

AjaxScope: Remotely Monitoring Client-side Web-App Behavior

Emre Kiciman

emrek@microsoft.com

Ben Livshits

livshits@microsoft.com

Internet Services Research Center
Microsoft Research

Runtime Analysis & Design
Microsoft Research

A Web Application

The screenshot displays a Windows Internet Explorer browser window with the URL `http://maps.live.com/`. The page title is "Live Search - Windows Internet Explorer". The search bar contains the text "skamania hotel". Below the search bar, there are tabs for "Businesses", "People", and "Maps". The "Maps" tab is selected, showing a map of the Skamania area in Washington state. The map includes a search result for "Skamania Lodge" with the address "1131 Skamania Lodge Dr, Stevenson, WA | 4.62mi (509) 427-7700" and a "Call for free" link. A "Scratch pad" panel is open on the right side of the map, displaying a message: "You must Sign in to save your collection" and instructions on how to add items to the scratch pad. The map interface includes a navigation pane on the left with options for "Road", "Aerial", "Hybrid", and "Mountain" views. The bottom of the browser window shows the status bar with "Done", "Internet | Protected Mode: On", and "100%" zoom level.

Live Search - Windows Internet Explorer
http://maps.live.com/

MSR Links Ajax View Statistics

Live Search

skamania hotel

Enter city, address, or landmark

Businesses People Maps

Web Images Video News Maps MSN More Academic Beta Books Beta

skamania hotel

Sort by relevance Map all

Skamania Lodge
1131 Skamania Lodge Dr,
Stevenson, WA | 4.62mi
(509) 427-7700
Call for free

Didn't find what you were looking for?
[Help us improve](#)

Welcome Collections Driving directions Traffic Locate me Share Print

2D 3D

Road
Aerial
Hybrid
Mountain

Scratch pad

Unsaved collection

You must Sign in to save your collection

Share Map all Clear

Your scratch pad is empty. To add items to your scratch pad, you can:

- Hover over a search result and choose **Add to collection**
- or
- Add a pushpin to the map

Bloucher
Winans Holsteen
Fir
Dee Trout Creek
Mt Hood
Mount Hood
Parkdale

9 miles

© 2007 Microsoft Corporation. © 2005. IANTEL
© AND

Microsoft Virtual Earth™

© 2007 Microsoft Corporation Privacy Code of Conduct Legal Trademarks Developers About | Help | Feedback

Done Internet | Protected Mode: On 100%

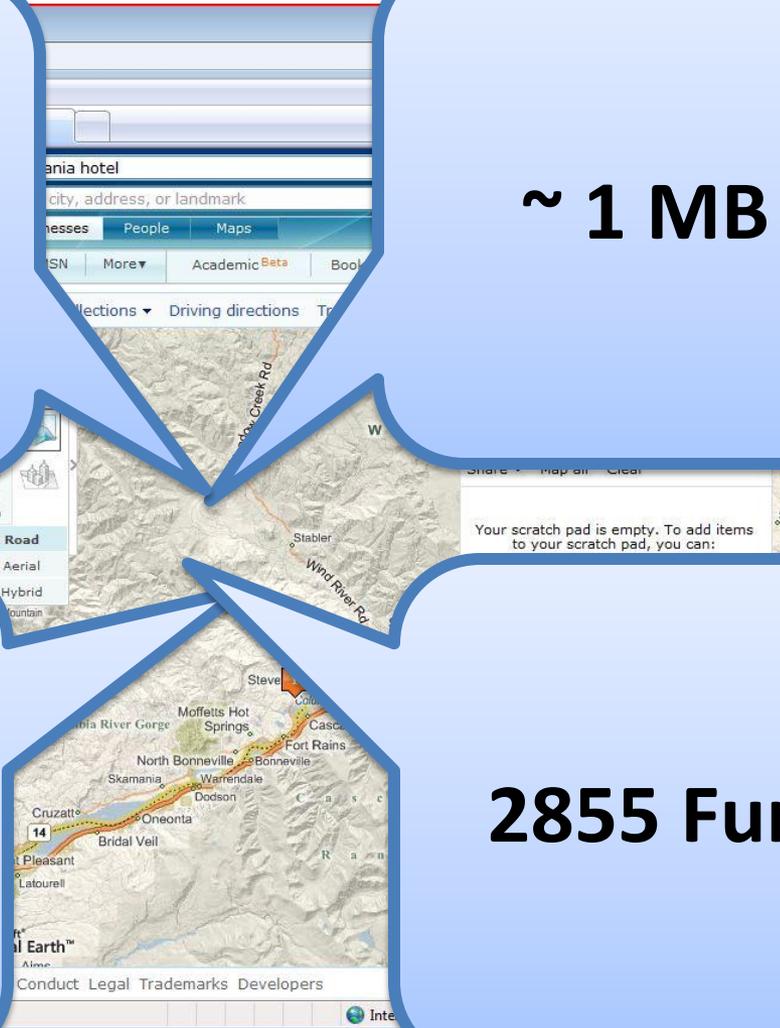
A Web Application

Talks to >14
backend services
(traffic, images,
search, directions,
ads, ...)

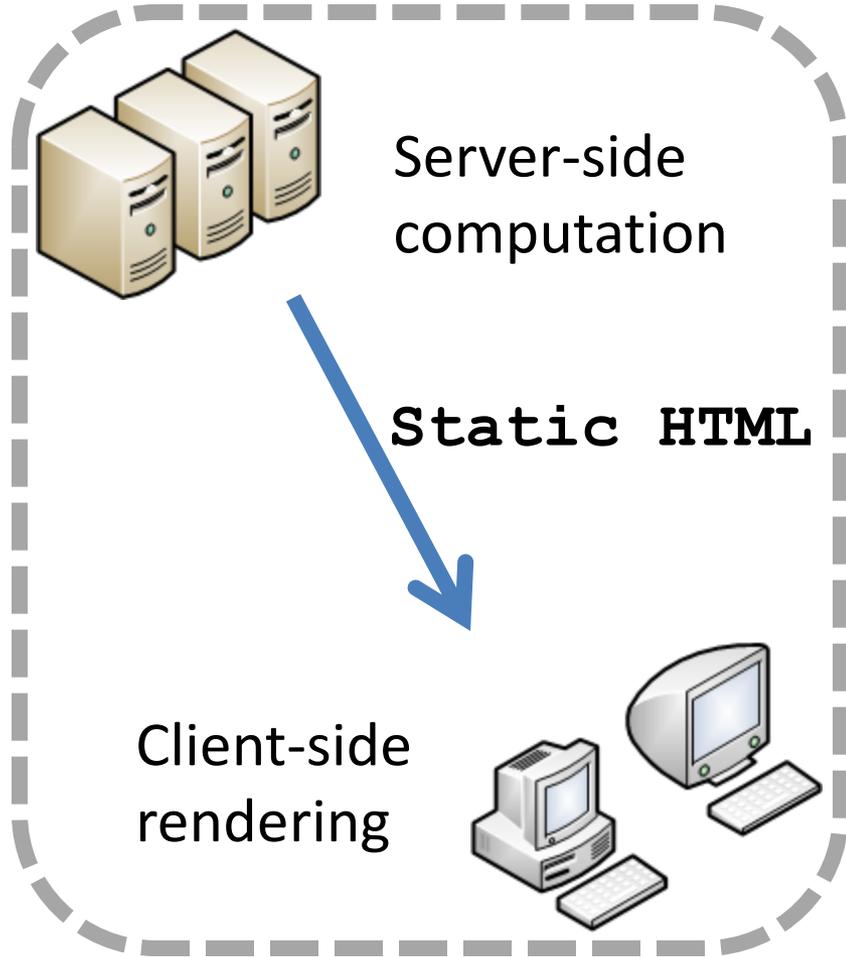
~ 1 MB code

70k lines of
JavaScript code
downloaded to
the client.

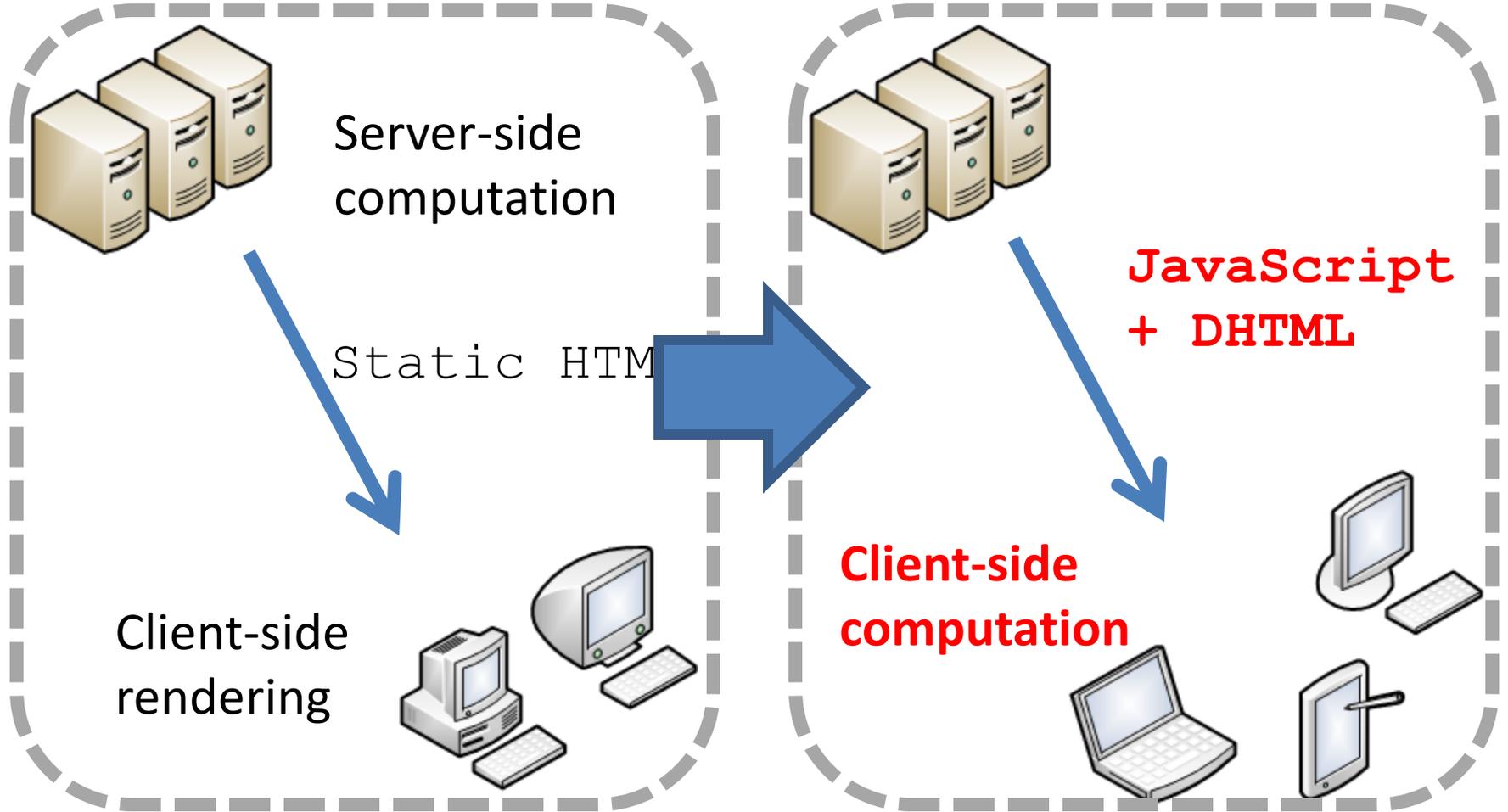
2855 Functions



Web 1.0 → Web 2.0



Web 1.0 → Web 2.0



Web App Challenges

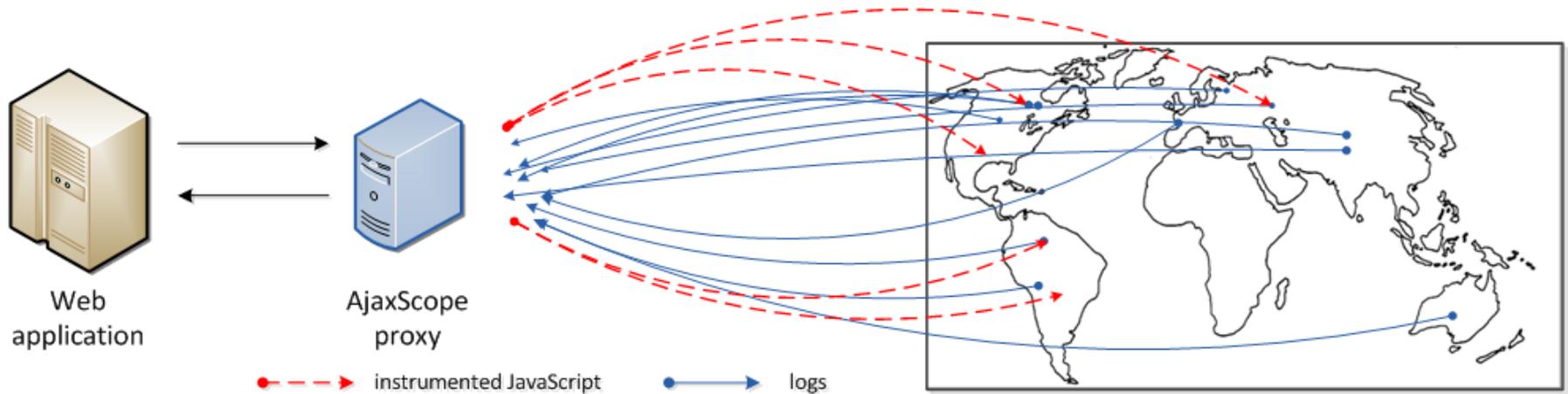
- **Code complexity: more client-side code**
 - Ex. Maps.live.com: 1MB of code, 70k LoC
 - Bugs, race conditions, memory leaks, ...
- **Non-standard execution environments**
 - Many APIs differ across browsers
 - Perf of simple ops vary 10x-100x across impl.
- **Third-party dependencies** (*e.g.*, mash-ups)

Missing: Visibility into behavior on clients

Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions

AjaxScope Approach



Goal: Detailed visibility into app behavior in the client

Approach: On-the-fly rewriting to add instrumentation

Key Enabler: Instant re-deployability of web apps

Monitoring Goals

- **Performance Optimization**
 - Performance profiling
 - String optimization; cache placement; ...
 - Code splitting and joining
- **Debugging**
 - Report function arguments, app state, errors
 - Memory leak checking
 - Statistical debugging
- **Test**
 - Code coverage
 - A/B tests
- **Operations**
 - Measure RPC network latencies
- **User interaction feedback**
 - What features are being used / discovered?

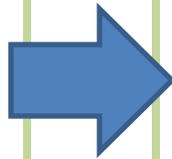
Approach: JavaScript Rewriting

- Simple but powerful monitoring
 - Inspect application state
 - Observe control flow
 - Limited only by JS sandbox
- Easy deployability
 - No changes required to original web app
 - No changes to client-side browsers

Example: Record Function Args

1. Search JavaScript AST for function definitions
2. For each function definition, add a statement to report every argument.

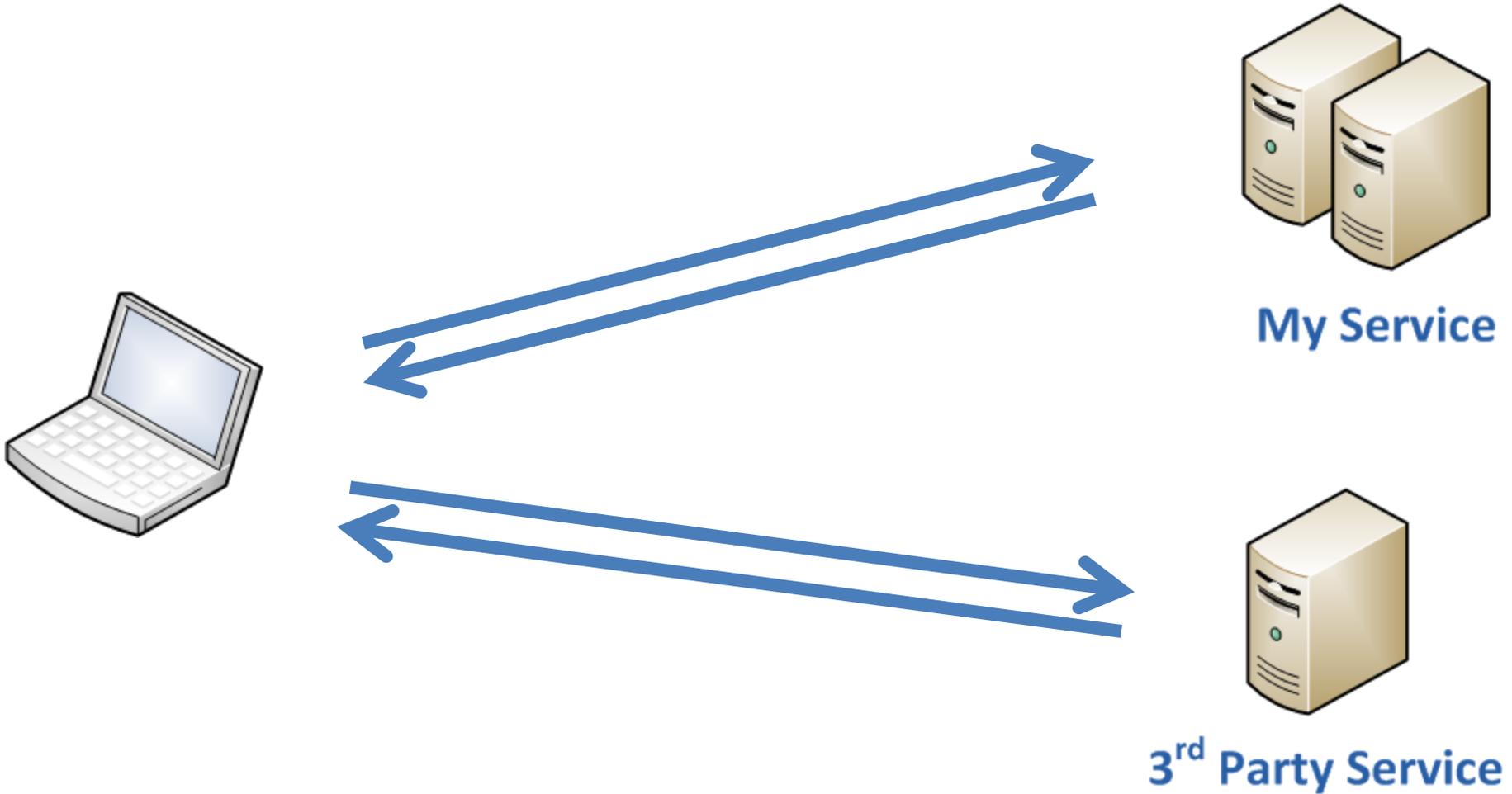
```
function foo(a,b) {  
  // do something  
}
```



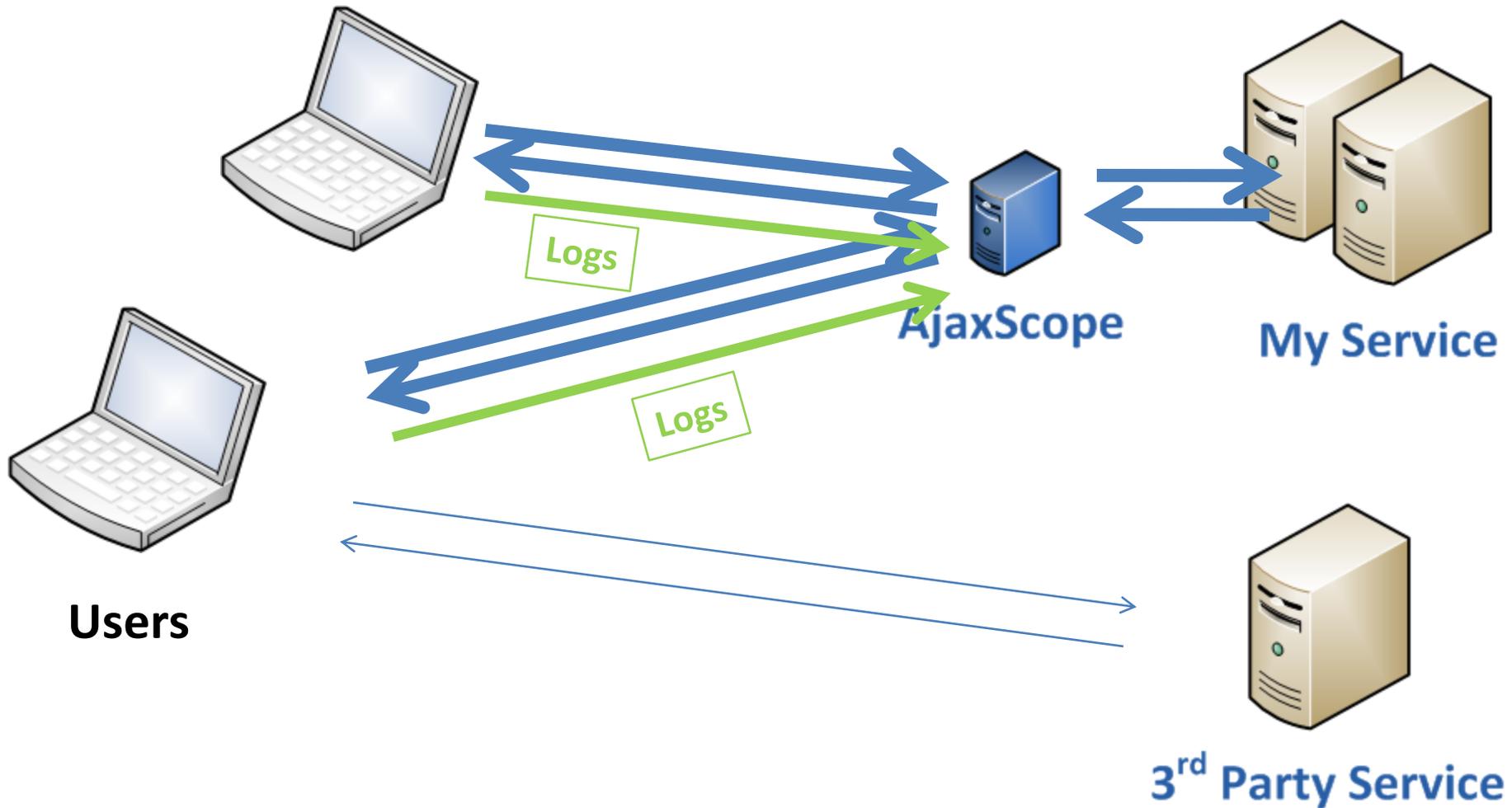
```
function foo(a,b) {  
  sendLog("value of a="+a);  
  sendLog("value of b="+b);  
  // do something  
}
```

sendLog() queues up messages for bulk reporting to AjaxScope

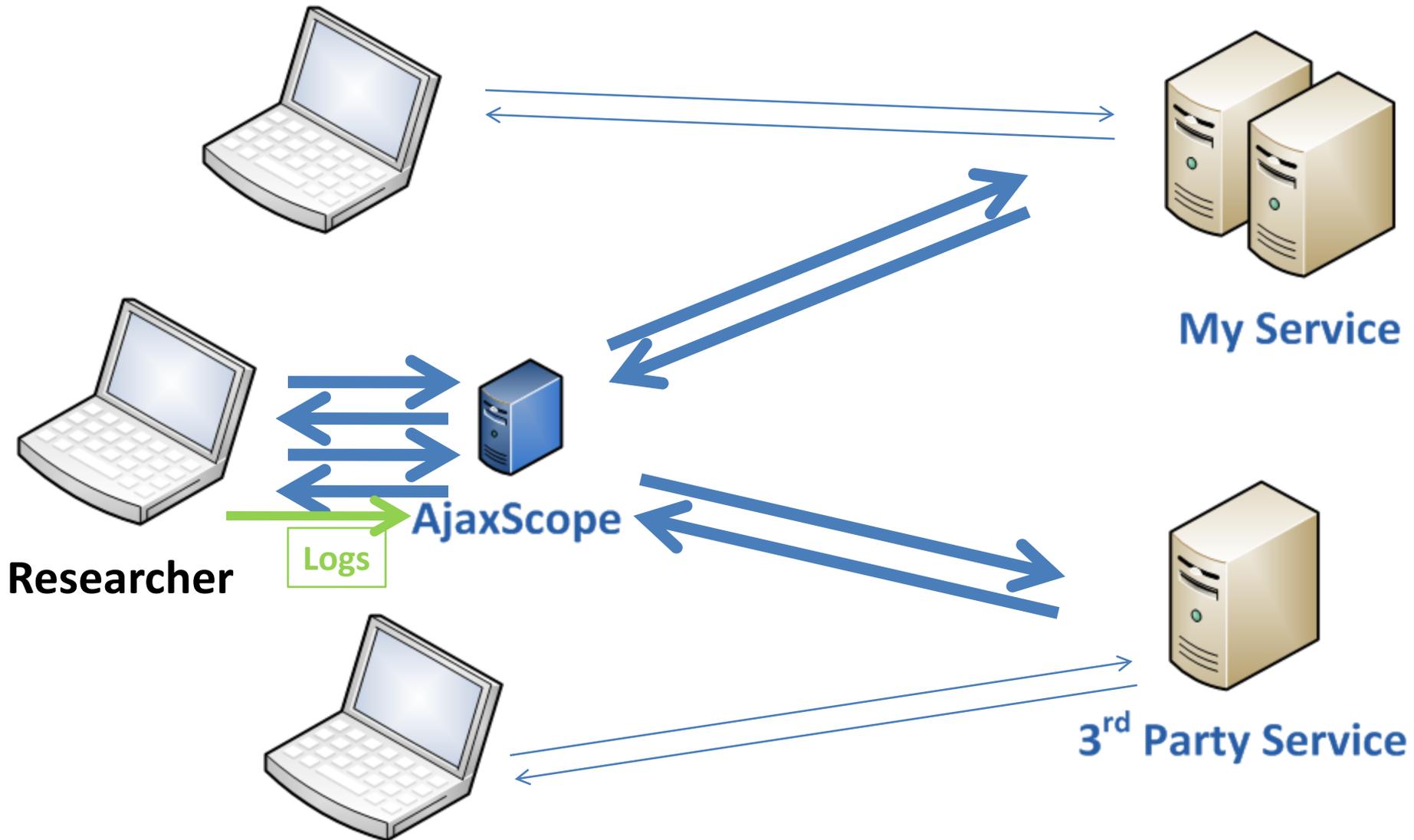
Deploying AjaxScope...



Server-side Deployment

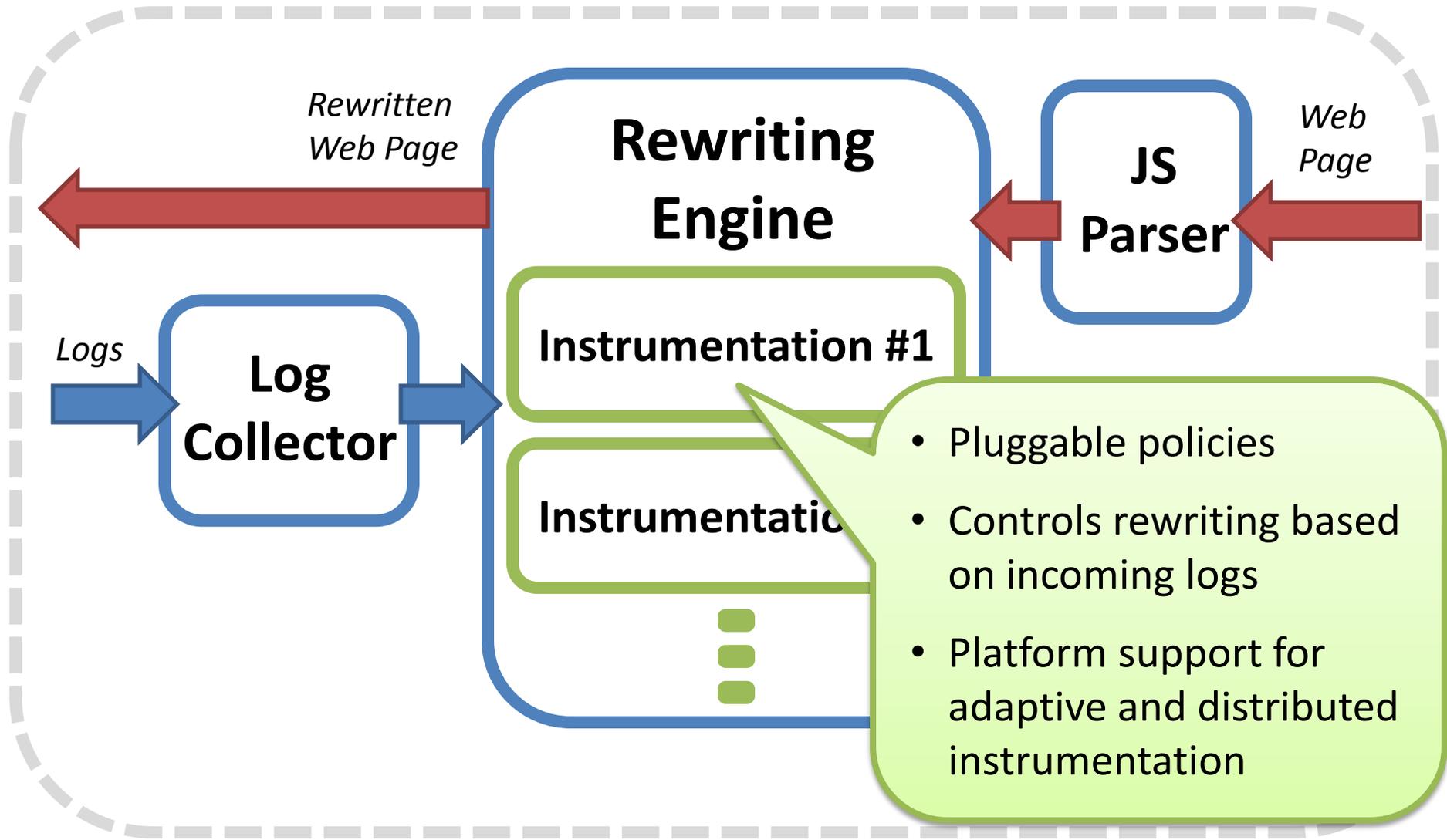


Our Prototype (Client-side)





AjaxScope Proxy



Rewrite “On-the-fly”

- Service has tight control over code running at client
 - Clients always download new version
 - Cache-ability controlled by service
- Enables dynamic instrumentation
- Use to reduce performance overhead
 1. Adaptive instrumentation
 2. Distributed Instrumentation
- Also enables A/B tests to compare versions

Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions

Experimental Setup

- Profile **90** web sites' "startup"
 - Client-side AjaxScope

	Site	Code Size (kB)	# of Functions	Exec Time (ms)
Maps	maps.google.com	295	1935	530
	maps.live.com	924	2855	190
Portals	msn.com	124	592	300
	yahoo.com	278	1097	670
	google.com/ig	135	960	190
	protopages.com	599	1862	13780

+ 6 more JS-heavy news & game sites

+ 78 sites randomly chosen, weighted by popularity

Adaptation: Drill-down Perf Profiling

- Naïve: Add timestamps everywhere
 - Too expensive! (both CPU and logging BW)
- Instead, auto-drill-down based on user experience

```
<script>
  LogTime();
  FastFunc1();
  FastFunc2();
  SlowFunc();
  LogTime();
</script>
```

If it's
slow

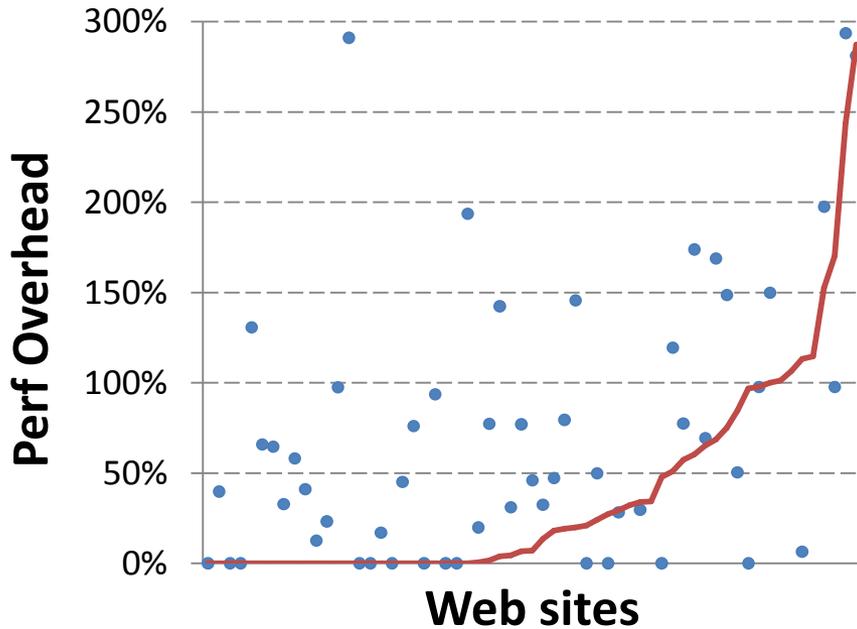
```
<script>
  LogTime();
  FastFunc1();
  LogTime();
  FastFunc2();
  LogTime();
  SlowFunc();
  LogTime();
</script>
```

Found
it!

```
<script>
  FastFunc1();
  FastFunc2();
  SlowFunc();
</script>
...
function SlowFunc() {
  // drill-down continues
}
```

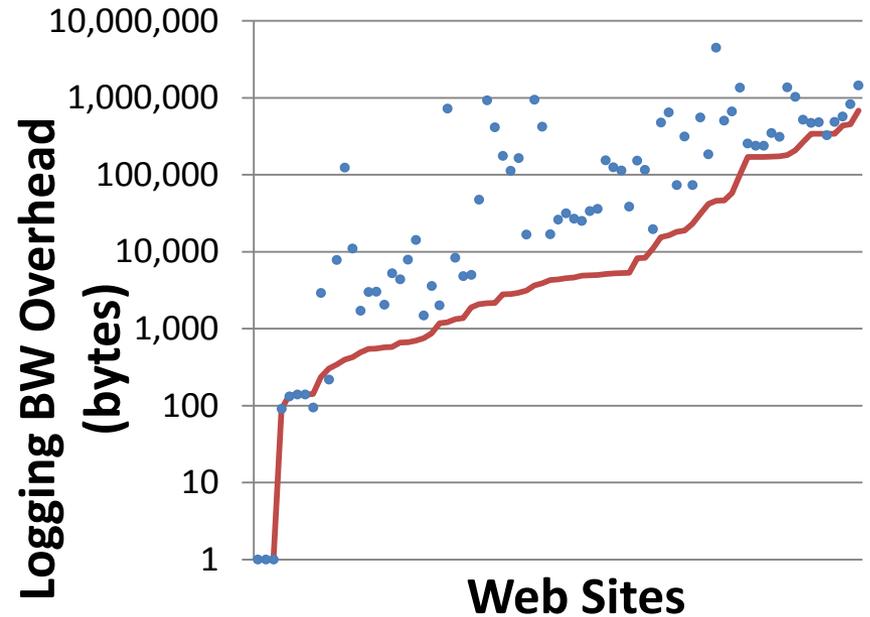
Adaptation Results

• Full Profiling — Drill-down Profiling



Avg 30% reduction in CPU Overhead

• Full Profiling — Drill-down Profiling



95% Avg Reduction in Logging Bandwidth

Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. Conclusions

Monitoring for JS Memory Leaks

- Mem leaks major problem in older browsers
 - Web apps can work-around
- E.g., Circular reference across DOM + JavaScript memory heaps
- Instrumentation looks for runtime patterns indicative of leak
- Expensive! Use distribution to reduce per-user overhead

Example: CNN.com

```
var pipelineContainers =  
    document.getElementById("cnnPipelineMod  
        getElementsByTagName("div");
```

First, get DOM
elements

...

```
for (var i=0; i<pipelineContainers.length; i++){  
    var pipelineContainer = pipelineContainers[i];  
    if(pipelineContainer.id.substr(0,9) == "plineCntr") {  
        pipelineContainer.onmouseover = function ()  
            {CNN_changeBackground(this,1); return false;}  
    }  
}
```

Then, set their event
handlers to a new
function

Example: CNN.com

```
var pipelineContainers =
  document.getElementById("cnnPipelineModule").
    getElementsByTagName("div");
...
for (var i=0; i<pipelineContainers.length; i++){
  var pipelineContainer = pipelineContainers[i];
  if(pipelineContainer.id.substr(0,9) == "plir")
    pipelineContainer.onmouseover = function ()
    {CNN_changeBackground(this,1); return false;}
}
```

DOM
object

JavaScript
Object

function closure references pipelineContainer

Checking for Memory Leaks

Check all object assignments for potential cycles

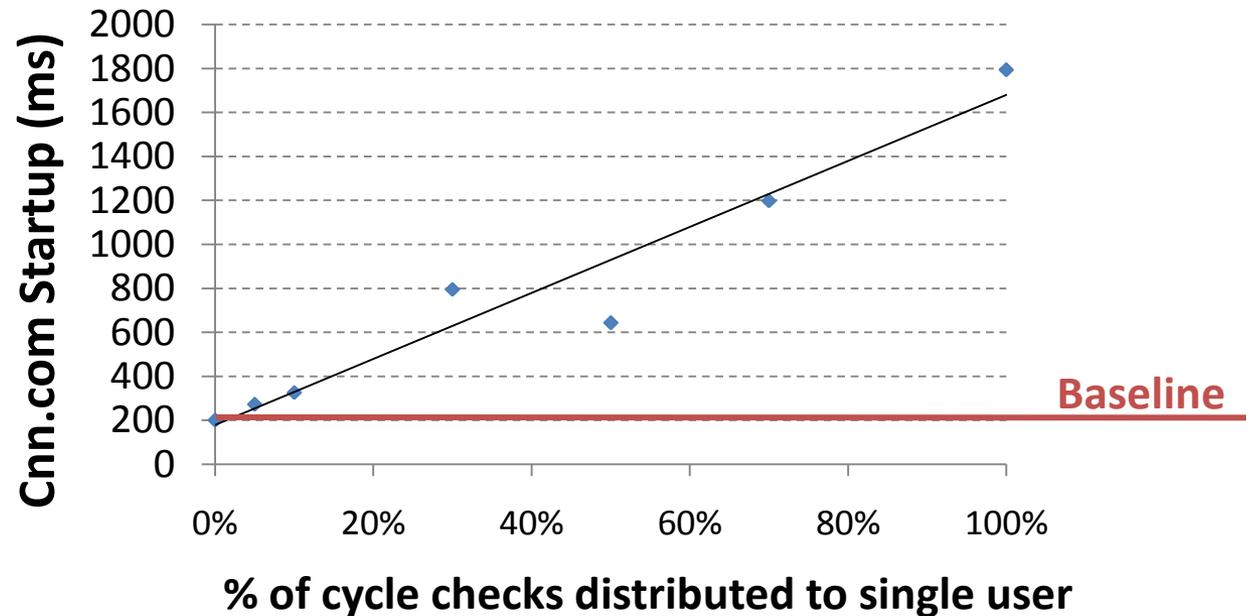
```
a.b = c;
```

```
a.b = c
```

```
TraverseHeapAndCheckForCycles(c, a);
```

- Distribute expensive traversals across users
 - Each user gets random N% of checks
 - Controls per-user overhead

Distribution Gives Fine Control of Per-User Overhead



Trade-off per-user overhead vs. detection speed

Outline

1. Motivation
2. AjaxScope Platform
3. Expt: Adaptive instrumentation
4. Expt: Distributed instrumentation
5. **Conclusions**

Related Work

- JavaScript rewriting for safety & security
 - BrowserShield and CoreScript
- Monitoring and tracing systems
 - E.g., Magpie, Project5
- Dynamic and adaptive instrumentation
 - In parallel computing cluster: ParaDyn
- Runtime program analysis for bug finding
 - Statistical debugging, taint analysis, ...

Future Work

- Platform improvements:
 - Integrate caching considerations into rewriting
 - Limit risk of bad rewriting with meta-monitoring
 - Improved information protection
- Collecting data and analysis:
 - Compare executions across users to find outliers
 - Collect dynamic call graphs to inform smart prefetching

Conclusions

- End-to-end visibility into client-side web app
 - Requires no client-side / server-side changes
- Distribution and adaptation controls overhead
 - While maintaining high coverage & detail
- Demonstrated variety of instrumentation policies
 - Performance profiler, memory leak checker, cache placement, ...
- Download and extend the prototype
 - <http://research.microsoft.com/projects/ajaxview/>
 - Supports plug-ins for new instrumentation policies

(this page intentionally left blank)