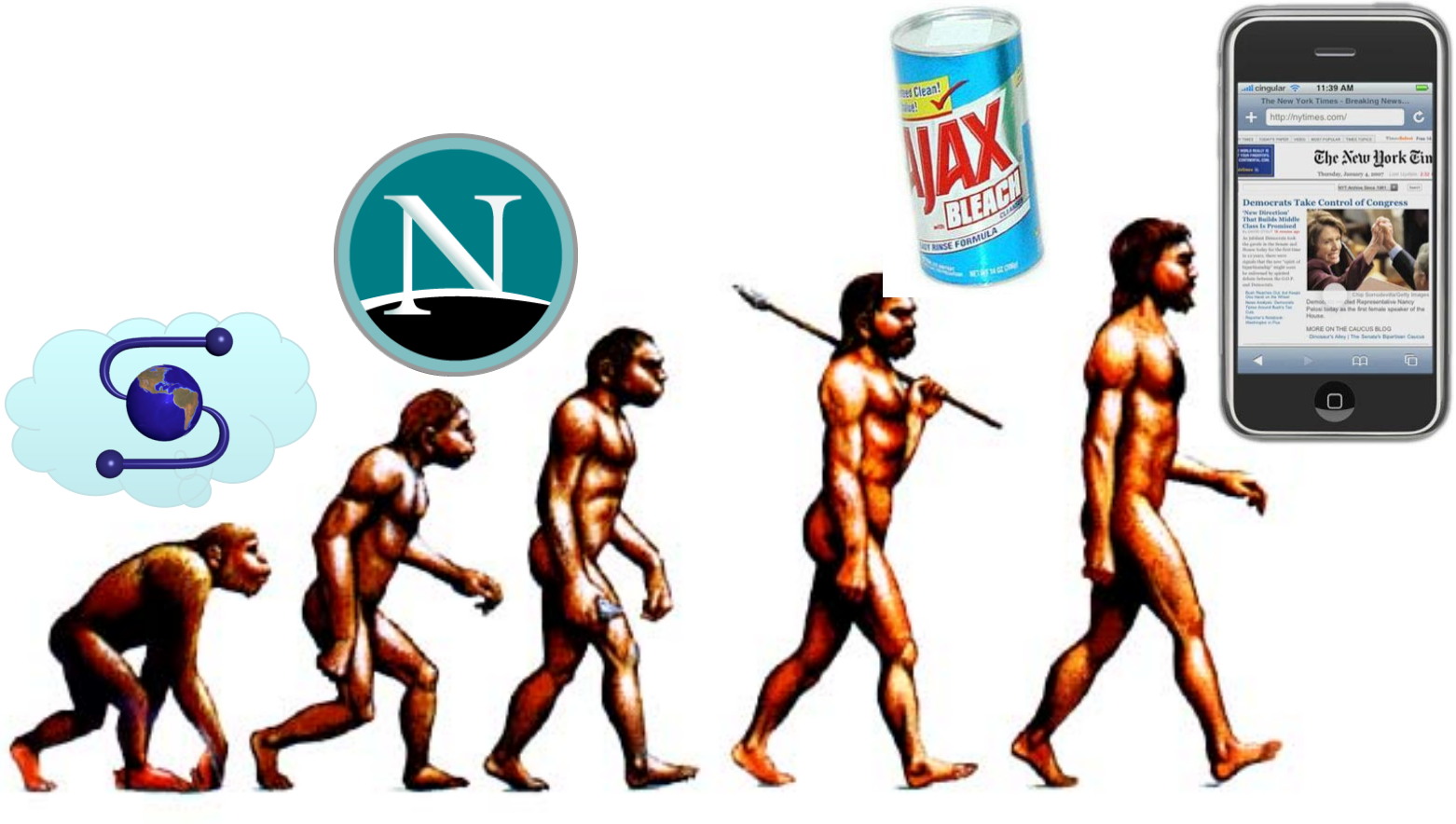
JavaScript & AJAX



JavaScript had to "look like Java" only less so—be Java's dumb kid brother or boy-hostage sidekick. Plus, I had to be done in ten days or something worse than JavaScript would have happened.

JavaScript: the Big Picture *(ELLS §11.1)*

Image: Wikimedia. Used under CC-SA license.

The Moving Parts

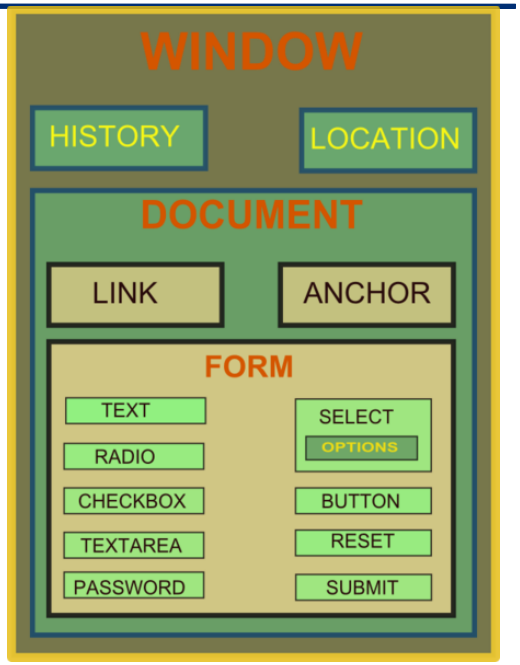
- 1995: Netscape includes ~~LiveScript~~ JavaScript as browser scripting language
  - Originally, for simple client-side code such as animations and form input validation
  - Document Object Model (DOM) lets JavaScript inspect & modify document elements
- 1997: standardized as ECMAScript
- 1998: Microsoft adds XmlHttpRequest to IE5
- 2005: Google Maps, AJAX takes off

JavaScript's privileged position

- Because it's embedded in browser, JavaScript code can:

1. be triggered by user-initiated *events* (mouse down, mouse hover, keypress, …)
2. make HTTP requests to server *without* triggering page reload
3. be triggered by network events (e.g. server responds to HTTP request)
4. examine & *modify* (causing redisplay) current document

DOM & JavaScript: Document = tree of objects



- DOM is a *language-independent, hierarchical* representation of HTML or XML document
- Browser *parses* HTML or XML => DOM
- JavaScript API (JSAPI) makes DOM data structures accessible from JS code
- Inspect DOM element values/attributes
- Change values/attributes → redisplay
- Implemented incompatibly across browsers

…but jQuery framework will help us

- *JavaScript is just another language; the JSAPI gives it power to script the browser.*
- *JavaScript is a single-threaded language. Therefore all JS interactions with browser must be nonblocking.*

Unobtrusive JavaScript

- Similar to motivation for separating out CSS
- Before: `<b><font size="+1">Big!</font></b>`
- After: `<span class="bigbold">Big!</span>`
- How: New HTML5 attributes `data-*` are specifically to be *ignored* by browser
- Gist: Replace “hardcoded” handlers with references to data-attributes

http://pastebin.com/y8Hj83b4U

Graceful degradation

- Browsers with JS disabled ("legacy browsers") should get a usable experience
  - should be easy if your app is RESTful already
- *Conditionally* show page elements that *require* JS
  - Add elements in a "setup" function in JS
  - Or, make elements hidden using CSS, then change CSS style in "setup" function

Client-side JavaScript code can interact with HTML page elements because this functionality:
(a) is part of the JavaScript language
(b) is part of the browser
(c) is provided by the JSAPI

- (a), (b) and (c)
- (a) & (b) only
- (b) & (c) only
- (a) only

JavaScript for Ruby Programmers *(ELLS §11.2)*

JavaScript basics

- Basic object type is a hash; keys sometimes called *slots* or *properties*

```
var movie={title: 'Up',
  releaseInfo: {rating: 'G', year: 2010}}
```

- Functions are *1st class objects* and *closures*
- passing functions is extremely common!
- var restricts scope of a variable; if absent or used in outermost scope → global

http://pastebin.com/srnEVH59

- The *global object* defines some constants that are part of JS's environment
- this in global scope refers to global object

More on functions

- Functions are *first class objects* & *closures*
- A function is a lambda expression

```
var make_times = function(mul) {
  return function(arg) { return arg * mul; }
}
// or:  function make_times(mul) { ... }

times2 = make_times(2)
times3 = make_times(3)
times2(5) ➔ 10
times3.call(null, 5) ➔ 15
```

JS, browsers, and scope

- Each loaded HTML page gets its own JS global object
  - must separately load any desired JS:=

javascript_include_tag 'application'→<script src="/public/javascripts/application.js">
</script>

- Unobtrusive technique: create a *single global object* to hold all your app's code
  - some object slots are like static/class variables
  - others are instance or class functions

Which is NOT true about functions in JavaScript?

- They can be anonymous
- They always return a value, even without an explicit return() statement
- They can be passed a function as an argument
- They can execute concurrently with other functions

Functions and Prototype Inheritance *(ELLS §11.3)*

Every object has a prototype

- No true Classes; object inheritance based on "prototypes"
- each newly-created object has a prototype
- obj.__proto__ (except in some versions of IE)
- when slot lookup fails in an object, its prototype is consulted, and so on up the chain
- so object "inherits" both value-slots and function-slots
- Q: how to create new objects from your own prototype?

http://pastebin.com/QqsqgGFp

Summary

- Call a function using new ➔ creates & returns an object (this) whose prototype is whatever the function's prototype is
  - so obj.constructor.prototype always works
- Call a function on that object ➔ object becomes the function's this
- *Call a function without a receiver* ➔ *assumed to be Global Object* window
- *Call a "constructor-like" function without* new *=> undefined*

```
var Square = function(side) {
 this.side = side;
 this.area = function() {
   return this.side*this.side;
 }
};
```

Which call will evaluate to 9?

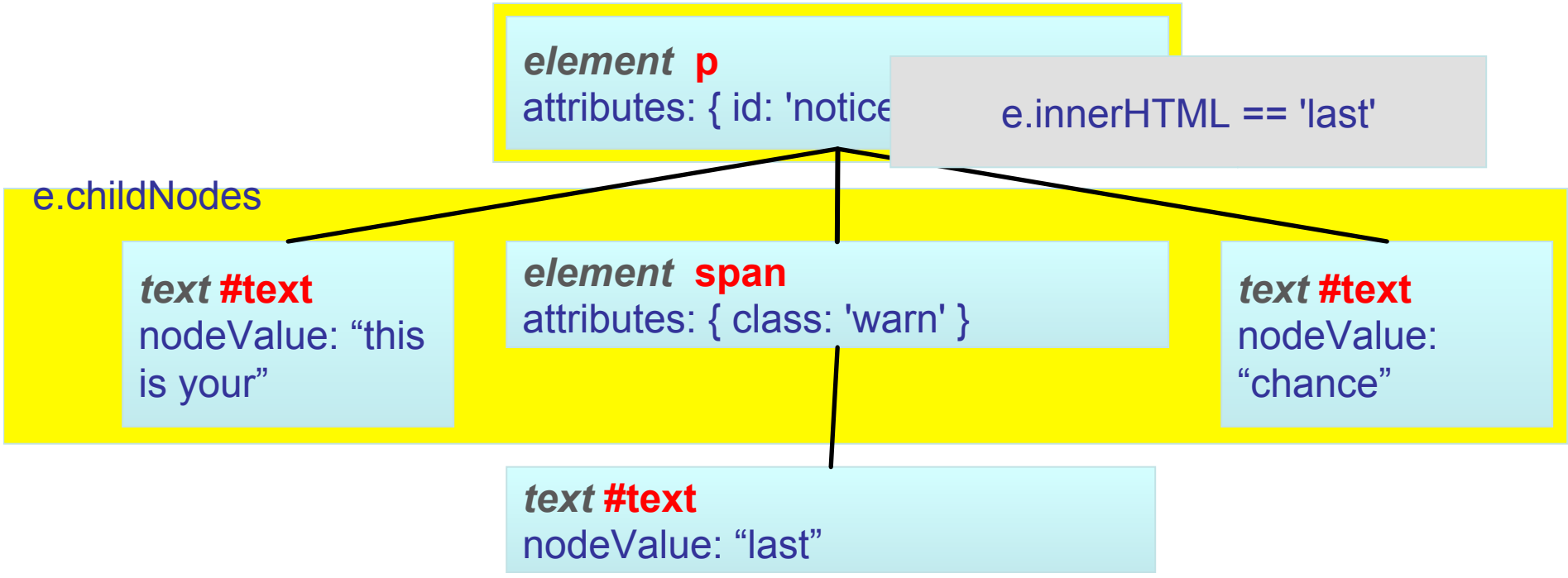
- Square(3).area
- Square(3).area()
- (new Square(3)).area
- var p = Square ; (new p(3)).area()

The Document Object Model (DOM), jQuery Intro, Events, and Callbacks *(ELLS §11.4-11.5)*

DOM Example

```
<p id="notice">
  This is your <span c
</p>
e = document.getElementById('notice)';
```

e.innerHTML == 'This is your last chance!'



jQuery

- A powerful framework for DOM manipulation
- adds many useful features over browsers' built-in JSAPI
- homogenizes incompatible JSAPI's across browsers
- Defines a single polymorphic global function jQuery(), aliased as $()

http://pastebin.com/jMMRGmN4

Selecting DOM elements

- $() or jQuery() with any CSS3 expression

$('#movies'), $('.heading'), $('table')

- Warning! different from document.getElementById()
- Warning! may return multiple elements, but it's not a JavaScript array!
- but there is an iterator for it

What can you do with selected element(s)?

e.find()

e.show(), e.hide(), e.fadeOut('slow')

e.addClass(), e.removeClass()

e.css({'background-color': 'green', color: 'white'})

e.html(), e.text()

e.attr('src')

Handlers on Buttons & Links

- A good summary of recognized handlers:
  http://www.chipchapin.com/WebTools/JavaScript/example1-04.html
- What about links & buttons that are clickable *without* JavaScript?
  - handler runs first
  - if handler returns *false*, no other action taken
  - otherwise, other actions follow handler
  - example use: client-side form validation
- Handler code can be inline or functions

Summary

- Select elements with $() (or wrap to give them secret jQuery powers)
- Inspect them…

  text() or html()

  is(:checked), is(:selected), etc.

  attr('src')
- Add/remove CSS classes, hide/show
- Create setup function that *binds* handler(s) on element(s)
- common ones: onclick, onsubmit, onchange
- Pass func to $() (alias for document.ready())