# 1) Introduction

1. Traditional model is also referred as blocking programming.
2. We will use project reactor library for reactive programming.
3. Learn operators and transform reactive streams.
4. Reactive programming + spring boot application

### 1.1) What is reactive programming?

* Reactive is different from async. They both are different.
* Reacting to something

Use case -> User events, I/O response, large applications which needs scaling

* Java used for server-side development.

### 1.2) Server-side web development

In basic traditional approach we do this

* Request comes in.
* We do processing.
* We return response.

### 1.3) Modern application development

1. High data scale
2. High usage scale
3. Cloud based costs
4. Basically we need to optimize code to save cloud costs
5. Vertical scaling -> increase the power of computer
6. Horizontal scaling -> increase the no of servers

## Problems with traditional approach

1. Idle threads and wasted hardware.
2. This makes application slow and high cost for cloud servers
3. Users will face slowness isssue
4. We code like single request in springboot. (each controller can handle multiple requests but we code for single spring takes care of it)
5. We have concurrency in java-8 already before reactive API (Future and completableFuture)

## Using completableFuture (Reactive program has better approach than this)

* Call userService with userId
* Call userProfileService in parallel
* When both return, merge the dataStructures
* Return merged object

Text

Description automatically generated

problems

* bothFutures.join() is problem here as we are blocking and thread is waiting
* -> Error handling is messy
* Its still sync
* We need a framework needs to support

# Glimpse of reactive programming and difference

Text

Description automatically generated

### advantages

* Simpler than future concurrent way in java8
* Reusable flexible functions
* Combine and reuse in powerful ways
* Benefits for large projects

### Reactive programming

* Different wa of thinking about flow.
* Different way of thinking about data
* Integrated with java! Flow interface (many lib available for this)
* Not worth it for small scale projects
* Familiar yet different
* Collections streams

## Java Streams

* Represent a sequence of data
* Focus on computations
* Vs collections which focus on storage
* Internal iteration

Stream operators

* Map
* Filter
* flatMap
* findFirst… etc