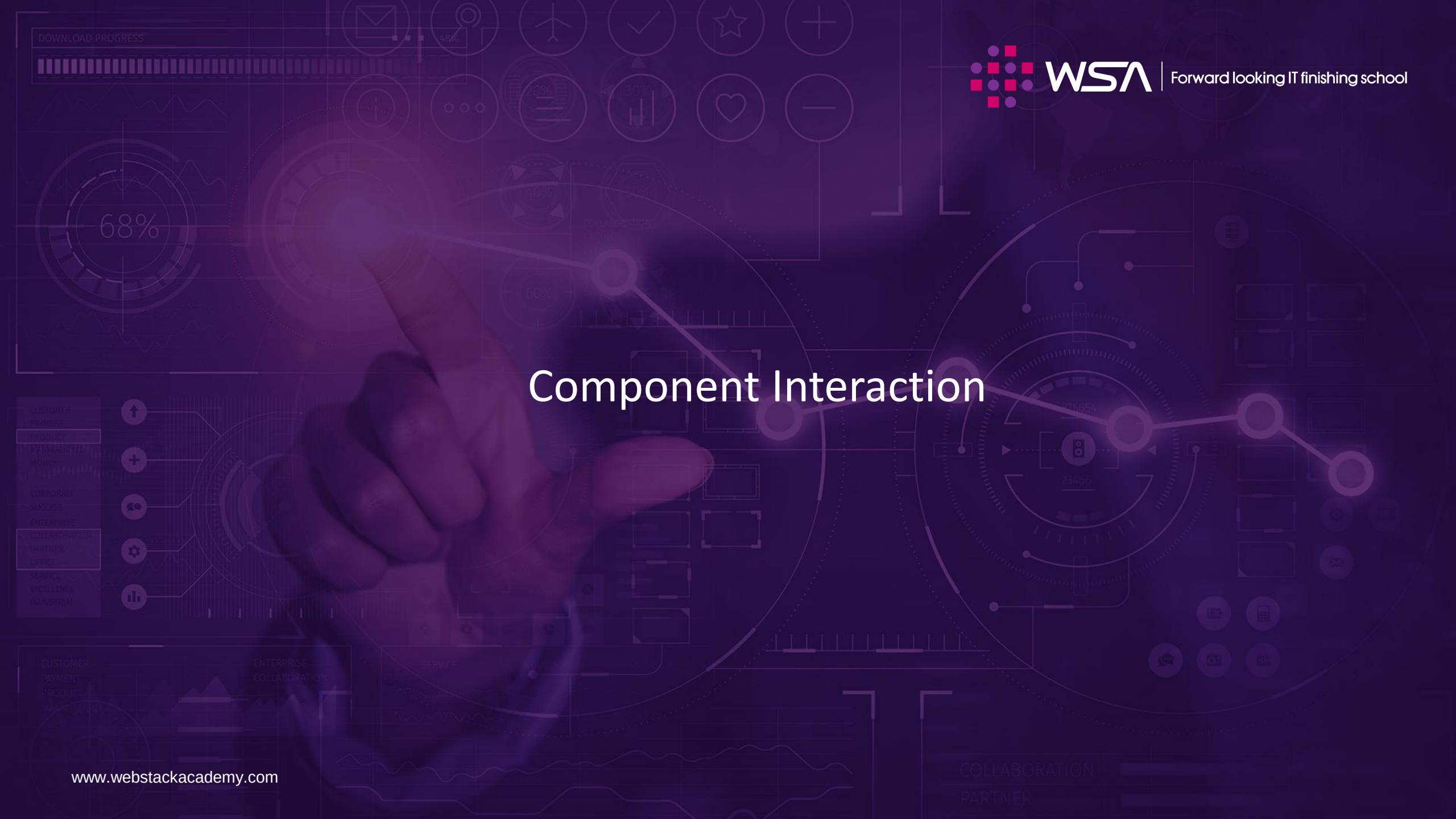
WSA | Forward looking IT finishing school

Sharing Data Between Components

Angular



WSA | Forward looking IT finishing school

Component Interaction

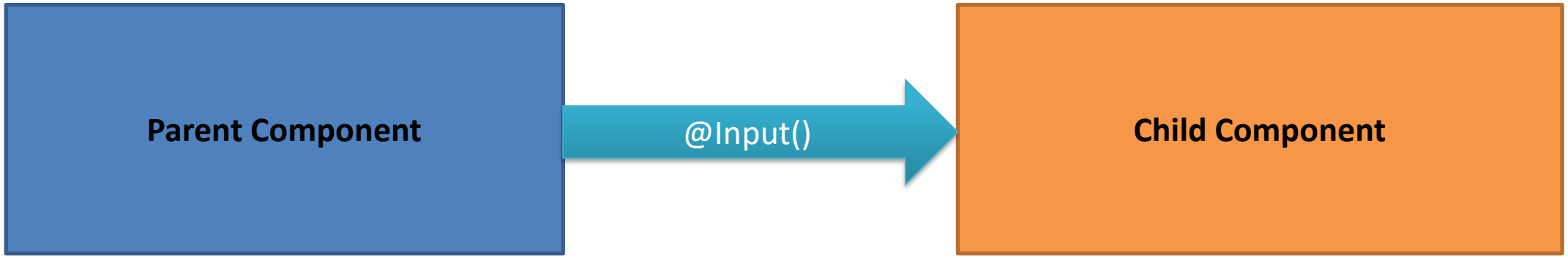
Component Interaction

- To share the data between the components or to interact with the components, first we need to know the relationship among the components.
- The relationship between the components can be anyone of the following
  - ✓ **Parent to child relationship**
  - ✓ **Child to parent relationship**
  - ✓ **sibling relationship or No relationship between components**

Component Interaction

Parent to child sharing via @Input Directive:

- In parent to child relationship is one of the most common and straightforward way of sharing data.
- It works by using the @Input() decorator to allow data to be passed via the template.



Component Interaction

@Input – Usage Example

- **app.component.ts**

```
export class AppComponent {
    parentVal:string = 'I am from Parent"
}
```

- **child.component.ts**

```
import { Component, OnInit, Input }
                                from '@angular/core';
export class ParenttochildComponent {
  @Input() parentVal:string;
}
```

- **app.component.html**

```
<h1>Demo of Parent to Child Interaction via @Input</h1>
<child [parentVal]='parentVal'></child>
```

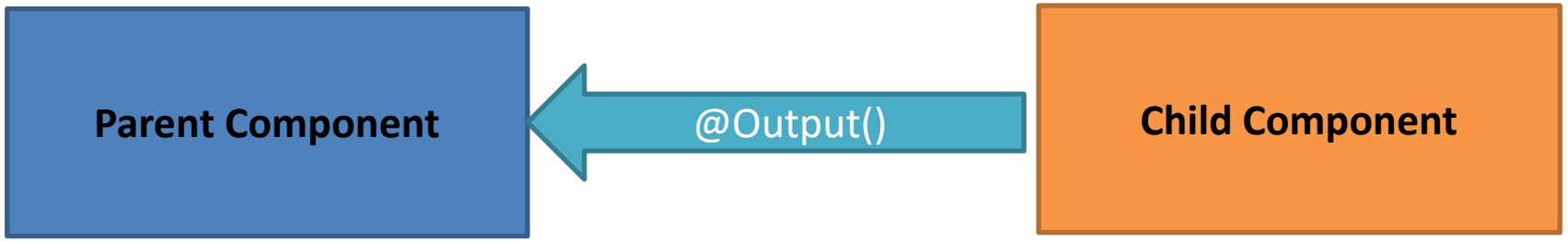
- **child.component.ts**

```
<h1>I Am Child</h1>
<p>Value from parent: {{parentVal}}</p>
```

Component Interaction

Child to Parent sharing via @Output Directive and EventEmitter:

- Another way to share data is to emit data from the child, which can be listed to by the parent.
- This approach is ideal when you want to share data changes that occur on things like button clicks, form entries, and other user events.
- In the parent, we create a function to receive the message and set it equal to the message variable.
- In the child, we declare a messageEvent variable with the Output decorator and set it equal to a new event emitter. Then we create a function named sendMessage that calls emit on this event with the message we want to send. Lastly, we create a button to trigger this function.
- The parent can now subscribe to this messageEvent that's outputted by the child component, then run the receive message function whenever this event occurs.



Component Interaction

@Output – Usage Example

- **app.component.ts**

```
export class AppComponent {
  title = 'ComponentInteraction';
  parentVal:string = "I am from Parent";
}
```

- **child.component.ts**

```
import { Component, OnInit, Input }
                                from '@angular/core';
export class ParenttochildComponent {
 message: string = "Hola Mundo!"
 @Output() messageEvent = new EventEmitter<string>();
  sendMessage() {
    this.messageEvent.emit(this.message)
  }
}
```

- **app.component.html**

```
<h1>Demo of Child to Parent Interaction via @Output</h1>
Message: {{message}}
<app-child (messageEvent)="receiveMessage($event)"></app-child>
```

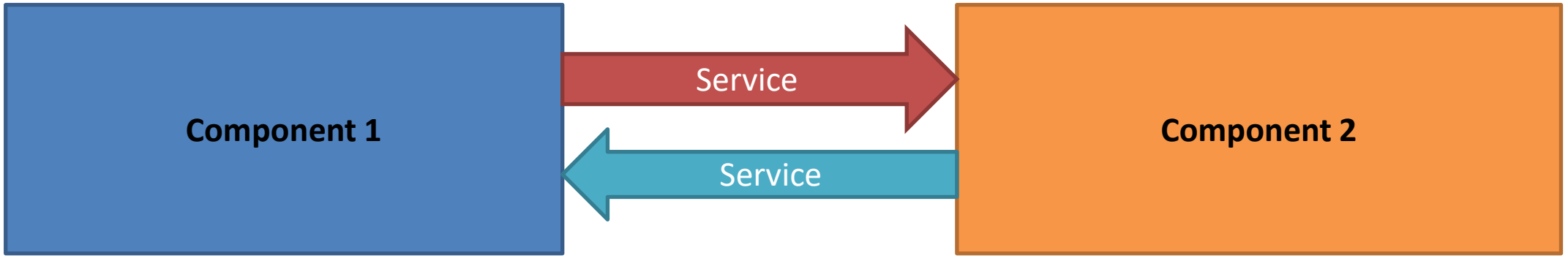
- **child.component.html**

```
<h1>I Am Child</h1>
<button (click)="sendMessage()">Send Message</button>
```

Component Interaction

Unrelated Components: Sharing Data with a Service:

- When passing data between components that lack a direct connection, such as siblings, grandchildren, etc, will make use shared services to pass the data.
- When you have data that should always been in sync, RxJS BehaviorSubject very useful in this situation.



Benefits of using RXJS BehaviorSubject:

- It will always return the current value on subscription - there is no need to call onnext
- It has a getValue() function to extract the last value as raw data.
- It ensures that the component always receives the most recent data.

Component Interaction

Unrelated components

- **Step 1: Create a service with cli. Write the following code in the service file.**

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable()
export class DataService {

  private messageSource = new BehaviorSubject('default message');
  currentMessage = this.messageSource.asObservable();

  constructor() { }

  changeMessage(message: string) {
    this.messageSource.next(message)
  }

}
```

Component Interaction

Unrelated components

- **Step 2: Subscribe the service in component by injecting in the component.**

```
import { Component, OnInit } from '@angular/core';
import { DataService } from "../data.service";

@Component({
selector: 'app-parent',
template: `
            {{message}}
          `,
styleUrls: ['./sibling.component.css']
})
export class ParentComponent implements OnInit {
message:string;

constructor(private data: DataService) { }

ngOnInit() {
    this.data.currentMessage.subscribe(message => this.message = message)
 }
}
```

Component Interaction

Unrelated components

- **Step 3: Change the value from another component by using service change method and observe the update in other component.**

```
@Component({
selector: 'app-sibling',
template: `
          {{message}}
          <button (click)="newMessage()">New Message</button>
          `,
styleUrls: ['./sibling.component.css']
})
export class SiblingComponent implements OnInit {
message:string;

constructor(private data: DataService) { }
ngOnInit() {
     this.data.currentMessage.subscribe(message => this.message = message)
}
newMessage() {
     this.data.changeMessage("Hello from Sibling")
}
}
```

WSA | Forward looking IT finishing school

Thank you

WebStack Academy

#83, Farah Towers,
1st Floor, MG Road,
Bangalore – 560001

M: +91-809 555 7332
E: training@webstackacademy.com

WSA in Social Media: