Key Combining Operators in the Mono 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

- Recognize key Mono operators
  - Concurrency & scheduler operators
  - Transforming operators
  - Combining operators
    - These operators create a Mono from multiple Mono objects
      - e.g., zipWith(), zip(), when(), & firstWithSignal()

Key Combining Operators in the Mono Class

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit

```
<T2,O> Mono<O>
zipWith(Mono<? extends T2> other,
        BiFunction<? super T,
                   ? super T2,
                   ? extends O>
                     combinator)
```

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

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
    - Combine the result from this & other Mono into another object via the given combinator BiFunction

```
<T2,O> Mono<O>
zipWith(Mono<? extends T2> other,
        BiFunction<? super T,
                   ? super T2,
                   ? extends O>
                     combinator)
```

Interface BiFunction<T,U,R>

Type Parameters:

T - the type of the first argument to the function

U - the type of the second argument to the function

R - the type of the result of the function

All Known Subinterfaces:

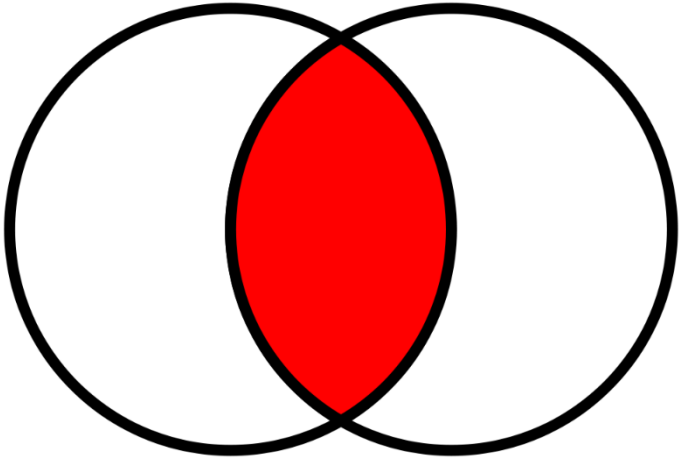
BinaryOperator<T>

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

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
    - Combine the result from this & other Mono into another object via the given combinator BiFunction
    - Returns a Mono that emits the combined results

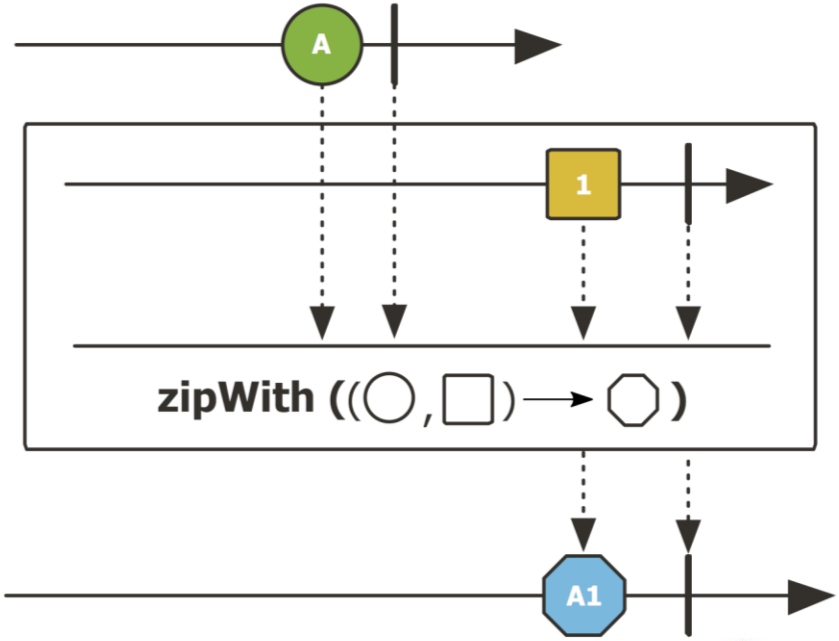
```
<T2,O> Mono<O>
zipWith(Mono<? extends T2> other,
        BiFunction<? super T,
                   ? super T2,
                   ? extends O>
                     combinator)
```



See en.wikipedia.org/wiki/Logical_conjunction

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
  - Can transform the type and/or value of elements it processes

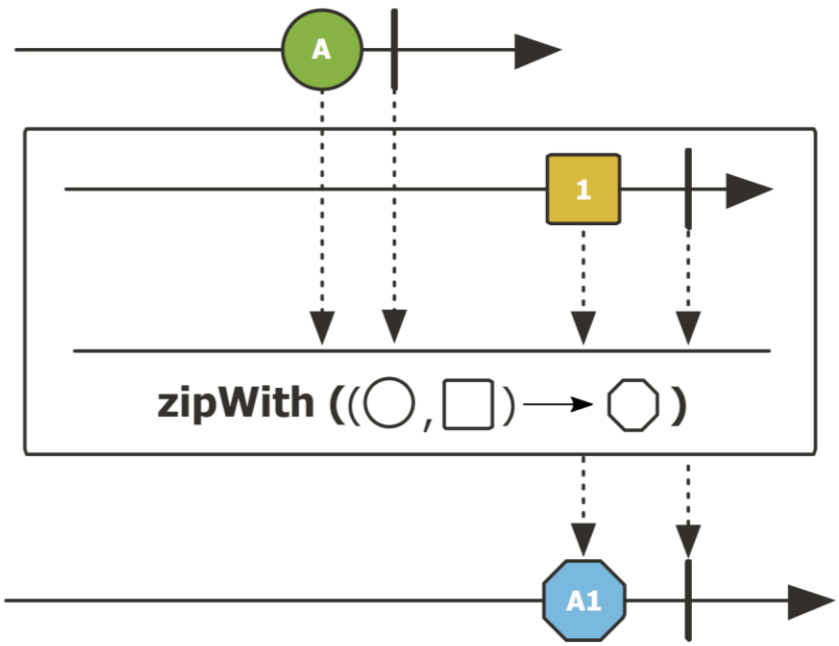


Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
  - Can transform the type and/or value of elements it processes

```
Mono<BigFraction> m1 =
  makeBigFractionAsync(...)
Mono<BigFraction> m2 =
  makeBigFractionAsync(...);

return m1
  .zipWith(m2,
           BigFraction::add);
```

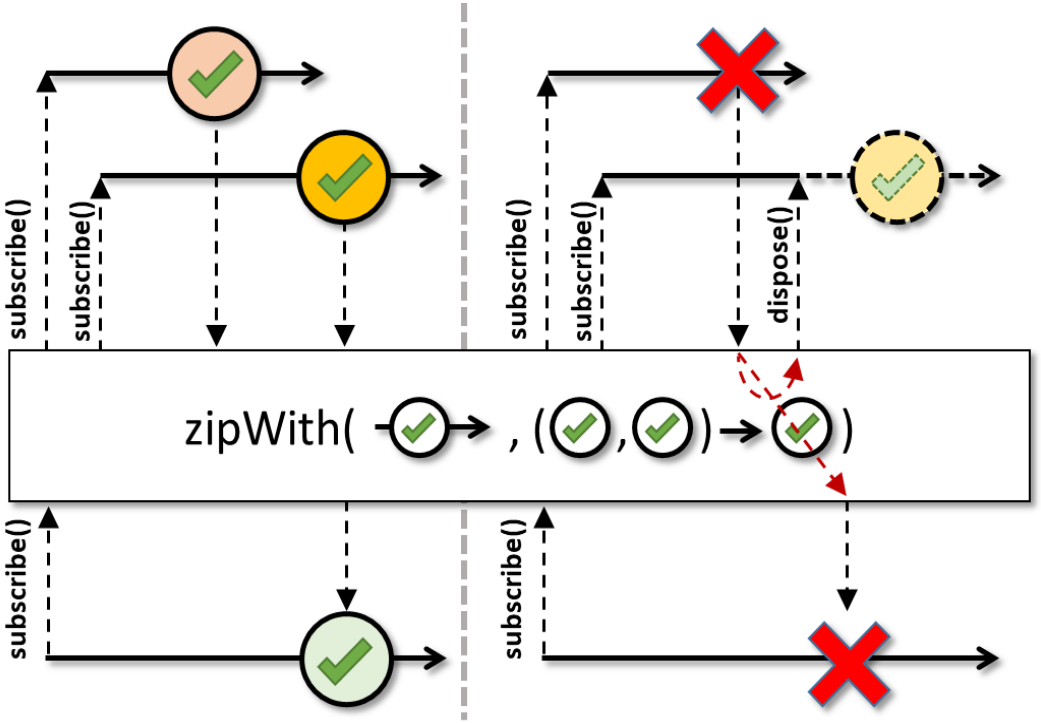


Add results after async computations complete

See Reactive/mono/ex3/src/main/java/MonoEx.java

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
  - Can transform the type and/or value of elements it processes
  - RxJava's Single.zipWith() works the same



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#zipWith

Key Combining Operators in the Mono Class

- The zipWith() method
  - Combine two results into one result after they both emit
  - Can transform the type and/or value of elements it processes
  - RxJava's Single.zipWith() works the same
  - Similar to Java Completable Future thenCombine() method

thenCombine

```
<U,V> CompletionStage<V> thenCombine(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn)
```

Returns a new CompletionStage that, when this and the other given stage both complete normally, is executed with the two results as arguments to the supplied function. See the `CompletionStage` documentation for rules covering exceptional completion.

Type Parameters:

`U` - the type of the other CompletionStage's result

`V` - the function's return type

Parameters:

`other` - the other CompletionStage

`fn` - the function to use to compute the value of the returned CompletionStage

Returns:

the new CompletionStage

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenCombine

Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined

```
static <R> Mono<R> zip
   (Function<? super Object[],
                 ? extends R>
      combinator,
    Mono<?>... monos)
```

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

Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
    - When all the given monos have produced an item, aggregate their values according to the provided combinator function

```
static <R> Mono<R> zip
   (Function<? super Object[],
                 ? extends R>
      combinator,
    Mono<?>... monos)
```

Interface Function<T,R>

Type Parameters:

T - the type of the input to the function

R - the type of the result of the function

All Known Subinterfaces:

UnaryOperator<T>

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

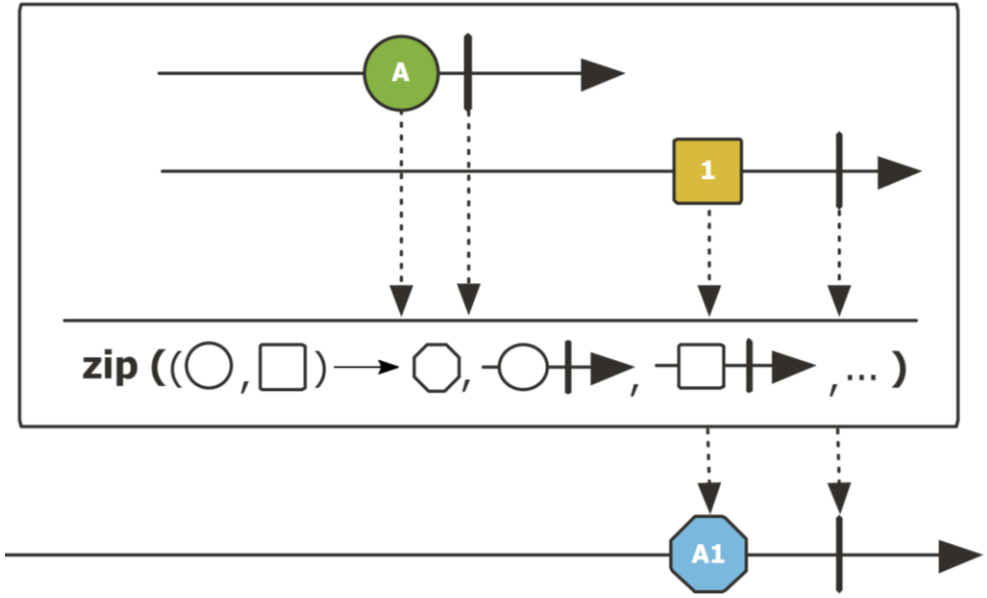
Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
    - When all the given monos have produced an item, aggregate their values according to the provided combinator function
    - Returns a Mono that emits the combined results

```
static <R> Mono<R> zip
   (Function<? super Object[],
                 ? extends R>
      combinator,
    Mono<?>... monos)
```

Key Combining Operators in the Mono Class

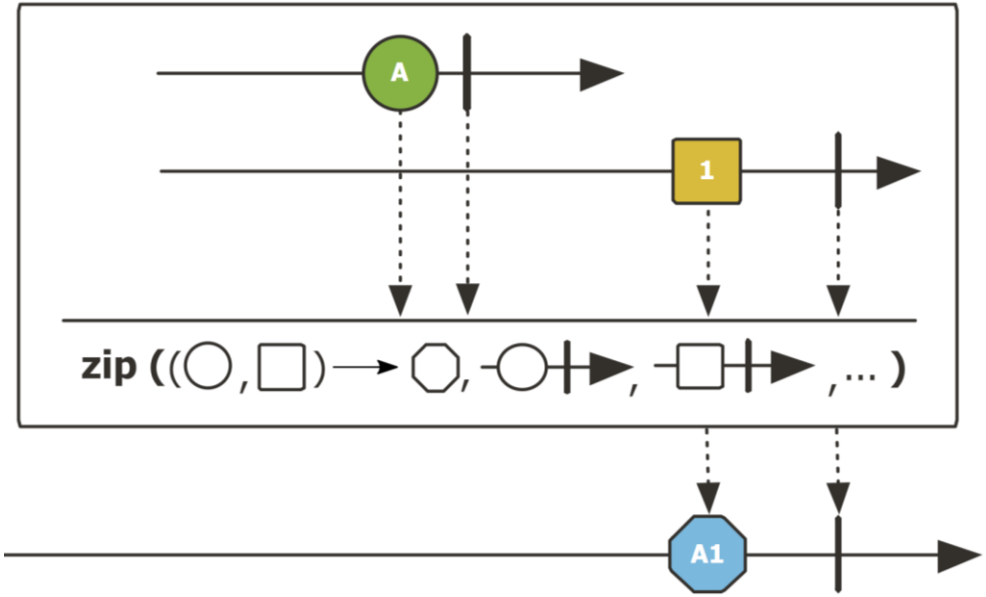
- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
  - Can transform the value and/or type of elements it processes



Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
  - Can transform the value and/or type of elements it processes

```
Mono<BigFraction> m1 =
  makeBigFractionAsync(...)
Mono<BigFraction> m2 =
  makeBigFractionAsync(...);
Mono<BigFraction> m3 =
  makeBigFractionAsync(...);
...
return Mono.zip(combiner, m1, m2, m3);
```

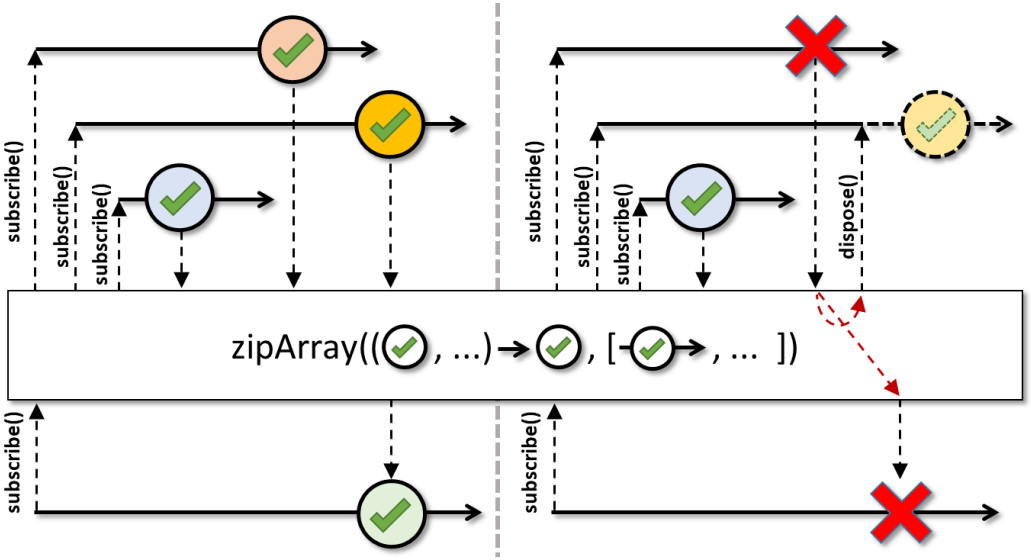


Use the **`combiner`** *to add results after async computations complete*

See [Reactive/mono/ex3/src/main/java/MonoEx.java]

Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
  - Can transform the value and/or type of elements it processes
  - RxJava's Single.zipArray() works the same



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#zipArray

Key Combining Operators in the Mono Class

- The zip() method
  - Aggregate given monos into a new Mono after they all emit & are combined
  - Can transform the value and/or type of elements it processes
  - RxJava's Single.zipArray() works the same
  - Similar to the Java Completable Future allOf() method

allOf

```
public static CompletableFuture<Void> allOf(
CompletableFuture<?>... cfs)
```

Returns a new CompletableFuture that is completed when all of the given CompletableFutures complete. If any of the given CompletableFutures complete exceptionally, then the returned CompletableFuture also does so, with a CompletionException holding this exception as its cause. Otherwise, the results, if any, of the given CompletableFutures are not reflected in the returned CompletableFuture, but may be obtained by inspecting them individually. If no CompletableFutures are provided, returns a CompletableFuture completed with the value `null`.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal

```
static <T> Mono<T>
  firstWithSignal
    (Mono<? extends T>... monos)
```

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

Key Combining Operators in the Mono Class

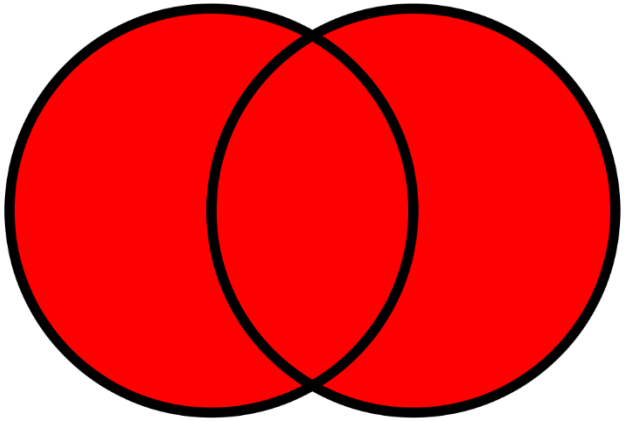
- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
    - This method picks the fastest of competing Mono sources

```
static <T> Mono<T>
  firstWithSignal
    (Mono<? extends T>... monos)
```

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
    - This method picks the fastest of competing Mono sources
    - Returns a new Mono behaving like the fastest of its sources

```
static <T> Mono<T>
   firstWithSignal
      (Mono<? extends T>... monos)
```



See en.wikipedia.org/wiki/Logical_disjunction

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
    - This method picks the fastest of competing Mono sources
    - Returns a new Mono behaving like the fastest of its sources
      - The others are disposed of

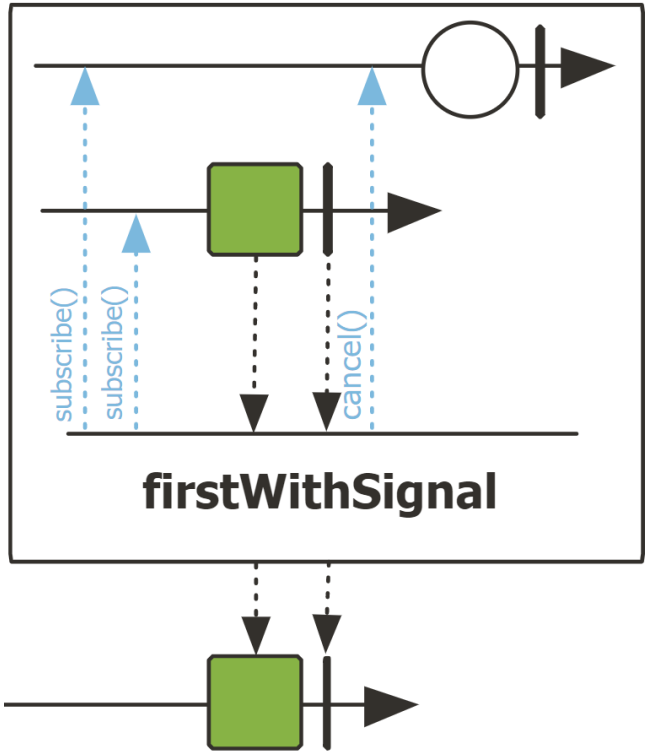
```
static <T> Mono<T>
   firstWithSignal
      (Mono<? extends T>... monos)
```

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
  - Does not transform the type of elements it processes

```
Mono<List<BigFraction>> qM = Mono
   .fromCallable(() -> quickSort(list))
   .subscribeOn(Schedulers.parallel());
Mono<List<BigFraction>> hM =  Mono
   .fromCallable(() -> heapSort(list))
   .subscribeOn(Schedulers.parallel());

return Mono.firstWithSignal(qM, hM)...
```



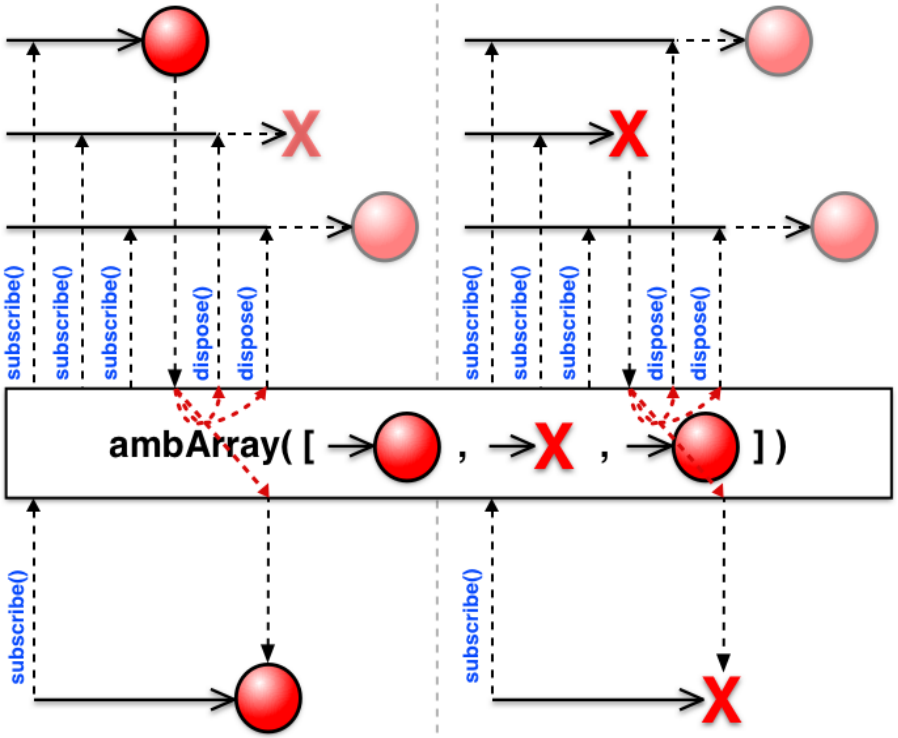
Select the result of which ever sort finishes first

See Reactive/flux/ex3/src/main/java/utils/BigFractionUtils.java

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
  - Does not transform the type of elements it processes
- RxJava's Single.ambArray() method works the same way

```
Single<List<BigFraction>> qS =
   ...
Single<List<BigFraction>> hS =
   ...
return Single.ambArray(qS, hS)...
```



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#ambArray

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
  - Does not transform the type of elements it processes
  - RxJava's Single.ambArray() method works the same way
  - Similar to the Java Completable Future.anyOf()

anyOf

```
public
static CompletableFuture<Object> anyOf(
CompletableFuture<?>... cfs)
```

Returns a new CompletableFuture that is completed when any of the given CompletableFutures complete, with the same result. Otherwise, if it completed exceptionally, the returned CompletableFuture also does so, with a CompletionException holding this exception as its cause. If no CompletableFutures are provided, returns an incomplete CompletableFuture.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#anyOf

Key Combining Operators in the Mono Class

- The firstWithSignal() method
  - Pick the first Mono to emit any signal (value, empty completion, or error) & replay that signal
  - Does not transform the type of elements it processes
  - RxJava's Single.ambArray() method works the same way
  - Similar to the Java Completable Future.anyOf()
    - Also a generalization of Completable Future.applyToEither()

applyToEither

```
<U> CompletionStage<U> applyToEither(
CompletionStage<? extends T> other,
Function<? super T,U> fn)
```

Returns a new CompletionStage that, when either this or the other given stage complete normally, is executed with the corresponding result as argument to the supplied function. See the CompletionStage documentation for rules covering exceptional completion.

Type Parameters:

U - the function's return type

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#applyToEither

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono

```
static Mono<Void> when
   (Publisher<?>... sources)
```

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

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
    - The param is an array of sources

```
static Mono<Void> when
    (Publisher<?>... sources)
```

Interface Flow.Publisher<T>

Type Parameters:

T - the published item type

All Known Subinterfaces:

Flow.Processor<T,R>, HttpRequest.BodyProcessor

All Known Implementing Classes:

SubmissionPublisher

Enclosing class:

Flow

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

See docs.oracle.com/javase/9/docs/api/java/util/concurrent/Flow.Publisher.html

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
    - The param is an array of sources
    - Returns a Mono<Void> that will be fulfilled when all the given sources have completed

```
static Mono<Void> when
   (Publisher<?>... sources)
```

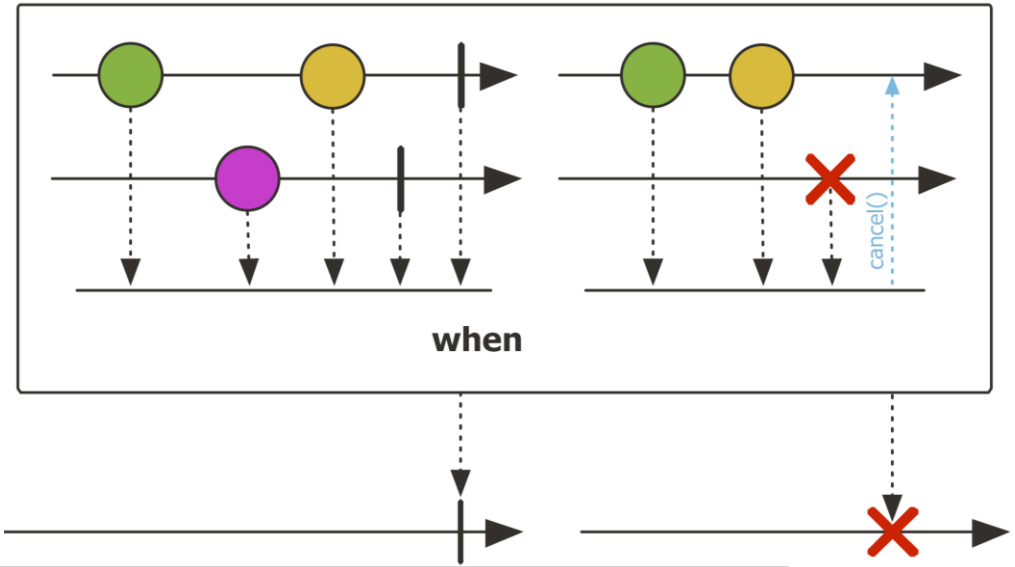
Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
    - The param is an array of sources
    - Returns a Mono<Void> that will be fulfilled when all the given sources have completed
      - An error causes cancellation of pending results & immediate error emission to the returned Mono

```
static Mono<Void> when
   (Publisher<?>... sources)
```

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
  - Typically used to wait for a fixed, but arbitrary, # of publishers to complete



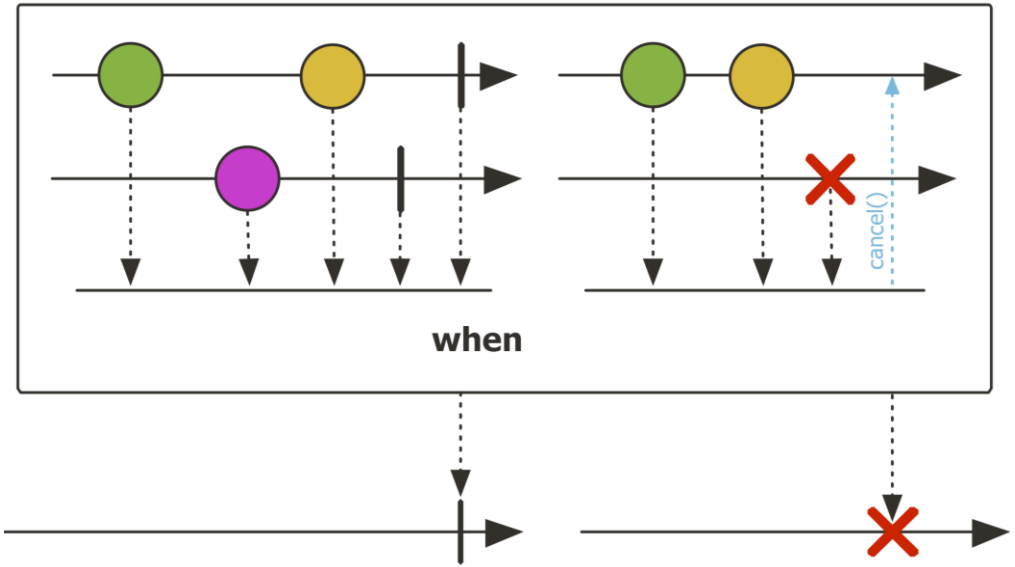

```
return monos -> Mono
   .when(monos)
   .materialize()
   .flatMap(v -> Flux.fromIterable(monos)
                     .map(Mono::block)
                     .collect(toList()));
```

Return a new Mono that completes when all Monos in the list complete

See Reactive/flux/ex3/src/main/java/utils/MonosCollector.java

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
  - Typically used to wait for a fixed, but arbitrary, # of publishers to complete
    - Often followed by materialize()



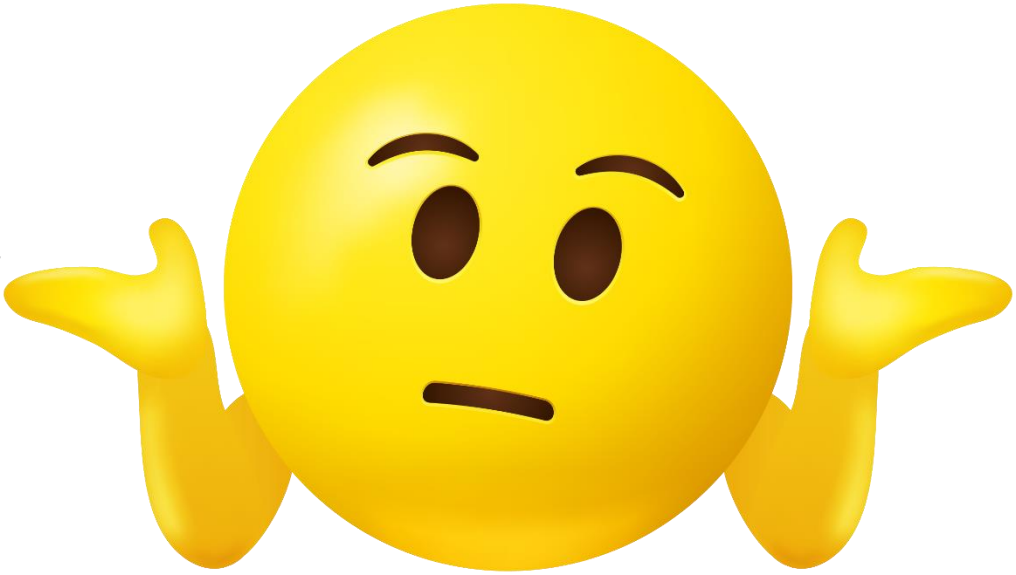

```
return monos -> Mono
   .when(monos)
   .materialize()
   .flatMap(v -> Flux.fromIterable(monos)
                     .map(Mono::block)
                     .collect(toList()));
```

Transform onNext, onError, & onComplete signals into a Signal instance, which emits onComplete()

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

Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
  - Typically used to wait for a fixed, but arbitrary, # of publishers to complete
  - There is no RxJava equivalent



Key Combining Operators in the Mono Class

- The when() method
  - Aggregate given publishers into a new Mono
  - Typically used to wait for a fixed, but arbitrary, # of publishers to complete
  - There is no RxJava equivalent
  - Similar to the Java Completable Future allOf() method

allOf

```
public static CompletableFuture<Void> allOf(
CompletableFuture<?>... cfs)
```

Returns a new CompletableFuture that is completed when all of the given CompletableFutures complete. If any of the given CompletableFutures complete exceptionally, then the returned CompletableFuture also does so, with a CompletionException holding this exception as its cause. Otherwise, the results, if any, of the given CompletableFutures are not reflected in the returned CompletableFuture, but may be obtained by inspecting them individually. If no CompletableFutures are provided, returns a CompletableFuture completed with the value `null`.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#allOf

End of Key Combining Operators in the Mono Class