

# **Java Parallel Streams Internals: Demo'ing Collector Performance**

**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 via the common fork-join pool
  - Configure the Java parallel stream common fork-join pool
  - Perform a reduction to combine partial results into a single result
  - Recognize key behaviors & differences of non-concurrent & concurrent collectors
  - Be aware of non-concurrent & concurrent collector APIs
  - Grok performance variance in concurrent & non-concurrent collectors

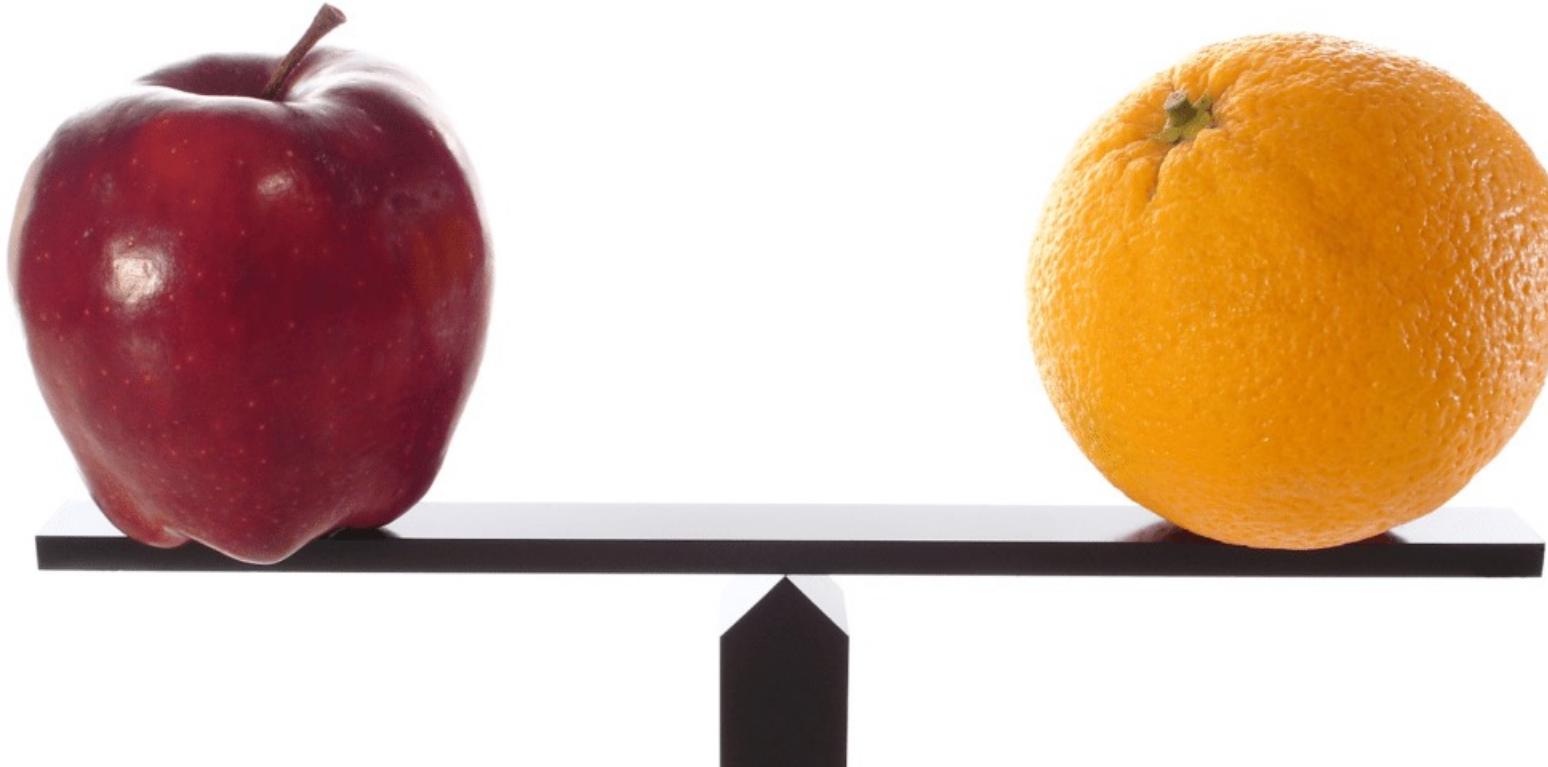
```
Starting collector tests for 1000 words..printing results
21 msec: sequential timeStreamCollectToSet()
30 msec: parallel timeStreamCollectToSet()
39 msec: sequential timeStreamCollectToConcurrentSet()
59 msec: parallel timeStreamCollectToConcurrentSet()
...
Starting collector tests for 100000 words..printing results
219 msec: parallel timeStreamCollectToConcurrentSet()
364 msec: parallel timeStreamCollectToSet()
657 msec: sequential timeStreamCollectToSet()
804 msec: sequential timeStreamCollectToConcurrentSet()
Starting collector tests for 883311 words..printing results
1782 msec: parallel timeStreamCollectToConcurrentSet()
3010 msec: parallel timeStreamCollectToSet()
6169 msec: sequential timeStreamCollectToSet()
7652 msec: sequential timeStreamCollectToConcurrentSet()
```

---

# Demonstrating Collector Performance

# Demonstrating Collector Performance

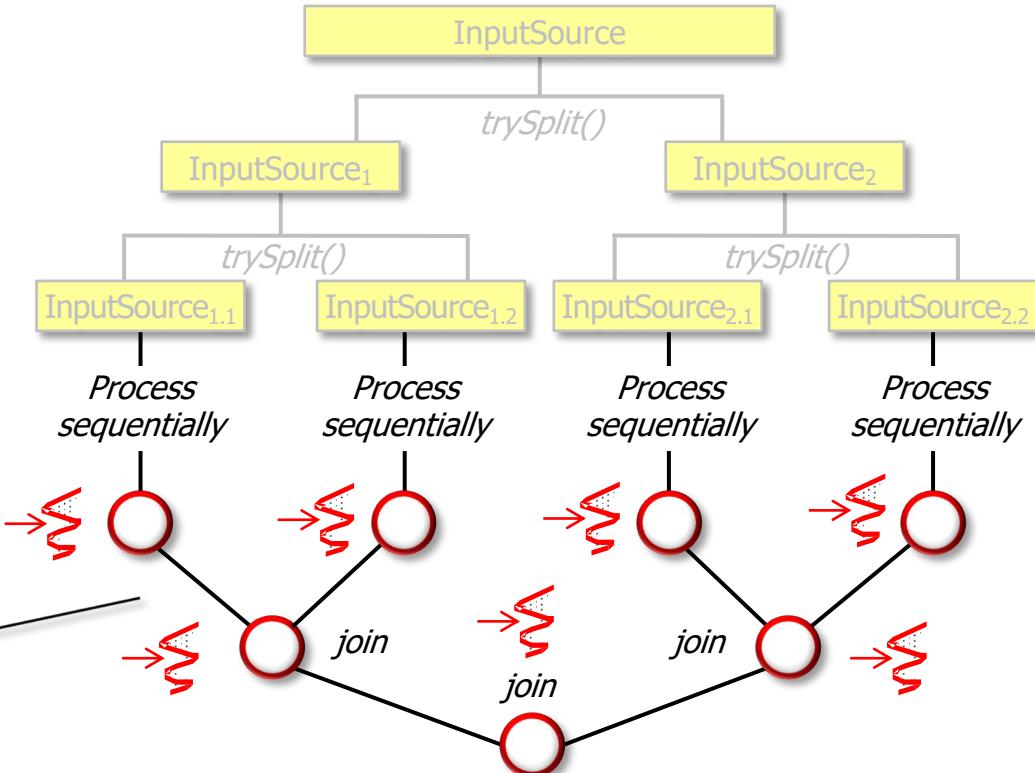
- Concurrent & non-concurrent collectors perform differently when used in parallel & sequential streams on different input sizes



See prior lessons on “Java Parallel Streams Internals: Non-Concurrent and Concurrent Collectors”

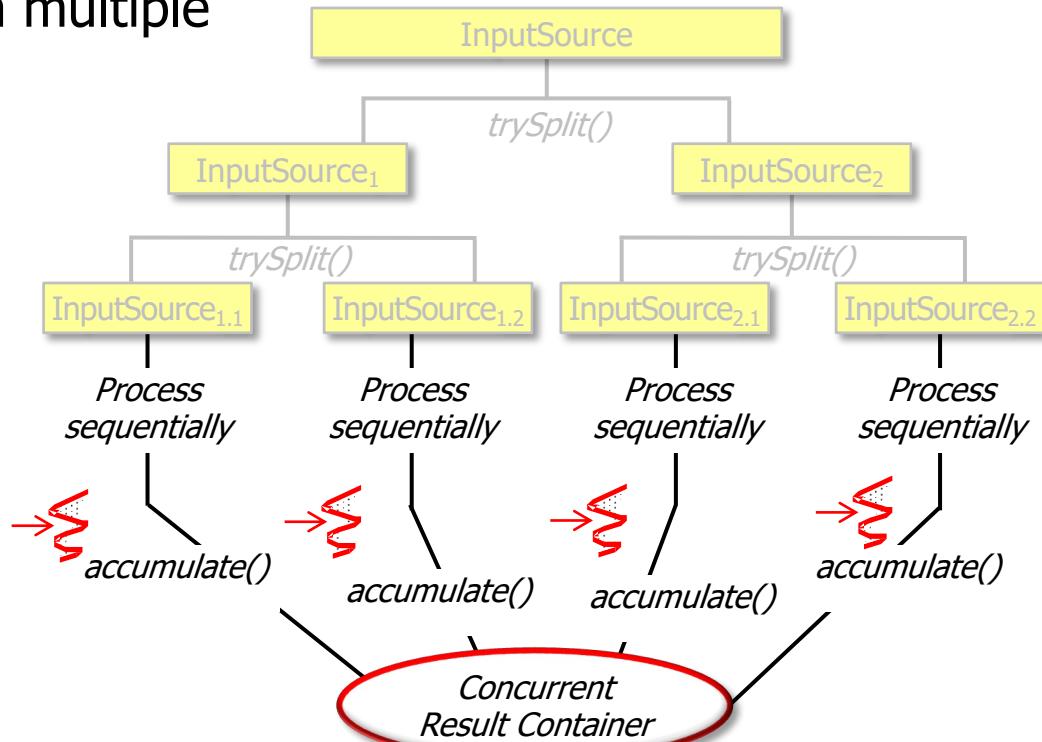
# Demonstrating Collector Performance

- A non-concurrent collector operates by merging sub-results



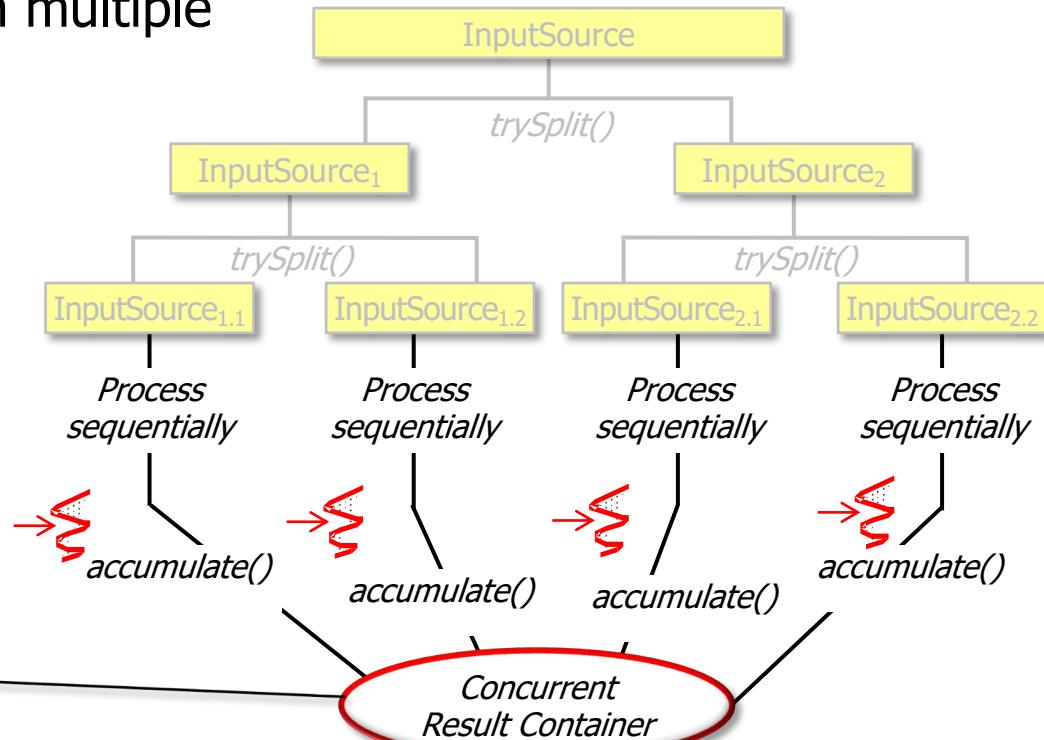
# Demonstrating Collector Performance

- A concurrent collector creates one concurrent mutable result container & accumulates elements into it from multiple threads in a parallel stream



# Demonstrating Collector Performance

- A concurrent collector creates one concurrent mutable result container & accumulates elements into it from multiple threads in a parallel stream



# Demonstrating Collector Performance

---

- The ex36 example showcases the different in performance of two collectors



# Demonstrating Collector Performance

- The ex36 example showcases the different in performance of two collectors
  - Various Set collectors defined by the Java Collectors utility class

<<Java Class>>	
.Collectors	
	Collectors()
	toCollection(Supplier<C>):Collector<T,?,C>
	toList():Collector<T,?,List<T>>
	toSet():Collector<T,?,Set<T>>
	toMap(Function<? super T,? extends K>,Function<? super T,? extends U>):Collector<T,?,Map<K,U>>

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html)

# Demonstrating Collector Performance

- The ex36 example showcases the different in performance of two collectors
  - Various Set collectors defined by the Java Collectors utility class
  - The ConcurrentSetCollector

 	ConcurrentSetCollector<T, E, S>
 	ConcurrentSetCollector(Function<T, E>, Supplier<S>)
 	supplier() Supplier<Set<E>>
 	toSet(Function<T, E>, Supplier<S>) Collector<T, ?, S>
 	finisher() Function<Set<E>, S>
 	accumulator() BiConsumer<Set<E>, T>
 	combiner() BinaryOperator<Set<E>>
 	characteristics() Set<Characteristics>

See <Java8/ex36/src/main/java/utils/ConcurrentSetCollector.java>

# Demonstrating Collector Performance

- The ex36 example showcases the different in performance of two collectors
  - Various Set collectors defined by the Java Collectors utility class
  - The ConcurrentSetCollector
  - Applied in conjunction with ConcurrentHashMap.

KeySetView

## Class ConcurrentHashMap.KeySetView<K,V>

java.lang.Object  
java.util.concurrent.ConcurrentHashMap.KeySetView<K,V>

### All Implemented Interfaces:

Serializable, Iterable<K>, Collection<K>, Set<K>

### Enclosing class:

ConcurrentHashMap<K,V>

---

```
public static class ConcurrentHashMap.KeySetView<K,V>
extends Object
implements Set<K>, Serializable
```

A view of a ConcurrentHashMap as a Set of keys, in which additions may optionally be enabled by mapping to a common value. This class cannot be directly instantiated. See keySet(), keySet(V), newKeySet(), newKeySet(int).

# Demonstrating Collector Performance

---

- Results show collector differences become more significant as input grows

Starting collector tests for 1000 words..printing results

```
21 msecs: sequential timeStreamCollectToSet()  
30 msecs: parallel timeStreamCollectToSet()  
39 msecs: sequential timeStreamCollectToConcurrentSet()  
59 msecs: parallel timeStreamCollectToConcurrentSet()
```

...

Starting collector tests for 100000 words....printing results

```
219 msecs: parallel timeStreamCollectToConcurrentSet()  
364 msecs: parallel timeStreamCollectToSet()  
657 msecs: sequential timeStreamCollectToSet()  
804 msecs: sequential timeStreamCollectToConcurrentSet()
```

Starting collector tests for 883311 words....printing results

```
1782 msecs: parallel timeStreamCollectToConcurrentSet()  
3010 msecs: parallel timeStreamCollectToSet()  
6169 msecs: sequential timeStreamCollectToSet()  
7652 msecs: sequential timeStreamCollectToConcurrentSet()
```

---

See upcoming lessons on “*When [Not] to Use Parallel Streams*”

# Demonstrating Collector Performance

The screenshot shows an IDE interface with a Java file named `ex36.java` open. The code in the file is as follows:

```
30
31 /**
32 * Main entry point into the tests program.
33 */
34 static public void main(String[] argv) {
35     System.out.println("Entering the test program with "
36                     + Runtime.getRuntime().availableProcessors()
37                     + " cores available");
38
39 // Warm up the threads in the fork/join pool so the timing
40 // results will be more accurate.
41 warmUpForkJoinPool();
42
43 // Run tests that demonstrate the performance differences
44 // between concurrent and non-concurrent collectors when
45 // collecting results in Java sequential and parallel streams
46 // that use unordered HashSets.
47 runCollectorTestsUnordered();
48
49 // Run tests that demonstrate the performance differences
50 // between concurrent and non-concurrent collectors when
51 // collecting results in Java sequential and parallel streams
52 // that use ordered TreeSets.
53 runCollectorTestsOrdered();
54 }
```

The IDE's left sidebar shows a project structure with a file named `build` selected. The bottom navigation bar includes tabs for Git, TODO, Problems, Terminal, and Dependencies.

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex36](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex36)

---

# End of Java Parallel Streams Internals: Demo'ing Collector Performance