I love and want to improve reactive Angular

```
<h3>
  Found {{ (movies$ | async)?.total }} Results
</h3>
<app-movies [movies]="movies$ | async"/>
```

```
class MoviesPageComponent {
  movies$: Observable<MovieModel[]>;

  constructor(http: HttpClient) {
    this.movies$ = http
      .get("/api/v1/movies")
      .pipe(map(res => res.data));
  }
}
```

```
<h3>
  Found {{ (movies$ | async)?.total }} Results
</h3>
<app-movies [movies]=“movies$ | async"/>
```

```
<h3>
  Found {{ (movies$ | async)?.total }} Results
</h3>
<app-movies [movies]="movies$ | async"/>
```

```
<ng-container *ngIf="movies$ | async as movies">
  <h3>
    Found {{ movies.length }} Results
  </h3>
  <app-movies [movies]="movies"/>
</ng-container>
```

```
<ng-container *ngIf="movies$ | async as movies">
  <h3>
    Found {{ movies.length }} Results
  </h3>
  <app-movies [movies]="movies"/>
</ng-container>
```

```
<ng-container *ngIf="movies$ | async as movies">
  <h3>
    Found {{ movies.length }} Results
  </h3>
  <app-movies [movies]="movies"/>
</ng-container>
```

```
<ng-container *ngIf="movies$ | async as movies">
  <h3>
    Found {{ movies.length }} Results
  </h3>
  <app-movies [movies]="movies"/>
</ng-container>
```

```
<ng-container *ngIf="user$ | async as user">
  <ng-container *ngIf="movies$ | async as movies">
    <h3>
      Found {{ movies.length }} Results
    </h3>
    <app-movies [movies]="movies"/>
  </ng-container>
</ng-container>
```

```
<ng-container *ngIf="covers$ | async as covers">
  <ng-container *ngIf="user$ | async as user">
    <ng-container *ngIf="movies$ | async as movies">
      <h3>
        Found {{ movies.length }} Results
      </h3>
      <app-movies [movies]="movies"/>
    </ng-container>
  </ng-container>
</ng-container>
```

```
<ng-container *ngIf="movies$ | async as movies">
  <h3>
    Found {{ movies.length }} Results
  </h3>
  <app-movies [movies]="movies"/>
</ng-container>
```

```
class MoviesPageComponent {
  movies$: Observable<MovieModel[]>;

  constructor(http: HttpClient) {
    this.movies$ = http
      .get("/api/v1/movies")
      .pipe(map(res => res.data));
  }
}
```

```
class MoviesPageComponent {
  movies: MovieModel[];
  subscription: Subscription;

  constructor(http: HttpClient) {
    this.subscription = http
      .get("/api/v1/movies")
      .subscribe(res => (this.movies = res.data));
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

```
class MoviesPageComponent {
  movies: MovieModel[];
  subscription: Subscription;

  constructor(http: HttpClient) {
    this.subscription = http
      .get("/api/v1/movies")
      .subscribe(res => (this.movies = res.data));
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

```
class MoviesPageComponent {
  movies: MovieModel[];
  subscription: Subscription;

  constructor(http: HttpClient) {
    this.subscription = http
      .get("/api/v1/movies")
      .subscribe(res => (this.movies = res.data));
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

```
class MoviesPageComponent {
  movies: MovieModel[];
  subscription: Subscription;

  constructor(http: HttpClient) {
    this.subscription = http
      .get("/api/v1/movies")
      .subscribe(res => (this.movies = res.data));
  }

  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

```
class MoviesPageComponent {
  movies: MovieModel[];

  constructor(http: HttpClient) {
    http
      .get("/api/v1/movies")
      .subscribe(res => (this.movies = res.data));
  }
}
```

```
webpackJsonp.../../../../zone.js/dist/zone.js.ZoneDelegate.scheduleTask
onScheduleTask
webpackJsonp.../../../../zone.js/dist/zone.js.ZoneDelegate.scheduleTask
webpackJsonp.../../../../zone.js/dist/zone.js.Zone.scheduleTask
webpackJsonp.../../../../zone.js/dist/zone.js.Zone.scheduleMacroTask
scheduleMacroTaskWithCurrentZone
(anonymous)
proto.(anonymous function)
(anonymous)
webpackJsonp.../../../../rxjs/_esm5/Observable.js.Observable._trySubscribe
webpackJsonp.../../../../rxjs/_esm5/Observable.js.Observable.subscribe
webpackJsonp.../../../../rxjs/_esm5/operators/timeout.js.TimeoutOperator.call
webpackJsonp.../../../../rxjs/_esm5/Observable.js.Observable.subscribe
webpackJsonp.../../../../rxjs/_esm5/operators/map.js.MapOperator.call
webpackJsonp.../../../../rxjs/_esm5/Observable.js.Observable.subscribe
webpackJsonp.../../../../rxjs/_esm5/operators/catchError.js.CatchOperator.call
webpackJsonp.../../../../rxjs/_esm5/Observable.js.Observable.subscribe
webpackJsonp.../../../../../src/components/signUp.ts.SignUpComponent.onSubmit
(anonymous)
handleEvent
callWithDebugContext
debugHandleEvent
dispatchEvent
(anonymous)
(anonymous)
webpackJsonp.../../../../zone.js/dist/zone.js.ZoneDelegate.invokeTask
onInvokeTask
webpackJsonp.../../../../zone.js/dist/zone.js.ZoneDelegate.invokeTask
```

Can we provide a better experience?

I want you to help build this

Mike Ryan

@MikeRyanDev

Software Architect at Synapse

Google Developer Expert

NgRx Co-Creator

Goals for improved reactivity

Goals for improved reactivity

- **Use values directly in the template without the async pipe**

Goals for improved reactivity

- Use values directly in the template without the async pipe
- Make it easier to not leak memory accidentally

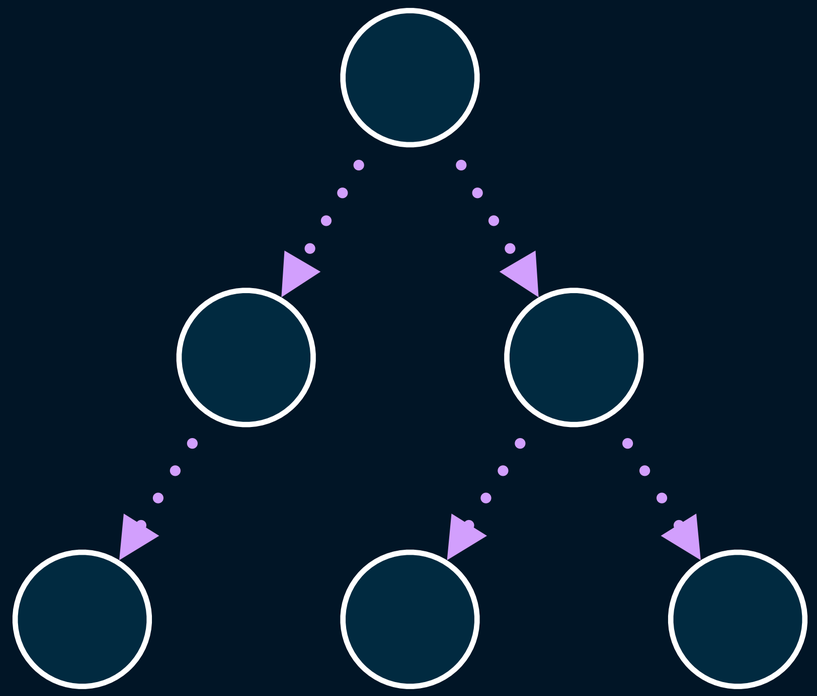
Goals for improved reactivity

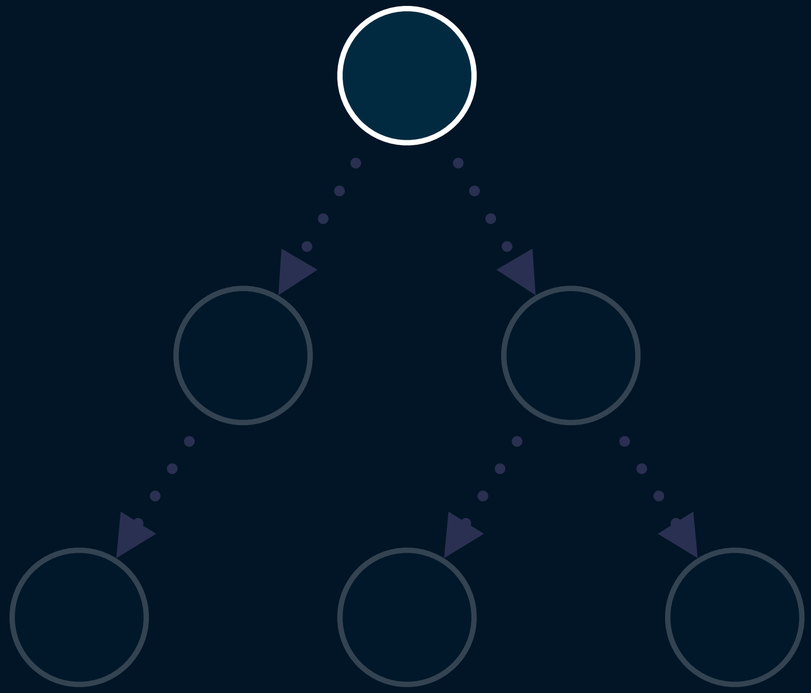
Use values directly in the template without the async pipe

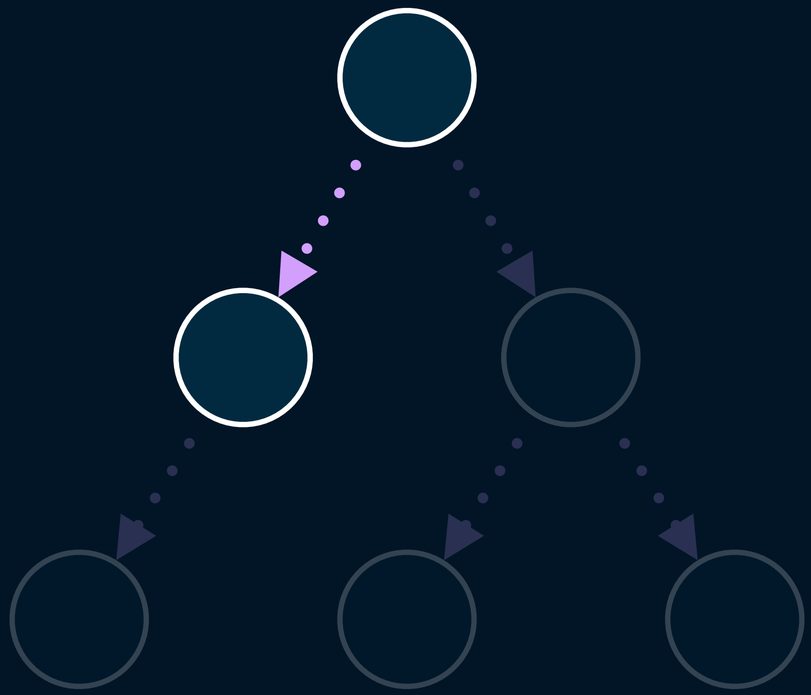
Make it easier to not leak memory accidentally

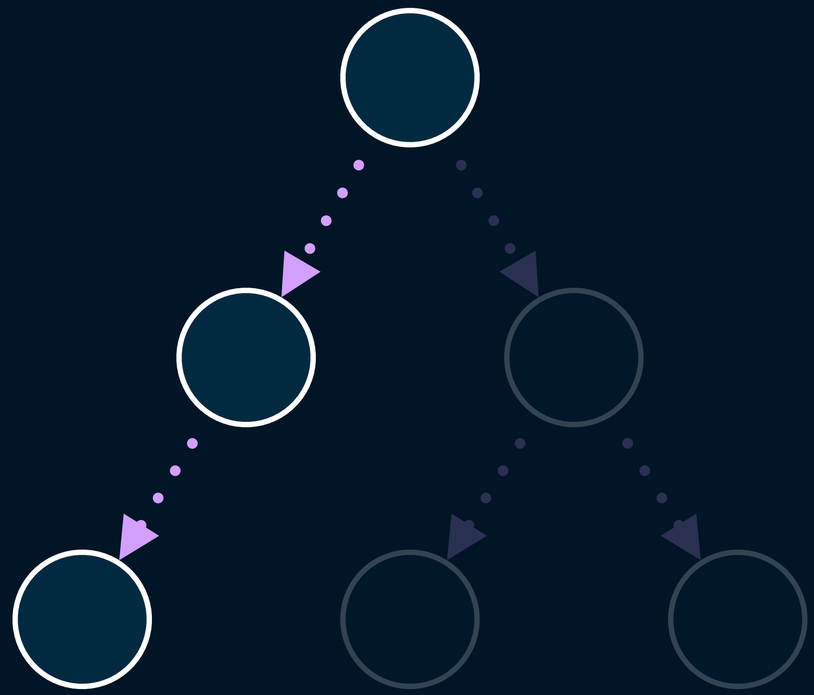
- Make it possible for apps to be built without using Zone.js

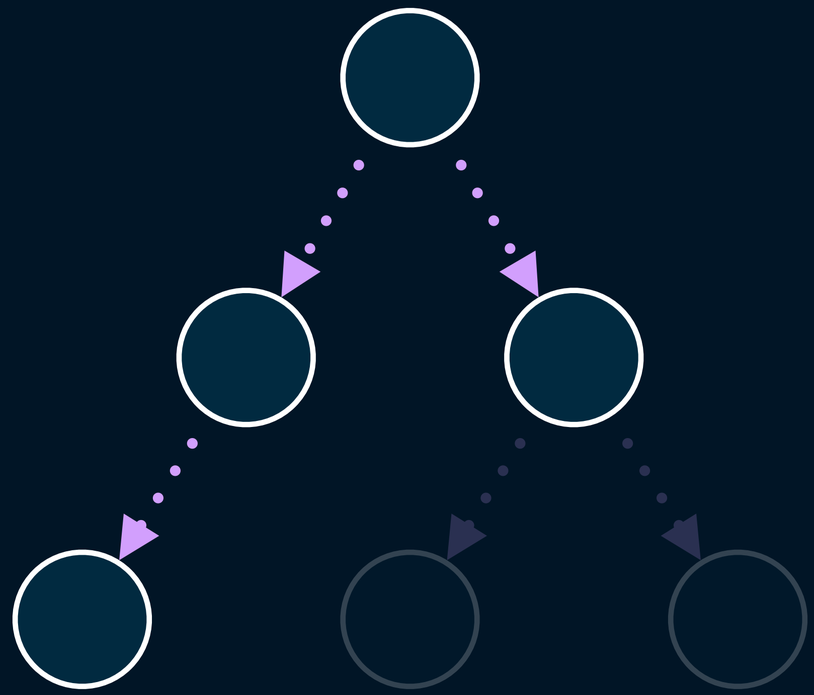
API Design

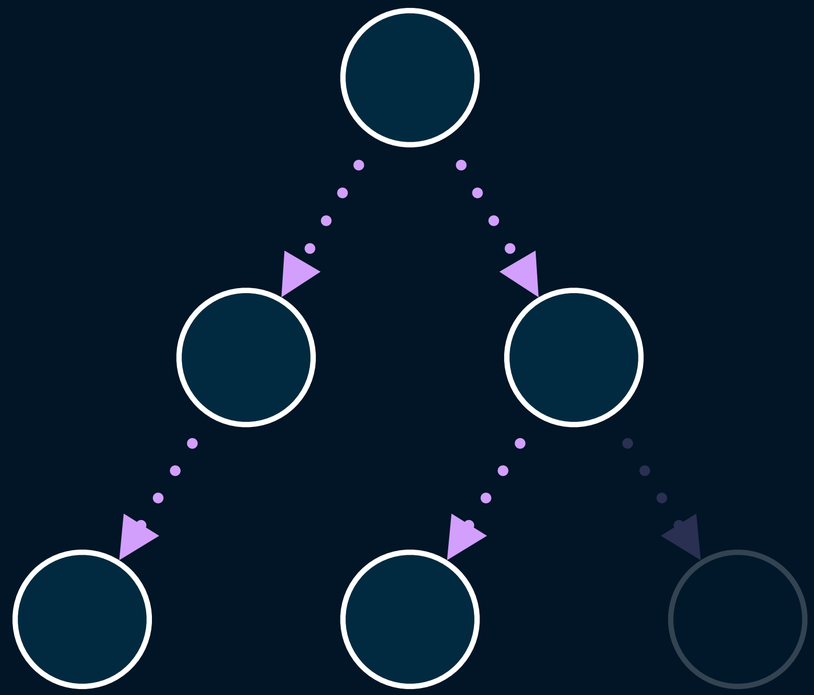


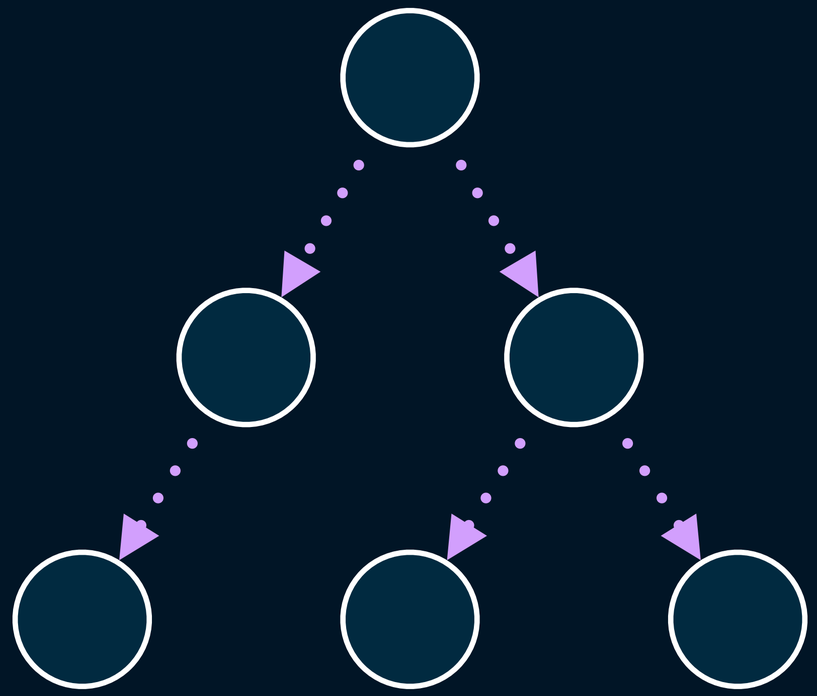












Zones and change detection

Zones and change detection

- **Zones tell Angular when to run change detection**

Zones and change detection

Zones tell Angular when to run change detection

- They can potentially run change detection too much

Zones and change detection

Zones tell Angular when to run change detection

They can potentially run change detection too much

- They can also accidentally skip change detection

Look around and learn from others

```tsx
export function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count - 1)}>-</button>
      <button onClick={() => setCount(count + 1)}>+</button>
    </div>
  );
}
```

```
export function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count - 1)}>-</button>
      <button onClick={() => setCount(count + 1)}>+</button>
    </div>
  );
}
```

```
export function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count - 1)}>-</button>
      <button onClick={() => setCount(count + 1)}>+</button>
    </div>
  );
}
```

React's Change Detection

React's Change Detection

- **Changes to state causes change detection**

React's Change Detection

Changes to state causes change detection

- **Developer controls state changes**

React's Change Detection

Changes to state causes change detection

Developer controls state changes

- Easier to keep the UI consistent and up to date

Angular doesn't have component state

```
@Injectable({ providedIn: "root" })
export class CountService {
  count$ = new BehaviorSubject(0);
}
```

```
class CountComponent {
  count$: Observable<number>;

  constructor(store: Store<State>) {
    this.count$ = store.select(selectCount);
  }
}
```

We use observables track state in Angular

```
class MoviesPageComponent {
  state: { movies: MovieModel[] };

  constructor(http: HttpClient) {
    this.state = connect({
      movies: http
        .get("/api/v1/movies")
        .pipe(map(res => res.data))
    });
  }
}
```

```
class MoviesPageComponent {
  state: { movies: MovieModel[] };

  constructor(http: HttpClient) {
    this.state = connect({
      movies: http
        .get("/api/v1/movies")
        .pipe(map(res => res.data))
    });
  }
}
```

```
class MoviesPageComponent {
  state: { movies: MovieModel[] };

  constructor(http: HttpClient) {
    this.state = connect({
      movies: http
        .get("/api/v1/movies")
        .pipe(map(res => res.data))
    });
  }
}
```

```
class MoviesPageComponent {
  state: { movies: MovieModel[] };

  constructor(http: HttpClient) {
    this.state = connect({
      movies: http
        .get("/api/v1/movies")
        .pipe(map(res => res.data))
    });
  }
}
```

```
<h3>
  Found {{ state.movies.length }} Results
</h3>
<app-movies [movies]="state.movies"/>
```

We need a demo app (with Ivy)

Why do we need Ivy?

Why do we need Ivy?

- **Ivy enables us to build higher order components**

Why do we need Ivy?

Ivy enables us to build higher order components

- **New change detection APIs will let us work around Zone.js**

Why do we need Ivy?

Ivy enables us to build higher order components

New change detection APIs will let us work around Zone.js

- Give us better type safety in our templates

Count

```
@Component({
  selector: "app-root",
  template: `
    <div>{{ count$ | async }}</div>
    <button (click)="values$.next(-1)">-</button>
    <button (click)="values$.next(+1)">+</button>
  `
})
export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <div>{{ count$ | async }}</div>
    <button (click)="values$.next(-1)">-</button>
    <button (click)="values$.next(+1)">+</button>
  `
})
export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <div>{{ count$ | async }}</div>
    <button (click)="values$.next(-1)">-</button>
    <button (click)="values$.next(+1)">+</button>
  `
})
export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

```
@Component({
  selector: "app-root",
  template: `
    <div>{{ count$ | async }}</div>
    <button (click)="values$.next(-1)">-</button>
    <button (click)="values$.next(+1)">+</button>
  `
})
export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

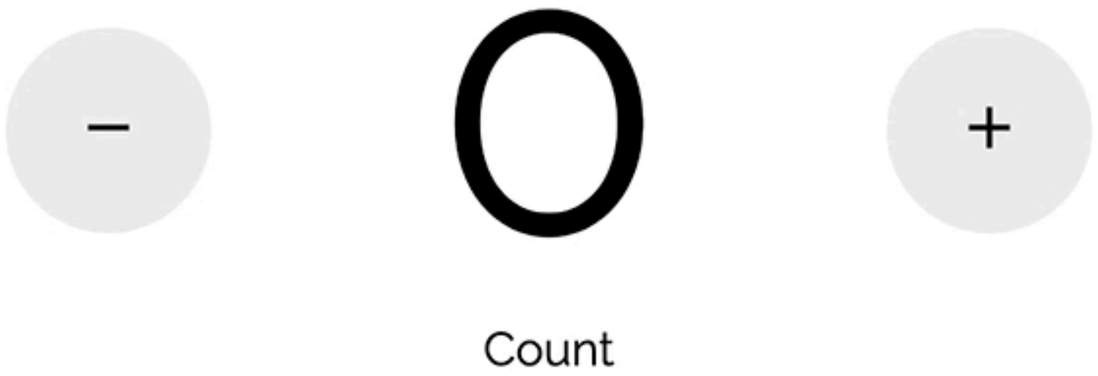
```
$ ng update @angular/core@next @angular/cli@next
```

```
/***************************************************
 * Zone JS is required by default for Angular itself.
 */
import "zone.js/dist/zone"; // Included with Angular CLI.
```

```
/***************************************************
 * Zone JS is required by default for Angular itself.
 */
// import "zone.js/dist/zone"; // Included with Angular CLI.
```

```
platformBrowserDynamic()
  .bootstrapModule(AppModule)
  .catch(console.error);
```

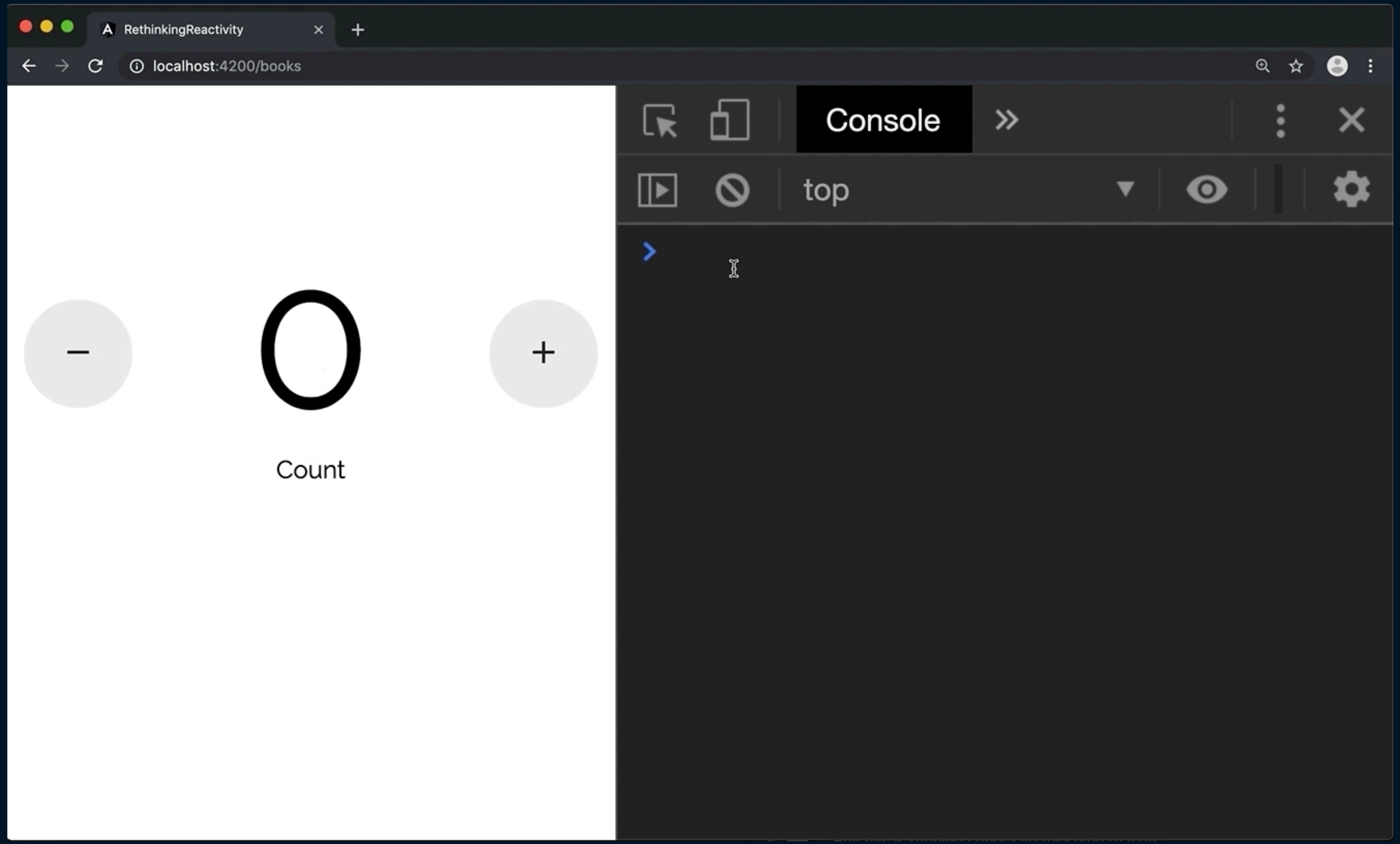
```
platformBrowserDynamic()
  .bootstrapModule(AppModule, { ngZone: "noop" })
  .catch(console.error);
```


0
Count

Change Detection with Ivy

```
class MyComponent {
  constructor(cdRef: ChangeDetectorRef) {
    cdRef.detectChanges();
  }
}
```

```
class MyComponent {
  constructor() {
    markDirty(this);
  }
}
```


RethinkingReactivity
localhost:4200/books
Console
top
0
Count

```ts
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```ts
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```ts
export interface RootContext {
  scheduler: (workFn: () => void) => void;
  clean: Promise<null>;
  components: {}[];
  playerHandler: PlayerHandler | null;
  flags: RootContextFlags;
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/interfaces/view.ts#L573

```ts
export interface RootContext {
  scheduler: (workFn: () => void) => void;
  clean: Promise<null>;
  components: {}[];
  playerHandler: PlayerHandler | null;
  flags: RootContextFlags;
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/interfaces/view.ts#L573

```ts
export interface RootContext {
  scheduler: (workFn: () => void) => void;
  clean: Promise<null>;
  components: {}[];
  playerHandler: PlayerHandler | null;
  flags: RootContextFlags;
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/interfaces/view.ts#L573

```
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

```ts
export function markDirty<T>(component: T) {
  const rootView = markViewDirty(
    getComponentViewByInstance(component)
  )!;

  scheduleTick(
    rootView[CONTEXT] as RootContext,
    RootContextFlags.DetectChanges
  );
}
```

github.com/angular/angular/blob/master/packages/core/src/render3/instructions/change_detection.ts#L49

Using markDirty in a Component

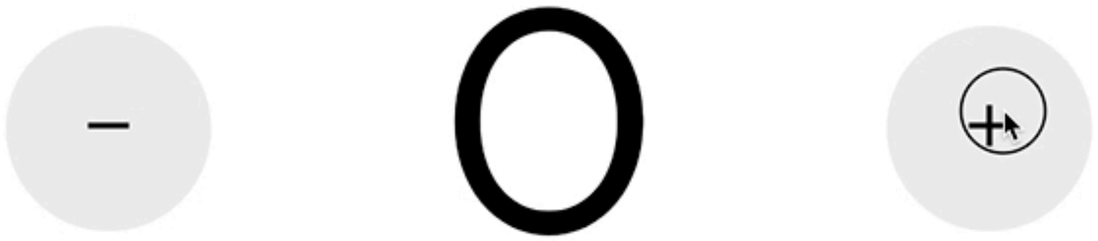
```
export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

```ts
import { ɵmarkDirty as markDirty } from "@angular/core";

export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0)
  );
}
```

```ts
import { ɵmarkDirty as markDirty } from "@angular/core";

export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0),
    tap(() => markDirty(this))
  );
}
```



Count

I don't want to call markDirty

Higher order components with Ivy

Higher order components with Ivy

- **Built using inheritance, decorators, or functions**

Higher order components with Ivy

Built using inheritance, decorators, or functions

- Modifies the behavior of the target component

Higher order components with Ivy

Built using inheritance, decorators, or functions

Modifies the behavior of the target component

- Can call new Ivy APIs like markDirty

```
class MoviesComponent {
  state = connect({
    movies: this.http
      .get("/api/v1/movies")
      .pipe(map(res => res.data))
  });

  constructor(private http: HttpClient) {}
}
```

```
class MoviesComponent extends ReactiveComponent {
  state = this.connect({
    movies: this.http
      .get("/api/v1/movies")
      .pipe(map(res => res.data))
  });

  constructor(private http: HttpClient) {}
}
```

```
class MoviesComponent extends ReactiveComponent {
  state = this.connect({
    movies: this.http
      .get("/api/v1/movies")
      .pipe(map(res => res.data))
  });

  constructor(private http: HttpClient) {}
}
```

```
class MoviesComponent extends ReactiveComponent {
  state = this.connect({
    movies: this.http
      .get("/api/v1/movies")
      .pipe(map(res => res.data))
  });

  constructor(private http: HttpClient) {}
}
```

```
class MoviesComponent extends ReactiveComponent {
  state = this.connect({
    movies: this.http
      .get("/api/v1/movies")
      .pipe(map(res => res.data))
  });

  constructor(private http: HttpClient) {}
}
```

```
class ReactiveComponent {
  connect(sources): ???;
}
```

Get the Types Right!

```ts
type Output = {
  user: UserModel;
  loading: boolean;
  movies: MovieModel[];
};
```

```ts
type Input = {
  user: Observable<UserModel>;
  loading: Observable<boolean>;
  movies: Observable<MovieModel[]>;
};
```

```
class ReactiveComponent {
  connect<T>(sources: ObservableDictionary<T>): T;
}
```

```
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```ts
type State = {
  user: UserModel;
  loading: boolean;
  movies: MovieModel[];
};
```

```ts
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```
"user" | "loading" | "movies";
```

```
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```
type State = {
  user:
  loading:
  movies:
};
```

```
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```ts
type State = {
  user: UserModel;
  loading: boolean;
  movies: MovieModel[];
};
```

```ts
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```ts
type State = {
  user: Observable<UserModel>;
  loading: Observable<boolean>;
  movies: Observable<MovieModel[]>;
};
```

```ts
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

```
type ObservableDictionary<T> = {
  [P in keyof T]: Observable<T[P]>;
};
```

Implementing "connect"

```
connect<T>(sources: ObservableDictionary<T>): T {

}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;

  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];

  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys)




  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey]



    })
  );



  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey].pipe(
        tap((sinkValue: any) => (sink[sourceKey] = sinkValue))
      );
    })
  );



  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey].pipe(
        tap((sinkValue: any) => (sink[sourceKey] = sinkValue))
      );
    })
  );

  updateSink$.subscribe(() => markDirty(this));

  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey].pipe(
        tap((sinkValue: any) => (sink[sourceKey] = sinkValue))
      );
    })
  );

  updateSink$.subscribe(() => markDirty(this));

  return sink;
}
```

```
state = this.connect({
  veggies: of("🥕", "🥦", "🌽")
});
```

```
state = this.connect({
  veggies: of("🥕", "🥦", "🌽")
});
```

```
const sink = {};
```

```
state = this.connect({
  veggies: of("🥕", "🥦", "🌽")
});
```

```
const sink = { veggies: "🥕" };
```

```
state = this.connect({
  veggies: of("🥕", "🥦", "🌽")
});
```

```
const sink = { veggies: "🥦" };
```

```
state = this.connect({
  veggies: of("🥕", "🥦", "🌽")
});
```

```
const sink = { veggies: "🌽" };
```

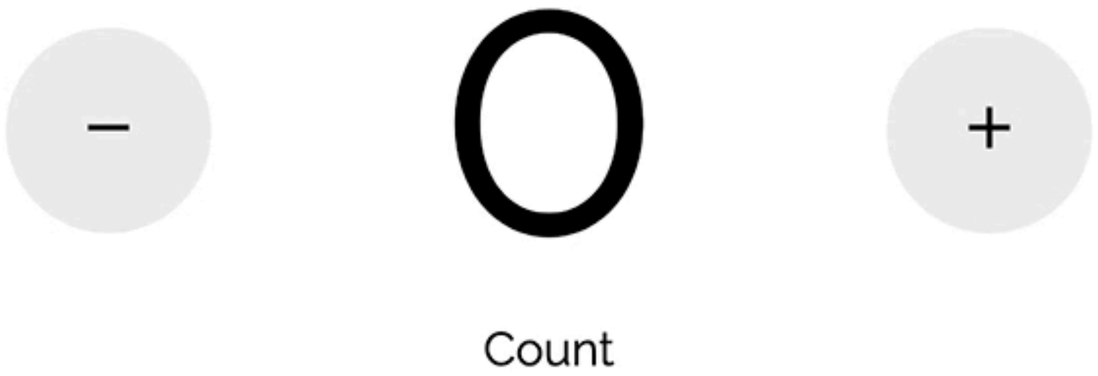
```ts
import { ɵmarkDirty as markDirty } from "@angular/core";

export class AppComponent {
  values$ = new Subject<number>();
  count$ = this.values$.pipe(
    startWith(0),
    scan((count, next) => count + next, 0),
    tap(() => markDirty(this))
  );
}
```

```
export class AppComponent extends ReactiveComponent {
  values$ = new Subject<number>();
  state = this.connect({
    count: this.values$.pipe(
      startWith(0),
      scan((count, next) => count + next, 0)
    )
  });
}
```

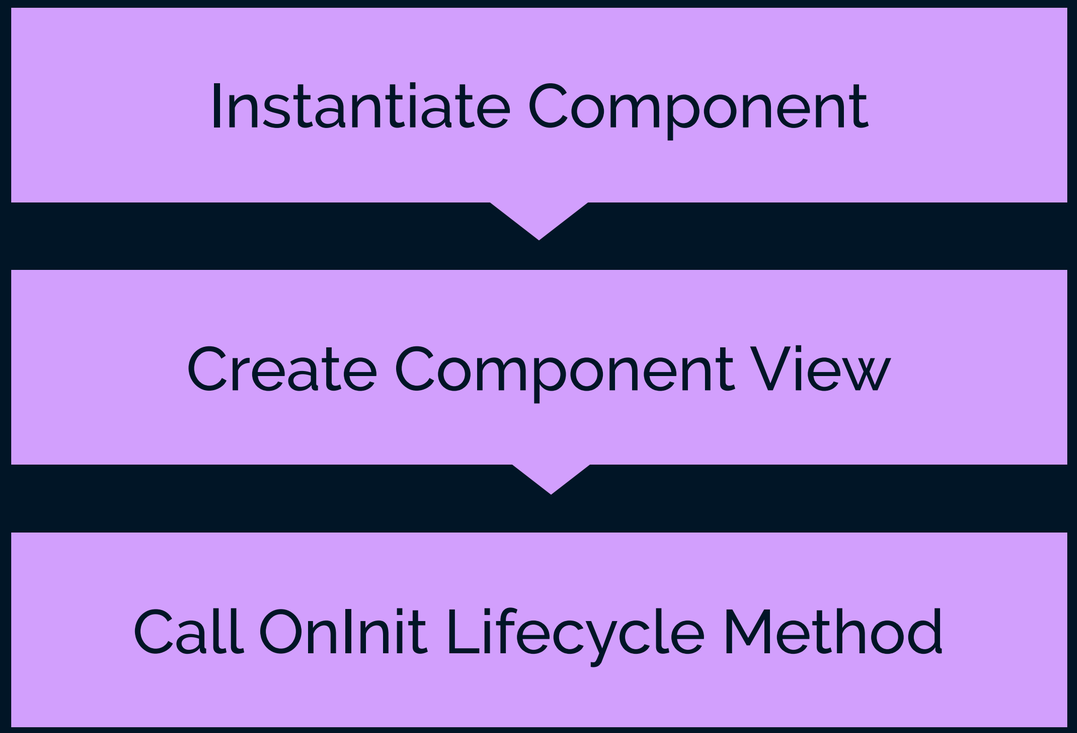
```
<div>{{ count$ | async }}</div>
<button (click)="values$.next(-1)">-</button>
<button (click)="values$.next(+1)">+</button>
```

```
<div>{{ state.count }}</div>
<button (click)="values$.next(-1)">-</button>
<button (click)="values$.next(+1)">+</button>
```


0
Count

```
▶Uncaught TypeError: Cannot read property 'nodeIndex' of undefined
    at getComponentViewByInstance (core.js:7610)
    at markDirty (core.js:20604)
    at SafeSubscriber._next (app.component.ts:62)
    at SafeSubscriber.__tryOrUnsub (Subscriber.js:185)
    at SafeSubscriber.next (Subscriber.js:124)
    at Subscriber._next (Subscriber.js:72)
    at Subscriber.next (Subscriber.js:49)
    at TakeUntilSubscriber._next (Subscriber.js:72)
    at TakeUntilSubscriber.next (Subscriber.js:49)
    at MergeMapSubscriber.notifyNext (mergeMap.js:69)
```

```
Uncaught TypeError: Cannot read property 'nodeIndex' of undefined
    at getComponentViewByInstance (core.js:7610)
    at markDirty (core.js:20604)
    at SafeSubscriber._next (app.component.ts:62)
    at SafeSubscriber.__tryOrUnsub (Subscriber.js:185)
    at SafeSubscriber.next (Subscriber.js:124)
    at Subscriber._next (Subscriber.js:72)
    at Subscriber.next (Subscriber.js:49)
    at TakeUntilSubscriber._next (Subscriber.js:72)
    at TakeUntilSubscriber.next (Subscriber.js:49)
    at MergeMapSubscriber.notifyNext (mergeMap.js:69)
```


Instantiate Component
Create Component View
Call OnInit Lifecycle Method

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey].pipe(
        tap((sinkValue: any) => (sink[sourceKey] = sinkValue))
      );
    })
  );

  updateSink$.subscribe(() => markDirty(this));

  return sink;
}
```

```
connect<T>(sources: ObservableDictionary<T>): T {
  const sink = {} as T;
  const sourceKeys = Object.keys(sources) as (keyof T)[];
  const updateSink$ = from(sourceKeys).pipe(
    mergeMap(sourceKey => {
      return sources[sourceKey].pipe(
        tap((sinkValue: any) => (sink[sourceKey] = sinkValue))
      );
    })
  );

  updateSink$.subscribe(() => markDirty(this));

  return sink;
}
```

Lifecycle Hooks in Higher Order Components

```
class ReactiveComponent {

}
```

```
class ReactiveComponent implements OnInit {
  ngOnInit() {}
}
```

```
class ReactiveComponent implements OnInit {
  onInit$ = new ReplaySubject<true>(1);

  ngOnInit() {
    this.onInit$.next(true);
    this.onInit$.complete();
  }
}
```

```
const OnInitSubject = Symbol("OnInitSubject");
class ReactiveComponent implements OnInit {
  private [OnInitSubject] = new ReplaySubject<true>(1);

  ngOnInit() {
    this[OnInitSubject].next(true);
    this[OnInitSubject].complete();
  }

  get onInit$() {
    return this[OnInitSubject].asObservable();
  }
}
```

```
const OnDestroySubject = Symbol("OnDestroySubject");
class ReactiveComponent implements OnDestroy {
  private [OnDestroySubject] = new ReplaySubject<true>(1);

  ngOnDestroy() {
    this[OnDestroySubject].next(true);
    this[OnDestroySubject].complete();
  }

  get onDestroy$() {
    return this[OnDestroySubject].asObservable();
  }
}
```

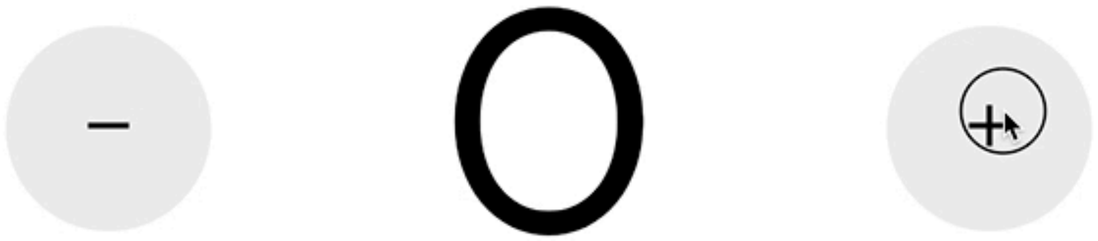
```
updateSink$.subscribe(() => markDirty(this));
```

```
concat(this.onInit$, updateSink$)
  .pipe(takeUntil(this.onDestroy$))
  .subscribe(() => markDirty(this));
```

```
concat(this.onInit$, updateSink$)
  .pipe(takeUntil(this.onDestroy$))
  .subscribe(() => markDirty(this));
```

```
concat(this.onInit$, updateSink$)
  .pipe(takeUntil(this.onDestroy$))
  .subscribe(() => markDirty(this));
```

```
concat(this.onInit$, updateSink$)
  .pipe(takeUntil(this.onDestroy$))
  .subscribe(() => markDirty(this));
```



Count

```
class MoviesComponent {
  movies$: Observable<MovieModel[]>;
  user$: Observable<UserModel>;

  constructor(
    movieService: MovieService,
    userService: UserService
  ) {
    this.movies$ = movieService.movies$;
    this.user$ = userService.activeUser$;
  }
}
```

```
class MoviesComponent extends ReactiveComponent {
  state: { movies: MovieModel[]; user: UserModel };

  constructor(
    movieService: MovieService,
    userService: UserService
  ) {
    this.state = this.connect({
      movies: movieService.movies$,
      user: userService.activeUser$
    });
  }
}
```

```
class MoviesComponent {
  movies$ = this.store.select(selectMovies);
  user$ = this.store.select(selectActiveUser);

  constructor(private store: Store<State>) {}
}
```

```
class MoviesComponent extends ReactiveComponent {
  state = this.connect({
    movies: this.store.select(selectMovies),
    user: this.store.select(selectActiveUser)
  });

  constructor(private store: Store<State>) {}
}
```

```
class MoviesComponent extends StoreComponent {
  state = this.connect({
    movies: selectMovies,
    user: selectActiveUser
  });
}
```

ReactiveComponent

ReactiveComponent

- **Use values directly in the template without the async pipe**

ReactiveComponent

Use values directly in the template without the async pipe

- Makes it easier to not leak memory accidentally

ReactiveComponent

Use values directly in the template without the async pipe

Makes it easier to not leak memory accidentally

- Makes it possible for apps to be built without using Zone.js

Want to use ReactiveComponent?

Help us build it

@ngrx/component

github.com/ngrx/platform

bit.ly/somethingOrOther

Mike Ryan

@MikeRyanDev

Thank You!