Pros & Cons of Java 8 Parallel Streams

Douglas C. Schmidt <u>d.schmidt@vanderbilt.edu</u> 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



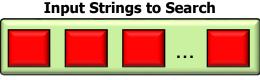
InputString_{2,2}

Process

sequentially

 The Java 8 streams framework simplifies InputString parallel programming by shielding developers from details of splitting, trySplit() InputString InputString₂ applying, & combining results trySplit() trySplit() InputString₁ InputString_{1,2} InputString₂ Process Process Process sequentially sequentially sequentially join join join

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



Search Phrases



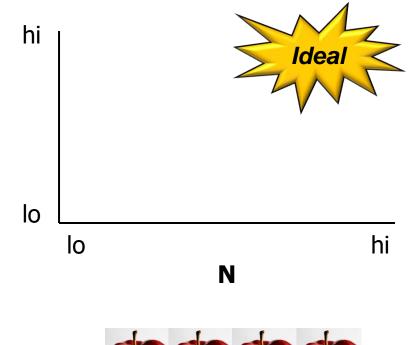


Q

 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 CPUintensive the processing is





• Apps often don't need explicit synchronization or threading





Alleviates many accidental & inherent complexities of concurrency/parallelism

Apps often don't need explicit synchronization or threading

				Jav	a La	nguag	le						
java	javac	javadoc	apt	jar					JPDA Troubleshoot			JConsole	
Security	Int'i	RMI	IDL	Deploy				Tro				criptin	g JVM T
Deployment				Java Web Start						Java Plug-in			
	AV	VT			5	Swing	8					Java 2	2D
Accessibility		Drag n Drop		Input M	ds Image FO		/0	Print Service		Sound			
IDL		JDBC		JNDI		RMI				RMI-IIOP			
Beans		Intl Support		Input/Output		JMX			JNI			Math	
Networki	ng	Override Mechanism		Security		Serialization		tion	n Exten Mecha				XML JAXI
lang and util		ollections	Concurrency Utilities		JAR		Logging		Management				
Preferenc API	es	Ref Objects	Reflection		Regular Expressions			Vei	Versioning		Zip Instrumentatio		
							The state of the s	APPROX OF					ALANA-

Java class library handles locking needed to protect shared mutable state

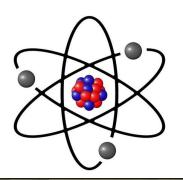
• Streams ensures that the structure of sequential & parallel code is the same

```
List<List<SearchResults>>
```

processStream() {

return getInput()
.parallelStream()
.map(this::processInput)

```
.collect(toList());
```



Converting sequential to parallel streams only require minuscule changes!

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

List<SearchResults> results = mPhrasesToFind

```
.parallelStream()
```

```
.map(phase ->
```

```
searchForPhrase(...,
```

```
false))
```

```
.collect(toList());
```

List<SearchResults> results = mPhrasesToFind .parallelStream() .map(phase ->

searchForPhrase(...,

```
true))
```

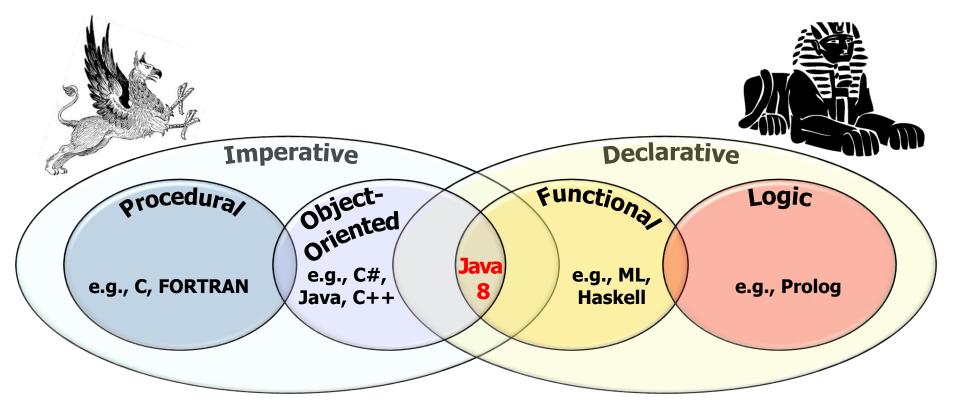
```
.filter(not(SearchResults
```

```
::isEmpty))
```

```
.collect(toList());
```

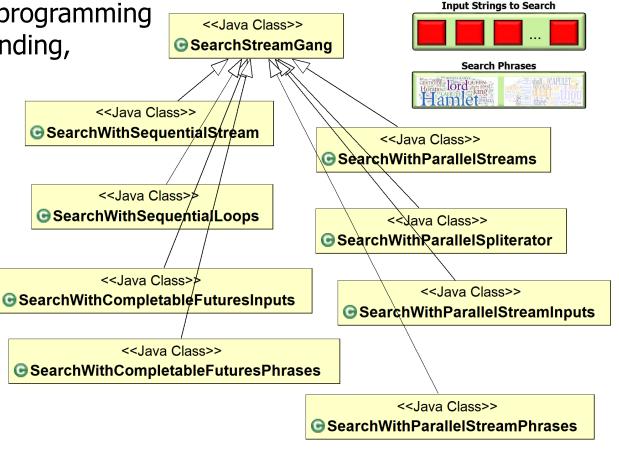
Converting sequential to parallel streams only require minuscule changes!

• Examples show synergies between functional & object-oriented programming

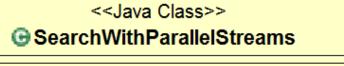


 Object-oriented design & programming features simplify understanding, reuse, & extensibility

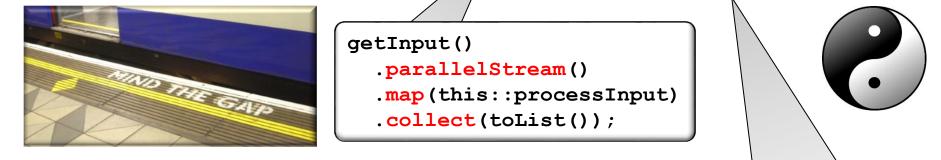




 Implementing object-oriented hook methods with functional programming features helps to close gap between domain intent & computations



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



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

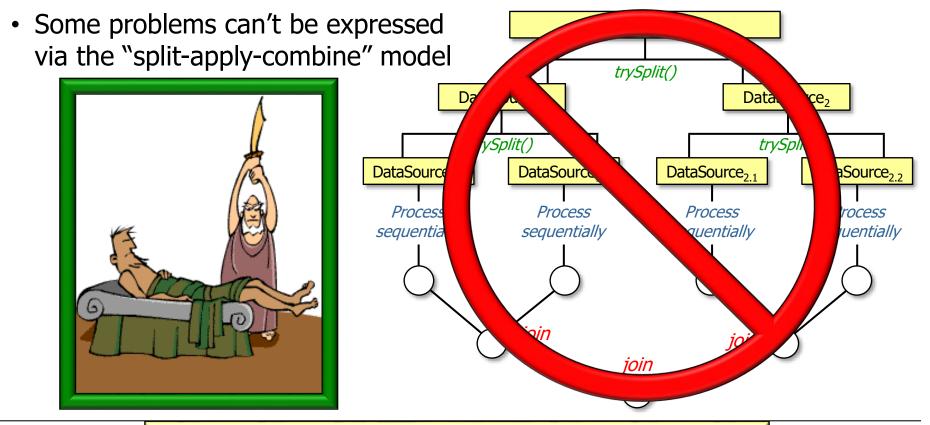
• There are some limitations with Java 8 parallel streams





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

• There are some limitations with Java 8 parallel streams, e.g.



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

- 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

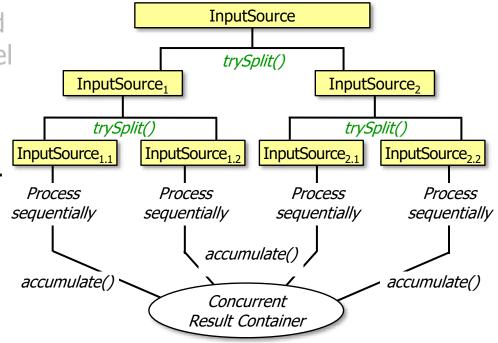
Thread₂ **Thread**₁ ≩⊱ **Shared State**

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

- 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...

- 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



- 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

- 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 <u>dzone.com/articles/think-twice-using-java-8</u>

- 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



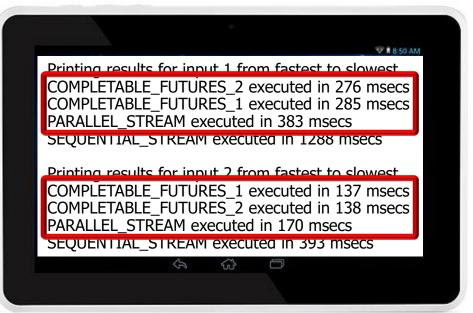
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html

- 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

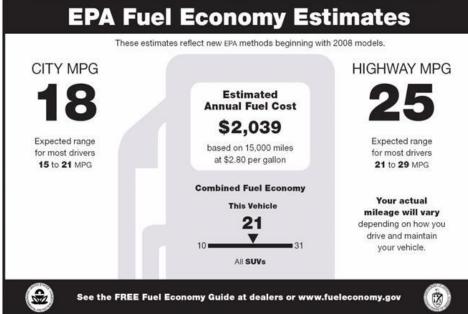


See coopsoft.com/dl/Blunder.pdf

- 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



- 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



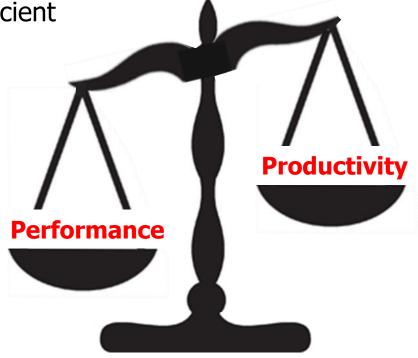
• Naturally, your mileage may vary..

- 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!

algorithms array avoiding worst **Dractices** BigDecimal binary serialization <u>bitset book review boxing byte buffer</u> collections CDU optimization data compression datatype optimization date dateformat double exceptions FastUtil FIX hashcode hashmap hdd hppc io Java 7 Java 8 java dates jdk 8 JMH JNI Koloboke map memory layout memo optimization multithreading parsing primitive collections profiler ssd String string concatenation string pool sun.misc.Unsafe tools trove

See java-performance.info/jmh

- 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

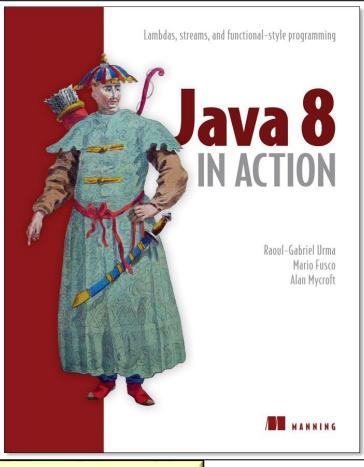


• In general, however, the pros of Java 8 parallel streams far outweigh the cons in many use cases!!



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

• 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