jsprobes

cross-platform browser instrumentation using JavaScript

Brian Burg (@brrian / burg@cs.uw.edu)

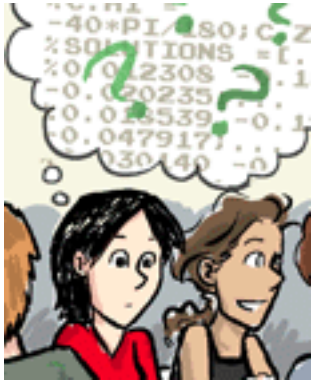
mozilla research

Background:
browser research template

1

Become puzzled by a question

2



Get real data and analyze

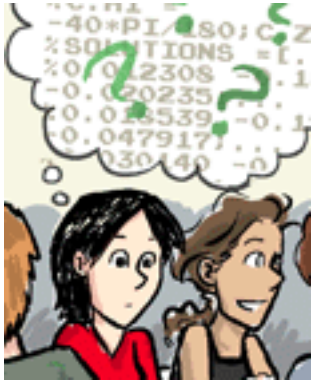
3

Write paper with answers

images from www.phdcomics.com

Right now, this is the hard part

1 | **2** | 3



Become puzzled by a question

Get real data and analyze

Write paper with answers

images from www.phdcomics.com

Browser instrumentation wishlist

- Cross-platform/architecture
- Low/no performance overhead
- Shareable/distributable
- Runtime flexibility
- Familiar programming model
- Cross-language/cross-component

Existing approaches: browser addons/extensions

Advantages

- Somewhat familiar programming model
- Easy to distribute
- Cross-platform
- Flexibility at runtime

Disadvantages

- Can't access all parts of the stack
- Bad for critical path instrumentation

Image: Mozilla Test Pilot (www.mozillalabs.com/testpilot/)

Existing approaches: platform instrumentation

ETW

Advantages

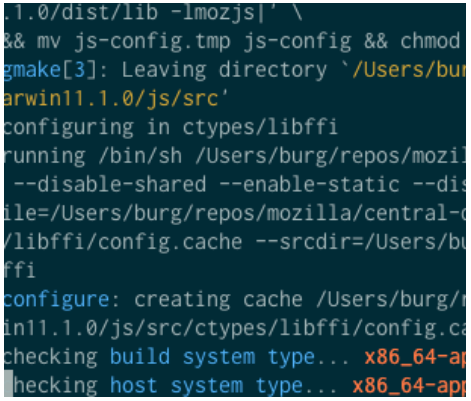
- Extremely low/no performance overhead
- Can instrument any browser component
- Runtime flexibility

Disadvantages

- Needs escalated (root) privileges/kernel mod
- Platform-specific
- Limited programming language/model
- Cannot distribute

Images: dtrace.org, en.wikipedia.org/wiki/SystemTap

Existing approaches: modifying browser source



Advantages

- Can do anything!
- (possible) low performance impact

Disadvantages

- Difficult to understand and modify
- Easy to cause crashes
- Hard to distribute/ share/reuse
- Very fragile vs. upstream changes

jsprobes: an experiment

- A browser instrumentation framework…
  - With instrumentation written in JavaScript
  - Accessible via XPCOM to addons
  - Available on all platforms/architectures
  - Which encourages experimentation
  - That can gather many kinds of data
  - Fast enough to gather low-level data

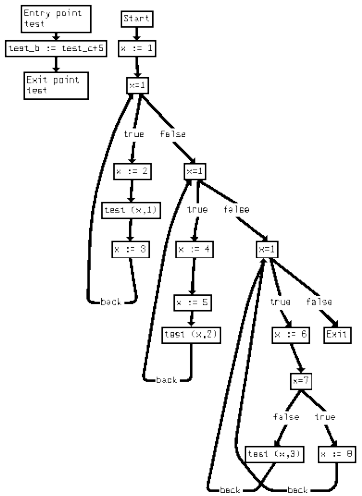
jsprobes terminology

probe point

```
390     if (!hasAvailableAr
391         removeFromAvail
392
393     JSRuntime *rt = inf
394     Probes::resizeHeap(
395     JS_ATOMIC_ADD(&rt->
396     JS_ATOMIC_ADD(&comp
397     if (comp->gcBytes >
398         TriggerCompartm
399
400     return aheader;
401 }
```

Source location to instrument

probe handler



What to do at probe point

probe values

Attributes

Attribute	Type	Description
applicationCache	nsIDOMOfflineResourceList	Get the applic Note: Prior
document	nsIDOMDocument	The documen
frames	nsIDOMWindowCollection	The child win
name	DOMString	Get or set the JavaScript.
parent	nsIDOMWindow	The window's is of a differe does not cros
scrollbars	nsIDOMBarProp	The object th window. This
scrollX	long	The current h "replaceable"
scrollY	long	The current v "replaceable"

Data to record at probe point

images from www.phdcomics.com

jsprobes use case

1. Find an interesting probe point

```
probes.GC_DID_START
```

2. Decide which probe values to use

```
env.currentTimeMS,
runtime.heapSize
```

3. Write a probe handler for that point

```
pendingGC = [env.currentTimeMS, 0,
                    runtime.heapSize, 0];
```

jsprobes use case (2)

4. Write a matching handler to `GC_WILL_END`:

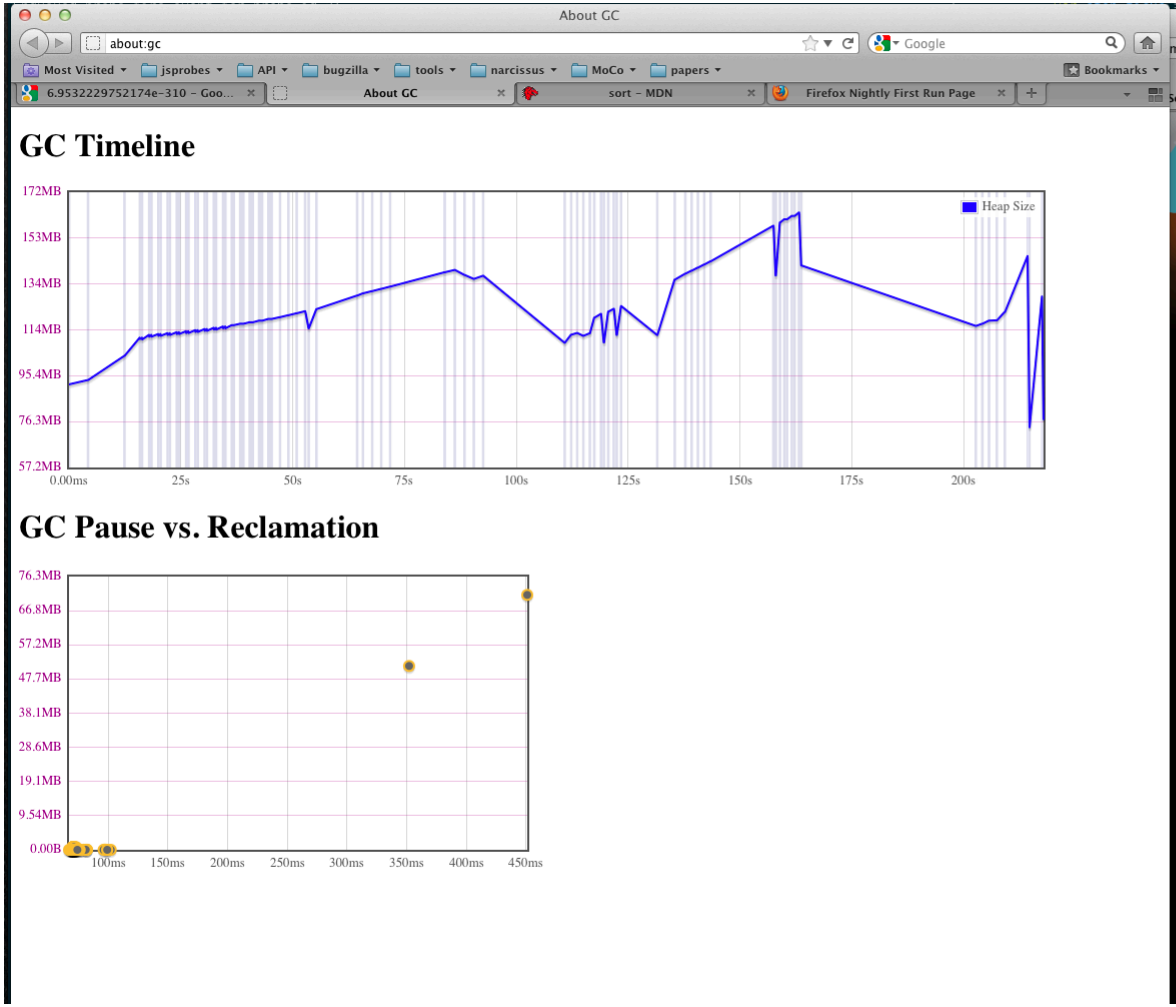
```
pendingGC[1] = env.currentTimeMS;
pendingGC[3] = runtime.heapSize;
data.push(pendingGC);
```

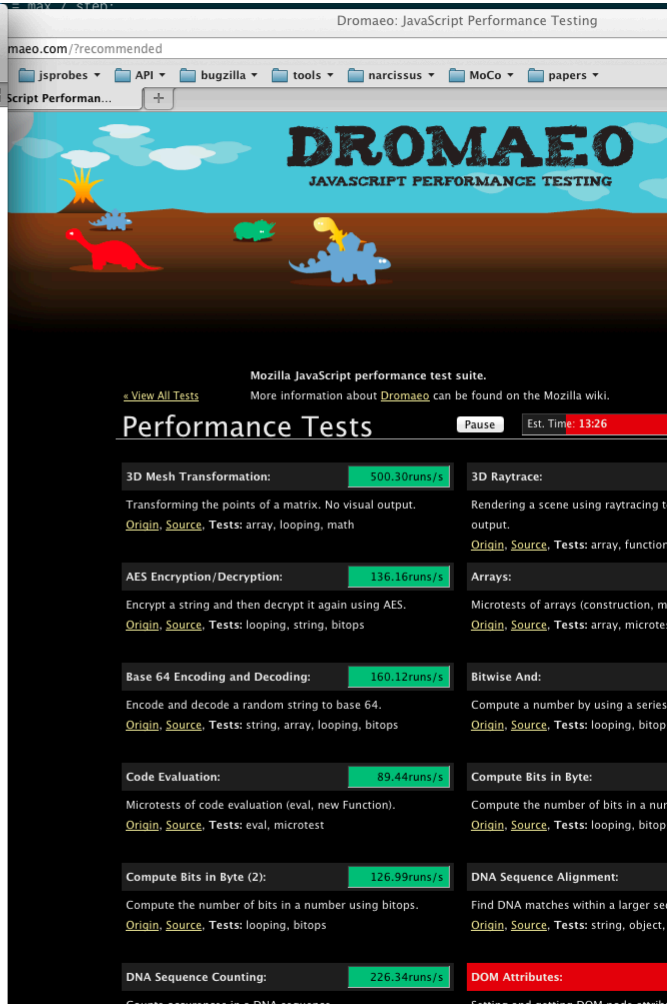
5. Register handlers with the probes service.

6. Periodically fetch data and do something, such as aggregate, graph, or report it.

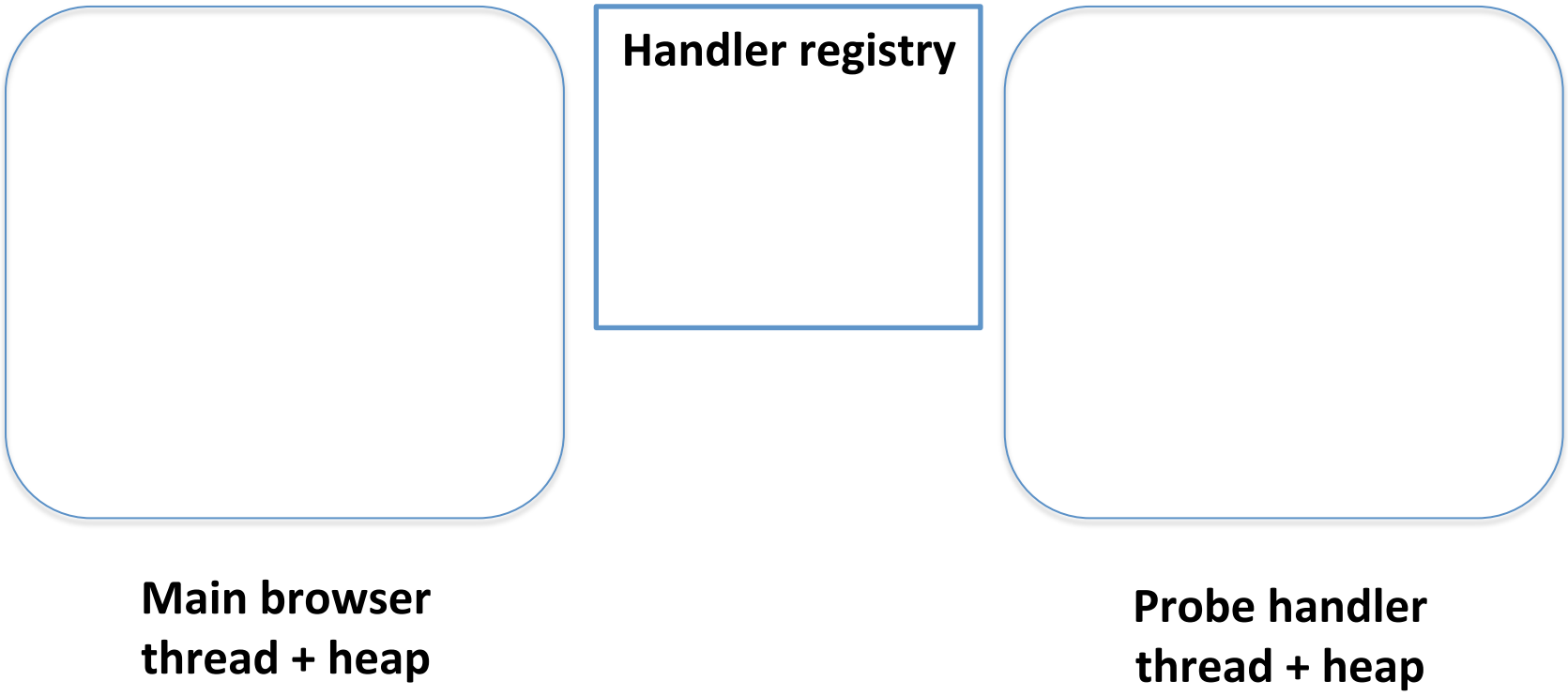
Demo!

(source available at https://bitbucket.org/burg/aboutgc)

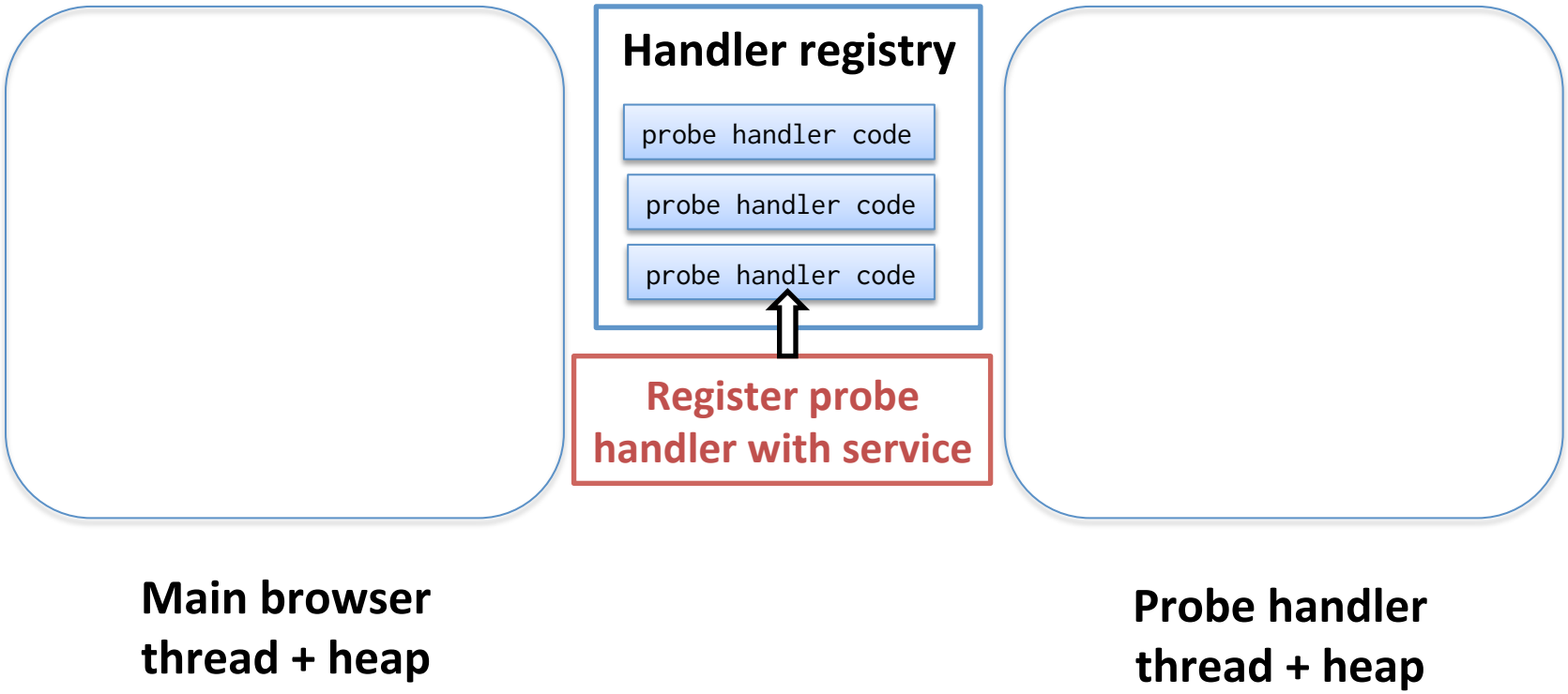




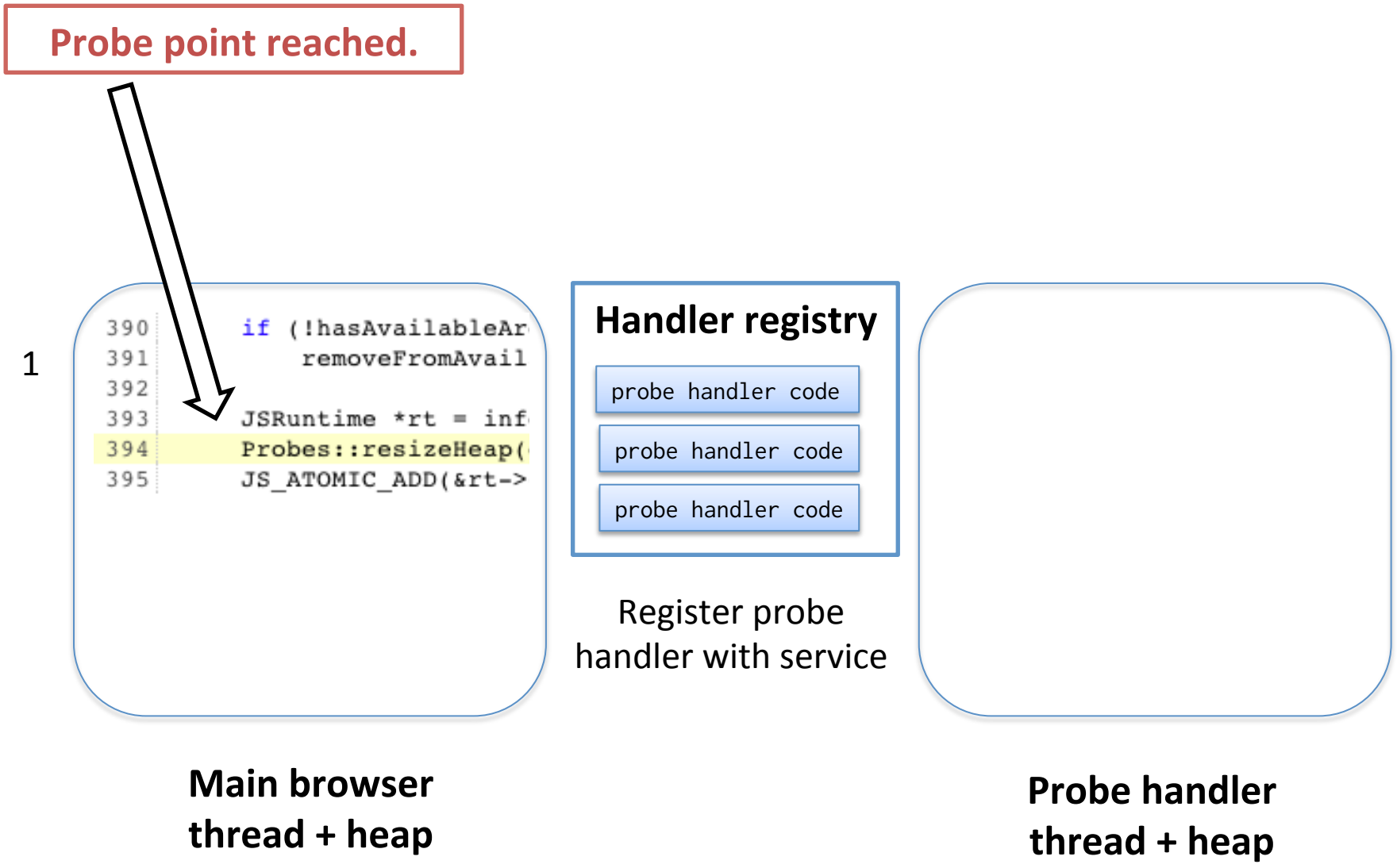
jsprobes architecture



jsprobes architecture



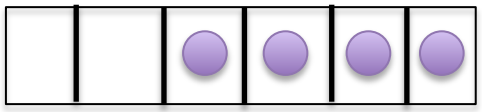
jsprobes architecture



jsprobes architecture

Probe point reached.

Data gathered, serialized, and enqueued



```
390      if (!hasAvailableAr
391          removeFromAvail
392
393      JSRuntime *rt = inf
394      Probes::resizeHeap(
395      JS_ATOMIC_ADD(&rt->
```

Handler registry

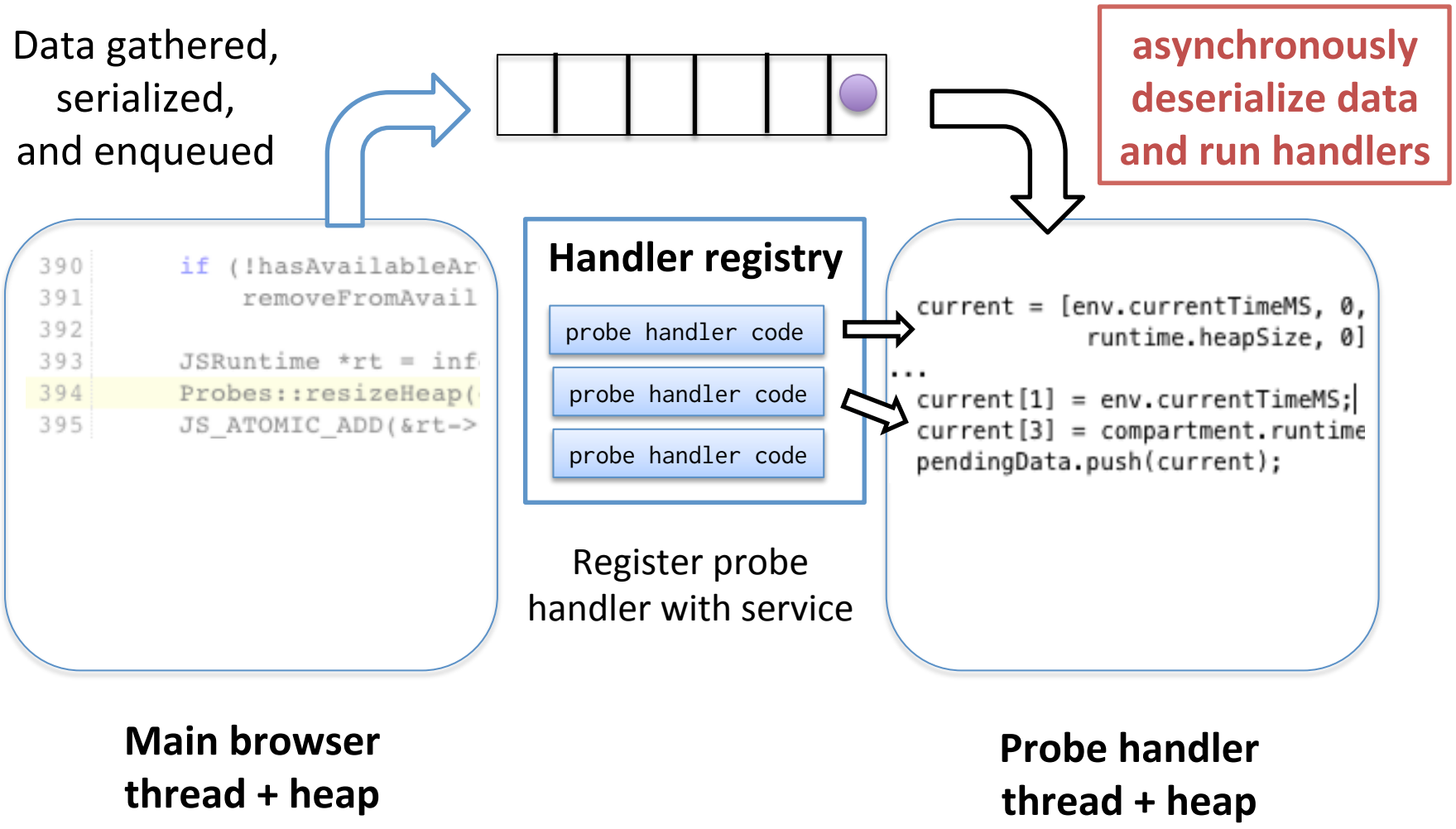
- probe handler code
- probe handler code
- probe handler code

Register probe handler with service

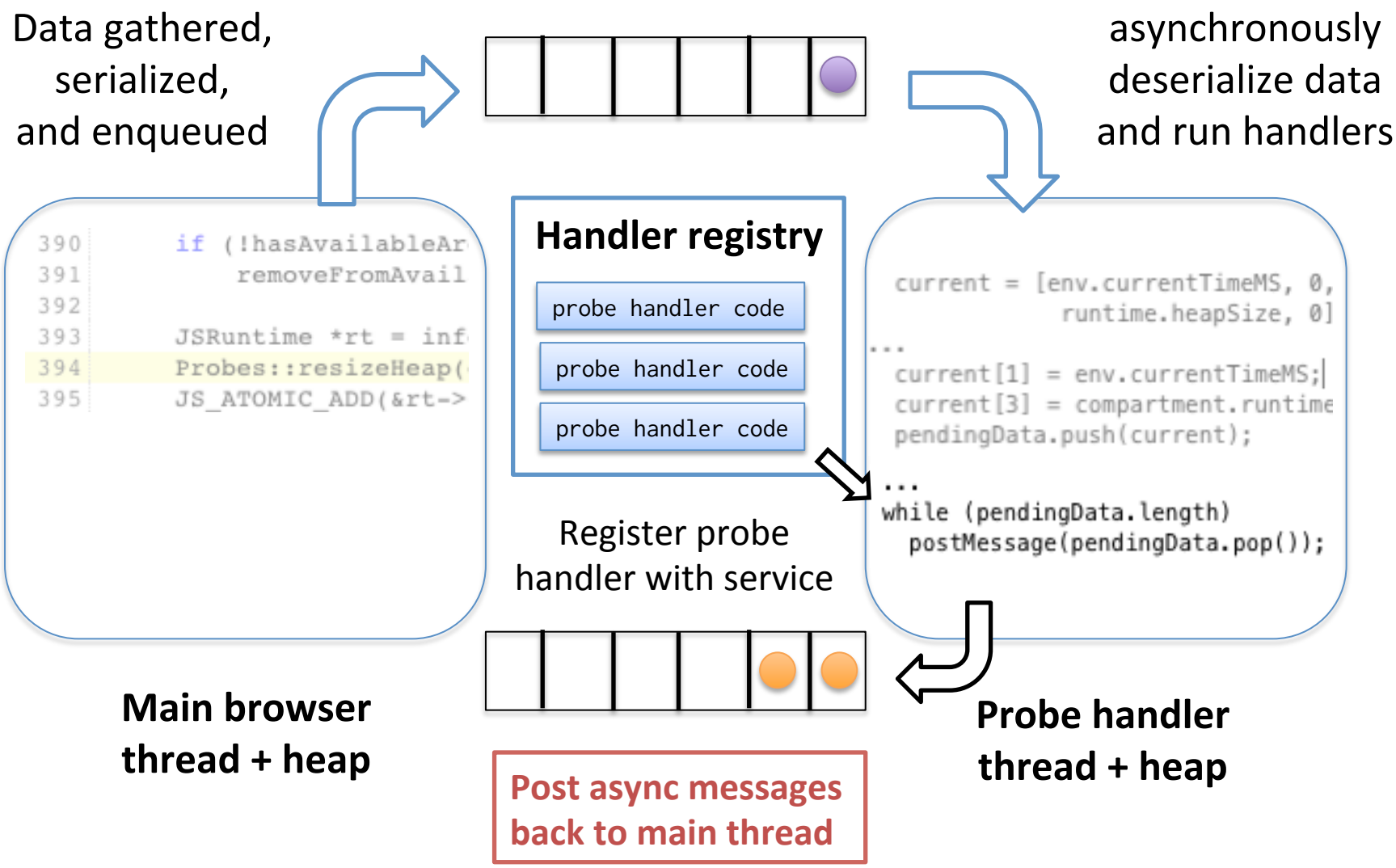
Main browser thread + heap

Probe handler thread + heap

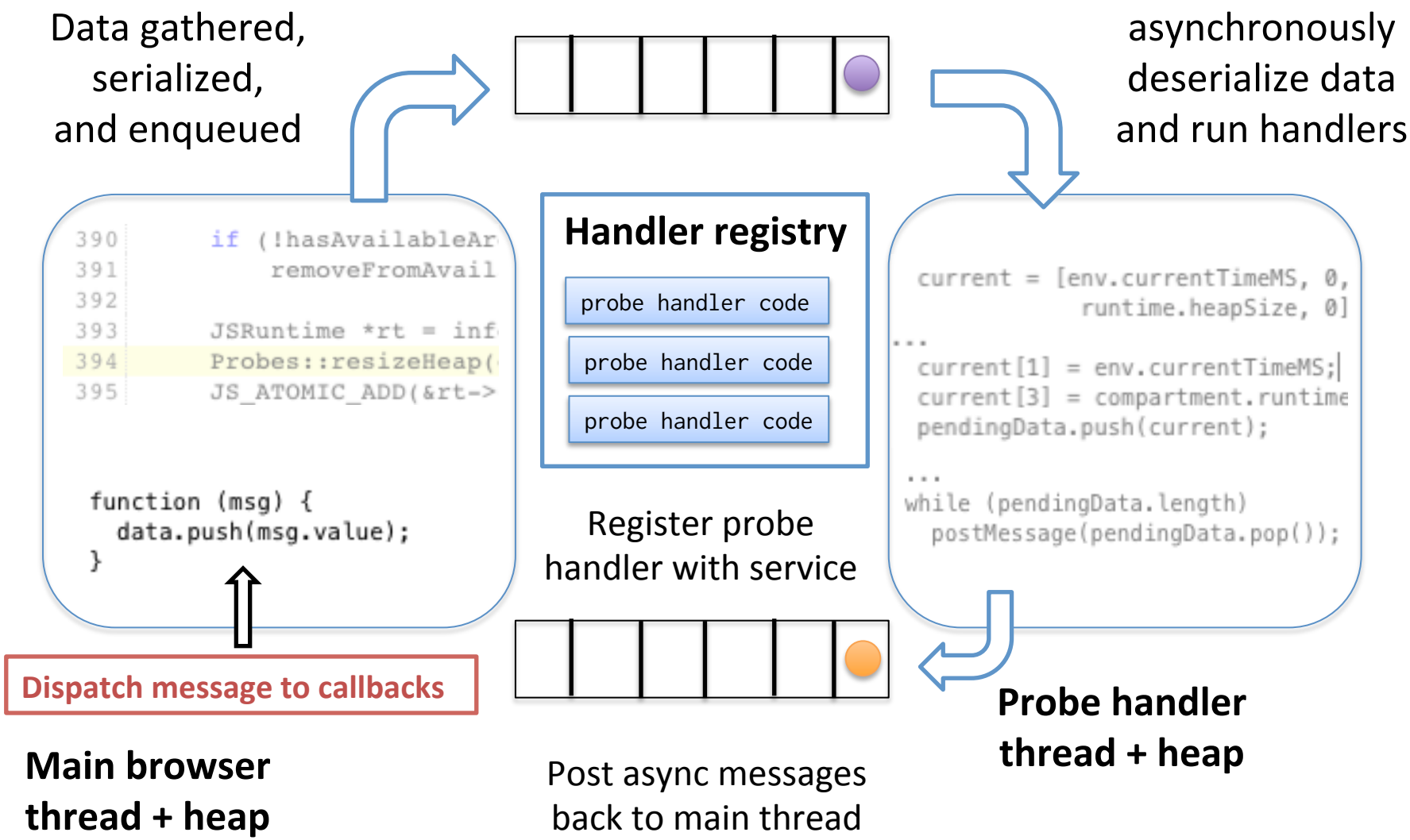
jsprobes architecture



jsprobes architecture



jsprobes architecture



Architecture implications

- Probe points can fire at times unsafe for JS
- Probe handlers have read-only access*
- Handler must specify what data to collect
- Probe data must be representable in JS
- Probe data must be be serializable**

* Side-effects would complicate reasoning when multiple handlers are registered for the same probe point

** Probe data is marshalled using the HTML 5 structured cloning algorithm. This can be extended to support new data types.

jsprobes: current status

- Cross-platform/architecture
- Low/no performance overhead (TODO)
- Shareable/distributable (TODO)
- Runtime flexibility
- Familiar programming model
- Cross-language/cross-component

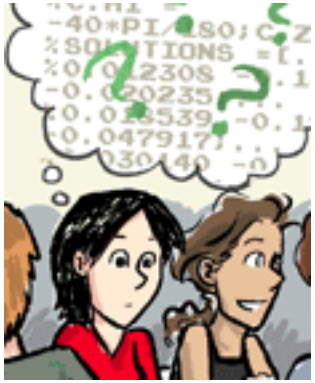
Current implementation available at https://bitbucket.org/burg/jsprobes-patches

Let's fill in the research template...

1

"Do websites have a typical heap size?"

2



Use jsprobes to make an addon that measures per-site heap size

3

Implement better heap size heuristics based on real data

images from www.phdcomics.com

Future work

- More sophisticated implementation
  - No "probe effect" when probes inactive
  - Low performance impact when active
- Add probe points to more components
- Expose more types of data to probe handlers

Brian Burg – University of Washington
burg@cs.uw.edu www.brrian.net
www.twitter.com/brrian
https://bitbucket.org/burg/
http://brrian.tumblr.com

mozilla research