Java Streams Intermediate Operations filter() & flatMap()

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 structure & functionality of stream aggregate operations
  - Intermediate operations
    - map() & mapToInt()
    - filter() & flatMap()

Input x

Stream flatMap(Function<…> mapper)

Output f(x)

Stream filter(Predicate<…> pred)

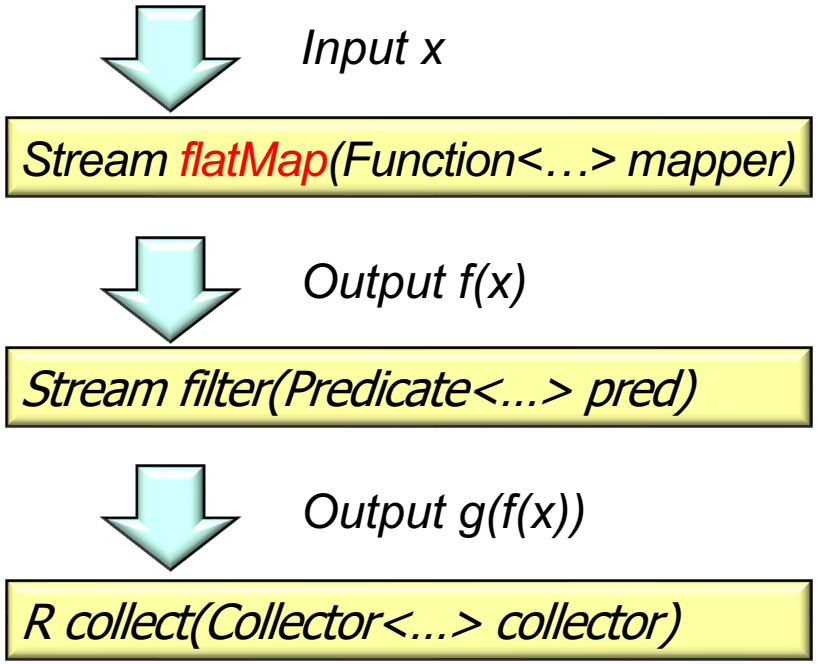
Output g(f(x))

R collect(Collector<…> collector)

These are both stateless, run-to-completion operations

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of stream aggregate operations
  - Intermediate operations
    - map() & mapToInt()
    - filter() & flatMap()
      - We also discuss a curious limitation with flatMap() that makes it ineffective for parallel streams

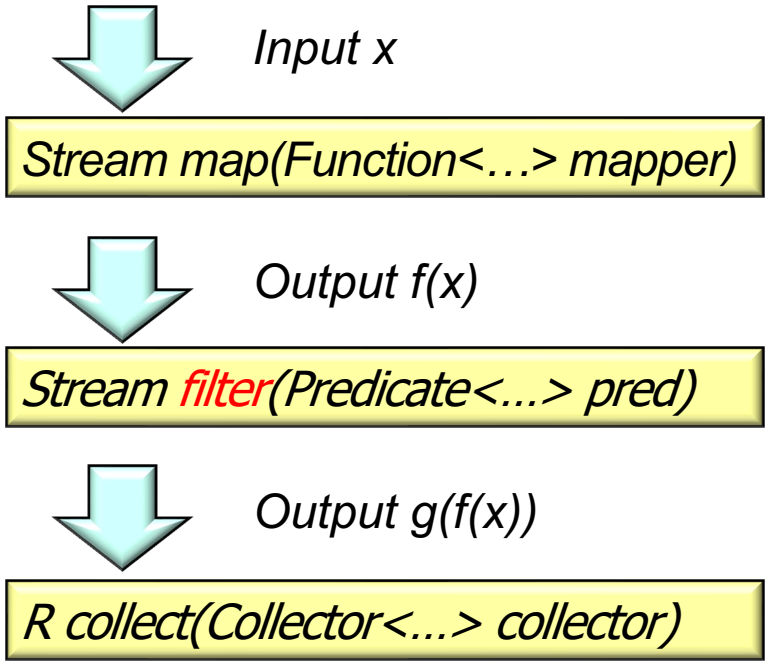


See stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078

Overview of the filter() Intermediate Operation

Overview of the filter() Intermediate Operation

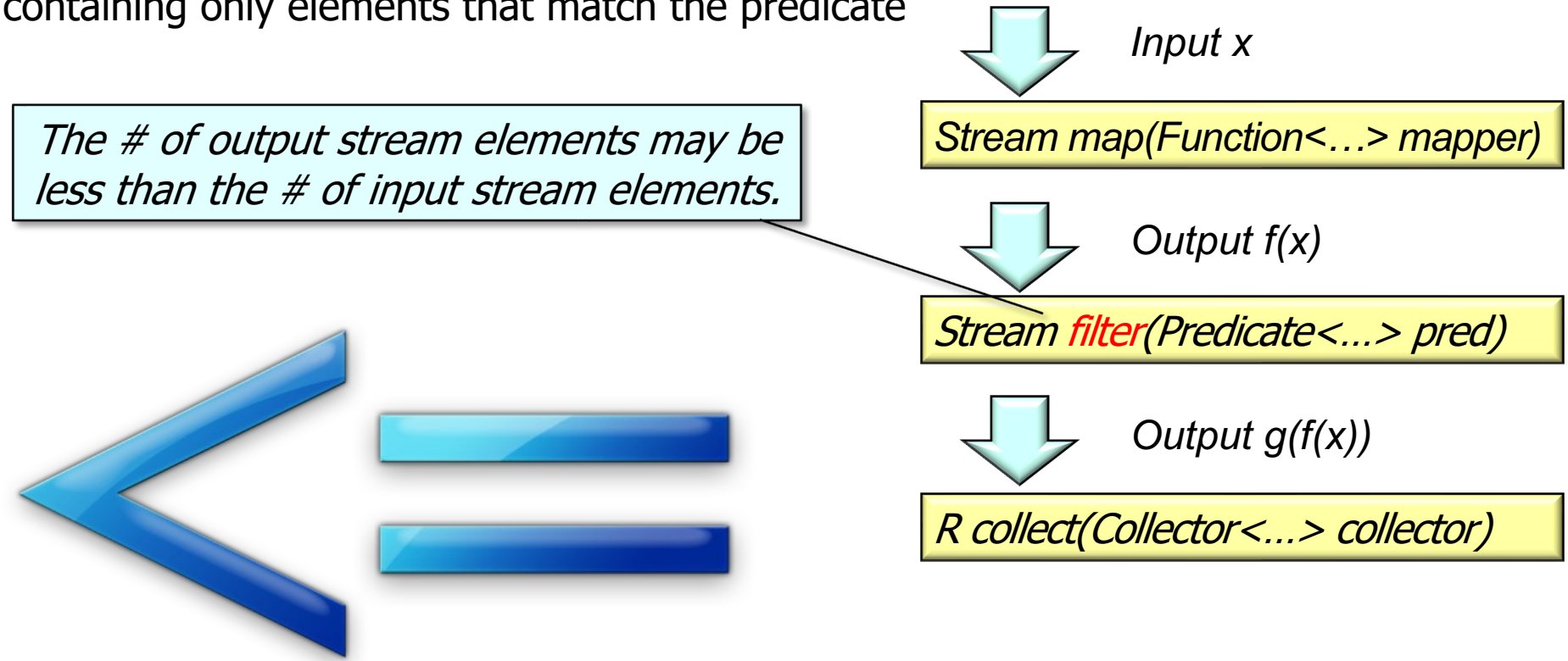
- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate



See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter

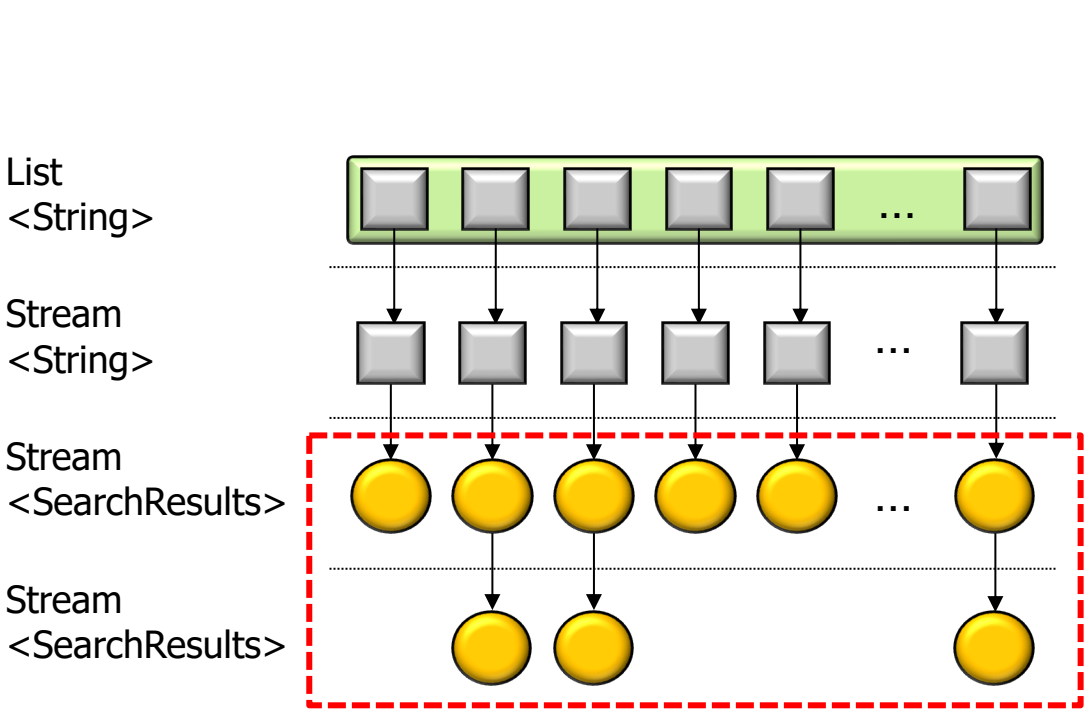
Overview of the filter() Intermediate Operation

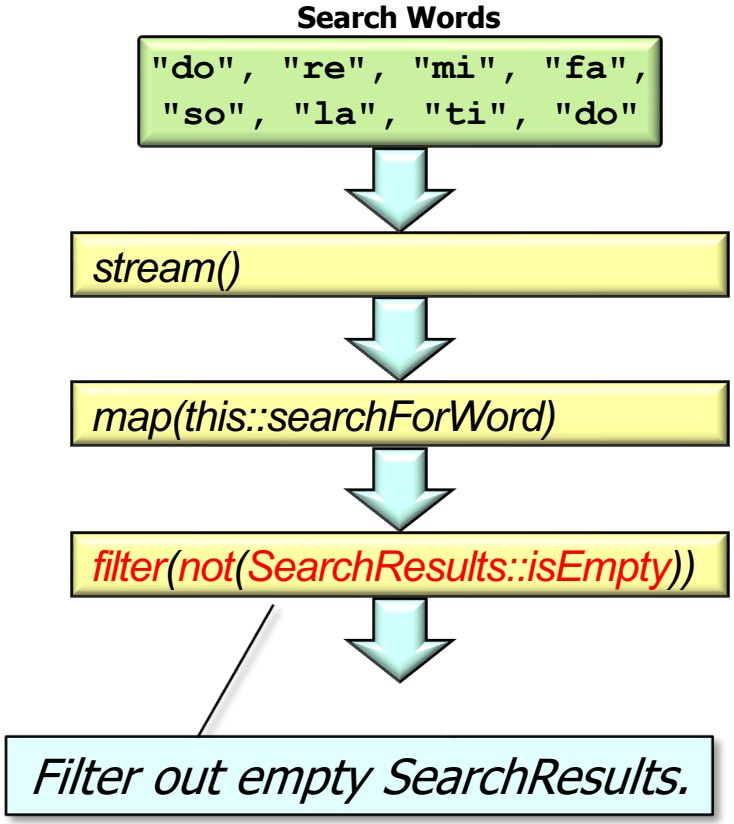
- Tests a predicate against each element of input stream & returns an output stream containing only elements that match the predicate



Overview of the filter() Intermediate Operation

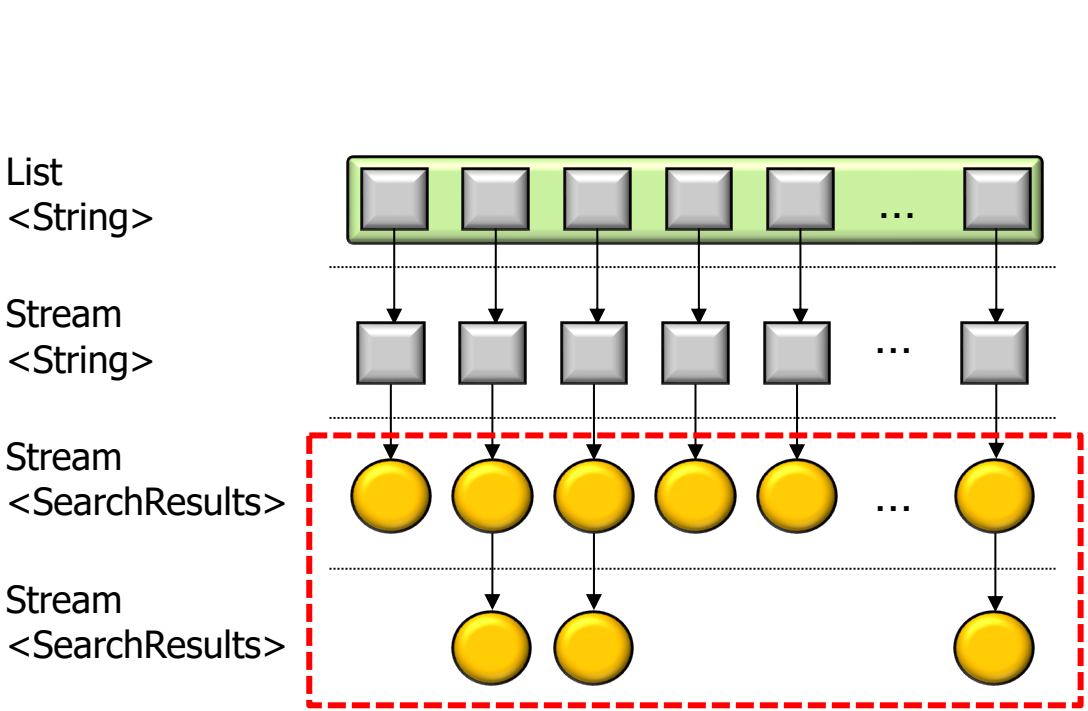
- Example of applying filter() & a predicate in the SimpleSearchStream program

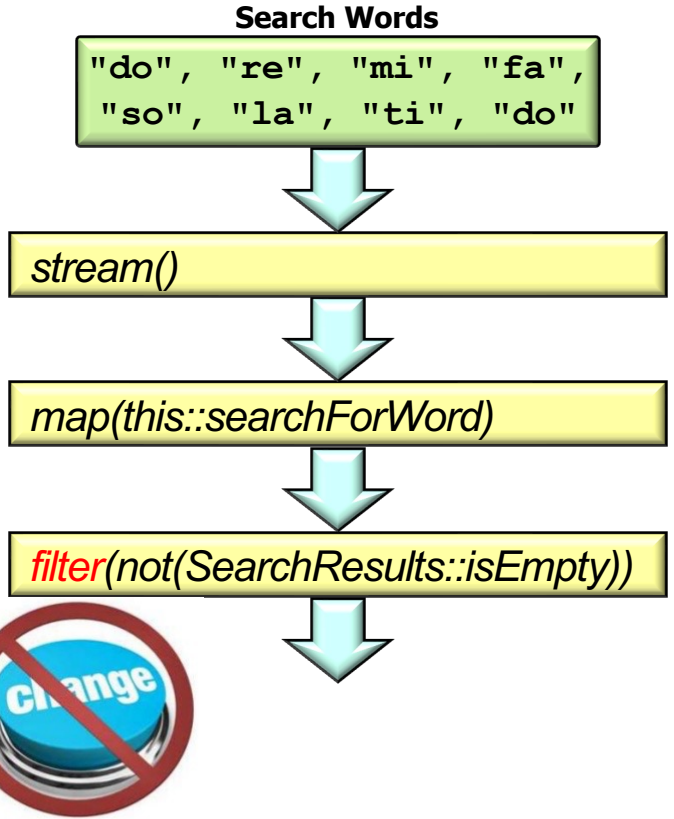




Overview of the filter() Intermediate Operation

- Example of applying filter() & a predicate in the SimpleSearchStream program



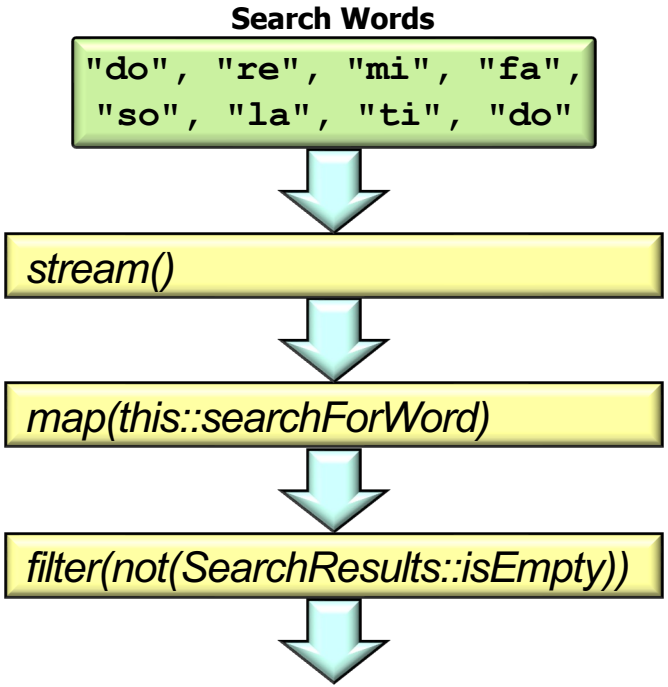


filter() *can't* change the type or value of elements it processes

Overview of the filter() Intermediate Operation

- Example of applying filter() & a predicate in the SimpleSearchStream program

```
List<SearchResults> results =
  wordsToFind
     .stream()
     .map(this::searchForWord)
     .filter(not
        (SearchResults::isEmpty))
     .toList();
```



Again, note the fluent interface style.

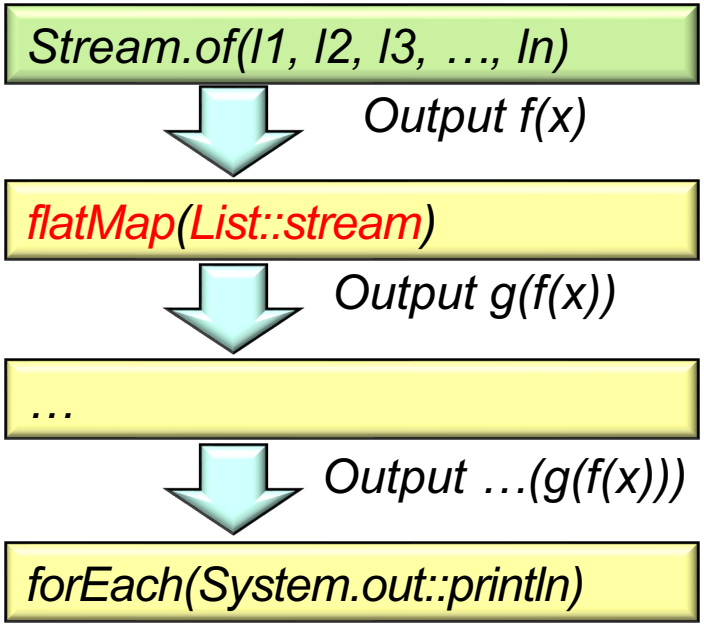
See en.wikipedia.org/wiki/Fluent_interface

Overview of the flatMap() Intermediate Operation

Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element

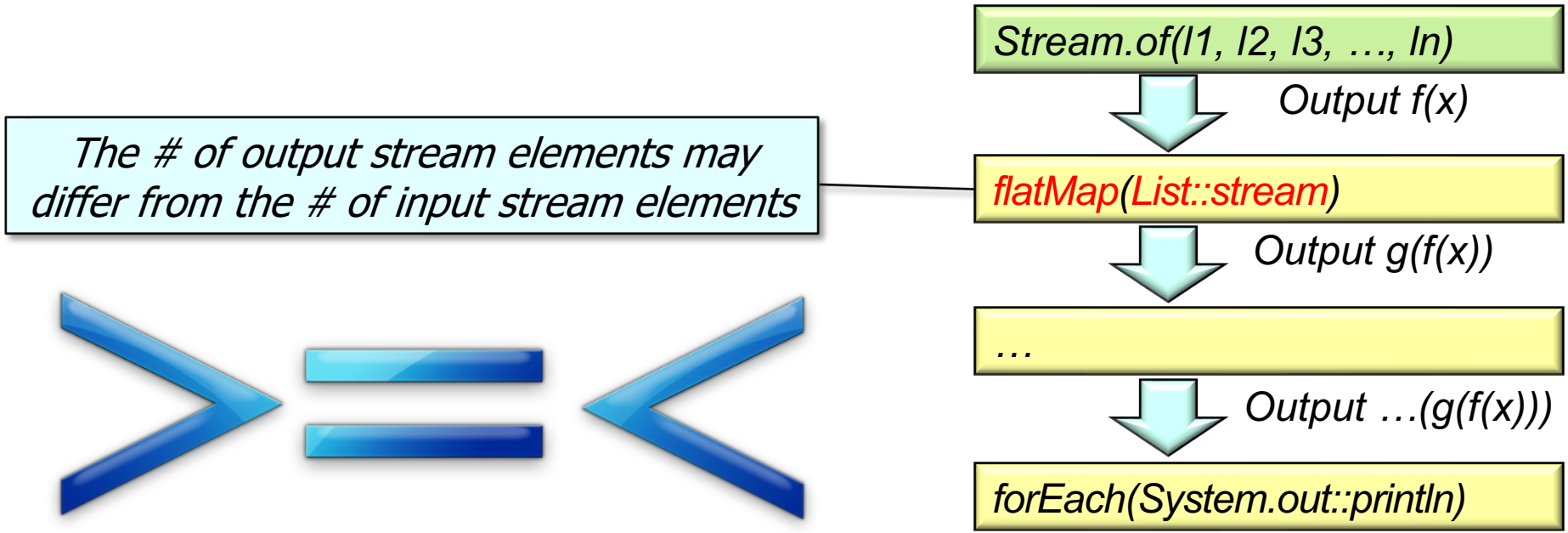
This definition sounds like map() at first glance, but there are important differences!



See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#flatMap

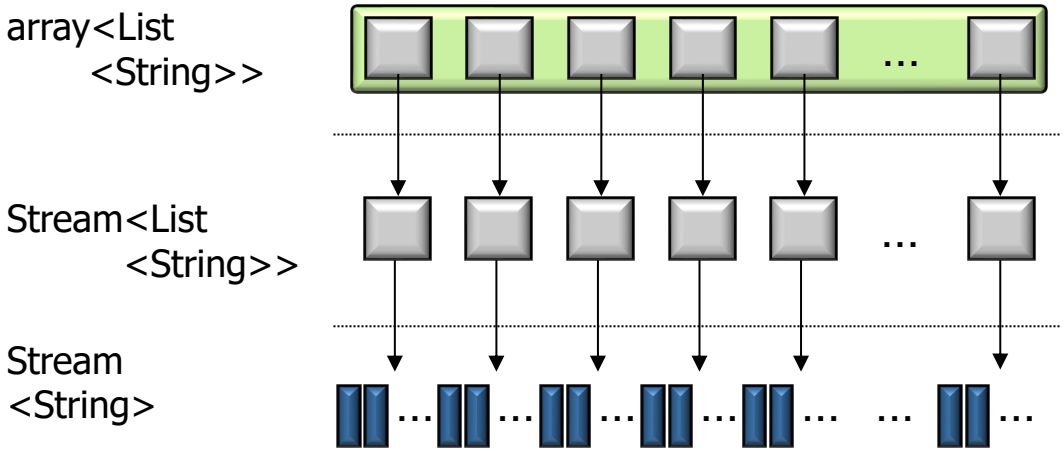
Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element



Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element



"Flatten" an array of lists of strings into a stream of strings

Stream.of(l1, l2, l3, …, ln)

Output f(x)

flatMap(List::stream)

Output g(f(x))

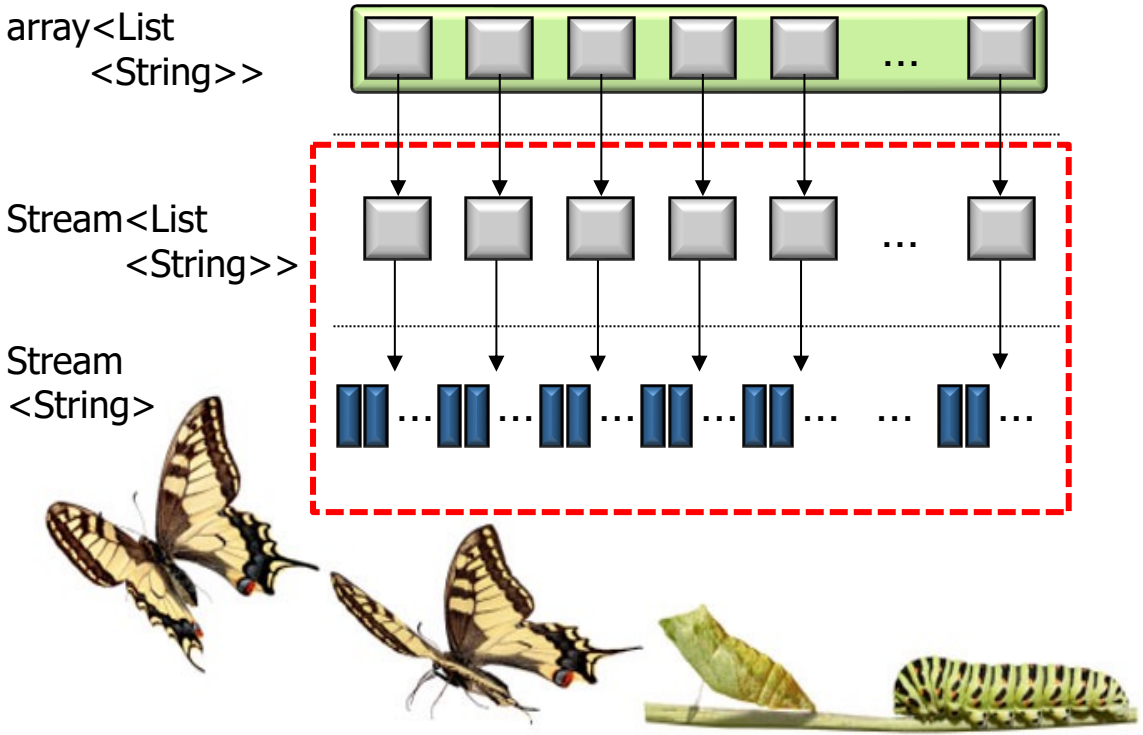
…

Output …(g(f(x)))

forEach(System.out::println)

Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element



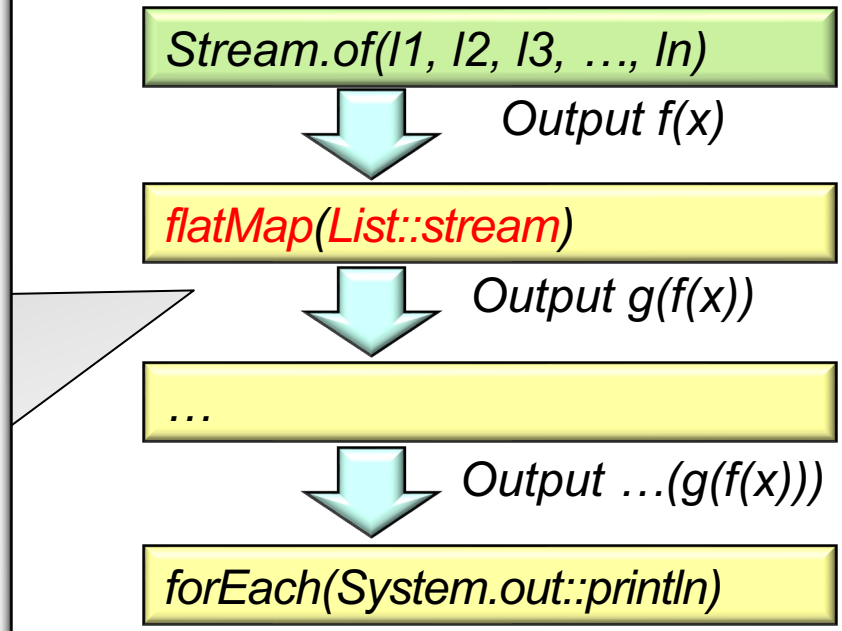
flatMap() *may* transform the type of elements it processes

Overview of the flatMap() Intermediate Operation

- Returns a stream that replaces each stream element w/contents of a mapped stream produced by applying the provided mapping function to each element

```
List<String> l1 = ...;
List<String> l2 = ...;
...
List<String> ln = ...;

Stream
   .of(l1, l2, l3, ..., ln)
   .flatMap(List::stream)
   .filter(s -> toLowerCase
     (s.charAt(0)) == 'h')
   ...
   .forEach(System.out::println);
```



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12

A Limitation with flatMap()

A Limitation with flatMap()

- A limitation with the flatMap() implementation forces sequential processing

This code always runs sequentially for "inner streams" that use flatMap()

```
<R> Stream<R> flatMap
  (Function<? super P_OUT,
   ? extends Stream<? extends R>>
   mapper) {
...
 public void accept(P_OUT u) {
   try(Stream<? extends R> result
       = mapper.apply(u)) {
   if (result != null) {
     if (...) {
       result
       .sequential()
       .forEach(downstream);
     ...
}
```

See stackoverflow.com/questions/45038120/parallel-flatmap-always-sequential/66386078

A Limitation with flatMap()

- A limitation with the flatMap() implementation forces sequential processing

```
List<Integer> list = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .flatMap(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .toList();
```

Due to a limitation with flatMap() this inner stream will always run sequentially, even though it is explicitly designated as .parallel()

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex35

A Limitation with flatMap()

- A simple workaround is to use replace flatMap() with map() & reduce(Stream::concat)

```
List<Integer> list = IntStream
   .rangeClosed(1, outerCount)
   .boxed()
   .parallel()

   .map(innerCount -> IntStream
      .rangeClosed(1, innerCount)
      .boxed()
      .parallel())

   .reduce(Stream::concat)
   .orElse(Stream.empty())

   .toList();
```

This inner stream now runs in parallel, as intended

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex35

End of Java Streams Intermediate Operations filter() & flatMap()