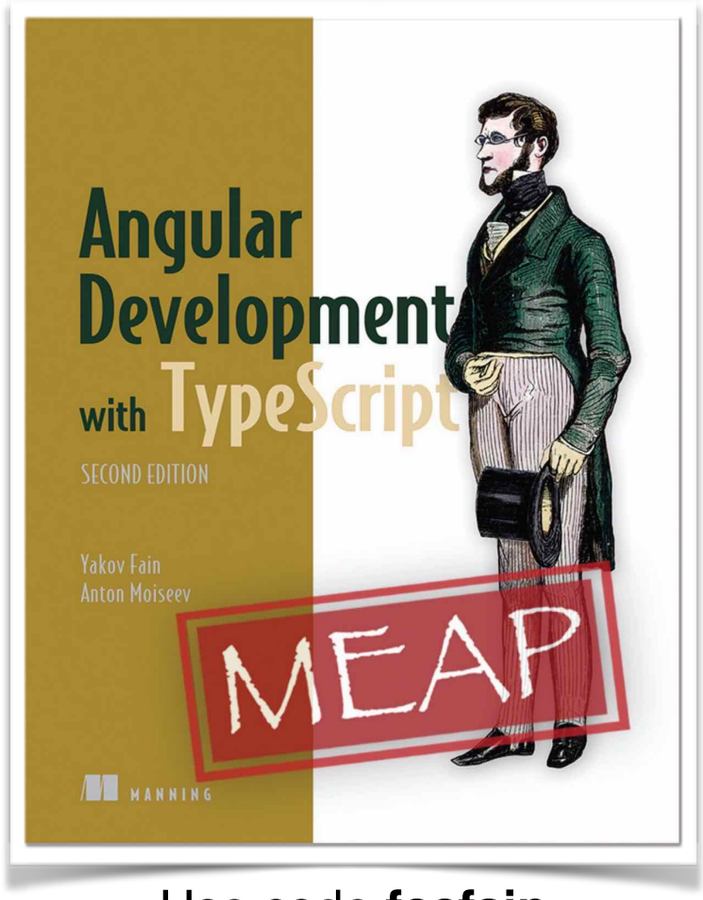
# Implementing inter-component communications in Angular


@yfain

# About myself

- Work for Farata Systems
- Angular consulting and training
- Java Champion
- Co-authored two editions of the book
  “Angular Development with TypeScript”
- Working on a book
  “Get Programming with TypeScript”




Use code **fccfain**
for 37% off
at manning.com

@yfain

# In this unit

- @Input and @Output properties
- A parent component as a mediator
- An injectable service as a mediator
- The ngrx store as a mediator
- Passing params via the router
- Projection of HTML

NYBS TR1 Jun 09 15:34 Logout

Spot | Settings | Floor Admin

123A: Date: 10-May-06 Time: 11:54
Long: USD 49
Short EUR 49
Total Net limit: 50

CREDIT ALERT: Prime Credit ABCD
CREDIT ALERT: Credit 123
CREDIT ALERT: Credit Group NYCD

| | | | | | |
|---|---|---|---|---|---|
| EUR/USD | 1.41 55 | 56 1.41 | 3-Sep | | |
| 54 | 26 | 55 BID | 56 OFFER | 43 | 57 |
| TRA 83 | | | | | |
| EUR/JPY | 163. 97 | 99 163. | 3-Sep | | |
| 97 | 13 | 97 BID | 99 OFFER | | 99 |
| TRA A3 | | | | | |
| USD/JPY | 115. 83 | 85 115. | 3-Sep | | |
| 81 | | 83 BID | 85 OFFER | | 86 |
| TRA A3 | | | | | |
| EUR/CHF | 1.66 35 | 36 1.66 | 3-Sep | | |
| 35 | | 35 BID | 37 OFFER | | 38 |
| TRA A3 | | | | | |
| EUR/GBP | 0.69 310 | 325 0.69 | 3-Sep | | |
| 305 | | 31 0 BID | 32 5 OFFER | | 335 |
| TRA A3 | | | | | |
| USD/CHF | 1.17 52 | 53 1.17 | 3-Sep | | |
| 50 | 7 | 52 BID | 53 OFFER | 12 | 55 |
| TRA A3 | | | | | |
| USD/MXN | 10.8 916 | 058 10.9 | 3-Sep | | |
| 901 | | 916 BID | 058 OFFER | | 064 |
| TRA A3 | | | | | |
| XAG/USD | 13.1 550 | 650 13.1 | 3-Sep | | |
| 400 | | 5 00 BID | 6 50 OFFER | | 750 |
| TRA A3 | | | | | |

USD/JPY 0.89 88
Messages go here 10 SEND

EUR/USD 56 1.41
Messages go here 10 SEND

USD/CHF 1.17 52 BUY Request
Work Balance Always
Messages go here 10 SEND

USD/CHF SELL Request
Work Balance Never 53 1.17
Messages go here 10 SEND

EUR/USD Sold 2 of 2 @0.8964 Detail...
Done: D4.4 (*Weighted Average)

EUR/USD Buy 5 of 20 + 5 @0.8964 Hide...

| | | | | |
|---|---|---|---|---|
| 0666 | Buy | 1 | @1.3648 | NYBT |
| 0668 | Buy | 3 | @1.3648 | BOAN |
| 0672 | Buy | 1 | @1.3648 | ABCD |

INTERRUPTED: D4.4 BUY

EUR/USD Buy 8 of 20 + 5 @0.8964 Hide...

| | | | | |
|---|---|---|---|---|
| 0666 | Buy | 1 | @1.3648 | NYBT |
| 0668 | Buy | 3 | @1.3648 | BOAN |
| 0672 | Buy | 1 | @1.3648 | ABCD |

DONE: 245.3 BID

EUR/USD Sold 2 of 2 @0.8964 Detail...
INTERRUPTED: D4.4 (*Weighted Average)

EUR/USD Sold 2 of 2 @0.8964 Detail...
INTERRUPTED: D4.4 (*Weighted Average)

Off All Transactions

**Rates**

| | | | |
|---|---|---|---|
| EUR/USD | 1.3648-49 | EUR/JPY | 156.88-8 |
| USD/JPY | 114.94-95 | EUR/CHF | 1.6365-6 |
| EUR/GBP | 0.67765-70 | USD/CHF | 1.1988-8 |
| XAU/USD | 668.25-50 | USD/CAD | 1.0594-9 |
| GBP/USD | 2.0154-55 | AUD/USD | 0.8121-2 |

**Trader Deals**

| Time | B/S | # | Price | CPTY | CCY Pair |
|---|---|---|---|---|---|
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |
| 17:65 | Buy | 8 | 115.70 | EBSN | USD/JPY |

**Deals**

| Time | Price | P/G | CCY Pair |
|---|---|---|---|
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |
| 16:31 | 136.42 | Given | EUR/USD |

# Input and Output Properties

- Think of a component as a black box with entry and exit doors

- Properties marked as **`@Input()`** are used for getting data from the parent component

- The parent component can pass data to its child using bindings to input properties

- Properties marked as **`@Output()`** are used for sending events (and data)  from a component

# Parent binds to input props

```
@Component({
    selector: 'app',
    template: `
    <input type="text" placeholder="Enter stock (e.g. AAPL)"  (change)="onInputEvent($event)">
    <br/>


    <order-processor [stockSymbol]="stock" quantity="100">
    </order-processor>
  `
})
class AppComponent {
    stock:string;

    onInputEvent({target}):void{
        this.stock=target.value;
    }
}
```

Binding

No binding

# Input Properties in Child

```
@Component({
    selector: 'order-processor',
    template: `
    Buying {{quantity}} shares of {{stockSymbol}}
    `})
class OrderComponent {

    @Input() quantity: number;

    private _stockSymbol: string;

    @Input()
    set stockSymbol(value: string) {
        this._stockSymbol = value;
        if (this._stockSymbol != undefined) {
          console.log(`Sending a Buy order to NASDAQ: ${this.stockSymbol} ${this.quantity}`);
        }
    }

    get stockSymbol(): string {
        return this._stockSymbol;
    }
}
```

# Output properties in a child

```
@Component({
    selector: 'price-quoter',
    template: `<strong>Inside PriceQuoterComponent:
                        {{stockSymbol}} {{price | currency:'USD'}}</strong>`,

    styles:[`:host {background: pink;}`]
})
class PriceQuoterComponent {

    @Output() lastPrice: EventEmitter <IPriceQuote> = new EventEmitter();

    stockSymbol: string = "IBM";
    price:number;

    constructor() {
        setInterval(() => {
            let priceQuote: IPriceQuote = {
                stockSymbol: this.stockSymbol,

                lastPrice: 100*Math.random()
            };

            this.price = priceQuote.lastPrice;

            this.lastPrice.emit(priceQuote);

        }, 1000);
    }
}
```

pipe

```
interface IPriceQuote {
    stockSymbol: string;
    lastPrice: number;
}
```

A child emits events via output properties

# The parent listens to the lastPrice event

```
@Component({
    selector: 'app',
    template: `
      <price-quoter (lastPrice)="priceQuoteHandler($event)"></price-quoter><br>

      AppComponent received: {{stockSymbol}} {{price | currency:'USD'}}
    `
})
class AppComponent {

    stockSymbol: string;
    price:number;

    priceQuoteHandler(event:IPriceQuote) {
        this.stockSymbol = event.stockSymbol;
        this.price = event.lastPrice;
    }
}
```

# Use of EventEmitter

- Use EventEmitter only to emit custom events to a parent component

- Even though EventEmitter extends Subject, don't manually subscribe to EventEmitter

- For TypeScript-toTypeScript communications use Subject

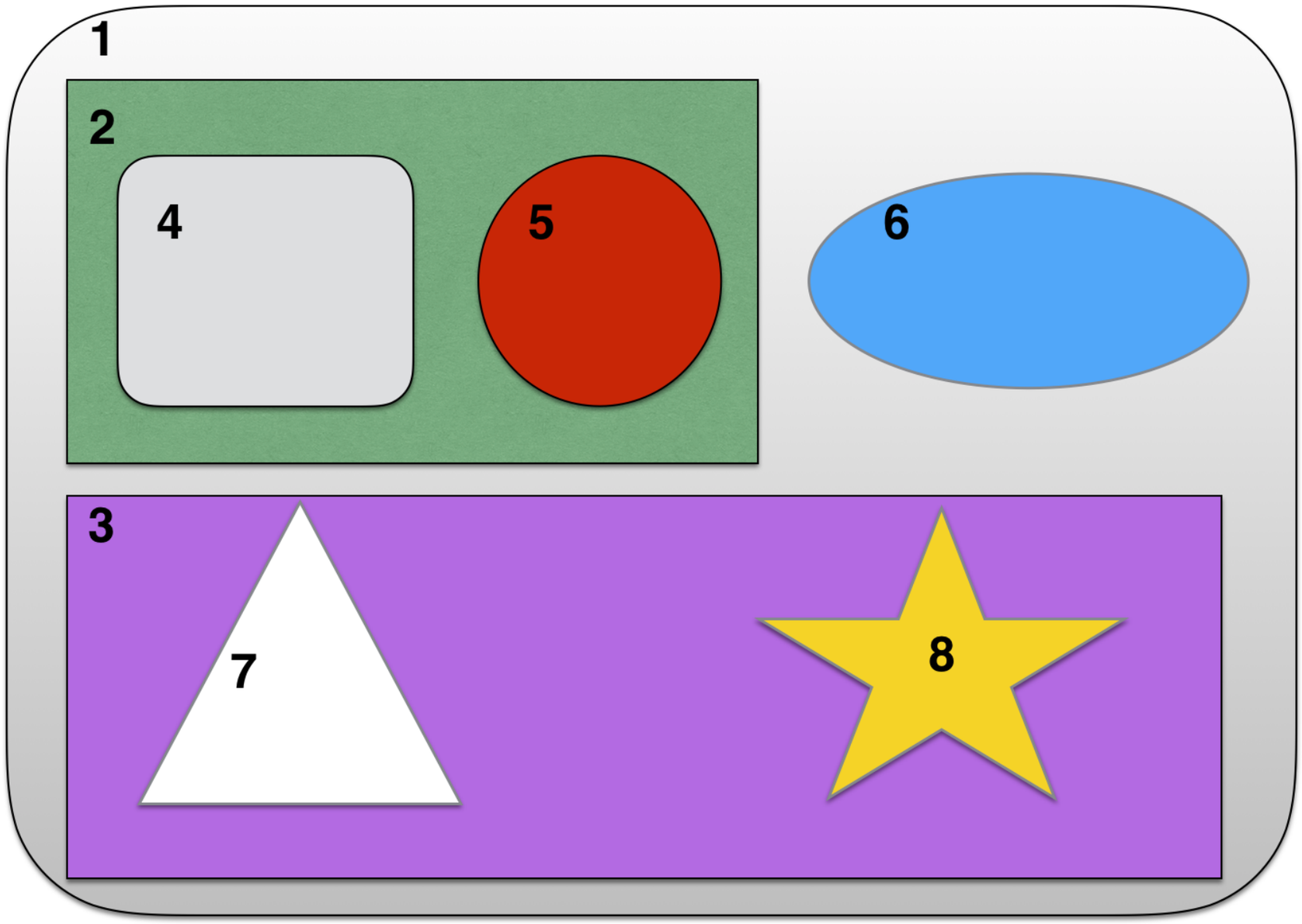
# Implementing the Mediator Design Pattern

# Loosely-Coupled Components

- The Mediator acts as a man in the middle
- A component A sends the data to a mediator, which passes the data to a component B
- **A parent component** can mediate siblings' communications
- **An injectable service** can mediate communication between any components
- **A library** can mediate communication between any components (e.g. ngrx)

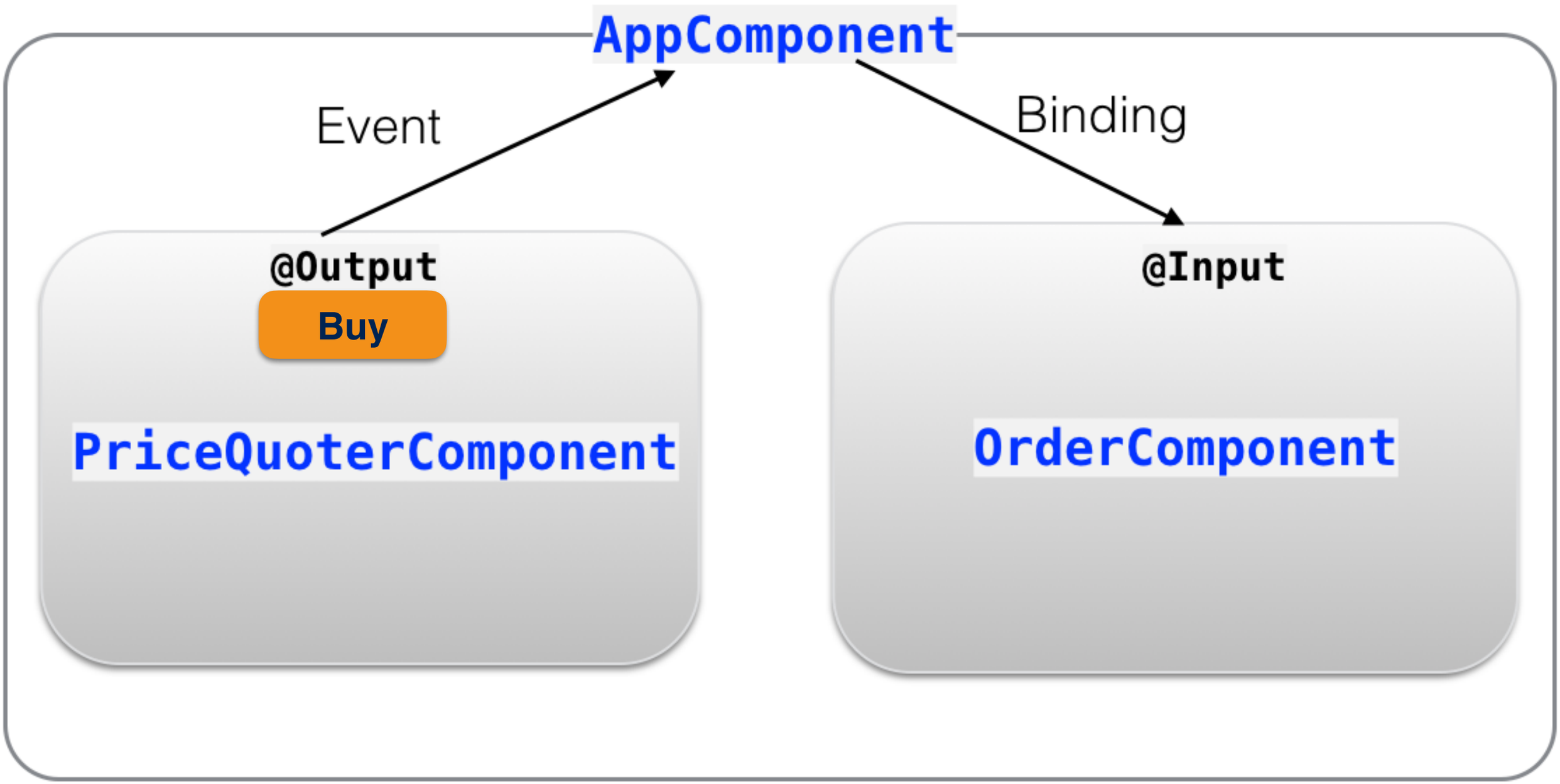
# A simple Angular app

# Parent components as mediators




**QUIZ**: How the number 7 can send the data to number 6?

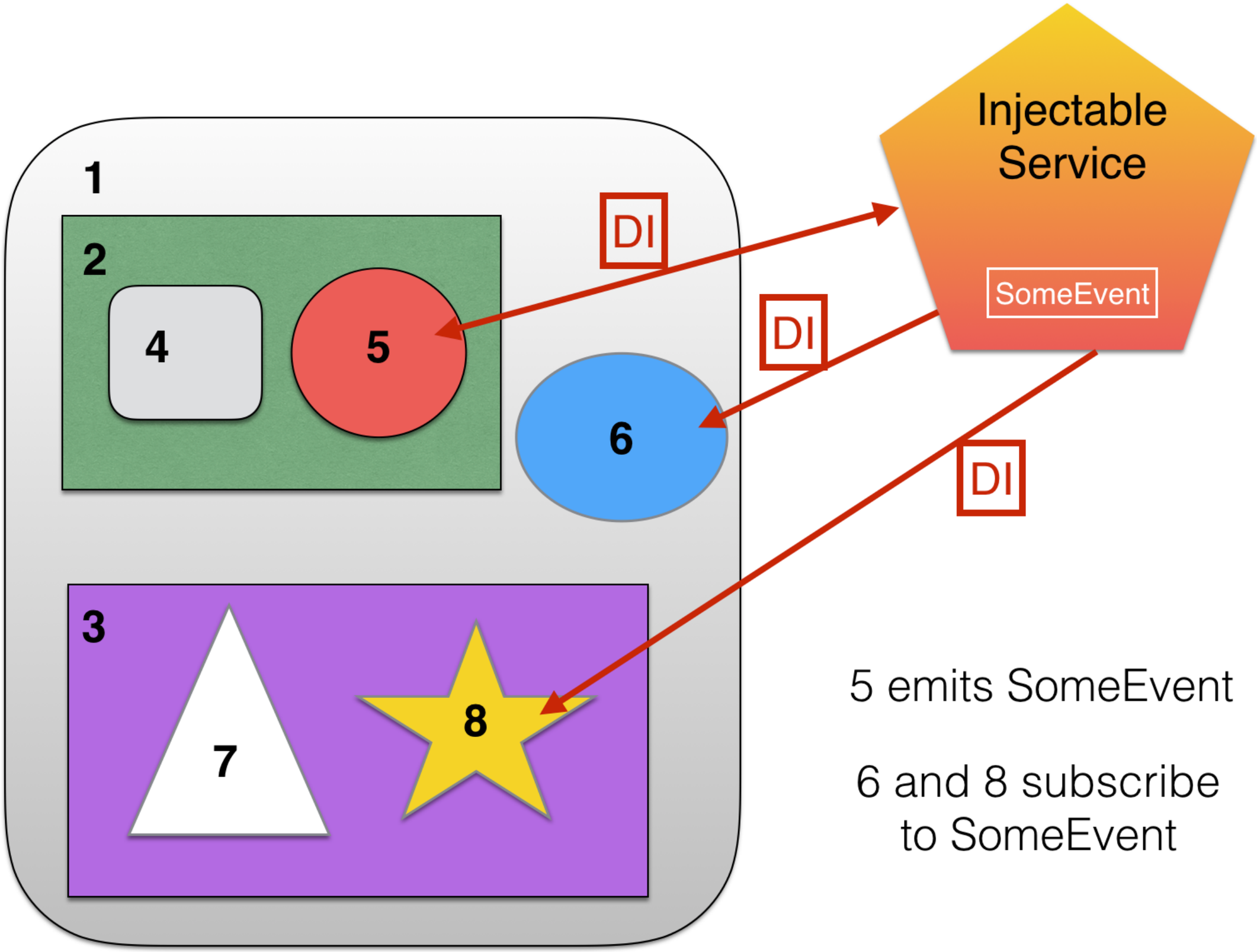
# Parent as a mediator



# Demo: Mediator parent

ng serve --app mediator-parent

# Injectable service as a mediator



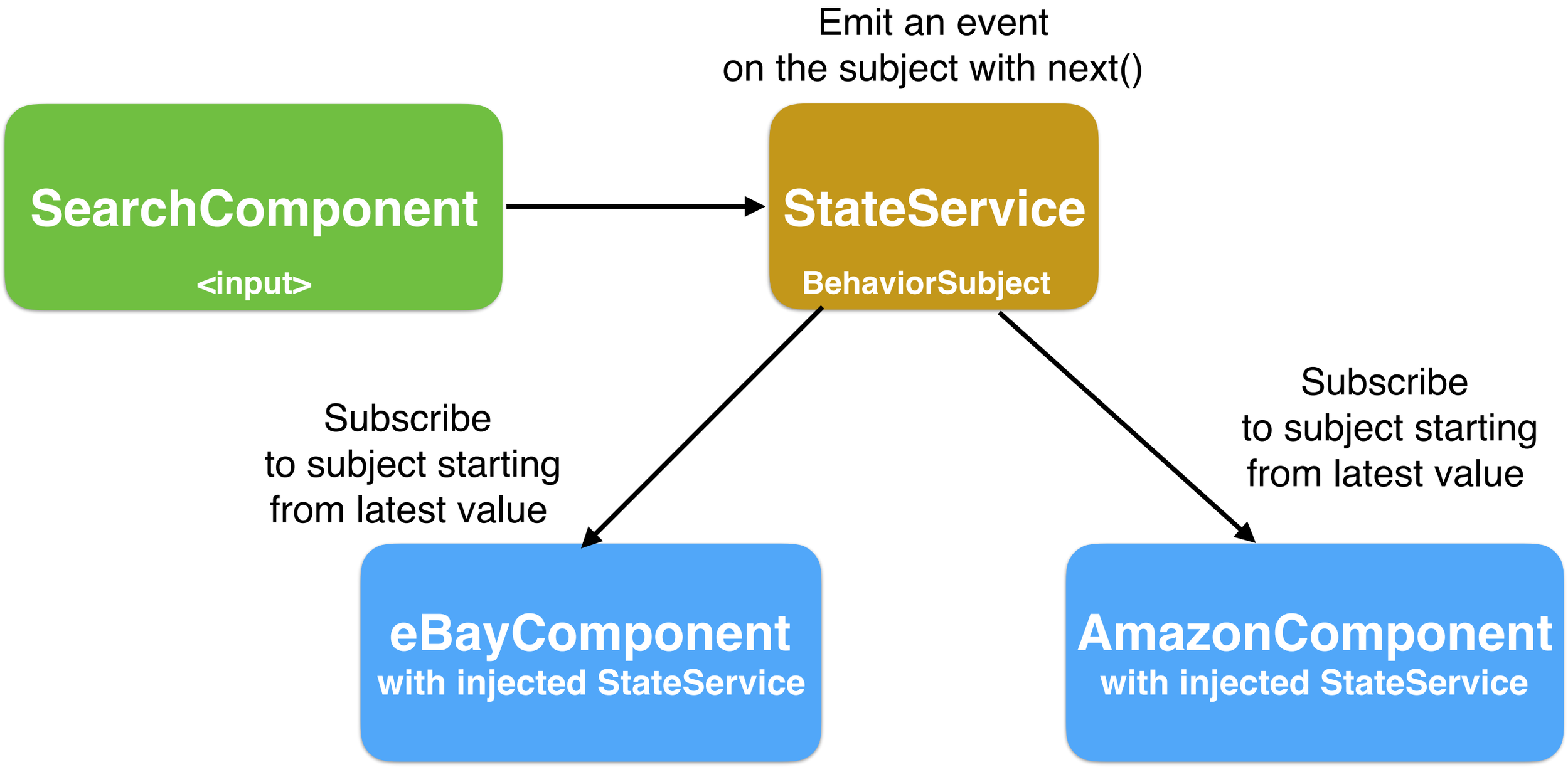

5 emits SomeEvent

6 and 8 subscribe
to SomeEvent

# BehaviorSubject

- Includes an observable and observer(s)
- Remembers the last emitted value
- Every subscriber gets the initial or the latest value emitted by the `BehaviorSubject`
- Examples: login status, state

# Inter-component communications with BehaviorSubject



```ts
export class StateService{

  stateEvent: BehaviorSubject<string> = new BehaviorSubject('');

  set searchCriteria(value: string) {

    this.stateEvent.next(value);
  }
}
```

```ts
export class SearchComponent {

  searchInput: FormControl;

  constructor(private state: StateService){
    this.searchInput = new FormControl('');

    this.searchInput.valueChanges
      .debounceTime(300)
      .subscribe(searchValue =>
          this.state.searchCriteria = searchValue);
  }
```

```ts
export class EbayComponent implements OnDestroy {

  searchFor: string;
  subscription: Subscription;

  constructor(private state: StateService){

    this.subscription = state.stateEvent
      .subscribe(event => this.searchFor = event);
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();  // a must
  }
}
```

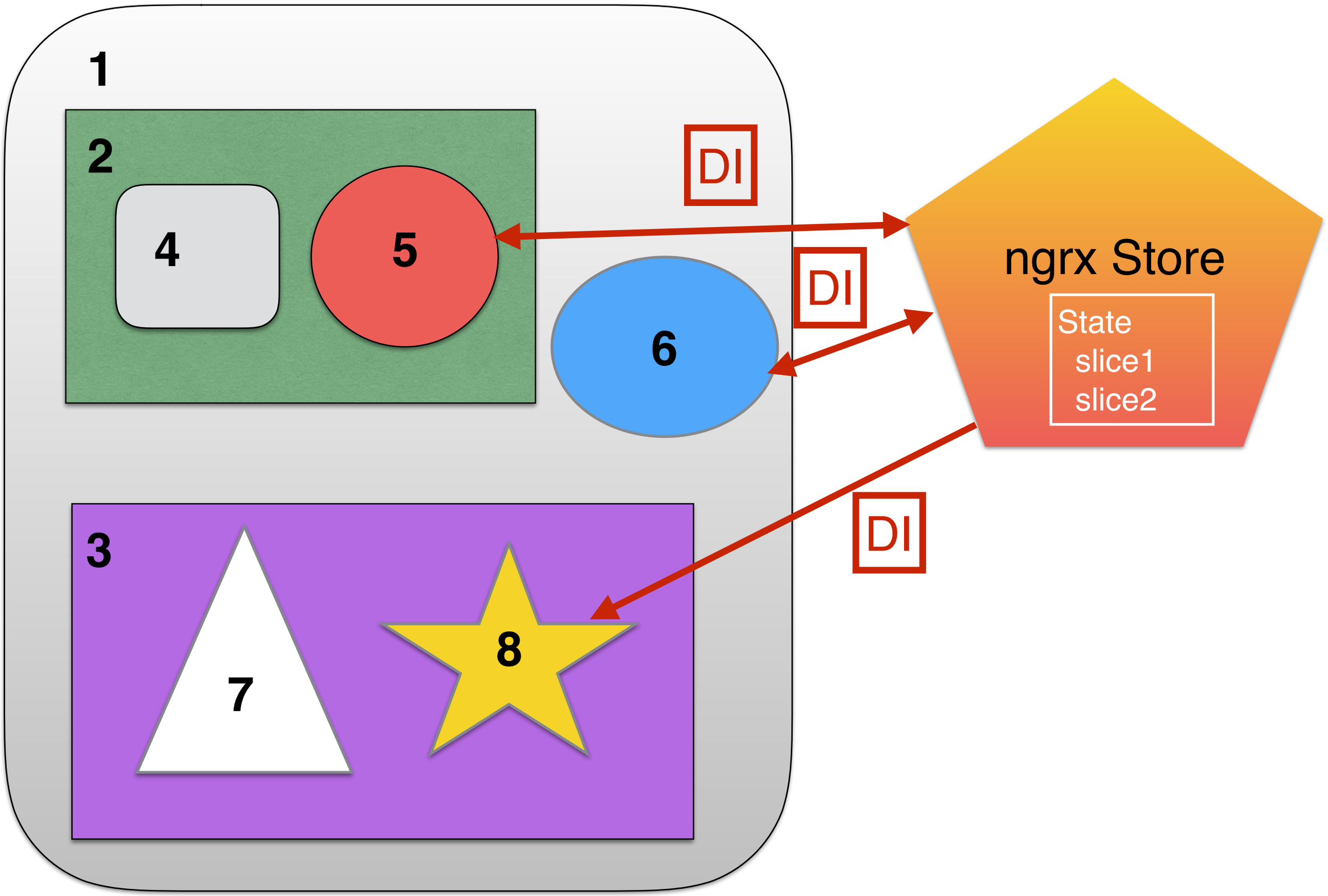
Whenever the `eBayComponent` subscribes,
the BehaviorSubject sends the latest value of the search criteria

# Demo

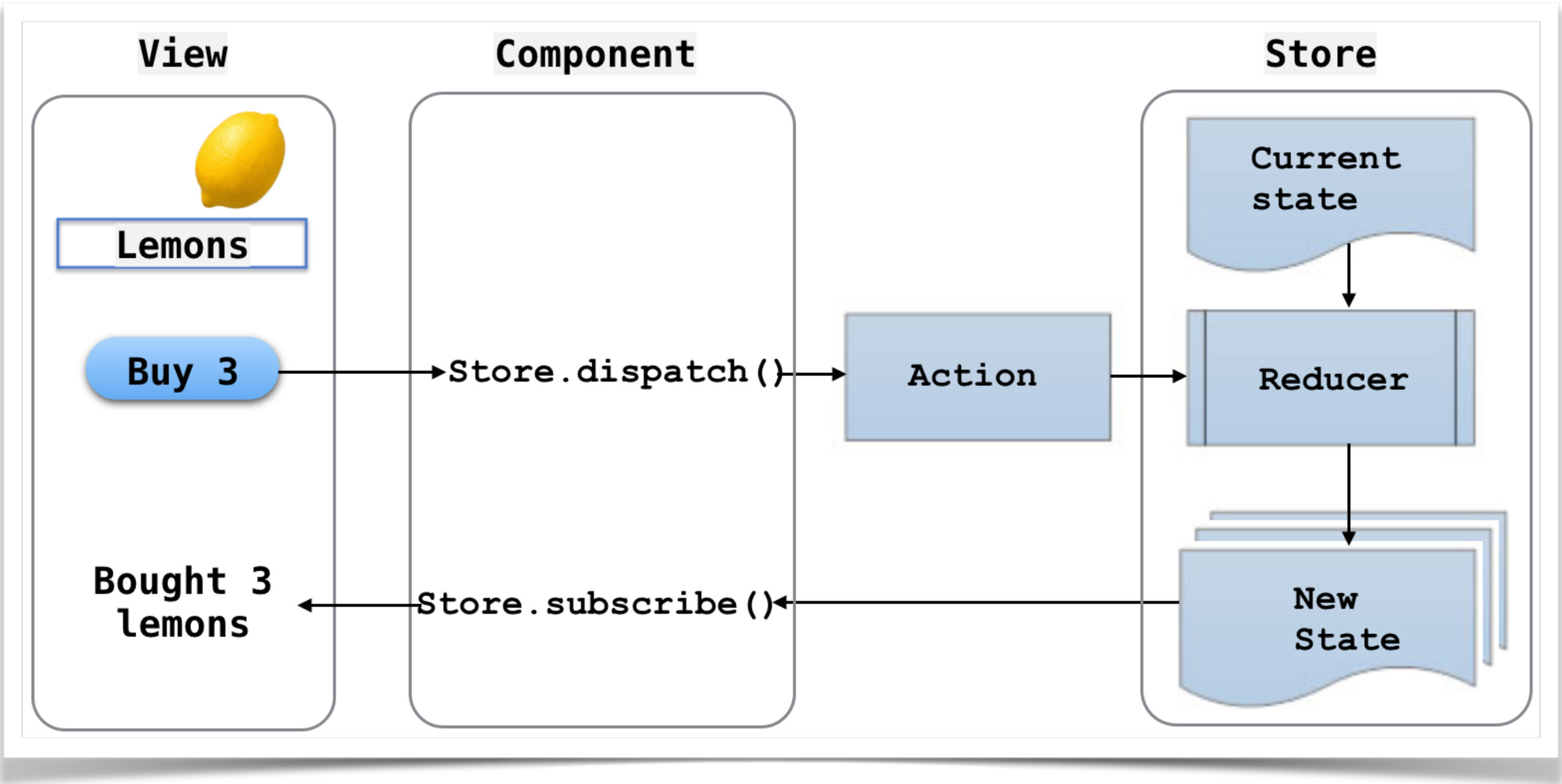
ng serve --app mediator-service

# ngrx: a state container and mediator

# A Store as a state container/mediator



# Redux data flow



# What's ngrx?

- A redux-inspired library for managing state in Angular apps

- ngrx is Redux married with RxJS

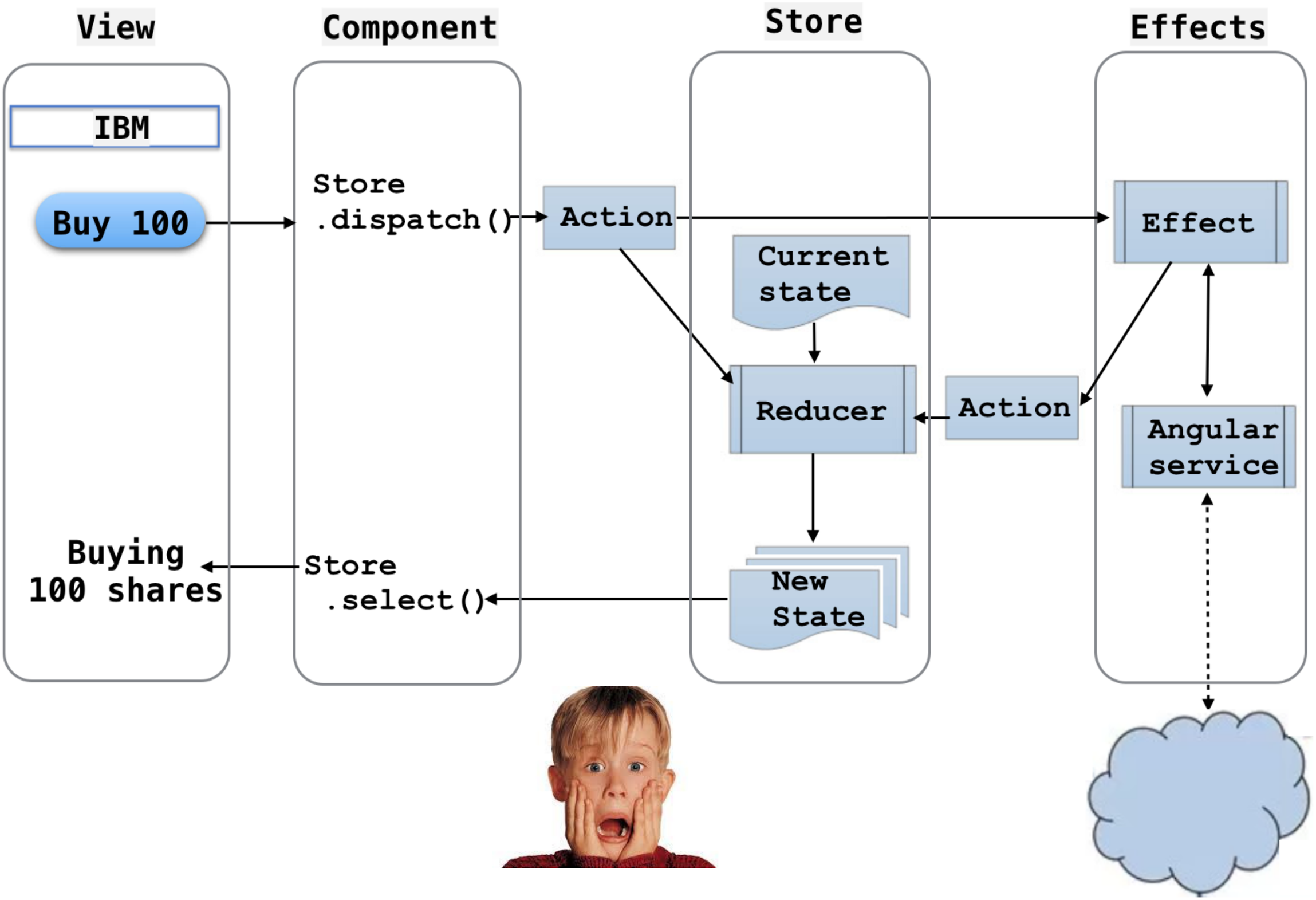

```
class Store<T> extends Observable<T> implements Observer<Action>
```

An alternative to ngrx is **ngxs**: https://github.com/ngxs

# ngrx principles

- A Store is a single source of truth
- State is read-only
- When state has to change, a new state is created with pure functions
- Communications with external parties is done in the *Effects* classes

# ngrx data flow



# Demo: a counter with ngrx store as a mediator

# Projecting HTML fragments to the templates of child components

# Projection

- Allows to change a template's content at runtime
- You can pass one or more HTML fragments to a child component
- If a child's template includes **<ng-content>**, it will be replaced by an HTML fragment given by the parent
- In AngularJS projection was called **transclusion**

# Basic projection

Child

```
@Component({
  selector: 'child',
  styles: ['.wrapper {background: lightgreen;}'],
  template: `
    <div class="wrapper">
     <h2>Child</h2>
      <div>This div is defined in the child's template</div>

     <ng-content></ng-content>      <---- insertion point

    </div>`,
  encapsulation: ViewEncapsulation.Native
})
class ChildComponent {}
```

# Basic projection

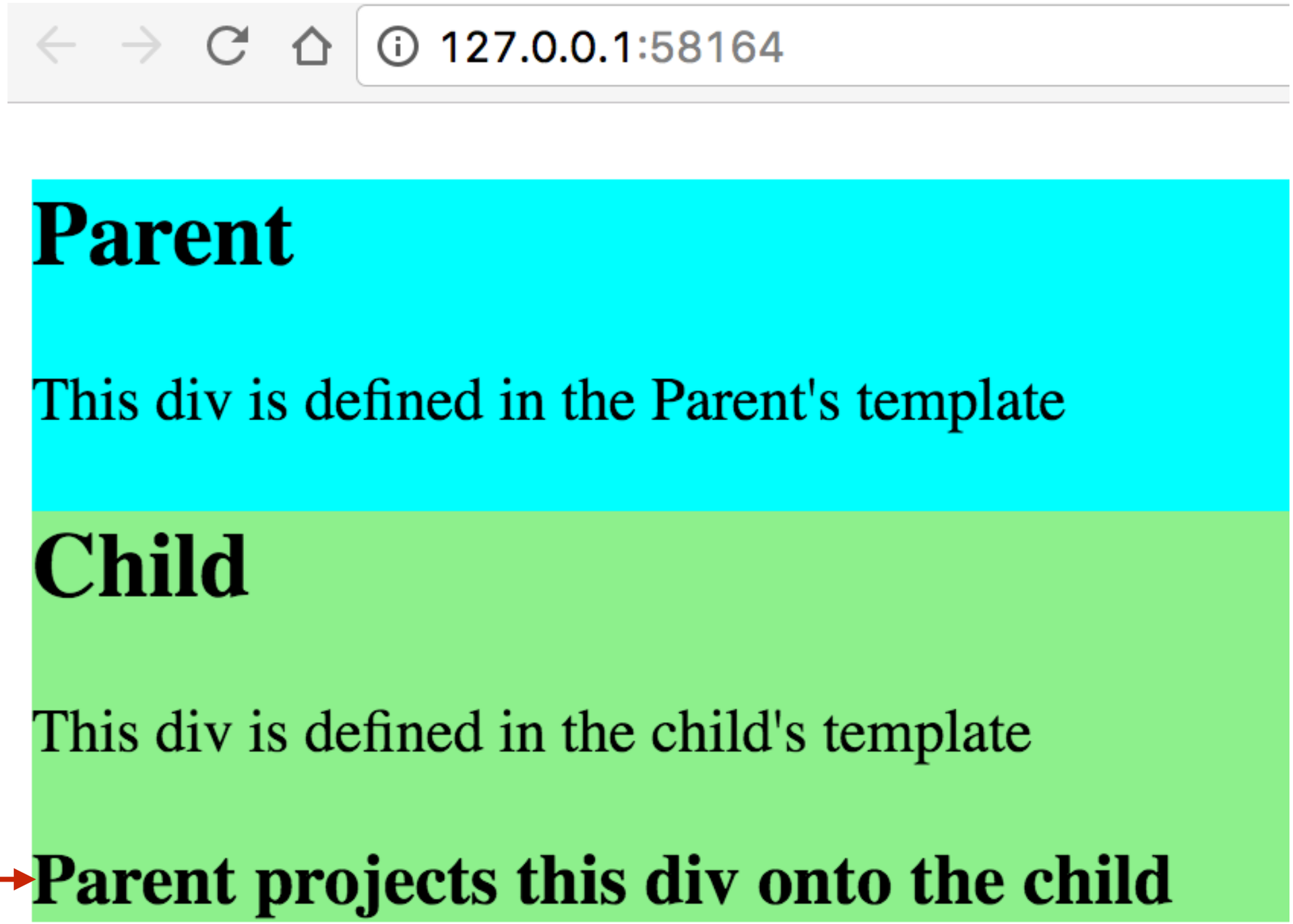
Child

```
@Component({
  selector: 'child',
  styles: ['.wrapper {background: lightgreen;}'],
  template: `
    <div class="wrapper">
     <h2>Child</h2>
      <div>This div is defined in the child's template</div>

     <ng-content></ng-content>      <---- insertion point

    </div>`,
  encapsulation: ViewEncapsulation.Native
})
class ChildComponent {}
```

Parent

```
@Component({
  selector: 'app',
  styles: ['.wrapper {background: cyan;}'],
  template: `
    <div class="wrapper">
     <h2>Parent</h2>
      <div>This div is defined in the Parent's template</div>
      <child>
        <div><h3>Parent projects this div onto the child </h3></div>
      </child>
    </div>
  `,
  encapsulation: ViewEncapsulation.Native
})
class AppComponent {}
```

# Projecting a  from parent to child



# Projecting into multiple areas

- A component can have more than one <ng-content> tag in its template
- The attribute `select` allows to distinguish <ng-content> areas

```
<ng-content select=".header" ></ng-content><p>

<div>This content is defined in child</div><p>

<ng-content select=".footer"></ng-content>
```

Parent

```
@Component({
  selector: 'app',
  styles: ['.wrapper {background: cyan;}'],
  template: `
    <div class="app">
     <h2>Parent</h2>
      <div>This div is defined in the Parent's template</div>
      <child>

        <div class="header" ><i>Child got this header from parent {{todaysDate}}</i></div>

        <div class="footer"><i>Child got this footer from parent</i></div>
      </child>
    </div>
  `
})
class AppComponent {
  todaysDate: string = new Date().toLocaleDateString();
}
```

Child

```
@Component({
  selector: 'child',
  styles: ['.wrapper {background: lightgreen;}'],
  template: `
    <div class="child">
     <h2>Child</h2>
      <ng-content select=".header"></ng-content><p>
      <div>This content is defined in child</div><p>
      <ng-content select=".footer"></ng-content>
    </div>
  `
})
class ChildComponent {}
```

# Shadow DOM Support

- Every web page is represented by a tree of DOM objects
- Shadow DOM allows to encapsulate a subtree of the HTML elements to create a boundary between them
- Such subtree is rendered as a part of the HTML document, but its elements are not attached to the main DOM tree
- With Shadow DOM, the CSS styles of the HTML elements won’t be merged with the main DOM CSS

# encapsulation property of @Component

```
encapsulation: ViewEncapsulation.None
```

- `ViewEncapsulation.Emulated` - Emulate encapsulation of the Shadow DOM (default)

- `ViewEncapsulation.Native` - Use the Shadow DOM natively supported by the browser

- `ViewEncapsulation.None` - Don’t use the Shadow DOM encapsulation

# ViewEncapsulation.Native

**Parent**

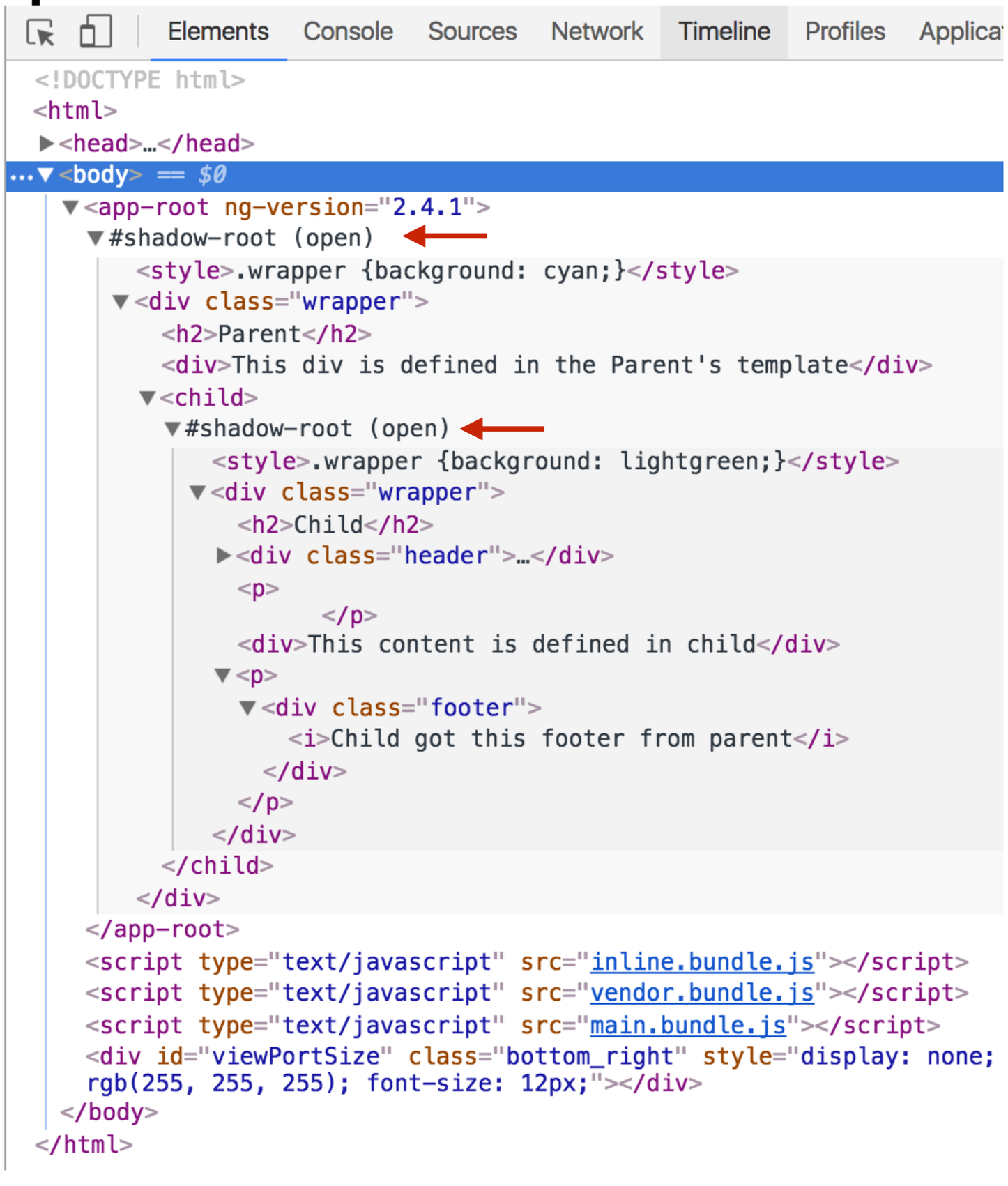
This div is defined in the Parent's template

**Child**

*Child got this header from parent 1/3/2017*

This content is defined in child

*Child got this footer from parent*



# ViewEncapsulation.Emulated

## Parent

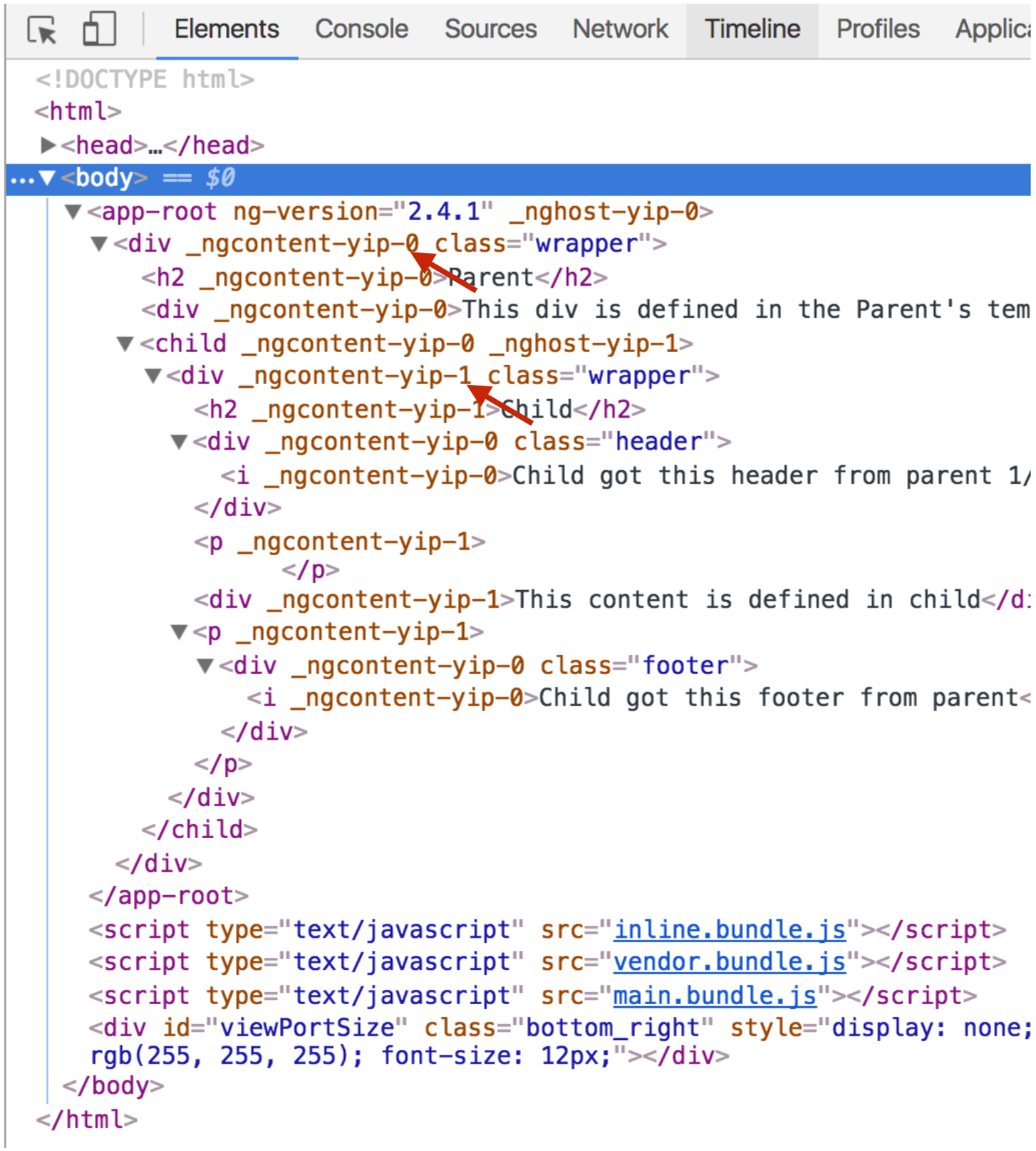
This div is defined in the Parent's template

## Child

*Child got this header from parent 1/3/2017*

This content is defined in child

*Child got this footer from parent*



# ViewEncapsulation.None

## Parent

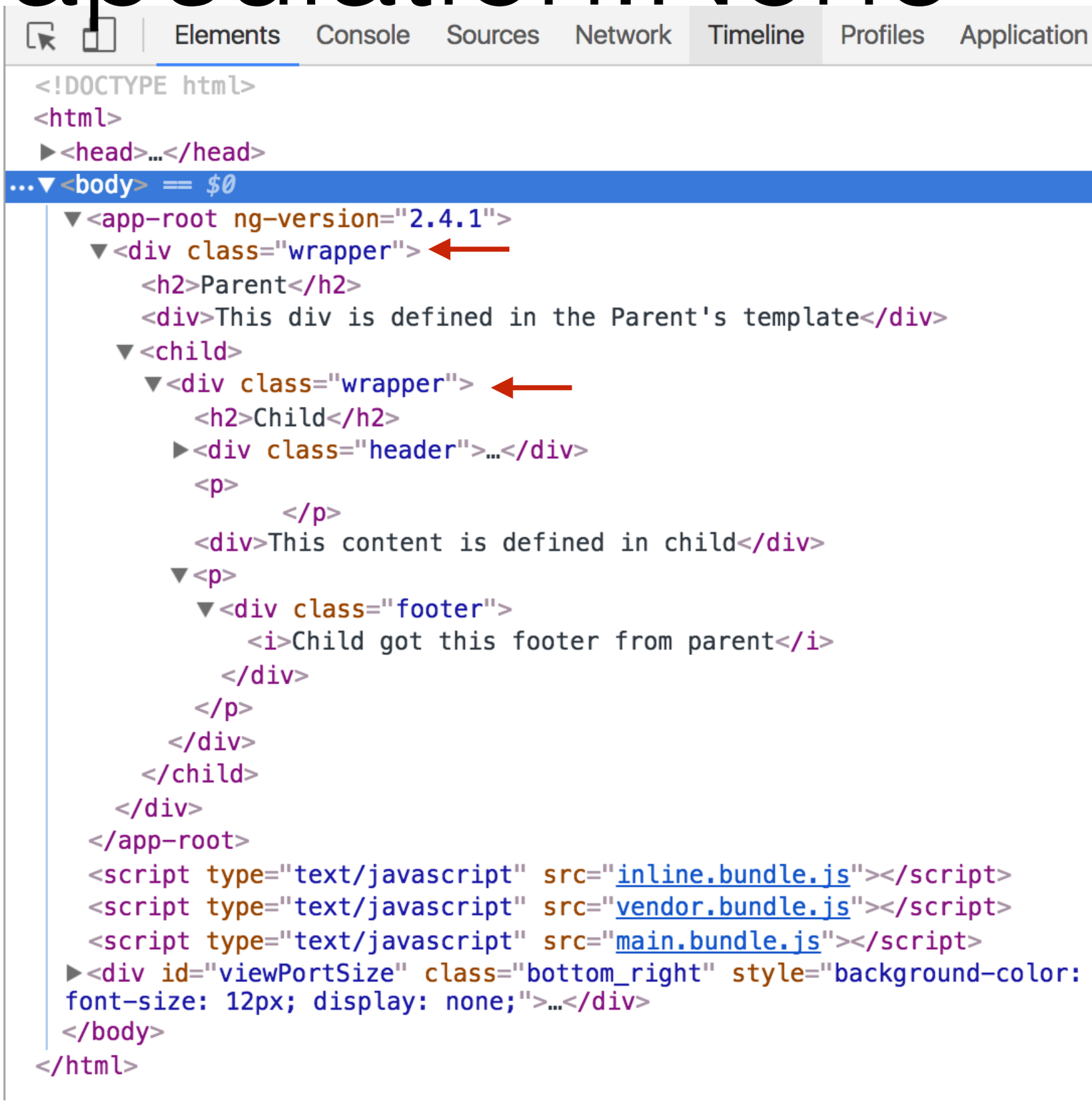
This div is defined in the Parent's template

## Child

*Child got this header from parent 1/3/2017*

This content is defined in child

*Child got this footer from parent*



# Demo: Projection

ng serve --app projection -o

# Passing params to routes

# Passing parameters to a route

```
const routes: Routes = [
    {path: '',         component: HomeComponent},
    {path: 'product/:id', component: ProductDetailComponentParam}
];

@Component({
    selector: 'app',
    template: `
        <a [routerLink]="['/']">Home</a>
        <a [routerLink]="['/product', productId]">Product Details</a>
        <router-outlet></router-outlet>
    `
})
class AppComponent {

    productId: number=1234;
}
```

# Receiving params in ActivatedRoute

- Inject `ActivatedRoute` into a component to receive the route params during navigation
- Use `ActivatedRoute.snapshot` to get params once
- Use `ActivatedRoute.paramMap.subscribe()` for receiving multiple params over time

# Receiving params in a route with a snapshot

```
@Component({
  selector: 'product',
  template: `<h1 class="product">Details for product {{productID}}</h1>`,
  styles: ['.product {background: cyan}']
})
export class ProductDetailComponentParam {
  productID: string;

  constructor(route: ActivatedRoute) {
    this.productID = route.snapshot.paramMap.get('id');
  }
}
```

# ActivatedRoute (fragment)

```
export declare class ActivatedRoute {
    /** An observable of the URL segments matched by this route */
    url: Observable<UrlSegment[]>;
    /** An observable of the matrix parameters scoped to this route */
    params: Observable<Params>;
    /** An observable of the query parameters shared by all the routes */
    queryParams: Observable<Params>;
    /** An observable of the URL fragment shared by all the routes */
    fragment: Observable<string>;
    /** An observable of the static and resolved data of this route. */
    data: Observable<Data>;
    /** The outlet name of the route. It's a constant */
    readonly paramMap: Observable<ParamMap>;
    readonly queryParamMap: Observable<ParamMap>;
    toString(): string;
    ...
}
```

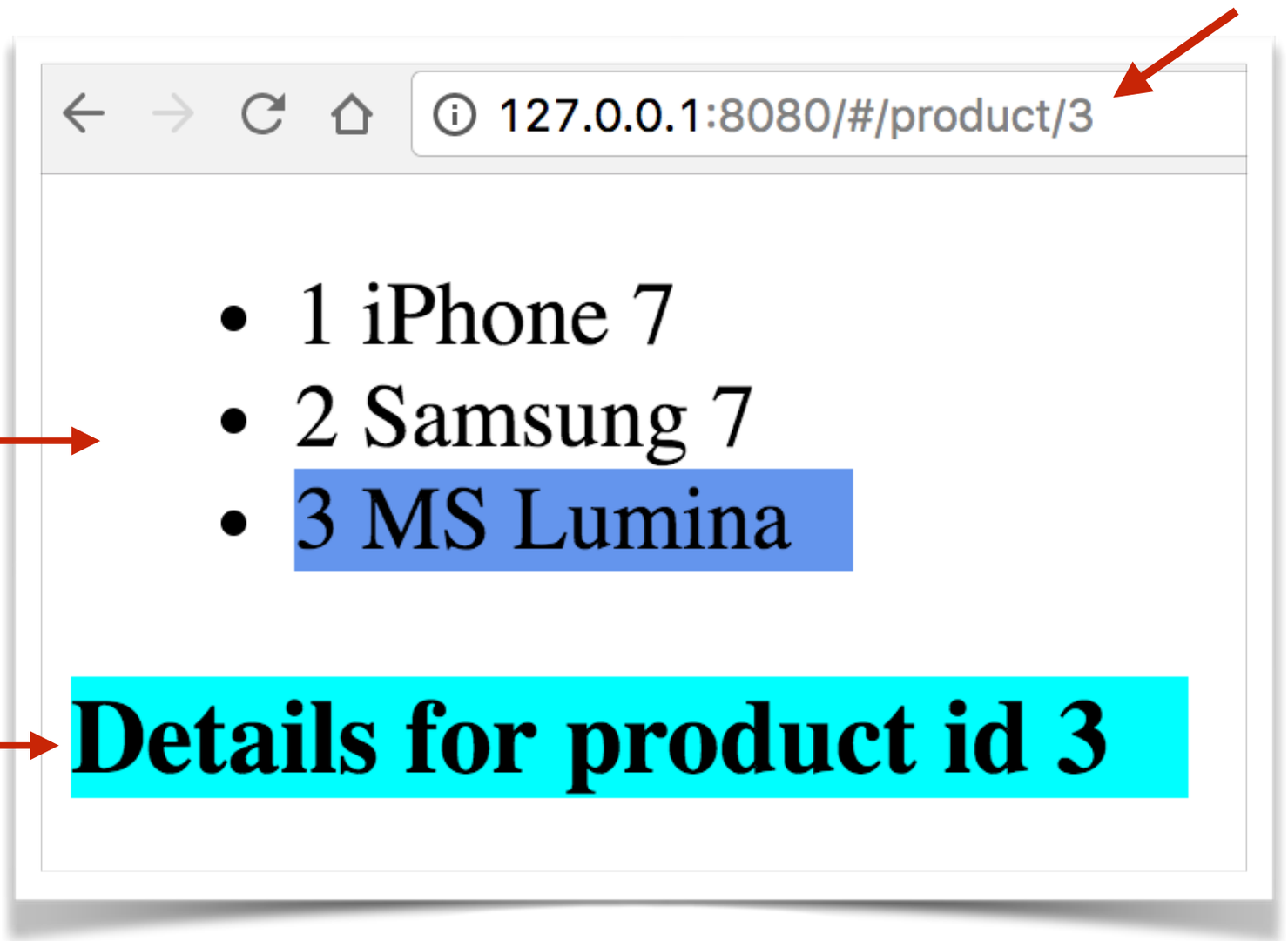
# Receiving params in a route by subscription

```
@Component({
  selector: 'product',
  template: `<h1 class="product">Details for product {{productID}}</h1>`,
  styles: ['.product {background: cyan}']
})
export class ProductDetailComponentParam {
  productID: string;

  constructor(route: ActivatedRoute) {
    // this.productID = route.snapshot.paramMap.get('id');

    route.paramMap.subscribe(
                    params => this.productID = params.get('id');
     );
  }
}
```

# Master-Detail with Router

Parent component

<router-outlet>



```
const routes: Routes = [
    {path: 'productDetail/:id', component: ProductDetailComponentParam}
];
…

class AppComponent {
  …
    constructor(private _router: Router){}

    onSelect(prod: Product): void {
         this._router.navigate(["/productDetail", prod.id]);
    }
}
```

```
export class ProductDetailComponentParam {
  productID: number;

  constructor(private route: ActivatedRoute) {

  this.route.paramMap
    .subscribe(
        params => this.productID = params.get('id')
  );
}

}
```

# Demo

ng serve --app router -o

# What have we learned

- Component can receive data from their parents via `@Input` properties

- Components can send data to their parents by emitting events via `@Output` properties

- A mediator design pattern is used for arranging inter-component communication in a loosely-coupled manner

- Using injectable services offers the most flexible way of inter-component communications

- A state-management library can serve as a mediator

- You can pass html fragments from one component to another

# Thank you!

- Code samples: https://bit.ly/2r8L71N
- Video “When ngrx is an overkill”: https://www.youtube.com/watch?v=xLTIDs0CDCM
- email: yfain@faratasystems.com
- Blog: yakovfain.com
- Twitter: @yfain