Overview of Java 8 Parallel Streams (Part 3)

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

- Recognize how Java 8 applies aggregate operations & functional programming features in the parallel streams framework
- Be able to avoid concurrency hazards in parallel streams
- Understand how a parallel stream splits its elements recursively, processes them independently & combines the results
- Know when to use parallel streams

Learning Objectives in this Part of the Lesson

- Recognize how Java 8 applies aggregate operations & functional programming features in the parallel streams framework
- Be able to avoid concurrency hazards in parallel streams
- Understand how a parallel stream splits its elements recursively, processes them independently & combines the results
- Know when to use parallel streams
  - & when *not* to use parallel streams

When to Use Java 8 Parallel Streams

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions

See gee.cs.oswego.edu/dl/html/StreamParallelGuidance.html

When to Use Java 8 Parallel Streams

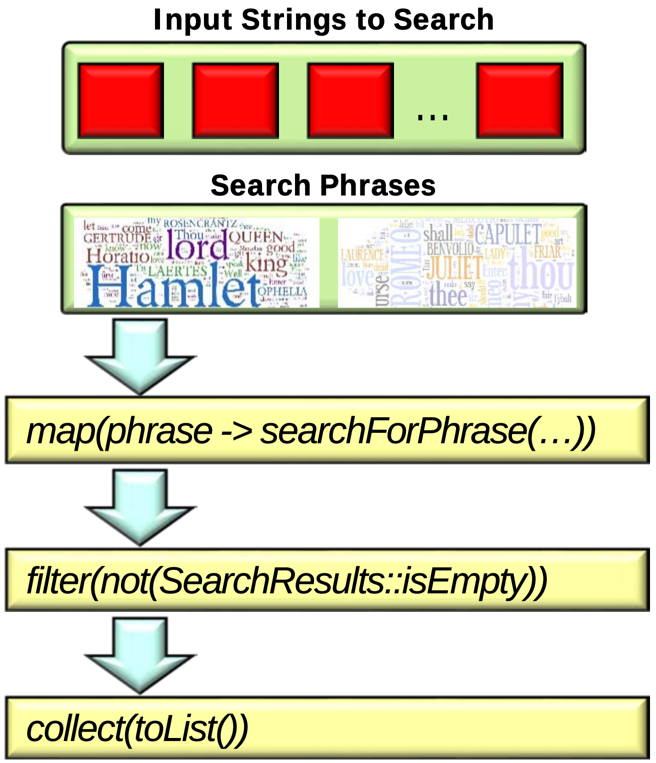
- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent

"Embarrassingly parallel" tasks have little/no dependency or need for communication between tasks or for sharing results between them

See en.wikipedia.org/wiki/Embarrassingly_parallel

When to Use Java 8 Parallel Streams

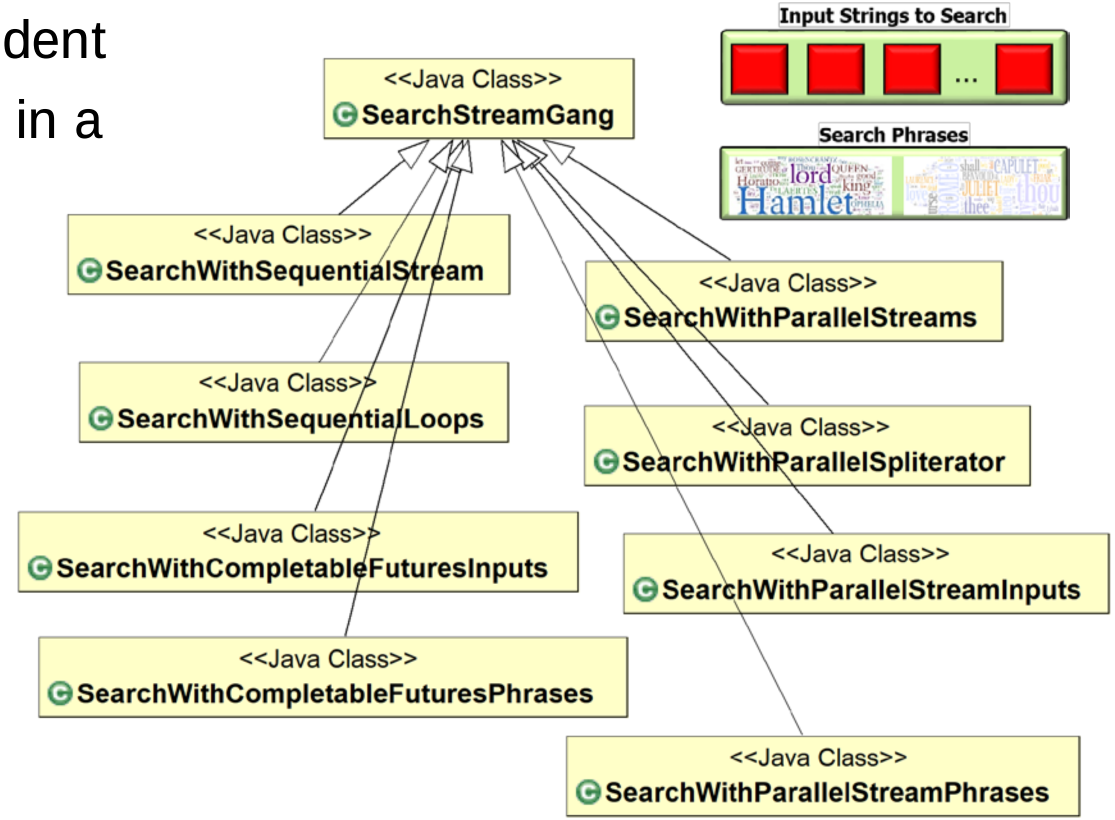
- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings


Input Strings to Search
...
Search Phrases
map(phrase -> searchForPhrase(…))
filter(not(SearchResults::isEmpty))
collect(toList())

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

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings

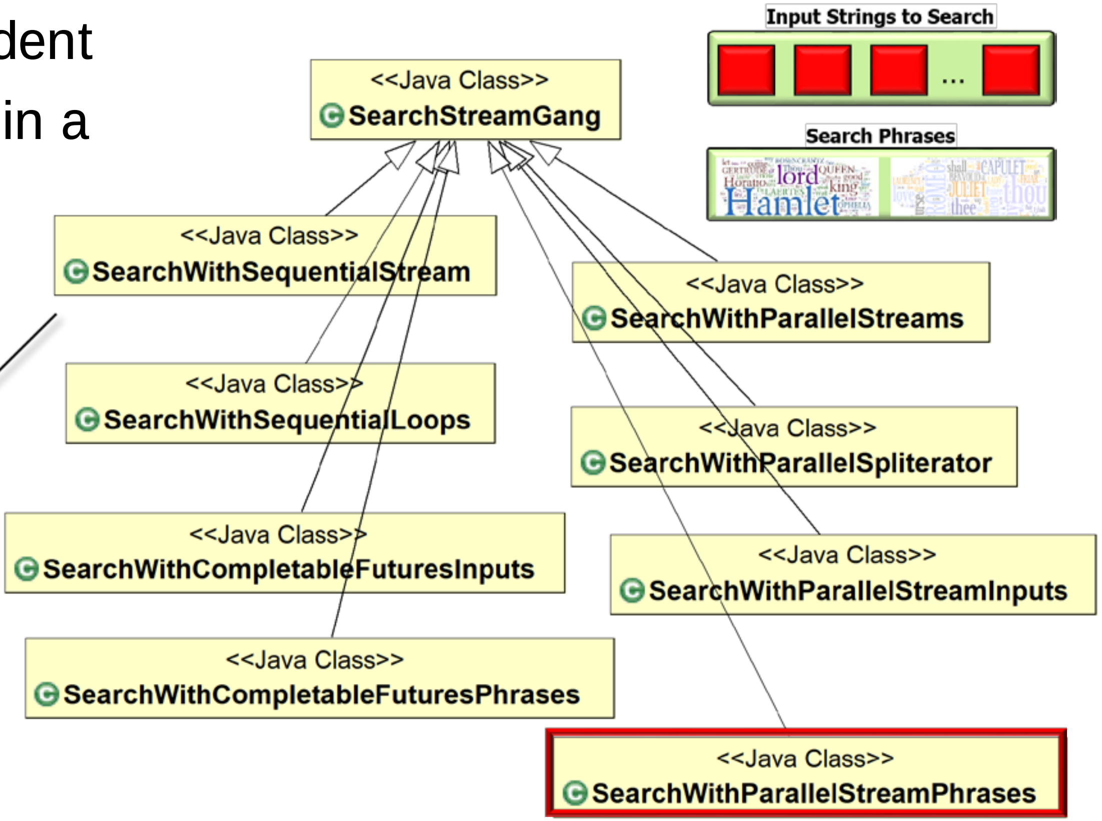


Input Strings to Search
...
Search Phrases
<<Java Class>>
SearchStreamGang
<<Java Class>>
SearchWithSequentialStream
<<Java Class>>
SearchWithSequentialLoops
<<Java Class>>
SearchWithCompletableFuturesInputs
<<Java Class>>
SearchWithCompletableFuturesPhrases
<<Java Class>>
SearchWithParallelStreams
<<Java Class>>
SearchWithParallelSpliterator
<<Java Class>>
SearchWithParallelStreamInputs
<<Java Class>>
SearchWithParallelStreamPhrases

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings

Parallel streams can be used to:
- *search chunks of phrases concurrently*
- *search chunks of input concurrently*
- *search chunks of each input string concurrently*

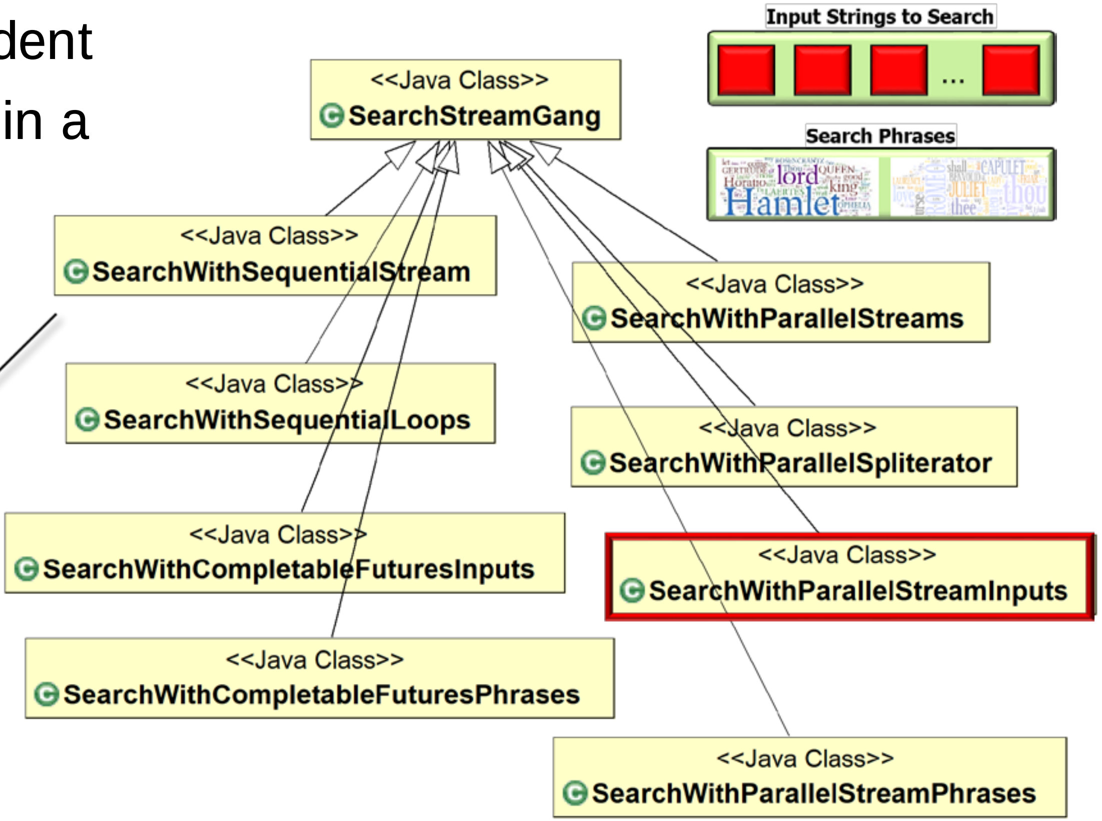

Input Strings to Search
...
Search Phrases
<<Java Class>>
SearchStreamGang
<<Java Class>>
SearchWithSequentialStream
<<Java Class>>
SearchWithParallelStreams
<<Java Class>>
SearchWithSequentialLoops
<<Java Class>>
SearchWithParallelSpliterator
<<Java Class>>
SearchWithCompletableFuturesInputs
<<Java Class>>
SearchWithParallelStreamInputs
<<Java Class>>
SearchWithCompletableFuturesPhrases
<<Java Class>>
SearchWithParallelStreamPhrases

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings

Parallel streams can be used to:

- *search chunks of phrases concurrently*
- *search chunks of input concurrently*
- *search chunks of each input string concurrently*



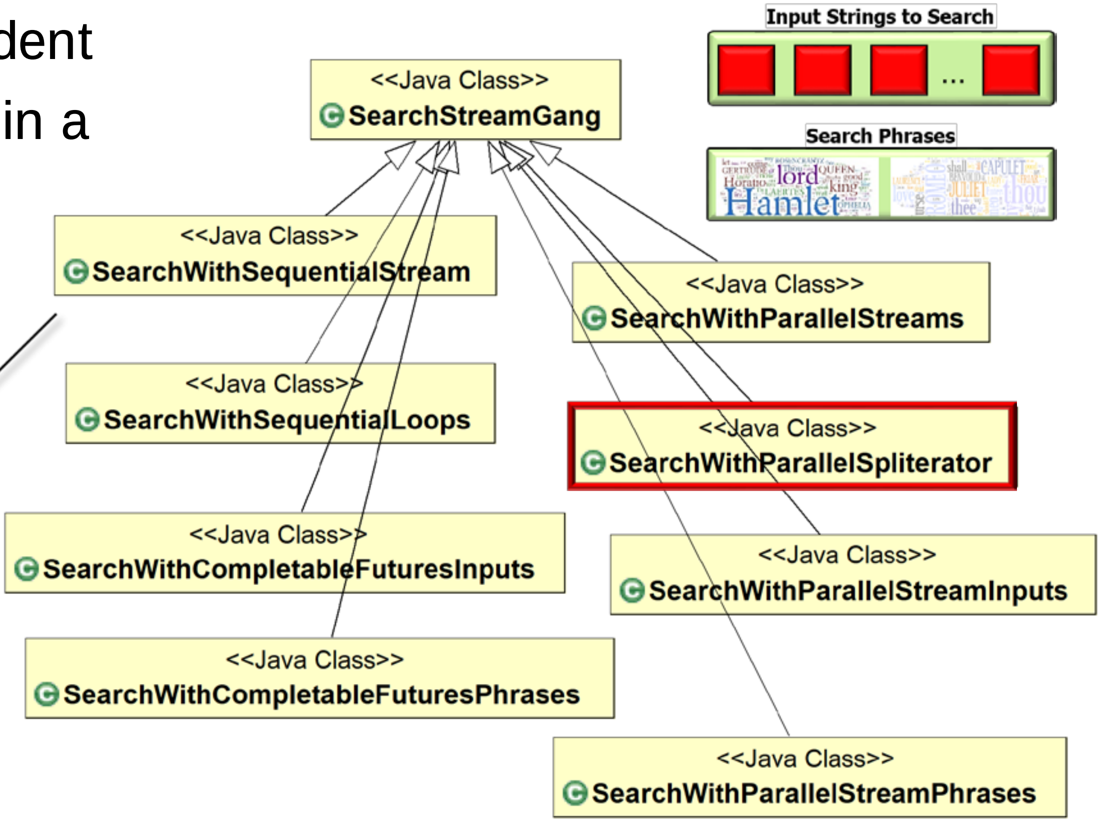
Input Strings to Search
...
Search Phrases
<<Java Class>>
SearchStreamGang
<<Java Class>>
SearchWithSequentialStream
<<Java Class>>
SearchWithSequentialLoops
<<Java Class>>
SearchWithCompletableFuturesInputs
<<Java Class>>
SearchWithCompletableFuturesPhrases
<<Java Class>>
SearchWithParallelStreams
<<Java Class>>
SearchWithParallelSpliterator
<<Java Class>>
SearchWithParallelStreamInputs
<<Java Class>>
SearchWithParallelStreamPhrases

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings

Parallel streams can be used to:

- *search chunks of phrases concurrently*
- *search chunks of input concurrently*
- *search chunks of each input string concurrently*

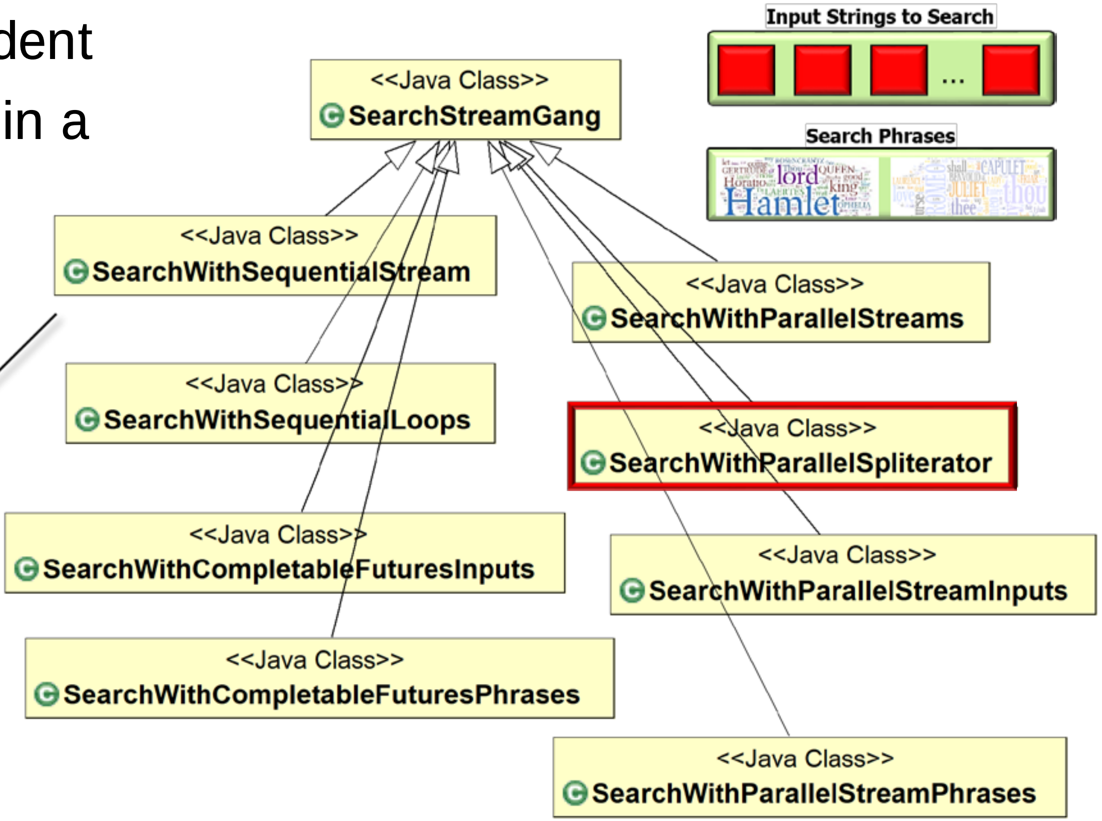


Input Strings to Search
...
Search Phrases
<<Java Class>>
SearchStreamGang
<<Java Class>>
SearchWithSequentialStream
<<Java Class>>
SearchWithParallelStreams
<<Java Class>>
SearchWithSequentialLoops
<<Java Class>>
SearchWithParallelSpliterator
<<Java Class>>
SearchWithCompletableFuturesInputs
<<Java Class>>
SearchWithParallelStreamInputs
<<Java Class>>
SearchWithCompletableFuturesPhrases
<<Java Class>>
SearchWithParallelStreamPhrases

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
    - e.g., searching for phrases in a list of input strings

Parallel streams can be used to:
- *search chunks of phrases concurrently*
- *search chunks of input concurrently*
- *search chunks of each input string concurrently*


Input Strings to Search
...
Search Phrases
<<Java Class>>
SearchStreamGang
<<Java Class>>
SearchWithSequentialStream
<<Java Class>>
SearchWithParallelStreams
<<Java Class>>
SearchWithSequentialLoops
<<Java Class>>
SearchWithParallelSpliterator
<<Java Class>>
SearchWithCompletableFuturesInputs
<<Java Class>>
SearchWithParallelStreamInputs
<<Java Class>>
SearchWithCompletableFuturesPhrases
<<Java Class>>
SearchWithParallelStreamPhrases

SearchWithParallelSpliterator is the most aggressively concurrent strategy!

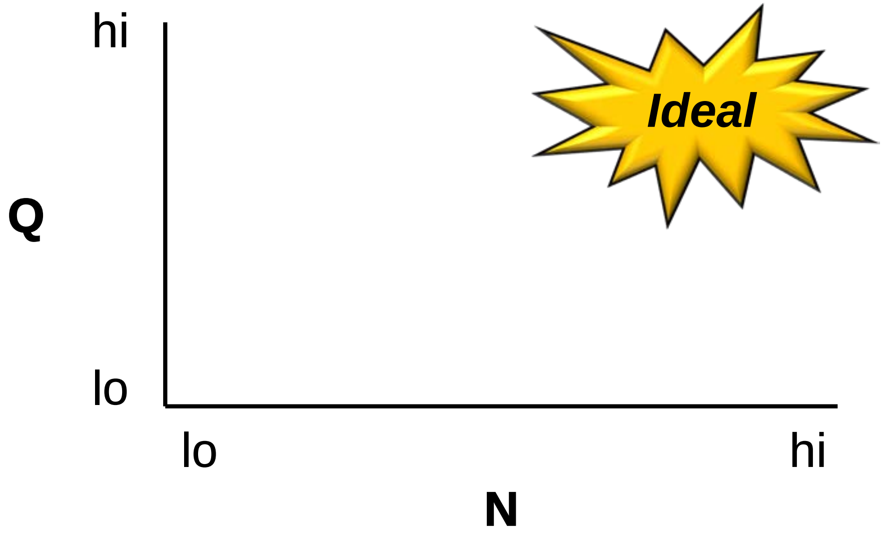
When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
  - When behaviors are expensive computationally and/or applied to many elements of efficiently splittable data structures

See www.ibm.com/developerworks/library/j-java-streams-5-brian-goetz

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
  - When behaviors are expensive computationally and/or applied to many elements of efficiently splittable data structures
    - i.e., the "NQ" model

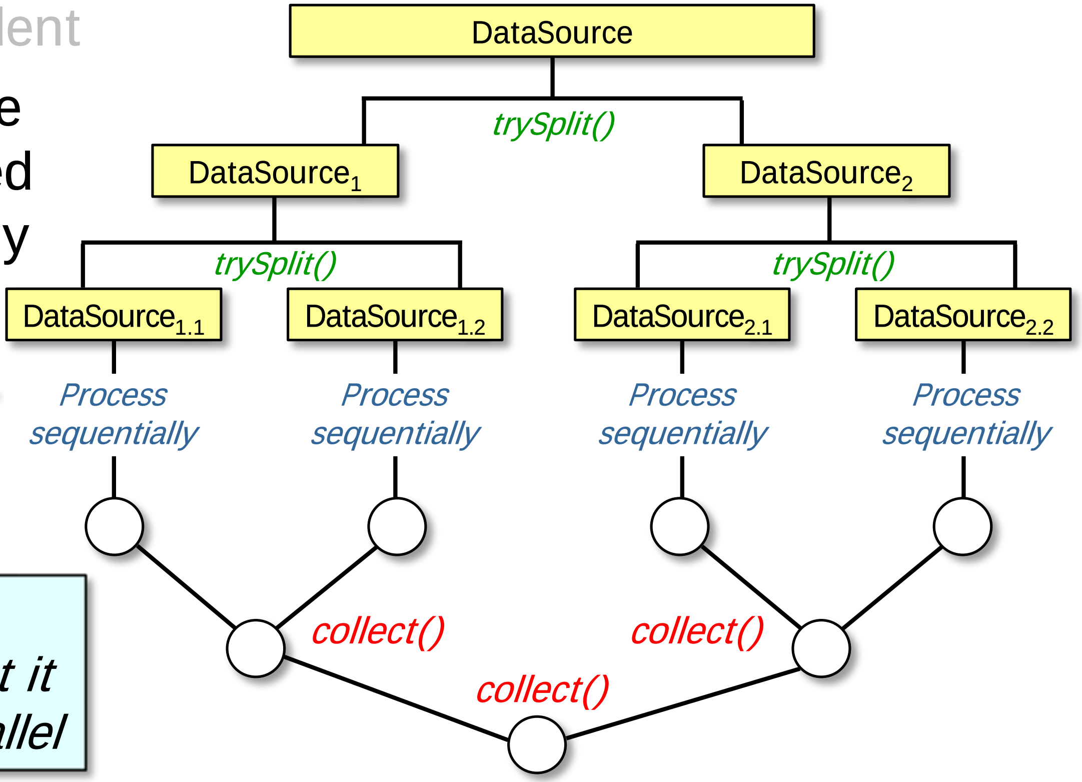


- *N is the # of data items to process per thread*
- *Q quantifies how CPU-intensive the processing is*

See on-sw-integration.epischel.de/2016/08/05/parallel-stream-processing-with-java-8-stream-api

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
  - When behaviors are expensive computationally and/or applied to many elements of efficiently splittable data structures
    - i.e., the "NQ" model

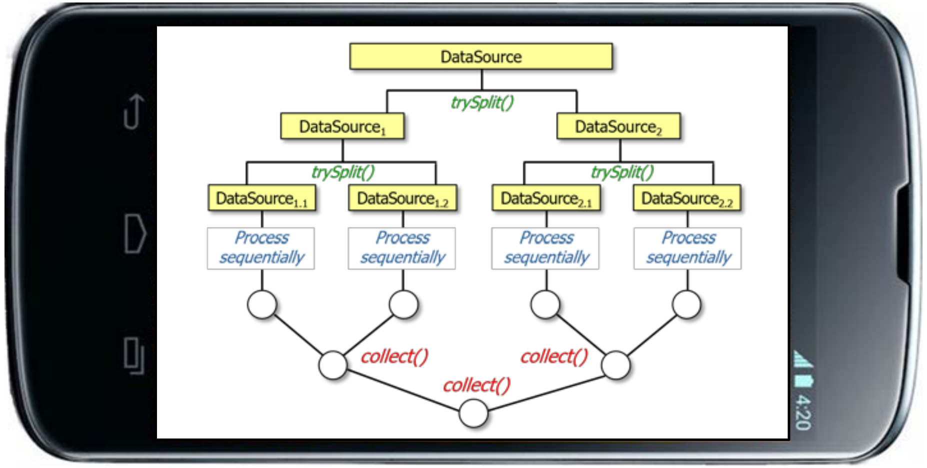


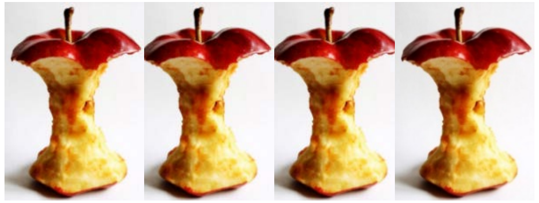
e.g., PhraseMatchSpliterator splits input strings into chunks that it searches for regex matches in parallel

See [SearchStreamGang/src/main/java/livelessons/utils/PhraseMatchSpliterator.java]()

When to Use Java 8 Parallel Streams

- Java 8 parallel streams are useful in some (but by no mean all) conditions
  - When behaviors are independent
  - When behaviors are expensive computationally and/or applied to many elements of efficiently splittable data structures
  - If there are multiple cores

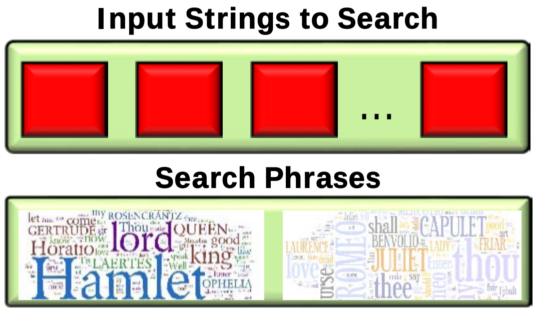


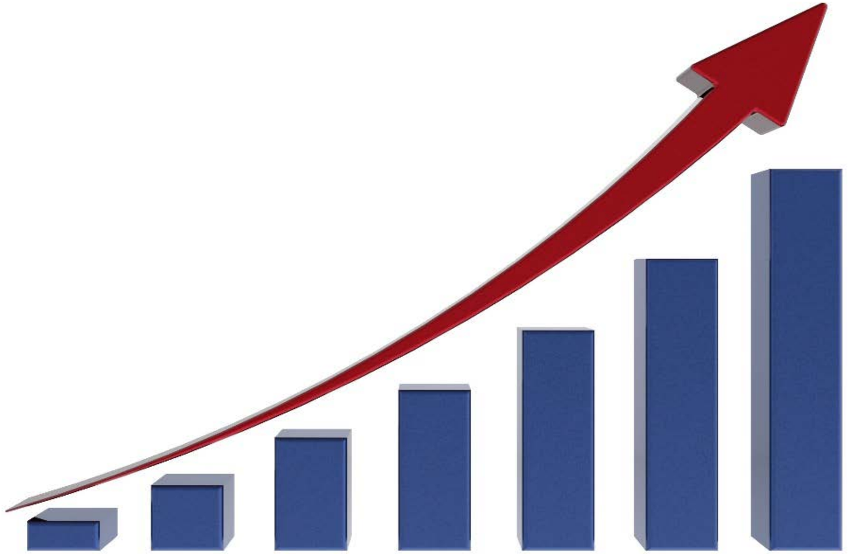


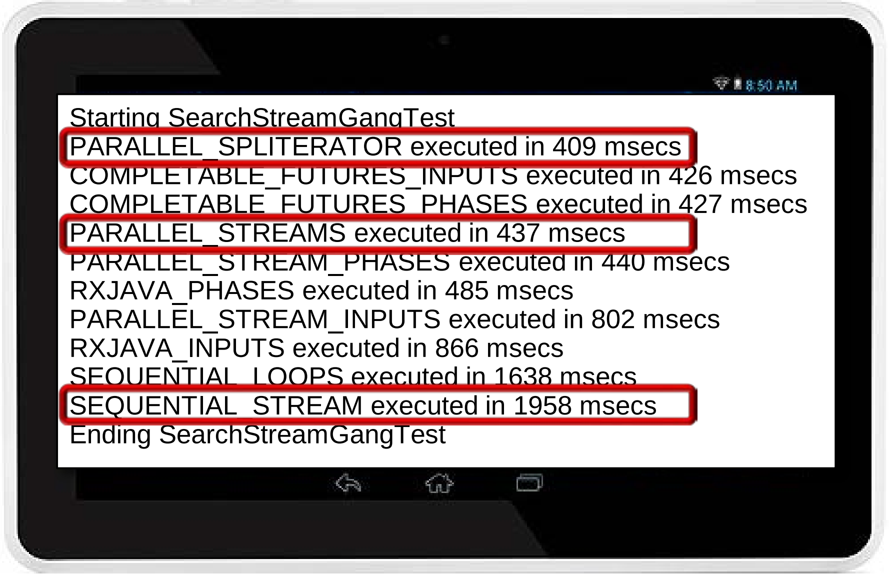
See blog.oio.de/2016/01/22/parallel-stream-processing-in-java-8-performance-of-sequential-vs-parallel-stream-processing

When to Use Java 8 Parallel Streams

- If the right conditions apply then Java 8 parallel streams can scale up nicely on multi-core/many-core processors







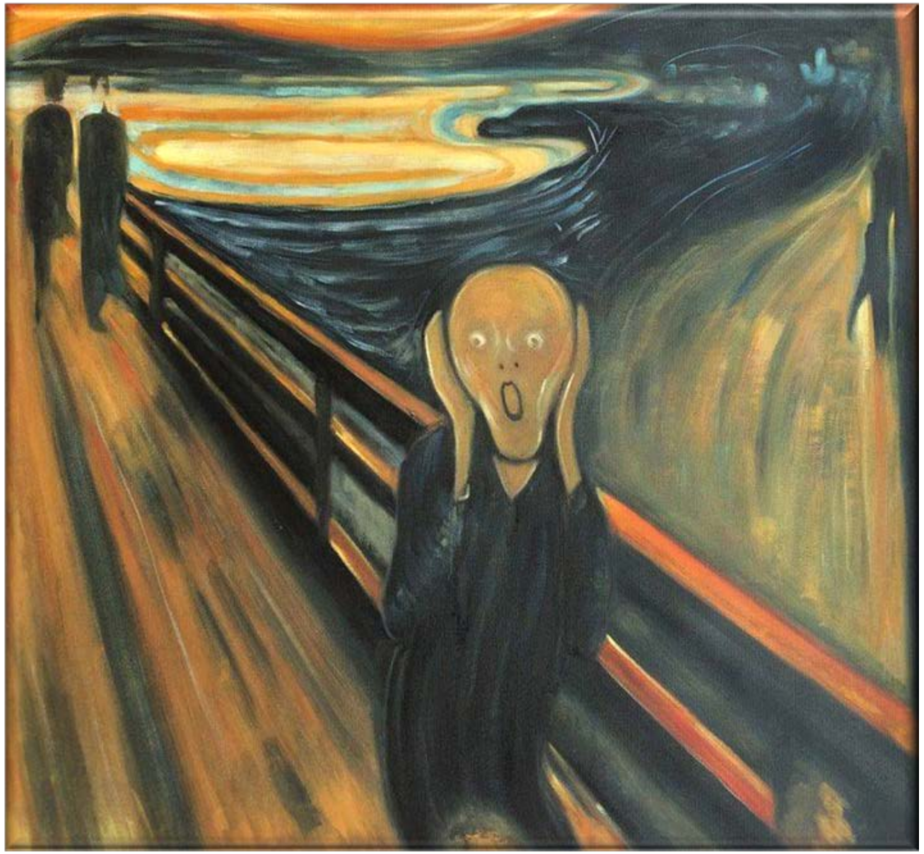
```
Starting SearchStreamGangTest
PARALLEL_SPLITERATOR executed in 409 msecs
COMPLETABLE_FUTURES_INPUTS executed in 426 msecs
COMPLETABLE_FUTURES_PHASES executed in 427 msecs
PARALLEL_STREAMS executed in 437 msecs
PARALLEL_STREAM_PHASES executed in 440 msecs
RXJAVA_PHASES executed in 485 msecs
PARALLEL_STREAM_INPUTS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

See www.infoq.com/presentations/parallel-java-se-8

When Not to Use Java 8 Parallel Streams

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs



See www.ibm.com/developerworks/library/j-java-streams-5-brian-goetz

When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly

```
List<CharSequence> arrayAllWords =
   TestDataFactory.getInput
     (sSHAKESPEARE_WORKS, "\\s+");

List<CharSequence> listAllWords =
   new LinkedList<>(arrayAllWords);


arrayAllWords.parallelStream()
              .count();

listAllWords.parallelStream()
             .count();
```

Make a LinkedList that contains all words in the works of Shakespeare

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly

```
List<CharSequence> arrayAllWords =
  TestDataFactory.getInput
    (sSHAKESPEARE_WORKS, "\\s+");

List<CharSequence> listAllWords =
  new LinkedList<>(arrayAllWords);
```

The ArrayList parallel stream is much faster than the LinkedList parallel stream

```
arrayAllWords.parallelStream()
                          .count();

listAllWords.parallelStream()
                         .count();
```

LinkedList splits poorly since finding the midpoint requires traversing ½ the list

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly

```
class ArraySpliterator {
  public Spliterator<T> trySplit(){
    int lo = index, mid =
       (lo + fence) >>> 1;
      return (lo >= mid)
        ? null
        : new ArraySpliterator<>
          (array,
           lo, index = mid,
           characteristics);
}
```

The ArrayList spliterator runs in O(1) constant time

See grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8-b132/java/util/Spliterators.java

When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly

```
class LLSpliterator {
  public Spliterator<E> trySplit(){
    ...
    int n = batch + BATCH_UNIT;
    ...
    Object[] a = new Object[n];
    int j = 0;
    do { a[j++] = p.item; }
    while ((p = p.next) != null
           && j < n);
    ...
    return Spliterators
      .spliterator(a, 0, j,
                   Spliterator.ORDERED);
```

The LinkedList spliterator runs in O(n) linear time

See grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8-b132/java/util/LinkedList.java

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data

```
class ParallelStreamFactorial {
  BigInteger factorial(long n) {
    return LongStream
      .rangeClosed(1, n)
      .parallel() ...
      .reduce(BigInteger.ONE,
              BigInteger::multiply);
  ...

class SequentialStreamFactorial {
  BigInteger factorial(long n) {
    return LongStream
      .rangeClosed(1, n) ...
      .reduce(BigInteger.ONE,
              BigInteger::multiply);
  ...
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data

```
class ParallelStreamFactorial {
  BigInteger factorial(long n) {
    return LongStream
      .rangeClosed(1, n)
      .parallel() ...
      .reduce(BigInteger.ONE,
              BigInteger::multiply);
  ...
```

The overhead of creating a parallel stream is > than the benefits of parallelism for small values of 'n'

```
class SequentialStreamFactorial {
  BigInteger factorial(long n) {
    return LongStream
      .rangeClosed(1, n) ...
      .reduce(BigInteger.ONE,
              BigInteger::multiply);
  ...
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

When Not to Use Java 8 Parallel Streams

- Parallel streams aren’t suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

```
List<CharSequence> allWords =
  new LinkedList<>
    (TestDataFactory.getInput
       (sSHAKESPEARE_DATA_FILE,
        "\\s+"));
...

Set<CharSequence> uniqueWords =
  allWords
    .parallelStream()
    ...
    .collect(toCollection
                (TreeSet::new));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

```
List<CharSequence> allWords =
  new LinkedList<>
    (TestDataFactory.getInput
       (sSHAKESPEARE_DATA_FILE,
        "\\s+"));
...

Set<CharSequence> uniqueWords =
  allWords
    .parallelStream()
    ...
    .collect(toCollection
                (TreeSet::new));
```

Performance will be poor due to the overhead of combining partial results for a Set in a parallel stream

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly

```
List<CharSequence> allWords =
  new LinkedList<>
    (TestDataFactory.getInput
       (sSHAKESPEARE_DATA_FILE,
        "\\s+"));
...

Set<CharSequence> uniqueWords =
  allWords
    .parallelStream()
    ...
    .collect(toCollection
               (TreeSet::new));
```

The combining cost can be alleviated by the amount of work performed per element (i.e., the "NQ model")

See www.ibm.com/developerworks/library/j-java-streams-5-brian-goetz

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

```
List<Double> result = Stream
   .iterate(2, i -> i + 1)
   .parallel()
   .filter(this::isEven)
   .limit(number)
   .map(this::findSQRT)
   .collect(toList());

List<Double> result = LongStream
   .range(2, (number * 2) + 1)
   .parallel()
   .filter(this::isEven)
   .mapToObj(this::findSQRT)
   .collect(toList());
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

```
List<Double> result = Stream
    .iterate(2, i -> i + 1)
    .parallel()
    .filter(this::isEven)
    .limit(number)
    .map(this::findSQRT)
    .collect(toList());

List<Double> result = LongStream
    .range(2, (number * 2) + 1)
    .parallel()
    .filter(this::isEven)
    .mapToObj(this::findSQRT)
    .collect(toList());
```

Stream.iterate() & limit() split & parallelize poorly...

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism

```
List<Double> result = Stream
    .iterate(2, i -> i + 1)
    .parallel()
    .filter(this::isEven)
    .limit(number)
    .map(this::findSQRT)
    .collect(toList());

List<Double> result = LongStream
    .range(2, (number * 2) + 1)
    .parallel()
    .filter(this::isEven)
    .mapToObj(this::findSQRT)
    .collect(toList());
```

LongStream.range() splits nicely & thus runs efficiently in parallel

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex15

When Not to Use Java 8 Parallel Streams

- Parallel streams aren't suitable for certain types of programs, e.g.
  - The source is expensive to split or splits unevenly
  - The startup costs of parallelism overwhelm the amount of data
  - Combining partial results is costly
  - A Java 8 feature doesn't enable sufficient exploitable parallelism
  - There aren't many/any cores

Older computing devices just have a single core, which limits available parallelism

End of Overview of
Java 8 Parallel Streams
(Part 3)