

# Intro to Web

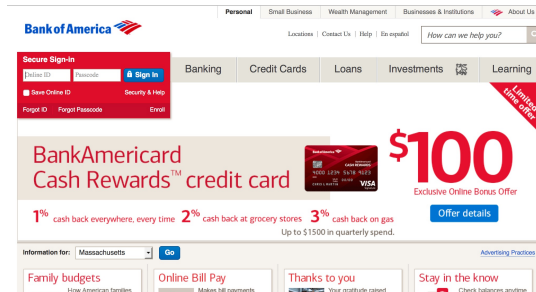
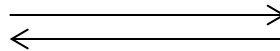
**CS 161 Fall 2021 - Lecture 19**

# What is the Web?

A platform for deploying applications and sharing information,  
*portably and securely*

client browser

web server



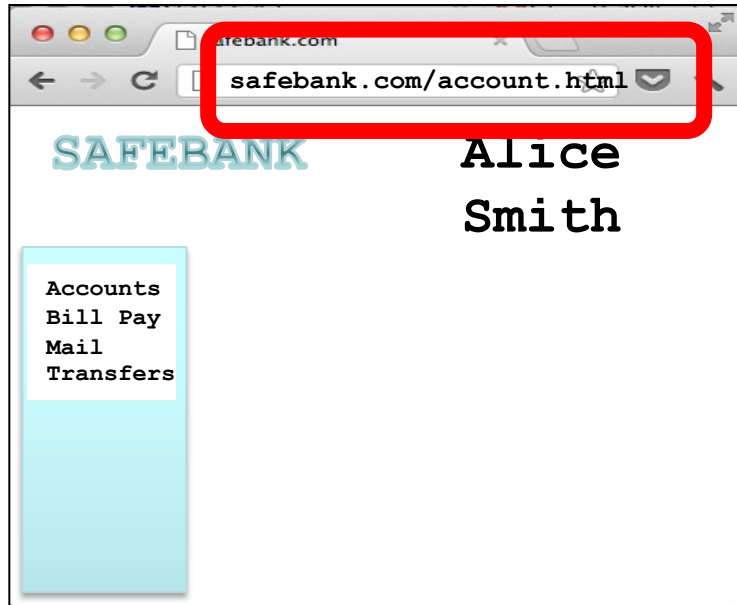
Bank of America 

# HTTP

## (Hypertext Transfer Protocol)

A common data communication protocol on the web

CLIENT BROWSER



WEB SERVER

### HTTP REQUEST:

```
GET /account.html HTTP/1.1  
Host: www.safebank.com
```



### HTTP RESPONSE:

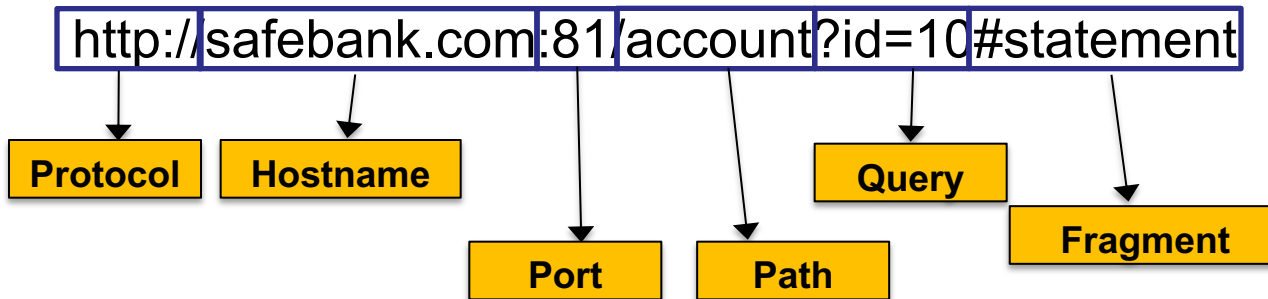
```
HTTP/1.0 200 OK  
<HTML> . . . </HTML>
```



# URLs

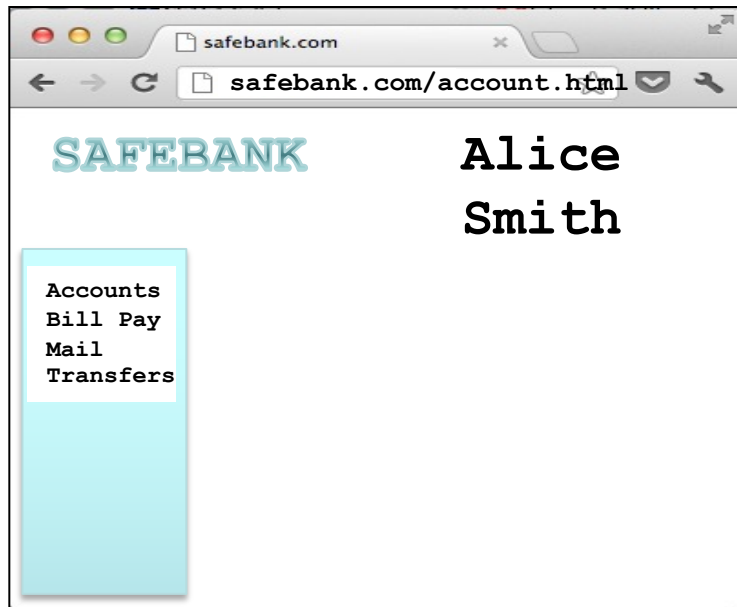
Global identifiers of network-retrievable resources

**Example:**



# HTTP

CLIENT BROWSER



WEB SERVER

HTTP REQUEST:

```
GET /account.html HTTP/1.1  
Host: www.safebank.com
```



HTTP RESPONSE:

```
HTTP/1.0 200 OK  
<HTML> . . . </HTML>
```



# HTTP Request

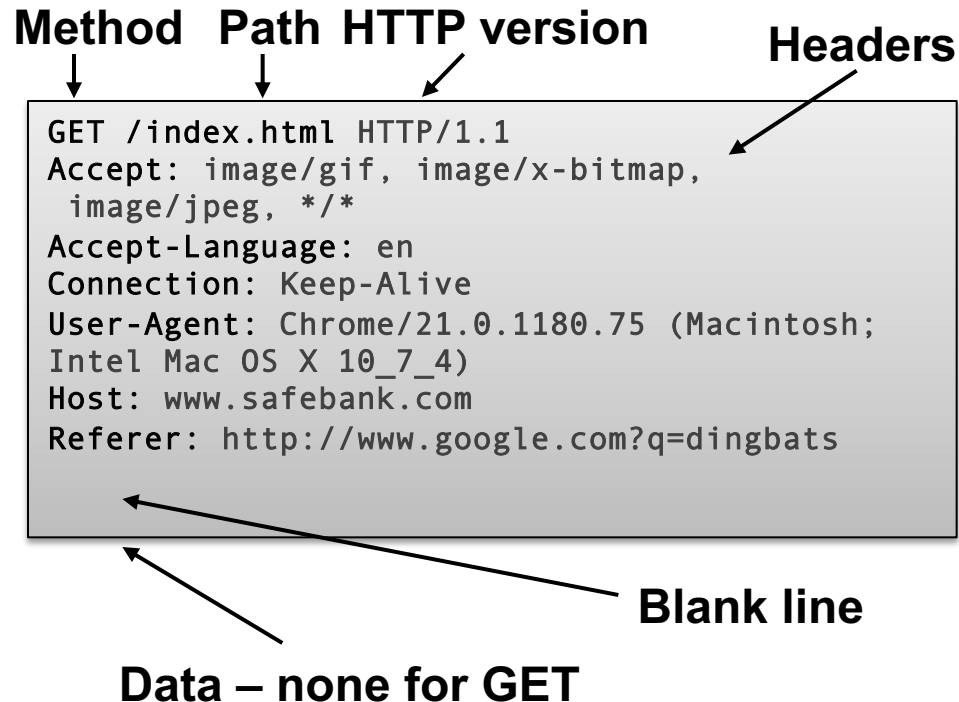
GET: no  
side effect  
POST:  
possible  
side effect

**Method Path HTTP version Headers**

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap,
image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh;
Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats
```

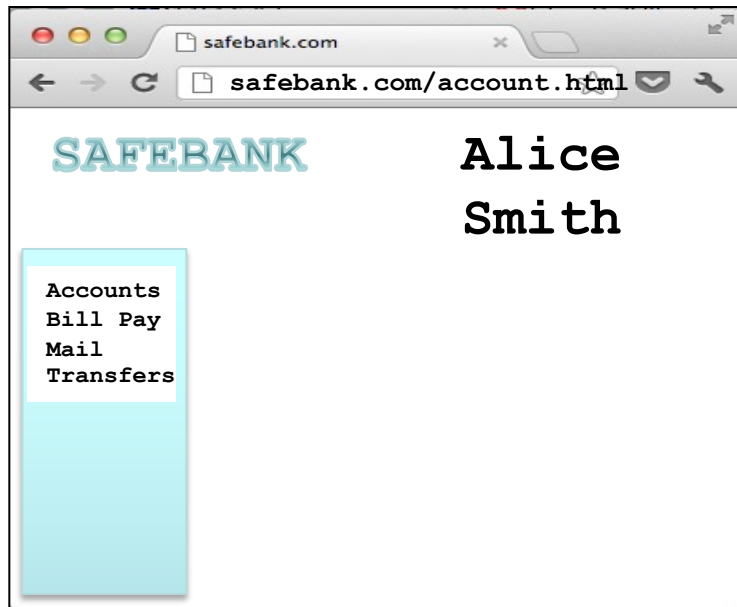
**Blank line**

**Data – none for GET**

The diagram illustrates the structure of an HTTP GET request. It shows a sequence of lines: the request line (GET /index.html HTTP/1.1), followed by several header lines (Accept, Accept-Language, Connection, User-Agent, Host, Referer), a blank line, and a data section. Labels with arrows point to these parts: 'Method' points to 'GET', 'Path' points to '/index.html', 'HTTP version' points to 'HTTP/1.1', 'Headers' points to the first header line, 'Blank line' points to the empty line after the headers, and 'Data – none for GET' points to the area below the blank line.

# HTTP

CLIENT BROWSER



WEB SERVER

HTTP REQUEST:

GET /account.html HTTP/1.1  
Host: www.safebank.com

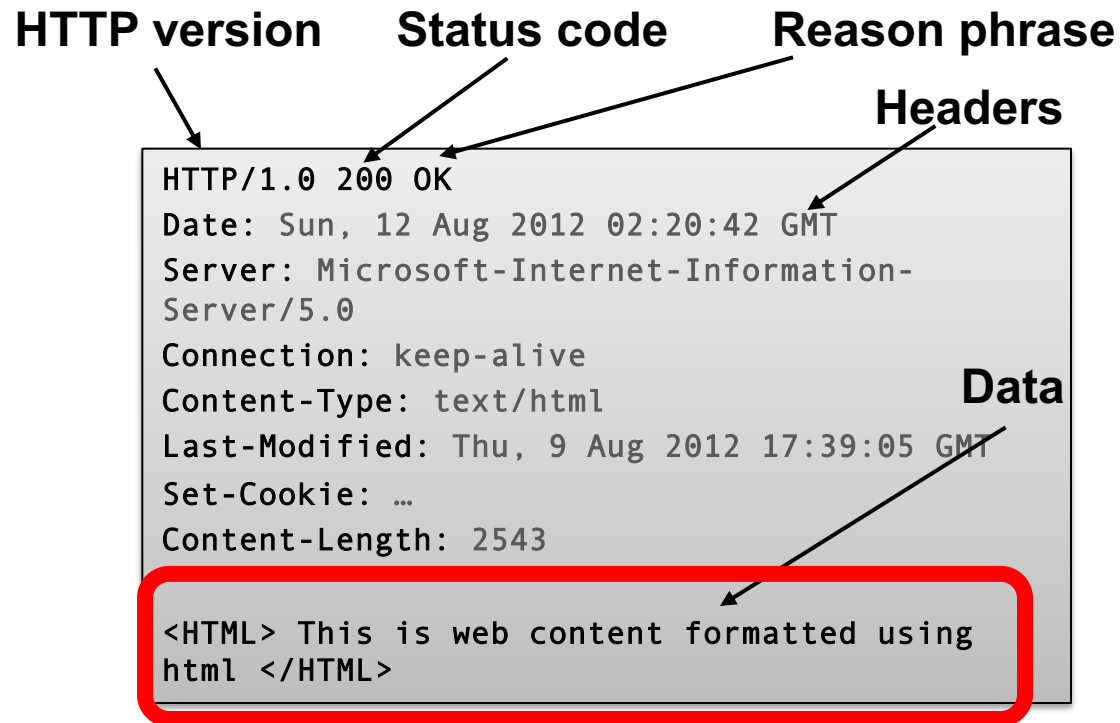


HTTP RESPONSE:

HTTP/1.0 200 OK  
<HTML> . . . </HTML>

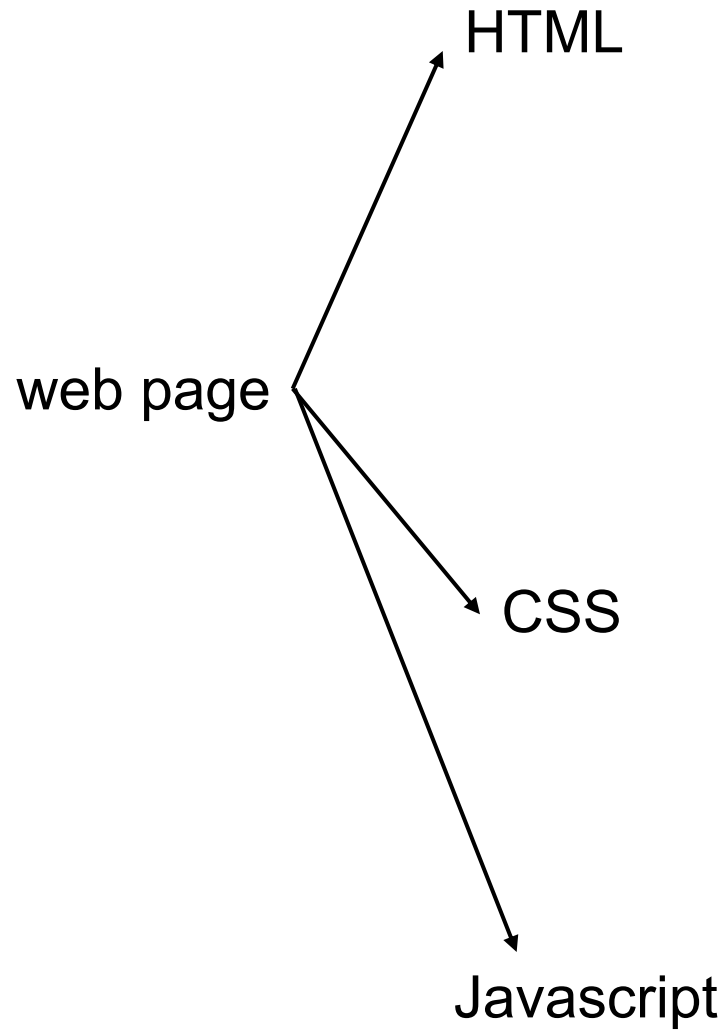


# HTTP Response



Can be a webpage

# Web page



# HTML

A language to create structured documents

One can embed images, objects, or create interactive forms

**index.html**

```
<html>
  <body>
    <div>
      foo
      <a href="http://google.com">Go to Google!</a>
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```

# CSS (Cascading Style Sheets)

Style sheet language used for describing the presentation of a document

## **index.css**

```
p.serif {  
  font-family: "Times New Roman", Times, serif;  
}  
p.sansserif {  
  font-family: Arial, Helvetica, sans-serif;  
}
```

# Javascript

Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

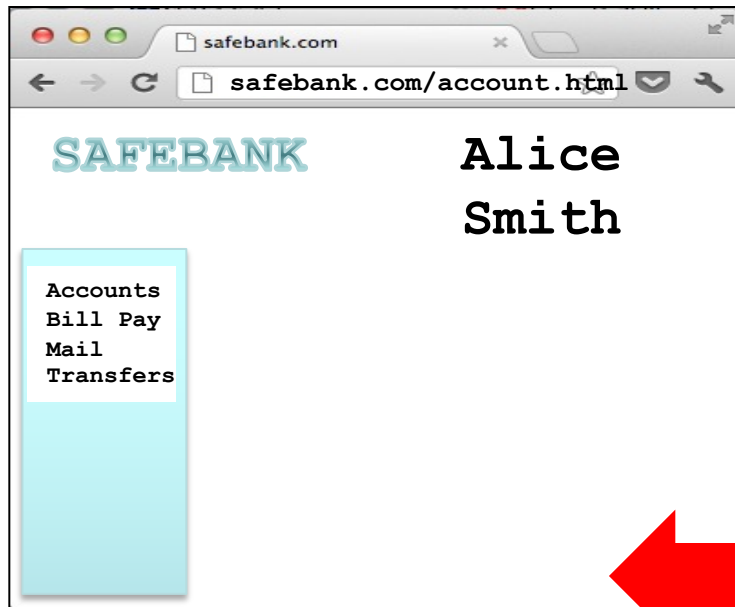
Supported by all web browsers

```
<script>
function myFunction() {
document.getElementById("demo").innerHTML = "Text changed.";
}
</script>
```

**Very powerful!**

# HTTP

CLIENT BROWSER



WEB SERVER

HTTP REQUEST:

GET /account.html HTTP/1.1  
Host: www.safebank.com



HTTP RESPONSE:

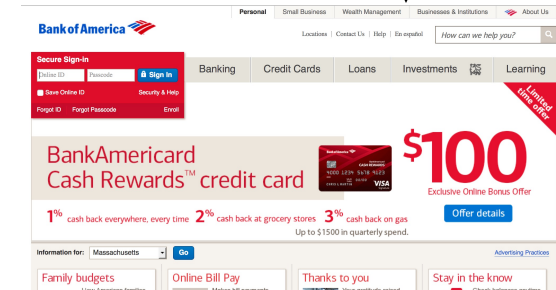
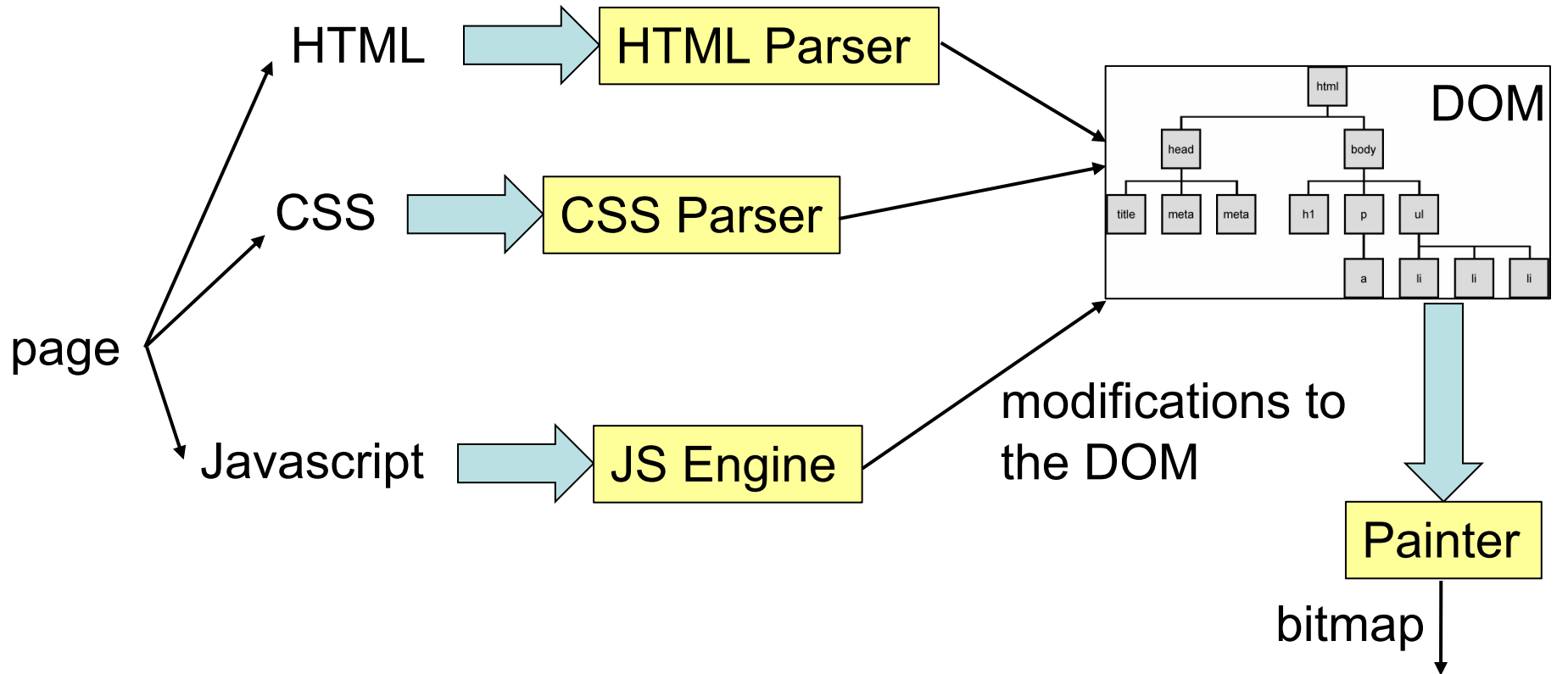
HTTP/1.0 200 OK  
<HTML> . . . </HTML>



**webpage**



# Page rendering

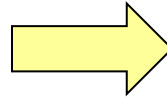


# DOM (Document Object Model)

a cross-platform model for representing and interacting with objects in HTML

## HTML

```
<html>
  <body>
    <div>
      foo
    </div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```

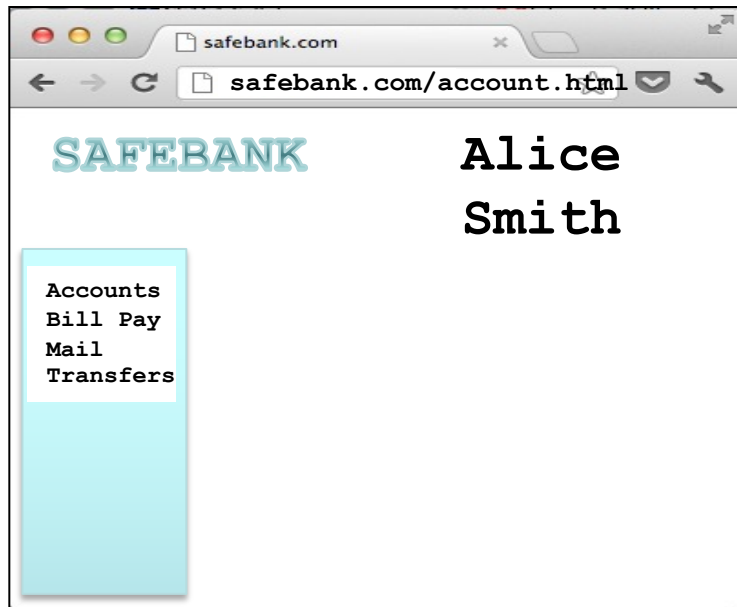


## DOM Tree

```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
      |-> Form
        |-> Text-box
        |-> Radio Button
        |-> Check Box
```

# Web & HTTP 101

CLIENT BROWSER



WEB SERVER

HTTP REQUEST:

GET /account.html HTTP/1.1  
Host: www.safebank.com



HTTP RESPONSE:

HTTP/1.0 200 OK  
<HTML> . . . </HTML>



# The power of Javascript

Get familiarized with it so that you can think of all the attacks one can do with it

# What can you do with Javascript?

Almost anything you want to the DOM!

A JS script embedded on a page can modify in almost arbitrary ways the DOM of the page. The same happens if an attacker manages to get you load a script into your page.

w3schools.com has nice interactive tutorials:  
<https://www.w3schools.com/w3css/tryit.asp>

# Example of what Javascript can do...

Can change HTML content:

```
<p id="demo">JavaScript can change HTML content.</p>
```

```
<button type="button"  
onclick="document.getElementById('demo').innerHTML =  
'Hello JavaScript!'">  
    Click Me!</button>
```

DEMO from [w3schools.com](http://w3schools.com)

# Other examples

Can change images

Can change style of elements

Can hide elements

Can unhide elements

Can change cursor

# Other example: can access cookies

Will learn later that cookies are useful for authentication.

JS can read cookie:

```
var x = document.cookie;
```

Change cookie with JS:

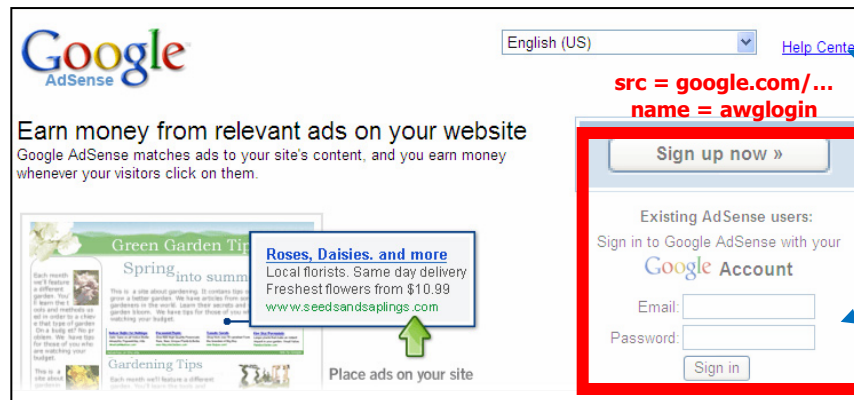
```
document.cookie = "username=John Smith; expires=Thu,  
18 Dec 2013 12:00:00 UTC; path="/;
```

# Frames

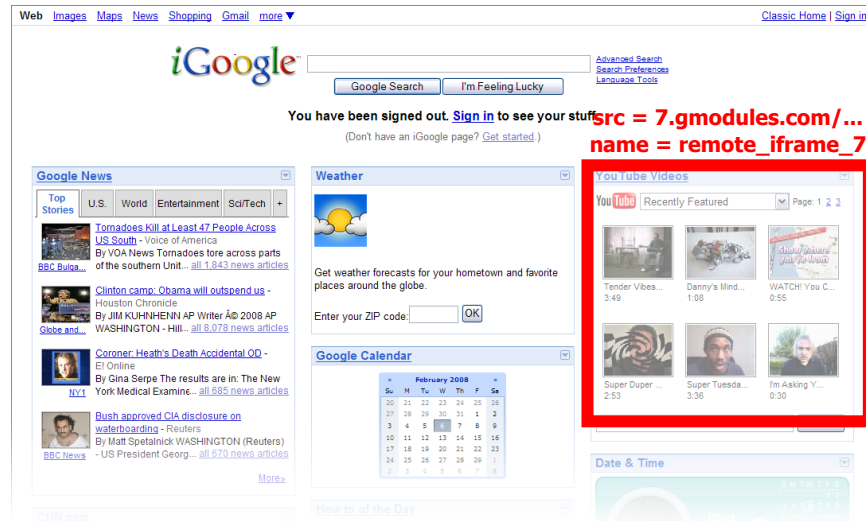
# Frames

- Enable embedding a page within a page

```
<iframe src="URL"></iframe>
```



# Frames



- Modularity
  - Brings together content from multiple sources
  - Client-side aggregation
- Delegation
  - Frame can draw only on its own rectangle

# Frames

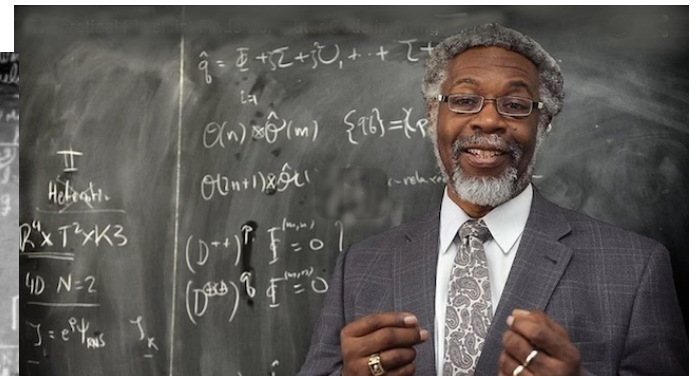
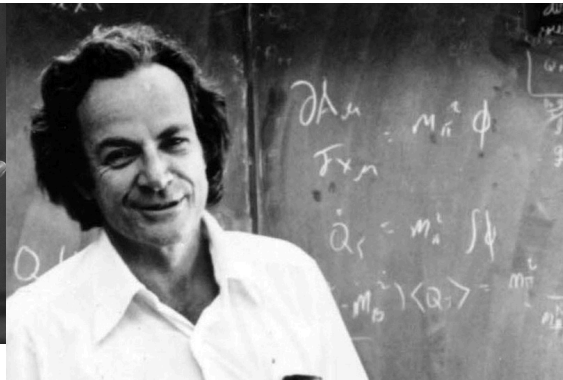
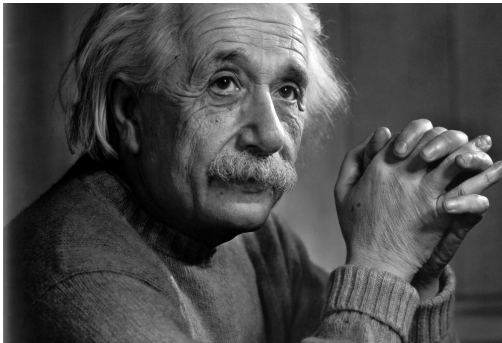
- Outer page can specify only sizing and placement of the frame in the outer page
  - demo
- Frame isolation: Our page cannot change contents of inner page, inner page cannot change contents of outer page

# Web security



# A historical perspective

- The web is an example of “bolt-on security”, the security was added as an after thought
- Originally, the web was invented to allow physicists to share their research papers
  - Only textual web pages + links to other pages;  
no threat model to speak of



# The web became complex and adversarial quickly

- Then we added embedded images
  - Crucial decision: a page can embed images loaded from another web server
- Then, Javascript, dynamic HTML, AJAX, CSS, frames, audio, video, ...
- Today, a web site is a distributed application
- Attackers have various motivations

**Web security is a challenge!**

# Desirable security goals

- **Integrity:** malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites
- **Confidentiality:** malicious web sites should not be able to learn confidential information from my computer or other web sites
- **Privacy:** malicious web sites should not be able to spy on me or my activities online
- **Availability:** attacker cannot make site unavailable

# Security on the web

- Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer
  - Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.

# Security on the web

- Risk #1: we don't want a malicious site to be able to trash my files/programs on my computer
  - Browsing to `awesomevids.com` (or `evil.com`) should not infect my computer with malware, read or write files on my computer, etc.
- Defense: Javascript is sandboxed; try to avoid security bugs in browser code; privilege separation; automatic updates; etc.

# Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account

# Security on the web

- Risk #2: we don't want a malicious site to be able to spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account
- Defense: **the same-origin policy**
  - A security policy grafted on after-the-fact, and enforced by web browsers

# Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access

# Security on the web

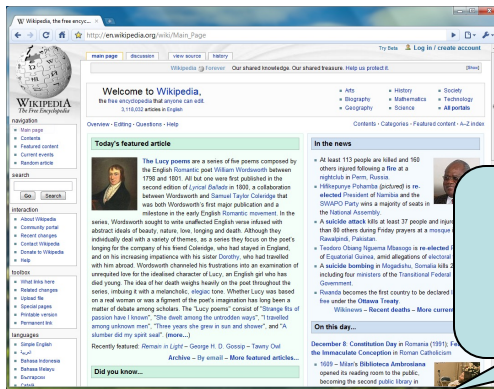
- Risk #3: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security

Same-origin policy

# Same-origin policy

- Each site in the browser is isolated from all others

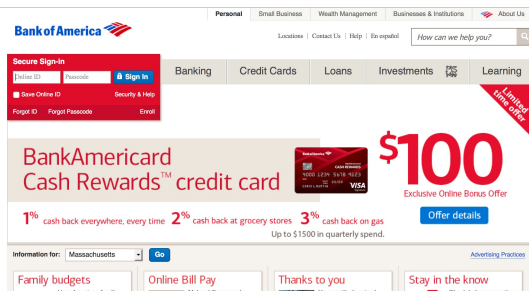
browser:



security  
barrier



wikipedia.org

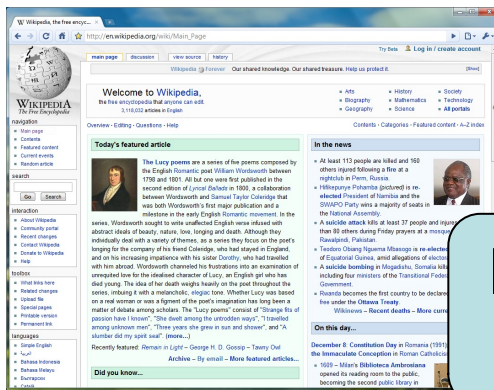


mozilla.org

# Same-origin policy

- Multiple pages from the same site are not isolated

browser:



No security barrier



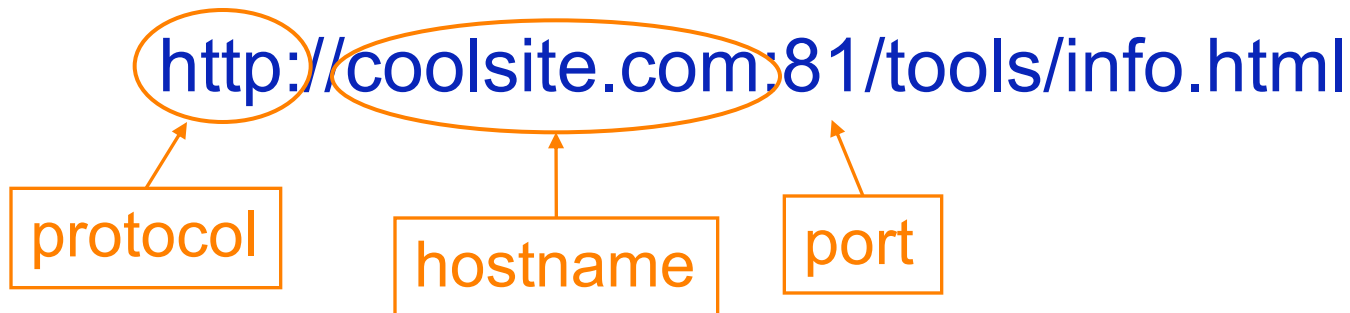
wikipedia.org



wikipedia.org

# Origin

- Granularity of protection for same origin policy
- Origin = (protocol, hostname, port)



- It is **string matching**! If these match, it is same origin, else it is not. Even though in some cases, it is logically the same origin, if there is no match, it is not

# Same-origin policy

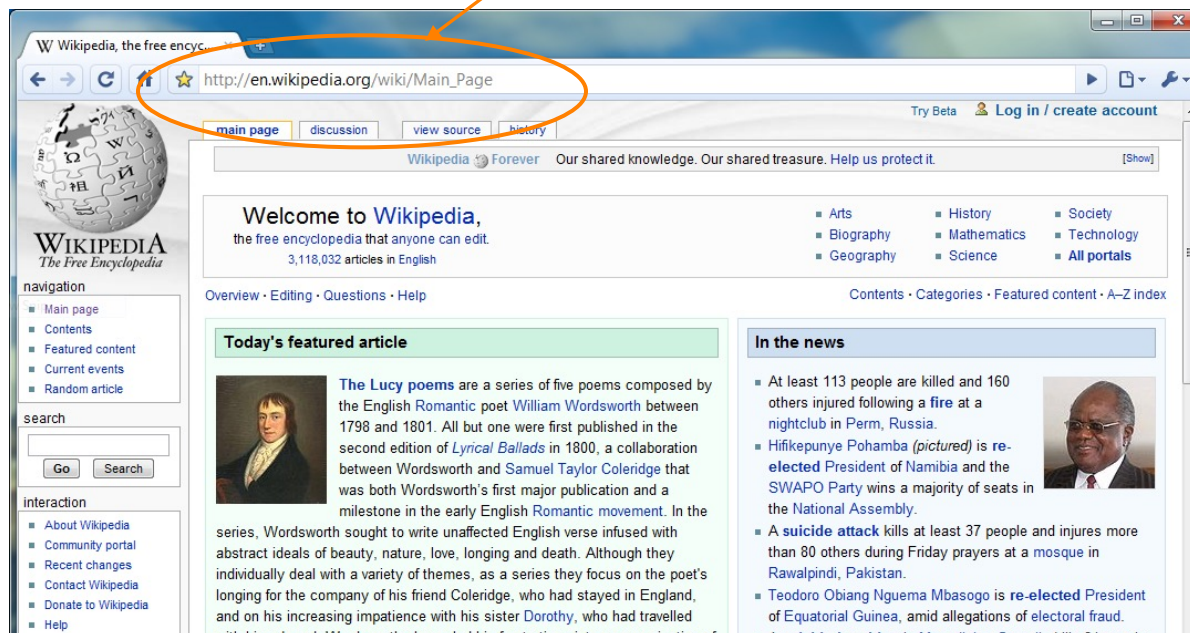
One origin should not be able to access the resources of another origin

Javascript on one page cannot read or modify pages from different origins

# Same-origin policy

- The origin of a page is derived from the URL it was loaded from

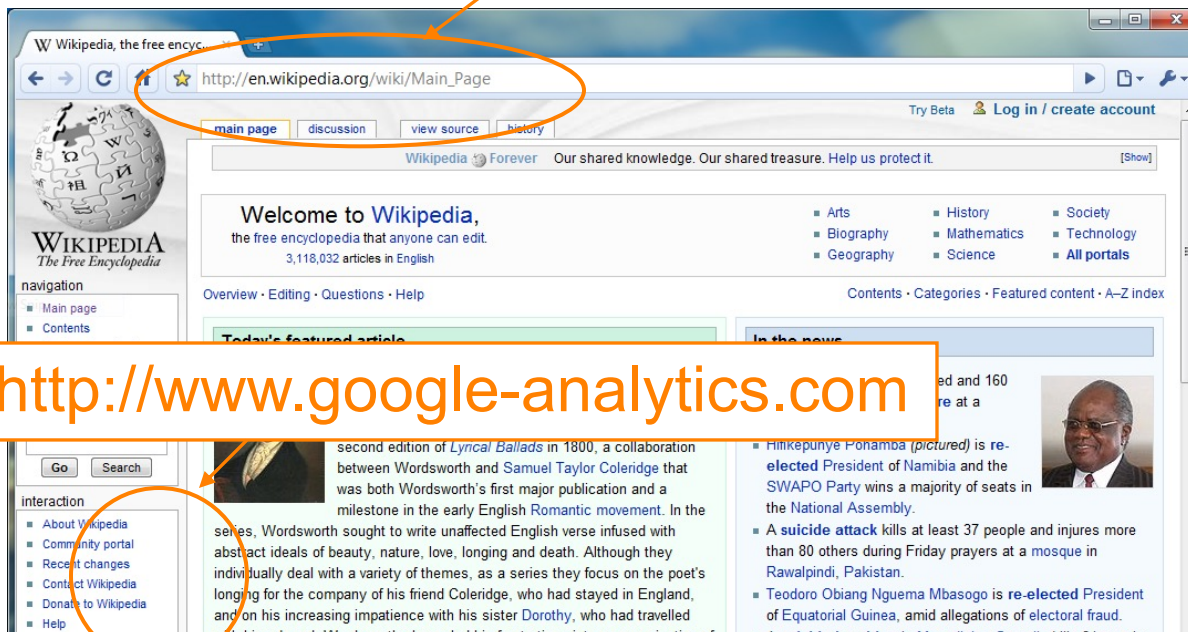
<http://en.wikipedia.org>



# Same-origin policy

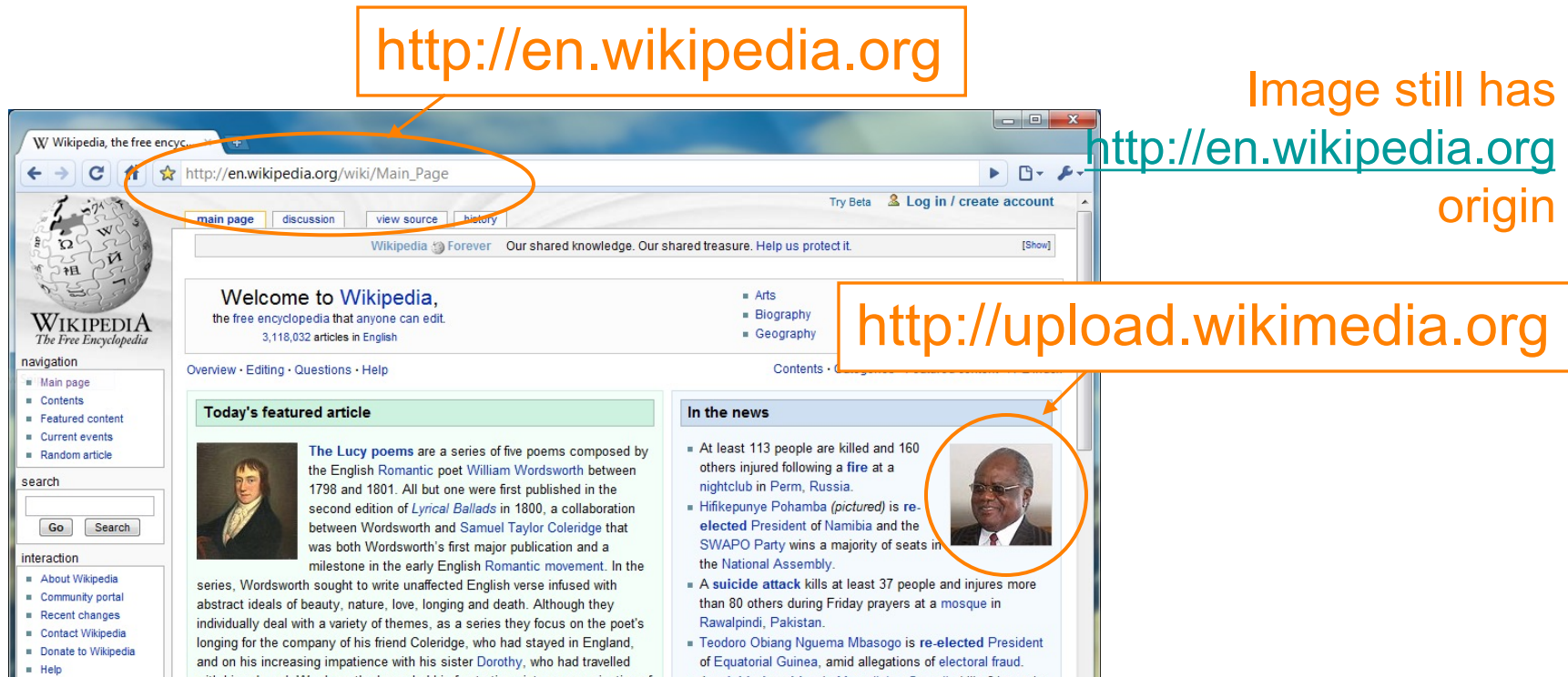
- The origin of a page is derived from the URL it was loaded from
- Special case: Javascript runs with the origin of the page that loaded it

<http://en.wikipedia.org>



# Origins of other components

- `<img src="">` the image is “copied” from the remote server into the new page so it has the origin of the embedding page (like JS) and not of the remote origin



# Origins of other components

- `iframe`: origin of the URL from which the `iframe` is served, and not the loading website.

# Exercises: Same origin?

Originating document	Accessed document
<a href="http://wikipedia.org/a/">http://wikipedia.org/a/</a>	<a href="http://wikipedia.org/b/">http://wikipedia.org/b/</a>
<a href="http://wikipedia.org/">http://wikipedia.org/</a>	<a href="http://www.wikipedia.org/">http://www.wikipedia.org/</a>
<a href="http://wikipedia.org/">http://wikipedia.org/</a>	<a href="https://wikipedia.org/">https://wikipedia.org/</a>
<a href="http://wikipedia.org:81/">http://wikipedia.org:81/</a>	<a href="http://wikipedia.org:82/">http://wikipedia.org:82/</a>
<a href="http://wikipedia.org:81/">http://wikipedia.org:81/</a>	<a href="http://wikipedia.org/">http://wikipedia.org/</a>

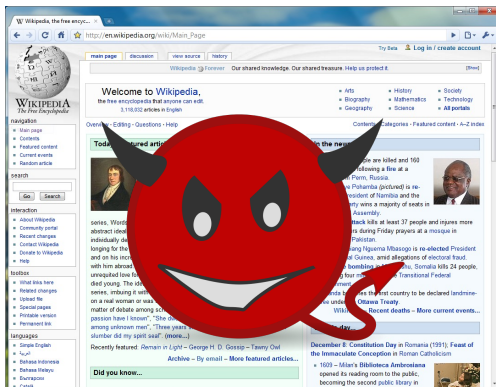


except

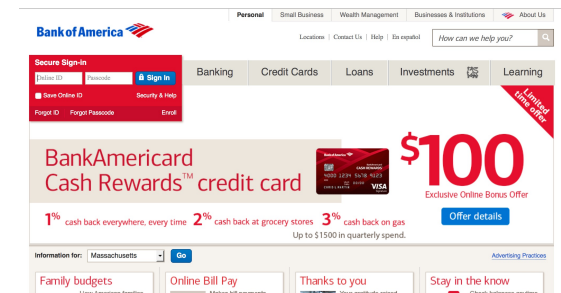


# Cross-origin communication

- Allowed through a narrow API: **postMessage**
- Receiving origin decides if to accept the message based on origin (whose correctness is enforced by browser)



**postMessage**  
("run this  
script",  
script)



**Check origin, and request!**