



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE

# Data Binding and OnPush

[ANGULARarchitects.io](https://ANGULARarchitects.io)

# Contents

- How does data binding work (underneath the covers)?
- Performance-Tuning with OnPush



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# Data Binding

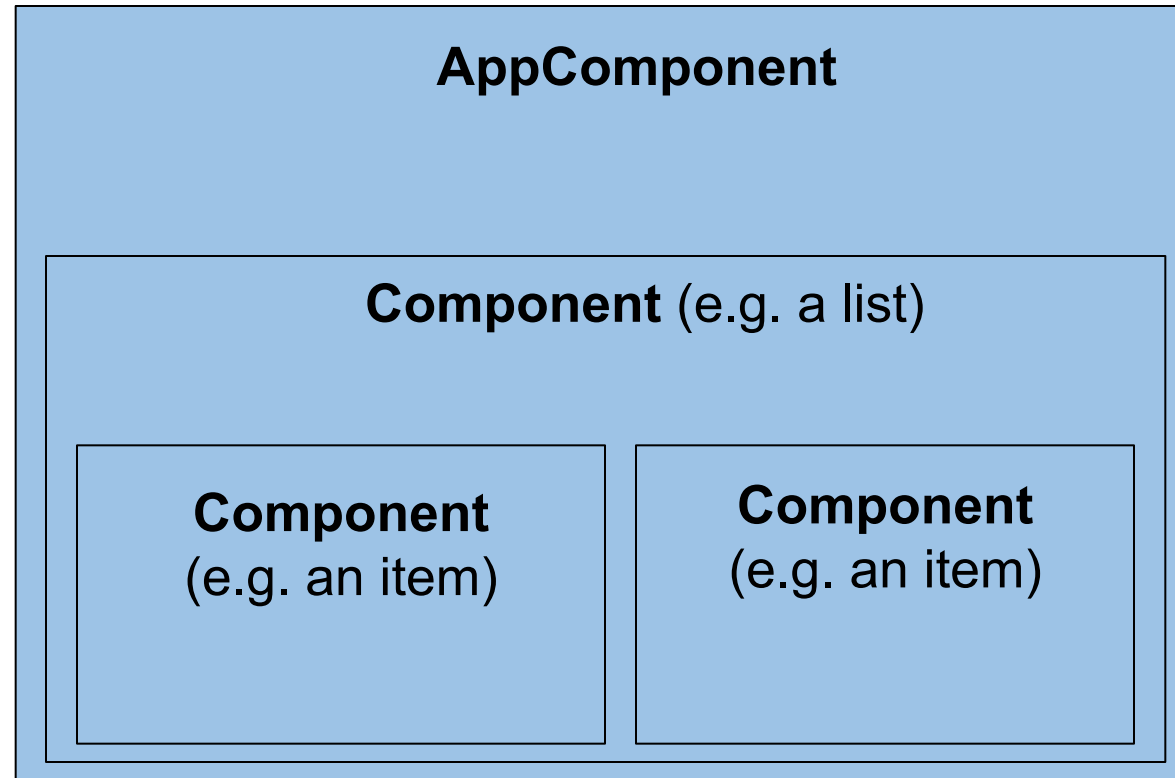


ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

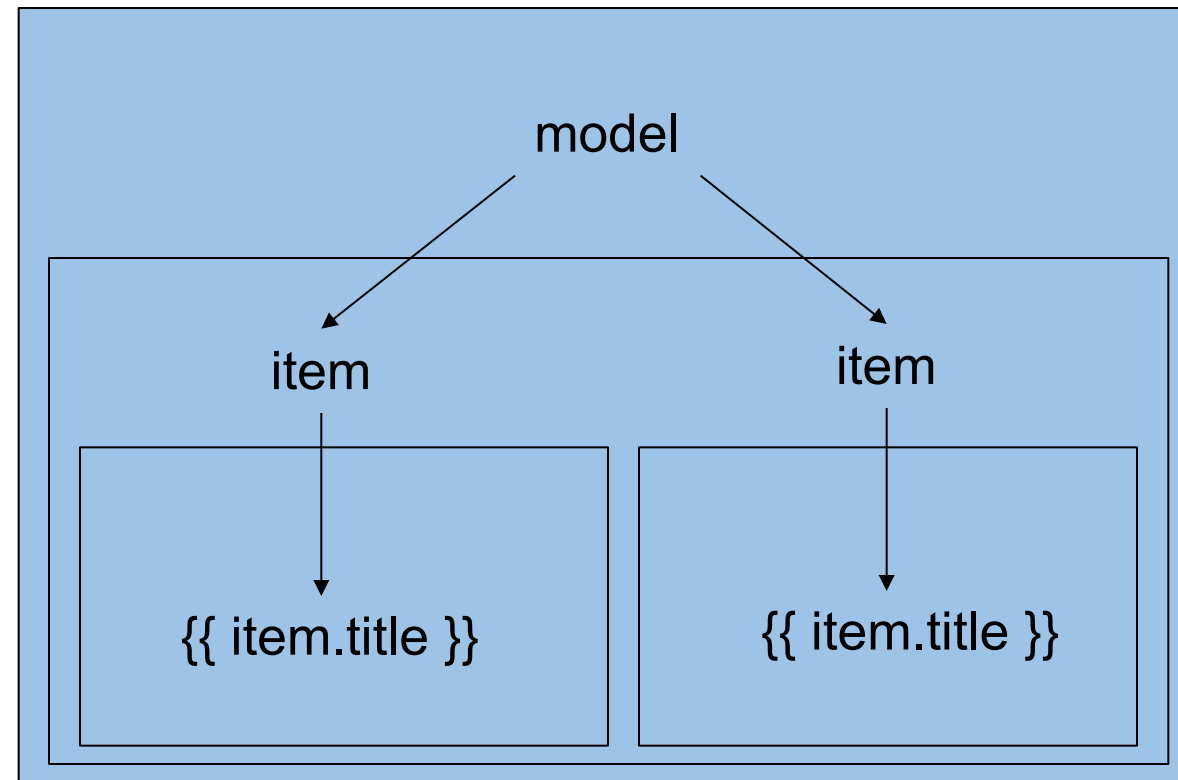
# Component Tree in Angular 2+



# Rules for Property-Bindings

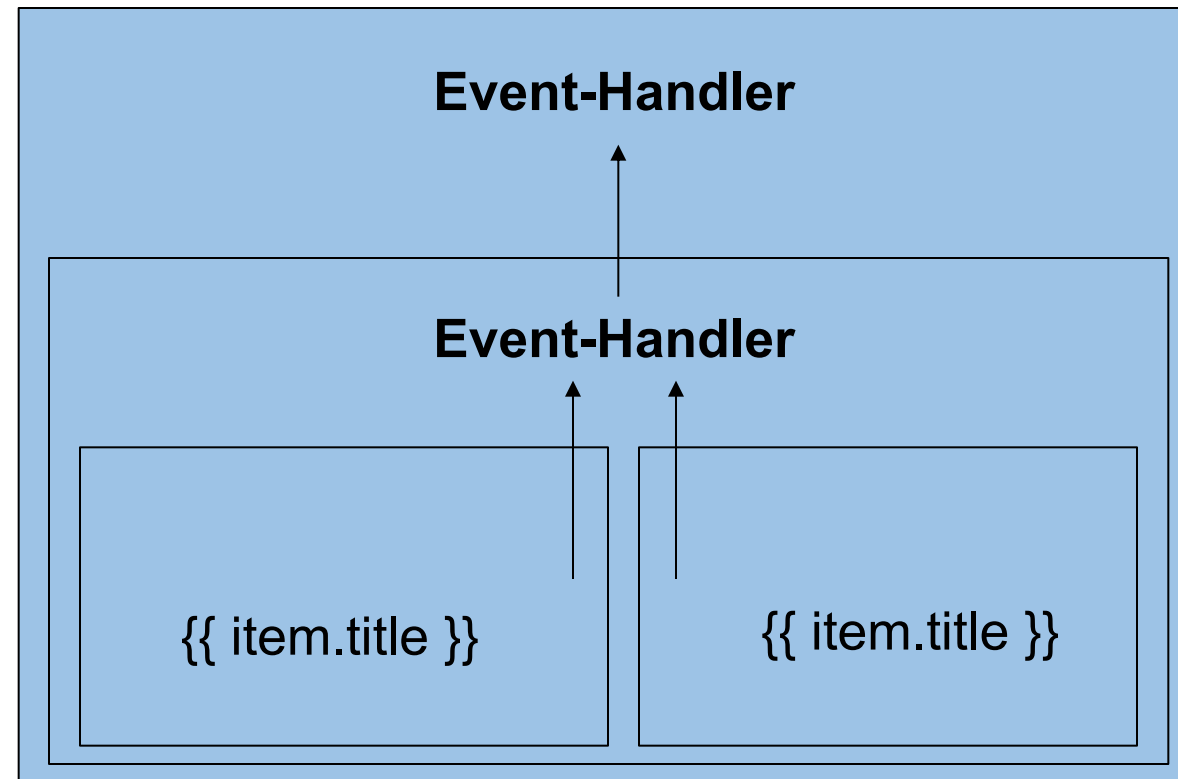
- Data flows top/down
  - Parent can send data to children
  - Children **cannot** send data to parent
- Dependency graph is a tree
- Angular only needs one "digest"

# Property Binding



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

# Event Bindings (One-Way, Bottom/Up)

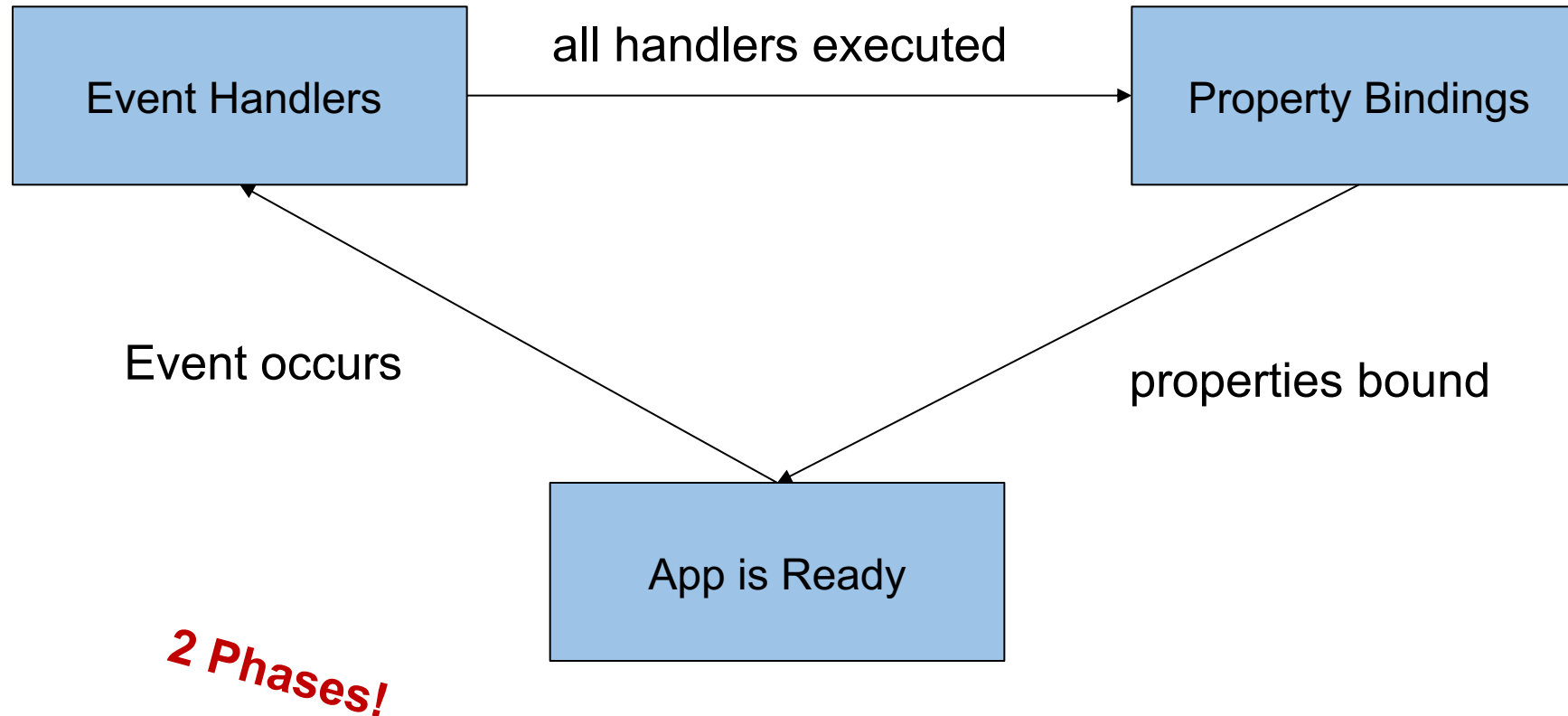


# Event Bindings (One-Way, Bottom/Up)

- Cheap: No "digest" needed!
- However: Events can change data → 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>
```

# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# Recap

- Property-Binding: One-Way; Top/Down
- Event-Binding: One-Way; Bottom/Up
- Two-Way-Binding?
- Two-Way = Property-Binding + Event-Binding



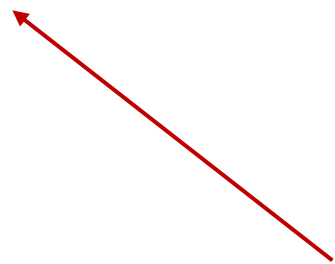
# Property and Event Bindings

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



# Property and Event Bindings

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



Property + *Change*

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



New Value



# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

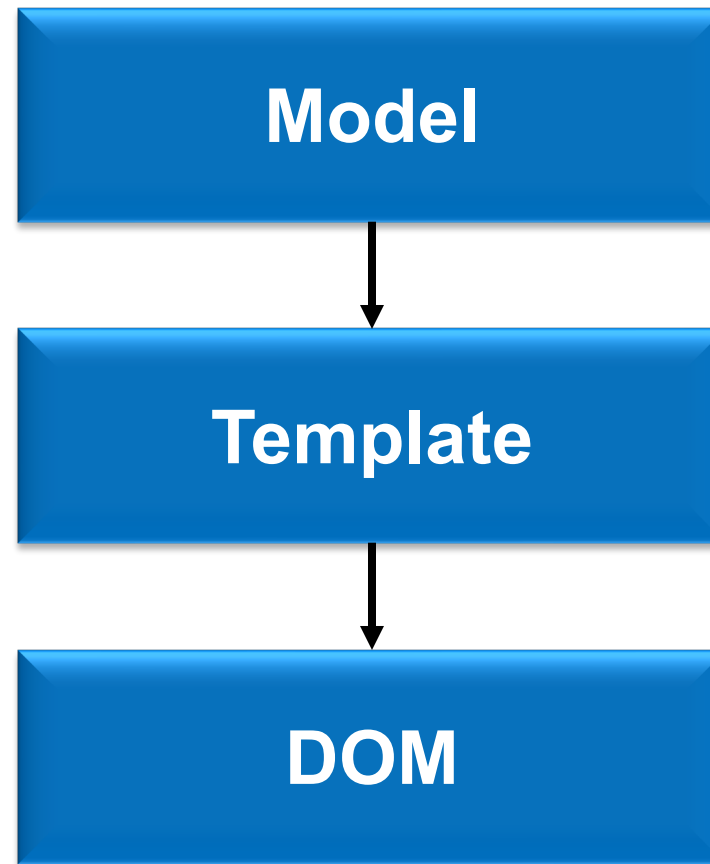
# Change Detection in Angular



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



# DOM Rendering



# Change Detection

- 1.) User or App changes the model (e.g. @Input() Binding)
- 2.) NG CD checks for every component (from root to leaves) if the corresponding component model has changes and thus its view (DOM) needs to be updated
- 3.) If yes then update / rerender the component's view (DOM)

# DEMO

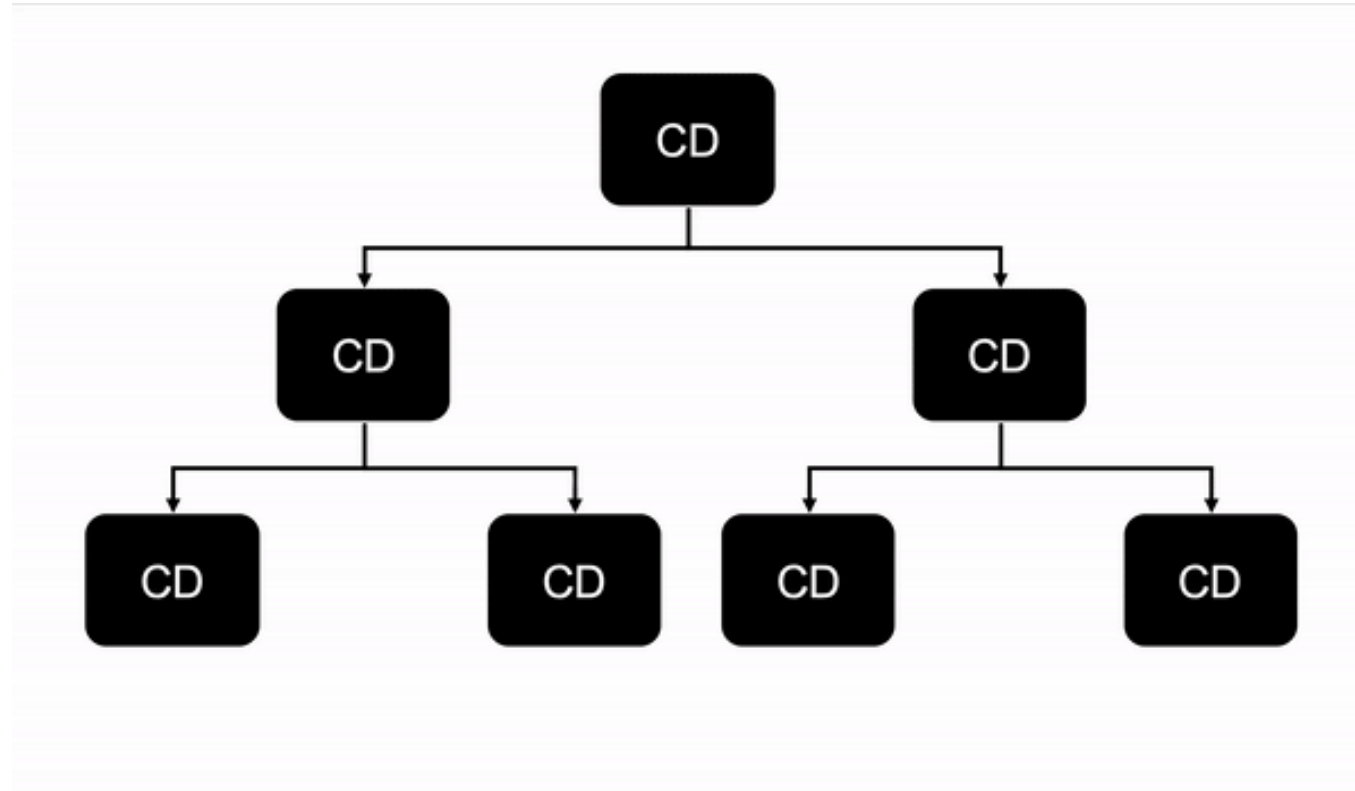


ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# Change Detection – From Root To Leaves



<https://mokkapps.de/blog/the-last-guide-for-angular-change-detection-you-will-ever-need/>



# Performance-Tuning with OnPush

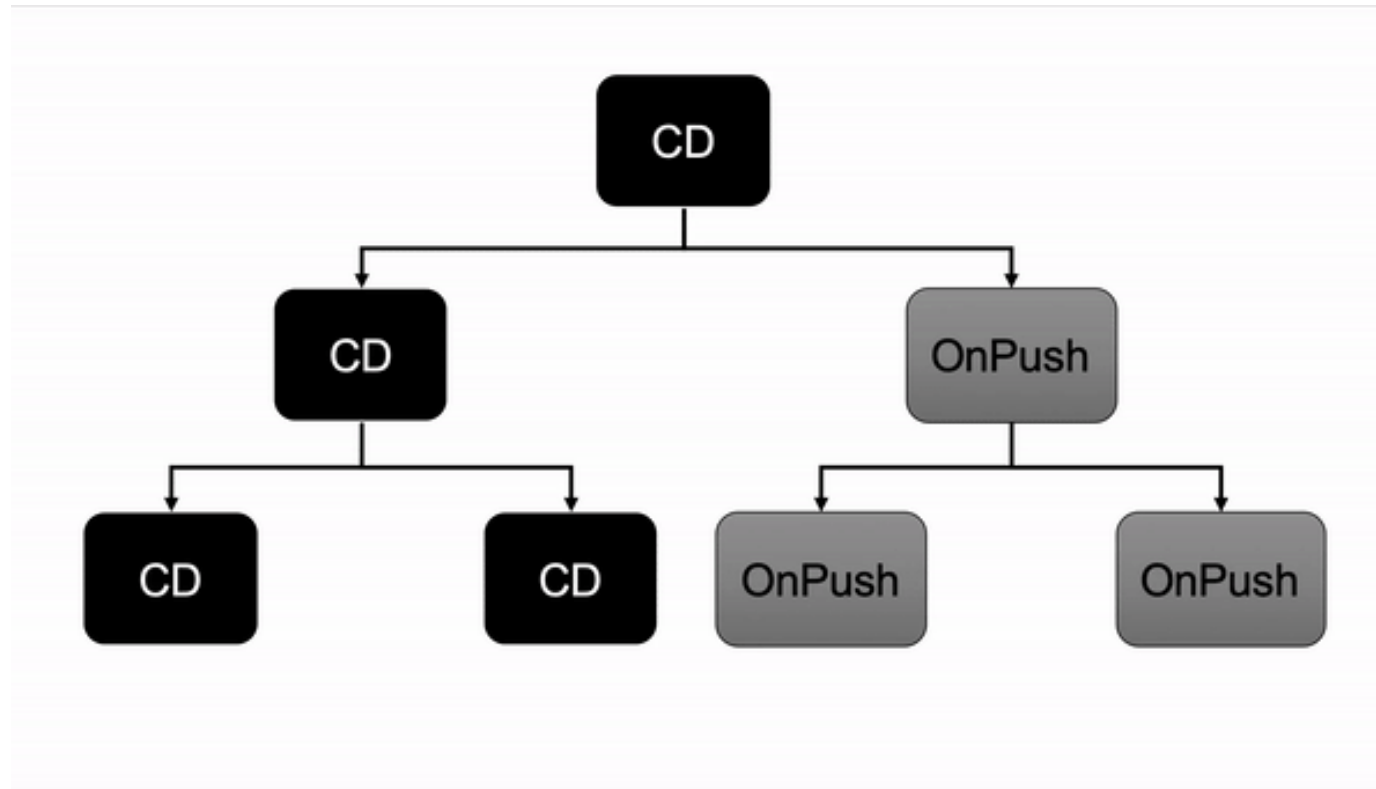


ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



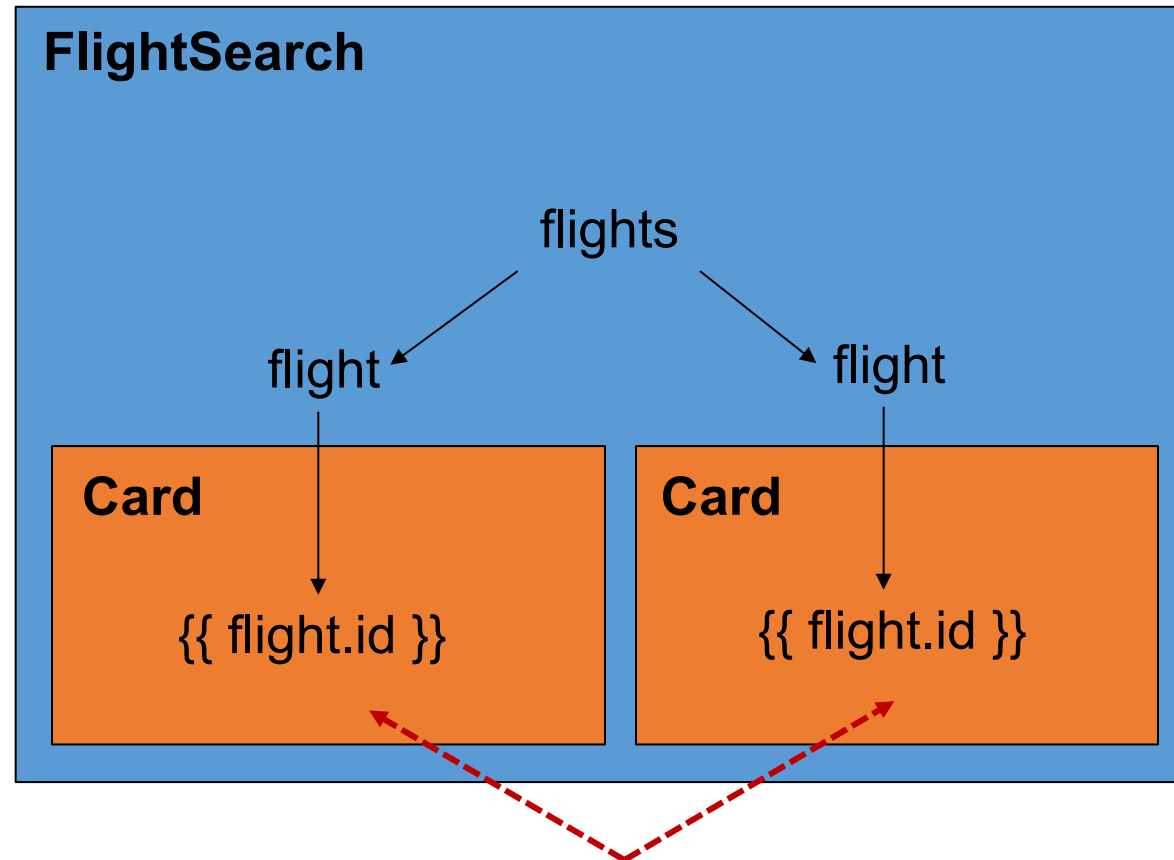
SOFTWARE  
**ARCHITECT**

# Change Detection – OnPush Strategy



<https://mokkapps.de/blog/the-last-guide-for-angular-change-detection-you-will-ever-need/>

# Change Detection – OnPush Strategy



Angular just checks when “notified”



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# "Notify" about change?

- Change bound data (@Input)
  - OnPush: Angular just compares the object reference!
  - e. g. `oldFlight === newFlight`
- Raise event within the component and its children (e.g. @Output)
- Notify a bound observable using the async pipe
  - `{{ flights$ | async }}`
- Trigger it manually
  - Don't do this at home ;-)
  - Try to avoid this – but there are reasonable cases



ANGULAR  
ARCHITECTS  
INSIDE KNOWLEDGE



SOFTWARE  
ARCHITECT



# Activate OnPush Strategy

```
@Component({  
  [...]  
  changeDetection: ChangeDetectionStrategy.OnPush  
})  
export class FlightCard {  
  [...]  
  @Input() flight;  
}
```



# DEMO



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# LAB



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# One more thing: ChangeDetectorReference

## detectChanges

- Runs Change Detector for the component and its children
- It runs CD once also for the component which is detached from the component tree

## markForCheck

- It marks component and all parents up to root as dirty
- In next cycle Angular runs CD for marked components

## reattach

- Re-attaches the component in the change detection tree
- If parent component's CD is detached, it won't help, so make sure to run markForCheck with reattach

## detach

- Detaches the component from the change detection tree
- Bindings will also not work for the component with detached CD

## checkNoChanges

- Changes the component and its children and throws error if change detected



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# CDR - detectChanges() vs markForCheck()

- Use **CDR.detectChanges()** to trigger **CD immediately** for this view and it's children respecting the its/their CD strategy
  - Useful when you've updated the model after angular has run it's change detection, or if the update hasn't been in Angular world at all
  - For the whole Application you can to ApplicationRef.tick()
- Use **CDR.markForCheck()** to notify the next **CD cycle** if using **OnPush**
  - Useful when you're bypassing the ChangeDetectionStrategy.OnPush e.g. by mutating some data or you've just updated the components model



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**

# Summary

- Event Bindings → Property Bindings
  - Two-way bindings
- CD Strategy OnPush
  - ChangeDetectorReference



ANGULAR  
**ARCHITECTS**  
INSIDE KNOWLEDGE



SOFTWARE  
**ARCHITECT**