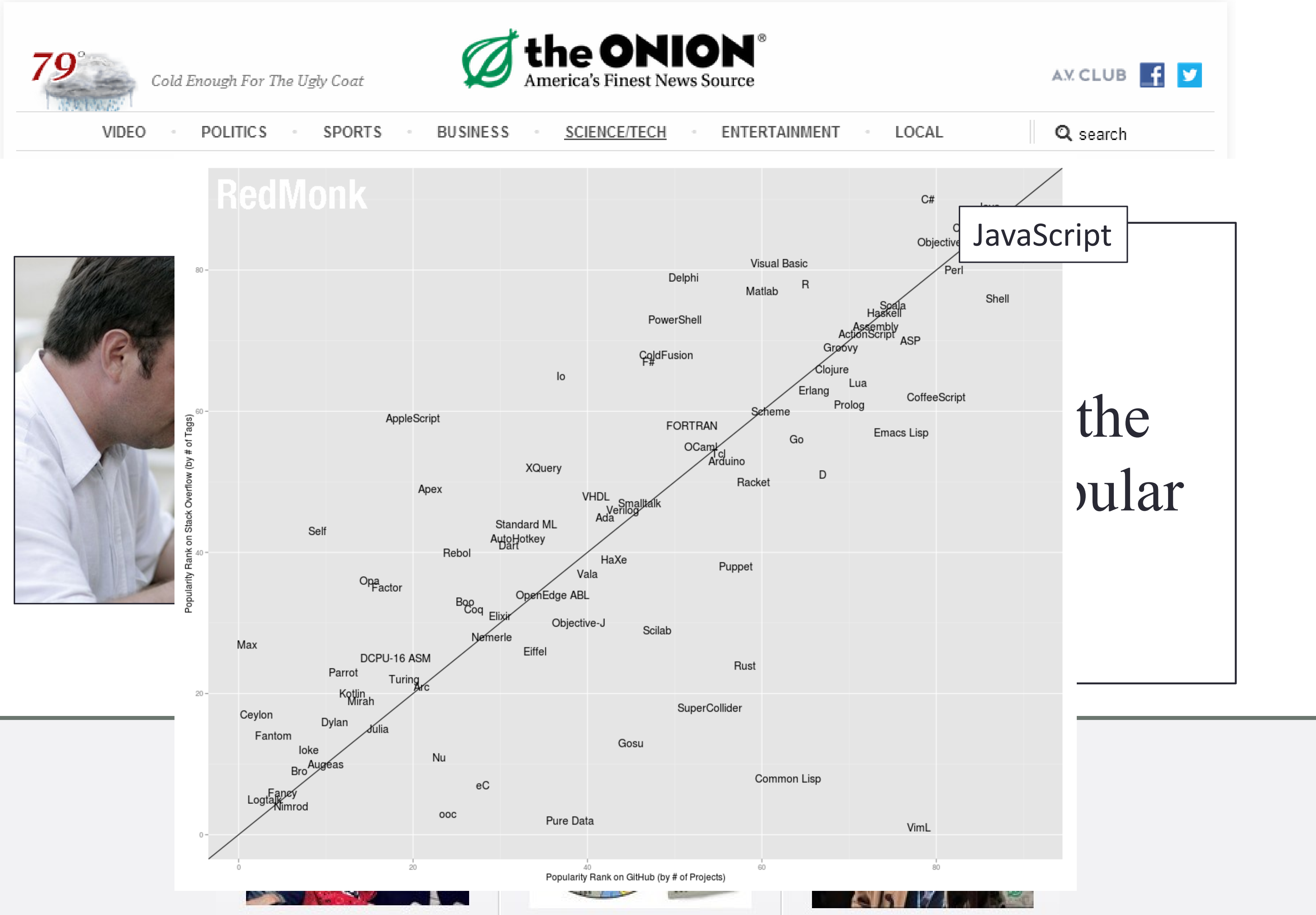
CHALLENGES IN POINTER ANALYSIS OF JAVASCRIPT

Ben Livshits

MSR


79°
Cold Enough For The Ugly Coat
the ONION
America's Finest News Source
A.V. CLUB
VIDEO
POLITICS
SPORTS
BUSINESS
SCIENCE/TECH
ENTERTAINMENT
LOCAL
search
RedMonk
JavaScript
the
ular
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
20
40
60
80
C#
Objective
Perl
Shell
Visual Basic
Delphi
R
Matlab
Scala
Haskell
PowerShell
Assembly
ActionScript
ASP
Groovy
ColdFusion
F#
Io
Clojure
Lua
Erlang
CoffeeScript
Prolog
Scheme
AppleScript
FORTRAN
Emacs Lisp
Go
OCaml
Tcl
Arduino
XQuery
D
Racket
Apex
VHDL
Smalltalk
Verilog
Ada
Standard ML
Self
AutoHotkey
Dart
Rebol
HaXe
Puppet
Vala
Opa
Factor
OpenEdge ABL
Boo
Coq
Elixir
Objective-J
Scilab
Nemerle
Max
Eiffel
DCPU-16 ASM
Rust
Parrot
Turing
Arc
Kotlin
Mirah
SuperCollider
Ceylon
Dylan
Julia
Fantom
Gosu
Ioke
Nu
Bro
Augeas
Common Lisp
eC
Fancy
Logtalk
Nimrod
ooc
Pure Data
VimL

Two Issues in JavaScript Pointer Analysis

1

Gulfstream

- JavaScript programs on the web are streaming
- Fully static analysis pointer analysis is not possible, calling for a hybrid approach
- Setting: analyzing pages before they reach the browser

2

Use analysis

- JavaScript programs interop with a set of reach APIs such as the DOM
- We need to understand these APIs for analysis to be useful
- Setting: analyzing Win8 apps written in JavaScript

Gulfstream

- *Staged Static Analysis for Streaming JavaScript Applications*, Salvatore Guarnieri, Ben Livshits, WebApps 2009

GULFSTREAM: Staged Static Analysis for Streaming JavaScript Applications

Salvatore Guarnieri
University of Washington

Benjamin Livshits
Microsoft Research

Abstract

The advent of Web 2.0 has led to the proliferation of client-side code that is typically written in JavaScript. Recently, there has been an upsurge of interest in static analysis of client-side JavaScript for applications such as bug finding and optimization. However, most approaches in static analysis literature assume that the *entire program* is available to analysis. This, however, is in direct contradiction with the nature of Web 2.0 programs that are essentially being streamed at the user's browser. Users can see data being streamed to pages in the form of page updates, but the same thing can be done with code, essentially delaying the downloading of code until it is needed. In essence, the entire program is never completely available. Interacting with the application causes more code to be sent to the browser.

This paper explores *staged static analysis* as a way to analyze streaming JavaScript programs. We observe while there is variance in terms of the code that gets sent to the client, much of the code of a typical JavaScript application can be determined statically. As a result, we advocate the use of combined offline-online static analysis as a way to accomplish fast, browser-based client-side online analysis at the expense of a more thorough and costly server-based offline analysis on the static code. We find that in normal use, where updates to the code are small, we can update static analysis results quickly enough in the browser to be acceptable for everyday use. We demonstrate the staged analysis approach to be advantageous especially in mobile devices, by experimenting on popular applications such as Facebook.

1 Introduction

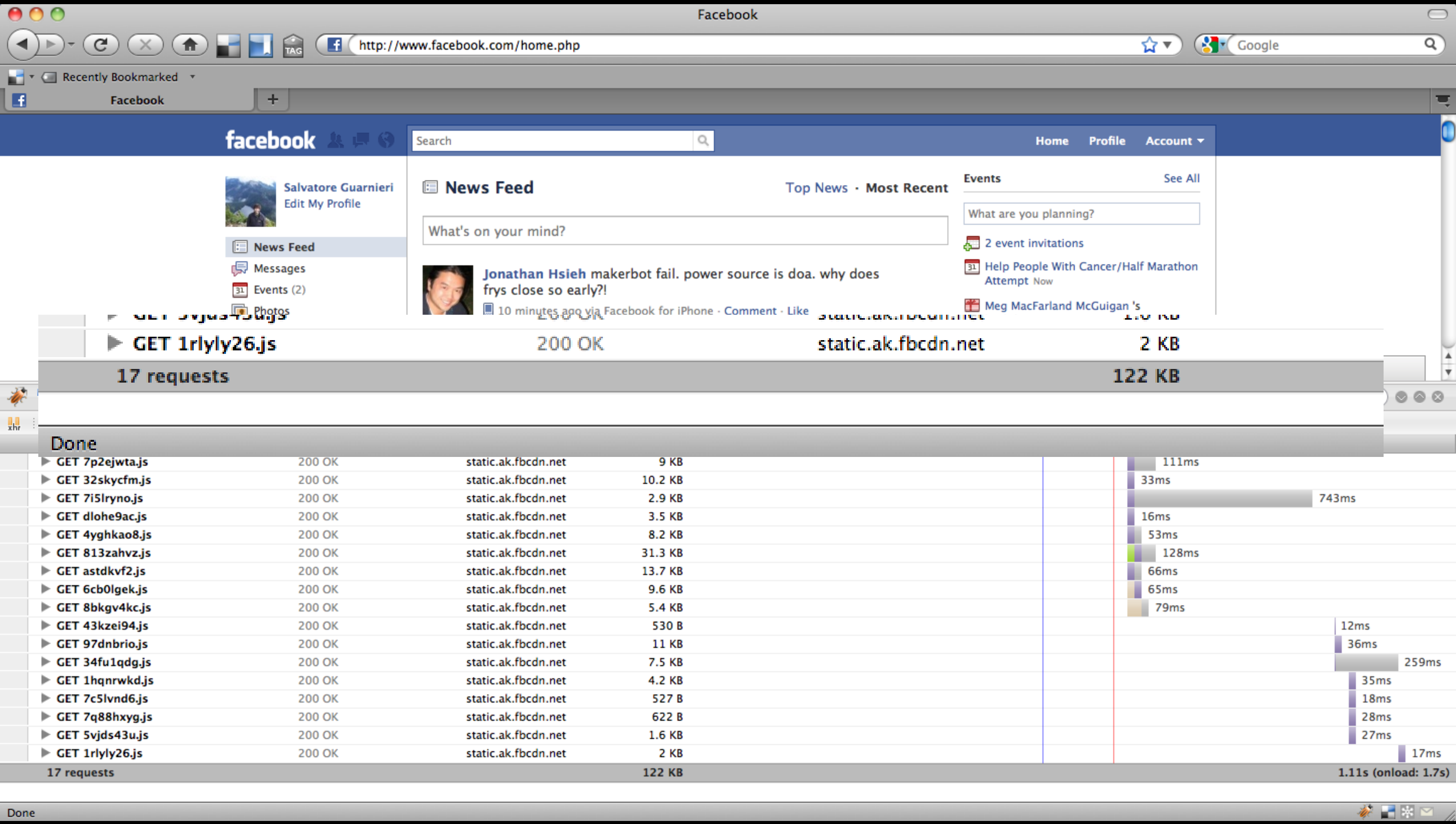
The advent of Web 2.0 has led to the proliferation of client-side code that is typically written in JavaScript. This code is often combined or *mashed-up* with other code and content from different third-party servers, making the application only fully available within the user's browser. Recently, there has been an upsurge of interest in static analysis of client-side JavaScript. However, most approaches in the static analysis literature assume that the entire program is available for analysis. This, however, is in direct contradiction with the nature of Web 2.0 programs that are essentially being *streamed* to the user's browser. In essence, the JavaScript application is never available in its entirety: as the user interacts with the application, more code is sent to the browser.

A pattern that emerged in our experiments with static analysis to enforce security properties [14], is that while most of the application can be analyzed offline, some parts of it will need to be analyzed on-demand, in the browser. In one of our experiments, while 157 KB (71%) of Facebook JavaScript code is downloaded right away, an additional 62 KB of code is downloaded when visiting event pages, etc. Similarly, Bing Maps downloads most of the code right away; however, requesting traffic requires additional code downloads. Moreover, often the parts of the application that are downloaded later are composed on the client by referencing a third-party library at a fixed CDN URL; common libraries are jQuery and `prototype.js`. Since these libraries change relatively frequently, analyzing this code ahead of time may be inefficient or even impossible.

The dynamic nature of JavaScript, combined with the incremental nature of code downloading in the browser leads to some unique challenges. For instance, consider the piece of HTML in Figure 1. Suppose we want to statically determine what code may be called from the `onclick` handler to ensure that none of the invoked functions may block. If we only consider the first `SCRIPT` block, we will conclude that the `onclick` handler may only call function `foo`. Including the second `SCRIPT` block adds function `bar` as a possible function that may be called. Furthermore, if the browser proceeds to download more code, either through more `SCRIPT` blocks or `XmlHttpRequests`, more code might need to be consid-

Whole program analysis?

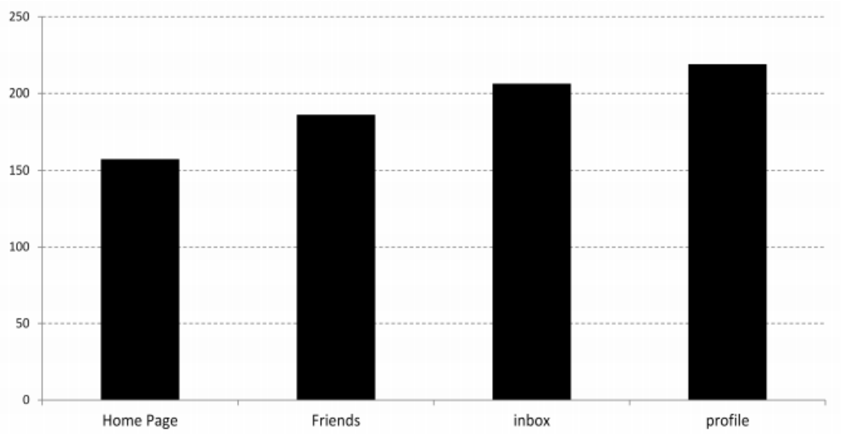
What whole program?


Facebook
http://www.facebook.com/home.php
Google
Recently Bookmarked
Facebook
facebook
Search
Home
Profile
Account
Salvatore Guarnieri
Edit My Profile
News Feed
Messages
Events (2)
Photos
News Feed
Top News · Most Recent
What's on your mind?
Jonathan Hsieh makerbot fail. power source is doa. why does frys close so early?!
10 minutes ago via Facebook for iPhone · Comment · Like
Events
See All
What are you planning?
2 event invitations
Help People With Cancer/Half Marathon Attempt Now
Meg MacFarland McGuigan 's
GET 1rlyly26.js 200 OK static.ak.fbcdn.net 2 KB
17 requests 122 KB
Done
GET 7p2ejwta.js 200 OK static.ak.fbcdn.net 9 KB 111ms
GET 32skycfm.js 200 OK static.ak.fbcdn.net 10.2 KB 33ms
GET 7i5lryno.js 200 OK static.ak.fbcdn.net 2.9 KB 743ms
GET dlohe9ac.js 200 OK static.ak.fbcdn.net 3.5 KB 16ms
GET 4yghkao8.js 200 OK static.ak.fbcdn.net 8.2 KB 53ms
GET 813zahvz.js 200 OK static.ak.fbcdn.net 31.3 KB 128ms
GET astdkvf2.js 200 OK static.ak.fbcdn.net 13.7 KB 66ms
GET 6cb0lgek.js 200 OK static.ak.fbcdn.net 9.6 KB 65ms
GET 8bkgv4kc.js 200 OK static.ak.fbcdn.net 5.4 KB 79ms
GET 43kzei94.js 200 OK static.ak.fbcdn.net 530 B 12ms
GET 97dnbrio.js 200 OK static.ak.fbcdn.net 11 KB 36ms
GET 34fu1qdg.js 200 OK static.ak.fbcdn.net 7.5 KB 259ms
GET 1hqnrwkd.js 200 OK static.ak.fbcdn.net 4.2 KB 35ms
GET 7c5lvnd6.js 200 OK static.ak.fbcdn.net 527 B 18ms
GET 7q88hxyg.js 200 OK static.ak.fbcdn.net 622 B 28ms
GET 5vjds43u.js 200 OK static.ak.fbcdn.net 1.6 KB 27ms
GET 1rlyly26.js 200 OK static.ak.fbcdn.net 2 KB 17ms
17 requests 122 KB 1.11s (onload: 1.7s)
Done

JavaScript programs are streaming

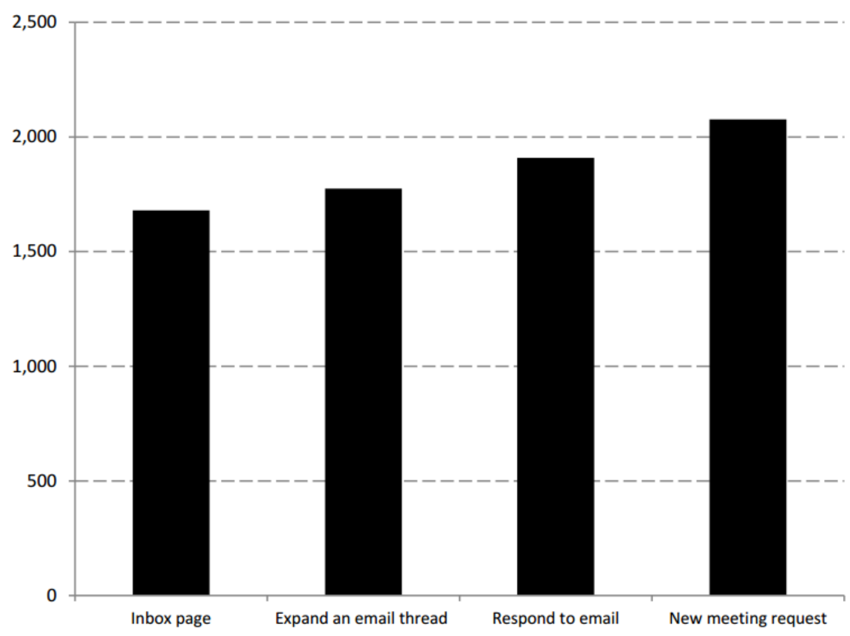
Facebook Code Exploration

Page visited or action performed	Added JavaScript files	KB
FACEBOOK FRONT PAGE		
Home page	19	157
Friends	7	186
Inbox	1	206
Profile	1	219



OWA Code Exploration

Outlook Web Access (OWA)		
Inbox page	7	1,680
Expand an email thread	1	95
Respond to email	2	134
New meeting request	2	168



Script Creation

```
<HTML>
  <HEAD>
    <SCRIPT>
      function foo(){...}
      var f = foo;
    </SCRIPT>
    <SCRIPT>
      function bar(){...}
      if (...) f = bar;
    </SCRIPT>
   </HEAD>
  <BODY onclick="f();"> ...</BODY>
</HTML>
```

What does f refer to?

Plan

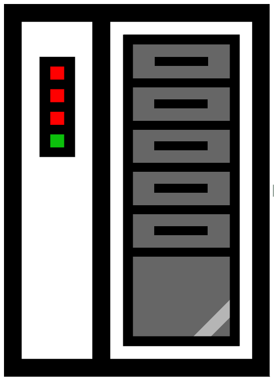
Server

- Pre-compute pointer information offline, for most of the program
- Optionally update server knowledge as more code is observed

Client

- When more code is discovered, do analysis of it
- Combine the incremental results with pre-computed results

Gulfstream In Action



✔ ✔

Is it faster to

1) transfer pre-computed results + add incremental results
2) Compute everything from scratch

Checking a safety property

Simulated Devices

ID	Configuration Name	CPU coef. c	Link type	La L i
1	G1	67.0	EDGE	
2	Palm Pre	36.0	Slow 3G	
3	iPhone 3G	36.0	Fast 3G	
4	iPhone 3GS 3G	15.0	Slow 3G	
5	iPhone 3GS WiFi	15.0	Fast WiFi	
6	MacBook Pro 3G	1	Slow 3G	
7	MacBook Pro WiFi	1	Slow WiFi	
8	Netbook	2.0	Fast 3G	
9	Desktop WiFi	0.8	Slow WiFi	
10	Desktop T1	0.8	T1	

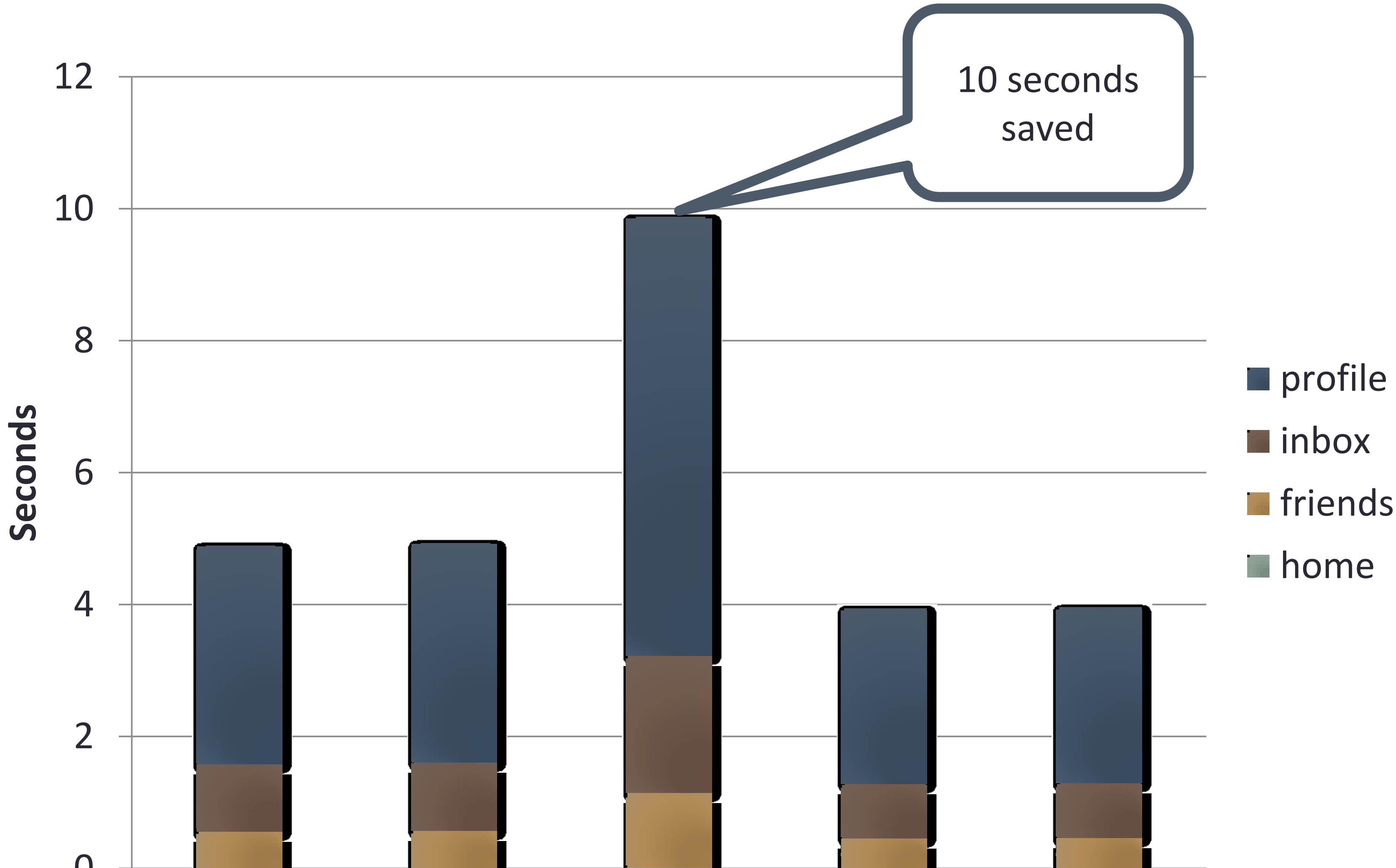
Try Different Configurations

Graph Graph	Incremental Size	1	2	3	4	5	6	7	8	9	10
		Settings									
6,914	88	+	+	+	+	+	-	+	+	-	+
7,608	619	+	+	+	+	+	-	-	+	-	+
8,332	1,138	+	+	+	+	+	-	-	+	-	+
11,045	1,644	+	+	+	+	+	-	-	-	-	+
9,400	2,186	+	+	+	+	+	-	-	+	-	+
10,058	2,767	+	+	+	+	+	-	-	-	-	+
12,846	3,293	+	+	+	+	+	-	-	-	-	+
11,269	3,846	+	+	+	+	+	-	-	-	-	+
12,494	4,406	+	+	+	+	+	-	-	-	-	+
12,578	5,008	+	+	+	+	+	-	-	-	-	+
9,526	5,559	+	+	+	+	+	-	-	-	-	+
13,788	6,087	+	+	+	+	+	-	-	-	-	+
14,447	6,668	+	+	+	+	+	-	-	-	-	+
15,095	7,249	+	+	+	+	+	-	-	-	-	+
15,751	7,830	+	+	+	+	+	-	-	-	-	+
16,306	8,333	+	+	+	+	+	-	-	-	-	+
16,866	8,861	+	+	+	+	+	-	-	-	-	+
17,413	9,389	+	+	+	+	+	-	-	-	-	+
17,969	9,917	+	+	+	+	+	-	-	-	-	+
18,520	10,445	+	+	+	-	+	-	-	-	-	+
19,075	10,973	+	+	+	-	+	-	-	-	-	+
19,633	11,501	+	+	+	-	+	-	-	-	-	+
20,184	12,029	+	+	+	-	+	-	-	-	-	+
20,750	12,557	-	-	+	-	+	-	-	-	-	+
34,570	14,816	+									
27,699	16,485	-									
35,941	17,103	-									
38,054	17,909	-									
27,296	20,197	-									
35,945	25,566	-									
17,108	31,465	-									

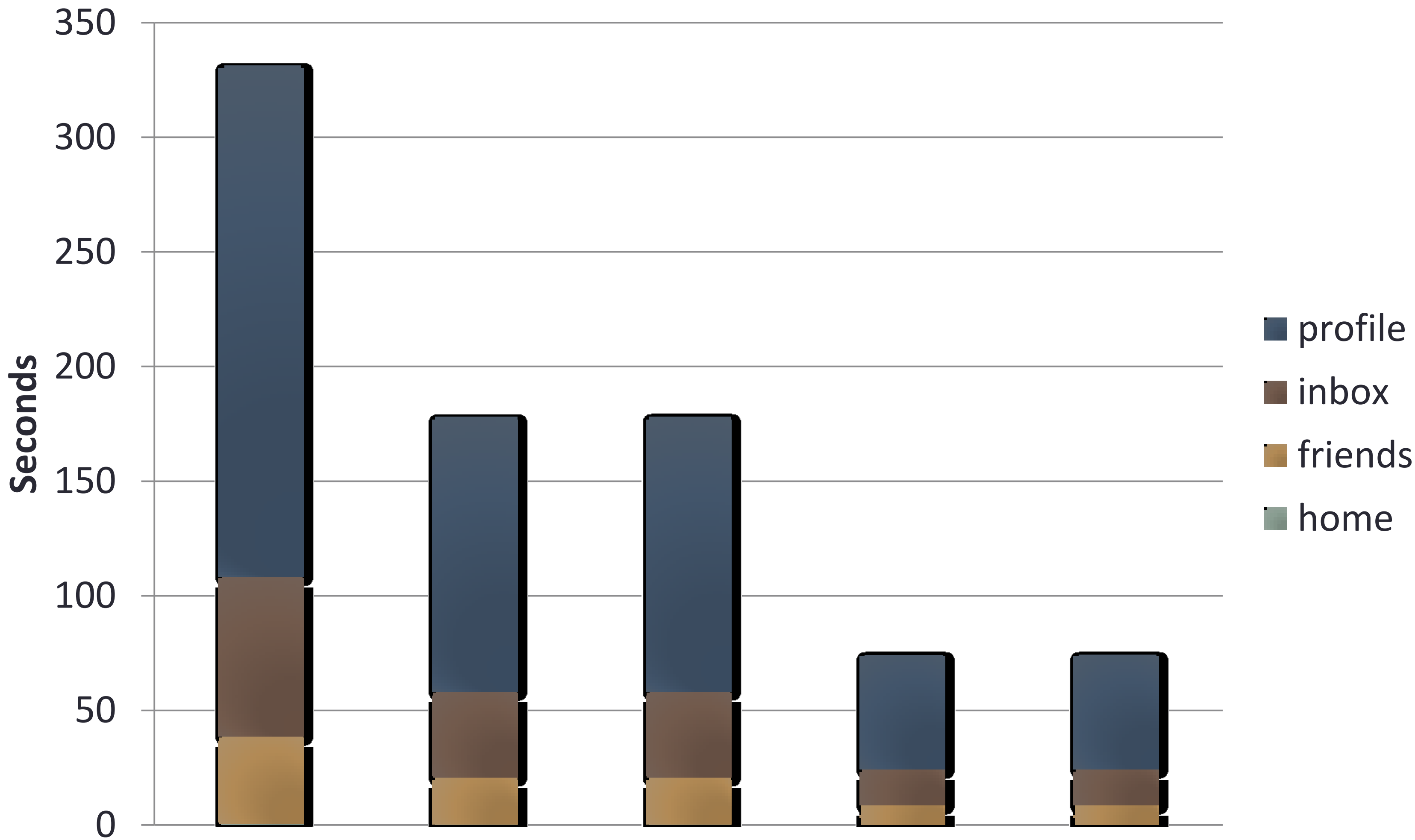
"+" means that staged incremental analysis is advantageous compared to full analysis on the client.

- **Slow devices** benefit from Gulfstream
- A **slow network** can negate the benefits of the staged analysis
- **Large page updates** don't benefit from Gulfstream

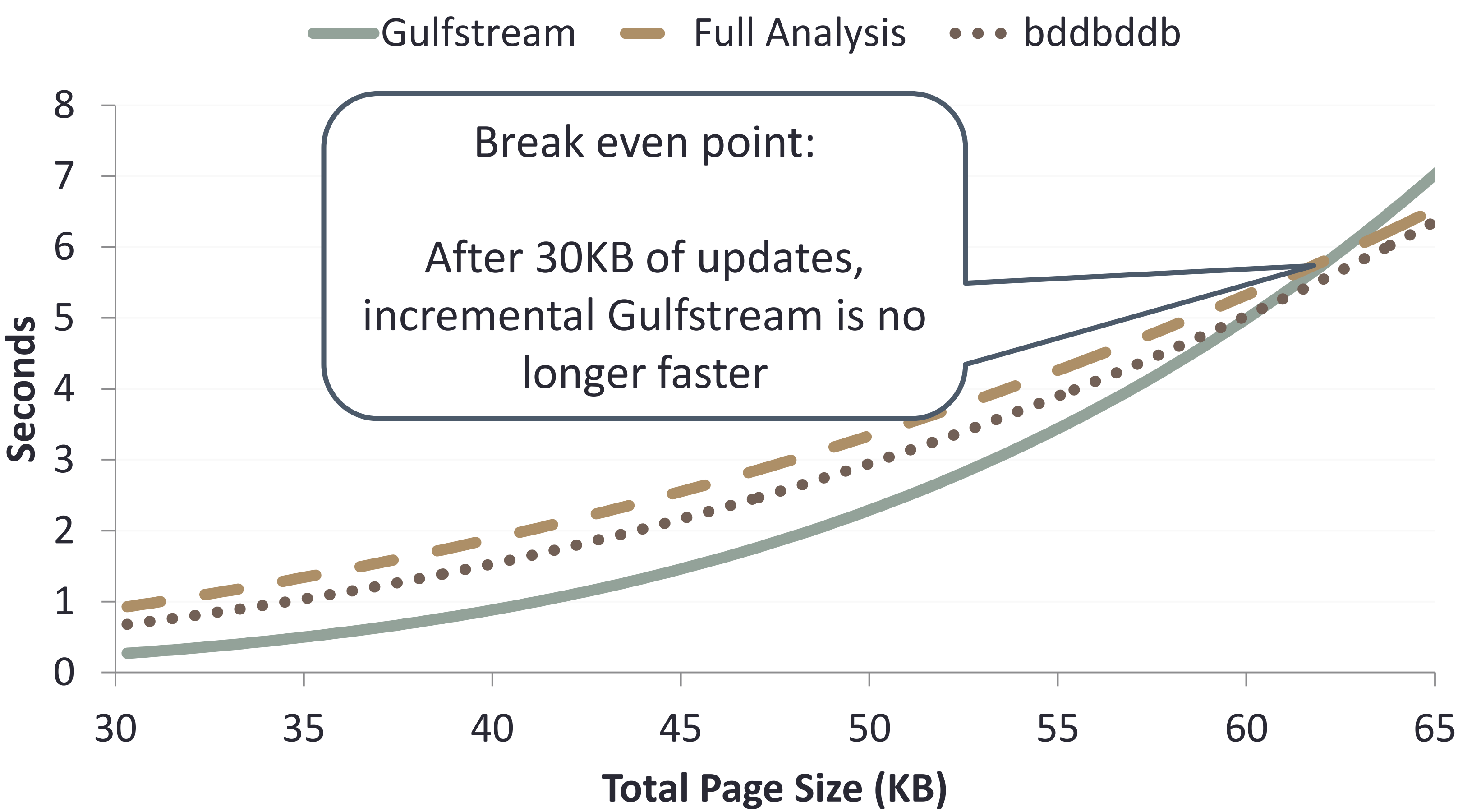
Gulfstream Savings: Fast Devices



Gulfstream Savings: Slow Devices



Laptop Running Time Comparison



Conclusion

- Gulfstream, staged analysis for JavaScript
- WebApps 2010

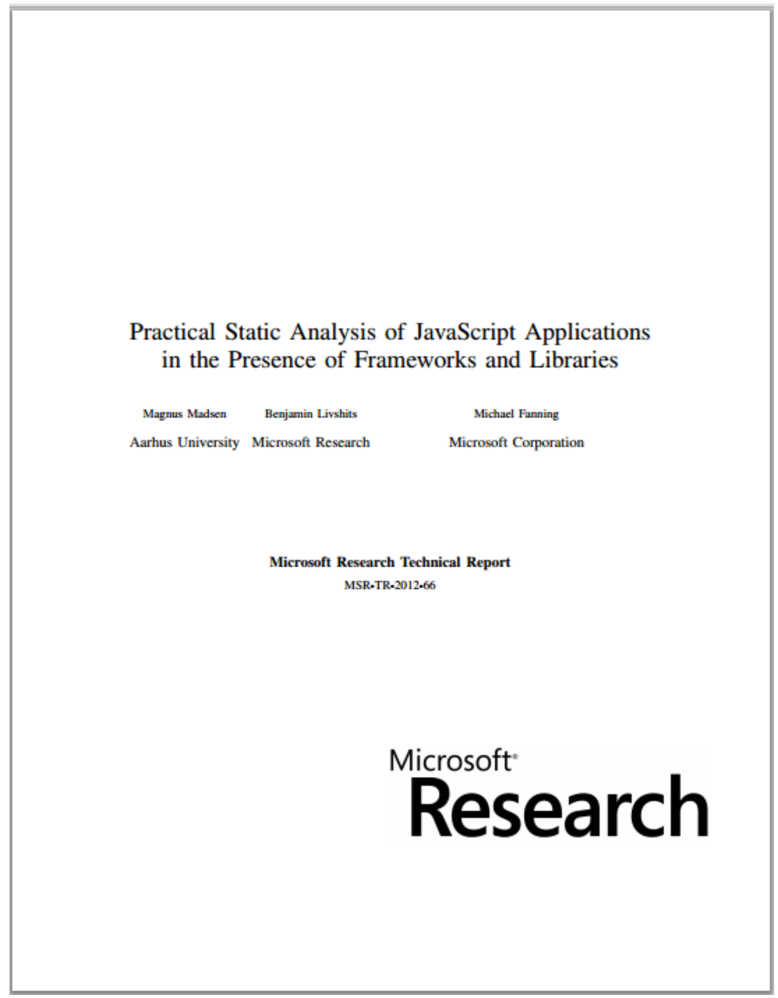
- Staged analysis
  - Offline on the server
  - Online in the browser

- Wide range of experiments
  - For small updates, Gulfstream is faster
  - Devices with slow CPU benefit most

Pointer Analysis and Use Analysis

Use Analysis

- *Practical Static Analysis of JavaScript Applications*
- *in the Presence of Frameworks and Libraries*, Madsen, Livshits, Fanning, in submission, 2013


Practical Static Analysis of JavaScript Applications
in the Presence of Frameworks and Libraries

Magnus Madsen — Aarhus University
Benjamin Livshits — Microsoft Research
Michael Fanning — Microsoft Corporation

Microsoft Research Technical Report
MSR-TR-2012-66

Microsoft Research

Motivation: Win8 App Store

Native C/C++ apps

.NET aps

JavaScript/HTML apps

Win8 & Web Applications

Windows 8 App

Builtin | DOM | WinJS | Win8

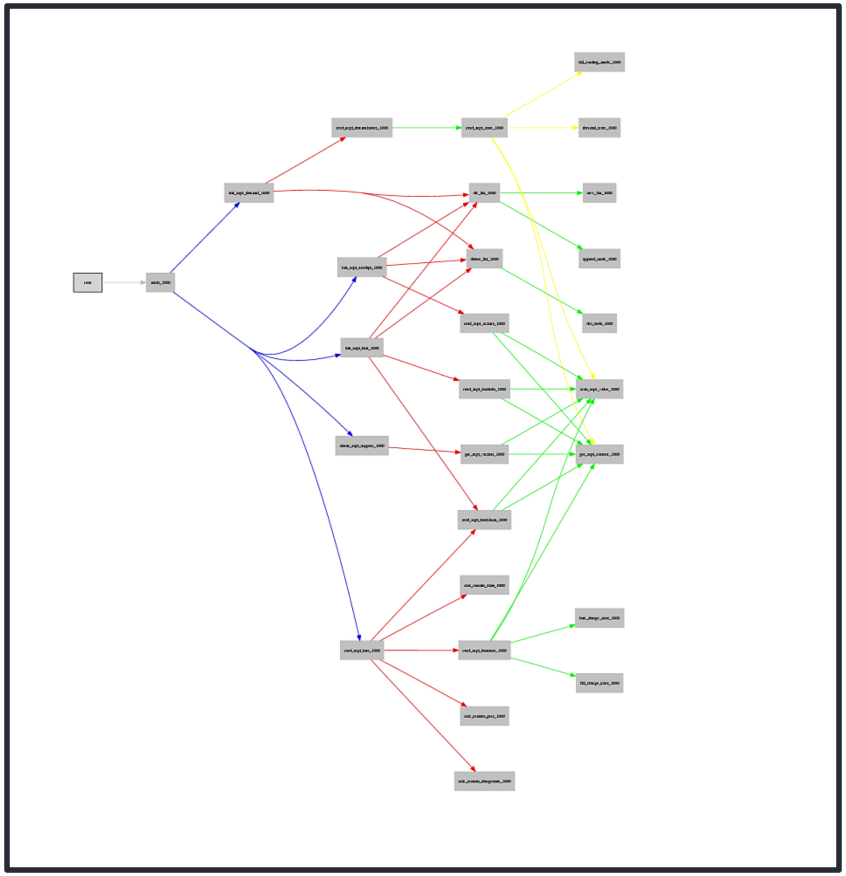
Web App

Builtin | DOM | jQuery | ...

Name	Lines	Functions	Alloc. sites	Fields
Builtin	225	161	1,039	190
DOM	21,881	12,696	44,947	1,326
WinJS	404	346	1,114	445
Windows 8 API	7,213	2,970	13,989	3,834
Total	29,723	16,173	61,089	5,795

Practical Applications

- **Call graph discovery**
- API surface discovery
- Capability analysis
- Auto-complete
- Concrete type inference
- Runtime optimizations



Practical Applications

- Call graph discovery
- **API surface discovery**
- Capability analysis
- Auto-complete
- Concrete type inference
- Runtime optimizations

```
Windows.Devices.Sensors
Windows.Devices.Sms
Windows.Media.Capture
Windows.Networking.Sockets
…
```

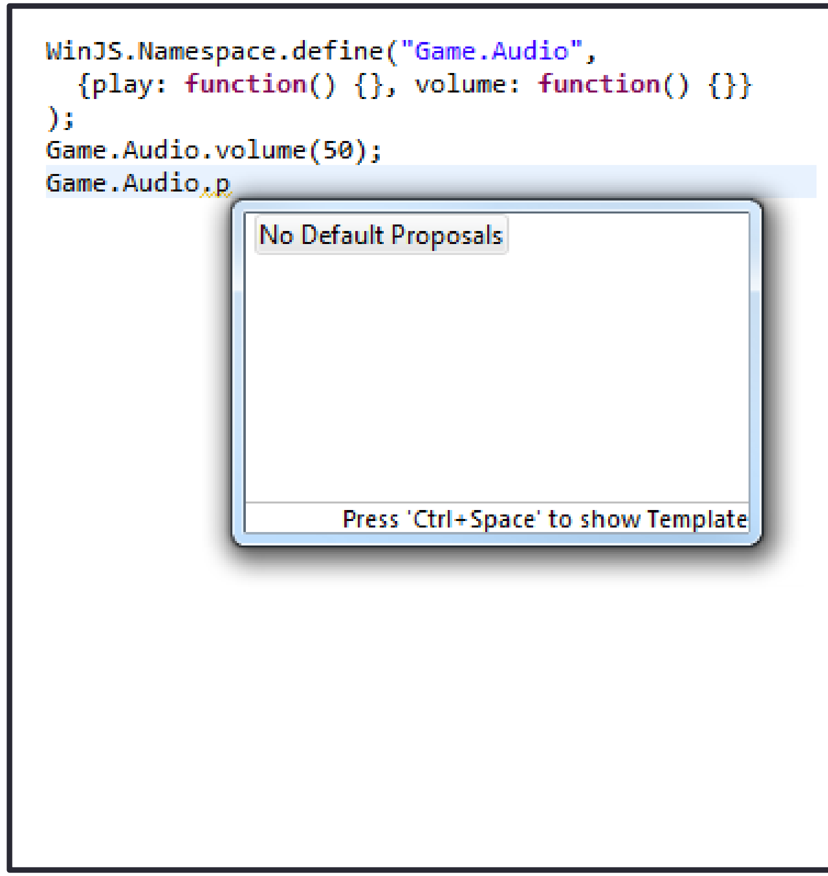
Practical Applications

- Call graph discovery
- API surface discovery
- **Capability analysis**
- Auto-complete
- Concrete type inference
- Runtime optimizations

```xml
<Package xmlns="http://schemas.microsoft.c
  <Identity Name="51e0e1dc-81a4-4bd0-964a-
  <Properties>
    <DisplayName>...</DisplayName>
    <Description>...</Description>
  </Properties>
  <Capabilities>
    <Capability Name="videosLibrary" />
    <Capability Name="picturesLibrary" />
    <Capability Name="internetClient" />
    <DeviceCapability Name="webcam" />
  </Capabilities>
</Package>
```

Practical Applications

- Call graph discovery
- API surface discovery
- Capability analysis
- **Auto-complete**
- Concrete type inference
- Runtime optimizations



Practical Applications

- Call graph discovery
- API surface discovery
- Capability analysis
- Auto-complete
- **Concrete type inference**
- Runtime optimizations

```
function Node(left, right) {
    this.color = "RED";
    this.height = 0;
    this.left = left;
    this.right = right;
}

var l = new Node(null, null);
var r = new Node(null, null);
var p = new Node(l, r);
```

Practical Applications

- Call graph discovery
- API surface discovery
- Capability analysis
- Auto-complete
- Concrete type inference
- **Runtime optimizations**

```
function Node(left, right) {
    this.color = "RED";
    this.height = 0;
    this.left = left;
    this.right = right;
}
```

str	int	ref	ref

memory layout

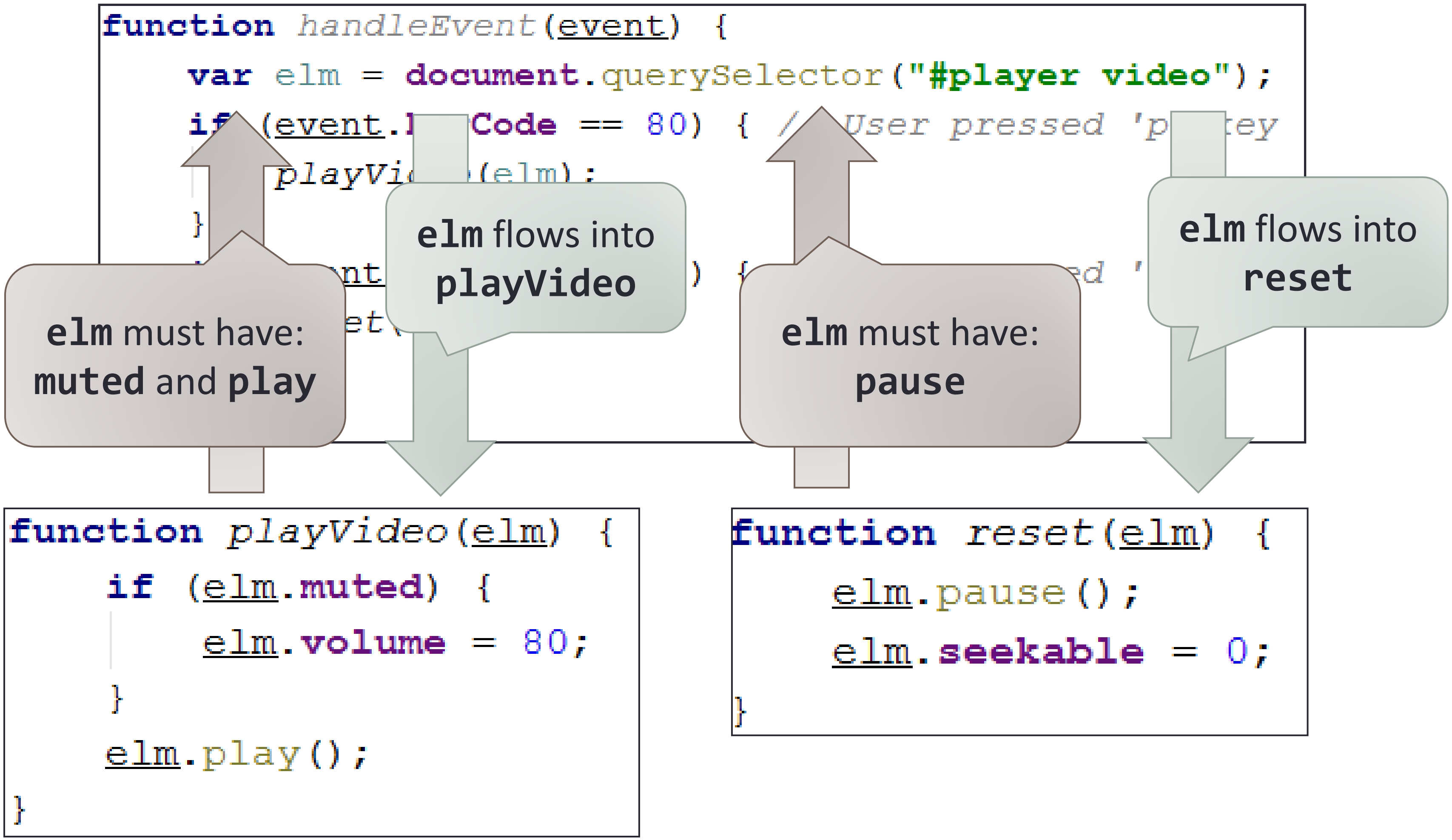
Canvas Dilemma

```
var canvas = document.querySelector("#leftcol .logo");
var context = canvas.getContext("2d");
context.fillRect(20, 20, c.width / 2, c.height / 2);
context.strokeRect(0, 0, c.width, c. height);
```

- model `querySelector` as returning a reference to **`HTMLElement:prototype`**
- However, `HTMLElement:prototype` does **not** define `getContext`, so `getContext` remains unresolved

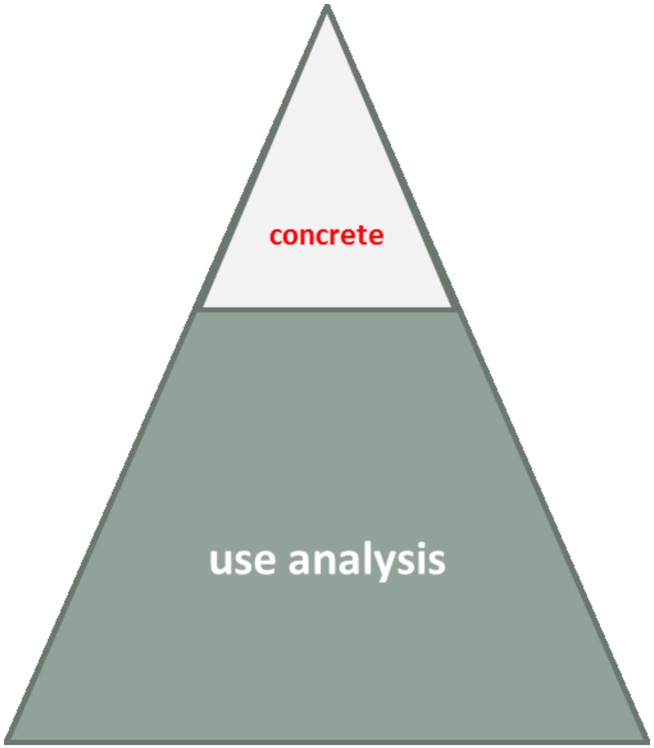
- Model `querySelector` as returning *any* HTML element within underlying page
- Returns elements on which `getContext` is undefined

Introducing Use Analysis

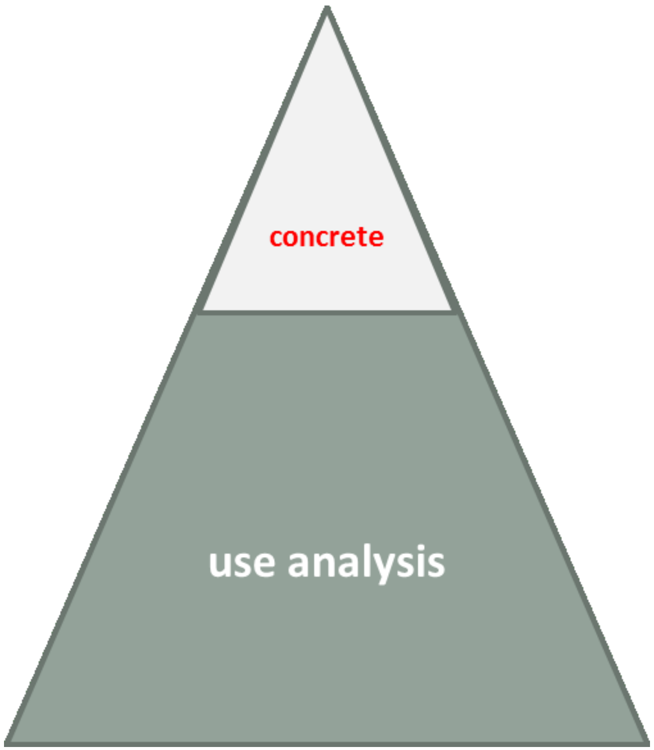


Pointer vs. Use Analysis

- Pointer analysis deals with “concrete” facts
- Facts we can observe
  - variables declared in the program
  - allocation sites



Pointer vs. Use Analysis



- Use analysis deals with the “invisible” part of the heap
- It can exist entirely outside the JavaScript heap
- Constraints flows from callers to callees

Promises

```
driveUtil.uploadFilesAsync(
      server.imagesFolderId).
            then( function (results) {...} ))
```

analysis correctly maps **then** to

```
      WinJS:Promise:prototype.then
```

Local Storage

```
var json =
     Windows.Storage.
          ApplicationData.current.
               localSettings.values[key];
```

correctly resolves **`localSettings`** to an instance of `Windows:Storage:ApplicationDataContainer`

Benchmarks

Lines	Functions	Alloc. sites	Call sites	Fields	Variables
245	11	128	113	231	470
345	74	606	345	298	1,749
402	27	236	137	298	769
434	51	282	[illegible]	[illegible]	[illegible]
488	53	369	[illegible]	[illegible]	[illegible]
2,351	192	[illegible]	[illegible]	[illegible]	[illegible]
2,524	228	[illegible]	[illegible]	[illegible]	[illegible]
3,159	161	2,335	[illegible]	[illegible]	[illegible]
3,189	244	2,333	939	534	6,297
3,243	108	1,654	740	515	4,517
3,638	305	2,529	1,153	537	7,139
6,169	506	3,682	2,994	725	12,667
1,587	**134**	**1,147**	**631**	**442**	**3,511**

25 Windows 8 Apps:

Average 1,587 lines of code

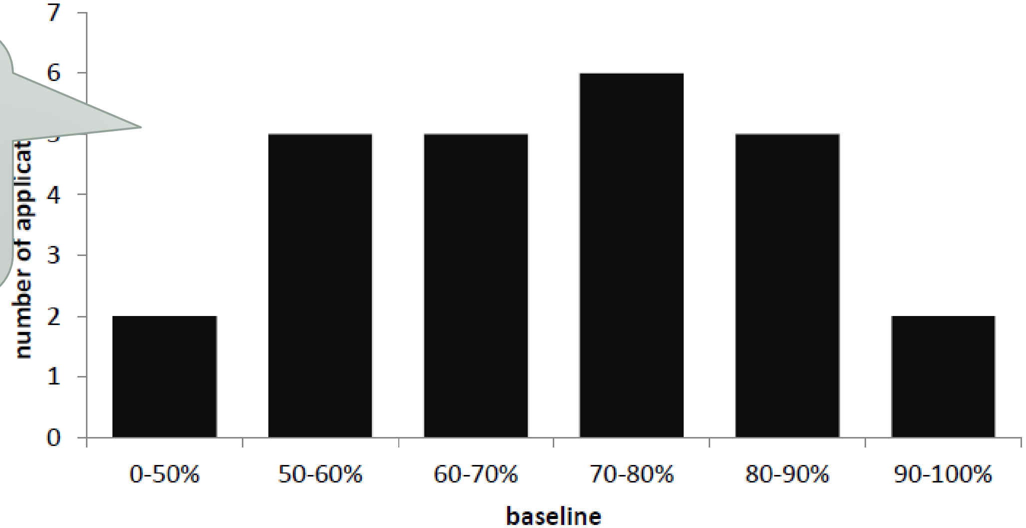
Approx. 30,000 lines of stubs

Evaluation: Summary

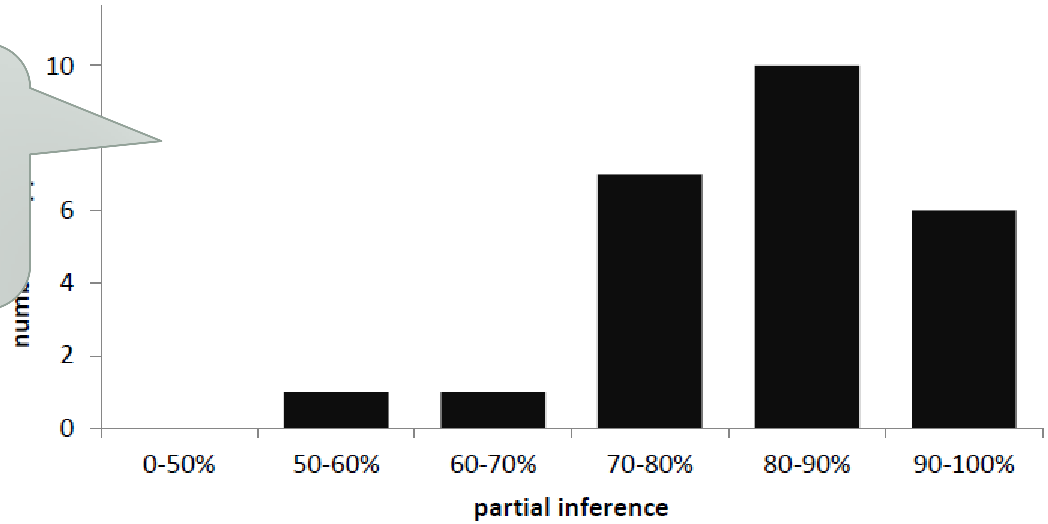
- The technique **improves** call graph resolution
- Unification is both **effective** and **precise**
- The technique improves auto-completion compared to what is found in four widely used IDEs
- Analysis completes in a reasonable amount of time

Call Graph Resolution

Median **baseline** resolution is **71.5%**



Median **partial** resolution is **81.5%**



Validating Results

App	OK	Incomplete	Unsound	Unknown	Stubs	Total
app1	16	1	2	0	1	20
app2	11	5	1	0	3	20
app3	12	5	0	0	3	20
app4	13	4	1	0	2	20
app5	13	4	0	1	2	20
app6	15	2	0	0	3	20
app7	20	0	0	0	0	20
app8	12	5	0	1	2	20
app9	12	5	0	0	3	20
app10	11	4	0	3	2	20
Total	135	35	4	5	21	200

- **Incomplete** is # of call sites which are sound, but have some spurious targets (i.e. imprecision is present)
- **Unsound** is the number of call sites for which some call targets are missing (i.e. the set of targets is too small)
- **Stubs** is the number of call sites which were unresolved due to missing or faulty stubs.

Auto-complete

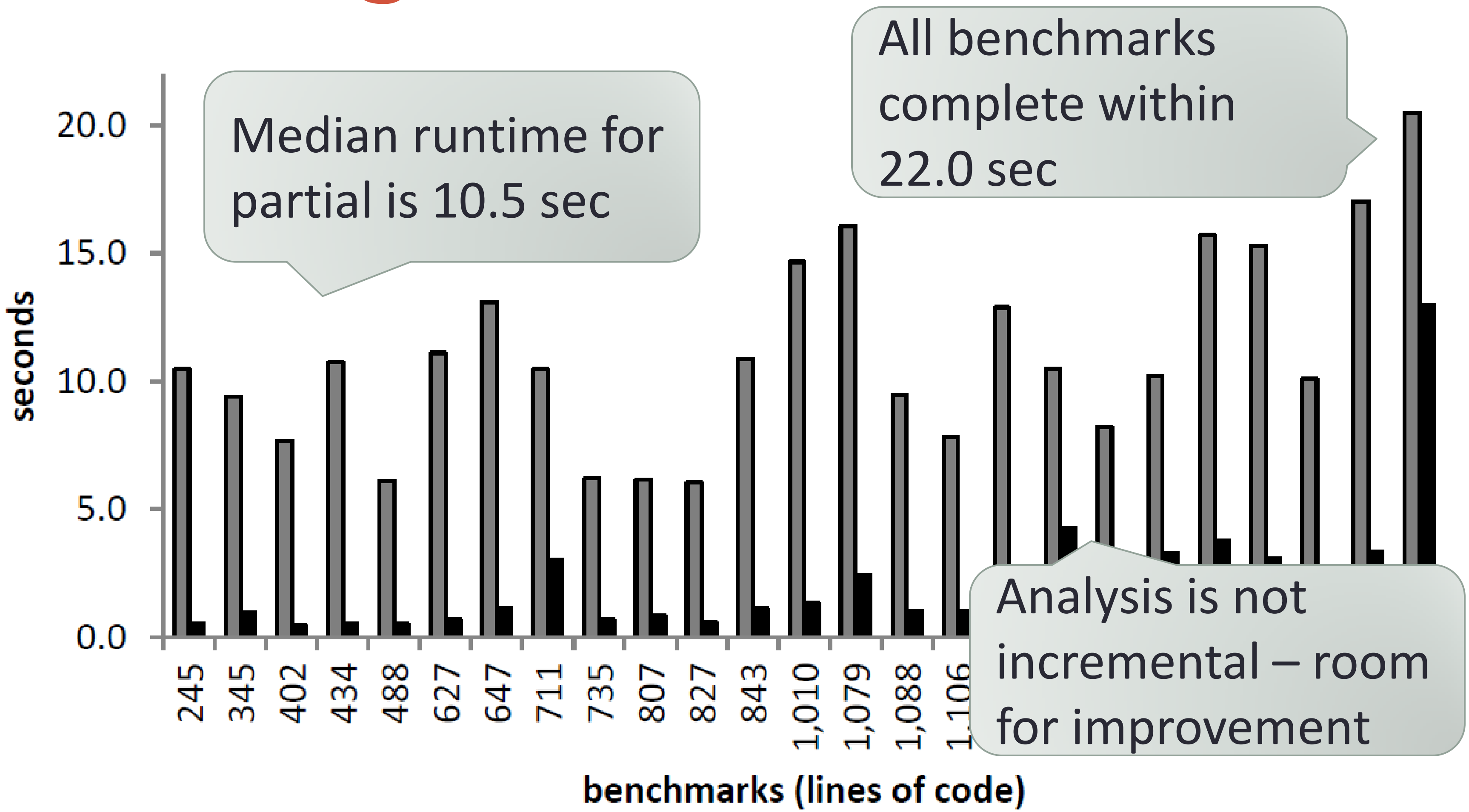
- We compared our technique to the auto-complete in four popular IDEs:
  - Eclipse for JavaScript developers
  - IntelliJ IDEA
  - Visual Studio 2010
  - *Visual Studio 2012*

- In all cases, where libraries were involved, our technique was an improvement

Auto-complete

	Category	Code	Eclipse ✓	Eclipse #	IntelliJ ✓	IntelliJ #	VS 2010 ✓	VS 2010 #	VS 2012 ✓	VS 2012 #
		PARTIAL INFERENCE								
1	DOM Loop	`var c = document.getElementById("canvas"); var ctx = c.getContext("2d"); var h = c.height; var w = c.w␣`	✗	0	✓	35	✗	26	✓	1
2	Callback	`var p = {firstName : "John", lastName : "Doe"}; function compare(p1, p2) { var c = p1.firstName < p2.firstName; if(c != 0) return c; return p1.last␣ }`	✗	0	✓	9	✗	7	✓*	k
3	Local Storage	`var p1 = {firstName : "John", lastName : "Doe"}; localStorage.putItem("person", p1); var p2 = localStorage.getItem("person"); document.writeln("Mr." + p2.lastName+ "," + p2.f␣);`	✗	0	✓	50+	✗	7	✗	7
		FULL INFERENCE								
4	Namespace	`WinJS.Namespace.define("Game.Audio", play : function() {}, volume : function() {} ); Game.Audio.volume(50); Game.Audio.p␣`	✗	0	✓	50+	✗	1	✓*	k
5	Paths	`var d = new Windows.UI.Popups.MessageDialog(); var m = new Windows.UI.␣`	✗	0	✗	250+	✗	7	✓*	k

Running Times



Two Issues in JavaScript Pointer Analysis

Gulfstream

- JavaScript programs on the web are streaming
- Fully static analysis pointer analysis is not possible, calling for a hybrid approach
- Setting: analyzing pages before they reach the browser

JSCap

- JavaScript programs interop with a set of reach APIs such as the DOM
- We need to understand these APIs for analysis to be useful
- Setting: analyzing Win8 apps written in JavaScript