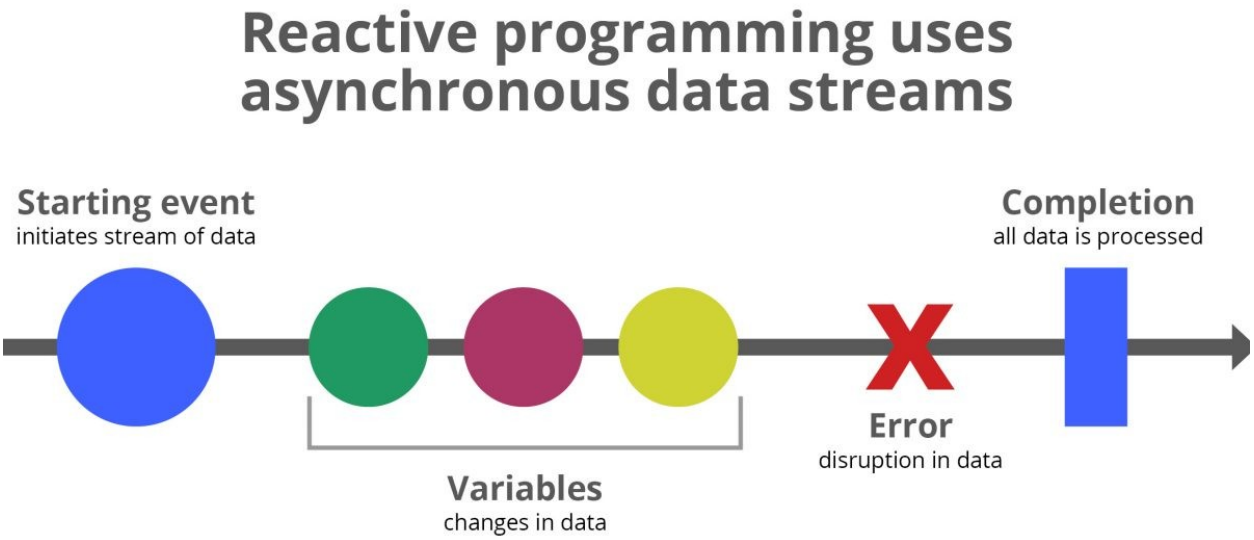
# WHY WE CHOOSE REACTIVE:

## What is Spring reactive and Reactive programming:

Reactive programming is a paradigm that allows applications to handle a large number of concurrent connections efficiently by using non-blocking I/O operations and an event-driven model. Spring provides two main modules for building reactive applications:

1. **Spring WebFlux**: A reactive web framework that provides a non-blocking and asynchronous programming model for building web applications. It supports reactive streams and back pressure, enabling efficient data streaming and processing.
2. **Spring Data MongoDB Reactive**: Part of the Spring Data project that provides reactive support for MongoDB, allowing developers to interact with MongoDB in a reactive and non-blocking manner.




## Spring WebFlux:

Spring WebFlux is designed to handle a high volume of concurrent requests by leveraging non-blocking I/O operations and reactive streams. Key features include:

- **Non-Blocking I/O**: Avoids thread blocking while waiting for I/O operations, enabling more efficient resource utilization and better scalability.
- **Reactive Streams Back pressure**: Supports reactive streams with back pressure, allowing control over data flow to prevent overloading and ensure efficient resource utilization.
- **Efficient Data Streaming**: Well-suited for streaming scenarios, where data can be processed as it becomes available, rather than waiting for the entire data set to be loaded into memory.

- **Concurrency and Parallelism**: Leverages the reactive programming model, enabling better concurrency and parallelism when handling multiple requests simultaneously.

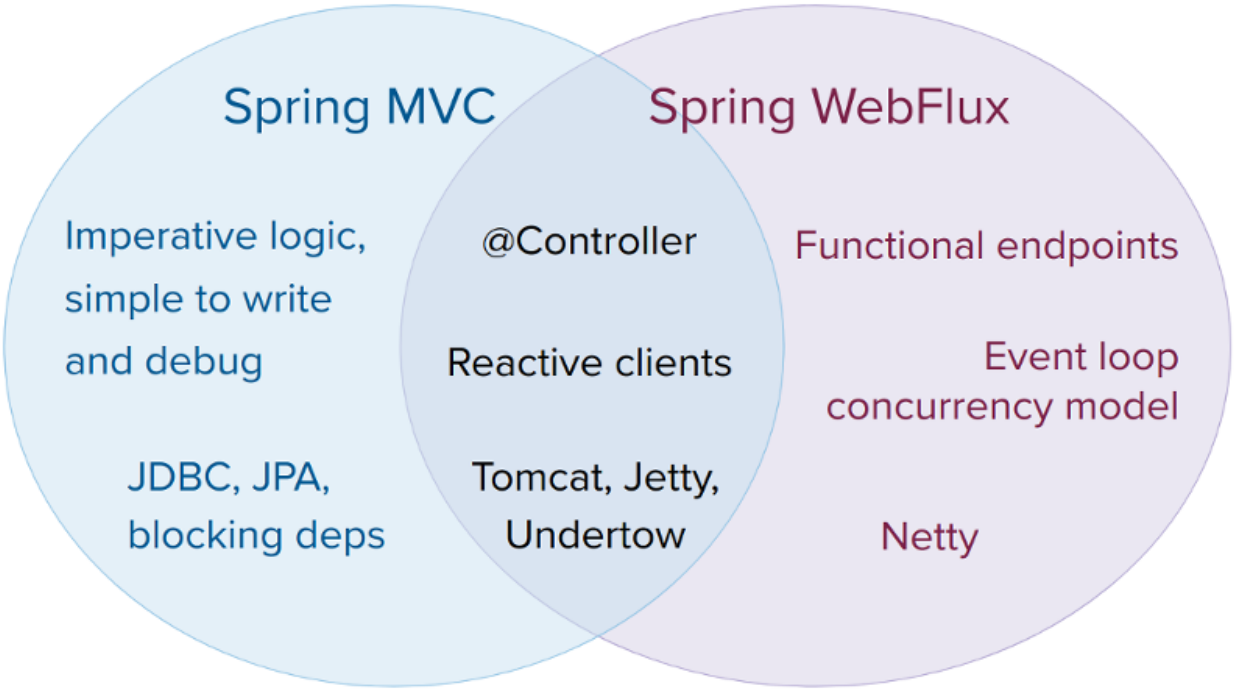
## Spring Data MongoDB Reactive:

Spring Data MongoDB Reactive allows developers to interact with MongoDB in a reactive and non-blocking manner, leveraging the benefits of reactive programming. Key features include:

- **Reactive CRUD Operations**: Perform Create, Read, Update, and Delete operations on MongoDB collections using a reactive and non-blocking API.
- **Integration with Spring WebFlux**: Seamless integration with Spring WebFlux, enabling end-to-end reactive applications for efficient data access and processing.

## SPRING BLOCKING VS SPRING REACTIVE:

Spring WebFlux and traditional Spring MVC (also known as Spring Web) differ in their approach to handling requests and managing concurrency. Here's a comparison between the blocking and reactive models:

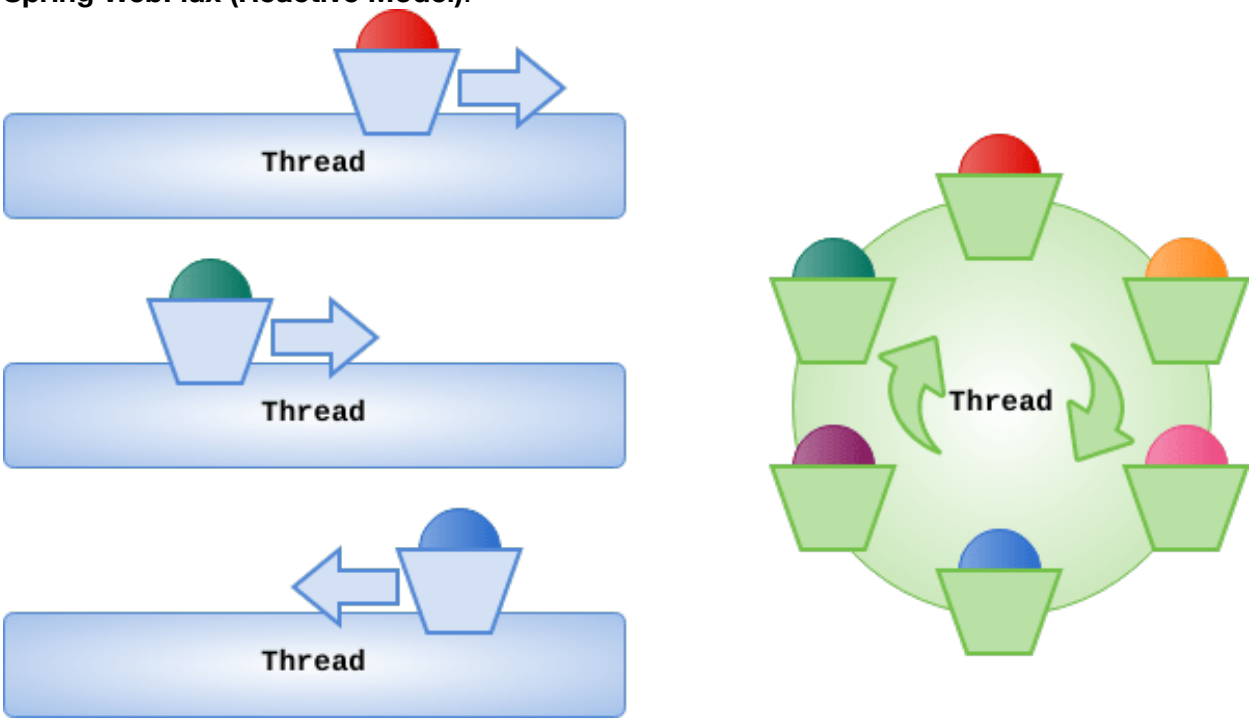


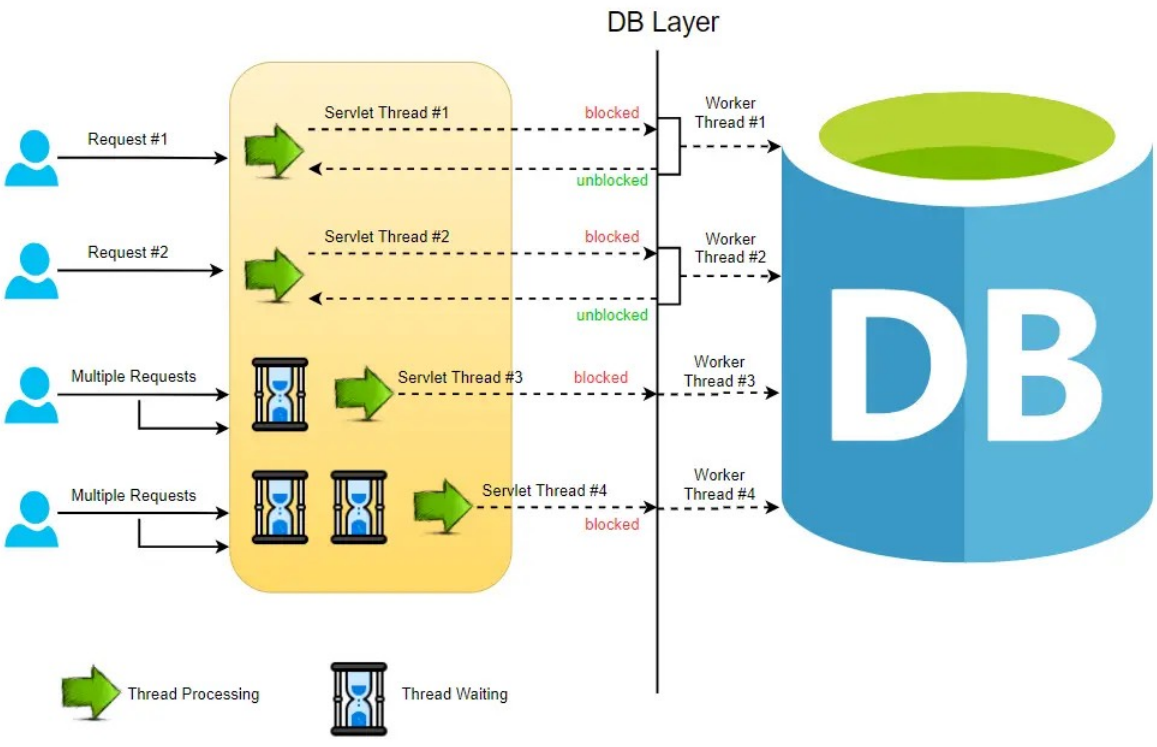

## Spring MVC (Blocking Model):

1. **Thread-per-Request**: In the traditional Spring MVC model, each incoming request is handled by a dedicated thread from a thread pool. This thread remains blocked until the request is fully processed and the response is sent back to the client.
2. **Synchronous Operations**: Spring MVC applications typically use synchronous operations, such as blocking I/O operations (e.g., database queries, remote service calls) or CPU-intensive tasks. These operations can cause the thread to be blocked, potentially leading to thread starvation and reduced scalability under high load.
3. **Scalability Limitations**: Since each request occupies a thread, the number of concurrent requests that can be handled is limited by the size of the thread pool. Increasing the thread

pool size can help to some extent, but it also increases resource consumption (memory, CPU) and can lead to other issues like context switching overhead.

**Spring WebFlux (Reactive Model)**:



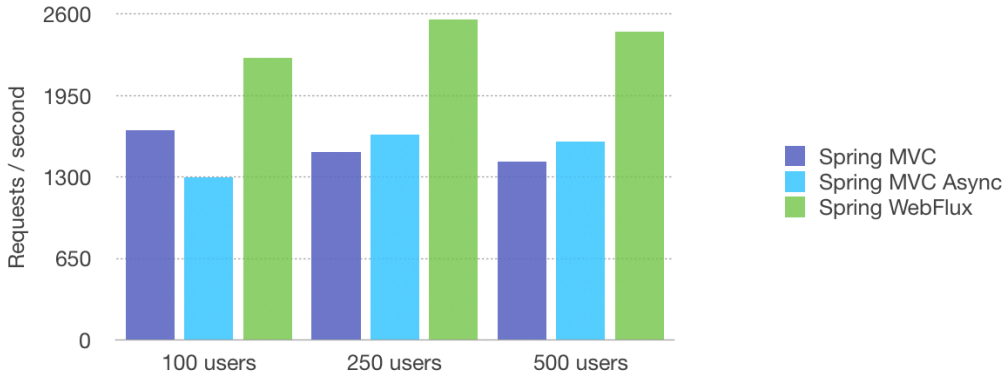



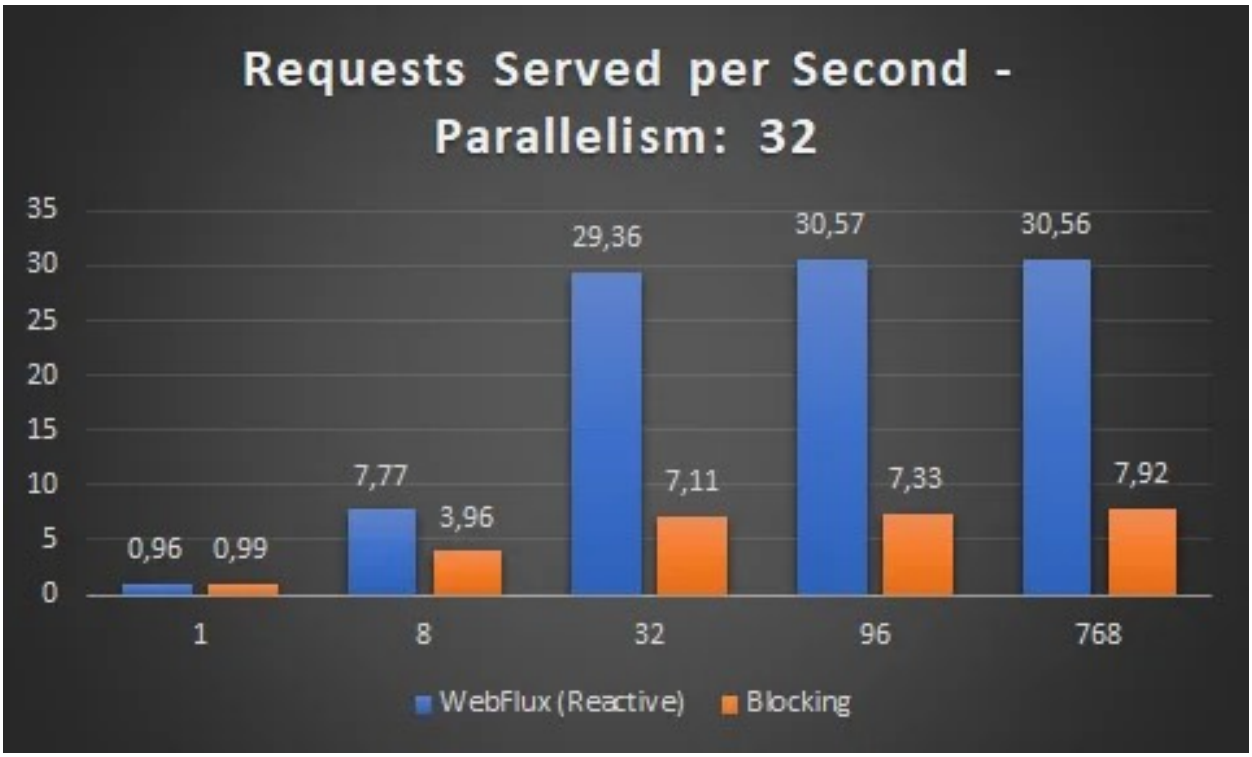
1. **Non-Blocking I/O**: Spring WebFlux is built on a reactive programming model that leverages non-blocking I/O operations. Instead of dedicating a thread to each request, WebFlux uses a small number of event loop threads to handle multiple concurrent requests.
2. **Asynchronous Operations**: WebFlux applications use asynchronous operations, which means that when an I/O operation is initiated, the thread is not blocked. Instead, the operation is offloaded to a separate worker thread or event loop, and the original thread can continue processing other requests.
3. **Backpressure**: Reactive streams in WebFlux support backpressure, which allows the consumer to control the rate at which data is produced, preventing overloading and ensuring efficient resource utilization.
4. **Improved Scalability**: By avoiding thread blocking and leveraging non-blocking I/O, WebFlux can handle a large number of concurrent requests with fewer threads compared to the traditional blocking model. This leads to better scalability and more efficient resource utilization, especially in scenarios with high concurrency or long-running I/O operations.
5. **Reactive Programming Model**: WebFlux embraces the reactive programming paradigm, which involves working with data streams and composing asynchronous operations using operators like **map**, **flatMap**, and **filter**. This programming model can be more complex than the traditional imperative style but offers benefits in terms of concurrency and composability.

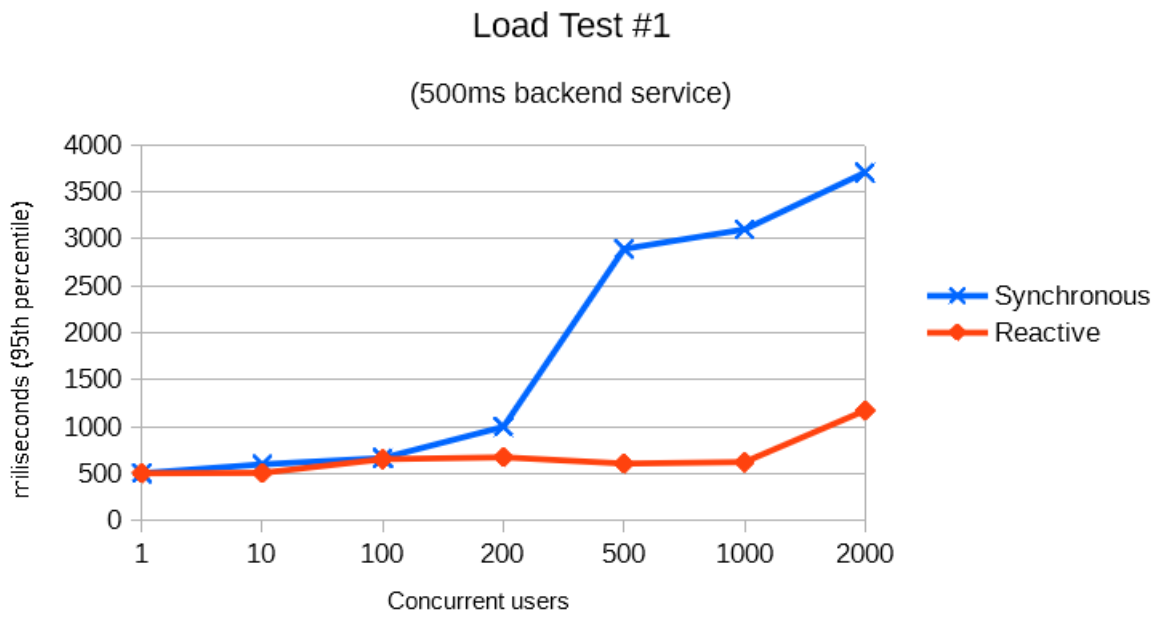
In summary, Spring MVC follows a blocking, thread-per-request model, while Spring WebFlux adopts a non-blocking, reactive approach. WebFlux is better suited for applications that need to handle a high volume of concurrent requests, long-running I/O operations, or scenarios where efficient resource utilization is crucial. However, the reactive programming model in WebFlux can have a steeper learning curve compared to the traditional imperative style of Spring MVC.

# SPRING REACTIVE PERFORMANCE

## SPRING MVC VS SPRING WEBFLUX






# CONCLUSION:

In summary, while Spring Reactive may have a steeper learning curve, its performance benefits, especially in high-concurrency and I/O-bound scenarios, make it a compelling choice for building modern, scalable, and responsive web applications that require efficient handling of concurrent requests and real-time data processin