Angular Components

Data-binding & Lifecycle Hooks

Alex Thalhammer

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

Outline

- Take a closer look on data binding
    - Property binding with @Input()
    - Event binding with @Output()
    - Two-way bindings

- View vs Content
    - ng content projection

- Component Lifecycle Hooks

- Smart vs dumb components

Data binding

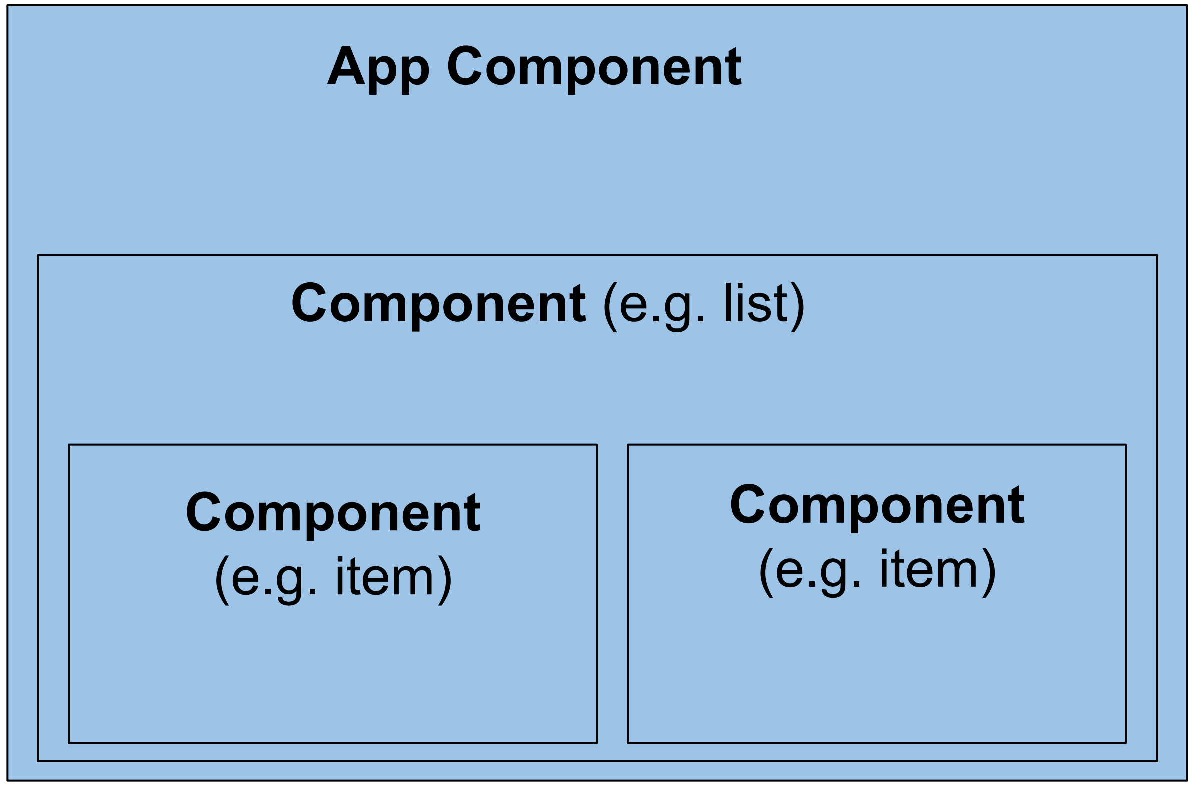
Performance

Components

Predictability

Architecture goals in Angular

Component tree in Angular 2+


App Component
Component (e.g. list)
Component (e.g. item)
Component (e.g. item)

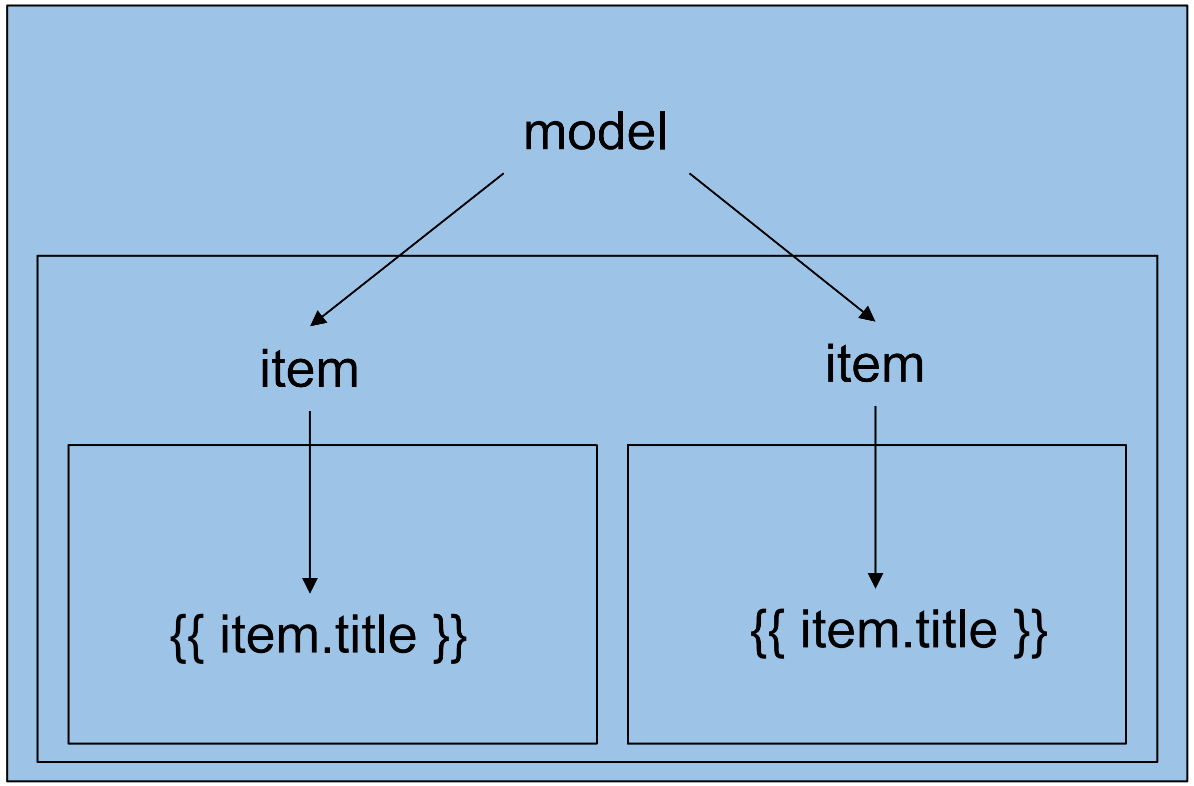
Rules for property binding []

- 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

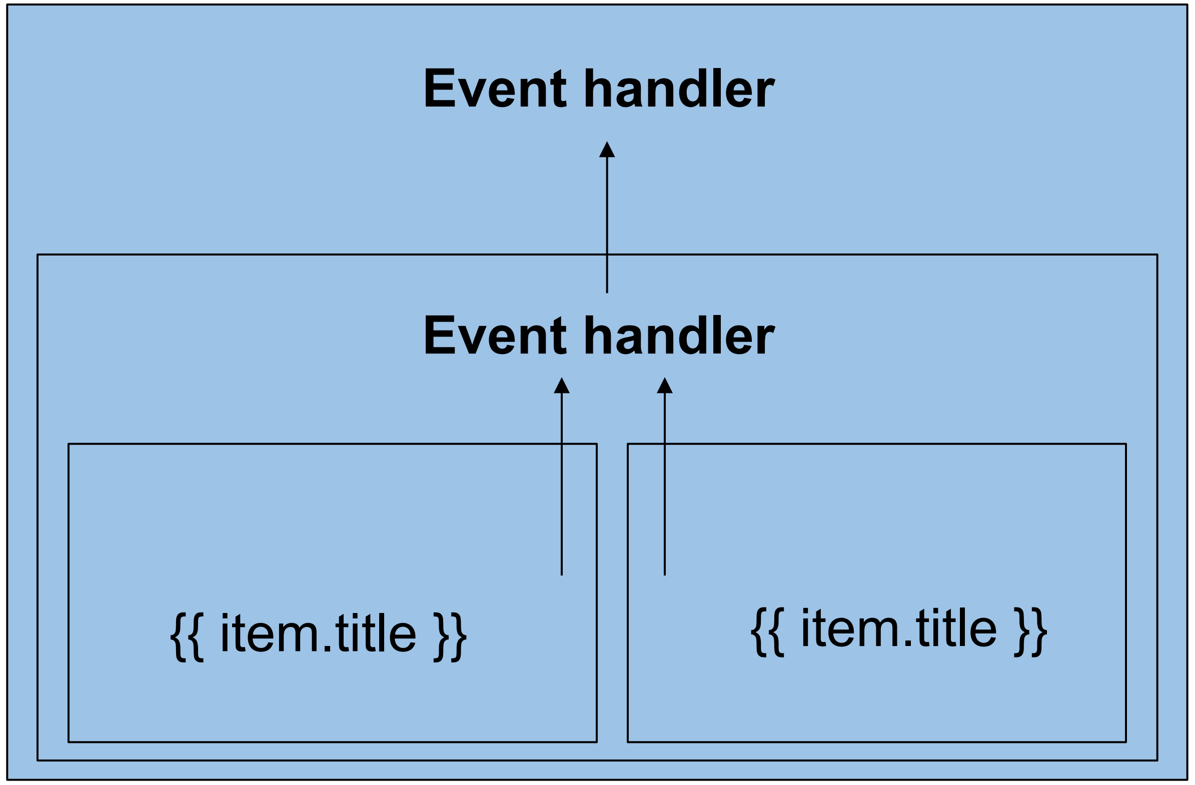
- Angular just takes a digest to compare tree with the browser DOM

Property binding []



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

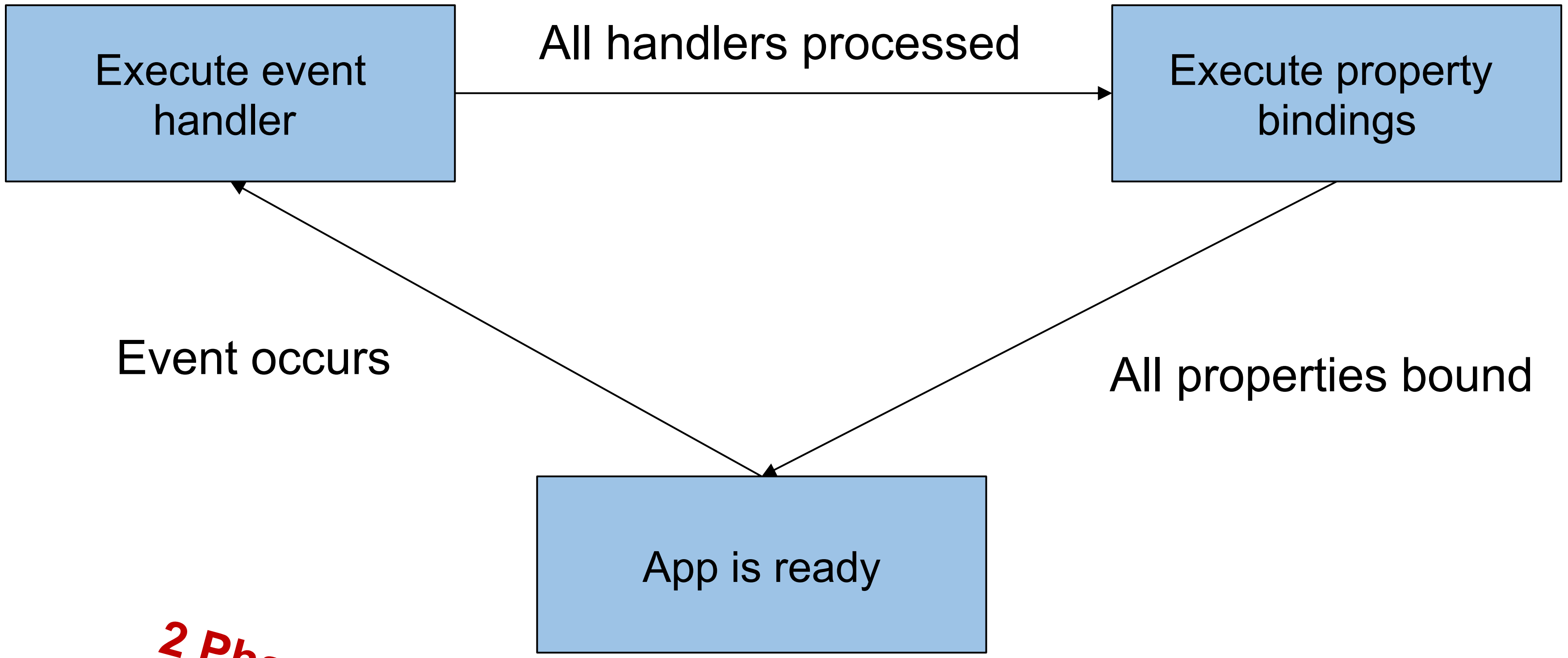
Event bindings (one way, bottom/up)



Event bindings (one way, bottom/up)

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

Property and event bindings



2 Phases!

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>{{ flight.from }}</td>
        <td>{{ flight.to }}</td>
        <td><a href="#" (click)="selectFlight(flight)">Select</a></td>
    </tr>
</table>
```

```
<td [text-content]="flight.date"></td>
```

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">
```

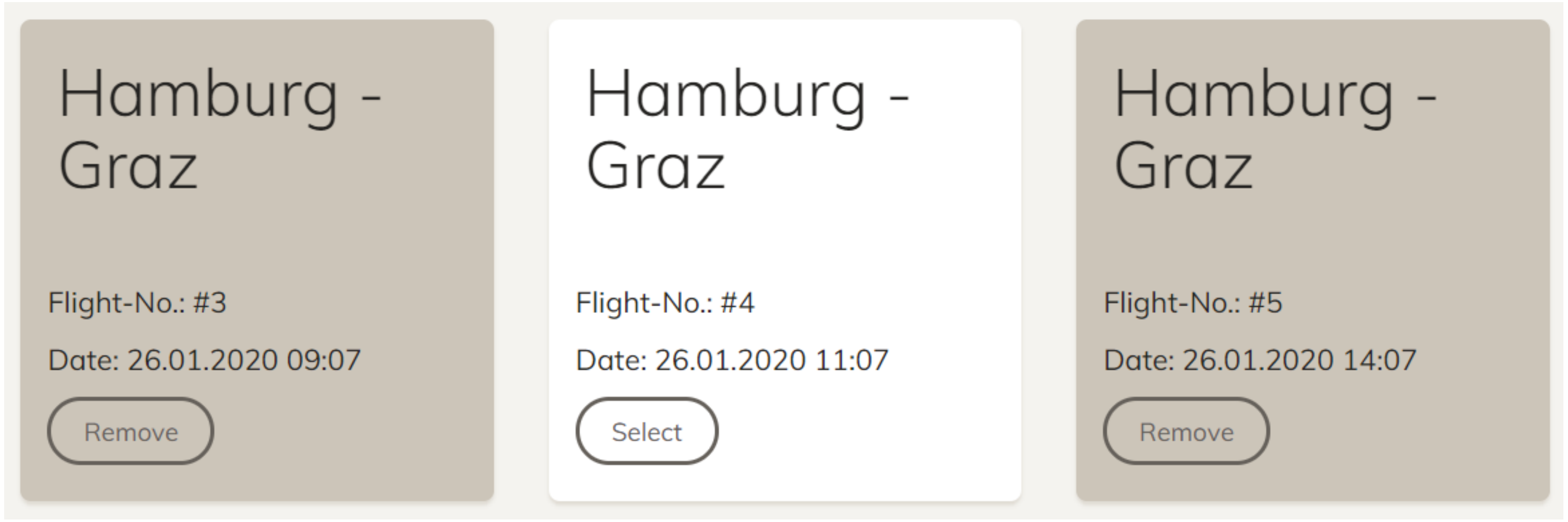
Property + *Change*

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

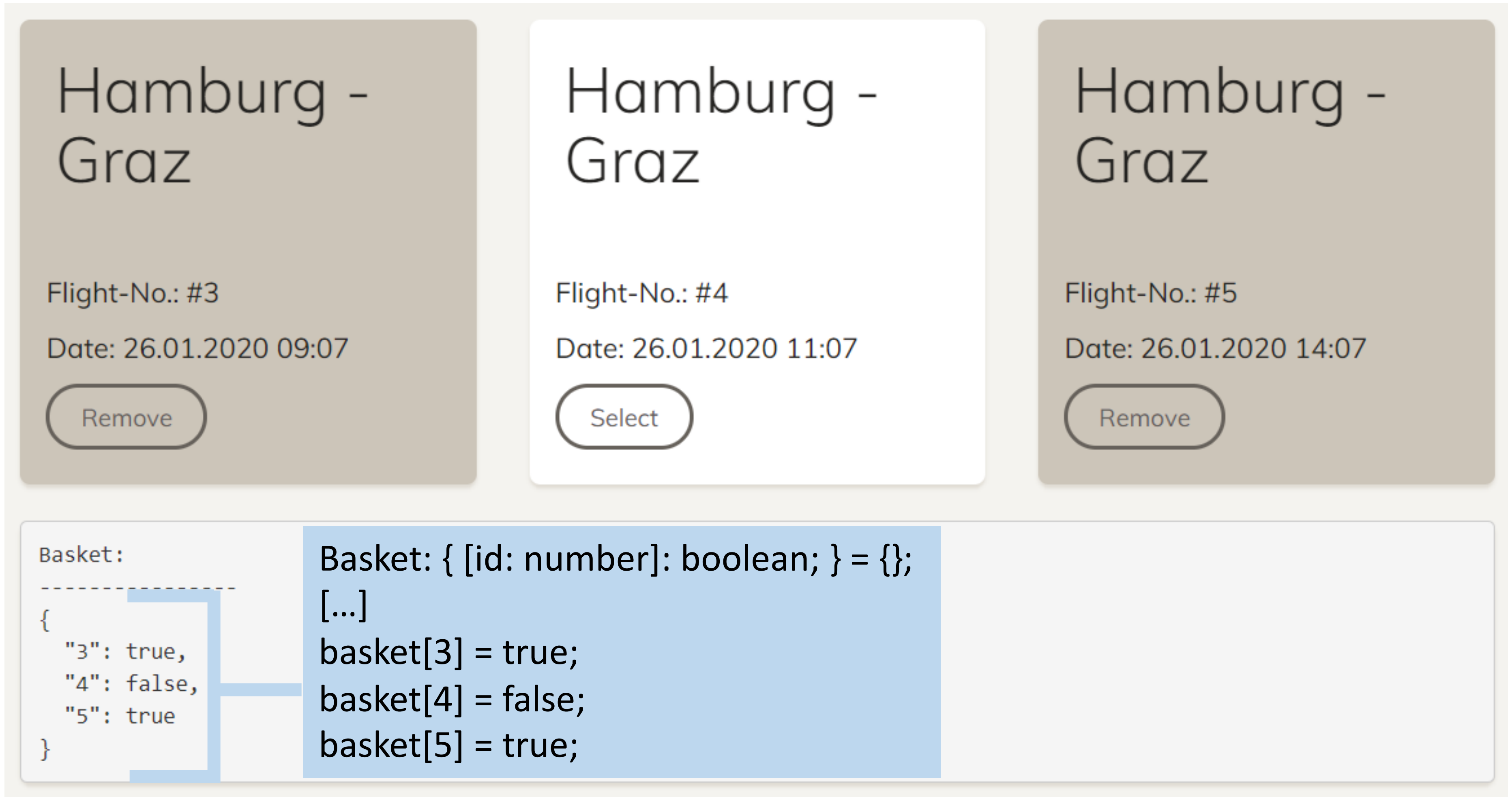
Changed value

Components data bindung

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

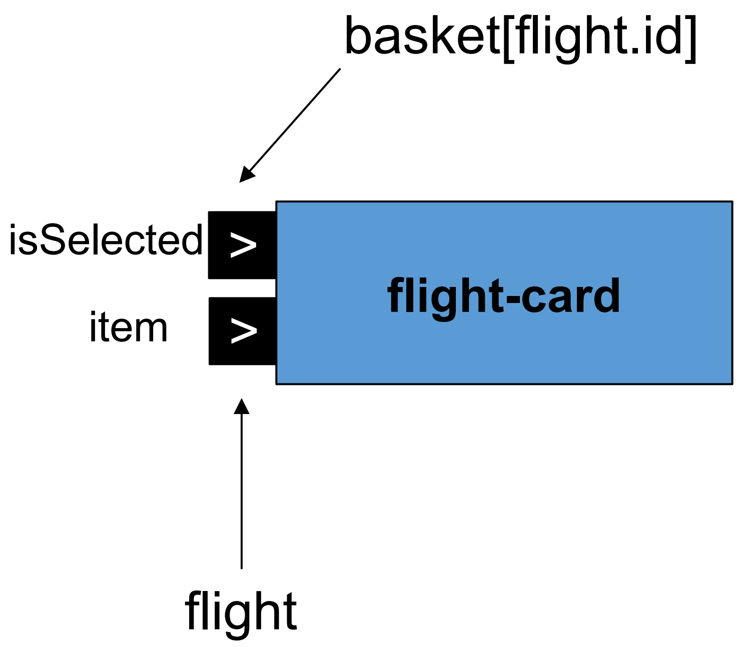
```
Basket:
----------------
{
  "3": true,
  "4": false,
  "5": true
}
```

```
Basket: { [id: number]: boolean; } = {};
[...]
basket[3] = true;
basket[4] = false;
basket[5] = true;
```

Example: flight-card in flight-search.html

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

flight-card



Example: flight-card

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

      […]

}
```

Example: flight-card

```
export class FlightCard {
      @Input({ required: true }) item!: Flight;
      @Input() isSelected = false;

      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()">Select</button>
        <button *ngIf="isSelected" (click)="deselect()">Deselect</button>
    </p>
</div>
```

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 (click)="isSelected ? deselect() : select()">
            {{ isSelected ? 'Deselect' : 'Select' }}
        </button>
    </p>
</div>
```

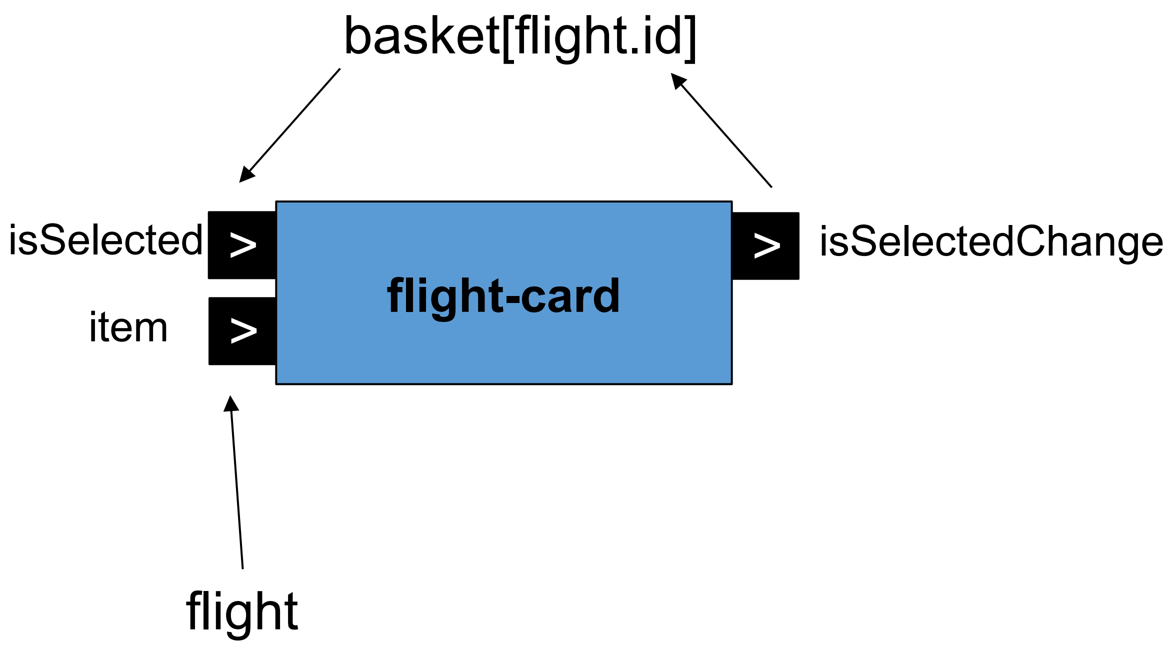
Register component

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

DEMO

Event bindings

flight-card


basket[flight.id]
isSelected
item
flight-card
isSelectedChange
flight

Example: flight-card event *isSelectedChange*

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

Example: flight-ca

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

```
export class FlightCard {
       @Input({ required: true
       @Input() isSelected = fa
       @Output() isSelectedChan

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

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

Example: flight-card event *two-way binding*

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

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

DEMO

View vs. Content

ANGULAR ARCHITECTS
INSIDE KNOWLEDGE

View vs. Content

```
@Component({
selector: 'tab',
template: `
    <div *ngIf="visible">
        <h1>{{title}}</h1>
        <div>
            <ng-content></ng-content>
        </div>
    </div>
`
})
export class TabComponent {
    @Input() title = '';
    protected visible = true;
}
```

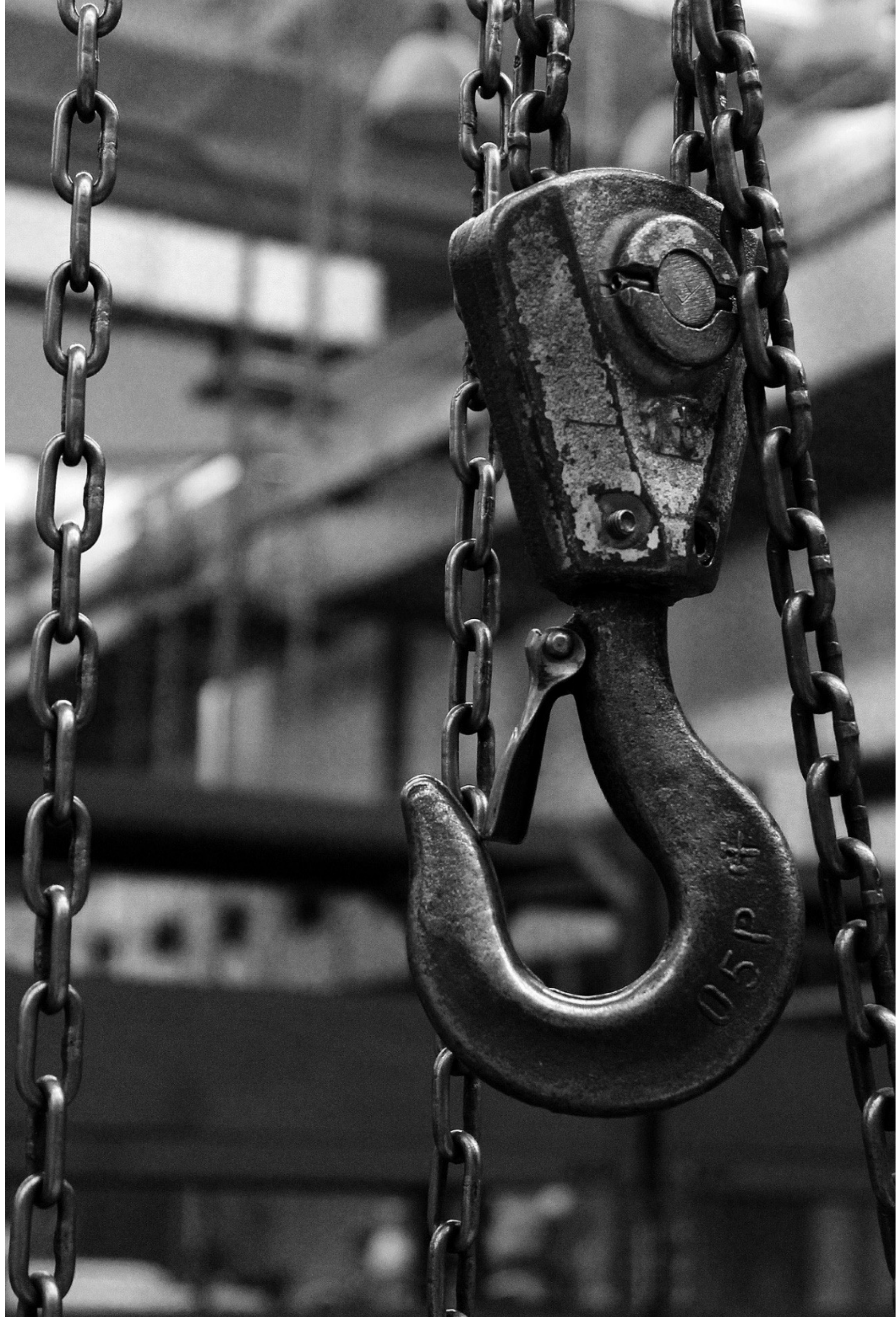
View

```
<tab title="Booked">
    Sample Text ...
</tab>
```

Content

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)

OnChanges	ngOnChanges
OnInit	ngOnInit
OnDestroy	ngOnDestroy

Lifecycle Hooks (all, in order)

0	constructor	
1?	ngOnChanges	
2	ngOnInit	
2,5	ngDoCheck	1
3	ngAfterContentInit	
3,5	ngAfterContentChecked	2
4	ngAfterViewInit	
4,5	ngAfterViewChecked	3
	ngOnDestroy	

Usage

```
@Component({
    selector: 'my-component',
    […]
})
export class Component implements OnChanges, OnInit {

    @Input() someData;

    ngOnChanges(changes: SimpleChanges): void {
        […]
    }

    ngOnInit(): void {
        […]
    }
}
```

Thought experiment

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

Smart vs. Dumb Components

Smart / Controller	Dumb / Presentational
• 1 per feature / use case / route	• Independent of Use Case
• Business logic	• Reusable
• Container	• Often Leafs

DEMO