



# Improving Our Components

- Strong typing & interfaces
- Encapsulating styles
- Lifecycle hooks
- Custom pipes
- Nested components



# Module Overview



Defining an Interface

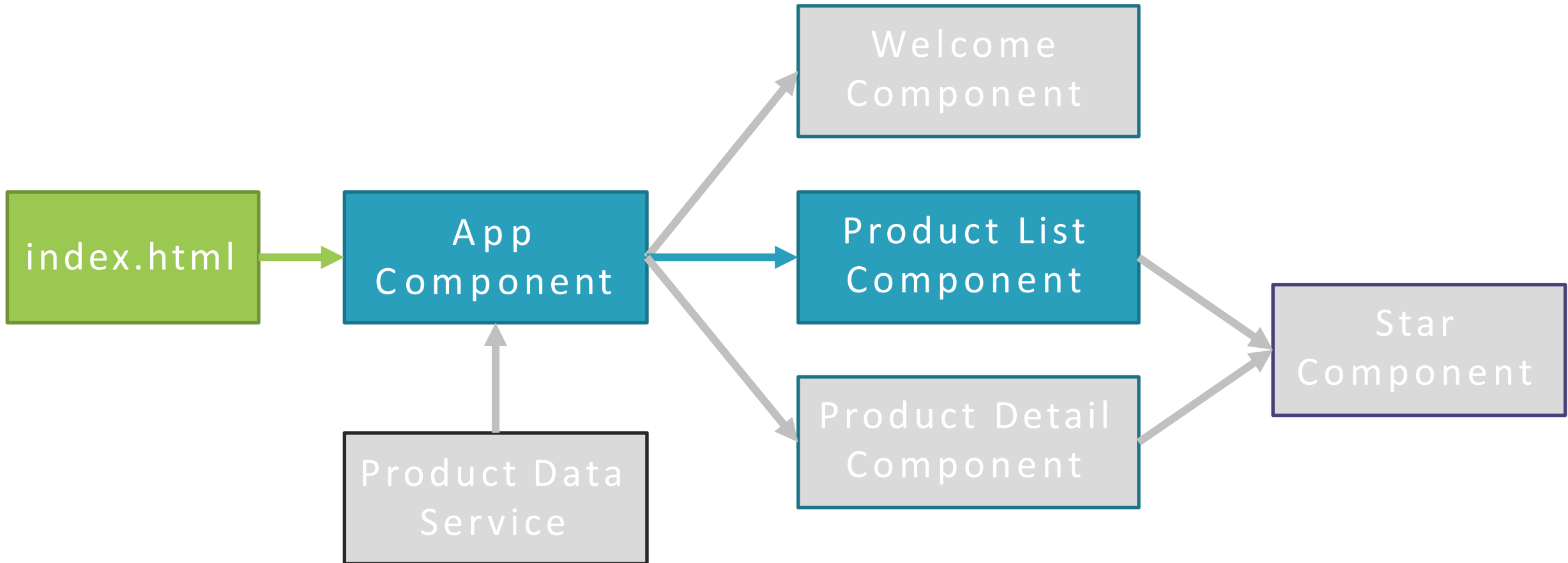
Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe



# Application Architecture



# Strong Typing

```
export class ProductListComponent {  
    pageTitle: string = 'Product List';  
    showImage: boolean = false;  
    listFilter: string = 'cart';  
    message: string;  
  
    products: any[] = [...];  
  
    toggleImage(): void {  
        this.showImage = !this.showImage;  
    }  
  
    onRatingClicked(message: string): void {  
        this.message = message;  
    }  
}
```



# Interface

A **specification** identifying a related set of properties and methods.

A class commits to supporting the specification by **implementing** the interface.

Use the interface as a **data type**.

Development time only!



# Interface Is a Specification

```
export interface IProduct {  
  productId: number;  
  productName: string;  
  productCode: string;  
  releaseDate: Date;  
  price: number;  
  description: string;  
  starRating: number;  
  imageUrl: string;  
  calculateDiscount(percent: number): number;  
}
```

export  
keyword

Interface  
Name

interface  
keyword



# Using an Interface as a Data Type

```
import { IProduct } from './product';

export class ProductListComponent {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';

  products: IProduct[] = [...];

  toggleImage(): void {
    this.showImage = !this.showImage;
  }
}
```





# Handling Unique Component Styles



Templates sometimes require unique styles

We can inline the styles directly into the HTML

We can build an external stylesheet and link it in index.html

There is a better way!



# Encapsulating Component Styles

## styles

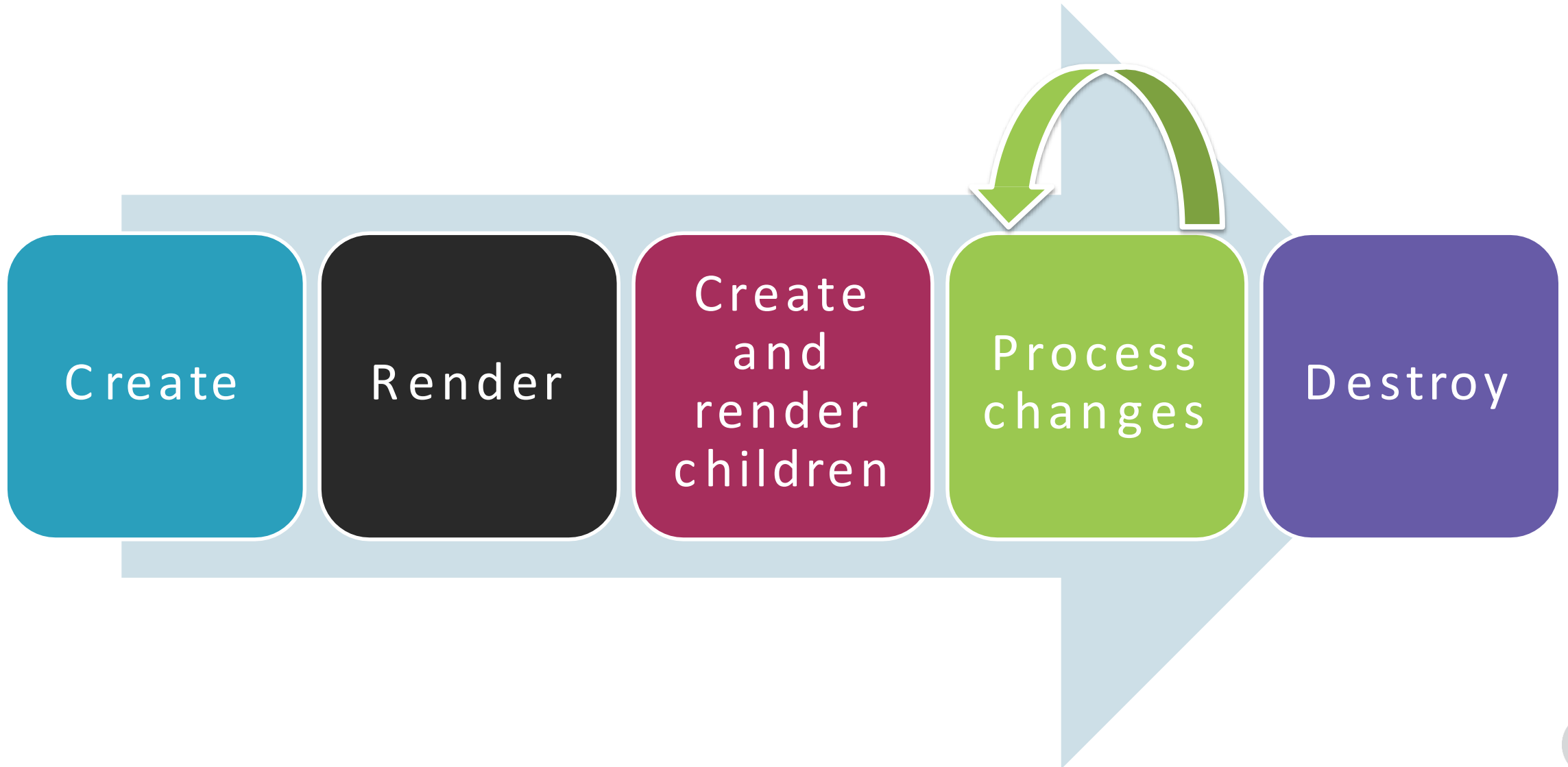
```
@Component({  
  selector: 'pm-products',  
  templateUrl: 'app/products/product-list.component.html',  
  styles: ['thead {color: #337AB7;}']})
```

## styleUrls

```
@Component({  
  selector: 'pm-products',  
  templateUrl: 'app/products/product-list.component.html',  
  styleUrls: ['app/products/product-list.component.css']})
```



# Component Lifecycle



# Component Lifecycle Hooks



**OnInit:** Perform component initialization, retrieve data

**OnChanges:** Perform action after change to input properties

**OnDestroy:** Perform cleanup

# Using a Lifecycle Hook

2

1

```
export class ProductListComponent
    implements OnInit {
  pageTitle: string = 'Product List';
  showImage: boolean = false;
  listFilter: string = 'cart';
  products: IProduct[] = [...];
```

3

```
}
```



# Transforming Data with Pipes

Transform  
bound  
properties  
before  
display

## Built-in pipes

- date
- number, decimal, percent, currency
- json, slice
- etc

Custom  
pipes



# Building a Custom Pipe

```
import { Pipe, PipeTransform } from 'angular2/core';

@Pipe({
  name: 'productFilter'
})
export class ProductFilterPipe
    implements PipeTransform {

  transform(value: IProduct[],
    args: string[]): IProduct[] {
  }
}
```



# Using a Custom Pipe

## Template

```
<tr *ngFor = '#product of products | productFilter: listFilter'>
```

## Component

```
import { ProductFilterPipe } from './product-filter.pipe';
@Component({
  selector: 'pm-products',
  templateUrl: 'app/products/product-list.component.html',
  pipes: [ProductFilterPipe]
})
export class ProductListComponent { }
```





# Checklist: Interfaces



Defines custom types

Creating interfaces:

- `interface` keyword
- `export` it

Implementing interfaces:

- `implements` keyword & interface name
- Write code for each property & method

# Checklist: Encapsulating Styles



\_\_\_\_\_



\_\_\_\_\_



\_\_\_\_\_

`styles` property

- Specify an array of style strings

`styleUrls` property

- Specify an array of stylesheet paths

# Checklist: Using Lifecycle Hooks



\_\_\_\_\_



\_\_\_\_\_



\_\_\_\_\_

Import the lifecycle hook interface

Implement the lifecycle hook interface

Write code for the hook method



# Checklist: Building a Custom Pipe



\_\_\_\_\_



\_\_\_\_\_



\_\_\_\_\_

Import Pipe and PipeTransform

Create a class that implements  
PipeTransform

- `export` the class

Write code for the Transform method

Decorate the class with the Pipe decorator



# Checklist: Using a Custom Pipe



Use the Pipe in the template

- Pipe character
- Pipe name
- Pipe arguments (separated with colons)

Import the custom pipe

Add the `pipes` property to the component's metadata



# Summary



Defining an Interface

Encapsulating Component Styles

Using Lifecycle Hooks

Building a Custom Pipe



# Application Architecture

