

Real Time Web

WITH NODE.JS

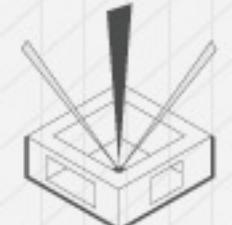
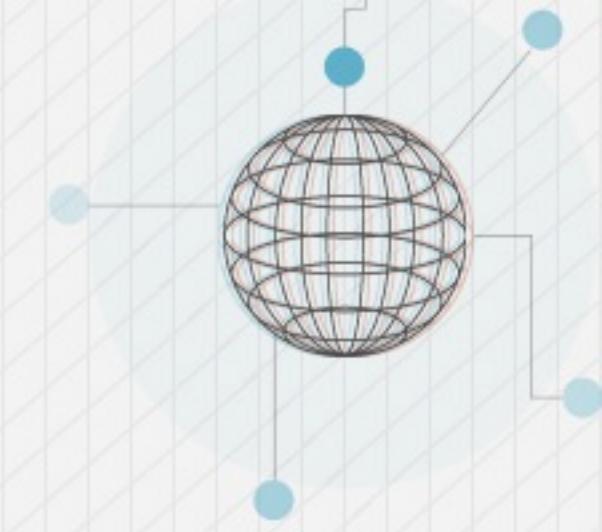
```
var readline = require('readline'),  
rl = readline.createInterface(process.stdin,  
process.stdout),  
prefix = 'OHAI> '  
  
rl.on('line', function(line) {  
  switch(line.trim()) {  
    case 'hello':  
      console.log('world!');  
      break;  
    default:  
      console.log('Say what? I might have heard ' +  
line.trim() + ''');  
      break;  
  }  
  rl.setPrompt(prefix, prefix.length);  
  rl.prompt();  
}).on('close', function() {  
  console.log('Have a great day!');  
  process.exit(0);  
});
```

```
console.log('Starting directory: ' +  
process.cwd());  
try {  
  process.chdir('/tmp');  
  console.log('New directory: ' +  
process.cwd());  
}  
catch (err) {  
  console.log('chdir: ' + err);  
}
```



INTRO TO NODE.JS

- LEVEL ONE -





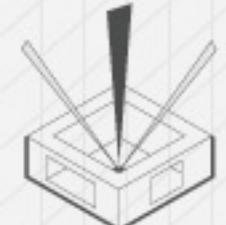
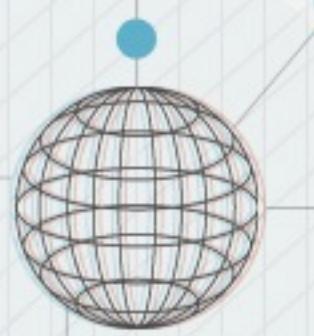
WHAT IS NODE.JS?

Allows you to build scalable network applications using JavaScript on the server-side.

Node.js

V8 JavaScript Runtime

It's fast because it's mostly C code



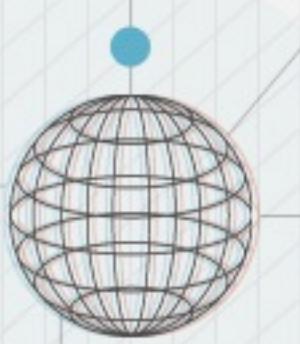
INTRO TO NODE.JS





WHAT COULD YOU BUILD?

- **Websocket Server** *Like a chat server*
- **Fast File Upload Client**
- **Ad Server**
- **Any Real-Time Data Apps**

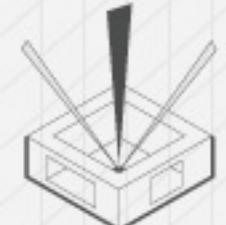
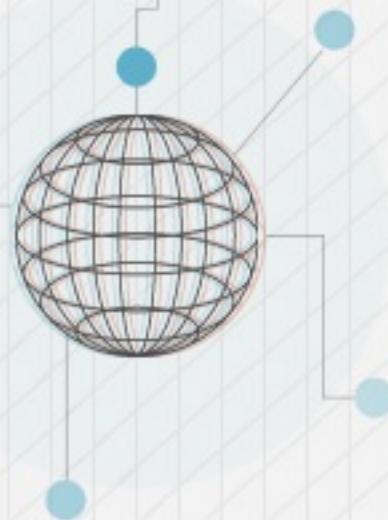




WHAT IS NODE.JS NOT?

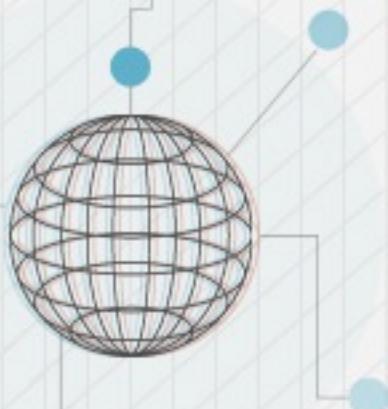
- A Web Framework
- For Beginners *It's very low level*
- Multi-threaded

You can think of it as a single threaded server





OBJECTIVE: PRINT FILE CONTENTS



- Blocking Code

Read file from Filesystem, set equal to “contents”

Print contents

Do something else

- Non-Blocking Code

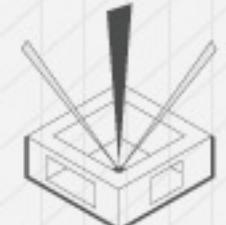
Read file from Filesystem

whenever you’re complete, print the contents

Do Something else

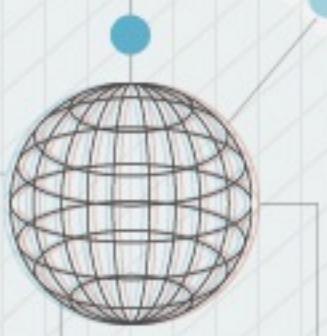


This is a “Callback”





BLOCKING VS NON-BLOCKING



- Blocking Code

```
var contents = fs.readFileSync('/etc/hosts');
console.log(contents);
console.log('Doing something else');
```

Stop process until complete



- Non-Blocking Code

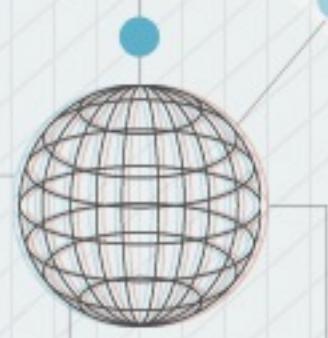
```
fs.readFile('/etc/hosts', function(err, contents) {
  console.log(contents);
});

console.log('Doing something else');
```





CALLBACK ALTERNATE SYNTAX



```
fs.readFile('/etc/hosts', function(err, contents) {  
  console.log(contents);  
});
```



Same as

```
var callback = function(err, contents) {  
  console.log(contents);  
}  
  
fs.readFile('/etc/hosts', callback);
```





BLOCKING VS NON-BLOCKING



```
var callback = function(err, contents) {
  console.log(contents);
}

fs.readFile('/etc/hosts', callback);
fs.readFile('/etc/inetcfg', callback);
```



blocking

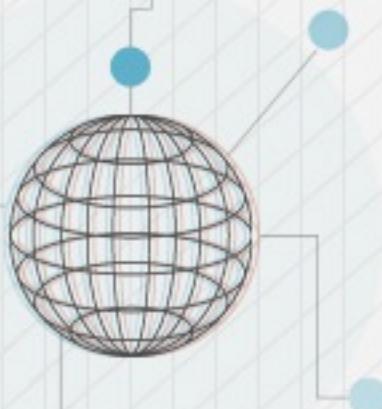


non-blocking





NODE.JS HELLO DOG



hello.js

```
var http = require('http'); How we require modules
```

```
http.createServer(function(request, response) {  
  response.writeHead(200); Status code in header  
  response.write("Hello, this is dog."); Response body  
  response.end(); Close the connection  
}).listen(8080, function(){ Listen for connections on this port  
  console.log('Listening on port 8080...');  
});
```

```
$ node hello.js Run the server
```

---> Listening on port 8080...

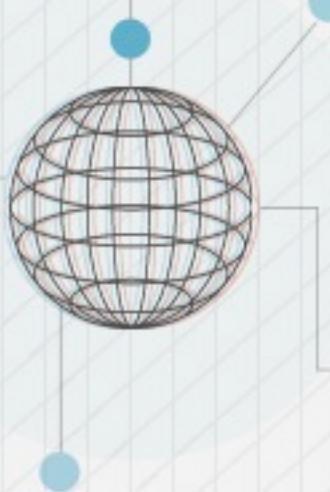
```
$ curl http://localhost:8080
```

----> Hello, this is dog.





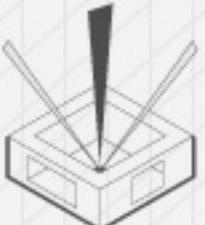
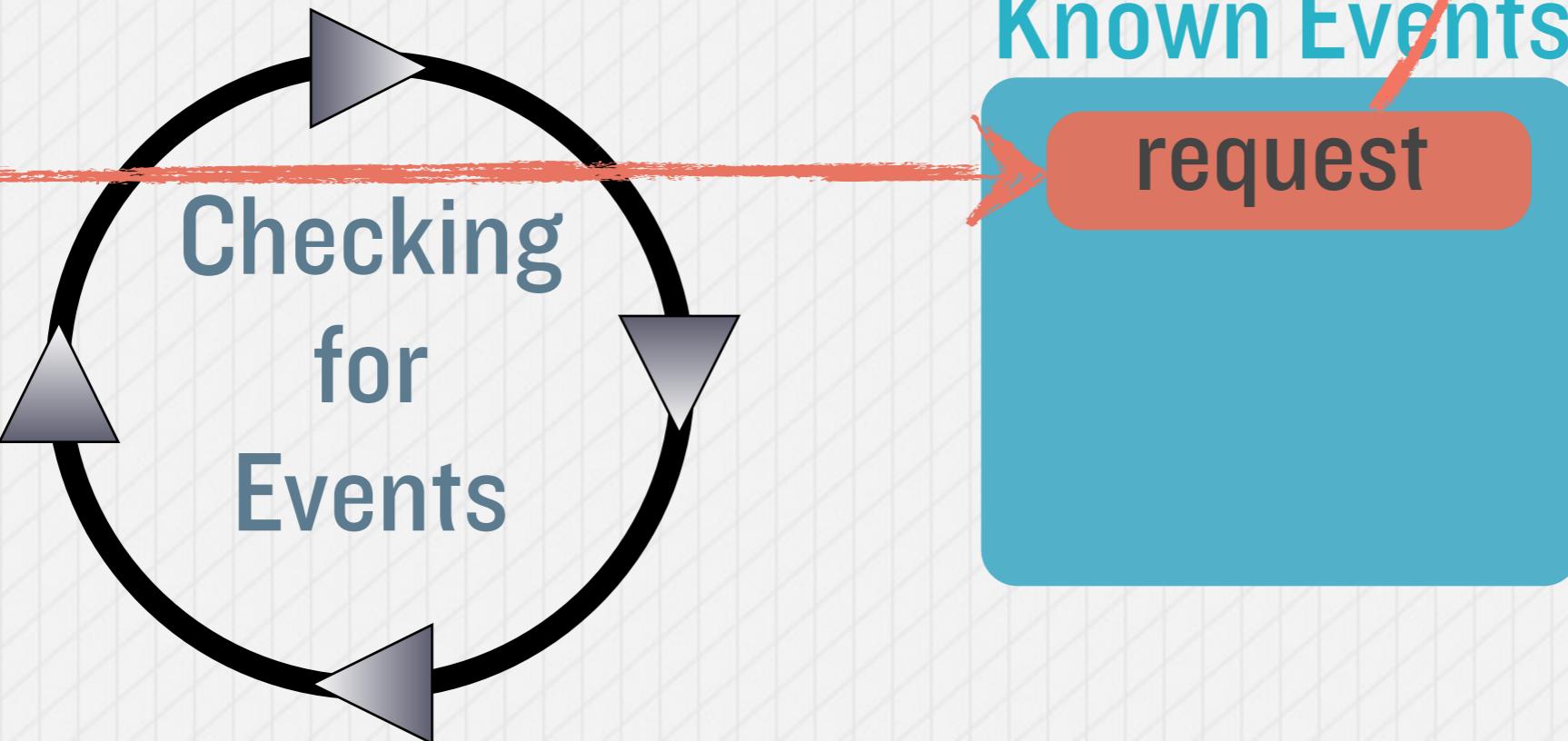
THE EVENT LOOP



```
var http = require('http');
http.createServer(function(request, response) {
  ...
}).listen(8080, function(){
  console.log('Listening on port 8080...');
});
```

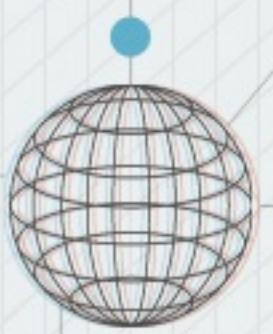
Starts the Event Loop when finished

Run the Callback



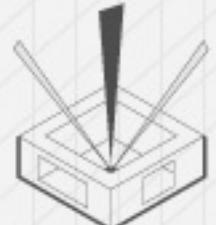


WHY JAVASCRIPT?



“JavaScript has certain characteristics that make it very different than other dynamic languages, namely that it has no concept of threads. Its model of concurrency is completely based around events.”

- Ryan Dahl

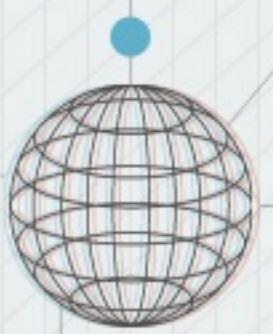


INTRO TO NODE.JS

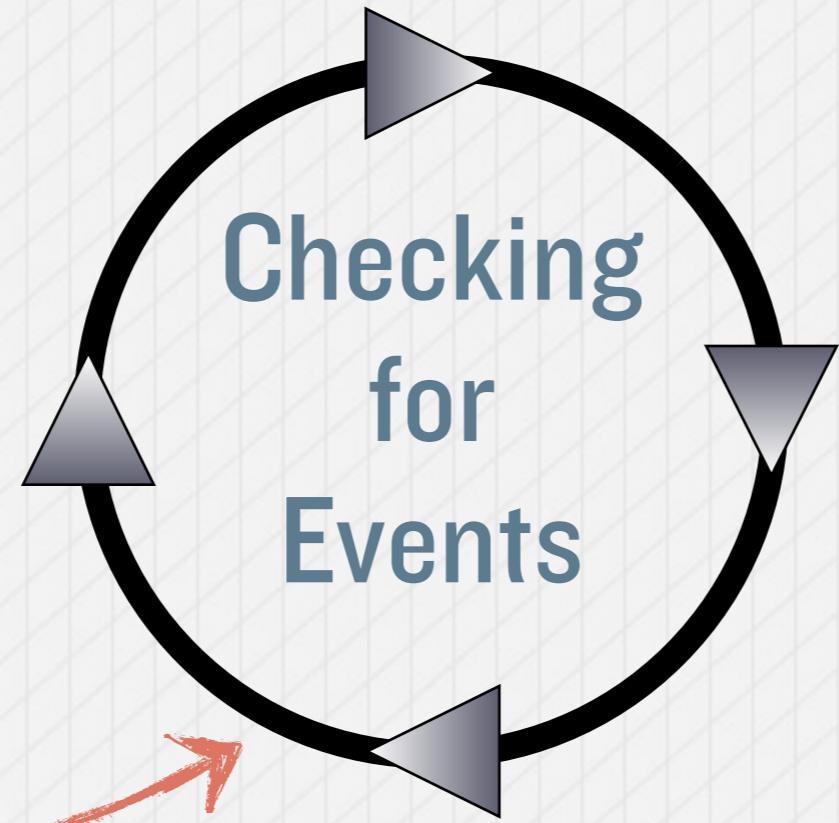
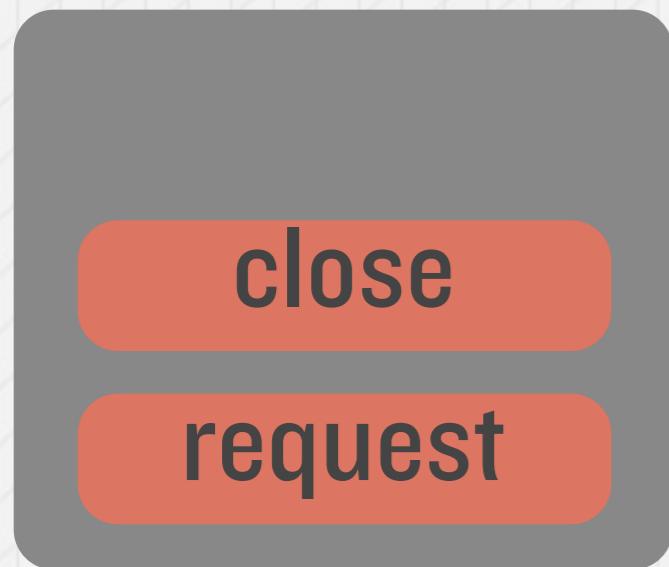




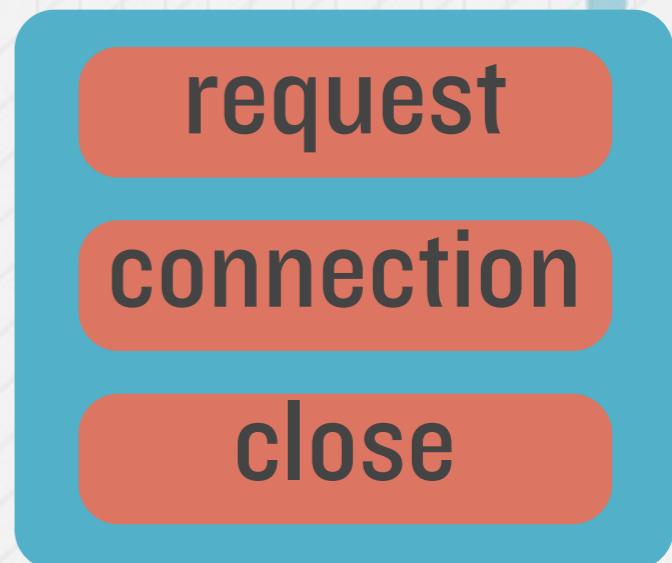
THE EVENT LOOP



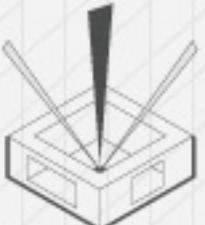
Event Queue



Known Events

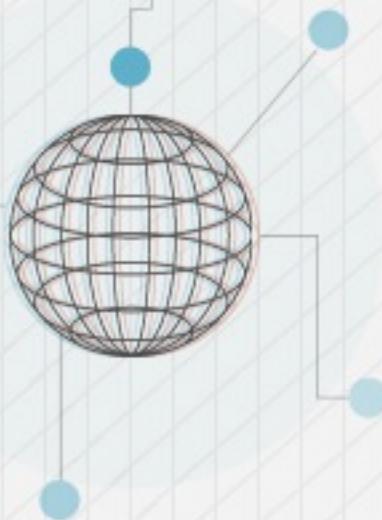


Events processed one at a time



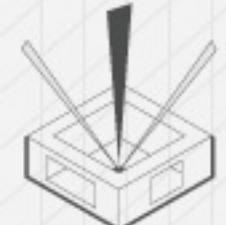


WITH LONG RUNNING PROCESS



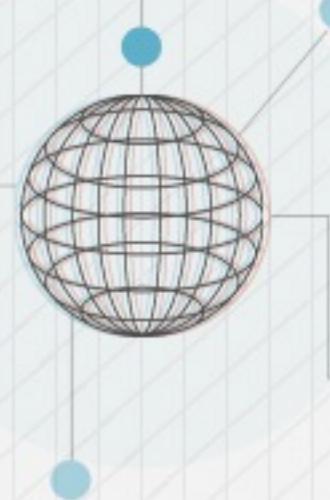
```
var http = require('http');

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Dog is running.");
  setTimeout(function(){   Represent long running process
    response.write("Dog is done.");
    response.end();
  }, 5000);  5000ms = 5 seconds
}).listen(8080);
```





TWO CALLBACKS HERE

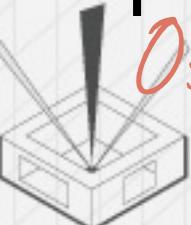
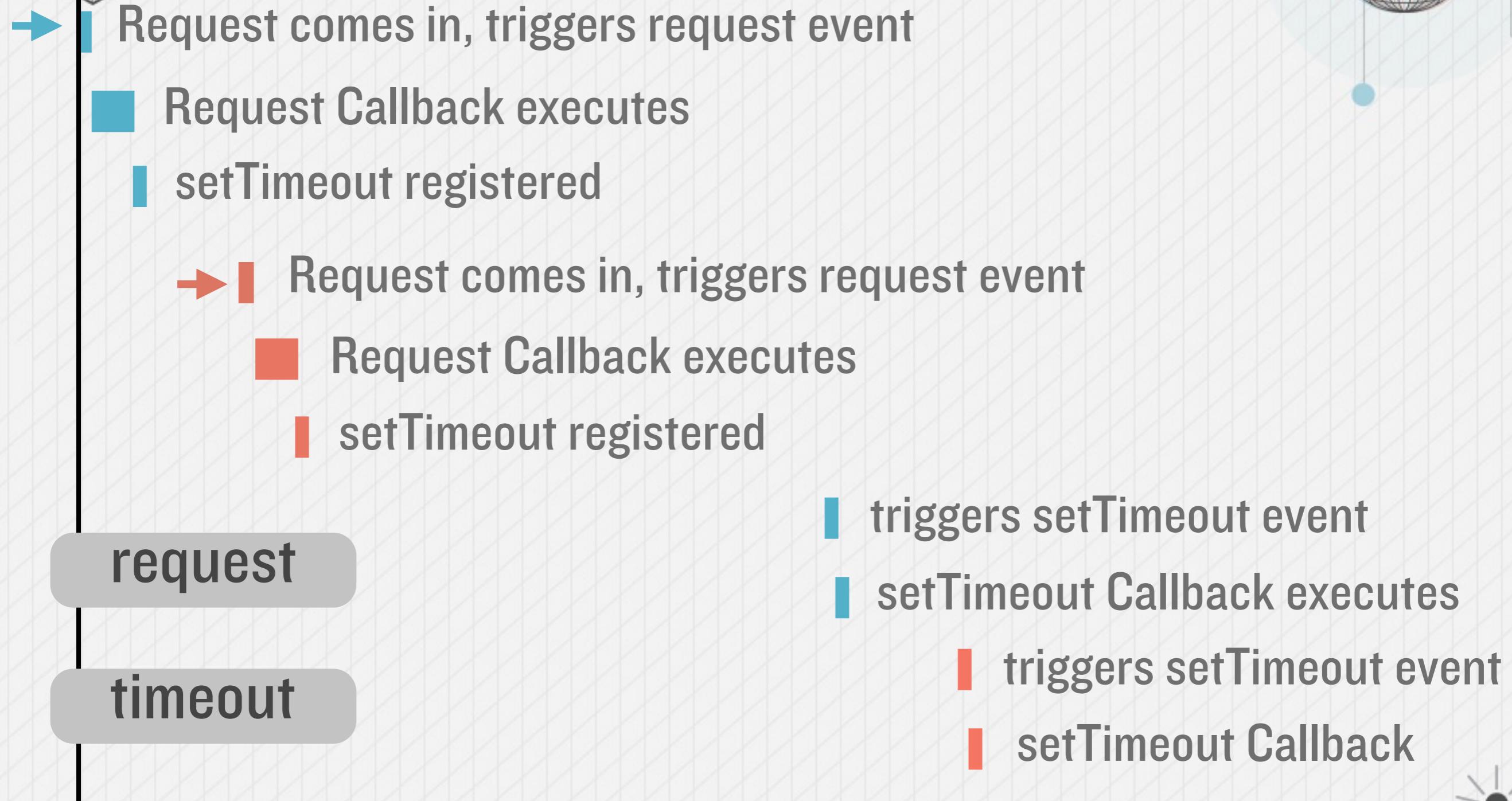
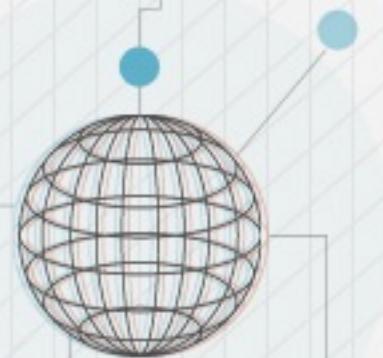


```
var http = require('http');
```

```
http.createServer(function(request, response) { ← request
  response.writeHead(200);
  response.write("Dog is running.");
  setTimeout(function(){ ← timeout
    response.write("Dog is done.");
    response.end();
  }, 5000);
}).listen(8080);
```



TWO CALLBACKS TIMELINE



WITH BLOCKING TIMELINE

→ Request comes in, triggers request event

Request Callback executes

setTimeout executed

Request comes in, waits for server



Wasted Time

triggers setTimeout event

setTimeout Callback executed

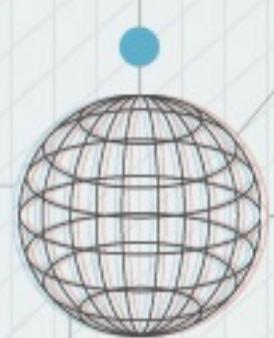
Request comes in

Request Callback executes

0s

5s

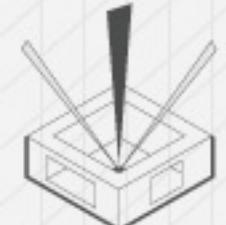
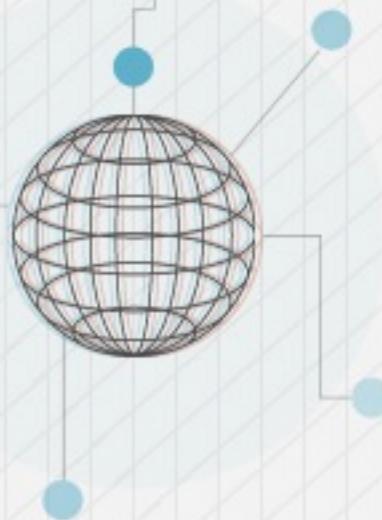
10s





TYPICAL BLOCKING THINGS

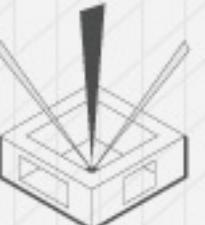
- Calls out to web services
- Reads/Writes on the Database
- Calls to extensions





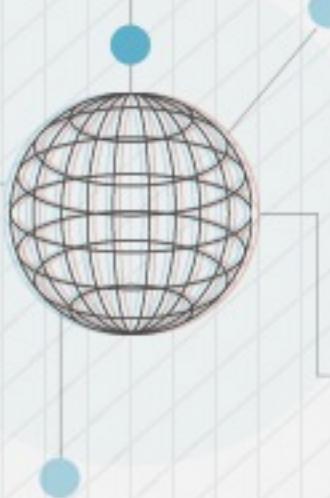
EVENTS

- LEVEL TWO -

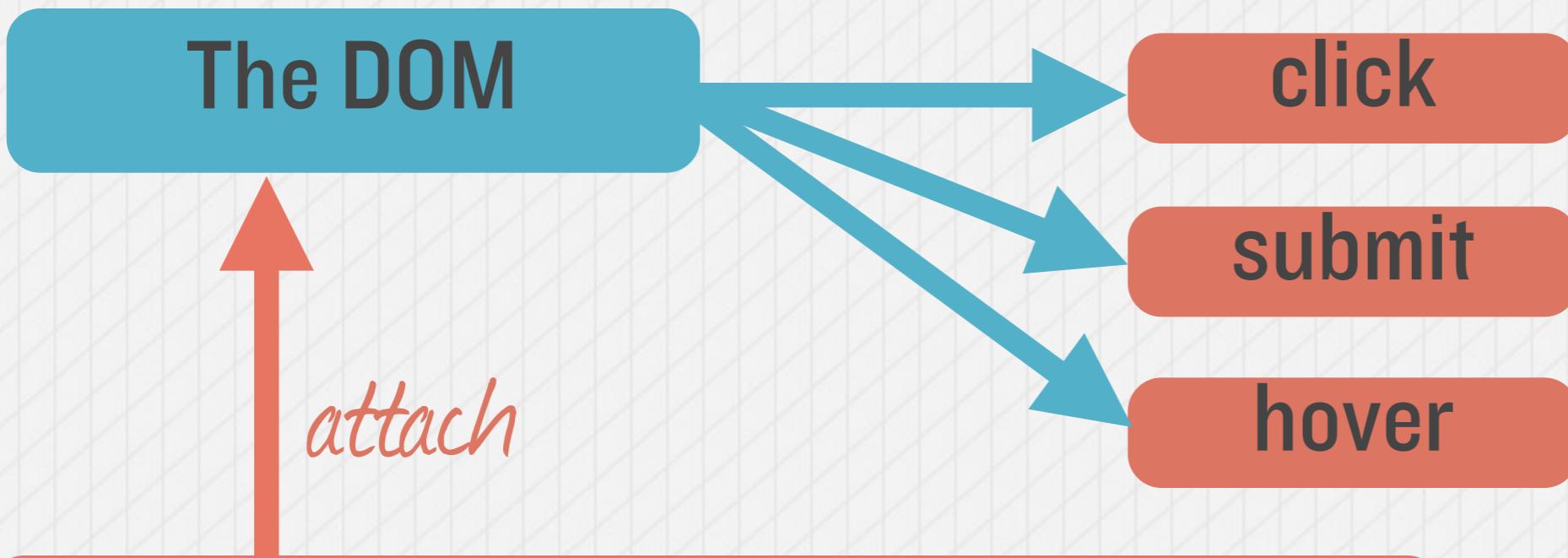




EVENTS IN THE DOM



The DOM triggers Events
you can listen for those events



```
$("p").on("click", function(){ ... });
```

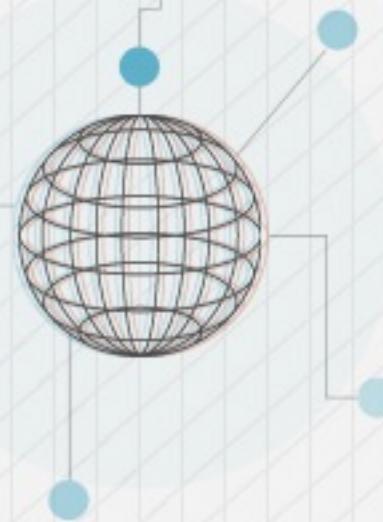
When 'click' event is triggered

EVENTS

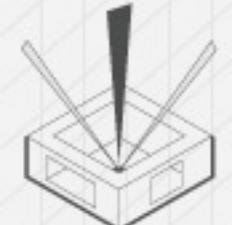
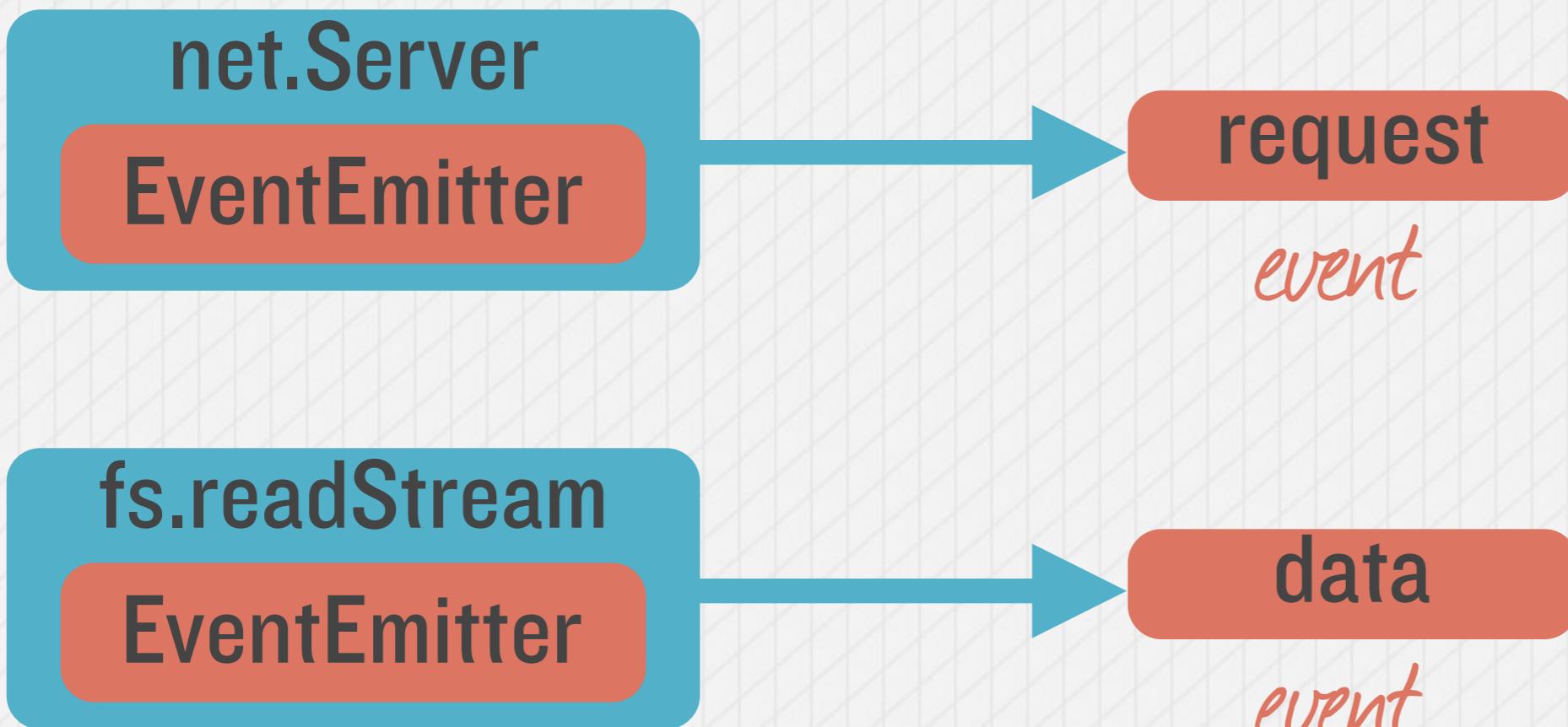




EVENTS IN NODE



Many objects in Node emit events

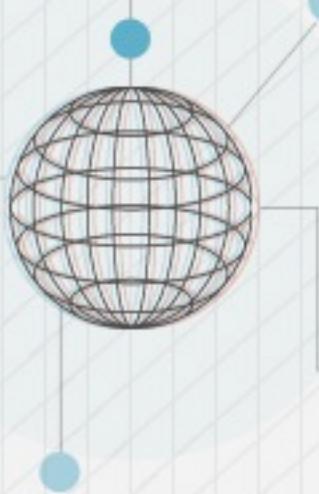


EVENTS





CUSTOM EVENT EMITTERS



```
var EventEmitter = require('events').EventEmitter;
```

```
var logger = new EventEmitter();
```

error

events

warn

info

```
logger.on('error', function(message){  
  console.log('ERR: ' + message);  
});
```

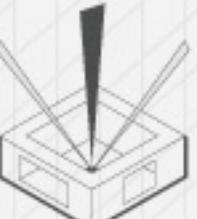
listen for error event

```
logger.emit('error', 'Spilled Milk');
```

- → ERR: Spilled Milk

```
logger.emit('error', 'Eggs Cracked');
```

- → ERR: Eggs Cracked

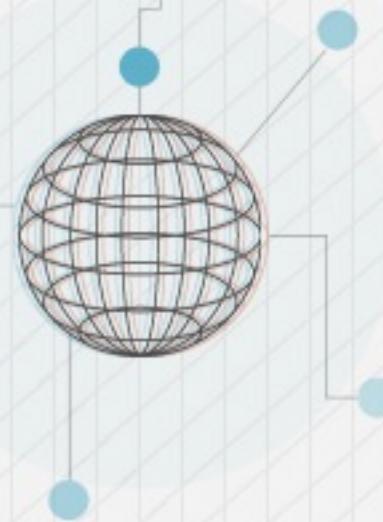


EVENTS





EVENTS IN NODE



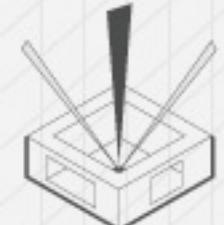
Many objects in Node emit events



attach

```
function(request, response){ .. }
```

When 'request' event is emitted



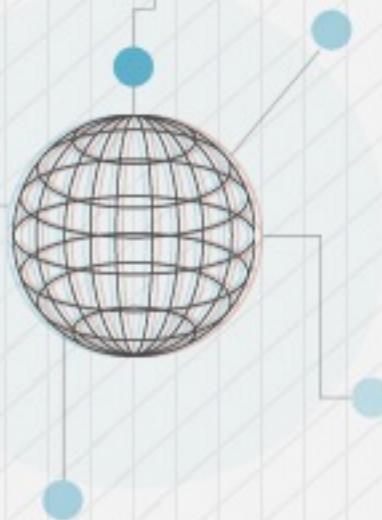
EVENTS





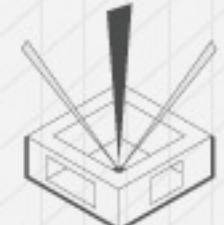
HTTP ECHO SERVER

```
http.createServer(function(request, response){ ... });
```



But what is really going on here?

<http://nodejs.org/api/>

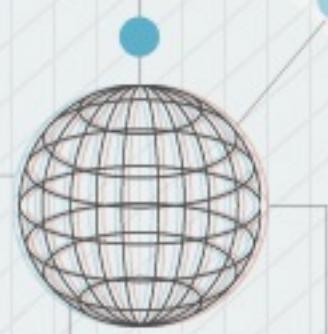


EVENTS





BREAKING IT DOWN



```
http.createServer(function(request, response){ ... });
```

http.createServer([requestListener])

Returns a new web server object.

The `requestListener` is a function which is automatically added to the `'request'` event.

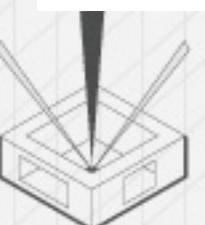
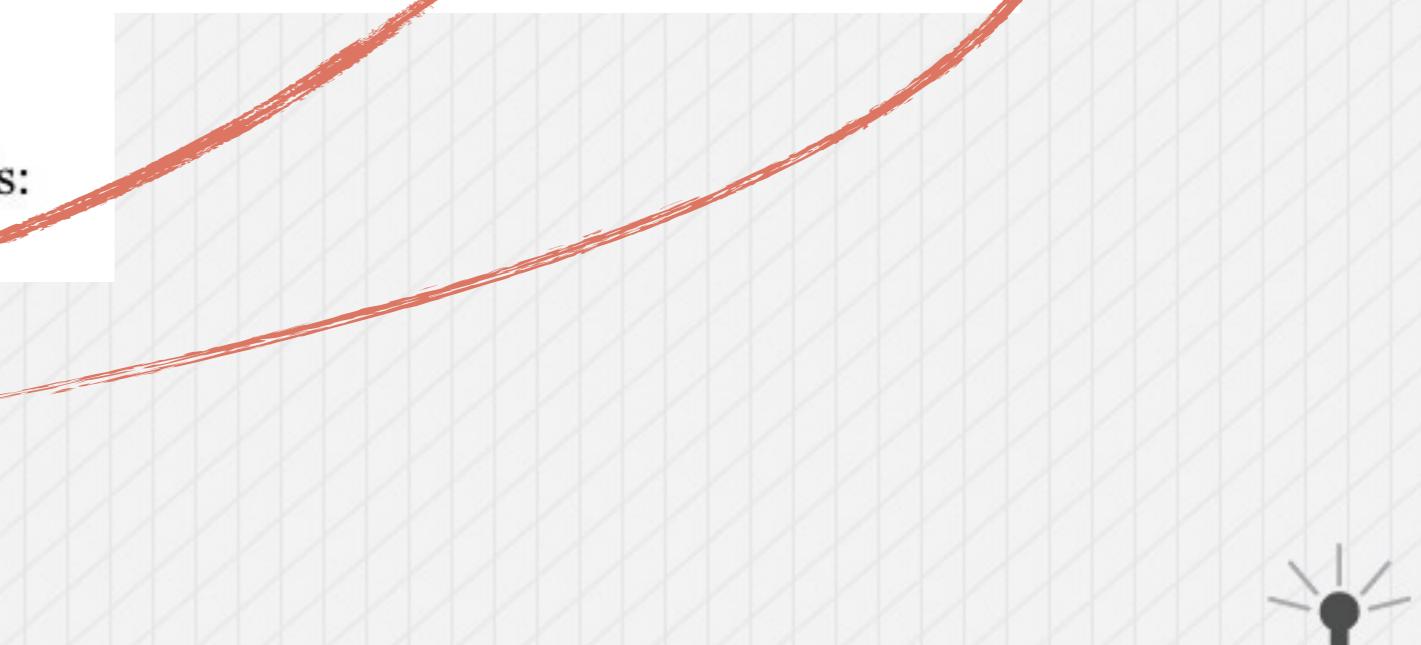
Class: http.Server

This is an `EventEmitter` with the following events:

Event: 'request'

```
function (request, response) { }
```

Emitted each time there is a request.

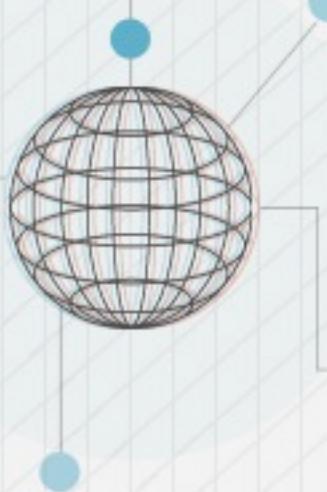


EVENTS





ALTERNATE SYNTAX



```
http.createServer(function(request, response){ ... });
```

Same as

```
var server = http.createServer();
server.on('request', function(request, response){ ... });
```

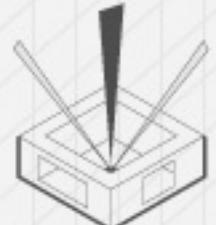
This is how we
add event listeners

```
server.on('close', function(){ ... });
```

Event: 'close'

```
function () { }
```

Emitted when the server closes.



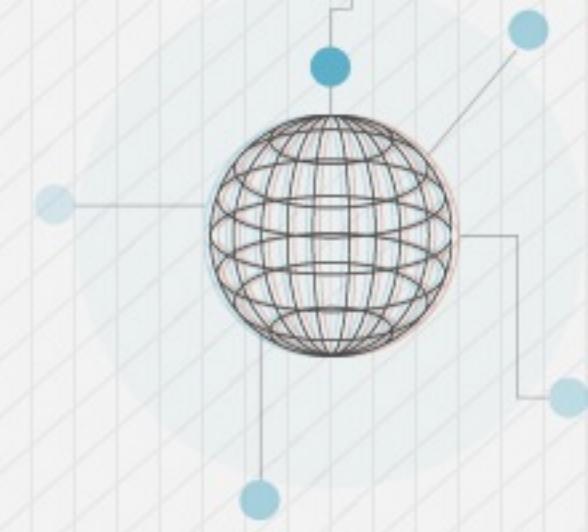
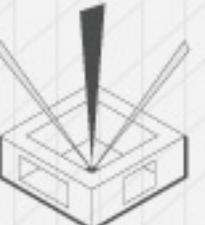
EVENTS



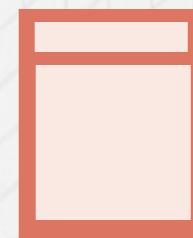
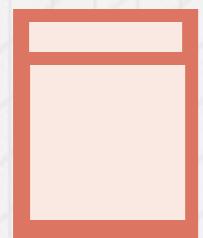


STREAMS

- LEVEL THREE -



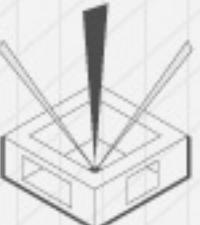
WHAT ARE STREAMS?



*Start Processing
Immediately*

Streams can be readable, writeable, or both

The API described here is for
streams in Node version v0.10.x
a.k.a. streams2

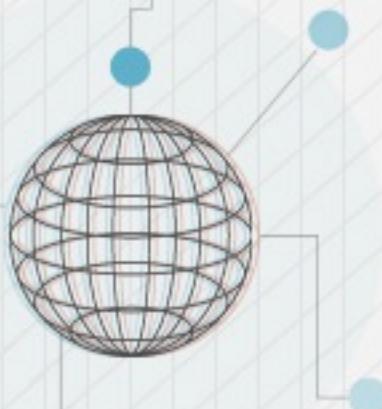


STREAMS





STREAMING RESPONSE



readable stream ↘ ↗ *writable stream*

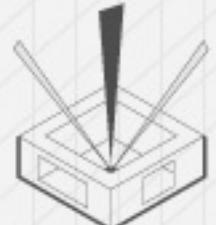
```
http.createServer(function(request, response) {  
  response.writeHead(200);  
  response.write("<p>Dog is running.</p>");  
  setTimeout(function(){  
    response.write("<p>Dog is done.</p>");  
    response.end();  
  }, 5000);  
}).listen(8080);
```

Our browser receives

"Dog is running."

(5 seconds later)

"Dog is done."



STREAMS





HOW TO READ FROM THE REQUEST?



Readable Stream
EventEmitter



Let's print what we receive from the request.

```
http.createServer(function(request, response) {  
  response.writeHead(200);  
  request.on('readable', function() {  
    var chunk = null;  
    while (null !== (chunk = request.read())) {  
      console.log(chunk.toString());  
    }  
  });  
  request.on('end', function() {  
    response.end();  
  });  
}).listen(8080)
```

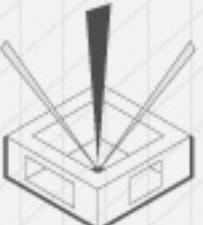




HOW TO READ FROM THE REQUEST?

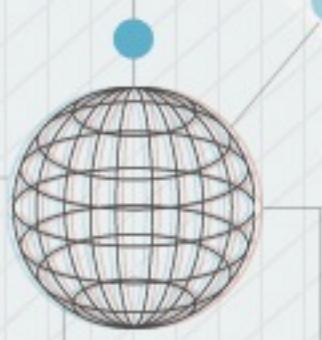


```
http.createServer(function(request, response) {  
  response.writeHead(200);  
  request.on('readable', function() {  
    var chunk = null;  
    while (null !== (chunk = request.read())) {  
      response.write(chunk);  
    }  
  });  
  request.on('end', function() {  
    response.end();  
  });  
}).listen(8080)
```





LET'S CREATE AN ECHO SERVER!



```
http.createServer(function(request, response) {  
  response.writeHead(200);  
  request.pipe(response);  
}).listen(8080)
```

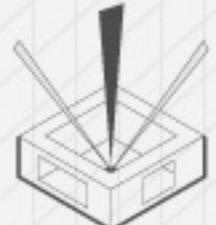


```
$ curl -d 'hello' http://localhost:8080
```

---→ Hello *on client*

Kinda like on the command line

```
cat 'bleh.txt' | grep 'something'
```



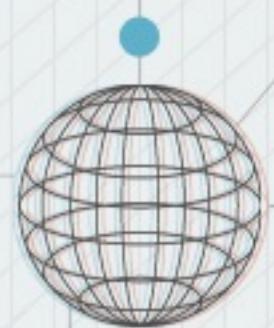
STREAMS





DOCUMENTATION

<http://nodejs.org/api/>



Stability Scores

File System

Stability: 3 - Stable

File I/O is provided by simple wrappers around standard POI objects. You can use `require('fs')`. All the methods have asynchronous and synchronous forms.

The asynchronous form always take a completion callback as the last argument. Completion callbacks depend on the method, but the first argument is always an error object. If no error object is present, then the operation was completed successfully, then the first argument is `null`.

When using the synchronous form any exceptions are immediately thrown as exceptions or allow them to bubble up.

Here is an example of the asynchronous version:

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

fs.unlink('/tmp/hello', function (err) {
```

STREAMS

Stream

Stability: 2 - Unstable

A stream is an abstract interface implemented by various objects in Node.js. Every `HTTP` request and response, every `server` and `client`, every `process`, every `socket`, every `file`, and even `stdout` and `stdin` are streams. Streams are readable, writable, or both.

You can load the Stream base classes by doing `require('stream')`. There are four main types of streams: Readable streams, Writable streams, Duplex streams, and Transform streams.

This document is split up into 3 sections. The first explains the parts of the Stream API that you need to use in your programs. If you never implement a streaming API yourself, then this section is for you.

The second section explains the parts of the API that you need to use if you implement a streaming API yourself. The API is designed to make this easy for you to do.

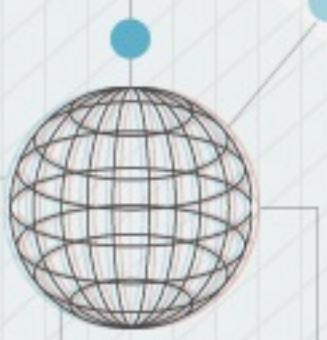
The third section goes into more depth about how streams work, including how to implement them and how to use them. It also contains some functions that you should probably not modify unless you definitely know what you're doing.

API for Stream Consumers





READING AND WRITING A FILE



```
var fs = require('fs');    require filesystem module
var file = fs.createReadStream("readme.md");
var newFile = fs.createWriteStream("readme_copy.md");

file.pipe(newFile);
```

[HOME](#) [DOCS](#) [CODE](#) [PLUGINS](#) [TWITTER](#)



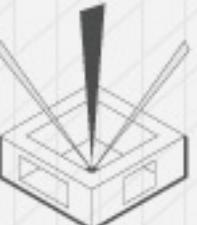
<http://gulpjs.com/>

Build system built on top of Streams



gulp.js

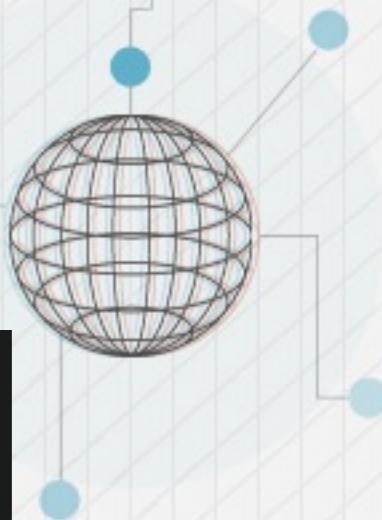
The streaming build system



STREAMS



UPLOAD A FILE



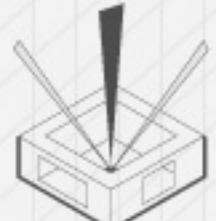
```
var fs = require('fs');
var http = require('http');

http.createServer(function(request, response) {
  var newFile = fs.createWriteStream("readme_copy.md");
  request.pipe(newFile);

  request.on('end', function() {
    response.end('uploaded!');
  });
}).listen(8080);
```

```
$ curl --upload-file readme.md http://localhost:8080
```

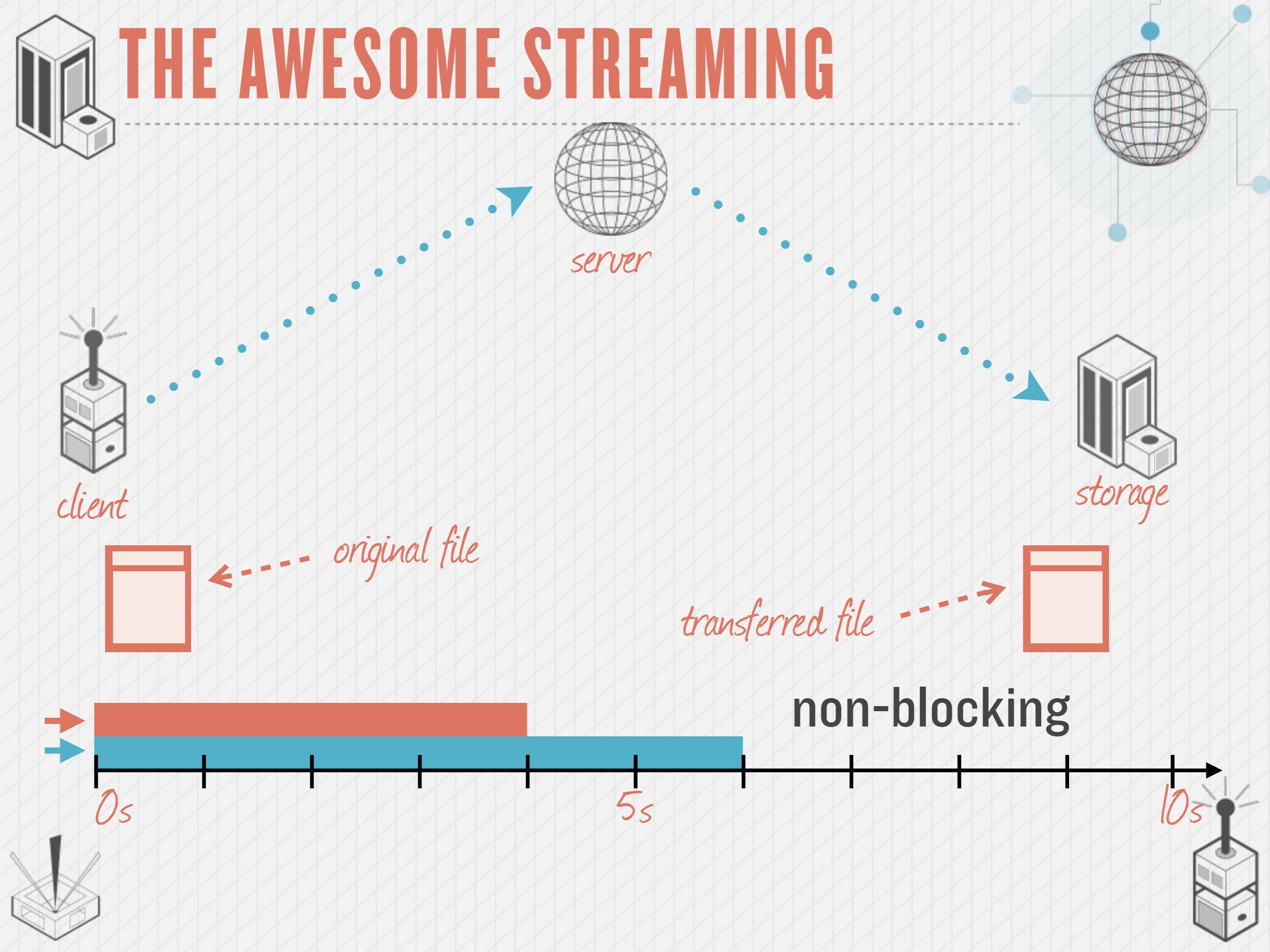
→ uploaded!



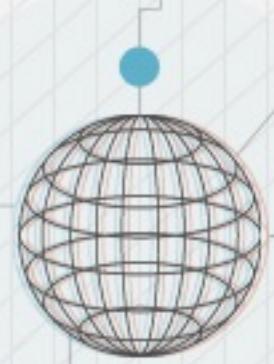
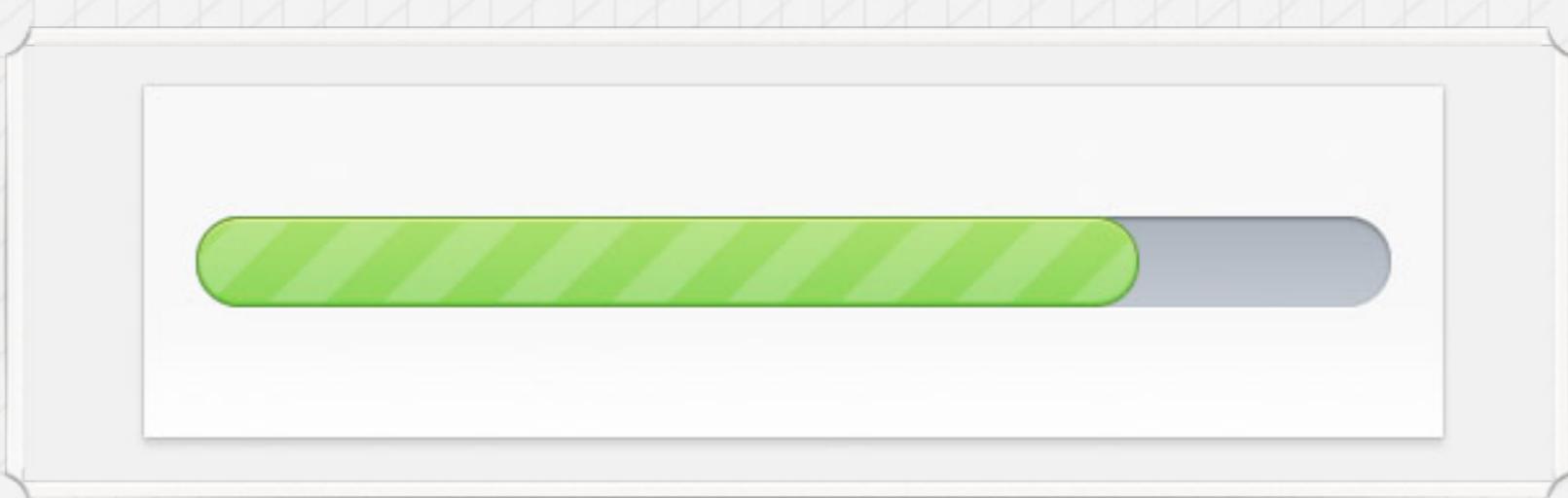
STREAMS



THE AWESOME STREAMING



FILE UPLOADING PROGRESS



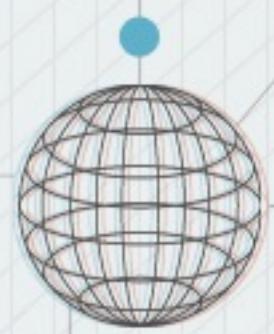
STREAMS





FILE UPLOADING PROGRESS

```
$ curl --upload-file file.jpg http://localhost:8080
```



Outputs:

```
progress: 3%
progress: 6%
progress: 9%
progress: 12%
progress: 13%
...
progress: 99%
progress: 100%
```

Choose File

No file chosen

Upload

We're going to need:

- **HTTP Server**
- **File System**

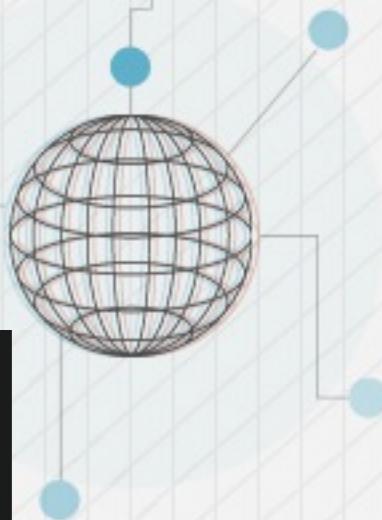


STREAMS





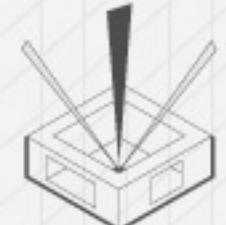
REMEMBER THIS CODE?



```
var fs = require('fs');
var http = require('http');

http.createServer(function(request, response) {
  var newFile = fs.createWriteStream("readme_copy.md");
  request.pipe(newFile);

  request.on('end', function() {
    response.end('uploaded!');
  });
}).listen(8080);
```

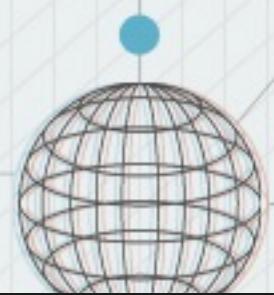


STREAMS



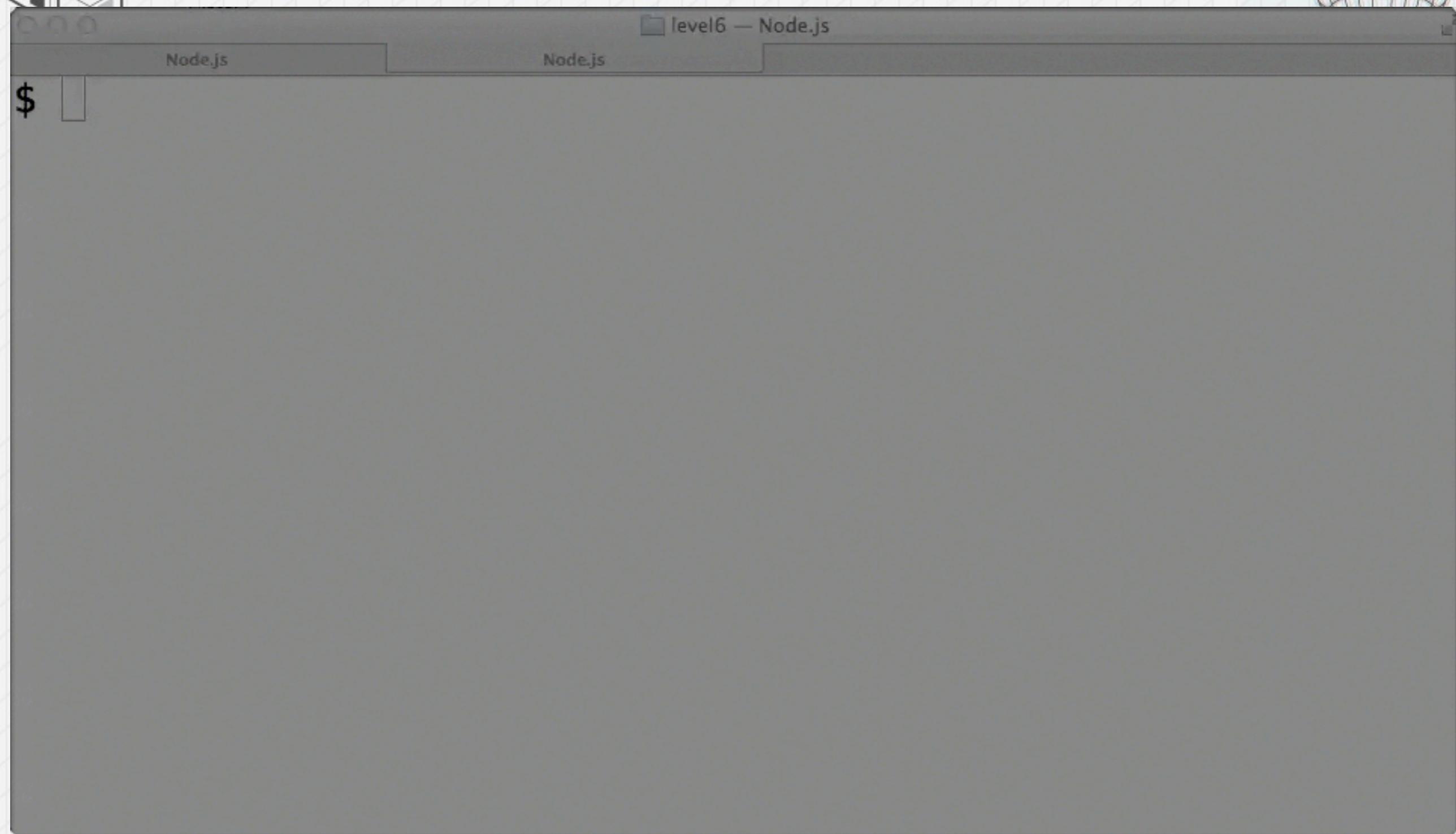


REMEMBER THIS CODE?



```
http.createServer(function(request, response) {  
  var newFile = fs.createWriteStream("readme_copy.md");  
  var fileBytes = request.headers['content-length'];  
  var uploadedBytes = 0;  
  
  request.on('readable', function() {  
    var chunk = null;  
    while(null !== (chunk = request.read())){  
      uploadedBytes += chunk.length;  
      var progress = (uploadedBytes / fileBytes) * 100;  
      response.write("progress: " + parseInt(progress, 10) + "%\n");  
    }  
  });  
  
  request.pipe(newFile);  
  ...  
}).listen(8080);
```

SHOWING PROGRESS

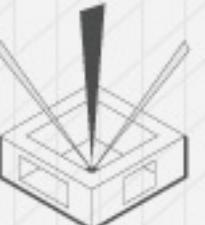


STREAMS



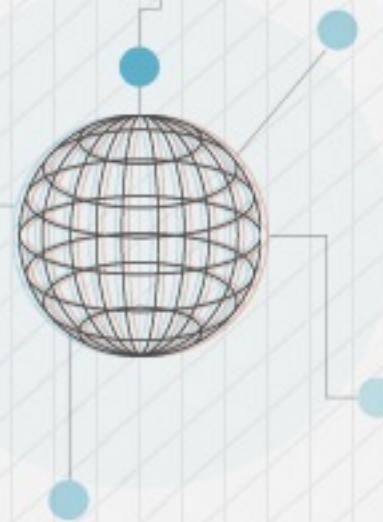
MODULES

- LEVEL FOUR -





REQUIRING MODULES



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

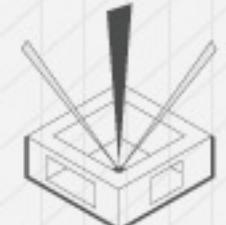
-----> **http.js**

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

-----> **fs.js**

How does 'require' return the libraries?

How does it find these files?

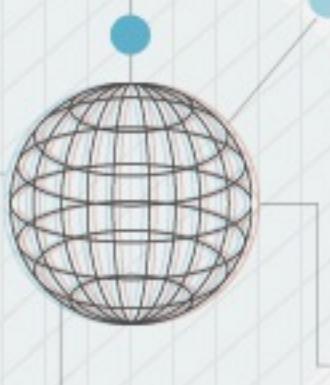


MODULES





LET'S CREATE OUR OWN MODULE



custom_hello.js

```
var hello = function() {  
  console.log("hello!");  
}  
  
module.exports = hello;
```

app.js

exports defines what require returns

```
var hello = require('./custom_hello');  
  
var gb = require('./custom_goodbye');  
  
hello();  
  
gb.goodbye();
```

```
require('./custom_goodbye').goodbye();
```

If we only need to call once

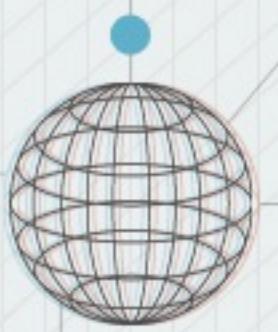


MODULES





EXPORT MULTIPLE FUNCTIONS

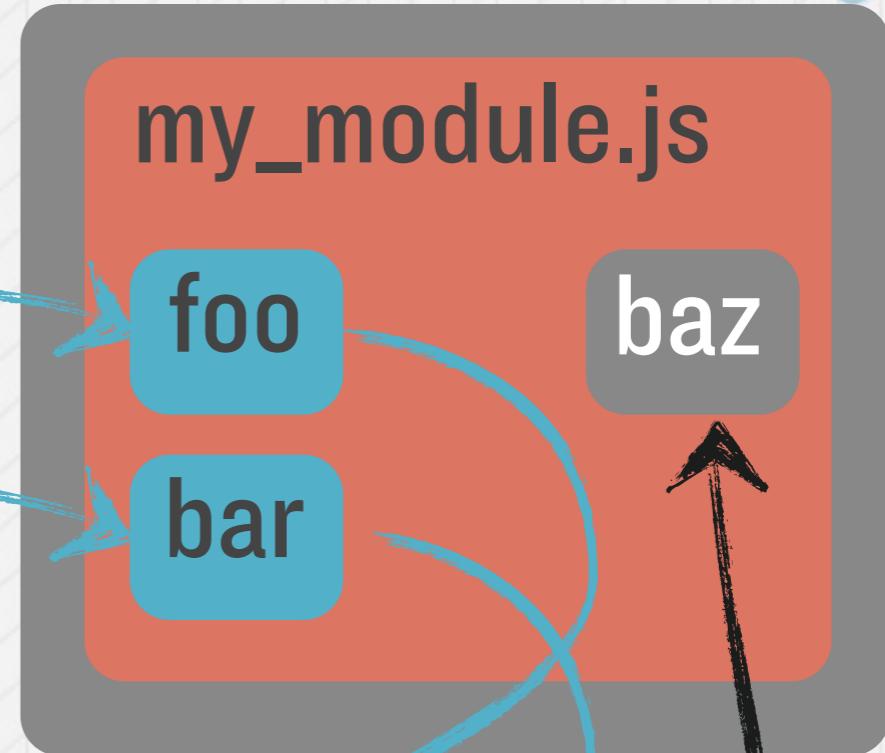


my_module.js

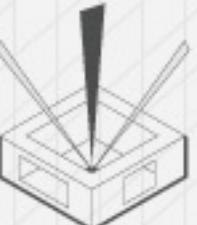
```
var foo = function() { ... }  
var bar = function() { ... }  
var baz = function() { ... }  
  
exports.foo = foo  
exports.bar = bar
```

app.js

```
var myMod = require('./my_module');  
myMod.foo();  
myMod.bar();
```



"private"

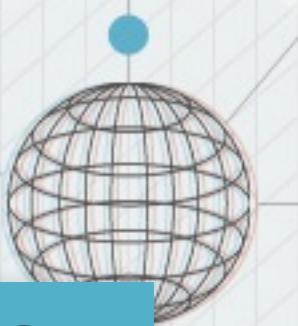


MODULES





MAKING HTTP REQUESTS

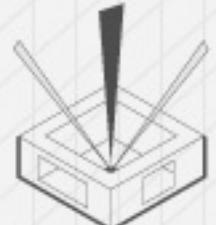


```
var http = require('http');
```

app.js

```
var message = "Here's looking at you, kid.";
var options = {
  host: 'localhost', port: 8080, path: '/', method: 'POST'
}
```

```
var request = http.request(options, function(response){
  response.on('data', function(data){
    console.log(data); logs response body
  });
});  
request.write(message); begins request  
request.end(); finishes request
```

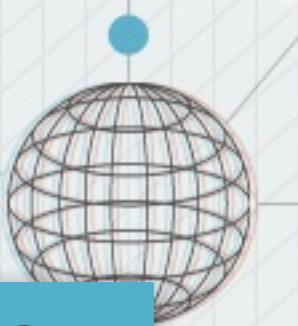


MODULES





ENCAPSULATING THE FUNCTION



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

app.js

```
var makeRequest = function(message) {
  var options = {
    host: 'localhost', port: 8080, path: '/', method: 'POST'
  }
```

```
var request = http.request(options, function(response){
  response.on('data', function(data){
    console.log(data);
  });
});
request.write(message);
request.end();
}
```

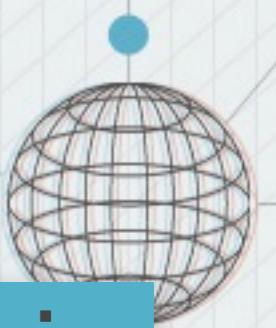
```
makeRequest("Here's looking at you, kid.");
```



MODULES



CREATING & USING A MODULE



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

make_request.js

```
var makeRequest = function(message) {  
  ...  
}  
  
module.exports = makeRequest;
```

```
var makeRequest = require('./make_request');
```

app.js

```
makeRequest("Here's looking at you, kid");  
makeRequest("Hello, this is dog");
```

Where does require look for modules?

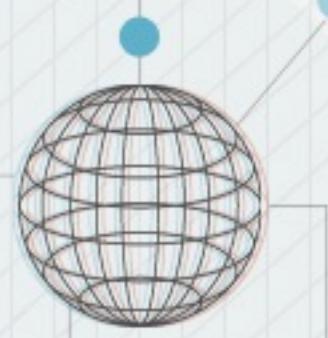


MODULES





REQUIRE SEARCH



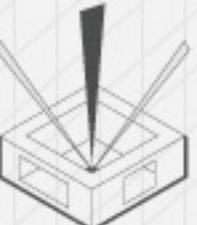
```
var make_request = require('./make_request') look in same directory
var make_request = require('../make_request') look in parent directory
var make_request = require('/Users/eric/nodes/make_request')
```

/Home/eric/my_app/app.js

Search in node_modules directories

```
var make_request = require('make_request')
```

- /Home/eric/my_app/node_modules/make_request.js
- /Home/eric/node_modules/make_request.js
- /Home/node_modules/make_request.js
- /node_modules/make_request.js



MODULES

However, all packages are directories





NPM: THE USERLAND SEA

Package manager for node

- Comes with node
- Module Repository
- Dependency Management
- Easily publish modules

<http://npmjs.org>

The screenshot shows the npmjs.org homepage with the following sections:

- HOME**: API, BLOG, NODEJS, JOBS.
- npm**: Search Packages.
- Total Packages: 61 828**
- Patches welcome!**
- Add your programs to this index by using `npm publish`.**
- Recently Updated**: A list of recently updated packages including `raspivid`, `grider`, `ensure-it`, `raw-body`, `node-linkedin`, `grunt-markdown-blog`, `winescript`, `bare`, `sweets-br brittle`, and `component-builder2`.
- Most Depended Upon**: A list of modules that are highly depended upon, such as `underscore`, `async`, `request`, `commander`, `express`, `optimist`, `lodash`, `coffee-script`, `colors`, and `mkdirp`.
- Most Starred**: A list of modules with the most stars, including `express`, `async`, `request`, `grunt`, and `socket.io`.
- Most Prolific Recently**: A list of users who have published the most recently, such as `sindresorhus`, `dbashford`, `jongleberry`, `rvagg`, and `pnidem`.
- TJ Krusinski**: User profile information.



MODULES

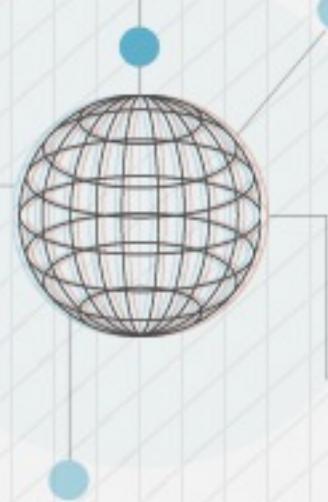




INSTALLING A NPM MODULE

In /Home/my_app

```
$ npm install request
```



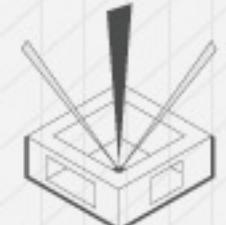
Installs into local node_modules directory

Home / my_app / node_modules / **request**

In /Home/my_app/app.js

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

Loads from local node_modules directory



MODULES





LOCAL VS GLOBAL

Install modules with executables globally

```
$ npm install coffee-script -g
```

global

```
$ coffee app.coffee
```

Global npm modules can't be required

```
var coffee = require('coffee-script');
```



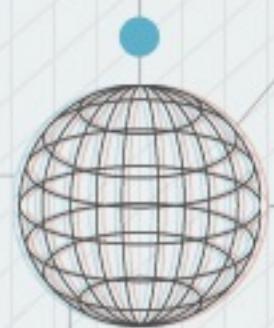
```
$ npm install coffee-script
```

Install them locally

```
var coffee = require('coffee-script');
```



MODULES



FINDING MODULES

npm registry

The screenshot shows the npm homepage. At the top is the npm logo and a search bar labeled "Search Packages". To the right is a user profile for "TJ Krusinski" with options to "Edit Profile" and "Log out". On the left, there's a sidebar with links to "HOME", "API", "BLOG", "NODE.JS", and "JOBS". The main content area is titled "Node Packaged Modules" and displays "Total Packages: 61 828". Below this are sections for "Patches welcome!", "Any package can be installed by using `npm install`.", and "Add your programs to this index by using `npm publish`.". At the bottom are two navigation tabs: "Recently Updated" and "Most Depended Upon".

npm command line

```
$ npm search request
```

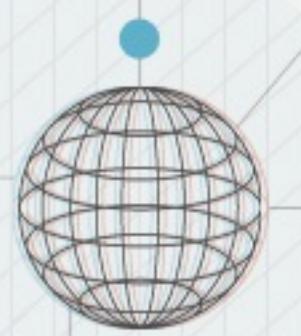
github search

The screenshot shows the GitHub search interface. At the top is a search bar with placeholder text "Search or type a command". Below it is a user profile for "TJkrusinski". The main area has tabs for "News Feed" (which is active) and "Pull Requests". A recent activity item is shown: "mathiasbynens forked passy/ctlmng to mathiasbynens/ctlmng an hour ago".

MODULES



DEFINING YOUR DEPENDENCIES



my_app/package.json

```
{  
  "name": "My App",  
  "version": "1",  
  "dependencies": {  
    "connect": "1.8.7" ← version number  
  }  
}
```

\$ npm install

Installs into the node_modules directory

my_app

/ node_modules

/ connect



MODULES

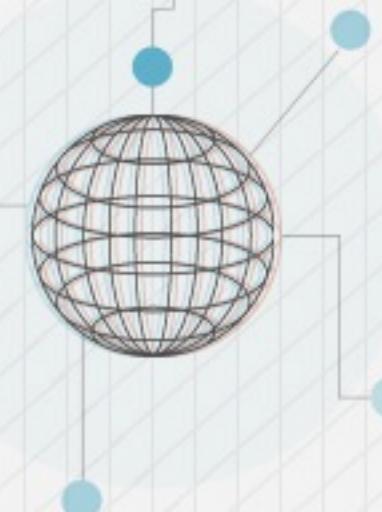




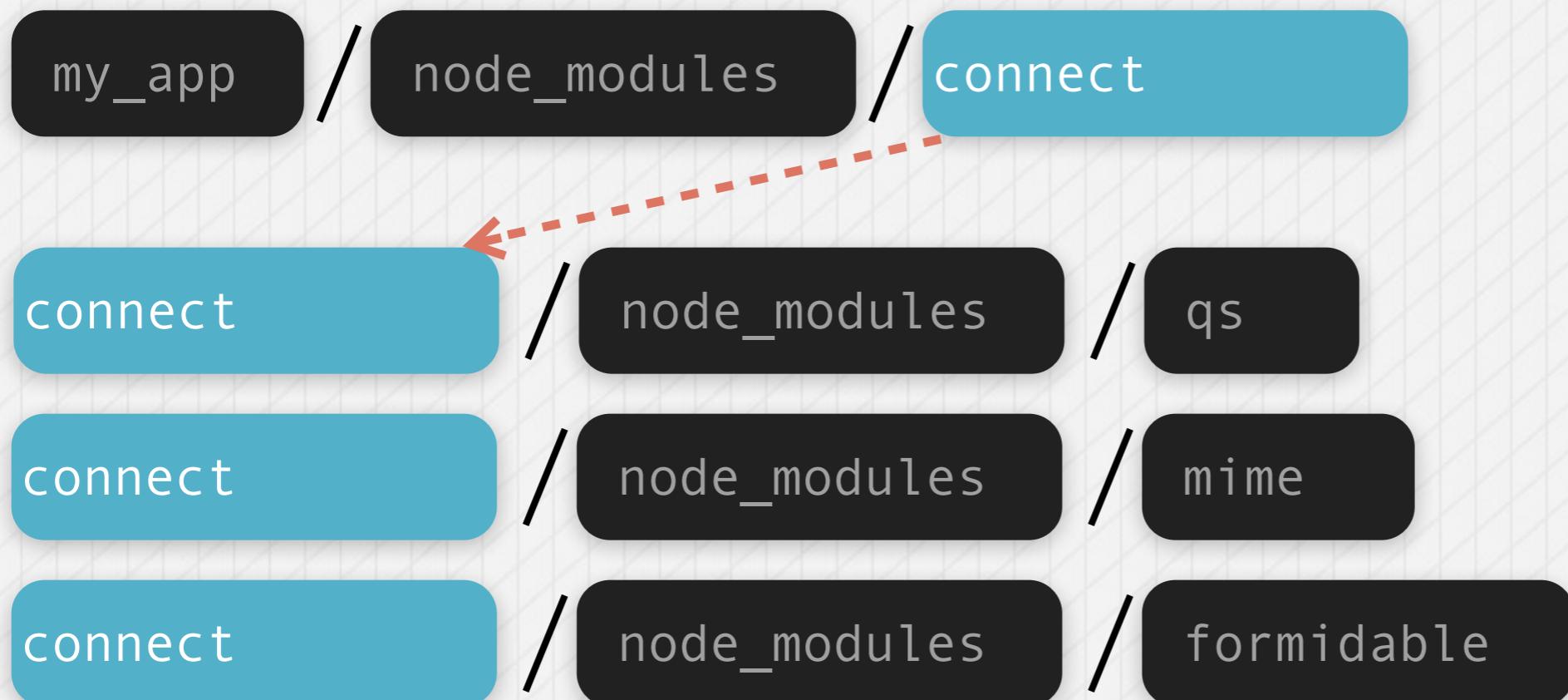
DEPENDENCIES

my_app/package.json

```
"dependencies": {  
  "connect": "1.8.7"  
}
```



Installs sub-dependencies

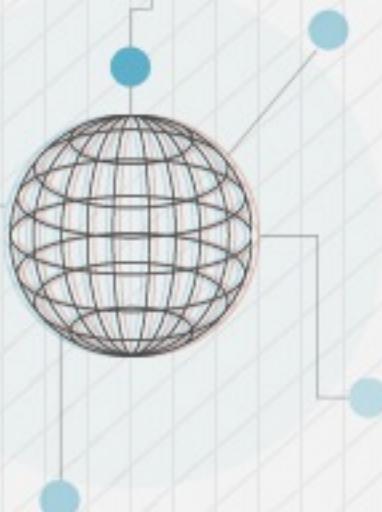


MODULES





SEMANTIC VERSIONING



```
"connect": "1.8.7"
```

Major	Minor	Patch
1	.	8

Ranges

```
"connect": "~1"
```

- ->

```
>=1.0.0 <2.0.0
```

Dangerous

```
"connect": "~1.8"
```

- ->

```
>=1.8.0 <1.9.0
```

API could change

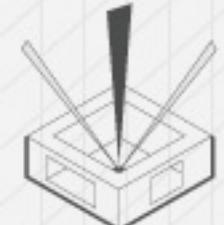
```
"connect": "~1.8.7"
```

- ->

```
>=1.8.7 <1.9.0
```

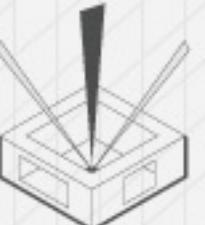
Considered safe

<http://semver.org/>



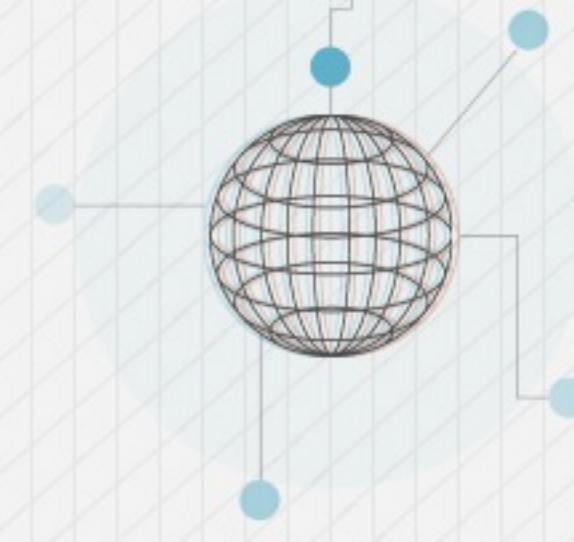
MODULES





EXPRESS

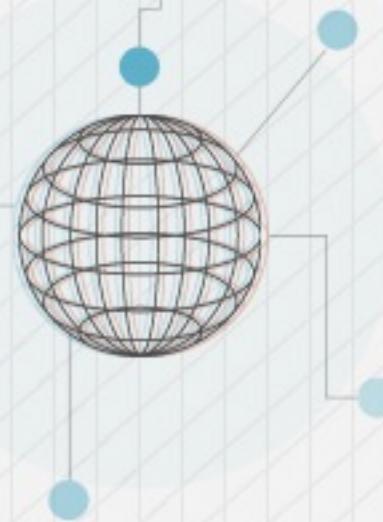
- LEVEL FIVE -



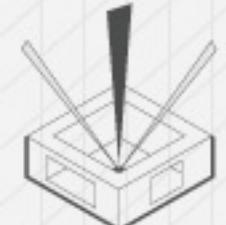


EXPRESS

“Sinatra inspired web development framework for Node.js -- insanely fast, flexible, and simple”



- Easy route URLs to callbacks
- Middleware (from Connect)
- Environment based configuration
- Redirection helpers
- File Uploads



EXPRESS





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

```
$ npm install --save express
```

```
var app = express();
```

Installs the module and adds to package.json

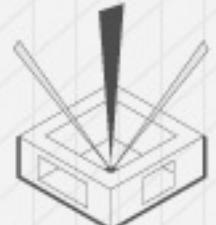
```
app.get('/', function(request, response) {  
  response.sendFile(__dirname + "/index.html");  
});
```

root route

current directory

```
app.listen(8080);
```

```
$ curl http://localhost:8080/  
> 200 OK
```



EXPRESS



INTRODUCING EXPRESS

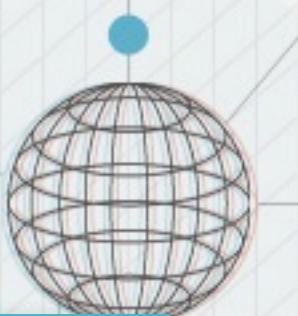


Hello World!

EXPRESS



EXPRESS ROUTES



app.js

```
var request = require('request');
var url = require('url');

app.get('/tweets/:username', function(req, response) {
  var username = req.params.username;

  options = {
    protocol: "http:",           get the last 10 tweets for screen_name
    host: 'api.twitter.com',
    pathname: '/1/statuses/user_timeline.json',
    query: { screen_name: username, count: 10}
  }

  var twitterUrl = url.format(options);
  request(twitterUrl).pipe(response);   pipe the request to response
});
```



EXPRESS

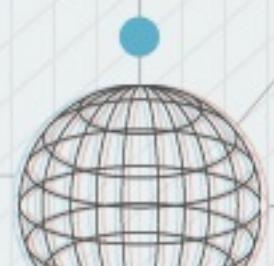
EXPRESS ROUTES



Node.js

level4 — Node.js

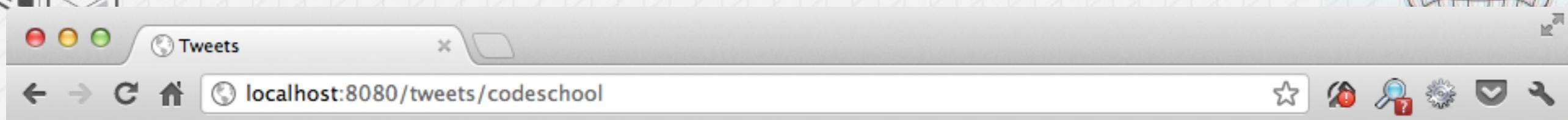
\$



EXPRESS



EXPRESS + HTML



Tweets for @codeschool

- @seandevineinc let us know how it goes, and good luck
- @larzconwell Nope, we didn't give those away. The one David has comes from <http://t.co/XrvybxnS> ^OL
- We just released a new Code TV screencast for enrolled members. Part of 1 of @markkendall's jQuery Mobile series. <http://t.co/FstmuYEM>
- ^vc
- We also have stickers..
- Are you at Railsconf? Come by the beginner track room.. we're giving away free Rails for Zombies T-shirts (while they last) #railsconf
- We themed out our Code School store. Check it out <http://t.co/VOZCgorM> ^vc
- Have you gotten your Code School & Zombies t-shirts yet? Check out our \$19 sale this week... <http://t.co/b7JUMfxy> ^vc



my_app/package.json

```
"dependencies": {  
  "express": "4.9.6",  
  "ejs": "1.0.0"  
}
```

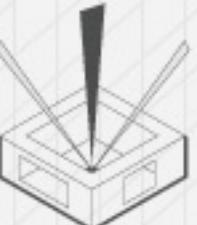
```
$ npm install --save ejs
```



Installs the module and adds to package.json

/Home/eric/my_app/views

 *default directory*

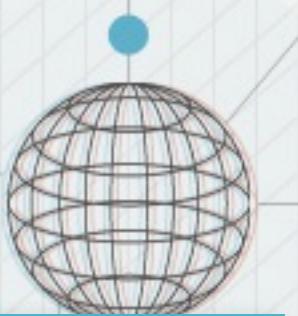


EXPRESS





EXPRESS TEMPLATES

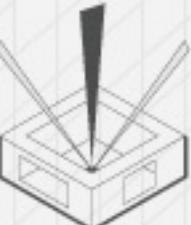


```
app.get('/tweets/:username', function(req, response) {  
  ...  
  request(url, function(err, res, body) {  
    var tweets = JSON.parse(body);  
    response.locals = {tweets: tweets, name: username};  
    response.render('tweets.ejs');  
  });  
});
```

app.js

```
<h1>Tweets for @<%= name %></h1>  
<ul>  
  <% tweets.forEach(function(tweet){ %>  
    <li><%= tweet.text %></li>  
  <% }); %>  
</ul>
```

views/tweets.ejs

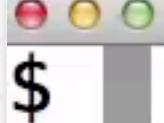
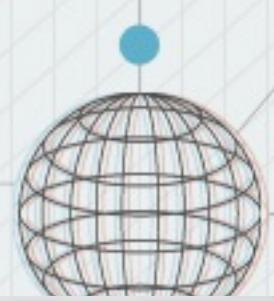


EXPRESS



EXPRESS TEMPLATES

level4 — Node.js



\$



EXPRESS





TEMPLATE LAYOUTS

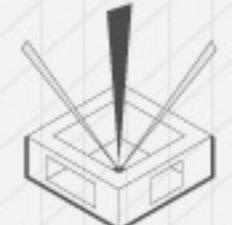
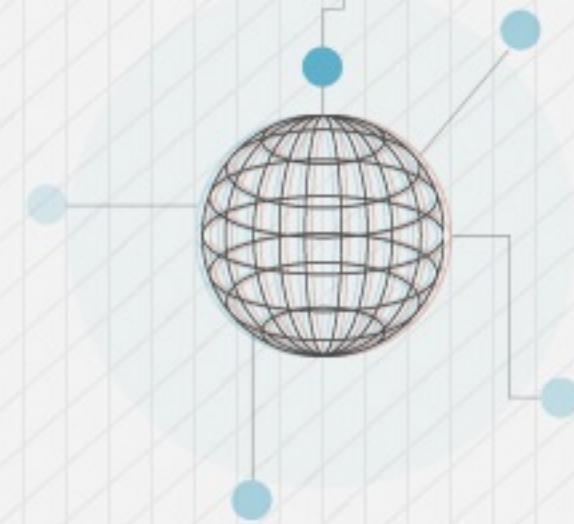


EXPRESS



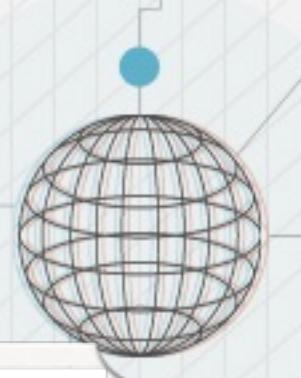
SOCKET.IO

- LEVEL SIX -





CHATTR



Hello from Chattr

ERIC

DERRICK

CONNECTED TO CHATTR

Eric joined the room

Derrick

Hey buddy!

Eric

I'm having a great time over here?

Derrick joined the room

Type your message

SEND



SOCKET.IO



WEBSOCKETS



Traditional request/response cycle

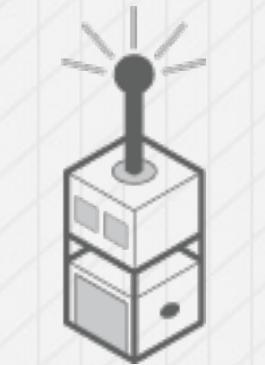
SOCKET.IO



traditional server



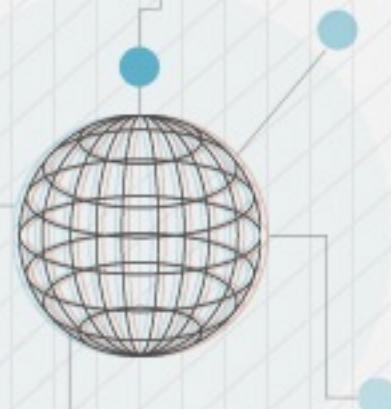
WEBSOCKETS



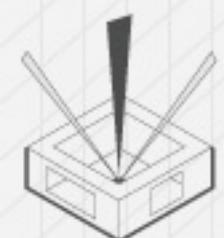
browser



socket.io



SOCKET.IO





SOCKET.IO FOR WEB SOCKETS

Abstracts websockets with fallbacks

```
$ npm install --save socket.io
```

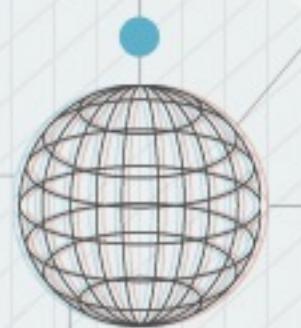
```
var express = require('express');
var app = express();
var server = require('http').createServer(app);
var io = require('socket.io')(server);

io.on('connection', function(client) {
  console.log('Client connected...');
});

app.get('/', function (req, res) {
  res.sendFile(__dirname + '/index.html');
});

server.listen(8080);
```

app.js



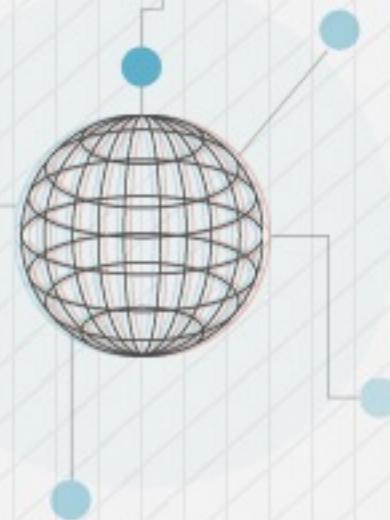
SOCKET.IO





SOCKET.IO FOR WEB SOCKETS

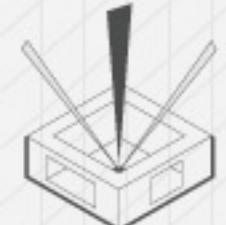
socket.io client connects to the server



```
<script src="/socket.io/socket.io.js"></script>
```

index.html

```
<script>
  var socket = io.connect('http://localhost:8080');
</script>
```



SOCKET.IO





SENDING MESSAGES TO CLIENT



```
io.on('connection', function(client) {  
  console.log('Client connected...');  
  
  emit the 'messages' event on the client  
  client.emit('messages', { hello: 'world' });  
});
```

app.js

```
<script src="/socket.io/socket.io.js"></script>  
<script>  
  var socket = io.connect('http://localhost:8080');  
  socket.on('messages', function (data) {  
    alert(data.hello);  
  });  
</script>
```

index.html

listen for 'messages' events



SOCKET.IO



CHATTR HELLO WORLD

demo — dash

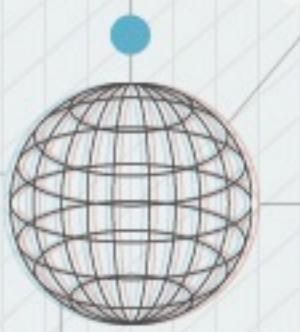
\$

I

SOCKET.IO



SENDING MESSAGES TO SERVER



```
io.on('connection', function(client) {  
  client.on('messages', function (data) {  
    console.log(data);  
  });  
});  
listen for 'messages' events
```

app.js

```
<script>  
  var socket = io.connect('http://localhost:8080');  
  
  $('#chat_form').submit(function(e){  
    var message = $('#chat_input').val();  
    emit the 'messages' event on the server  
    socket.emit('messages', message);  
  });  
</script>
```

index.html

CHATTR HELLO WORLD

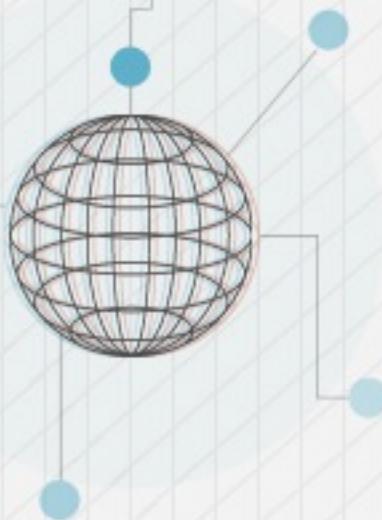
level4 — bash

\$

I

SOCKET.IO

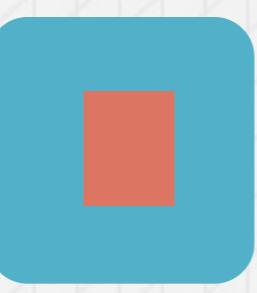
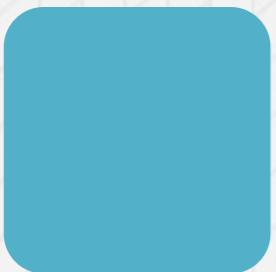
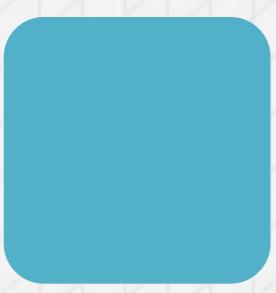
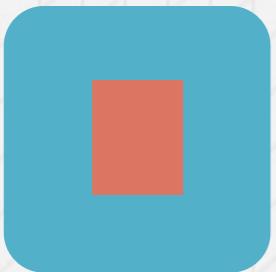
BROADCASTING MESSAGES



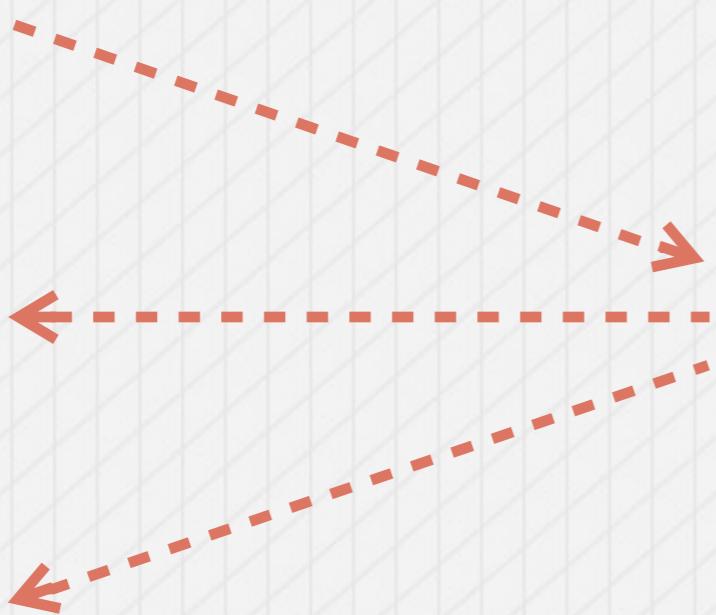
app.js

```
socket.broadcast.emit("message", 'Hello');
```

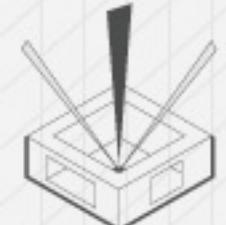
clients



server

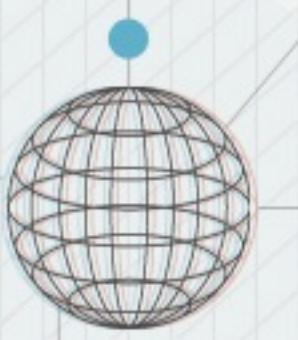


SOCKET.IO





BROADCASTING MESSAGES



```
io.on('connection', function(client) {  
  client.on('messages', function (data) {  
    client.broadcast.emit("messages", data);  
  });  
});  
broadcast message to all other clients connected  
});
```

app.js

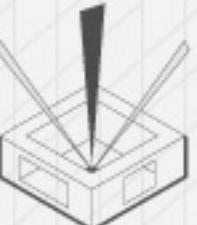
```
<script>
```

```
...
```

```
socket.on('messages', function(data) { insertMessage(data) });  
</script>
```

index.html

insert message into the chat



SOCKET.IO



BROADCASTING MESSAGES

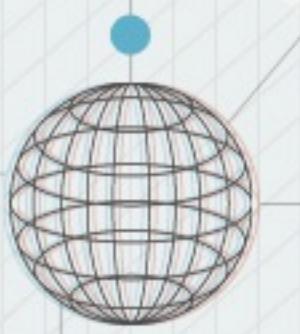
level4 — bash

\$

SOCKET.IO



SAVING DATA ON THE SOCKET

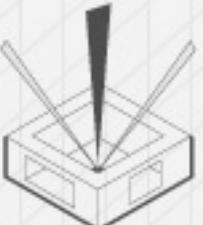


```
io.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.nickname = name;      set the nickname associated  
    });                                with this client  
  });  
});
```

app.js

```
<script>  
  var server = io.connect('http://localhost:8080');  
  
  server.on('connect', function(data) {  
    $('#status').html('Connected to chattr');  
    nickname = prompt("What is your nickname?");  
  
    server.emit('join', nickname);      notify the server of the  
    });                                users nickname  
  </script>
```

index.html

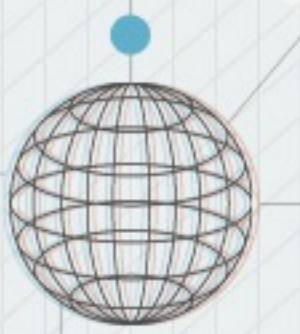


SOCKET.IO





SAVING DATA ON THE CLIENT



app.js

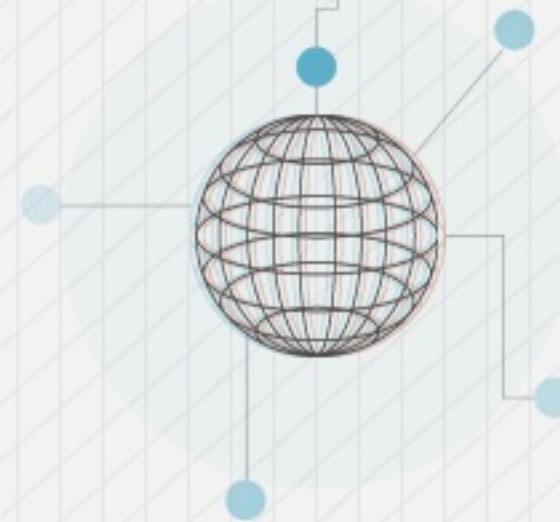
```
io.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.nickname = name;      set the nickname associated  
  });                                with this client  
  client.on('messages', function(data){  
    var nickname = client.nickname;    get the nickname of this client  
    client.broadcast.emit("message", nickname + ": " + message);  
                                         before broadcasting message  
    client.emit("messages", nickname + ": " + message);  
                                         broadcast with the name and message  
  });  
});
```



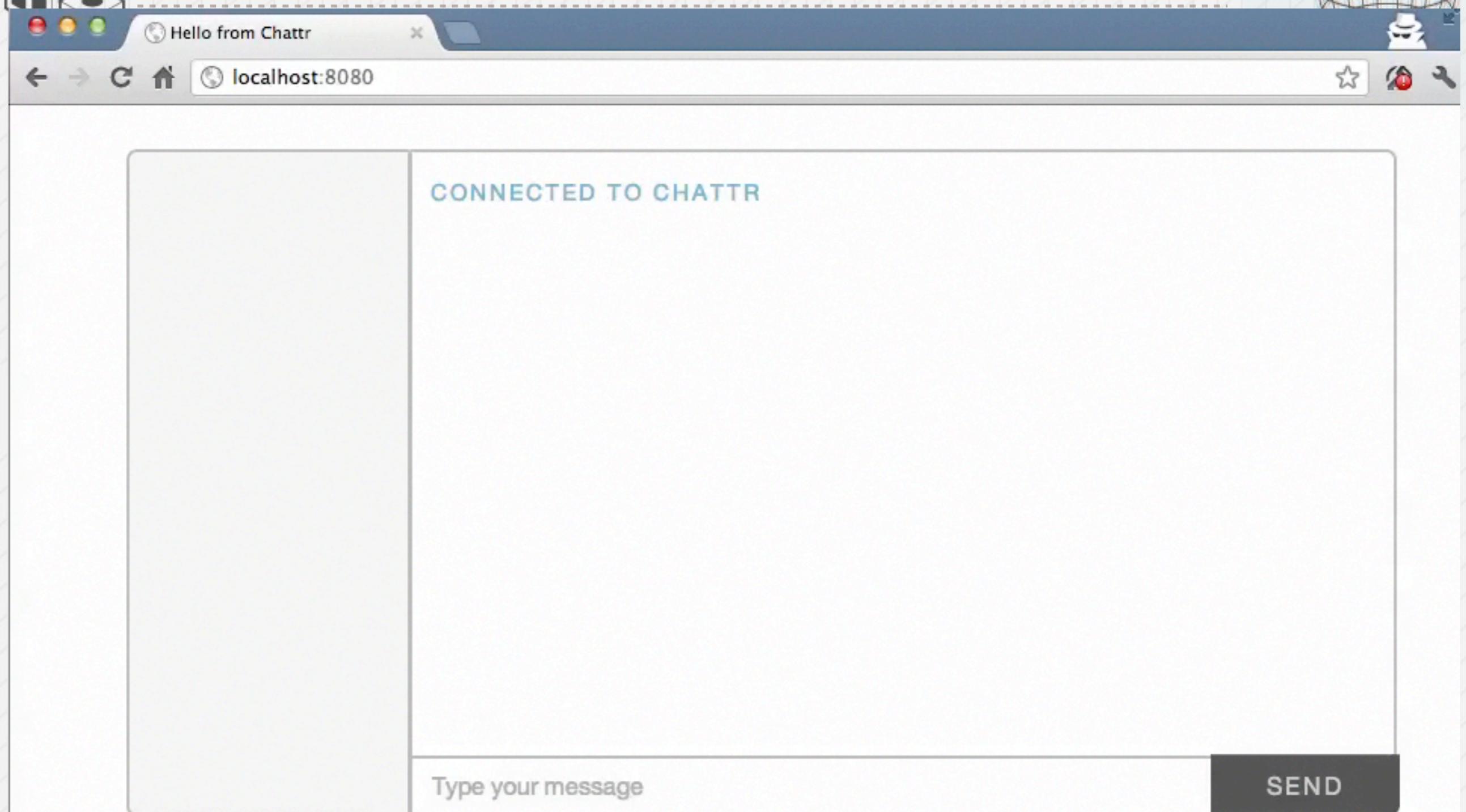
SOCKET.IO

PERSISTING DATA

- LEVEL SEVEN -



RECENT MESSAGES



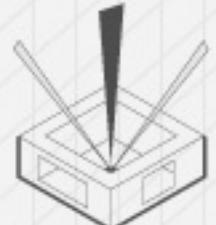
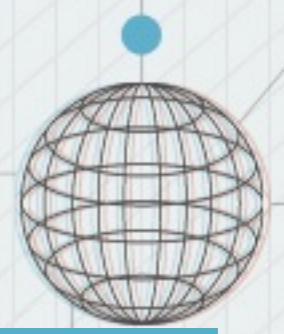
PERSISTING DATA



RECENT MESSAGES

app.js

```
io.sockets.on('connection', function(client) {  
  client.on('join', function(name) {  
    client.nickname = name;  
    client.broadcast.emit("chat", name + " joined the chat");  
  });  
  client.on("messages", function(message){  
    client.broadcast.emit("messages", client.nickname +  
      ": " + message);  
    client.emit("messages", client.nickname +  
      ": " + message);  
  });  
});
```

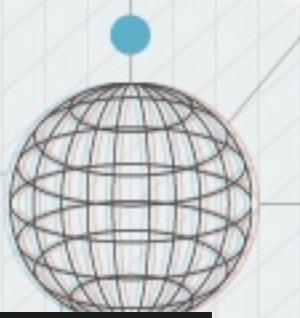


PERSISTING DATA





STORING MESSAGES



```
var messages = [];
```

store messages in array

app.js

```
var storeMessage = function(name, data){
```

messages.push({name: name, data: data}); add message to end of array

```
  if (messages.length > 10) {
```

if more than 10 messages long,
remove the first one

```
    messages.shift();
```

```
}
```

```
}
```

```
io.sockets.on('connection', function(client) {
```

```
  client.on("messages", function(message){
```

```
    client.broadcast.emit("messages", client.nickname +  
      ": " + message);
```

```
    client.emit("messages", client.nickname + ": " + message);
```

```
    storeMessage(client.nickname, message);
```

```
  });
```

```
});
```

*when client sends a message
call storeMessage*

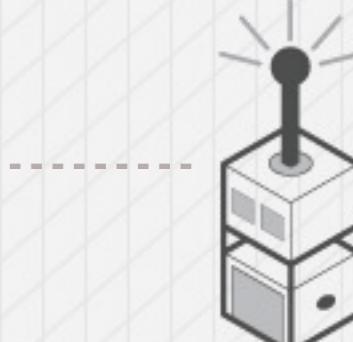




EMITTING MESSAGES

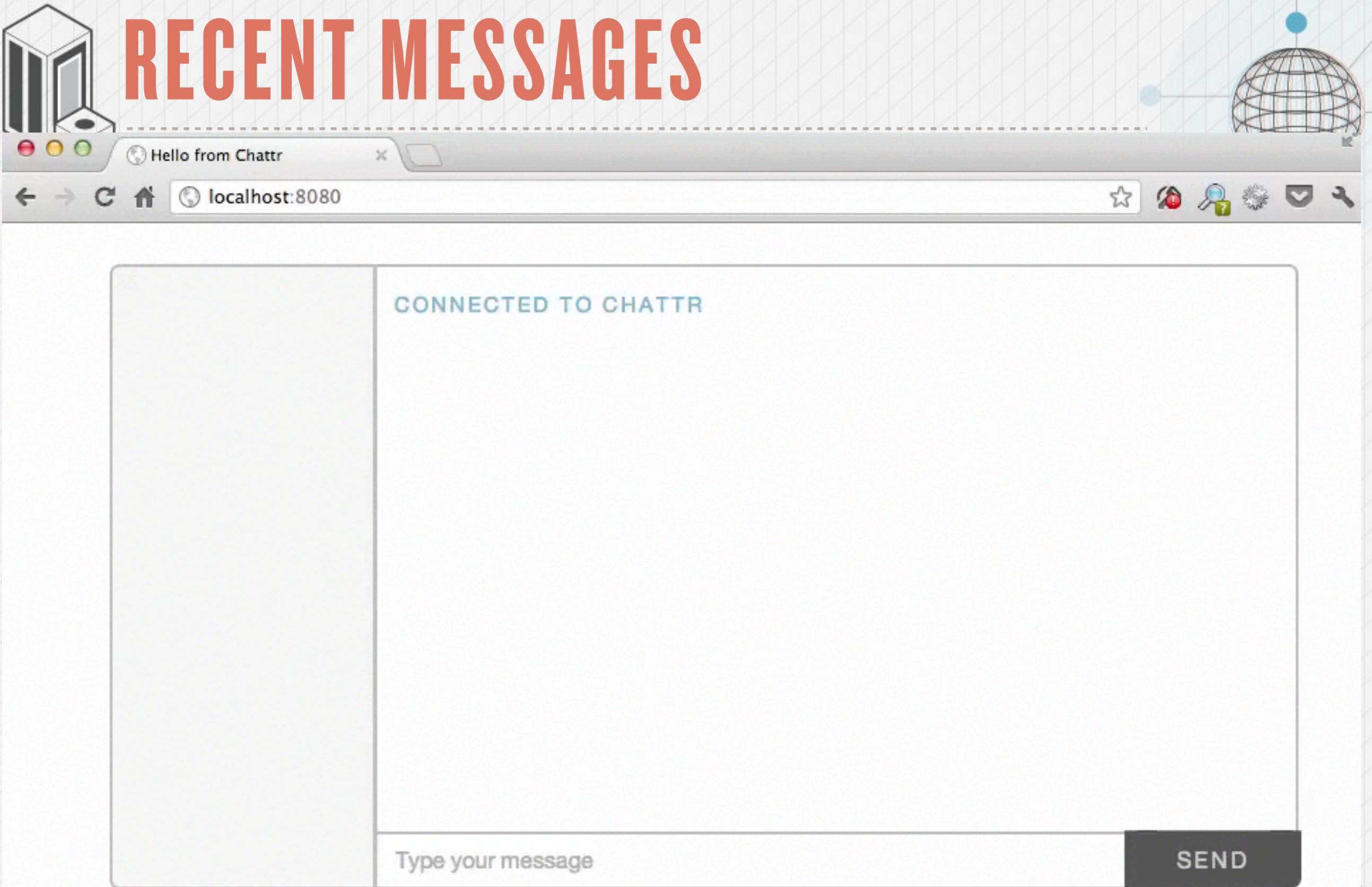
app.js

```
io.sockets.on('connection', function(client) {  
  ...  
  client.on('join', function(name) {  
    messages.forEach(function(message) {  
      client.emit("messages", message.name + ": " + message.data);  
    });    iterate through messages array  
  });  and emit a message on the connecting  
});  client for each one
```



PERSISTING DATA

RECENT MESSAGES



PERSISTING DATA



PERSISTING STORES

- MongoDB
- CouchDB
- PostgreSQL
- Memcached
- Riak



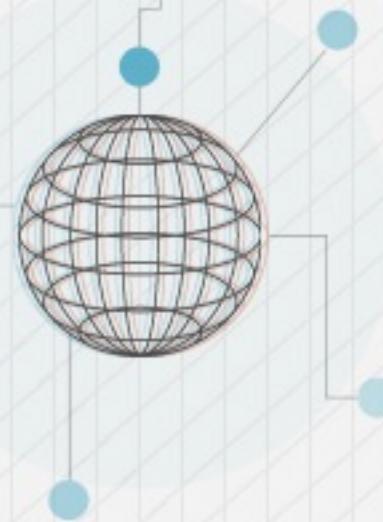
redis

Redis is a key-value store



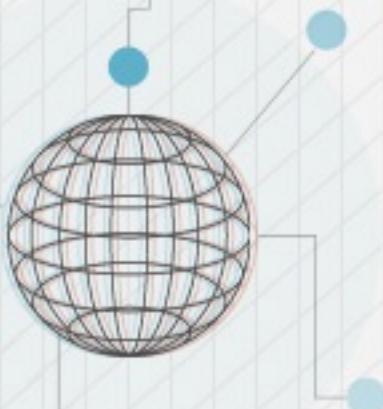
PERSISTING DATA

All non-blocking!





REDIS DATA STRUCTURES



data structure

commands

Strings

SET, GET, APPEND, DECR, INCR...

Hashes

HSET, HGET, HDEL, HGETALL...

Lists

LPUSH, LREM, LTRIM, RPOP, LINSERT...

Sets

SADD, SREM, SMOVE, SMEMBERS...

Sorted Sets

ZADD, ZREM, ZSCORE,ZRANK...



PERSISTING DATA



REDIS COMMAND DOCUMENTATION

A screenshot of a web browser displaying the Redis website. The title bar shows 'Redis' and the address bar shows 'redis.io'. The page features a navigation menu with links to 'Commands', 'Clients', 'Documentation', 'Community', 'Download', and 'Issues'. A sidebar on the left contains sections for 'Try it' and 'PERSISTING DATA'.

Redis is an open source, advanced **key-value store**. It is often referred to as a **data structure server** since keys can contain strings, hashes, lists, sets and sorted sets.

[Learn more →](#)

Try it

Ready for a test drive? Check this [interactive tutorial](#) that will walk you through the most important features of Redis.

Download it

[Redis 2.4.13 is the latest stable version.](#) Interested in release candidates or unstable versions? [Check the downloads page.](#)

What people are saying



Facebook Sets I.P.O.
Price Range
<http://t.co/7qTOhWMx>



@tinkertim No more spaces screwing my Redis commands.
Pretty major to me ;-)



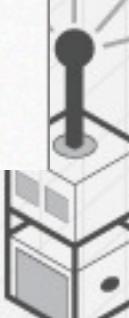
#RedMango #coupon?
Get a \$2 OFF one
@coupons.com. Just enter your ZIP code in the upper left! US only.
<http://t.co/P0i9nvUh>



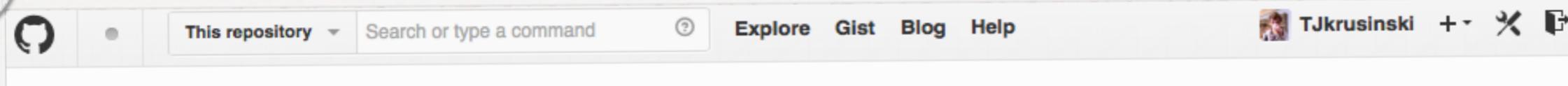
redis (@DIRTYBIITCH)



PERSISTING DATA



NODE REDIS



This repository Search or type a command Explore Gist Blog Help TJkrusinski + X ⌂



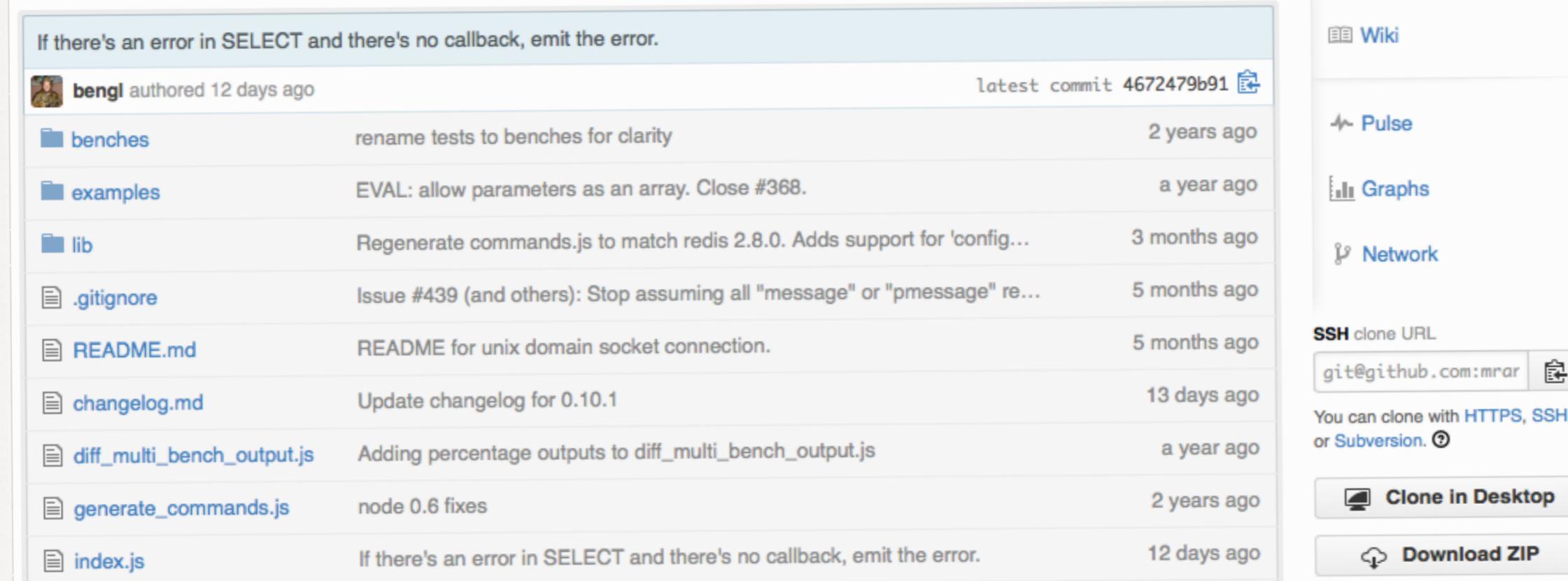
mranney / node_redis Watch 176 Star 3,102 Fork 422

redis client for node



509 commits 3 branches 36 releases 63 contributors

branch: master node_redis / +



If there's an error in SELECT and there's no callback, emit the error.

Author	Commit Message	Date
bengi	authored 12 days ago	latest commit 4672479b91
	rename tests to benches for clarity	2 years ago
	EVAL: allow parameters as an array. Close #368.	a year ago
	Regenerate commands.js to match redis 2.8.0. Adds support for 'config...	3 months ago
	Issue #439 (and others): Stop assuming all "message" or "pmessage" re...	5 months ago
	README for unix domain socket connection.	5 months ago
	Update changelog for 0.10.1	13 days ago
	Adding percentage outputs to diff_multi_bench_output.js	a year ago
	node 0.6 fixes	2 years ago
	If there's an error in SELECT and there's no callback, emit the error.	12 days ago

SSH clone URL
git@github.com:mranney ⌂
You can clone with HTTPS, SSH, or Subversion. ?

Clone in Desktop Download ZIP

PERSISTING DATA



REDIS

```
$ npm install redis --save
```

```
var redis = require('redis');
var client = redis.createClient();

client.set("message1", "hello, yes this is dog");
client.set("message2", "hello, no this is spider");
```

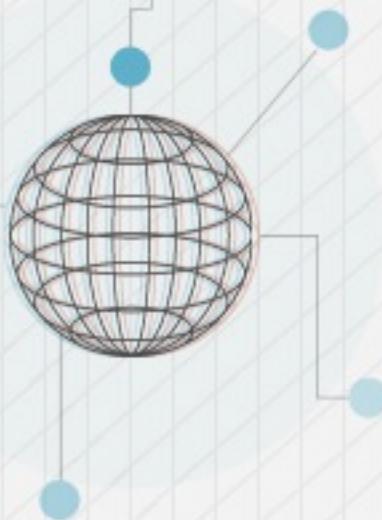
key

value

```
client.get("message1", function(err, reply){
  console.log(reply); -----> "hello, yes this is dog"
});
```

commands are non-blocking

PERSISTING DATA





REDIS LISTS: PUSHING

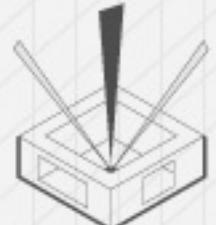
Add a string to the “messages” list

```
var message = "Hello, this is dog";
client.lpush("messages", message, function(err, reply){
  console.log(reply); - - -> "1"  replies with list length
});
```



Add another string to “messages”

```
var message = "Hello, no this is spider";
client.lpush("messages", message, function(err, reply){
  console.log(reply); - - -> "2"
});
```



PERSISTING DATA



REDIS LISTS: RETRIEVING

Using LPUSH & LTRIM

```
var message = "Hello, this is dog";
client.lpush("messages", message, function(err, reply){
  client.ltrim("messages", 0, 1);    trim keeps first two strings
});                                and removes the rest
```

Retrieving from list

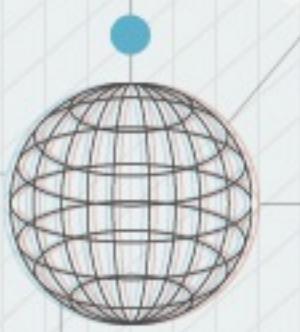
```
client.lrange("messages", 0, -1, function(err, messages){
  console.log(messages);    replies with all strings in list
})
```



```
["Hello, no this is spider", "Oh sorry, wrong number"]
```



PERSISTING DATA

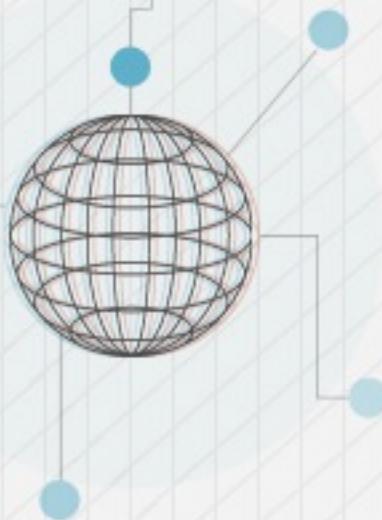




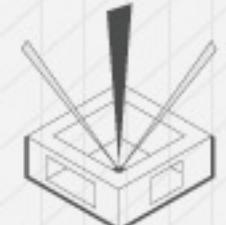
CONVERTING MESSAGES TO REDIS

```
var storeMessage = function(name, data){  
  messages.push({name: name, data: data});  
  
  if (messages.length > 10) {  
    messages.shift();  
  }  
}
```

app.js



Let's use the List data-structure

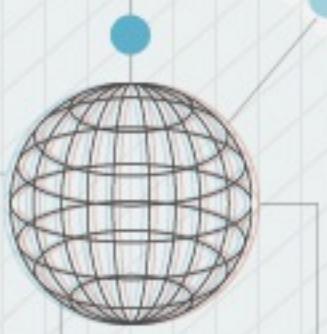


PERSISTING DATA





CONVERTING STOREMESSAGE



```
var redisClient = redis.createClient();
```

app.js

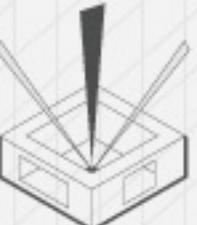
```
var storeMessage = function(name, data){  
  var message = JSON.stringify({name: name, data: data});
```

need to turn object into string to store in redis

```
redisClient.lpush("messages", message, function(err, response) {  
  redisClient.ltrim("messages", 0, 9);  
});  
}
```



keeps newest 10 items

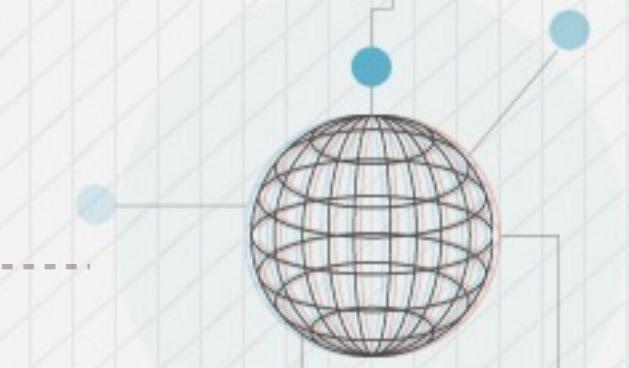


PERSISTING DATA



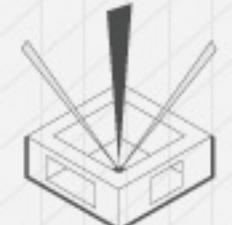


OUTPUT FROM LIST



```
client.on('join', function(name) {  
  messages.forEach(function(message) {  
    client.emit("messages", message.name + ": " + message.data);  
  });  
});
```

app.js

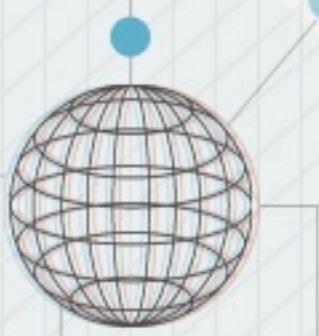


PERSISTING DATA



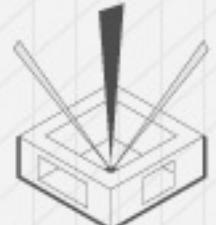


OUTPUT FROM LIST



app.js

```
client.on('join', function(name) {  
  redisClient.lrange("messages", 0, -1, function(err, messages){  
    messages = messages.reverse();           reverse so they are emitted  
    in correct order  
    messages.forEach(function(message) {  
      message = JSON.parse(message);       parse into JSON object  
      client.emit("messages", message.name + ":" + message.data);  
    });  
  });  
});
```

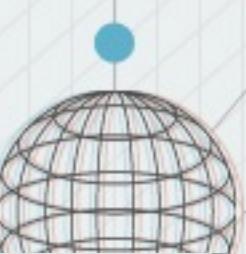


PERSISTING DATA





IN ACTION



A screenshot of a web browser window titled "Hello from Chatr". The address bar shows "localhost:8080". The main content area displays the text "CONNECTED TO CHATTR". At the bottom, there is a input field with the placeholder "Type your message" and a "SEND" button.

Connected to Chatr

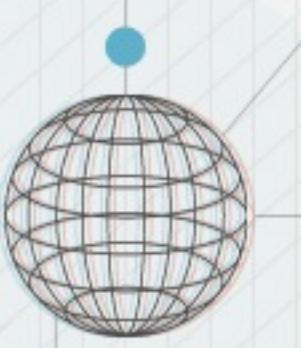
Type your message **SEND**

PERSISTING DATA





CURRENT CHATTER LIST



Sets are lists of unique data

DOG
SPIDER
GREGG

add & remove members of the names set

```
client.sadd("names", "Dog");
client.sadd("names", "Spider");
client.sadd("names", "Gregg");
```

```
client.srem("names", "Spider");
```

reply with all members of set

```
client.smembers("names", function(err, names){
  console.log(names);
});
```



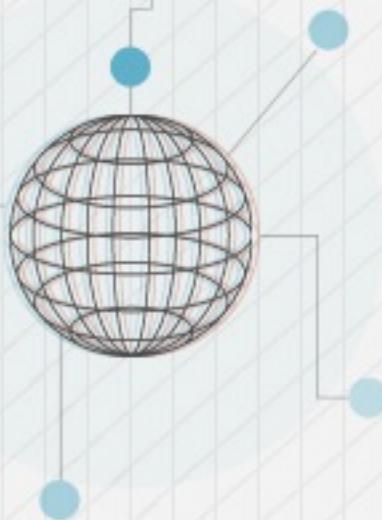
```
["Dog", "Gregg"]
```

PERSISTING DATA





ADDING CHATTERS



```
client.on('join', function(name){
```

app.js

notify other clients a chatter has joined

```
    client.broadcast.emit("add chatter", name);
```

```
    redisClient.sadd("chatters", name);
```

```
});  
    add name to chatters set
```

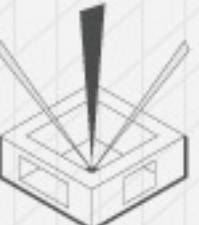
```
socket.on('add chatter', function(name) {
```

index.html

```
    var chatter = $('<li>' + name + '</li>').data('name', name);
```

```
    $('#chatters').append(chatter);
```

```
});
```

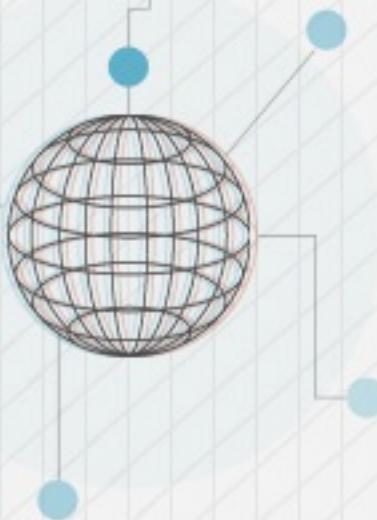


PERSISTING DATA



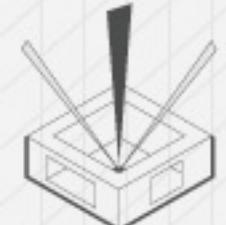


ADDING CHATTERS (CONT)



```
client.on('join', function(name){  
  client.broadcast.emit("add chatter", name);  
  
  redisClient.smembers('names', function(err, names) {  
    names.forEach(function(name){  
      client.emit('add chatter', name);  
    });  
  });  
  emit all the currently logged in chatters  
  to the newly connected client  
  redisClient.sadd("chatters", name);  
});
```

app.js



PERSISTING DATA





REMOVING CHATTERS

remove chatter when they disconnect from server

```
client.on('disconnect', function(name){  
    client.broadcast.emit("remove chatter", client.nickname);  
    redisClient.srem("chatters", client.nickname);  
});
```

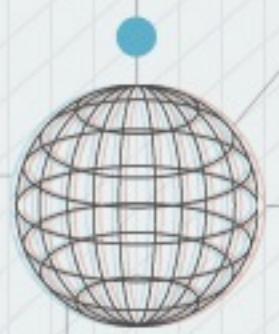
app.js

```
server.on('remove chatter', function(name) {  
    $('#chatters li[data-name=' + name + ']').remove();  
});
```

index.html

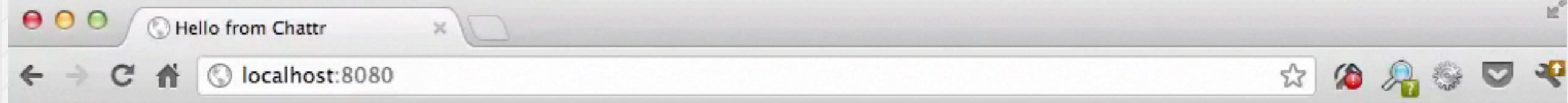


PERSISTING DATA





WELCOME TO CHATTR



Hello from Chattr

DOG

CONNECTED TO CHATTR

PERSISTING DATA