

Java Parallel Streams Internals: Mapping Onto the Common Fork-Join Pool

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

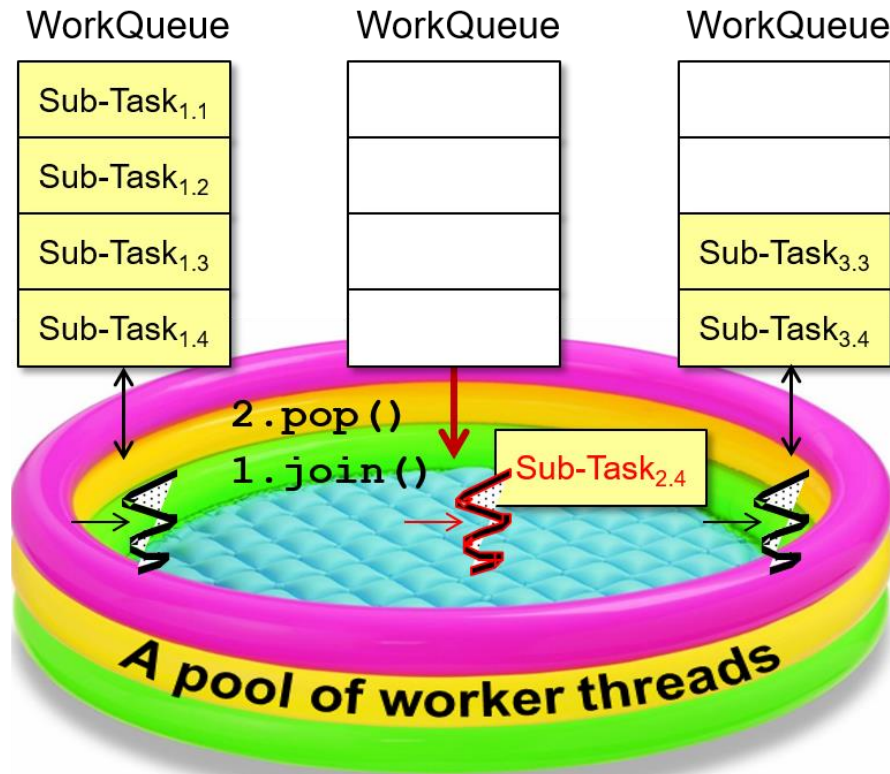
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

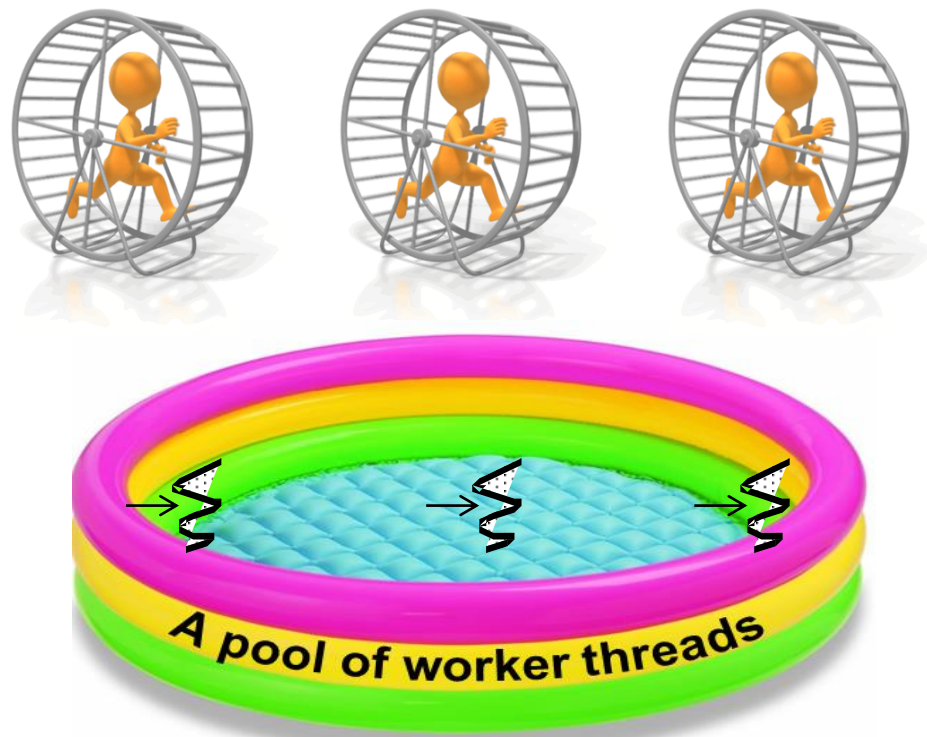
- Understand parallel stream internals, e.g.
 - Know what can change & what can't
 - Partition a data source into "chunks"
 - Process chunks in parallel via the common fork-join pool
 - Recognize how parallel streams are mapped onto the common fork-join pool framework



Mapping Parallel Streams Onto the Java Fork-Join Pool

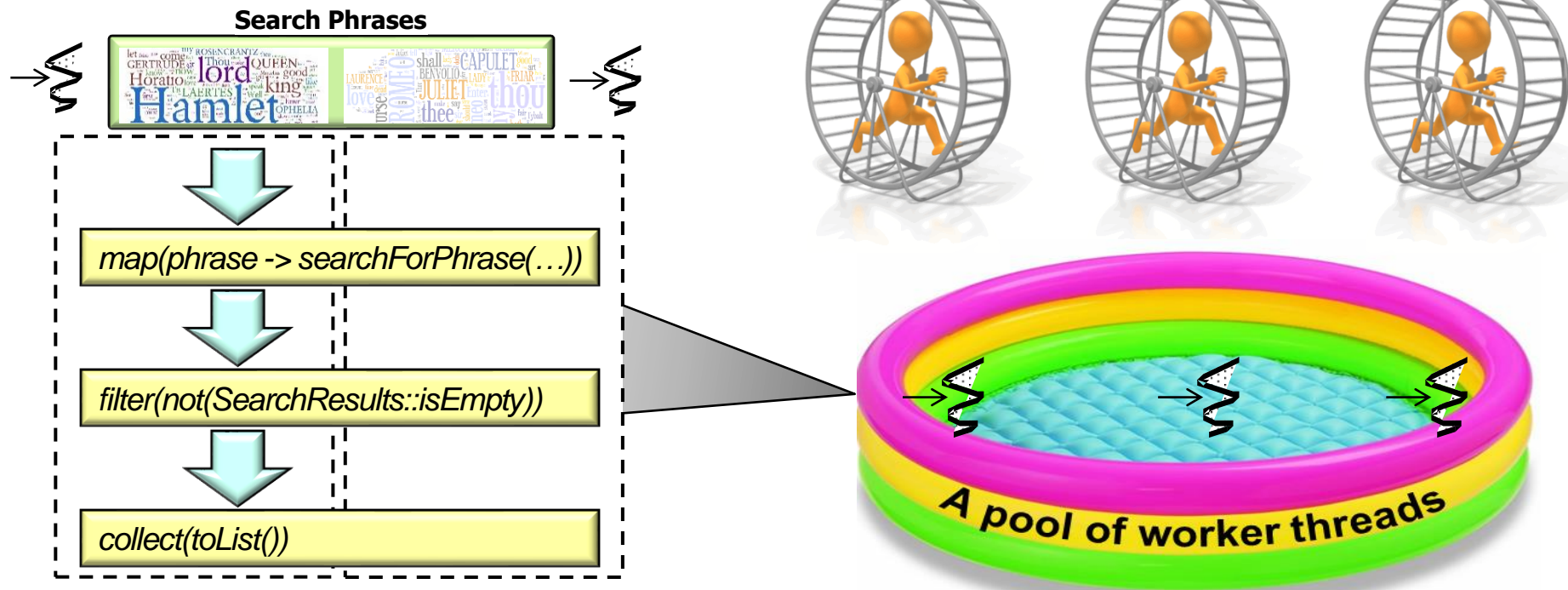
Mapping Parallel Streams Onto the Common Fork-Join Pool

- Each worker thread in the common fork-join pool runs a loop scanning for tasks to run



Mapping Parallel Streams Onto the Common Fork-Join Pool

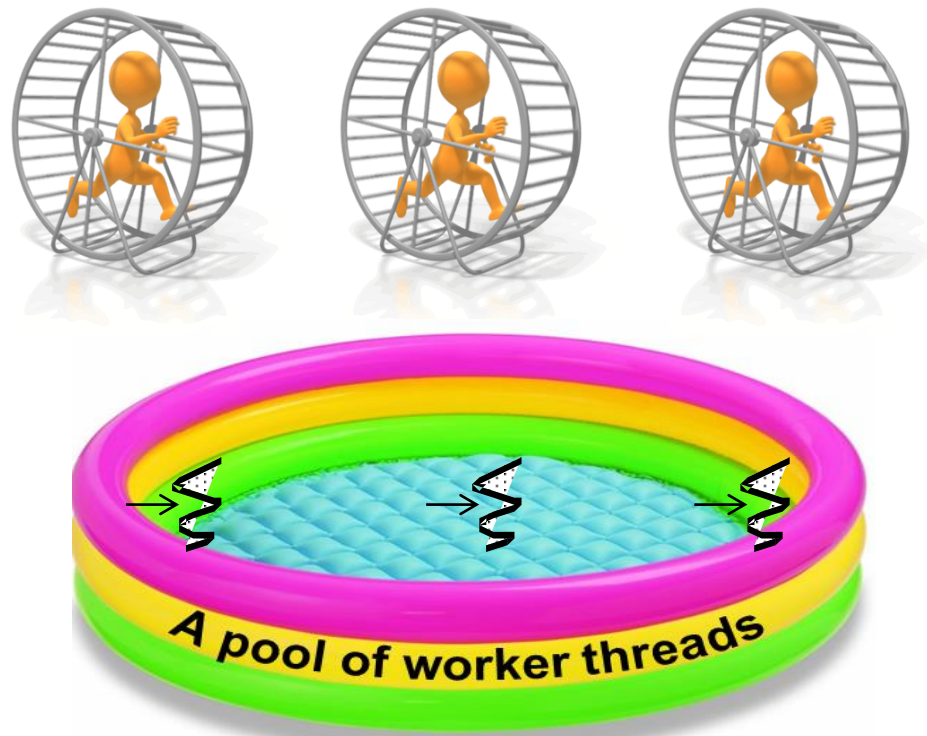
- Each worker thread in the common fork-join pool runs a loop scanning for tasks to run



In this lesson, we just care about tasks associated with parallel streams

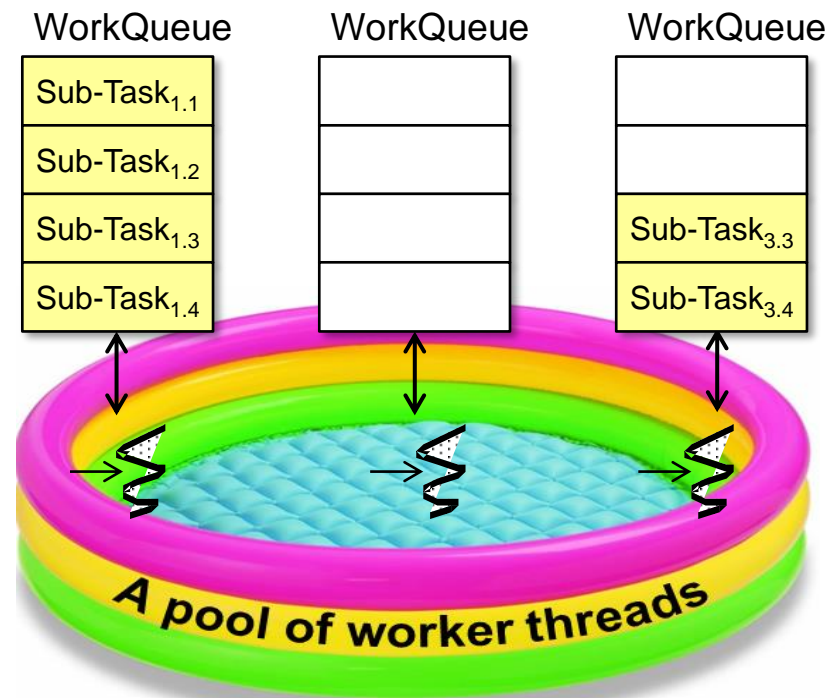
Mapping Parallel Streams Onto the Common Fork-Join Pool

- Each worker thread in the common fork-join pool runs a loop scanning for tasks to run
- Goal is to keep worker threads & cores as busy as possible!



Mapping Parallel Streams Onto the Common Fork-Join Pool

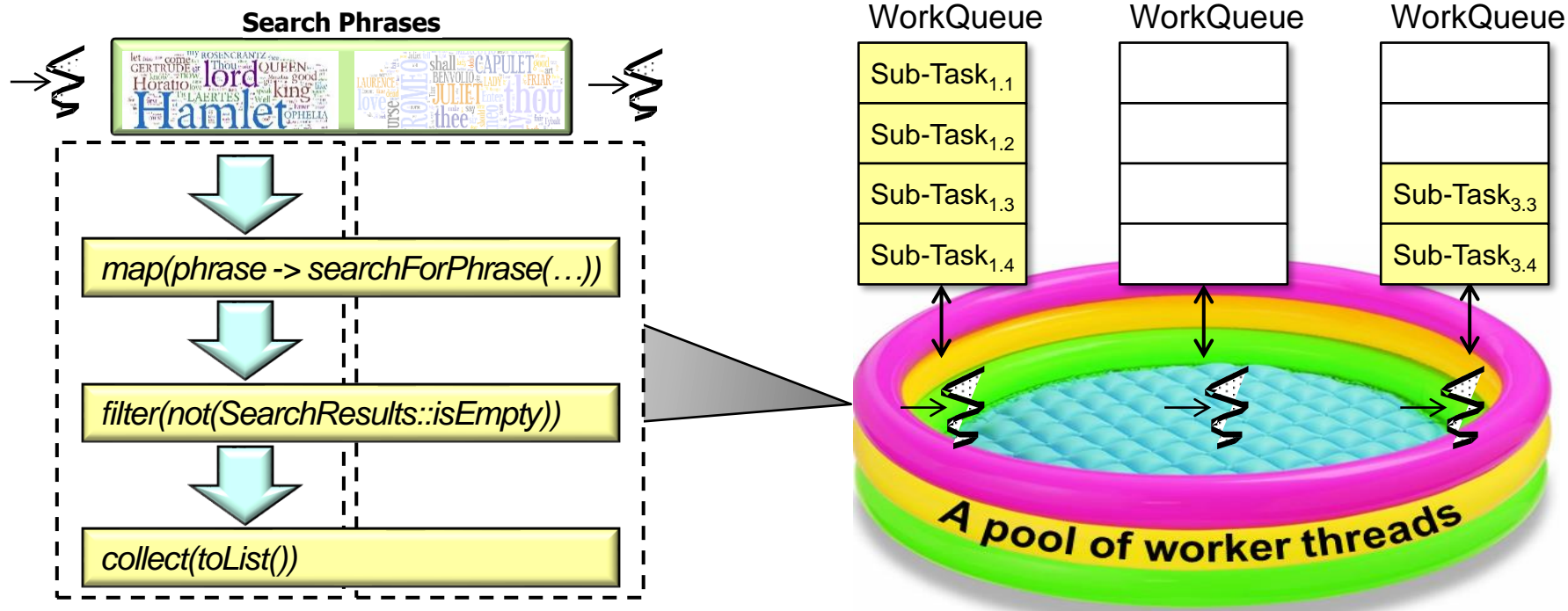
- Each worker thread in the common fork-join pool runs a loop scanning for tasks to run
 - Goal is to keep worker threads & cores as busy as possible!
- A worker thread has a “double-ended queue” (aka “deque”) that serves as its main source of tasks



See en.wikipedia.org/wiki/Double-ended_queue

Mapping Parallel Streams Onto the Common Fork-Join Pool

- The parallel streams framework automatically creates fork-join tasks that are run by worker threads in the common fork-join pool



Mapping Parallel Streams Onto the Common Fork-Join Pool

- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework

```
abstract class AbstractTask ... {  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            K taskToFork;  
            if (forkRight)  
            { forkRight = false; ... taskToFork = ...makeChild(rs); }  
            else  
            { forkRight = true; ... taskToFork = ...makeChild(ls); }  
            taskToFork.fork();  
        }  
    }  
} ...
```

Manages splitting logic, tracking of child tasks, & intermediate results

See openjdk/8-b132/java/util/stream/AbstractTask.java

Mapping Parallel Streams Onto the Common Fork-Join Pool

- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework

```
abstract class AbstractTask ... {  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            K taskToFork;  
            if (forkRight)  
            { forkRight = false; ... taskToFork = ...makeChild(rs); }  
            else  
            { forkRight = true; ... taskToFork = ...makeChild(ls); }  
            taskToFork.fork();  
        }  
    }  
} ...
```

Decides whether to split a task further and/or compute it directly

Mapping Parallel Streams Onto the Common Fork-Join Pool

- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework

```
abstract class AbstractTask ... { ...  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            K taskToFork;  
            if (forkRight)  
            { forkRight = false; ... taskToFork = ...makeChild(rs); }  
            else  
            { forkRight = true; ... taskToFork = ...makeChild(ls); }  
            taskToFork.fork();  
        }  
    } ...
```

*Keep partitioning input source
until trySplit() returns null*

See docs.oracle.com/javase/8/docs/api/java/util/Spliterator.html#trySplit

Mapping Parallel Streams Onto the Common Fork-Join Pool

- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework


```
abstract class AbstractTask ... { ...  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            K taskToFork;  
            if (forkRight)  
            { forkRight = false; ... taskToFork = ...makeChild(rs); }  
            else  
            { forkRight = true; ... taskToFork = ...makeChild(ls); }  
            taskToFork.fork();  
        }  
    } ...  
}
```

*Alternate which child is forked
to avoid biased spliterators*

Mapping Parallel Streams Onto the Common Fork-Join Pool

- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework

```
abstract class AbstractTask ... { ...  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            K taskToFork;  
            if (forkRight)  
            { forkRight = false; ... taskToFork = ...makeChild(rs); }  
            else  
            { forkRight = true; ... taskToFork = ...makeChild(ls); }  
            taskToFork.fork() ;  
        }  
    } ...
```



Fork a new sub-task & continue processing the other in the loop

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html#fork

Mapping Parallel Streams Onto the Common Fork-Join Pool

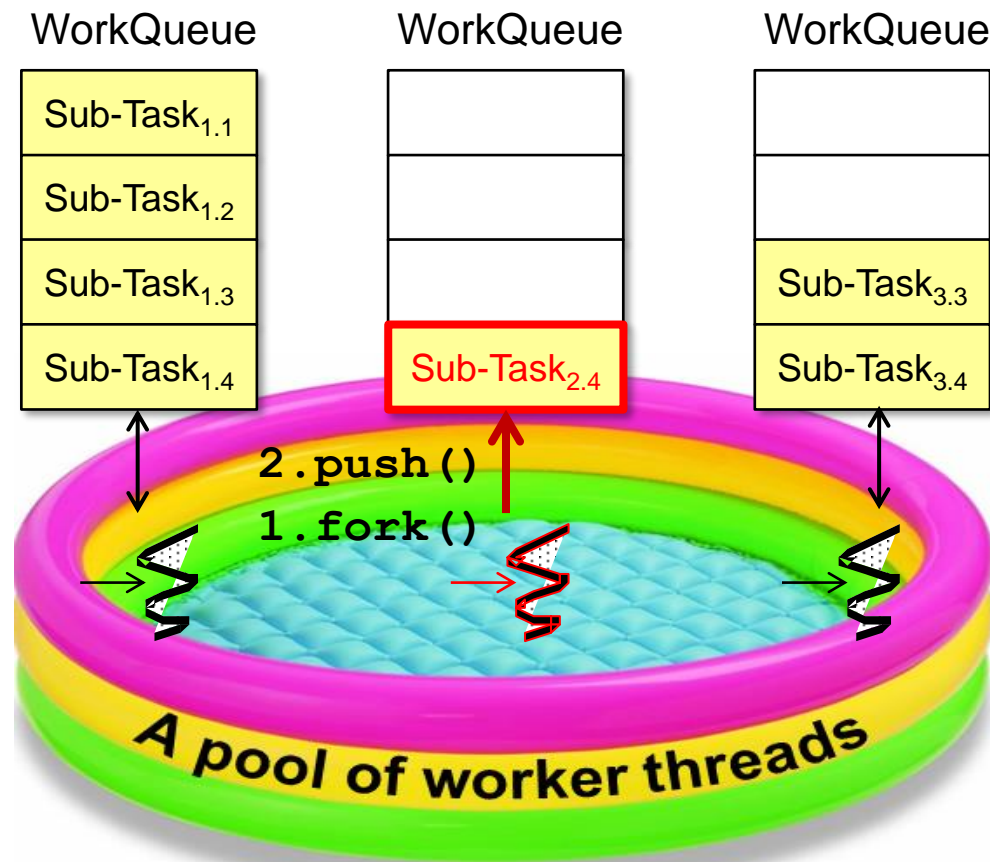
- The AbstractTask super class is used by most fork-join tasks to implement the parallel streams framework

```
abstract class AbstractTask ... { ...  
    public void compute() {  
        Spliterator<P_IN> rs = spliterator, ls;  
        boolean forkRight = false; ...  
        while(... (ls = rs.trySplit()) != null){  
            ...  
        }  
        task.setLocalResult(task.doLeaf());  
    } ...
```

After trySplit() returns null this method typically calls forEachRemaining(), which then processes all elements sequentially by calling tryAdvance()

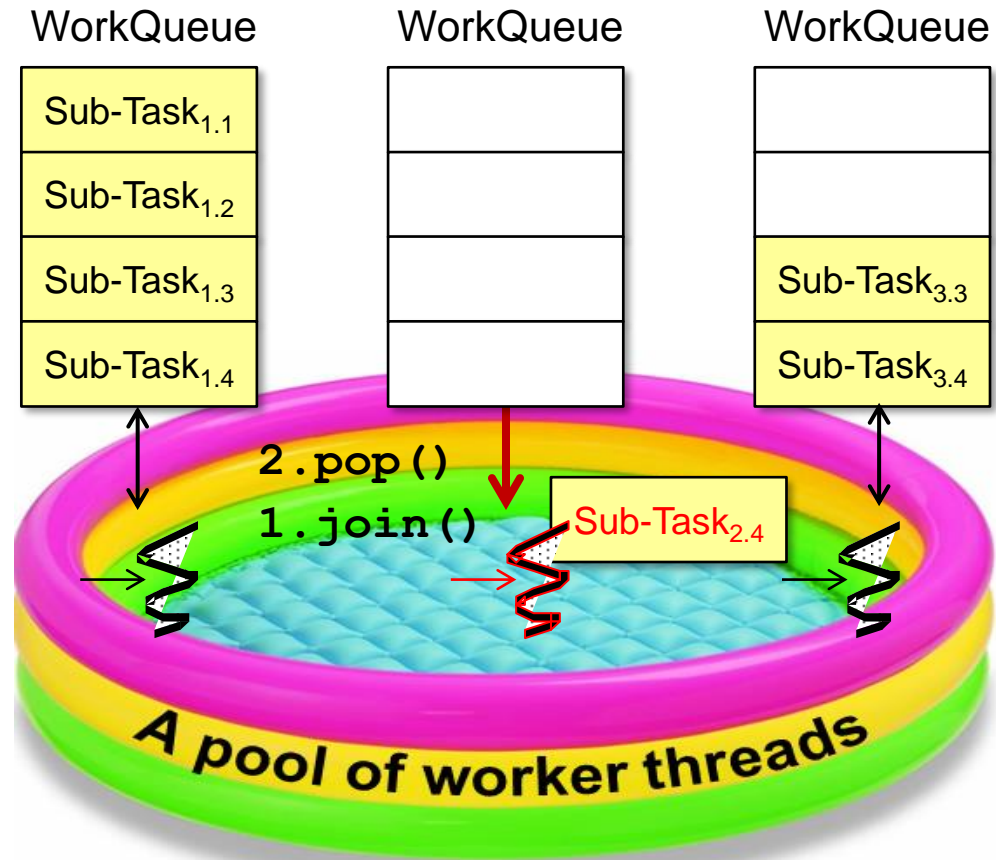
Mapping Parallel Streams Onto the Common Fork-Join Pool

- After the `AbstractTask.compute()` method calls `fork()` on a task this task is pushed onto the head of its worker thread's deque



Mapping Parallel Streams Onto the Common Fork-Join Pool

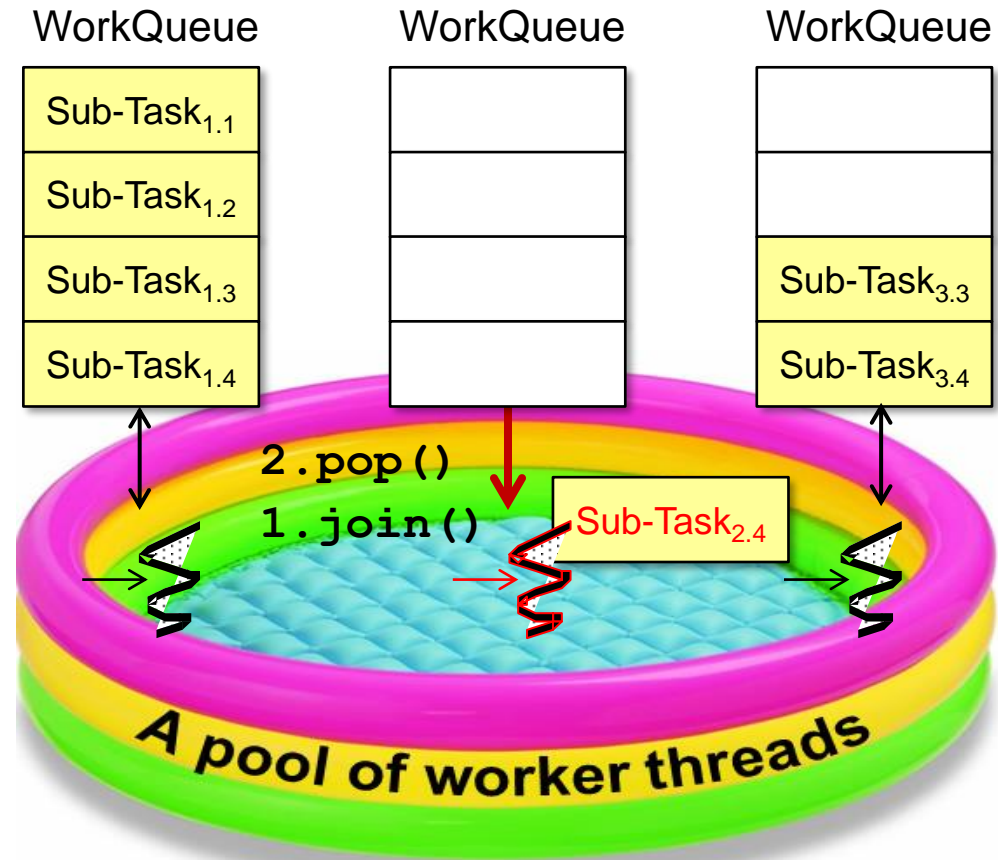
- Each worker thread processes its deque in LIFO order



See [en.wikipedia.org/wiki/Stack \(abstract data type\)](https://en.wikipedia.org/wiki/Stack_(abstract_data_type))

Mapping Parallel Streams Onto the Common Fork-Join Pool

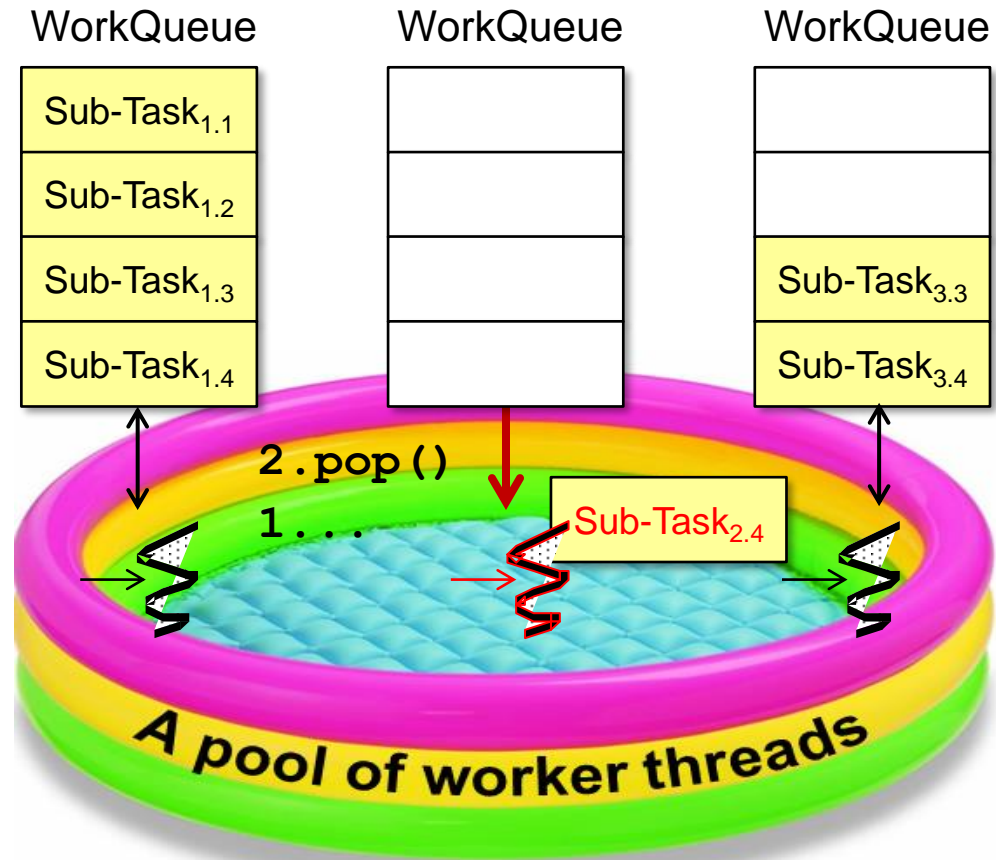
- Each worker thread processes its deque in LIFO order
- A task pop'd from the head of a deque is run to completion



See en.wikipedia.org/wiki/Run_to_completion_scheduling

Mapping Parallel Streams Onto the Common Fork-Join Pool

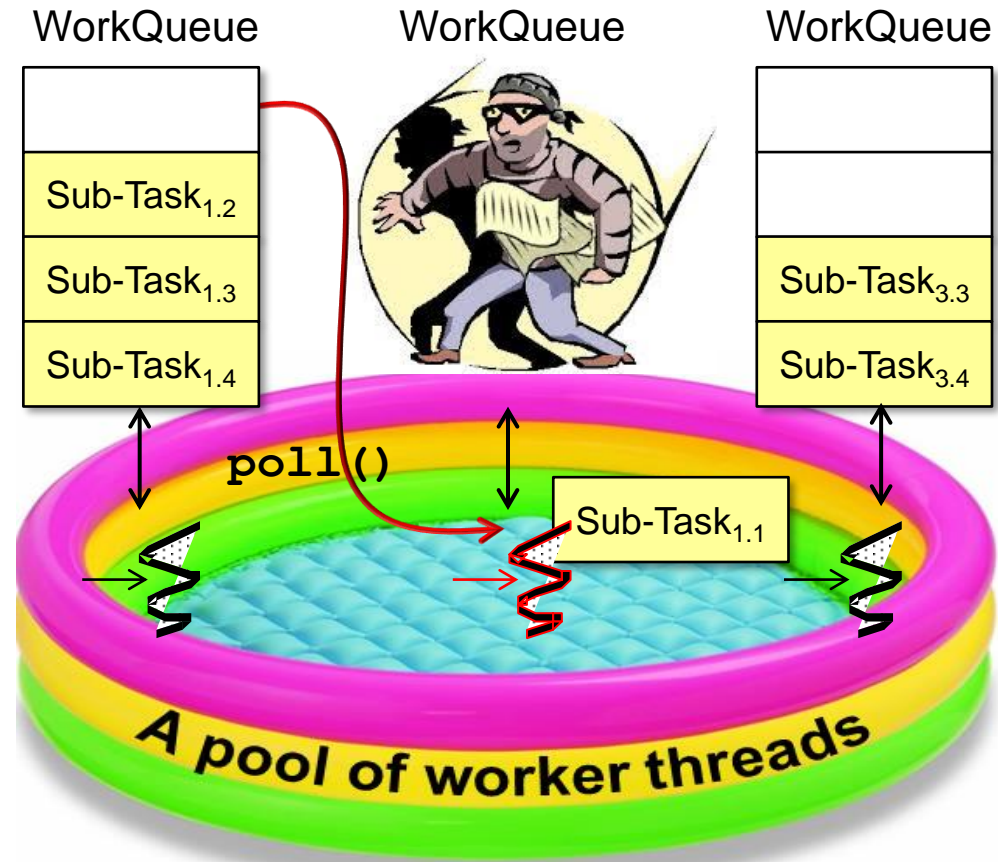
- Each worker thread processes its deque in LIFO order
 - A task pop'd from the head of a deque is run to completion
- LIFO order improves locality of reference & cache performance



See en.wikipedia.org/wiki/Locality_of_reference

Mapping Parallel Streams Onto the Common Fork-Join Pool

- To maximize core utilization, idle worker threads “steal” work from the tail of busy threads’ deques



See upcoming lessons on “*The Java Fork-Join Framework*”

End of Java Parallel Stream Internals: Mapping Onto the Common Fork-Join Pool