1)Reactive communication.

import java.util.List;

import reactor.core.publisher.Flux;

public class SequentialStreamReactive {

    public static final List<Integer> paralleltimeline = List.of(1,2,3,4,5,6,7,8);

    public static final List<Integer> sequentialtimeline = List.of(1,2,3,4,5,6,7,8);

    public static void main(String[] args) {

        // Sequential reactive stream

        Flux<Integer> sequentialFlux = Flux.fromIterable(sequentialtimeline)

                                           .map(i -> i \* 10)

                                           .doOnNext(i -> System.out.println("Sequential emits: " + i));

        // Parallel reactive stream

        Flux<Integer> parallelFlux = Flux.fromIterable(paralleltimeline)

                                         .parallel()

                                         .runOn(reactor.core.scheduler.Schedulers.parallel())

                                         .map(i -> i \* 10)

                                         .doOnNext(i -> System.out.println("Parallel emits: " + i))

                                         .sequential();

        // Reactive communication — merge both streams

        Flux.merge(sequentialFlux, parallelFlux)

            .subscribe(i -> System.out.println("Receiver gets: " + i),

                       err -> System.err.println("Error: " + err),

                       () -> System.out.println("All data received!"));

    }

}

Output:

Sequential emits: 10

Parallel emits: 10

Receiver gets: 10

Parallel emits: 20

Sequential emits: 20

Receiver gets: 2040

20

80

Sequential Stream

20

40

60

80