**The Problem: JavaScript Concurrency**

There are a number of bottlenecks preventing interesting applications from being ported (say, from server-heavy implementations) to client-side JavaScript. Some of these include browser compatibility, static typing, accessibility, and performance. Fortunately, the latter is quickly becoming a thing of the past as browser vendors rapidly improve the speed of their JavaScript engines.

One thing that's remained a hindrance for JavaScript is actually the language itself. JavaScript is a single-threaded environment, meaning multiple scripts cannot run at the same time. As an example, imagine a site that needs to handle UI events, query and process large amounts of API data, and manipulate the DOM. Pretty common, right? Unfortunately all of that can't be simultaneous due to limitations in browsers' JavaScript runtime. Script execution happens within a single thread.

Developers mimic 'concurrency' by using techniques like setTimeout(),setInterval(), XMLHttpRequest, and event handlers. Yes, all of these features run asynchronously, but non-blocking doesn't necessarily mean concurrency. Asynchronous events are processed after the current executing script has yielded. The good news is that HTML5 gives us something better than these hacks!

**Introducing Web Workers: Bring Threading to JavaScript**

The [Web Workers](http://www.whatwg.org/specs/web-workers/current-work/) specification defines an API for spawning background scripts in your web application. Web Workers allow you to do things like fire up long-running scripts to handle computationally intensive tasks, but without blocking the UI or other scripts to handle user interactions