# Creating Asynchronous Services


**Brice Wilson**

@brice_wilson www.BriceWilson.net

# Overview

What are asynchronous services?

Why should you create them?

Observables

Promises

async/await

# What Are Asynchronous Services?

**Services with methods that execute asynchronously**

**Provided to injectors like any other service**

**Injected into components like any other service**

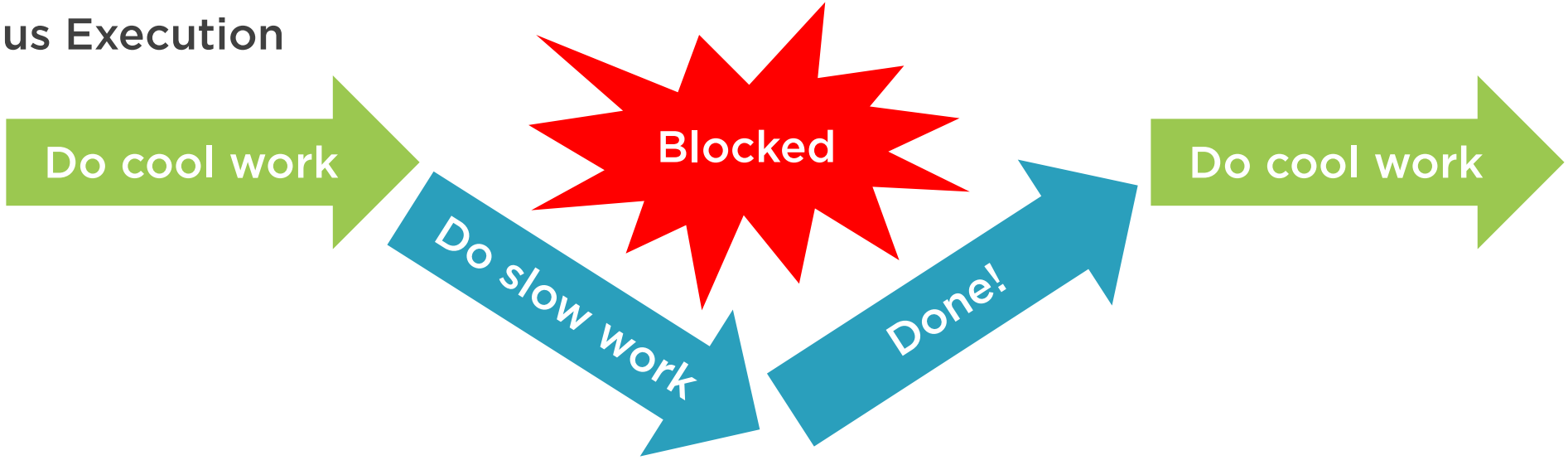
**Return different types**

- Observables
- Promises

**Callers must write additional code to process different return types**

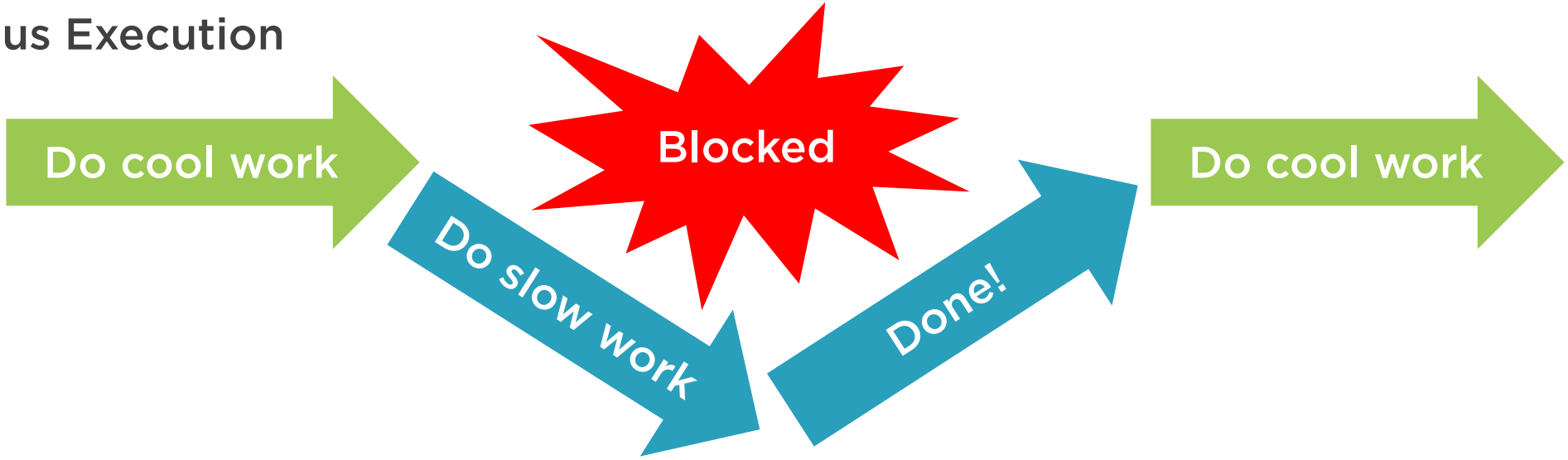
# Why Asynchronous Code Matters

**Synchronous Execution**

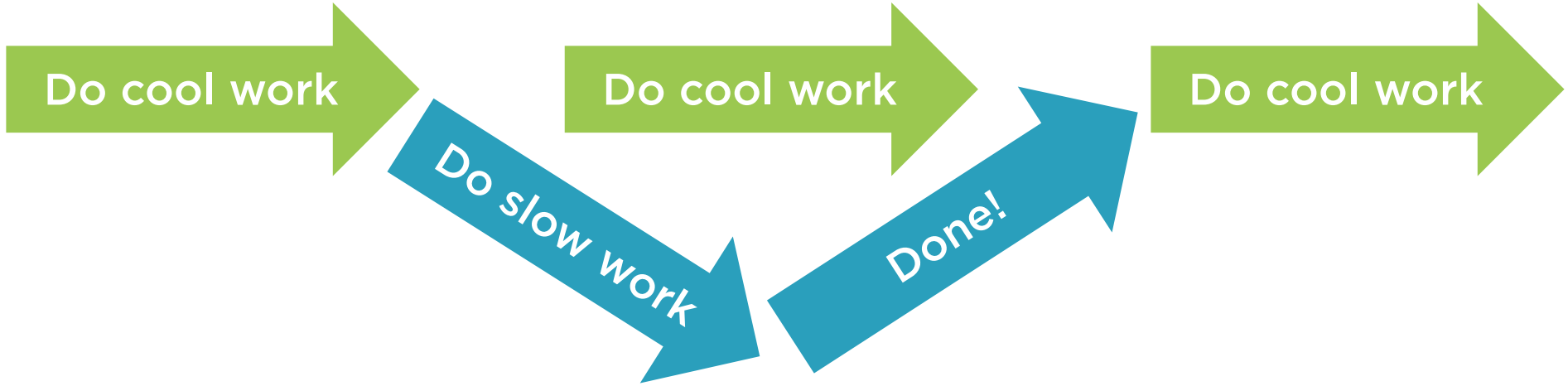


# Why Asynchronous Code Matters

## Synchronous Execution

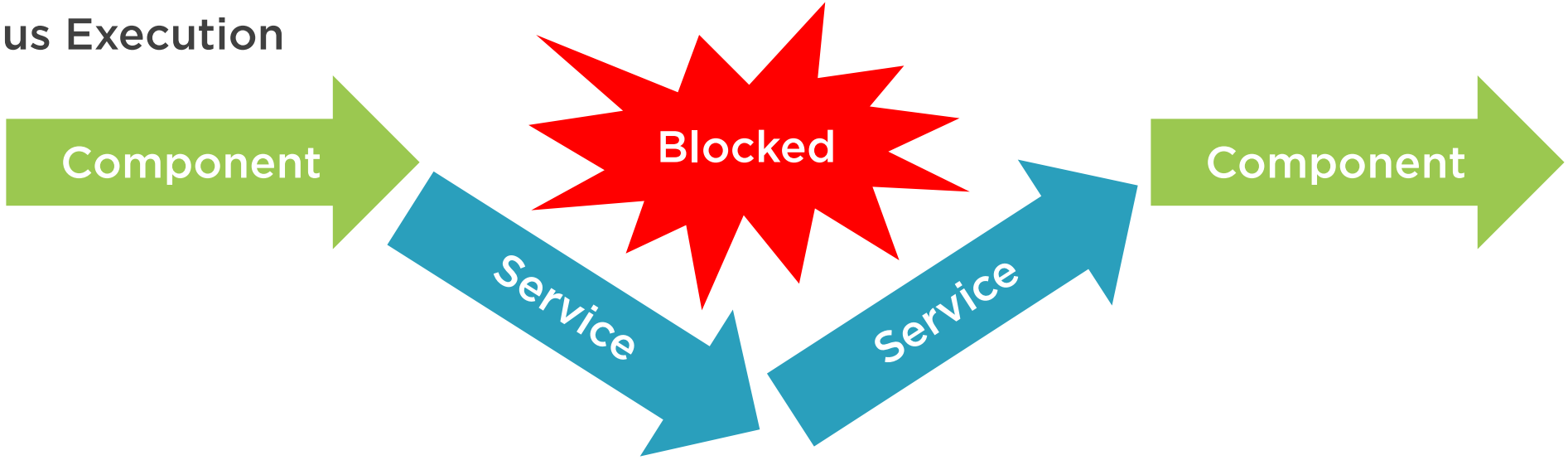



## Asynchronous Execution



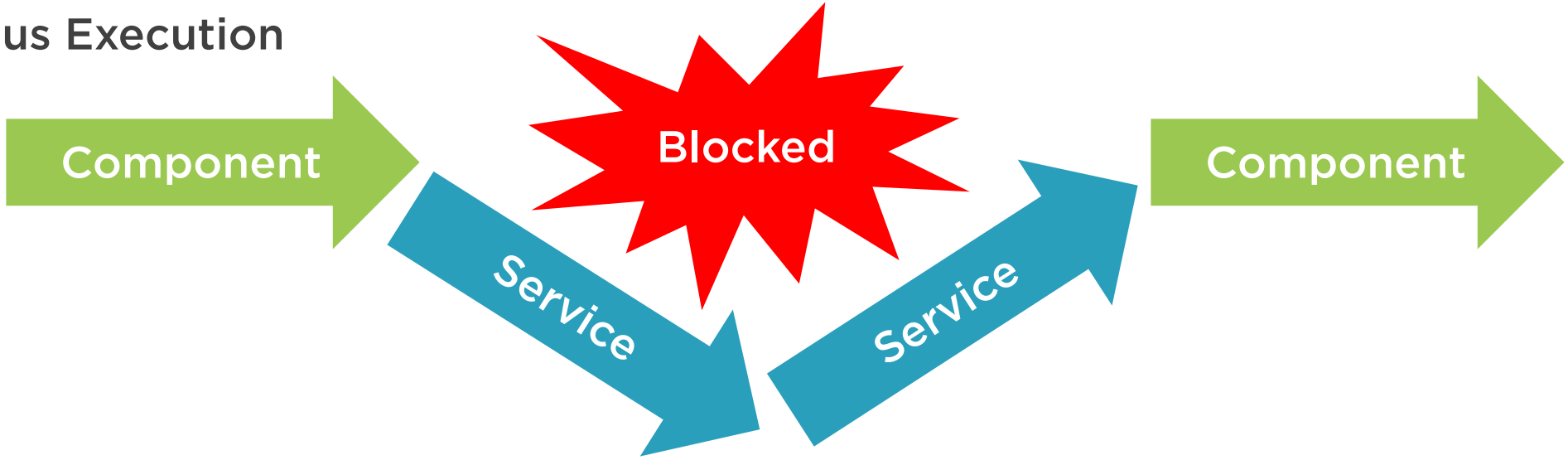
# Services Matter
# Why Asynchronous ~~Code Matters~~

**Synchronous Execution**

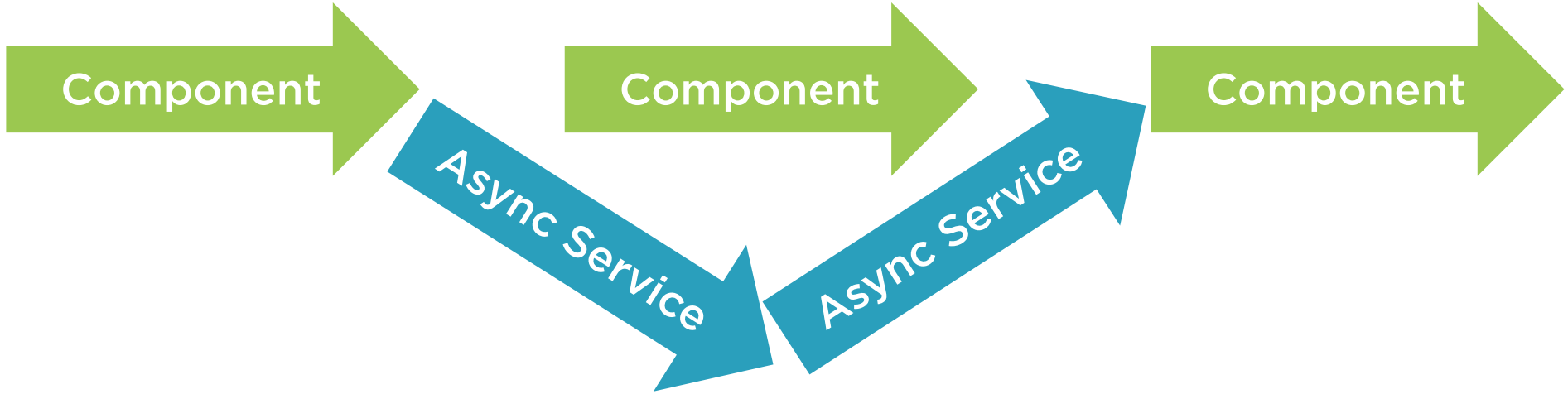


# Services Matter
# Why Asynchronous ~~Code Matters~~

## Synchronous Execution




## Asynchronous Execution



# Observables

# Observables

- Part of RxJS
- Returned by methods on Angular's HttpClient
- Can be used with much more than http requests
- Very large API

# Returning an Observable

```
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs/Rx';

import { Reader } from 'app/models/reader';
```

# Returning an Observable

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Rx';
import { Reader } from 'app/models/reader';


getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
}
```

# Returning an Observable

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Rx';
import { Reader } from 'app/models/reader';


getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
}
```

# Returning an Observable

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Rx';
import { Reader } from 'app/models/reader';

getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
}
```

# Processing an Observable

```
// edit-reader.component.ts
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);




}
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
}
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
}
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log(err),
      () => console.log('All done!')
    );
}
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log(err),
      () => console.log('All done!')
    );
}
```

# Processing an Observable

```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log(err),
      () => console.log('All done!')
    );
}
```

# Demo

## Processing an asynchronous HTTP request with an Observable

# Demo

## Abstracting away HTTP errors

# Promises

# Promises

Part of ES2015 specification

Polyfills shipped with Angular allow them to be used with ES5

General-purpose solution for performing asynchronous work

Results processed with callback functions

Will either be “resolved” or “rejected”

# Returning a Promise

```
updateSchedule(empID: number): Promise<string> {


}
```

# Returning a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```

# Returning a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```

# Returning a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```

# Returning a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {



}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {
```

```
}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
    resolve('Done updating schedule.');
  }



}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
    resolve('Done updating schedule.');
  }


}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
    resolve('Done updating schedule.');
  }
  else {
    reject('Unable to update schedule.');
  }
}
```

# Promise Constructor Parameter

```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
    resolve('Done updating schedule.');
  }
  else {
    reject('Unable to update schedule.');
  }
}
```

# Passing an Arrow Function to a Promise

```
updateSchedule(empID: number): Promise<string> {



}
```

# Passing an Arrow Function to a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise((resolve, reject) => {



}
```

# Passing an Arrow Function to a Promise

```
updateSchedule(empID: number): Promise<string> {
  return new Promise((resolve, reject) => {
    let result: string = this.processCalendar();
    if (result === 'success') {
      resolve('Done updating schedule.');
    }
    else {
      reject('Unable to update schedule.');
    }
  });
}
```

# Processing a Promise

```
ngOnInit() {



}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)



}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    )



}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    )
    .catch(
      err => console.log(`ERROR: ${err}`)

}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    )
    .catch(
      err => console.log(`ERROR: ${err}`)

}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    )
    .catch(
      err => console.log(`ERROR: ${err}`)

}
```

# Processing a Promise

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    )
    .catch(
      err => console.log(`ERROR: ${err}`)

}
```

# Demo

## Asynchronously executing a task with a Promise

# async/await

# async/await

- Work with Promises
- Write code in a more linear style
- Functions declared with the “async” keyword
- Execution will “await” the resolution of a Promise

# Demo

## Handling a Promise with async/await

# Summary

Network calls and long-running tasks should be in services

Asynchronous services are important

Not difficult to implement

Do it for your users!!!