

# The Java ForkJoinTask Class

**Douglas C. Schmidt**

**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)**



**Professor of Computer Science**

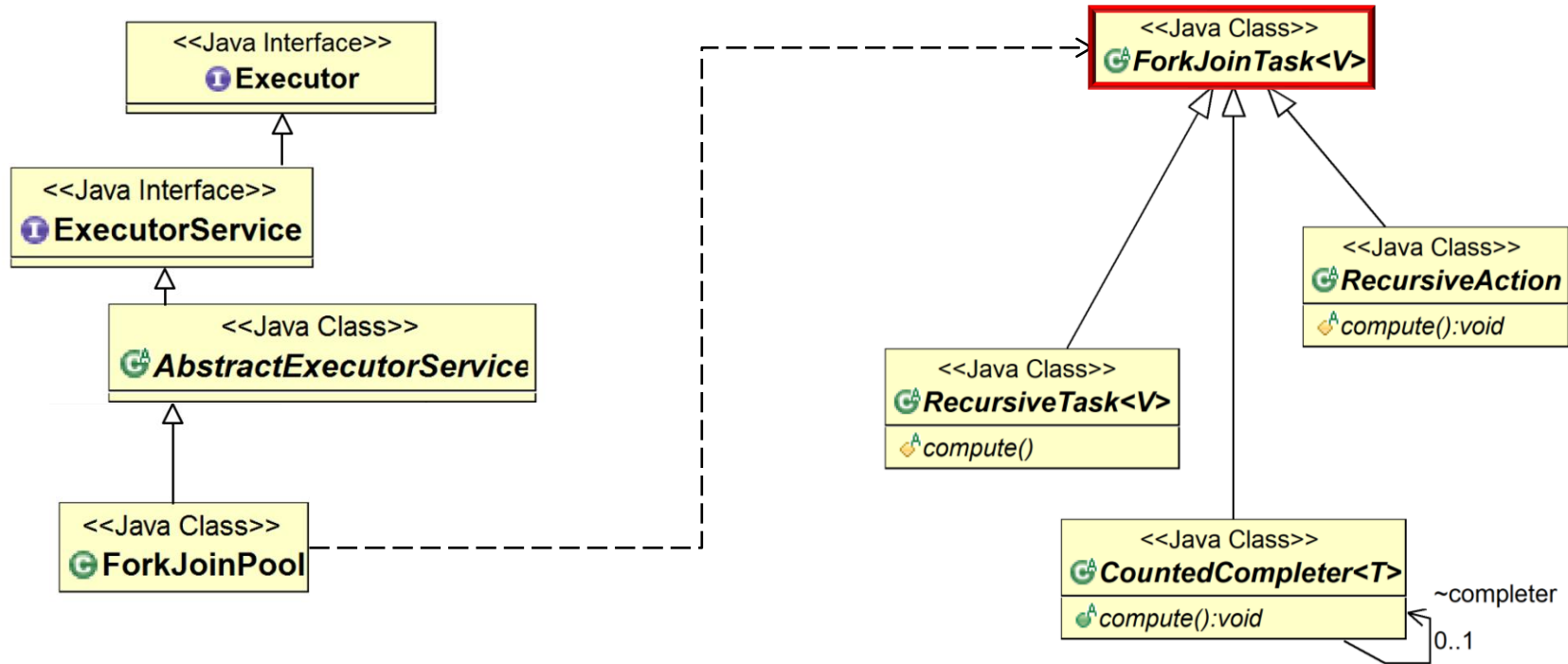
**Institute for Software  
Integrated Systems**

**Vanderbilt University  
Nashville, Tennessee, USA**



# Learning Objectives in this Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure & functionality of the fork-join framework



---

# Overview of the ForkJoinTask Class

# Overview of the ForkJoinTask Class

- A ForkJoinTask associates a chunk of data along with a computation on that data

## Class ForkJoinTask<V>

java.lang.Object  
java.util.concurrent.ForkJoinTask<V>

### All Implemented Interfaces:

Serializable, Future<V>

### Direct Known Subclasses:

CountedCompleter, RecursiveAction, RecursiveTask

---

```
public abstract class ForkJoinTask<V>
    extends Object
    implements Future<V>, Serializable
```

Abstract base class for tasks that run within a `ForkJoinPool`. A `ForkJoinTask` is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a `ForkJoinPool`, at the price of some usage limitations.

A "main" `ForkJoinTask` begins execution when it is explicitly submitted to a `ForkJoinPool`, or, if not already engaged in a `ForkJoin` computation, commenced in the `ForkJoinPool.commonPool()` via `fork()`, `invoke()`, or related methods. Once started, it will usually in turn start other subtasks. As indicated by the name of this class, many programs using `ForkJoinTask` employ only methods `fork()` and `join()`, or derivatives such as `invokeAll`. However, this class also provides a number of other methods that can come into play in advanced usages, as well as extension mechanics that allow support of new forms of `fork/join` processing.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html)

# Overview of the ForkJoinTask Class

- A ForkJoinTask associates a chunk of data along with a computation on that data
- This enables fine-grained data parallelism



## Class ForkJoinTask<V>

java.lang.Object  
java.util.concurrent.ForkJoinTask<V>

### All Implemented Interfaces:

Serializable, Future<V>

### Direct Known Subclasses:

CountedCompleter, RecursiveAction, RecursiveTask

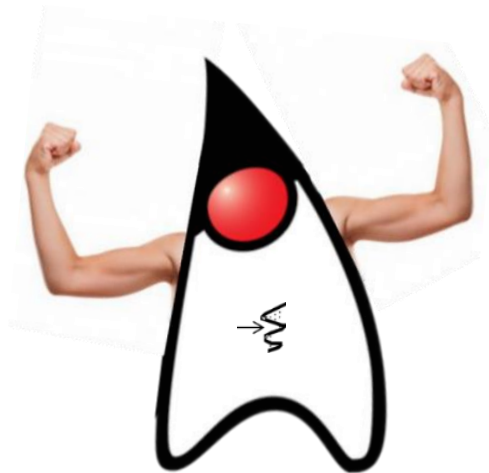
```
public abstract class ForkJoinTask<V>  
extends Object  
implements Future<V>, Serializable
```

Abstract base class for tasks that run within a `ForkJoinPool`. A `ForkJoinTask` is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a `ForkJoinPool`, at the price of some usage limitations.

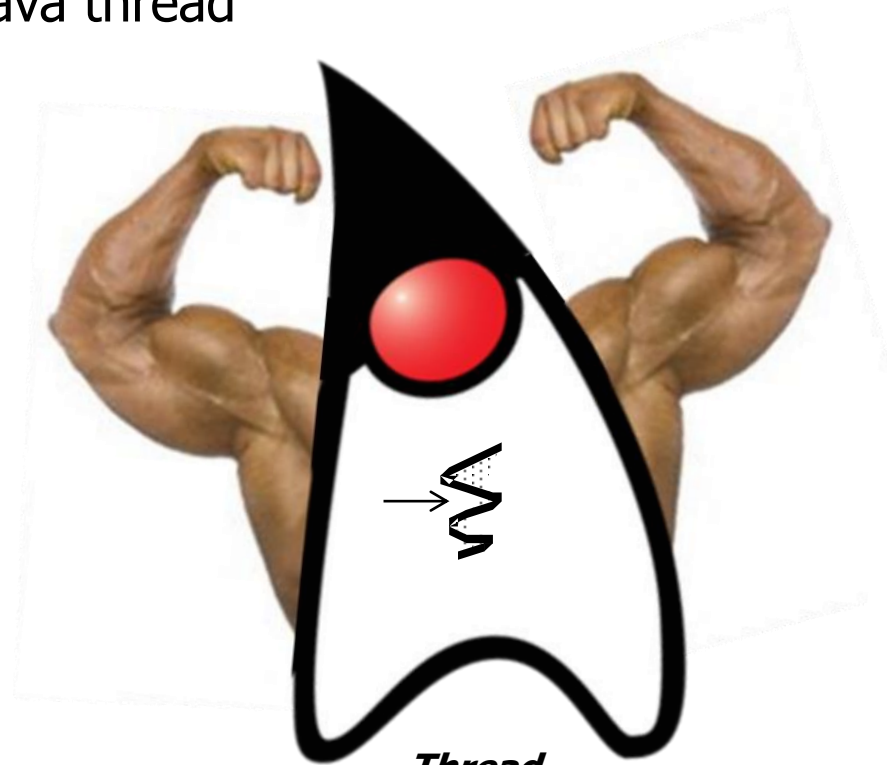
A "main" `ForkJoinTask` begins execution when it is explicitly submitted to a `ForkJoinPool`, or, if not already engaged in a `ForkJoin` computation, commenced in the `ForkJoinPool.commonPool()` via `fork()`, `invoke()`, or related methods. Once started, it will usually in turn start other subtasks. As indicated by the name of this class, many programs using `ForkJoinTask` employ only methods `fork()` and `join()`, or derivatives such as `invokeAll`. However, this class also provides a number of other methods that can come into play in advanced usages, as well as extension mechanics that allow support of new forms of `fork/join` processing.

# Overview of the ForkJoinTask Class

- A ForkJoinTask is lighter weight than a Java thread



***ForkJoinTask***



***Thread***

e.g., it doesn't maintain its own run-time stack, registers, thread-local storage, etc.

# Overview of the ForkJoinTask Class

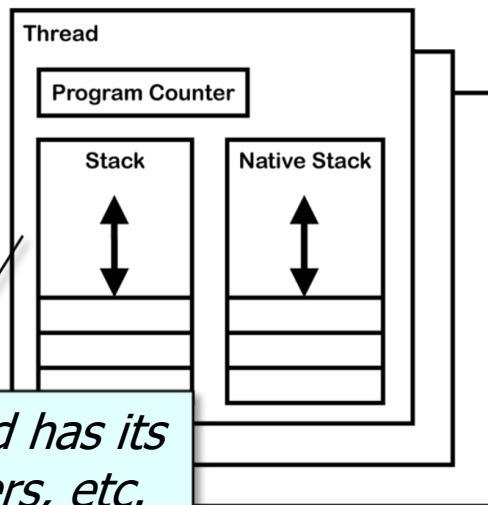
- A ForkJoinTask is lighter weight than a Java thread
- A large # of ForkJoinTasks can thus run in a small # of worker threads in a fork-join pool





# Overview of the ForkJoinTask Class

- A ForkJoinTask is lighter weight than a Java thread
- A large # of ForkJoinTasks can thus run in a small # of worker threads in a fork-join pool



*Each worker thread has its own stack, registers, etc.*



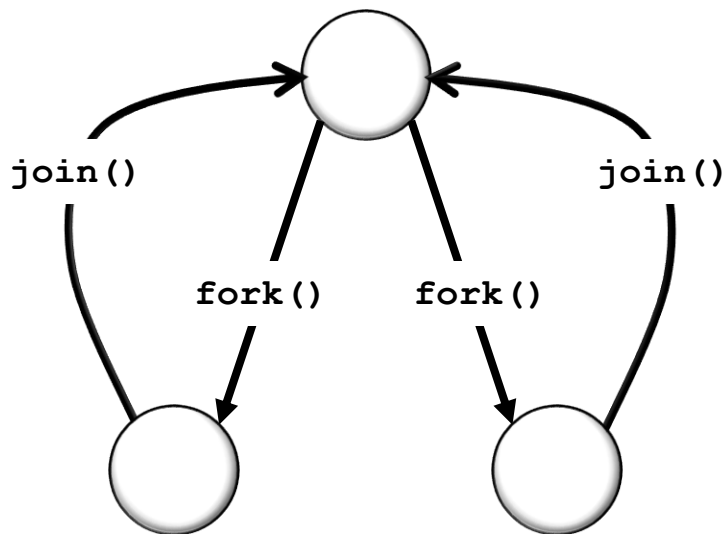
See [blog.jamesdbloom.com/JVMInternals.html](http://blog.jamesdbloom.com/JVMInternals.html)



# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging

*Parent ForkJoinTask*



*Child ForkJoinTasks*

ForkJoinTask  
<T>

**fork()** – Arranges to asynchronously execute this task in the appropriate pool

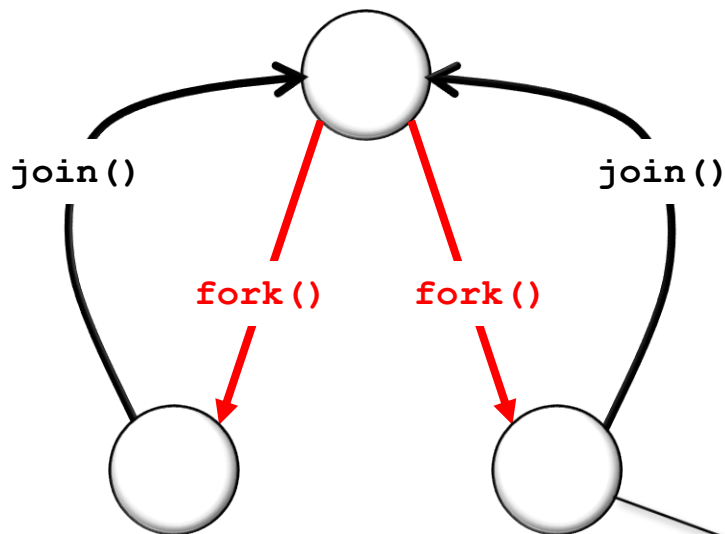
V

**join()** – Returns result of computation when it is done

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging

*Parent ForkJoinTask*



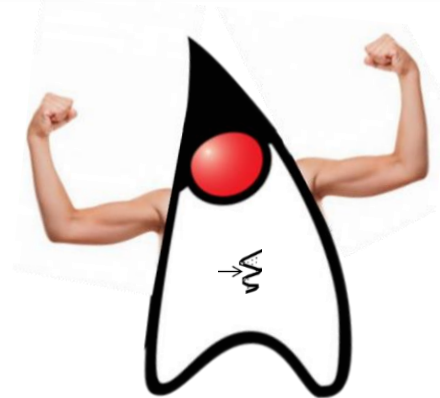
*Child ForkJoinTasks*

ForkJoinTask  
<T>

**fork()** – Arranges to asynchronously execute this task in the appropriate pool

V

**join()** – Returns result of computation when it is done

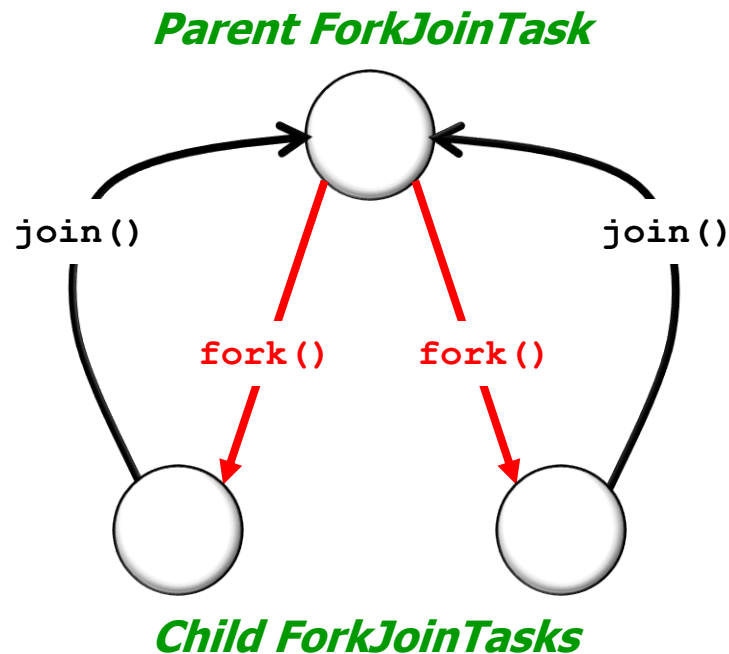


**ForkJoinTask**

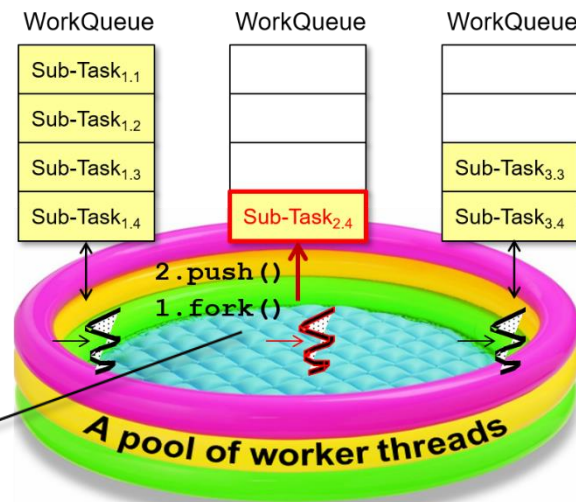
*fork() is a lightweight variant of Thread.start() that creates a child ForkJoinTask*

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging



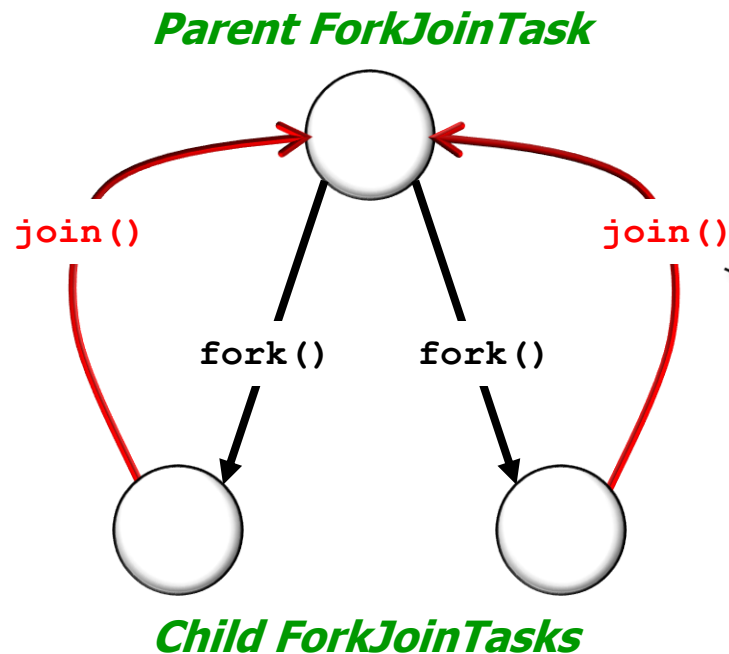
ForkJoinTask <T>	<b>fork()</b> – Arranges to asynchronously execute this task in the appropriate pool
V	<b>join()</b> – Returns result of computation when it is done



*fork() doesn't run the task, but places it on a work queue in the calling worker thread*

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging



ForkJoinTask  
<T>

**fork()** – Arranges to asynchronously execute this task in the appropriate pool

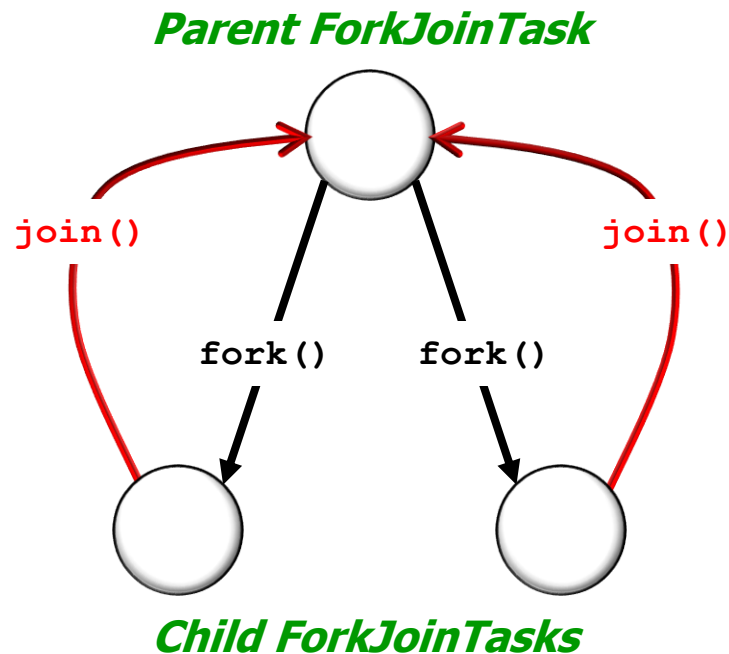
V

**join()** – Returns result of computation when it is done

*join() returns the result of a child task to the parent task that forked it*

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging

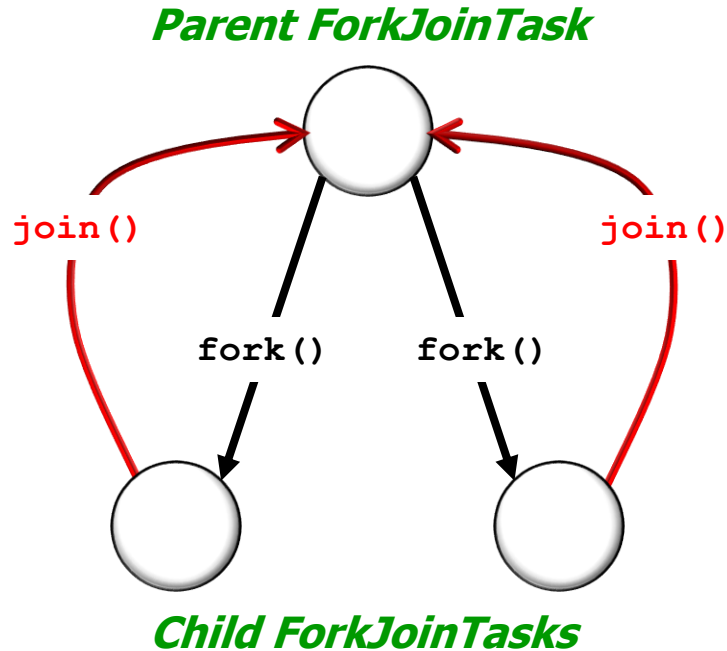


ForkJoinTask <T>	<b>fork()</b> – Arranges to asynchronously execute this task in the appropriate pool
V	<b>join()</b> – Returns result of computation when it is done

- Unlike Thread.join(), ForkJoinTask.join() doesn't simply block the calling thread

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging



ForkJoinTask <T>	<b>fork()</b> – Arranges to asynchronously execute this task in the appropriate pool
V	<b>join()</b> – Returns result of computation when it is done

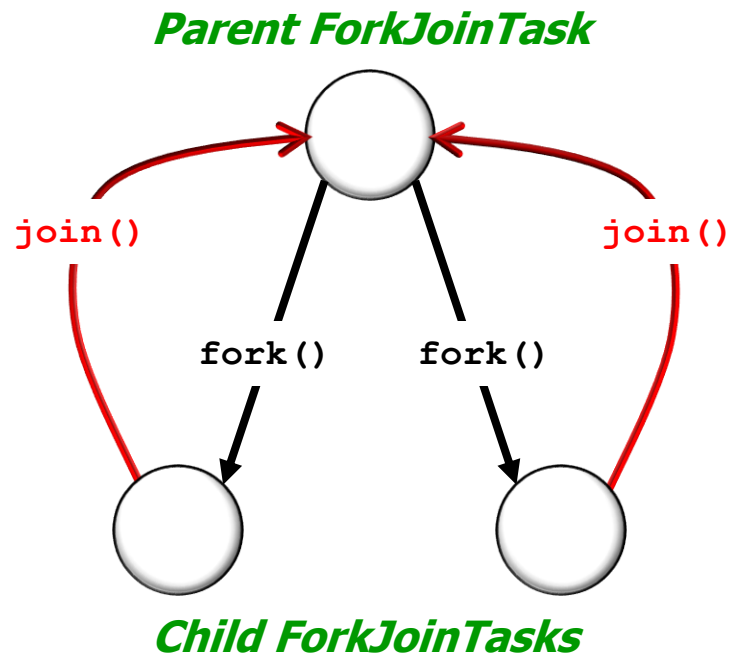
- Unlike Thread.join(), ForkJoinTask.join() doesn't simply block the calling thread
  - It uses a worker thread to run tasks



It “pitches in” via the “Collaborative Jiffy Lube” model of processing!

# Overview of the ForkJoinTask Class

- A ForkJoinTask has two methods that control parallel processing/merging



ForkJoinTask <T>	<b>fork()</b> – Arranges to asynchronously execute this task in the appropriate pool
V	<b>join()</b> – Returns result of computation when it is done

- Unlike Thread.join(), ForkJoinTask.join() doesn't simply block the calling thread
  - It uses a worker thread to run tasks
  - When a worker thread encounters a join() it processes other tasks until it notices the target sub-task is done



# Overview of the ForkJoinTask Class

- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks

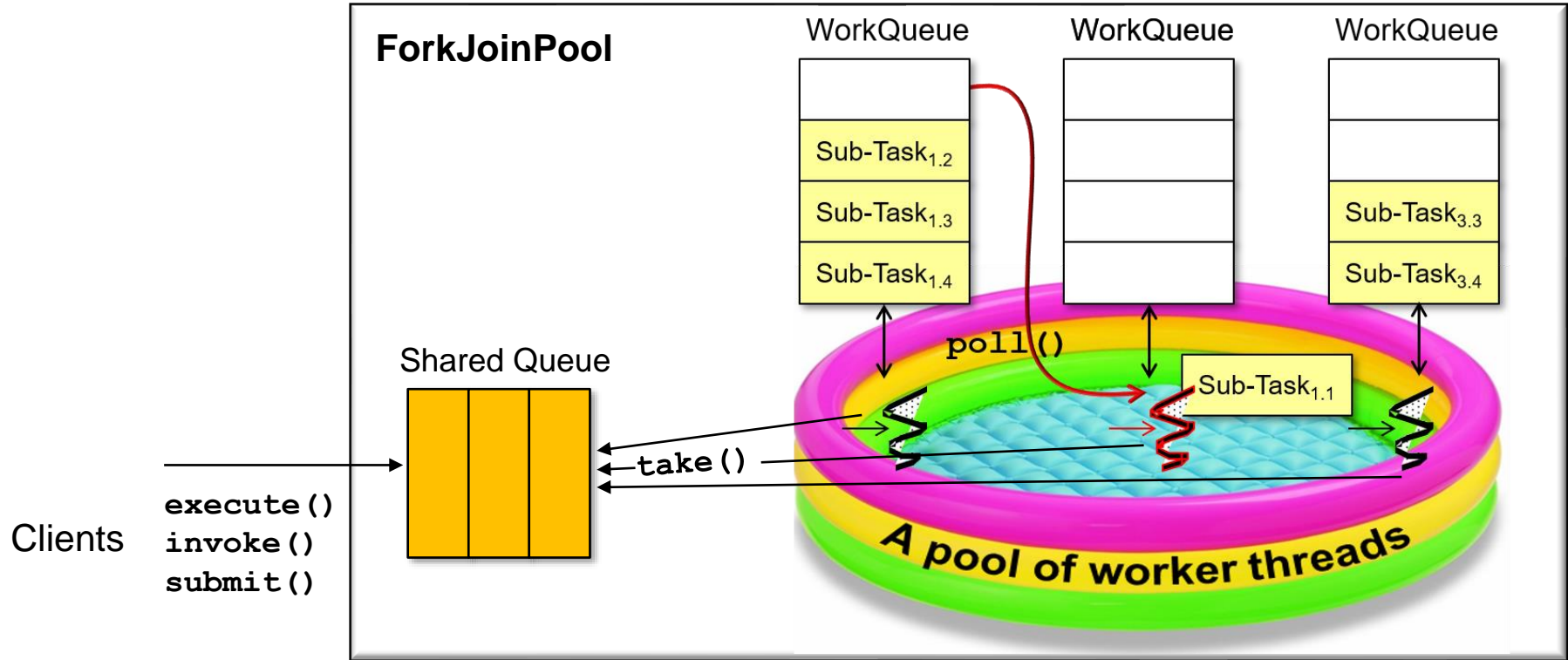
void	<a href="#"><u>execute(ForkJoinTask&lt;T&gt;)</u></a> – Arrange async execution
T	<a href="#"><u>invoke(ForkJoinTask&lt;T&gt;)</u></a> – Performs the given task, returning its result upon completion
<a href="#"><u>ForkJoinTask</u></a> <a href="#"><u>&lt;T&gt;</u></a>	<a href="#"><u>submit(ForkJoinTask)</u></a> – Submits a ForkJoinTask for execution, returns a future



See upcoming lesson on “*The Java Fork-Join Pool: Key Methods in ForkJoinPool*”

# Overview of the ForkJoinTask Class

- Clients insert new fork-join tasks onto a fork-join pool's shared queue, which feeds "work-stealing" queues managed by worker threads



See [en.wikipedia.org/wiki/Work\\_stealing](https://en.wikipedia.org/wiki/Work_stealing)

# Overview of the ForkJoinTask Class

- Clients insert new fork-join tasks onto a fork-join pool's shared queue, which feeds "work-stealing" queues managed by worker threads
- The goal of "work-stealing" is to maximize processor core utilization



---

# End of the Java ForkJoinTask Class