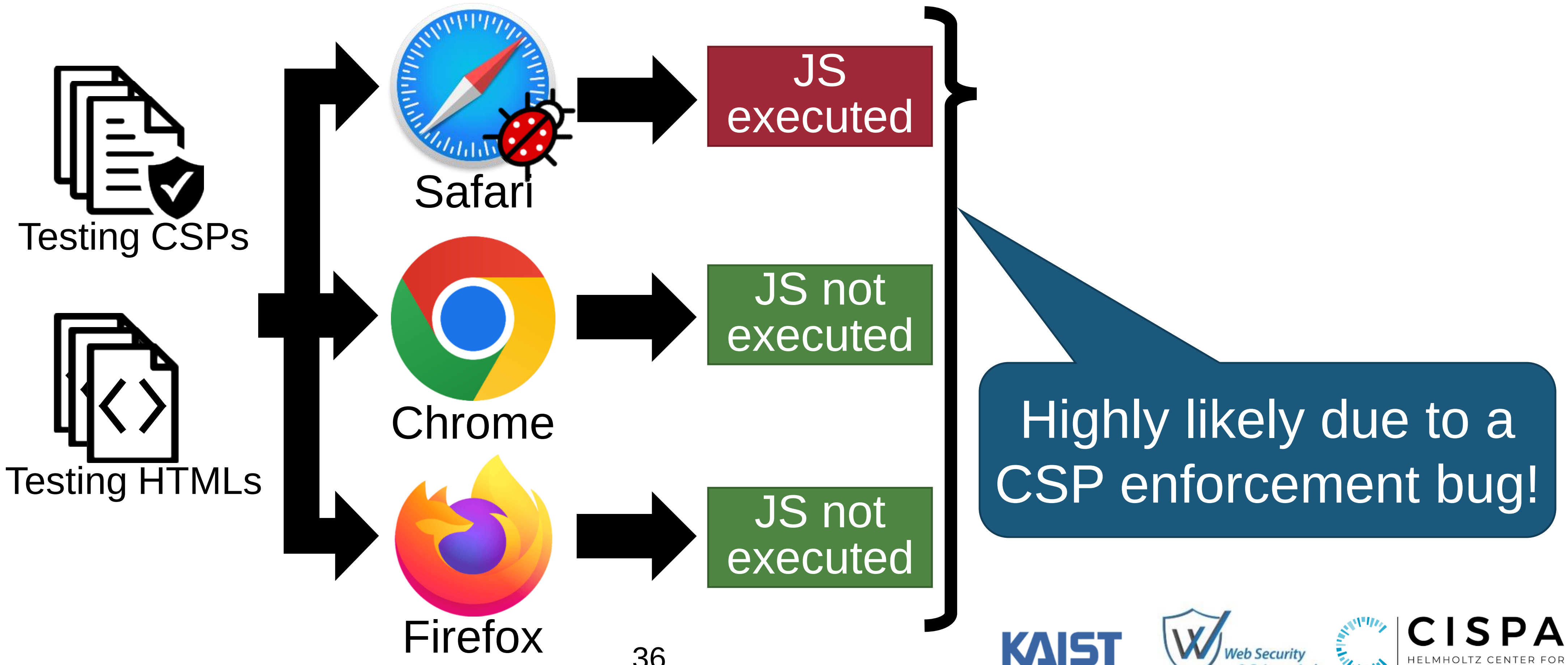
Differential Testing as a Bug Oracle



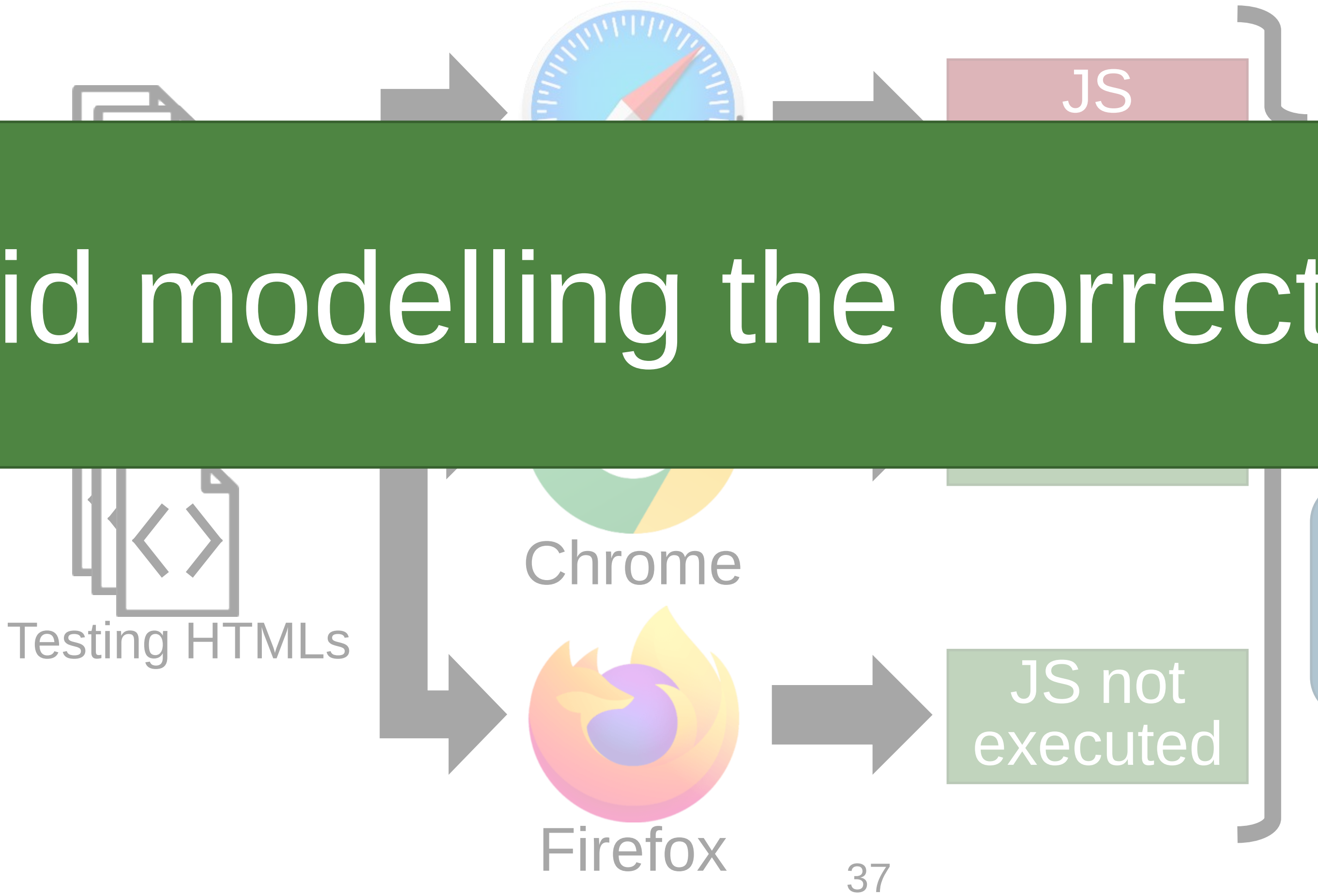
Differential Testing as a Bug O



Differential Testing as a Bug Oracle



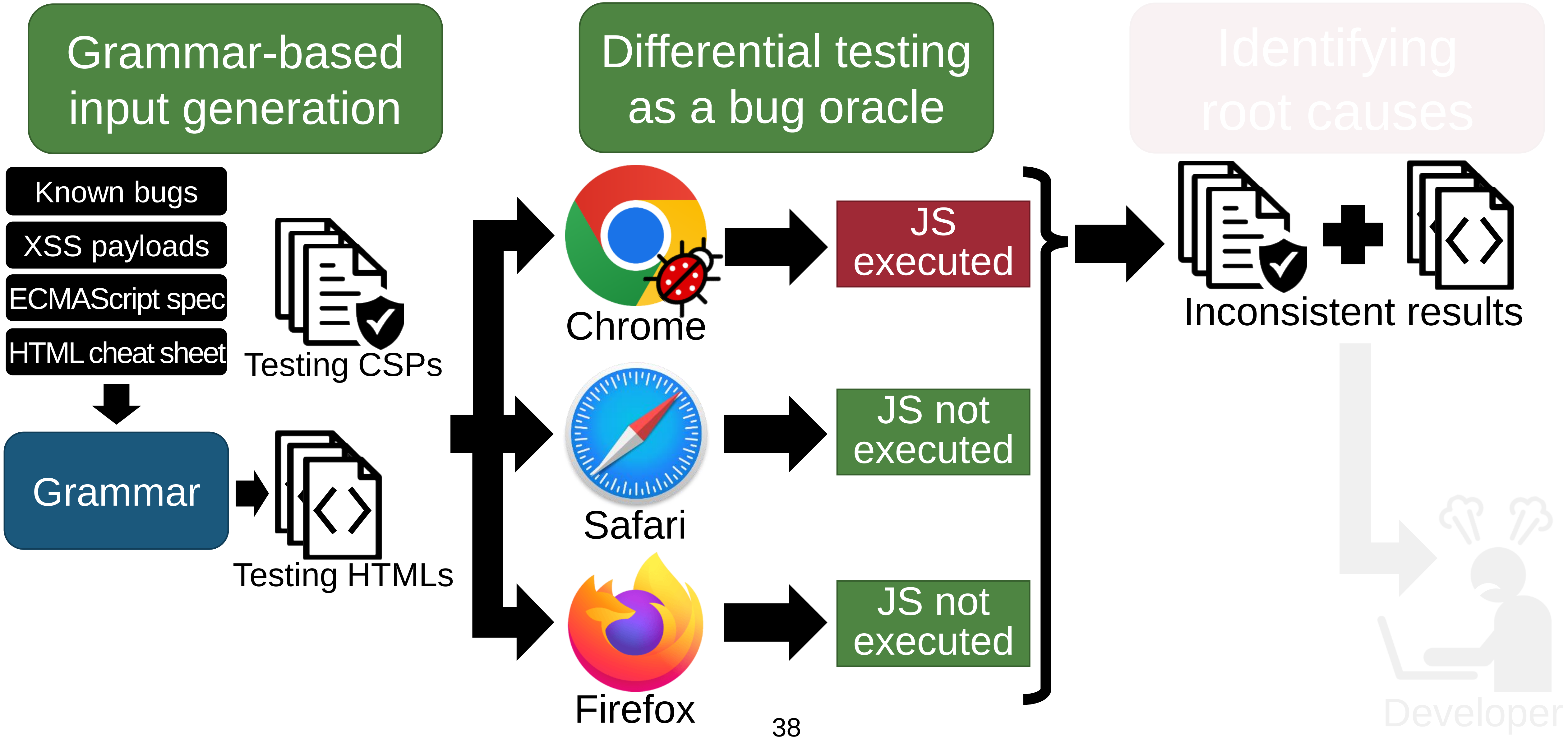
Differential Testing as a Bug Oracle



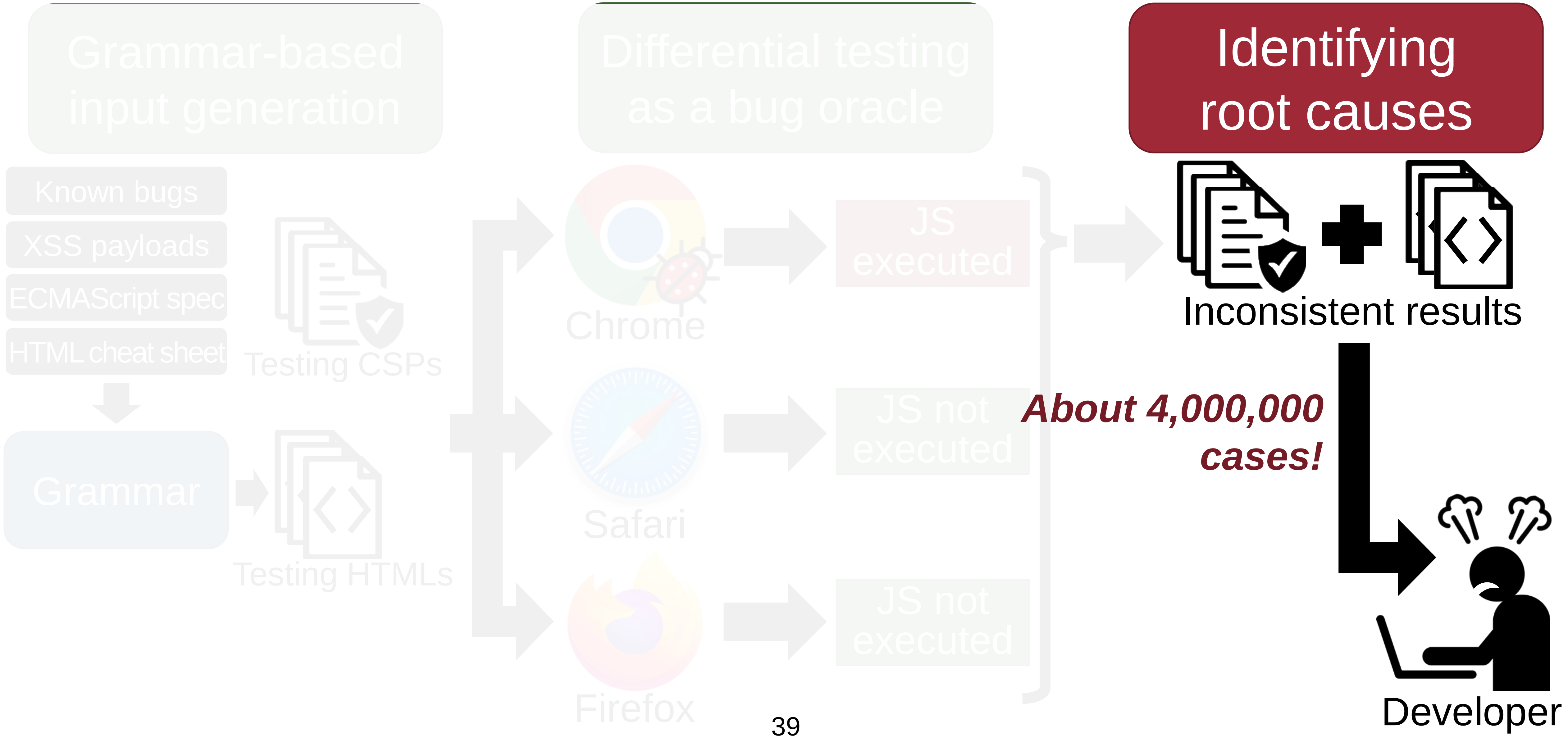
Avoid modelling the correct behaviors!

Highly likely due to a CSP enforcement bug!

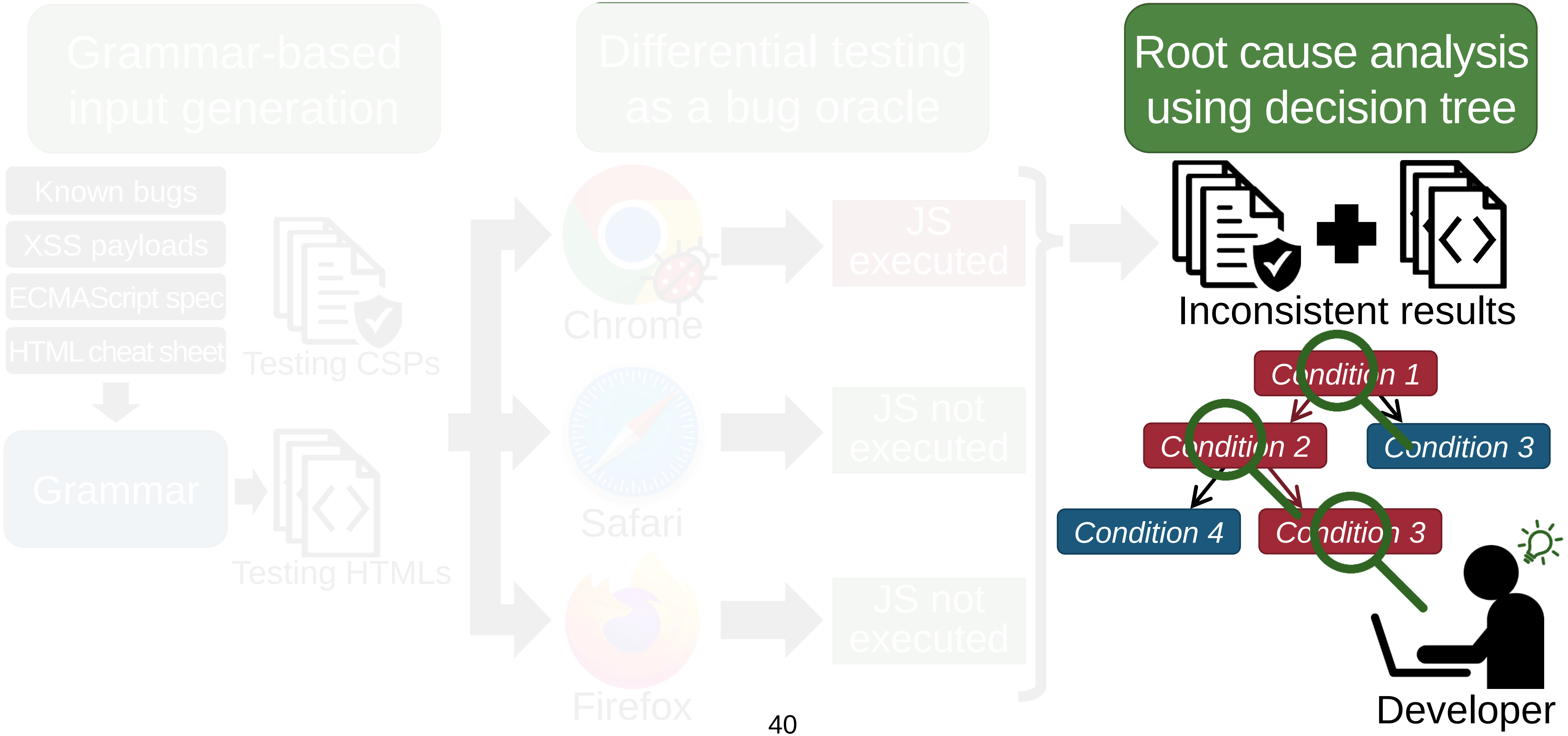
Differential Testing as a Bug Oracle



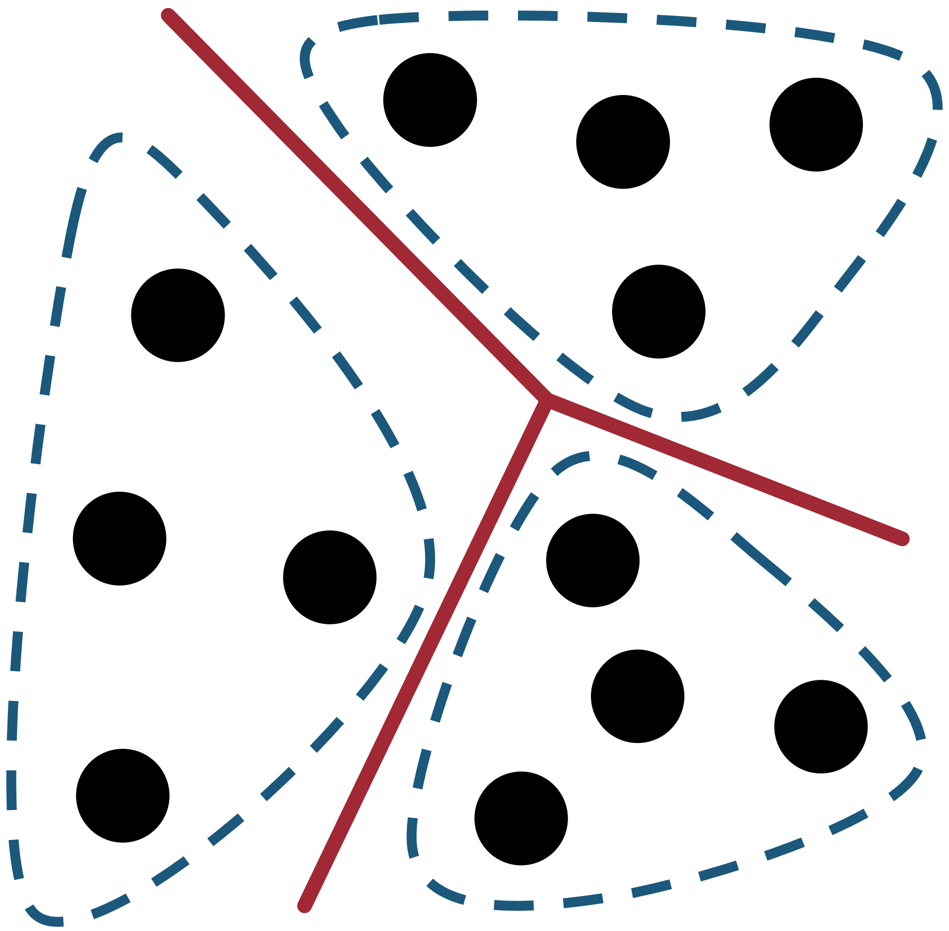
Identifying Root Causes in DiffCSP



Identifying Root Causes in DiffCSP



Decision Tree-based Root Cause Analysis



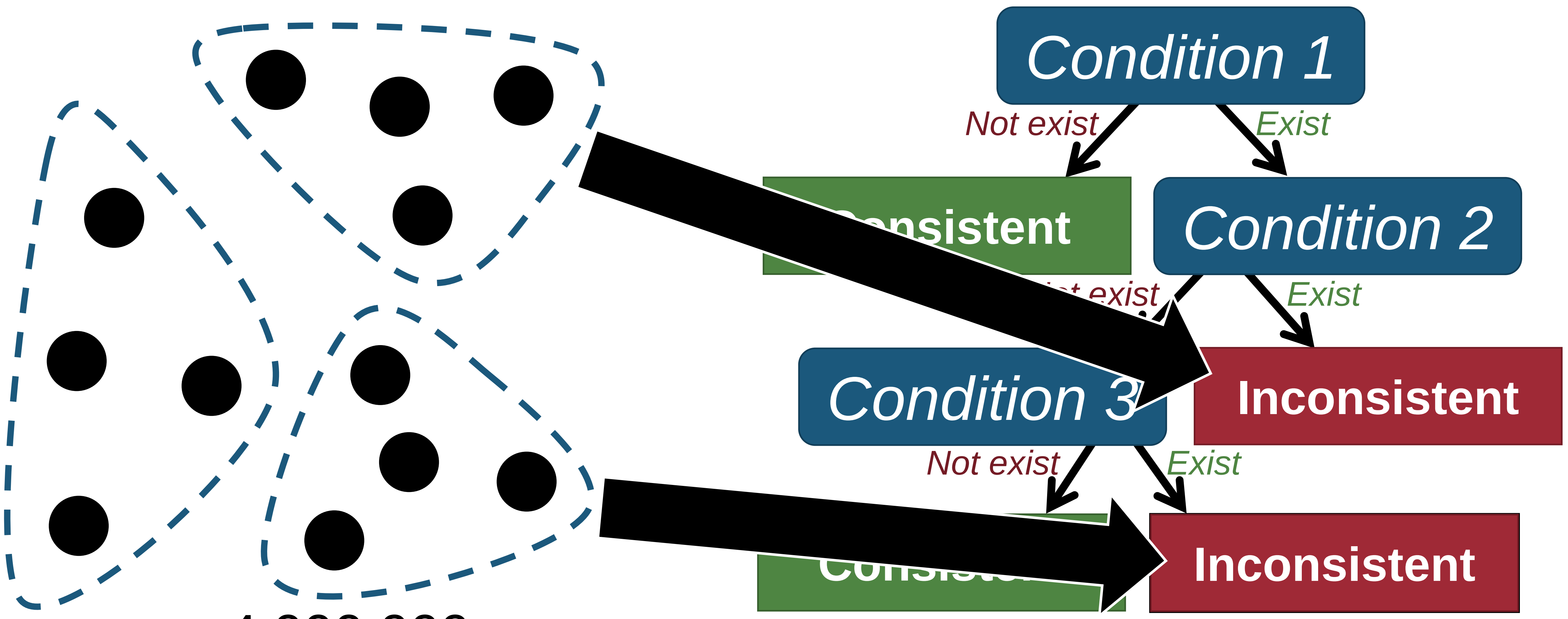
4,000,000
inconsistent results

We need to cluster the results!

Under what conditions are they clustered together?

Decision tree-based root cause analysis

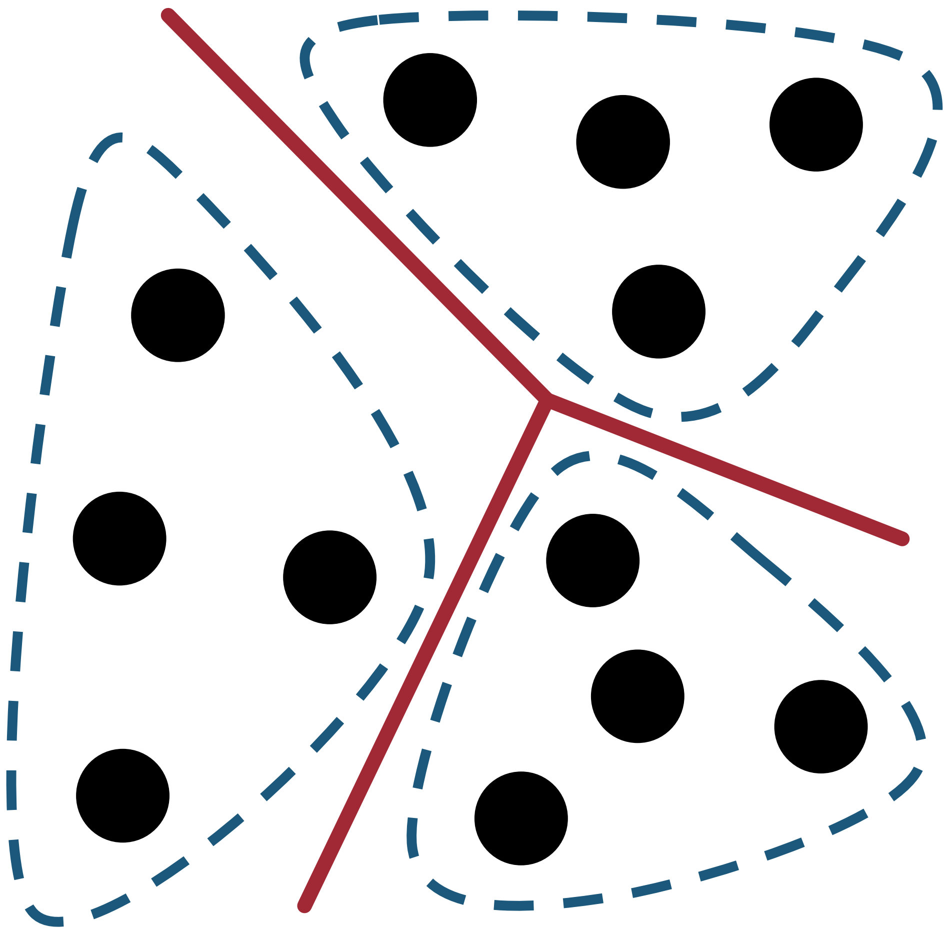
Decision Tree-based Root Cause Analysis



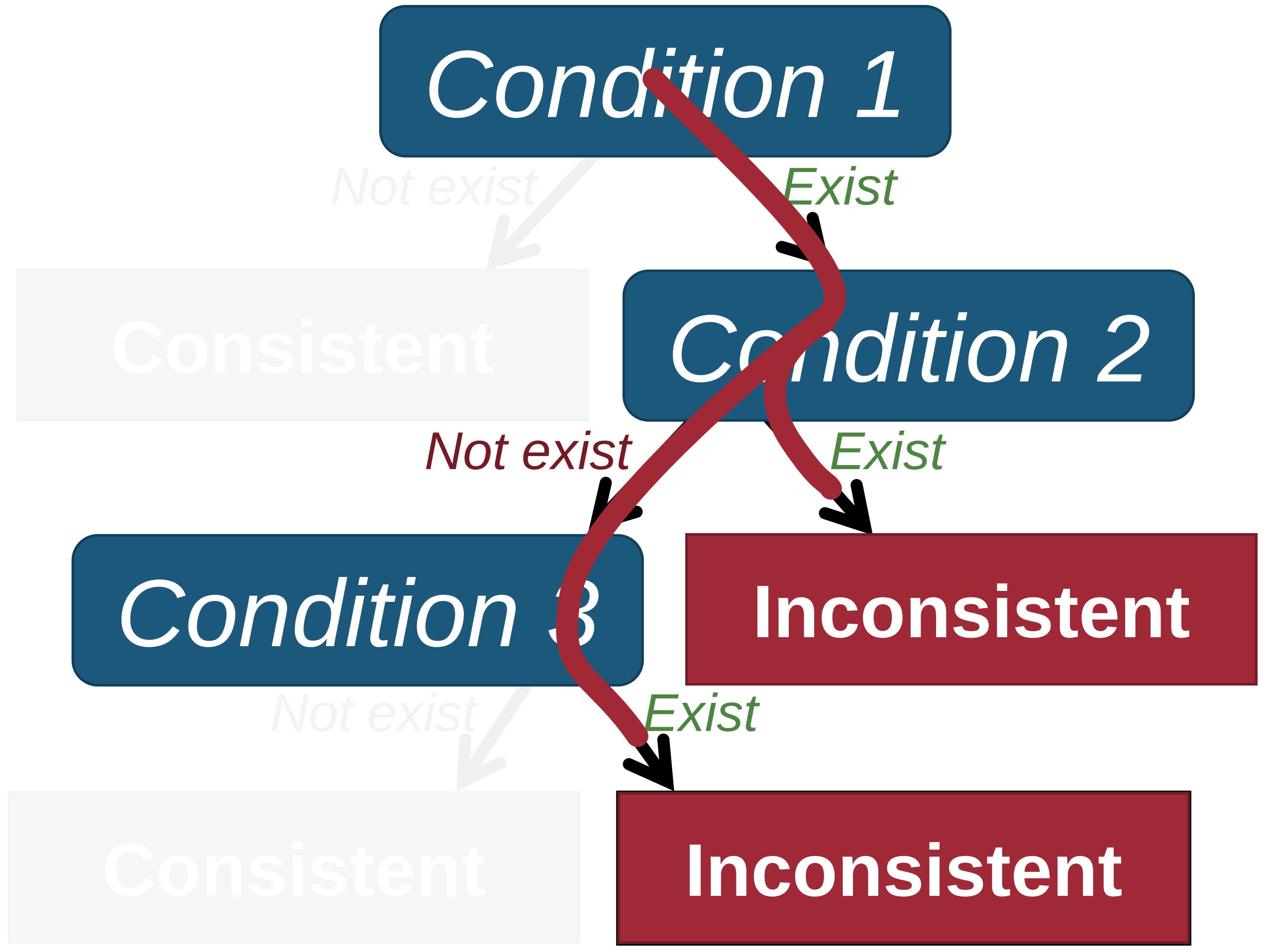
4,000,000 inconsistent results

Decision tree

Decision Tree-based Root Cause Analysis

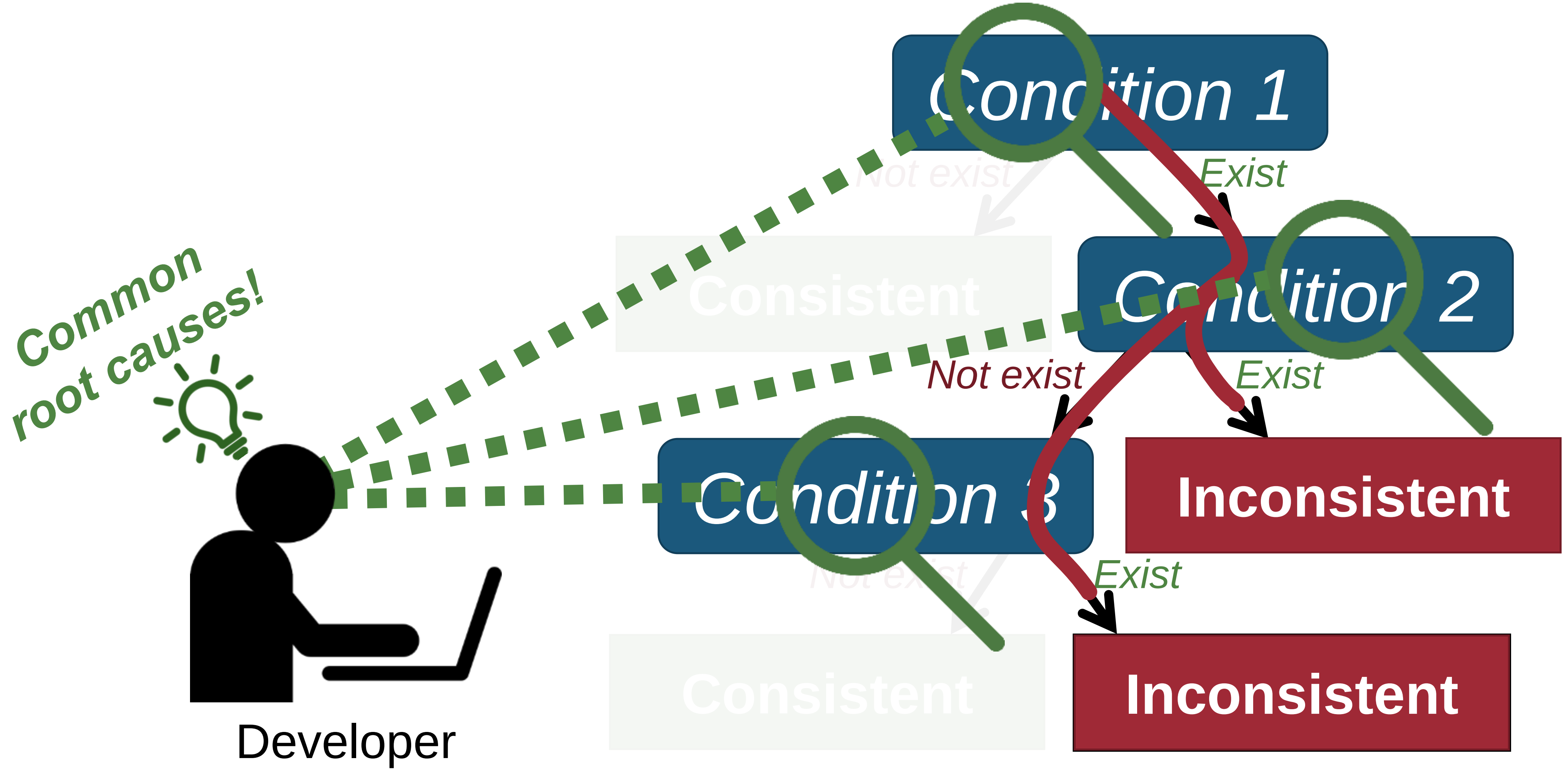


4,000,000
inconsistent results



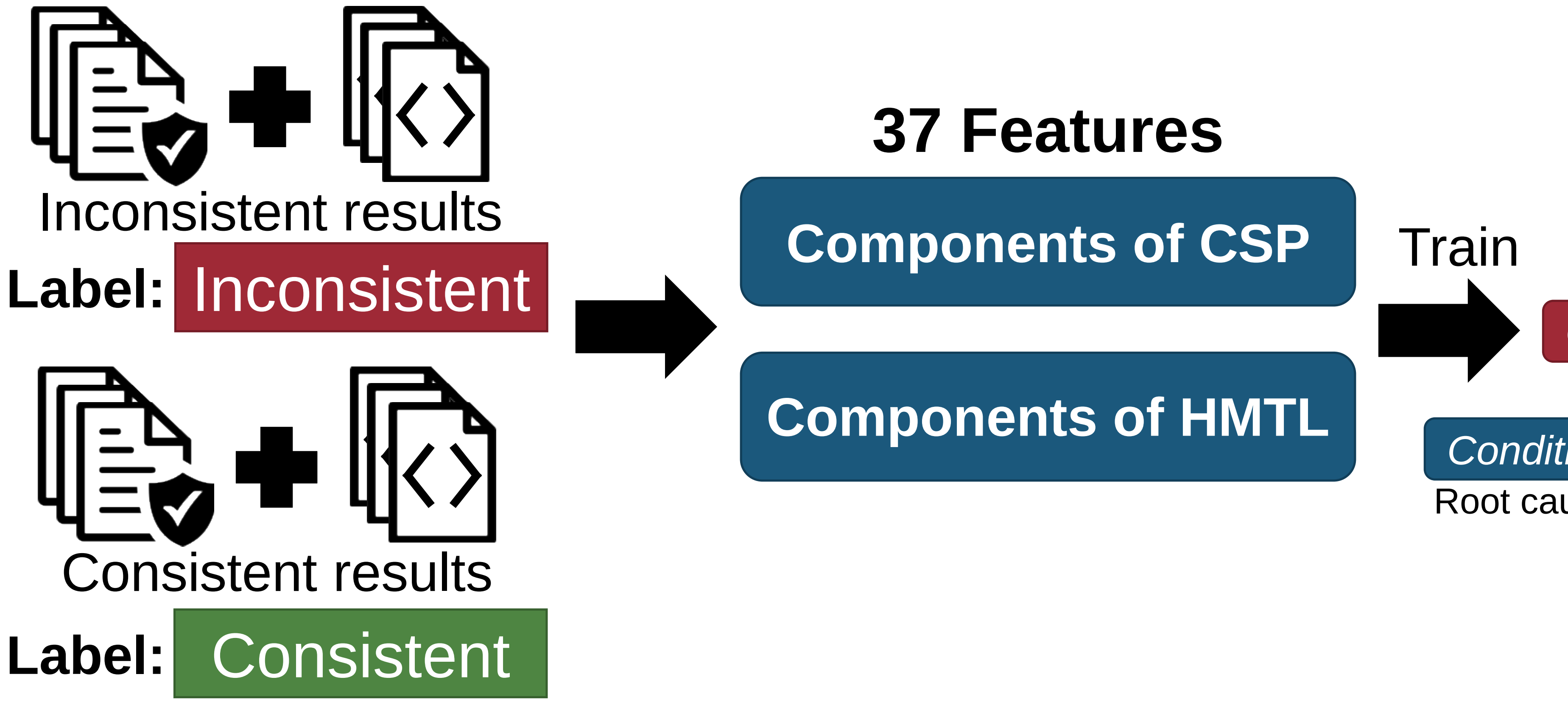
Decision tree

Decision Tree-based Root Cause Analysis



Decision tree

Decision Tree-based Root Cause Analysis



Decision Tree-based Root Cause Analysis

4,000,000 cases $\longrightarrow$ **525 paths**

Inconsistent results
Label: Inconsistent
37 Features
Components of CSP
Components of HMTL
Train
Condition 1
Condition 2
Condition 3
Root cause #3
Condition 4
Root cause #1
Condition 5
Root cause #2

We analyzed **only 525 paths**
to pinpoint the root causes

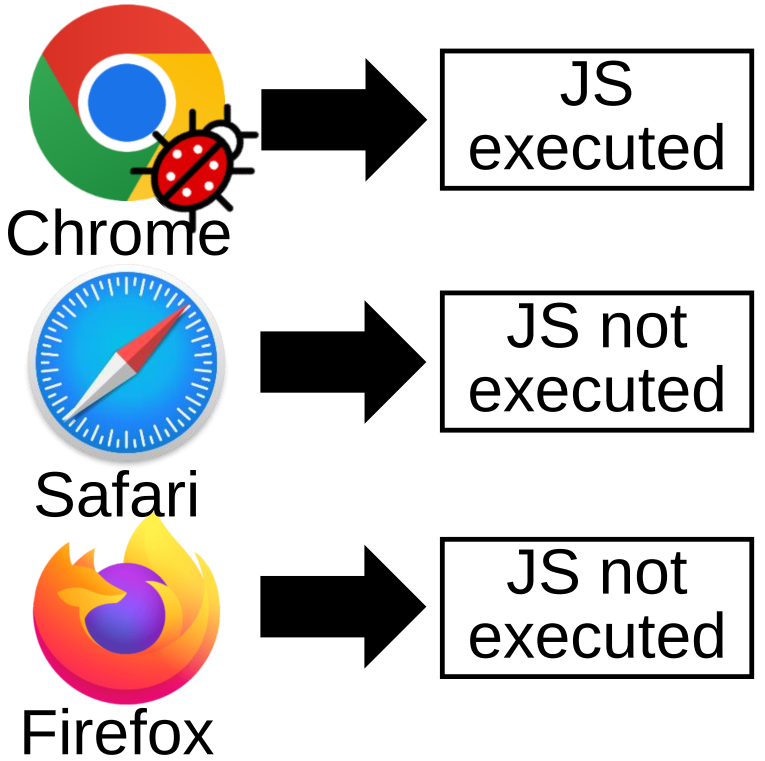
Decision Tree-based Root Cause Analysis

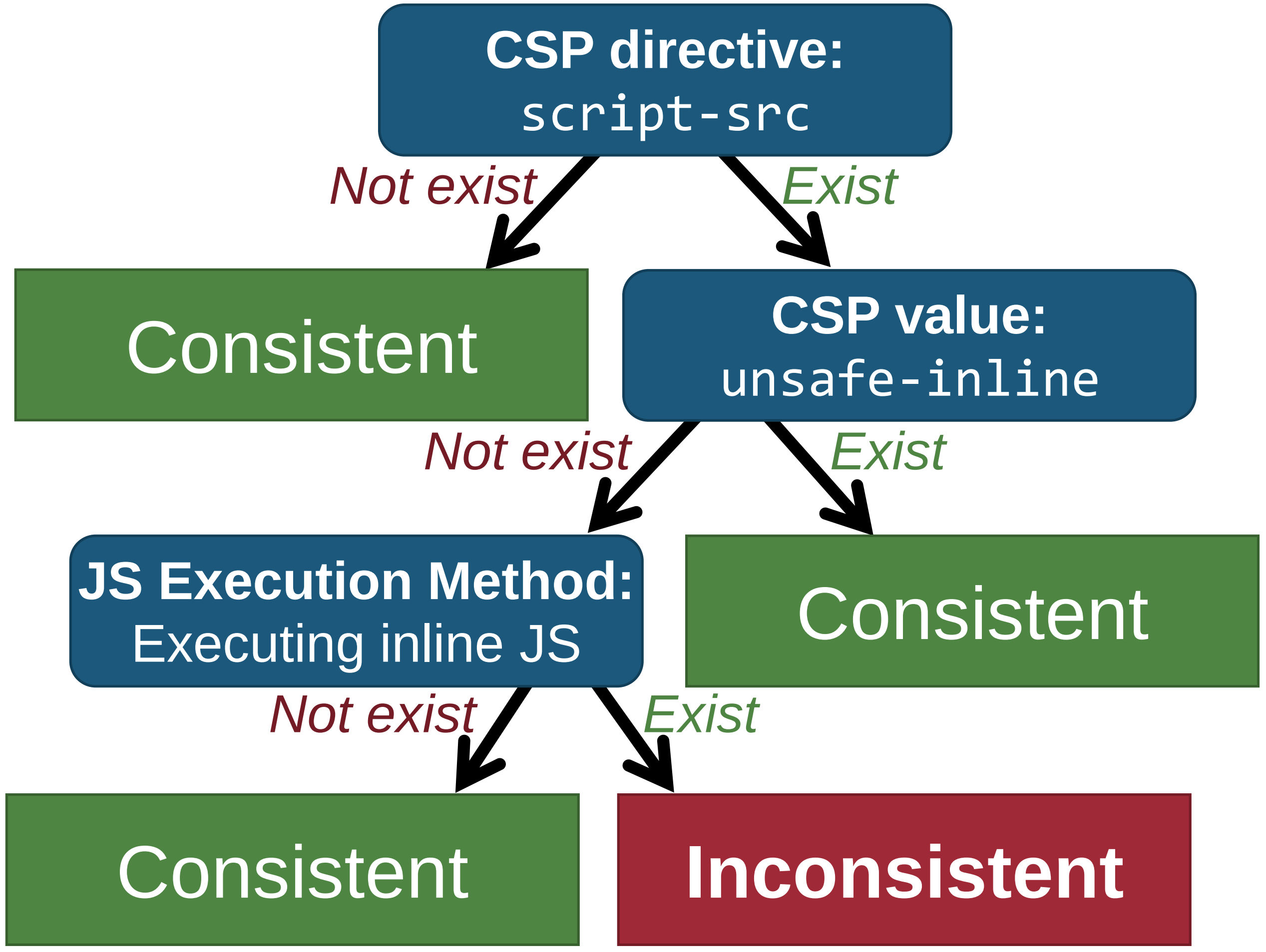
CSP

```
script-src benign.com
```

HTML

```
<script>attack()</script>
```





Decision tree

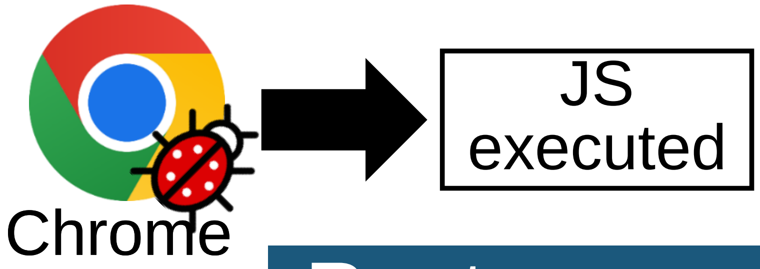
Decision Tree-based Root Cause Analysis

CSP

```
script-src benign.com
```

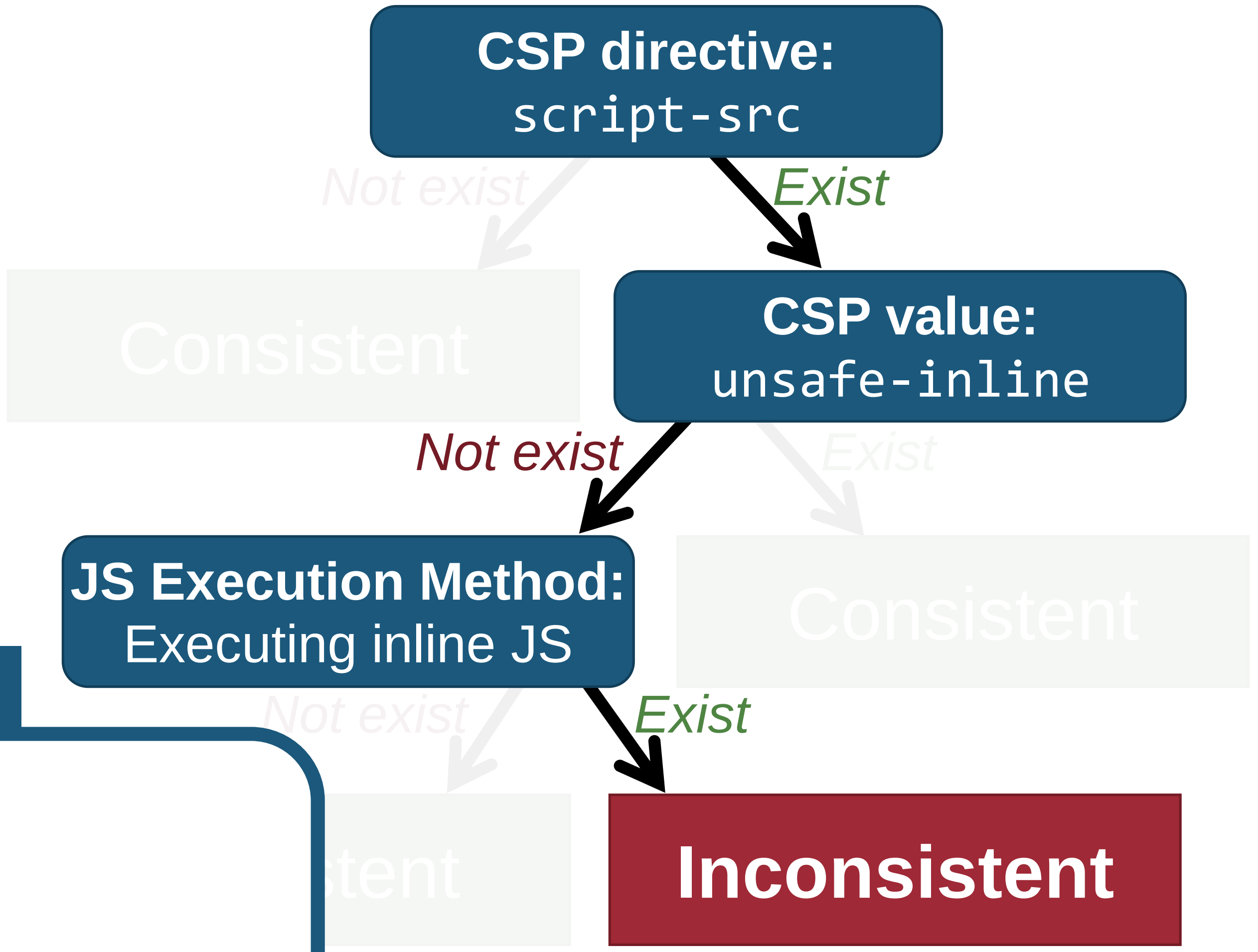
HTML

```
<script>attack()</script>
```



Root cause analysis

Inconsistencies occurred when



Decision tree

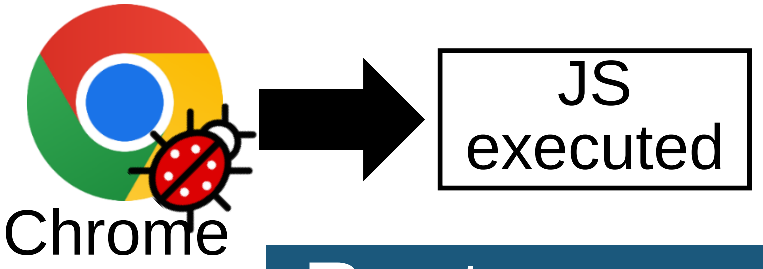
Decision Tree-based Root Cause Analysis

CSP

```
script-src benign.com
```

HTML

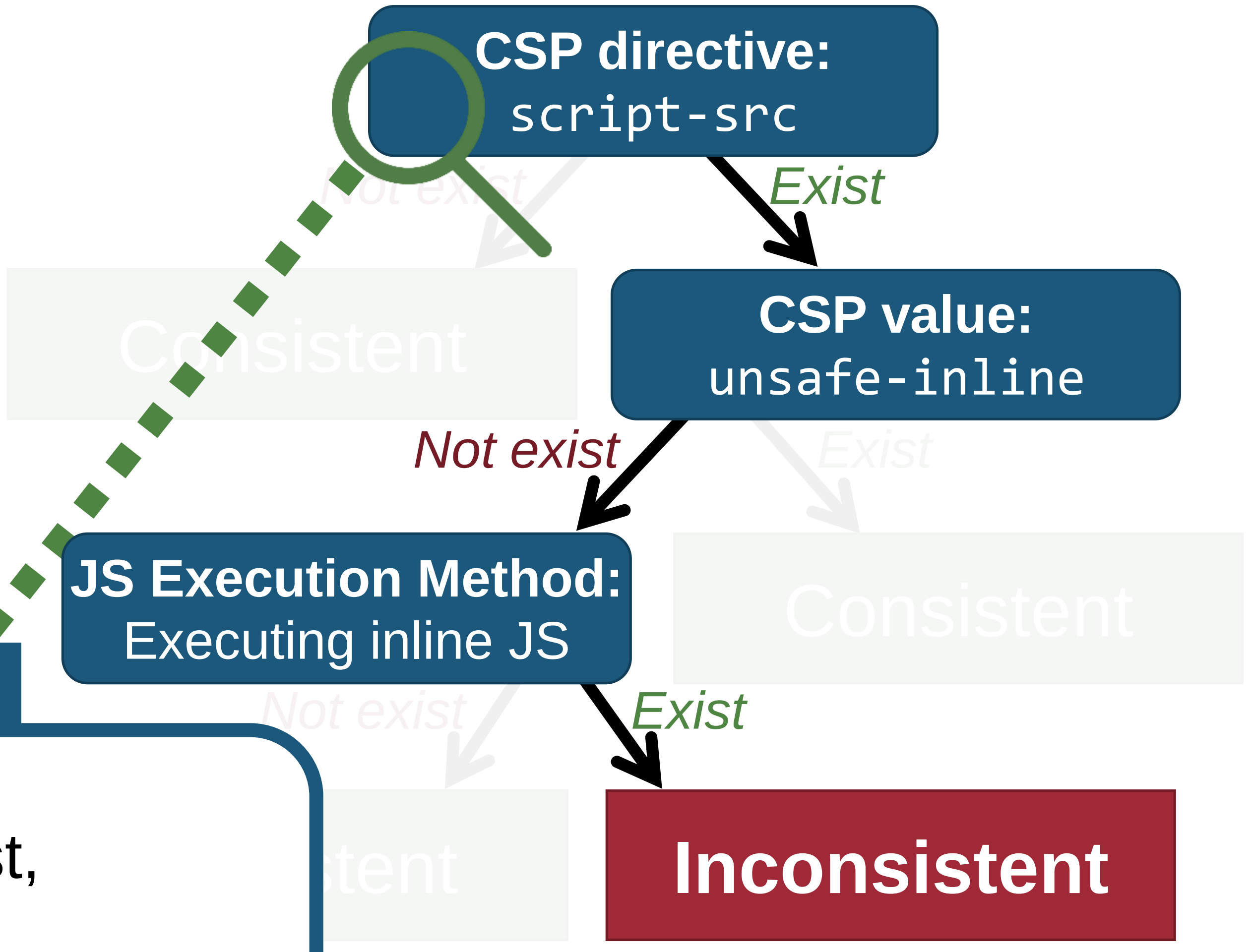
```
<script>attack()</script>
```



Root cause analysis

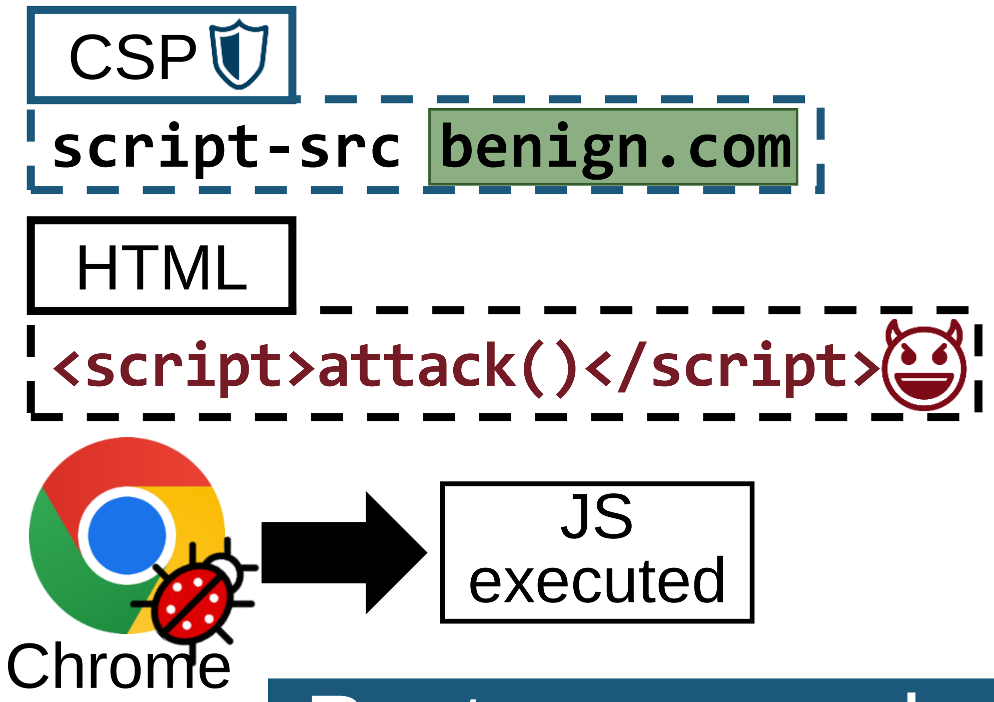
Inconsistencies occurred when

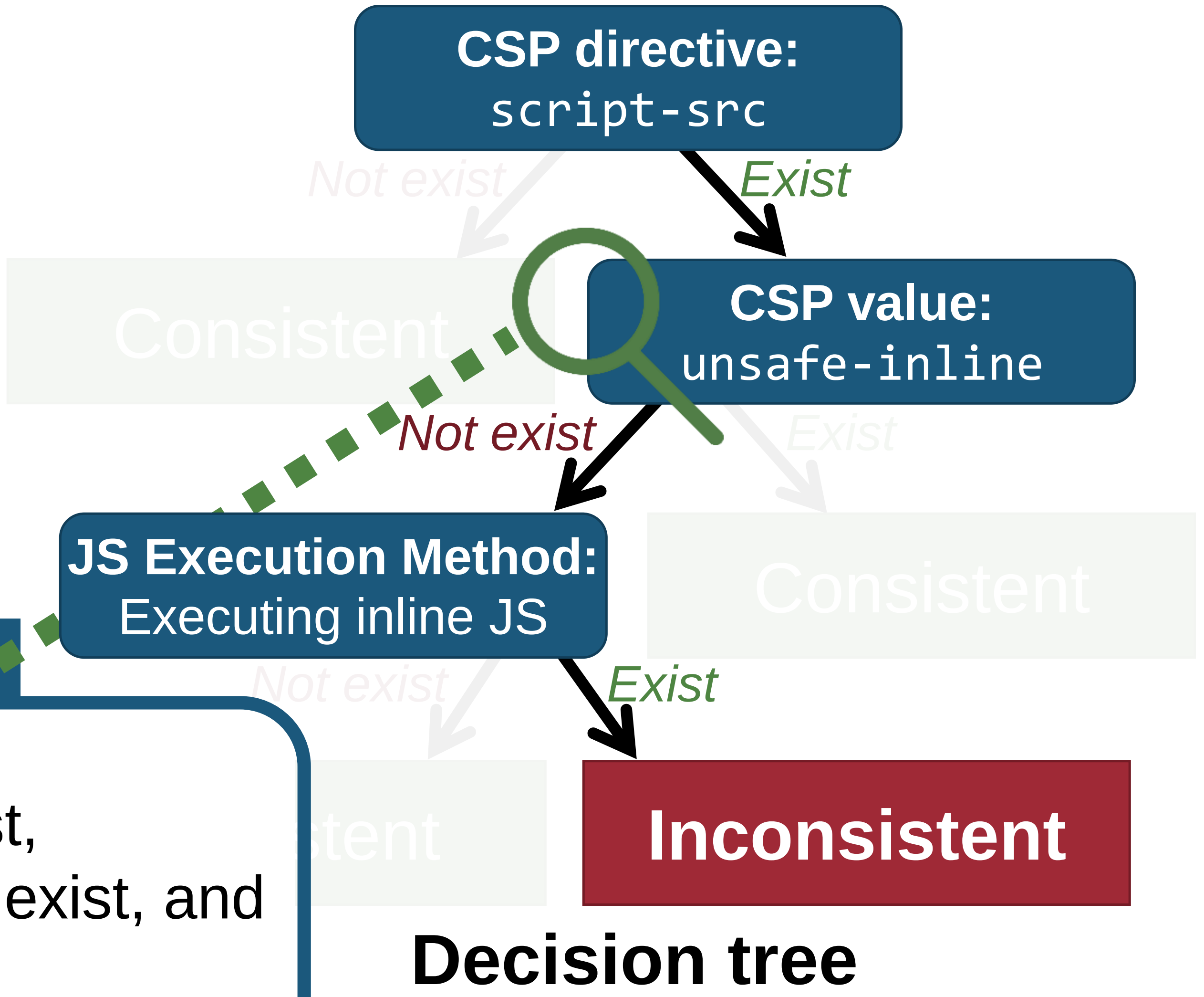
1. the **`script-src` directive** is exist,



Decision tree

Decision Tree-based Root Cause Analysis





Root cause analysis

Inconsistencies occurred when

1. the **script-src directive** is exist,
2. the **unsafe-inline value** is not exist, and

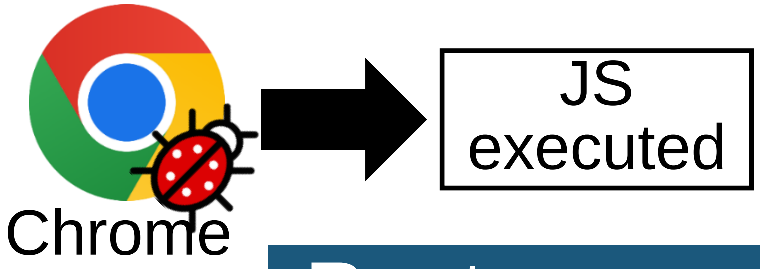
Decision Tree-based Root Cause Analysis

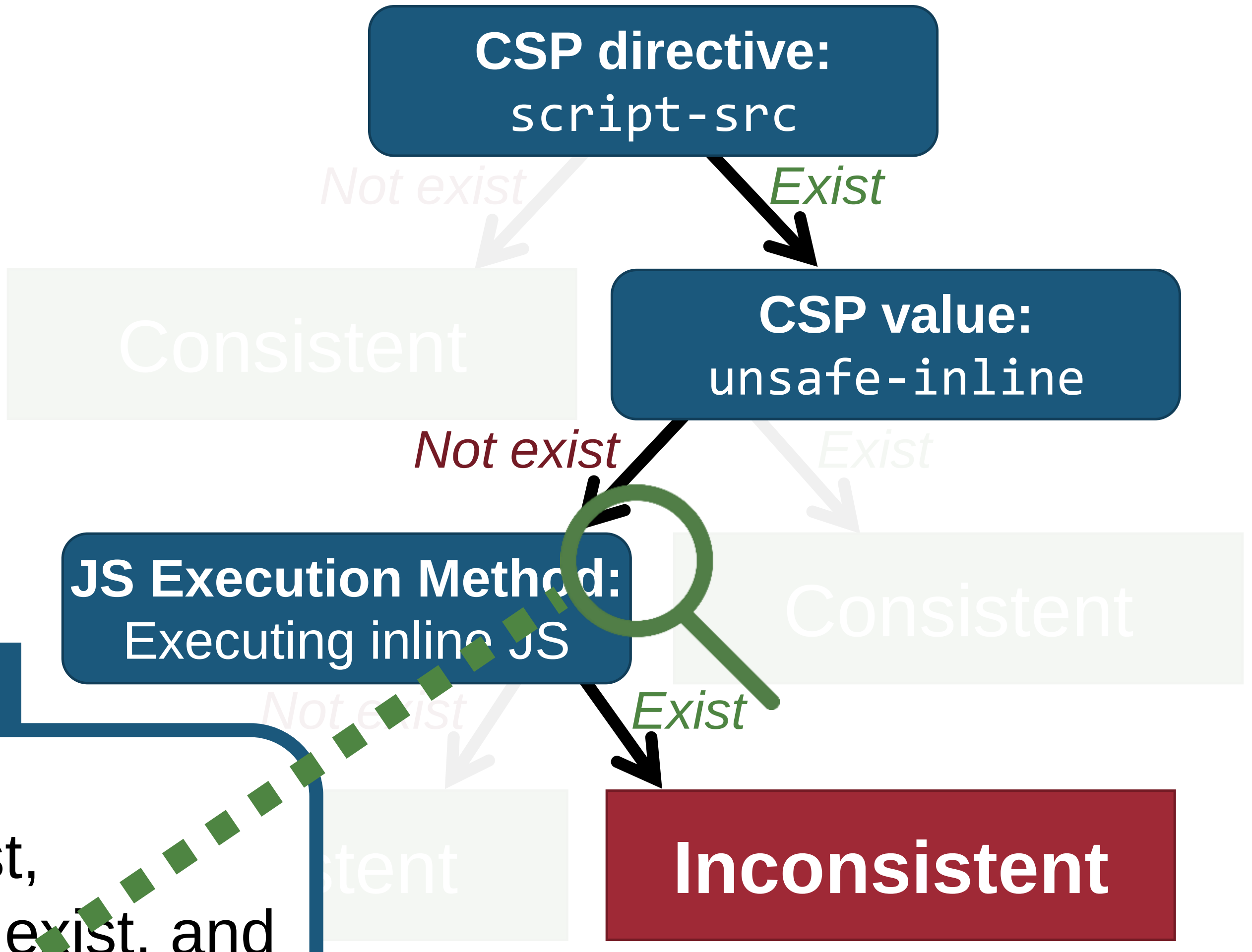
CSP

```
script-src benign.com
```

HTML

```
<script>attack()</script>
```





Decision tree

Root cause analysis

Inconsistencies occurred when

1. the **`script-src` directive** is exist,
2. the **`unsafe-inline` value** is not exist, and
3. the **inline script** is exist in the HTML snippet!

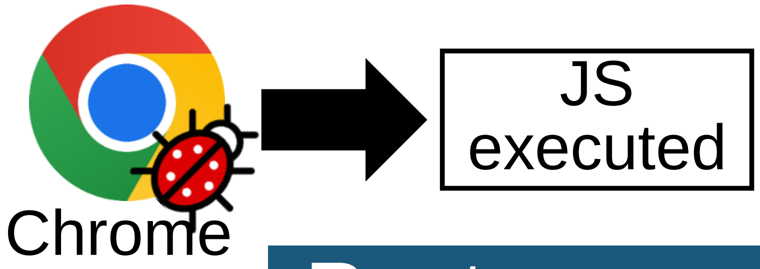
Decision Tree-based Root Cause Analysis

CSP

```
script-src benign.com
```

HTML

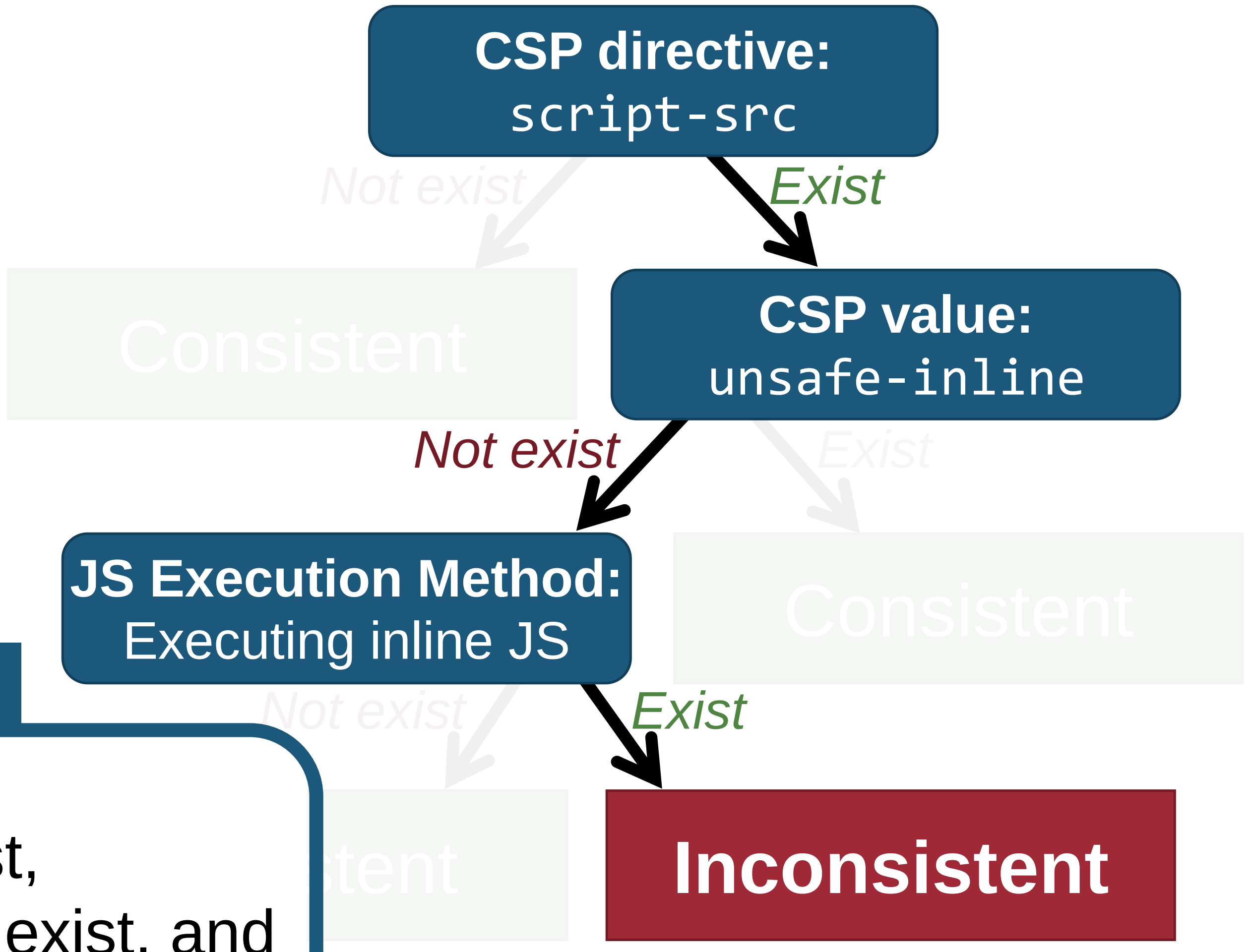
```
<script>attack()</script>
```



Root cause analysis

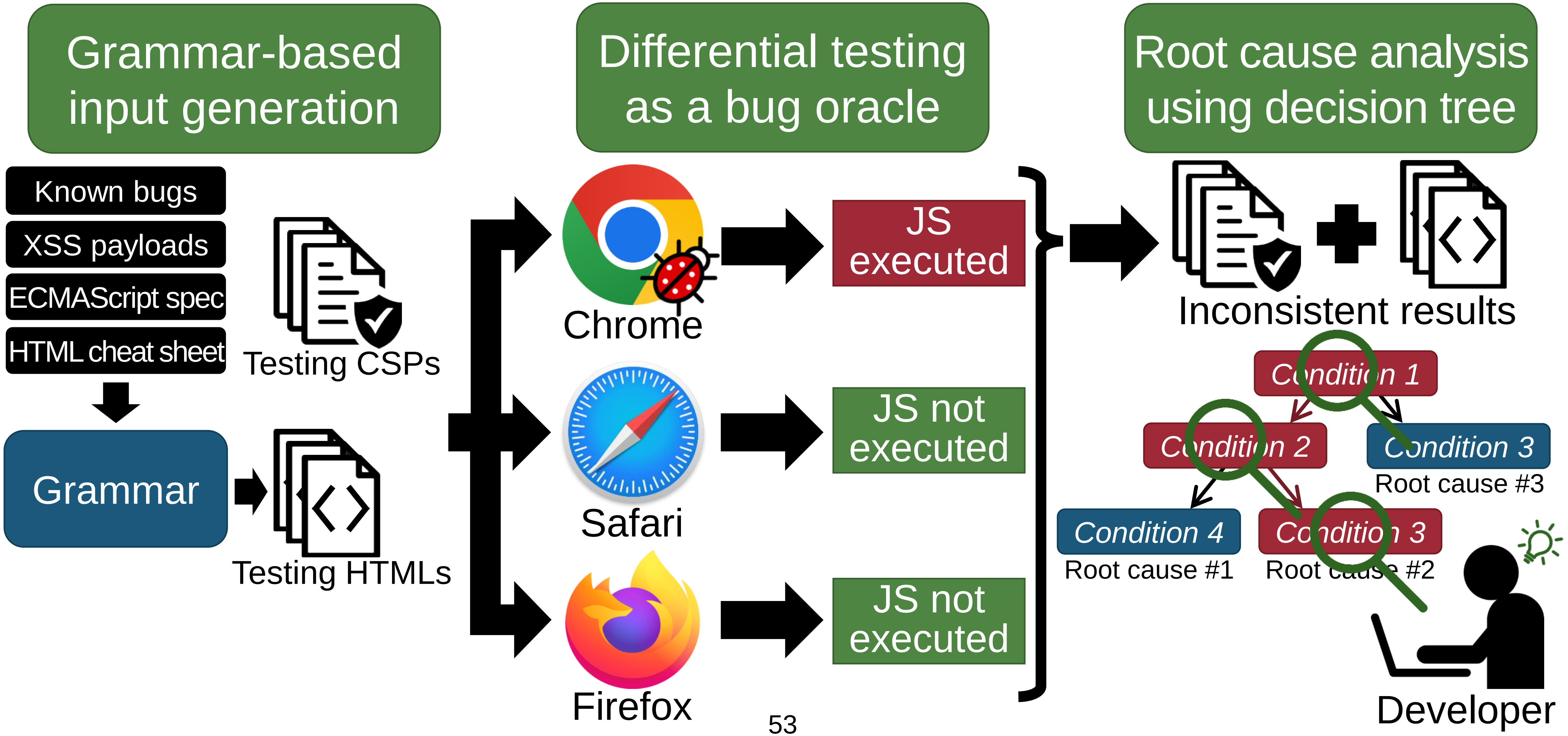
Inconsistencies occurred when

1. the **`script-src` directive** is exist,
2. the **`unsafe-inline` value** is not exist, and
3. the **inline script** is exist in the HTML snippet!



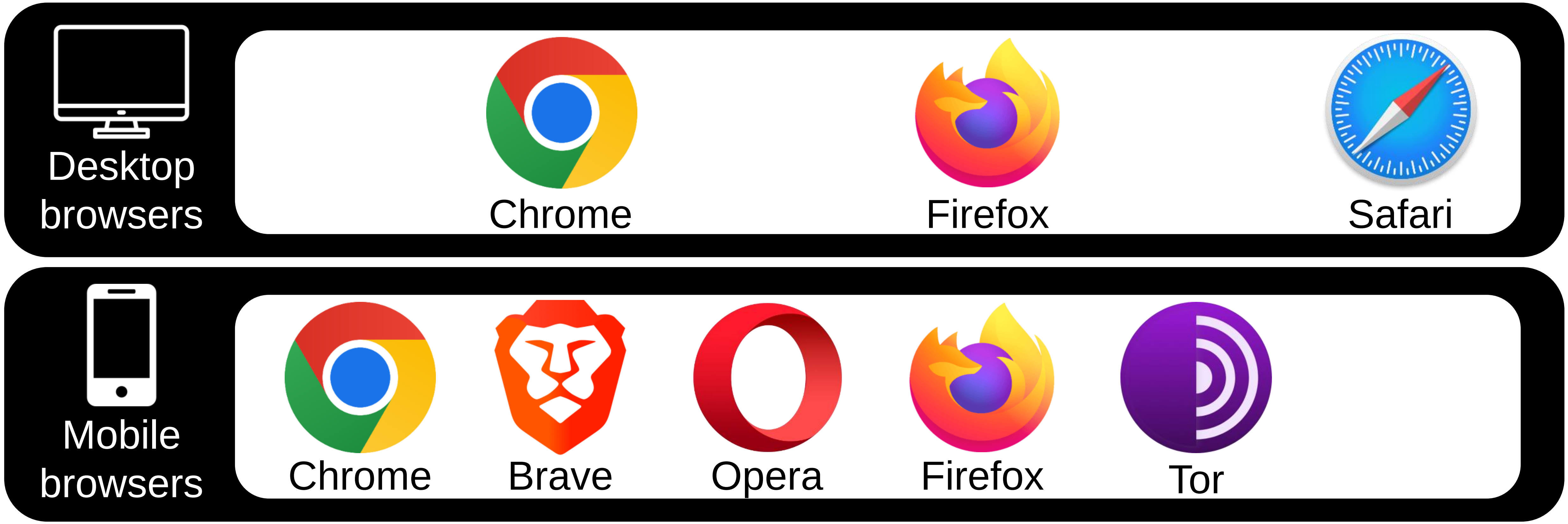
Decision tree

Evaluation


Grammar-based input generation
Differential testing as a bug oracle
Root cause analysis using decision tree
Known bugs
XSS payloads
ECMAScript spec
HTML cheat sheet
Testing CSPs
Grammar
Testing HTMLs
Chrome
Safari
Firefox
JS executed
JS not executed
JS not executed
Inconsistent results
Condition 1
Condition 2
Condition 3
Root cause #3
Condition 4
Root cause #1
Condition 3
Root cause #2
Developer

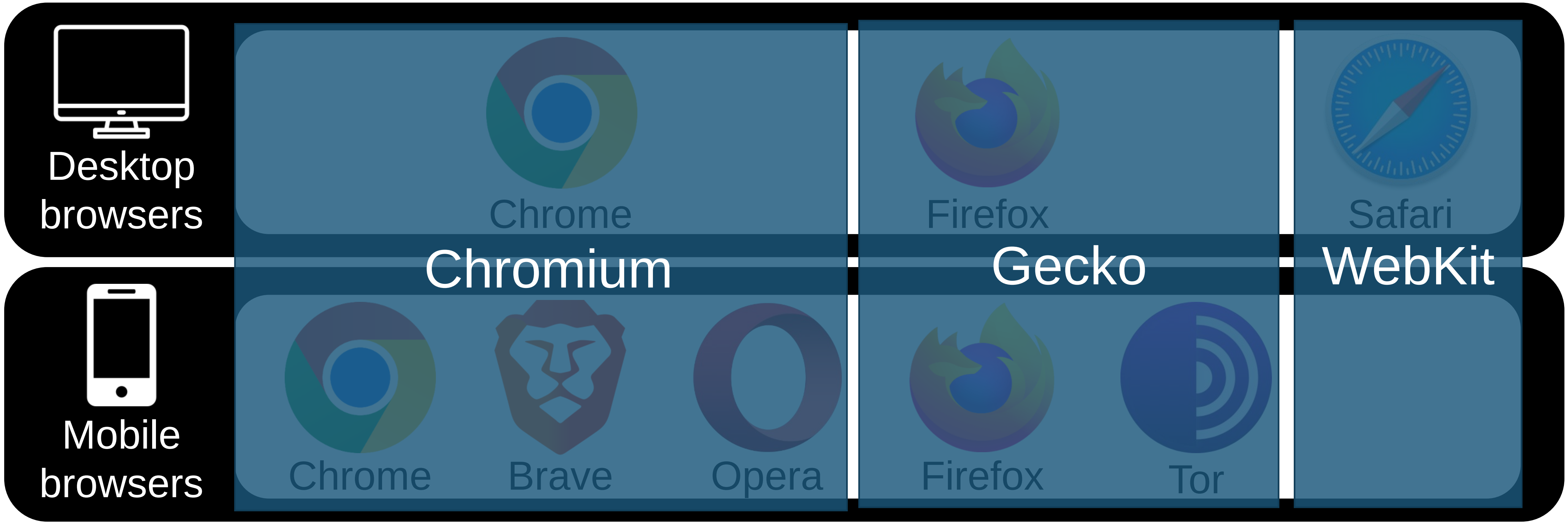
Experimental Setup

- Target browsers: eight popular browsers



Experimental Setup

- Target browsers: eight popular browsers



Bugs Found

- Found **37 CSP enforcement bugs** in three browser engines with 4M inconsistent results
  - # of security bugs: 27
  - # of specification bugs: 3
  - # of functional bugs: 7

- We reported **27 security bugs** resulting from vendor's mistakes
  - 23 bugs have been patched (12 bugs were patched due to our report)

Google rewarded with $4,000!

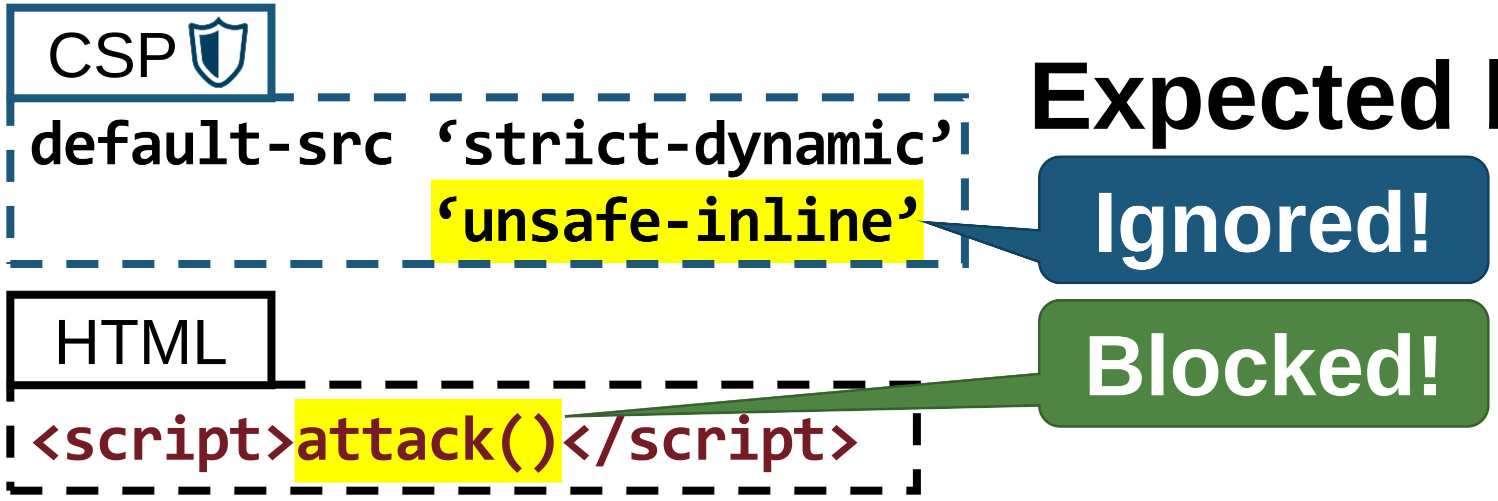
Root Causes

Incorrect CSP inheritance	2	0	6
Incorrect hash handling	1	0	2
Non-ignored directive values	1	0	1
Non-supporting specific directives	0	2	0
Non-supporting specific directive values	0	3	1
Auto-enabling directive values by default	0	1	1
Auto-enabling directive values on specific conditions	0	0	5
Non-supporting CSP for specific status code	1	0	0
Incorrect handling of malformed CSPs	0	1	0
Allowing out-going request	1	1	0

Case Study: Non-ignored CSP Values

CSP3 specification

✓ `unsafe-inline` **should be ignored** when `strict-dynamic` is specified



```
default-src 'strict-dynamic'
            'unsafe-inline'
```

```
<script>attack()</script>
```

Expected behavior:

Case Study: Non-ignored CSP Values

CSP3 specification

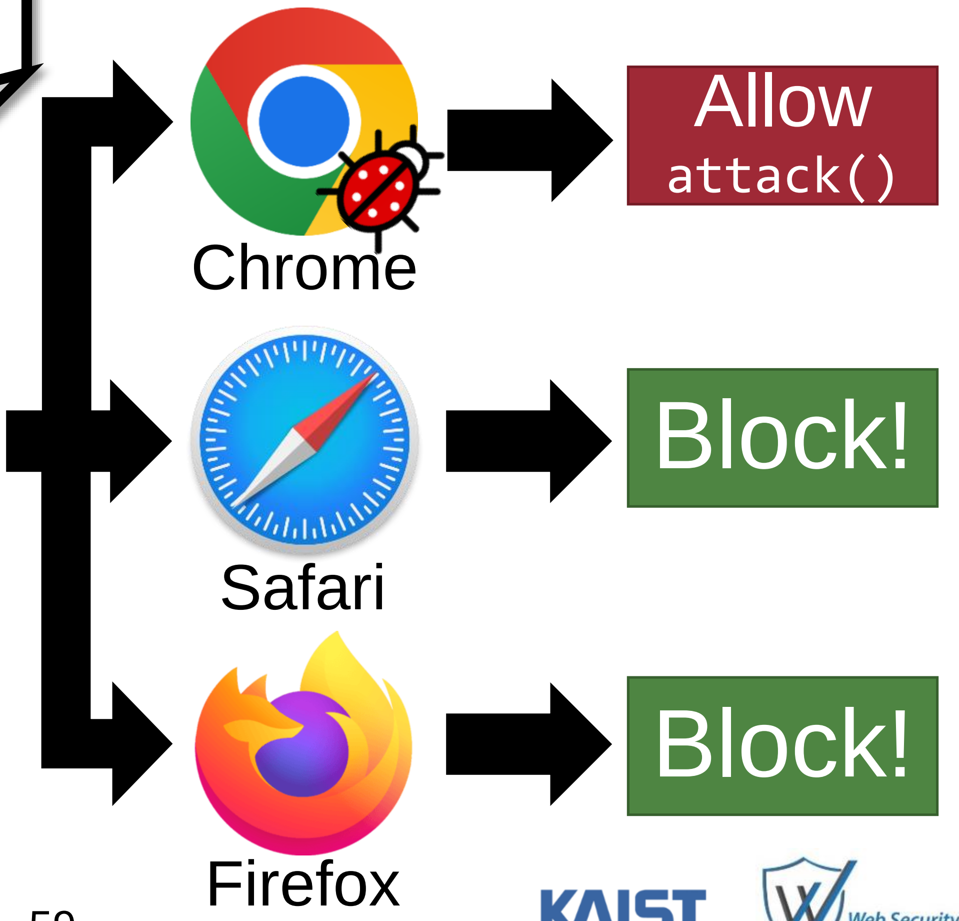
✓ `unsafe-inline` **should be ignored** when `strict-dynamic` is specified

CSP

```
default-src 'strict-dynamic'
            'unsafe-inline'
```

HTML

```
<script>attack()</script>
```



Case Study: Non-ignored CSP Values

CSP3 specification

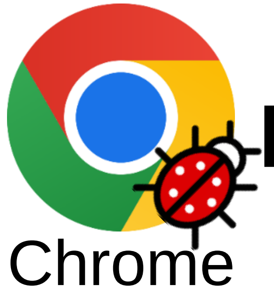
✓ `unsafe-inline` **should be ignored** when `strict-dynamic` is specified

CSP

```
default-src 'strict-dynamic'
            'unsafe-inline'
```

HTML

```
<script>attack()</script>
```



Allow `attack()`

Lesson #1: Complex CSP Specification

CSP3 specification

✓ `unsafe-inline` **should be ignored** when `strict-dynamic` is specified

83% bugs are caused by CSP Level 3

- CSP Level 1, 2012
- CSP Level 2, 2014 — + hash handling, nonce handling, …
- CSP Level 3, 2015 — + **`‘strict-dynamic’`**, `‘unsafe-hashes’`, …

KAIST Web Security & Privacy Lab CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Lesson #1: Complex CSP Specification

CSP specification

Complex description

Require more **comprehensive browser testing!**

- CSP Level 1, 2012
- CSP Level 2, 2014 — + hash handling, nonce handling, …
- CSP Level 3, 2015 — + **`'strict-dynamic'`**, `'unsafe-hashes'`, …

Case Study: Incorrect CSP Inheritance

CSP specification

- ✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src ‘nonce-123’
```

HTML

```
<iframe src=“javascript:attack()”>
</iframe>
```

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src 'nonce-123'
```

HTML

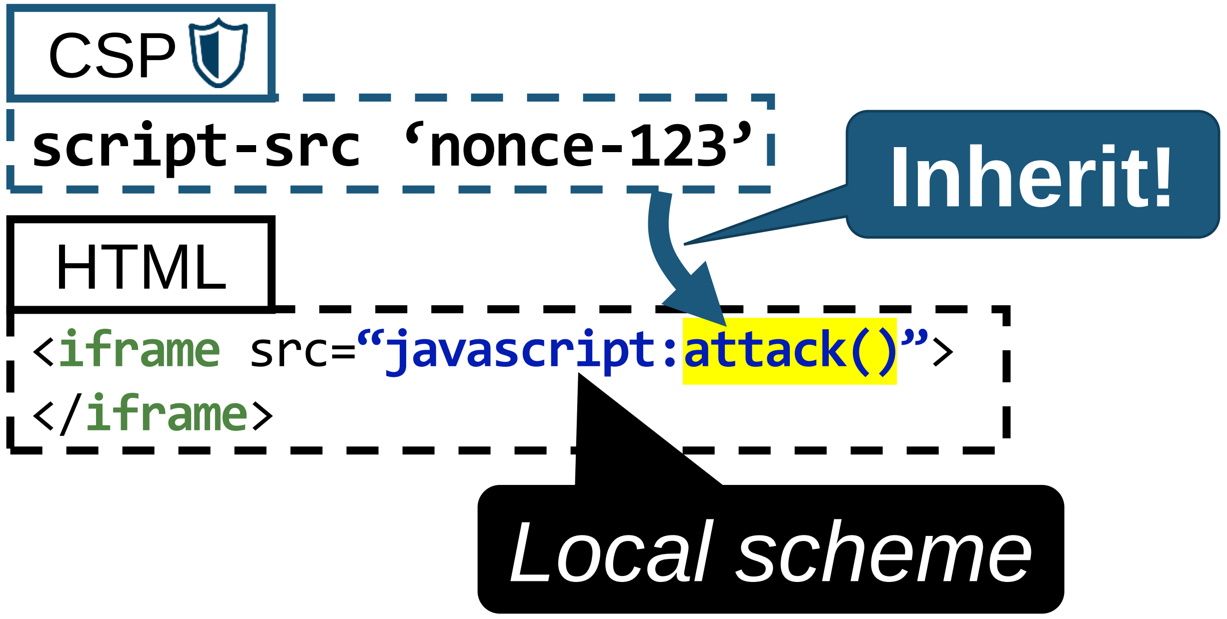
```
<iframe src="javascript:attack()">
</iframe>
```

Local scheme

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page



```
script-src 'nonce-123'
```

```
<iframe src="javascript:attack()">
</iframe>
```

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Work Flow

```
src="tmp.html"
```

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src ‘nonce-123’
```

HTML

```
<iframe id=“z” src=“tmp.html” />
<script nonce=123>
  z.addEventListener(“load”, () => {
     z.src=“javascript:attack()”
  ;});
</script>
```

Work Flow

`src=“tmp.html”`

↓ on “load” event

`src=“javascript:attack()”`

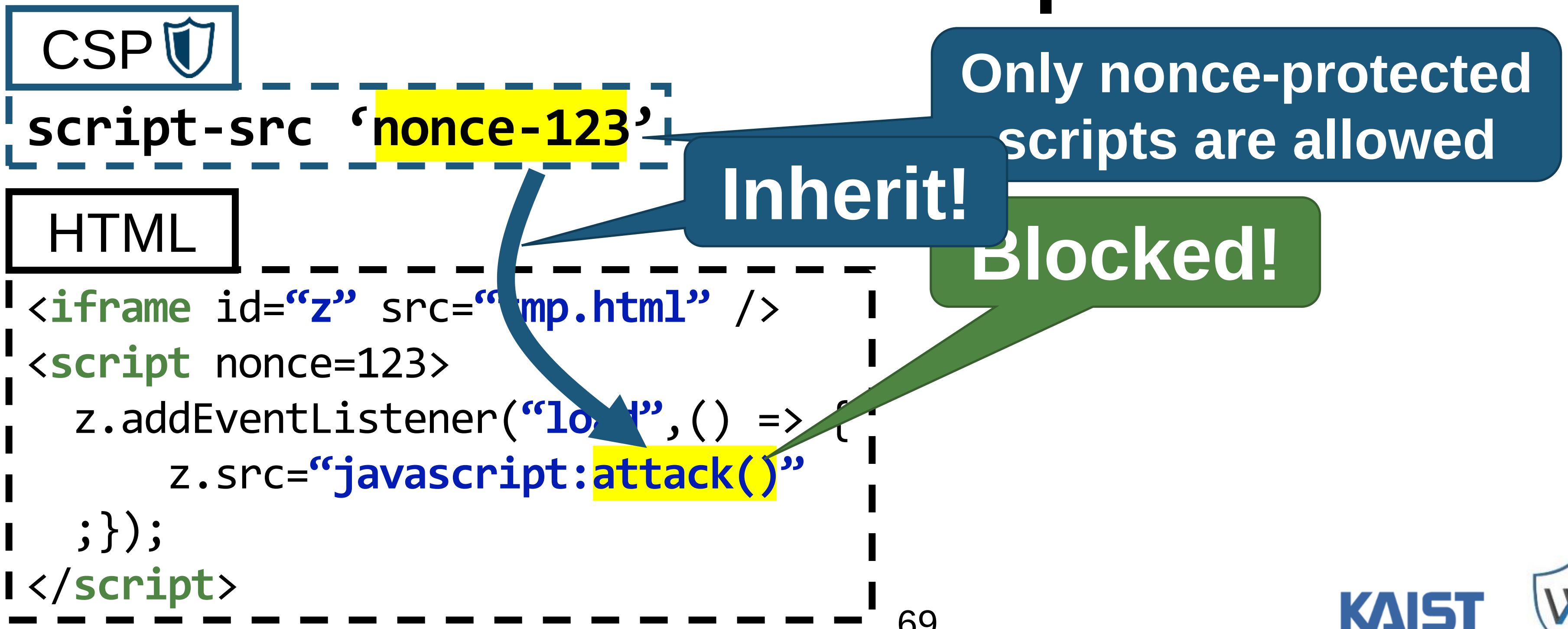
Dynamic page **redirection**

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

Expected behavior:



```
script-src 'nonce-123'
```

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Case Study: Incorrect CSP Inheritance

CSP specification

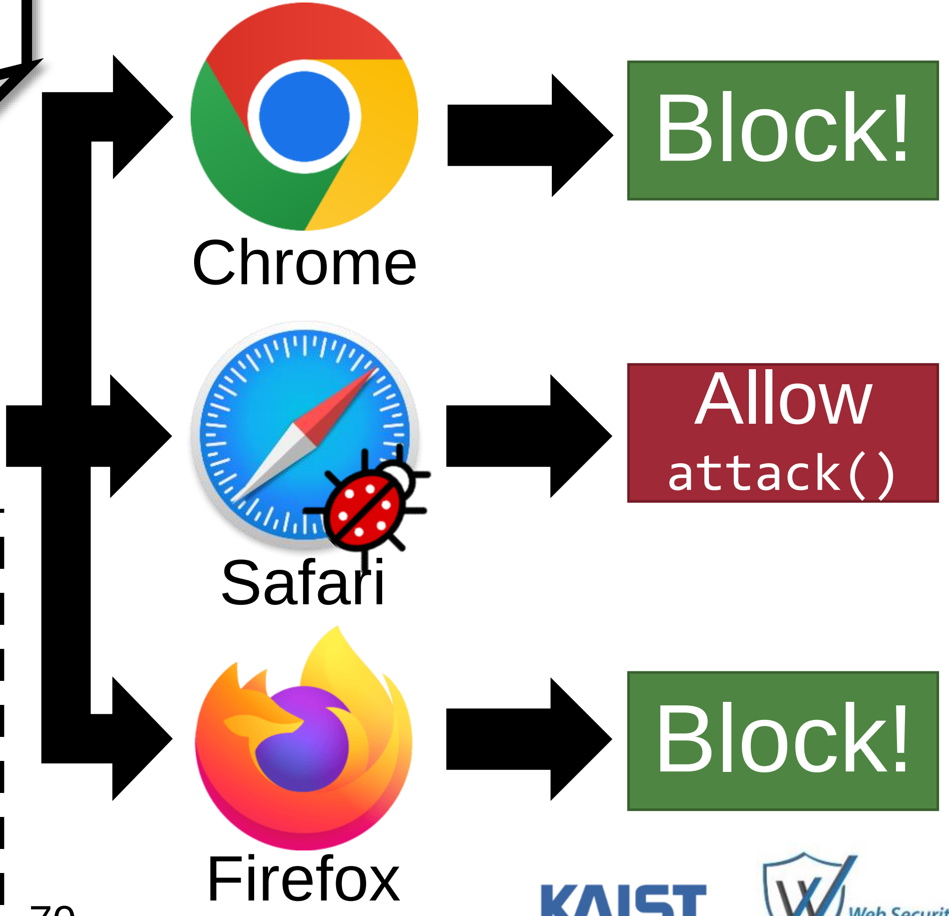
✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```



Lesson #2: Page Redirection

CSP specification

- ✓ Documents from *local schemes* will **inherit the CSP** of their parent page

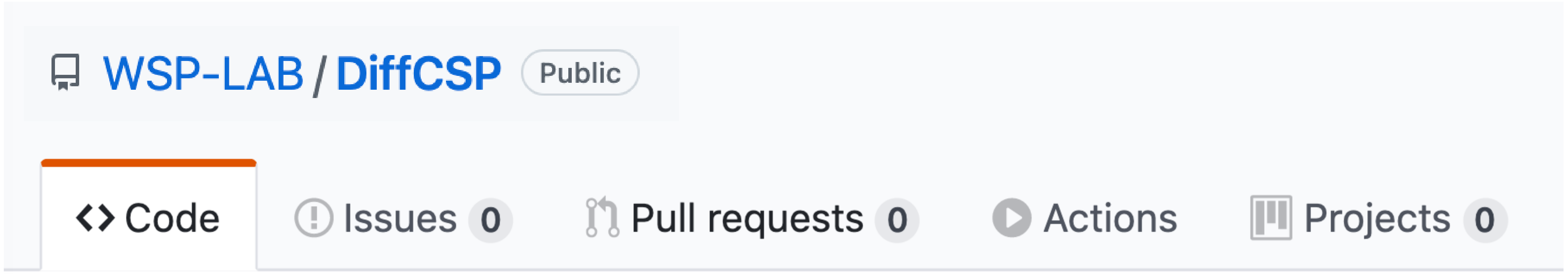
28% bugs are caused by <u>page redirection!</u>

Limitations

- DiffCSP cannot find a bug if all the browsers **exhibit the same bug**

- DiffCSP cannot find a bug if there exist **unknown HTML forms** of executing JS snippets

Open Science


WSP-LAB / DiffCSP Public

https://github.com/WSP-LAB/DiffCSP

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement
- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement
- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?

Conclusion

- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

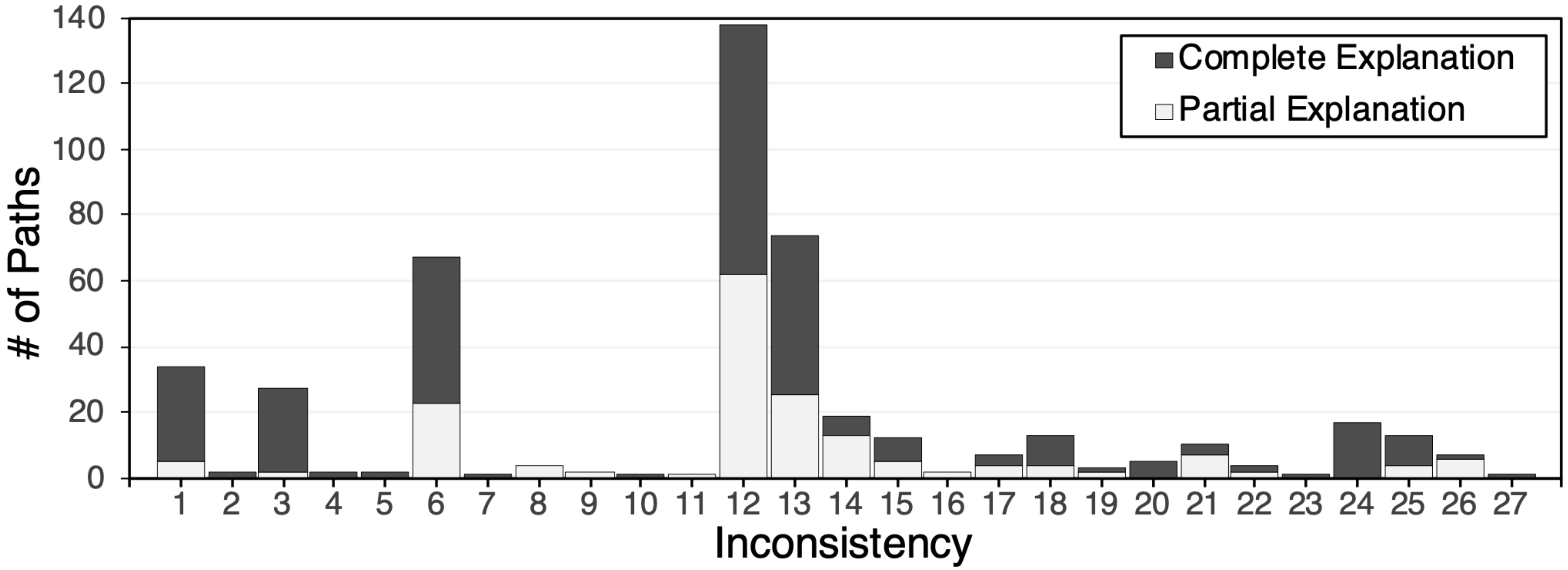
Thank you! Questions?

Conclusion

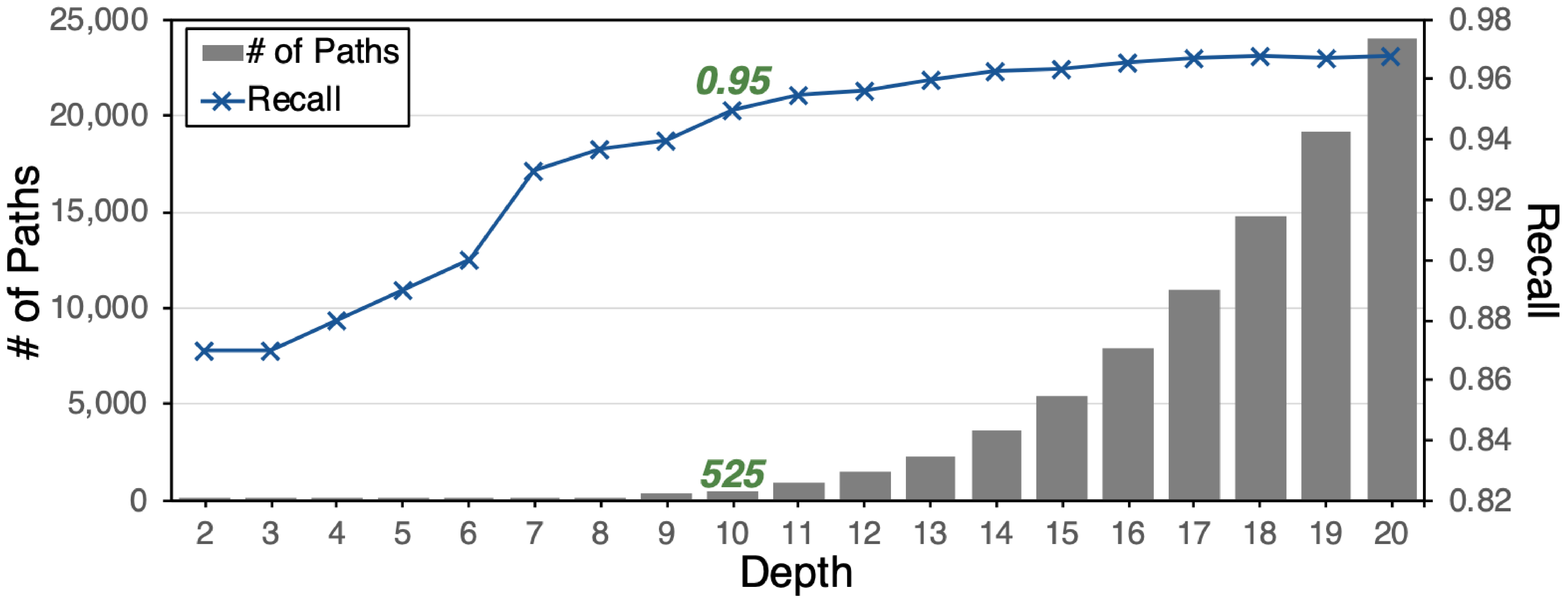
- We propose DiffCSP, the differential testing framework designed to **identify CSP enforcement bugs**
  - We propose an HTML grammar to generate diverse test inputs
  - We conduct differential testing to identify the correct behavior
  - We leverage decision trees to pinpoint the root causes for erroneous CSP enforcement

- We found **29 security bugs** and eight functional bugs

Thank you! Questions?



(a) Desktop browsers.



# of Paths
Recall
0.95
525
Depth
25,000
20,000
15,000
10,000
5,000
0
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

More in the Paper

- **CSP generation**
- **JS category**
  - Executing inline JS
  - Evaluating string
  - Dynamically fetching JS
  - Redirecting to scheme
  - Expanding document
  - Writing to opened document
- **HTML category**
  - Executing inline JS in script tag
  - Fetching JS in script tag
  - Redirecting to scheme
  - Executing inline JS in event handler
  - Writing to frame
  - Changing location of iframe
  - Evaluating string via frame's function
  - Expanding document

Grammar-based Input Generation

Known CSP bugs

```
document.body.innerHTML+=
  “<script>
     eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

```
document.body.innerHTML+=
  “[HTML]”
```

```
eval(‘[JS]’)
```

```
attack()
```

[HTML]

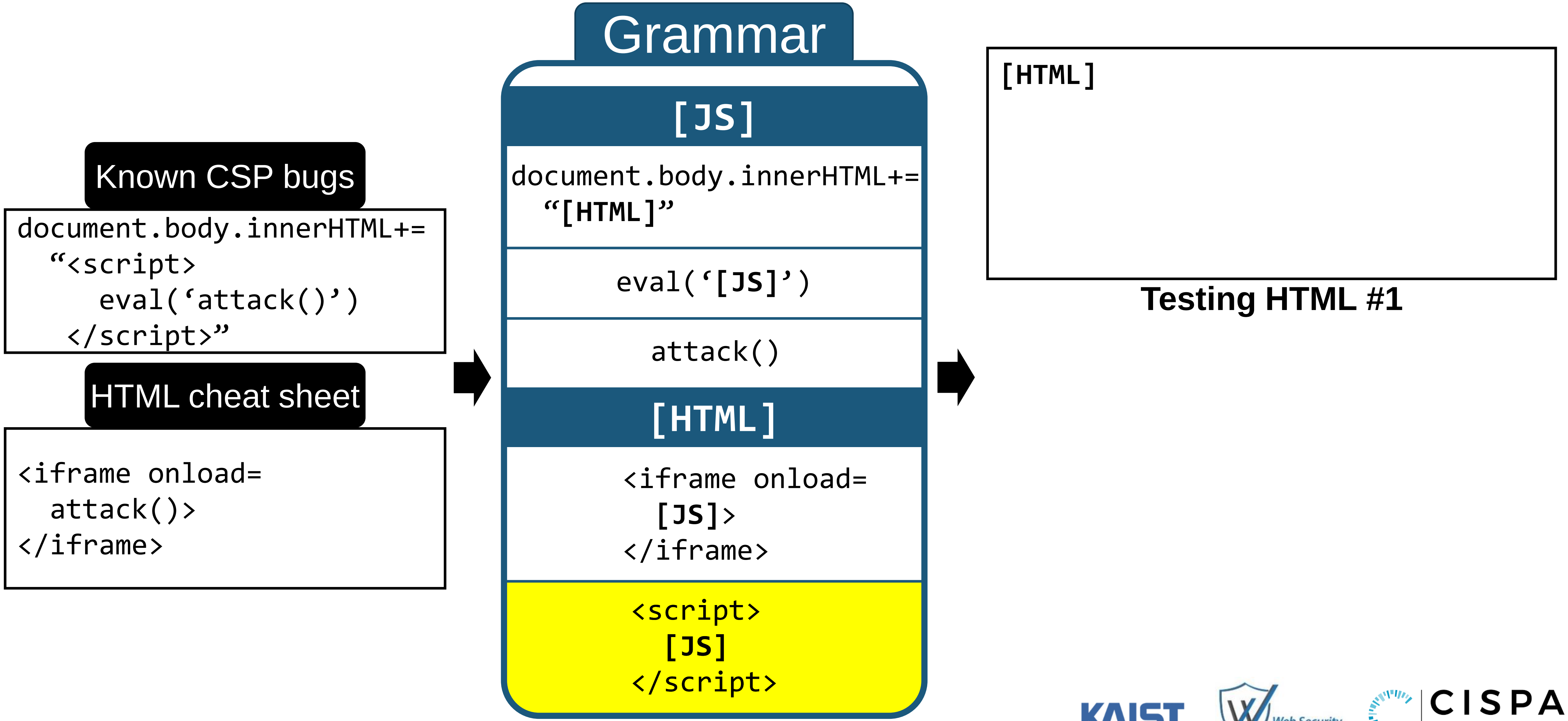
```
<iframe onload=
  [JS]>
</iframe>
```

```
<script>
  [JS]
</script>
```

```
[HTML]
```

Testing HTML #1

Grammar-based Input Generation



Known CSP bugs

```
document.body.innerHTML+=
  “<script>
     eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

```
document.body.innerHTML+=
  “[HTML]”
```

```
eval(‘[JS]’)
```

```
attack()
```

[HTML]

```
<iframe onload=
  [JS]>
</iframe>
```

```
<script>
  [JS]
</script>
```

```
[HTML]
```

Testing HTML #1

Grammar-based Input Generation

Known CSP bugs

```
document.body.innerHTML+=
  “<script>
      eval(‘attack()’)
   </script>”
```

HTML cheat sheet

```
<iframe onload=
  attack()>
</iframe>
```

Grammar

[JS]

```
document.body.innerHTML+=
  “[HTML]”
```

```
eval(‘[JS]’)
```

```
attack()
```

[HTML]

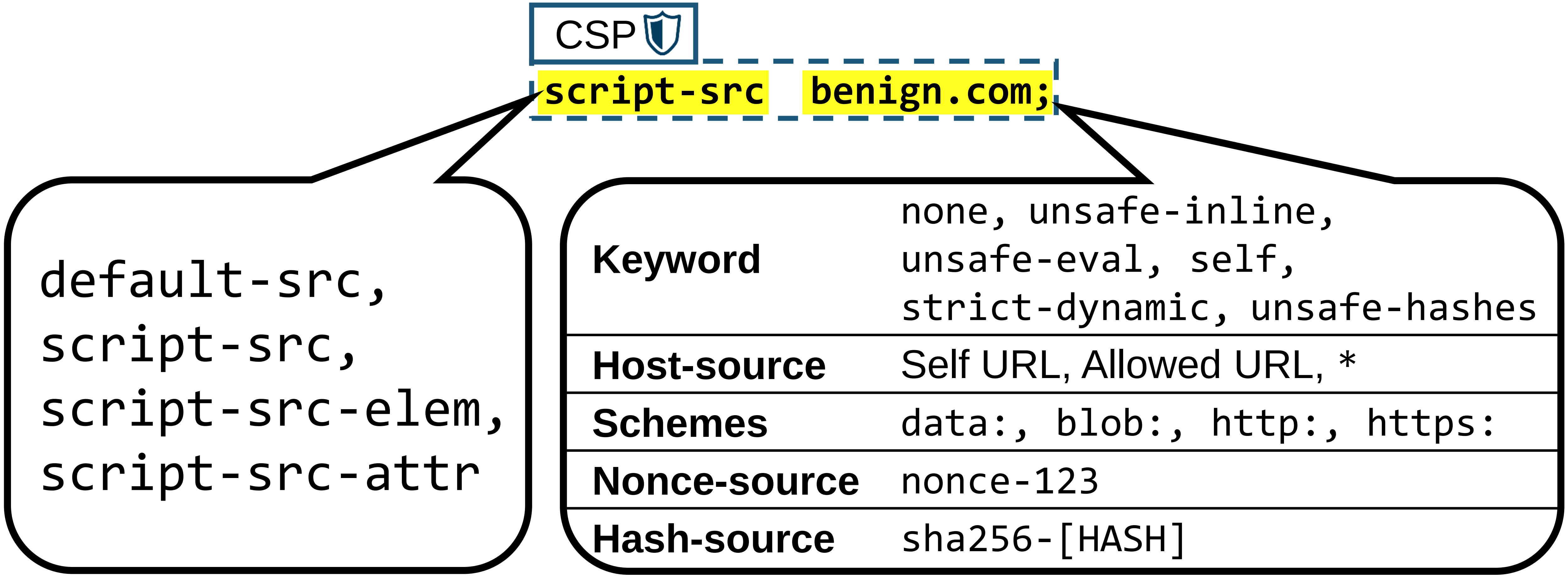
```
<iframe onload=
  [JS]>
</iframe>
```

```
<script>
  [JS]
</script>
```

```
<script>
  document.body.innerHTML+=
    “[HTML]”
</script>
```

Testing HTML #1

More in the Paper – CSP generation



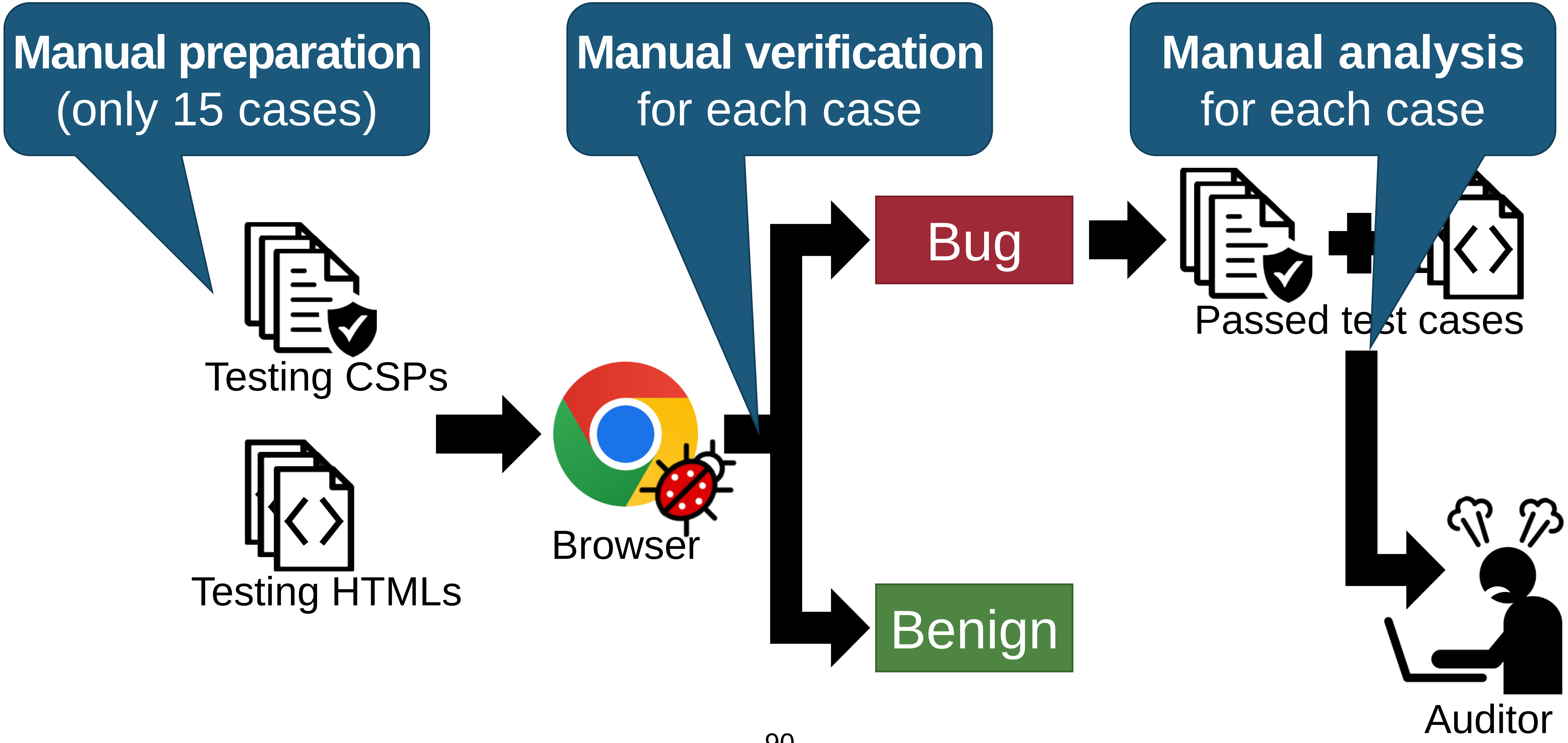
```
script-src benign.com;
```

```
default-src,
script-src,
script-src-elem,
script-src-attr
```

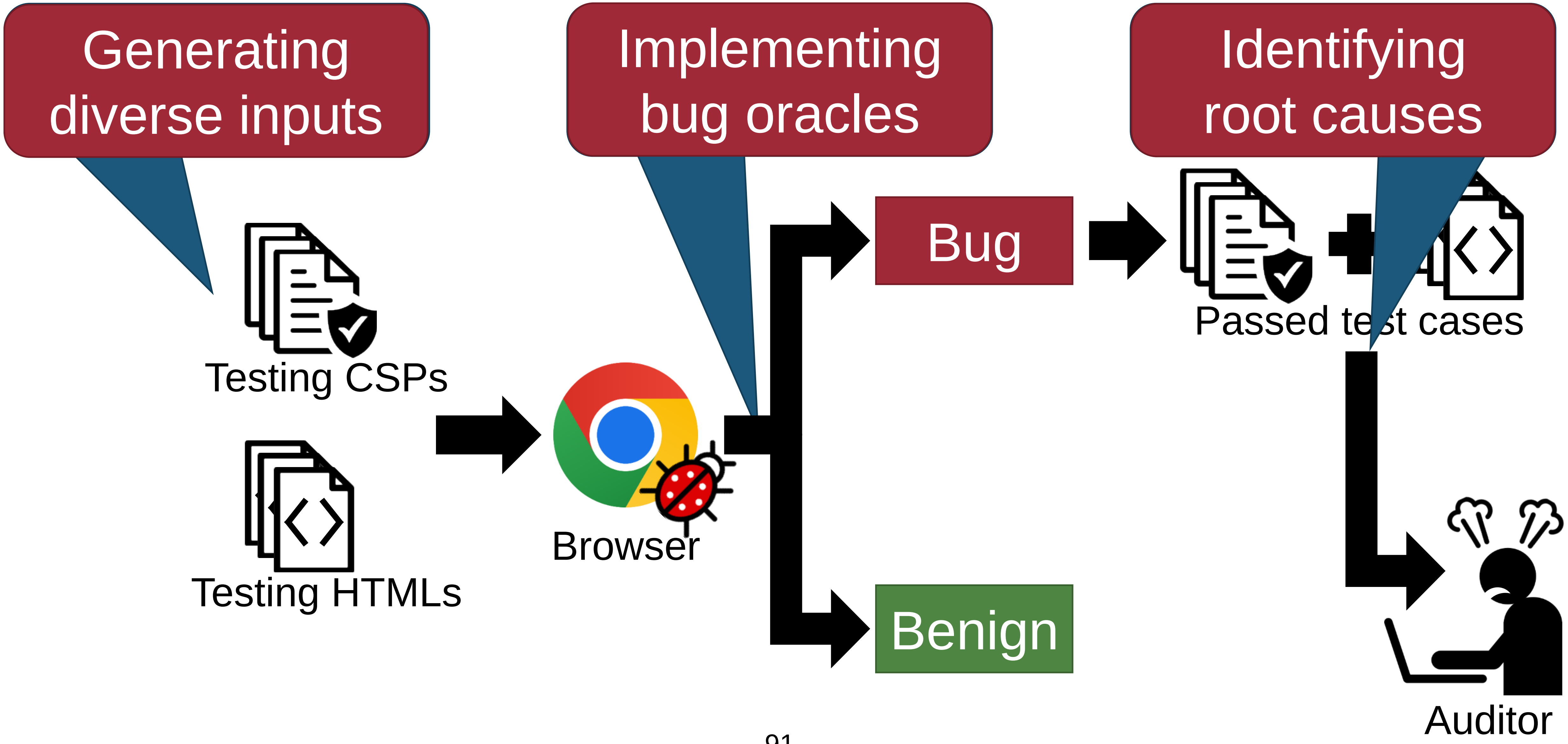
Keyword	none, unsafe-inline, unsafe-eval, self, strict-dynamic, unsafe-hashes
Host-source	Self URL, Allowed URL, *
Schemes	data:, blob:, http:, https:
Nonce-source	nonce-123
Hash-source	sha256-[HASH]

Generate **1,006 policies**

Content Security Problems?, *CCS '16*



Content Security Problems?, *CCS '16*

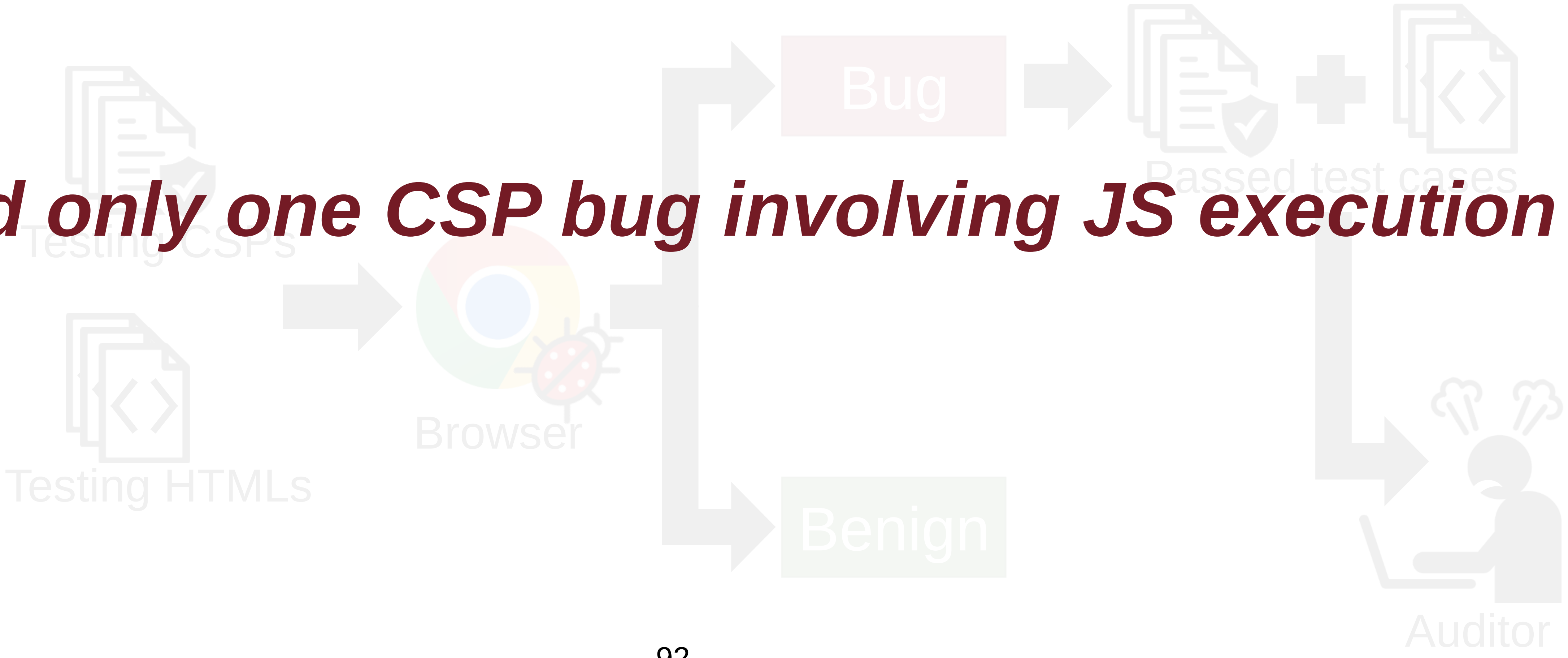


Content Security Problems?, *CCS '16*

Generating diverse inputs

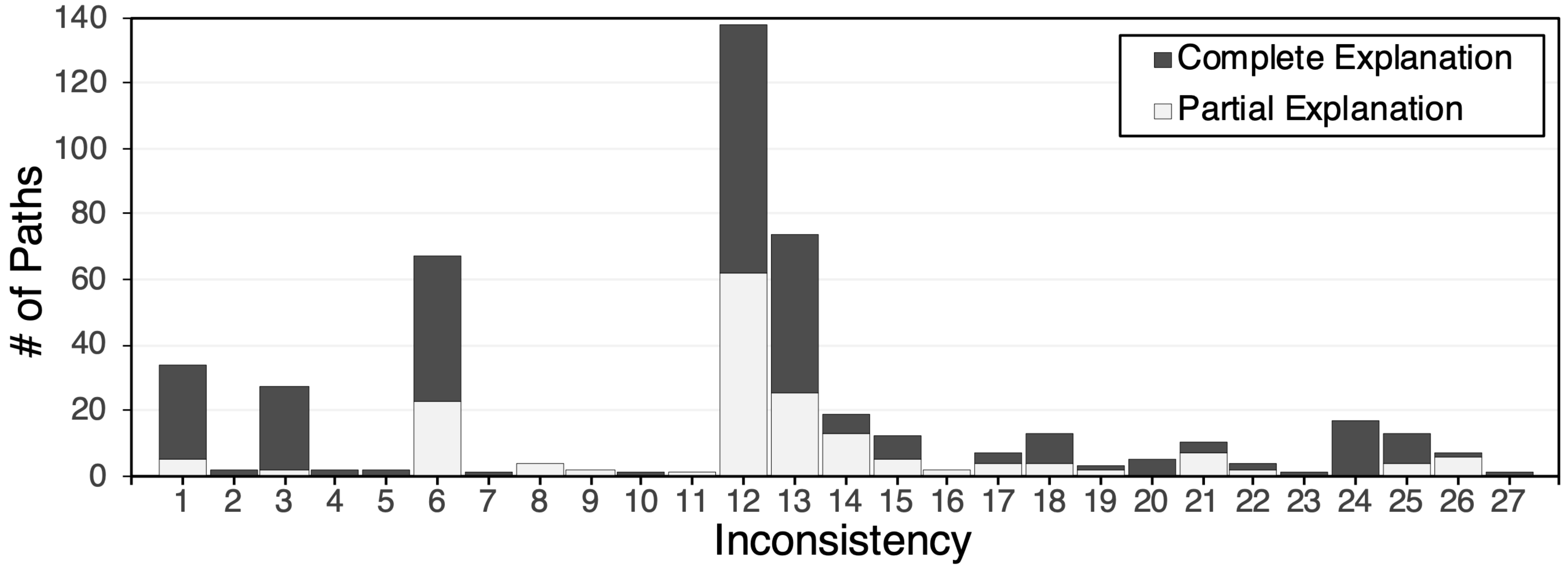
Implementing bug oracles

Identifying root causes



Found only one CSP bug involving JS execution

Evaluation – Decision Tree



Lesson #3: Specification Bugs

CSP specification

✓ When a CSP contains non-ASCII characters, the whole policy should be discarded

CSP

```
script-src http://a.com
           http://가.com
```

HTML

```
<script>attack()</script>
```

Expected behavior:
Whole policy should be discarded

Lesson #3: Specification Bugs

CSP specification

✓ When a CSP contains non-ASCII characters, the whole policy should be discarded

CSP

```
script-src http://a.com
           http://가.com
```

HTML

```
<script>attack()</script>
```

Expected behavior:
Whole policy should be discarded
➔ Inline script should be allowed?

Lesson #3: Specification Bugs

CSP specification

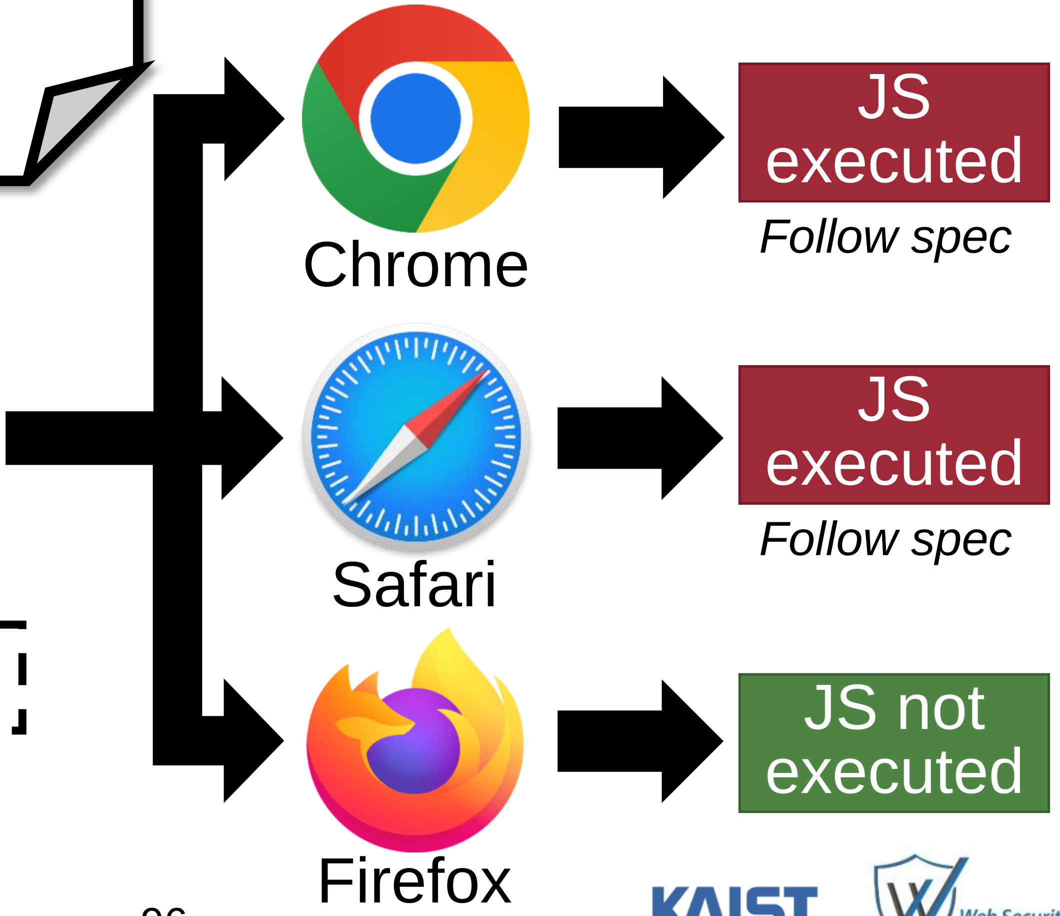
✓ When a CSP contains non-ASCII characters, the <u>whole policy should be discarded</u>

CSP

```
script-src http://a.com
           http://가.com
```

HTML

```
<script>attack()</script>
```



Case Study: Incorrect CSP Inheritance

CSP specification

✓ Sources from a local scheme (e.g., `javascript`) should inherit the CSP of their parent page

CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Expected behavior:
The nonce-protected JS will be allowed
➔ Inline script should be blocked

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Documents from *local schemes* will **inherit the CSP** of their parent page

CSP

```
script-src 'nonce-123'
```

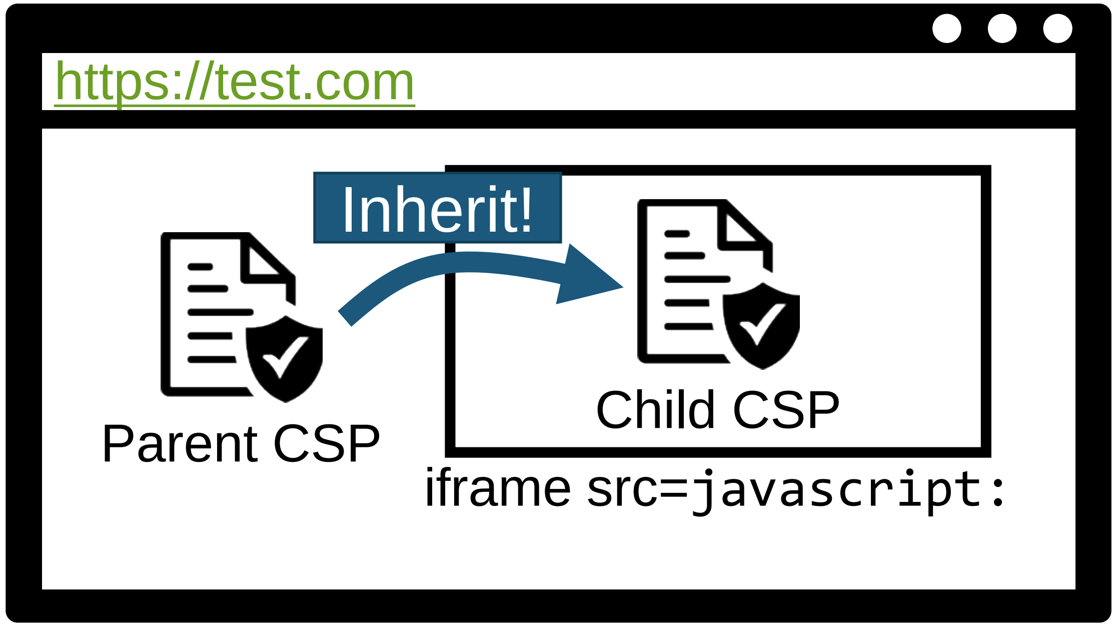
HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Case Study: Incorrect CSP Inheritance

CSP specification

- ✓ Sources from a local scheme (e.g., `javascript`) should <u>inherit the CSP of their parent page</u>



Case Study: Incorrect CSP Inheritance

CSP specification

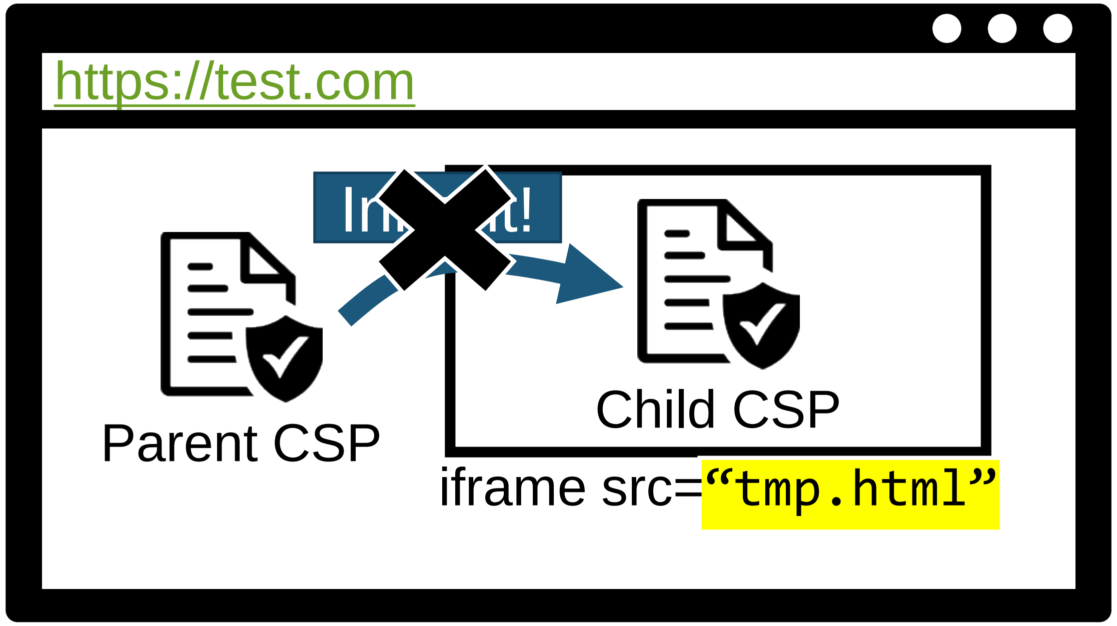
✓ Documents from local schemes will inherit the CSP of the parent document

CSP

```
script-src ‘nonce-123’
```

HTML

```
<iframe id=“z” src=“tmp.html” />
<script nonce=123>
  z.addEventListener(“load”,() => {
      z.src=“javascript:attack()”
  ;});
</script>
```



Case Study: Incorrect CSP Inheritance

CSP specification

✓ Sources from a local scheme (e.g., `javascript`) should inherit the CSP of their parent page

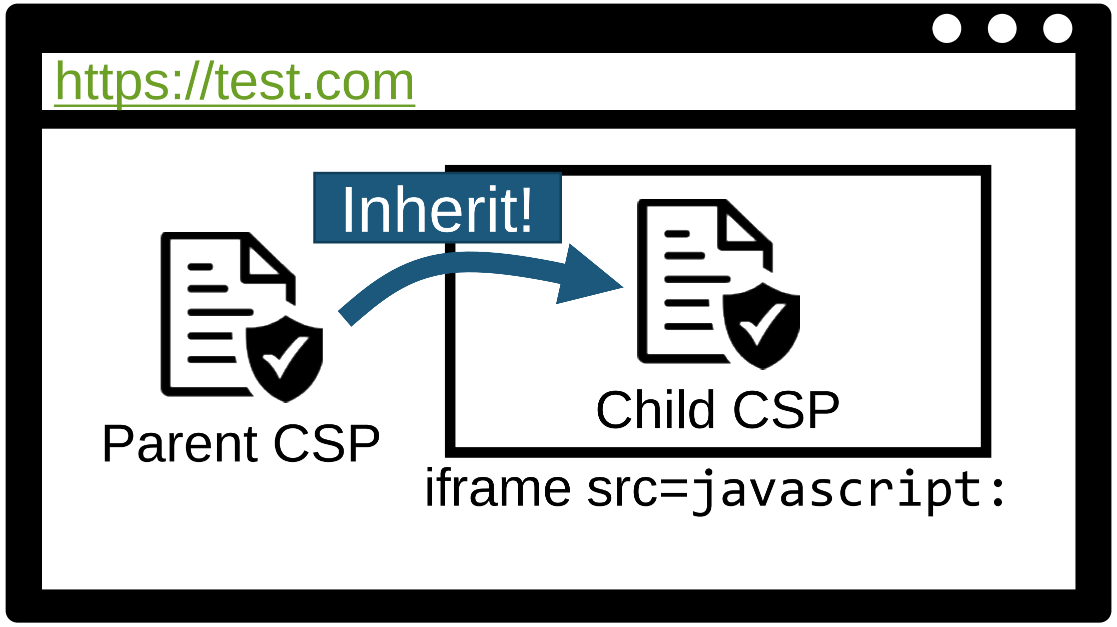
CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Dynamic page redirection



Case Study: Incorrect CSP Inheritance

CSP specification

✓ Sources from a local scheme (e.g., `javascript`) should <u>inherit the CSP of their parent page</u>

CSP

```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Expected behavior:
The nonce-protected JS will be allowed

Case Study: Incorrect CSP Inheritance

CSP specification

✓ Sources from a local scheme (e.g., `javascript`) should inherit the CSP of their parent page

CSP

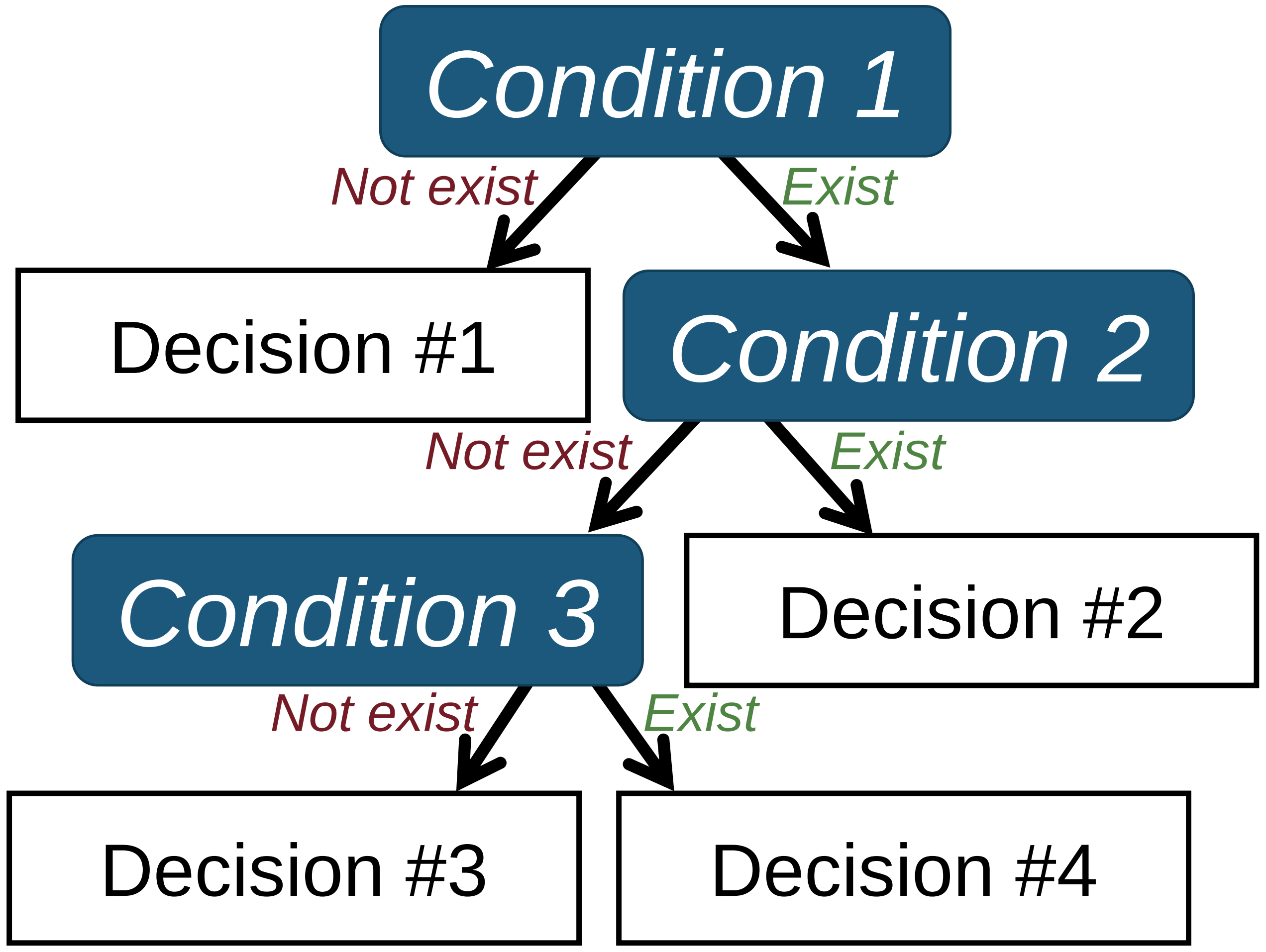
```
script-src 'nonce-123'
```

HTML

```
<iframe id="z" src="tmp.html" />
<script nonce=123>
  z.addEventListener("load",() => {
      z.src="javascript:attack()"
  ;});
</script>
```

Expected behavior:
The nonce-protected JS will be allowed
➔ Inline script should be blocked

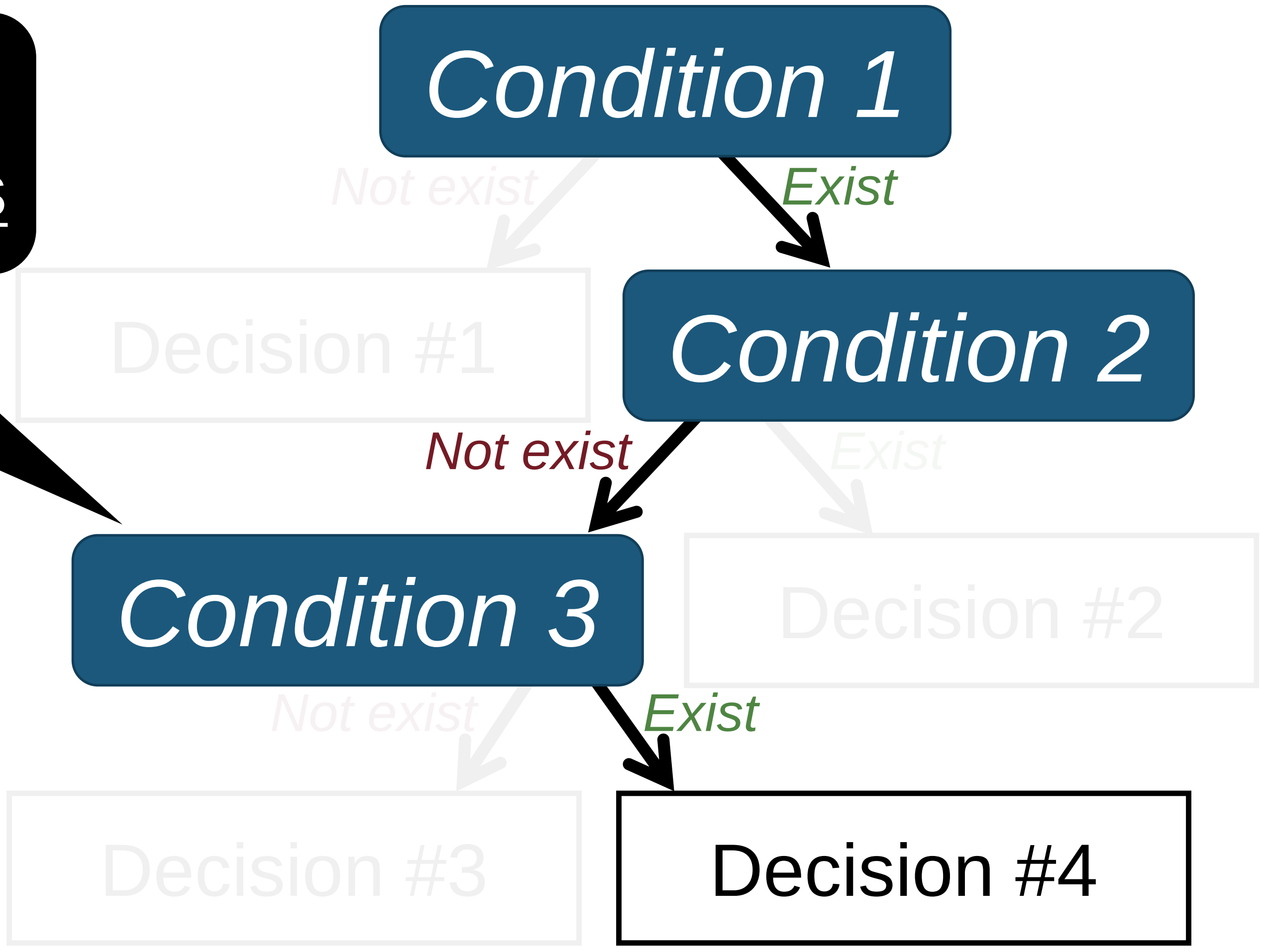
Decision Tree-based Root Cause Analysis



Decision tree

Decision Tree-based Root Cause Analysis

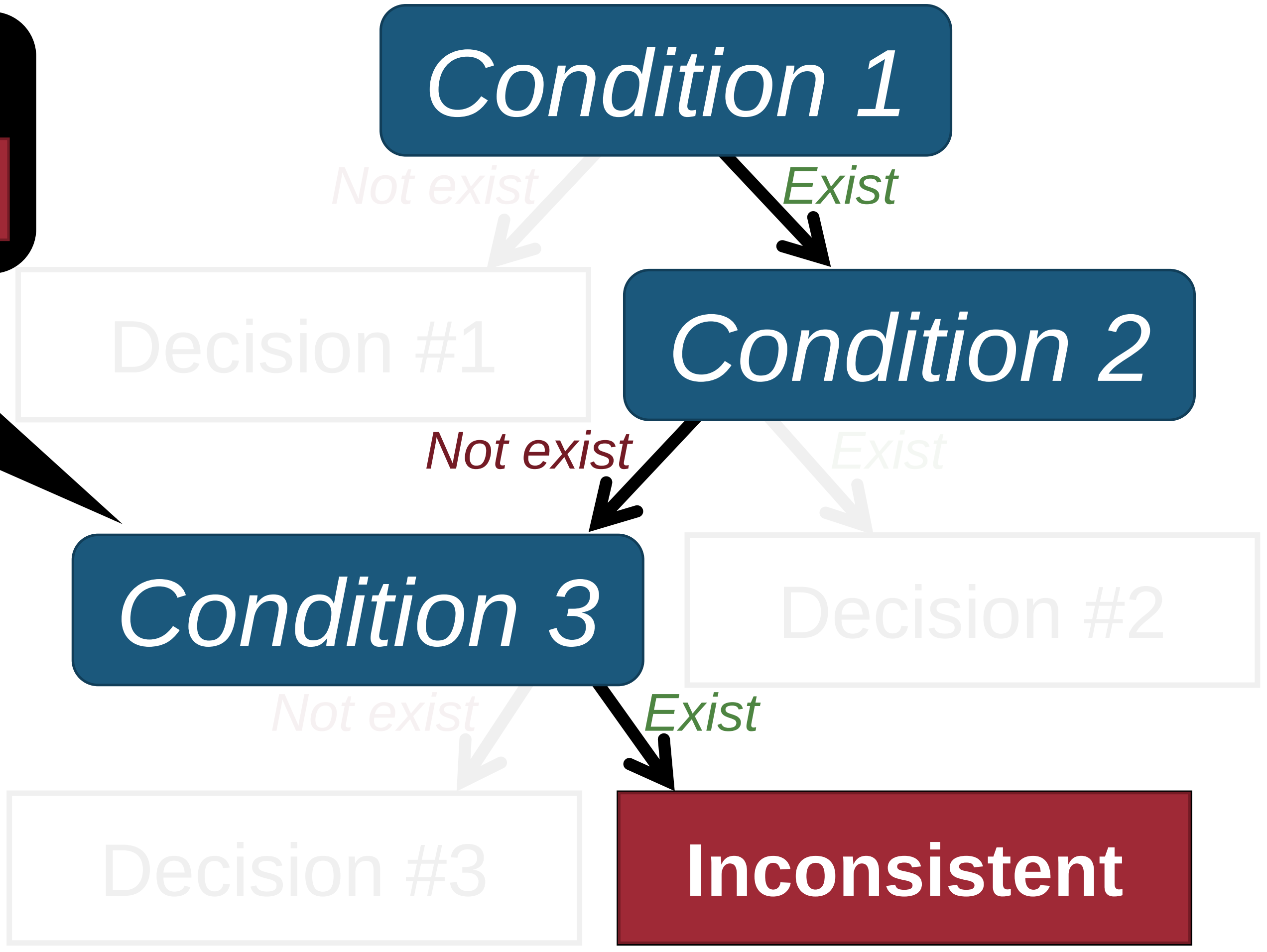
Human-readable conditions
leading to classification decisions



Decision tree

Decision Tree-based Root Cause Analysis

Human-readable conditions
leading to **inconsistent results**



Decision tree

Root Causes

Incorrect CSP inheritance
Incorrect hash handling
Non-ignored directive values
Non-supporting specific directives
Non-supporting specific directive values
Auto-enabling directive values by default
Auto-enabling directive values on specific conditions
Non-supporting CSP for specific status code
Incorrect handling of malformed CSPs
Allowing out-going request

Execute an arbitrary JS code (97%)

Sending a request to an arbitrary endpoint (3%)