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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
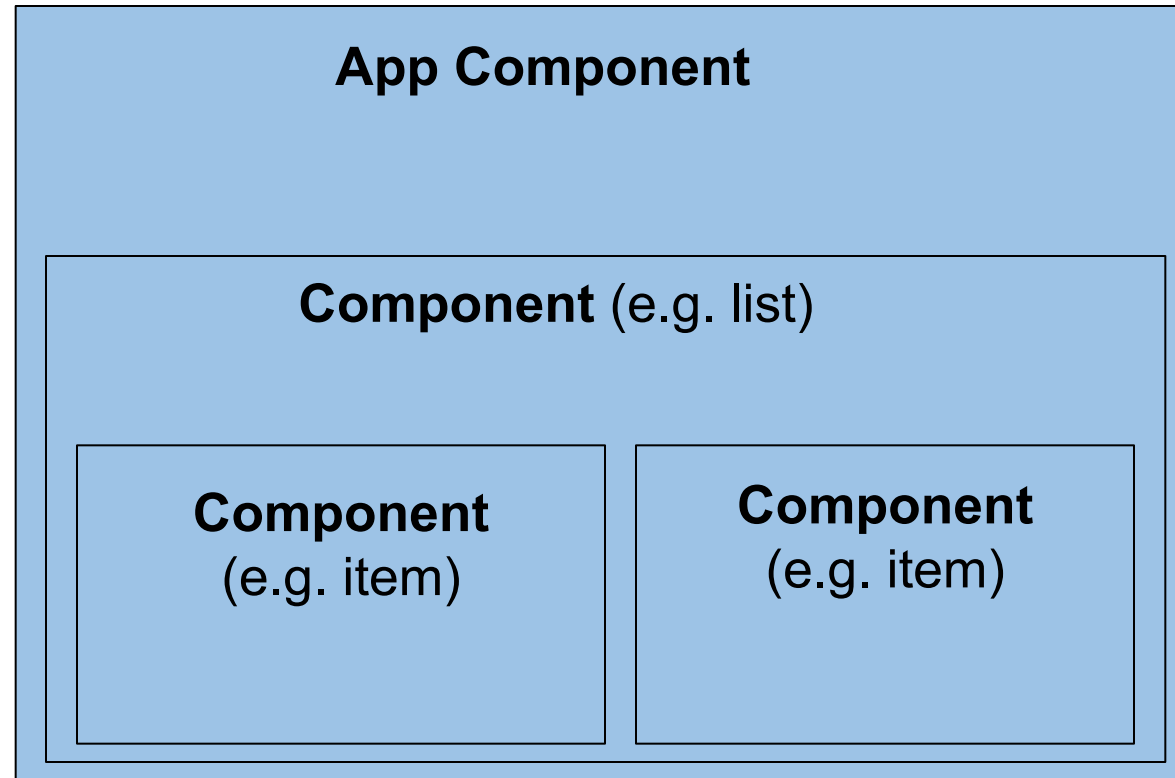**ARCHITECT**

Data binding

Performance Components Predictability
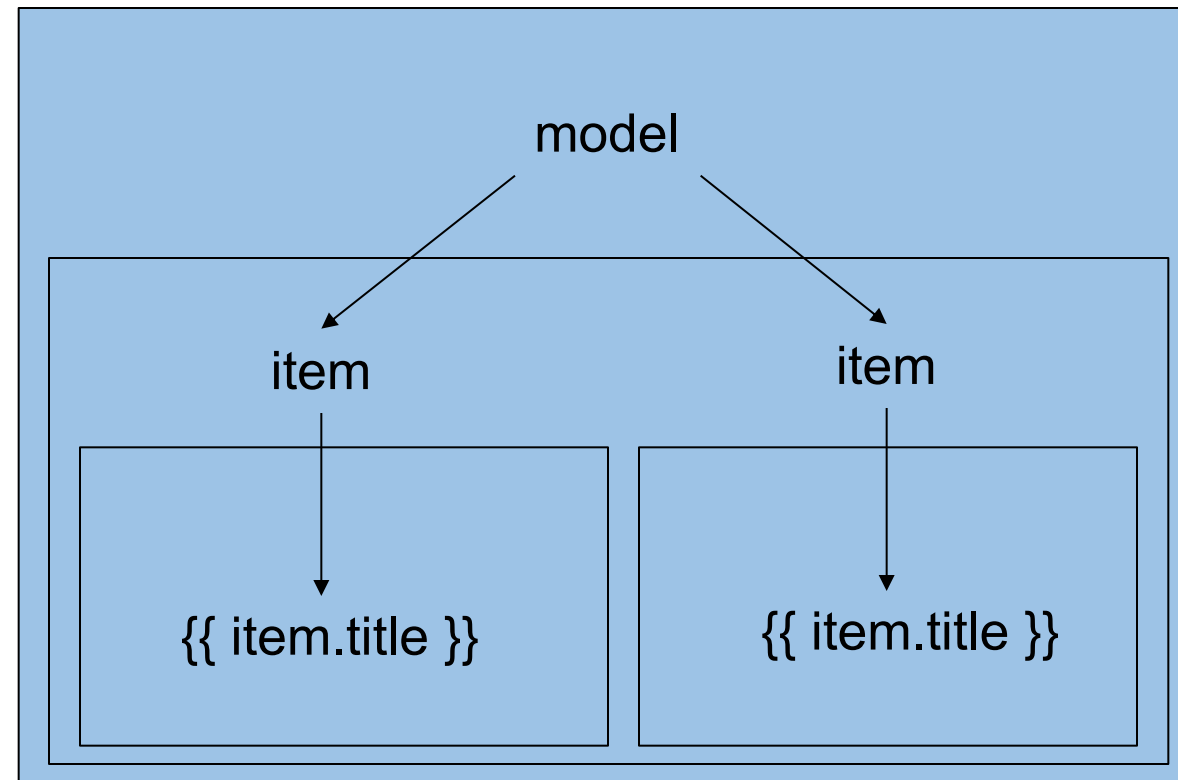
Architecture goals in Angular

Component tree in Angular 2+
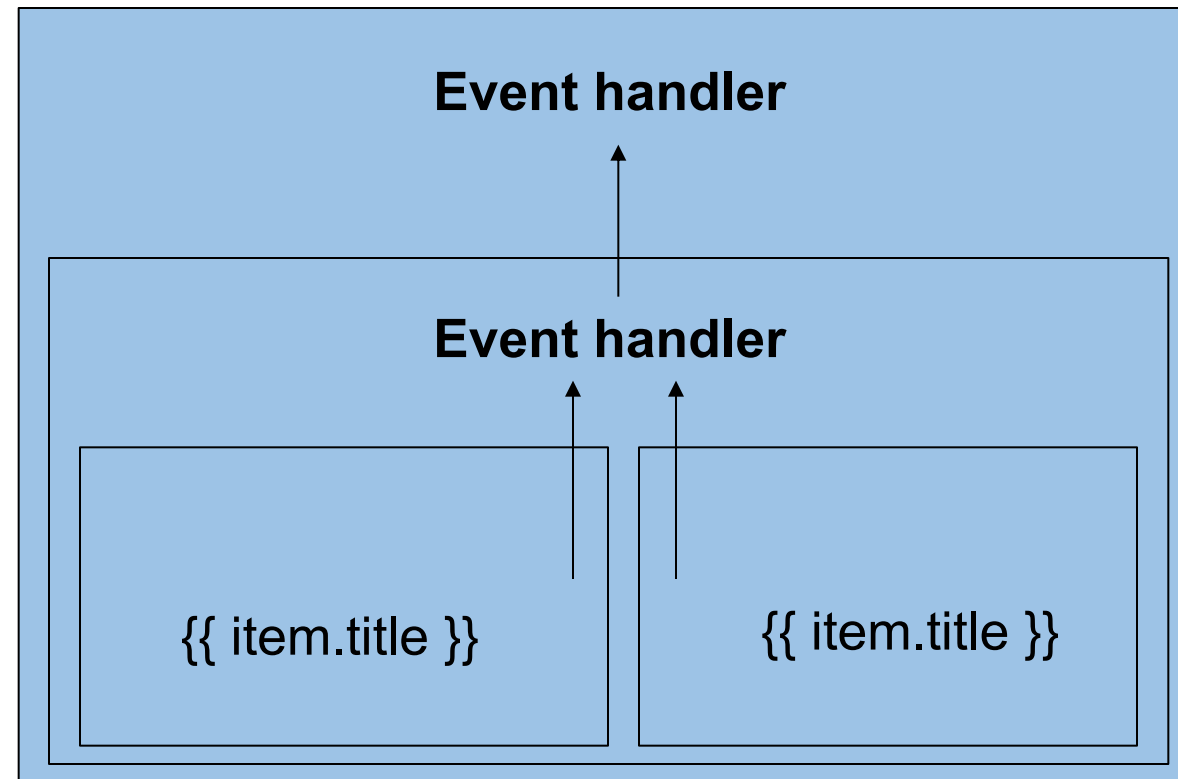
Rules for property binding

- 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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE
SOFTWARE ARCHITECT

Property binding

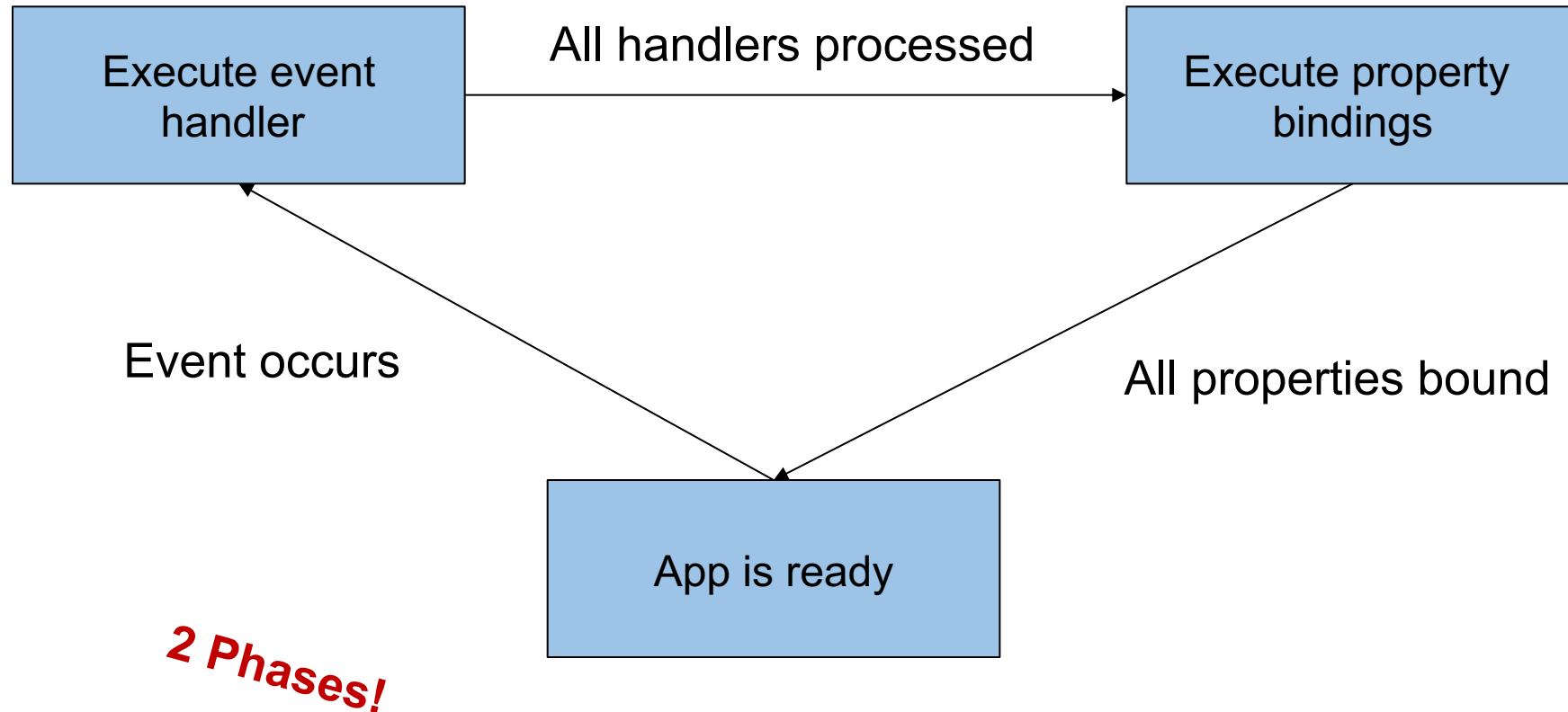[http://victorsavkin.com/post/110170125256/change-detection-in-angular-2]

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>
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Recap

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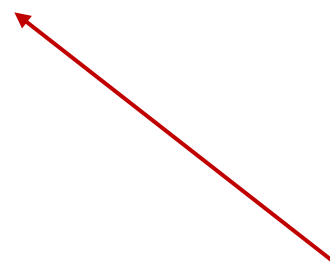
Property + event binding

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

Property + event binding

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

Property + *Change*

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

Changed value

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT
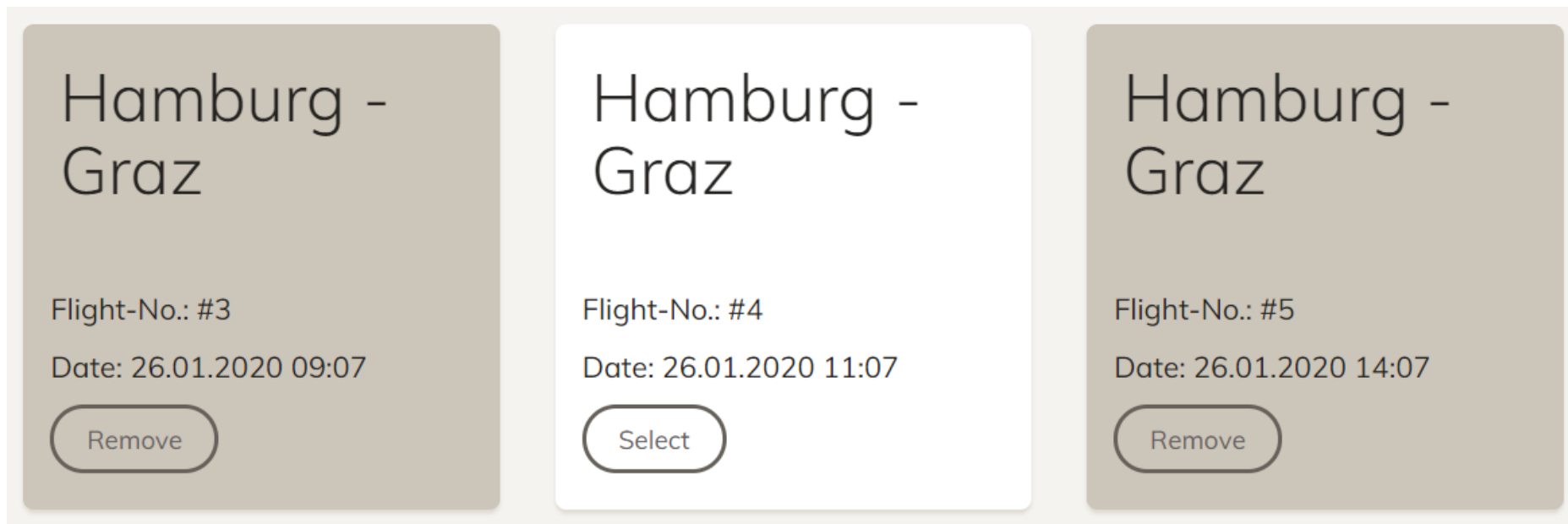
Components data bindung

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

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;
```
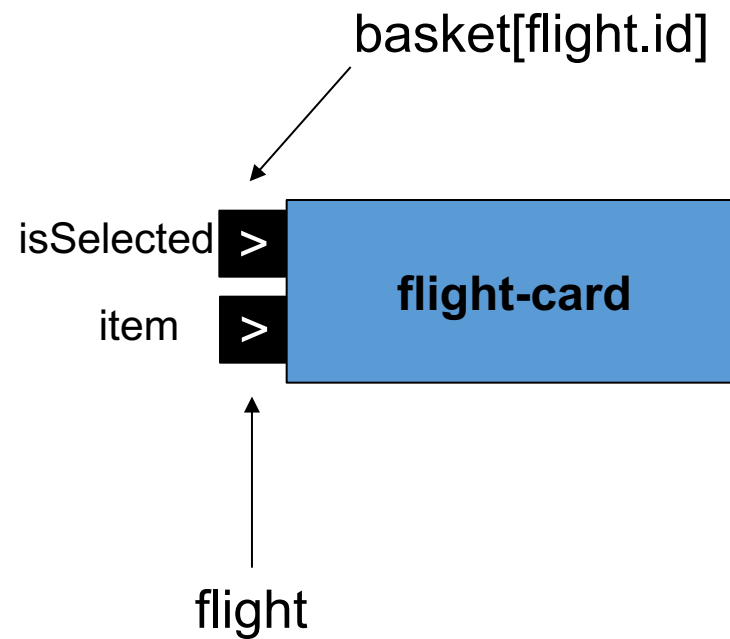
Example: flight-card in flight-search.html

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


    <flight-card [item]="f" [isSelected]="basket[f.id]">

    </flight-card>


</div>
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

flight-card

Example: flight-card

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

    […]

}
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Example: flight-card

```typescript
export class FlightCard {

    @Input() item: Flight;
    @Input() isSelected: boolean;

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

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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Template

```html
<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>
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Register component

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

DEMO

Event bindings
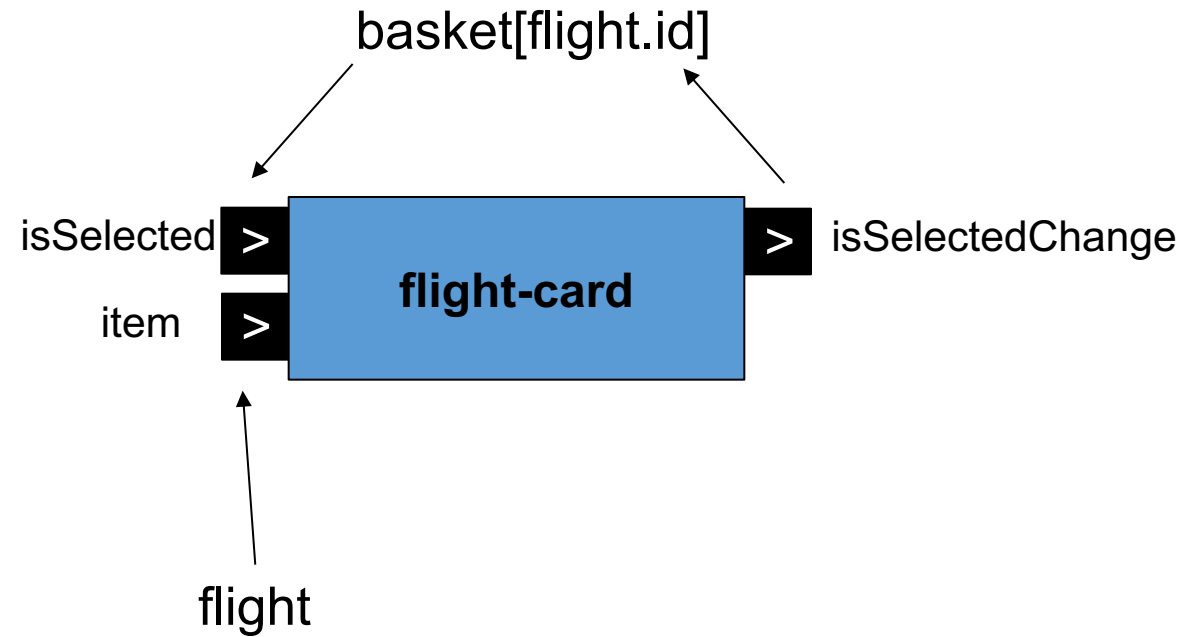
Example: flight-card event *isSelectedChange*

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


    <flight-card [item]="f"

                 [isSelected]="basket[f.id]"

                 (isSelectedChange)="basket[f.id] = $event">

    </flight-card>


</div>
```

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

flight-card

Example: flight-ca

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

    <flight-card [item]="f"

                 [isSelected]="basket[f.id]"

          (isSelectedChange)="basket[f.id] = $event">

    </flight-card>

</div>
```

```
export class FlightCard {

    @Input() item: Flight;
    @Input() isSelected: boo
    @Output() isSelectedChan


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


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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

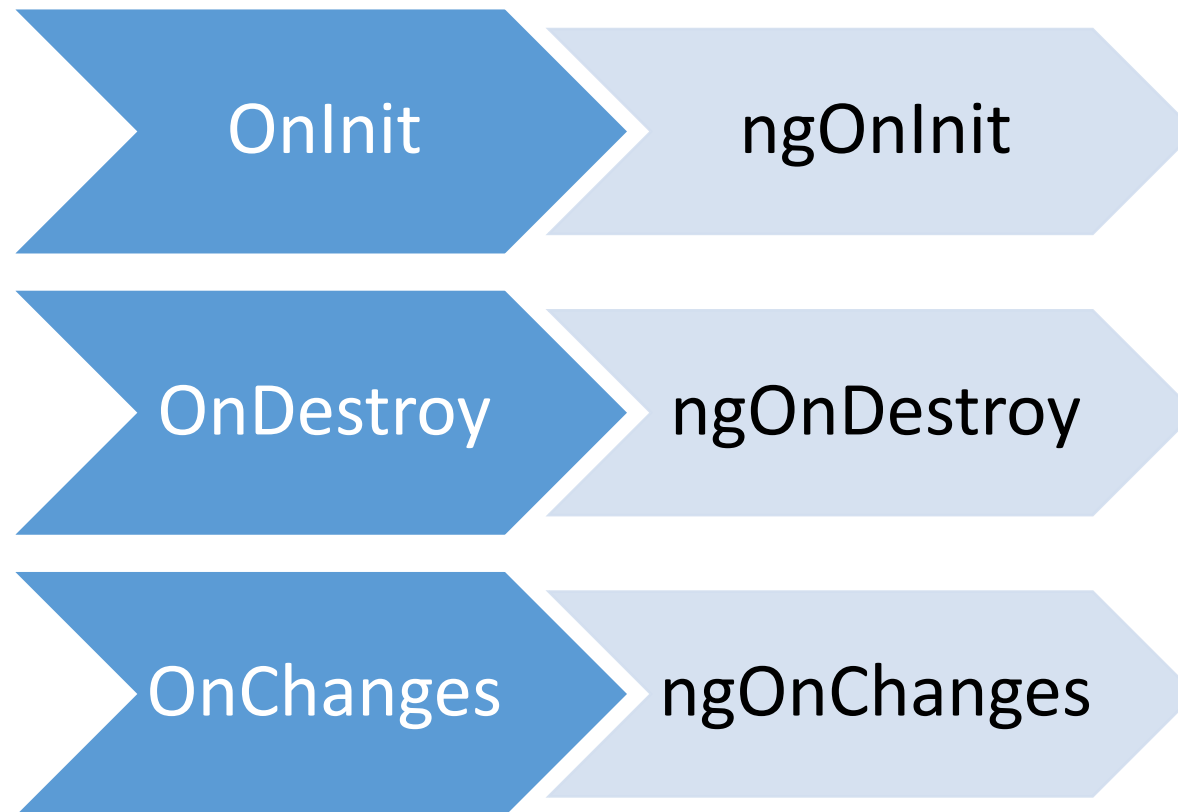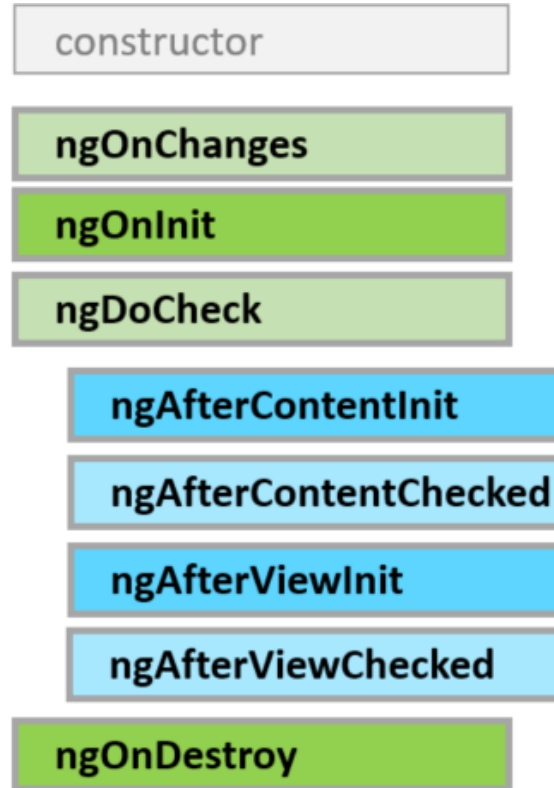# DEMO

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)

OnInit → ngOnInit

OnDestroy → ngOnDestroy

OnChanges → ngOnChanges

Lifecycle Hooks (all, in order)

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

Usage

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

    @Input() someData;

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

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

ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

SOFTWARE
ARCHITECT

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	Dumb
• Use Case controller • Container	• Independent of Use Case • Reusable • Leave