Understanding Key Classes in the Project Reactor API

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

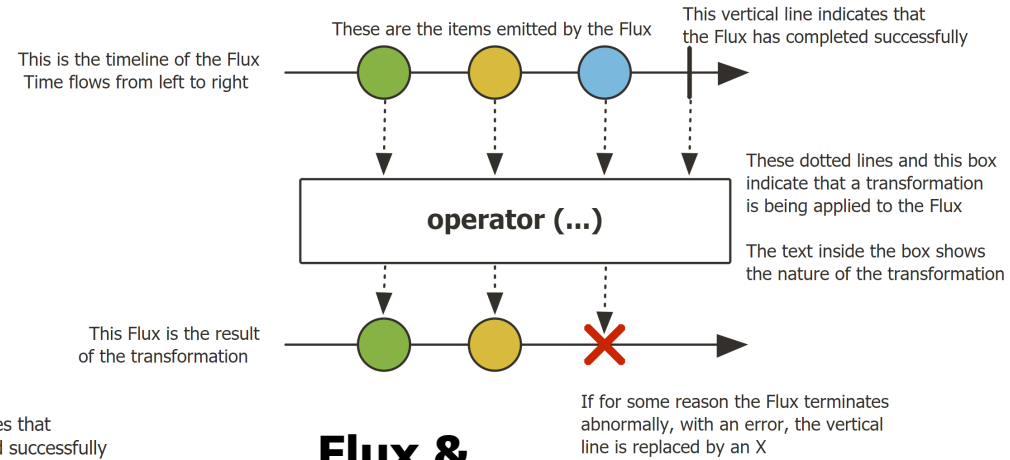
Institute for Software Integrated Systems

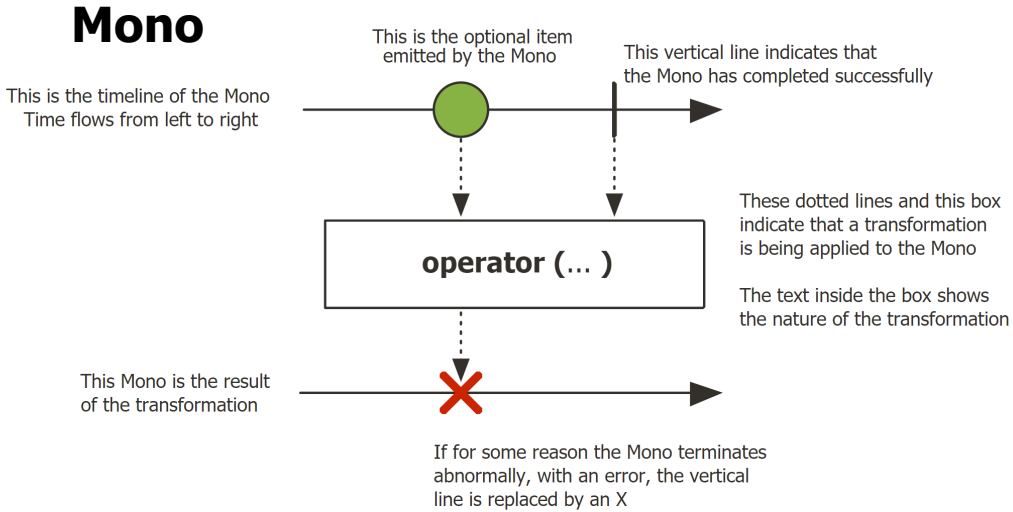
Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand key classes in the Project Reactor API





Key Classes in the Project Reactor API

Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API

Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value

Class Mono<T>

java.lang.Object
    reactor.core.publisher.Mono<T>

Type Parameters:

T - the type of the single value of this class

All Implemented Interfaces:

Publisher<T>, CorePublisher<T>

Direct Known Subclasses:

MonoOperator, MonoProcessor

```
public abstract class Mono<T>
extends Object
implements CorePublisher<T>
```

A Reactive Streams `Publisher` with basic rx operators that completes successfully by emitting an element, or with an error.

The recommended way to learn about the `Mono` API and discover new operators is through the reference documentation, rather than through this javadoc (as opposed to learning more about individual operators). See the "which operator do I need?" appendix.

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html

Key Classes in the Project Reactor API

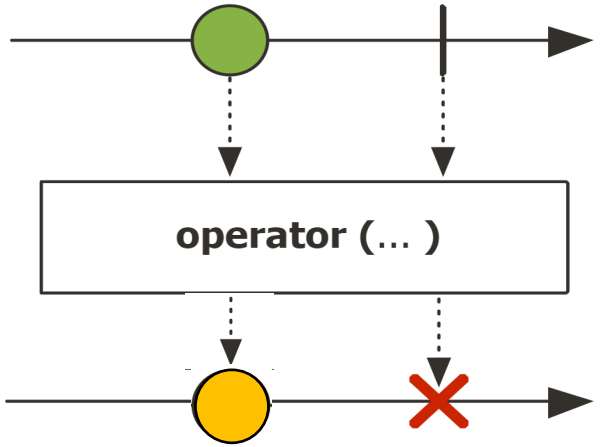
- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>

```
BigFraction unreducedFraction =
  makeBigFraction(...);

Mono
   .fromCallable(() -> BigFraction
        .reduce(unreducedFraction))

   .subscribeOn
       (Schedulers.single())

   .map(ibf -> ibf.toMixedString())

   .doOnSuccess(bf ->
      System.out.println
        ("result = " + bf + "\n"));
```

See stackoverflow.com/questions/54866391/mono-vs-completablefuture

Key Classes in the Project Reactor API

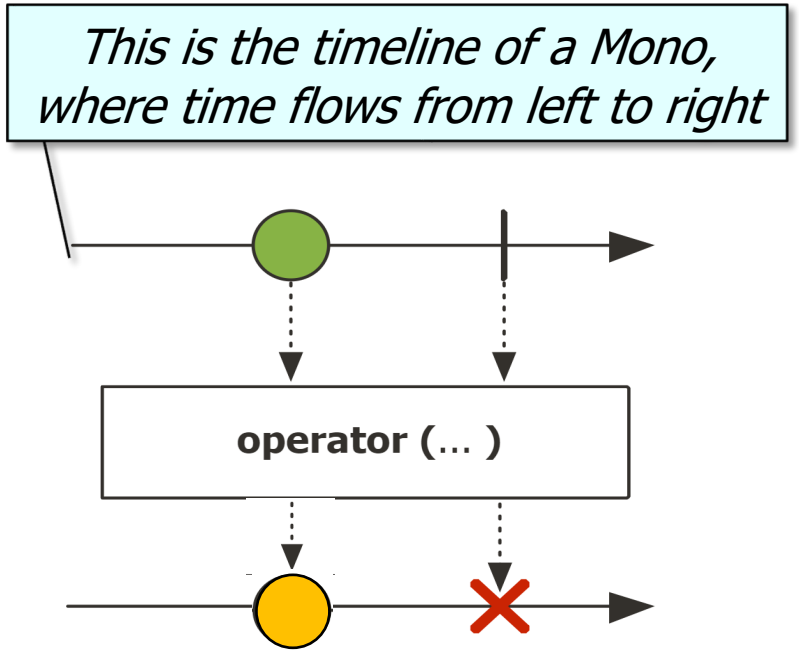
- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



See projectreactor.io/docs/core/release/reference/#howtoReadMarbles

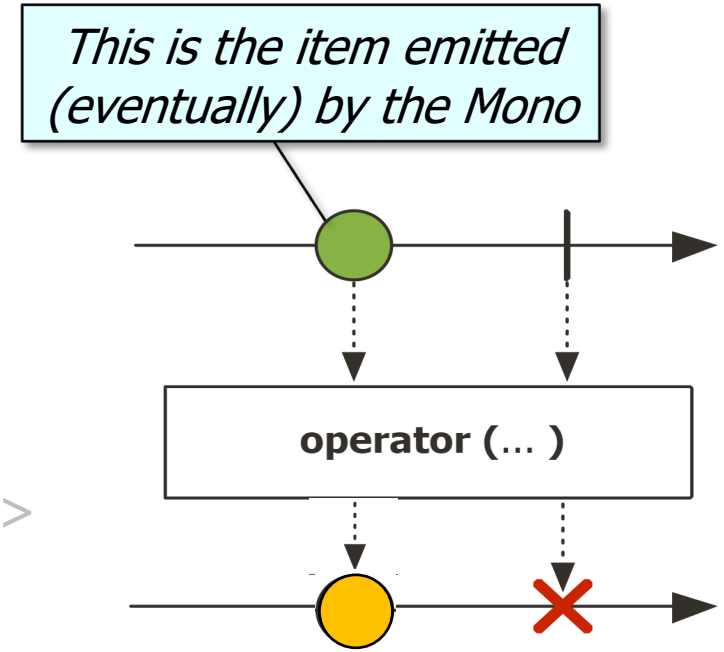
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



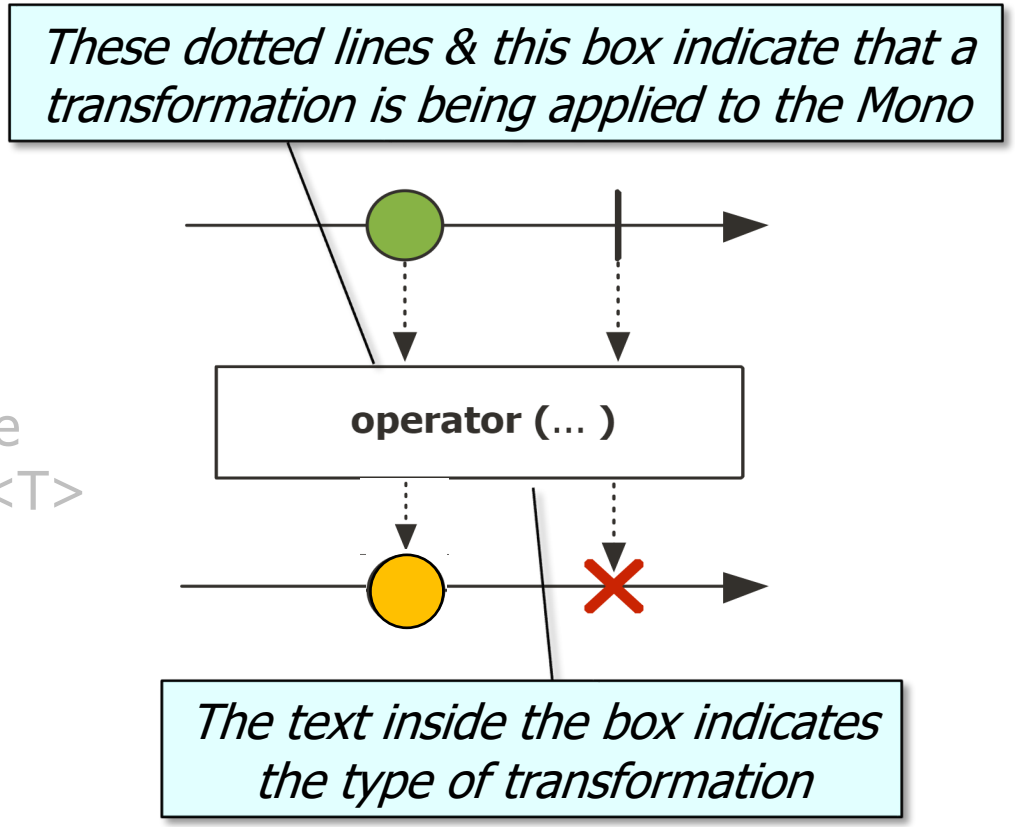
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



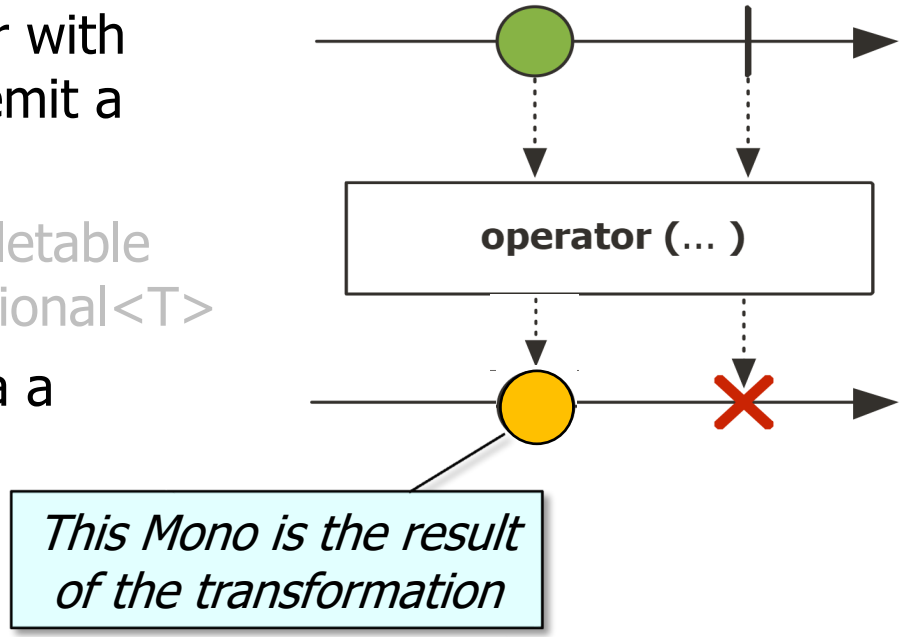
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



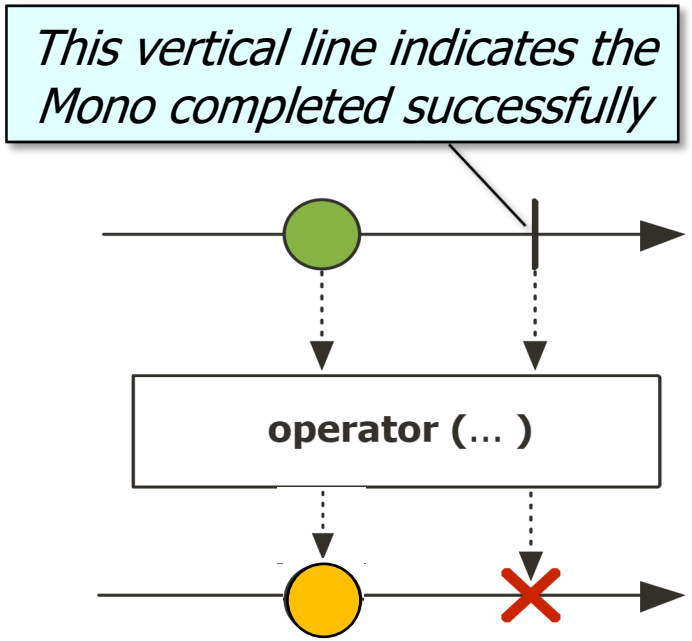
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



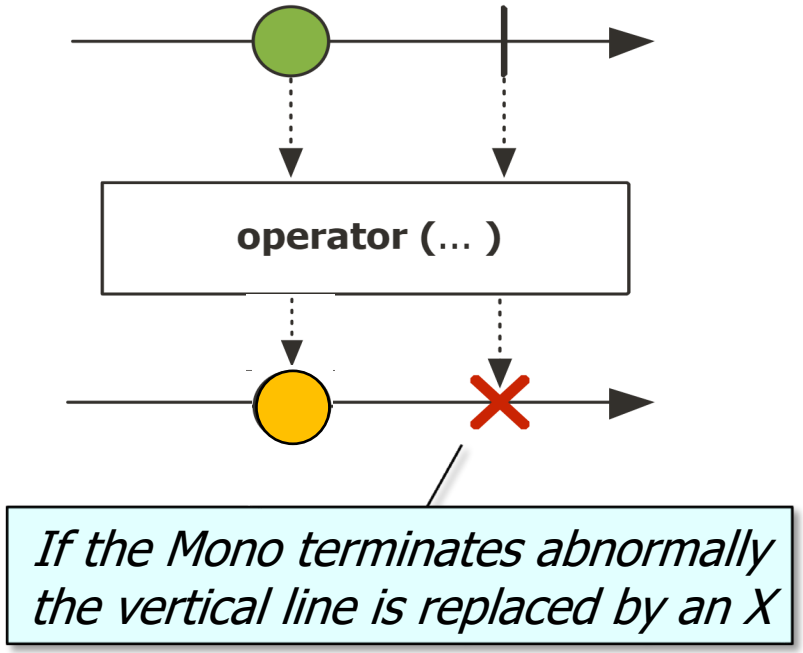
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a "marble diagram"



Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
      - Can be documented via a "marble diagram"
    - Provides a wide range of operators
      - Factory method operators
      - Transforming operators
      - Action operators
      - Concurrency & scheduler operators
      - Combining operators
      - Suppressing operators
      - Blocking operators
      - etc.

Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure

Class Flux<T>

java.lang.Object
    reactor.core.publisher.Flux<T>

Type Parameters:

T - the element type of this Reactive Streams `Publisher`

All Implemented Interfaces:

Publisher<T>, CorePublisher<T>

Direct Known Subclasses:

ConnectableFlux, FluxOperator, FluxProcessor, GroupedFlux

```
public abstract class Flux<T>
extends Object
implements CorePublisher<T>
```

A Reactive Streams `Publisher` with rx operators that emits 0 to N elements, and then completes (successfully or with an error).

The recommended way to learn about the `Flux` API and discover new operators is through the reference documentation, rather than through this javadoc (as opposed to learning more about individual operators). See the "which operator do I need?" appendix.

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html

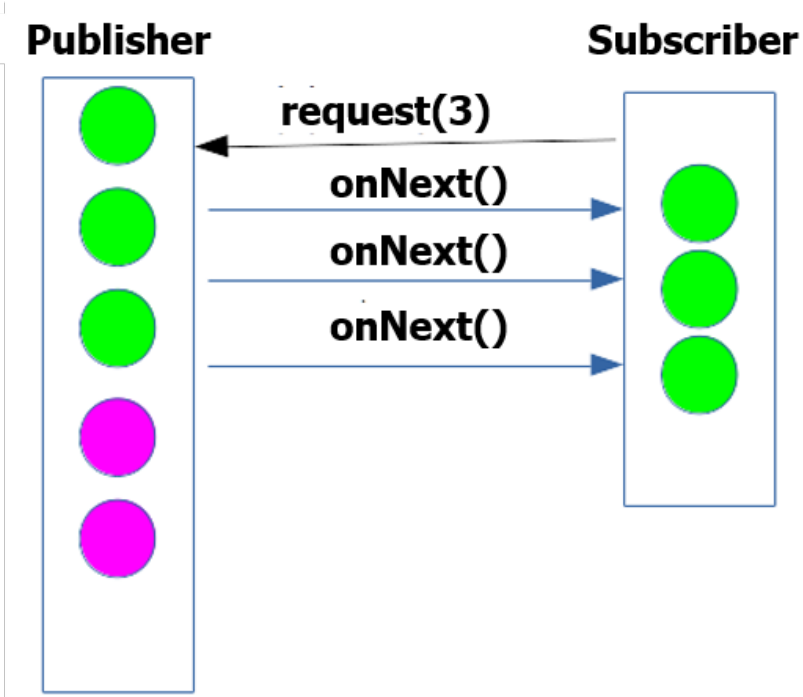
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
      - i.e., completable futures used with a Java stream

```
Flux
   .create
      (bigFractionEmitter)

   .take(sMAX_FRACTIONS)

   .flatMap(unreducedFraction ->
         reduceAndMultiplyFraction
          (unreducedFraction,
           Schedulers.parallel()))
   .collectList()

   .flatMap(list ->
            sortAndPrintList
              (list, sb));
```

Key Classes in the Project Reactor API

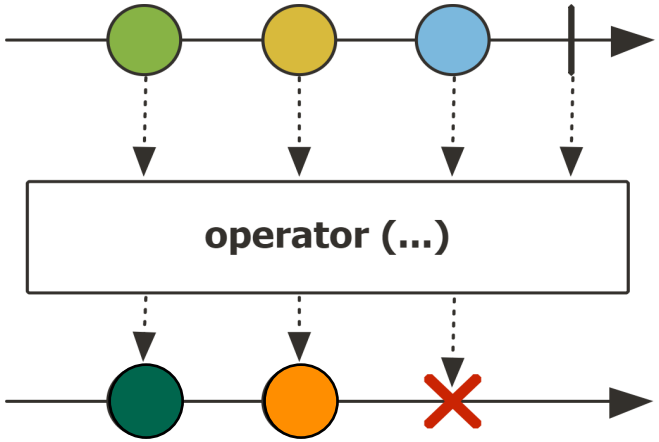
- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
      - The subscriber indicates to the publisher how much data it can consume



See jstobigdata.com/java/backpressure-in-project-reactor

Key Classes in the Project Reactor API

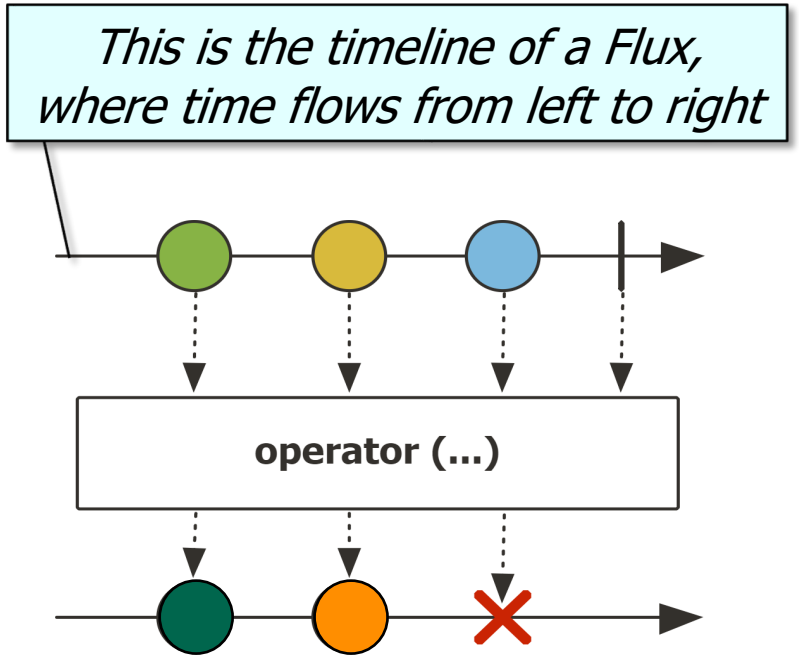
- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



See medium.com/@jshvarts/read-marble-diagrams-like-a-pro-3d72934d3ef5

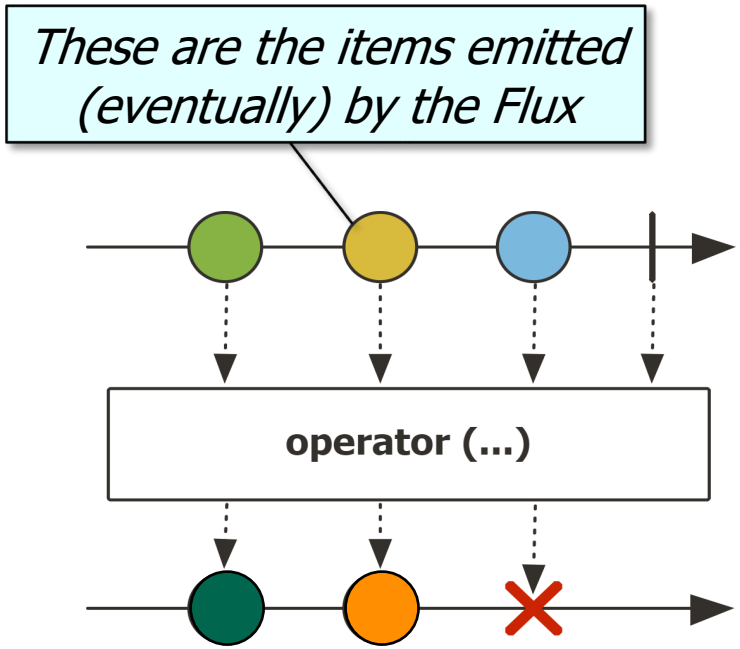
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



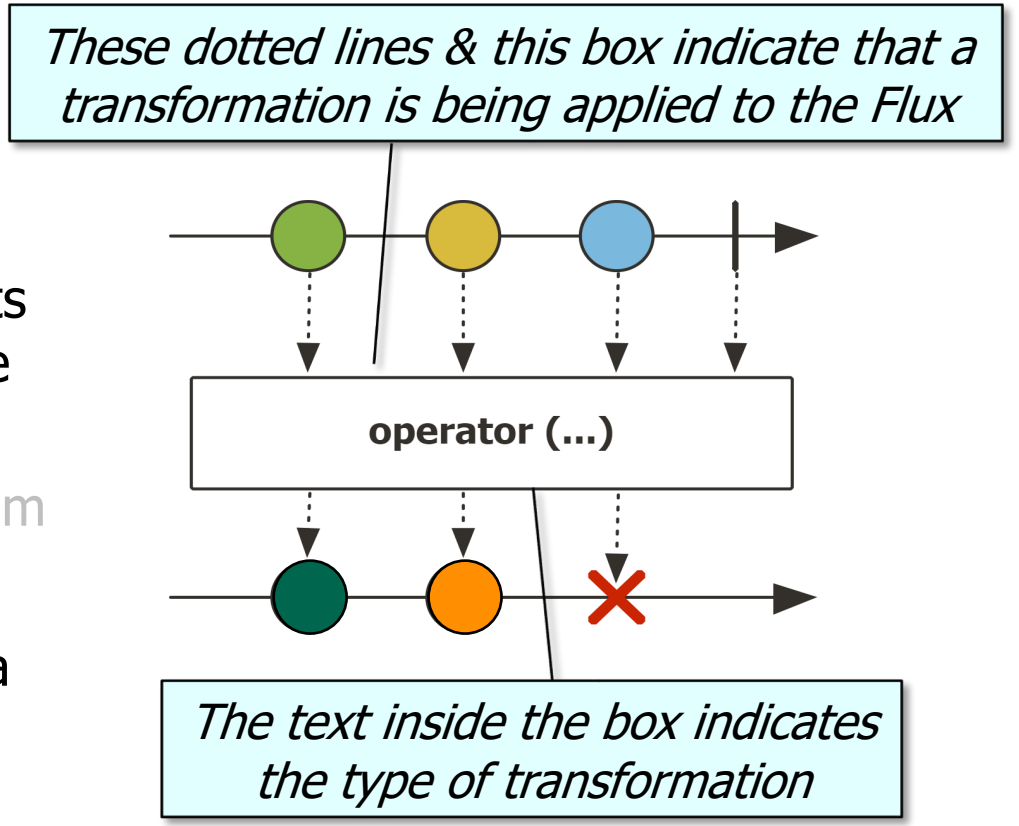
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



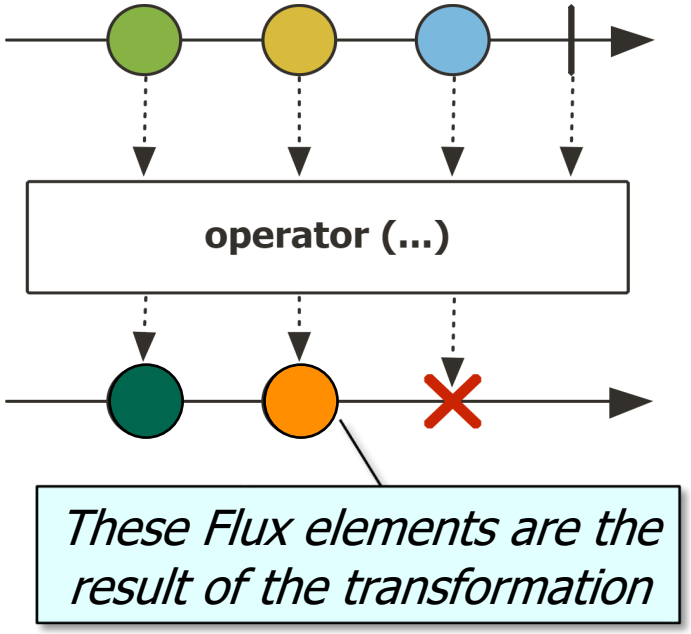
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



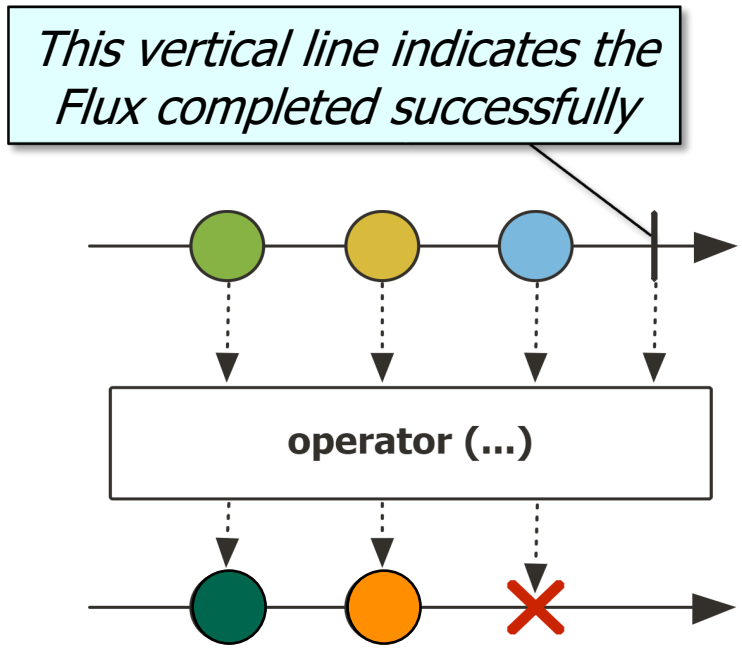
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



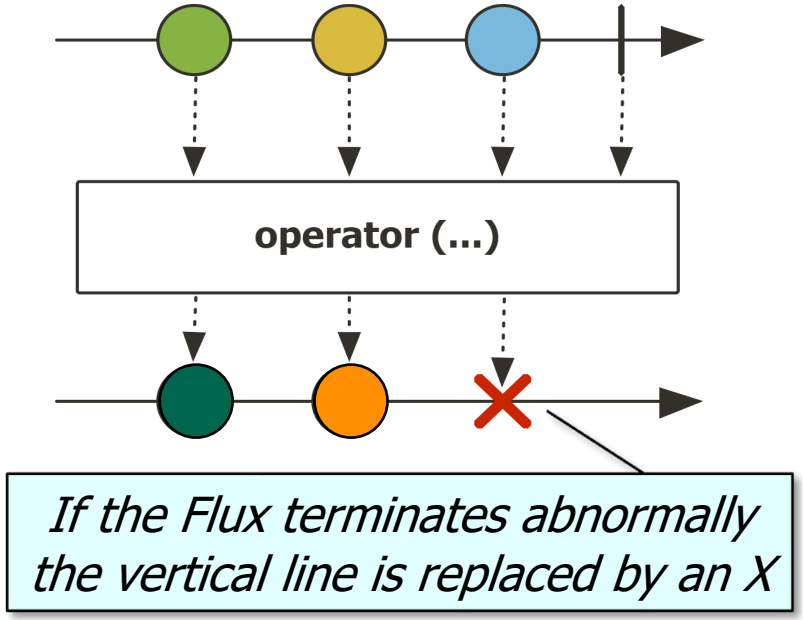
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram



Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram
    - Provides a wide range of operators
      - Factory method operators
      - Transforming operators
      - Action operators
      - Concurrency & scheduler operators
      - Combining operators
      - Terminal operators
      - Suppressing operators
      - Blocking operators
      - etc.

Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - Flux
  - **ParallelFlux**
    - Defines a subset of Flux operators that provide a concise means of processing elements in parallel

```
public abstract class ParallelFlux<T>
extends Object
implements CorePublisher<T>
```

A ParallelFlux publishes to an array of Subscribers, in parallel 'rails' (or `'groups'`).

Use `from(reactor.core.publisher.ParallelFlux<T>)` to start processing a regular Publisher in 'rails', which each cover a subset of the original Publisher's data. `Flux.parallel()` is a convenient shortcut to achieve that on a `Flux`.

Use `runOn(reactor.core.scheduler.Scheduler)` to introduce where each 'rail' should run on thread-wise.

Use `sequential()` to merge the sources back into a single `Flux`.

Use `then()` to listen for all rails termination in the produced `Mono`

See projectreactor.io/docs/core/release/api/reactor/core/publisher/ParallelFlux.html

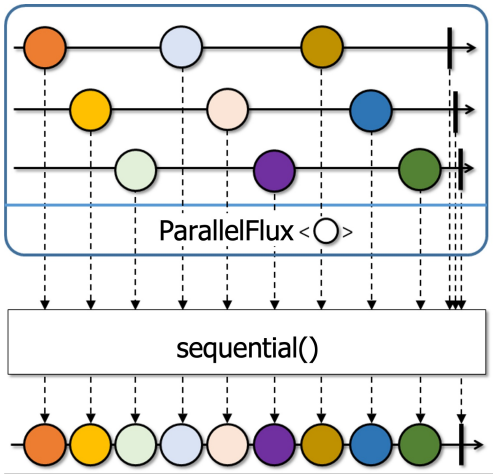
Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - Mono
  - Flux
  - **ParallelFlux**
    - Defines a subset of Flux operators that provide a concise means of processing elements in parallel
    - Operators convert Flux to Parallel Flux & vice versa

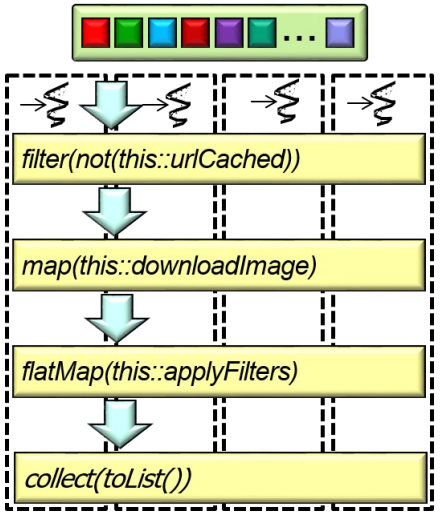
```
List<Image> imgs = Flux
   .fromIterable(Options.
      instance().getUrlList())

   .parallel(parallelism)

   .runOn(scheduler)

   .map(downloadAndStoreImage)

   .sequential()

   .collectList()

   .block();
```

Key Classes in the Project Reactor API

- There are three key classes in the Project Reactor API
  - **Mono**
  - **Flux**
  - **ParallelFlux**
    - Defines a subset of Flux operators that provide a concise means of processing elements in parallel
    - Operators convert Flux to Parallel Flux & vice versa
    - Similar in structure & functionality to a Java parallel stream



Parallel Flux



Parallel Stream

See chat.openai.com/share/c78d3a92-eced-4414-83d9-aacd86f41209

End of Understanding Key Classes in the Project Reactor API