Angular Signal

New way of Reactivity in Angular

Why Angular16 introduced the Signal?

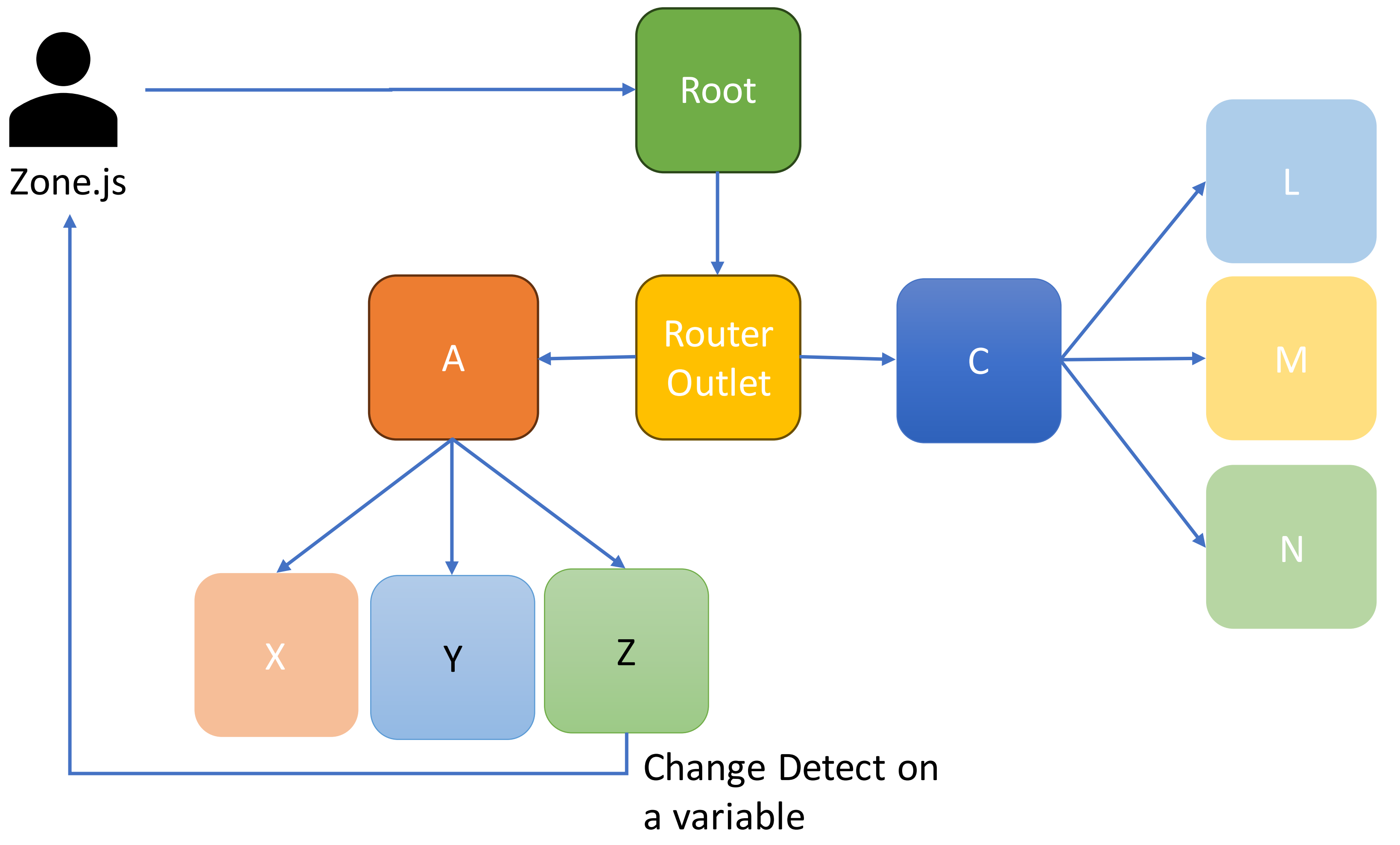
Angular have Rxjs support that provides everything to achieve reactivity.

All over the world there are many projects are running in Angular , it means we have everything today to achieve reactivity in Angular.

But why Angular Introduced the Signal after 7 years from the first release in May 2016.

So Major problem with Angular is performance for the large application due to change detection mechanism. So Signal solves this problem just to apply synchronous reactivity to the particular variable when it changes rather than triggering change detection.

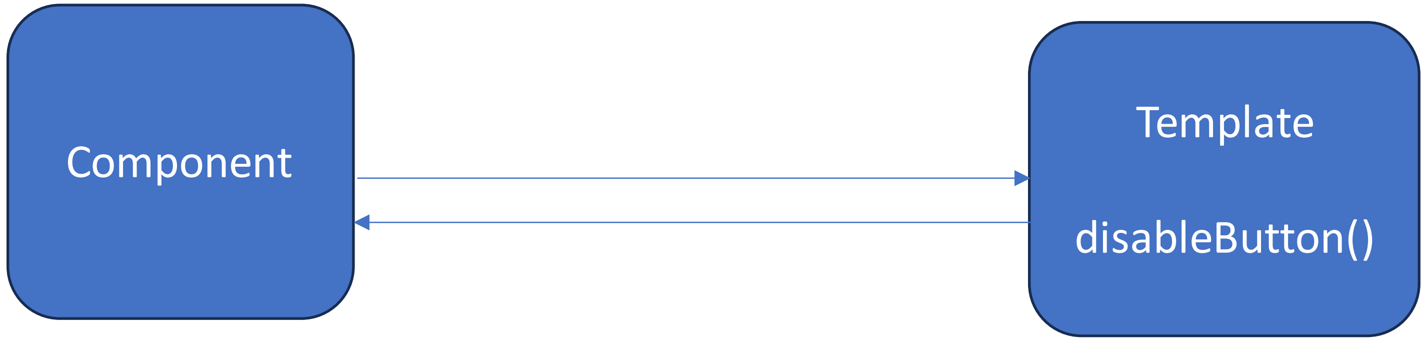
Current Change Detection Mechanism



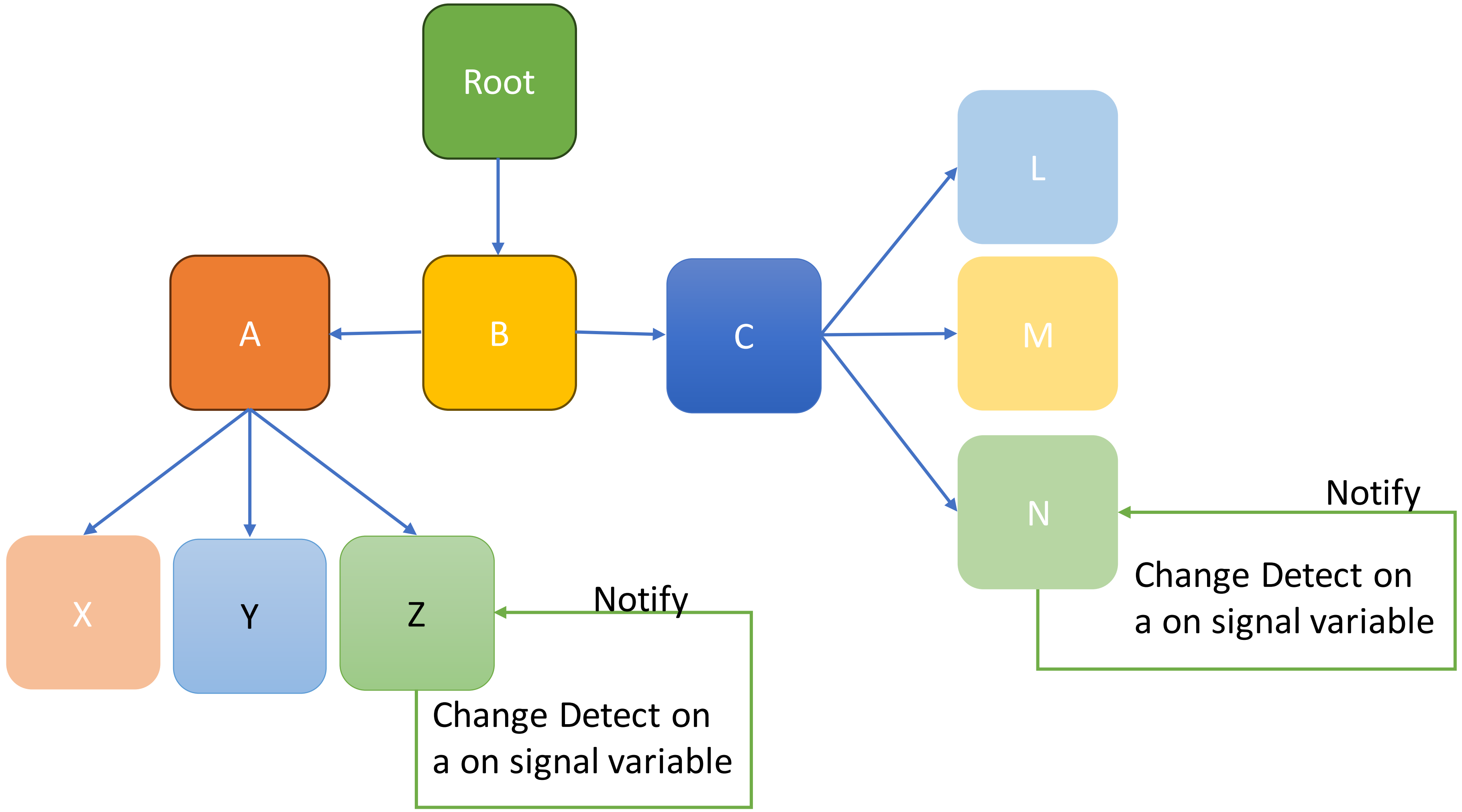
Change Detection can also get triggered by markForCheck method and detectChanges method if you are using push strategy.

Calling a function from template

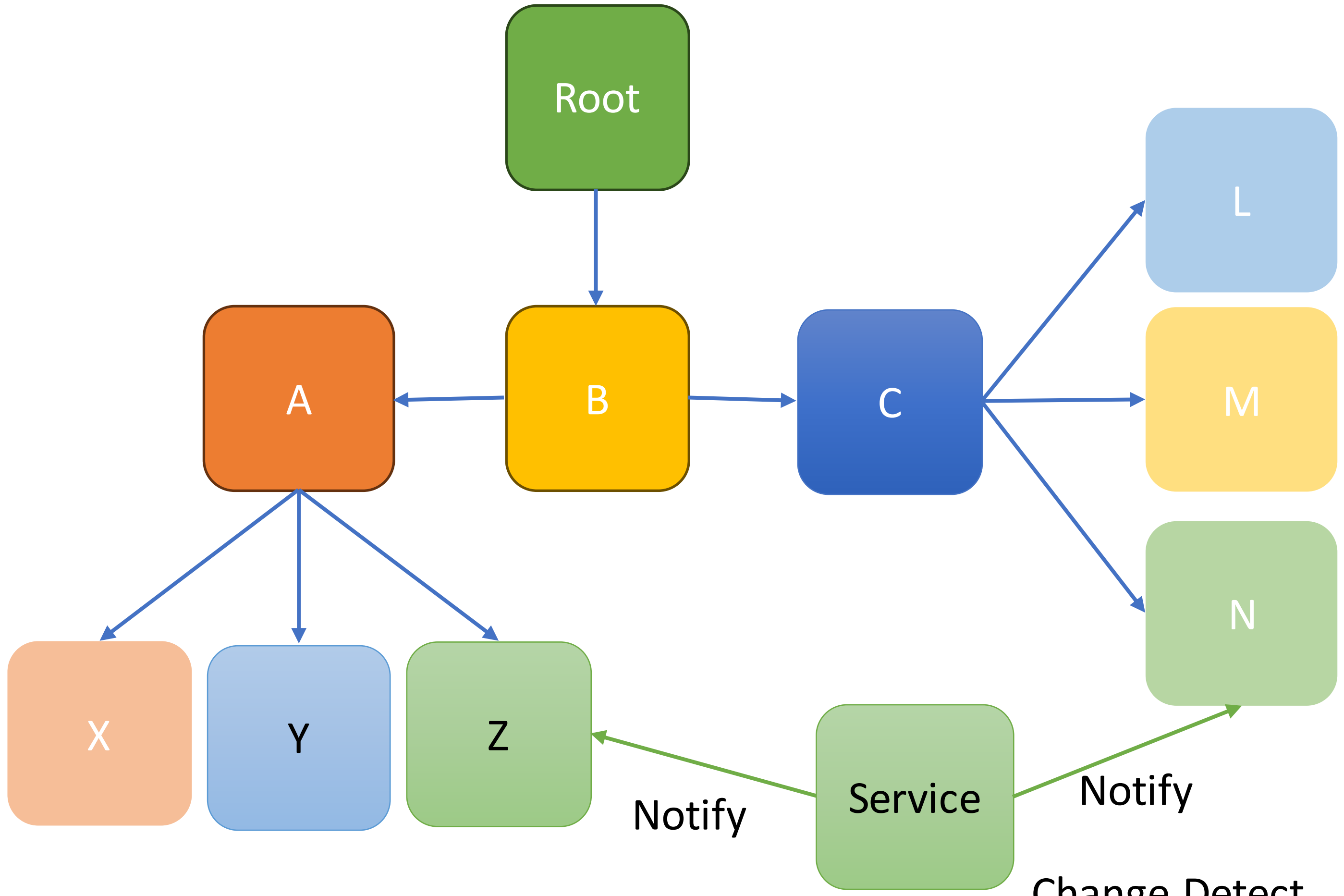
- If the application started build around 2017 or 2018 that time most of developer were new to Angular and they calling a function from template that drive Angular Change Detection crazy.



When Signal variable changed it will get update without Zone.js (Zoneless Angular)


Root
A
B
C
L
M
N
X
Y
Z
Notify
Change Detect on a on signal variable
Notify
Change Detect on a on signal variable

When Signal variable changed it will get update without Zone.js (Zoneless Angular)



Change Detect on a on signal variable

Angular Signal Introduction

A wrapper around a value that can notify interested consumer when that value changes.

Signal can contain any value, from simple primitives to complex data structure.

Angular 16 introduces reactivity through signal.

A Signal is a value that tells Angular when it changes.

Signal is Synchronous unlike Rxjs.

Signal is not going to replace Rxjs but some operator we will stop using like

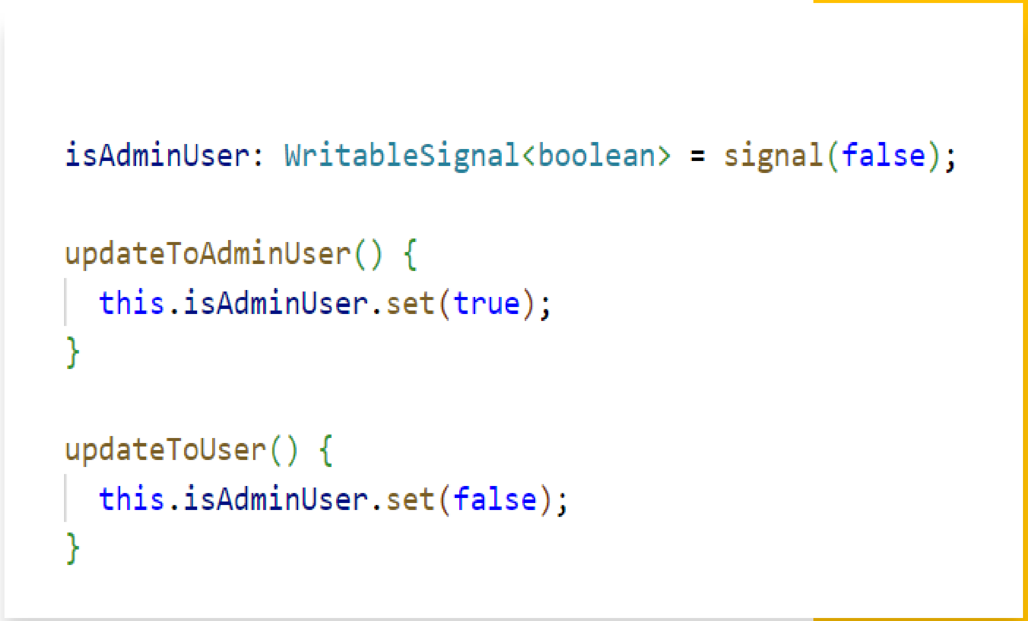
- Subject and BehaviorSubject - Subjects are a type of observable in Angular that allow you to send and receive values between components, NOW this can be achieved by signal

Signal API and types from the @angular/core

- API – signal
- Type – Signal<T> and WritableSignal<T>
- By default signal is writable (can be set, updated and mutated using signal API)
  - Ex: isEditMode = signal(false);
- Signal can be defined as readonly if we specify the the type `Signal` (can be readonly)
  - Ex: isEditMode: Signal<boolean> = signal(false);

How to use `set` function

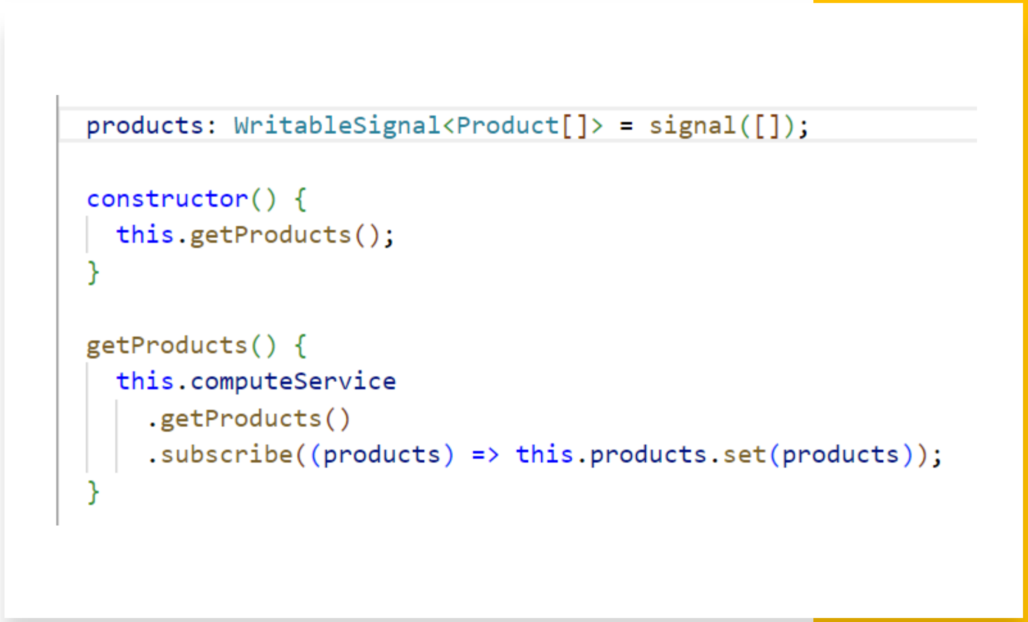
- Set function is available with Writable signal to assign a new value to signal
- Set function can be used with all types of data
- Set can be used to modify the primitives value, if variable depending on the current value then update function is the right choice
- Set can be used to modify the reference type also if we want to replace the previous object



```
isAdminUser: WritableSignal<boolean> = signal(false);

updateToAdminUser() {
  this.isAdminUser.set(true);
}

updateToUser() {
  this.isAdminUser.set(false);
}
```



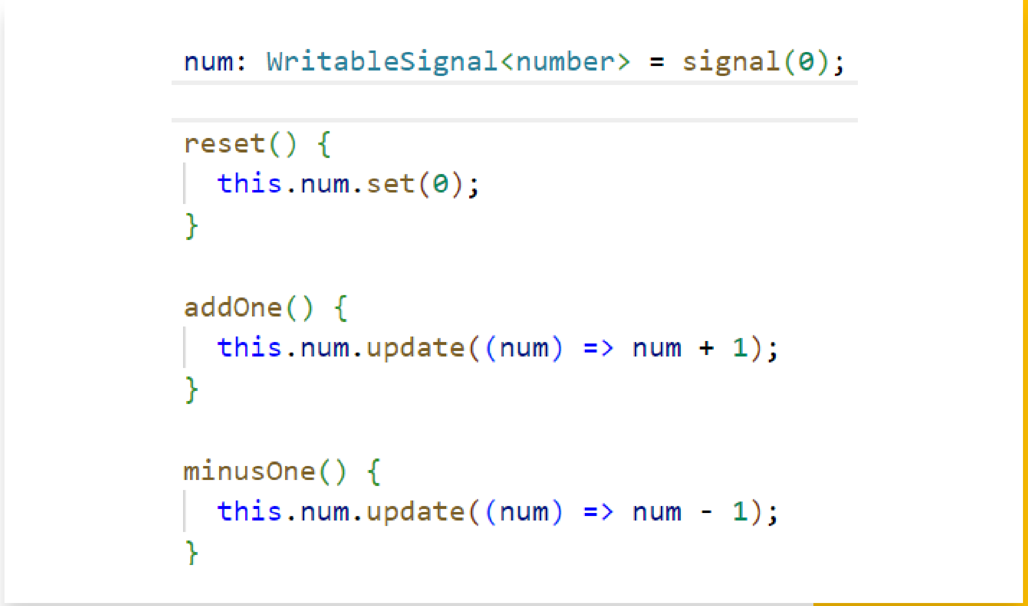
```
products: WritableSignal<Product[]> = signal([]);

constructor() {
  this.getProducts();
}

getProducts() {
  this.computeService
    .getProducts()
    .subscribe((products) => this.products.set(products));
}
```

How to use `update` function

- Update function is available with Writable signal to modify the signal
- Update function can be used on any types, If the new value depending on previous value
- Update function return type should match with previous object type

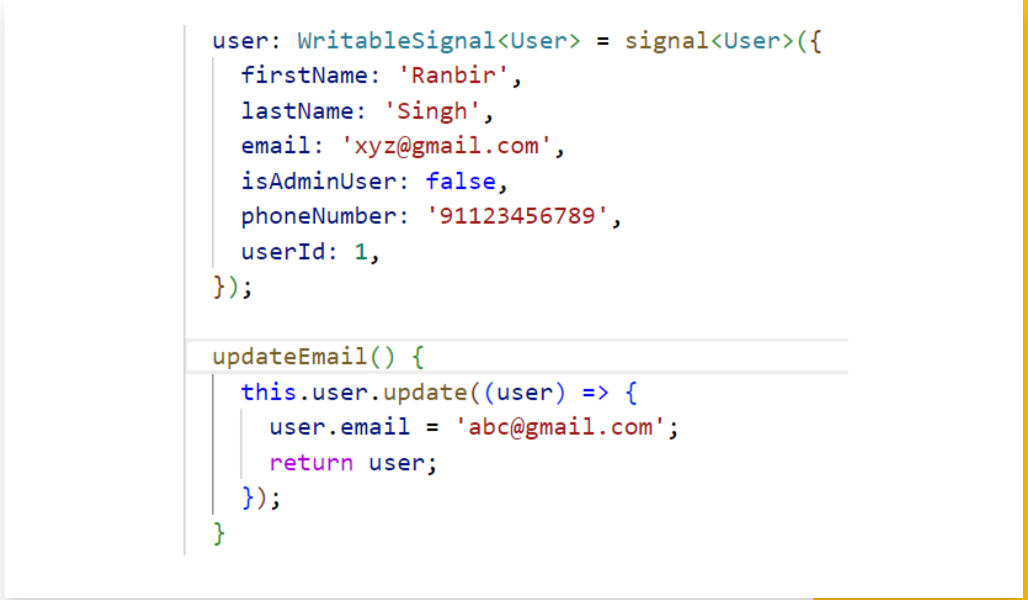


```
num: WritableSignal<number> = signal(0);

reset() {
  this.num.set(0);
}

addOne() {
  this.num.update((num) => num + 1);
}

minusOne() {
  this.num.update((num) => num - 1);
}
```

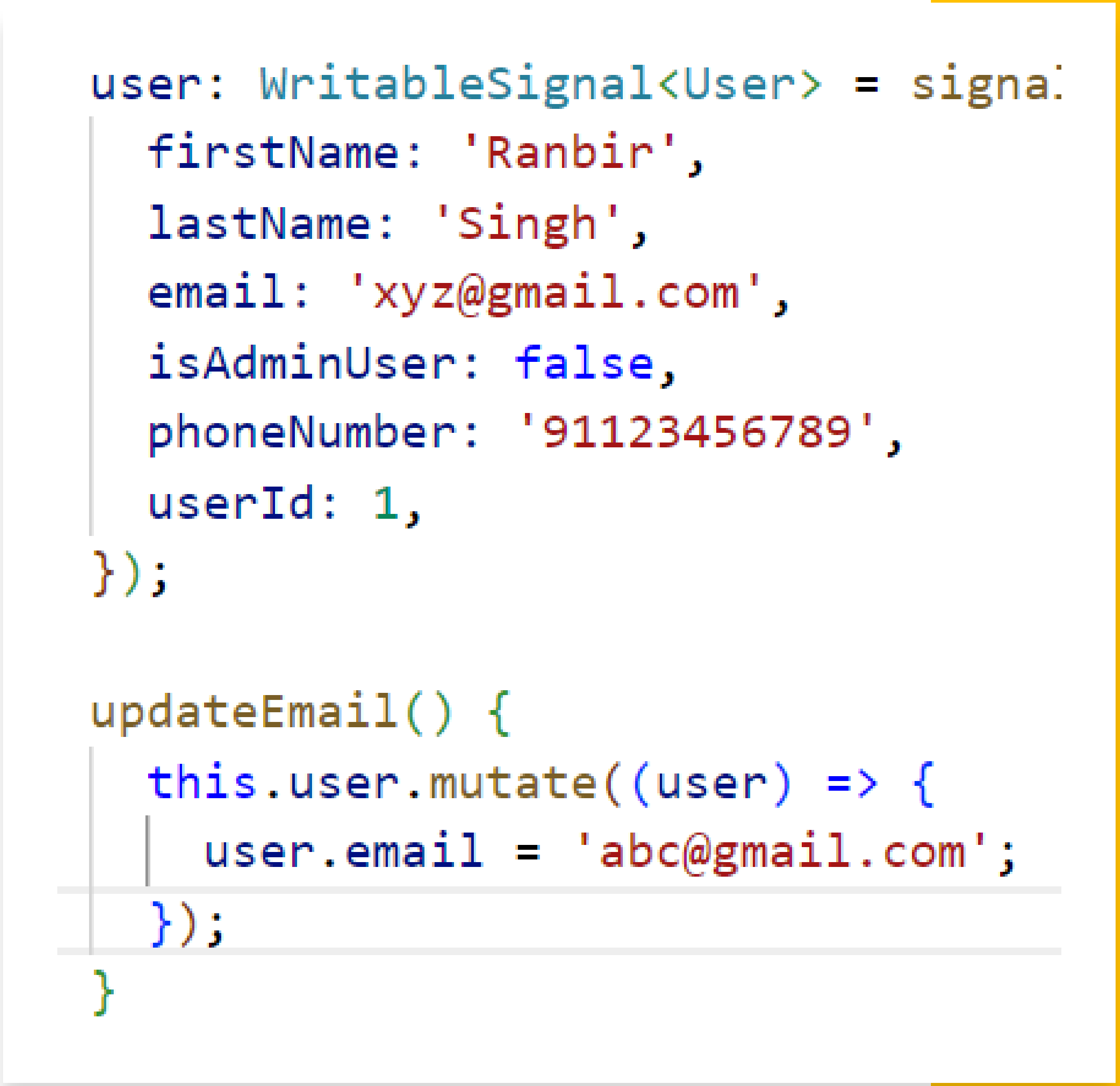


```
user: WritableSignal<User> = signal<User>({
  firstName: 'Ranbir',
  lastName: 'Singh',
  email: 'xyz@gmail.com',
  isAdminUser: false,
  phoneNumber: '91123456789',
  userId: 1,
});

updateEmail() {
  this.user.update((user) => {
    user.email = 'abc@gmail.com';
    return user;
  });
}
```

How to use`mutate` function

- Mutate function is available with Writable signal to modify the signal
- Mutate function mostly used with Array and Object type
- Mutate is used to modify the part of array and object
- Mutate function by default return the mutated object so we need not to return

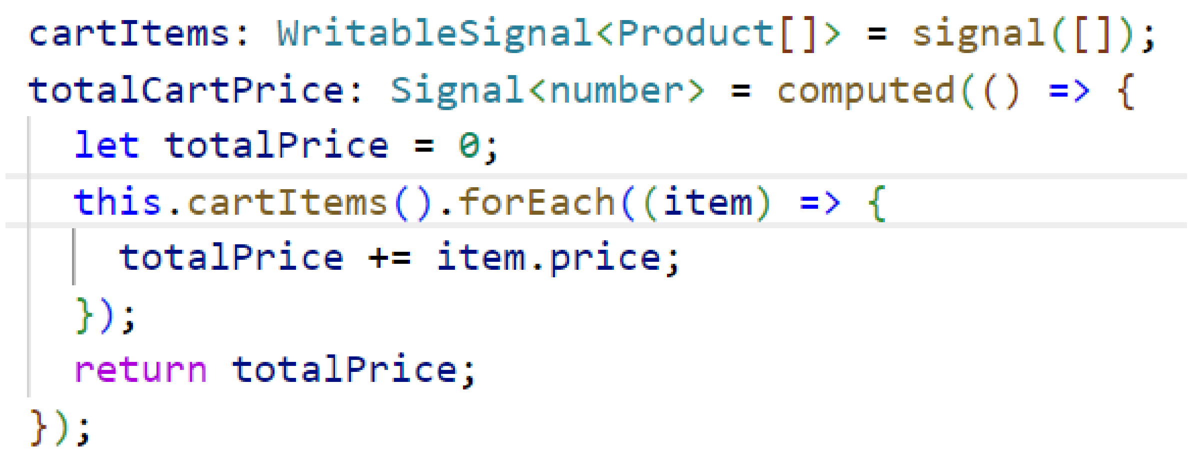


```
user: WritableSignal<User> = signa
  firstName: 'Ranbir',
  lastName: 'Singh',
  email: 'xyz@gmail.com',
  isAdminUser: false,
  phoneNumber: '91123456789',
  userId: 1,
});

updateEmail() {
  this.user.mutate((user) => {
    user.email = 'abc@gmail.com';
  });
}
```

How to use `Computed` API from Signal

- Computed API from Signal is depending on signal variable, if the signal variable change then this computed function trigger automatically and create new signal with new computed value from signal
- Example: on change cartItems signal automatically executed computed function and calculate the totalCartPrice



```
cartItems: WritableSignal<Product[]> = signal([]);
totalCartPrice: Signal<number> = computed(() => {
  let totalPrice = 0;
  this.cartItems().forEach((item) => {
    totalPrice += item.price;
  });
  return totalPrice;
});
```

Reading signals in OnPush components

- When an OnPush component uses a signal's value in its template, Angular will track the signal as a dependency of that component. When that signal is updated, Angular automatically marks the component to ensure it gets updated the next time change detection runs.

Effects

Signals are useful because they can notify interested consumers when they change. An effect is an operation that runs whenever one or more signal values change.

Effects always run at least once.

Effects always execute asynchronously, during the change detection process.

Use cases for effects

- Effects are rarely needed in most application code, but may be useful in specific circumstances.
  - Logging data being displayed and when it changes, either for analytics or as a debugging tool
  - Keeping data in sync with window.localStorage
  - Adding custom DOM behavior that can't be expressed with template syntax
  - Performing custom rendering to a <canvas>, charting library, or other third party UI library

When not to use effects

- Avoid using effects for propagation of state changes. This can result in ExpressionChangedAfterItHasBeenChecked errors, infinite circular updates, or unnecessary change detection cycles.
- Because of these risks, setting signals is disallowed by default in effects, but can be enabled if absolutely necessary.

Destroying effects

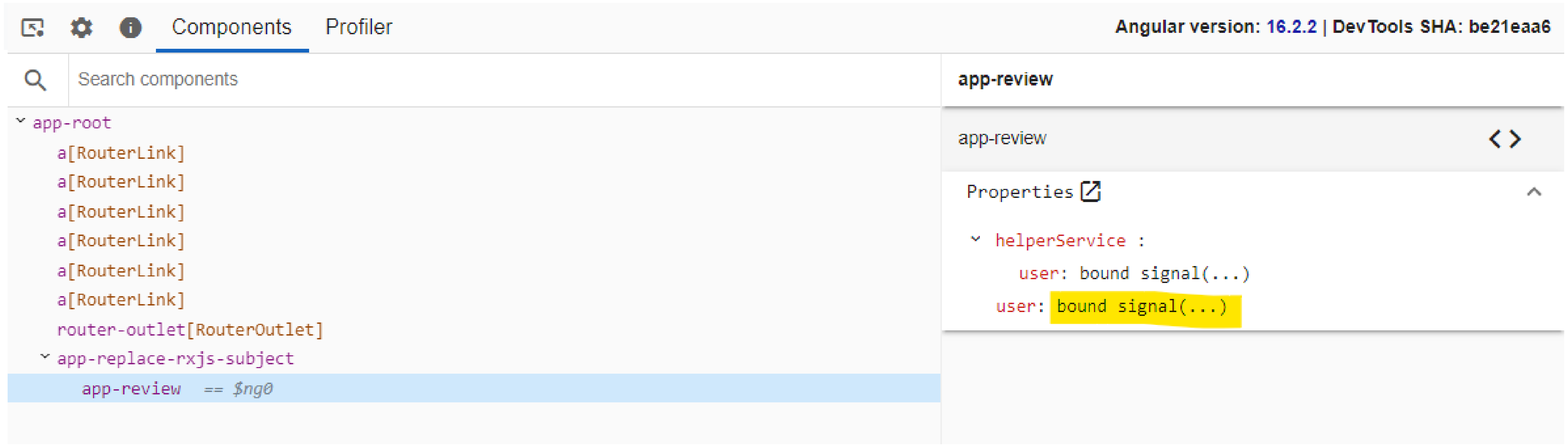
- When you create an effect, it is automatically destroyed when its enclosing context is destroyed. This means that effects created within components are destroyed when the component is destroyed. The same goes for effects within directives, services, etc.
- Effects return an EffectRef that can be used to destroy them manually, via the .destroy() operation. Be careful to actually clean up such effects when they're no longer required.

Advanced topics

- Signal equality functions
- Reading without tracking dependencies
- Effect cleanup function

- Note: we are not covering advanced topic now because in Angular 16 signal is only for developer preview, once it is production ready in Angular 17 then we will cover advanced topic.

Angular Devtools not able to display the signal value, It may be fixed in Angular17



```
app-root
  a[RouterLink]
  a[RouterLink]
  a[RouterLink]
  a[RouterLink]
  a[RouterLink]
  a[RouterLink]
  router-outlet[RouterOutlet]
  app-replace-rxjs-subject
    app-review  == $ng0
```

```
helperService :
  user: bound signal(...)
user: bound signal(...)
```