JavaScript Concurrency

JavaScript supports *closures.*

Closures form the basis for *concurrency*[1].

In a *concurrent* design, all routines are *closures*, except one distinguished routine – the *dispatcher* – that invokes closures "at random".

Closures communicate with each other solely through message queues. Yes, I am describing a message passing architecture. We have the prejudice that message passing fails This prejudice is based on the idea that message-passing is used to build a flat system. The way to tame message passing is the same way that GOTOs were tamed – through the use of structured encapsulation[2] and hierarchy.

Flat-*anything* is bad. Flat-*anything* results in spaghetti. Global variables were bad because they were "flat". Local variable scoping solved that problem. A flat type-system is bad (have we recognized this fact yet?). A flat function space is bad (we've been playing whack-a-mole with packaging, packaging systems, imports, exports, etc., trying to fix this problem). Flat data is bad. We fixed this problem with OO.

That's it – to implement concurrency in JS, you need closures, message queues and a dispatcher. A good dollop of hierarchical organization will help.

(See my essay on *isolation.* See my essay on simple systems).

---

1 Note that *concurrency* and *parallelism* are not the same things. See Rob Pike's talk "Concurrency is not Parallelism". Concurrency is a programming paradigm. Parallelism is a solution that requires the use of the concurrent paradigm. Parallelism requires concurrency, but not the reverse – concurrency does not imply parallelism.

2 Nesting, scoping, etc.