Node JS Workshop

Aaron Czichon

Aaron Czichon

- Senior Software Engineer @ PlanB. GmbH
- Past: Conclurer GmbH, cellent AG
- Focus on Web & Javascript Technologies
- Mostly: NodeJS., Angular, Ionic
- Speaker at Conferences
- Author for Print and Online Magazines

Agenda – Day 1

Time	Chapter
09:00 – 09:15	Introduction / Get to know
09:15 – 10:45 (with 5min break at 09:55)	History of Node, Javascript Client vs. Server, Alternatives to Node
10:45– 11:00	Break
11:00 – 12:45 (with 5min break at 11:55)	Runtime V8, Modules in Node, NodeJS Path Module
12:45 – 13:45	Lunch Break
13:45 – 15:30 (with 10min break at 14:45)	Event Handling, Synchronous Execution, Asynchronous Execution
15:30 – 15:45	Break
15:45 – 16:30	Event Loop, Streams and Buffers, File Handling

Agenda – Day 2

Time	Chapter
09:00 – 09:15	Wrap-Up / Open Questions
09:15 – 10:45 (with 5min break at 09:55)	Asynchronous JS, WebAPIs with Node, Getting Started with Express
10:45– 11:00	Break
11:00 – 12:45 (with 5min break at 11:55)	Architecture of Express Apps, Routing, Middlewares
12:45 – 13:45	Lunch Break
13:45 – 15:30 (with 10min break at 14:45)	Security, Working with Databases, OR-Mapper
15:30 – 15:45	Break
15:45 – 16:30	Tooling and Documentation of APIs

Agenda – Day 3

Time	Chapter
09:00 – 09:15	Wrap-Up / Open Questions
09:15 – 10:45 (with 5min break at 09:55)	Authentication and Authorization, Versioning of APIs, (Client-First with GraphQL)
10:45– 11:00	Break
11:00 – 12:45 (with 5min break at 11:55)	Server-Side Rendering, View Engines
12:45 – 13:45	Lunch Break
13:45 – 15:30 (with 10min break at 14:45)	Deployment
15:30 – 15:45	Break
15:45 – 16:30	Code-Sharing between Frontend and Backend, Review & Feedback

The History of NodeJS

History of NodeJS

Initially written
from Ryan Dahl

2009

History of NodeJS

Beginning of Express and socket.io

2010

History of NodeJS

Release of NPM, Adoption of Node by Uber, LinkedIn, etc.

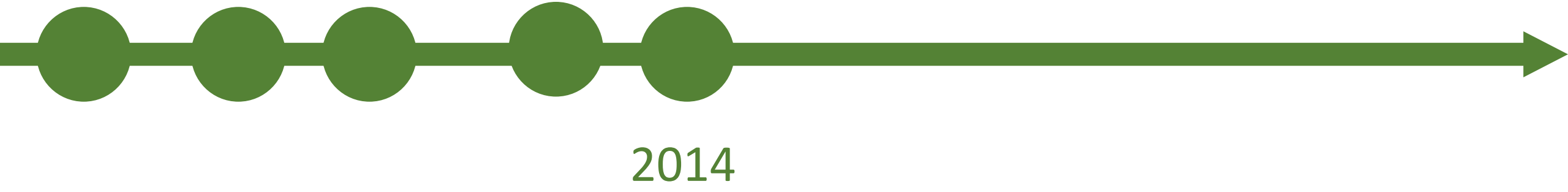
2011

History of NodeJS

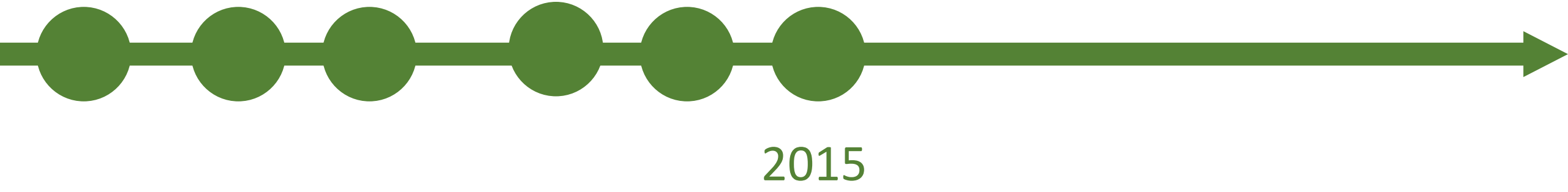
Blogging
Platform: Ghost

2013

History of NodeJS

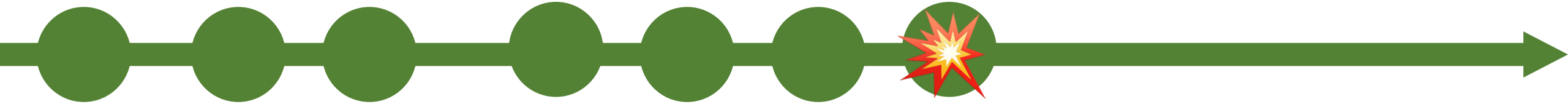


History of NodeJS



History of NodeJS

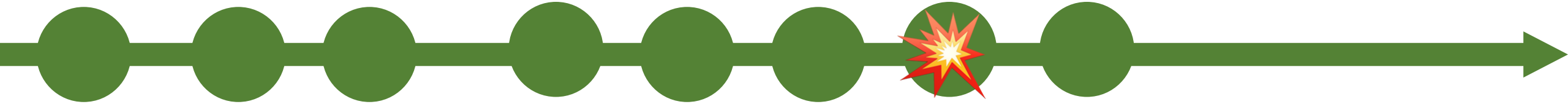
Leftpad Incident,
Start of Yarn



2016

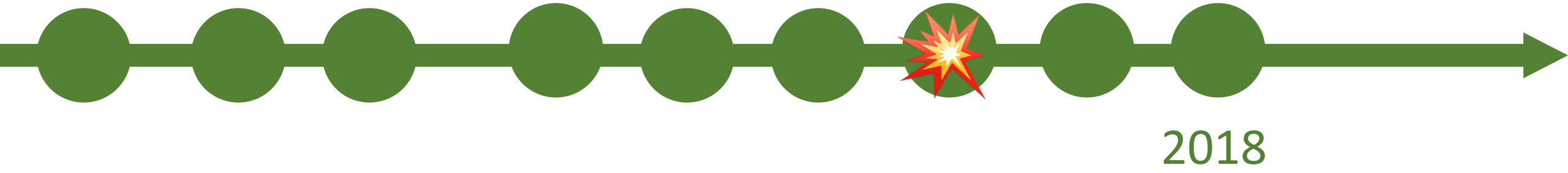
History of NodeJS

NPM security update,
HTTP/2,
V8 official target

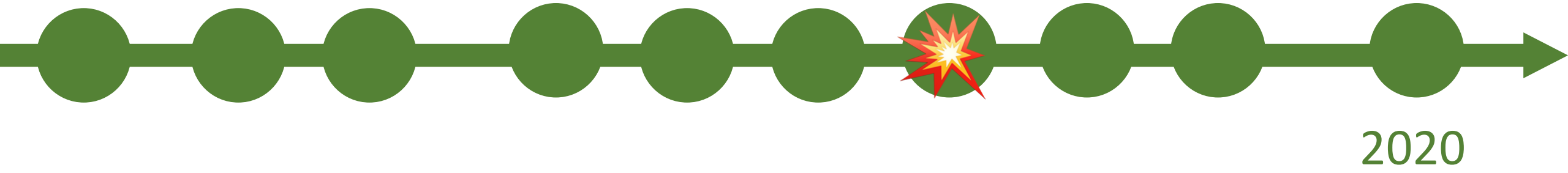


2017

History of NodeJS



History of NodeJS

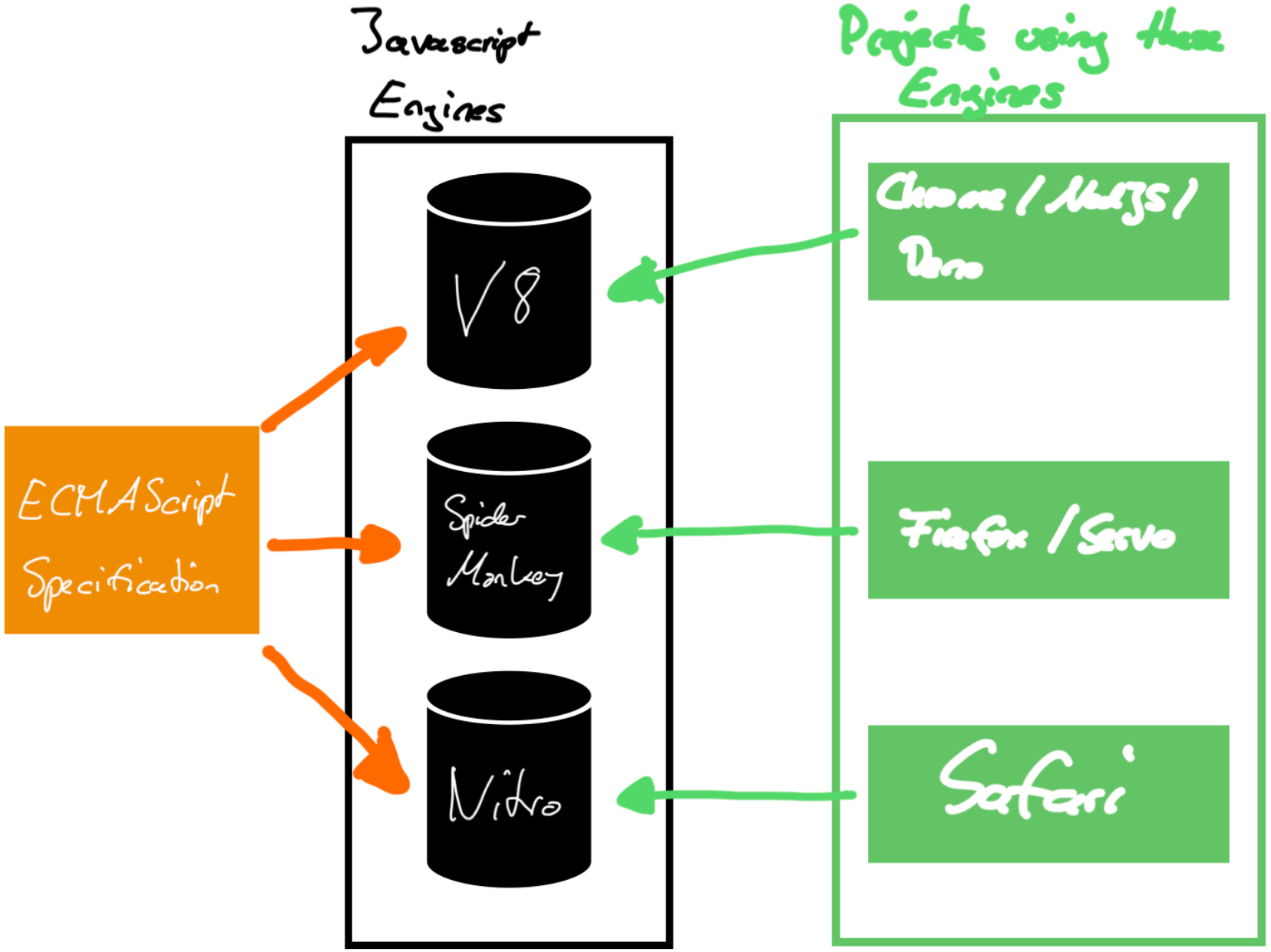


Javascript – Client vs. Server

Javascript Client vs. Server

- For Javascript there only exist the ECMAScript Specification
- Defines how the implementation should work and react, not how it’s implemented

Javascript Client vs. Server



Javascript Client vs. Server

- Depending on your project you may not use all engine features

Javascript Client vs. Server

- Depending on your project you may not use all engine features
- For server-side projects this mean e.g. that they don’t use the ”window” context

Javascript Client vs. Server

- On the other hand, server-side projects have additional functions like:
  - I/O
  - Steams and Buffers
  - File Handling

Alternatives to NodeJS

Alternatives to NodeJS

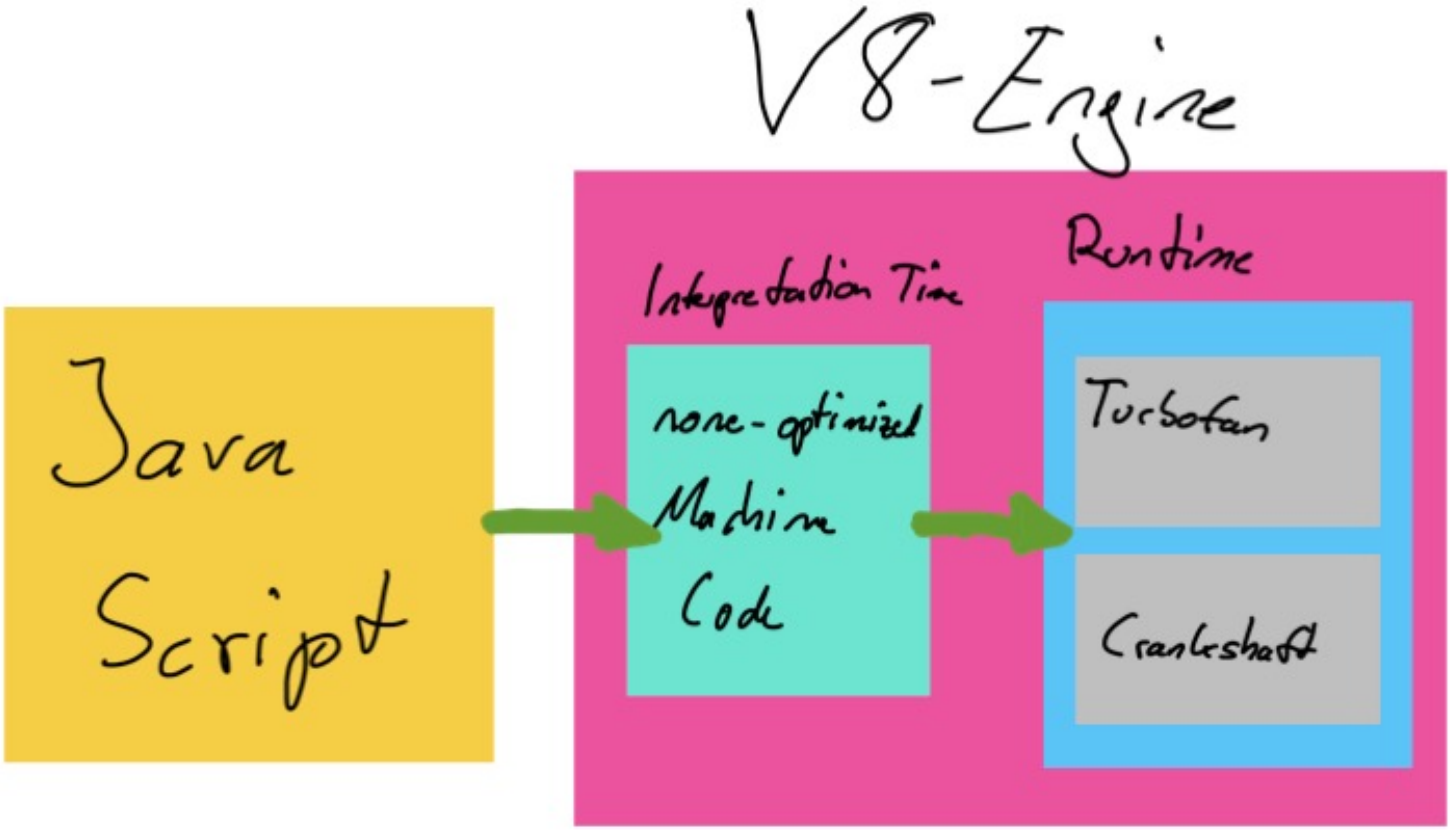
- Deno
- Bun

Runtime V8 in NodeJS

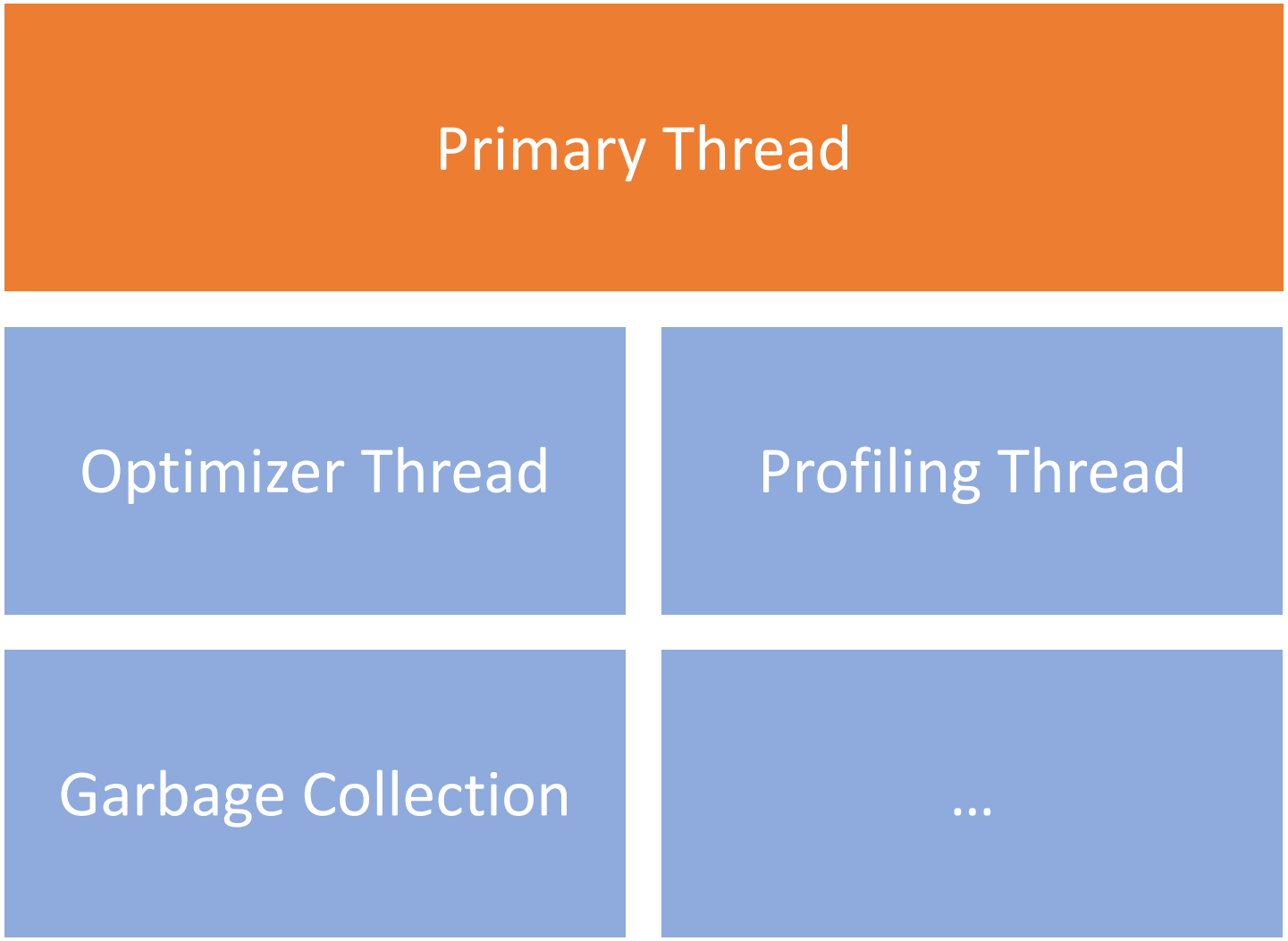
Runtime V8

- Engine is an Interpreter which executes Javascript
  - Standard Interpreter (like SpiderMonkey)
  - Just in Time (JIT) Compilation (like V8)

Runtime V8


V8-Engine
Java Script
Integretation Time
none-optimized Machine Code
Runtime
Turbofan
Crankshaft

Runtime V8


Primary Thread
Optimizer Thread
Profiling Thread
Garbage Collection
...

Getting Started with NodeJS

Tasks in this Workshop

Tasks in this workhop

- Tasks slides are colored green and have a number
- I’ll post the tasks into the classroom of the workshop if we’re there
- After a specific time I post the solution for the task
- You can also use the Github repository which I post in the chat

Task 1

Check NodeJS if installed correctly and init

Modules in NodeJS

Modules

- Encapsulated and reusable code
- Has it’s own context
- NodeJS treads every file as it’s own module

Module Types

Local Modules

Module Types

Local Modules

Build-In Modules

Module Types

Local Modules

Build-In Modules

Third-Party Modules

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

CommonJS Modules

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

CommonJS Modules

```
1  module.exports.firstName = () ⇒ {
2    return 'Aaron';
3  }
4
5  module.exports.lastName = () ⇒ {
6    return 'Czichon';
7  }
8
9  module.exports.fullName = () ⇒ {
10   return `${this.firstName()} ${this.lastName()}`;
11 };
```

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

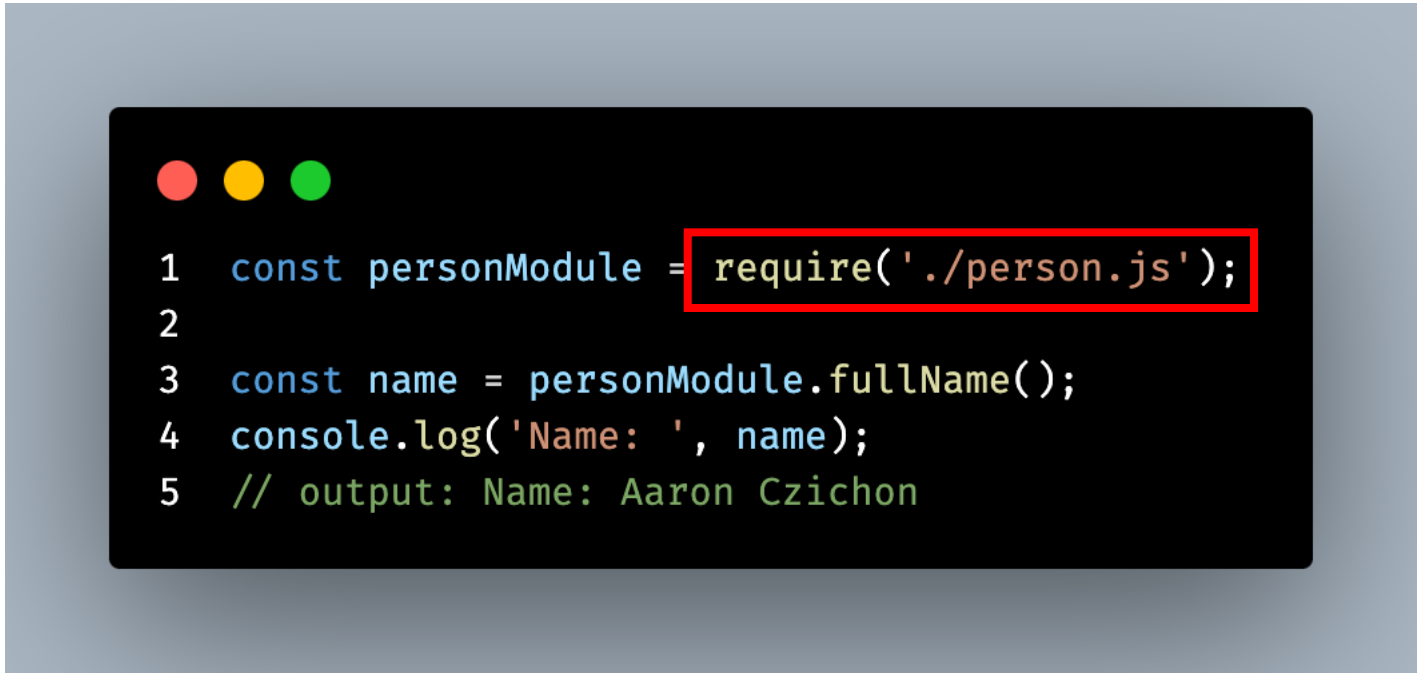
CommonJS Modules

```
1  module.exports.firstName = () => {
2    return 'Aaron';
3  }
4
5  module.exports.lastName = () => {
6    return 'Czichon';
7  }
8
9  module.exports.fullName = () => {
10   return `${this.firstName()} ${this.lastName()}`;
11 };
```

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

CommonJS Modules



```
1  const personModule = require('./person.js');
2
3  const name = personModule.fullName();
4  console.log('Name: ', name);
5  // output: Name: Aaron Czichon
```

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

CommonJS Modules

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

CommonJS Modules

ES Modules

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

```
1  export const firstName = () => {
2    return 'Aaron';
3  }
4
5  export const lastName = () => {
6    return 'Czichon';
7  }
8
9  export const fullName = () => {
10   return `${firstName()} ${lastName()}`;
11 }
```

ES Modules

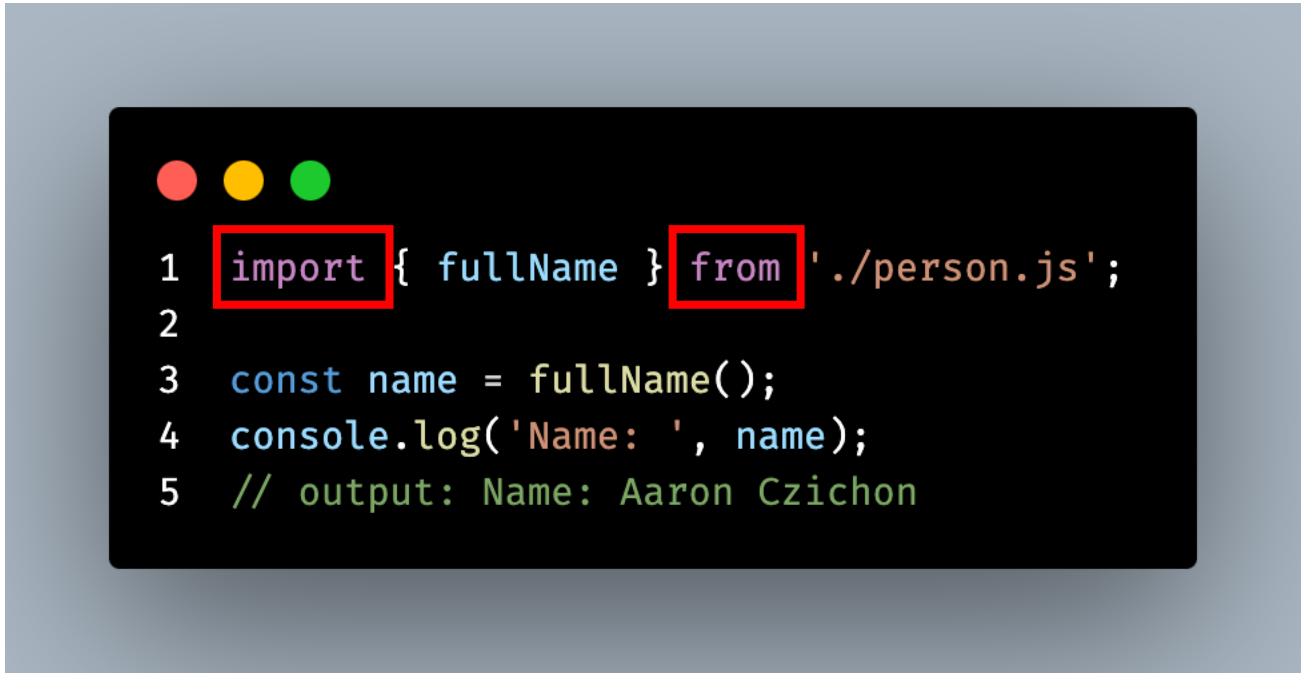
Module Types on JS Levels

- On Javascript Level there are also multiple types of modules

ES Modules

Module Types on JS Levels

- On Javascript Level there are also multiple types of modules



ES Modules

Module Types on JS Levels

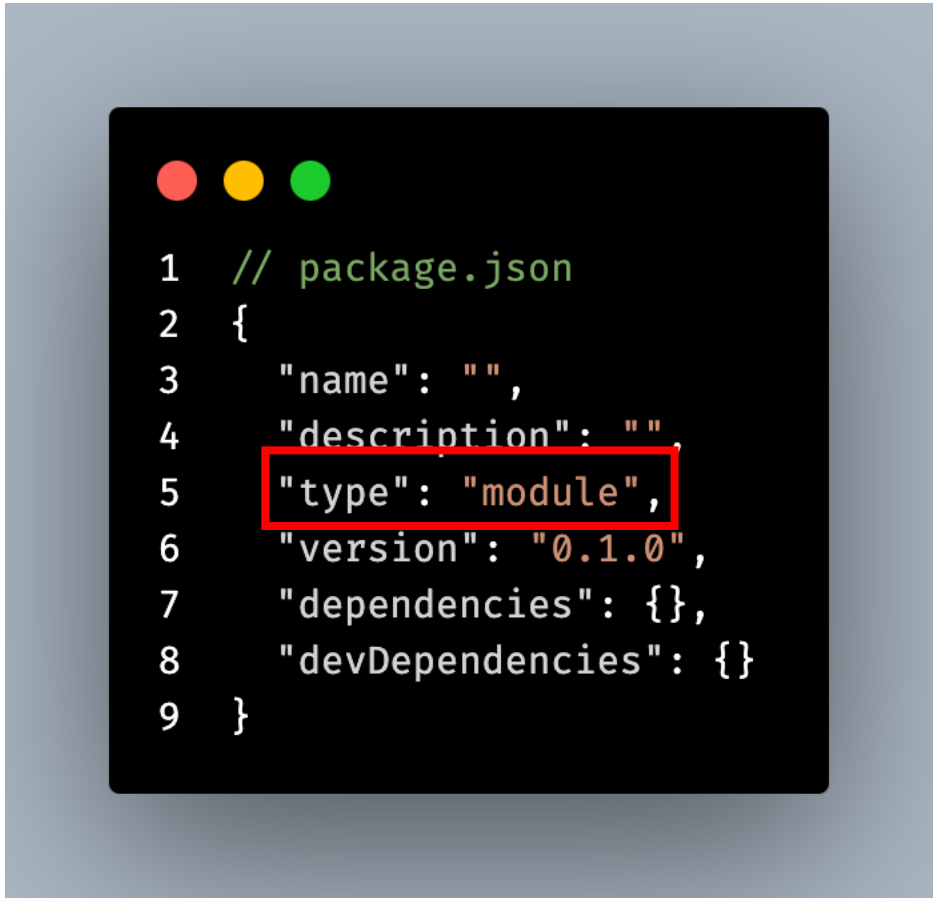
- NodeJS uses per default CommonJS Modules
- Every JS File is treated as a CommonJS Module on import
- You can force Node to import it a specific way by changing your file extensions:
  - .mjs -> NodeJS will treat this as an ESModule
  - .cjs -> NodeJS will treat this as a CommonJS Module

Module Types on JS Levels

- You can also change NodeJS default behavior

Module Types on JS Levels

- You can also change NodeJS default behavior



Task 2

Create your first ESModule and import it

Task 2.1

CommonJS Module in ESModule App

NodeJS Path Module

Path Module

- Handlings paths on all platforms (Windows and POSIX)
- Resolving paths
- Building paths
- Getting relative paths
- Checking for absolute paths
- And more!

Path Module - Usage

Path Module - Usage

Path Module - Usage

Event Handling in Node

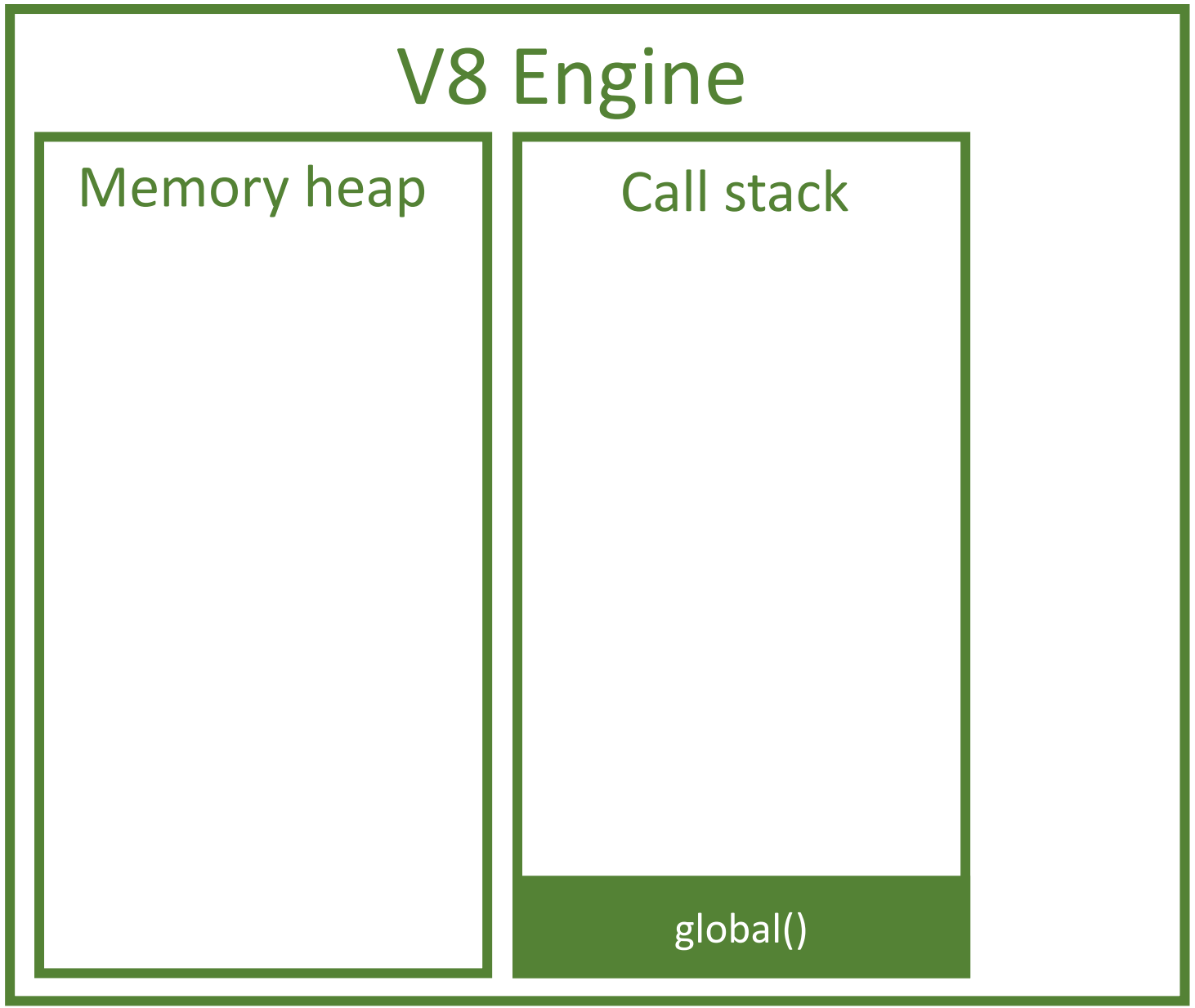
Event Handling

- Per default Javascript is single threaded and synchronous
- Node uses Libuv for asynchronous executions

Synchronous Execution

Event Handling

```
1  console.log('Hello');
2  console.log('World');
3  console.log('NodeJS!');
```



Event Handling

Output Console

Hello

V8 Engine

Memory heap

Call stack

console.log('Hello')

global()

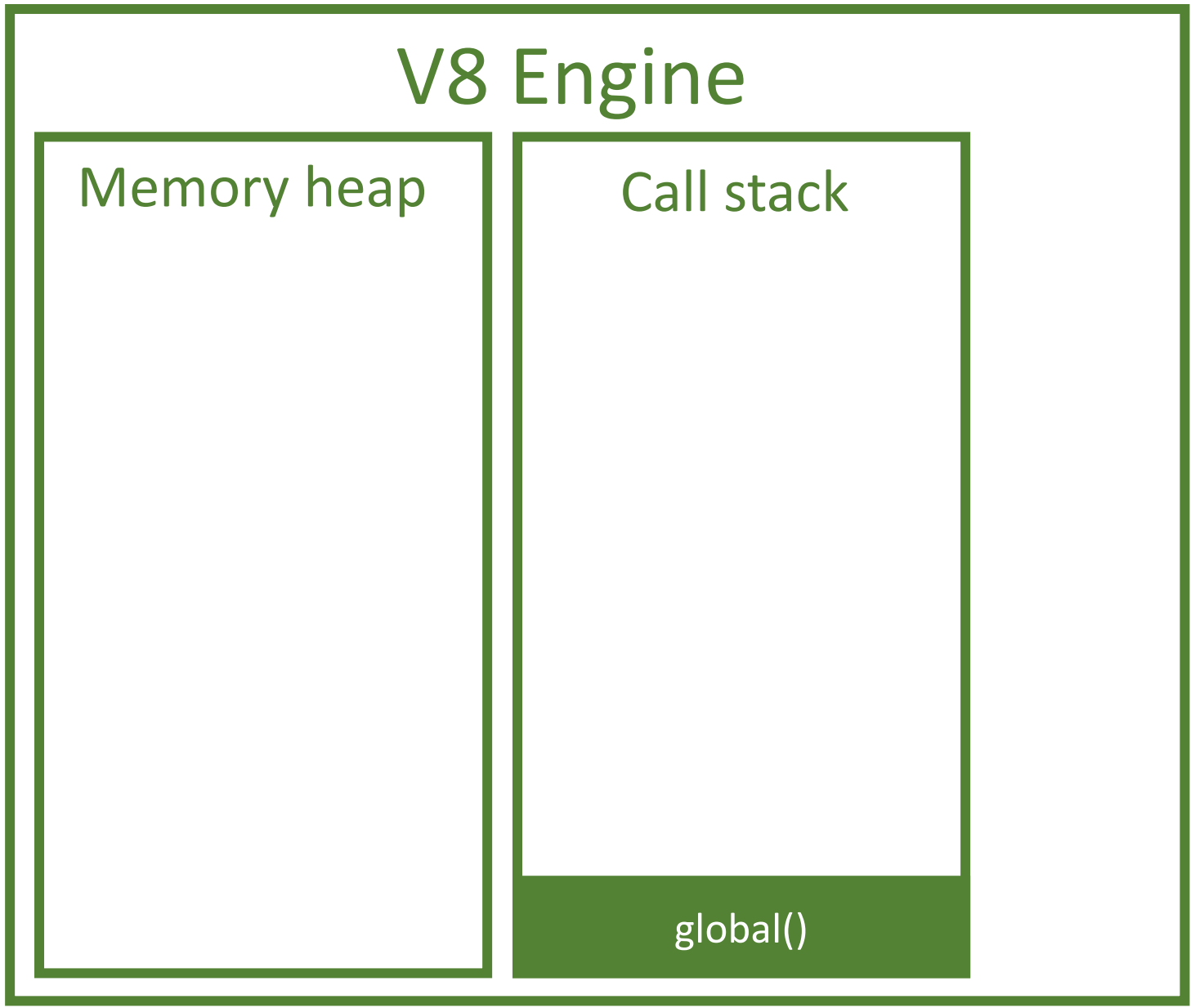
1ms

Event Handling

```
1  console.log('Hello');
2  console.log('World');
3  console.log('NodeJS!');
```

Output Console

Hello



Event Handling

Output Console

Hello

World

V8 Engine

Memory heap

Call stack

console.log('World')

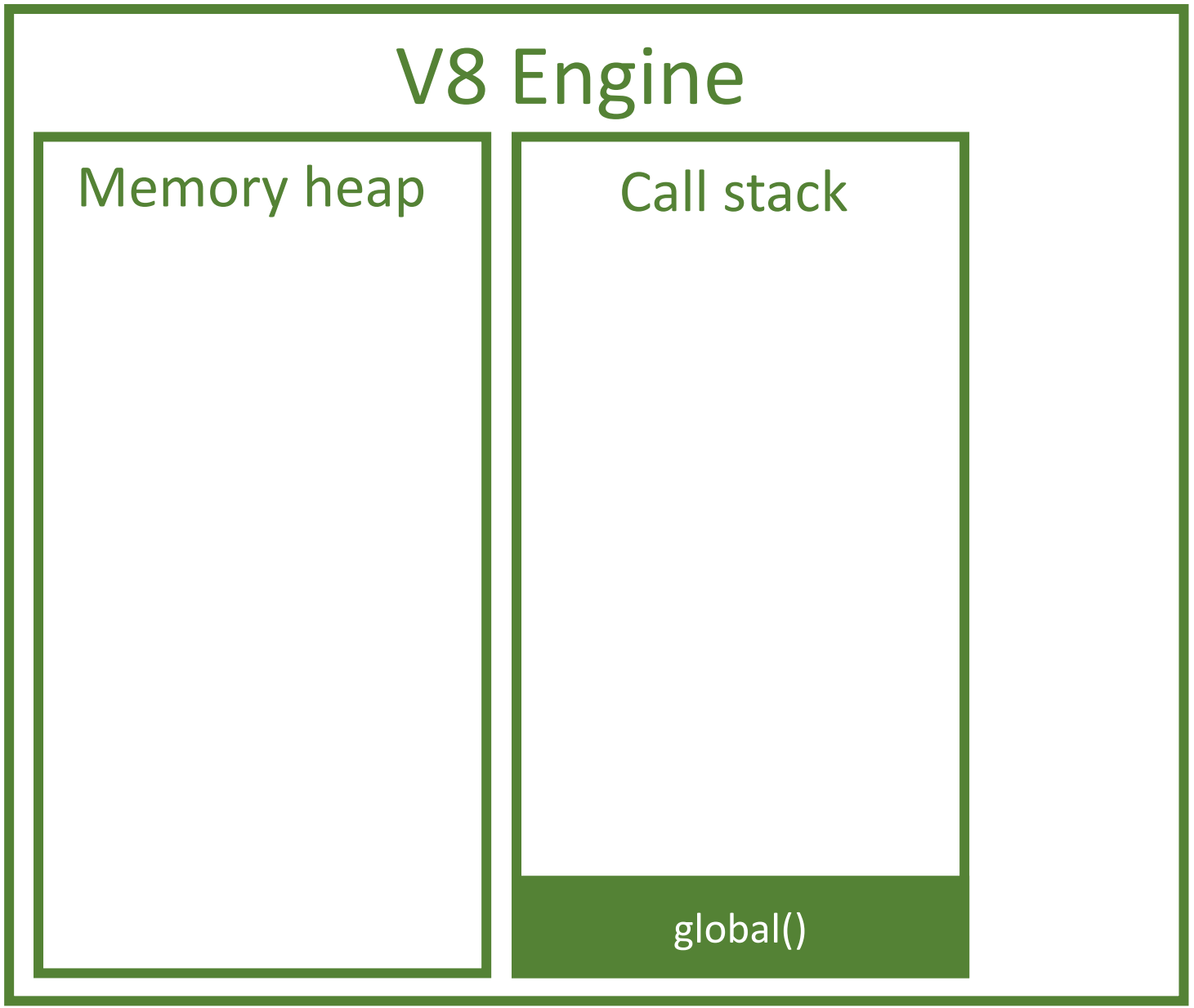
global()

2ms

Event Handling

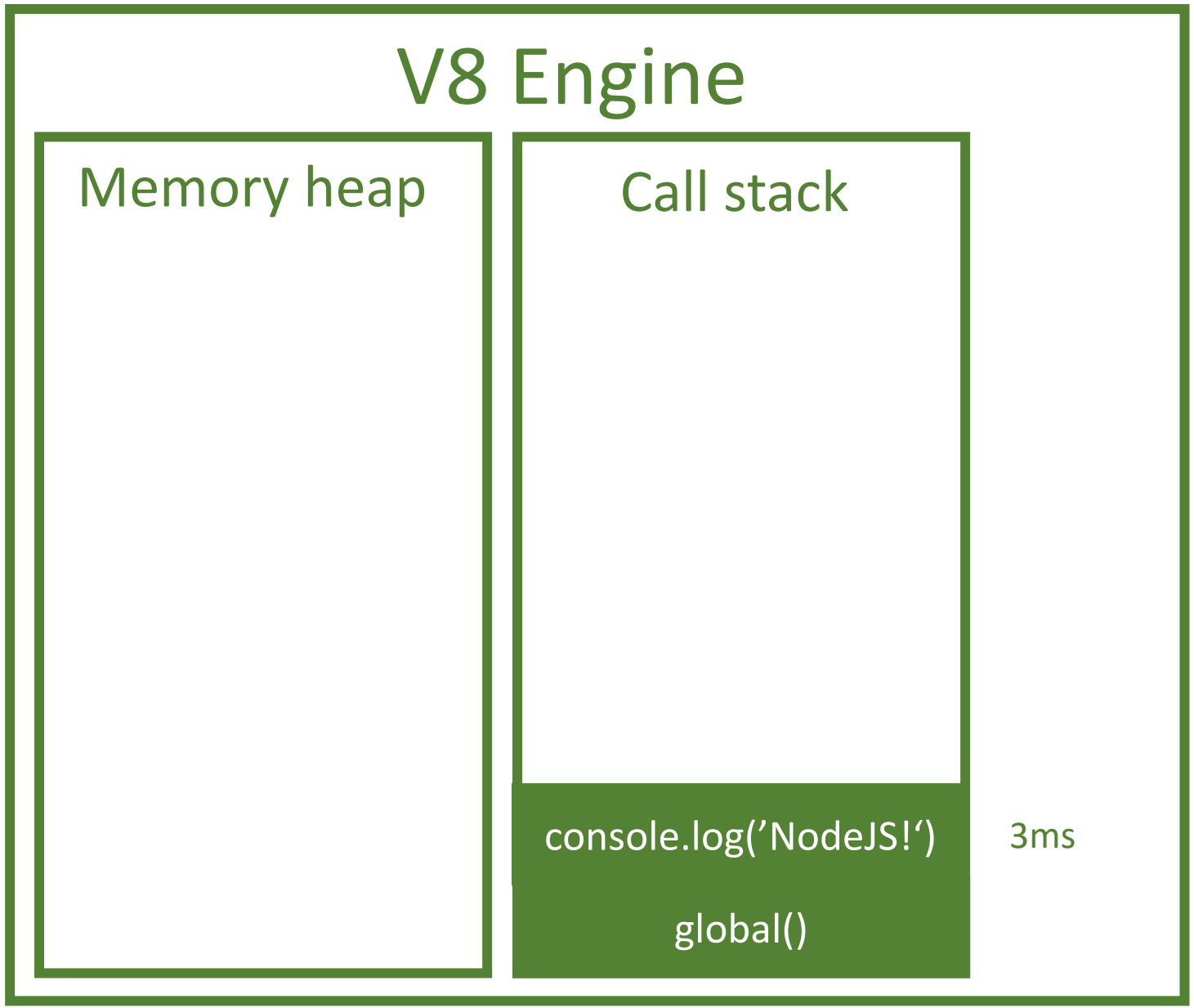
```
1  console.log('Hello');
2  console.log('World');
3  console.log('NodeJS!');
```

Output Console
Hello
World



Event Handling

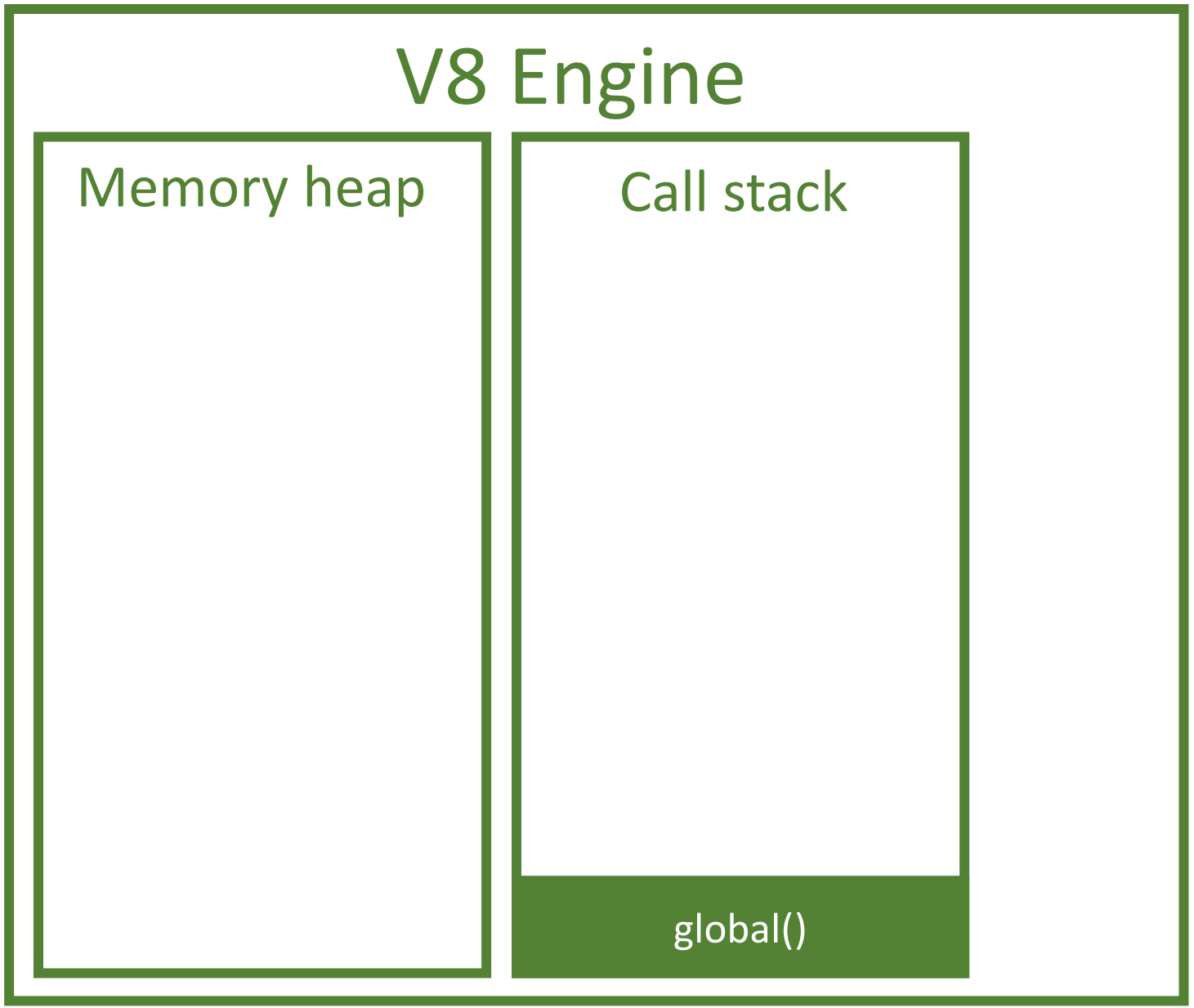
Output Console
Hello
World
NodeJS!



Event Handling

```
1  console.log('Hello');
2  console.log('World');
3  console.log('NodeJS!');
```

Output Console
Hello
World
NodeJS!



Event Handling

Output Console
Hello
World
NodeJS!

V8 Engine

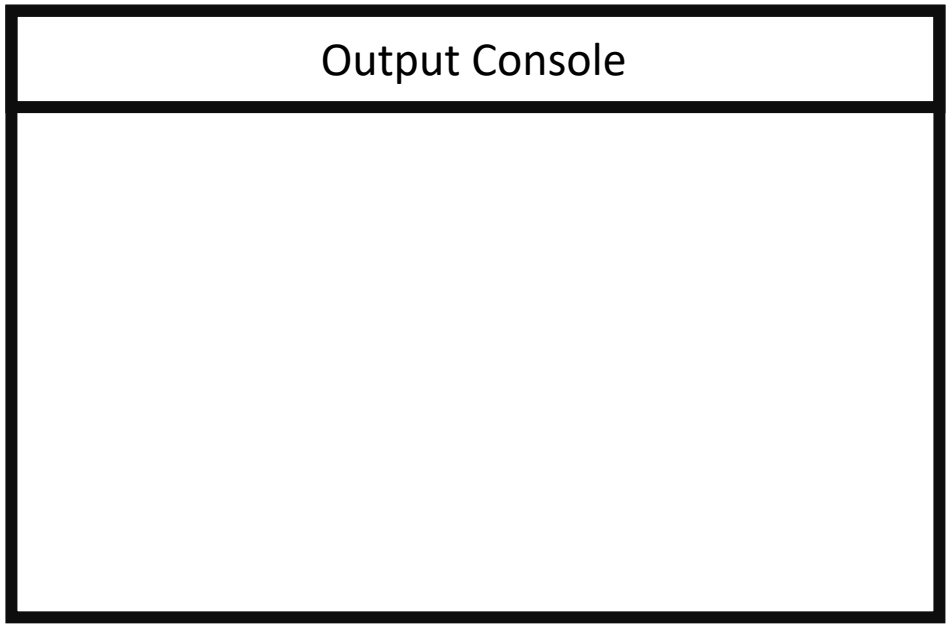
Memory heap

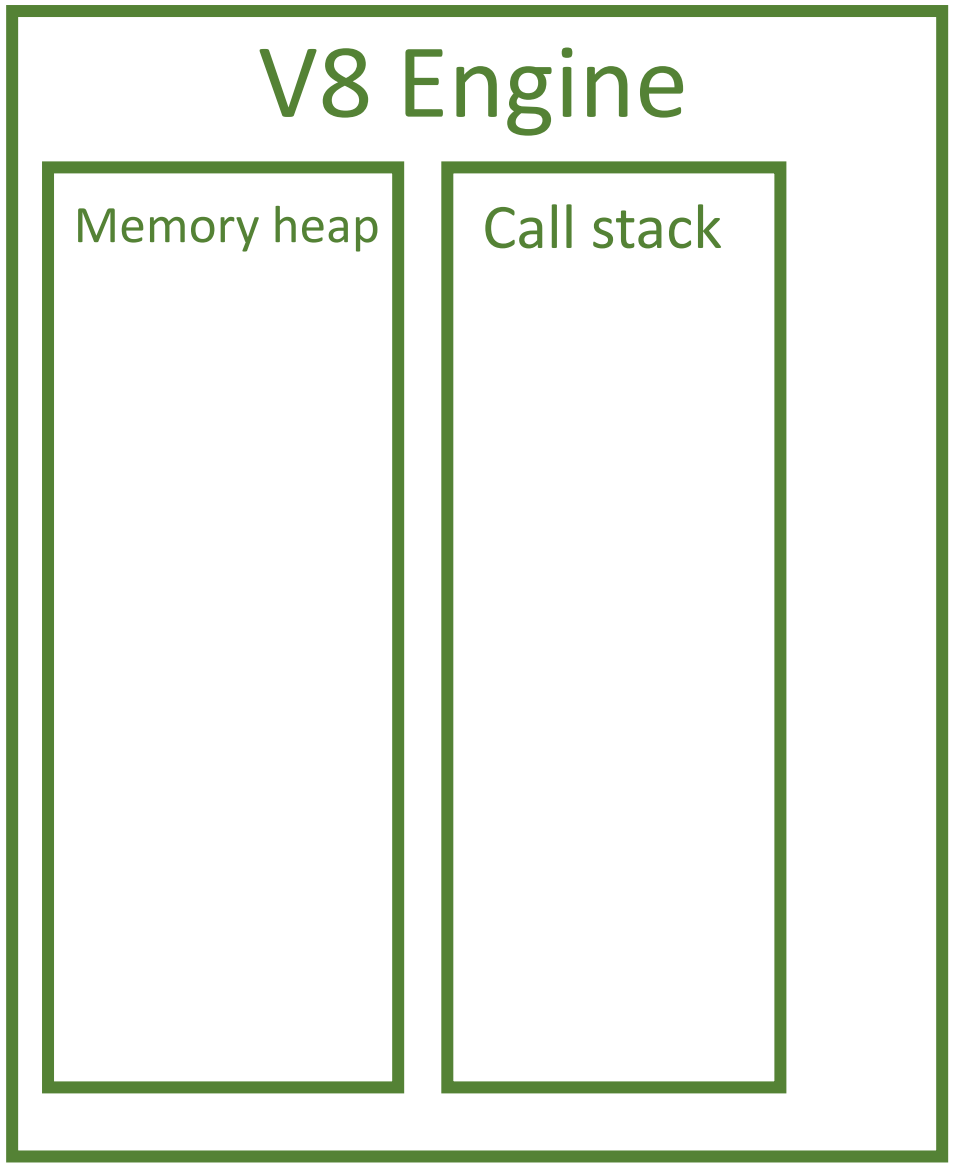
Call stack

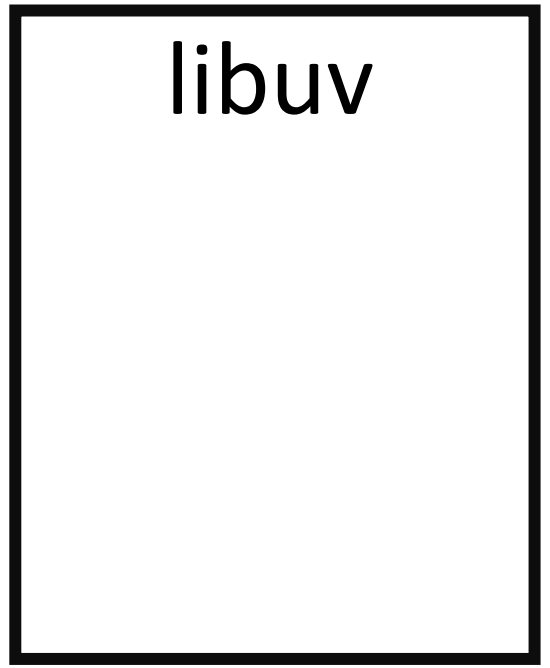
Asynchronous Execution

Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

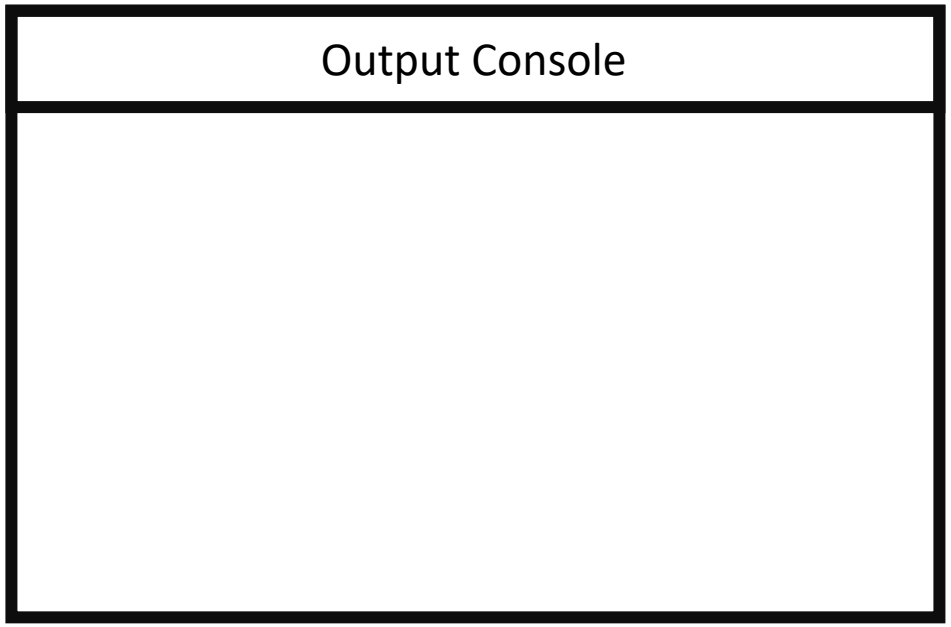


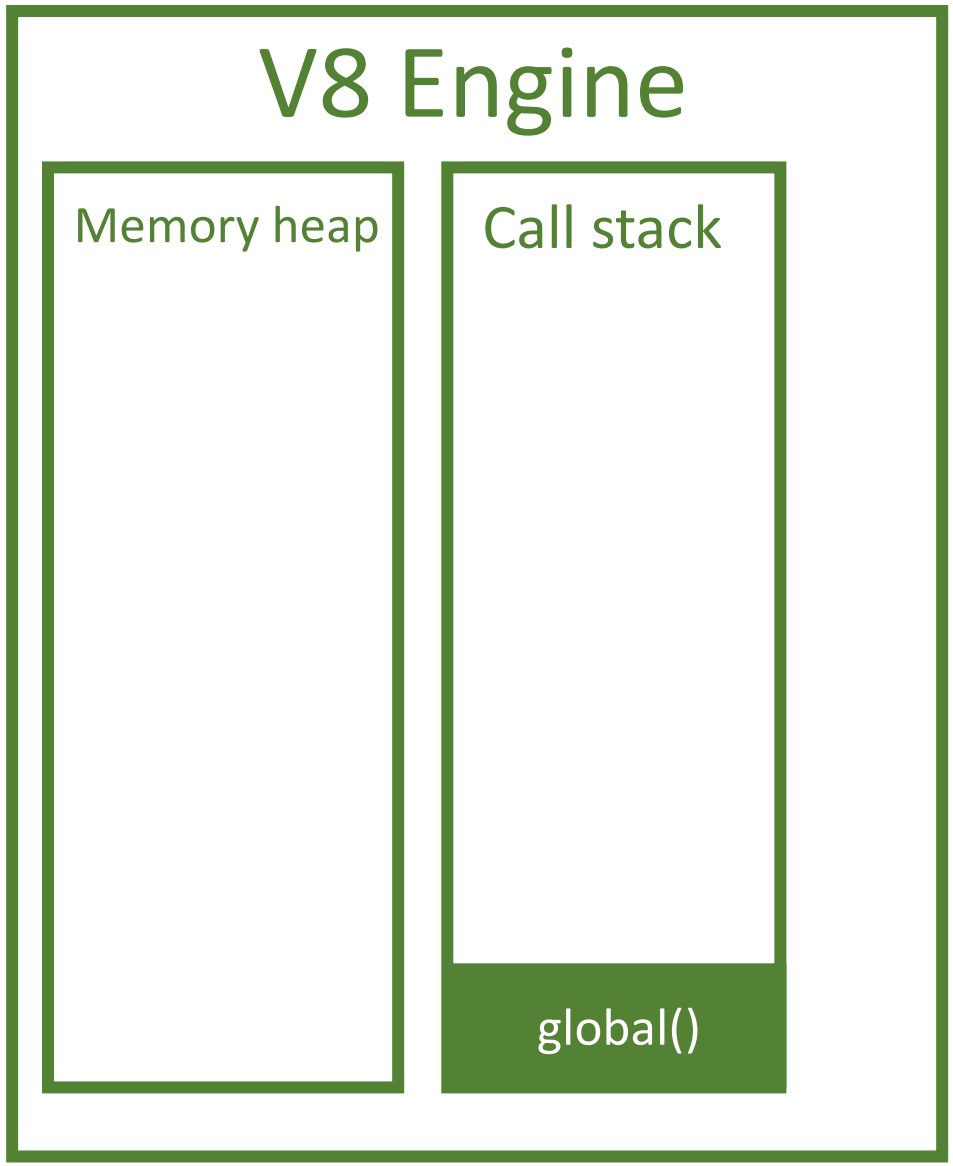


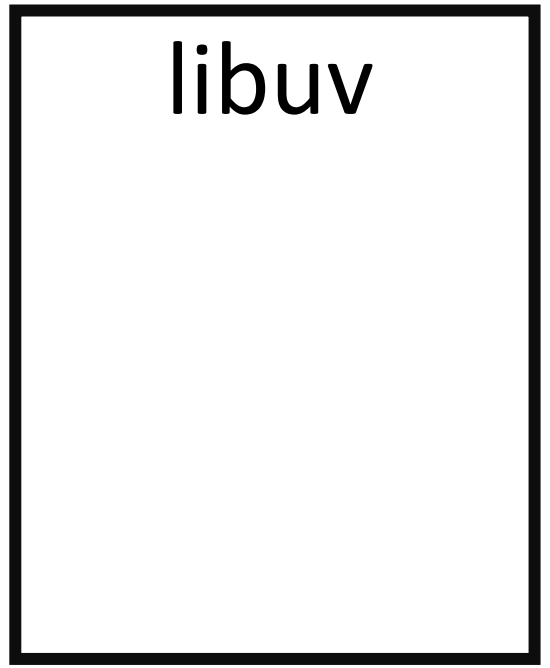


Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

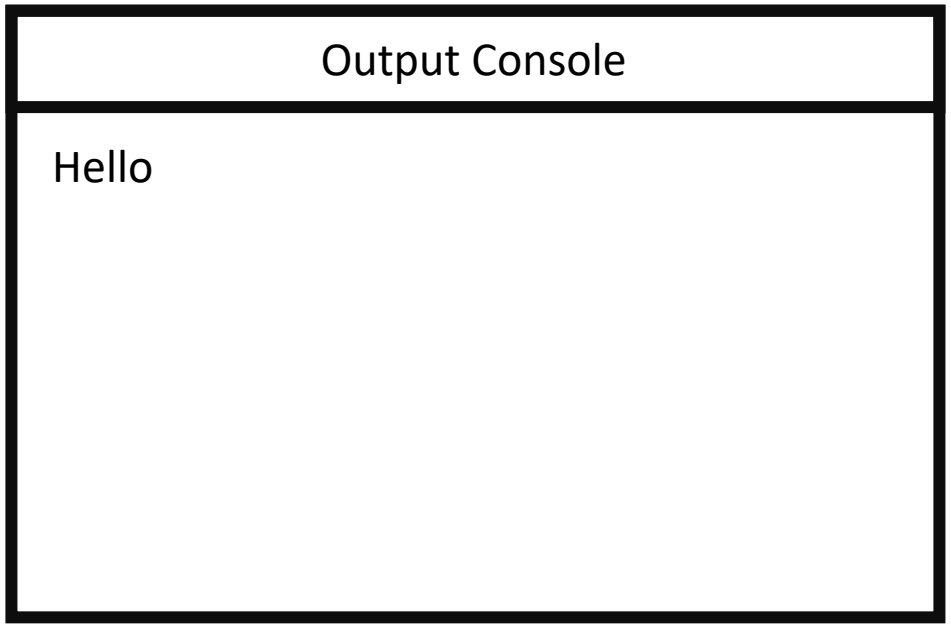


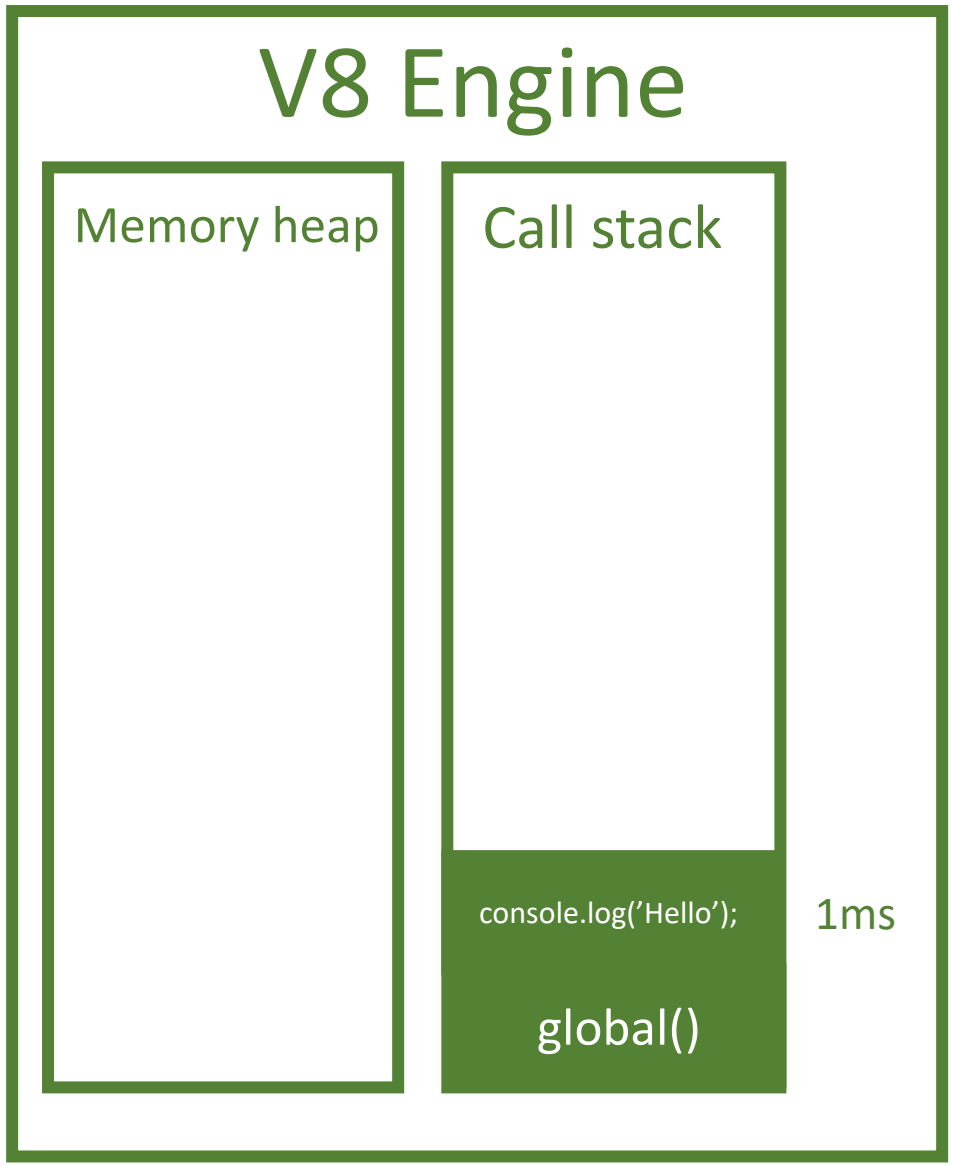


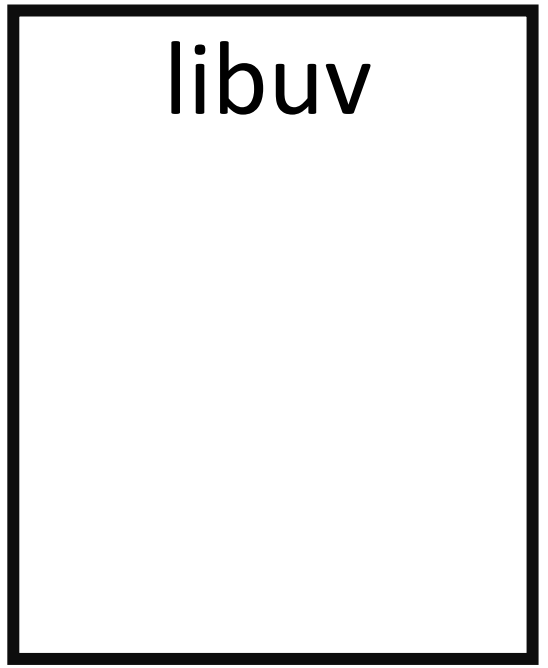


Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

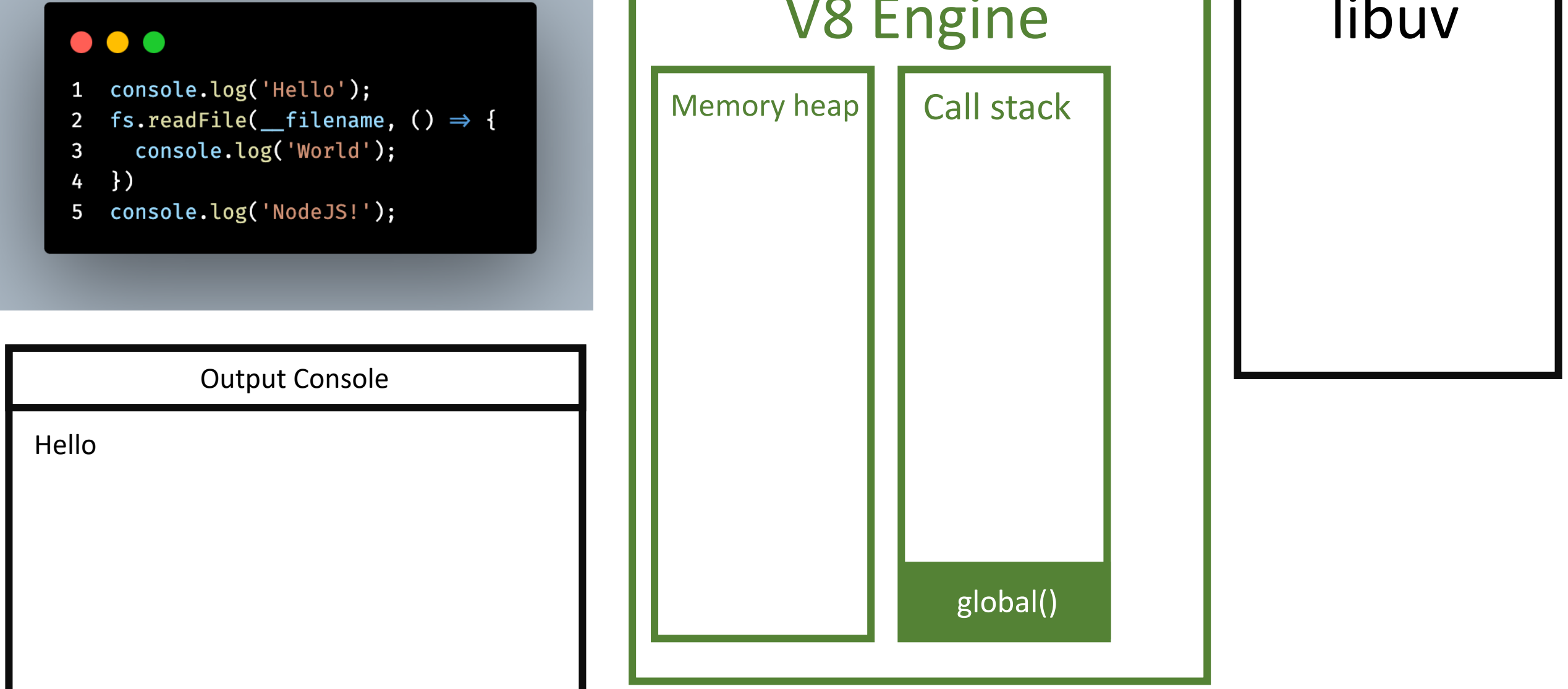






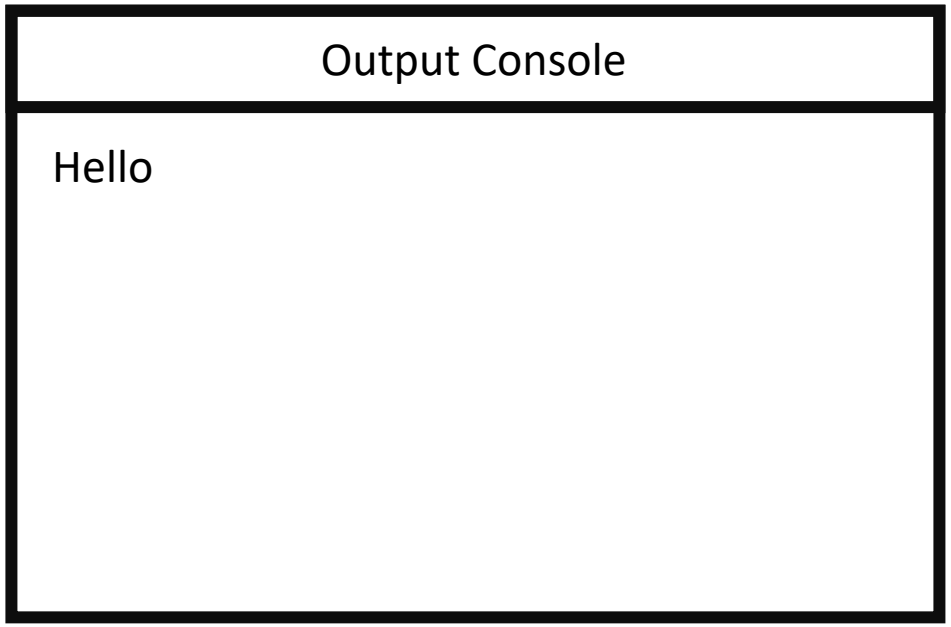
Event Handling – Asynchronous Execution

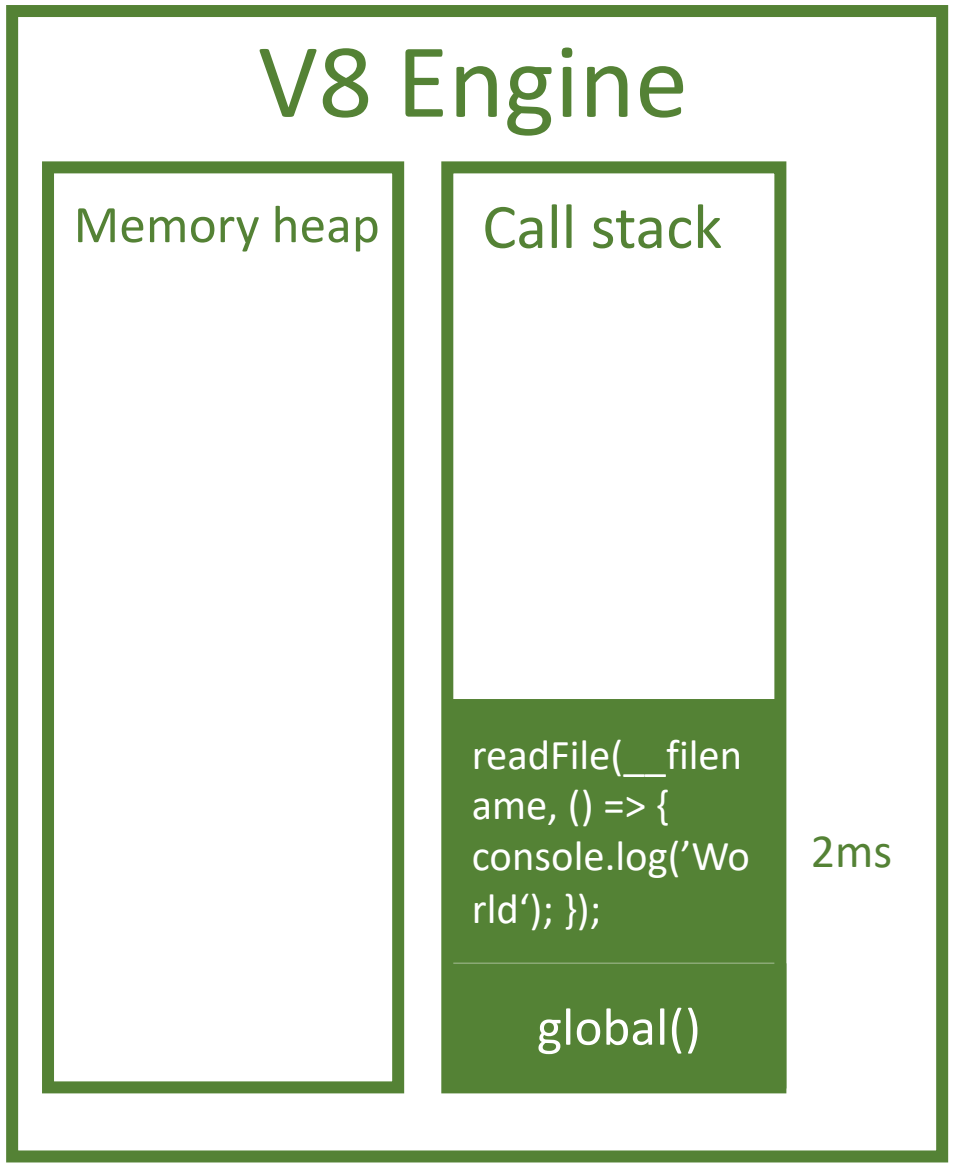
```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

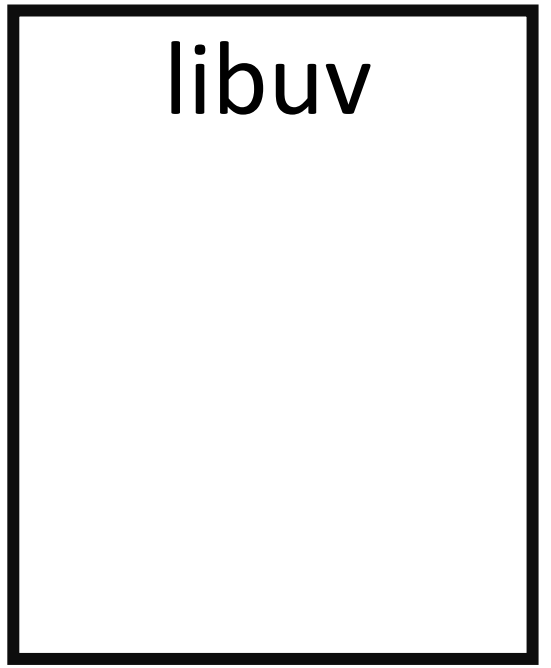


Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

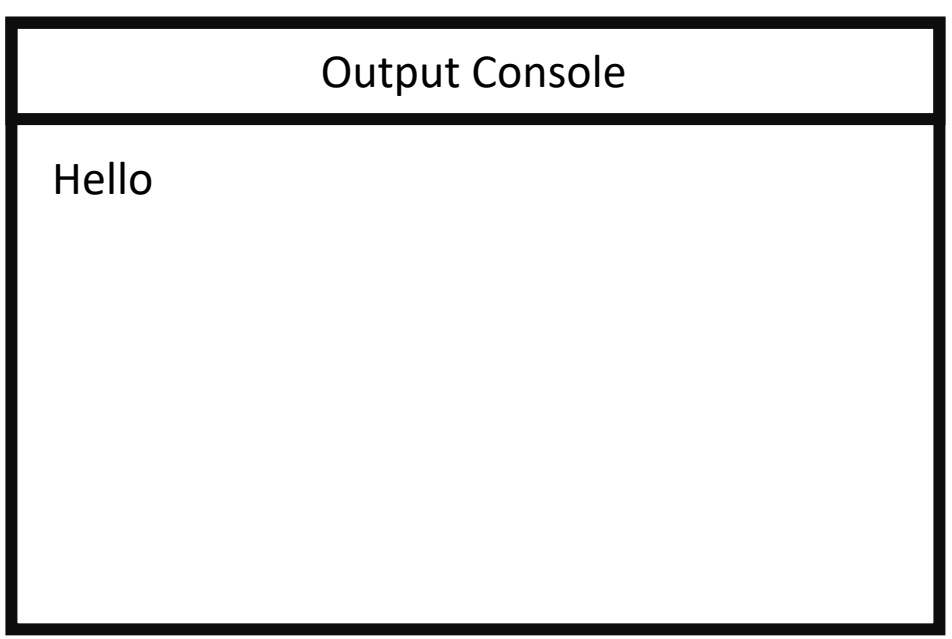


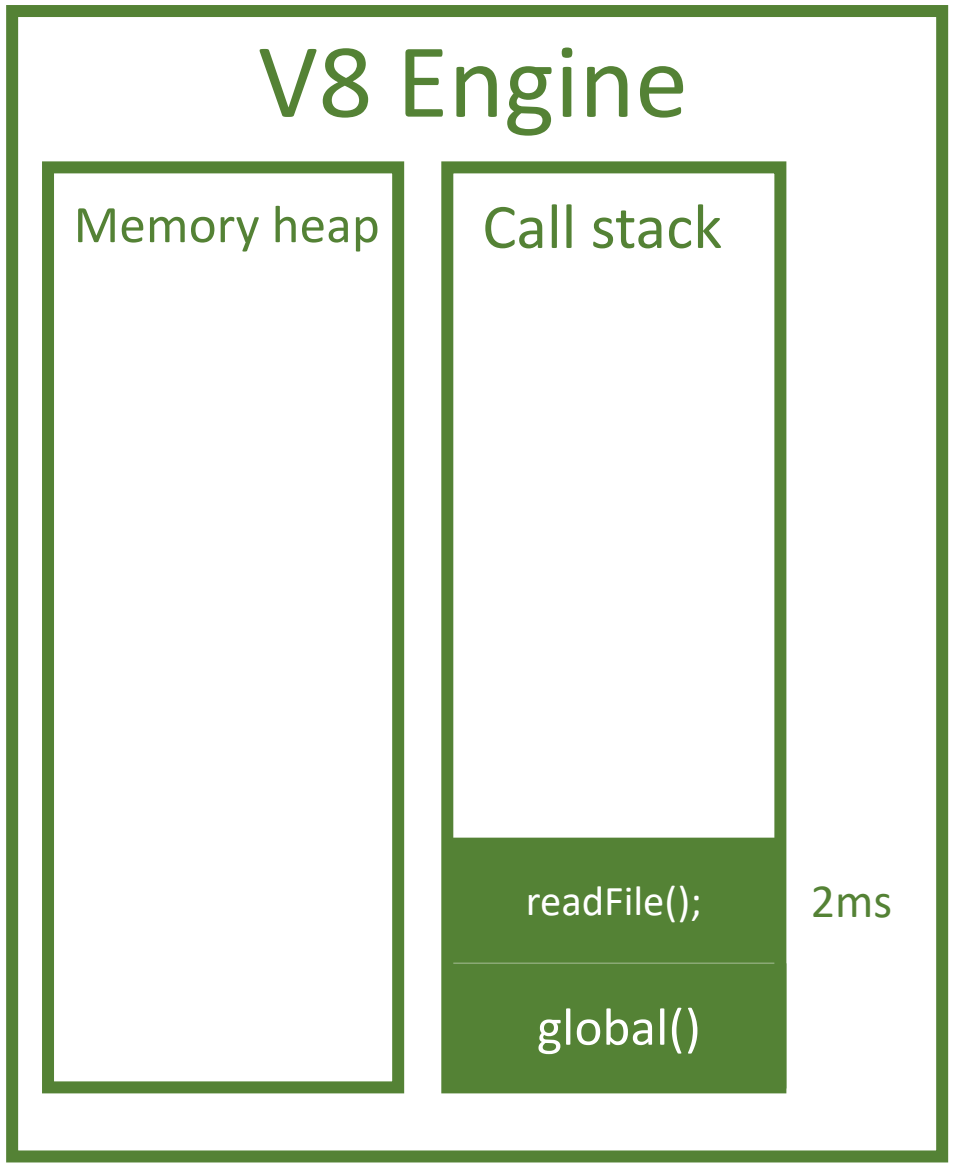


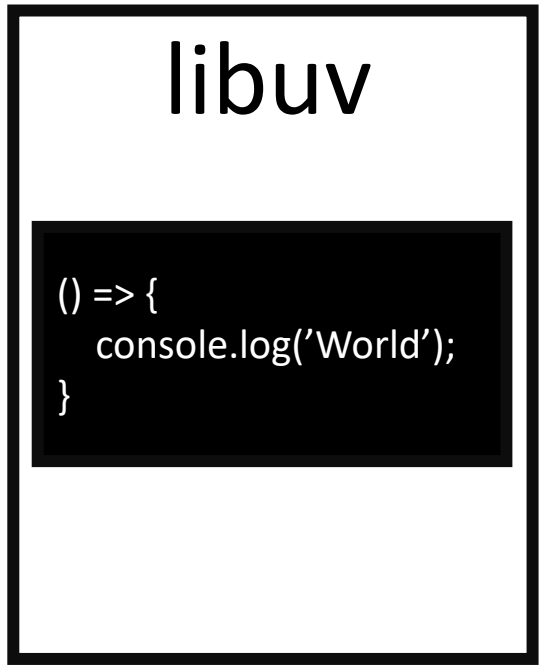


Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

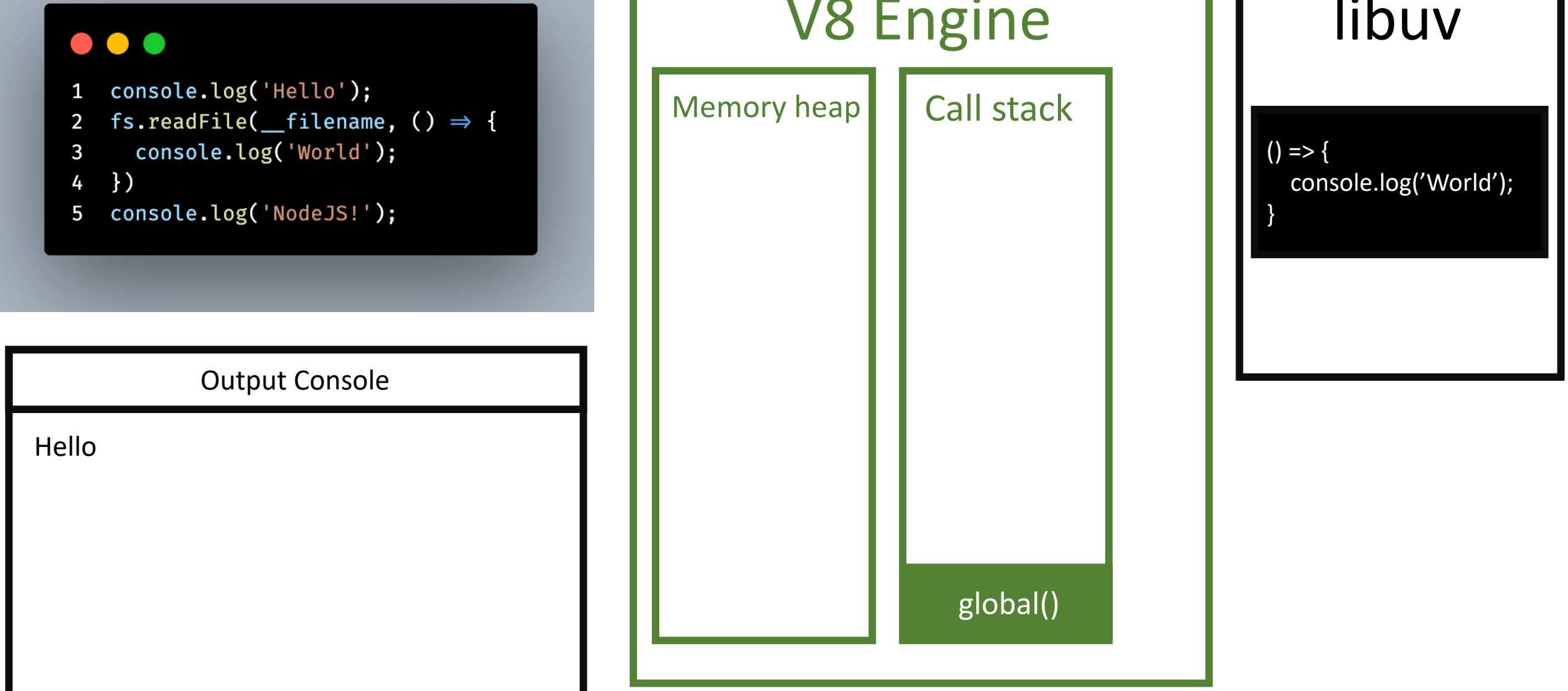






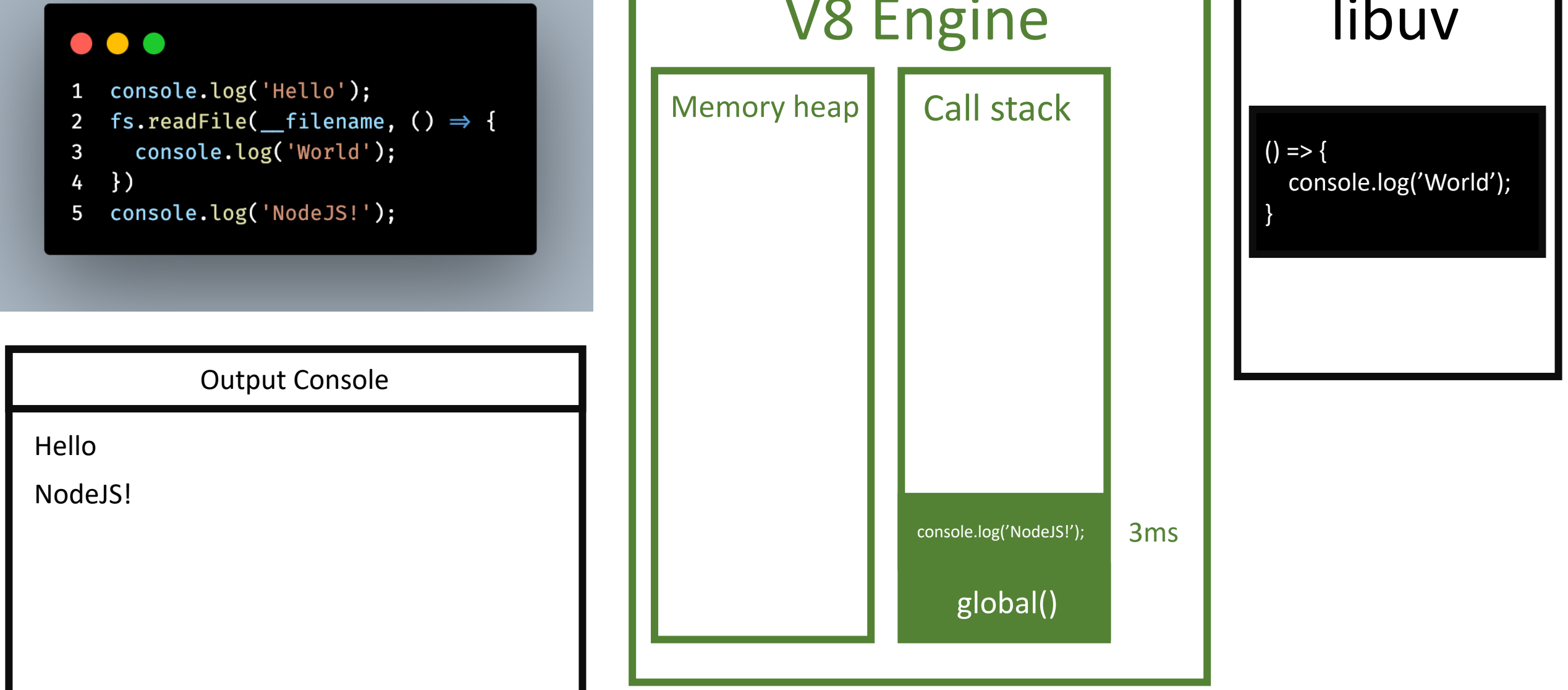
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



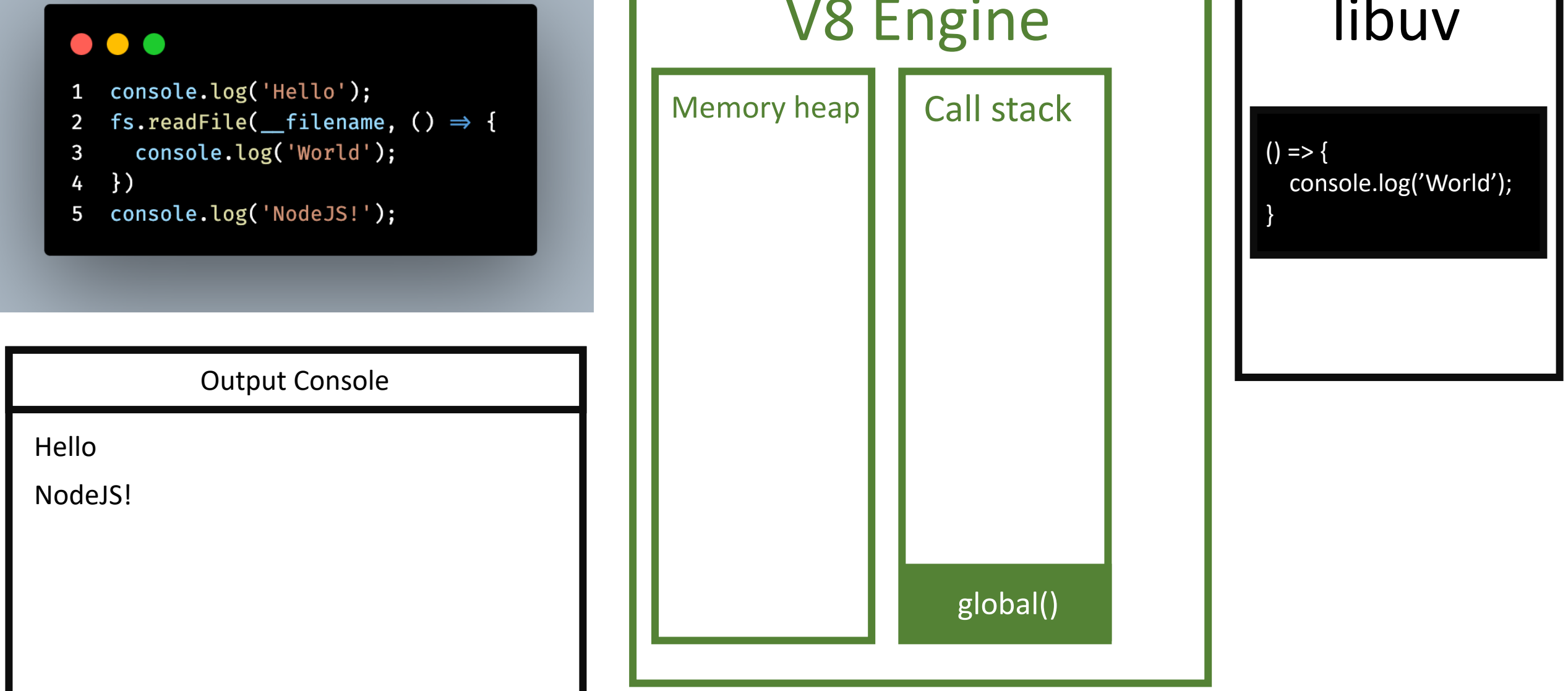
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



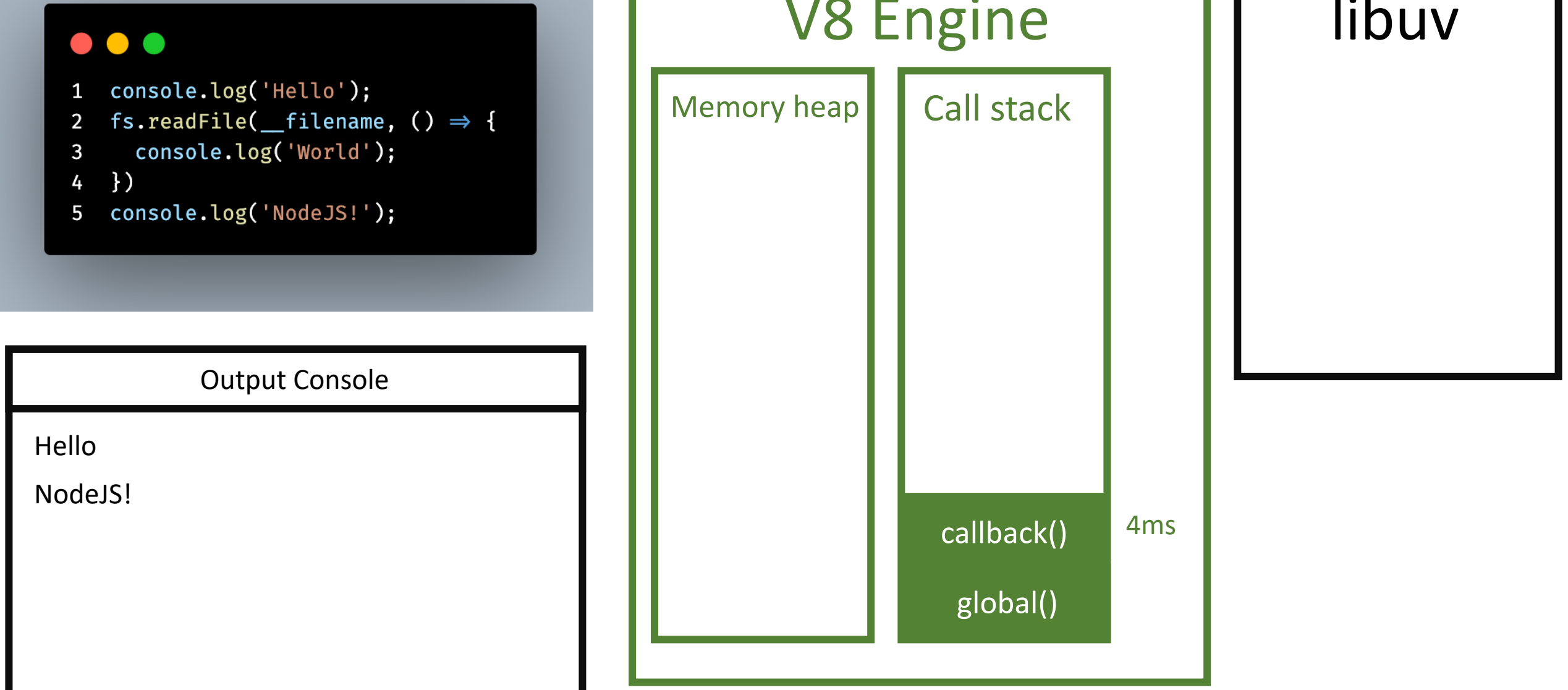
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



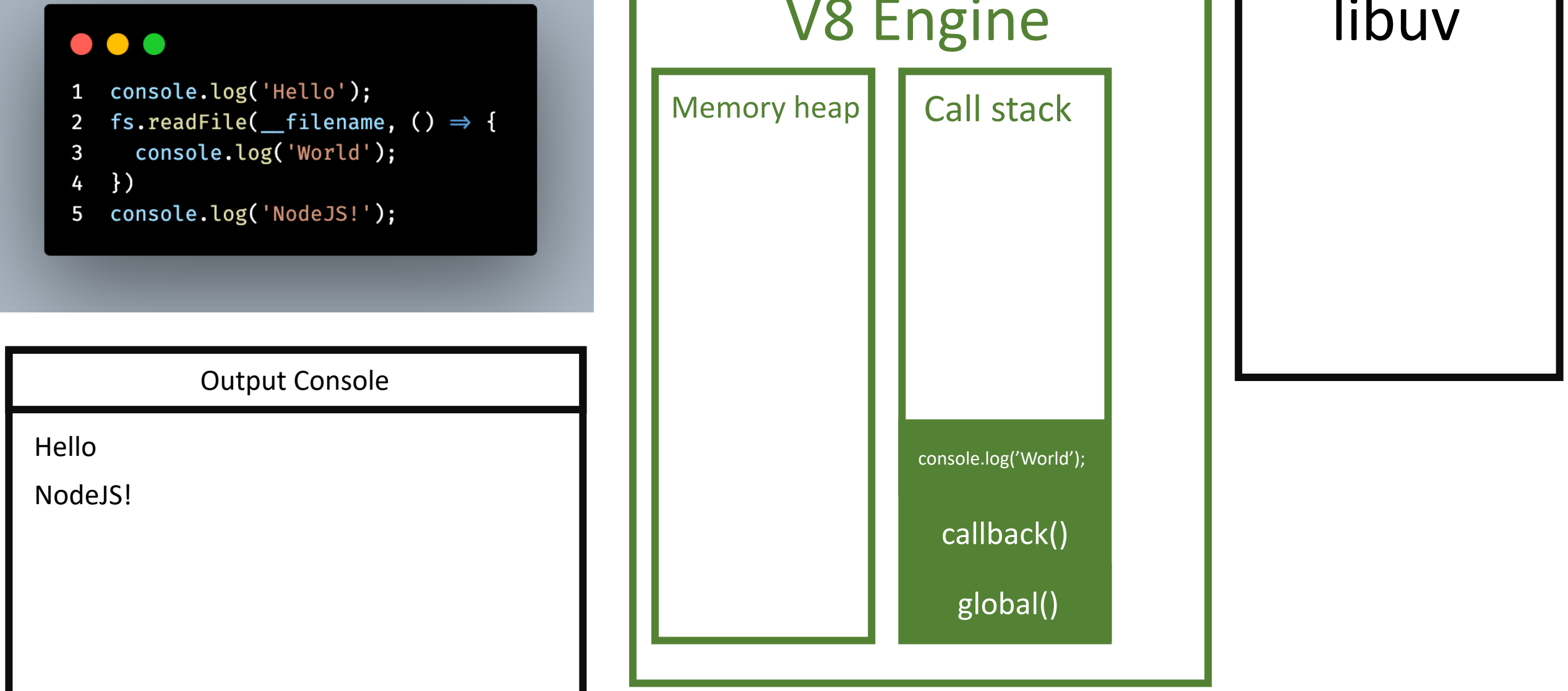
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



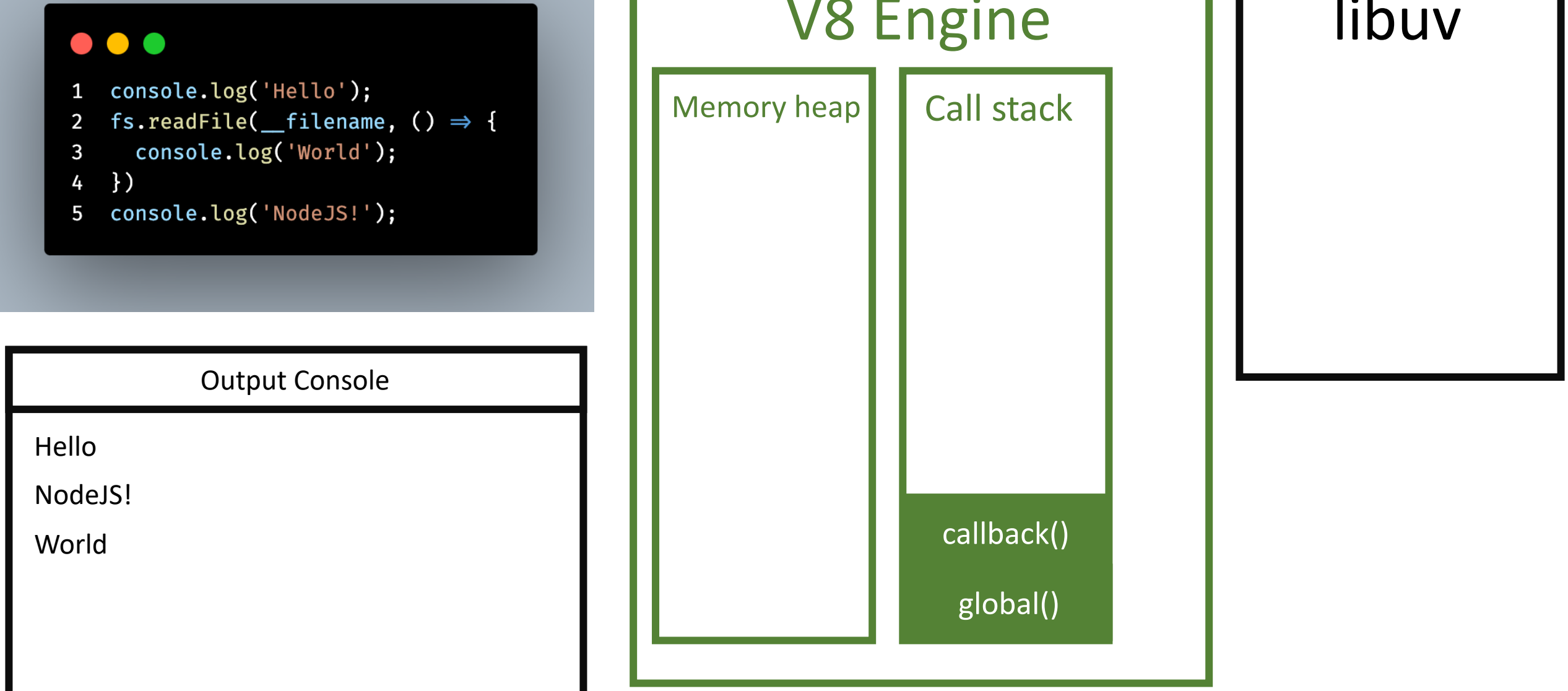
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



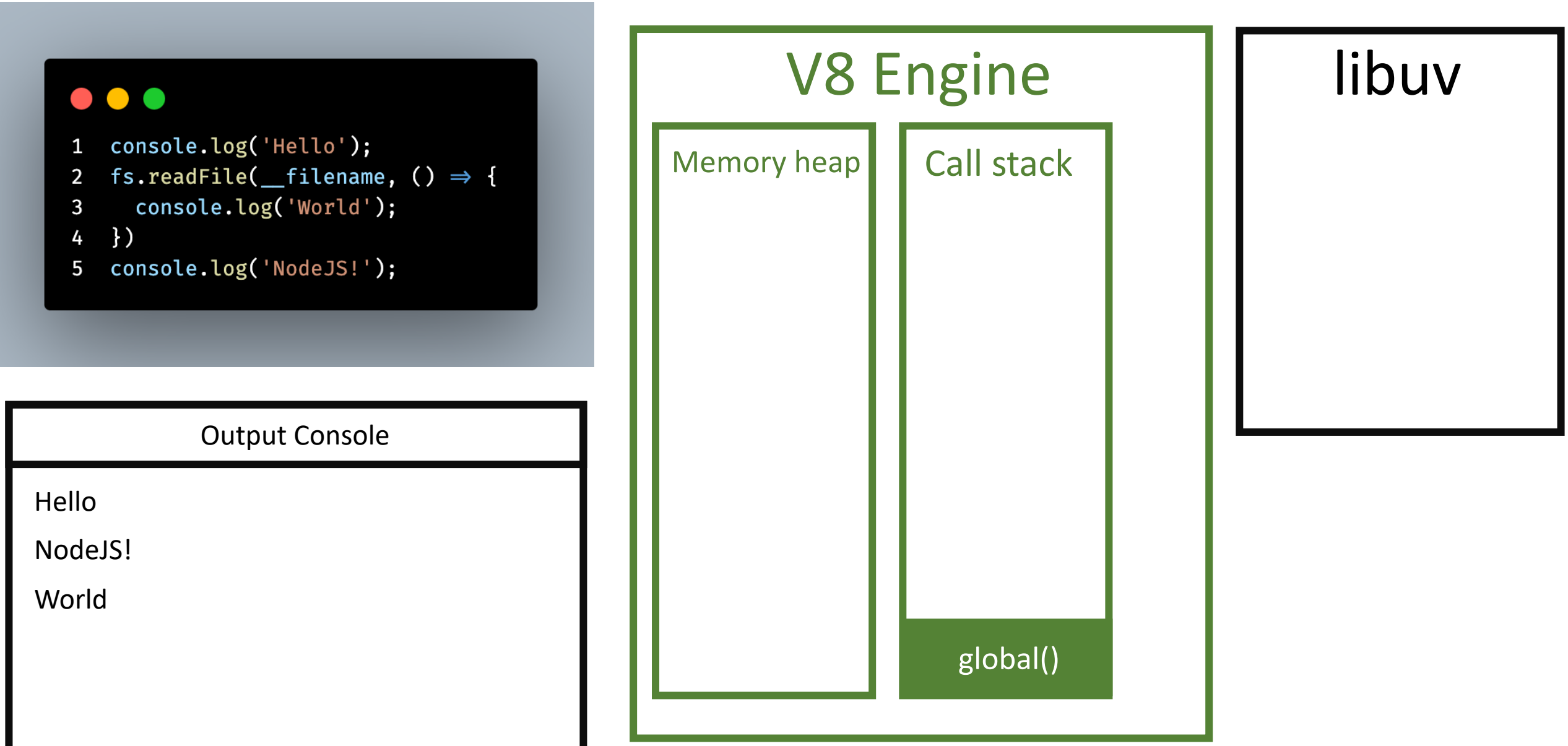
Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```



Event Handling – Asynchronous Execution

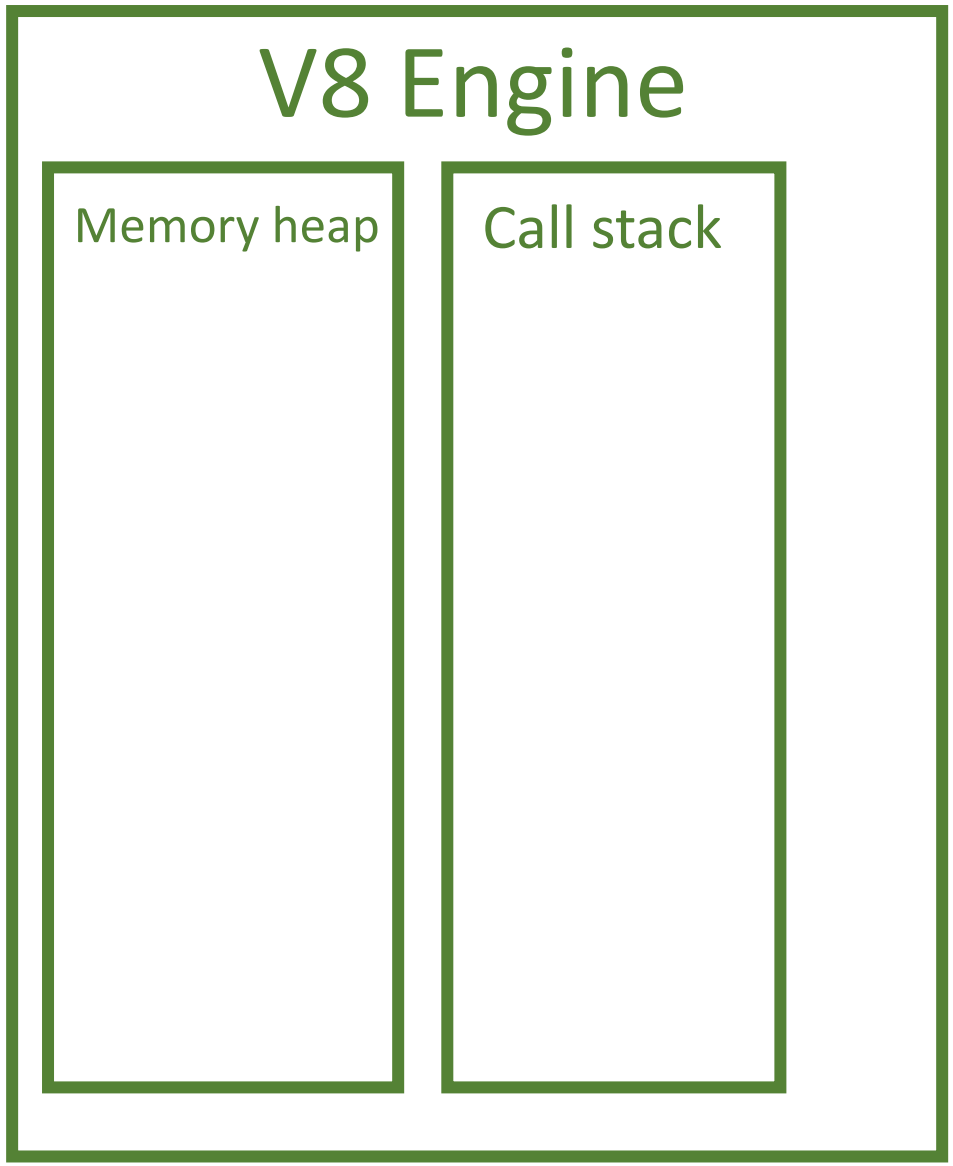
```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

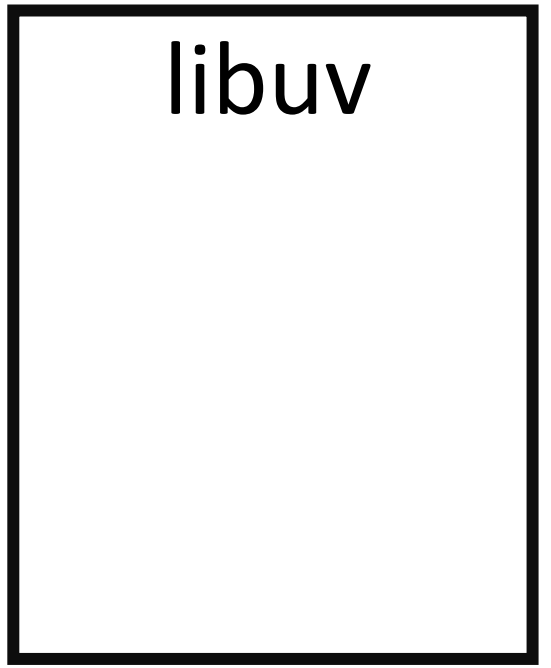


Event Handling – Asynchronous Execution

```
1  console.log('Hello');
2  fs.readFile(__filename, () ⇒ {
3    console.log('World');
4  })
5  console.log('NodeJS!');
```

Output Console
Hello
NodeJS!
World





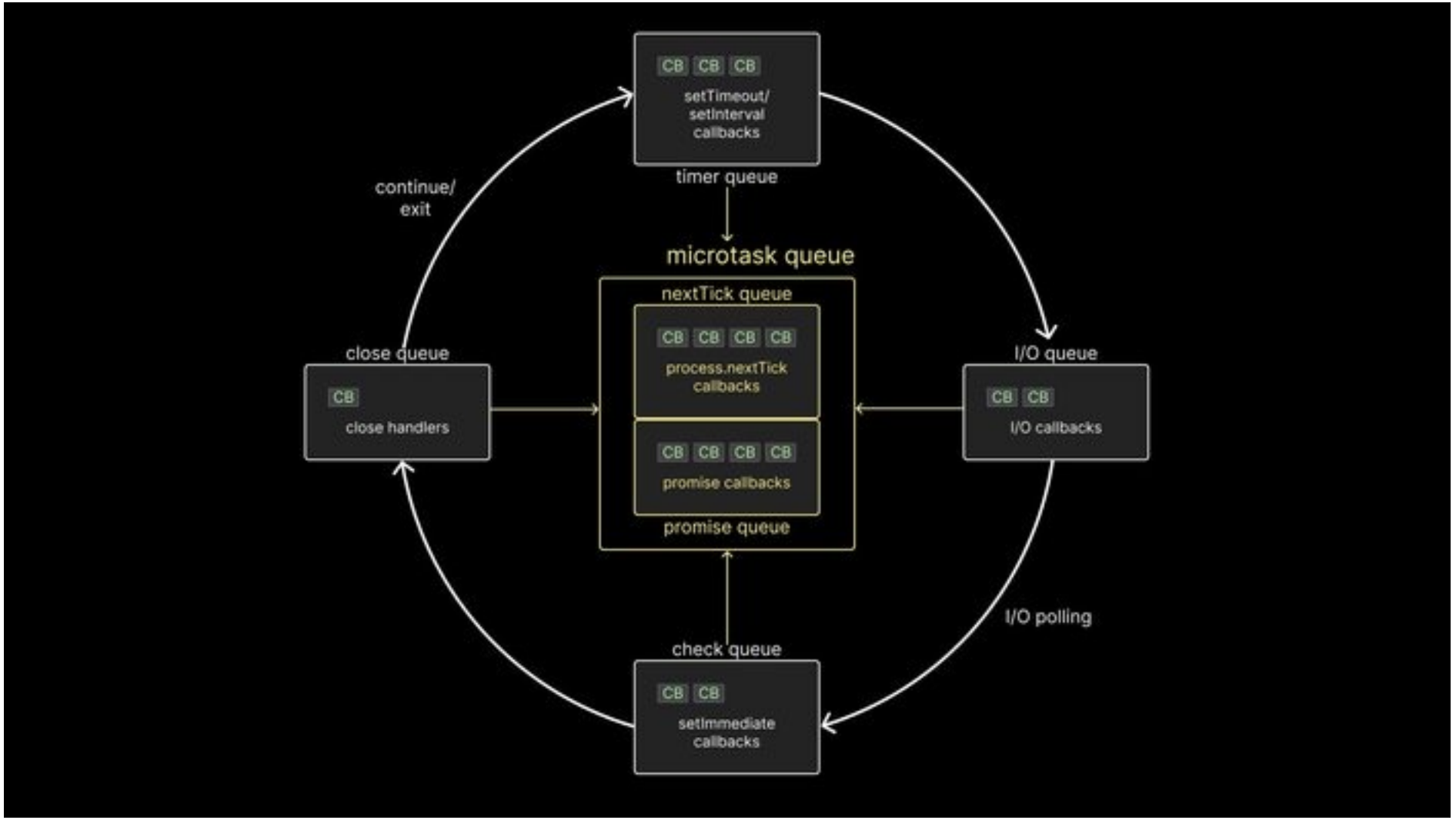
Things we should keep in mind

- When a task is completed in libuv, when does NodeJS decide to run the callback function on the call stack?
- How does async functions like setTimeout or setInterval work?
- If two functions like setTimeout and readFile finishing at the same time, which runs first on the call stack?

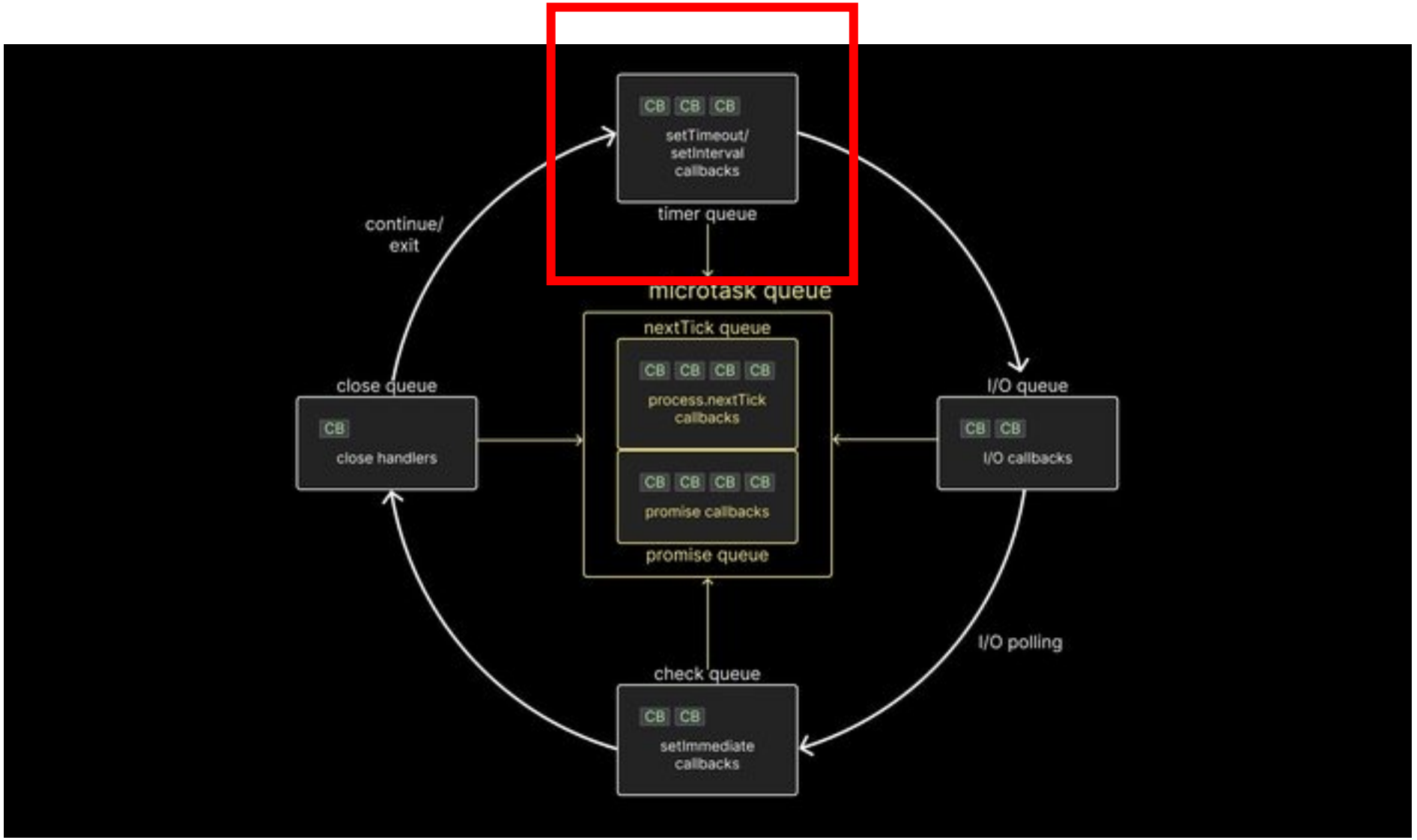
Event Loop

- To solve these problems: Event Loop
- A program which is part of libuv
- Basically a design pattern to orchestrate execution of code

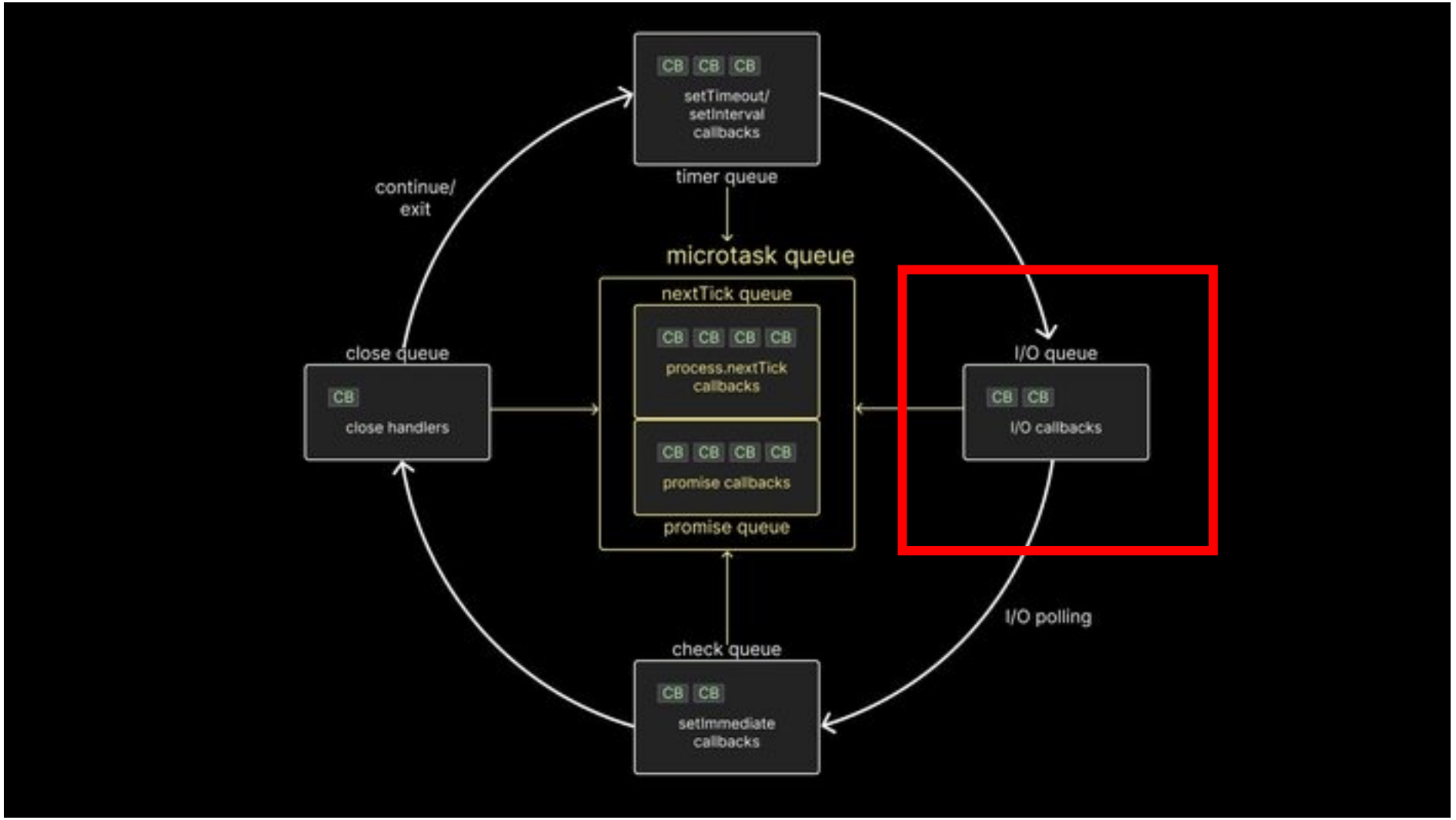
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

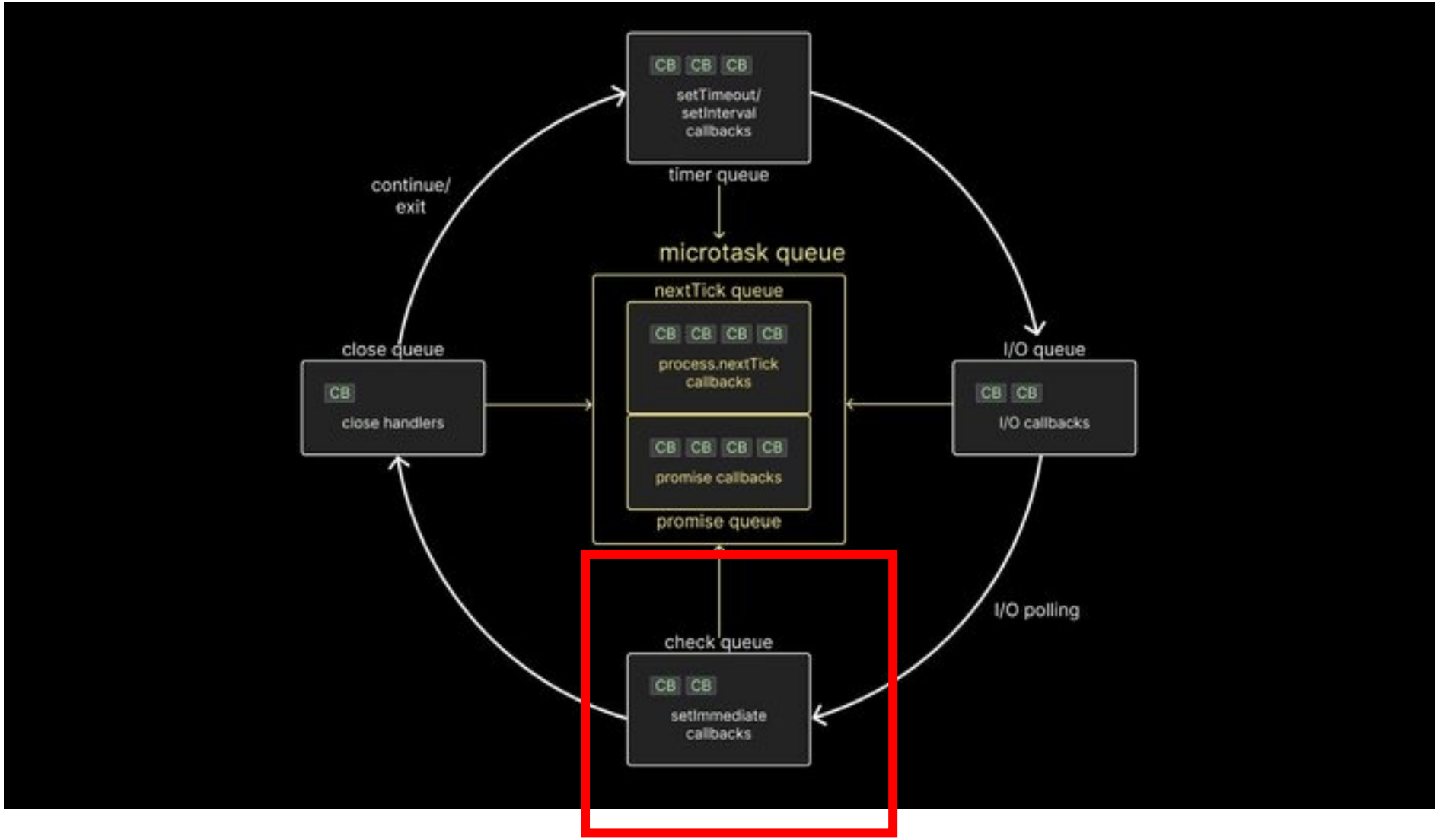
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

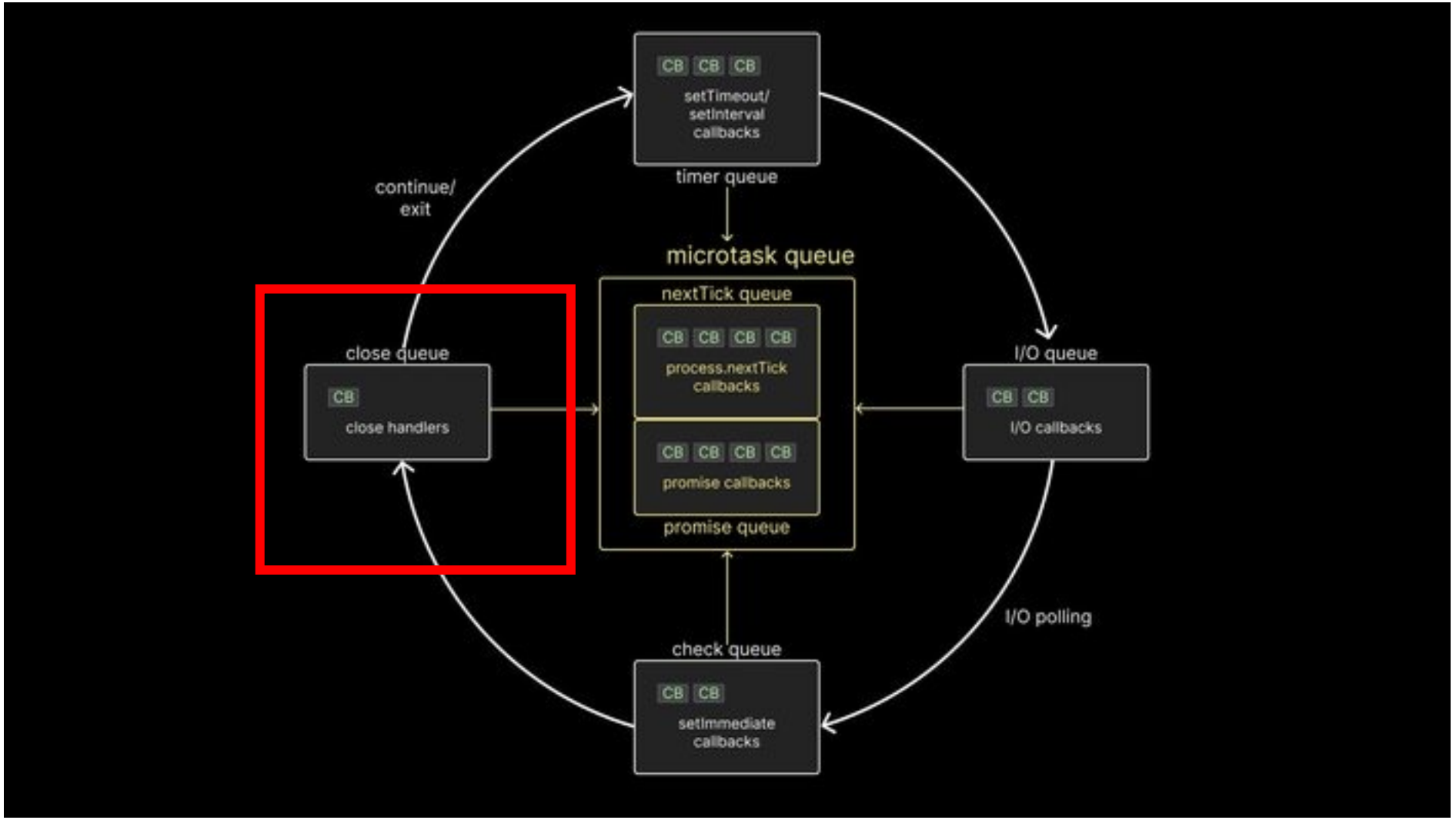
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

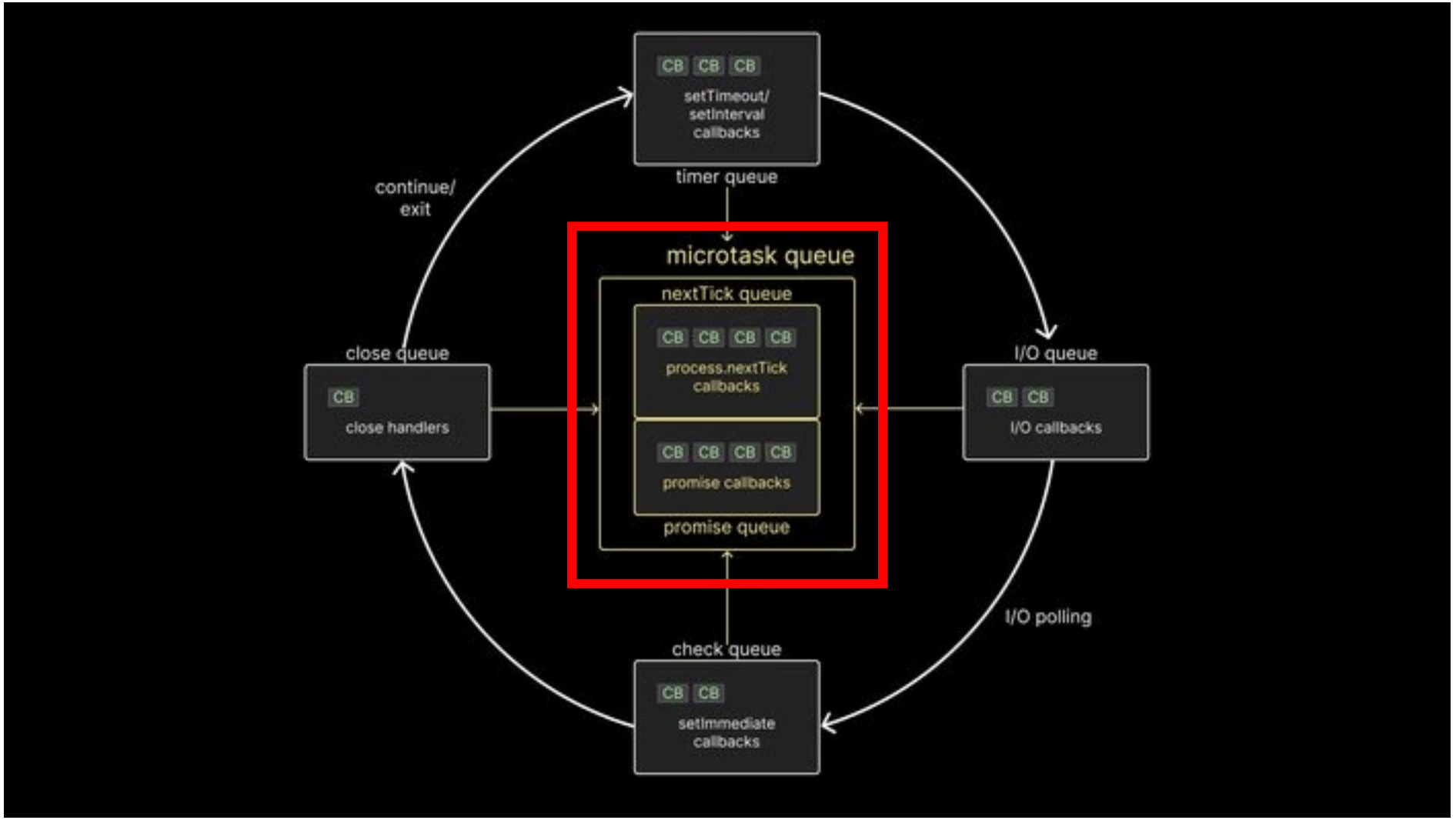
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

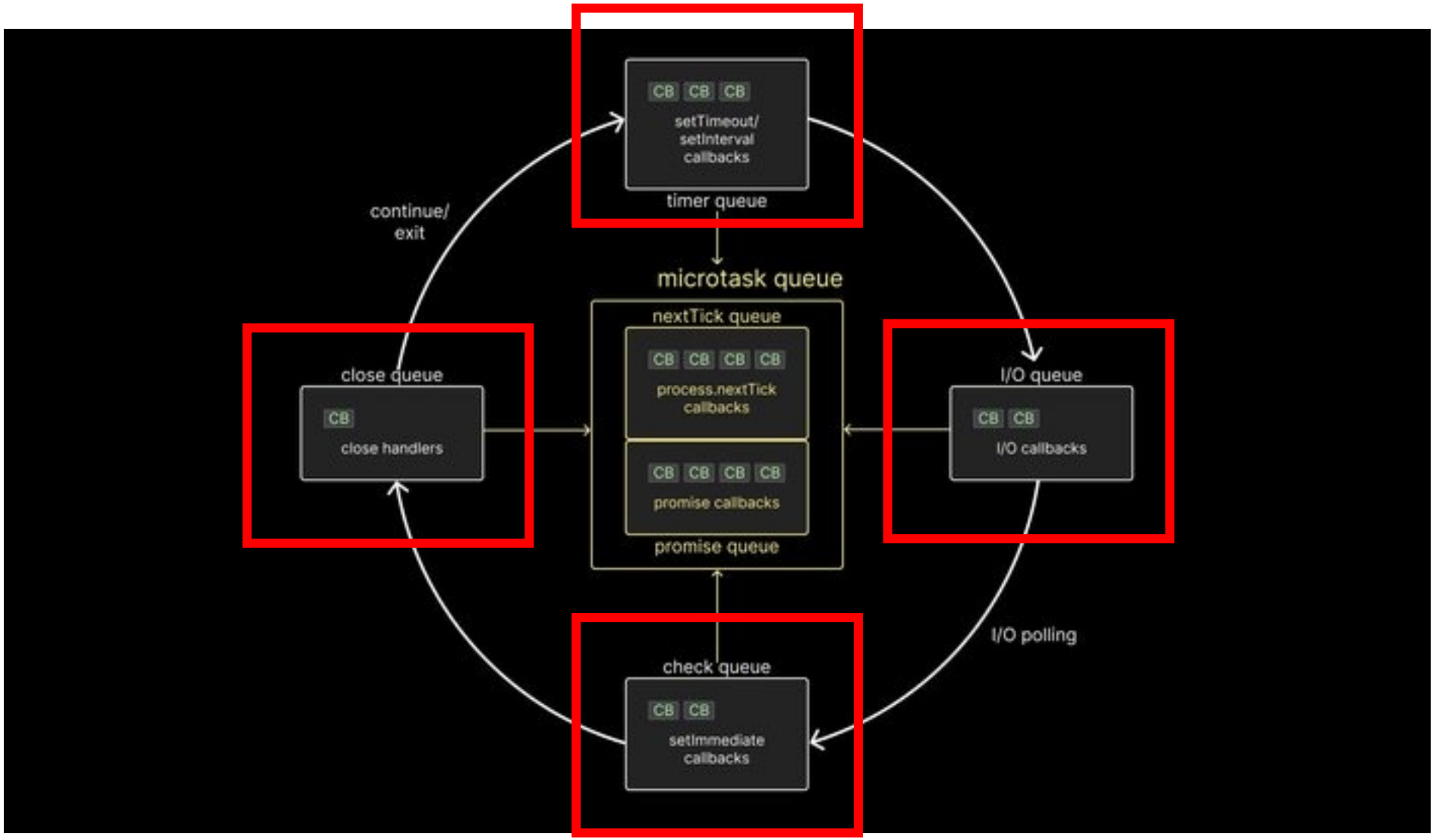
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

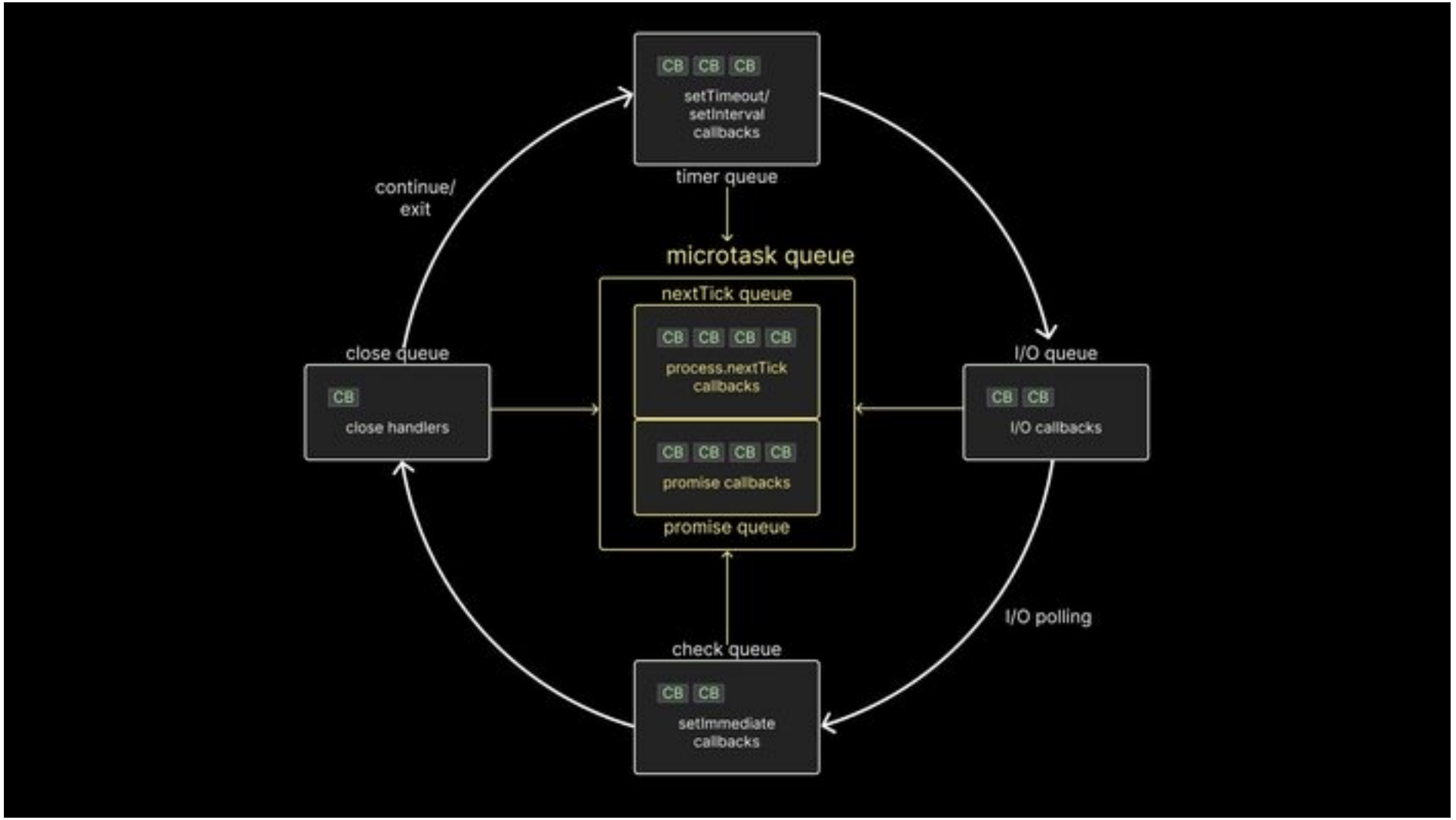
Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue

Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue
  - Tasks inside the timer queue are executed

Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue
  - Tasks inside the timer queue are executed
  - Back to microtask queue, execute it

Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue
  - Tasks inside the timer queue are executed
  - Back to microtask queue, execute it
  - Tasks inside the I/O queue are executed

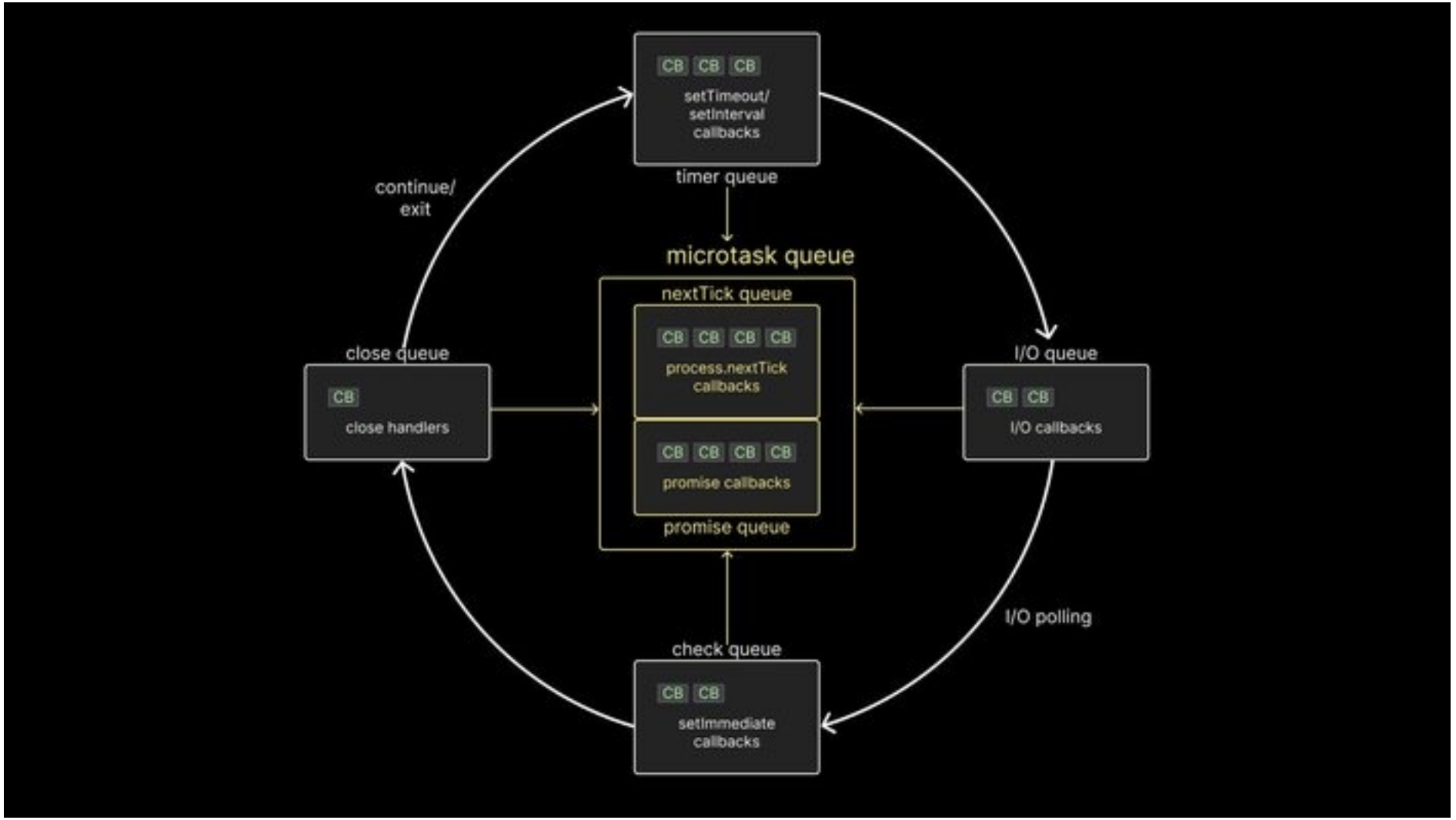
Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue
  - Tasks inside the timer queue are executed
  - Back to microtask queue, execute it
  - Tasks inside the I/O queue are executed
  - Microtask queue execution

Event Loop – Execution Order

- Syncronous JS always has the highest priority
- Inside the event loop:
  - Callbacks inside the microtask queue are executed
  - First nextTick queue and after that inside the promise queue
  - Tasks inside the timer queue are executed
  - Back to microtask queue, execute it
  - Tasks inside the I/O queue are executed
  - Microtask queue execution
  - Task inside the check queue are executed
  - Microtask queue execution
  - Tasks inside the close queue are executed
  - Microtask queue execution

Event Loop


CB CB CB
setTimeout/
setInterval
callbacks
timer queue
continue/
exit
microtask queue
nextTick queue
CB CB CB CB
process.nextTick
callbacks
CB CB CB CB
promise callbacks
promise queue
close queue
CB
close handlers
I/O queue
CB CB
I/O callbacks
I/O polling
check queue
CB CB
setImmediate
callbacks

Event Loop - Demo

Streams and Buffers

Streams and Buffers

- For large data like:
- Video
- file content
- data transfers

Type of streams

- Readable streams
- Writable streams
- Duplex streams
- Transform streams

Buffers inside streams

- Buffers can temporary allocate memory for streams
- Used to store data inside memory for later usage
- Size need to be given on creation

Streams are EventEmitters

- Streams extending the Event Emitters in Node
- We listen to specific events of a stream
- This stream emits events

Task 3

Create a simple ReadStream

Task 3.1

Pause and Resume the stream

File Handling

File Handling

File Handling

```
1  import * as fs from 'fs';
2
3  fs.writeFile
4  fs.writeFileSync
5  fs.readFile
6  fs.readFileSync
```

```
1  import * as fs from 'fs/promises';
```

File Handling

```
1  import * as fs from 'fs';
2
3  fs.writeFile
4  fs.writeFileSync
5  fs.readFile
6  fs.readFileSync
```

```
1  import * as fs from 'fs/promises';
2
3  fs.writeFile
4  fs.readFile
```

File Handling

Task 4

Read the file (async and sync)

Asynchronous Javascript

Asynchronous Javascript - Callbacks

```
1  import * as fs from 'fs';
2
3  fs.readFile('file.txt', 'utf8', (err, data) ⇒ {
4      if (err) {
5          console.error(err);
6          return;
7      }
8      console.log('File Content: ', data);
9  });
```

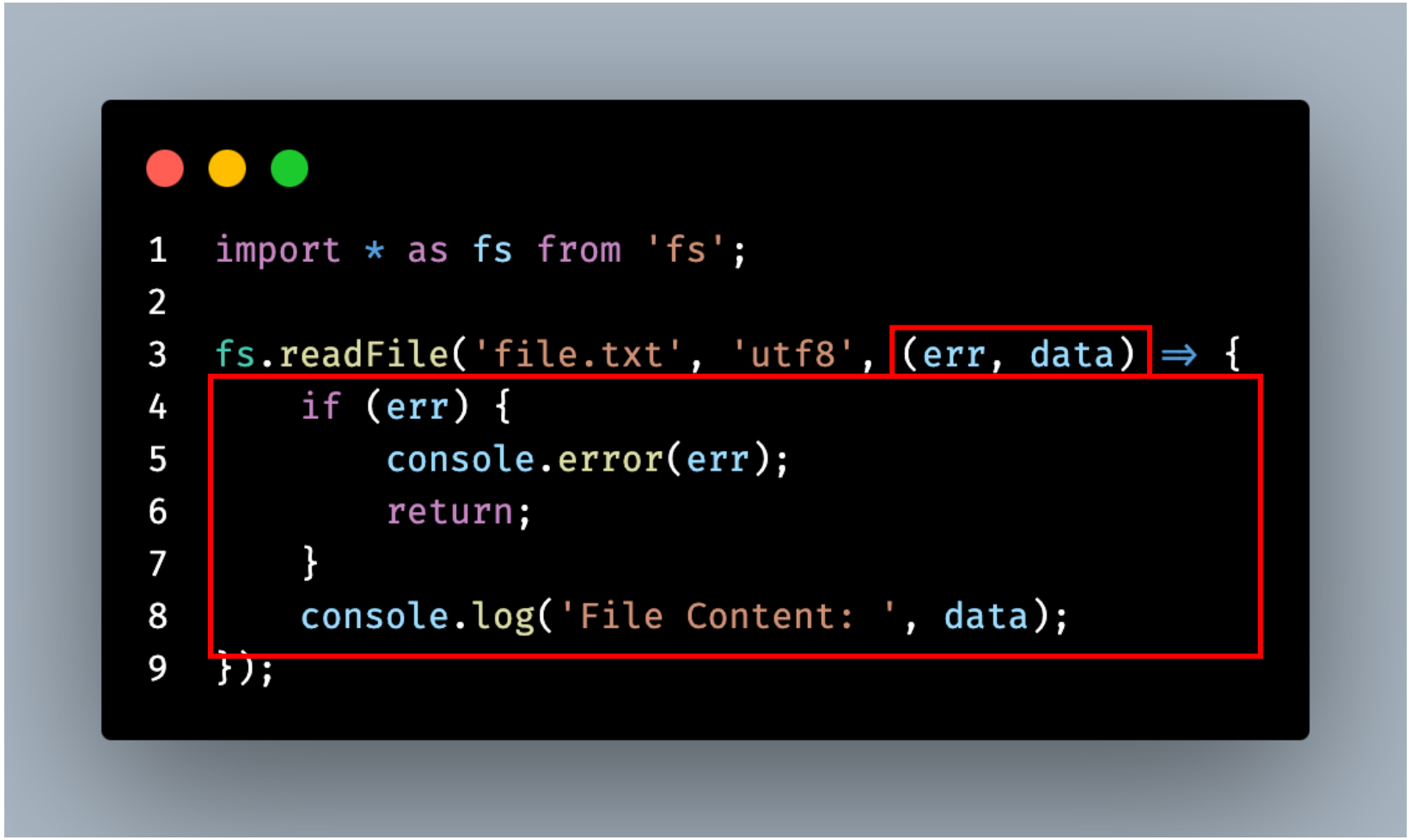
Asynchronous Javascript - Callbacks

```
1  import * as fs from 'fs';
2
3  fs.readFile('file.txt', 'utf8', (err, data) => {
4      if (err) {
5          console.error(err);
6          return;
7      }
8      console.log('File Content: ', data);
9  });
```

Asynchronous Javascript - Callbacks

```
1  import * as fs from 'fs';
2
3  fs.readFile('file.txt', 'utf8', (err, data) => {
4      if (err) {
5          console.error(err);
6          return;
7      }
8      console.log('File Content: ', data);
9  });
```

Asynchronous Javascript - Callbacks



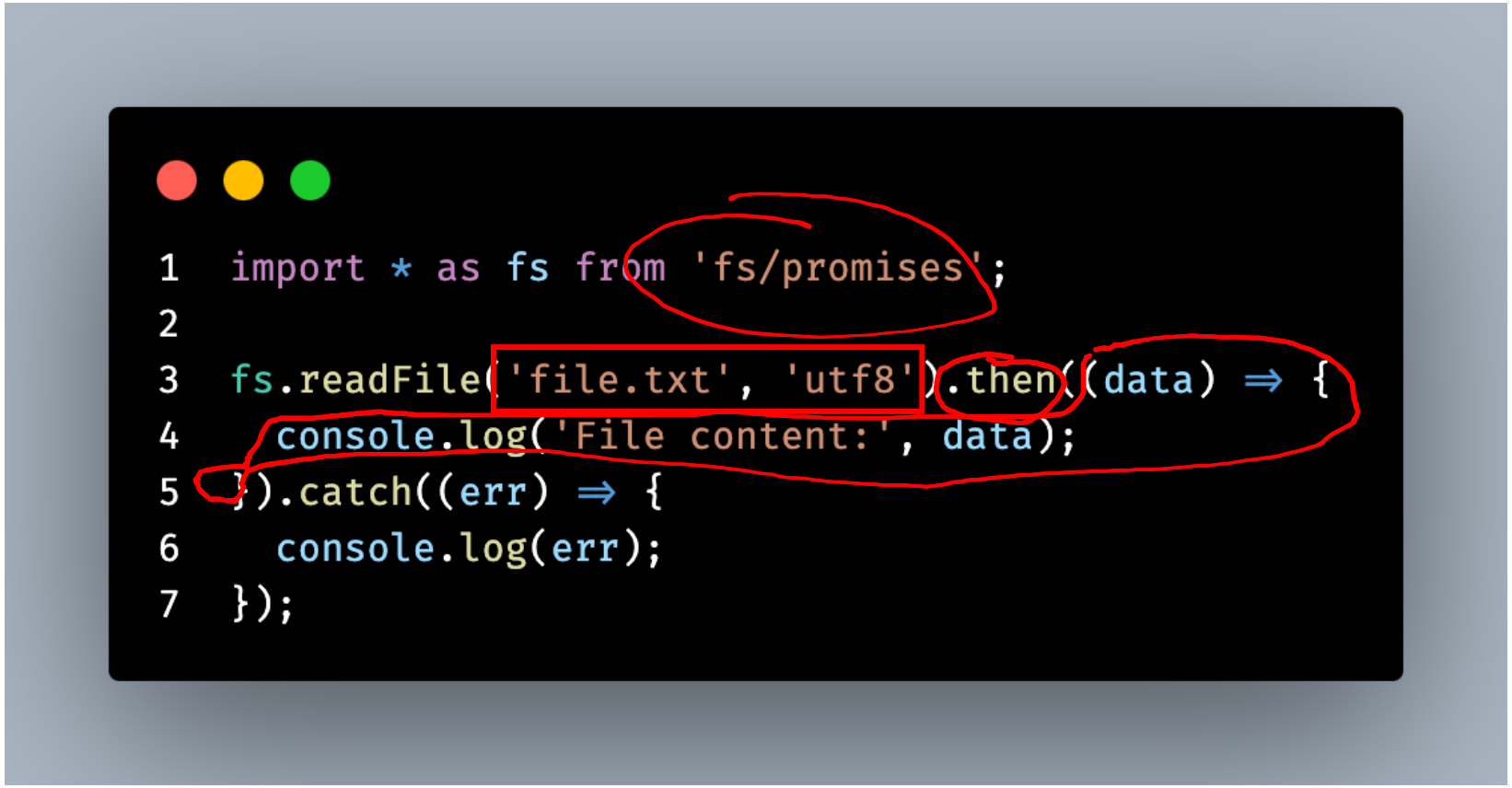
```
1  import * as fs from 'fs';
2
3  fs.readFile('file.txt', 'utf8', (err, data) => {
4      if (err) {
5          console.error(err);
6          return;
7      }
8      console.log('File Content: ', data);
9  });
```

Asynchronous Javascript - Callbacks

```
import * as fs from 'fs';

fs.readFile('file.txt', 'utf8', (err, data) => {
    if (err) {
        console.error(err);
        return;
    }
    console.log('File Content: ', data);

    fs.readFile('file2.txt', 'utf8', (err, data) => {
        if (err) {
            console.error(err);
            return;
        }
        console.log('File2 Content: ', data);

        fs.readFile('file3.txt', 'utf8', (err, data) => {
            if (err) {
                console.error(err);
                return;
            }
            console.log('File3 Content: ', data);
        });
    });
});
```

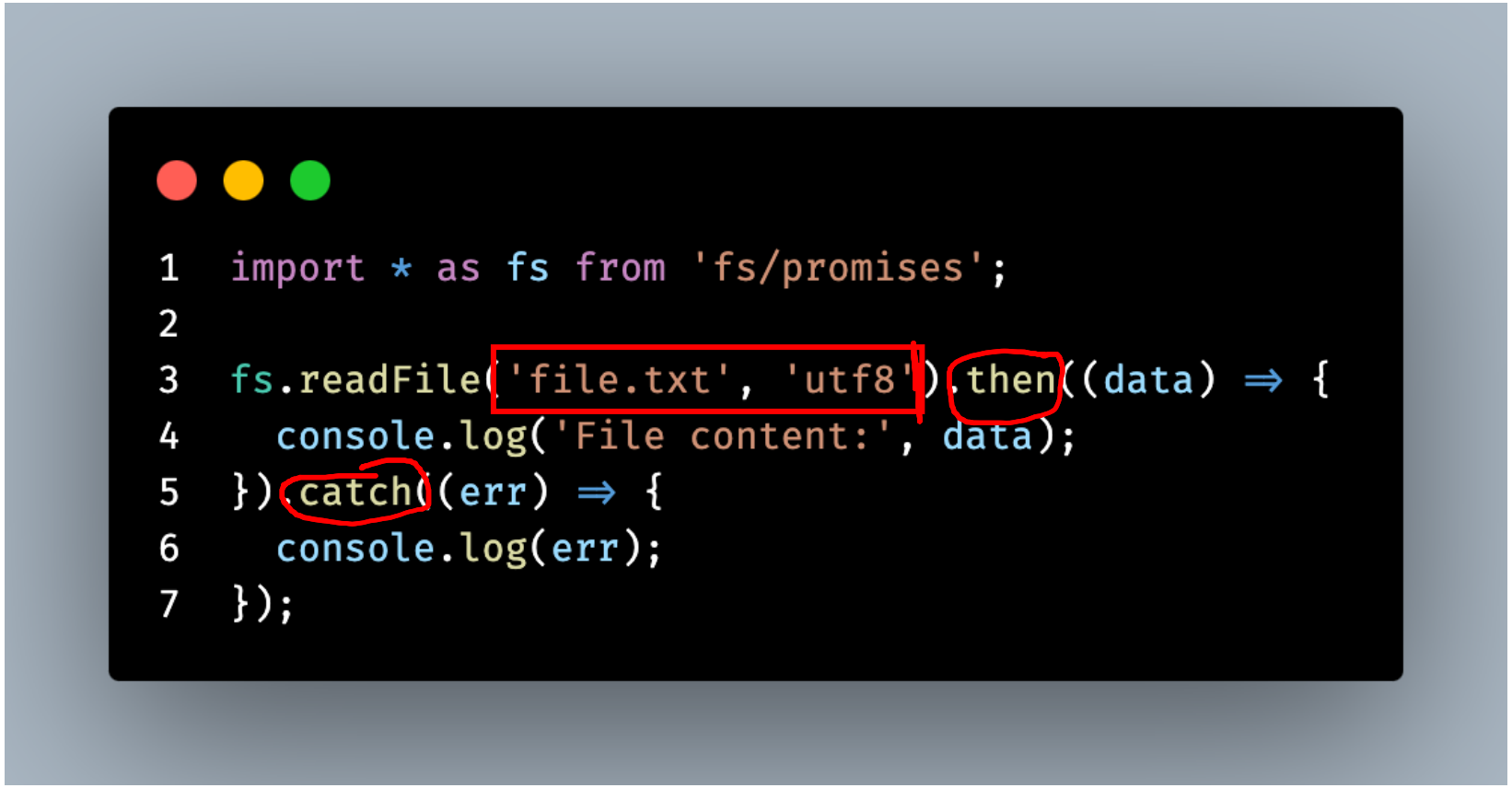
Asynchronous Javascript - Promises

Asynchronous Javascript - Promises



```
1  import * as fs from 'fs/promises';
2
3  fs.readFile('file.txt', 'utf8').then((data) => {
4    console.log('File content:', data);
5  }).catch((err) => {
6    console.log(err);
7  });
```

Asynchronous Javascript - Promises

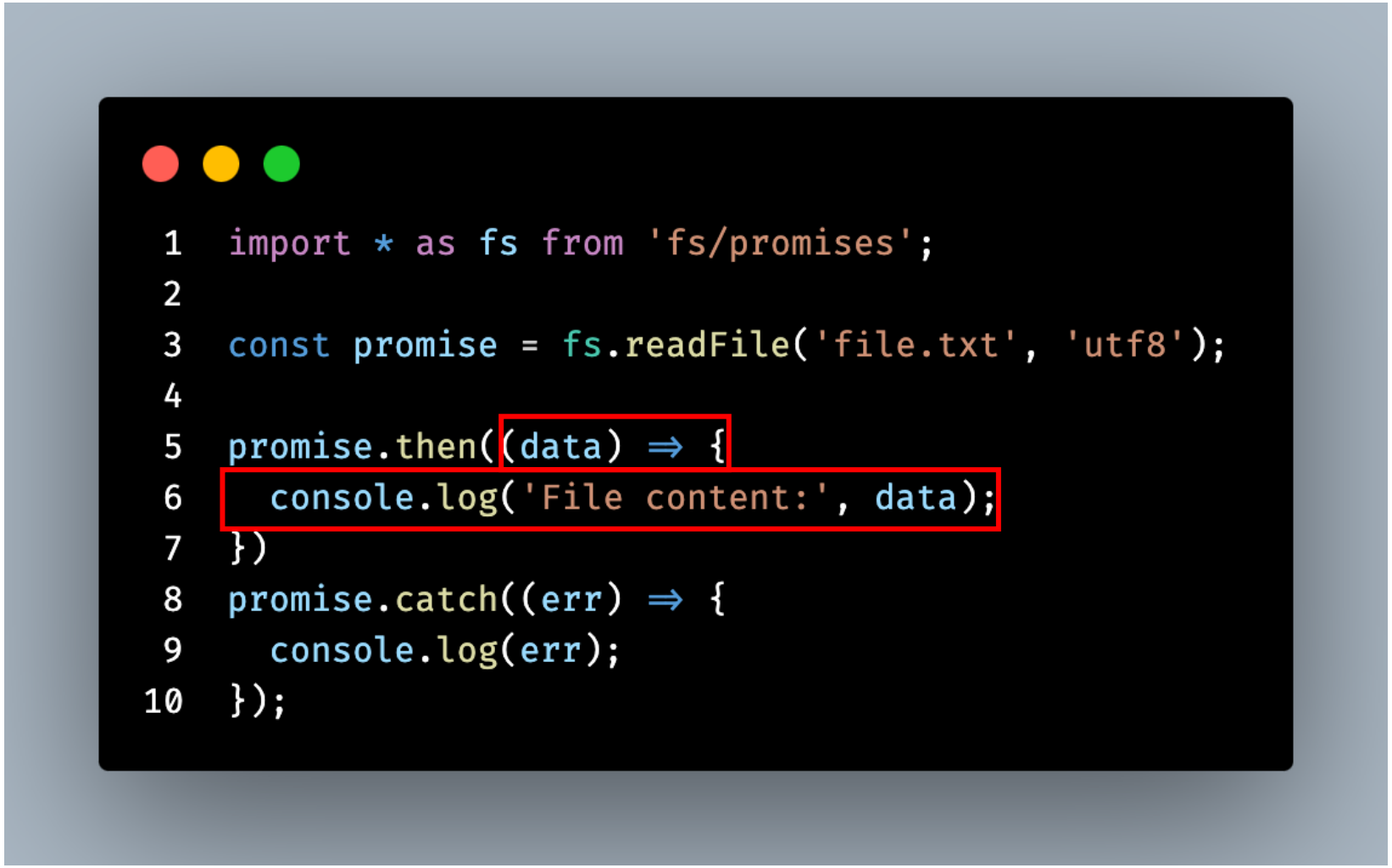


```
1  import * as fs from 'fs/promises';
2
3  fs.readFile('file.txt', 'utf8').then((data) => {
4    console.log('File content:', data);
5  }).catch((err) => {
6    console.log(err);
7  });
```

Asynchronous Javascript - Promises

```
1  import * as fs from 'fs/promises';
2
3  const promise = fs.readFile('file.txt', 'utf8');
4
5  promise.then((data) => {
6    console.log('File content:', data);
7  })
8  promise.catch((err) => {
9    console.log(err);
10 });
```

Asynchronous Javascript - Promises

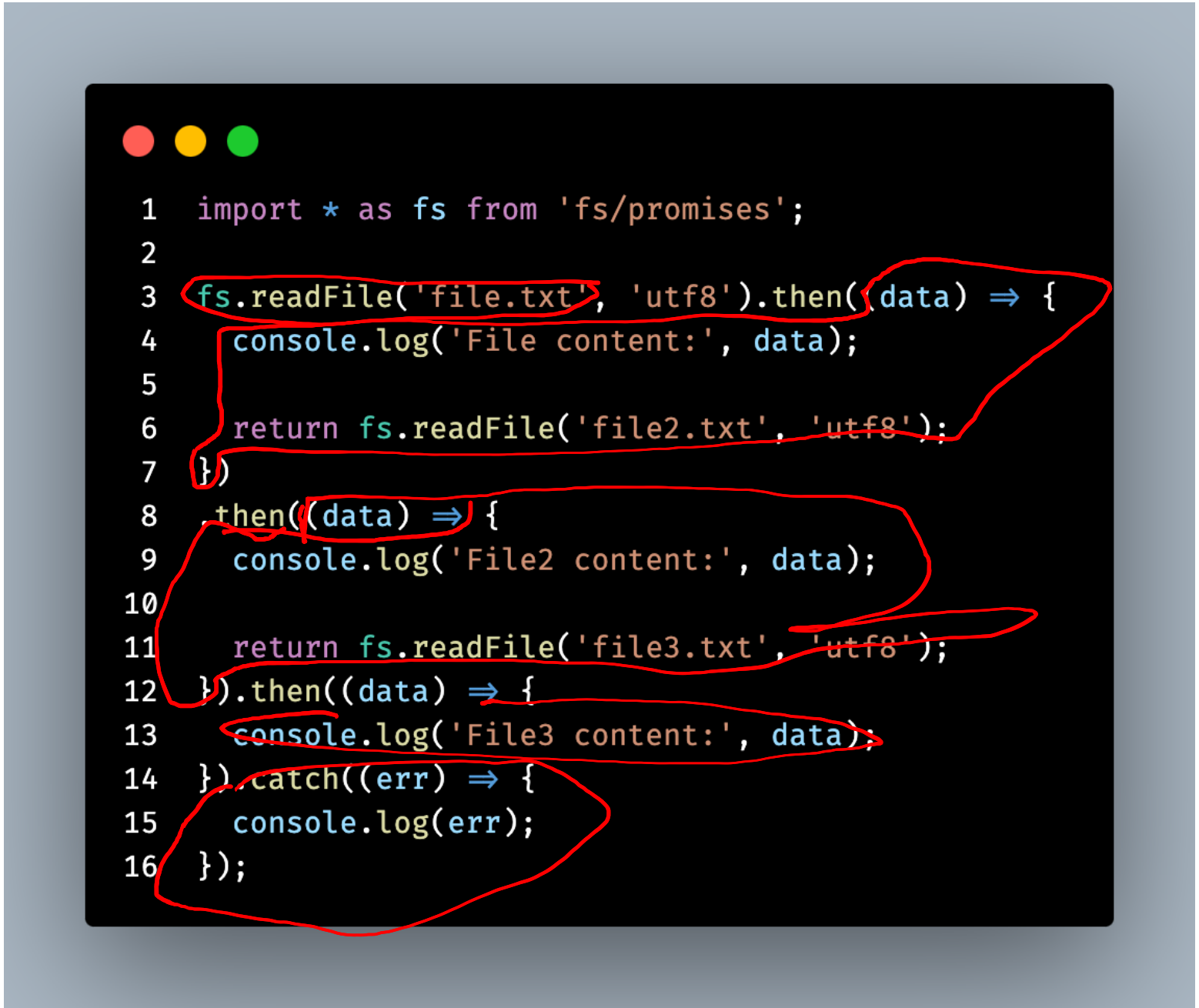


```
1  import * as fs from 'fs/promises';
2
3  const promise = fs.readFile('file.txt', 'utf8');
4
5  promise.then((data) => {
6    console.log('File content:', data);
7  })
8  promise.catch((err) => {
9    console.log(err);
10 });
```

Asynchronous Javascript - Callbacks

```
import * as fs from 'fs';

fs.readFile('file.txt', 'utf8', (err, data) => {
    if (err) {
        console.error(err);
        return;
    }
    console.log('File Content: ', data);

    fs.readFile('file2.txt', 'utf8', (err, data) => {
        if (err) {
            console.error(err);
            return;
        }
        console.log('File2 Content: ', data);

        fs.readFile('file3.txt', 'utf8', (err, data) => {
            if (err) {
                console.error(err);
                return;
            }
            console.log('File3 Content: ', data);
        });
    });
});
```

Asynchronous Javascript - Promises



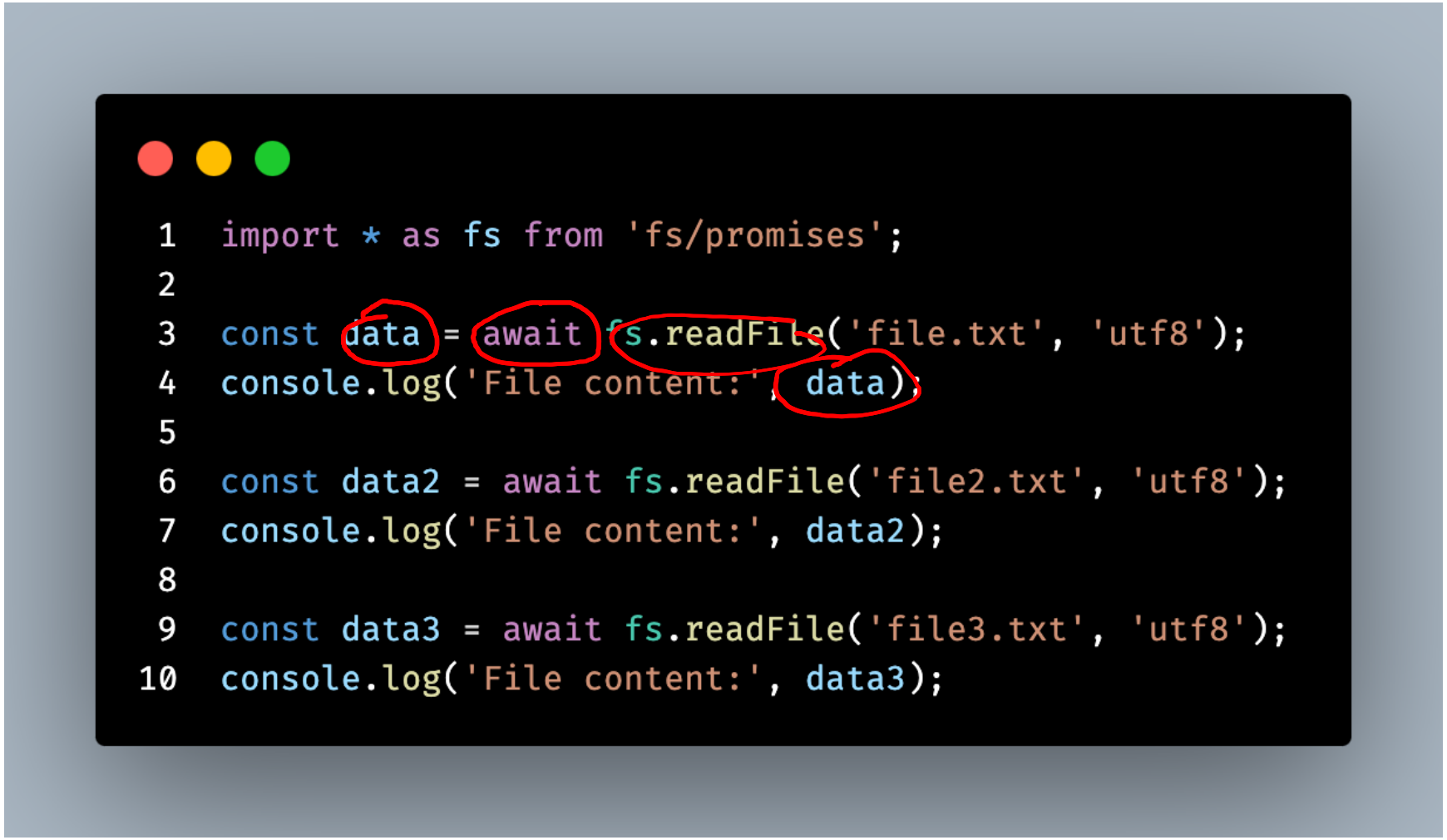
```
1  import * as fs from 'fs/promises';
2
3  fs.readFile('file.txt', 'utf8').then((data) => {
4    console.log('File content:', data);
5
6    return fs.readFile('file2.txt', 'utf8');
7  })
8  .then((data) => {
9    console.log('File2 content:', data);
10
11   return fs.readFile('file3.txt', 'utf8');
12 }).then((data) => {
13   console.log('File3 content:', data);
14 }).catch((err) => {
15   console.log(err);
16 });
```

Asynchronous Javascript - Promises

```
1  import * as fs from 'fs/promises';
2
3  fs.readFile('file.txt', 'utf8').then((data) ⇒ {
4    console.log('File content:', data);
5
6    return fs.readFile('file2.txt', 'utf8');
7  })
8  .then((data) ⇒ {
9    console.log('File2 content:', data);
10
11   return fs.readFile('file3.txt', 'utf8');
12 }).then((data) ⇒ {
13   console.log('File3 content:', data);
14 }).catch((err) ⇒ {
15   console.log(err);
16 });
```

Asynchronous Javascript – Async/Await

Asynchronous Javascript – Async/Await



```
1  import * as fs from 'fs/promises';
2
3  const data = await fs.readFile('file.txt', 'utf8');
4  console.log('File content:', data);
5
6  const data2 = await fs.readFile('file2.txt', 'utf8');
7  console.log('File content:', data2);
8
9  const data3 = await fs.readFile('file3.txt', 'utf8');
10 console.log('File content:', data3);
```

Asynchronous Javascript – Async/Await

```
1  import * as fs from 'fs/promises';
2
3  try {
4    const data = await fs.readFile('file.txt', 'utf8');
5    console.log('File content:', data);
6
7    const data2 = await fs.readFile('file2.txt', 'utf8');
8    console.log('File content:', data2);
9
10   const data3 = await fs.readFile('file3.txt', 'utf8');
11   console.log('File content:', data3);
12 }
13 catch (err) {
14   console.error(err);
15 }
```

What if I want them to run all at the same time?

What if I want them to run all at the same time?

```
1  import * as fs from 'fs/promises';
2
3  const requests = [];
4  requests.push(fs.readFile('file.txt', 'utf8'));
5  requests.push(fs.readFile('file2.txt', 'utf8'));
6  requests.push(fs.readFile('file3.txt', 'utf8'));
7
8  try {
9    const datas = await Promise.all(requests);
10   console.log('File Contents:', data[0], data[1], data[2]);
11 }
12 catch (err) {
13   console.error(err);
14 }
```

Task 5

Use promises to read the files

Recap / Open Questions

- IntelliJ Plugins for Node
  - https://plugins.jetbrains.com/plugin/6098-node-js
  - https://blog.jetbrains.com/webstorm/2017/08/how-to-configure-code-completion-in-full-stack-javascript-projects/
- Code Sample for Microtask-Queue

Web-APIs with NodeJS

Libraries for API development

Libraries for API development

- Express
- HAPI

Libraries for API development

- Express
- HAPI
- Koa
- Fastify

Frameworks for API development

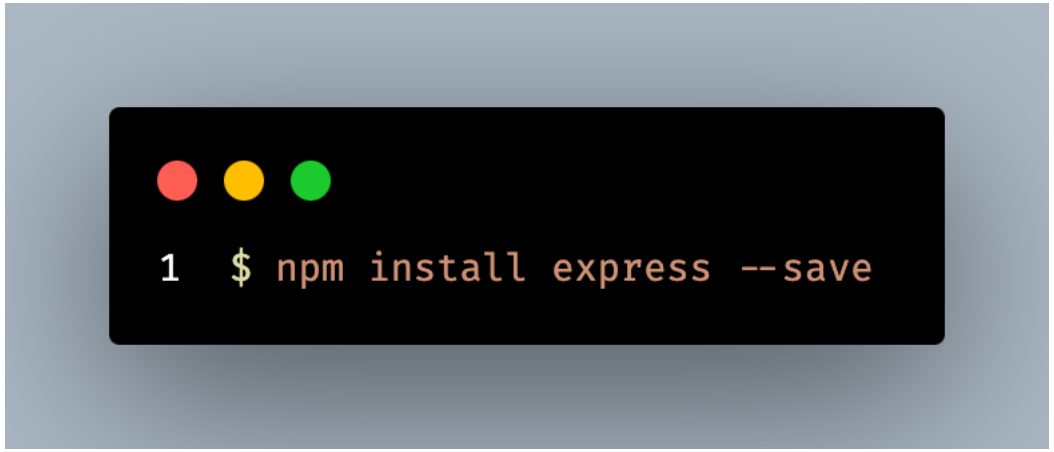
- NestJS

Frameworks for API development

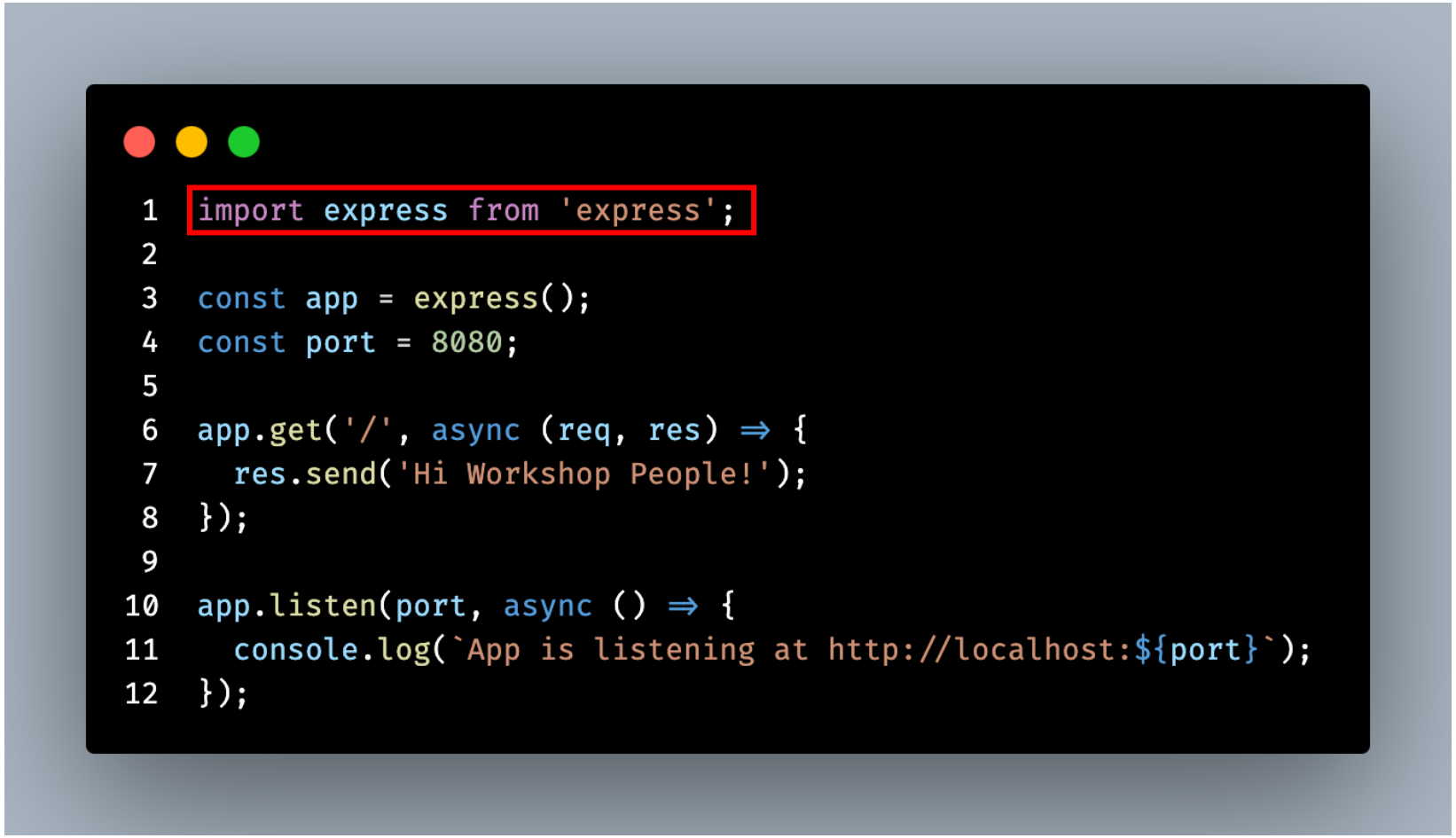
- NestJS
  - Express
  - GraphQL
  - TypeORM
  - Passport

Getting started with Express

Getting started with Express

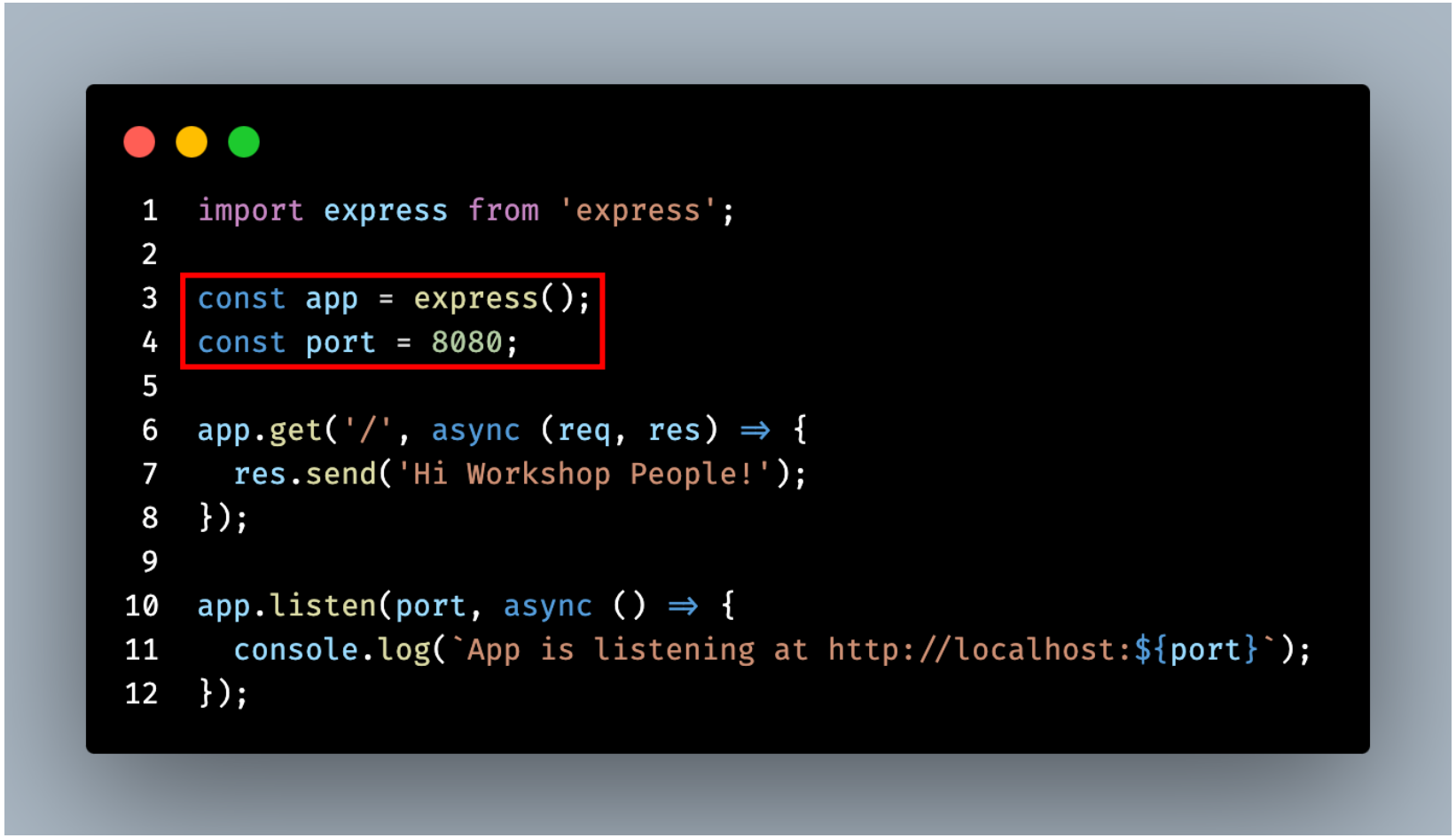


Getting started with Express



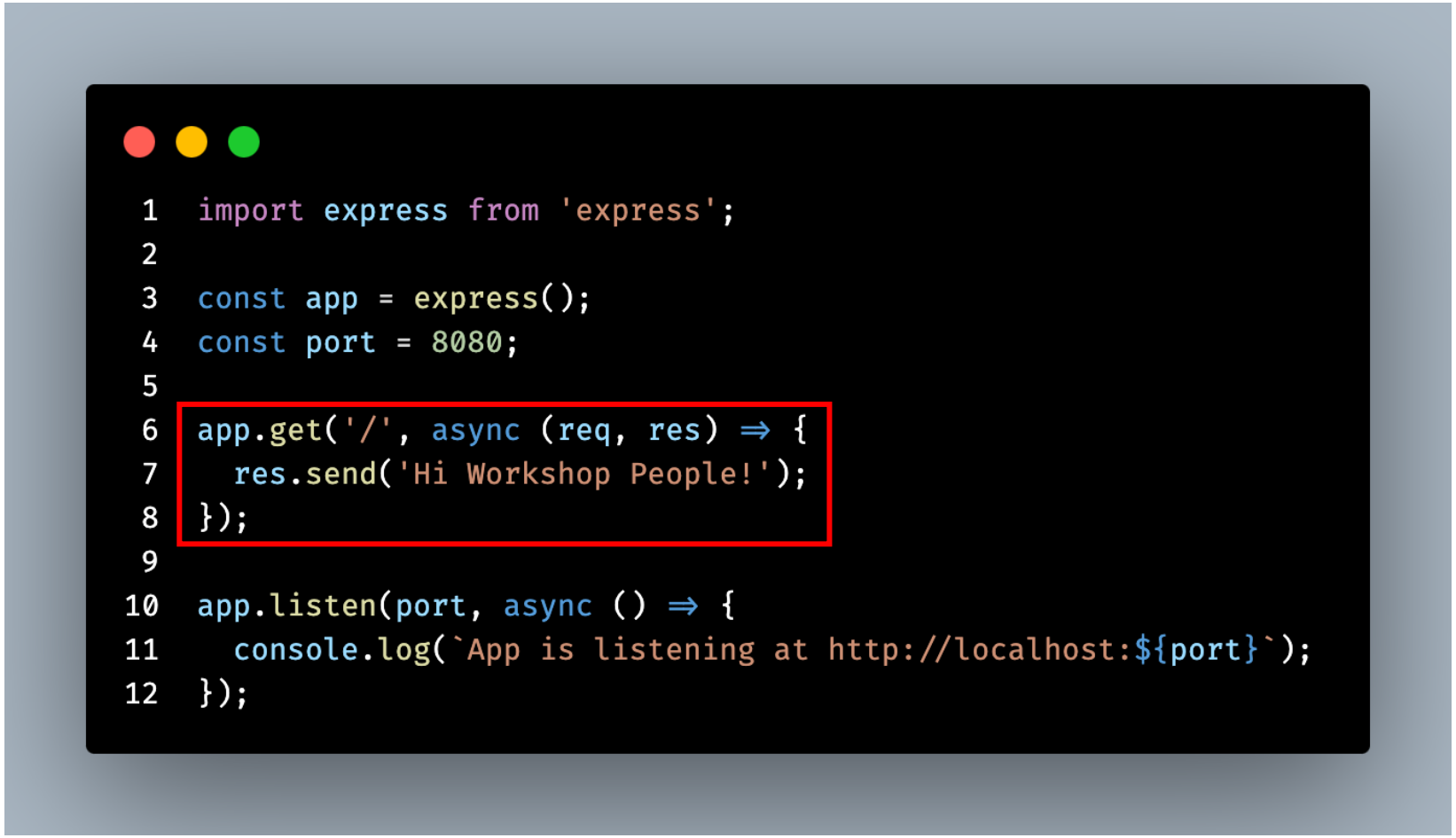
```
1  import express from 'express';
2
3  const app = express();
4  const port = 8080;
5
6  app.get('/', async (req, res) => {
7    res.send('Hi Workshop People!');
8  });
9
10 app.listen(port, async () => {
11   console.log(`App is listening at http://localhost:${port}`);
12 });
```

Getting started with Express



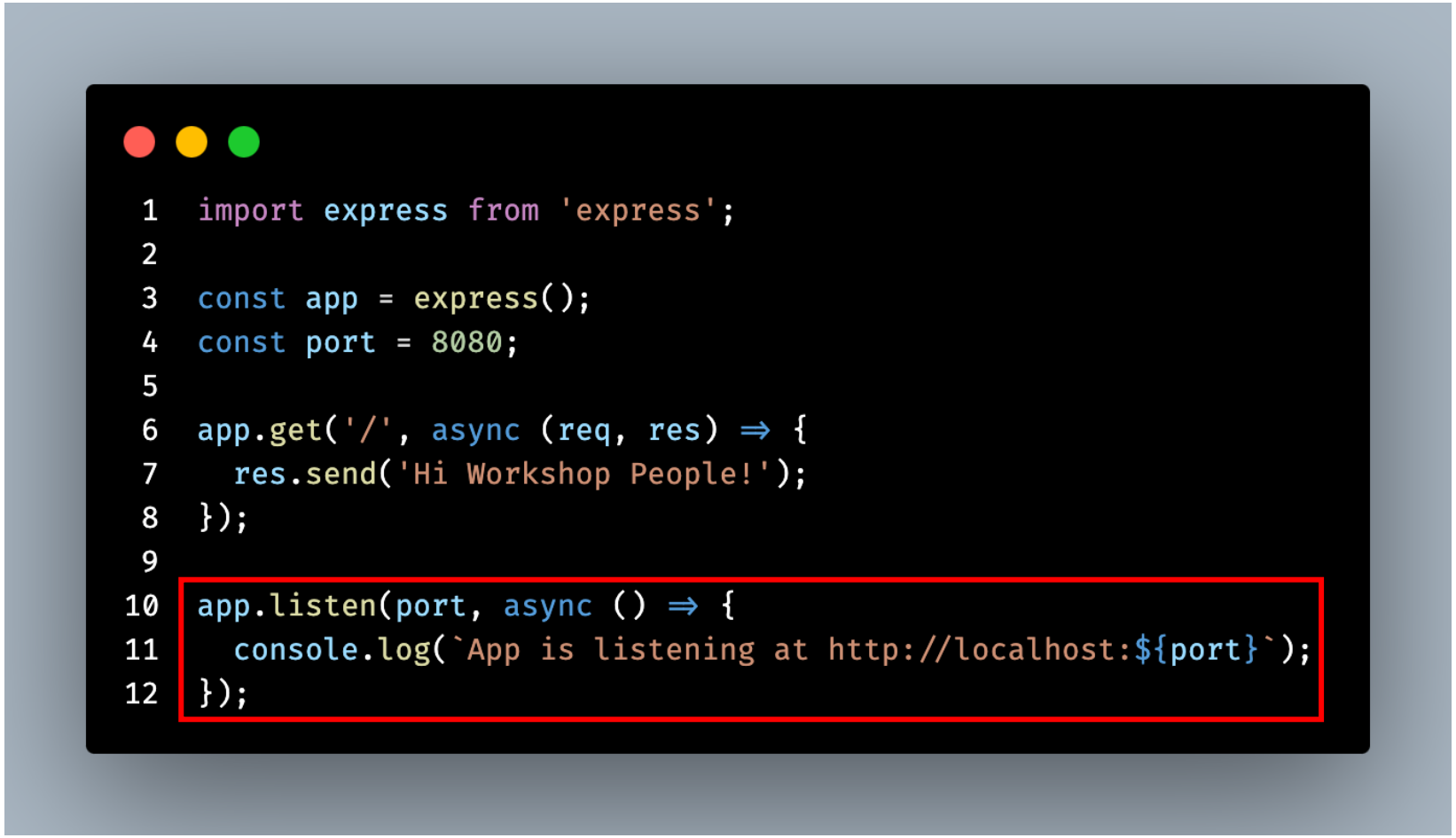
```
1  import express from 'express';
2
3  const app = express();
4  const port = 8080;
5
6  app.get('/', async (req, res) => {
7    res.send('Hi Workshop People!');
8  });
9
10 app.listen(port, async () => {
11   console.log(`App is listening at http://localhost:${port}`);
12 });
```

Getting started with Express



```
1  import express from 'express';
2
3  const app = express();
4  const port = 8080;
5
6  app.get('/', async (req, res) => {
7    res.send('Hi Workshop People!');
8  });
9
10 app.listen(port, async () => {
11   console.log(`App is listening at http://localhost:${port}`);
12 });
```

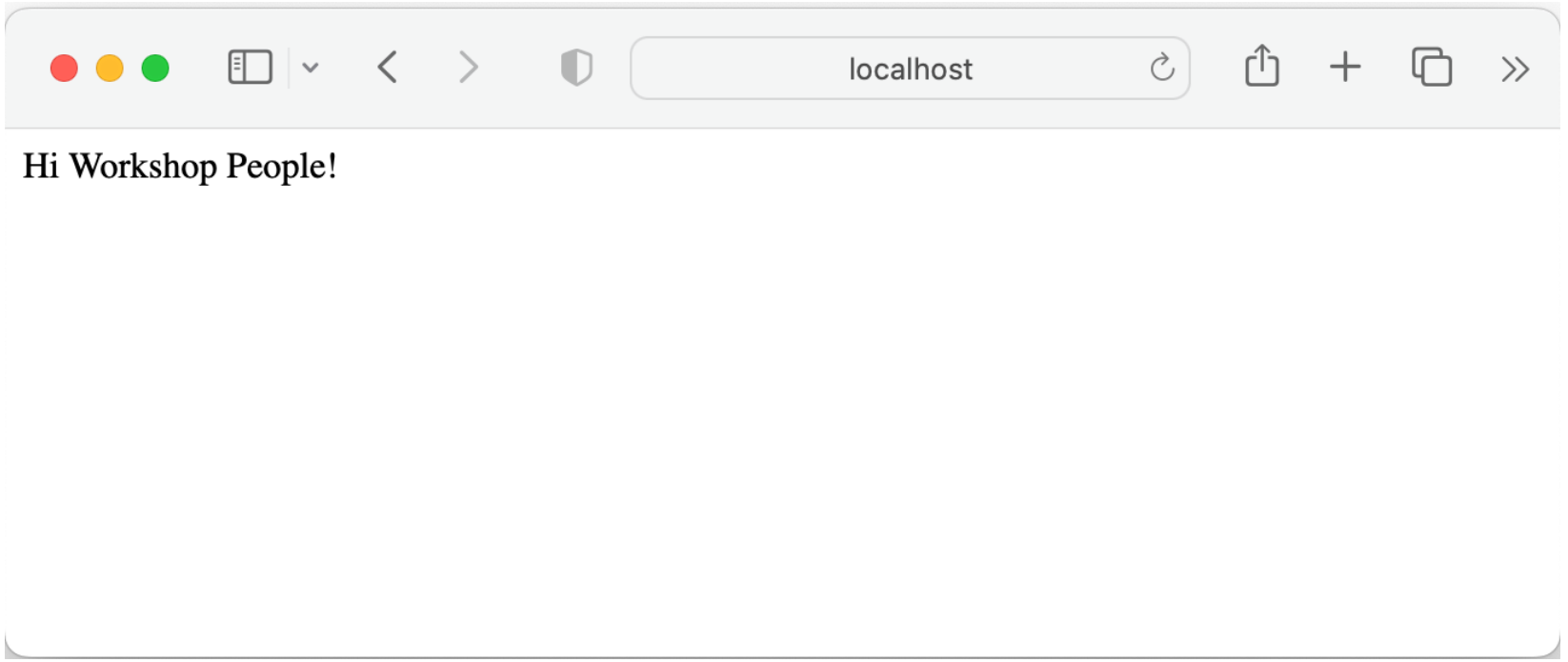
Getting started with Express



```
1  import express from 'express';
2
3  const app = express();
4  const port = 8080;
5
6  app.get('/', async (req, res) => {
7    res.send('Hi Workshop People!');
8  });
9
10 app.listen(port, async () => {
11   console.log(`App is listening at http://localhost:${port}`);
12 });
```

Getting started with Express

```
└─(13:46:37)──> node main.js
App is listening at http://localhost:8080
```



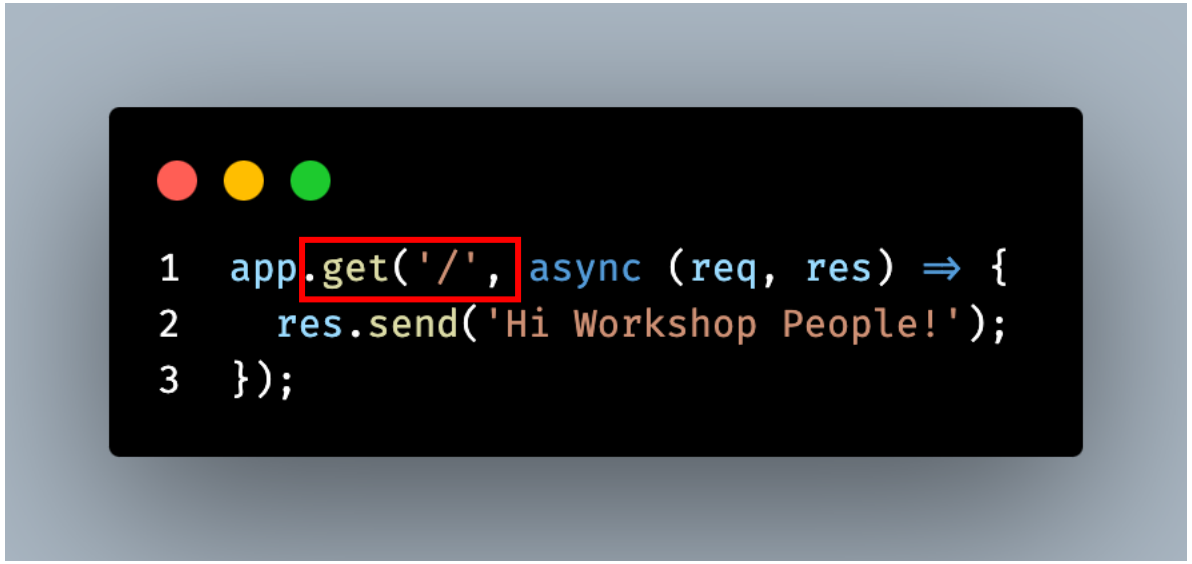
Task 6

Create a simple API

Some important functions on Express

- app.[http method]

Some important functions on Express



Some important functions on Express

- app.[http method]
- app.use

Some important functions on Express

- app.[http method]
- app.use
- express.static

Some important functions on Express

- app.[http method]
- app.use
- express.static

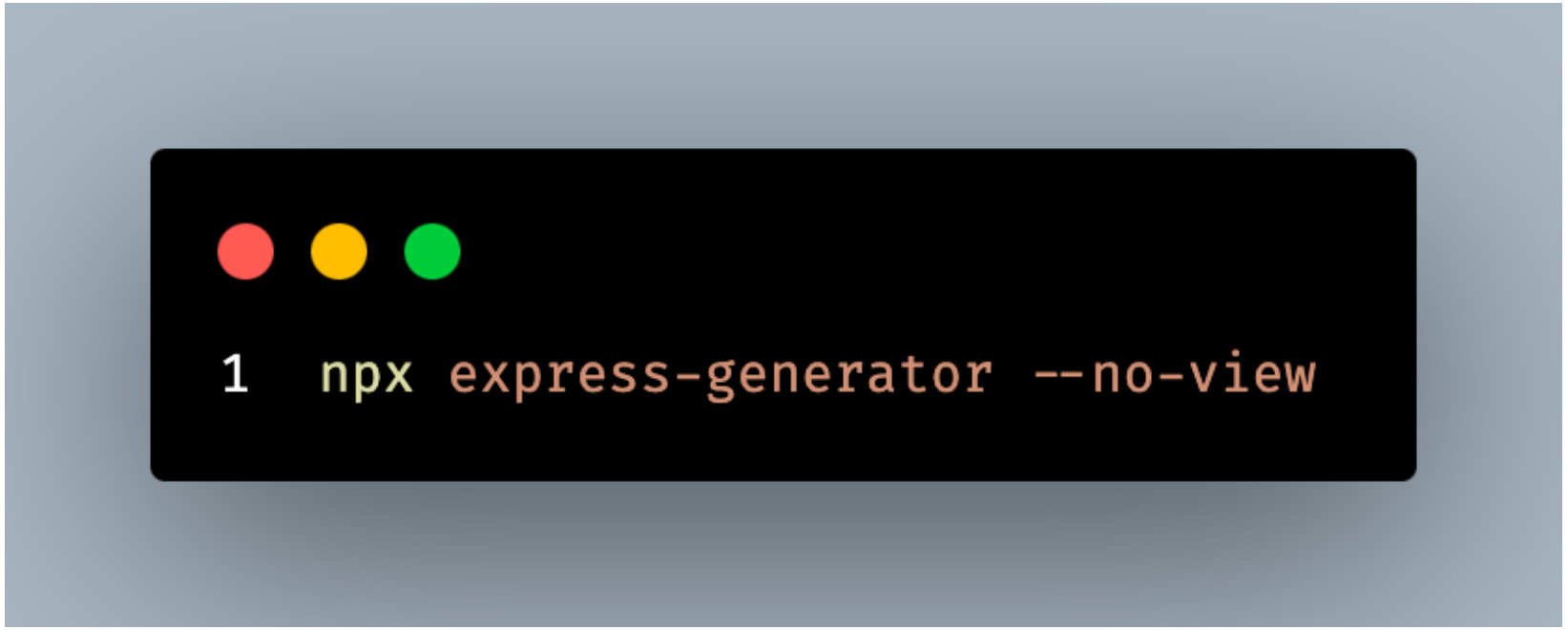
Architecture of Express Apps

Architecture of Express Apps

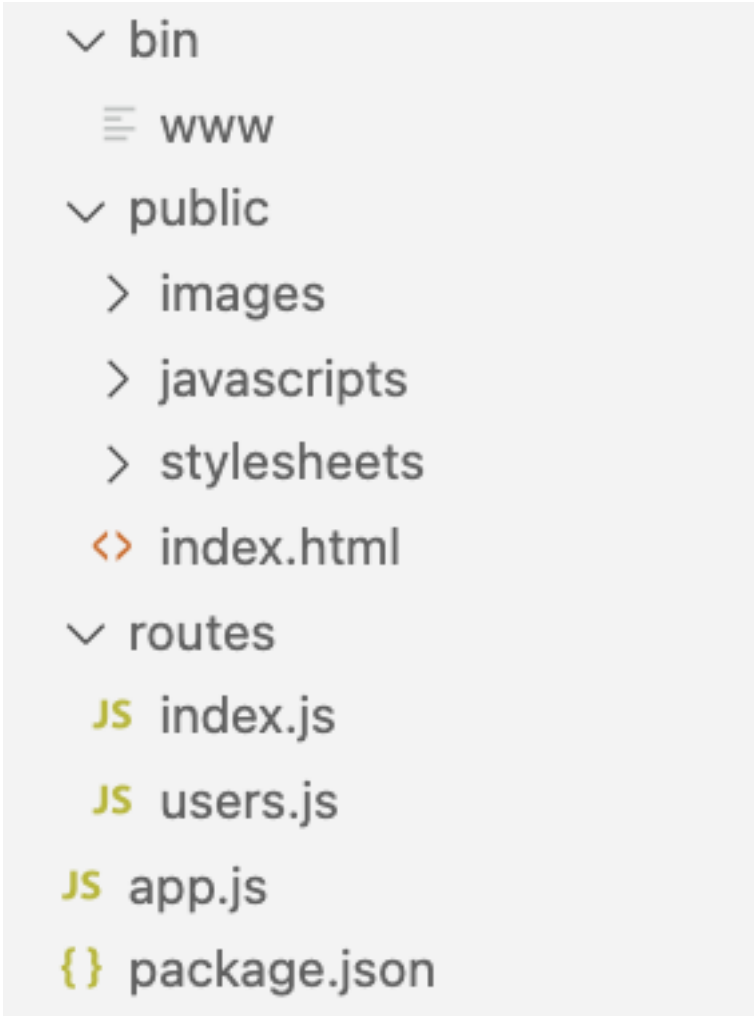
- The express team as a recommended structure
- Package for it: express-generator
- Generator also has support for template engines and styles

Architecture of Express Apps

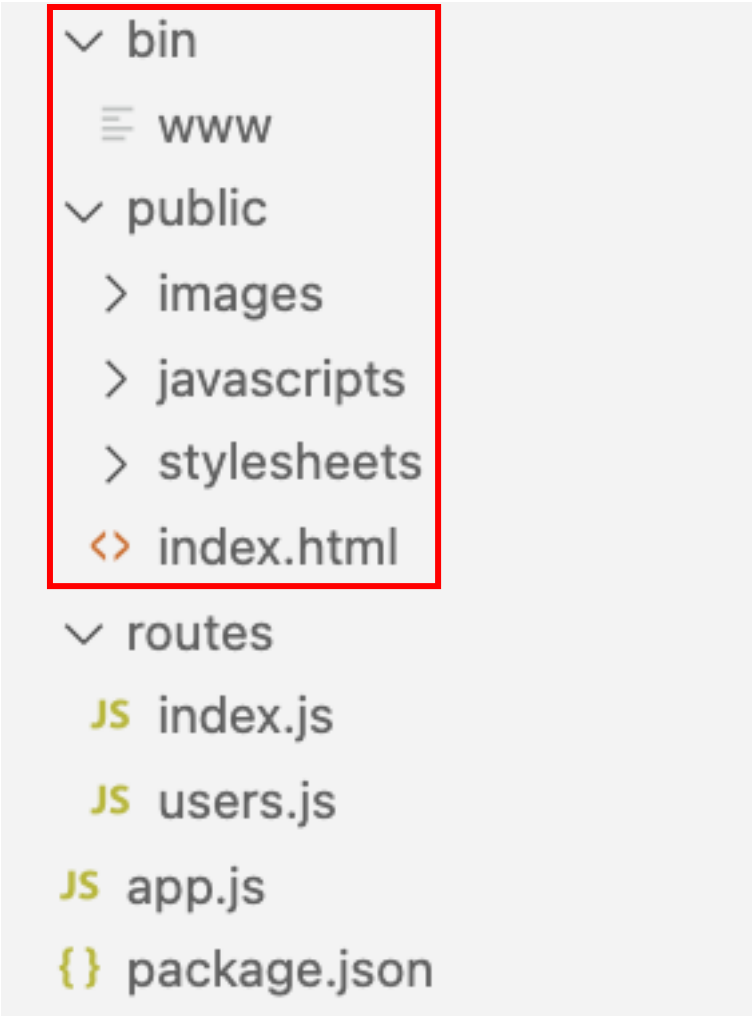
Architecture of Express Apps



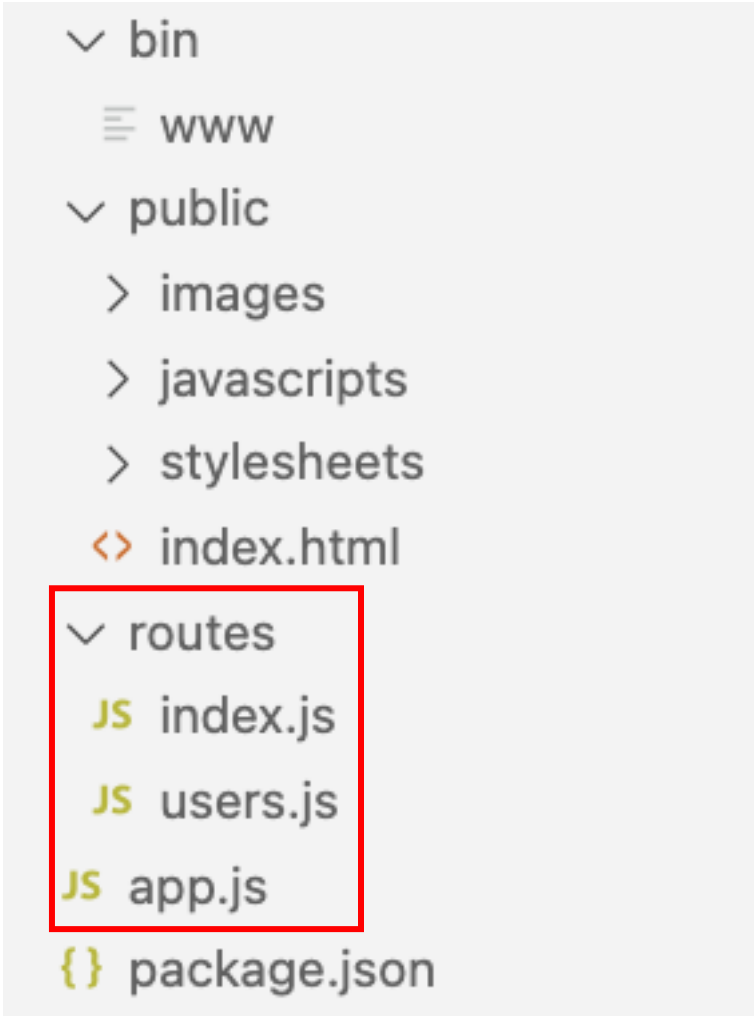
Architecture of Express Apps



Architecture of Express Apps



Architecture of Express Apps



Architecture of Express Apps

```
// routes/index.js
var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

Architecture of Express Apps

```
1  // routes/index.js
2  var express = require('express');
3  var router = express.Router();
4
5  /* GET home page. */
6  router.get('/', function(req, res, next) {
7    res.render('index', { title: 'Express' });
8  });
9
10 module.exports = router;
```

Architecture of Express Apps

```
1  // routes/index.js
2  var express = require('express');
3  var router = express.Router();
4
5  /* GET home page. */
6  router.get('/', function(req, res, next) {
7    res.render('index', { title: 'Express' });
8  });
9
10 module.exports = router;
```

Architecture of Express Apps

```
1  // routes/index.js
2  var express = require('express');
3  var router = express.Router();
4
5  /* GET home page. */
6  router.get('/', function(req, res, next) {
7    res.render('index', { title: 'Express' });
8  });
9
10 module.exports = router;
```

Architecture of Express Apps

```
// routers/users.js
var express = require('express');
var router = express.Router();

/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```

Architecture of Express Apps

```
1  // routers/users.js
2  var express = require('express');
3  var router = express.Router();
4
5  /* GET users listing. */
6  router.get('/', function(req, res, next) {
7    res.send('respond with a resource');
8  });
9
10 module.exports = router;
```

Architecture of Express Apps

```
// app.js
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

module.exports = app;
```

Architecture of Express Apps

```
1  // app.js
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 app.use(logger('dev'));
13 app.use(express.json());
14 app.use(express.urlencoded({ extended: false }));
15 app.use(cookieParser());
16 app.use(express.static(path.join(__dirname, 'public')));
17
18 app.use('/', indexRouter);
19 app.use('/users', usersRouter);
20
21 module.exports = app;
```

Architecture of Express Apps

```
1  // app.js
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 app.use(logger('dev'));
13 app.use(express.json());
14 app.use(express.urlencoded({ extended: false }));
15 app.use(cookieParser());
16 app.use(express.static(path.join(__dirname, 'public')));
17
18 app.use('/', indexRouter);
19 app.use('/users', usersRouter);
20
21 module.exports = app;
```

Architecture of Express Apps

```
1  // app.js
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 app.use(logger('dev'));
13 app.use(express.json());
14 app.use(express.urlencoded({ extended: false }));
15 app.use(cookieParser());
16 app.use(express.static(path.join(__dirname, 'public')));
17
18 app.use('/', indexRouter);
19 app.use('/users', usersRouter);
20
21 module.exports = app;
```

Architecture of Express Apps

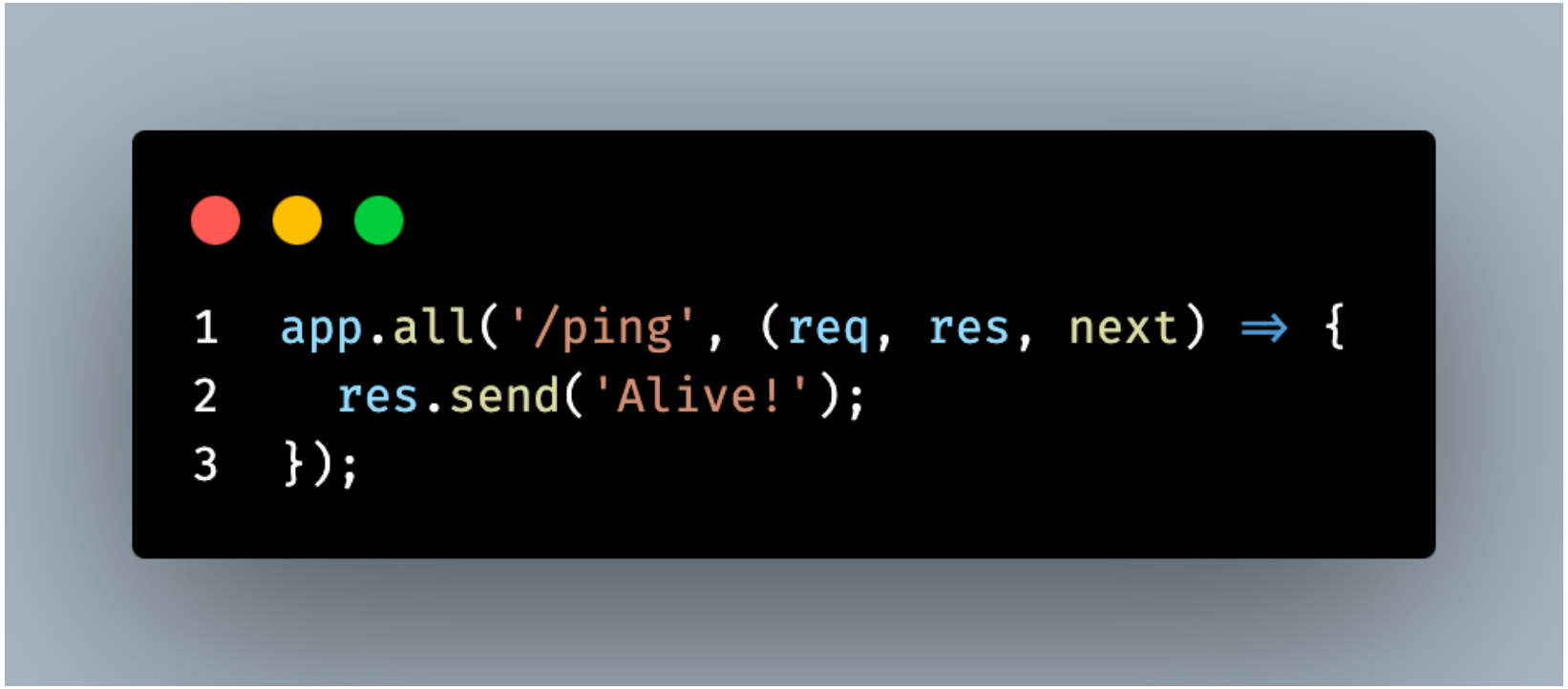
```
// app.js
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

module.exports = app;
```

Task 7

Generate an express app

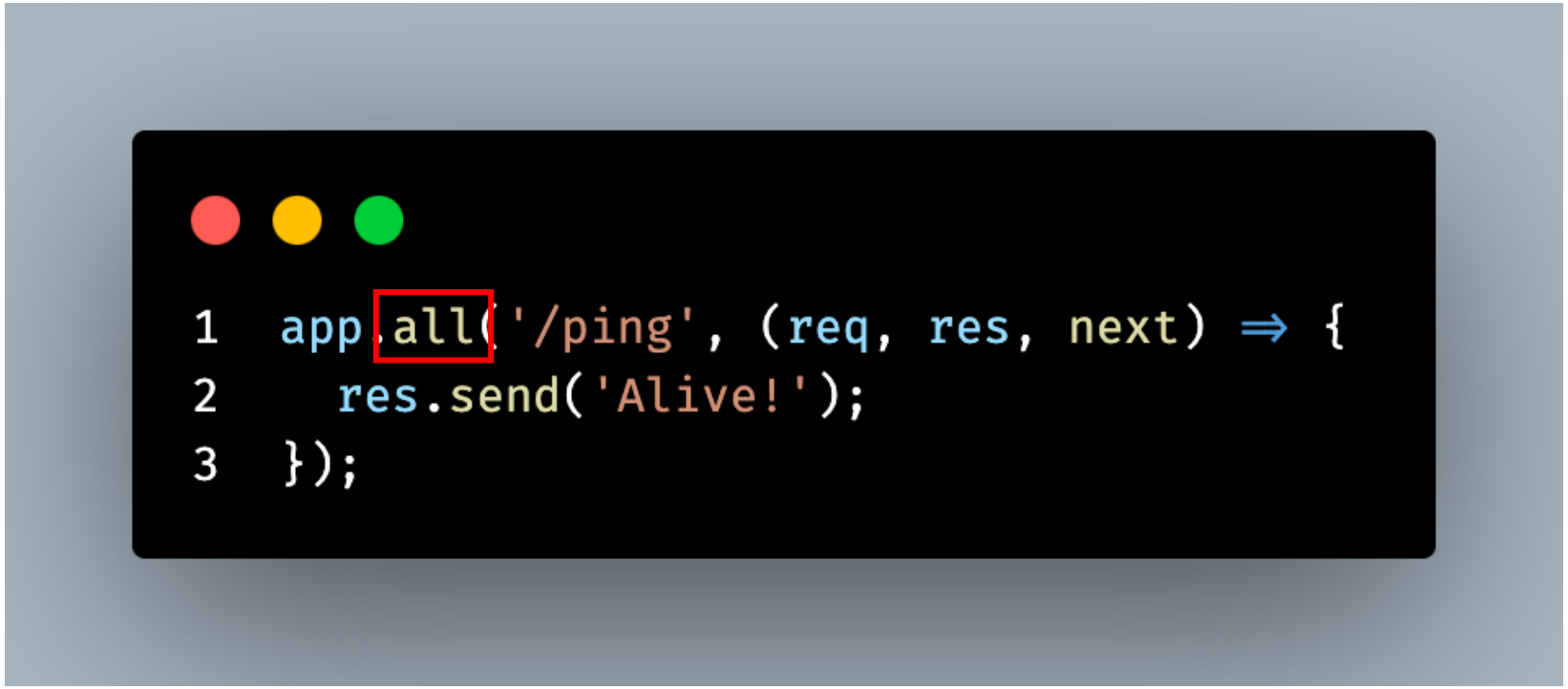
Express - Routing

Express - Routing



```
1  app.all('/ping', (req, res, next) ⇒ {
2    res.send('Alive!');
3  });
```

Express - Routing



```
1  app.all('/ping', (req, res, next) => {
2    res.send('Alive!');
3  });
```

Express - Routing

Express - Routing

- Route definition support Regex

Express - Routing

```
1  app.get(/.*fly$/, (req, res) ⇒ {
2    res.send('/.*fly$/')
3  })
```

Express - Routing

- “/butterfly”
- “/dragonfly”
- “/butterflyman”
- “/dragonflyman”

Express– Routing Parameters

Express– Routing Parameters

```
app.get('/manufacturers/:manufacturerId/cars/:carId', (req, res) ⇒ {
  res.send(req.params)
})
```

Express– Routing Parameters

```
1  app.get('/manufacturers/:manufacturerId/cars/:carId', (req, res) ⇒ {
2    res.send(req.params)
3  })
```

/manufacturers/3/cars/8

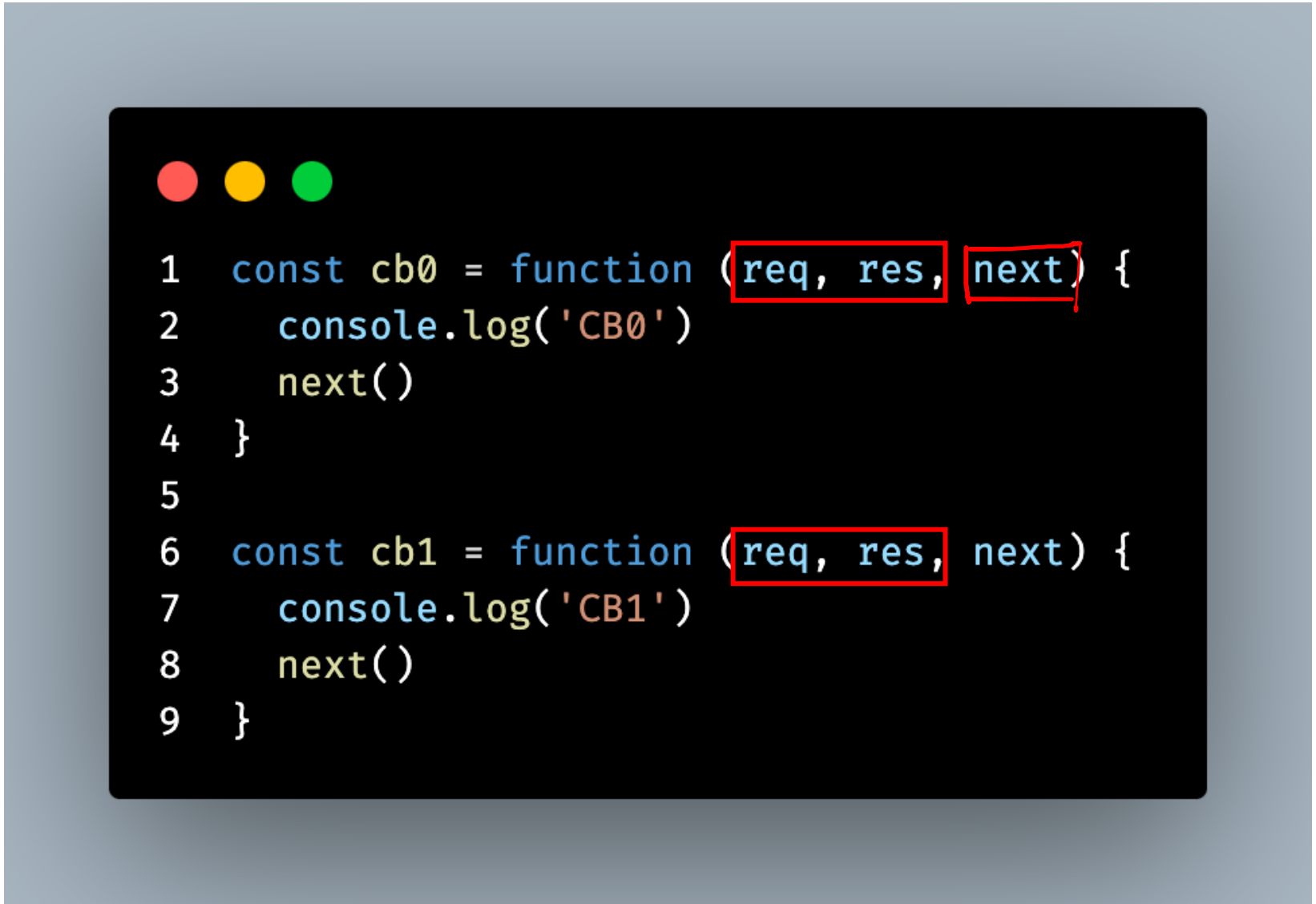
Express– Routing Parameters

```
1  app.get('/manufacturers/:manufacturerId/cars/:carId', (req, res) ⇒ {
2    console.log(req.params.manufacturerId, req.params.carId);
3    // output: 3 8
4    res.send(req.params)
5  })
```

Express – Route Chaining

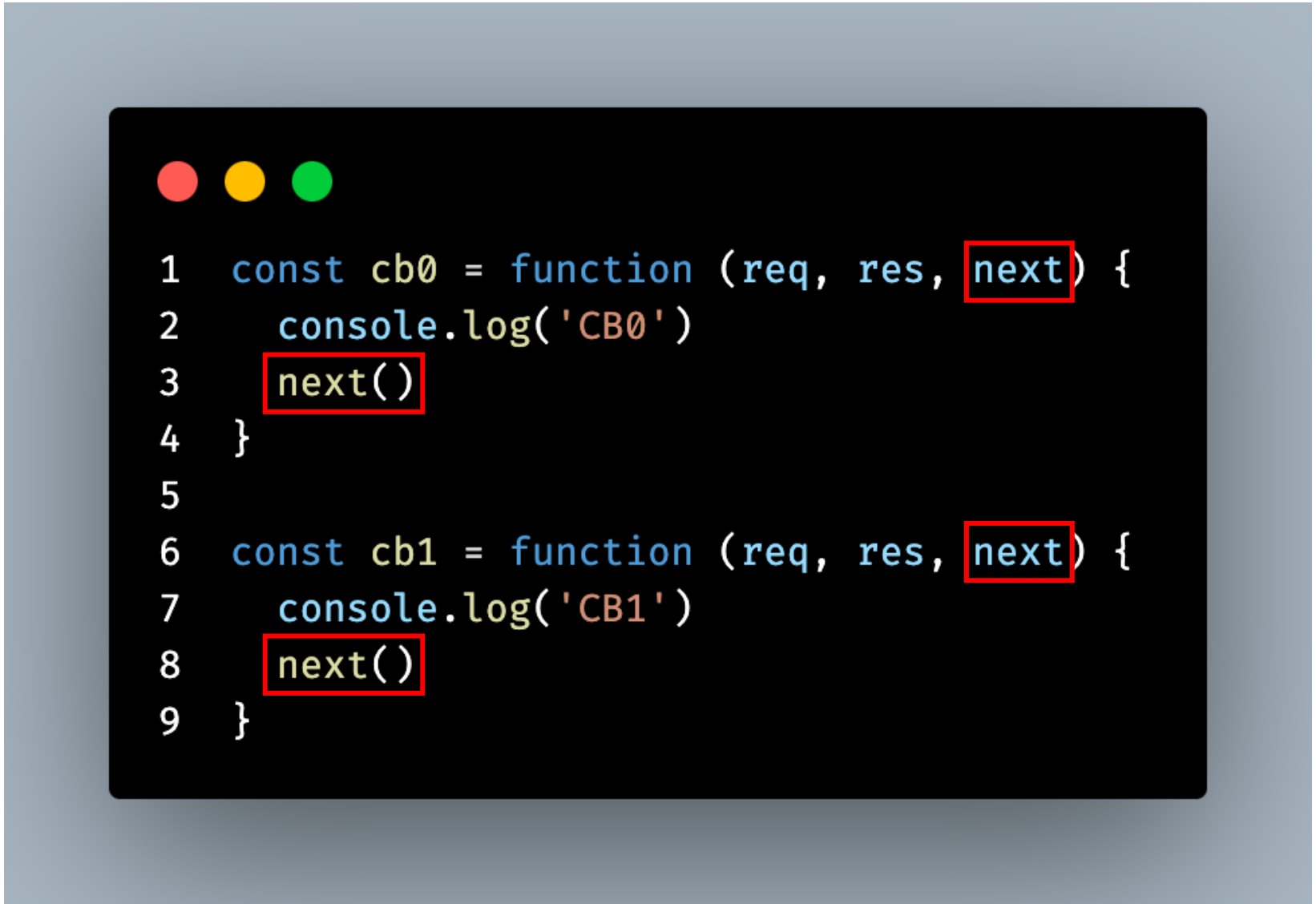
- Express supports chaining callback functions
- Route-Handling by chaining with ”next” function
- Chaining HTTP methods by the route

Express Route-Chaning



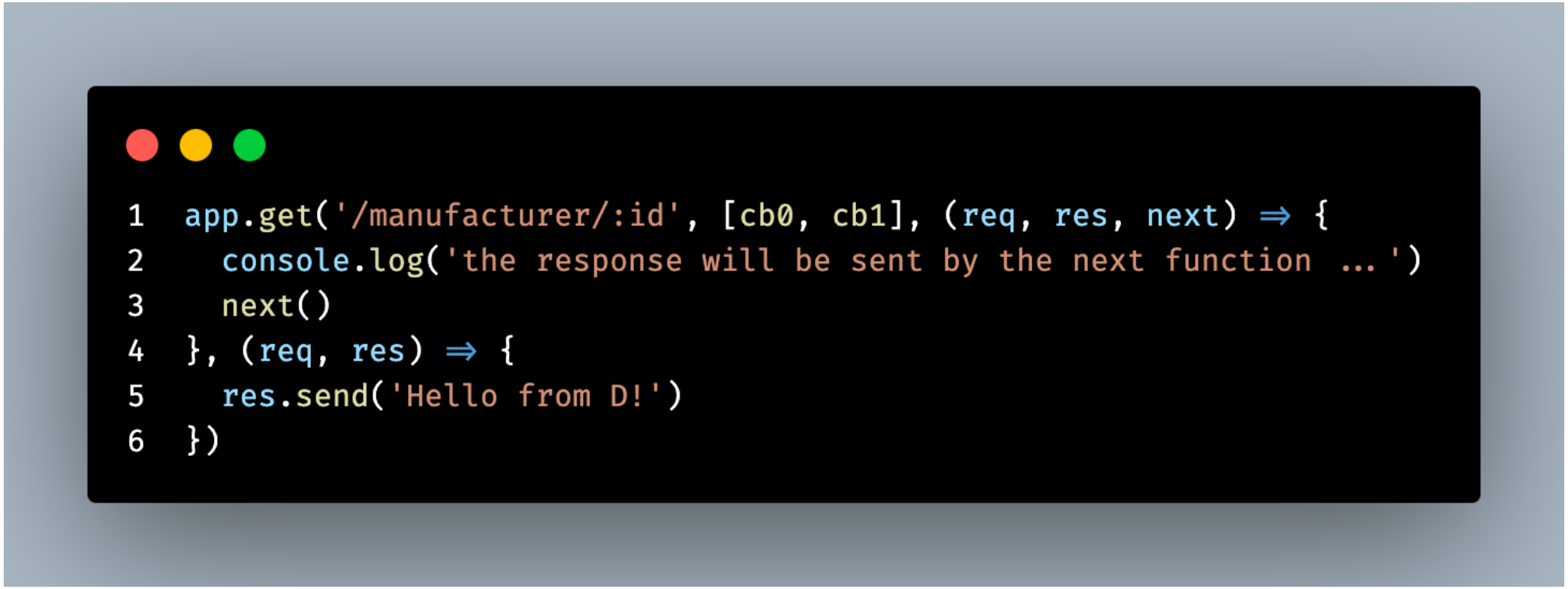
```
const cb0 = function (req, res, next) {
  console.log('CB0')
  next()
}

const cb1 = function (req, res, next) {
  console.log('CB1')
  next()
}
```

Express Route-Chaning



```
1  const cb0 = function (req, res, next) {
2    console.log('CB0')
3    next()
4  }
5
6  const cb1 = function (req, res, next) {
7    console.log('CB1')
8    next()
9  }
```

Express – Chain Routing

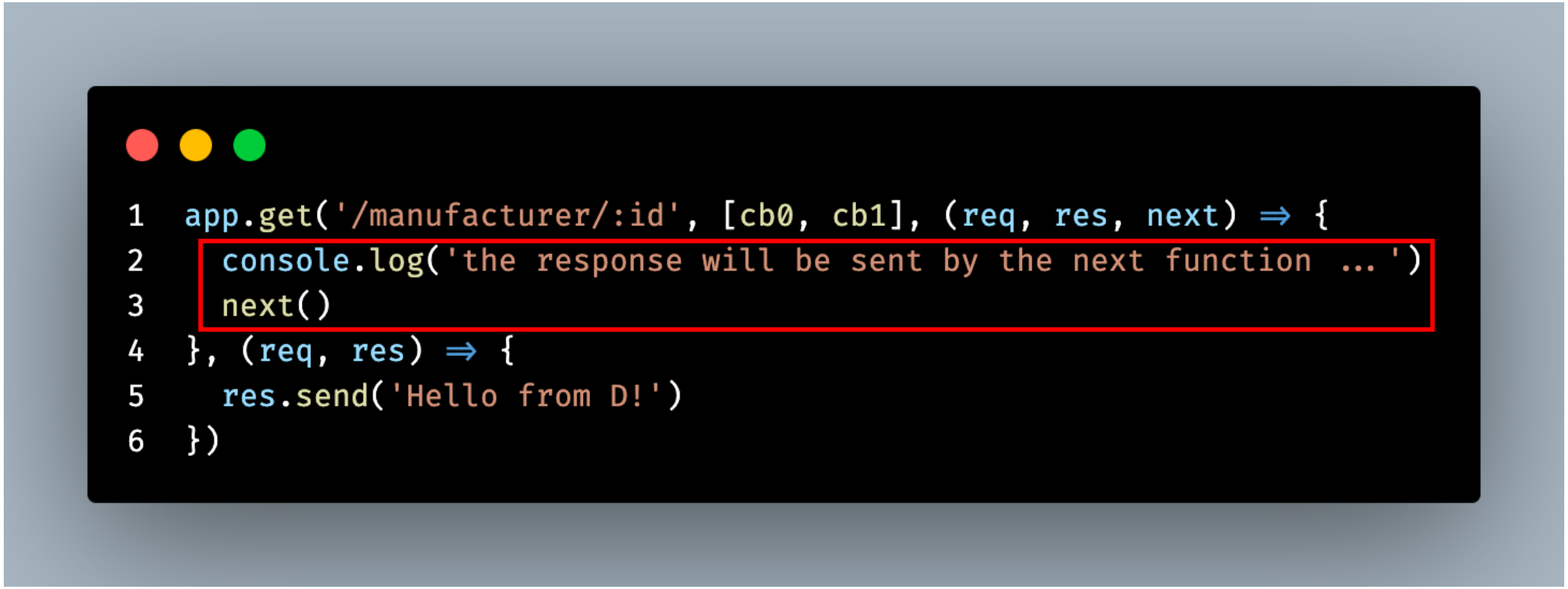


```
app.get('/manufacturer/:id', [cb0, cb1], (req, res, next) ⇒ {
  console.log('the response will be sent by the next function ...')
  next()
}, (req, res) ⇒ {
  res.send('Hello from D!')
})
```

Express – Chain Routing

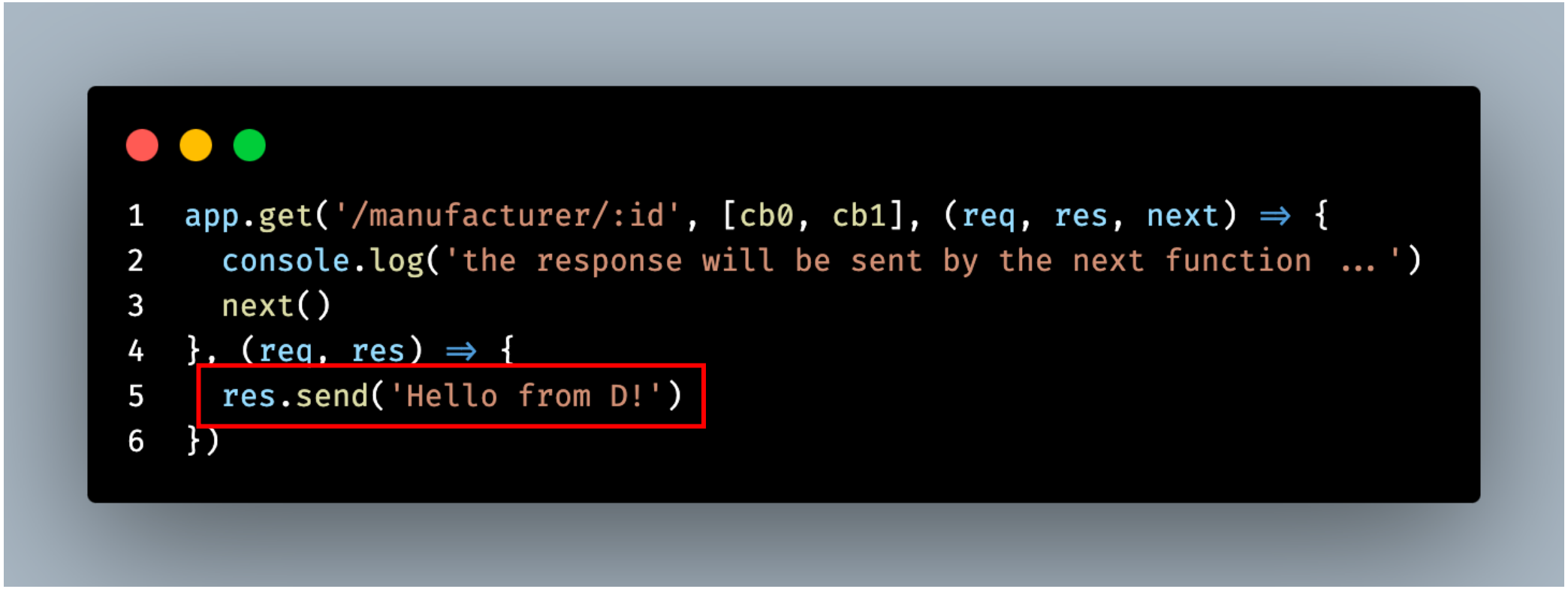
```
1  app.get('/manufacturer/:id', [cb0, cb1], (req, res, next) ⇒ {
2    console.log('the response will be sent by the next function ...')
3    next()
4  }, (req, res) ⇒ {
5    res.send('Hello from D!')
6  })
```

Express – Chain Routing



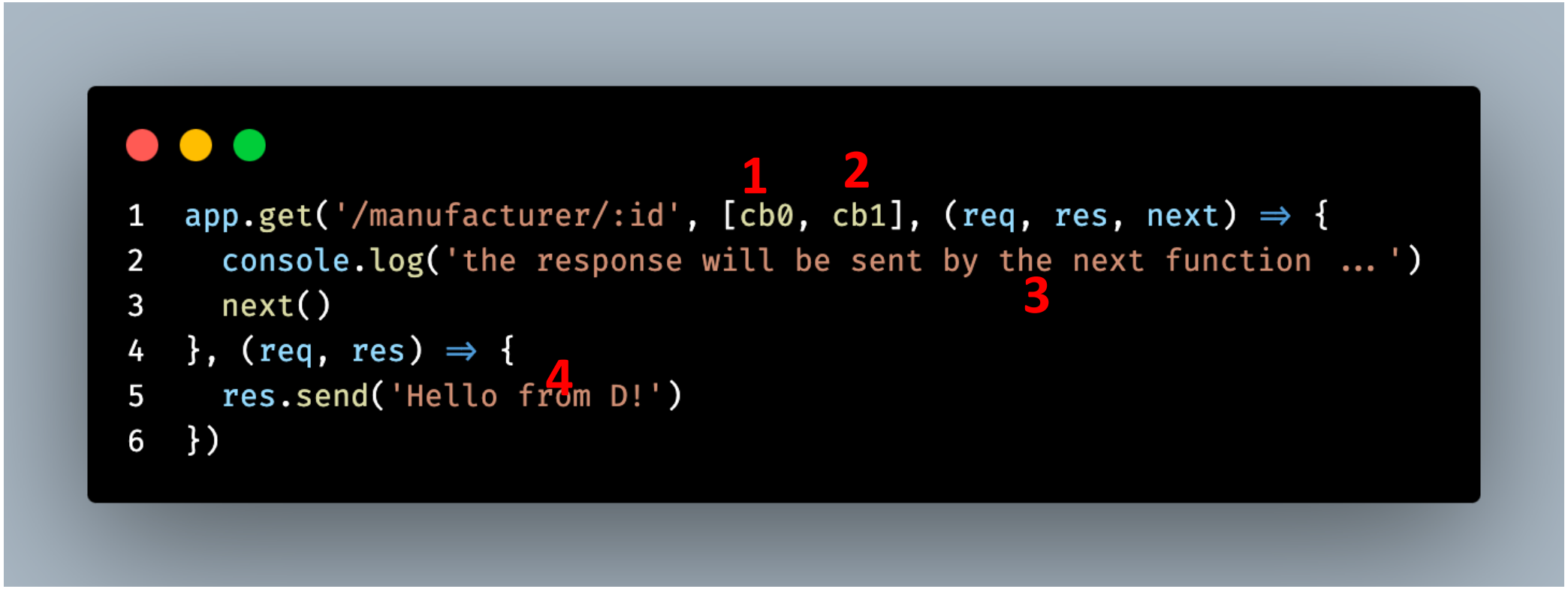
```
1  app.get('/manufacturer/:id', [cb0, cb1], (req, res, next) ⇒ {
2    console.log('the response will be sent by the next function ...')
3    next()
4  }, (req, res) ⇒ {
5    res.send('Hello from D!')
6  })
```

Express – Chain Routing



```
1  app.get('/manufacturer/:id', [cb0, cb1], (req, res, next) ⇒ {
2    console.log('the response will be sent by the next function ...')
3    next()
4  }, (req, res) ⇒ {
5    res.send('Hello from D!')
6  })
```

Express – Chain Routing



Task 8

Define a car API

Express - Middlewares

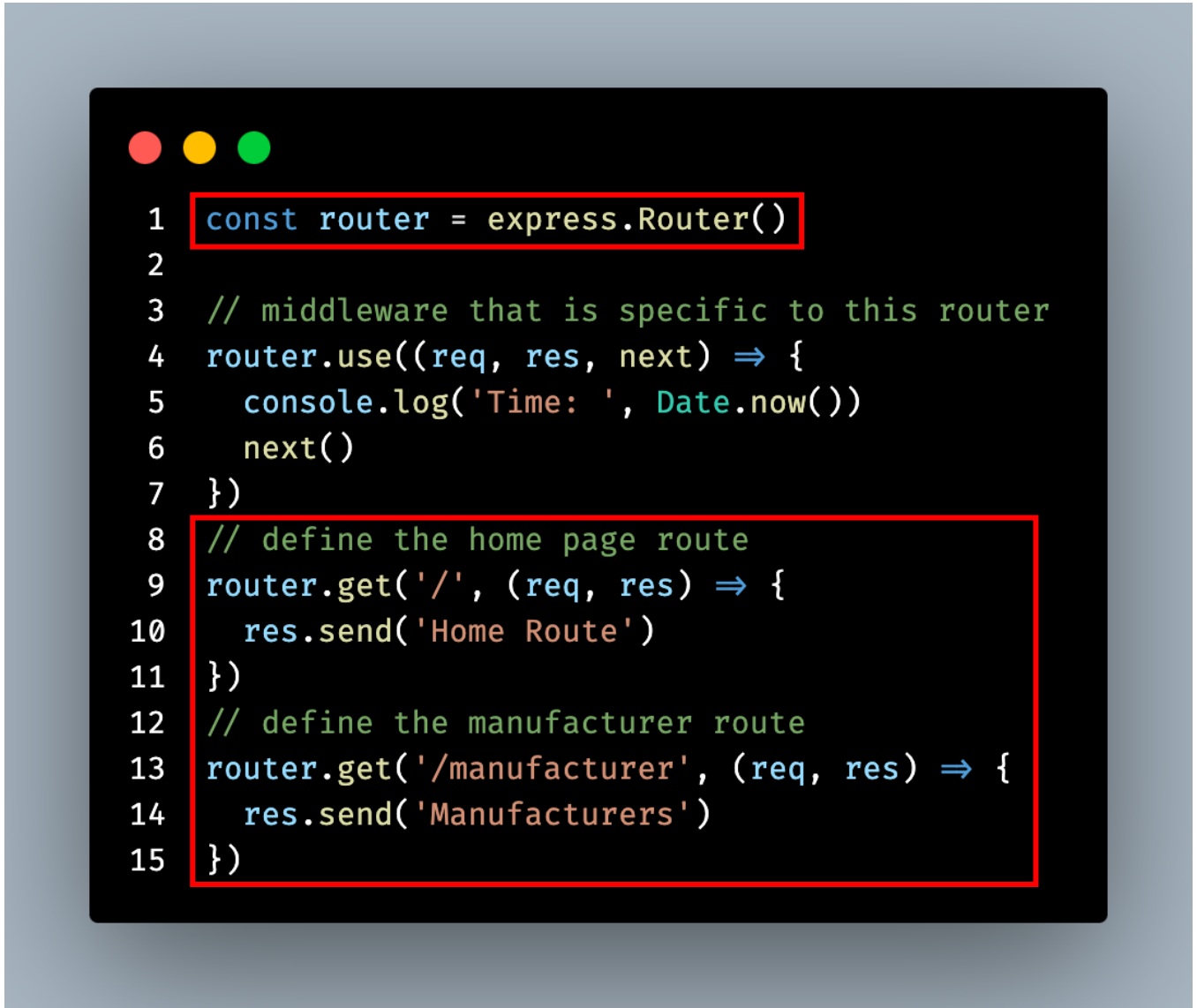
Express - Middlewares

- Nearly everything attached to the ”app” scope are middlewares
- Middlewares can be attached to most Express objects like:
  - Global “app”
  - “Router” object
  - Route scope

Express - Middlewares

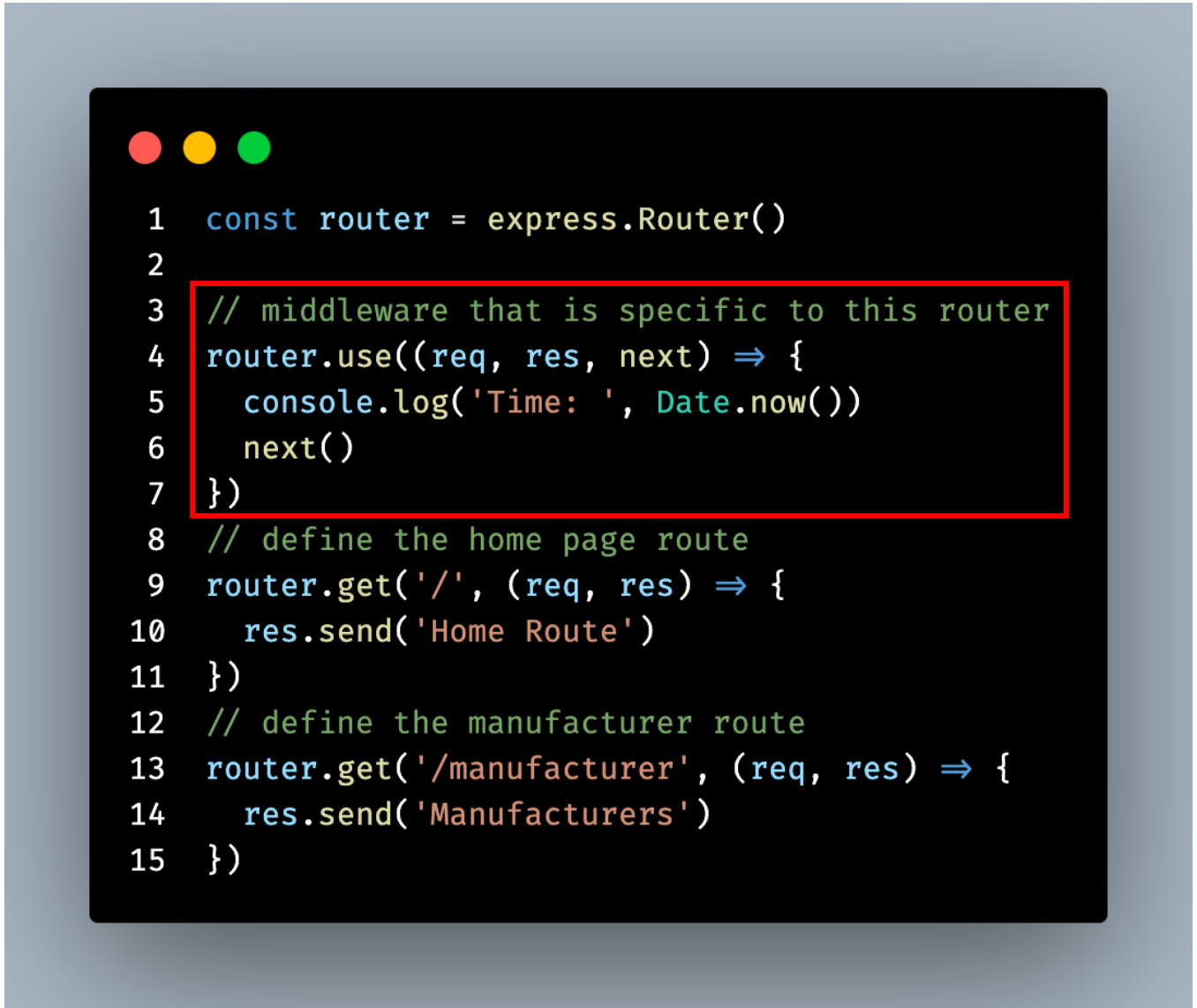
- Nearly everything attached to the ”app” scope are middlewares
- Middlewares can be attached to most Express objects like:
  - Global “app”
  - “Router” object
  - Route scope
- Route functions also just Middlewares

Express - Middlewares



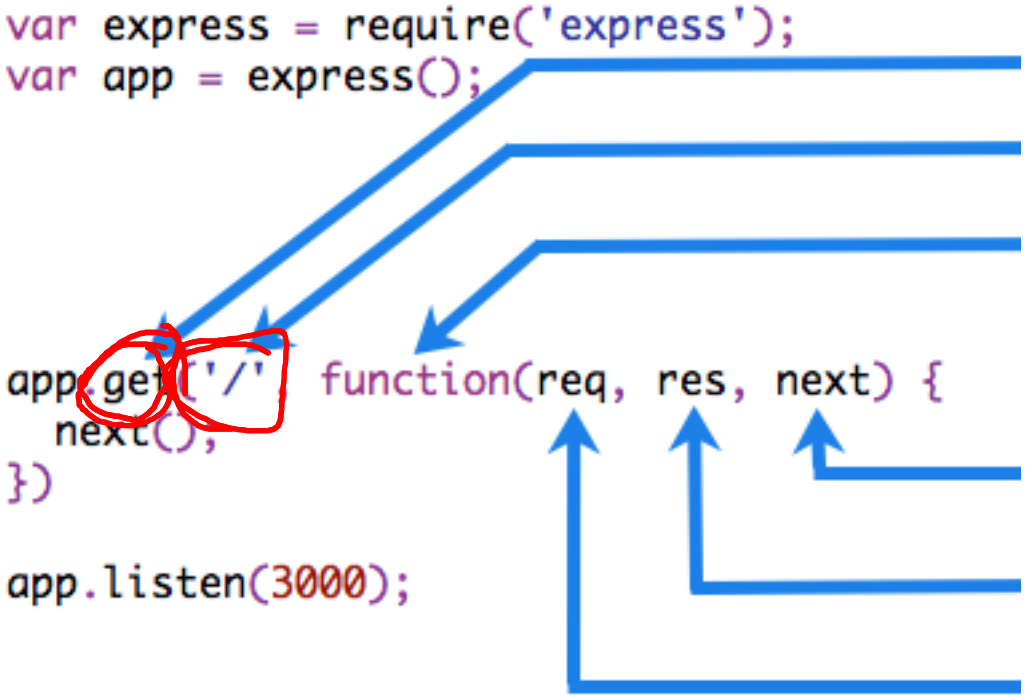
```
1  const router = express.Router()
2
3  // middleware that is specific to this router
4  router.use((req, res, next) => {
5    console.log('Time: ', Date.now())
6    next()
7  })
8  // define the home page route
9  router.get('/', (req, res) => {
10   res.send('Home Route')
11 })
12 // define the manufacturer route
13 router.get('/manufacturer', (req, res) => {
14   res.send('Manufacturers')
15 })
```

Express - Middlewares



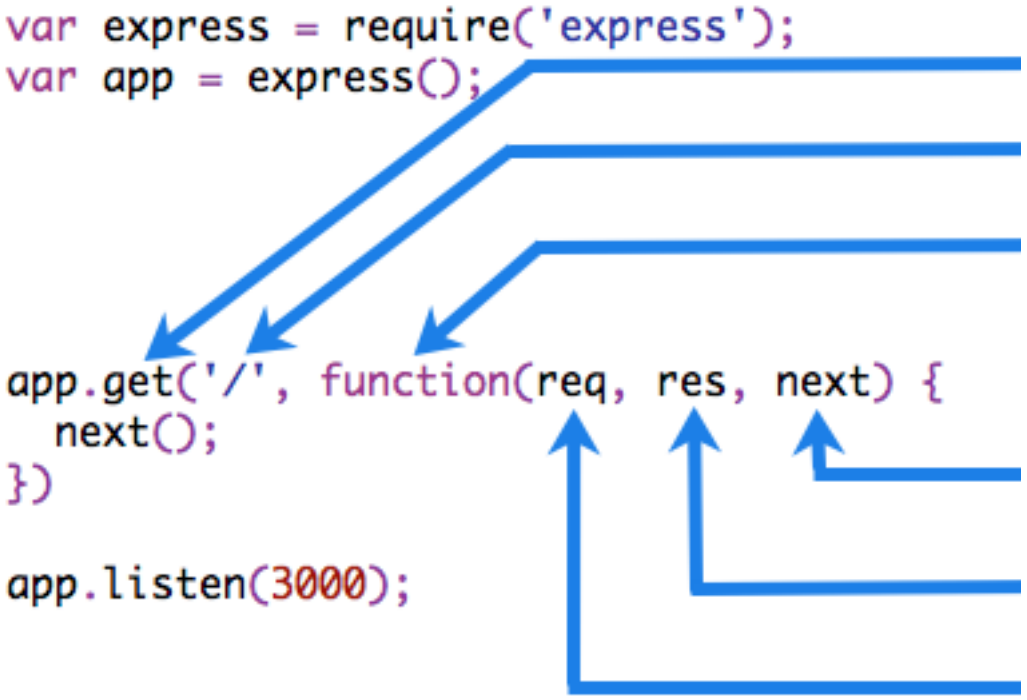
```
const router = express.Router()

// middleware that is specific to this router
router.use((req, res, next) => {
  console.log('Time: ', Date.now())
  next()
})
// define the home page route
router.get('/', (req, res) => {
  res.send('Home Route')
})
// define the manufacturer route
router.get('/manufacturer', (req, res) => {
  res.send('Manufacturers')
})
```

Express - Middlewares



HTTP method for which the middleware function applies.

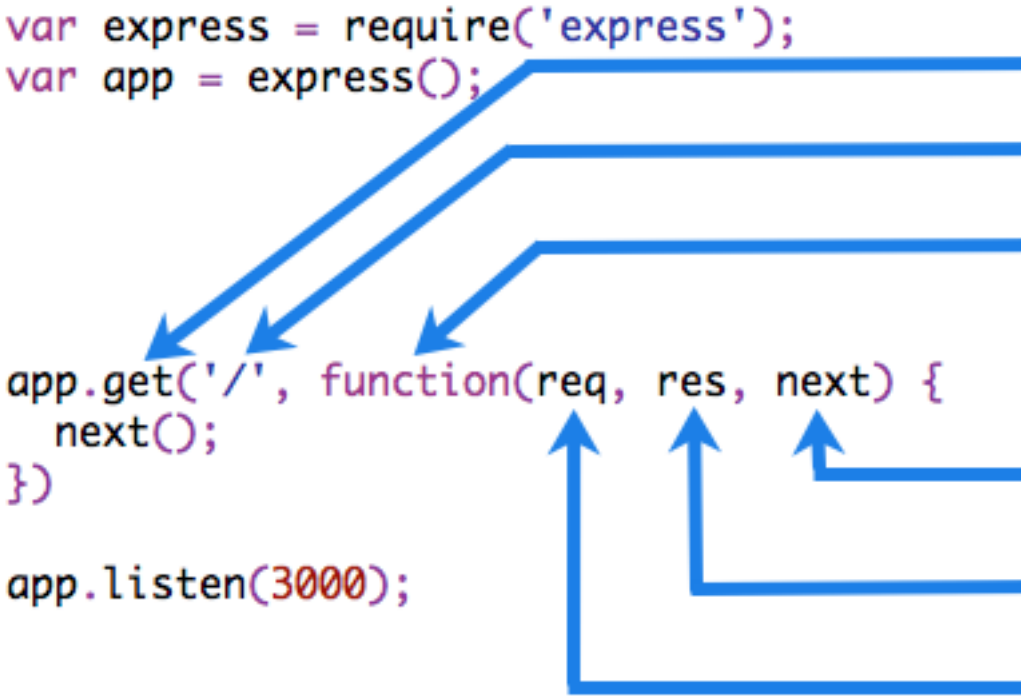
Express - Middlewares



HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

Express - Middlewares

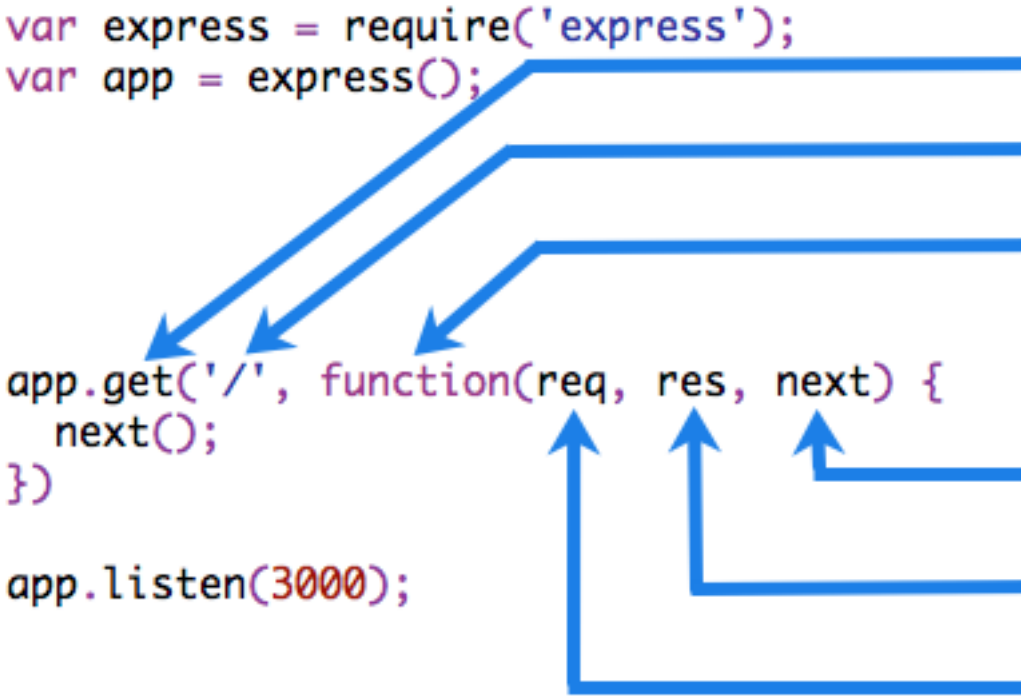


HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Express - Middlewares



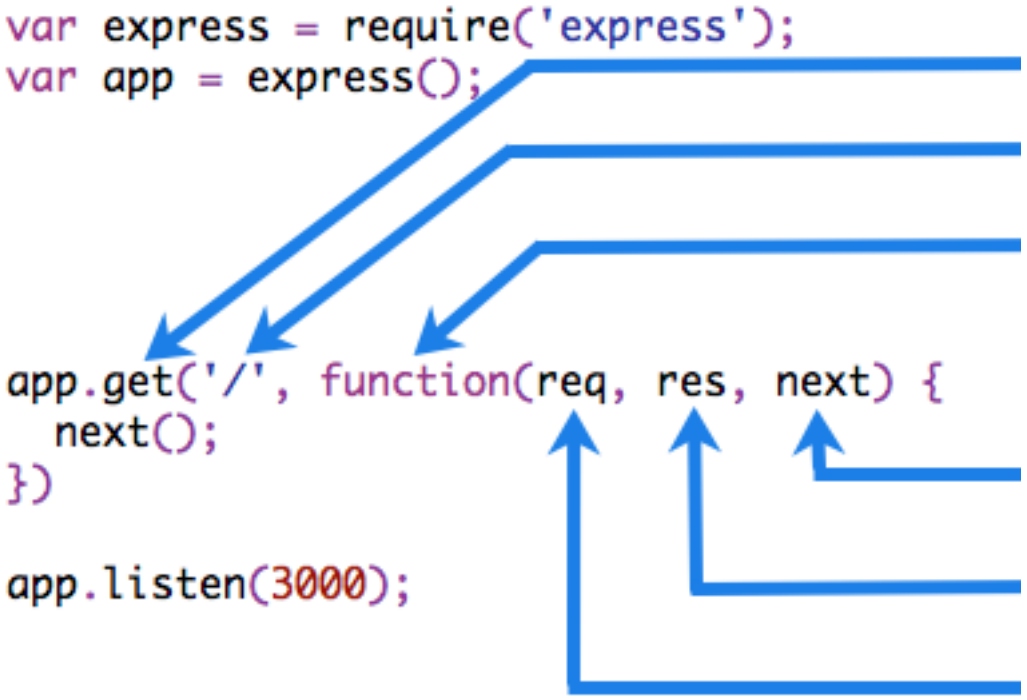
HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

Express - Middlewares



HTTP method for which the middleware function applies.

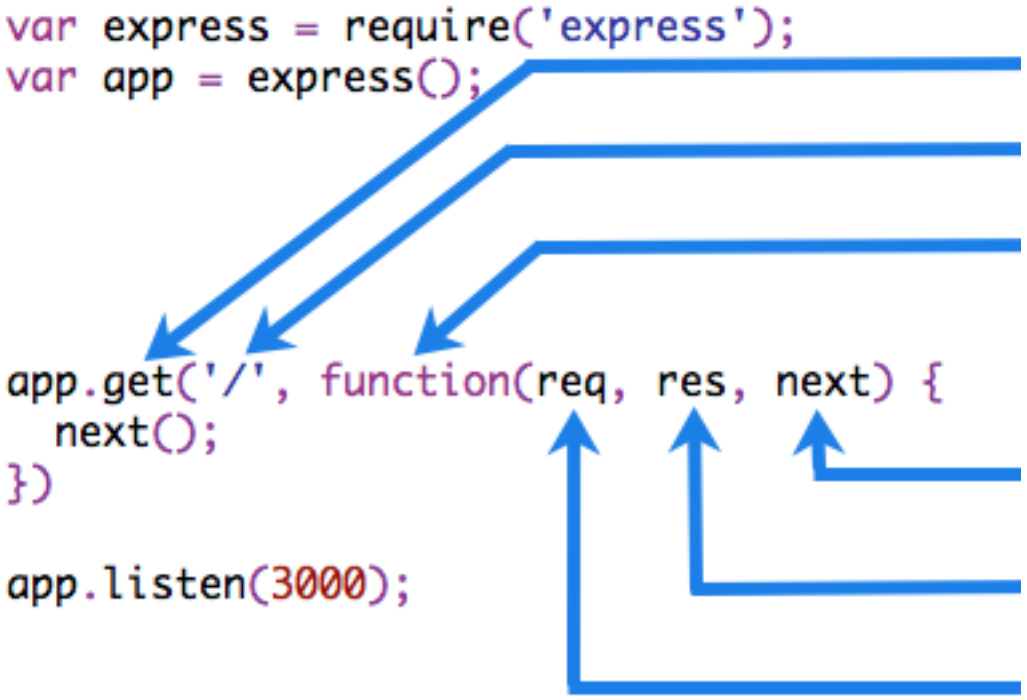
Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

Express - Middlewares



HTTP method for which the middleware function applies.

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function, called "next" by convention.

HTTP response argument to the middleware function, called "res" by convention.

HTTP request argument to the middleware function, called "req" by convention.

Task 9

Create a global timestamp middleware

Security for Express Apps

Checklist for a secure Express App

- Don’t use deprecated or outdated Express versions
- Use TLS
- Use Helmet
- Use cookies the right way
- Ensure you use safe dependencies

Deprecated and outdated Express versions

- Latest Express Version: 4.x
- Check security update page

Use TLS

- Avoid Man-in-the-middle attacks
- Avoid packet sniffing
- Many recommended security settings for most web servers
- Free SSL certificate: Let’s encrypt

Use Helmet

- Helps protecting against common HTTP vulnerabilities
- Collection of middleware functions
  - Sets “Content-Security-Policy” to prevent cross-site-scripting
  - Sets “Strict-Transport-Security” to enforce SSL connections via HTTPS
  - Sets “X-Frame-Options” to provide clickjacking protection

Use Helmet

```
1  npm install helmet --save
```

```
1  const helmet = require('helmet')
2  app.use(helmet())
```

Cookie Handling

- Don’t use default cookie names
- Ensure cookies only send via HTTPS
- Specify the domain for which the cookie is valid
- Set the cookie path to ensure the right path is using the cookie
- Set an expiry for the cookie

Cookie Handling

```
const session = require('cookie-session')
const express = require('express')
const app = express()

const expiryDate = new Date(Date.now() + 60 * 60 * 1000) // 1 hour
app.use(session({
  name: 'session',
  keys: ['key1', 'key2'],
  cookie: {
    secure: true,
    httpOnly: true,
    domain: 'example.com',
    path: 'foo/bar',
    expires: expiryDate
  }
}))
```

Audit your dependencies

- Use NPM audit
- Avoid known vulnerabilities
  - Using Github Advisory Database
- Take a look at the NodeJS security checklist

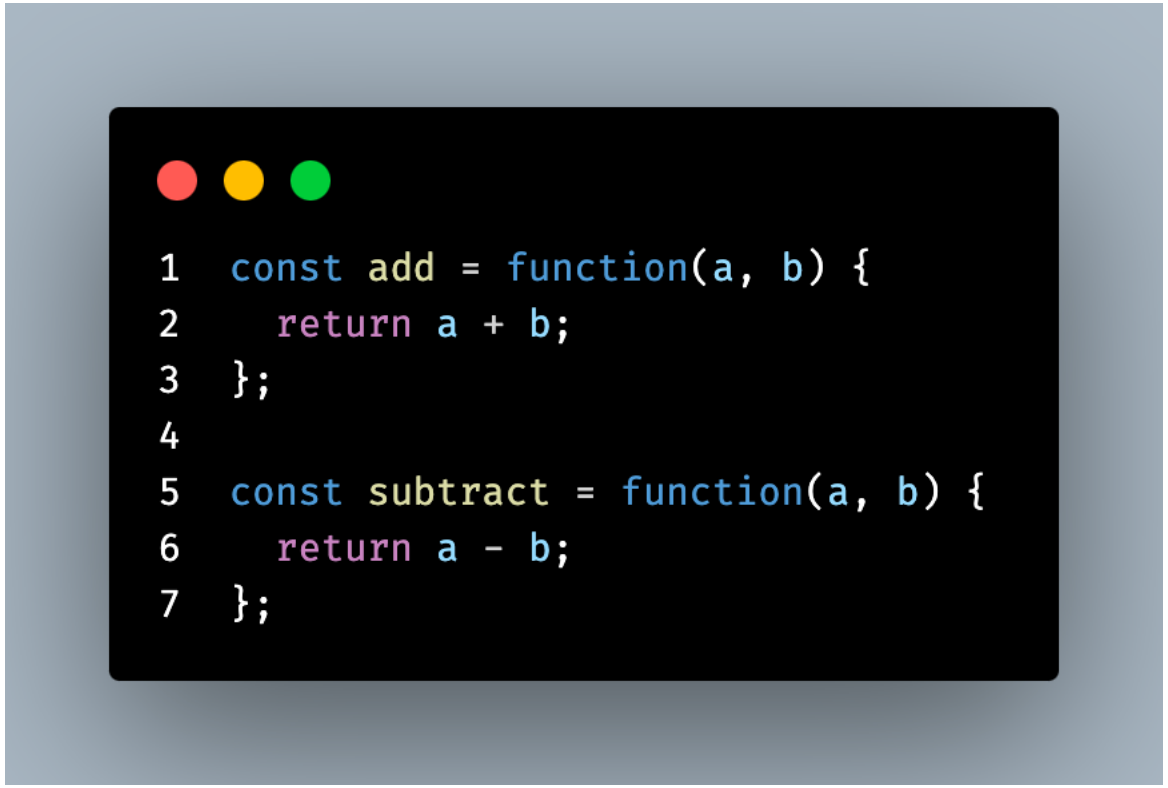
Custom Error Messages

- Set custom error messages in express
- 404 and 500 e.q. are supported

Task 10

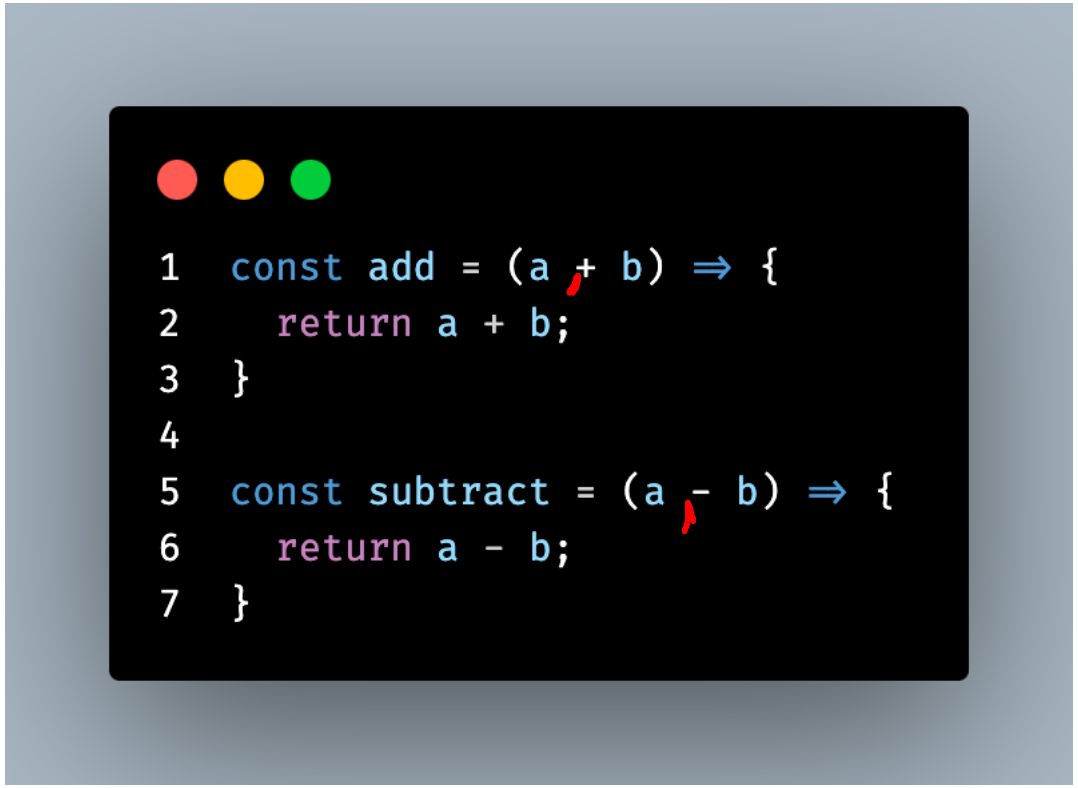
Configure Helmet

Javascript Syntax - Functions

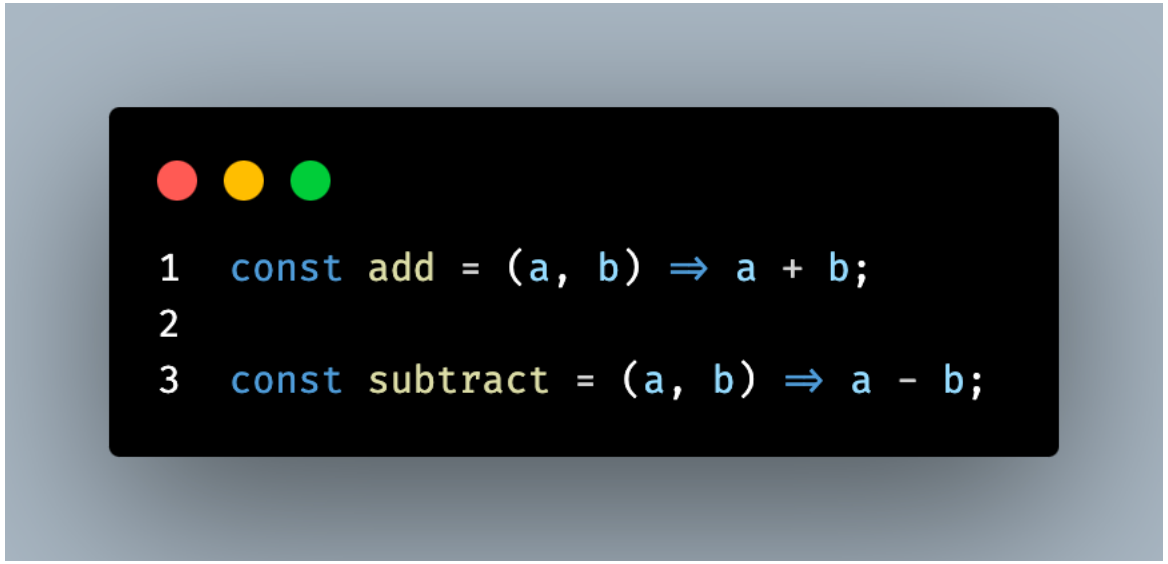


```
1  const add = function(a, b) {
2    return a + b;
3  };
4
5  const subtract = function(a, b) {
6    return a - b;
7  };
```

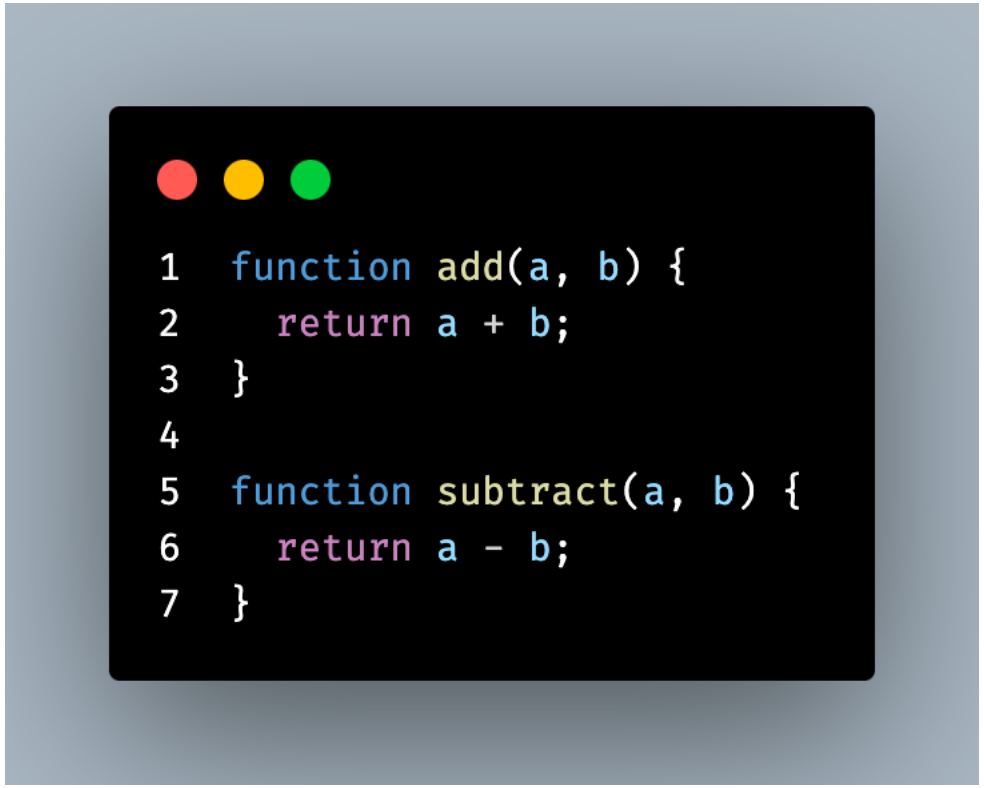
Javascript Syntax - Functions



Javascript Syntax - Functions



Javascript Syntax - Functions



Javascript Syntax - Functions

Javascript Syntax - Functions

```
1  router.get('/', (req, res, next) ⇒ {
2    res.render('index');
3  });
```

```
1  router.get('/', function (req, res, next) {
2    res.render('index');
3  });
```

Javascript Syntax - Functions

```
1  router.get('/', (req, res, next) ⇒ {
2    res.render('index');
3  });
```

```
1  router.get('/', function (req, res, next) {
2    res.render('index');
3  });
```

```
1  const handler = (req, res, next) ⇒ {
2    res.render('index');
3  }
4
5  router.get('/', handler);
```

Javascript Syntax - Functions

```
1  router.get('/', (req, res, next) => {
2    res.render('index');
3  });
```

```
1  router.get('/', function (req, res, next) {
2    res.render('index');
3  });
```

```
1  const handler = (req, res, next) => {
2    res.render('index');
3  }
4
5  router.get('/', handler);
```

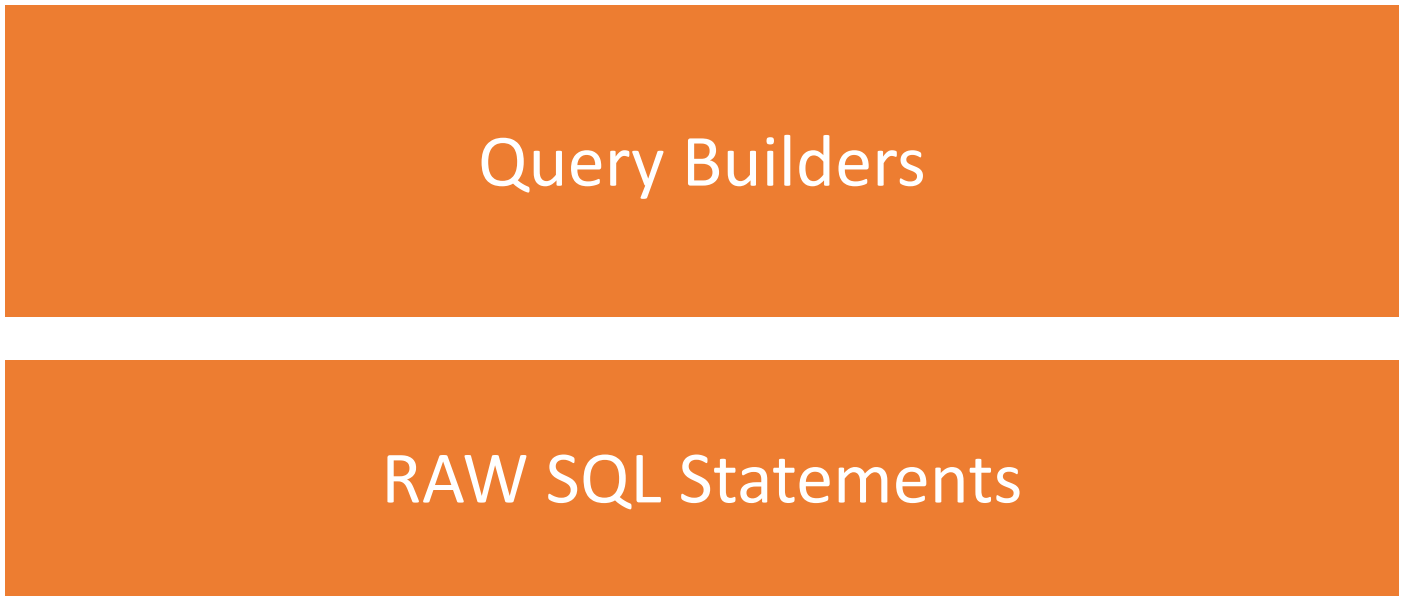
```
1  router.get('/', (req, res, next) => res.render('index'));
```

Working with Databases

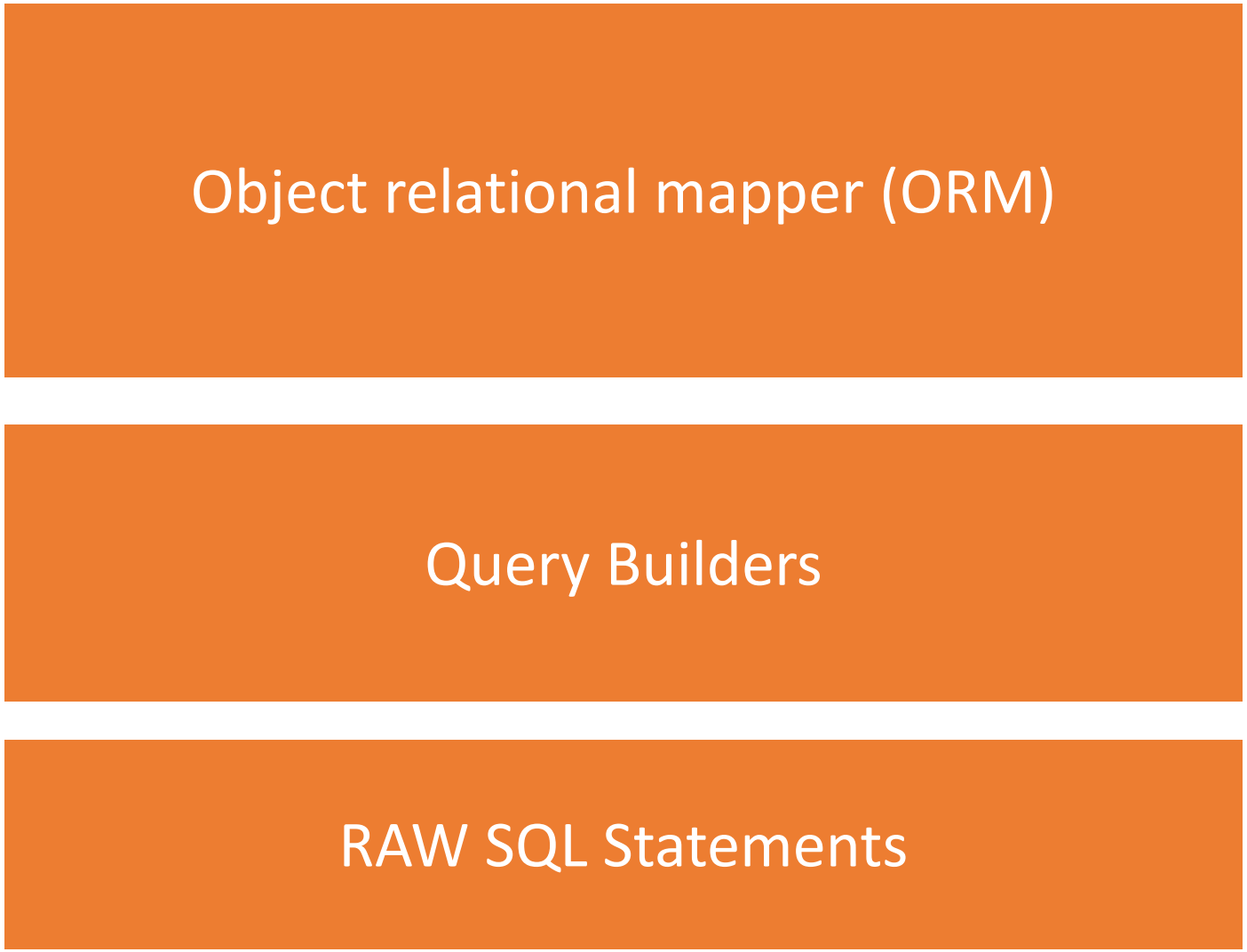
Stages of database interaction

RAW SQL Statements

Stages of database interaction



Stages of database interaction



Working with databases

- It’s recommended to use an ORM for connecting to databases
- If your API is pretty small you can still use a query builder only
- There are many ORMs out there for NodeJS

NodeJS ORMs – Most common ones

- Prisma
- Sequelize
- TypeORM

Prisma ORM

Prisma

- A ORM containing multiple tools
- Including a CLI
- Works with multiple databases
  - PostgreSQL
  - MySQL
  - SQLite
  - MongDB
  - CockroachDB
  - …

Prisma

Defining Prisma Schema

Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:../dev.db"
}

model Manufacturer {
  id           Int    @id @default(autoincrement())
  name         String
  foundingYear Int
  headquarters String
  cars         Car[]
}

model Car {
  id              Int          @id @default(autoincrement())
  model           String
  yearBuilt       Int
  price           Float
  manufacturer_id Int?
  manufacturer    Manufacturer? @relation(fields: [manufacturer_id], references: [id], onDelete: NoAction, onUpdate: NoAction)
}

```

Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:../dev.db"
}

model Manufacturer {
  id           Int    @id @default(autoincrement())
  name         String
  foundingYear Int
  headquarters String
  cars         Car[]
}

model Car {
  id              Int          @id @default(autoincrement())
  model           String
  yearBuilt       Int
  price           Float
  manufacturer_id Int?
  manufacturer    Manufacturer? @relation(fields: [manufacturer_id], references: [id], onDelete: NoAction, onUpdate: NoAction)
}

```

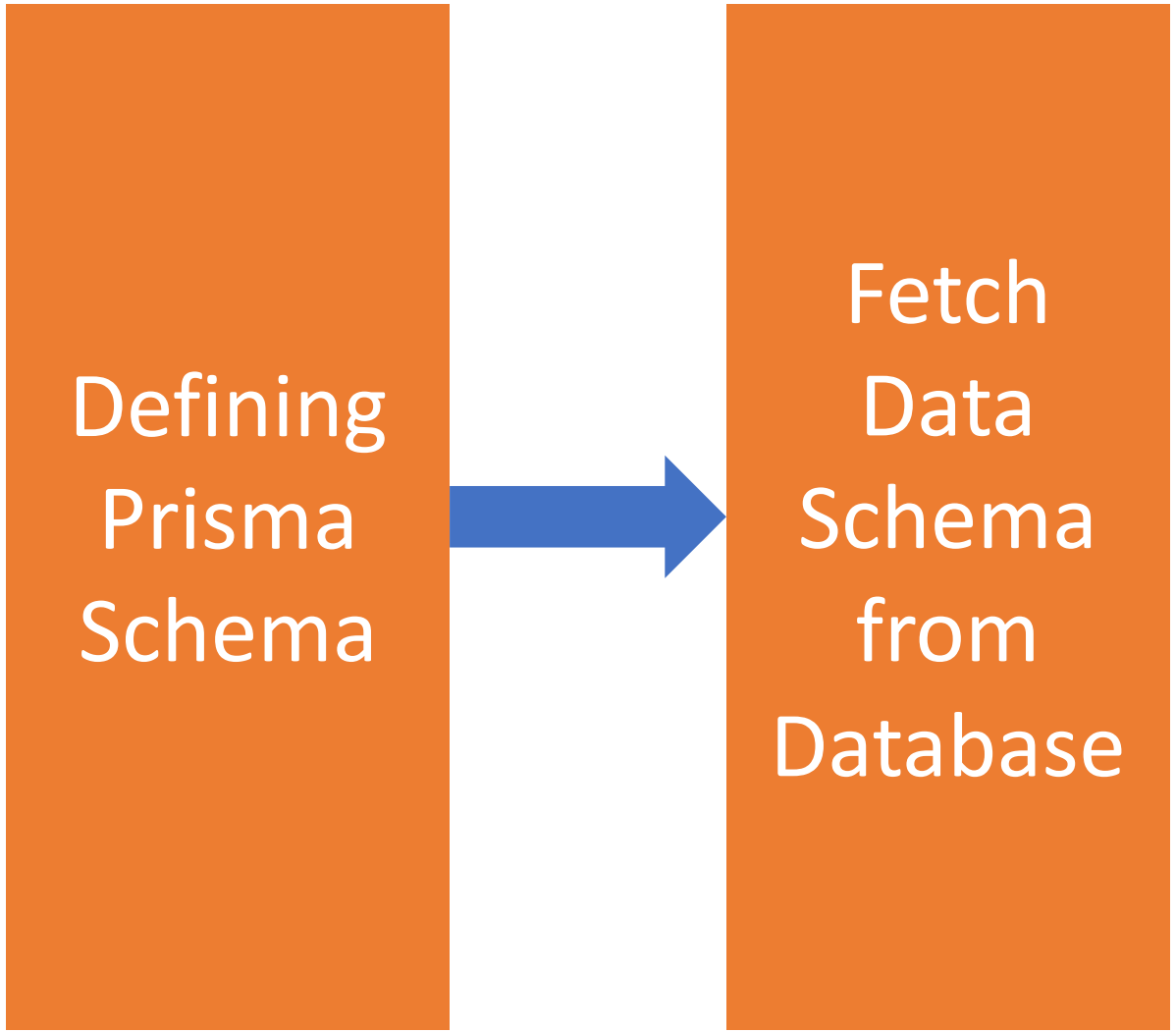
Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:../dev.db"
}

model Manufacturer {
  id           Int    @id @default(autoincrement())
  name         String
  foundingYear Int
  headquarters String
  cars         Car[]
}

model Car {
  id              Int          @id @default(autoincrement())
  model           String
  yearBuilt       Int
  price           Float
  manufacturer_id Int?
  manufacturer    Manufacturer? @relation(fields: [manufacturer_id], references: [id], onDelete: NoAction, onUpdate: NoAction)
}

```

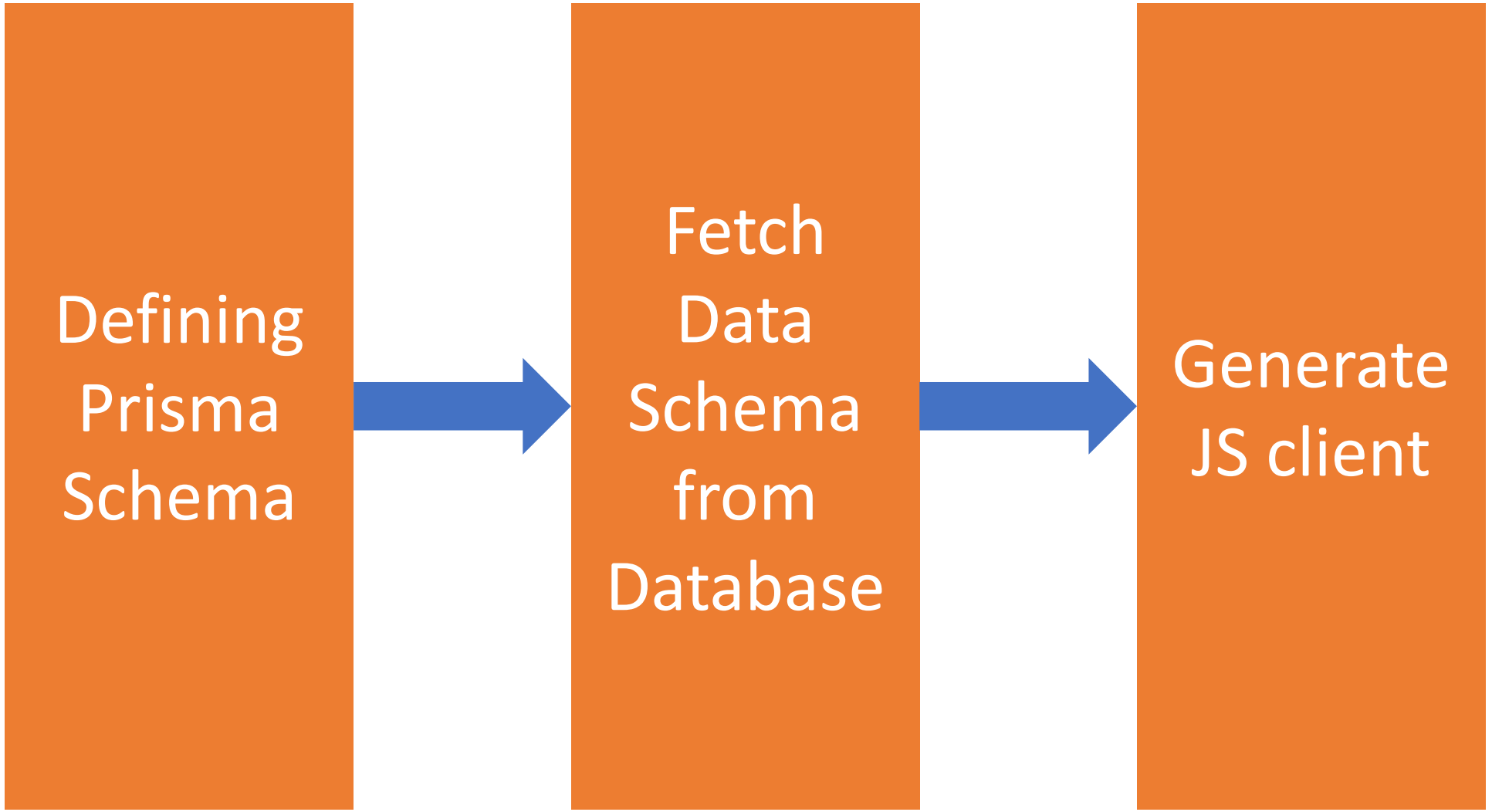
Prisma

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:../dev.db"
}

model Manufacturer {
  id           Int    @id @default(autoincrement())
  name         String
  foundingYear Int
  headquarters String
  cars         Car[]
}

model Car {
  id              Int          @id @default(autoincrement())
  model           String
  yearBuilt       Int
  price           Float
  manufacturer_id Int?
  manufacturer    Manufacturer? @relation(fields: [manufacturer_id], references: [id], onDelete: NoAction, onUpdate: NoAction)
}

```

Prisma



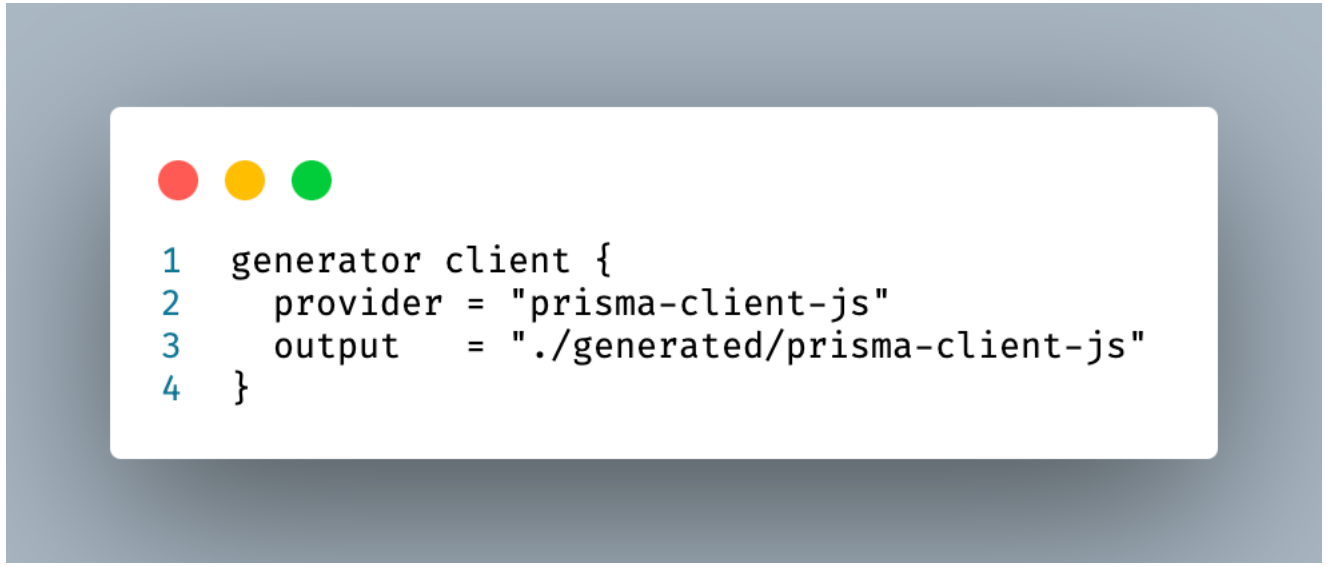
Prisma


Defining Prisma Schema
Fetch Data Schema from Database
Generate JS client

Prisma

```
# Installing Prisma
npm run install prisma --save-dev

# Prisma initialization
npx prisma init

# Updating Prisma schema with database
npx prisma db pull

# Generating Prisma client
npx prisma generate
```

Prisma


generator client {
provider = "prisma-client-js"
output = "./generated/prisma-client-js"
}

Task 11

Setup Prisma

Tooling and Documentation of APIs

Tooling and Documentation

- Code Documentation

Tooling and Documentation

- Code Documentation
  - Documenting inline code
  - Use JSDoc commenting format

Tooling and Documentation

- Code Documentation
  - Documenting inline code
  - Use JSDoc commenting format
- API Documentation
  - Swagger generated docs
  - Using “npm install express-swagger-generator”
  - Needs documentation with JSDocs

Task 12

API Docs

Authentication in Express

Using Passport

Authentication

- Used to authenticate a client against the API
- Multiple Strategies for authentication
- PassportJS as support library

Authentication

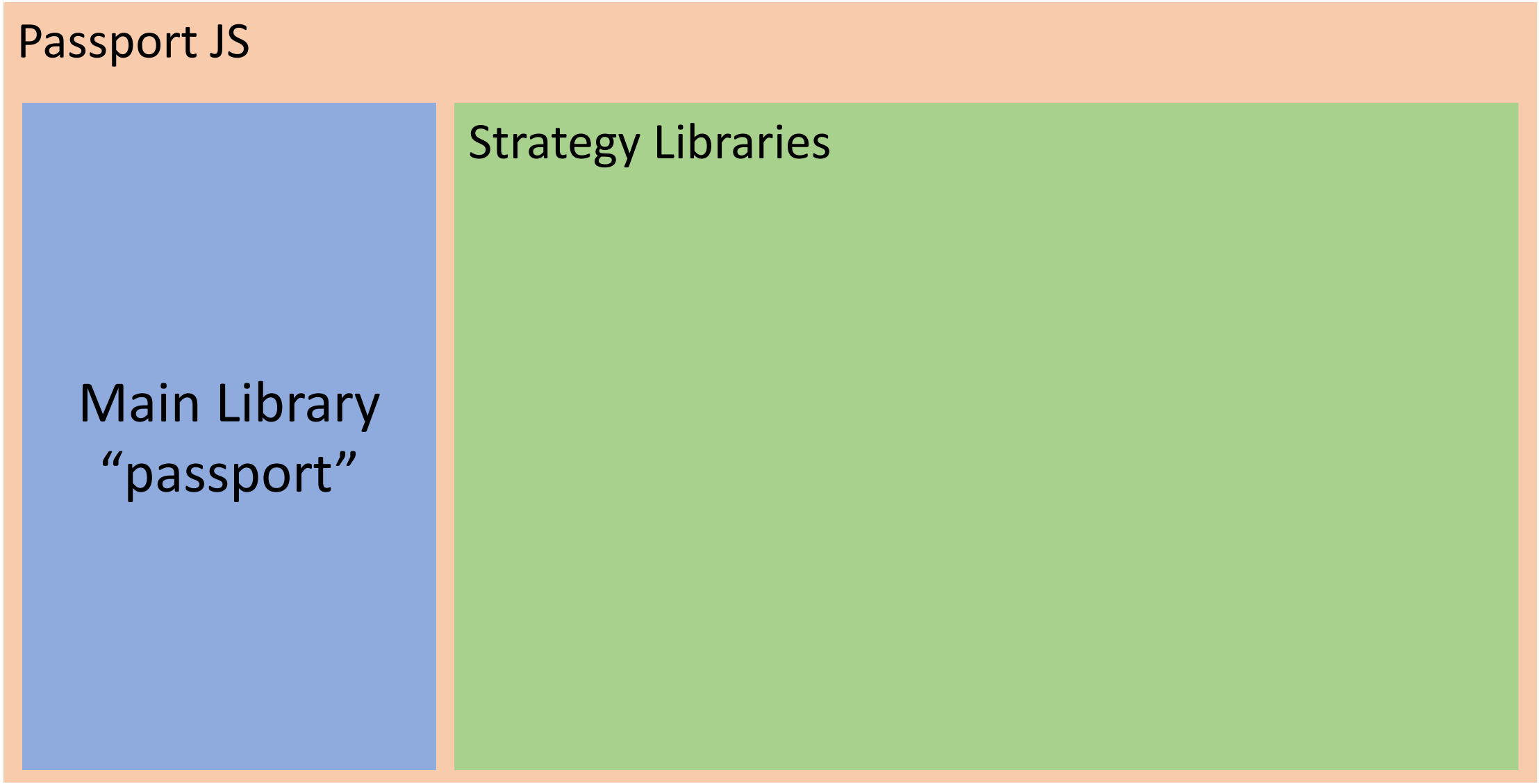
Passport JS

Authentication

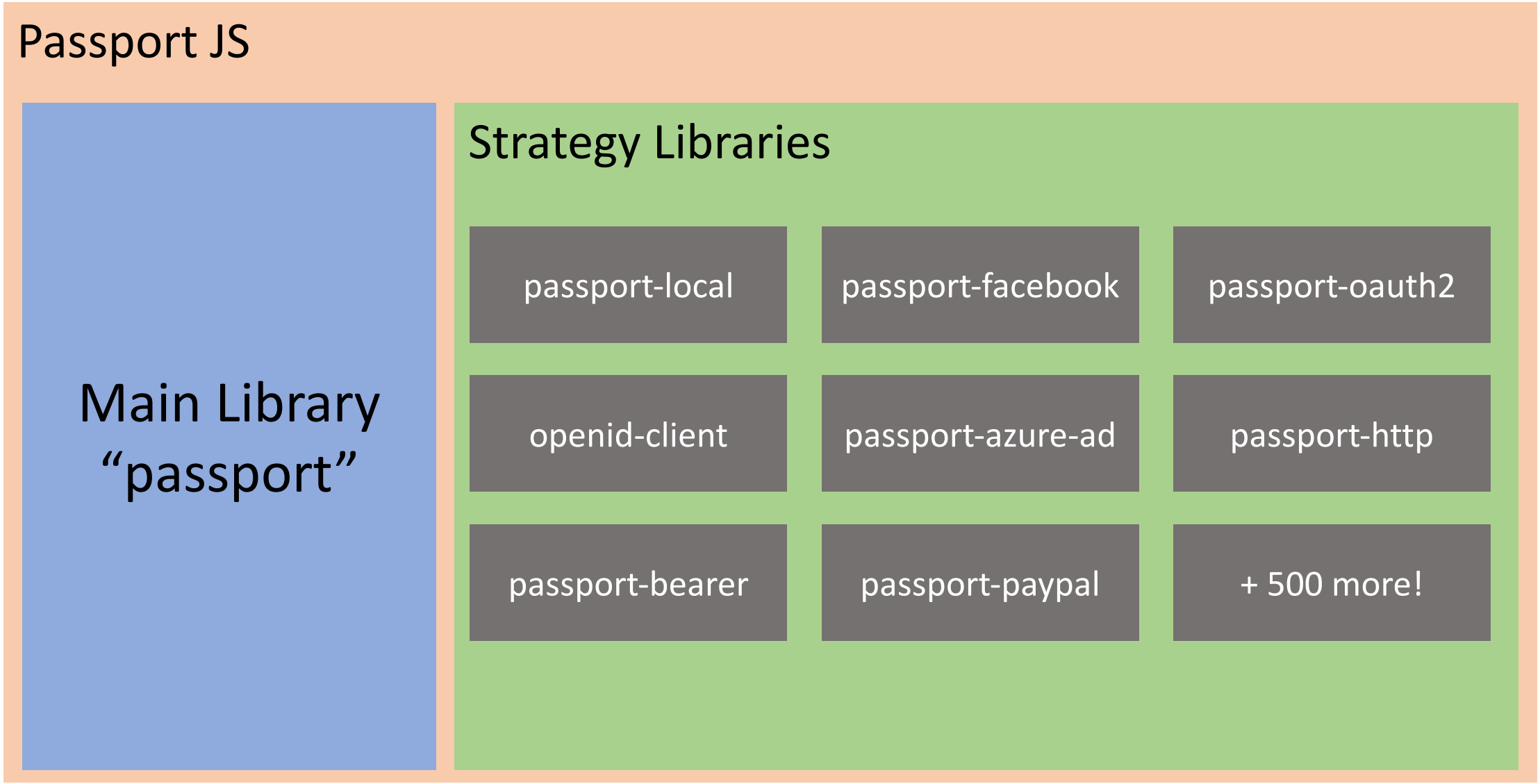
Passport JS

Main Library
“passport”

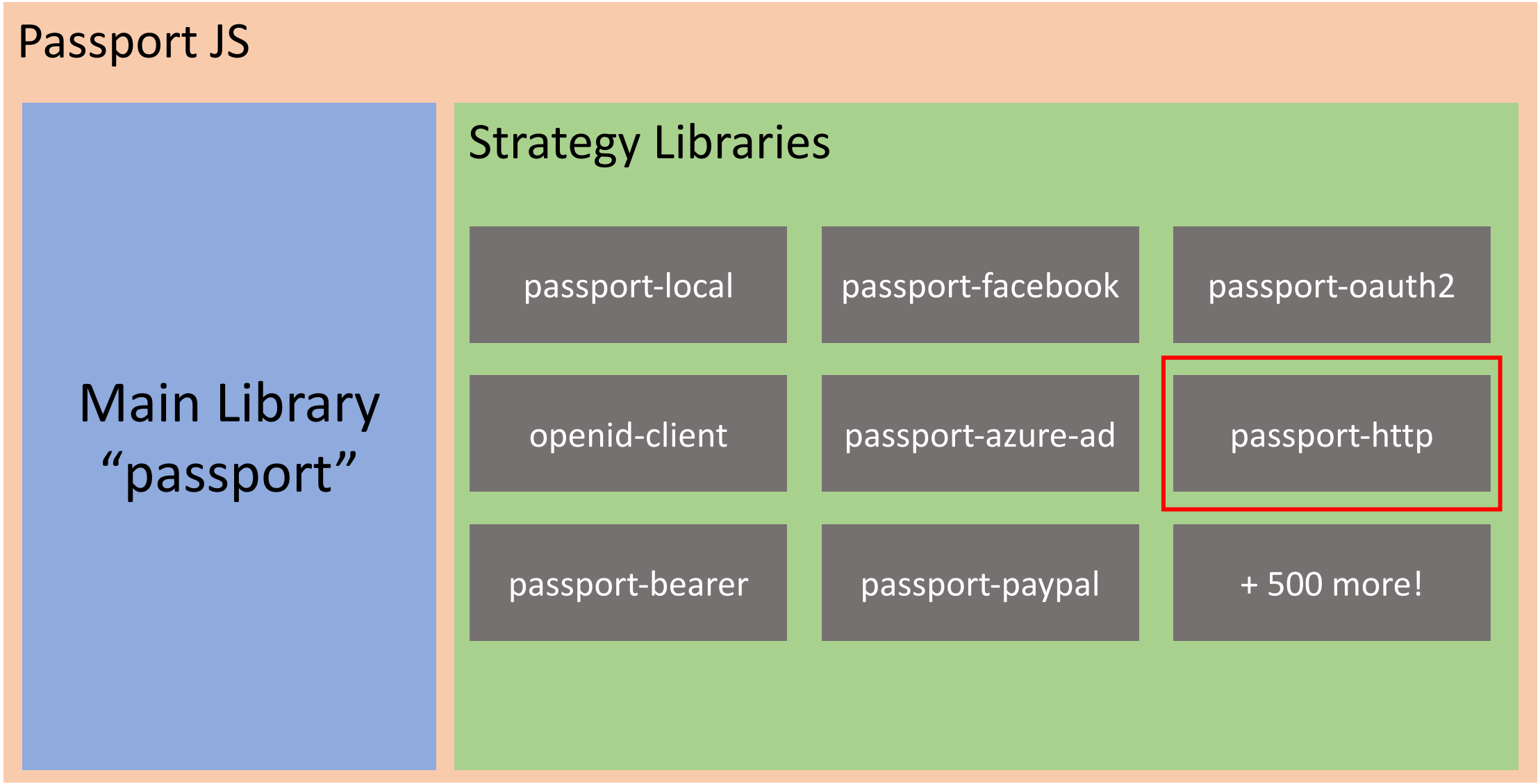
Authentication



Authentication



Authentication



Authentication

Authentication

```
1  // authentication/basic-strategy.js
2  const passport = require('passport');
3  const BasicStrategy = require('passport-http').BasicStrategy;
4
5  const passportStrategyInit = () => {
6    passport.use(new BasicStrategy(
7      function(username, password, done) {
8
9        if (username && password) {
10         return done(null, {
11           username
12         });
13       }
14
15       return done('Unauthorized', null);
16     }
17   ));
18 }
19
20 module.exports.passportStrategyInit = passportStrategyInit;
```

Authentication

```
1  const passport = require('passport');
2  const { passportStrategyInit } = require('./authentication/basic-strategy');
```

Authentication

Task 13

Authentication with Passport

API Versioning

API Versioning

- Version of APIs is important if you developing an API which a larger user base
- You should support multiple API version for a specific time period
- If you developing an API without external API users you don’t need versioning there

Server-side Rendering

E.g. with an Angular Application

Server-Side Rendering

- Two ways of server-side rendering
  - Using a View/Template Engine in Express directly

Server-Side Rendering

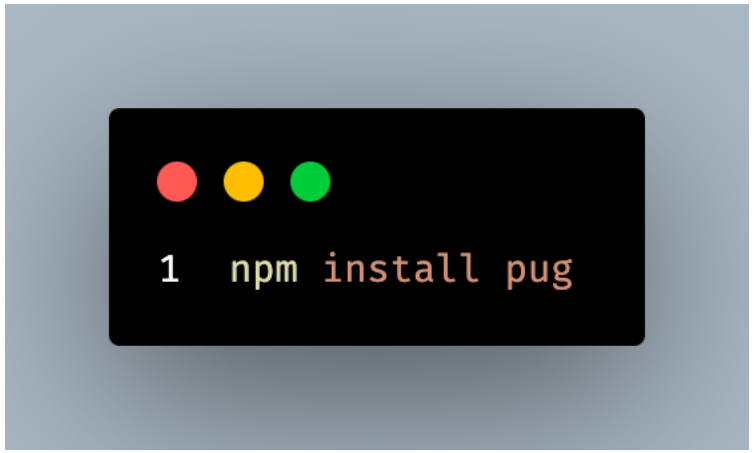
- Two ways of server-side rendering
  - Using a View/Template Engine in Express directly
  - Server-side Rendering for SPAs

Server-Side Rendering

- Two ways of server-side rendering
  - Using a View/Template Engine in Express directly
  - Server-side Rendering for SPAs

Example: Render Templates with Pug in Express

Server-Side Rendering



Server-Side Rendering

```
1  // app.js
2  const app = express();
3
4  app.set('view engine', 'pug');
```

Server-Side Rendering

```
// app.js
const app = express();

app.set('view engine', 'pug');
```

Server-Side Rendering

```
1  // app.js
2  const app = express();
3
4  app.set('view engine', 'pug');
```


```
1  // app.js
2  // app.use(express.static(path.join(__dirname, 'public')));
```

Server-Side Rendering

Server-Side Rendering

```
// routes/index.js
const express = require('express');
const router = express.Router();

/* GET home page. */
router.get('/', (req, res, next) => {
  res.render('index', { title: 'CarAPI', message: 'Welcome here!' });
});

module.exports = router;
```

Task 14

Integrate Pug View Engine

Deploy an Express App

Deploy an Express App

- We need to provide the App the variable for PORT

Deploy an Express App

- We need to provide the App the variable for PORT
- Use the package “compression” to GZIP responses

Deploy an Express App

- We need to provide the App the variable for PORT
- Use the package “compression” to GZIP responses

Deploy an Express App

- We need to provide the App the variable for PORT
- Use the package “compression” to GZIP responses
- Optional: Use a Bundler

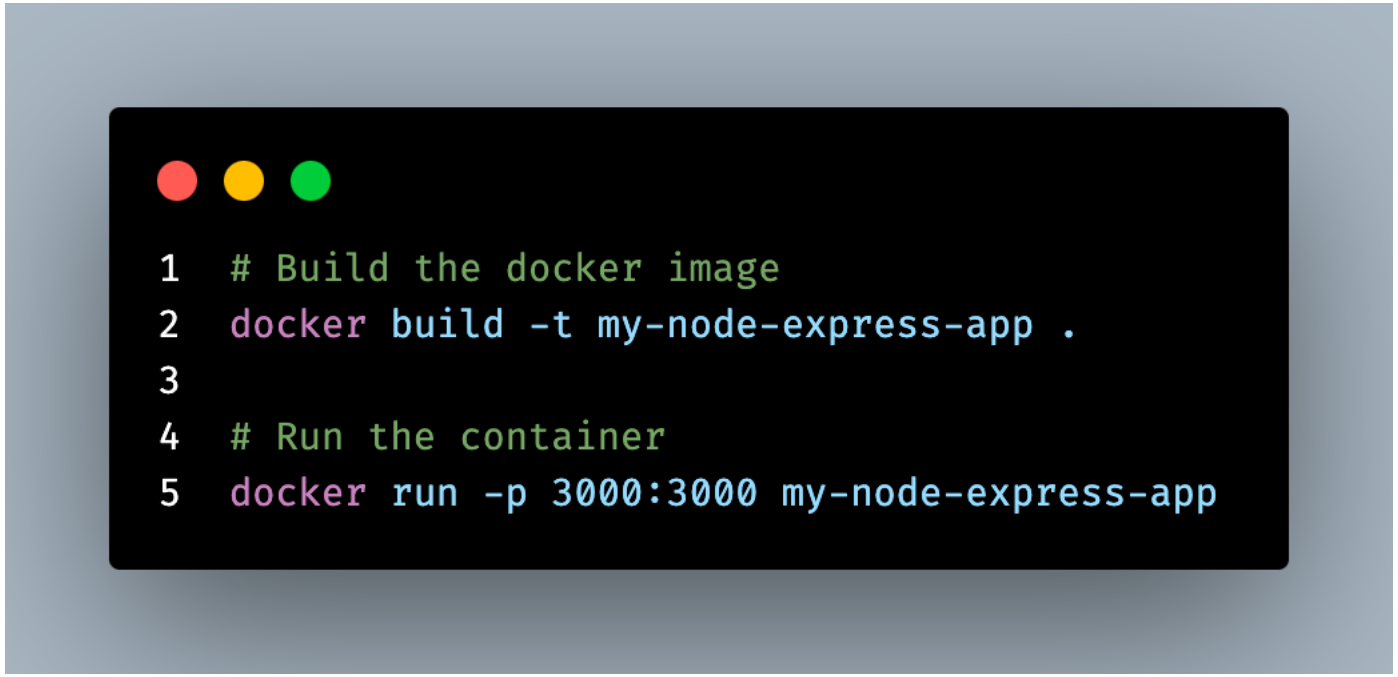
Deploy an Express App

- We need to provide the App the variable for PORT
- Use the package “compression” to GZIP responses
- Optional: Use a Bundler
- Create a dockerfile for your application to deploy it everywhere

Create Dockerfile

```
# Verwenden Sie ein offizielles Node.js-Image als Basis
FROM node:18

# Erstellen Sie das Verzeichnis für Ihre Anwendung im Container
WORKDIR /usr/src/app

# Kopieren Sie die package.json- und package-lock.json-Dateien in den Container
COPY package*.json ./

# Installieren Sie die Abhängigkeiten
RUN npm install

# Kopieren Sie den Rest Ihrer Anwendung in den Container
COPY . .

# Öffnen Sie den Port, auf dem Ihre Express-Anwendung laufen wird
EXPOSE 3000

# Starten Sie Ihre Express-Anwendung
CMD ["node", "bin/www"]
```

Build the Image and Run it



```
# Build the docker image
docker build -t my-node-express-app .

# Run the container
docker run -p 3000:3000 my-node-express-app
```

Give me feedback!

Contact Data

- Mail: mail@aaronczichon.de
- Web: https://aaronczichon.de
- LinkedIn: https://www.linkedin.com/in/aaron-czichon-3526a87a/