The Future of JavaScript

I mean ECMAScript

Douglas Crockford

Yahoo!

Welcome to the Future!

Such as it is.

The World's Most Popular Programming Language

The World's Most Popular Programming Language

The World's Most Unpopular Programming Language

ECMAScript is the language that people use without bothering to learn it first.

Programming is complicated business. It should never be undertaken in ignorance.

Functions are first class.

Static Scoping. (Mostly.)

Writing in ECMAScript language without understanding closure is like writing in Java without understanding classes.

Global

```
var names = ['zero', 'one', 'two',
        'three', 'four', 'five', 'six',
        'seven', 'eight', 'nine'];

var digit_name = function (n) {
    return names[n];
};



alert(digit_name(3));    // 'three'
```

Slow

```
var digit_name = function (n) {
    var names = ['zero', 'one', 'two',
        'three', 'four', 'five', 'six',
        'seven', 'eight', 'nine'];


    return names[n];
};



alert(digit_name(3));    // 'three'
```

Closure

```
var digit_name = (function () {
    var names = ['zero', 'one', 'two',
        'three', 'four', 'five', 'six',
        'seven', 'eight', 'nine'];

    return function (n) {
        return names[n];
    };

}());

alert(digit_name(3));    // 'three'
```

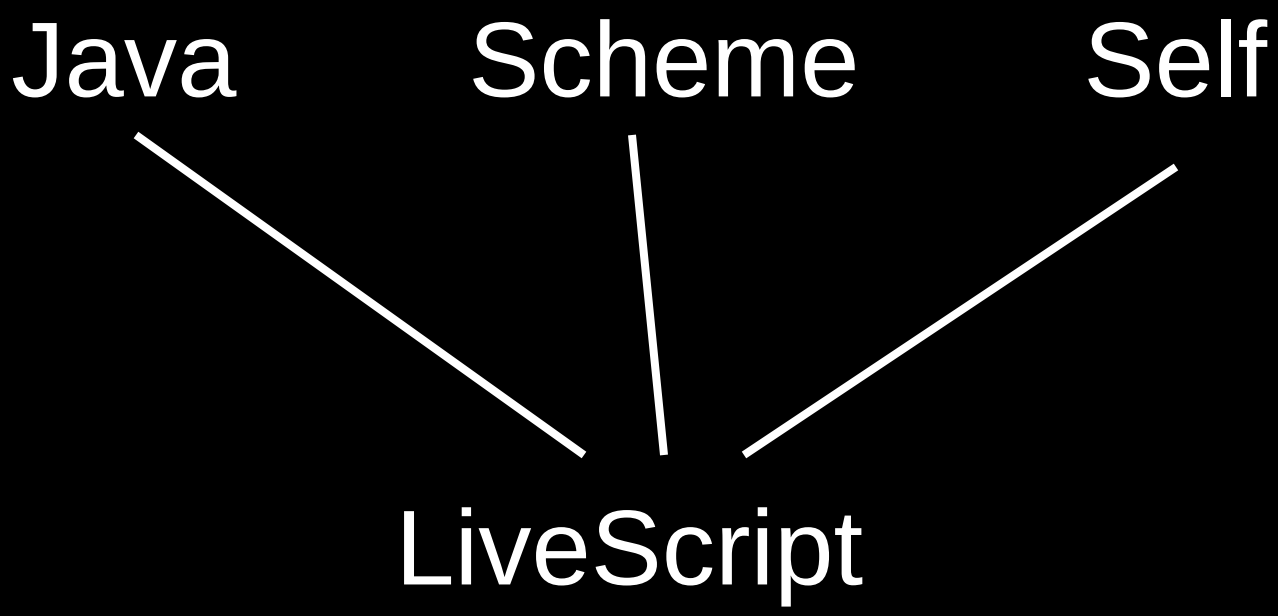
Soft Objects

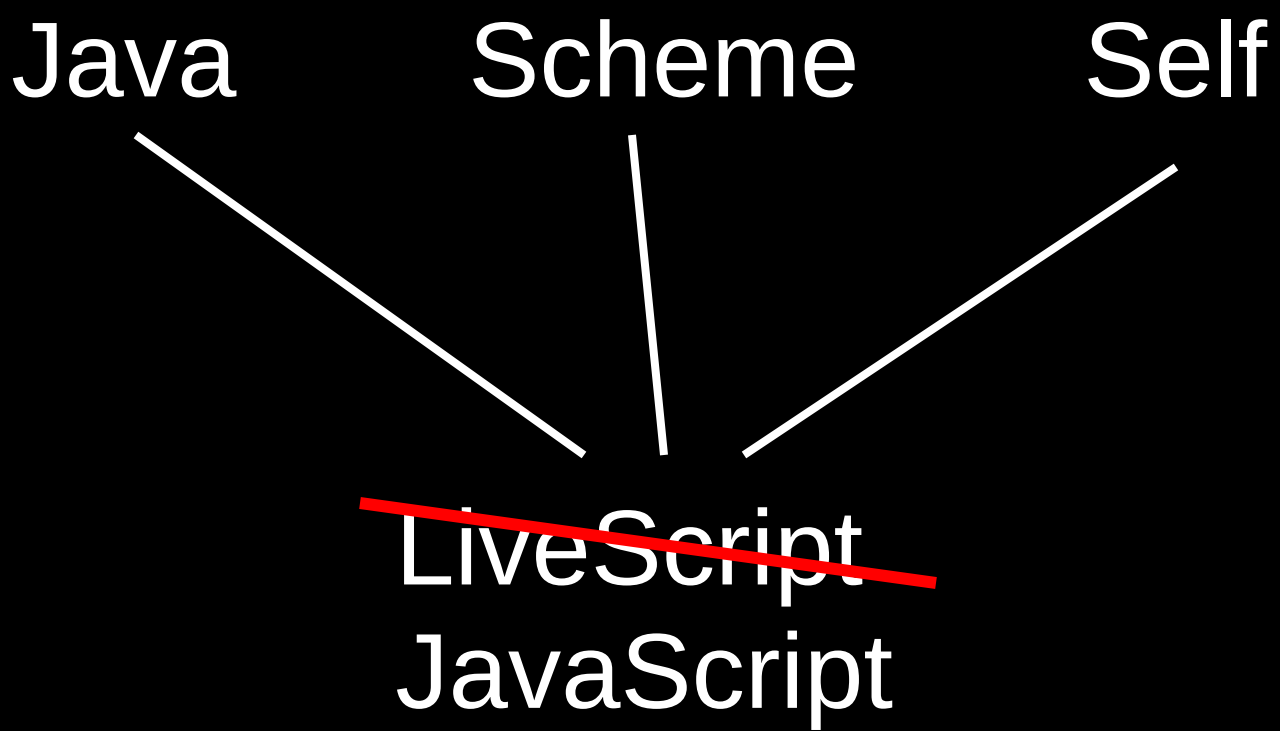
- An object is simply a dynamic container of name/value pairs.
- New pairs (or properties) may be added at any time.
- The value of a property may be a function.
- The pseudo-parameter **`this`** is bound at invocation time.
- An object can inherit from another object.

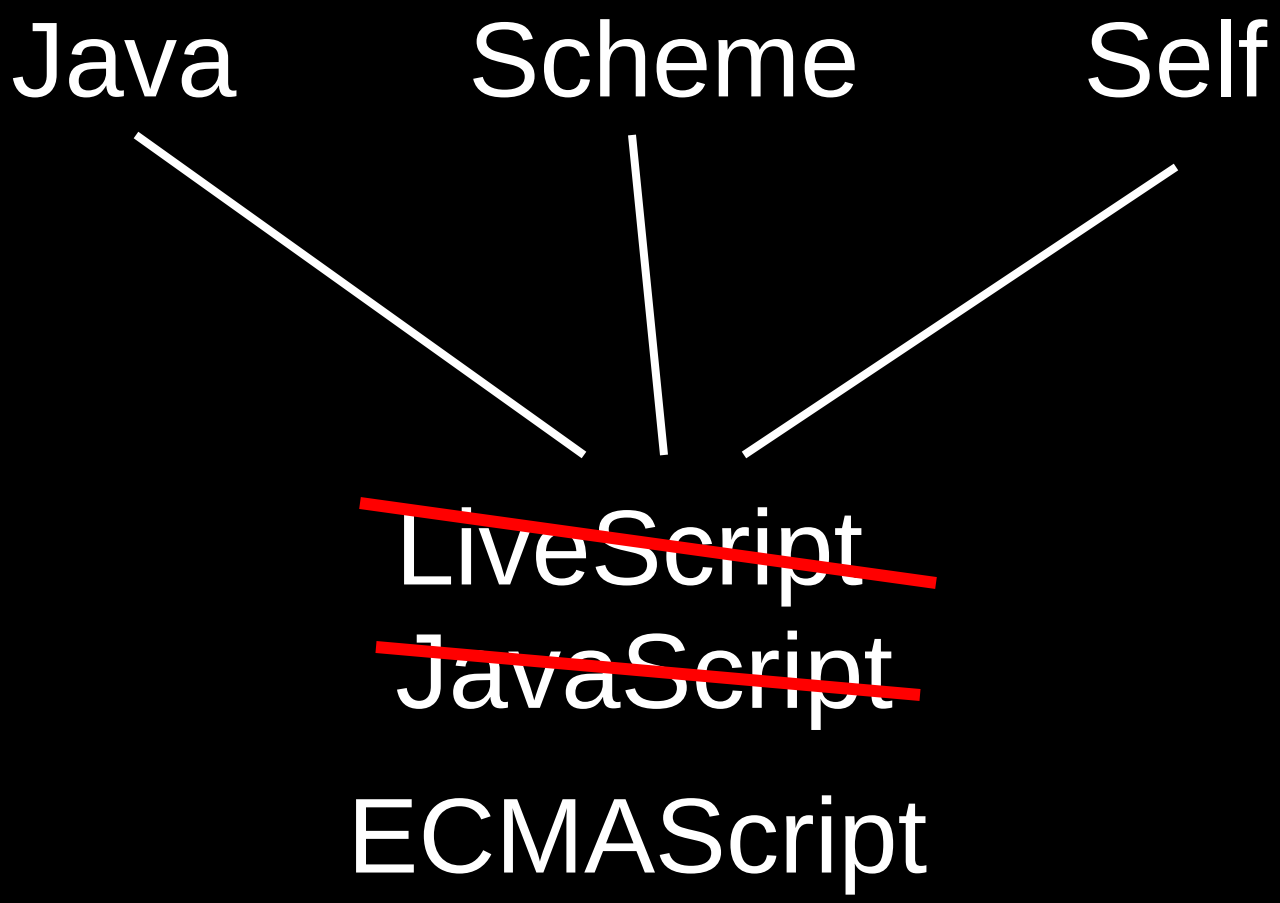
Object Literals

- Like JSON objects, but more powerful.
- Values can be obtained from expressions.
- Values can be functions.


YUI
YUICONF 2009
OCTOBER 28-29


Java
Scheme
Self
LiveScript


Java
Scheme
Self
LiveScript
JavaScript


Java
Scheme
Self
LiveScript
JavaScript
ECMAScript

ECMAScript

- 1999 Third Edition ES3
- 2009 Fifth Edition ES5
  - Default Strict
- Avoid ES5/Default.
- For the short term, work in the intersection of ES3 and ES5/Strict
- For the long term, work with ES5/Strict.

Harmony

- The code name of the next proposal is Harmony, not ES6.
- We want to avoid giving proposals edition numbers because it gives the false appearance of inevitability or momentum.
- Harmony will be built on the Strict Language.
- Harmony will probably have incompatible syntax, so programs written in the Harmony language will fail on all pre-Harmony browsers.
- Hopefully the IE6 problem will be gone by the time our work is done.

JavaScript is the virtual machine of the Internet.

Server Side JavaScript

- This is not a new idea.
- Netscape offered an SSJS product in 1996.

- It was a page template system using a **`<server>`** tag and a **`write`** function to insert matter in the output stream.

- It had all of the disadvantages of the other page template systems with a really slow JS engine.

Threading

Pro

- No rethinking is necessary.
- Blocking programs are ok.
- Execution continues as long as any thread is not blocked.

Con

- Stack memory per thread.
- If two threads use the same memory, a race *may* occur.
- Overhead.
- Deadlock.
- Thinking about reliability is extremely difficult.
- System/Application confusion.

Fortunately, there is a model that completely avoids all of the reliability hazards of threads.

The Event Loop!

Browser Event Loop

- Event queue containing callback functions. (timer, ui, network)
- Turn: Remove one callback from the queue. It runs to completion.
- Prime Directive: Never block. Never wait. Finish fast.
- The Event Loop is one of the best parts of the browser.
- Avoid: **`alert`**, **`confirm`** , **`prompt`**, XHR synchronous.

JavaScript does not have **`READ`**.

That has always been seen as a huge disadvantage, but it is actually a wonderful thing.

Event Loop

Pro

- Free of races and deadlocks.
- Only one stack.
- Very low overhead.
- Resilient. If a turn fails, the program can still go on.

Con

- Programs must never block.
- Programs are inside out! Waa!
- Turns must finish quickly.

Long running tasks

- Two solutions for long running programs:

1. Eteration: Break the task into multiple turns.

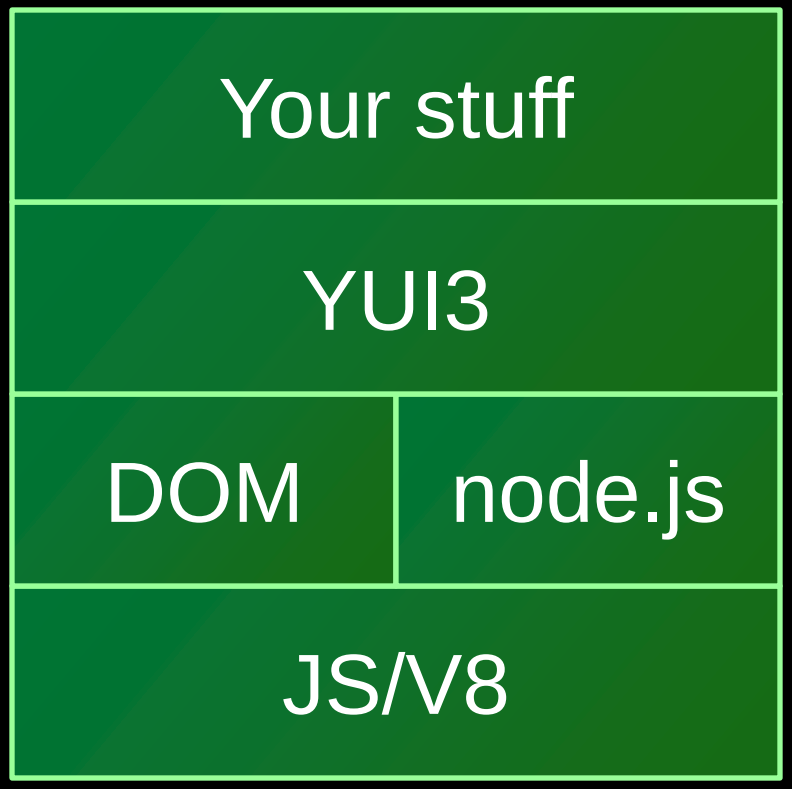
2. Move the task into a separate process (workers).

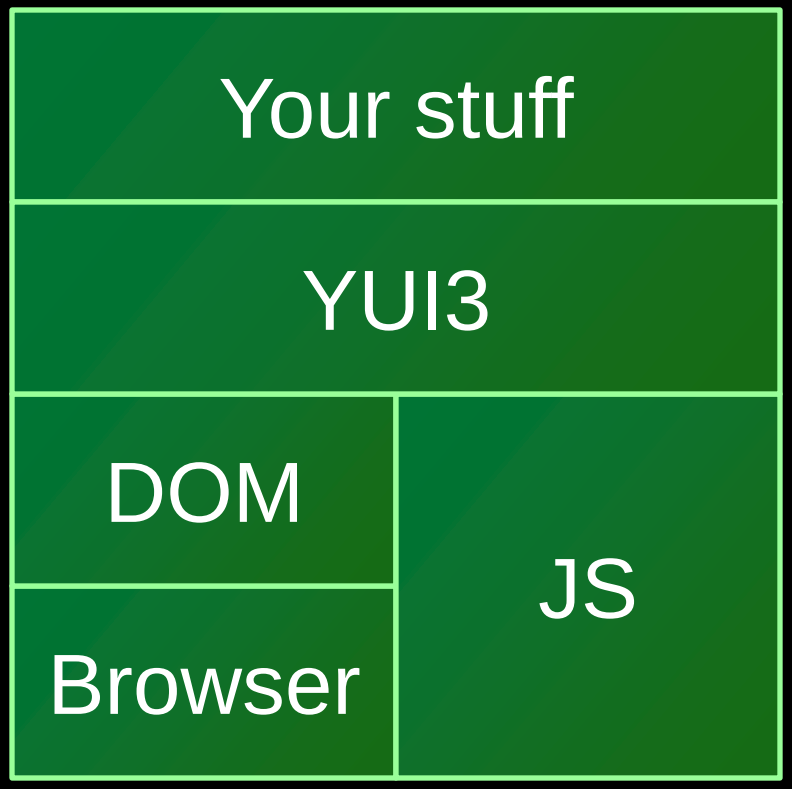
What about Server Side JavaScript with an Event Loop?

node.js

- node.js implements a web server in a JavaScript event loop.
- It is a high-performance event pump.
- `fs.read(`fd, length, position, encoding, `function (`err, str, bytesRead`) {...})`
- Everything is non-blocking.
- Except:
  - some synchronous functions
  - `require`

Your stuff runs on both sides



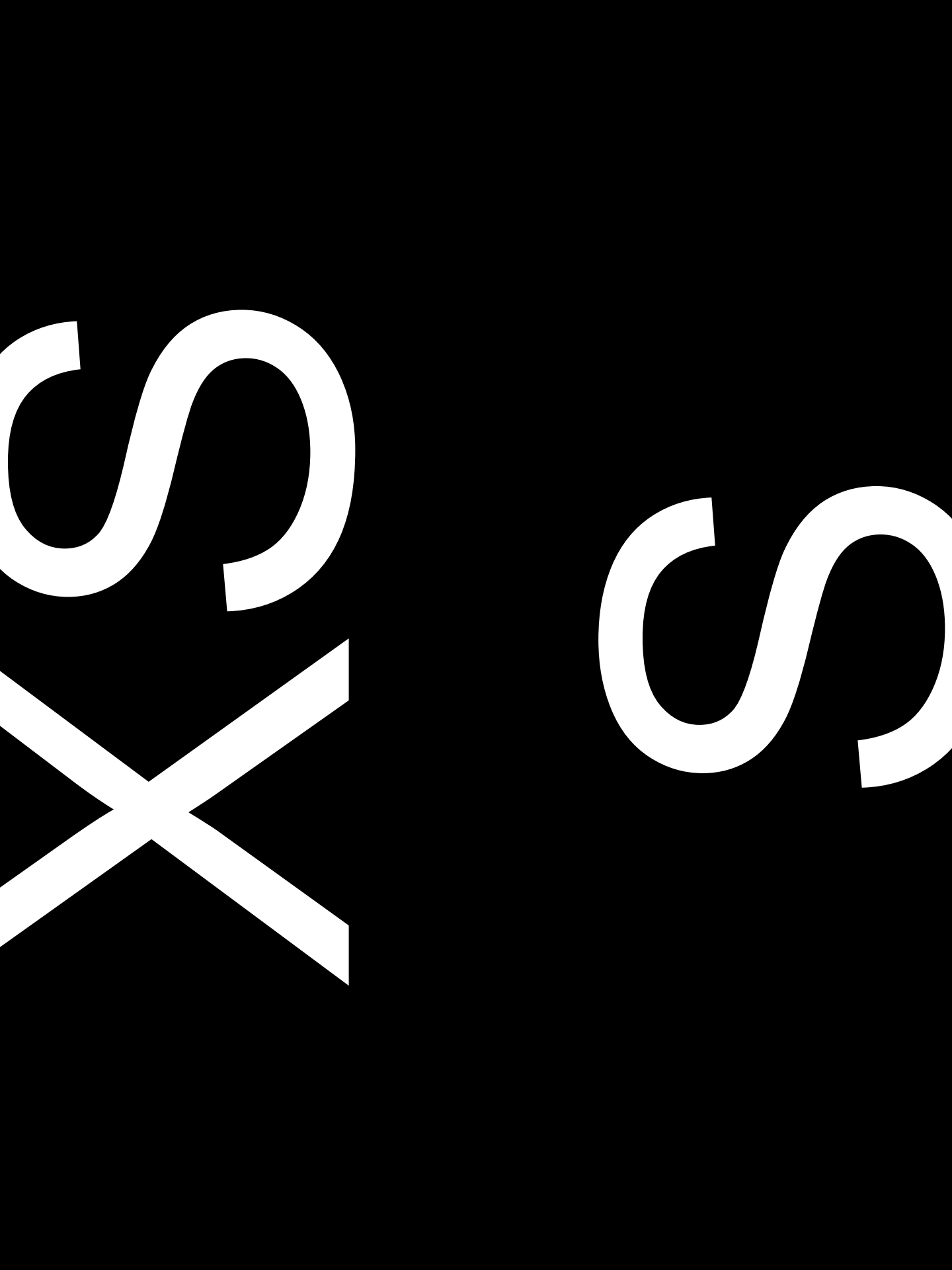


Requestor

```
myRequestor = function (sync) {
    service_request(arguments,
        function (result) {
            sync(result, error);
        });
};

par([requestors…], sync, timeout);
seq([requestors…], sync, timeout);
```

Security



Cross site scripting attacks were invented in 1995.

We made no progress on the fundamental problem.

XSS has two causes:

1. Sharing of the global object. The Principle of Excessive Authority.

2. Misinterpretation of HTML.

What can an attacker do if he gets some script into your page?

An attacker can request additional scripts from any server in the world.

Once it gets a foothold, it can obtain all of the scripts it needs.

An attacker can read the document.

The attacker can see everything the user sees.

An attacker can make requests of your server.

Your server cannot detect that the request did not originate with your application.

If your server accepts SQL queries, then the attacker gets access to your database.

SQL was optimized for
SQL Injection Attacks

An attacker has control over the display and can request information from the user.

The user cannot detect that the request did not originate with your application.

An attacker can send information to servers anywhere in the world.

The consequences of a successful attack are horrible.

Harm to customers.
Loss of trust.
Legal liabilities.

The browser does not prevent any of these terrible things.

Web standards require these weaknesses.

15 Years of XSS

HTML5

A big step in the wrong direction.

Tragically, HTML5 ignores and worsens the XSS problem.

"…HTML doesn't ever have markup injection vulnerabilities…"

http://lists.w3.org/Archives/Public/public-webapps/2010AprJun/0648.html

My Recommendation

- Suspend the HTML5 standards process.
- Repair the XSS hazard.
- Review the HTML5 proposals with respect to the new security discipline.

And then there is the Mashup Problem

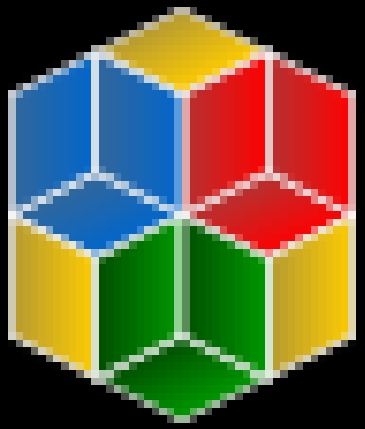
- A mashup is the combining of programs representing multiple interests.
- The browser confuses those interests.
- A mashup is a self-inflicted XSS attack.
- So an advertiser on a page gets the same privileges as an Ajax library or an analytics file, which is the same as the main applications, which is the same as any XSS code that falls into the page.
- Advertising is a self-inflicted XSS attack.

Safe JavaScript Subsets

Deny access to the global object and the DOM.

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

ADsafe. http://www.ADsafe.org/



ECMAScript Fifth Edition Strict

December 2009

ES5/Strict makes it possible to have static verification of third party code without over-constraining the programming model.

The best of both Caja and ADsafe.

The IE6 Problem

IE6 MUST DIE!

IE7 MUST DIE!

IE8 MUST DIE!

IE9

?

Thank you and good night.