BCDV 1022
Node Core Concepts

2023 Fall

week 03 - class 07

Topics

- Node Core Concepts

Node Core Concepts

Global Scope & Objects

- Browser JavaScript by default puts everything into its global scope, which is bad!
- Node.js was designed to behave differently with everything being local by default.
- When we need access to globals, there is a global object.

- These objects are modules, functions, strings and object:

__filename	- the filename of the code being executed
__dirname	- the name of the directory that the currently executing script resides in
setTimeout(cb, ms)	- used to run callback cb after at least ms milliseconds
clearTimeout(t)	- used to stop a timer that was previously created with setTimeout()
setInterval(cb, ms)	- used to run callback cb repeatedly after at least ms milliseconds
Console	- used to print information on stdout and stderr
Process	- used to get information on current process

Process Object

- The process object is a one of the few global objects provided by the Node.js core API.
- It can be access from anywhere, thus its methods can also be accessed.
- Such is a method called process.nextTick() which allows a callback to be scheduled for the next Event Loop Iteration

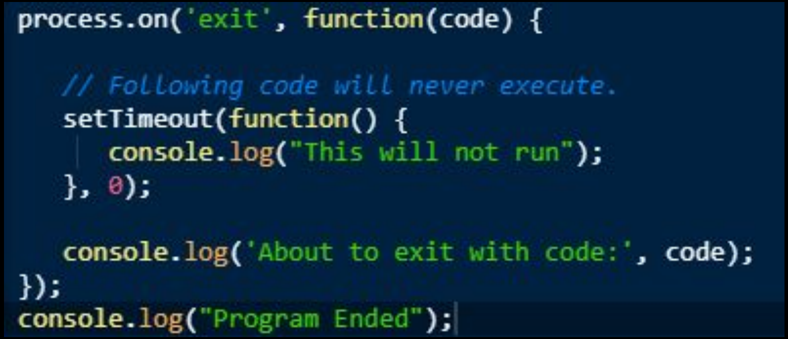
- Some process properties include:
  - process.pid - the PID of the process
  - process.stdout - a writable stream to stdout
  - process.stdin - a writable stream to stdin
  - process.stderr - a writable stream to stderr
  - process.argv - an array containing cmd line arguments

- Some process methods include:
  - process.cwd() - returns the current directory
  - process.kill(pid,[signal]) - sends a signal to kill process
  - process.memoryusage()
  - process.chdir() - change process directory
  - process.uptime()
  - process.nextTick()

Process Events

- The process object is an instance of EventEmitter and emits the following events:

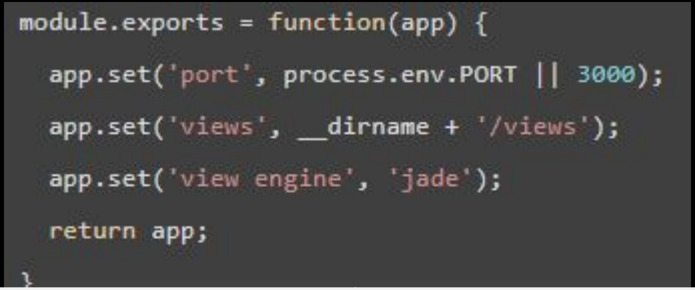
exit	- emitted when the process is about to exit
beforeExit	- emitted when node empties it's event loop and has nothing else scheduled
uncaughtException	- emitted when an exception bubbles all the way back to the event loop
Signal Events	- emitted when the processes receives a signal ie. SIGINT (Signal Interrupt) or SIGHUP (Signal Hang up)



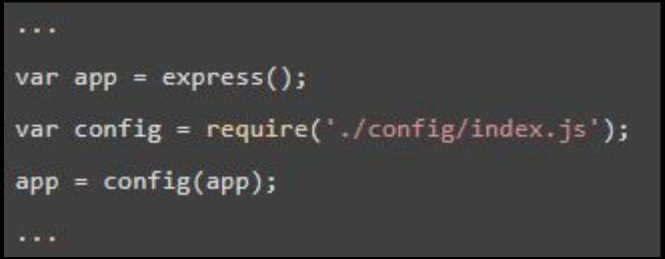
```
process.on('exit', function(code) {

   // Following code will never execute.
   setTimeout(function() {
      console.log("This will not run");
   }, 0);

   console.log('About to exit with code:', code);
});
console.log("Program Ended");
```

Exporting and Importing Modules

- Another bad part in browser JavaScript was that there was no way to include modules (Node was written pre-ES6)
- Node.js borrowed an AJAX inspired module approach from CommonJs
- To export an object in Node.js, use the exports.name = object and then in the file we import use the require keyword
- Another approach is to invoke a constructor ie. when we attach properties in Express.js app



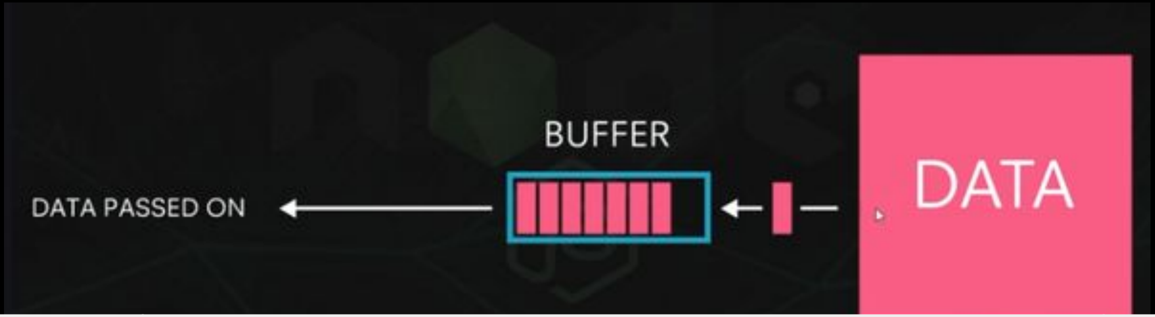
```
module.exports = function(app) {
  app.set('port', process.env.PORT || 3000);
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  return app;
}
```



```
...
var app = express();
var config = require('./config/index.js');
app = config(app);
...
```

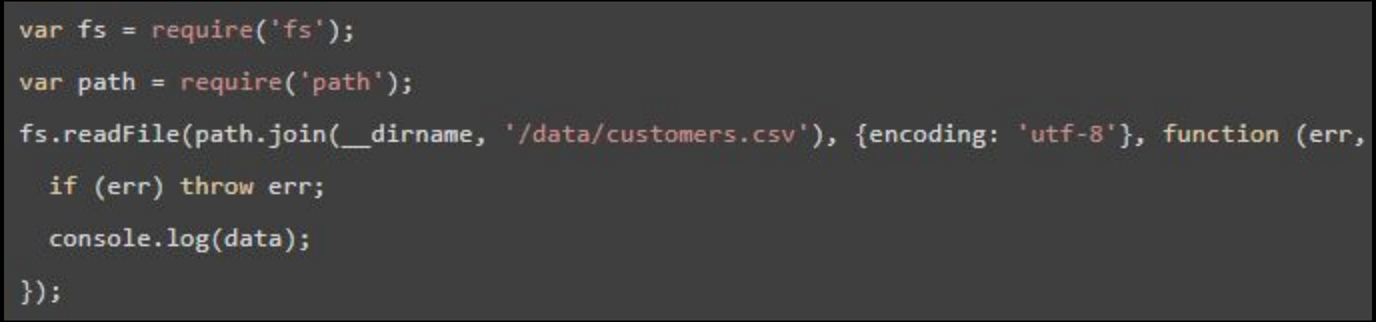
Buffer

- Buffer is a Node.js addition to the four primitives (boolean, string, number and RegExp)
- We can think of buffers as extremely efficient data stores
- Node.js will try to use buffers any time it can eg., reading from file system, receiving packets over the network

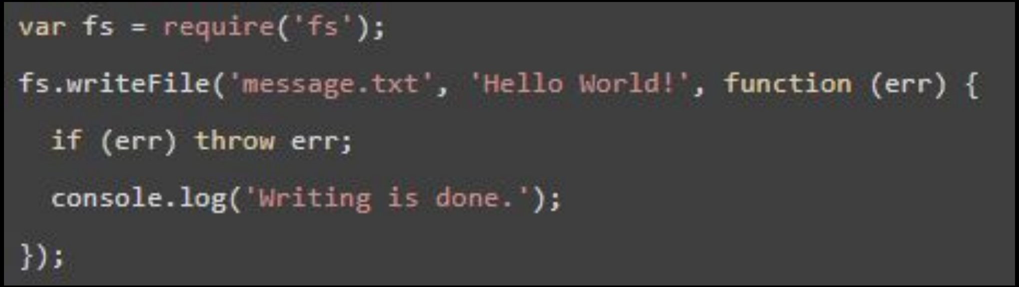


File System

- Reading from files is done via the core fs module
- There are two sets of methods: async and sync, in most cases we should use async methods eg. fs.readFile



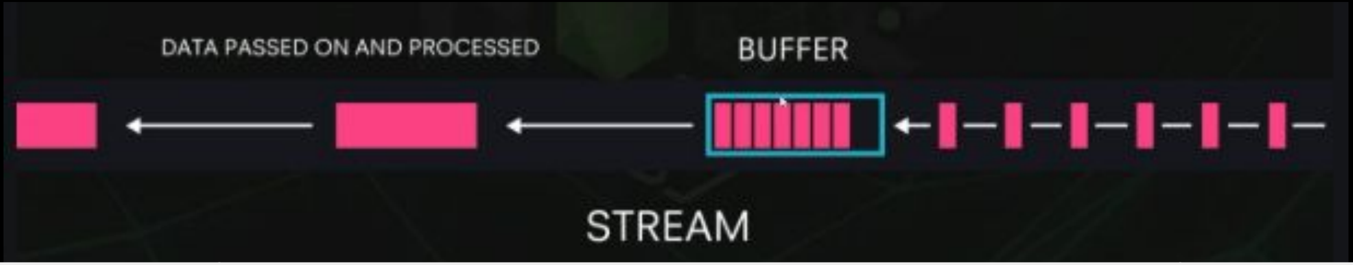
```
var fs = require('fs');
var path = require('path');
fs.readFile(path.join(__dirname, '/data/customers.csv'), {encoding: 'utf-8'}, function (err,
  if (err) throw err;
  console.log(data);
});
```



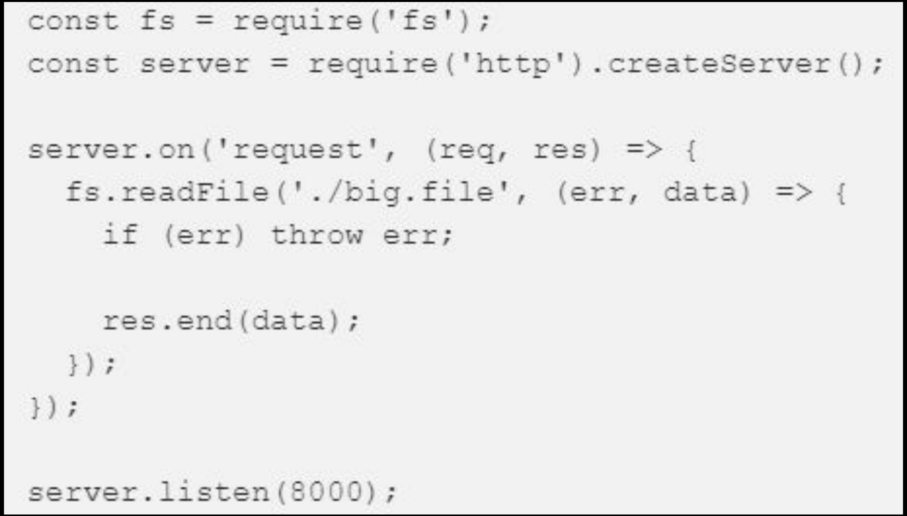
```
var fs = require('fs');
fs.writeFile('message.txt', 'Hello World!', function (err) {
  if (err) throw err;
  console.log('Writing is done.');
});
```

Data Streams

- Streaming data is a term that means processing the data while still receiving it
- Streams might not be available all at once, and they don't have to fit in memory
- This is useful for extra large datasets, like video or data migration
- By default, Node.js uses buffers for streams



Data Streams cont..



```
const fs = require('fs');
const server = require('http').createServer();

server.on('request', (req, res) => {
  fs.readFile('./big.file', (err, data) => {
    if (err) throw err;

    res.end(data);
  });
});

server.listen(8000);
```

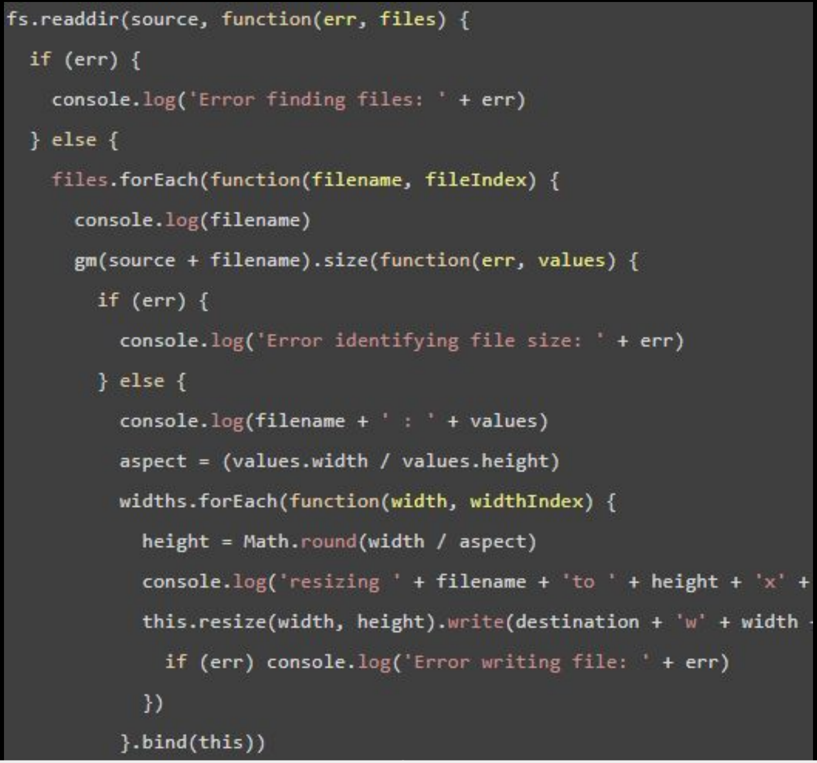
- Reading a file with fs.readFile will put the whole file content in memory before writing out the response.

  Memory consumption will jump over 400 MB of memory

- When we stream it one chunk at a time, using fs.createReadStream, we don't buffer it in memory at all.

  The memory usage grew by 25 MB

Callback Hell



```
fs.readdir(source, function(err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function(filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function(err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function(width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' +
            this.resize(width, height).write(destination + 'w' + width
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
```

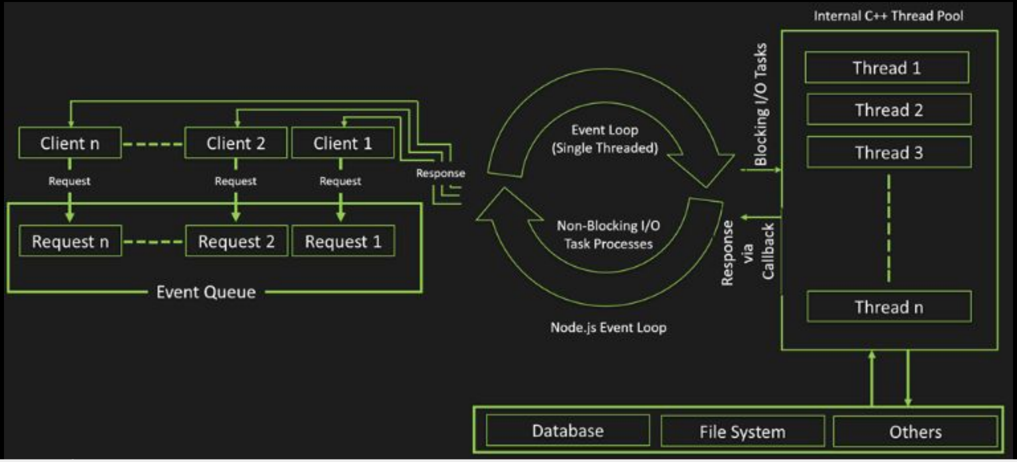
- As long as we have two-space indentation, callback hell is nothing to be afraid of in Node.js
- However, we should try to simplify this with event emitters, promises or using the async library

Event Loop Revisited

Event Loop

- Node.js is an event-based platform. This means that everything that happens in Node is the reaction to an event. A transaction passing through Node traverses a cascade of callbacks..

- This is all handled by a library called libuv which provides a mechanism called an event loop

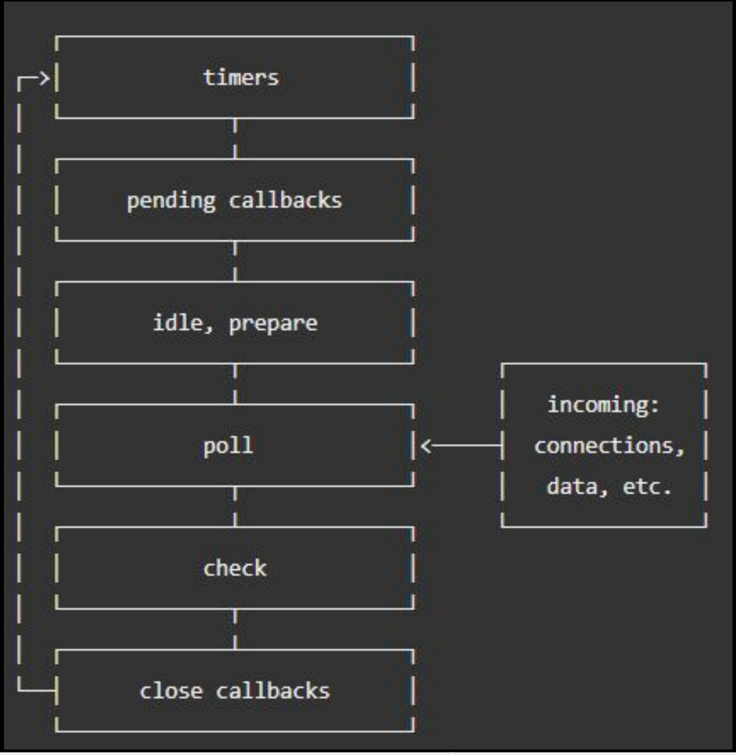


Event Loop Misconceptions

1. **The event loop runs in a separate thread than the user code.**
   - There is only one thread that executes the JavaScript code and this is the thread where the event loop is running
2. **The event loop is something like a stack or queue**
   - The event loop process is a set of phases with specific tasks that are processed in a round-robin manner.

Event Loop phases

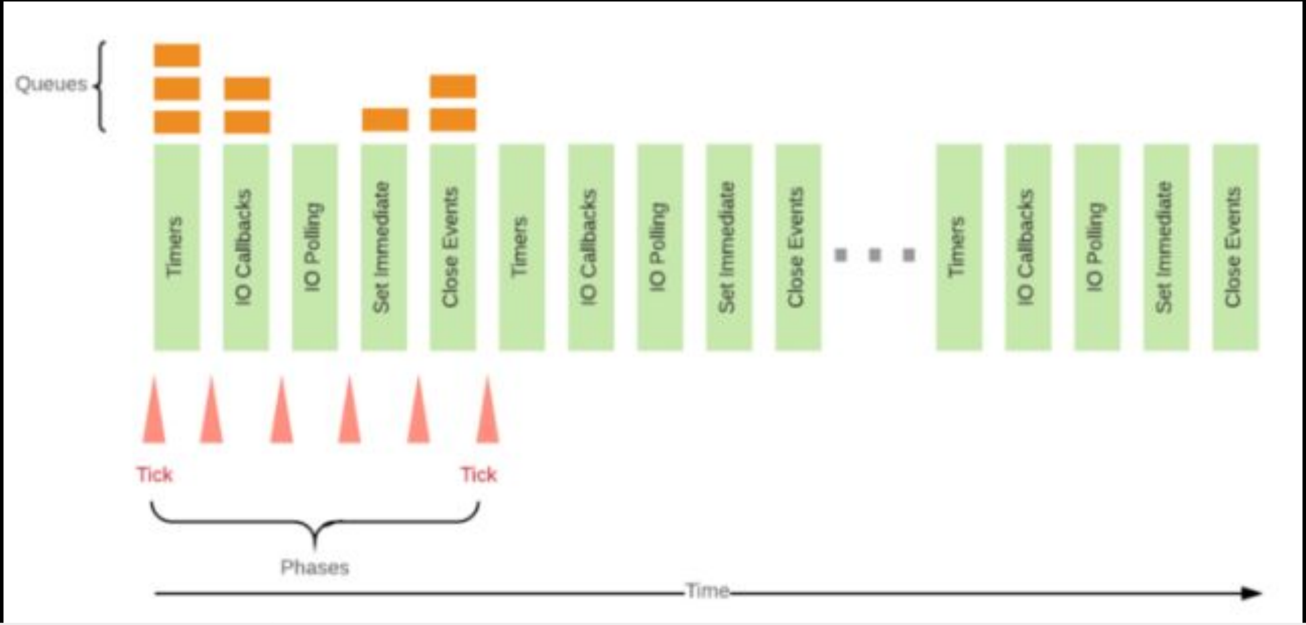
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/



- **Timers**
  - Everything that was scheduled via setTimeOut() or setInterval()
- **IO Callbacks**
  - All code in Node.js is a callback and processed here.
- Idle, prepare
  - Used only internally
- IO Polling
  - Polls for new events to be processed on the next run
- Check
  - setImmediate() callbacks are run here
- Close
  - Here all close callbacks are called (ie. socket.on('close',)

Event Loop ticks and phases

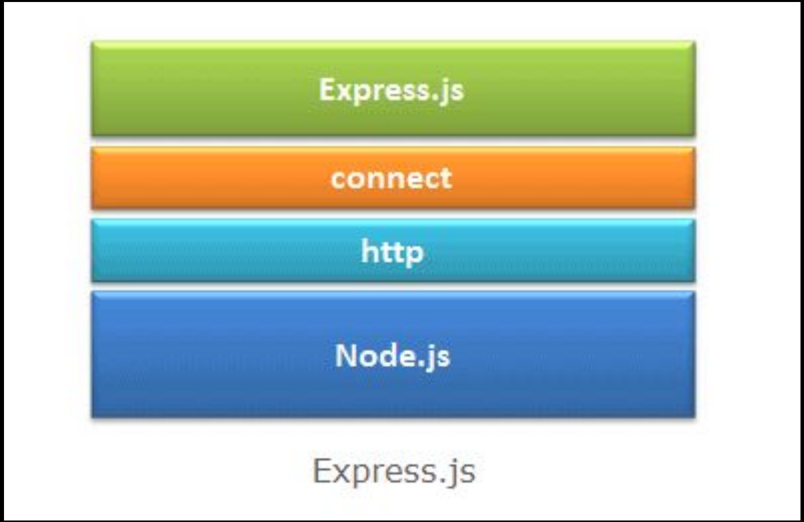
- In Node.js, each iteration of an Event Loop is called a tick.



Express Revisited

Express

Express.js is based on the Node.js middleware module called *connect* which in turn uses http module. So, any middleware which is based on connect will also work with Express.js.



Advantages of Express

1. Makes Node.js web application development fast and easy.
2. Easy to configure and customize.
3. Allows you to define routes of your application based on HTTP methods and URLs.
4. Includes various middleware modules which you can use to perform additional tasks on request and response.
5. Easy to integrate with different template engines like Jade, Vash, EJS etc.
6. Allows you to define an error handling middleware.
7. Easy to serve static files and resources of your application.
8. Allows you to create REST API server.
9. Easy to connect with databases such as MongoDB, Redis, MySQL

Video - Express