dnode

remote execution in the kingdom of node.js

James Halliday (substack)

March 22, 2011

King Ryan

Ryan decrees

King Ryan

Ryan decrees

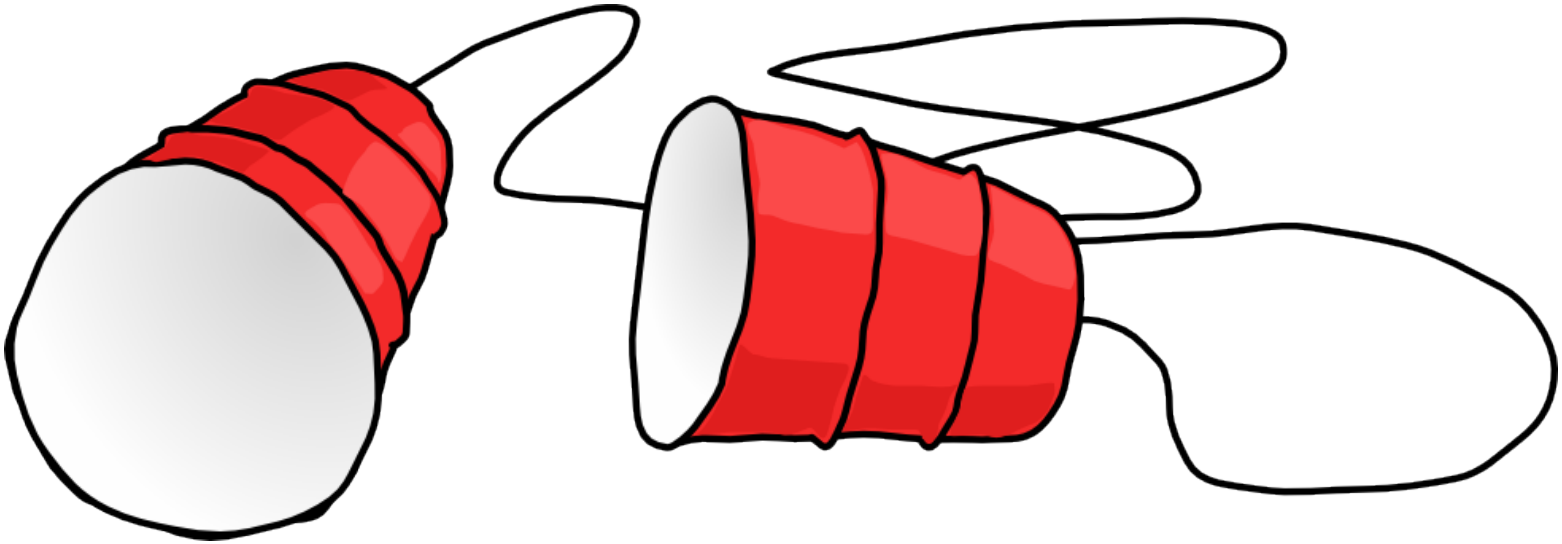
- Thou shalt have only one thread.

King Ryan

Ryan decrees

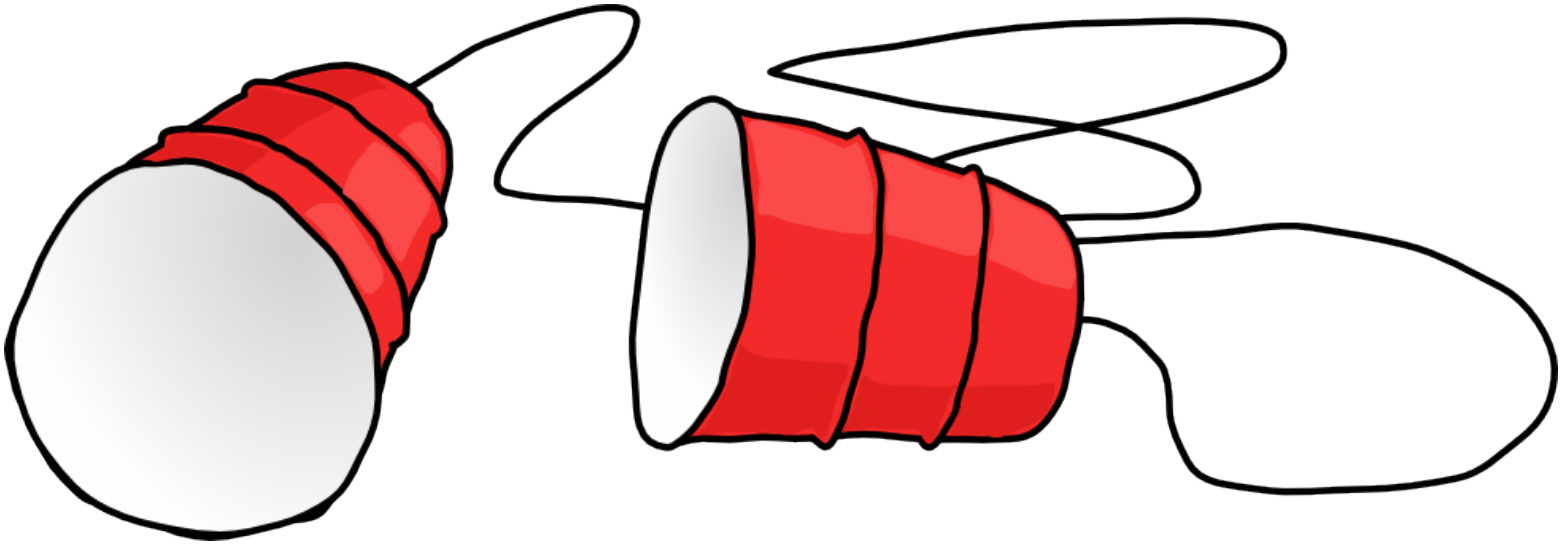
- Thou shalt have only one thread.
- Thy IO shalt be asynchronous.

call remote functions!



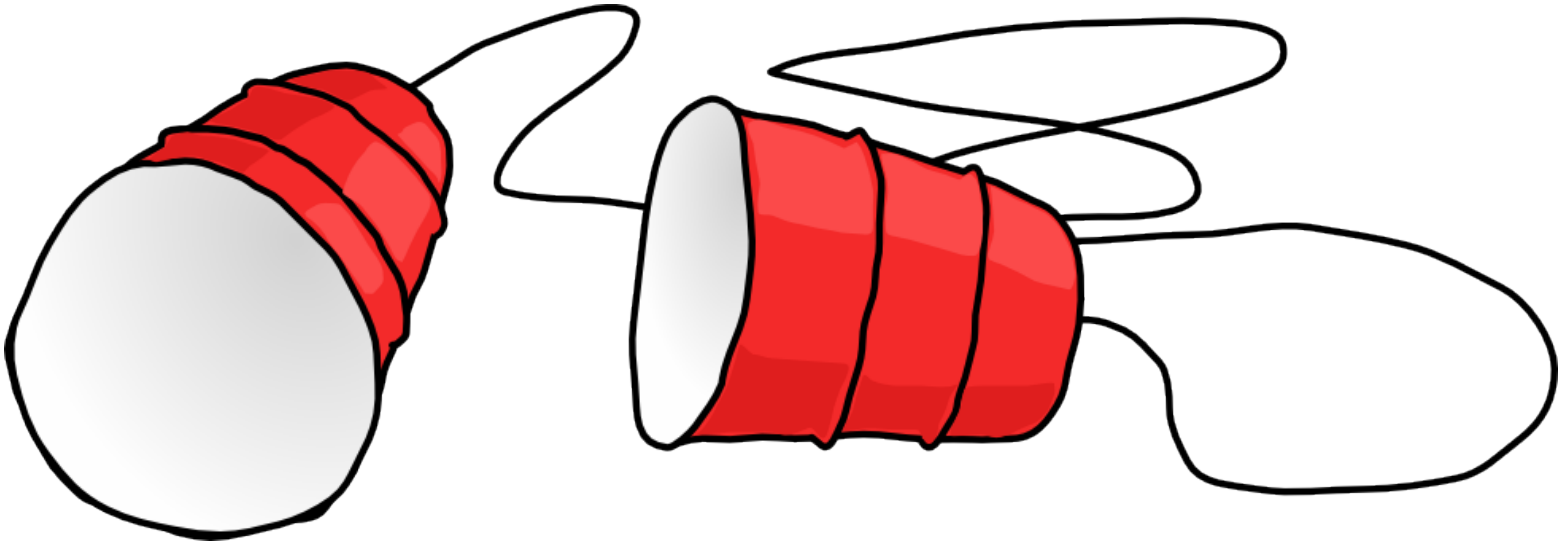
If only there were an easier way to expose backend code

call remote functions!



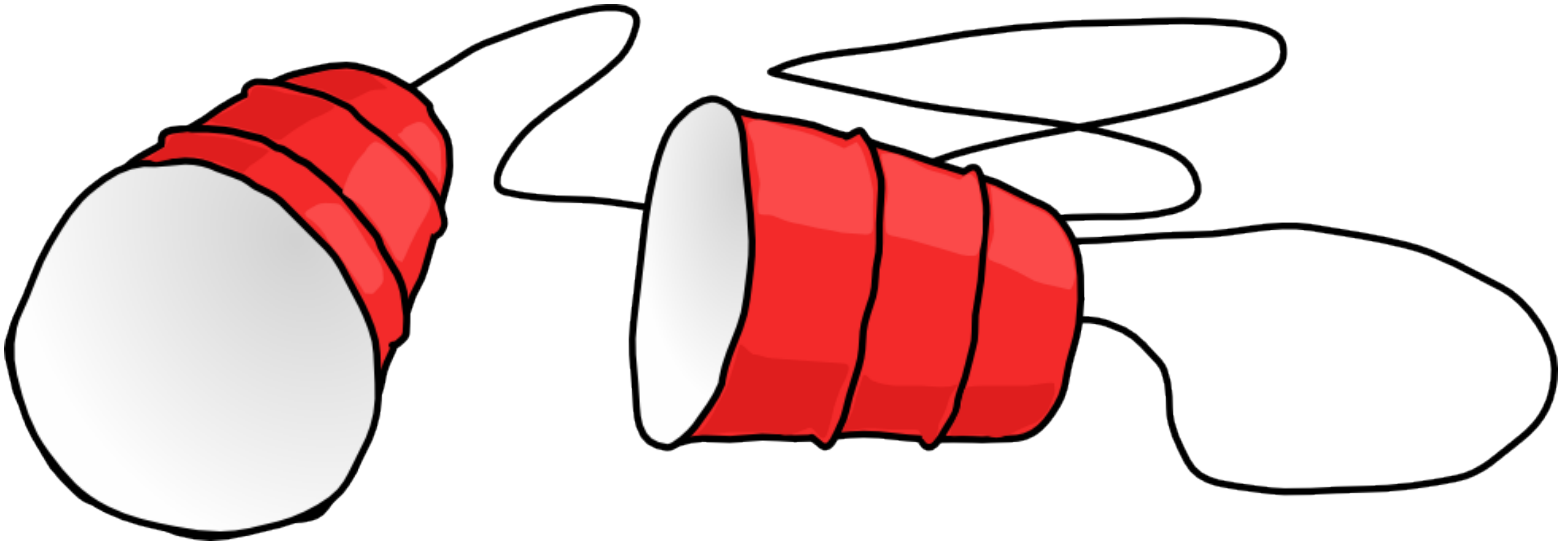
If only there were an easier way to expose backend code that complied with the King's wishes.

call remote functions!



If only there were an easier way to expose backend code that complied with the King's wishes.
Well there are lots of these actually.

call remote functions!



If only there were an easier way to expose backend code that complied with the King's wishes.
Well there are lots of these actually. Most of them suck.

At a glance

dnode gets out of your way

At a glance

dnode gets out of your way

- fundamentally asynchronous
  - no return values

At a glance

dnode gets out of your way

- fundamentally asynchronous
  - no return values
- symmetric
  - both sides can call the other's methods

At a glance

dnode gets out of your way

- fundamentally asynchronous
  - no return values
- symmetric
  - both sides can call the other's methods
- abstract
  - use it on the browser or the server

At a glance

dnode gets out of your way

- fundamentally asynchronous
  - no return values
- symmetric
  - both sides can call the other's methods
- abstract
  - use it on the browser or the server
- simple yet powerful semantics
  - generalized callback wrapping
  - no implicit return callbacks

Zing example server!

```
var dnode = require('dnode');

var server = dnode({
    zing : function (n, cb) { cb(n * 100) }
});
server.listen(5000);
```

Zing example client!

```
var dnode = require('dnode');

dnode.connect(5000, function (remote) {
    remote.zing(66, function (n) {
        console.log('n = ' + n);
    });
});
```

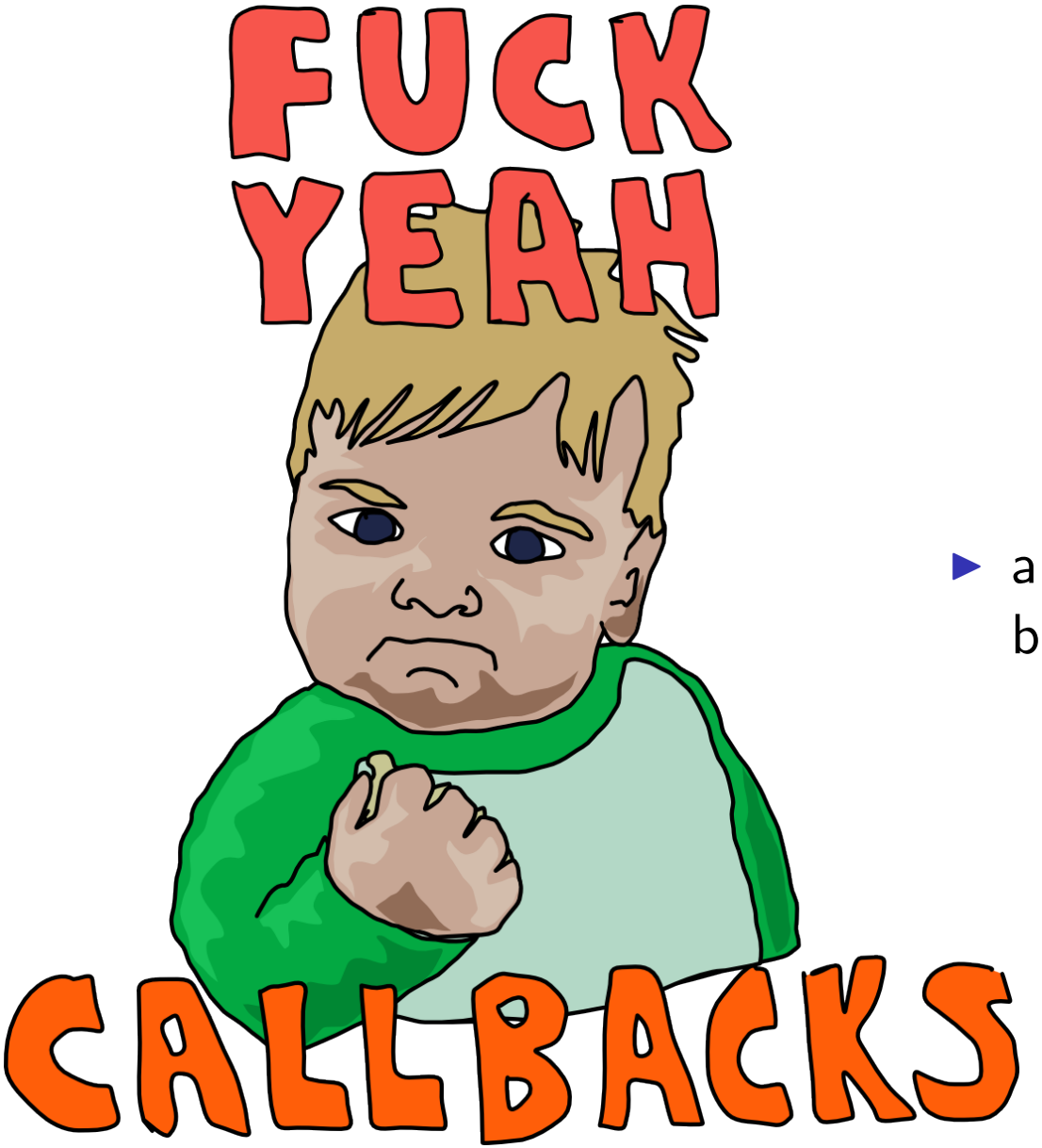
Zing example client!

```
var dnode = require('dnode');

dnode.connect(5000, function (remote) {
    remote.zing(66, function (n) {
        console.log('n = ' + n);
    });
});
```

```
code $ node zing_server.js &
[1] 21671
code $ node zing_client.js
n = 6600
^C
```

Fuck yeah callbacks

Fuck yeah callbacks



- and now let's do that in the browser

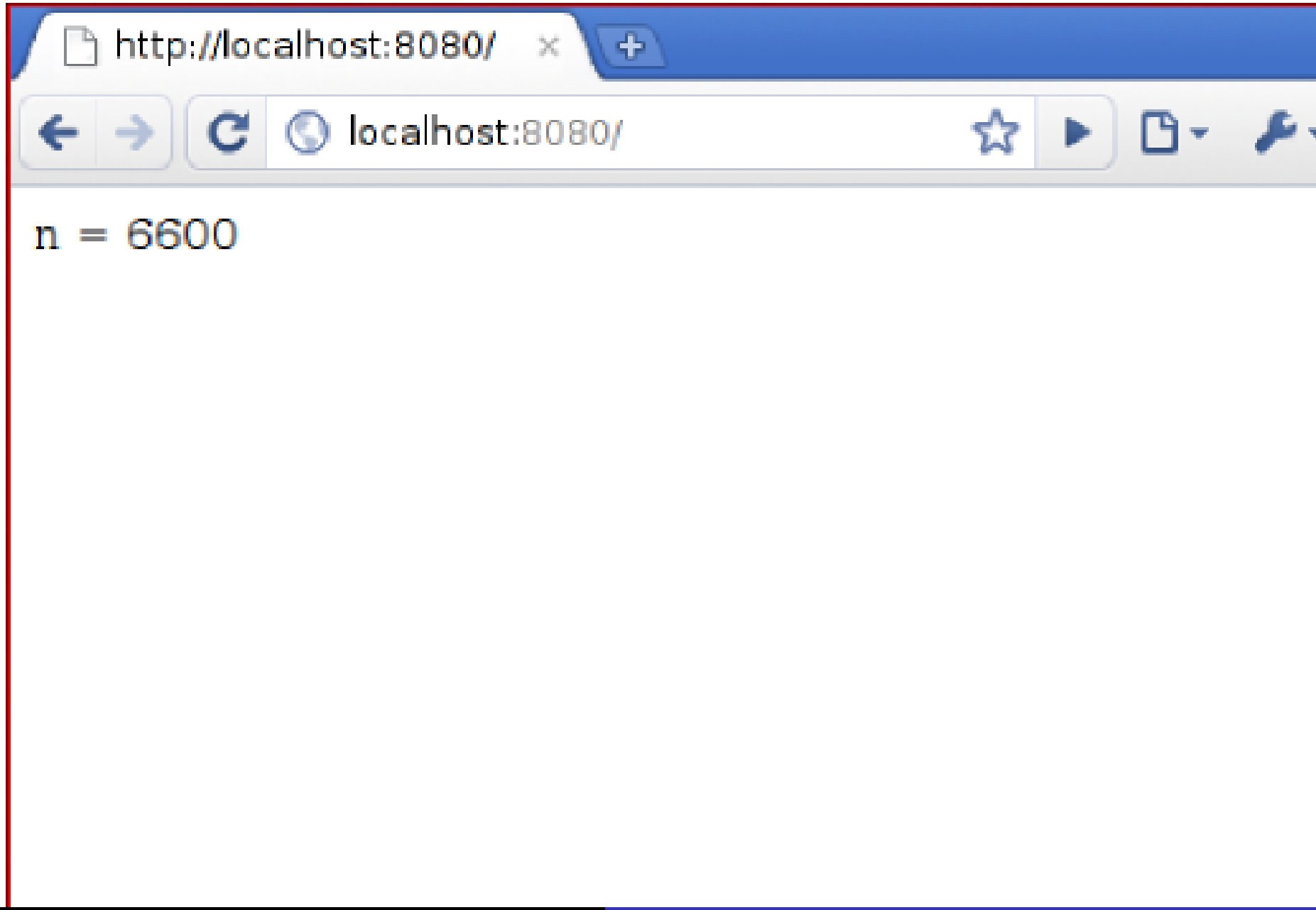
Retrofitting the server

```
var express = require('express');
var app = express.createServer();
app.use(express.static(__dirname));
app.listen(8080);

var dnode = require('dnode');

var server = dnode({
    zing : function (n, cb) { cb(n * 100) }
});
server.listen(app);
```

Browser code

```
<html>
<head>
  <script type="text/javascript" src="/dnode.js">
  </script>
  <script type="text/javascript">
    DNode.connect(function (remote) {
      remote.zing(66, function (n) {
        document.getElementById("res").innerHTML
          = 'n = ' + n;
      });
    });
  </script>
</head>
<body>
  <div id="res"></div>
</body>
</html>
```

It works in the browser! Huzzah!



How it doesn't work

dnode DOESN'T use these

- eval()
- Function.prototype.toString()
- <script> tag injection

How it does work

When a function gets called...

- a recursive walk pulls out all the functions from the arguments
- function paths and IDs sent alongside
- handles cycles too

How it does work

When a function gets called...

- a recursive walk pulls out all the functions from the arguments
- function paths and IDs sent alongside
- handles cycles too

```
{
    "method" : "zing",
    "arguments" : [ 100, "[Function]" ],
    "callbacks" : { "0" : ["1"] }
}
```

How it does work

When a function gets called...

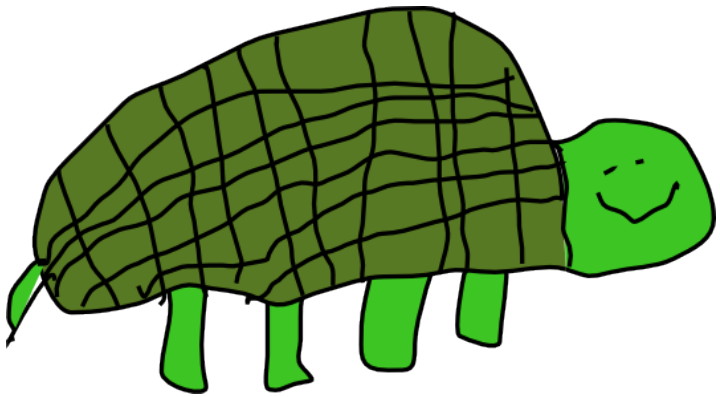
- a recursive walk pulls out all the functions from the arguments
- function paths and IDs sent alongside
- handles cycles too

```
{
    "method" : "zing",
    "arguments" : [ 100, "[Function]" ],
    "callbacks" : { "0" : ["1"] }
}
```

This transformation is recursive!

It's callbacks all the way down!

```
remote.turtles(function (f) {
    f(function (g) {
        g(function (h) {
            h("It's callbacks all the way down");
        });
    });
});
```



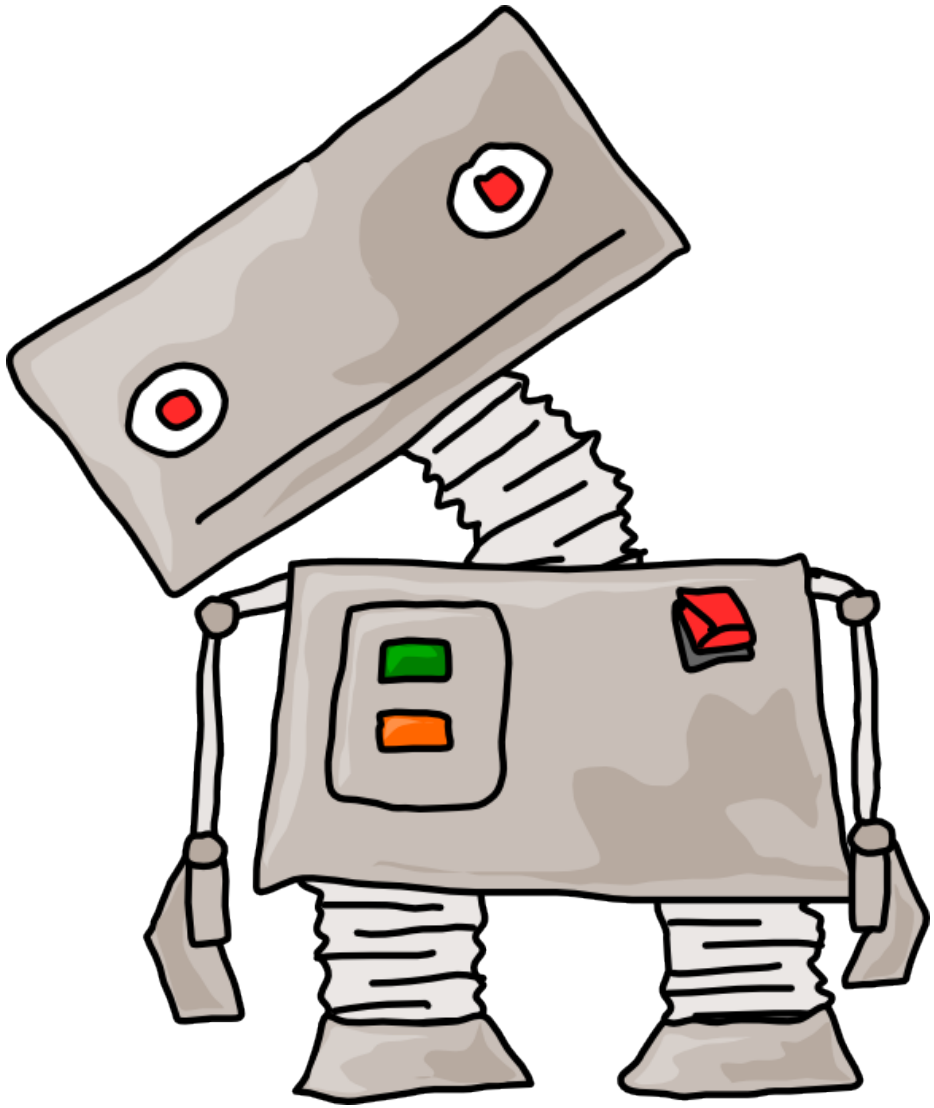
OOP javascript style

```
remote.meta(function (cb) {
    cb({
        foo : function (x, f) {
            f(2 * x + 1);
        },
        bar : function (x, f) {
            f(3 * x - 2);
        },
        baz : 5,
    });
});
```

Good news for you!

No need to write your own

- method dispatcher
- state machine
- serialization protocol



Git it!

github.com/substack/dnode

github.com/substack/dnode-slides

substack.net