A ***change detection***, ***zone.js*** and ***signals*** story 📖

A *change detection*, *zone.js* and *signals* story 📖

Enea Jahollari / @Enea_Jahollari
GDE for Angular / Software Engineer
Push-based.io

PUSH BASED

Experts 2024

app.component.ts

```ts
@Component({
  template: `
    <div>{{ count }}</div>
    <button (click)="onClick()">Increase</button>
  `,
})
export class AppComponent {
  count = 0;
  onClick() {
    this.count++;
  }
}
```

Editor Preview Both

app.component.ts

```ts
@Component({
  template: `
    <div>{{ count }}</div>
    <button (click)="onClick()">Increase</button>
  `,
})
export class AppComponent {
  count = 0;
  onClick() {
    this.count++;
  }
}
```

Editor Preview Both

How does Angular know when something changes?

How does Angular know when something changes?

🤔 Maybe it runs change detection on every event that happens ???

app.component.ts

```ts
@Component({
  template: `
    <div>{{ count }}</div>
    <button (click)="onClick()">Increase</button>
  `,
})
export class AppComponent {
  count = 0;
  onClick() {
    this.count++;
  }
}
```

Editor Preview Both

Imaginary change detection with **synchronous** code

```
export class AppComponent {}

button.addEventListener('click', () => {
  AppComponent.onClick();
  angular.runChangeDetection();
});
```

⚠ imaginary code

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    this.count++;
  }
}
```

Editor Preview Both

app.component.ts

```
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    setTimeout(() => {
       this.count++;
    }, 1000);
  }
}
```

Editor Preview Both

app.component.ts

```
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log(‘before’, this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
    }, 1000);
    console.log('after', this.count);
  }
}
```

Editor Preview Both

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
    }, 1000);
    console.log('after', this.count);
  }
}
```

Editor Preview Both

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
    }, 1000);
    console.log('after', this.count);
  }
}
```

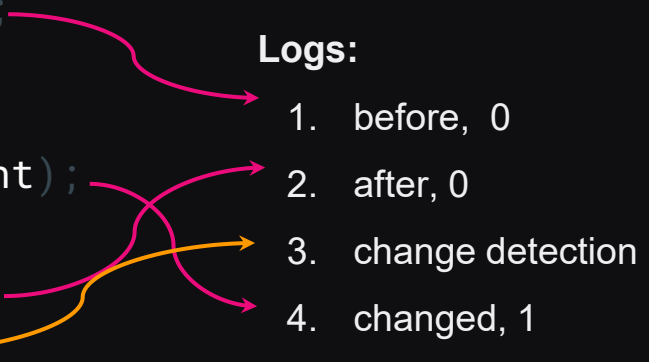
Logs:

1. before, 0
2. after, 0
3. changed, 1 (after 1s)

Editor Preview Both

app.component.ts

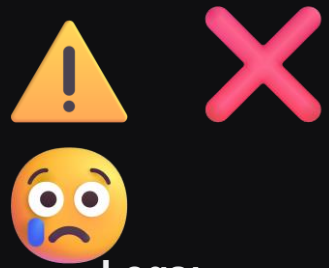
```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
    }, 1000);
    console.log('after', this.count);
    angular.runChangeDetection();
  }
}
```



Editor Preview Both

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
    }, 1000);
    console.log('after', this.count);
    angular.runChangeDetection();
  }
}
```



Logs:

1. before, 0
2. after, 0 (count is still 0)
3. change detection
4. changed, 1

Editor Preview Both

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
      angular.runChangeDetection();
    }, 1000);
    console.log('after', this.count);
  }
}
```

Logs:

1. before, 0
2. after, 0
3. changed, 1
4. change detection

Editor Preview Both

app.component.ts

```
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
      $scope.$digest(); 💀
    }, 1000);
    console.log('after', this.count);
  }
}
```

Logs:

1. before, 0
2. after, 0
3. changed, 1
4. change detection

Editor Preview Both

app.component.ts

```ts
@Component()
export class AppComponent {
  count = 0;

  onClick() {
    console.log('before', this.count);
    setTimeout(() => {
      this.count++;
      console.log('changed', this.count);
      angular.runChangeDetection();
    }, 1000);
    console.log('after', this.count);
  }
}
```

Logs:

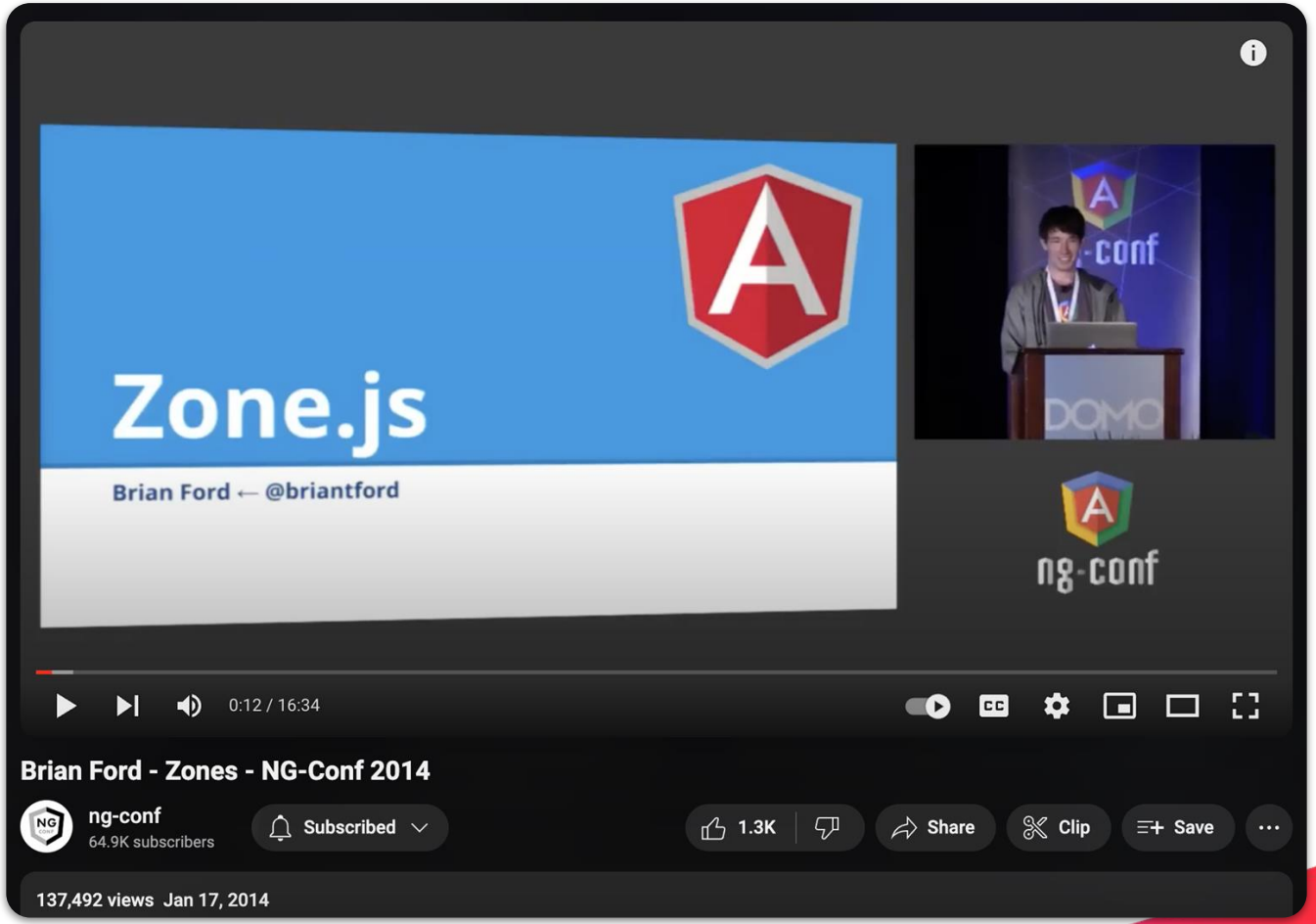
1. before, 0
2. after, 0
3. changed, 1
4. change detection

Editor Preview Both

How to run code after some asynchronous events?

How to run code after some asynchronous events?

Angular solved this issue in **2014** with **zone.js**

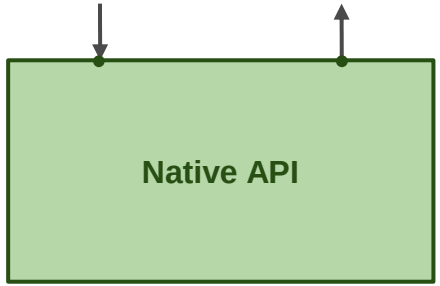


https://www.youtube.com/watch?v=3IqtmUscE_U

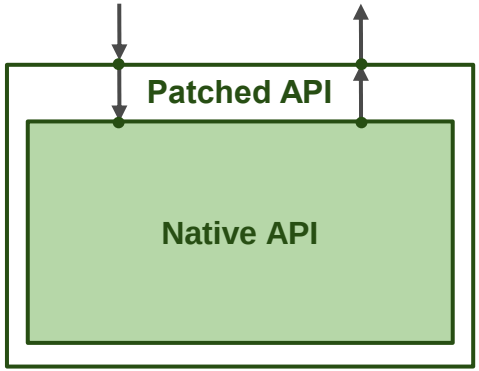
zone.js / NgZone

Zone.js monkey patches almost all the browser API-s in order to get notified whenever an event happens!

Browser



Browser + zone.js



zone.js / NgZone

Zone.js monkey patches almost all the browser API-s in order to get notified whenever an event happens!

- target.**addEventListener**
- target.**onProp** (target.*onclick* and so on)
- **setTimeout/setInterval/setImmediate**
- **XMLHttpRequest**
- **Promise**.then, PromiseRejectEvent
- **requestAnimationFrame**
- **alert/prompt/confirm**
- Full List 🔗

zone.js / NgZone

Zone.js monkey patches alm
get notified whenever an ev

- target.**addEventListener**
- target.**onProp** (target.*onclick* an
- **setTimeout/setInterval/setImme**
- **XMLHttpRequest**
- **Promise**.then, PromiseRejectEve
- **requestAnimationFrame**
- **alert/prompt/confirm**
- Full List

Learn zone.js by practice!

main.ts

```ts
setTimeout(() => {
  console.log('Hello world');
}, 1000);
```

Editor Preview Both

main.ts

```
1  import "zone.js";
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

Editor Preview Both

main.ts

```ts
import "zone.js";

const zone = Zone.current.fork({
  onInvokeTask: (delegate, current, target, task, applyThis, applyArgs) => {
    console.log('Before task');
    delegate.invokeTask(target, task, applyThis, applyArgs);
    console.log('After task');
  }
});
```

Editor Preview Both

main.ts

```
1  import "zone.js";
2
3  const zone = Zone.current.fork({
4    onInvokeTask: (delegate, current, target, task, applyThis, applyArgs) => {
5      console.log('Before task');
6      delegate.invokeTask(target, task, applyThis, applyArgs);
7      console.log('After task');
8    }
9  });
10
11
12
13
14
15
```

If an event is registered inside this zone,
every time it emits, will trigger this function.

Editor Preview Both

main.ts

```ts
import "zone.js";

const zone = Zone.current.fork({
  onInvokeTask: (delegate, current, target, task, applyThis, applyArgs) => {
    console.log('Before task');
    delegate.invokeTask(target, task, applyThis, applyArgs);
    console.log('After task');
  }
});

zone.run(() => {
  setTimeout(() => {
    console.log('Hello world');
  }, 1000);
});
```

Editor Preview Both

main.ts

```ts
import "zone.js";

const zone = Zone.current.fork({
  onInvokeTask: (delegate, current, target, task, applyThis, applyArgs) => {
    console.log('Before task');
    delegate.invokeTask(target, task, applyThis, applyArgs);
    console.log('After task');
  }
});

zone.run(() => {
  setTimeout(() => {
    console.log('Hello world');
  }, 1000);
});
```

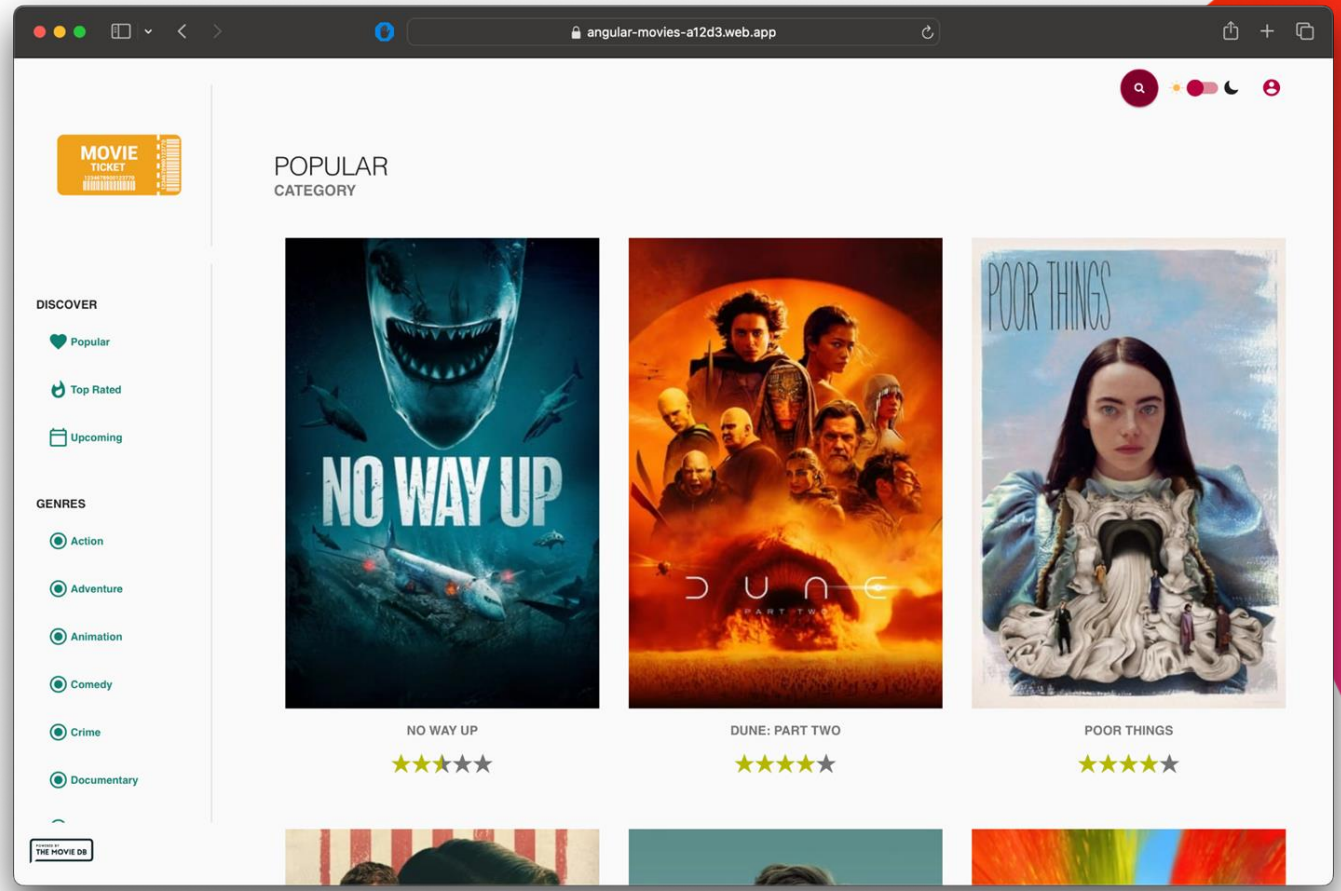
Logs:

1. Before task
2. Hello world
3. After task

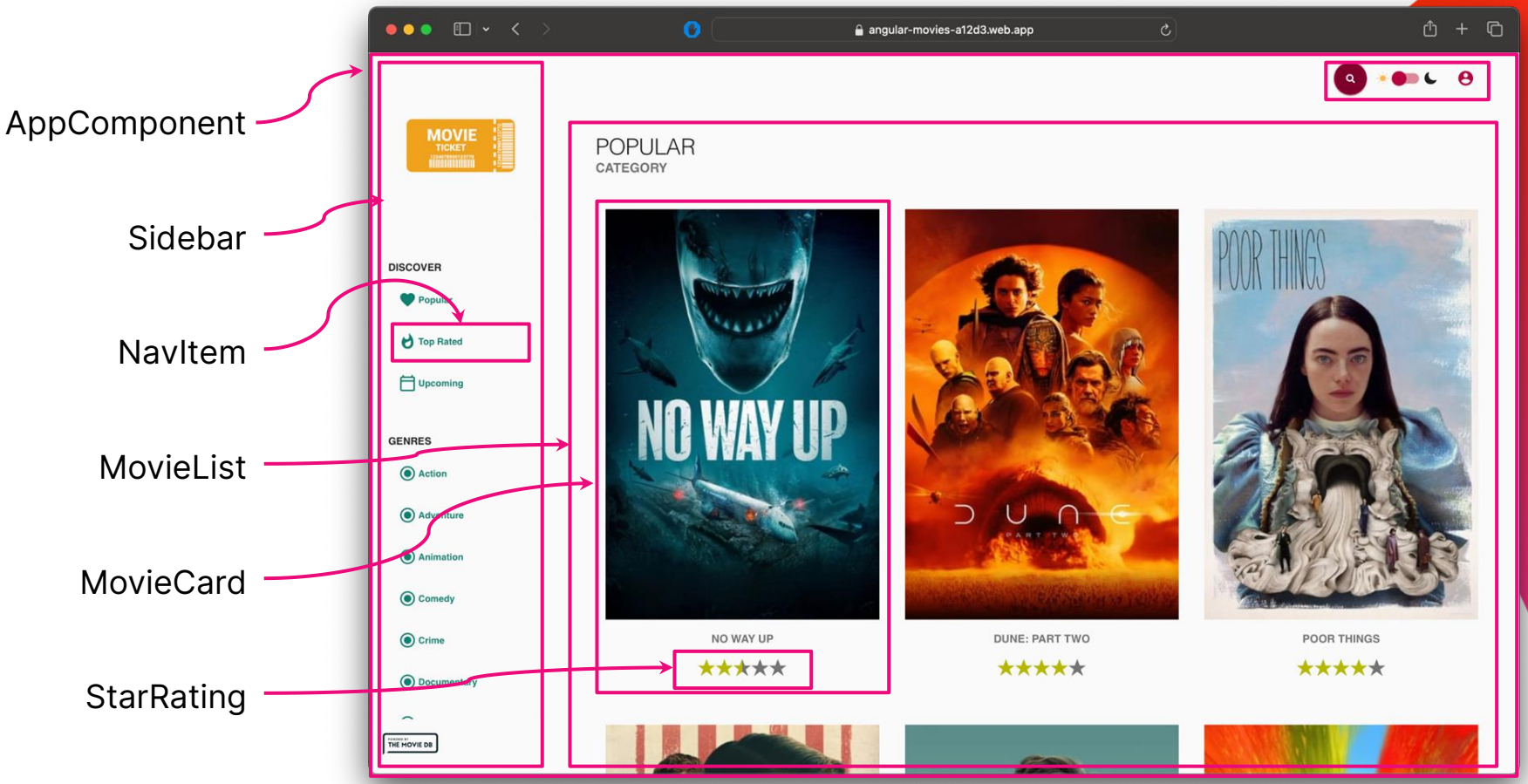
Editor Preview Both

Angular with zone.js

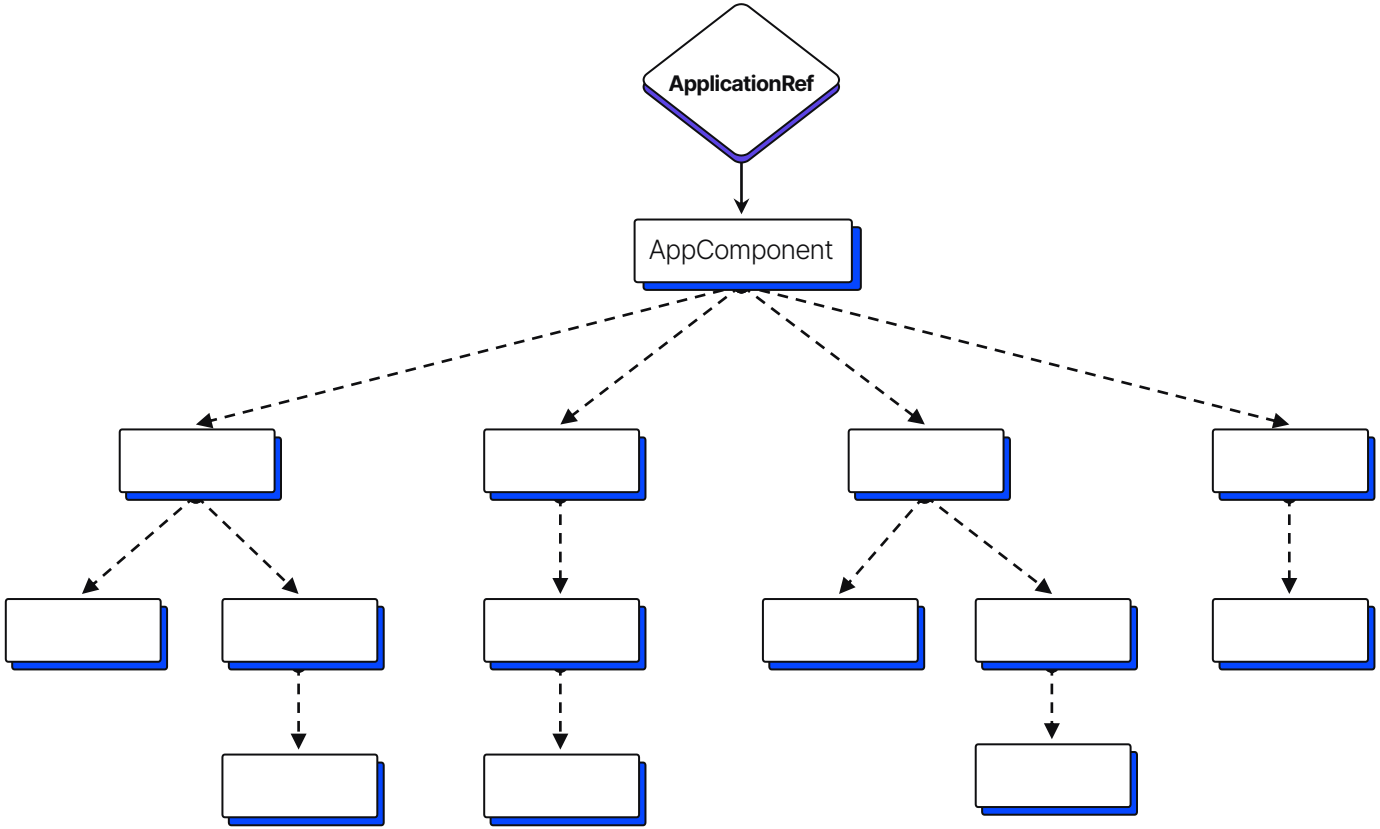
Component structure



Component structure



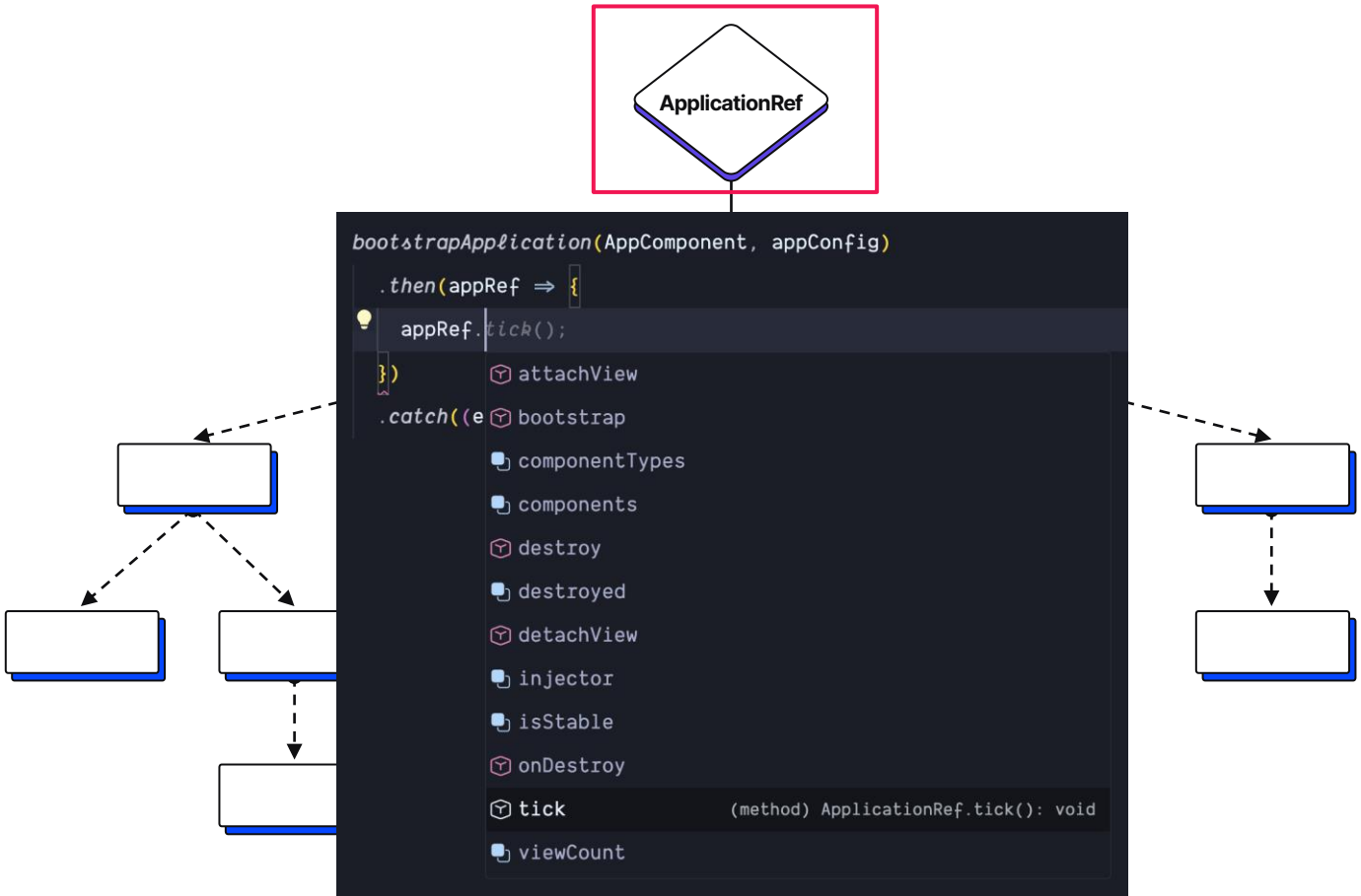
Angular Component Tree



Legend

Component

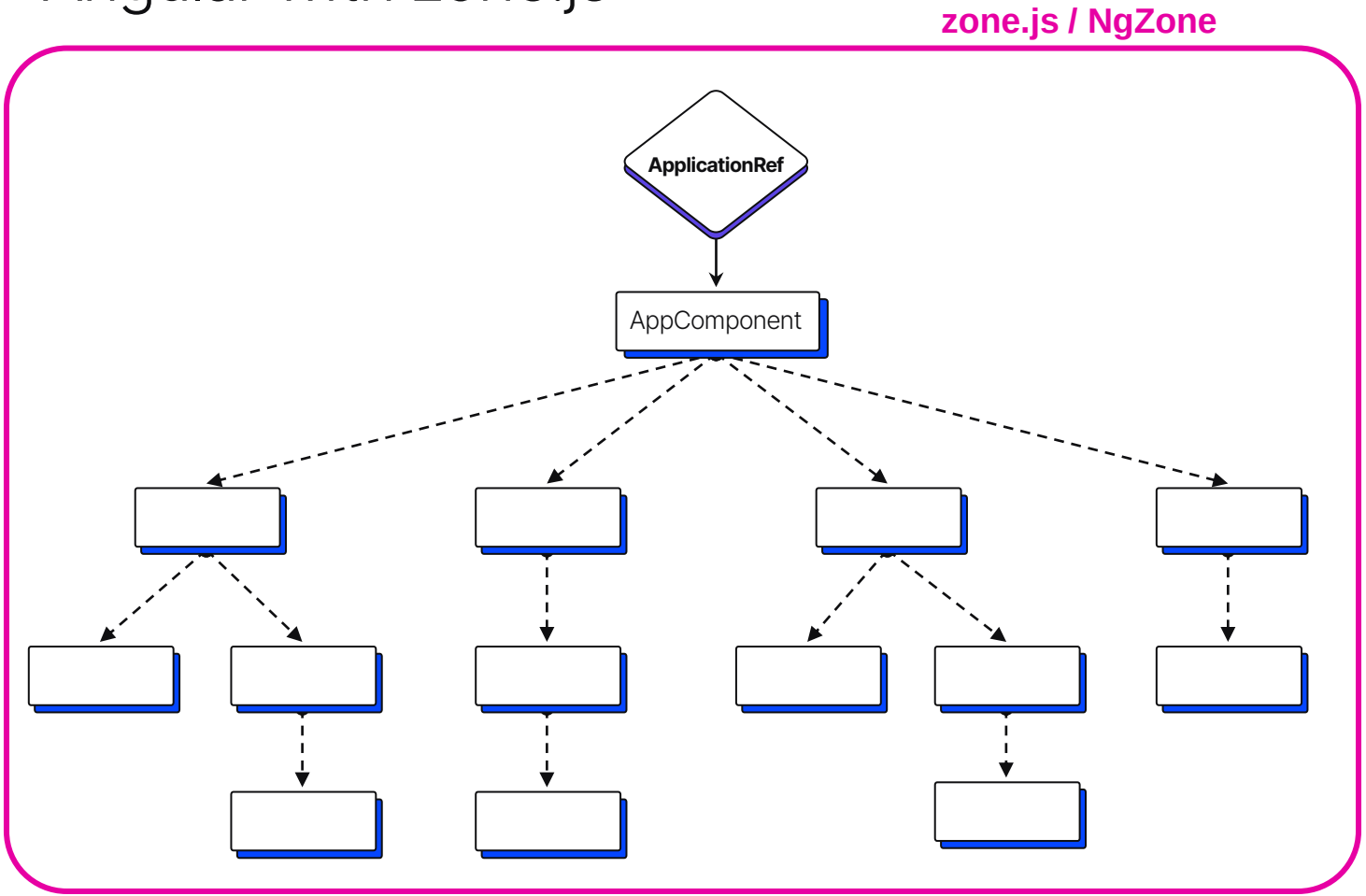
ApplicationRef



Legend

Component

Angular with zone.js



Legend

Component

zone.js

ng_zone.ts

```
export declare class NgZone {
    /** NNotifies when all the asynchronous operations are finished. */
    readonly onMicrotaskEmpty: EventEmitter<any>;
    ...
    constructor() {
        ...
        forkInnerZoneWithAngularBehavior(self);
    }
}

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

Editor Preview Both

ng_zone.ts

```
export declare class NgZone {
    /** Notifies when all the asynchronous operations are finished. */
    readonly onMicrotaskEmpty: EventEmitter<any>;
    ...
    constructor() {
        ...
        forkInnerZoneWithAngularBehavior(self);
    }
}

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

Editor Preview Both

ng_zone.ts

```
export declare class NgZone {
    /** Notifies when all the asynchronous operations are finished. */
    readonly onMicrotaskEmpty: EventEmitter<any>;
    ...
    constructor() {
        ...
        forkInnerZoneWithAngularBehavior(self);
    }
}

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

Editor Preview Both

ng_zone_scheduling.ts

```ts
@Injectable({ providedIn: 'root' })
export class NgZoneChangeDetectionScheduler {
  private readonly zone = inject(NgZone);
  // ...

  initialize(): void {
    this._onMicrotaskEmptySubscription = this.zone.onMicrotaskEmpty.subscribe({
      next: () => {
        // ...
        this.zone.run(() => {
          this.applicationRef.tick();
        });
      },
    });
  }
}
```

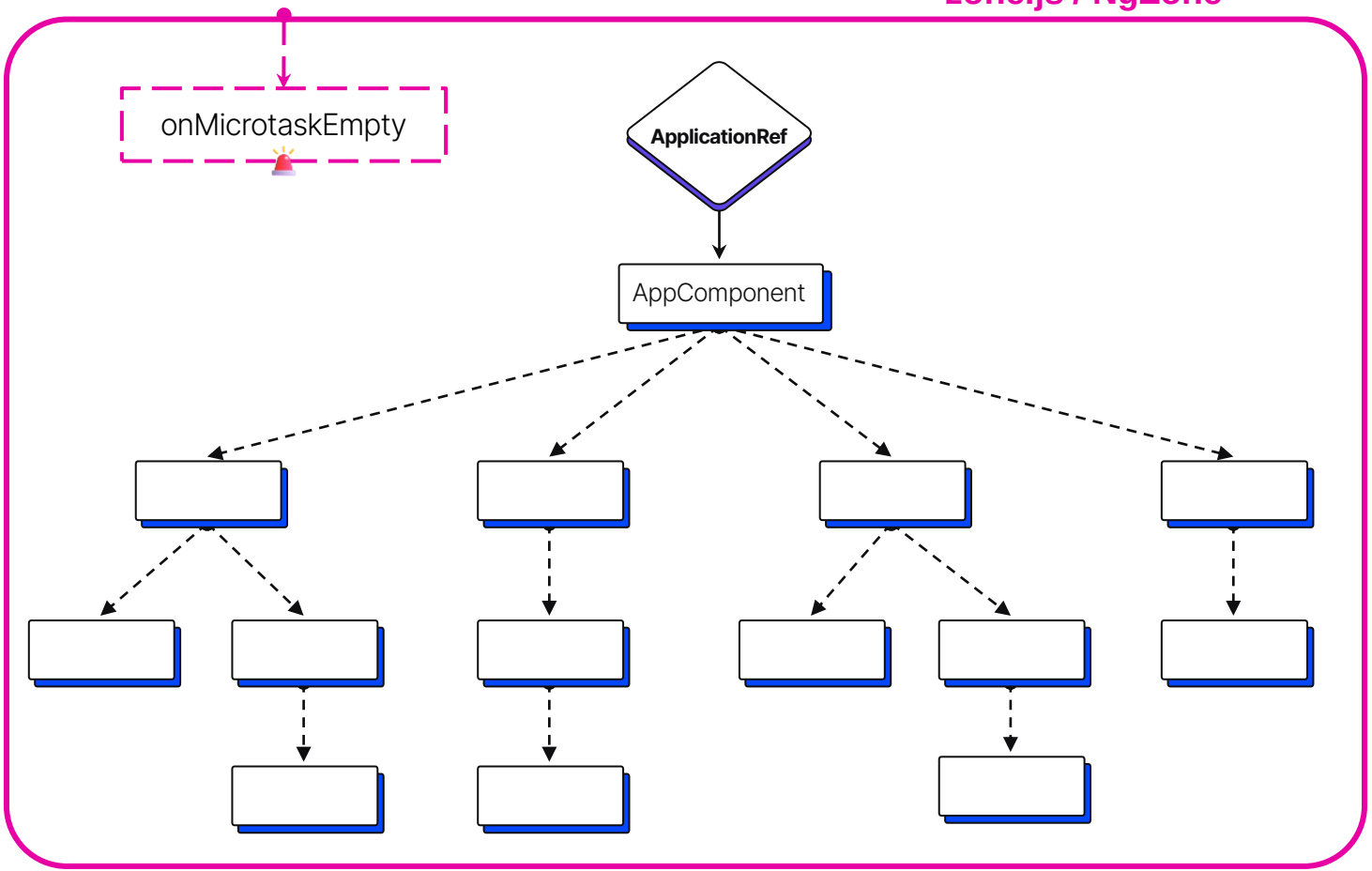
Editor Preview Both

ng_zone_scheduling.ts

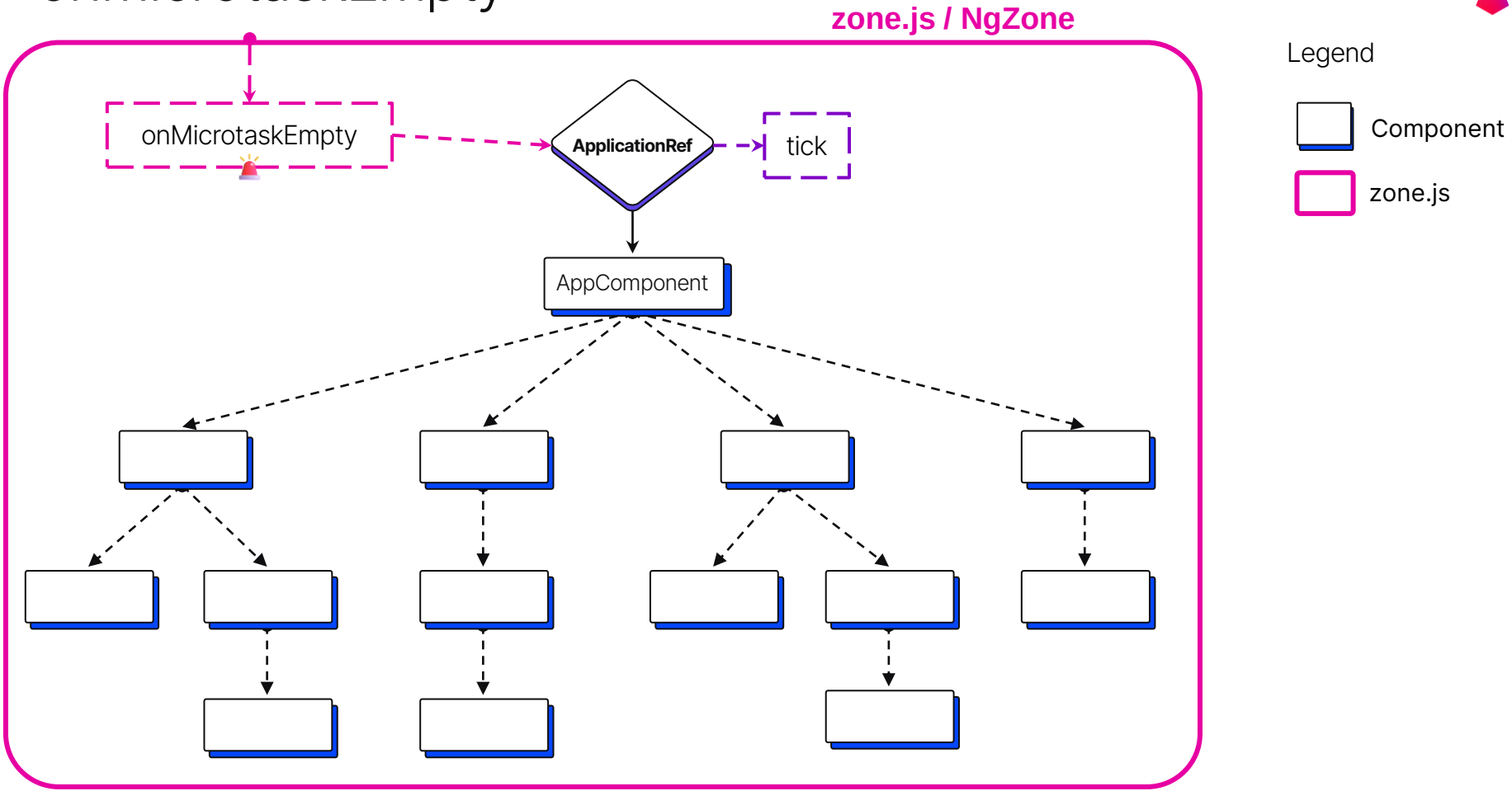
```ts
@Injectable({ providedIn: 'root' })
export class NgZoneChangeDetectionScheduler {
  private readonly zone = inject(NgZone);
  // ...

  initialize(): void {
    this._onMicrotaskEmptySubscription = this.zone.onMicrotaskEmpty.subscribe({
      next: () => {
        // ...
        this.zone.run(() => {
          this.applicationRef.tick();
        });
      },
    });
  }
}
```

Editor Preview Both

onMicrotaskEmpty



onMicrotaskEmpty



What is this **tick()** method?

What is this **tick()** method?

Remember the imaginary ***angular.runChangeDetection()*** ?

application_ref.ts

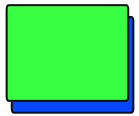
```ts
@Injectable({ providedIn: 'root' })
export class ApplicationRef {

  tick(): void {
    // ...
    this.detectChangesInAttachedViews(...);
  }

  private detectChangesInAttachedViews(...) {
    for (let {lView} of this._views) {
       detectChangesInternal(lView, ...);
    }
  }

}

```

Editor Preview Both

Change detection states

Skipped

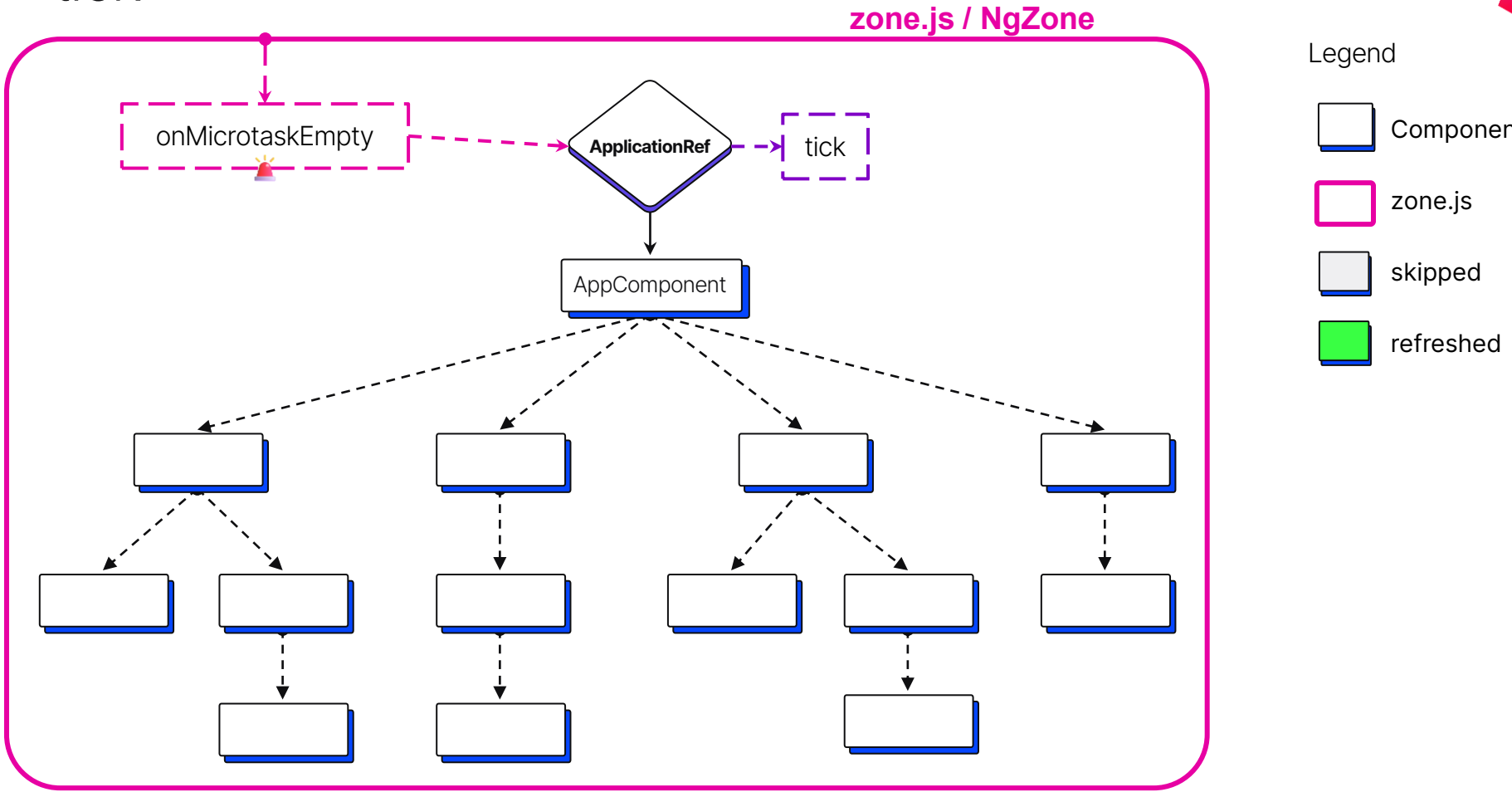
Nothing was done!



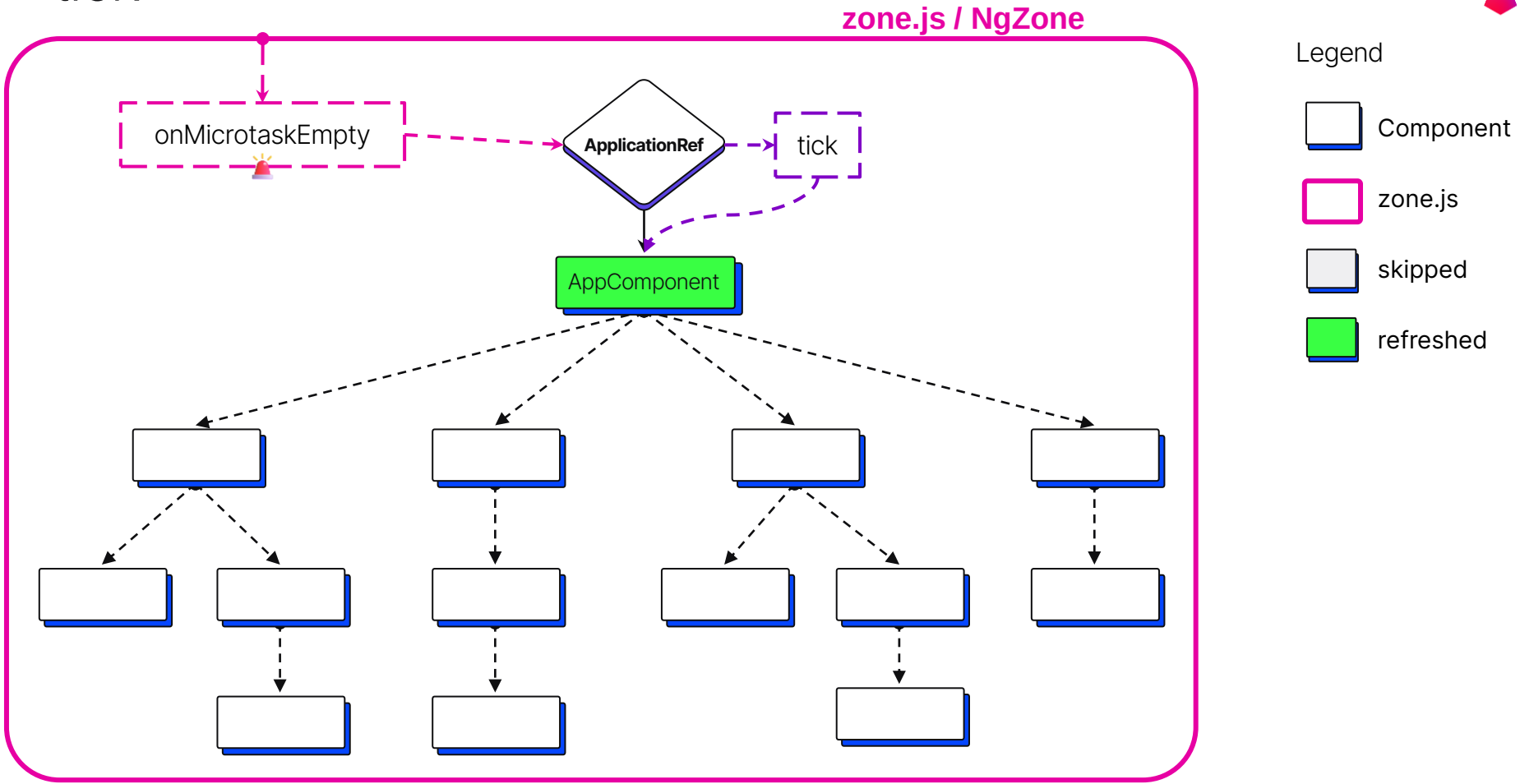
Refreshed Bindings

Update phase & lifecycle hooks were run

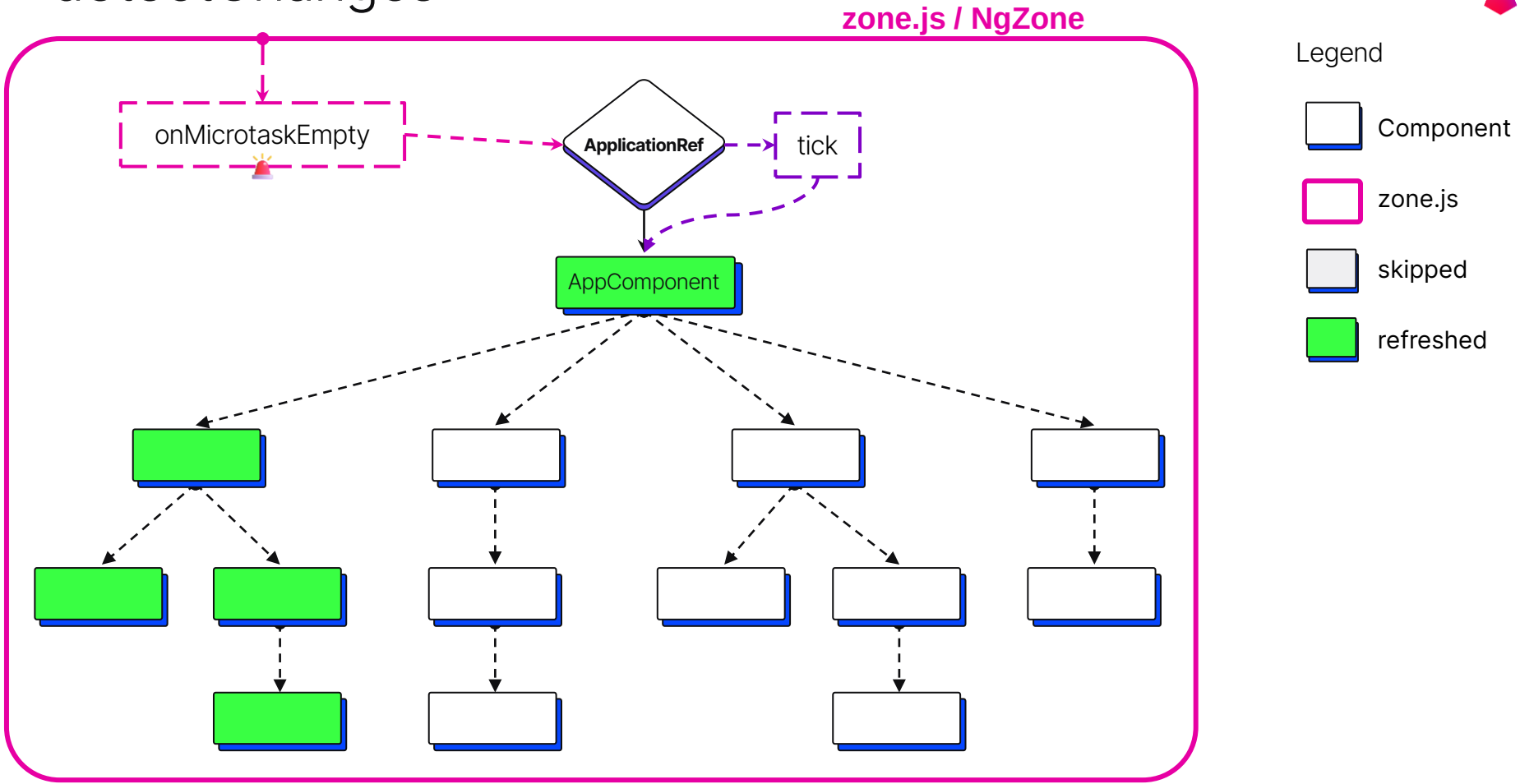
tick


zone.js / NgZone
onMicrotaskEmpty
ApplicationRef
tick
AppComponent
Legend
Component
zone.js
skipped
refreshed

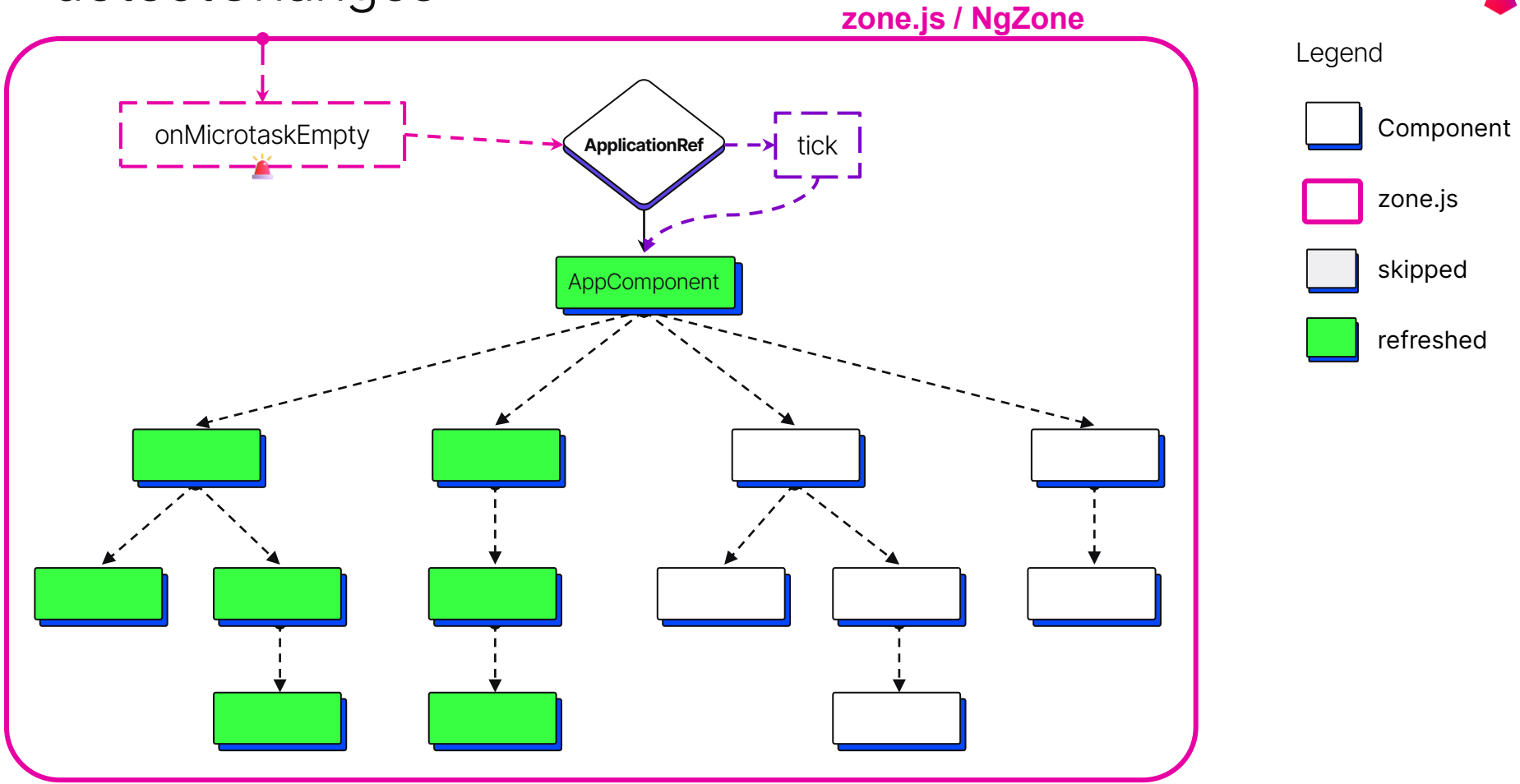
tick


zone.js / NgZone
onMicrotaskEmpty
ApplicationRef
tick
AppComponent
Legend
Component
zone.js
skipped
refreshed

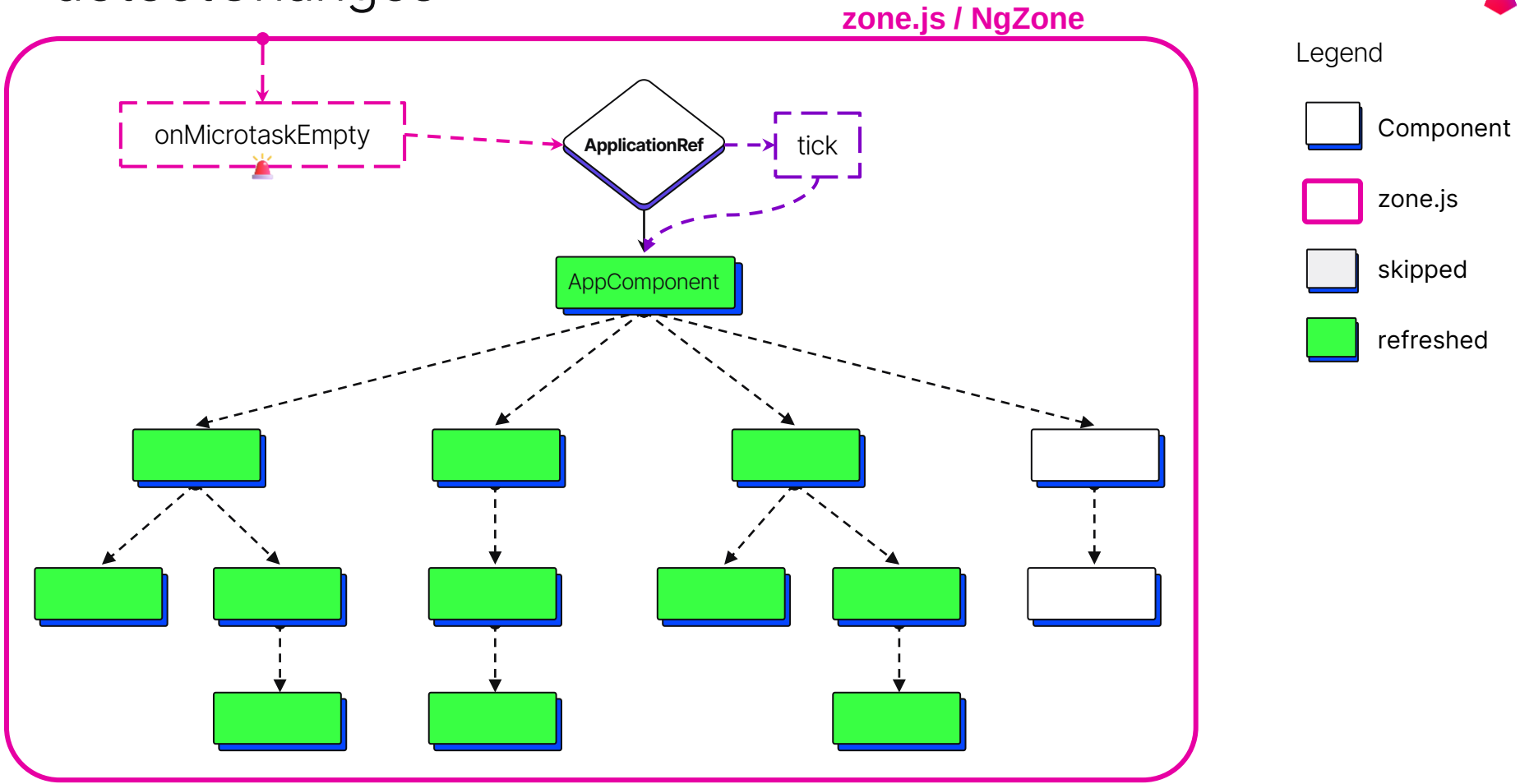
detectChanges



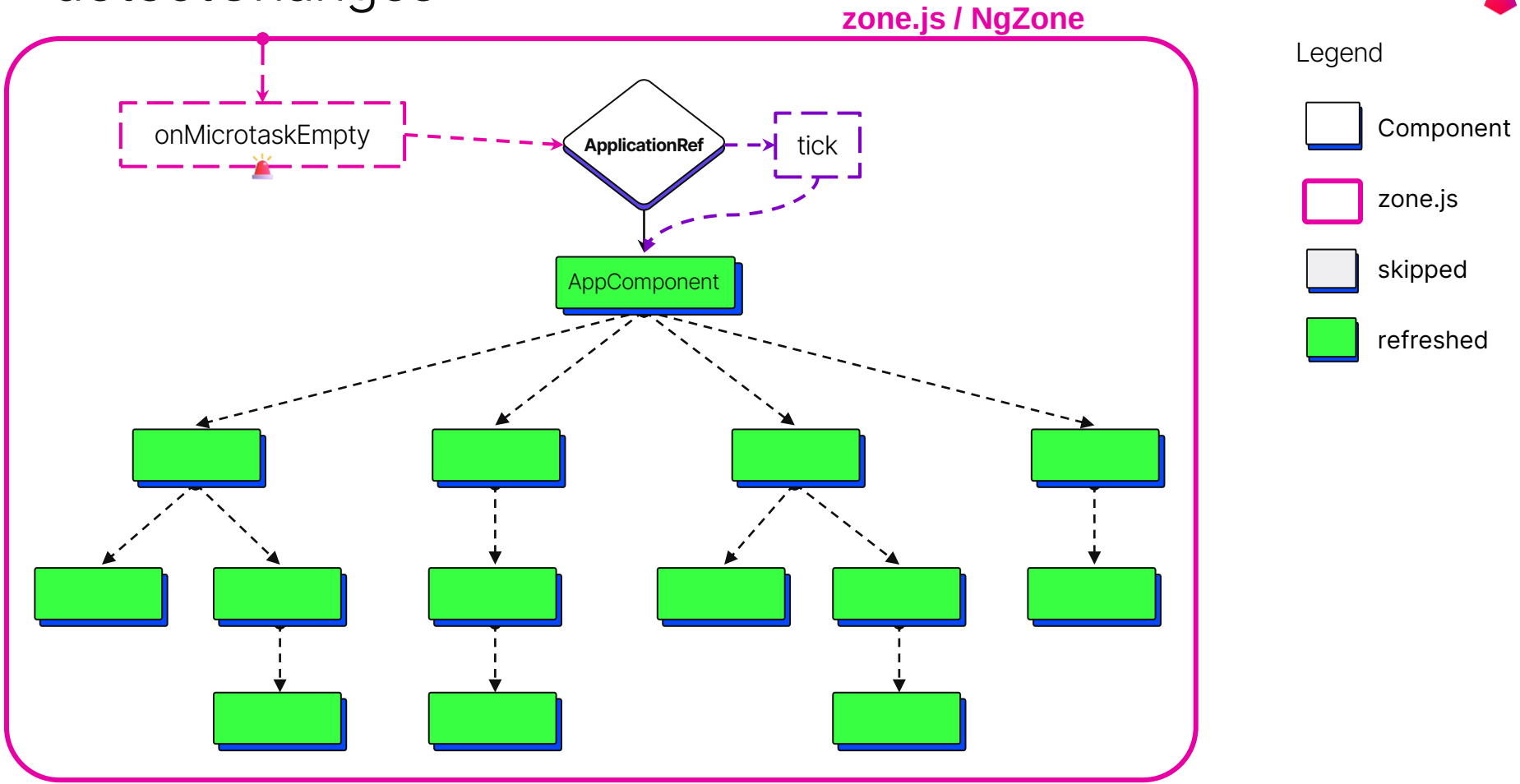
detectChanges



detectChanges



detectChanges



Angular refreshes bindings top-down ↓

Angular refreshes bindings top-down ↓

NOTE: Refresh bindings means:

-> is the previous value different from new value?

-> yes, update DOM

-> no, skip DOM update (no work)

How to skip checking everything everytime?

How to skip checking everything everytime?

Dirty Marking + OnPush to the rescue!

Dirty marking

A component is marked as **dirty** by:

- Event listeners (click, mouseover)
- Output emissions
- Changed inputs using ***setInput***
- ***?***

Dirty marking

A component is marked as **dirty** by:

- 👉 Event listeners (click, mouseover)
- 👉 Output emissions
- Changed inputs using ***setInput***
- ***?***

```
<button (click)="onClick()">Hi</button>
<hlm-select (openedChange)="onOpen()"></hlm-select>


// instructions/listener.ts
function wrapListener(): EventListener {
  return function wrapListenerIn_markDirtyAndPreventDefault(e) {
    // ... code removed for brevity
    markViewDirty(startView); // mark the component as dirty
  };
}
```

Dirty marking

A component is marked as **dirty** by:

- Event listeners (click, mouseover)
- Output emissions
- 👉 Changed inputs using ***setInput***
- ***?***

```
setInput(name: string, value: unknown): void {
  // Do not set the input if it is the same as the last value
  if (Object.is(this.previousInputValues.get(name), value)) {
    return;
  }
  // code removed for brevity
  setInputsForProperty(lView[TVIEW], lView, dataValue, name, value);
  markViewDirty(childComponentLView); // mark the component as dirty
}
```

Dirty marking

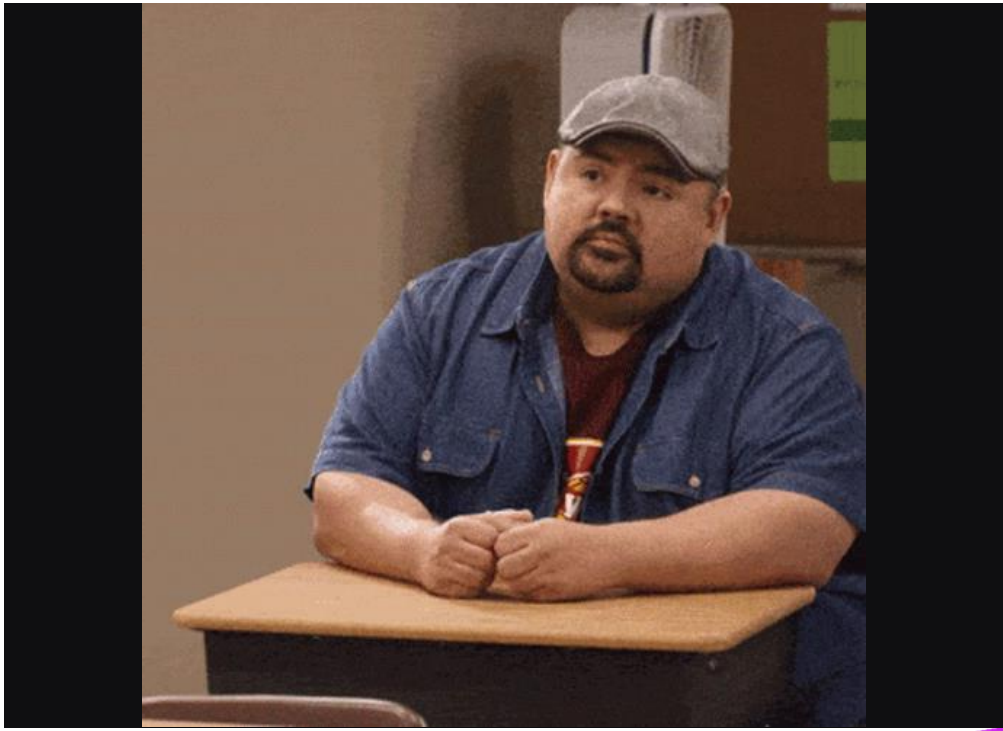
A component is marked as **dirty** by:

- Event listeners (click, mouseover)
- Output emissions
- Changed inputs using ***setInput***
- 👉 ***?***

Dirty marking

A component is marked as **dirty** by:

- Event listeners (click, mouseover)
- Output emissions
- Changed inputs using ***setInput***
- 👉 ***markForCheck()***



Dirty marking

A component is marked as **dirty** by:

- Event listeners (click, mouseover)
- Output emissions
- Changed inputs using ***setInput***
- 👉 ***markForCheck()***

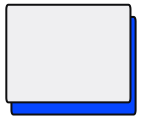
```
export class ChangeDetectorRef {
  markForCheck(): void {
    markViewDirty(...);
  }
}
```

mark_view_dirty.ts

```typescript
/**
 * Marks current view and all ancestors dirty.
 */
export function markViewDirty(lView: LView, source: NotificationSource): LView | null {
  while (lView) {
    lView[FLAGS] |= LViewFlags.Dirty;
    const parent = getLViewParent(lView);
    if (isRootView(lView) && !parent) {
      return lView;
    }
    lView = parent!;
  }
  return null;
}


```

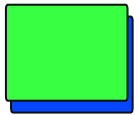
Editor Preview Both

Change detection states



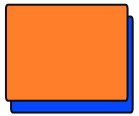
Skipped

Nothing was done!



Refreshed Bindings

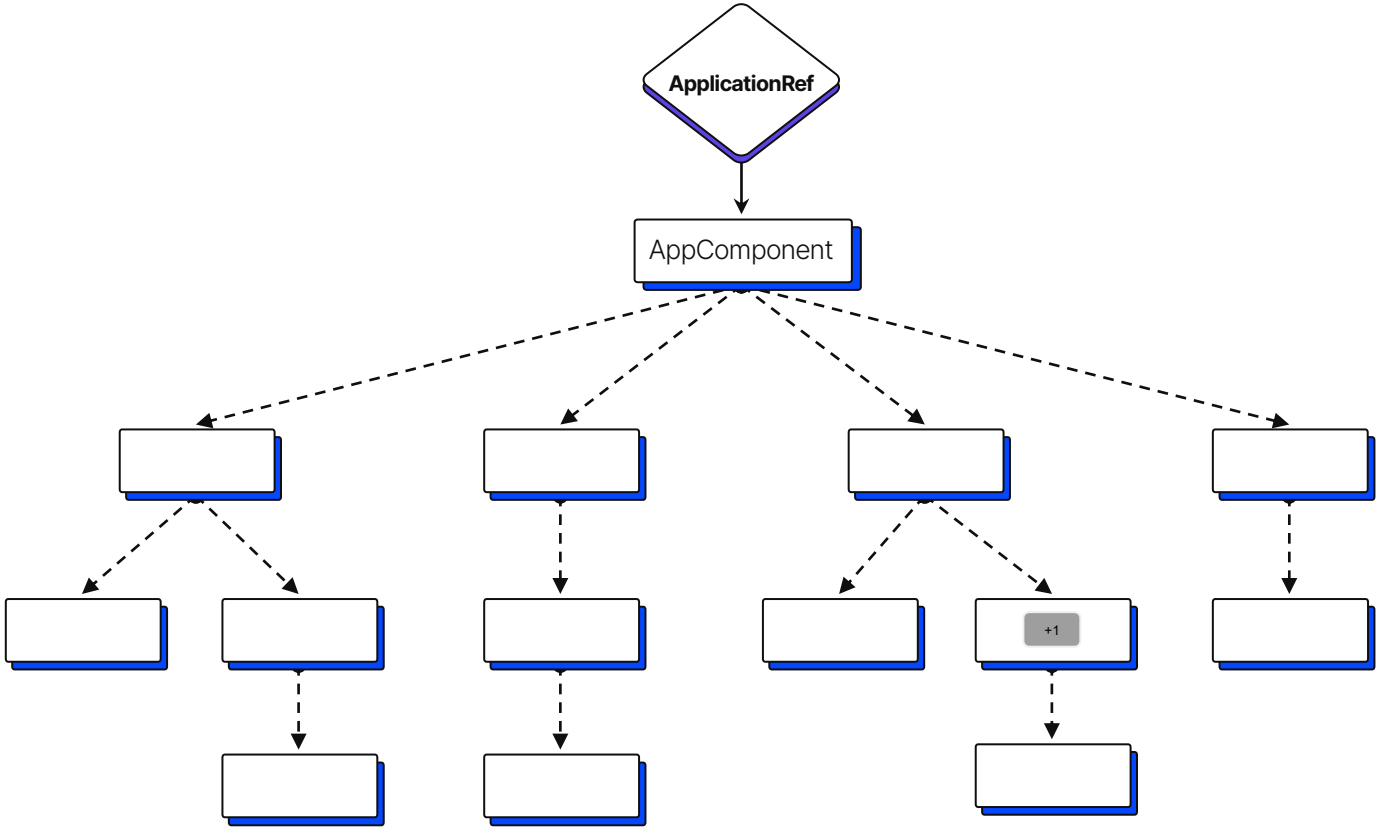
Update phase & lifecycle hooks were run



Marked as Dirty

No check is done yet!

Let’s see how a button click works



Legend

Component

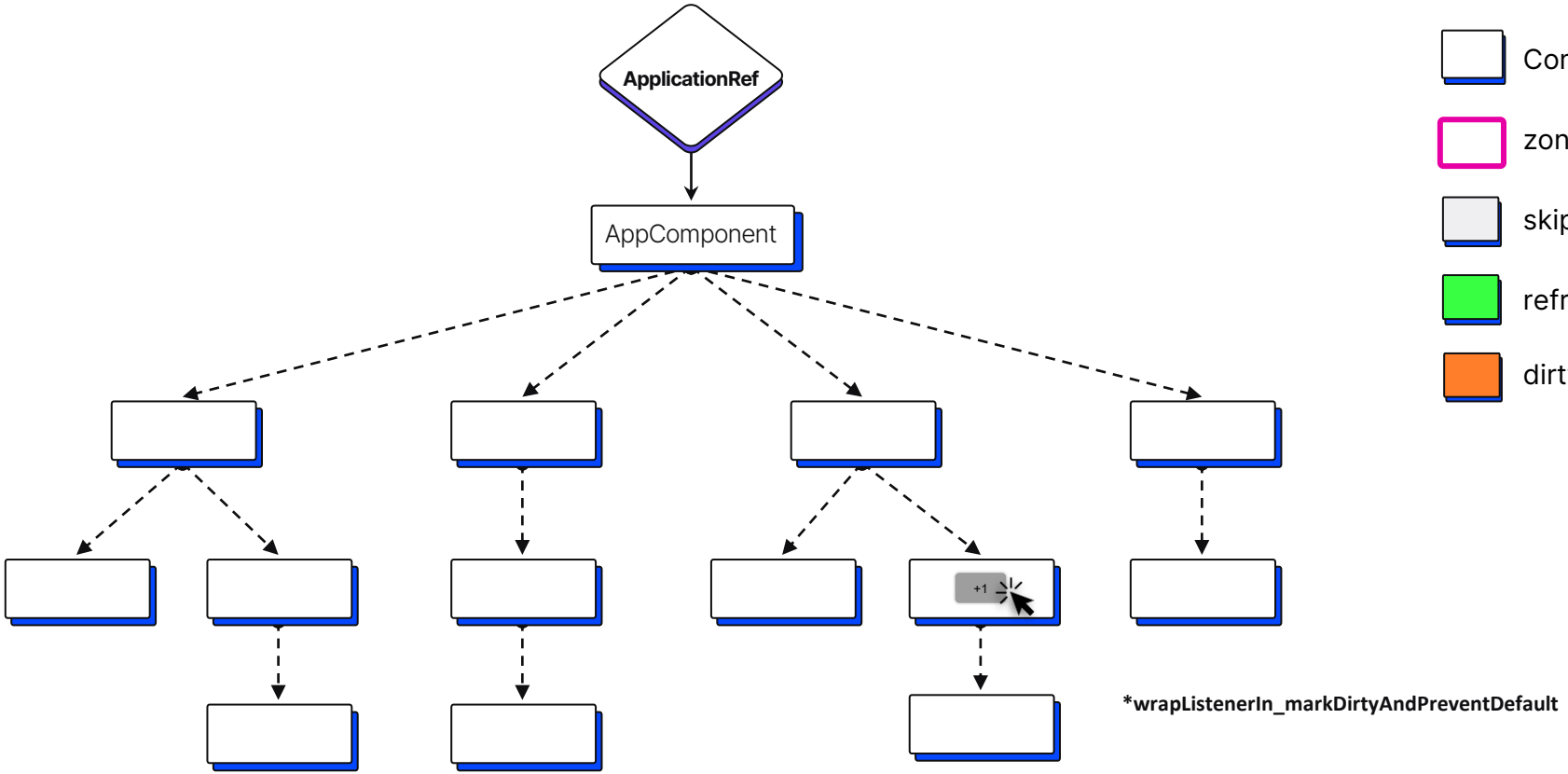
zone.js

skipped

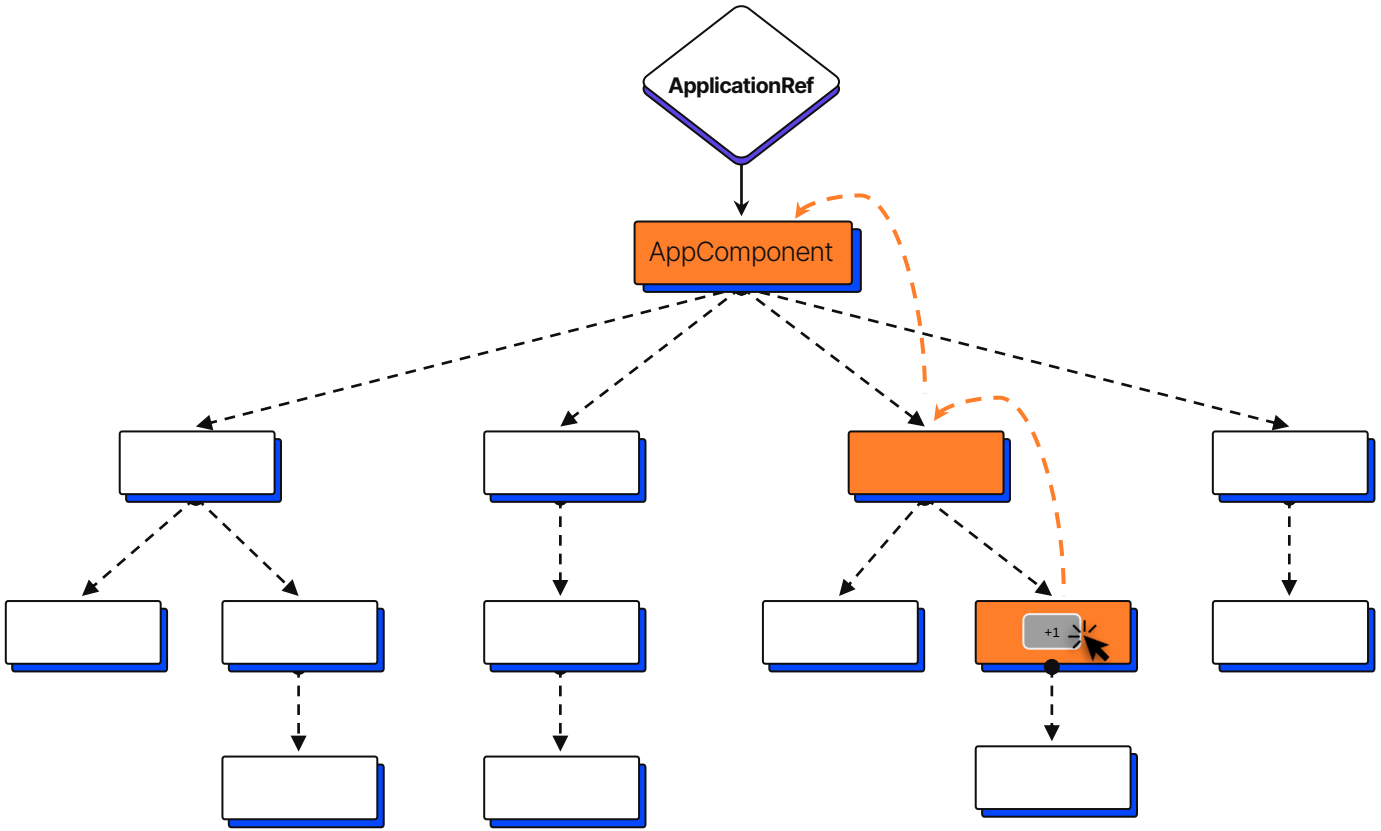
refreshed

dirty

Clicking the button



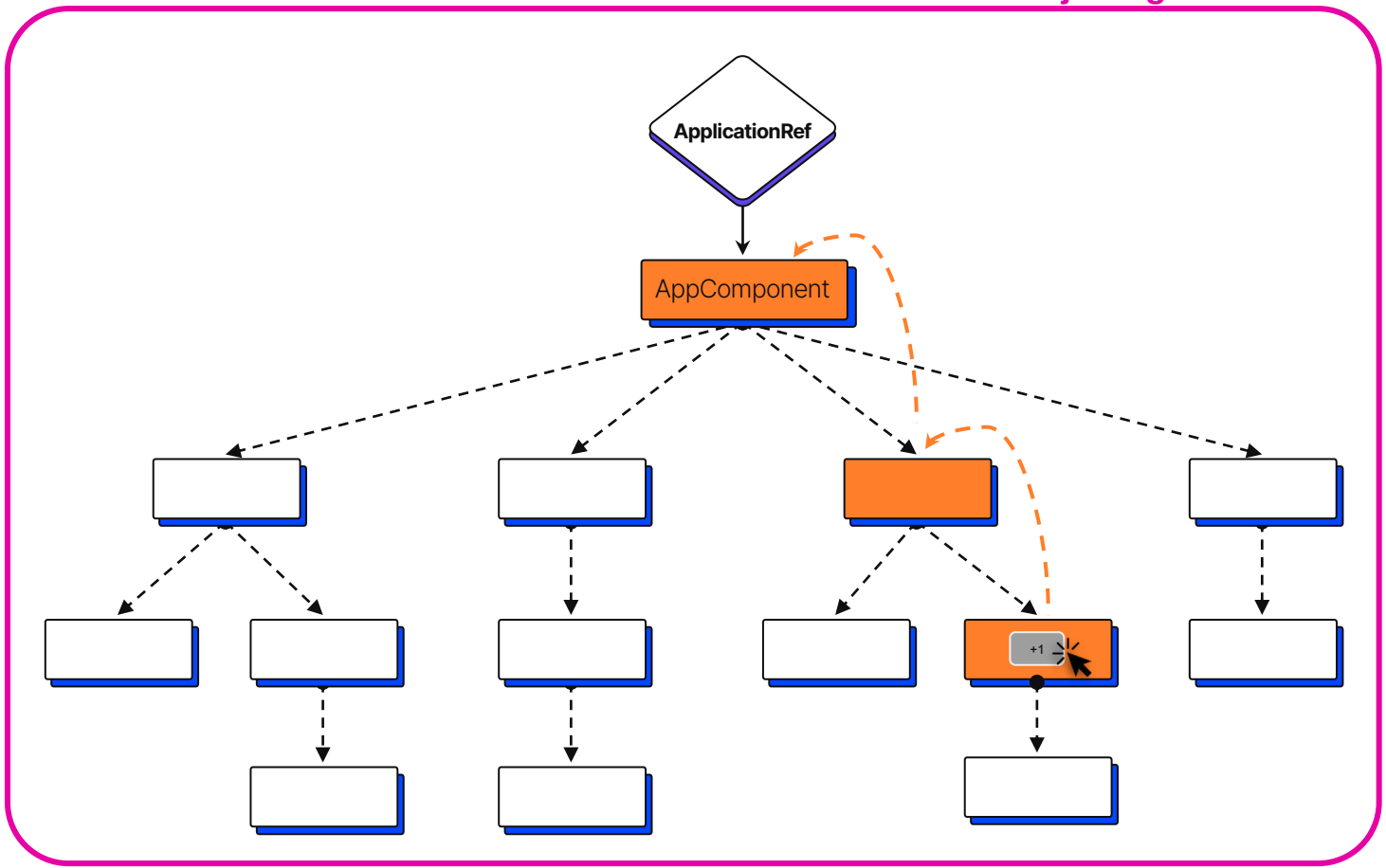
markViewDirty



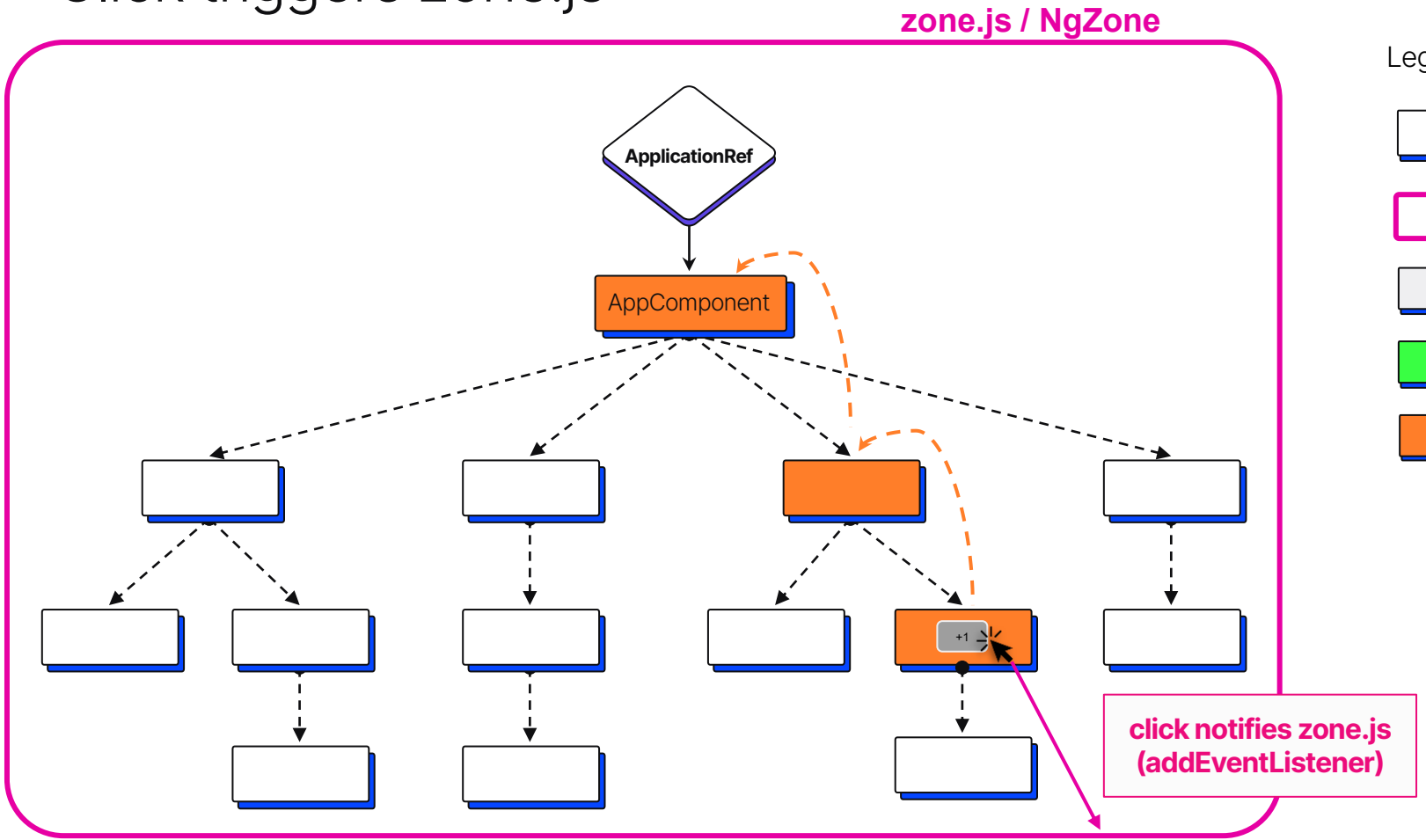
Legend

- Component
- zone.js
- skipped
- refreshed
- dirty

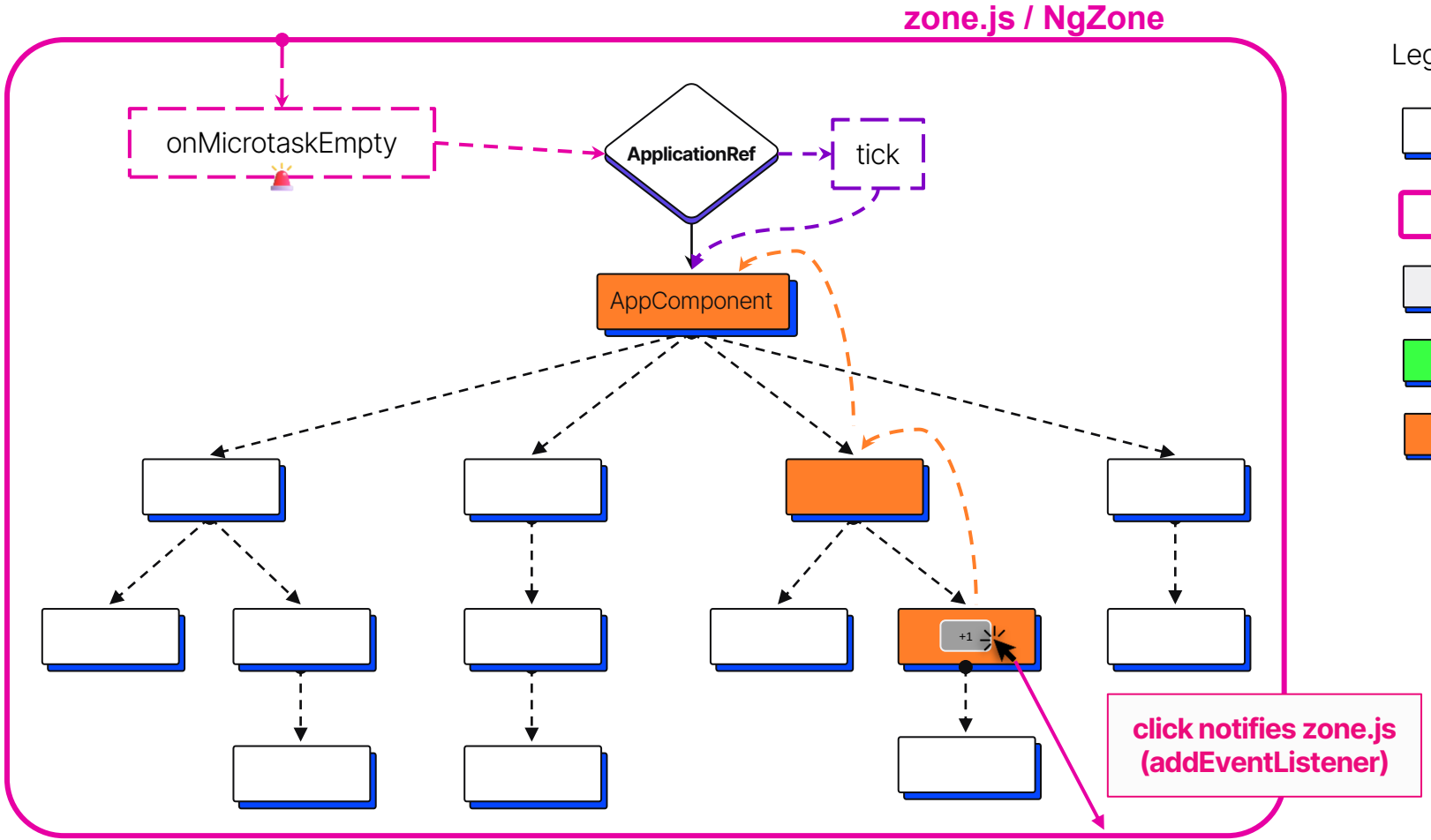
Don’t forget zone.js



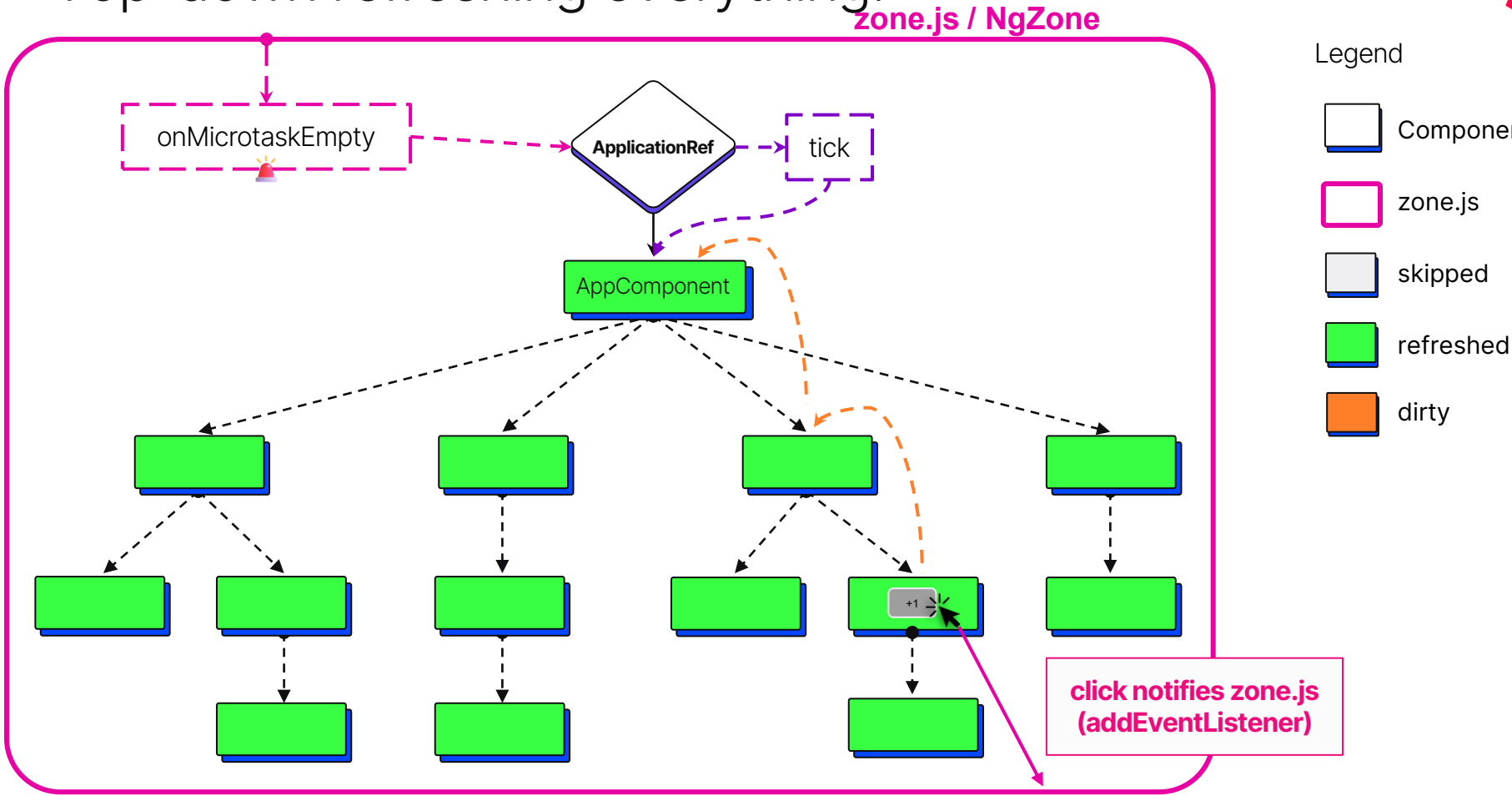
Click triggers zone.js



Tick!


zone.js / NgZone
onMicrotaskEmpty
ApplicationRef
tick
AppComponent
+1
click notifies zone.js
(addEventListener)
Legend
Component
zone.js
skipped
refreshed
dirty

Top-down refreshing everything!



OnPush, OnPush, OnPush, OnPush

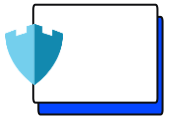
Let’s optimize it even further!

Enable OnPush on component

A component is marked as **OnPush** by setting it in the Component decorator.

```
@Component({
  selector: 'movie-card'
  template: `...`,
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class MovieCard {}
```

OnPush Change detection states



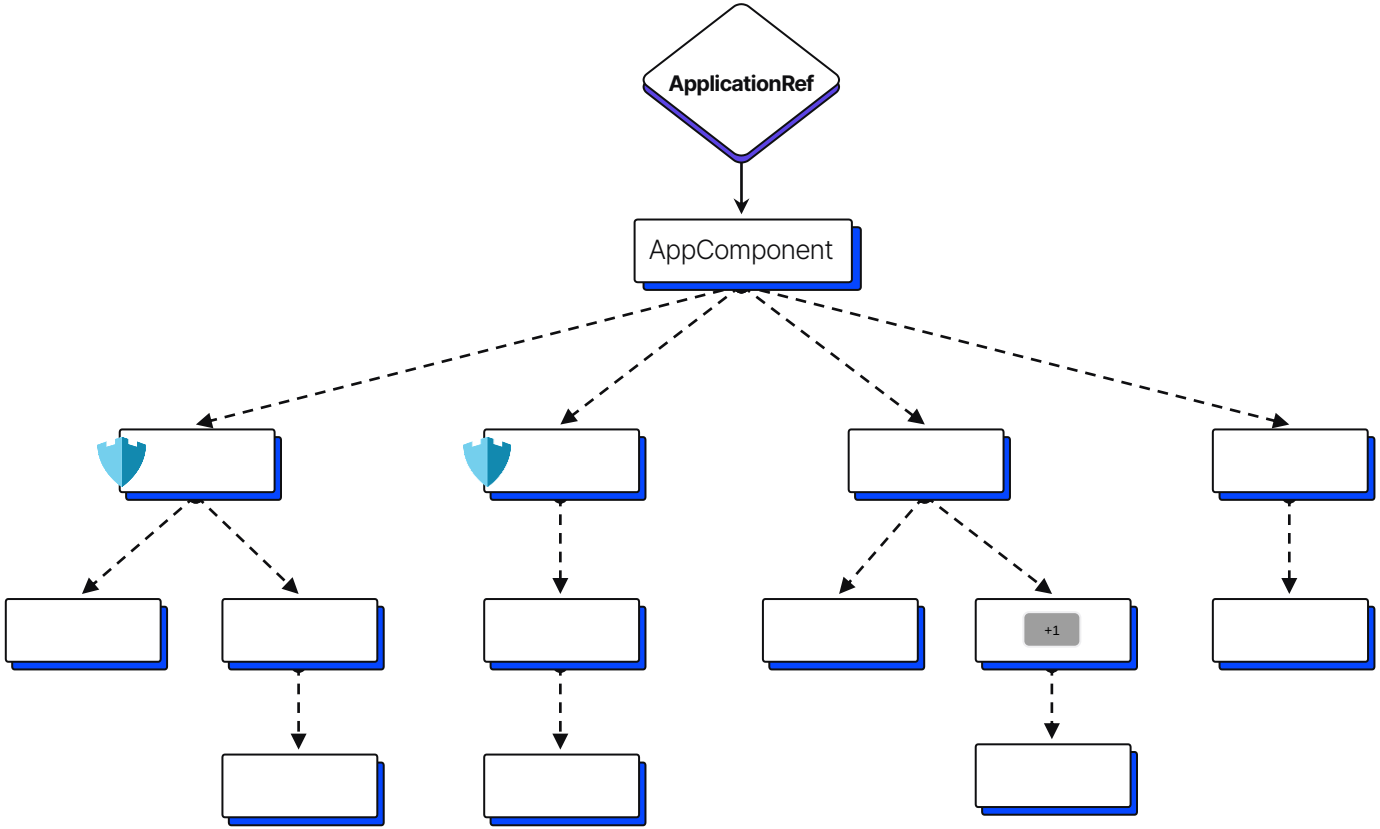
OnPush

Will be **refreshed** if the **inputs** were changed/updated.
(uses '===' for checks)

OnPush + Dirty

Marked as Dirty.
Will be **refreshed** on the next **CD**

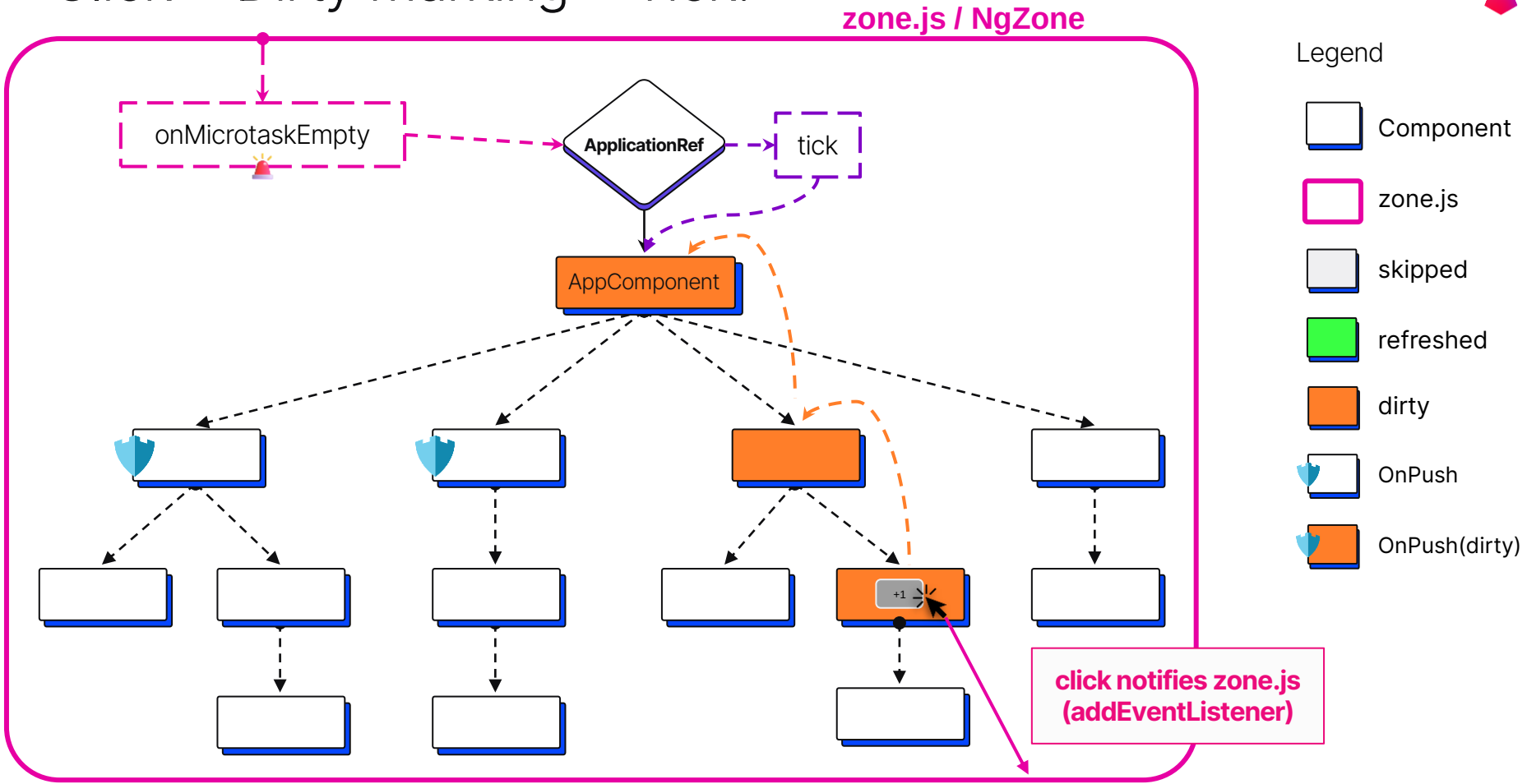
Marked two components as OnPush



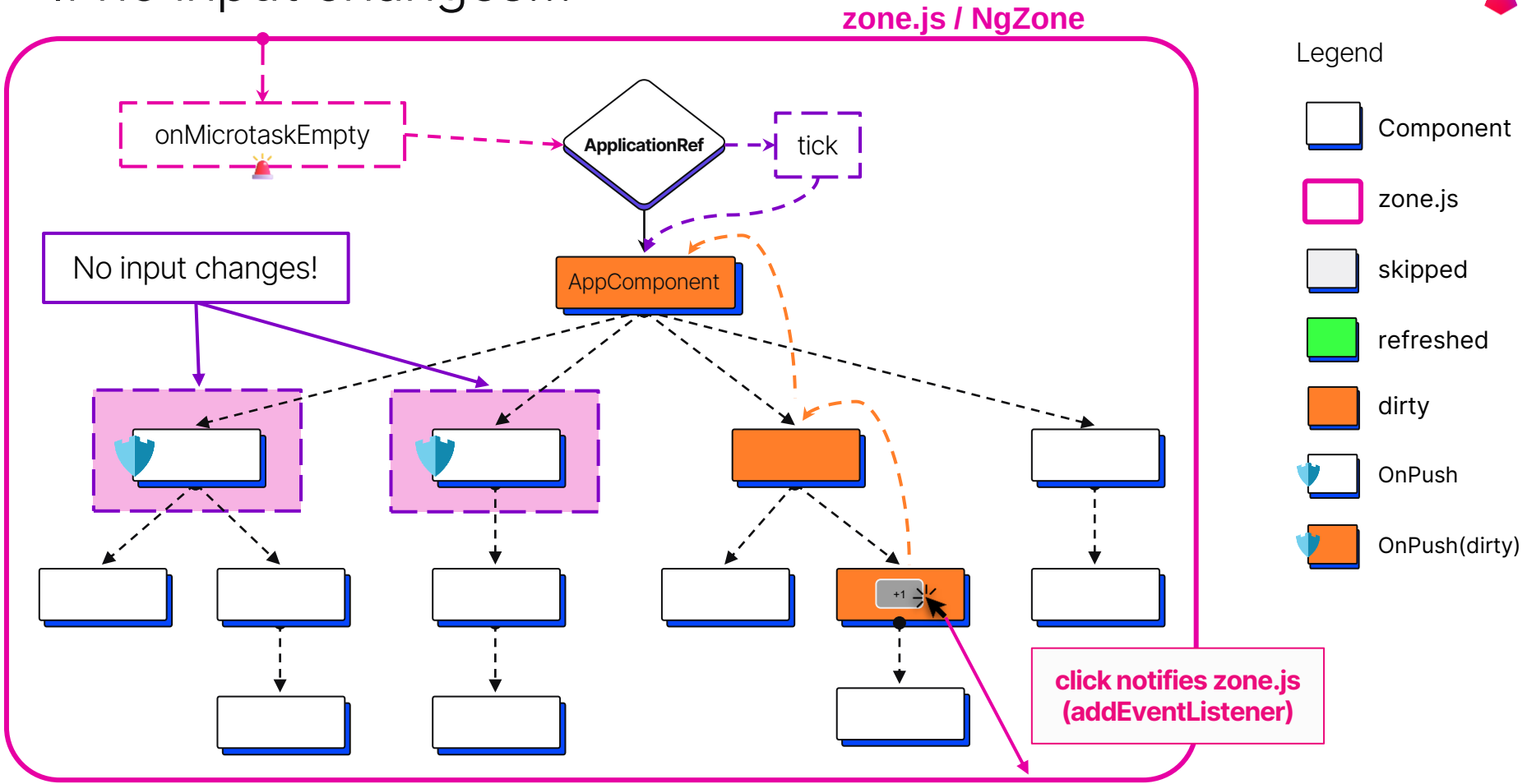
Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)

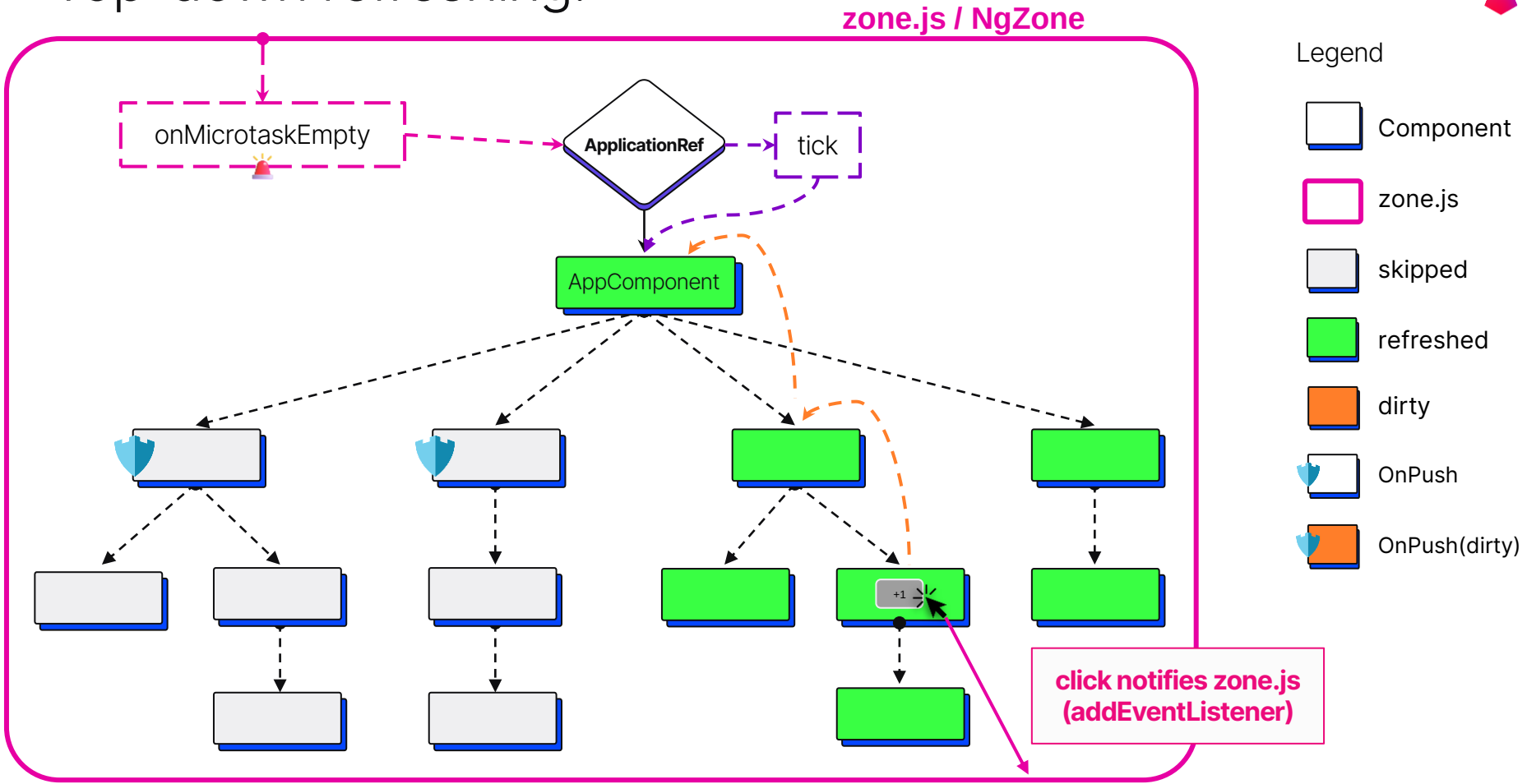
Click + Dirty Marking + Tick!



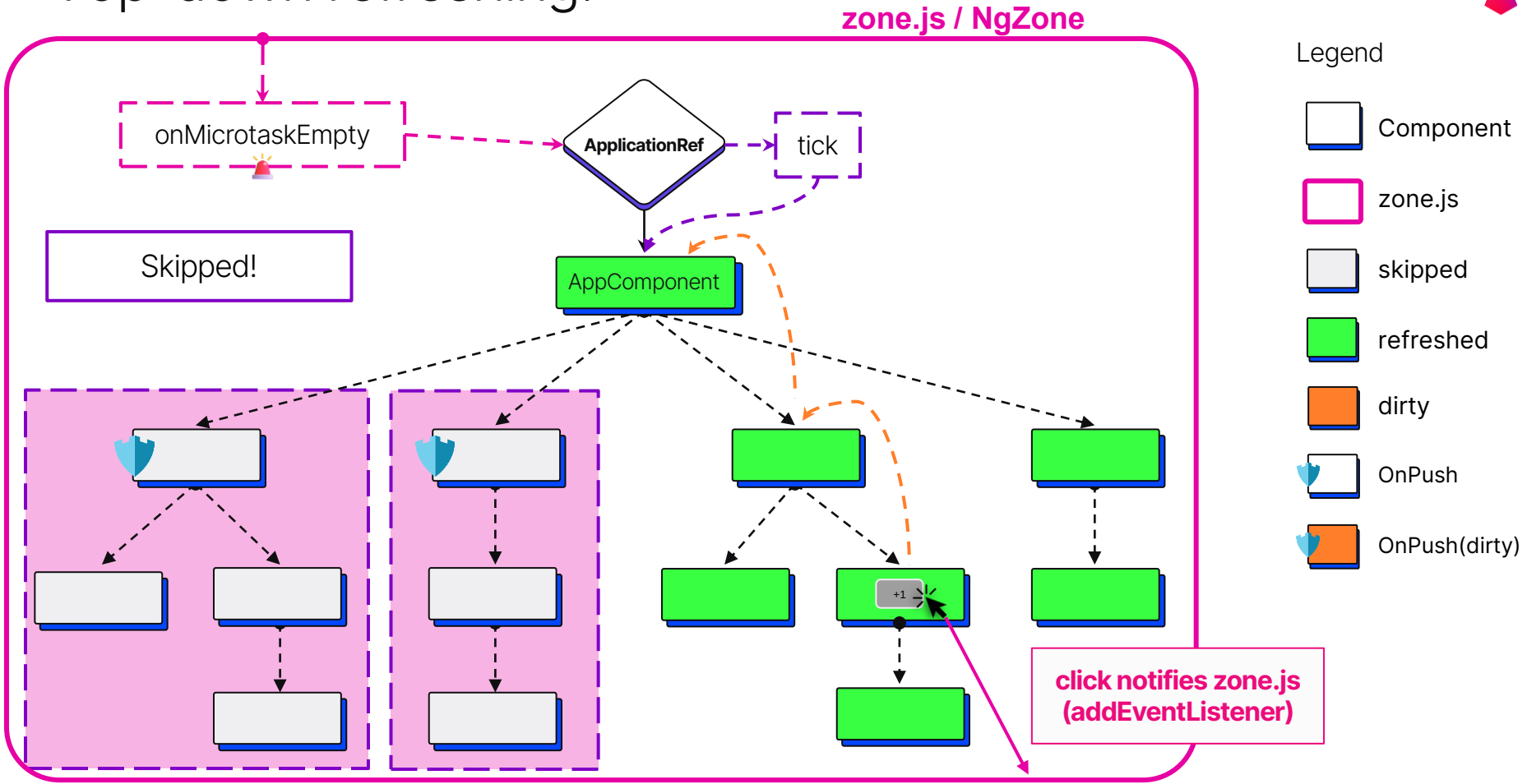
If no input changes...



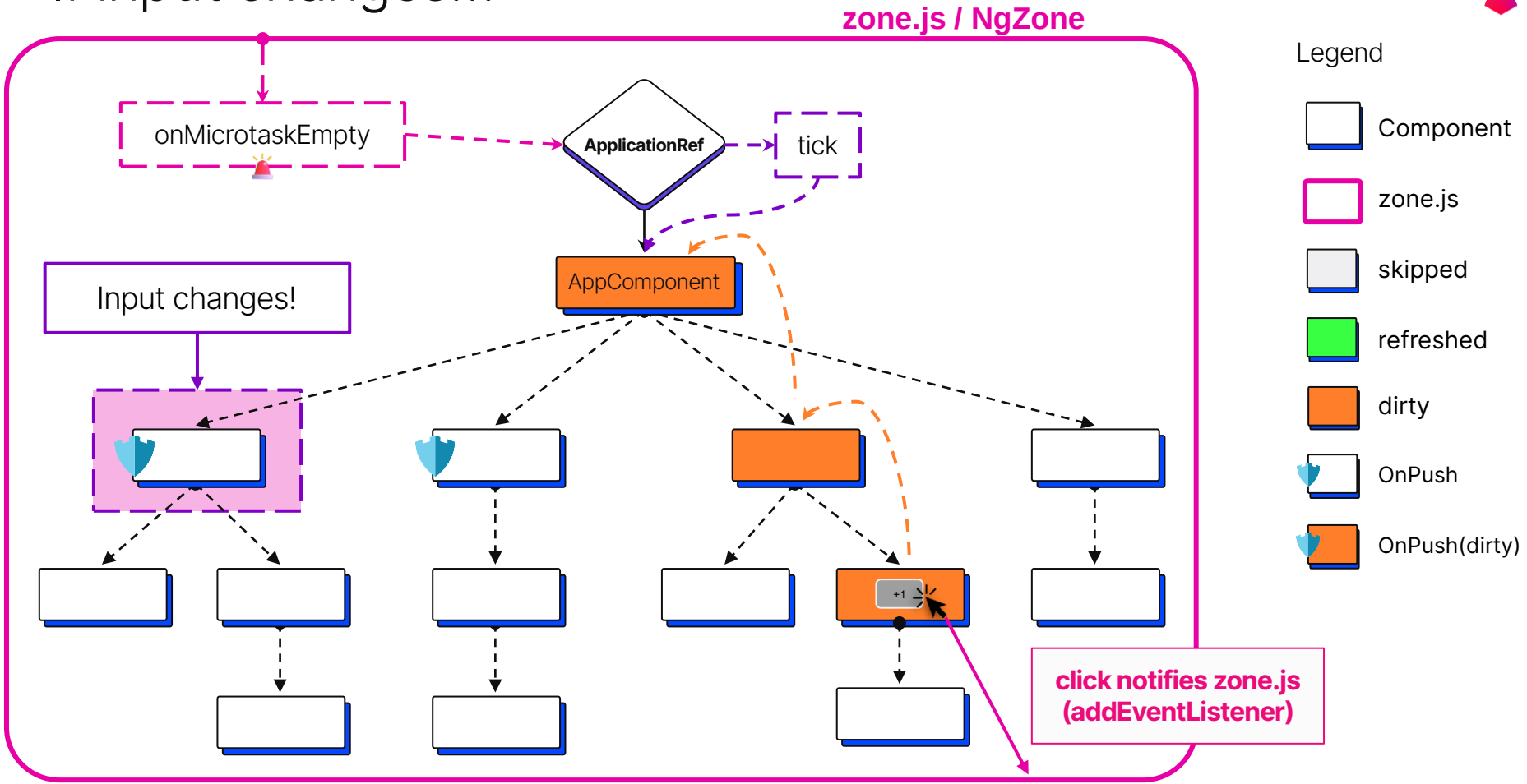
Top-down refreshing!



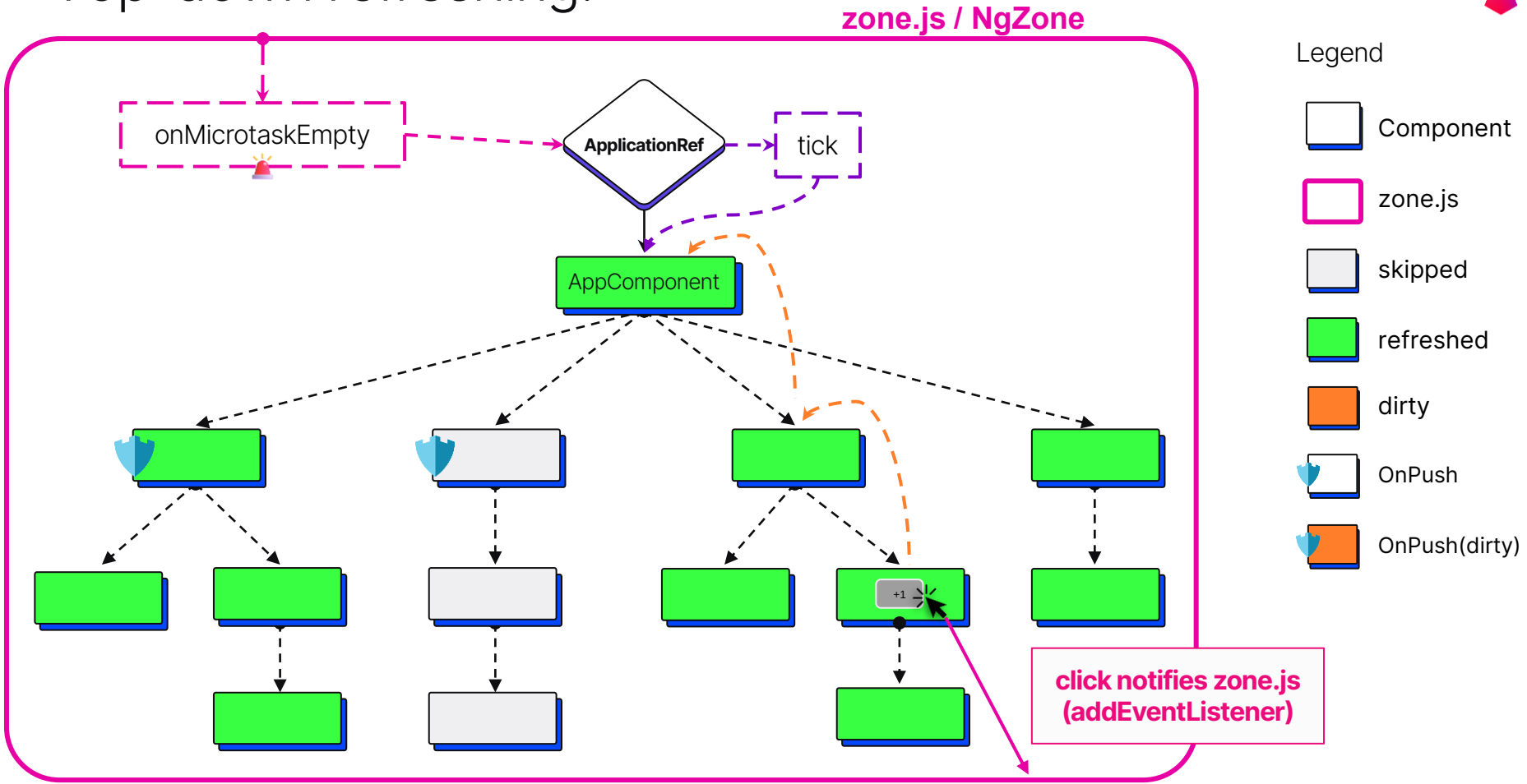
Top-down refreshing!



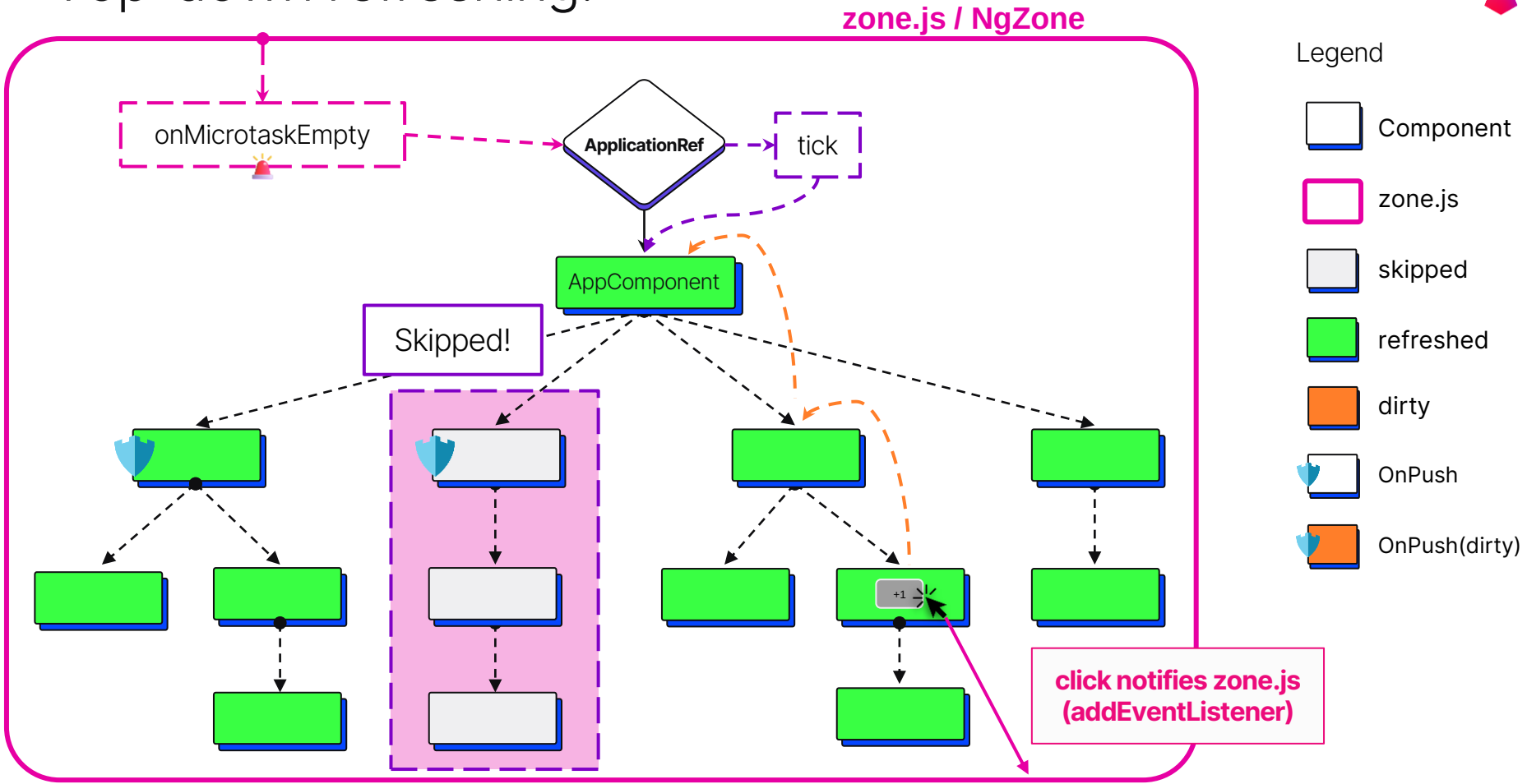
If input changes...



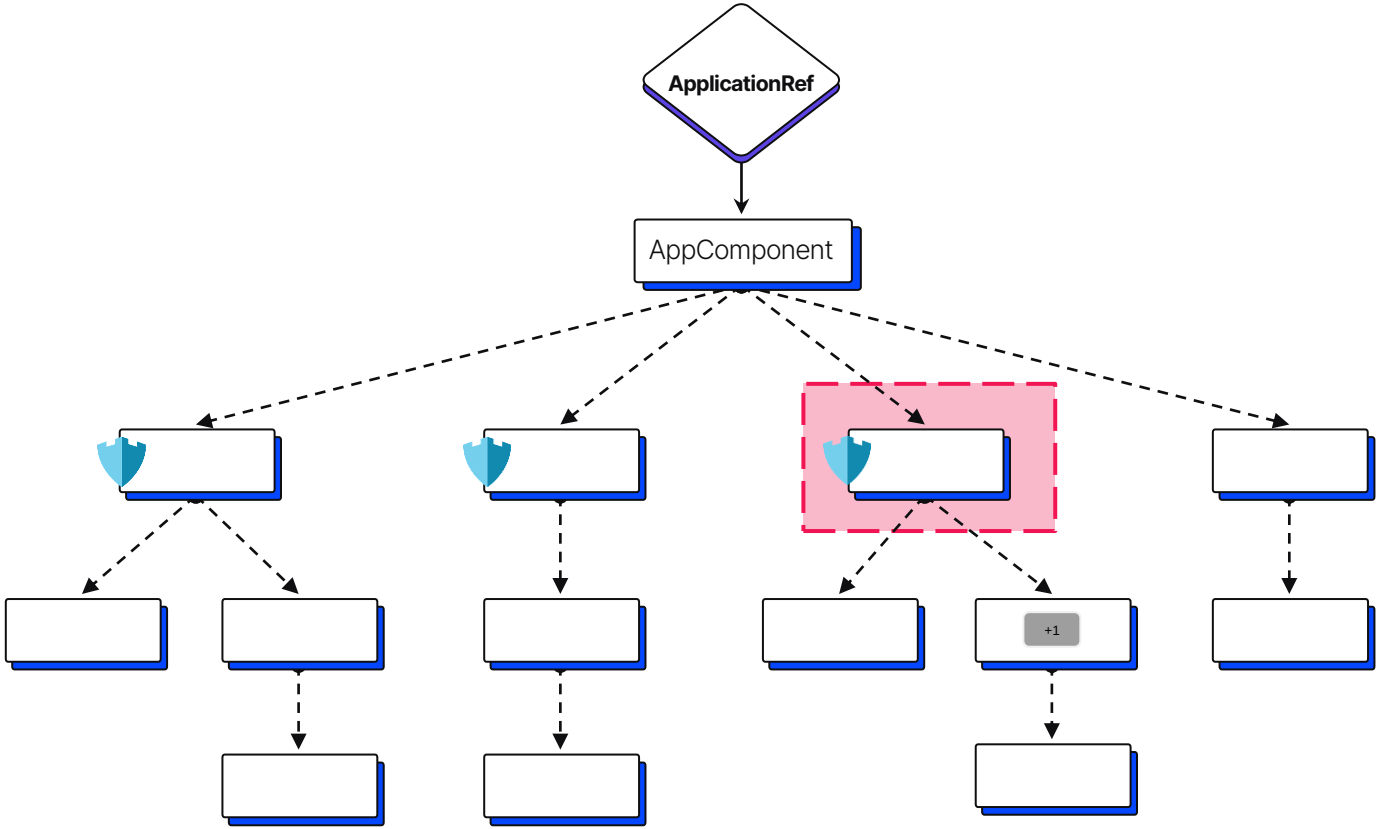
Top-down refreshing!



Top-down refreshing!



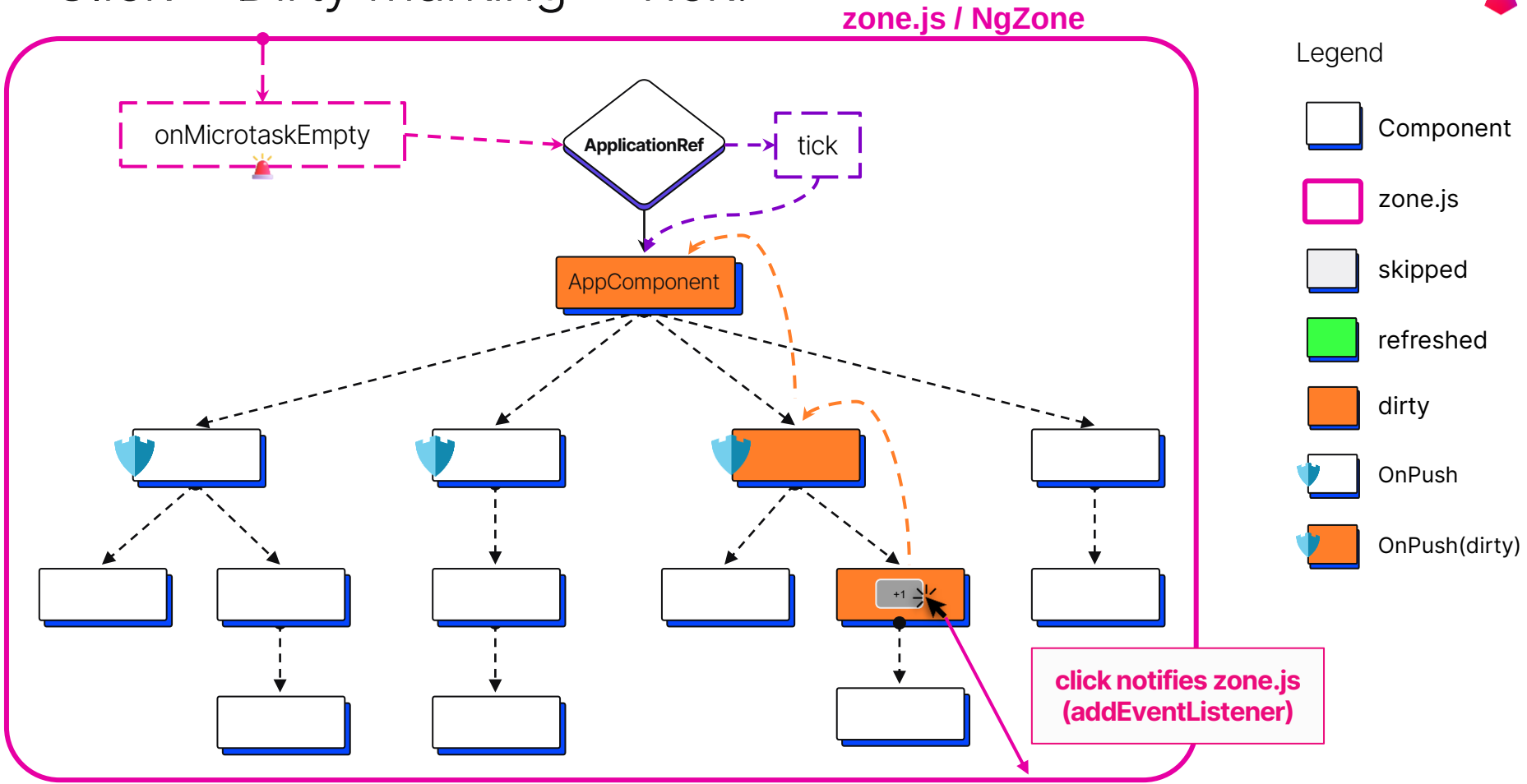
Marked **three** components as OnPush



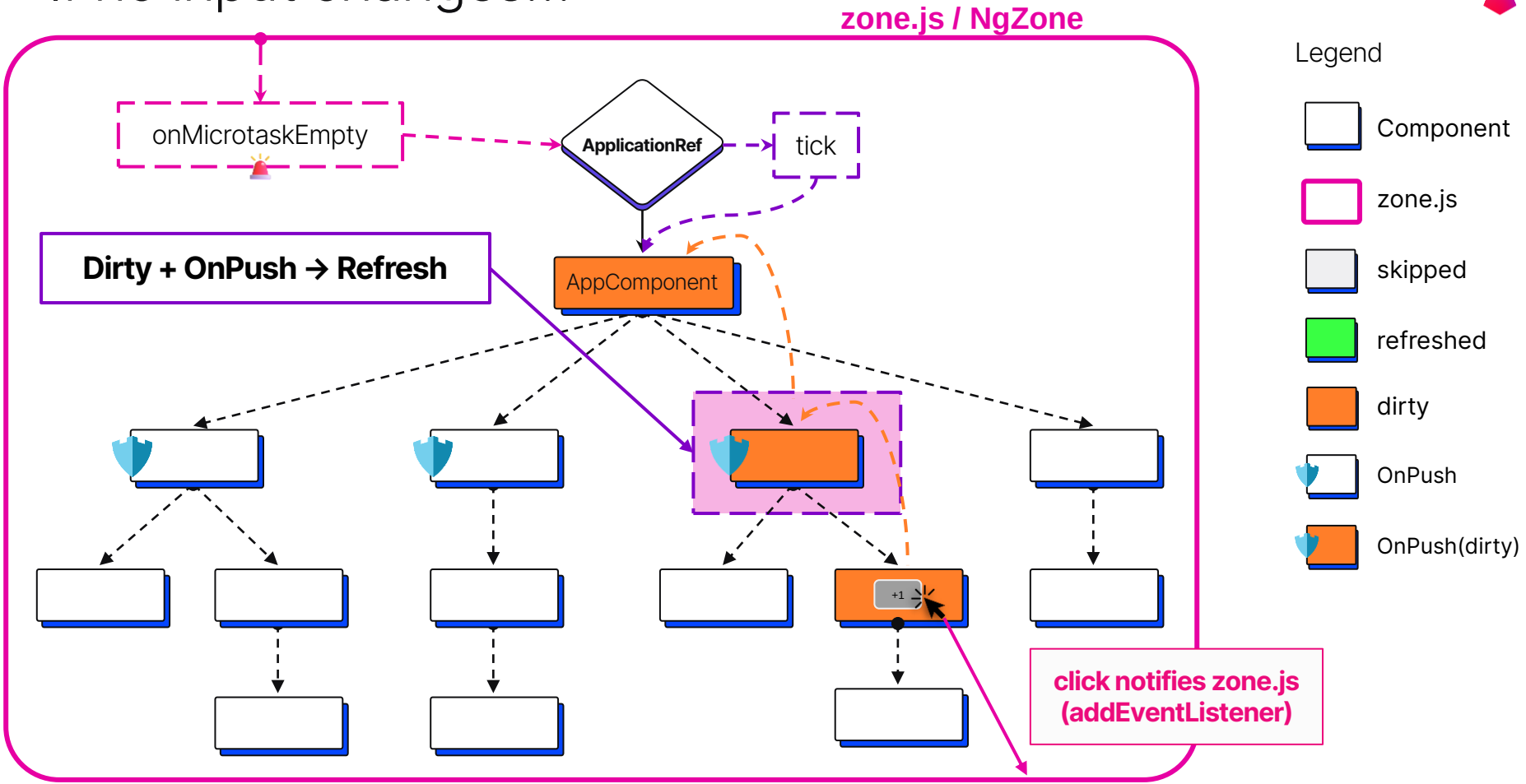
Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)

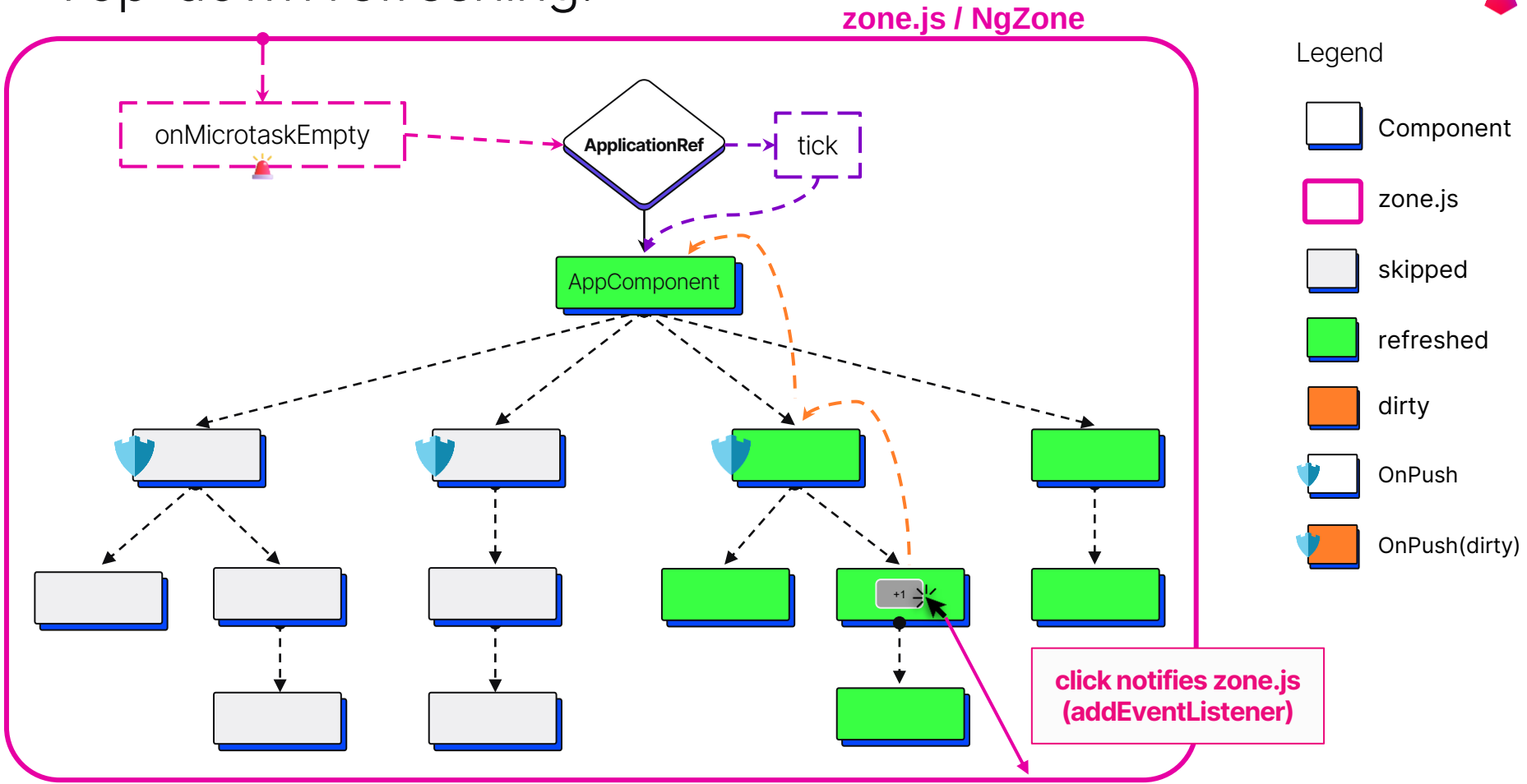
Click + Dirty Marking + Tick!



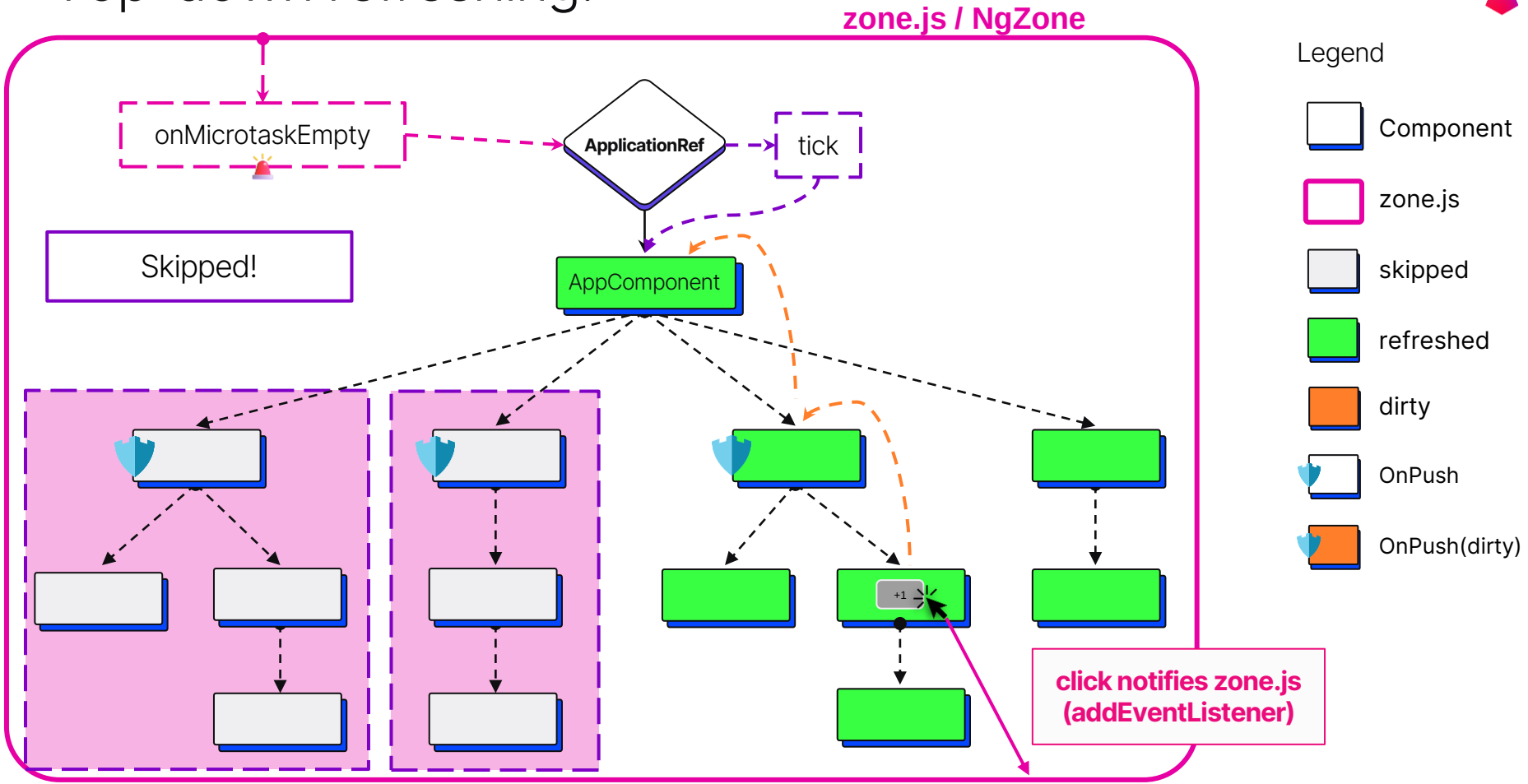
If no input changes...



Top-down refreshing!



Top-down refreshing!



Default CD

⚠️ Too much checking!

OnPush CD

 Skips branches and does less checking!

Async Pipe

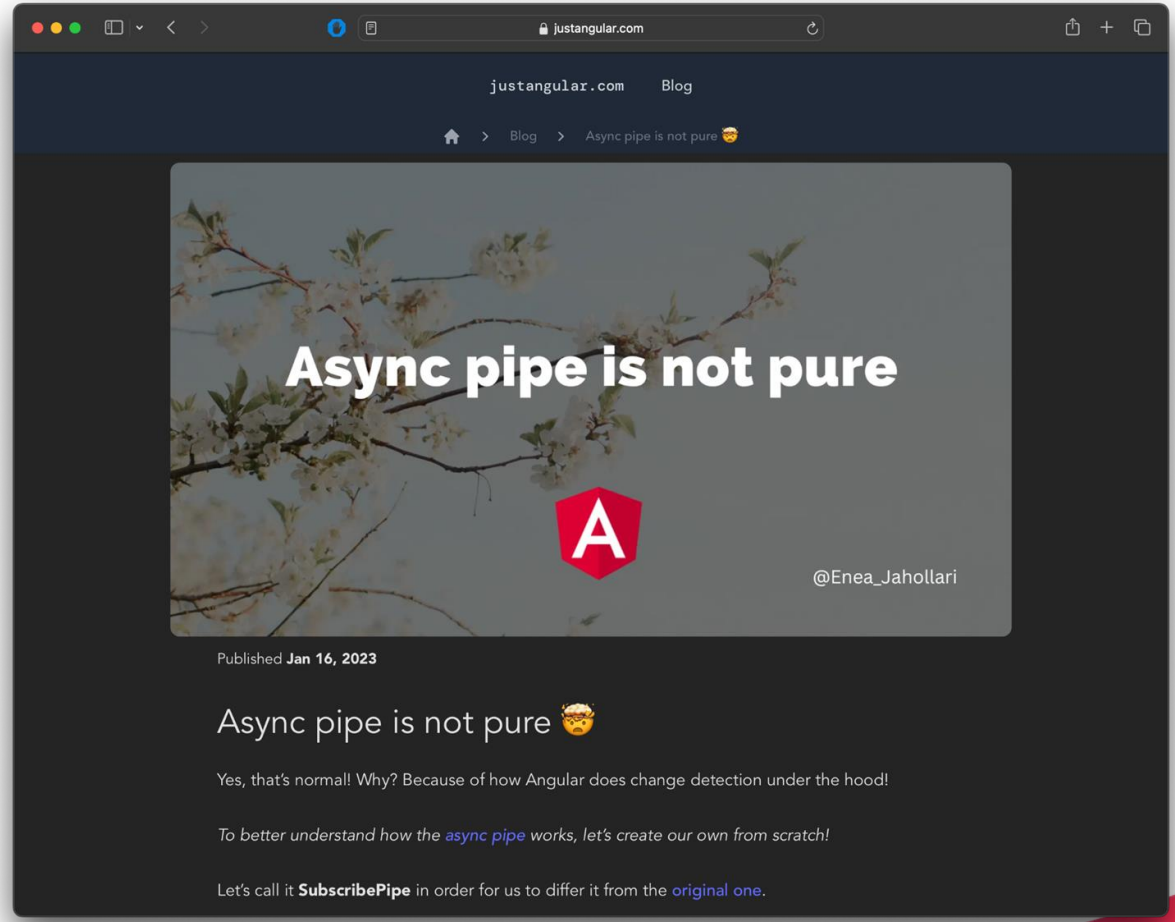
Does it really do any wonders 🧐?

It just uses **markForCheck** under the hood
🤷

mark_view_dirty.ts

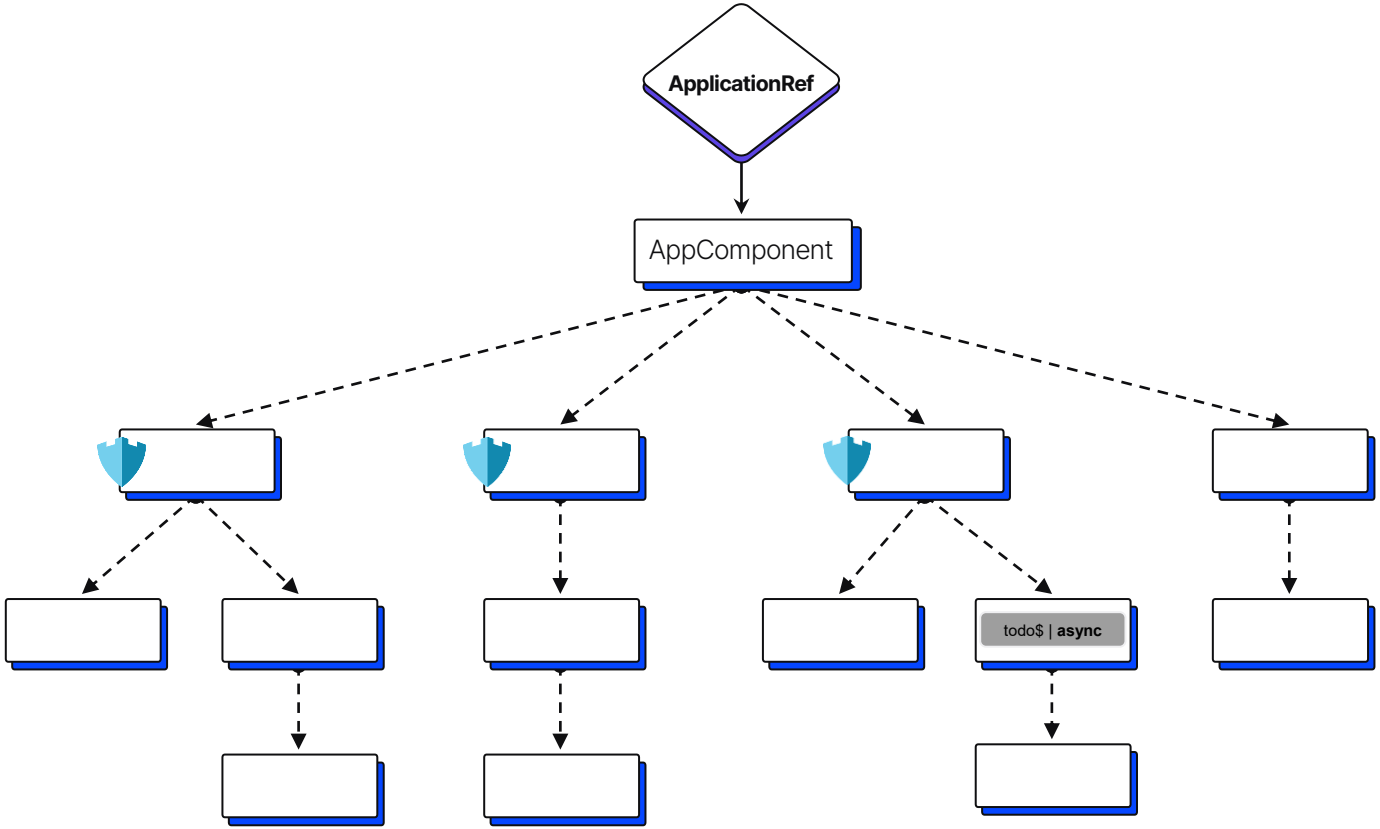
```
@Pipe()
export class AsyncPipe implements OnDestroy, PipeTransform {
  constructor(ref: ChangeDetectorRef) {}

  transform<T>(obj: Observable<T>): T|null {
    // code removed for brevity
  }

  private _updateLatestValue(async: any, value: Object): void {
    // code removed for brevity
    this._ref!.markForCheck();
  }
}
```

Editor Preview Both



https://justangular.com/blog/async-pipe-is-not-pure

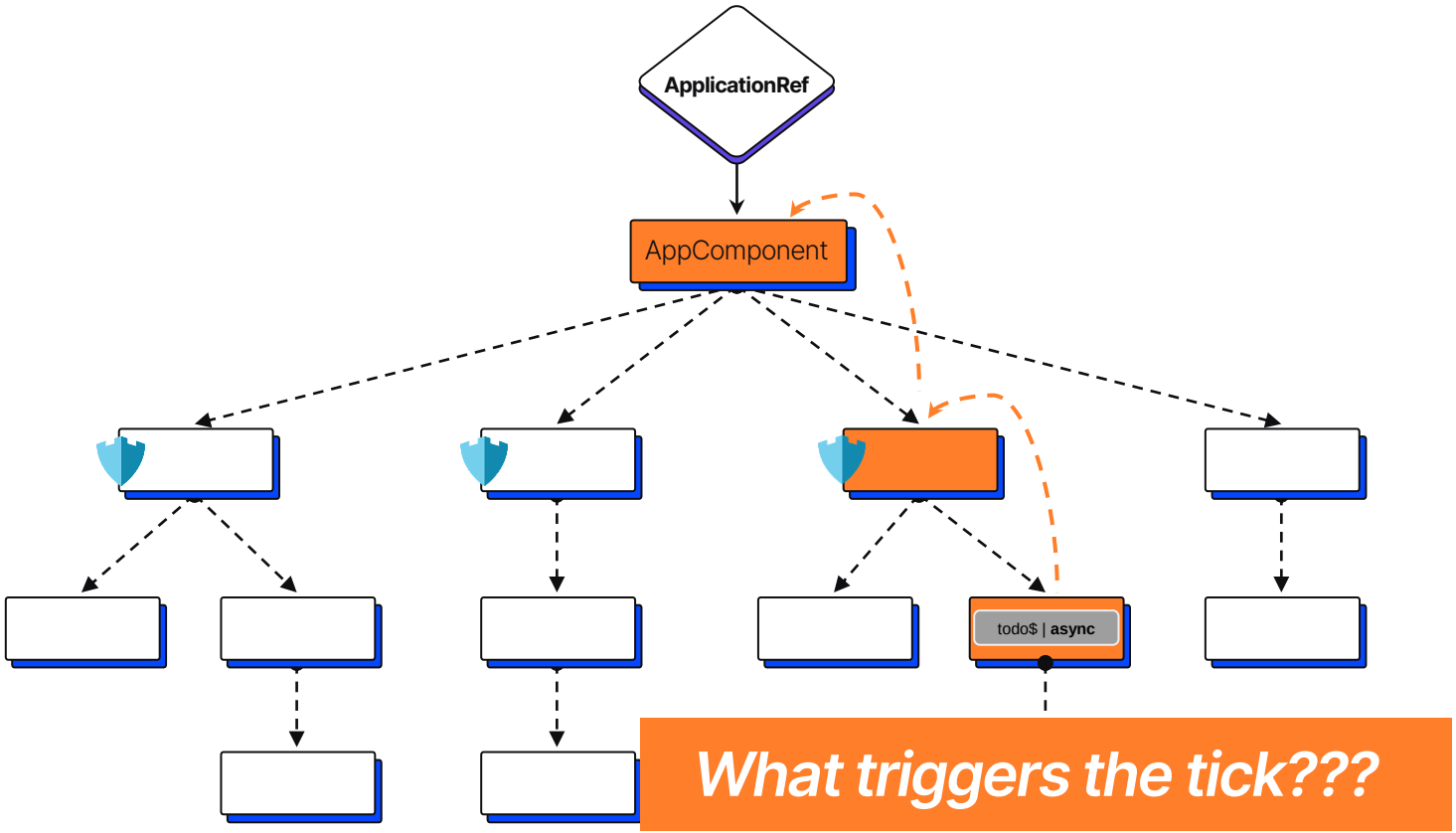
Let’s use the **async** pipe!



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)

Dirty Marking (bottom -> top)



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)

todos.ts

```
todo$ = this.http.get('/todo/1'); // uses XMLHttpRequest
```

Editor Preview Both

todos.ts

```
todo$ = this.http.get('/todo/1'); // uses XMLHttpRequest

todo$ = new BehaviorSubject<Todo | null>(null);
onClick() { // click handler
    this.todo$.next({ name: 'Click handler!' });
}
```

Editor Preview Both

todos.ts

```
todo$ = this.http.get('/todo/1'); // uses XMLHttpRequest

todo$ = new BehaviorSubject<Todo | null>(null);
onClick() { // click handler
    this.todo$.next({ name: 'Click handler!' });
}

ngOnInit() {
    this.todo$.next({ name: 'It's synchronous!' }); // Angular knows immediately
}
```

Editor Preview Both

todos.ts

```
todo$ = this.http.get('/todo/1'); // uses XMLHttpRequest

todo$ = new BehaviorSubject<Todo | null>(null);
onClick() { // click handler
    this.todo$.next({ name: 'Click handler!' });
}

ngOnInit() {
    this.todo$.next({ name: 'It's synchronous!' }); // Angular knows immediately

    setTimeout(() => { // setTimeout
        this.todo$.next({ name: 'Make it great!' });
    }, 1000);



}
```

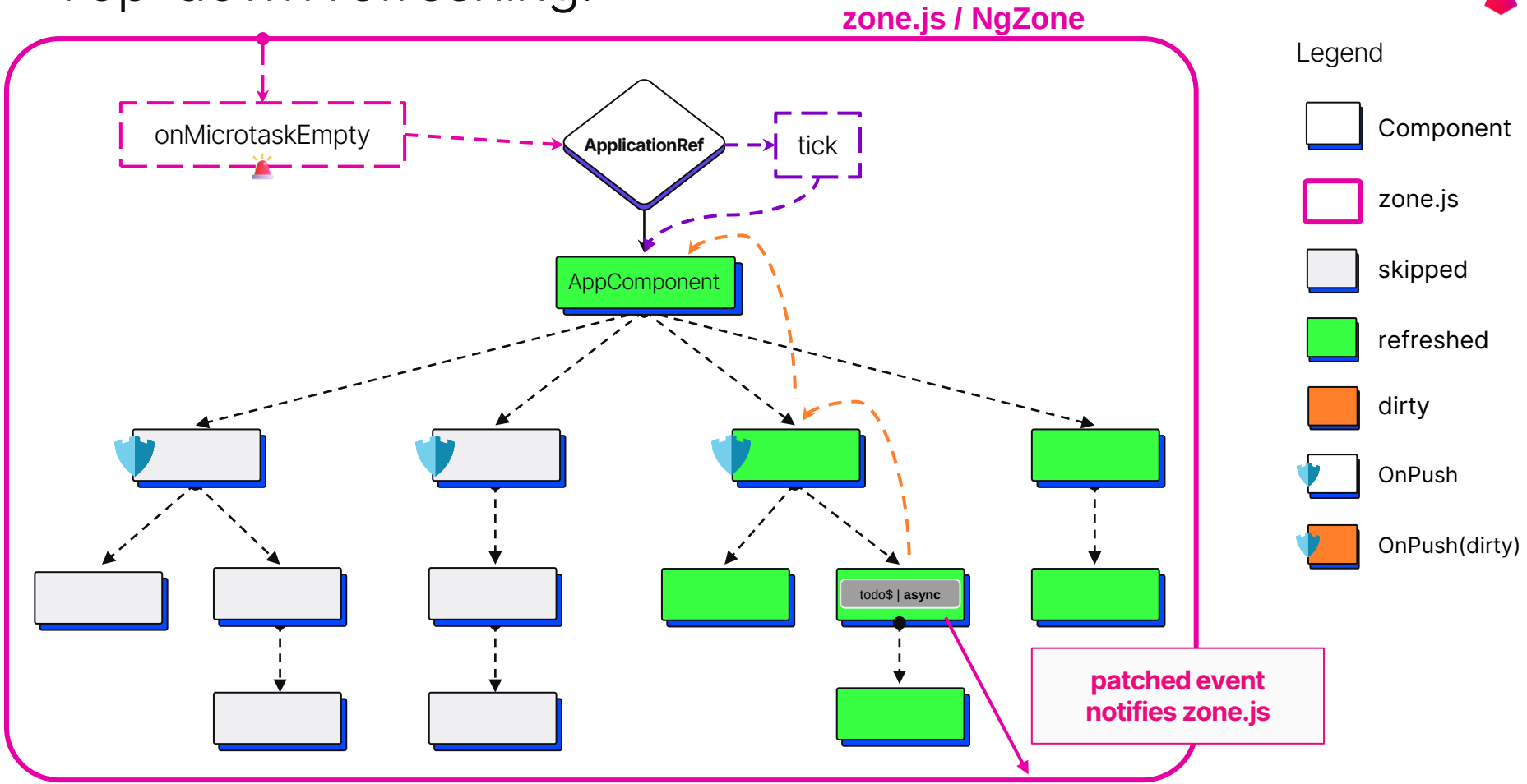
Editor Preview Both

todos.ts

```
todo$ = this.http.get('/todo/1'); // uses XMLHttpRequest

todo$ = new BehaviorSubject<Todo | null>(null);
onClick() { // click handler
    this.todo$.next({ name: 'Click handler!' });
}

ngOnInit() {
    this.todo$.next({ name: 'It's synchronous!' }); // Angular knows immediately

    setTimeout(() => { // setTimeout
        this.todo$.next({ name: 'Make it great!' });
    }, 1000);

    this.ngZone.runOutsideAngular(() => {
        setTimeout(() => {
            this.todo$.next({ name: 'Is it broken?' }); // Before v17 yes, in v18 it works!
        }, 1000);
    })
}
```

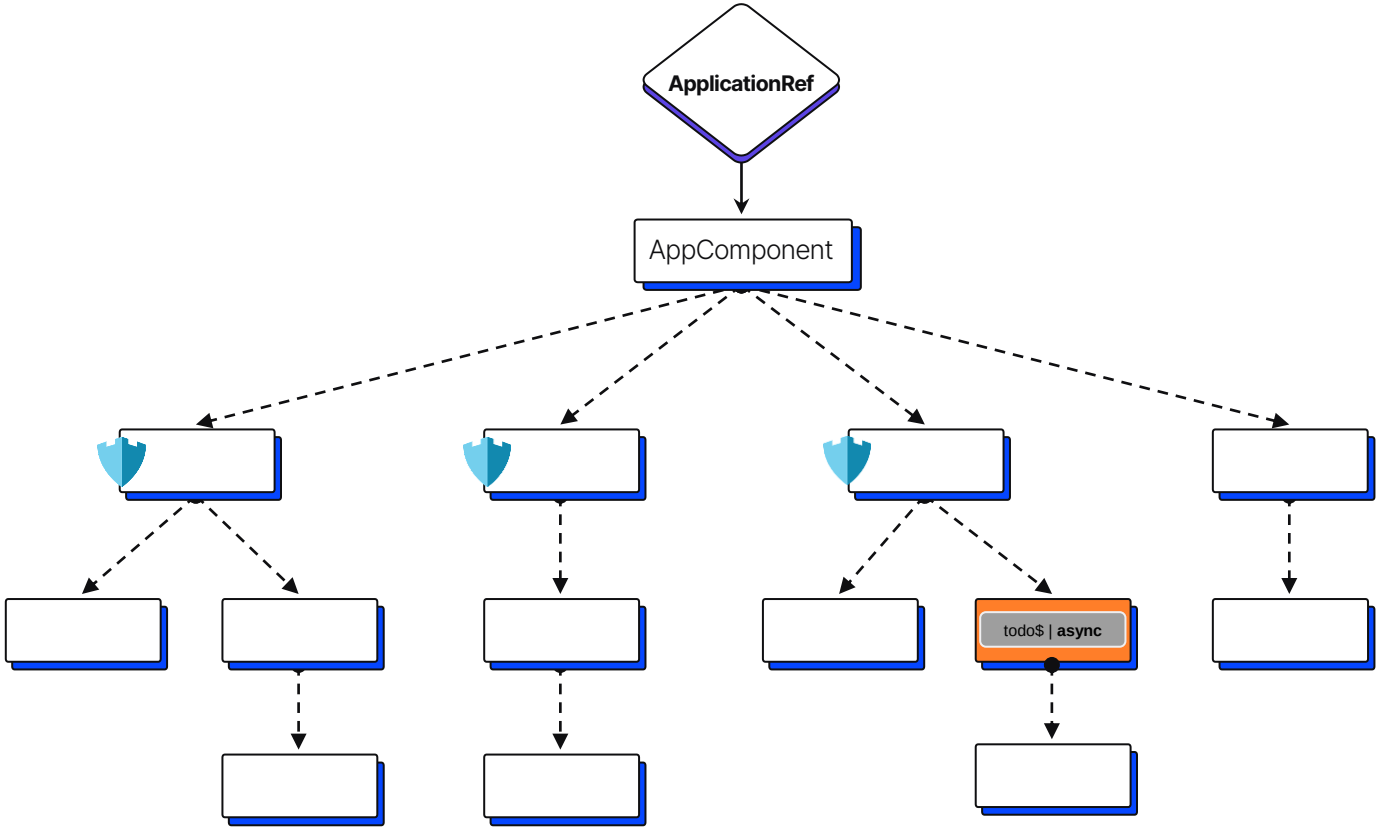
Editor Preview Both

Top-down refreshing!



Why do we need to mark all the ancestors dirty?

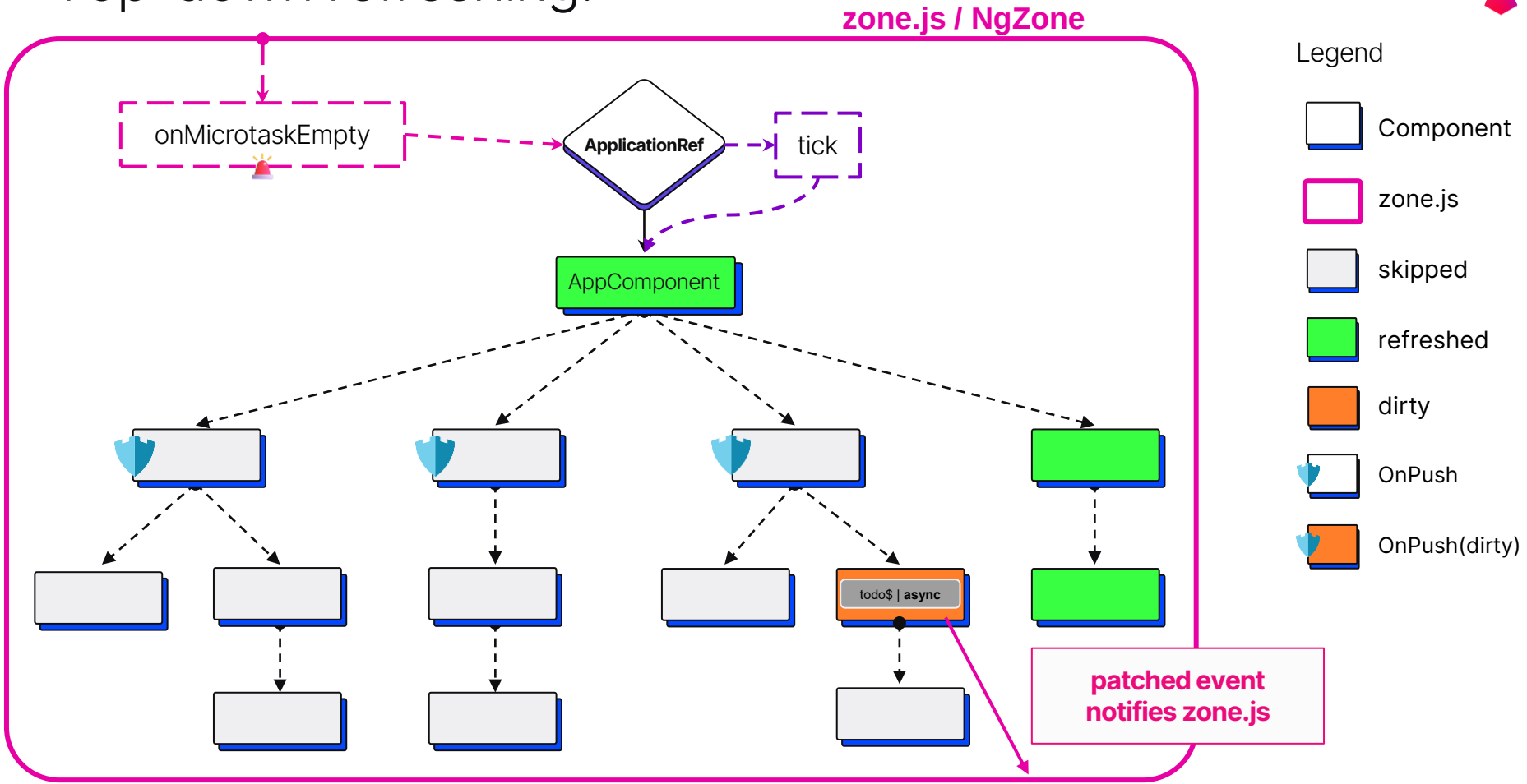
Dirty marking only the updated component



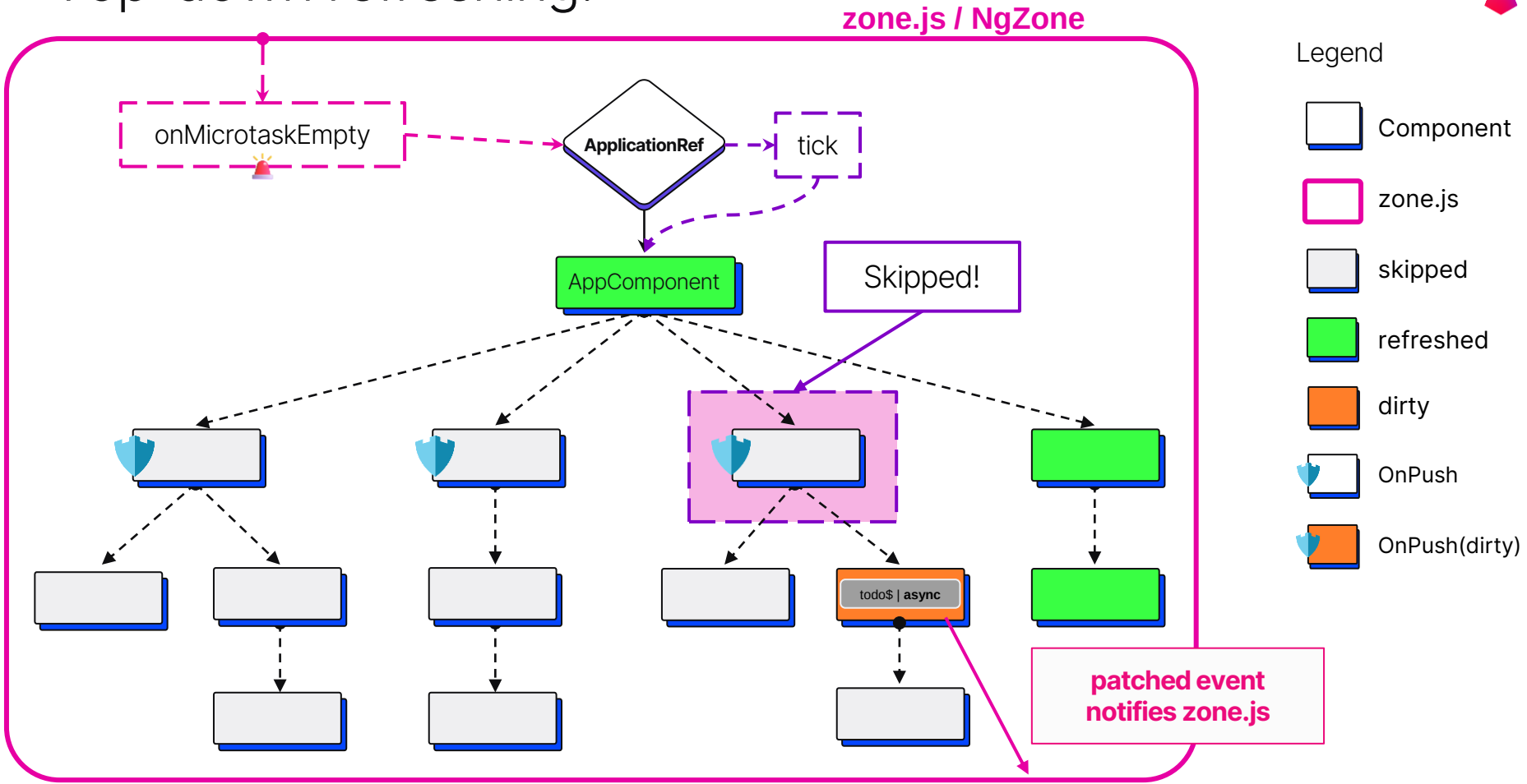
Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)

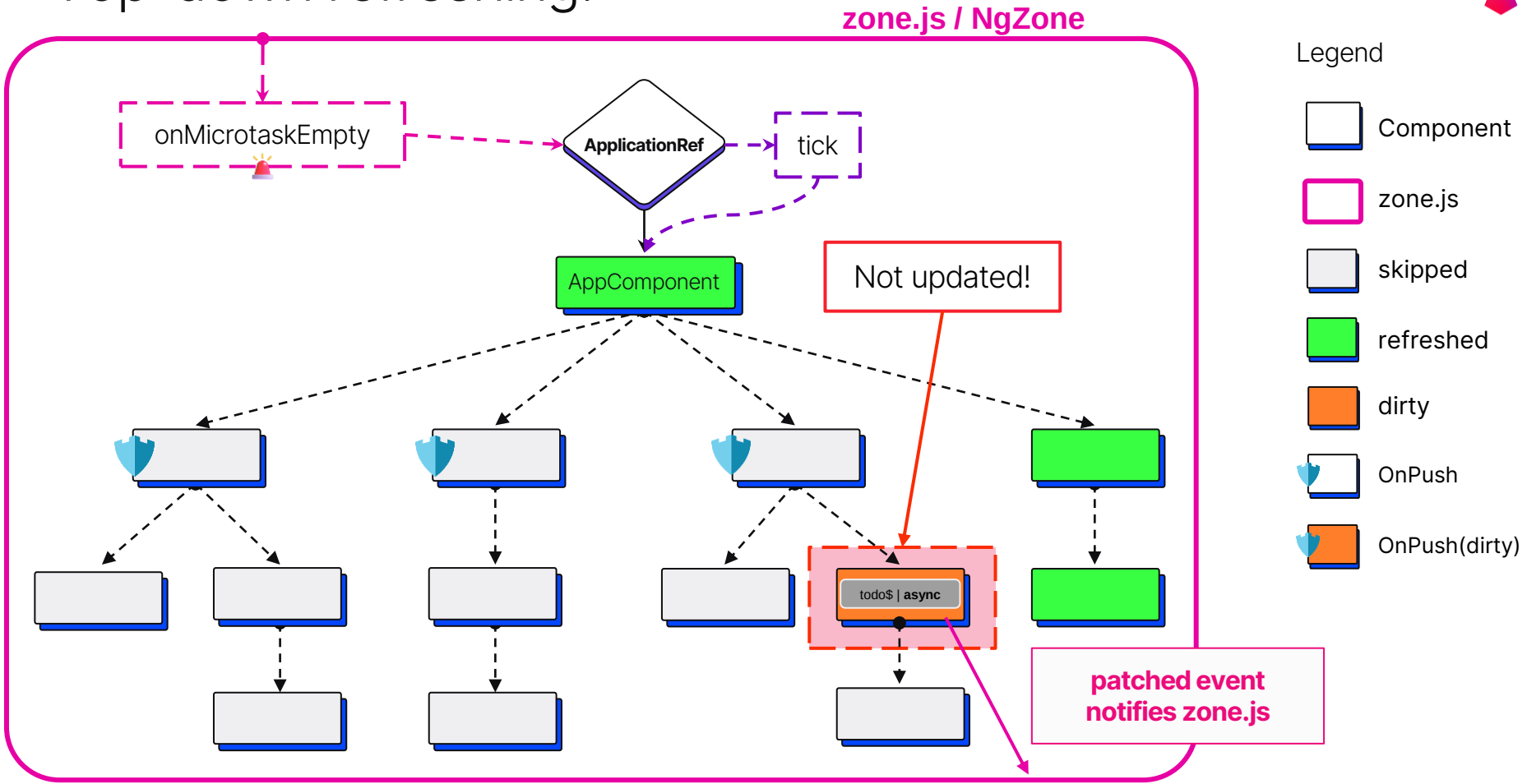
Top-down refreshing!



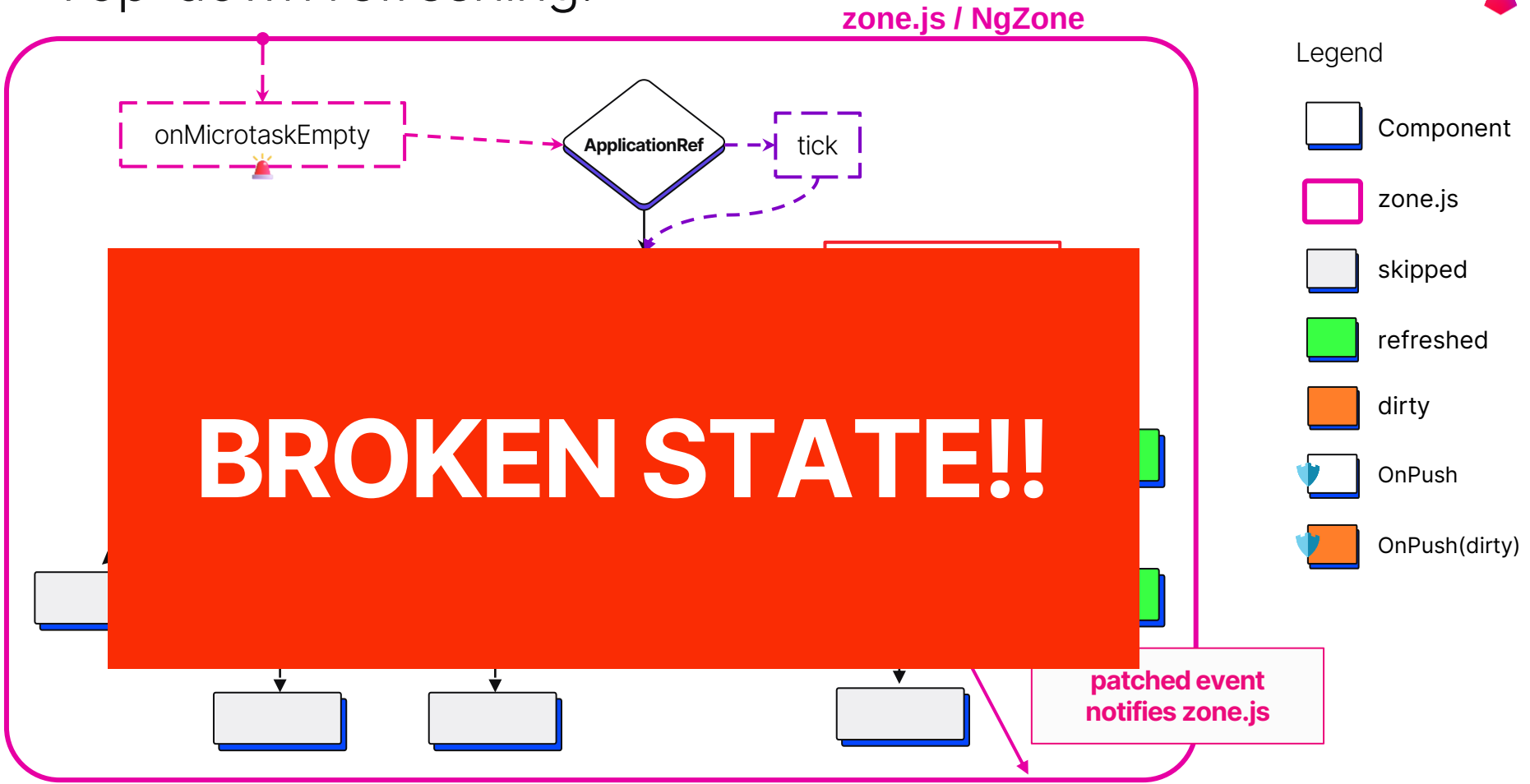
Top-down refreshing!



Top-down refreshing!



Top-down refreshing!



Can we do better?

signals.ts

```ts
// create a signal with initial value
const name = signal('John');

// create a computed signal
const upperCaseName = computed(() => name().toUpperCase());

effect(() => { // run side effect when name or upperCaseName changes
    console.log(name() + ' ' + upperCaseName());
});

setTimeout(() => { // change the name after 1 second
  name('Jane');
}, 1000);

// Output:
// John JOHN
// Jane JANE
```

Editor Preview Both

signal-in-template.ts

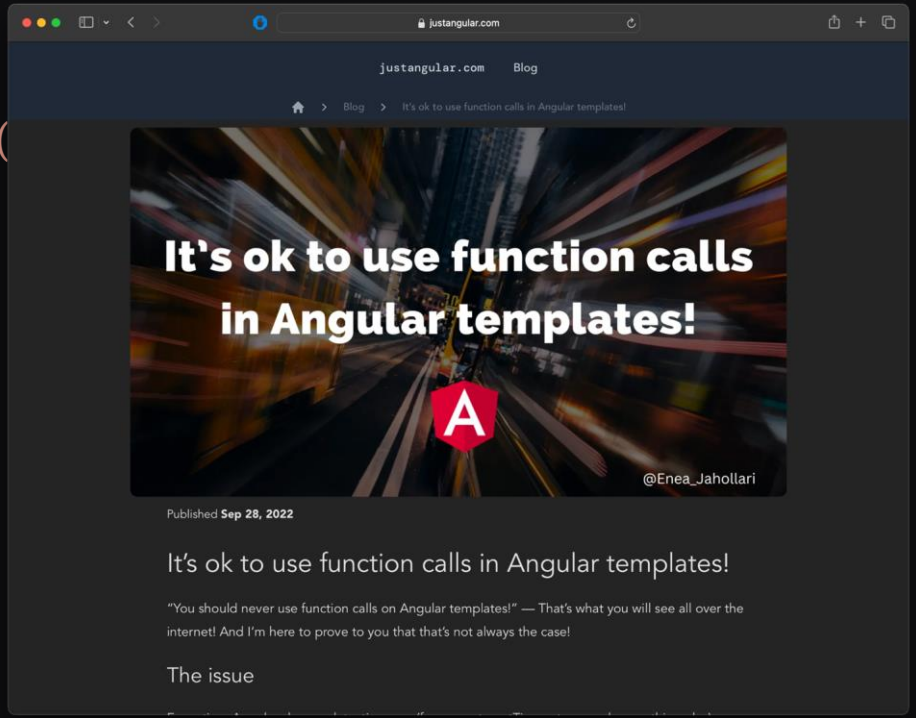
```ts
@Component({
  template: `
    <button (click)="name.set('Jane')">Change name</button>
    <p>{{ name() }}</p>
  `
})
export class AppComponent {
  name = signal('John');
}
```

Editor Preview Both

signal-in-template.ts

```ts
@Component({
  template: `
    <button (click)="name.set(
    <p>{{ name() }}</p>
  `
})
export class AppComponent {
  name = signal('John');
}
```

It's ok to use function calls in Angular templates!

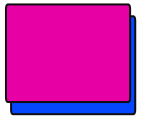


Editor Preview Both

Angular templates are now **effects (consumers)**!

Everytime a signal that is read inside a template is updated,
the component is marked as ...

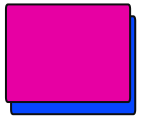
Signal Markers



RefreshView

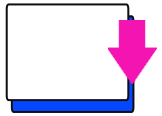
Tells Angular to refresh this view on every change detection

Signal Markers



RefreshView

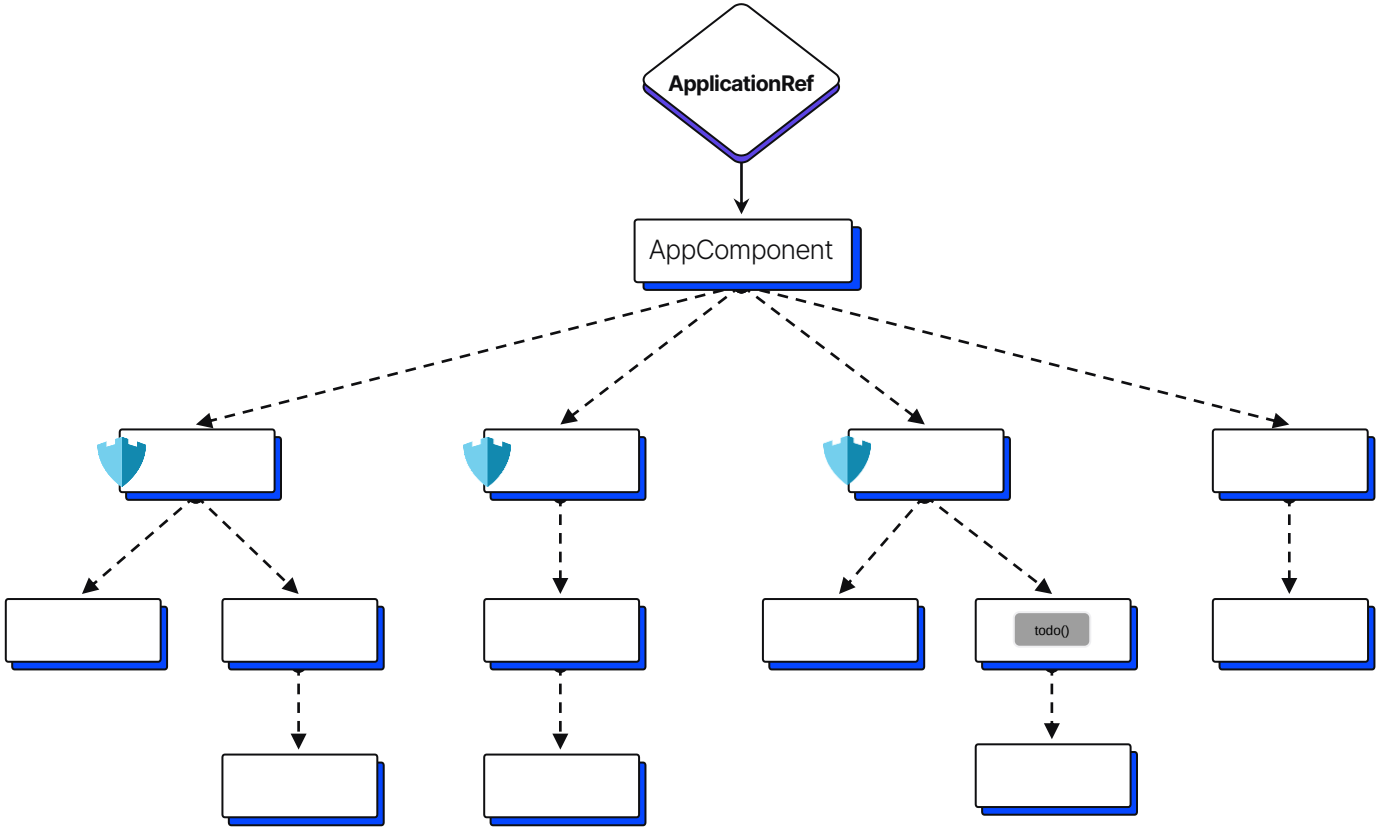
Tells Angular to refresh this view on every change detection



HasChildViewsToRefresh

Is applied to the ancestors of a component with **RefreshView**.

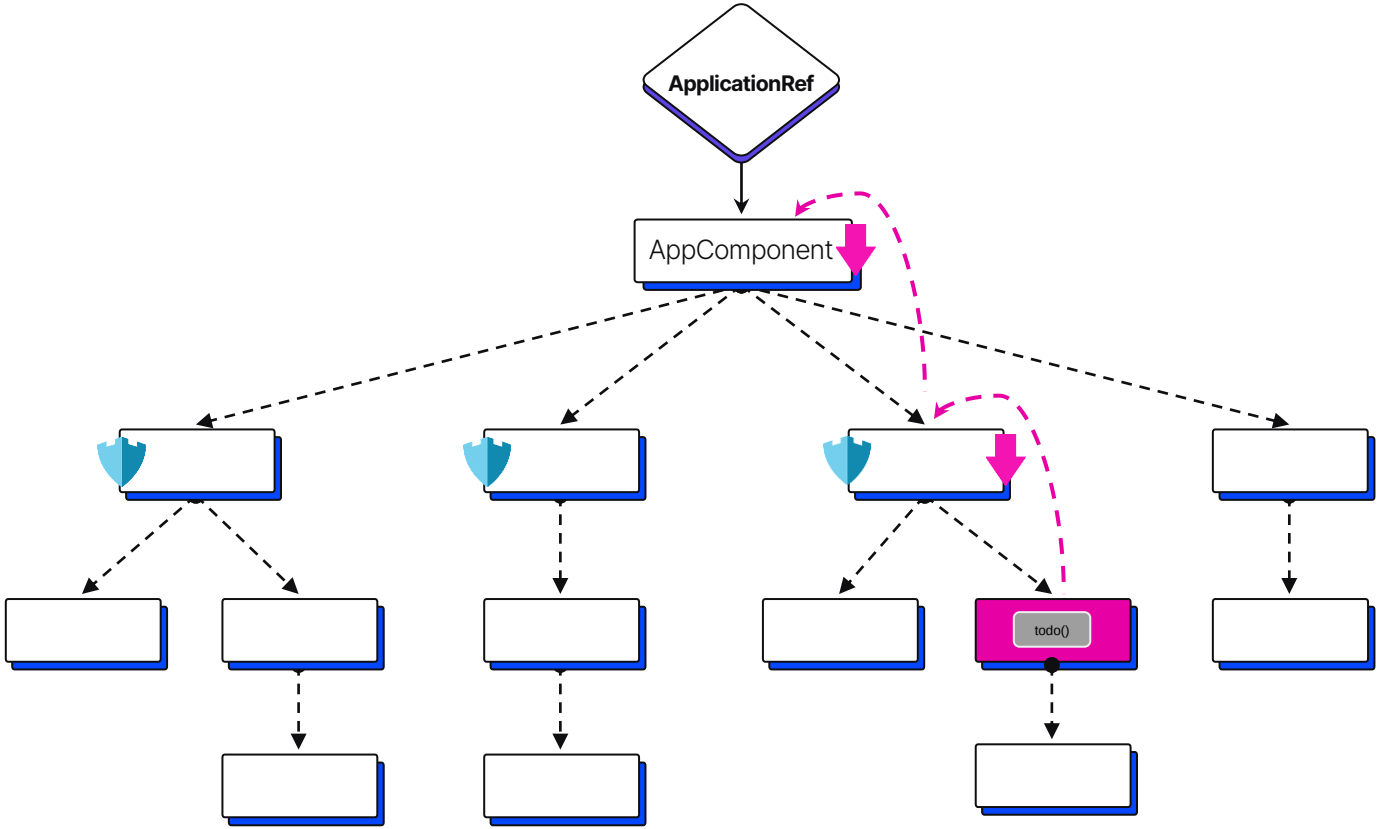
Signal read in the template



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

Signal update

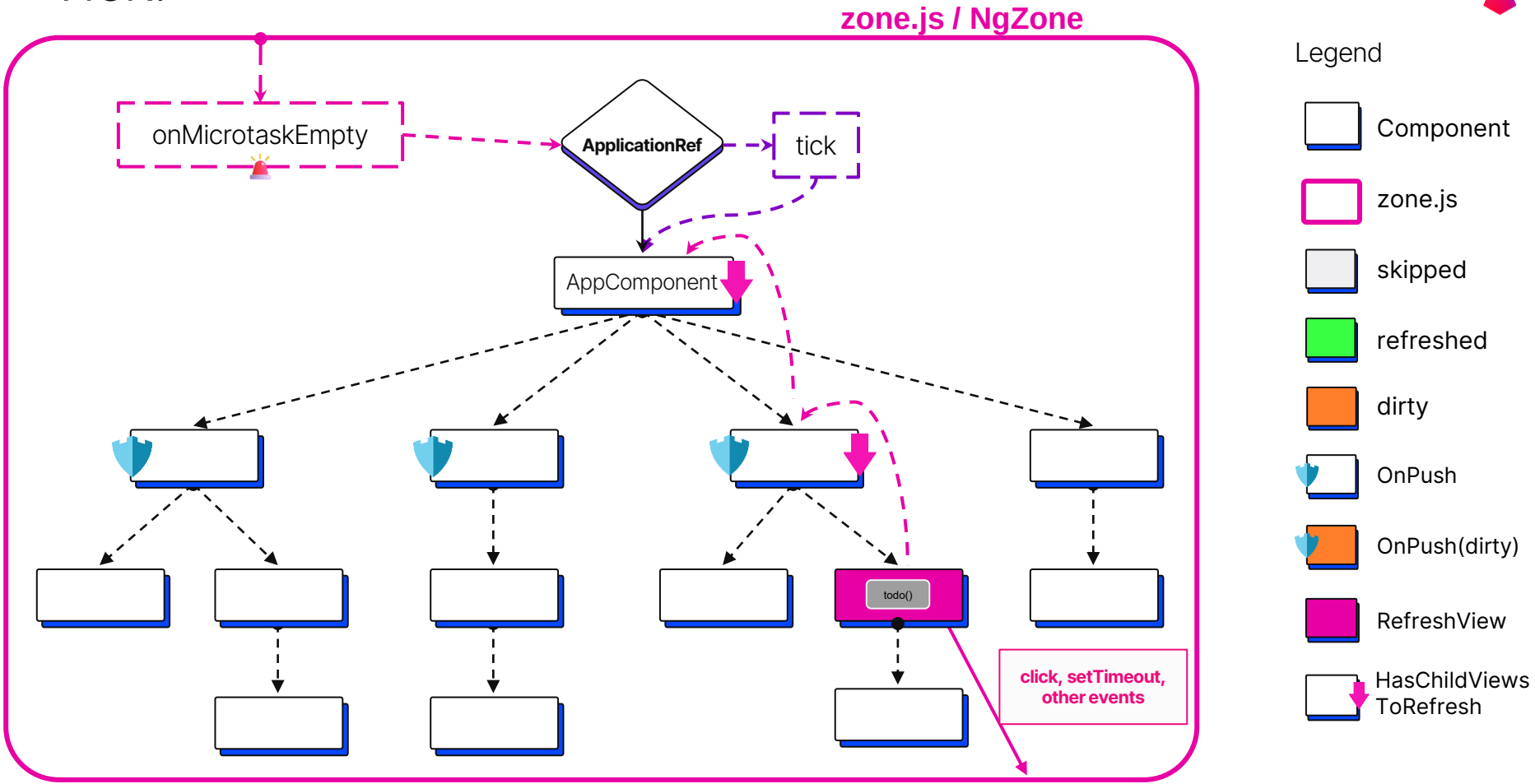


Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

In **v17** signals still rely on zone.js to trigger **tick()!**

Tick!


zone.js / NgZone
onMicrotaskEmpty
ApplicationRef
tick
AppComponent
todo()
click, setTimeout, other events
Legend
Component
zone.js
skipped
refreshed
dirty
OnPush
OnPush(dirty)
RefreshView
HasChildViews ToRefresh

New **Change Detection** Mechanism ✨ !

Targeted Mode

Change Detection Modes

Global

Refreshes:

- **CheckAlways** (default CD strategy)
- **OnPush + Dirty**

Change Detection Modes

Global

Refreshes:

- **CheckAlways** (default CD strategy)
- **OnPush + Dirty**

Activates on:
OnPush + HasChildViewsToRefresh

Targeted

Change Detection Modes

Global

Refreshes:

- **CheckAlways** (default CD strategy)
- **OnPush + Dirty**

Activates on:
OnPush +
HasChildViewsToRefresh

Targeted

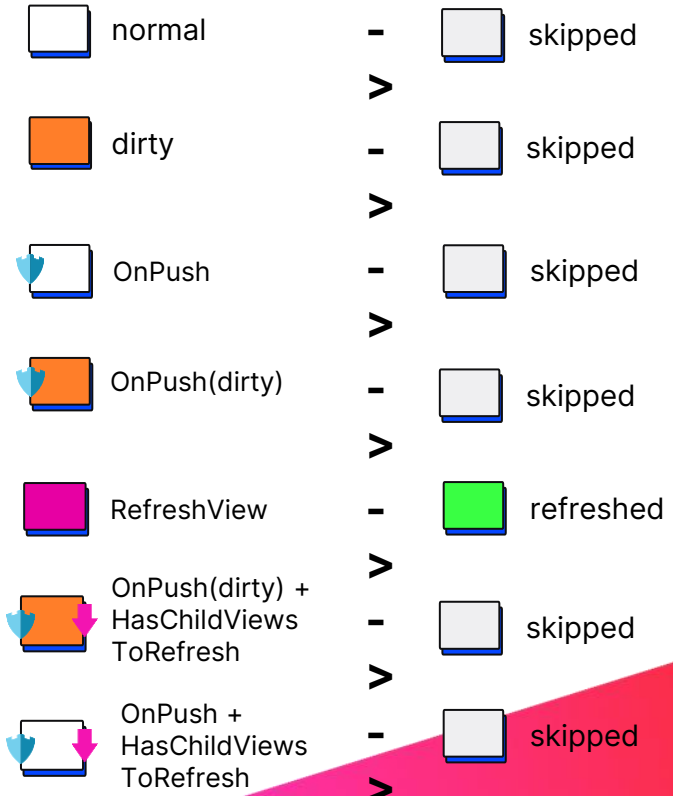
Refreshes:

- **RefreshView**
  (traverse children in **GlobalMod**

Rules

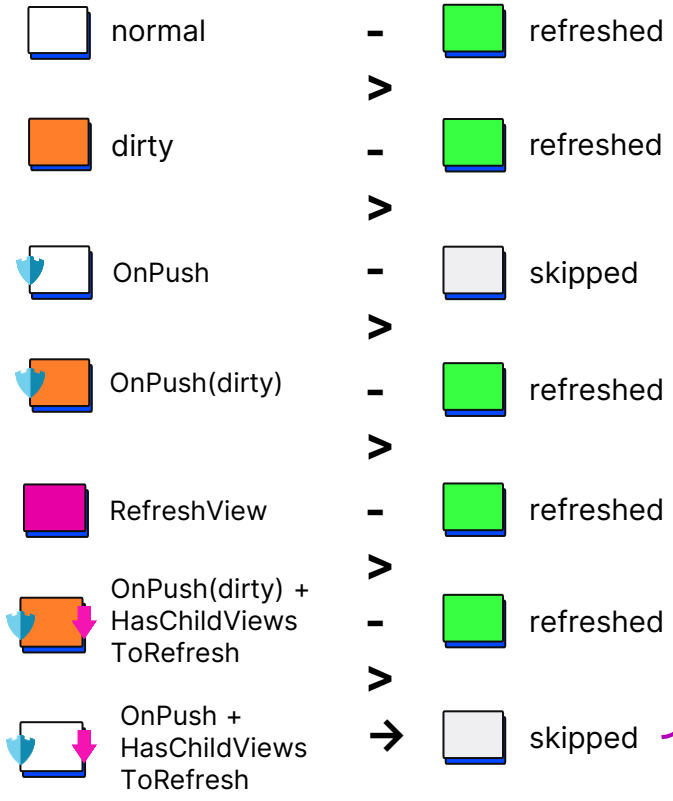
Global Mode

Targeted Mode

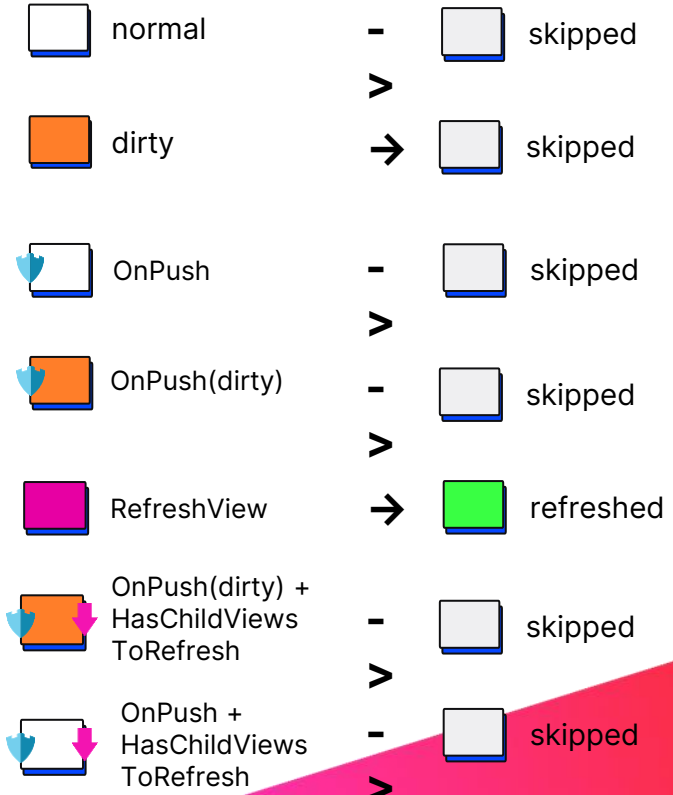


Rules

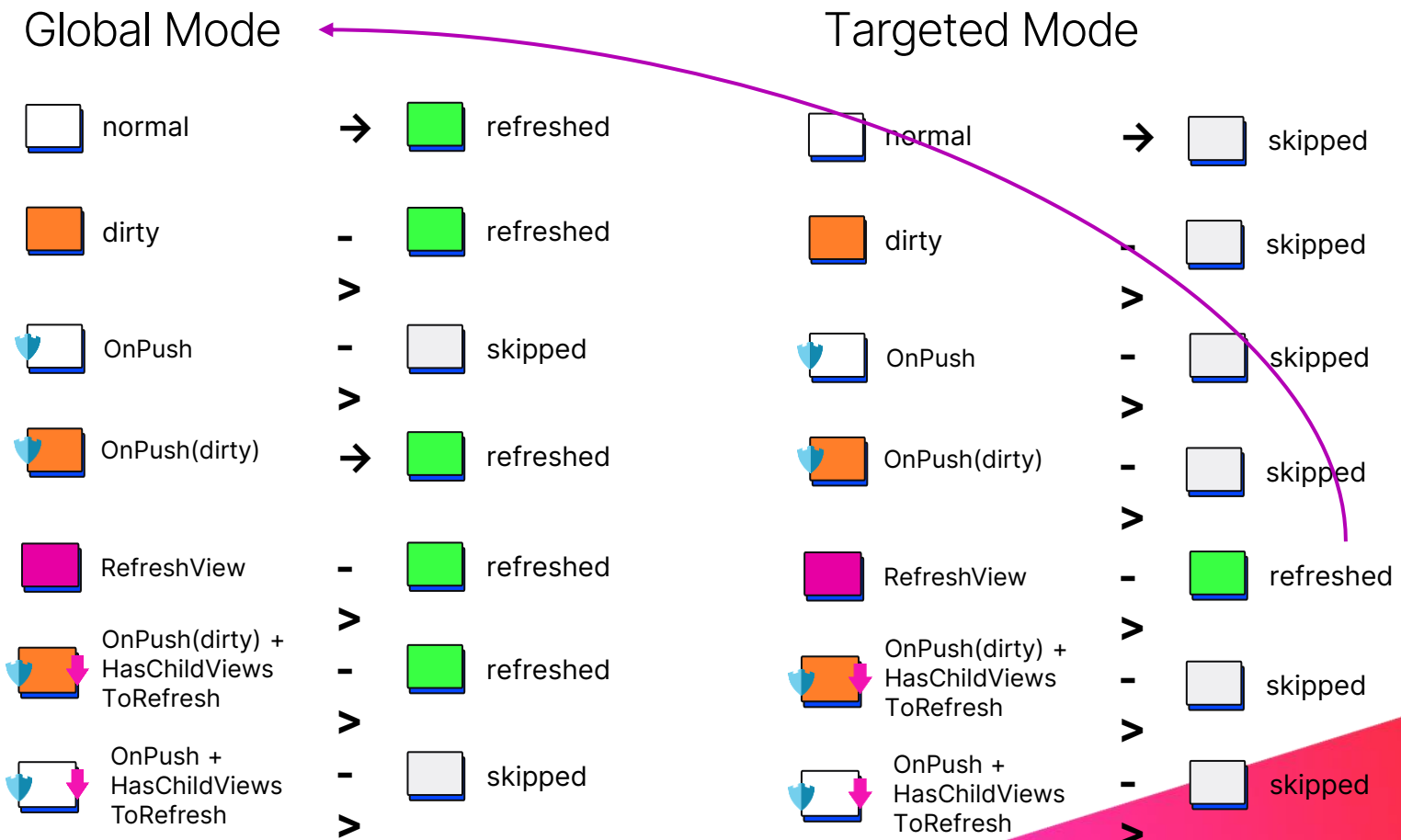
Global Mode



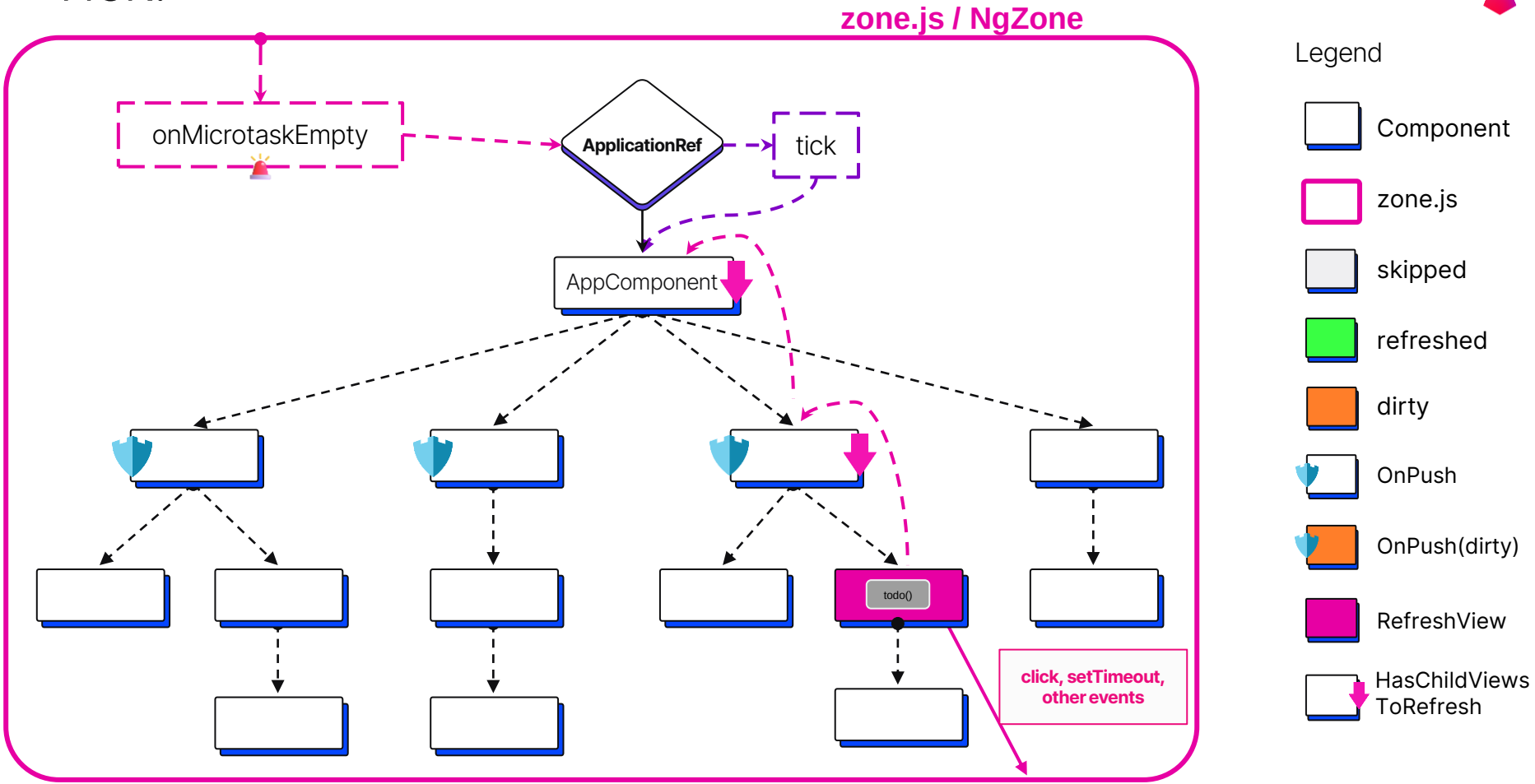
Targeted Mode



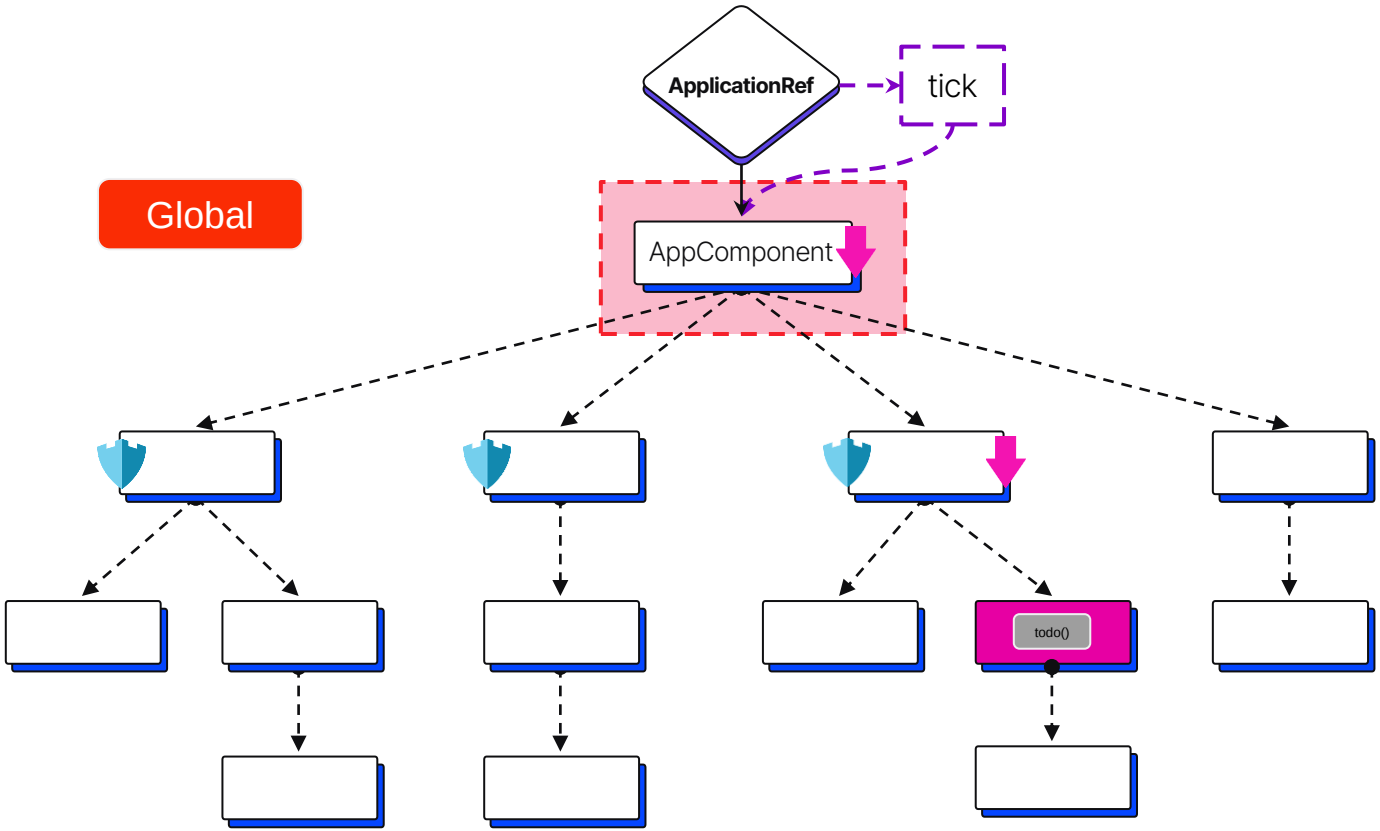
Rules


Global Mode
normal
refreshed
dirty
refreshed
OnPush
skipped
OnPush(dirty)
refreshed
RefreshView
refreshed
OnPush(dirty) + HasChildViews ToRefresh
refreshed
OnPush + HasChildViews ToRefresh
skipped
Targeted Mode
normal
skipped
dirty
skipped
OnPush
skipped
OnPush(dirty)
skipped
RefreshView
refreshed
OnPush(dirty) + HasChildViews ToRefresh
skipped
OnPush + HasChildViews ToRefresh
skipped

Tick!


zone.js / NgZone
onMicrotaskEmpty
ApplicationRef
tick
AppComponent
todo()
click, setTimeout,
other events
Legend
Component
zone.js
skipped
refreshed
dirty
OnPush
OnPush(dirty)
RefreshView
HasChildViews
ToRefresh

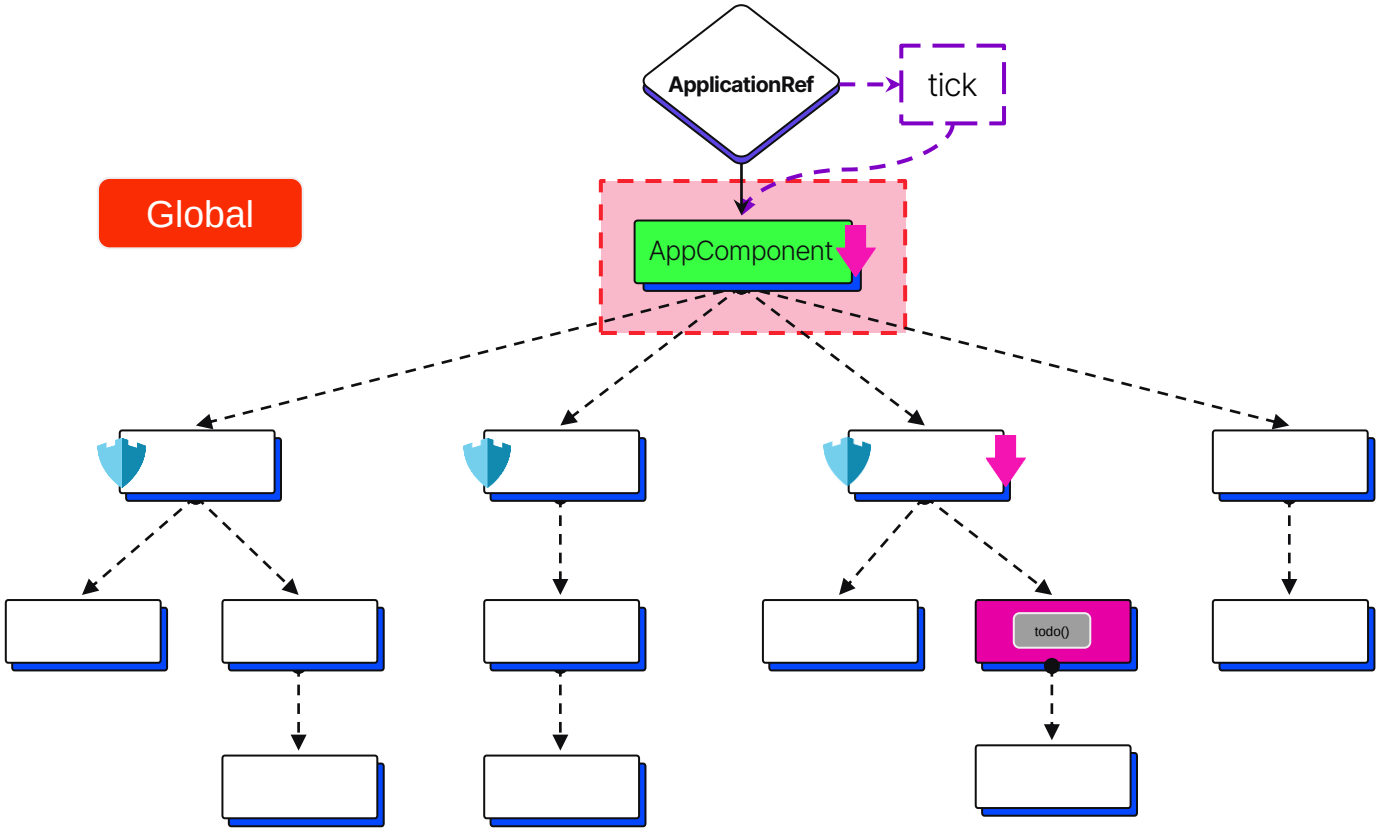
Global + Normal



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

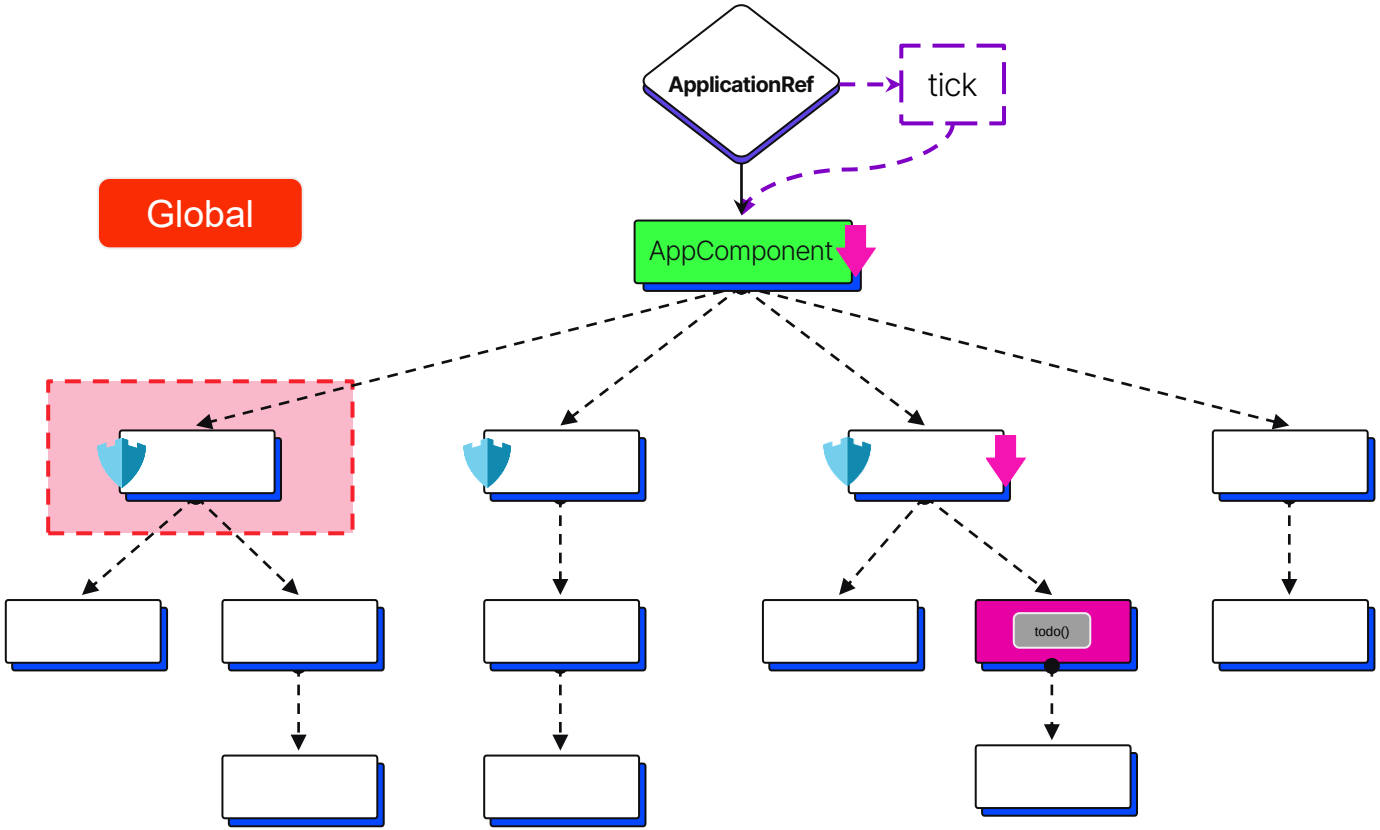
Global + Normal -> **Refresh**



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

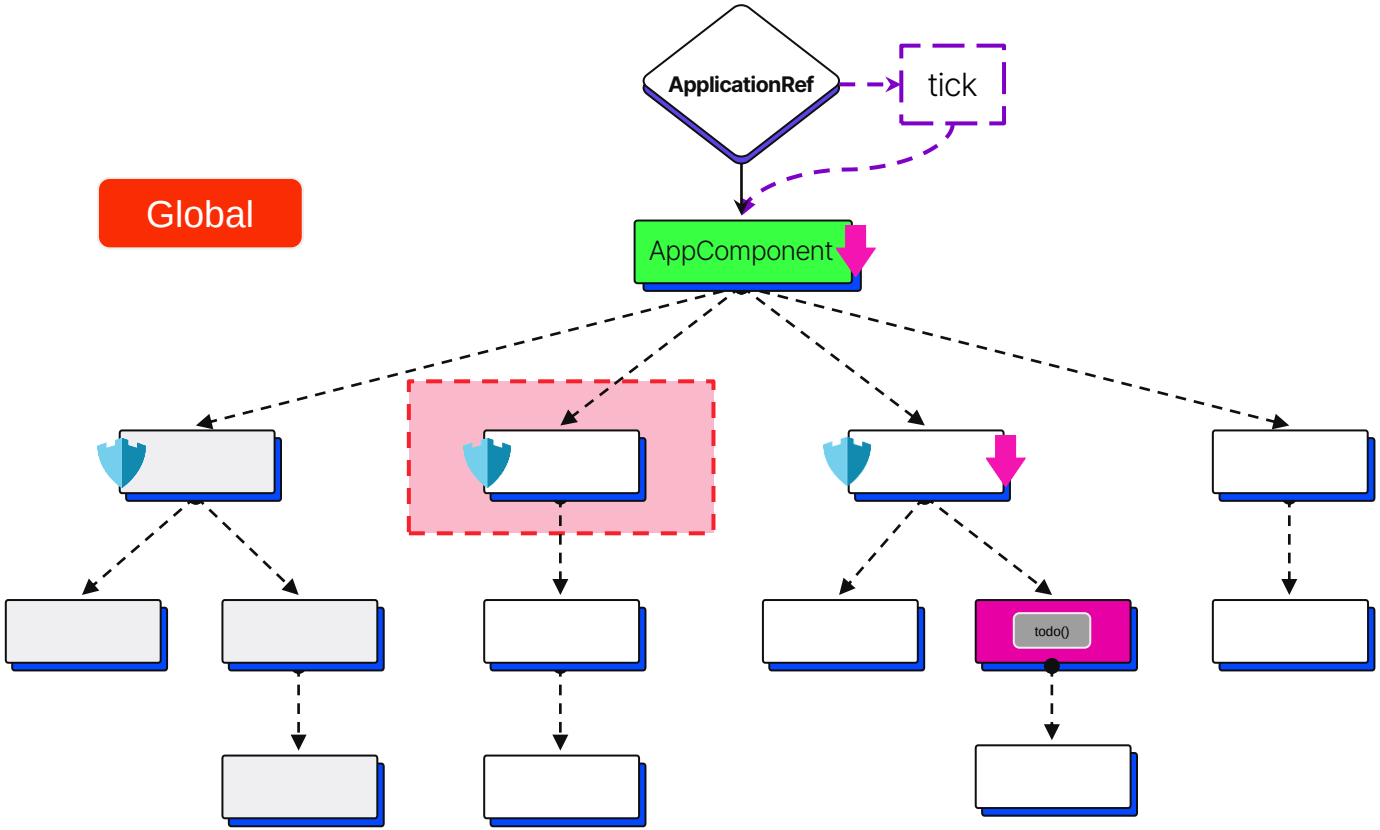
Global + OnPush(non-dirty) -> Skip



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

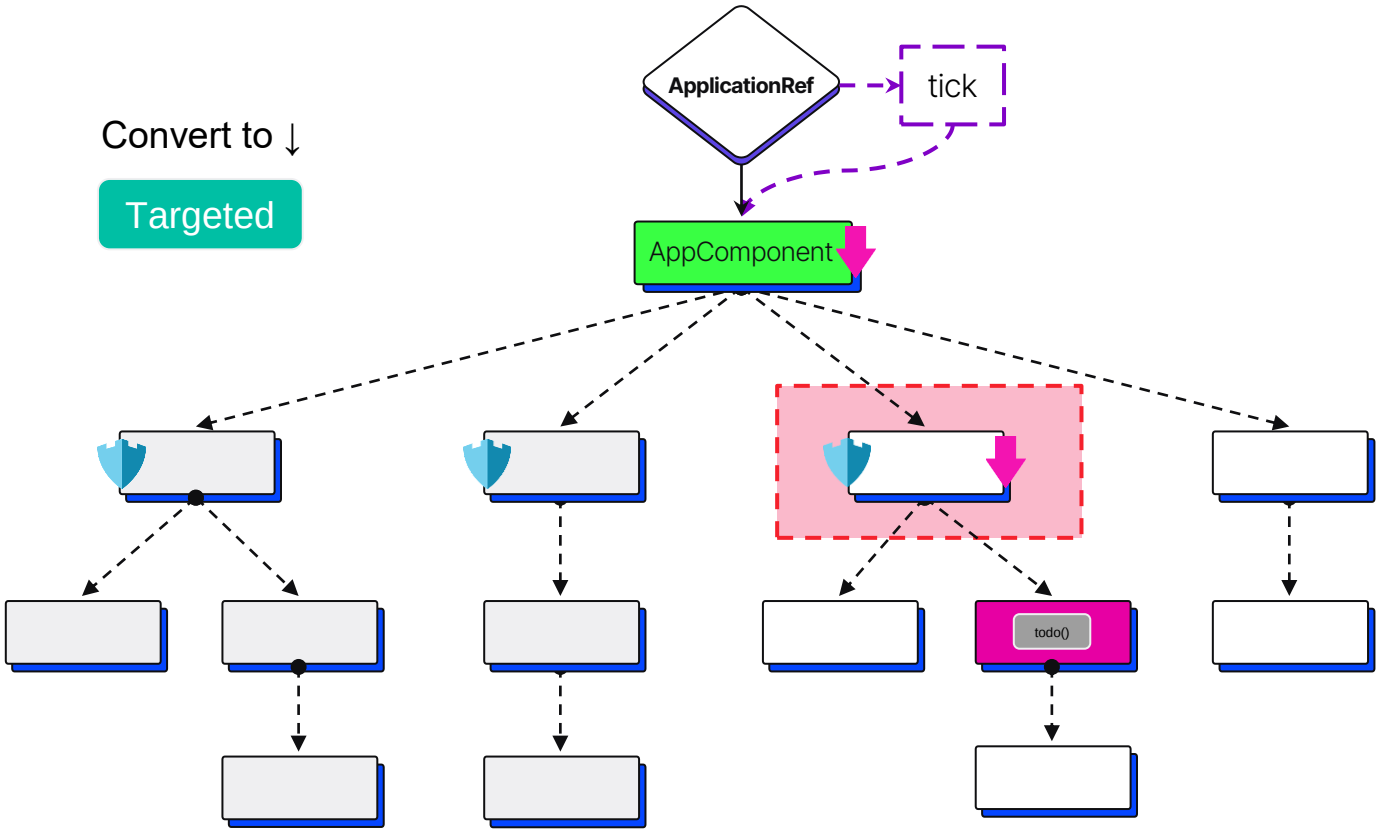
Global + OnPush(non-dirty) -> Skip



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

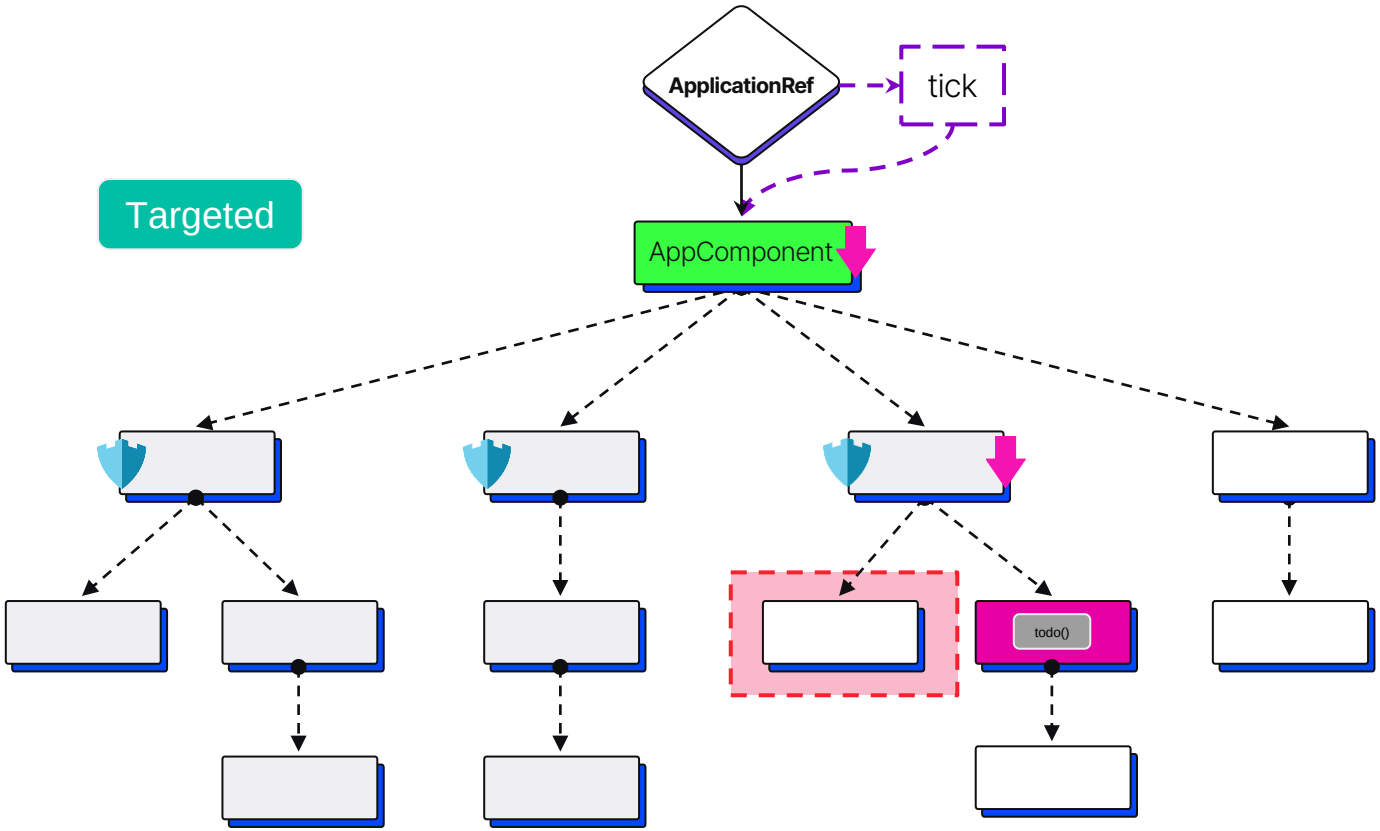
OnPush + HasChildViewsToRefresh -> Skip



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

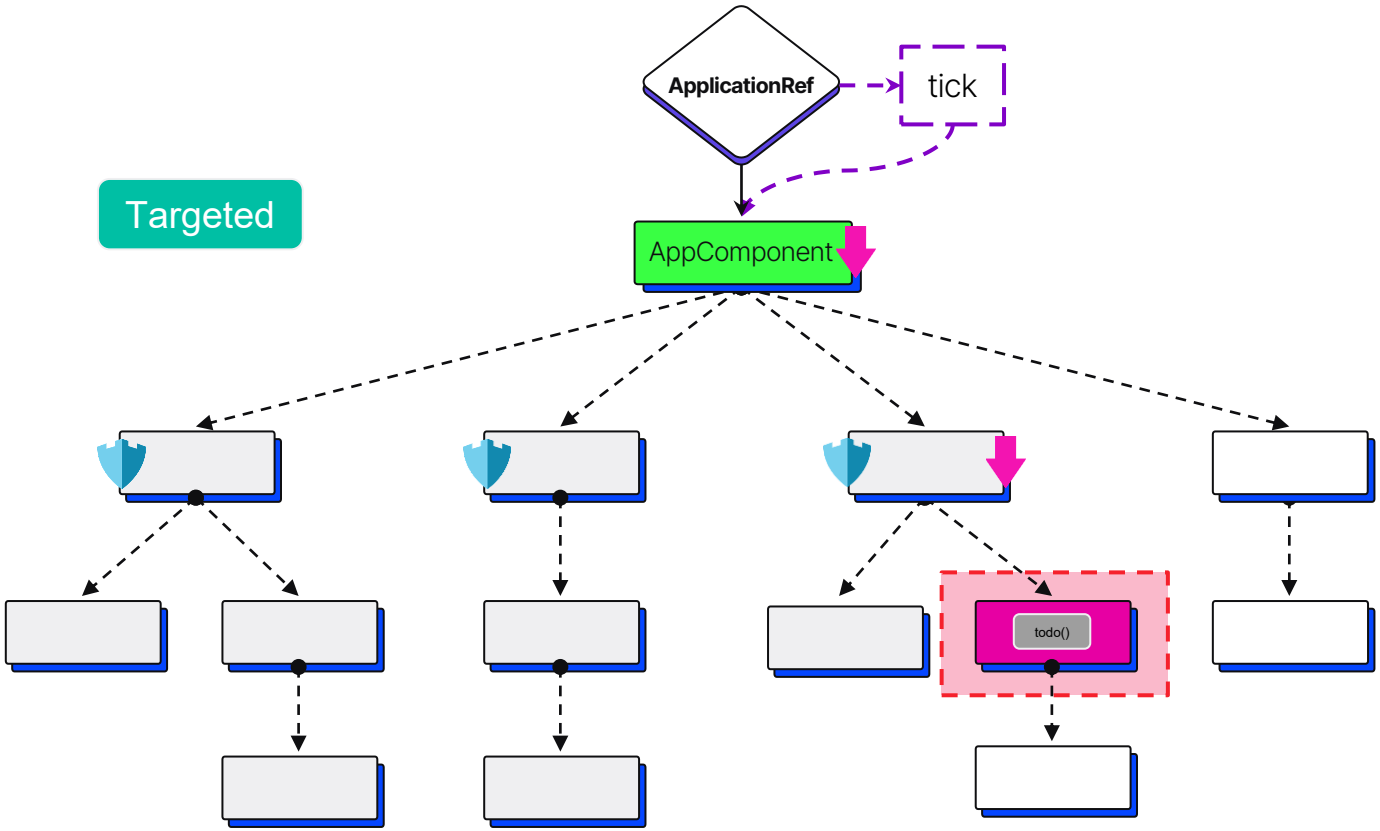
Targeted + Normal -> Skip



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

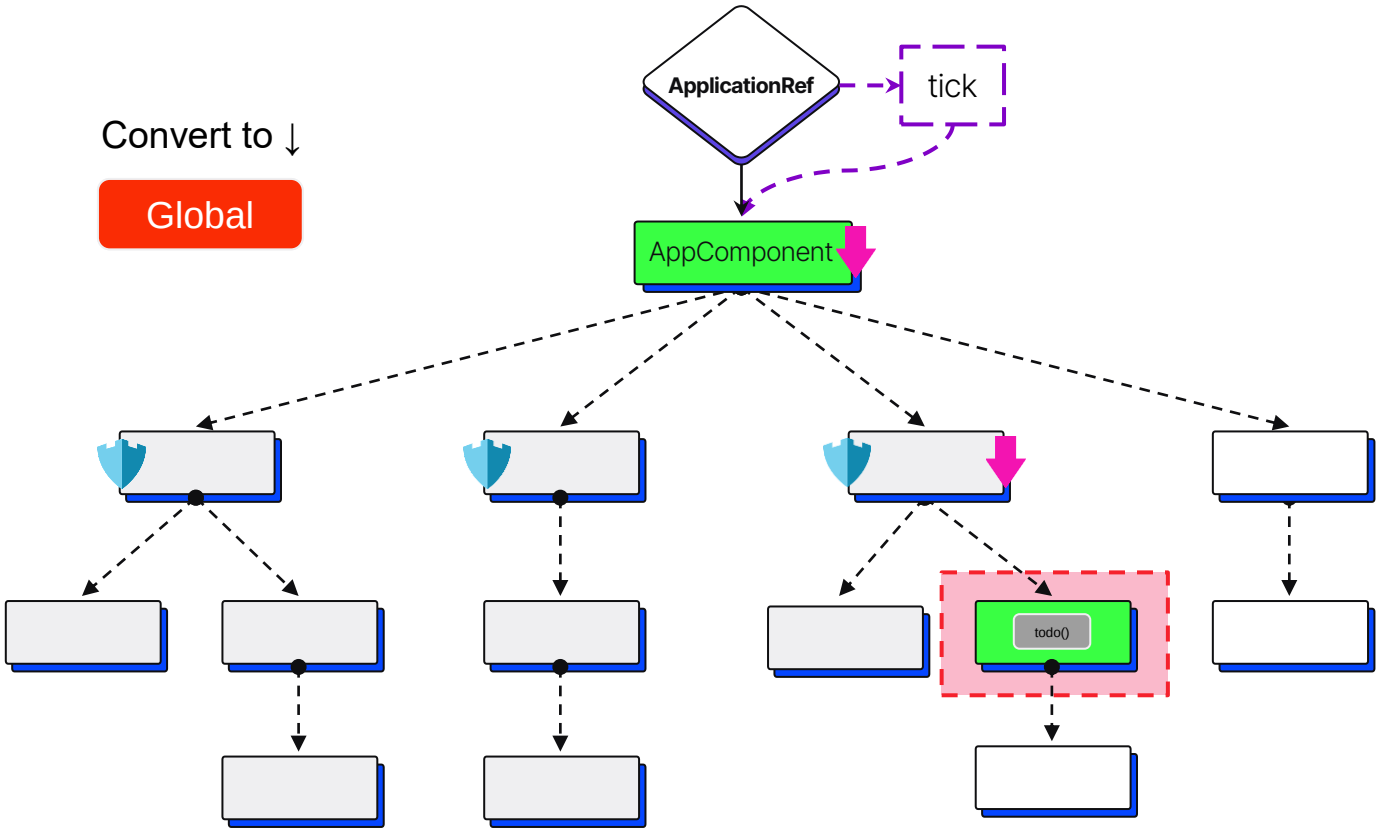
Targeted + RefreshView -> Refresh



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

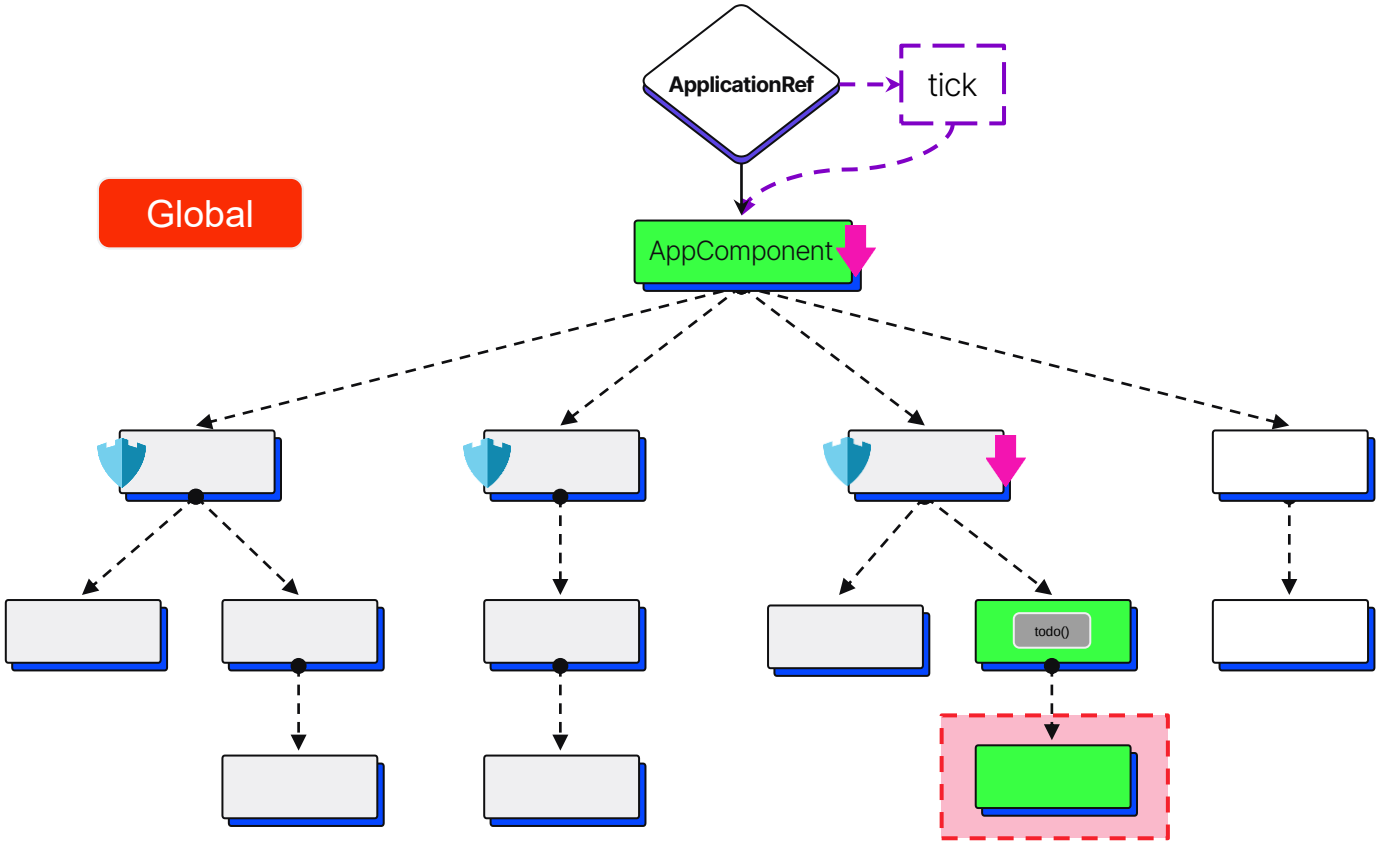
Targeted + RefreshView -> Refresh



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

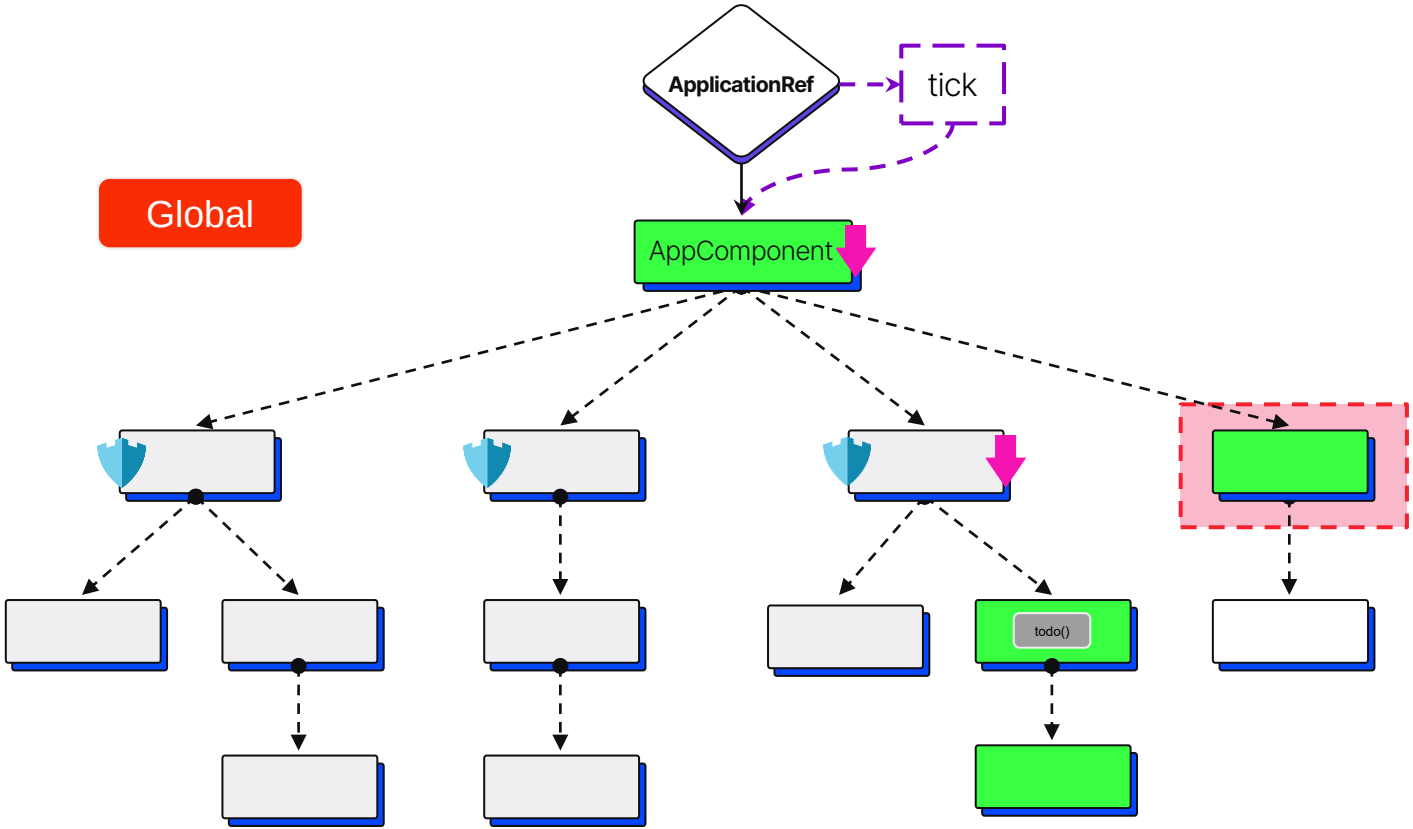
Global + Normal -> Refresh



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

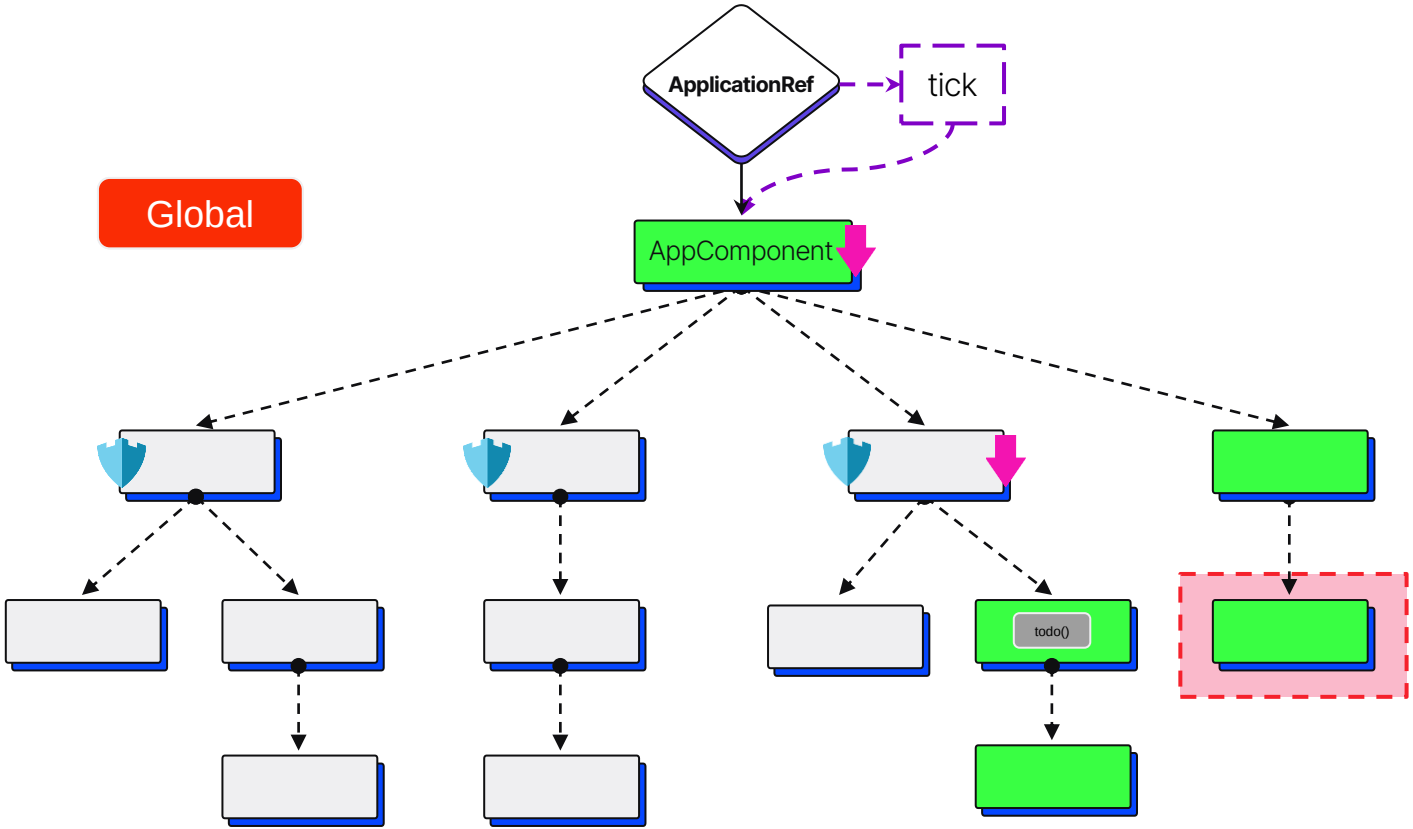
Global + Normal -> Refresh



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

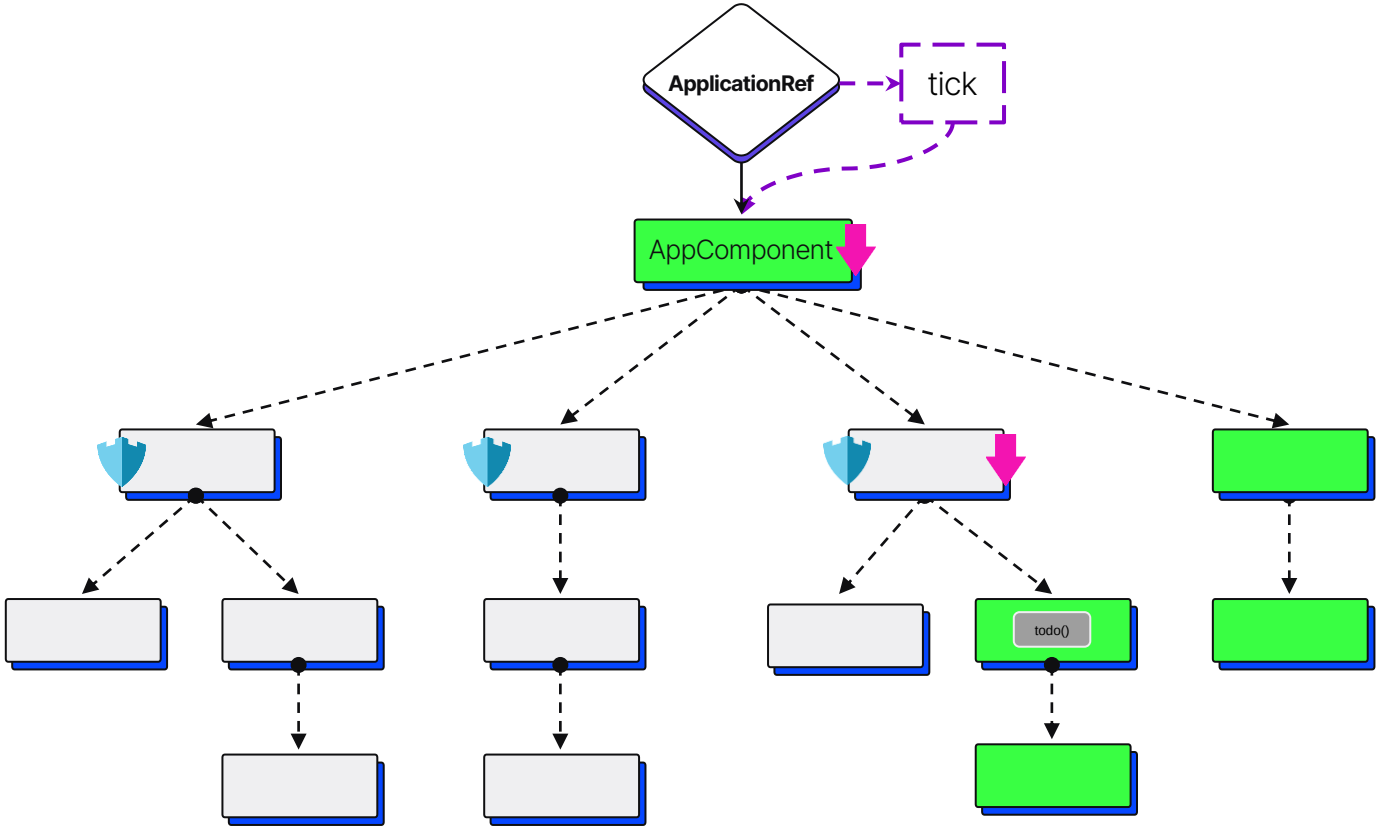
Global + Normal -> Refresh



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

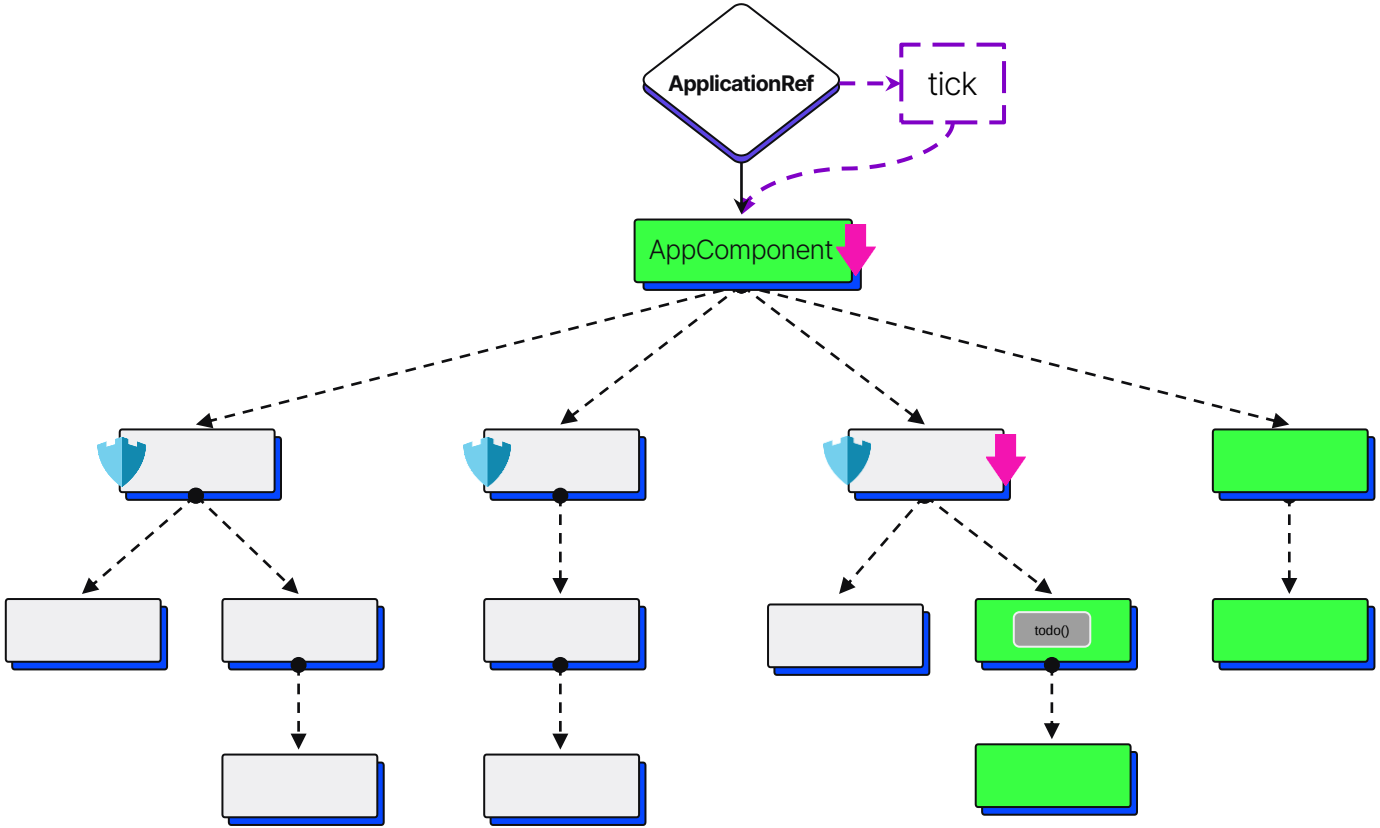
Way better!



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

Keep in mind: OnPush (not dirty) enables **TargetedMode**



Legend

- Component
- zone.js
- skipped
- refreshed
- dirty
- OnPush
- OnPush(dirty)
- RefreshView
- HasChildViews ToRefresh

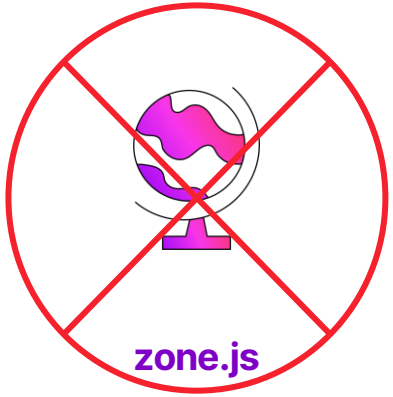
How does Angular know when something changes?

zone.js

markForCheck

signals

Let’s go zoneless!

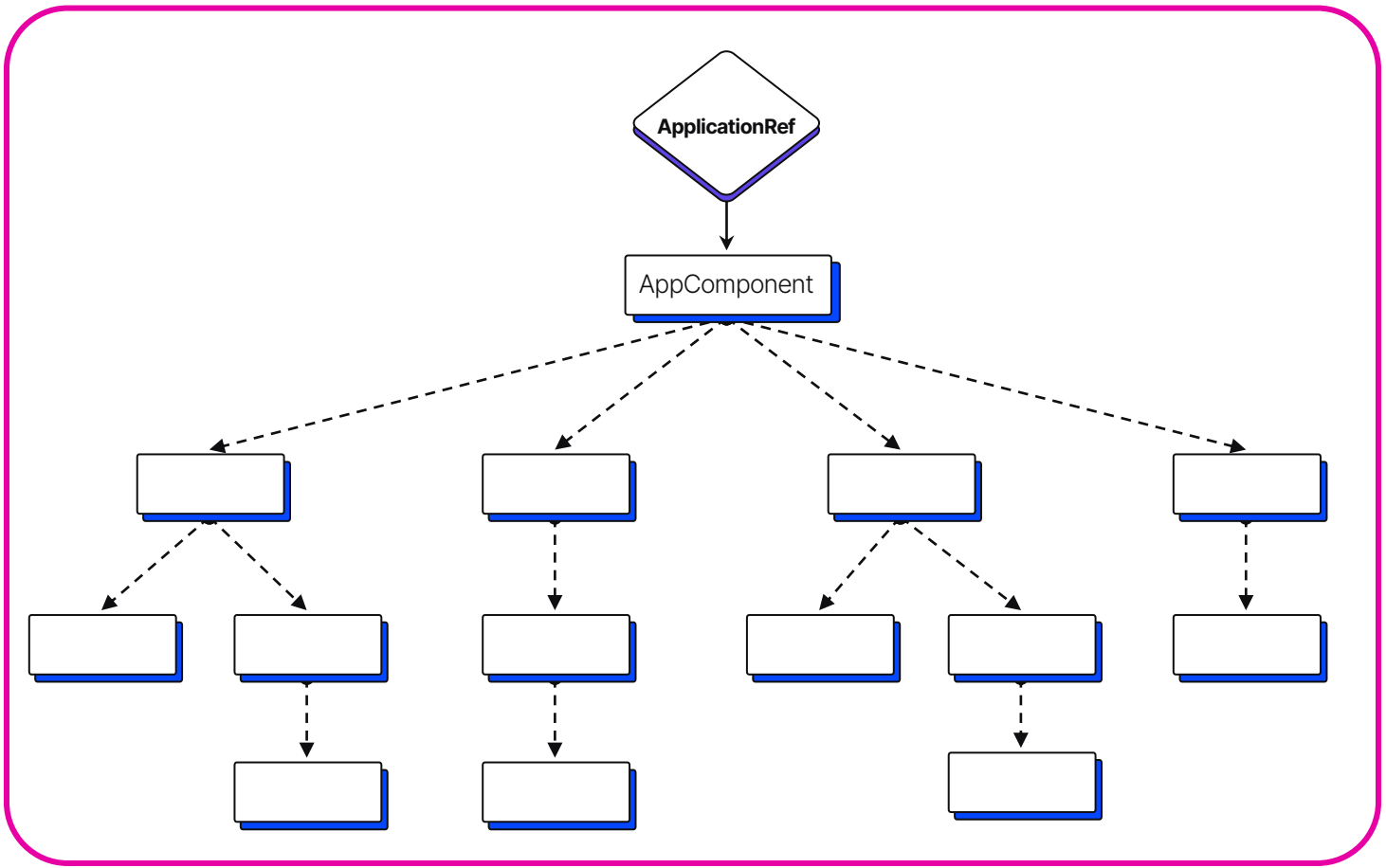


zone.js

markForCheck

signals

Angular with zone.js

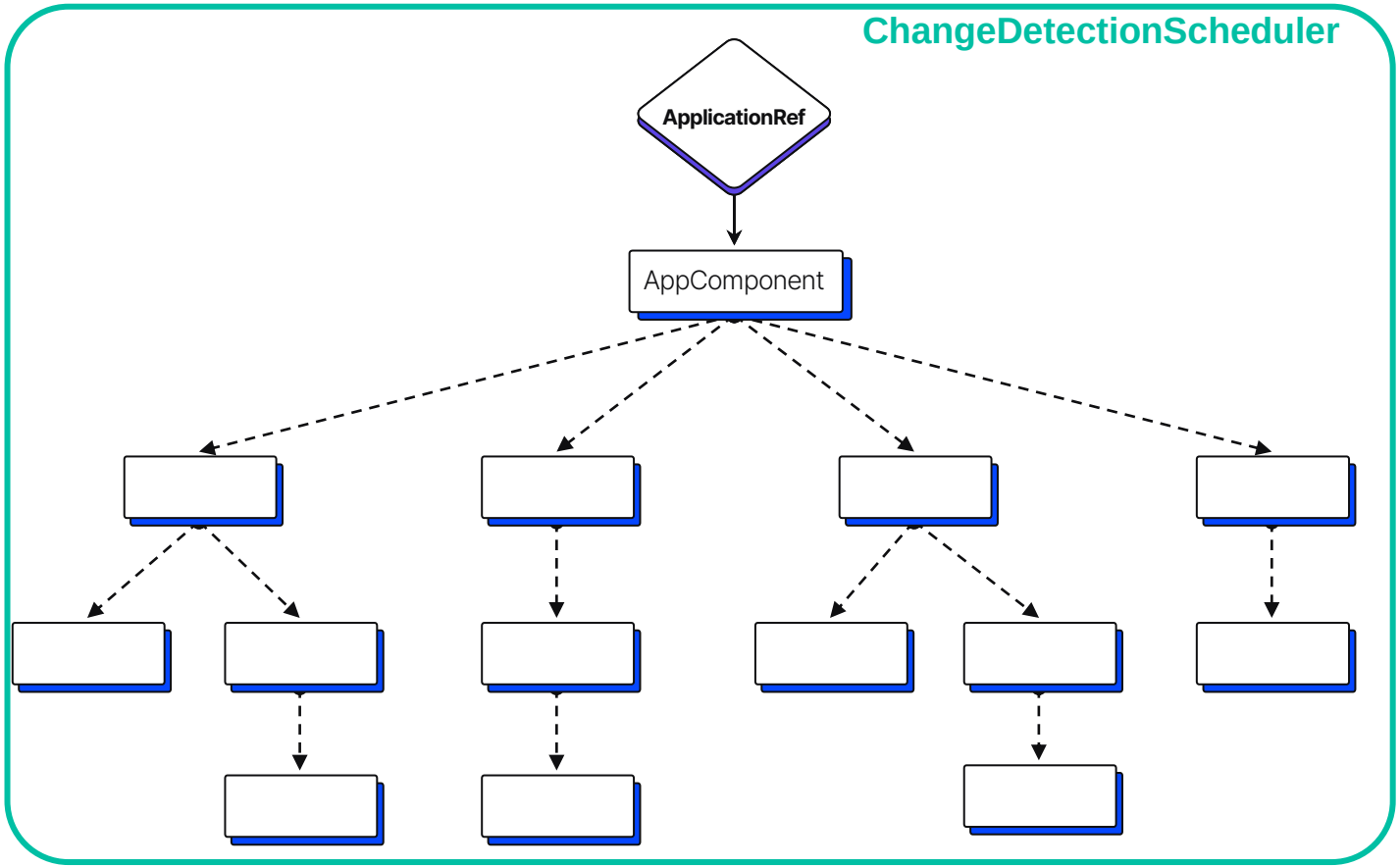


Legend

Component

zone.js

Angular with ChangeDetectionScheduler

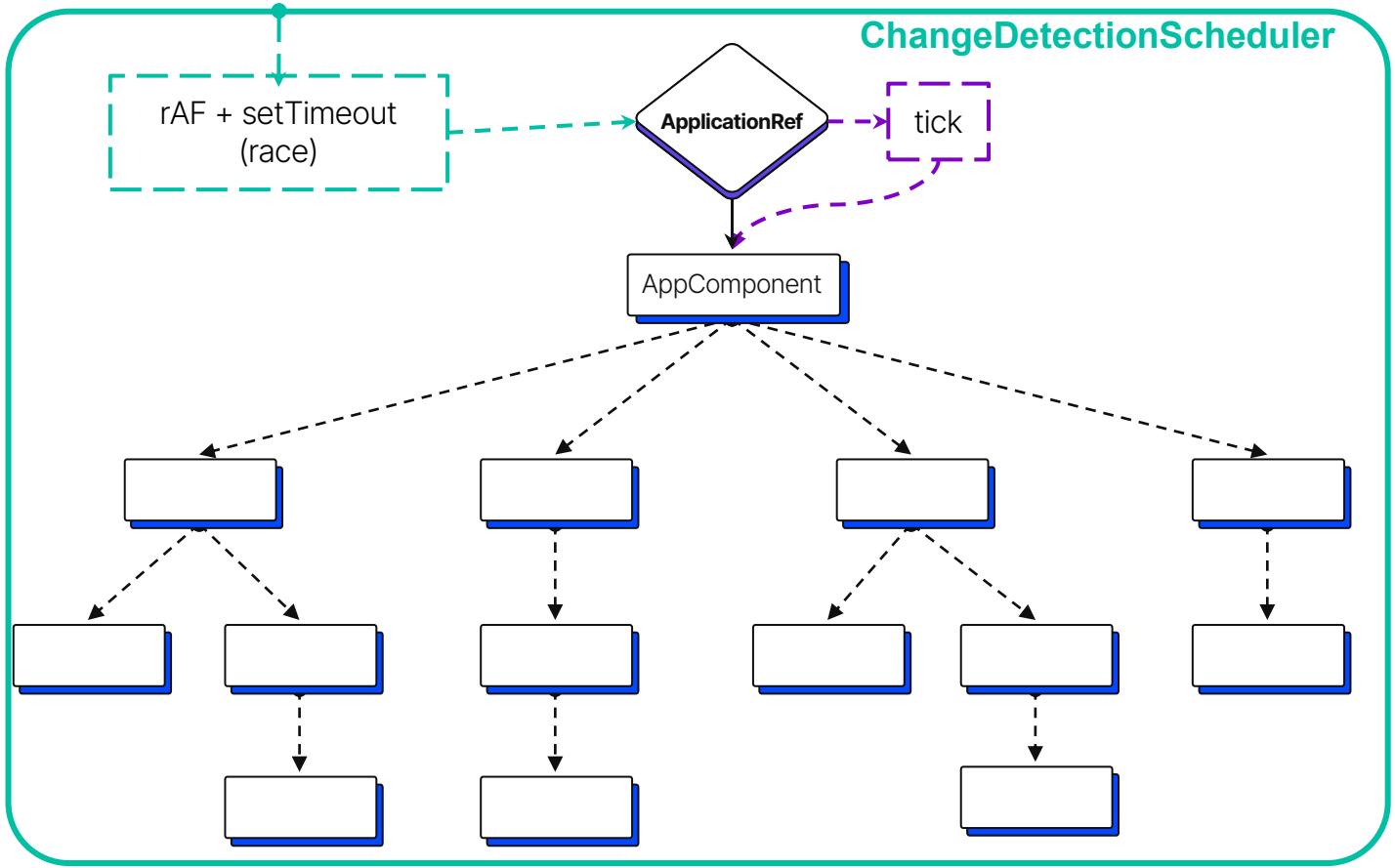


Legend

Component

ChangeDetection Scheduler

Angular with ChangeDetectionScheduler



Legend

Component

ChangeDetection
Scheduler

zone.js vs ChangeDetectionScheduler

zone.js

Browser asynchronous events

(addEventListener, onProp, setTimeout, setInterval, alert, confirm, etc.)

ChangeDetectionScheduler

Angular Updates/Changes

(Signal Update, MarkForCheck, Listener, SetInput, View Attach / Detach, AsyncAnimationsLoaded, RenderHook, DeferBlockStateUpdate)

mark_view_dirty.ts

```ts
/**
 * Marks current view and all ancestors dirty.
 */
export function markViewDirty(lView: LView, source: NotificationSource): LView | null {

  lView[ENVIRONMENT].changeDetectionScheduler?.notify();

  while (lView) {
    lView[FLAGS] |= LViewFlags.Dirty;
    const parent = getLViewParent(lView);
    if (isRootView(lView) && !parent) {
      return lView;
    }
    lView = parent!;
  }
  return null;
}
```

Editor Preview Both

app.config.ts

```ts
export const appConfig: ApplicationConfig = {
  providers: [
    // ...
    provideExperimentalZonelessChangeDetection(),
  ]
};
```

Editor Preview Both

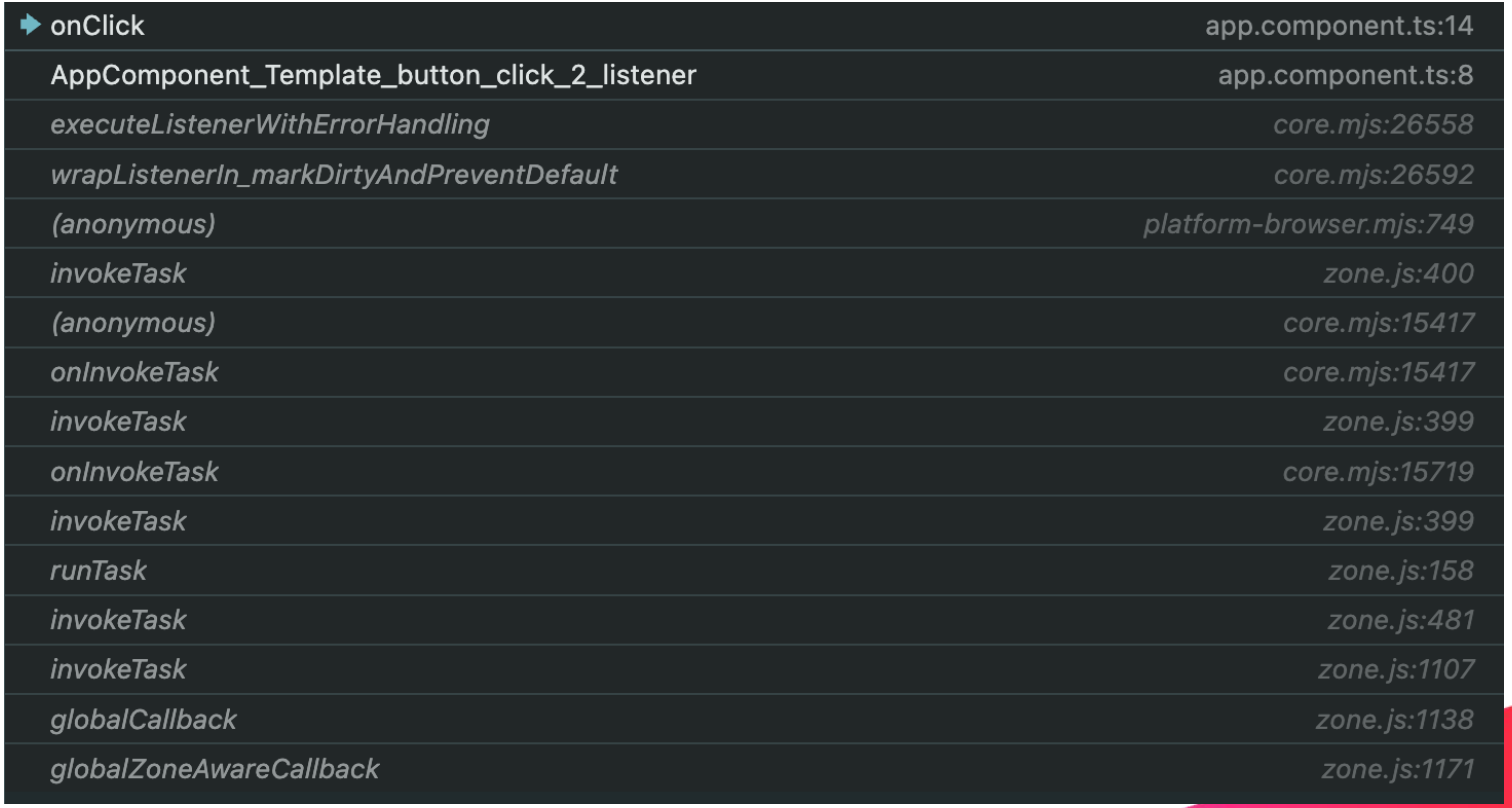
angular.json

```
{
    "build": {
      "builder": "@angular-devkit/build-angular:application",
        "options": {
          "polyfills": [
            "zone.js"
          ]
        }
    }
}
```

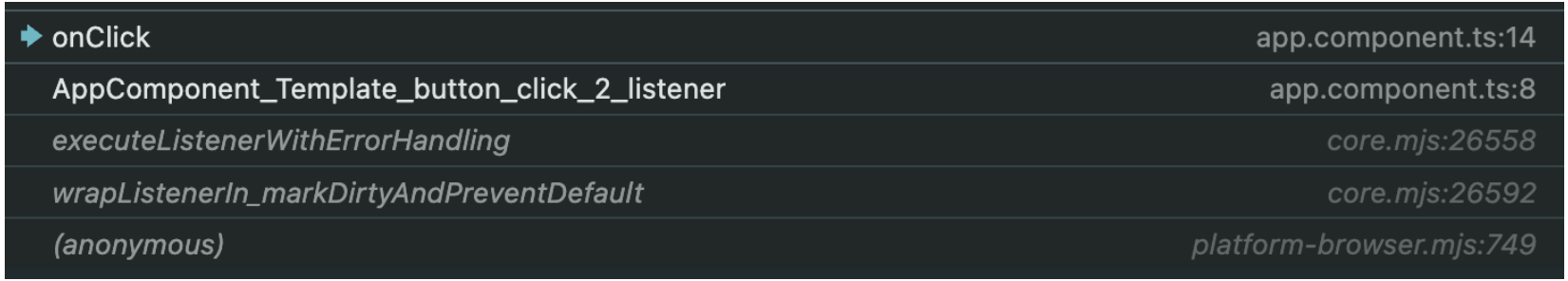
-15kb 🎉

Editor Preview Both

With zone.js

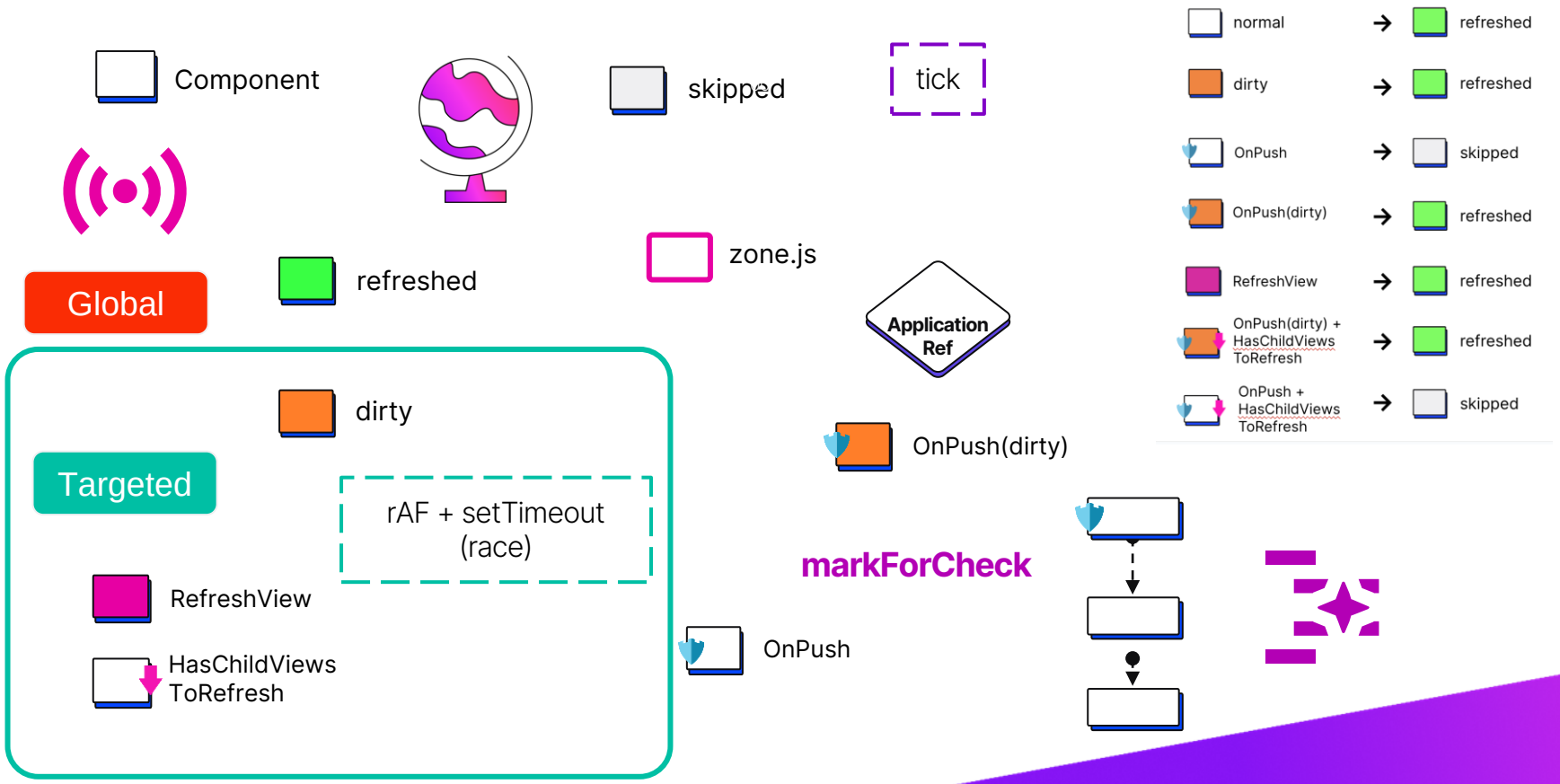

onClick app.component.ts:14
AppComponent_Template_button_click_2_listener app.component.ts:8
executeListenerWithErrorHandling core.mjs:26558
wrapListenerIn_markDirtyAndPreventDefault core.mjs:26592
(anonymous) platform-browser.mjs:749
invokeTask zone.js:400
(anonymous) core.mjs:15417
onInvokeTask core.mjs:15417
invokeTask zone.js:399
onInvokeTask core.mjs:15719
invokeTask zone.js:399
runTask zone.js:158
invokeTask zone.js:481
invokeTask zone.js:1107
globalCallback zone.js:1138
globalZoneAwareCallback zone.js:1171

Without zone.js



In v18 Angular works **zoneless** by default!

You don’t have to care about all this!



You don’t have to care about all this!

It’s Angular’s job to do that!

Just use **signals** (or **markForCheck**) with **OnPush** (or not) and everything will be fine!

Libraries that help you with **signals**!

NgRx

RxAngular

ngxtension

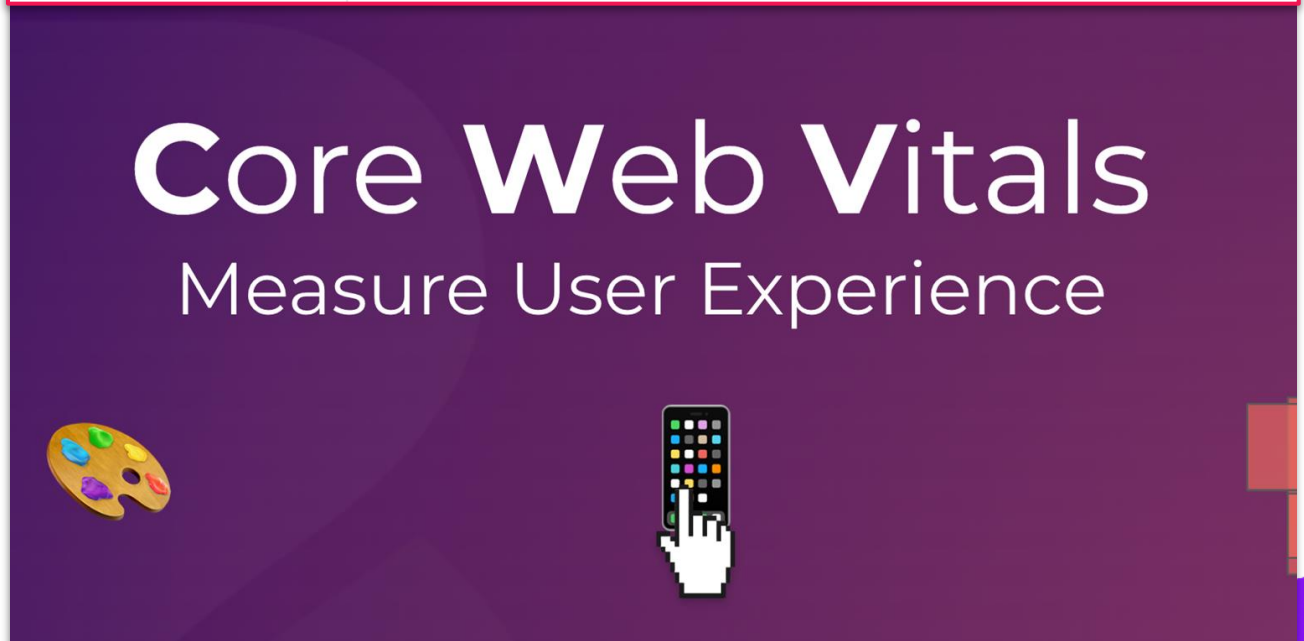
You want better performance??

11:40 AM - 11:55 AM

Angular Performance and Core Web Vitals in 2024

Michael Hladky

Chief Executive Officer - Push-Based.io



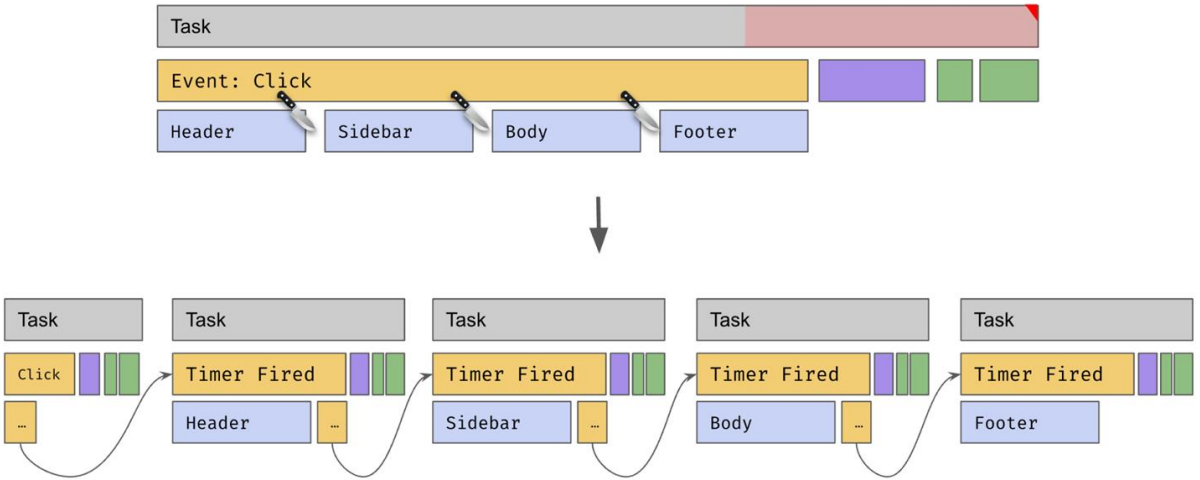
You want better performance??

02:55 PM - 03:20 PM

Concurrent Mode in Angular

Julian Jandl

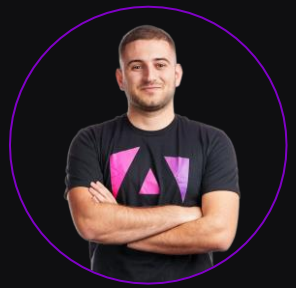
Lead Performance Engineer - Push-Based.io



Slides

SCAN ME

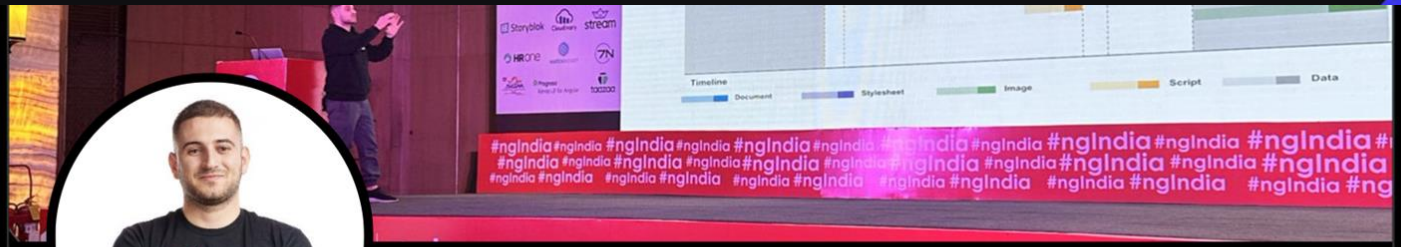
A ***change detection***, ***zone.js*** and ***signals*** story 📖



Enea Jahollari / @Enea_Jahollari
GDE for Angular / Software Engineer
Push-based.io

PUSH BASED

Experts 2024



Edit profile

Enea Jahollari A

@Enea_Jahollari

Angular Hype Developer | GDE A. ⚡ Signals, updates, PRs & more. Software Engineer push-based.io. Created ngx-isr & ngx-libs, co-created ngxtension 🔨

View more

Science & Technology · Albania · eneajaho.me · Joined April 2015

1,000 Following **9,647** Followers