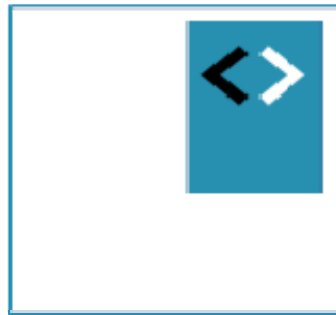


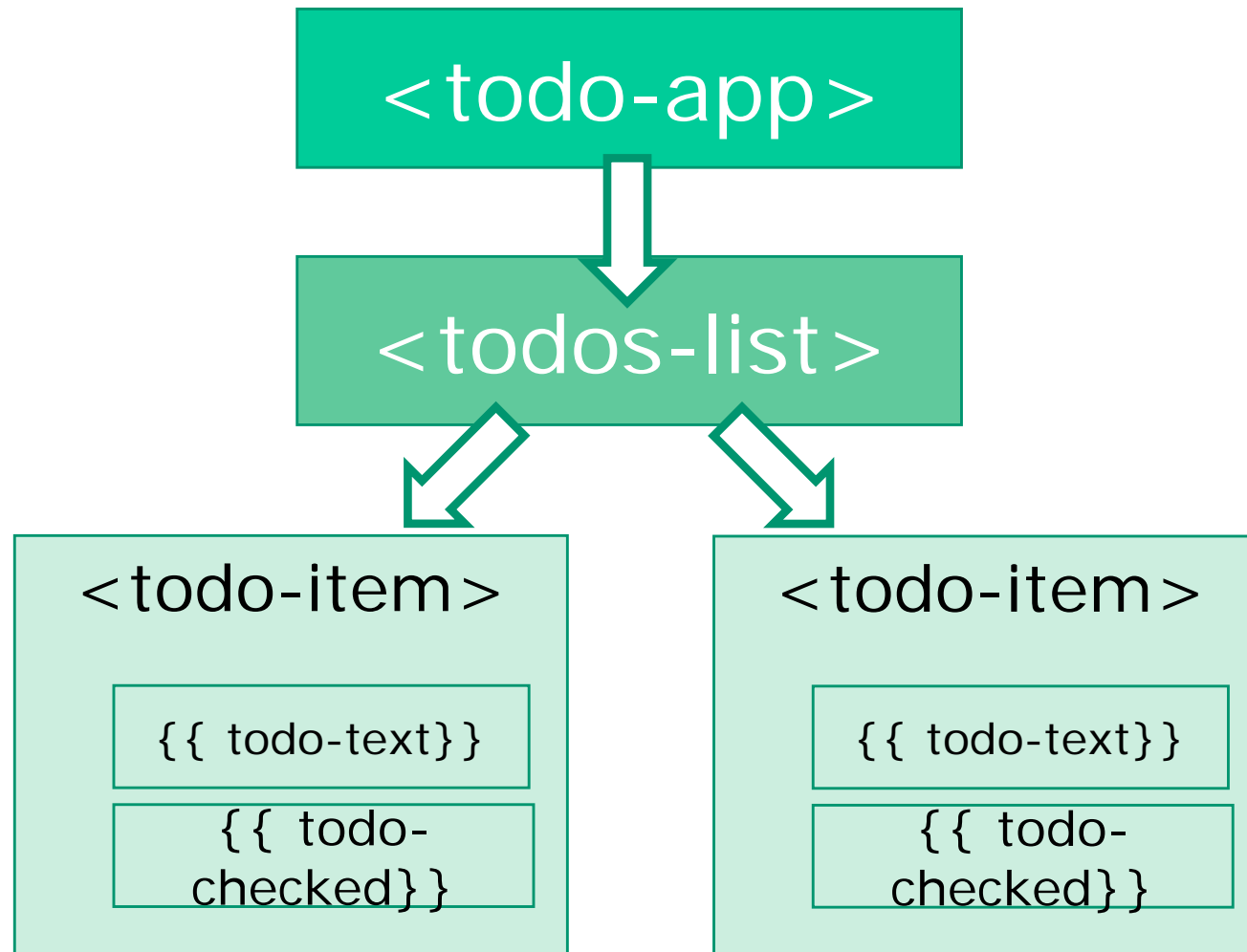
# Angular Fundamentals

## Module 4 – Building apps with multiple components



Peter Kassenaar –  
[info@kassenaar.com](mailto:info@kassenaar.com)

# Angular-app: Tree of components



# Application as a tree of components

- Multiple components?
  1. Create files manually – or let CLI handle this for you
    1. `ng generate component <component-name>`
    2. `Ng g c <component-name>`
  2. Import in module – or (again) let CLI take care of this for you
  3. Add to declarations : [...] section of `@ngModule`.
  4. Add via HTML to parent-component
- Repeat for every component

# 1. Add DetailComponent

```
// city.detail.ts
import { Component } from '@angular/core';

@Component({
  selector: 'city-detail',
  template: `
    <h2>City details</h2>
    <ul class="list-group">
      <li class="list-group-item">Name: [city name]</li>
      <li class="list-group-item">Province: [province]</li>
      <li class="list-group-item">Highlights: [highlights]</li>
    </ul>
  `
})

export class CityDetail{

}
```

## 2. Add to Module

*// Angular Modules*

...

*// Custom Components*

```
import {AppComponent} from './app.component';  
import {CityDetail} from './city.detail';  
import {CityService} from './city.service';
```

New component

*// Module declaration*

```
@NgModule({  
  imports      : [BrowserModule, HttpClientModule],  
  declarations: [AppComponent, CityDetail],  
  bootstrap   : [AppComponent],  
  providers    : [CityService]  
})  
export class AppModule {  
  
}
```

Add to  
declarations:[ ]

### 3. Encapsulate in HTML

```
<!-- app.html -->
```

```
<div class="row">
```

```
...
```

```
<div class="col-md-6">
```

```
...
```

```
<city-detail></city-detail>
```

```
</div>
```

```
</div>
```



Combine with other HTML

## 4. Result

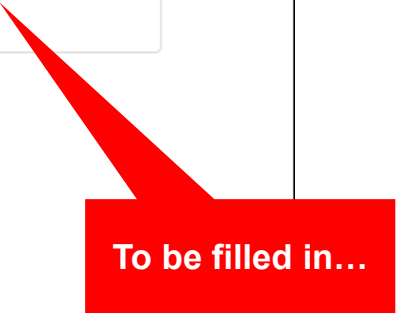
### Cities via een service

Mijn favoriete steden zijn :

|               |
|---------------|
| 1 - Groningen |
| 2 - Hengelo   |
| 3 - Den Haag  |
| 4 - Enschede  |
| 5 - Heerlen   |
| 6 - Mechelen  |

### City details

|                          |
|--------------------------|
| Naam: [naam van stad]    |
| Provincie: [provincie]   |
| Highlights: [highlights] |



Goal: show details of selected city in child-component



# Data flow between components

Using inputs and outputs

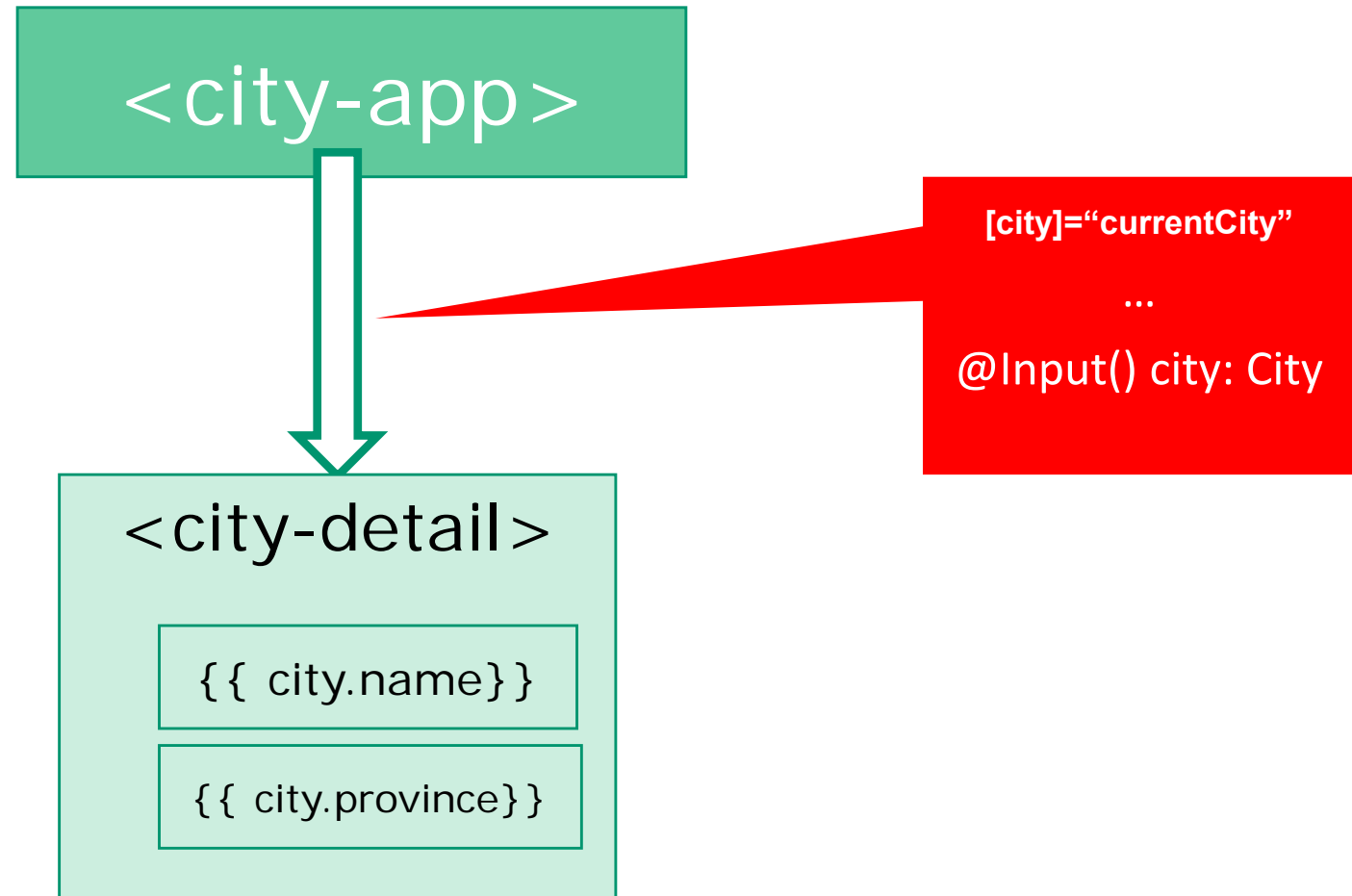


## Data flow between components

*"Data flows in to a component  
via @Input( )'s"*

*Data flows out of a component  
via @Output( )'s"*

# Parent-Child flow: decorator @Input ( )



# Working with @Input ( )

1. Import `Input` decorator in component
2. Use annotation `@Input ( )` in class definition

```
// city.detail.ts
import { Component, Input } from '@angular/core';
import { City } from './city.model';
```

```
@Component({
  ...
})
```

```
export class CityDetail {
  @Input() city: City;
}
```



Input

# Adapt Parent Component for @Input

```
<!-- app.html -->
<div class="row">
  <div class="col-md-6">
    ...
    <ul class="list-group">
      <li *ngFor="let city of cities" class="list-group-item"
        (click)="getCity(city)">
        {{ city.id }} - {{ city.name }}
      </li>
    </ul>
    <button *ngIf="currentCity" class="btn btn-primary"
      (click)="clearCity()">Clear</button>
  </div>
  <div class="col-md-6">
    <div *ngIf="currentCity">
      <city-detail [city]="currentCity"></city-detail>
    </div>
  </div>
</div>
```



Edit!

# Result

## Cities via een service

Mijn favoriete steden zijn :

1 - Groningen

2 - Hengelo

3 - Den Haag

4 - Enschede

5 - Heerlen

6 - Mechelen

Clear

## City details

Naam: Enschede

Provincie: Grote Markt

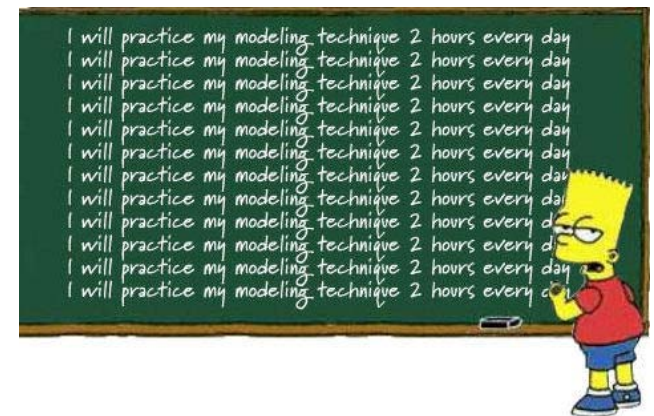
Highlights: Twentse Welle museum



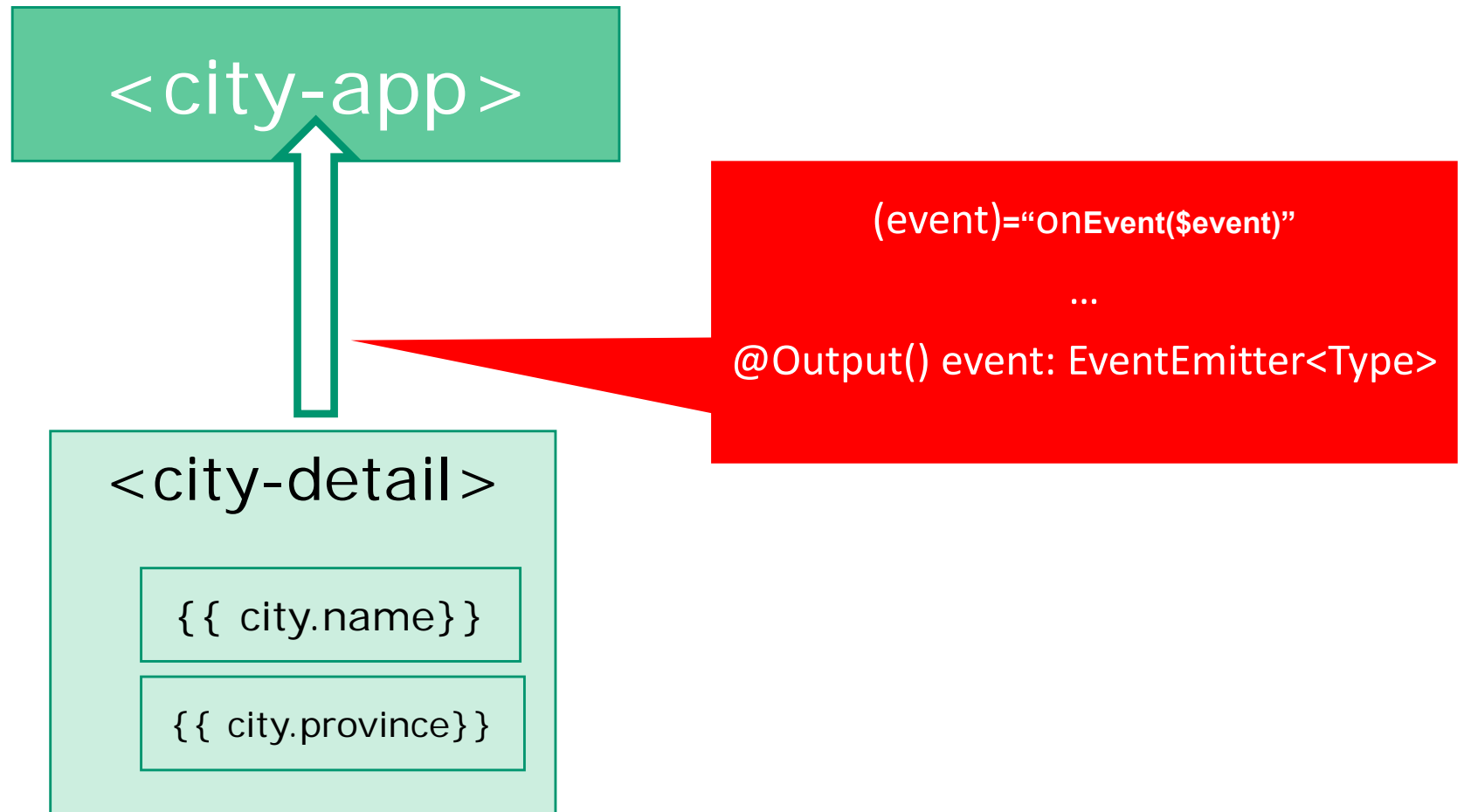
# Checkpoint

- Components can be placed inside other components
- Enhance the HTML of the Parent Component with declaration of the Child Component
- Remember to import the Child Component in `@ngModule`
- Data flow to Child Component : use `@Input()` and `[propName]="data"`
- Exercise: 6a) and 6b)
- Example: /300-components

## Exercise....



# Child-Parent flow: decorator @Output ( )



# Method – equally, but the other way around

1. Import `Output` in component
2. Use decorator `@Output()` in class definition
3. New: define `EventEmitter` to emit events of certain type

*“With @Output,  
data flows up the Component Chain”*



# Rating our cities

```
// city.detail.ts
```

```
import { Component, Input, Output, EventEmitter } from '@angular/core';
```

```
@Component({
```

```
  ...
```

```
  template: `
```

```
    <h2>City details
```

```
      <button (click)="rate(1)">+1</button>
```

```
      <button (click)="rate(-1)">-1</button>
```

```
    </h2>
```

```
  ...
```

```
  `
```

```
});
```

```
export class CityDetail {
```

```
  @Input() city: City;
```

```
  @Output() rating: EventEmitter<number> = new EventEmitter<number>();
```

```
  rate(num) {
```

```
    console.log('rating for ', this.city.name, ': ', num);
```

```
    this.rating.emit(num);
```

```
  }
```

```
}
```

Imports

Bind custom  
events to DOM

handle  
custom  
@Output

# Prepare parent component for custom event

```
<!-- app.html -->  
<div *ngIf="currentCity">  
  <city-detail [city]="currentCity" (rating)="updateRating($event)">  
  </city-detail>  
</div>
```



Capture custom  
event

```
// app.component.ts  
// increase or decrease rating on Event Emitted  
updateRating(rating){  
  this.currentCity.rating += rating;  
}
```

# Show rating in HTML

```
<li *ngFor="let city of cities"  
    class="list-group-item" (click)="getCity(city)">  
    {{ city.id }} - {{ city.name }} ({{i}})  
    <span class="badge">{{city.rating}}</span>  
</li>
```



Rating

# Result

## Cities via een service

Mijn favoriete steden zijn :

|               |    |
|---------------|----|
| 1 - Groningen | 0  |
| 2 - Hengelo   | 0  |
| 3 - Den Haag  | -3 |
| 4 - Enschede  | 0  |
| 5 - Heerlen   | 2  |
| 6 - Mechelen  | 5  |

Clear

## City details +1 -1

Naam: Den Haag

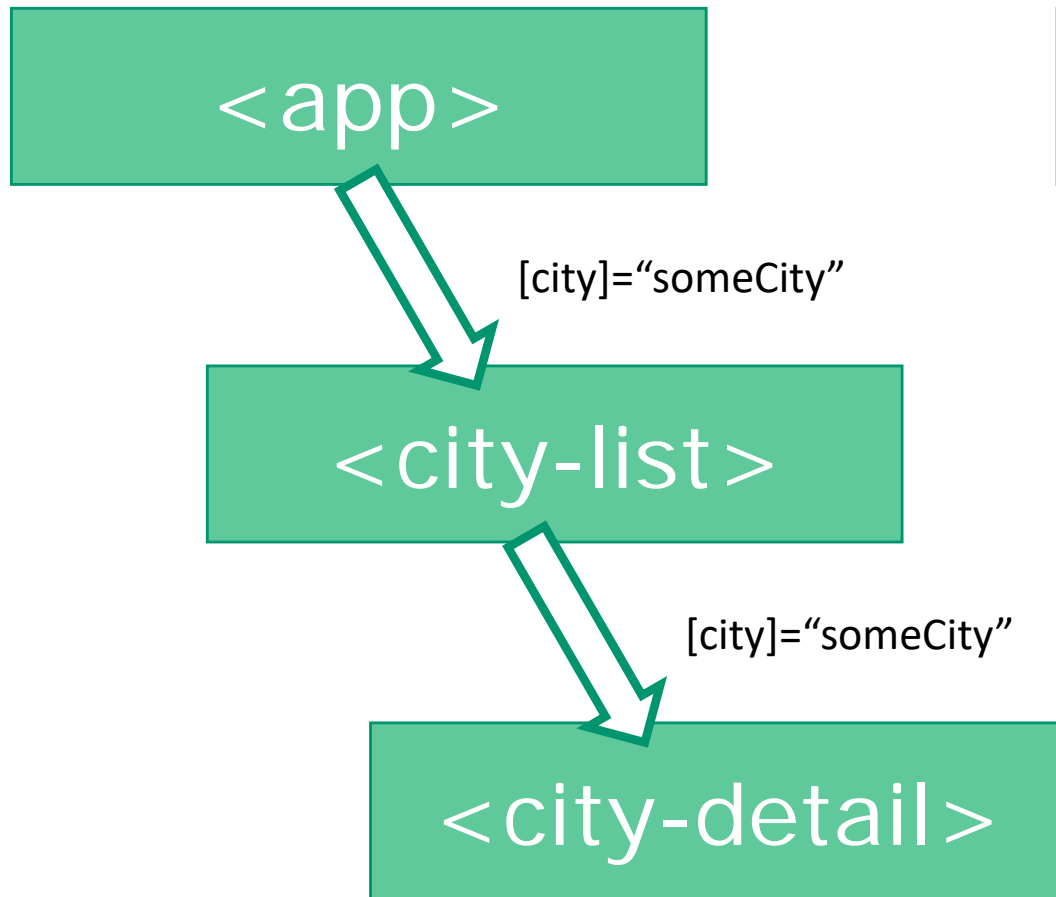
Provincie: Zuid-Holland

Highlights: Binnenhof

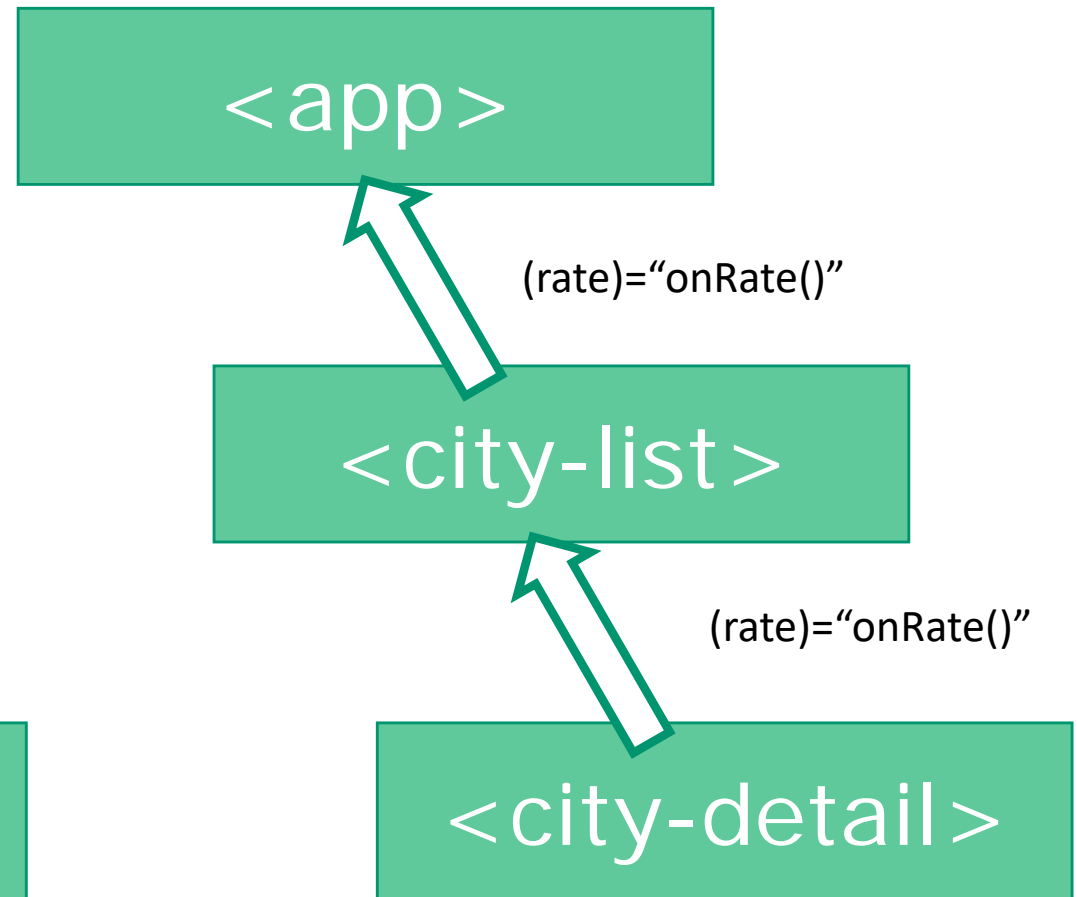


# Summary

Parent → Child



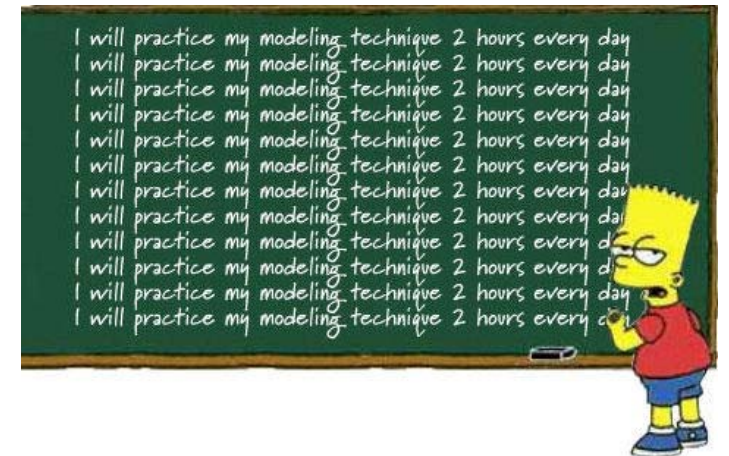
Child → Parent



# Checkpoint

- Data flow to Parent Component : using `@Output()` and `(eventName)="eventHandler($event)"`
- You can throw *any type* of data with the `EventEmitter`.
- Exercise: 6c)
- Example: `/302-components-output`
- More info: <http://victorsavkin.com/post/118372404541/the-core-concepts-of-angular-2>

## Exercise....

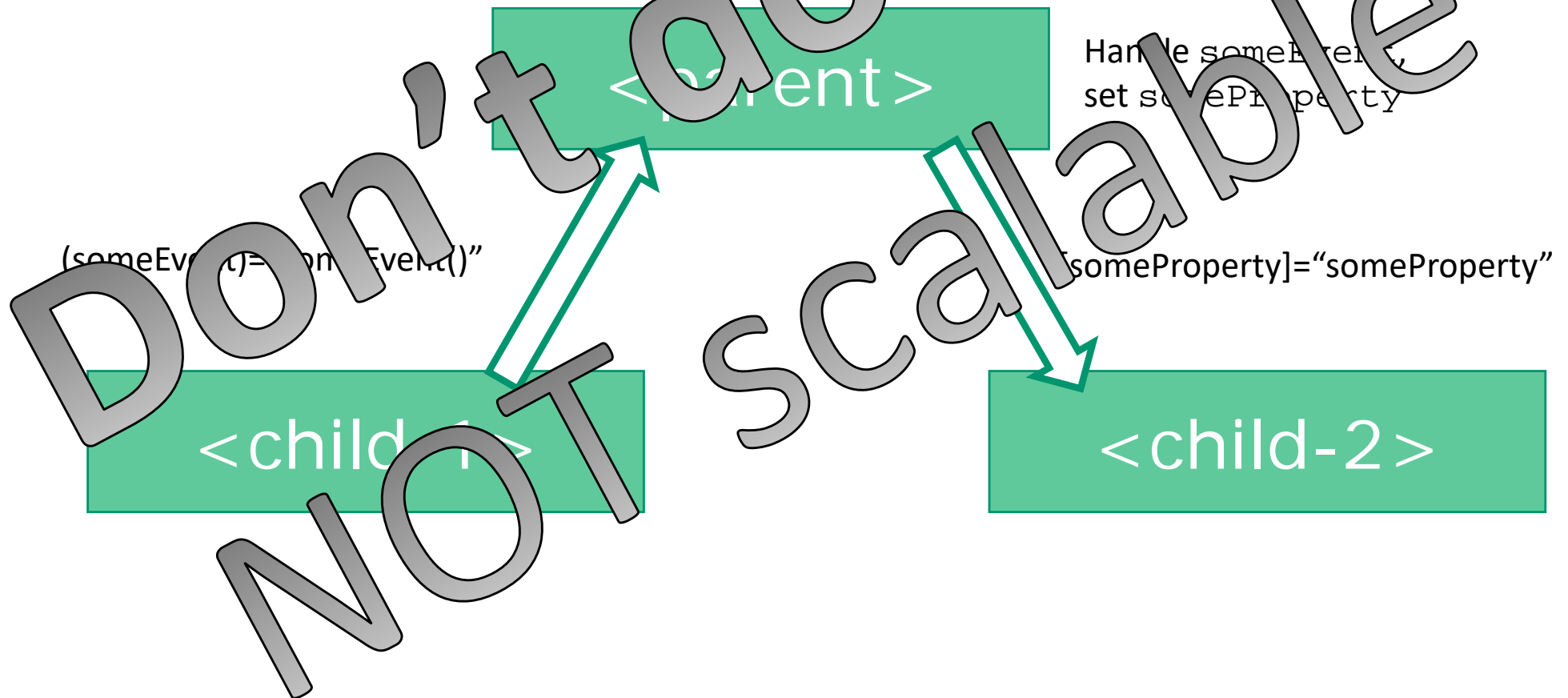




# **Sibling communication**

# Sibling communication

- Pass `Output()` of `childcomponent`, to `Input()` of other `childcomponent`

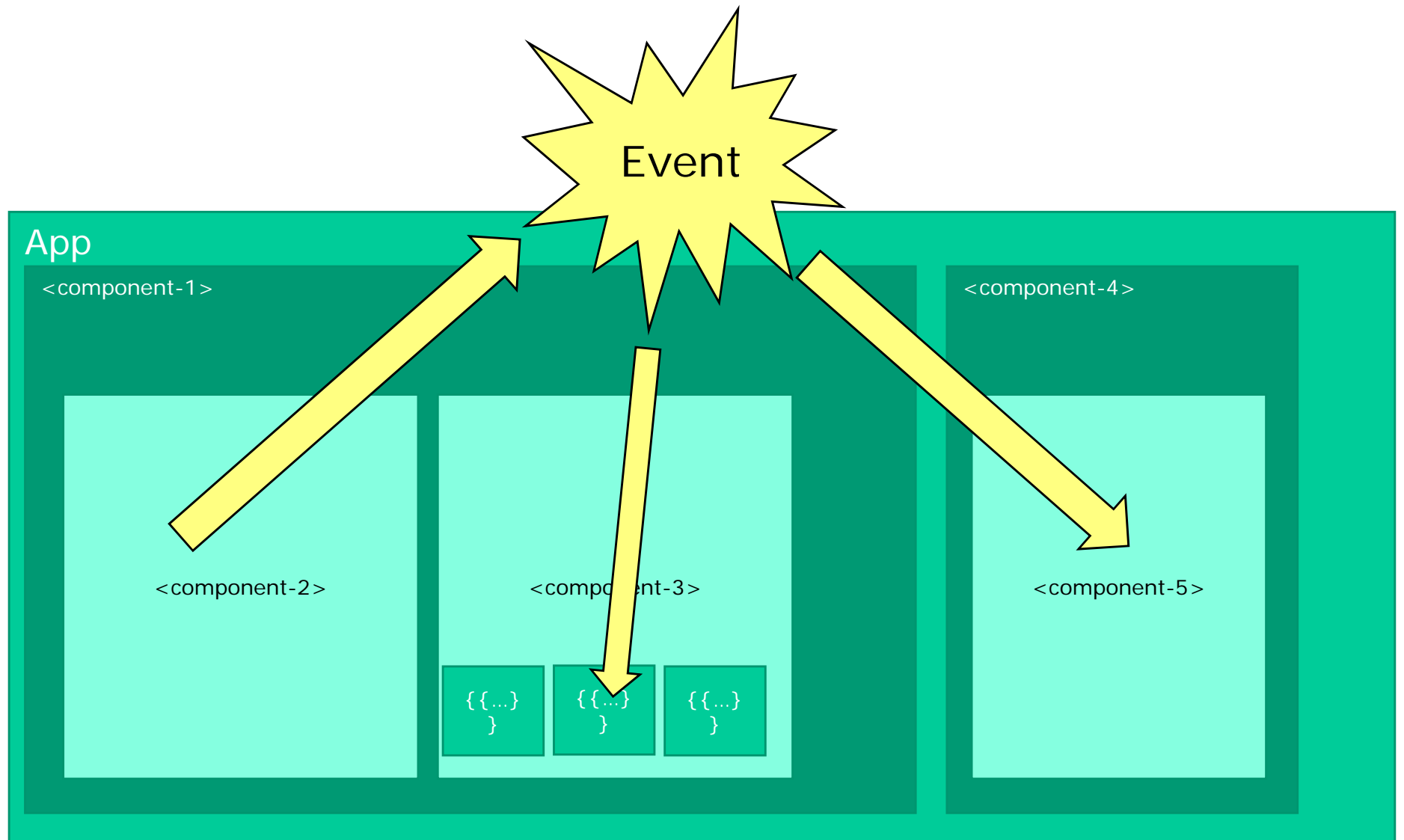




## Better solution – using a Pub/Sub-system with Observables

- <http://www.syntaxsuccess.com/viewarticle/pub-sub-in-angular-2.0>

“Custom events,  
write an event bus”

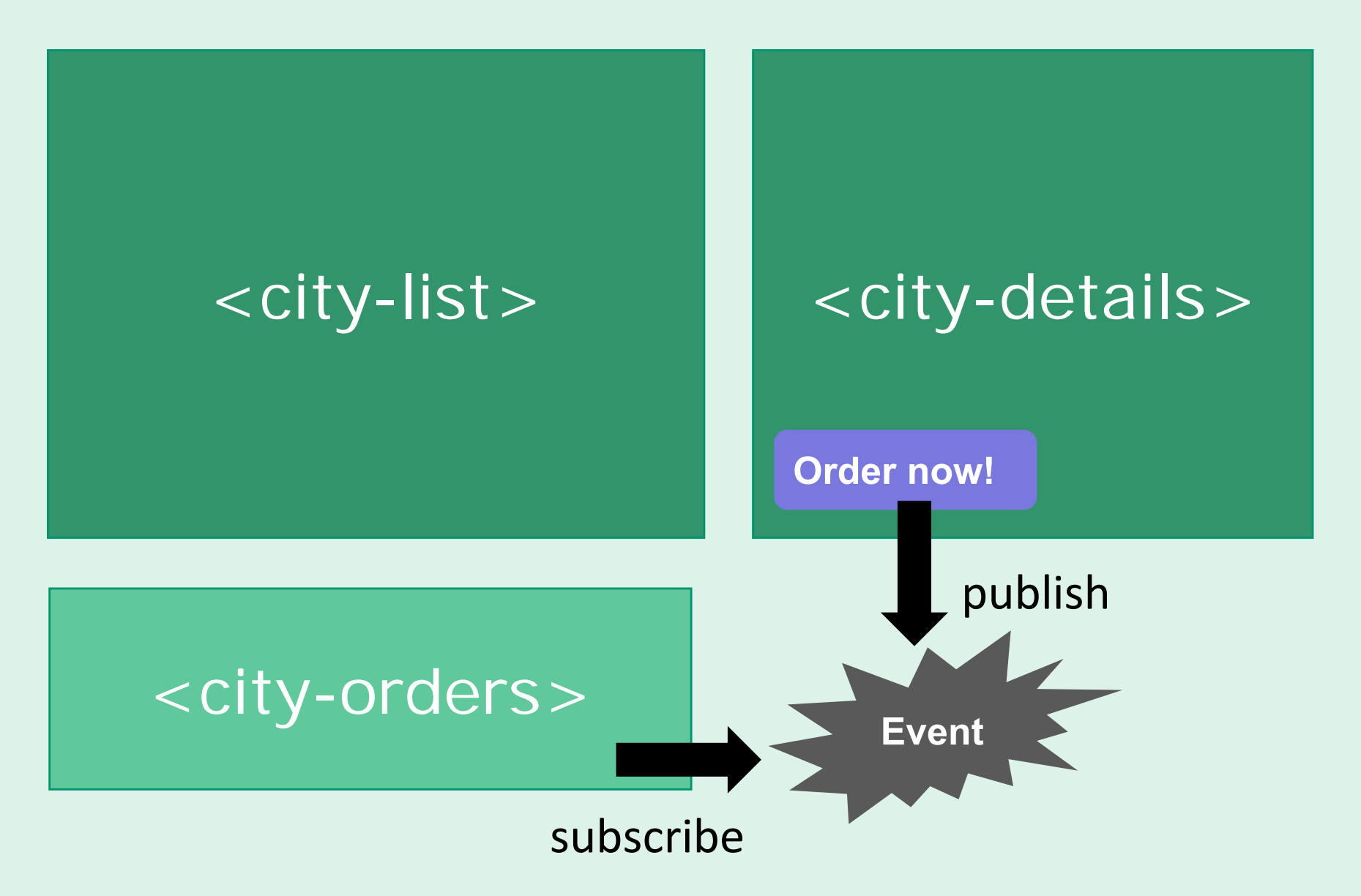


# Options

From RxJS library, work with

- `EventEmitter()`
- `Observable()`
- `Observer()`
- `Subject()` (implements `Observable` and `Observer`)

*"Publish en Subscribe"* – PubSub systeem



# Create PubSub-service

- Step 1 – create Publication Service
- Step 2 – Create 'Producer', or 'Publish' – component
- Step 3 – Create Subscriber-component

# 1. OrderService

```
// order.service.ts

import {Subject} from "rxjs/Subject";
import {Injectable} from "@angular/core";
import {City} from "../model/city.model";

@Injectable()
export class OrderService {
  Stream:Subject<City>;

  constructor() {
    this.Stream = new Subject<City>();
  }
}
```

## 2. Producer component ('Order now' button)

HTML: `<h2>Price of city trip:  
{{ city.price | currency:'EUR':true:'1.2' }}  
<button class="btn btn-lg btn-info"  
 (click)="order(city)">Order Now!</button>  
</h2>`

Class: *// Place order. Emit event for this city.*  
*// Catch the event in city.orders.ts*  
`order(city) {  
 console.log(`City trip booked for: ${this.city.name});  
 this.orderService.Stream.next(city);  
}`

### 3. Subscriber component

*// city.orders.ts – a kind of simple shopping cart,  
// register which city trips are booked.*

```
import ...
```

```
@Component({  
  selector: 'city-orders',  
  template: `  
    <div *ngIf="currentOrders.length > 0">  
      ...  
    </div>  
  `})
```

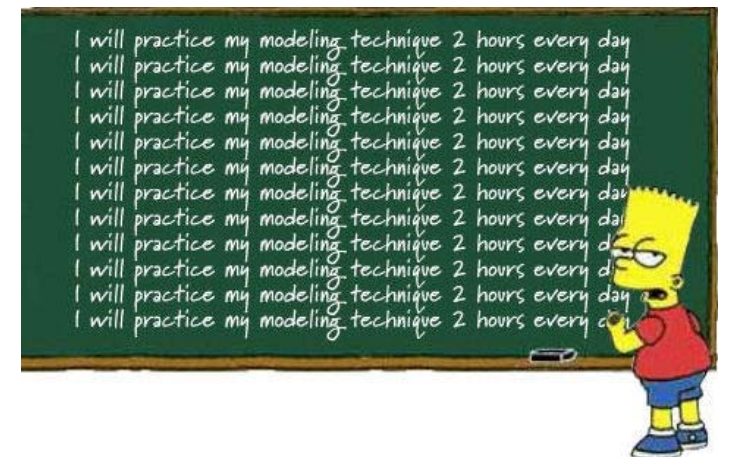
```
export class CityOrders {  
  ...  
  ngOnInit() {  
    this.orderService.Stream  
      .subscribe(  
        (city:City) => this.processOrder(city),  
        (err)=>console.log('Error handling City-order'),  
        ()=>console.log('Complete...')  
      )  
  }  
  ...  
}
```



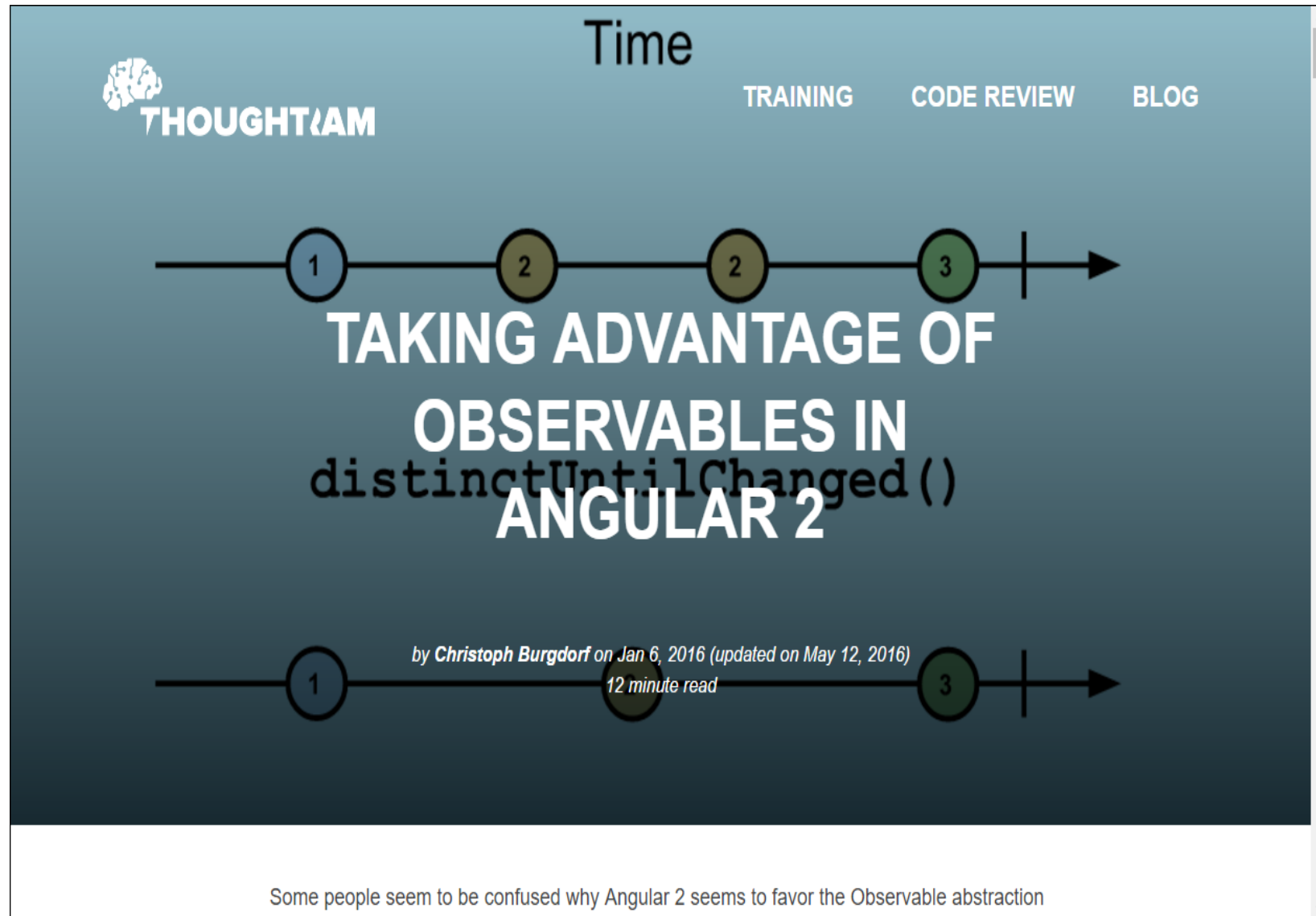
# Checkpoint

- Event Bus : work with 'invisible' Streams and Subject
- There are options on working with Observable Streams.
- (Optional/Advanced: use a Redux-store (for example @ngrx/store))
- Example: /303-pubsub-ordercomponent
- Exercise : 6d) e-commerce application

## Exercise....




# More on observables



<http://blog.thoughttram.io/angular/2016/01/06/taking-advantage-of-observables-in-angular2.html>



My name is [Cory Rylan](#), Senior Front End Engineer at [Vintage Software](#) and [Angular Boot Camp](#) instructor. I specialize in creating fast, scalable, and responsive web applications.

 Follow @SplinterCode

# Angular 2 Observable Data Services

Nov 17, 2015

Updated May 6, 2016 - 8 min read

Angular 2 brings many new concepts that can improve our JavaScript applications. The first new concept to Angular is the use of Observables. Observables are a proposed feature for ES2016 (ES7). I won't go in depth into Observables but will just cover some of the high level concepts. If you want an introduction to Observables check out my screen cast.

## INTRO TO RXJS OBSERVABLES AND ANGULAR 2

The rest of this post will cover more data and application state management in an Angular 2 application. At the time of this writing Angular is on version [Beta 1](#). This post has been updated as of [Beta 15](#). The syntax of how Observables and their

<https://coryrylan.com/blog/angular-2-observable-data-services>

Check out my [Angular 2.0 article series](#)



Home

Most Popular

Most Recent

[Angular](#)

React

Aurelia

JavaScript

NodeJS

MongoDB

Unit Testing

.Net

Q&A

All

## Observables In Angular 2.0

**Author:** [Torgeir Helgevold](#)

Published: Wed Jan 06 2016

Viewed 3375 times

The RxJs community has presented the idea that any series of events can be modeled as one or many asynchronous or synchronous arrays. In the following post I want to explore this by modeling a series of different user inputs as Observables.

I am still learning about Observables and their potential, but I figured it would be interesting to implement a custom text editor, from scratch, using Observables to represent keyboard and mouse events.

Building a perfect text editor is not really the point here, but I want to see if there is any added value from looking at input sequences as Observables. The first step when building a text editor is identifying which input events to support. In my sample I have decided to focus on adding the ability to input and delete characters. Currently I have limited the input

<http://www.syntaxsuccess.com/viewarticle/observables-in-angular-2.0>