Jalangi: A Dynamic Analysis Framework for JavaScript

Koushik Sen

University of California, Berkeley

Joint work with

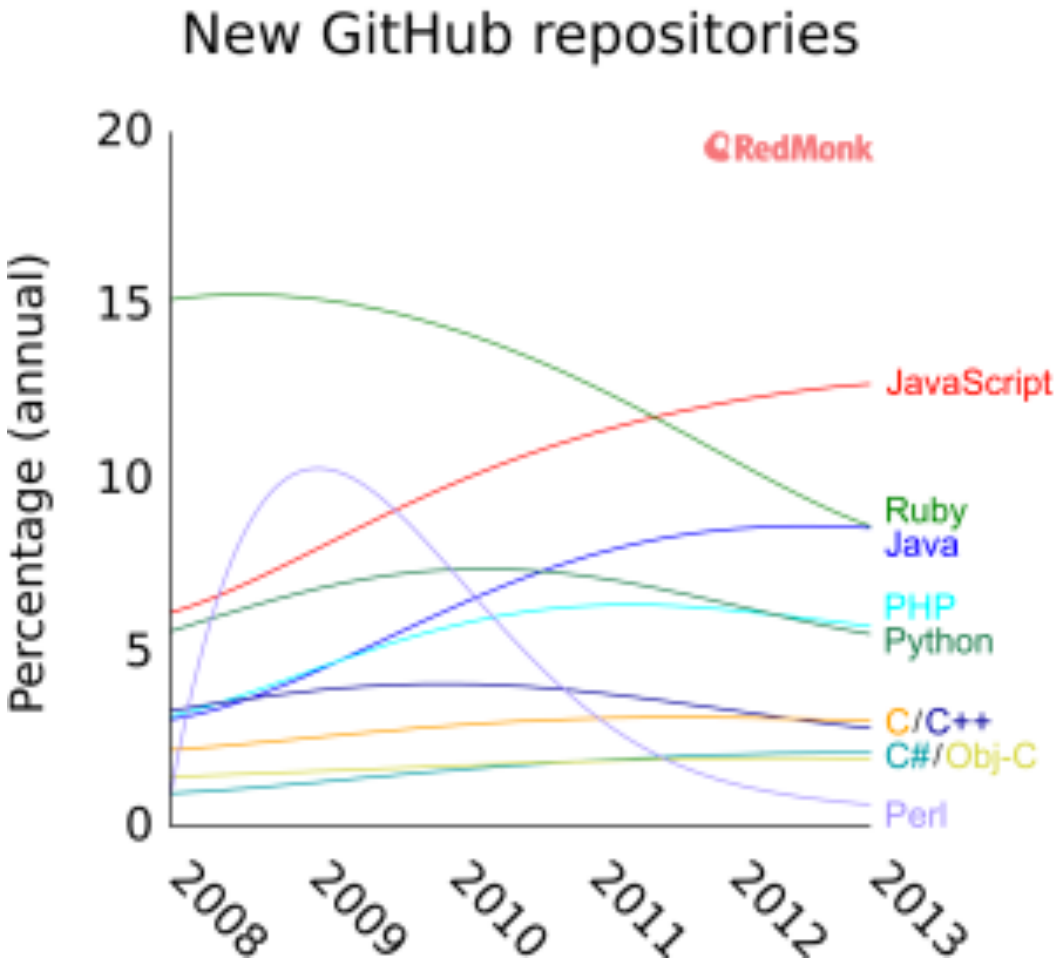
Tasneem Brutch, Satish Chandra, Simon Gibbs, Liang Gong,

Simon Jenson, Swaroop Kalasapur, Michael Pradel, Manu Sridharan

SAMSUNG RESEARCH AMERICA

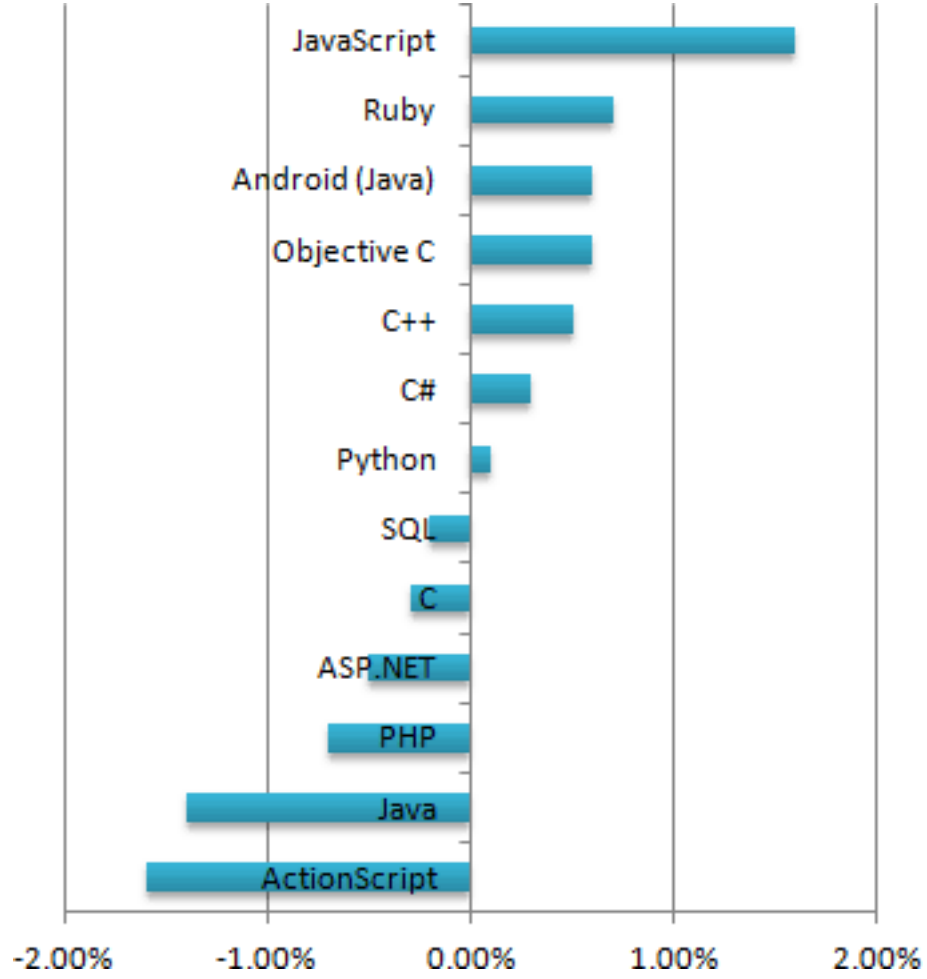
Why JavaScript?

- The RedMonk Programming Language Rankings (Popularity): January 2015
  - Based on projects hosted at GitHub and questions posted at StackOverflow



Why JavaScript?

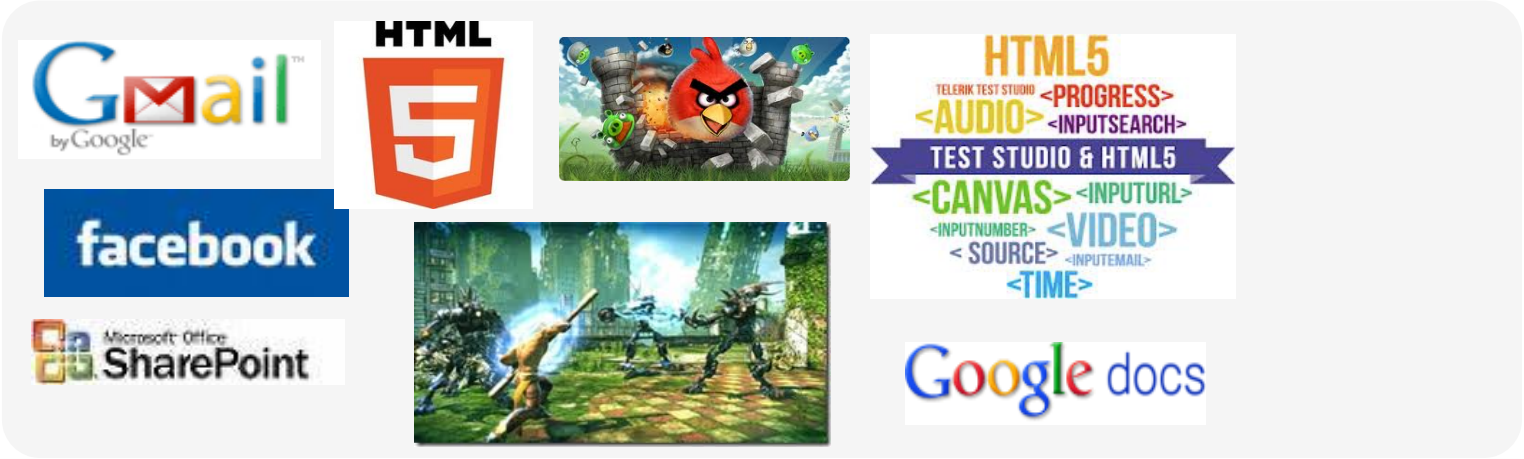
Growth in popularity (based on jobs available) from 2012 – 2013



Source: http://blog.learntoprogram.tv/five-resons-javascript-important-programming-language-learn/

Why JavaScript?

- Client-side JavaScript in Rich Web Applications



- Desktop Apps (Windows 8 and Gnome), Firefox OS, Tizen OS
- Server-side (node.js)
  - Paypal, Ebay, Uber, NYtimes, Linkedin, and many more
- Assembly Language for the Web: emscripten, coffeescript, TypeScript
- A language to implement DSL frameworks
  - Angular.js, Knockout.js, React.js

Why JavaScript?

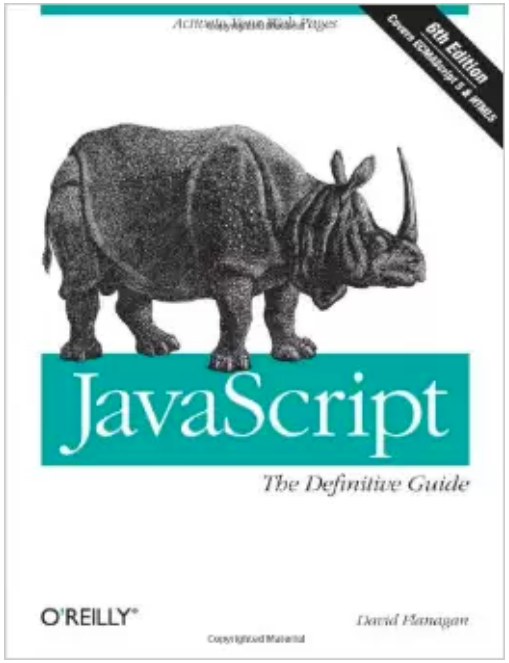
- Huge ecosystem of libraries and frameworks
- JavaScript has low learning curve
  - people can start coding and get results quickly
- No special installation/execution environment
  - Just use a modern browser
- JavaScript supports functional programming
  - higher order functions
- Modern JavaScript VMs are fast

Atwood's Law

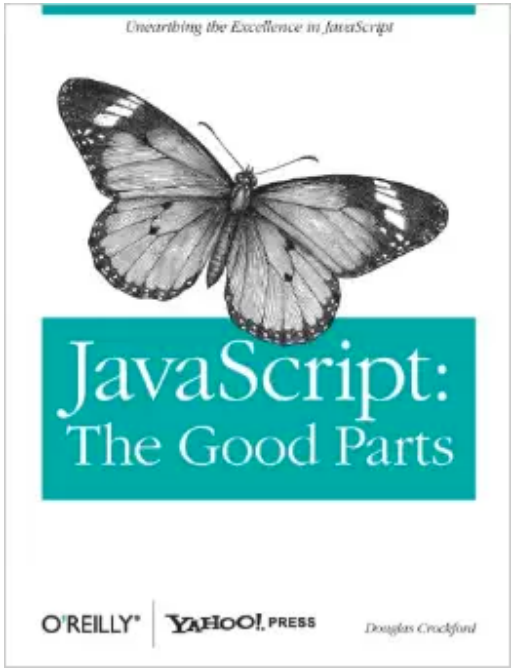
"Any application that can be written in JavaScript, will eventually be written in JavaScript."

Why Tools for JavaScript?

- JavaScript has its quirks (many)



1096 pages



176 pages

Why Tools for JavaScript?

```
var x = “1”;
++x;
console.log(x);
```

```
var x = “1”;
x += 1;
console.log(x);
```

Why Tools for JavaScript?

```
var x = “1”;
++x
console.log(x);

// prints 2
```

```
var x = “1”;
x += 1;
console.log(x);

// prints 11
```

Why Tools for JavaScript?

- Easy to introduce bugs: correctness, performance, memory
  - Degrees of equality == vs. ===
- Loosely-typed
  - forgiving: implicit type conversion
  - tries hard to execute without throwing exception
    - Like HTML
- Highly reflective
  - eval any dynamically created string
- Asynchronous programming


MICHIGAN
MEGA
MILLIONS
WITH MEGAPLIER.
JACKPOT IS NOW
undefined
MEGAPLIER INCREASES NON-JACKPOT PRIZES ONLY.
ODDS OF WINNING JACKPOT: 1 IN 259 MILLION.

www.ebay.com/itm/Matte-Black-PVD-Shark-Mesh-Watch-Band-Strap-fits-Seiko-18mm-20mm-22mm-24mm-/161475533202

- Nato Watch Straps
- Expansion Bands
- Straps for Cartier
- DASSARI
- Straps for Breitling
- Straps for Omega
- Straps for Bell & Ross
- Watch Parts
- Bands for Rolex
- Bands for Tissot
- Other

sizes

10 mm	12 mm	14 mm
15 mm	16 mm	18 mm
19 mm	20 mm	21 mm
22 mm	23 mm	24 mm
26 mm	28 mm	30 mm

colors



hot sellers

StrapsCo Genuine Patent Leather Watch Strap Womens Band in Black

$NaN buy now

18mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

20mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

22mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling

$NaN buy now

Shark Mesh Watch Band Strap Breitling Navitimer Superocean 18mm

$NaN buy now

Matte Black PVD Shark Mesh Watch Band Strap fits Seiko 18mm 20mm

$NaN buy now

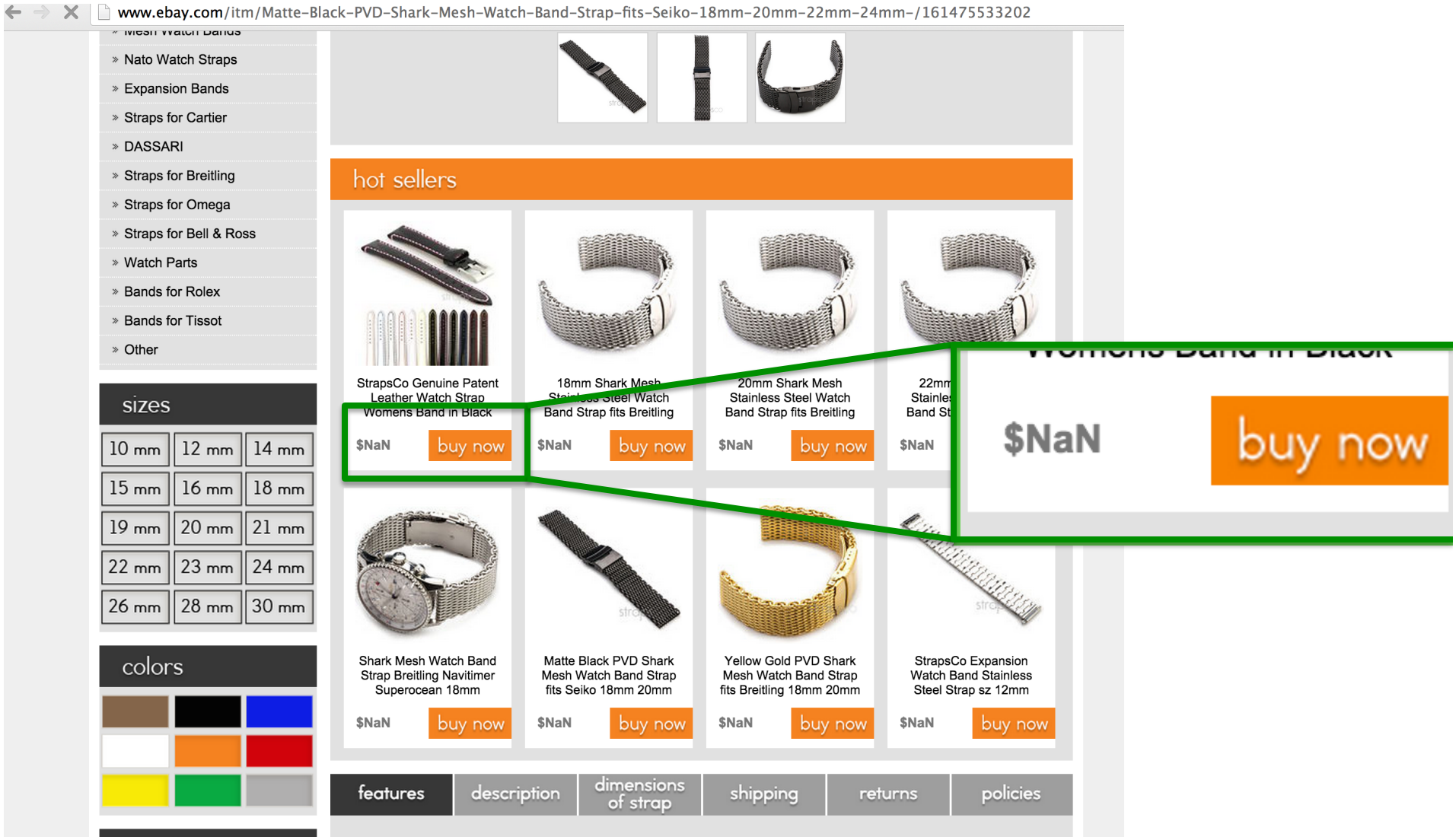
Yellow Gold PVD Shark Mesh Watch Band Strap fits Breitling 18mm 20mm

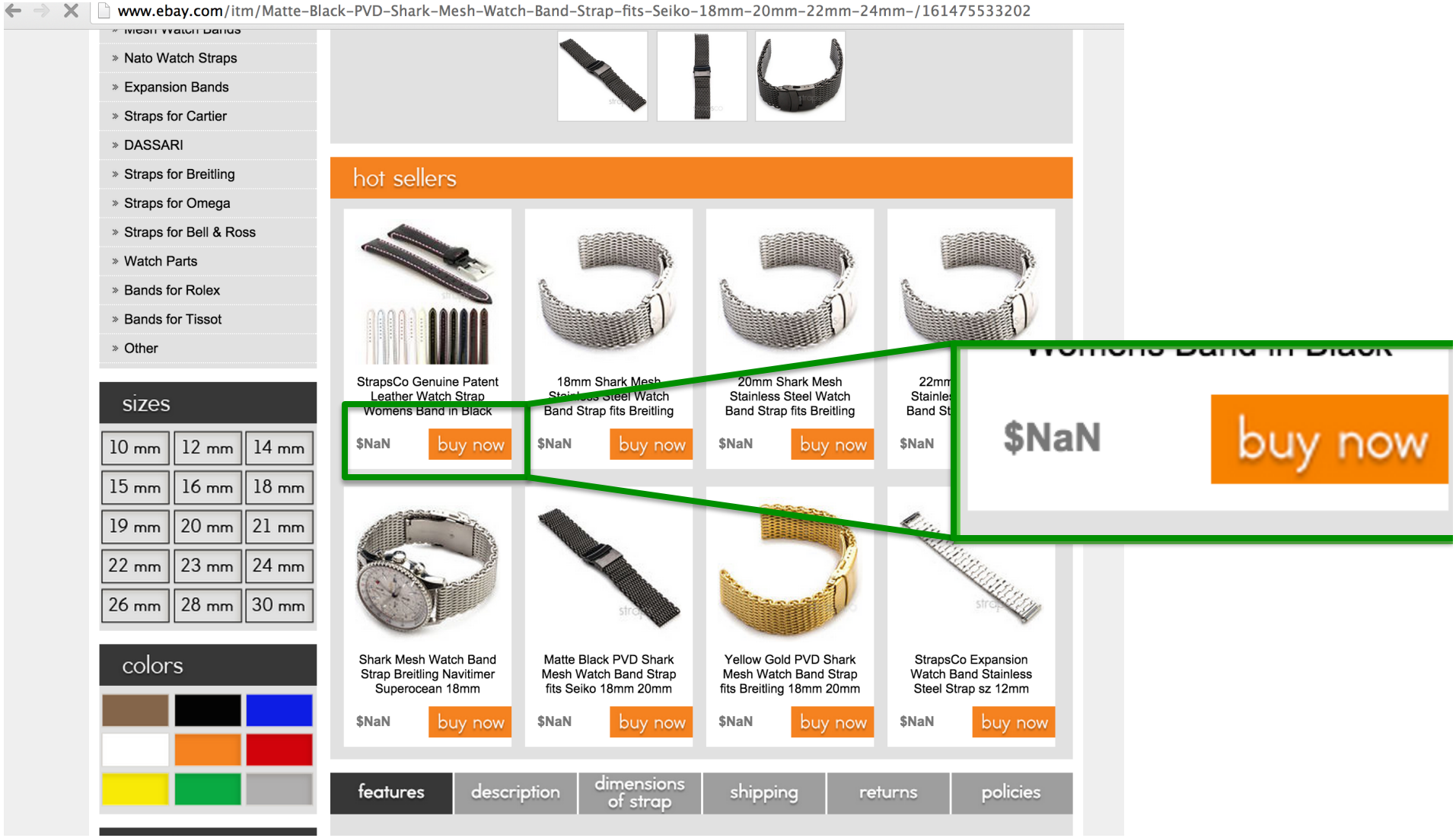
$NaN buy now

StrapsCo Expansion Watch Band Stainless Steel Strap sz 12mm

$NaN buy now

features | description | dimensions of strap | shipping | returns | policies


www.ebay.com/itm/Matte-Black-PVD-Shark-Mesh-Watch-Band-Strap-fits-Seiko-18mm-20mm-22mm-24mm-/161475533202
Nato Watch Straps
Expansion Bands
Straps for Cartier
DASSARI
Straps for Breitling
Straps for Omega
Straps for Bell & Ross
Watch Parts
Bands for Rolex
Bands for Tissot
Other
sizes
10 mm
12 mm
14 mm
15 mm
16 mm
18 mm
19 mm
20 mm
21 mm
22 mm
23 mm
24 mm
26 mm
28 mm
30 mm
colors
hot sellers
StrapsCo Genuine Patent Leather Watch Strap Womens Band in Black
$NaN
buy now
18mm Shark Mesh Band Strap fits Breitling
20mm Shark Mesh Stainless Steel Watch Band Strap fits Breitling
Shark Mesh Watch Band Strap Breitling Navitimer Superocean 18mm
Matte Black PVD Shark Mesh Watch Band Strap fits Seiko 18mm 20mm
Yellow Gold PVD Shark Mesh Watch Band Strap fits Breitling 18mm 20mm
StrapsCo Expansion Watch Band Stainless Steel Strap sz 12mm
features
description
dimensions of strap
shipping
returns
policies



Arithmetic operations on non-numbers may yield NaNs

Tools for Bug Finding and Security Analysis

- Remarkable progress in program-analysis and constraint solving
  - Commercial tools: Coverity, Klocwork, Grammatech, TotalView, Parallocity, Static Device Verifier from Microsoft, WALA at IBM
  - Open-source tools: GDB, lint, FindBugs, Valgrind
  - Academic tools: SLAM, BLAST, ESP, JPF, Bandera, Saturn, MAGIC, DART, CUTE, jCUTE
  - Mostly focused on C/C++ and Java programs
- Hardly any software quality tool for JavaScript and HTML5
  - Static analysis is difficult for dynamic languages

Jalangi

A powerful browser-independent (dynamic) analysis framework for JavaScript

https://github.com/Samsung/jalangi2

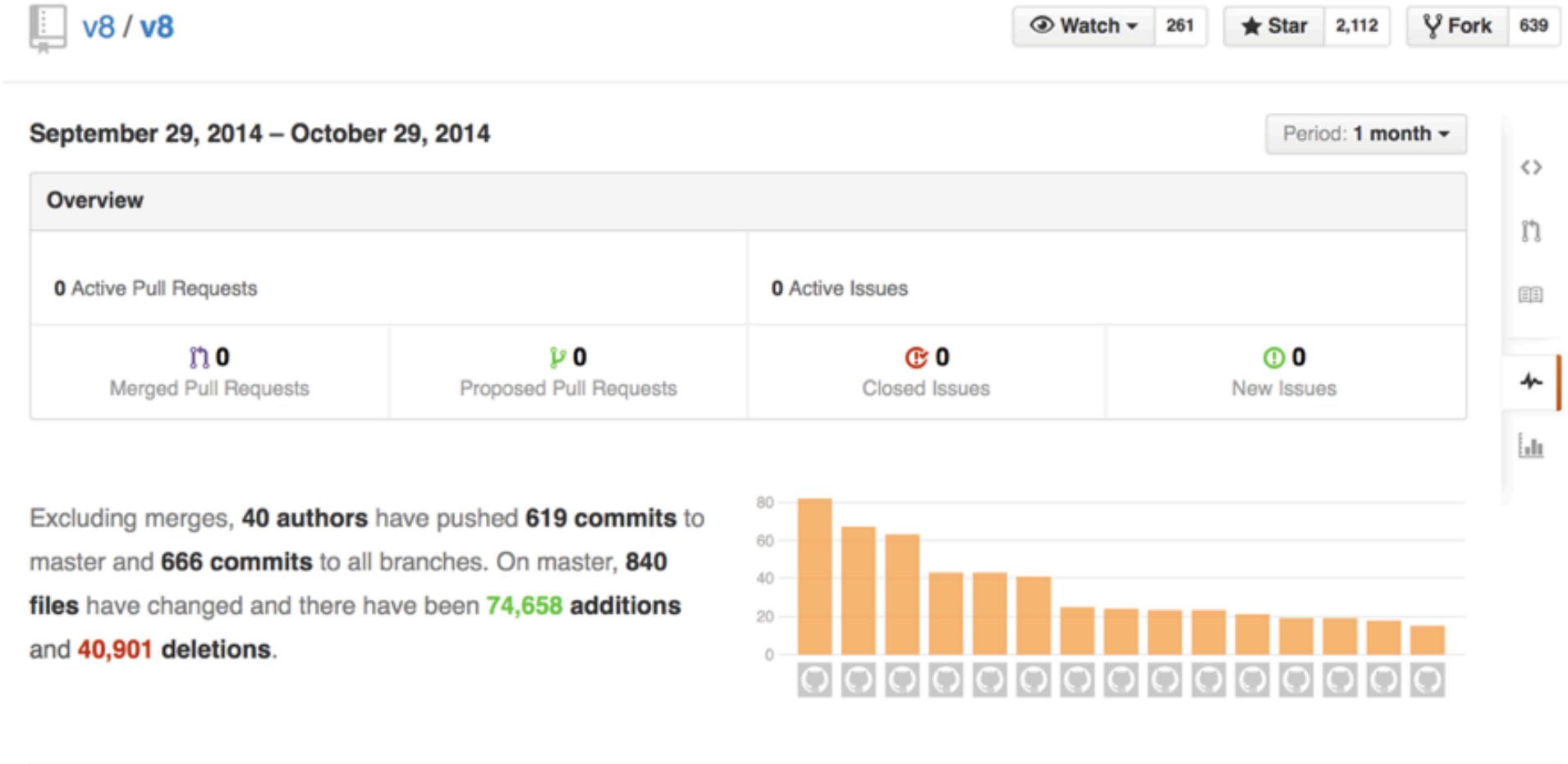
- Jalangi: A selective record-replay and dynamic analysis framework for JavaScript. Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. In ESEC/FSE, 2013.

Jalangi: Goals and Requirements

- Framework for Dynamic and hybrid Static/Dynamic analysis
  - supports symbolic execution, memory analysis, runtime type analysis, value tracking, taint tracking, performance analysis
- Handle ALL dynamic features
  - not OK to ignore eval, new Function
- Independent of browser
  - source-to-source code instrumentation
  - instrumented program when executed performs analysis
- Easy Implementation of Dynamic Analysis
  - Observe an execution passively: (conventional dynamic analysis)
  - Modify semantics/values
  - Repeatedly execute arbitrary paths within a function

Why not Modify a Browser?

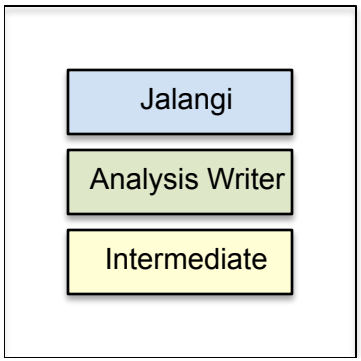
- Hard to keep up with browser development
- Harder to get people to use of customized browser



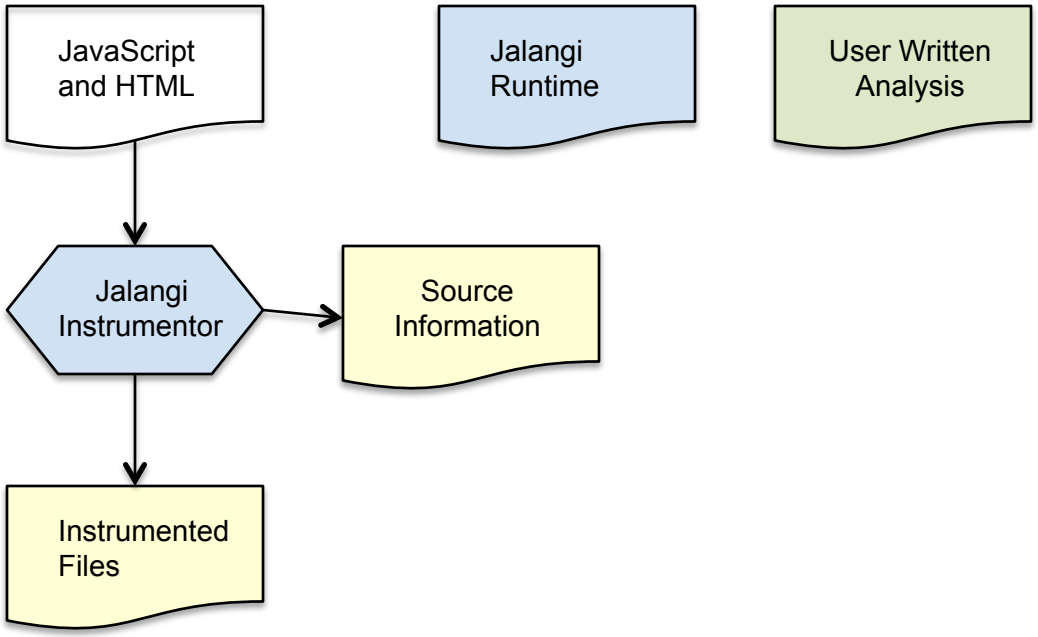
Jalangi 1 and 2

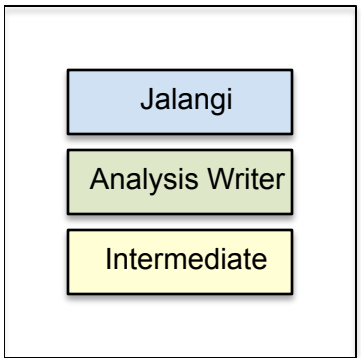
- Jalangi 1:
  - https://github.com/SRA-SiliconValley/jalangi
  - record execution and replay to perform analysis
  - Shadow values (wrapped objects)
  - No longer supported
- Jalangi 2:
  - https://github.com/Samsung/jalangi2
  - no record/replay or shadow values
  - active development

How Jalangi Works?

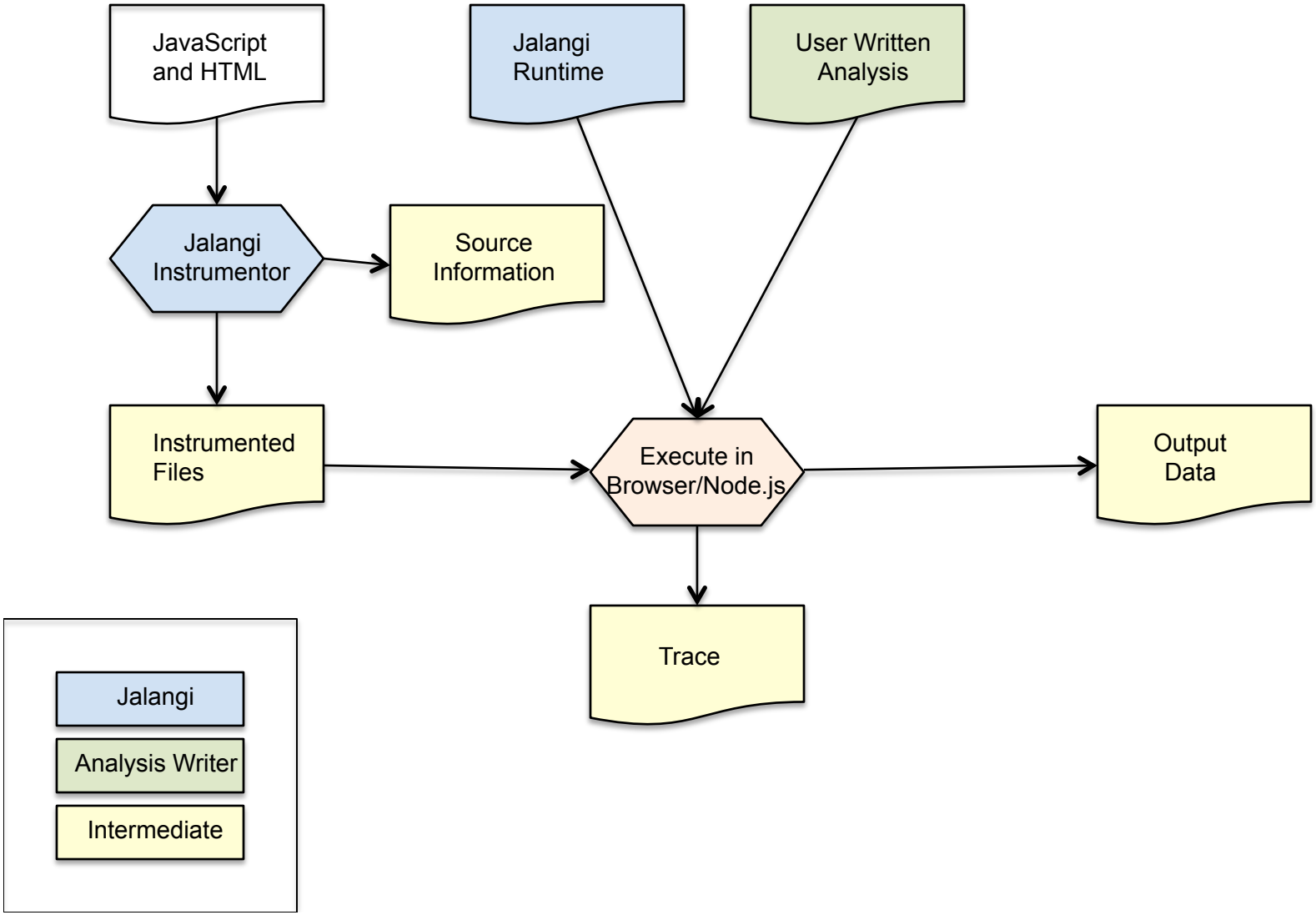


How Jalangi Works?

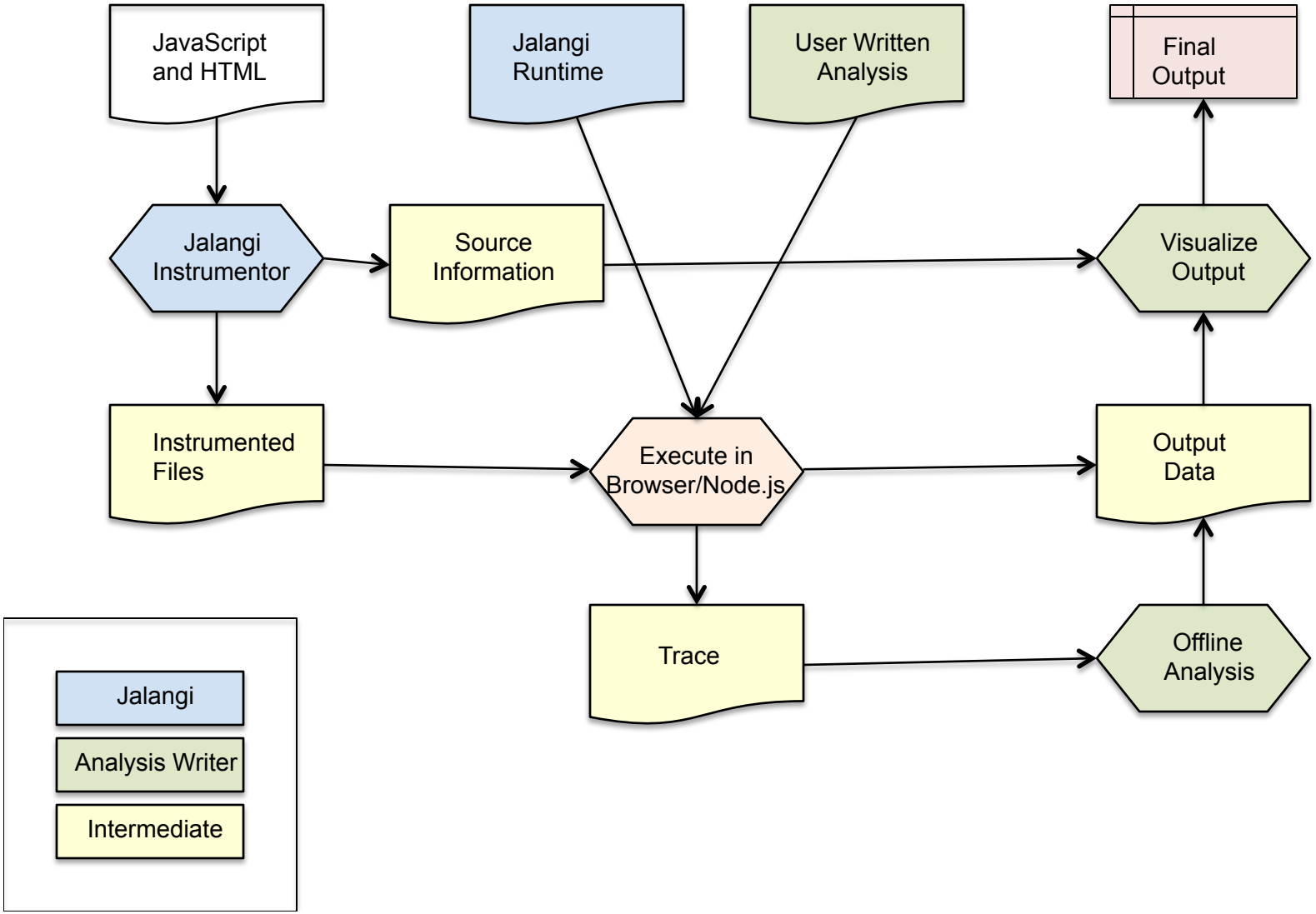




How Jalangi Works?



How Jalangi Works?



Jalangi Instrumentation (simplified)

x = y + 1	⇒	x = Write(“x”, Binary(‘+’,Read(“y”, y), Literal(1), x)
a.f = b.g	⇒	PutField(Read(“a”, a), “f”, GetField(Read(“b”, b), “g”))
if (a.f()) …	⇒	if (Branch(Method(Read(“a”, a), “f”)())) …

Jalangi Runtime

```
function Binary(op, left, right, ...) {

        result = left op right;

        return result;
}
```

Jalangi Runtime

```
function Binary(op, left, right, ...) {

    var aret = analysis.binaryPre(op, left, write, ...);

        result = left op right;
    aret = analysis.binary(op, left, right, result, ...);

        return result;
}
```

Jalangi Runtime

```
function Binary(op, left, right, ...) {
    var skip = false;
    var aret = analysis.binaryPre(op, left, write, ...);
        if (aret) {
            op = aret.op;
            left = aret.left;
            right = aret.right;
            skip = aret.skip; }}
    if (!skip)
        result = left op right;
    aret = analysis.binary(op, left, right, result, ...);



        return result;
}
```

Jalangi Runtime

```
function Binary(op, left, right, ...) {
    var skip = false;
    var aret = analysis.binaryPre(op, left, write, ...);
        if (aret) {
            op = aret.op;
            left = aret.left;
            right = aret.right;
            skip = aret.skip; }}
    if (!skip)
        result = left op right;
    aret = analysis.binary(op, left, right, result, ...);
    if (aret)
        return aret.result;
    else
        return result;
}
```

Jalangi Callbacks

function **invokeFunPre** (iid, f, base, args, isConstructor, isMethod, functionIid);
function **invokeFun** (iid, f, base, args, result, isConstructor, isMethod, functionIid);
function literal (iid, val, hasGetterSetter);
function forinObject (iid, val);
function declare (iid, name, val, isArgument, argumentIndex, isCatchParam);
function getFieldPre (iid, base, offset, isComputed, isOpAssign, isMethodCall);
function getField (iid, base, offset, val, isComputed, isOpAssign, isMethodCall);
function putFieldPre (iid, base, offset, val, isComputed, isOpAssign);
function putField (iid, base, offset, val, isComputed, isOpAssign);
function read (iid, name, val, isGlobal, isScriptLocal);
function write (iid, name, val, lhs, isGlobal, isScriptLocal);
function _return (iid, val);
function _throw (iid, val);
function _with (iid, val);

function functionEnter (iid, f, dis, args);
function functionExit (iid, returnVal, wrappedExceptionVal);
function scriptEnter (iid, instrumentedFileName, originalFileName);
function scriptExit (iid, wrappedExceptionVal);
function binaryPre (iid, op, left, right, isOpAssign, isSwitchCaseComparison, isComputed);
function binary (iid, op, left, right, result, isOpAssign, isSwitchCaseComparison, isComputed);
function unaryPre (iid, op, left);
function unary (iid, op, left, result);
function conditional (iid, result);
function instrumentCodePre (iid, code);
function instrumentCode (iid, newCode, newAst);
function endExpression (iid);
function endExecution();
function runInstrumentedFunctionBody (iid, f, functionIid);
function onReady (cb);

- Each analysis needs to implement a subset of these callbacks.
- Multiple analyses classes can be chained

function binaryPre (iid, op, left, right, isOpAssign, isSwitchCaseComparison, isComputed**);**

function binary (iid, op, left, right, result, isOpAssign, isSwitchCaseComparison, isComputed**);**

Sample analysis:
check if undefined is concatenated with a string

```
(function (sandbox) {
  function MyAnalysis () {

    this.binary = function(iid, op, left, right, result){
      if (op === '+' && typeof result==='string' &&
          (left===undefined || right===undefined))
              console.log("Concatenated undefined with string at "+
                          J$.iidToLocation(J$.sid, iid));
    }
  }
  sandbox.analysis = new MyAnalysis();
}(J$));
```

Sample analysis:
count branches

```
(function (sandbox) {
  var trueBranches = {};
  var falseBranches = {};

  function MyAnalysis() {

    this.conditional(iid, result) {
      var id = J$.getGlobalIID(iid);
      if (result)
        trueBranches[id]++;
      else
        falseBranches[id]++;
    }

    this.endExecution = function () {
      print(trueBranches, "True");
      print(falseBranches, "False");
    }
  }
  sandbox.analysis = new MyAnalysis();
}(J$));

function print(map, str) {
  for (var id in map)
    if (map.hasOwnProperty(id)){
      console.log(str+ " branch taken at " +
        J$.iidToLocation(id)+ " " +maps[id] +
        " times";
    }
 }
```

Sample analysis:
count number of objects allocated at each site

```
(function (sandbox) {
  var allocCount= {};
  function MyAnalysis() {
    this.literal = function (iid, val) {
      var id = J$.getGlobalIID(iid);
      if (typeof val === 'object')
        allocCount[id]++;
    };
    this.invokeFunPre = function (iid, f,
                base, args, isConstructor) {
      var id = J$.getGlobalIID(iid);
      if (isConstructor)
        allocCount[id]++;
    };
    this.endExecution = function () {
      print(allocCount);
    }
  }
  sandbox.analysis = new MyAnalysis();
}(J$));

function print(map) {
  for (var id in map)
    if (map.hasOwnProperty(id)){
      console.log(" Object allocated at " +
        J$.iidToLocation(id)+"="+maps[id]);
    }
}
```

Sample analysis (modify semantics):
interpret '-' as '*'

```
(function (sandbox) {
  function MyAnalysis() {
    this.binaryPre = function (iid, op, left, right) {
      if (op === '-')
        return {op: op, left: left, right: right, skip: true};
    };

    this.binary = function (iid, op, left, right, result) {
      if (op === '-')
        return {result: left * right};
    };
  }
  sandbox.analysis = new MyAnalysis();
}(J$));
```

Sample analysis (modify semantics):
skip execution of an evil function

```
(function (sandbox) {
  function MyAnalysis() {

    this.invokeFunPre = function (iid, f, base, args) {
        if (f === evilFunction)
            return {f: f, base: base, args: args, skip: true};
    };
  }
  sandbox.analysis = new MyAnalysis();
}(J$));
```

Sample analysis (modify semantics):
loop a function body

```
function loop(n) {
   var ret = ret? ret-1: n;
   // do something
   console.log(ret);
   return ret;
}
loop(10);
```

Sample analysis (modify semantics):
loop a function body


```
function loop(n) {
    var ret = ret? ret-1: n;
    // do something
    console.log(ret);
    return ret;
}
loop(10);
```

Sample analysis (modify semantics):
loop a function body

```
(function (sandbox) {
   function MyAnalysis() {
      this.functionExit = function (iid, rv, ex) {
         return {returnVal: rv, wrappedExceptionVal: ex, isBacktrack: rv?true:false};
      };
   }
   sandbox.analysis = new MyAnalysis();
}(J$));
-------------------------------- Program ---------------------------------
function loop(n) {
   var ret = ret? ret-1: n;
   // do something
   console.log(ret);
   return ret;
}
loop(10);
```

Sample analysis (modify semantics):

MultiSE: Multi-Path Symbolic Execution using Value Summaries

- Symbolic execution
- Explore all paths in a function
  - but merge state from all paths before exiting the function
- Override default semantics to perform symbolic evaluation
- Backtrack within a function until all paths are explored
- Custom semantics and backtracking
  - for simple abstract interpretation
  - for simple dataflow analysis

Jalangi 2 Summary

- Observe an execution and collect information
- Change values used in an execution
- Change semantics of operators
- Explore arbitrary path in a function
- Re-execute the body of a function repeatedly
- Maintain your own (abstract) state and call stack
- 3x-100x slowdown

Serious Analyses with Jalangi

- **"TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript,"**
  - Michael Pradel and Parker Schuh and Koushik Sen (ICSE'15)
- **"JITProf: Pinpointing JIT-unfriendly JavaScript Code,"**
  - Liang Gong and Michael Pradel and Koushik Sen (To appear at ESEC/FSE'15)
- **"MemInsight: Platform-Independent Memory Debugging for JavaScript,"**
  - Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (To appear at ESEC/FSE'15)
- **"DLint: Dynamically Checking Bad Coding Practices in JavaScript,"**
  - Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (To appear at ISSTA'15)
- **"MultiSE: Multi-Path Symbolic Execution using Value Summaries,"**
  - Koushik Sen and George Necula and Liang Gong and Wontae Choi, (To appear at ESEC/FSE'15)
- **"The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"**
  - Michael Pradel and Koushik Sen (To appear at ECOOP'15)
- **"EventBreak: Analyzing the Responsiveness of User Interfaces through Performance-Guided Test Generation,"**
  - Michael Pradel and Parker Schuh and George Necula and Koushik Sen (OOPSLA'14)

Serious Analyses with Jalangi

- **"TypeDevil: Dynamic Type Inconsistency Analysis for JavaScript,"**
  - Michael Pradel and Parker Schuh and Koushik Sen (ICSE'15)
- **"JITProf: Pinpointing JIT-unfriendly JavaScript Code,"**
  - Liang Gong and Michael Pradel and Koushik Sen (To appear at ESEC/FSE'15)
- **"MemInsight: Platform-Independent Memory Debugging for JavaScript,"**
  - Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (To appear at ESEC/FSE'15)
- **"DLint: Dynamically Checking Bad Coding Practices in JavaScript,"**
  - Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (To appear at ISSTA'15)
- **"MultiSE: Multi-Path Symbolic Execution using Value Summaries,"**
  - Koushik Sen and George Necula and Liang Gong and Wontae Choi, (To appear at ESEC/FSE'15)
- **"The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"**
  - Michael Pradel and Koushik Sen (To appear at ECOOP'15)
- **"EventBreak: Analyzing the Responsiveness of User Interfaces through Performance-Guided Test Generation,"**
  - Michael Pradel and Parker Schuh and George Necula and Koushik Sen (OOPSLA'14)

Michael Pradel and Parker Schuh and Koushik Sen (ICSE'15)

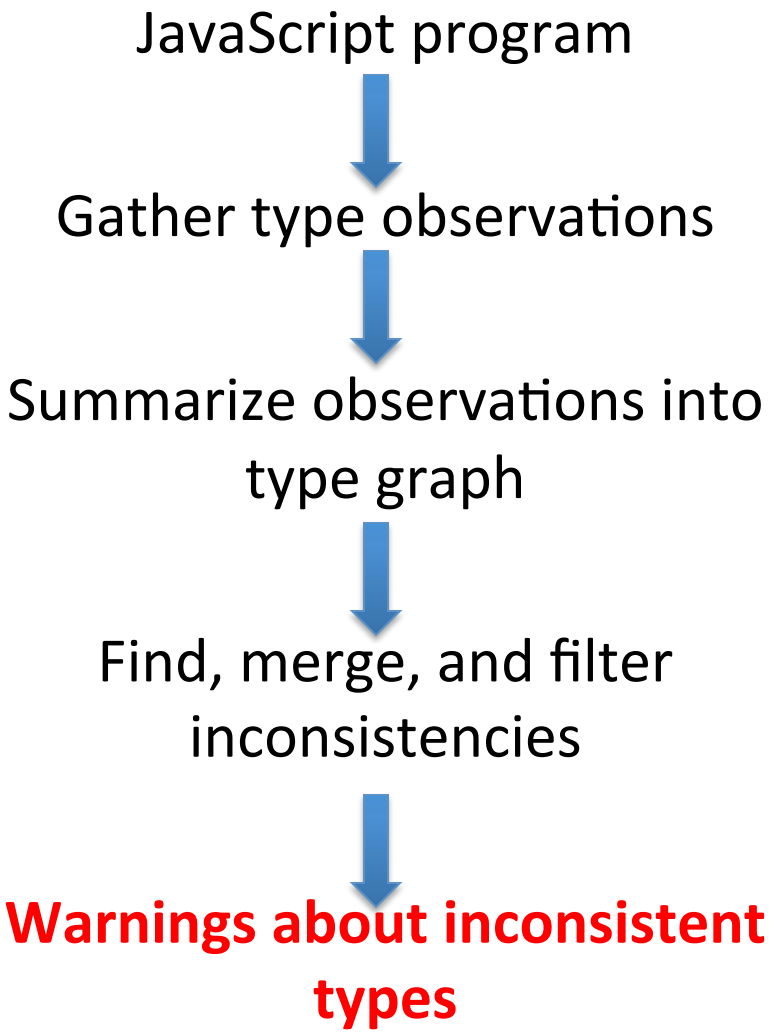
TYPEDEVIL: DYNAMIC TYPE INCONSISTENCY ANALYSIS FOR JAVASCRIPT (JALANGI POWERED)

Observations

- Most code follows implicit type rules
  - A single type per variable
  - A single type per object property
  - Functions have fixed signatures
- Many bugs are violations of these rules

TypeDevil

- Goals: find inconsistent types of
  - local variables
  - properties of objects
  - function signatures



Evaluation

- Setup:
  - Sunspider, Octane, and 7 web apps
- Main results:
  - Finds relevant problems:
    - 33 warnings, 15 are relevant
  - Pruning and merging is crucial:
    - 578 warnings → 33 warnings

Example

- SunSpider's regexp-dna:

```
var dnaOutputStr;
for (i in seqs) {
    dnaOutputStr += seqs[i].source;
}
```

Problem: Incorrect string value "undefinedGTAGG..."

Liang Gong, Michael Pradel, Koushik Sen (To appear at ESEC/FSE 2015)

JITPROF: PINPOINTING JIT-UNFRIENDLY JAVASCRIPT CODE (JALANGI POWERED)

JIT-Unfriendly Code Patterns

- JIT can compile JavaScript programs to efficient code
  - based on the premise that dynamic features are used in a regular and systematic way
- Programmers may use dynamic features in unexpected ways
  - prohibits profitable JIT optimizations
  - poor performance
  - no JIT compilation feedback to programmers
- Need a tool that can identify JIT-unfriendly code
  - JIT-aware profiler

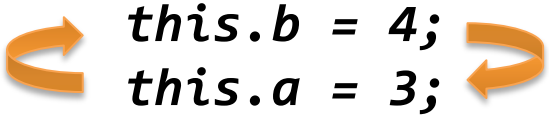
A Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

runtime
1.38s

A Motivating Example

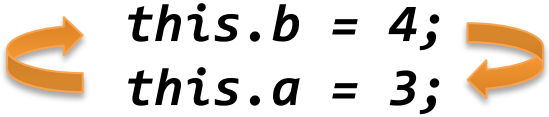
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```



runtime
1.38s

A Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```



20% performance boost

runtime
1.03s

A Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

Cause of poor performance:

- Hidden class and inline caching
- ***o*** has two layouts:
  offset of ***a*** in ***o*** can be ***0*** or ***1***

- JIT cannot replace ***o.a*** with ***o[0]*** or ***o[1]***

A Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.a = 3;
        this.b = 4;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

Better performance:

- Hidden class and inline caching
- ***o*** has one layout:
  offset of ***a*** in ***o*** is ***0***

- JIT cannot replace ***o.a*** with ***o[0]***

A Motivating Example

```
function Thing(flag) {
    if (!flag) {
        this.a = 3;
        this.b = 4;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

for (var i = 0; i < 1000000; i++) {
    var o = new Thing(i%2);
    result += o.a + o.b;
}
```

Better performance:

- Hidden class and inline caching
- ***o*** has one layout:
  offset of ***a*** in ***o*** is ***0***

- JIT cannot replace ***o.a*** with ***o[0]***

Hidden classes:
Map *in V8*
Shape *in SpiderMonkey*
Structure *in JavaScriptCore*

JIT Engines: Common Assumptions

- objects have fixed layout
- undefined properties or array elements are not accessed
- non-numeric values are not stored in mostly numeric arrays
- contiguous indices are used in arrays
- all properties of an object are initialized in constructors
- … and many more

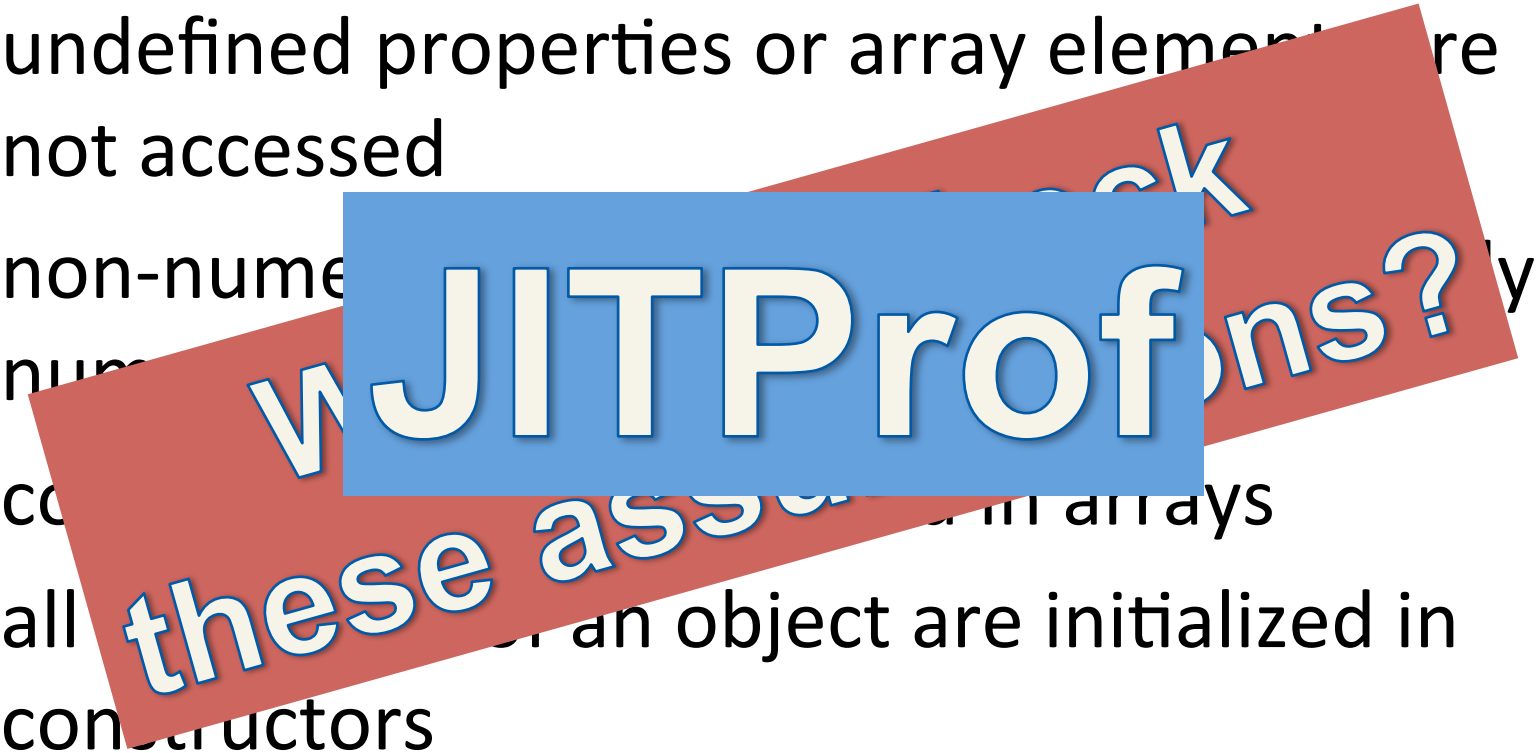
JIT Engines: Common Assumptions

- objects have fixed layout
- undefined properties or array elements are not accessed
- non-numeric values are not used in numeric operations
- constants are not overwritten in arrays
- all properties of an object are initialized in constructors
- … and many more

Why not check these assumptions?

JIT Engines: Common Assumptions

- objects have fixed layout
- undefined properties or array elements are not accessed
- non-numbers are not used in numeric expressions
- coercions are not used in arrays
- all properties of an object are initialized in constructors
- … and many more



JITProf Overview

- Checks seven patterns
  - one analysis class for each pattern
- Each analysis associates and updates meta-data with each object and each location
  - e.g. counters with each location
  - hidden class with objects
- Ranks warnings based on counters
  - number of time a code location involved in a pattern got executed

Evaluation

- Setup
  - JavaScript benchmarks: Octane and SunSpider
  - Top 50 websites (Alexa)
- Results
  - optimization opportunities in 15 out of 39 benchmark programs
  - 1.1% to 26.3% speedup on Firefox and Chrome

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var bin = Array();
  ...


  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var bin = Array();
  ...

  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on undefined value

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Pre-allocate array space to eliminate loading undeclared array elements

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on numbers

Pre-allocate array space to eliminate loading undeclared array elements

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on numbers

Performance boost:

1.5% 

31% 

Pre-allocate array space to eliminate loading undeclared array elements

Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen (To appear at ISSTA 2015)

DLINT: DYNAMICALLY CHECKING BAD CODING PRACTICES IN JAVASCRIPT (JALANGI POWERED)

Coding Problem Detection

- Lint-like static checkers
  - Rule-based checking
  - Detect common mistakes
  - Enforce coding conventions

The Problem

- Static checking is limited
  - Approximates behavior
  - Lint tools favor precision over soundness
- Common problems
  - Unknown types
  - Unknown aliases
  - Unknown code

Example

- *www.google.com/chrome*, included from Modernizr:

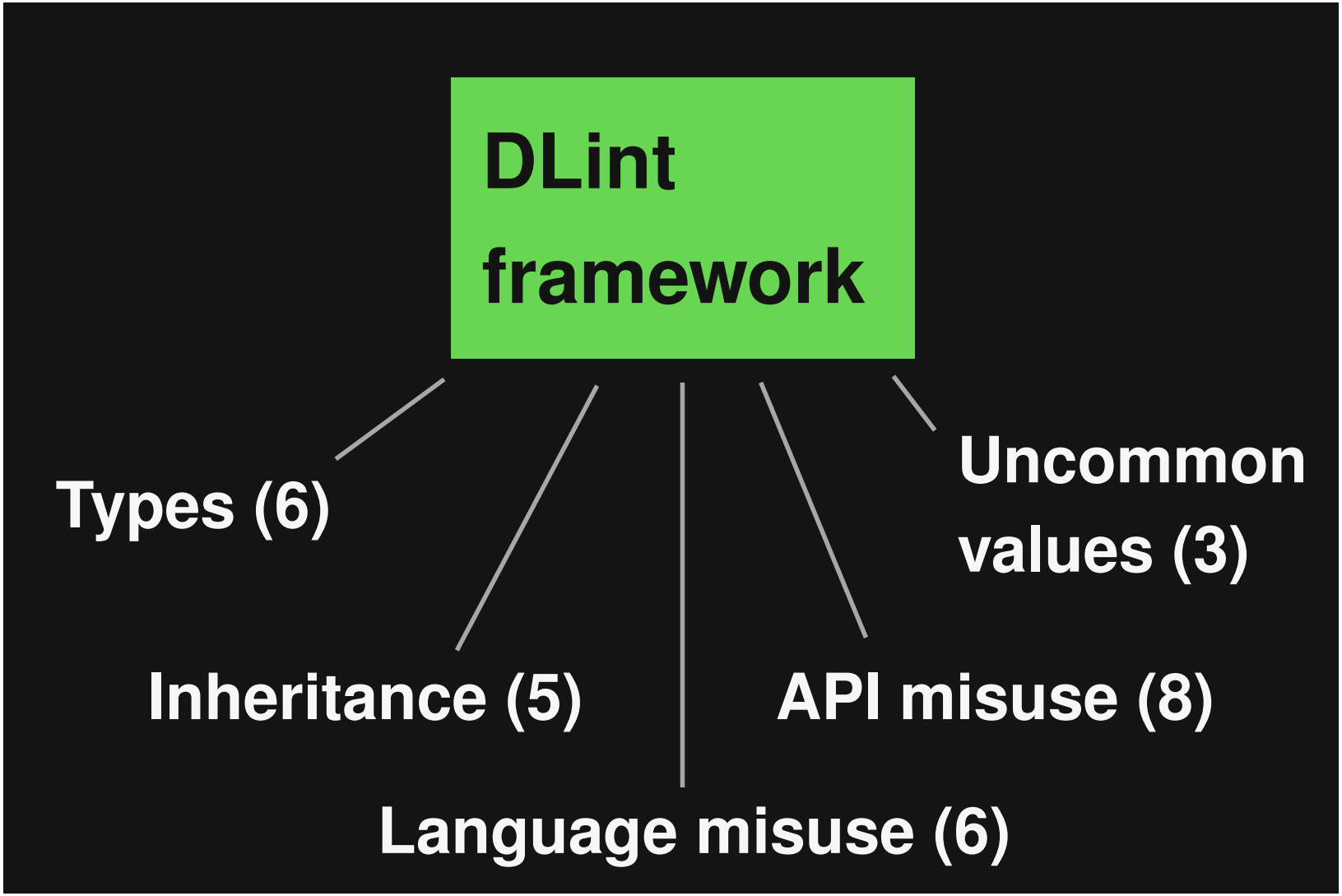
```
for (i in props) { // props is an array
    prop = props[i];
    before = mStyle.style[prop];
    ...
}
```

- Rule: Don't iterate over arrays with for-in.
- https://github.com/Modernizr/Modernizr/pull/1419

DLint

- DLint: Dynamic lint-like checking of code quality rules for JavaScript
- Formalized and implemented 28 rules
  - Counterparts of static rules
  - Additional rules
- Analyses written in Jalangi
- Empirical study: Comparison between static and dynamic checking

DLint Checkers



Type-Related

- Avoid accessing the *undefined* property.

```
var x; // undefined
var y = {};
y[x] = 23;
```

Language Misuse

- Avoid setting properties of primitives, which has no effect.

```
var fact = 42;
fact.isTheAnswer = true;
console.log(fact.isTheAnswer); // undefined
```

Evaluation

- Questions
  - Warnings that are missed statically?
  - Additional warnings for statically checked rules?
  - Warnings vs. popularity?
- Setup
  - 200 web sites (top 50 Alexa + others)
  - Standard benchmarks: Octane, SunSpider
  - Comparison to JSHint

Summary: Static vs. Dynamic

- DLint complements static checkers:
  - 49 warnings per site that are missed statically
  - Additional warnings for static rules
  - Additional warnings from rules that cannot be checked statically

Problems Detected by DLint

- From craigslist.org:

```
if (document.body.style === "width:100\%") {
  ...
}
```

- Never succeeds because style is an object

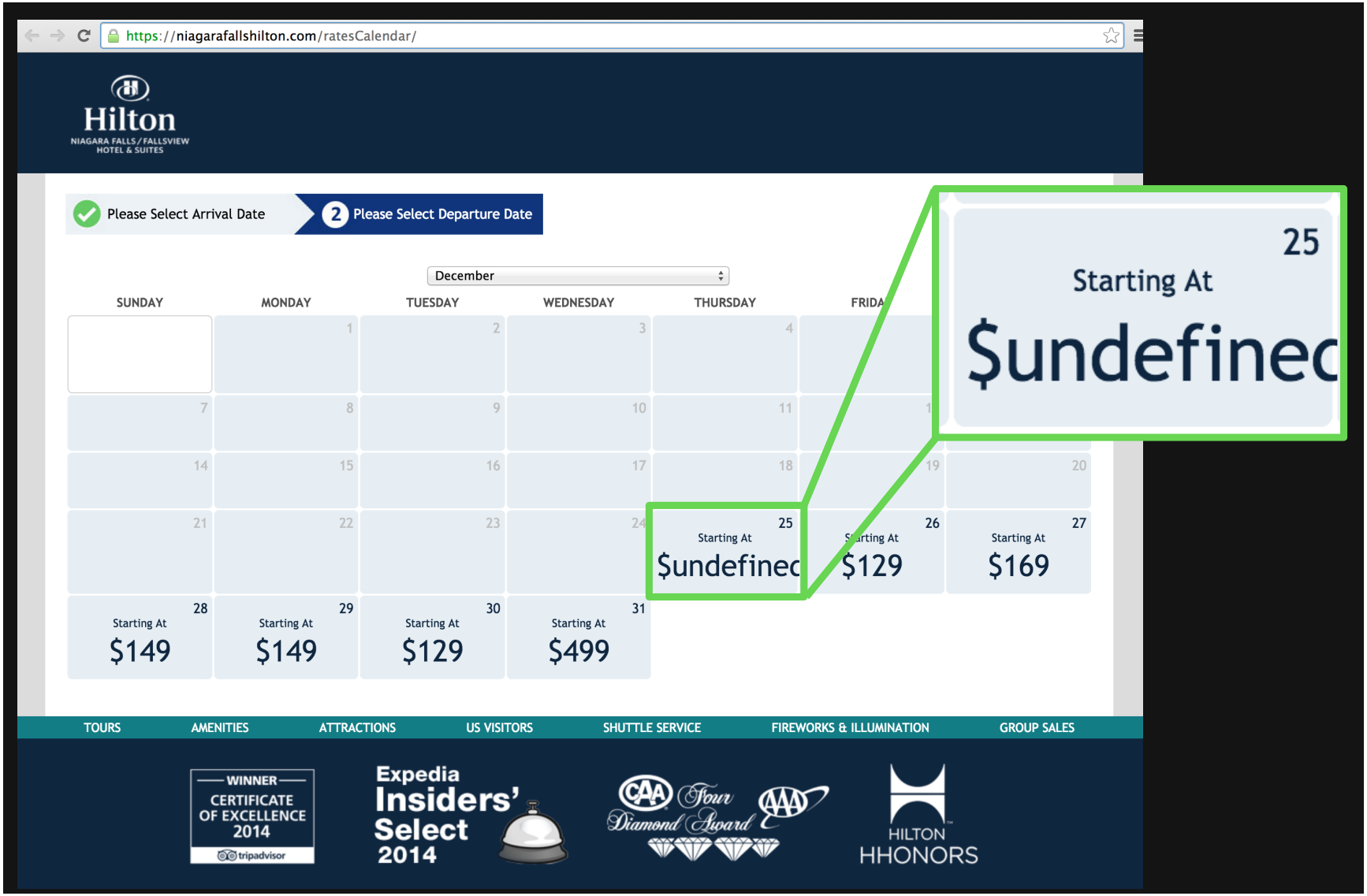
Problems Detected by DLint

- From Octane GB Emulator benchmark:

```
var decode64 = "";
if (dataLength > 3 && dataLength % 4 == 0) {
    while (index < dataLength) {
        decode64 += String.fromCharCode(...)
    }
    if (sixbits[3] >= 0x40) {
        decode64.length -= 1;
    }
}
```

- No effect because decode64 is a primitive

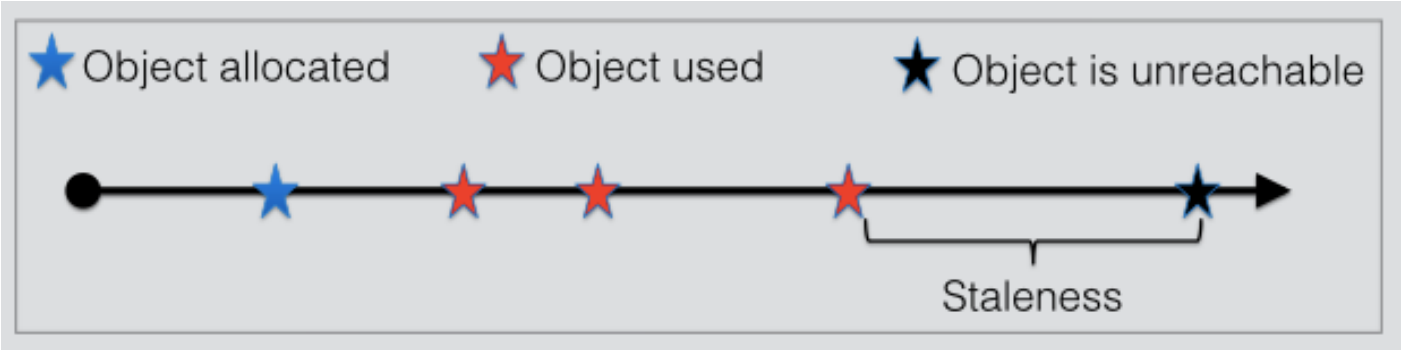
Problems Detected by DLint



Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra
(To appear at ESEC/FSE'15)

MEMINSIGHT: PLATFORM-INDEPENDENT MEMORY DEBUGGING FOR JAVASCRIPT (JALANGI POWERED)

Memory Issue: Drag or Staleness



- Staleness: long gap between last use and garbage collection
- A high number of stale objects indicates a potential problem
- Staleness can be avoided by ensuring objects become unreachable
- Typical causes of staleness: Closures, caches, and event listeners

Other Memory Issues

Churn

```
if (this.canRevert([ni, nj], color, board) &&
                !this.isContain([ni, nj], ret)){
       ret.push([parseInt(ni), parseInt(nj)]);
}
canRevert: function(place, color, _board){
      var i = parseInt(place[0]);
      var j = parseInt(place[1]);
     // no further usage of the place array
}
isContain: function(place, _array) {
   ... uses place[0] and place[1] ...
},
```

Bloat

```
return {
  type: type,
  value: id,
  lineNumber: lineNumber,
  lineStart: lineStart,
  range: [start, index]
};
```

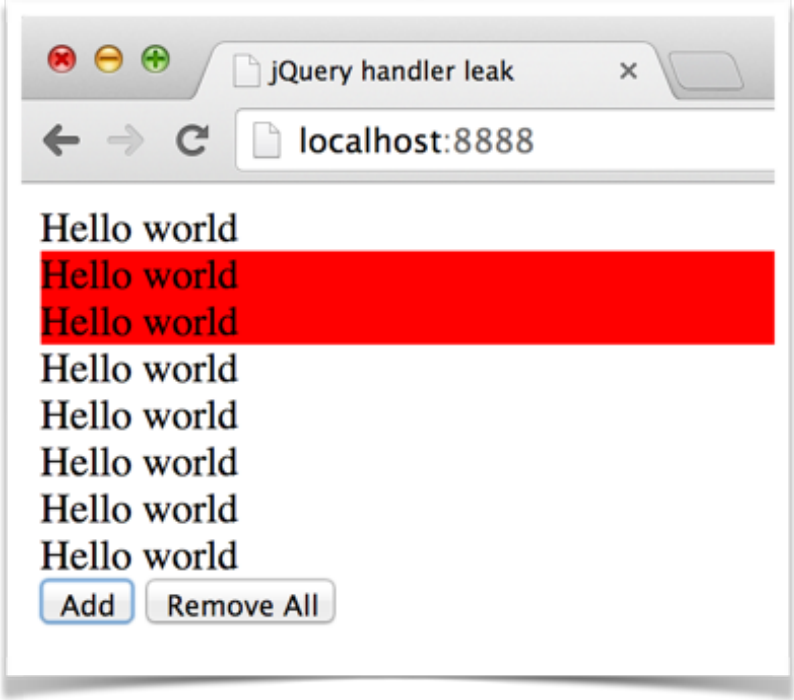
```
return {
  type: type,
  value: id,
  lineNumber: lineNumber,
  lineStart: lineStart,
  start: start,
  end: index
};
```

Existing Tools: Heap Snapshots

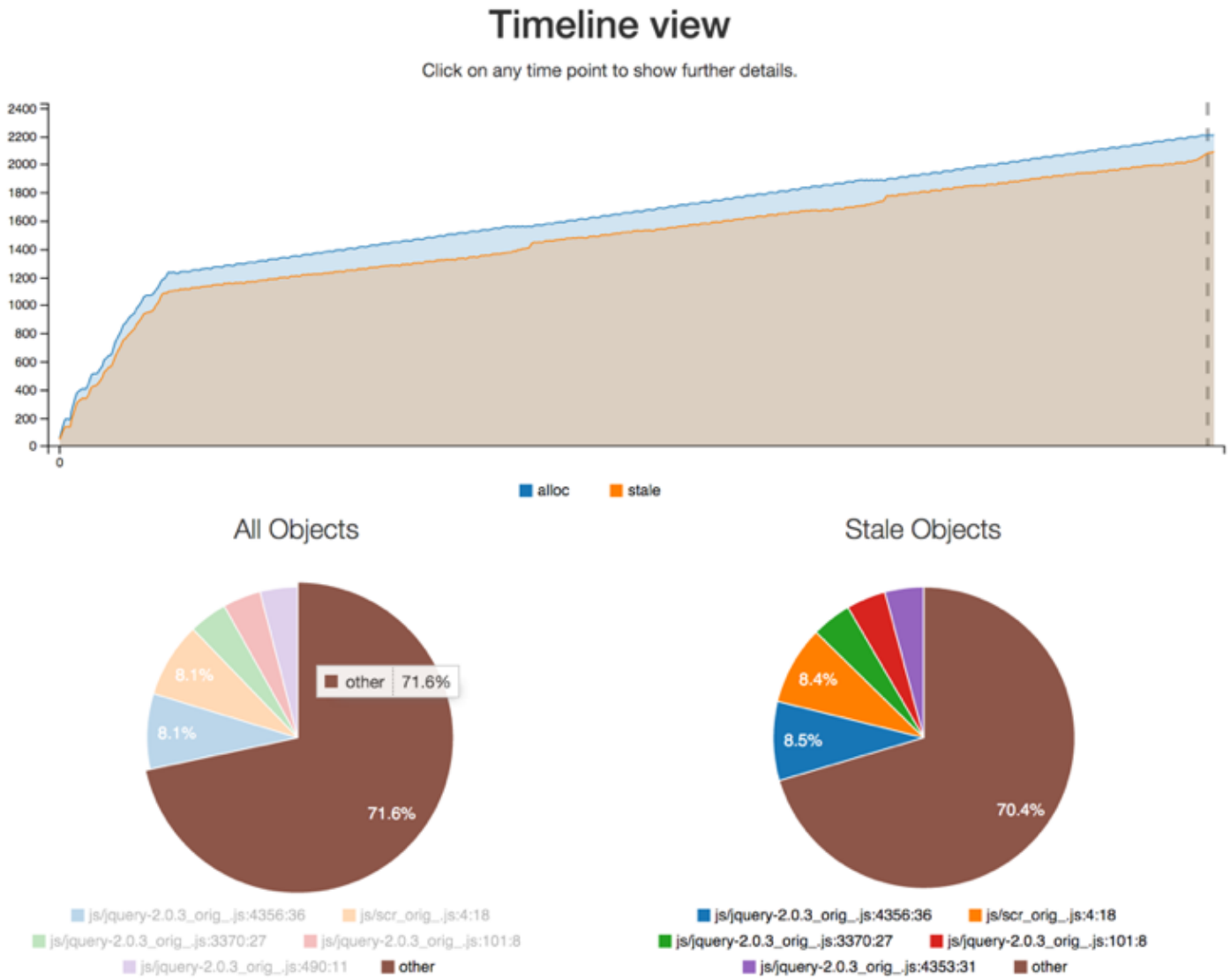
- Chrome developer tools offers a heap snapshot capture and diff functionality
- Potentially more efficient, but,
  - No information on staleness
  - Can miss excessive churn
  - In general, can't handle fine-grained time-varying properties

```
function f() {
    var newDiv = $('<div/>');
    newDiv.html("Hello world");
    newDiv.click(function () {
        newDiv.css("backgroundColor", "red");
    });
    newDiv.appendTo('#contents');
}
function g() {
    document.getElementById('contents').innerHTML = '';
}
```

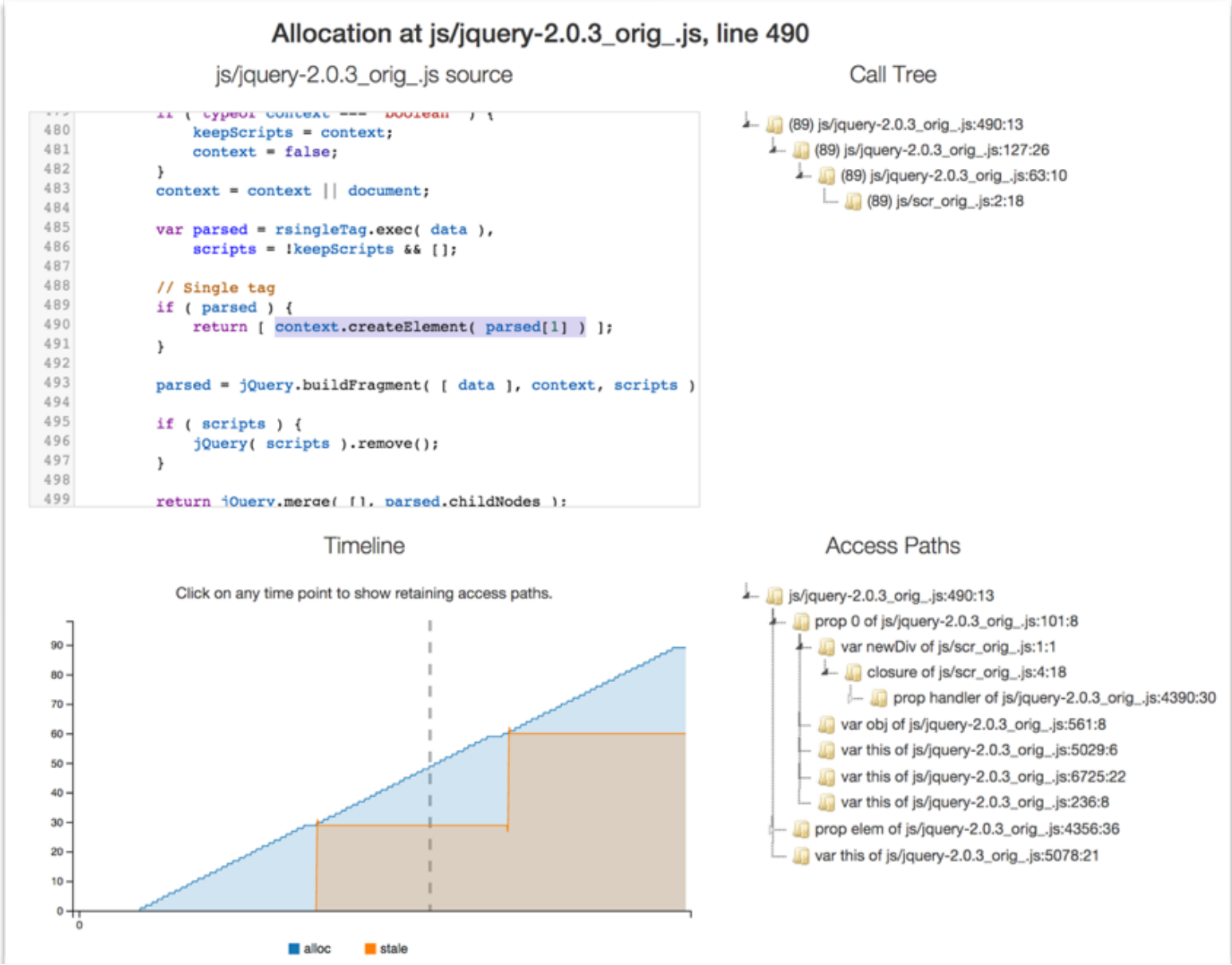
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>jQuery handler leak</title>
</head>
<body>
<div id="contents"></div>
<script src="js/jquery-2.0.3.js"></script>
<script src="js/scr.js"></script>
<button onclick="f()">Add</button>
<button onclick="g()">Remove All</button>
</body>
</html>
```



Memory Leak!


Timeline view
Click on any time point to show further details.
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
0
alloc
stale
All Objects
8.1%
8.1%
other 71.6%
71.6%
js/jquery-2.0.3_orig_.js:4356:36
js/scr_orig_.js:4:18
js/jquery-2.0.3_orig_.js:3370:27
js/jquery-2.0.3_orig_.js:101:8
js/jquery-2.0.3_orig_.js:490:11
other
Stale Objects
8.4%
8.5%
70.4%
js/jquery-2.0.3_orig_.js:4356:36
js/scr_orig_.js:4:18
js/jquery-2.0.3_orig_.js:3370:27
js/jquery-2.0.3_orig_.js:101:8
js/jquery-2.0.3_orig_.js:4353:31
other

Memory Leak - Details



jQuery issue

```
1  function f() {
2      var newDiv = $('<div/>');
3      newDiv.html("Hello world");
4      newDiv.click(function () {
5          newDiv.css("backgroundColor", "red");
6      });
7      newDiv.appendTo('#contents');
8  }
9  function g() {
10     //document.getElementById('contents').innerHTML = '';
11     $('#contents').empty();
12 }
13
```

Jalangi Trace Generation

```
1   var x = {};
2   var y = {};
3   function m(p,q)
4   {
5     p.f = q;
6   };
7   m(x,y);
8   x = null;
```

```
DECLARE x,y,m;
ALLOCOBJ 2 at 1;
WRITE x,2 at 1;
ALLOCOBJ 3 at 2;
WRITE y,3 at 2;
ALLOCFUN 4 at 3;
WRITE m,4 at 3;
CALL 4 at 7;
DECLARE p = 2,
        q = 3;
PUTFIELD 2,"f",3
         at 5;
```

```
LASTUSE 2 at 5;
RETURN at 7;
LASTUSE 4 at 7;
WRITE x,0 at 8;
UNREACHABLE
        2 at 8;
UNREACHABLE
        3 at end;
UNREACHABLE
        4 at end;
```

Object Lifetimes

- From trace, model runtime heap
  - Including call stack and closures
- Reference counting to compute unreachability time
  - Handle cycles with Merlin algorithm [Hertz et al. ASPLOS'06]
- Insert unreachability times in the enhanced trace

Handling DOM

```
1  var elem = document.createElement("div");
2  elem.innerHTML = "<p><h1>Hello World!</h1></p>";
3  document.getElementById("x").appendChild(elem);
```

- Handle various DOM modification events and generate putfield/getfield events
  - uses HTML5 “mutation observer”
  - also handles appendChild
  - Special handling of innerHTML provides line numbers

Success Stories

- Leaks
  - Fixed in one Tizen app (patch accepted)
  - Confirmed existing patch fixes leak in dataTables
  - Leaks found by internal users in other apps
- Churn
  - Fixed in one Tizen app for 10% speedup (patch accepted)
  - 10X speedup for escodegen (patch accepted)
- Bloat
  - Found object inlining opportunity in old esprima version

Final slide

- Jalangi 2 is a powerful dynamic analysis tool
  - much powerful than traditional dynamic analyses tools
  - can change semantics/values
  - can repeatedly execute arbitrary paths in a function
  - portable and browser agnostic
  - simple API
  - under active development and actively maintained
  - 10X to 100X slowdown, but barely noticeable on event-driven apps
- Give it a try today!

  https://github.com/Samsung/jalangi2

Extra slides

Rule #1: Use Consistent Object Layout

```
function access_field(o) {
    return o.a;
}

access_field({a:1, b:2})
access_field({b:2, a:1})
access_field({a:1, b:2})
```

Rule #1: Check Consistent Object Layout

```
function access_field(o) {
 loc1: return o.a;
}

access_field({a:1, b:2})
access_field({b:2, a:1})
access_field({a:1, b:2})
```

Meta info. per location

Locations	Layouts	Cache Misses
loc1	[a,b] [b,a]	2
…	…	…
…	…	…

- Override *analysis.getField* and *analysis.putField* to create the table.
- Rank locations by number of cache misses

Rule #2: Check if Arrays are Purely Numeric

For each assignment location count number of times non-numeric value has been assigned to a numeric array

```
var a = new Array();
a[0] = 77;
a[1] = 88;
a[2] = {};
```

Rule #2: Check if Arrays are Purely Numeric

For each assignment location count number of times non-numeric value has been assigned to a numeric array

```
        var a = new Array();
loc1:   a[0] = 77;
loc2:   a[1] = 88;
loc3:   a[2] = {};
```

Meta info. per location

Locations	# of bad assignments
loc1	0
loc2	0
loc3	1

- Override *analysis.getField* and *analysis.putField* to create the table
- Rank locations by number of bad assignments

Rule #3: Check Access of Uninitialized Elements

Do not load uninitialized (or deleted) array elements

```
var array = [0];
var c;
for (var i=0;i<100000000;i++){
    c = array[1];
}
```

Accessing uninitialized element takes 4.907s

Rule #3: Check Access of Uninitialized Elements

Do not load uninitialized (or deleted) array elements

```
var array = [0, 1];
var c;
for (var i=0;i<100000000;i++){
    c = array[1];
}
```

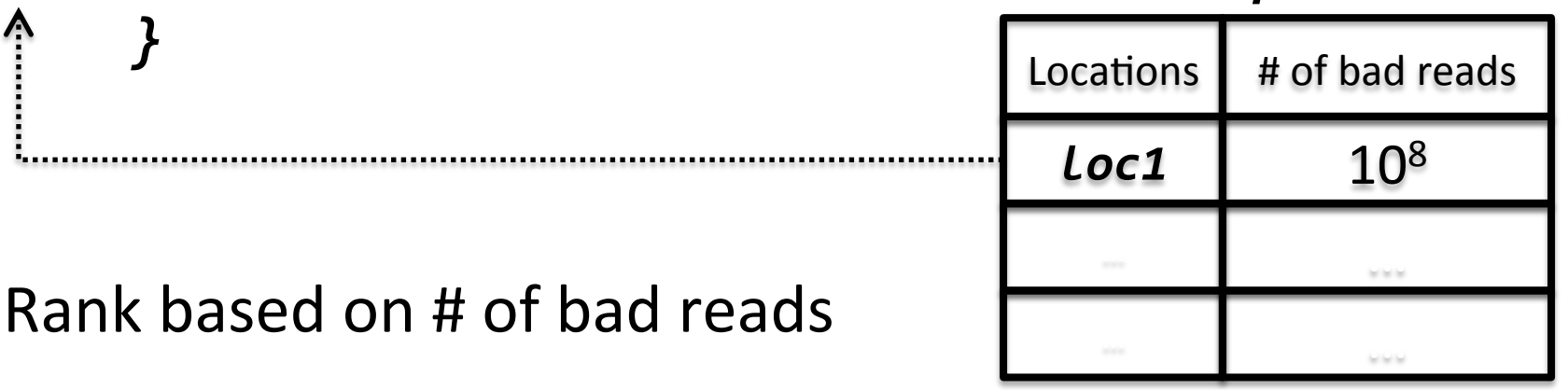
Accessing initialized element takes 0.121s

40X speedup!

Rule #3: Check Access of Uninitialized Elements

Do not load uninitialized (or deleted) array elements

```
var array = [0, 1];
var c;
for (var i=0;i<100000000;i++){
loc1:    c = array[1];
}
```



Rank based on # of bad reads

splay.js in Google Octane

1st

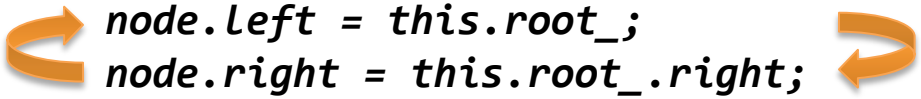
```
"Report of polymorphic statements:"
[Line: 235, Column: 13] <- No. layouts: 3
"count: 1         -> layout:"key|value|"
"count: 114088    -> layout:"key|value|left|right|"
"count: 110836    -> layout:"key|value|right|left|"
```

splay.js in Google Octane

1st

```
"Report of polymorphic statements:"
[Line: 235, Column: 13] <- No. layouts: 3
"count: 1          -> layout:"key|value|"
"count: 114088     -> layout:"key|value|left|right|"
"count: 110836     -> layout:"key|value|right|left|"
```

```
SplayTree.prototype.insert = function(key, value) {
  ...
  var node = new SplayTree.Node(key, value);
  if (key > this.root_.key) {
    node.left = this.root_;
    node.right = this.root_.right;
    ...
  } else {
    node.right = this.root_;
    node.left = this.root_.left;
    ...
  }
  this.root_ = node;
};
```



Performance boost:

15%

6.7%

date-format-tofte.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (date-format-tofte.js:274:12)]    <- No. usages: 1000
```

```
var ia = input.split("");
var ij = 0;
while (ia[ij]) {
    ...
    ij++;
}
```

date-format-tofte.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (date-format-tofte.js:274:12)]    <- No. usages: 1000
```

```
var ia = input.split("");
var ij = 0;
while (ia[ij]) {
    ...
    ij++;
}
```

refactor

```
var ia = input.split("");
var ij = 0;
var len = ia.length;
while (ij < len) {
    ...
    ij++;
}
```

date-format-tofte.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (date-format-tofte.js:274:12)]    <- No. usages: 1000
```

```
var ia = input.split("");
var ij = 0;
while (ia[ij]) {
    ...
    ij++;
}
```

refactor

```
var ia = input.split("");
var ij = 0;
var len = ia.length;
while (ij < len) {
    ...
    ij++;
}
```

Performance boost:

2.05% 

5.33% 

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var bin = Array();
  ...


  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var bin = Array();
  ...

  for(var i = 0; i < str.length * chrsz; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on undefined value

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Pre-allocate array space to eliminate loading undeclared array elements

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on numbers

Pre-allocate array space to eliminate loading undeclared array elements

crypto-sha1.js in SunSpider

1st

```
Report of loading undeclared or deleted array elements:
[location: (crypto-md5.js:206:5)]     <- No. usages: 3956
[location: (crypto-md5.js:43:3)]      <- No. usages: 1
```

```
function str2binl(str)
{
  var len = str.length * chrsz, len1 = len>>5;
  var bin = []; bin.length = len1;
  for(var i=0;i<len;i+=chrsz) {bin[i>>5] = 0;}
  ...
  for(var i = 0; i < len; i += chrsz)
    bin[i>>5] |= ...
  return bin;
}
```

Operations on numbers

Performance boost:

1.5% 

31% 

Pre-allocate array space to eliminate loading undeclared array elements

Benchmark: SunSpider (SS) & Octane (Oct)	CPR FF|CH (function level)	JITProf Rank (statement level)	Ch. LOC	Avg. improvement (stat. significant)	
				Firefox	Chrome
SS-Crypto-SHA1	2 |5+	1 in UAE, PO, BOU	6	3.3±0.9%	26.3±0.4%
SS-Str-Tagcloud	5 | -	1 in IOL	15	-	11.7±0.7%
SS-Crypto-MD5	3 |5+	1 in UAE, PO, BOU	6	-	24.6±0.1%
SS-Format-Tofte	2 | 1	1 in UAE	2	-	3.4±0.2%
SS-3d-Cube	5+| 5	1 in NCA	1	-	1.1±0.1%
SS-Format-Xparb	4 | 1	1 in PO	2	19.7±0.5%	22.4±0.3%
SS-3d-Raytrace	5 | 5	1 in NNA	4	-	2.6±0.2%
SS-3d-Morph	1 | 1	1 in GA	1	-	1.5±0.3%
SS-Fannkuch	1 | 1	1 in GA	3	8.3±0.9%	5.4±2.3%
Oct-Splay	5 |5+	1 in IOL	2	3.5±0.9%	15.1±0.3%
Oct-SplayLatency	5 |5+	1 in IOL	2	-	3.8±0.6%
Oct-DeltaBlue	5+|5+	2 in IOL	6	1.4±0.2%	-
Oct-RayTrace	5+| 1	1 in IOL	18	-	12.9±1.9%
Oct-Box2D	5+|5+	2 in IOL	1	-	7.5±0.6%
Oct-Crypto	5+|5+	1 in GA	1	13.8±4.9%	3.3±0.4%

CPR means CPU Profiler Rank. FF means Firebug Profiler, CH means Google Chrome's profiler. Ch. LOC is the number of changed LOC. Short names (e.g., IOL) in the third column refers to the profilers defined in Table 2. - means no ranking or no statistically significant difference. Confidence intervals of improvements of Firefox and Chrome in the last two columns are at 95% confidence level [13].

Example 2

- *www.repl.it*, included from jQuery:

```
globalEval: function(code) {
        var script, indirect = eval;
        code = jQuery.trim( code );
        if (code) {
                if (code.indexOf("use strict") === 1) { ...
                } else {
                        indirect(code); // indirect call of eval
                } }
}
```

- Rule: Avoid eval(). (Nothing wrong with this code.)

Code Quality Rules

- Informal description of a pattern of code or execution behavior that should be avoided or that should be used in a particular way
- Increase correctness, maintainability, code readability, performance, or security

Inheritance-Related

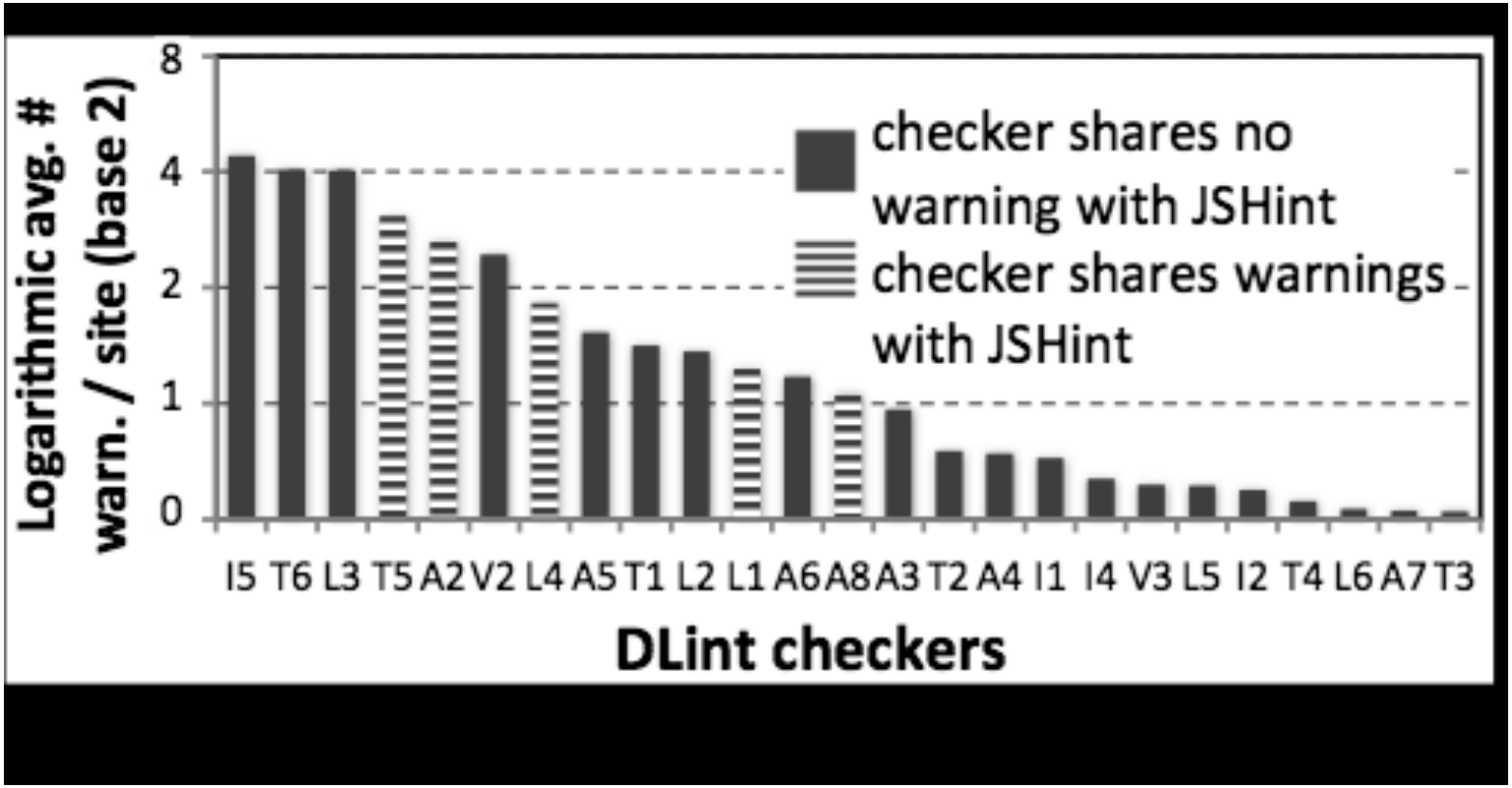
- Avoid shadowing a prototype property with an object property.

```
function C() {};
C.prototype.x = 3;
var obj = new C();
obj.x = 5;
console.log(obj.x);
```

DLint vs. JSHint

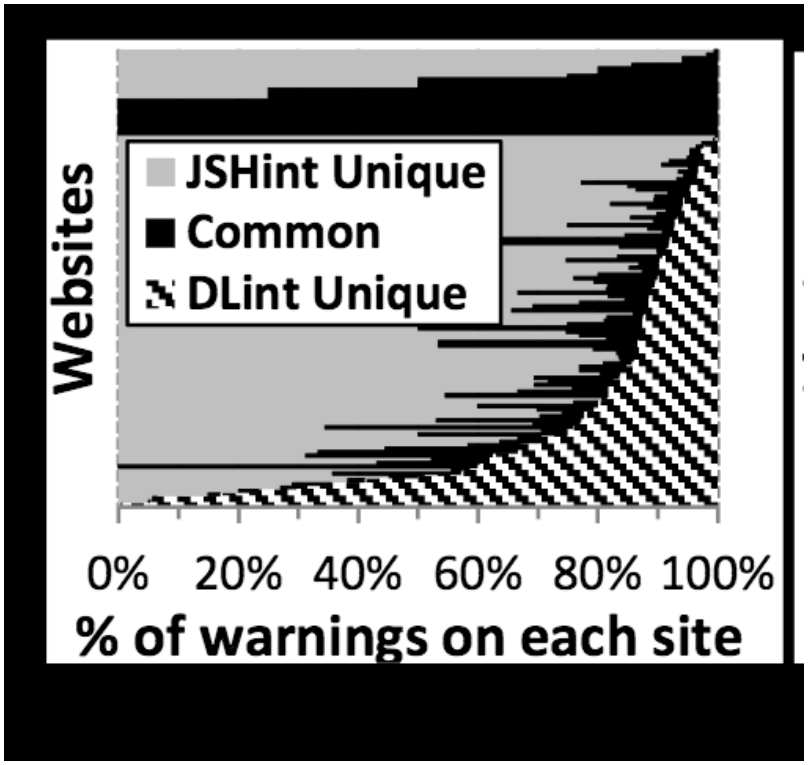
- DLint
  - 28 rules
  - 9018 warnings
- JSHint
  - 150 rules
  - 580k warnings
  - Focus for comparison: 9 rules

DLint Warnings



- 53 warnings per page
- 49 are missed by JSHint

DLint vs. JSHint Warnings



- Some sites: One approach finds all
- Most sites: Better together