Implementing Asynchronous Patterns

Brice Wilson

@brice_wilson www.BriceWilson.net

Overview

Why asynchronous code is important

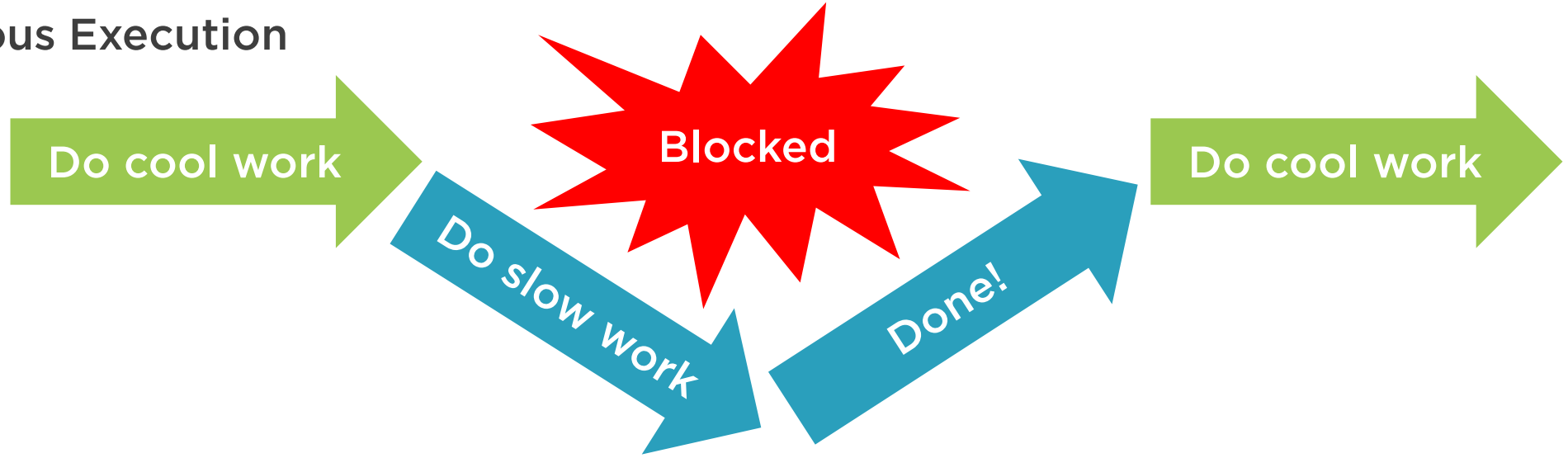
Callback functions

Promises

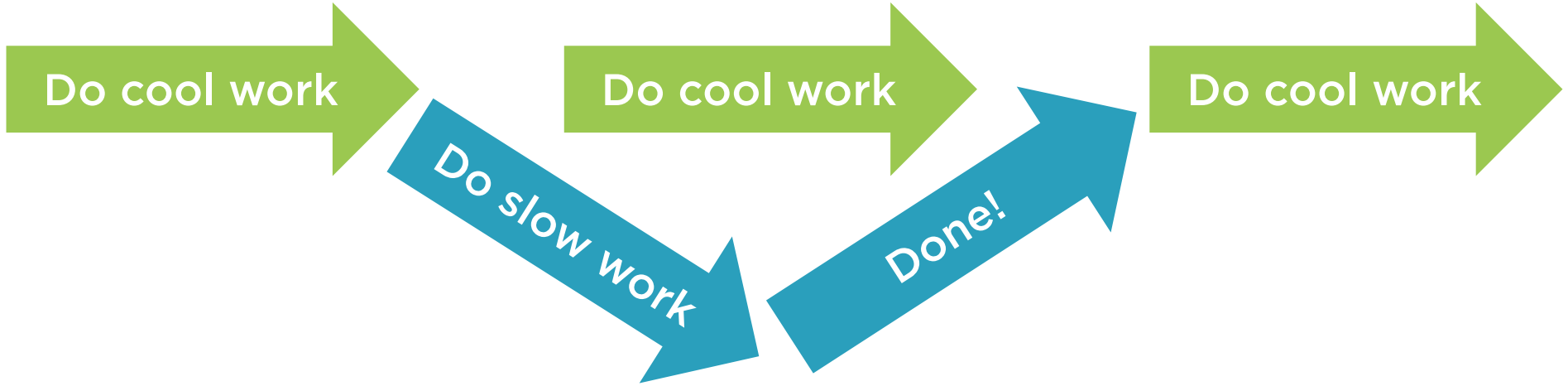
async/await

Why Asynchronous Code Matters

Synchronous Execution



Asynchronous Execution



Callback Functions

A “higher order” function may be passed functions as parameters

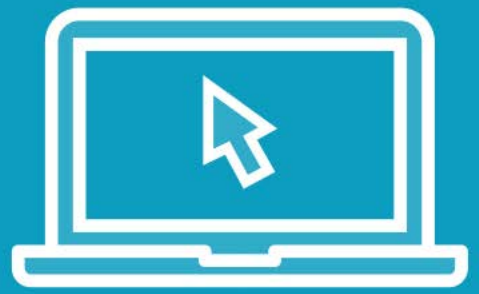
Callbacks execute after an asynchronous operation

Commonly used to process asynchronous results

May have any signature

Conventionally accept an error and a data parameter

Demo



Using callbacks with asynchronous code

Promise

The Promise object is used for asynchronous computations. A Promise represents a value which may be available now, or in the future, or never.

Mozilla Developer Network
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Promises

Native support in ES2015

- Requires TypeScript *--target* compiler option set to ES2015

Simple API

- then
- catch

Similar to Tasks in C#

May be chained together

Created by passing a function to the Promise constructor

Creating a Promise

```
function doAsyncWork(resolve, reject) {


}
```

Creating a Promise

```
function doAsyncWork(resolve, reject) {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
}
```

Creating a Promise

```
function doAsyncWork(resolve, reject) {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
}
```

Creating a Promise

```
function doAsyncWork(resolve, reject) {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
}
let p: Promise<string> = new Promise(doAsyncWork);
```

Creating a Promise

```
function doAsyncWork(resolve, reject) {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
}
let p: Promise<string> = new Promise(doAsyncWork);
```

Creating a Promise

```
function doAsyncWork(resolve, reject) {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
}
let p: Promise<string> = new Promise(doAsyncWork);

let p: Promise<string> = new Promise((resolve, reject) => {
    // perform async calls
    if (success) resolve(data);
    else reject(reason);
});
```

Handling Promise Results

```
let p: Promise<string> = MethodThatReturnsPromise();
p.then(stringData => console.log(stringData))
```

Handling Promise Results

```
let p: Promise<string> = MethodThatReturnsPromise();
p.then(stringData => console.log(stringData))
```

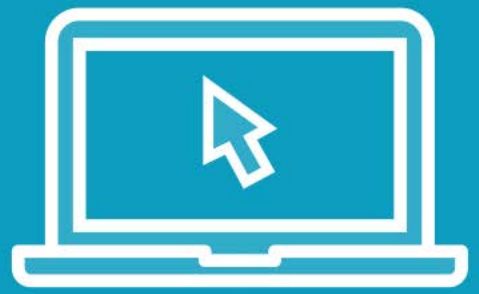
Handling Promise Results

```
let p: Promise<string> = MethodThatReturnsPromise();
p.then(stringData => console.log(stringData))
 .catch(reason => console.log(reason));
```

Handling Promise Results

```
let p: Promise<string> = MethodThatReturnsPromise();
p.then(stringData => console.log(stringData))
 .catch(reason => console.log(reason));
```

Demo



Creating and using promises

async/await

Allows code to be written more linearly

Very similar to async/await in C#

Uses ES2015 features

- Promises
- Generators
- Iterators

Coming soon to ES5

async/await Syntax

```
async function doAsyncWork() {
    let results = await GetDataFromServer();
    console.log(results);
}
```

async/await Syntax

```
async function doAsyncWork() {
    let results = await GetDataFromServer();
    console.log(results);
}
```

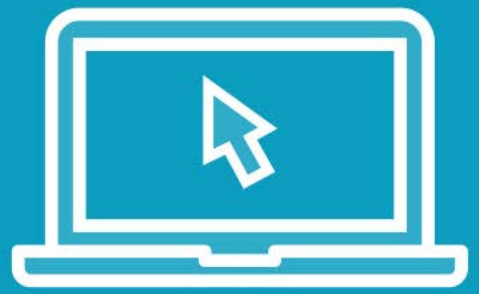
async/await Syntax

```
async function doAsyncWork() {
    let results = await GetDataFromServer();
    console.log(results);
}
```

async/await Syntax

```
async function doAsyncWork() {
    let results = await GetDataFromServer();
    console.log(results);
}
console.log('Calling server to retrieve data...');
doAsyncWork();
console.log('Results will be displayed when ready...');
```

Demo



Writing asynchronous code with async/await

Summary

Asynchronous code keeps your application responsive

Callbacks

Promises

async/await