# Java 8 Parallel SearchStreamGang Example (Part 1)

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

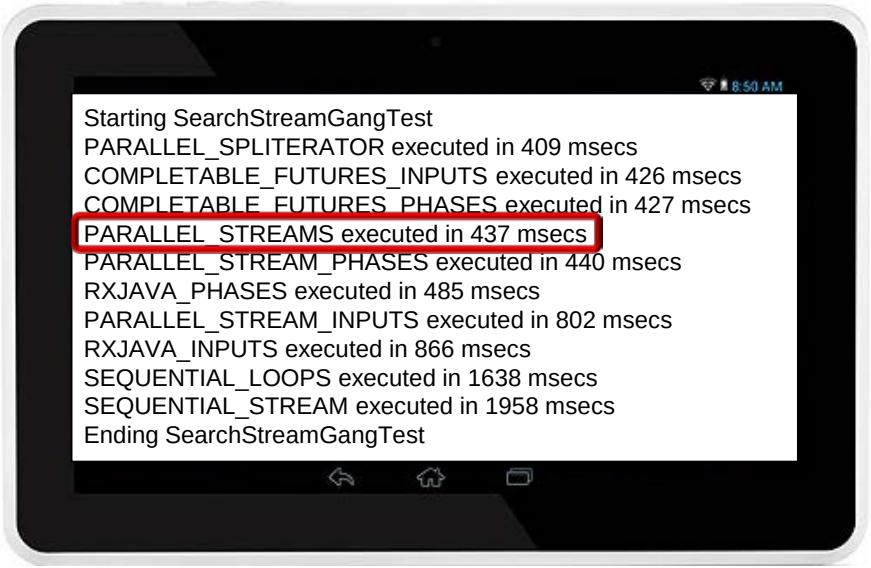
Professor of Computer Science

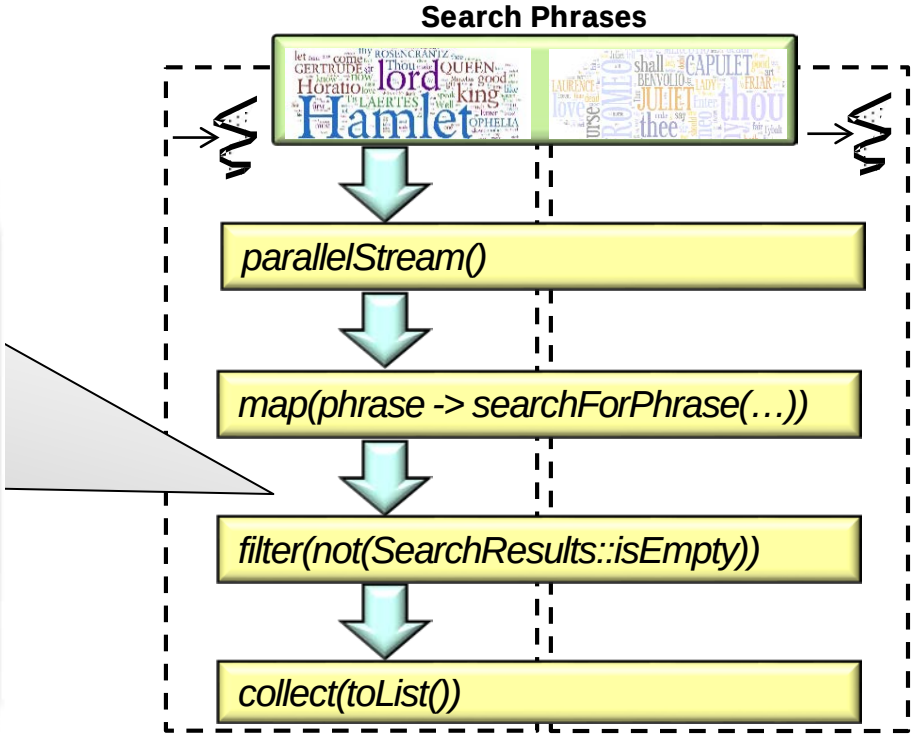
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang

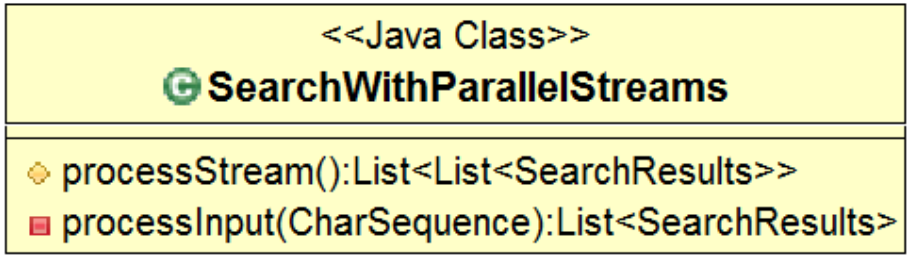



See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang

# Learning Objectives in this Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang
- Understand the pros & cons of the SearchWithParallelStreams class

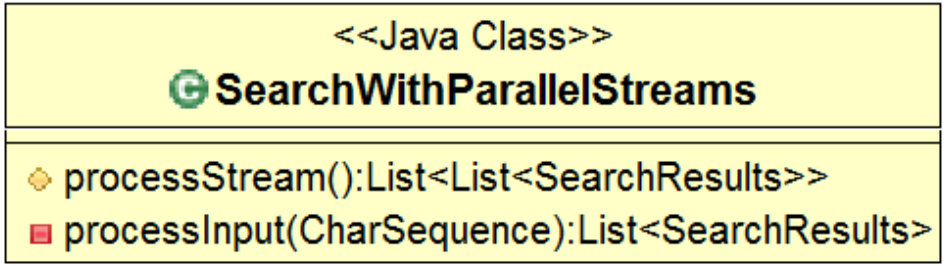



See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelStreams.java

# Applying Parallel Streams to SearchStreamGang

# Applying Parallel Streams to SearchStreamGang

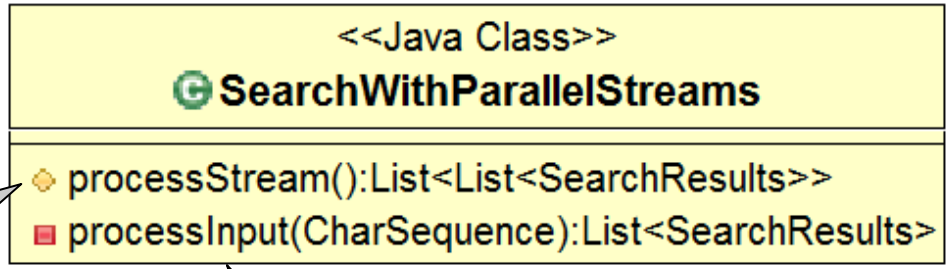
- We focus on parallel streams in processStream() & processInput() from SearchWithParallelStreams




See github.com/douglascraigschmidt/LiveLessons/tree/master/SearchStreamGang

# Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in processStream() & processInput() from SearchWithParallelStreams



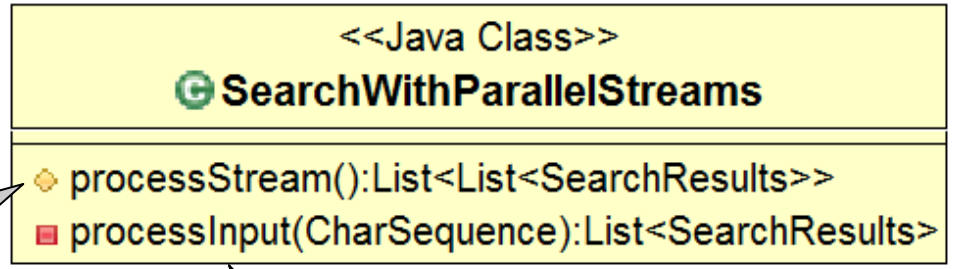

```
getInput()
   .parallelStream()
   .map(this::processInput)
   .collect(toList());
```

```
return mPhrasesToFind
   .parallelStream()
   .map(phrase -> searchForPhrase(phrase, input, title, false))
   .filter(not(SearchResults::isEmpty)
   .collect(toList());
```

See SearchStreamGang/src/main/java/livelessons/streamgangs/SearchWithParallelStreams.java

# Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in processStream() & processInput() from SearchWithParallelStreams



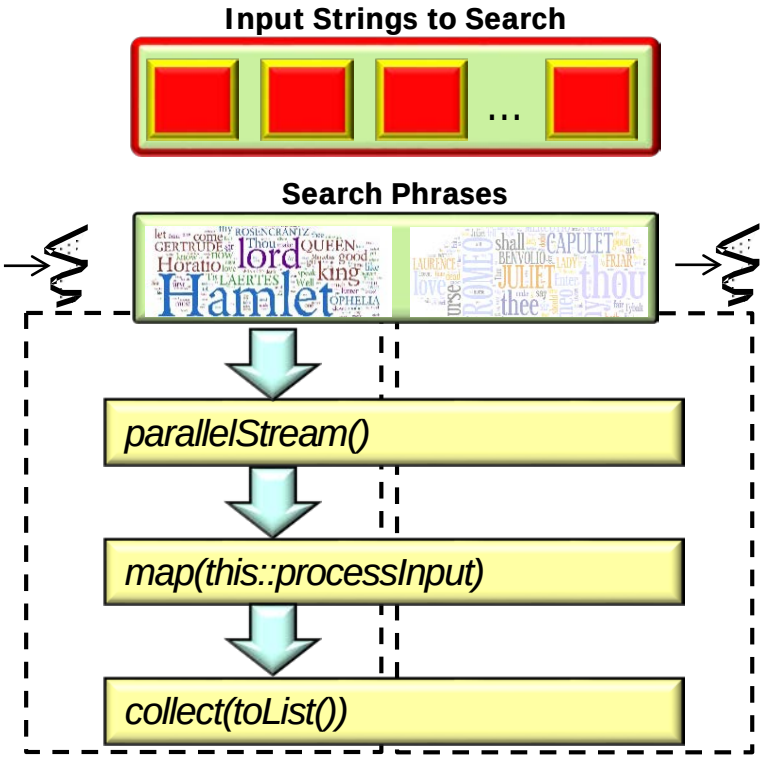

```
getInput()
   .parallelStream()
   .map(this::processInput)
   .collect(toList());
```

```
return mPhrasesToFind
   .parallelStream()
   .map(phrase -> searchForPhrase(phrase, input, title, false))
   .filter(not(SearchResults::isEmpty)
   .collect(toList());
```

i.e., the map(), filter(), & collect() aggregate operations

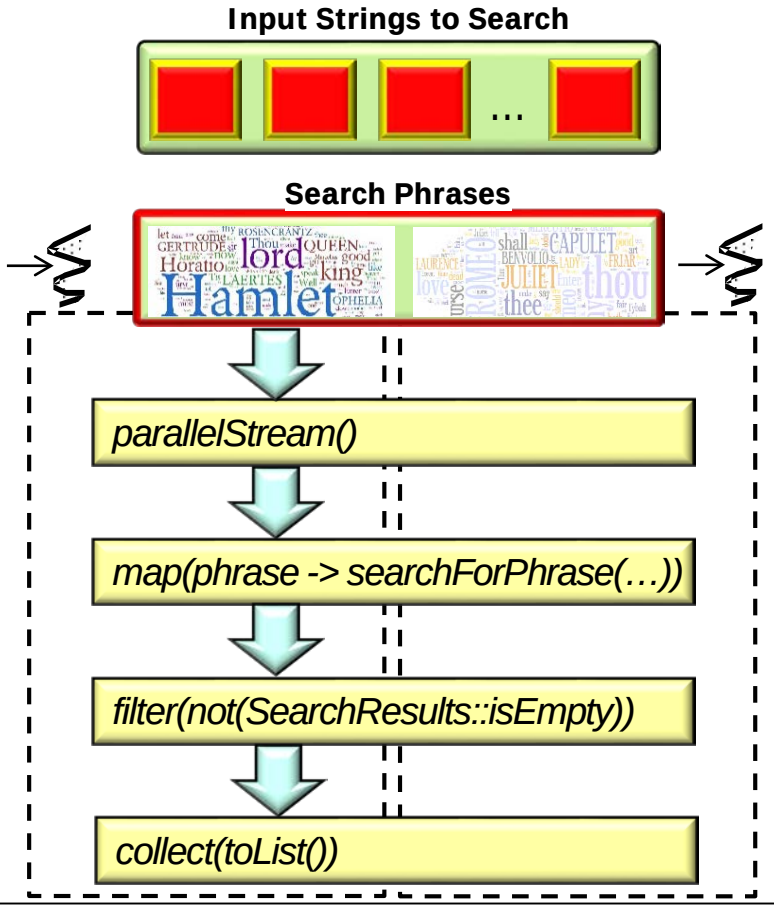
# Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in processStream() & processInput() from SearchWithParallelStreams
  - **processStream()**
    - Uses a parallel stream to search a list of input strings in parallel



# Applying Parallel Streams to SearchStreamGang

- We focus on parallel streams in processStream() & processInput() from SearchWithParallelStreams
  - processStream()
  - **processInput()**
    - Uses a parallel stream to search each input string to locate all occurrences of phases

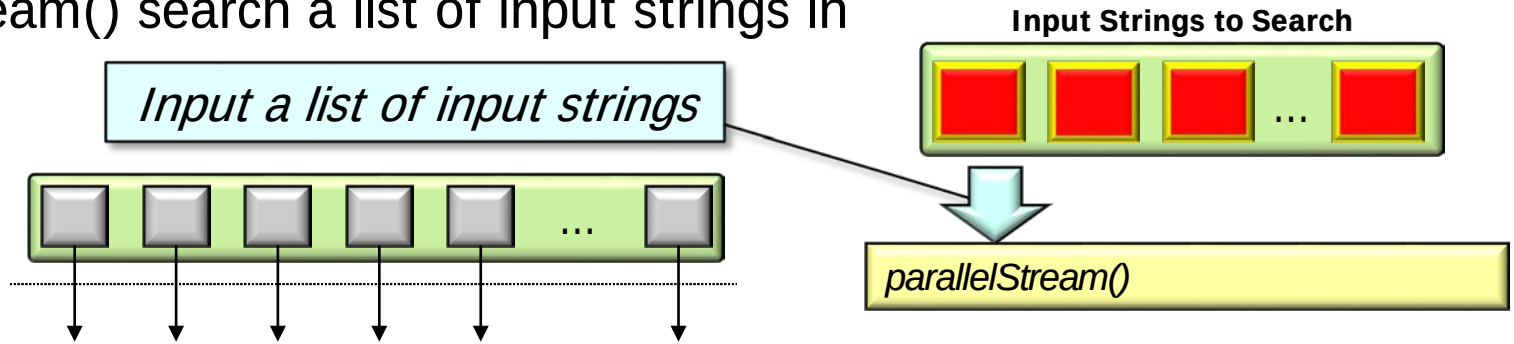


# Visualizing processStream() & processInput()

# Visualizing processStream() & processInput()

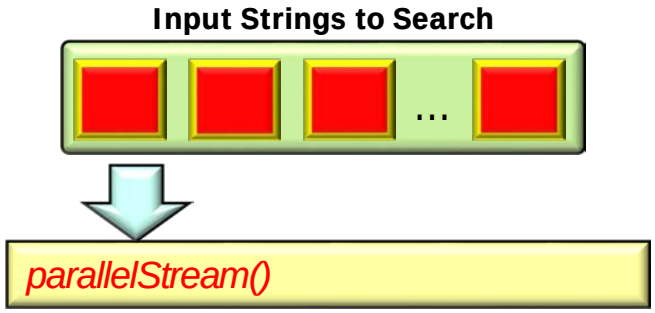
- processStream() search a list of input strings in parallel

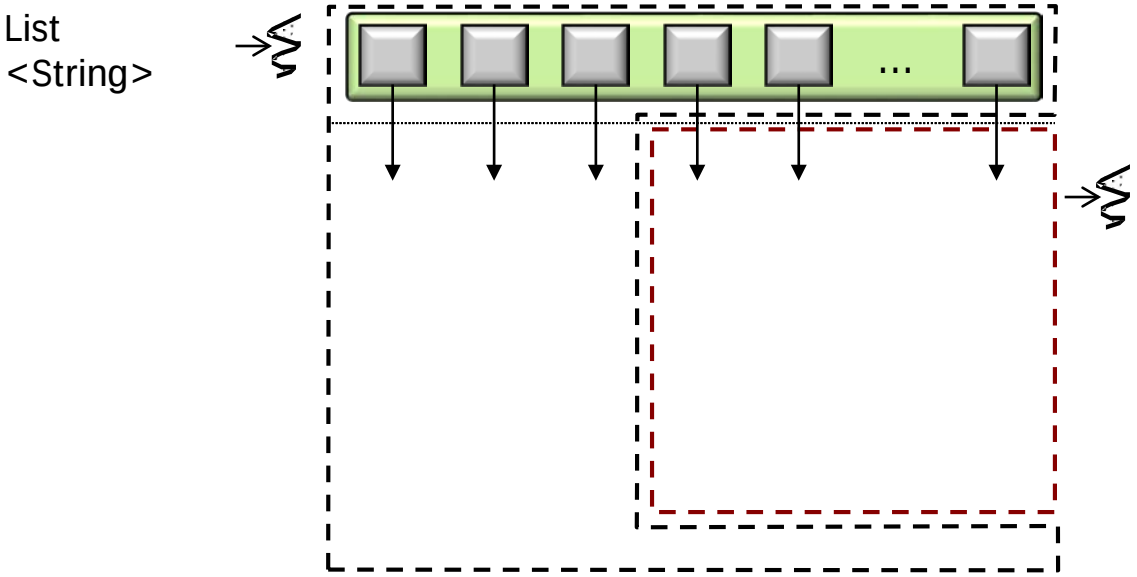
List
<String>



# Visualizing processStream() & processInput()

- processStream() search a list of input strings in parallel

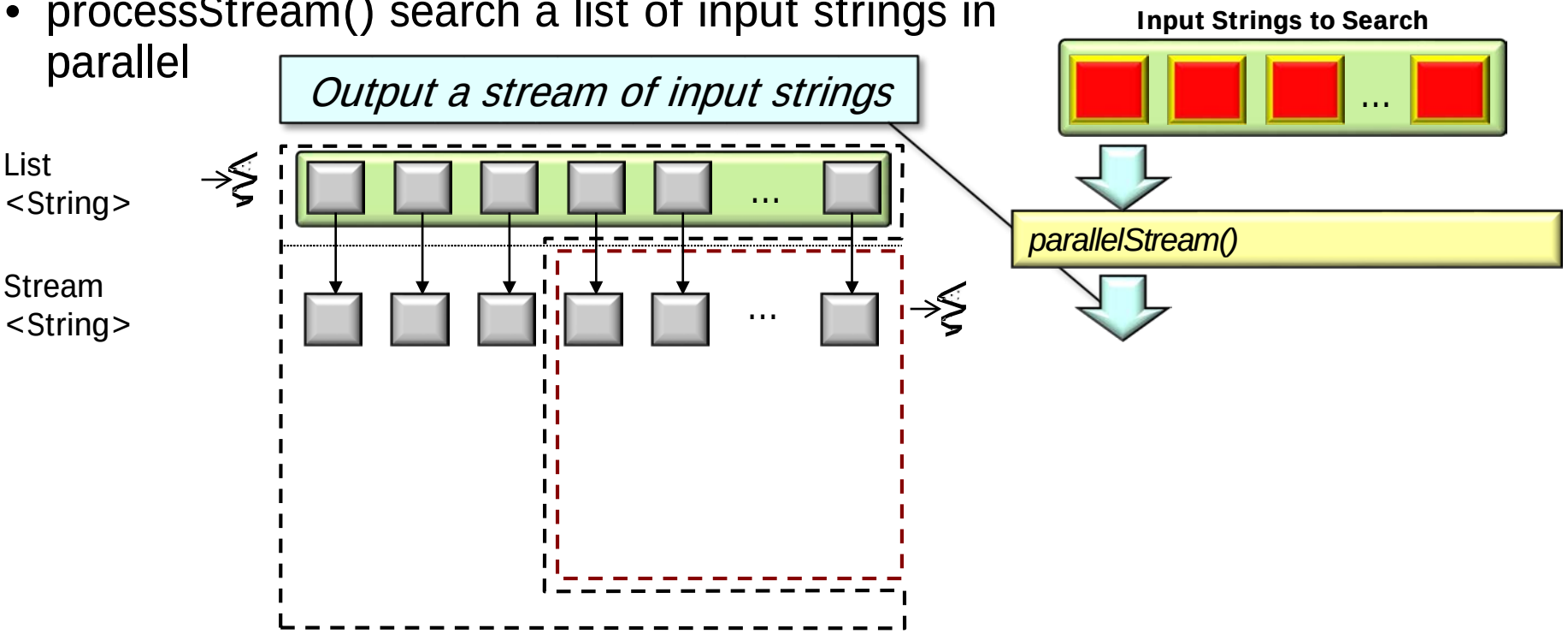



Convert collection to a parallel stream, i.e., substreams with chunks of input strings

# Visualizing processStream() & processInput()

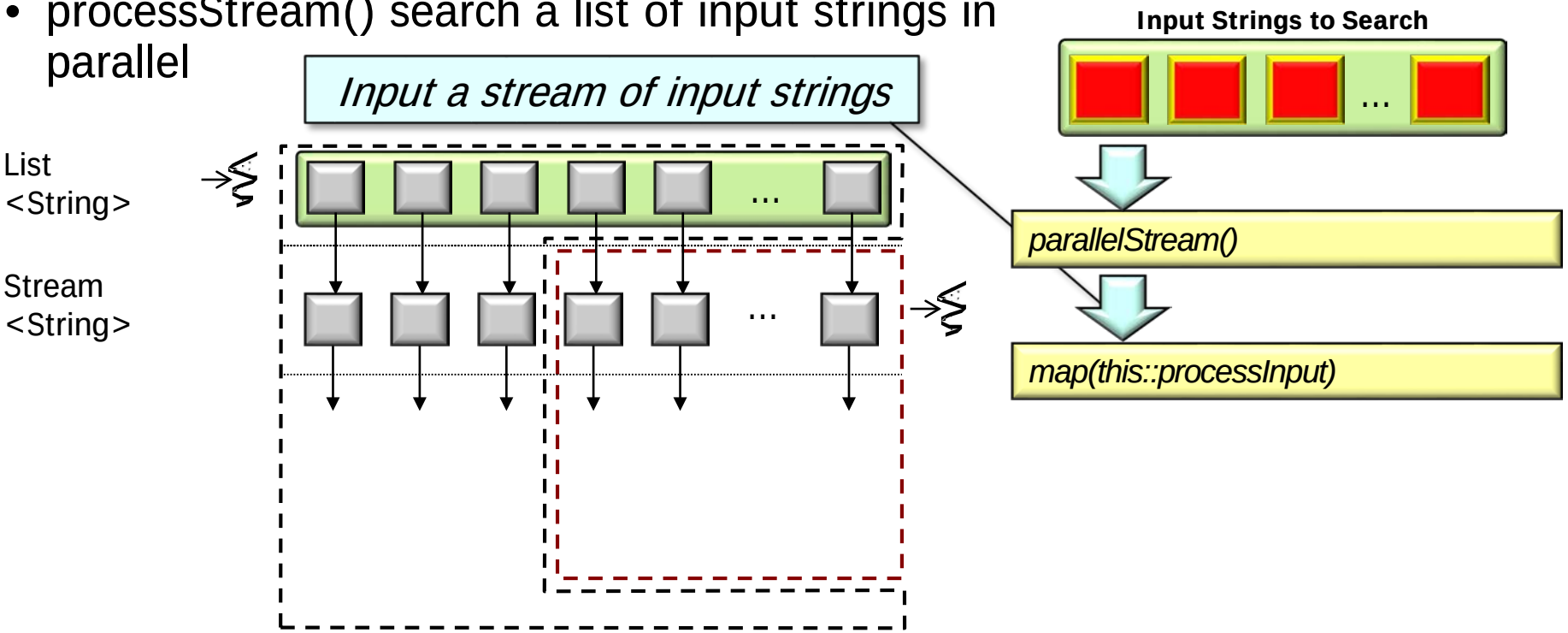
- processStream() search a list of input strings in parallel




Chunks of input strings are processed in parallel on separate threads/cores

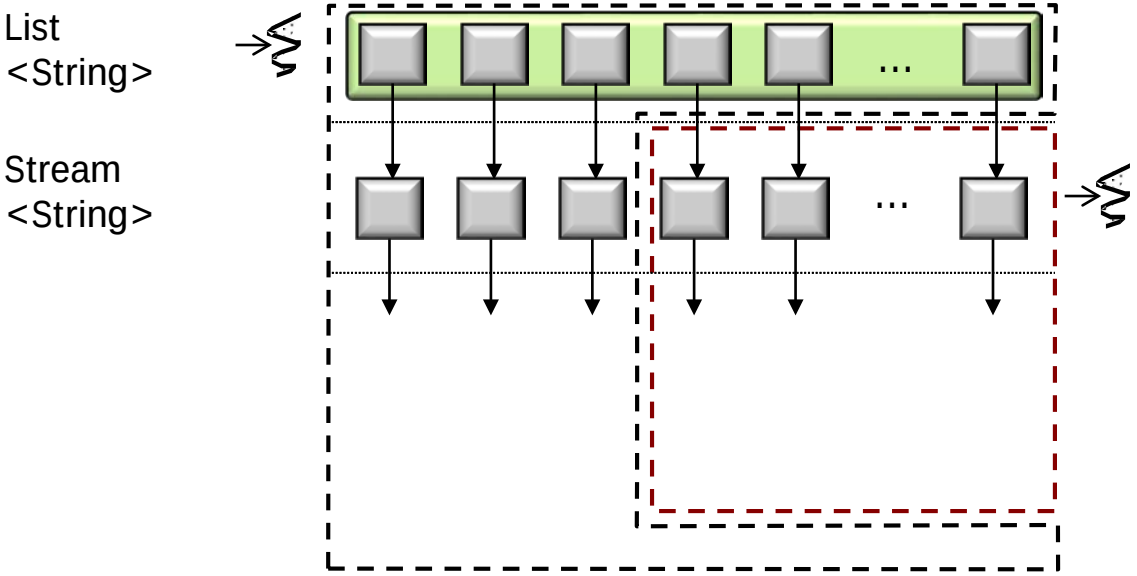
# Visualizing processStream() & processInput()

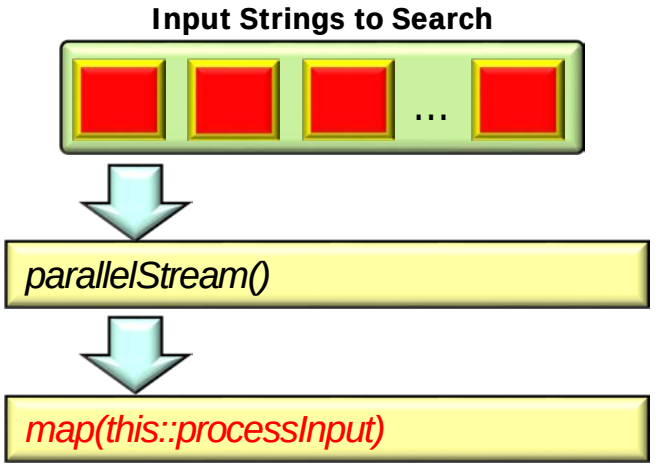
- processStream() search a list of input strings in parallel



# Visualizing processStream() & processInput()

- processStream() search a list of input strings in parallel

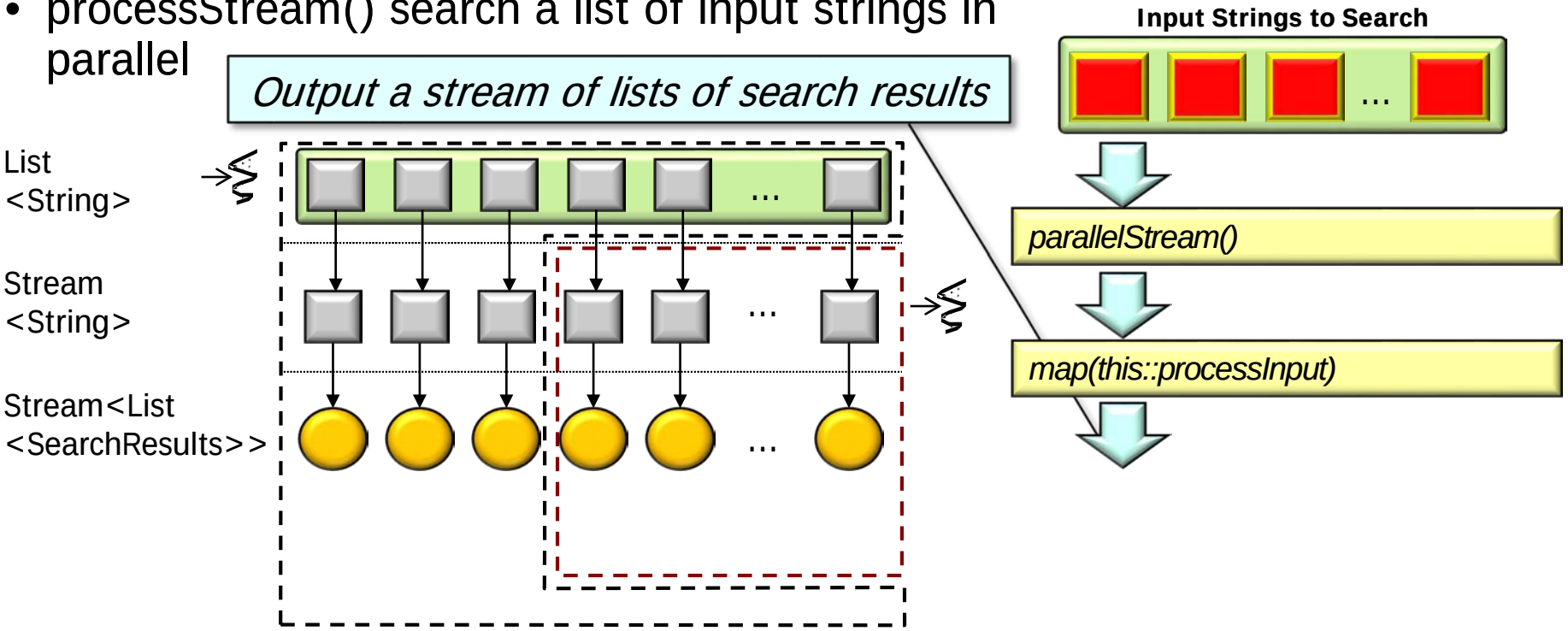



Call processInput() to search for phrases in each input string in parallel

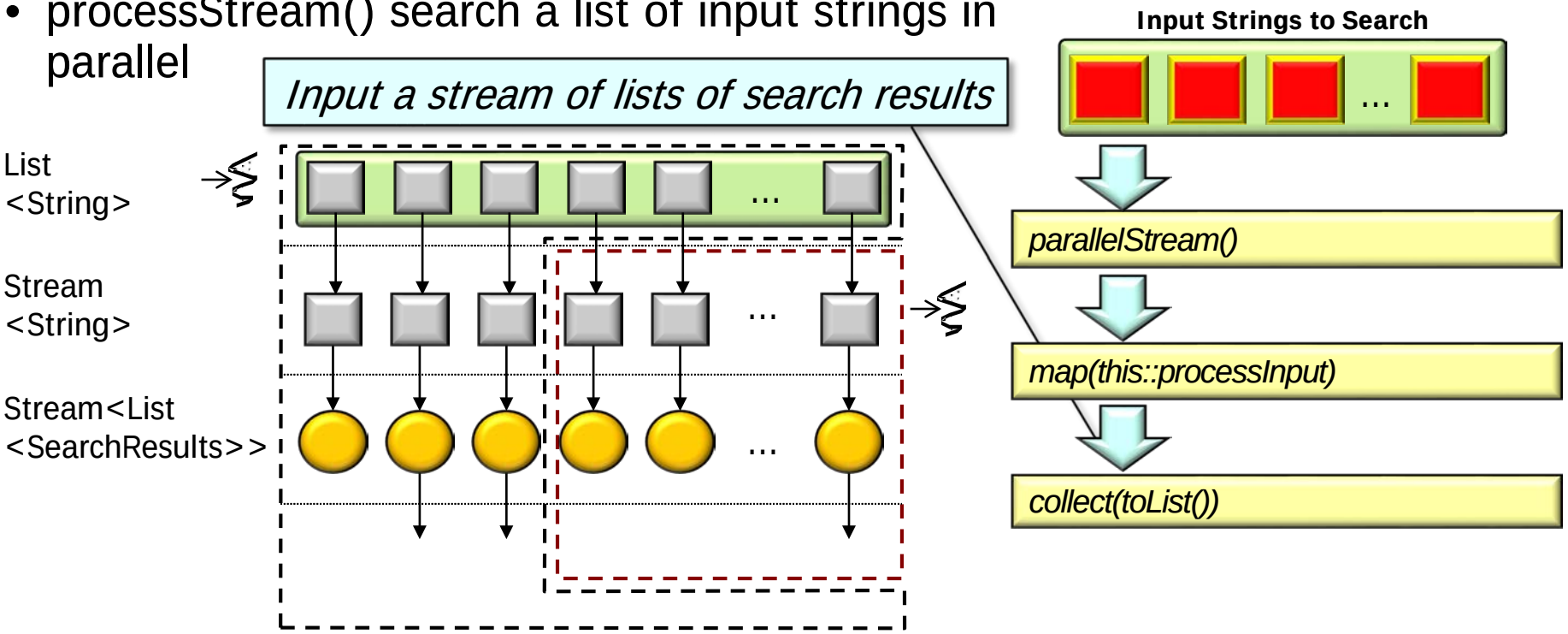
# Visualizing processStream() & processInput()

- processStream() search a list of input strings in parallel



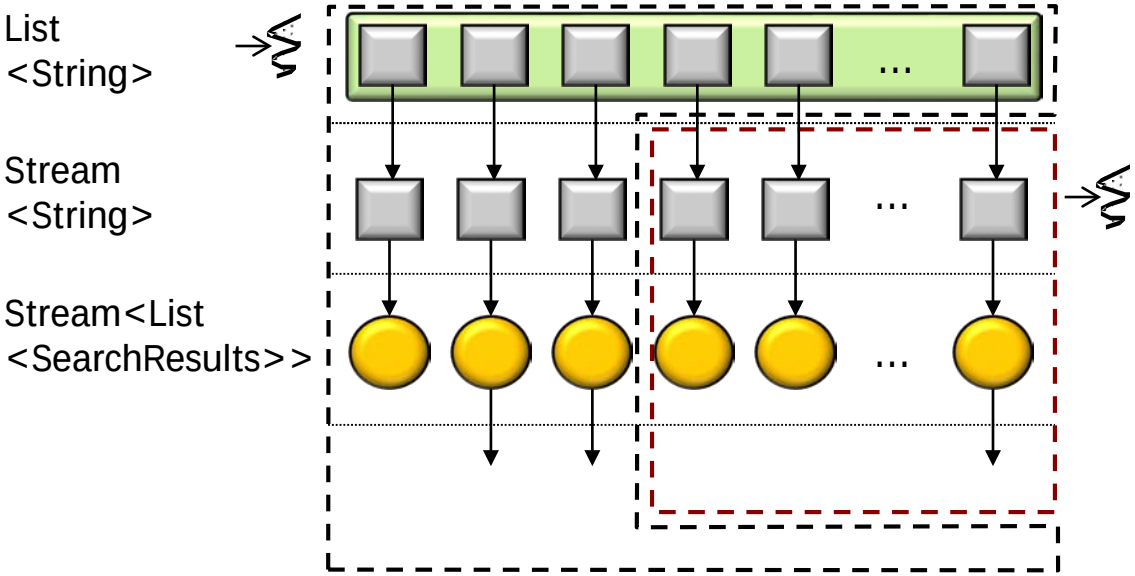
# Visualizing processStream() & processInput()

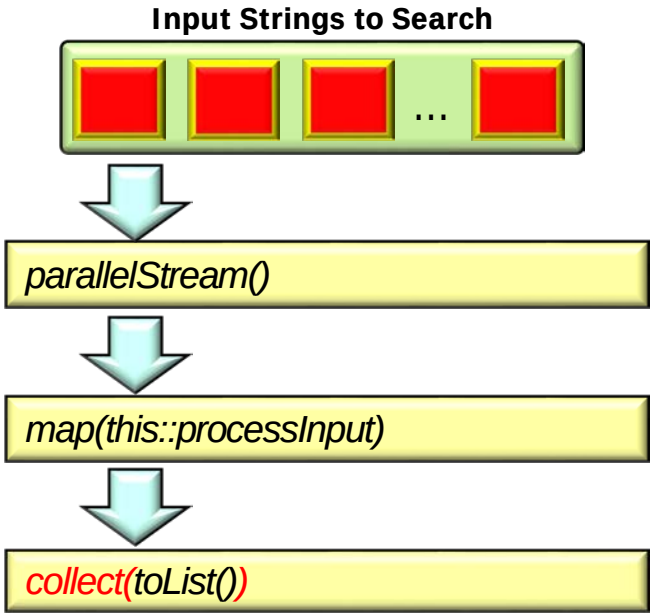
- processStream() search a list of input strings in parallel



# Visualizing processStream() & processInput()

- processStream() search a list of input strings in parallel

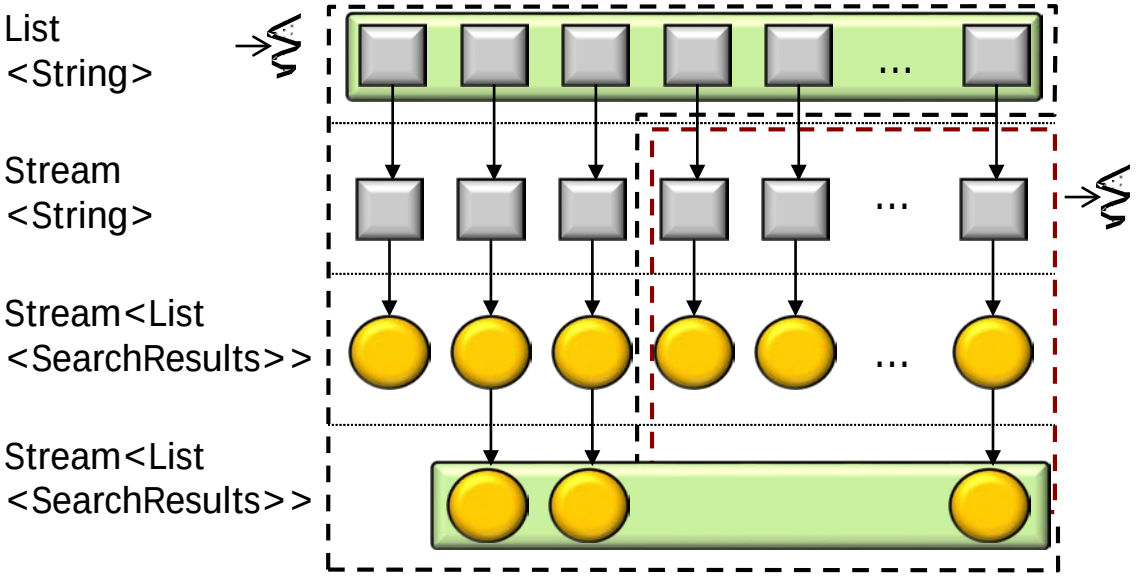


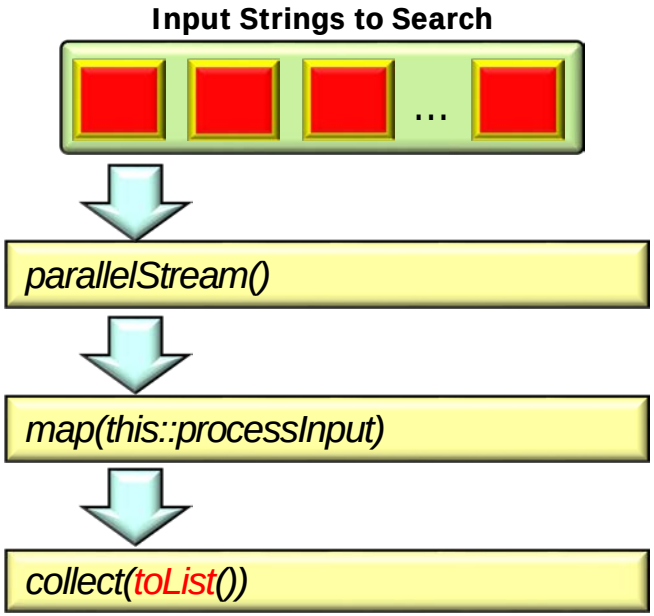




Trigger intermediate operation processing to run on multiple threads/cores

# Visualizing processStream() & processInput()

- processStream() search a list of input strings in parallel



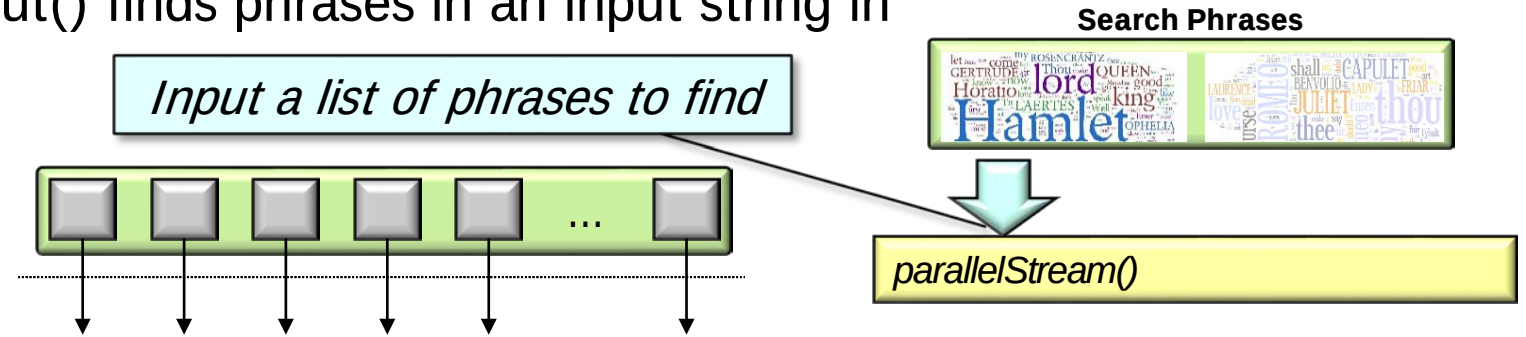




Return a list of lists of search results based on “encounter order”

# Visualizing processStream() & processInput()

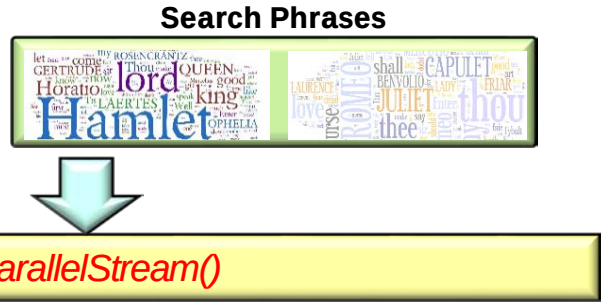
- processInput() finds phrases in an input string in parallel

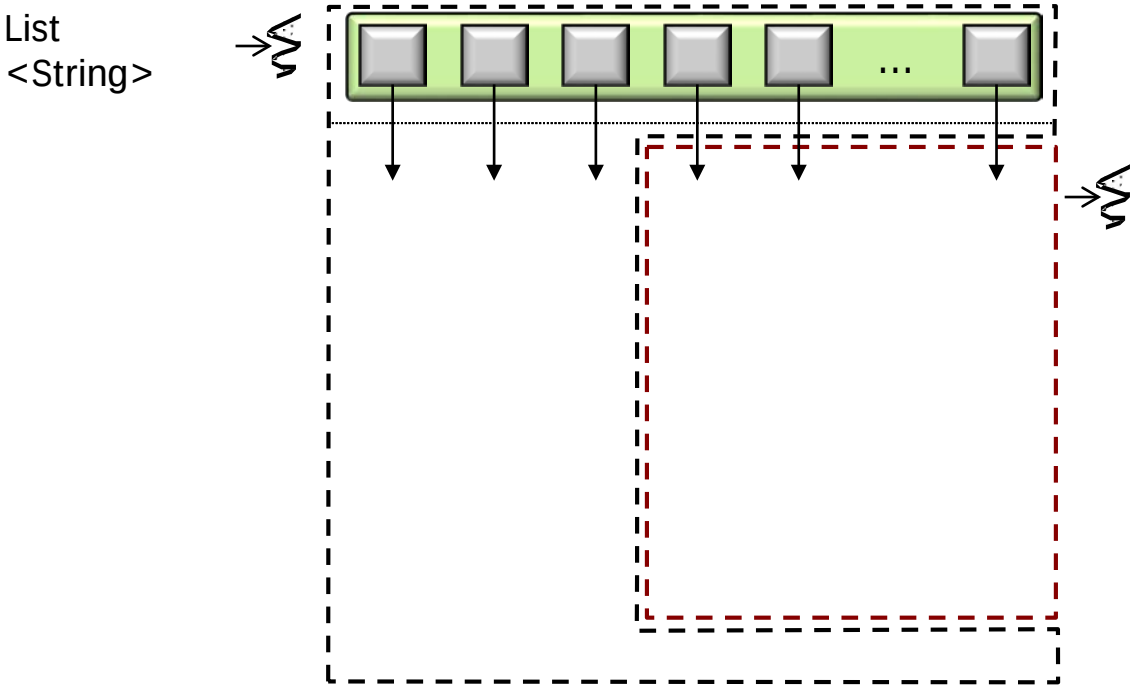
List
<String>



# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel

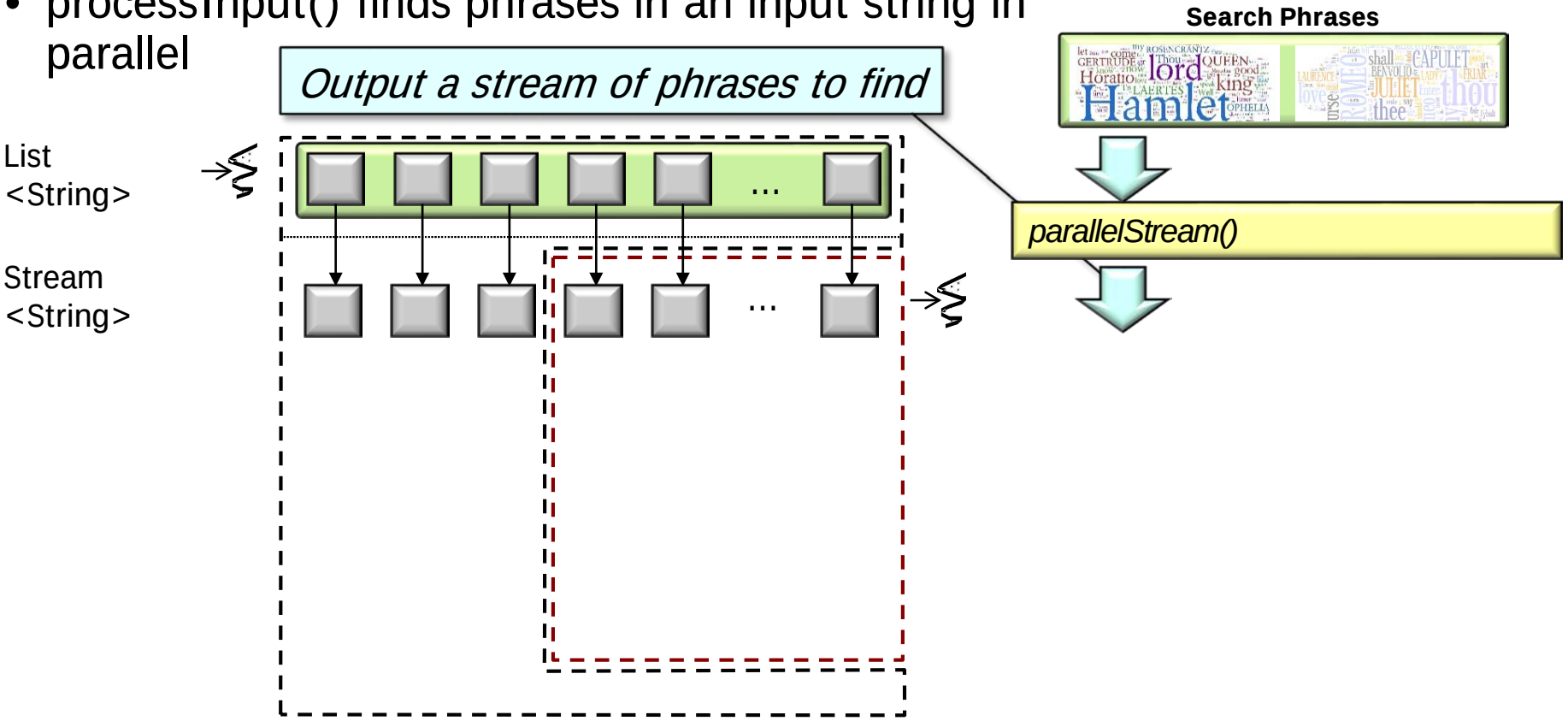



Convert collection to a parallel stream, i.e., substreams with chunks of phrases

# Visualizing processStream() & processInput()

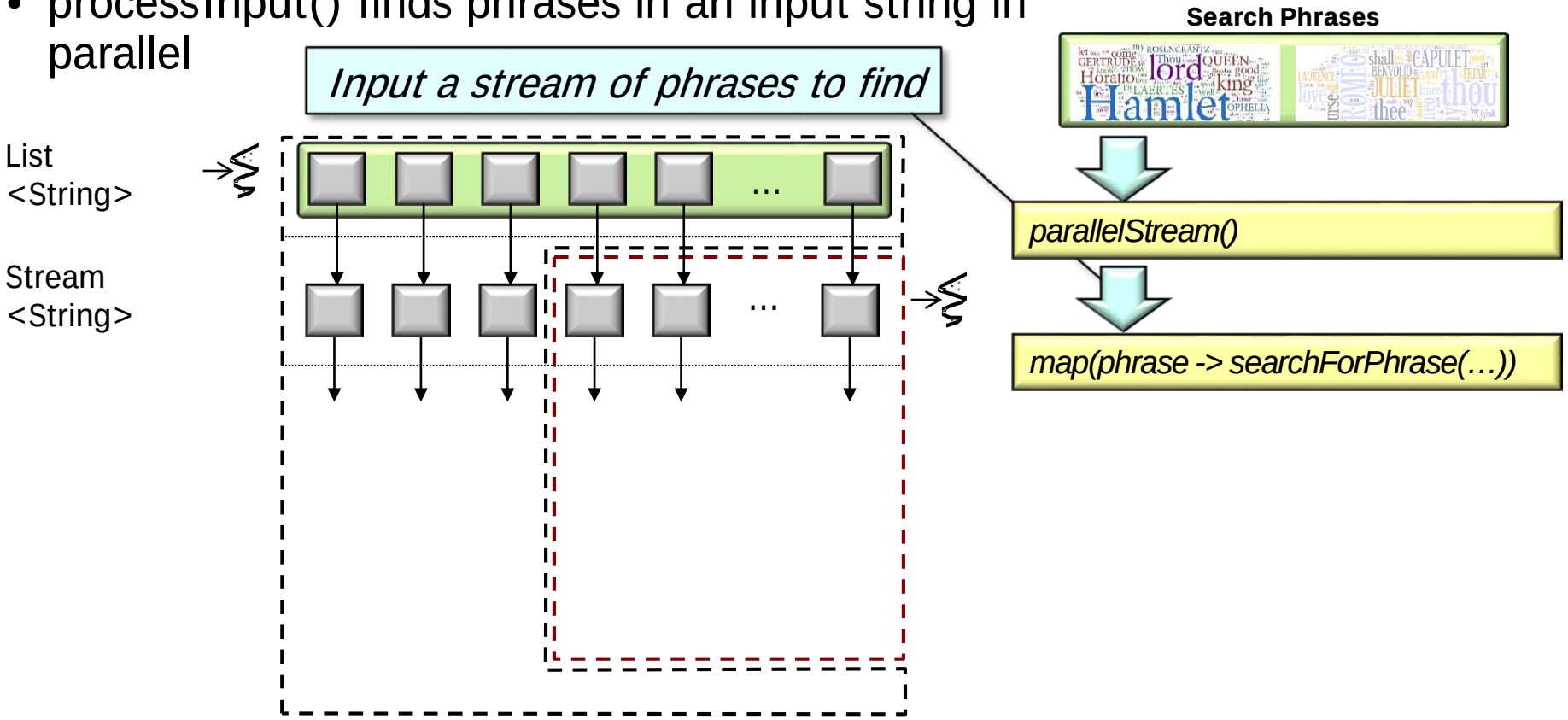
- processInput() finds phrases in an input string in parallel




Different chunks of phrases are processed in parallel on separate threads/cores

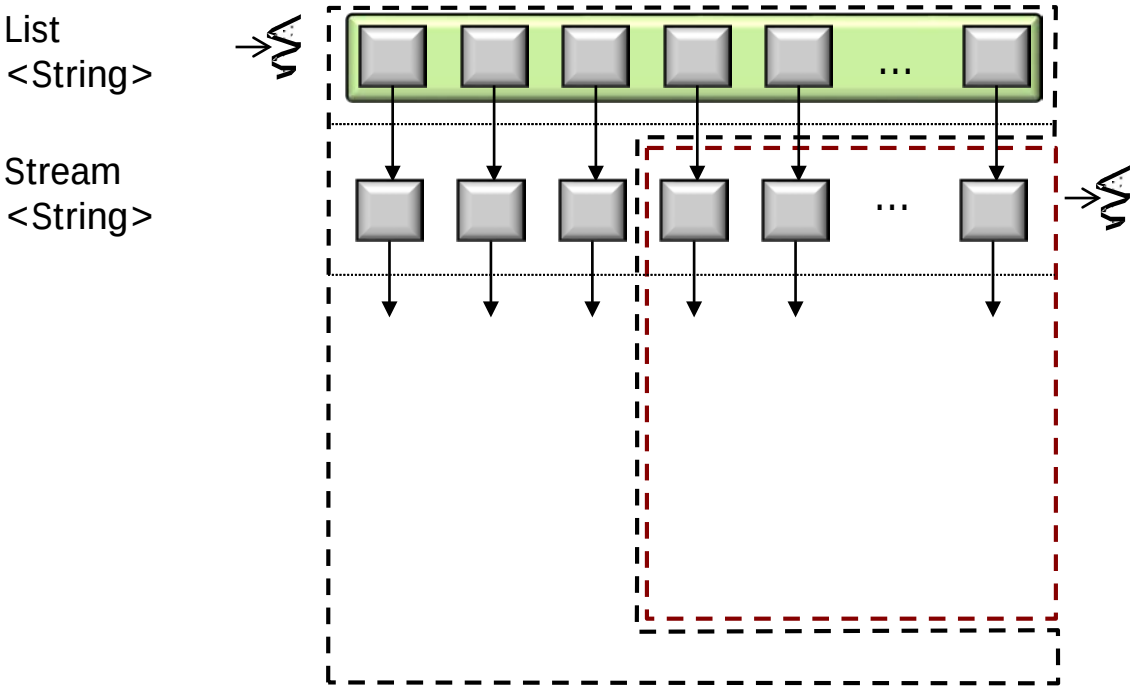
# Visualizing processStream() & processInput()

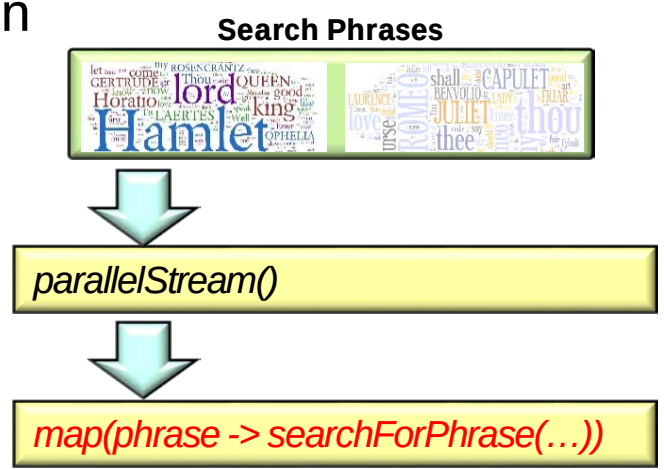
- processInput() finds phrases in an input string in parallel



# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel

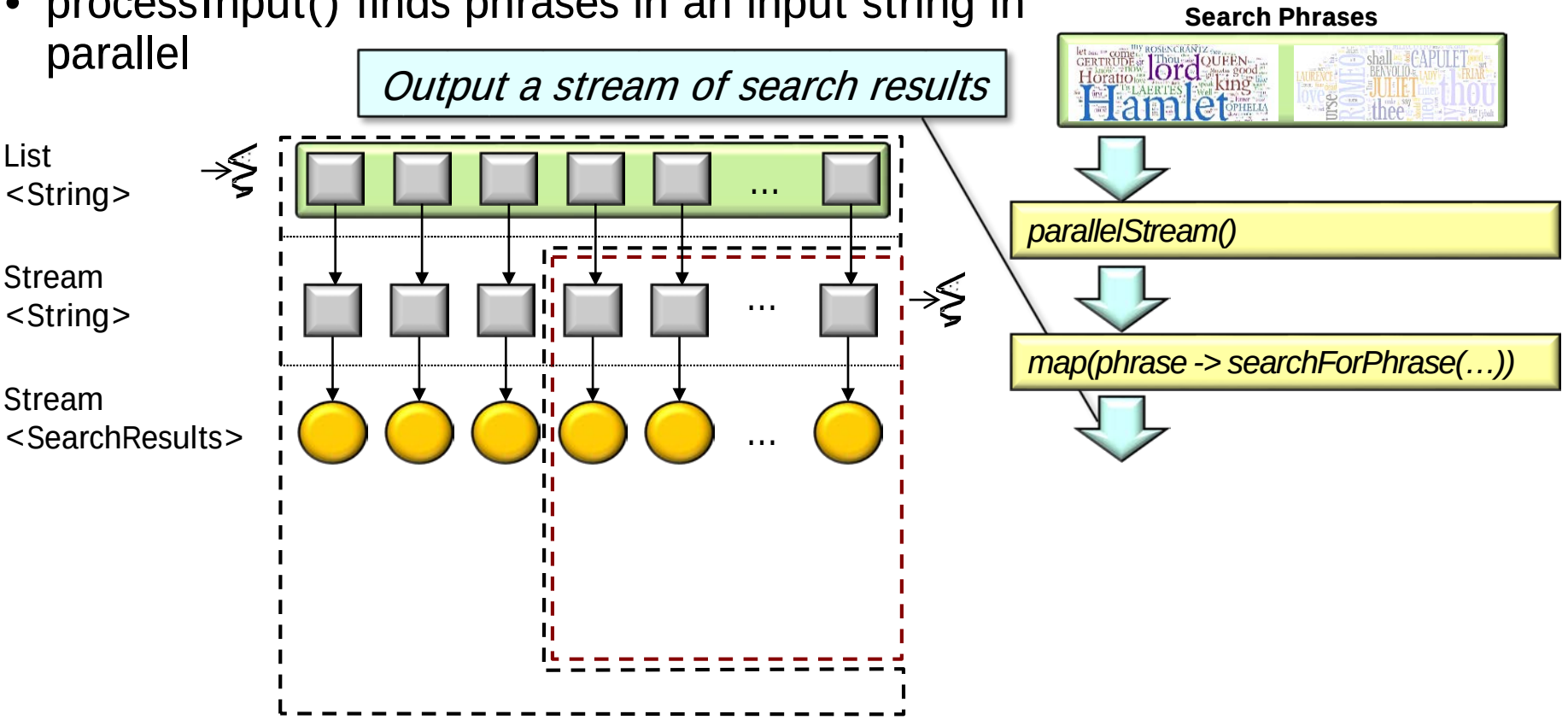



Search for phrases in each input string in parallel

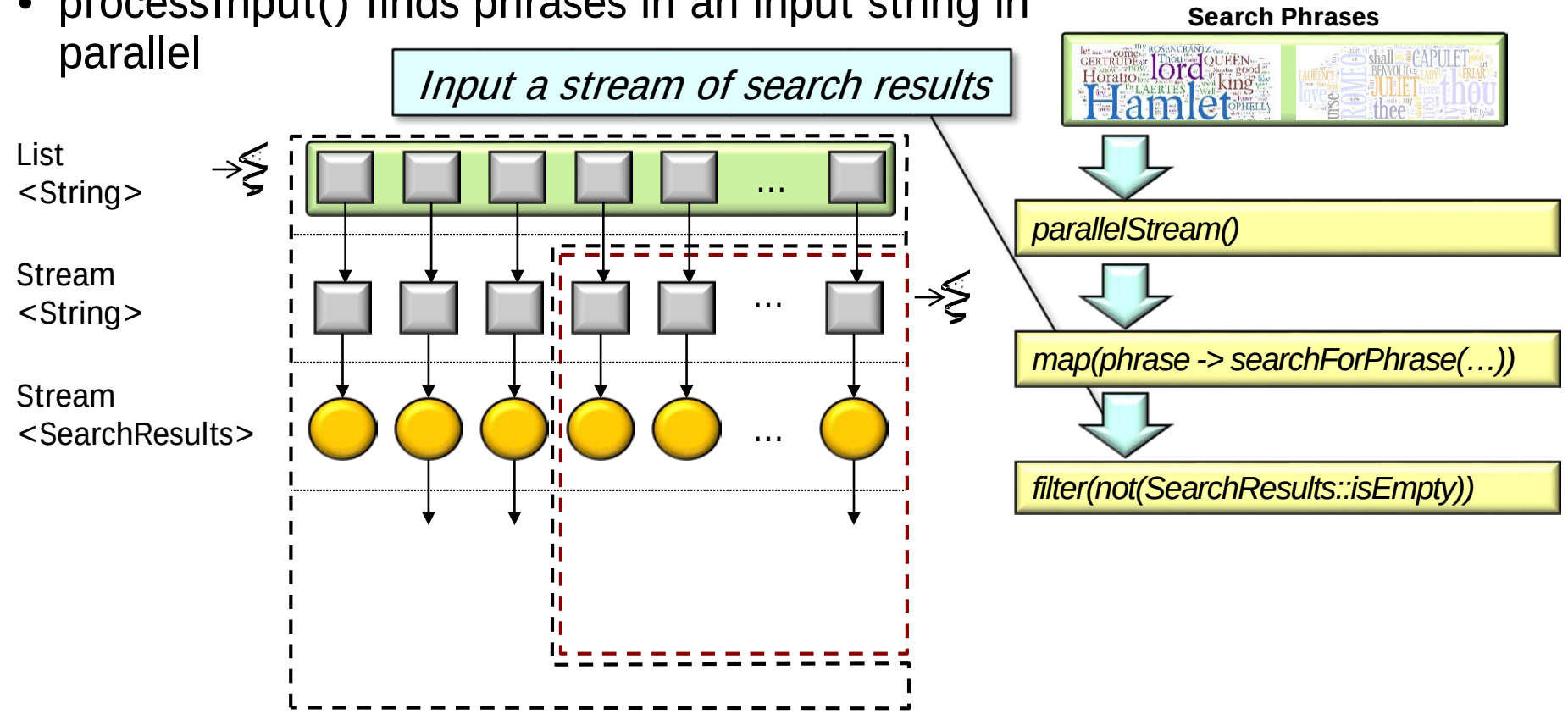
# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel



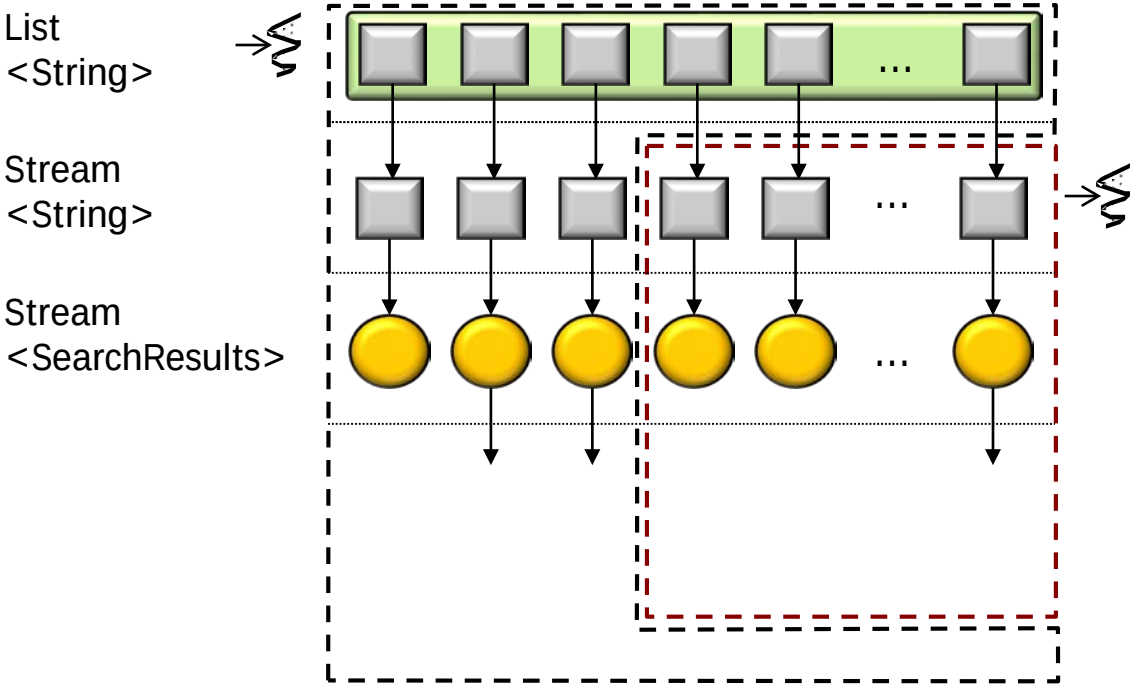
# Visualizing processStream() & processInput()

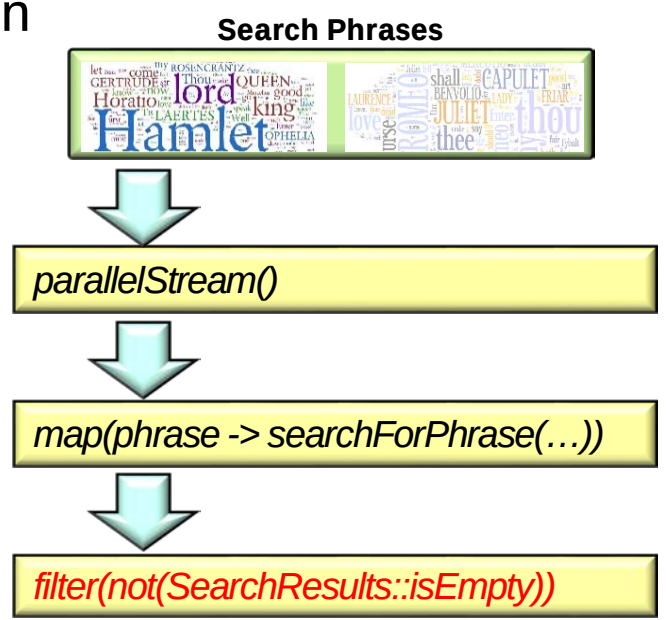
- processInput() finds phrases in an input string in parallel



# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel

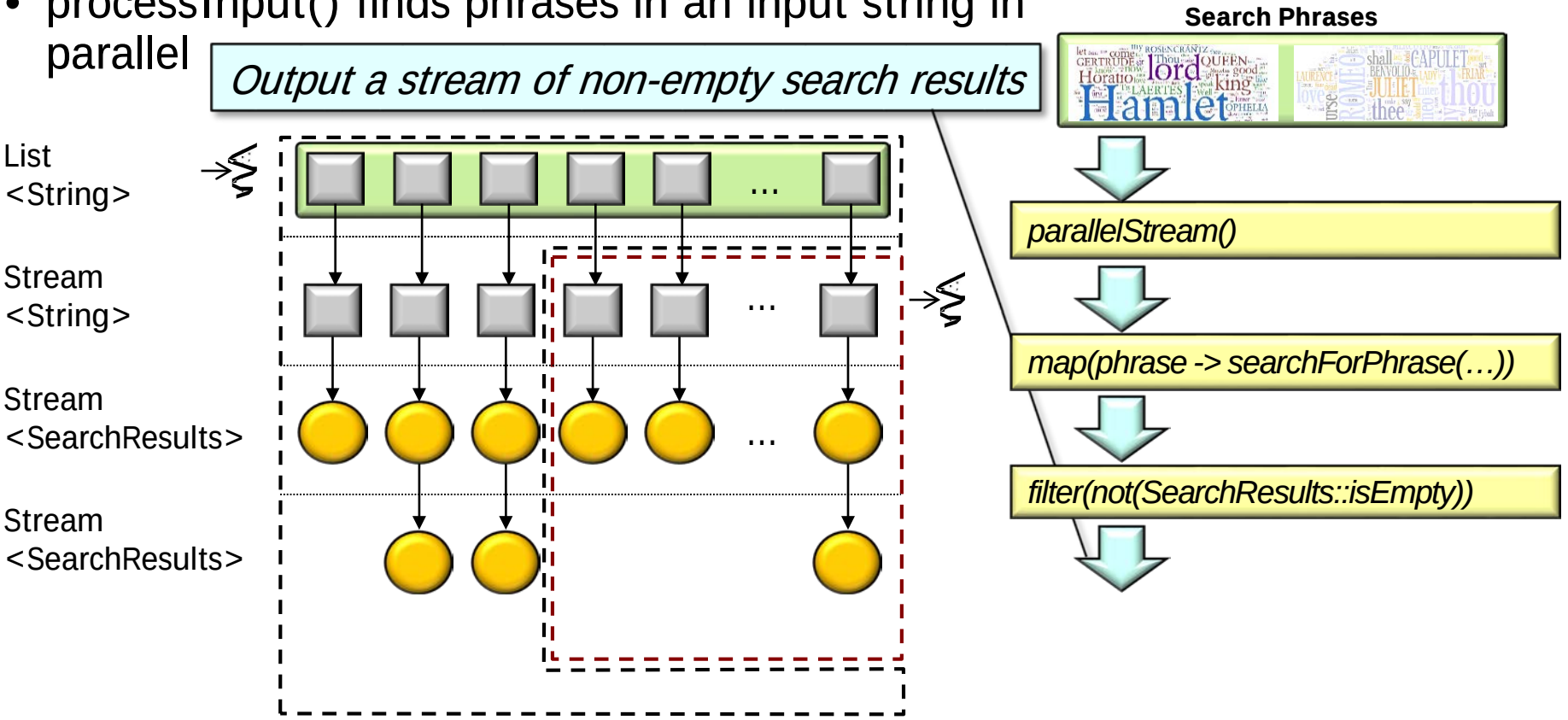



Remove empty search results from substreams in parallel

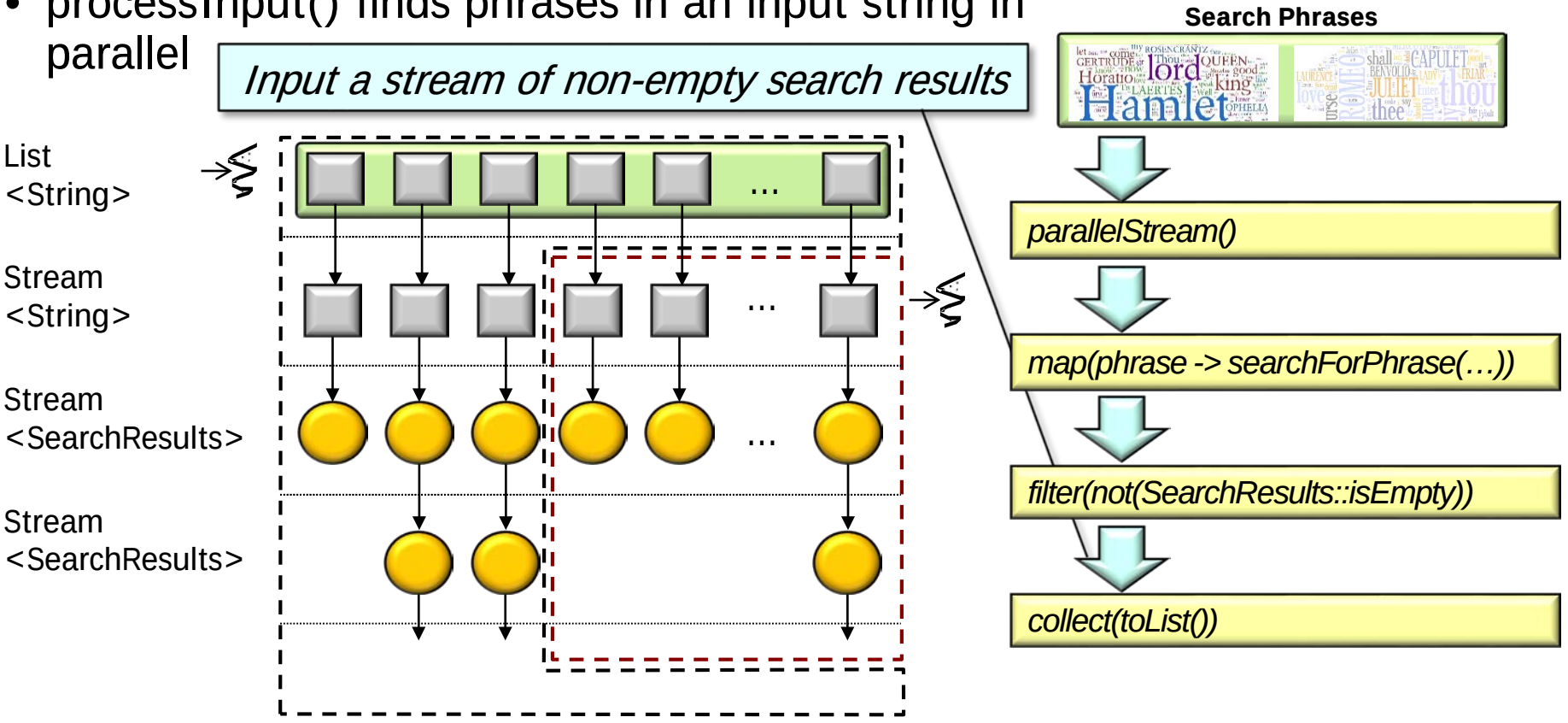
# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel



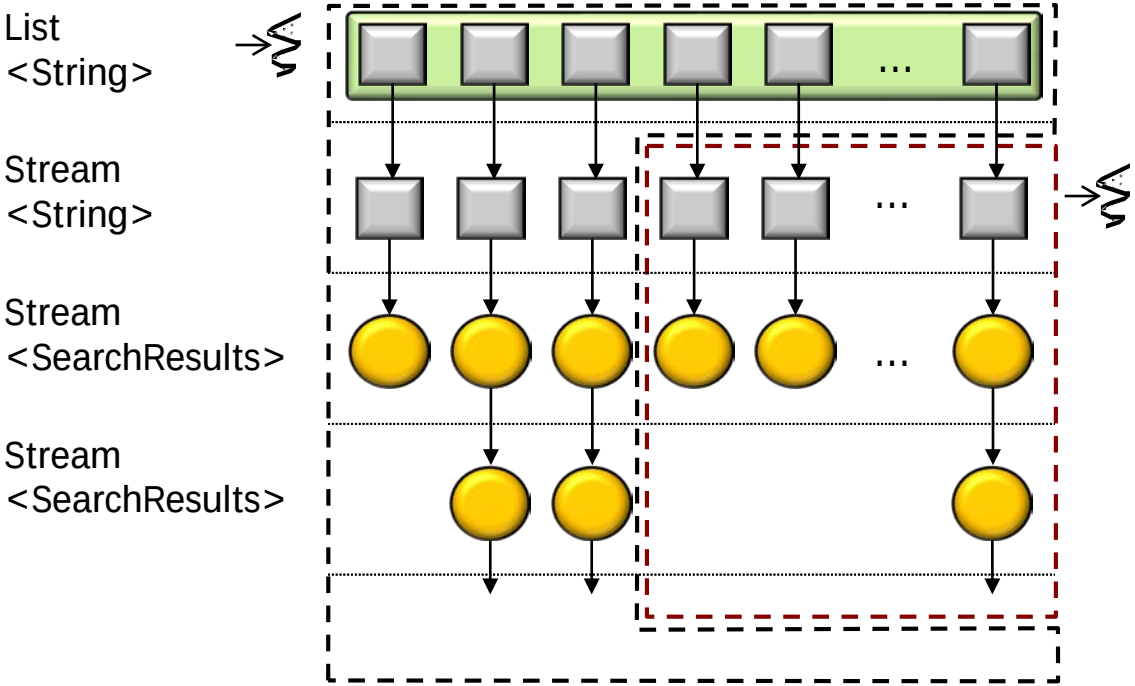
# Visualizing processStream() & processInput()

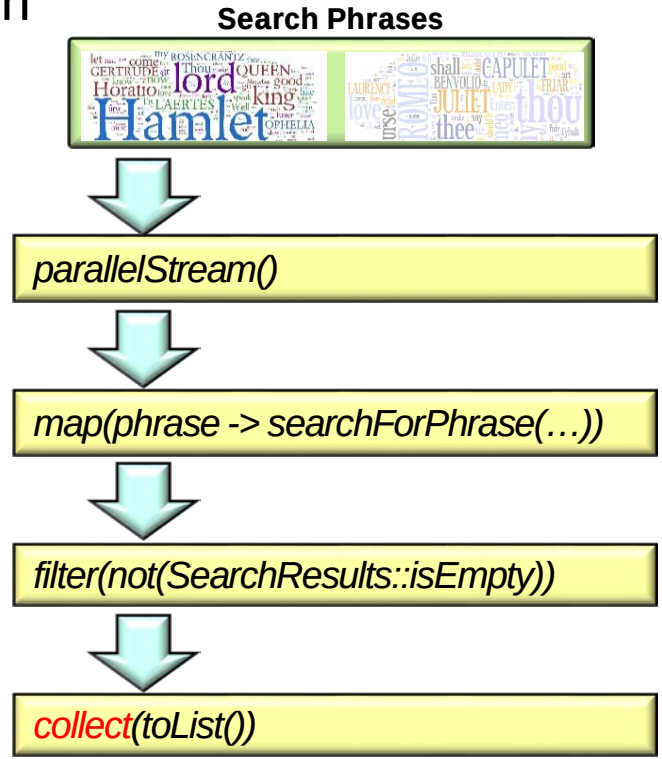
- processInput() finds phrases in an input string in parallel



# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel

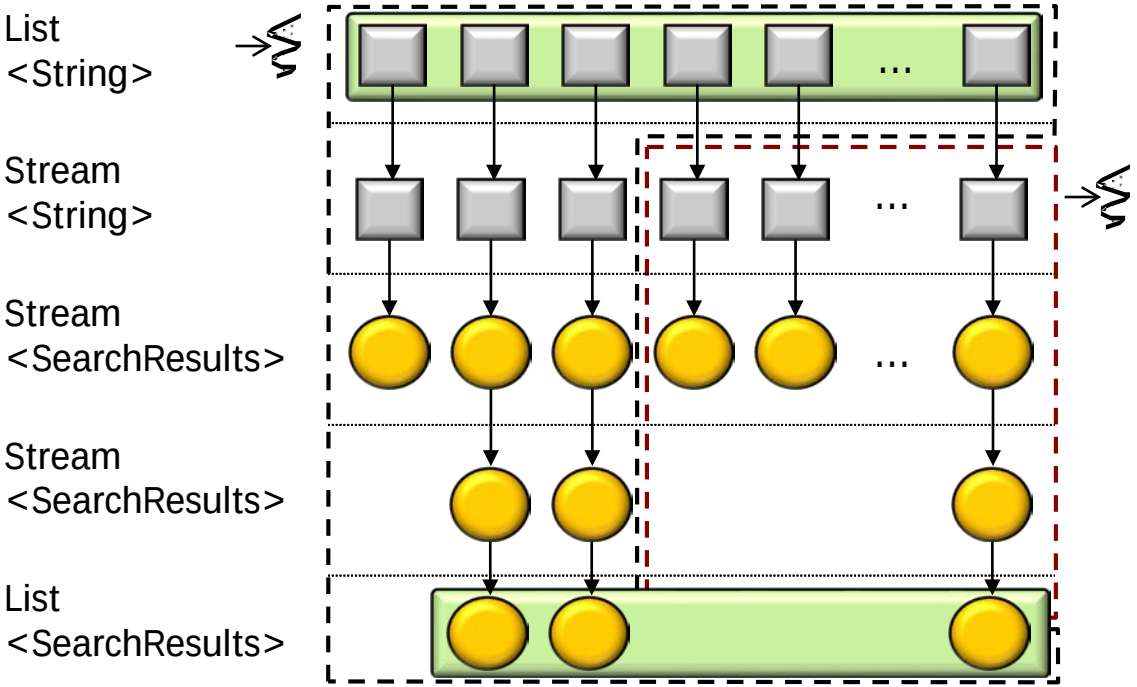


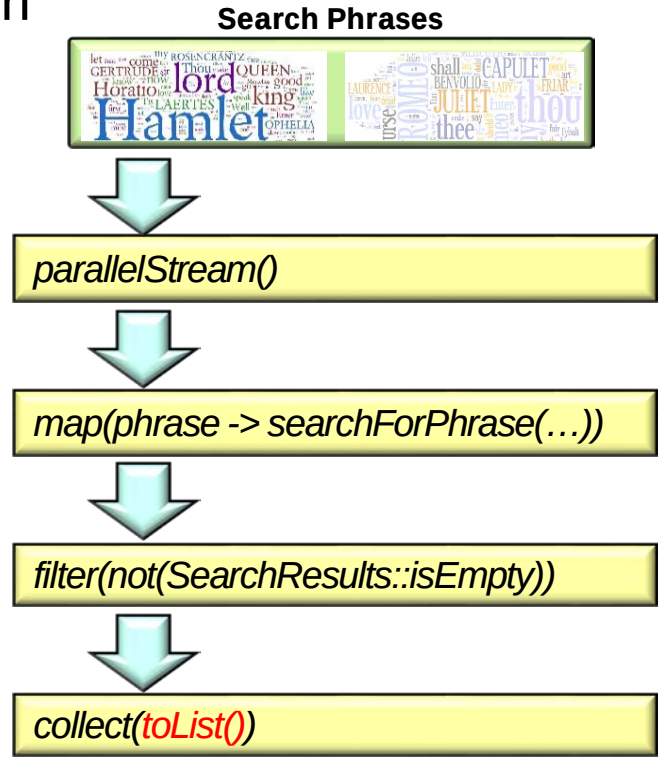




Trigger intermediate operation processing to run on multiple threads/cores

# Visualizing processStream() & processInput()

- processInput() finds phrases in an input string in parallel

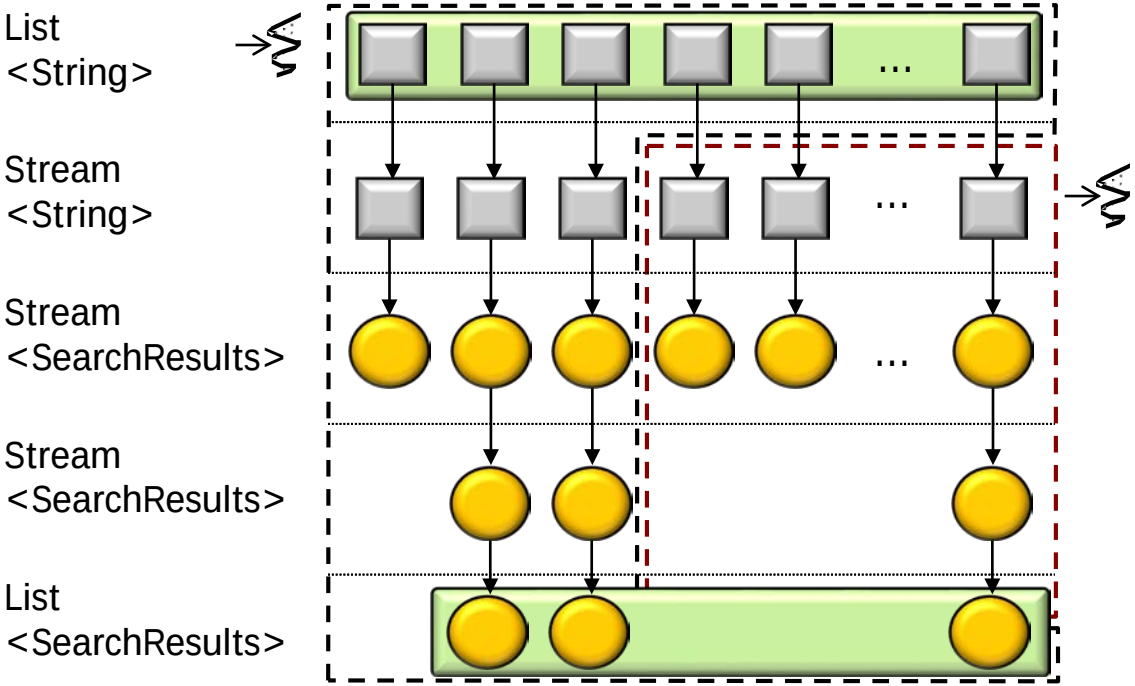


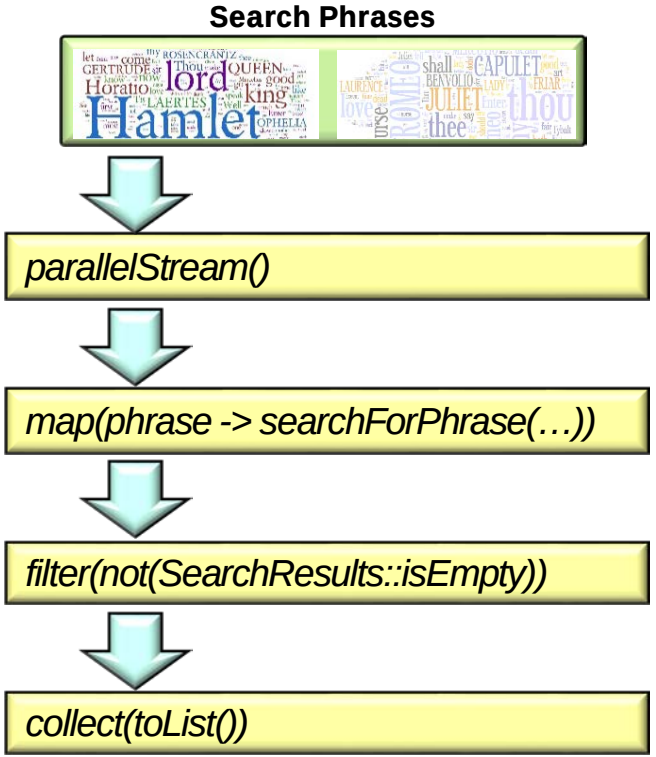




Return a list of search results in the originating thread based on “encounter order”

# Visualizing processStream() & processInput()

- Note that the actual processing of (parallel) streams differs from this visualization..




See www.ibm.com/developerworks/library/j-java-streams-3-brian-goetz

# Implementing processStream() as a Parallel Stream

# Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

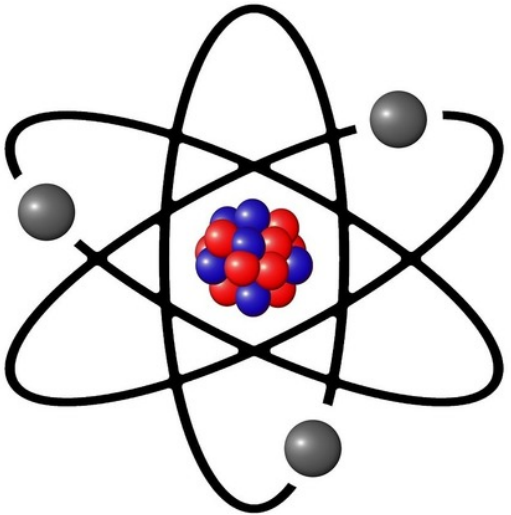
```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return getInput()

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```



# Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return getInput()

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```

*Creates a parallel stream that searches a list of input strings in parallel*

# Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return getInput()

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```

*Searches each input string to locate all occurrences of phases*

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

# Implementing processStream() as a Parallel Stream

- Parallel processStream() has one minuscule change wrt the sequential version

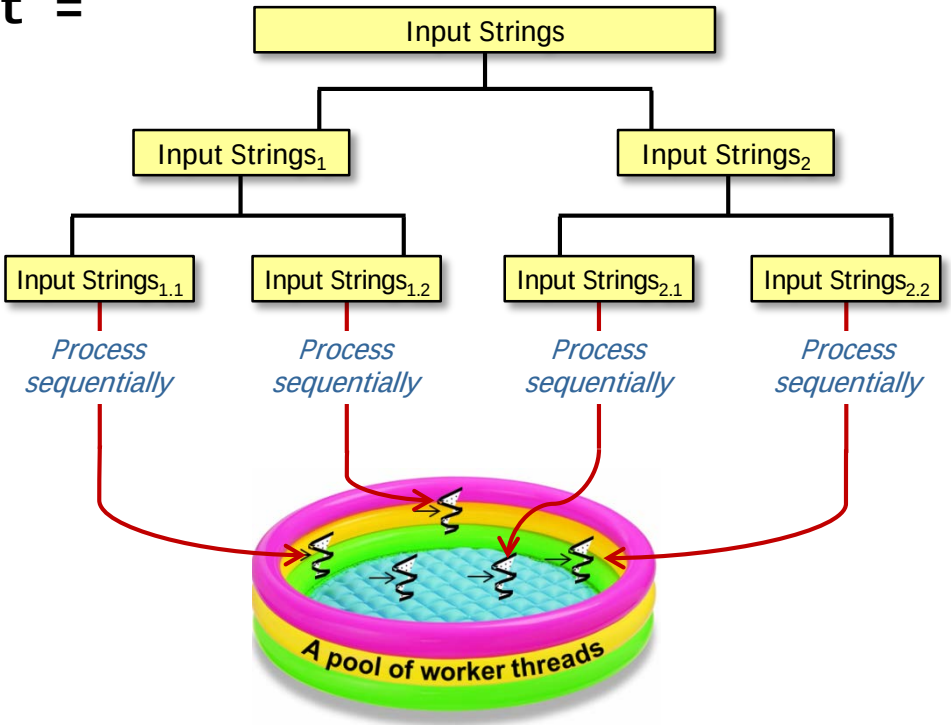
```
protected List<List<SearchResults>> processStream() {
  List<CharSequence> inputList =
    getInput();

  return getInput()

    .parallelStream()

    .map(this::processInput)

    .collect(toList());
}
```




"Chunks" of input strings are processed in parallel in the common fork-join pool

# Implementing processInput() as a Parallel Stream

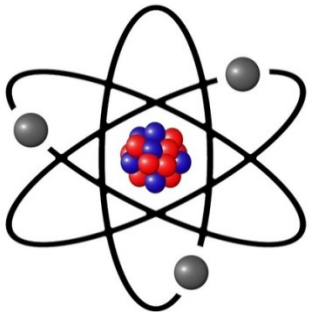
# Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputString);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```



# Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputString);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```

*Create a parallel stream that searches each input string to locate all occurrences of phases*

# Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputString);
  CharSequence input = inputSeq.subSequence(...);

  List<SearchResults> results = mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(phase, input, title, false))
    .filter(not(SearchResults::isEmpty))

    .collect(toList());
  return results;
}
```

*In this implementation strategy the spliterator is used to break the input into "chunks" that are processed sequentially*

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

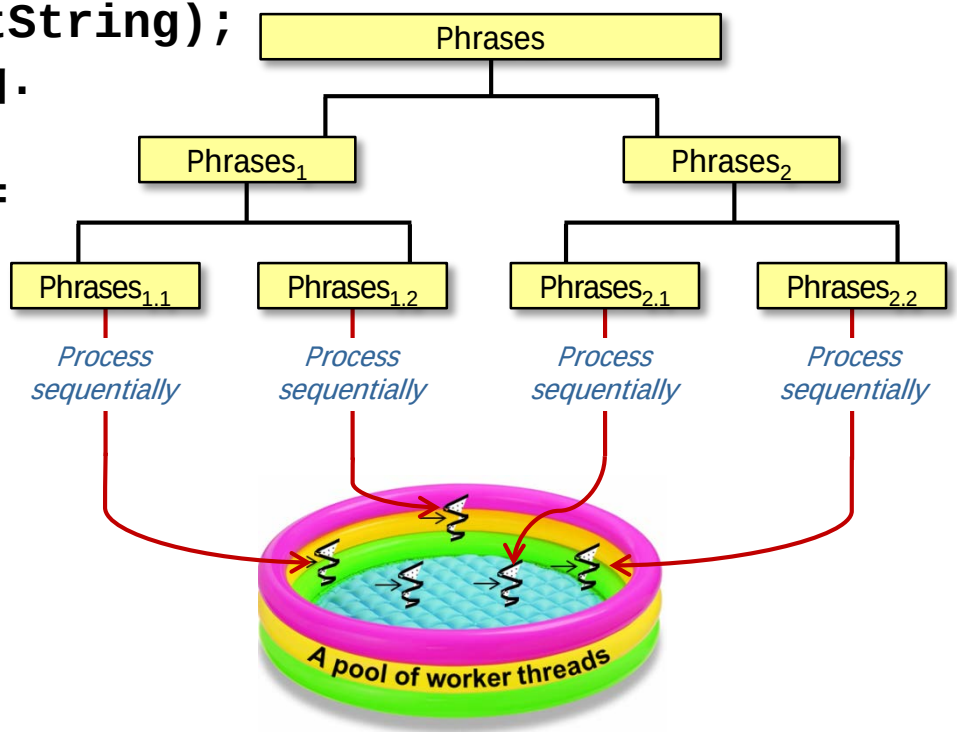
# Implementing processInput() as a Parallel Stream

- Likewise, this processInput() implementation has just one minuscule change

```
List<SearchResults> processInput(CharSequence inputSeq) {
  String title = getTitle(inputString);
  CharSequence input = inputSeq.

  List<SearchResults> results =
    .parallelStream()
    .map(phase ->
         searchForPhrase)
    .filter(not(SearchResults

    .collect(toList());
  return results;
}
```

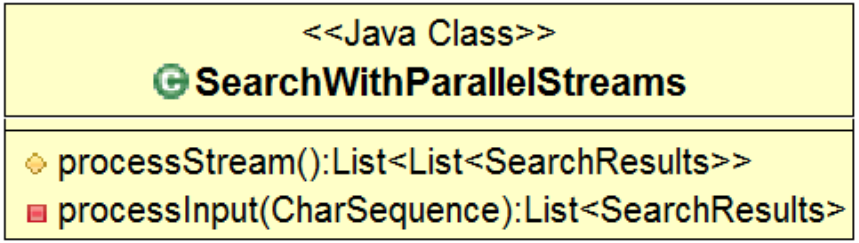


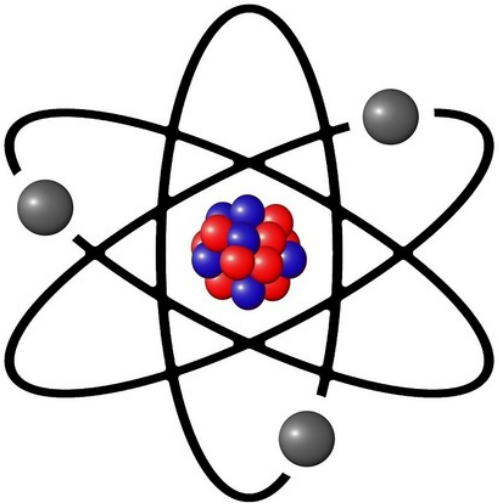

"Chunks" of phrases are processed in parallel in the common fork-join pool

# Pros of the SearchWith ParallelStreams Class

# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!



See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!

```
List<List<SearchResults>>
                processStream() {
  return getInput()
    .stream()
    .map(this::processInput)
    .collect(toList());
}
```

*Here's processStream() from SearchWithSequentialStream that we examined earlier*

# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!

```
List<List<SearchResults>>
                processStream() {
  return getInput()
    .stream()
    .map(this::processInput)
    .collect(toList());
}
```

vs

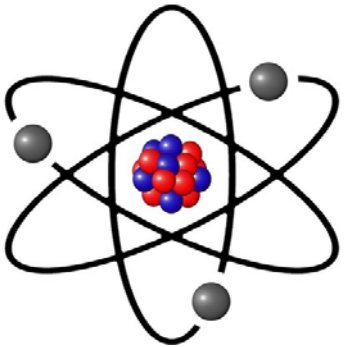
```
List<List<SearchResults>>
                processStream() {
  return getInput()
    .parallelStream()
    .map(this::processInput)
    .collect(toList());
}
```

*Here's processStream() in SearchWithParallelStreams*

# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!

*Changing all the stream() calls to parallelStream() calls is the minuscule difference between implementations!!*



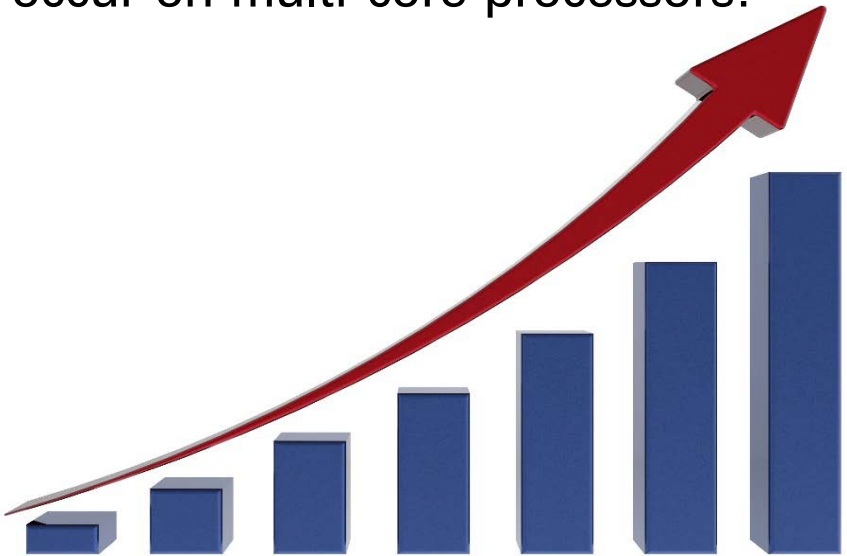
```
List<List<SearchResults>>
                processStream() {
  return getInput()
    .stream()
    .map(this::processInput)
    .collect(toList());
}
```

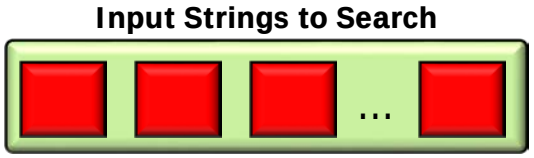
vs

```
List<List<SearchResults>>
                processStream() {
  return getInput()
    .parallelStream()
    .map(this::processInput)
    .collect(toList());
}
```

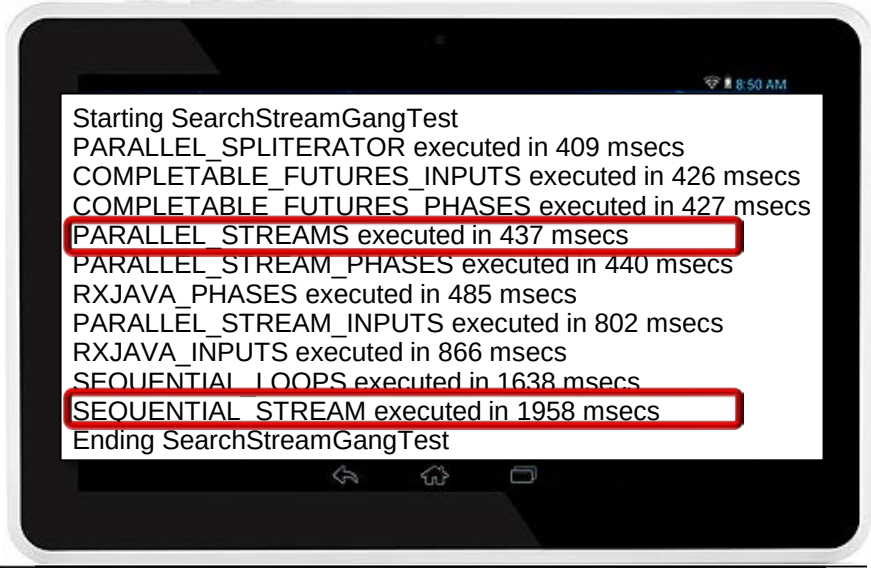
# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!
  - Moreover, substantial speedups can occur on multi-core processors!





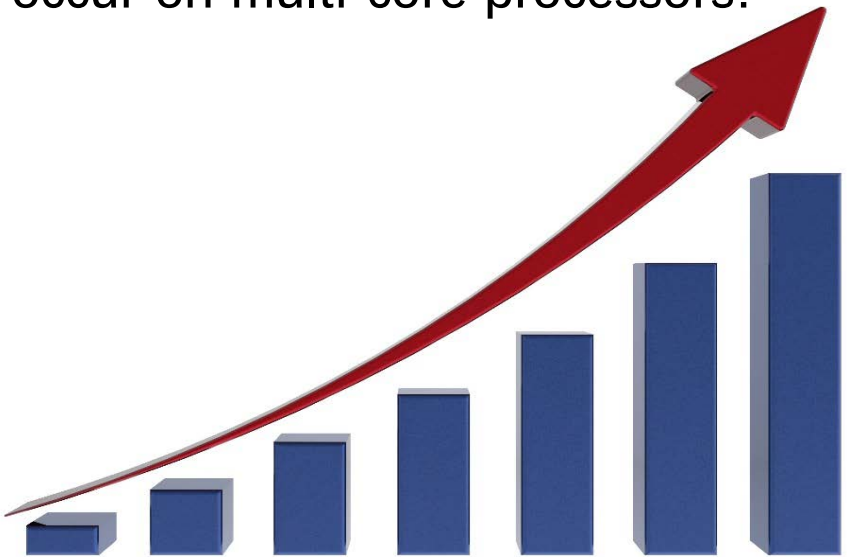


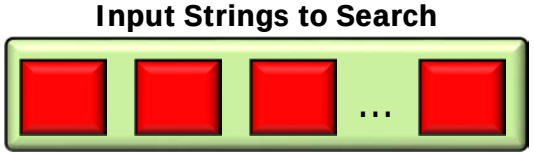



Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM

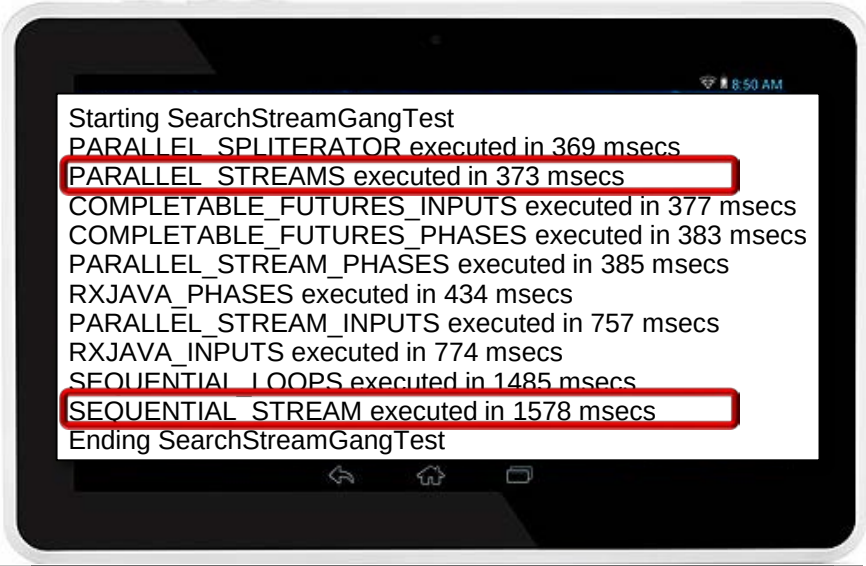
# Pros of the SearchWithParallelStreams Class

- This example shows that the difference between sequential & parallel streams is often minuscule!
  - Moreover, substantial speedups can occur on multi-core processors!







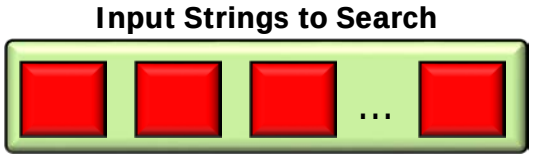



Tests conducted on a 2.9GHz quad-core MacBook Pro with 16 Gbytes of RAM

# Cons of the SearchWith ParallelStreams Class

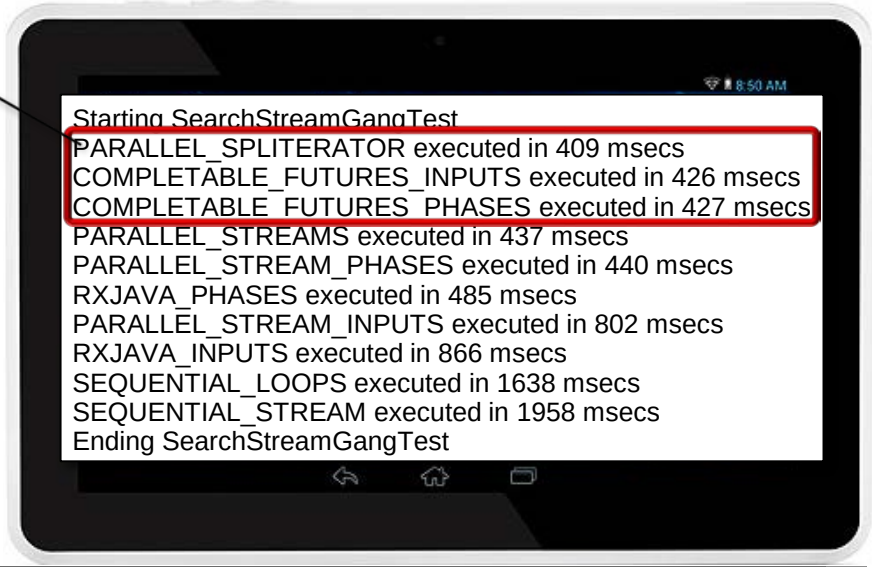
# Cons of the SearchWithParallelStreams Class

- Just because two minuscule changes are needed doesn't mean this is the best implementation!





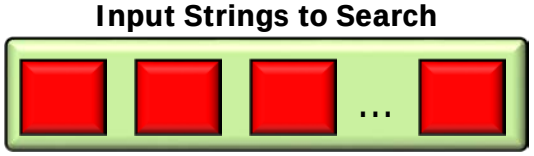
*Other Java 8 concurrency/parallelism strategies are even more efficient..*

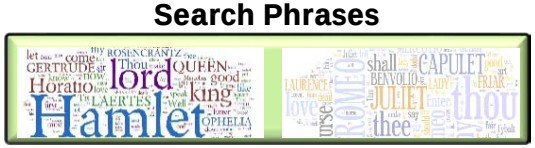


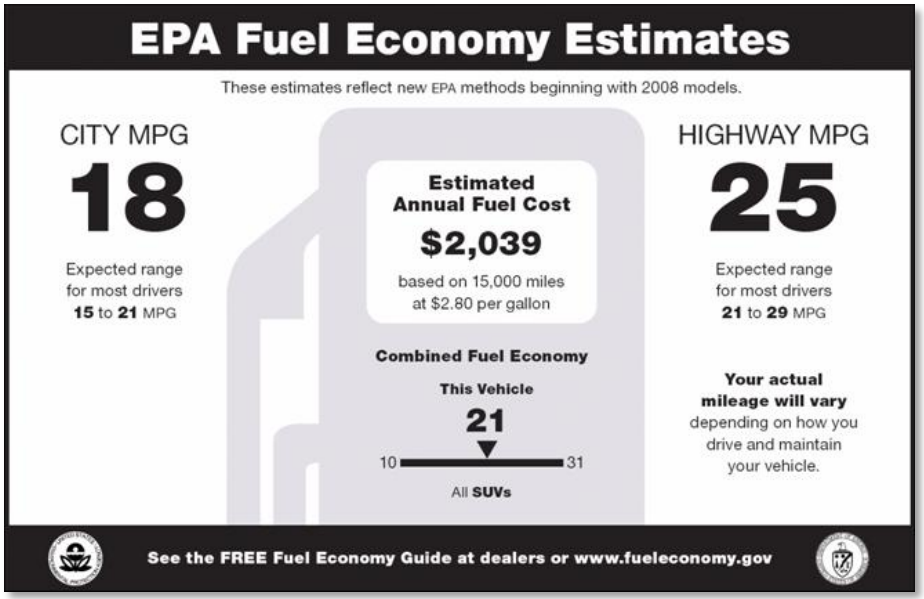

Tests conducted on a 2.7GHz quad-core Lenovo P50 with 32 Gbytes of RAM

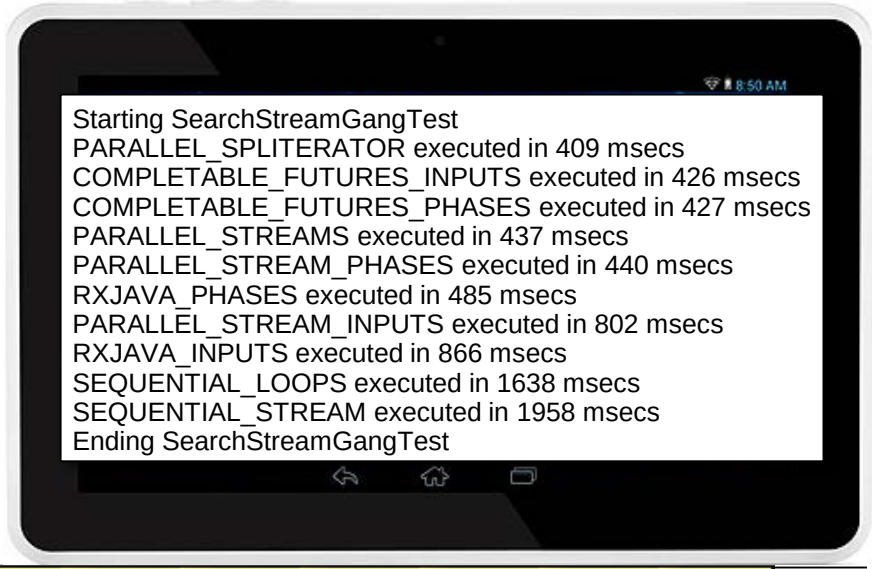
# Cons of the SearchWithParallelStreams Class

- Just because two minuscule changes are needed doesn't mean this is the best implementation!




There's no substitute for systematic benchmarking & experimentation

# End of Java 8 Parallel SearchStreamGang Example (Part 1)