Understanding the basics of Node.js

[](https://medium.com/@sindhu.a?source=post_page-----99e01c5d844f----------------------)

[Sindhu Ananthakrishnan](https://medium.com/@sindhu.a?source=post_page-----99e01c5d844f----------------------)

Follow

[Jan 2, 2018](https://medium.com/prod-io/understanding-the-basics-of-node-js-99e01c5d844f?source=post_page-----99e01c5d844f----------------------) · 6 min read

**What is Node.js?**

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. It enables JavaScript to be used for server-side scripting, and runs scripts server-side to produce dynamic web page content before the page is sent to the user’s web browser.

Node.js is a platform built on Chrome’s Javascript run-time for building fast and scalable network applications.

Node.js is **NOT** a framework. There are web-based frameworks, such as Express, which are based on Node.js.

**Why Node.js?**

* Very fast
* Asynchronous and event driven
* Single Threaded but highly scalable
* No Buffering

**Why Node.js is fast?**

The speed of Node.js is mainly down to two things: the V8 engine and its non-blocking feature.

**The V8 engine:**

Node.js uses Google Chrome’s ultra fast V8 execution engine. Until the release of Chrome, most browsers read JavaScript inefficiently: the code was read and interpreted bit by bit. It took a lot of time to read JavaScript and to convert it to machine language so it could be understood by the processor.

Google Chrome’s V8 engine works completely different. It is highly optimized and carries out what we call JIT (Just In Time) compilation. It quickly transforms JavaScript code into machine language.We don’t need to know how V8 works to be able to use Node.js. Just remember that it makes the execution of JavaScript ultra fast.

**Non-blocking model**

As JavaScript is a language built around the idea of events, Node.js allows an entirely non-blocking code algorithm to be put into place.By the way, do we know the difference between a blocking code and a non-blocking code? Hmmm, perhaps a little explanation is called for!

**Blocking model vs. non-blocking model**

Imagine a program which is created to upload a file and then display it. This is how the code is written in sequential programming languages(**blocking model)**:

Upload the file  
Display the file  
Do something else

The actions are carried out in order. The lines need to be read from top to bottom:

1. The program will upload a file to the Internet.
2. The program will display the file to the user.
3. Then, the program can do other things (i.e. carry out other actions).

Now, we can write the same code in a **non-blocking model**:

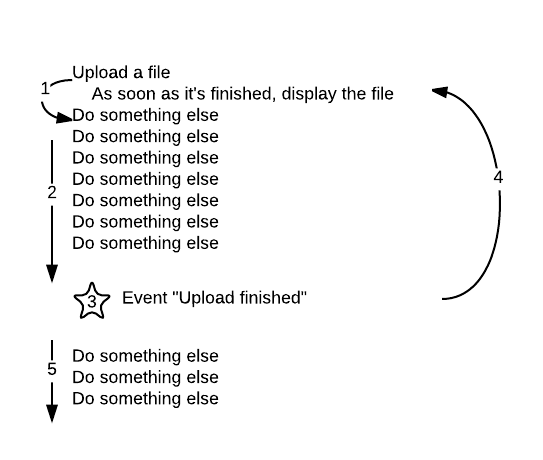
Upload a file  
As soon as it’s finished, display the file

Do something else

The program no longer carries out the tasks in the order that they are written. It does this:

1. The program launches the upload of the file to the internet.
2. The program does other things (the program follows its course).
3. As soon as the upload is finished, the program carries out the actions that we asked it to do: it displays the file.

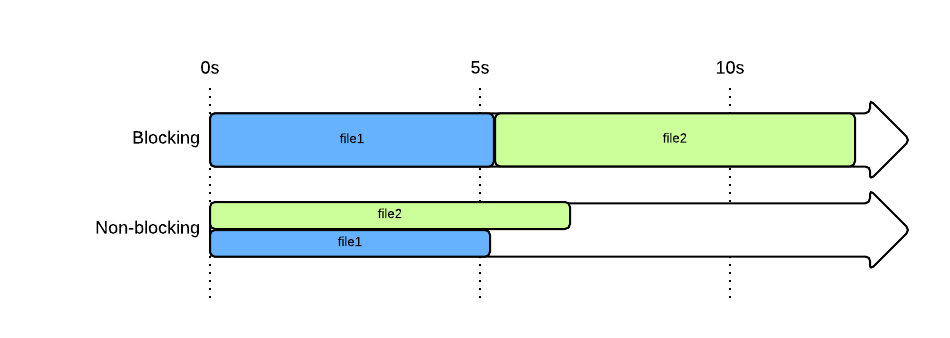
https://miro.medium.com/max/60/0*V9pUgGvmKvBFWv4b.?q=20



The non-blocking model in programming

This is exactly how Node.js works. As soon as “File uploaded” appears, a function known as a *callback function* is called and carries out actions (here, the callback function displays the file).

https://miro.medium.com/max/60/0*yTEYwofKPwKmS0WX.?q=20



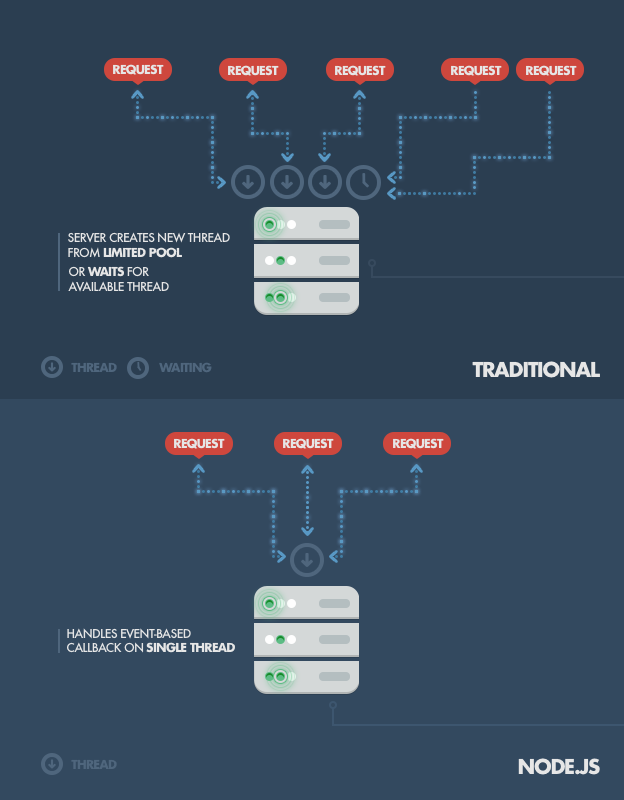
In a non-blocking model (like Node.js), the 2 files are uploaded at the same time and the whole thing finishes quicker

**Why is it scalable?**

Node.js capable of handling a huge number of simultaneous connections with high throughput, which equates to high scalability.

How it works under-the-hood is pretty interesting. Compared to traditional web-serving techniques where each connection (request) spawns a new thread, taking up system RAM and eventually maxing-out at the amount of RAM available, Node.js operates on a single-thread, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections ([held in the event loop](https://en.wikipedia.org/wiki/Nodejs)).

https://miro.medium.com/max/46/0*kJ6vNeUGrJ0x115j.?q=20



A quick calculation: assuming that each thread potentially has an accompanying 2 MB of memory with it, running on a system with 8 GB of RAM puts us at a theoretical maximum of 4,000 concurrent connections *(calculations taken from Michael Abernethy’s article “Just what is Node.js?”, published on IBM developerWorks in 2011; unfortunately, the article is not available anymore)*, plus the cost of [context-switching between threads](http://www.slideshare.net/marcusf/nonblocking-io-event-loops-and-nodejs). That’s the scenario you typically deal with in traditional web-serving techniques. By avoiding all that, Node.js achieves scalability levels of over [1M concurrent connections](http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/), and over [600k concurrent websockets connections](https://blog.jayway.com/2015/04/13/600k-concurrent-websocket-connections-on-aws-using-node-js/).

**Where to use Node.js?**

Data-intensive applications  
Application that involves fewer computations

**Why is node.js not suitable for applications that involve more computations?**

**Analogy:**

Think of web application as a restaurant. You have waiters (web server) and cooks (workers). Waiters are in contact with clients and do simple tasks like providing menu or explaining if some dish is vegetarian. On the other hand, they delegate harder tasks to the kitchen. Because waiters are doing only simple things they respond quickly, and cooks can concentrate on their job.

Node.js here would be a single but very talented waiter that can process many requests at a time, and Apache would be a gang of dumb waiters that just process one request each. If this one Node.js waiter would begin to cook, it would be an immediate catastrophe. Still, cooking could also exhaust even a large supply of Apache waiters, not mentioning the chaos in the kitchen and the progressive decrease of responsitivity.

Hence heavy load tasks should be delegated to standalone programs.

**Why should you choose node.js?**Nodejs is basically server-side JavaScript. Working with Nodejs is much less of a context switch for a developer who is also working with JS in the frontend. It’s much easier to get started and even gain expertise as a full stack developer if you have a Nodejs server. Since, its JavaScript in the backend as well, there is a lot of consistency in terms of the format of the data — JSON! More common things in the front and back end too. If there is any common functionality that is needed in both front and back end, you can write it in a common file and just “require” it.If you use different language in the front end and back end there will be places where you will have to repeat the functionalities for front and back end.

Another point in favour of Node.js is its deep friendship with MongoDB. Again, the protocol for communication here is JSON, so there’s very little overhead in processing the data.

Did I mention it’s JavaScript? Have you missed underscore library while writing server-side code? Well, you get all the goodness that JavaScript universe has to offer in the backend as well now. This can help you build a robust and multi-functional server in a very short time.

To gain expertise in Node.js and start building applications, following online tutorials can be followed.

**Online tutorials**

**Express:**

<https://codeforgeek.com/2014/06/express-nodejs-tutorial/>

**Mongo DB:**

<https://codeforgeek.com/2015/08/restful-api-node-mongodb/>

**CRUD in node.js**

<https://medium.freecodecamp.org/building-a-simple-node-js-api-in-under-30-minutes-a07ea9e390d2>

**Sources:**

**[Node.js: what is it for exactly?](https://openclassrooms.com/courses/ultra-fast-applications-using-node-js/node-js-what-is-it-for-exactly?source=post_page-----99e01c5d844f----------------------)**

[Let's start by answering all the questions that are running through your mind: Why do web developers seem to like Node…](https://openclassrooms.com/courses/ultra-fast-applications-using-node-js/node-js-what-is-it-for-exactly?source=post_page-----99e01c5d844f----------------------)

[openclassrooms.com](https://openclassrooms.com/courses/ultra-fast-applications-using-node-js/node-js-what-is-it-for-exactly?source=post_page-----99e01c5d844f----------------------)