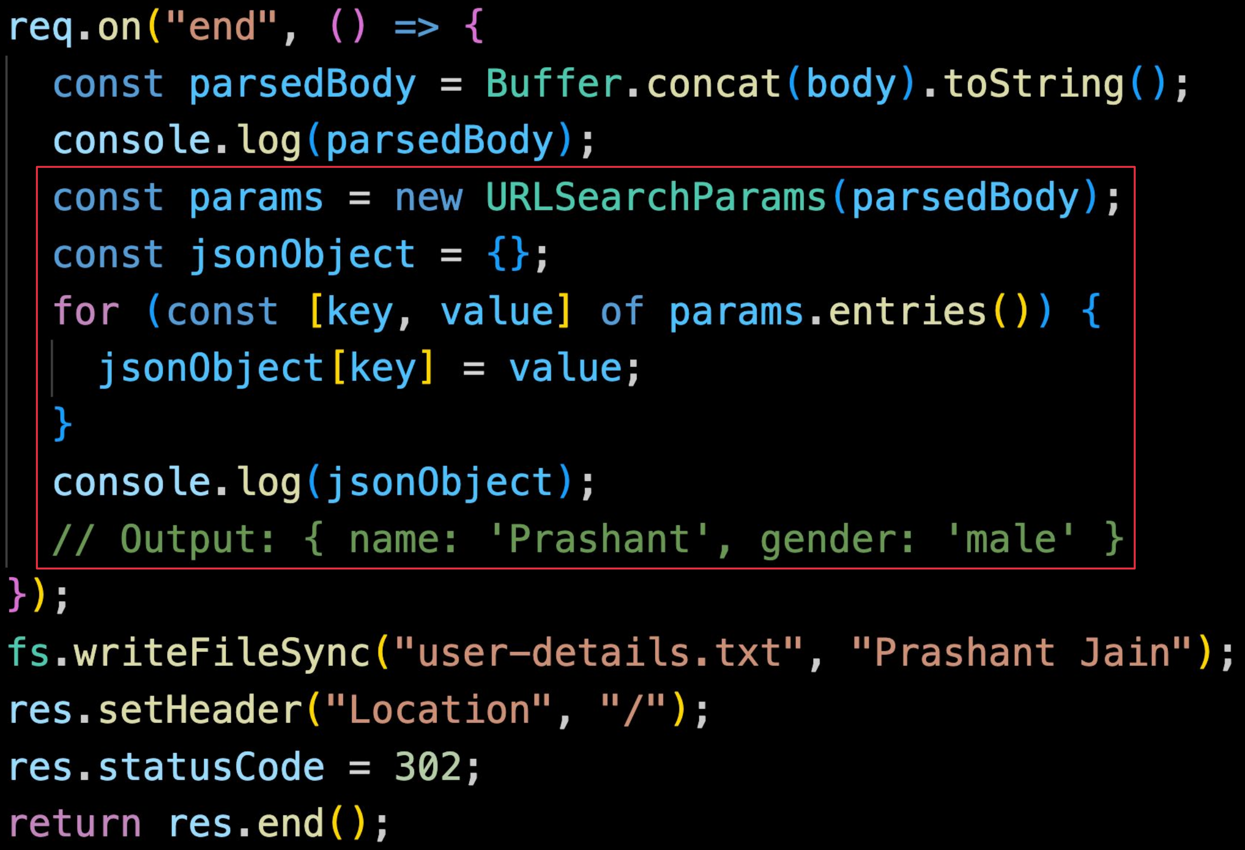
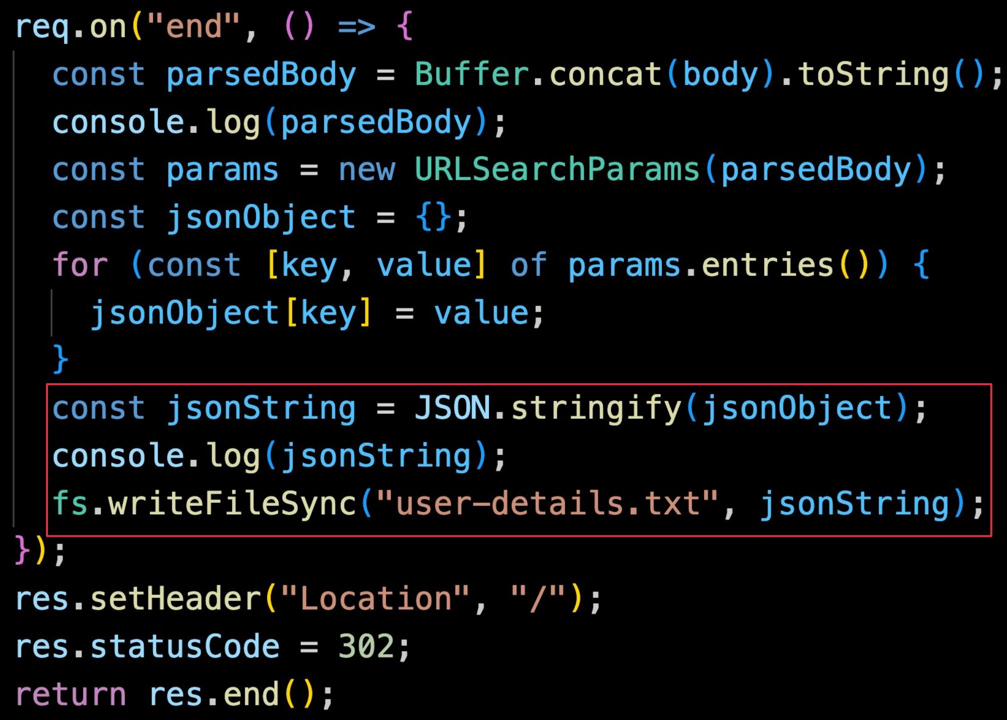
5.6 Parsing Request

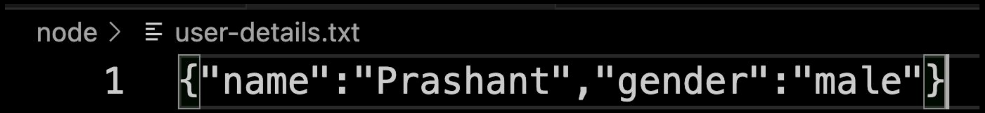


```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  console.log(jsonObject);
  // Output: { name: 'Prashant', gender: 'male' }
});
fs.writeFileSync("user-details.txt", "Prashant Jain");
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```

5.6 Parsing Request



```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
});
res.setHeader("Location", "/");
res.statusCode = 302;
return res.end();
```

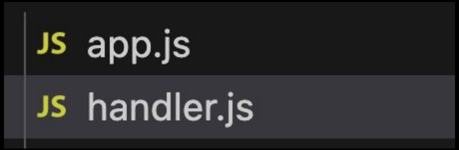


node > user-details.txt

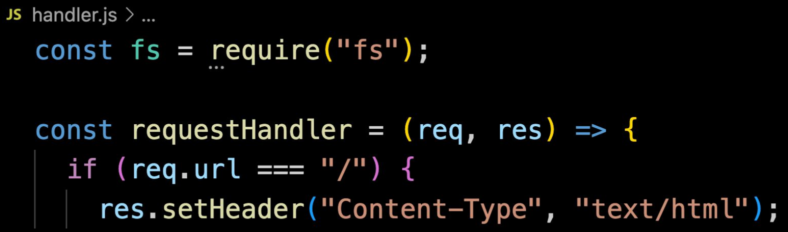
```
{"name":"Prashant","gender":"male"}
```

React

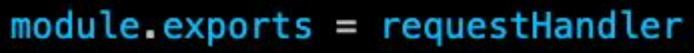
5.7 Using Modules


JS app.js

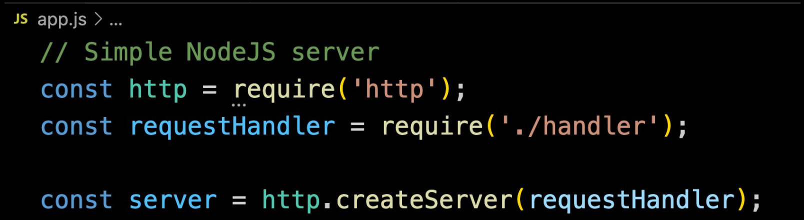
JS handler.js


JS handler.js > ...

```javascript
const fs = require("fs");

const requestHandler = (req, res) => {
  if (req.url === "/") {
    res.setHeader("Content-Type", "text/html");
```

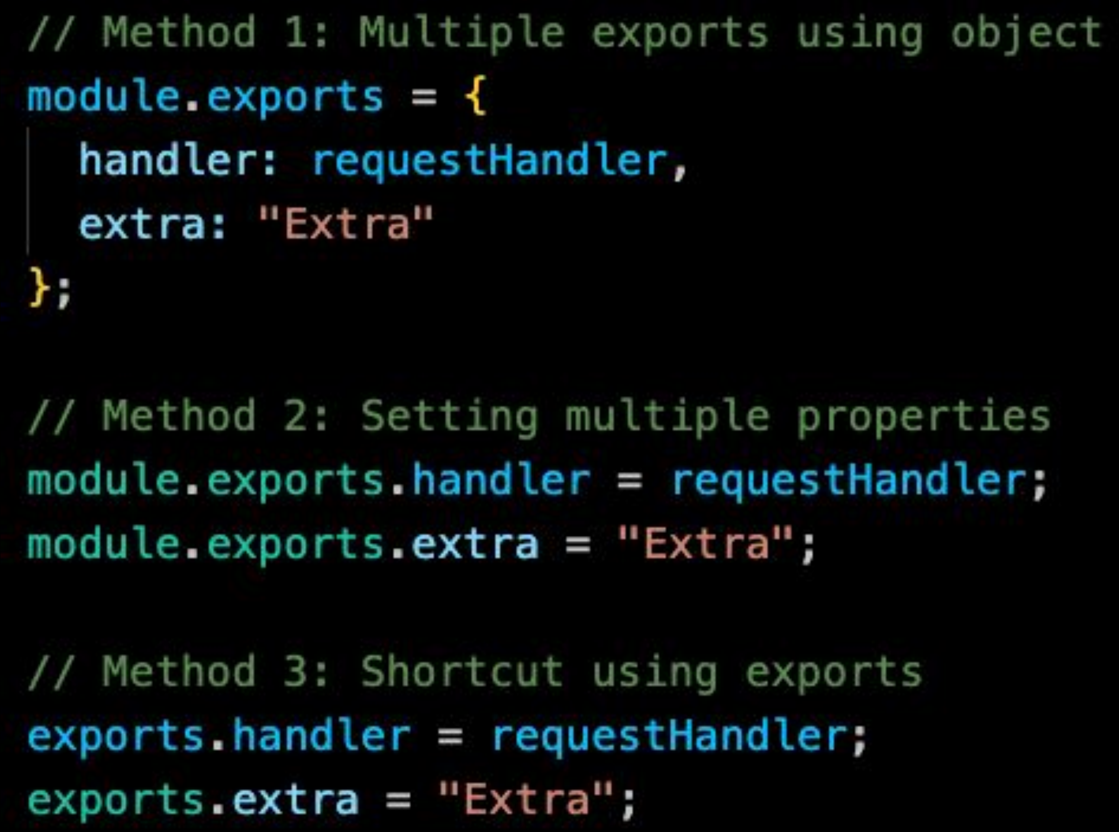


```javascript
module.exports = requestHandler
```


JS app.js > ...

```javascript
// Simple NodeJS server
const http = require('http');
const requestHandler = require('./handler');

const server = http.createServer(requestHandler);
```

5.7 Using Modules



```
// Method 1: Multiple exports using object
module.exports = {
  handler: requestHandler,
  extra: "Extra"
};

// Method 2: Setting multiple properties
module.exports.handler = requestHandler;
module.exports.extra = "Extra";

// Method 3: Shortcut using exports
exports.handler = requestHandler;
exports.extra = "Extra";
```

React

Practise Set

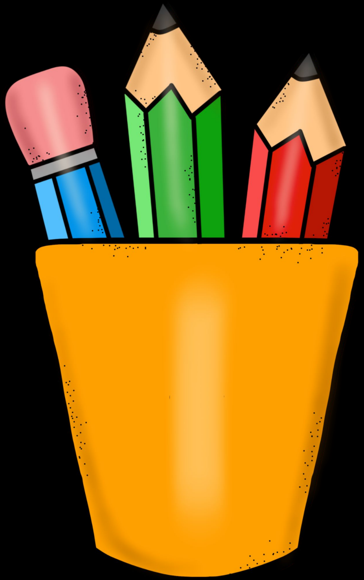
Create a Calculator

1. Create a new Node.js project named "Calculator".
2. On the home page (route "/"), show a welcome message and a link to the calculator page.
3. On the "/calculator" page, display a form with two input fields and a "Sum" button.
4. When the user clicks the "Sum" button, they should be taken to the "/calculate-result" page, which shows the sum of the two numbers.
    - Make sure the request goes to the server.
    - Create a separate module for the addition function.
    - Create another module to handle incoming requests.
    - On the "/calculate-result" page, parse the user input, use the addition module to calculate the sum, and display the result on a new HTML page.

React

Revision

1. Streams
2. Chunks
3. Buffers
4. Reading Chunk
5. Buffering Chunks
6. Parsing Request
7. Using Modules



React

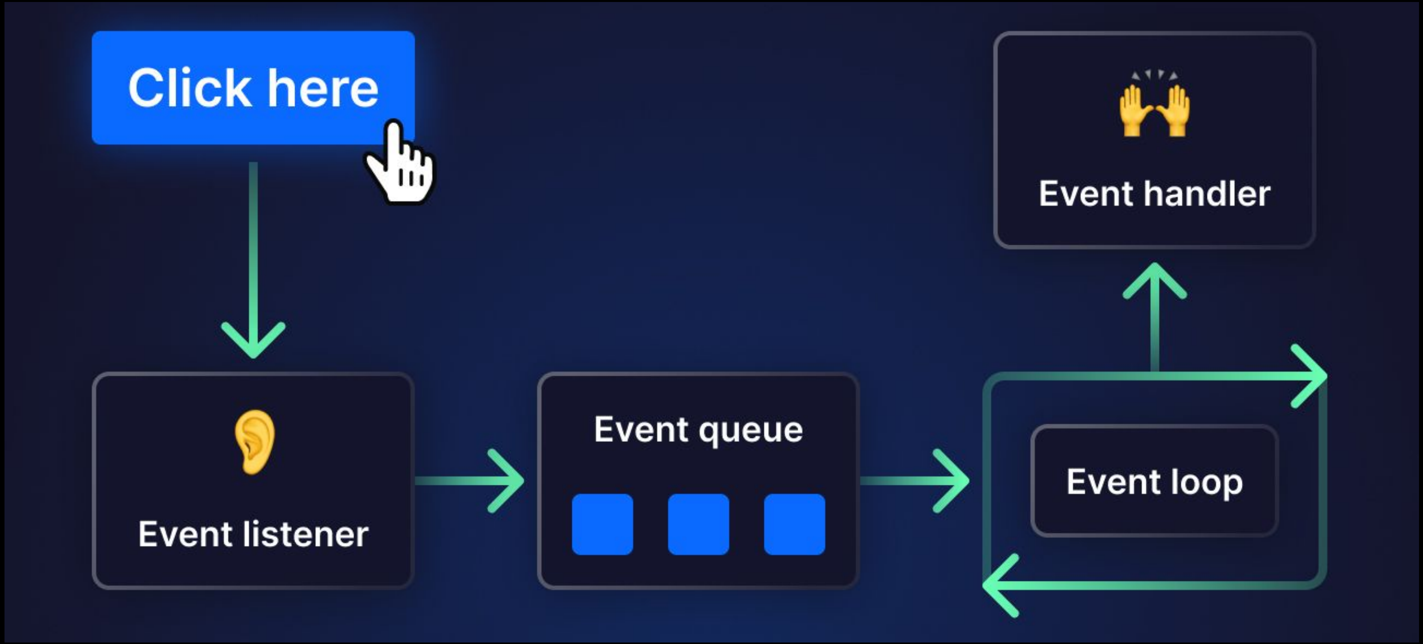
React

6. Event Loop

1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code

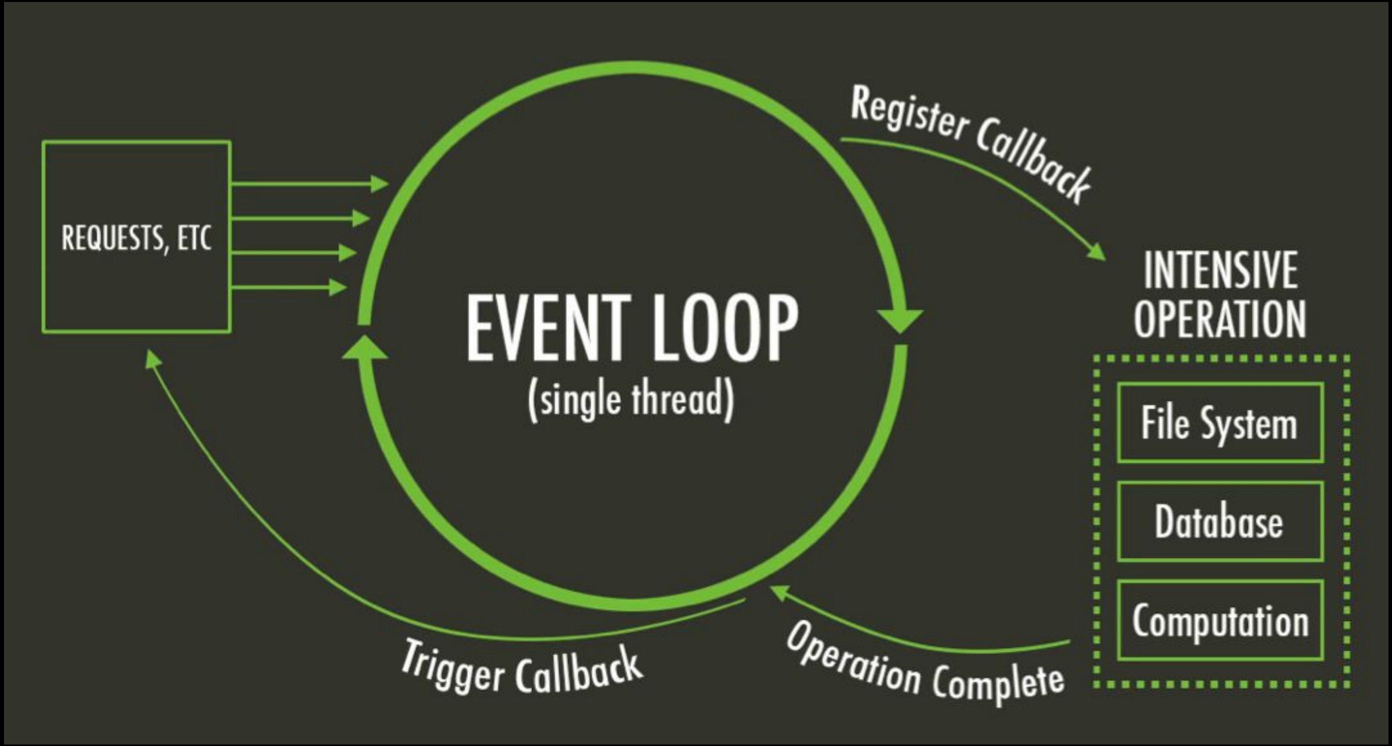
React

6.1 Event Driven



React

6.2 Single Threaded



React

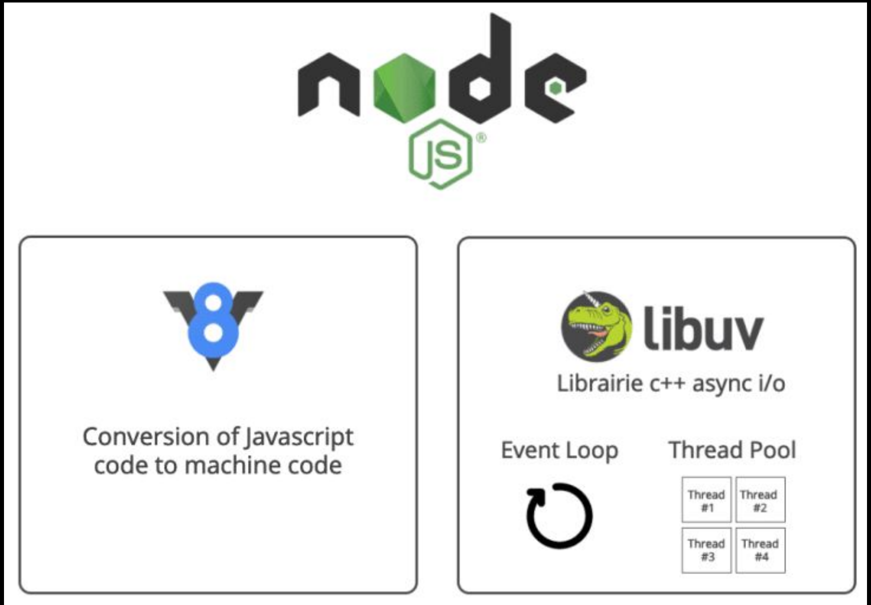
6.3 V8 vs libuv

V8:

1. Open-source JavaScript engine by Google.
2. Used in Chrome and Node.js.
3. Compiles JavaScript to native machine code.
4. Ensures high-performance JavaScript execution.

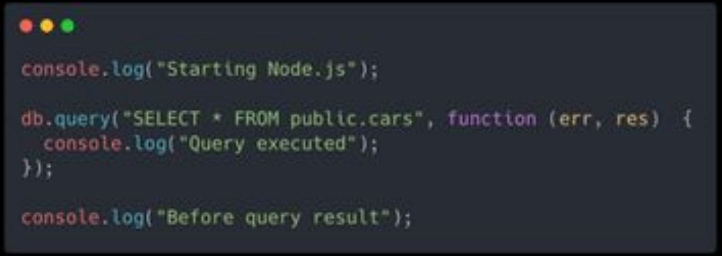
libuv:

1. Multi-platform support library for Node.js.
2. Handles asynchronous I/O operations.
3. Provides event-driven architecture.
4. Manages file system, networking, and timers non-blocking across platforms.

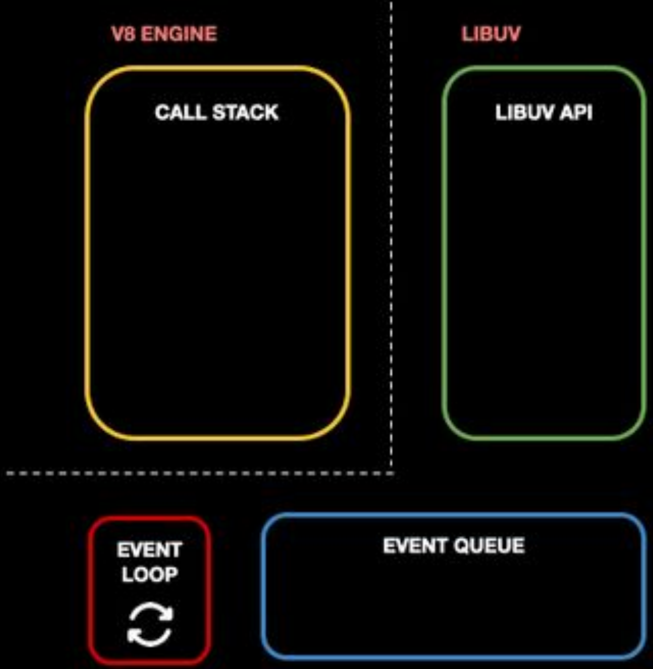


6.4 Node Runtime

An invoked function is added to the call stack. Once it returns a value, it is popped off.



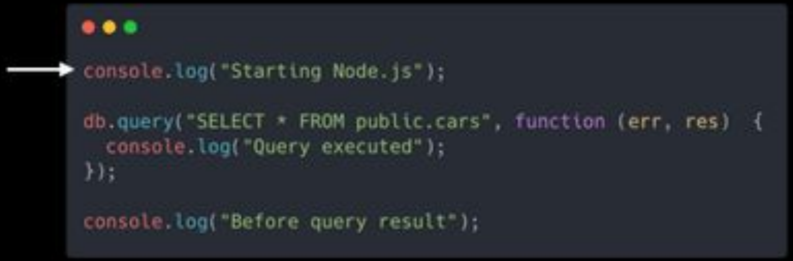
```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```



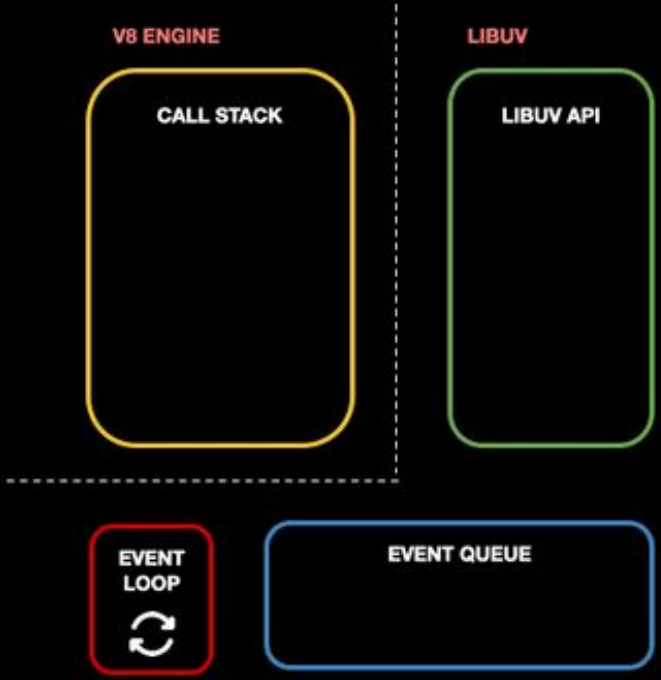
OUTPUT

6.4 Node Runtime

Database queries or other I/O ops do not block Node.js single thread because Libuv API handles them.



```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```

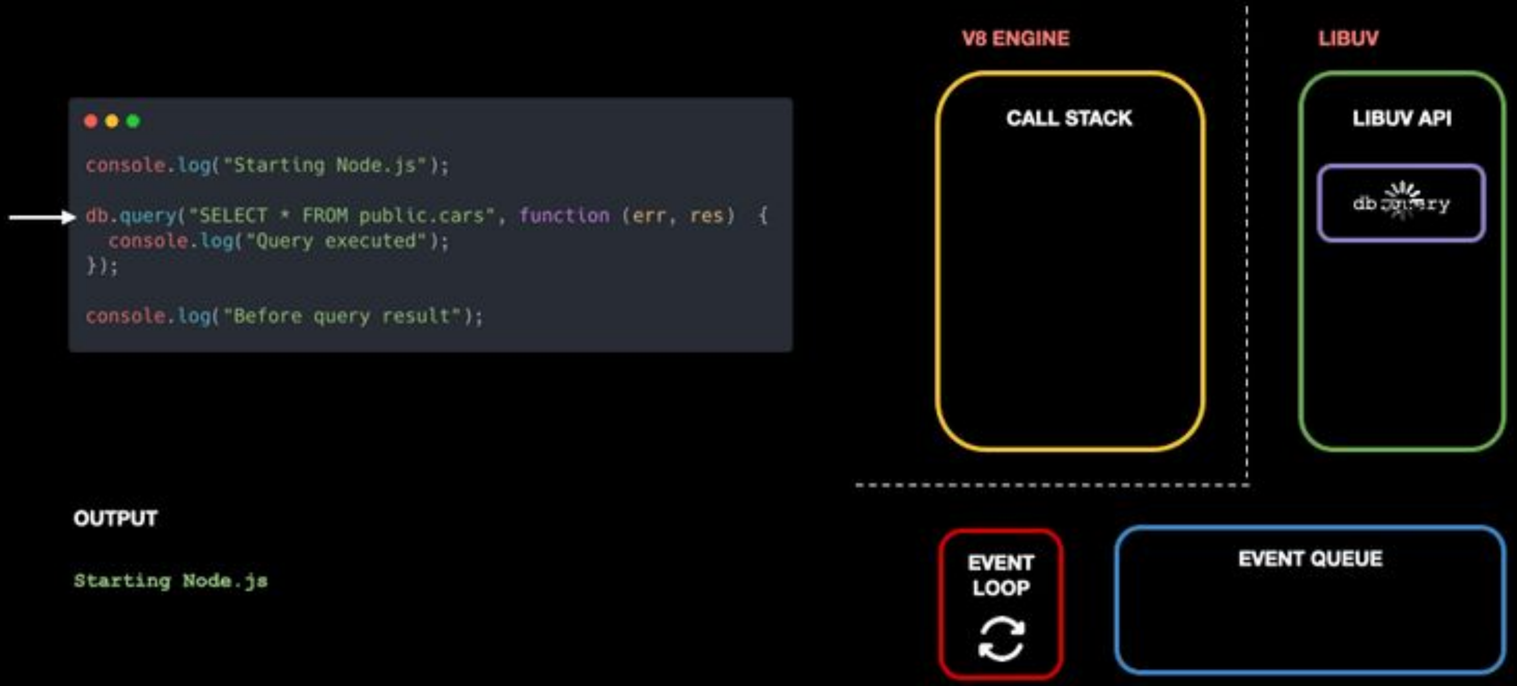


OUTPUT

```
Starting Node.js
```

6.4 Node Runtime

While Libuv asynchronously handles I/O operations, Node.js single thread keeps running code.



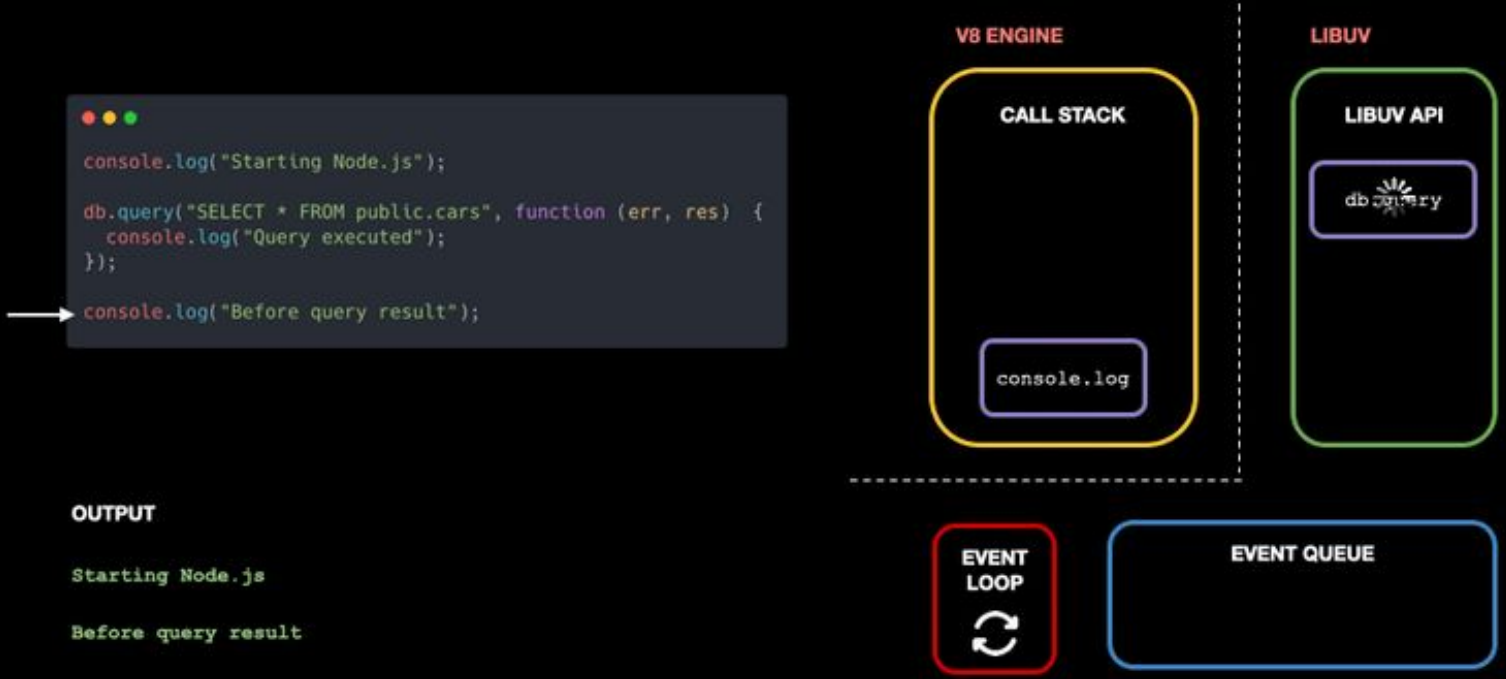
```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```

OUTPUT

```
Starting Node.js
```

6.4 Node Runtime

Callbacks of completed queries are moved to the event queue. If the call stack is empty, the event loop checks for callbacks and transfers the first.



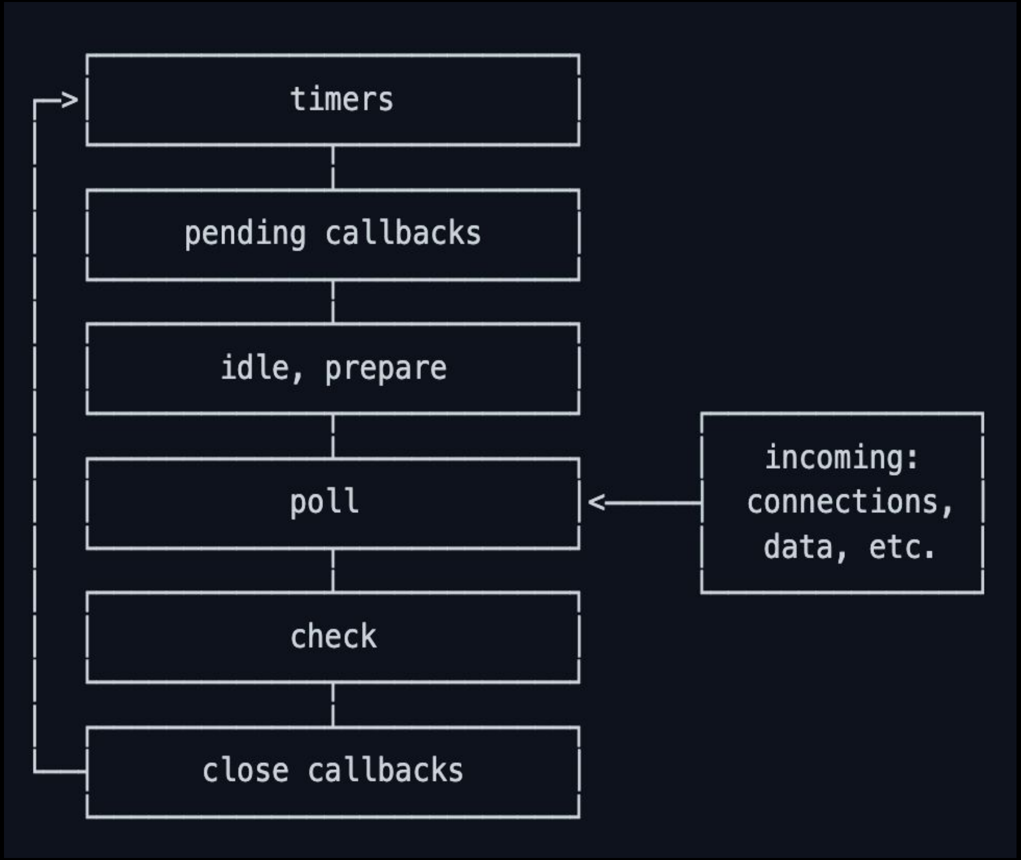
```
console.log("Starting Node.js");

db.query("SELECT * FROM public.cars", function (err, res)  {
  console.log("Query executed");
});

console.log("Before query result");
```

OUTPUT

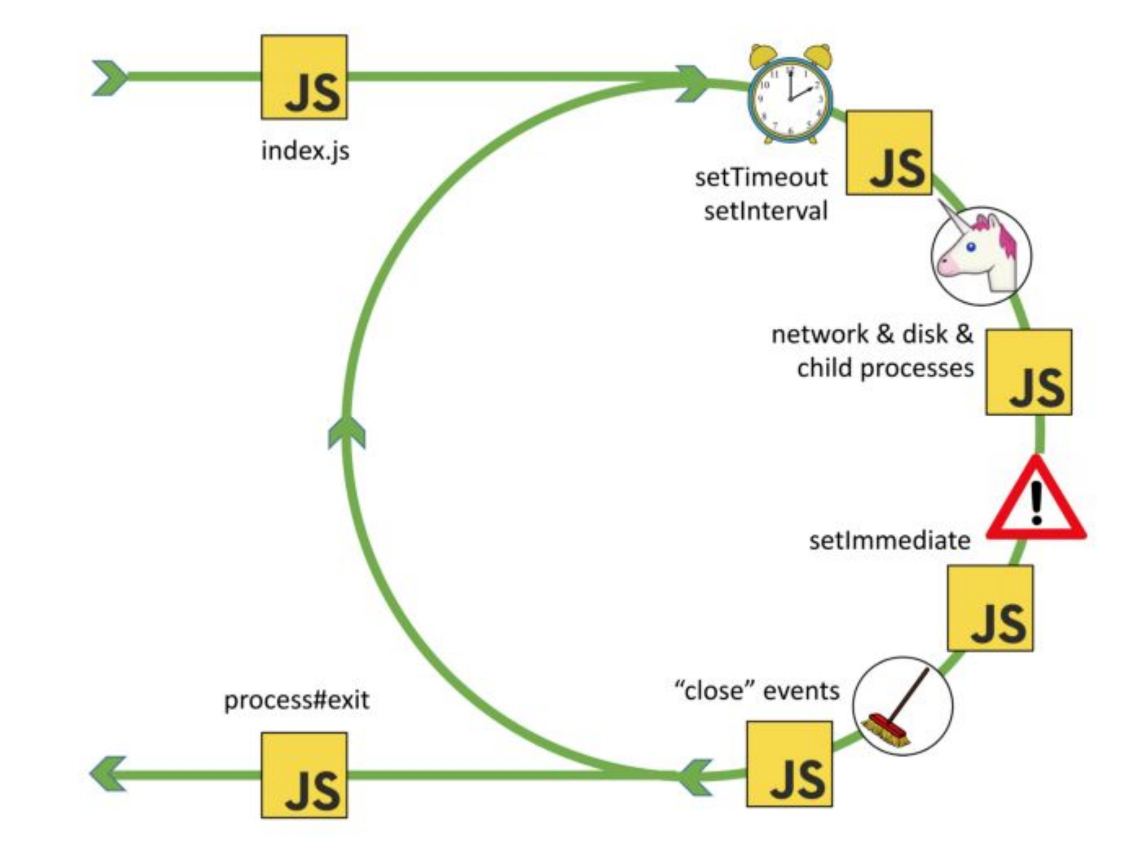
```
Starting Node.js

Before query result
```

6.5 Event Loop

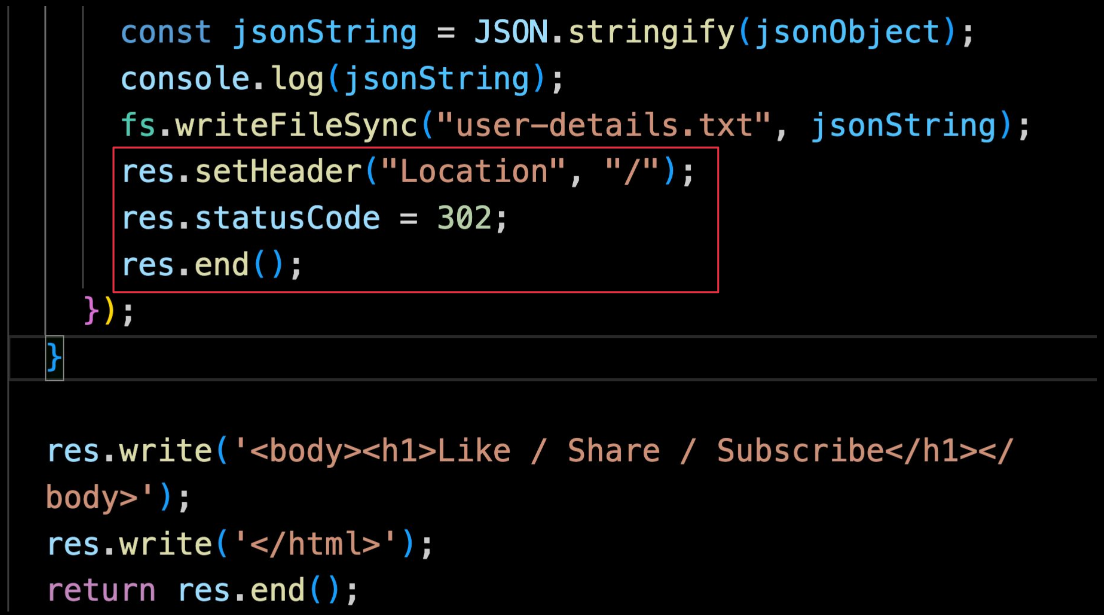
- timers: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- pending callbacks: executes I/O callbacks deferred to the next loop iteration.
- idle, prepare: only used internally.
- poll: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and `setImmediate()`); node will block here when appropriate.
- check: `setImmediate()` callbacks are invoked here.
- close callbacks: some close callbacks, e.g. `socket.on('close', ...)`.



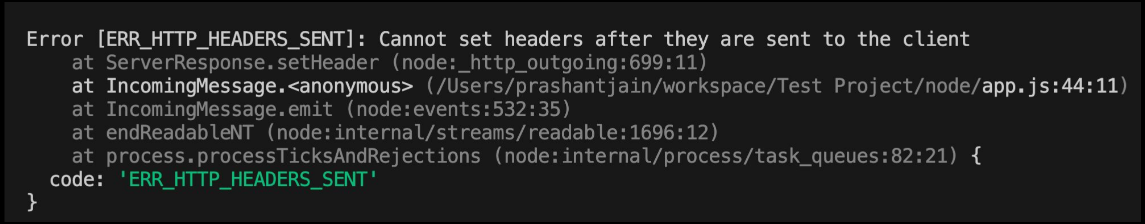
6.5 Event Loop



6.6 Async Code

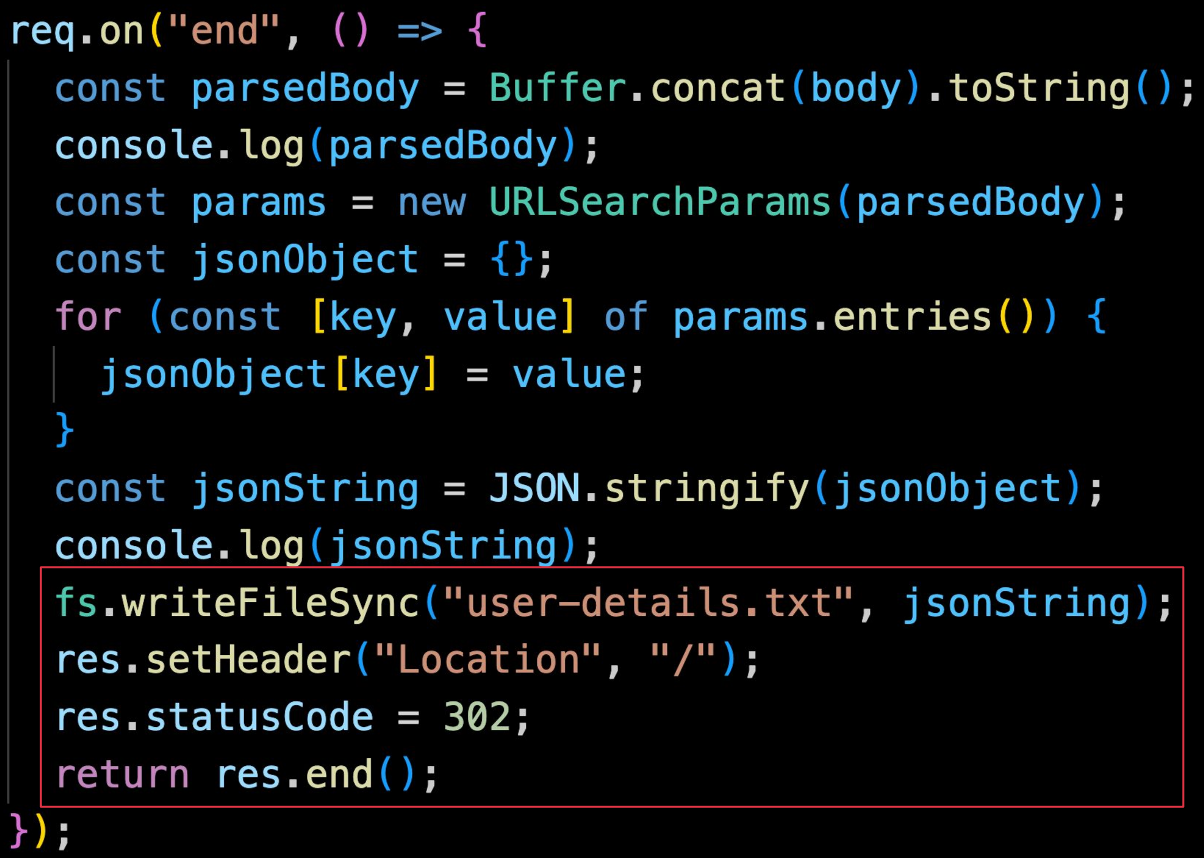


```
    const jsonString = JSON.stringify(jsonObject);
    console.log(jsonString);
    fs.writeFileSync("user-details.txt", jsonString);
    res.setHeader("Location", "/");
    res.statusCode = 302;
    res.end();
  });
}

res.write('<body><h1>Like / Share / Subscribe</h1></
body>');
res.write('</html>');
return res.end();
```



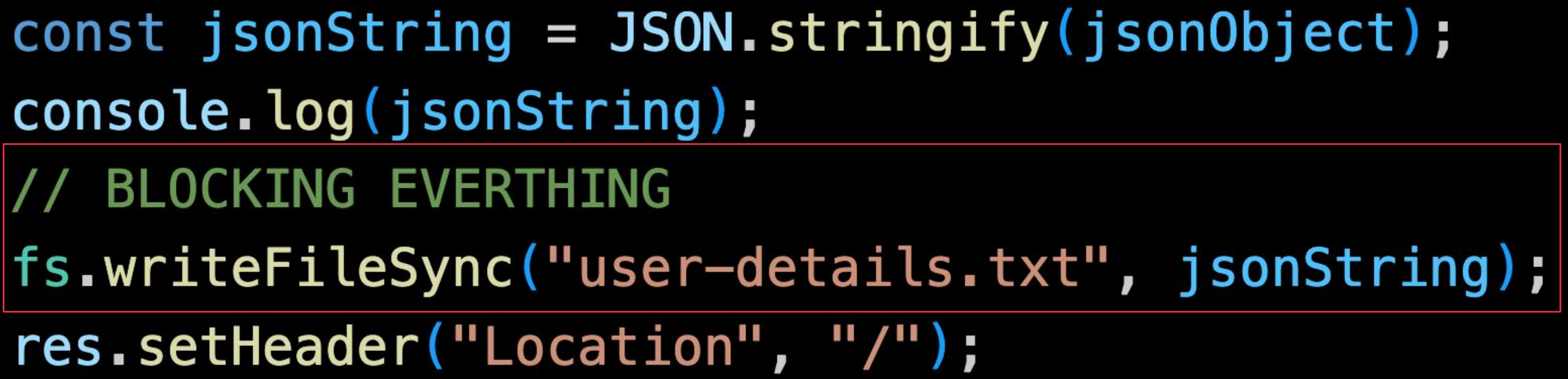
```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (node:_http_outgoing:699:11)
    at IncomingMessage.<anonymous> (/Users/prashantjain/workspace/Test Project/node/app.js:44:11)
    at IncomingMessage.emit (node:events:532:35)
    at endReadableNT (node:internal/streams/readable:1696:12)
    at process.processTicksAndRejections (node:internal/process/task_queues:82:21) {
  code: 'ERR_HTTP_HEADERS_SENT'
}
```

6.6 Async Code



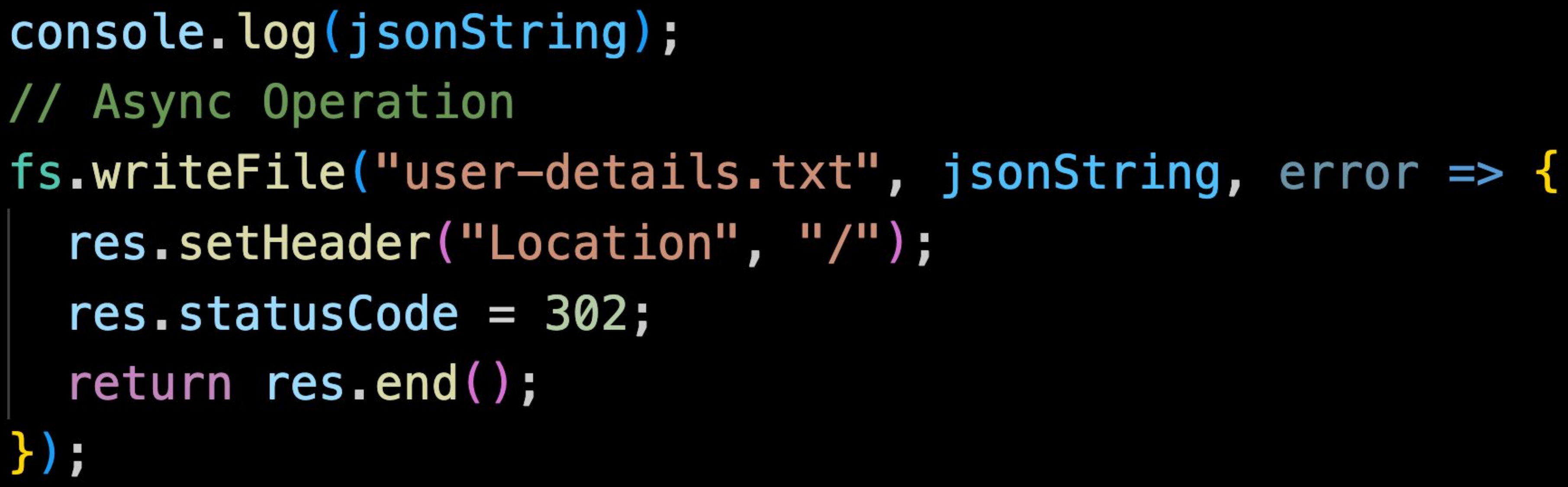
```
req.on("end", () => {
  const parsedBody = Buffer.concat(body).toString();
  console.log(parsedBody);
  const params = new URLSearchParams(parsedBody);
  const jsonObject = {};
  for (const [key, value] of params.entries()) {
    jsonObject[key] = value;
  }
  const jsonString = JSON.stringify(jsonObject);
  console.log(jsonString);
  fs.writeFileSync("user-details.txt", jsonString);
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```

6.7 Blocking Code



```
const jsonString = JSON.stringify(jsonObject);
console.log(jsonString);
// BLOCKING EVERTHING
fs.writeFileSync("user-details.txt", jsonString);
res.setHeader("Location", "/");
```

6.7 Blocking Code



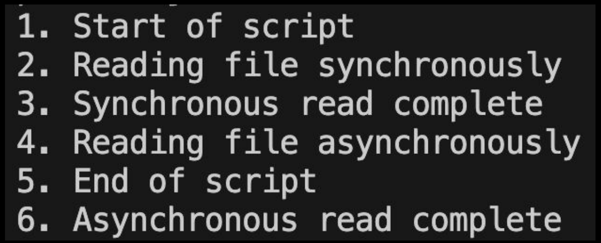
```
console.log(jsonString);
// Async Operation
fs.writeFile("user-details.txt", jsonString, error => {
  res.setHeader("Location", "/");
  res.statusCode = 302;
  return res.end();
});
```

React

Run & Observe

Blocking vs Async

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

console.log('1. Start of script');

// Synchronous (blocking) operation
console.log('2. Reading file synchronously');
const dataSync = fs.readFileSync('user-details.txt', 'utf8');
console.log('3. Synchronous read complete');

// Asynchronous (non-blocking) operation
console.log('4. Reading file asynchronously');
fs.readFile('user-details.txt', 'utf8', (err, dataAsync) => {
 if (err) throw err;
 console.log('6. Asynchronous read complete');
});

console.log('5. End of script');
```



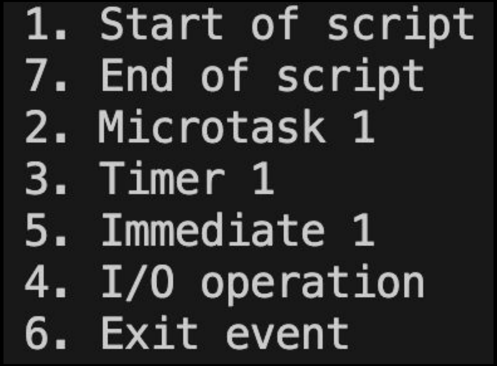
```
1. Start of script
2. Reading file synchronously
3. Synchronous read complete
4. Reading file asynchronously
5. End of script
6. Asynchronous read complete
```

React

Run & Observe

Event Loop Sequence

```
console.log('1. Start of script');

// Microtask queue (Promise)
Promise.resolve().then(() => console.log('2. Microtask 1'));

// Timer queue
setTimeout(() => console.log('3. Timer 1'), 0);

// I/O queue
const fs = require('fs');
fs.readFile('user-details.txt', () => console.log('4. I/O operation'));

// Check queue
setImmediate(() => console.log('5. Immediate 1'));

// Close queue
process.on('exit', (code) => {
 console.log('6. Exit event');
});

console.log('7. End of script');
```



```
1. Start of script
7. End of script
2. Microtask 1
3. Timer 1
5. Immediate 1
4. I/O operation
6. Exit event
```

React

Revision

1. Event Driven
2. Single Threaded
3. V8 vs libuv
4. Node Runtime
5. Event Loop
6. Async Code
7. Blocking Code