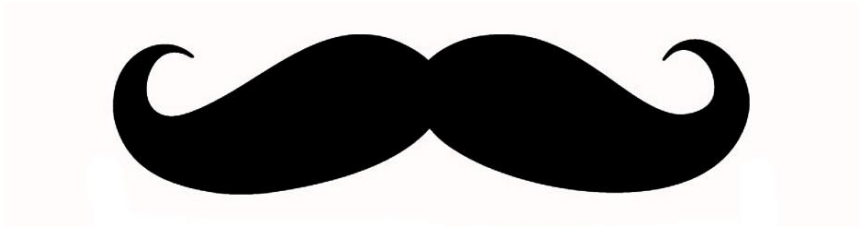


Change Detection

Animé par Mazen Gharbi

▷ La « change detection » est l'une des principales richesses d'Angular ;

▷ C'est le système qui maintient synchronisés le modèle de données et la vue ;

▷ Contrairement au "digest cycle" d'AngularJS, Angular offre un meilleur contrôle de la "change detection" et permet donc de gagner facilement en performance.

▷ Comprenons le fonctionnement

AngularJS

- $scope.a
  - 1 variable = 1 watcher

- Plusieurs contraintes :
  - La variable doit impérativement obligatoirement être déclarée dans un contexte AngularJS (si on passe par un setTimeout « natif », il sera nécessaire d'appliquer le cycle de vérification manuellement)
  - Un watcher est créé à chaque expression dynamique définie dans le template, et on peut se retrouver avec beaucoup… beaucoup de watchers. Cela a des répercutions sur les performances
- A chaque fois qu'AngularJS détecte un changement de variable, il déclenche un **« digest cycle »**

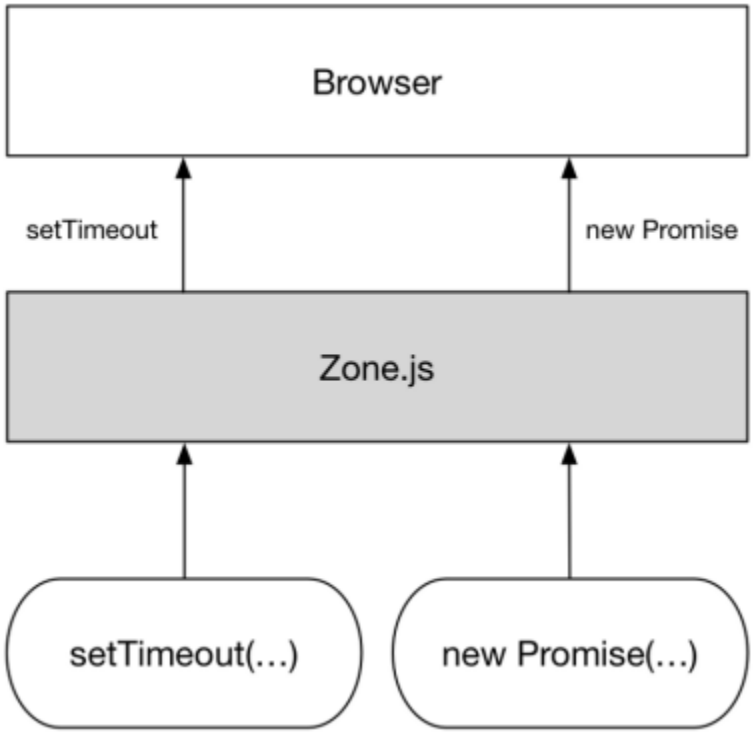
AngularJS

- Evalue l'ensemble des valeurs stockées dans les watchers ;
- Le cycle est lancé **au moins 2 fois** ;

- Comment Angular fonctionne ?

Angular – Zone.js

▷ « Zone.js » on le voit partout, mais qu'est ce que c'est ?



Zone.js

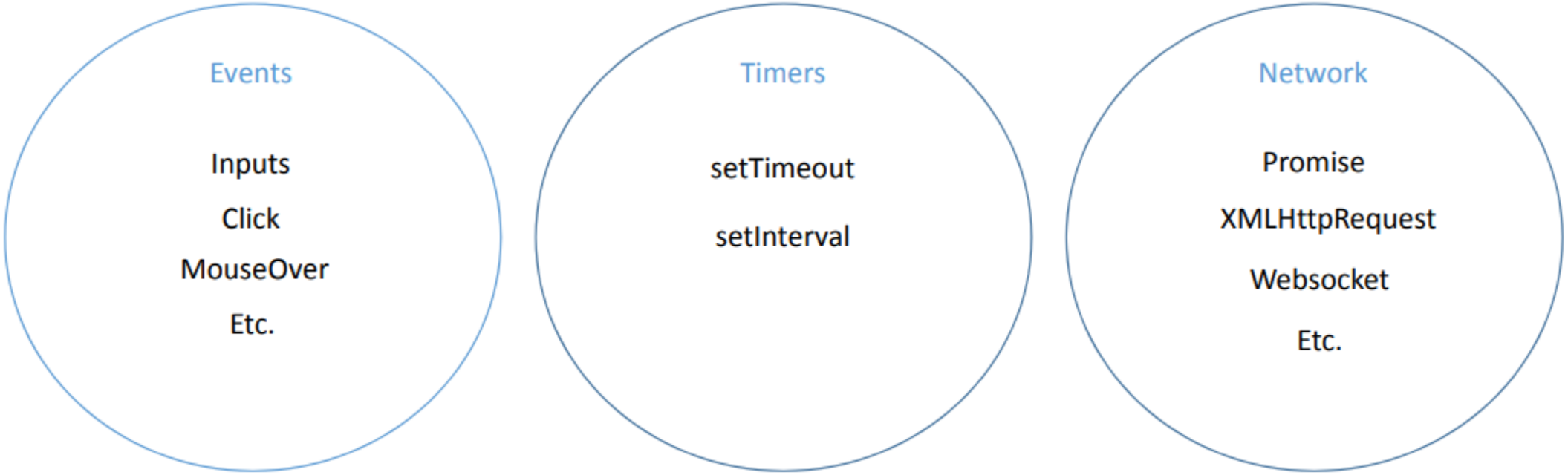
```
const maZone = Zone.current.fork({ name: 'maZone' });

maZone(() => {
    const value = getValue();
    updateUser(user, value); // asynchrone
});
```

▷ Plusieurs fonctions sont fournies pour réagir à un nouvel état de la zone :

onInvoke:	Sera appelé juste avant l'éxecution de la zone
onHasTask:	Appelé après l'éxécution du code de la zone
onHandleError:	Appelé dès lors qu'une erreur est lancée dans la zone
onFork:	Lancé au moment de la création de la zone

Fournis par défaut avec zone.js

Zone.js avec Angular



Zone.js

▷ NgZone - La surcouche Angular pour zone.js

onTurnStart: Notifie les listeners avant chaque cycle

onTurnDone: Notifie les listeners après la fin de chaque évènements du cycle

onEventDone: Appelé après l'appel du dernier onTurnDone

Un fork des zones créé par Angular

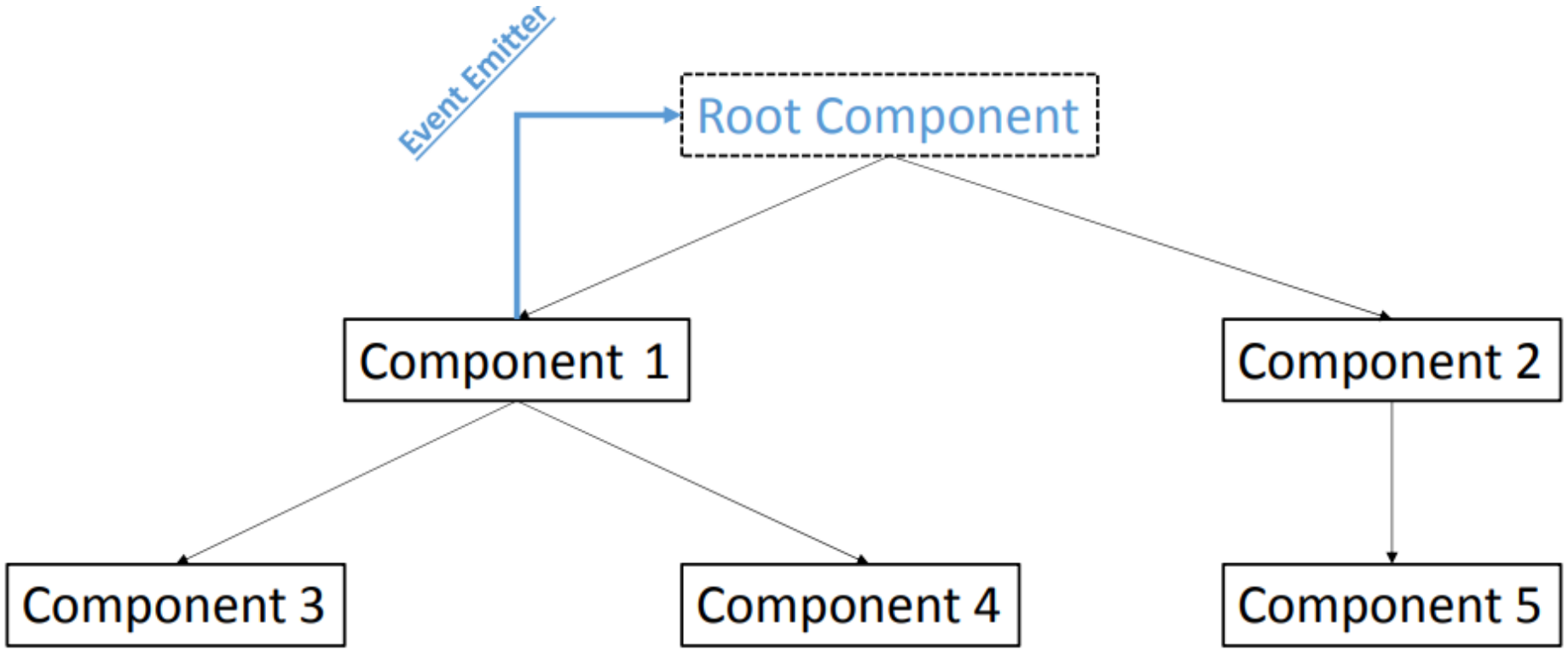
▷ Angular utilise les observable pour la gestion de ses évènements. Pour être notifié, il sera nécessaire de créer un Observateur qui viendra s'abonner aux observables mis à disposition ;

Quand sont remplis les zones ?

▷ Au chargement de notre application, Zone.js va se charger de patcher l'ensemble des méthodes asynchrones pour les associées à des zones. Donc à chaque call asynchrone que l'on effectuera, on appellera en fait la version « patchée » afin qu'Angular puisse savoir exactement lorsque ce code se termine et ainsi lancer le cycle de comparaison ;

▷ C'est le principe du monkey-patching

Change Detector



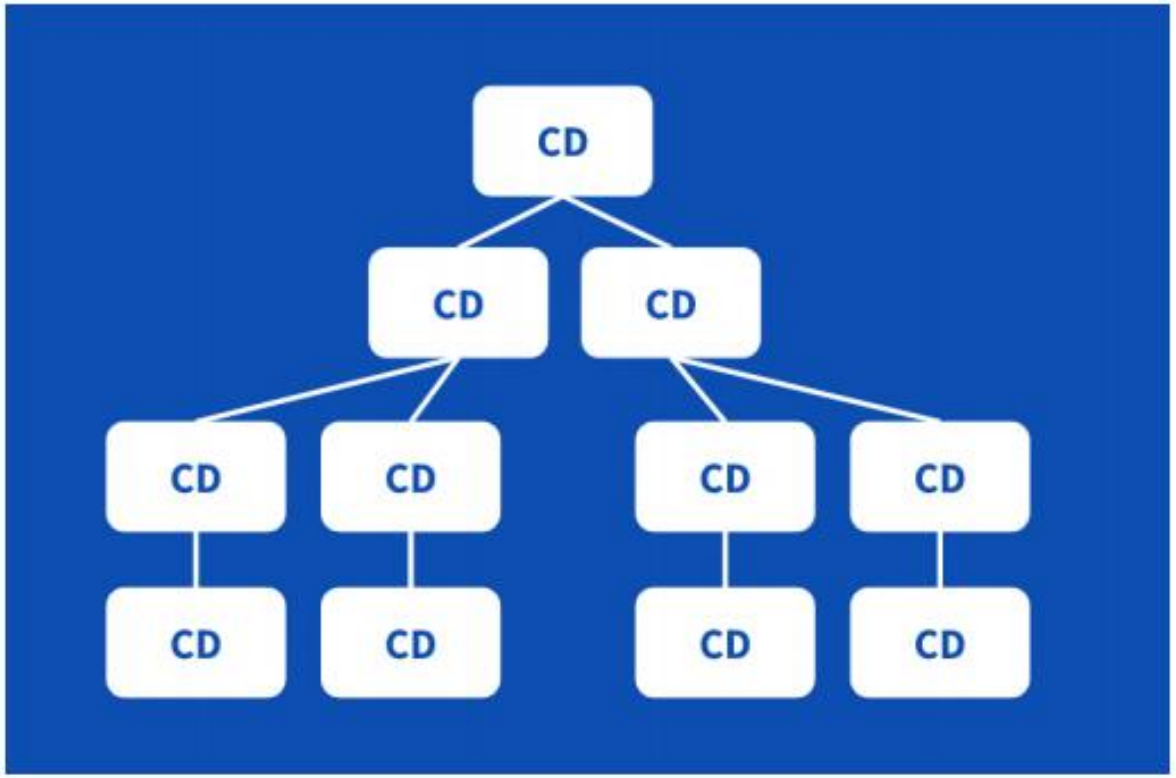
Manipuler les Change Detector

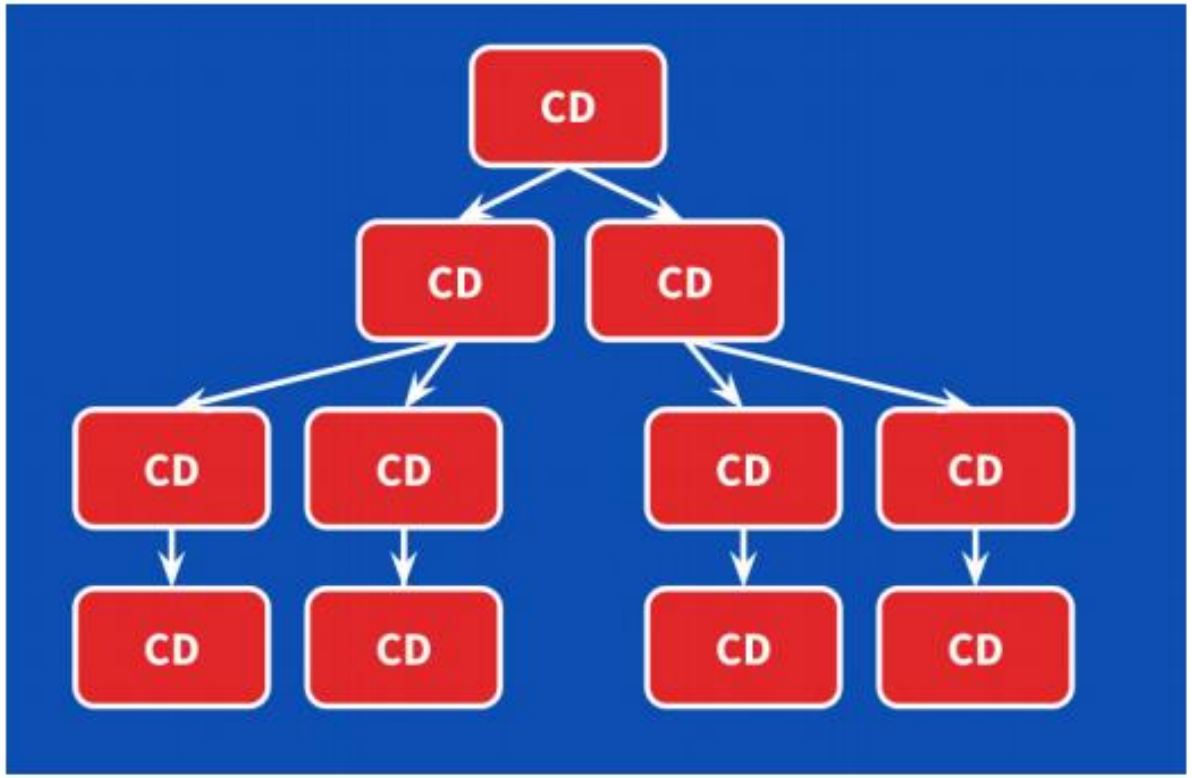
▷ Cela peut se faire directement via les attributs du component. Nous avons 2 types de changeDetection, Onpush et Default.

▷ Default active toutes les zones là ou onPush n'active que la zone relative aux évènements utilisateurs (clavier / souris / etc.)

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css'],
    changeDetection: ChangeDetectionStrategy.OnPush
})
```

Change Detector





VM Javascript

« Si un programme appelle une méthode beaucoup de fois avec le même type d'objet, la VM se rappelera de quelle façon elle évalue les propriétés des objets en question ».

▷ Le cache se crée au fur et à mesure, et la VM commence toujours pas vérifier dans le cache si elle connait ou non le type d'objet qu'elle reçoit. Si c'est le cas, elle pourra donc optimiser la méthode au chargement

Il faut donc respecter certaines conditions pour optimiser l'utilisation de ce cache

Polymorphe vs Monomorphe

▷ Lorsque les arguments ont toujours la même forme, on dit que le cache est monomorphique. A l'inverse, lorsqu'ils ont des formes différentes, on le qualifie de polymorphique.

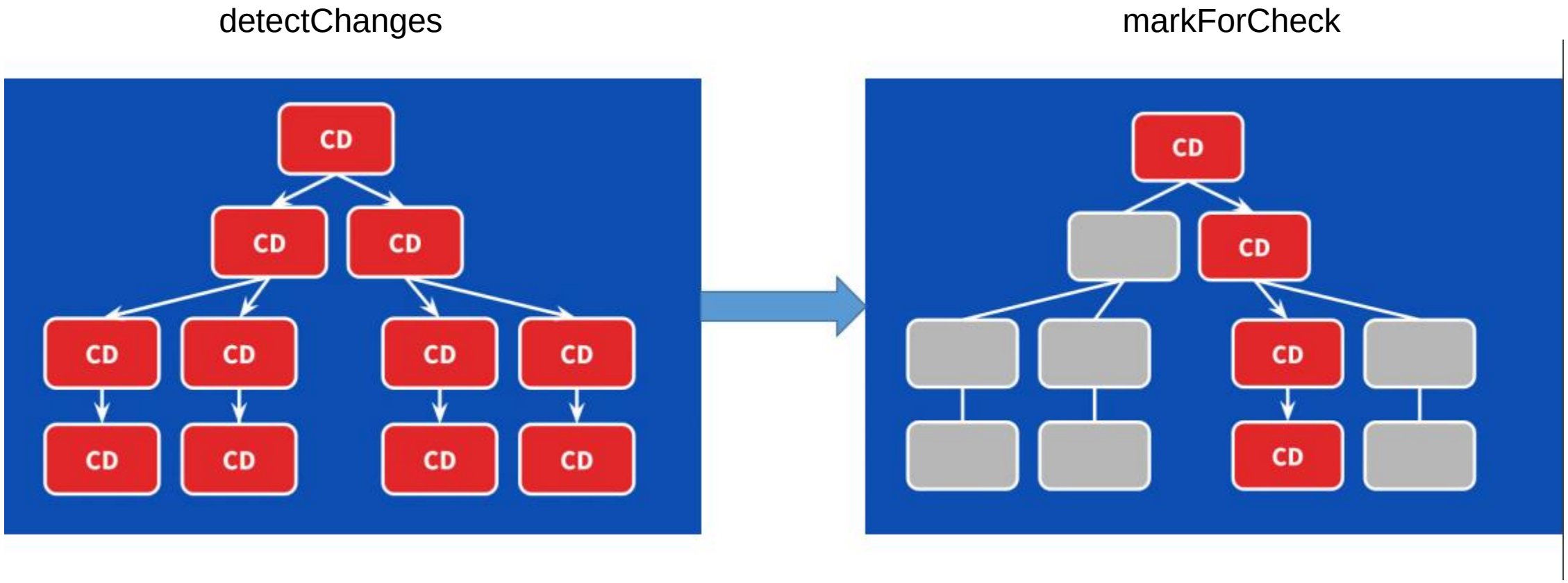
ChangeDetector

app.component.ts

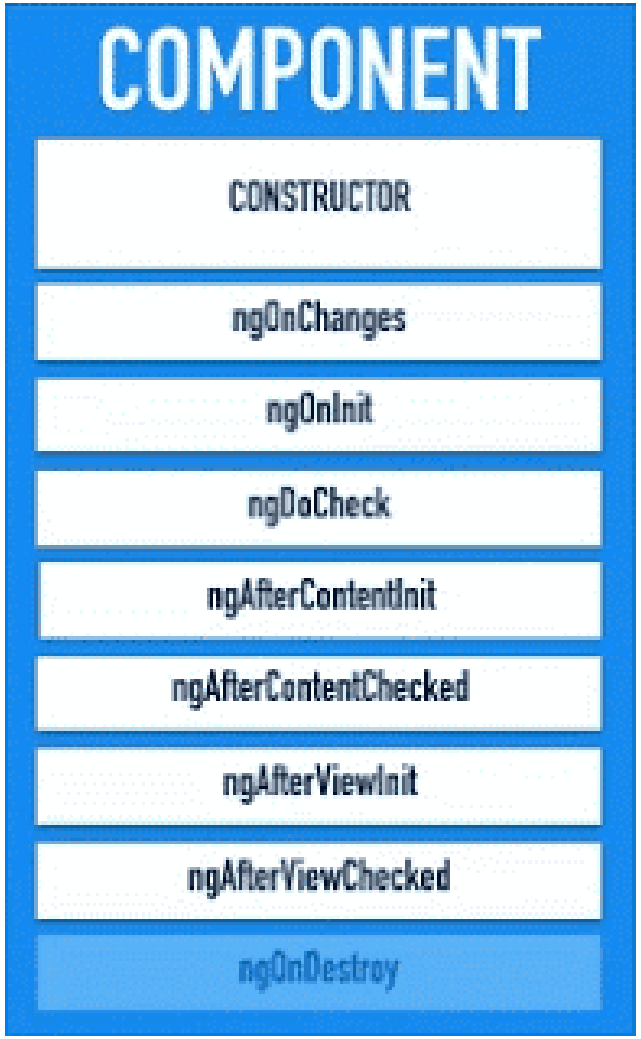
```
@Component({
    selector: 'my-app',
    template: `Number of ticks: {{ numberOfTicks }}`,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class AppComponent  {
    numberOfTicks = 0;

    constructor(private ref: ChangeDetectorRef) {
        setInterval(() => {
            this.numberOfTicks ++
            this.ref.markForCheck();
        }, 1000);
    }
}
```

https://stackblitz.com/edit/angular-macademia-change-detection

ChangeDetector



Cycle de vie d'un composant



- ngOnChanges : Déclenchée quand une valeur d' "Input/Output" change.
- ngOnInit : Déclenchée après le premier `ngOnChanges`.
- ngDoCheck : Déclenchée à chaque "change detection". Il permet d'implémenter une "change detection" personnalisée.
- ngAfterContentInit : Déclenchée quand le contenu projeté (Cf. Projection) est initialisé.
- ngAfterContentChecked : Déclenchée après chaque "change detection" sur le contenu projeté (Cf. Projection).
- ngAfterViewInit : Déclenchée après l'initialisation de la vue.
- ngAfterVIewChecked : Déclenchée après chaque "change detection" de la vue du composant.
- ngOnDestroy : Déclenchée juste avant que le composant ne soit détruit.

Questions