Understand Java Parallel Streams Internals: Demo'ing Collector Performance

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

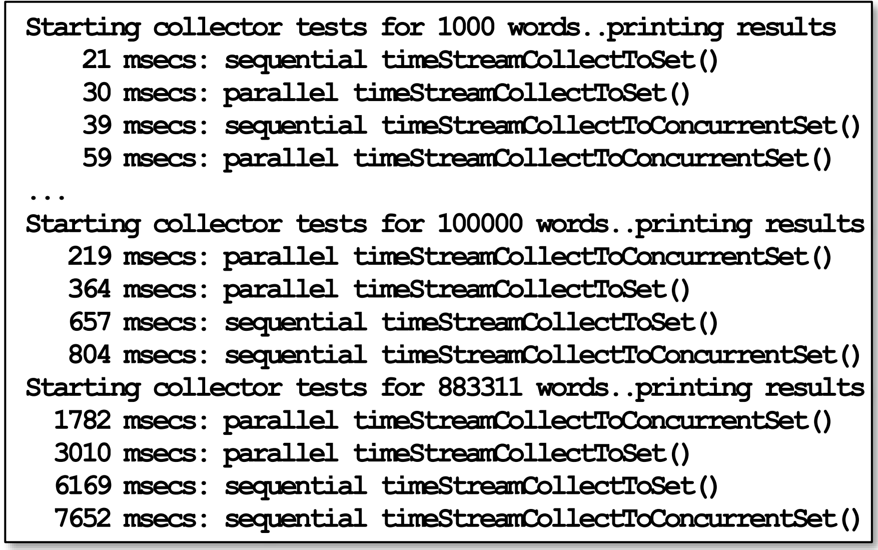
Professor of Computer Science

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
  - Learn how to implement non-concurrent & concurrent collectors
  - Be aware of performance variance in concurrent & non-concurrent collectors



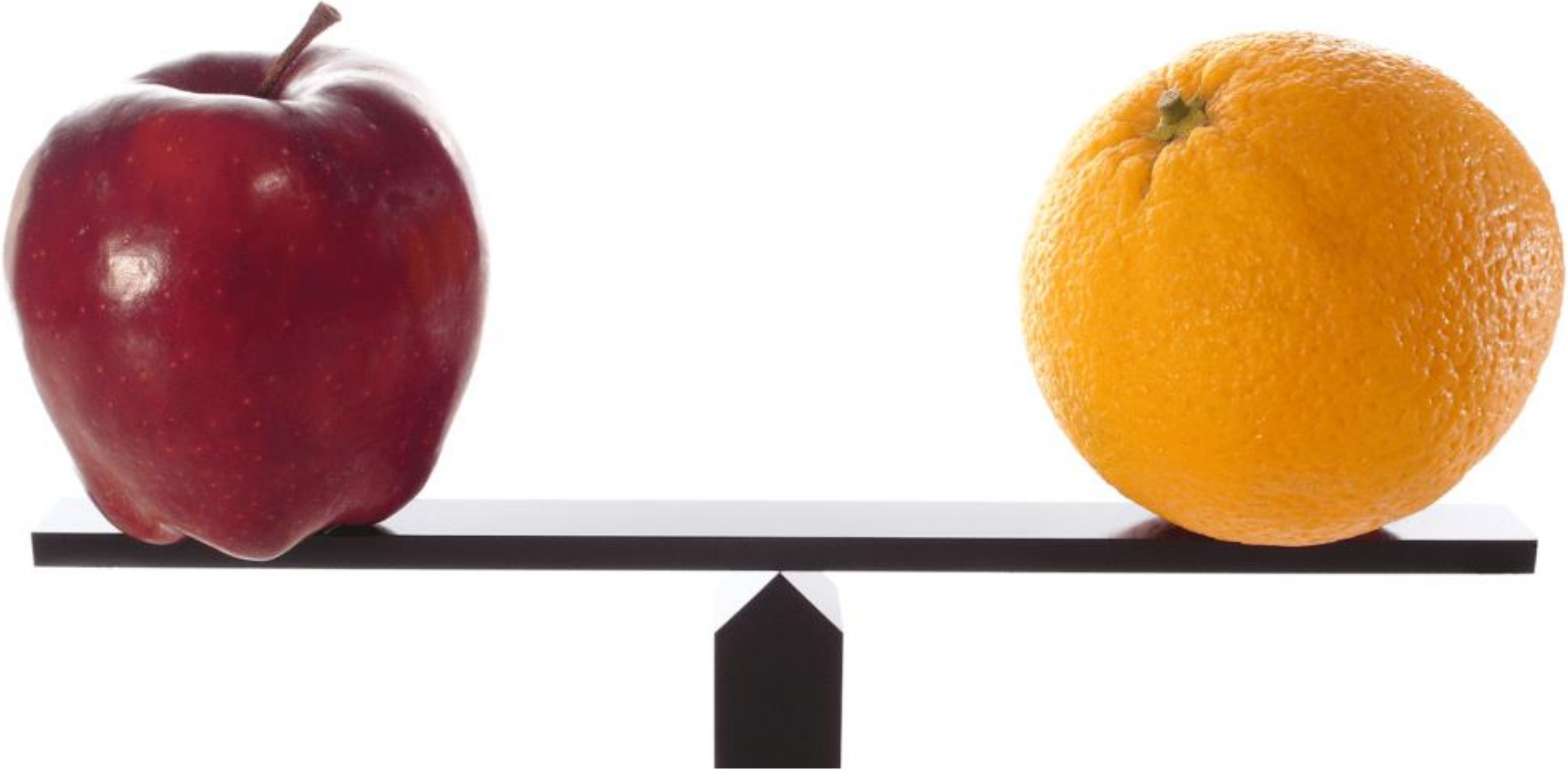
```
Starting collector tests for 1000 words..printing results
    21 msecs: sequential timeStreamCollectToSet()
    30 msecs: parallel timeStreamCollectToSet()
    39 msecs: sequential timeStreamCollectToConcurrentSet()
    59 msecs: parallel timeStreamCollectToConcurrentSet()
...
Starting collector 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 github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex14

Demonstrating Collector Performance

Demonstrating Collector Performance

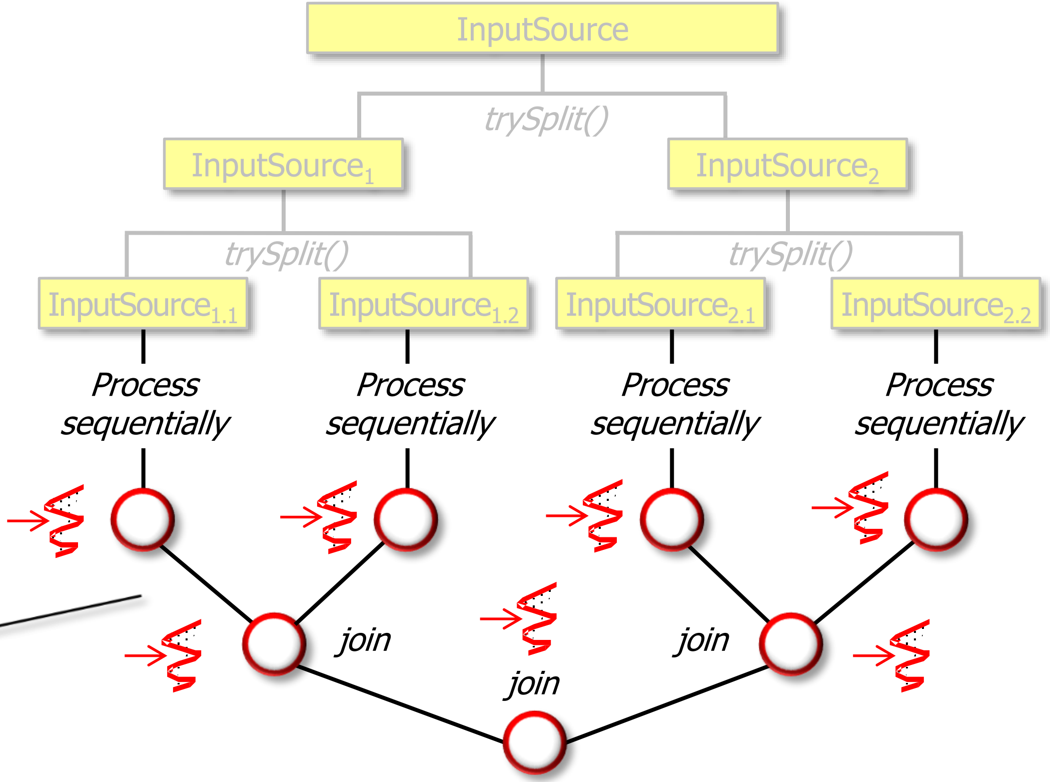
- Concurrent & non-concurrent collectors perform differently when used in parallel & sequential streams on different input sizes



See prior lessons on "*Java Parallel Streams Internals: Non-Concurrent and Concurrent Collectors*"

Demonstrating Collector Performance

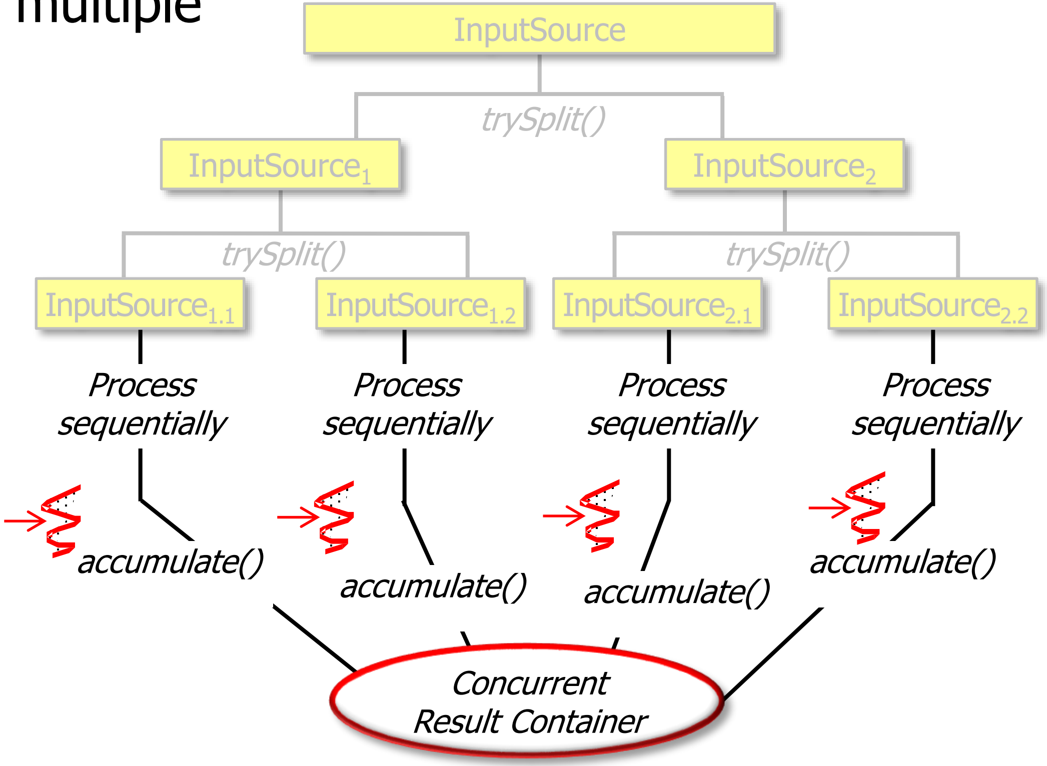
- A non-concurrent collector operates by merging sub-results



Different threads operate on different instances of the intermediate result containers

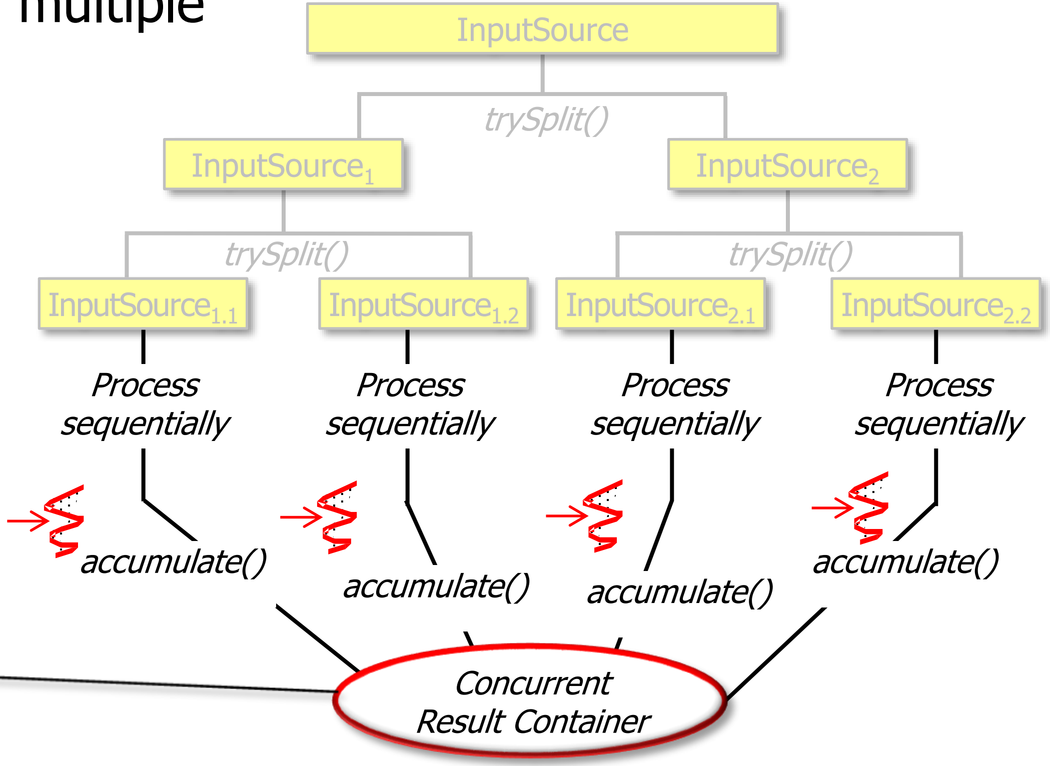
Demonstrating Collector Performance

- 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



Thus there's no need to merge any intermediate sub-results!

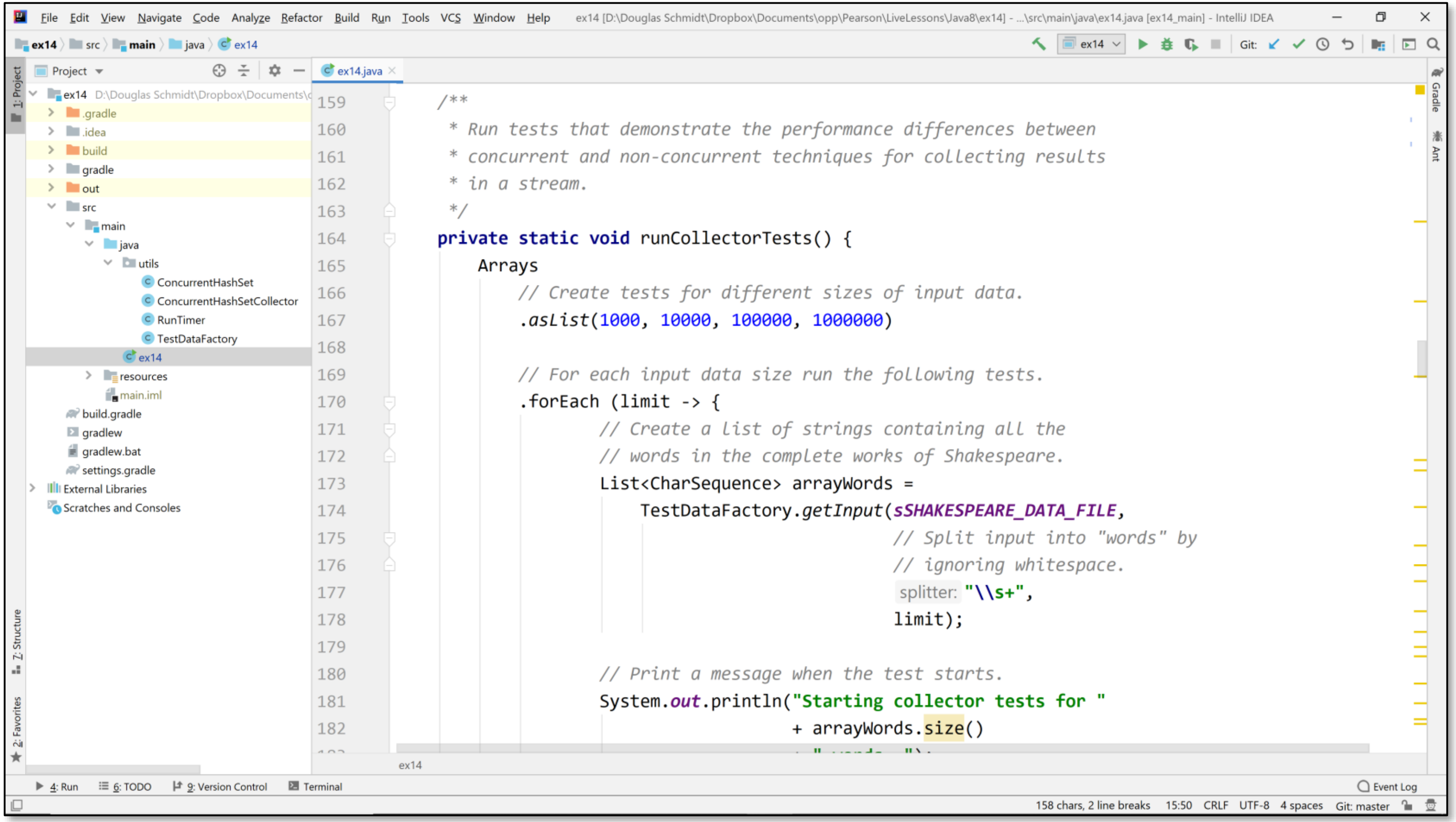
Demonstrating Collector Performance

- 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


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ex14 [D:\Douglas Schmidt\Dropbox\Documents\opp\Pearson\LiveLessons\Java8\ex14] - ...\src\main\java\ex14.java [ex14_main] - IntelliJ IDEA
ex14 › src › main › java › ex14
Project
ex14 D:\Douglas Schmidt\Dropbox\Documents\
.gradle
.idea
build
gradle
out
src
main
java
utils
ConcurrentHashSet
ConcurrentHashSetCollector
RunTimer
TestDataFactory
ex14
resources
main.iml
build.gradle
gradlew
gradlew.bat
settings.gradle
External Libraries
Scratches and Consoles
ex14.java
Git:
4: Run
6: TODO
9: Version Control
Terminal
Event Log
158 chars, 2 line breaks 15:50 CRLF UTF-8 4 spaces Git: master

```java
/**
 * Run tests that demonstrate the performance differences between
 * concurrent and non-concurrent techniques for collecting results
 * in a stream.
 */
private static void runCollectorTests() {
    Arrays
        // Create tests for different sizes of input data.
        .asList(1000, 10000, 100000, 1000000)

        // For each input data size run the following tests.
        .forEach (limit -> {
                // Create a list of strings containing all the
                // words in the complete works of Shakespeare.
                List<CharSequence> arrayWords =
                    TestDataFactory.getInput(sSHAKESPEARE_DATA_FILE,
                                             // Split input into "words" by
                                             // ignoring whitespace.
                                             splitter: "\\s+",
                                             limit);

                // Print a message when the test starts.
                System.out.println("Starting collector tests for "
                                   + arrayWords.size()
```

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

End of Understand Java Parallel Streams Internals: Demo’ing Collector Performance