Overview of Java Stream Internals: Construction & Execution

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

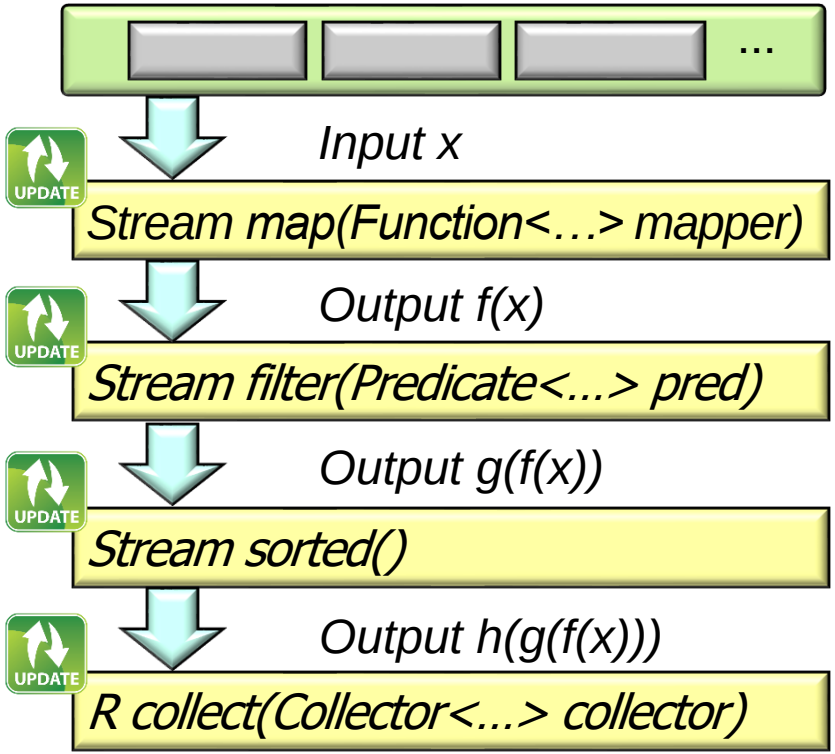
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

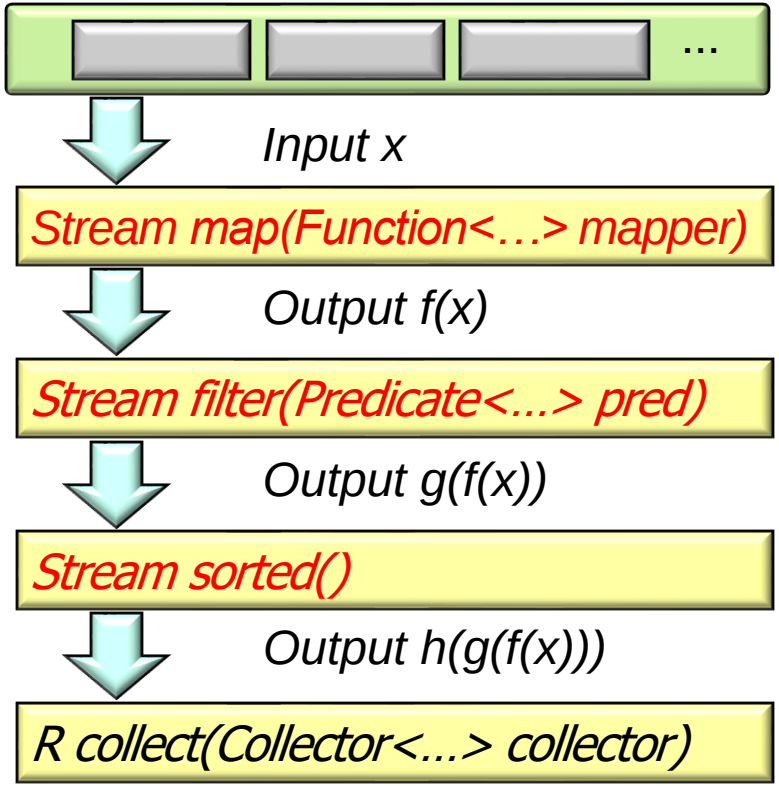
- Understand parallel stream internals, e.g.
  - Know what can change & what can't
  - Recognize how a Java stream is constructed & executed



Java Stream Construction & Execution

Java Stream Construction & Execution

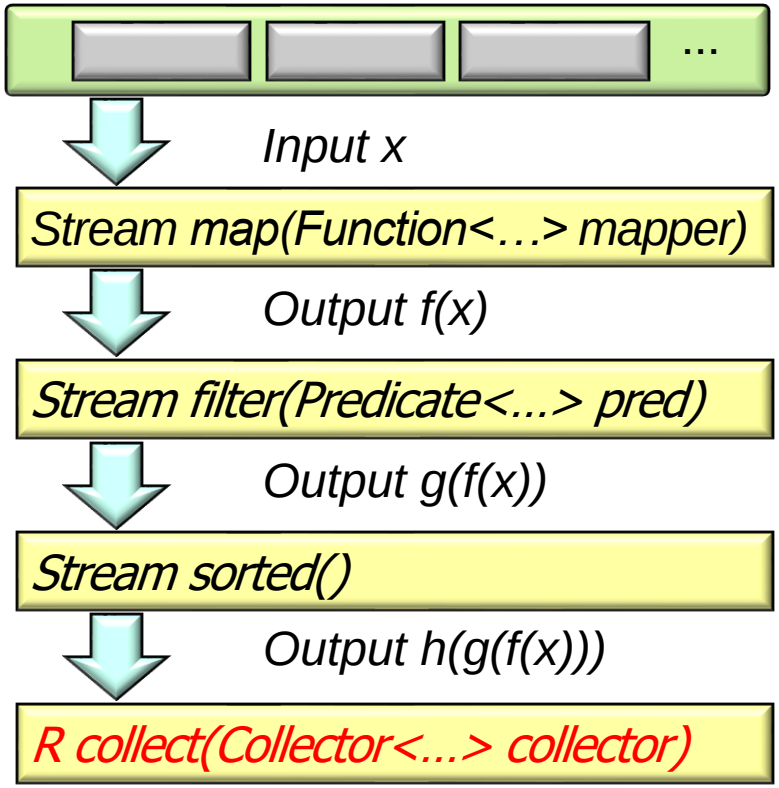
- Recall that intermediate operations are "lazy"



See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation

Java Stream Construction & Execution

- Recall that intermediate operations are "lazy"
  - i.e., they don't start to run until a terminal operator is reached



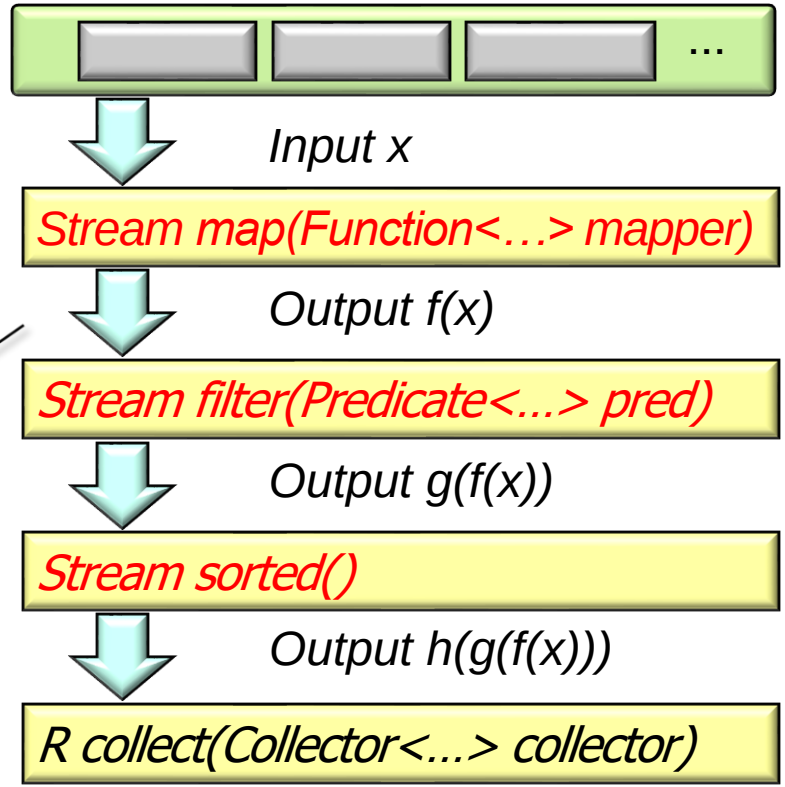
See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation

Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation

```
List<String> ls = ...
List<String> sortedAWords = ls
   .stream()
   .map(String::toUpperCase)
   .filter(s ->
            s.startsWith("A"))
   .sorted()
   .collect(toList());
```

At runtime a linked list of stream source & intermediate operations is built, one per "stage" in pipeline

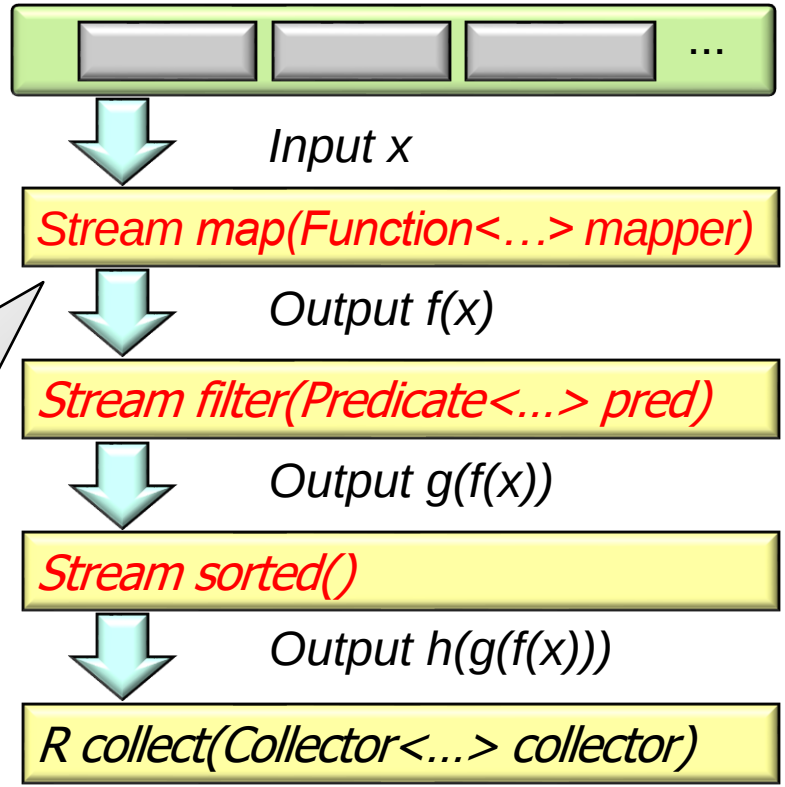


See www.ibm.com/developerworks/library/j-java-streams-3-brian-goetz/index.html#N1014E

Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally

Stream Flag	Interpretation
SIZED	Size of stream is known
DISTINCT	Elements of stream are distinct
SORTED	Elements of the stream are sorted in natural order
ORDERED	Stream has meaningful encounter order

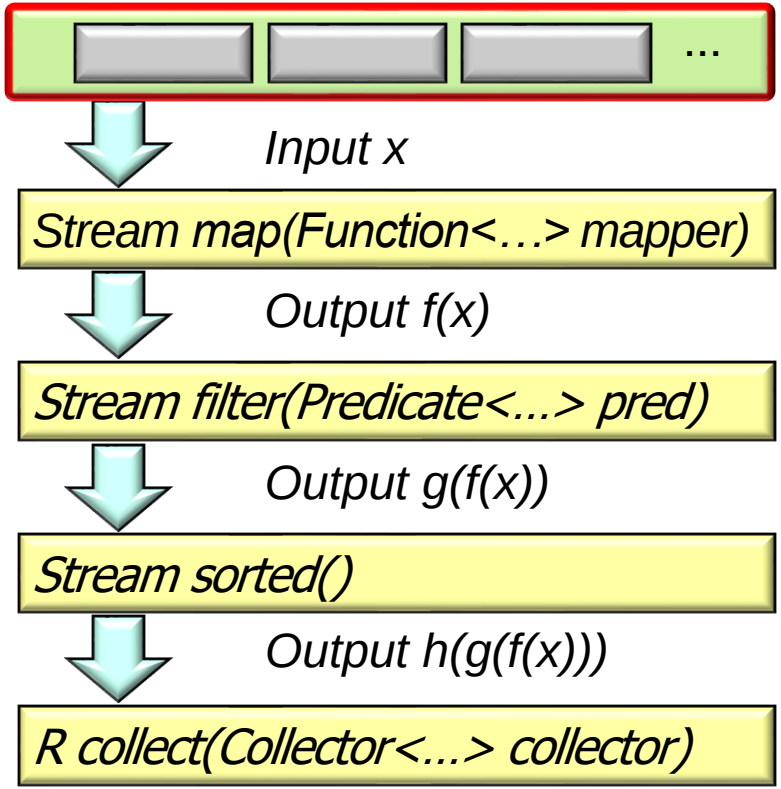


These flags are a subset of the flags that can be defined by a spliterator

Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics, e.g.

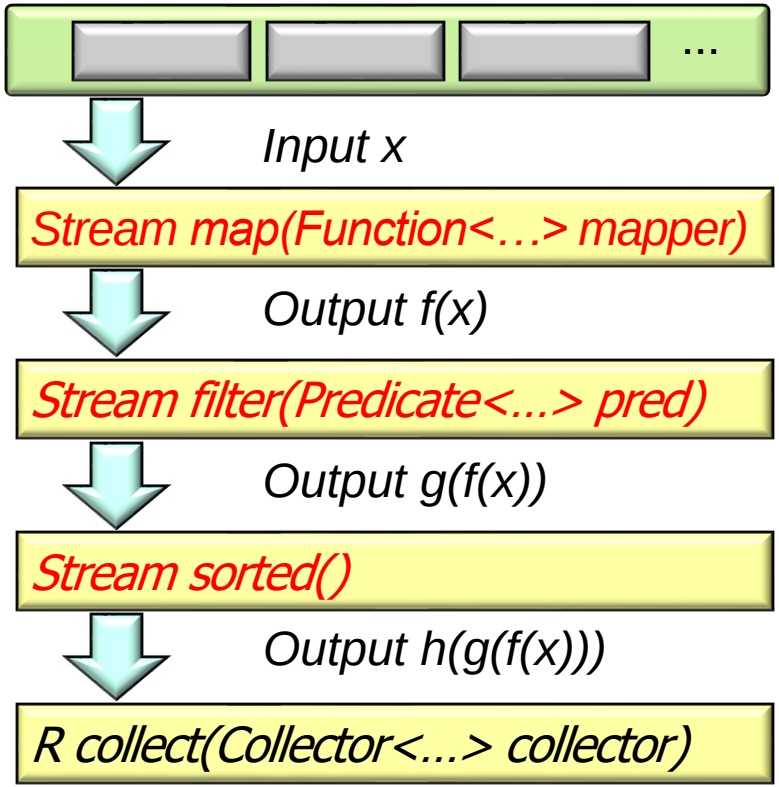
Collection	Sized	Ordered	Sorted	Distinct
ArrayList	✓	✓		
HashSet	✓			✓
TreeSet	✓	✓	✓	✓



Stream generate() & iterate() methods create streams that are *not* sized!

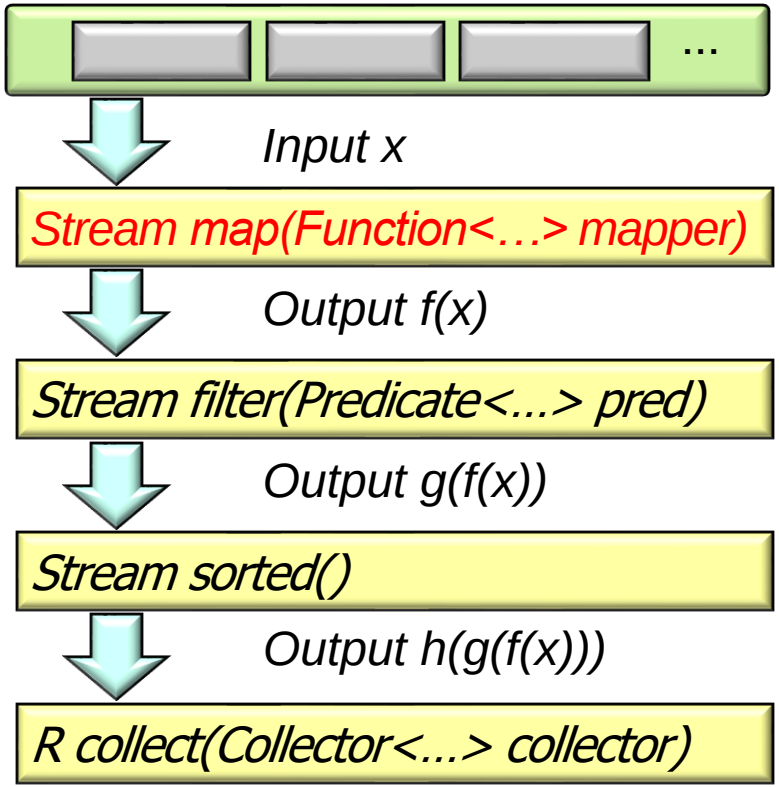
Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags



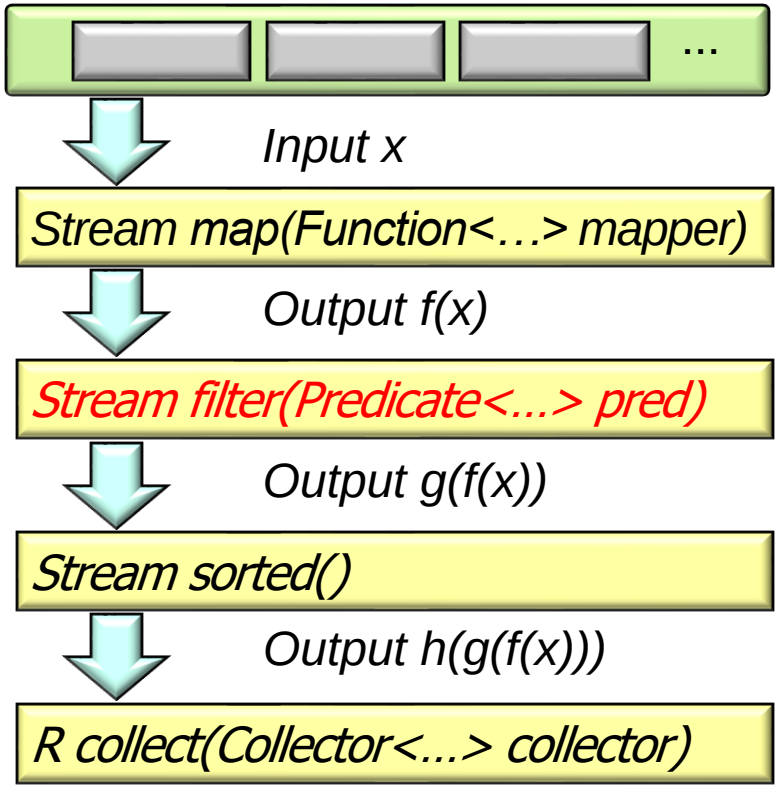
Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags, e.g.
    - map()
      - Clears SORTED & DISTINCT but keeps SIZED



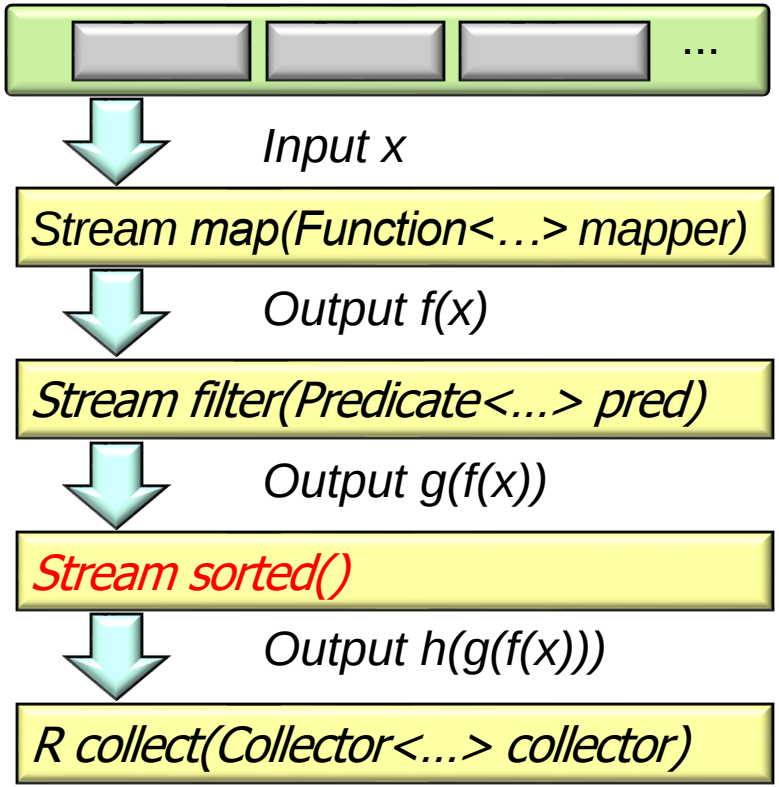
Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags, e.g.
    - map()
    - filter()
      - Keeps SORTED & DISTINCT but clears SIZED



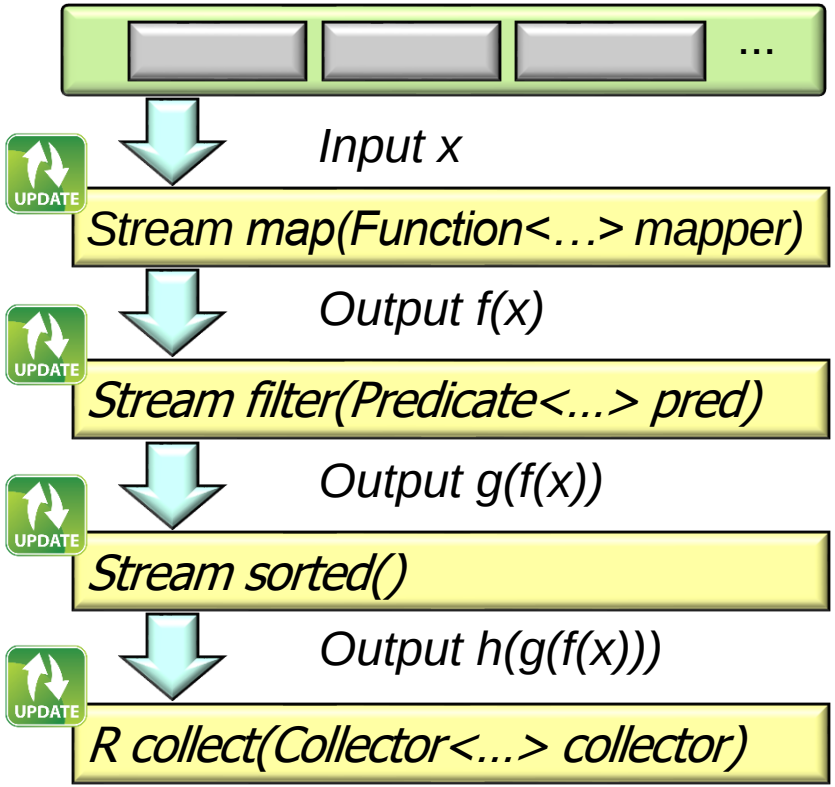
Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags, e.g.
    - map()
    - filter()
    - sorted()
      - Keeps SIZED & DISTINCT & adds SORTED



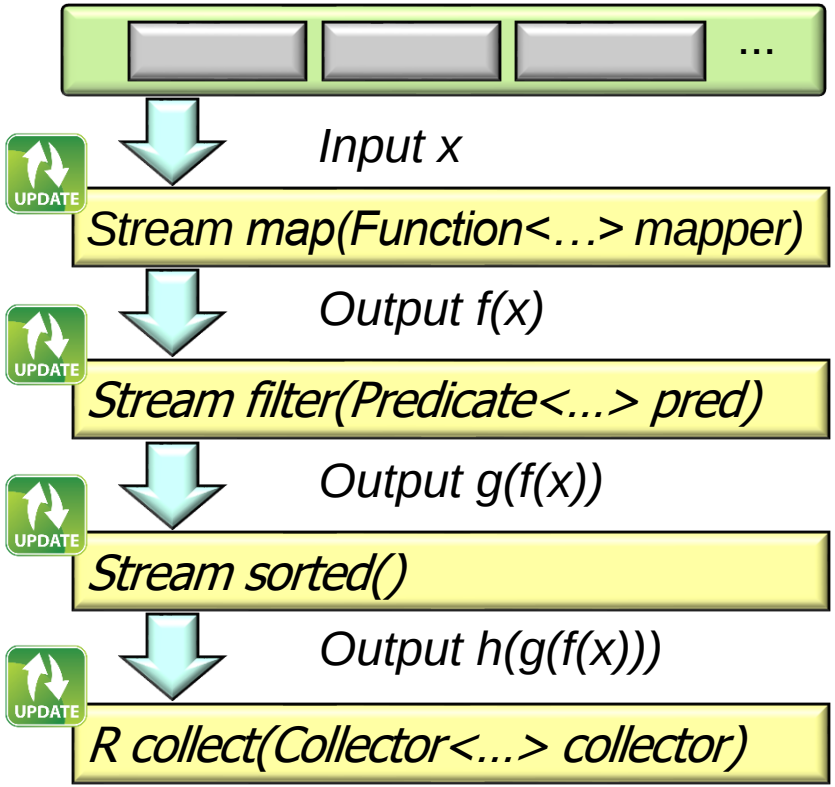
Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags
  - The flags at each stage are updated as the pipeline is being constructed



Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags
  - The flags at each stage are updated as the pipeline is being constructed
    - e.g., flags for a previous stage are combined with the current stage's behavior to derive a new set of flags



Java Stream Construction & Execution

- A stream pipeline is constructed at runtime via an internal representation
  - Each pipeline stage is described by a bitmap of *stream flags* internally
  - Source stage stream flags are derived from spliterator characteristics
  - Each intermediate operation affects the stream flags
  - The flags at each stage are updated as the pipeline is being constructed
    - e.g., flags for a previous stage are combined with the current stage's behavior to derive a new set of flags

```
Set<String> ts =
  new TreeSet<>(...);

List<String> sortedAWords =
  ts
    .stream()
    .filter(s ->
              s.startsWith("a"))
    .sorted()
    .collect(toList());
```

SORTED

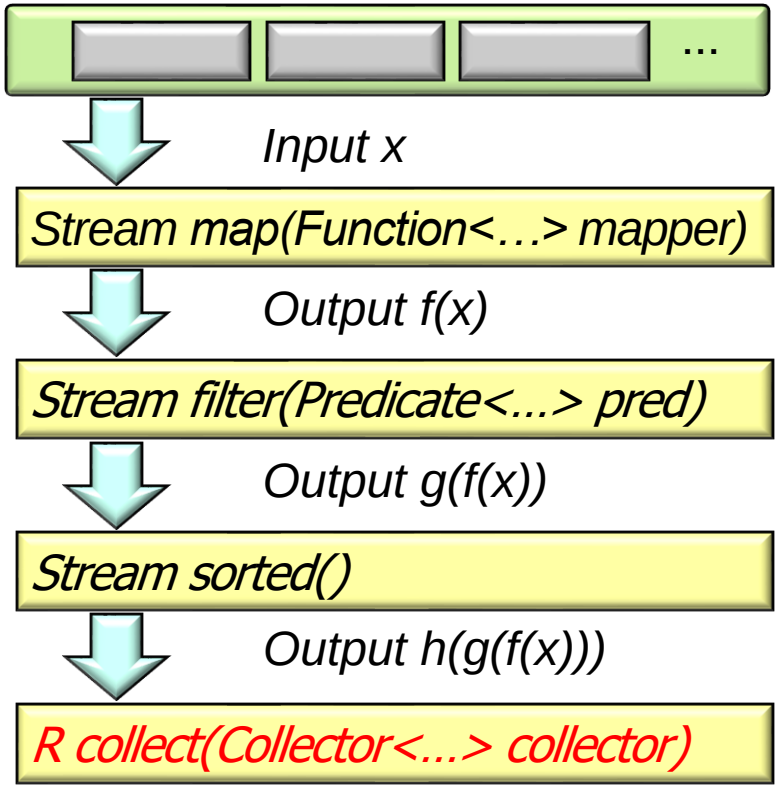
SORTED

SORTED

The streams framework removes redundant operations since the source is already sorted

Java Stream Construction & Execution

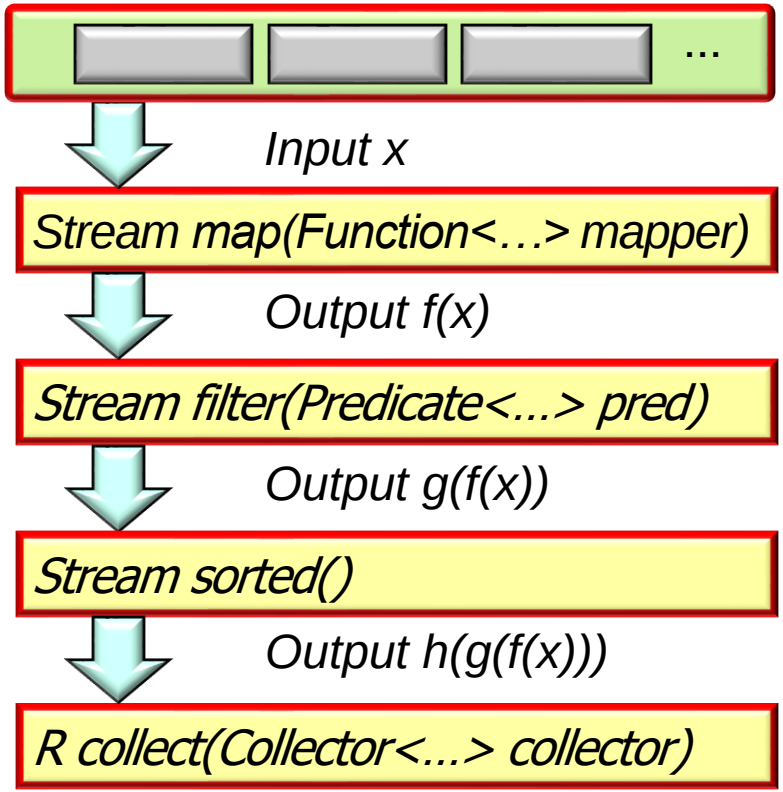
- When terminal operation runs the stream framework picks an execution plan



See www.ibm.com/developerworks/library/j-java-streams-3-brian-goetz/index.html#N101F6

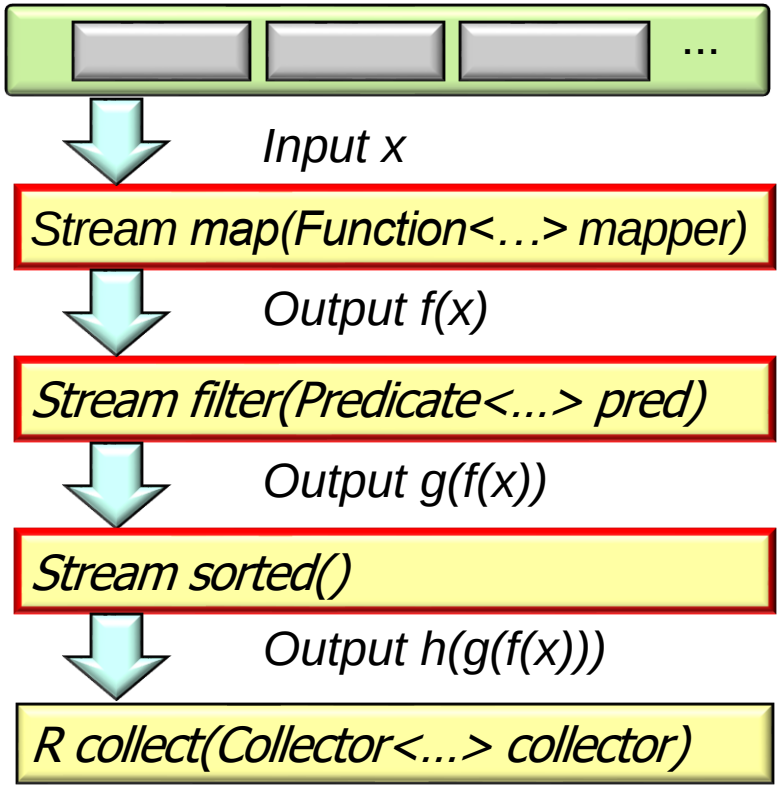
Java Stream Construction & Execution

- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations



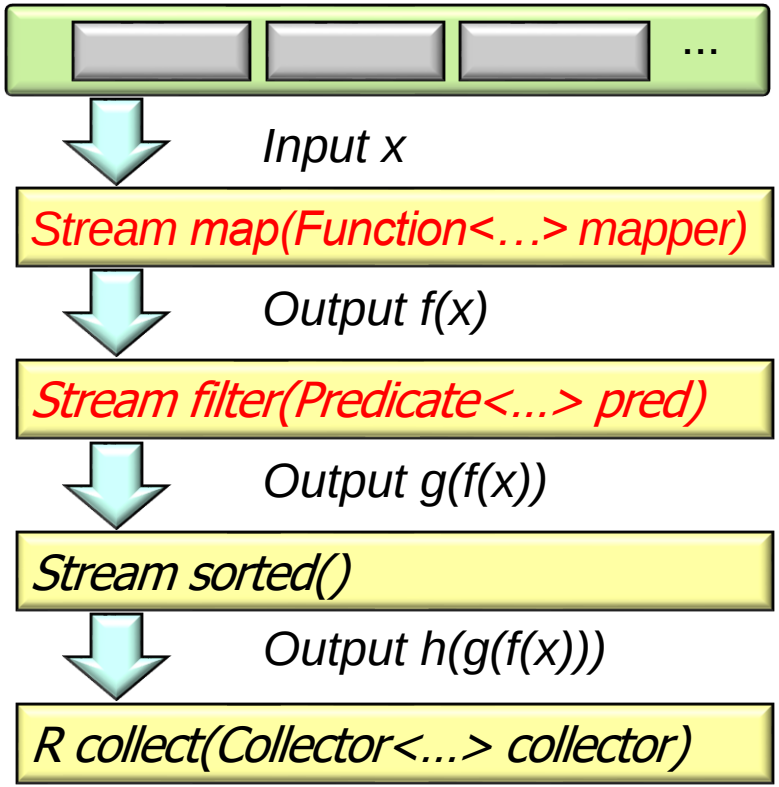
Java Stream Construction & Execution

- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories



Java Stream Construction & Execution

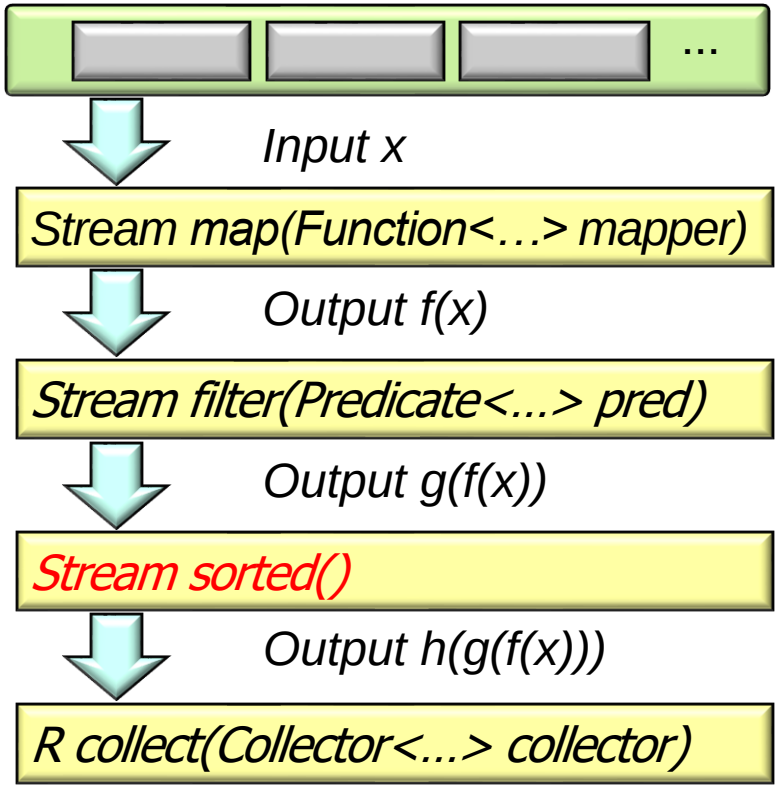
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories:
    - Stateless
      - e.g., filter(), map(), flatMap(), etc.



A pipeline with only stateless operations runs in one pass (even if it's parallel)

Java Stream Construction & Execution

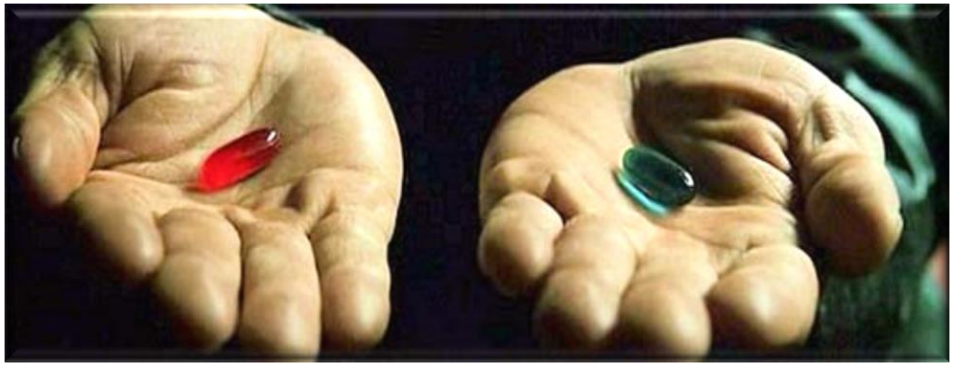
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories:
    - Stateless
    - Stateful
      - e.g., sorted(), limit(), distinct(), dropWhile(), etc.

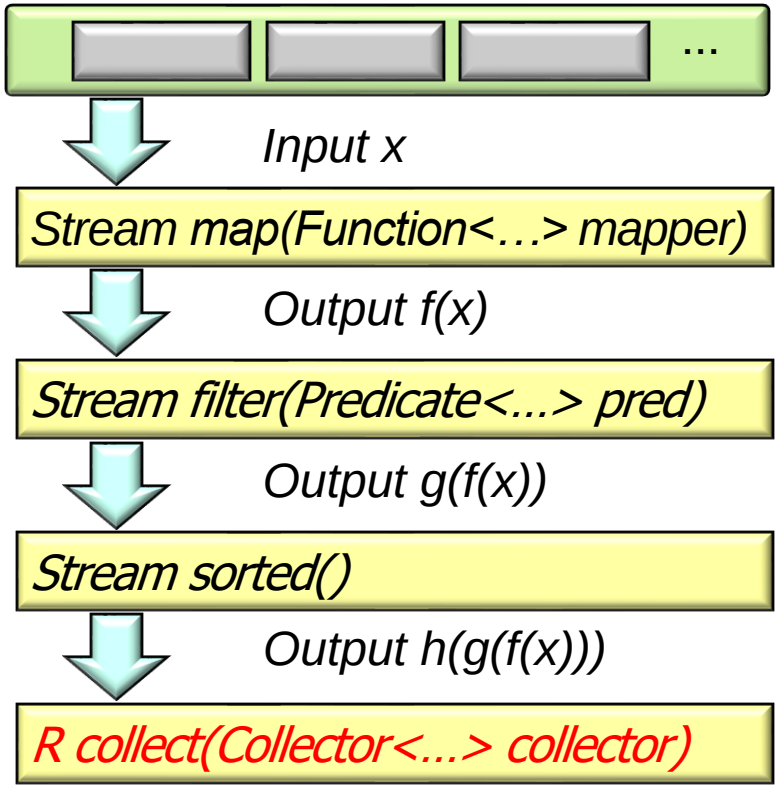


A pipeline with stateful operations is divided into sections & runs in multiple passes

Java Stream Construction & Execution

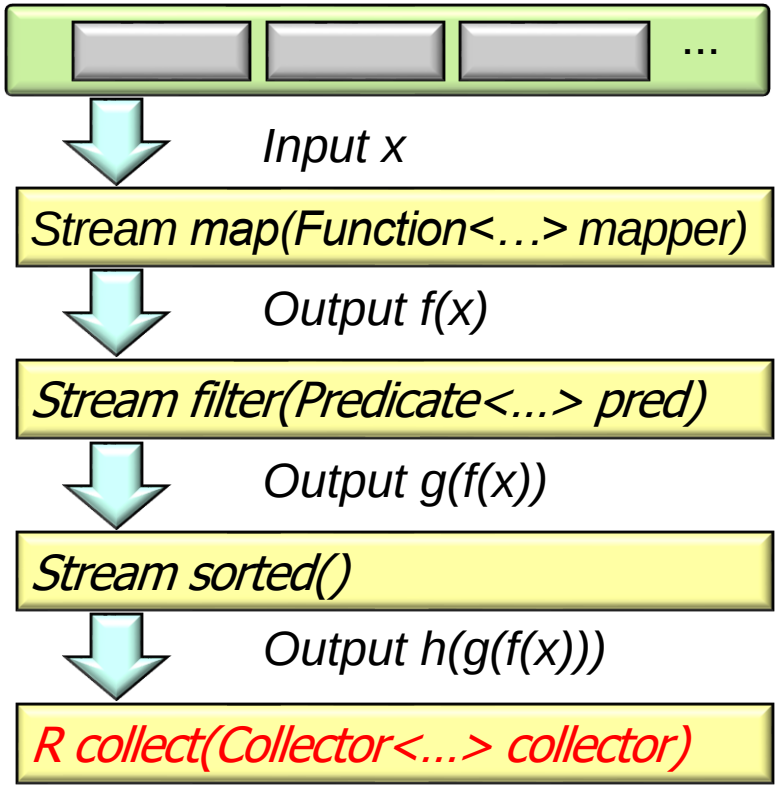
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories





Java Stream Construction & Execution

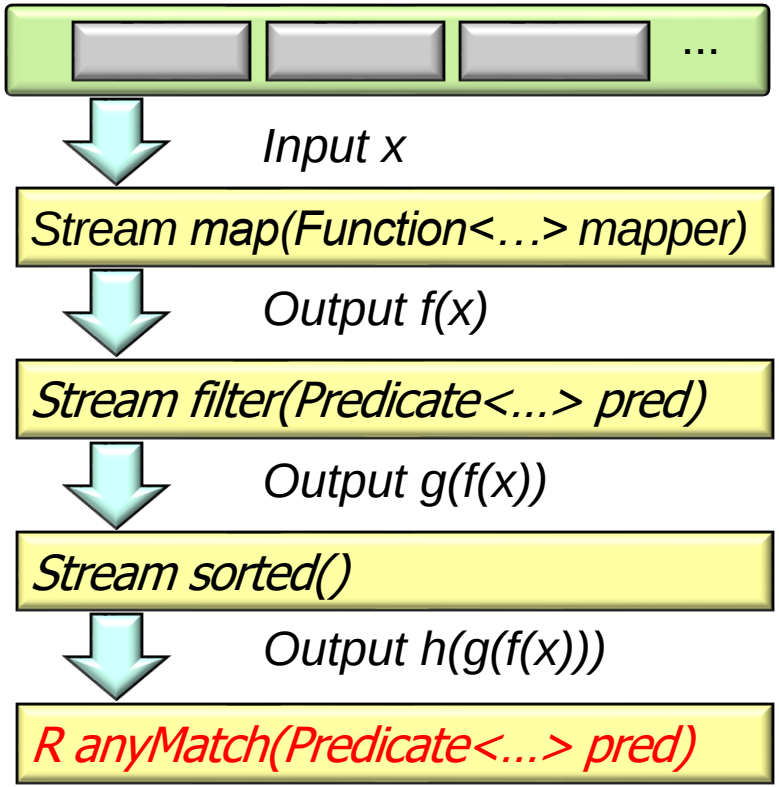
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories
    - Non-short-circuiting
      - e.g., reduce(), collect(), forEach(), etc.



Terminal operation can process data in bulk using spliterator's forEachRemaining()

Java Stream Construction & Execution

- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories
    - Non-short-circuiting
    - Short-circuiting
      - e.g., anyMatch(), findFirst(), etc.



Terminal operation must process data one element at a time using tryAdvance()

End of Overview of Java Stream Internals: Construction & Execution