Reactive Programming & Java Completable Futures

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

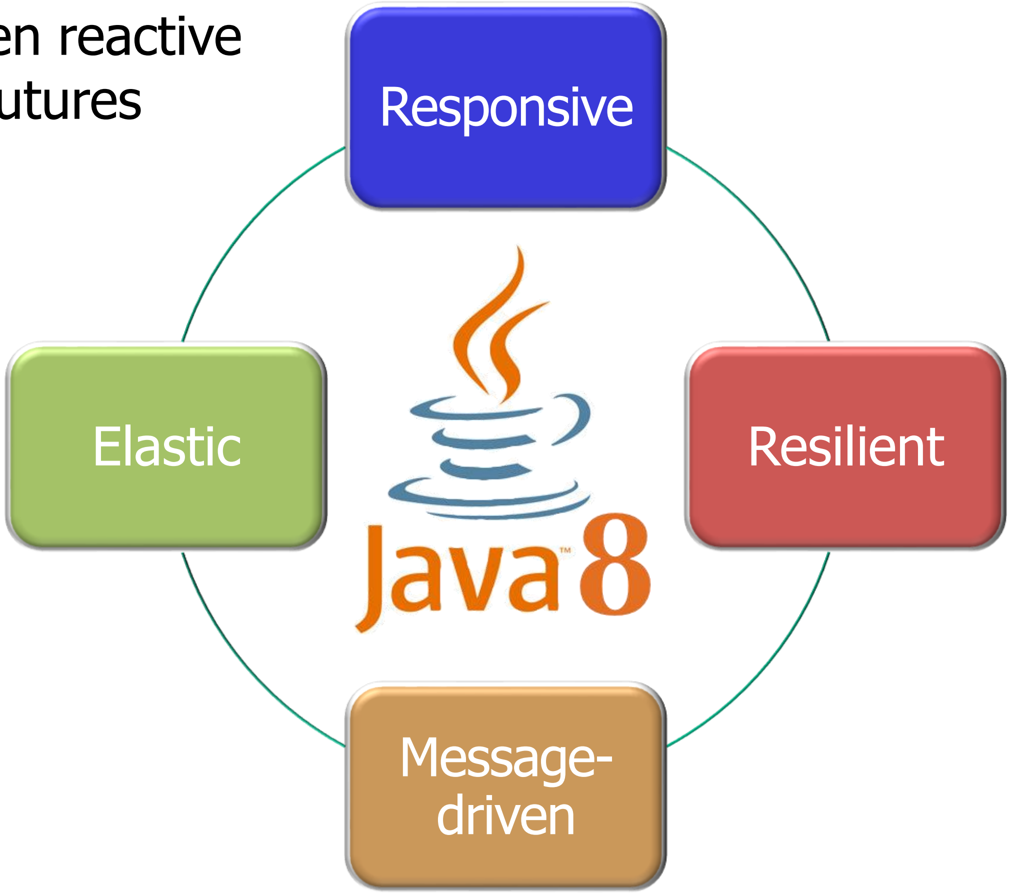
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand the relationship between reactive programming & Java completable futures



Overview of Reactive Programming

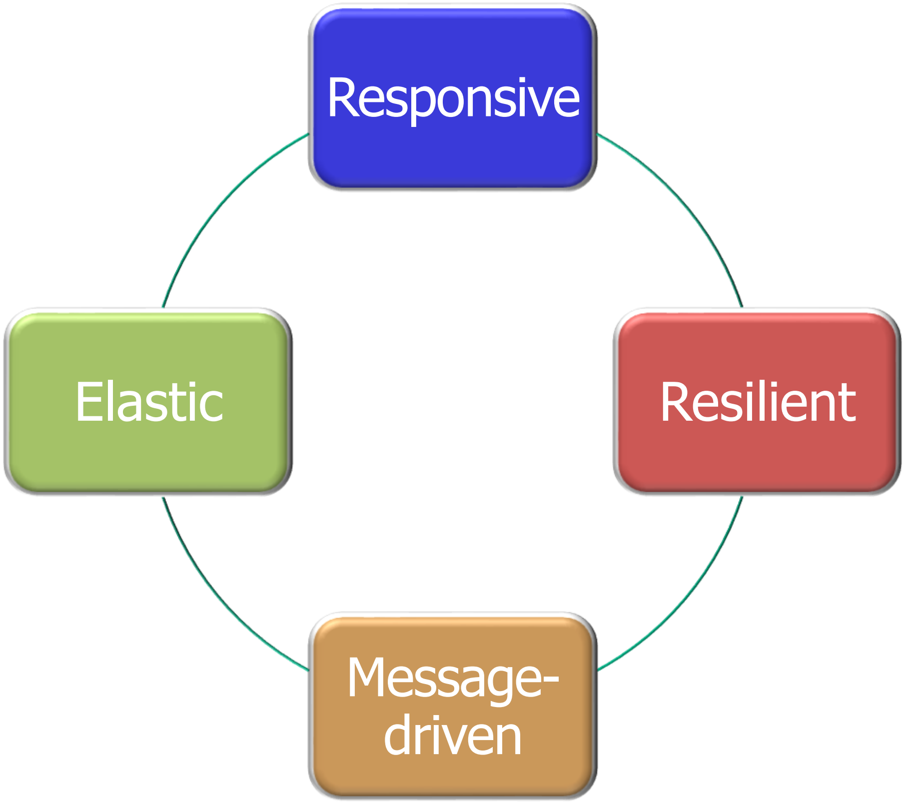
Overview of Reactive Programming

- Reactive programming is an asynchronous programming paradigm concerned with processing data streams & propagation of changes

See en.wikipedia.org/wiki/Reactive_programming

Overview of Reactive Programming

- Reactive programming is based on four key principles



See www.reactivemanifesto.org

Overview of Reactive Programming

- Reactive programming is based on four key principles, e.g.
  - **Responsive**
    - Provide rapid & consistent response times

Establish reliable upper bounds to deliver consistent quality of service & prevent delays

See en.wikipedia.org/wiki/Responsiveness

Overview of Reactive Programming

- Reactive programming is based on four key principles, e.g.
  - Responsive
  - **Resilient**
    - The system remains responsive, even in the face of failure

Failure of some operations should not bring the entire system down

See en.wikipedia.org/wiki/Resilience_(network)

Overview of Reactive Programming

- Reactive programming is based on four key principles, e.g.
  - Responsive
  - Resilient
  - **Elastic**
    - A system should remain responsive, even under varying workload

It should be possible to "auto-scale" performance

See en.wikipedia.org/wiki/Autoscaling

Overview of Reactive Programming

- Reactive programming is based on four key principles, e.g.
  - Responsive
  - Resilient
  - Elastic
  - **Message-driven**
    - Asynchronous message-passing ensures loose coupling, isolation, & location transparency between components

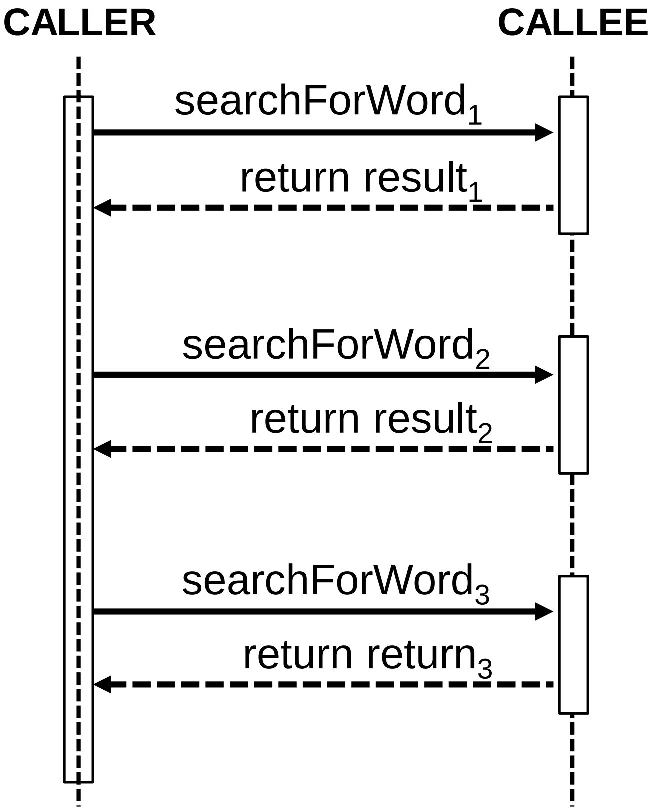
This principle is an "implementation detail" wrt the others..

See en.wikipedia.org/wiki/Message-oriented_middleware

Reactive Programming & Java Completable Futures

Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - **Responsive**
    - Avoid blocking in user code
      - Blocking underutilizes cores, impedes inherent parallelism, & complicates program structure

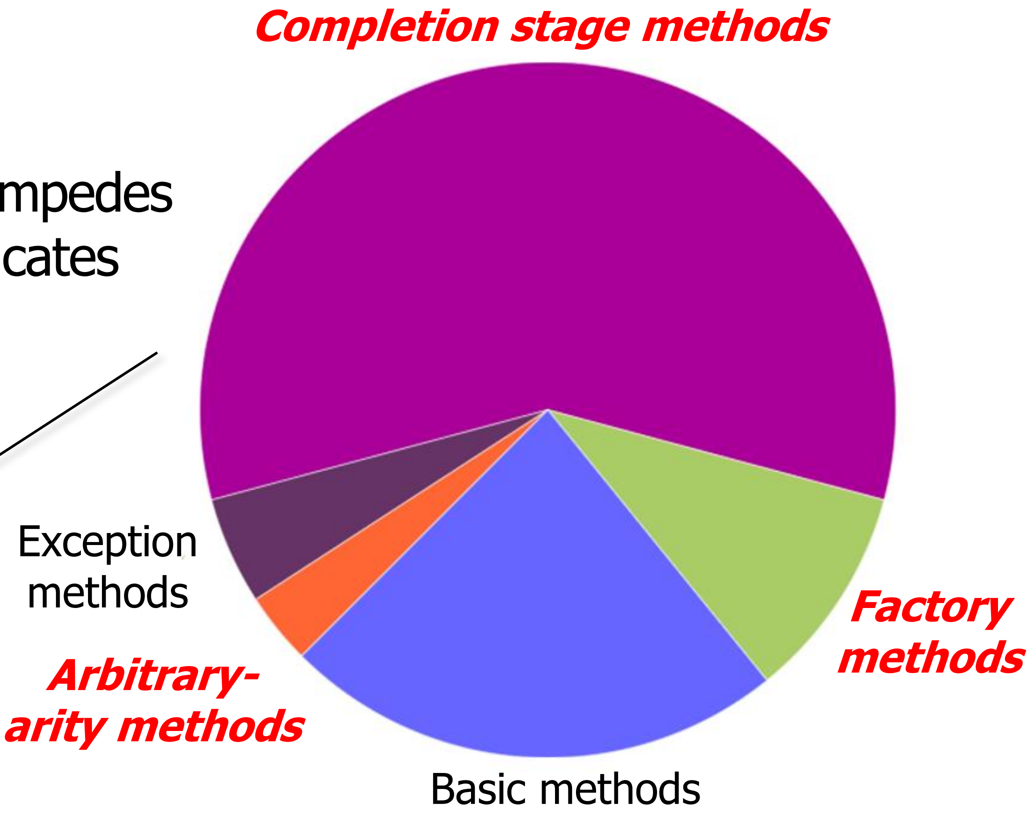


See www.ibm.com/developerworks/library/j-jvmc3

Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - **Responsive**
    - Avoid blocking in user code
      - Blocking underutilizes cores, impedes inherent parallelism, & complicates program structure

Factory, completion stage, & arbitrary-arity methods avoid blocking threads



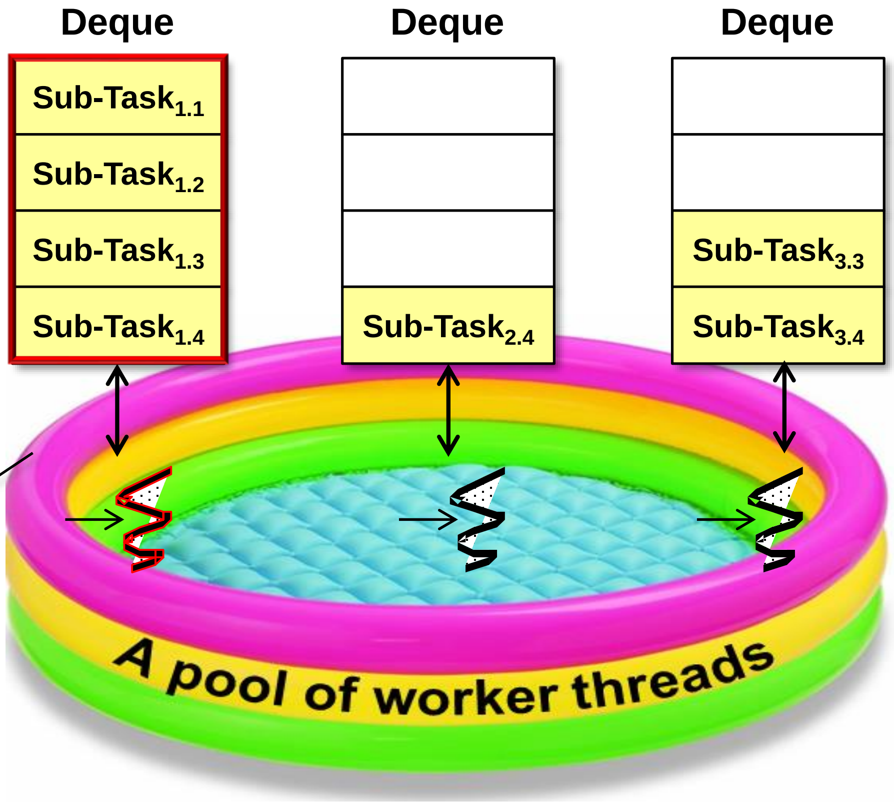
Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - **Responsive**
    - Avoid blocking in user code
    - Avoid changing threads
      - Incurs excessive overhead wrt synchronization, context switching, & memory/cache management

See gee.cs.oswego.edu/dl/papers/fj.pdf

Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - **Responsive**
    - Avoid blocking in user code
    - Avoid changing threads
      - Incurs excessive overhead wrt synchronization, context switching, & memory/cache management

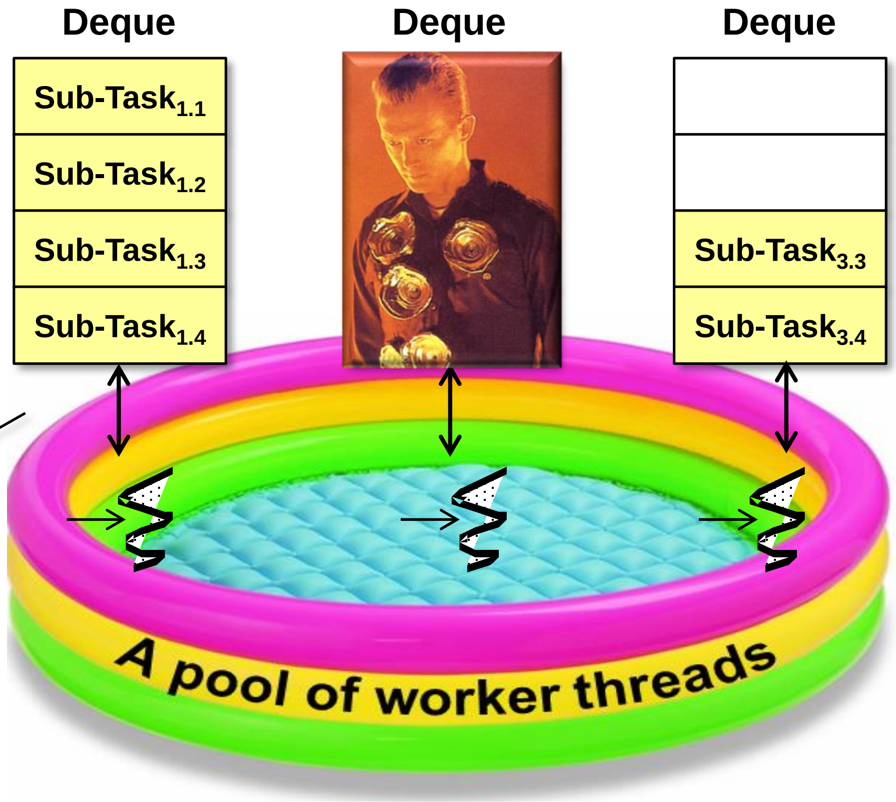


*The fork-join pool & non-*Async() methods avoid changing threads*

See gee.cs.oswego.edu/dl/papers/fj.pdf

Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - Responsive
  - **Resilient**
    - Exception methods make more programs resilient to failures

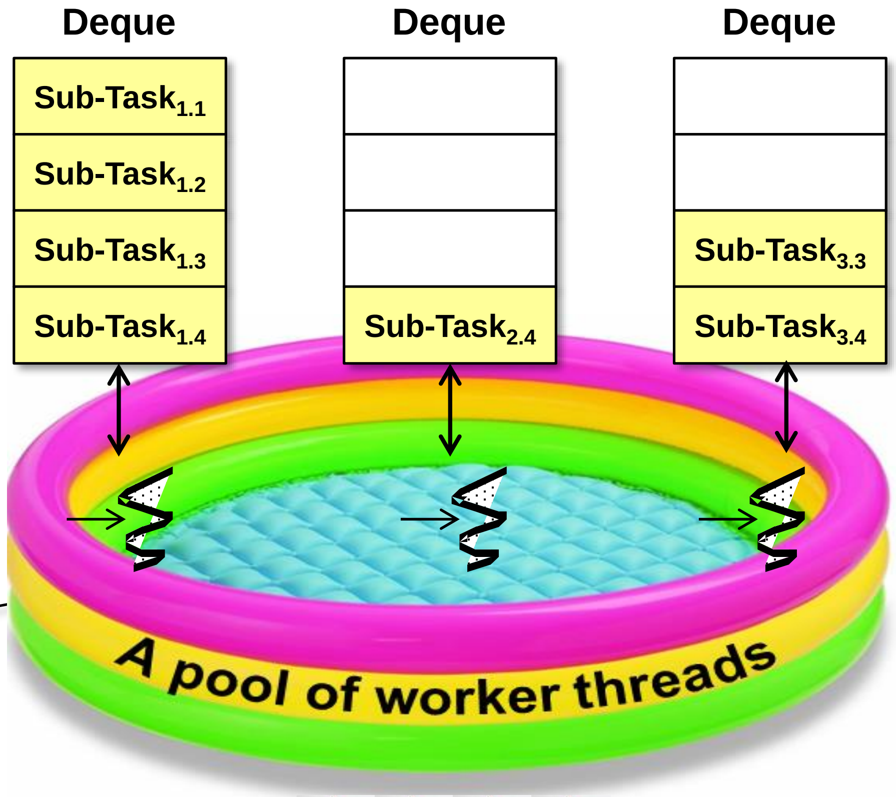


Exceptions decouple error processing from normal operations

However, completable futures are localized to a single process, *not* a cluster!

Reactive Programming & Java Completable Futures

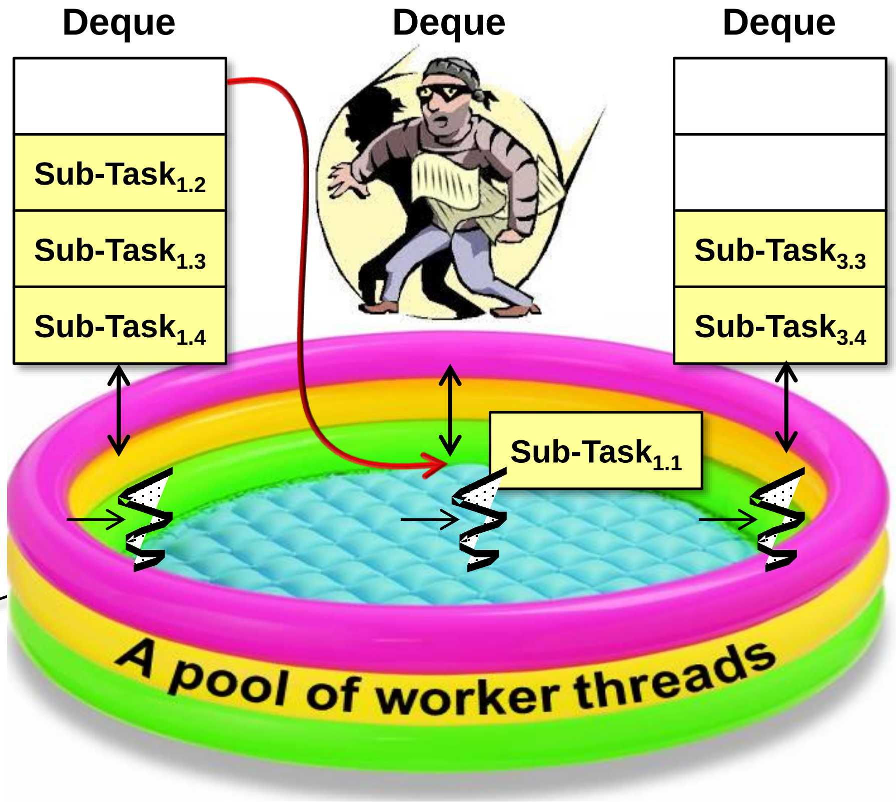
- Java completable futures map onto key reactive programming principles, e.g.
  - Responsive
  - Resilient
  - **Elastic**
    - Async computations can run scalably in a pool of threads atop a set of cores



Can be a (common) fork-join pool or a custom thread pool

Reactive Programming & Java Completable Futures

- Java completable futures map onto key reactive programming principles, e.g.
  - Responsive
  - Resilient
  - Elastic
  - **Message-driven**
    - The Java fork-join pool passes messages between threads in the pool internally



Java's fork-join pool implements "work-stealing" between deques

See en.wikipedia.org/wiki/Work_stealing

Reactive Programming & Java Reactive Streams

Reactive Programming & Java Reactive Streams

- Java 9 support reactive programming via "Reactive Streams" & the Flow API

Class Flow

java.lang.Object
    java.util.concurrent.Flow

```
public final class Flow
extends Object
```

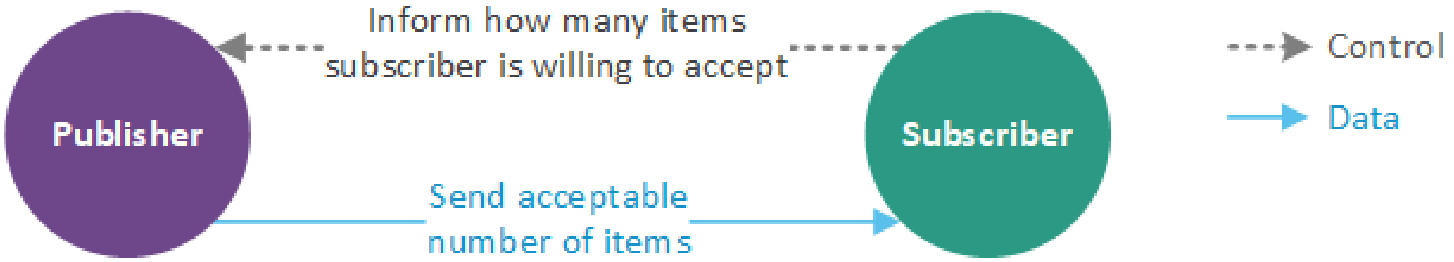
Interrelated interfaces and static methods for establishing flow-controlled components in which `Publishers` produce items consumed by one or more `Subscribers`, each managed by a `Subscription`.

These interfaces correspond to the reactive-streams specification. They apply in both concurrent and distributed asynchronous settings: All (seven) methods are defined in `void` "one-way" message style. Communication relies on a simple form of flow control (method `Flow.Subscription.request(long)`) that can be used to avoid resource management problems that may otherwise occur in "push" based systems.

See community.oracle.com/docs/DOC-1006738

Reactive Programming & Java Reactive Streams

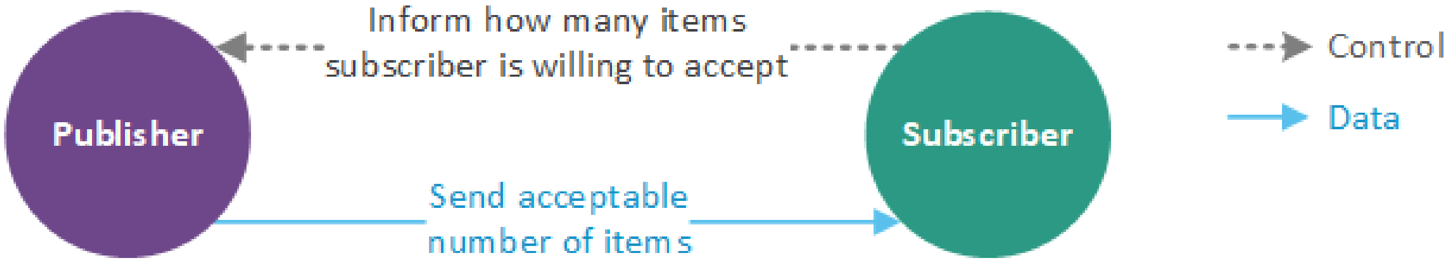
- Java 9 support reactive programming via "Reactive Streams" & the Flow API
  - Adds support for stream-oriented pub/sub patterns



See javasampleapproach.com/java/java-9/java-9-flow-api-example-publisher-and-subscriber

Reactive Programming & Java Reactive Streams

- Java 9 support reactive programming via "Reactive Streams" & the Flow API
  - Adds support for stream-oriented pub/sub patterns

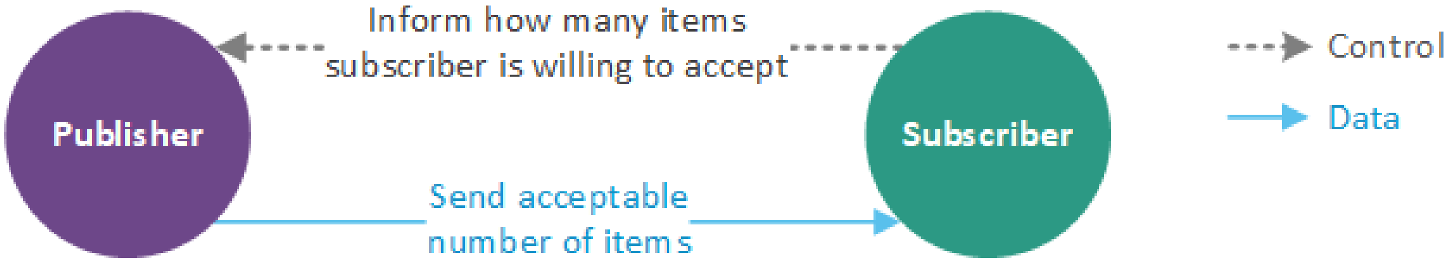


  - Combines two patterns
    - *Iterator*, which applies a pull model where apps pulls items from a source
    - *Observer*, which applies a push model that reacts when item is pushed from a source to a subscriber

See www.journaldev.com/20723/java-9-reactive-streams

Reactive Programming & Java Reactive Streams

- Java 9 support reactive programming via "Reactive Streams" & the Flow API
  - Adds support for stream-oriented pub/sub patterns

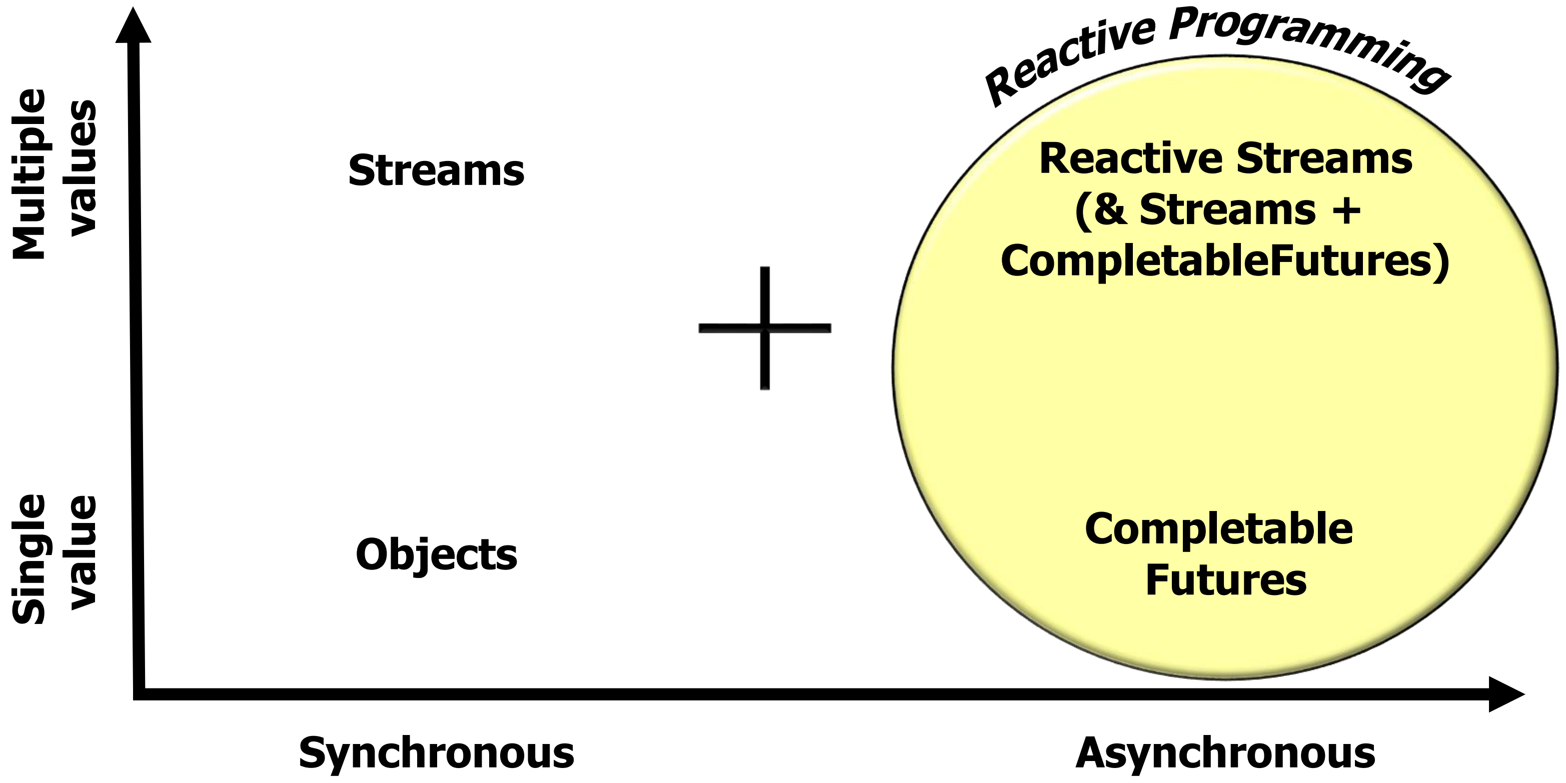


  - Combines two patterns
  - Intended as an interoperable foundation for other reactive programming frameworks

See www.baeldung.com/java-9-reactive-streams

Reactive Programming & Java Reactive Streams

- Comparing reactive programming with other Java programming paradigms



End of Reactive Programming & Java Completable Futures