In Angular, if you want a **pipe** to recompute whenever an **RxJS event** (Observable emission) occurs, you have a few options depending on whether you're talking about:

- **Template pipes** (pure or impure)
- **Custom pipes** that subscribe to Observables
- **AsyncPipe** (recommended for most cases)

1. Recommended: Use `AsyncPipe` in the Template

If your data comes from an RxJS `Observable`, you can bind it directly in the template with `| async`.
Angular will automatically re-render when the observable emits.

Example:

```
// component.ts
import { Component } from '@angular/core';
import { BehaviorSubject, interval } from 'rxjs';
import { map } from 'rxjs/operators';

@Component({
  selector: 'app-root',
  template: `
    <p>Counter: {{ counter$ | async }}</p>
  `
})
export class AppComponent {
  private counterSubject = new BehaviorSubject<number>(0);
  counter$ = this.counterSubject.asObservable();

  constructor() {
    // Simulate updates every second
    interval(1000).subscribe(val => this.counterSubject.next(val));
  }
}
```

✅ **Advantages**: No manual subscription, automatic change detection, no memory leaks.

2. Custom Pipe That Recomputes on Observable Emission

If you need transformation logic inside a pipe that reacts to an RxJS event, you can make an **impure pipe** or a pipe that internally subscribes.

Example:

```
import { Pipe, PipeTransform, OnDestroy } from '@angular/core';
import { Observable, Subscription } from 'rxjs';

@Pipe({
  name: 'observableTransform',
  pure: false // important: allows recomputation on new emissions
})
export class ObservableTransformPipe implements PipeTransform,
OnDestroy {
  private latestValue: any;
  private subscription?: Subscription;

  transform(obs$: Observable<any>, transformFn: (val: any) => any):
any {
    if (!this.subscription) {
      this.subscription = obs$.subscribe(value => {
        this.latestValue = transformFn(value);
      });
    }
    return this.latestValue;
  }

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

Usage:

```
<p>{{ myObservable$ | observableTransform: (v => v * 2) }}</p>
```

⚠️ **Note**: Impure pipes run more often and can impact performance.

3. Trigger Pipe Recalculation via `ChangeDetectorRef`

If you have a **pure pipe** but want it to recompute when an RxJS event happens, you can:

- Subscribe in the component
- Store the latest value in a property
- Let Angular's change detection trigger the pipe again

Example:

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

@Component({
  selector: 'app-root',
  template: `<p>{{ value | myCustomPipe }}</p>`
})
export class AppComponent {
  value = 0;
  event$ = new Subject<number>();

  constructor(private cdr: ChangeDetectorRef) {
    this.event$.subscribe(val => {
      this.value = val;
      this.cdr.markForCheck(); // forces view update
    });
  }
}
```

Best Practice

- If possible, **keep transformation logic in the component or use** `AsyncPipe`.
- Use **impure pipes** only when necessary, as they run on every change detection cycle.

  - Avoid manual subscriptions in pipes unless you handle cleanup properly.

If you want, I can give you a **ready-made Angular pipe** that automatically recomputes whenever an RxJS `Subject` or `BehaviorSubject` emits, without needing manual change detection.
Do you want me to prepare that?