Java 8 Parallel Stream Internals (Part 6)

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

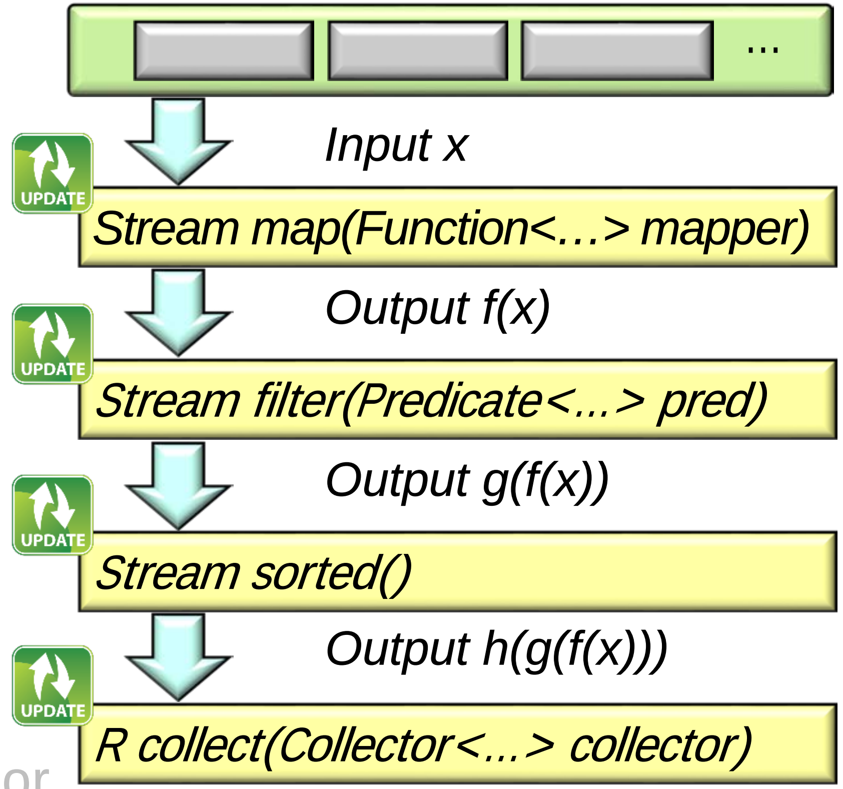
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

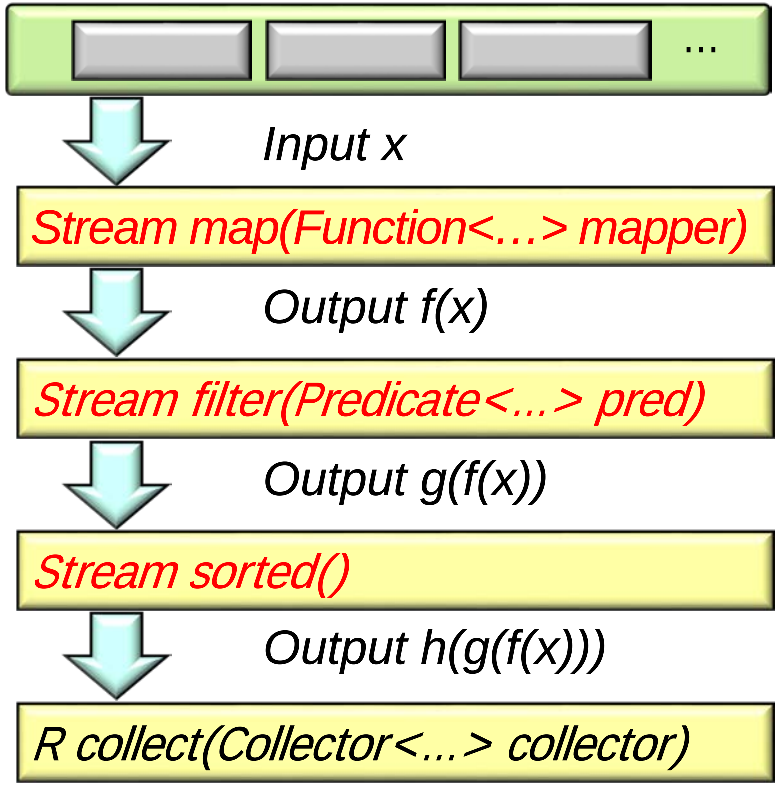
- Understand parallel stream internals, e.g.
  - Know what can change & what can't
  - Partition a data source into "chunks"
  - Process chunks in parallel
  - Configure the Java 8 parallel stream common fork-join pool
  - Avoid pool starvation & improve performance w/ManagedBlocker
  - Perform a reduction that combines partial results into a single result
  - Learn to implement a concurrent collector
  - Recognize how a parallel stream is constructed & executed


...
Input x
UPDATE
Stream map(Function<...> mapper)
Output f(x)
UPDATE
Stream filter(Predicate<...> pred)
Output g(f(x))
UPDATE
Stream sorted()
Output h(g(f(x)))
UPDATE
R collect(Collector<...> collector)

Parallel Stream Construction & Execution

Parallel Stream Construction & Execution

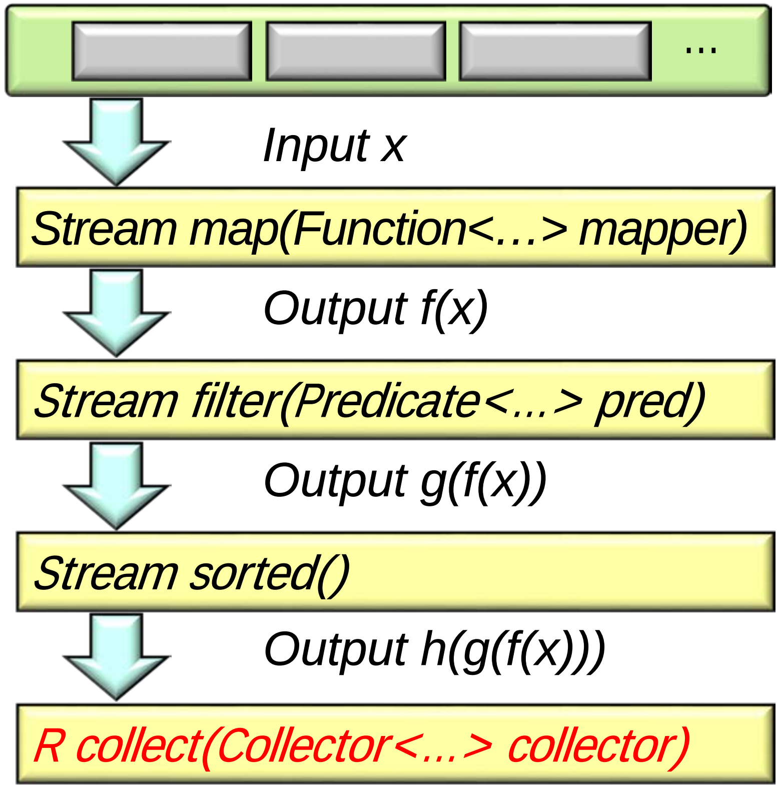
- Recall that intermediate operations are "lazy"


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel Stream Construction & Execution

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


...
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)

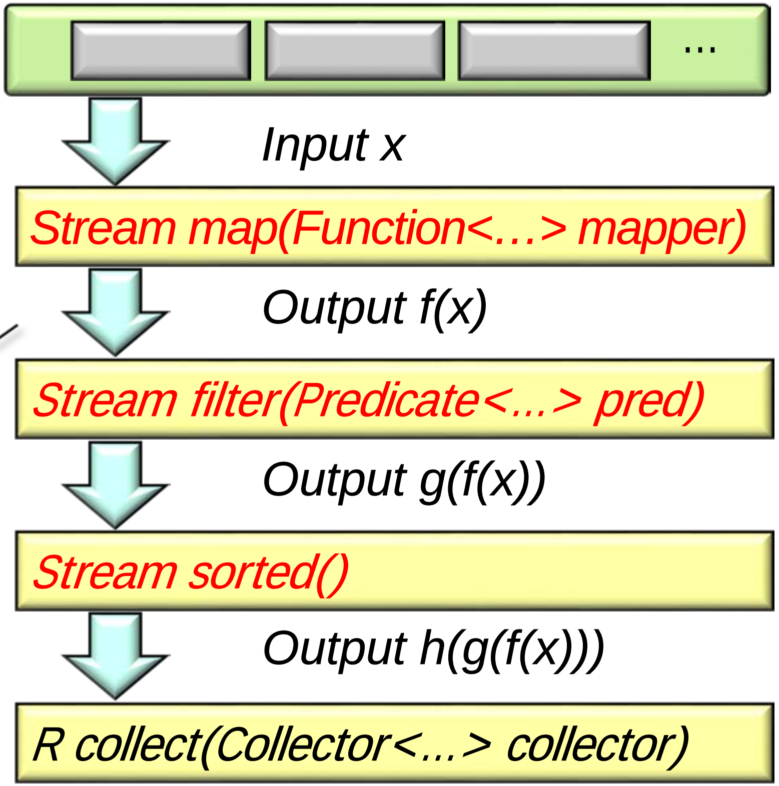
See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation

Parallel 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 build, one per "stage" in pipeline

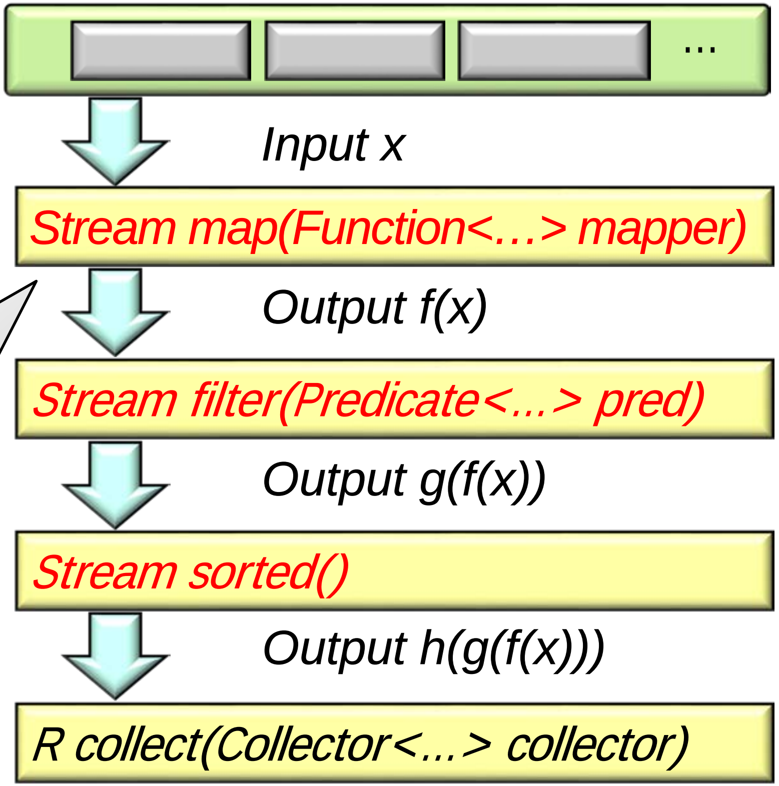

...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel 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

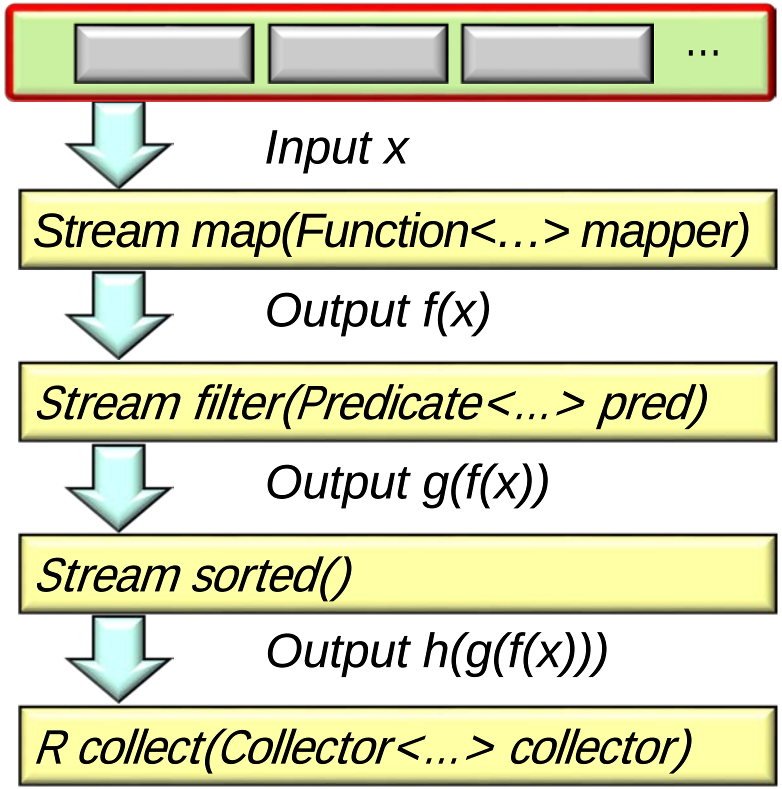

...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel 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.

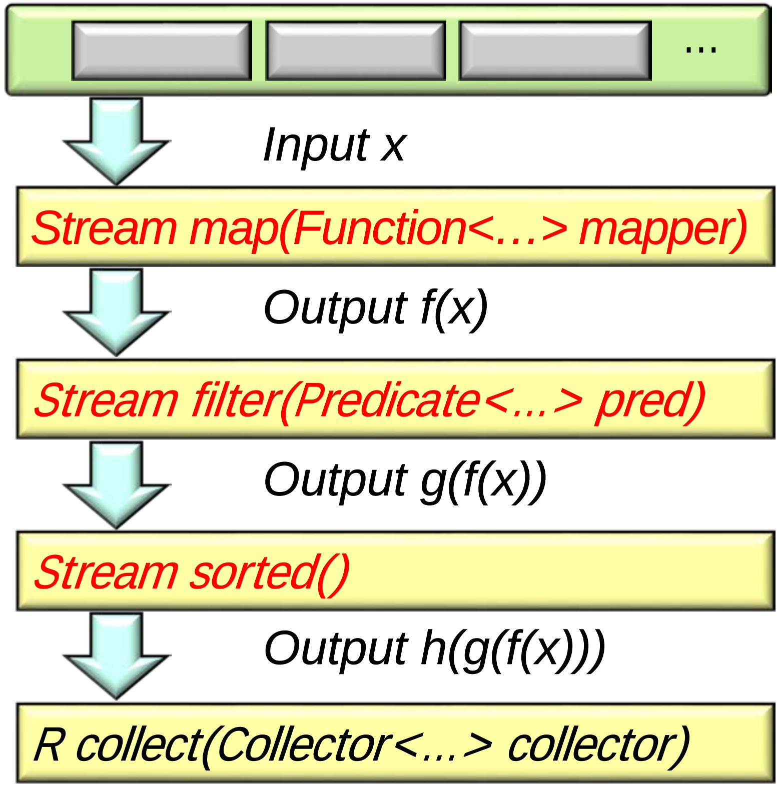
Collection	Sized	Ordered	Sorted	Distinct
ArrayList	✓	✓		
HashSet	✓			✓
TreeSet	✓	✓	✓	✓


...
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)

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

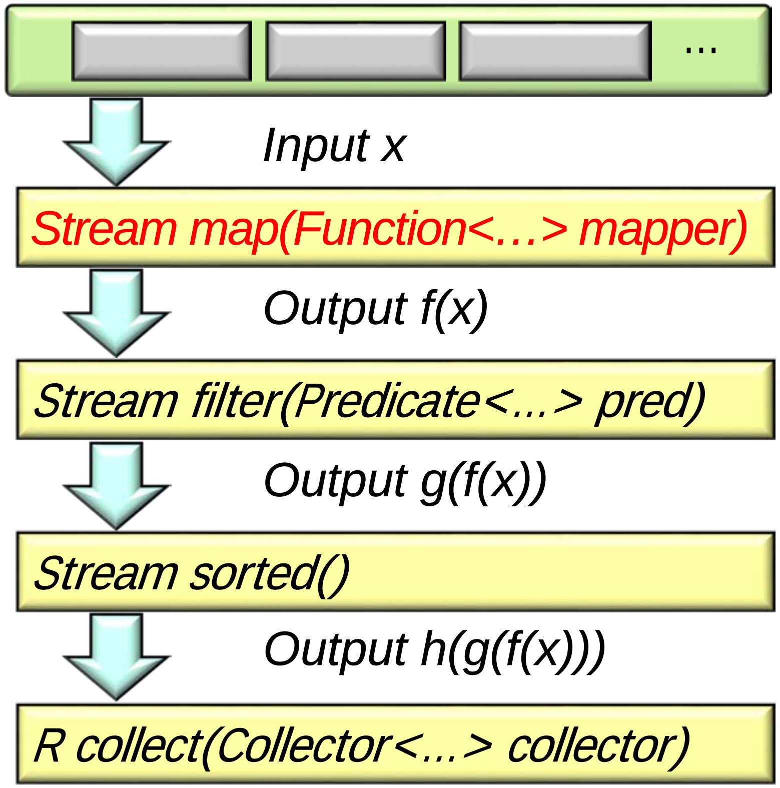
Parallel Stream Construction & Execution

- 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


...
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)

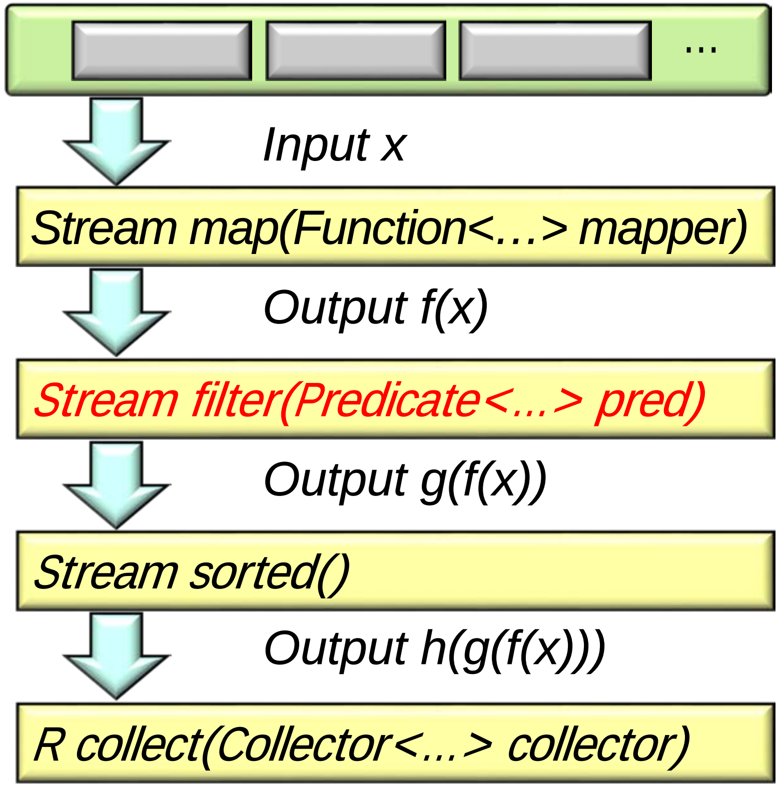
Parallel Stream Construction & Execution

- 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


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

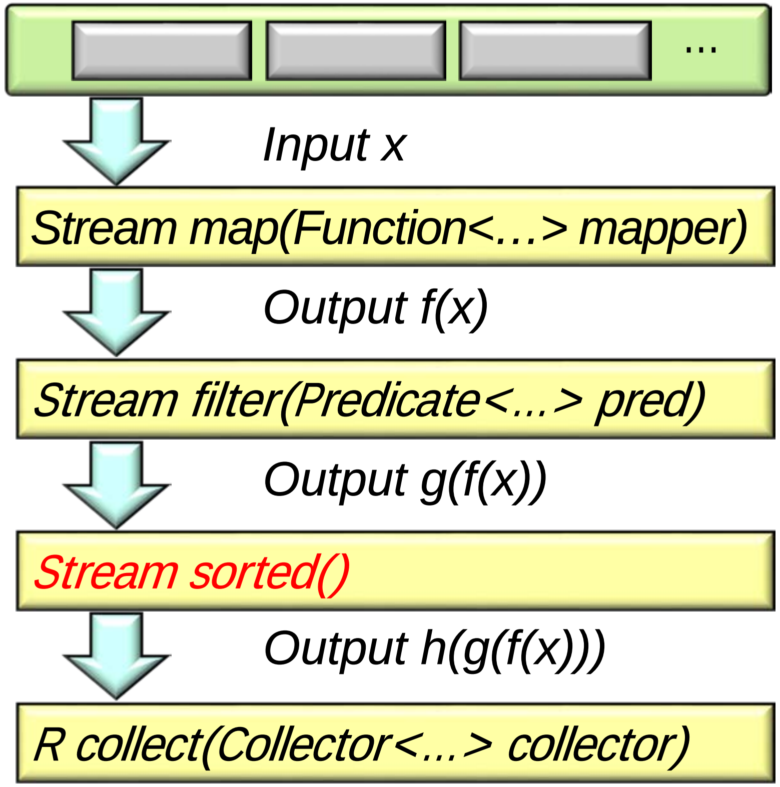
Parallel Stream Construction & Execution

- 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


...
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)

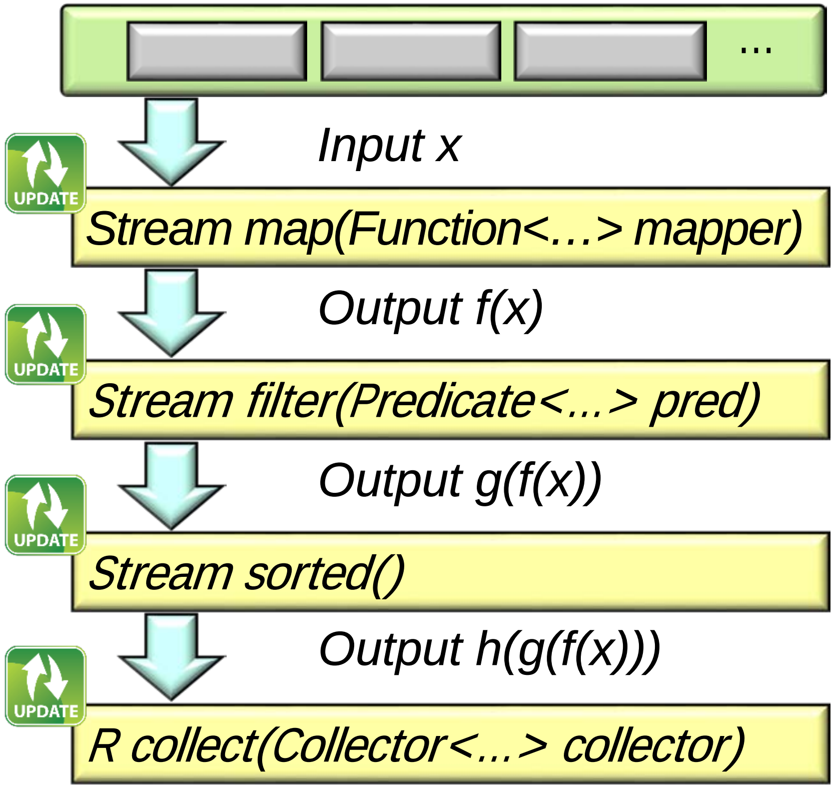
Parallel Stream Construction & Execution

- 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


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

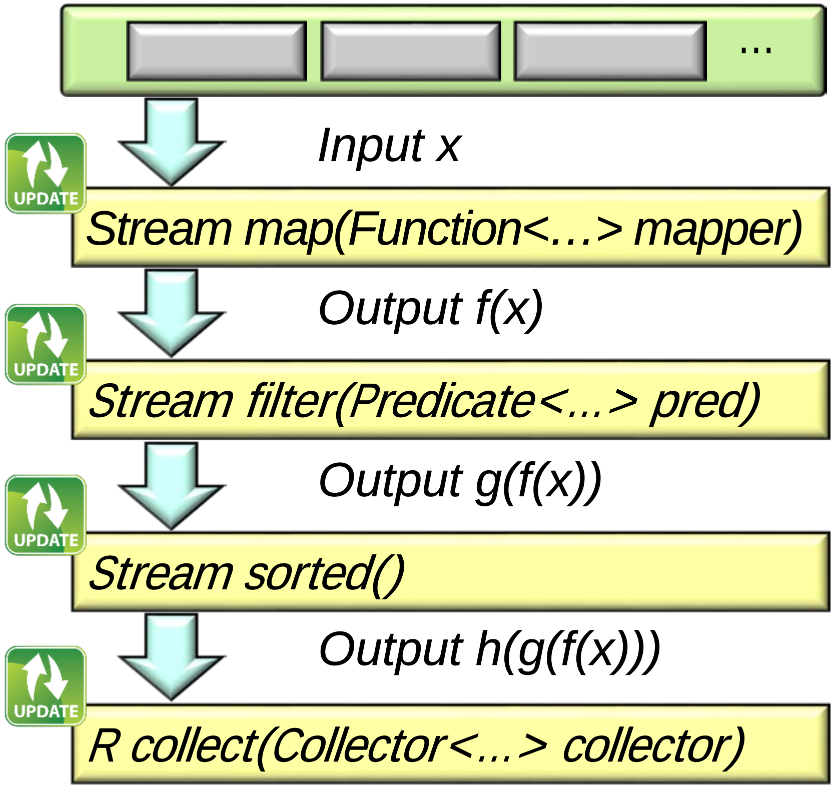
Parallel Stream Construction & Execution

- 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
  - As the pipeline is being constructed the flags at each stage are updated


...
Input x
UPDATE
Stream map(Function<…> mapper)
Output f(x)
UPDATE
Stream filter(Predicate<…> pred)
Output g(f(x))
UPDATE
Stream sorted()
Output h(g(f(x)))
UPDATE
R collect(Collector<…> collector)

Parallel 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
  - As the pipeline is being constructed the flags at each stage are updated
    - e.g., flags for a previous stage are combined with the current stage's behavior to derive a new set of flags


...
Input x
UPDATE
Stream map(Function<…> mapper)
Output f(x)
UPDATE
Stream filter(Predicate<…> pred)
Output g(f(x))
UPDATE
Stream sorted()
Output h(g(f(x)))
UPDATE
R collect(Collector<…> collector)

Parallel 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
  - As the pipeline is being constructed the flags at each stage are updated
    - e.g., flags for a previous stage are combined with the current stage's behavior to derive a new set of flags

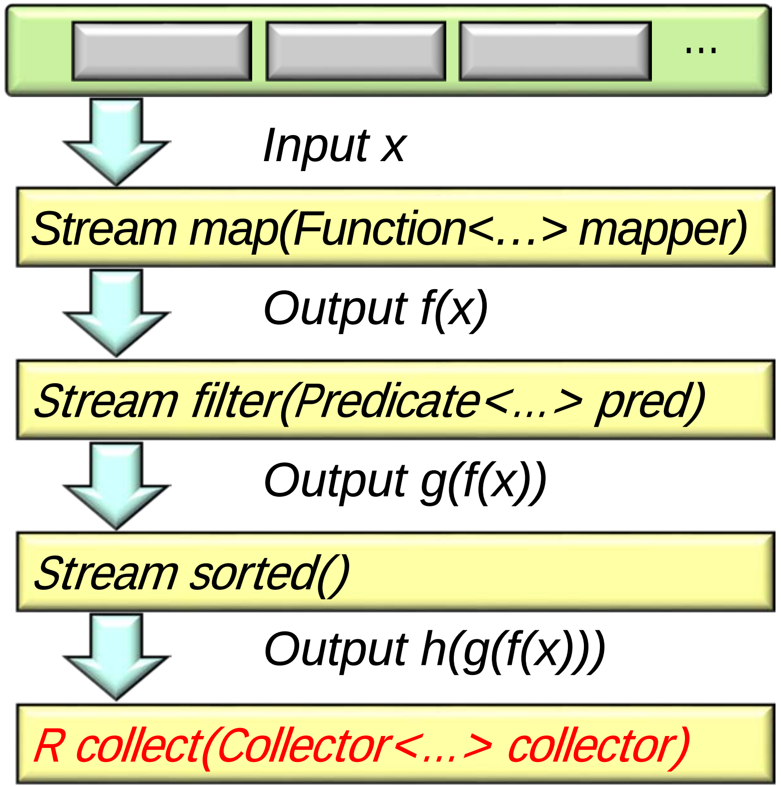
```
Set<String> ts =
  new TreeSet<>(...);

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

Redundant operation can be elided since the source is already sorted

Parallel Stream Construction & Execution

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



Stream map(Function<…> mapper)

Stream filter(Predicate<…> pred)

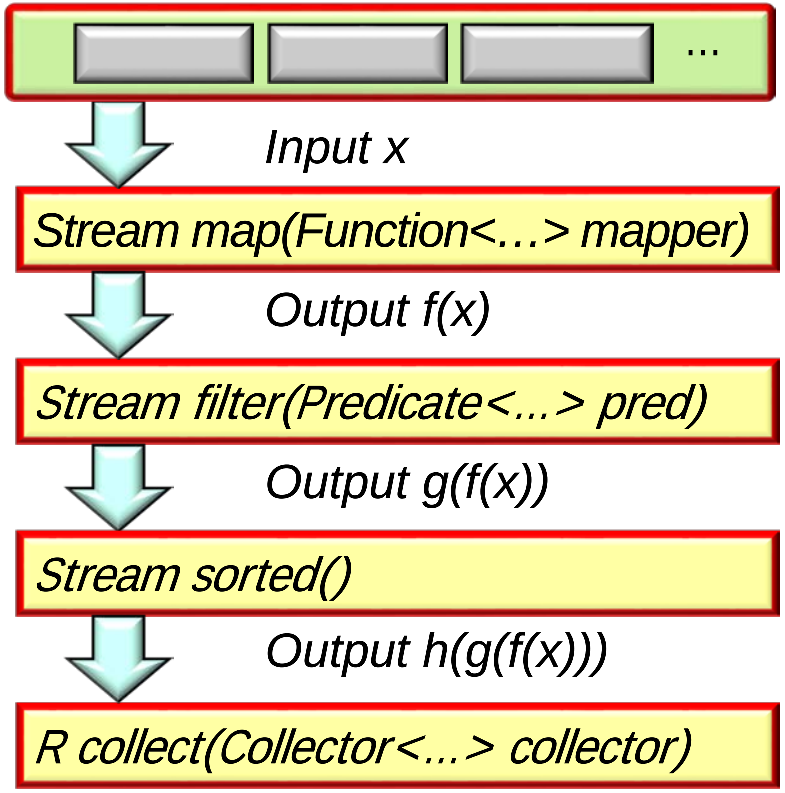
Stream sorted()

R collect(Collector<…> collector)

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

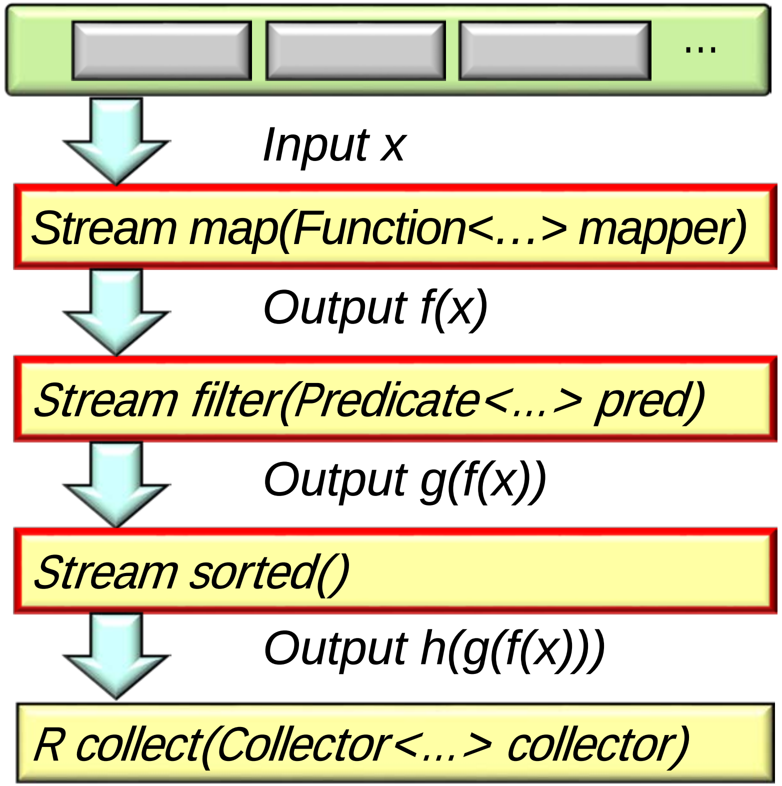
Parallel Stream Construction & Execution

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


...
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)

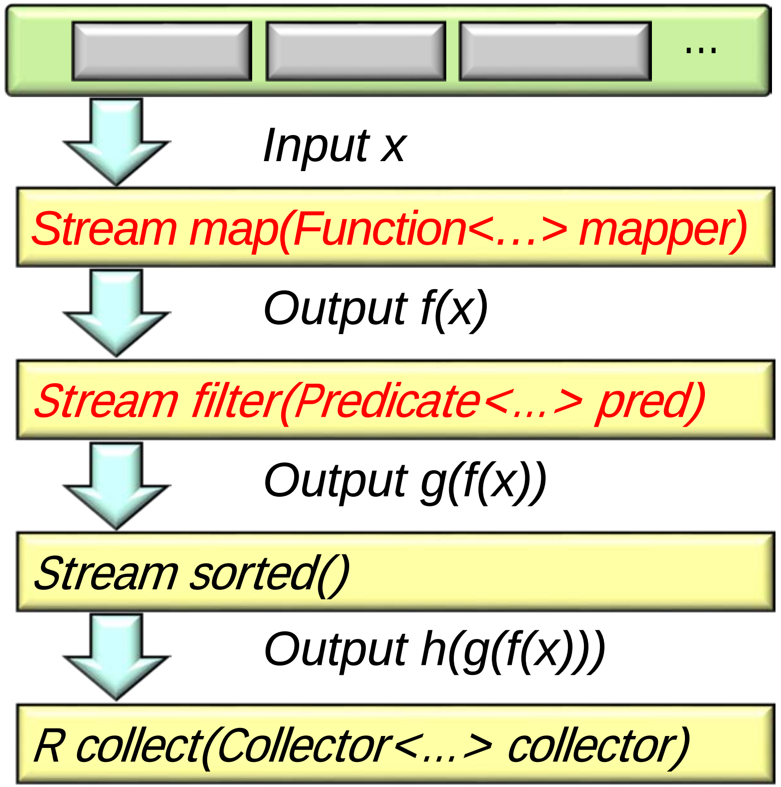
Parallel Stream Construction & Execution

- 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


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

Parallel Stream Construction & Execution

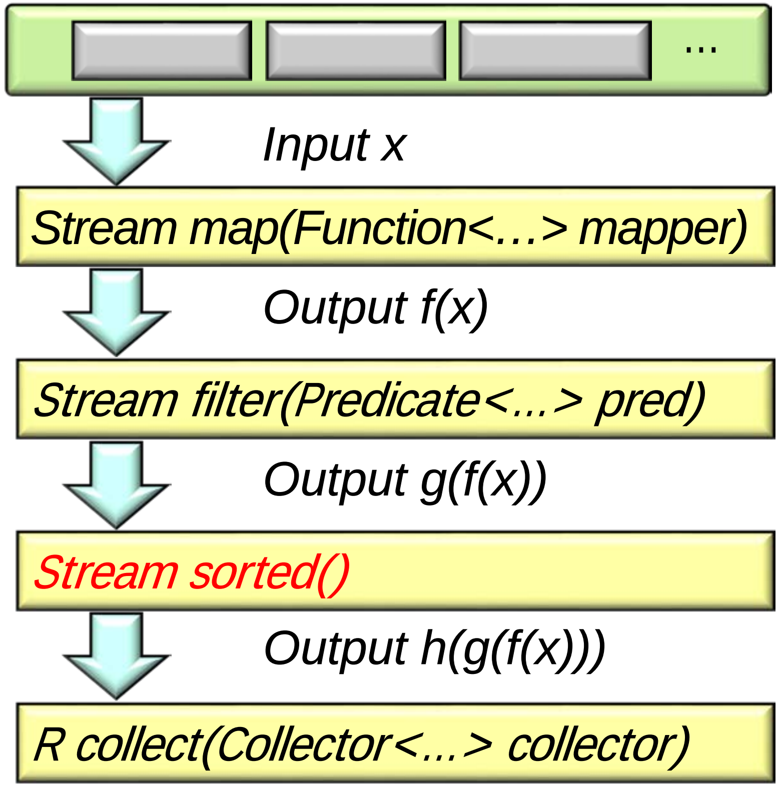
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories:
    - Stateless
      - e.g., filter(), map(), flatMap(), etc.


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel Stream Construction & Execution

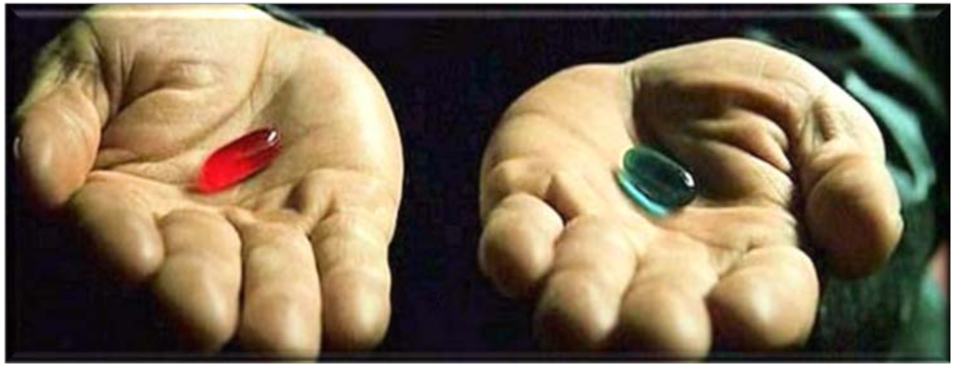
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories:
    - Stateless
    - Stateful
      - e.g., sorted(), limit(), distinct(), etc.

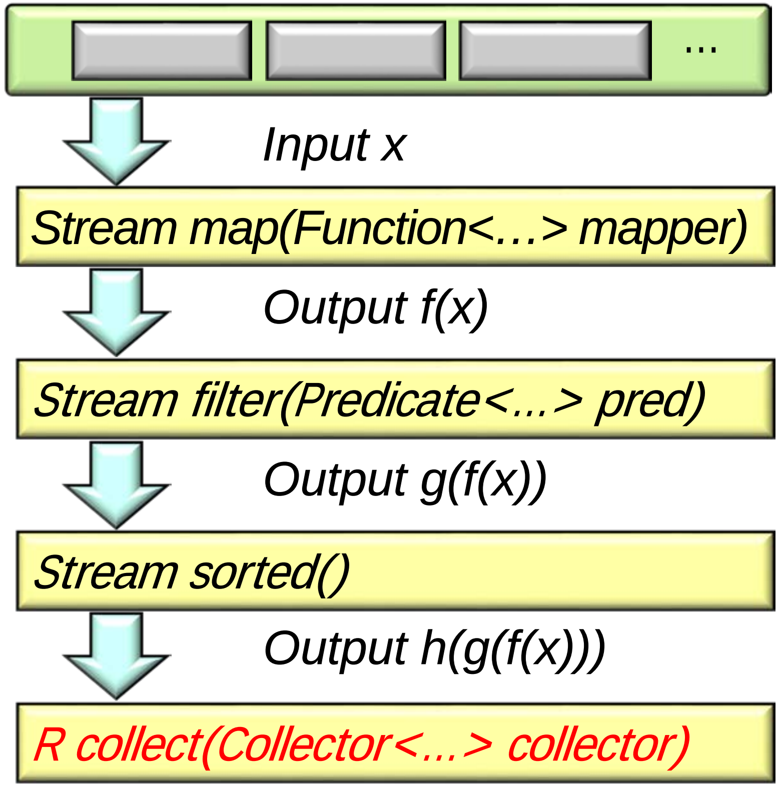

...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel Stream Construction & Execution

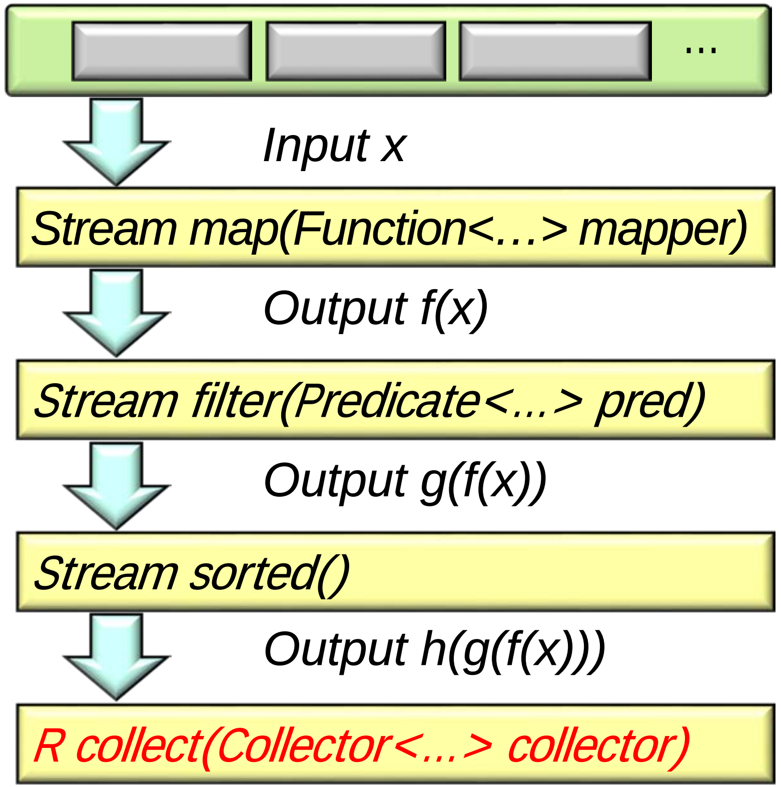
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories




...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

Parallel Stream Construction & Execution

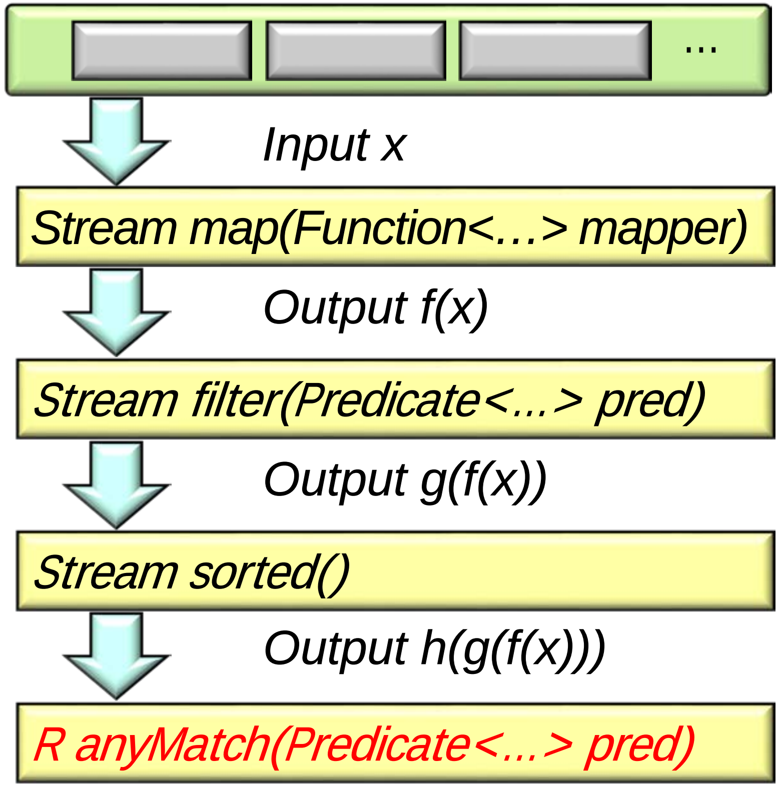
- When terminal operation runs the stream framework picks an execution plan
  - The plan is based on properties of the source & aggregate operations
  - Intermediate operations are divided into two categories
  - Terminal operations are also divided into two categories
    - Non-short-circuiting
   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


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<...> collector)

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

Parallel 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.


...
Input x
Stream map(Function<...> mapper)
Output f(x)
Stream filter(Predicate<...> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R anyMatch(Predicate<...> pred)

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

End of Java 8 Parallel Stream Internals (Part 6)