# Understanding Key Classes in the Project Reactor API

## Douglas C. Schmidt
[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
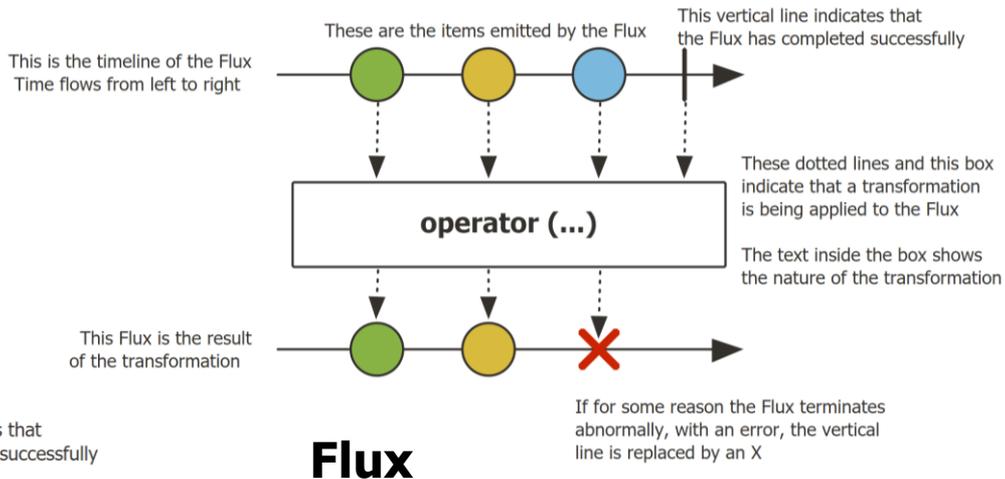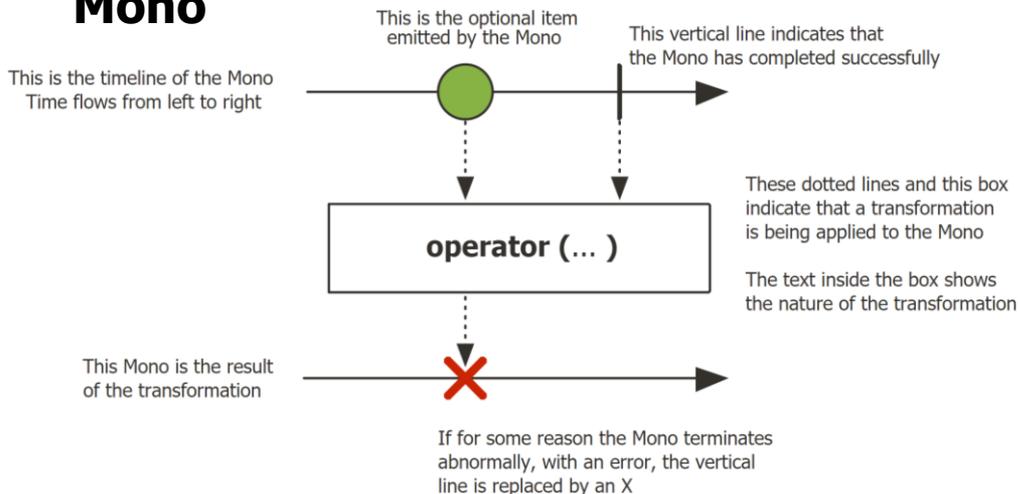
Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Lesson

- Understand key classes in the Project Reactor API

These are the items emitted by the Flux

This is the timeline of the Flux
Time flows from left to right

This vertical line indicates that the Flux has completed successfully

These dotted lines and this box indicate that a transformation is being applied to the Flux

operator (...)

The text inside the box shows the nature of the transformation

This Flux is the result of the transformation

If for some reason the Flux terminates abnormally, with an error, the vertical line is replaced by an X

**Flux**

**Mono**

This is the optional item emitted by the Mono

This vertical line indicates that the Mono has completed successfully

This is the timeline of the Mono
Time flows from left to right

These dotted lines and this box indicate that a transformation is being applied to the Mono

operator (… )

The text inside the box shows the nature of the transformation

This Mono is the result of the transformation

If for some reason the Mono terminates abnormally, with an error, the vertical line is replaced by an X

# Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API

# Key Classes in the Project Reactor API

- There are two 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 two 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(result ->
      result.toMixedString())
  .doOnSuccess(result ->
            System.out.println
              ("big fraction = "
              + result + "\n"));
```
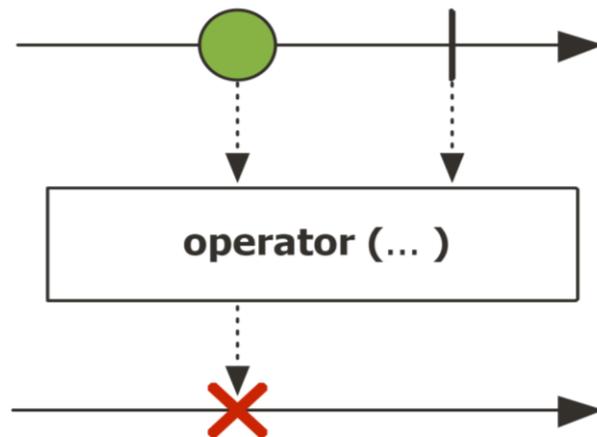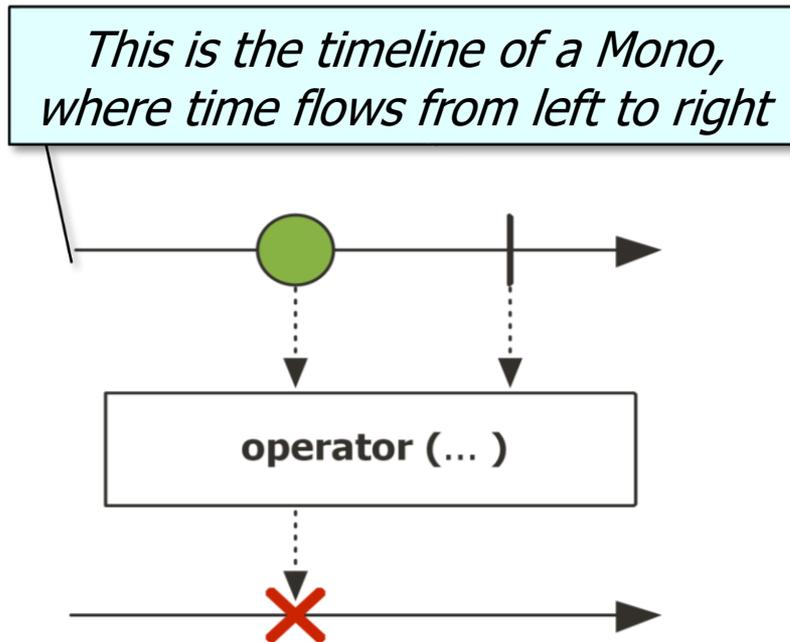
See stackoverflow.com/questions/54866391/mono-vs-completablefuture

# Key Classes in the Project Reactor API

- There are two 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"
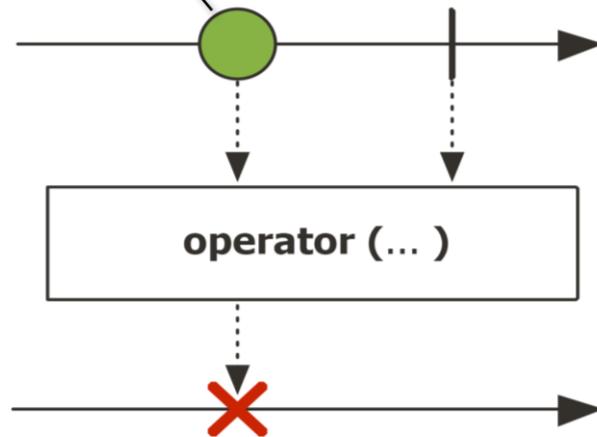
# Key Classes in the Project Reactor API

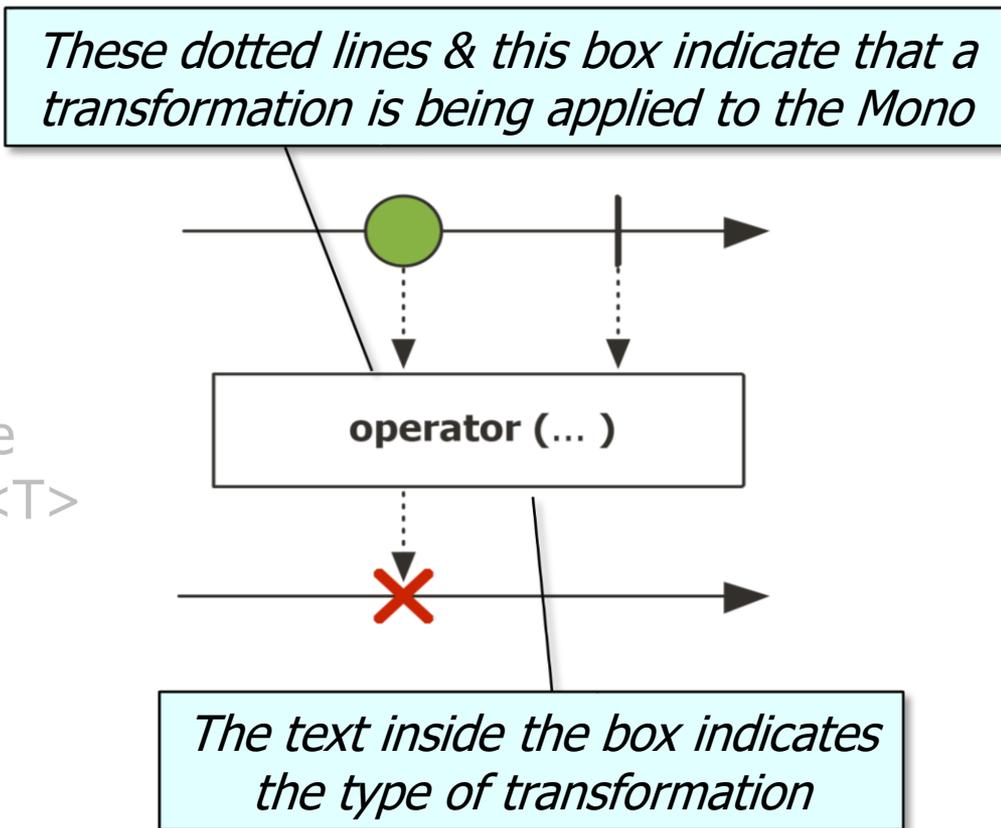- There are two key classes in the Project Reactor API

  - **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"

*This is the timeline of a Mono, where time flows from left to right*
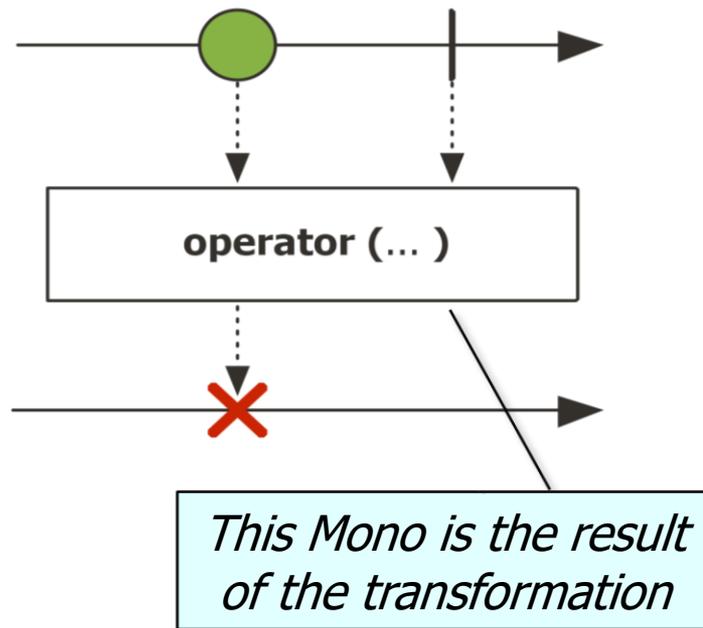
operator ( … )

# Key Classes in the Project Reactor API

- There are two 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"

This is the optional item emitted by the Mono
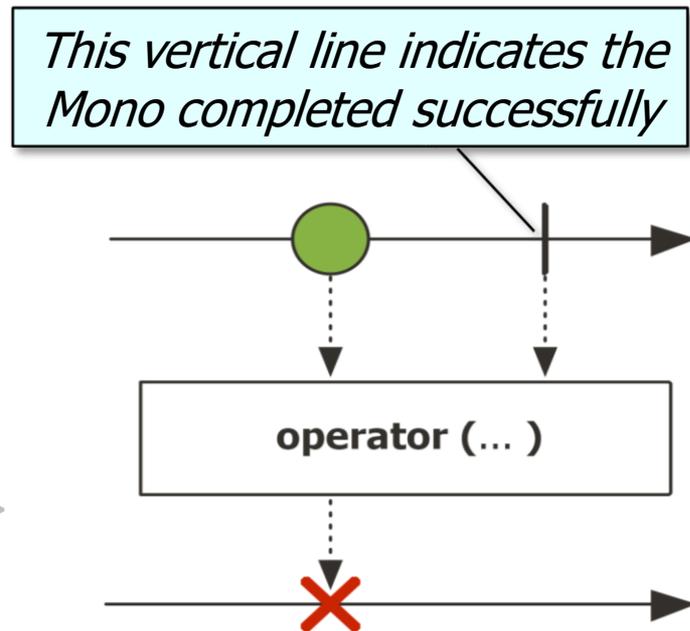
operator ( … )

# Key Classes in the Project Reactor API

- There are two 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"

These dotted lines & this box indicate that a transformation is being applied to the Mono

operator ( … )

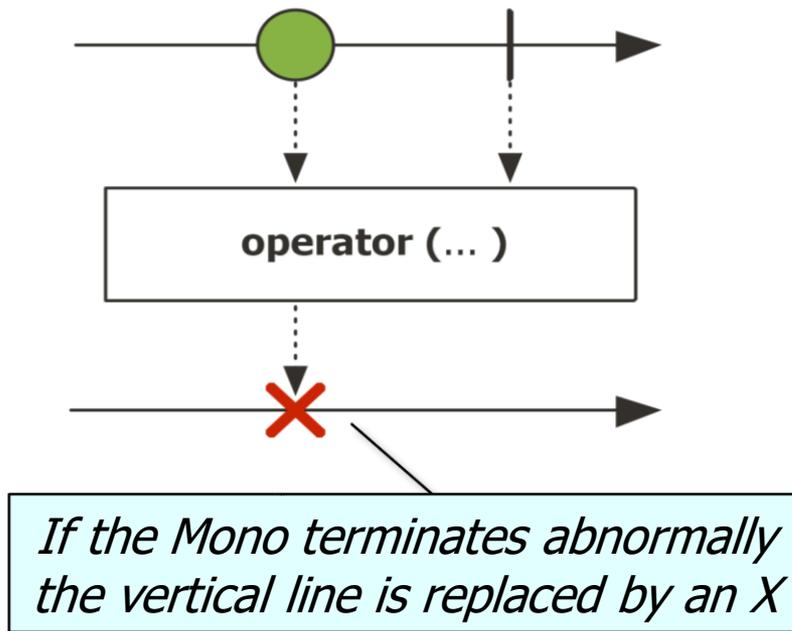The text inside the box indicates the type of transformation

# Key Classes in the Project Reactor API

- There are two 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"

operator (… )

This Mono is the result of the transformation

# Key Classes in the Project Reactor API

- There are two 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"



This vertical line indicates the Mono completed successfully

operator ( … )

# Key Classes in the Project Reactor API

- There are two 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"



operator ( … )

If the Mono terminates abnormally the vertical line is replaced by an X

# Key Classes in the Project Reactor API

- There are two 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 two 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

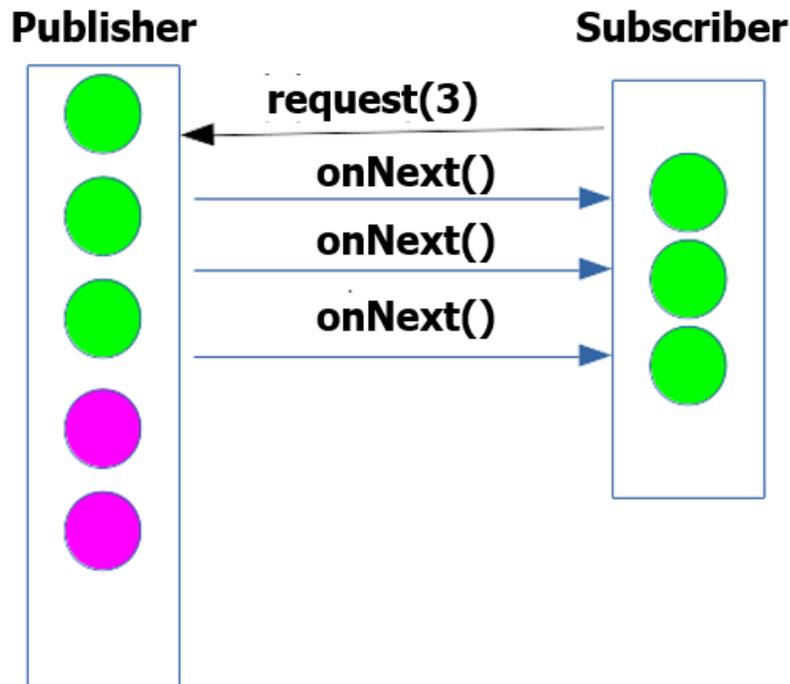# Key Classes in the Project Reactor API

- There are two 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 ->
        BigFractionUtils
          .sortAndPrintList
            (list, sb));
```
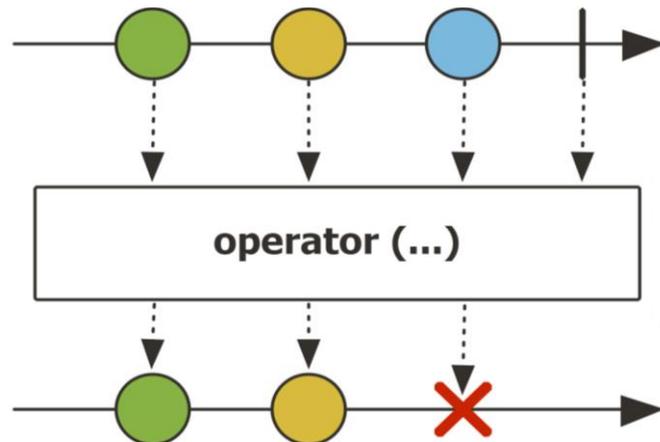
# Key Classes in the Project Reactor API

- There are two 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

**Publisher**                    **Subscriber**

request(3)

onNext()

onNext()

onNext()

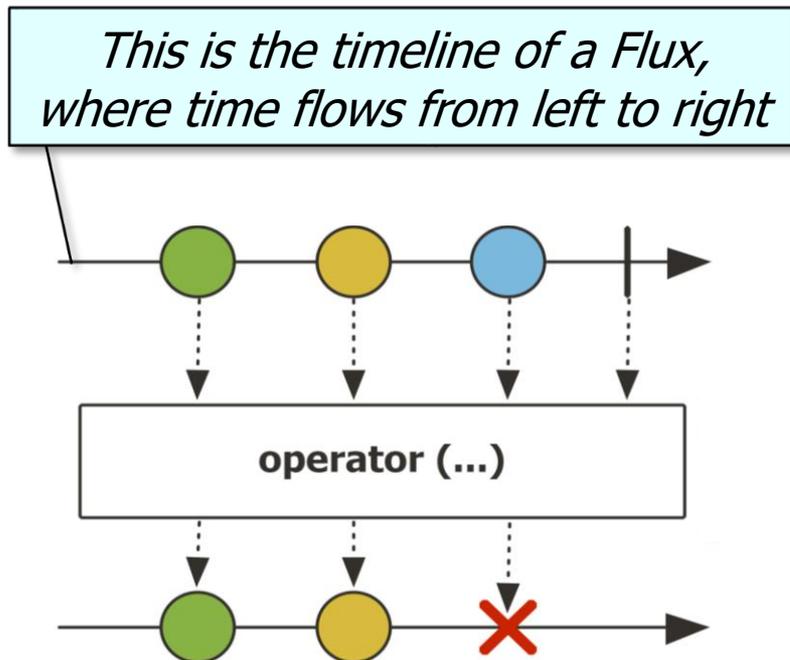See jstobigdata.com/java/backpressure-in-project-reactor

# Key Classes in the Project Reactor API

- There are two 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
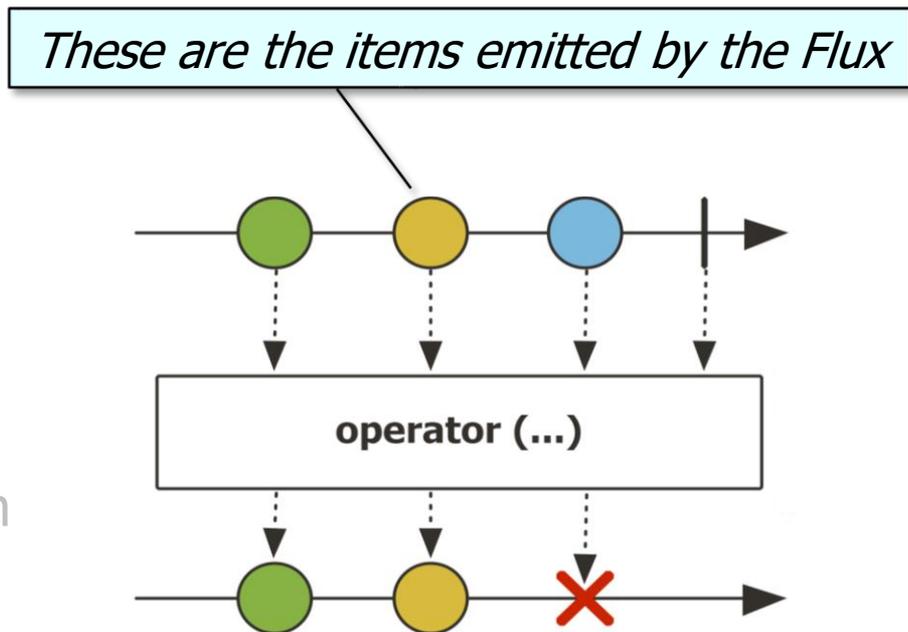
# Key Classes in the Project Reactor API

- There are two 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



This is the timeline of a Flux, where time flows from left to right
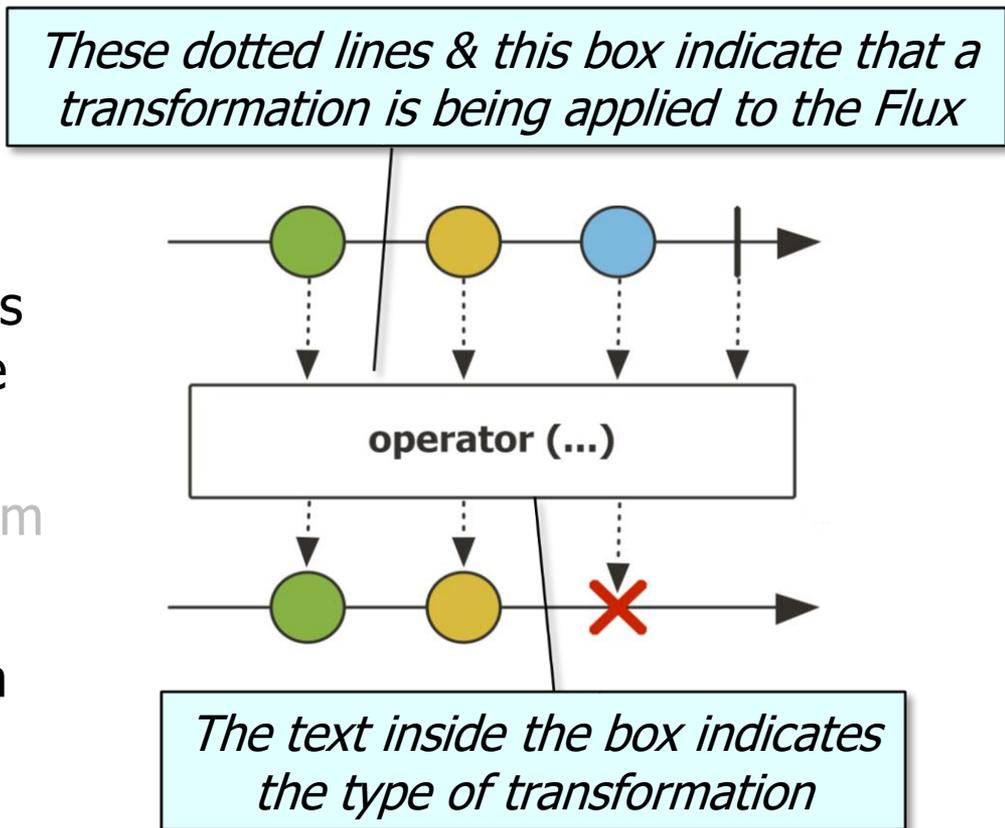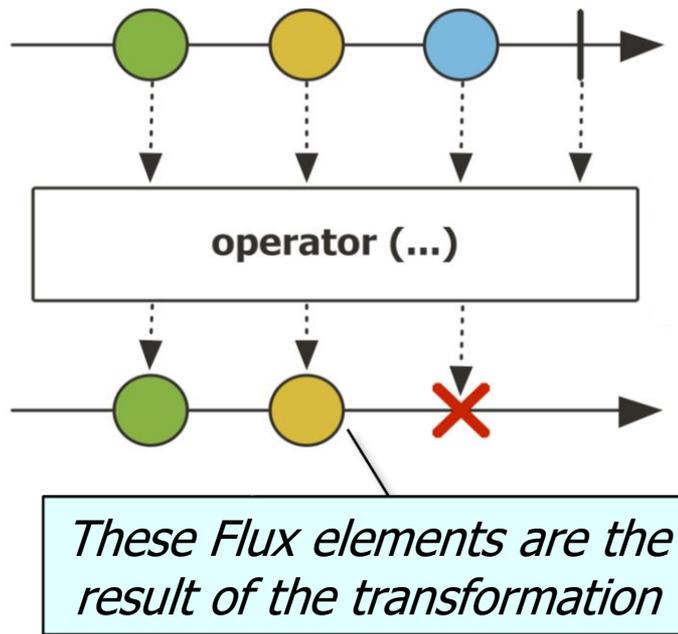
operator (...)

# Key Classes in the Project Reactor API

- There are two 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

*These are the items emitted by the Flux*

operator (...)

- There are two 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

*These dotted lines & this box indicate that a transformation is being applied to the Flux*

operator (...)

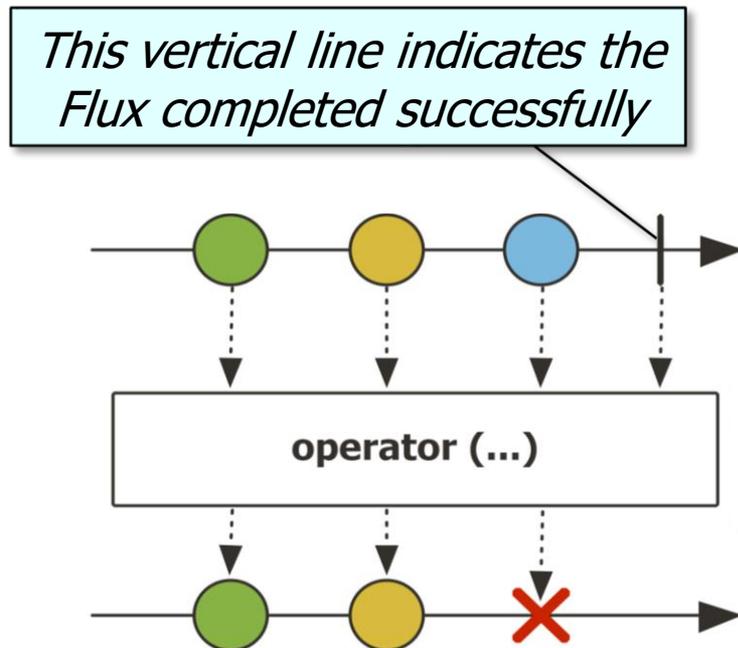*The text inside the box indicates the type of transformation*

# Key Classes in the Project Reactor API

- There are two 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



These Flux elements are the result of the transformation

- There are two 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

*This vertical line indicates the Flux completed successfully*
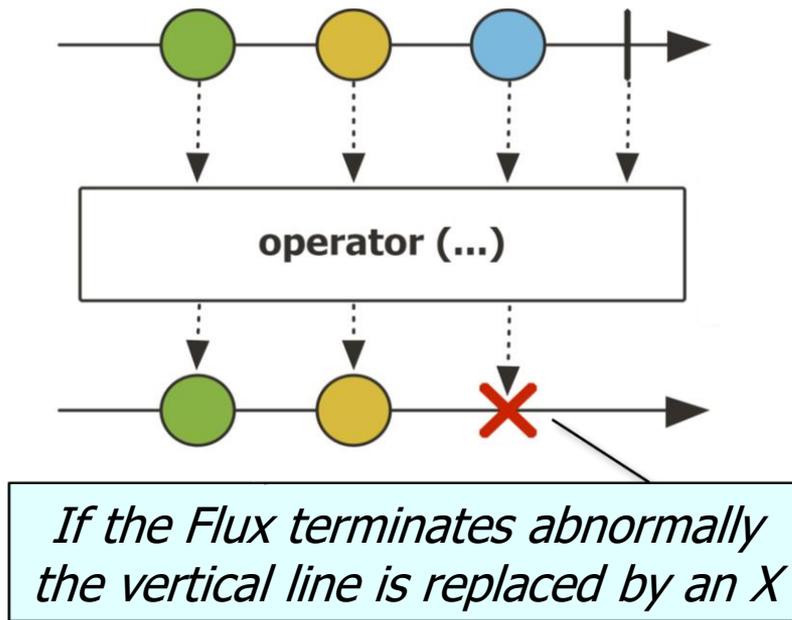
operator (...)

# Key Classes in the Project Reactor API

- There are two 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



*If the Flux terminates abnormally
the vertical line is replaced by an X*

# Key Classes in the Project Reactor API

- There are two 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.

# End of Understanding Key Classes in the Project Reactor API