COMPONENTS AND TEMPLATES

TBC

📖 INTERPOLATION

You can display data by binding controls in an HTML template to properties of an Angular component.

📖 INTERPOLATION

```
1. import { Component } from '@angular/core';
2.
3. @Component({
4.   selector: 'app-root',
5.   template: `
6.     <h1>{{title}}</h1>
7.     <h2>My favorite hero is: {{myHero}}</h2>
8.     `
9. })
10. export class AppComponent {
11.   title = 'Tour of Heroes';
12.   myHero = 'Windstorm';
13. }
```

TBC

📖 *NGFOR

```
export class AppComponent {
  title = 'Tour of Heroes';
  heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
  myHero = this.heroes[0];
}
```

TBC

📖 *NGFOR

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero }}
    </li>
  </ul>
`
```

TBC

📖 *NGIF

```
<p *ngIf="heroes.length > 3">There are many heroes!</p>
```

TBC

📖 TEMPLATE SYNTAX

- Interpolation
- Template expression

TBC

📖 TEMPLATE EXPRESSION

```
<!-- "The sum of 1 + 1 is 2" -->

<p>The sum of 1 + 1 is {{1 + 1}}.</p>
```

TBC

📖 TEMPLATE EXPRESSION

You can't use JavaScript expressions that have or promote side effects, including:

- Assignments (=, +=, -=, ...)
- Operators such as new, typeof, instanceof, etc.
- Chaining expressions with ; or ,
- The increment and decrement operators ++ and --
- Some of the ES2015+ operators

TBC

📖 TEMPLATE STATEMENTS

```
<button (click)="deleteHero()">Delete hero</button>
```

TBC

📖 DATA BINDING

- One-way from data source to view target
- One-way from view target to data source
- Two-way

TBC

📖 HTML VS DOM

Data-binding works with properties of DOM elements, components, and directives, not HTML attributes.

Attributes initialize DOM properties and then they are done. Property values can change; attribute values can't.

📖 PROPERTY BINDING

Property binding flows a value in one direction, from a component's property into a target element property.

📖 PROPERTY BINDING

```
<!-- Bind button disabled state to `isUnchanged` property -->
<button [disabled]="isUnchanged">Disabled Button</button>
```

TBC

📖 PROPERTY BINDING

```
<!-- Bind button disabled state to `isUnchanged` property -->
<button [disabled]="isUnchanged">Disabled Button</button>
```

TBC

📖 PROPERTY BINDING

```
<app-item-detail [childItem]="parentItem"></app-item-detail>
```

```
@Input() childItem: string;
```

TBC

📖 PROPERTY BINDING

```
<p><img src="{{itemImageUrl}}"> is the <i>interpolated</i> image.</p>
<p><img [src]="itemImageUrl"> is the <i>property bound</i> image.</p>

<p><span>"{{interpolationTitle}}" is the <i>interpolated</i> title.</span></p>
<p>"<span [innerHTML]="propertyTitle"></span>" is the <i>property bound</i> title.</p>
```

TBC

📖 PROPERTY BINDING

- attribute binding
- class binding
- style binding

📖 ATTRIBUTE BINDING

```
<!-- create and set an aria attribute for assistive technology -->
<button [attr.aria-label]="actionName">{{actionName}} with Aria</button>
```

TBC

📖 CLASS BINDING

```
<h3>toggle the "special" class on/off with a property:</h3>
<div [class.special]="isSpecial">The class binding is special.</div>
```

TBC

📖 STYLE BINDING

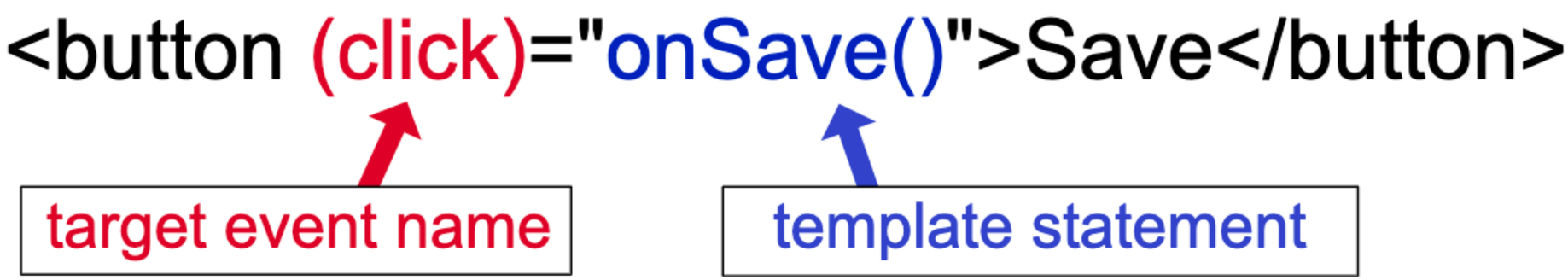
```
<button [style.color]="isSpecial ? 'red': 'green'">Red</button>
<button [style.background-color]="canSave ? 'cyan': 'grey'" >Save</button>
```

TBC

📖 EVENT BINDING

Event binding allows you to listen for certain events such as keystrokes, mouse movements, clicks, and touches.

📖 EVENT BINDING



TBC

📖 $EVENT

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

TBC

📖 TWO-WAY BINDING

Two-way binding gives your app a way to share data between a component class and its template.

TBC

📖 TWO-WAY BINDING

```
@Input()  size: number | string;
@Output() sizeChange = new EventEmitter<number>();

dec() { this.resize(-1); }
inc() { this.resize(+1); }

resize(delta: number) {
  this.size = Math.min(40, Math.max(8, +this.size + delta));
  this.sizeChange.emit(this.size);
}
```

TBC

📖 TWO-WAY BINDING

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
<div [style.font-size.px]="fontSizePx">Resizable Text</div>
```

```
<app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>
```

TBC

📖 BUILT-IN DIRECTIVES

- NgClass
- NgStyle
- NgModel
- NgFor
- NgIf
- NgSwitch

TBC

📖 TEMPLATE VARIABLES

```
<input #phone placeholder="phone number" />

<!-- lots of other elements -->

<!-- phone refers to the input element; pass its `value` to an event handler -->
<button (click)="callPhone(phone.value)">Call</button>
```

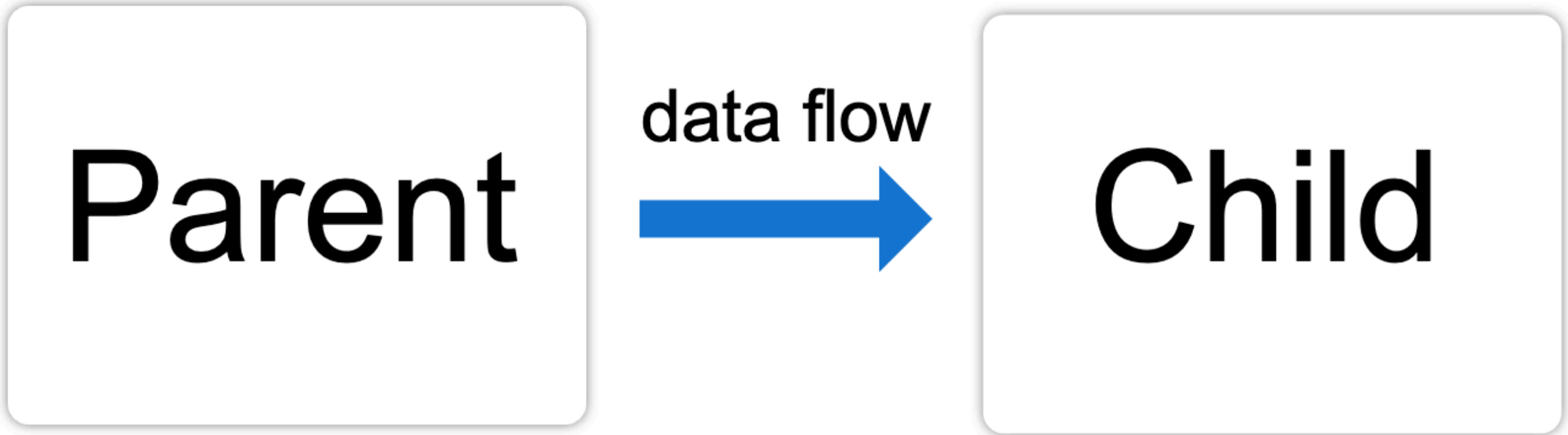
TBC

📖 @INPUT AND @OUTPUT

@Input() and @Output() allow Angular to share data between the parent context and child directives or components. An @Input() property is writable while an @Output() property is observable.

📖 @INPUT

@Input



TBC

📖 @INPUT CHILD COMPONENT

```
import { Component, Input } from '@angular/core'; // First, import Input
export class ItemDetailComponent {
  @Input() item: string; // decorate the property with @Input()
}
```

```
<p>
  Today's item: {{item}}
</p>
```

TBC

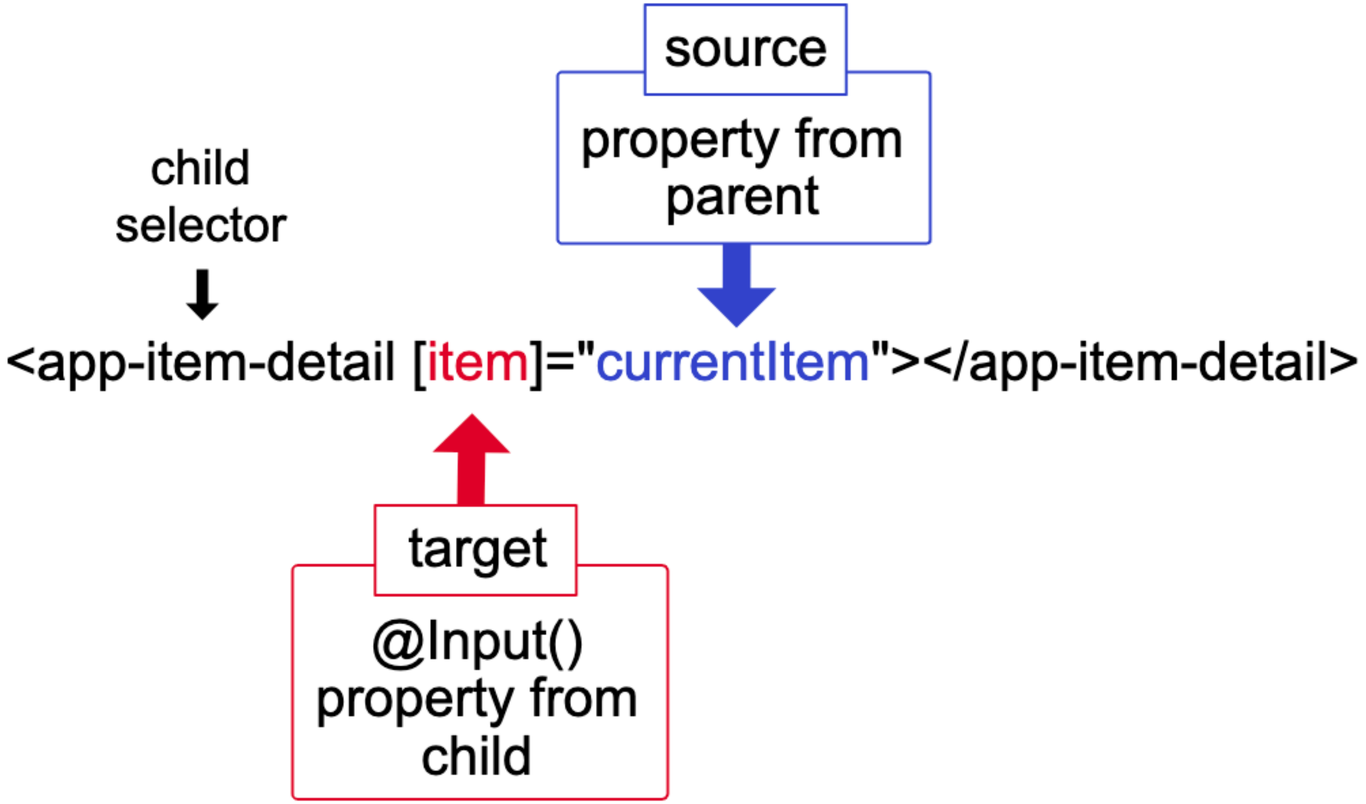
📖 @INPUT PARENT COMPONENT

```
<app-item-detail [item]="currentItem"></app-item-detail>
```

```
export class AppComponent {
  currentItem = 'Television';
}
```

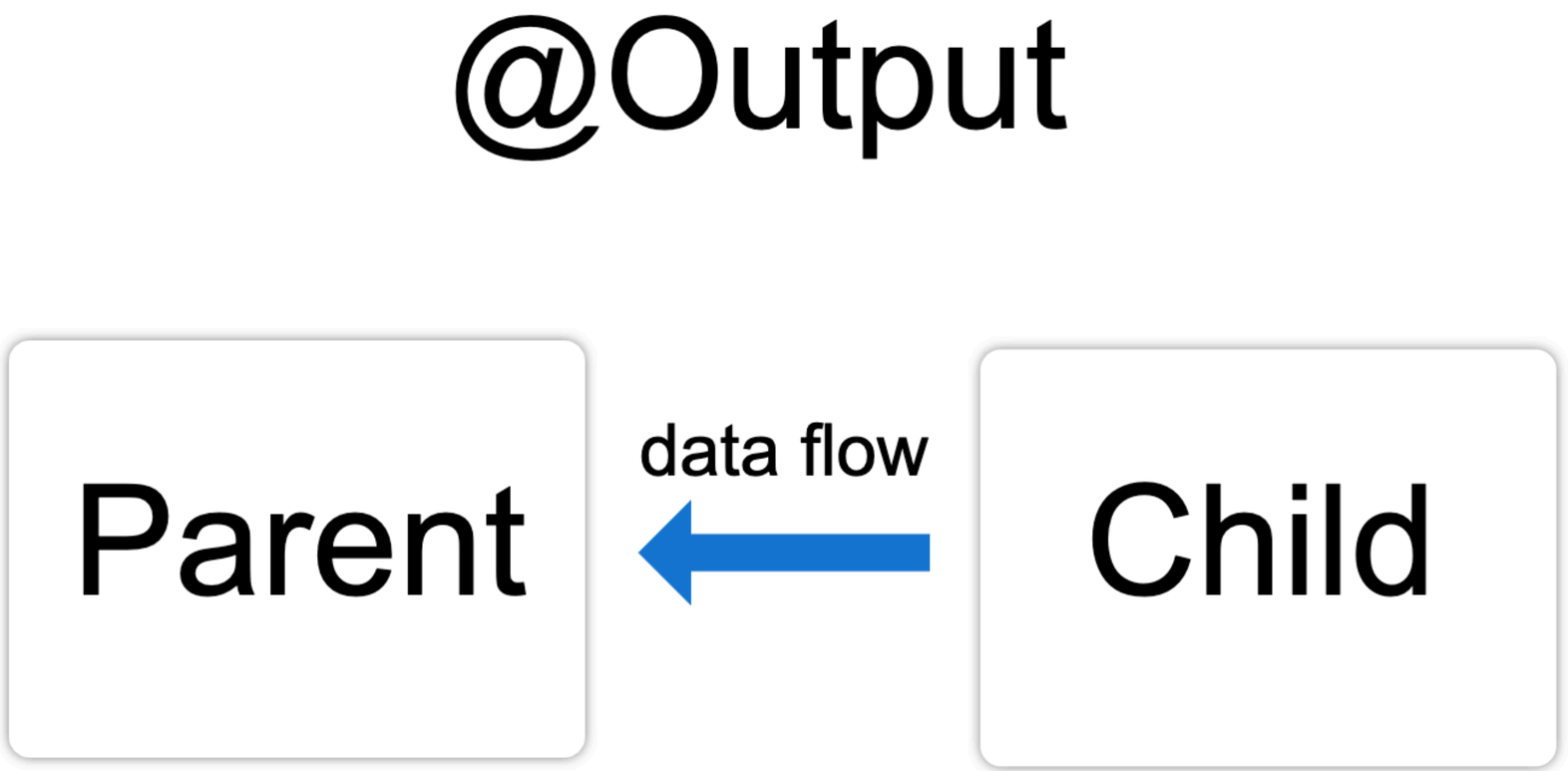
TBC

📖 @INPUT


source
property from parent
child selector
<app-item-detail [item]="currentItem"></app-item-detail>
target
@Input() property from child

TBC

📖 @OUTPUT



TBC

📖 @OUTPUT CHILD COMPONENT

```
export class ItemOutputComponent {

  @Output() newItemEvent = new EventEmitter<string>();

  addNewItem(value: string) {
    this.newItemEvent.emit(value);
  }
}
```

TBC

📖 @OUTPUT CHILD COMPONENT

```
<label>Add an item: <input #newItem></label>
<button (click)="addNewItem(newItem.value)">Add to parent's list</button>
```

📖 @OUTPUT PARENT COMPONENT

```
export class AppComponent {
  items = ['item1', 'item2', 'item3', 'item4'];

  addItem(newItem: string) {
    this.items.push(newItem);
  }
}
```

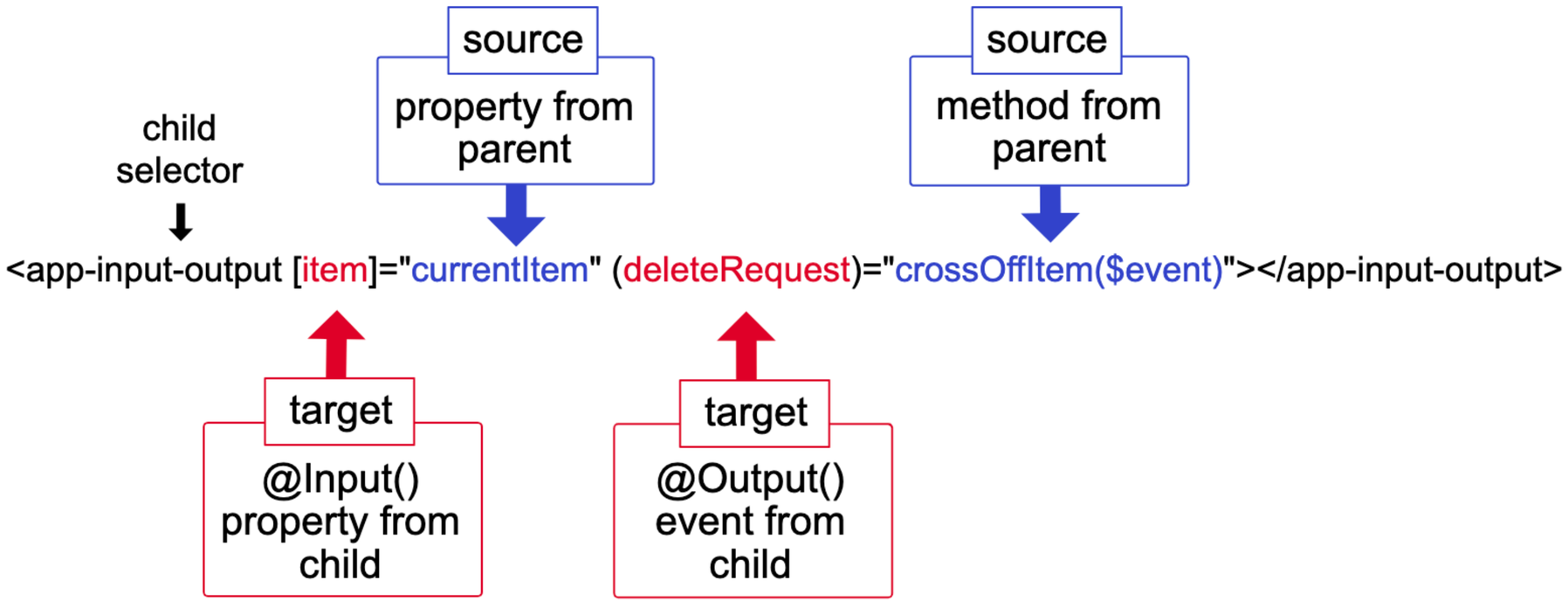
TBC

📖 @OUTPUT PARENT COMPONENT

```
<app-item-output (newItemEvent)="addItem($event)"></app-item-output>
```

TBC

📖 @INPUT AND @OUTPUT



TBC

📖 TEMPLATE OPERATORS

- Pipe
- Safe navigation operator and null property paths
- Non-null assertion operator

📖 LIFECYCLE HOOKS

Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.

TBC

📖 LIFECYCLE HOOKS

```
export class PeekABoo implements OnInit {
  constructor(private logger: LoggerService) { }

  // implement OnInit's `ngOnInit` method
  ngOnInit() { this.logIt(`OnInit`); }

  logIt(msg: string) {
    this.logger.log(`#${nextId++} ${msg}`);
  }
}
```

TBC

📖 LIFECYCLE HOOKS

- ngOnChanges()
- ngOnInit()
- ngDoCheck()
- ngAfterContentInit()
- ngAfterContentChecked()
- ngAfterViewInit()
- ngAfterViewChecked()
- ngOnDestroy()

TBC

📖 COMPONENT STYLES

Angular applications are styled with standard CSS. That means you can apply everything you know about CSS stylesheets, selectors, rules, and media queries directly to Angular applications.

TBC

📖 COMPONENT STYLES

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Tour of Heroes</h1>
    <app-hero-main [hero]="hero"></app-hero-main>
  `,
  styles: ['h1 { font-weight: normal; }']
})
export class HeroAppComponent {
/* . . . */
}
```

TBC

📖 COMPONENT STYLES

```
@Component({
  selector: 'app-root',
  template: `
    <h1>Tour of Heroes</h1>
    <app-hero-main [hero]="hero"></app-hero-main>
  `,
  styleUrls: ['./hero-app.component.css']
})
export class HeroAppComponent {
/* . . . */
}
```

TBC

📖 COMPONENT STYLES

```
/* The AOT compiler needs the `./` to show that this is local */
@import './hero-details-box.css';
```

TBC

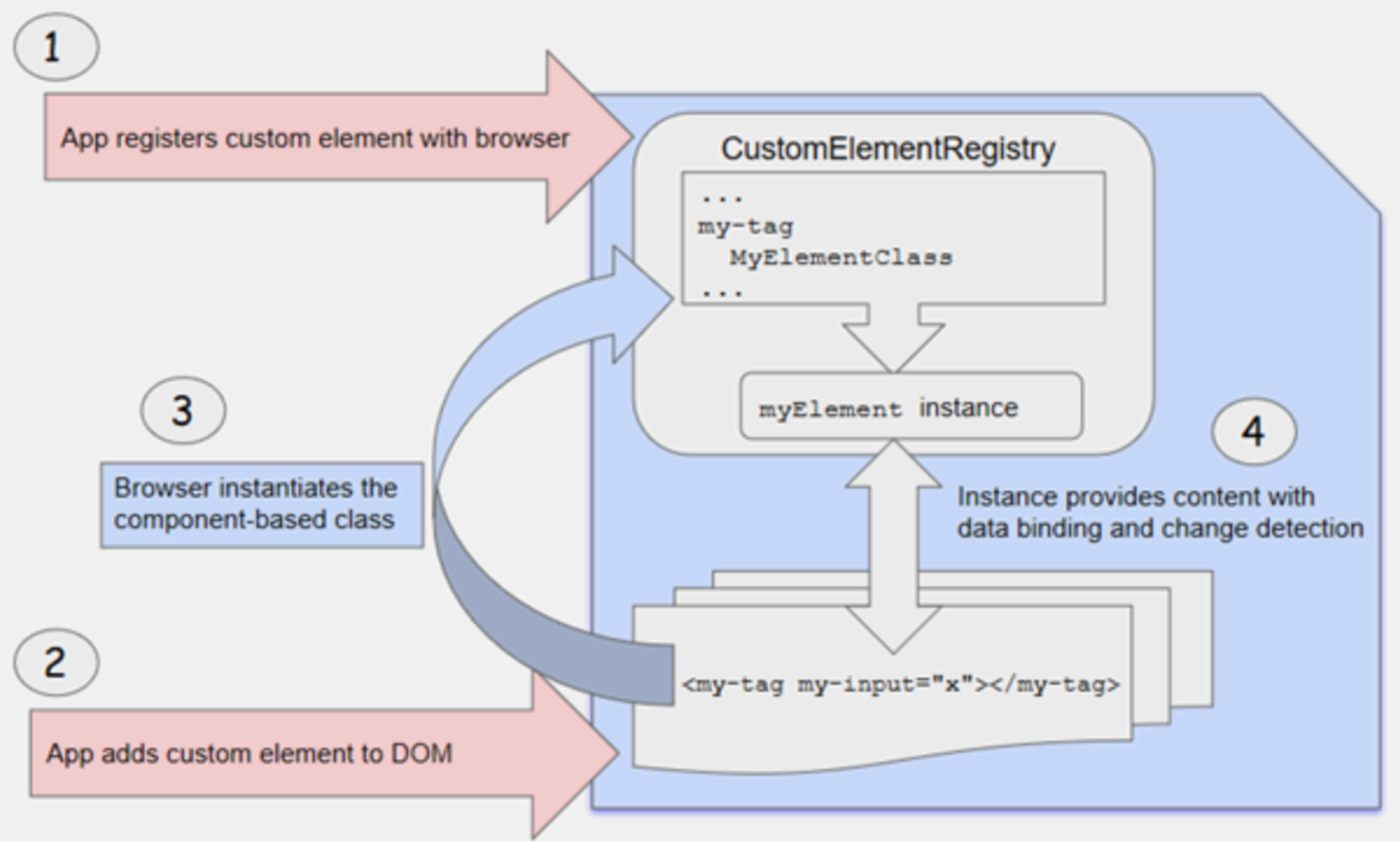
📖 COMPONENT STYLES

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
...
```

📖 ANGULAR ELEMENTS

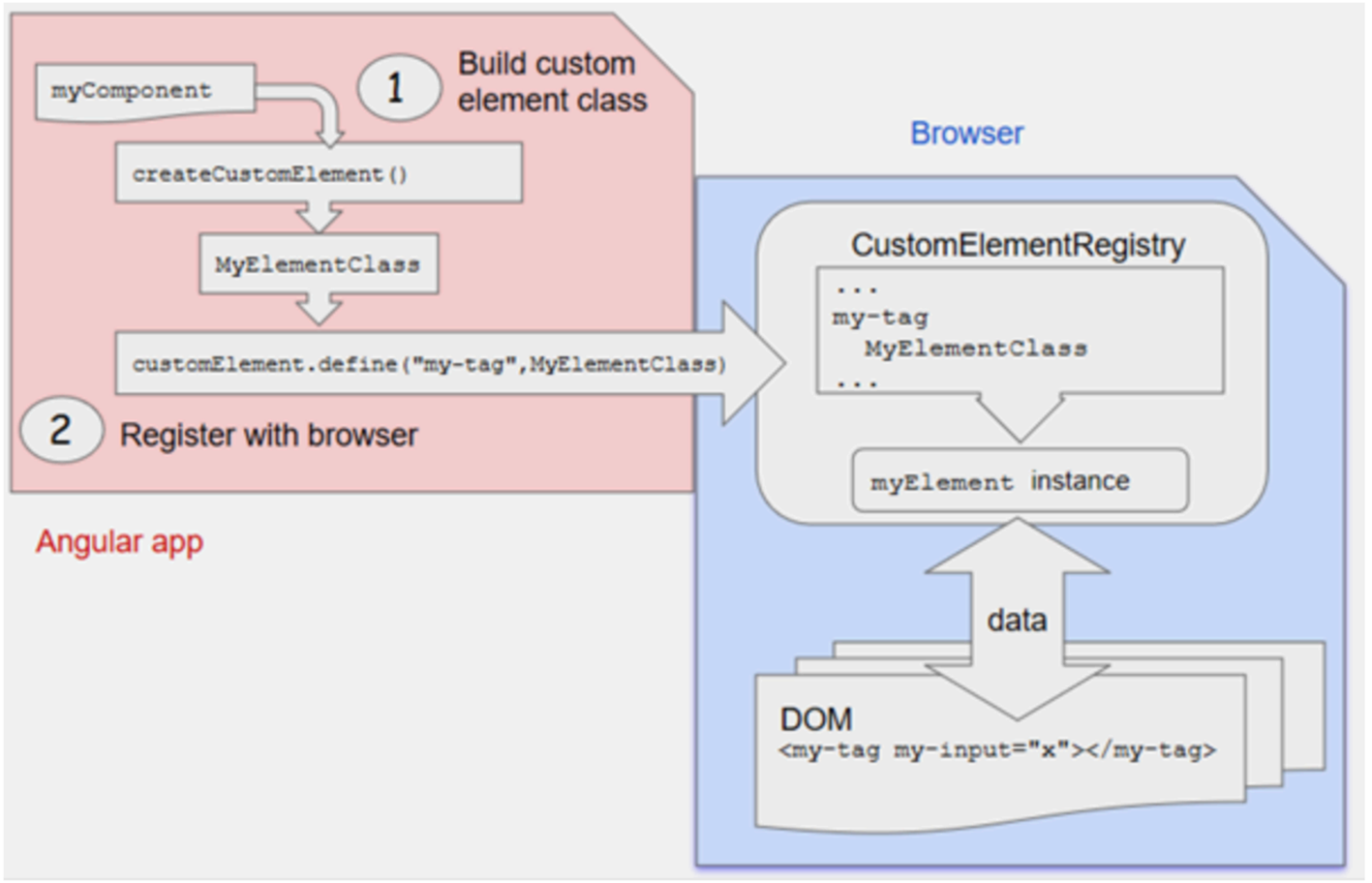
Angular elements are Angular components packaged as custom elements (also called Web Components), a web standard for defining new HTML elements in a framework-agnostic way.

📖 ANGULAR ELEMENTS



TBC

📖 ANGULAR ELEMENTS



TBC

📖 DIRECTIVES

- Components
- Structural directives
- Attribute directives

TBC

📖 ATTRIBUTE DIRECTIVES

```
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
    constructor(el: ElementRef) {
       el.nativeElement.style.backgroundColor = 'yellow';
    }
}
```

TBC

HERE GOES!

TBC