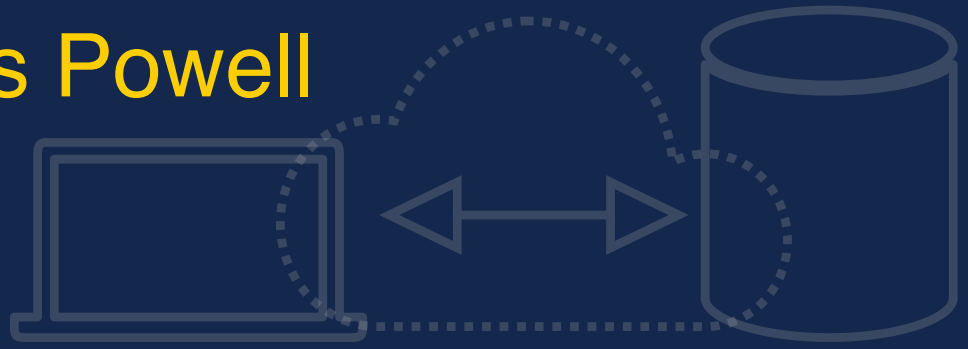
NodeJS Intro

with Thomas Powell

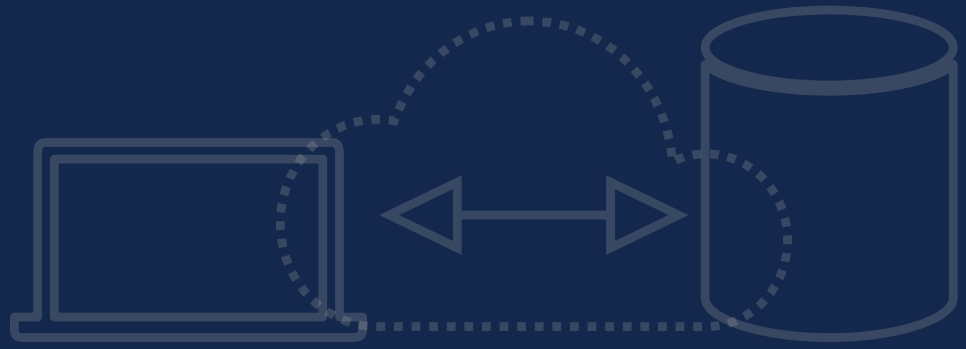


Intro

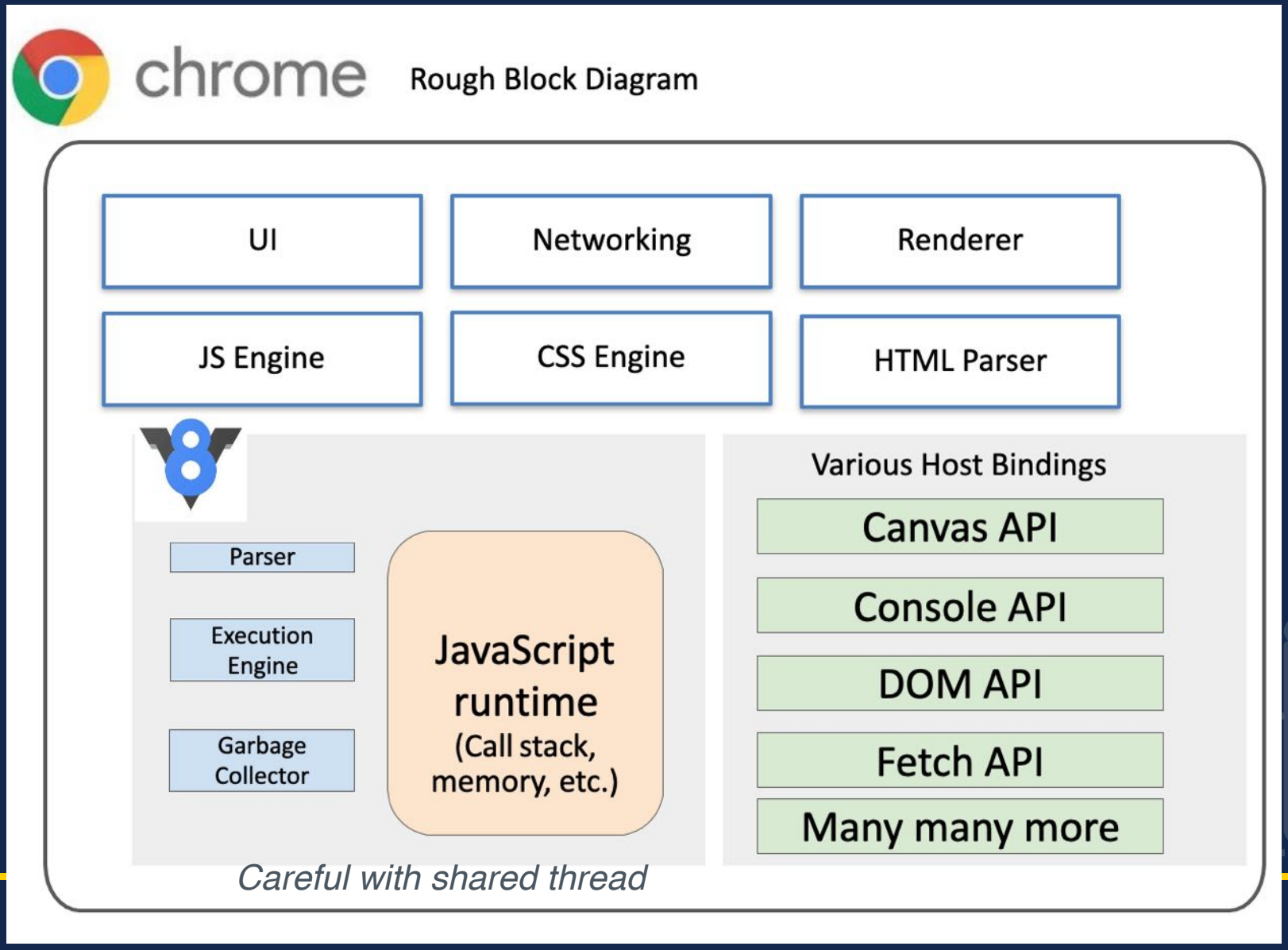
- So far we have tended to look at the idea of files being served or apps being run by a web server such as Apache
- The so-called 4th model for server-side programming is for our code to be the server itself.
- NodeJS is the most common way this done today, but historically even since Perl this approach has been taken
- The approach is interesting in fitting just to what is needed, but in practice many benefits of the pattern are sadly lost

JavaScript in Context

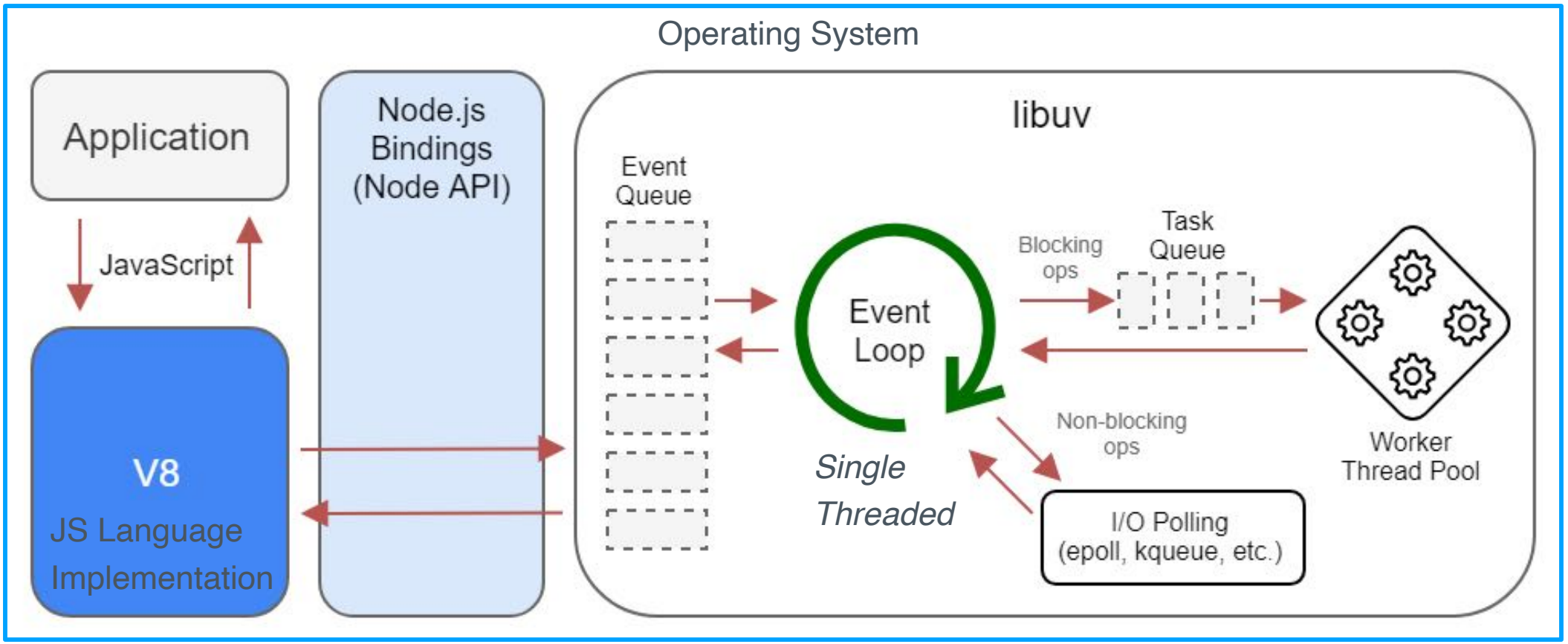
- To properly understand NodeJS we need to be a bit clearer on JavaScript and its role in a host environment
- We compare the use of JavaScript in a browser like Chrome to on a raw operating system such as how NodeJS works
- What we see here is that there is often a confusion between what is host and what is language. Clearing this up in your mind is important to be a good web dev.



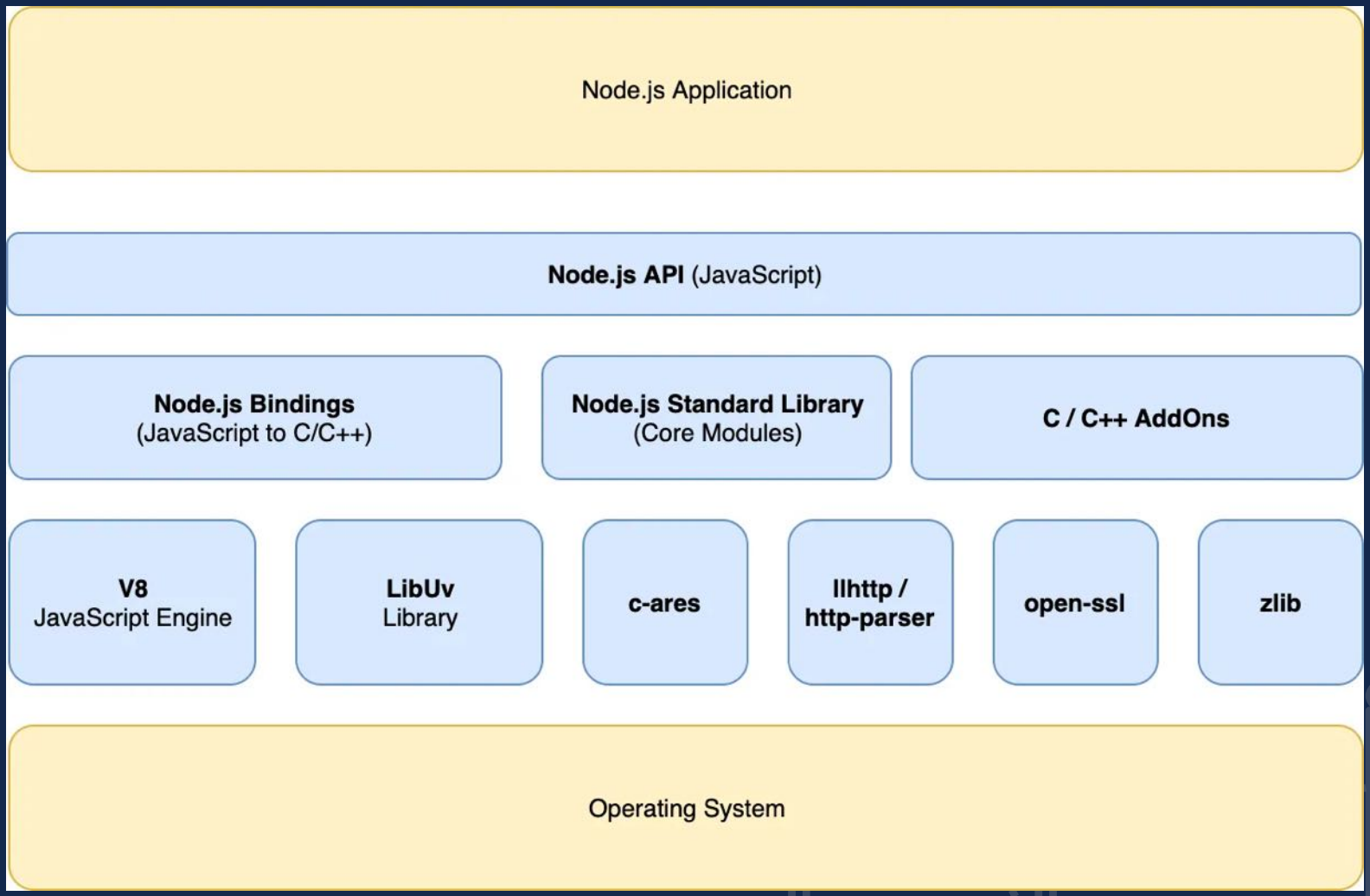
Rough Browser Structure



Rough NodeJS Structure



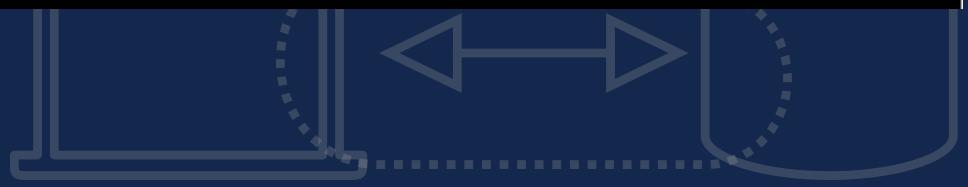
Rough NodeJS Structure Take 2



https://chathuranga94.medium.com/nodejs-architecture-concurrency-model-f71da5f53d1d

Node CLI - REPL (Read-Eval-Print-Loop)

```
 tpowell@thomas  ~  node -v
v20.0.0
 tpowell@thomas  ~  node
Welcome to Node.js v20.0.0.
Type ".help" for more information.
> let school = 'UCSD';
undefined
> let year = 2023;
undefined
> let quarter = 'Spring';
undefined
> console.log(`Hello ${school} students welcome to ${quarter} ${year} CSE135`);
Hello UCSD students welcome to Spring 2023 CSE135
undefined
>
```

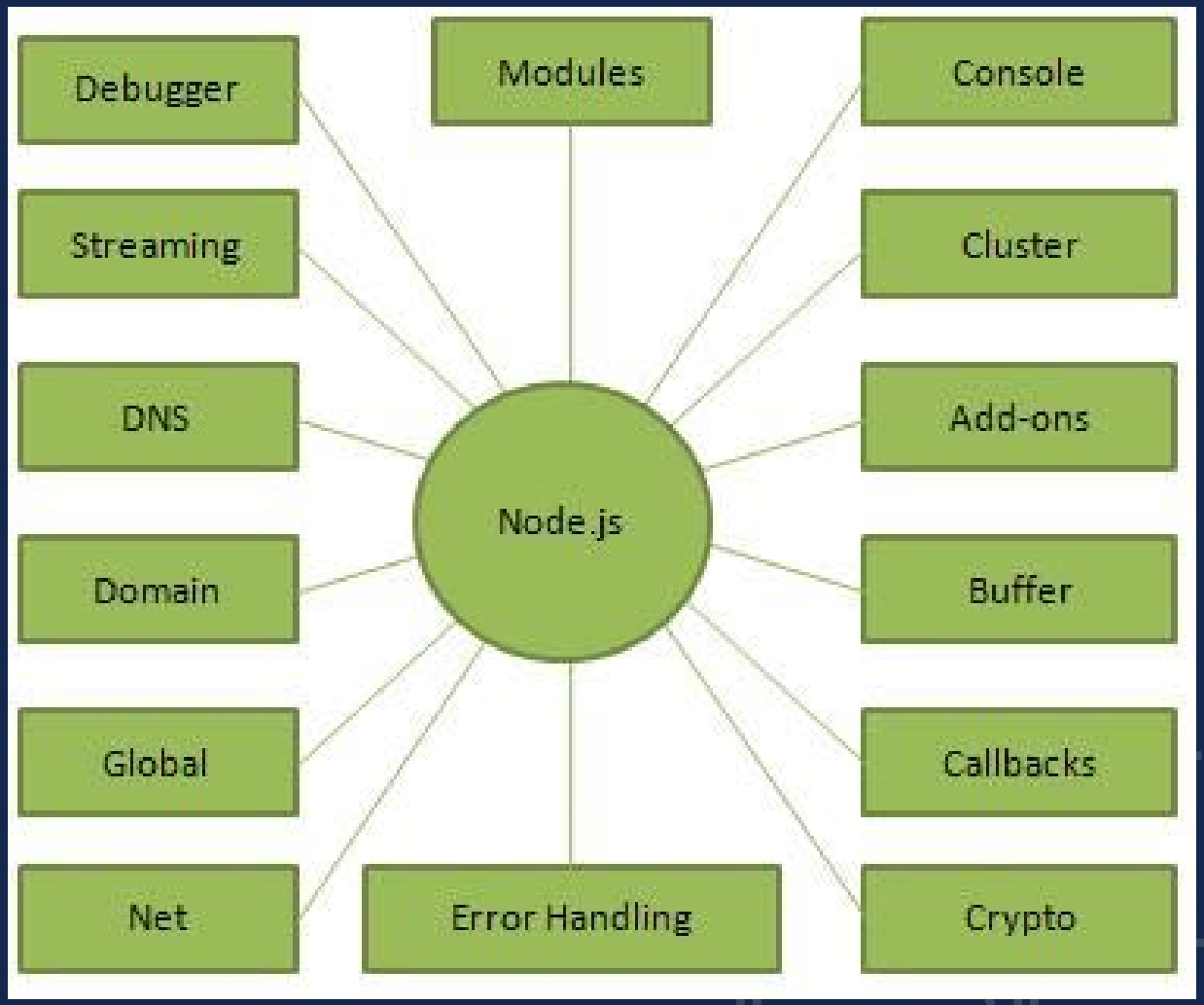


Notice Something API Binding Wise?

```
 tpowell@thomas  ~/tmp  node
Welcome to Node.js v20.0.0.
Type ".help" for more information.
> alert('hello everyone?');
Uncaught ReferenceError: alert is not defined
> document.write('hello everyone?');
Uncaught ReferenceError: document is not defined
> console.log('hello everyone?');
hello everyone?
undefined
>
```

Mostly !=

Common NodeJS Standard Library Overview



REPL Tips

```
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file
```

```
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function foo() {
  console.log('Foo has run!');
  }

undefined
> foo()
Foo has run!
undefined
> .save replfun.js
Session saved to: replfun.js
```

Running NodeJS - Code from a File

Previous hello script saved as a file called ‘hello.js’

```
 tpowell@thomas  ~/tmp  node hello.js
Hello UCSD students welcome to Spring 2023 CSE 135
```

```
 tpowell@thomas  ~/tmp  node < hello.js
Hello UCSD students welcome to Spring 2023 CSE 135
```

```
 tpowell@thomas  ~/tmp  node -i -e "$(< hello.js)"

Welcome to Node.js v20.0.0.
Type ".help" for more information.
> Hello UCSD students welcome to Spring 2023 CSE 135
|
```

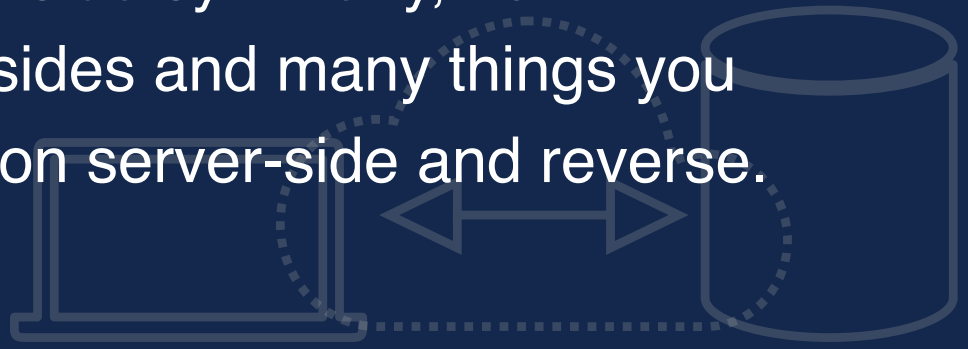
Running NodeJS Loading a File

Previous hello script saved as a file called ‘hello.js’

```
 tpowell@thomas  ~/tmp  node
Welcome to Node.js v20.0.0.
Type ".help" for more information.
> .load hello.js
let school = 'UCSD';
let year = 2023;
let quarter = 'Spring';

console.log(`Hello ${school} students welcome to ${quarter} ${year} CSE 135`);

Hello UCSD students welcome to Spring 2023 CSE 135
undefined
>
```

Some NodeJS JS Tips

- Don’t abandon ES6 concepts if anything lean into them as you are in a controlled server-side environment - rest, spread, template literals, promises, etc.
- Understand the ongoing transition from standard NodeJS require syntax and ES6 import style syntax for modules
- Be careful with client-side / server-side symmetry, not everything makes sense on both sides and many things you take for granted client-side aren’t on server-side and reverse.

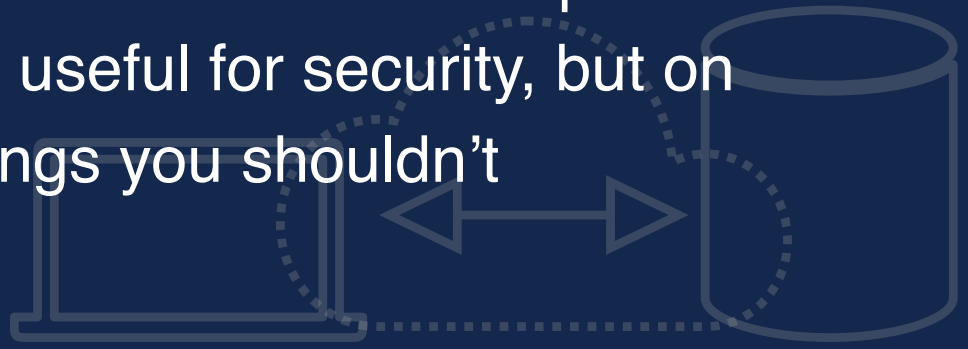


NodeJS Hello HTTP Server

```
import * as http from 'http';

const PORT = 8081;
let template = `
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Hello Node World</title>
</head>
<body>
<h1>Hello Node World at ${new Date()}</h1>
</body>
</html>
`;

let server = http.createServer(function(req, res) {
    res.statusCode = 200;
    res.statusMessage = 'Groovy!';

    res.setHeader('Content-Length', template.length);
    res.setHeader('Server', 'CSE135');
    res.setHeader('X-Powered-By', 'UCSD CSE!');
    res.setHeader('Content-Type', 'text/html');

    res.end(template);
});

server.listen(PORT, '127.0.0.1');
console.log(`Server running on http://127.0.0.1:${PORT}`);
```

Building Your Own Server

- You certainly can do that, but you are both positively and negatively trivializing what Apache does
- Remember what comes free with Apache: HTTP protocol hardening, file system based routing, basic authentication, logging, etc.
- You can of course re-do *some* of this which on the pro sides limits surface area which is useful for security, but on the con side might miss many things you shouldn't



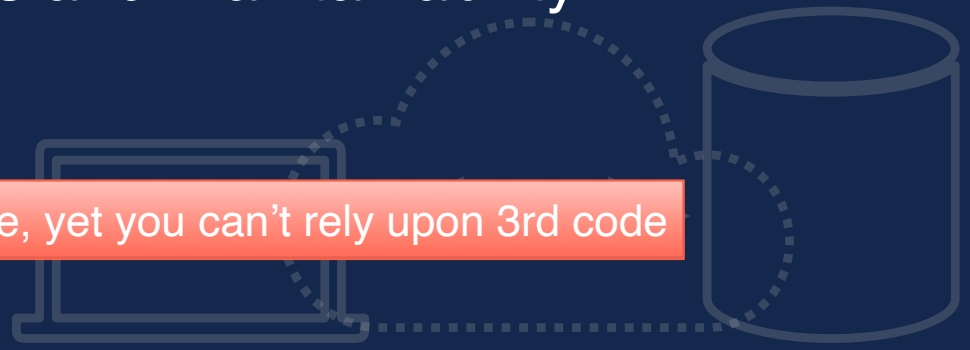
ExpressJS

- ExpressJS is a web application framework for Node.js that eases the process of building web applications and APIs
- Provides easy ways to handle requests, responses, mapping routes to files or code, rendering templates, and more.
- A key aspect of Express is the use of middleware and a rich 3rd party ecosystem of middleware packages.

https://expressjs.com/

Npm

- The default package manager for NodeJS
- Allows easy discovery, installation and management of 3rd party code to be used in a NodeJS application
- Super useful, but also leads people to install too much or be unaware of dependency problems including quality, performance, security, robustness and maintainability



Pro Tip? You need to rely on 3rd party code, yet you can't rely upon 3rd code

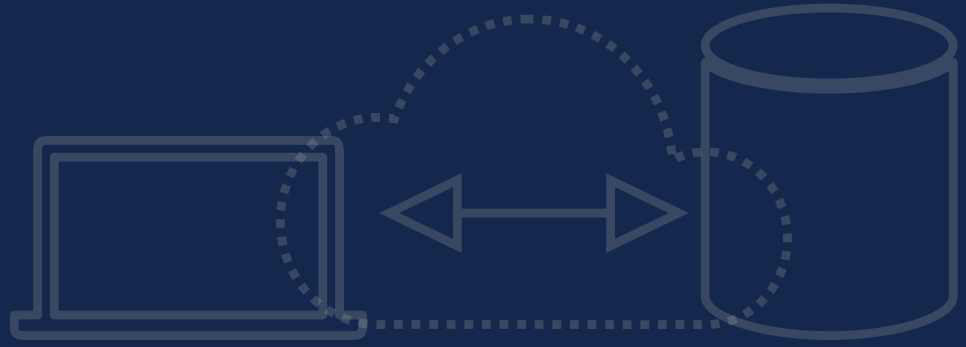
Hello Express

```
const express = require('express');
const app = express();
const PORT = 3000;

app.get('/', (req, res) => {
    res.send('Hello World!');
});

app.listen(PORT, () => {
    console.log(`Express listening on http://localhost:${PORT}`);
});
```

package.json

- `package.json` is a JSON file that stores metadata about a Node.js application and its dependencies

- The file can contain many things but the main concepts include:
  - Project meta data such as name, version, and description
  - Specifies dependencies and their version requirements
  - Defines custom scripts for automating common tasks, like testing and building

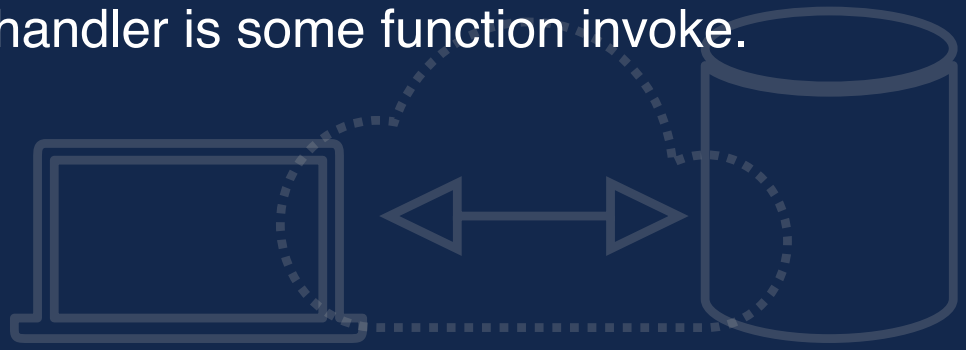


Route Handling in Express

- A route is a somewhat jargonny way of saying a URL that triggers something or returns some content.
  - Basically all URLs served by a web server are routes!

- In applications though we tend to associate some piece of code with a particular route and HTTP method. Express makes this easy with the following syntax `app.METHOD(PATH, HANDLER)`

  - Replace METHOD = GET, POST, PUT, etc. PATH gets replaced with a relate path like /, /api, etc. and the handler is some function invoke.



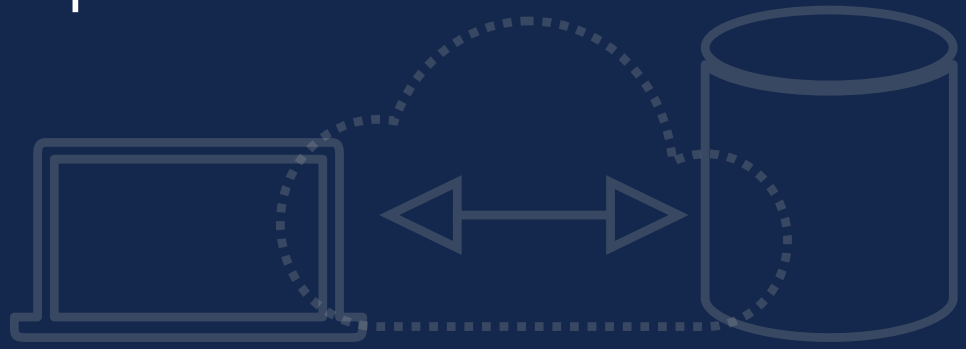
Simple Route Examples

- The snippet here shows listening for a GET request on the URL /api/dogs as well as a DELETE request with a parameter (:id) of a record to delete

```
app.get('/api/dogs', (req, res) => {
    res.send('GET all the dogs from API');
});

// A route with a different method and the use of a parameter
app.delete('/api/dogs/:id', (req, res) => {
    let id = req.params.id;
    res.send(`DELETE: Deleting ${id} from with the dogs API`);
});
```

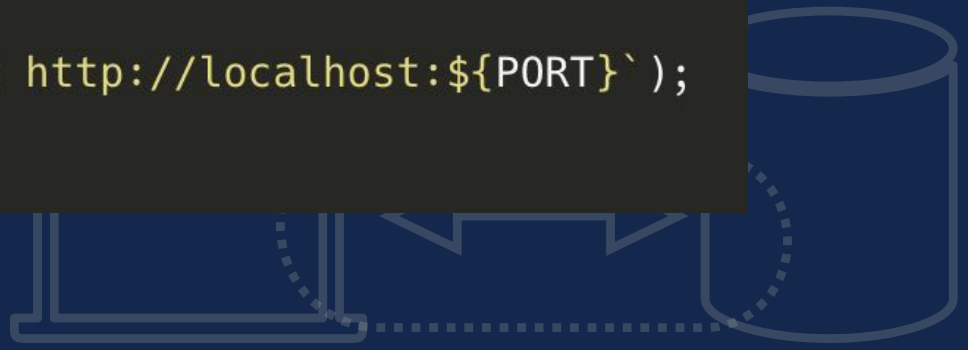
File Routing in Express

- Obviously slavishly adding in static file routes and such would be a bit of work
- Further how do we did with 404s not specified URLs?
- We can address much of this using file serving with ExpressJS which is done easily static file handling middleware which is common to Express



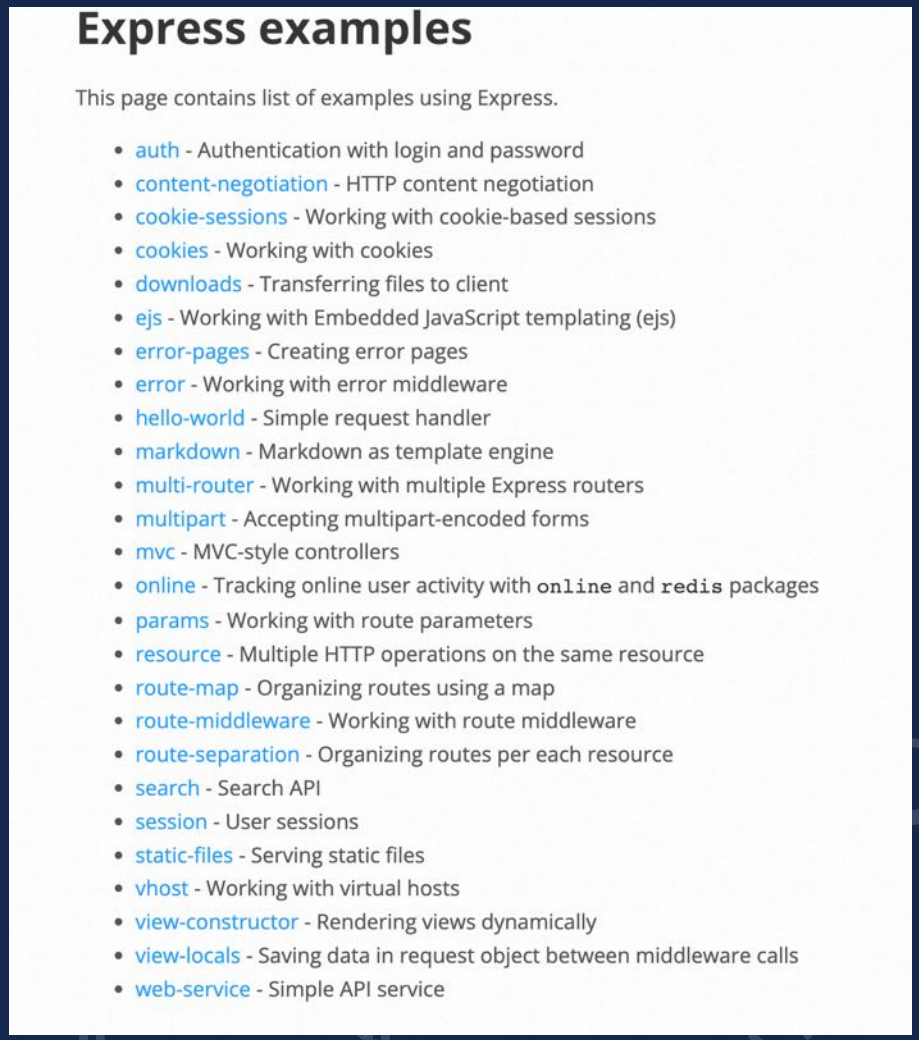
File Routing Demo

```
const express = require('express');
const app = express();
const PORT = 3000;

app.use(express.static('3-public'));

app.get('/api', (req, res) => {
    res.send('API placeholder');
});

app.listen(PORT, () => {
    console.log(`Express listening on http://localhost:${PORT}`);
});
```



Tip of the Web Server, HTTP, App Ice Berg

- *A sample of the examples for ExpressJS showing some familiar topics*



Express examples

This page contains list of examples using Express.

- auth - Authentication with login and password
- content-negotiation - HTTP content negotiation
- cookie-sessions - Working with cookie-based sessions
- cookies - Working with cookies
- downloads - Transferring files to client
- ejs - Working with Embedded JavaScript templating (ejs)
- error-pages - Creating error pages
- error - Working with error middleware
- hello-world - Simple request handler
- markdown - Markdown as template engine
- multi-router - Working with multiple Express routers
- multipart - Accepting multipart-encoded forms
- mvc - MVC-style controllers
- online - Tracking online user activity with `online` and `redis` packages
- params - Working with route parameters
- resource - Multiple HTTP operations on the same resource
- route-map - Organizing routes using a map
- route-middleware - Working with route middleware
- route-separation - Organizing routes per each resource
- search - Search API
- session - User sessions
- static-files - Serving static files
- vhost - Working with virtual hosts
- view-constructor - Rendering views dynamically
- view-locals - Saving data in request object between middleware calls
- web-service - Simple API service

Summary

- NodeJS is server-side development using JavaScript
- NodeJS shows us clearly that JS is not a function of a web browser, it can be hosted anywhere we have an execution engine like V8
- NodeJS is well suited to building *microservices* and helps us leverage our JavaScript and web knowledge quickly
- The use of 3rd party packages such as ExpressJS installed with npm is both a blessing (ease of dev!) and a curse (choice issues, resource explosion, and dependency nightmares)