

# Asynchronous web apps using Ajax



Children with

Advertising F

google will		<a href="#">Advanced Search</a>
		<a href="#">Preferences</a>
		<a href="#">Language Tools</a>
google will eat itself	434,000 results	
google will not load	10,500,000 results	
google will take over the world	16,000,000 results	
google will not open	81,000,000 results	by Google.
google will rule the world	12,500,000 results	
google will not search for chuck norris	266,000 results	
google will pay you to type	8,520,000 results	<a href="#">Indonesia</a>
google wills	2,190,000 results	
google will you marry me	429,000 results	
google will harm your computer	245,000 results	
		<a href="#">close</a>

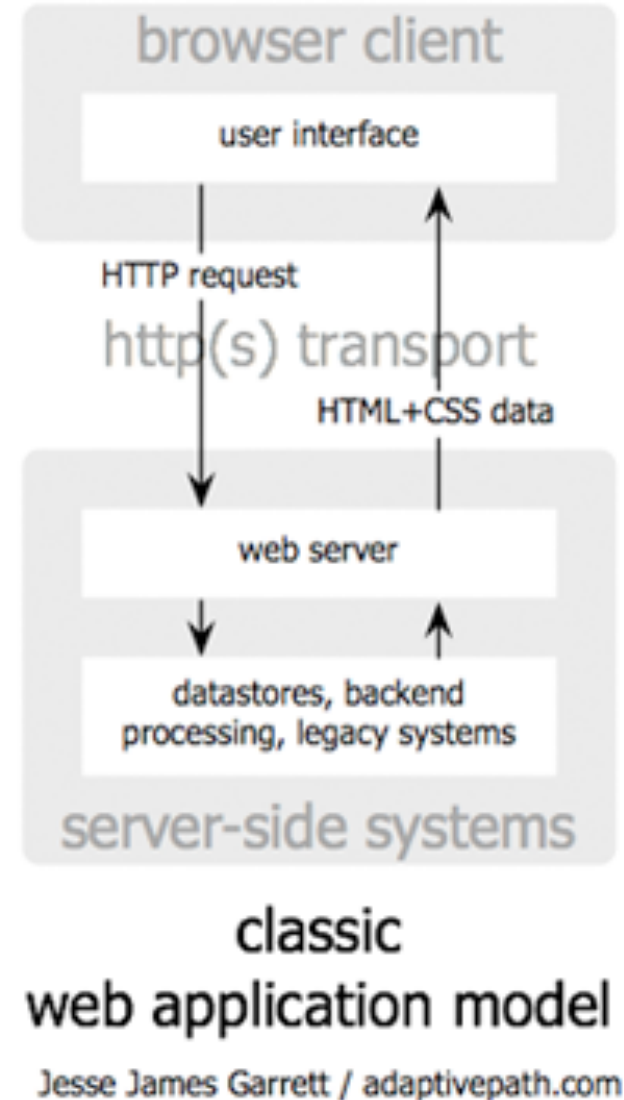
# Overview

- Ajax: Asynchronous JavaScript and XML
  - History
  - Basic idea
  - Examples of what it can do
- How it works
  - XMLHttpRequest object
    - Methods and properties
  - Example application



# History of Ajax

- In the age before Ajax:
  - Client requests a page
  - Server delivers page
  - **Client-side JavaScript** can add **some interactivity**
    - e.g. Validate forms, pop up alert messages, falling Tetris blocks
    - Whatever the page needs has to be contained in the served page
  - To obtain **new information**, requires **some user action**
    - e.g. Hit button to refresh email



# A small change...



- 1999

- IE5: Microsoft introduces XMLHttpRequest ActiveX control
- Mozilla, Safari and other browser follow suit:
  - XMLHttpRequest JavaScript object
  - IE7: Microsoft eventually adopts XMLHttpRequest model
- Key idea: allows pages to go back for more data
  - Separate from user actions on the page
  - Enables new responsive, user-friendly web apps

- 2000

- Microsoft uses in Outlook Web Access

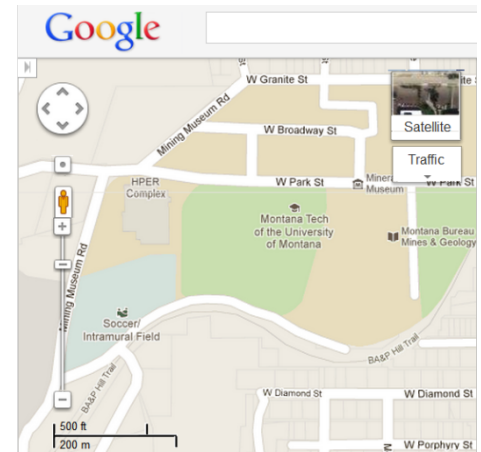


- 2002

- Oddpost uses for web email, bought by Yahoo!

# And then...

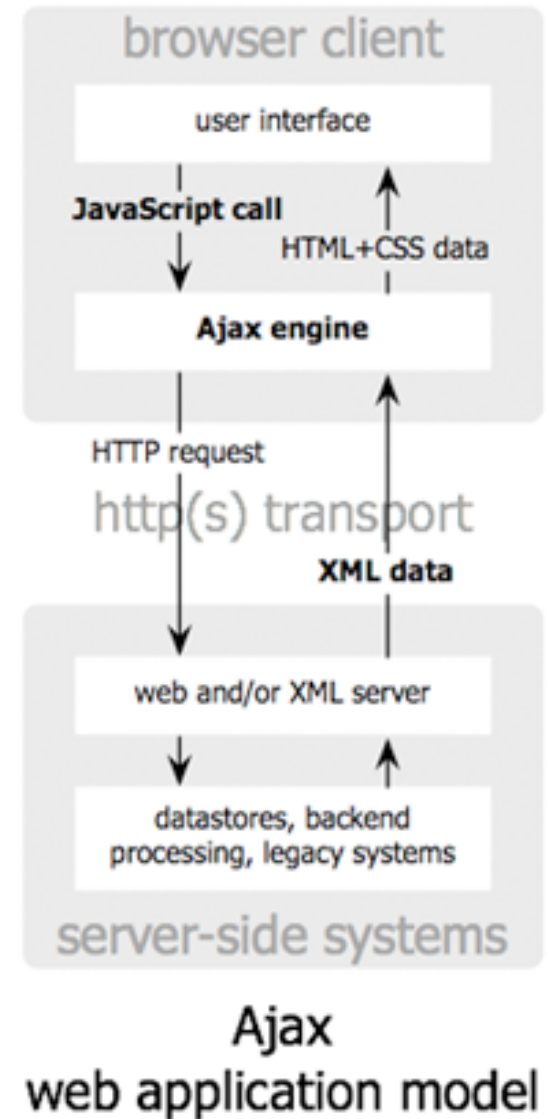
- 2004, 2005
  - Google uses in Gmail and Maps
  - The world takes notice
- 2005
  - Jesse James Garrett, coins term Ajax
  - Needed something to call combination of:
    - XHTML and CSS
    - Dynamic display and interaction using the DOM
    - Data interchange using XML
    - Asynchronous data retrieval using XMLHttpRequest
    - JavaScript binds things together
- 2006
  - W3C publishes a working draft



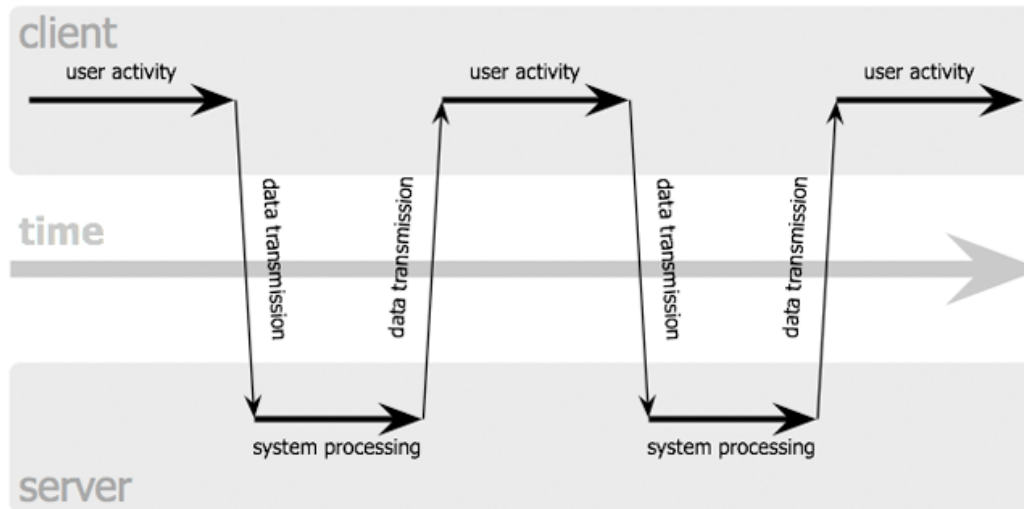
*Jesse James Garrett*

# Ajax model

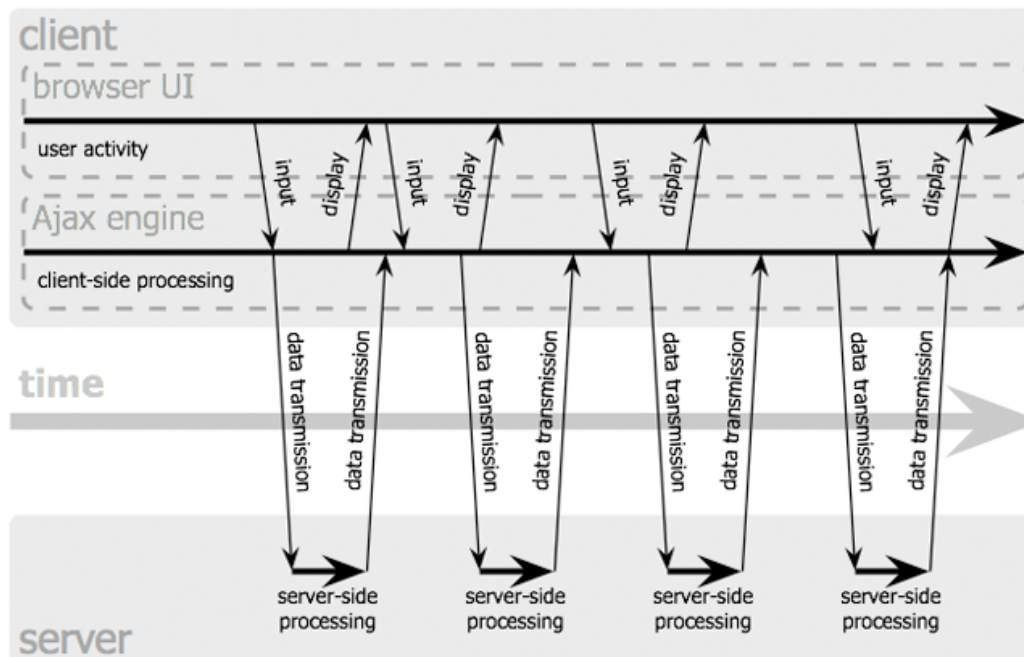
- **Eliminates start-stop-start-stop**
  - Ajax engine communicates with server periodically
    - Based on user events, e.g. key up
    - Based on a timer
  - Meanwhile user continues interacting with page
  - HTTP requests by Ajax engine happen asynchronously
  - Repopulate only a portion
    - Instead of re-rendering entire page from scratch



## classic web application model (synchronous)



## Ajax web application model (asynchronous)



# Examples of Ajax apps

- Google maps: <http://maps.google.com>
- Google suggest: <http://www.google.com/>
- Amazon suggest: <http://www.amazon.com/>
- Lyrics/band search: <http://lyricsfly.com/>
- Regular expression editor: <http://www.rexv.org/>
- Wikipedia browser: <http://gollum.easycp.de>
- Network tools: <http://www.ajaxutils.com/>
- Multi-player game: <http://www.travians.com/>



# How it works

- XMLHttpRequest object

- Allows JavaScript to retrieve data from web server

- Same origin policy

- Only can request **data from same domain** of served page
- But there are ways around this (stay tuned)

```
var hr = new ActiveXObject("Microsoft.XMLHTTP");
```

Before IE7 (2006), creating object using ActiveX.

```
var hr = new XMLHttpRequest();
```

Creating object in Mozilla, Firefox, Safari, Chrome, Opera.

```
var hr;  
if (window.XMLHttpRequest)  
    hr = new XMLHttpRequest();  
else  
    hr = new ActiveXObject("Microsoft.XMLHTTP");
```

Cross-browser code for creating the object.

# XMLHttpRequest methods

**open(method, url, async, user, password)**

Initializes the XMLHttpRequest object

method	"GET", "POST", ...
url	URL of page to obtain, relative or absolute. Must be on same domain as page that called open().
async	Is request asynchronous? Optional, default = true
user	Username for authentication, optional
password	Password for authentication, optional

**setRequestHeader(name, value)**

Set HTTP headers that are sent with request

name	Text string name of the header field.
value	Text string value of the header field.

**send(data)**

Actually fires off the HTTP request

data	Any payload in the HTTP request (e.g. POST data), optional
------	--

# XMLHttpRequest methods

**abort()**

Cancels the current request

**getResponseHeader(name)**

Retrieve value of a specified HTTP header

name

Text string name of the header field.

**getAllResponseHeaders()**

Retrieve all the response name/value pairs

**overrideMimeType(mime)**

Overrides the MIME type of HTTP response

mime

Text string of new MIME type (e.g. "text/xml")

# XMLHttpRequest attributes

- **onreadystatechange**
  - Specifies method that is called on every state change of object
  - Usually the event handler for your application
- **readyState**
  - Status of the object, changes from 0 to 4:
    - 0: request not initialized
    - 1: server connection established
    - 2: request received
    - 3: processing request
    - 4: request finished and response ready
- **status**
  - HTTP response code, 200 = OK, 404 = not found, ....
- **statusText**
  - The string representation of the HTTP response, e.g. "Not Found"

# XMLHttpRequest attributes

- `responseText`
  - Response from the server as a string
- `responseXML`
  - Response from the server as XML
  - DOM object that you can call methods on

# Putting it all together: example 1

- **Goal: Suggestions based on typed letters**
  - Two components:
    - **keyword.html** - HTML + JavaScript interface
    - **keyword.php** - Returns list of matching countries
  - HTML page with JavaScript
    - `<input>` element for typing part of **country name**
    - Call JavaScript function every time key is pressed
    - Make HTTP request, GET param based on current text
    - When result arrives, display in `<div>` element
  - PHP page
    - Using GET parameter, match against a list of data
    - Return result as HTML text

# Putting it all together: example 2

- **Goal: Stock price update ever second**
  - Two components:
    - **stock.html** - HTML + JavaScript interface
    - **stock.php** - Returns list of matching countries
  - HTML page with JavaScript
    - `<input>` element for input of **stock ticker symbol**
    - Call JavaScript function every 1000ms
    - Make HTTP request, GET param based on current text
    - When result arrives, display in `<div>` element
  - PHP page
    - Using GET parameter, match against a list of data
    - Return result as HTML text

# Cross-domain Ajax

- Same origin policy

- Prevents security problems

- Page at [example.com](http://example.com) sharing data with [evil.com](http://evil.com)

- But often we want to do this for good

- e.g. Retrieving translation results from Bing web API

- How to allow cross-domain Ajax:

- Run a **proxy** on your server

- [http://example.com/cgi-bin/proxy?req=bing\\_query\\_details](http://example.com/cgi-bin/proxy?req=bing_query_details)

- **Cross-origin resource sharing**

- Use a `<script>` tag with a source URL that returns a **JavaScript callback** function

- e.g. The approach taken in the Bing web API samples





# Cross-domain with JSON callback

```
function Search()
{
    var requestStr = "http://api.bing.net/json.aspx?"
        + "AppId=" + AppId
        + "&Query=rabbit%20site:wikipedia.org"
        + "&Sources=Web"
        + "&Version=2.0"
        + "&Web.Count=1"
        + "&Web.Offset=0"
        + "&JsonType=callback"
        + "&JsonCallback=SearchCompleted";

    var commScript      = document.createElement("script");
    commScript.src      = requestStr;
    commScript.type     = "text/javascript";
    commScript.charset = "UTF-8";

    if(document.head)
        head = document.head;
    else if(document.getElementsByTagName)
        head = document.getElementsByTagName('head')[0];
    else
        document.write("An error occured");
    head.appendChild(commScript);
}
```

# Result from Bing request

```
if(typeof SearchCompleted == 'function')

SearchCompleted(

{"SearchResponse":
  {"Version":"2.0",
   "Query":
    {"SearchTerms":"rabbit site:wikipedia.org"},
    "Web":
      {"Total":5280000,
       "Offset":0,
       "Results":
        [
          {"Title":"Rabbit - Wikipedia, the free encyclopedia",
           "Description":"Rabbits are small mammals in the family
Leporidae of the order Lagomorpha, found in several parts of the world. There
are eight different genera in the family ...",
           "Url":"http://en.wikipedia.org/wiki/Rabbit",
           "DisplayUrl":"en.wikipedia.org/wiki/Rabbit",
           "DateTime":"2012-02-05T17:40:00Z"}
        ]
      }
    }
  }
} /* pageview_candidate */;
```

# Other Ajax problems

- Requires JavaScript & XMLHttpRequest support
  - Trouble on mobile devices?
- Breaks browser back button, bookmarking
  - Solutions involve using # tag, HTML 5 API
- Web crawlers don't use JavaScript
  - If Ajax used to expose content, won't be indexed
- Lots of HTTP requests to server
  - Increases load on your server and network
    - Especially if you proxy for cross-domain requests
- Tricky asynchronous interface
  - Multiple threads of execution, harder to get right

# Summary

- Ajax – Asynchronous JavaScript and XML
  - Not really a new technology
  - A style combining existing technologies:
    - HTML + JavaScript
    - XMLHttpRequest object
      - Asynchronously makes HTTP requests
    - Use DOM to update client's page
  - XML is actually not required
    - Results could be plain text, comma separated, JSON, ...
  - Provides many of useful features seen on the web
    - e.g. Google/Amazon/... auto-complete in search field