

Understand Java Parallel Streams Internals: Combining Results (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

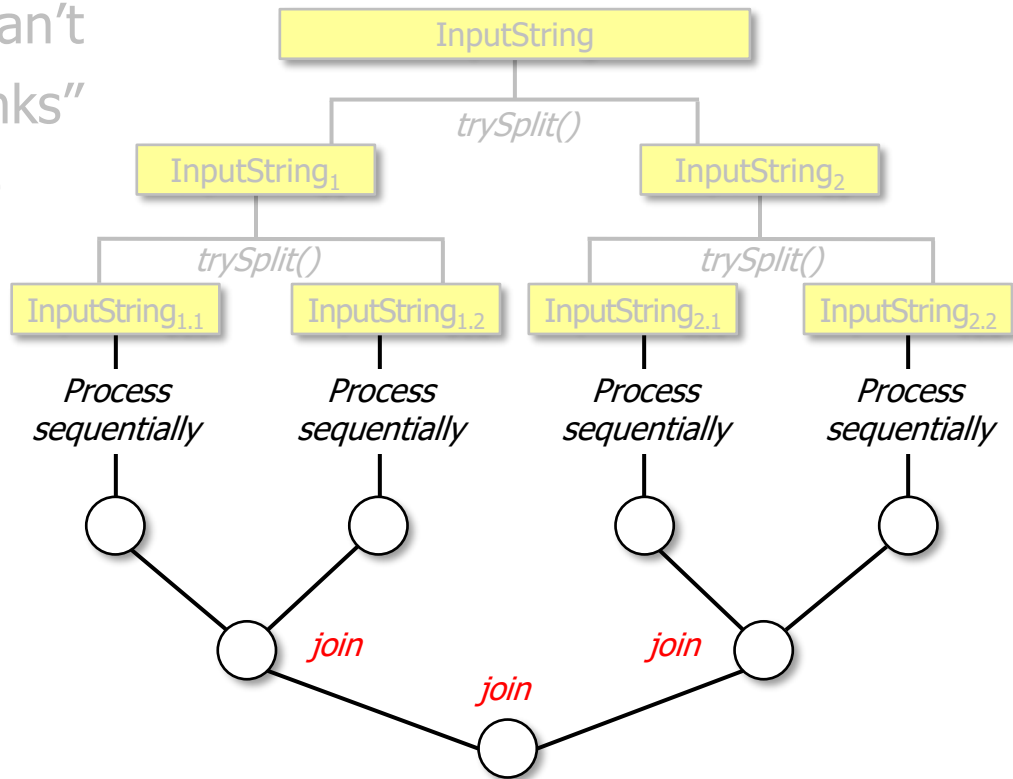
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

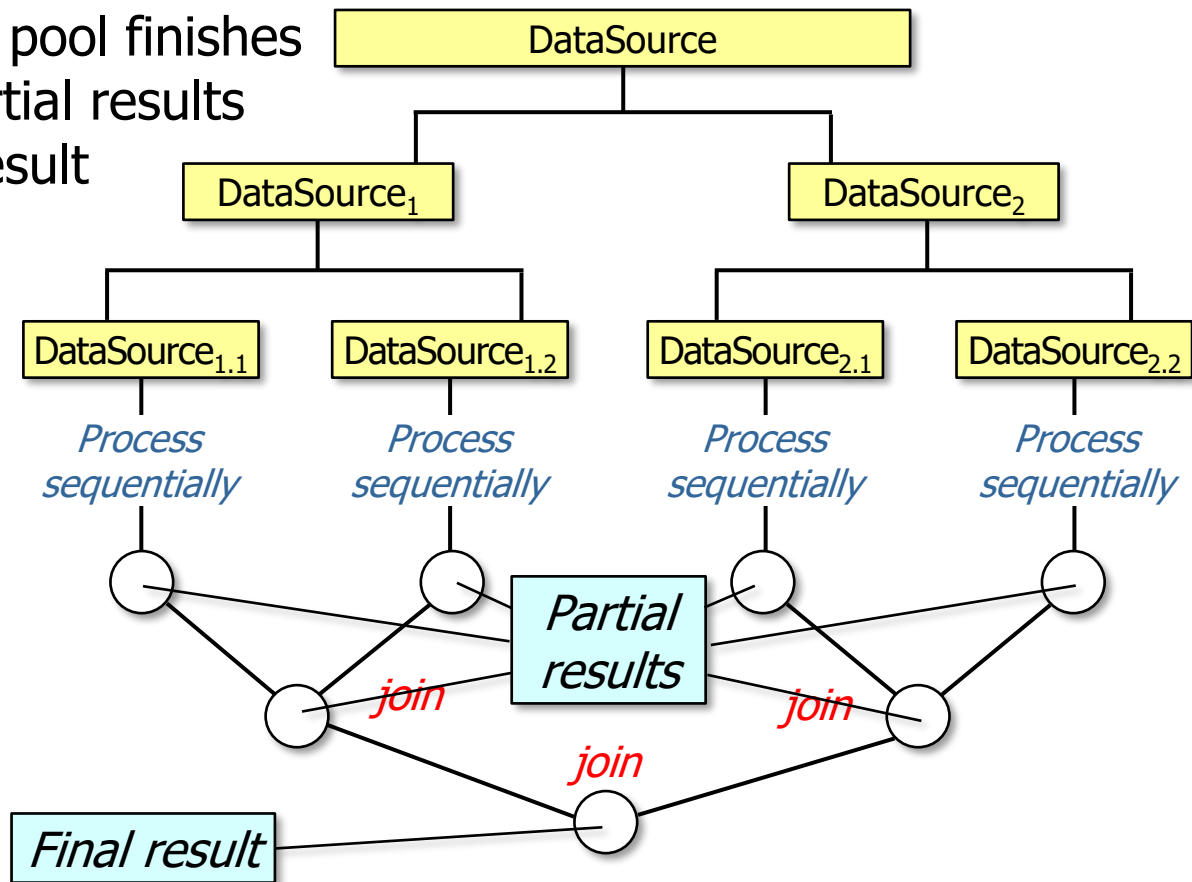
- Understand parallel stream internals, e.g.
 - Know what can change & what can't
 - Partition a data source into "chunks"
 - Process chunks in parallel via the common fork-join pool
 - Configure the Java parallel stream common fork-join pool
- Perform a reduction to combine partial results into a single result



Combining Results in a Parallel Stream

Combining Results in a Parallel Stream

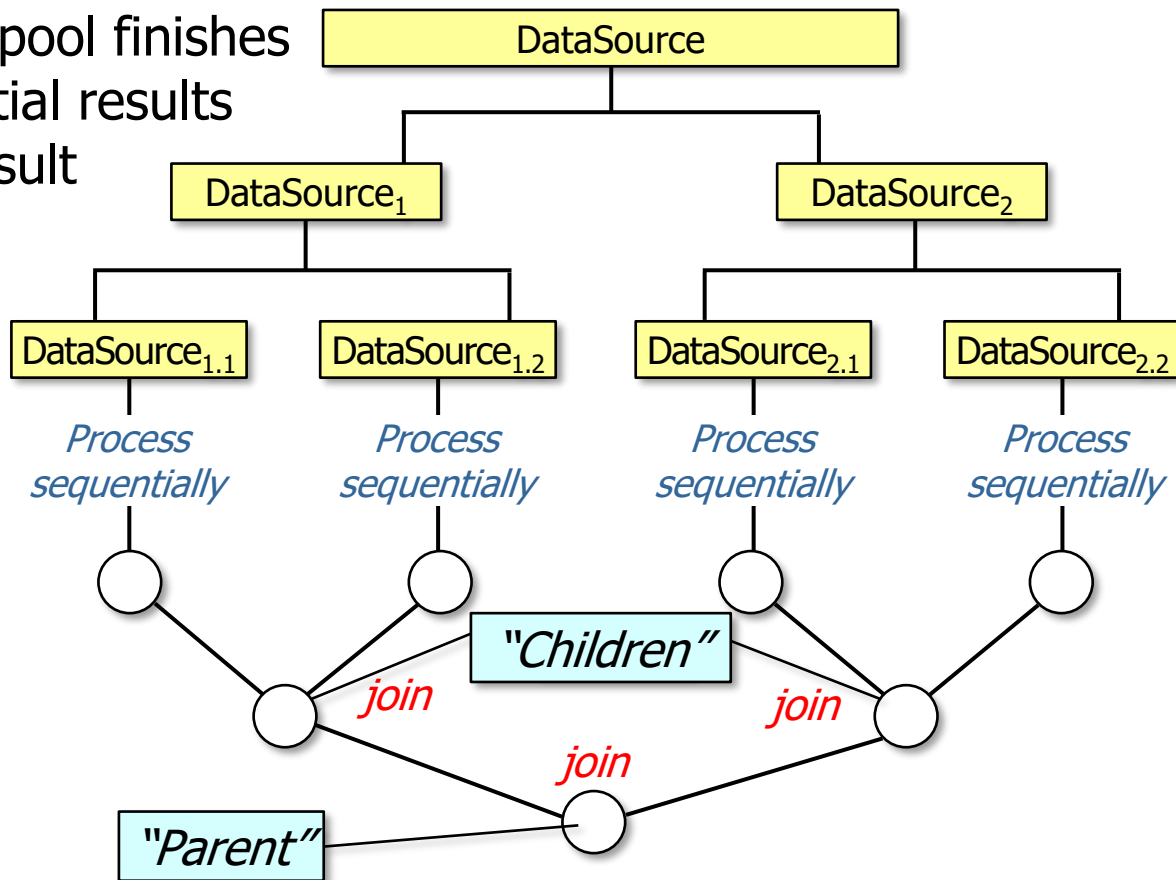
- After the common fork-join pool finishes processing chunks their partial results are combined into a final result



This discussion assumes a non-concurrent collector (other discussions follow)

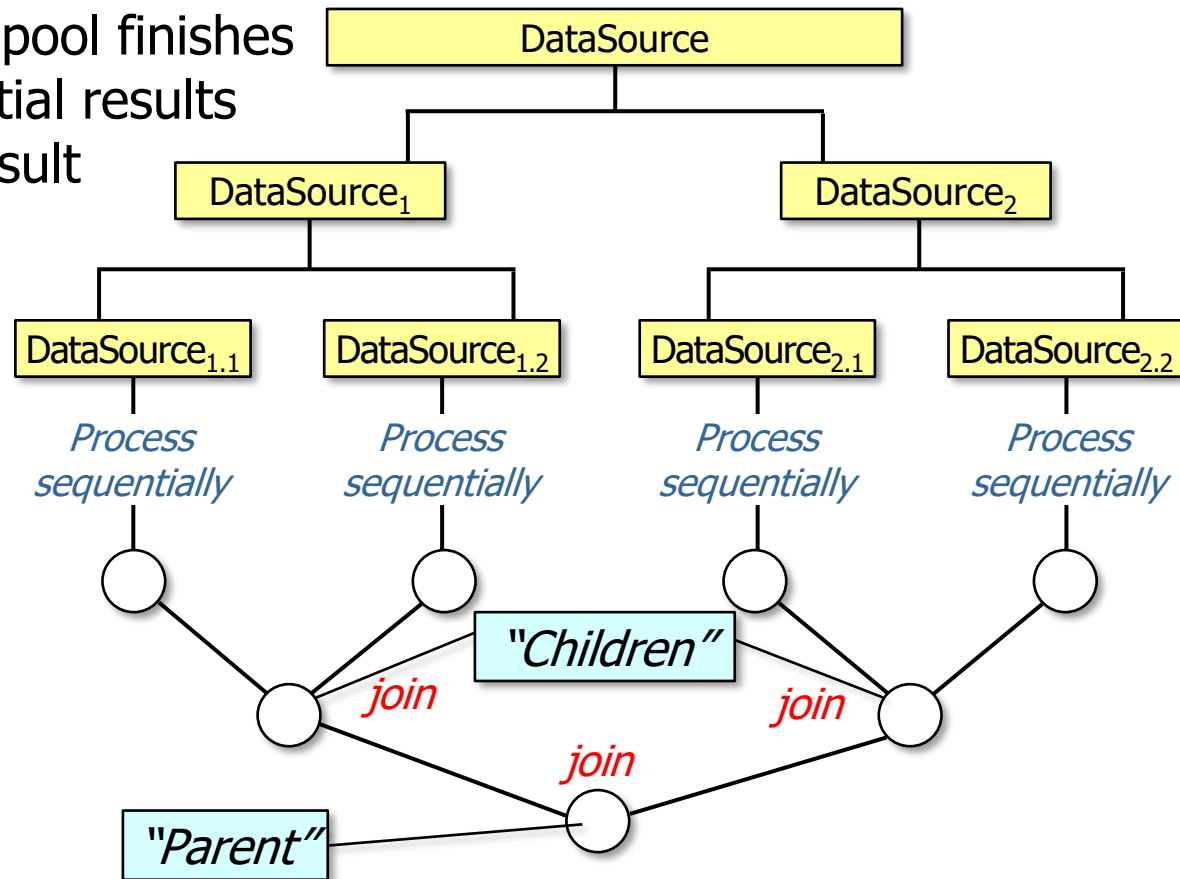
Combining Results in a Parallel Stream

- After the common fork-join pool finishes processing chunks their partial results are combined into a final result
 - `join()` occurs in a single thread at each level
 - i.e., the “parent”



Combining Results in a Parallel Stream

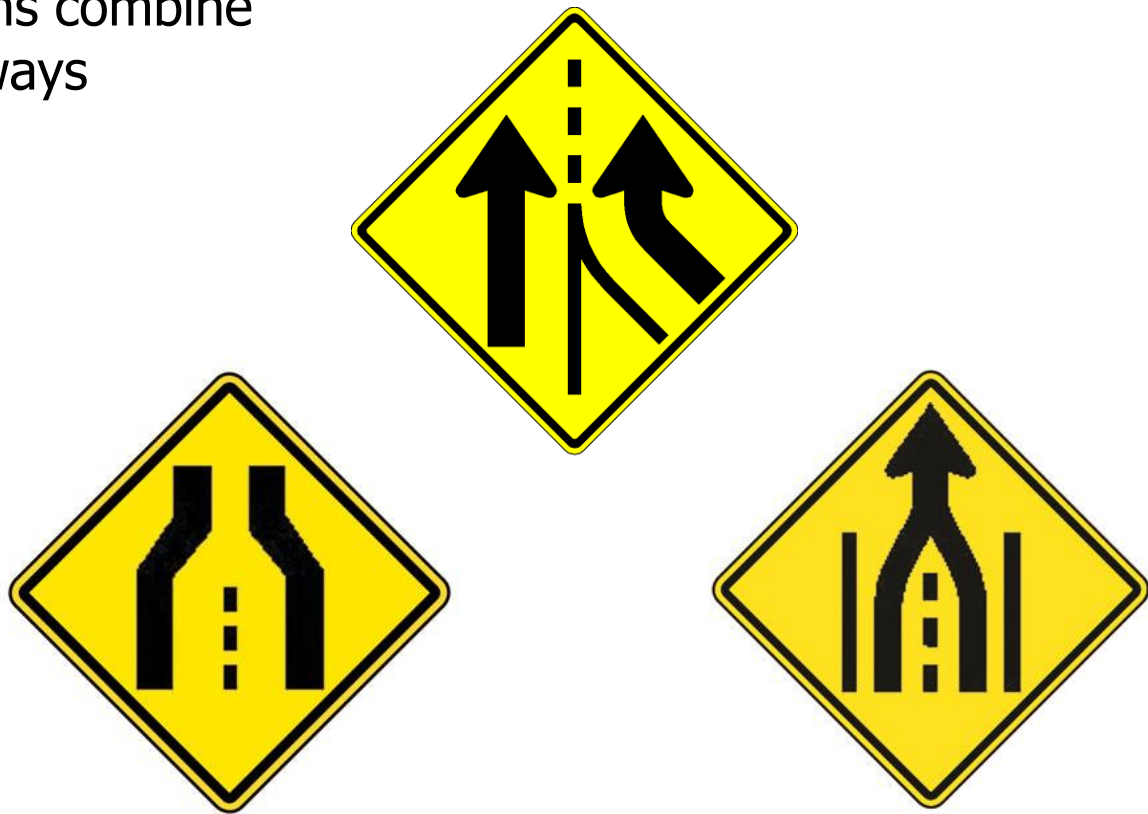
- After the common fork-join pool finishes processing chunks their partial results are combined into a final result
 - `join()` occurs in a single thread at each level
 - i.e., the “parent”



As a result, there's typically no need for synchronizers during the joining

Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways



Understanding these differences is particularly important for parallel streams

Combining Results in a Parallel Stream

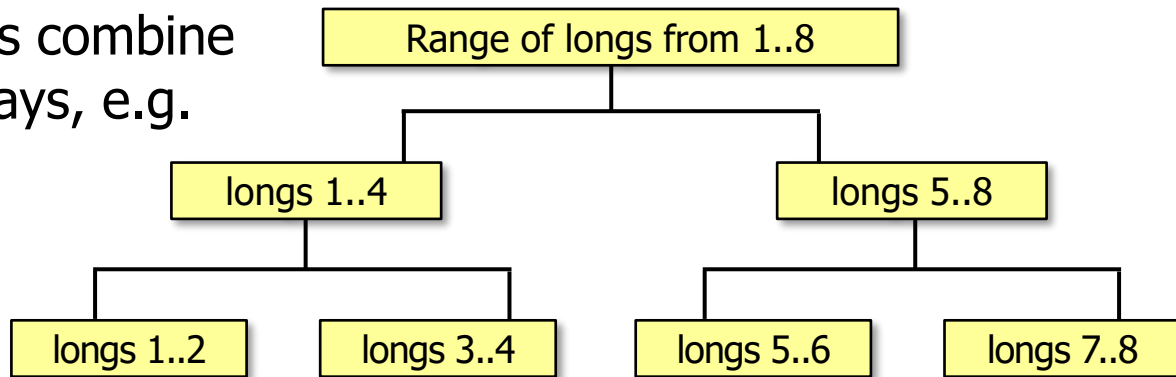
- Different terminal operations combine partial results in different ways, e.g.
 - `reduce()` creates a new immutable value



See docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html

Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
 - `reduce()` creates a new immutable value



```
long factorial(long n) {  
    return LongStream  
        .rangeClosed(1, n)  
        .parallel()  
        .reduce(1, (a, b) -> a * b,  
                (a, b) -> a * b);  
}
```

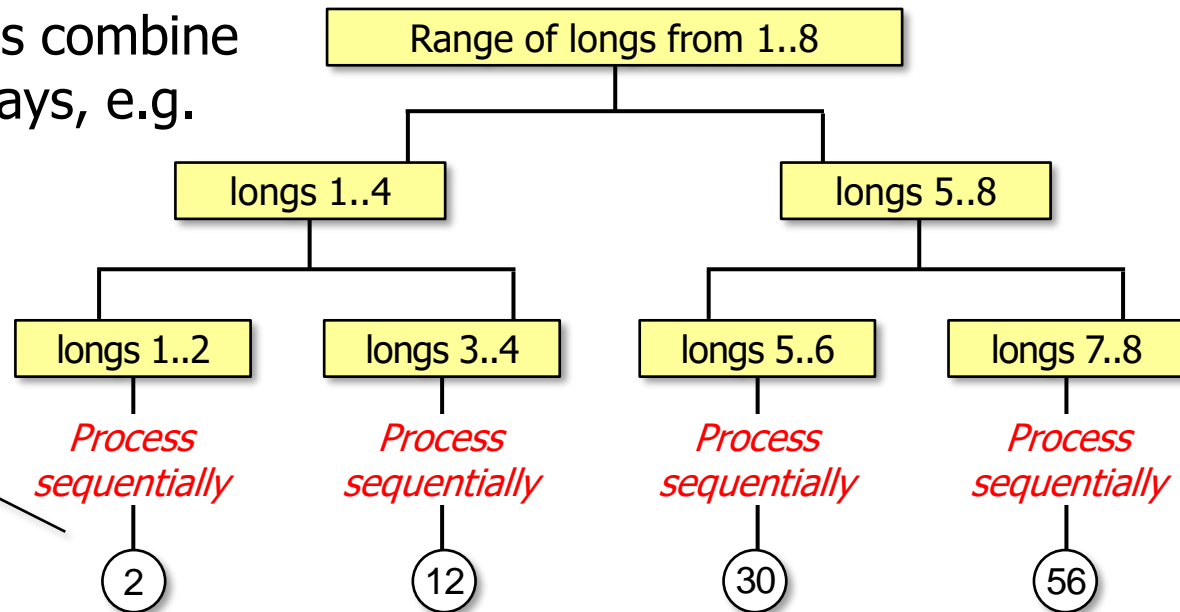
*Generate a range of longs
from 1..8 in parallel*

Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
 - `reduce()` creates a new immutable value

Multiply pair-wise values

```
long factorial(long n) {  
    return LongStream  
        .rangeClosed(1, n)  
        .parallel()  
        .reduce(1, (a, b) -> a * b);  
}
```

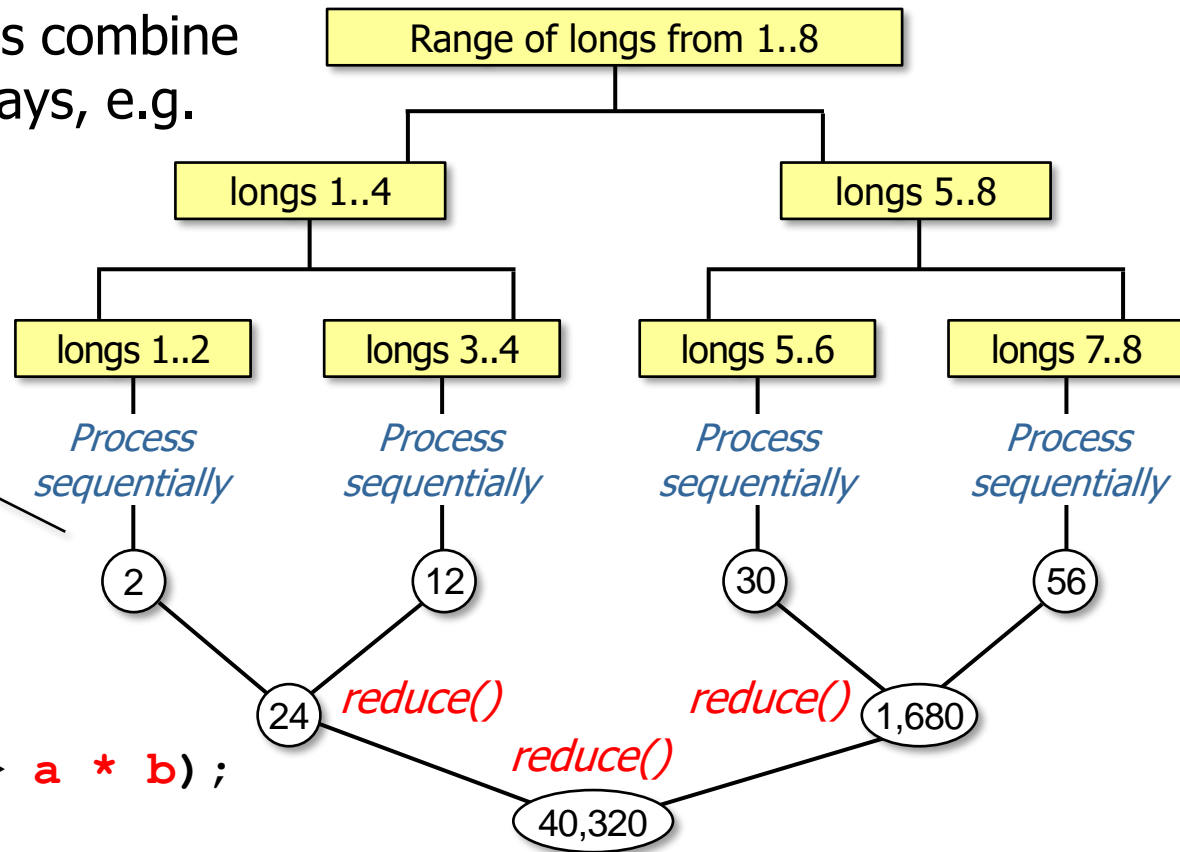


Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.
 - `reduce()` creates a new immutable value

Multiply pair-wise values

```
long factorial(long n) {  
    return LongStream  
        .rangeClosed(1, n)  
        .parallel()  
        .reduce(1, (a, b) -> a * b);  
}
```



`reduce()` combines two immutable values (e.g., long) & produces a new one

Combining Results in a Parallel Stream

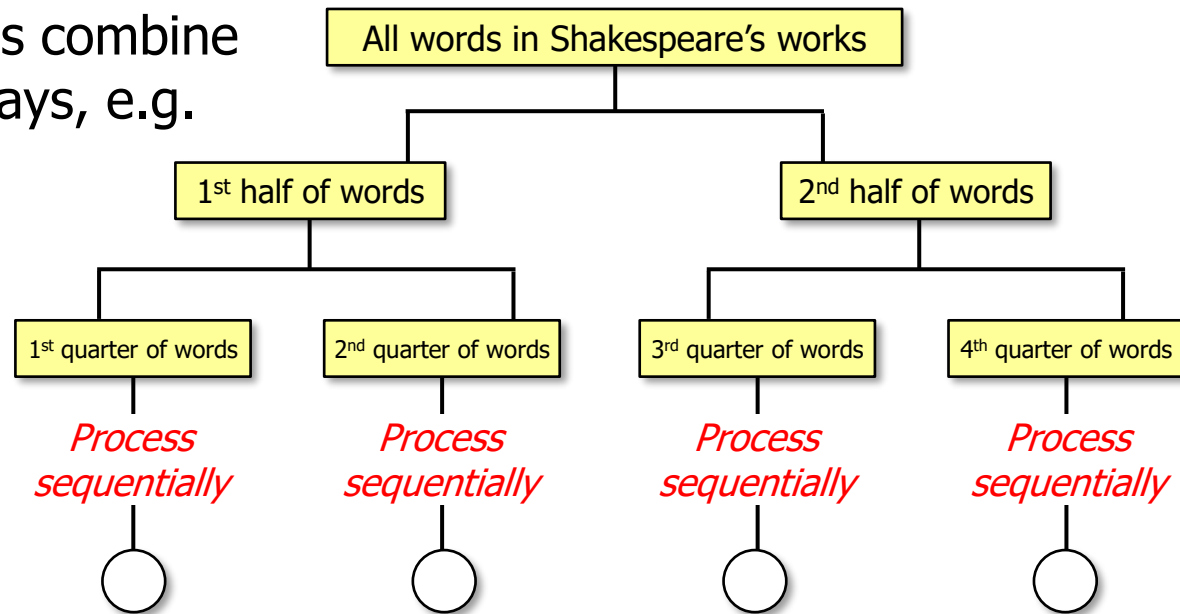
- Different terminal operations combine partial results in different ways, e.g.
 - `reduce()` creates a new immutable value
 - `collect()` mutates an existing value



Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.

- `reduce()` creates a new immutable value
- `collect()` mutates an existing value



```
Set<CharSequence>  
    uniqueWords =  
    getInput(sSHAKESPEARE, "  
        \"\\s+\" )  
    .parallelStream()  
    ...  
    .collect(toCollection(TreeSet::new));
```

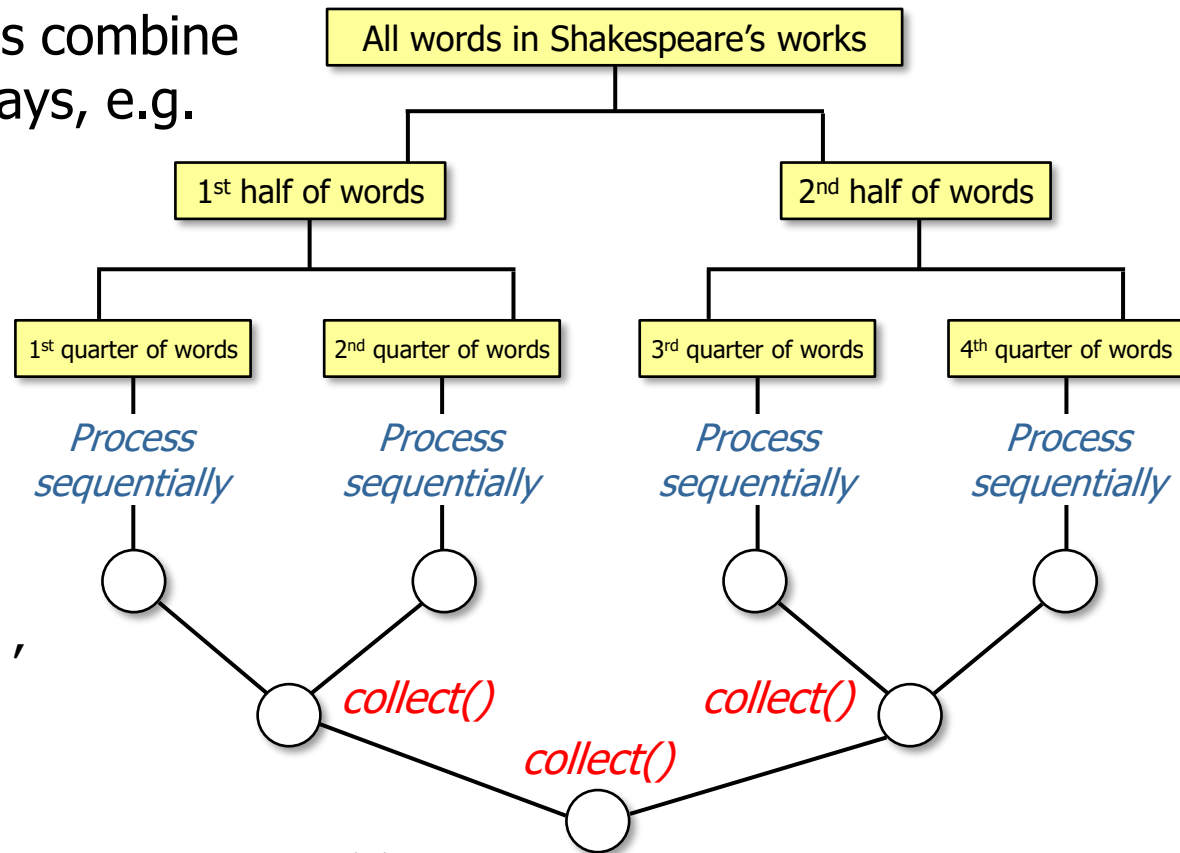
See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8/ex14

Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.

- `reduce()` creates a new immutable value
- `collect()` mutates an existing value

```
Set<CharSequence>  
    uniqueWords =  
    getInput(sSHAKESPEARE) ,  
        "\\s+")  
    .parallelStream()  
    ...  
    .collect(toCollection(TreeSet::new)) ;
```



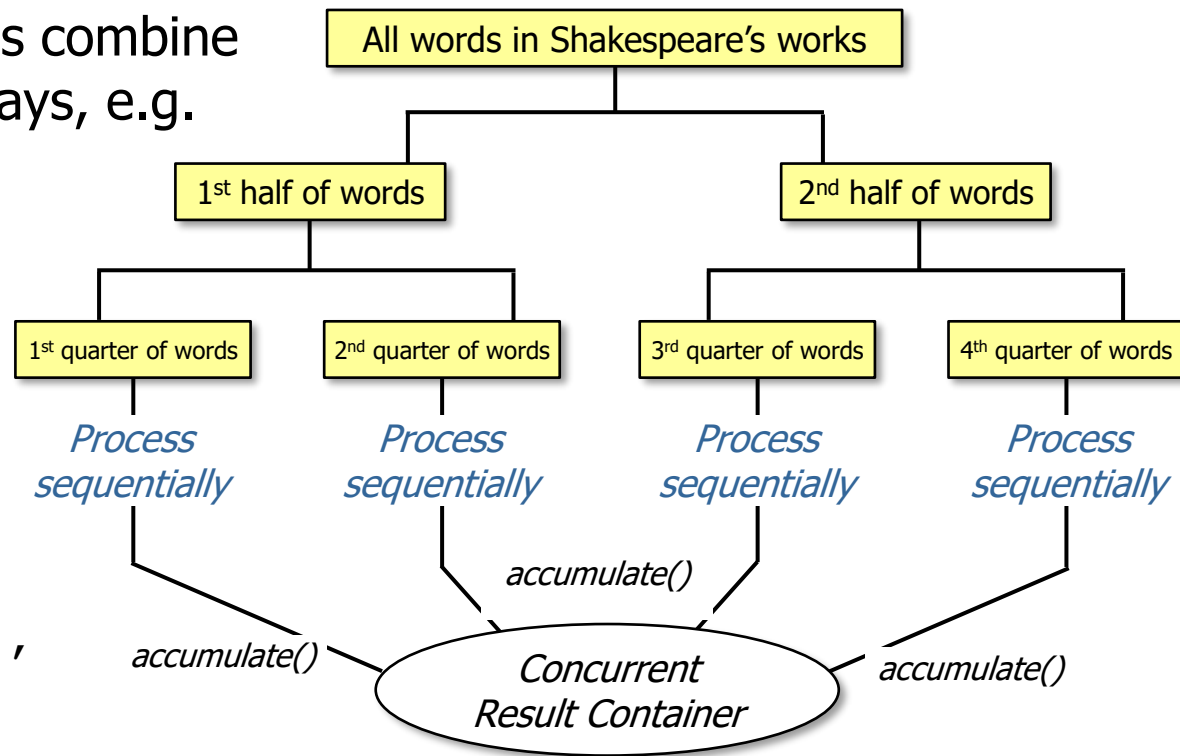
`collect()` mutates a container to accumulate the result it's producing

Combining Results in a Parallel Stream

- Different terminal operations combine partial results in different ways, e.g.

- `reduce()` creates a new immutable value
- `collect()` mutates an existing value

```
Set<CharSequence>  
    uniqueWords =  
    getInput(sSHAKESPEARE) ,  
        "\\s+")  
    .parallelStream()  
    ...  
    .collect(ConcurrentHashSetCollector.toSet());
```



Concurrent collectors (covered later) are different than non-concurrent collectors

End of Understand Java Parallel Streams Internals: Combining Results (Part 1)