What is node?

Taking JS Beyond the Browser

Node.js is a framework for building scalable server-side applications and network oriented programs with asynchronous javascript (js).

JS in one slide !!!


```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to trigger a function that will output "Hello World" in a p element with id="demo".</p>

<button onclick="myFunction()">Click me</button>

<p id="demo"></p>

<script>
function myFunction() {
  document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```

EVENT

Callback function

http://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_onclick

Error-first callback

When the readFile has finished
It calls the callback

The callback function

```
fs.readFile('/foo.txt', function(err, data) {
  // TODO: Error Handling Still Needed!
  console.log(data);
});
```

The cost of I/O

L1-cache	3 cycles
L2-cache	14 cycles
RAM	250 cycles
Disk	41 000 000 cycles
Network	240 000 000 cycles

http://thenodeway.io/posts/understanding-error-first-callbacks/

What if something goes wrong

```
fs.readFile('/foo.txt', function(err, data) {
        // If an error occurred, handle it (throw, propagate, etc)
        if(err) {
                console.log('Unknown Error');
                return;
        }
        // Otherwise, log the file contents
        console.log(data);
});
```

http://thenodeway.io/posts/understanding-error-first-callbacks/

Error management (ignore, handle or propagate) if up to the caller!

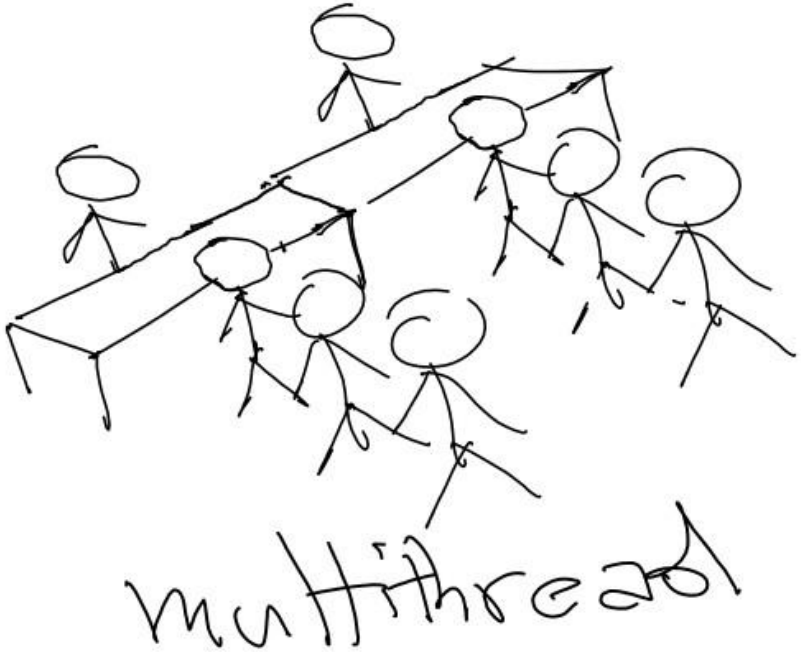
```
if(err) {
  // Handle "Not Found" by responding with a custom error page
  if(err.fileNotFound) {
    return this.sendErrorMessage('File Does not Exist');
  }
  // Ignore "No Permission" errors, this controller knows that we don't care
  // Propagate all other errors (Express will catch them)
  if(!err.noPermission) {
    return next(err);
  }
}
```

http://thenodeway.io/posts/understanding-error-first-callbacks/

The non-blocking notion

- Single threaded
- Instead of waiting use events
- Never wait for IO (socket, disk etc.)
- JS is natural for building async programs
- For blocking operation the node internals uses thread pool to wait for operations to finish.


monothread


multithread


An hot dog
Sir the hamburger is ready!
Event

Traditional I/O

```
var data = file.read('file.txt');
process(data);
```

Traditional I/O

```
var data = file.read('file.txt');
 ZzZzZZZzz…
process(data);
```

Why wasting those cycles?!

Non-Blocking I/O

```
file.read('file.txt', function (data) {
    process(data);
    return success;
});

DoWhateverYouWishMeanwhile();
```

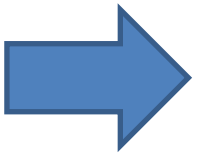
Callback Hell

```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

http://callbackhell.com/

From callback to promises

```
function isUserTooYoung(id, callback) {
  openDatabase(function(db) {
    getCollection(db, 'users', function(col) {
      find(col, {'id': id},function(result) {
        result.filter(function(user) {
          callback(user.age < cutoffAge)
        }
      }
    }
  }
}
```



```
function isUserTooYoung(id) {
  return openDatabase(db)
    .then(getCollection)
    .then(find.bind(null, {'id': id}))
    .then(function(user) {
      return user.age < cutoffAge;
    });
}
```

require('<name of module>'

External modules

Node modules (fs,tcp http)

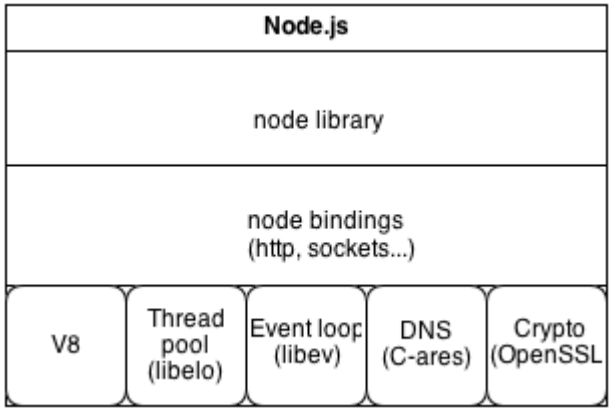
Node internals

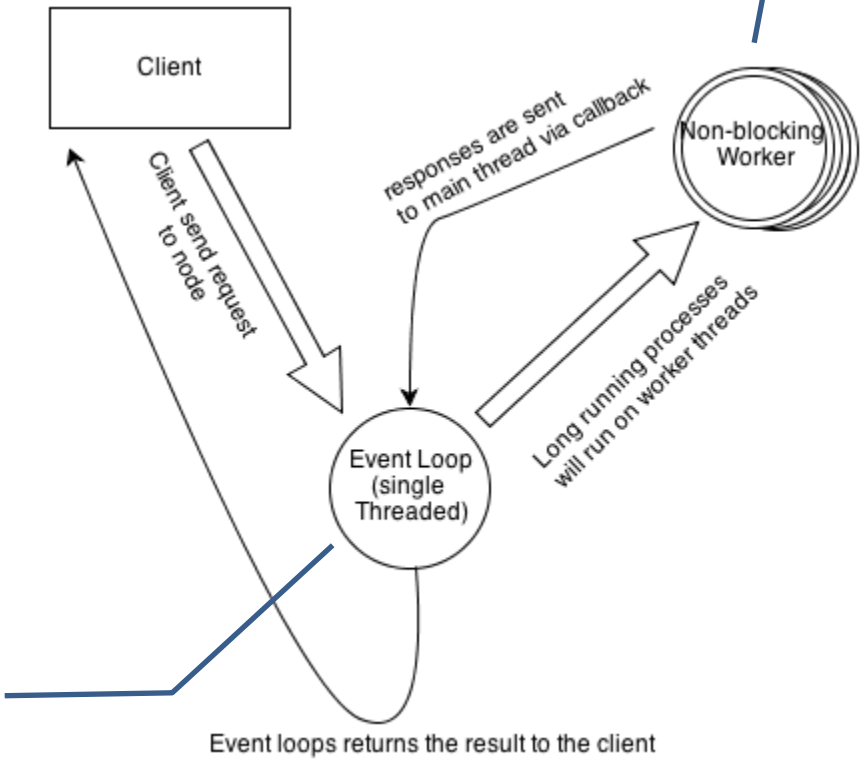
JS Interpreter

Google V8

Internals

Multi Thread → long running jobs hadled over to internal workers threads





Non blocking I/O → Events →
Single Thread → avoids context switching

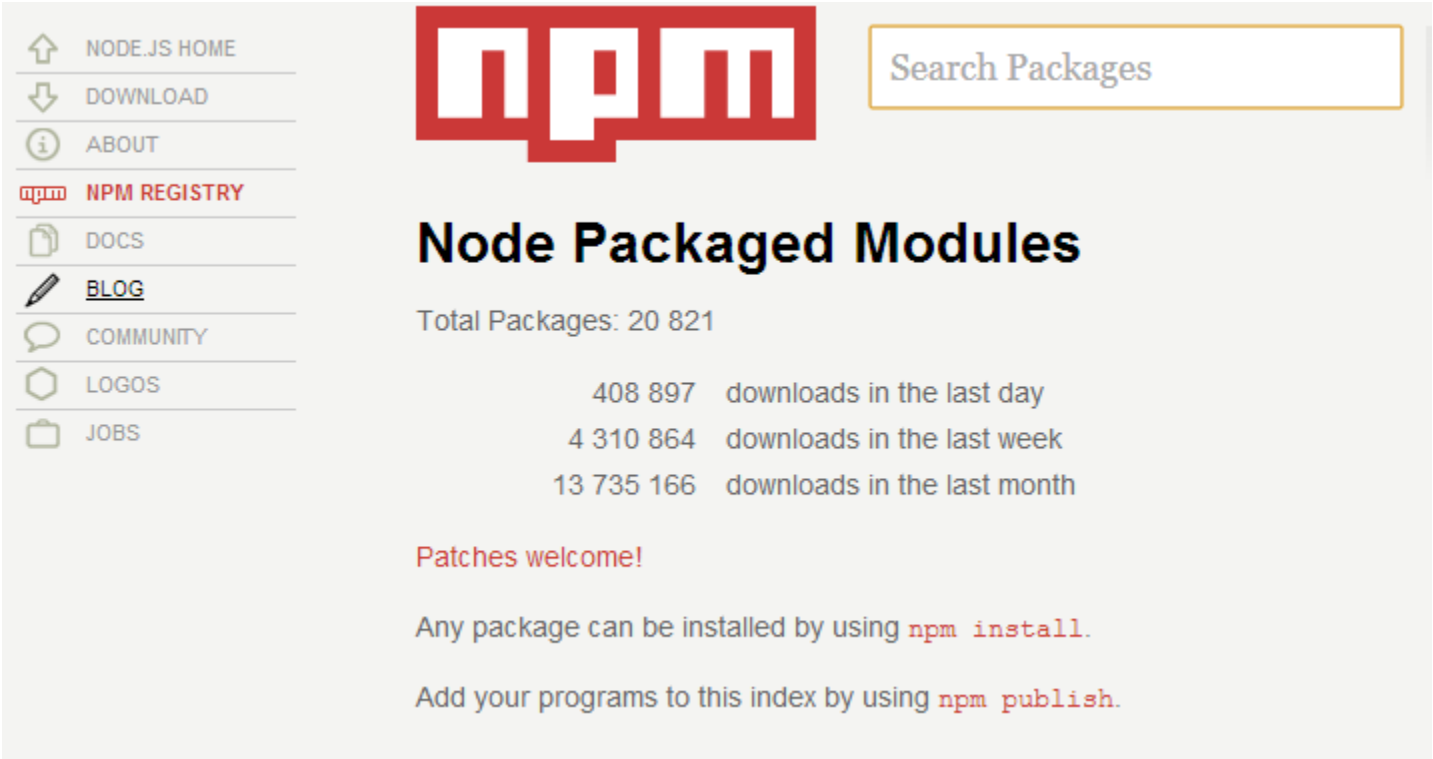
http://orange-coding.net/2013/06/29/xfiles-part-i-learning-how-to-walk/

Node Modules

The true force of node

NPM

- NPM is a package manager for node.js
  - http://npmjs.org/



Express Package

- RESTful module for node webapps
- Supports cookies, sessions, caching etc.
- www.expressjs.com

Example of Express API

```
var Express = require('express'),
     app = Express.createServer();

app.get('/users/(:user)/?', function (req, res) {
       res.send('hello ' + req.params.user);
});

app.listen(process.env.PORT || process.argv[3] || 8080);
```

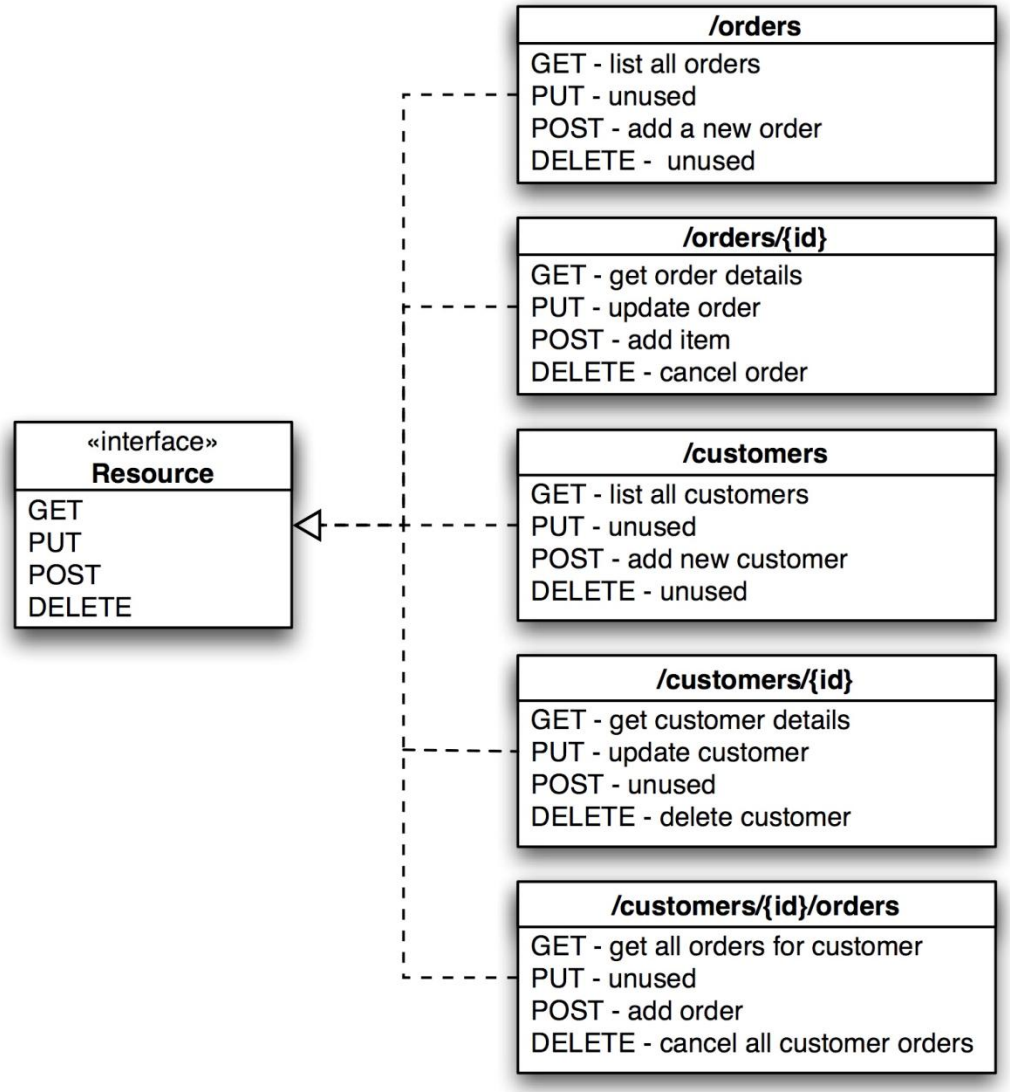
http://expressjs.com/

PROS AND CONS OF NODE.JS

Pros	Cons
1. Asynchronous event driven IO helps concurrent request handling.	1. Node.js doesn't provide scalability. One CPU is not going to be enough; the platform provides no ability to scale out to take advantage of the multiple cores commonly present in today's server-class hardware.
2. Uses JavaScript, which is easy to learn.	2. Dealing with relational database is a pain if you are using Node.
3. Share the same piece of code with both server and client side.	3. Every time using a callback end up with tons of nested callbacks.
4. npm, the Node packaged modules has already become huge, and still growing.	4. Without diving in depth of JavaScript if someone starts Node, he may face conceptual problem.
5. Active and vibrant community, with lots of code shared via github, etc.	5. Node.js is not suited for CPU-intensive tasks. It is suited for I/O stuff only (like web servers).
6. You can stream big files.	

http://voidcanvas.com/describing-node-js/

REST


«interface»
Resource
GET
PUT
POST
DELETE
/orders
GET - list all orders
PUT - unused
POST - add a new order
DELETE - unused
/orders/{id}
GET - get order details
PUT - update order
POST - add item
DELETE - cancel order
/customers
GET - list all customers
PUT - unused
POST - add new customer
DELETE - unused
/customers/{id}
GET - get customer details
PUT - update customer
POST - unused
DELETE - delete customer
/customers/{id}/orders
GET - get all orders for customer
PUT - unused
POST - add order
DELETE - cancel all customer orders