WHAT IS NG-ZONE AND HOW IT TRIGGER CHANGE DETECTION IN ANGULAR

Tu Hoang

ABOUT ME

Hi, My name is Tu

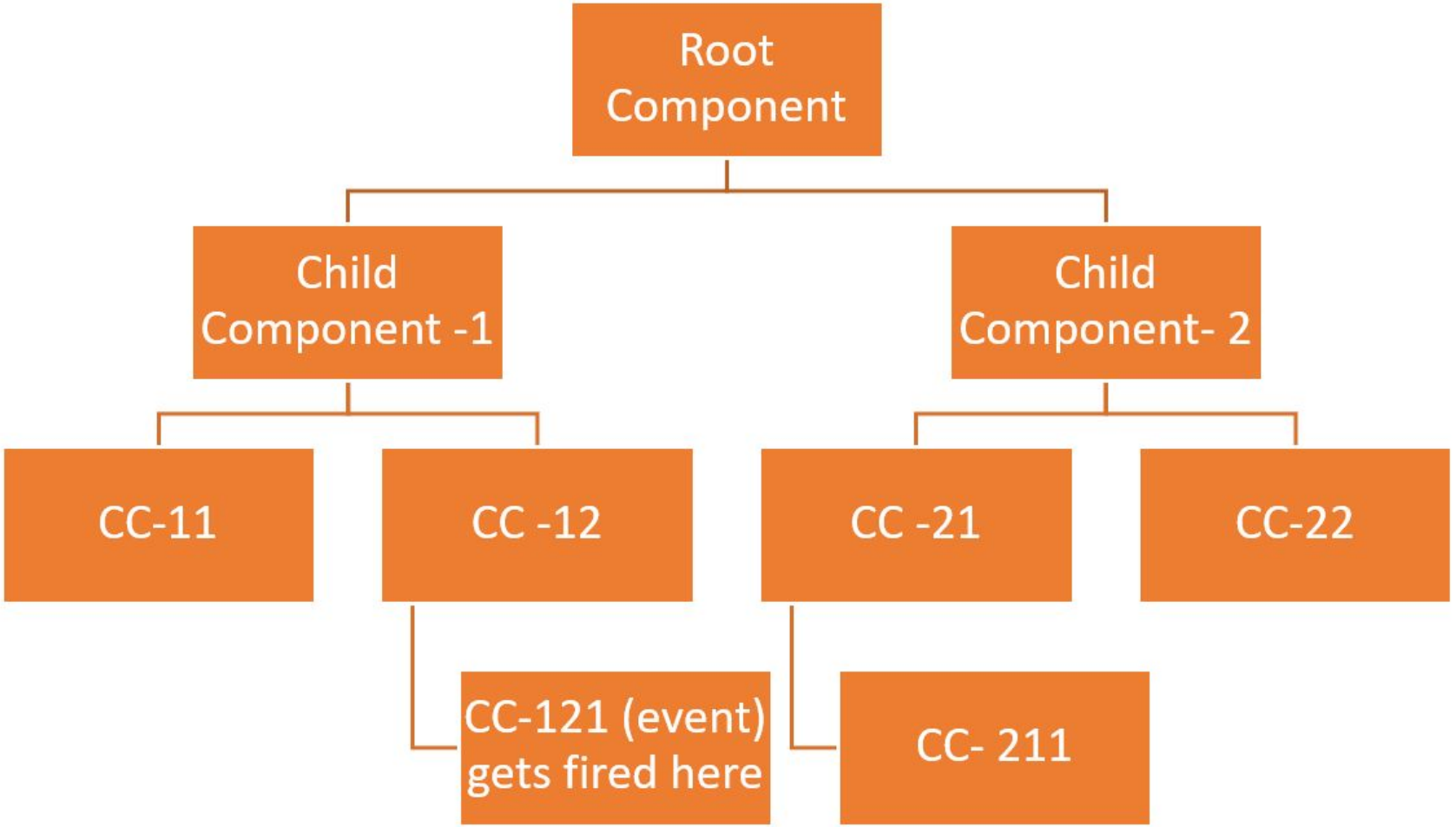
- Software Engineer at **Tagform**
- https://andyt2503.github.io/

OUTLINE

1. What is Change Detection?
2. What is NgZone?
3. How NgZone trigger Change Detection?

WHAT IS CHANGE DETECTION?

- Change detection is the process through which Angular checks to see whether your application state has changed, and if any DOM needs to be updated.
- At a high level, Angular walks your components from top to bottom, looking for changes.


Root Component
Child Component -1
Child Component- 2
CC-11
CC -12
CC -21
CC-22
CC-121 (event) gets fired here
CC- 211

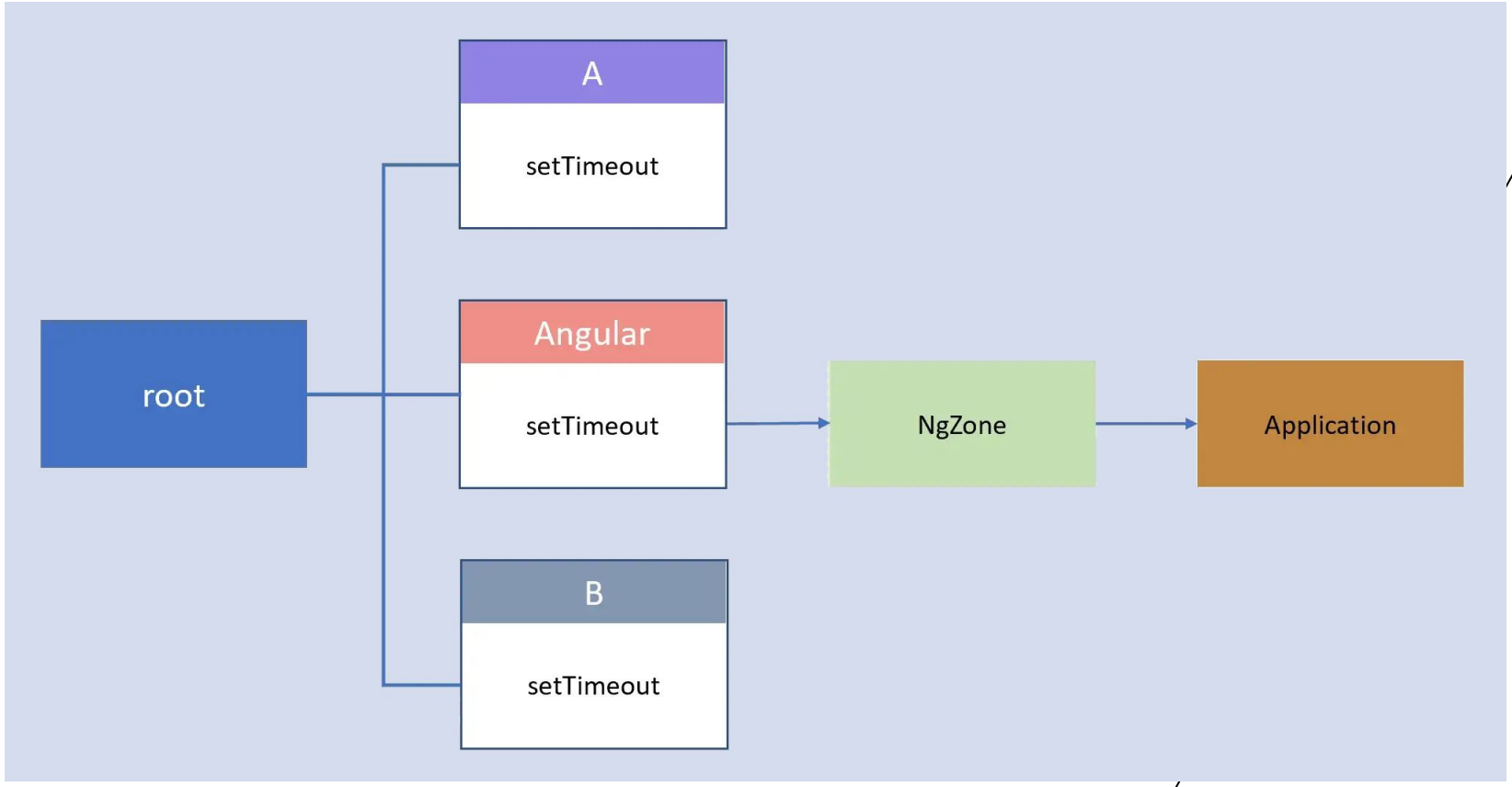
WHEN ANGULAR TRIGGER CHANGE DETECTION?

- Component initialization.
- The DOM event listener.
- HTTP Data Request.
- Invoke some macroTasks such as setTimeout() or setInterval().
- Invoke some microTasks such as Promise.then().
- Other async operations like WebSocket.onmessage() and Canvas.toBlob().

WHAT IS ZONE.JS?

- A **Zone** is an execution context that persists across async tasks
- **Zone.js** provide a mechanism to intercept the scheduling and calling of asynchronous operations
- Interceptor logic can execute additional code before or after the task.
- These rules are defined individually for each zone when it’s being created.

WHAT IS ZONE.JS?



WHAT IS NGZONE?

```
export class NgZone {
  constructor(...) {
    forkInnerZoneWithAngularBehavior(self);
  }
}

function forkInnerZoneWithAngularBehavior(zone: NgZonePrivate) {
  zone._inner = zone._inner.fork({ ... });
}
```

WHAT IS NGZONE?

```
function forkInnerZoneWithAngularBehavior(zone: NgZonePrivate) {
  const delayChangeDetectionForEventsDelegate = () => {
    delayChangeDetectionForEvents(zone);
  };
  zone._inner = zone._inner.fork({
    name: 'angular',
    properties: <any>{'isAngularZone': true},
    onInvokeTask: ···
        },

    onInvoke: ···
        },

    onHasTask:
        (delegate: ZoneDelegate, current: Zone, target: Zone, hasTaskState: HasTaskState) => {···
        },

    onHandleError: (delegate: ZoneDelegate, current: Zone, target: Zone, error: any): boolean => {···
    }
  });
}
```

WHAT IS NGZONE?

HOOKS	DETAILS
onInvokeTask	Triggers when the callback of asynchronous task is about to run
onHasTask	Triggers when the status of one kind of task inside a zone changes from stable to unstable or from unstable to stable
onInvoke	Triggers when a synchronous function is going to run in the zone.
onHandleError	Triggers when when asynchronous task has error

HOW NGZONE TRIGGER CHANGE DETECTION?

```
export class NgZone {
    constructor({ enableLongStackTrace, shouldCoalesceEventChangeDetection, shouldCoalesceRunChangeDetection }: {
        enableLongStackTrace?: boolean | undefined;
        shouldCoalesceEventChangeDetection?: boolean | undefined;
        shouldCoalesceRunChangeDetection?: boolean | undefined;
    });
    // (undocumented)
    static assertInAngularZone(): void;
    // (undocumented)
    static assertNotInAngularZone(): void;
    // (undocumented)
    readonly hasPendingMacrotasks: boolean;
    // (undocumented)
    readonly hasPendingMicrotasks: boolean;
    // (undocumented)
    static isInAngularZone(): boolean;
    readonly isStable: boolean;
    readonly onError: EventEmitter<any>;
    readonly onMicrotaskEmpty: EventEmitter<any>;
    readonly onStable: EventEmitter<any>;
    readonly onUnstable: EventEmitter<any>;
    run<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[]): T;
    runGuarded<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[]): T;
    runOutsideAngular<T>(fn: (...args: any[]) => T): T;
    runTask<T>(fn: (...args: any[]) => T, applyThis?: any, applyArgs?: any[], name?: string): T;
}
```

```
onInvokeTask:
        (delegate: ZoneDelegate, current: Zone, target: Zone, task: Task, applyThis: any,
         applyArgs: any): any => {
          try {
            onEnter(zone);
            return delegate.invokeTask(target, task, applyThis, applyArgs);
          } finally {
            if ((zone.shouldCoalesceEventChangeDetection && task.type === 'eventTask') ||
                zone.shouldCoalesceRunChangeDetection) {
              delayChangeDetectionForEventsDelegate();
            }
            onLeave(zone);
          }
        },
```

```
onInvoke:
        (delegate: ZoneDelegate, current: Zone, target: Zone, callback: Function, applyThis: any,
         applyArgs?: any[], source?: string): any => {
          try {
            onEnter(zone);
            return delegate.invoke(target, callback, applyThis, applyArgs, source);
          } finally {
            if (zone.shouldCoalesceRunChangeDetection) {
              delayChangeDetectionForEventsDelegate();
            }
            onLeave(zone);
          }
        },
```

```
onHasTask:
        (delegate: ZoneDelegate, current: Zone, target: Zone, hasTaskState: HasTaskState) => {
          delegate.hasTask(target, hasTaskState);
          if (current === target) {
            // We are only interested in hasTask events which originate from our zone
            // (A child hasTask event is not interesting to us)
            if (hasTaskState.change == 'microTask') {
              zone._hasPendingMicrotasks = hasTaskState.microTask;
              updateMicroTaskStatus(zone);
              checkStable(zone);
            } else if (hasTaskState.change == 'macroTask') {
              zone.hasPendingMacrotasks = hasTaskState.macroTask;
            }
          }
        },
```


function onLeave(zone: NgZonePrivate) {
zone._nesting--;
checkStable(zone);
}

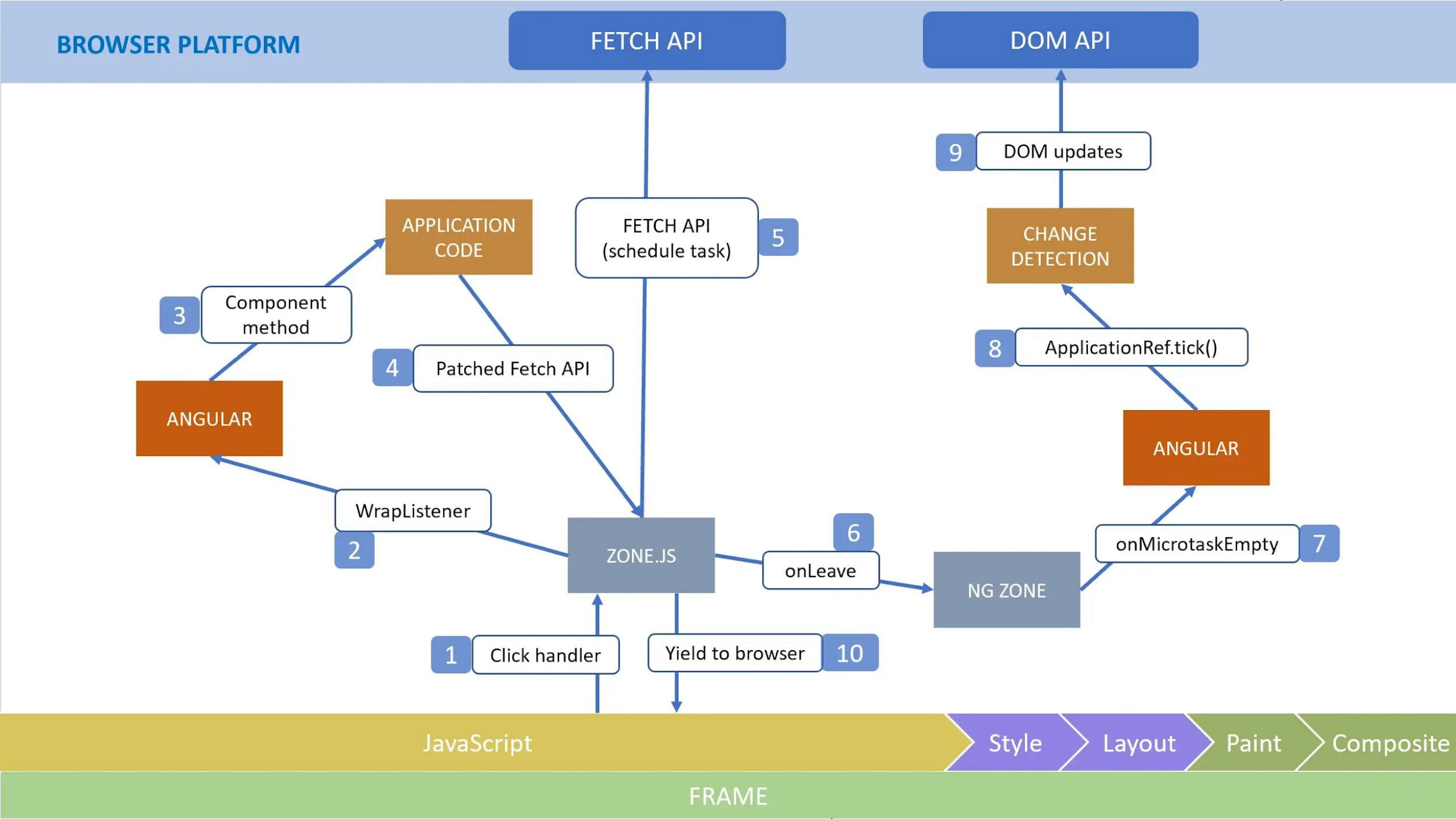
```
function checkStable(zone: NgZonePrivate) {
  if (zone._nesting == 0 && !zone.hasPendingMicrotasks && !zone.isStable) {
    try {
      zone._nesting++;
      zone.onMicrotaskEmpty.emit(null);
    } finally {
      zone._nesting--;
      if (!zone.hasPendingMicrotasks) {
        try {
          zone.runOutsideAngular(() => zone.onStable.emit(null));
        } finally {
          zone.isStable = true;
        }
      }
    }
  }
}
```

```
@Injectable({providedIn: 'root'})
export class ApplicationRef {


  /** @internal */
  constructor(
      private _zone: NgZone,
      private _injector: EnvironmentInjector,
      private _exceptionHandler: ErrorHandler,
  ) {
    this._onMicrotaskEmptySubscription = this._zone.onMicrotaskEmpty.subscribe({
      next: () => {
        this._zone.run(() => {
          this.tick();
        });
      }
    });
```

```
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();
        }
      }
    } catch (e) {
      // Attention: Don't rethrow as it could cancel subscriptions to Observables!
      this._zone.runOutsideAngular(() => this._exceptionHandler.handleError(e));
    } finally {
      this._runningTick = false;
    }
  }
```


BROWSER PLATFORM
FETCH API
DOM API
9
DOM updates
CHANGE DETECTION
APPLICATION CODE
FETCH API (schedule task)
5
3
Component method
4
Patched Fetch API
8
ApplicationRef.tick()
ANGULAR
ANGULAR
WrapListener
2
ZONE.JS
6
onLeave
NG ZONE
onMicrotaskEmpty
7
1
Click handler
Yield to browser
10
JavaScript
Style
Layout
Paint
Composite
FRAME

CAN I USING ANGULAR WITHOUT ZONEJS?

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


export class DashboardComponent implements OnInit {
value = 1;
constructor(
private applicationRef: ApplicationRef
) { }
ngOnInit(): void {
}
update(): void {
this.value = 2;
this.applicationRef.tick();
}
}

```
export class DashboardComponent implements OnInit {
    value = 1;
    constructor(
        private cdr: ChangeDetectorRef
    ) { }

    ngOnInit(): void {
    }

    update(): void {
        this.value = 2;
        this.cdr.detectChanges();
    }

}
```

THANK YOU FOR LISTENING

Tu Hoang