In Angular, if your `ElementRef` becomes `undefined` inside a mouse click event handler in a directive, it usually means one of these is happening:

- `this` **context is lost** — If you attach the event listener manually (e.g., `element.addEventListener('click', this.onClick)`), the method loses its `this` binding, so `this.elementRef` becomes `undefined`.
- **Directive destroyed before event fires** — If the element is removed from the DOM before the event is processed, Angular destroys the directive instance, and `ElementRef` is no longer valid.
- **Event handler not bound via Angular** — Using native DOM listeners without proper cleanup can cause unexpected behavior.

✅ Correct way: Use `@HostListener`

Angular's `@HostListener` automatically binds the method to the directive instance and preserves `this`.

```
import { Directive, ElementRef, HostListener } from '@angular/core';

@Directive({
  selector: '[appClickHighlight]'
})
export class ClickHighlightDirective {
  constructor(private el: ElementRef) {}

  @HostListener('click', ['$event'])
  onClick(event: MouseEvent) {
    // ElementRef is always available here while directive is alive
    this.el.nativeElement.style.backgroundColor = 'yellow';
    console.log('Clicked element:', this.el.nativeElement);
  }
}
```

❌ Problematic approach (causes `undefined`)

```
constructor(private el: ElementRef) {
```

```
  // ❌ Loses `this` context
  this.el.nativeElement.addEventListener('click', this.onClick);
}

onClick(event: MouseEvent) {
  // Here, `this` is not the directive instance → `this.el` is
undefined
}
```

✅ If you must use `addEventListener`

Bind the method to preserve `this`:

```
constructor(private el: ElementRef) {
  this.el.nativeElement.addEventListener('click',
this.onClick.bind(this));
}

onClick(event: MouseEvent) {
  this.el.nativeElement.style.backgroundColor = 'yellow';
}
```

Also, remember to **remove the listener** in `ngOnDestroy` to prevent memory leaks:

```
private boundClick = this.onClick.bind(this);

constructor(private el: ElementRef) {
  this.el.nativeElement.addEventListener('click', this.boundClick);
}

ngOnDestroy() {
  this.el.nativeElement.removeEventListener('click',
this.boundClick);
}
```

Key Takeaways

- Use `@HostListener` whenever possible — it's Angular's safe, idiomatic way.
- If using native listeners, bind `this` and clean up in `ngOnDestroy`.
- If `ElementRef` is `undefined`, check if the directive is still alive when the event fires.

If you want, I can give you a **minimal runnable Angular StackBlitz** showing both the broken and fixed versions so you can debug it live.
Do you want me to prepare that?