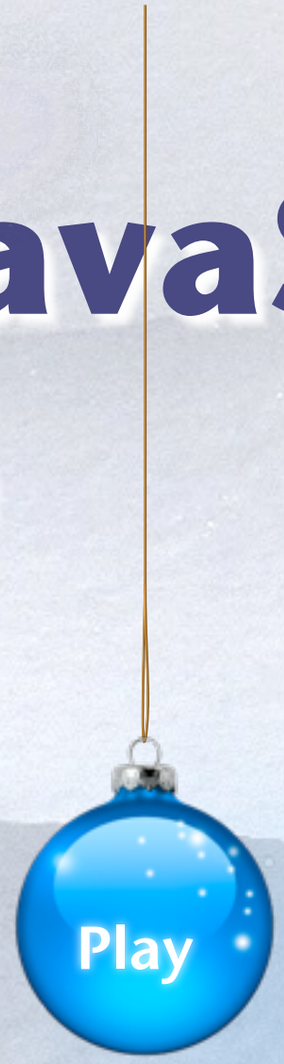
Lecture 12
Asynchronous JavaScript and XML (AJAX)

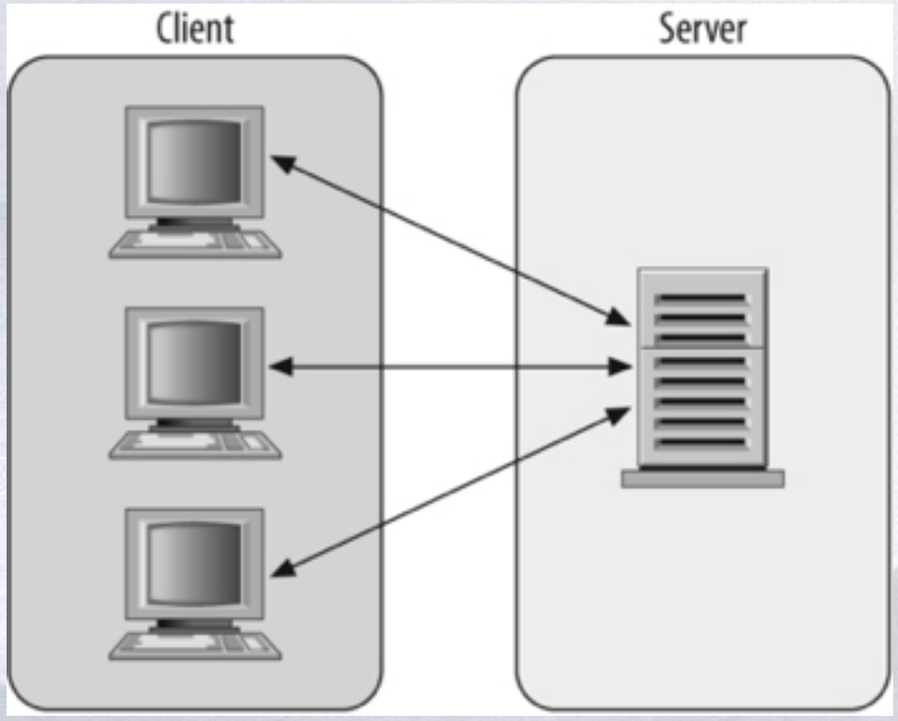


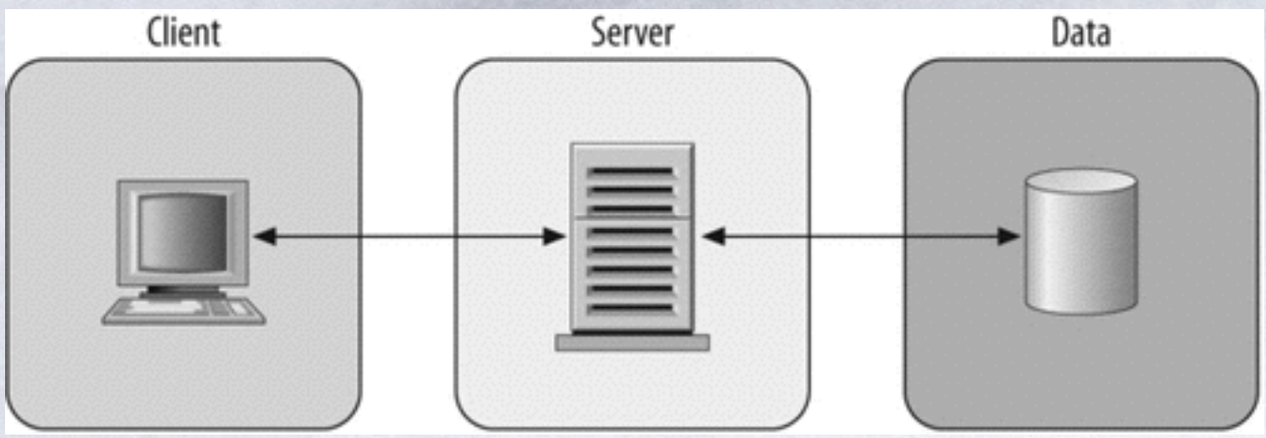
Outline

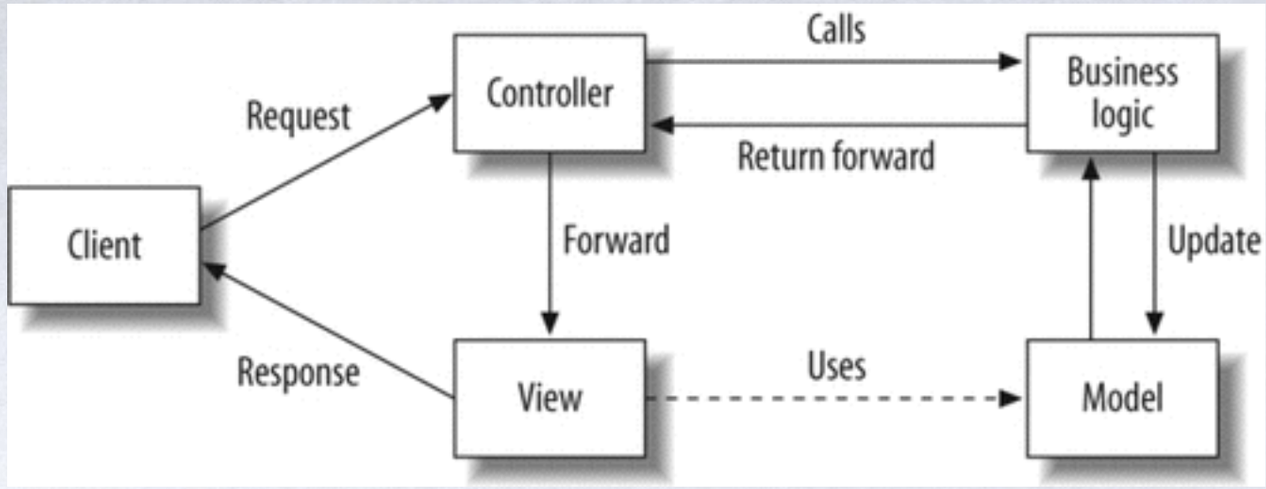
- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

From Web Sites to Web Applications

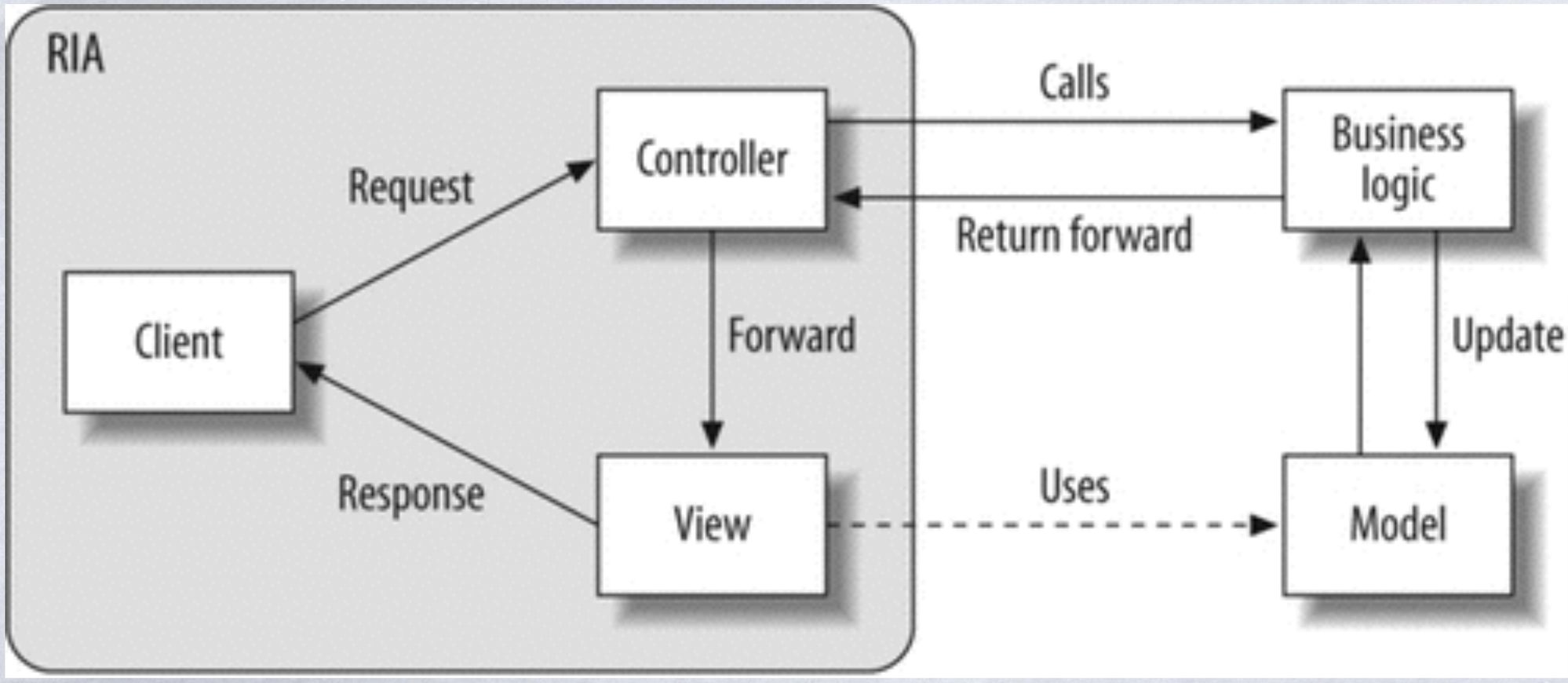
- Client/Server
  - static content
- Basic Three-Tier
  - user interface, business or process logic, and a data access module
- Model-View-Controller
- Rich Internet Applications







Rich Internet Applications



An RIA implementation on top of MVC

Merits of RIA application

- Ajax web applications require no installation, updating, or distribution, as everything is served up by a web server.
- Ajax web applications are less prone to virus attacks (generally).
- Ajax web applications can be accessed anywhere, and if they are built properly, you can run them on any operating system.

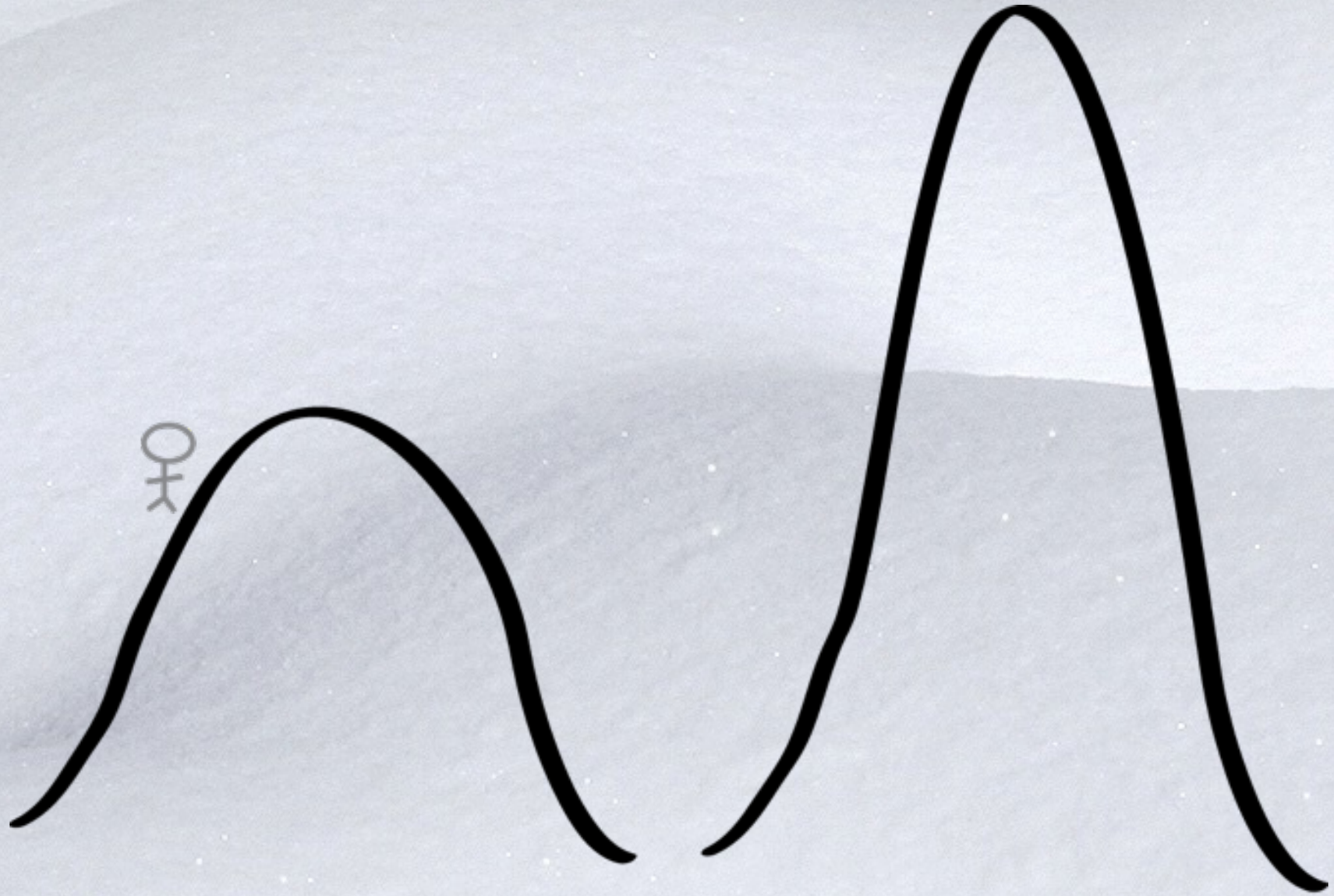
RIA Approaches

- Browser plug-in
  - Flash/Flex, Java Swing, Silverlight
  - Potentially greater interactivity, higher barrier to adoption
  - Concerns about openness/control
- In browser, no plug-ins
  - AJAX
  - Lower barrier to adoption
  - Cross-browser mayhem?

Questions for today

- Can we approach AJAX development like "regular" GUI development?
- What are the approaches/tradeoffs?
- What's likely to become popular?

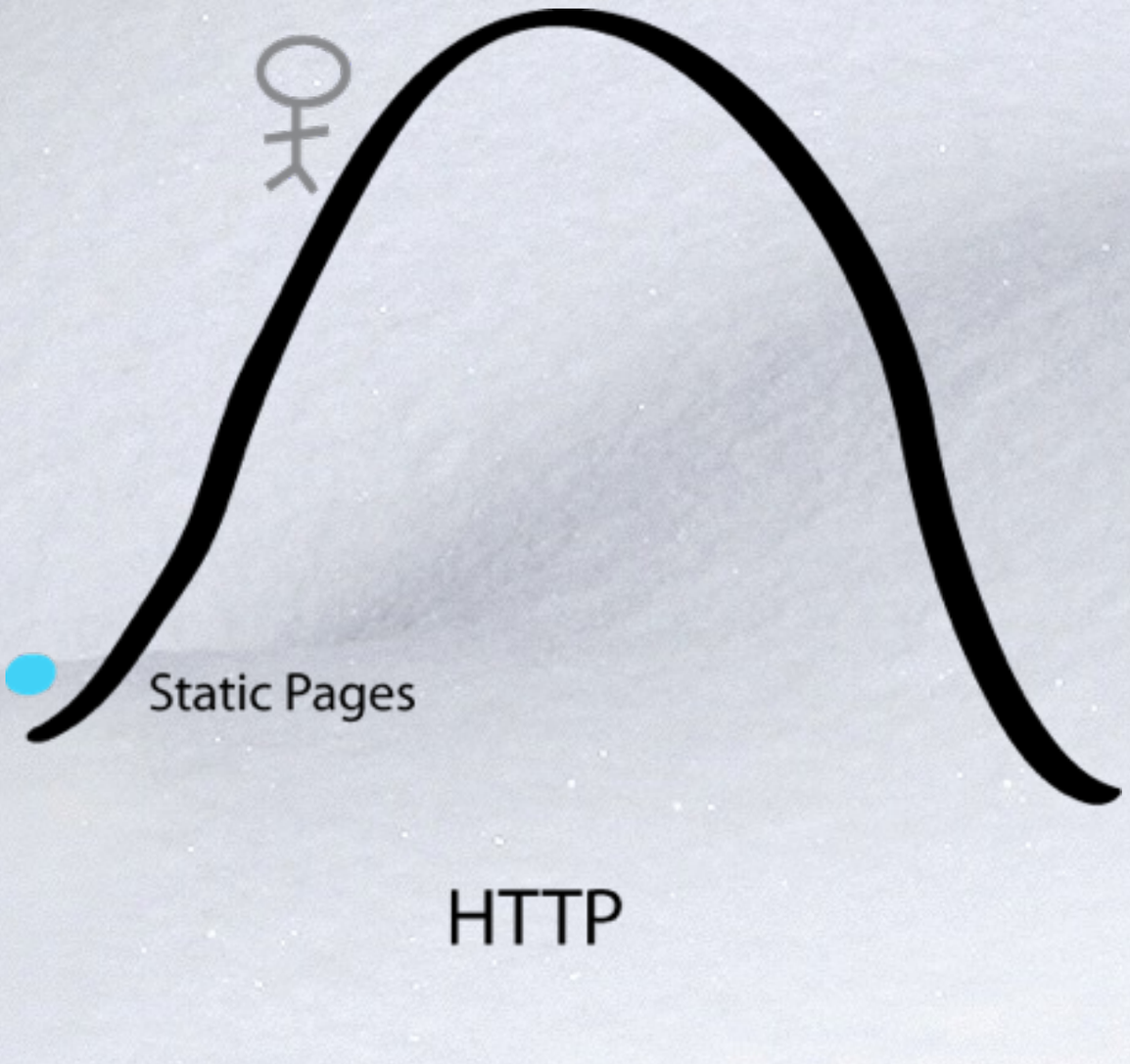
RIA: Which hill to climb?



Approaching RIAs from two hills

HTTP	Direct Manipulation
Origins in early web sites	Origins in desktop GUIs
Built around the HTTP protocol	Built around user events
Generating HTML	Laying out graphical objects

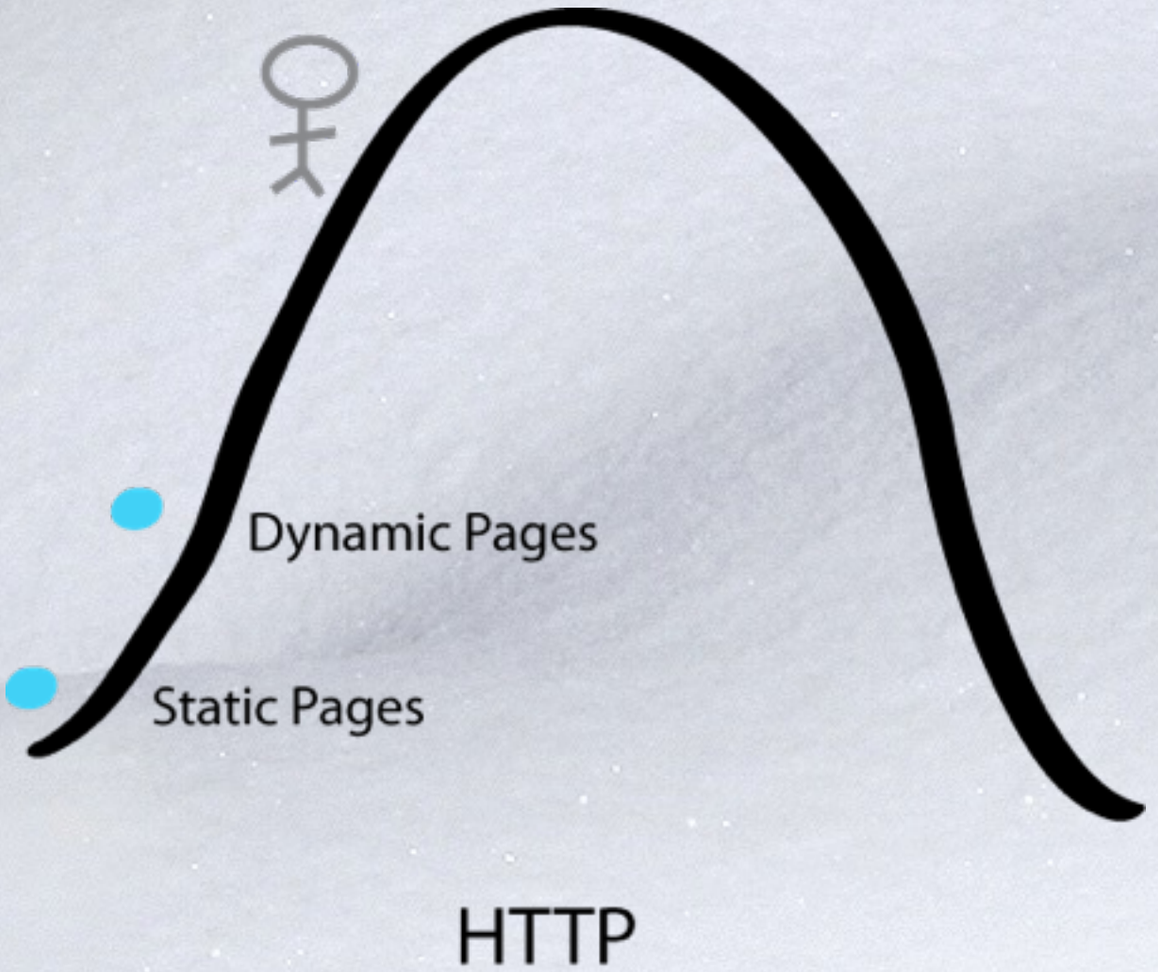
The HTTP Hill



Static Pages

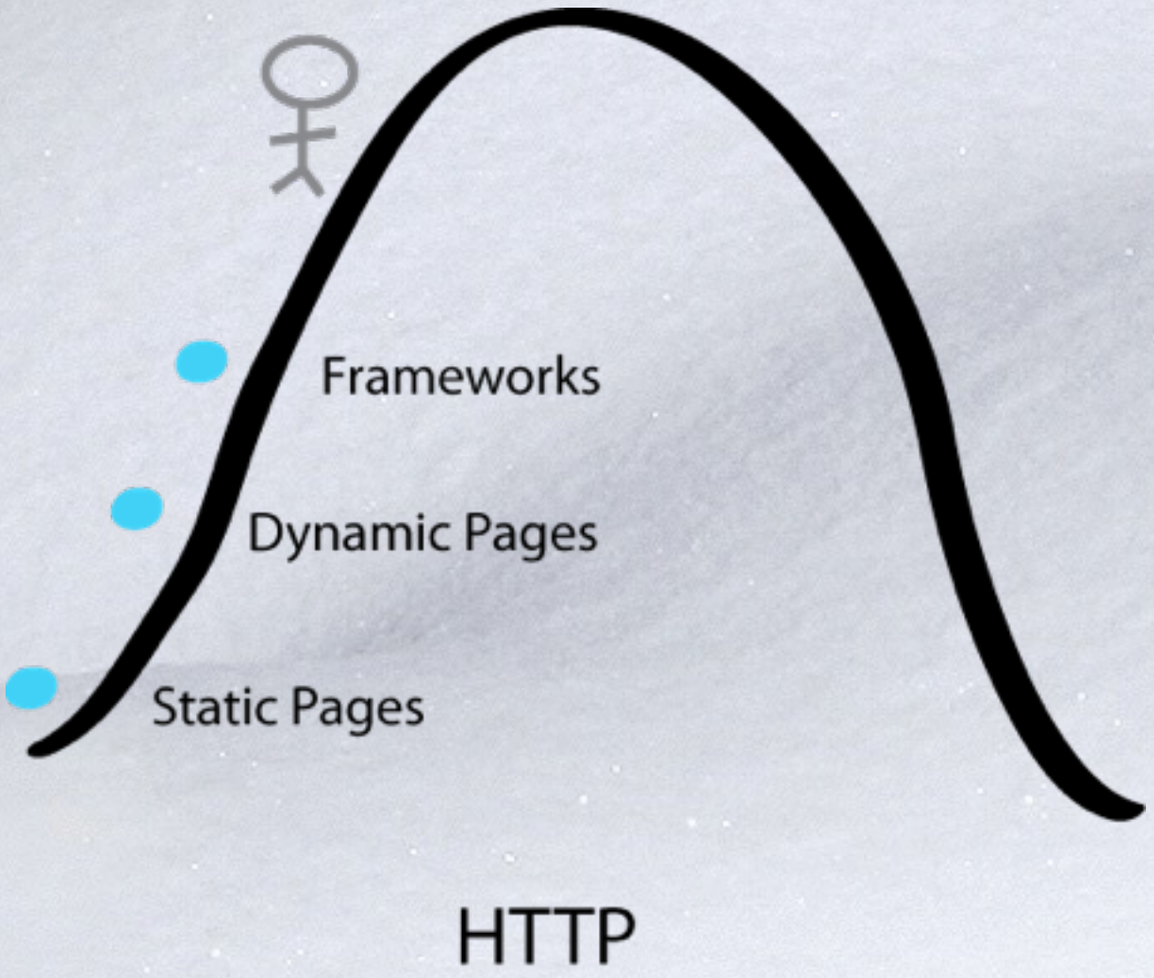
- Server fetches and returns a web page
- Initially just text-based
- With Mosaic, pictures too

The HTTP Hill



- Server-side
  - CGI (mostly perl)
- Client-side
  - Javascript

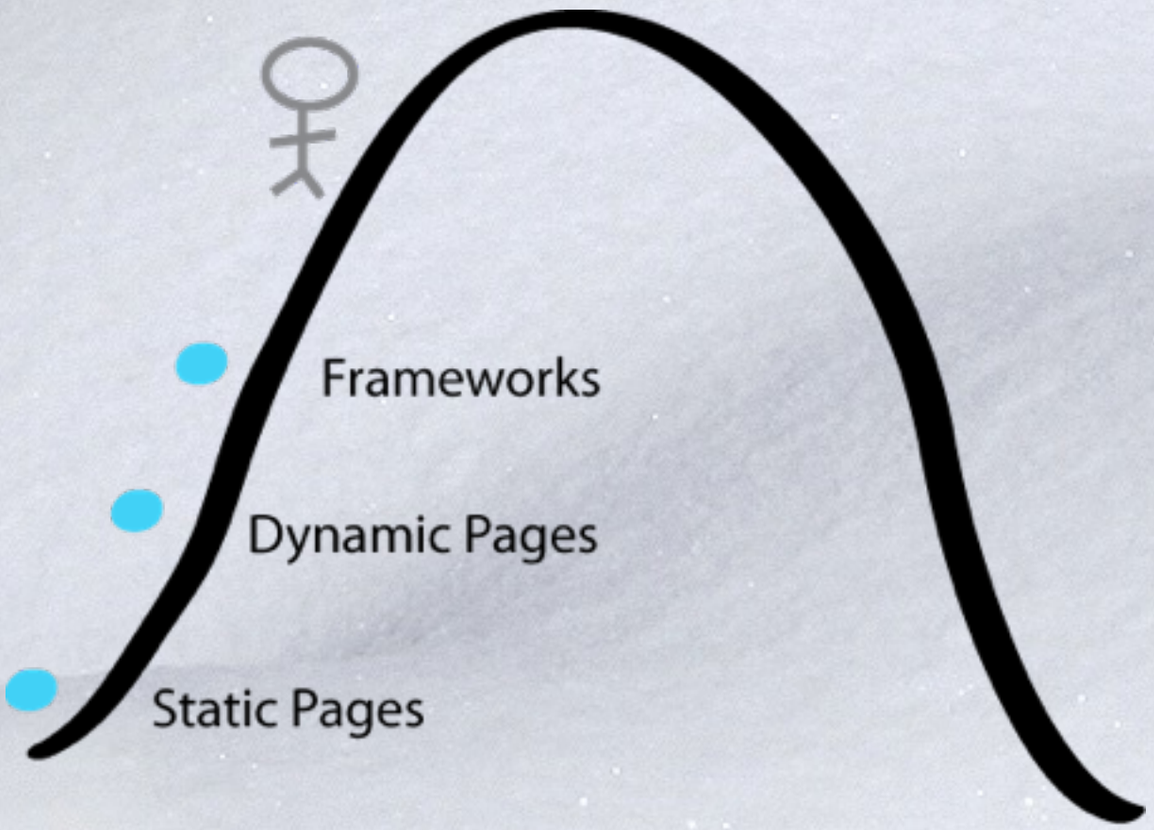
The HTTP Hill



“Frameworks”

- MVC support
  (Struts 1 & 2, Rails, Django)
- Easier HTML generation
  (JSP, ERB, Freemarker, …)
- State/sessions
- Javascript libraries
  (Prototype, DOJO, jQuery)

The HTTP Hill

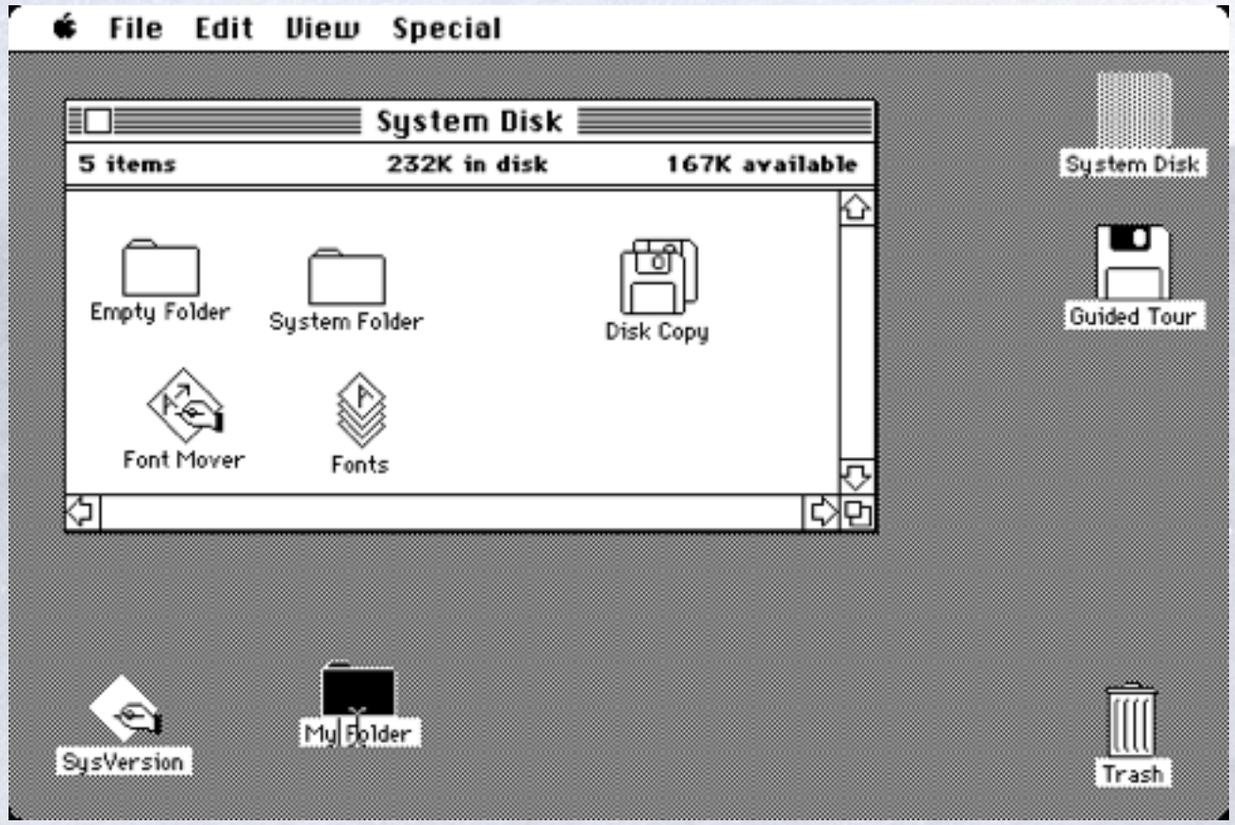


Pros/Cons (prior to AJAX)

+ Very cheap for simple sites
+ Reasonably flexible
+ Web-friendly
  - Bookmarkable
  - Indexable
– Slow feedback
– Minimal interactivity
– Cross-browser mayhem

The Direct Manipulation Hill

Direct Manipulation

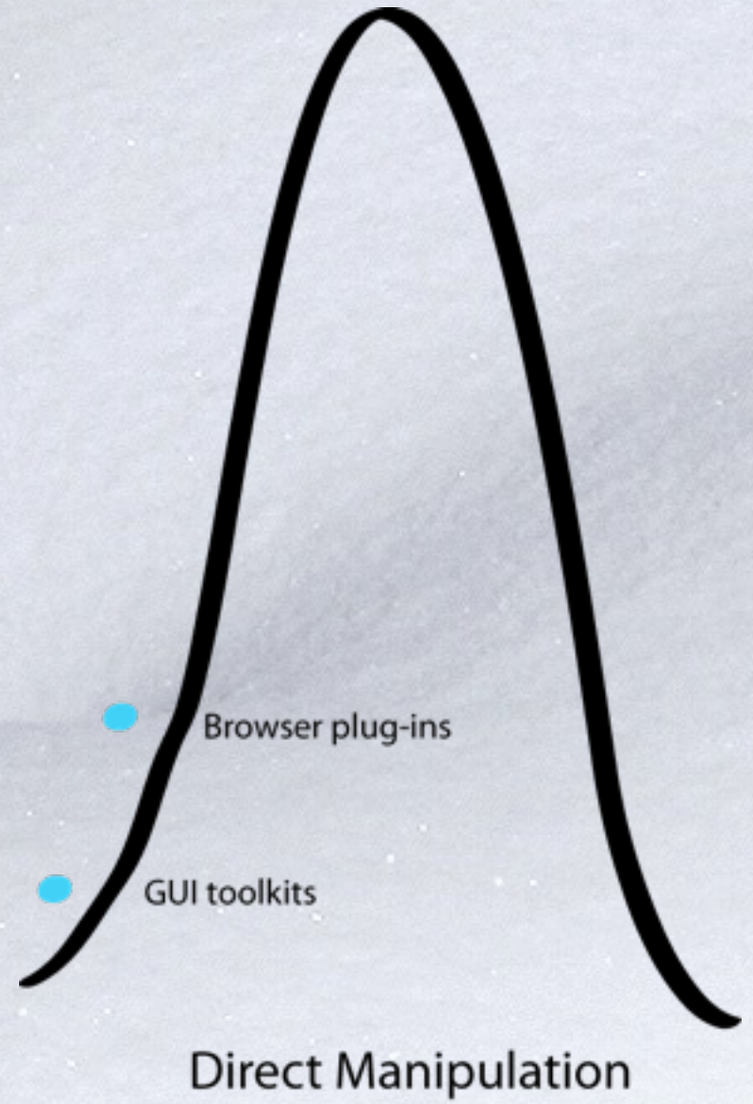


The Direct Manipulation Hill

GUI Toolkits

- Common widget set across applications
- Standalone or client-server

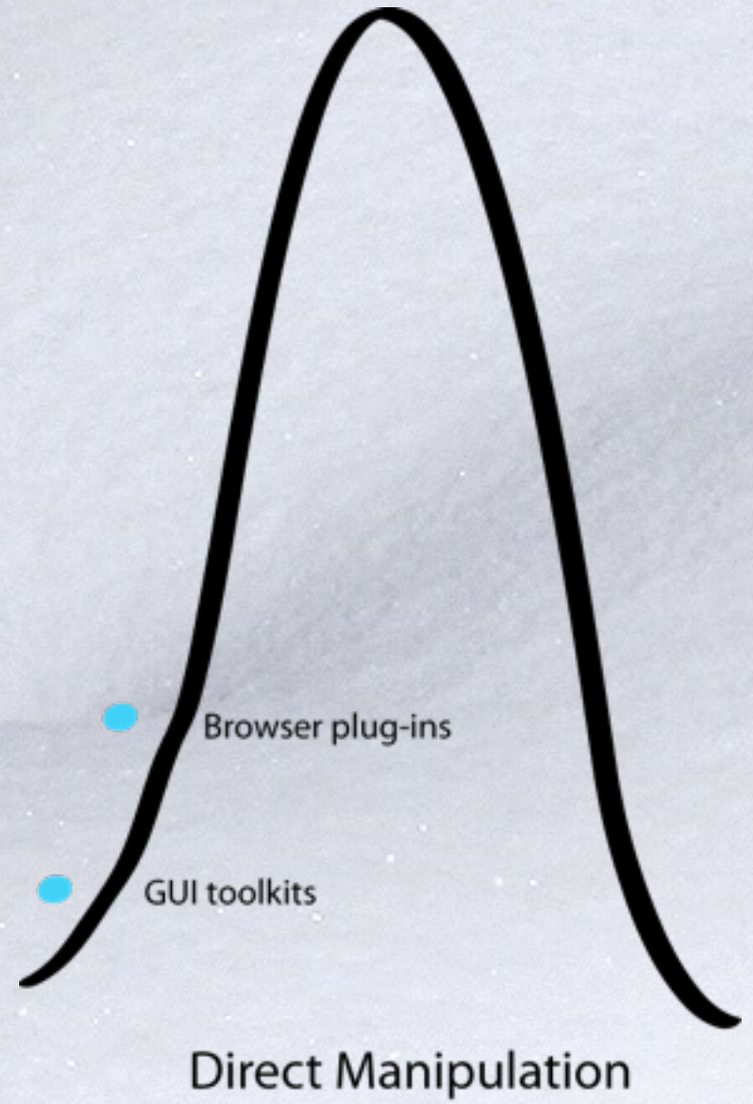
The Direct Manipulation Hill for Internet applications



Browser Plug-Ins

- Flash, Java, Silverlight
- Took a long time to catch on

The Direct Manipulation Hill



Pros/Cons

+ Timely feedback
+ Programming power (behaviors, constraints – at least possible)
+ Common widgets (consistency, usability)
+ Flash/etc: more consistent runtime platform
– Flash/etc: needs a plug-in
– Cross-platform issues still exist
– Proprietary runtime platform

Where does AJAX fit in?

Both hills!

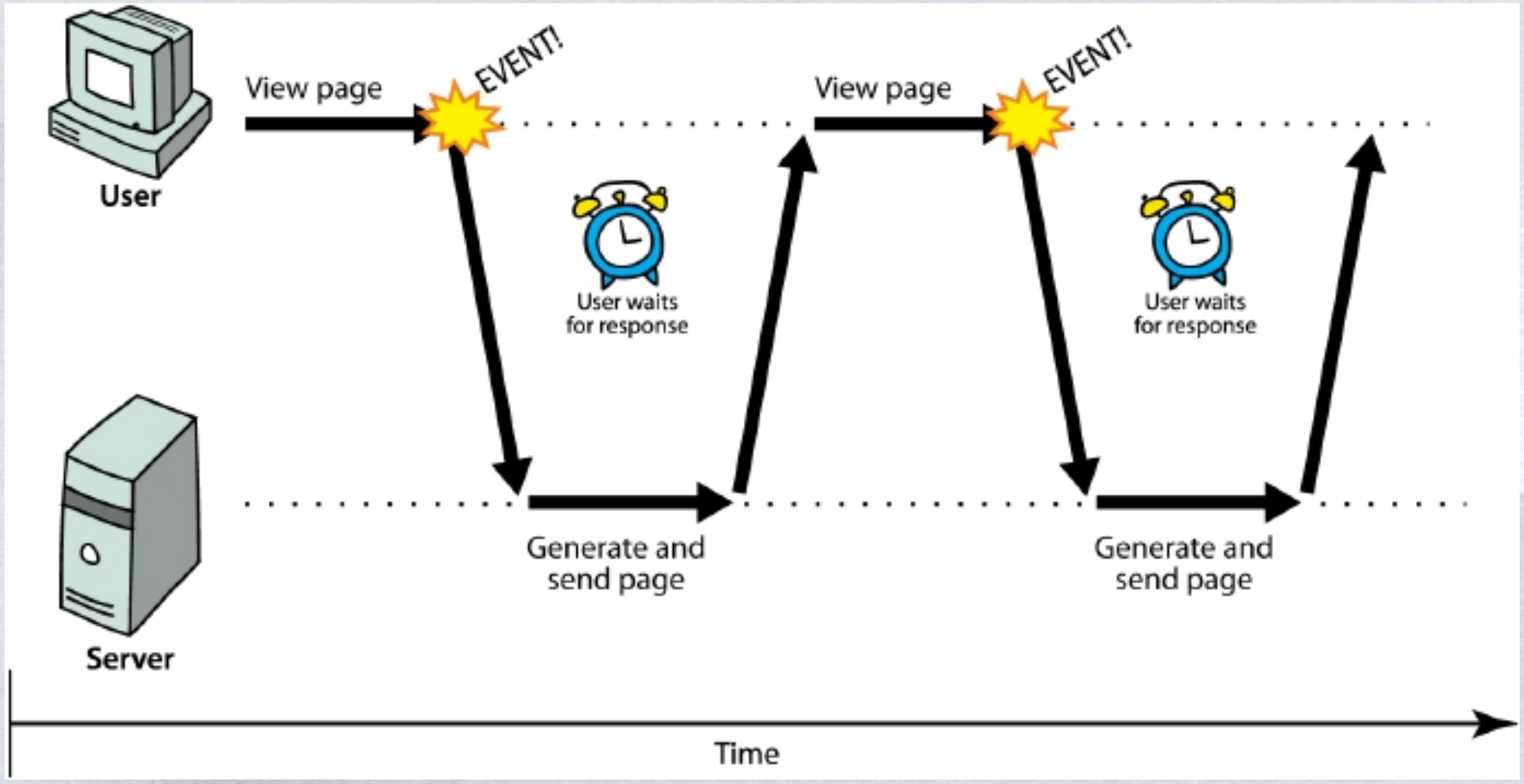
HTTP

Direct Manipulation

Outline

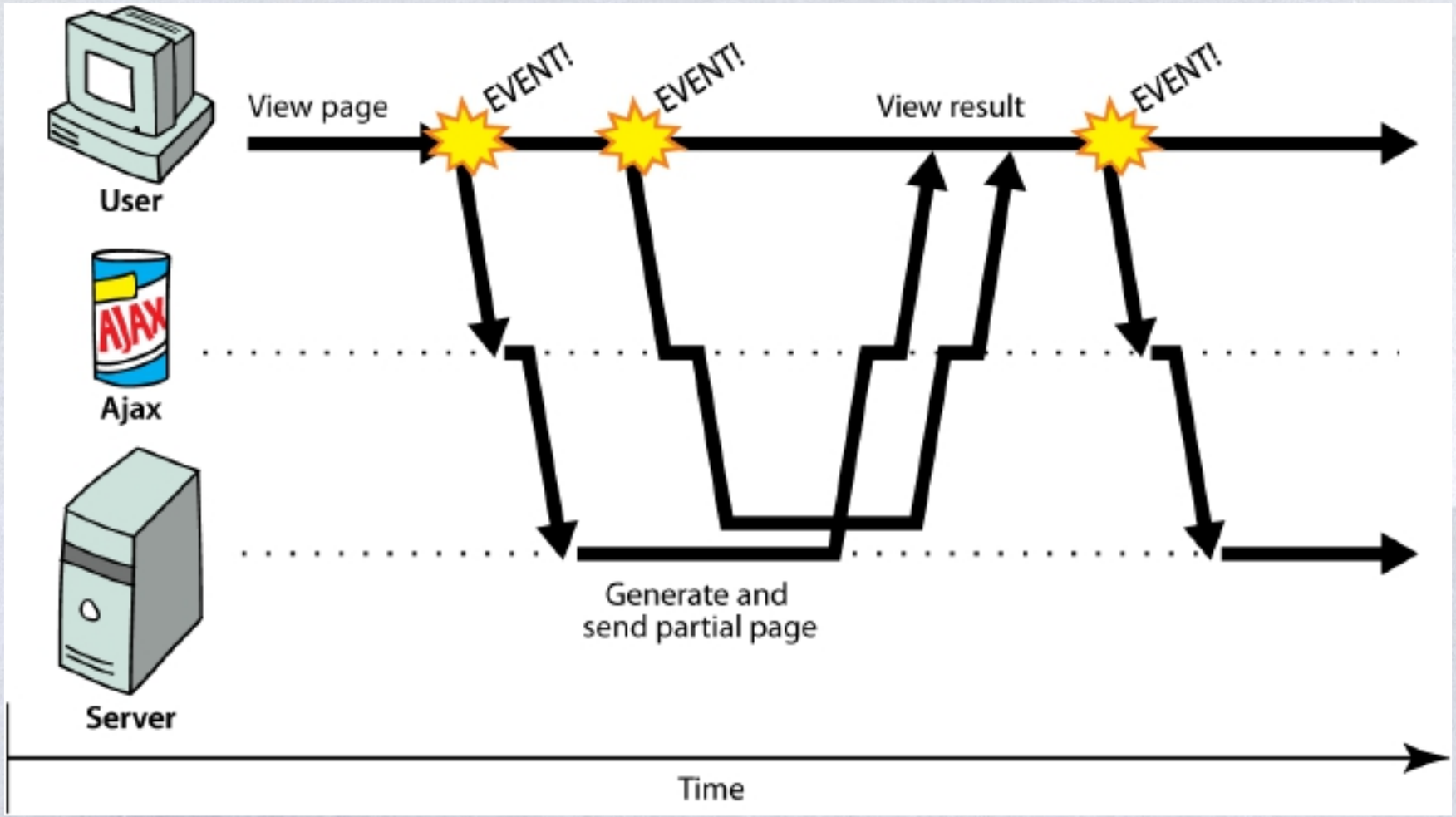
- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

Synchronous web communication



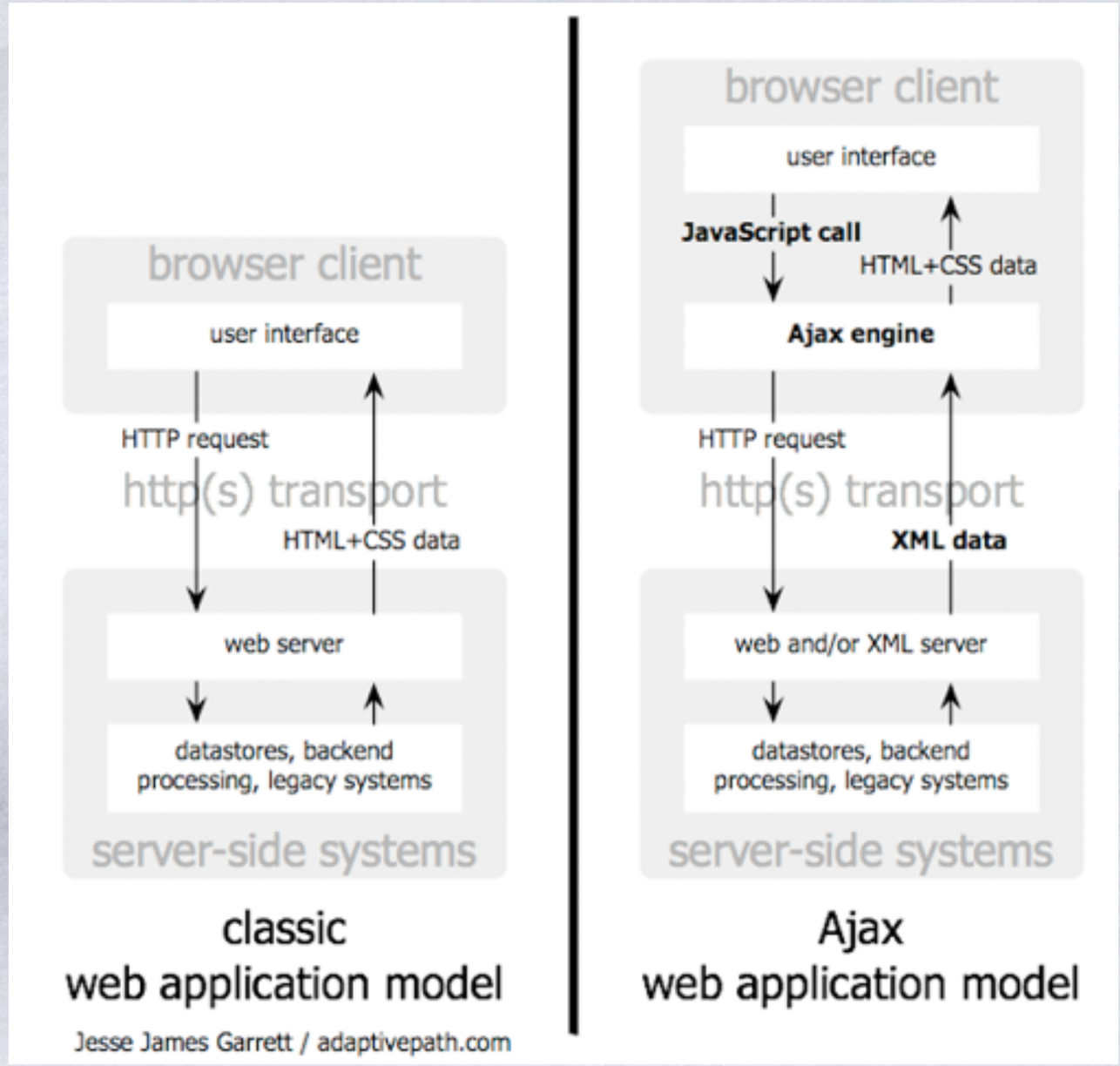
- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)
- almost all changes with new data lead to page refresh

Asynchronous web communication



- asynchronous: user can keep interacting with page while data loads
  - communication pattern made possible by Ajax
- Changing with new data but without page refresh

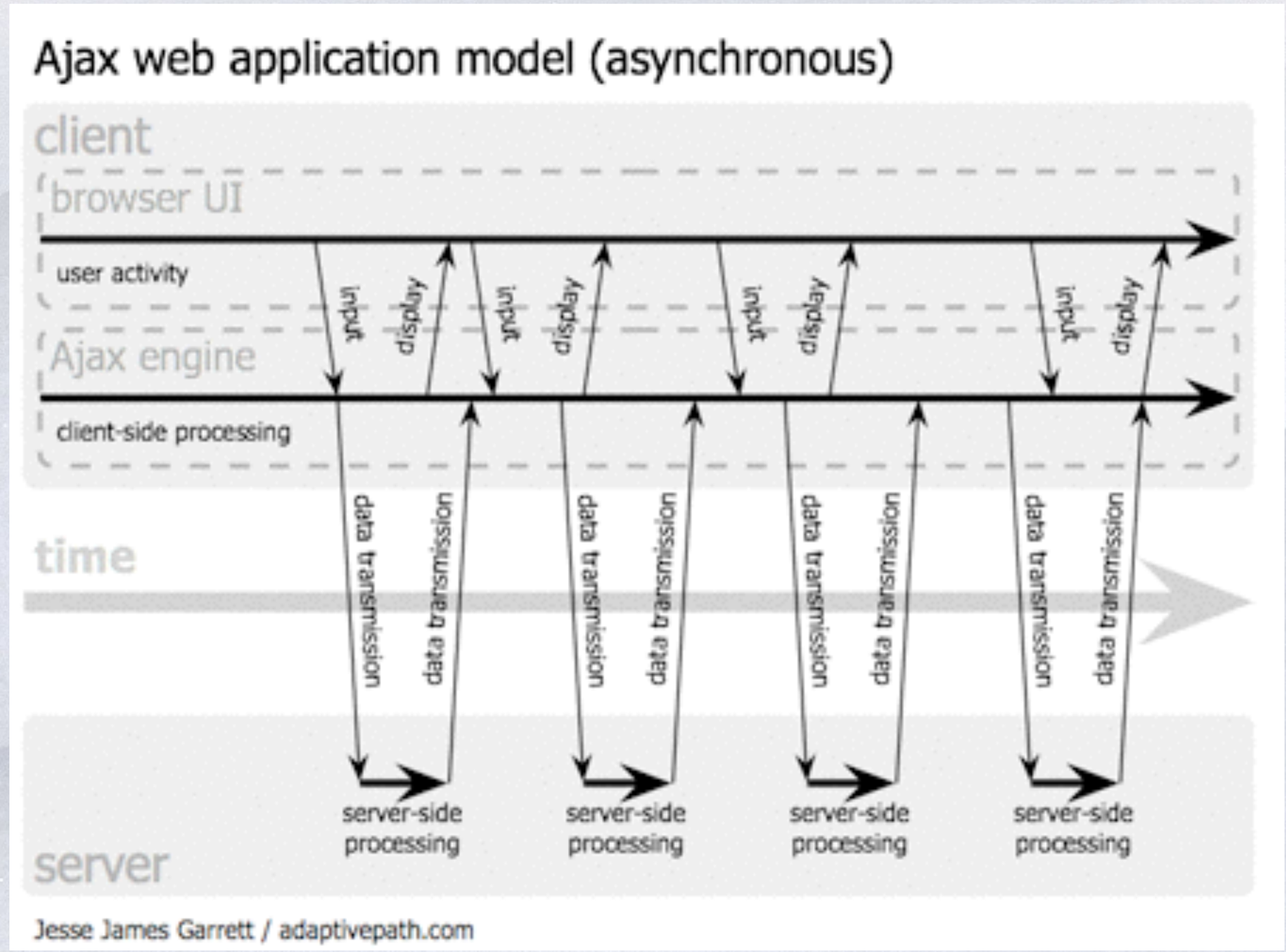
What is AJAX?


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend processing, legacy systems
server-side systems
classic web application model
Jesse James Garrett / adaptivepath.com
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend processing, legacy systems
server-side systems
Ajax web application model

Web applications and Ajax

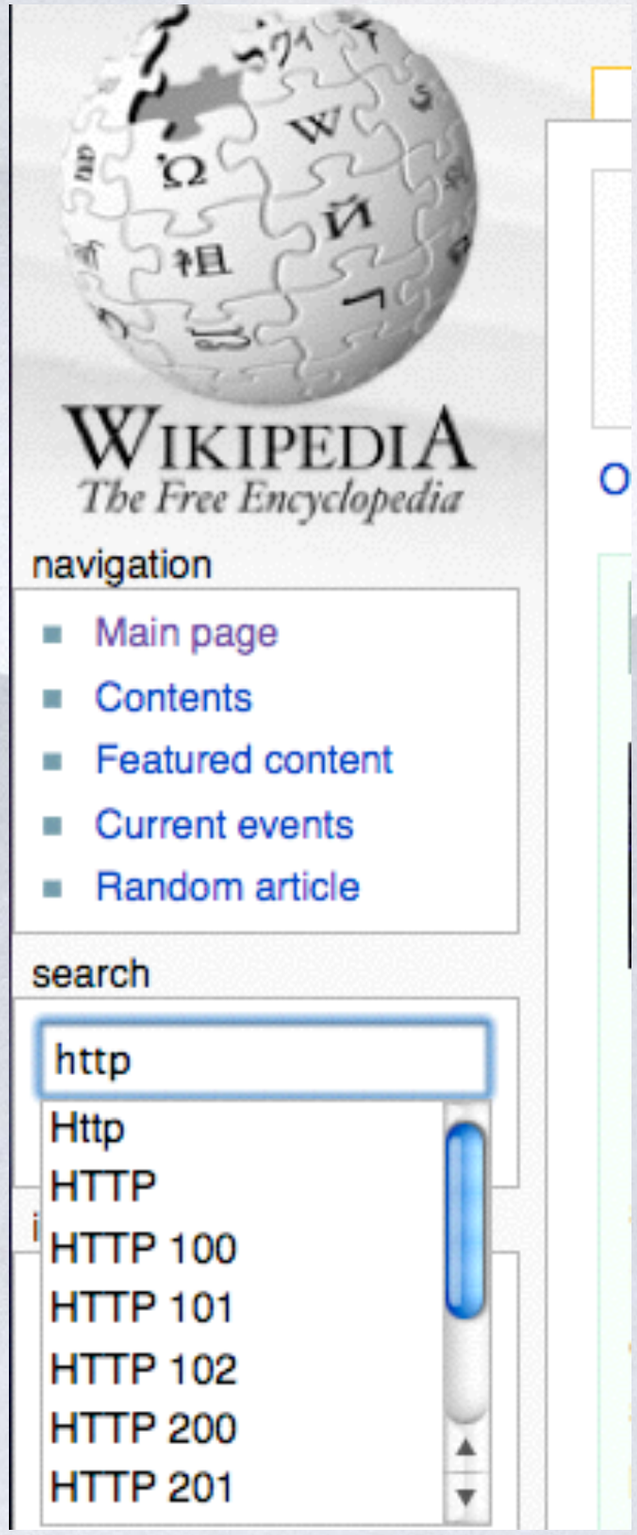
- web application: a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9
- Ajax: Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page
  - avoids the "click-wait-refresh" pattern
  - examples: Google Suggest

AJAX event handling



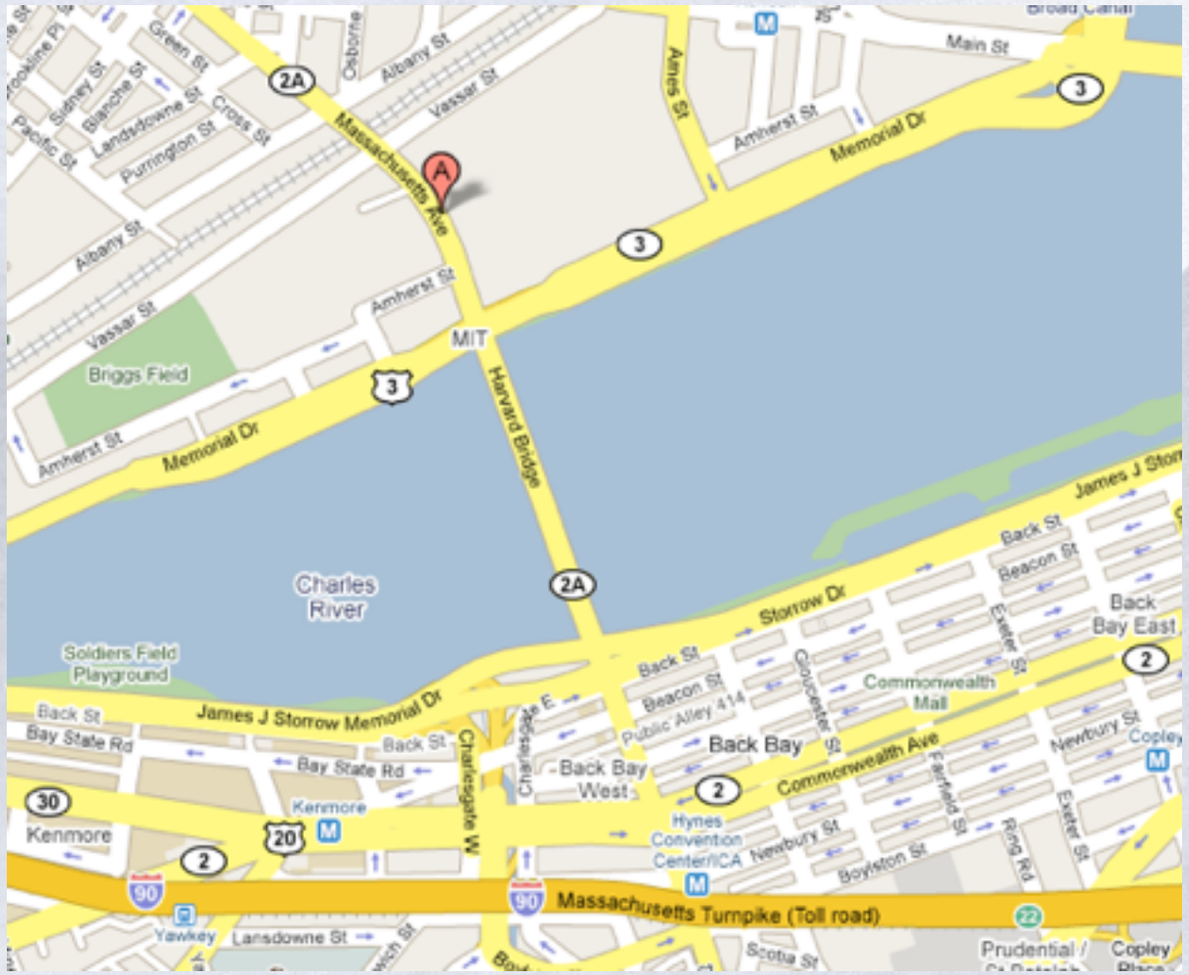
AJAX on the HTTP Hill

- Tactical features
  - Autocomplete
  - Drag and drop
- AJAX-aware code
  - Raw Javascript/HTML/CSS
  - Or with a library
- Okay for some applications
- Too limiting for RIAs

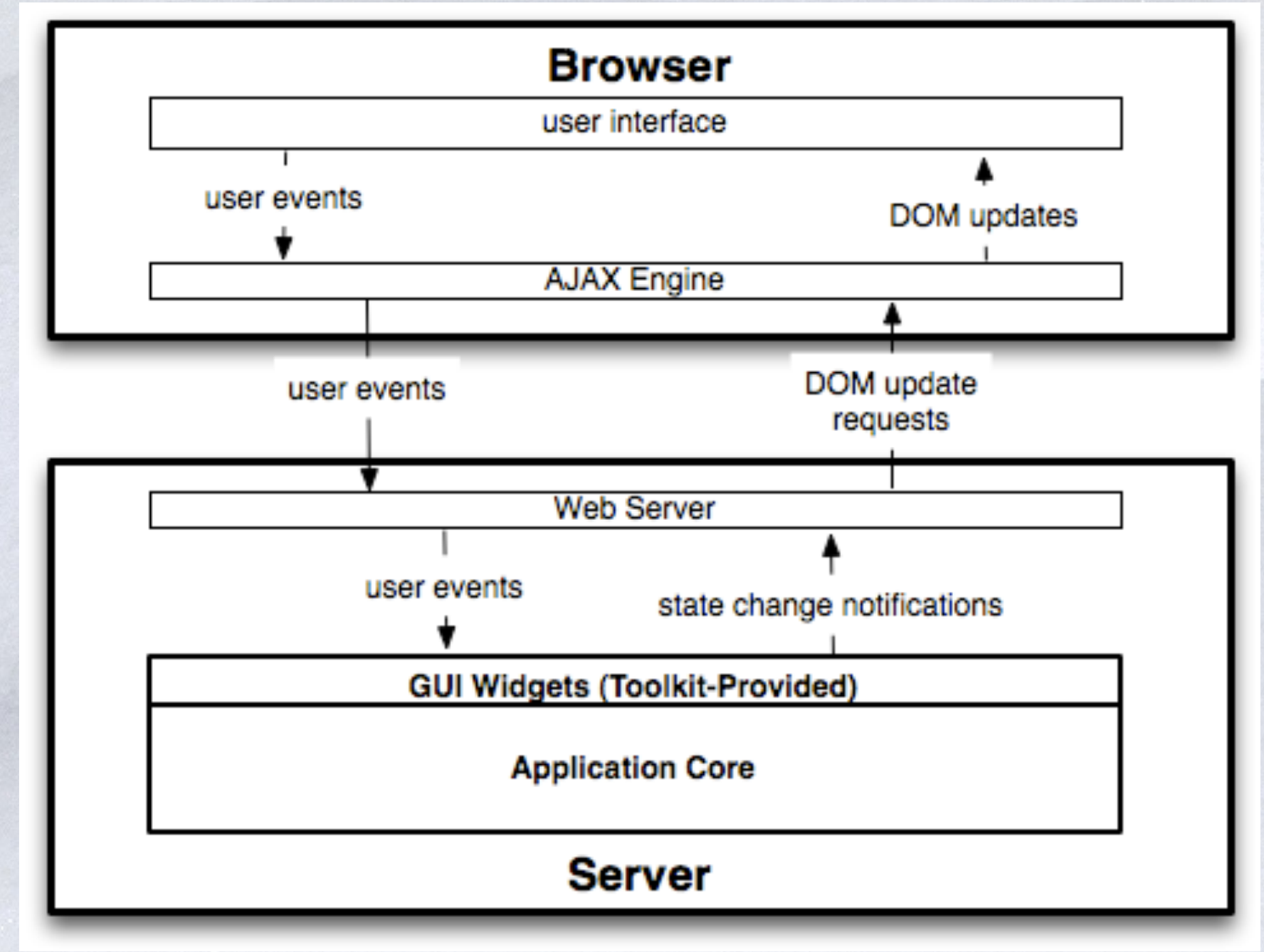


AJAX on the Direct Manipulation Hill

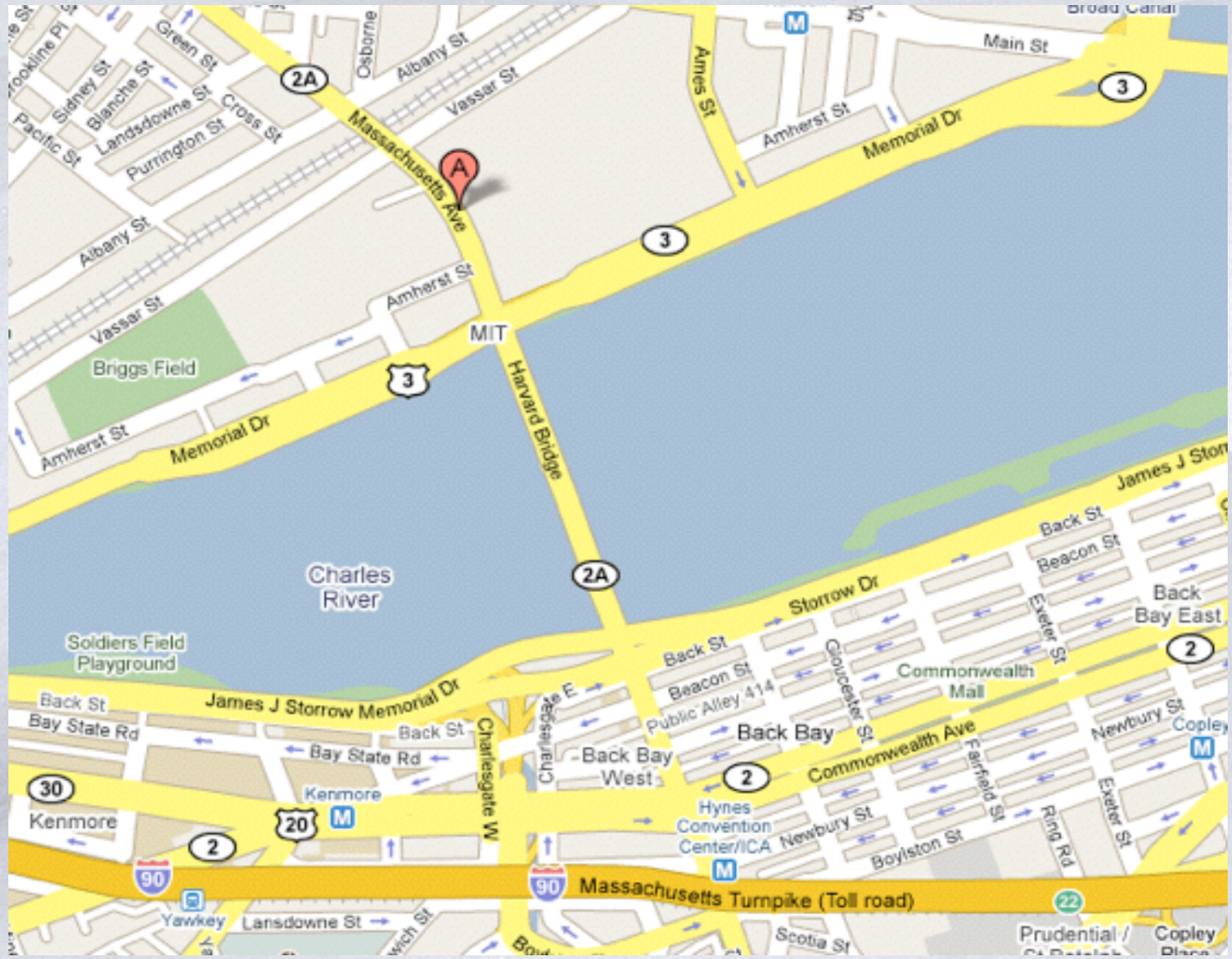
- Separate development environment from runtime environment.
- Runtime environment: HTML/Javascript/CSS (AJAX)
- Development environment: toolkit in another language
- Two approaches: thin and fat



Thin Client AJAX Approach



Example: Google Maps (pretend it's a thin client app)

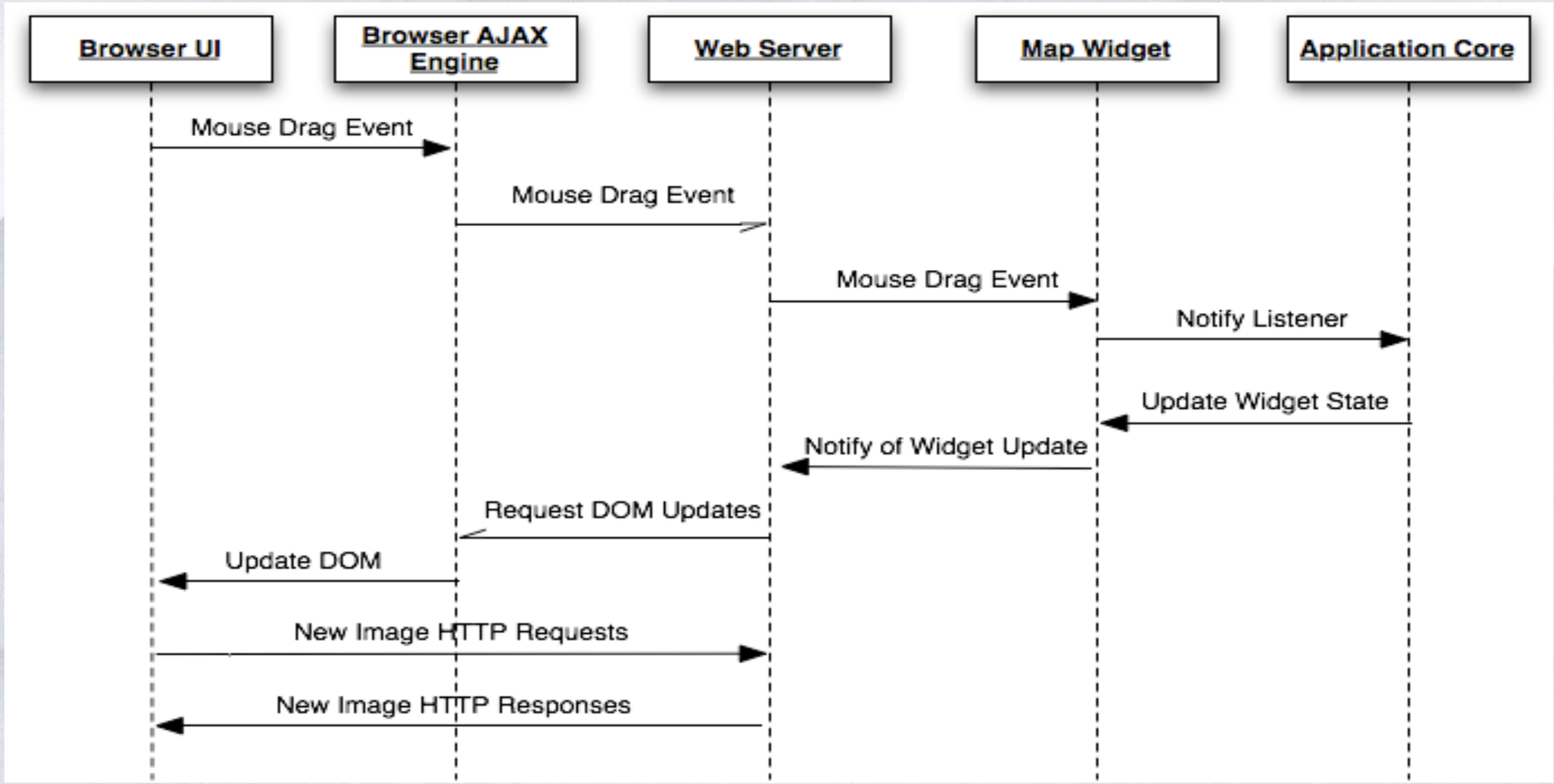


Example: Google Maps

OFFSCREEN


1
2
3
10
4
5
6
11
7
8
9
12
13
14
15
16
Charles River
James J Storrow Memorial Dr
Storrow Dr
Back St
Beacon St
Public Alley 414
Back Bay
Back Bay East
Commonwealth Mall
Commonwealth Ave
Newbury St
Boylston St
Exeter St
Copley
Kenmore
Hynes Convention Center/ICA
Massachusetts Turnpike (Toll road)
Prudential /
Copley Place
Lansdowne St
Bay State Rd
Charlesgate W
Charlesgate E
Harvard Bridge
Amherst St
Memorial Dr
MIT
James J Storr

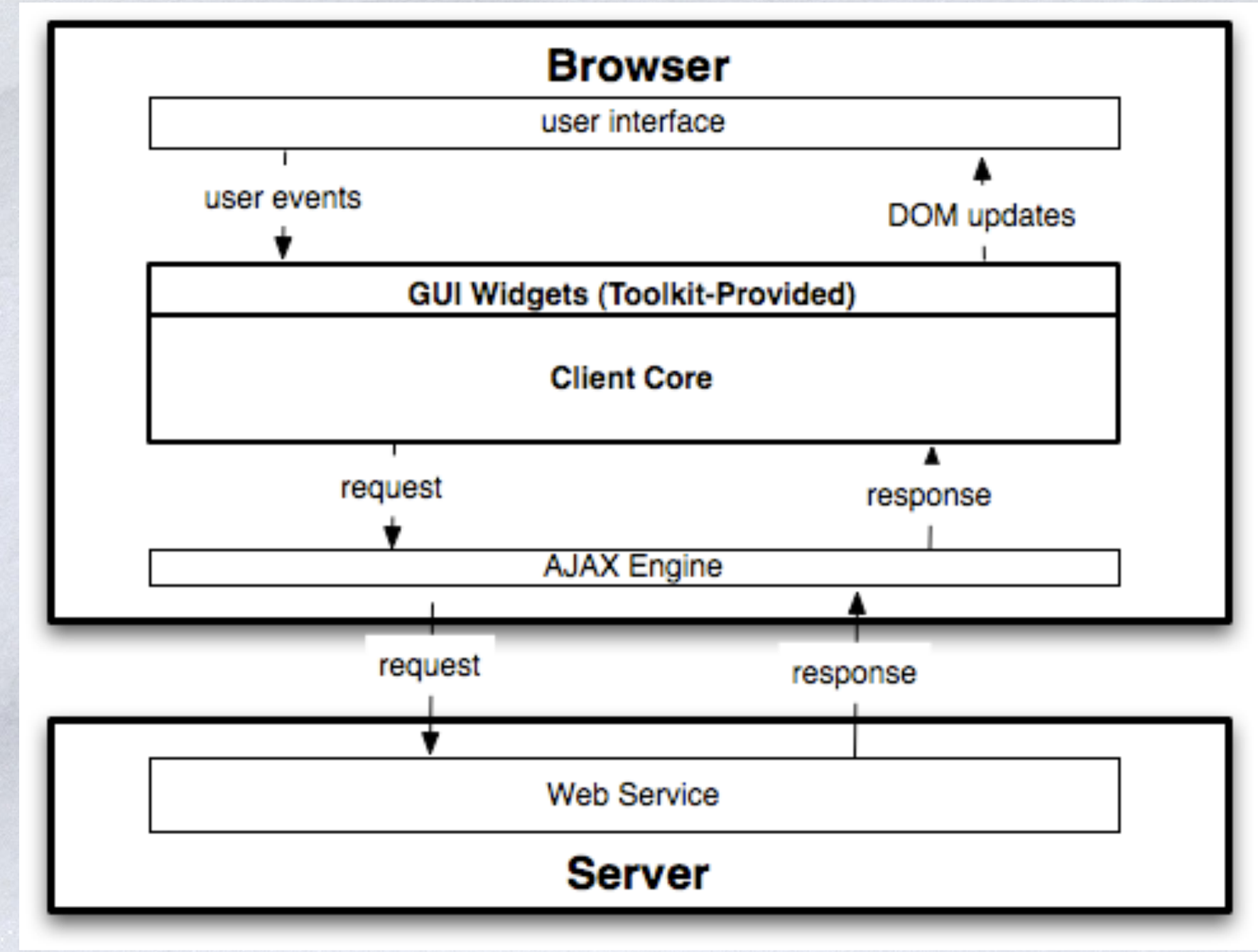
Sequence


Browser UI
Browser AJAX Engine
Web Server
Map Widget
Application Core
Mouse Drag Event
Mouse Drag Event
Mouse Drag Event
Notify Listener
Update Widget State
Notify of Widget Update
Request DOM Updates
Update DOM
New Image HTTP Requests
New Image HTTP Responses

Thin Client Pros and Cons

- + Simple programming: ignore the network
  - + All your code runs server-side
  - + Programmers love it!
- + Undo, behaviors, constraints: all possible!
- – Scalability (server-side state, lots of requests)
- – Slow feedback: network hop for each user action

Fat Client AJAX Approach

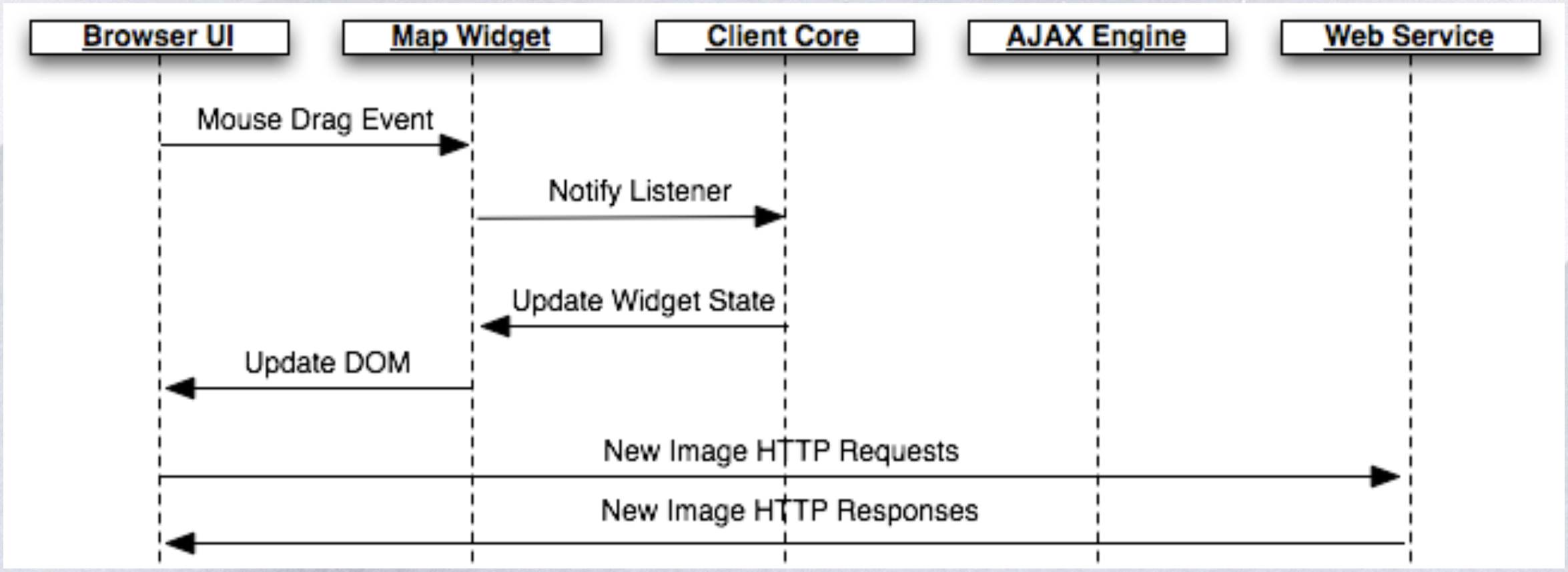


Example: Google Maps

OFFSCREEN

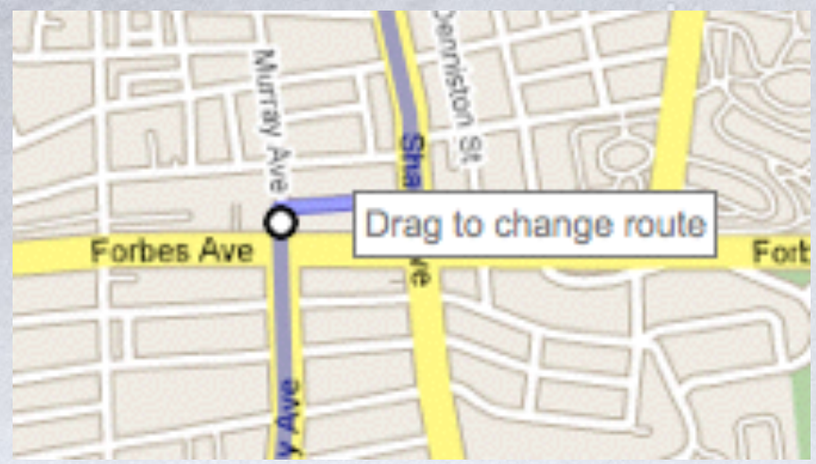

1
2
3
10
4
5
6
11
7
8
9
12
13
14
15
16
Charles River
James J Storrow Memorial Dr
Storrow Dr
Back St
Beacon St
Public Alley 414
Back Bay
Back Bay East
Commonwealth Mall
Commonwealth Ave
Newbury St
Boylston St
Exeter St
Copley
Kenmore
Hynes Convention Center/ICA
Massachusetts Turnpike (Toll road)
Prudential /
Copley Place
Lansdowne St
Bay State Rd
Charlesgate W
Charlesgate E
Harvard Bridge
Amherst St
Memorial Dr
MIT
James J Storr

Sequence


Browser UI
Map Widget
Client Core
AJAX Engine
Web Service
Mouse Drag Event
Notify Listener
Update Widget State
Update DOM
New Image HTTP Requests
New Image HTTP Responses

Wait a second...

- No AJAX calls involved in moving the map around!
  - Mostly Javascript.
  - New image requests are synchronous
- Example AJAX call: adding an intermediate destination



Fat Client Pros and Cons

+ Scalable (client-side state, fewer HTTP calls)
+ Fast feedback
+ Undo, behaviors, constraints possible...
– ...but undo more complex than on the desktop
– More complicated: network-aware, distributed

Example AJAX Toolkits

- Google Web Toolkit: Fat Client
  - Write in Java, compiled to Javascript
- Cappuccino: Fat Client
- Echo2: Thin Client
  - Write in Java
  - No HTML/CSS (proprietary stylesheet language)
- Echo3 (Java – Beta): hybrid
  - Thin widgets in Java
  - Fat widgets in Javascript

So, is AJAX viable for RIAs?

	Fat AJAX	Thin AJAX	Plugin (Flash, Swing)
Feedback Speed	**Winner (tied)**		**Winner (tied)**
Interactive Potential			**Winner**
Scalability	**Winner (tied)**		**Winner (tied)**
Cross-platform Consistency			**Winner**
Momentum	Google does a lot of work for you.	?	Adobe does a lot of work for you.
Ease of Programming		**Winner**	

Thin vs Fat AJAX?

- Thin AJAX: Squeezed out
  - Insufficient if interactivity matters
  - Not as easy as an HTTP-oriented application
- Fat AJAX: How does it compare to plug ins?
  - Developer adoption?
  - Application philosophy?

Outline

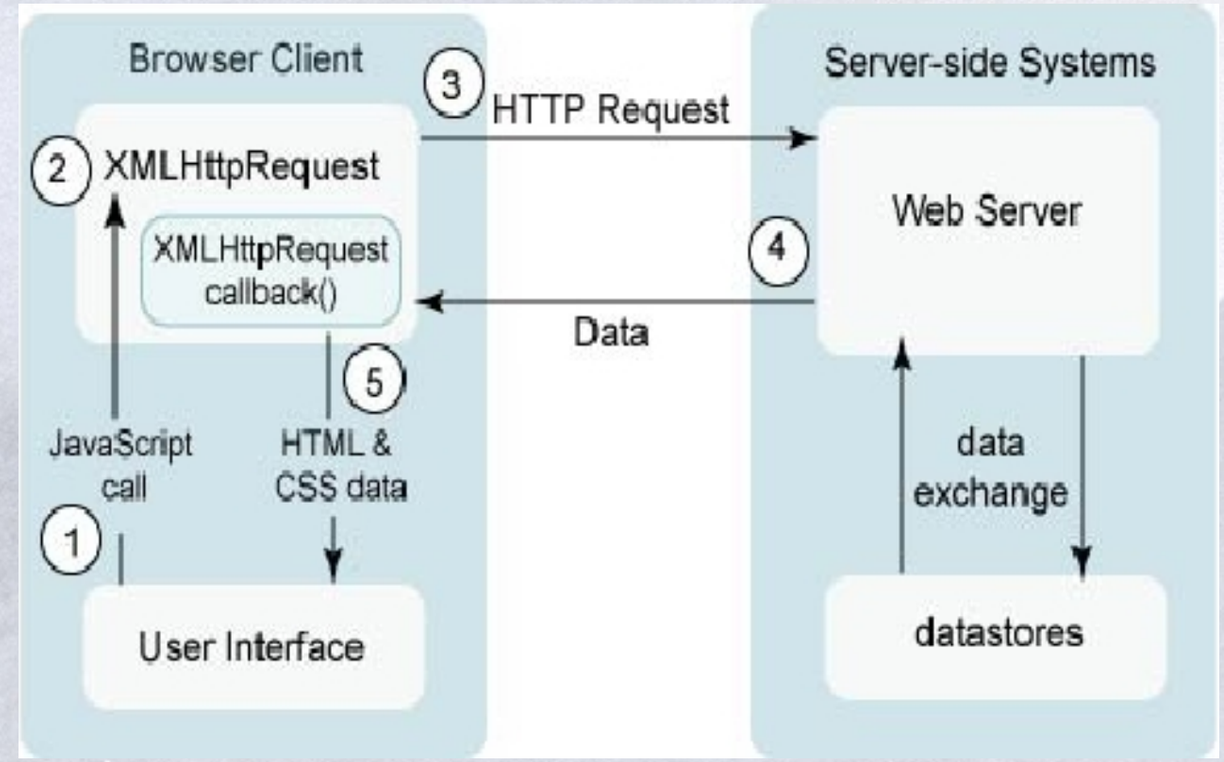
- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

XMLHttpRequest

- JavaScript includes an XMLHttpRequest object that can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- it can do this asynchronously (in the background, transparent to user)
- the contents of the fetched file can be put into current web page using the DOM
- sounds great!...
- ... but it is clunky to use, and has various browser incompatibilities
- Prototype provides a better wrapper for Ajax, so we will use that instead

A typical Ajax request

- user clicks, invoking an event handler
- handler's code creates an XMLHttpRequest object
- XMLHttpRequest object requests page from server
- server retrieves appropriate data, sends it back
- XMLHttpRequest fires an event when data arrives
  - this is often called a callback
  - you can attach a handler function to this event
- your callback event handler processes the data and displays it



AJAX基本实现步骤 - 客户端

- 创建XMLHttpRequest对象(需要考虑各浏览器兼容的问题)
- 使用XMLHttpRequest对象打开一个连接(指定连接方式<post/get>和连接地址以及是否同步)
- 设置请求的头部(请求的类型和请求的编码格式)
- 设置回调函数
- 发送请求
- 更新页面显示

Creating the XMLHttpRequest object

```
function createXMLHttpRequest( ) {
    var request = false;

    /* Does this browser support the XMLHttpRequest object? */
    if (window.XMLHttpRequest) {
        if (typeof XMLHttpRequest != 'undefined')
            /* Try to create a new XMLHttpRequest object */
            try {
                request = new XMLHttpRequest( );
            } catch (e) {
                request = false;
            }
    /* Does this browser support ActiveX objects? */
    } else if (window.ActiveXObject) {
        /* Try to create a new ActiveX XMLHTTP object */
        try {
            request = new ActiveXObject('Msxml2.XMLHTTP');
        } catch(e) {
            try {
                request = new ActiveXObject('Microsoft.XMLHTTP');
            } catch (e) {
                request = false;
            }
        }
```

XML Requests and Responses

```
if (request) {
    request.open('GET', URL, true);
    request.onreadystatechange = parseResponse;
    request.send('');
}
```

Example Handling the server's response

```
function parseResponse( ) {
    /* Is the /readyState/ 4? */
    if (request.readyState == 4) {
        /* Is the /status/ 200? */
        if (request.status == 200) {
            /* Grab the /responseText/ from the request (XMLHttpRequest) */
            var response = request.responseText;

            alert(response);
            // here is where the parsing would begin.
        } else
            alert('There was a problem retrieving the data: \n' +
                request.statusText);
        request = null;
    }
}
```

The XMLHttpRequest object's properties - readyState

- This property represents the current state that the object is in. It is an integer that takes one of the following:
  - 0 = uninitialized (The open( ) method of the object has not been called yet.)
  - 1 = loading (The send( ) method of the object has not been called yet.)
  - 2 = loaded (The send( ) method has been called, and header and status information is available.)
  - 3 = interactive (The responseText property of the object holds some partial data.)
  - 4 = complete (The communication between the client and server is finished.)

Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

Prototype's Ajax model

- construct a Prototype Ajax.Request object to request a page from a server using Ajax
- constructor accepts 2 parameters:
  - the URL to fetch, as a String,
  - a set of options, as an array of key : value pairs in {} braces (an anonymous JS object)
- hides icky details from the raw XMLHttpRequest; works well in all browsers

```
new Ajax.Request("url",
   {
      option : value,
      option : value,
      ...
      option : value
   }
);
```
JS

Prototype Ajax methods and properties

option	description
method	how to fetch the request from the server (default "post")
parameters	query parameters to pass to the server, if any
asynchronous (default true), contentType, encoding, requestHeaders	

- options that can be passed to the Ajax.Request constructor

event	description
onSuccess	request completed successfully
onFailure	request was unsuccessful
onException	request has a syntax error, security error, etc.
onCreate, onComplete, on### (for HTTP error code ###)	

- events in the Ajax.Request object that you can handle

Basic Prototype Ajax template

- attach a handler to the request's onSuccess event
- the handler takes an Ajax response object, which we'll name ajax, as a parameter

```
new Ajax.Request("url",
   {
      method: "get",
      onSuccess: functionName
   }
);
...

function functionName(ajax) {
   do something with ajax.responseText;
}
```
JS

The Ajax response object

- most commonly property is responseText, to access the fetched page

property	description
status	the request's HTTP error code (200 = OK, etc.)
statusText	HTTP error code text
responseText	the entire text of the fetched page, as a String
responseXML	the entire contents of the fetched page, as an XML DOM tree (seen later)

```
function handleRequest(ajax) {
  alert(ajax.responseText);
}
```
JS

Prototype's Ajax Updater

- Ajax.Updater fetches a file and injects its content into an element as innerHTML
- additional (1st) parameter specifies the id of element to inject into
- onSuccess handler not needed (but onFailure, onException handlers may still be useful)

```JS
new Ajax.Updater(
   "id",
   "url",
   {
     method: "get"
   }
);
```

Creating a POST request

- Ajax.Request can also be used to post data to a web server
- method should be changed to "post" (or omitted; post is default)
- any query parameters should be passed as a parameters parameter
  - written between { } braces as a set of name : value pairs (another anonymous object)
  - get request parameters can also be passed this way, if you like

```
new Ajax.Request("url",
   {
     method: "post",    // optional
     parameters: { name: value, name: value, ..., name: value },
     onSuccess: functionName,
     onFailure: functionName,
     onException: functionName
   }
);
```
JS

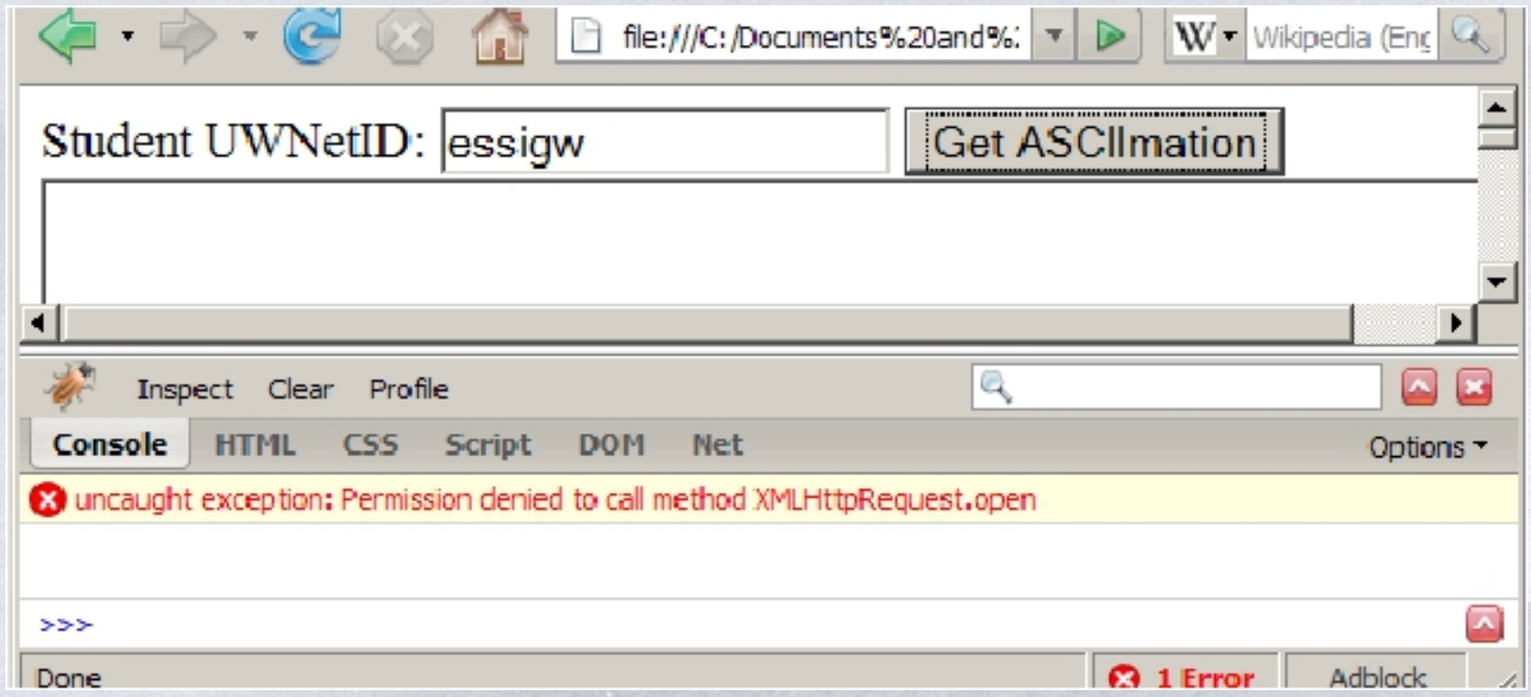
Outline

- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

Ajax Risk References

- Bookmarking Issues
- Back and Forward Button Problems
- Security Risks
- Search Engines

XMLHttpRequest security restrictions



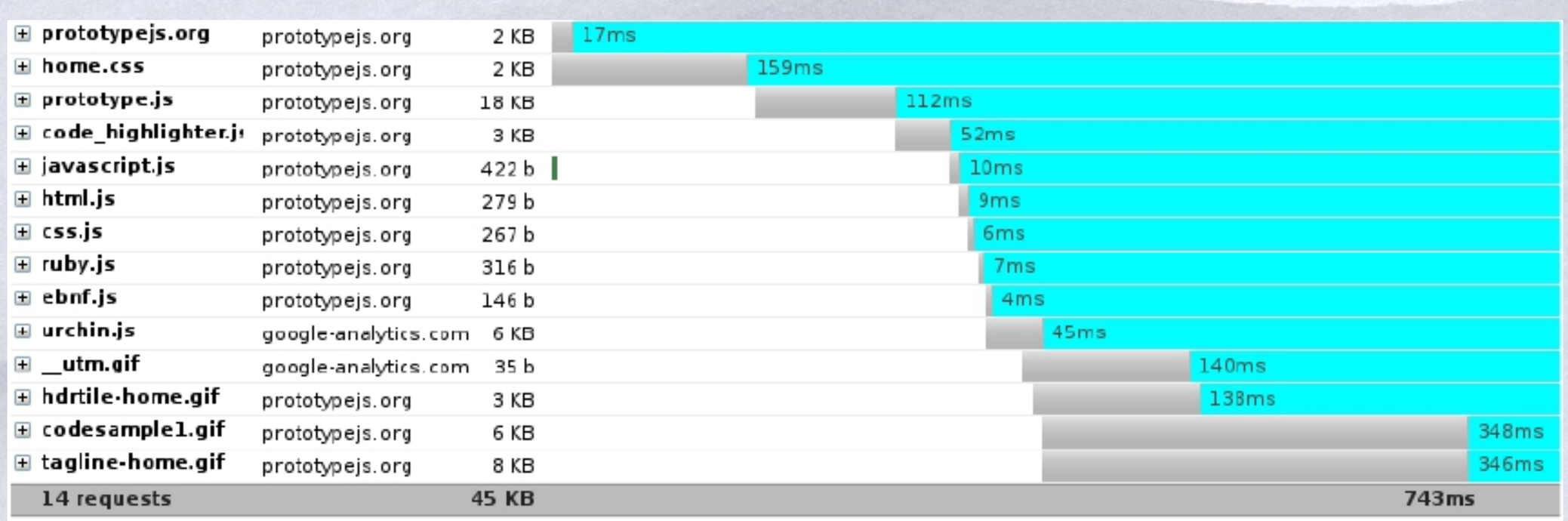
- cannot be run from a web page stored on your hard drive
- can only be run on a web page stored on a web server
- XSS

Same Origin Policy

- The Same Origin Policy (SOP) limits browsers only fetching content from the same origin site.
  - except resources: images, scripts, videos, etc.
- The Same Origin Policy (SOP) essentially mandates that Ajax requests cannot access another fully qualified domain than the page from which they're running.
  - even the same domain on another port!
- SOP is all about XSS (cross site script)

Two-request limit

- The HTTP 1.1 (RFC 2616) recommends that a single-user client SHOULD NOT maintain more than 2 connections with any server or proxy.
- Most browsers (including IE) abide by this rule



Outline

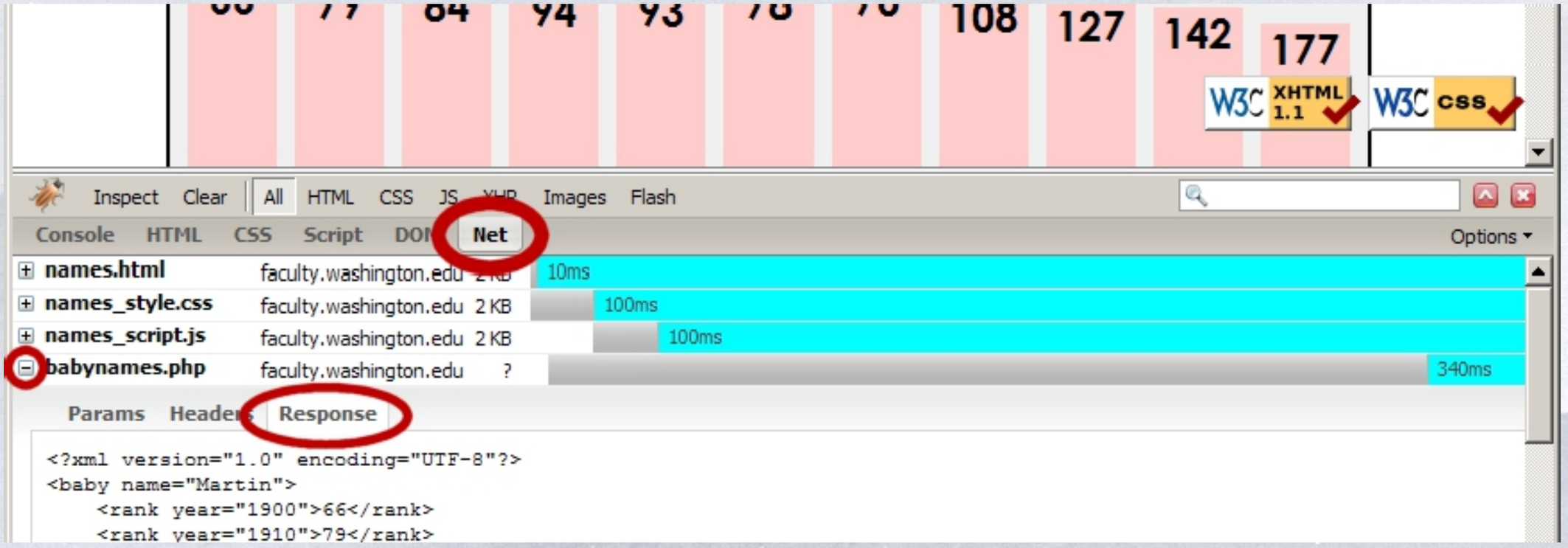
- Rich Internet Applications (RIAs)
- Ajax
- XMLHttpRequest
- Ajax in Prototype
- Limits of Ajax
- Debugging Ajax

Handling Ajax errors

```
new Ajax.Request("url",
   {
     method: "get",
     onSuccess: functionName,
     onFailure: ajaxFailure,
     onException: ajaxFailure
   }
);
  ...
function ajaxFailure(ajax, exception) {
  alert("Error making Ajax request:" +
        "\n\nServer status:\n" + ajax.status + " " + ajax.statusText +
        "\n\nServer response text:\n" + ajax.responseText);
  if (exception) {
    throw exception;
  }
}
```
JS

- for user's (and developer's) benefit, show an error message if a request fails

Debugging Ajax code



- Net tab shows each request, its parameters, response, any errors
- expand a request with + and look at Response tab to see Ajax result

Recommendations

- If you're serious about RIAs, climb the direct manipulation hill.
- Don't limit yourself to Thin AJAX.
- AJAX sweet spot: Applications that are part of the web.
- AJAX is an implementation alternative for applications deployed over the web.

References

- Ajax: The Definitive Guide.Anthony T. Holdener III. O'Reilly Media.

Thanks!!!