Pros & Cons of Java 8 Parallel Streams

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

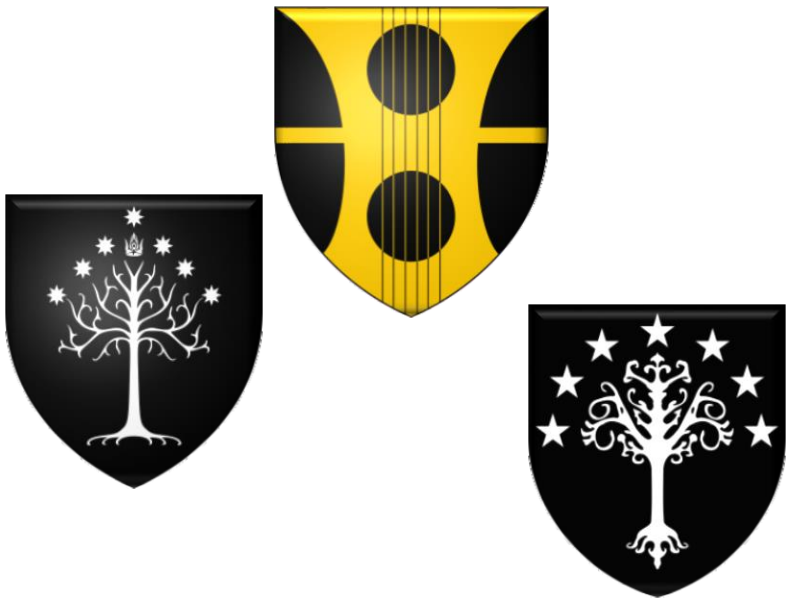
Learning Objectives in this Lesson

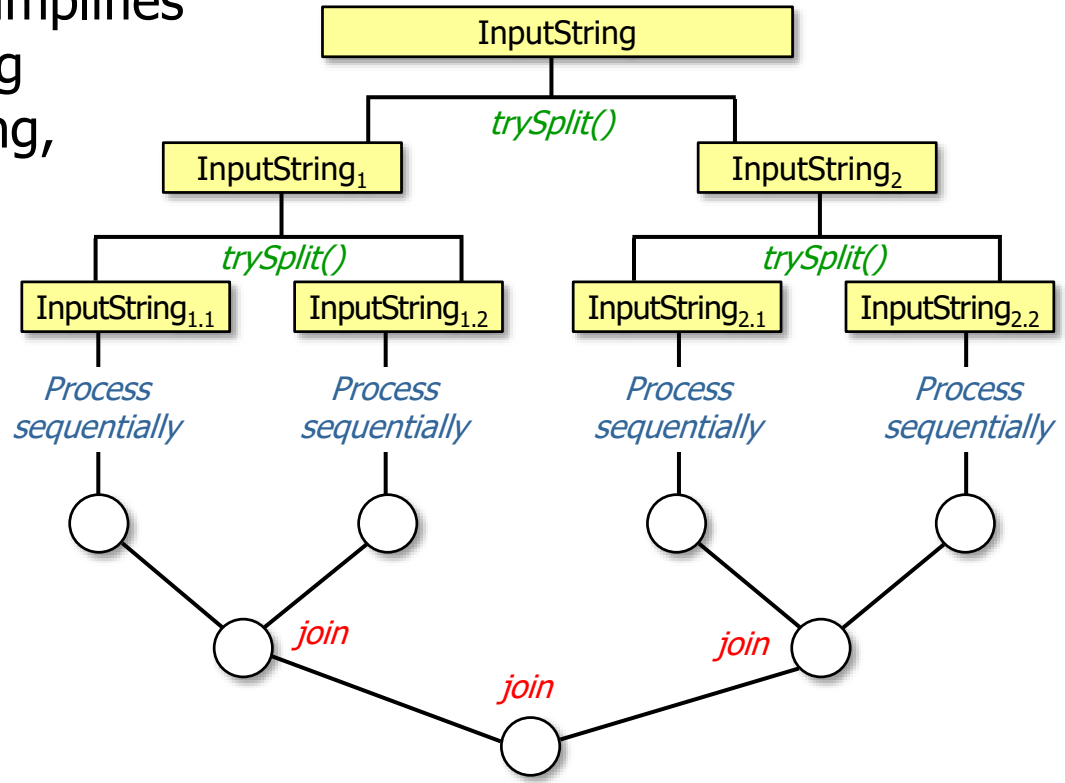
- Evaluate the pros & cons of Java 8 parallel streams

Pros of Java 8
Parallel Streams

Pros of Java 8 Parallel Streams

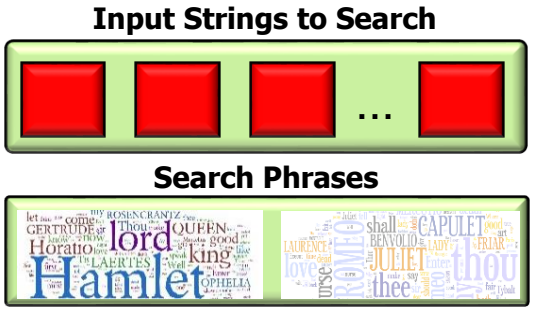
- The Java 8 streams framework simplifies parallel programming by shielding developers from details of splitting, applying, & combining results

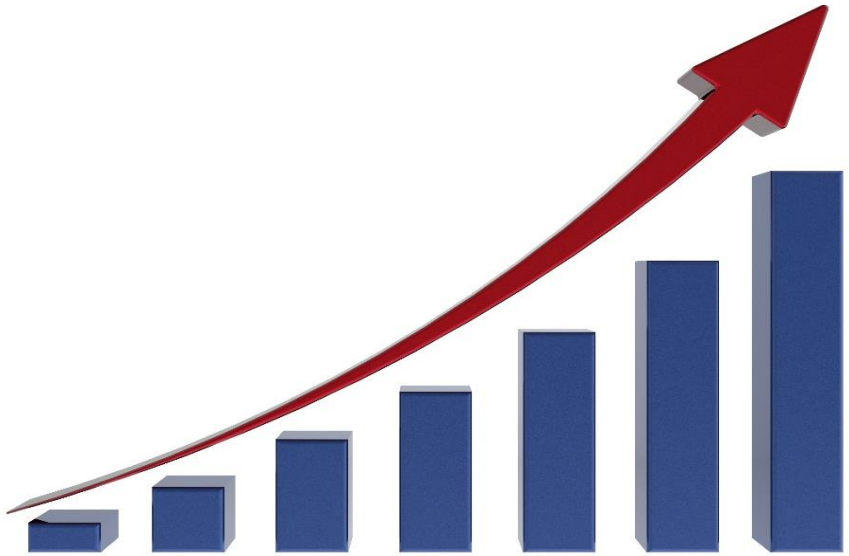


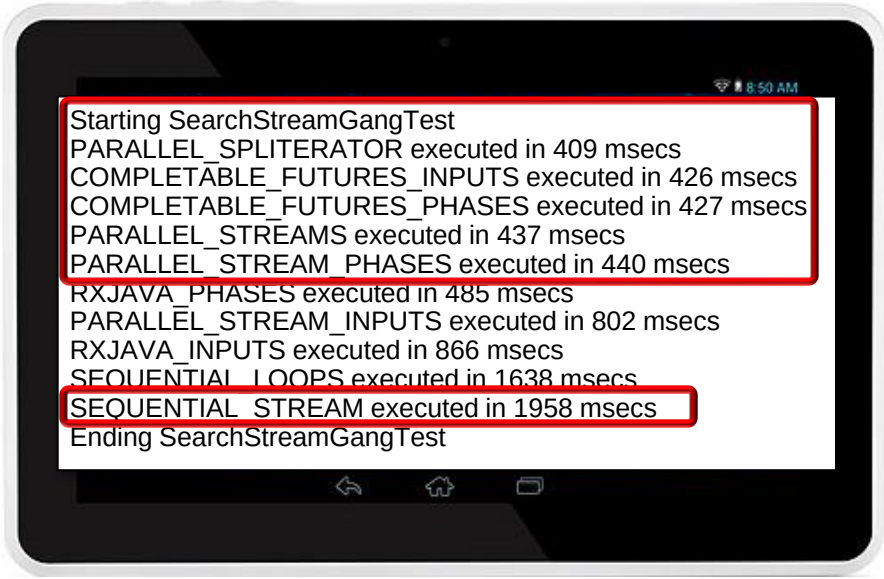


Pros of Java 8 Parallel Streams

- Parallel stream implementations are often (much) faster & more scalable than sequential (stream & loops) implementations

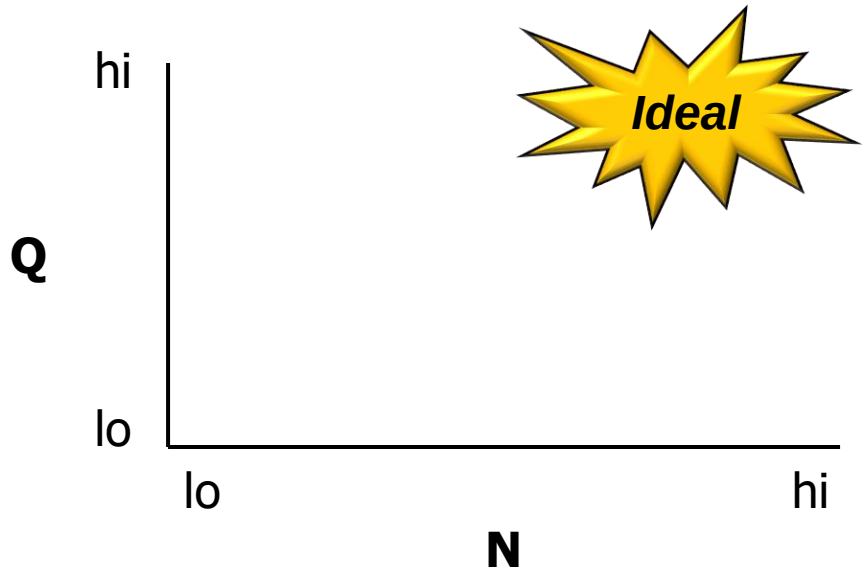






Pros of Java 8 Parallel Streams

- The performance speedup is a largely a function of the partitioning strategy for the input (N), the amount of work performed (Q), & the # of cores



The NQ model

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

Pros of Java 8 Parallel Streams

- Apps often don't need explicit synchronization or threading

Alleviates many accidental & inherent complexities of concurrency/parallelism

Pros of Java 8 Parallel Streams

- Apps often don't need explicit synchronization or threading

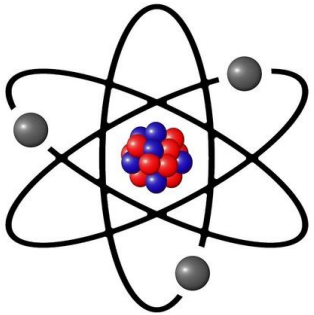
Java class library handles locking needed to protect shared mutable state

Pros of Java 8 Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```
List<List<SearchResults>>
                processStream() {
  return getInput()
     .stream()
     .map(this::processInput)
     .collect(toList());
}
```

```
List<List<SearchResults>>
                processStream() {
  return getInput()
     .parallelStream()
     .map(this::processInput)
     .collect(toList());
}
```



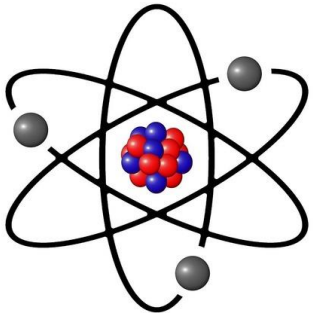
Converting sequential to parallel streams only require minuscule changes!

Pros of Java 8 Parallel Streams

- Streams ensures that the structure of sequential & parallel code is the same

```
List<SearchResults> results =
  mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(...,
                         false))
    .filter(not(SearchResults
                ::isEmpty))
    .collect(toList());
```

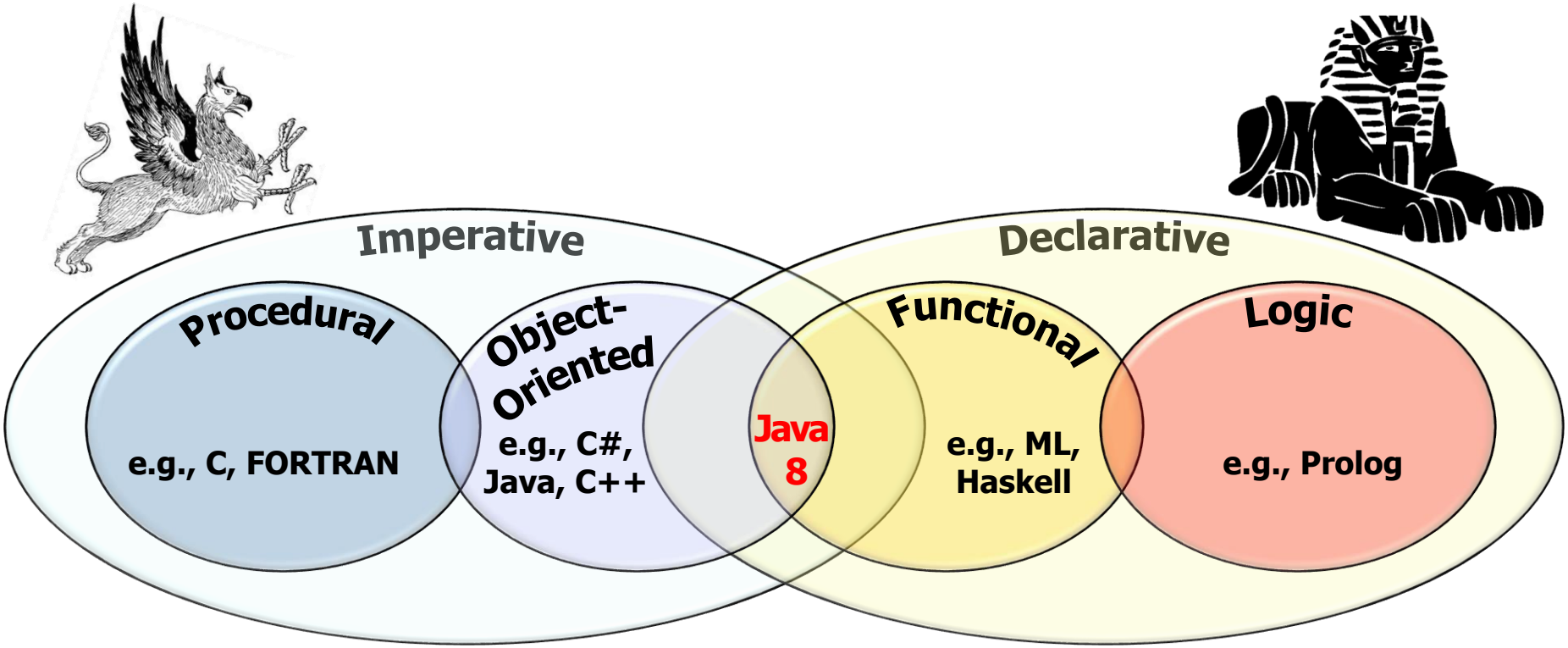
```
List<SearchResults> results =
  mPhrasesToFind
    .parallelStream()
    .map(phase ->
         searchForPhrase(...,
                         true))
    .filter(not(SearchResults
                ::isEmpty))
    .collect(toList());
```



Converting sequential to parallel streams only require minuscule changes!

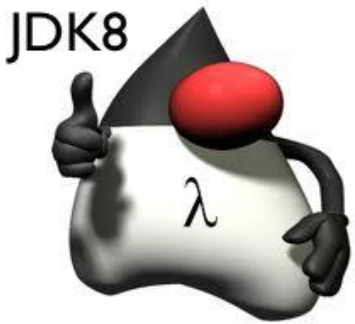
Pros of Java 8 Parallel Streams

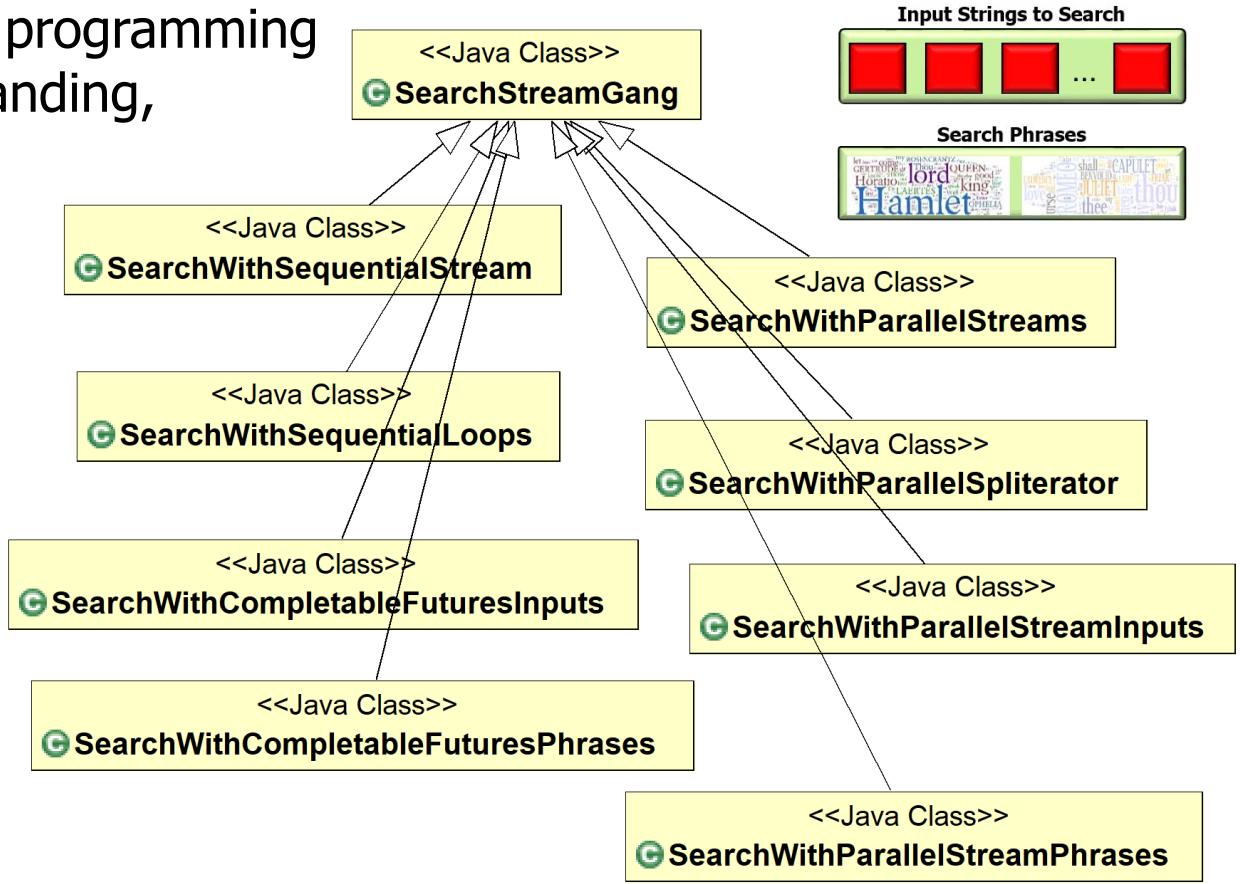
- Examples show synergies between functional & object-oriented programming



Pros of Java 8 Parallel Streams

- Object-oriented design & programming features simplify understanding, reuse, & extensibility





Pros of Java 8 Parallel Streams

- Implementing object-oriented hook methods with functional programming features helps to close gap between domain intent & computations

<<Java Class>>
SearchWithParallelStreams

processStream():List<List<SearchResults>>
processInput(CharSequence):List<SearchResults>

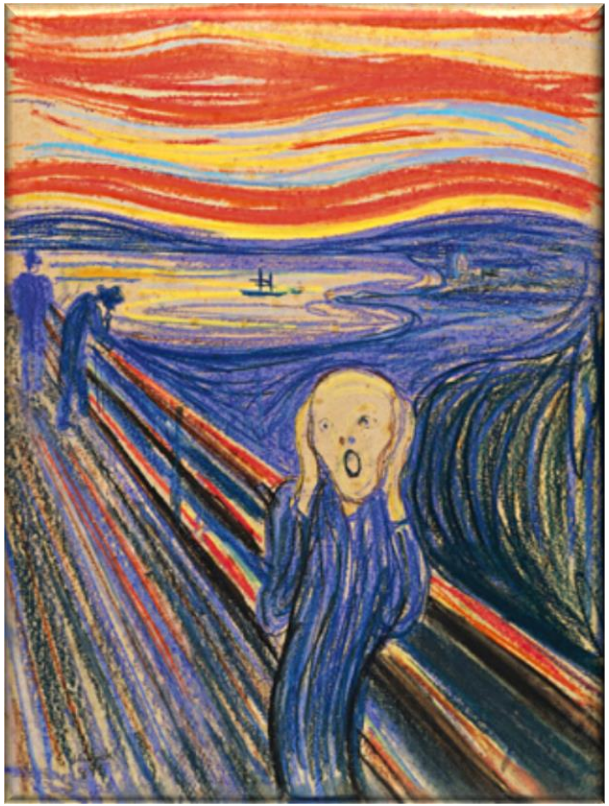
```
getInput()
   .parallelStream()
   .map(this::processInput)
   .collect(toList());
```

```
return mPhrasesToFind
   .parallelStream()
   .map(phrase -> searchForPhrase(phrase, input, title, false))
   .filter(not(SearchResults::isEmpty)
   .collect(toList());
```

Cons of Java 8 Parallel Streams

Cons of Java 8 Parallel Streams

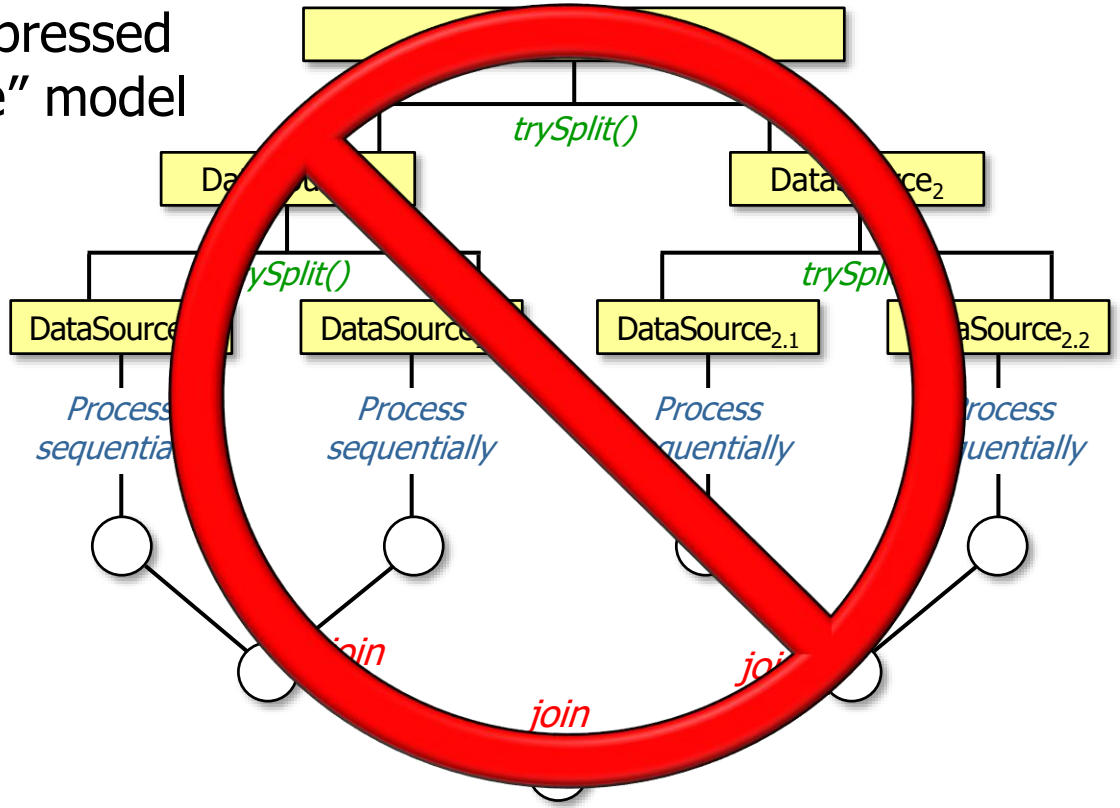
- There are some limitations with Java 8 parallel streams



The Java 8 parallel streams framework is not all unicorns & rainbows!!

Cons of Java 8 Parallel Streams

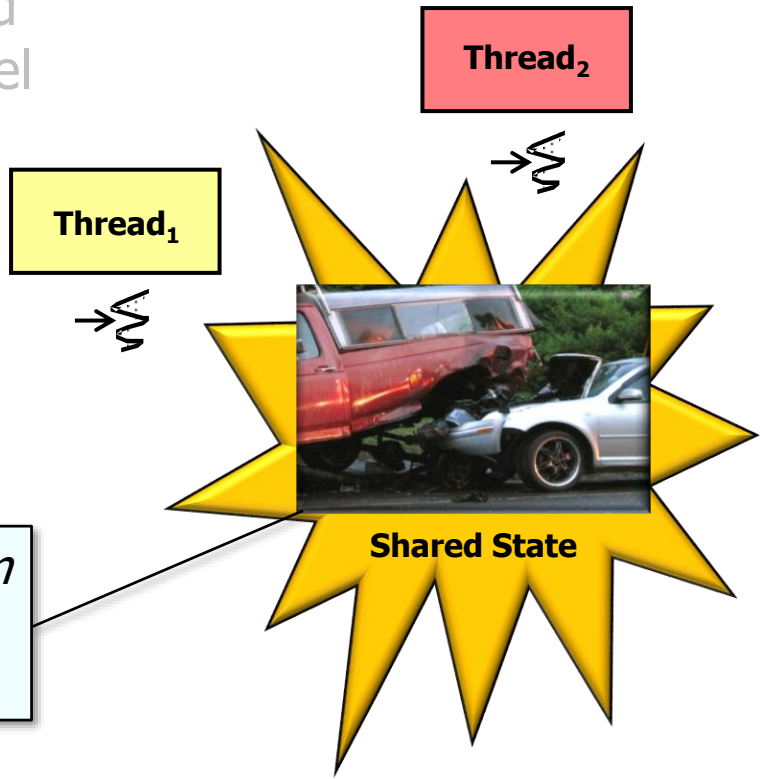
- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model



See dzone.com/articles/whats-wrong-java-8-part-iii

Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur



Race conditions occur when a program depends on the sequence or timing of threads for it to operate properly

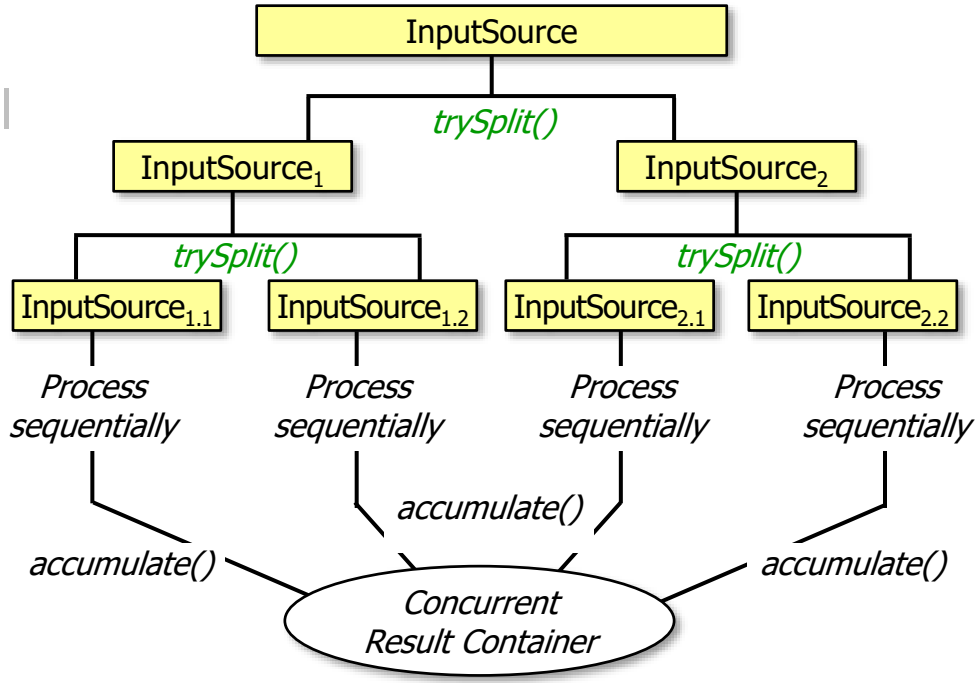
See en.wikipedia.org/wiki/Race_condition#Software

Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…

Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
    - Concurrent collectors are easier



Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool

See dzone.com/articles/think-twice-using-java-8

Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
    - Java 8 completable futures don't have this limitation

See dzone.com/articles/think-twice-using-java-8

Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
    - Java 8 completable futures don't have this limitation
    - It's important to know how to apply ManagedBlockers

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html

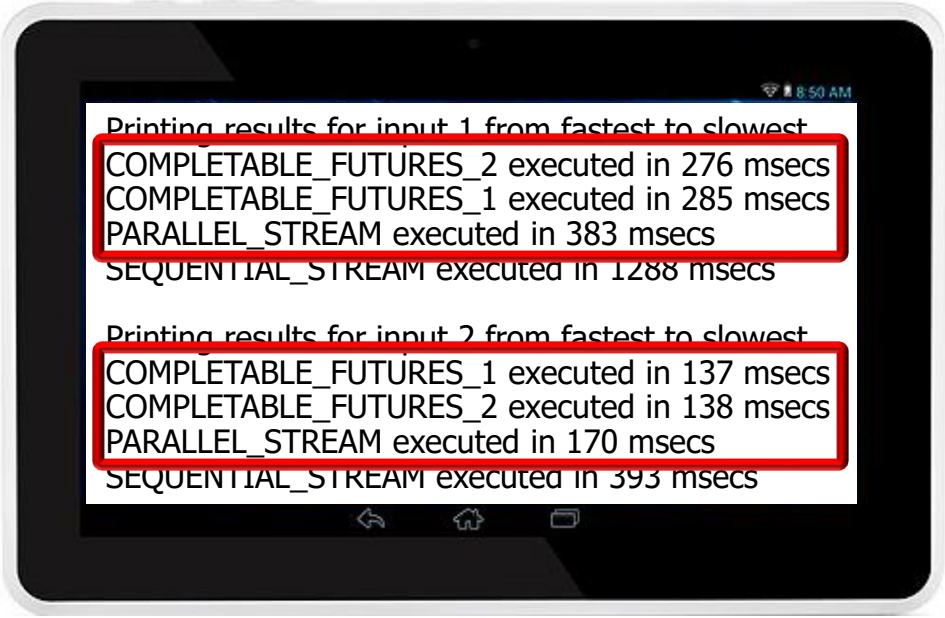
Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
  - Some overhead occurs from use of spliterators & fork-join framework

See coopsoft.com/dl/Blunder.pdf

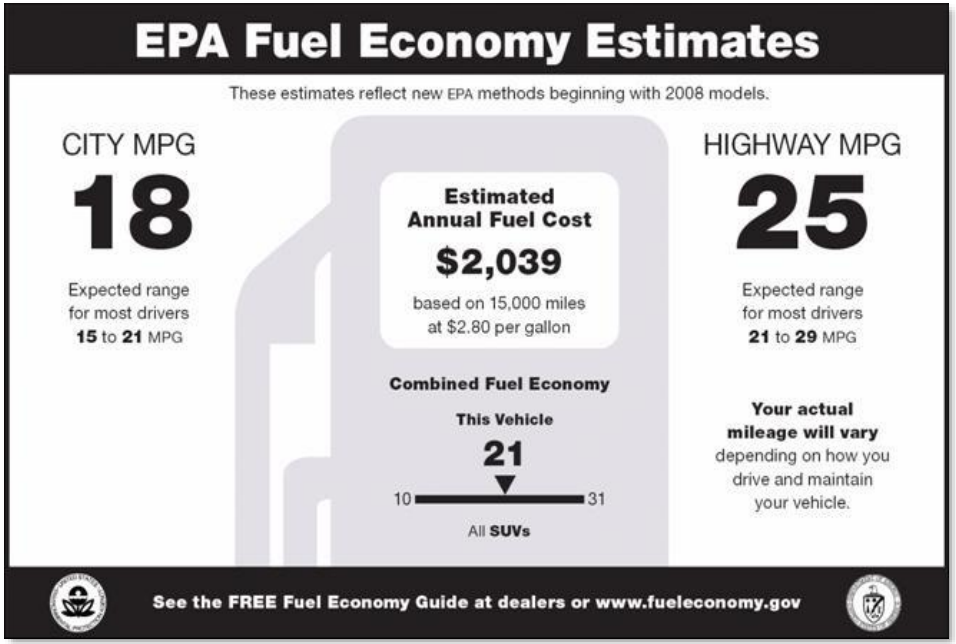
Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
  - Some overhead occurs from use of spliterators & fork-join framework
    - Java 8 completable futures may be more efficient & scalable



Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
  - Some overhead occurs from use of spliterators & fork-join framework
    - Java 8 completable futures may be more efficient & scalable



- Naturally, your mileage may vary..

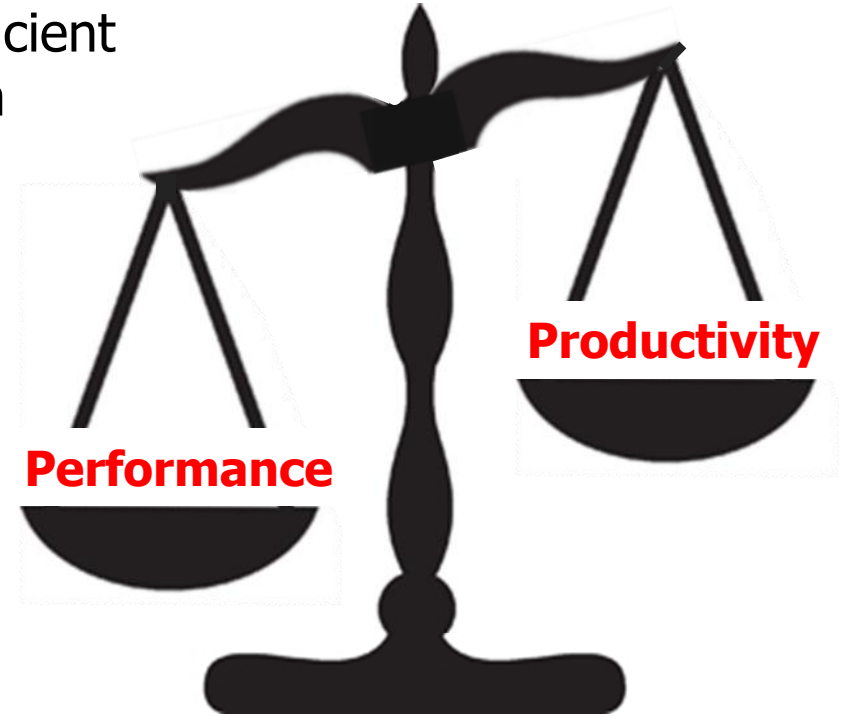
Cons of Java 8 Parallel Streams

- There are some limitations with Java 8 parallel streams, e.g.
  - Some problems can't be expressed via the "split-apply-combine" model
  - If behaviors aren't thread-safe race conditions may occur
  - Parallel spliterators may be tricky…
  - All parallel streams share a common fork-join pool
  - Some overhead occurs from use of spliterators & fork-join framework
  - There's no substitute for benchmarking!

See java-performance.info/jmh

Cons of Java 8 Parallel Streams

- In general, there's a tradeoff between computing performance & programmer productivity when choosing amongst these frameworks
  - i.e., completable futures are more efficient & scalable, but are harder to program



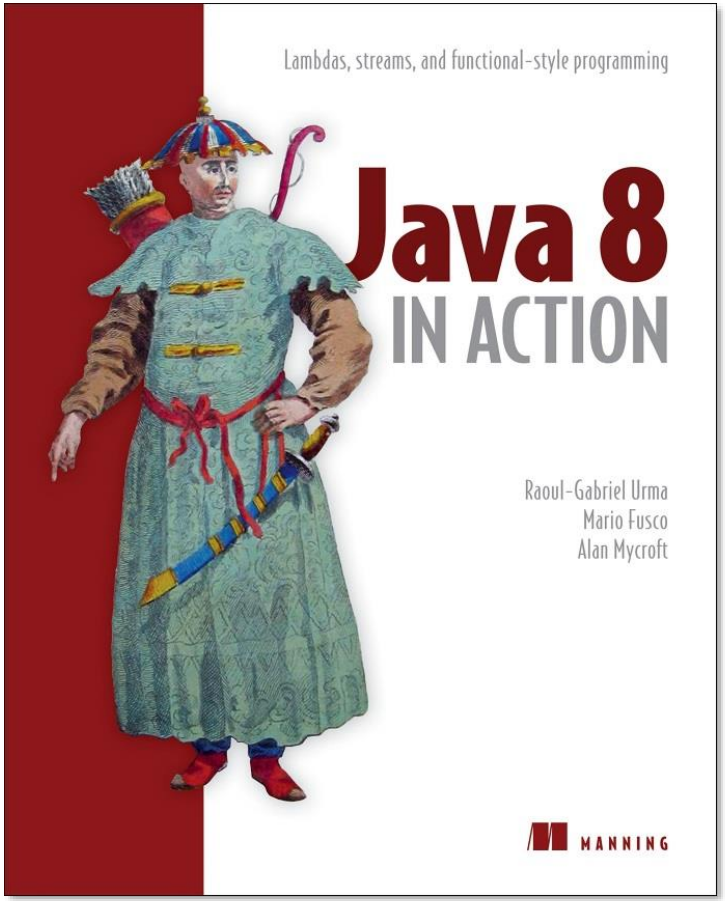
Cons of Java 8 Parallel Streams

- In general, however, the pros of Java 8 parallel streams far outweigh the cons in many use cases!!

See www.ibm.com/developerworks/library/j-jvmc2

Cons of Java 8 Parallel Streams

- Good coverage of Java 8 parallel streams appears in the book "Java 8 in Action"



See www.manning.com/books/java-8-in-action

End of Pros & Cons of Java 8 Parallel Streams