



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE

# Angular Components

## Data-binding & Lifecycle Hooks

Alex Thalhammer

# Outline

- Take a closer look on data binding
  - Property binding with @Input()
  - Event binding with @Output()
  - Use component bindings
- Component Lifecycle Hooks
- Smart vs dumb components



# Data binding



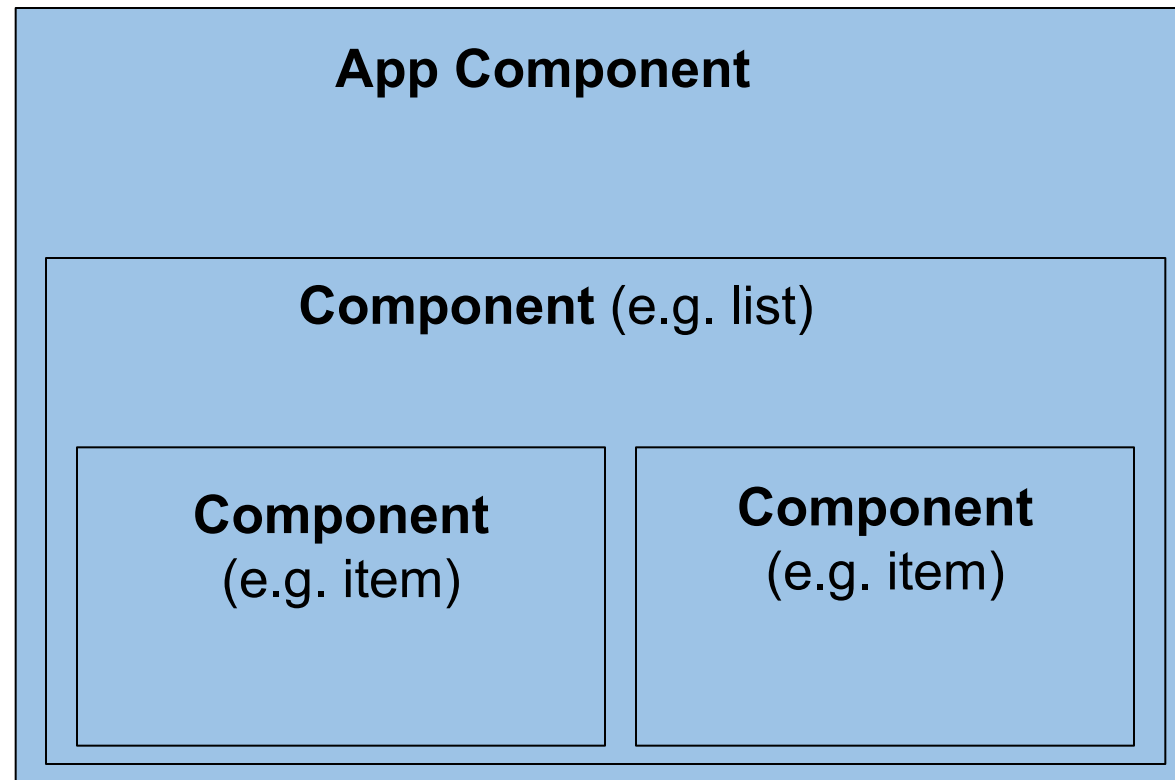
Performance

Components

Predictability

Architecture goals in Angular

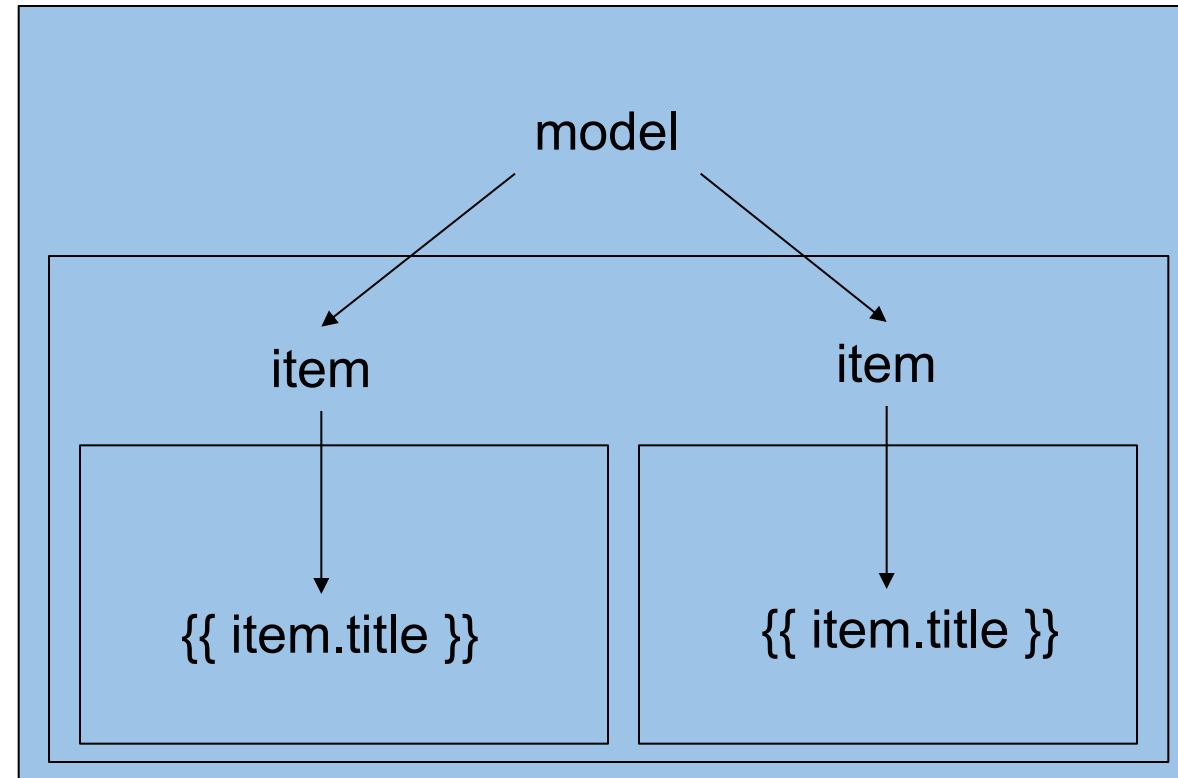
# Component tree in Angular 2+



# Rules for property binding

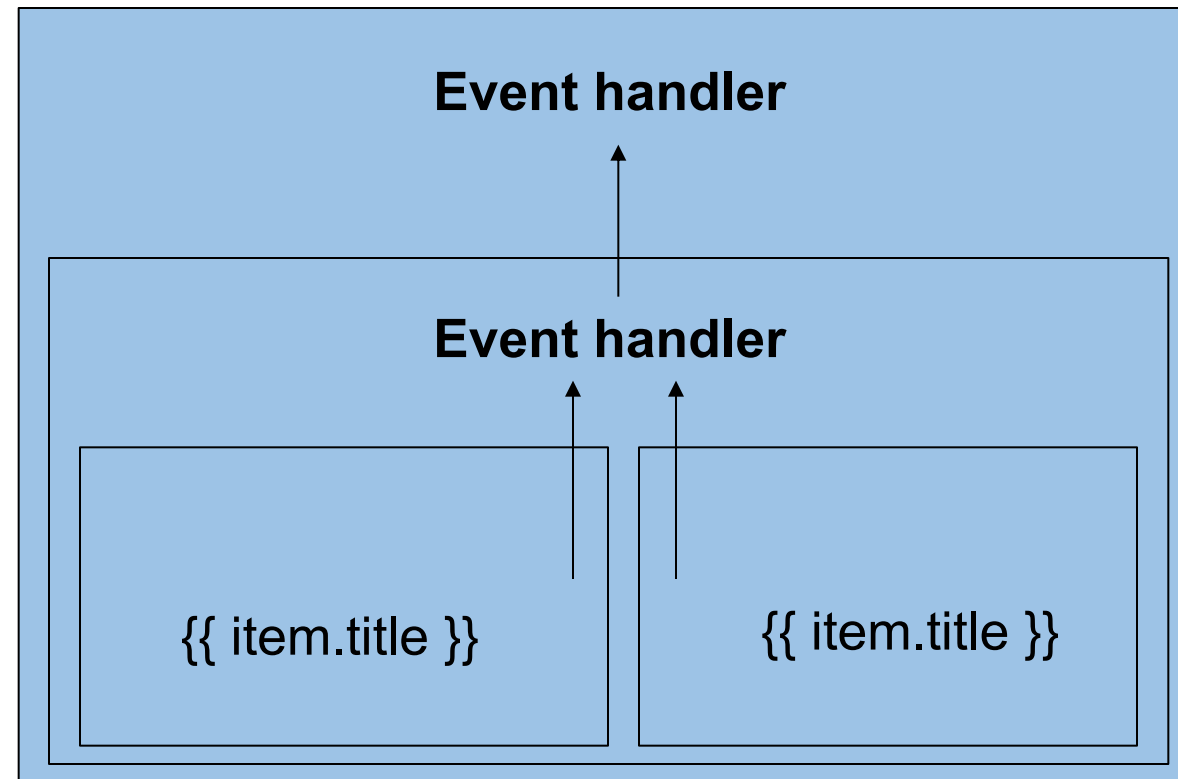
- Data can only be passed from top to bottom (top/down)
  - Parent can pass data to children
  - Children cannot pass data to parent (we need events for that)
- Dependency graph is a tree
- Angular just takes a digest to compare tree with the browser DOM

# Property binding



[<http://victorsavkin.com/post/110170125256/change-detection-in-angular-2>]

# Event bindings (one way, bottom/up)

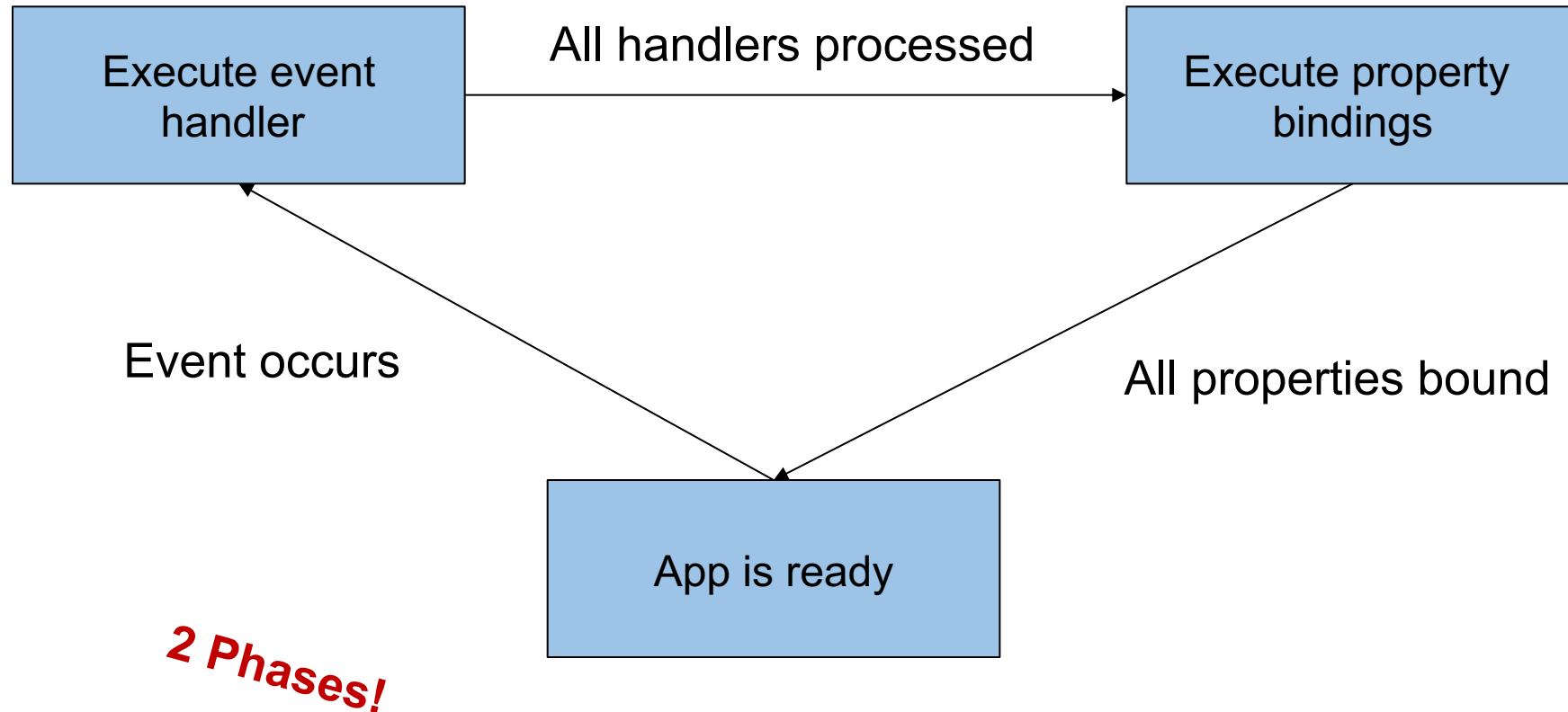




# Event bindings (one way, bottom/up)

- No digest necessary to send events
- But: Events can trigger data change → Property Binding

# Property and event bindings



# View

```
<button [disabled]="!from || !to" (click)="search()">  
  Search  
</button>
```

```
<table>  
  <tr *ngFor="let flight of flights">  
    <td>{{flight.id}}</td>  
    <td>{{flight.date}}</td> ← - - - - - > <td [text-content]="flight.date"></td>  
    <td>{{flight.from}}</td>  
    <td>{{flight.to}}</td>  
    <td><a href="#" (click)="selectFlight(flight)">Select</a></td>  
  </tr>  
</table>
```



# Recap

- Property binding: one way; top/down
- Event binding: one way; bottom/up
- Two way bindings?
- Two way = property binding + event binding



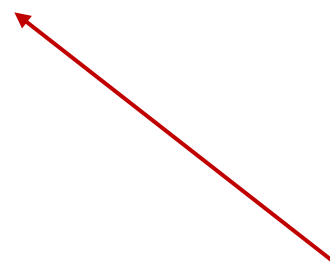
# Property + event binding

```
<input [ngModel]="from" (ngModelChange)="update($event)">
```



# Property + event binding

`<input [ngModel]="from" (ngModelChange)="from = $event">`



Property + *Change*

`<input [(ngModel)]="from">`



Changed value

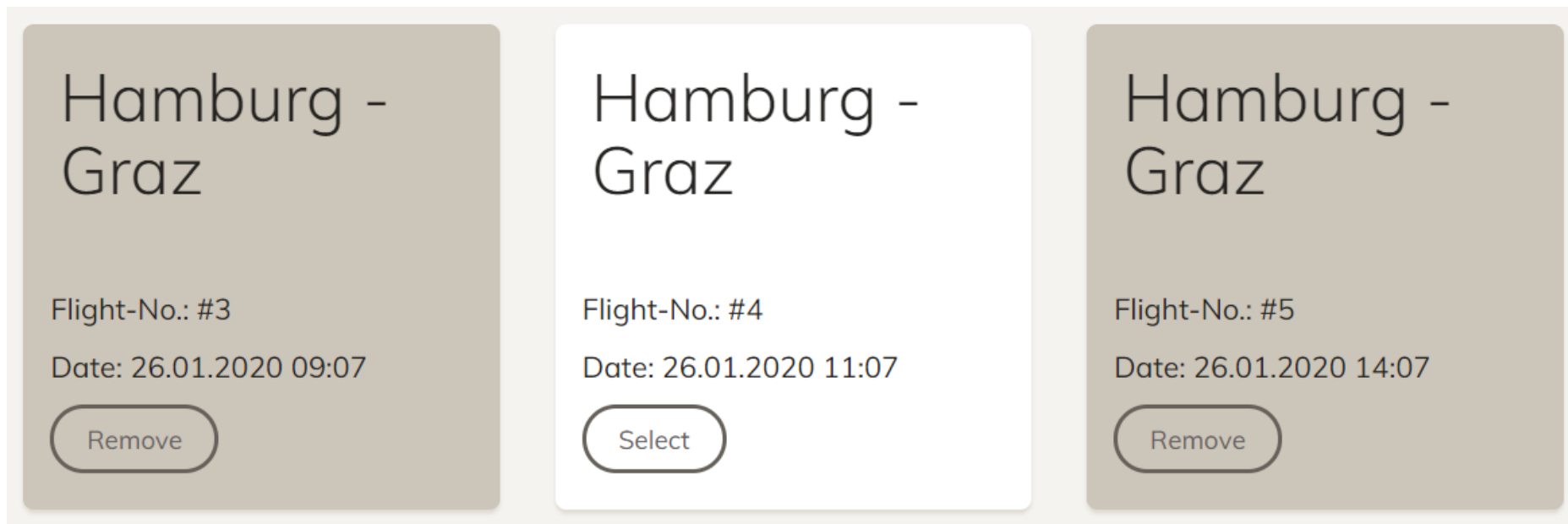






# Components data binding

## Example: flight-card



# Example: flight-card

Hamburg -  
Graz

Flight-No.: #3  
Date: 26.01.2020 09:07

Remove

Hamburg -  
Graz

Flight-No.: #4  
Date: 26.01.2020 11:07

Select

Hamburg -  
Graz

Flight-No.: #5  
Date: 26.01.2020 14:07

Remove

Basket:  
-----  
{  
 "3": true,  
 "4": false,  
 "5": true  
}

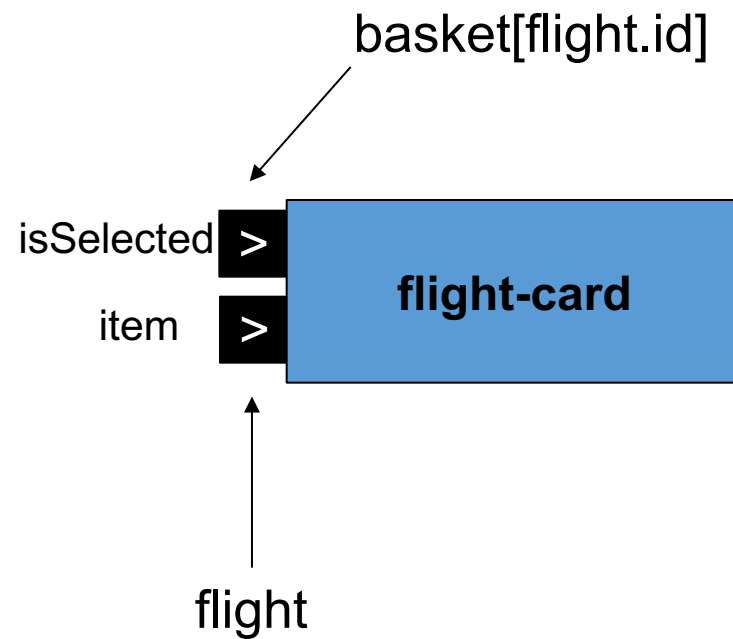
public basket = {};  
[...]  
basket[3] = true;  
basket[4] = false;  
basket[5] = true;

# Example: flight-card in flight-search.html

```
<div *ngFor="let f of flights">  
  
  <flight-card [item]="f" [isSelected]="basket[f.id]">  
  </flight-card>  
  
</div>
```



# flight-card



# Example: flight-card

```
@Component({  
  selector: 'flight-card',  
  templateUrl: './flight-card.component.html'  
})  
export class FlightCard {  
  
  [...]  
  
}
```





# Example: flight-card

```
export class FlightCard {  
  
    @Input() item: Flight;  
    @Input() isSelected: boolean;  
  
    select(): void {  
        this.isSelected = true;  
    }  
  
    deselect(): void {  
        this.isSelected = false;  
    }  
}
```



# Template

```
<div style="padding:20px;"
  [class.is-selected]="isSelected">

  <h2>{{item.from}} - {{item.to}}</h2>
  <p>Flightnr. #{{item.id}}</p>
  <p>Date: {{item.date | date:'dd.MM.yyyy'}}</p>

  <p>
    <button *ngIf="!isSelected" (click)="select()">Add</button>
    <button *ngIf="isSelected" (click)="deselect()">Remove</button>
  </p>
</div>
```



# Register component

```
@NgModule({
  imports: [
    CommonModule, FormsModule, SharedModule
  ],
  declarations: [
    FlightSearchComponent, FlightCardComponent
  ],
  exports: [
    FlightSearchComponent
  ]
})
export class FlightBookingModule{}
```



# DEMO





# Event bindings

# Example: flight-card event *isSelectedChange*

```
<div *ngFor="let f of flights">

  <flight-card [item]="f"
               [isSelected]="basket[f.id]"
               (isSelectedChange)="basket[f.id] = $event">

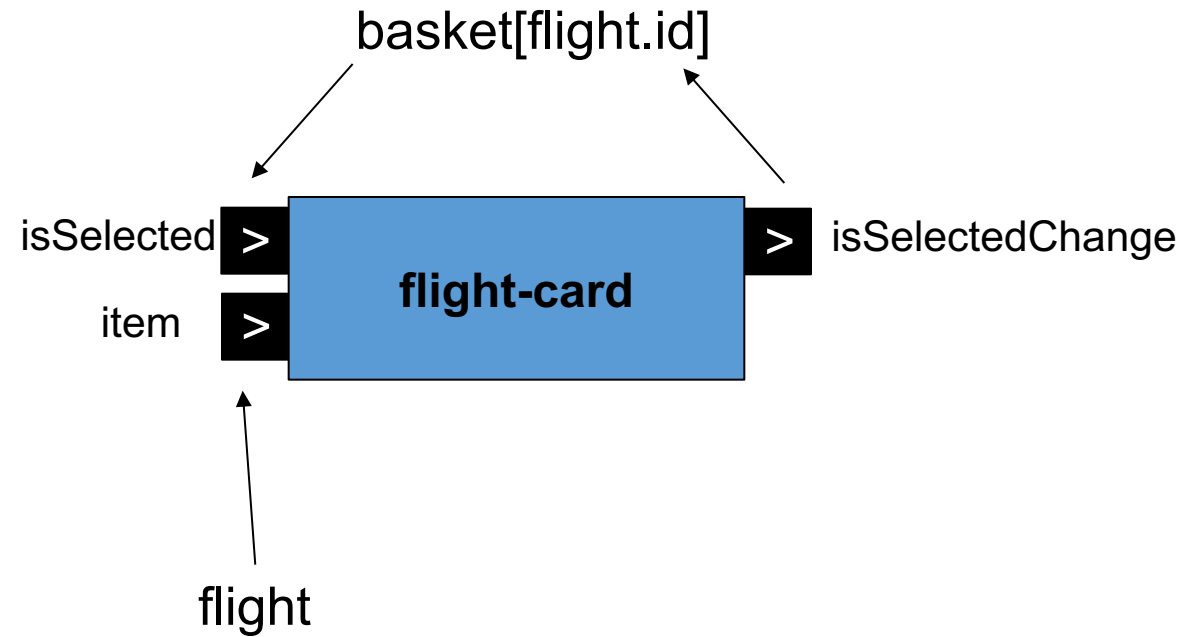
  </flight-card>

</div>
```





# flight-card



# Example: flight-ca

```
export class FlightCard {
```

```
  @Input() item: Flight;
```

```
  @Input() isSelected: boolean;
```

```
  @Output() isSelectedChange = new EventEmitter<boolean>();
```

```
  select() {
    this.isSelected = true;
    this.isSelectedChange.emit(this.isSelected);
  }
```

```
  deselect() {
    this.isSelected = false;
    this.isSelectedChange.emit(this.isSelected);
  }
```

```
}
```

```
<div *ngFor="let f of flights">
  <flight-card [item]="f"
    [isSelected]="basket[f.id]"
    (isSelectedChange)="basket[f.id] = $event">
  </flight-card>
</div>
```

# DEMO

# LAB



# Lifecycle Hooks



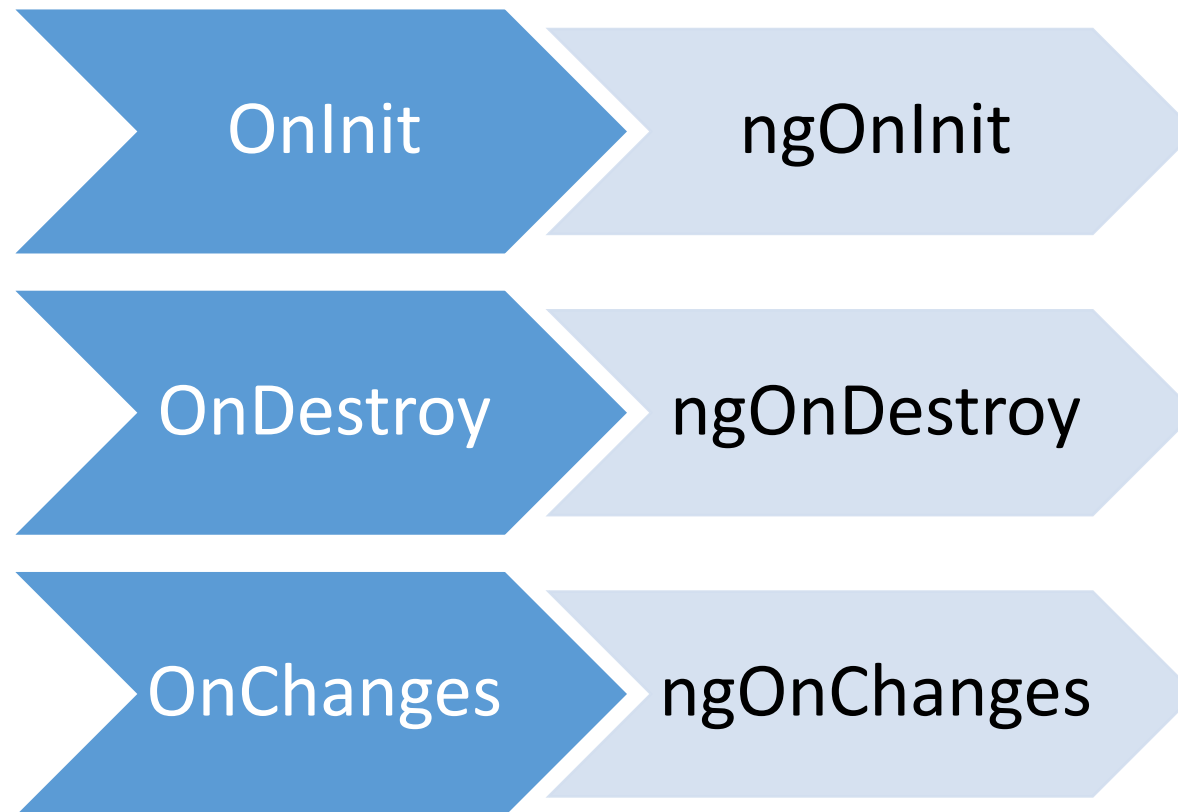
# What are Lifecycle Hooks?

- Built in methods in our components & directives
- Will be called at a certain time by Angular

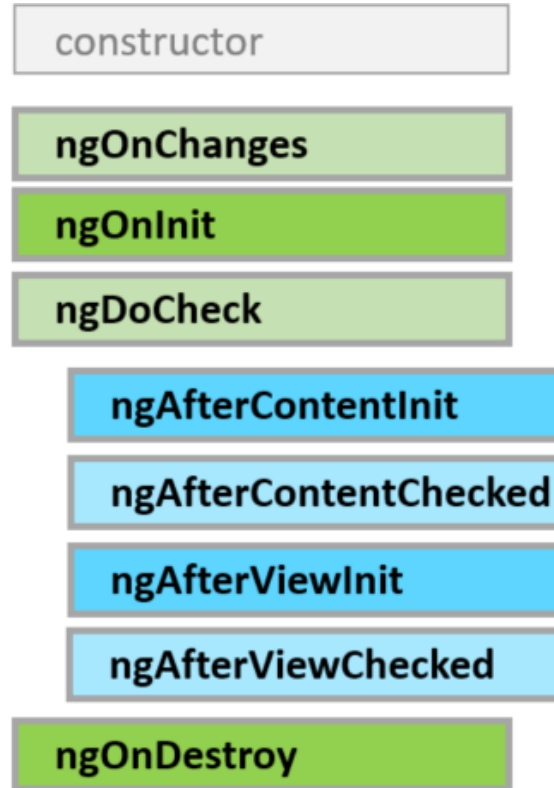




# Lifecycle Hooks (selection)



# Lifecycle Hooks (all, in order)



# Usage

```
@Component({  
  selector: 'my-component',  
  [...]  
})  
export class Component implements OnChanges, OnInit {  
  
  @Input() someData;  
  
  ngOnChanges(changes: SimpleChanges): void {  
    [...]  
  }  
  
  ngOnInit(): void {  
    [...]  
  }  
}
```



# DEMO



# Thought experiment

- What if <flight-card> would handle use case logic?
  - e.g. communicate with API
- Number of requests ==> Performance?
- Traceability?
- Reusability?



# Smart vs. Dumb Components

## Smart Component

- Use Case controller
- Container

## Dumb

- Independent of Use Case
- Reusable
- Leave

