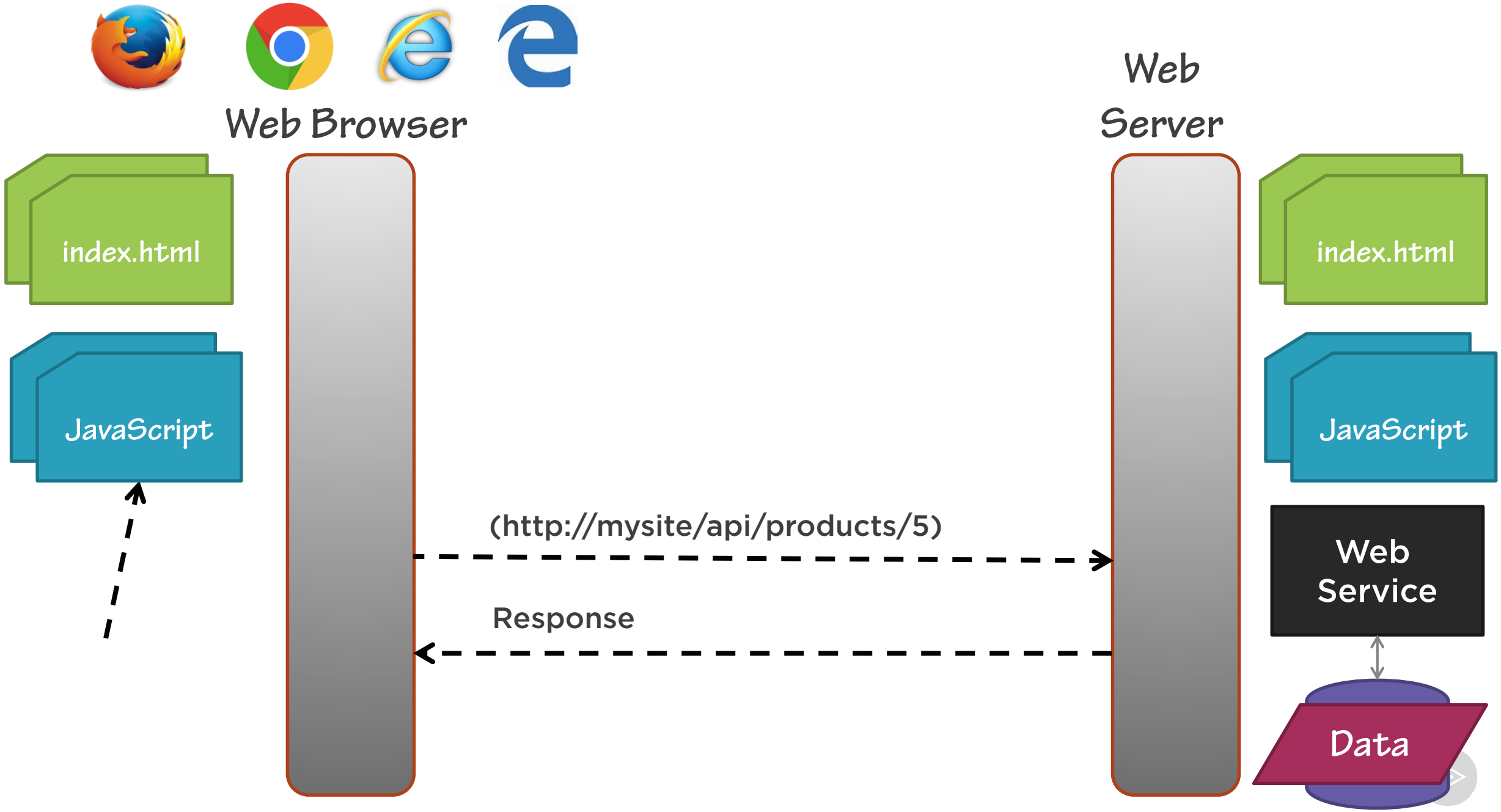
Retrieving Data Using HTTP

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR

@deborahkurata | blogs.msmvps.com/deborahk/


Web Browser
Web Server
index.html
JavaScript
(http://mysite/api/products/5)
Response
index.html
JavaScript
Web Service
Data

Module Overview

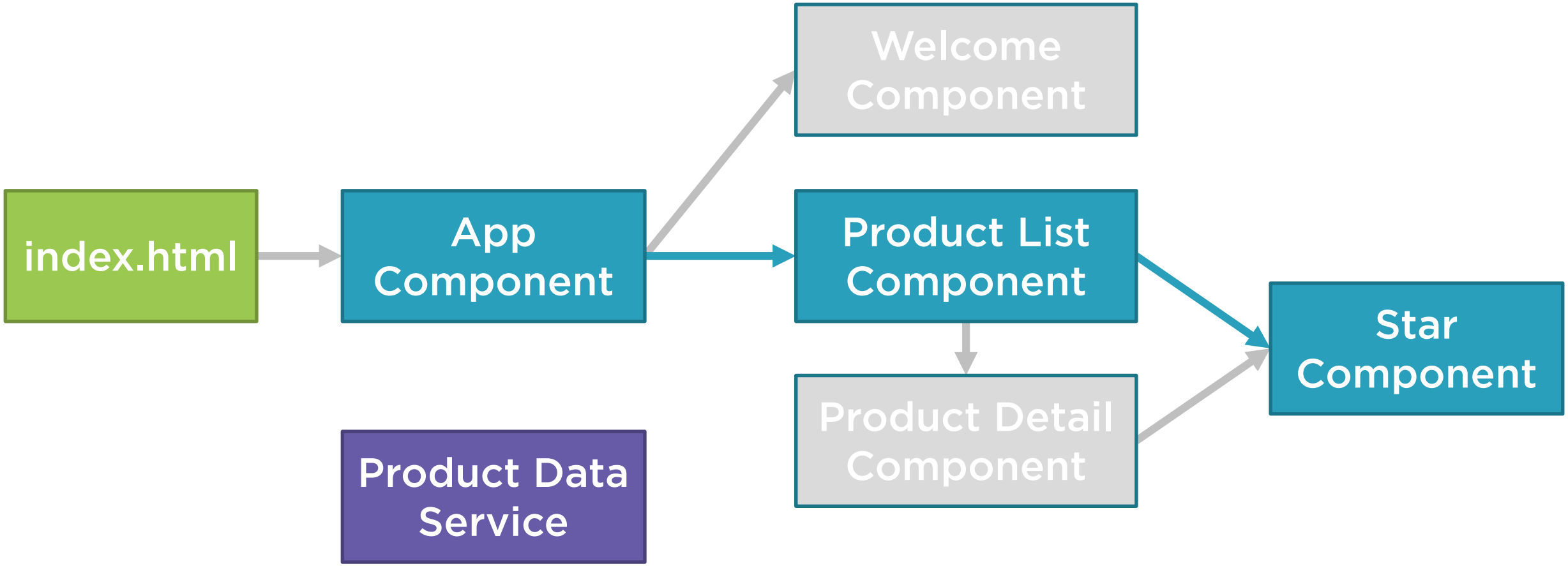
Observables and Reactive Extensions

Sending an Http Request

Exception Handling

Subscribing to an Observable

Application Architecture



Observables and Reactive Extensions

Help manage asynchronous data

Treat events as a collection

- An array whose items arrive asynchronously over time

Are a proposed feature for ES 2016

Use Reactive Extensions (RxJS)

Are used within Angular

Observable Operators

Methods on observables that compose new observables

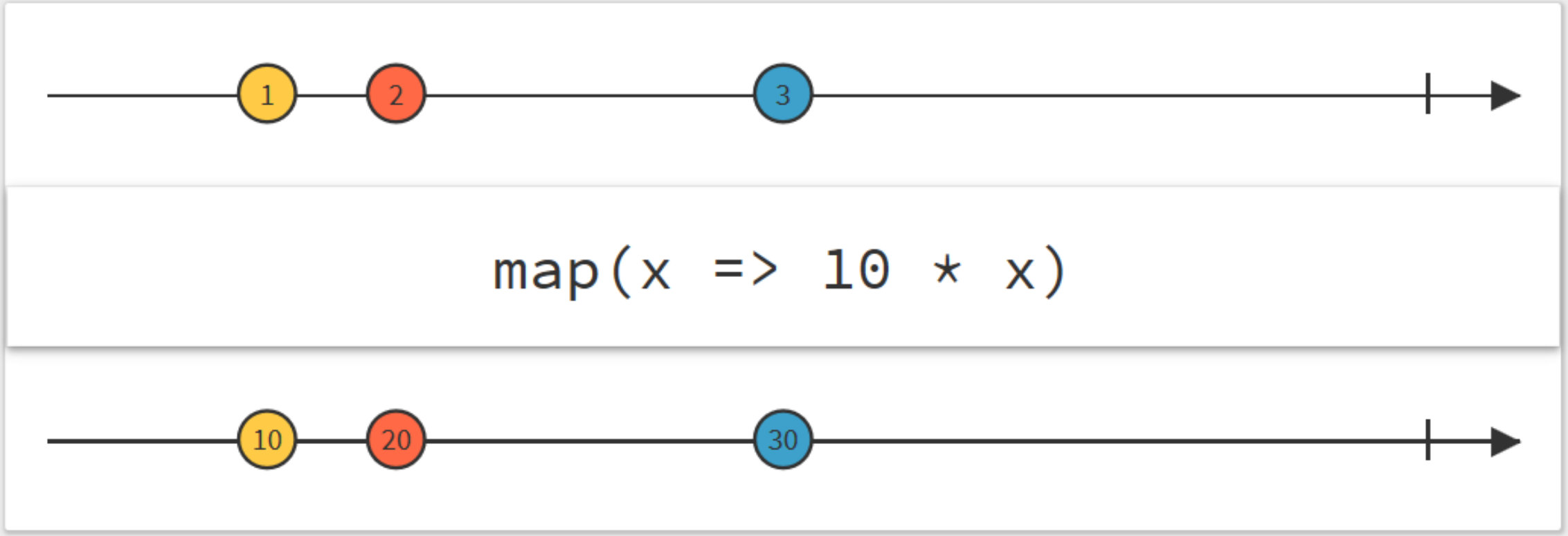
Transform the source observable in some way

Process each value as it is emitted

Examples: map, filter, take, merge, ...

Observables

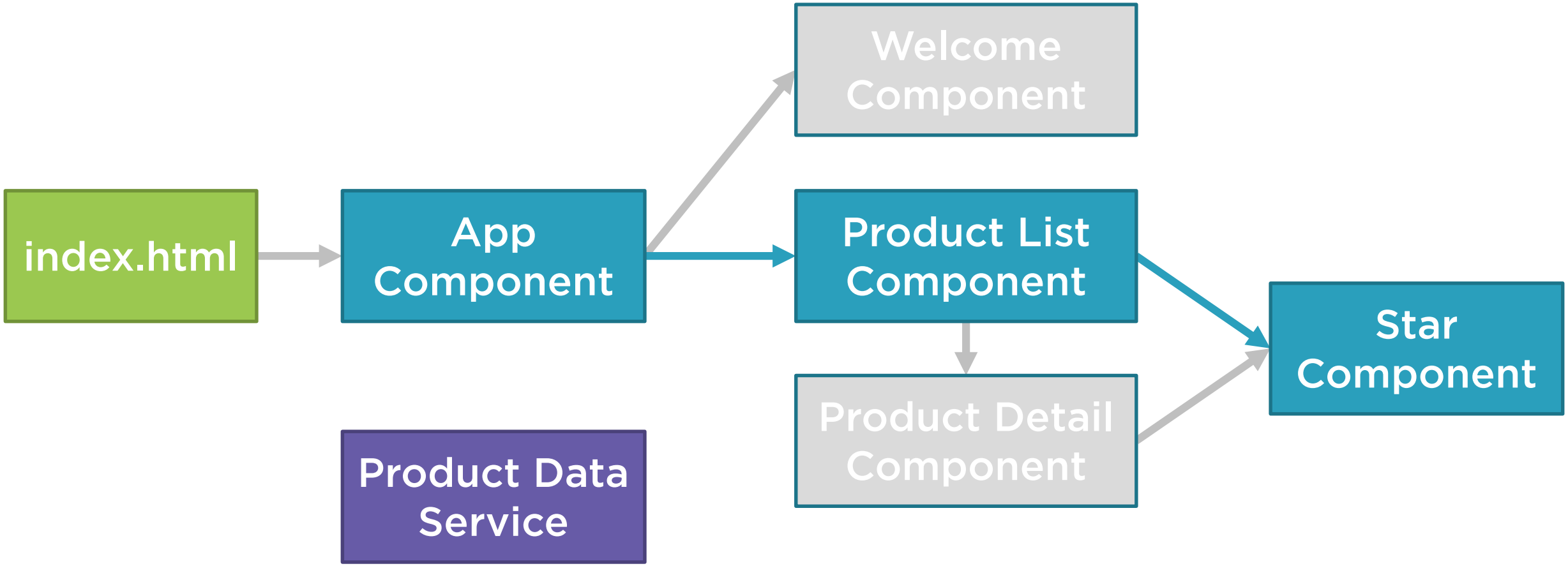
Interactive diagrams of Rx Observables



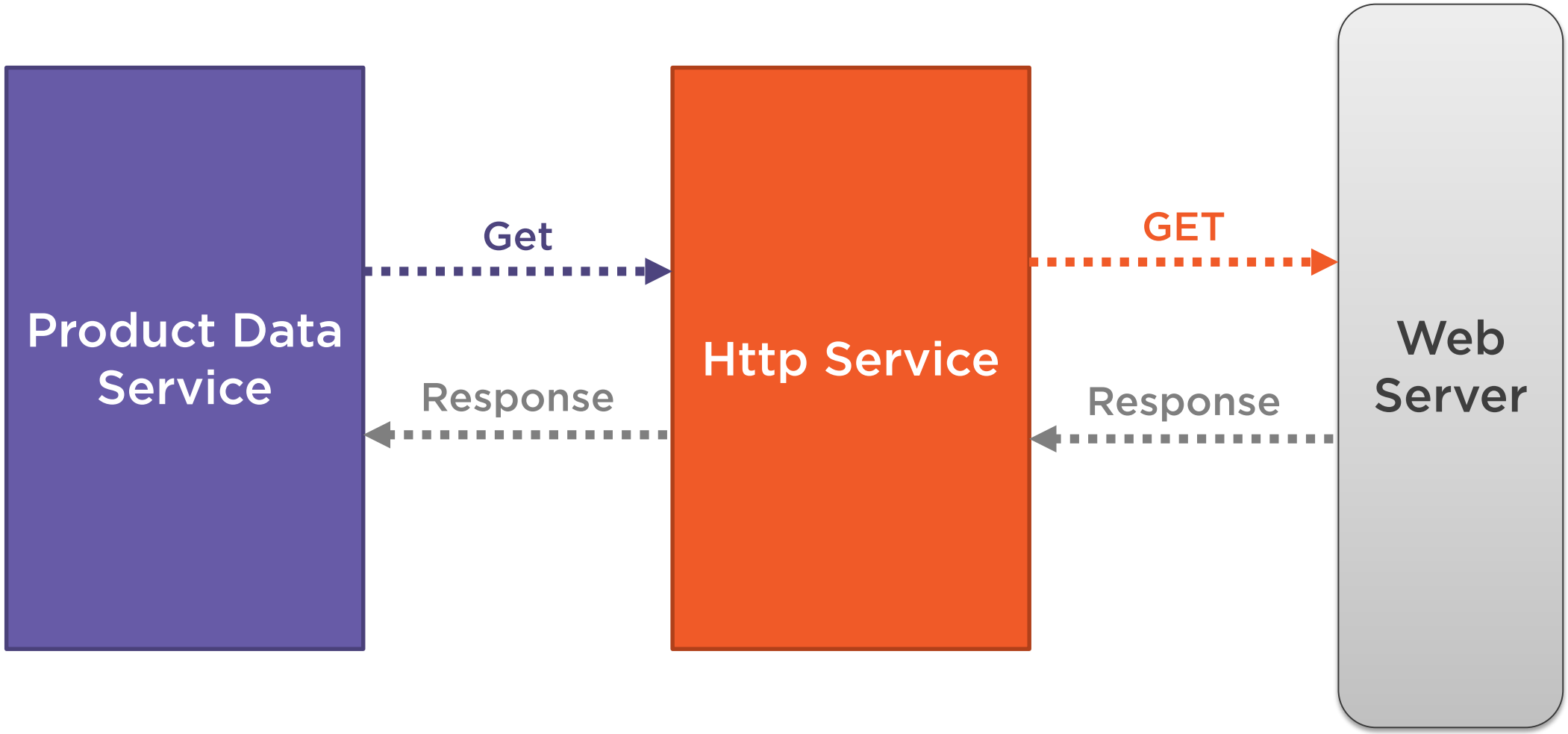
Promise vs Observable

Promise	Observable
Provides a single future value	Emits multiple values over time
Not lazy	Lazy
Not cancellable	Cancellable
	Supports map, filter, reduce and similar operators

Application Architecture



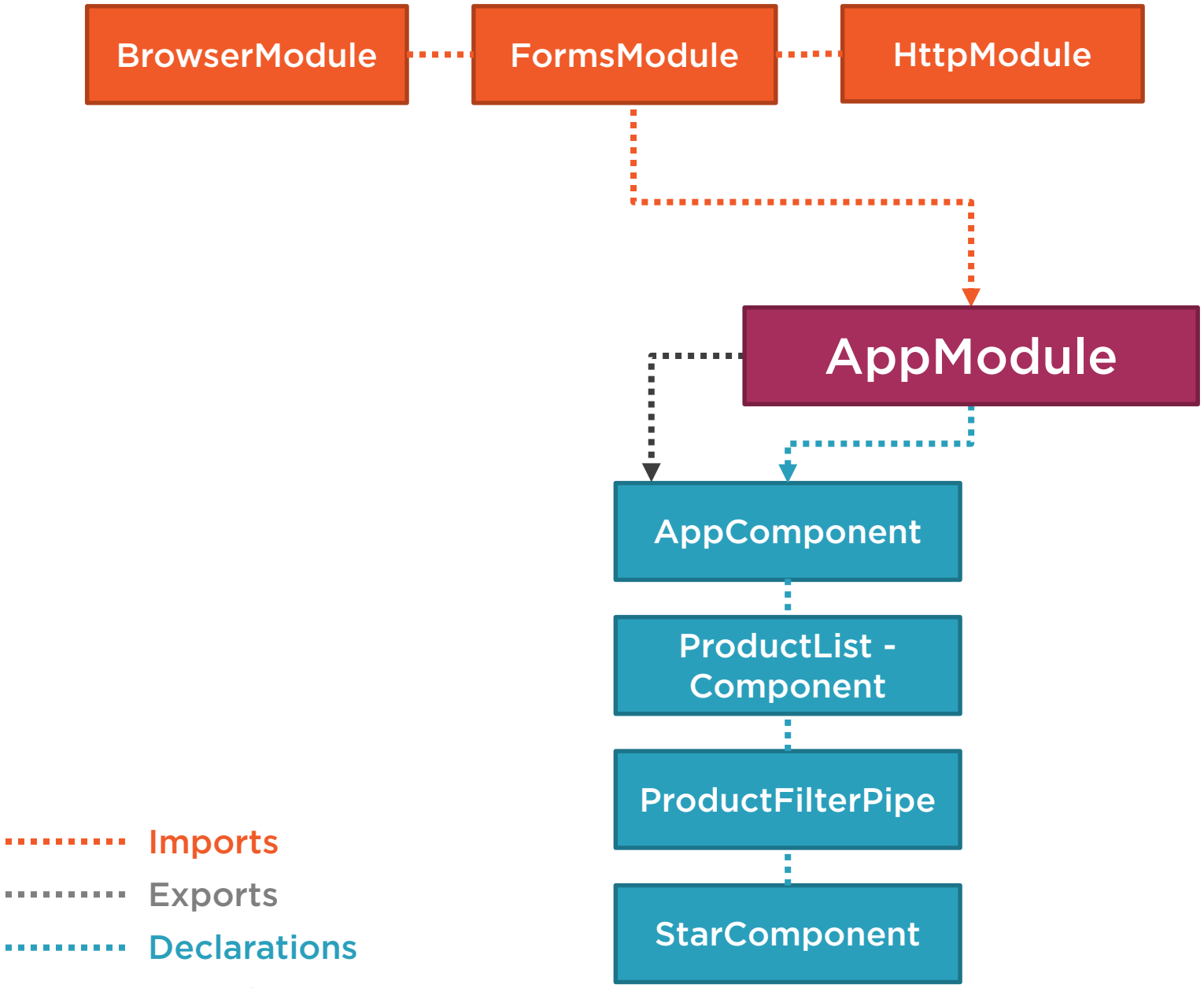
Sending an Http Request



Sending an Http Request

product.service.ts

```
...
import { Http } from '@angular/http';



@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts() {
   return this._http.get(this._productUrl);

  }
}
```


BrowserModule
FormsModule
HttpModule
AppModule
AppComponent
ProductList - Component
ProductFilterPipe
StarComponent
Imports
Exports
Declarations
Providers
Bootstrap

Registering the Http Service Provider

app.module.ts

```
...
import { HttpModule } from '@angular/http';

@NgModule({
  imports: [
      BrowserModule,
      FormsModule,
      HttpModule ],
  declarations: [
      AppComponent,
      ProductListComponent,
      ProductFilterPipe,
      StarComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Sending an Http Request

product.service.ts

```
...
import { Http } from '@angular/http';



@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts() {
   return this._http.get(this._productUrl);

  }
}
```

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';


@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts(): Observable<Response> {
   return this._http.get(this._productUrl);

  }
}
```

Sending an Http Request

product.service.ts

```
...
import { Http, Response} from '@angular/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

@Injectable()
export class ProductService {
  private _productUrl = 'www.myWebService.com/api/products';

  constructor(private _http: Http) { }

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
                .map((response: Response) => <IProduct[]>response.json());
  }
}
```

Exception Handling

product.service.ts

```
...
import 'rxjs/add/operator/do';
import 'rxjs/add/operator/catch';
...

  getProducts(): Observable<IProduct[]> {
   return this._http.get(this._productUrl)
                .map((response: Response) => <IProduct[]>response.json())
                .do(data => console.log('All: ' + JSON.stringify(data)))
                .catch(this.handleError);
  }

  private handleError(error: Response) {
  }
```

Subscribing to an Observable

```
x.then(valueFn, errorFn)                        //Promise
x.subscribe(valueFn, errorFn)                   //Observable
x.subscribe(valueFn, errorFn, completeFn)       //Observable
let sub = x.subscribe(valueFn, errorFn, completeFn)
```

product-list.component.ts

```
ngOnInit(): void {
    this._productService.getProducts()
            .subscribe(products => this.products = products,
                       error => this.errorMessage = <any>error);
}
```

Http Checklist: Setup

Add HttpModule to the imports array of one of the application's Angular Modules

Http Checklist: Service

Import what we need

Define a dependency for the http client service

- Use a constructor parameter

Create a method for each http request

Call the desired http method, such as get

- Pass in the Url

Map the Http response to a JSON object

Add error handling

Http Checklist: Subscribing

Call the subscribe method of the returned observable

Provide a function to handle an emitted item

- Normally assigns a property to the returned JSON object

Provide an error function to handle any returned errors

Summary

- Observables and Reactive Extensions
- Sending an Http Request
- Exception Handling
- Subscribing to an Observable

Application Architecture