Overview

Data Binding

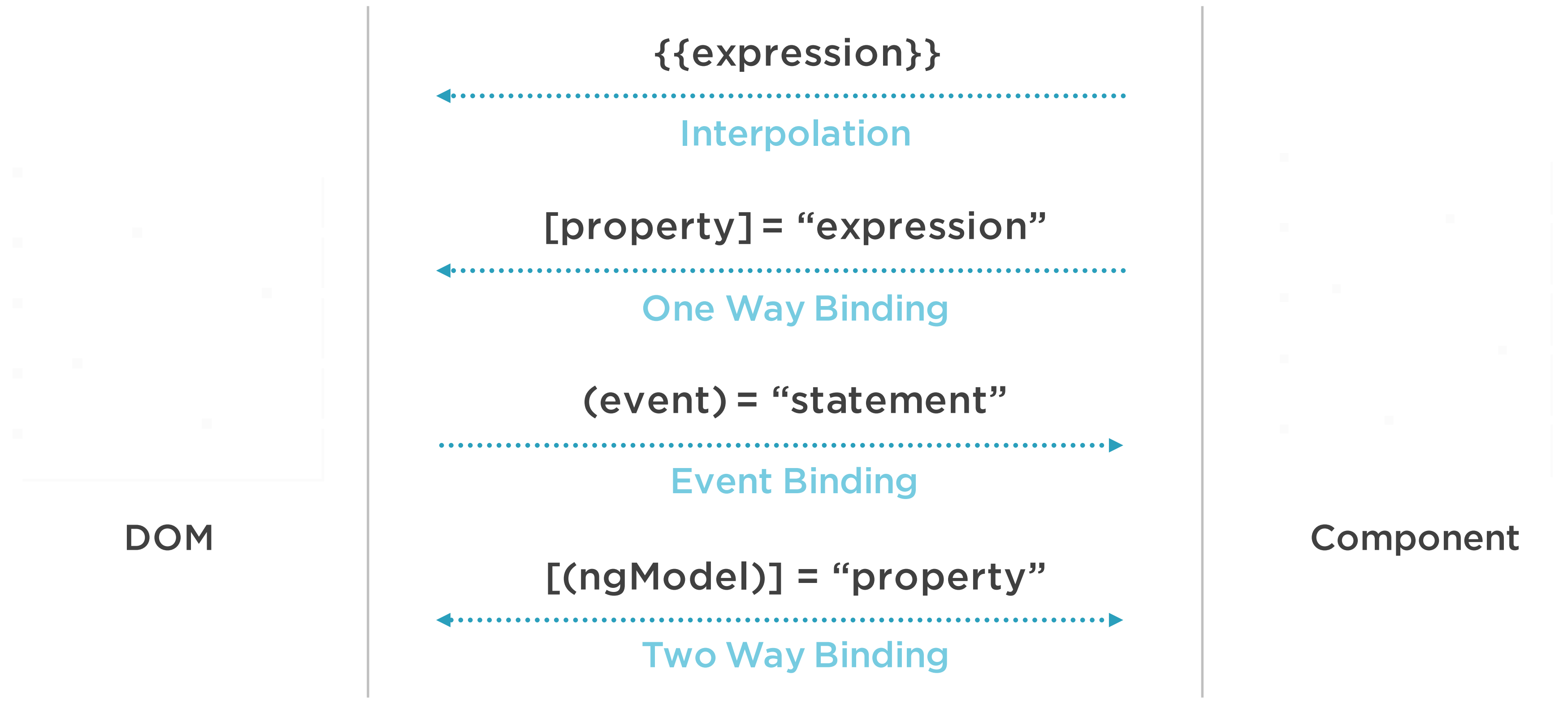
Built-in Directives

Pipes

Data Binding

Data Binding

We use data binding to help coordinate communication between a Component and its Template.


DOM
{{expression}}
Interpolation
[property] = "expression"
One Way Binding
(event) = "statement"
Event Binding
[(ngModel)] = "property"
Two Way Binding
Component

Angular 2’s change detection is based on unidirectional data flow

Benefits of Angular 2’s Unidirectional Data Flow

Easier widget integration

No more $apply

No more repeated digest cycles

No more watchers

No more performance issues with digest cycle and watcher limits

Interpolation

Using the {{ }} to render the bound value to the Component's Template

One Way In

```html
<h3>Vehicle: {{vehicle.name}}</h3>
<div>
  <img src="{{vehicle.imageUrl}}">
  <a href="{{vehicle.wikiLink}}">Wiki</a>
</div>
```

Interpolation

Evaluate an expression between double curly braces

{{ expression }}

Demo

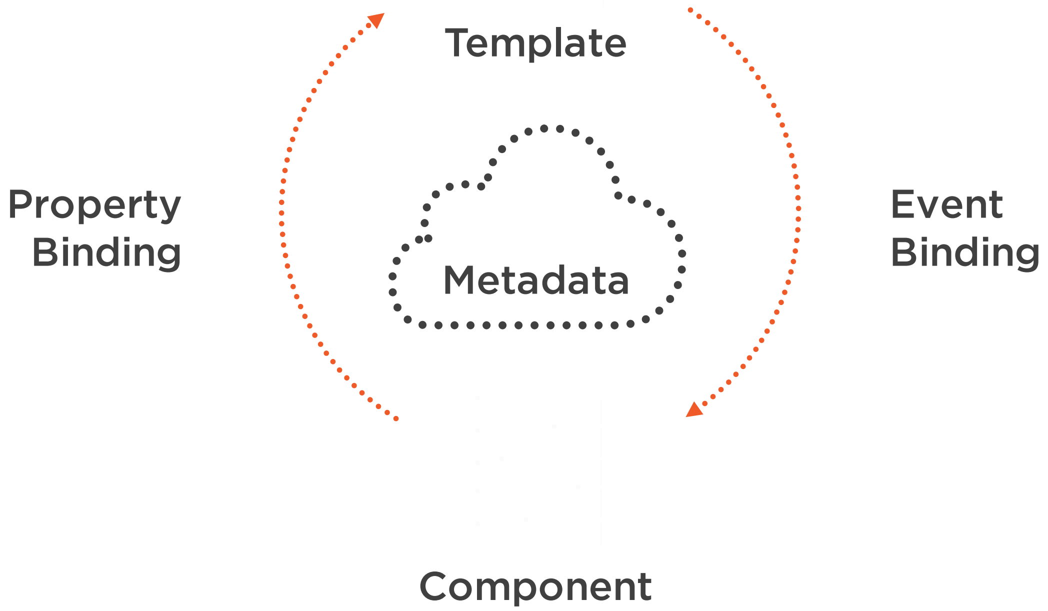
Interpolation

{{ }}

Property Binding

Using the [ ] to send values from the Component to the Template

Data Binding Communication



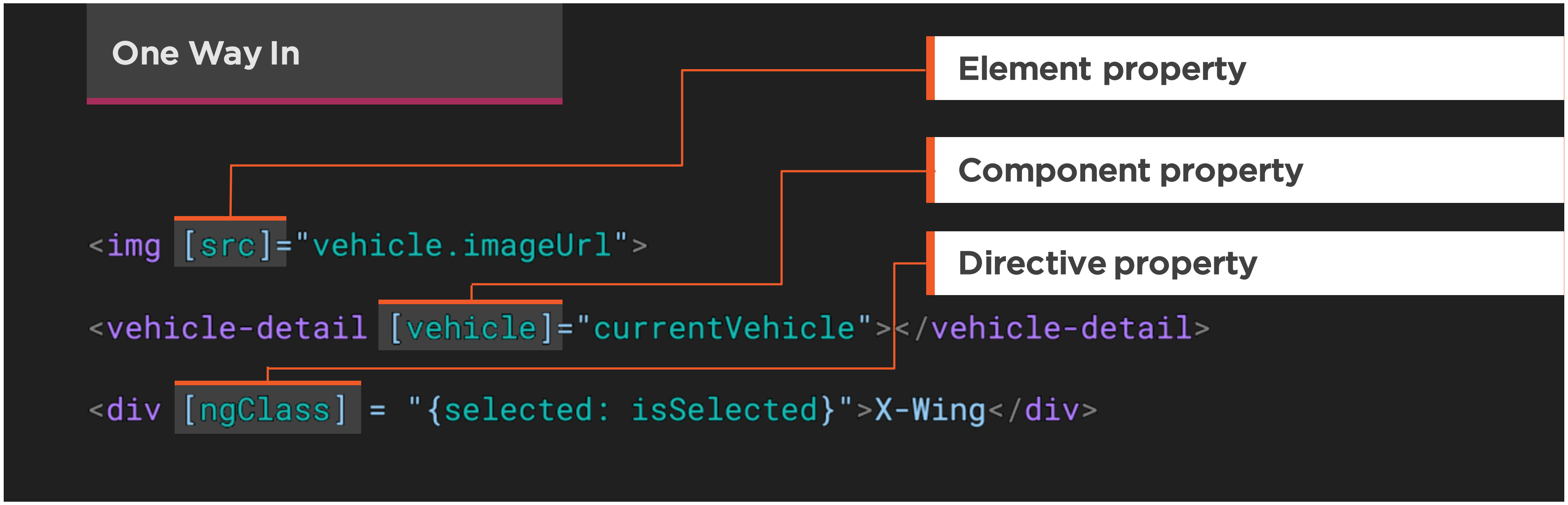
We set properties and events of DOM elements, not attributes

One Way

Binding target property

```
{{expression}}
[target] = "expression"
bind-target = "expression"
```

Data source to view target

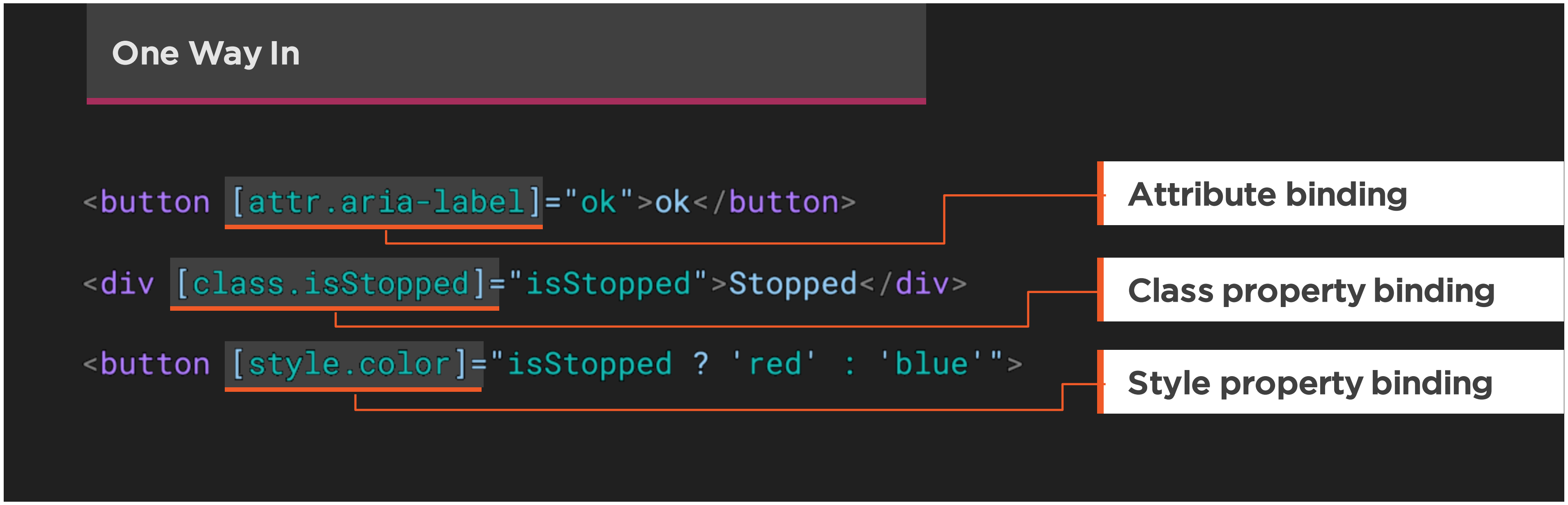


```
<img [src]="vehicle.imageUrl">

<vehicle-detail [vehicle]="currentVehicle"></vehicle-detail>

<div [ngClass] = "{selected: isSelected}">X-Wing</div>
```

Property Binding

[property]="expression"

Bind to element, Component or a directive property



One Way In

```
<button [attr.aria-label]="ok">ok</button>
<div [class.isStopped]="isStopped">Stopped</div>
<button [style.color]="isStopped ? 'red' : 'blue'">
```

Property Binding

For attributes use attr

Use dots for nested properties

Demo

Property Binding

Event Binding

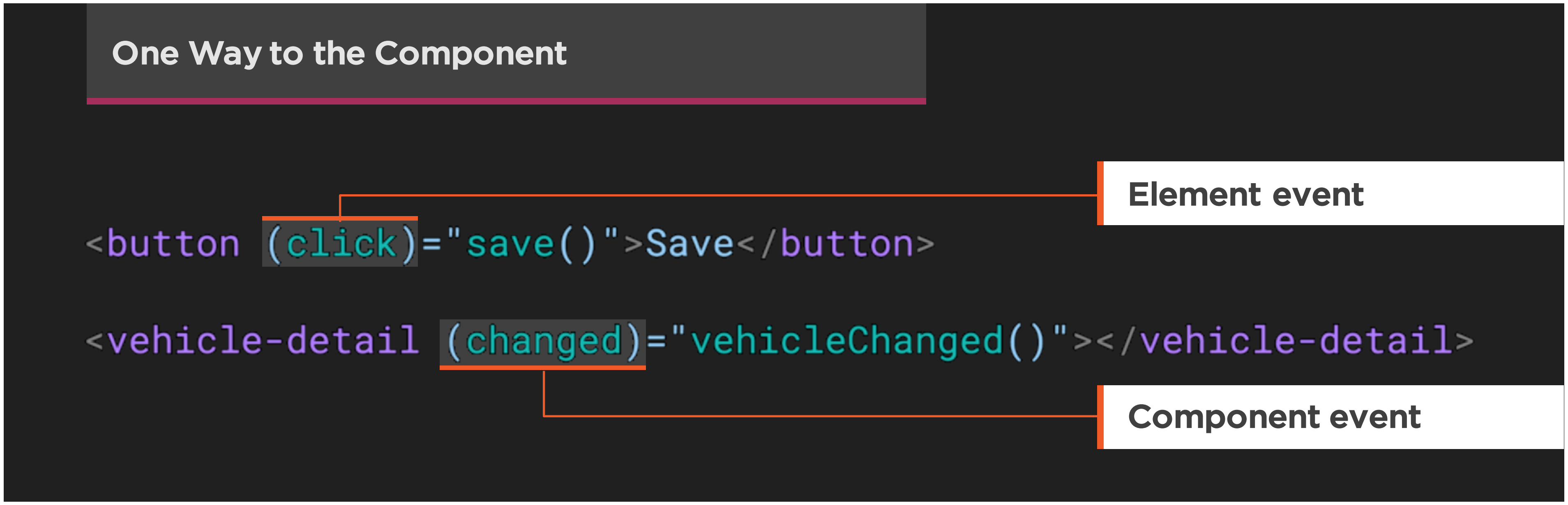
Using the ( ) to send events from the Template to the Component

One Way

Binding target event

(target) = "statement"
on-target = "statement"

View target to data source

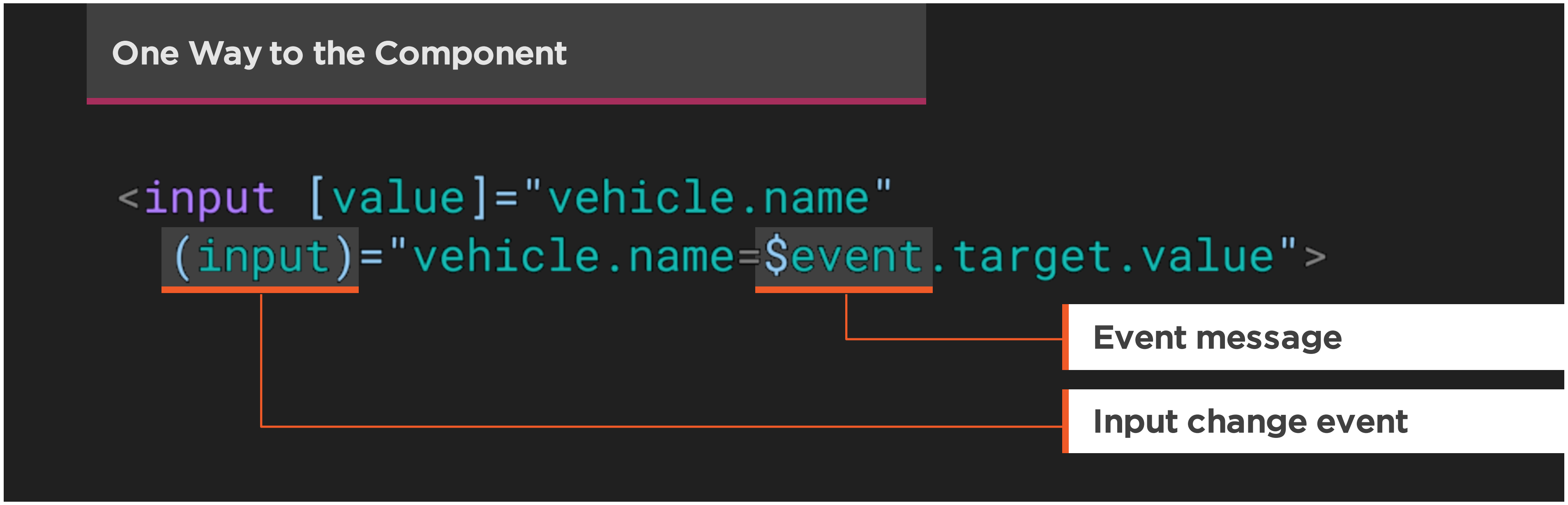


```
<button (click)="save()">Save</button>

<vehicle-detail (changed)="vehicleChanged()"></vehicle-detail>
```

Event Binding

Execute an expression when an event occurs

(event-target)="statement"

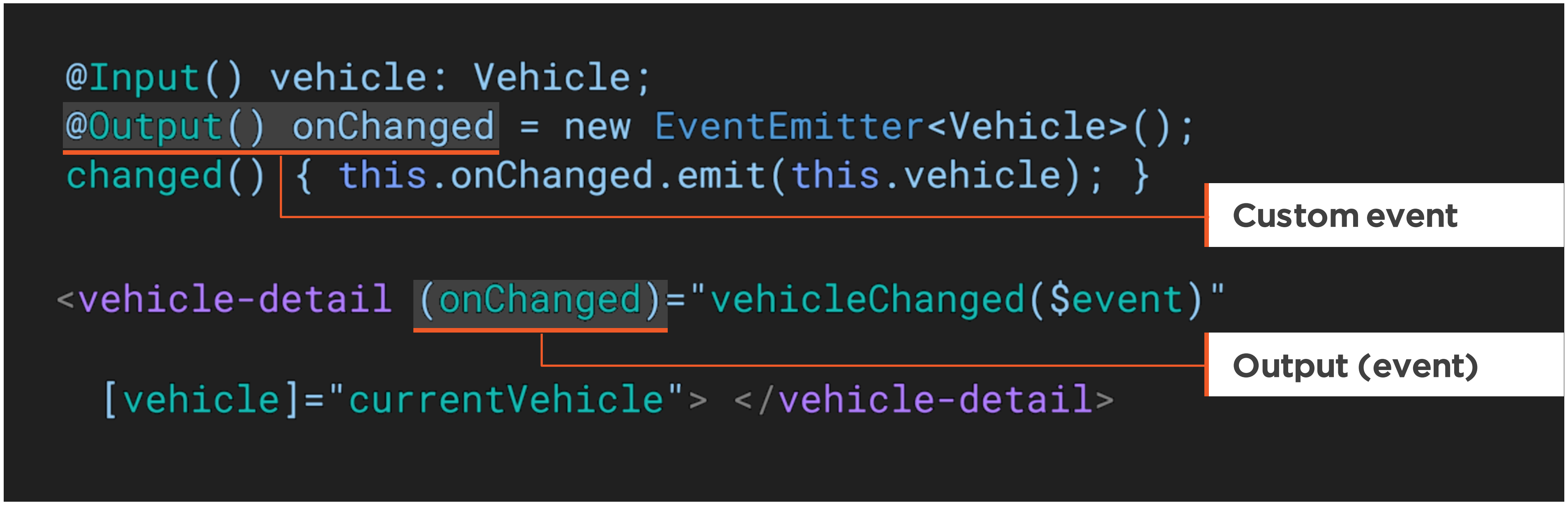


One Way to the Component

```
<input [value]="vehicle.name"
  (input)="vehicle.name=$event.target.value">
```

$event

Contains a message about the event



```
@Input() vehicle: Vehicle;
@Output() onChanged = new EventEmitter<Vehicle>();
changed() { this.onChanged.emit(this.vehicle); }

<vehicle-detail (onChanged)="vehicleChanged($event)"
  [vehicle]="currentVehicle"> </vehicle-detail>
```

Custom Events

EventEmitter defines a new event

Fire its emit method to raise event with data

Bind to the event on the Component's Template

Demo

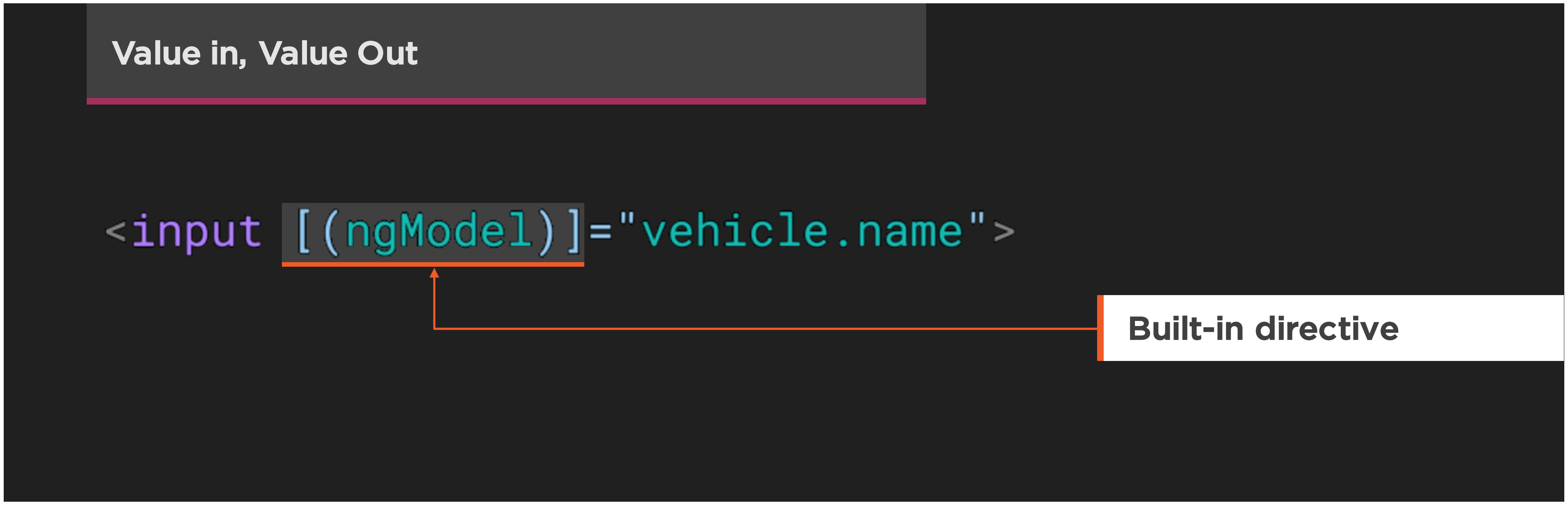
Event Binding

Two Way Binding

[( )] sends a value from Component to Template, and sends value changes in the Template to the Component

Two Way

```
[(ngModel)] = "expression"
bindon-ngModel="expression"
```



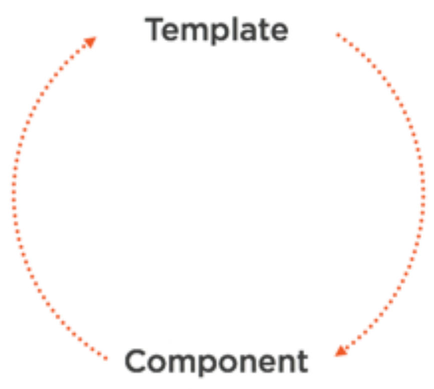
```
<input [(ngModel)]="vehicle.name">
```

Two Way Binding

[( )] = Banana in a box

Demo

Data Binding



Built-in Directives

Directives

When Angular renders templates, it transforms the DOM according to instructions from Directives

Angular Class and Style Directives

Angular 1 🟦

Angular 2 🟥

ng-class

ngClass

ng-class="{active: isActive, color: myColor}"

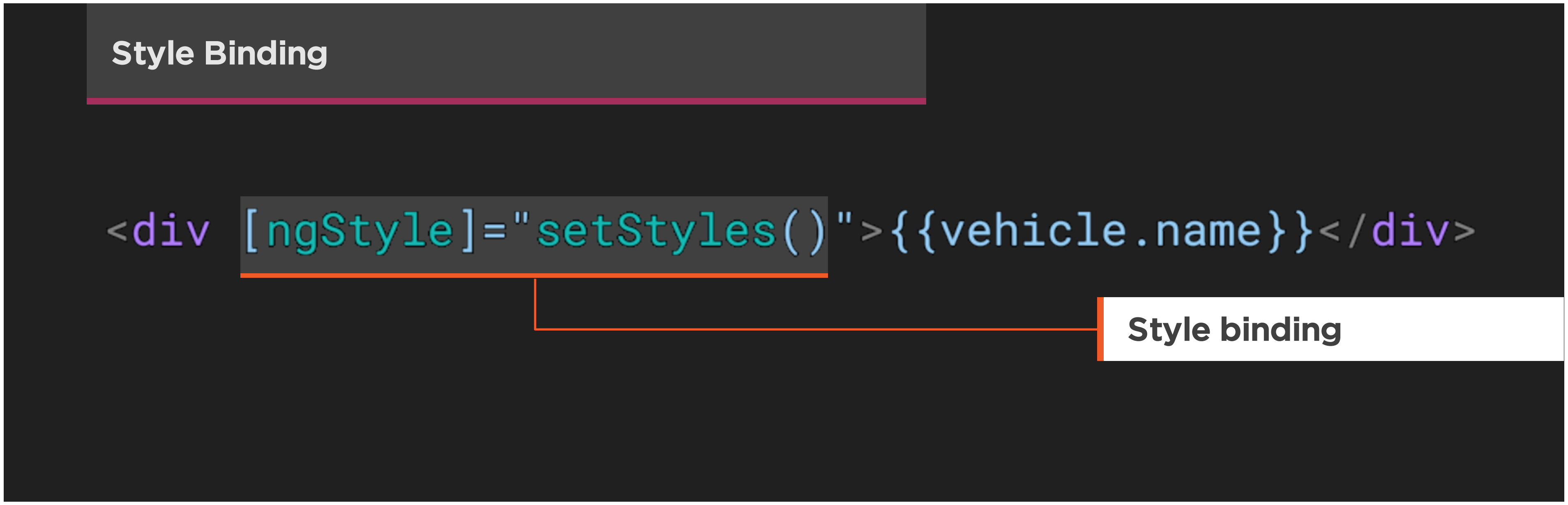
[ngClass]="{active: isActive, color: myColor}"

ng-style

ngStyle

ng-style="{color: colorPreference}

[ngStyle]="{color: colorPreference}"
[style.color]="colorPreference"

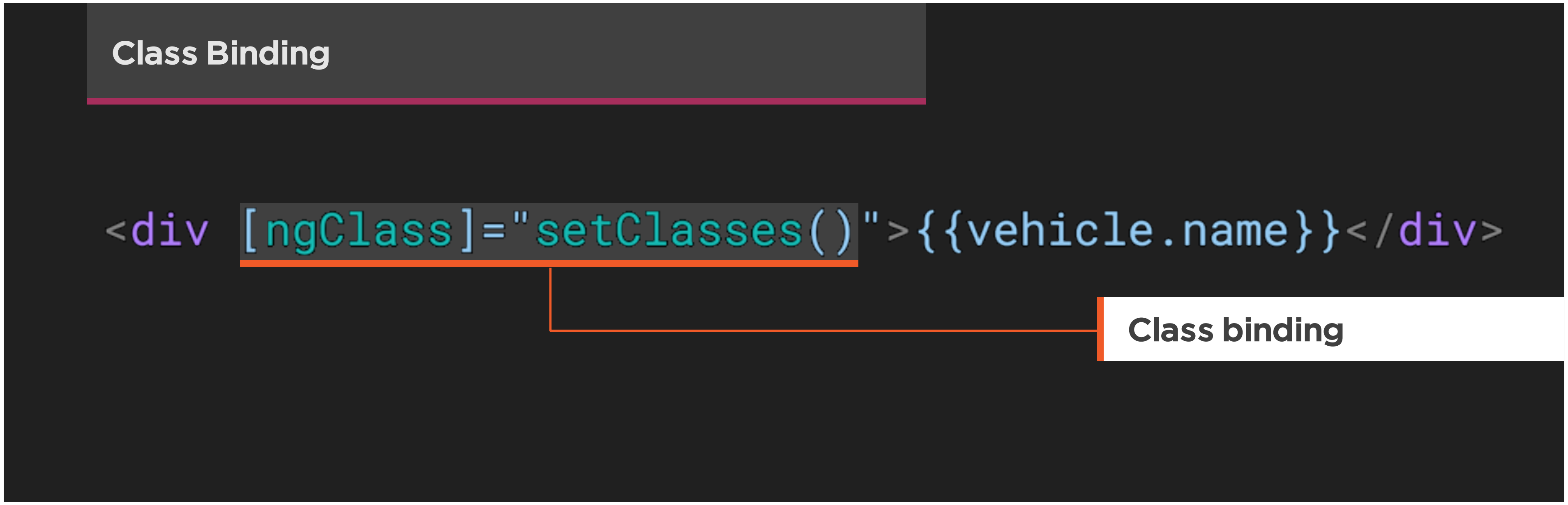


```
<div [ngStyle]="setStyles()">{{vehicle.name}}</div>
```

ngStyle

Alternative to [style.style-name]

Setting multiple styles



Class Binding

```
<div [ngClass]="setClasses()">{{vehicle.name}}</div>
```

ngClass

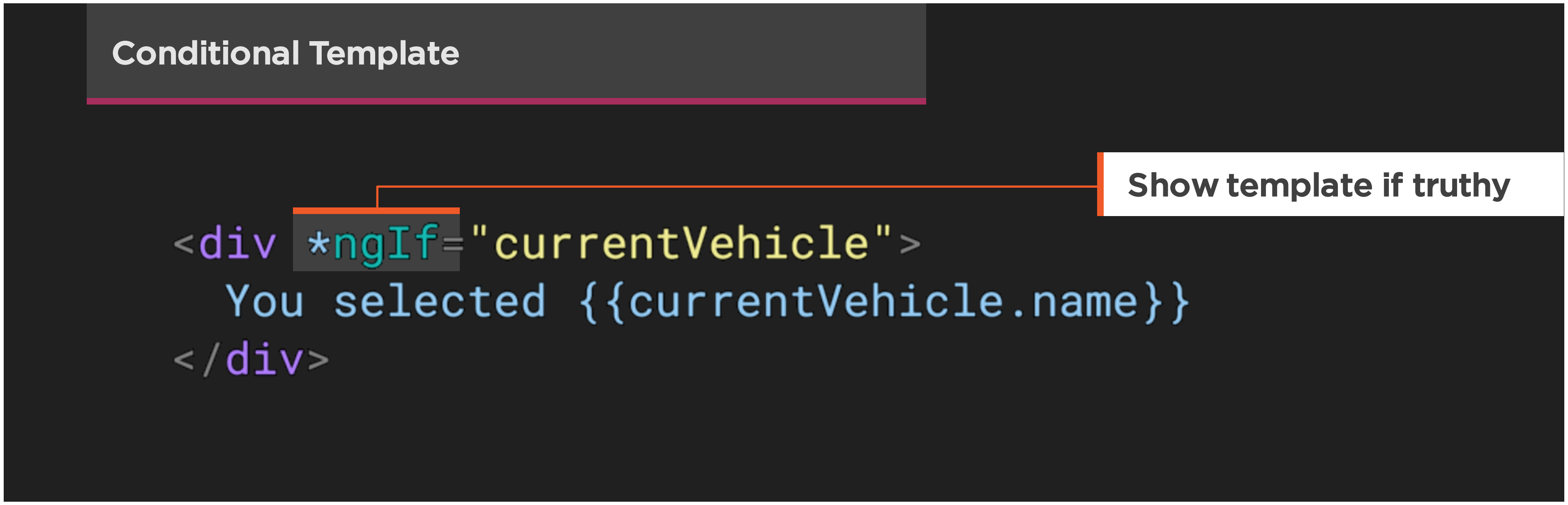
Alternative to [class.class-name]

Setting multiple classes

Angular Structural Directives

Angular 1	🟦
Angular 2	🟥

🟦	ng-repeat
🟥	*ngFor
🟦	ng-if
🟥	*ngIf
🟦	ng-switch
🟥	*ngSwitch



Conditional Template

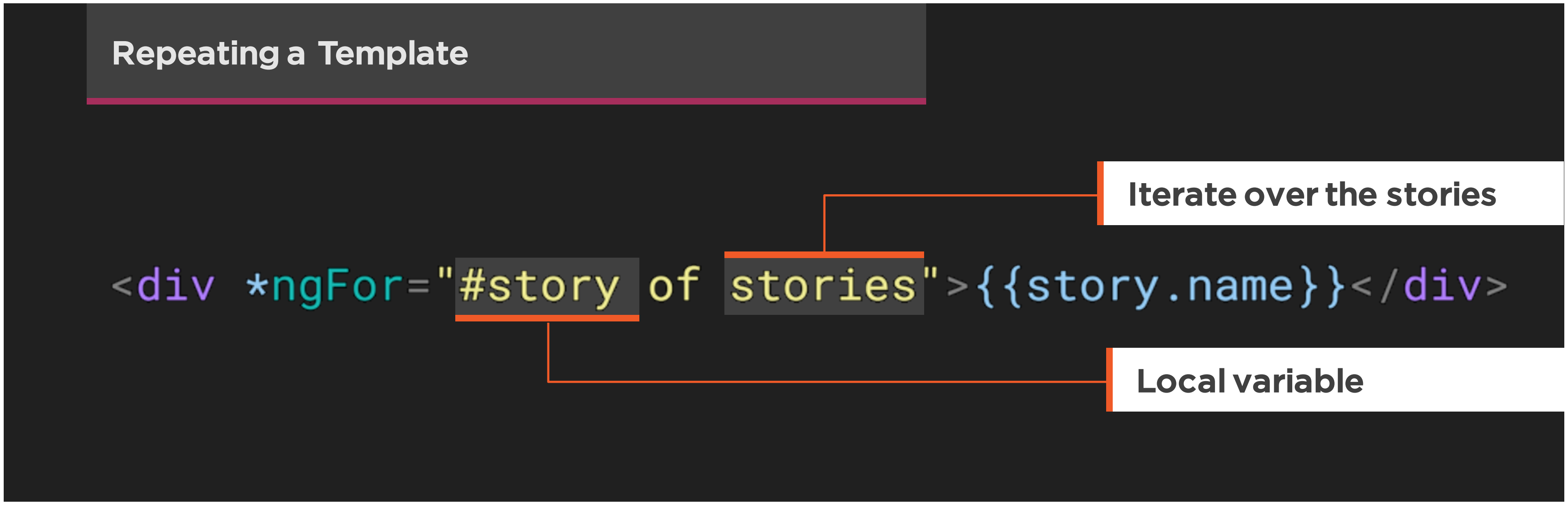
```
<div *ngIf="currentVehicle">
  You selected {{currentVehicle.name}}
</div>
```

*ngIf

Conditionally removes elements from the DOM

Structural directive

Use [style.visibility]="isVisible()" to hide


Repeating a Template

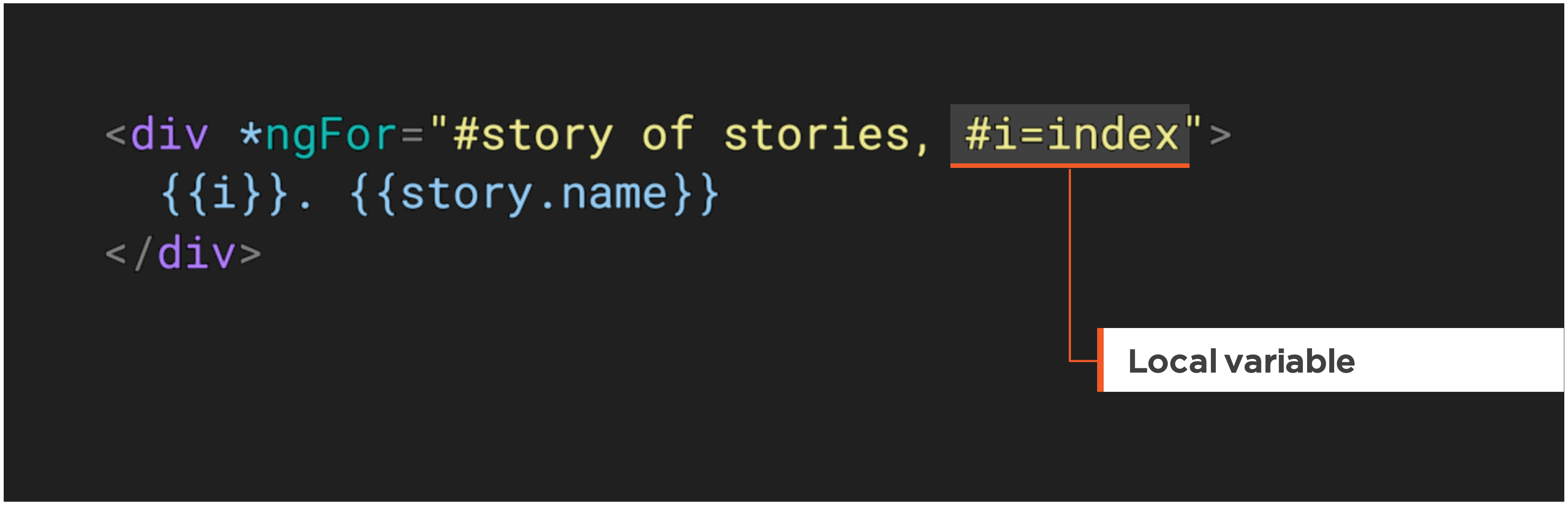
```
<div *ngFor="#story of stories">{{story.name}}</div>
```

*ngFor

Structural directive

Show an element n number of times

# declares a local variable



```
<div *ngFor="#story of stories, #i=index">
  {{i}}. {{story.name}}
</div>
```

Local Variables

# declares a local variable

Can also use var i = index

Demo

Directives

Pipes

Pipes

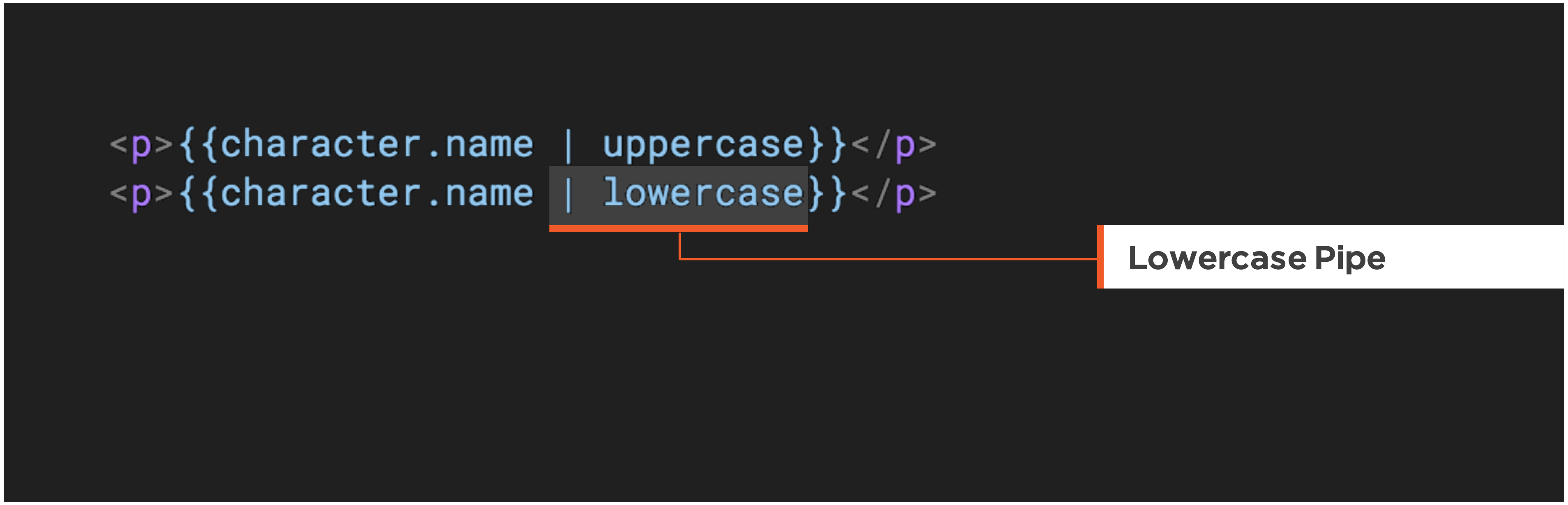
Pipes allow us to transform data for display in a Template.

Angular Formatters

Angular 1	🟦
Angular 2	🟥

filters

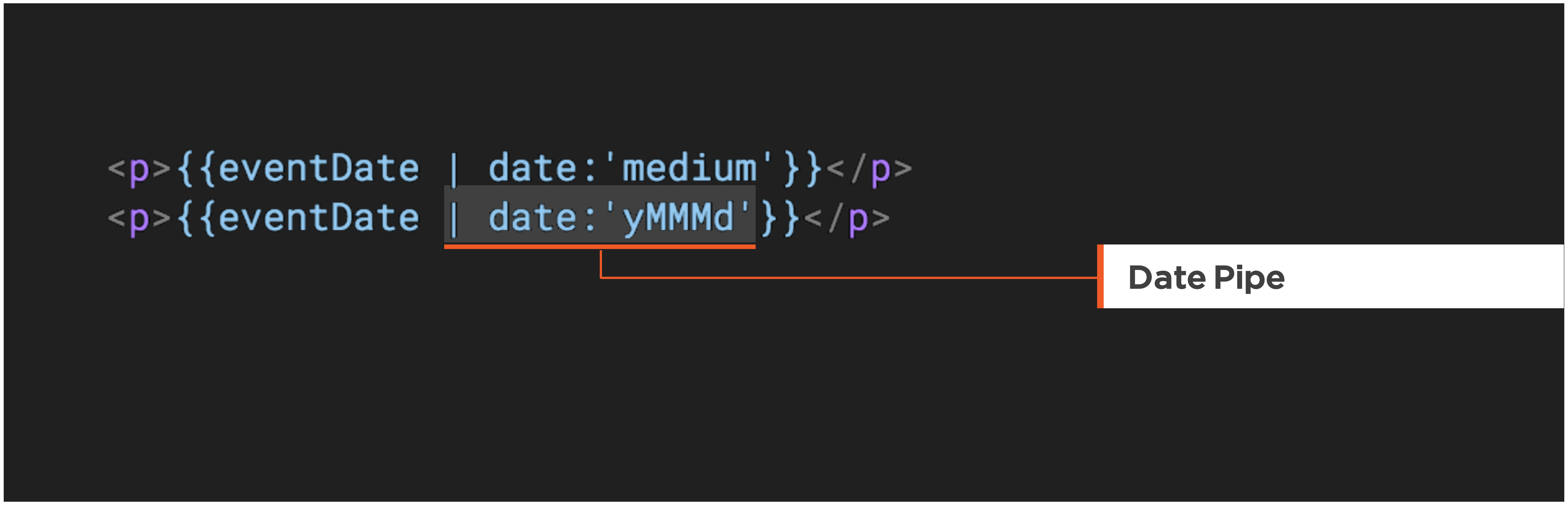
pipes



```
<p>{{character.name | uppercase}}</p>
<p>{{character.name | lowercase}}</p>
```

Built-in Pipes

Format a value in a Template



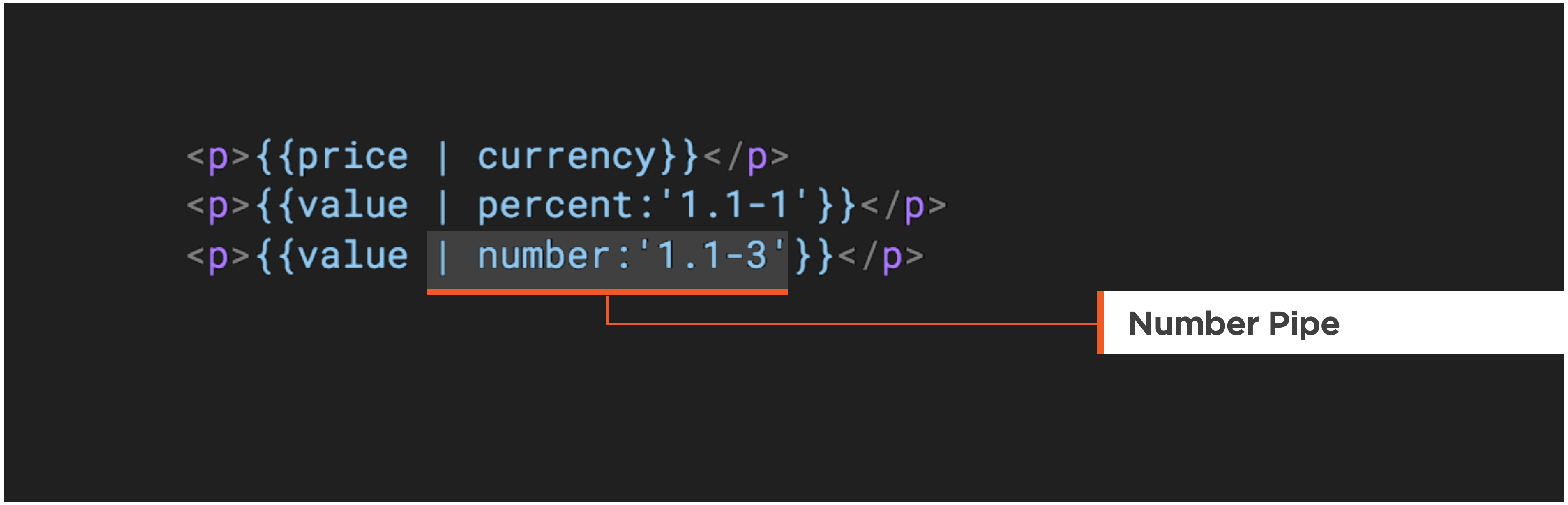
```
<p>{{eventDate | date:'medium'}}</p>
<p>{{eventDate | date:'yMMMd'}}</p>
```

Date Pipe

https://angular.io/docs/ts/latest/api/

Date accepts format

expression | date[:format]



```
<p>{{price | currency}}</p>
<p>{{value | percent:'1.1-1'}}</p>
<p>{{value | number:'1.1-3'}}</p>
```

Numeric Pipes

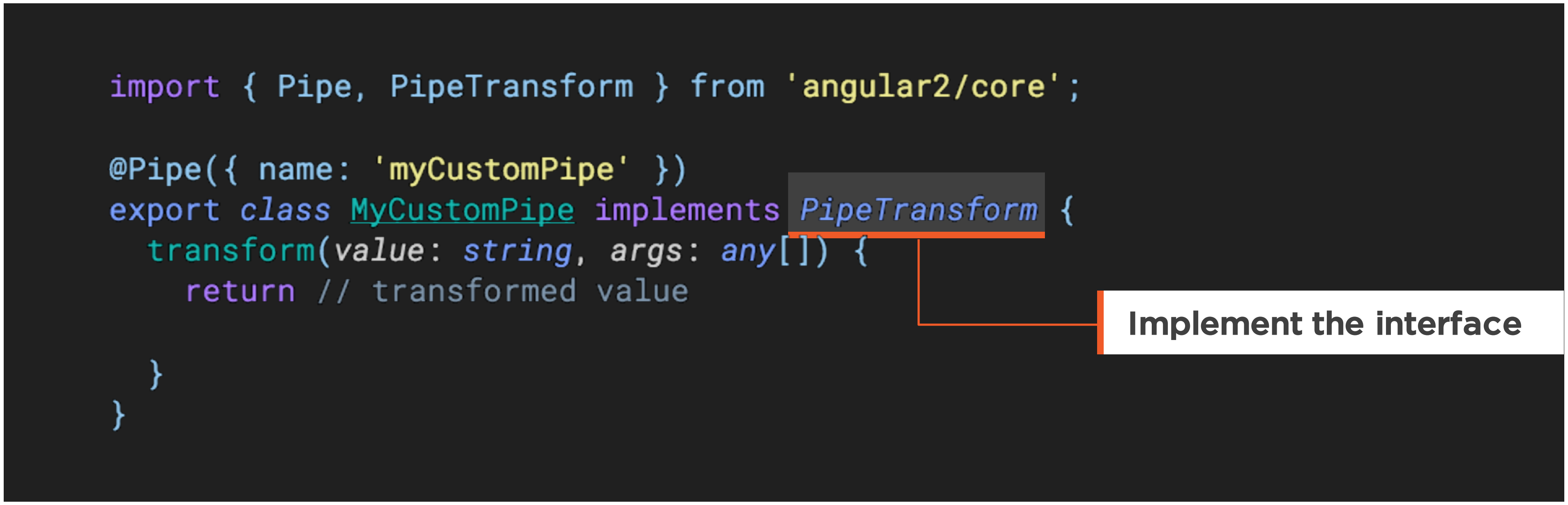
Number and Percent accept digitInfo

Expression | number[:digitInfo]

{minIntegerDigits}.{minFractionDigits}-{maxFractionDigits}

Async Pipe

Subscribes to a Promise or an Observable, returning the latest value emitted



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

@Pipe({ name: 'myCustomPipe' })
export class MyCustomPipe implements PipeTransform {
  transform(value: string, args: any[]) {
    return // transformed value

  }
}
```

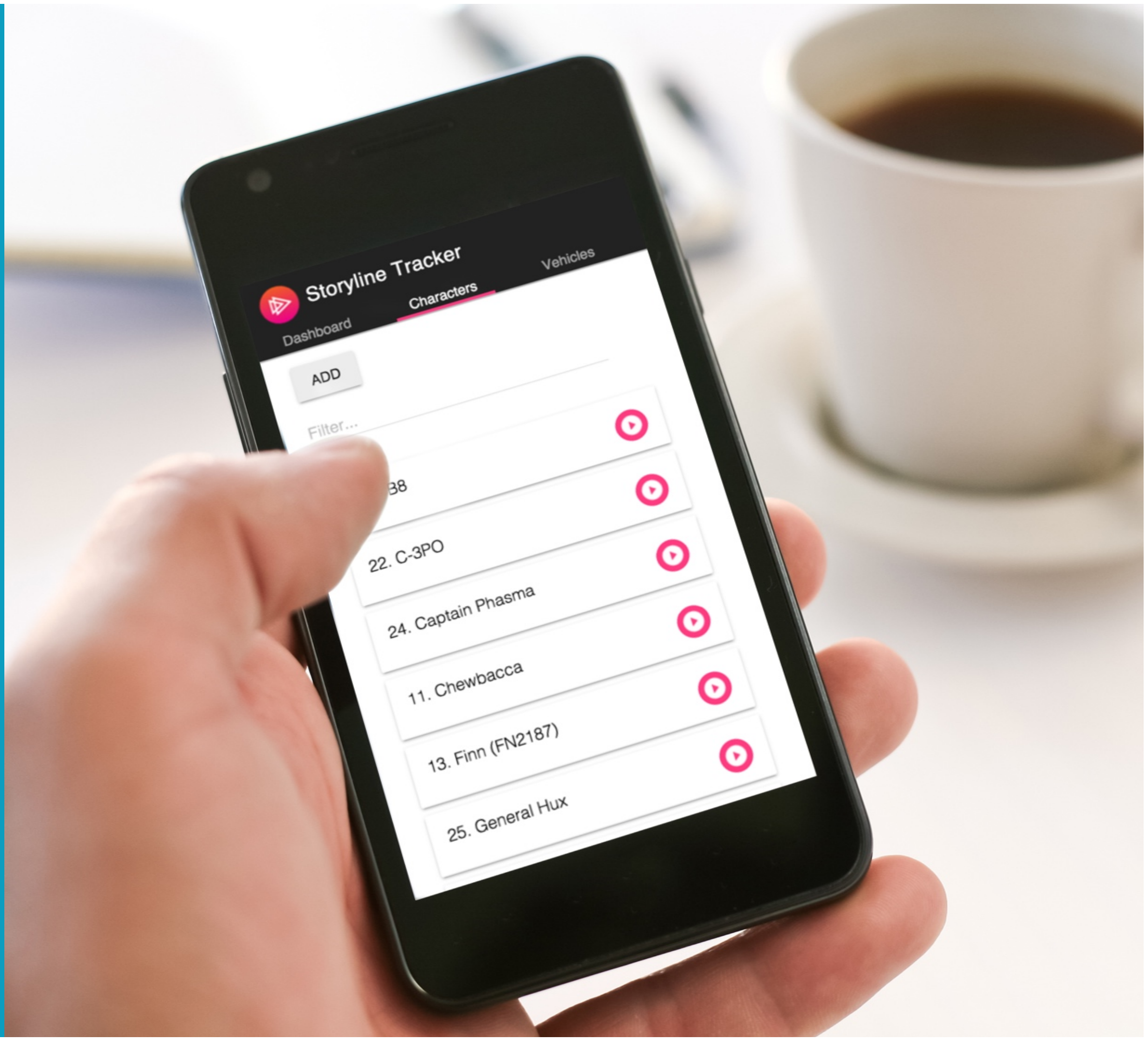
Custom Pipes

value to transform

Optional args

Demo

Putting it all Together



Template Syntax

Data Binding

Unidirectional Data Flow

Attribute Directives

Structural Directives

Pipes