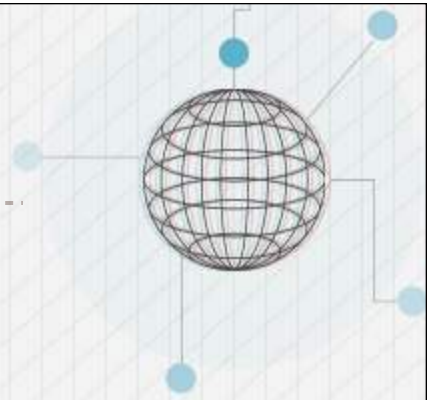
WHAT IS NODE.JS?



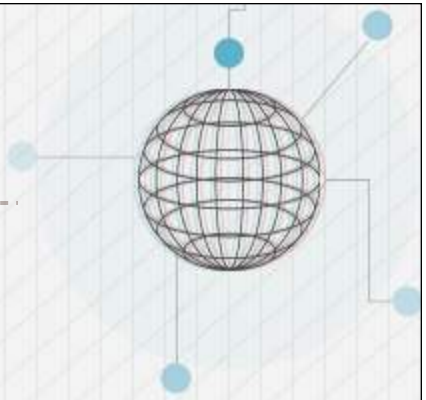
Allows you to build scalable network applications using JavaScript on the server-side.

Node.js

V8 JavaScript Runtime

It's fast because it's mostly C code

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

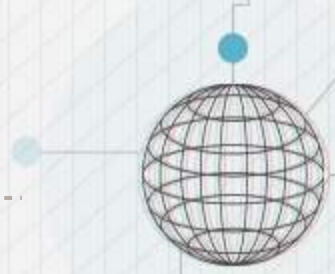
```
Read file from Filesystem, set equal to “contents”
Print contents
Do something else
```

- 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



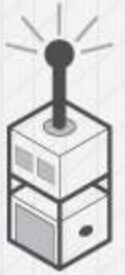
- Blocking Code

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

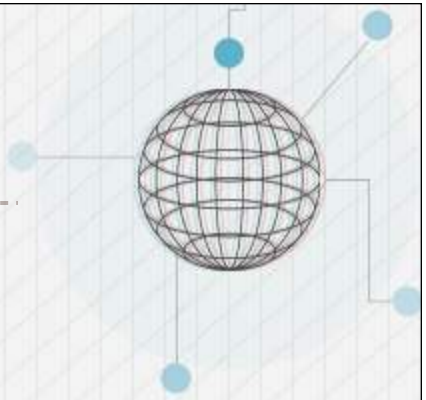
Stop process until complete

- Non-Blocking Code

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



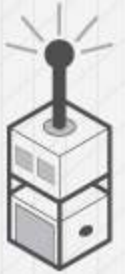
CALLBACK ALTERNATE SYNTAX



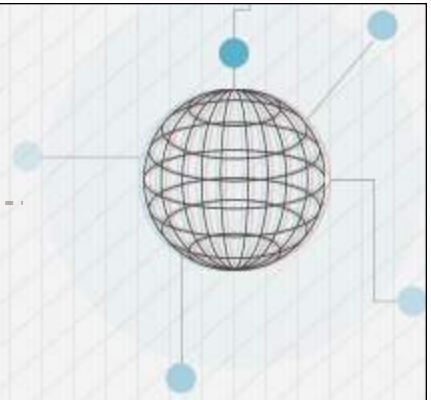
```
fs.readFile('/etc/hosts', function(err, contents) {
  console.log(contents);
});
```

Same as

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

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

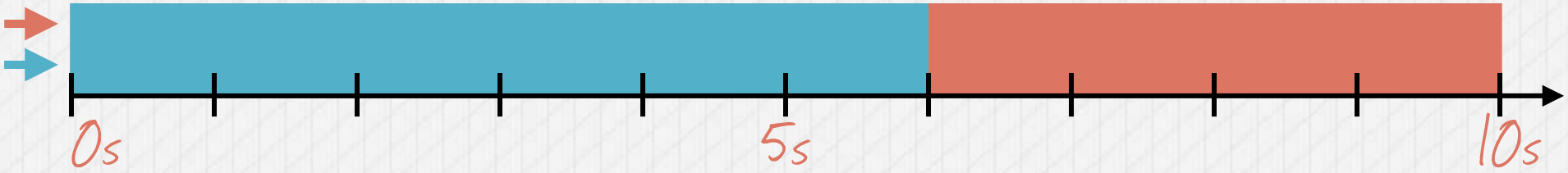


BLOCKING VS NON-BLOCKING

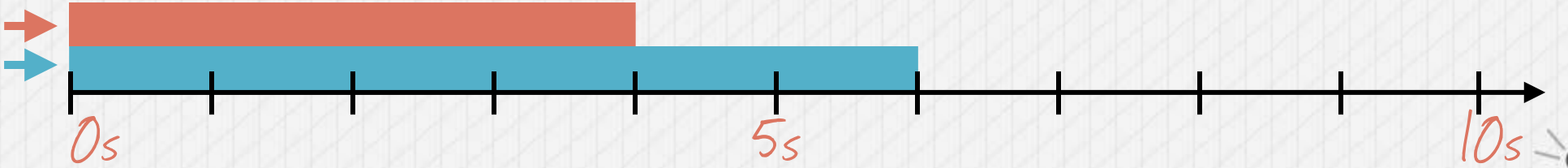


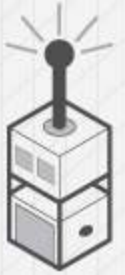
```
var callback = function(err, contents) {
  console.log(contents);
}

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

blocking

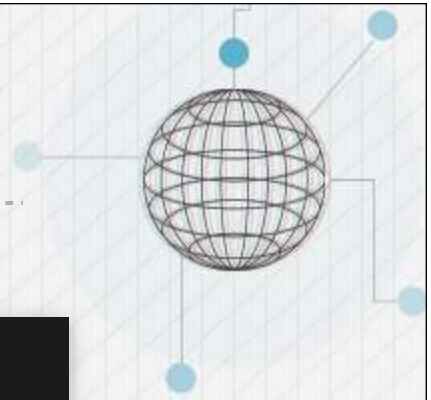


non-blocking





NODE.JS HELLO DOG



hello.js

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

http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Hello, this is dog.");
  response.end();
}).listen(8080);
console.log('Listening on port 8080...');
```

How we require modules

Status code in header

Response body

Close the connection

Listen for connections on this port

```
$ node hello.js
```

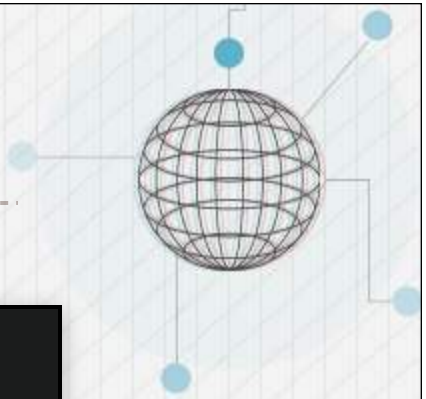
Run the server

----> 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);
console.log('Listening on port 8080...');
```

Starts the Event Loop when finished

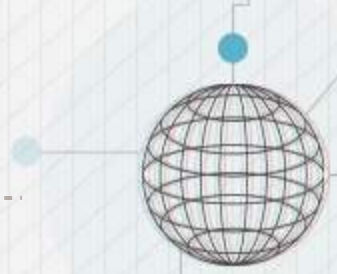
Run the Callback

Known Events

request

Checking for Events

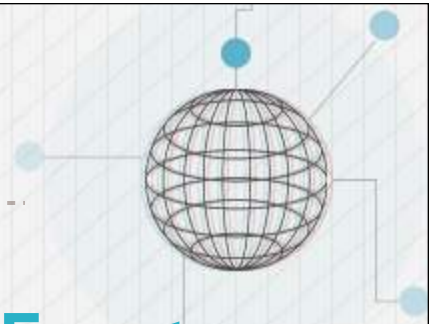
WHY JAVASCRIPT?



"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

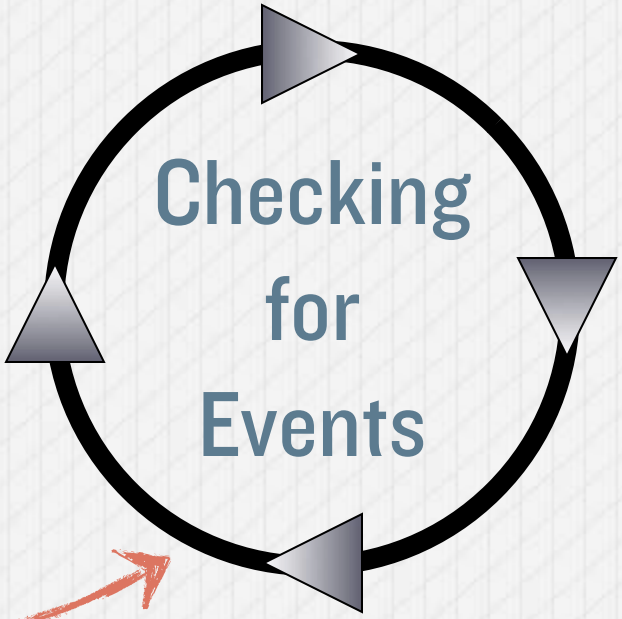
THE EVENT LOOP



Event Queue

close

request



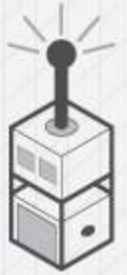
Known Events

request

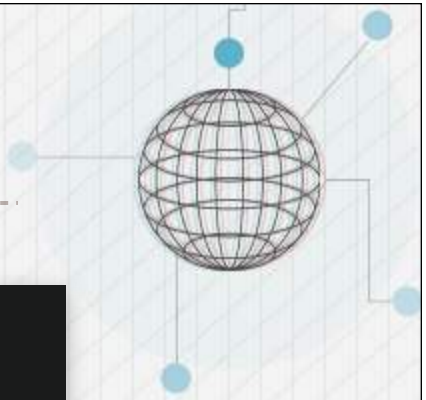
connection

close

Events processed one at a time

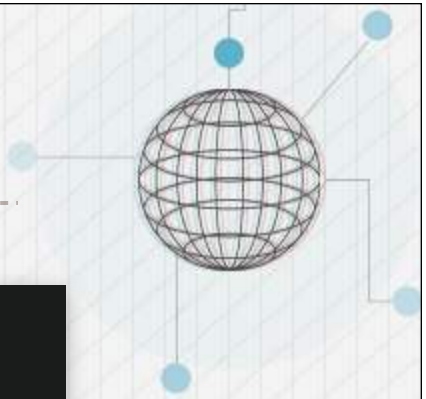


WITH LONG RUNNING PROCESS



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

request

timeout

TWO CALLBACKS TIMELINE

Request comes in, triggers request event

Request Callback executes

setTimeout registered

Request comes in, triggers request event

Request Callback executes

setTimeout registered

triggers setTimeout event

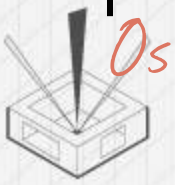
setTimeout Callback executes

triggers setTimeout event

setTimeout Callback

request

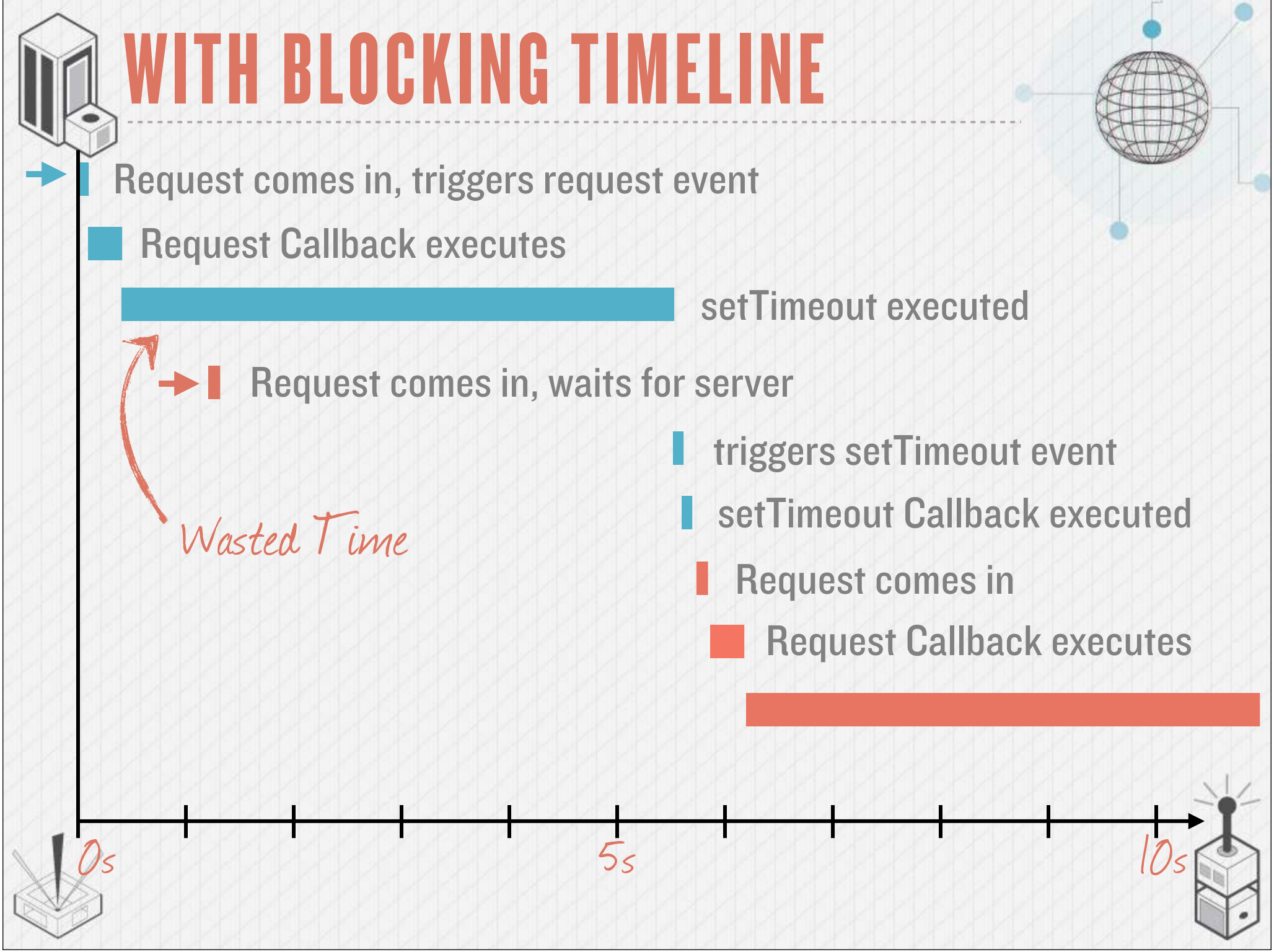
timeout



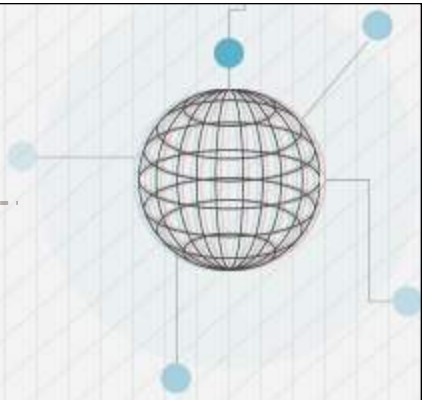
0s

5s

10s


WITH BLOCKING TIMELINE
Request comes in, triggers request event
Request Callback executes
setTimeout executed
Request comes in, waits for server
triggers setTimeout event
setTimeout Callback executed
Request comes in
Request Callback executes
Wasted Time
0s
5s
10s

TYPICAL BLOCKING THINGS

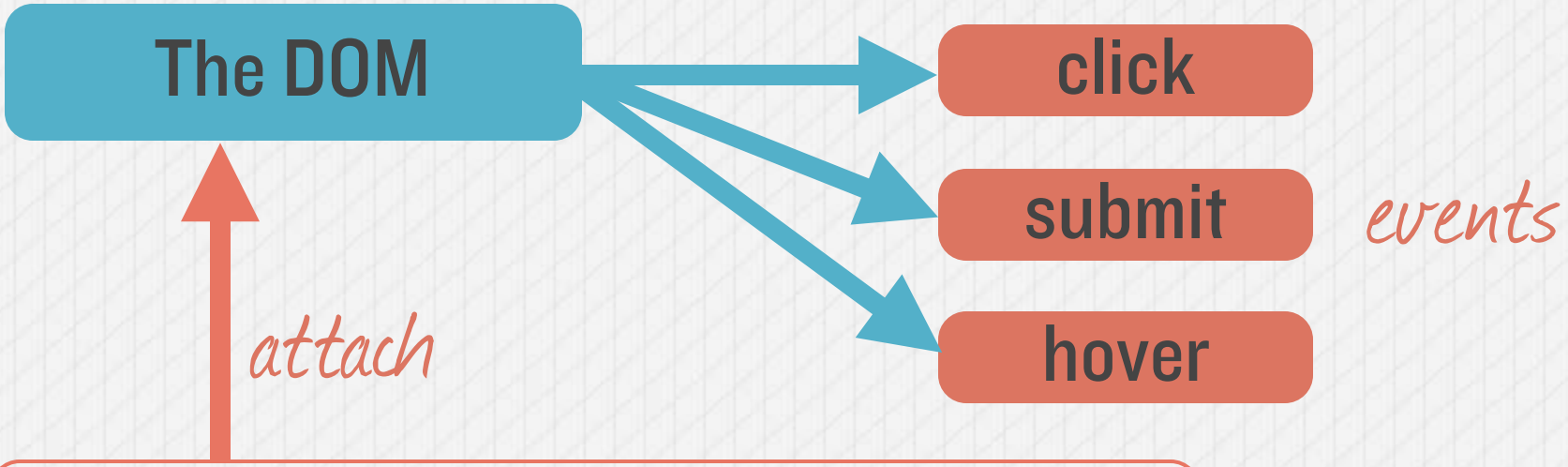


- Calls out to web services
- Reads/Writes on the Database
- Calls to extensions

EVENTS IN THE DOM

The DOM triggers Events

you can listen for those events

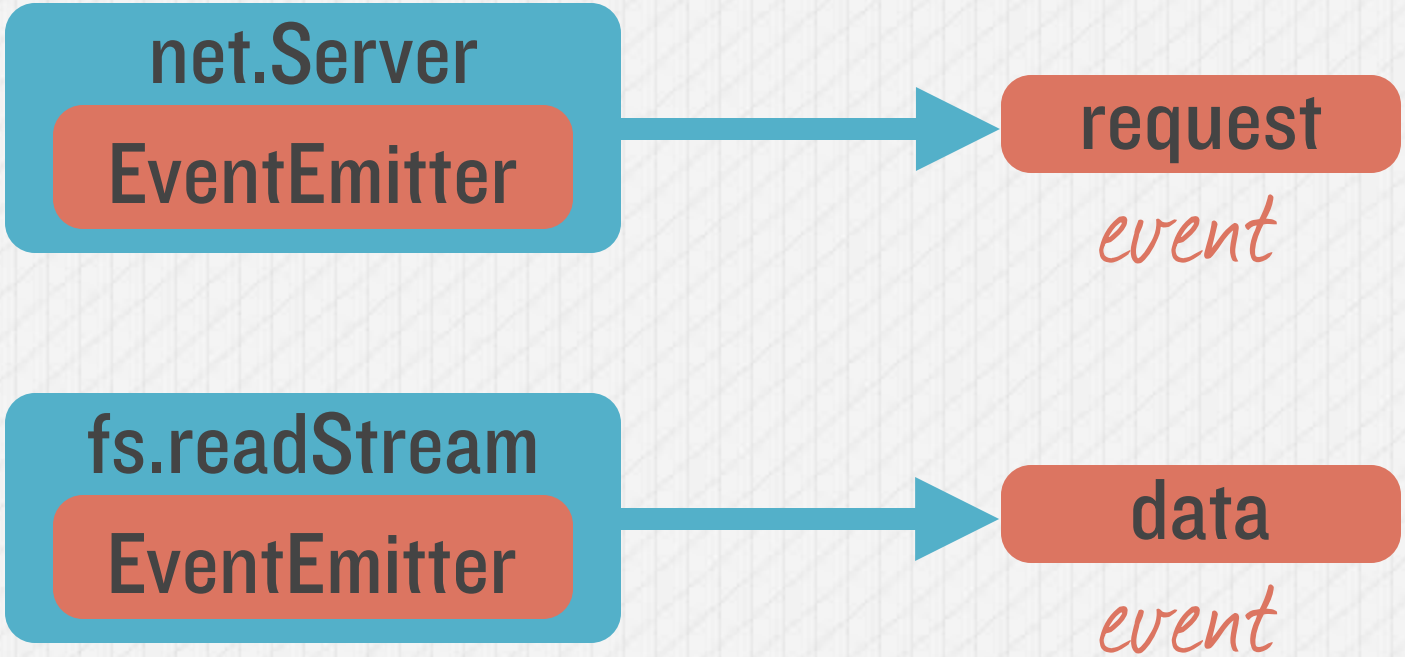



```
$("p").on("click", function(){ ... });
```

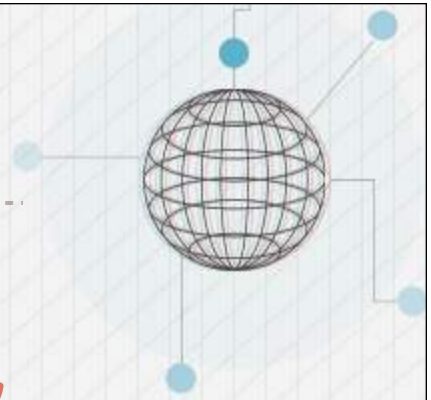
When ‘click’ event is triggered

EVENTS IN NODE

Many objects in Node emit events



CUSTOM EVENT EMITTERS



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

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

events

error warn info

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

listen for error event

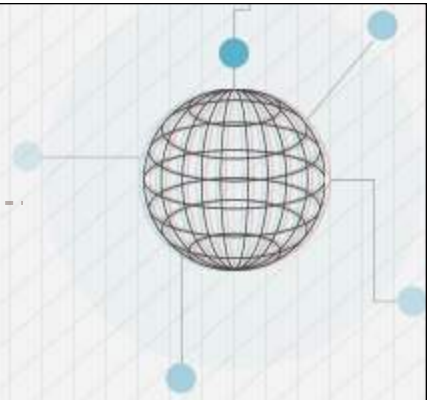
```
logger.emit('error', 'Spilled Milk');
```

-> ERR: Spilled Milk

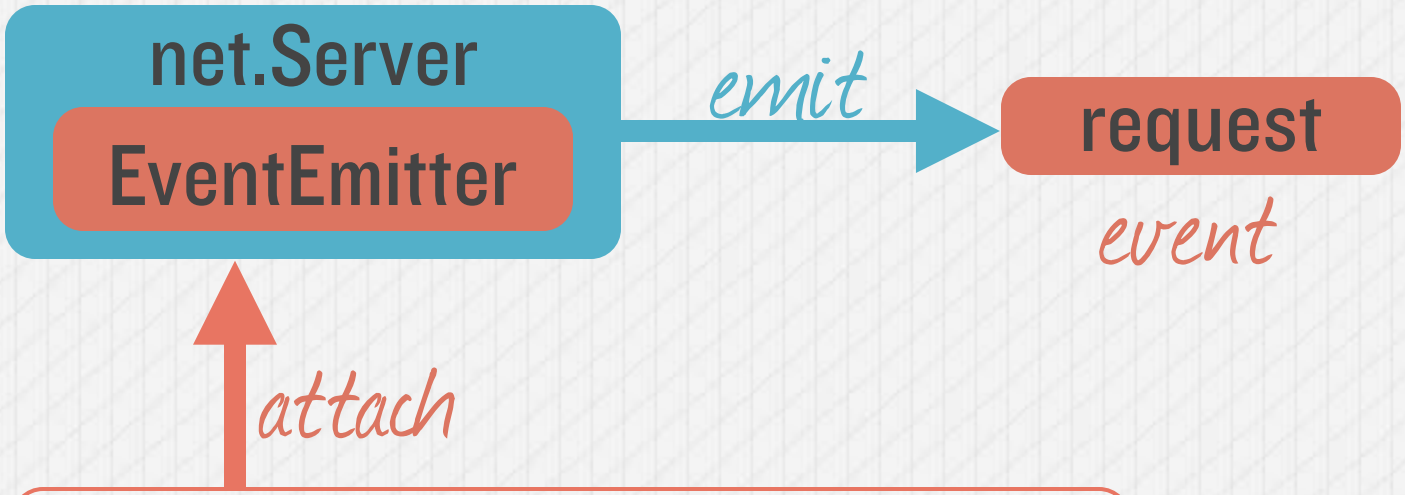
```
logger.emit('error', 'Eggs Cracked');
```

-> ERR: Eggs Cracked

EVENTS IN NODE



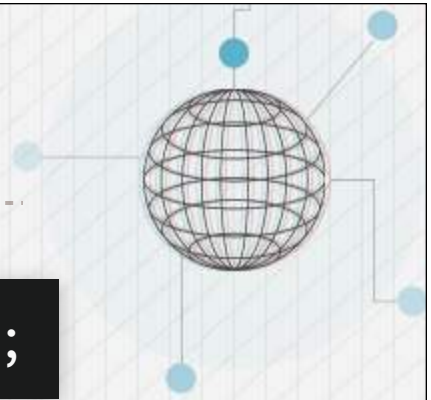
Many objects in Node emit events




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

When ‘request’ event is emitted

HTTP ECHO SERVER

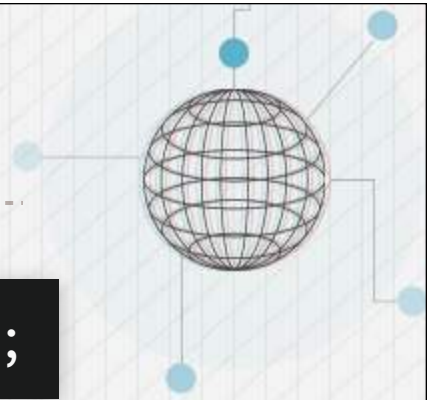


```
http.createServer(function(request, response){ ... });
```

But what is really going on here?

http://nodejs.org/api/

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

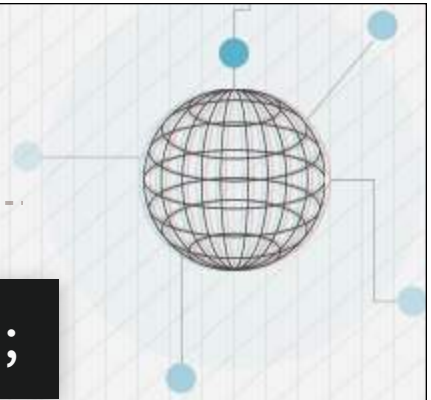
This is an `EventEmitter` with the following events:

Event: 'request'

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

Emitted each time there is a request.

ALTERNATE SYNTAX



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

Same as

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

This is how we add
add event listeners

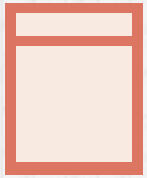
Event: 'close'

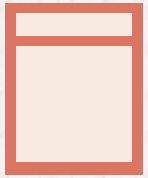
```
function () { }
```

Emitted when the server closes.

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

WHAT ARE STREAMS?





Start Processing Immediately

Streams can be readable, writeable, or both

STREAMING RESPONSE

readable stream

writable stream

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

Our clients receive

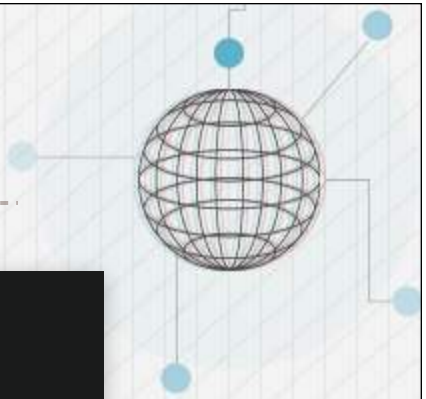
```
"Dog is running."
(5 seconds later)
"Dog is done."
```

HOW TO READ FROM THE REQUEST?

Lets print what we receive from the request.

```
http.createServer(function(request, response) {
  response.writeHead(200);
  request.on('data', function(chunk) {
    console.log(chunk.toString());
  });

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

LETS CREATE AN ECHO SERVER

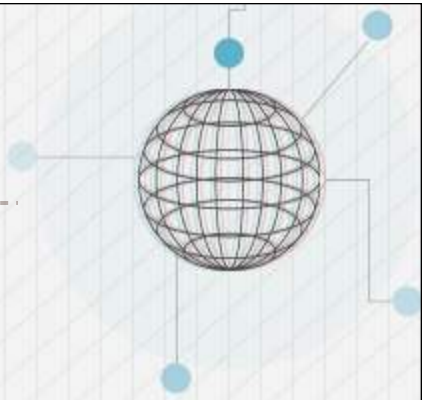


```
http.createServer(function(request, response) {
  response.writeHead(200);
  request.on('data', function(chunk) {
    response.write(chunk);
  });

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

```
request.pipe(response);
```

STREAMS

LETS CREATE AN ECHO SERVER!



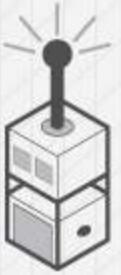
```
http.createServer(function(request, response) {
  response.writeHead(200);
  request.pipe(response);
}).listen(8080)
```

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

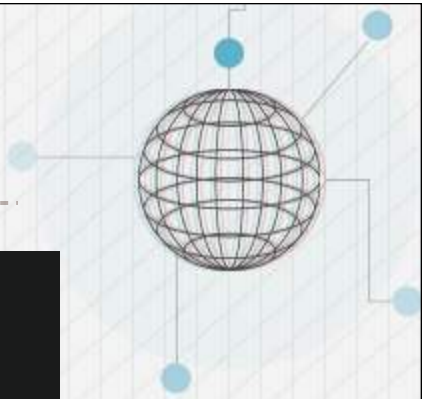
----> Hello *on client*

Kinda like on the command line

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



READING AND WRITING A FILE

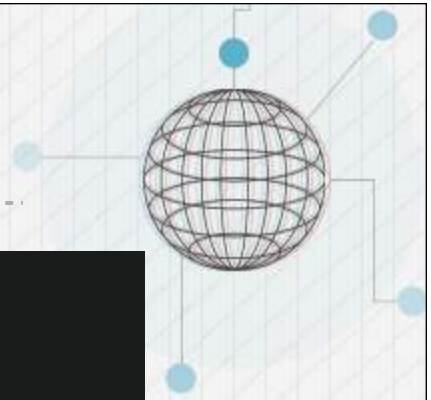


```
var fs = require('fs');
```

require filesystem module

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

file.pipe(newFile);
```

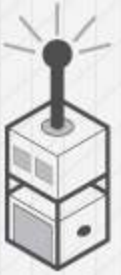
UPLOAD A FILE



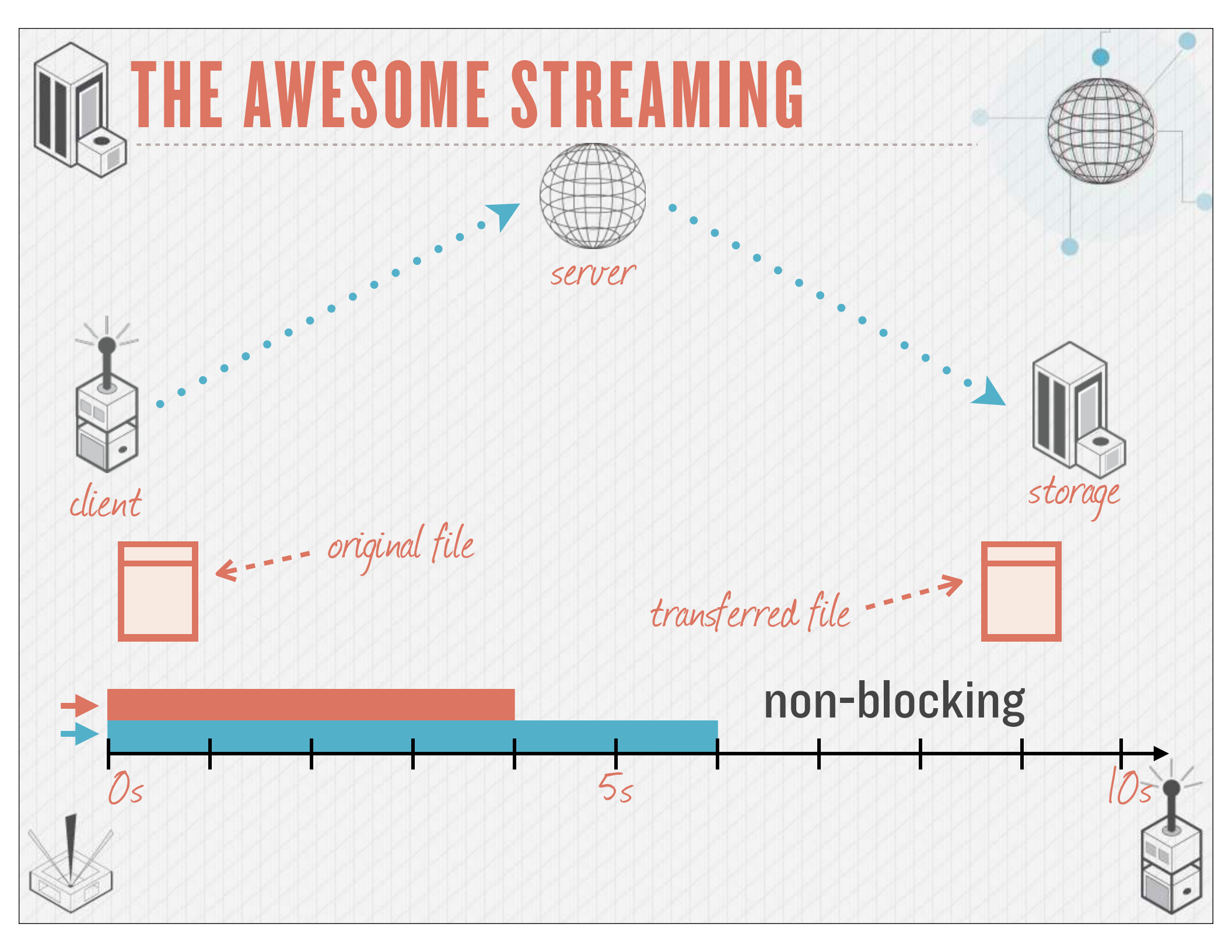
```
var fs = require('fs');
var http = require('http');

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
```

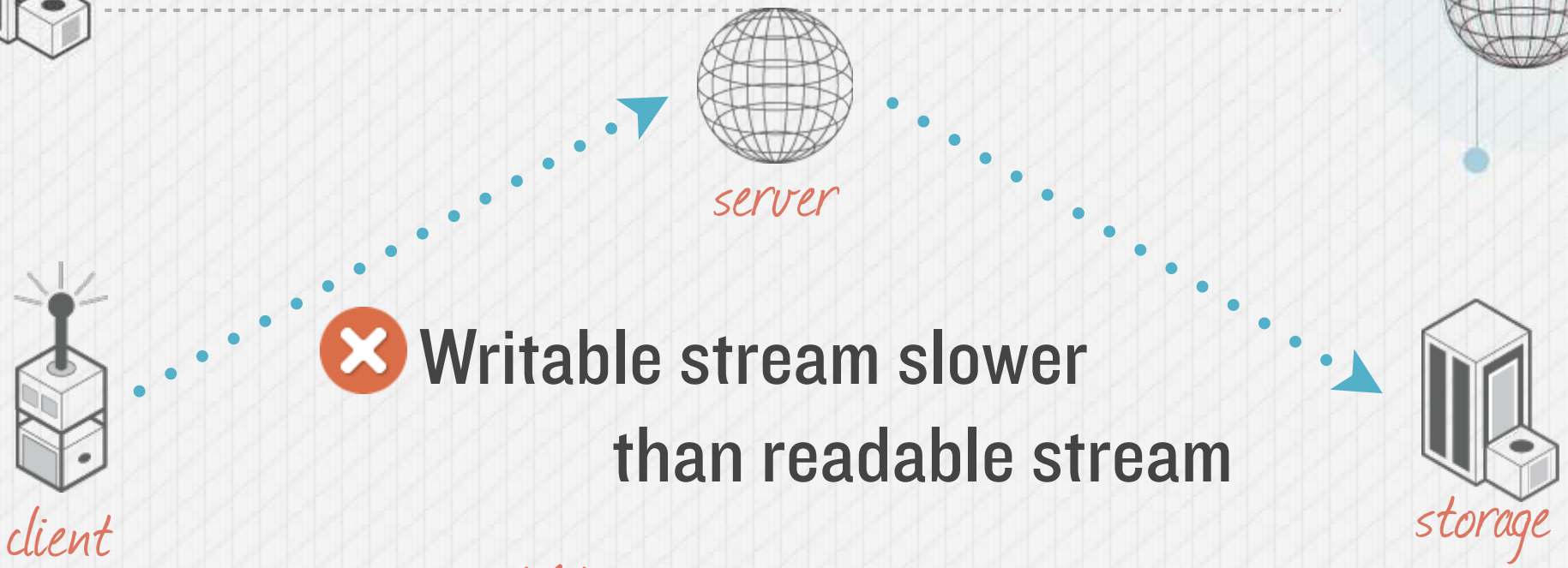
----> uploaded!



THE AWESOME STREAMING



BACK PRESSURE!

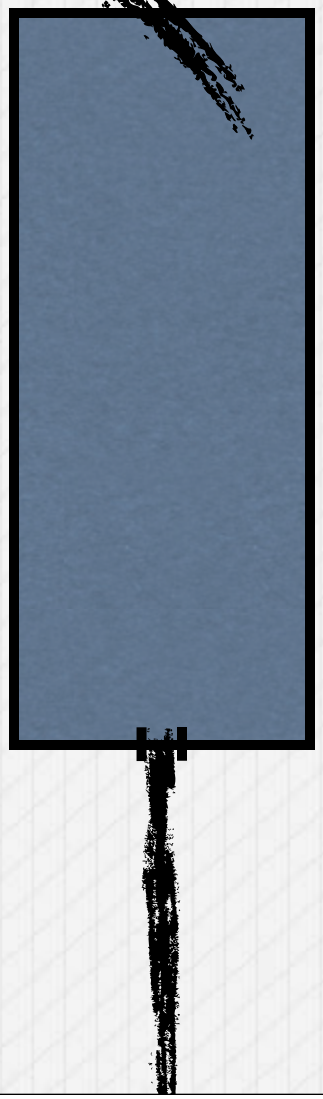


Writable stream slower than readable stream

Using pipe solves this problem

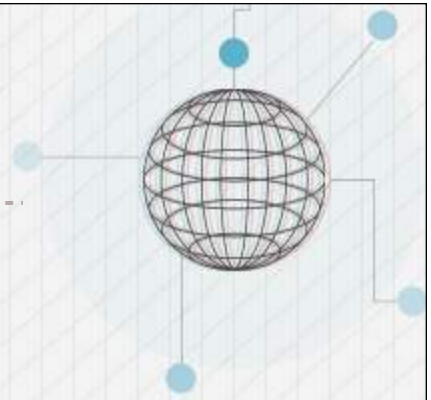
THINK OF A MILK JUG

```
milkStream.pause();
```



```
Once milk jug is drained
    milkStream.resume();
```

PIPE SOLVES BACKPRESSURE



Pause when writeStream is full

```
readStream.on('data', function(chunk) {
   var buffer_good = writeStream.write(chunk);
   if (!buffer_good) readStream.pause();
});
```

returns false if kernel buffer full

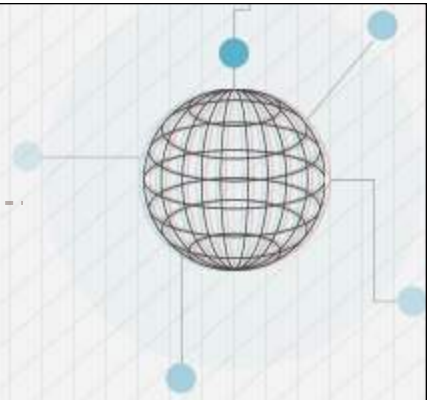
Resume when ready to write again

```
writeStream.on('drain', function(){
   readStream.resume();
});
```

All encapsulated in

```
readStream.pipe(writeStream);
```

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%
```

We're going to need:

- HTTP Server
- File System

STREAMS

DOCUMENTATION

http://nodejs.org/api/

Stability Scores

File System

```
Stability: 3 - Stable
```

File I/O is provided by simple wrappers around standard PO
`require('fs')`. All the methods have asynchronous and sy

The asynchronous form always take a completion callback as
completion callback depend on the method, but the first arg
operation was completed successfully, then the first argumer

When using the synchronous form any exceptions are imme
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) {
```

Stream

```
Stability: 2 - Unstable
```

A stream is an abstract interface implemented by various objects in Node
server is a stream, as is stdout. Streams are readable, writable, or both. Al
`EventEmitter`.

You can load up the Stream base class by doing `require('stream')`.

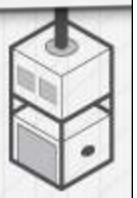
Readable Stream

A `Readable Stream` has the following methods, members, and events.

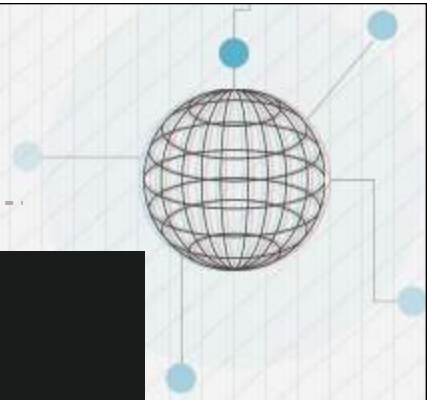
Event: 'data'

`function (data) { }`

The `'data'` event emits either a `Buffer` (by default) or a string if `setE`



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);
```

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.pipe(newFile);

   request.on('data', function(chunk) {
     uploadedBytes += chunk.length;
     var progress = (uploadedBytes / fileBytes) * 100;
     response.write("progress: " + parseInt(progress, 10) + "%\n");
   });
   ...
}).listen(8080);
```