Asynchronous Programming in Node

Example: Read contents of a File

Blocking

```
Read file from Filesystem, set equal to “contents”
Print contents
Do something else
```

Non Blocking

```
Read file from Filesystem
  whenever you’re complete, print the contents
Do Something else
```

callback

test.txt

```
- one
- two
- three
```

Synchronous Function - readFileSync

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

function readTheFiles() {
  console.log("about to read...");
  const contents = fs.readFileSync("test.txt", "utf8");
  console.log(contents);
  console.log("...done");
}

readTheFiles();
```

output:

```
about to read...
- one
- two
- three
...done
```

test.txt

```
- one
- two
- three
```

Callback - anonymous function

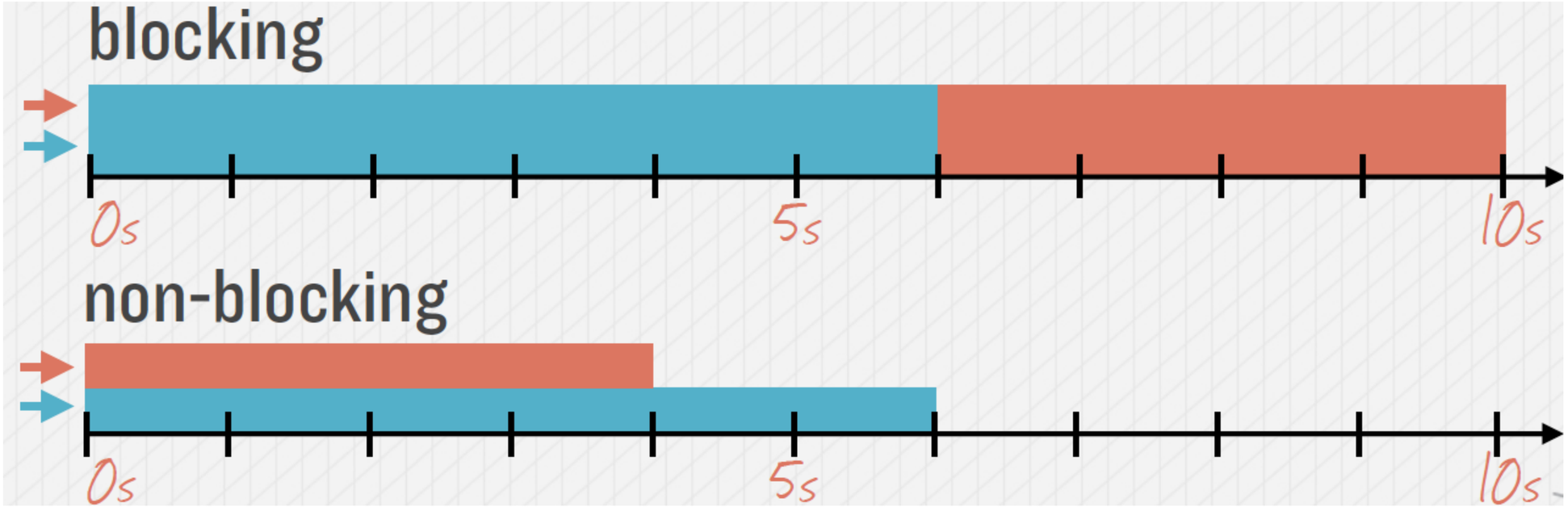
```
const fs = require("fs");

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test.txt", "utf8", function(err, contents) {
    console.log(contents);
  });
  console.log("...done");
}

readTheFiles();
```

output:

```
about to read...
...done
- one
- two
- three
```

Blocking vs Non-blocking Performance

```
const contents1 = fs.readFileSync("test1.txt", "utf8");
const contents2 = fs.readFileSync("test2.txt", "utf8");
console.log(contents1);
console.log(contents2);
```




```
const readFunc = function(err, contents) {
  console.log(contents);
}
fs.readFile("test1.txt", "utf8", readFunc);
fs.readFile("test2.txt", "utf8", readFunc);
```

node.js Hello World

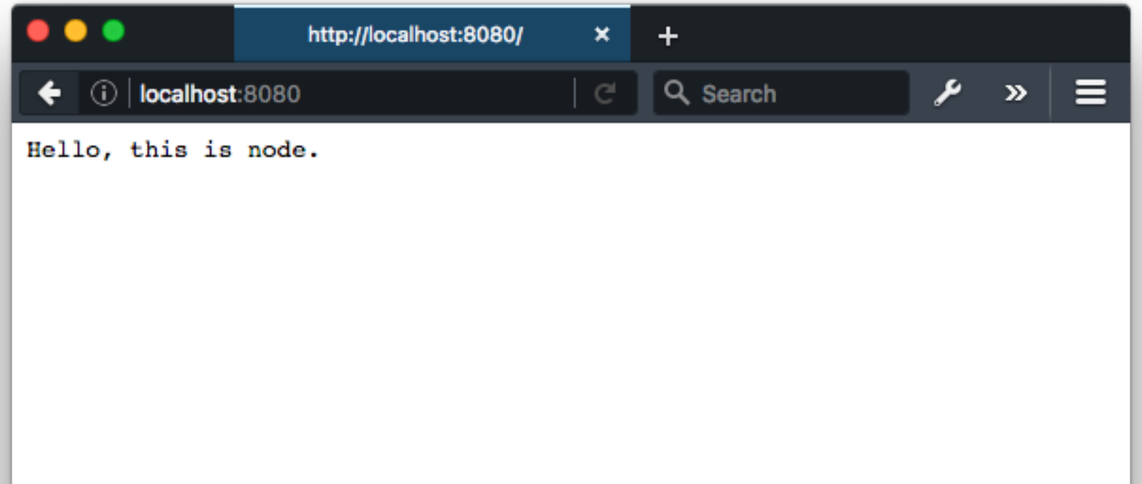
```
const http = require('http');
http.createServer(function(request, response) {
  response.writeHead(200);
  response.write("Hello, this is node.");
  response.end();
}).listen(8080);
console.log('Listening on port 8080...');
```

```
$ node hello.js
```

```
Listening on port 8080...
```

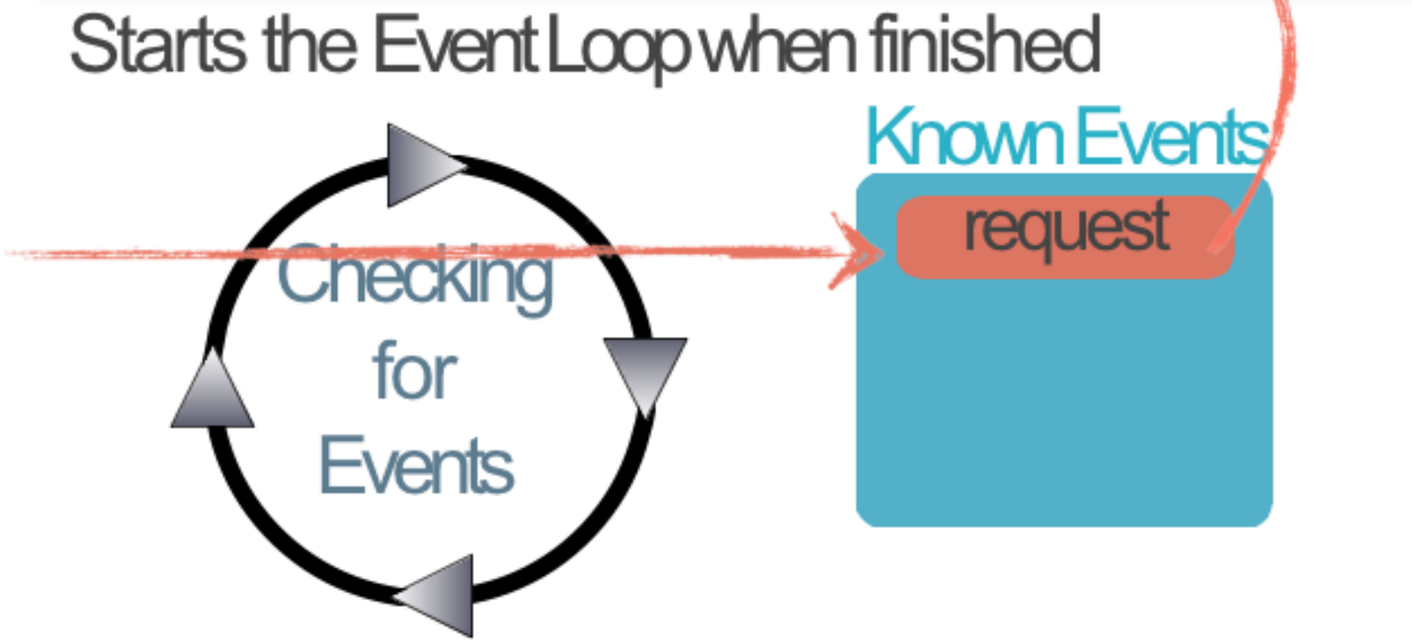
```
$ curl http://localhost:8080
```

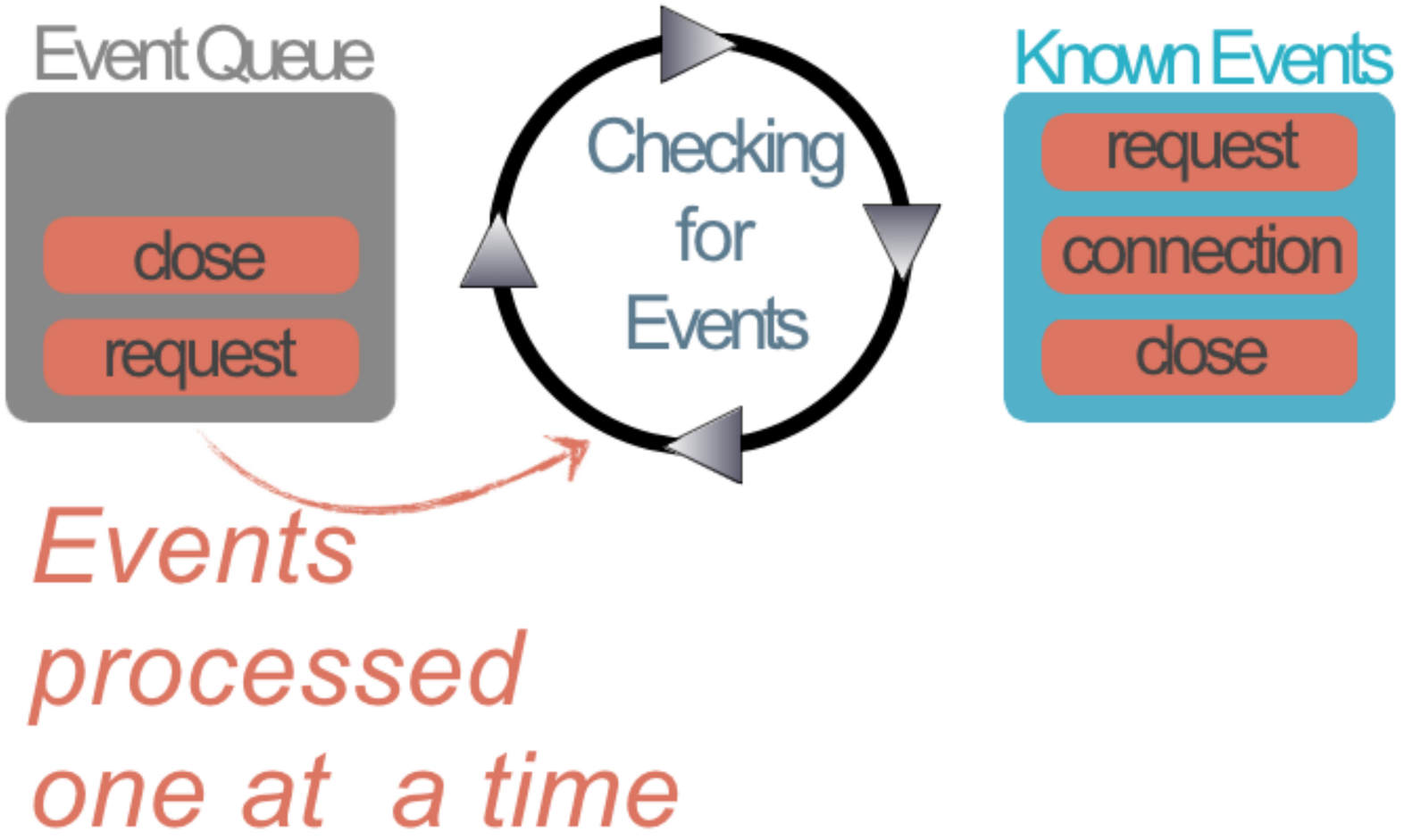
```
Hello, this is node.
```



The Event Loop

```
var http = require('http');
http.createServer(function(request, response) {
  ...
}).listen(8080);
console.log('Listening on port 8080...');
```




Event Queue
close
request
Checking for Events
Known Events
request
connection
close
Events processed one at a time

Avoid Blocking Calls

Typical potential blocking calls

- Calls out to web services
- Reads/Writes on the Database
- Calls to extensions

- Synchronous version of these types of calls must be avoided in node applications
- Instead, any activity likely to be blocked is to be called asynchronously

Callbacks => Promises => Async/Await

Asynchronous Styles

- Anonymous Callback Function
- Named Callback Function
- Promise
- Async/Await

test.txt

```
- one
- two
- three
```

Callback - anonymous function

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

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test.txt", "utf8", function(err, contents) {
    console.log(contents);
  });
  console.log("...done");
}

readTheFiles();
```

output:

```
about to read...
...done
- one
- two
- three
```

test.txt

```
- one
- two
- three
```

output:

```
about to read...
...done
- one
- two
- three
```

Callback - Named Function: readFunc

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

const readFunc = function(err, contents) {
  console.log(contents);
};

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test.txt", "utf8", readFunc);
  console.log("...done");
}

readTheFiles();
```

Promise Syntax - Experimental for fs

test.txt

```
- one
- two
- three
```

```
const fs = require("fs").promises;

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test.txt", "utf8").then(function (contents) {
    console.log(contents);
  });
  console.log("...done");
}

readTheFiles();
```

output:

```
about to read...
...done
- one
- two
- three
```

test.txt

```
- one
- two
- three
```

Async/Await Syntax - experimental for fs

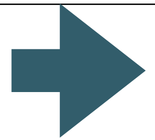
```
const fs = require("fs").promises;

async function readTheFiles() {
  console.log("about to read...");
  const contents = await fs.readFile("test.txt", "utf8");
  console.log(contents);
  console.log("...done");
}

readTheFiles();
```

output:

```
about to read...
- one
- two
- three
...done
```

For Single Asynchronous Request

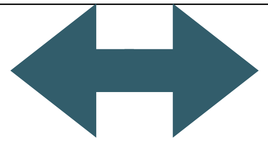
- Anonymous Callback Function
- Named Callback Function
- Promise
- Async/Await



Each approach reasonable / no major advantages

For Multiple Asynchronous Request

- Anonymous Callback Function
- Named Callback Function
- Promise
- Async/Await



Potential Major Advantages to use async/await

test-1.txt

```
- one
- two
- three
```

test-2.txt

```
- four
- five
- six
```

test-3.txt

```
- seven
- eight
- nine
```

Read all three file in sequence and output them in correct order

```
- one
- two
- three
- four
- five
- six
- seven
- eight
- nine
```

Sequential Callbacks

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

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test-1.txt", "utf8", function(err, contents) {
    console.log(contents);
  });
  fs.readFile("test-2.txt", "utf8", function(err, contents) {
    console.log(contents);
  });
  fs.readFile("test-3.txt", "utf8", function(err, contents) {
    console.log(contents);
  });
  console.log("...done");
}

readTheFiles();
```

```
- four
- five
- six
- seven
- eight
- nine
- one
- two
- three
```

Ordering indeterminate

Nested Callbacks

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

function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test-1.txt", "utf8", function(err, contents) {
    console.log(contents);
    fs.readFile("test-2.txt", "utf8", function(err, contents) {
      console.log(contents);
      fs.readFile("test-3.txt", "utf8", function(err, contents) {
        console.log(contents);
      });
    });
  });
  console.log("...done");
}

readTheFiles();
```

```
- one
- two
- three
- four
- five
- six
- seven
- eight
- nine
```

Ordering determinate

Async/Await

```
const fs = require("fs").promises;

async function readTheFiles() {
  console.log("about to read...");
  const contents1 = await fs.readFile("test-1.txt", "utf8");
  console.log(contents1);
  const contents2 = await fs.readFile("test-2.txt", "utf8");
  console.log(contents2);
  const contents3 = await fs.readFile("test-3.txt", "utf8");
  console.log(contents3);
  console.log("...done");
}

readTheFiles();
```

```
- one
- two
- three
- four
- five
- six
- seven
- eight
- nine
```

Ordering determinate

```
function readTheFiles() {
  console.log("about to read...");
  fs.readFile("test-1.txt", "utf8", function(err, contents) {
    console.log(contents);
    fs.readFile("test-2.txt", "utf8", function(err, contents) {
      console.log(contents);
      fs.readFile("test-3.txt", "utf8", function(err, contents) {
        console.log(contents);
      });
    });
  });
  console.log("...done");
}
```

Leads to
“Callback Hell”
Hard to read/
debug

```
async function readTheFiles() {
  console.log("about to read...");
  const contents1 = await fs.readFile("test-1.txt", "utf8");
  console.log(contents1);
  const contents2 = await fs.readFile("test-2.txt", "utf8");
  console.log(contents2);
  const contents3 = await fs.readFile("test-3.txt", "utf8");
  console.log(contents3);
  console.log("...done");
}
```

Retains
synchronous
control flow.
More natural
to read/debug