

Asynchronous Patterns in JavaScript

Callbacks

Promises

Async/Await

Synchronous vs. Asynchronous

Buying A Book

- **Synchronous:** You go to your local book store, wait impatiently in a queue, then pay for the book and take it home.
- **Asynchronous:** You order the book on Amazon, and then get on with other things in your life. At some indeterminate point in the future, when the book is ready for you, the postman raises a knocking event on your door so that the book can be delivered to you.

Sync Programming is Easy

```
function getStockPrice(name) {  
    let symbol = getStockSymbol(name);  
    let price = getStockPrice(symbol);  
    return price;  
}
```

Call a function, suspend the caller
and wait for the return value to arrive

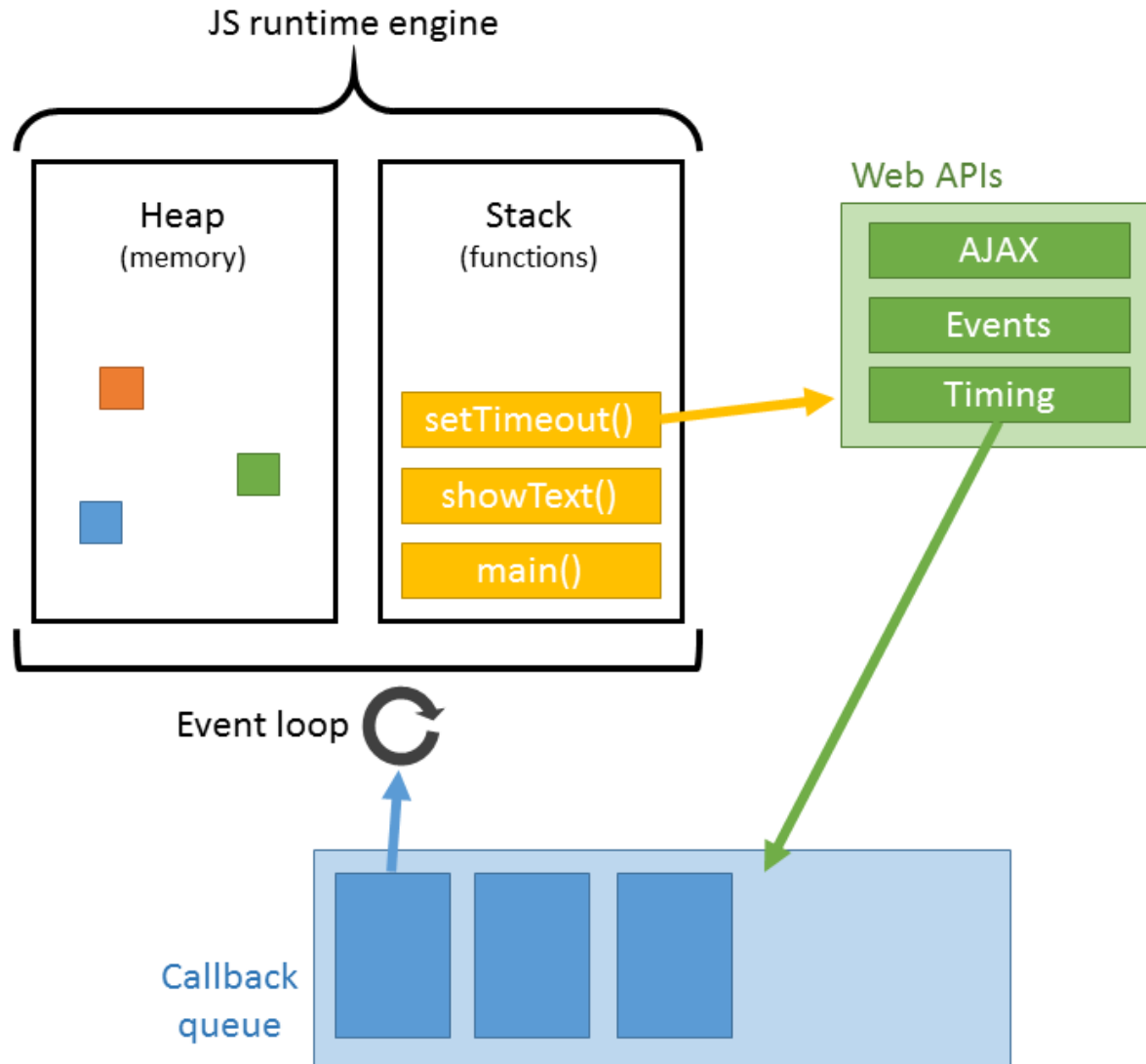
Synchronous Programming Problems

- CPU demanding tasks delay execution of all other tasks => **UI may become unresponsive**
- Accessing resources blocks entire program
 - Especially problematic with web resources
 - Resource may be large
 - Server may hang
 - Slow connection means slow loading causing UI blocks

Why use Async Programming?

- JavaScript is single-threaded
 - Long-running operations block other operations
- Async Programming is required to **prevent blocking** on long-running operations
- Benefits:
 - ***Responsiveness: prevent blocking of the UI***
 - => Doesn't lock UI on long-running computations
 - Better server-side ***Scalability: prevent blocking of request-handling threads***

JavaScript Event Loop



Watch <https://www.youtube.com/watch?v=8aGhZQkoFbQ>

Asynchronous programming techniques

Async JavaScript programming can be done using either:

- Callbacks
- Promises
- Async/Await

Callback-oriented Programming

- A callback is a function that is passed to another function as a parameter:
 - The other function can call the passed one
 - The other function can give arguments
- Examples of callbacks:
 - Event handlers are sort-of callbacks
 - setTimeout and setInterval take a callback argument
- Problems:
 - Heavily nested functions are hard to understand
=> **Callback hell** i.e., Non-trivial to follow path of execution
 - Errors and exceptions are a nightmare to process

Callback Example

```
function getLocation() {  
    navigator.geolocation.getCurrentPosition(showPosition);  
}  
  
function showPosition(position) {  
    let p = document.getElementById("demo");  
    p.innerHTML += "Latitude: " + position.coords.latitude +  
        "<br>Longitude: " + position.coords.longitude + '<BR>';  
}
```

Callback Hell...

```
function getStockPrice(name, cb) {  
  getStockSymbol(name, (error, symbol) => {  
    if (error) {  
      cb(error);  
    }  
    else {  
      getStockPrice(symbol, (error, price) => {  
        if (error) {  
          cb(error);  
        }  
        else {  
          cb(price);  
        }  
      })  
    }  
  })  
}
```



Promises

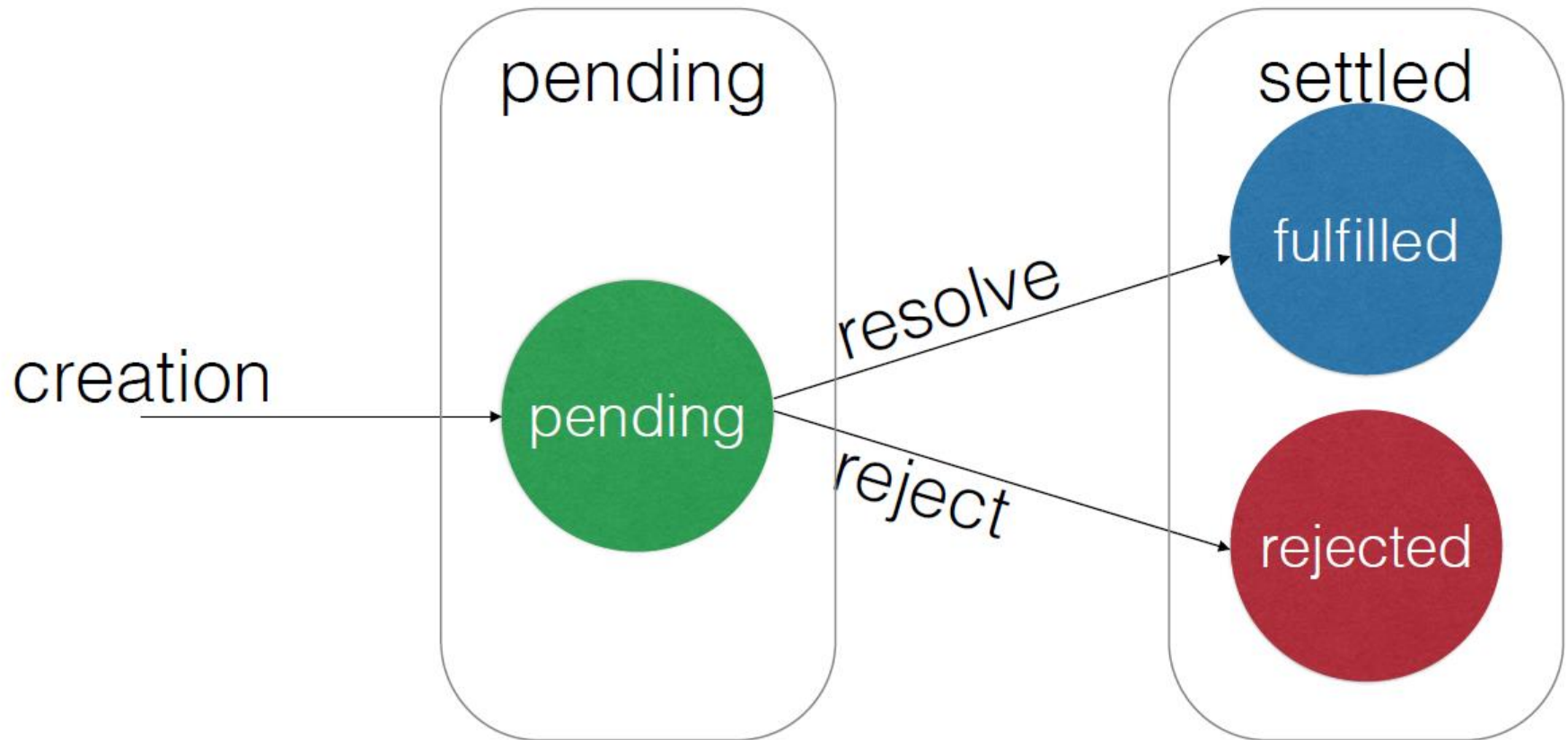
- Promise = object that represents an eventual (future) value
- A producer returns a promise which it can later fulfill or reject
- Promise has one of three states: pending, fulfilled, or rejected
- Consumers listen for state changes with **.then** method:

```
promise..then(onSuccess)
```

```
    .catch(onError);
```

- onSuccess is function to process the data received
- onError is a function to handle errors

State of a Promise



How to create a Promise

```
let promise = new Promise((resolve, reject)
=> {
    try {
        ...
        resolve(value);
    } catch(e) {
        reject(e);
    }
});
```

sync vs. async

- **sync**

```
function getStockPrice(name) {  
    var symbol = getStockSymbol(name);  
    var price = getStockPrice(symbol);  
    return price;  
}
```

- **async**

```
function getStockPrice(name) {  
    return getStockSymbol(name).  
        then(symbol => getStockPrice(symbol));  
}
```

Chaining

```
let promise = new Promise((resolve, reject)
=> {
    resolve(1);
}));
```

```
promise.then(function(val) {
    console.log(val); // 1
    return val + 2;
}).then(function(val) {
    console.log(val); // 3
});
```

Promise Usage

```
getUser()  
  .then(function(user) {  
    return getRights(user);  
  })  
  .then(function(rights) {  
    updateMenu(rights);  
  })
```

Better Syntax

```
getUser()  
  .then(user => getRights(user))  
  .then(rights => updateMenu(rights))
```


Promise Utilities

- **Promise.all** calls two or more promises and returns only when all the specified promises have completed or been rejected. The result returned is an array of values returned by the completed promises.

```
Promise.all([p1, p2, ..., pN]).then(allResults => { ... });
```

- **Promise.race** calls two or more promises and returns the first response received (and ignores the remaining ones)

```
Promise.race([p1, p2, ..., pN]).then(firstResult => { ... });
```

ES2016 async / await

- Allows easier composition of promises compared to chaining using **.then**
- async function can halt without blocking and waits for the result of a promise.

```
async function f() {  
  try {  
    var a = await Promise.resolve("foo");  
    var b = await Promise.resolve("bar");  
    var x = a + b;  
    return x;  
  } catch(err) {  
    throw err;  
  }  
}  
  
f().then(x => console.log(x))  
  .catch(err => console.error("Caught:", err));
```

Example - Getting a resource from Url using node-fetch API

- Fetch content from the server

```
let url = "https://api.github.com/users/github";
fetch(url).then(response => response.json())
    .then(user => {
        console.log(user);
    })
    .catch(err => console.log(err));
```

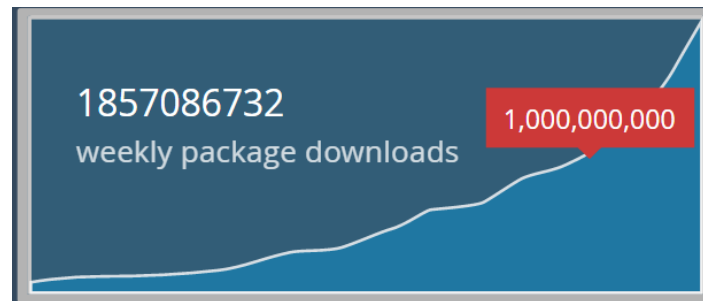
- Fetch returns a Promise. Promise-fulfilled event(**.then**) receives a **response** object.
- **.json()** method is used to get the response body into a JSON object

Node.js & Node Package Manager (NPM)

<https://www.npmjs.com/>



253,375
total packages



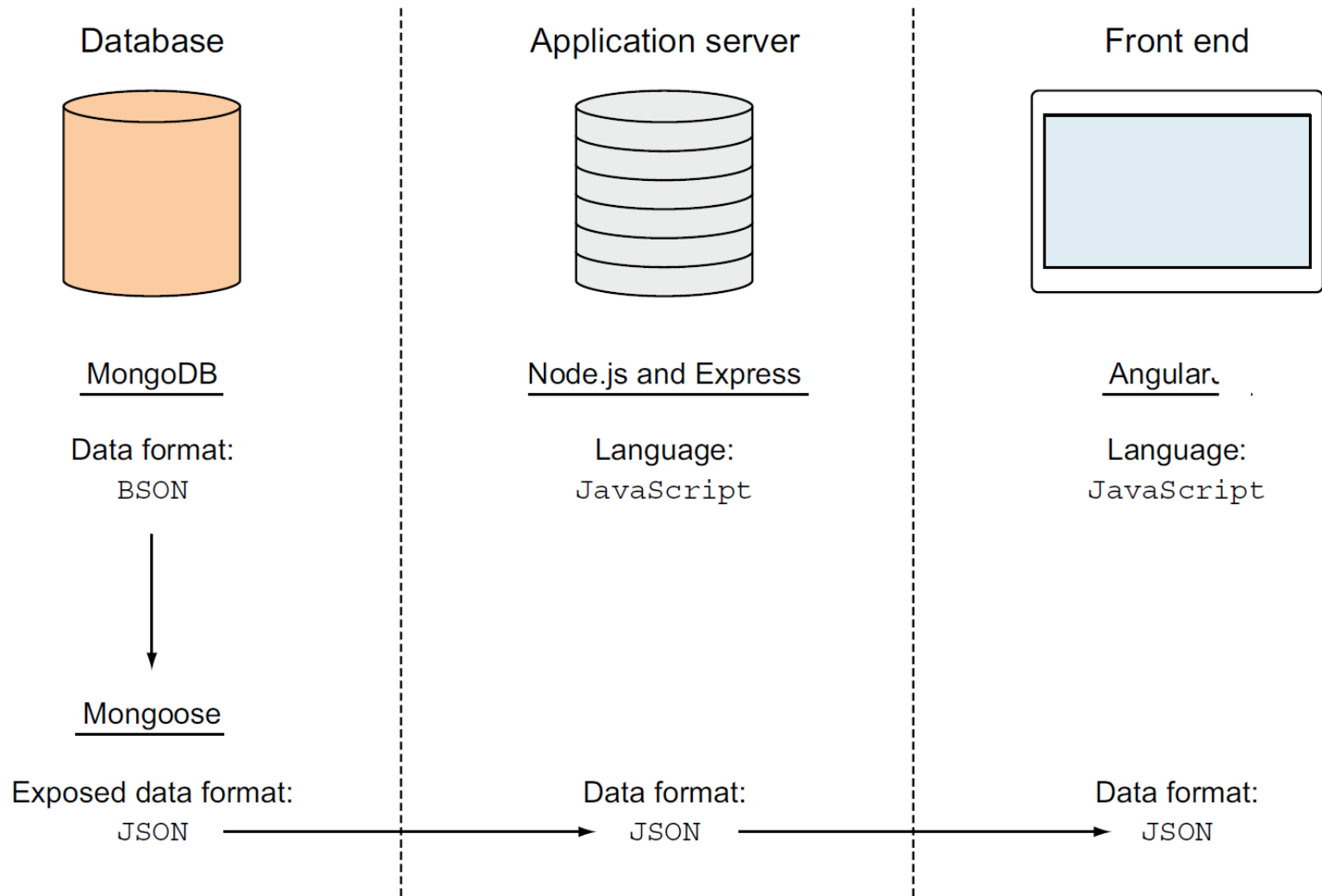
What is Node.js

- Cross platform environment for running JavaScript applications (outside the browser)
- Open source <https://nodejs.org>
- It is light, fast and easy to learn and use
- Widely used

NODE IS DEPLOYED BY BIG BRANDS Big brands are using Node to power their business

Manufacturing	Financial	eCommerce	Media	Technology
				
				
				
				
				

JavaScript is the common language throughout the **meAN** stack, and JSON is the common data format



Node.js plays such a pivotal role in the **MEAN** (Mongo, Express, Angular, and Node) stack

MEAN Stack

Database



Server



Client



Node Package Management (NPM)

- ◆ npm is used to download Node.js packages. First,

npm init

can be used to initialize an *package.json* file to define the **project dependencies**

```
$ npm init
//enter package details
name: "NPM demos"
version: 0.0.1
description: "Demos for the NPM package management"
entry point: main.js
test command: test
git repository: http://github.com/user/repository-name
keywords: npm, package management
author: ae@qu.edu.qa
license: MIT
```


Node Package Management (NPM)

- ◆ Installing modules

```
$ npm install package-name [--save]
```

- Installs a package and adds dependency in *package.json*

```
npm install node-fetch --save
```

- ◆ Do not push the downloaded packages to github by adding *node_modules/* to *.gitignore* file
- ◆ When getting a project before running it do:

```
$ npm install
```

- ◆ Installs all missing packages from *package.json*