Overview of the ParallelFlux Class

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

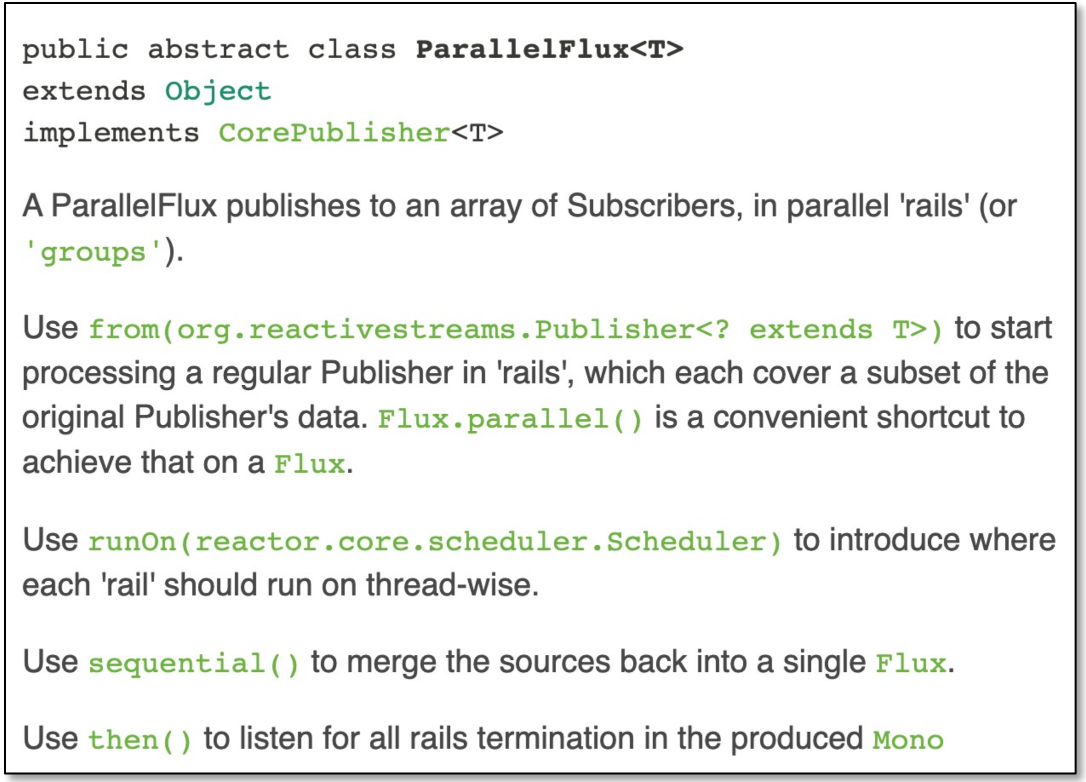
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the capabilities of the ParallelFlux class



```
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(org.reactivestreams.Publisher<? extends 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

Learning Objectives in this Part of the Lesson

- Understand the capabilities of the ParallelFlux class
  - Simplifies parallel processing *cf.* the flatMap() concurrency idiom

COMPLEX

```
return Flux
  .fromArray(bigFractionArray)
  .flatMap(bf -> Mono
     .fromCallable(() -> bf
         .multiply(sBigFraction))
     .subscribeOn(Schedulers
                  .computation()))
  .reduce(BigFraction::add)...
```

SIMPLE

```
return Flux
  .fromArray(bigFractionArray)
  .parallel()
  .runOn(Schedulers.computation())
  .map(bf -> bf.multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
```

See earlier lesson on *"Key Transforming Operators in the Flux Class (Part 3)"*

Overview of the ParallelFlux Class

Overview of the ParallelFlux Class

- The Project Reactor flatMap() concurrency idiom performs well, but is also somewhat convoluted..

```
return Flux
   .fromArray(bigFractionArray)

   .flatMap(bf -> Mono
      .fromCallable(() -> bf
          .multiply(sBigFraction))

      .subscribeOn(Schedulers
                     .computation()))

   .reduce(BigFraction::add)
   ...
```

Return a Flux that emits multiplied BigFraction objects via the Project Reactor flatMap() concurrency idiom

See previous lessons on *"Key Transforming Operators in the Flux Class (Part 3)"*

Overview of the ParallelFlux Class

- The Project Reactor flatMap() concurrency idiom performs well, but is also somewhat convoluted..
  - Particularly in comparison with Java parallel streams

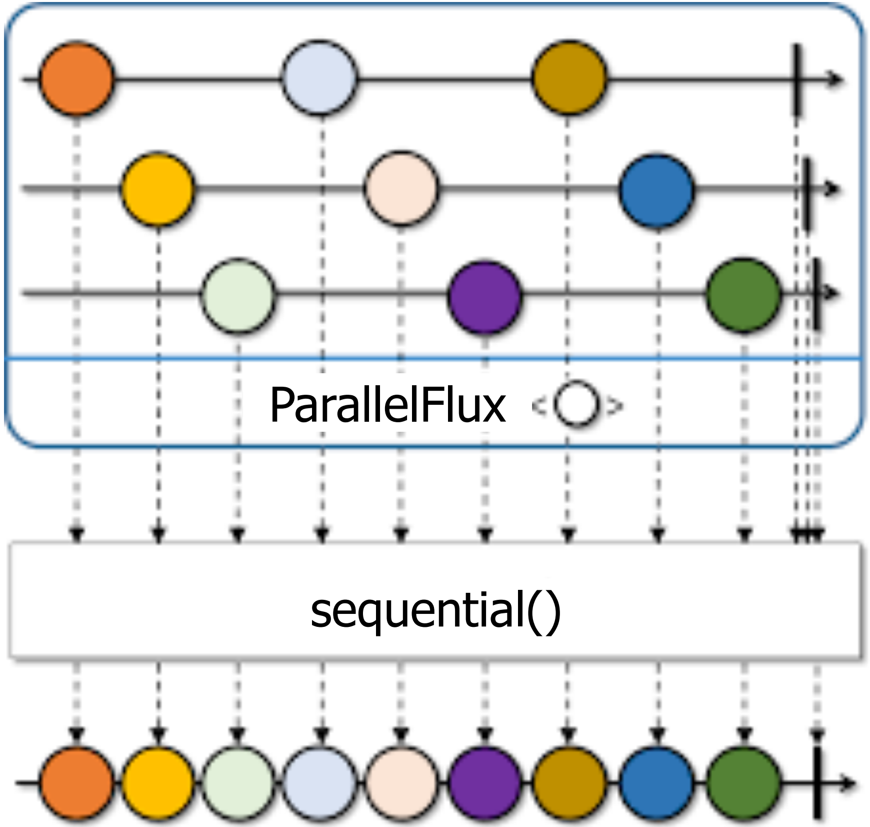
```
return Stream
  .of(bigFractionArray)

  .parallel()

  .map(bf -> bf
       .multiply(sBigFraction))

  .reduce(ZERO, BigFraction::add)
```

```
return Flux
  .fromArray(bigFractionArray)

  .flatMap(bf -> Mono
     .fromCallable(() -> bf
         .multiply(sBigFraction))

     .subscribeOn(Schedulers
                   .computation()))

  .reduce(BigFraction::add)
  ...
```

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Overview of the ParallelFlux Class

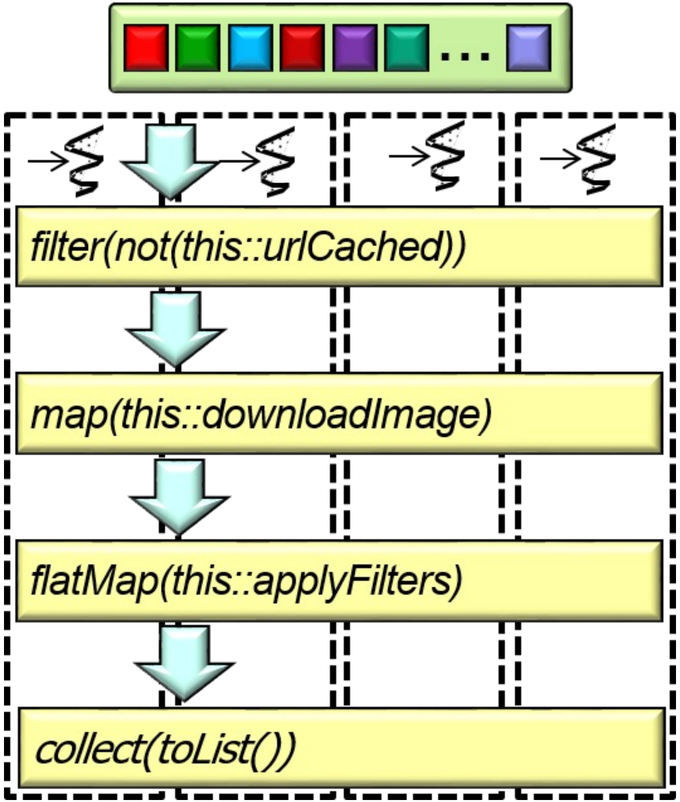
- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel



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

Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams



See dzone.com/articles/rxjava-idiomatic-concurrency-flatmap-vs-parallel

Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
    - i.e., intended for "embarrassingly parallel" tasks

"Embarrassingly parallel" tasks have little/no dependency or need for communication between tasks or for sharing results between them

See en.wikipedia.org/wiki/Embarrassingly_parallel

Overview of the ParallelFlux Class

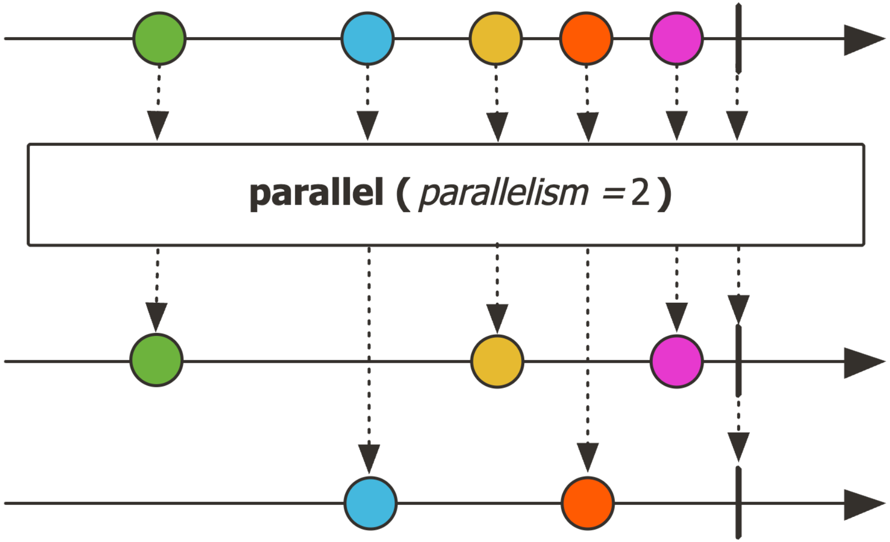
- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom

See dzone.com/articles/rxjava-idiomatic-concurrency-flatmap-vs-parallel

Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
  - The Flux.parallel() factory method creates a ParallelFlux

`ParallelFlux<T> parallel()`

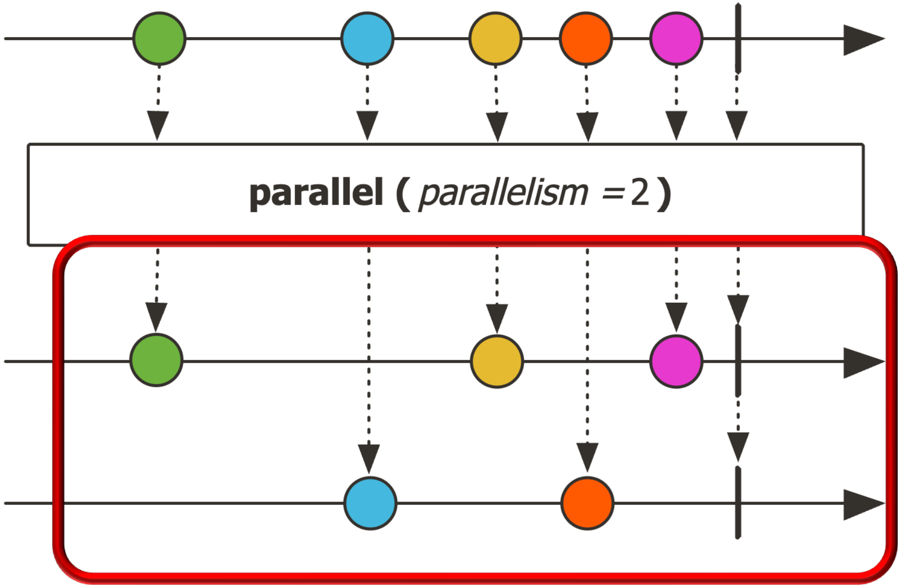


See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#parallel

Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
  - The Flux.parallel() factory method creates a ParallelFlux
    - Elements are processed in parallel via 'rails' in round-robin order

```
ParallelFlux<T> parallel()
```



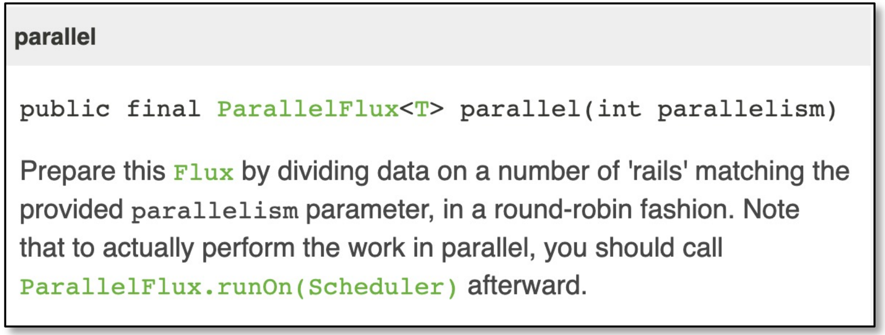
Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
  - The Flux.parallel() factory method creates a ParallelFlux
    - Elements are processed in parallel via 'rails' in round-robin order
    - By default, the # of rails is set to the # of available CPU cores

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors

Overview of the ParallelFlux Class

- ParallelFlux is a subset of Flux that provides a more concise means of processing multiple values in parallel
  - Similar to Java parallel streams
  - Avoids the convoluted syntax of the flatMap() concurrency idiom
  - The Flux.parallel() factory method creates a ParallelFlux
    - Elements are processed in parallel via 'rails' in round-robin order
    - By default, the # of rails is set to the # of available CPU cores
      - This setting can be changed programmatically



parallel

```
public final ParallelFlux<T> parallel(int parallelism)
```

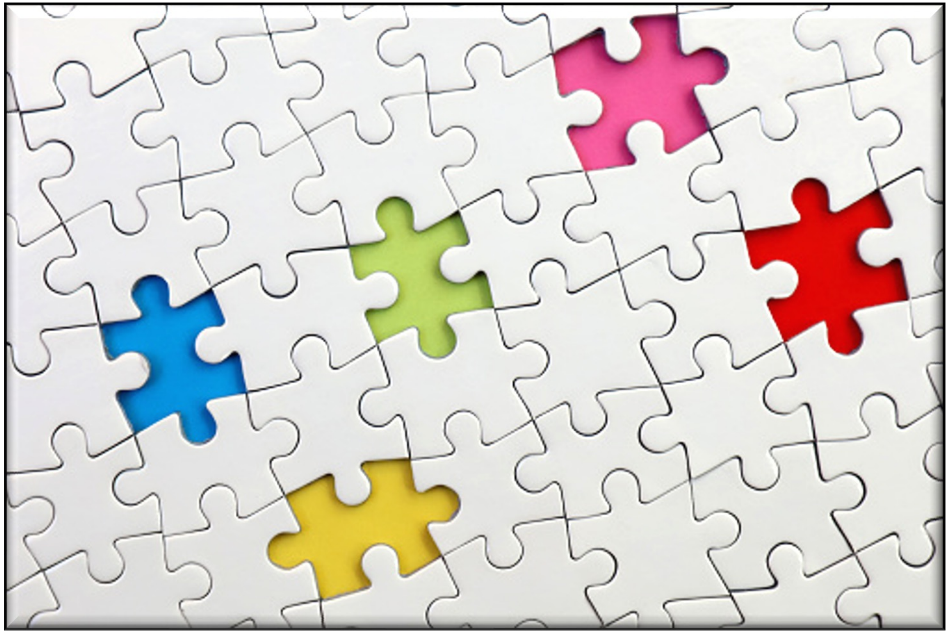
Prepare this `Flux` by dividing data on a number of 'rails' matching the provided `parallelism` parameter, in a round-robin fashion. Note that to actually perform the work in parallel, you should call `ParallelFlux.runOn(Scheduler)` afterward.

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

Key Operators in the ParallelFlux Class

Key Operators in the ParallelFlux Class

- ParallelFlux supports a subset of Flux operators that process elements in parallel across the rails
  - e.g., map(), filter(), concatMap(), flatMap(), collect(), & reduce()



See www.vinsguru.com/reactor-parallel-flux

Key Operators in the ParallelFlux Class

- The runOn() operator specifies where each 'rail' will observe its incoming elements

```
ParallelFlux<T> runOn(Scheduler
                          scheduler)
```

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

Key Operators in the ParallelFlux Class

- The runOn() operator specifies where each 'rail' will observe its incoming elements
  - Specified via a Scheduler that performs no work-stealing

```
ParallelFlux<T> runOn(Scheduler
                              scheduler)
```

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html

Key Operators in the ParallelFlux Class

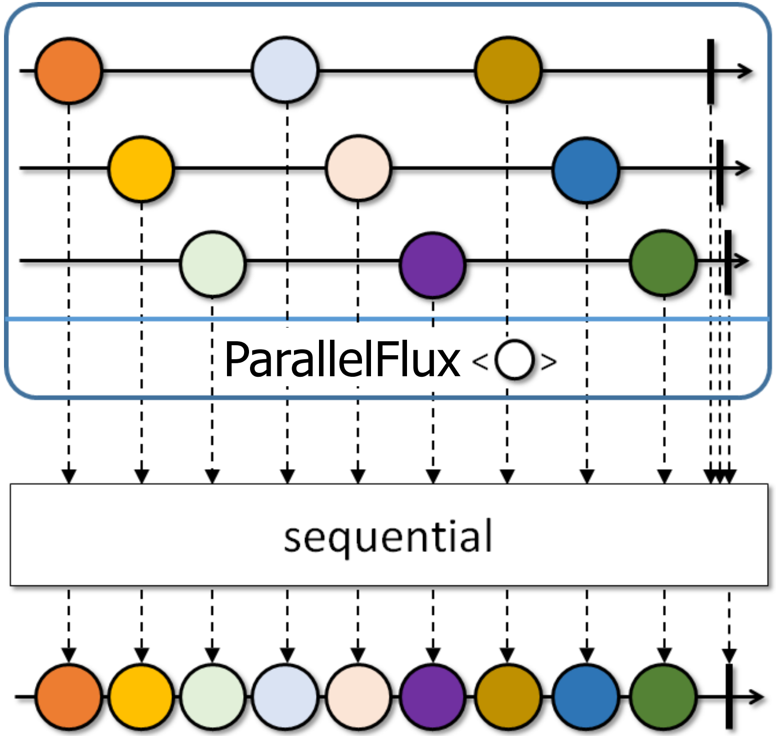
- The runOn() operator specifies where each 'rail' will observe its incoming elements
  - Specified via a Scheduler that performs no work-stealing
  - Returns the new Parallel Flux instance

```
ParallelFlux<T> runOn(Scheduler
                          scheduler)
```

Key Operators in the ParallelFlux Class

- A ParallelFlux can be converted back into a Flux via sequential()

```
Flux<T> sequential()
```

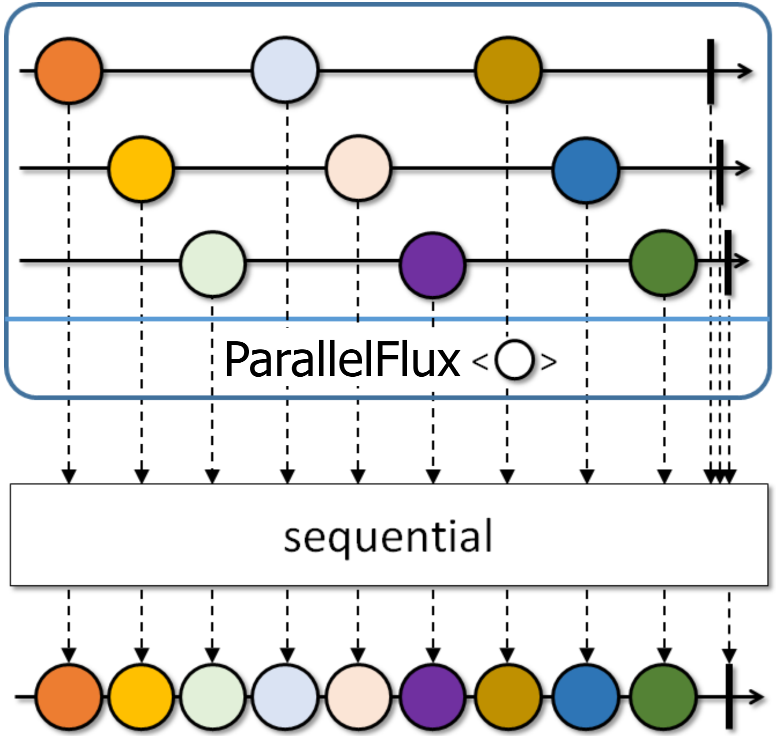


See projectreactor.io/docs/core/release/api/reactor/core/publisher/ParallelFlux.html#sequential

Key Operators in the ParallelFlux Class

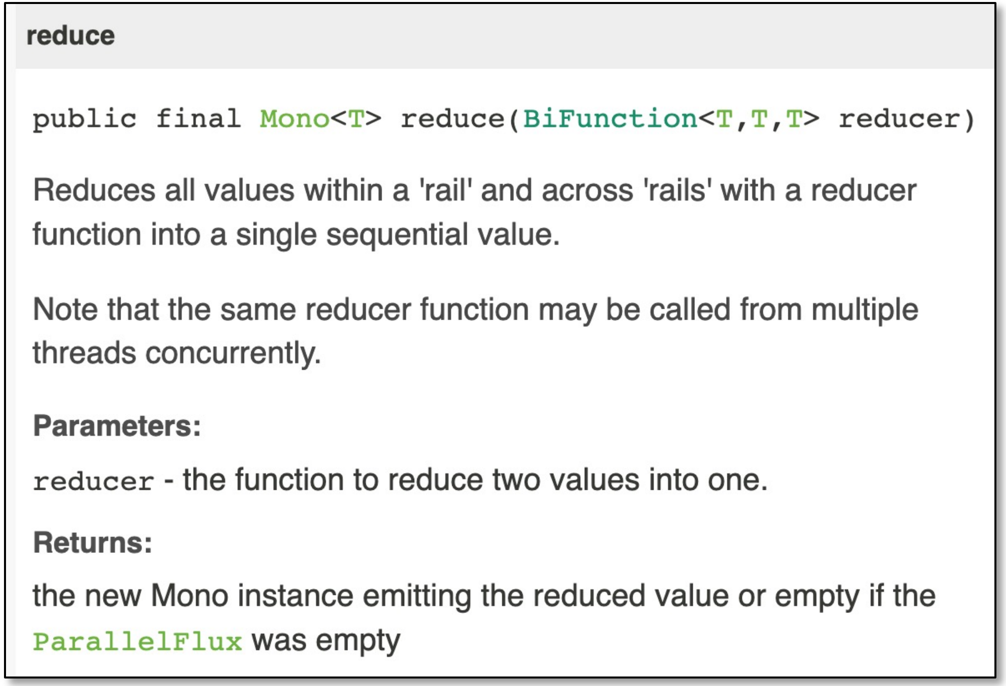
- A ParallelFlux can be converted back into a Flux via sequential()
  - Merge the values from each 'rail' in a round-robin fashion

```
Flux<T> sequential()
```



Key Operators in the ParallelFlux Class

- ParallelFlux.reduce() can also be used to convert back into a Mono



reduce

```
public final Mono<T> reduce(BiFunction<T,T,T> reducer)
```

Reduces all values within a 'rail' and across 'rails' with a reducer function into a single sequential value.

Note that the same reducer function may be called from multiple threads concurrently.

Parameters:

`reducer` - the function to reduce two values into one.

Returns:

the new Mono instance emitting the reduced value or empty if the `ParallelFlux` was empty

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

Key Operators in the ParallelFlux Class

- ParallelFlux.reduce() can also be used to convert back into a Mono
  - Reduces all values within a 'rail’ & across 'rails' into a single sequential value

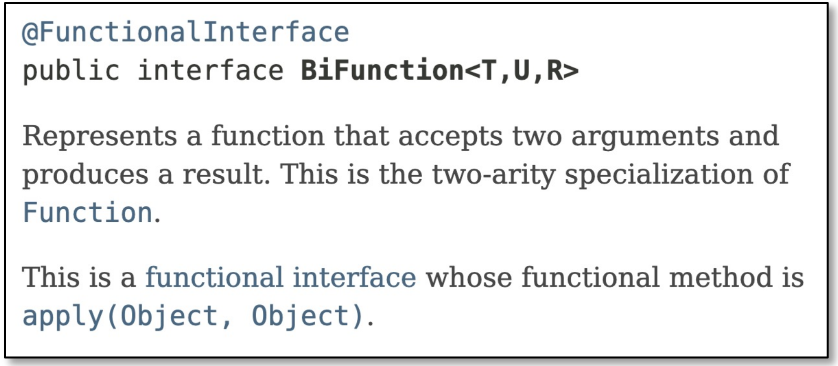
```
Mono<T> reduce
  (BiFunction<T,T,T> reducer)
```

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

Key Operators in the ParallelFlux Class

- ParallelFlux.reduce() can also be used to convert back into a Mono
  - Reduces all values within a 'rail' & across 'rails' into a single sequential value
  - The BiFunction param reduces two values into one successively

```
Mono<T> reduce
   (BiFunction<T,T,T> reducer)
```



@FunctionalInterface
public interface **BiFunction<T,U,R>**

Represents a function that accepts two arguments and produces a result. This is the two-arity specialization of Function.

This is a functional interface whose functional method is apply(Object, Object).

See docs.oracle.com/javase/8/docs/api/java/util/function/BiFunction.html

Key Operators in the ParallelFlux Class

- ParallelFlux.reduce() can also be used to convert back into a Mono
  - Reduces all values within a 'rail' & across 'rails' into a single sequential value
  - The BiFunction param reduces two values into one successively
  - Return a Mono that emits the reduced value or empty if the ParallelFlux was empty

```
Mono<T> reduce
  (BiFunction<T,T,T> reducer)
```

Key Operators in the ParallelFlux Class

- Elements that flow through the operators in a ParallelFlux stream are processed in parallel

```
return Flux
  .fromArray(bigFractionArray)
  .parallel()
  .runOn
     (Schedulers.parallel())
  .map(bf -> bf
         .multiply(sBigReducedFrac))
  .reduce(BigFraction::add)
  .doOnSuccess(displayResults)
  .then();
```

Multiply an array of BigFraction objects in parallel using Project Reactor's ParallelFlux operators

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex5

Key Operators in the ParallelFlux Class

- Elements that flow through the operators in a ParallelFlux stream are processed in parallel

```
return Flux
   .fromArray(bigFractionArray)
   .parallel()
   .runOn
      (Schedulers.parallel())
   .map(bf -> bf
          .multiply(sBigReducedFrac))
   .reduce(BigFraction::add)
   .doOnSuccess(displayResults)
   .then();
```

Designate the parallel Scheduler that multiplies each BigFraction in parallel

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel

End of Overview of the ParallelFlux Class