On the (Same) Origin of Scripties

Evolving a new security policy for the web

Jasvir Nagra and Mike Samuel

Google, Inc.
{jasvir,msamuel}@google.com

OWASP
23 June 2010

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

Argument in a nutshell

Social Networks compose web applications from small apps

This breaks the **same origin** policy

A network that gives developers the most **authority** will **grow**.

The bigger networks can neither **trust** nor **police** developers.

And they can't **predict** all the threats they will face.

Virtualization lets you promiscuously grant authority to grow.

And **dial it back** later, after you understand threats.

Without breaking **APIs**.

What is Authority?

Authority : ability to influence or exercise power

In browsers, web applications can:

- initiate network requests
- display a user interface
- observe user activity
- ...

Most of this authority is available "ambiently".

Ambient Authority: authority available regardless of how a web application was loaded

Ambient Authority in Browsers

Irrespective of origin

top.location = ...
Content-Disposition: attachment
window.open(...)
window.getComputedStyle(...)
<img src=http://... onerror=...>
<form action=http://...>
<script src=http://...>
<html>...</html>
xhr.open(..., ..., false)
<iframe><input type=file></iframe>

In same origin

document.cookie
document.body
xhr.open(...)
body.onkeypress
Object.prototype.toString = ...
window.forms
<input autocomplete=yes>
window.createEventObject

Ambient Authority in Browsers

Irrespective of origin

top.location = ...	redirect any reachable frame
Content-Disposition: attachment	initiate a download
window.open(...)	create a window (modulo user interaction)
window.getComputedStyle(...)	sniff browser history
<img src=http://... onerror=...>	scan local network
<form action=http://...>	GET or POST to any domain with cookies
<script src=http://...>	load code from any source
<html>...</html>	impersonate another website
xhr.open(..., ..., false)	deny service
<iframe><input type=file></iframe>	present file upload controls

In same origin

document.cookie	modify and read cookies
document.body	modify and inspect the entire UI
xhr.open(...)	read result of GET or POST to same origin
body.onkeypress	intercept user events
Object.prototype.toString = ...	change behavior of language intrinsics
window.forms	read forms before submission
<input autocomplete=yes>	present an input that might be autofilled
window.createEventObject	spoof user events

What's a Social Network To Do?

	Introduces New Tools	**Improves Existing Tools**
Large Audience (All Web Devs)	Limited. Slow to take hold. E.g. window.toStaticHtml	Good, but doesn't address zero-days. E.g. PHP magic quotes.
Small Audience (Library Authors & Security folk)	Good, but targets very particular attacks. E.g. Uniform Messaging	*The sweet spot*. A small group can address emerging threats. E.g. native JSON

Virtualization

Caja, browser virtualization. No plugins required.

A layer of software between the real authority and the invoker.

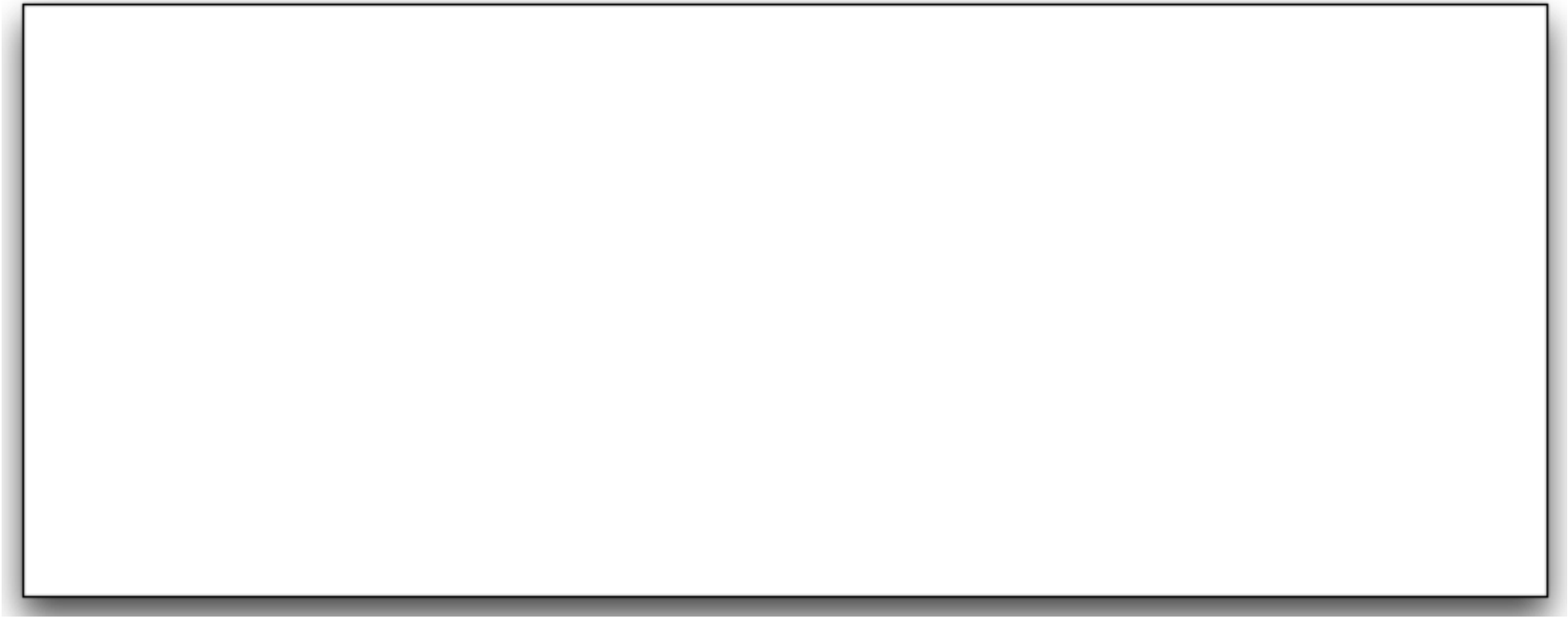
When a threat emerges, tame the APIs involved.

Preserve APIs, but bound authority.

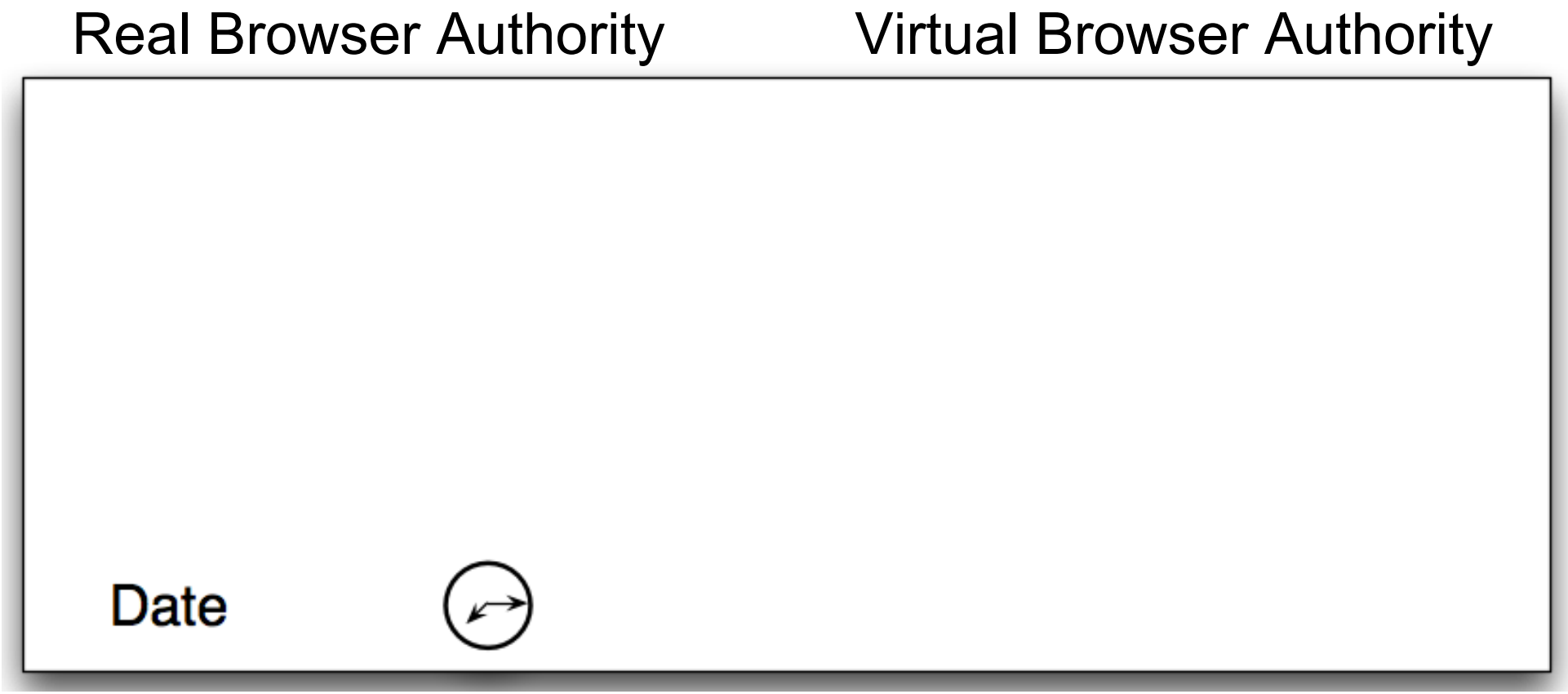
Dealing with Ambient APIs

Real Browser Authority

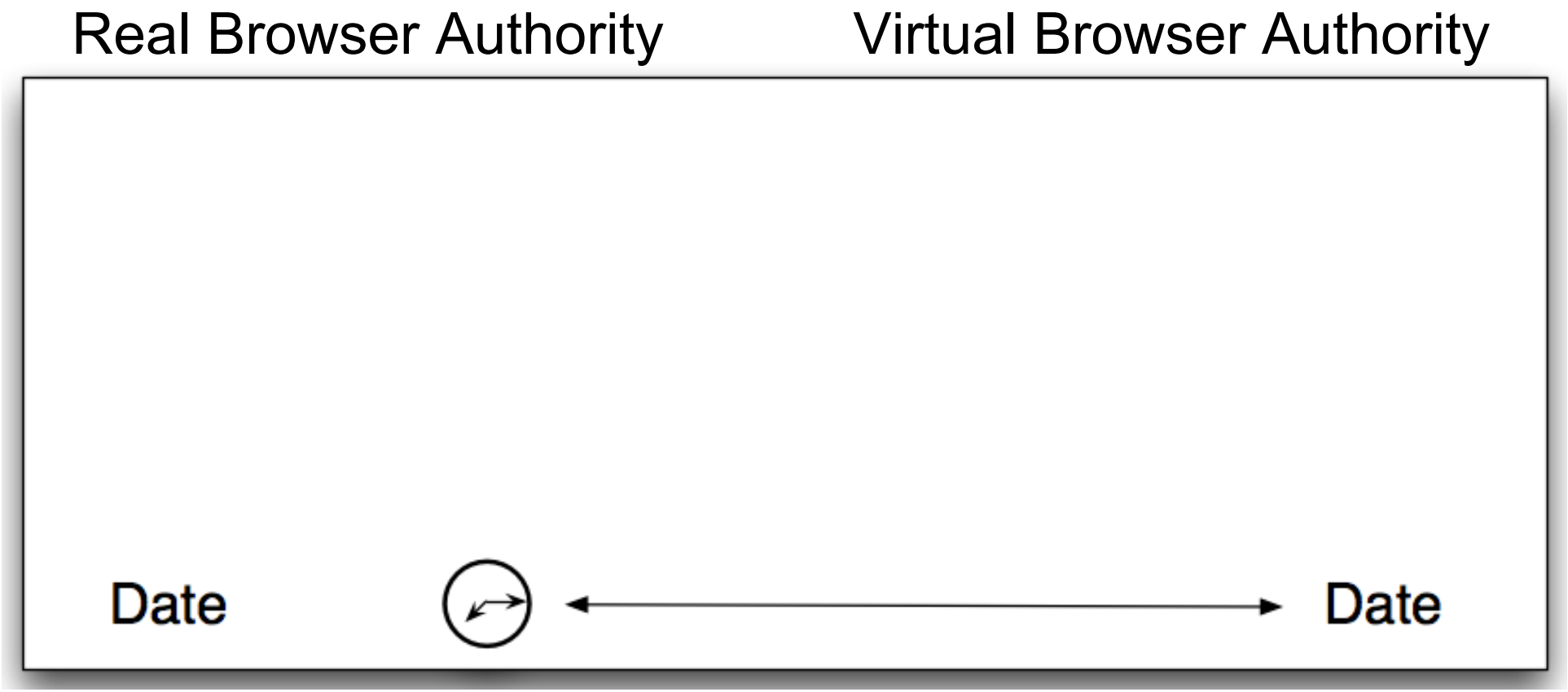
Virtual Browser Authority



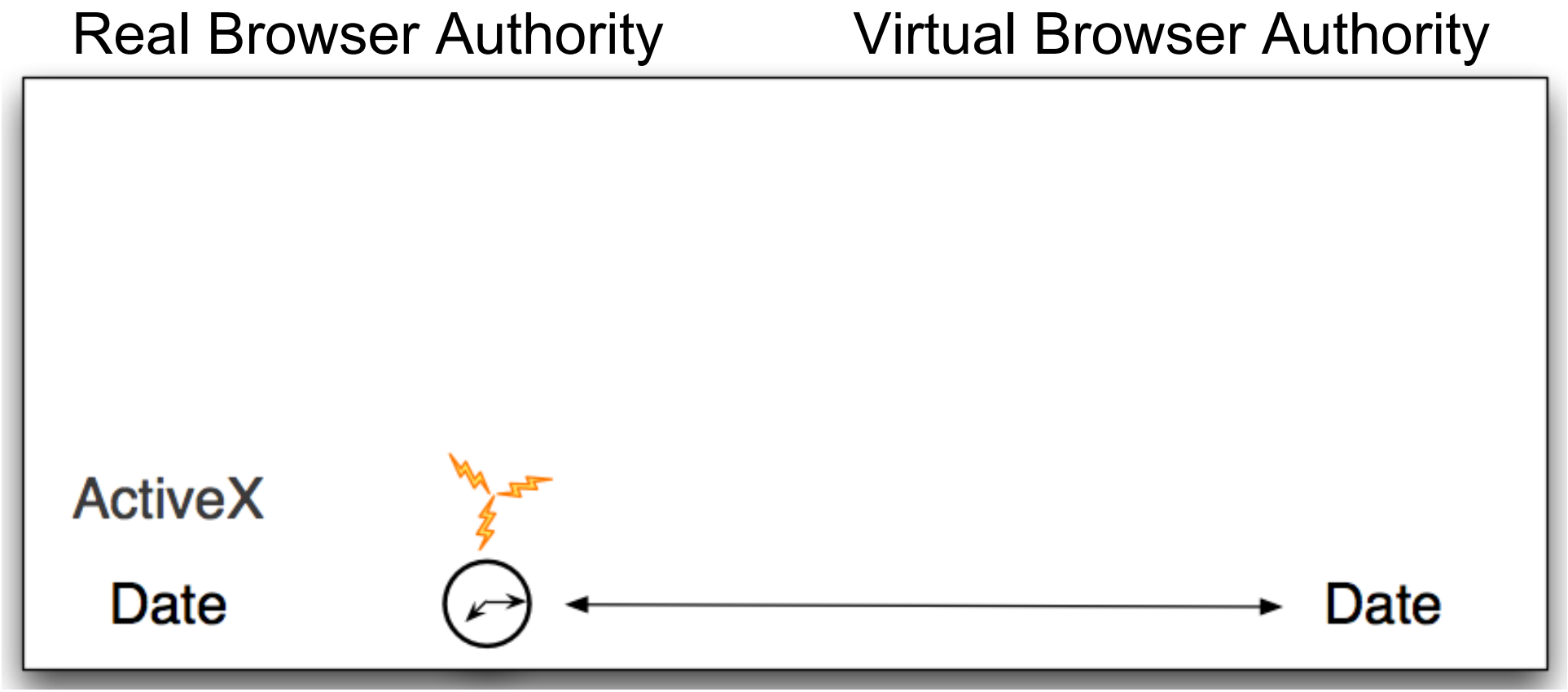
Dealing with Ambient APIs



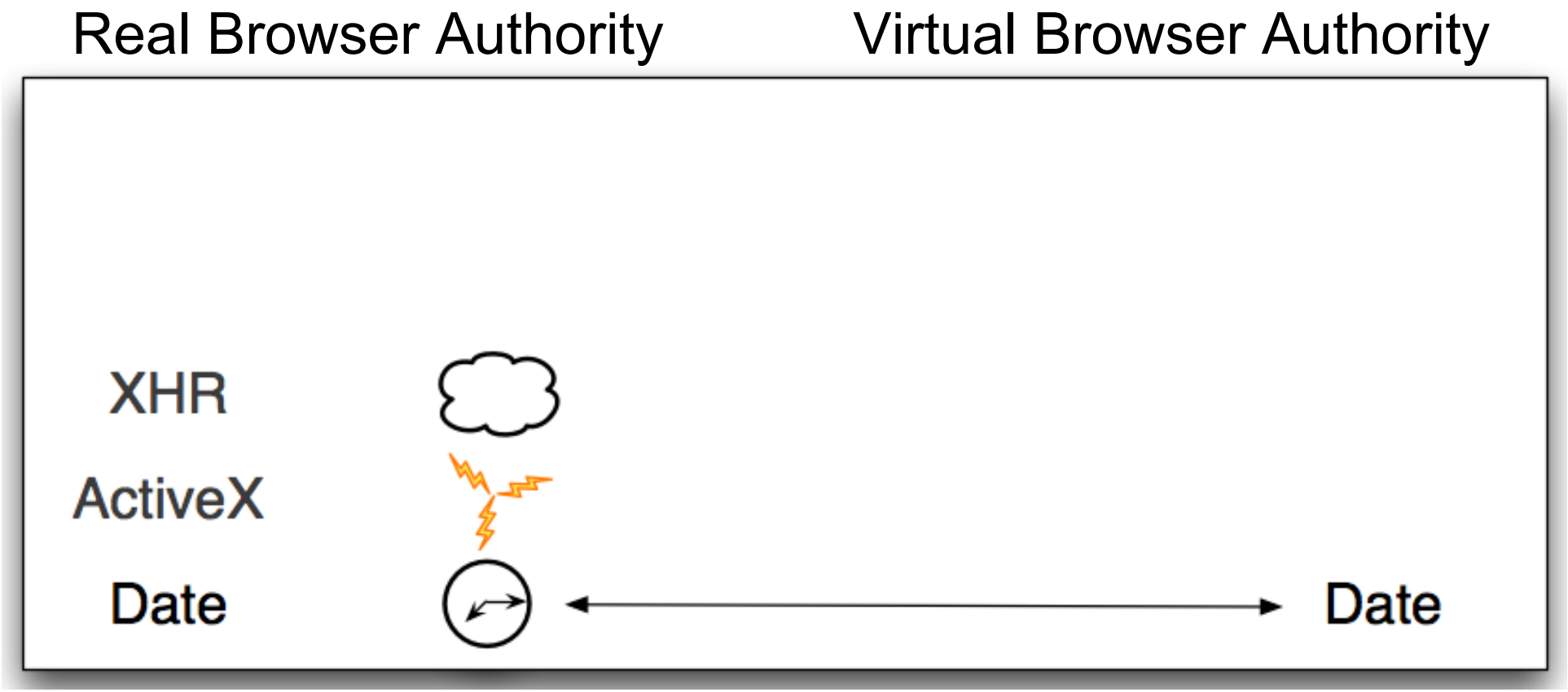
Dealing with Ambient APIs



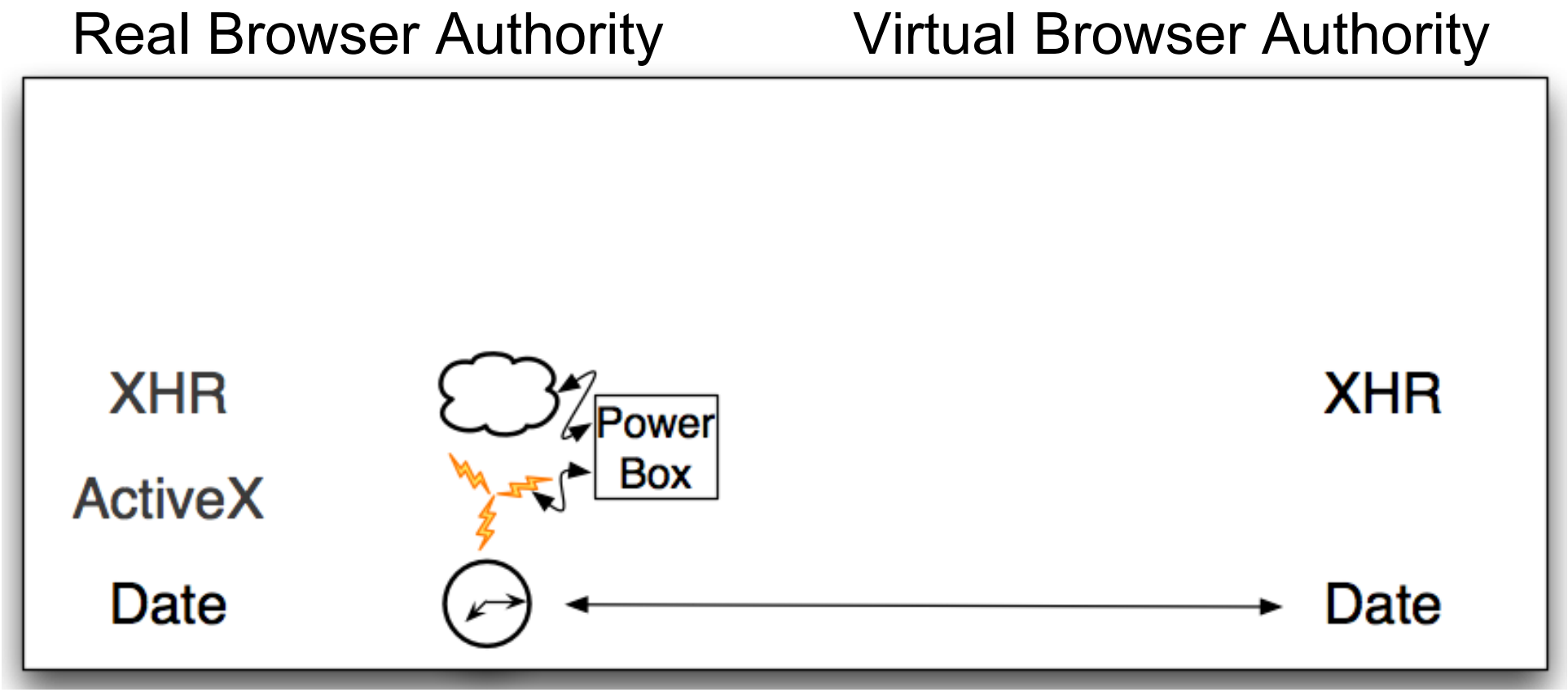
Dealing with Ambient APIs



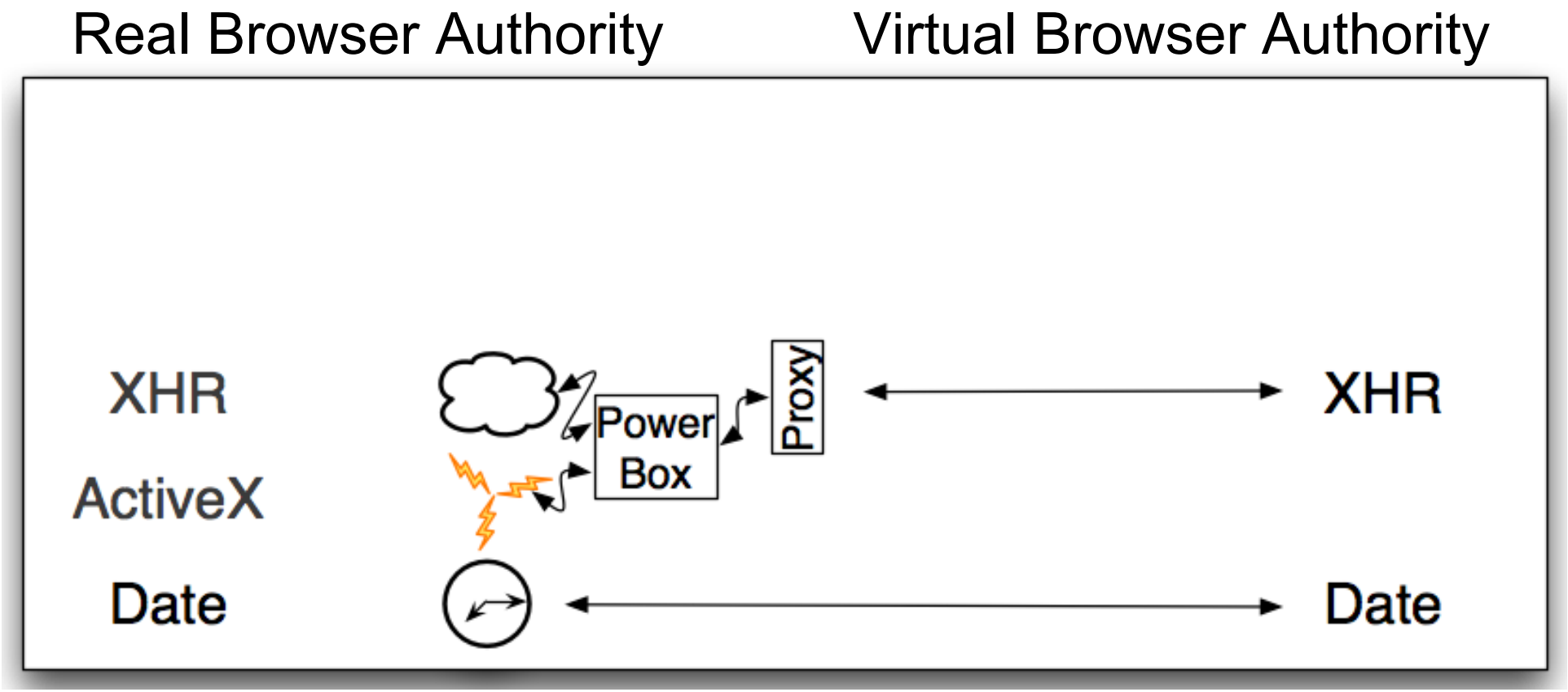
Dealing with Ambient APIs



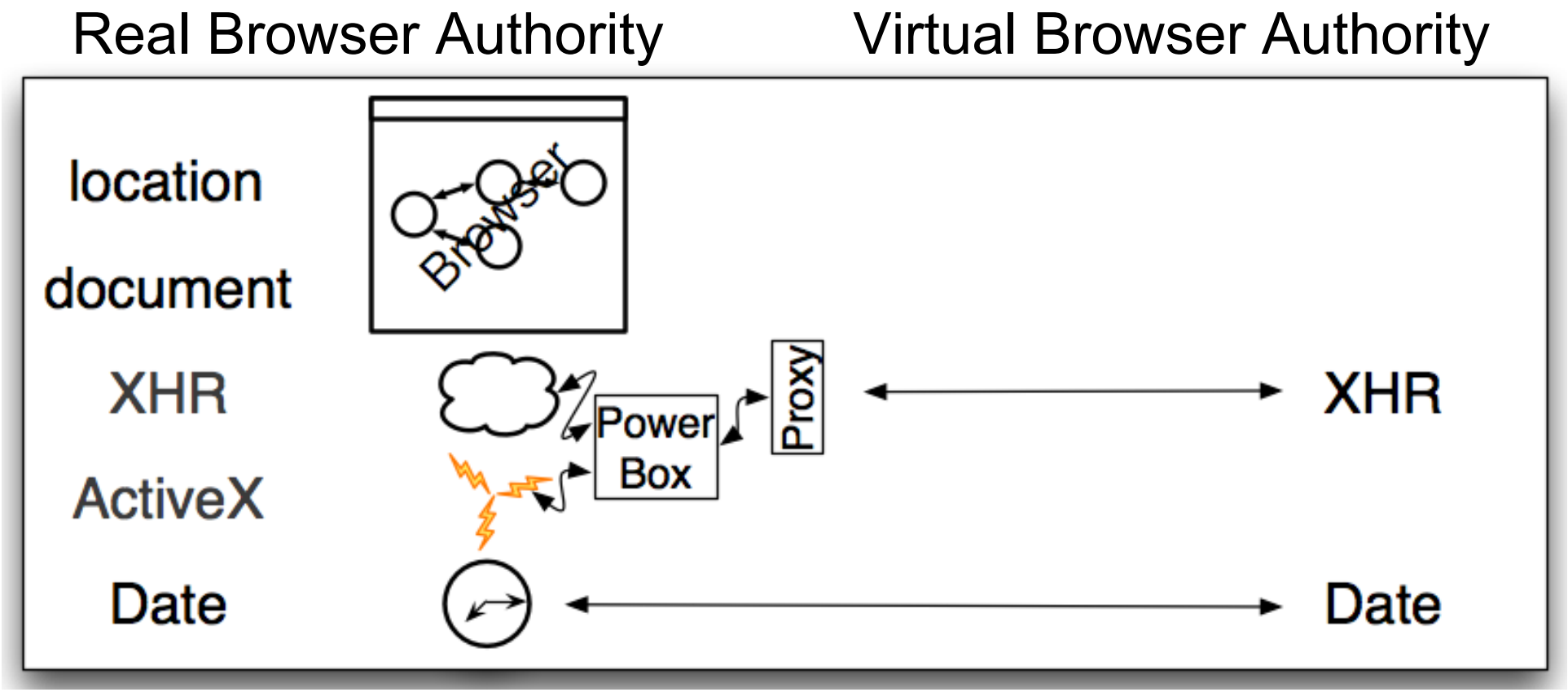
What do we want to Protect?



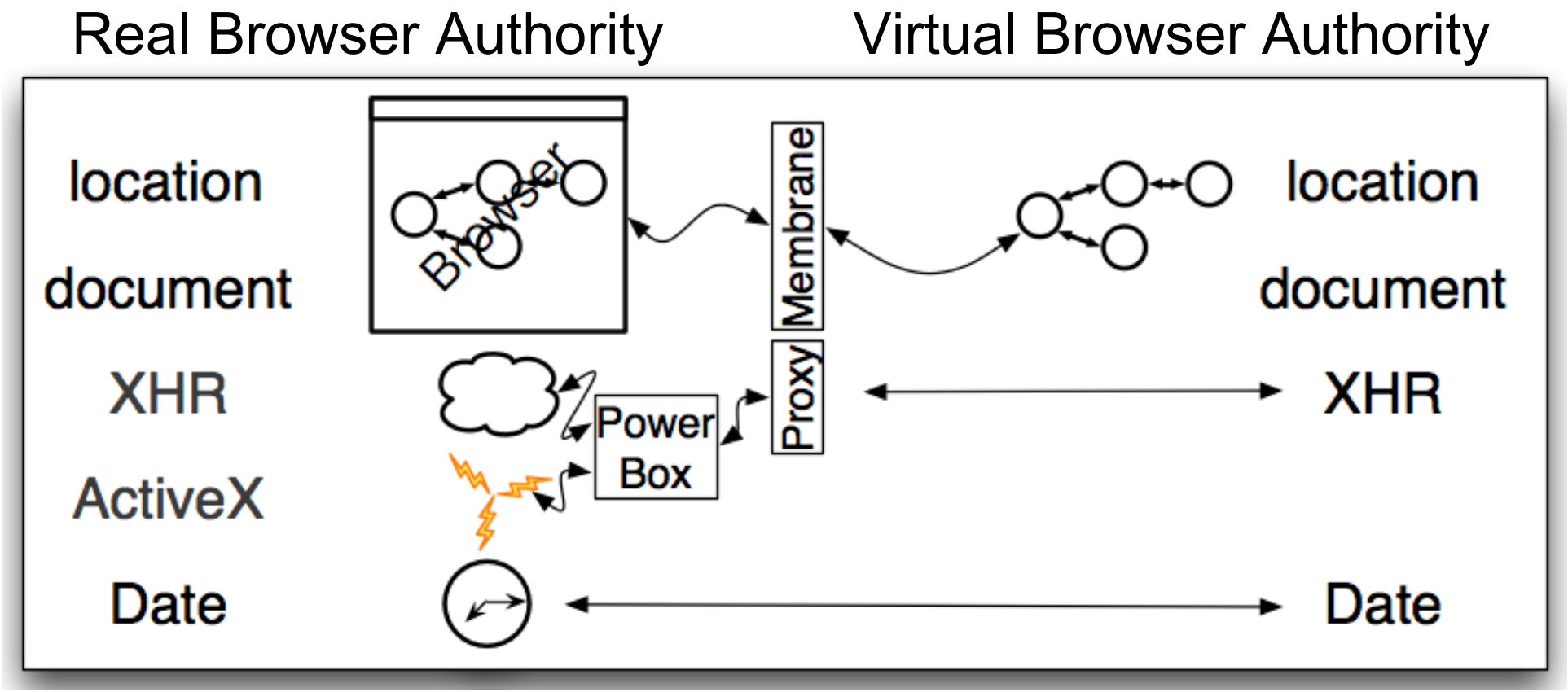
Dealing with Ambient APIs



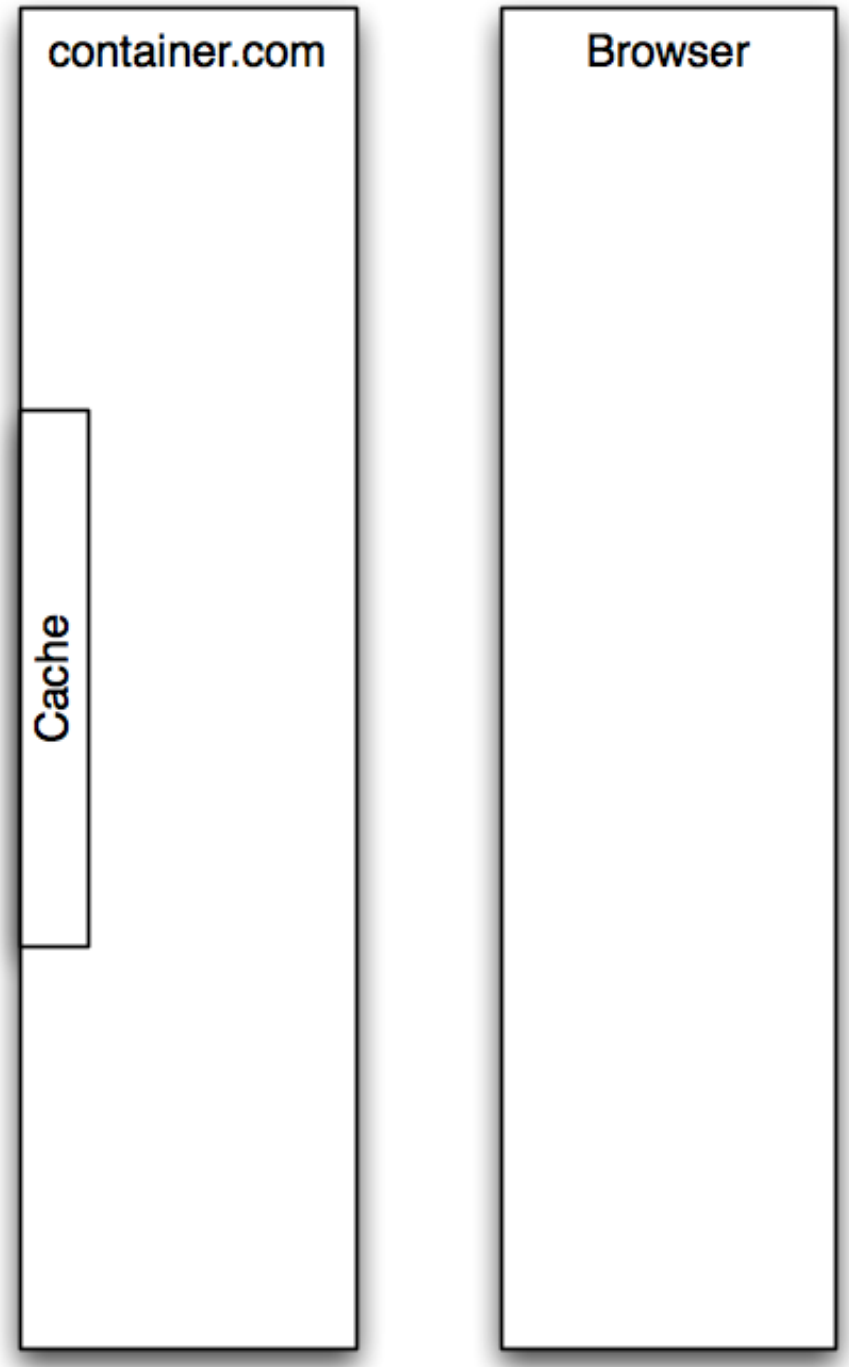
Dealing with Ambient APIs



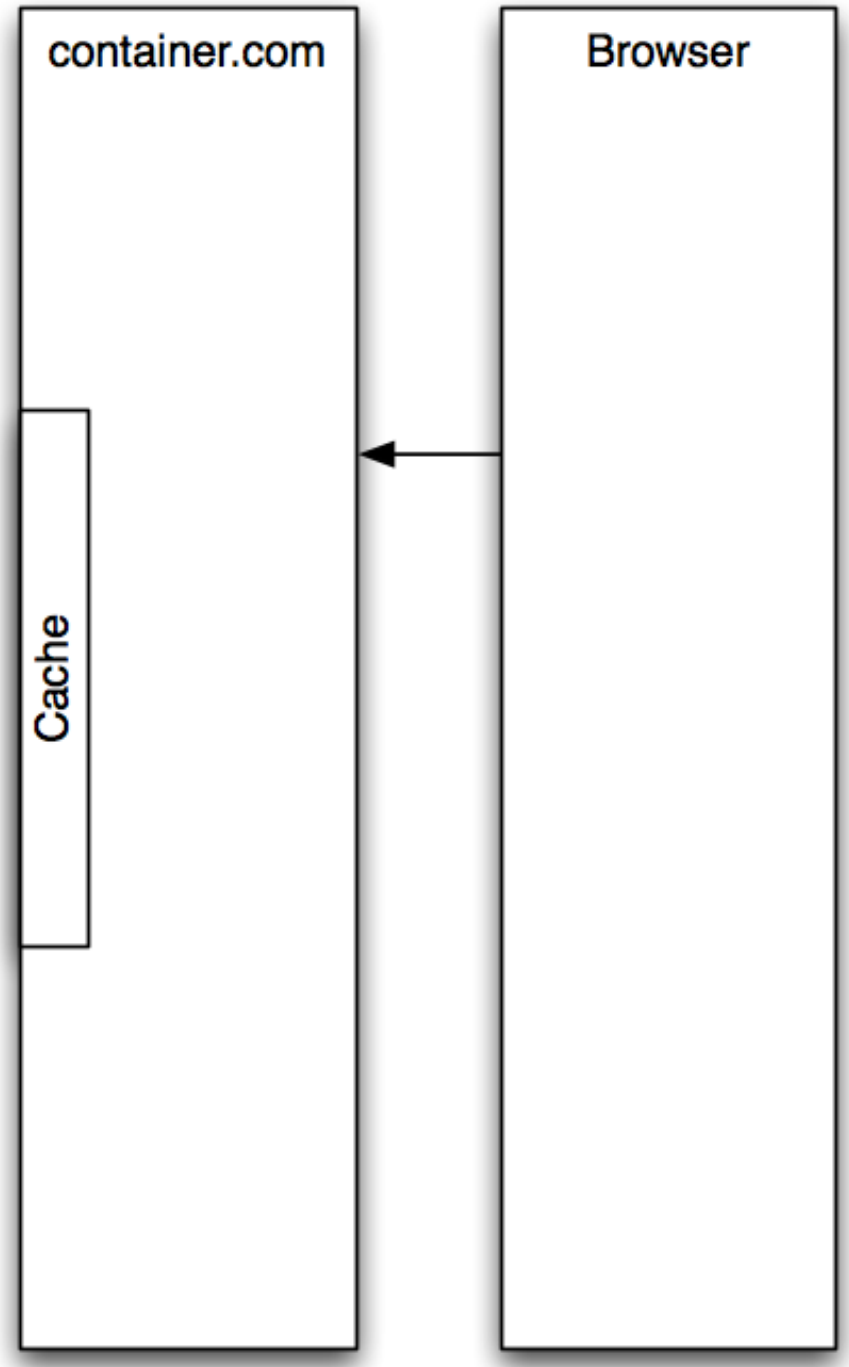
Dealing with Ambient APIs



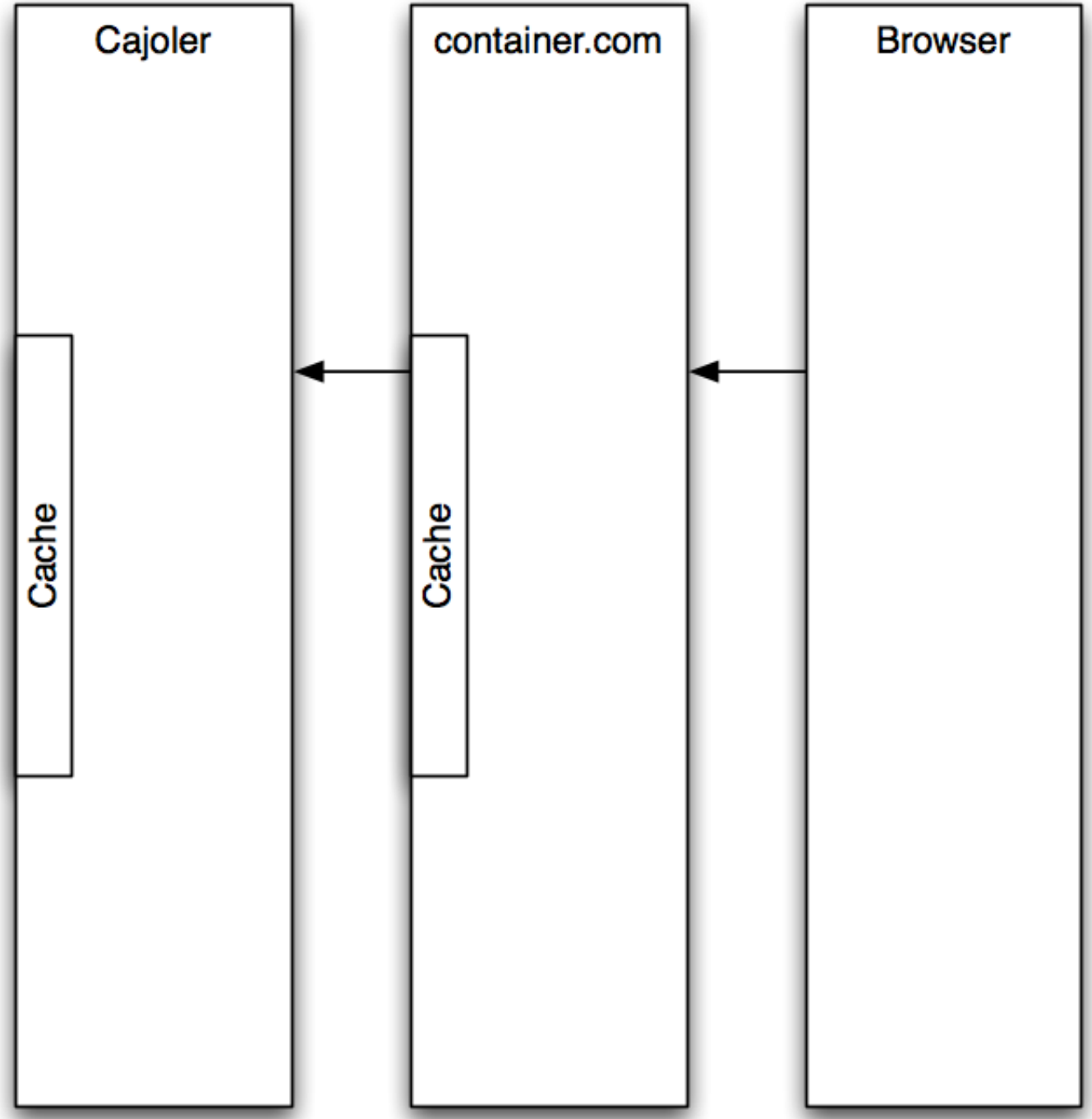
Architecture



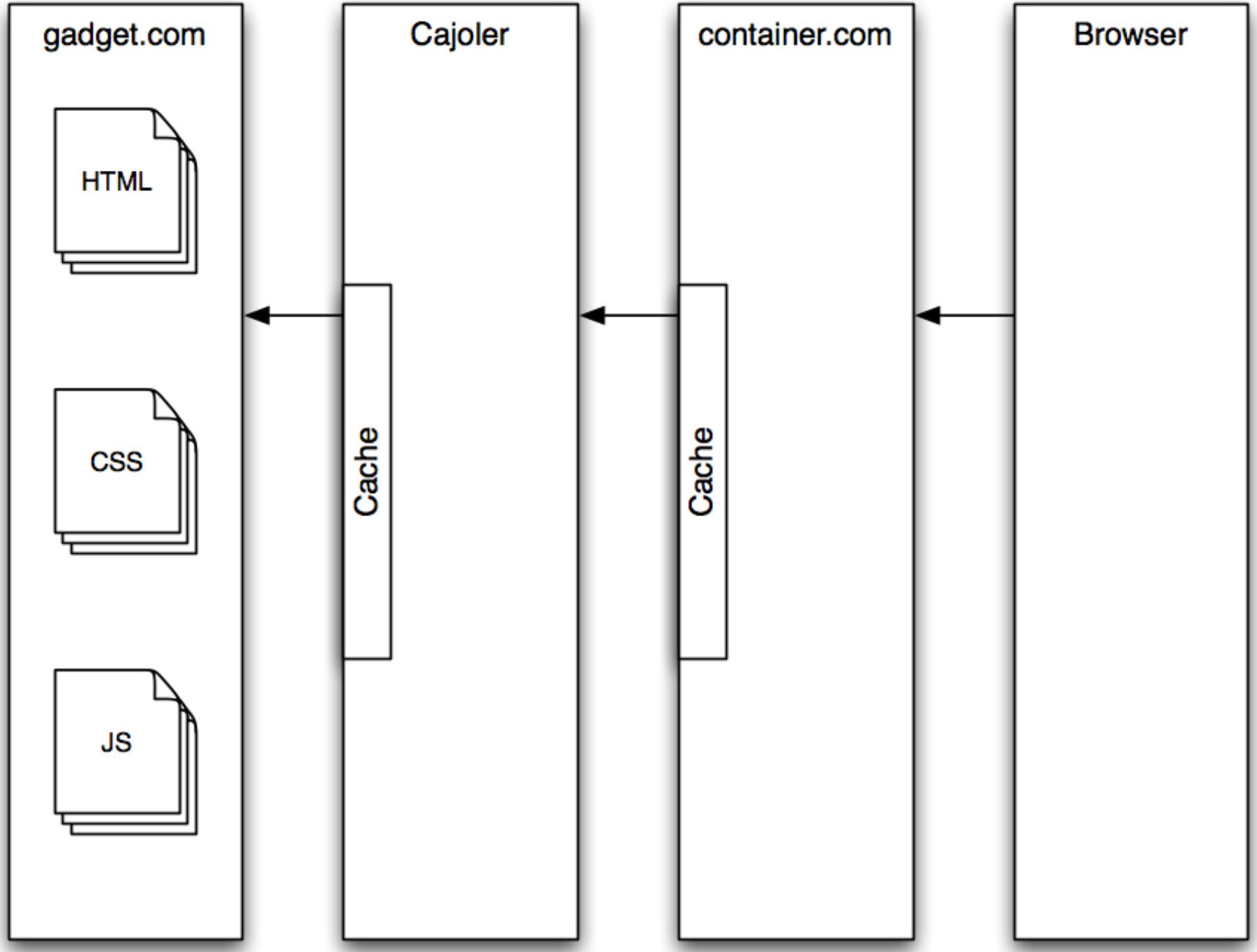
Architecture



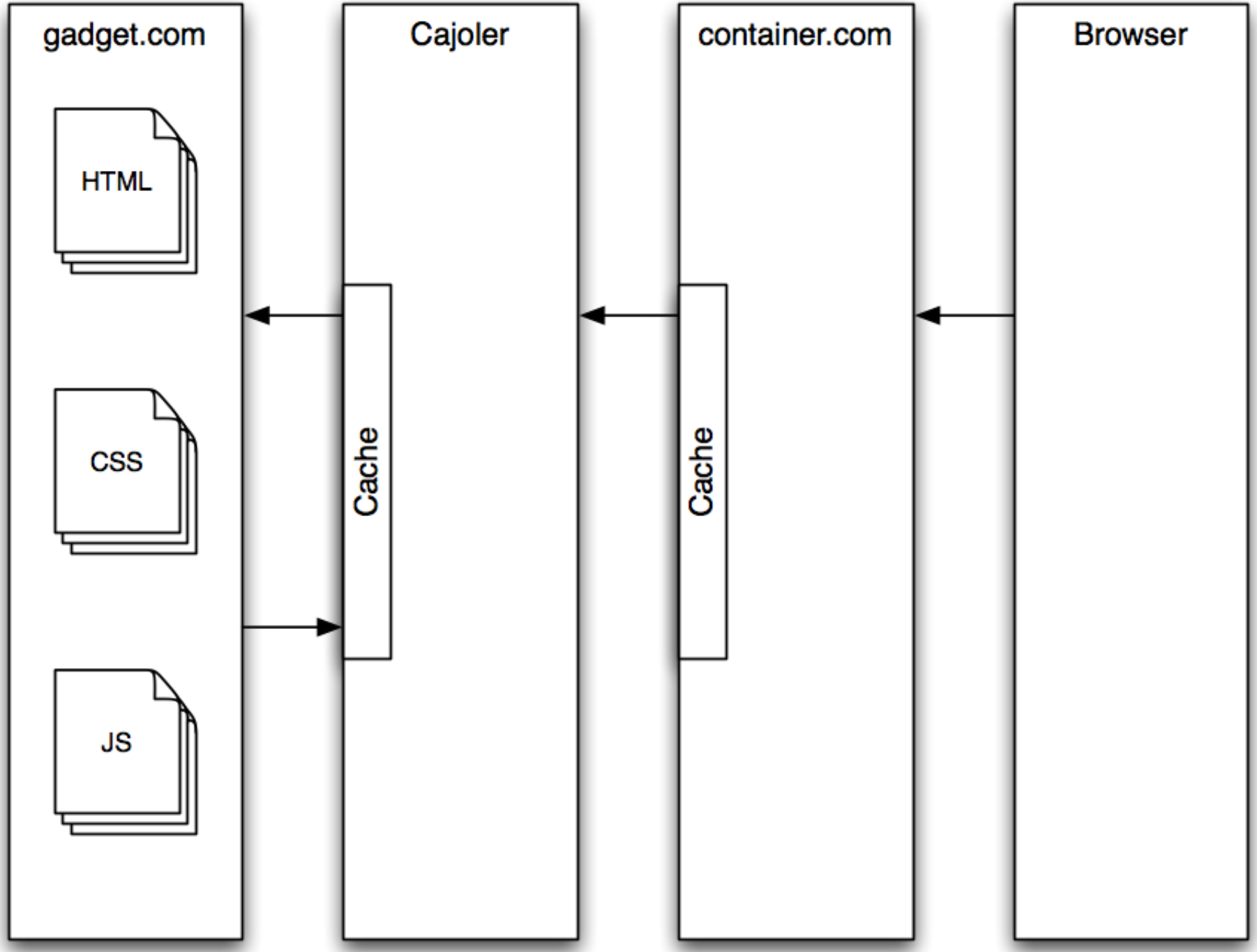
Architecture



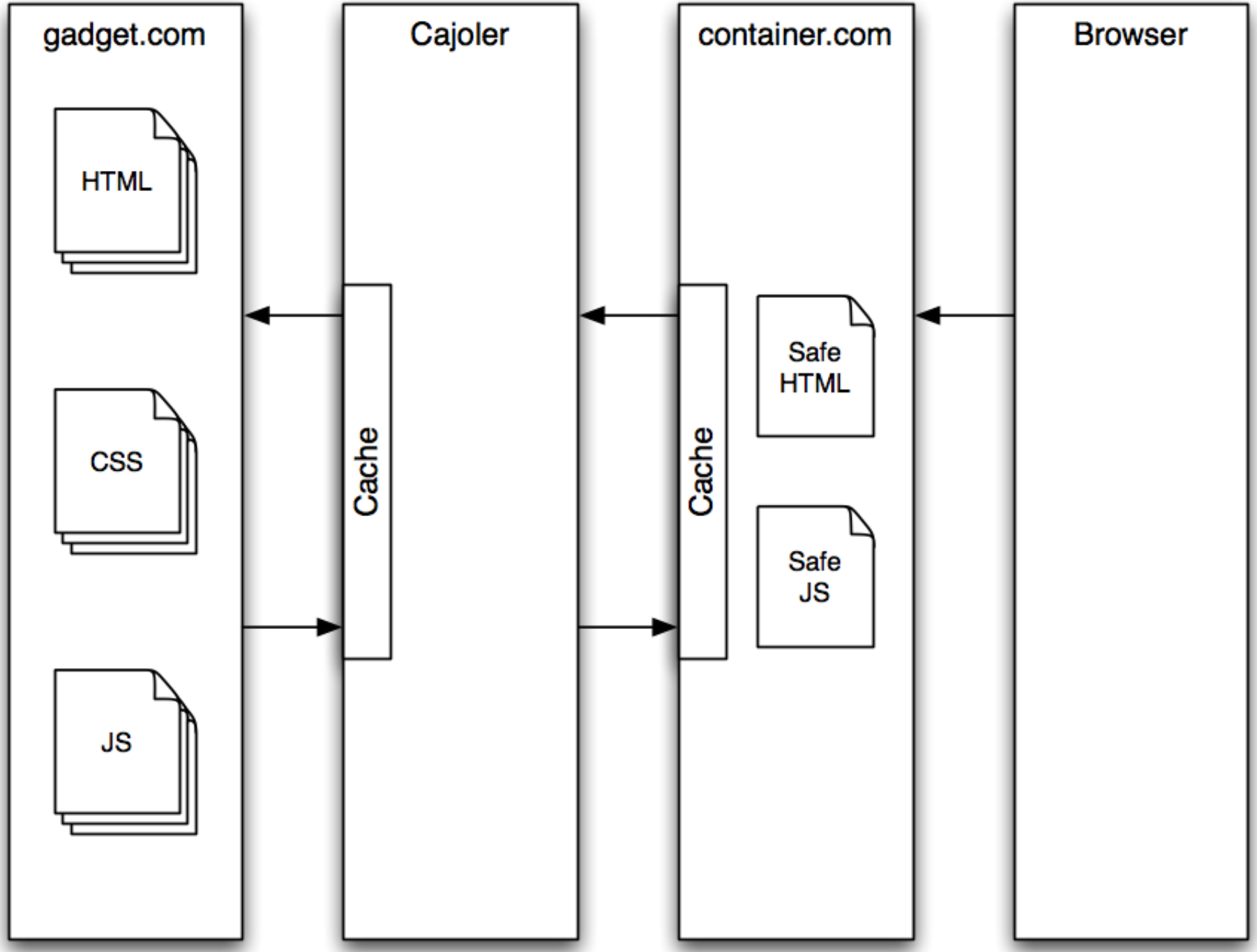
Architecture



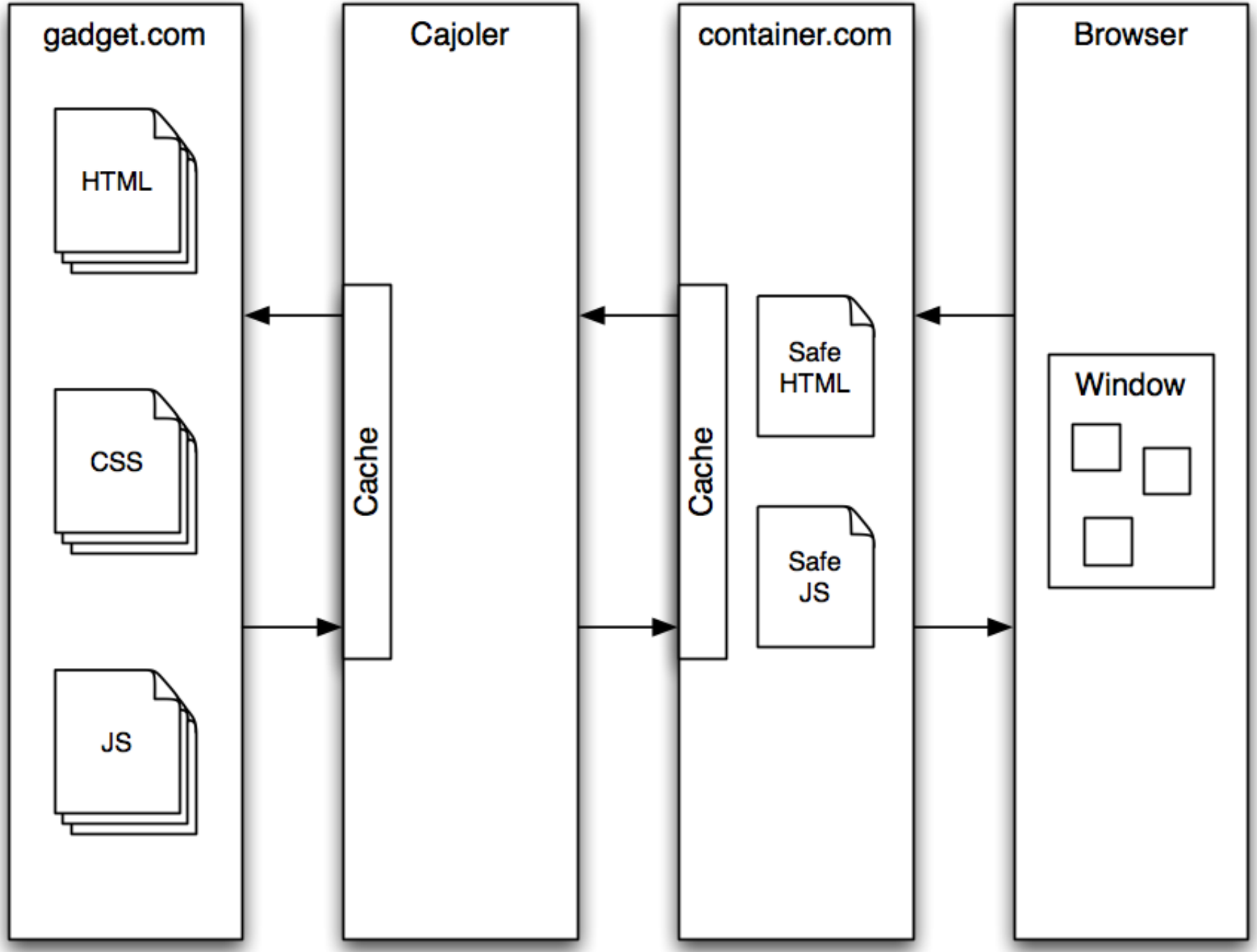
Architecture



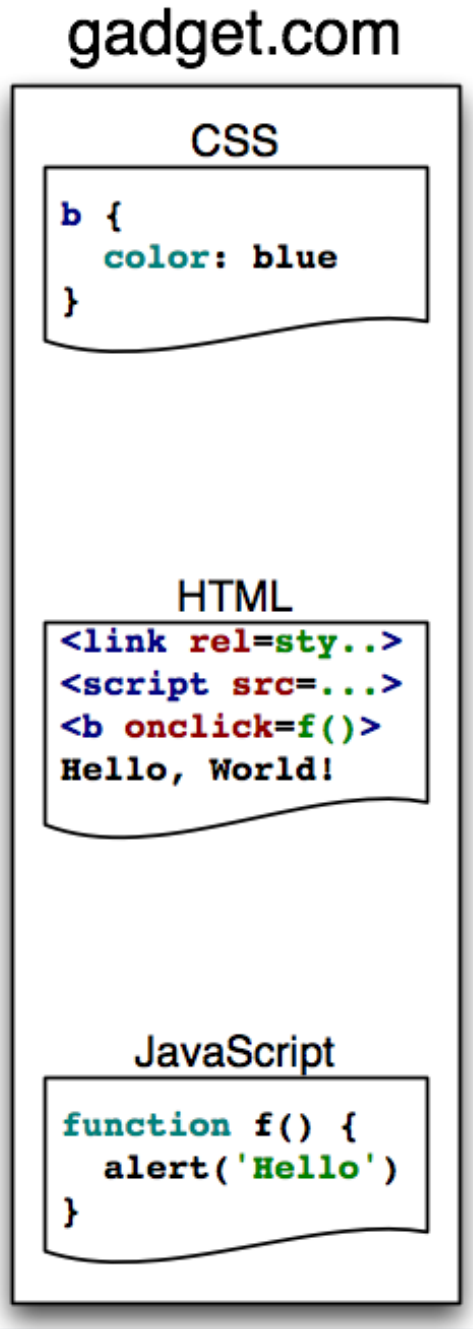
Architecture



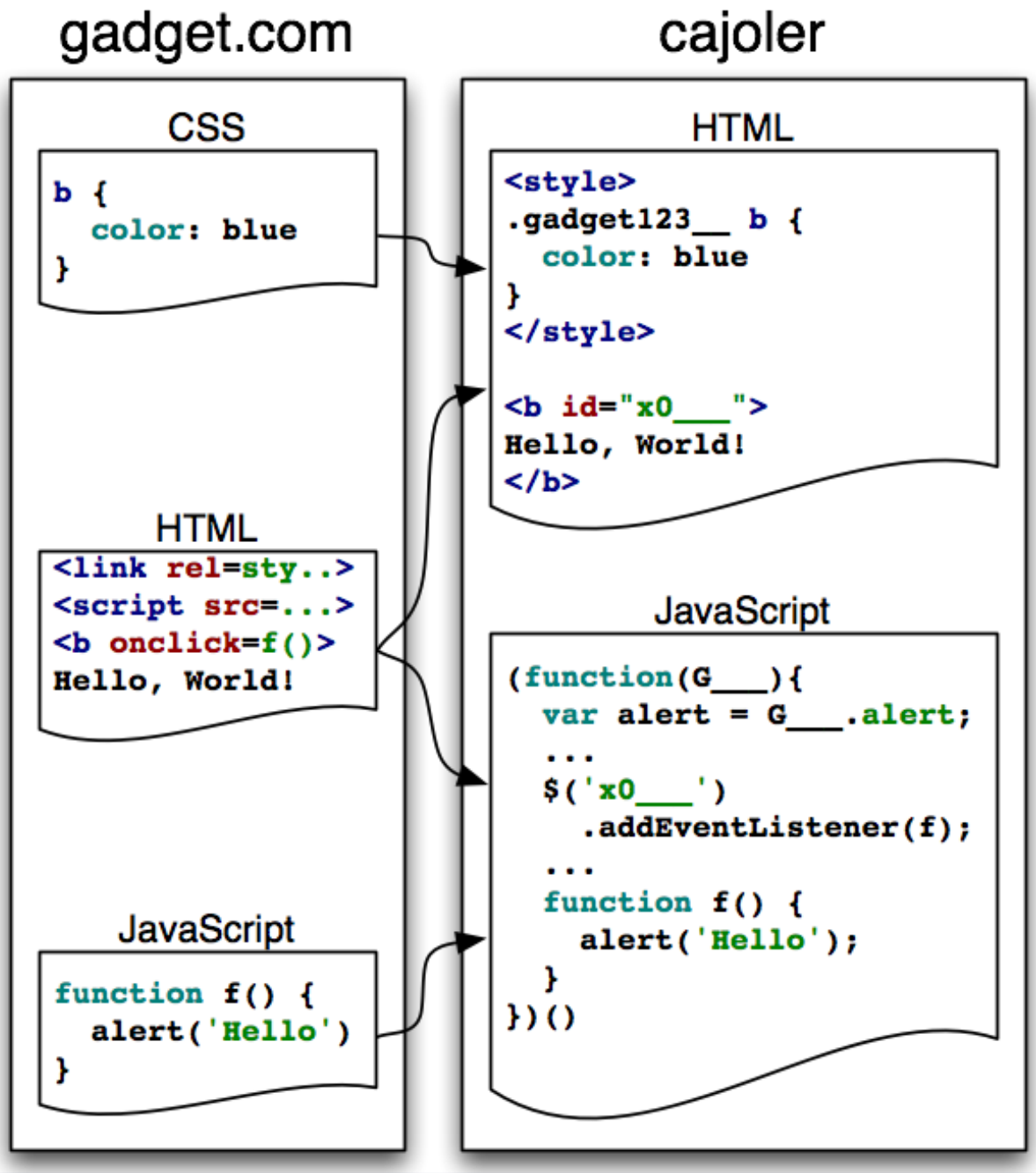
Architecture



Example App


gadget.com
CSS
b {
color: blue
}
HTML
<link rel=sty..>
<script src=...>
<b onclick=f()>
Hello, World!
JavaScript
function f() {
alert('Hello')
}

Example App

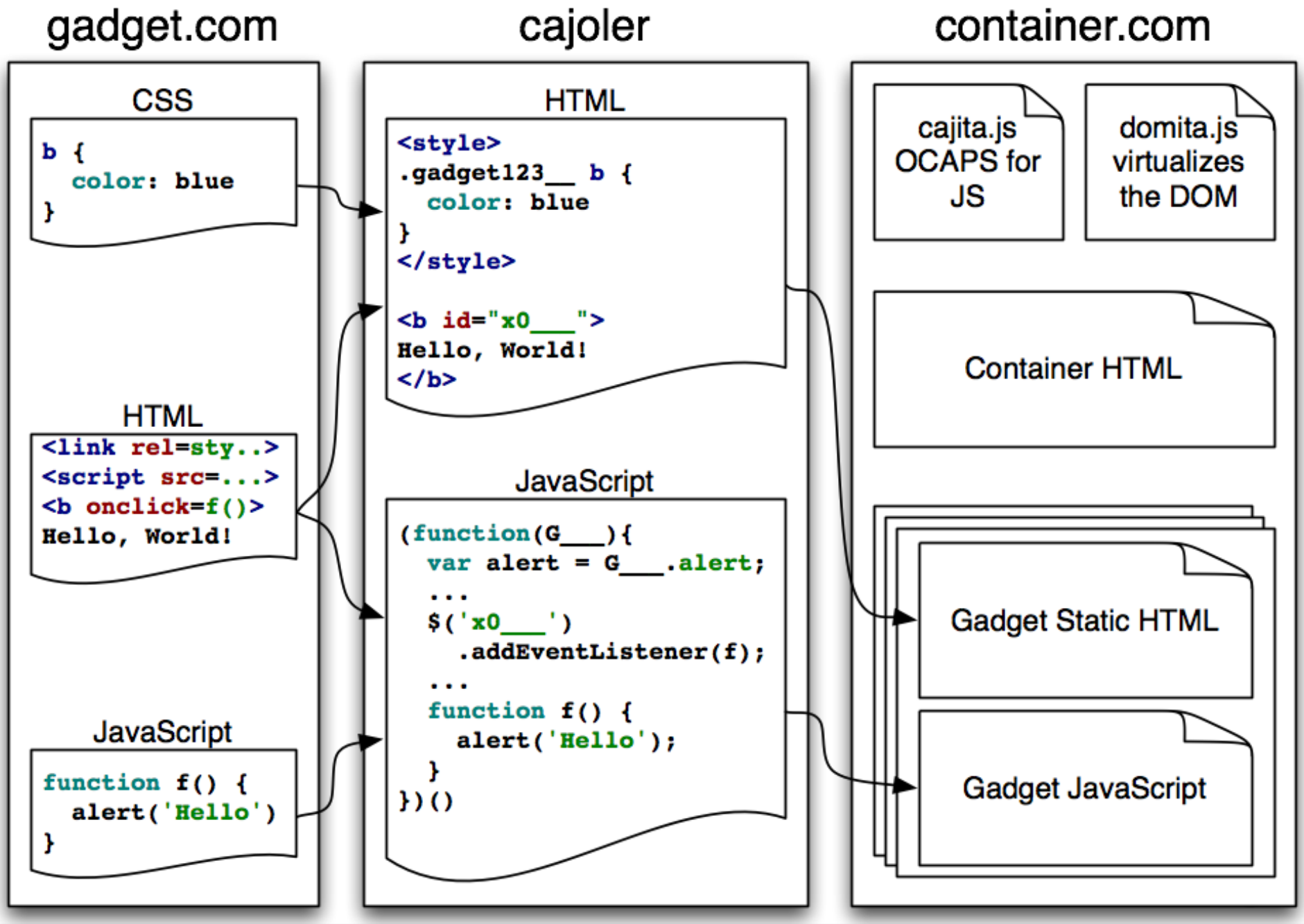

gadget.com
cajoler
CSS
b {
color: blue
}
HTML
<link rel=sty..>
<script src=...>

Hello, World!
JavaScript
function f() {
alert('Hello')
}
HTML
<style>
.gadget123__ b {
color: blue
}
</style>

Hello, World!

JavaScript
(function(G___){
var alert = G___.alert;
...
$('x0___')
.addEventListener(f);
...
function f() {
alert('Hello');
}
})()

Example App


gadget.com
cajoler
container.com
CSS
b {
color: blue
}
HTML
<link rel=sty..>
<script src=...>

Hello, World!
JavaScript
function f() {
alert('Hello')
}
HTML
<style>
.gadget123__ b {
color: blue
}
</style>

Hello, World!

JavaScript
(function(G___){
var alert = G___.alert;
...
$('x0___')
.addEventListener(f);
...
function f() {
alert('Hello');
}
})()
cajita.js
OCAPS for
JS
domita.js
virtualizes
the DOM
Container HTML
Gadget Static HTML
Gadget JavaScript

Why Virtualize?

Problem: Implemented policy is not what you want

If you

- can't wait for new standards
- can't wait for browsers to roll out fixes to most of your users
- can't wait for third party dev to rewrite their code

Solution: You need your security policy in code you control.

Why Virtualize?

Problem: Required security policy changes

If your threat model changes because

- cost of an exploit may decrease
- cost of weaponizing an exploit may decrease
- the value your are protecting may increase
- you may overlook an attack vector

Solution: You need your security policy in code you control.

Argument in a nutshell

Social Networks compose web applications from small apps

This breaks the **same origin** policy

A network that gives developers the most **authority** will **grow**.

The bigger networks can neither **trust** nor **police** developers.

And they can't **predict** all the threats they will face.

Virtualization lets you promiscuously grant authority to grow.

And **dial it back** later, after you understand threats.

Without breaking **APIs**.

Software Interposition for the Web

Google Caja

http://code.google.com/p/google-caja/

jasvir@google.com
msamuel@google.com

google-caja-discuss@googlegroups.com

http://caja.appspot.com/

Appendix: What is an OCAP Language

Authority follows from Object references.

If you can reference an object then you have all the authority its public API exposes.

To grant authority to a piece of code, you pass it objects.

In an OCAP Language

- Objects are inviolable - only manipulable through public API
- Objects are unforgeable. To create an object you must have authority to do so granted via an object reference.
- Objects are not ambiently available. All authority flows from granted references.

Appendix: Language Support

EcmaScript version 5

- Backwards compatible strict mode
- Statically Analyzable scopes
- Runtime message interception (no doesNotUnderstand)
- Object freezing

EcmaScript Harmony (version 6?)

- Proxies
- Ephemeron tables

Appendix: Efficiency

Overhead from

- Code bloat
- Runtime checks
- Virtualization

Strategies

- Speed : do as much analysis statically as possible.
- Latency : memoize work per module

EcmaScript 5

- Our transformer becomes a verifier. No runtime checks / code bloat. (except when code dynamically loaded)

EcmaScript 6

- Proxies reduce virtualization overhead