

Java 8 Parallel SearchStreamGang

Example (Part 2)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software
Integrated Systems

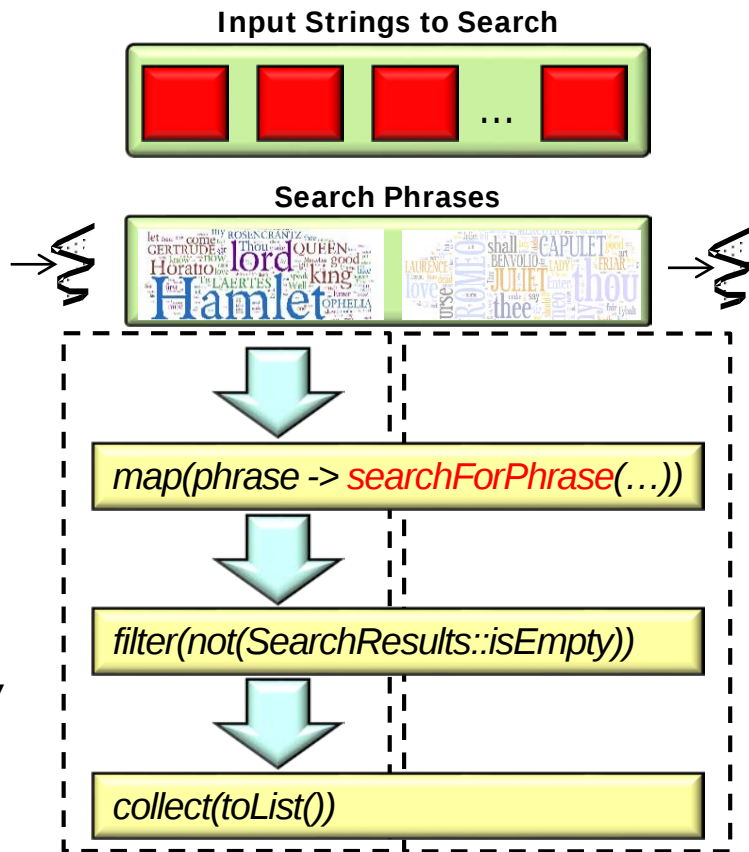
Vanderbilt University
Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

- Know how Java 8 parallel streams are applied in the SearchStreamGang
- Understand the pros & cons of the SearchWithParallelStreams class
- Recognize how a parallel spliterator can improve parallel stream performance

```
SearchResults searchForPhrase  
(..., boolean parallel) {  
    return new SearchResults  
        (... , StreamSupport.stream  
            (new PhraseMatchSpliterator(...),  
             parallel)  
            .collect(toList()));  
}
```



This solution addresses a “con” covered in the first part of this lesson

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
- Recognize how a parallel spliterator can improve parallel stream performance
- Understand the pros & cons of the SearchWithParallelSpliterator class

<<Java Class>>	
 SearchWithParallelSpliterator	
◆	processStream():List<List<SearchResults>>
■	processInput(CharSequence):List<SearchResults>



Overview of SearchWith ParallelSpliterator

- SearchWithParallelSpliterator is another implementation strategy in the SearchStreamGang program



Overview of SearchWithParallelSpliterator

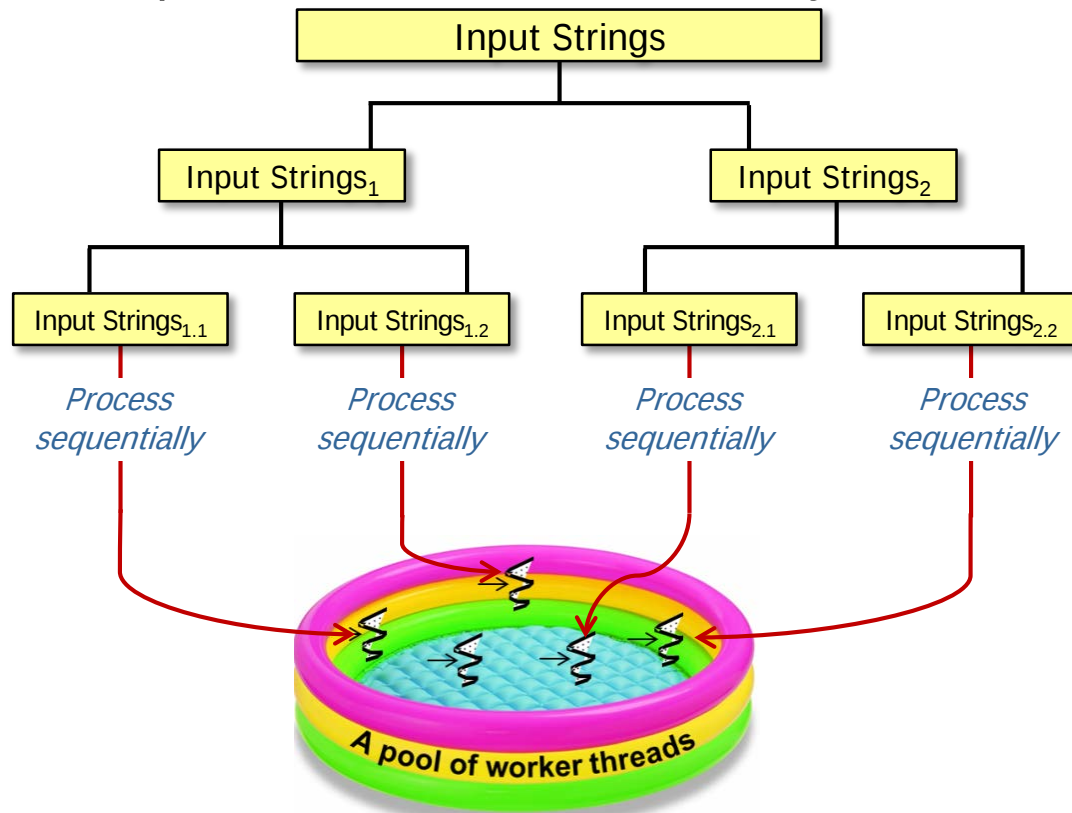
- SearchWithParallelSpliterator uses parallel streams in three ways

<<Java Class>>	
SearchWithParallelSpliterator	
◆	processStream():List<List<SearchResults>>
■	processInput(CharSequence):List<SearchResults>



Overview of SearchWithParallelSpliterator

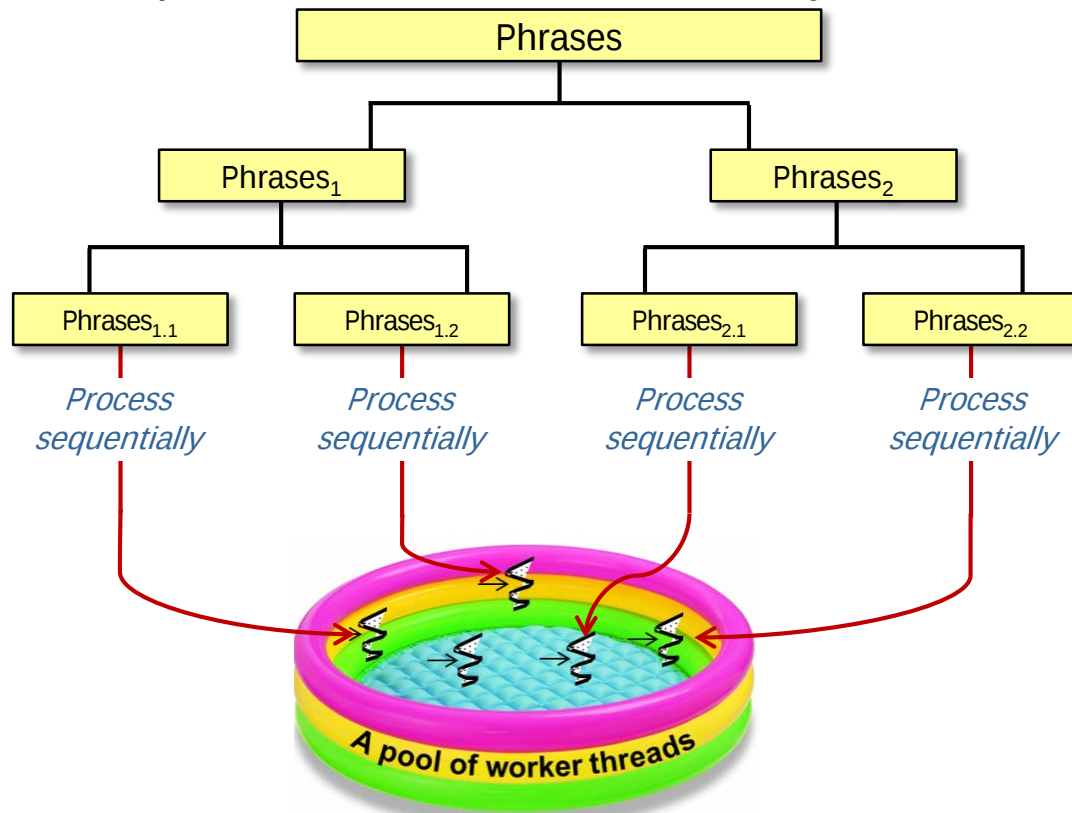
- SearchWithParallelSpliterator uses parallel streams in three ways
 - Search chunks of input in parallel



Overview of SearchWithParallelSpliterator

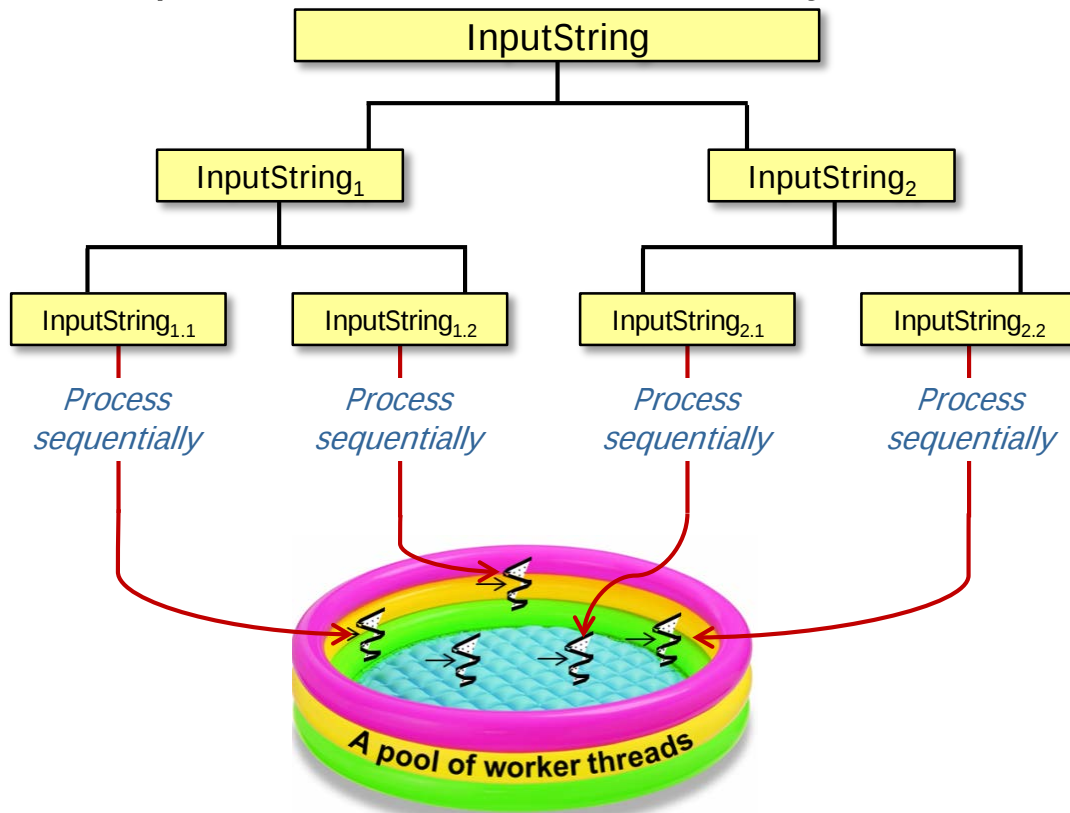
- SearchWithParallelSpliterator uses parallel streams in three ways

- Search chunks of input in parallel
- Search chunks of phrases in parallel



Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
 - Search chunks of input in parallel
 - Search chunks of phrases in parallel
 - Search chunks of each input string in parallel



Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator uses parallel streams in three ways
 - Search chunks of input in parallel
 - Search chunks of phrases in parallel
 - Search chunks of each input string in parallel



SearchWithParallelSpliterator is thus the most aggressively parallelism strategy!

Overview of SearchWithParallelSpliterator

- The relative contribution of each parallel streams model is shown here:
 - Time for 38 strings = 703 ms ([parallelSpliterator](#)|[parallelPhrases](#)|[parallelInput](#))
 - Time for 38 strings = 706 ms ([sequentialSpliterator](#)|[parallelPhrases](#)|[parallelInput](#))
 - Time for 38 strings = 726 ms ([parallelSpliterator](#)|[sequentialPhrases](#)|[parallelInput](#))
 - Time for 38 strings = 739 ms ([sequentialSpliterator](#)|[sequentialPhrases](#)|[parallelInput](#))
 - Time for 38 strings = 749 ms ([sequentialSpliterator](#)|[parallelPhrases](#)|[sequentialInput](#))
 - Time for 38 strings = 759 ms ([parallelSpliterator](#)|[parallelPhrases](#)|[sequentialInput](#))
 - Time for 38 strings = 1760 ms ([parallelSpliterator](#)|[sequentialPhrases](#)|[sequentialInput](#))
 - Time for 38 strings = 3000 ms([sequentialSpliterator](#)|[sequentialPhrases](#)|[sequentialInput](#))

Overview of SearchWithParallelSpliterator

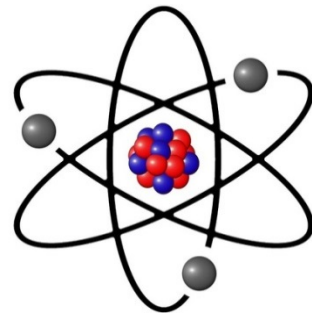
- Longer input strings leverage the parallel spliterator even better:
 - Time for 2 strings = 700 ms (parallelSpliterator|parallelPhrases|parallelInput)
 - Time for 2 strings = 738 ms (parallelSpliterator|parallelPhrases|sequentialInput)
 - Time for 2 strings = 761 ms (sequentialSpliterator|parallelPhrases|parallelInput)
 - Time for 2 strings = 780 ms (sequentialSpliterator|parallelPhrases|sequentialInput)
 - Time for 2 strings = 1008 ms (parallelSpliterator|sequentialPhrases|parallelInput)
 - Time for 2 strings = 1617 ms (parallelSpliterator|sequentialPhrases|sequentialInput)
 - Time for 2 strings = 1986 ms (sequentialSpliterator|sequentialPhrases|parallelInput)
 - Time for 2 strings = 2870 ms (sequentialSpliterator|sequentialPhrases|sequentialInput)

Longer strings may provide better opportunity to leverage benefits of parallelism

Overview of SearchWithParallelSpliterator

- SearchWithParallelSpliterator processInput() 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, true))  
        .filter(not(SearchResults::isEmpty))  
  
        .collect(toList());  
    return results;  
}
```



The value of “true” triggers the use of a parallel search for a phrase in an input string

Overview of SearchWithParallelSpliterator

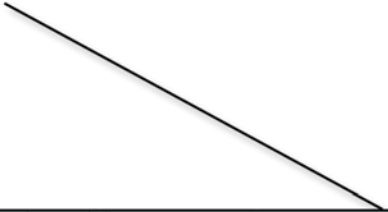
- `searchForPhrase()` uses a parallel spliterator to break the input into “chunks” that are processed in parallel

```
SearchResults searchForPhrase(String phrase, CharSequence input,  
                                String title, boolean parallel) {  
    return new SearchResults  
        (... , ..., phrase, title, StreamSupport  
            .stream(new PhraseMatchSpliterator(input, phrase),  
                parallel)  
            .collect(toList()));  
}
```

Overview of SearchWithParallelSpliterator

- `searchForPhrase()` uses a parallel spliterator to break the input into “chunks” that are processed in parallel

```
SearchResults searchForPhrase(String phrase, CharSequence input,  
                             String title, boolean parallel) {  
    return new SearchResults  
        (... , ..., phrase, title, StreamSupport  
          .stream(new PhraseMatchSpliterator(input, phrase),  
                parallel)  
          .collect(toList()));  
}
```



*StreamSupport.stream() creates a sequential
or parallel stream via PhraseMatchSpliterator*

Overview of SearchWithParallelSpliterator

- searchForPhrase() uses a parallel spliterator to break the input into “chunks” that are processed in parallel

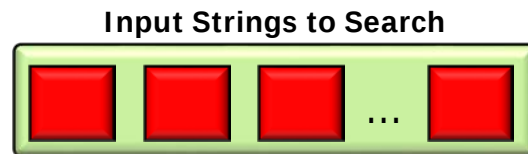
```
SearchResults searchForPhrase(String phrase, CharSequence input,  
                             String title, boolean parallel) {  
    return new SearchResults  
        (... , ..., phrase, title, StreamSupport  
            .stream(new PhraseMatchSpliterator(input, phrase),  
                parallel)  
            .collect(toList()));  
}
```

The value of “parallel” is true when searchForPhrase() is called in the SearchWithParallelSpliterator program

Using Parallel Splitter in SearchStreamGang

Using a Parallel Splitter in SearchStreamGang

- SearchStreamGang uses PhraseMatchSplitter that works for both sequential & parallel streams



`map(phrase -> searchForPhrase(...))`



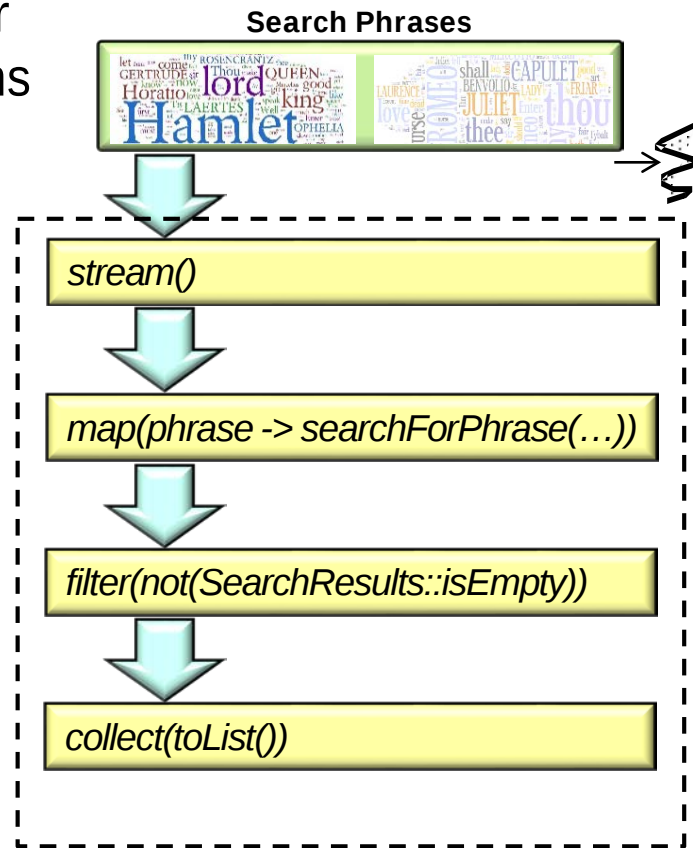
`filter(not(SearchResults::isEmpty))`



`collect(toList())`

Using a Parallel Spliter in SearchStreamGang

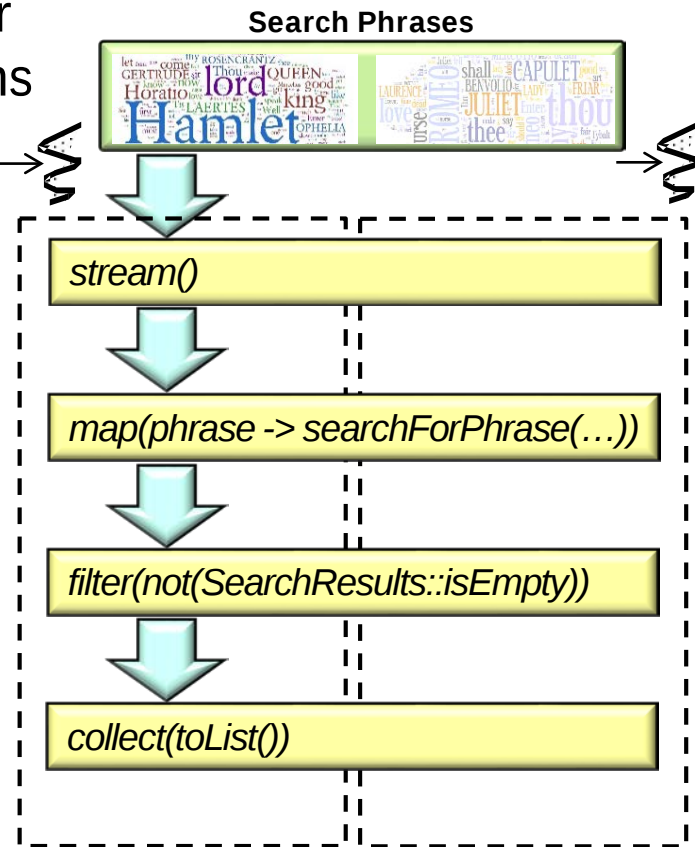
- SearchStreamGang uses PhraseMatchSplitter that works for both sequential & parallel streams
- We focused on the sequential portions earlier



See “Java 8 Sequential SearchStreamGang Example (Part 2)”

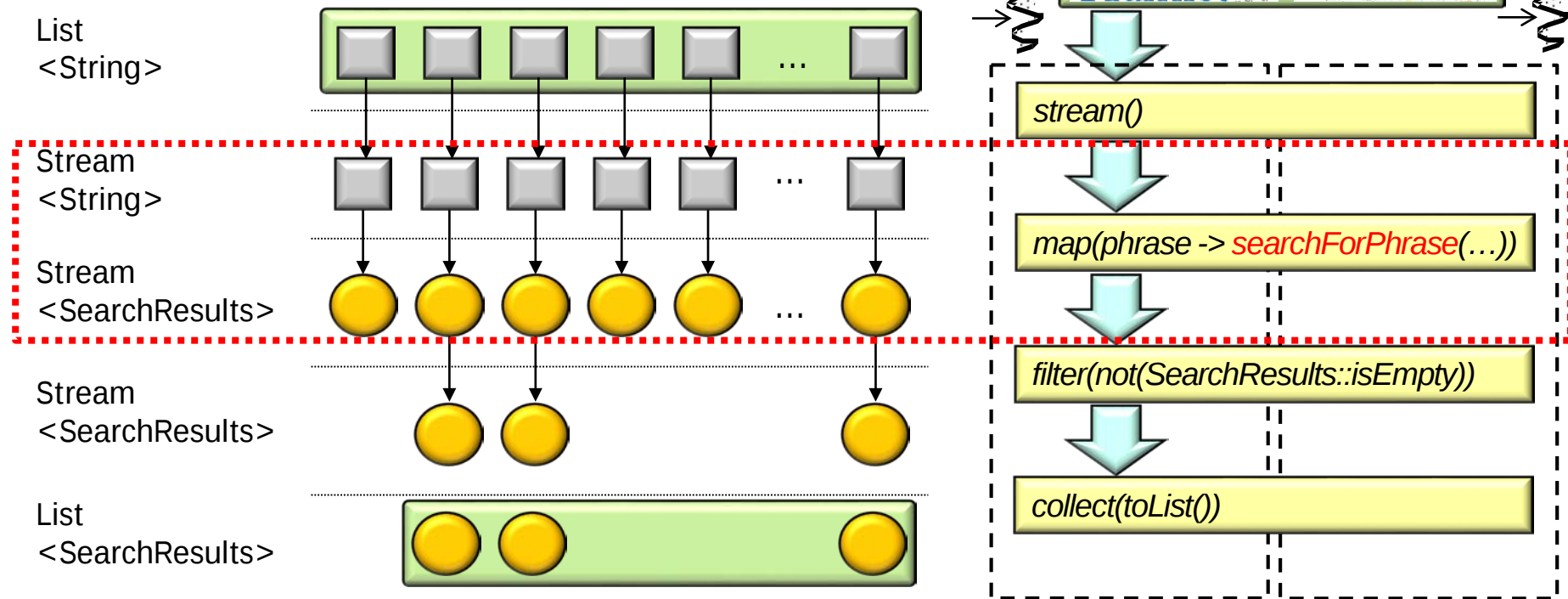
Using a Parallel Splitter in SearchStreamGang

- SearchStreamGang uses PhraseMatchSplitter that works for both sequential & parallel streams
 - We focused on the sequential portions earlier
 - We'll cover the parallel portions now



The goal is to further optimize the performance of the parallel streams solution

Using a Parallel Spliterator in SearchStreamGang



Using a Parallel Spliterator in SearchStreamGang

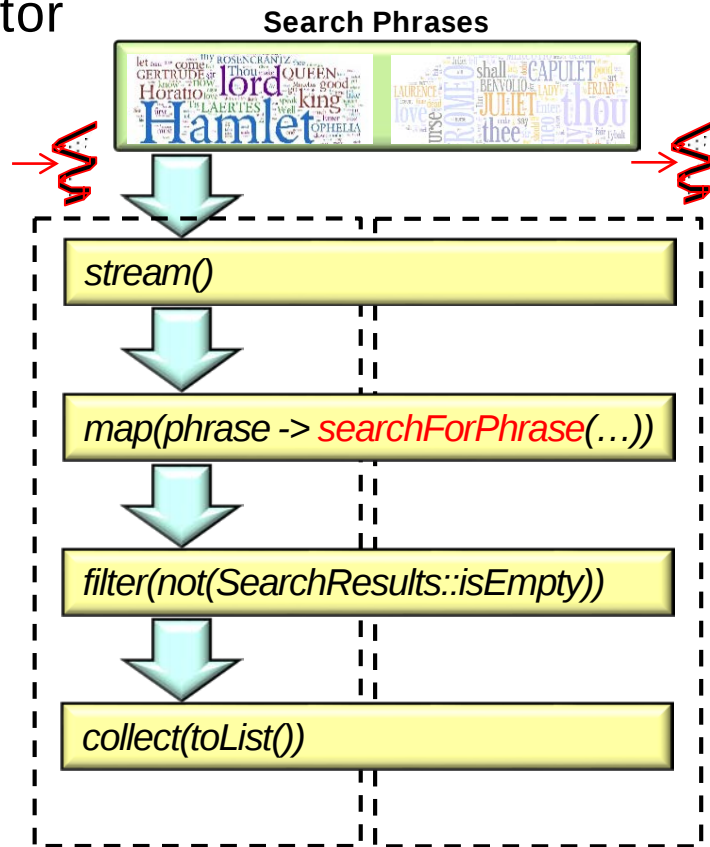
- Here's the input/output of PhraseMatchSpliterator for SearchWithParallelSpliterator

“ ...

“Brevity is the soul of wit” not found!

My liege, and madam, to expostulate
What majesty should be, what duty is,
Why day is day, night is night, and time is time.
Were nothing but to waste night, day, and time.
Therefore, since **brevity is the soul of**”

“**wit**, And tediousness the limbs and outward
flourishes, I will be brief. Your noble son is mad.
Mad call I it; for, to define true madness,
What is't but to be nothing else but mad?
But let that go....”



However, the spliterator must be careful not to split input *across* phrases...

Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {  
    ...  
    PhraseMatchSpliterator  
        (CharSequence input,  
         String phrase) { ... }  
  
    boolean tryAdvance  
        (Consumer<? super Result> action)  
    { ... }  
    ...  
}
```

Spliterator is an interface that defines eight methods, including tryAdvance() & trySplit()

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
PhraseMatchSplitter
```

```
(CharSequence input,  
String phrase) { ... }
```

*Earlier we analyzed several
of its methods that are
used in sequential streams*

```
boolean tryAdvance
```

```
(Consumer<? super Result> action)  
{ ... }
```

```
...
```

See “Java 8 Sequential SearchStreamGang Example (Part 2)”

Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
```

```
...
```

```
Spliterator<Result> trySplit() { ... }
```

```
int computeStartPos(int splitPos) { ... }
```

```
int tryToUpdateSplitPos(int startPos,  
                        int splitPos) { ... }
```

```
PhraseMatchSpliterator splitInput(int splitPos) { ... }
```

```
...
```

*We'll now explore its methods
that are used for parallel streams*

Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
```

```
...
```

```
Spliterator<Result> trySplit() { ... }
```

```
int computeStartPos(int splitPos) { ... }
```

```
int tryToUpdateSplitPos(int startPos,  
                        int splitPos) { ... }
```

```
PhraseMatchSpliterator splitInput(int splitPos) { ... }
```

```
...
```



*We'll now explore its methods
that are used for parallel streams*

Note that there is *no* synchronization in any of these methods!!!

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {
```

```
...
```

```
Spliterator<Result> trySplit() {
```

```
    if (input is below minimum size) return null
```

```
    else {
```

```
        split input in 2 relatively
```

```
        even-sized chunks
```

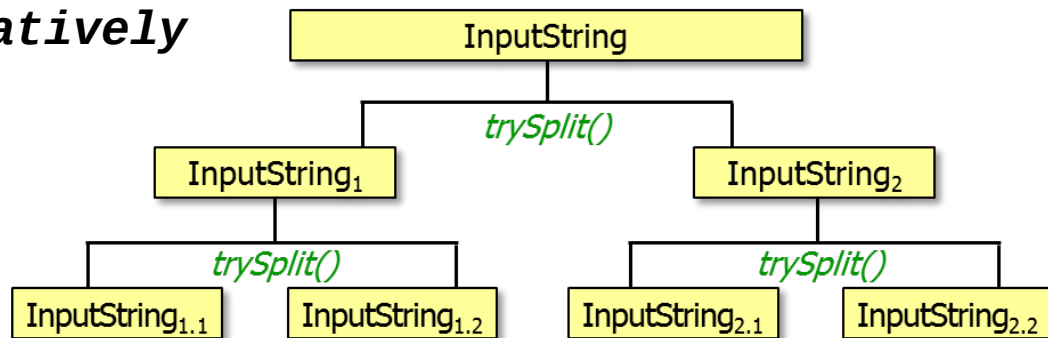
```
        return a spliterator
```

```
        for "left chunk"
```

```
    }
```

```
}
```

```
...
```



`trySplit()` attempts to split the input “evenly” so phrases can be matched in parallel

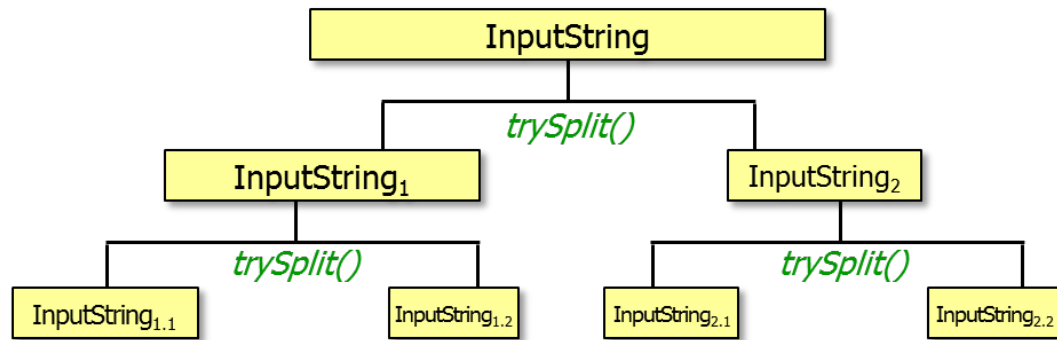
Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

class PhraseMatchSplitter implements Splitter<Result> {

...

Splitter<Result> trySplit() {



Splits don't needn't be perfectly equal in order for the splitter to be efficient

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {  
    ...  
    Splitter<Result> trySplit() {  
        if (mInput.length() <= mMinSplitSize) return null;  
        int startPos,  
            splitPos = mInput.length() / 2;  
  
        if ((startPos = computeStartPos(splitPos)) < 0) return null;  
  
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)  
            return null;  
  
        return splitInput(splitPos); ...  
    }  
}
```

This code is heavily commented, so please check it out

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
Splitter<Result> trySplit() {
```

```
    if (mInput.length() <= mMinSplitSize) return null;
```

```
    int startPos,
```

```
        splitPos = mInput.length() / 2;
```

Bail out if input is too small to split further

```
    if ((startPos = computeStartPos(splitPos)) < 0) return null;
```

```
    if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)
        return null;
```

```
    return splitInput(splitPos); ...
```

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
Splitter<Result> trySplit() {
```

```
    if (mInput.length() <= mMinSplitSize) return null;
```

```
    int startPos,
```

```
        splitPos = mInput.length() / 2;
```

*Initial guess at
the split position*

```
    if ((startPos = computeStartPos(splitPos)) < 0) return null;
```

```
    if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)  
        return null;
```

```
    return splitInput(splitPos); ...
```

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {  
    ...  
    Splitter<Result> trySplit() {  
        if (mInput.length() <= mMinSplitSize) return null;  
        int startPos,  
            splitPos = mInput.length() / 2;  
  
        if ((startPos = computeStartPos(splitPos)) < 0) return null;  
  
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)  
            return null;  
  
        return splitInput(splitPos); ...  
    }  
}
```

*Initial guess at where
to start the search*

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {  
    ...  
    Splitter<Result> trySplit() {  
        if (mInput.length() <= mMinSplitSize) return null;  
        int startPos,  
            splitPos = mInput.length() / 2;  
  
        if ((startPos = computeStartPos(splitPos)) < 0) return null;  
  
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)  
            return null;  
  
        return splitInput(splitPos); ...  
    }  
}
```

*Update splitPos if phrase
spans the initial splitPos*

Using a Parallel Spliterator in SearchStreamGang

- PhraseMatchSpliterator uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSpliterator implements Spliterator<Result> {  
    ...  
    Spliterator<Result> trySplit() {  
        if (mInput.length() <= mMinSplitSize) return null;  
        int startPos,  
            splitPos = mInput.length() / 2;  
  
        if ((startPos = computeStartPos(splitPos)) < 0) return null;  
  
        if ((splitPos = tryToUpdateSplitPos(startPos, splitPos)) < 0)  
            return null;  
  
        return splitInput(splitPos); ...  
    }  
}
```

*Create & return a
new spliterator*

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
int tryToUpdateSplitPos(int startPos, int splitPos) {
```

```
    int endPos =
```

```
        splitPos + mPattern.toString().length();
```

```
    if (endPos >= mInput.length()) return -1;
```

```
    CharSequence substr =
```

```
        mInput.subSequence(startPos, endPos);
```

```
    Matcher pm = mPattern.matcher(substr);
```

```
    if (pm.find()) splitPos = startPos
```

```
        + pm.start() + pm.group().length();
```

```
    return splitPos;
```

```
}
```

*Don't split a string
across a phrase*

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
int tryToUpdateSplitPos(int startPos, int splitPos) {
```

```
    int endPos =
```

```
        splitPos + mPattern.toString().length();
```

```
    if (endPos >= mInput.length()) return -1;
```

```
    CharSequence substr =
```

```
        mInput.subSequence(startPos, endPos);
```

```
    Matcher pm = mPattern.matcher(substr);
```

```
    if (pm.find()) splitPos = startPos
```

```
        + pm.start() + pm.group().length();
```

```
    return splitPos;
```

```
}
```

*Ensure phrase
isn't longer than
the input string!*

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
int tryToUpdateSplitPos(int startPos, int splitPos) {
```

```
    int endPos =
```

```
        splitPos + mPattern.toString().length();
```

```
    if (endPos >= mInput.length()) return -1;
```

```
    CharSequence substr =
```

```
        mInput.subSequence(startPos, endPos);
```

```
    Matcher pm = mPattern.matcher(substr);
```

```
    if (pm.find()) splitPos = startPos
```

```
        + pm.start() + pm.group().length();
```

```
    return splitPos;
```

```
}
```

Check to see if the phrase spans across the initial splitPos

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
int tryToUpdateSplitPos(int startPos, int splitPos) {
```

```
    int endPos =
```

```
        splitPos + mPattern.toString().length();
```

```
    if (endPos >= mInput.length()) return -1;
```

```
    CharSequence substr =
```

```
        mInput.subSequence(startPos, endPos);
```

```
    Matcher pm = mPattern.matcher(substr);
```

```
    if (pm.find()) splitPos = startPos
```

```
        + pm.start() + pm.group().length();
```

```
    return splitPos;
```

```
}
```

If there's a match update the splitPos to handle phrase spanning newlines

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
int tryToUpdateSplitPos(int startPos, int splitPos) {
```

```
    int endPos =
```

```
        splitPos + mPattern.toString().length();
```

```
    if (endPos >= mInput.length()) return -1;
```

```
    CharSequence substr =
```

```
        mInput.subSequence(startPos, endPos);
```

```
    Matcher pm = mPattern.matcher(substr);
```

```
    if (pm.find()) splitPos = startPos
```

```
        + pm.start() + pm.group().length();
```

```
    return splitPos;
```

```
}
```

*Return the
final splitPos*

Using a Parallel Splitter in SearchStreamGang

- PhraseMatchSplitter uses Java regex to create a stream of SearchResults Result objects that match the # of times a phrase appears in an input string

```
class PhraseMatchSplitter implements Splitter<Result> {
```

```
...
```

```
Splitter<Result> splitInput(int splitPos) {
```

```
    CharSequence lhs = mInput.subSequence(0, splitPos);
```

```
    mInput = mInput.subSequence(splitPos, mInput.length());
```

```
    mPhraseMatcher = mPattern.matcher(mInput);
```

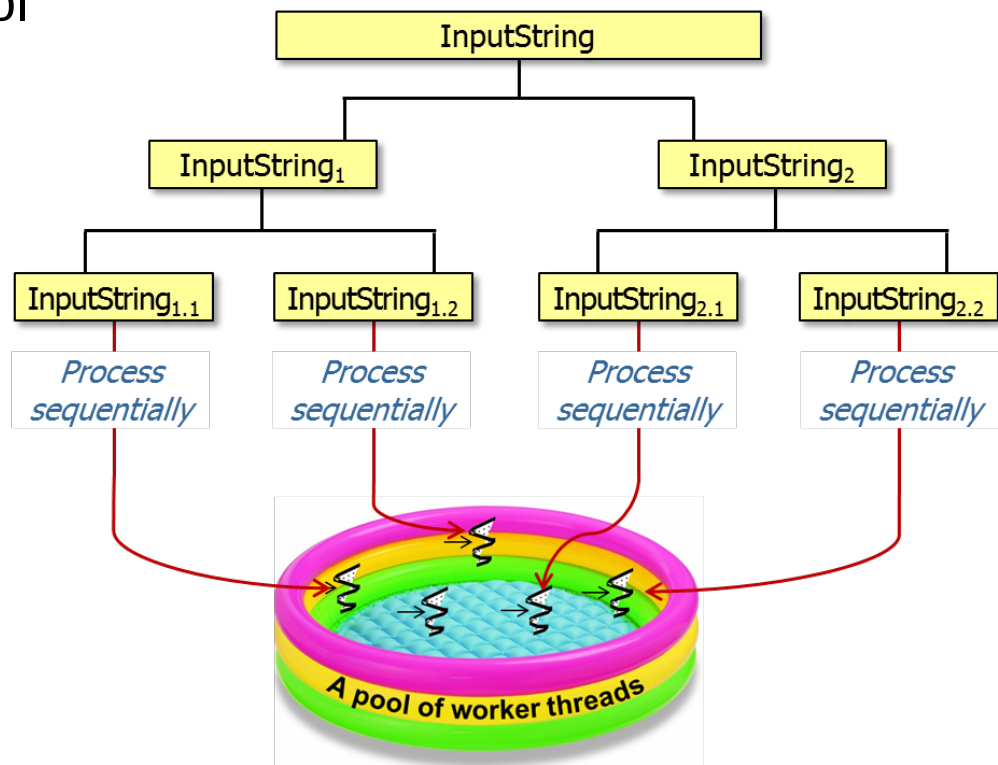
```
...
```

*Return a Splitter that handles “left hand” portion of input,
while “this” object handles “right hand” portion of input*

```
return new PhraseMatchSplitter(lhs, ...); ...
```

Using a Parallel Splitter in SearchStreamGang

- The Java 8 parallel streams runtime processes all the splitter chunks in parallel in the common fork-join pool



This parallelism is in addition to parallelism of input string & phrase chunks!!

Pros of the SearchWith ParallelSpliterator Class

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance

Input Strings to Search



Search Phrases



```
Starting SearchStreamGangTest
PARALLEL_SPLITATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS executed in 802 msecs
RXJAVA_INPUTS executed in 866 msecs
SEQUENTIAL_LOOPS executed in 1638 msecs
SEQUENTIAL_STREAM executed in 1958 msecs
Ending SearchStreamGangTest
```

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance

Input Strings to Search



Search Phrases



Starting SearchStreamGangTest

PARALLEL_SPLITERATOR executed in 409 msec

COMPLETABLE_FUTURES_INPUTS executed in 426 msec

COMPLETABLE_FUTURES_PHASES executed in 427 msec

PARALLEL_STREAMS executed in 437 msec

PARALLEL_STREAM_PHASES executed in 440 msec

RXJAVA_PHASES executed in 485 msec

PARALLEL_STREAM_INPUTS executed in 802 msec

RXJAVA_INPUTS executed in 866 msec

SEQUENTIAL_LOOPS executed in 1638 msec

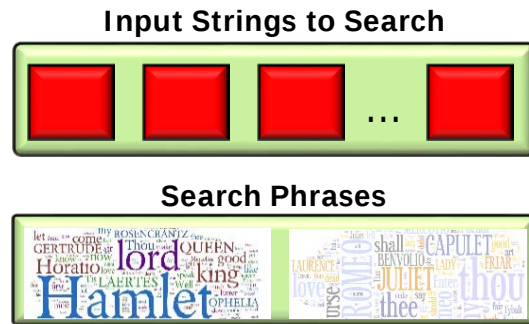
SEQUENTIAL_STREAM executed in 1958 msec

Ending SearchStreamGangTest

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

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance



Starting SearchStreamGangTest

PARALLEL_SPLITERATOR executed in 369 msecs

PARALLEL_STREAMS executed in 373 msecs

COMPLETABLE_FUTURES_INPUTS executed in 377 msecs

COMPLETABLE_FUTURES_PHASES executed in 383 msecs

PARALLEL_STREAM_PHASES executed in 385 msecs

RXJAVA_PHASES executed in 434 msecs

PARALLEL_STREAM_INPUTS executed in 757 msecs

RXJAVA_INPUTS executed in 774 msecs

SEQUENTIAL_LOOPS executed in 1485 msecs

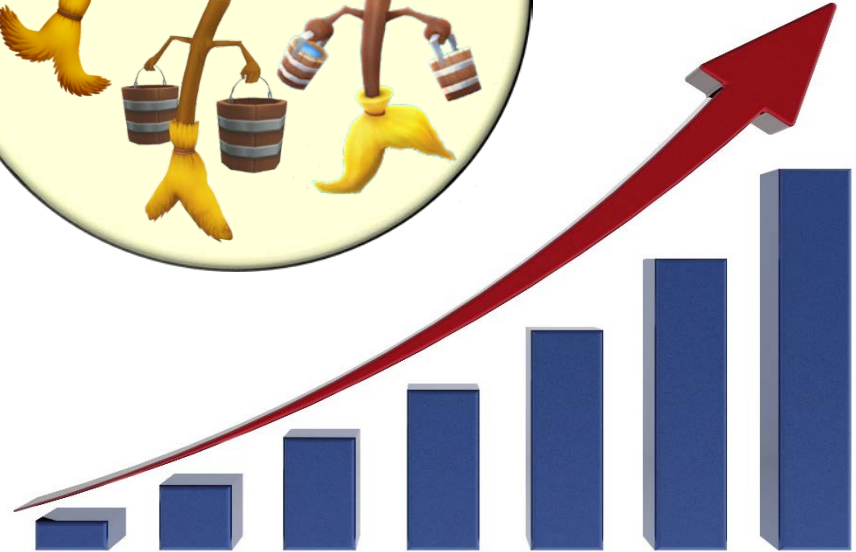
SEQUENTIAL_STREAM executed in 1578 msecs

Ending SearchStreamGangTest

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

Pros of the SearchWithParallelSpliterator Class

- This example shows how a parallel spliterator can help transparently improve program performance
- These speedups occur since the granularity of parallelism is smaller & thus better able to leverage available cores

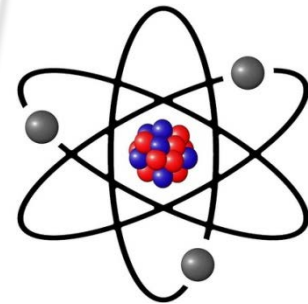


See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Pros of the SearchWithParallelSpliterator Class

- This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

```
SearchResults searchForPhrase(String phrase, CharSequence input,  
                             String title, boolean parallel) {  
    return new SearchResults  
        (... , ..., phrase, title, StreamSupport  
            .stream(new PhraseMatchSpliterator(input,  
                                                phrase),  
                parallel)  
            .collect(toList()));  
}
```



Switching this boolean from “false” to “true” controls whether the spliterator runs sequentially or in parallel

Pros of the SearchWithParallelSpliterator Class

- This example also shows that the difference between using sequential vs parallel spliterator can be minuscule!

```
SearchResults searchForPhrase(String phrase, CharSequence input,  
                              String title, boolean parallel) {  
    return new SearchResults  
        (... , ..., phrase, title, StreamSupport  
            .stream(new PhraseMatchSpliterator(input,  
                                                    phrase),  
                parallel)  
            .collect(toList()));  
}
```



Of course, it took non-trivial time/effort to create `PhraseMatchSpliterator`..

Cons of the SearchWith ParallelSpliterator Class

Cons of the SearchWithParallelSpliterator Class

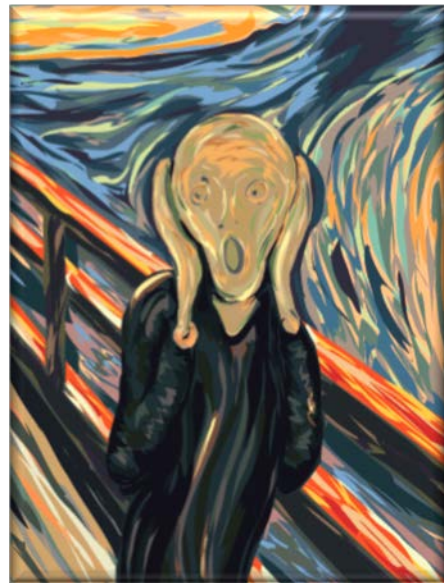
- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```
class PhraseMatchSpliterator
    implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ... }

    int tryToUpdateSplitPos(int startPos,
                           int splitPos)
        { ... }
```

```
PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
```



Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```
class PhraseMatchSpliterator
    implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ... }

    int tryToUpdateSplitPos(int startPos,
                           int splitPos)
    { ... }

    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
}
```

Must split carefully..



JUnit tests are extremely useful..

Cons of the SearchWithParallelSpliterator Class

- The parallel-related portions of PhraseMatchSpliterator are *much* more complicated to program than the sequential-related portions...

```
class PhraseMatchSpliterator
    implements Spliterator<Result> {
    ...
    Spliterator<Result> trySplit() { ... }

    int computeStartPos(int splitPos) { ...

    int tryToUpdateSplitPos(int startPos,
                            int splitPos)
        { ... }

    PhraseMatchSpliterator splitInput(int splitPos) { ... }
    ...
}
```



Writing the parallel spliterator took longer than writing the rest of the program!

End of Java 8 Parallel SearchStreamGang Example (Part 2)