Basics of Node.js

Renuka Kirve
Mindstix Labs

What is Node.js?

- Node.js is a server-side JavaScript environment that uses an asynchronous event-driven and non-blocking I/O model.
- Node.js allows JavaScript to be executed on the server side, and it uses the fast V8 JavaScript engine which was developed by Google for the Chrome browser.
- It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications.
- Node.js is different from client-side JavaScript in that it removes certain things, like DOM manipulation, and adds support for Evented I/O, processes, streams, HTTP, SSL, DNS, string and buffer processing.

Why Node.js?

- Reading and writing to network connections, file System, and to databases very very fast in node.
- Node allows you to build fast, scalable network applications capable of handling a huge number of simultaneous connections with high throughput.
- Using Node.js allows you to use the same language on the client, on the server, and in the database. You can keep your data in its native JSON format from browser to disk.

Why Node.js?

- **Unique points about Node.js:**
  - JavaScript is used in client-side, but Node.js puts the JavaScript on server-side. Thus making communication between client and server in same language.
  - Servers are normally thread based but Node.JS is “Event” based.
  - Node.JS serves each request in a Evented loop that can handle simultaneous requests.

Key Points - Node.js

- **Asynchronous:**
  - All APIs of Node.js library are asynchronous that is non-blocking. It means a Node.js based server never waits for a API to return data. Server moves to next API after calling it .
- **Single Threaded:**
  - Node.js uses a single threaded model with event looping. Event mechanism helps server to respond in a non-blocking ways and makes server highly scalable. Node.js uses a single threaded program and same program can serve much larger number of requests.
- **No Buffering:**
  - Node.js applications never buffer any data. These applications simply output the data in chunks.
- **Built-in support for the most important protocols** (HTTP, DNS, TLS).

Key Points - Node.js

- **Event Driven:**
  - In a normal process cycle, the web server, while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.
  - Node.JS process each request as events, server doesn't wait for the IO operation to complete while it can handle other request at the same time.
  - When the IO operation of first request is completed it will call-back the server to complete the request.

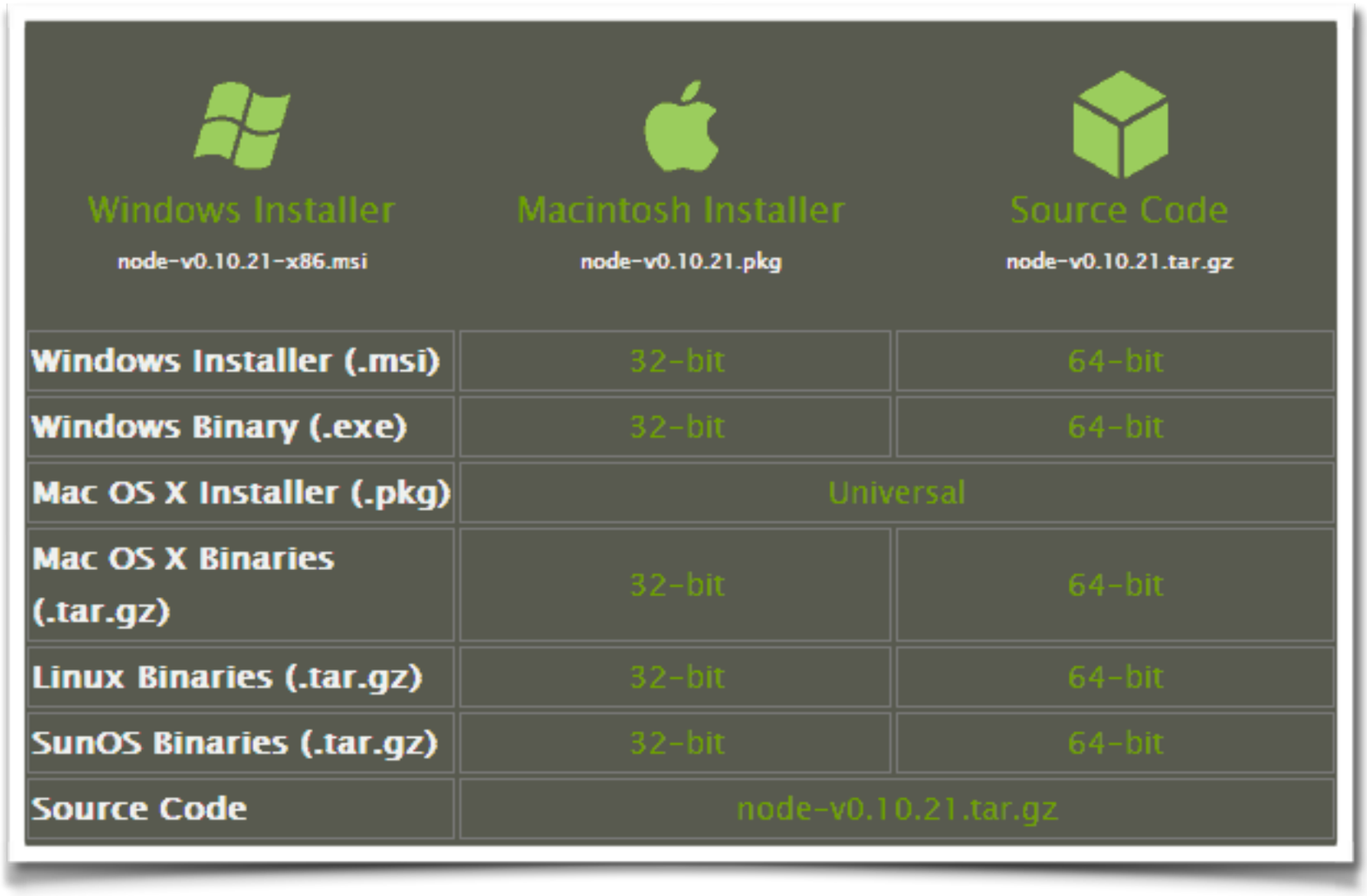
Install Node.js

- Node.js binaries are available for Linux, Windows, MAC operating systems.

- Node.js installers can be downloaded from https://nodejs.org/en/download/

Install Node.js on Windows

- Download Node.js installer (.msi) for Windows platform from https://nodejs.org/en/download/ and follow the instructions to setup Node.js on Windows platform.



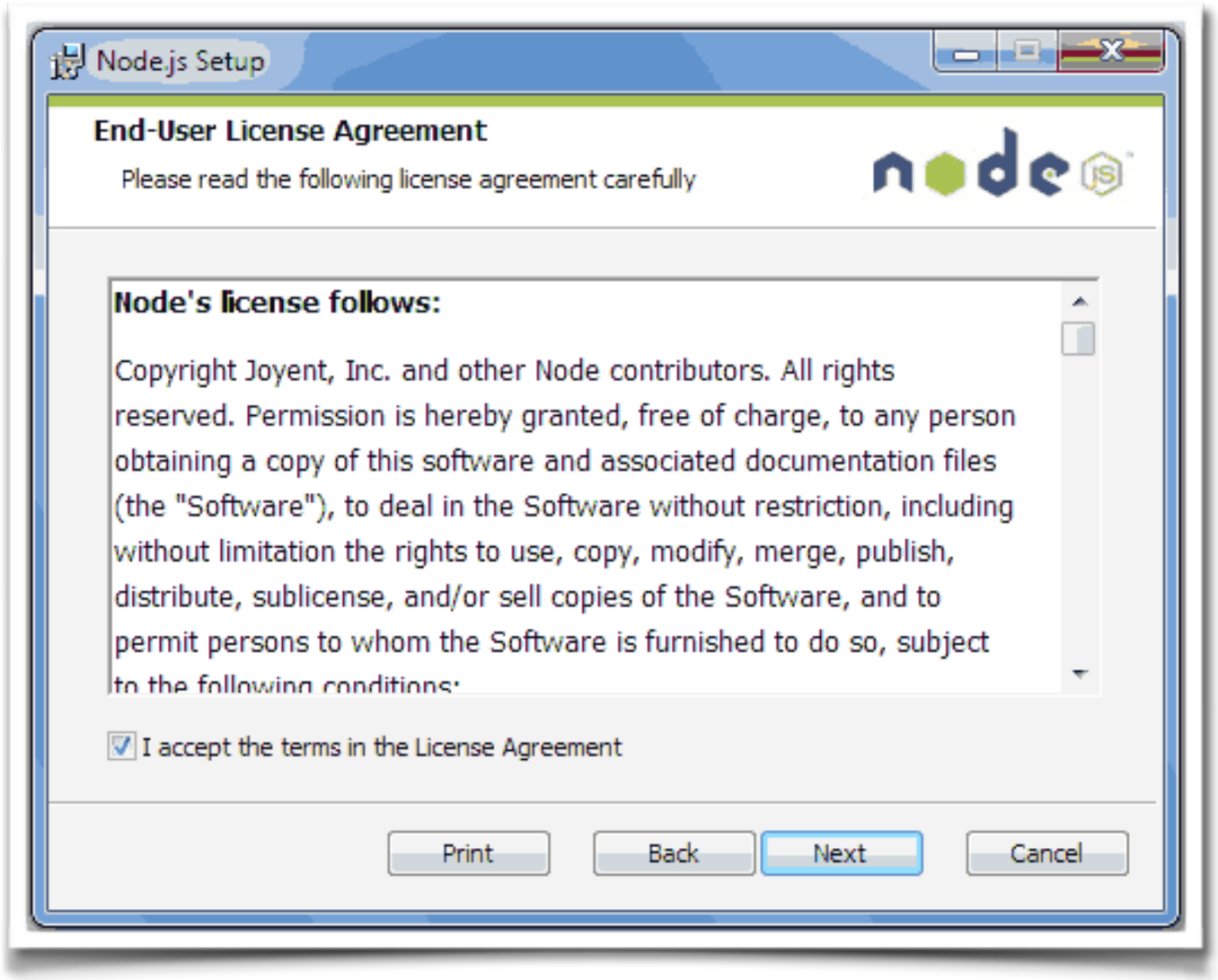
Windows Installer node-v0.10.21-x86.msi	Macintosh Installer node-v0.10.21.pkg	Source Code node-v0.10.21.tar.gz
Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Installer (.pkg)	Universal	
Mac OS X Binaries (.tar.gz)	32-bit	64-bit
Linux Binaries (.tar.gz)	32-bit	64-bit
SunOS Binaries (.tar.gz)	32-bit	64-bit
Source Code	node-v0.10.21.tar.gz	

Install Node.js on Windows

- After downloading, double-click on the (.msi) file to start the Node.js Setup Wizard. Click Next.

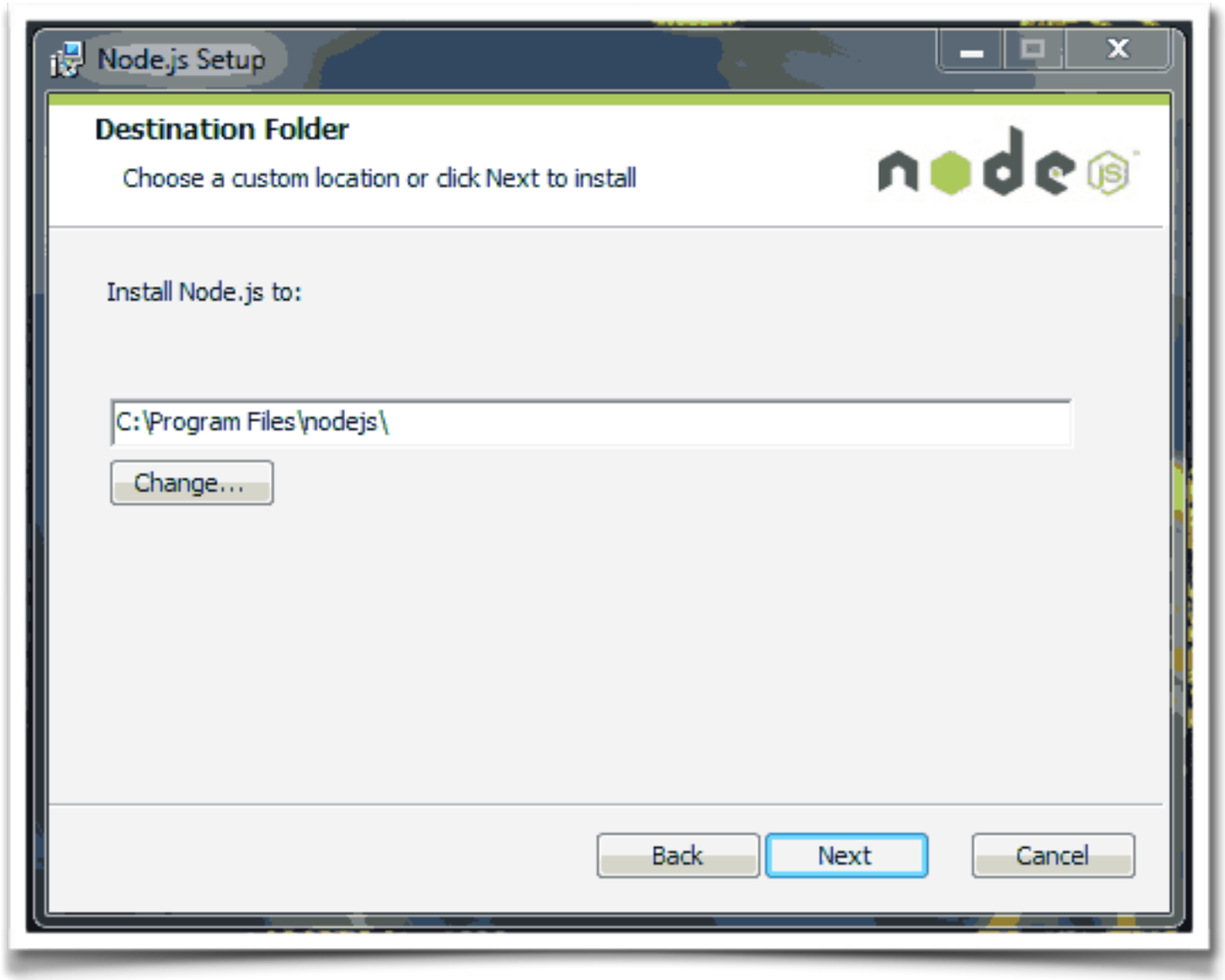
Install Node.js on Windows

- If you accept the End-User License Agreement, check “I accept the terms in the License Agreement”.Click Next.



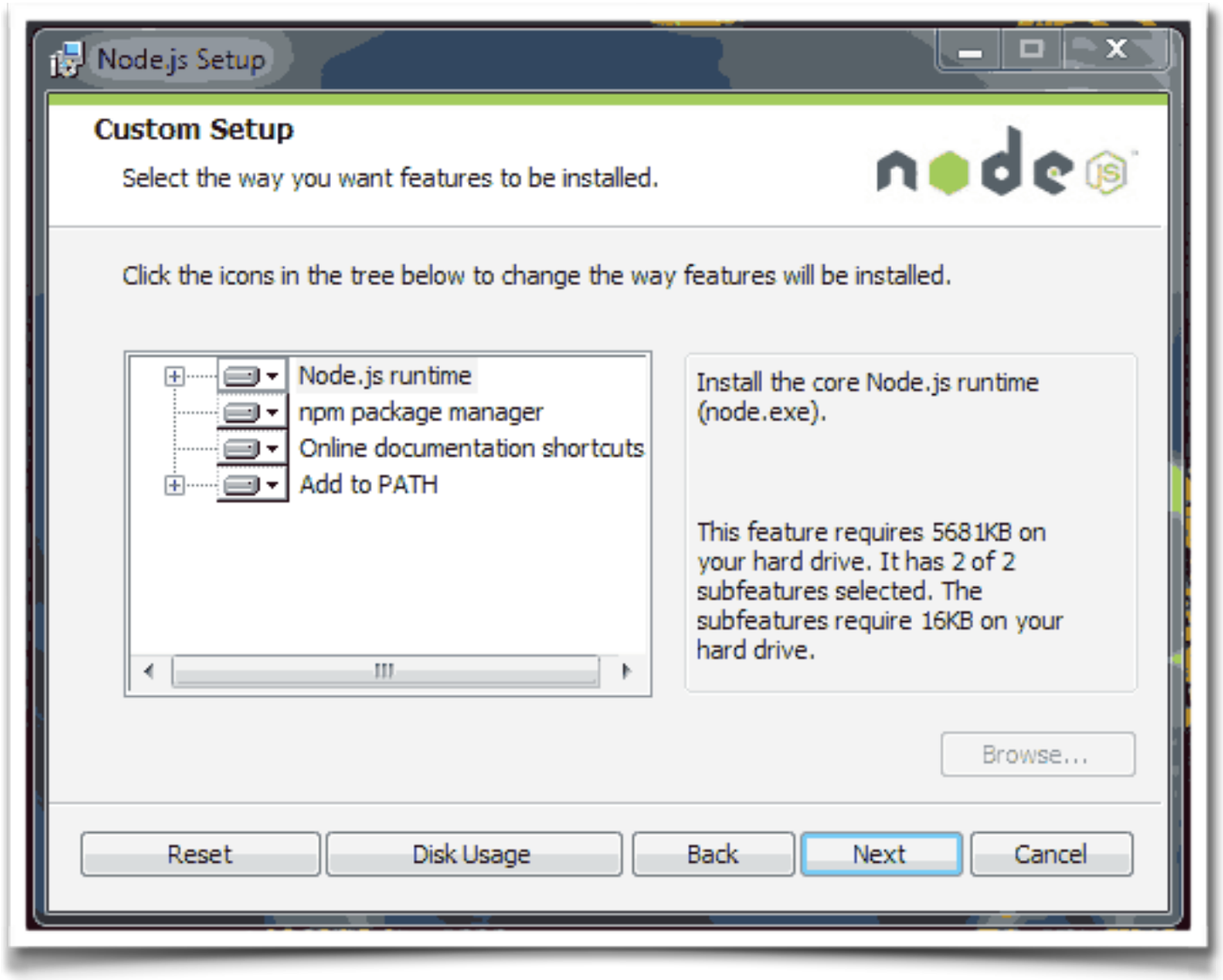
Install Node.js on Windows

- By default Node.js gets installed at **'C:\Program Files\nodejs',** you can modify the installation directory. Click Next.



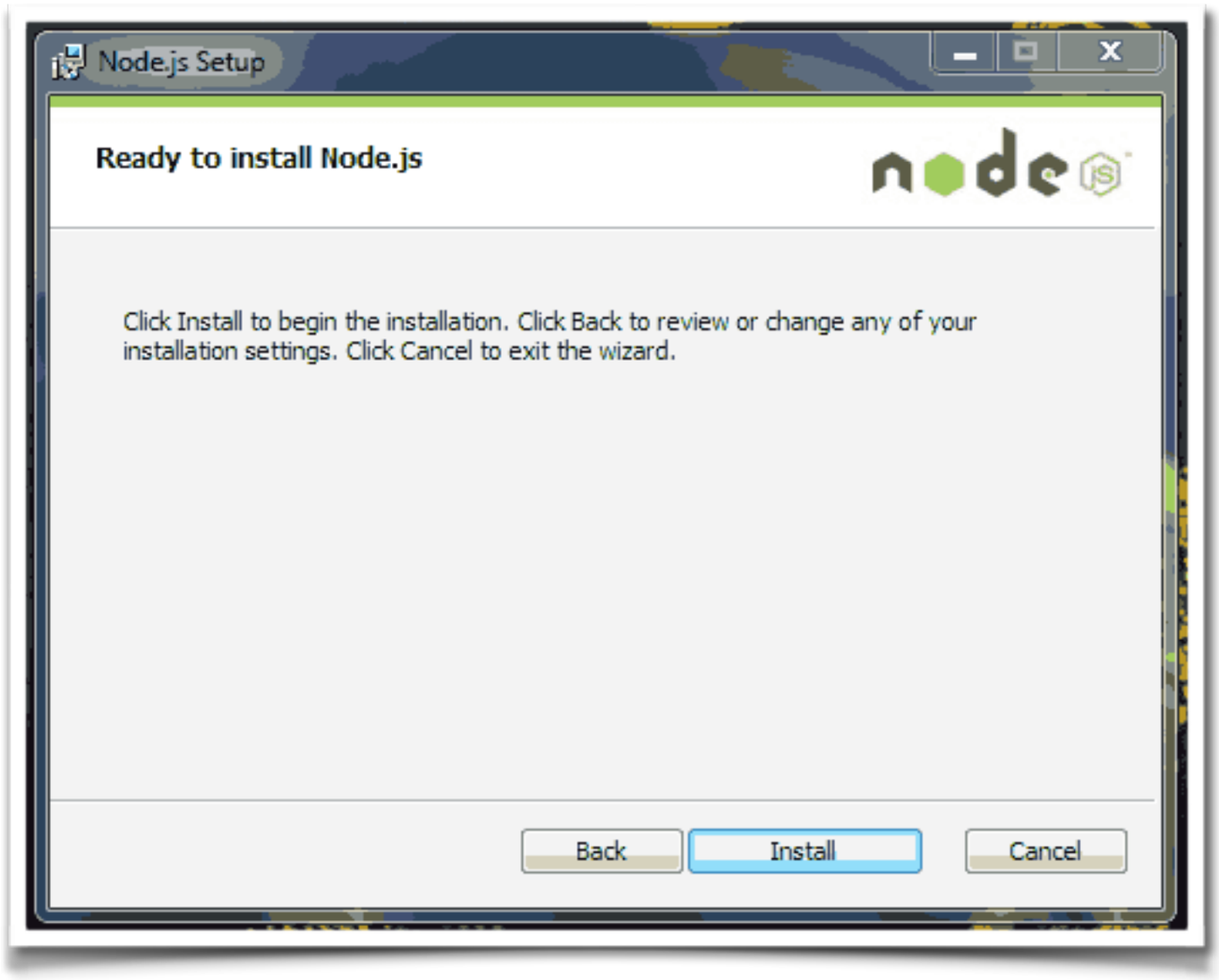
Install Node.js on Windows

- Select the features you want to install. Click Next.



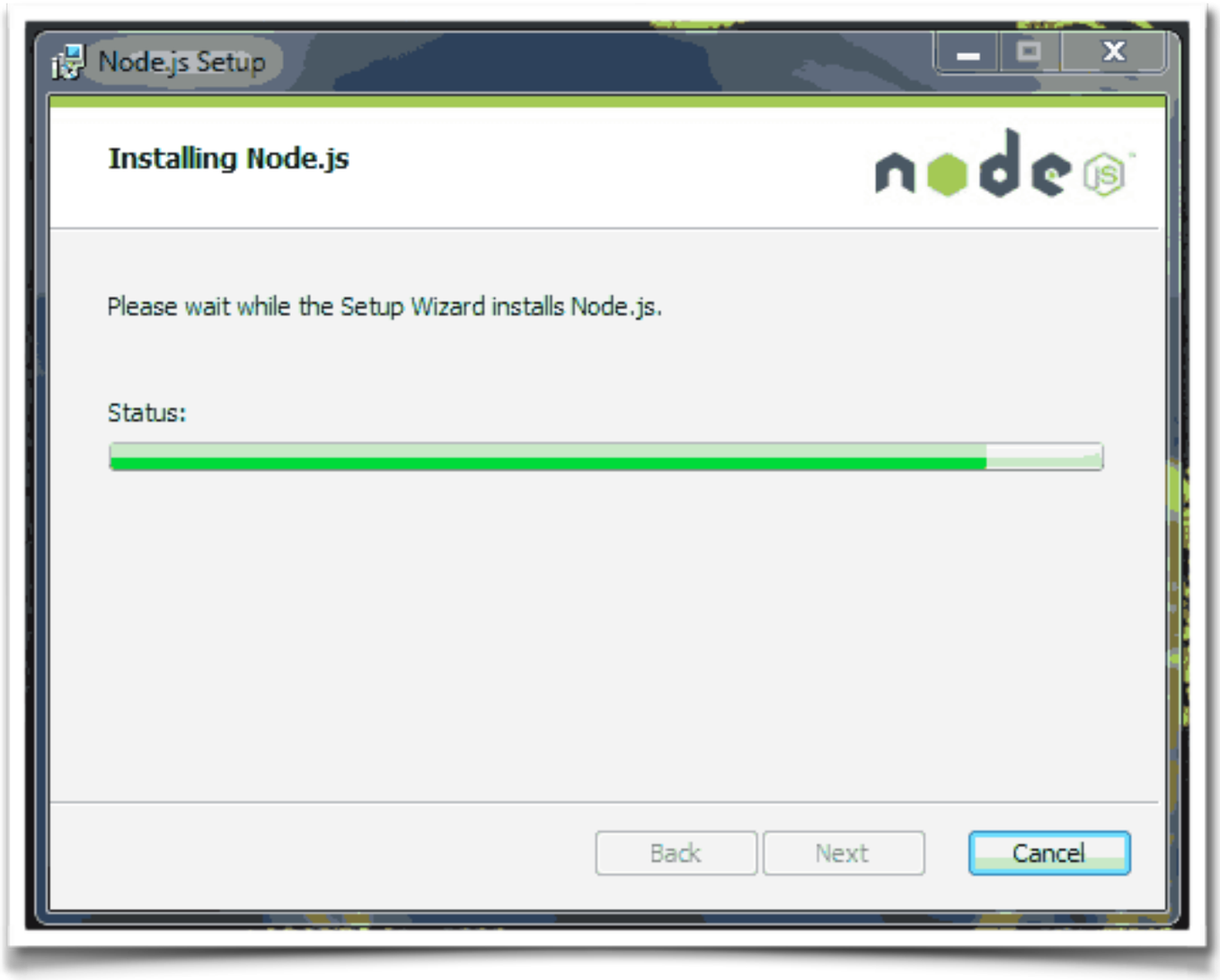
Install Node.js on Windows

- Click Install to begin the installation.



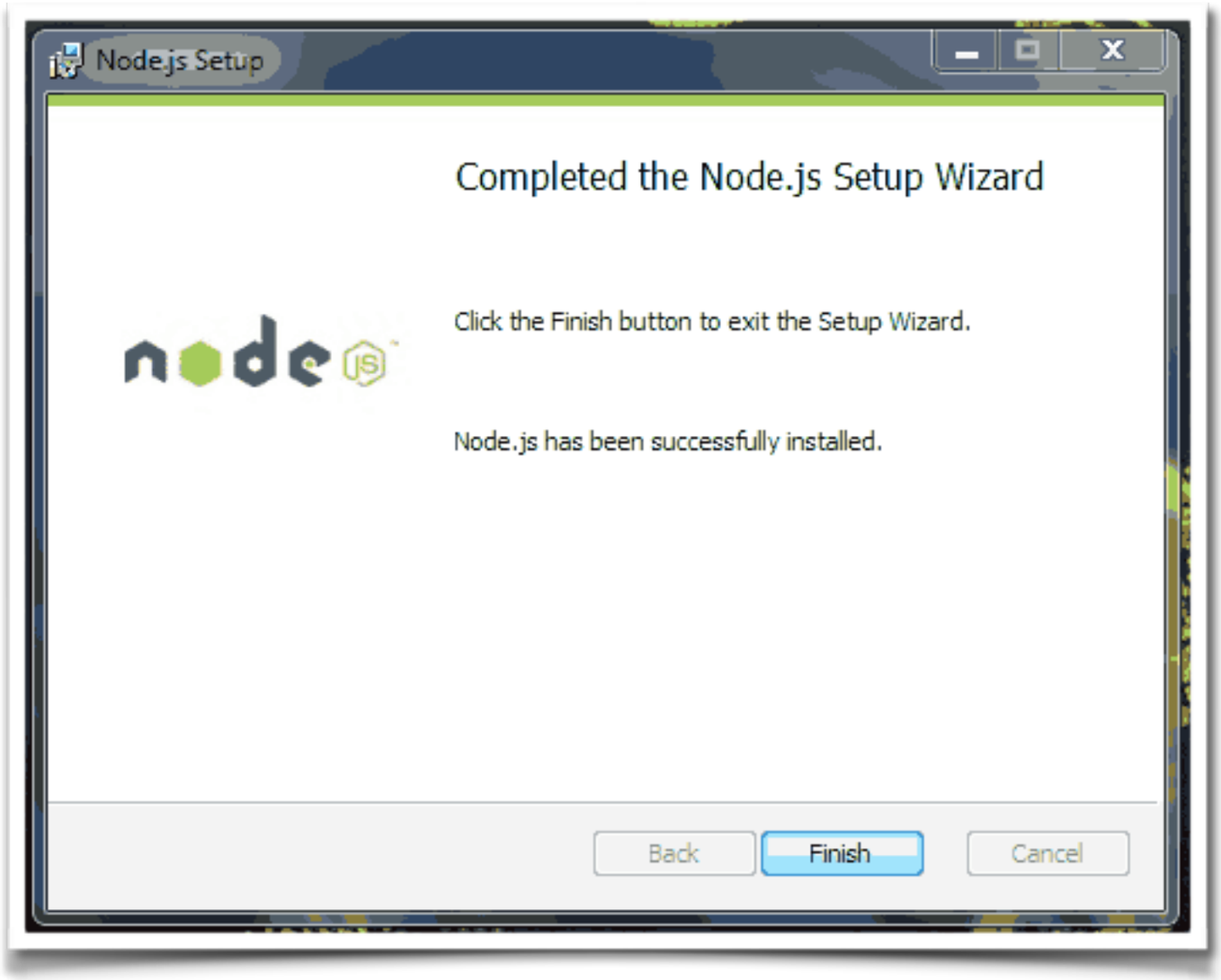
Install Node.js on Windows

- You will see the installation progress as shown below.



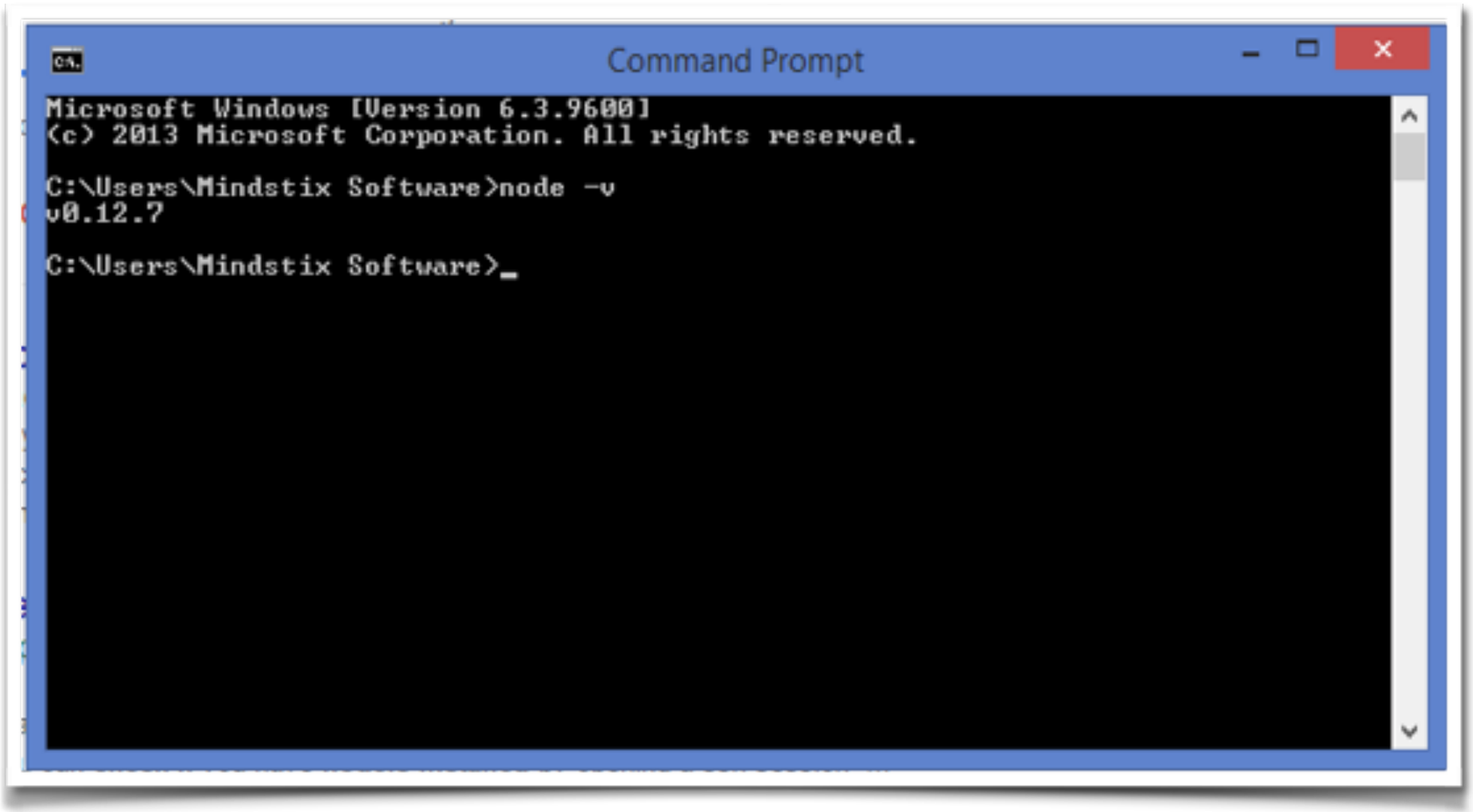
Install Node.js on Windows

- Node.js has been successfully installed. Click Finish to exit the Node.js Setup Wizard.



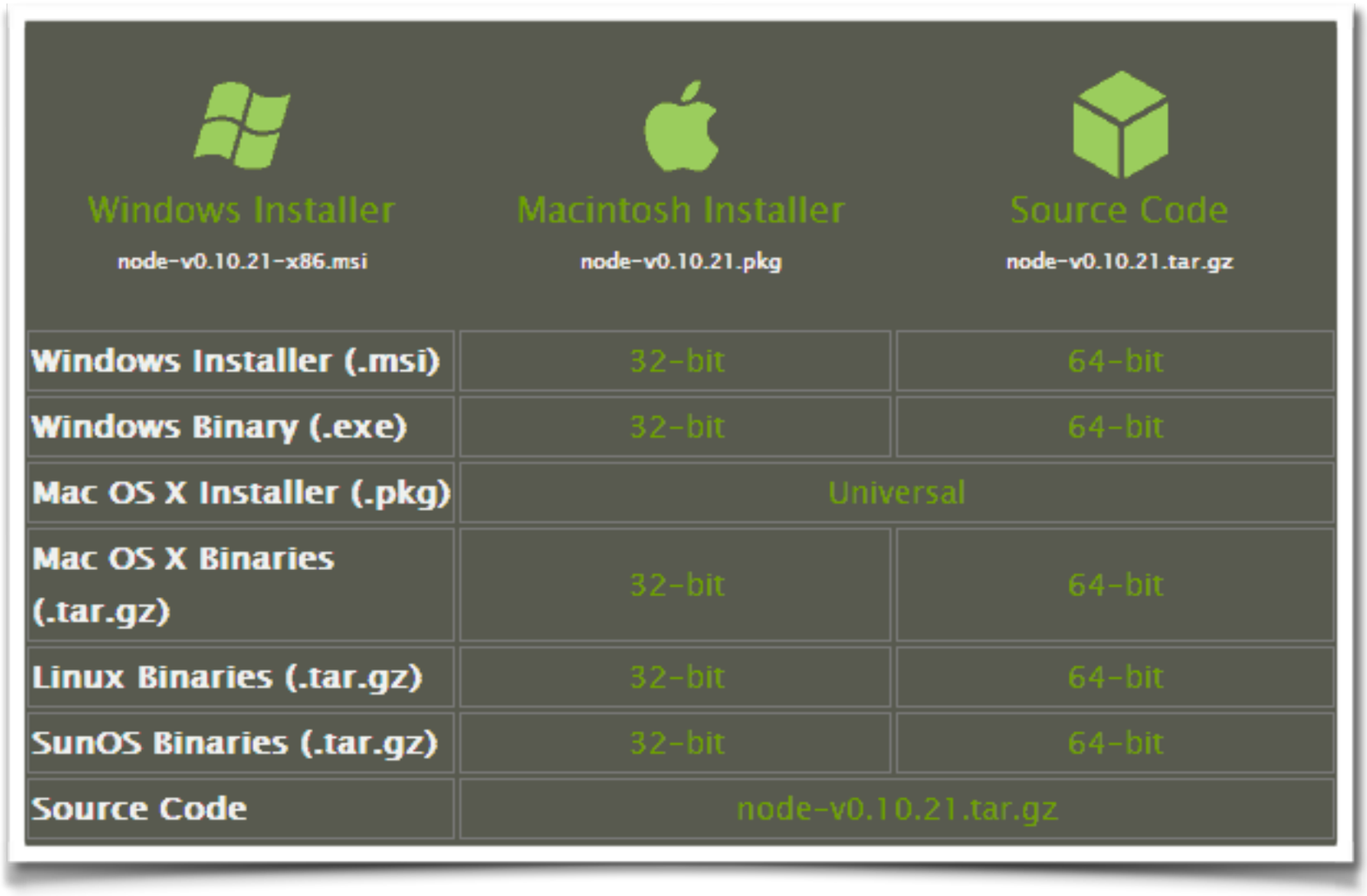
Install Node.js on Windows

- To validate whether Node.js was installed successfully or not, run '**node -v**' command on command prompt. It will return you the version of Node you just installed.



Install Node.js on Mac OS

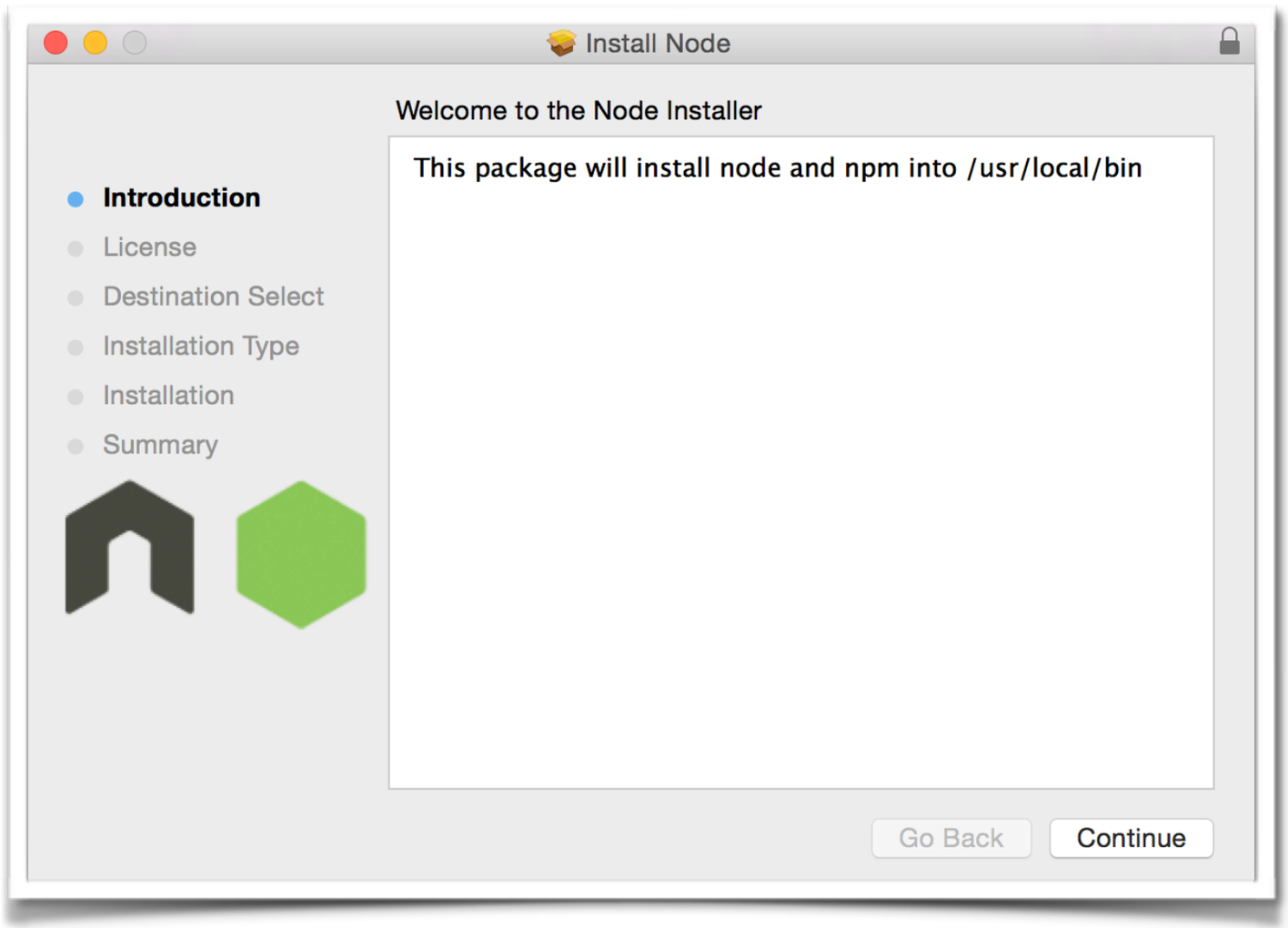
- Download Node.js installer (.pkg) for Mac OS from https://nodejs.org/en/download/ and follow the instructions to setup Node.js on Mac OS.



Windows Installer node-v0.10.21-x86.msi	Macintosh Installer node-v0.10.21.pkg	Source Code node-v0.10.21.tar.gz
Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Installer (.pkg)	Universal	
Mac OS X Binaries (.tar.gz)	32-bit	64-bit
Linux Binaries (.tar.gz)	32-bit	64-bit
SunOS Binaries (.tar.gz)	32-bit	64-bit
Source Code	node-v0.10.21.tar.gz	

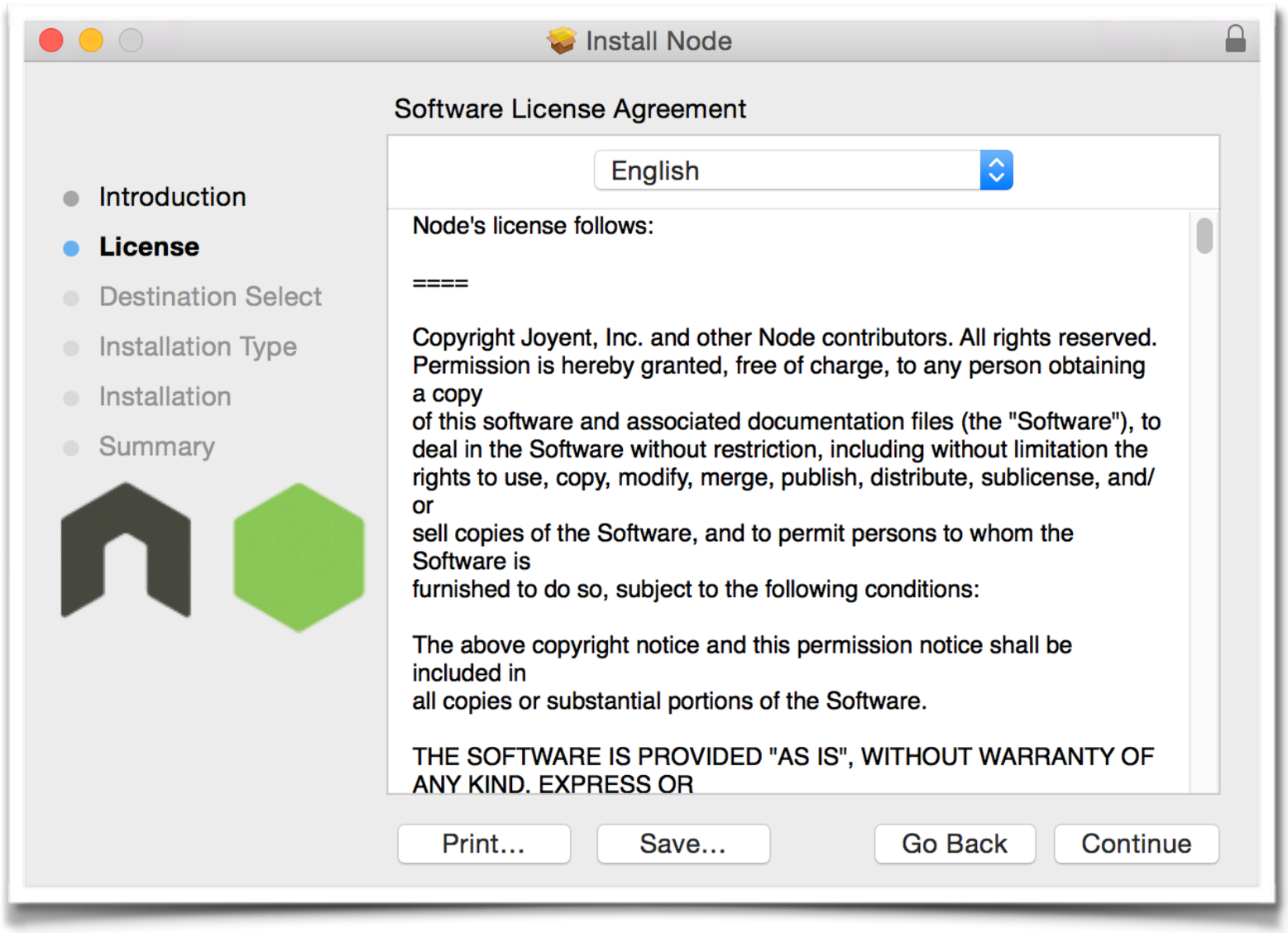
Install Node.js on Mac OS

- After downloading, double-click on the (.pkg) file to start the Node.js Setup Wizard. Click Continue.



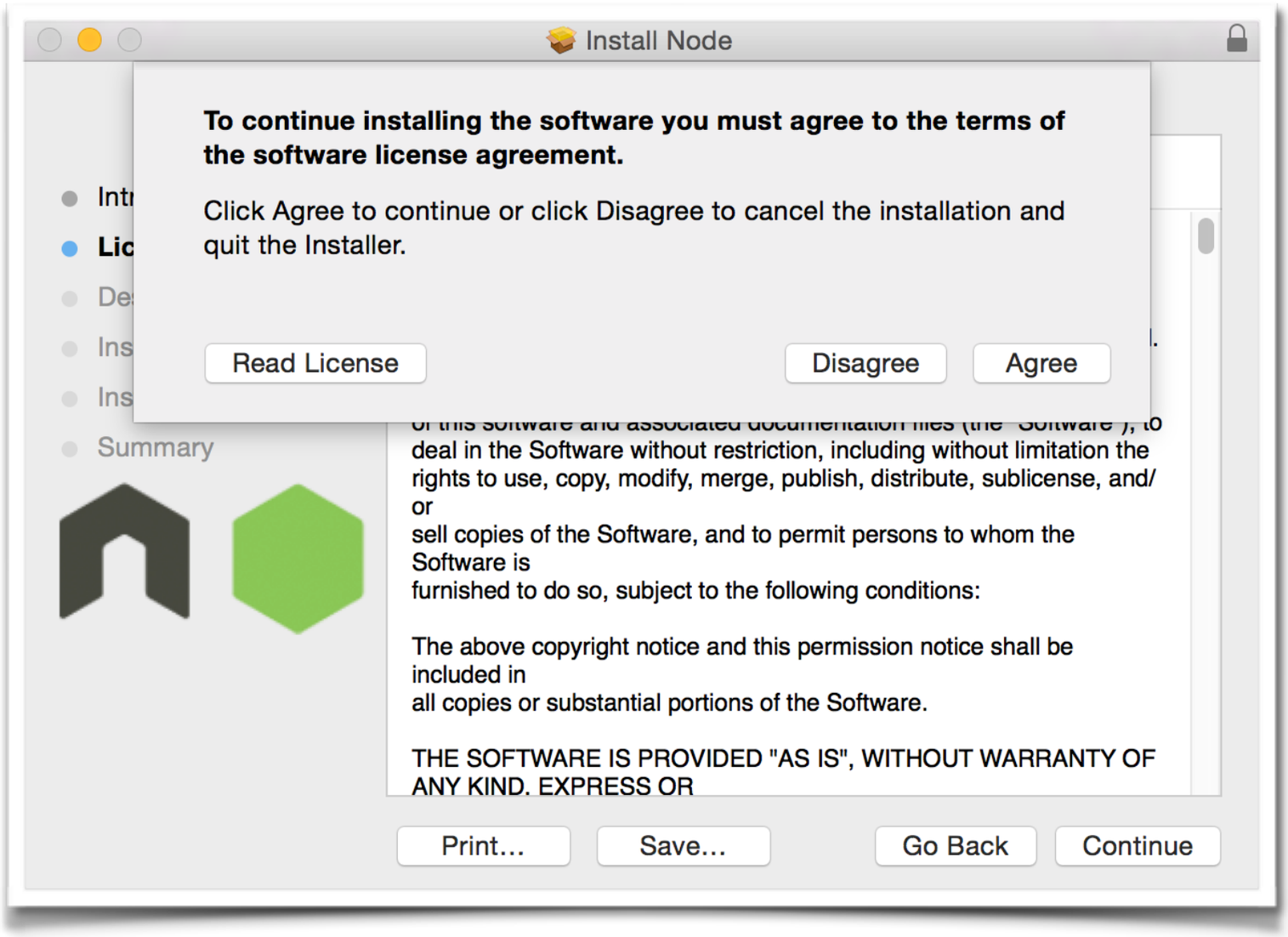
Install Node.js on Mac OS

- If you accept the End-User License Agreement. Click Continue.



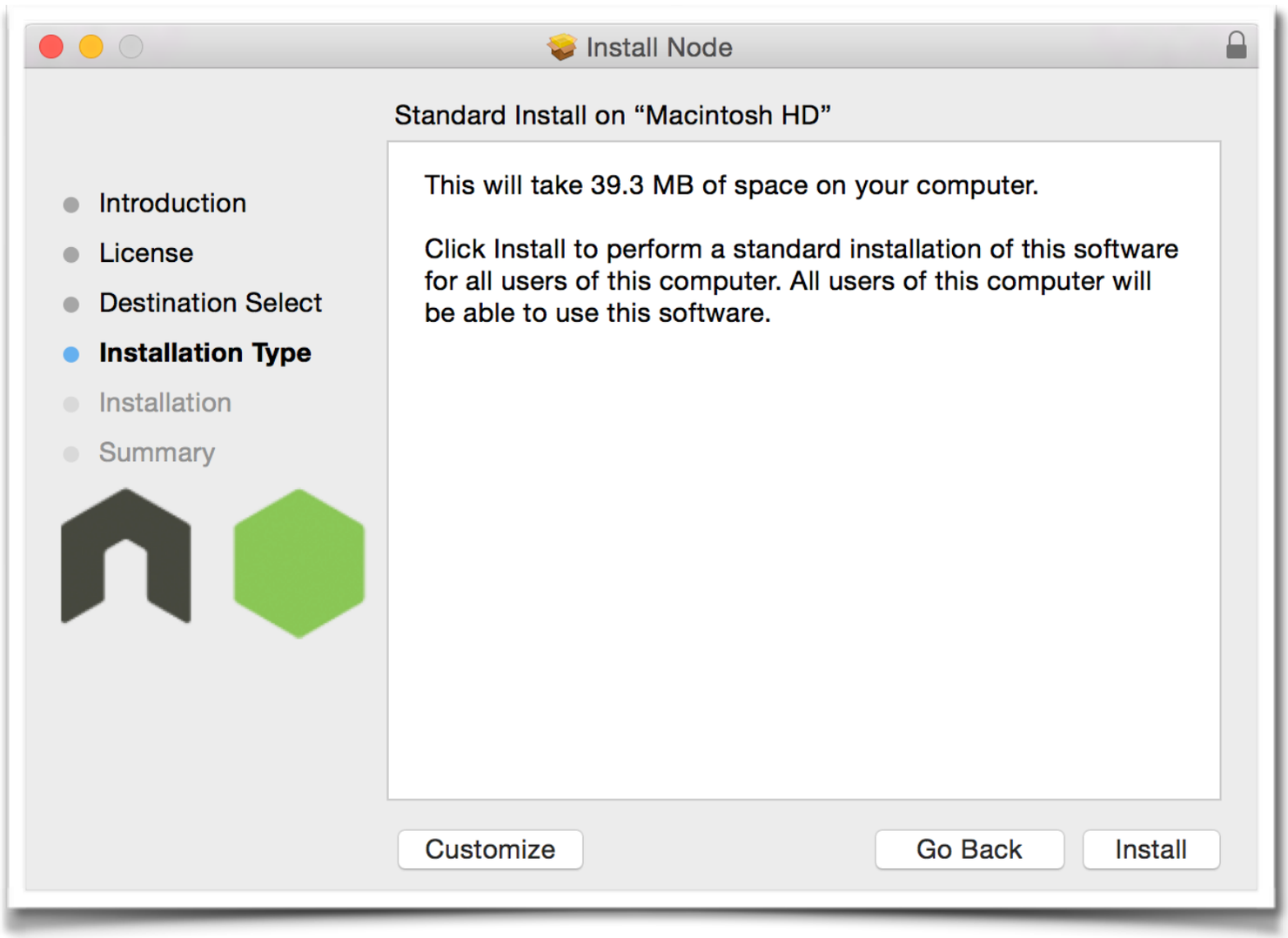
Install Node.js on Mac OS

- If you accept the End-User License Agreement. Click Agree.



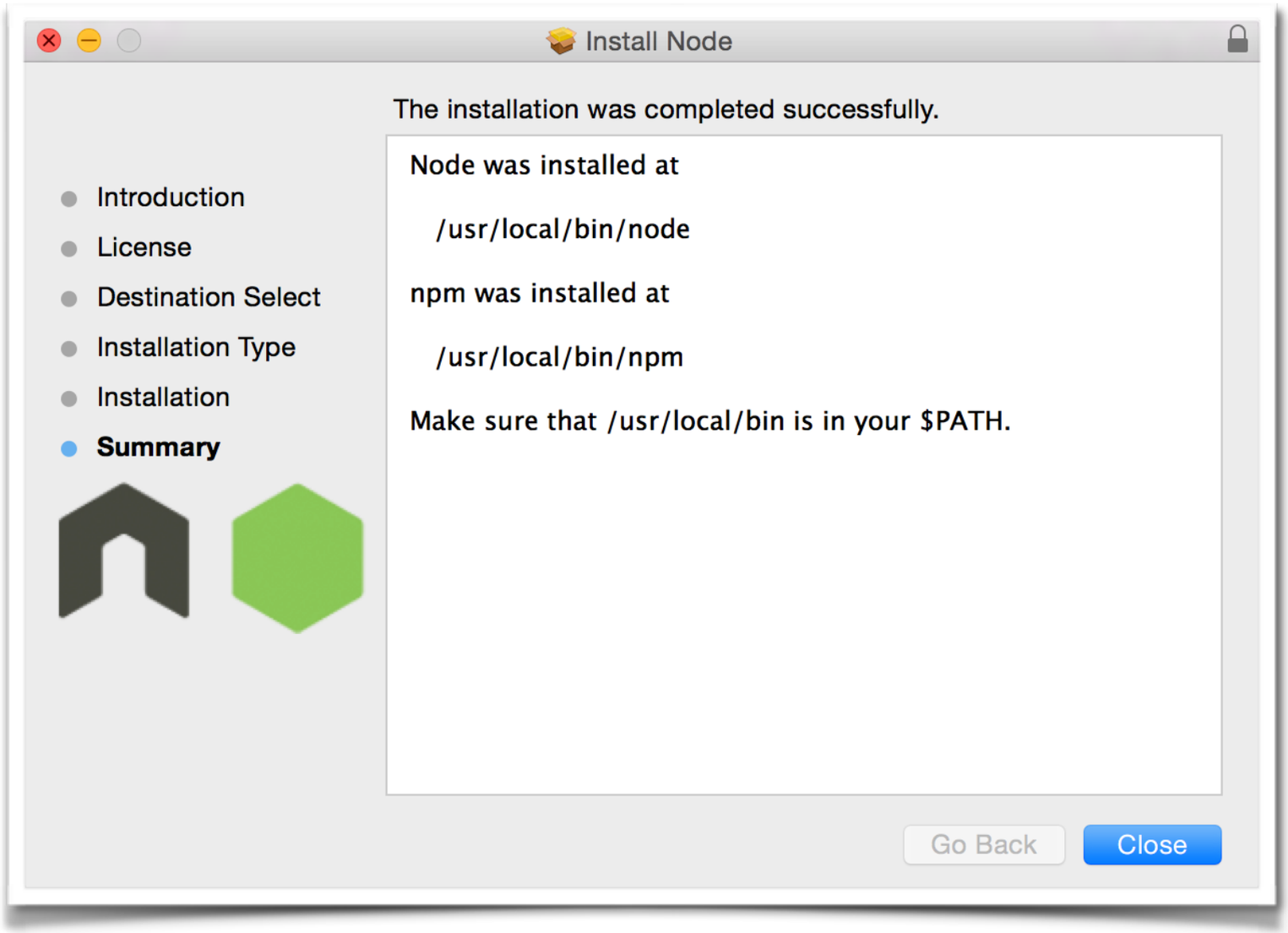
Install Node.js on Mac OS

- Click Install to perform standard installation.



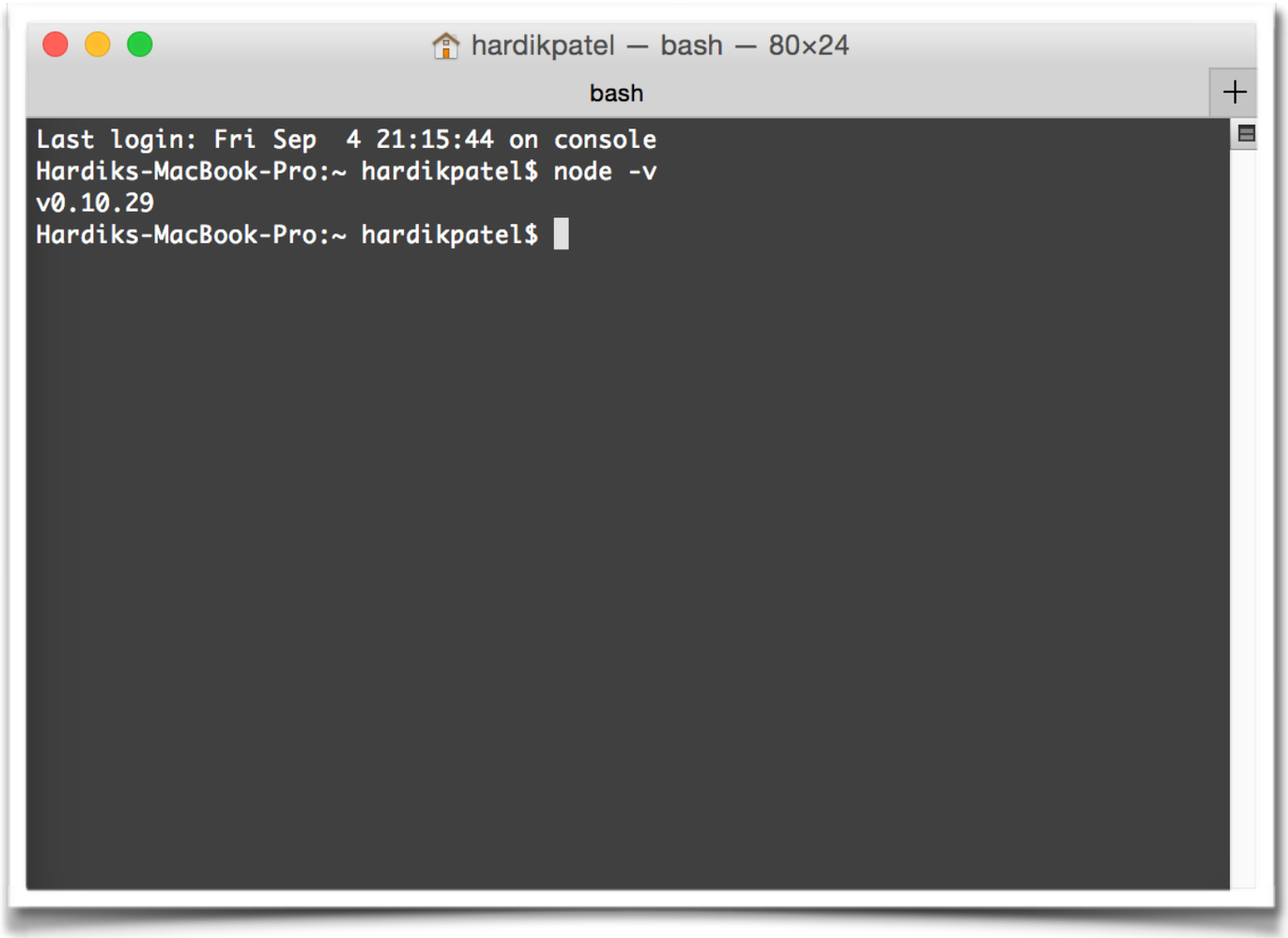
Install Node.js on Mac OS

- Node.js has been successfully installed. Click Close to exit the Node.js Setup Wizard.



Install Node.js on Mac OS

- To validate whether Node.js was installed successfully or not, run '**node -v**' command on terminal. It will return you the version of Node you just installed.



Install Node.js on Linux

- There are three ways to install Node.js on Linux platform.
  - By using package manager (apt-get for Ubuntu/ Debian)
  - By compiling source code on the targeted machine.
  - By downloading pre-compiled binaries from https://nodejs.org/en/download/

Install Node.js on Linux

By using package manager:

- In order to get the latest version, we should first refresh our local package index.

```
$ sudo apt-get update
```

- Now, install Node.js from repository using package manager.

```
$ sudo apt-get install nodejs
```

Install Node.js on Linux

By using package manager:

- You need to install NPM separately. It's Node's package manager.

```
$ sudo apt-get install npm
```

- NPM will allow you to install third party node modules and packages.

Install Node.js on Linux

By compiling source code:

- You can get the latest source code from https://nodejs.org/en/download/

- You can download it from browser directly or by using command,

```
$ wget http://nodejs.org/dist/node-v0.12.7.tar.gz
```

- Extract downloaded .tar.gz file. You can extract it by using ‘Archive Manager’ or by using command,

```
$ tar -xzf node-v0.12.7.tar.gz
```

Install Node.js on Linux

By compiling source code:

- Now, move to extracted directory and run following commands to compile the source code.

```
$ cd node-v0.12.7

$ ./configure

$ sudo make install
```

Install Node.js on Linux

By downloading pre-compiled binaries:

- You can get the latest pre-compiled binaries from https://nodejs.org/en/download/

- You can download it from browser directly or by using command,

```
$ wget https://nodejs.org/dist/node-v0.12.7-linux-x86.tar.gz
```

- Extract downloaded .tar.gz file. You can extract it by using 'Archive Manager' or by using command,

```
$ tar -xzf node-v0.12.7-linux-x86.tar.gz
```

Install Node.js on Linux

By downloading pre-compiled binaries:

- You can directly use the extracted directory. Pre-compiled binaries doesn't need any installation. So, move extracted directory at the desired destination directory.

- However, to use 'node' and 'npm' commands from anywhere, you need to export paths of node and npm binaries.

- To add node and nom paths open '/etc/environment' file.

```
$ sudo gedit /etc/environment
```

- You will see PATH variable in the file contents. Below is example for reference,

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
```

Install Node.js on Linux

By downloading pre-compiled binaries:

- You need to add paths of node and npm binaries to PATH variable. Node and npm binaries are present in '**bin**' directory of downloaded node binary.

- For example, if you moved downloaded node binary to user's home directory then paths of node and nom binaries will be as below,

  /home/{user_name}/node-v0.12.7-linux-x86/bin/node

  /home/{user_name}/node-v0.12.7-linux-x86/bin/npm

- Append node and npm binaries paths to PATH variable. See below for reference,

  PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/mindstix/node-v0.12.7-linux-x86/bin/node:/home/mindstix/node-v0.12.7-linux-x86/bin/npm"

- Save and exit from file.

NPM (Node Package Manager)

- When you install Node.js, you get the utility called ‘NPM’ i.e. node package manager.
- NPM is used to install node modules and packages from npm repository (https://www.npmjs.com/)
- It is the way to extend the functionality of Node.js.
- Main tasks of NPM:
  - Provides online repository for Node.js modules.
  - Provides command line utility to install Node.js modules.
  - Do version and dependency management for Node.js packages.

NPM (Node Package Manager)

- With node package manager, you can also use the ‘package.json’ file, which allows you to specify the dependencies that your Node.js application needs.

- It usually contains node application name, version, license and dependencies that your Node.js application will need.

- For example,

```
{
  “name”: “MyNodeApp”,
  “version”: “0.0.1”,
  “license”: “MIT”,
  “dependencies”: {
    “extfs”: “~0.0.7”,
    “xml2js”: “~0.4.11”
  }
}
```

NPM (Node Package Manager)

- In package.json file, under dependencies we write the node modules, which are needed in our Node.js application. We need to provide valid node module name and version in front of them.

- Significance of '~' character before version number, means approximate version number.

- More usage guidelines about package.json can be found here, https://docs.npmjs.com/files/package.json

- Package.json is more useful when Node.js application is being developed in one environment and it is going to be run in another environment. You can simply install all node dependencies using package.json.

Node - ‘Hello World’

- Here, we will see how to write a simple ‘Hello World’ Node application.

- What you will need to start with?

  - Node.js installed (as described in earlier slides).

  - A simple text editor.

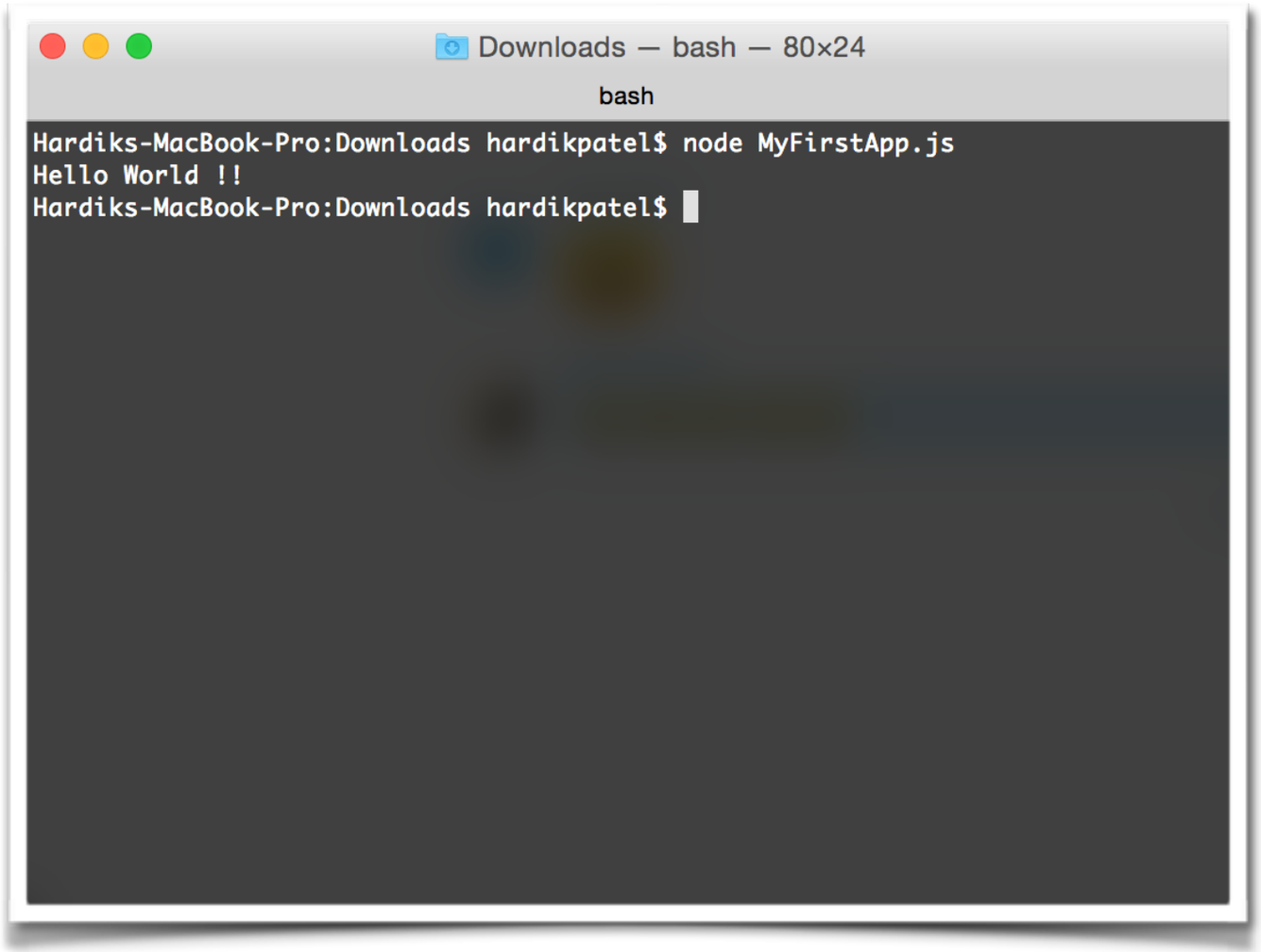
- Open a text editor and write following lines of code.

```
// My first Node.js program.
console.log(“Hello World !!”);
```

Node - ‘Hello World’

- Save file with ‘.js’ extension. Let’s say, ‘MyFirstApp.js’ and run using following command.

  $ node MyFirstApp.js



Concept of Callback

- A callback is a piece of executable code that is passed as an argument to other code, which is expected to call back (execute) the argument at some convenient time.

- To understand callback functions you first have to understand regular functions. This might seen like a “duh” thing to say, but functions in Javascript are a bit odd.

- Functions in Javascript are actually objects. Specifically, they’re Function objects created with the Function constructor. A Function object contains a string which contains the Javascript code of the function.

```
// You can create a function by passing the
// function constructor a string of code.

var multiply = new Function("arg1", "arg2",
                                    "return arg1 * arg2;");

multiply(5,10); // => 50
```

- One benefit of this function-as-object concept is that you can pass code to another function in the same way you would pass a regular variable or object (because the code is literally just an object).

Concept of Callback

Passing a function as a callback:

- Passing a function as an argument is easy.

```
// Define our function with the callback argument
function myFunction(arg1, arg2, callback) {

    // This generates a random number between arg1 and arg2
    var number = Math.ceil(Math.random() * (arg1 - arg2) + arg2);

    // Then we're done, so we'll call the callback and pass our result
    callback(number);
}

// Call the function
myFunction(5, 15, function(num) {

    // This anonymous function will run when
    // the callback is called
    console.log("Callback called: " + num);
});
```

- It might seem silly to go through all that trouble when the value could just be returned normally, but there are situations where that's impractical and callbacks are necessary.

Concept of Callback

Don't block the way:

- Traditionally functions work by taking input in the form of arguments and returning a value using a return statement (ideally a single return statement at the end of the function: one entry point and one exit point). This makes sense. Functions are essentially mappings between input and output.

- Javascript gives us an option to do things a bit differently. Rather than wait around for a function to finish by returning a value, we can use callbacks to do it asynchronously.

- This is useful for things that take a while to finish, like performing I/O. If we want to read a huge file, which takes time to get entire file contents into memory. We can keep on doing other things while waiting for the callback to be called. In fact, very often we are required (or, rather, strongly encouraged) to do things asynchronously in Javascript.

Concept of Callback

```
// Read file from disk.
fs.readfile('example.txt', function readData(error, data) {
    // Gets executed when 'fs' returns callback.
    console.log("Inside callback function.");
});

// Continue execution, don't wait for
// 'fs' to return a callback.
console.log("Continued executing after I/O.");
```

- In this example we are reading contents of a text file. The typical paradigm of returning a value at the bottom of the function no longer works here. Our request is handled asynchronously, meaning that we start the file read operation and tell it to call our function when it finishes.

- In the meantime, the statements immediately after '`fs.readfile`' keeps on executing. So, the `console.log` statement won't wait for `fs.readfile` to get completed. It will get executed and later when file contents are returned function 'readData' will get executed.