# PDP 18-05

# Reattività

## Reactive Extensions

Riferimento: Foreword | Introduction to Rx.NET (introtorx.com)

Motivazioni:

- Protocollo esplicito per gestire gli stream
- Gestione asincrona su degli Observable

## Reactive Manifesto

Il *Reactive Manifesto* definisce le caratteristiche dei sistemi *reattivi*:

- Pronti alla risposta (*Responsive*)
  - Bassa latenza
- Resilienti (*Resilient*)
  - Gestione fallimenti
- Elastici (*Elastic*)
  - Quantità variabile di risorse
- Orientati ai messaggi (*Message Driven*)
  - Primitiva di comunicazione

**Fast Data**: dati di dimensioni paragonabili al Big Data, in arrivo continuo; impensabile e/o inutile persisterli per elaborarli a partire dal supporto di salvataggio (stateless).

- Il componente più lento diventerà il collo di bottiglia e stabilirà la velocità massima dell'elaborazione
- *Back-pressure*: la resistenza che il componente successivo può opporre ai dati provenienti dal componente precedente della catena di elaborazione

## Reactive Streams

"The main goal of Reactive Streams is to govern the exchange of stream data across an asynchronous boundary while ensuring that the receiving side is not forced to buffer

arbitrary amounts of data"

- Back-pressure: impedisce che un nodo sia sovraccaricato
- API asincrone/protocolli di comunicazioni per vari linguaggi

```
@Test
public void whenSubscribeToIt_thenShouldConsumeAll()
 throws InterruptedException {

    // given
    SubmissionPublisher<String> publisher = new SubmissionPublisher<>();
    EndSubscriber<String> subscriber = new EndSubscriber<>();
    publisher.subscribe(subscriber);
    List<String> items = List.of("1", "x", "2", "x", "3", "x");

    // when
    assertThat(publisher.getNumberOfSubscribers()).isEqualTo(1);
    items.forEach(publisher::submit);
    publisher.close();

    // then
     await().atMost(1000, TimeUnit.MILLISECONDS)
       .until(
         () -> assertThat(subscriber.consumedElements)
         .containsExactlyElementsOf(items)
     );
}
```

La *back-pressure* infatti è necessaria non solo fra thread, ma anche fra nodi di calcolo distribuito.

Framework:

- RxJava
- Akka Streams
- Flow in Java 9

Il modello concettuale di Reactive Streams è compatto tanto quanto quello di ReactiveX:

- Publisher
- Subscriber
- Subscription

- Processor

## Operatori

- Map
- FlatMap
- Filter
- Skip
- Zip
- Debounce
- Window

Operatori comuni: map, flatMap (concatena risultati), filter, skip, zip (combina coppie di flussi), debounce (emette se passa tempo dall'ultimo), window (partiziona il flusso).

I Virtual Threads nascono quindi come contrapposizione diretta al paradigma reattivo.

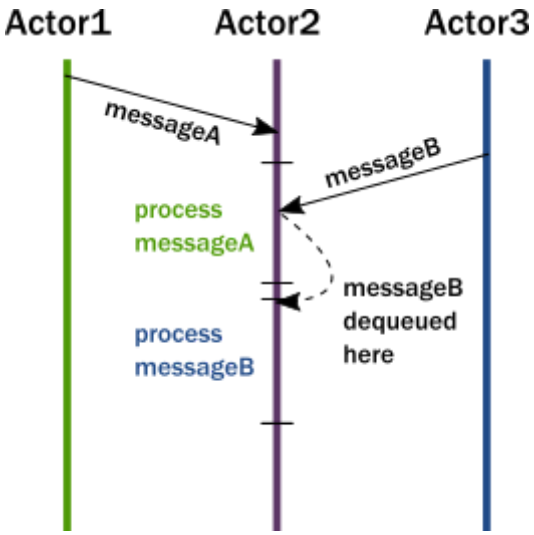
- It were introduced in Java via Project Loom as an alternative solution for parallel processing. They are lightweight, user-mode threads managed by the Java Virtual Machine (JVM)

# Attori

Un Attore è una unità indipendente di elaborazione, dotata di uno stato privato, inaccessibile dall'esterno.

Nel reagire ad un messaggio, un attore può:

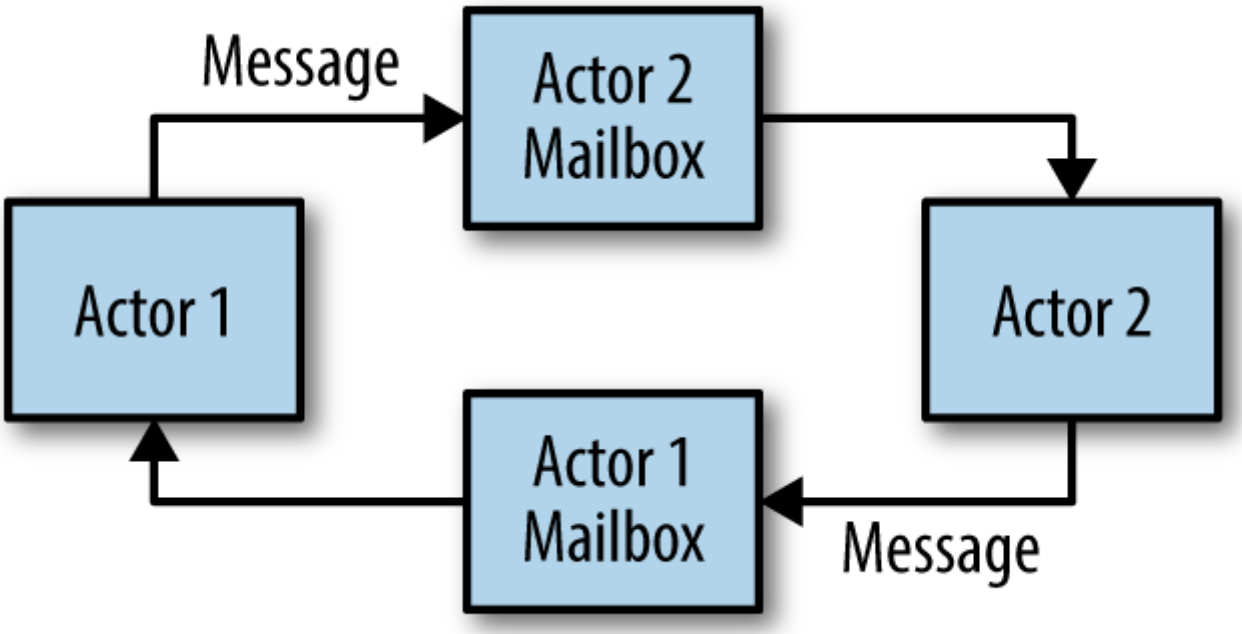
- mutare il proprio stato interno
- creare nuovi attori inviare
- messaggi ad attori noti
- cambiare il suo comportamento




Esecuzione prettamente sequenziale. Al di fuori dell'attore tutto è concorrente e distribuito:

- un altro attore può trovarsi dovunque
- ogni attore è concorrente a tutti gli altri
- ogni messaggio è concorrente a tutti gli altri

Message-driven: messaggi/notifiche (Observer)

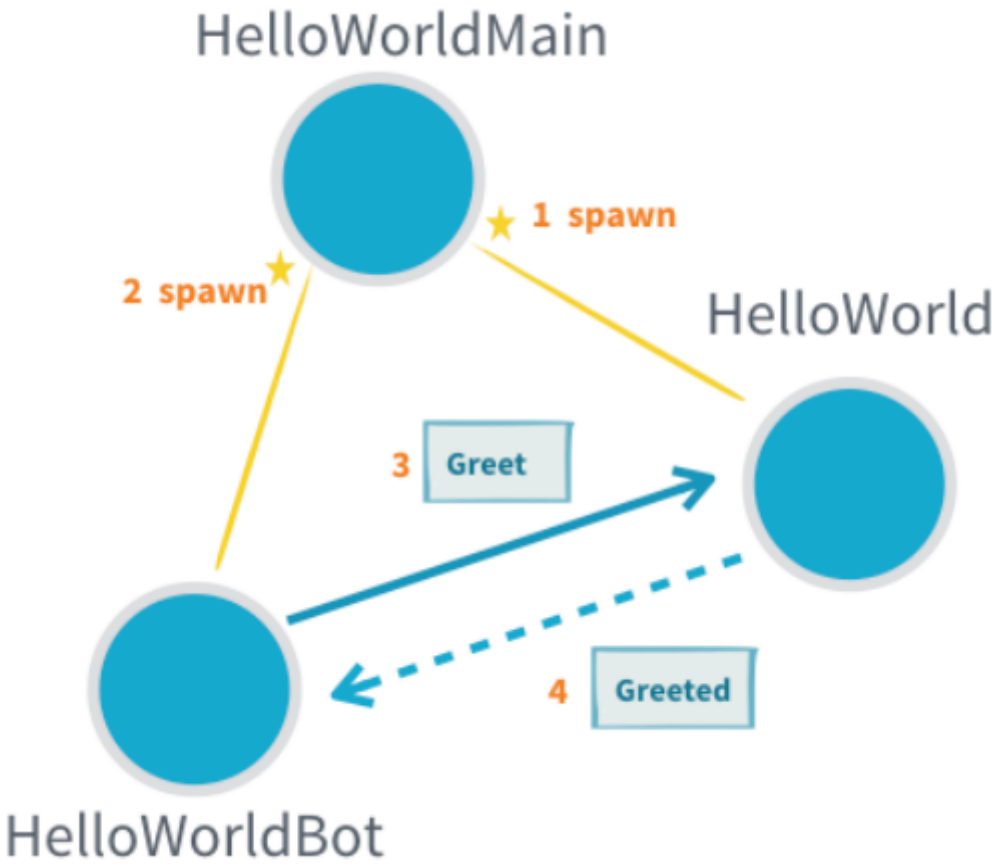


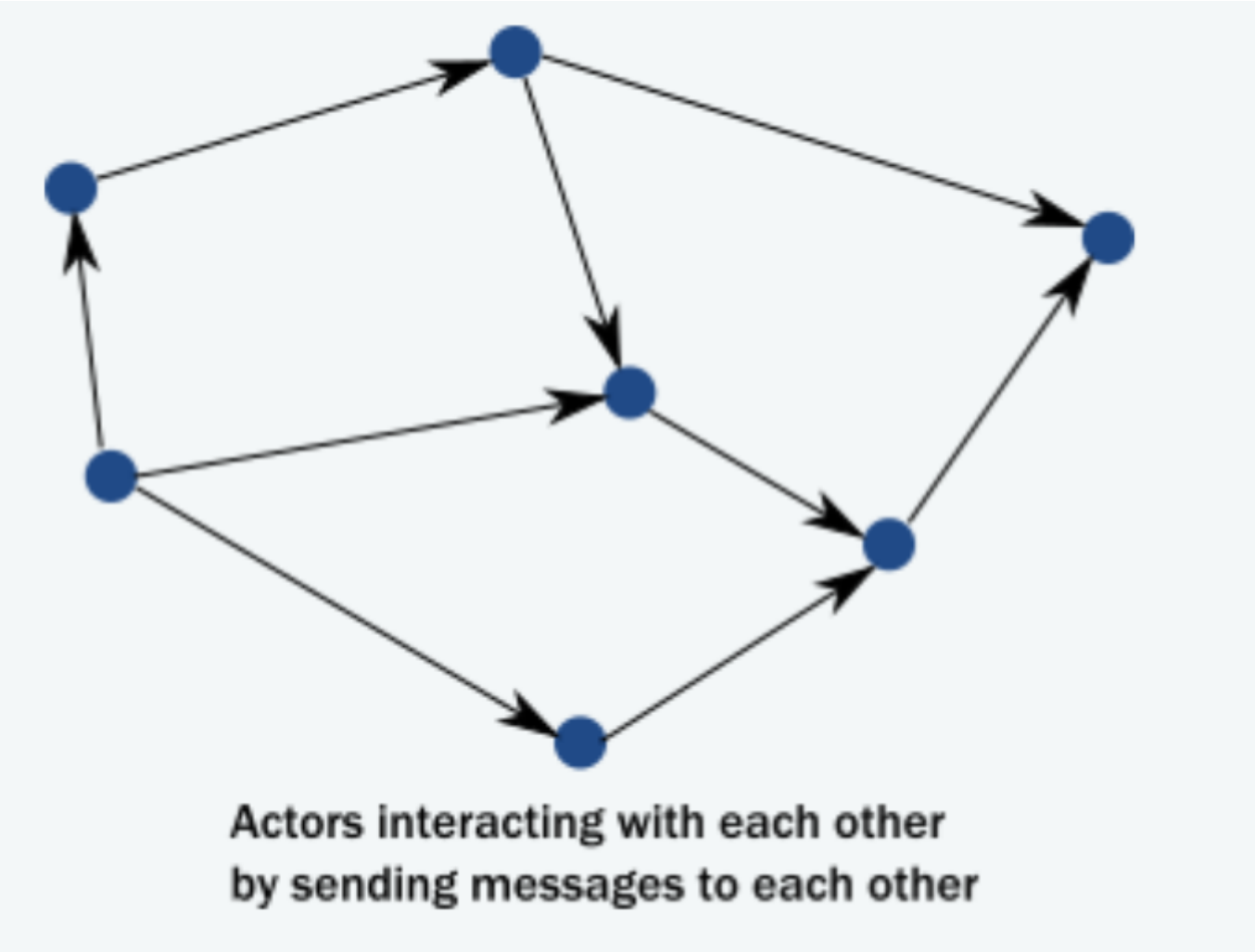

- Approccio economico
- Affidabilità elevata
- Scalabilità lineare

Problema principale: Modello principalmente teorico

Esempi di utilizzo:

- Akka Stream
  - Adotta il modello di Erlang per implementate sulla JVM un modello reattivo e scalabile






# Java 23

JEP - JDK Enhancement Proposal

Utili per noi:

- Stream gatherers, previously previewed in JDK 22, would enhance the stream API to support custom intermediate operations.
    - They would allow stream pipelines to transform data in ways not easily achievable with the existing built-in intermediate operations.

Principalmente:

- Potential features include further advancements in pattern matching, value-based classes, and performance enhancements

Riferimento: JDK 22 and JDK 23: What We Know So Far - InfoQ