Change Detection and Signals

What is Change Detection ?

Change Detection Processes

Process 1: View Checking

Synchronization of the component view with the data model

Process 2: Re-run Process 1

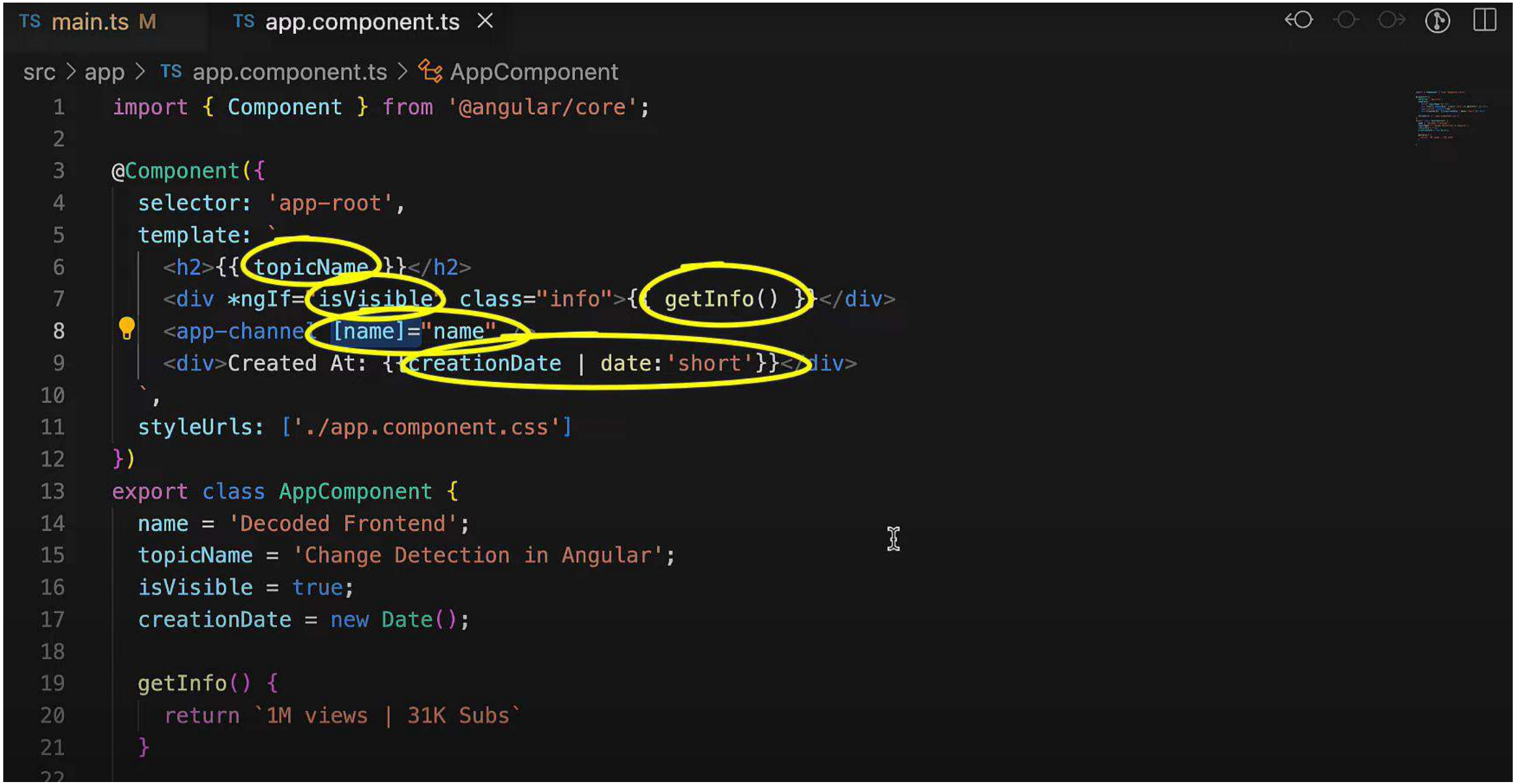
Automatically re-execute the View Checking when application state might change

Disabling Zone.js

src > ts main.ts > ngZone

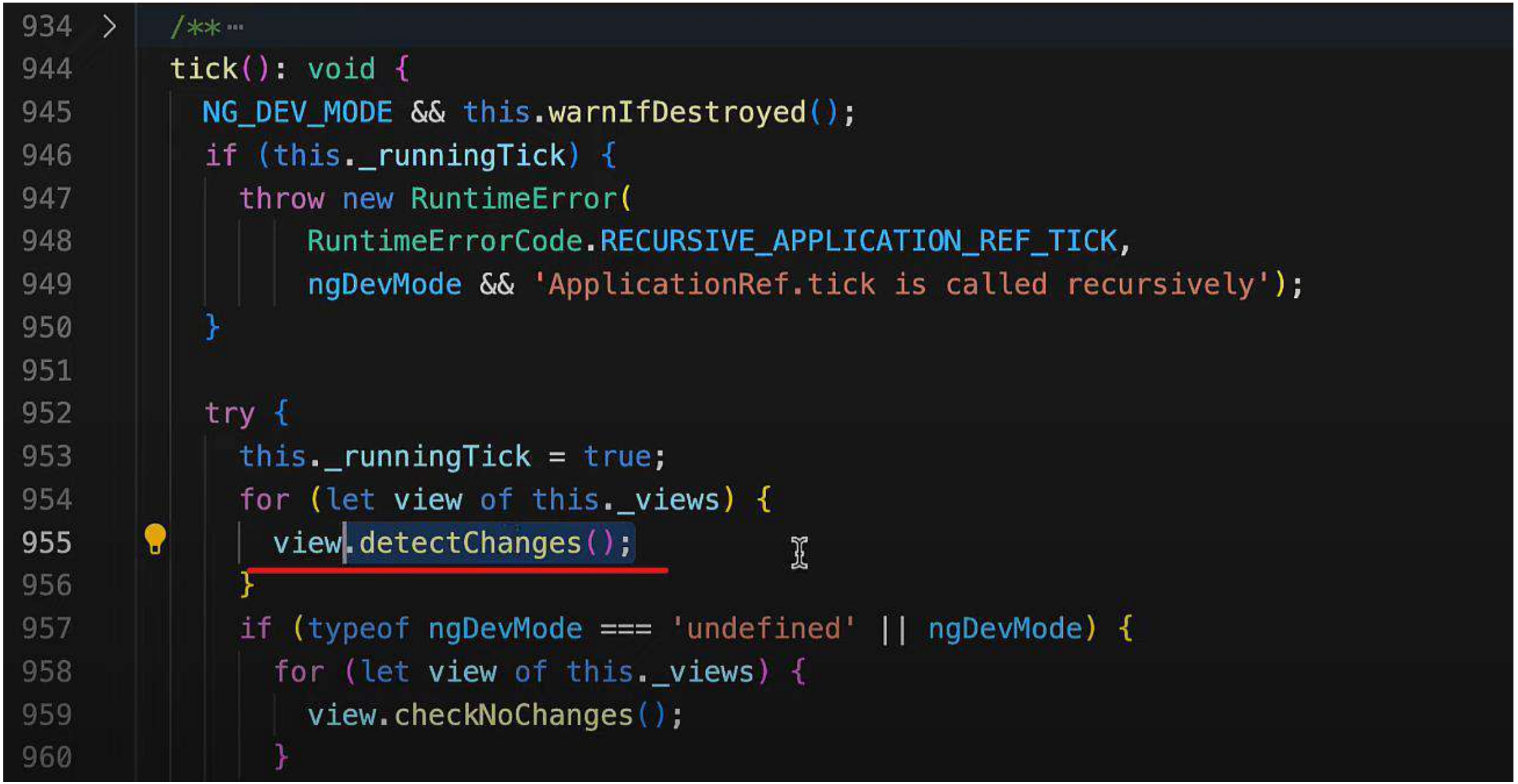
```ts
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';


platformBrowserDynamic().bootstrapModule(AppModule, { ngZone: 'noop' })
  .catch(err => console.error(err));

```

View Checking


main.ts M
app.component.ts
src > app > app.component.ts > AppComponent
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
template: `
<h2>{{ topicName }}</h2>
<div *ngIf="isVisible" class="info">{{ getInfo() }}
<app-channel [name]="name" />
Created At: {{creationDate | date:'short'}}
`,
styleUrls: ['./app.component.css']
})
export class AppComponent {
name = 'Decoded Frontend';
topicName = 'Change Detection in Angular';
isVisible = true;
creationDate = new Date();
getInfo() {
return `1M views | 31K Subs`
}

```
18
19    constructor() {
20      setTimeout(() => {
21        this.topicName = 'ZoneJS in Angular';
22        console.log('Topic name changed to: ', this.topicName);
23      }, 3000);
24    }
25
```

```
19    constructor(private cdr: ChangeDetectorRef) {
20      setTimeout(() => {
21        this.topicName = 'ZoneJS in Angular';
22        console.log('Topic name changed to: ', this.topicName);
23        this.cdr.detectChanges();
24      }, 3000);
25    }
```


/**…
tick(): void {
NG_DEV_MODE && this.warnIfDestroyed();
if (this._runningTick) {
throw new RuntimeError(
RuntimeErrorCode.RECURSIVE_APPLICATION_REF_TICK,
ngDevMode && 'ApplicationRef.tick is called recursively');
}
try {
this._runningTick = true;
for (let view of this._views) {
view.detectChanges();
}
if (typeof ngDevMode === 'undefined' || ngDevMode) {
for (let view of this._views) {
view.checkNoChanges();
}

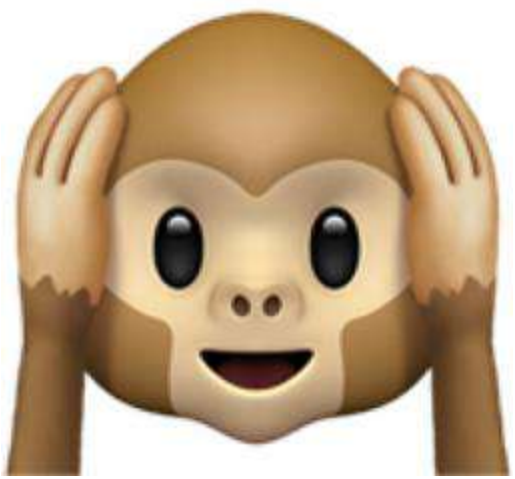
Re-run View Checking

Zone.js

1. setTimeout, setInterval, etc is fired
2. Handling events like click, focus, etc
3. When HTTP request completes

```
1  var originalDelegate = window.setTimeout;
2  window.setTimeout = function(callback, delay) {
3    var zone = Zone.current;
4    originalDelegate.call(window, function() {
5      zone.run(callback);
6    }, delay);
7  }
```

zone-monkey-patch.js hosted with ❤ by GitHub view raw



```
export class AppComponent implements OnInit {

  constructor(private ngZone: NgZone) { }

  ngOnInit() {
    this.ngZone.runOutsideAngular(() => {
      setTimeout(() => {
        // Update component data
        // without triggering change detection!
        this.ngZone.run(() => {
          // Any update in here will
          // trigger change detection!
        })
      });
    });
  }
}
```

```
340     // But this fix causes some issues in g3, so this fix will be
341     // launched in another PR.
342     if (zone._nesting == 0 && !zone.hasPendingMicrotasks && !zone.isStable) {
343       try {
344         zone._nesting++;
345         zone.onMicrotaskEmpty.emit(null);
346       } finally {
347         zone._nesting--;
348         if (!zone.hasPendingMicrotasks) {
349           try {
350             zone.runOutsideAngular(() => zone.onStable.emit(null));
351           } finally {
352             zone.isStable = true;
353           }
354         }
355       }
356     }
357   }
```

```
Andrew Scott, last month | 1 author (Andrew Scott)
1205  @Injectable({providedIn: 'root'})
1206  export class NgZoneChangeDetectionScheduler {
1207    private readonly zone = inject(NgZone);
1208    private readonly applicationRef = inject(ApplicationRef);
1209
1210    private _onMicrotaskEmptySubscription?: Subscription;
1211
1212    initialize(): void {
1213      if (this._onMicrotaskEmptySubscription) {
1214        return;
1215      }
1216
1217      this._onMicrotaskEmptySubscription = this.zone.onMicrotaskEmpty.subscribe({
1218        next: () => {
1219          this.zone.run(() => {
1220            this.applicationRef.tick();
1221          });
1222        }
1223      });
1224    }
```

src > app > TS app.component.ts > AppComponent > doSomething

```ts
import { ChangeDetectorRef, Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <h2 (click)="doSomething()">{{ topicName }}</h2>
    <app-channel [name]="name" />
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  name = 'Decoded Frontend';
  topicName = 'Change Detection in Angular';

  doSomething() {}

  constructor(public cdr: ChangeDetectorRef) {
```

Change Detection Strategy

onPush

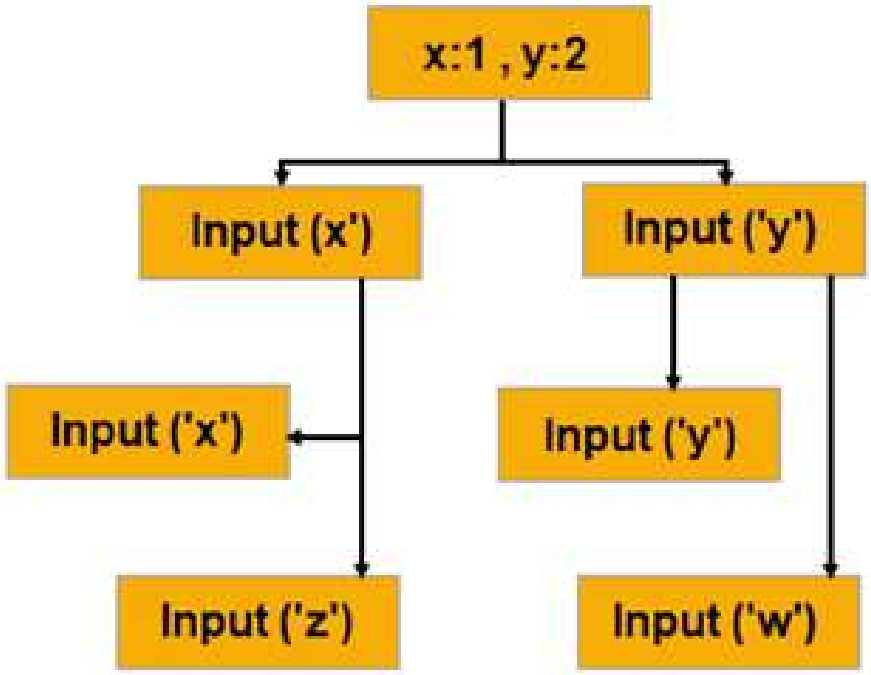
1. The Input reference changes;

2. An event originated from the component or one of its children;

3. Run change detection explicitly (`componentRef.markForCheck()`);

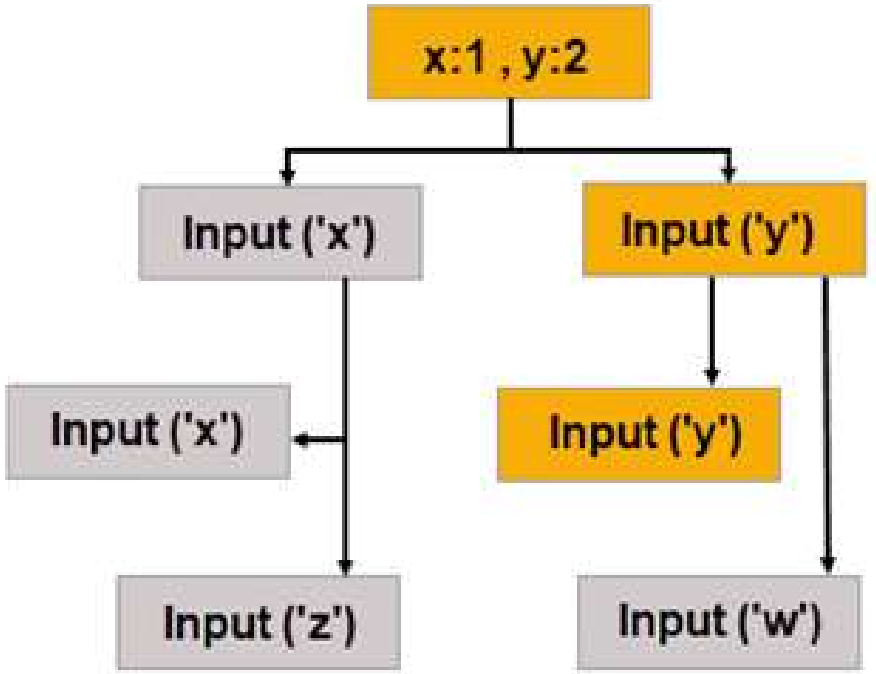
4. Use the async pipe in the view.

Default Angular Detection Mode

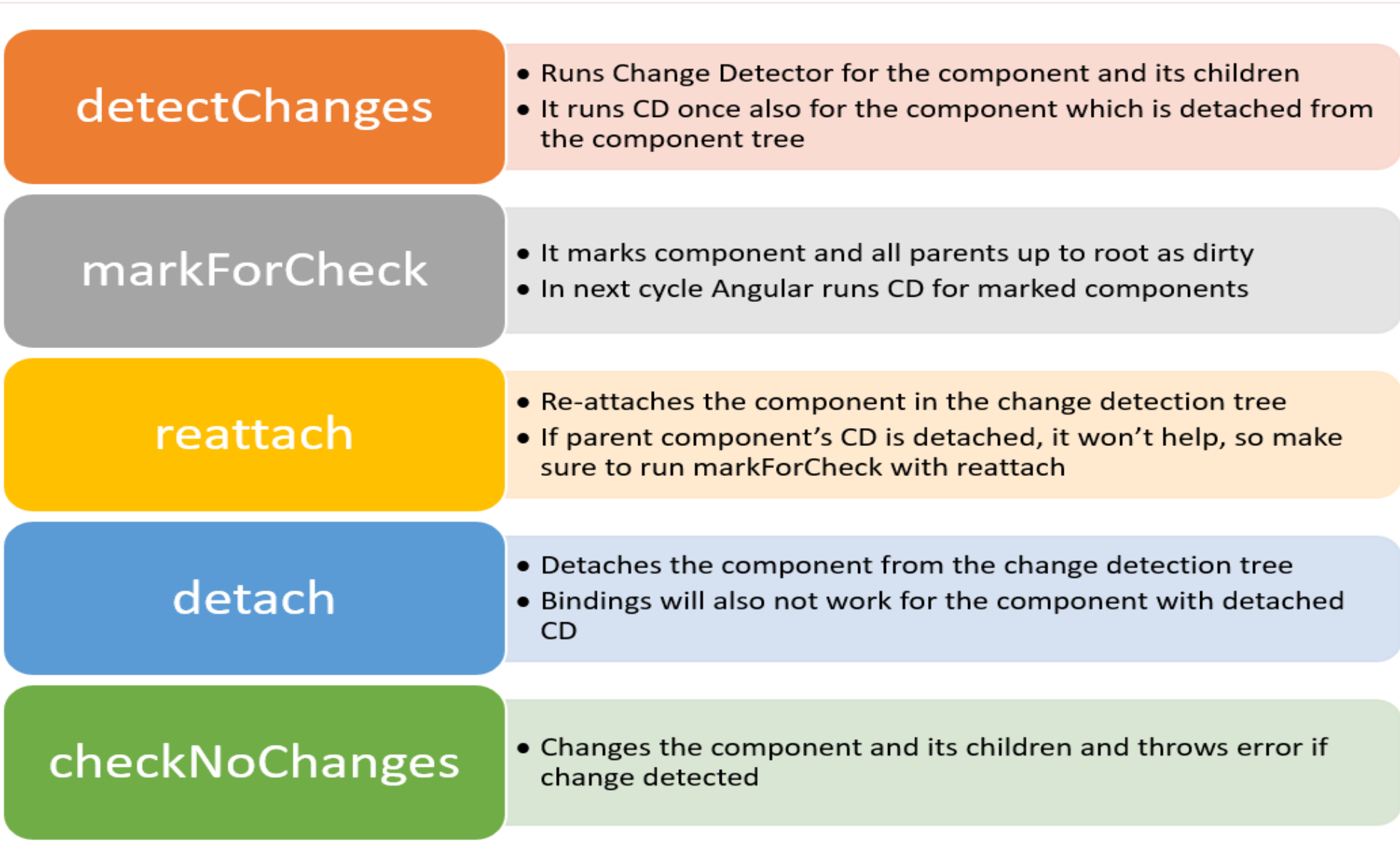


All the components in the tree are checked for changes when the **'y'** property is changed to a new value.

OnPush Angular Detection Mode

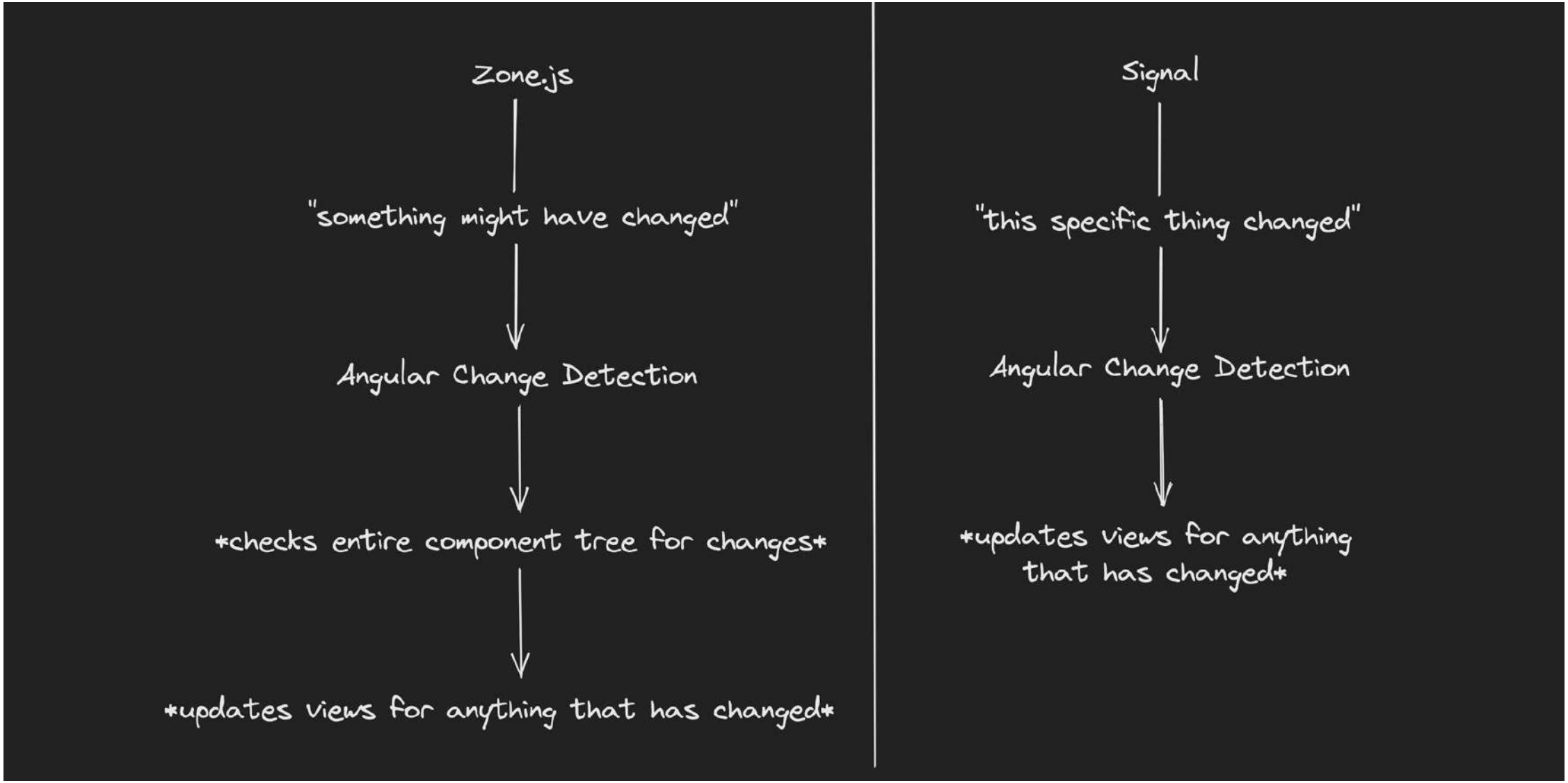


Only the children of the input of the **'y'** component are checked for changes.


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

Signals

```ts
export class AppComponent {

  count = signal(0);
  count$ = new BehaviorSubject(0);

  double = computed(() => this.count() * 2);
  double$ = this.count$.pipe(
    map(count => count * 2)
  )

  changeCount() {
    this.count.set(5);
    this.count$.next(5);
  }
}
```


Zone.js
"something might have changed"
Angular Change Detection
checks entire component tree for changes
updates views for anything that has changed
Signal
"this specific thing changed"
Angular Change Detection
updates views for anything that has changed

```
count = 0; // will get updated
doubleCount = this.count * 2; // will always be 0
doubleDoubleCount = this.doubleCount * 2; // will always be 0
```

```
count = signal(0);
doubleCount = computed(() => this.count() * 2);
doubleDoubleCount = computed(() => this.doubleCount() * 2);
```

```
count$ = new BehaviorSubject(0);
doubleCount$ = this.count$.pipe(map((count) => count * 2));
doubleDoubleCount$ = this.doubleCount$.pipe(map((doubleCount) => doubleCount * 2))
```

```ts
@Component({
  selector: 'debounced',
  standalone: true,
  imports: [CommonModule],
  template: `
    <p>Hello from {{fullName$ | async}}!</p>
    <p>{{fullNameCounter}}</p>

    <button (click)="changeName()">Change Name</button>
  `,
})
export class DebouncedComponent {
  public firstName = new BehaviorSubject('Peter');
  public lastName = new BehaviorSubject('Parker');

  public fullNameCounter = 0;

  public fullName$ = combineLatest([this.firstName, this.lastName]).pipe(
    debounceTime(0),
    tap(() => {
      this.fullNameCounter++;
    }),
    map(([firstName, lastName]) => `${firstName} ${lastName}`)
  );

  public changeName() {
    this.firstName.next('Debounced Spider');
    this.lastName.next('Man');
  }
}
```

```ts
@Component({
  selector: 'my-app',
  standalone: true,
  template: `
    <p>{{ fullName() }}</p>
    <p>{{signalCounter}}</p>
    <button (click)="changeName()">Increase</button>
  `,
})
export class App {
  firstName = signal('Peter');
  lastName = signal('Parker');

  signalCounter = 0;

  fullName = computed(() => {
    this.signalCounter++;
    console.log('signal name change');
    return `${this.firstName()} ${this.lastName()}`;
  });

  changeName() {
    this.firstName.set('Signal Spider');
    this.lastName.set('Man');
  }
}
```

```ts
const counter = signal(0);

counter.set(2);
counter.update(count => count + 1);
```

```ts
const todoList = signal<Todo[]>([]);

todoList.mutate(list => {
    list.push({title: 'One more task', completed: false});
});
```

```ts
const counter = signal(0);
effect(() => console.log('The counter is:', counter()));
// The counter is: 0
```

```ts
const counter = signal(0);

// Automatically updates when `counter` changes:
const isEven = computed(() => counter() % 2 === 0);
```

```
@Component({
  standalone: true,
  template:`{{ counter() }}`,
})
export class FooComponent {
  counter$ = interval(1000).pipe(startWith(0));
  counter = toSignal(this.counter$, { requireSync: true });
}
```

```
@Component({
  selector: 'foo',
  standalone: true,
  template:`{{ counter() }}`,
})
export class FooComponent {
  counter$ = interval(1000);
  counter: Signal<number | undefined>;
  private injector = inject(Injector);

  ngOnInit() {
    this.counter = toSignal(this.counter$, { injector: this.injector } );
  }
}
```

The End !!