Angular Change Detection

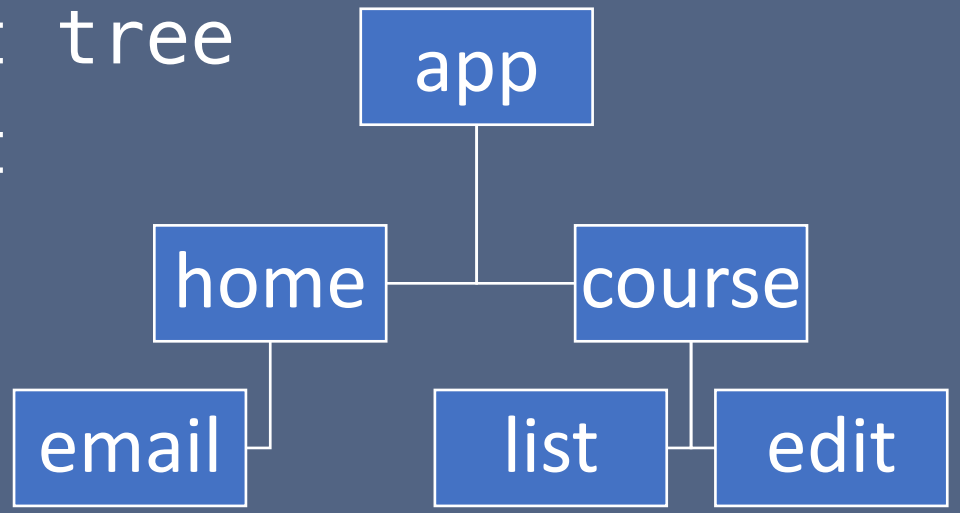
What’s the deal?

What is Change Detection?

- Takes the state of an application
  - Push it to the user interface (DOM)
    - Divs, Tables, Paragraphs, what so ever
- The state can be any kind of
  - Objects
  - Arrays
  - Primitives like Strings, Numbers etc…

Angular Change Detection

- Angular maintains a Component tree
- When the state of a Component changes
  - Angular must update the view
    - Re-create all or part of the HTML DOM



So, what causes changes?

- Three things can cause the state to change
  - DOM Events
    - Click, submit, mouseover, mousedown ...
  - Timers
    - setInterval(), setTimeout()
  - XHR (XMLHttpRequest)
    - Grabbing data from remote servers

Changes 1

course-edit.component.html

```
…
<app-instructor [instructor]="instructor"></app-instructor>
<button (click)="changeInstructor()">Change Instructor</button>
```

course-edit.component.ts

```
export class CourseEditComponent implements OnInit {
  instructor: Instructor = { firstName: Michael' };

  changeInstructor() {
    this.instructor.firstName= 'Chris';
  }
}
```

Changes 2

```
@Component({
selector: 'app-course-list',
templateUrl: './course-list.component.html',
styles: []
})
export class CourseListComponent implements OnInit {
  courses: Course[];

  constructor( private _courseService: CourseService) { }

  ngOnInit() {
    this._courseService.getCourses().subscribe(courses => this.courses =
      courses);
  }

}
```

What do the code have in common?

- They are all asynchronous
  - Every time an asynchronous action occurs
  - Someone tells Angular that a change has ocurred
  - Angular updates the view

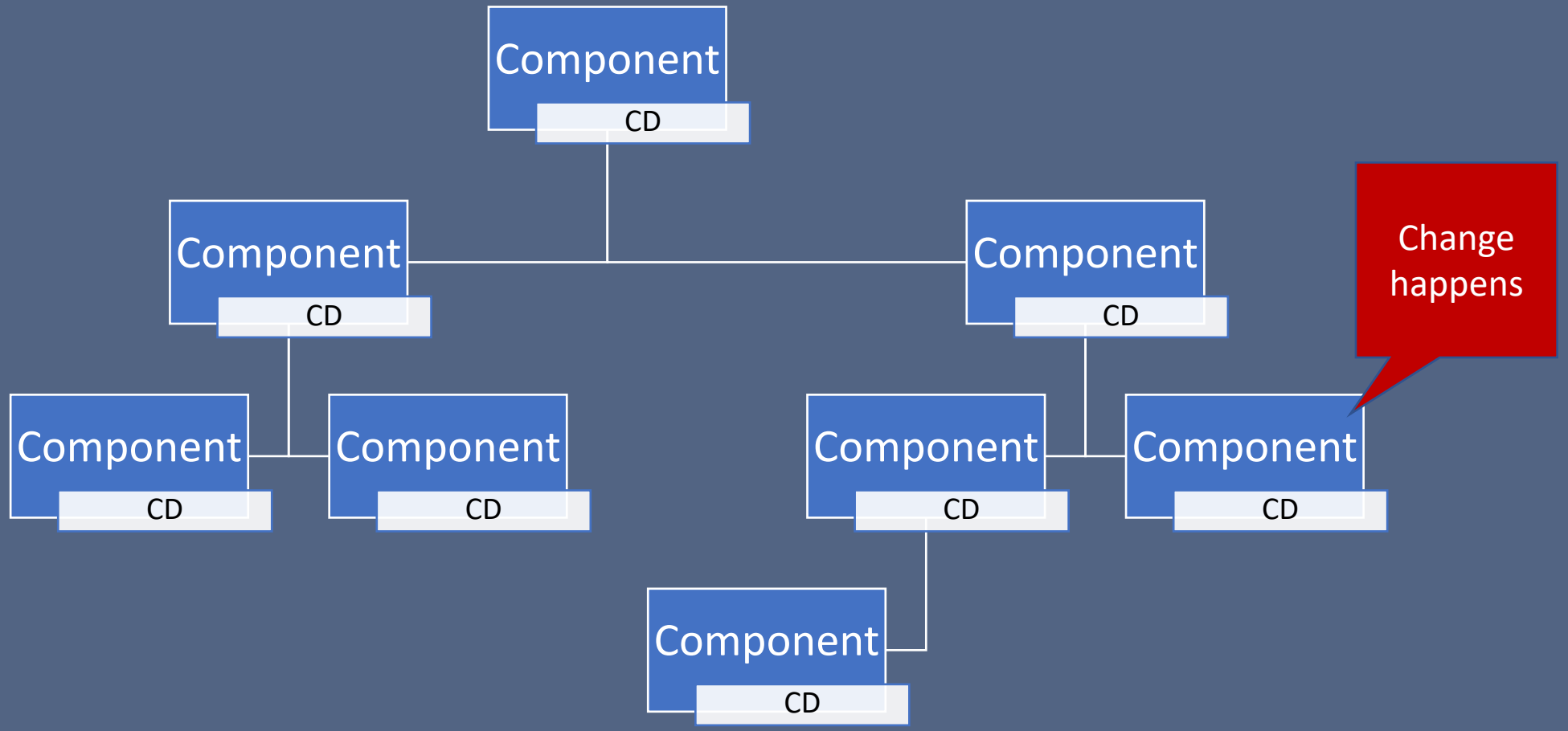
Who tells Angular?

- Is it Magic?
- No, Angular uses Zone
  - Zone is a execution context that persists across asynchronous tasks.
  - A thread-safe environment(storage)
- Specifically it’s own implementation ngZone
  - Angular use in its source code ***ApplicationRef*** to listens to
    - onTurnDone event
    - It executes a tick() function
    - Which in turn performs the change detection

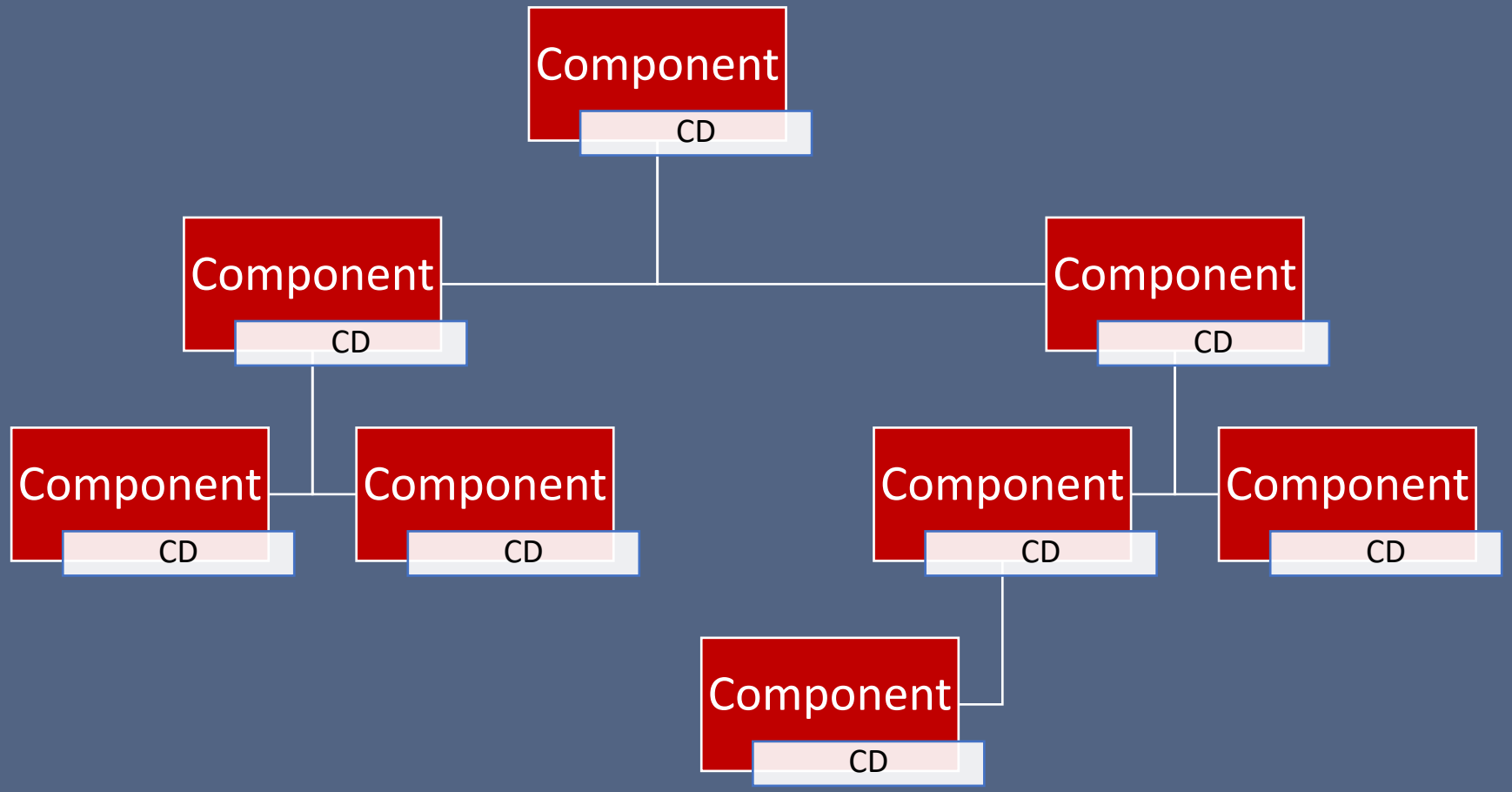
Change Detection

- Important!
- Each component has its own change detector
  - Gives us individual control for each component how and when to run change detection
- Default behavior
  - The change detection flows from top to bottom
  - Regardless of where in the component tree the change happens
  - However it just makes one single pass from top to bottom then the change detection is in a stable state.

Default behavior



Default behavior



Performance

- Even if Angular has to run top to bottom it is scary fast.
  - Angular can perform hundreds of thousands checks within milliseconds
    - Source ”Pascal Precht”
- However in large component trees it could be performance degrading
  - Why should we check every component if we know which one has changed?

Take control

- Use Immutable data structures
- Use Observables
- This gives us some guarantees of when changes or not has happened

Que!

Mutability

- Default CD Behavior
  - Checks every change to every property change in the component
  - So even a small change to a value triggers cd
  - Remember the first code slide

```
changeInstructor() {
    this.instructor.firstName = 'Chris';
}
```

We are changing the property of an object. We are mutating existing value with a new value. Changing the object.

Observables

- Gives us some guarantee of when changes happens
- Does not give us a new reference when changes happens
- However we can subscribe to events fired
  - And react accordingly

So now when we know the difference between mutability and immutability

- Angular's default change detection (again)
  - Angular checks for changes inside an object/component
  - If a value changes the change detection is executed
  - Is very fast, which we said before
- ChangeDetection Strategy
  - We can change cd strategy to only check reference difference
  - Tell the component to use *ChangeDetectionStrategy.OnPush*

```
@Component({
  …
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

Even more Control!

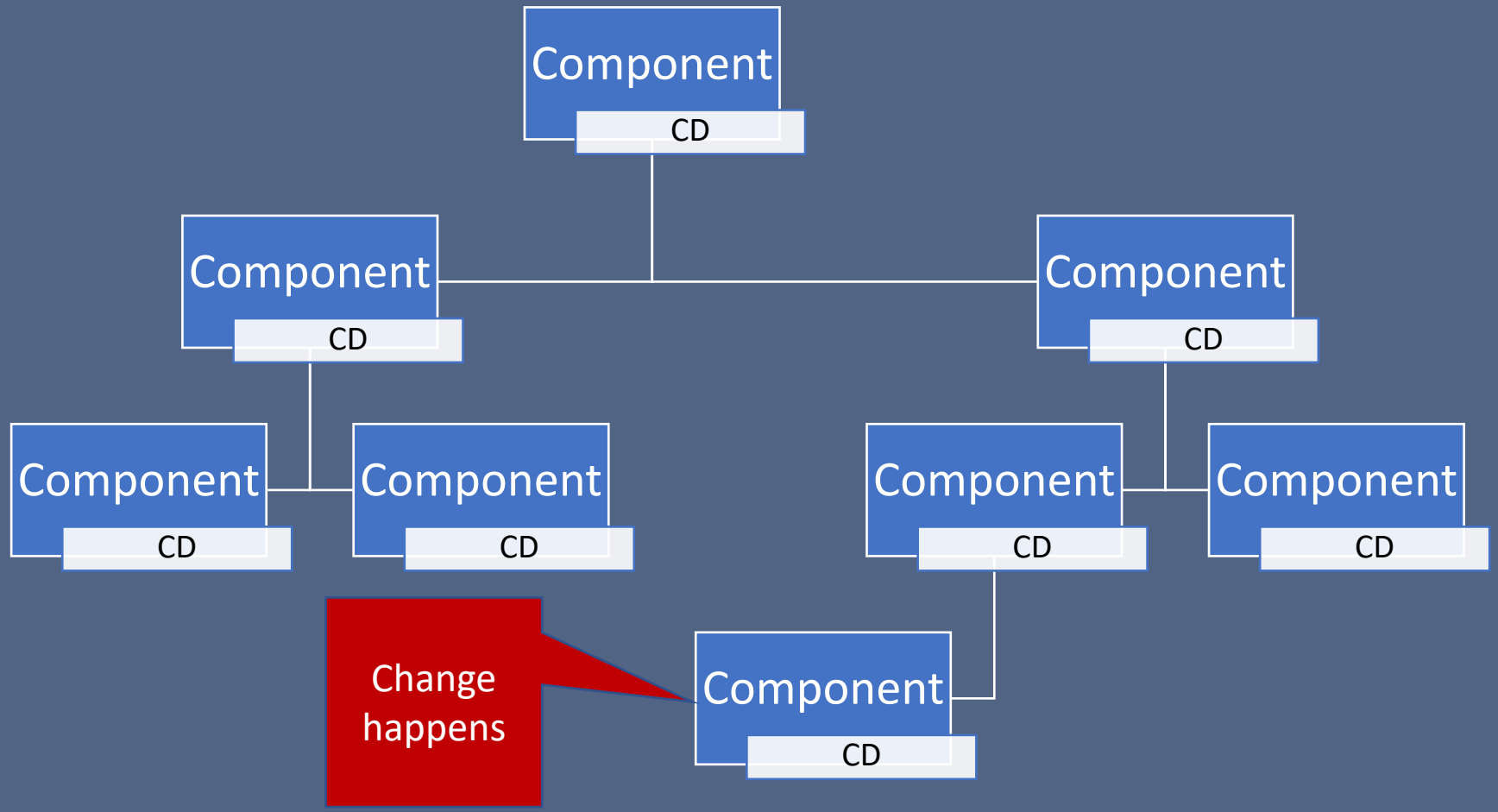
- Even with onPush activated
  - The component tree is traversed top to bottom
  - What if we only want to traverse the subtree for the component that is changed
  - We inject ChangeDetectorRef inside our component

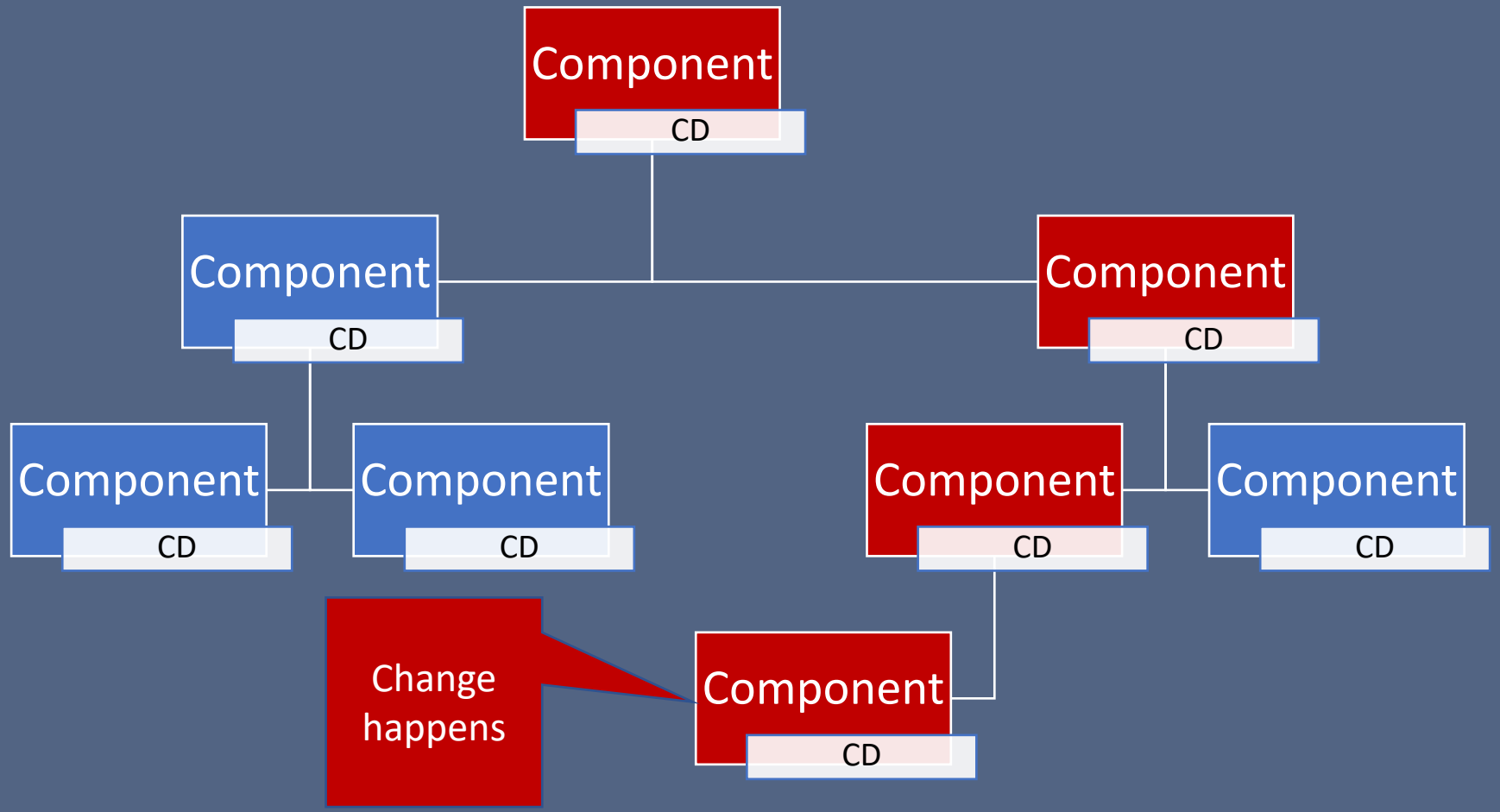
```
@Component({
  …
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class CourseEditComponent implements OnInit {
  …
  constructor(private cd: ChangeDetectorRef) { }
}
```

Even more Control (continued)

```
@Component({
  …
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class CourseEditComponent implements OnInit {
  …
  constructor(private _cd: ChangeDetectorRef) { }

  ngOnInit() {
    this._courseService.getCourse().subscribe(course => {
      this.course = course;
      this._cd.markForCheck();
    );
  }
}
```


Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Change happens
Component
CD


Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Component
CD
Change happens
Component
CD

Wait there is more

```
@Component({
  …
})
export class CourseEditComponent implements OnInit {
  …
  constructor(private _cd: ChangeDetectorRef) {
    _cd.detach();

    setInterval(() => {
      _cd.detectChanges();
    }, 2000);
  }
}
```

That's All