Jalangi: A Dynamic Analysis Framework for JavaScript

Manu Sridharan
Uber

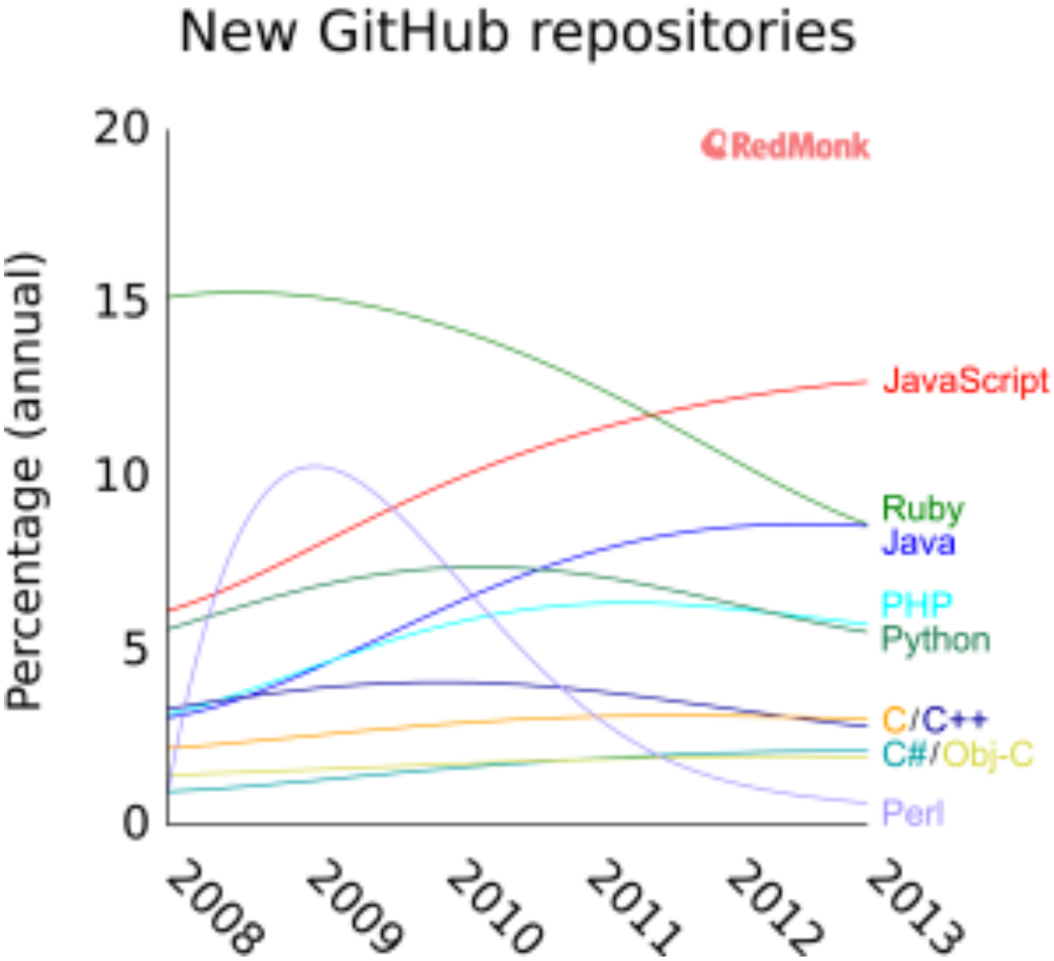
Koushik Sen, Liang Gong
University of California, Berkeley

With contributions from:

Christoffer Adamsen, Esben Andreasen, Tasneem Brutch, Satish Chandra, Colin S. Gordon, Simon Gibbs, Simon Jenson, Swaroop Kalasapur, Rezwana Karim, Magnus Madsen, Michael Pradel, Frank Tip

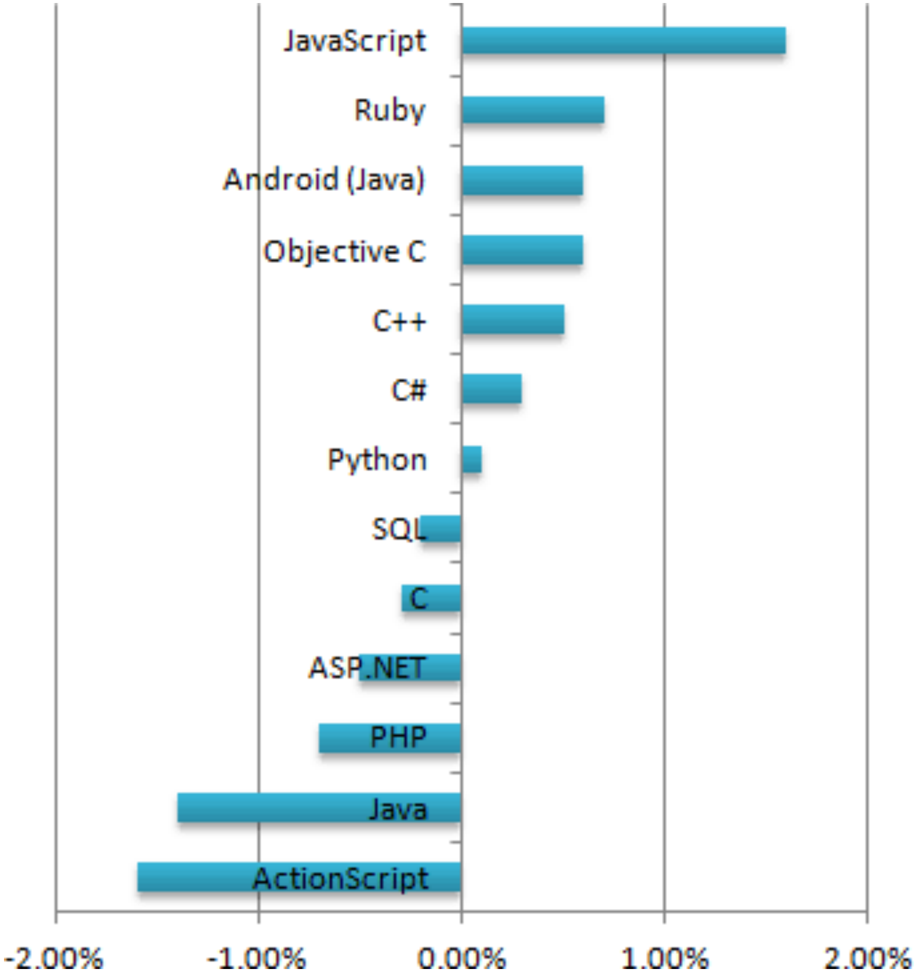
Why JavaScript?

- The RedMonk Programming Language Rankings (Popularity): January 2015 and 2016
  - Based on projects hosted at GitHub and questions posted at StackOverflow



Why JavaScript?

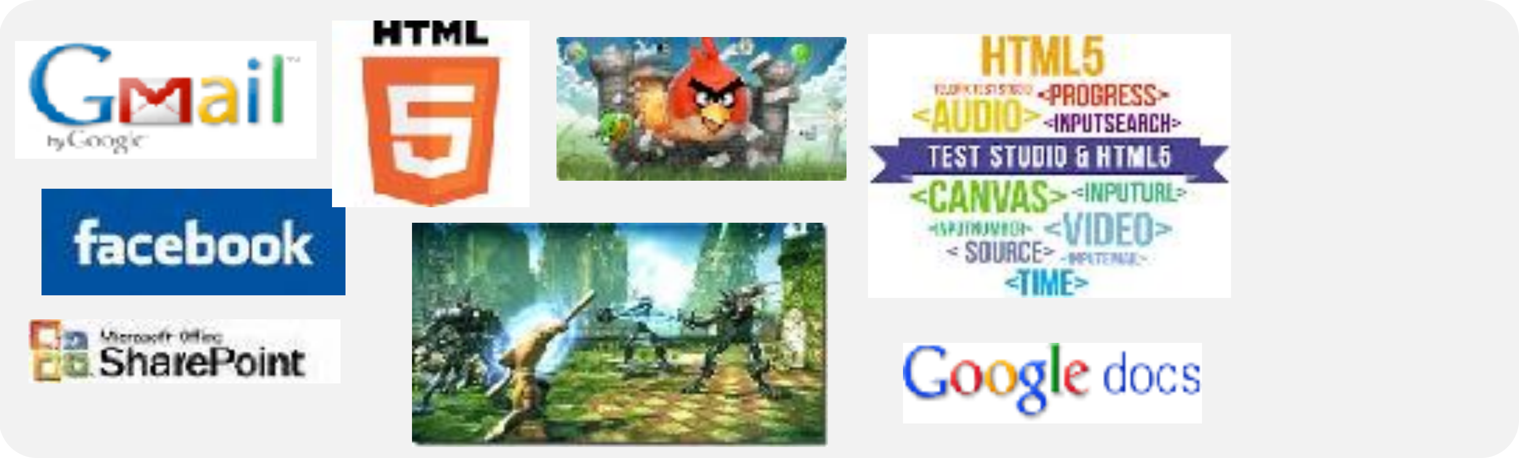
Growth in popularity (based on jobs available) from 2012 – 2013



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)

Why Tools for JavaScript?

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

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

Why Tools for JavaScript?

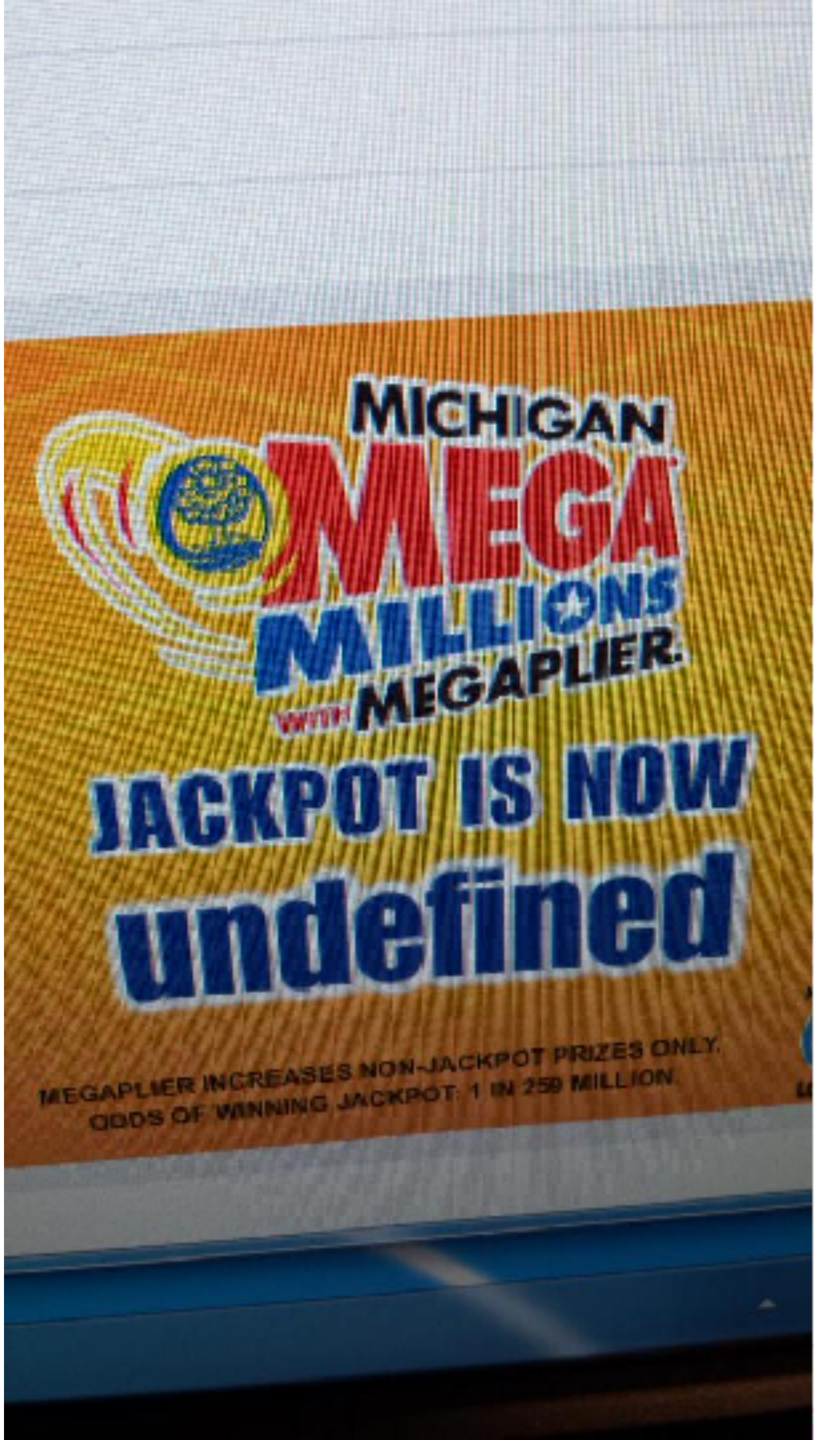
```
var x = “1”;
++x;
console.log(x);

// 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

- Loosely-typed
  - forgiving: implicit type conversion
  - tries hard to execute without throwing exception
    - Like HTML


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

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

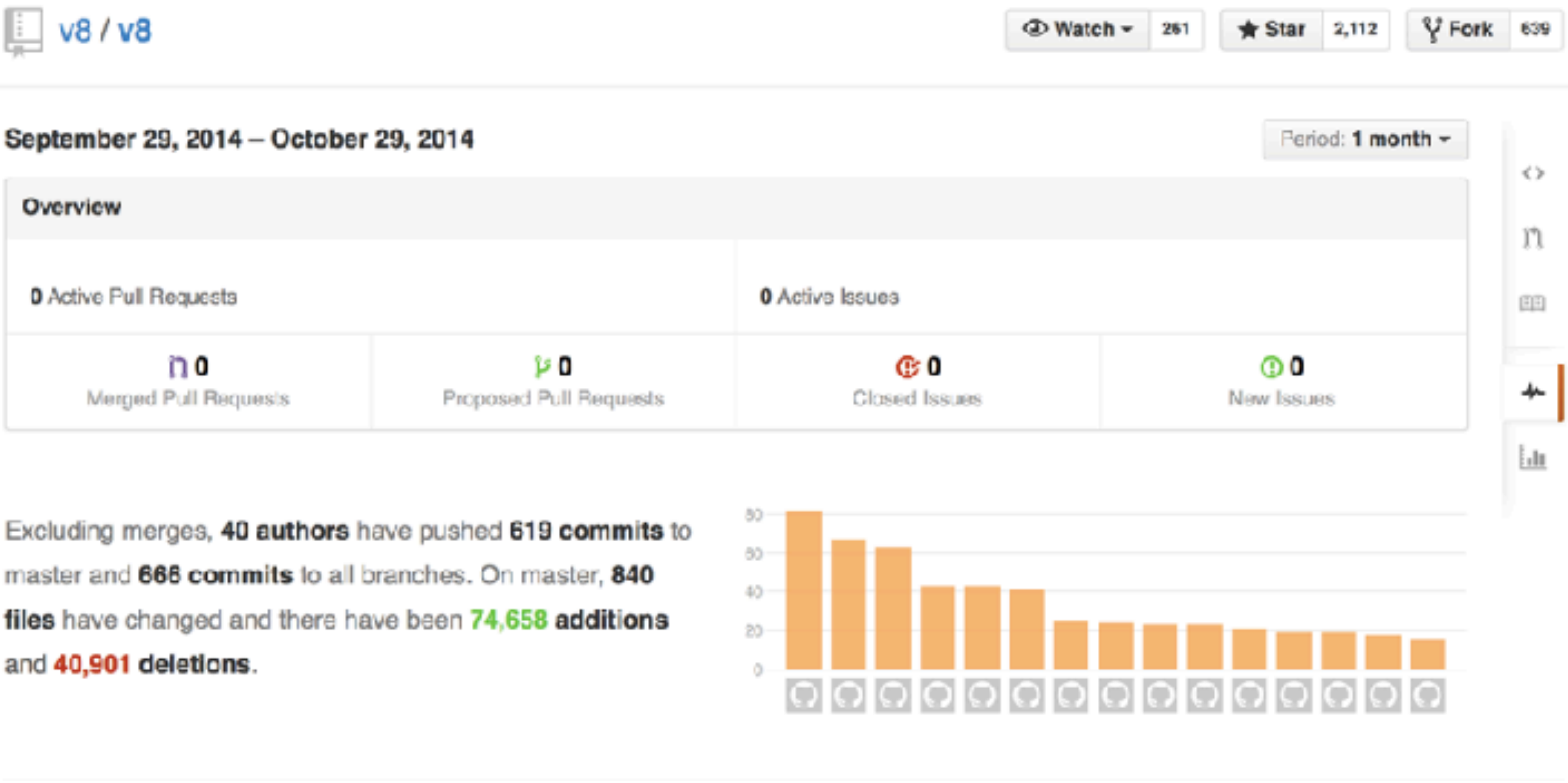
- Jalangi: A selective record-replay and dynamic analysis framework for JavaScript. Koushik Sen, Swaroop Kalasapur, Tasneem Brutch, and Simon Gibbs. In ESEC/FSE, 2013.

Jalangi: Goals and Requirements

- Framework for Dynamic and hybrid Static/Dynamic analysis
  - supports symbolic execution, bug finding, 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



v8 / v8

Watch 261 | Star 2,112 | Fork 639

September 29, 2014 – October 29, 2014

Period: 1 month

Overview

0 Active Pull Requests

0 Active Issues

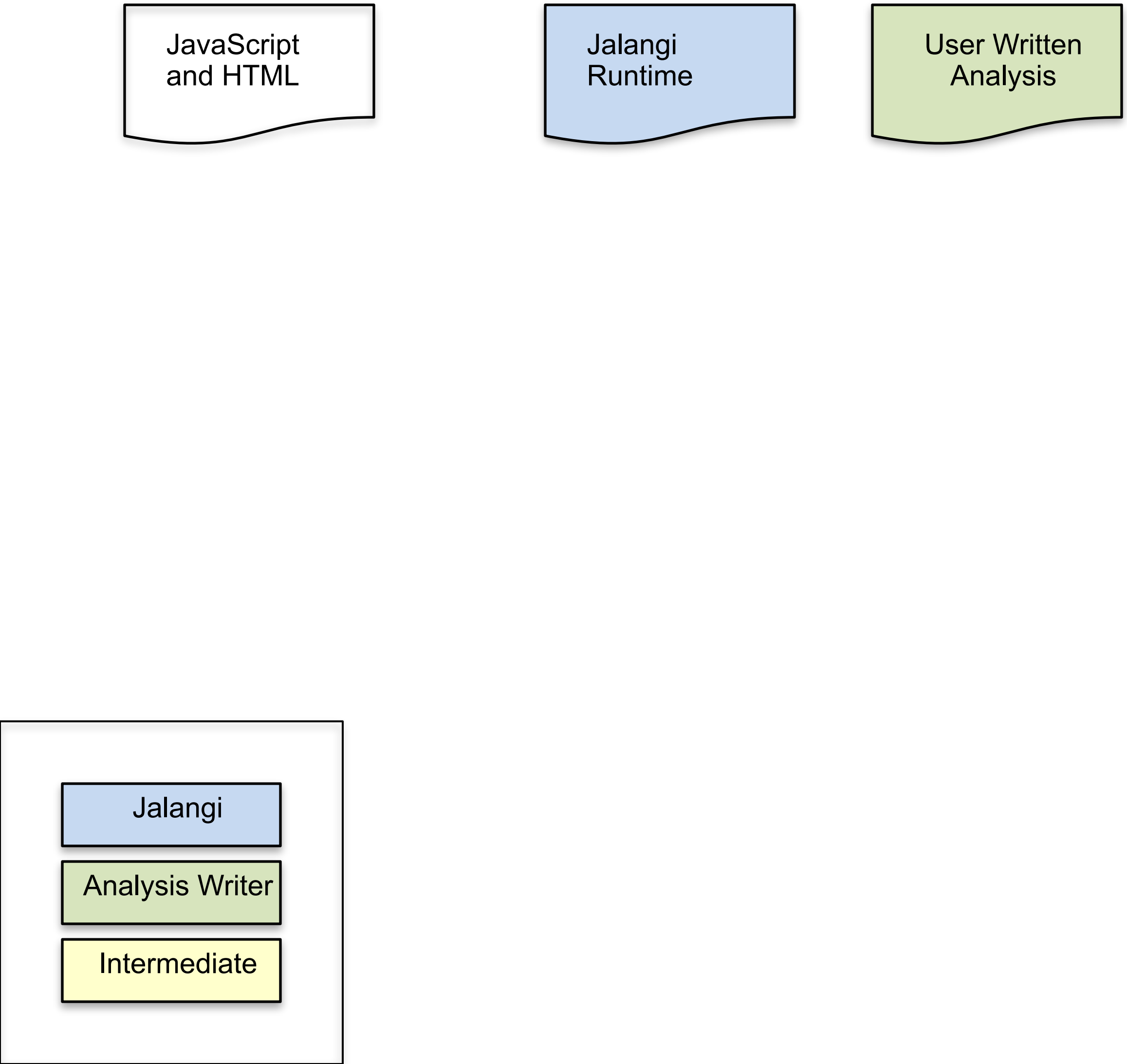
0	0	0	0
Merged Pull Requests	Proposed Pull Requests	Closed Issues	New Issues

Excluding merges, **40 authors** have pushed **619 commits** to master and **666 commits** to all branches. On master, **840 files** have changed and there have been **74,658 additions** and **40,901 deletions**.

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
    - optional shadow memory
  - active development

How Jalangi Works?



JavaScript and HTML

Jalangi Runtime

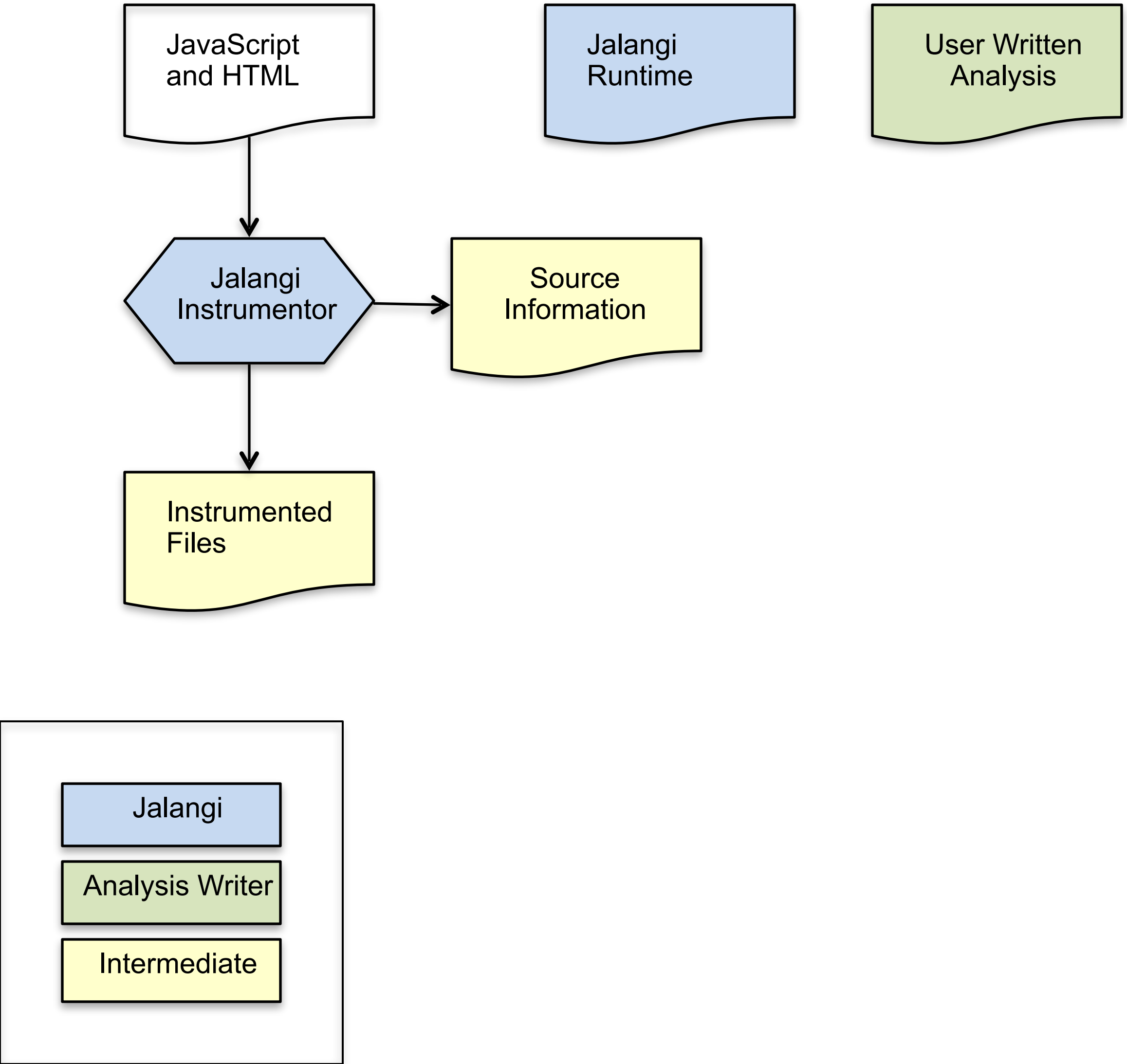
User Written Analysis

Jalangi

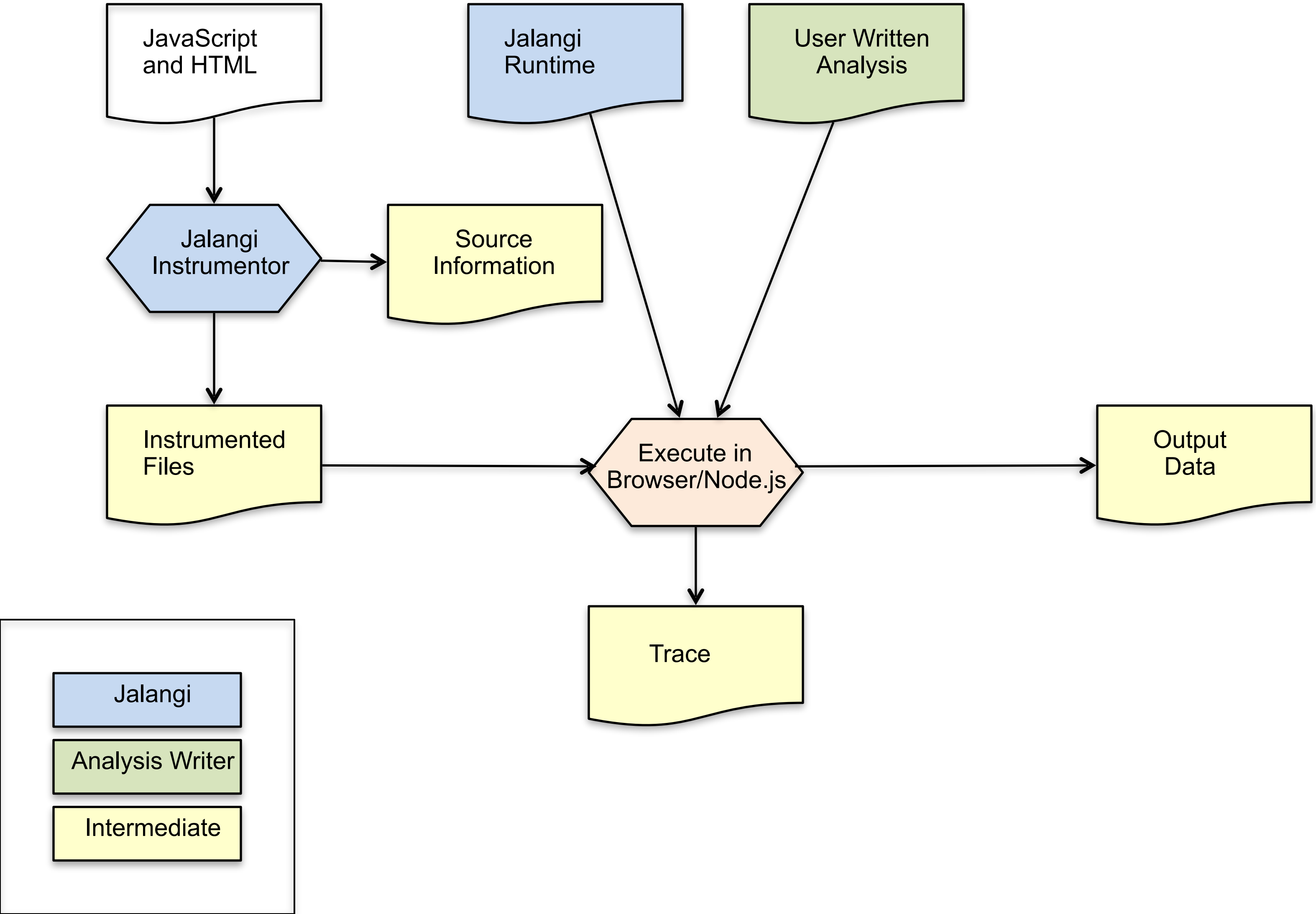
Analysis Writer

Intermediate

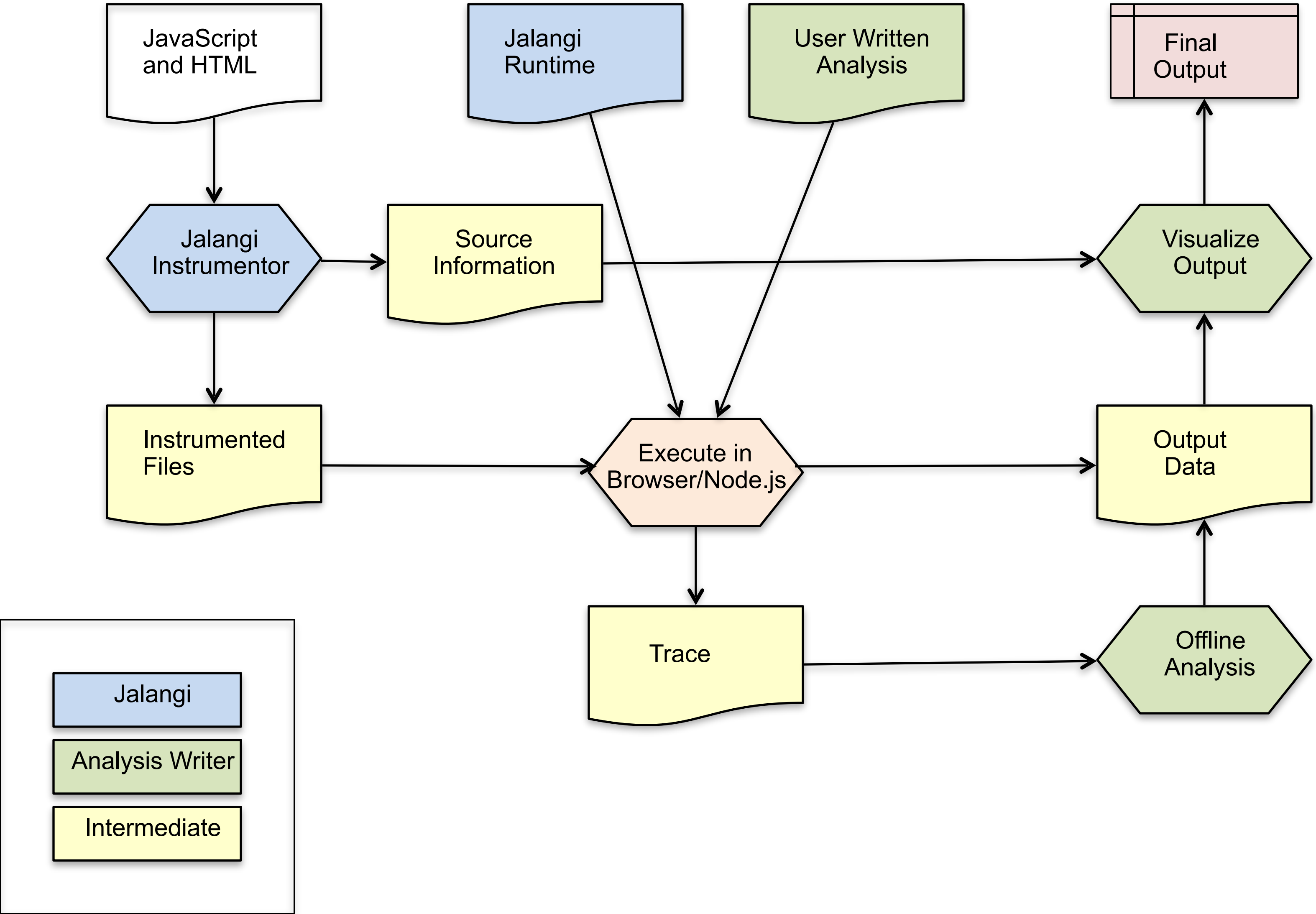
How Jalangi Works?



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;
}
```

Download and Install Jalangi 2

Download:

```
git clone https://github.com/Samsung/jalangi2.git
cd jalangi2
```

Install:

```
npm install
```

Test:

```
python scripts/test.traceall.py
python scripts/test.analysis.py
python scripts/test.dlint.py
```

Jalangi Callbacks

Documentation: jalangi2/docs/MyAnalysis.html

```
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**);**

TraceAll.js analysis: prints all callbacks

For Node.js

-
  ```
  node src/js/commands/jalangi.js --inlineIID --inlineSource --analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js --analysis src/js/sample_analyses/pldi16/TraceAll.js tests/pldi16/TraceAllTest.js
  ```

For browser:

-
  ```
  node src/js/commands/esnstrument_cli.js --inlineIID --inlineSource --analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js --analysis src/js/sample_analyses/pldi16/TraceAll.js --out /tmp/pldi16/TraceAllTest.html  tests/pldi16/TraceAllTest.html
  ```

-
  ```
  node src/js/commands/esnstrument_cli.js --inlineIID --inlineSource --analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js --analysis src/js/sample_analyses/pldi16/TraceAll.js --out /tmp/pldi16/TraceAllTest.js  tests/pldi16/TraceAllTest.js
  ```

-
  ```
  open file:///tmp/pldi16/TraceAllTest.html
  ```

Sample Analyses

Examples: src/js/sample_analyses/pldi16

Tests: tests/pldi16

Sample analysis:

check if undefined is concatenated with a string

See: src/js/sample_analyses/pldi16/CheckUndefinedConcatenatedToString.js

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

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

Source Locations

- Instrumentation associates an iid with every expression
- At runtime, each loaded script is given a unique script ID (sid)
- sid of current script stored in J$.sid
- J$.getGlobalIID(iid) gets a globally unique id
- J$.iidToLocation(J$.sid, iid) gets source location
    - filename:start_line:start_col:end_line:end_col
- Tracks locations of enclosing evals

Sample analysis: count branches

See: src/js/sample_analyses/pldi16/BranchCoverage.js

```
var trueBranches = {};
var falseBranches = {};
// initialize ....

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

```
this.endExecution = function () {
    print(trueBranches, “True”);
    print(falseBranches, “False”);
}
```

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

Sample analysis:
count number of objects allocated at each site

See: src/js/sample_analyses/pldi16/CountObjectsPerAllocationSite.js

```
var allocCount= {};

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);
}
```

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

Shadow Objects (SMemory.js)

- Associates a shadow object with each JavaScript object (excludes primitive values including strings and null)
- Associates a shadow object with each activation frame
- Shadow object can store meta-information
- A shadow object contains an unique id
  - can be used as logical address of an object/frame

```
--analysis src/js/sample_analyses/ChainedAnalyses.js --analysis src/js/runtime/SMemory.js
```

SMemory.js API

Documentation: jalangi2/docs/SMemory.html

- **getShadowObject(obj, prop, isGetField)**

This method should be called on a base object and a property name to retrieve the shadow object associated with the object that actually owns the property

- **getShadowObjectOfObject(val)**

This method returns the shadow object associated with the argument. If the argument cannot be associated with a shadow object, the function returns undefined.

- **getShadowFrame(name)**

This method returns the shadow object associated with the activation frame that contains the variable "name". To get the current activation frame's shadow object, call getShadowFrame('this')

- **getIDFromShadowObjectOrFrame(obj)**

Given a shadow object or frame, it returns the unique id of the shadow object or frame. It returns undefined, if obj is undefined, null, or not a valid shadow object.

- **getActualObjectOrFunctionFromShadowObjectOrFrame**(obj)

Given a shadow object/frame, it returns the actual object/the function whose invocation created the frame.

Associate Allocation Site

See: src/js/sample_analyses/pldi16/LogLoadStoreAlloc.js

```
this.literal = function (iid, val, hasGetterSetter) {
    if (typeof val === "object" && val !== null) {
        var sobj = sandbox.smemory.getShadowObjectOfObject(val);
        sobj.allocSite = J$.iidToLocation(J$.sid, iid);
    }
};
```

```
this.getFieldPre = function (iid, base, offset, isComputed, isOpAssign, isMethodCall) {
    var sobj = sandbox.smemory.getShadowObject(base, offset, true).owner;
    var ret = "Load '"+offset+ "' of object allocated at" + sobj.allocSite;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
};
```

Log All Loads and Stores

See: src/js/sample_analyses/pldi16/LogLoadStoreAlloc.js

```
this.getFieldPre = function (iid, base, offset, isComputed, isOpAssign, isMethodCall) {
    var sobj = sandbox.smemory.getShadowObject(base, offset, true).owner;
    var actualObjectId = sandbox.smemory.getIDFromShadowObjectOrFrame(sobj);
    var ret = "Load of object(id=" + actualObjectId + ")." + offset;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
};

this.write = function (iid, name, val, lhs, isGlobal, isScriptLocal) {
    var sobj = sandbox.smemory.getShadowFrame(name);
    var frameId = sandbox.smemory.getIDFromShadowObjectOrFrame(sobj);
    var ret = "Store of frame(id=" + frameId + ")." + name;
    ret += " at " + J$.iidToLocation(J$.sid, iid);
    log(ret);
    return {result: val};
};
```

Sample analysis (modify semantics):
interpret ‘*’ as ‘+’

See: src/js/sample_analyses/pldi16/ChangeSematicsOfMult.js

```
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};
};
```

Sample analysis (modify semantics):

skip execution of an evil function

See: src/js/sample_analyses/pldi16/SkipFunction.js

```
this.invokeFunPre = function (iid, f, base, args) {
    if (typeof evilFunction === "function" && f === evilFunction) {
            return {f: f, base: base, args: args, skip: true};
};
```

Sample analysis (modify semantics):

loop a function body

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

```
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

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

Prints 10

```
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

See: src/js/sample_analyses/pldi16/BackTrackLoop.js

```
this.functionExit = function (iid, rv, ex) {
   return {returnVal: rv, wrappedExceptionVal: ex, isBacktrack: rv?true:false};
};
```

---------------------------------- Program ----------------------------------

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

Prints 10 to 0

Sample analysis (modify semantics):

MultiSE: Multi-Path Symbolic Execution using Value Summaries (ESEC/FSE 2015)

- 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/functions
- 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

- **"Feedback-Directed Instrumentation for Deployed JavaScript Applications,"**
  - Magnus Madsen and Frank Tip and Esben Andreasen and Koushik Sen and Anders Moller (ICSE'16)
- **"Trace Typing: An Approach for Evaluating Retrofitted Type Systems,"**
  - Esben Andreasen and Colin S. Gordon and Satish Chandra and Manu Sridharan and Frank Tip and Koushik Sen (ECOOP'16)
- **"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 (ESEC/FSE'15)
- **"MemInsight: Platform-Independent Memory Debugging for JavaScript,"**
  - Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (ESEC/FSE'15)
- **"DLint: Dynamically Checking Bad Coding Practices in JavaScript,"**
  - Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (ISSTA'15)
- **"MultiSE: Multi-Path Symbolic Execution using Value Summaries,"**
  - Koushik Sen and George Necula and Liang Gong and Wontae Choi, (ESEC/FSE'15)
- **"The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"**
  - Michael Pradel and Koushik Sen (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

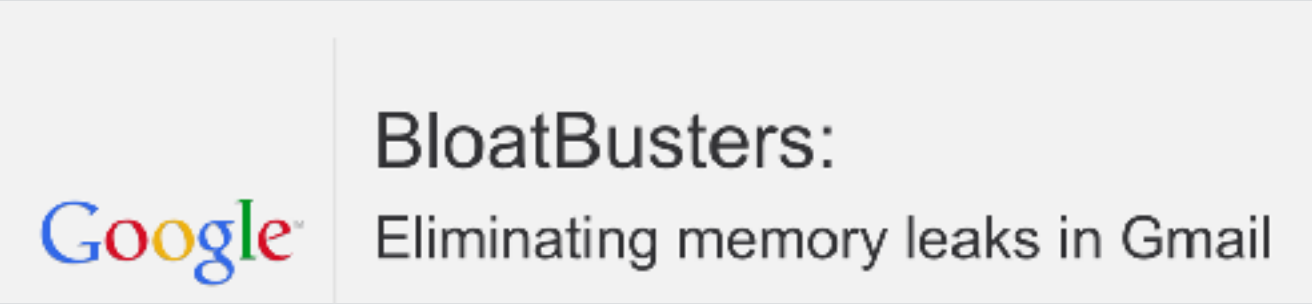
- **"Feedback-Directed Instrumentation for Deployed JavaScript Applications,"**
  - Magnus Madsen and Frank Tip and Esben Andreasen and Koushik Sen and Anders (ICSE'16)
- **"Trace Typing: An Approach for Evaluating Retrofitted Type Systems,"**
  - Esben Andreasen and Colin S. Gordon and Satish Chandra and Manu Sridharan and Frank Tip and Koushik Sen (ECOOP'16)
- **"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 (ESEC/FSE'15)
- **"MemInsight: Platform-Independent Memory Debugging for JavaScript,"**
  - Simon Jensen and Manu Sridharan and Koushik Sen and Satish Chandra (ESEC/FSE'15)
- **"DLint: Dynamically Checking Bad Coding Practices in JavaScript,"**
  - Liang Gong and Michael Pradel and Manu Sridharan and Koushik Sen (ISSTA'15)
- **"MultiSE: Multi-Path Symbolic Execution using Value Summaries,"**
  - Koushik Sen and George Necula and Liang Gong and Wontae Choi, (ESEC/FSE'15)
- **"The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript,"**
  - Michael Pradel and Koushik Sen (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)

MemInsight

Platform-Independent Memory Debugging for JavaScript

http://github.com/Samsung/meminsight

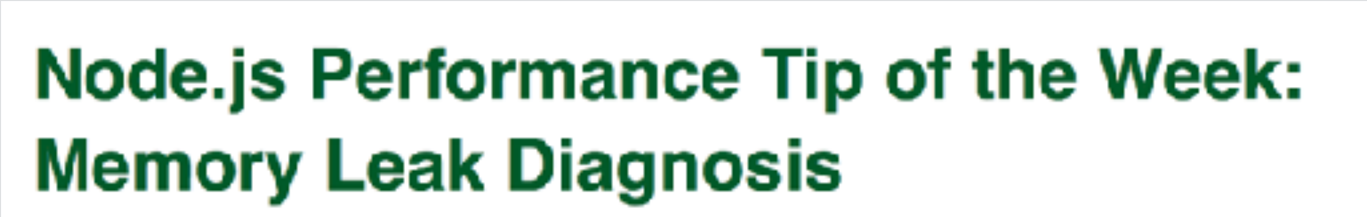
JS Apps and Memory



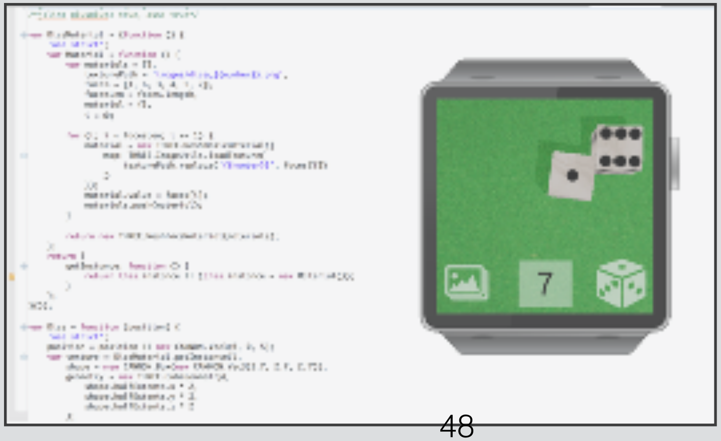
Google

BloatBusters:

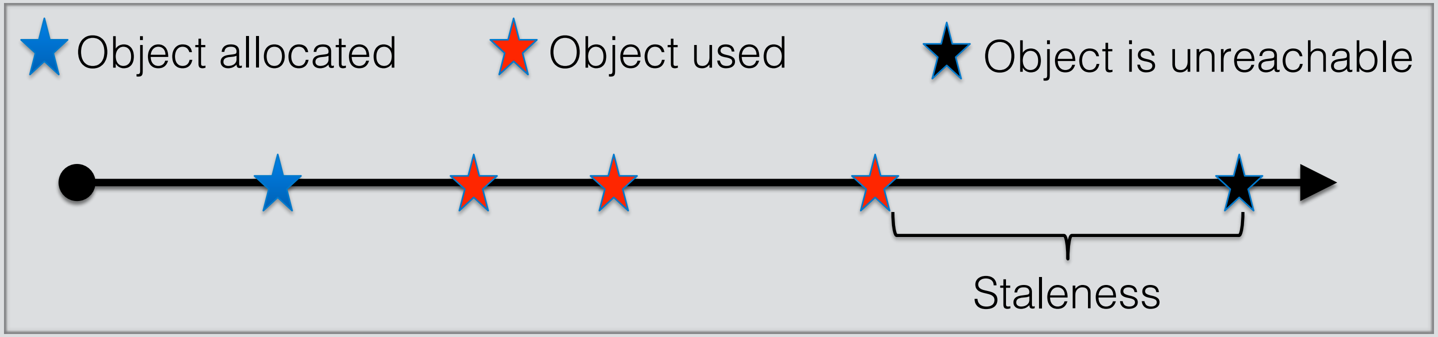
Eliminating memory leaks in Gmail



Node.js Performance Tip of the Week: Memory Leak Diagnosis



Leaks and Staleness

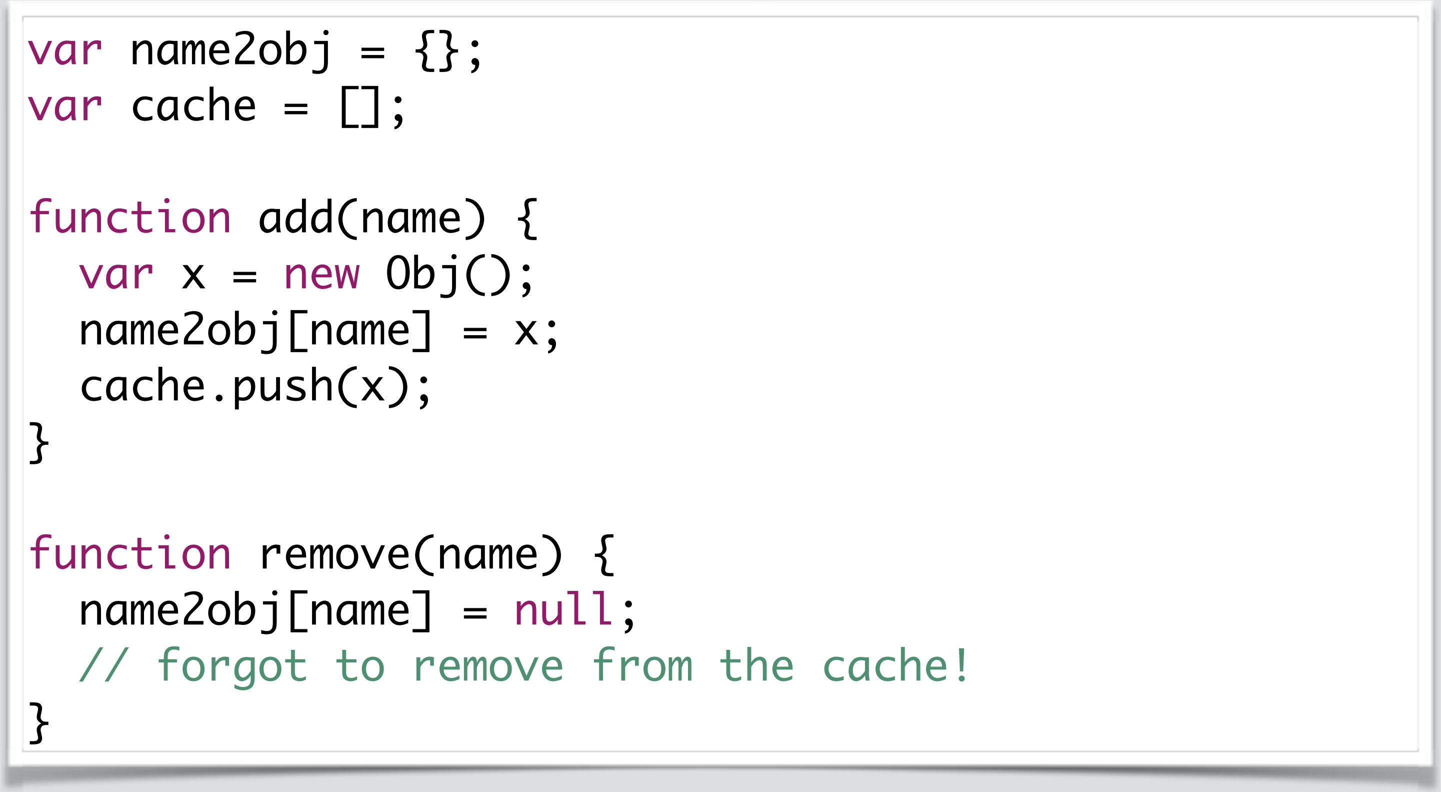


- **Staleness**: long gap between last use and unreachable

- **Leak**: never unreachable

- Many stale objects indicates a potential problem

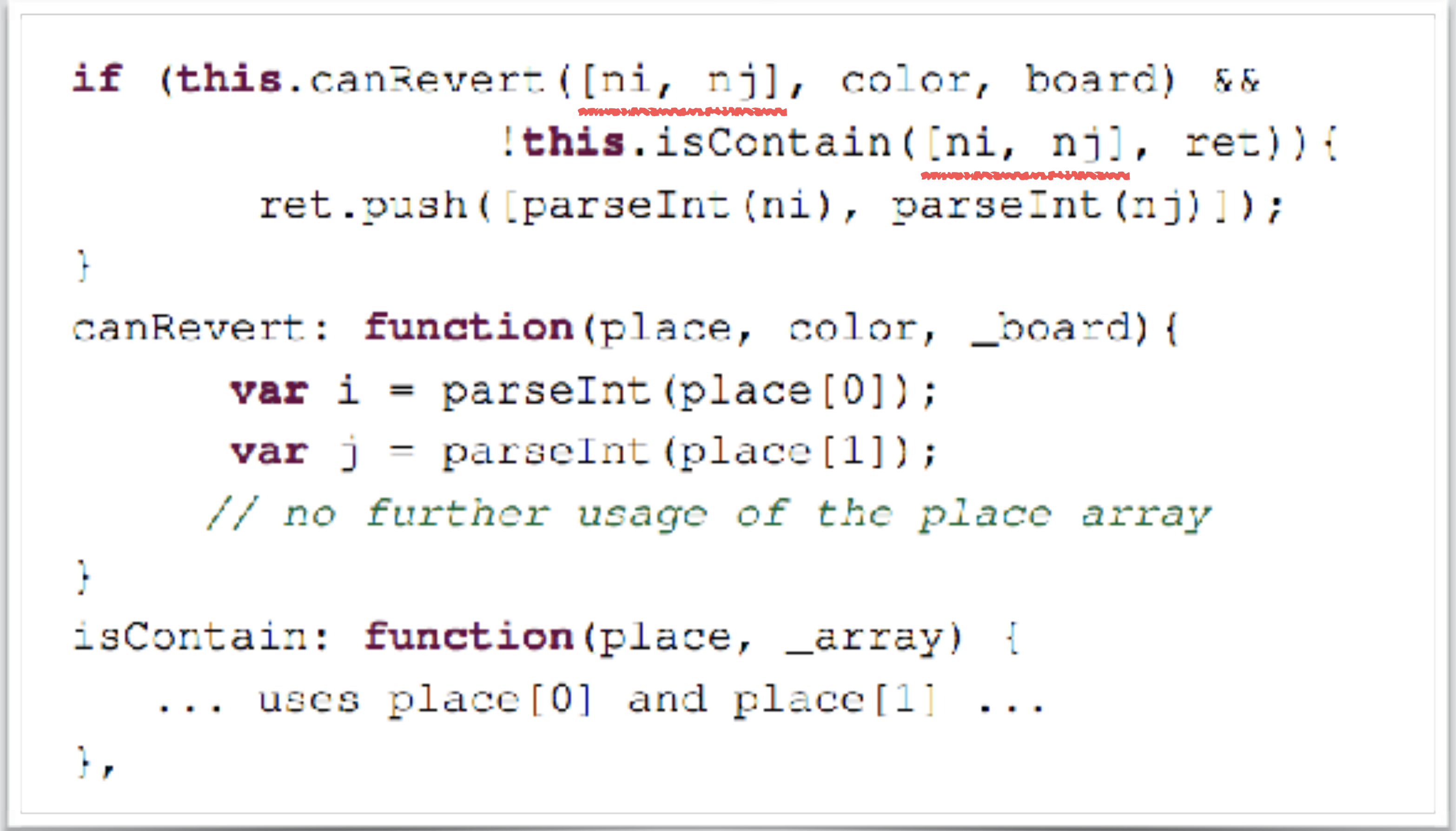
Leak Example



```
var name2obj = {};
var cache = [];

function add(name) {
  var x = new Obj();
  name2obj[name] = x;
  cache.push(x);
}

function remove(name) {
  name2obj[name] = null;
  // forgot to remove from the cache!
}
```

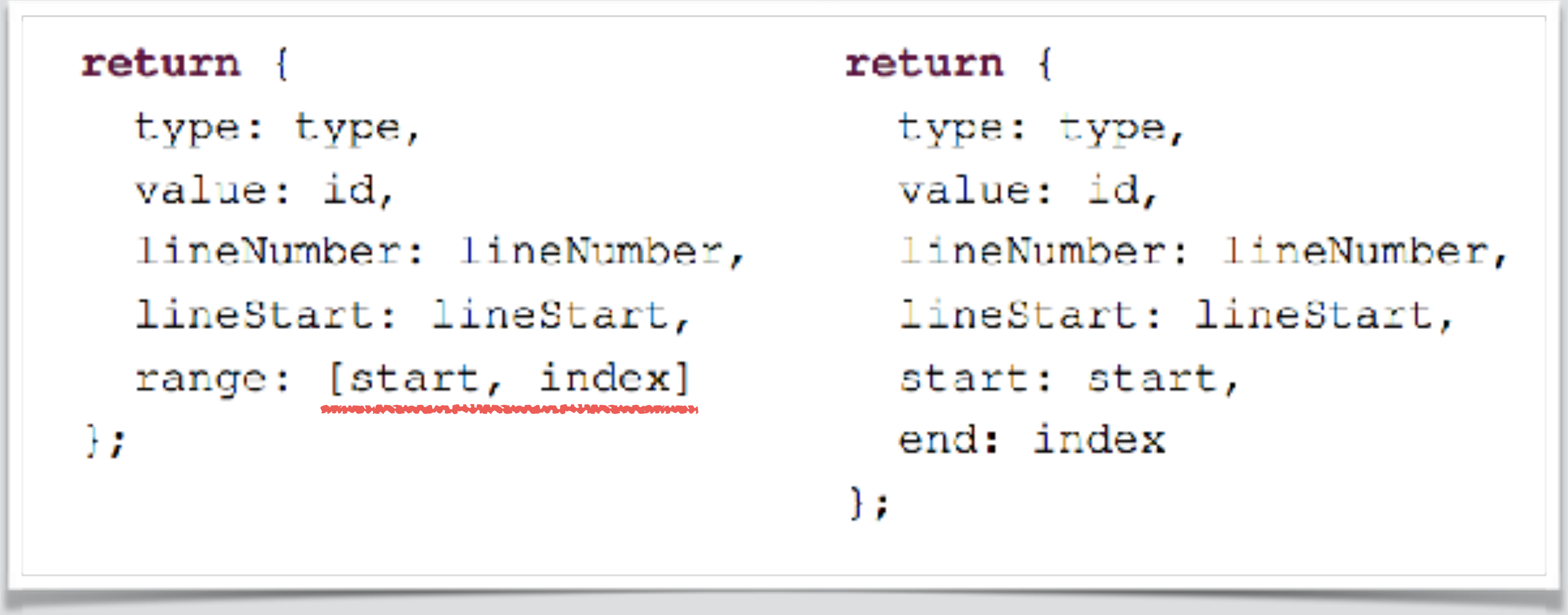
More insidious in web apps, where DOM nodes are involved

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] ...
},
```

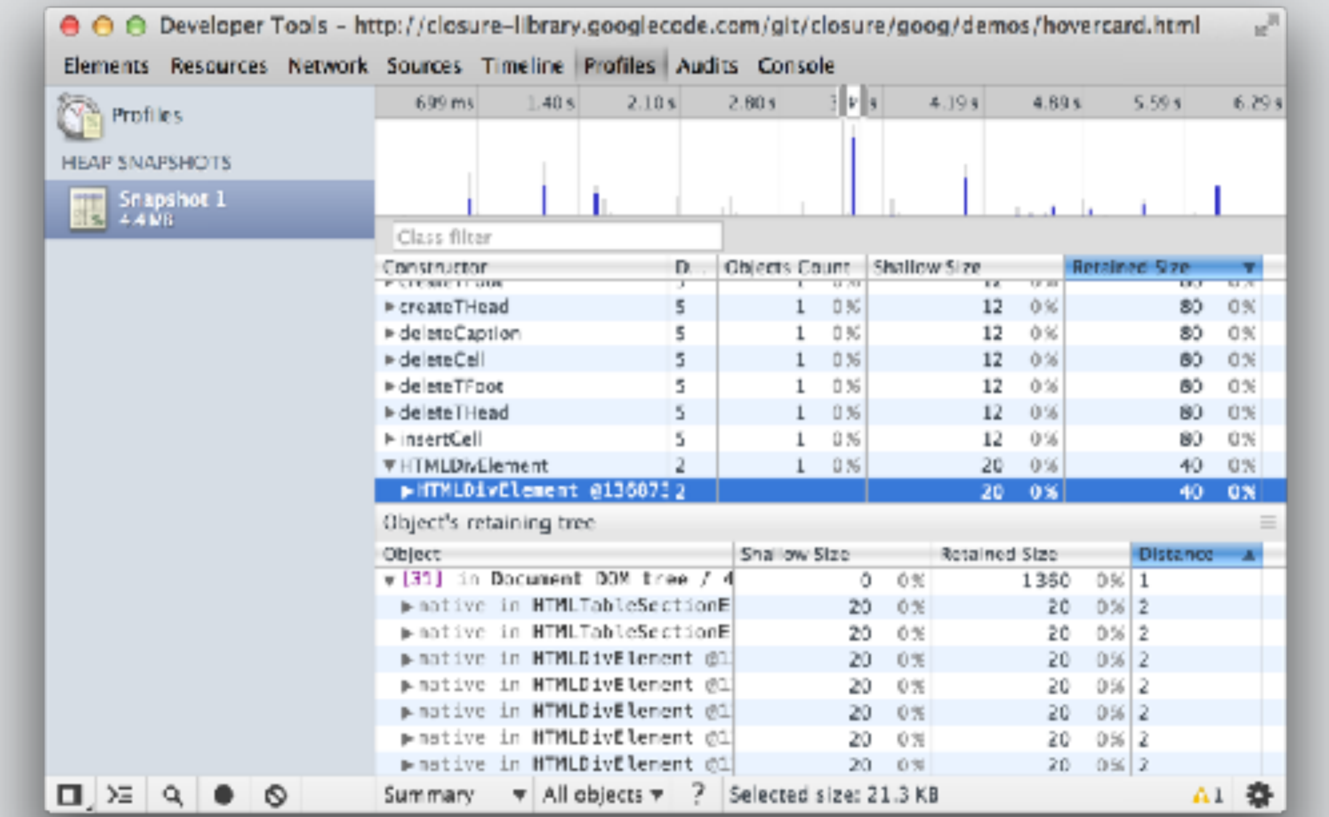
Bloat



```
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
};
```

Heap Snapshots



Constructor	D...	Objects Count		Shallow Size		Retained Size	
▸ createTHead	5	1	0%	12	0%	80	0%
▸ deleteCaption	5	1	0%	12	0%	80	0%
▸ deleteCell	5	1	0%	12	0%	80	0%
▸ deleteTFoot	5	1	0%	12	0%	80	0%
▸ deleteTHead	5	1	0%	12	0%	80	0%
▸ insertCell	5	1	0%	12	0%	80	0%
▾ HTMLDivElement	2	1	0%	20	0%	40	0%
▸ HTMLDivElement @136071	2			20	0%	40	0%

Object's retaining tree

Object	Shallow Size		Retained Size		Distance
▾ [31] in Document DOM tree / 4	0	0%	1360	0%	1
▸ native in HTMLTableSectionE	20	0%	20	0%	2
▸ native in HTMLTableSectionE	20	0%	20	0%	2
▸ native in HTMLDivElement @1	20	0%	20	0%	2
▸ native in HTMLDivElement @1	20	0%	20	0%	2
▸ native in HTMLDivElement @1	20	0%	20	0%	2
▸ native in HTMLDivElement @1	20	0%	20	0%	2
▸ native in HTMLDivElement @1	20	0%	20	0%	2

Selected size: 21.3 KB

Chrome Dev Tools

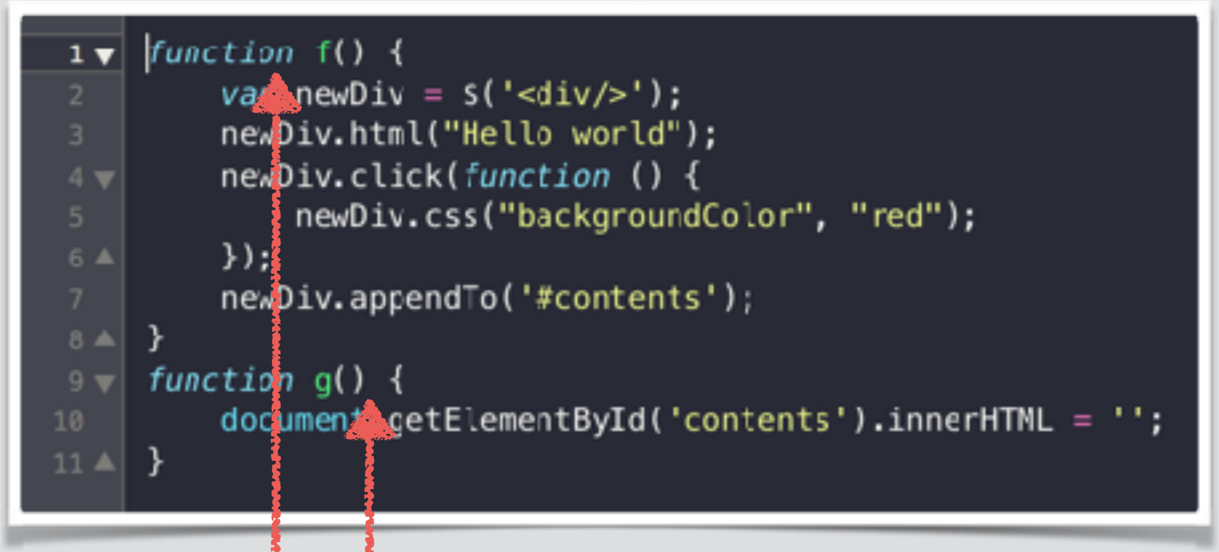
https://developers.google.com/chrome-developer-tools/docs/javascript-memory-profiling

Heap Snapshots

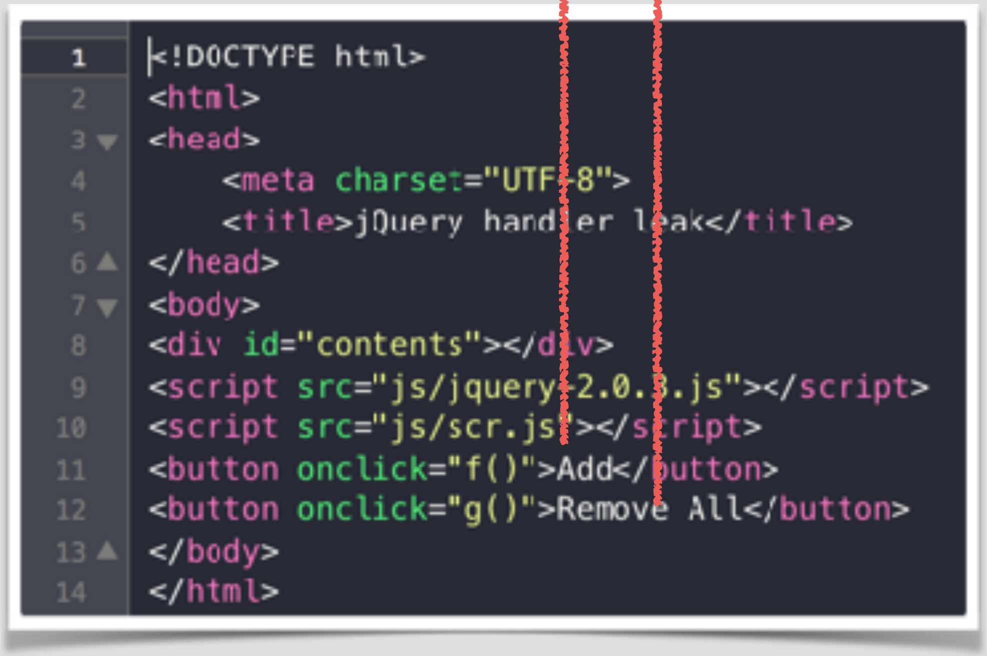
- Capture several snapshots, diff to find possible leaks
- Low overhead, but:
    - No information on staleness (does not track uses)
    - Can miss excessive churn
    - Cannot handle fine-grained time-varying properties

MemInsight

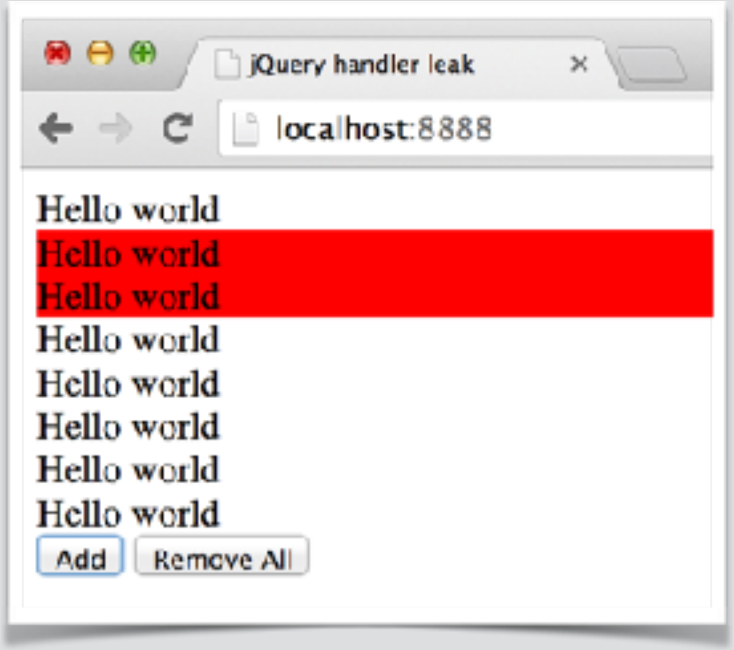
- **Platform independent:** use on any modern browser or node.js
- **Fine-grained behaviors via detailed tracing**
    - computes *exact object lifetimes*
    - enables a variety of client analyses
- **Exposes DOM manipulation**
- **Reasonable overhead**



```
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 = '';
}
```

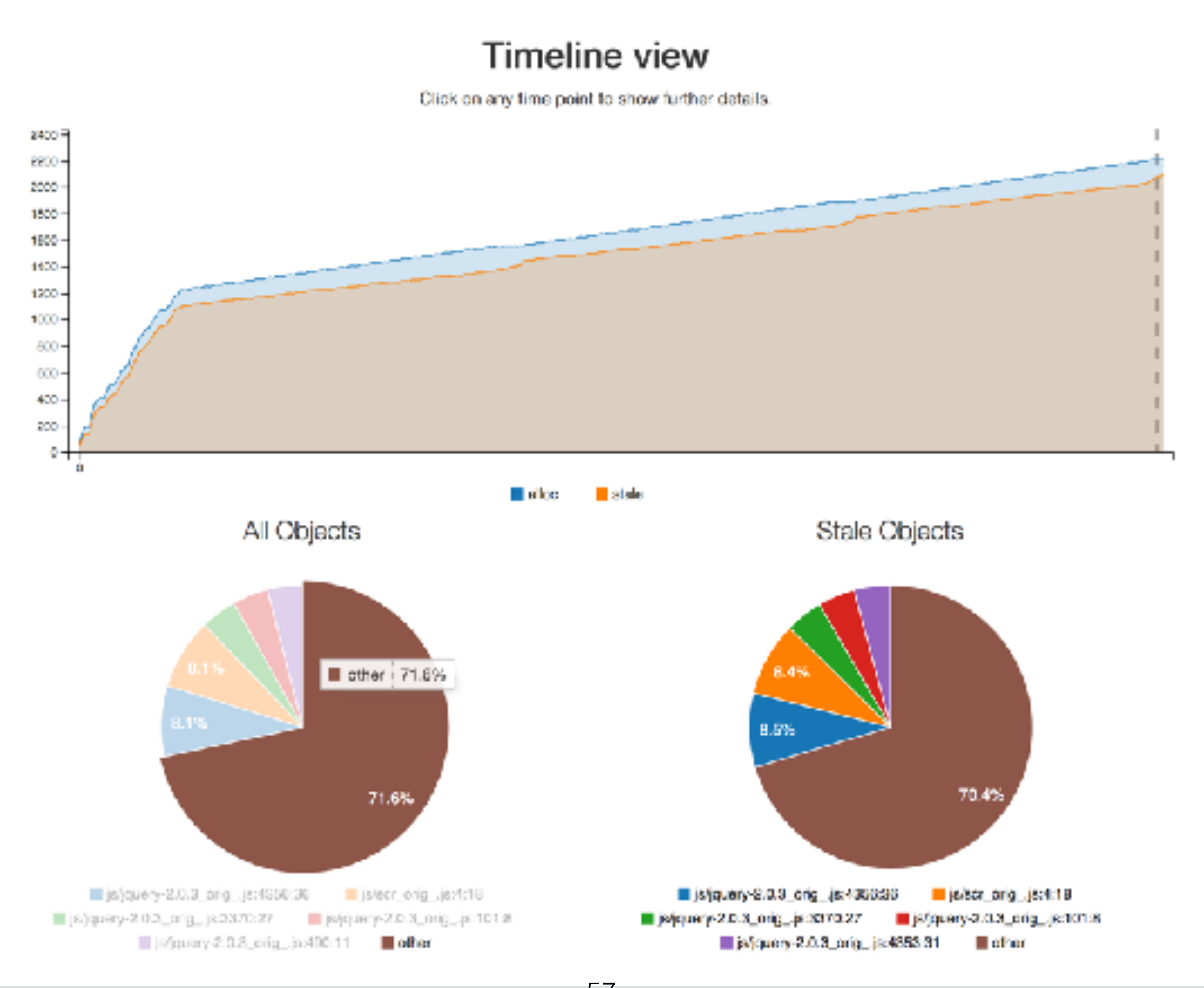


```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>jQuery handler leak</title>
</head>
<body>
<div id="contents"></div>
<script src="js/jquery-2.0.3.js"></script>
<script src="js/scr.js"></script>
<button onclick="f()">Add</button>
<button onclick="g()">Remove All</button>
</body>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



```
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
Hello world
```

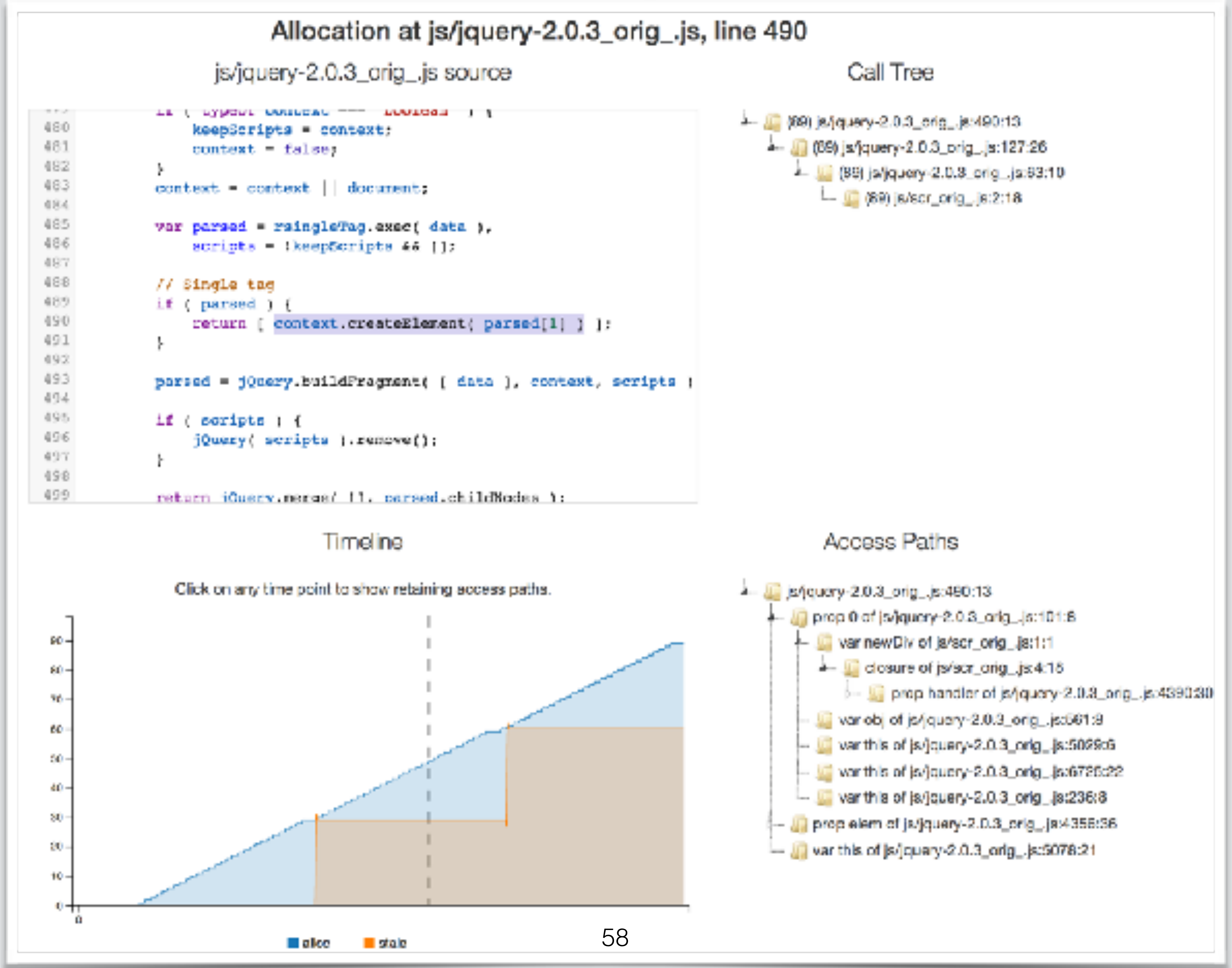
Memory leak!



Timeline view

Click on any time point to show further details.

Memory leak - Details



Allocation at js/jquery-2.0.3_orig_.js, line 490

js/jquery-2.0.3_orig_.js source

```
if ( typeof context === "boolean" ) {
    keepScripts = context;
    context = false;
}
context = context || document;

var parsed = rsingleTag.exec( data ),
    scripts = !keepScripts && [];

// Single tag
if ( parsed ) {
    return [ context.createElement( parsed[1] ) ];
}

parsed = jQuery.buildFragment( [ data ], context, scripts );

if ( scripts ) {
    jQuery( scripts ).remove();
}

return jQuery.merge( [], parsed.childNodes );
```

Call Tree

```
(69) js/jquery-2.0.3_orig_.js:490:13
  (69) js/jquery-2.0.3_orig_.js:127:26
    (69) js/jquery-2.0.3_orig_.js:93:10
      (69) js/scr_orig_.js:2:18
```

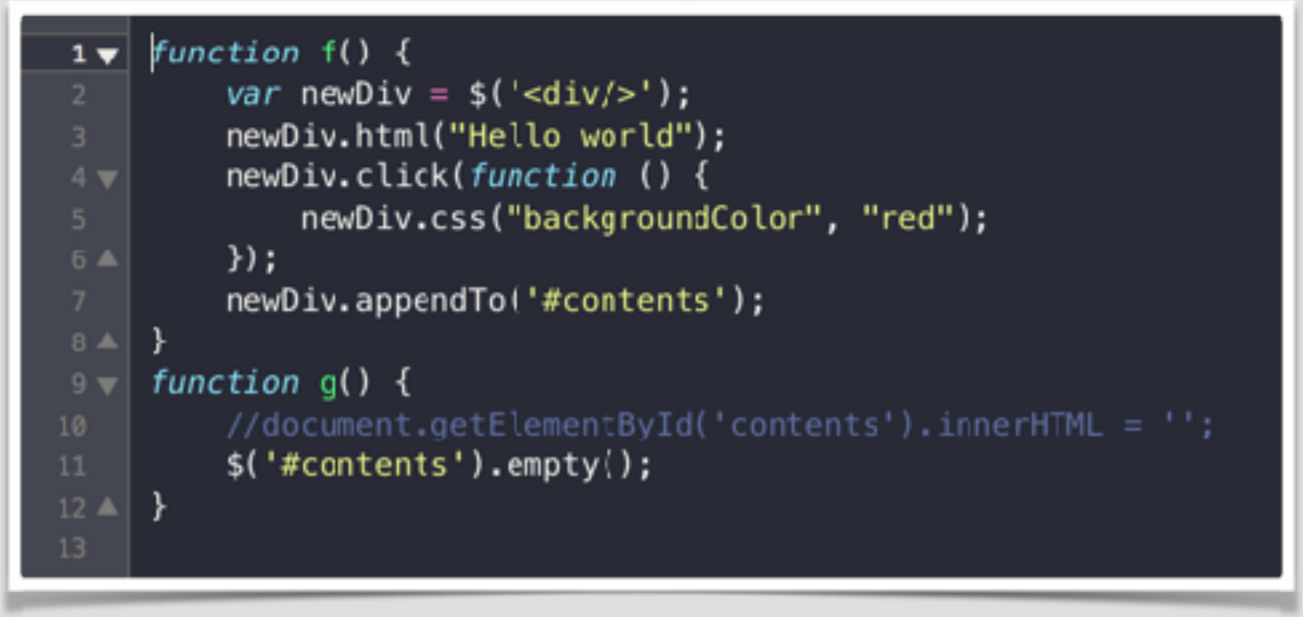
Timeline

Click on any time point to show retaining access paths.

Access Paths

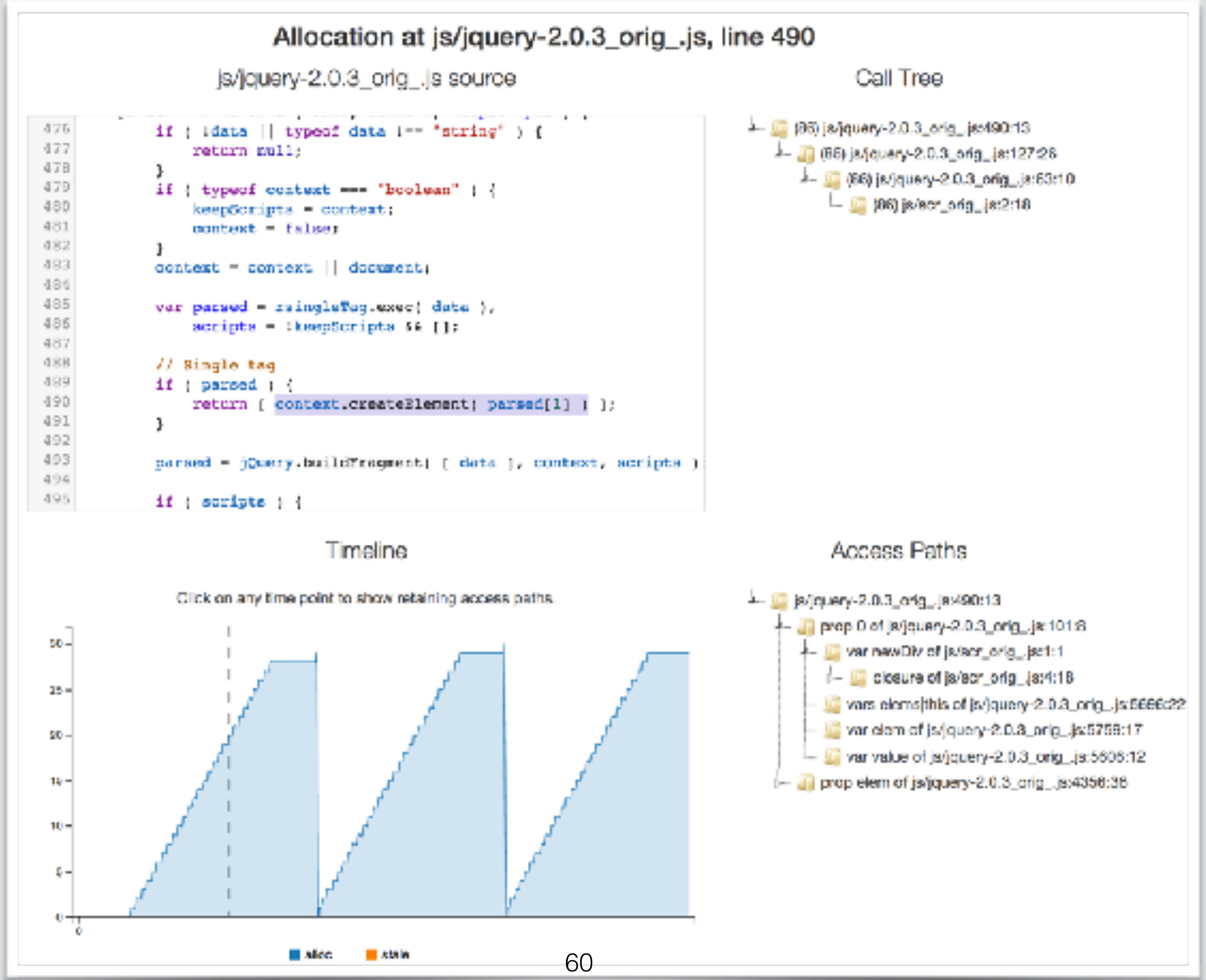
```
js/jquery-2.0.3_orig_.js:490:13
  prop 0 of js/jquery-2.0.3_orig_.js:101:8
    var newDiv of js/scr_orig_.js:1:1
      closure of js/scr_orig_.js:4:15
        prop handler of js/jquery-2.0.3_orig_.js:4390:30
    var obj of js/jquery-2.0.3_orig_.js:561:9
    var this of js/jquery-2.0.3_orig_.js:5099:5
    var this of js/jquery-2.0.3_orig_.js:6725:22
    var this of js/jquery-2.0.3_orig_.js:236:8
  prop elem of js/jquery-2.0.3_orig_.js:4356:36
  var this of js/jquery-2.0.3_orig_.js:5078:21
```

jQuery issue!



```
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 = '';
    $('#contents').empty();
}
```

Memory leak - Details



Allocation at js/jquery-2.0.3_orig_.js, line 490

js/jquery-2.0.3_orig_.js source

```
476         if ( !data || typeof data !== "string" ) {
477             return null;
478         }
479         if ( typeof context === "boolean" ) {
480             keepScripts = context;
481             context = false;
482         }
483         context = context || document;
484
485         var parsed = rsingleTag.exec( data ),
486             scripts = !keepScripts && [];
487
488         // Single tag
489         if ( parsed ) {
490             return [ context.createElement( parsed[1] ) ];
491         }
492
493         parsed = jQuery.buildFragment( [ data ], context, scripts );
494
495         if ( scripts ) {
```

Call Tree

```
(86) js/jquery-2.0.3_orig_.js:490:13
  (86) js/jquery-2.0.3_orig_.js:127:26
    (86) js/jquery-2.0.3_orig_.js:63:10
      (86) js/acr_orig_.js:2:18
```

Timeline

Click on any time point to show retaining access paths.

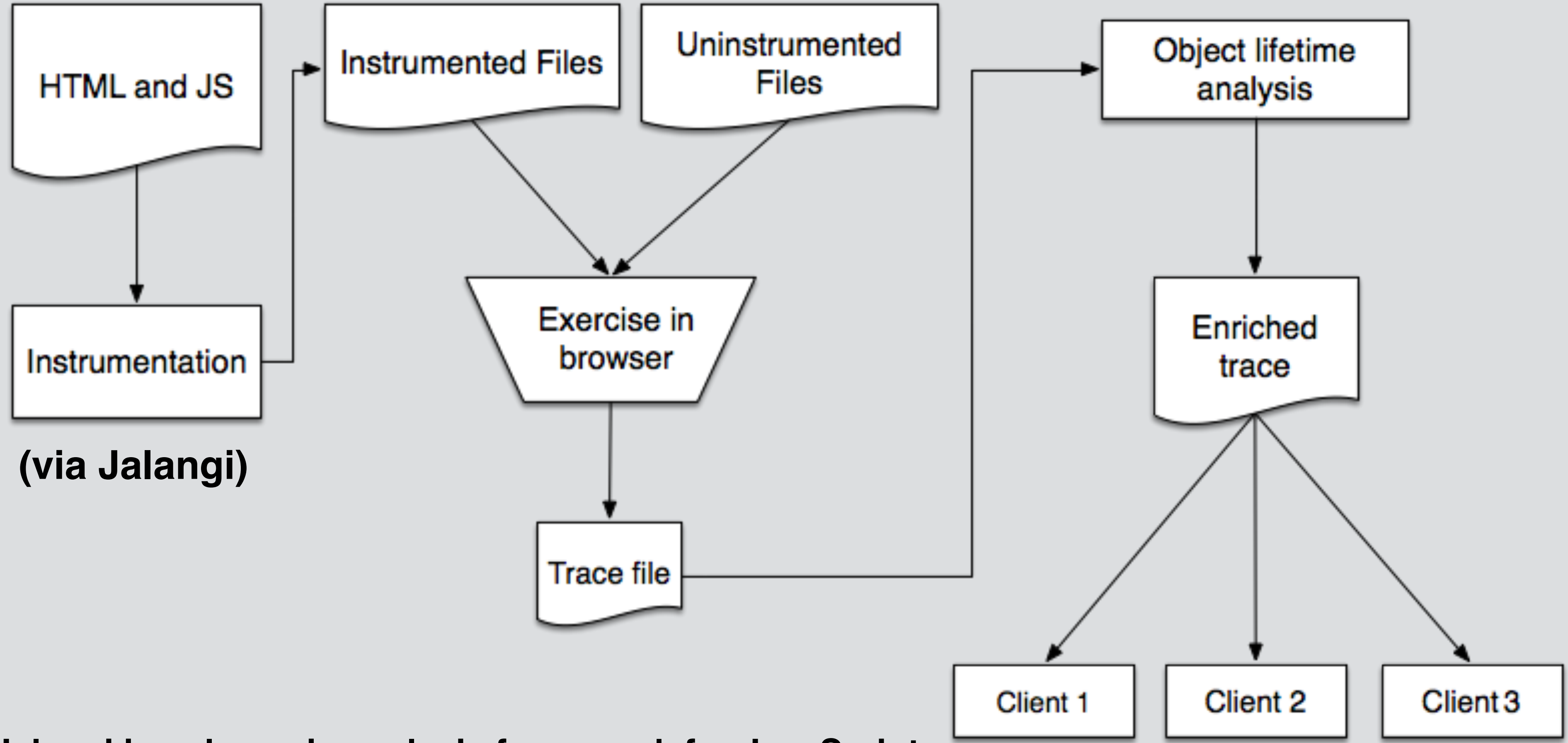
Access Paths

```
js/jquery-2.0.3_orig_.js:490:13
  prop 0 of js/jquery-2.0.3_orig_.js:101:8
    var newDiv of js/acr_orig_.js:1:1
      closure of js/acr_orig_.js:4:18
    vars elems|this of js/jquery-2.0.3_orig_.js:5666:22
    var elem of js/jquery-2.0.3_orig_.js:5758:17
    var value of js/jquery-2.0.3_orig_.js:5606:12
  prop elem of js/jquery-2.0.3_orig_.js:4356:38
```

Challenges

- Prefer not to modify a browser engine
  - Yet handle full JavaScript
  - Keep overhead reasonable
- Want to report staleness of DOM nodes, without modifying browser
- Figure out object lifetimes accurately without information from the garbage collector

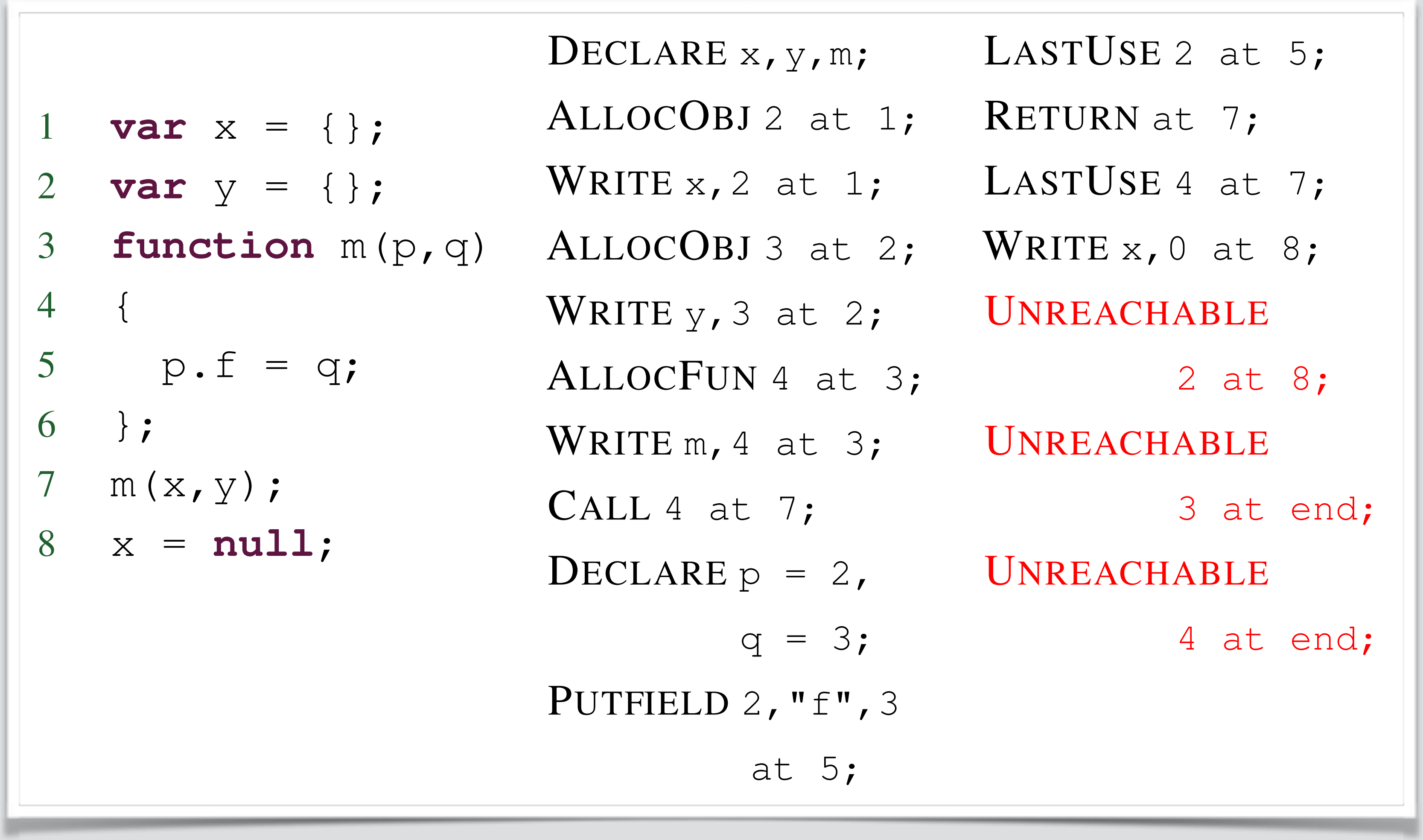
How does MemInsight work?



(via Jalangi)

Jalangi is a dynamic analysis framework for JavaScript
See FSE 2013, Sen et al.

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;
```

Trace generation

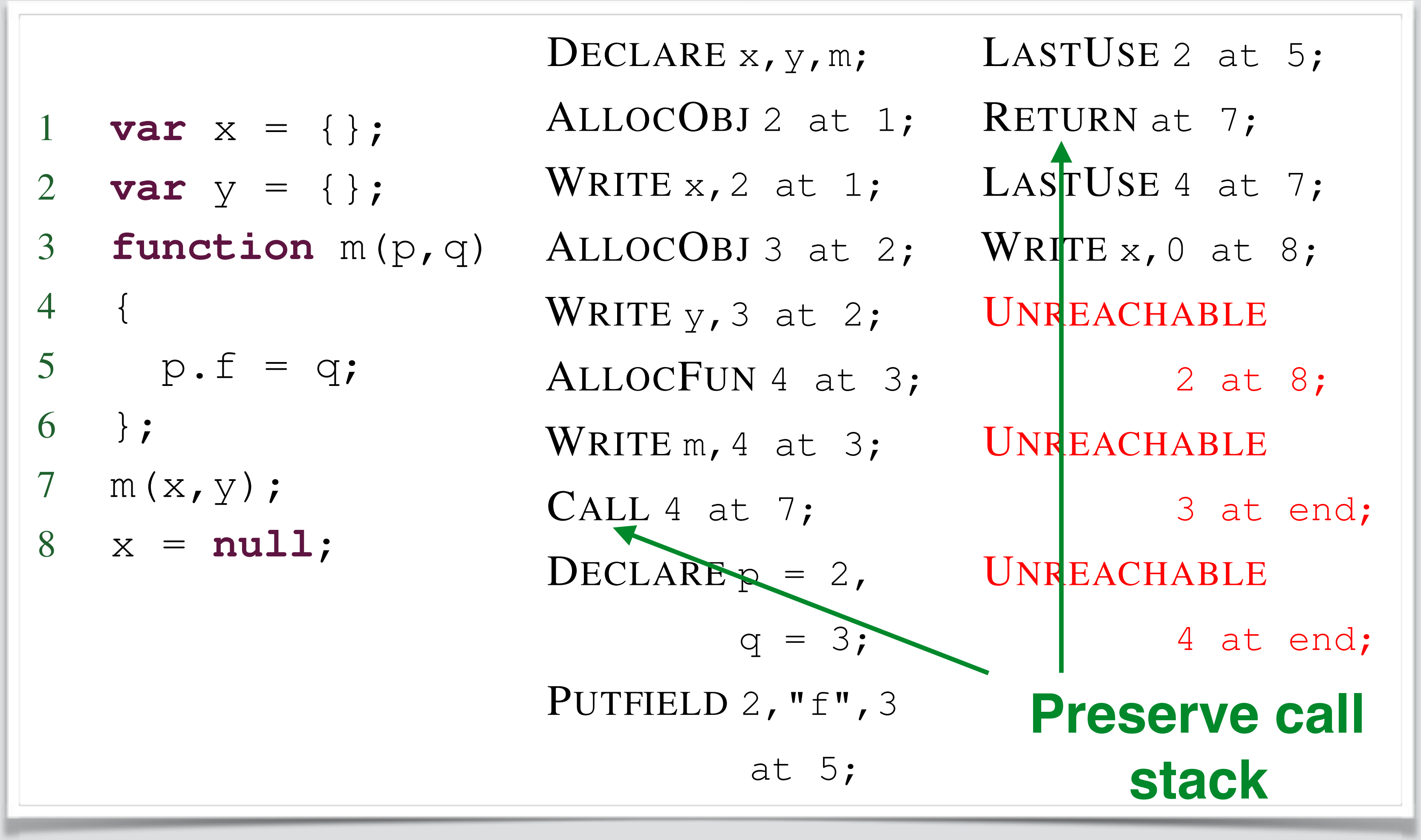
```
1  var x = {};
2  var y = {};
3  function m(p,q)
4  {
5    p.f = q;
6  };
7  m(x,y);
8  x = null;
```

```
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;
```

Preserve line numbers

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;
```

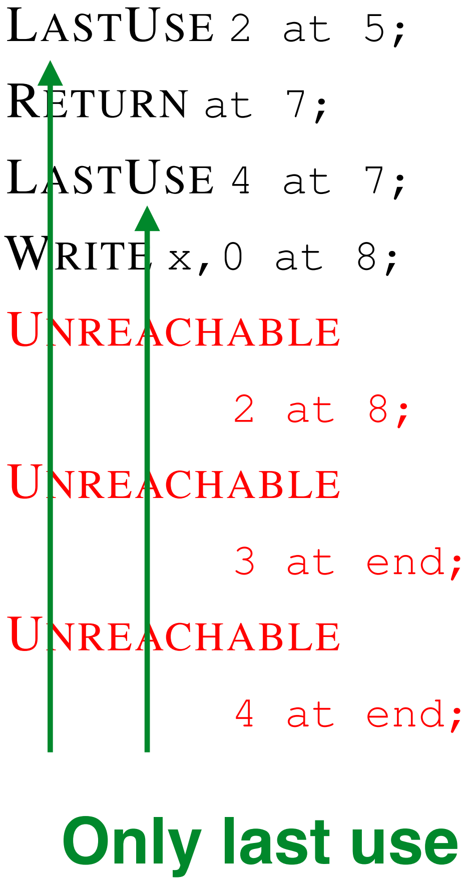
```
LASTUSE 2 at 5;
RETURN at 7;
LASTUSE 4 at 7;
WRITE x,0 at 8;
UNREACHABLE
        2 at 8;
UNREACHABLE
        3 at end;
UNREACHABLE
        4 at end;
```

Preserve call stack

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;
```

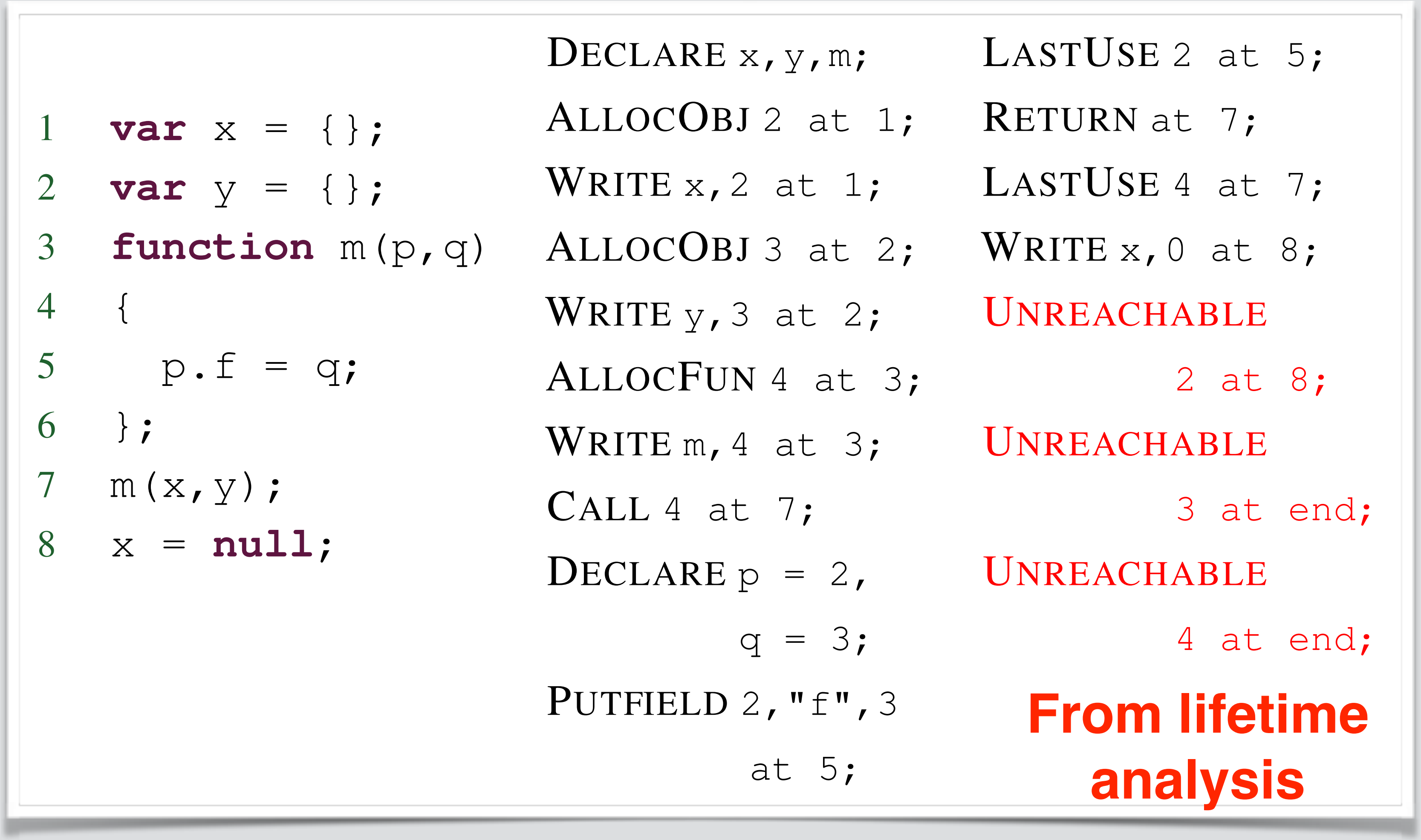
```
DECLARE x,y,m;
ALLOCOBJ 2 at 1;
WRITE x,2 at 1;
ALLOCOBJ 3 at 2;
WRITE y,3 at 2;
ALLOCFUN 4 at 3;
WRITE m,4 at 3;
CALL 4 at 7;
DECLARE p = 2,
        q = 3;
PUTFIELD 2,"f",3
        at 5;
```



```
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;
```

Only last use

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;
```

From lifetime analysis

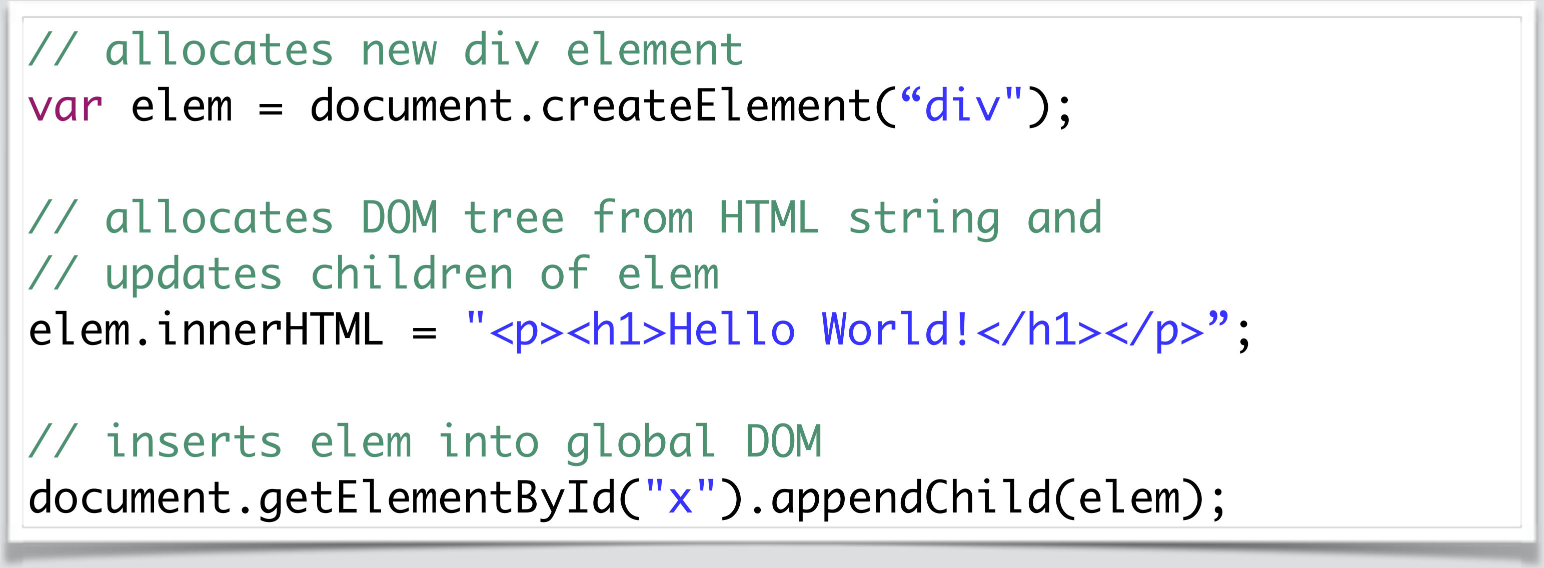
Object lifetimes

- 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

DOM Challenges

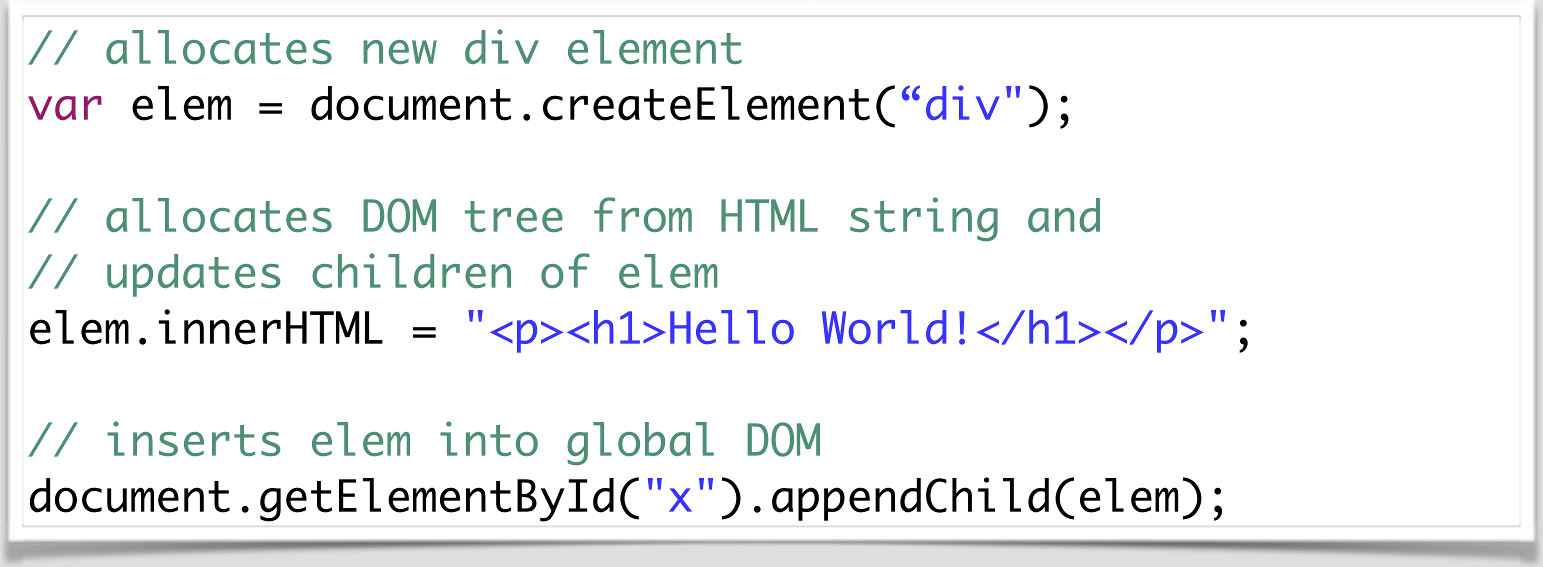
- DOM: tree data structure representing rendered HTML
  - Often involved in web app memory leaks

- Many manipulations not directly visible to JavaScript



```
// allocates new div element
var elem = document.createElement(“div");

// allocates DOM tree from HTML string and
// updates children of elem
elem.innerHTML = "<p><h1>Hello World!</h1></p>”;

// inserts elem into global DOM
document.getElementById("x").appendChild(elem);
```

Our DOM Handling



```
// allocates new div element
var elem = document.createElement(“div");

// allocates DOM tree from HTML string and
// updates children of elem
elem.innerHTML = "<p><h1>Hello World!</h1></p>";

// inserts elem into global DOM
document.getElementById("x").appendChild(elem);
```

- `elem` gets reified into a fresh object ID
  - no special handling of `createElement`

- For DOM manipulations, leverage HTML5 mutation observers
  - Provide asynchronous notifications of DOM mutation
  - Handles `innerHTML` manipulation and `appendChild`

- Additional handling of `innerHTML` for better source locations

Other tricky features

- **Constructors:** need to properly handle `this`, and get good source locations
- **Eval:** instrument on the fly
- **Getters / setters:** don’t treat calls as reads / writes
- Global object, prototypes, further native models, …

Clients built atop MemInsight

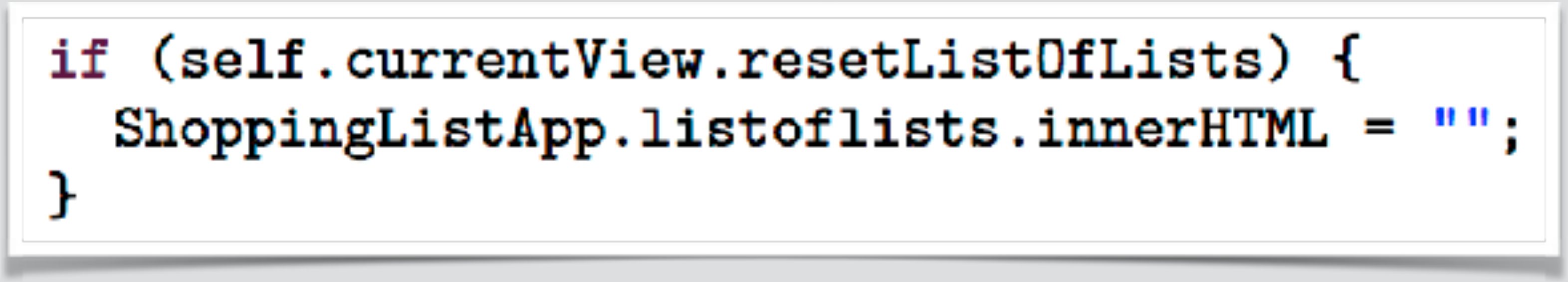
- **Leak detection:** increasing stale object count at idle points (empty call stack)
- **Non-escaping:** no object escapes allocating function
    - Leverages *execution index* [Xin et al. PLDI'08]
- **Inlineable:** objects consistently "owned" by objects from another site
- Many more are possible!

Case Studies

(see paper for details)

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

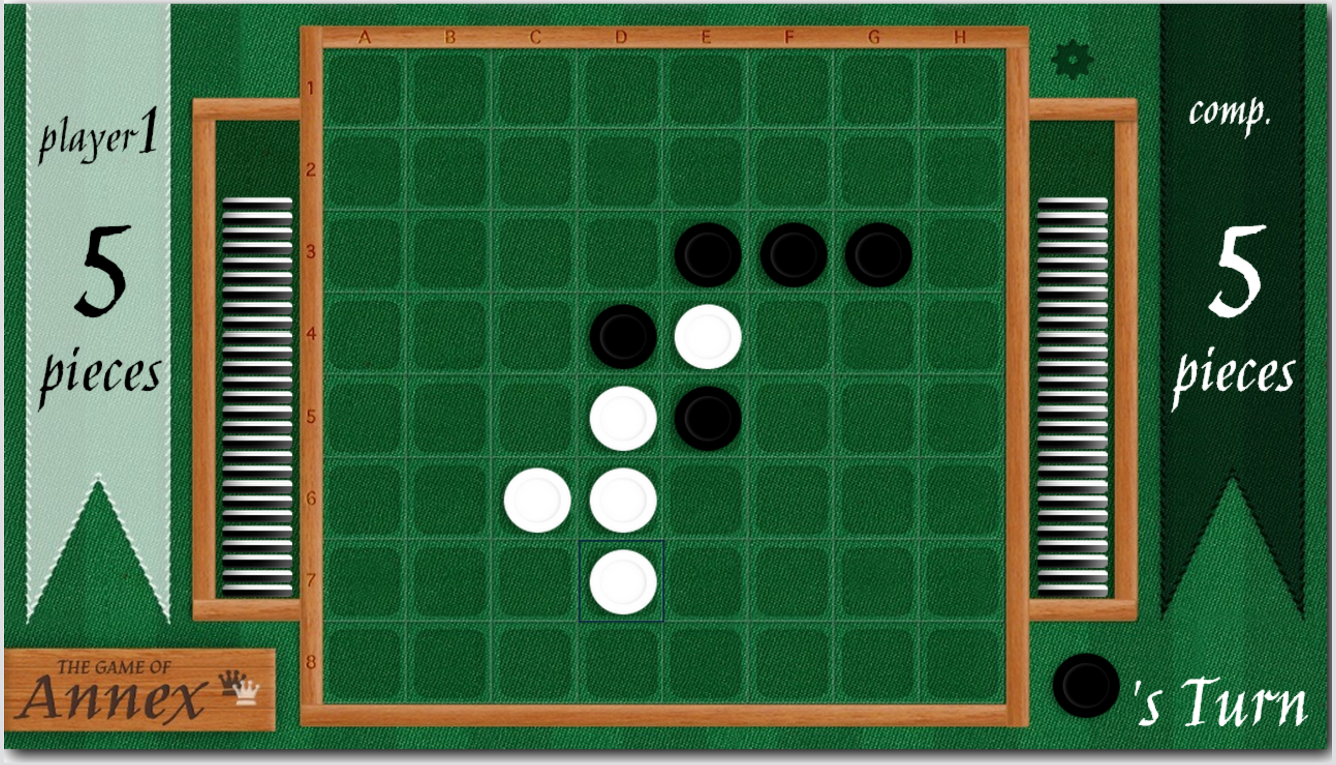
Leak in Shopping List app



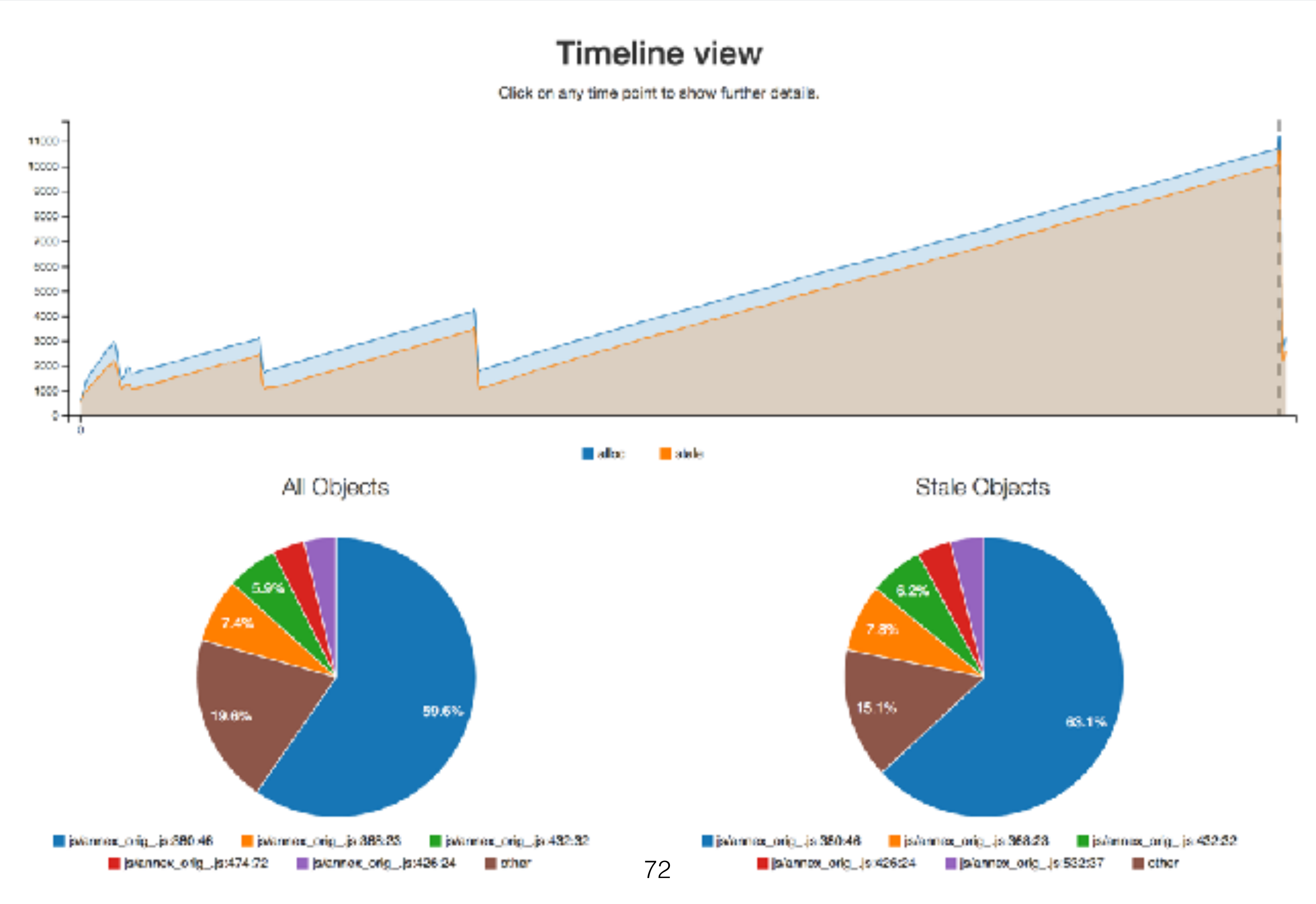
```
if (self.currentView.resetListOfLists) {
  ShoppingListApp.listoflists.innerHTML = "";
}
```

Should have used `$.empty()`!

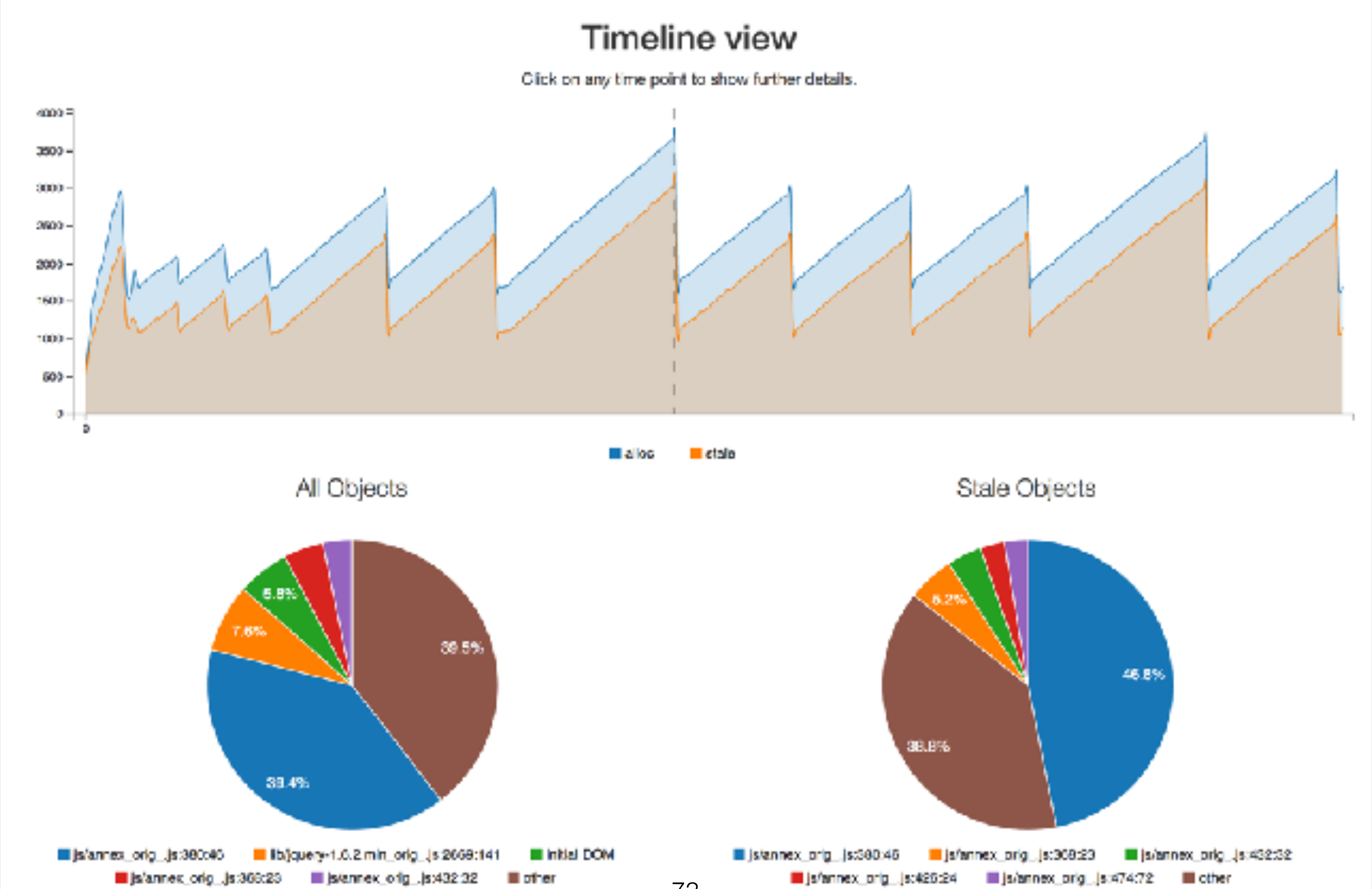
Run an instrumented app



Interactive staleness analysis



Interactive staleness analysis



Timeline view

Click on any time point to show further details.

Overhead

benchmark	overhead
richards	10.4X
deltablue	15X
crypto	47.1X
raytrace	41.3X
earley-boyer	99.8X
regexp	26.7X
splay	43.4X
navier-stokes	45.4X
pdfjs	31.8X
box2d	35.8X
typescript	77.2X

Low overhead for (most) interactive apps

Reducing Overhead

- Only log the last use of an object (**not** all uses)
- Don't log operations on primitive fields
- Enhanced Jalangi to do selective instrumentation
- Binary trace format
- Work with simulated heap as opposed to real heap
    - Reflection too expensive / fragile

Advanced Jalangi Usage

Tracing

- Common technique: store a trace, and do heavyweight analysis over the trace
    - Supported directly in Jalangi 1 via record/replay
    - But, hard to debug and write analyses
- `lib/analysis/Loggers.ts` has all analysis tracing code
- Under Node.js, dump trace to file system (`BinaryFSLogger`)
- From web, trace over web socket (`BinaryWebSocketLogger`)
    - `lib/server/server.ts` has server code
    - pipes trace directly to running lifetime analysis

Integrating Static Analysis

- MemInsight needs the “free variables” of each function
  - Captured by closures, relevant for lifetimes
- Computed by `freeVarsAstHandler.ts`
- Provided as an *AST handler* to Jalangi instrumentation
- Jalangi stores result of AST handler inside instrumented code
- For eval’d code, use the `instrumentCode` callback

Native Methods

- Built-in methods that cannot be instrumented
  - Standard JS library, DOM routines
  - (In general, any uninstrumented code)
- Modeling is analysis-specific
  - For MemInsight, `lib/analysis/NativeModels.ts`
- Also, careful with callbacks from native methods
  - may see `functionEnter` without `invokeFunPre`

Analysis Configuration

- May want analysis-wide configuration options
  - E.g., MemInsight allows for a debug function for dumping ref counts
- Use --`initParam` option to `instrument.js` (web) or `esnstrument_cli.js` (node.js)
- values stored in `J$.initParams`

Debugging with JSDelta

https://github.com/WALA/jsdelta

JSDelta: motivation

- Building a Jalangi analysis
- Works great on unit tests
- But, crashes on jQuery!
- What went wrong? Need a **minimized input**
- Jsdelta does **automatic input minimization**
  - Via **delta debugging** [Zeller, FSE'99]

JSDelta: Demo

Google “JS Delta Walkthrough”

Using JSDelta

- Easy: write a script that prints a message when error occurs
- Also works for JSON, entire directories
- For a Jalangi analysis:
  - Check for errors in uninstrumented program first
  - Always run with a timeout (e.g., with `timeout` command)
  - For browser code, use PhantomJS, Selenium, etc.

DLint and JITProf

DLint: Dynamically Checking JS Coding Practice

[ISSTA’15] DLint: Dynamically Checking Bad Coding Practices in JavaScri

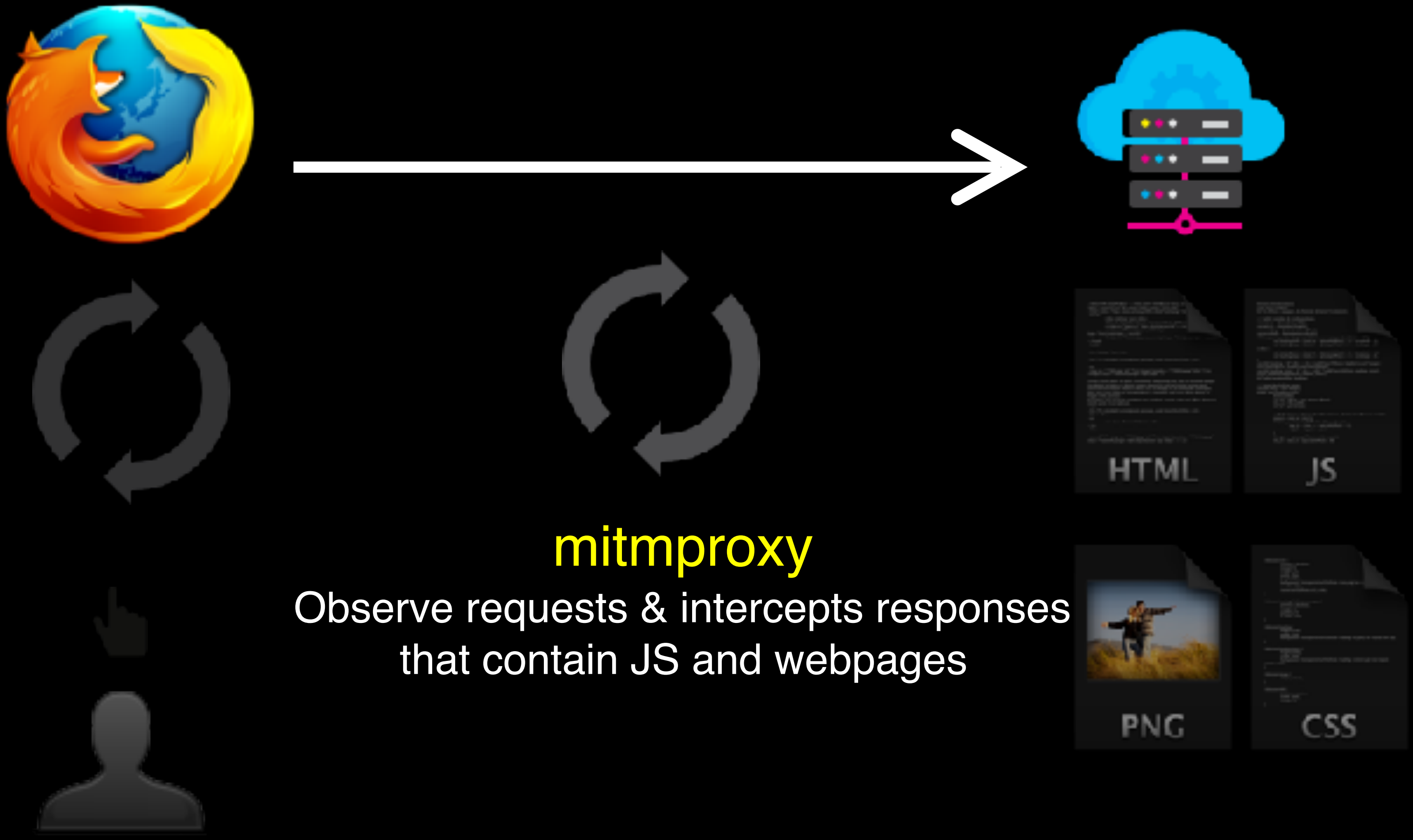
Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization

[FSE’15] JITProf: Pinpointing JIT-unfriendly JavaScript code

Liang Gong, Michael Pradel, Koushik Sen

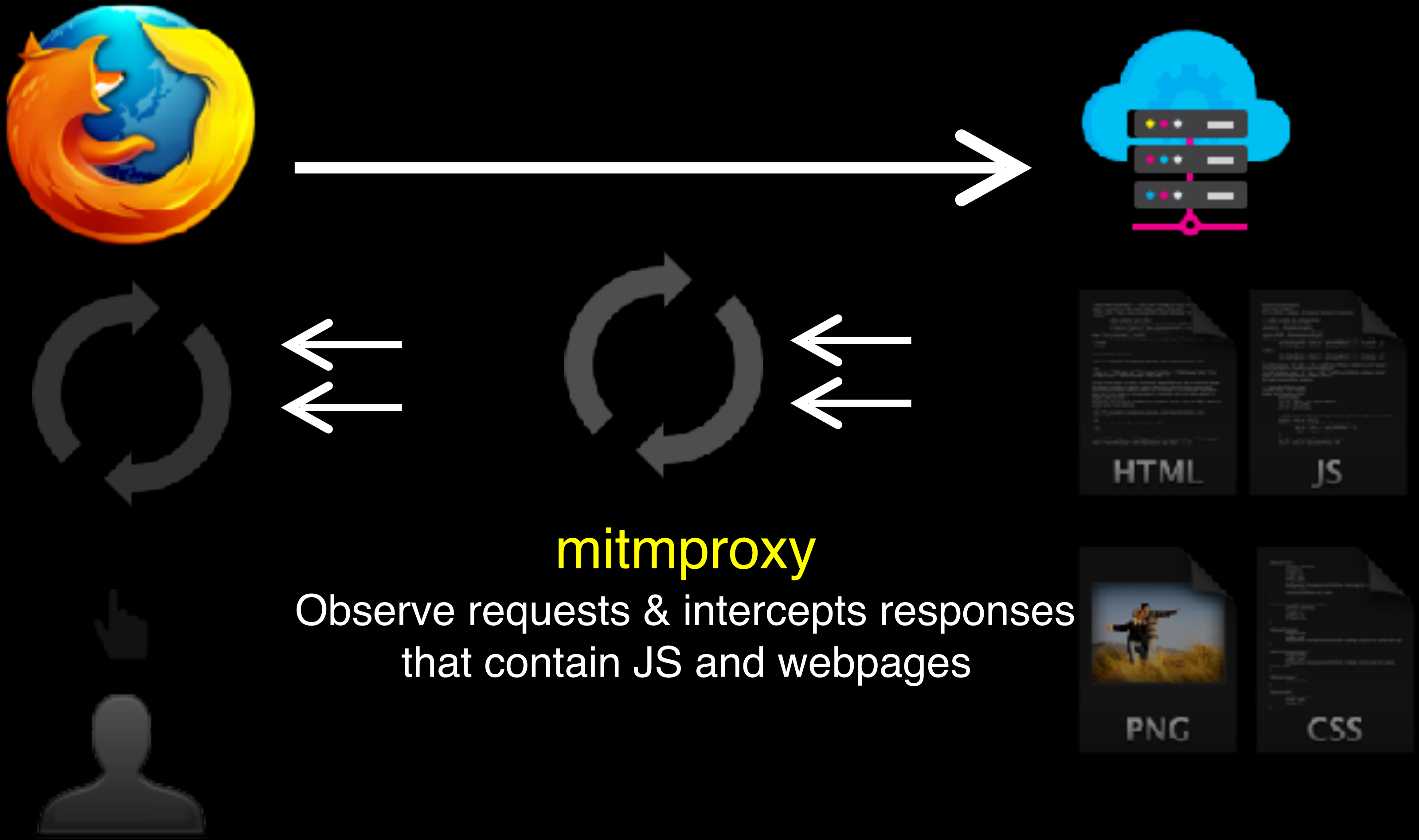
DLint and JITProf for Web Pages



mitmproxy

Observe requests & intercepts responses
that contain JS and webpages

DLint and JITProf for Web Pages



mitmproxy

Observe requests & intercepts responses that contain JS and webpages

DLint and JITProf

DLint: Dynamically Checking JS Coding Practice

[ISSTA'15] DLint: Dynamically Checking Bad Coding Practices in JavaScri

Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization

[FSE'15] JITProf: Pinpointing JIT-unfriendly JavaScript code

Liang Gong, Michael Pradel, Koushik Sen

What are coding practices?

- Good coding practices
  - Informal rules
  - Improve code quality

- Better quality means:
  - Fewer correctness issues
  - Better performance
  - Better usability
  - Better maintainability
  - Fewer security loopholes
  - Fewer surprises
  - …

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

✗ `11 + 22 + 33 => 66`

array index
(not array value)

✗ `0 + 1 + 2 => 3`

array index : string

✓ `0+"0"+"1"+"2" => "0012"`

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

☒ `11 + 22 + 33 => 66`

array index
(not array value)

☒ `0 + 1 + 2 => 3`

array index : string

☒ ✓ `0+"0"+"1"+"2" => "0012"`

`> "0012indexOftoString..."`

- Cross-browser issues
- Result depends on the Array prototype object

Rule: avoid using *for..in* over arrays

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

✓

```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

✓

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

Rule: avoid using *for..in* over arrays

⚠

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

✓

```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

✓

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

Coding Practices and Lint Tools

- Existing Lint-like checkers
  - Inspect source code
  - Detect common mistakes
- Limitations:
  - Approximates behavior
  - Unknown aliases
  - Lint tools favor precision over soundness
- Difficulty: Precise static program analysis

DLint

- Dynamic Linter checking code quality rules for JS
- Open-source, robust, and extensible framework
- Formalized and implemented 28 rules
  - Counterparts of static rules
  - Additional rules
- Empirical study
  - It is better to use DLint and static linter together

Detect *for..in* over arrays with Jalangi

⚠

```
var sum = 0, value;
var array = [11, 22, 33];
for (value in array) {
    sum += value;
}
> sum ?
```

✓

```
for (i=0; i < array.length; i++) {
    sum += array[i];
}
```

✓

```
function addup(element, index, array) {
  sum += element;
}
array.forEach(addup);
```

Detect *for..in* over arrays with Jalangi

```
for (value in obj) {
    sum += value;
}
```

Detect *for..in* over arrays with Jalangi

```
for (value in obj) {
    sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

Detect *for..in* over arrays with Jalangi

```
for (value in obj) {
    sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

Detect *for..in* over arrays with Jalangi

```
for (value in obj) {
    sum += value;
}
```

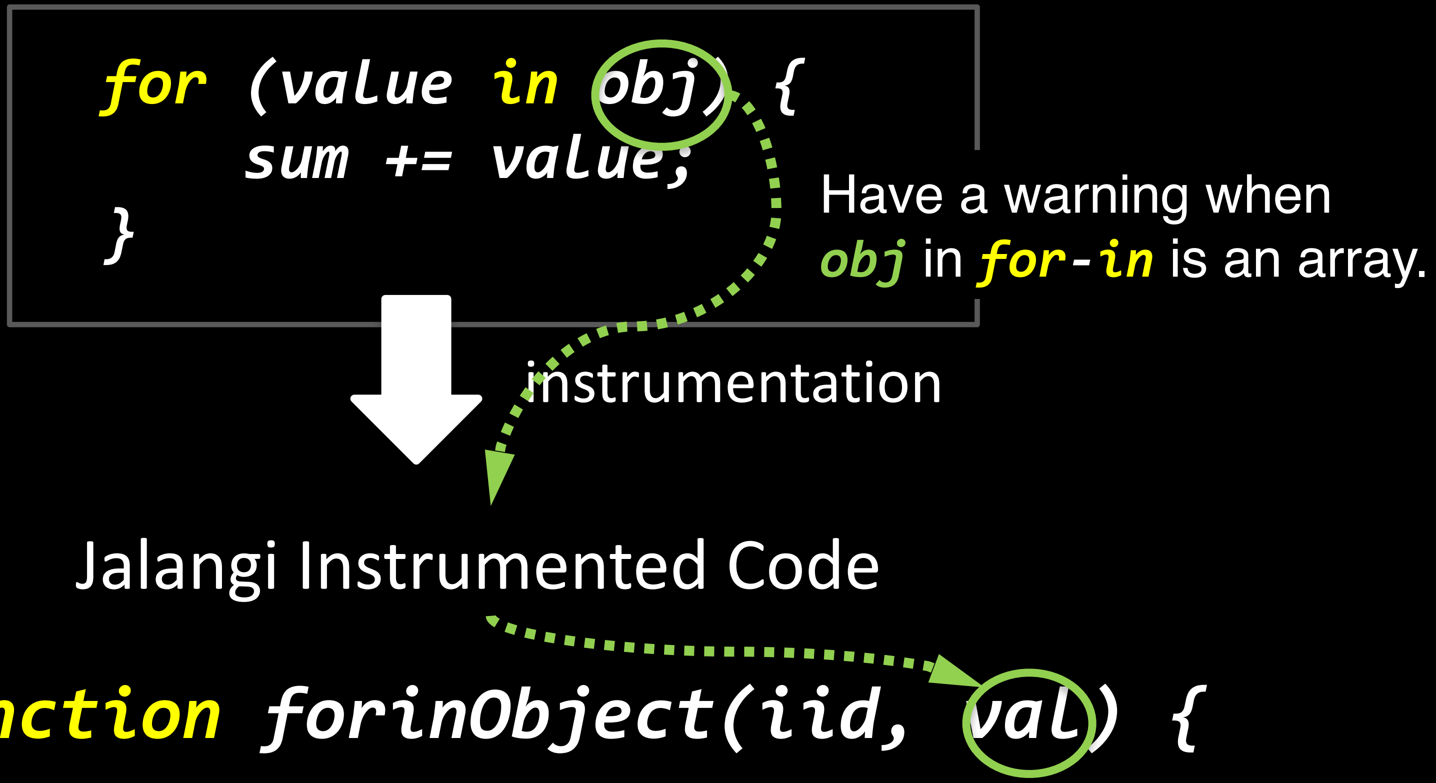
Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {



}
```

Detect *for..in* over arrays with Jalangi



```
for (value in obj) {
    sum += value;
}
```

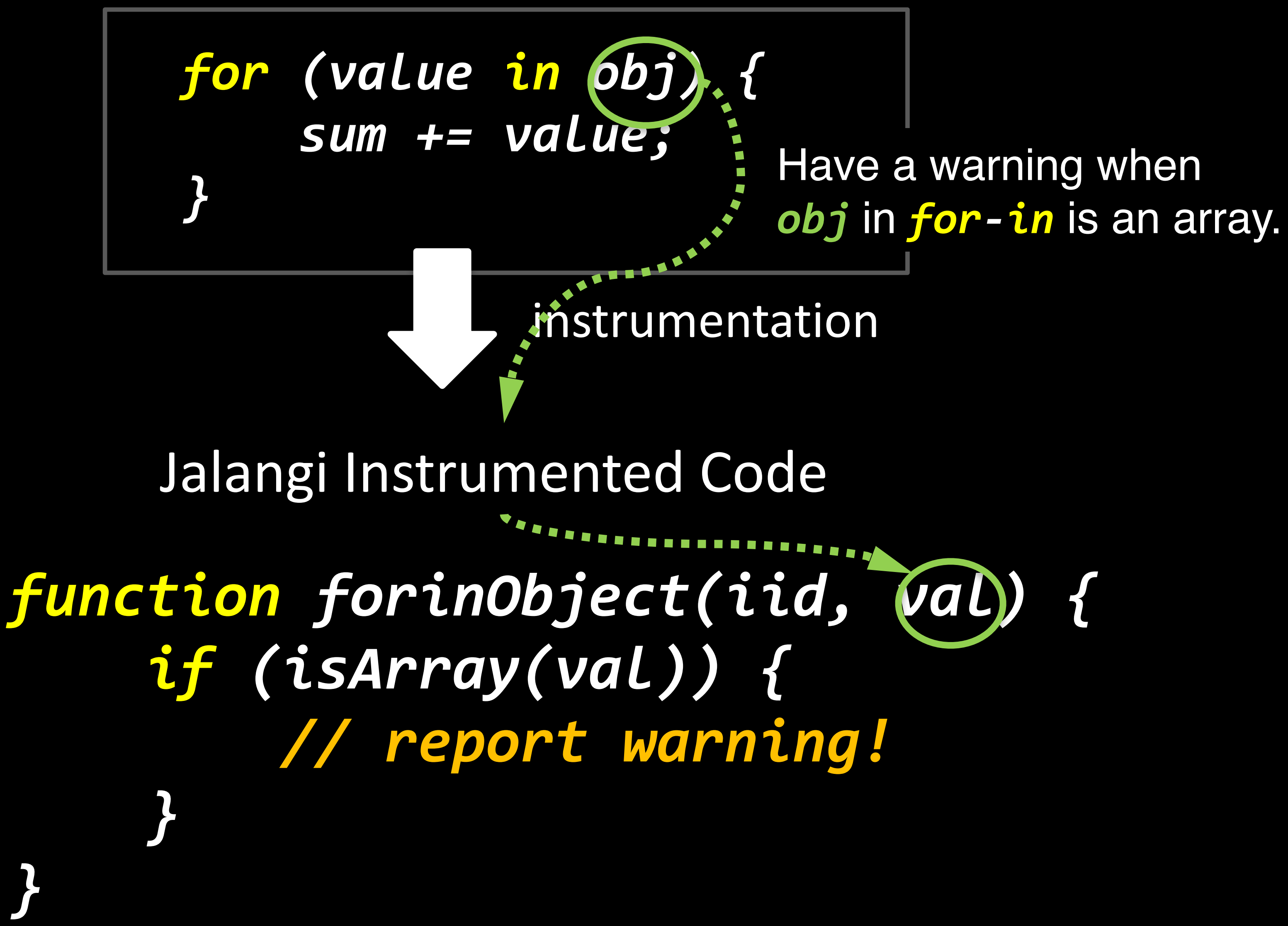
Have a warning when
obj in for-in is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {




}
```

Detect *for..in* over arrays with Jalangi



```
for (value in obj) {
    sum += value;
}
```

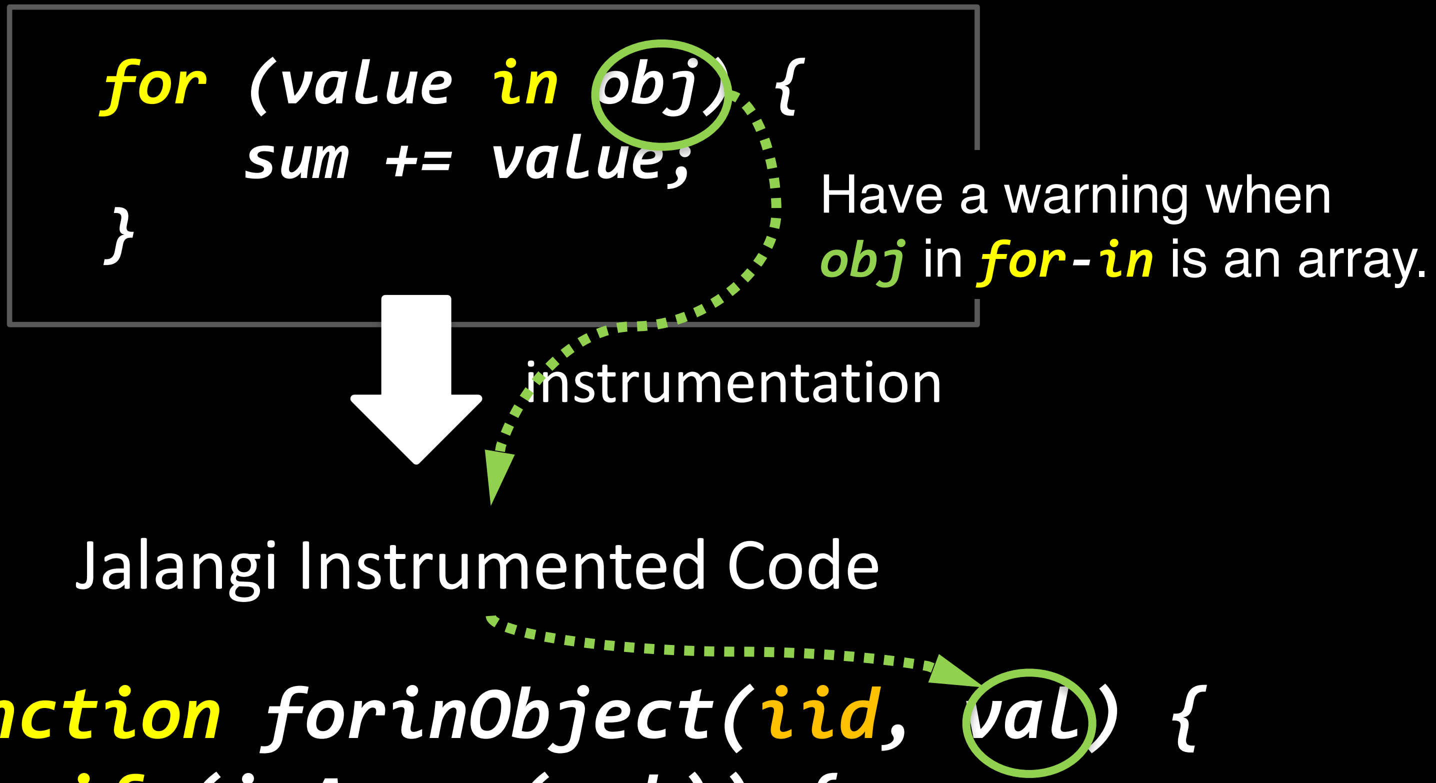
Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
    }
}
```

Detect *for..in* over arrays with Jalangi



```
for (value in obj) {
    sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
    }
}
```

Detect *for..in* over arrays with Jalangi

```
for (value in obj) {
      sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
        J$.iidToLocation(iid);
    }
}
```

Detect `for..in` over arrays with Jalangi

```
for (value in obj) {
        sum += value;
}
```

Have a warning when *obj* in ***for-in*** is an array.

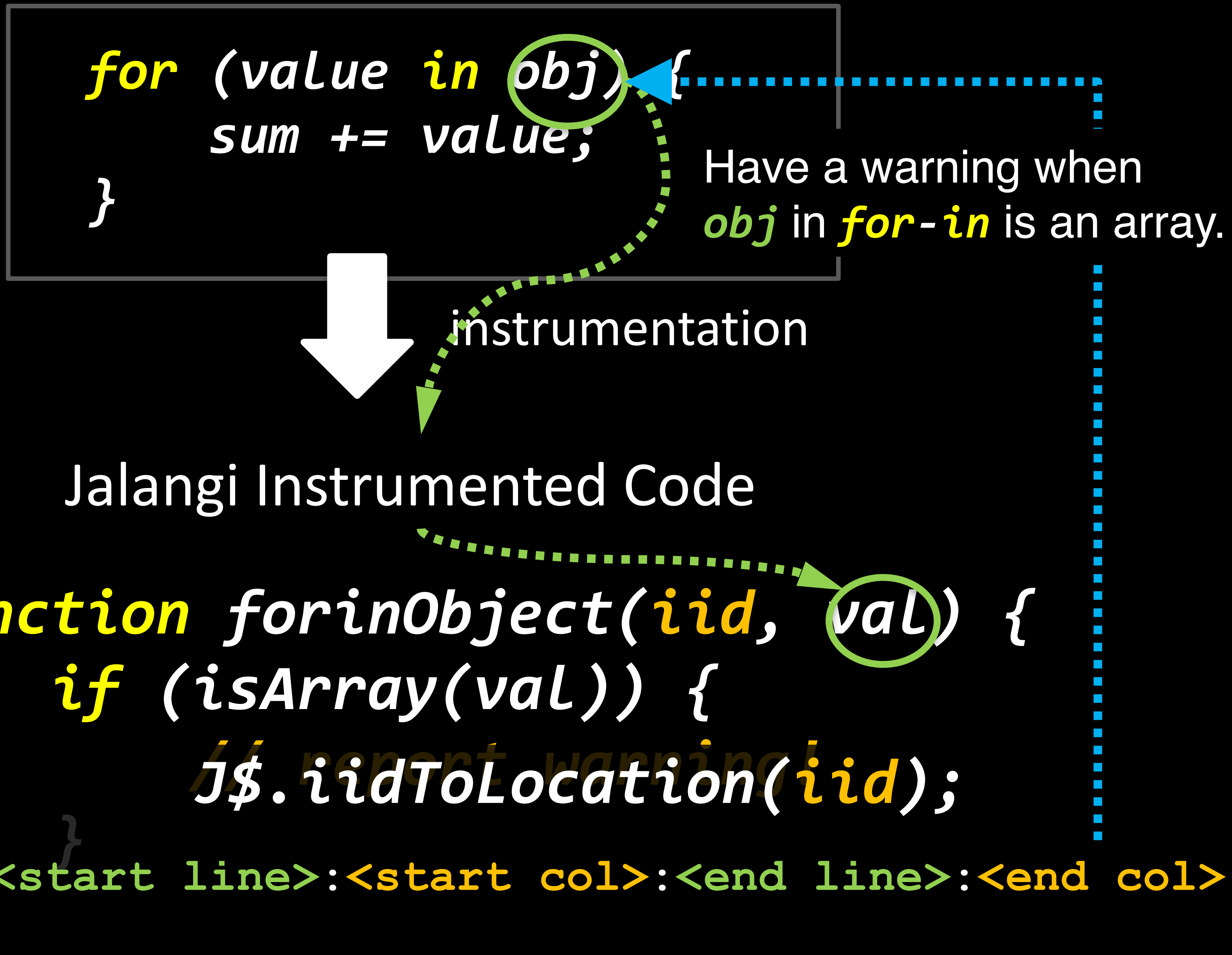
instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
        J$.iidToLocation(iid);
    }
}
```

`file.js:<start line>:<start col>:<end line>:<end col>`

Detect *for..in* over arrays with Jalangi



```
for (value in obj) {
      sum += value;
}
```

Have a warning when *obj* in *for-in* is an array.

instrumentation

Jalangi Instrumented Code

```
function forinObject(iid, val) {
    if (isArray(val)) {
        // report warning!
        J$.iidToLocation(iid);
    }
}
```

file.js:<start line>:<start col>:<end line>:<end col>

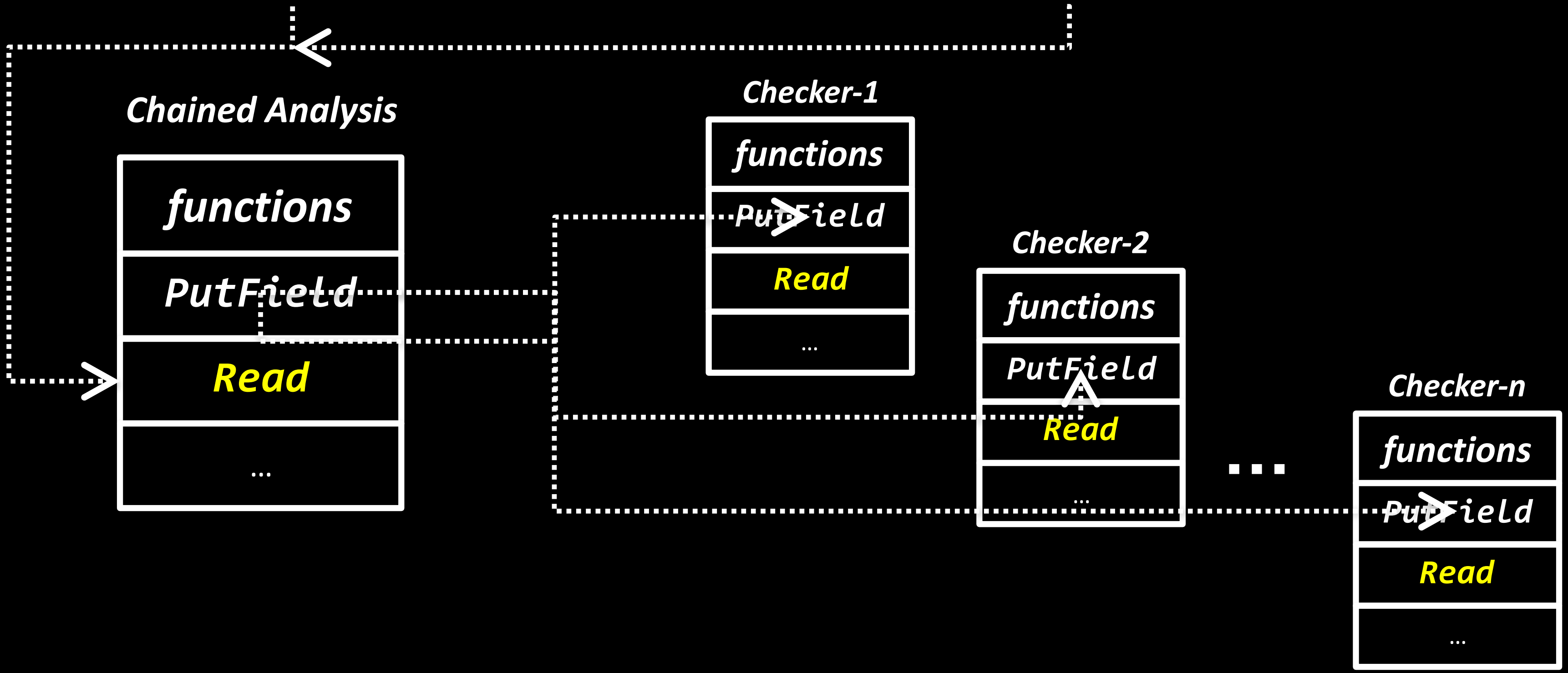
Checkers

CheckNaN.js
ConcatUndefinedToString.js
NonObjectPrototype.js
SetFieldToPrimitive.js
OverFlowUnderFlow.js
StyleMisuse.js
ToStringGivesNonString.js
UndefinedOffset.js
NoEffectOperation.js
AddEnumerablePropertyToObject.js
ConstructWrappedPrimitive.js
InconsistentNewCallPrefix.js
UncountableSpaceInRegexp.js
FloatNumberEqualityComparison.js
FunctionToString.js
ShadowProtoProperty.js
ForInArray.js
NonNumericArrayProperty.js
OverwrittenPrototype.js
GlobalThis.js
CompareFunctionWithPrimitives.js
InconsistentConstructor.js
FunctionCalledWithMoreArguments.js
IllegalUseOfArgumentsVariable.js
DoubleEvaluation.js
EmptyClassInRegexp.js
UseArrObjConstrWithoutArg.js
MissRadixArgInParseNum.js

Chained Analysis

```
a.f = b.g
```

```
PutField(Read("a", a), "f", GetField(Read("b", b), "g"))
```



Liang Gong, Electric Engineering & Computer Science, University of California, Berkeley.

Other Resources

Jalangi (v2) Github

https://github.com/Samsung/jalangi2

DLint + JITProf Github based on Jalangi (v2)

https://github.com/ksen007/jalangi2analyses

JITProf Visualization Github based on Jalangi (v2)

https://github.com/JacksonGL/jitprof-visualization

DLint and JITProf

DLint: Dynamically Checking JS Coding Practice

[ISSTA'15] DLint: Dynamically Checking Bad Coding Practices in JavaScri

Liang Gong, Michael Pradel, Manu Sridharan, Koushik Sen

JITProf: Find JS code that prohibit JIT-optimization

[FSE'15] JITProf: Pinpointing JIT-unfriendly JavaScript code

Liang Gong, Michael Pradel, Koushik Sen

Motivation of JITProf

Dynamic language features:

Simplifies coding

- Write less, do more
  → more productive
- Code is less verbose
  → easier to understand

Motivation of JITProf

Dynamic language features:

Simplifies coding
- Write less, do more
  → more productive
- Code is less verbose
  → easier to understand

Slow execution
- Too many runtime checks
- Object property lookup -> hash table lookup

  ...

Pinpointing JIT-unfriendly JavaScript Code

- Code snippet from Google Octane Benchmark:

```
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;
};
```

Pinpointing JIT-unfriendly JavaScript Code

- Code snippet from Google Octane Benchmark:

```
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;
};
```

Cause of poor performance:

- ***node*** has two layouts: offset of ***left*** in ***node*** can be ***0*** or ***1***
- JIT cannot replace ***node.left*** with ***node[0]*** or ***node[1]***

Pinpointing JIT-unfriendly JavaScript Code

- Code snippet from Google Octane Benchmark:

```
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%

Pinpointing JIT-unfriendly JavaScript Code

- Code snippet from Google Octane Benchmark:

```
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%

JITProf Simulates the Hidden Classes based on the information provided by Jalangi

Back to the 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;
}
```

- Each object has a meta information associated with it

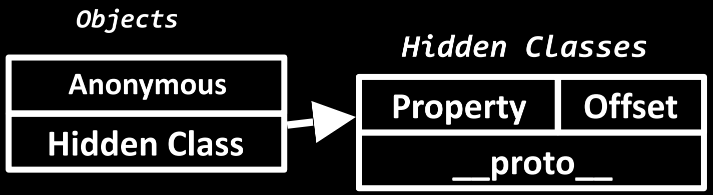
- The meta information keeps track of its object layout and its transition history.

Back to the 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;
}
```

Back to the Motivating Example

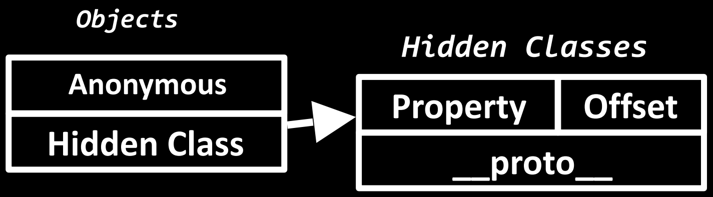
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

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



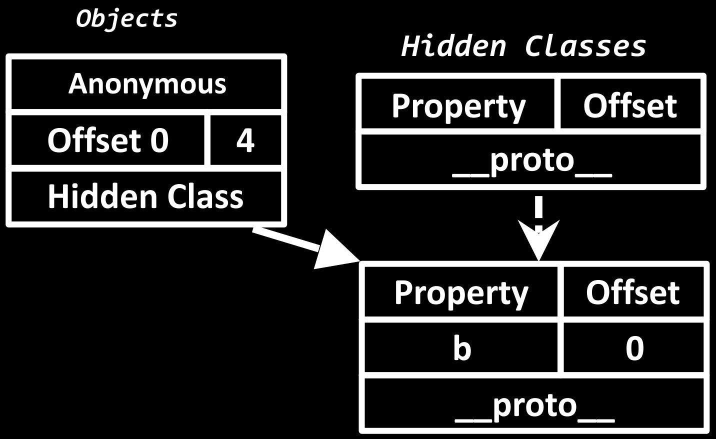
Hidden class simulation before the statement

Back to the 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;
}
```



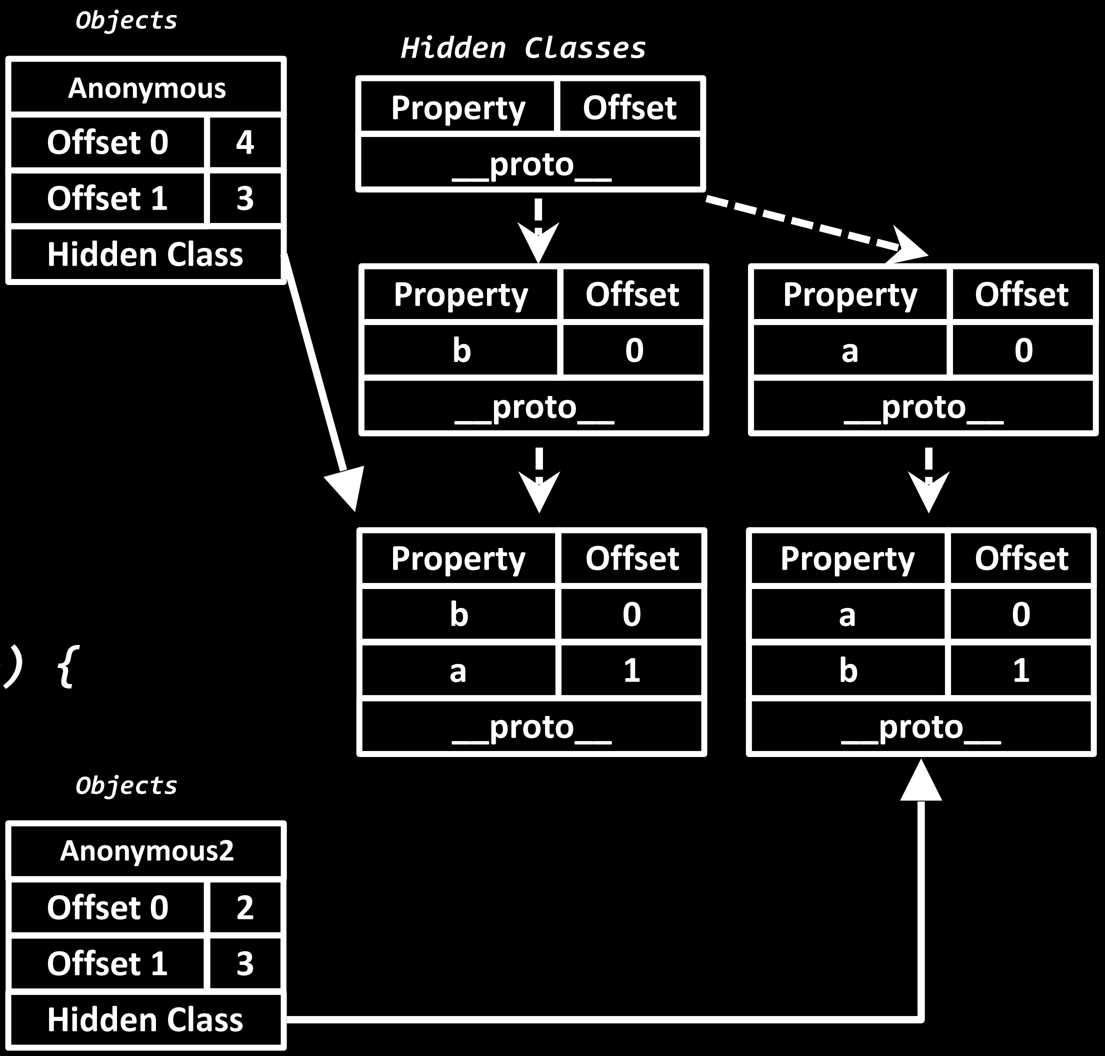
Hidden class simulation before the statement



Hidden class simulation after the statement

Back to the 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;
}
```



Objects

Anonymous	
Offset 0	4
Offset 1	3
Hidden Class	

Hidden Classes

Property	Offset
__proto__	

Property	Offset
b	0
__proto__	

Property	Offset
a	0
__proto__	

Property	Offset
b	0
a	1
__proto__	

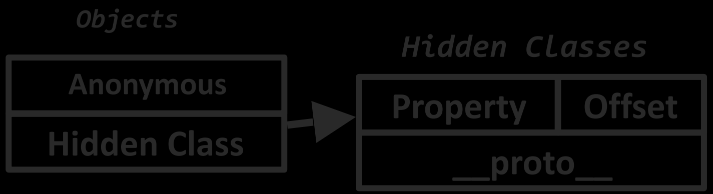
Property	Offset
a	0
b	1
__proto__	

Objects

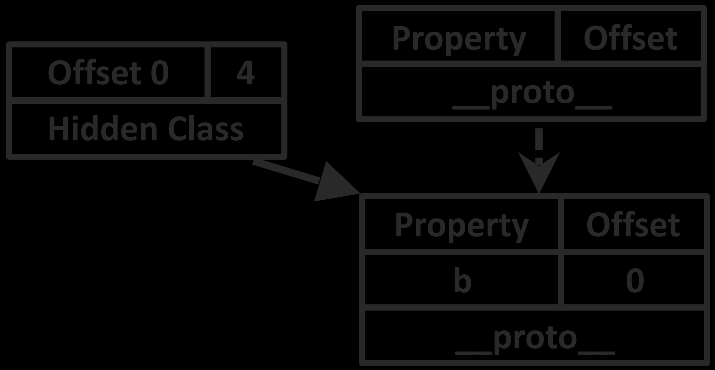
Anonymous2	
Offset 0	2
Offset 1	3
Hidden Class	

Back to the 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;
}
```

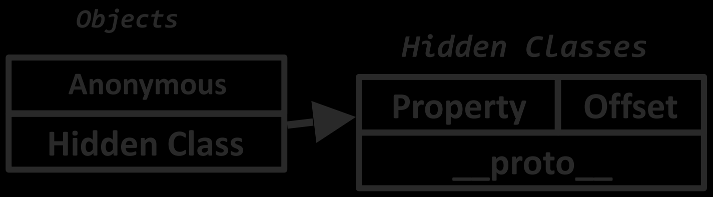


Hidden class simulation before the statement



Hidden class simulation after the statement

Back to the Motivating Example



Hidden class simulation before the statement

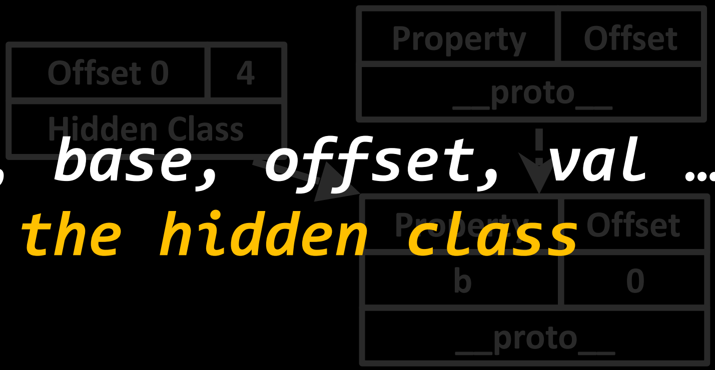
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}
```

Jalangi

invoke

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

```
function putFieldPre (iid, base, offset, val … ) {
    // logic for updating the hidden class

}
```



Hidden class simulation after the statement

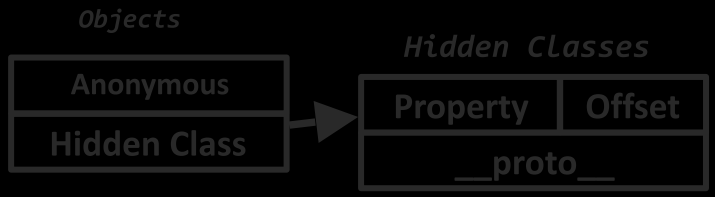
Back to the Motivating Example

```
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;
}
```

Jalangi

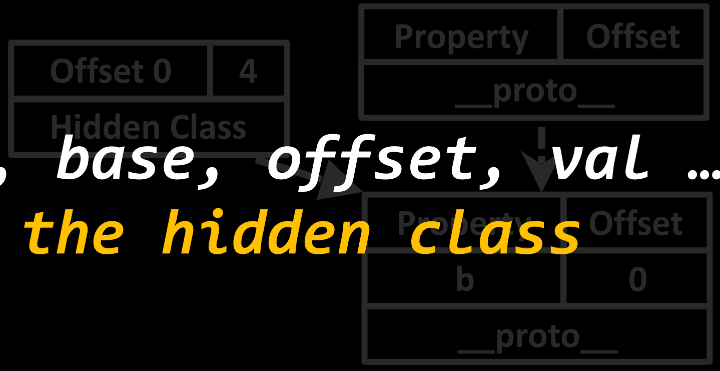
invoke

```
this.b = 4;
```



Hidden class simulation before the statement

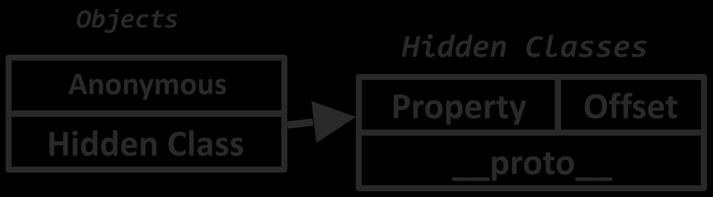
```
function putFieldPre (iid, base, offset, val … ) {
    // logic for updating the hidden class

}
```



Hidden class simulation after the statement

Back to the 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;
}
```



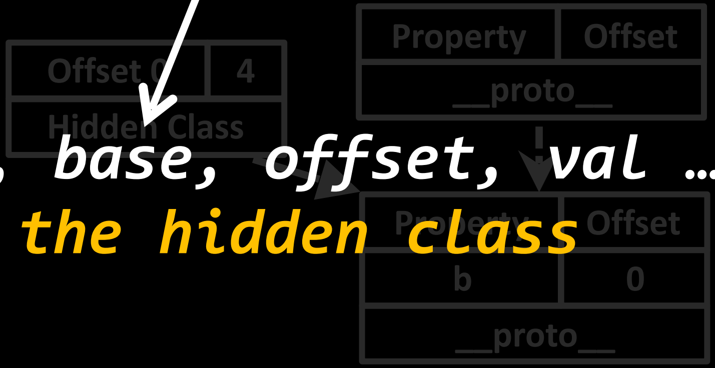
Hidden class simulation before the statement

Jalangi

invoke

```
this.b = 4;
```

```
function putFieldPre (iid, base, offset, val … ) {
    // logic for updating the hidden class

}
```



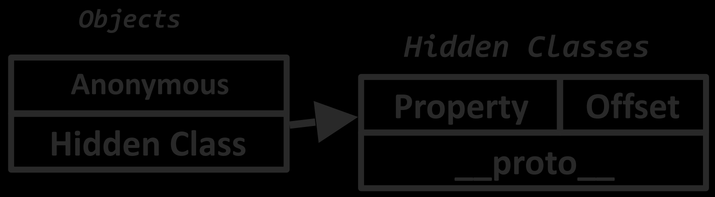
Hidden class simulation after the statement

Back to the Motivating Example

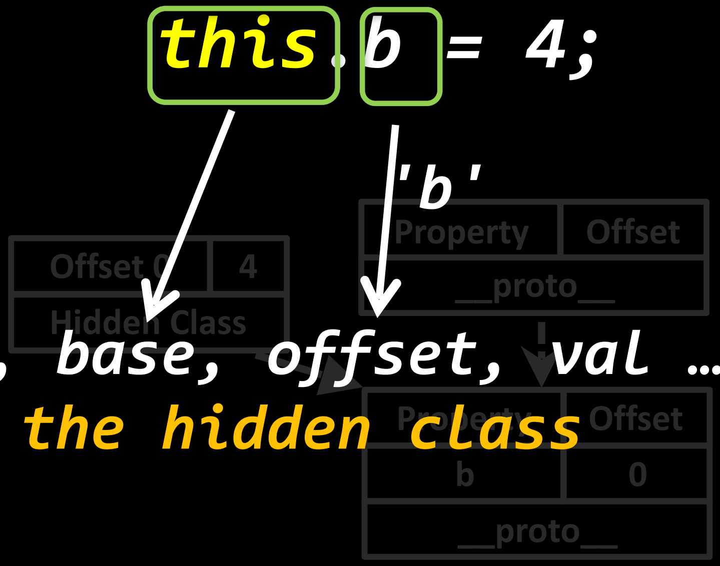
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

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

Jalangi

invoke



Hidden class simulation before the statement



```
function putFieldPre (iid, base, offset, val … ) {
    // logic for updating the hidden class

}
```

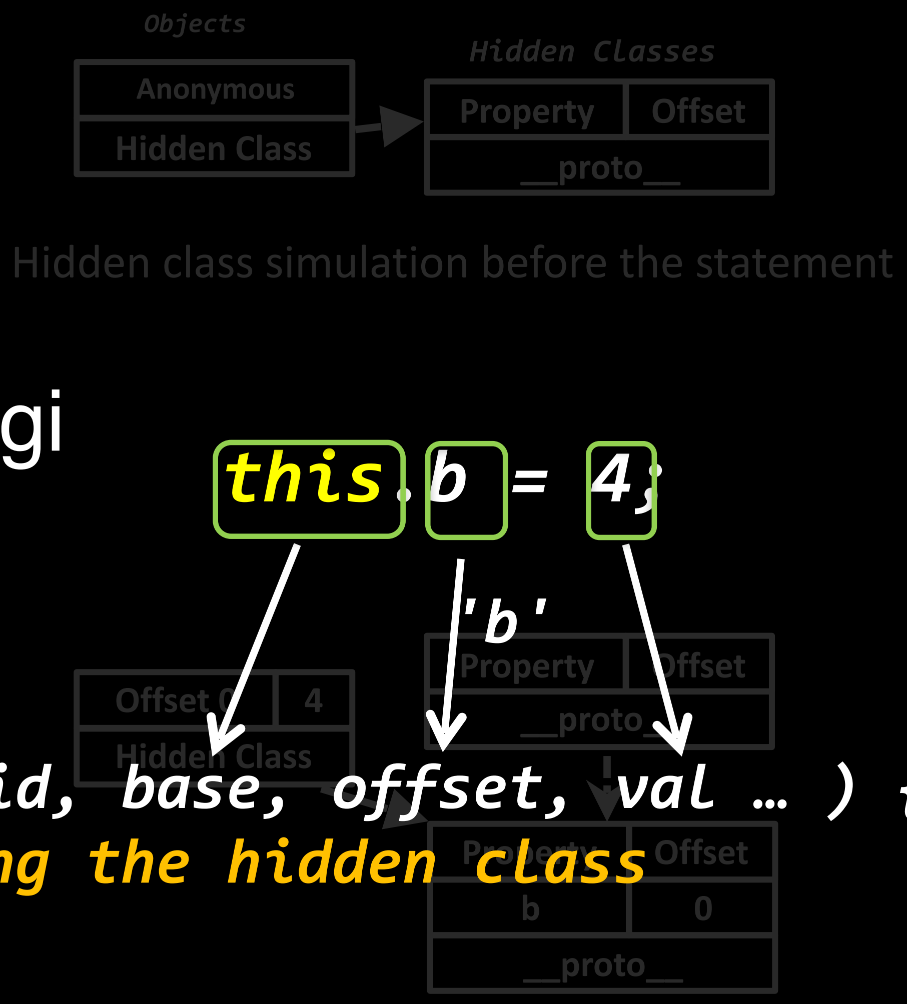
Hidden class simulation after the statement

Back to the 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;
}
```

Jalangi

invoke



Hidden class simulation before the statement

```
function putFieldPre (iid, base, offset, val … ) {
    // logic for updating the hidden class

}
```

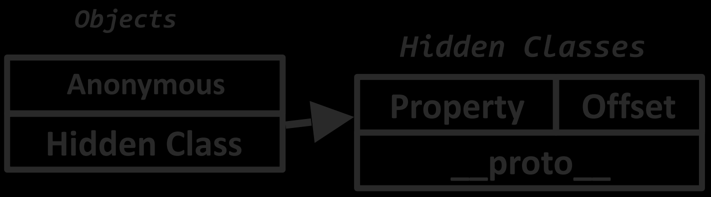
Hidden class simulation after the statement

Back to the Motivating Example

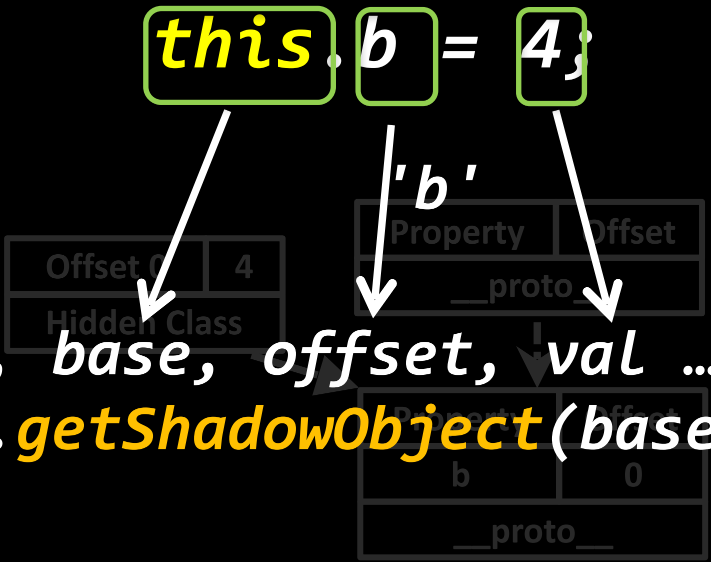
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

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

Jalangi

invoke



Hidden class simulation before the statement



```
function putFieldPre (iid, base, offset, val … ) {
    var sobj = J$.smemory.getShadowObject(base);
    sobj.hiddenClass ...
}
```

Hidden class simulation after the statement

Back to the 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;
}

var o = {a: 1, b: 2};
```

Intercept *putField* to update the hidden class

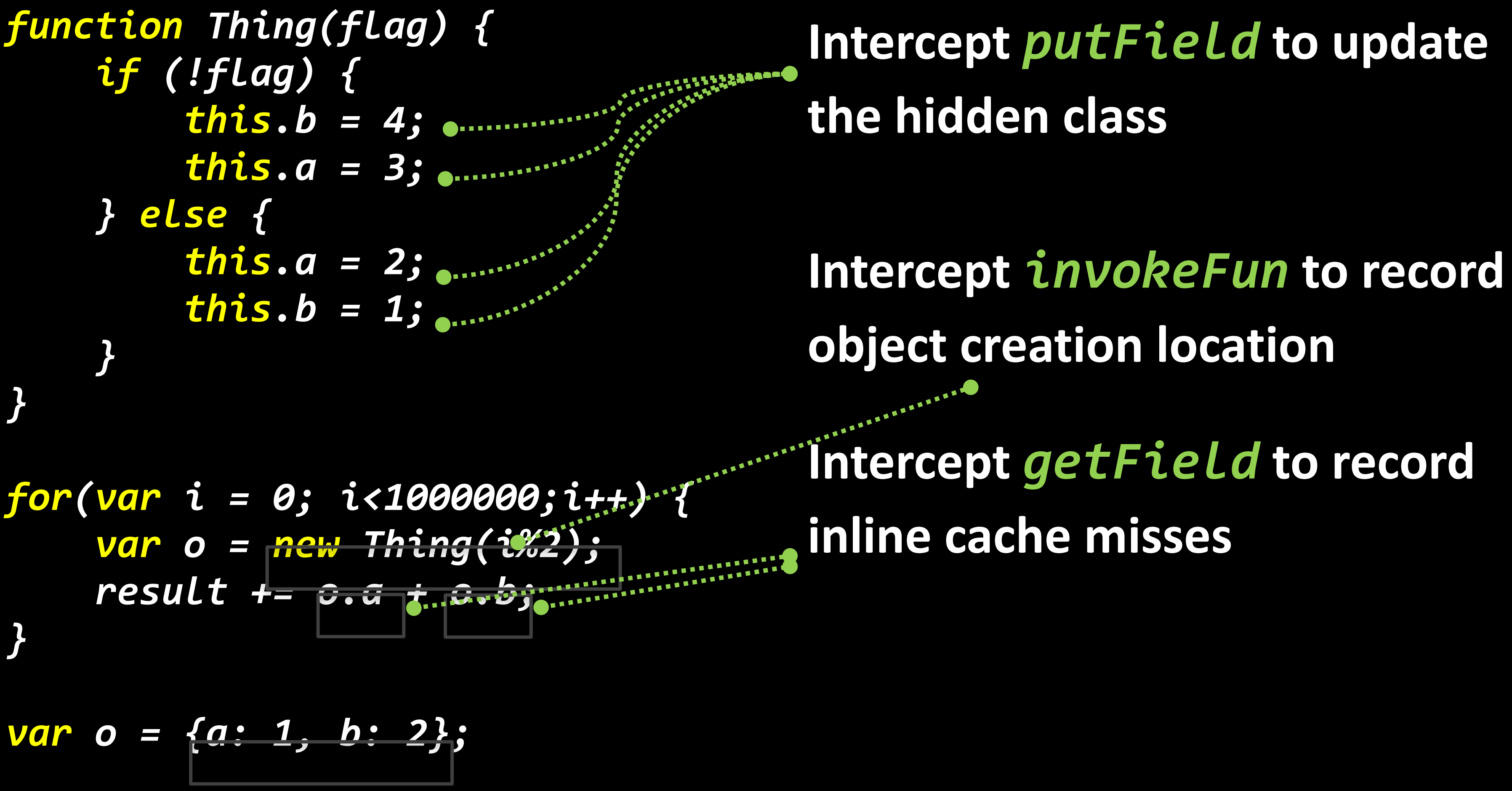
Back to the Motivating Example

```
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;
}

var o = {a: 1, b: 2};
```

Intercept *putField* to update the hidden class

Intercept *invokeFun* to record object creation location

Back to the Motivating Example



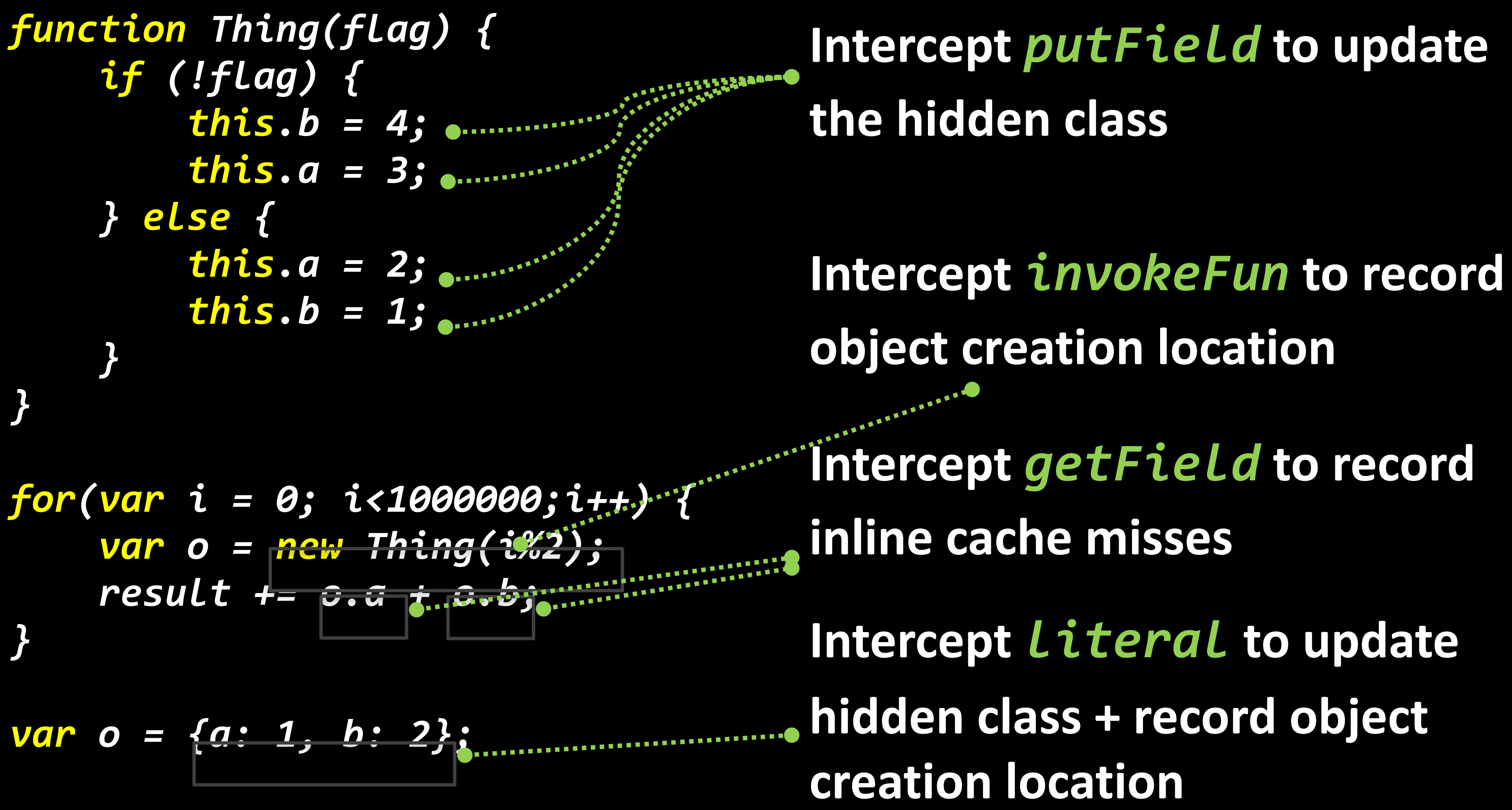
```
function Thing(flag) {
    if (!flag) {
        this.b = 4;
        this.a = 3;
    } else {
        this.a = 2;
        this.b = 1;
    }
}

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

var o = {a: 1, b: 2};
```

Intercept *putField* to update the hidden class

Intercept *invokeFun* to record object creation location

Intercept *getField* to record inline cache misses

Back to the 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;
}

var o = {a: 1, b: 2};
```

Intercept `putField` to update the hidden class

Intercept `invokeFun` to record object creation location

Intercept `getField` to record inline cache misses

Intercept `Literal` to update hidden class + record object creation location

JIT-unfriendly Code Checked by JITProf

- Use inconsistent object layout
- Access undeclared property or array element
- Store non-numeric value in numeric arrays
- Use in-contiguous keys for arrays
- Not all properties are initialized in constructors
- … and more

Rule #5: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

Rule #5: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

```
array[10000] = 10000;
array[9999] = 9999;
   ...
```

- non-contiguous array
- To save memory, JIT-engine decides to represent the array with slow data structures like hash table.

Rule #5: Use Contiguous Keys for Array

```
var array = [];
for (var i=10000;i>=0;i--){
    array[i] = i;
}
```

```
for (var i=0;i<=10000;i++){
    array[i] = i;
}
```

10X+ speedup!

Rule #5: Use Contiguous Keys for Array

```
      var array = [];
      for (var i=10000;i>=0;i--){
loc1: array[i] = i;
      }
```

- Intercept *putField* operation of arrays
- Rank locations by number assignments to non-contiguous arrays

(*)means smaller is better	group	average	improve rate
sunspider-chrome-sha1 (*)	original	1884.7588	26.3%
	refactored	1299.0706	
octane-firefox-Splay	original	11331.59	3.5%
	refactored	12198.65	
Sunspider-String-Tagcloud (*)	original	9178.76	11.7%
	refactored	9457.53	
octane-firefox-DeltaBlue	original	28473.53	1.4%
	refactored	31154.06	
octane-chrome-Box2D	original	24569.47	7.5%
	refactored	24915.00	
octane-chrome-RayTrace	original	43595.94	12.9%
	refactored	48140.35	

higher → better

(*)means smaller is better	group	average	improve rate
octane-chrome-Splay	**original**	10278.59	15.1%
	refactored	11885.71	
octane-chrome-SplayLatency	**original**	20910.24	3.8%
	refactored	21994.82	
sunspider-chrome-3d-Cube (*)	**original**	597.047059	1.1%
	refactored	593.744118	
sunspider-firefox-sha1 (*)	**original**	680.476471	3.3%
	refactored	669.932353	
sunspider-firefox-Xparb (*)	**original**	364.6824	19.7%
	refactored	357.2235	
sunspider-chrome-md5 (*)	**original**	774.3500	24.6%
	refactored	665.8382	
sunspider-chrome-format-tofte (*)	**original**	212.2029	3.4%
	refactored	200.9000	

higher → better

Install DLint and JITProf with Jalangi2

https://github.com/ksen007/jalangi2analyses

```
npm install
```

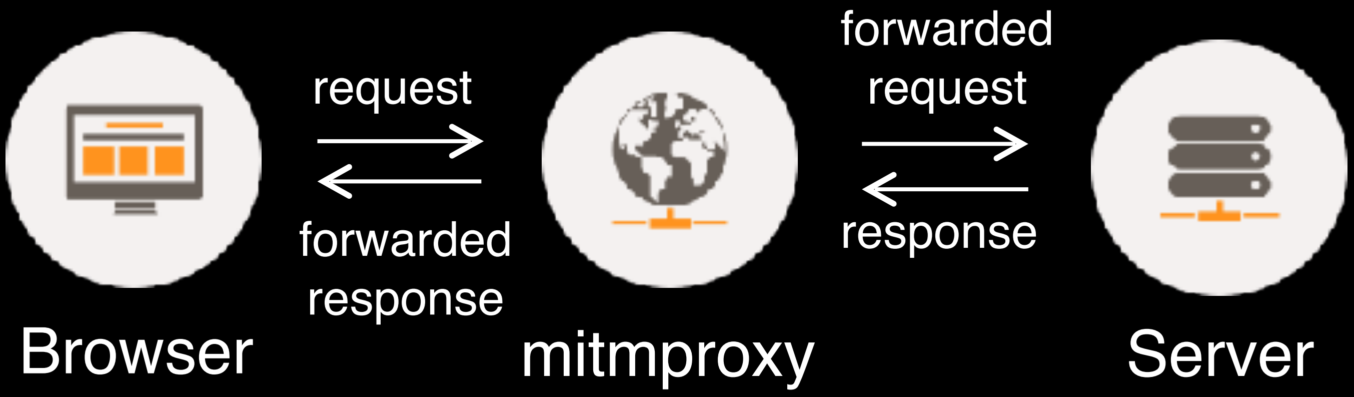
mitmproxy (third-party framework)

```
pip install pyOpenSSL
pip install mitmproxy==0.11.3
```

Install the mitmproxy certificate manually (drag-and-drop)

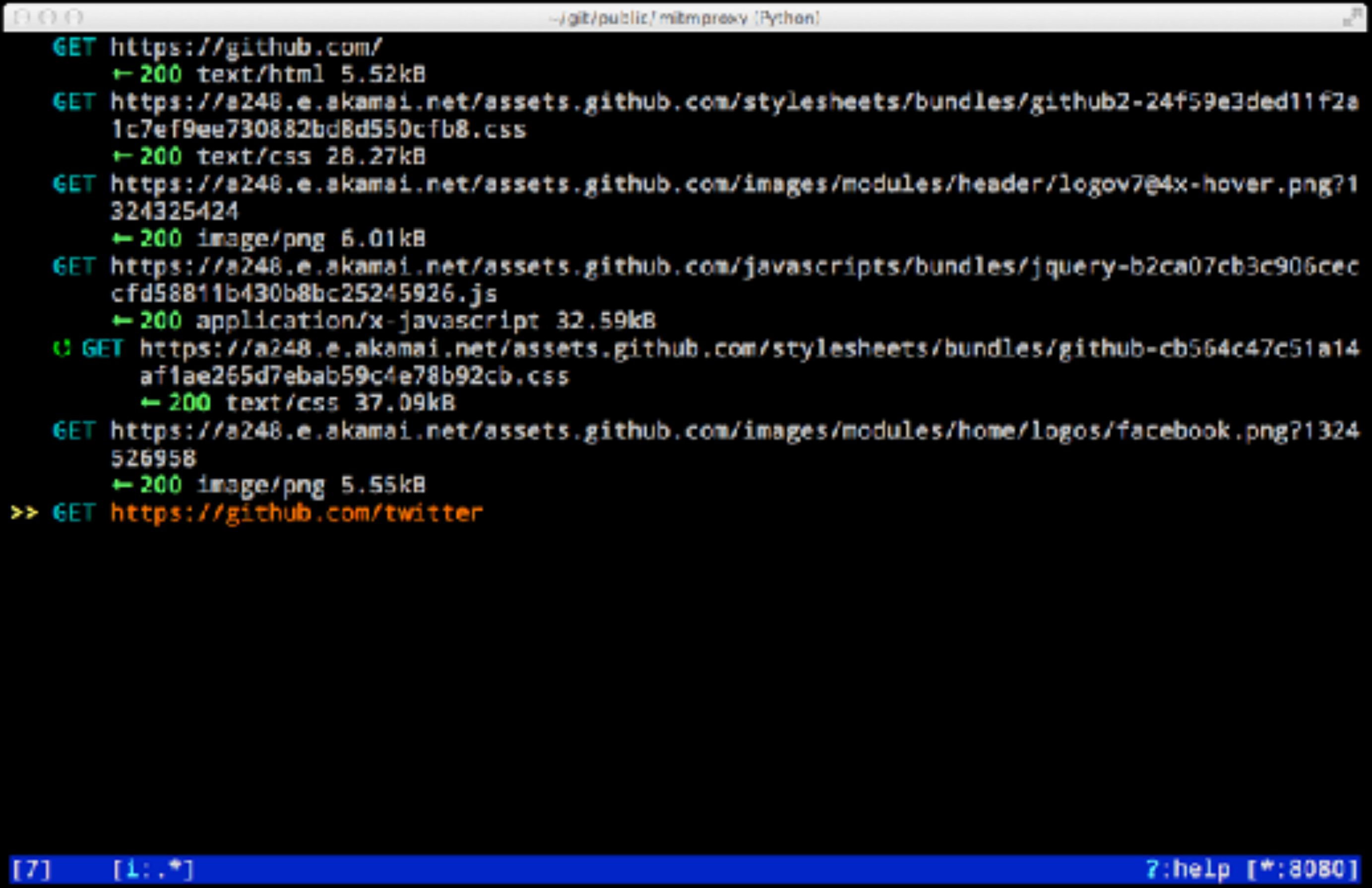
mitmproxy (third-party framework)

- man-in-the-middle proxy
- Interactive, SSL-capable proxy for HTTP with a console interface.
- Intercept http communication between the client and the server for instrumentation.



Install mitmproxy

- `pip install pyOpenSSL`
- `pip install mitmproxy==0.11.3`

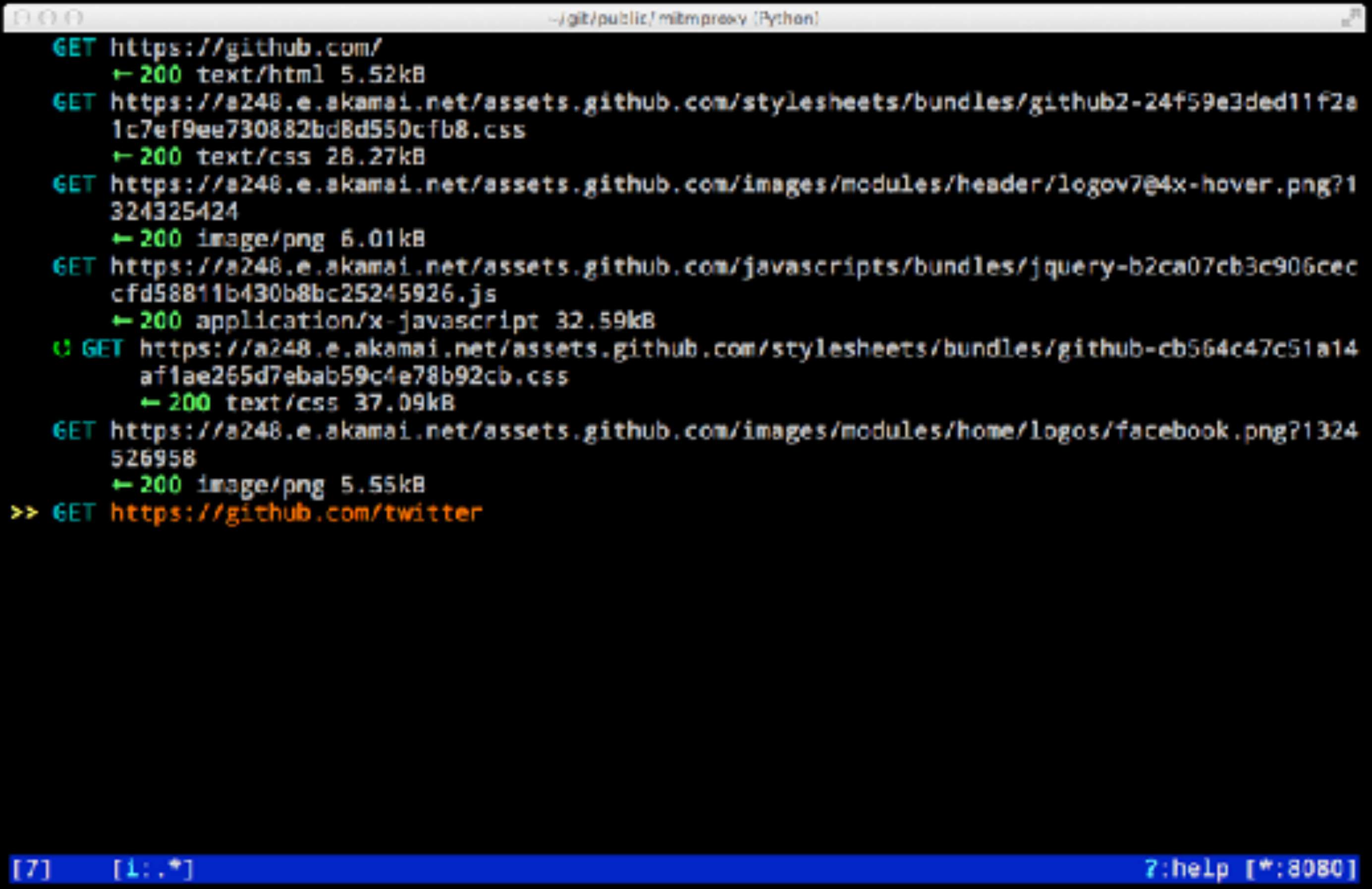


```
    GET https://github.com/
        ← 200 text/html 5.52kB
    GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github2-24f59e3ded11f2a
        1c7ef9ee730882bd8d550cfb8.css
        ← 200 text/css 28.27kB
    GET https://a248.e.akamai.net/assets.github.com/images/modules/header/logov7@4x-hover.png?1
        324325424
        ← 200 image/png 6.01kB
    GET https://a248.e.akamai.net/assets.github.com/javascripts/bundles/jquery-b2ca07cb3c906cec
        cfd58811b430b8bc25245926.js
        ← 200 application/x-javascript 32.59kB
    ↺ GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github-cb564c47c51a14
          af1ae265d7ebab59c4e78b92cb.css
          ← 200 text/css 37.09kB
    GET https://a248.e.akamai.net/assets.github.com/images/modules/home/logos/facebook.png?1324
        526958
        ← 200 image/png 5.55kB
>> GET https://github.com/twitter
```

```
[7]    [i:.*]                                                              ?:help [*:8080]
```

Install mitmproxy

- `pip install pyOpenSSL`
- `pip install mitmproxy==0.11.3`

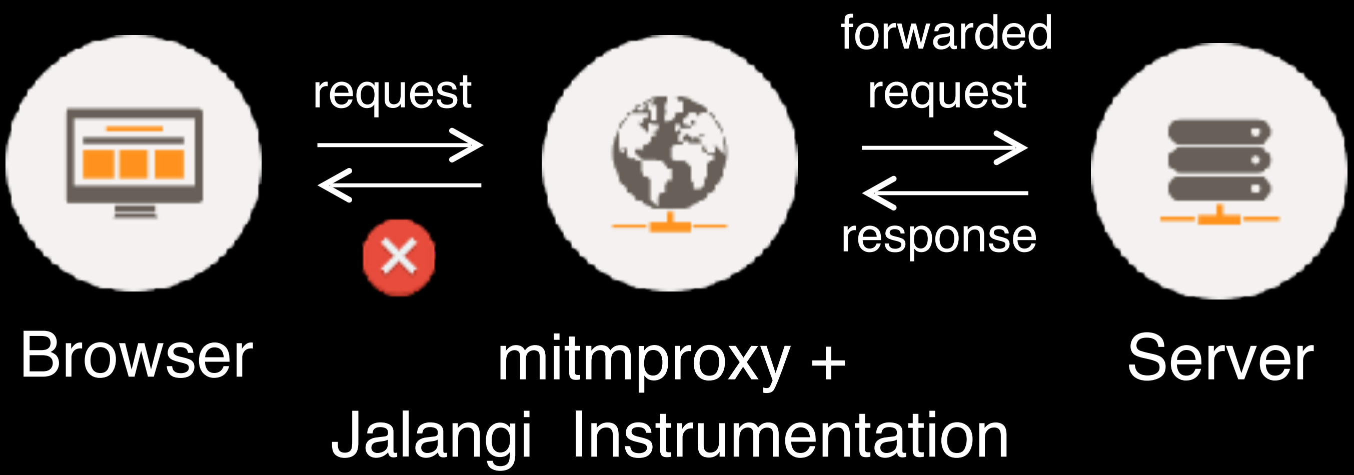


```
    GET https://github.com/
        ← 200 text/html 5.52kB
    GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github2-24f59e3ded11f2a
        1c7ef9ee730882bd8d550cfb8.css
        ← 200 text/css 28.27kB
    GET https://a248.e.akamai.net/assets.github.com/images/modules/header/logov7@4x-hover.png?1
        324325424
        ← 200 image/png 6.01kB
    GET https://a248.e.akamai.net/assets.github.com/javascripts/bundles/jquery-b2ca07cb3c906cec
        cfd58811b430b8bc25245926.js
        ← 200 application/x-javascript 32.59kB
    ↻ GET https://a248.e.akamai.net/assets.github.com/stylesheets/bundles/github-cb564c47c51a14
          af1ae265d7ebab59c4e78b92cb.css
          ← 200 text/css 37.09kB
    GET https://a248.e.akamai.net/assets.github.com/images/modules/home/logos/facebook.png?1324
        526958
        ← 200 image/png 5.55kB
>>  GET https://github.com/twitter
```

```
[7]     [i:.*]                                                              ?:help [*:8080]
```

The HTTPS Problem

- Man-in-the-middle Proxy
- SSL and HTTPS is designed against MITM
- HTTPS Handle shake error due to uncertified modification via instrumentation



request

forwarded request

response

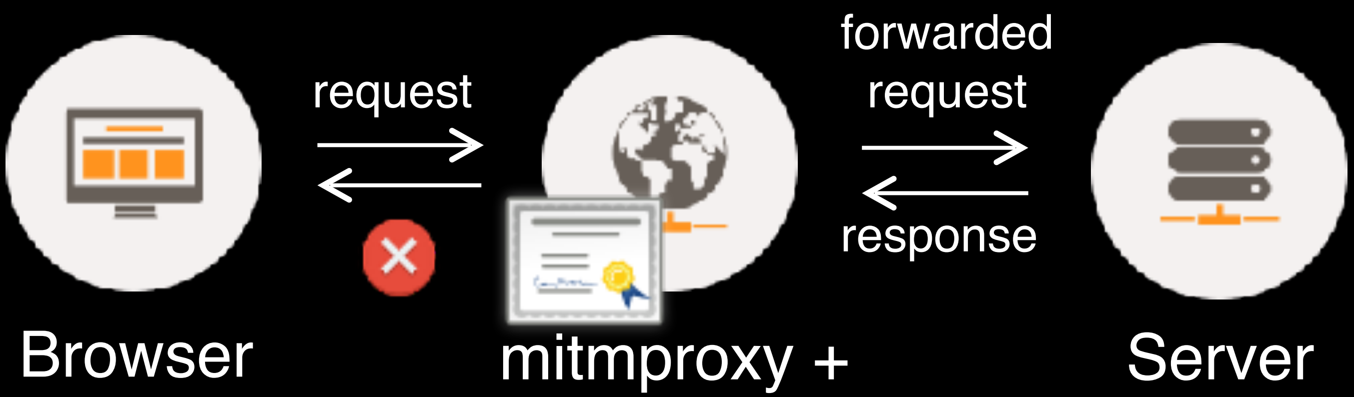
Browser

mitmproxy + Jalangi Instrumentation

Server

The HTTPS Problem

- Man-in-the-middle Proxy
- SSL and HTTPS is designed against MITM
- HTTPS Handle shake error due to uncertified modification via instrumentation

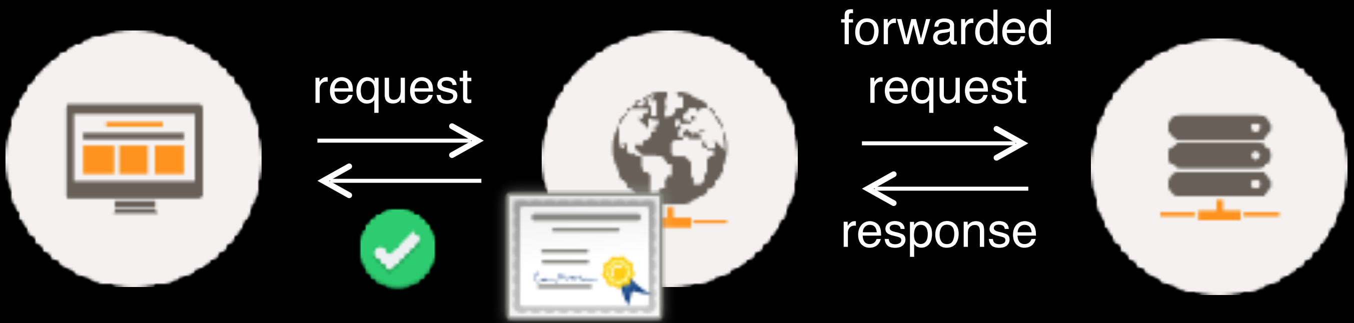


Jalangi Instrumentation

+ a Certificate Authority Implementation

The HTTPS Problem

- Man-in-the-middle Proxy
- SSL and HTTPS is designed against MITM
- HTTPS Handle shake error due to uncertified modification via instrumentation



request

forwarded request

response

Browser

mitmproxy + Jalangi Instrumentation + a Certificate Authority Implementation

Server

Other Resources

Jalangi (v2) Github

https://github.com/Samsung/jalangi2

DLint + JITProf Github based on Jalangi (v2)

https://github.com/ksen007/jalangi2analyses

JITProf Visualization Github based on Jalangi (v2)

https://github.com/JacksonGL/jitprof-visualization

Questions ?

Thank
You
köszönöm
Kiitos
Asante
Shukria
Vinaka
Maake
Dhanyavadagalu
감사합니다
Kam Sah Hammida
شكرا
Manana
Dankon
Dankscheen
спасибо
Mauruuru
Biyan
Diolch i Chi
Terima Kasih
Taiku
Matondo
Dank Je
Tack
Blagodaram
Chokrane
Ngiyabonga
Dziekuje
Grazie
Juspaxar
Arigato
Mochchakkeram
Gracias
Paldies
Tingki
நன்றி
Bedankt
D'akujem
धन्यवाद
cảm ơn bạn
Kop Khun Khap
Gratias Tibi
Ua Tsaug Rau Koj
Grazas
Di Ou Mèsi
Kia Ora
Obrigado
Děkuji
Nirringrazzjak
ありがとう
Suksama
Matur Nuwun
Rahmat
Hvala
谢谢
хвала
Welalin
Djiere Dieuf
Misaotra
Danke
Merci
Go Raibh Maith Agat
Eskerrik Asko
Salamat
ขอขอบคุณคุณ
Najis Tuke