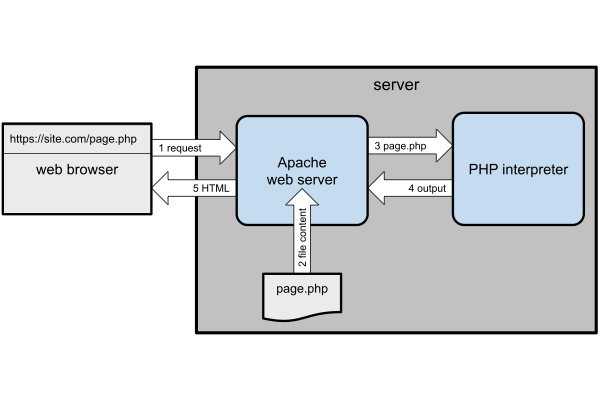
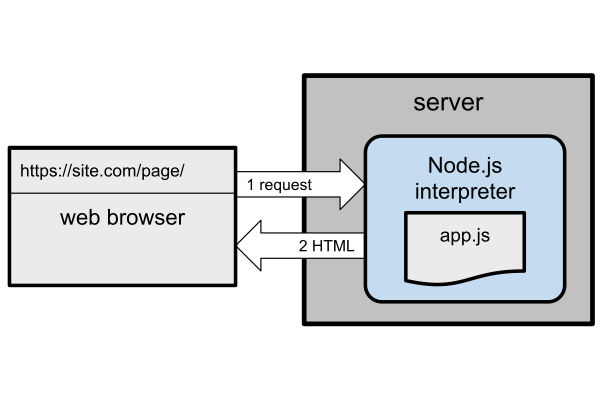
What best describes the Node.js non-blocking, event-driven I/O?

* a. Code that runs in separate processing threads.
* b. Code that runs synchronously; the next command runs after the current command has completed.
* c. Code that runs asynchronously; the next command could run before the current command has completed.
* d. Code that runs in parallel with other processes.

Node.js takes a different approach: your JavaScript application is a web server. This sounds as though it’s complex to code, but the [HTTP](https://nodejs.org/api/http.html) and [HTTPS](https://nodejs.org/api/https.html) standard libraries do much of the work for you.

Create a directory for your project, such as server:

mkdir server

cd server

Then add a file named webhello.js with the following content:

#!/usr/bin/env node

const

port = (process.argv[2] || process.env.PORT || 3000),

http = require('http');

http.createServer((req, res) => {

console.log(req.url);

res.statusCode = 200;

res.setHeader('Content-Type', 'text/html');

res.end(`<p>Hello World!</p>`);

}).listen(port);

console.log(`Server running at http://localhost:${ port }/`);

Run it with node webhello.js and you’ll see Server running at http://localhost:3000/ or similar. Open that address in your web browser to view a web page with a “Hello World!” paragraph.

The code does the following:

* It defines a variable for the server’s port. This can be passed on the command line, a PORT environment variable, or it falls back to 3000.
* It uses the [HTTP createServer](https://nodejs.org/api/http.html#http_http_createserver_options_requestlistener) library to create a web server which listens on that port. When its callback function receives a request, it can examine the details in the req object and return a response using the res object.

This is a simple example, and the server returns the same “Hello World!” response regardless of the URL. Try accessing http://localhost:3000/, http://localhost:3000/abc/, or http://localhost:3000/abc/123/: every page is the same.

Live production servers can use a web server such as NGINX to forward requests to Node.js.

**Restarting Node.js Applications with Nodemon**

You must restart a running Node.js application every time you make a change. Pressing Ctrl | Cmd + C and launching again will quickly become tiresome.

[Nodemon](https://nodemon.io/) is a utility that monitors your source files for changes and automatically restarts the application. Install it globally with npm:

npm install -g nodemon

You can now use nodemon in place of node to launch any Node.js application. For example:

nodemon webhello.js

*(You can pass any arguments as before.)*

When you save a code change, Nodemon restarts the application and you’ll see a log entry in the terminal:

[nodemon] restarting due to changes...

[nodemon] starting `node webhello.js`

If it doesn’t work, try running nodemon with the --legacy-watch / -L argument:

nodemon -L webhello.js

<https://vimeo.com/707851682/165b441f04>