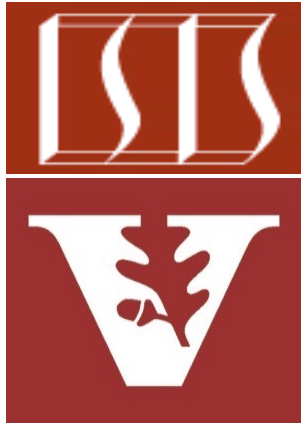


Java Parallel Streams Internals: Splitting, Combining, & Pooling

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 parallel stream internals, e.g.
 - Know what can change & what can't
 - Splitting, combining, & pooling mechanisms

```
final class Collectors {  
    ...  
    public static <T> Collector<T, ?, List<T>>  
        toList() { ... }  
  
    public static <T> Collector<T, ?, Set<T>>  
        toSet() { ... }  
    ...  
}
```

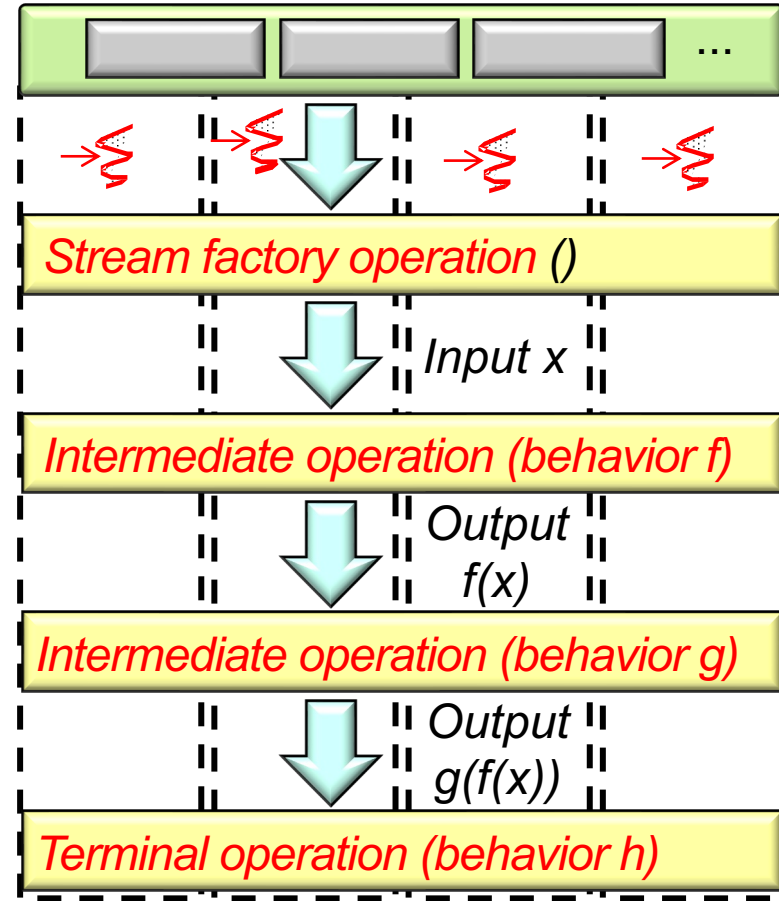
```
public interface Spliterator<T> {  
    boolean tryAdvance  
        (Consumer<? Super T> action);  
  
    Spliterator<T> trySplit();  
  
    long estimateSize();  
  
    int characteristics();  
}
```

```
public interface ManagedBlocker {  
    boolean block()  
        throws InterruptedException;  
  
    boolean isReleasable();  
}
```

Parallel Stream Splitting, Combining, & Pooling Mechanisms

Parallel Stream Splitting, Combining, & Pooling Mechanisms

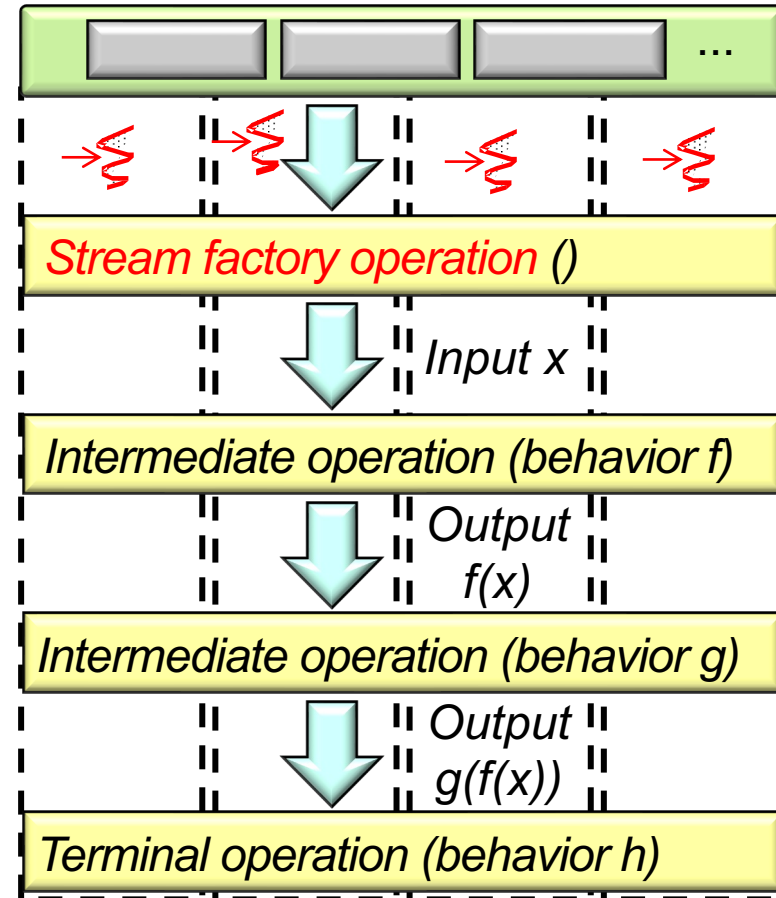
- A parallel stream's splitting, combining, & pooling mechanisms are often invisible



Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

```
interface Collection<E> {  
    ...  
    default Spliterator<E> spliterator() {  
        return Spliterators  
            .spliterator(this, 0);  
    }  
  
    default Stream<E> parallelStream() {  
        return StreamSupport  
            .stream(spliterator(), true);  
    }  
    ...  
}
```

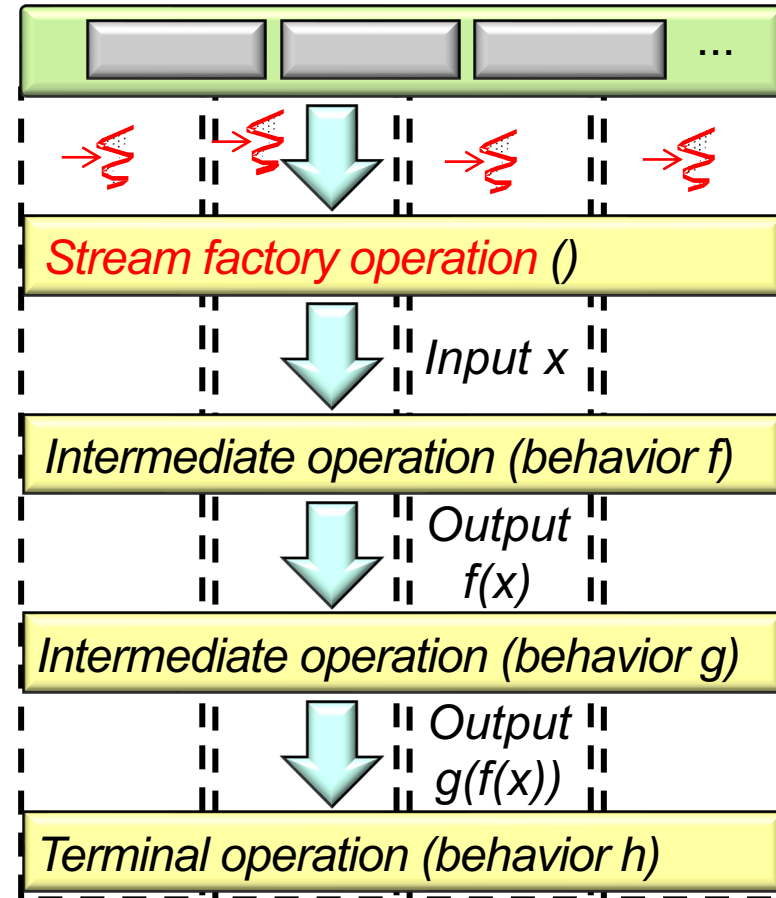


See docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

```
interface Collection<E> {  
    ...  
    default Spliterator<E> spliterator() {  
        return Spliterators  
            .spliterator(this, 0);  
    }  
  
    default Stream<E> parallelStream() {  
        return StreamSupport  
            .stream(spliterator(), true);  
    }  
    ...  
}
```

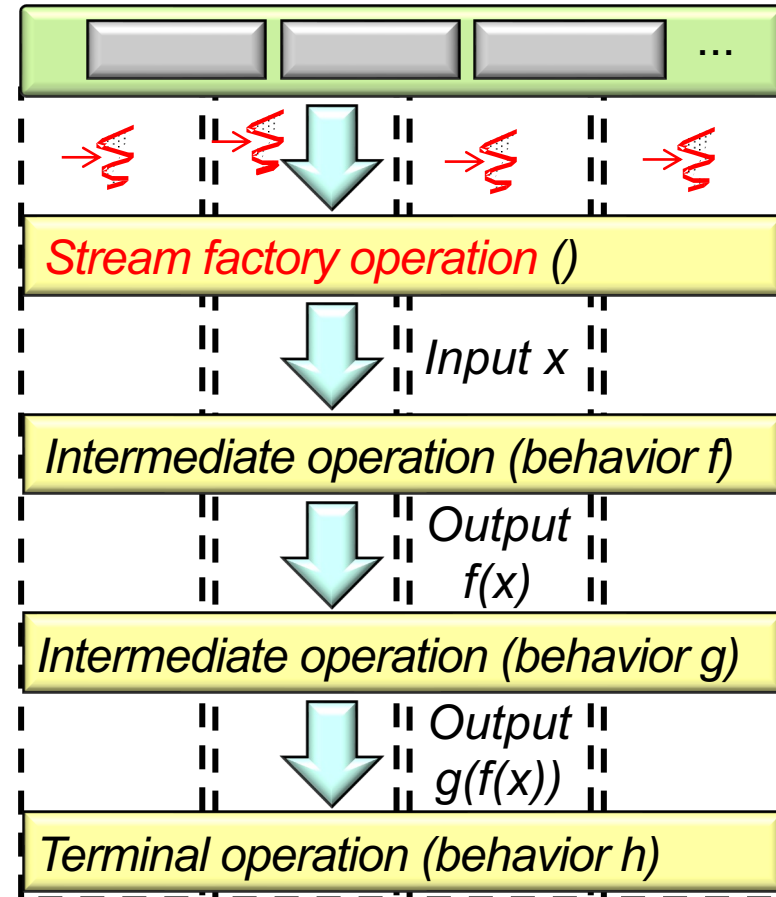


See docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html

Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
- All Java collections have predefined spliterators that create parallel streams

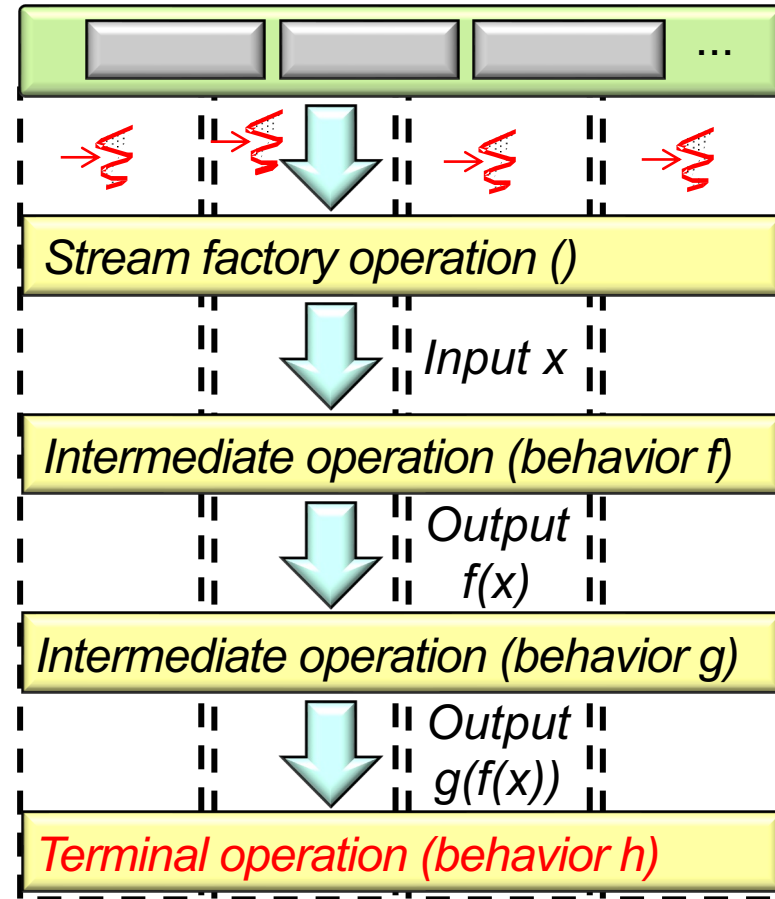
```
interface Collection<E> {  
    ...  
    default Spliterator<E> spliterator() {  
        return Spliterators  
            .spliterator(this, 0);  
    }  
  
    default Stream<E> parallelStream() {  
        return StreamSupport  
            .stream(spliterator(), true);  
    }  
    ...  
}
```



Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
 - All Java collections have predefined spliterators that create parallel streams
- Java also predefines collector factory methods in the Collectors utility class

```
final class Collectors {  
    ...  
    public static <T> Collector<T, ?, List<T>>  
        toList() { ... }  
  
    public static <T> Collector<T, ?, Set<T>>  
        toSet() { ... }  
    ...  
}
```

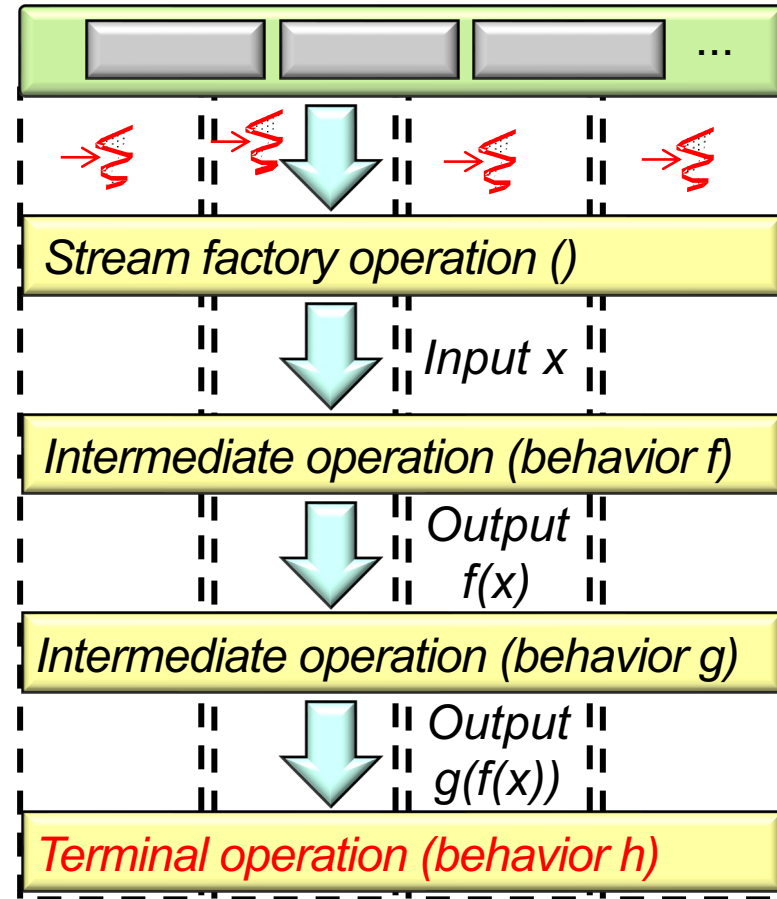


See docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html

Parallel Stream Splitting, Combining, & Pooling Mechanisms

- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
 - All Java collections have predefined spliterators that create parallel streams
- Java also predefines collector factory methods in the Collectors utility class

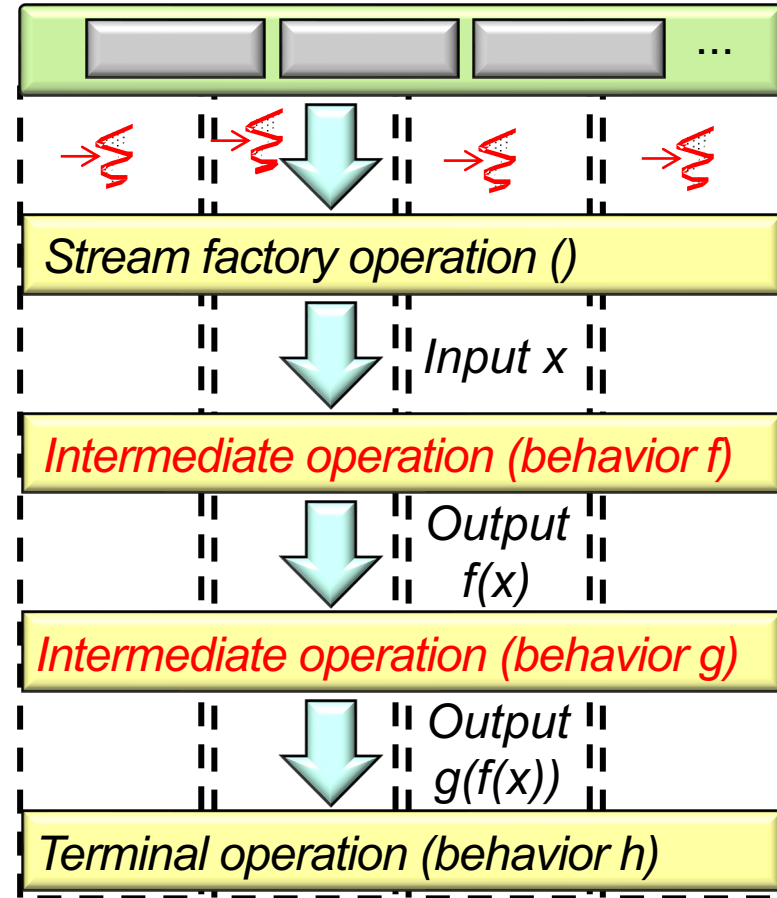
```
final class Collectors {  
    ...  
    public static <T> Collector<T, ?, List<T>>  
        toList() { ... }  
  
    public static <T> Collector<T, ?, Set<T>>  
        toSet() { ... }  
    ...  
}
```



These non-concurrent collectors can work seamlessly with parallel streams

Parallel Stream Splitting, Combining, & Pooling Mechanisms

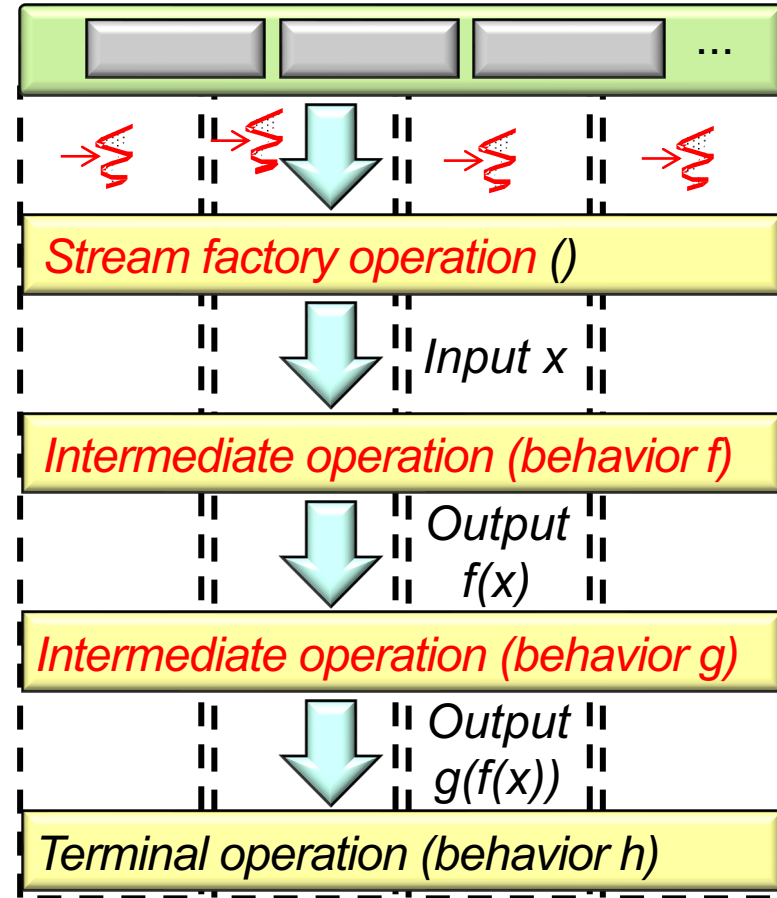
- A parallel stream's splitting, combining, & pooling mechanisms are often invisible
 - All Java collections have predefined spliterators that create parallel streams
 - Java also predefines collector factory methods in the Collectors utility class
- The common fork-join pool is used to run intermediate operations on chunks of data



See www.baeldung.com/java-fork-join

Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms



See upcoming lessons on "Java Parallel Stream Internals"

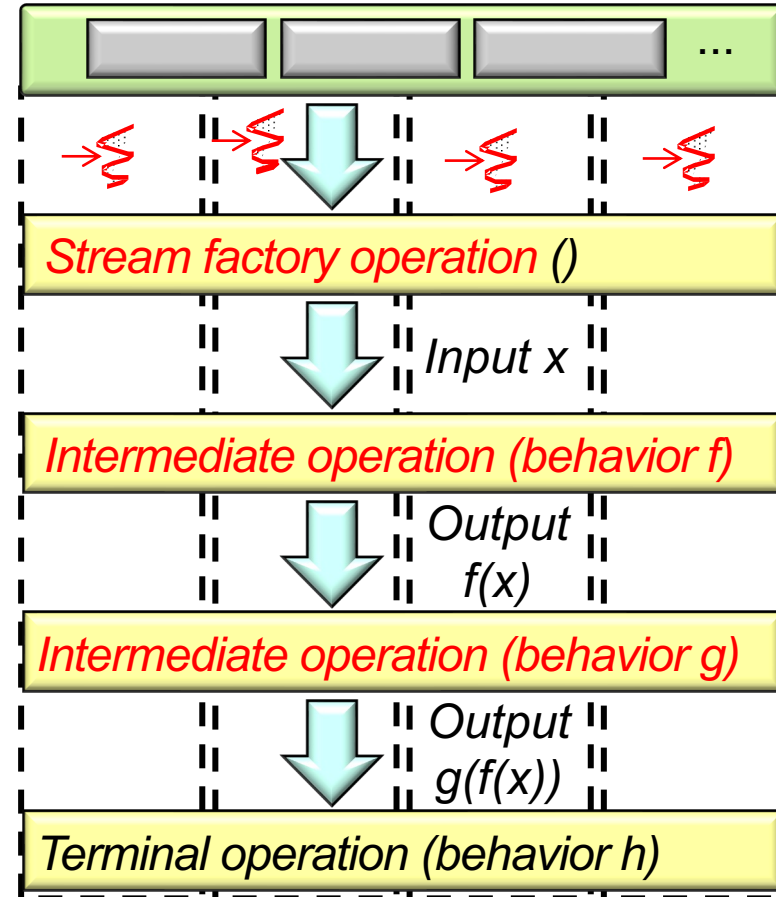
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms



```
interface Splitter<T> {  
    boolean tryAdvance  
        (Consumer<? Super T> action);  
    Splitter<T> trySplit();  
    void forEachRemaining  
        (Consumer<? Super T> action);  
    long estimateSize();  
    int characteristics();  
}
```

An interface used to traverse & partition elements of a source.



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

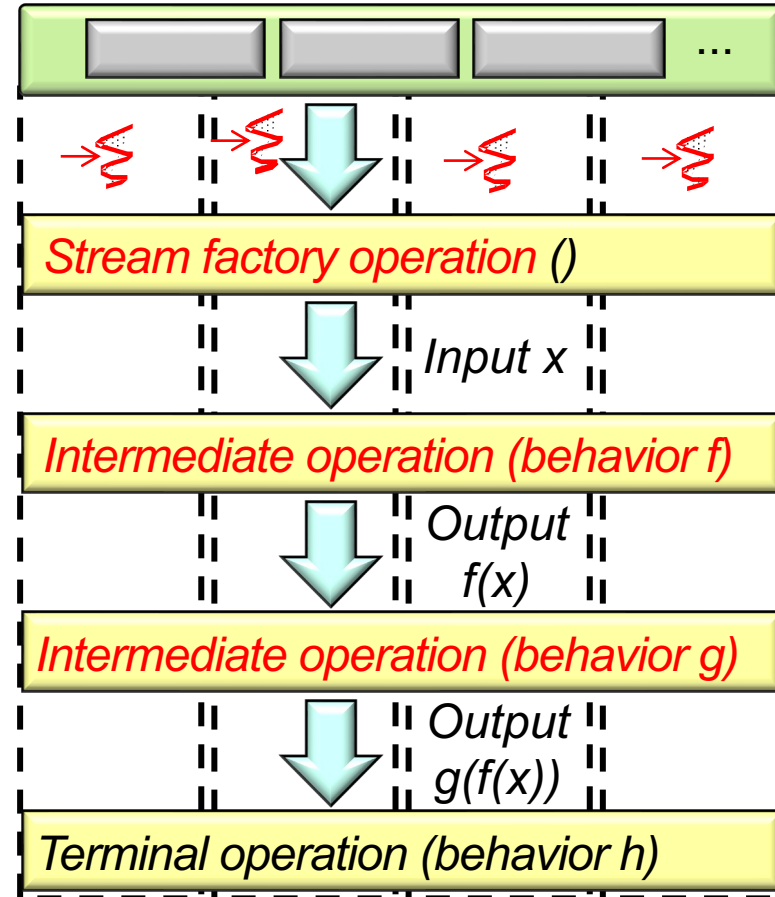
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms



```
interface Spliterator<T> {  
    boolean tryAdvance  
        (Consumer<? Super T> action);  
    Spliterator<T> trySplit();  
    void forEachRemaining  
        (Consumer<? Super T> action);  
    long estimateSize();  
    int characteristics();  
}
```

The streams framework uses this method to process elements in sequential and parallel streams.



See earlier lesson on "Java Streams: Applying Spliterators"

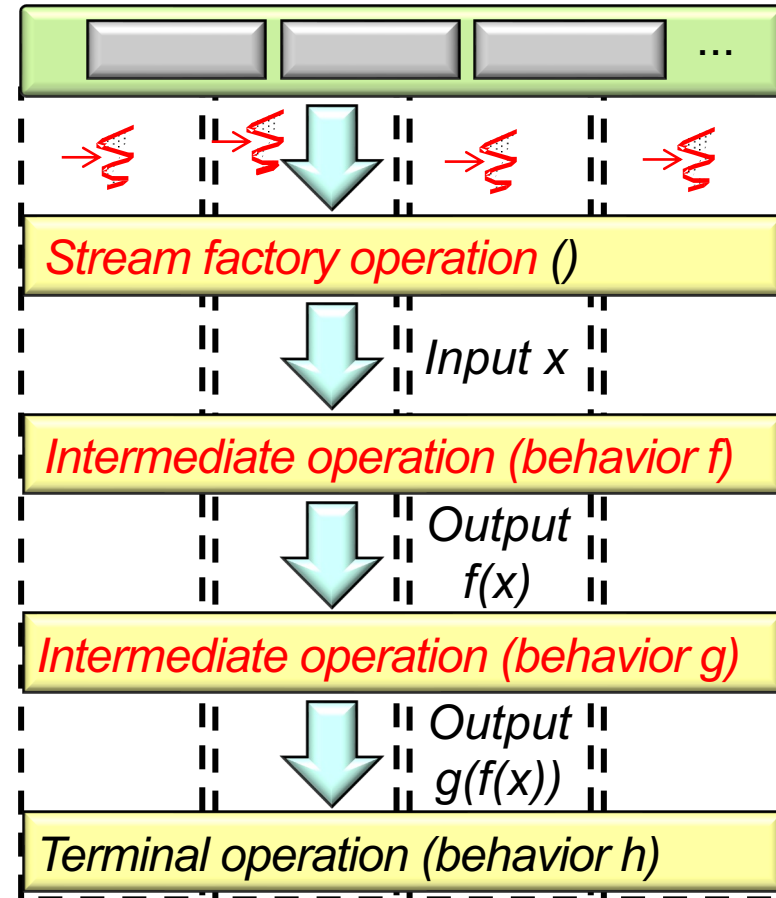
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms



```
interface Splitter<T> {  
    boolean tryAdvance  
        (Consumer<? Super T> action);  
    Splitter<T> trySplit();  
    void forEachRemaining  
        (Consumer<? Super T> action);  
    long estimateSize();  
    int characteristics();  
}
```

The streams framework uses this method to partition elements in a parallel stream.



See upcoming lesson on "Java Parallel Streams Internals: Partitioning"

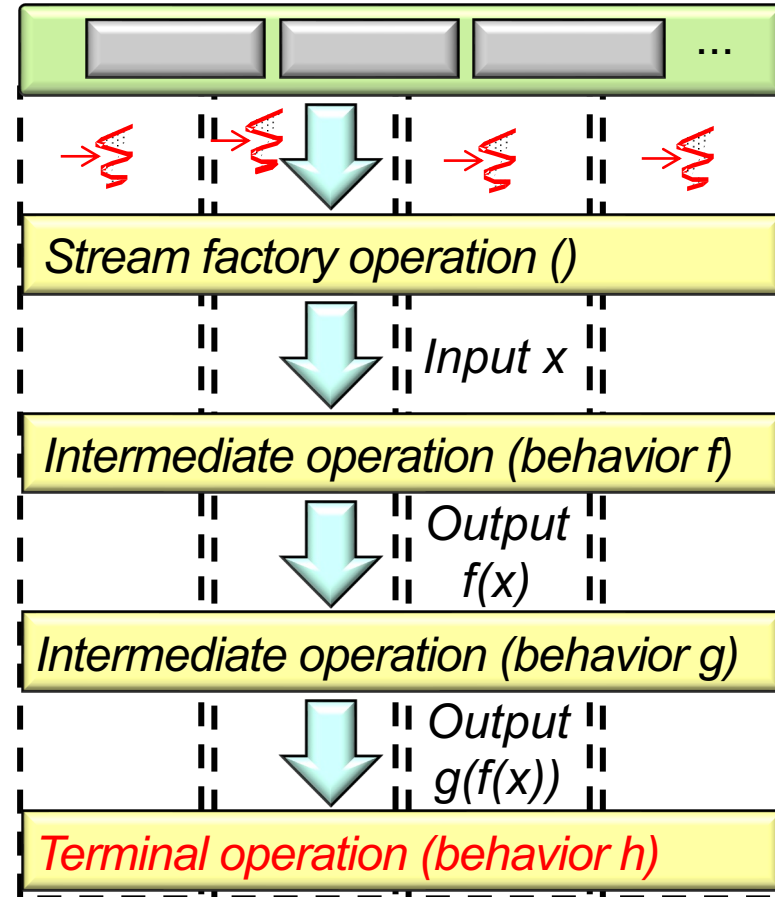
Parallel Stream Splitting, Combining, & Pooling Mechanisms

- However, parallel streams programmers can also customize these mechanisms



```
interface Collector<T,A,R> {  
    Supplier<A> supplier();  
    BiConsumer<A, T> accumulator();  
    BinaryOperator<A> combiner();  
    Function<A, R> finisher();  
    Set<Collector.Characteristics>  
        characteristics()  
    ...  
}
```

A framework that accumulates input elements into a concurrent and/or non-concurrent mutable result containers.



See docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html

Parallel Stream Splitting, Combining, & Pooling Mechanisms

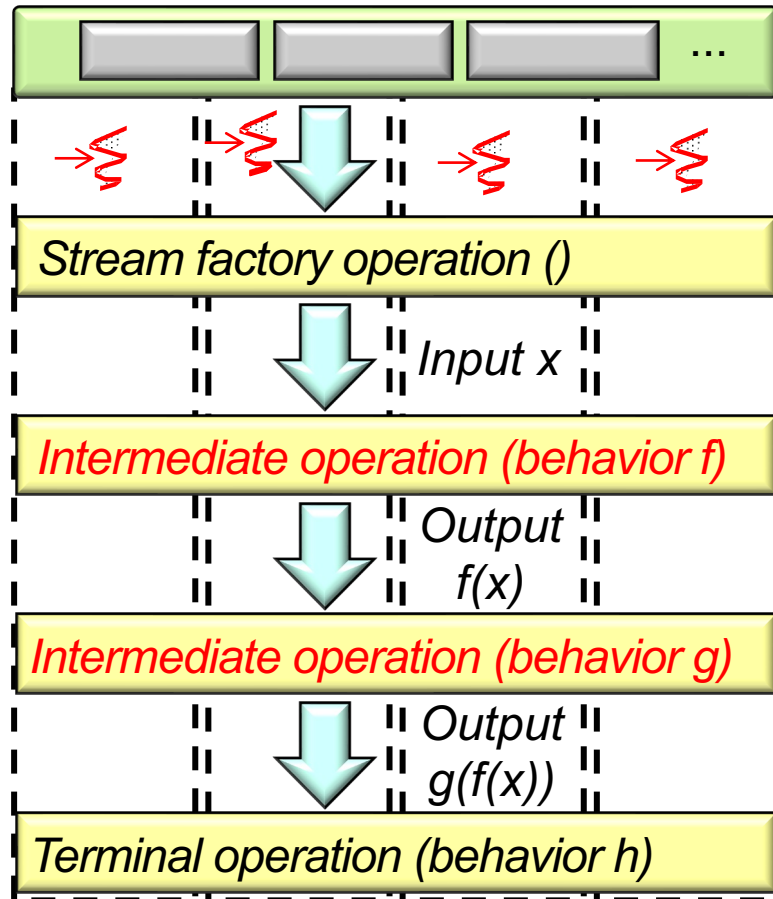
- However, parallel streams programmers can also customize these mechanisms



```
public interface ManagedBlocker {  
    boolean block()  
        throws InterruptedException;  
    boolean isReleasable();  
}
```



This interface provides managed parallelism for tasks running in the common fork-join pool.



End of Java Parallel Stream Internals: Splitting, Combining, & Pooling