Blocking vs Non-Blocking (Imagine a food order system):

Blocking (Traditional Way):

- You go to a food stall and place your order.
- You **wait there doing nothing** until your food is ready.
- The cook finishes it, gives it to you, and then you leave.
- ➜ **You're blocked (stuck)** while waiting!

This is like traditional Java (Servlet model) — one thread per request, and it waits (blocks) for a response (DB/API/etc.).

Non-Blocking (Reactive Way):

- You place your order and **get a token**.
- You walk away and continue doing other things (maybe place another order!).
- When your food is ready, they **notify you** (call your token number).
- You come, collect, and enjoy.
- ➜ You never waited — you're **non-blocked and more efficient**!

In reactive programming, the system doesn't wait. It uses **callbacks** or **events** to notify when a task (like DB or API call) is done.

Why non-blocking is good:

Feature	**Blocking (Old)**	**Non-Blocking (Reactive)**
Threads per request	1	Very few
Waits on DB/API	Yes	No
Handles many users	Hard	Easy
Speed	Slower with scale	Fast under load
Resources	Wasted while waiting	Used efficiently

In Spring Boot:

With **Spring WebFlux + R2DBC**,

Uses a **small number of threads**.

- Handles **thousands of requests** by **reacting to events** (instead of blocking).
- Is a **good choice for high-performance**, **low-latency apps**, like chat apps, stock apps, or IoT.