Great question! In **Spring Boot**, you can make **non-blocking API calls** using **WebClient**, which is part of **Spring WebFlux**.

✅ Step-by-Step Guide to Call API Non-Blocking in Spring Boot

🎁 1. Add the Dependency

If you're using **Maven**, add:

```
xml
CopyEdit
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Even if you're not building a full reactive app, you can still use `WebClient`.

□ 2. Configure WebClient

You can create a `WebClient` bean:

```
@Configuration
public class WebClientConfig {

    @Bean
    public WebClient webClient() {
        return WebClient.builder().build();
    }
}
```

Or use inline:

```
WebClient webClient = WebClient.create();
```

🚀 3. Make a Non-Blocking API Call

Example – GET Request:

```
public Mono<String> getData() {
    return WebClient.create()
            .get()
            .uri("https://api.example.com/data")
            .retrieve().bodyToMono(String.class);}
```

Example – POST Request:

```
public Mono<ResponseEntity<String>> sendData(MyRequestObject request) {
    return WebClient.create()
            .post()
            .uri("https://api.example.com/send")
            .bodyValue(request)
            .retrieve()
            .toEntity(String.class);
}
```

□ 4. How to Use It?

Use `Mono`/`Flux` to handle the response reactively:

```
@GetMapping("/proxy")
public Mono<String> proxyCall() {
    return myService.getData(); // Returns Mono<String>
}
```

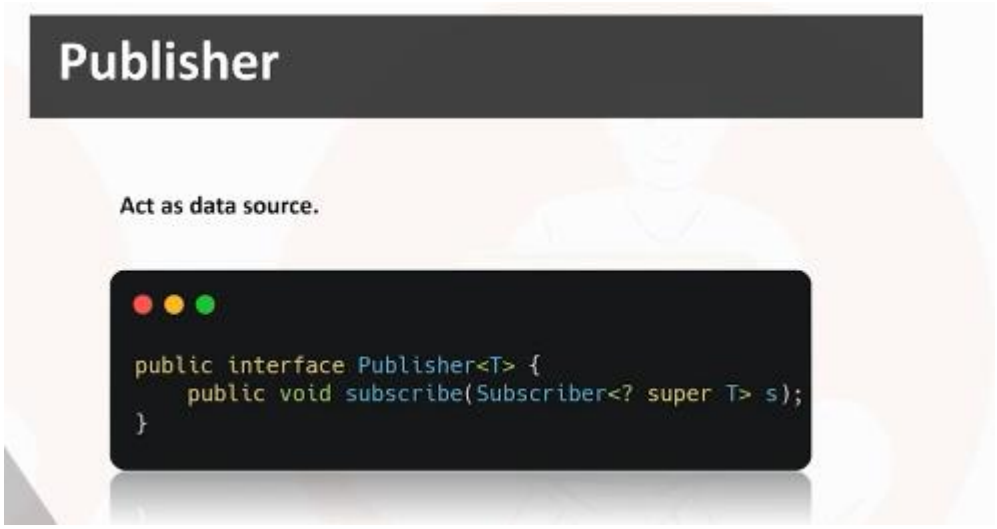
⚠□ Avoid `block()` in Production

```
String result = getData().block(); // ✕ Blocking! Only use for testing or small CLI tools
```

Use `.subscribe()`, `.map()`, `.flatMap()`, or return `Mono`/`Flux` from your controllers/services for **true non-blocking behavior**.

How Mono And Flux work internally

Here Publisher work as data source and having one method ->Subscriber



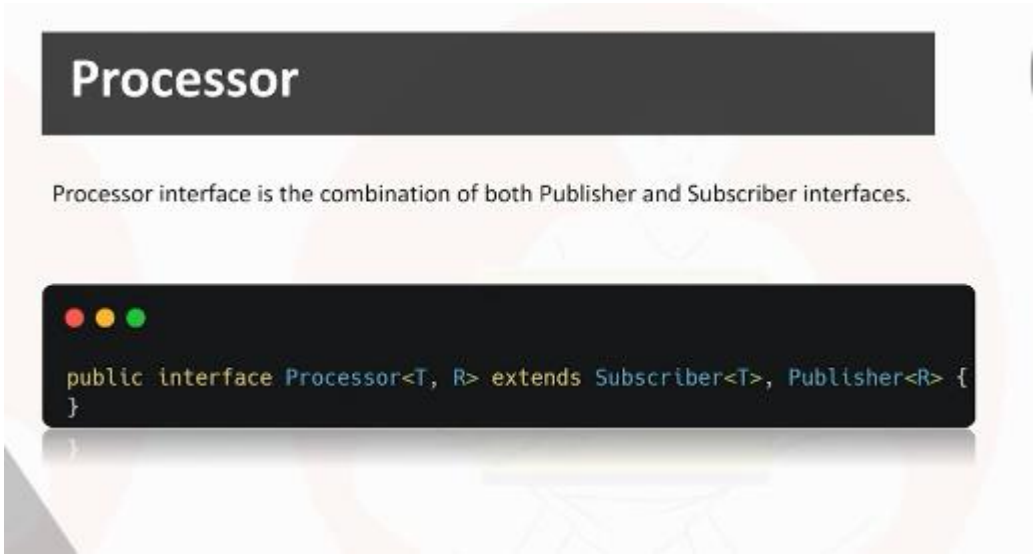
Here subscriber work as receiver

Here publics will call internally onSubscribe to send data and receive data they will call onNext

If error occurs then it will call onError if data complete it will call onComplete

If any number of data or limit want to apply then we have to apply on onSubscribe

If subscriber complet all task and call subscription then we can call bellow all method



□ Bonus: With Spring Boot 3 & Project Reactor, you can also use:

- Retry
- Timeout
- Error handling with `.onErrorResume()`, `.doOnError()`
-