The Java Fork-Join Pool

Introduction

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel

Overview of the Java Fork-Join Pool Computation Model

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high-performance, fine-grained task execution framework for Java data parallelism.

Class ForkJoinPool

java.lang.Object
    java.util.concurrent.AbstractExecutorService
        java.util.concurrent.ForkJoinPool

All Implemented Interfaces:

Executor, ExecutorService

```
public class ForkJoinPool
extends AbstractExecutorService
```

An ExecutorService for running ForkJoinTasks. A ForkJoinPool provides the entry point for submissions from non-ForkJoinTask clients, as well as management and monitoring operations.

A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

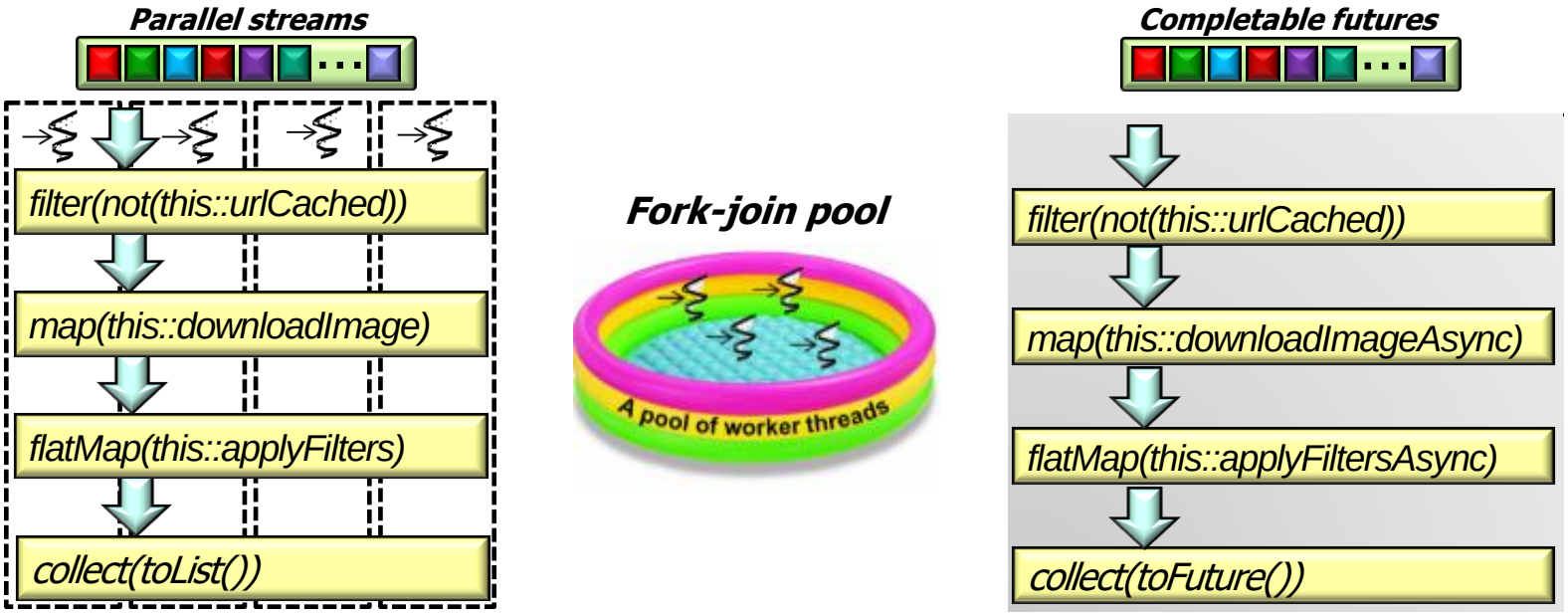
A static commonPool() is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).

For applications that require separate or custom pools, a ForkJoinPool may be constructed with a given target parallelism level; by default, equal to the number of available processors. The pool attempts to maintain enough active (or available) threads by dynamically adding, suspending, or resuming internal worker threads, even if some tasks are stalled waiting to join others. However, no such adjustments are guaranteed in the face of blocked I/O or other unmanaged synchronization. The nested ForkJoinPool.ManagedBlocker interface enables extension of the kinds of synchronization accommodated.

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

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool provides a high-performance, fine-grained task execution framework for Java data parallelism.
  - Its parallel computing engine is used by many higher-level frameworks.

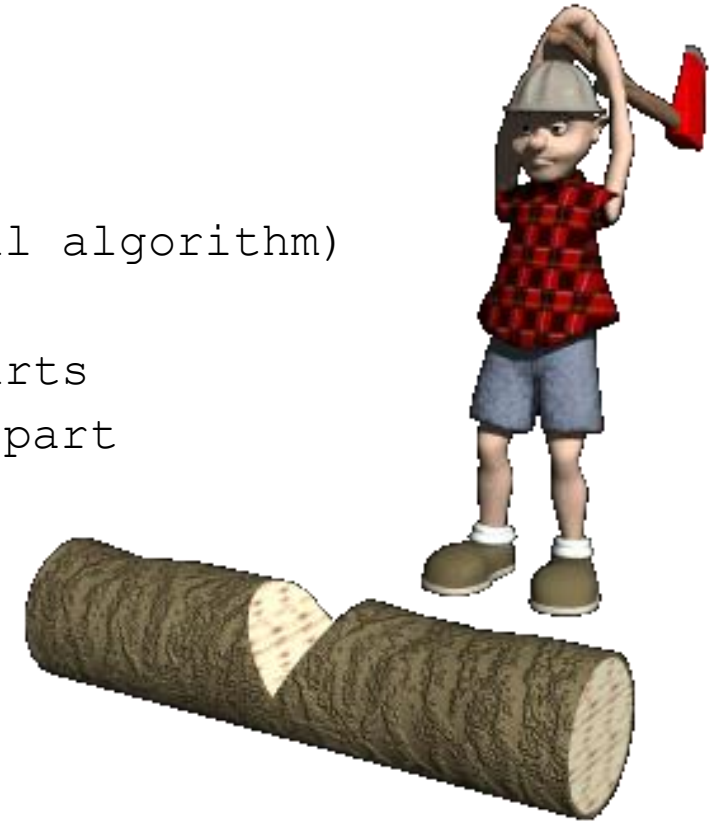


See www.infoq.com/interviews/doug-lea-fork-join

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."

```
Solve(problem)
  if problem is small enough
    solve problem directly (sequential algorithm)
  else
    split problem into independent parts
    fork new sub-tasks to solve each part
    join all sub-tasks
    compose result from sub-results
```

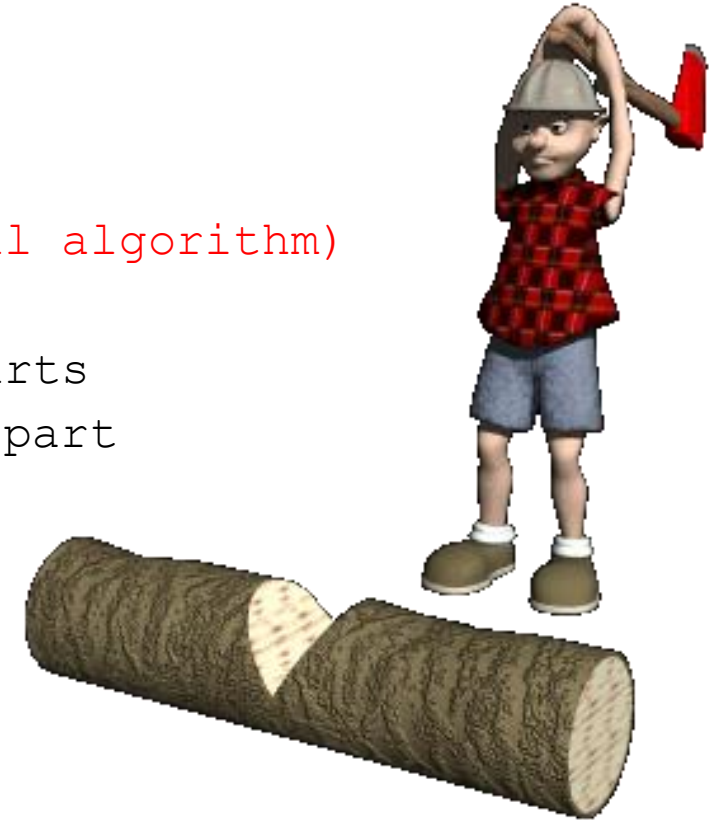


See en.wikipedia.org/wiki/Divide_and_conquer_algorithm

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."

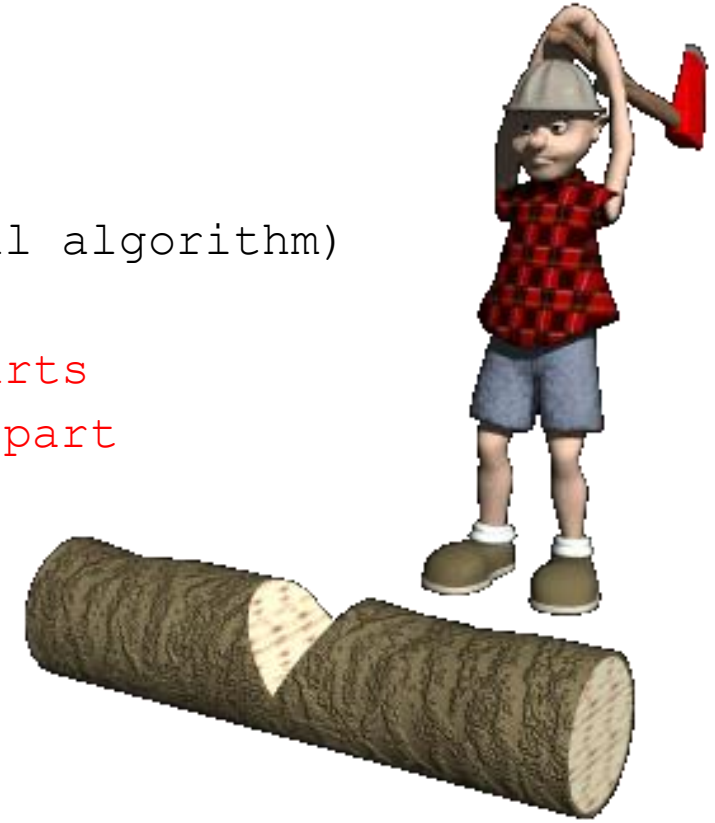
```
Solve(problem)
  if problem is small enough
    solve problem directly (sequential algorithm)
  else
    split problem into independent parts
    fork new sub-tasks to solve each part
    join all sub-tasks
    compose result from sub-results
```



Overview of the Java Fork-Join Pool Computation Model

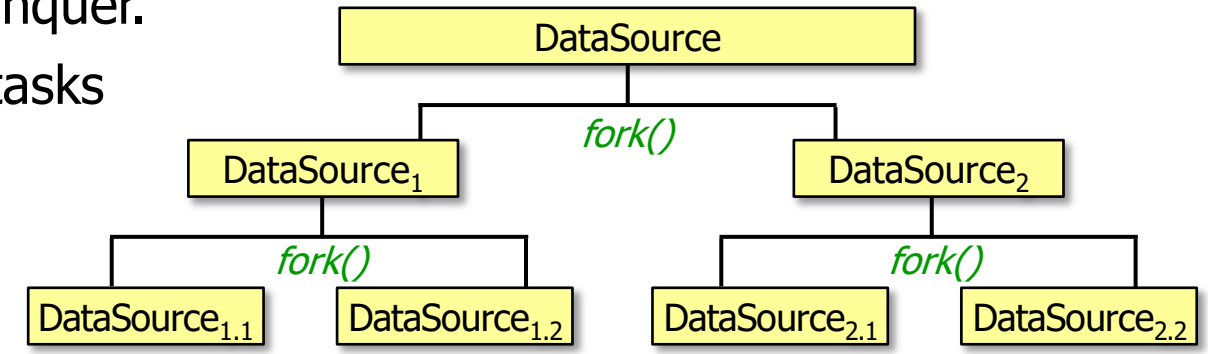
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."

```
Solve(problem)
  if problem is small enough
    solve problem directly (sequential algorithm)
  else
    split problem into independent parts
    fork new sub-tasks to solve each part
    join all sub-tasks
    compose result from sub-results
```



Overview of the Java Fork-Join Pool Computation Model

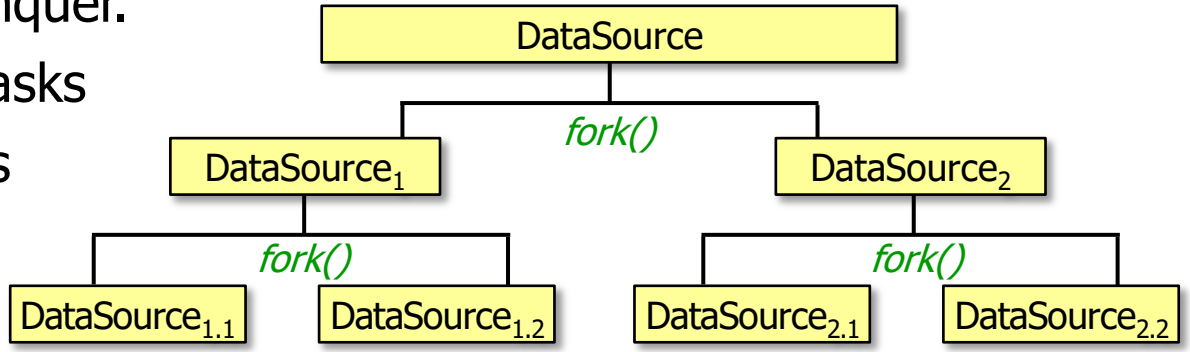
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks



See en.wikipedia.org/wiki/Fork-join_model

Overview of the Java Fork-Join Pool Computation Model

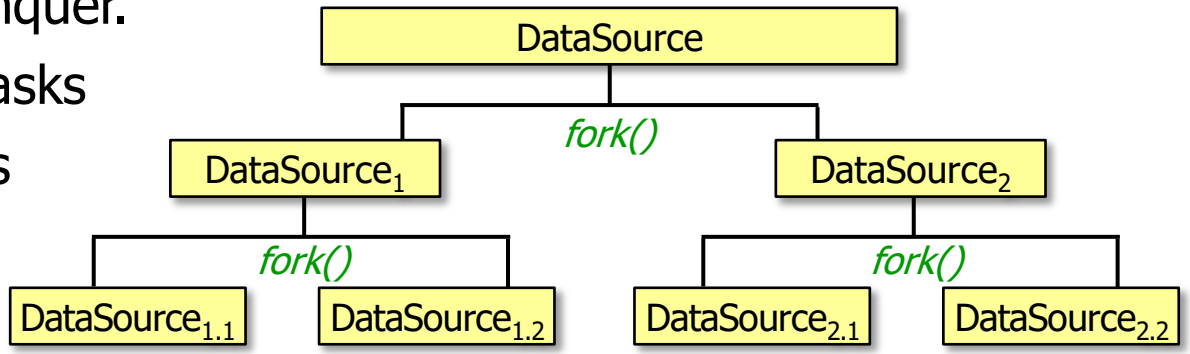
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
    - A task creates sub-tasks by fork()'ing



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

Overview of the Java Fork-Join Pool Computation Model

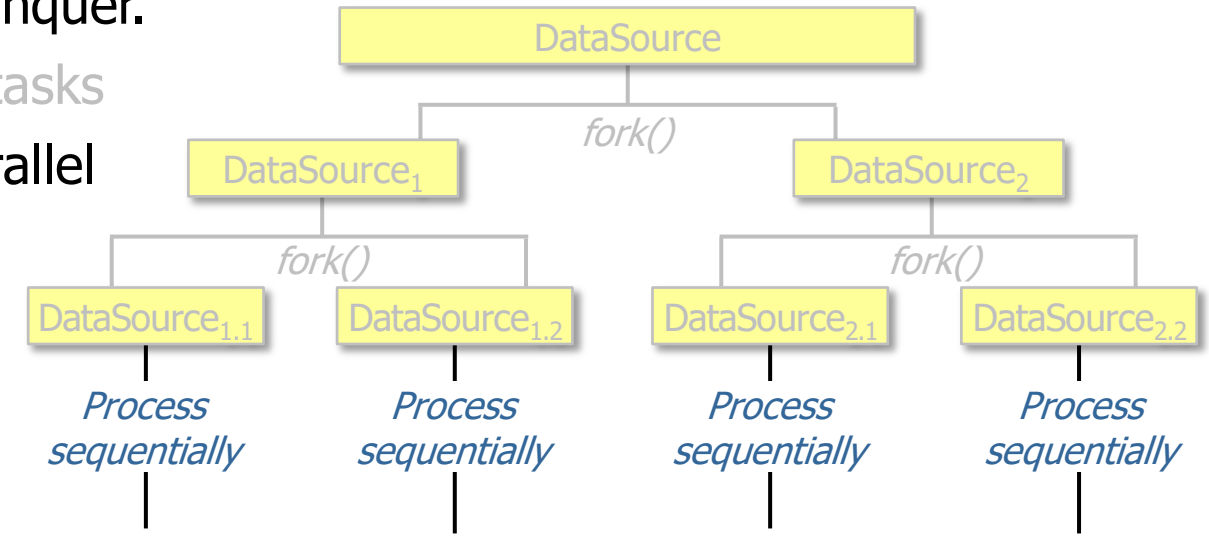
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
    - A task creates sub-tasks by fork()'ing



A (sub)-task only splits itself into (more) sub-tasks if the work is sufficiently big.

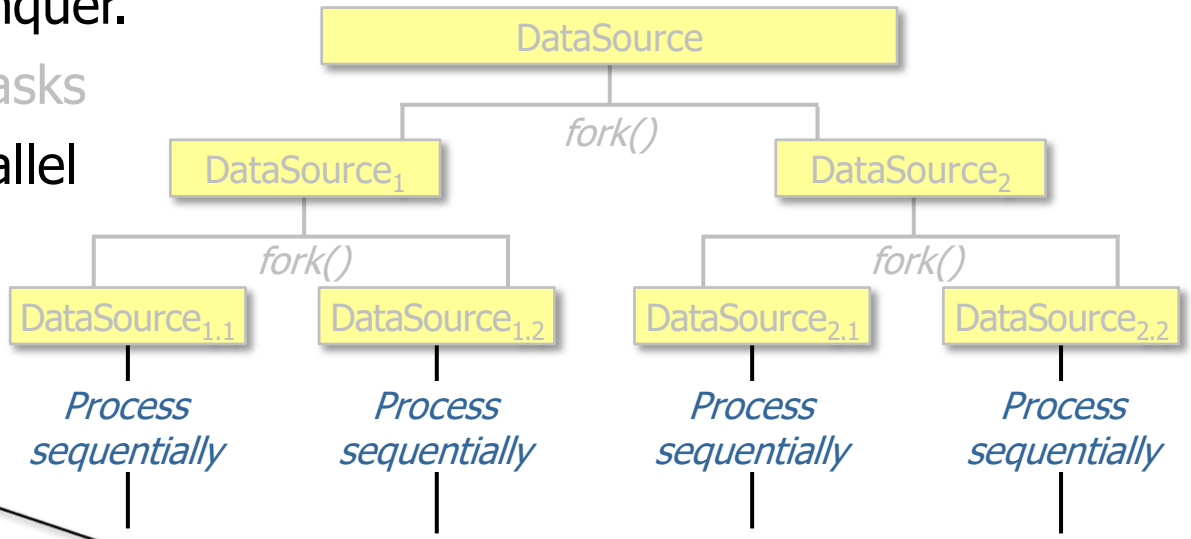
Overview of the Java Fork-Join Pool Computation Model

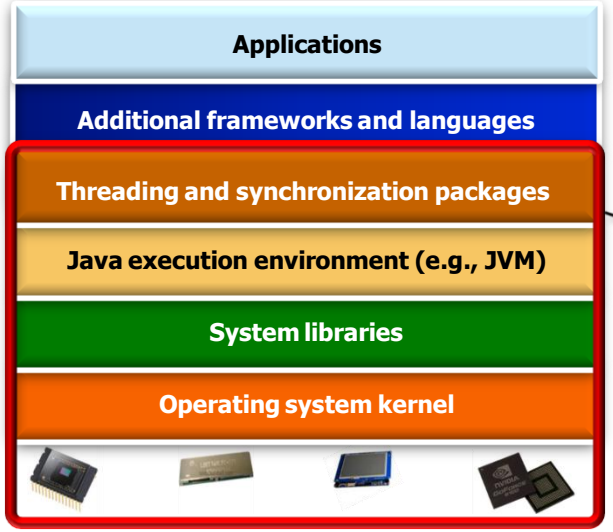
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel



Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel

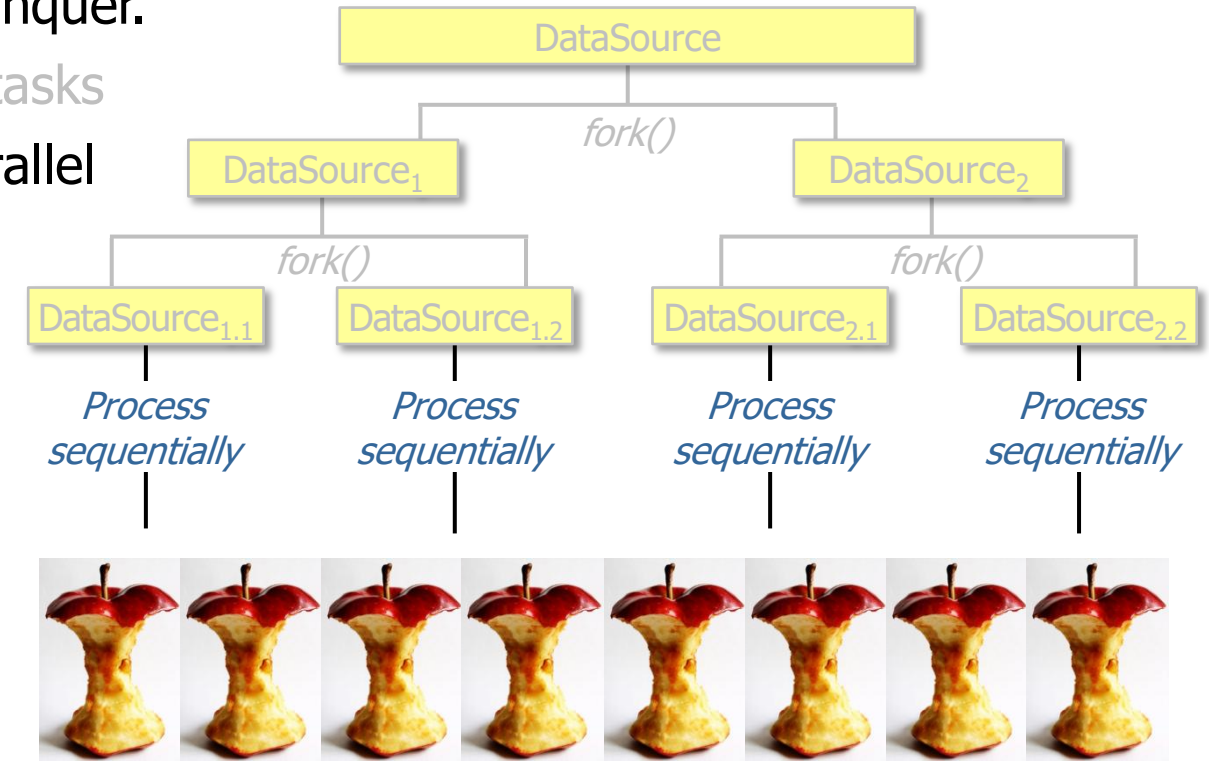




Implemented by fork-join framework, Java execution environment, OS, and hardware

Overview of the Java Fork-Join Pool Computation Model

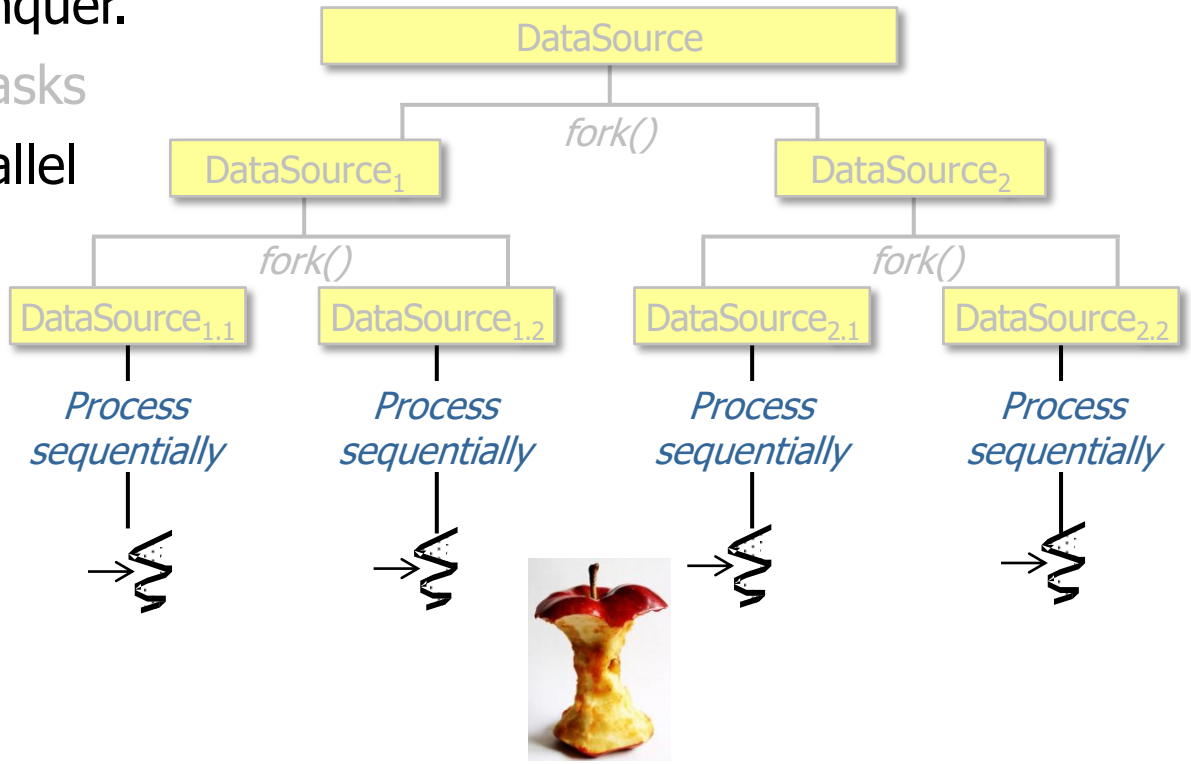
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
    - Sub-tasks run in parallel on different cores.



The more cores, the better the performance, depending on certain characteristics.

Overview of the Java Fork-Join Pool Computation Model

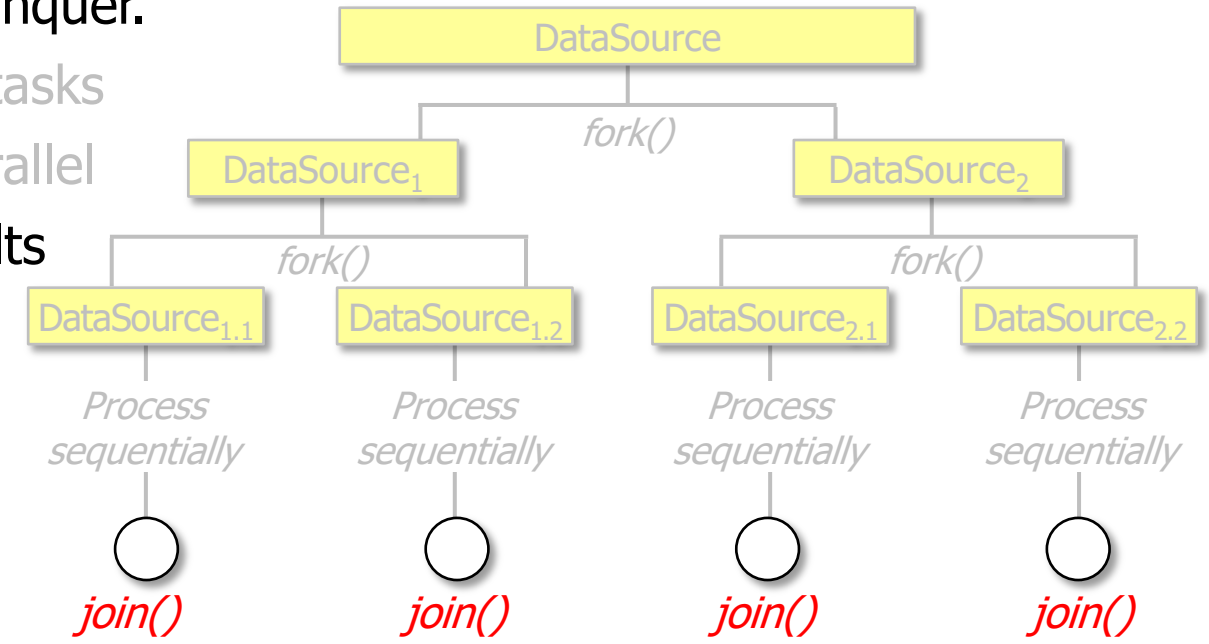
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
    - Sub-tasks run in parallel on different cores.
    - Sub-tasks can also run concurrently in different threads on a single core.



This configuration may not enhance performance unless sub-tasks are I/O bound.

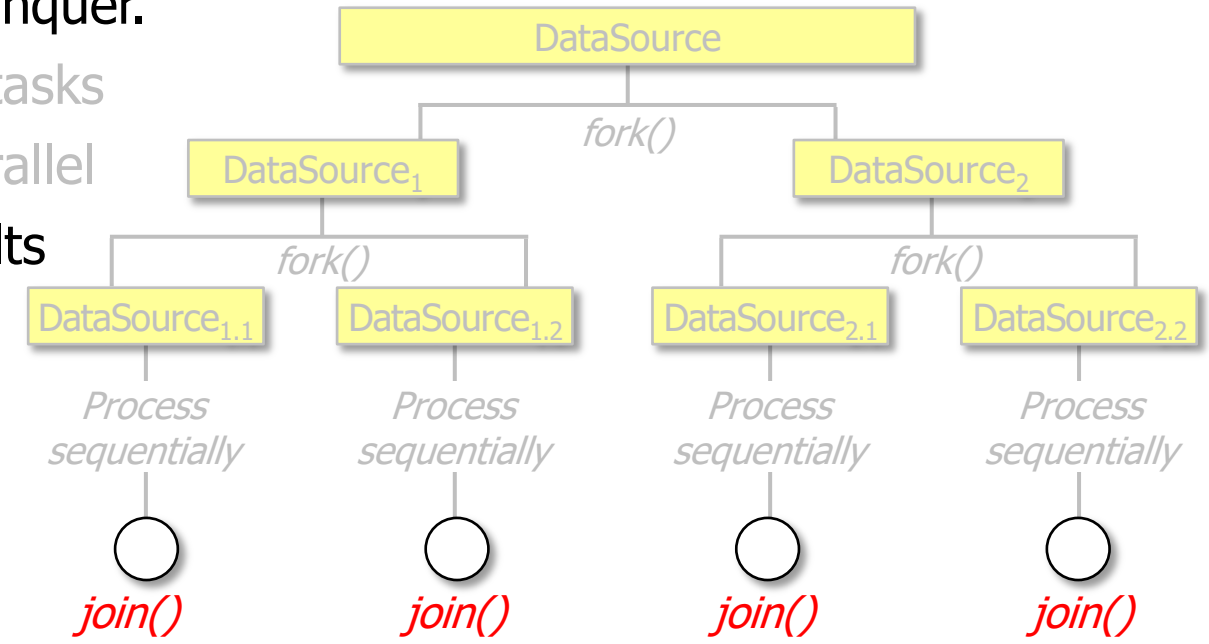
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results



Overview of the Java Fork-Join Pool Computation Model

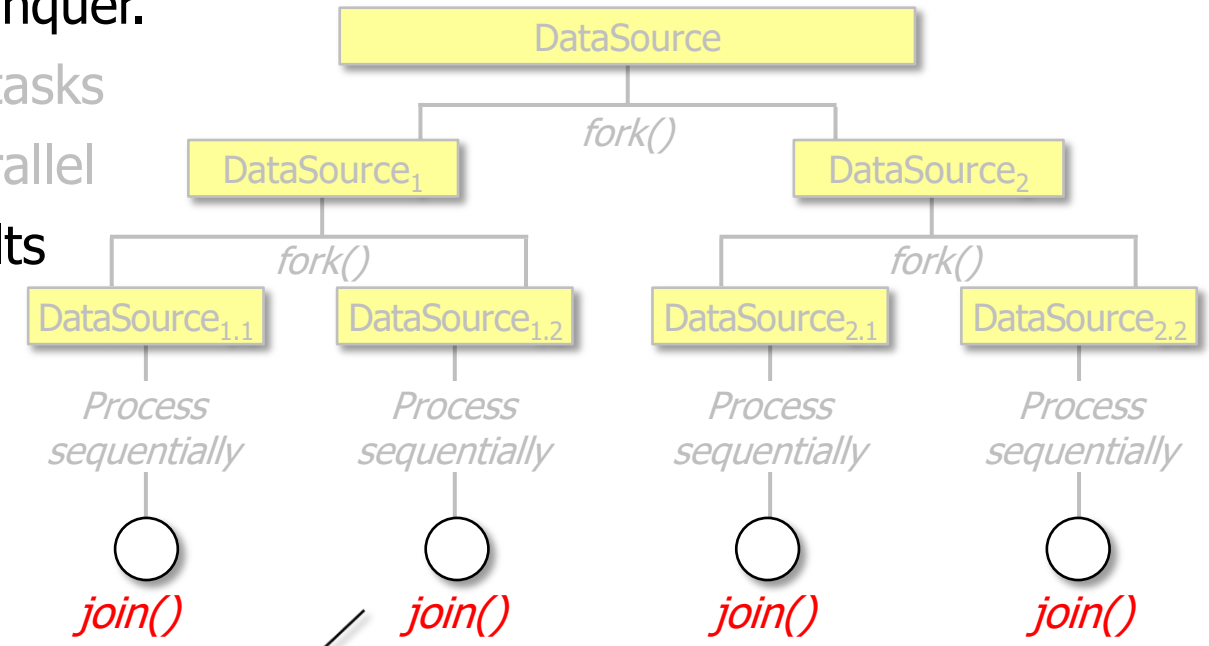
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - join() waits for a sub-task to finish



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

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - join() waits for a sub-task to finish

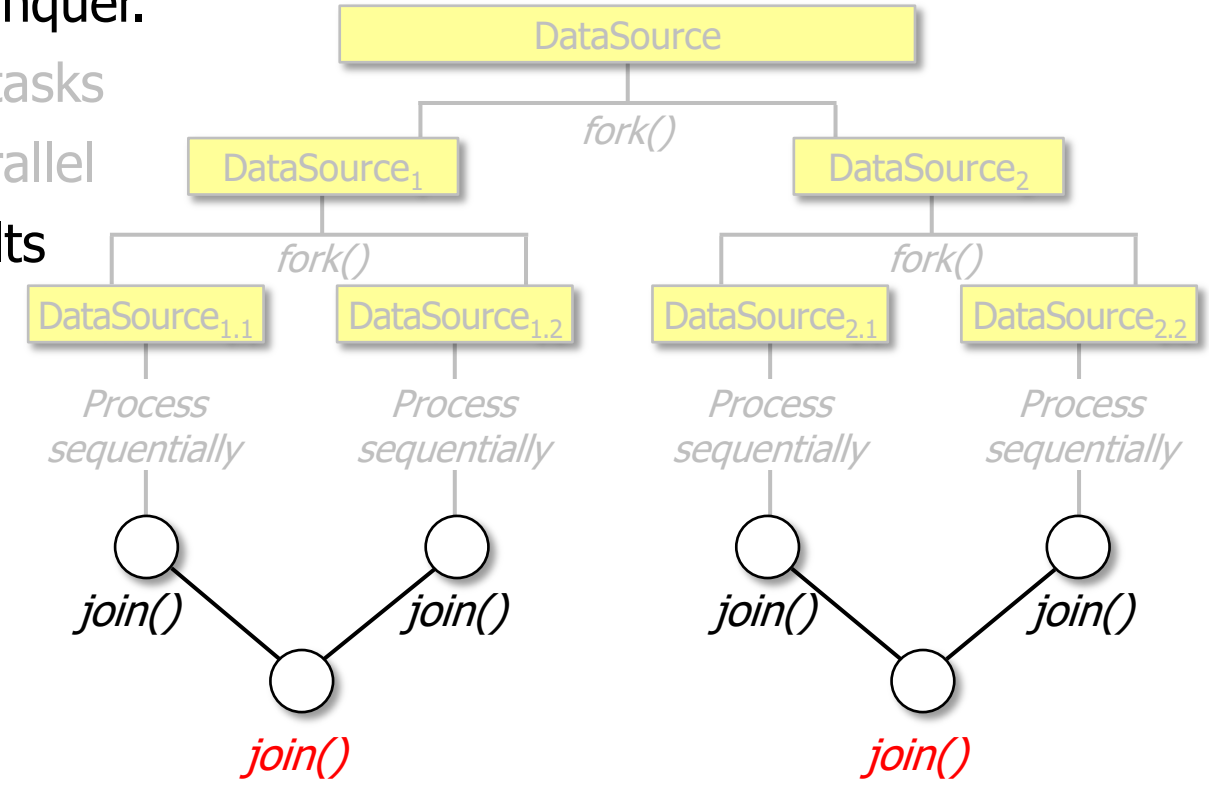


join() also plays a role in executing sub-tasks.

See upcoming lesson on *The Java Fork-Join Pool: Key Methods in ForkJoinTask*

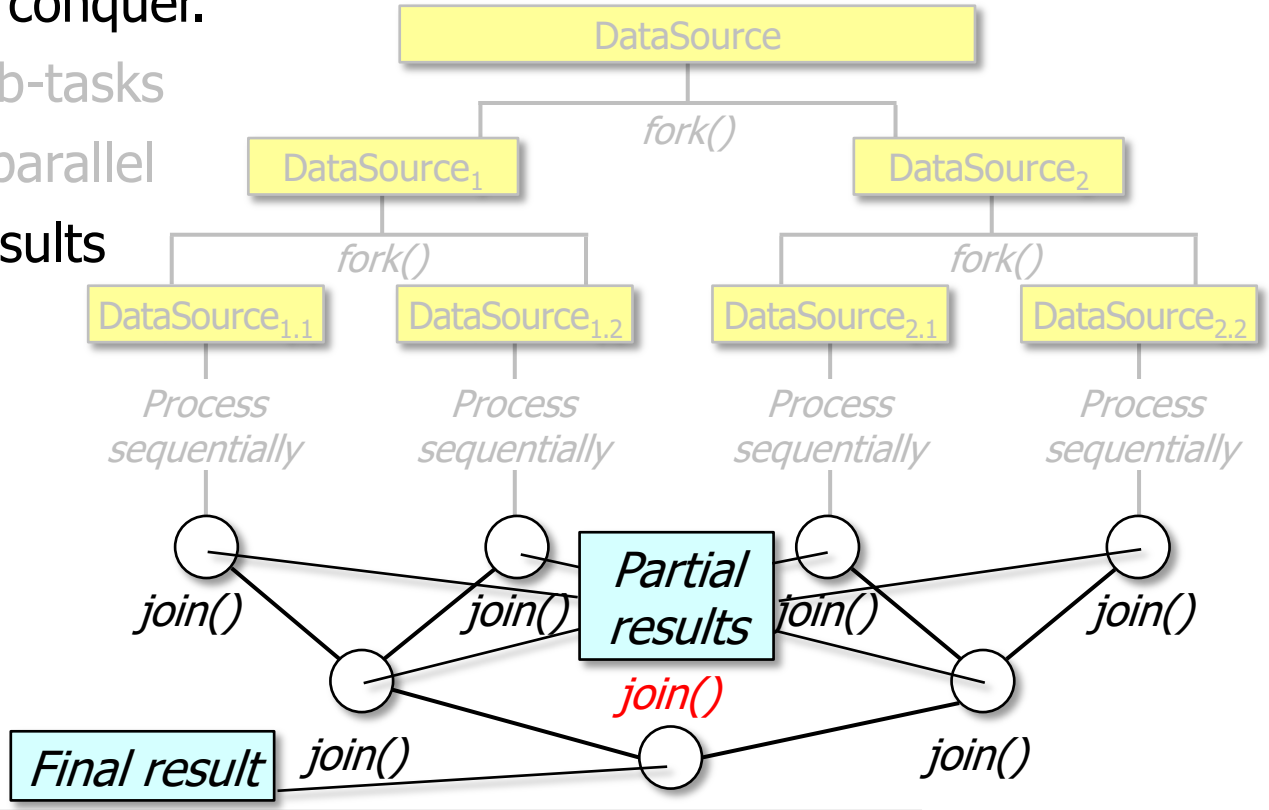
Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - join() waits for a sub-task to finish
      - and merges the results



Overview of the Java Fork-Join Pool Computation Model

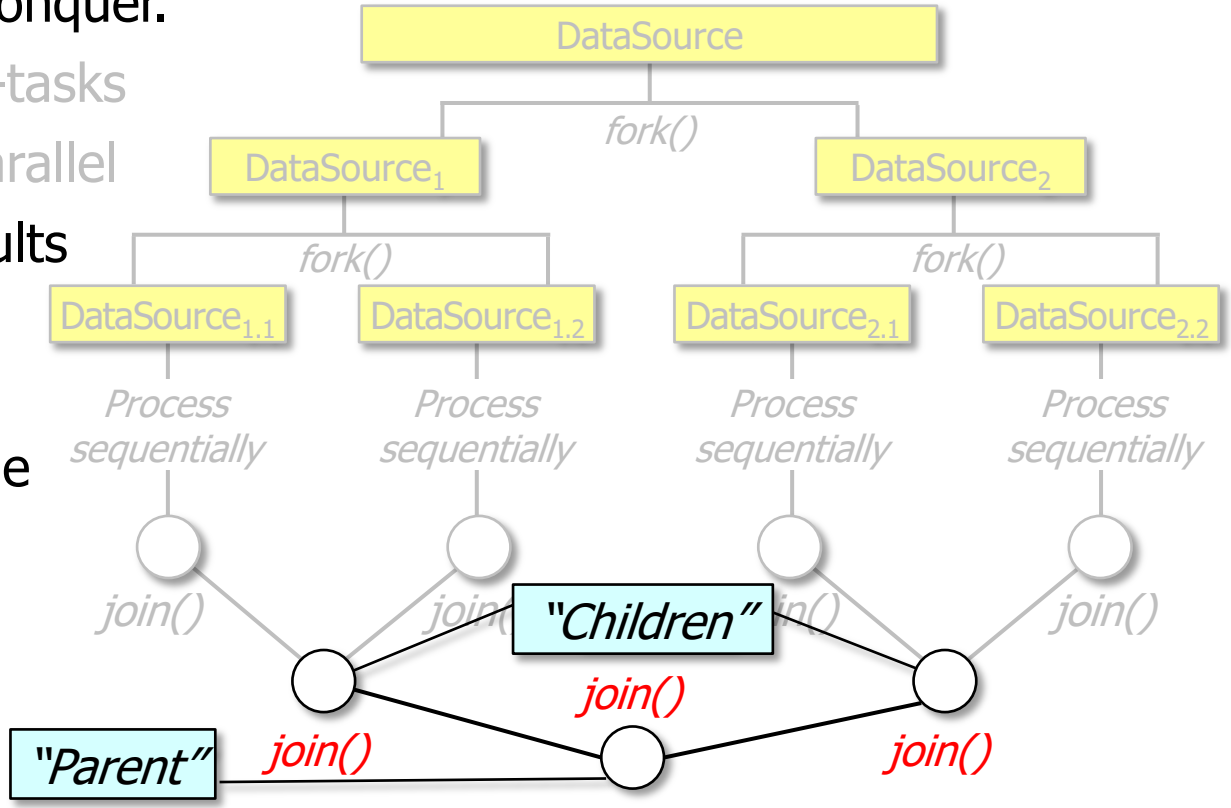
- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - join() waits for a sub-task to finish
      - and merges the results



Partial (sub-)results are merged into a final result.

Overview of the Java Fork-Join Pool Computation Model

- The fork-join pool supports a style of parallel programming that solves problems by "divide and conquer."
  - Splitting a task into sub-tasks
  - Applying sub-tasks in parallel
  - Combining sub-task results
    - join() waits for a sub-task to finish
    - join() occurs in a single thread at each level



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

The End

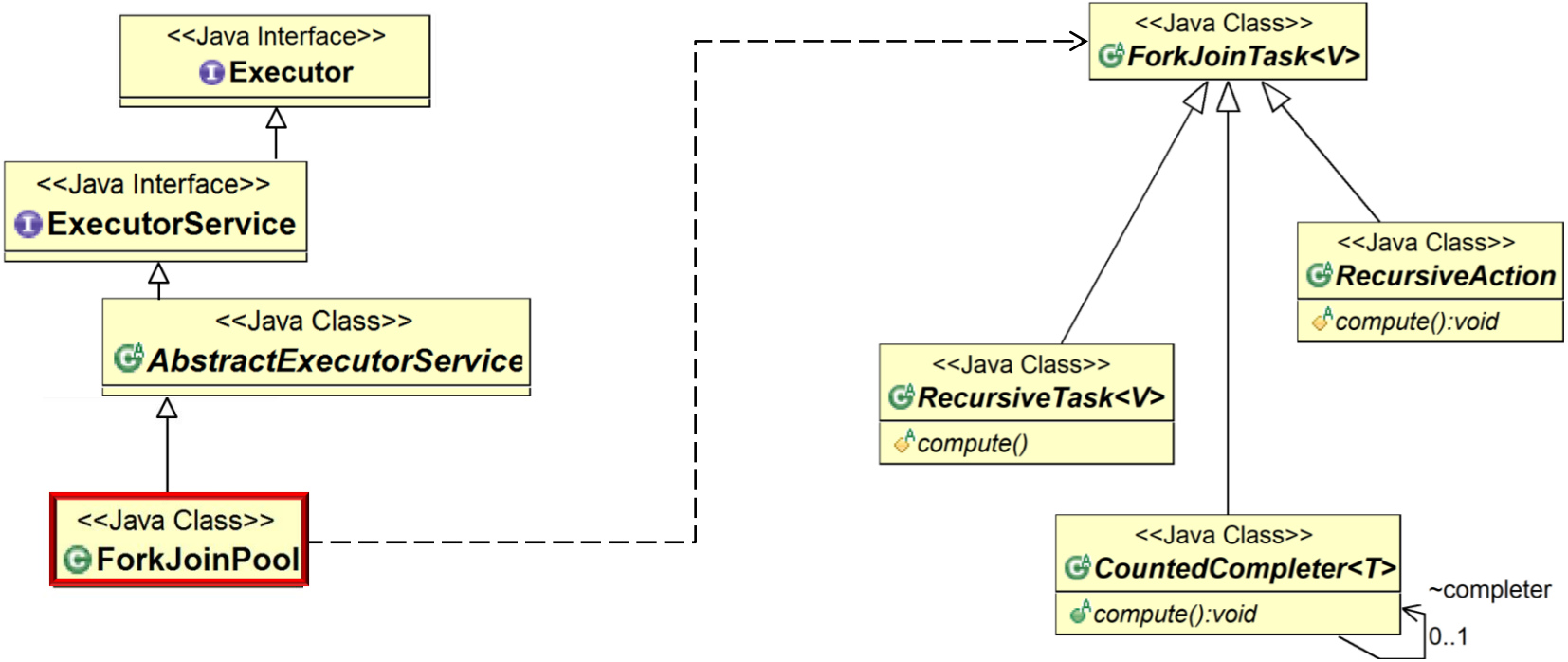
The Java Fork-Join Pool

The ForkJoinPool Class

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

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



Overview of the ForkJoinPool Class

Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface.

Class ForkJoinPool

java.lang.Object
    java.util.concurrent.AbstractExecutorService
        java.util.concurrent.ForkJoinPool

All Implemented Interfaces:

Executor, ExecutorService

```
public class ForkJoinPool
extends AbstractExecutorService
```

An ExecutorService for running ForkJoinTasks. A ForkJoinPool provides the entry point for submissions from non-ForkJoinTask clients, as well as management and monitoring operations.

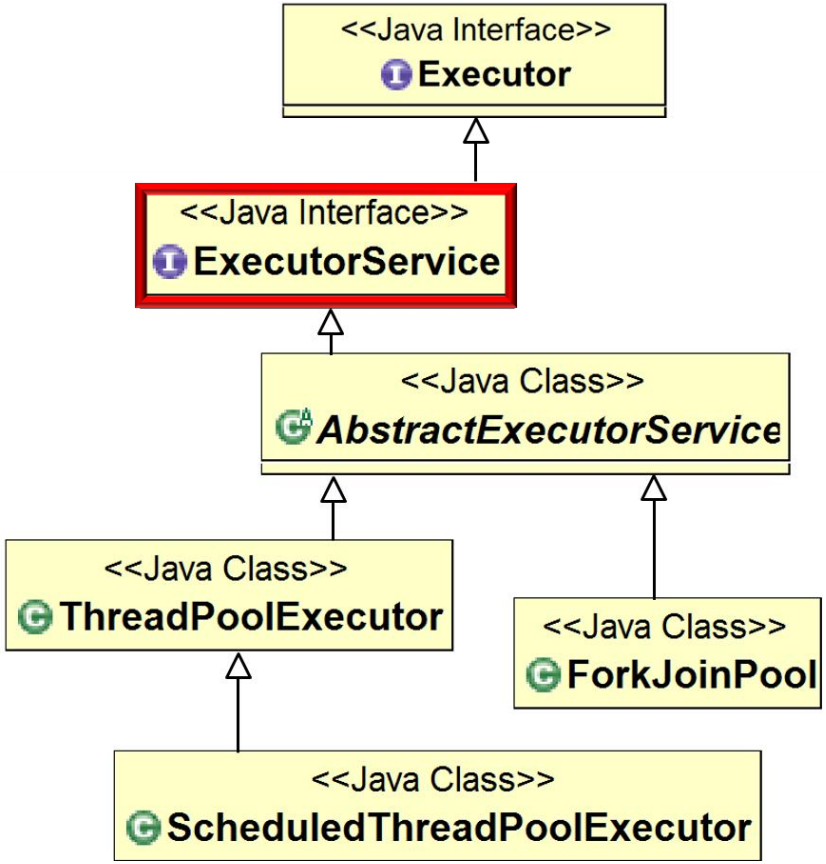
A ForkJoinPool differs from other kinds of ExecutorService mainly by virtue of employing *work-stealing*: all threads in the pool attempt to find and execute tasks submitted to the pool and/or created by other active tasks (eventually blocking waiting for work if none exist). This enables efficient processing when most tasks spawn other subtasks (as do most ForkJoinTasks), as well as when many small tasks are submitted to the pool from external clients. Especially when setting *asyncMode* to true in constructors, ForkJoinPools may also be appropriate for use with event-style tasks that are never joined.

A static commonPool() is available and appropriate for most applications. The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool. Using the common pool normally reduces resource usage (its threads are slowly reclaimed during periods of non-use, and reinstated upon subsequent use).

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

Overview of the ForkJoinPool Class

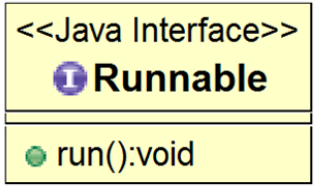
- ForkJoinPool implements the ExecutorService interface.
  - This interface is the basis for Java Executor framework subclasses.

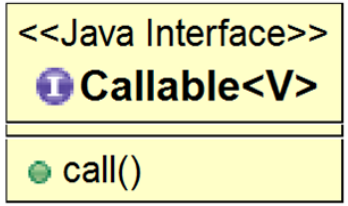


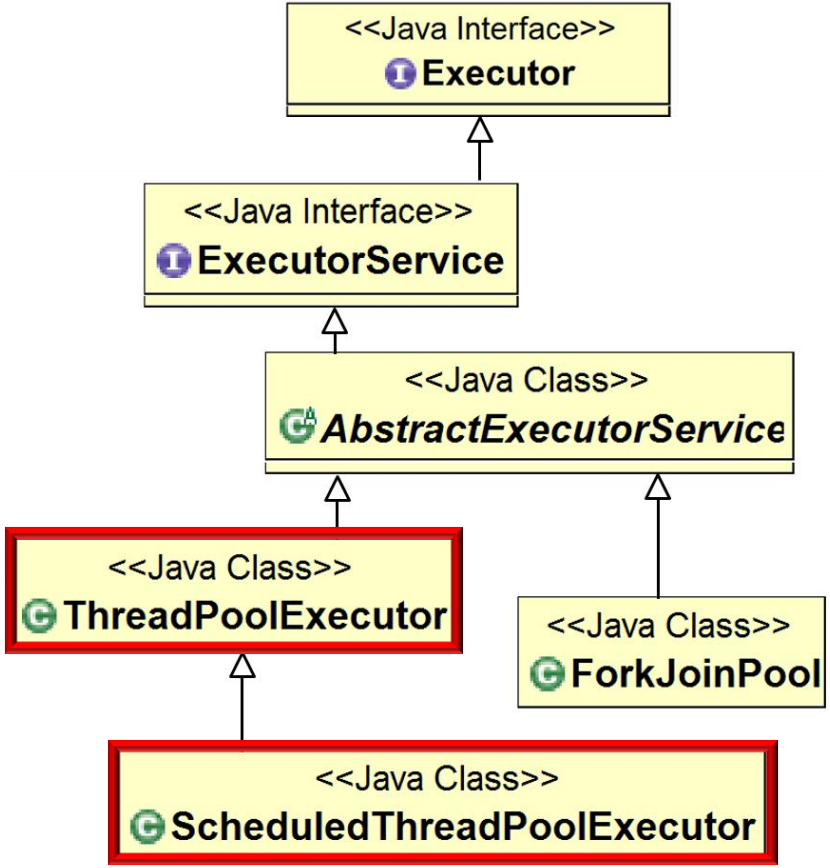
See docs.oracle.com/javase/tutorial/essential/concurrency/executors.html

Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface.
  - This interface is the basis for Java Executor framework subclasses.
  - Other implementations of Executor Service execute runnables or callables.

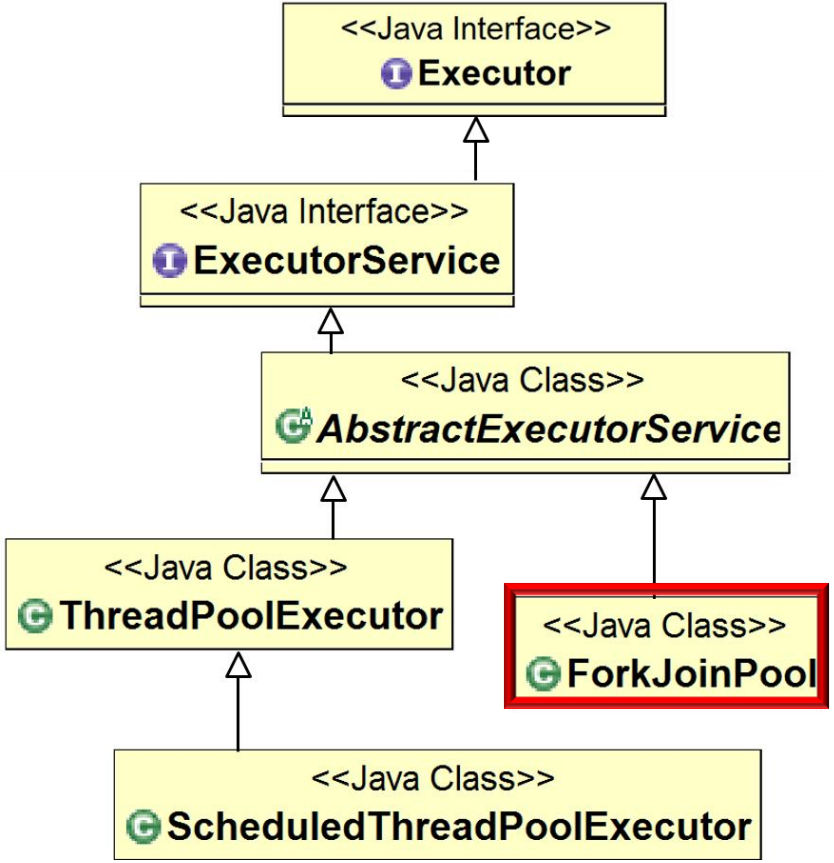






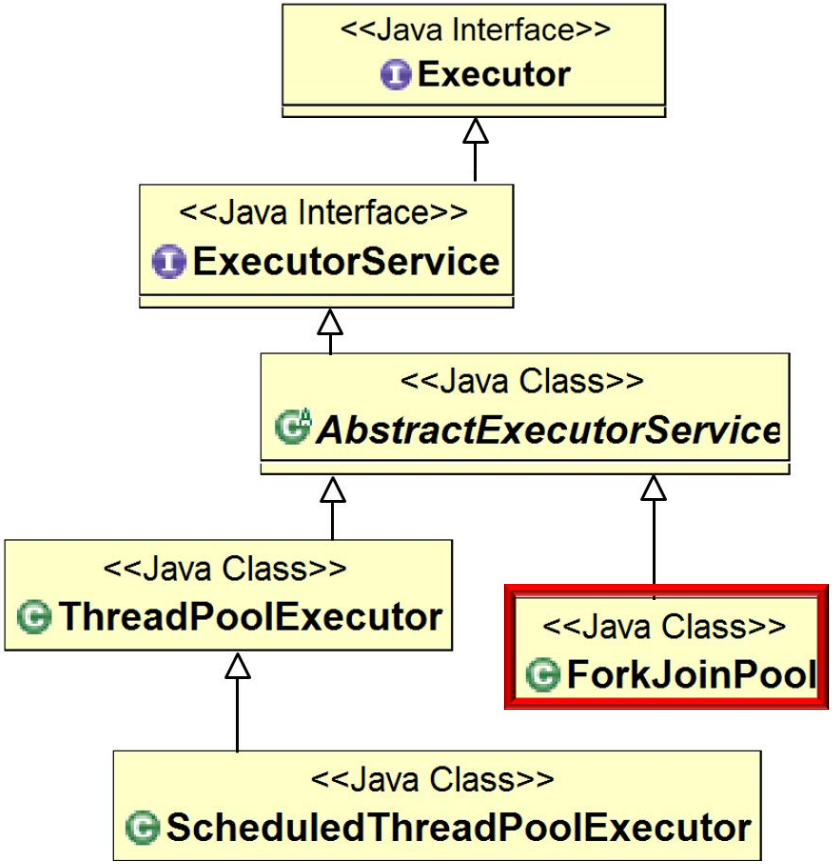
Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface.
  - This interface is the basis for Java Executor framework subclasses.
  - Other implementations of Executor Service execute runnables or callables.
  - In contrast, the ForkJoinPool executes ForkJoinTasks.



Overview of the ForkJoinPool Class

- ForkJoinPool implements the ExecutorService interface.
  - This interface is the basis for Java Executor framework subclasses.
  - Other implementations of Executor Service execute runnables or callables.
  - In contrast, the ForkJoinPool executes ForkJoinTasks.



It can also execute runnables and callables, but that's not its main purpose.

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool.

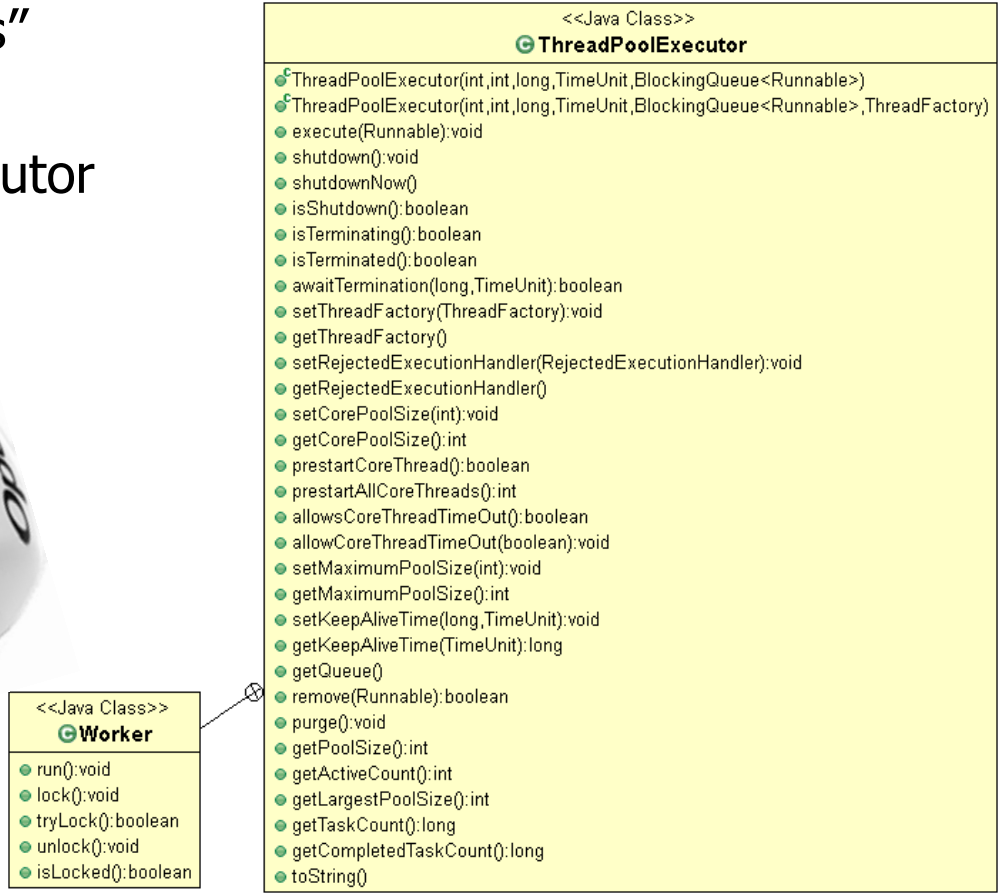
<<Java Class>>
ForkJoinPool

- ForkJoinPool()
- ForkJoinPool(int)
- ForkJoinPool(int,ForkJoinWorkerThreadFactory,UncaughtExceptionHandler,boolean)
- commonPool():ForkJoinPool
- invoke(ForkJoinTask<T>)
- execute(ForkJoinTask<?>):void
- execute(Runnable):void
- submit(ForkJoinTask<T>):ForkJoinTask<T>
- submit(Callable<T>):ForkJoinTask<T>
- submit(Runnable,T):ForkJoinTask<T>
- submit(Runnable):ForkJoinTask<?>
- invokeAll(Collection<Callable<T>>):List<Future<T>>
- shutdown():void
- shutdownNow():List<Runnable>
- isTerminated():boolean
- isTerminating():boolean
- isShutdown():boolean
- awaitTermination(long,TimeUnit):boolean

See www.youtube.com/watch?v=sq0MX3fHkro

Overview of the ForkJoinPool Class

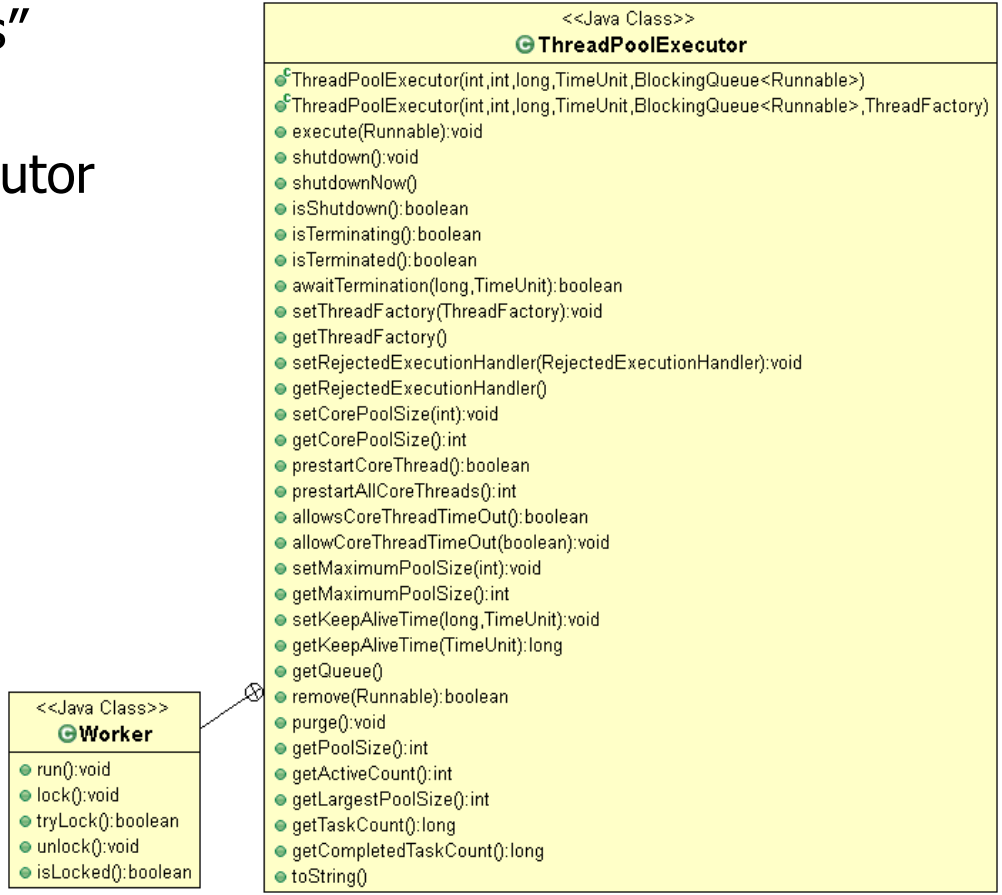
- There are (intentionally) few "knobs" that can control a ForkJoinPool.
  - Contrast with the ThreadPoolExecutor framework



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

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool.
  - Contrast with the ThreadPoolExecutor framework
    - CorePool size
    - MaxPool size
    - WorkQueue
    - KeepAliveTime
    - ThreadFactory
    - RejectedExecutionHandler



See dzone.com/articles/a-deep-dive-into-the-java-executor-service

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool.
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool.
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool

```
System.setProperty
        ("java.util.concurrent"
         + ".ForkJoinPool.common"
         + ".parallelism", "8");
```

# of desired threads

See lesson on "*The Java Fork-Join Pool: Overview of the Common Fork-Join Pool*"

Overview of the ForkJoinPool Class

- There are (intentionally) few "knobs" that can control a ForkJoinPool.
  - Contrast with the ThreadPoolExecutor framework
  - However, you *can* configure the size of the common fork-join pool

Interface ForkJoinPool.ManagedBlocker

Enclosing class:
`ForkJoinPool`

`public static interface` **`ForkJoinPool.ManagedBlocker`**

Interface for extending managed parallelism for tasks running in `ForkJoinPools`.

See lesson on "*The Java Fork-Join Pool: the ManagedBlocker Interface*"

The End

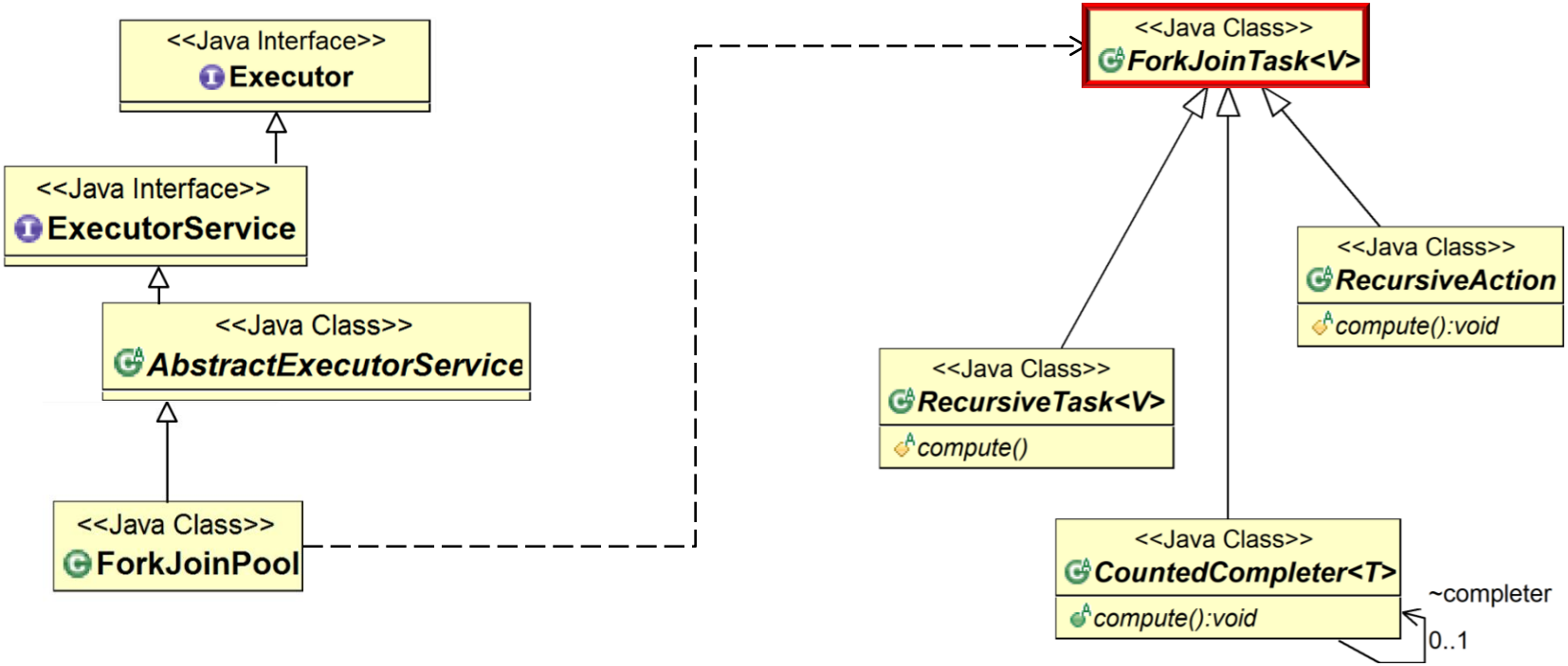
The Java Fork-Join Pool

The ForkJoinTask Class

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure and 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

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
```

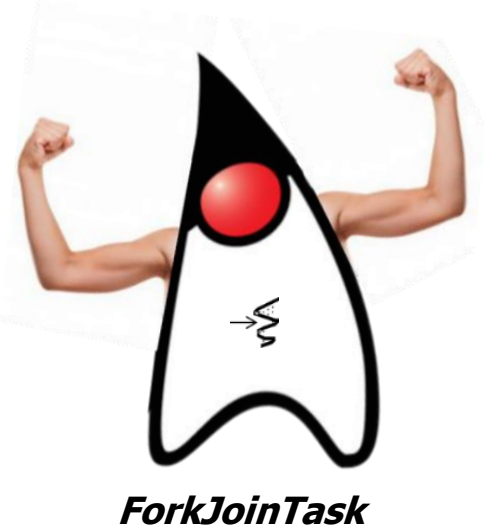
Abstract base class for tasks that run within a ForkJoinPool. A ForkJoinTask is a thread-like entity that is much lighter weight than a normal thread. Huge numbers of tasks and subtasks may be hosted by a small number of actual threads in a ForkJoinPool, at the price of some usage limitations.

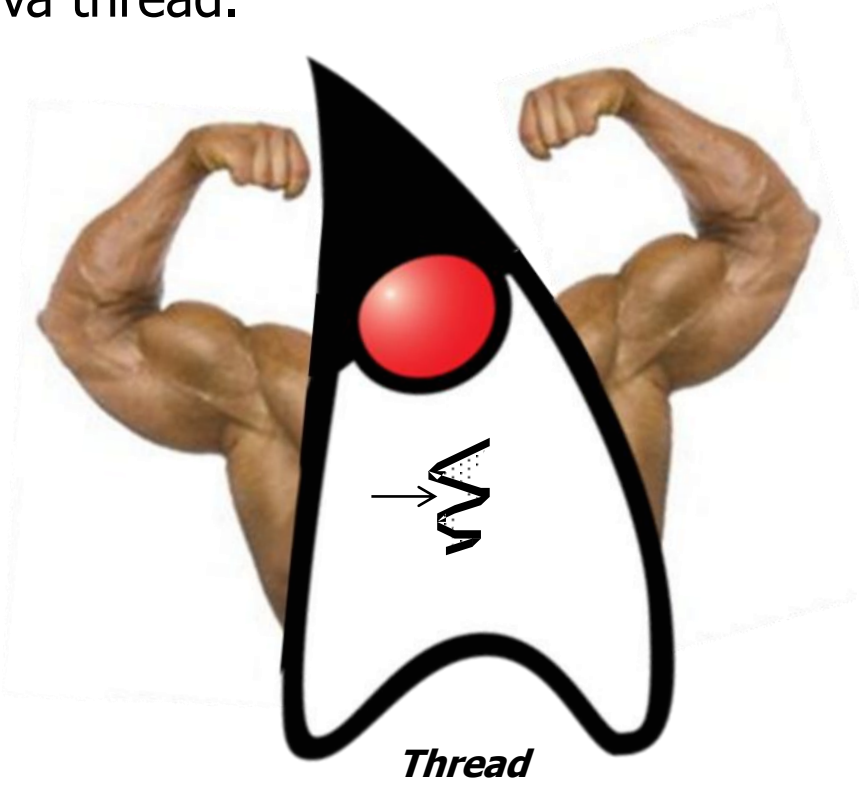
A "main" ForkJoinTask begins execution when it is explicitly submitted to a ForkJoinPool, or, if not already engaged in a ForkJoin computation, commenced in the ForkJoinPool.commonPool() via fork(), invoke(), or related methods. Once started, it will usually in turn start other subtasks. As indicated 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.

See www.dre.Vanderbilt.edu/~schmidt/PDF/DataParallelismInJava.pdf

Overview of the ForkJoinTask Class

- A ForkJoinTask is lighter weight than a Java thread.

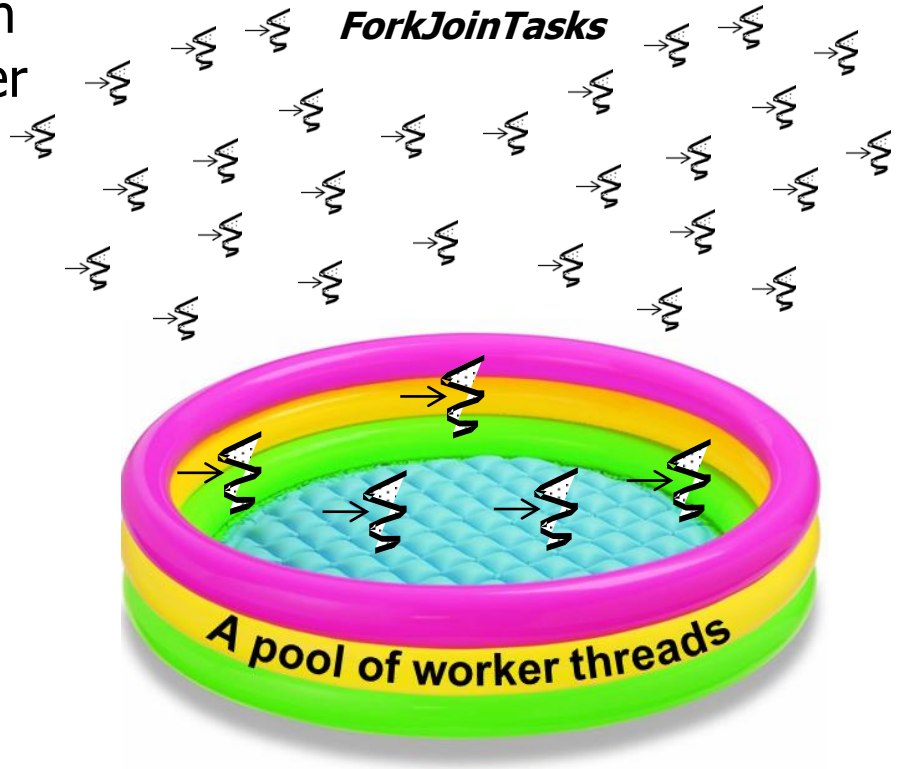




It doesn’t maintain its own run-time stack, registers, thread-local storage, etc.

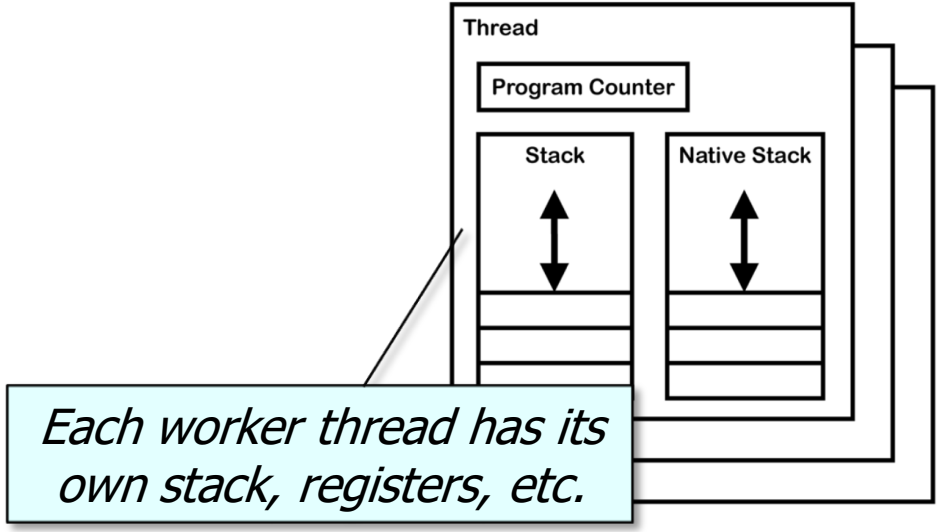
Overview of the ForkJoinTask Class

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



Overview of the ForkJoinTask Class

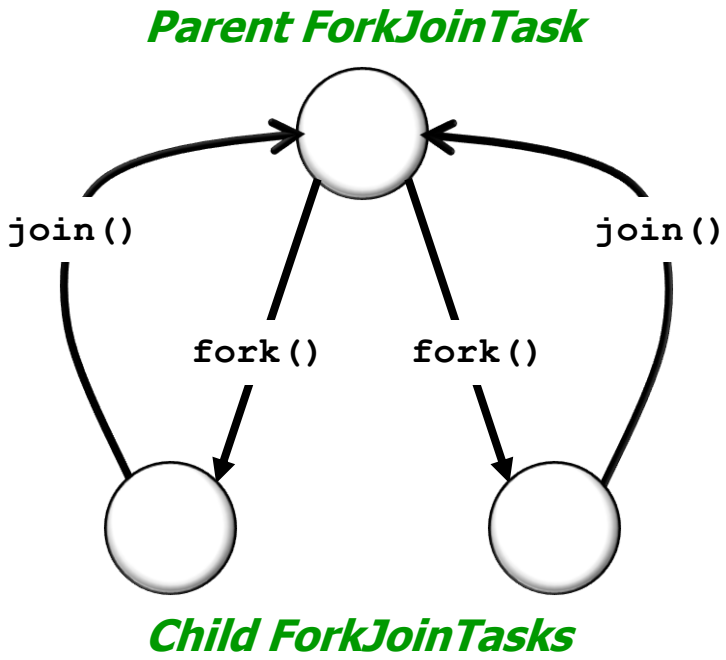
- A ForkJoinTask is lighter weight than a Java thread.
  - A large number of ForkJoinTasks can thus run in a small number of worker threads in a fork-join pool.



See blog.jamesdbloom.com/JVMInternals.html

Overview of the ForkJoinTask Class

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

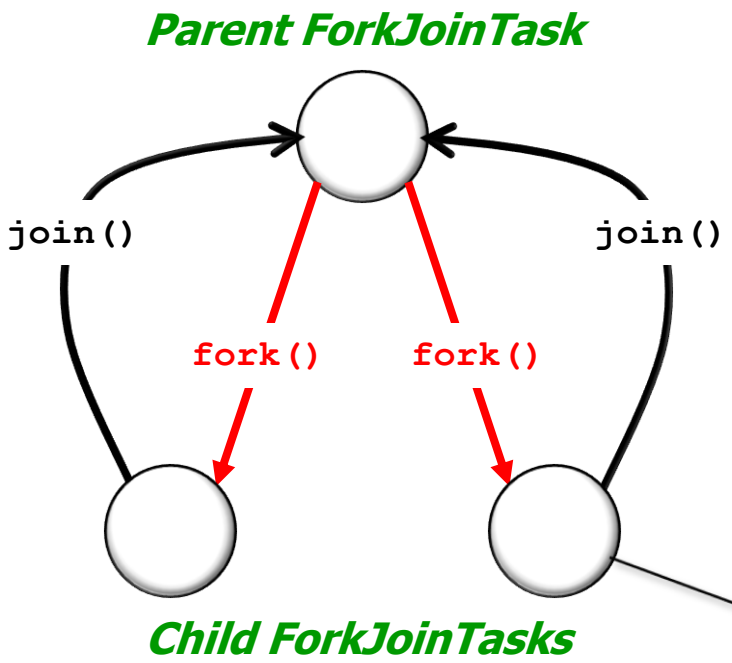


ForkJoinTask <T>	**fork**()—Arranges to asynchronously execute this task in the appropriate pool
V	**join**()—Returns result of computation when it is done

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



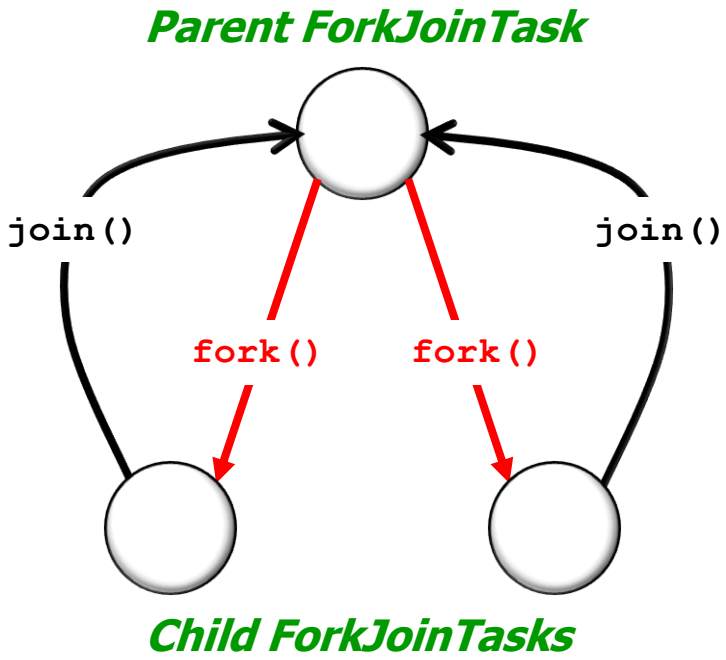
ForkJoinTask

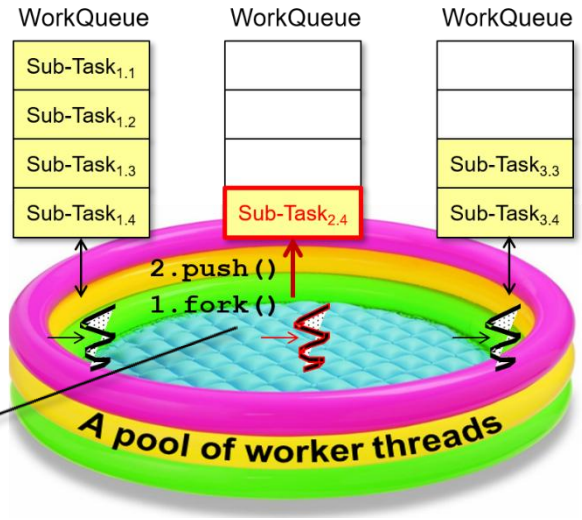
fork() is a lightweight variant of Thread. start() that creates a child ForkJoinTask.

Overview of the ForkJoinTask Class

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

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



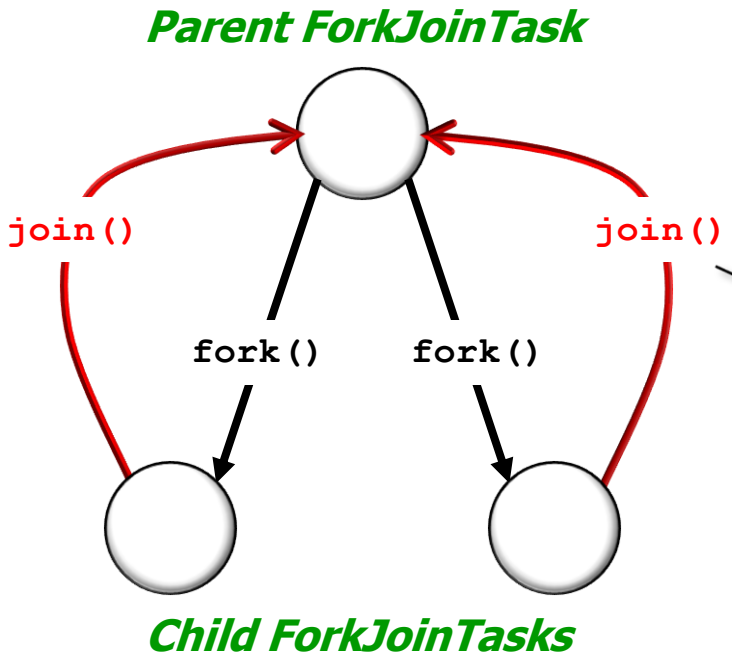


fork() doesn't run the task but places it on a work queue in the calling worker thread.

Overview of the ForkJoinTask Class

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

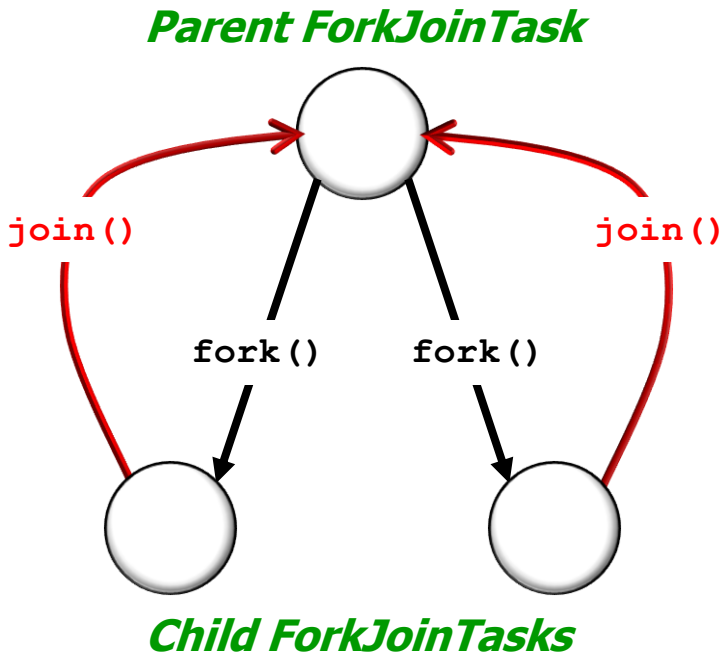
ForkJoinTask <T>	**fork**()—Arranges to asynchronously execute this task in the appropriate pool
V	**join**()—Returns result of computation when it is done



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.



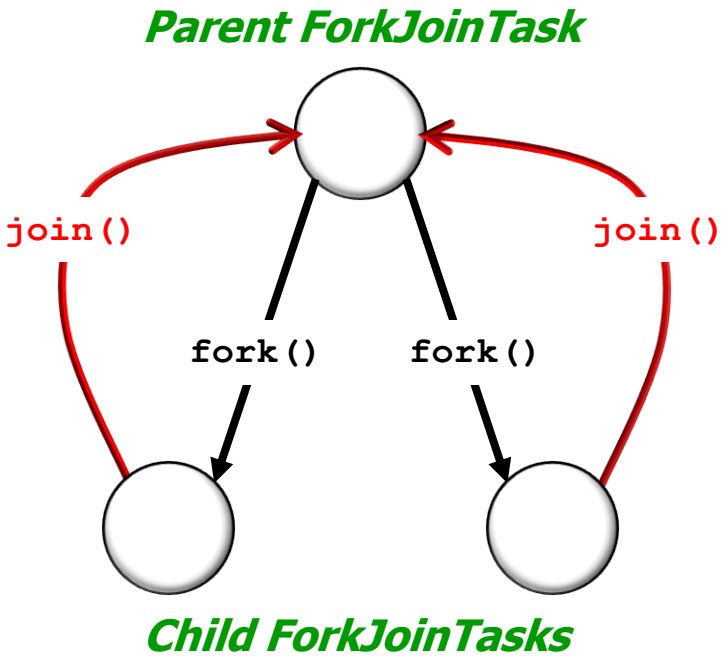
ForkJoinTask <T>	**fork**()—Arranges to asynchronously execute this task in the appropriate pool
V	**join**()—Returns result of computation when it is done

- 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>	**fork**()—Arranges to asynchronously execute this task in the appropriate pool
V	**join**()—Returns result of computation when it is done



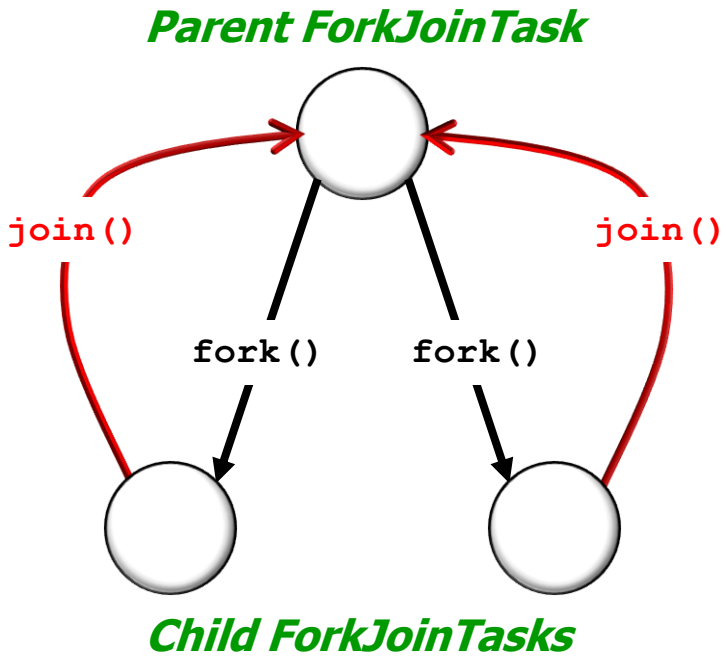
- Unlike Thread.join(), ForkJoinTask.join() doesn't simply block the calling thread.
  - It uses a worker thread to run tasks.

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>	**fork**()—Arranges to asynchronously execute this task in the appropriate pool
V	**join**()—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

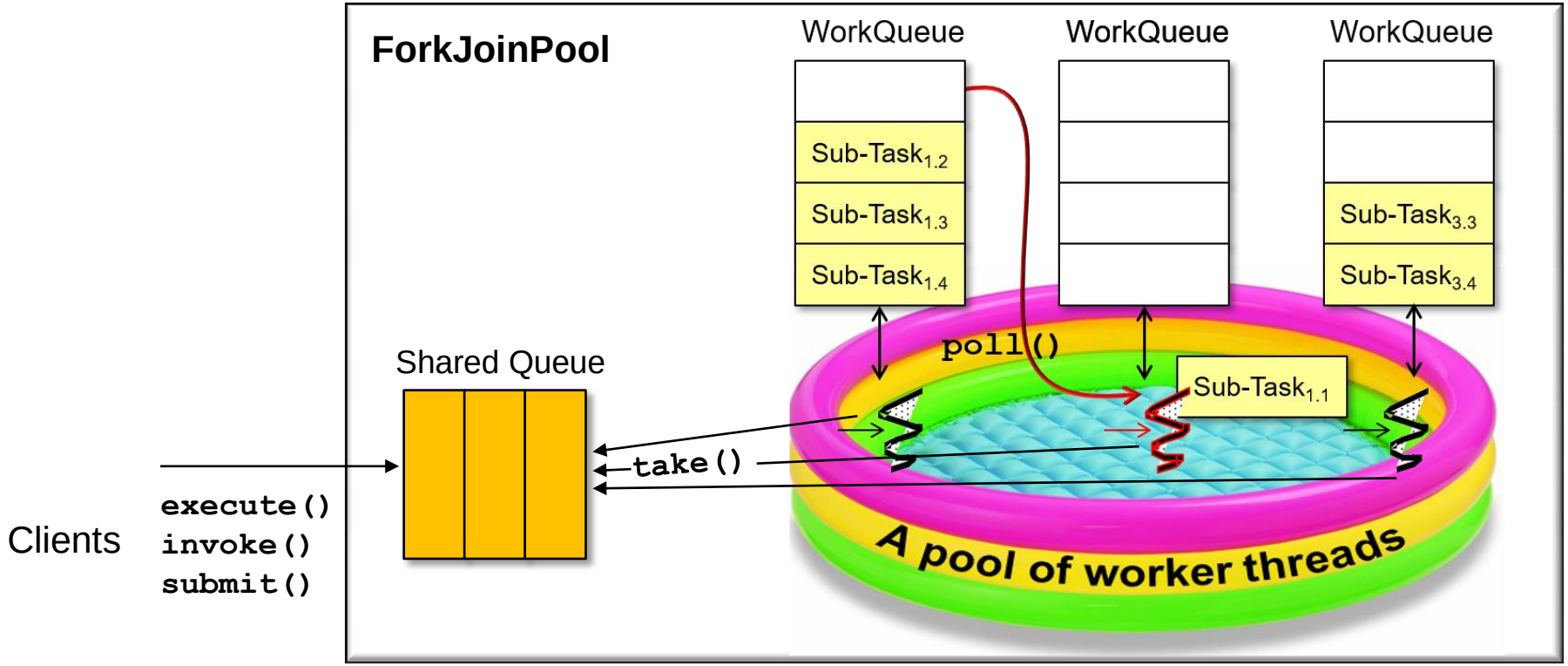
- ForkJoinPool enables non-ForkJoinTask clients to process ForkJoinTasks.

void	execute(ForkJoinTask<T>)—Arrange async execution
T	invoke(ForkJoinTask<T>)—Performs the given task, returning its result upon completion
ForkJoinTask<T>	submit(ForkJoinTask)—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 queued, which feeds "work-stealing" queues managed by worker threads.



See en.wikipedia.org/wiki/Work_stealing

Overview of the ForkJoinTask Class

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

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

The End

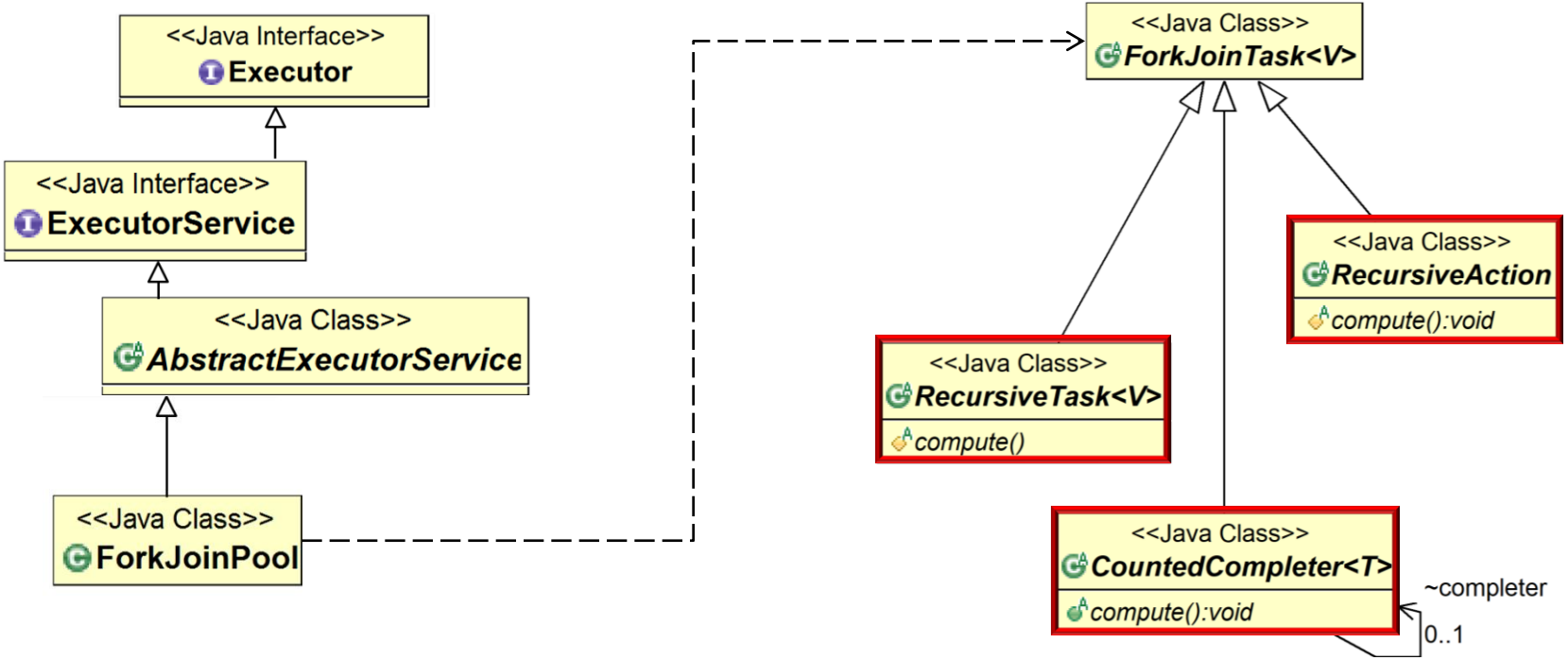
The Java Fork-Join Pool

Subclasses of ForkJoinTask

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

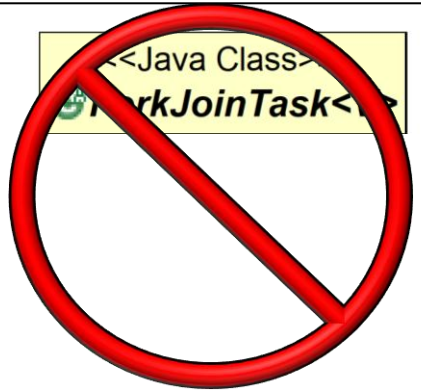
- Understand how the Java fork-join framework processes tasks in parallel
- Recognize the structure and functionality of the fork-join framework



The Subclasses of ForkJoinTask

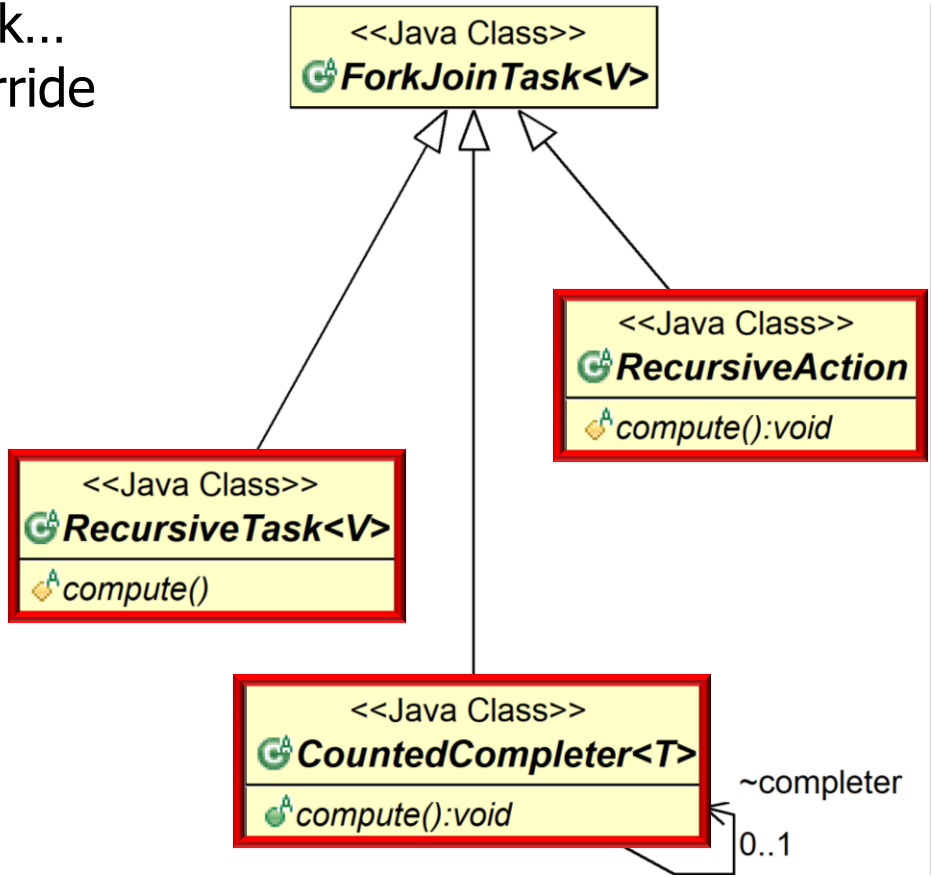
The Subclasses of ForkJoinTask

- Programs don’t implement ForkJoinTask…



The Subclasses of ForkJoinTask

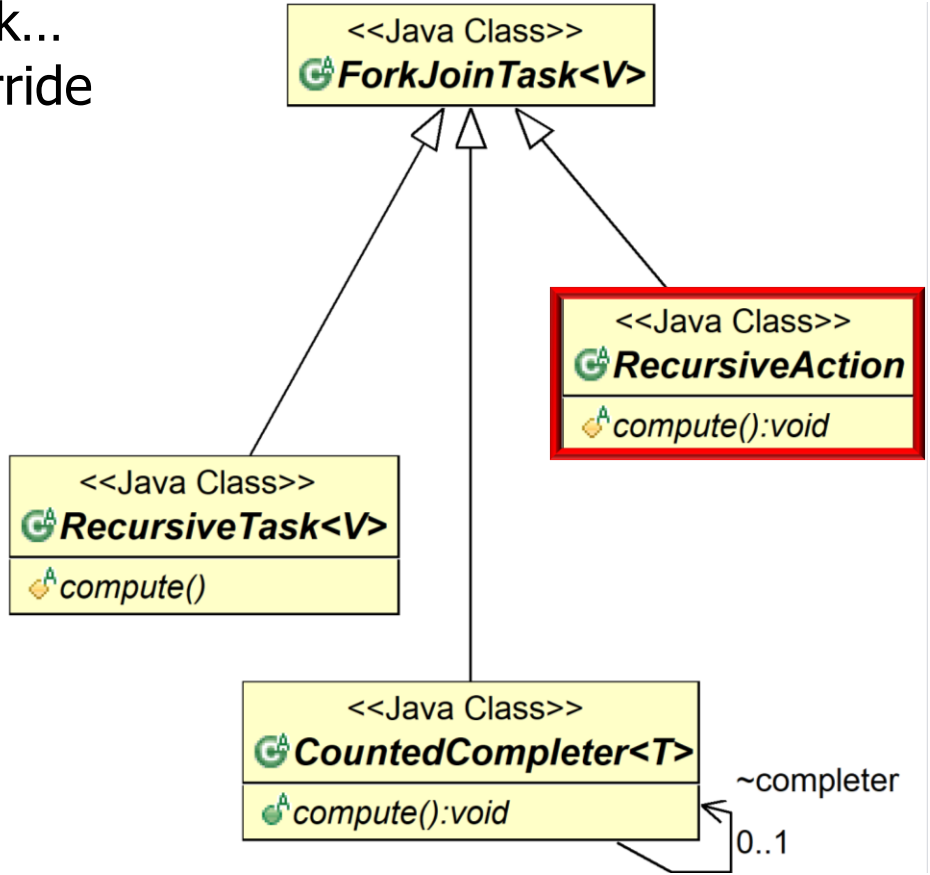
- Programs don't implement ForkJoinTask… but instead extend a subclass and override its compute() hook method.



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-tree.html

The Subclasses of ForkJoinTask

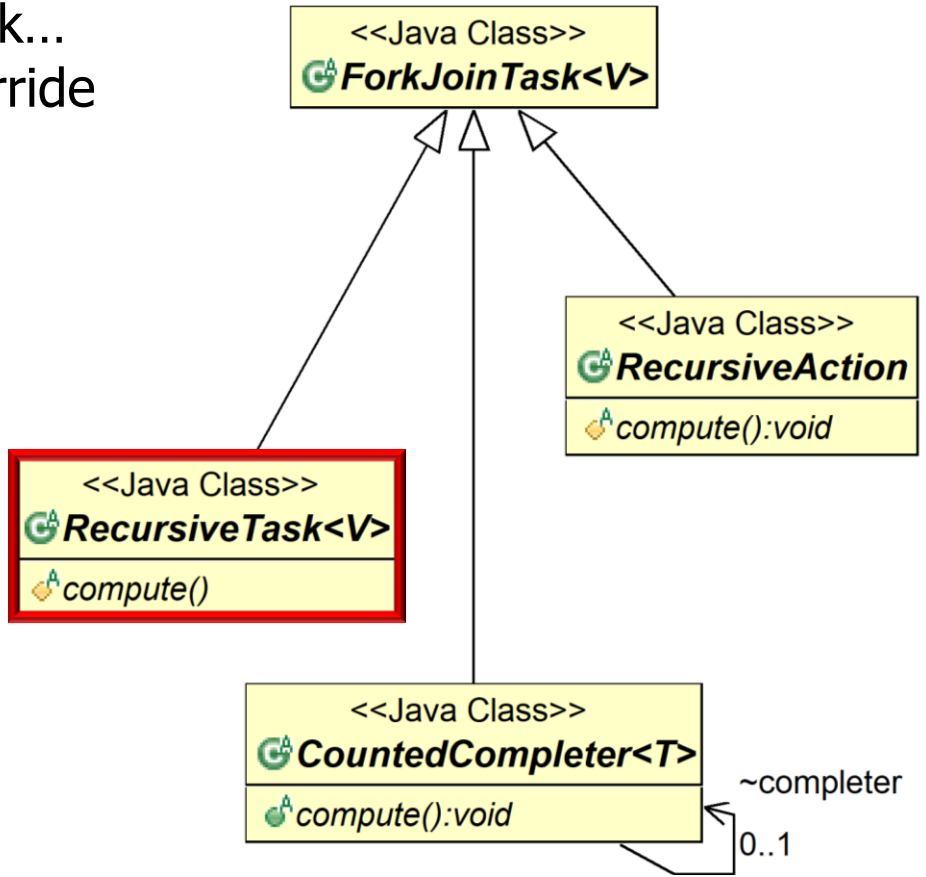
- Programs don't implement ForkJoinTask… but instead extend a subclass and override its compute() hook method.
  - **RecursiveAction**
    - Use for computations that do not return results



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

The Subclasses of ForkJoinTask

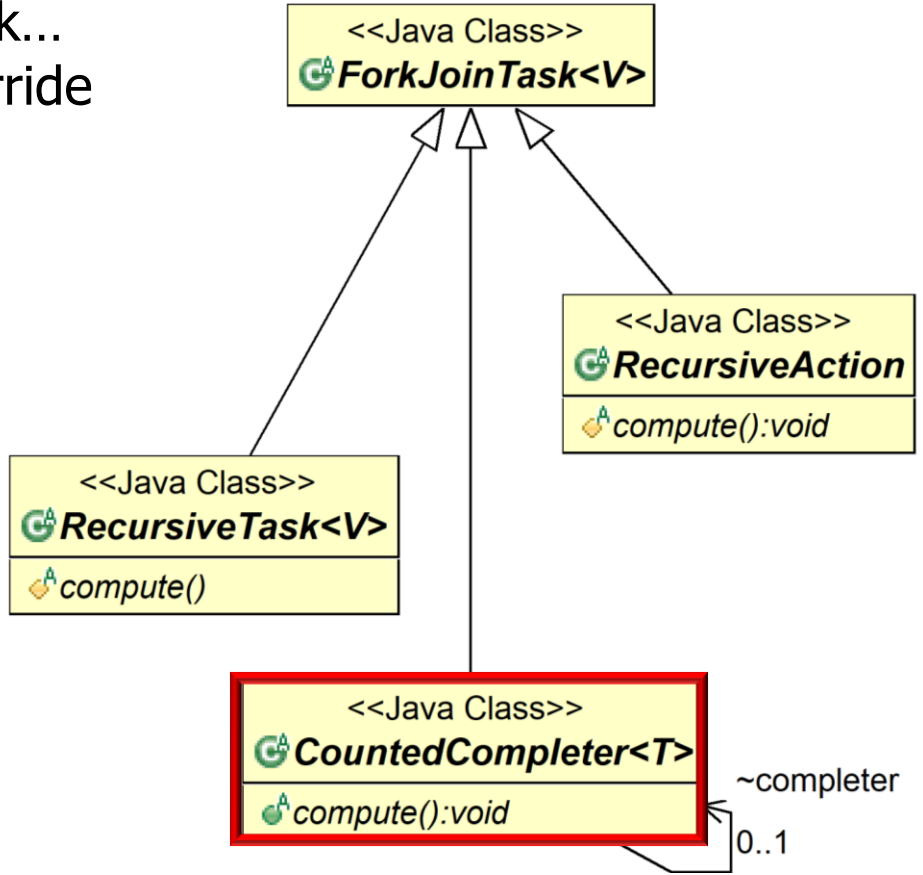
- Programs don't implement ForkJoinTask… but instead extend a subclass and override its compute() hook method.
  - RecursiveAction
  - **RecursiveTask**
    - Use for computations that *do* return results



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

The Subclasses of ForkJoinTask

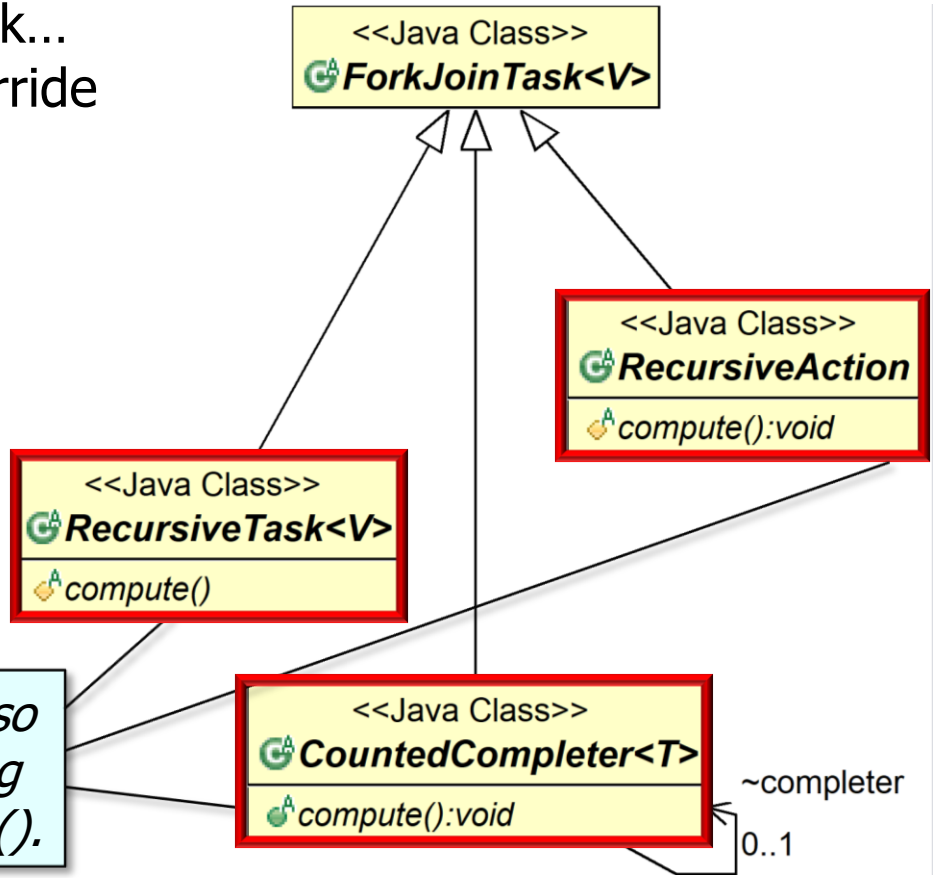
- Programs don't implement ForkJoinTask… but instead extend a subclass and override its compute() hook method.
  - RecursiveAction
  - RecursiveTask
  - **CountedCompleter**
    - Used for computations in which completed actions trigger other actions



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

The Subclasses of ForkJoinTask

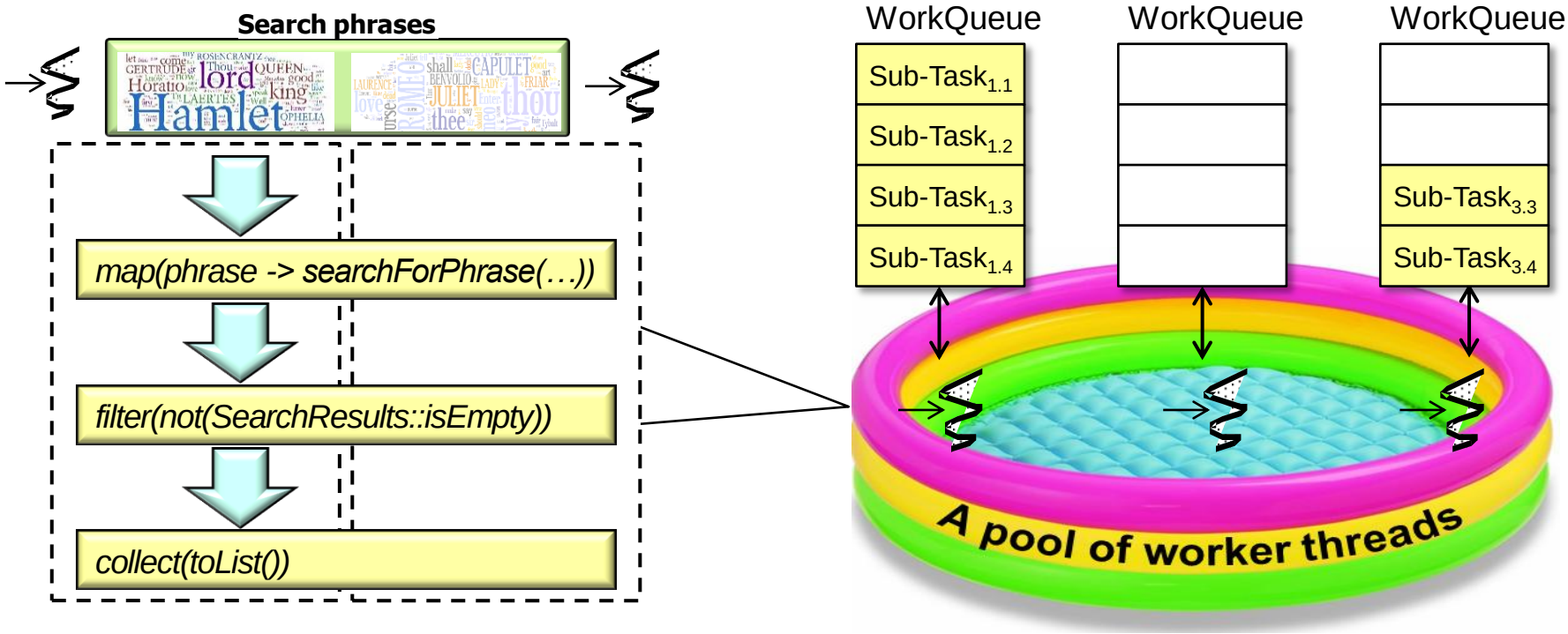
- Programs don't implement ForkJoinTask… but instead extend a subclass and override its compute() hook method.
  - **RecursiveAction**
  - **RecursiveTask**
  - **CountedCompleter**



These classes aren't functional interfaces, so they must be subclassed, rather than using lambda expressions to implement compute().

The Subclasses of ForkJoinTask

- The parallel streams framework provides a functional facade to ForkJoinTask, etc.



See lesson on "*Java Parallel Stream Internals: Mapping Onto the Common Fork-Join Pool.*"

The End

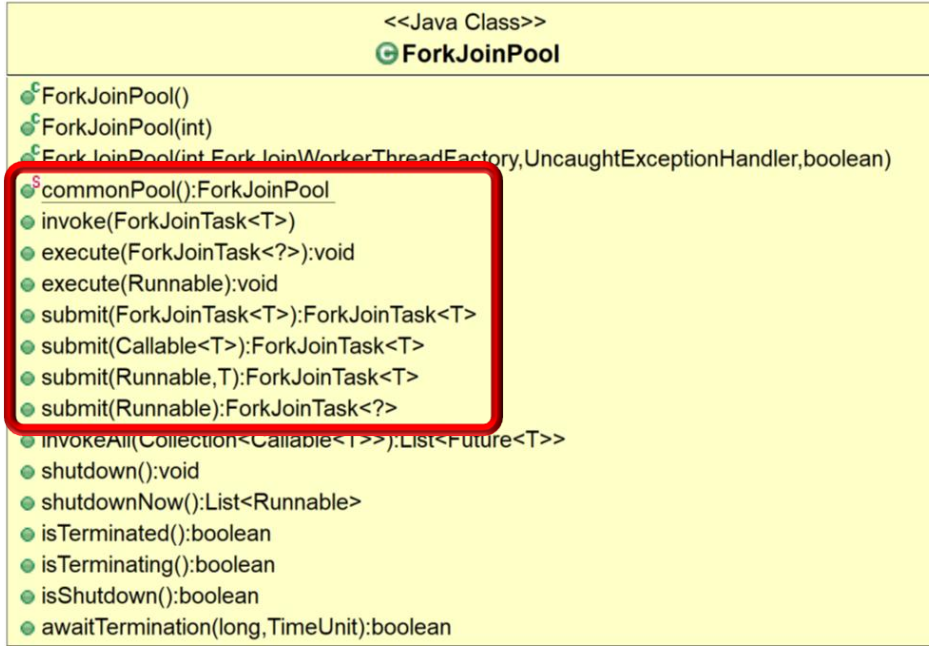
The Java Fork-Join Pool

Key Methods in ForkJoinPool

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

- Recognize the key methods in the ForkJoinPool class



Key Methods in Java ForkJoinPool

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

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

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
    - Arrange async execution of a one-way task.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
    - Arrange async execution of a one-way task.
    - Submit a two-way task for execution, return a future.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
    - Arrange async execution of a one-way task.
    - Submit a two-way task for execution, return a future.
    - Run all tasks in the collection and wait for them all to finish.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

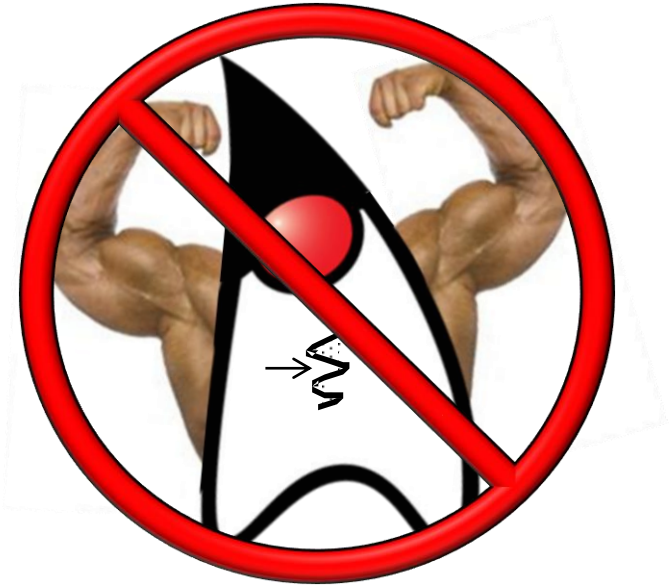
Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
    - Arrange async execution of a one-way task.
    - Submit a two-way task for execution, return a future.
    - Run all tasks in the collection and wait for them all to finish.
    - Run all tasks in the collection and wait for the first to finish.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.

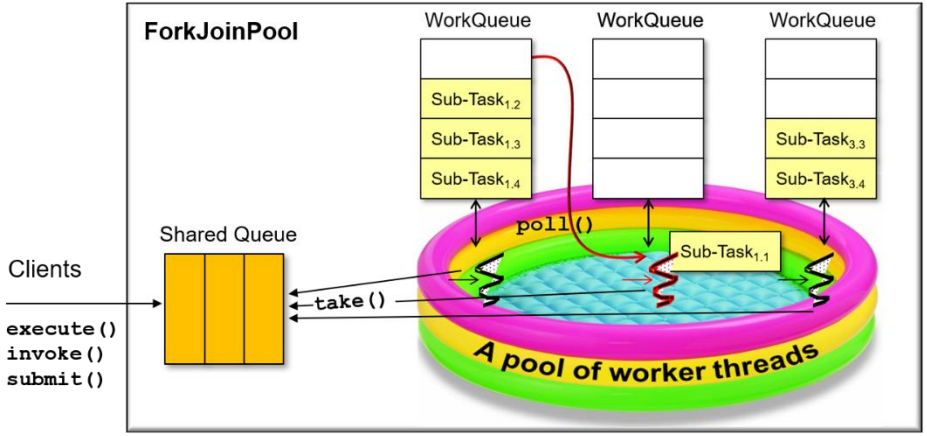


```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(Runnable cmd){...}

  <T> Future<T> submit
          (Callable<T> task){...}

  <T> List<Future<T>> invokeAll
      (Collection<? extends
       Callable<T>> tasks){...}

  <T> T invokeAny
     (Collection<? extends
      Callable<T>> tasks){...}
```

However, these methods don't leverage the powerful fork-join pool features.

Key Methods in Java ForkJoinPool

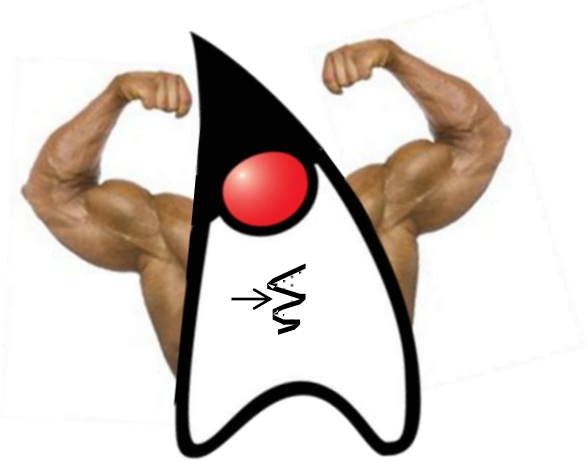
- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
  - It also implements key methods for non-ForkJoinTask clients.




```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(ForkJoinTask<T>
                 task)
  { ... }

  T invoke(ForkJoinTask<T> task)
  { ... }

  ForkJoinTask<T> submit
            (ForkJoinTask<T> task)
  { ... }
```

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
  - It also implements key methods for non-ForkJoinTask clients.



```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(ForkJoinTask<T>
                 task)
  { ... }

  T invoke(ForkJoinTask<T> task)
  { ... }

  ForkJoinTask<T> submit
            (ForkJoinTask<T> task)
  { ... }
```

These methods *can* leverage the powerful properties of the fork-join pool.

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
  - It also implements key methods for non-ForkJoinTask clients.
    - Arrange async execution of one-way task.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(ForkJoinTask<T>
                 task)
  { ... }

  T invoke(ForkJoinTask<T> task)
  { ... }

  ForkJoinTask<T> submit
            (ForkJoinTask<T> task)
  { ... }
```

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

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
  - It also implements key methods for non-ForkJoinTask clients.
    - Arrange async execution of one-way task.
    - Perform the task, blocking until it completes.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(ForkJoinTask<T>
                 task)
  { ... }

  T invoke(ForkJoinTask<T> task)
  { ... }

  ForkJoinTask<T> submit
            (ForkJoinTask<T> task)
  { ... }
```

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

Key Methods in Java ForkJoinPool

- ForkJoinPool extends Abstract ExecutorService.
  - It therefore implements the ExecutorService methods.
  - It also implements key methods for non-ForkJoinTask clients.
    - Arrange async execution of one-way task.
    - Perform the task, blocking until it completes.
    - Submit a ForkJoinTask for execution, return a future.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  void execute(ForkJoinTask<T>
                 task)
  { ... }

  T invoke(ForkJoinTask<T> task)
  { ... }

  ForkJoinTask<T> submit
            (ForkJoinTask<T> task)
  { ... }
```

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

Key Methods in Java ForkJoinPool

- The ForkJoinPool size defaults to number of cores available to Java runtime.

```
class ForkJoinPool extends
      AbstractExecutorService {
  public ForkJoinPool() {
    this(Math.min(MAX_CAP,
         Runtime.getRuntime()
         .availableProcessors()),
         ...);
  }

  public ForkJoinPool
            (int parallelism) {
    this(parallelism, ...);
  }
  ...
```

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

Key Methods in Java ForkJoinPool

- The ForkJoinPool size defaults to number of cores available to Java runtime.

Returns the number of processor cores available to the Java execution environment

```
class ForkJoinPool extends
      AbstractExecutorService {
  public ForkJoinPool() {
    this(Math.min(MAX_CAP,
          Runtime.getRuntime()
          .availableProcessors()),
          ...);
  }

  public ForkJoinPool
              (int parallelism) {
    this(parallelism, ...);
  }
  ...
```

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors

Key Methods in Java ForkJoinPool

- The ForkJoinPool size defaults to number of cores available to Java runtime.
  - This size can also be controlled programmatically.

```
class ForkJoinPool extends
      AbstractExecutorService {
  public ForkJoinPool() {
    this(Math.min(MAX_CAP,
         Runtime.getRuntime()
         .availableProcessors()),
         ...);
  }

  public ForkJoinPool
            (int parallelism) {
    this(parallelism, ...);
  }
  ...
```

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

Key Methods in Java ForkJoinPool

- The common fork-join pool can be accessed via a static method.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  static final ForkJoinPool
          common;

  public static ForkJoinPool
              commonPool() {
    return common;
  }
```

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

Key Methods in Java ForkJoinPool

- The common fork-join pool can be accessed via a static method.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  static final ForkJoinPool
          common;

  public static ForkJoinPool
              commonPool() {
    return common;
  }
```

This method accesses a static field that can be accessed via all threads in a process.

Key Methods in Java ForkJoinPool

- The common fork-join pool can be accessed via a static method.
  - The common pool is used by any ForkJoinTask that is not explicitly submitted to a specified pool.

```
class ForkJoinPool extends
      AbstractExecutorService {
  ...
  static final ForkJoinPool
          common;

  public static ForkJoinPool
                commonPool() {
    return common;
  }
```

Key Methods in Java ForkJoinPool

- ForkJoinPool also provides various management and monitoring operations.

int	getParallelism()—Returns the targeted parallelism level of this pool
int	getPoolSize()—Returns the number of worker threads that have started but not yet terminated
int	getQueuedSubmissionCount()—Returns an estimate of the number of tasks submitted to this pool that have not yet begun executing
long	getStealCount()—Returns an estimate of the total number of tasks stolen from one thread's work queue by another

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

Key Methods in Java ForkJoinPool

- ForkJoinPool also provides various management and monitoring operations.

int	getParallelism()—Returns the targeted parallelism level of this pool
int	getPoolSize()—Returns the number of worker threads that have started but not yet terminated
int	getQueuedSubmissionCount()—Returns an estimate of the number of tasks submitted to this pool that have not yet begun executing
long	getStealCount()—Returns an estimate of the total number of tasks stolen from one thread's work queue by another

Key Methods in Java ForkJoinPool

- ForkJoinPool also provides various management and monitoring operations.

int	getParallelism()—Returns the targeted parallelism level of this pool
int	getPoolSize()—Returns the number of worker threads that have started but not yet terminated
int	getQueuedSubmissionCount()—Returns an estimate of the number of tasks submitted to this pool that have not yet begun executing
long	getStealCount()—Returns an estimate of the total number of tasks stolen from one thread's work queue by another

Key Methods in Java ForkJoinPool

- ForkJoinPool also provides various management and monitoring operations.

int	getParallelism()—Returns the targeted parallelism level of this pool
int	getPoolSize()—Returns the number of worker threads that have started but not yet terminated
int	getQueuedSubmissionCount()—Returns an estimate of the number of tasks submitted to this pool that have not yet begun executing
long	getStealCount()—Returns an estimate of the total number of tasks stolen from one thread's work queue by another

Key Methods in Java ForkJoinPool

- ForkJoinPool also provides various management and monitoring operations.

int	getParallelism()—Returns the targeted parallelism level of this pool
int	getPoolSize()—Returns the number of worker threads that have started but not yet terminated
int	getQueuedSubmissionCount()—Returns an estimate of the number of tasks submitted to this pool that have not yet begun executing
long	getStealCount()—Returns an estimate of the total number of tasks stolen from one thread's work queue by another

The End

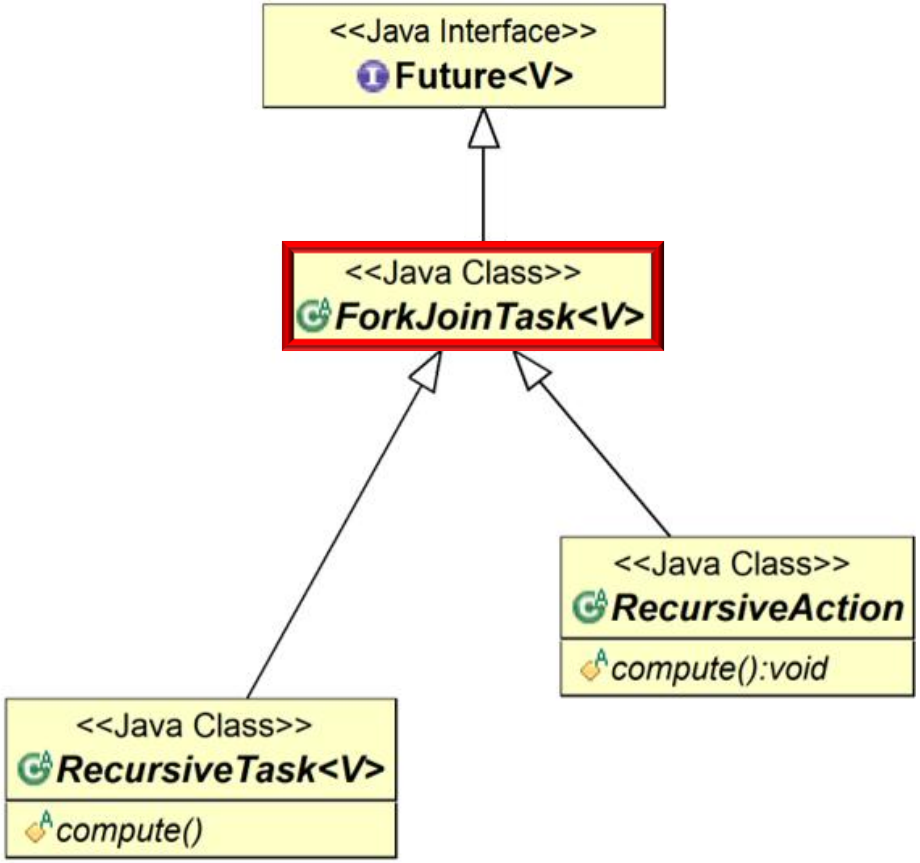
The Java Fork-Join Pool

Key Methods in ForkJoinTask

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

- Recognize the key methods in the ForkJoinPool class
- Recognize the key methods in the ForkJoinTask class

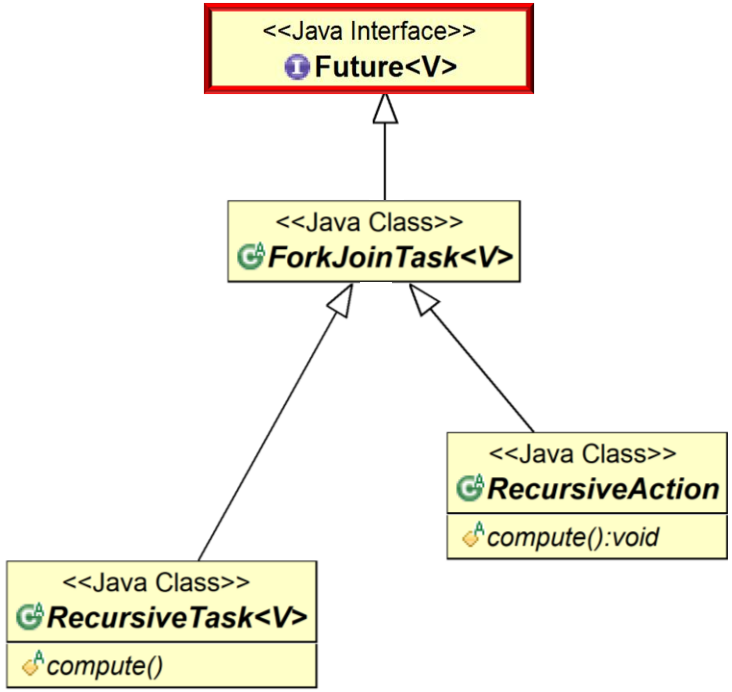


Key Methods in Java ForkJoinTask

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
```

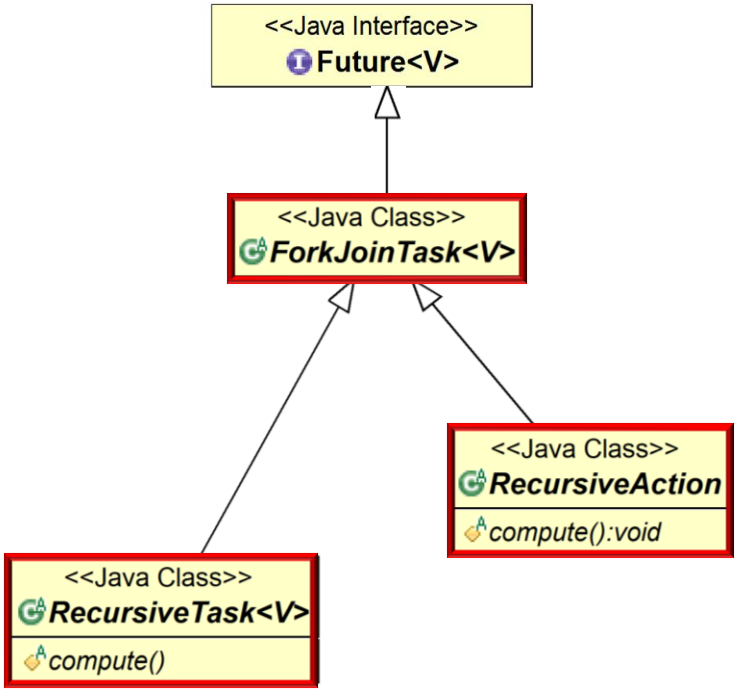


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinTask.html

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
...
```



It's uncommon to use these future methods but rather use subclass methods.

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

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

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
    - Arrange to execute this task asynchronously in the current task's pool or ForkJoinPool's common pool.

```
abstract class ForkJoinTask<V>
   implements Future<V>,
              Serializable {
   ...
   final ForkJoinTask<V> fork()
   { ... }

   final V join() { ... }

   final V invoke() { ... }
```

Key Methods in Java ForkJoinTask

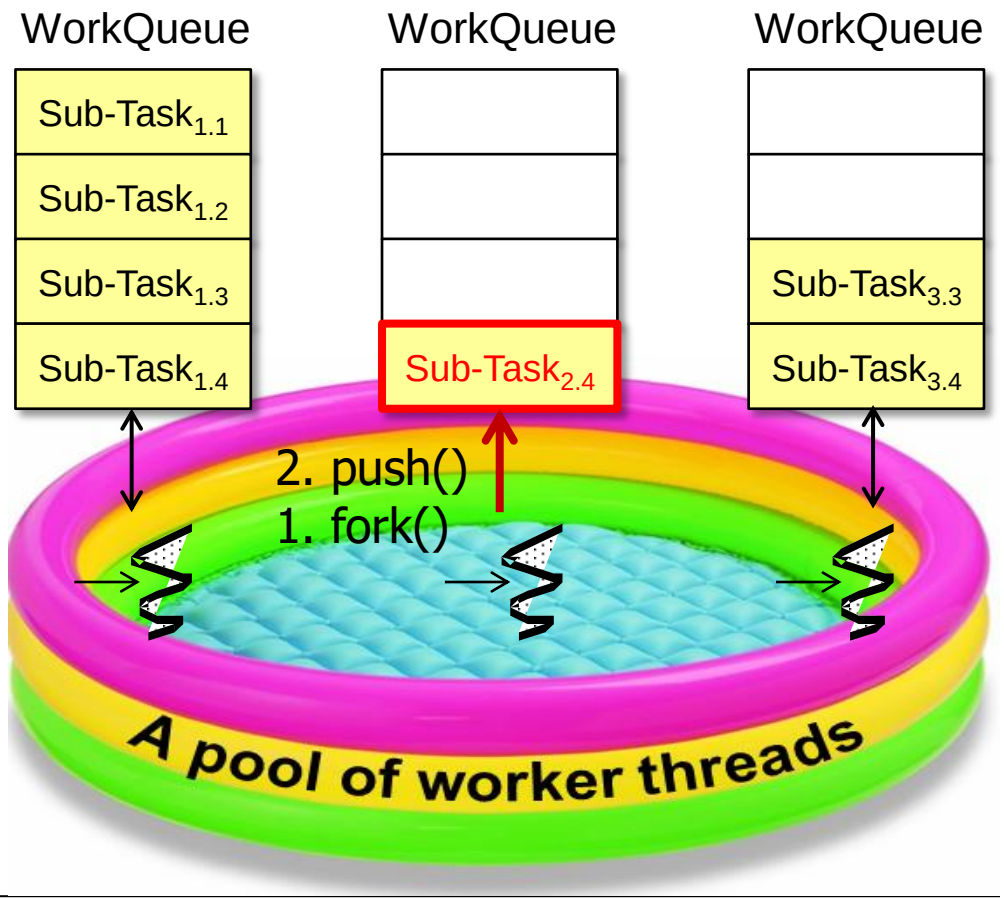
- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
    - Arrange to execute this task asynchronously in the current task's pool or ForkJoinPool's common pool.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

The fork() method does not block the caller.

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
    - Arrange to execute this task asynchronously in the current task's pool or ForkJoinPool's common pool.
    - It pushes the task on the head of the deque owned by the current worker thread.



Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

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

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
    - Calling task is "blocked" until forked sub-task is done.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
    - Calling task is "blocked" until forked sub-task is done.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

"Collaborative Jiffy Lube" model of processing!

See en.wikipedia.org/wiki/Jiffy_Lube

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
    - Calling task is "blocked" until forked sub-task is done.
    - It defines a synchronization point.

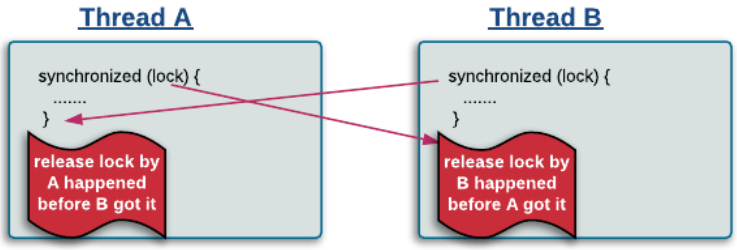
```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

See stackoverflow.com/questions/4800503/memory-visibility-in-fork-join

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
    - Calling task is "blocked" until forked sub-task is done
    - It defines a synchronization point.
      - It ensures all writes in a worker thread that "happen-before" join() are made visible to other threads after the join().

```
abstract class ForkJoinTask<V>
   implements Future<V>,
              Serializable {
   ...
   final ForkJoinTask<V> fork()
   { ... }

   final V join() { ... }

   final V invoke() { ... }
```



See www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/happens-before.html

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
  - invoke() performs this task, awaits its completion, if needed, and returns its result.

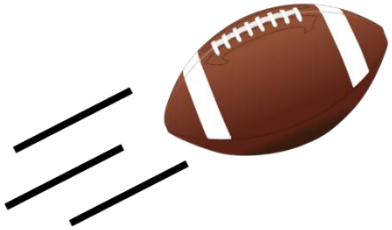
```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```

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

Key Methods in Java ForkJoinTask

- ForkJoinTask implements Future.
  - fork() enables a task to create sub-tasks that run in parallel.
  - join() returns the result of a previously forked computation when it's done.
  - invoke() performs this task, awaits its completion, if needed, and returns its result.
    - It throws RuntimeException or Error if the underlying computation did so.

```
abstract class ForkJoinTask<V>
  implements Future<V>,
             Serializable {
  ...
  final ForkJoinTask<V> fork()
  { ... }

  final V join() { ... }

  final V invoke() { ... }
```



See www.baeldung.com/java-exceptions

The End

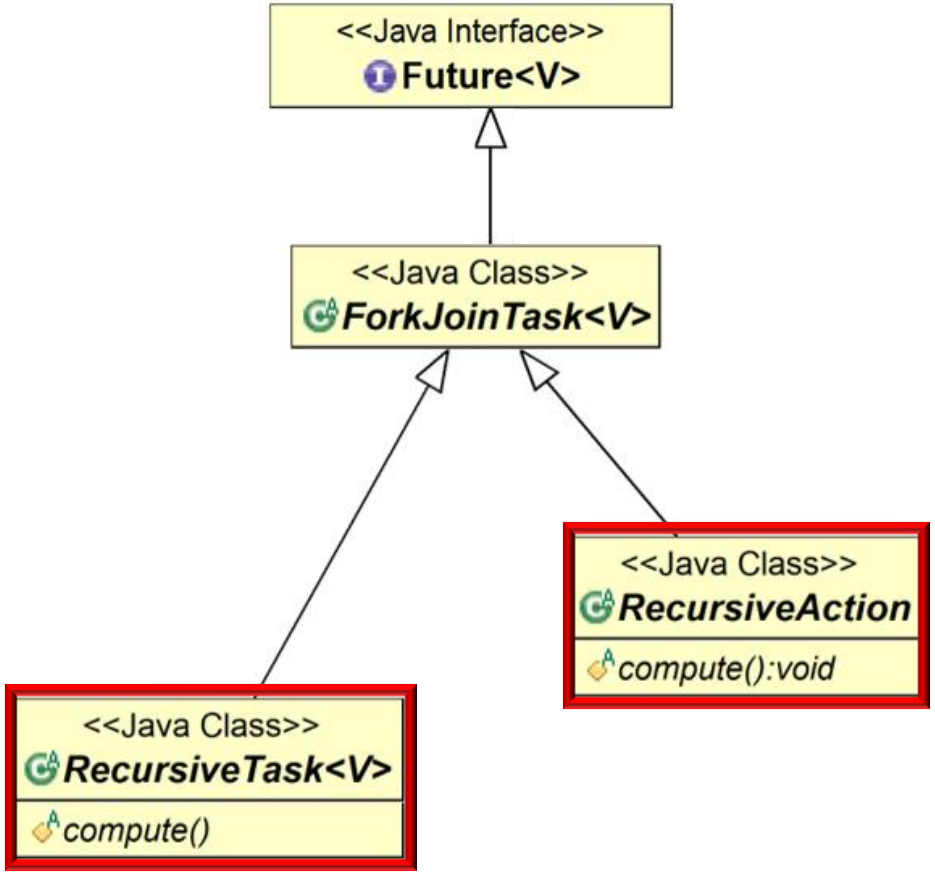
The Java Fork-Join Pool

Key Methods in RecursiveAction and RecursiveTask

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

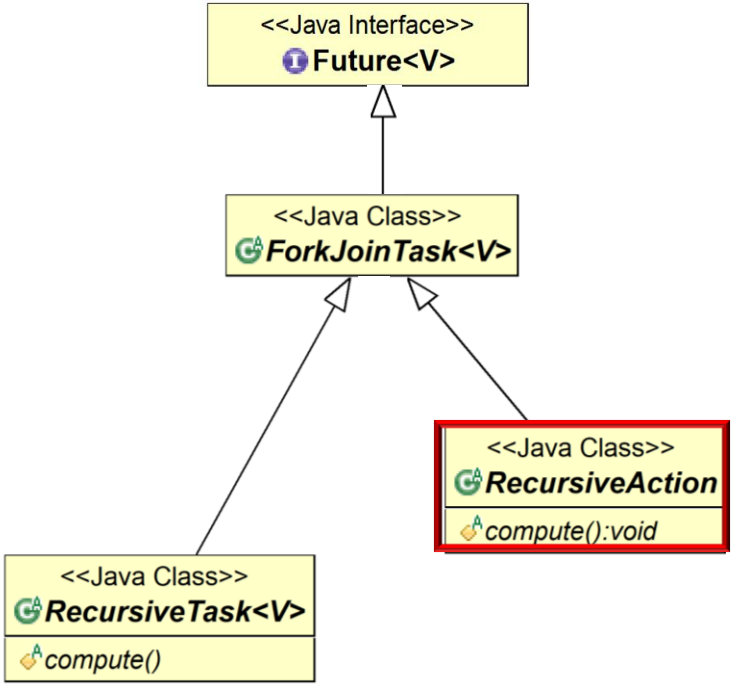
- Recognize the key methods in the ForkJoinPool class
- Recognize the key methods in the ForkJoinTask class
- Recognize the key methods in the RecursiveAction and RecursiveTask classes



Key Methods in the Java RecursiveAction

Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.



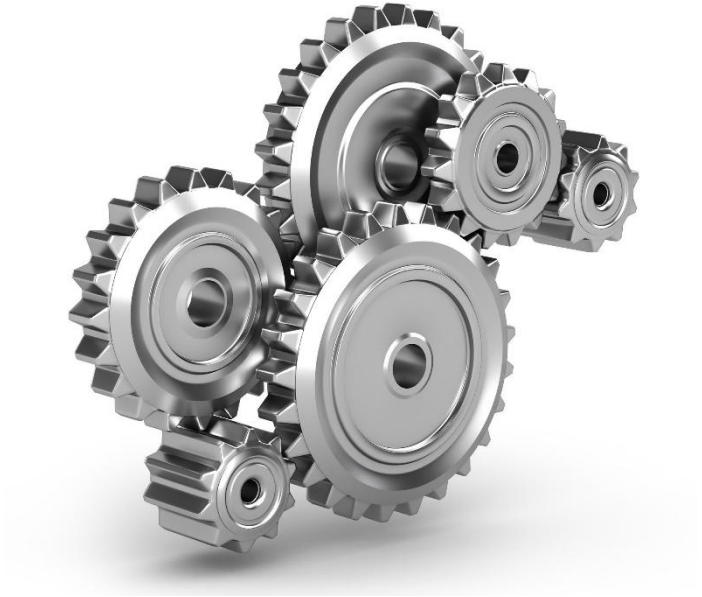

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  ...
```

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

Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();
  ...
```

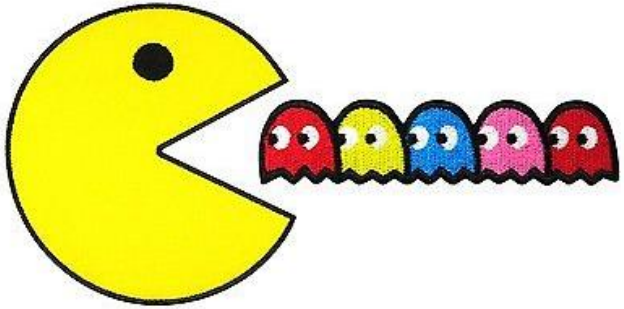


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html#compute

Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly

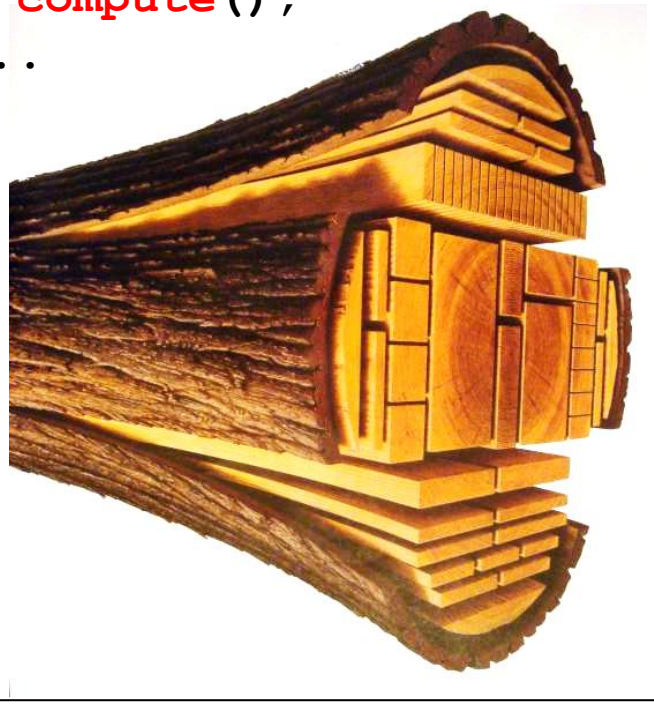
```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();
  ...
```



Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly
    - If data size is large, split work into smaller sub-tasks that are fork()'d to run in parallel

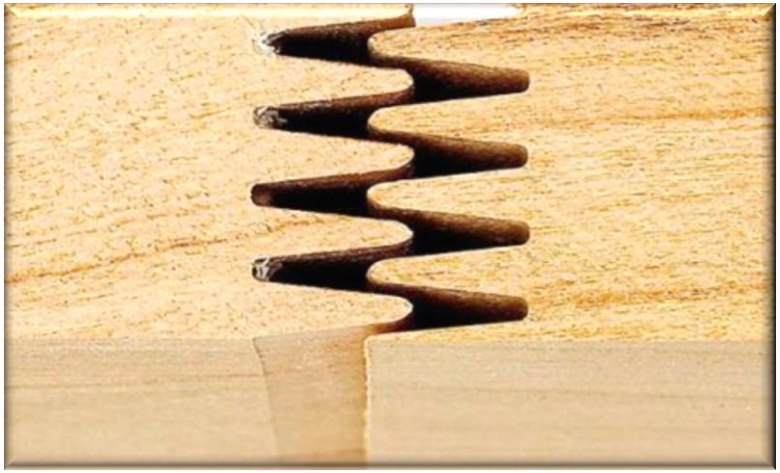
```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();
  ...
```



Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly
    - If data size is large, split work into smaller sub-tasks that are fork()'d to run in parallel
    - These smaller sub-tasks are join()'d, but a result is not returned directly
      - Results may be stored in an array

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();
...
```



Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.
  - The fork-join pool framework calls exec() to execute the task.

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();

  protected final boolean exec(){
    compute();
    return true;
  }
  ...
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html#exec

Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoinTask and does not return a result.
  - Subclasses override compute() to perform task's main computation.
  - The fork-join pool framework calls exec() to execute the task.

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();

  protected final boolean exec(){
    compute();
    return true;
  }
  ...
```

exec() is a template method, and compute() is a hook method.

See en.wikipedia.org/wiki/Template_method_pattern

Key Methods in Java RecursiveAction

- RecursiveAction extends ForkJoin Task and does not return a result.
  - Subclasses override compute() to perform task's main computation.
  - The fork-join pool framework calls exec() to execute the task.

```
abstract class RecursiveAction
  extends ForkJoinTask<Void> {
  protected abstract Void
    compute();

  protected final boolean exec(){
    compute();
    return true;
  }
  ...
```

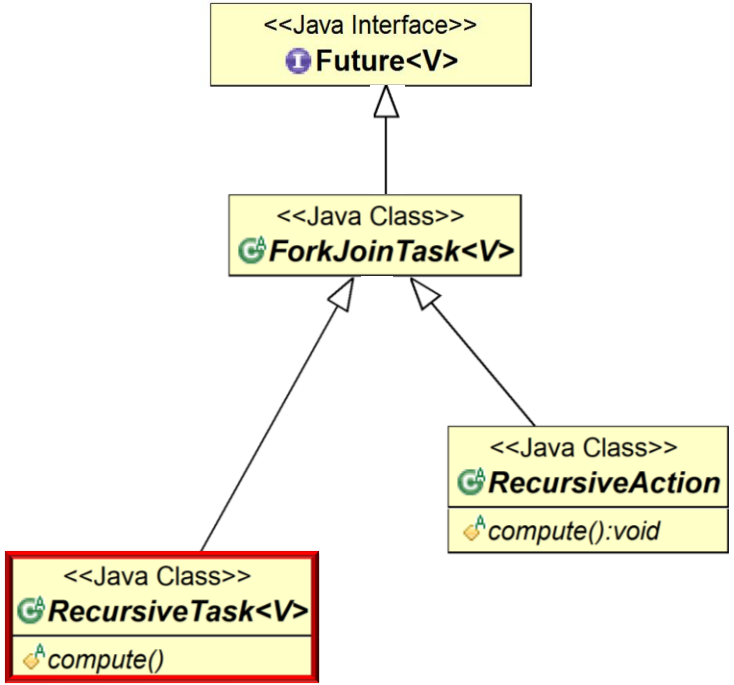
The result of compute() is not stored for subsequent access.

Key Methods in the Java RecursiveTask

Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.

```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  ...
```

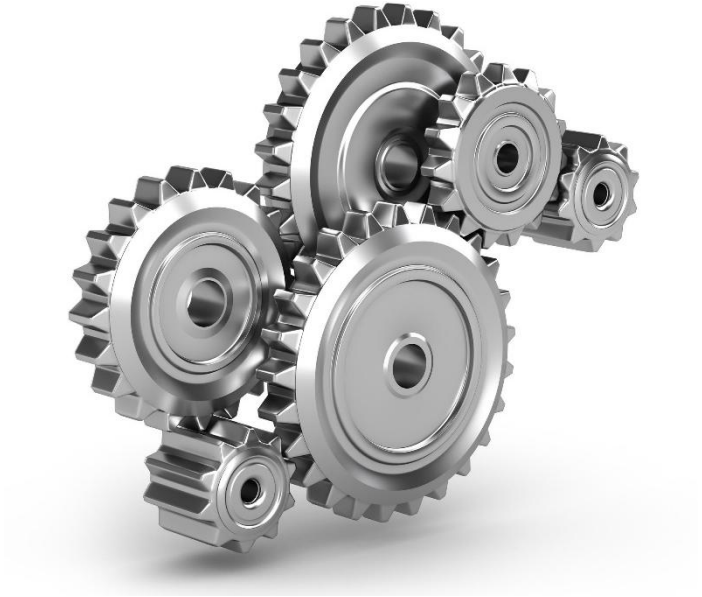


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html

Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.

```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();
  ...
```

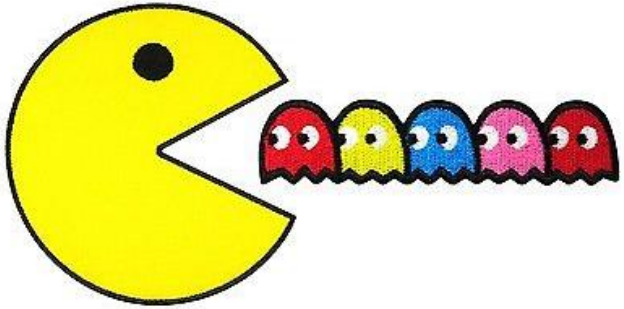


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html#compute

Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly

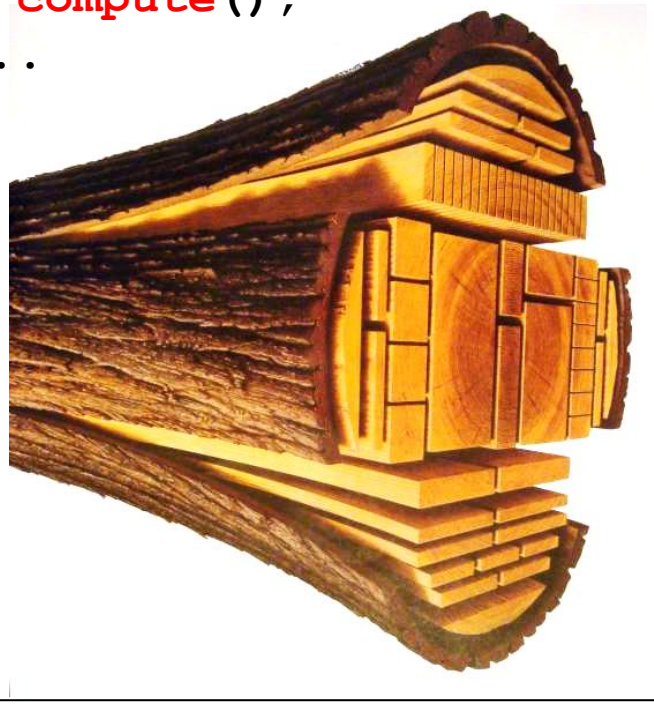
```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();
  ...
```



Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly
    - If data size is large, split work into smaller sub-tasks that are fork()'d to run in parallel

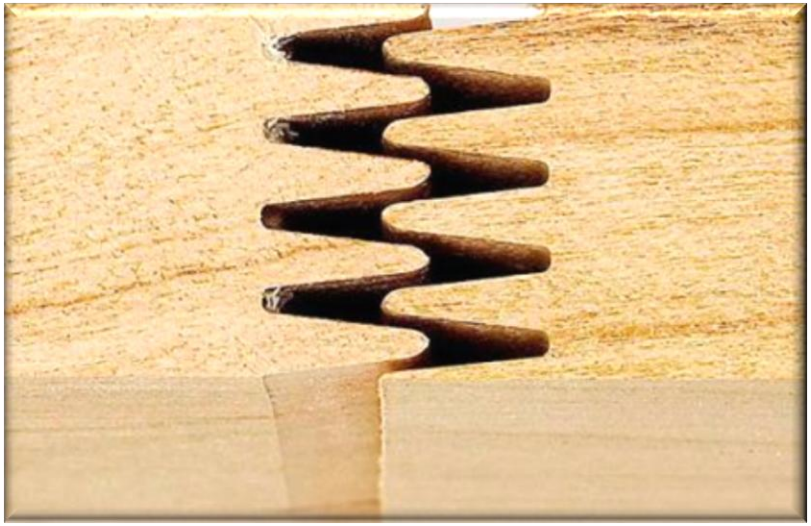
```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();
  ...
```



Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.
    - If data size is below a certain threshold, perform work directly
    - If data size is large, split work into smaller sub-tasks that are fork()'d to run in parallel
    - Results of these smaller sub-tasks are join()'d into a merged result

```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();
  ...
```



Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.
  - The fork-join pool framework calls exec() to execute the task.

```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();

  V result;

  protected final boolean exec(){
    result = compute();
    return true;
  }
  ...
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/RecursiveTask.html#exec

Key Methods in Java RecursiveTask

- RecursiveTask extends ForkJoinTask to return a result.
  - Subclasses override compute() to perform task's main computation.
  - The fork-join pool framework calls exec() to execute the task.

```
abstract class RecursiveTask<V>
  extends ForkJoinTask<V> {
  protected abstract V
    compute();

  V result;

  protected final boolean exec(){
    result = compute();
    return true;
  }
  ...
```

The result of compute() is stored for subsequent access.

Java Fork-Join Pool: Key Methods in RecursiveAction and RecursiveTask

The End

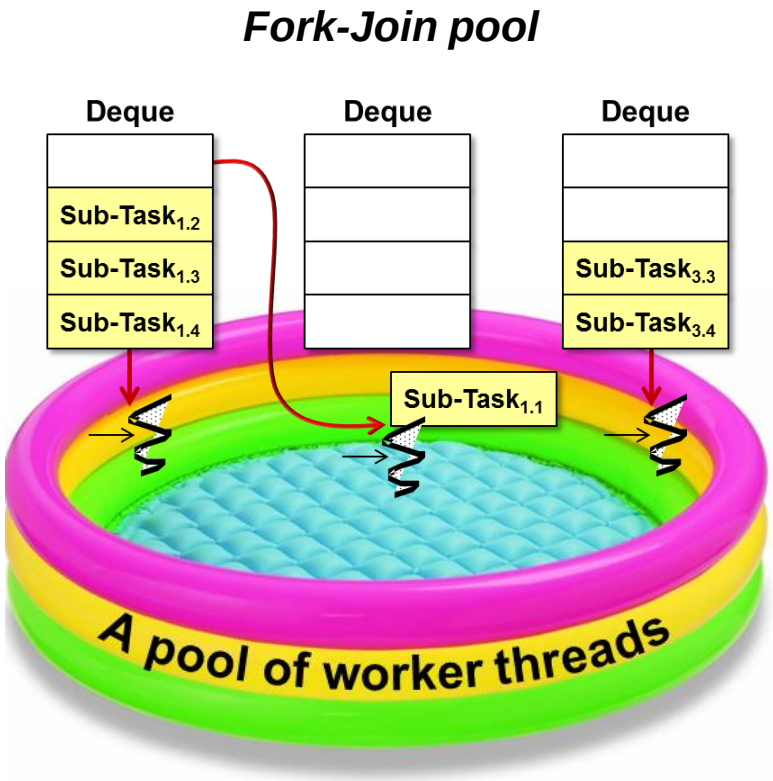
The Java Fork-Join Pool Internals

Worker Threads

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

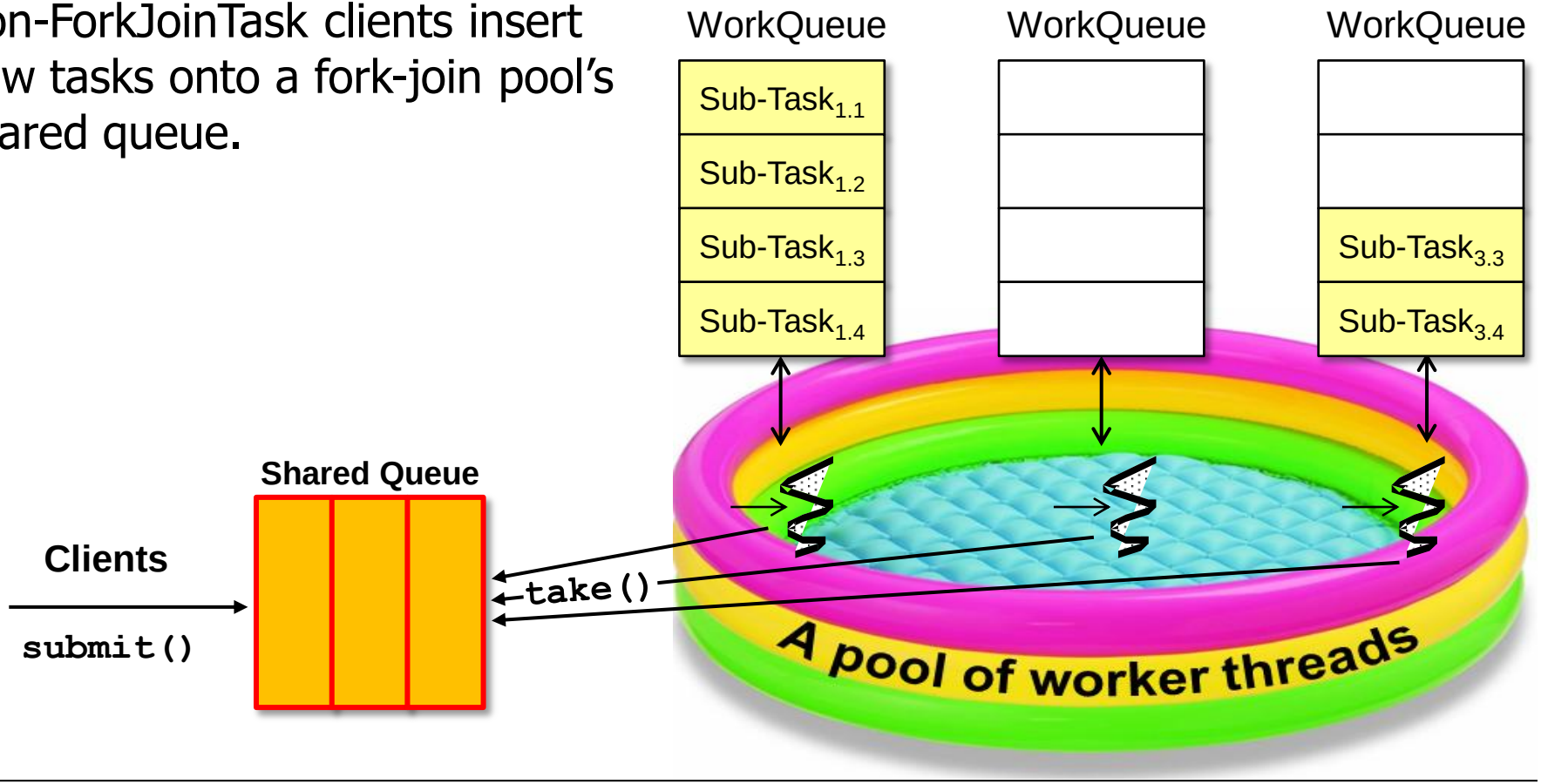
- Know how the Java fork-join framework implements worker threads



Worker Threads in a Java Fork-Join Pool

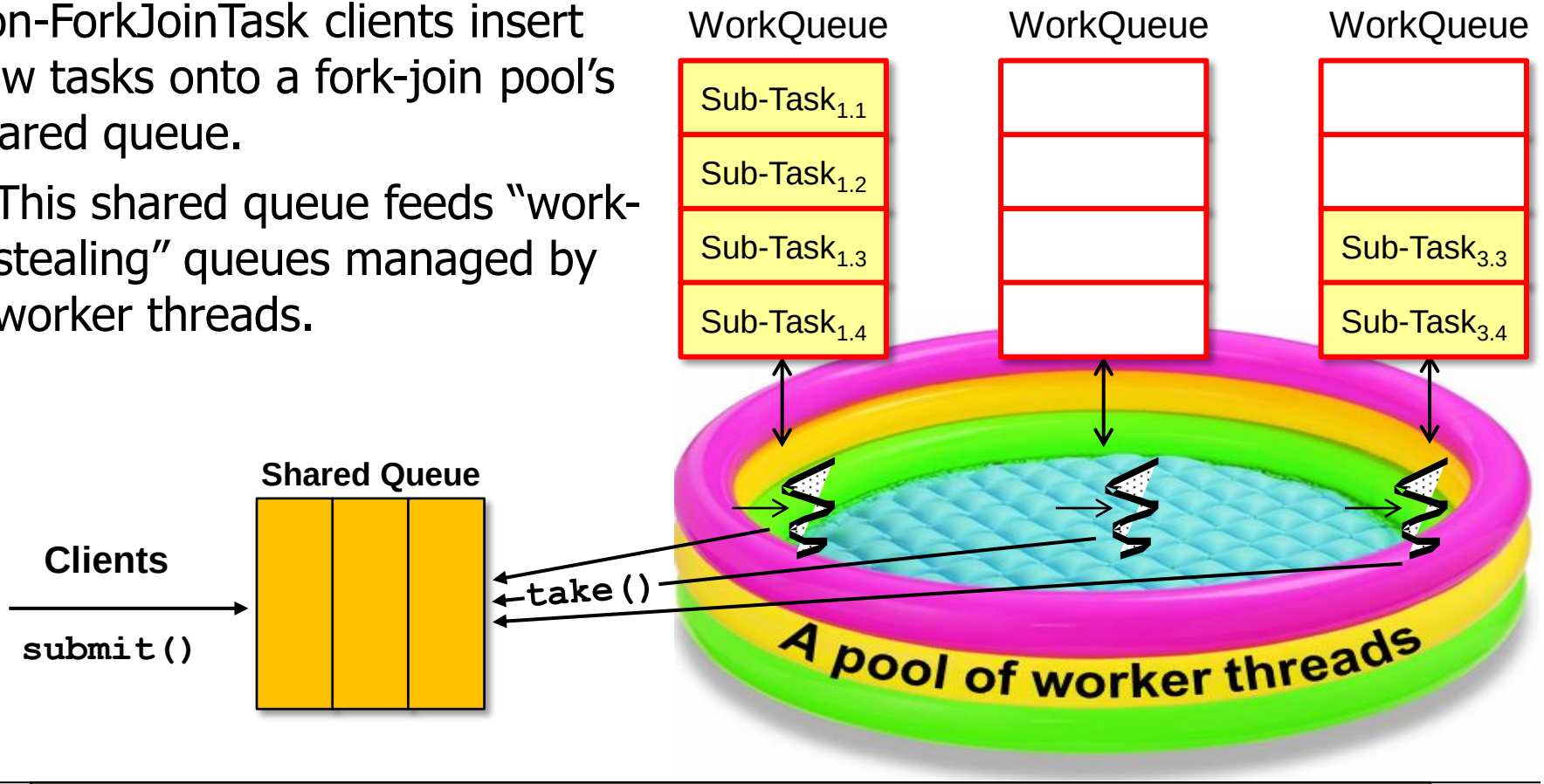
Worker Threads in a Java Fork-Join Pool

- Non-ForkJoinTask clients insert new tasks onto a fork-join pool's shared queue.



Worker Threads in a Java Fork-Join Pool

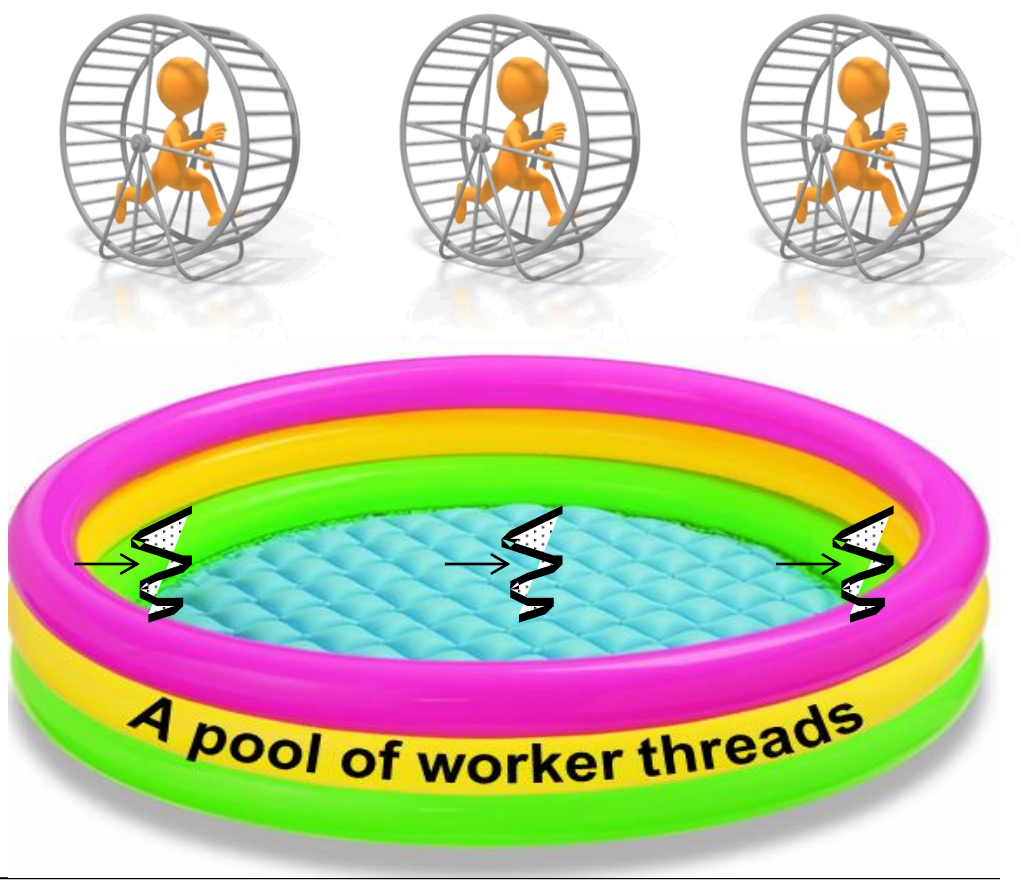
- Non-ForkJoinTask clients insert new tasks onto a fork-join pool's shared queue.
  - This shared queue feeds "work-stealing" queues managed by worker threads.



See upcoming lessons on "*The Java Fork-Join Pool: Work Stealing*"

Worker Threads in a Java Fork-Join Pool

- Each worker thread in a fork-join pool runs a loop that scans for (sub)tasks to execute.

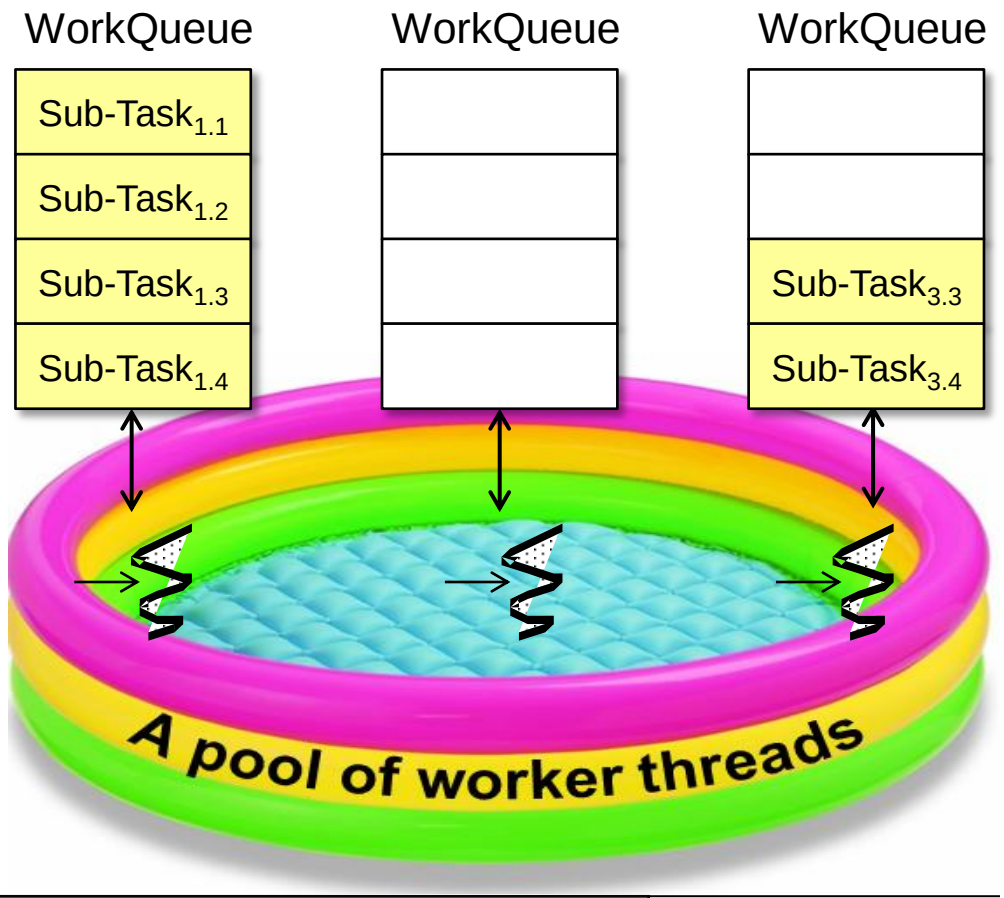


Worker Threads in a Java Fork-Join Pool

- Each worker thread in a fork-join pool runs a loop that scans for (sub)tasks to execute.
  - The goal is to keep the worker threads as busy as possible!

Worker Threads in a Java Fork-Join Pool

- A worker thread has a "double-ended queue" (aka "deque") that serves as its main source of tasks.

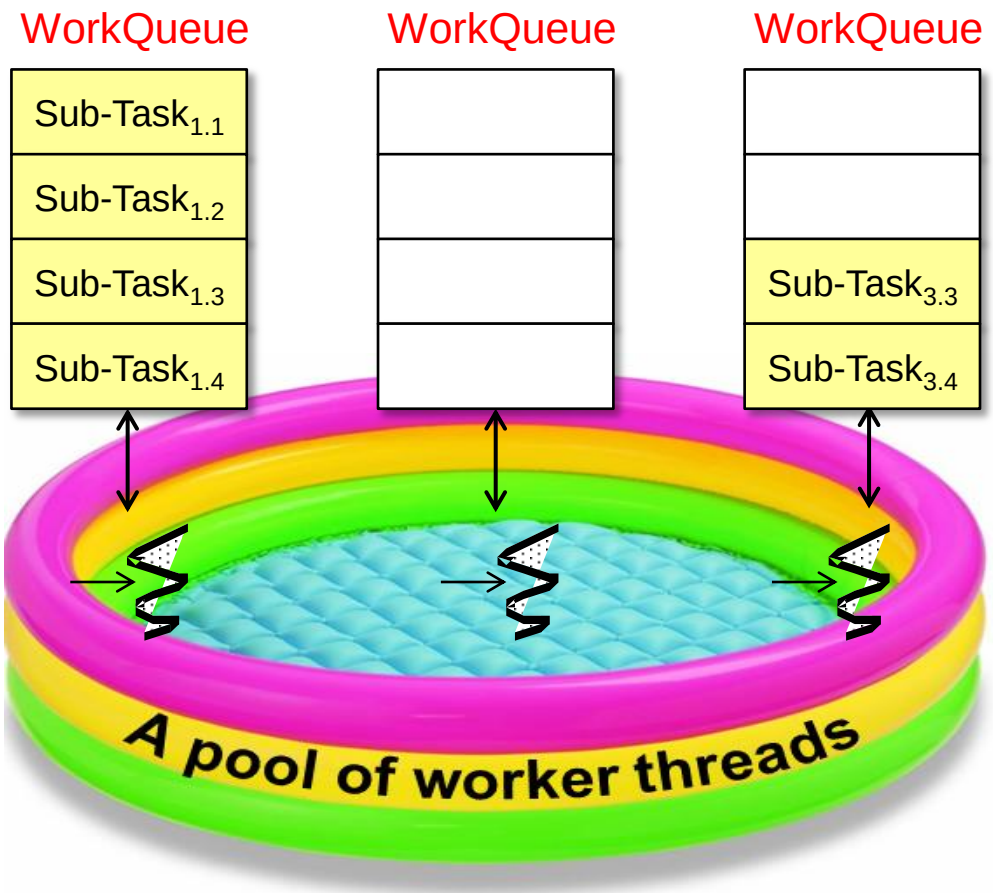


See en.wikipedia.org/wiki/Double-ended_queue

Worker Threads in a Java Fork-Join Pool

- A worker thread has a "double-ended queue" (aka "deque") that serves as its main source of tasks.
  - Implemented by WorkQueue

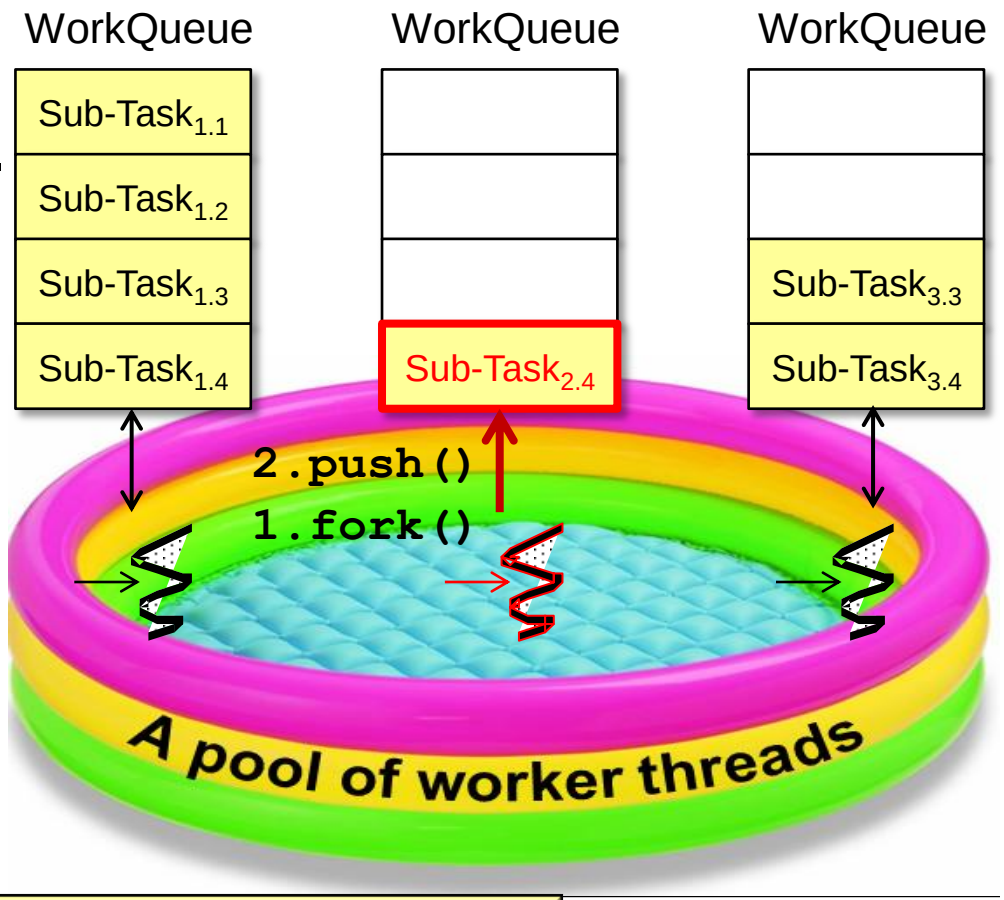
```
<<Java Class>>
WorkQueue
WorkQueue(ForkJoinPool,ForkJoinWorkerThread,int,int)
queueSize():int
isEmpty():boolean
push(ForkJoinTask<?>):void
growArray():ForkJoinTask<?>
pop():ForkJoinTask<?>
pollAt(int):ForkJoinTask<?>
poll():ForkJoinTask<?>
peek():ForkJoinTask<?>
cancelAll():void
pollAndExecAll():void
runTask(ForkJoinTask<?>):void
tryRemoveAndExec(ForkJoinTask<?>):boolean
isApparentlyUnblocked():boolean
```



See [java8/util/concurrent/ForkJoinPool.java]

Worker Threads in a Java Fork-Join Pool

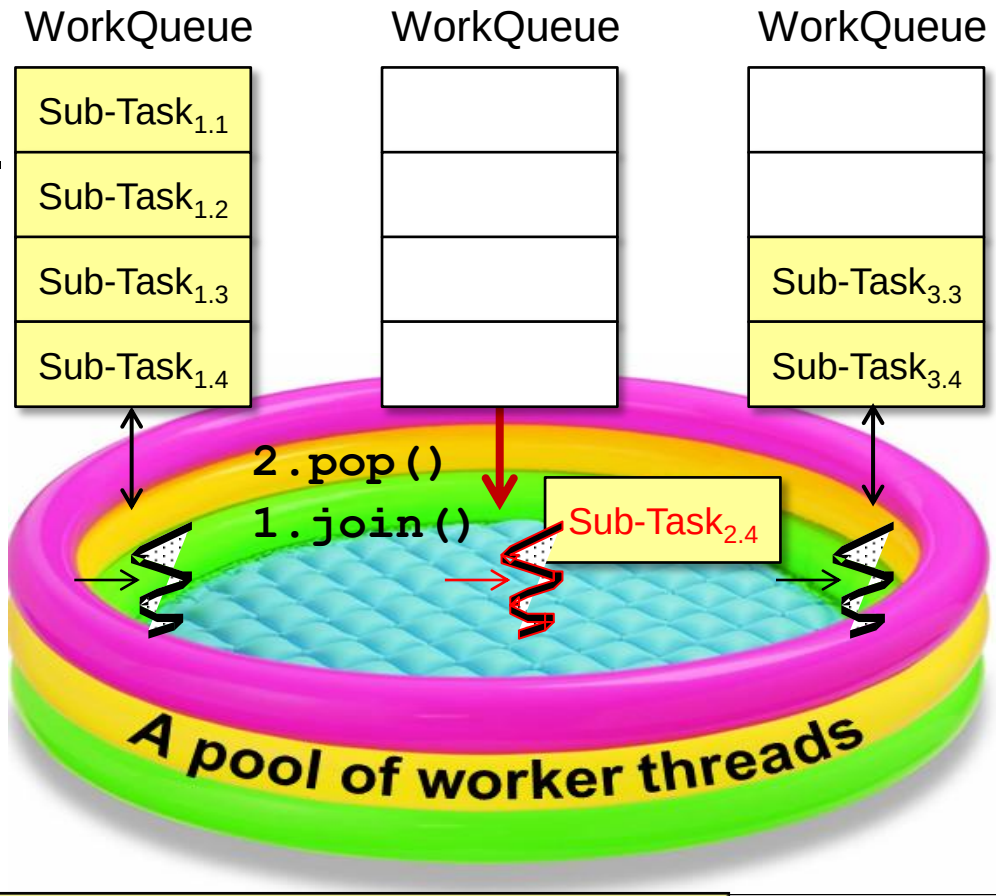
- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.



See gee.cs.oswego.edu/dl/papers/fj.pdf

Worker Threads in a Java Fork-Join Pool

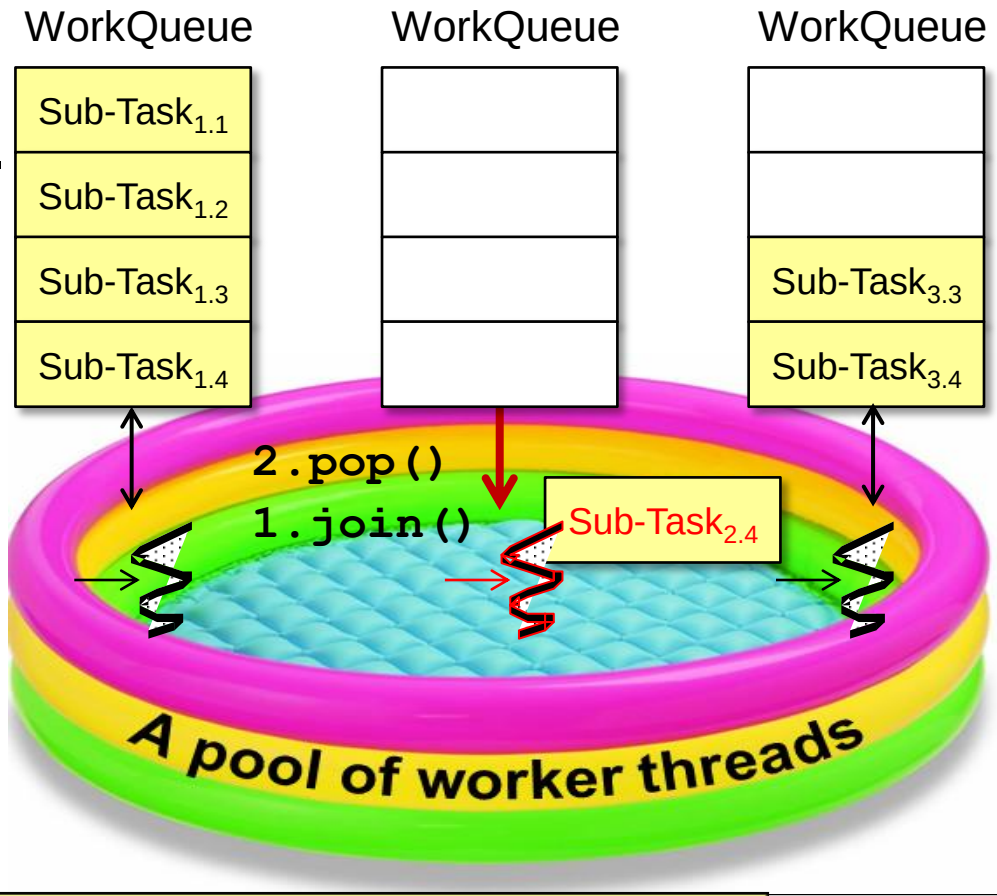
- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.
  - A worker thread processes its deque in LIFO order.



See en.wikipedia.org/wiki/Stack_(abstract_data_type)

Worker Threads in a Java Fork-Join Pool

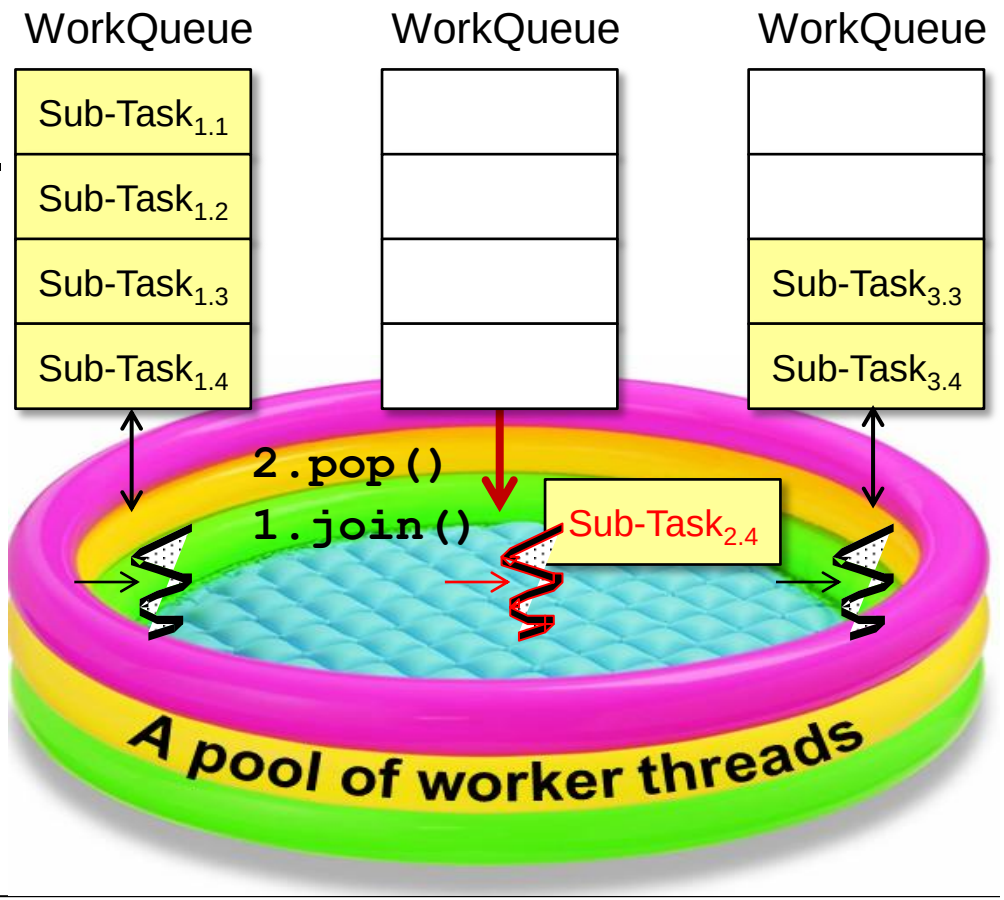
- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.
  - A worker thread processes its deque in LIFO order, that is:
    - A task pop'd from the head of a deque is run to completion



See en.wikipedia.org/wiki/Run_to_completion_scheduling

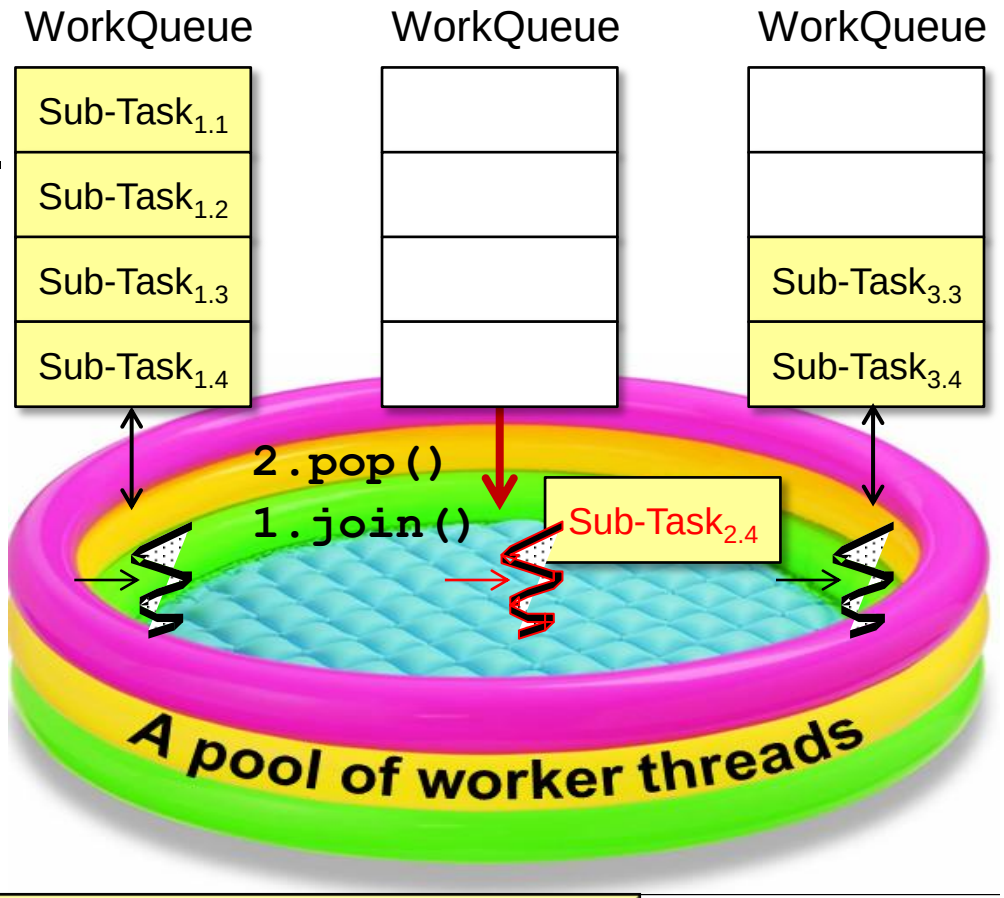
Worker Threads in a Java Fork-Join Pool

- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.
  - A worker thread processes its deque in LIFO order, that is:
    - A task pop'd from the head of a deque is run to completion
    - join() "pitches in" to pop and execute (sub)tasks



Worker Threads in a Java Fork-Join Pool

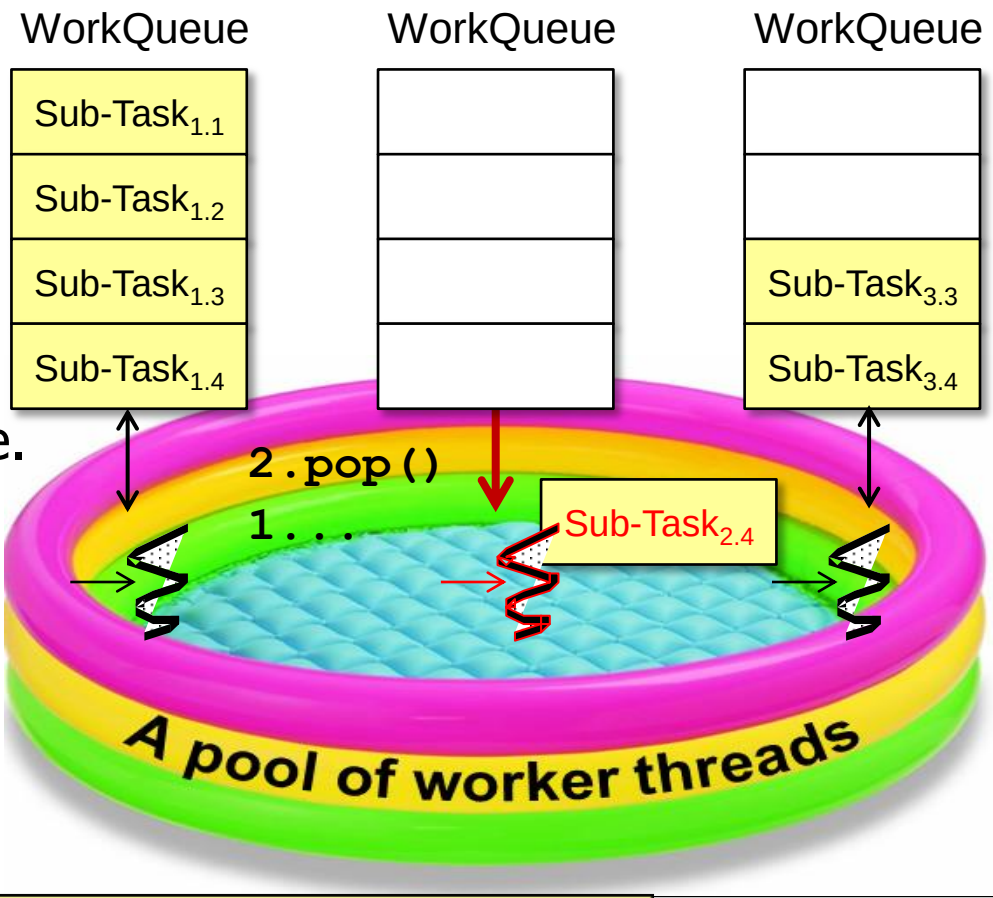
- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.
  - A worker thread processes its deque in LIFO order, that is:
    - A task pop'd from the head of a deque is run to completion
    - join() "pitches in" to pop and execute (sub-)tasks



"Collaborative Jiffy Lube" model of processing!

Worker Threads in a Java Fork-Join Pool

- If a task run by a worker thread calls fork(), the new task is pushed on the head of the worker's deque.
  - A worker thread processes its deque in LIFO order.
  - LIFO order improves locality of reference and cache performance.



See en.wikipedia.org/wiki/Locality_of_reference

The End

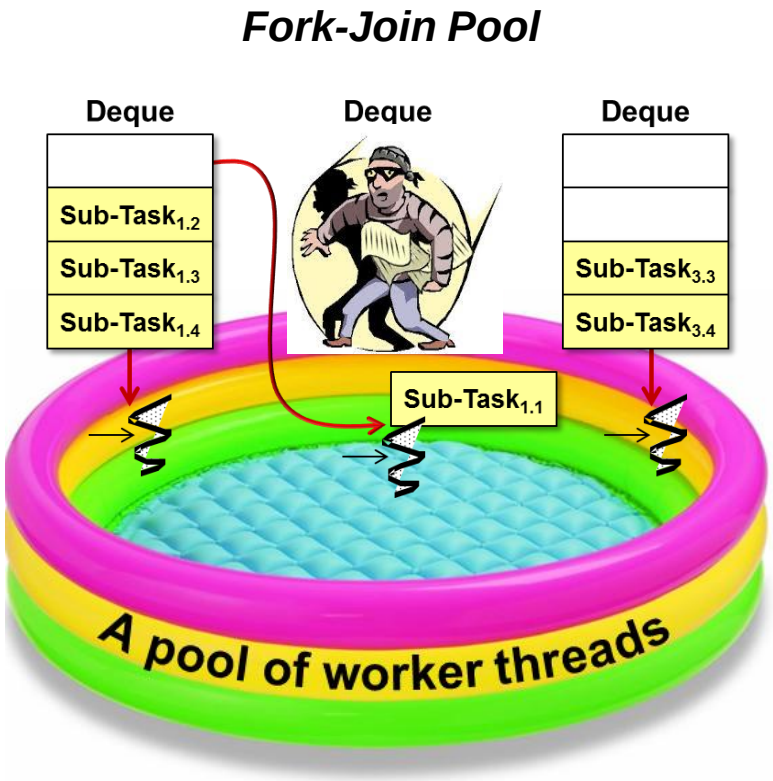
The Java Fork-Join Pool Internals

Work Stealing

Douglas C. Schmidt

Learning Objectives in This Part of the Lesson

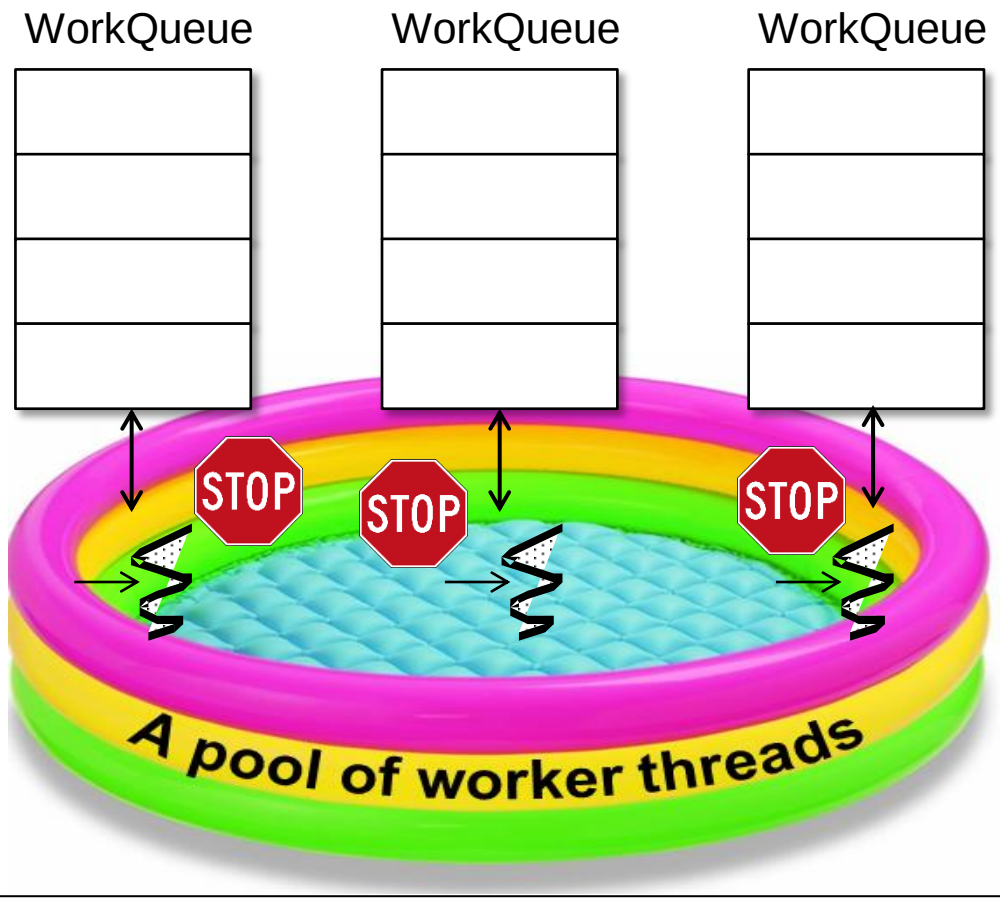
- Know how the Java fork-join framework implements worker threads
- Recognize how the Java fork-join framework implements work stealing



Working Stealing in a Java Fork-Join Pool

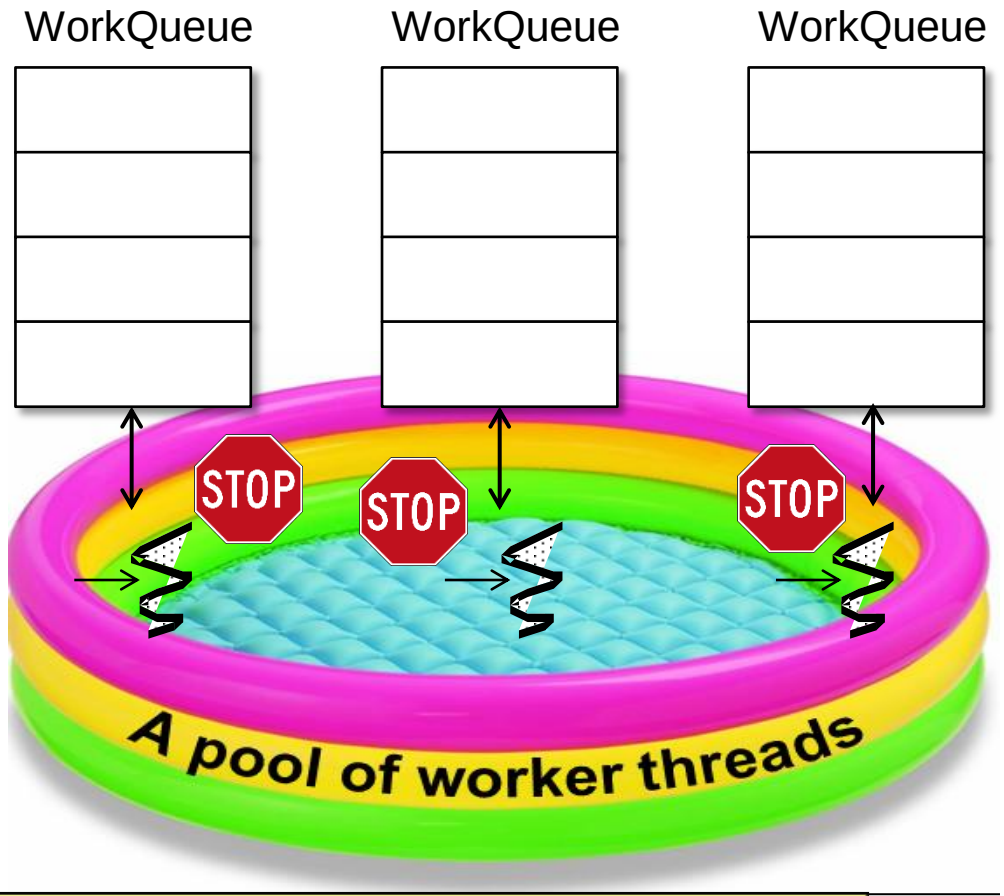
Work Stealing in a Java Fork-Join Pool

- Worker threads only block if there are no tasks available to run.



Work Stealing in a Java Fork-Join Pool

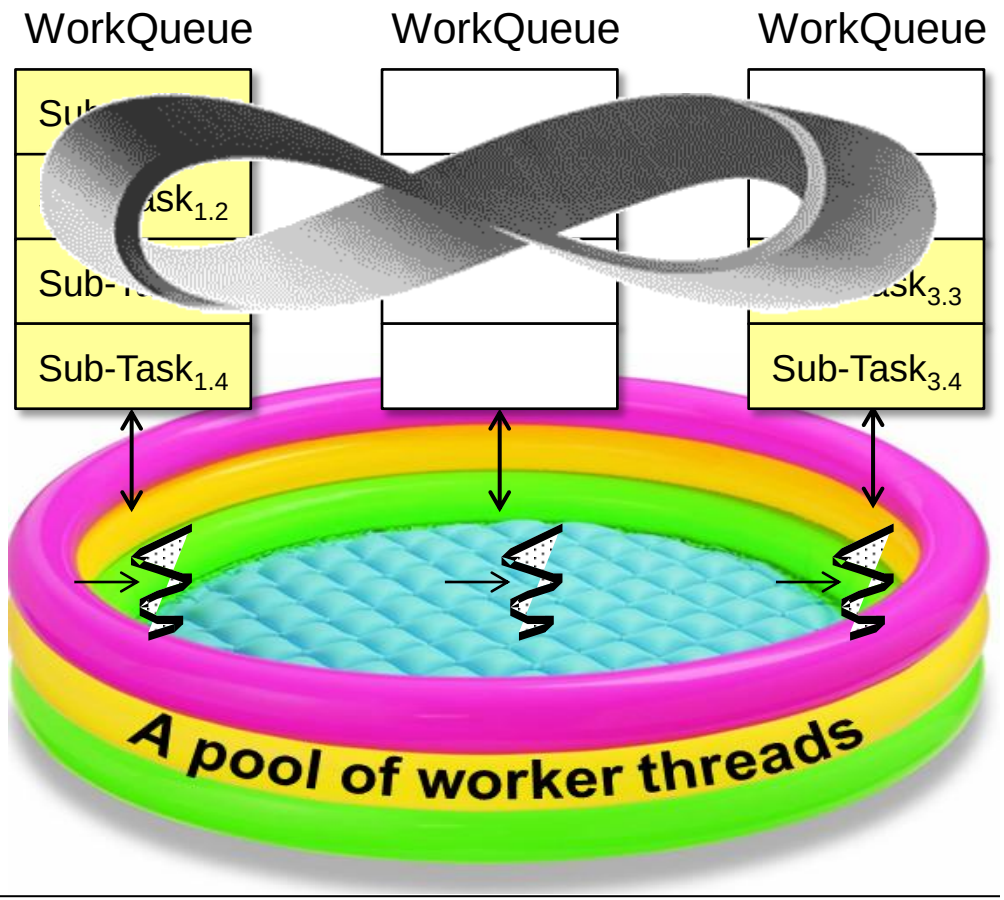
- Worker threads only block if there are no tasks available to run.
  - Blocking threads and cores is costly on modern processors.



See Doug Lea's talk at www.youtube.com/watch?v=sq0MX3fHkro

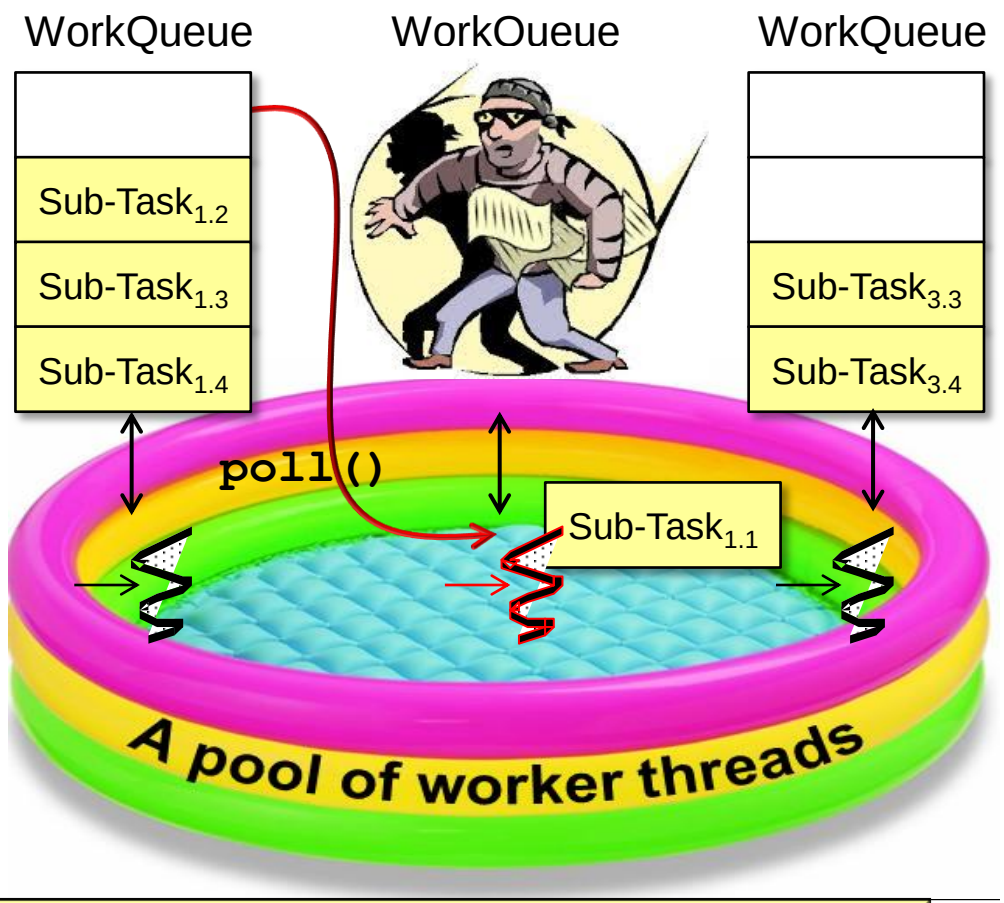
Work Stealing in a Java Fork-Join Pool

- Worker threads only block if there are no tasks available to run.
  - Blocking threads and cores is costly on modern processors.
  - Each worker thread therefore checks other deques in the pool to find other tasks to run.



Work Stealing in a Java Fork-Join Pool

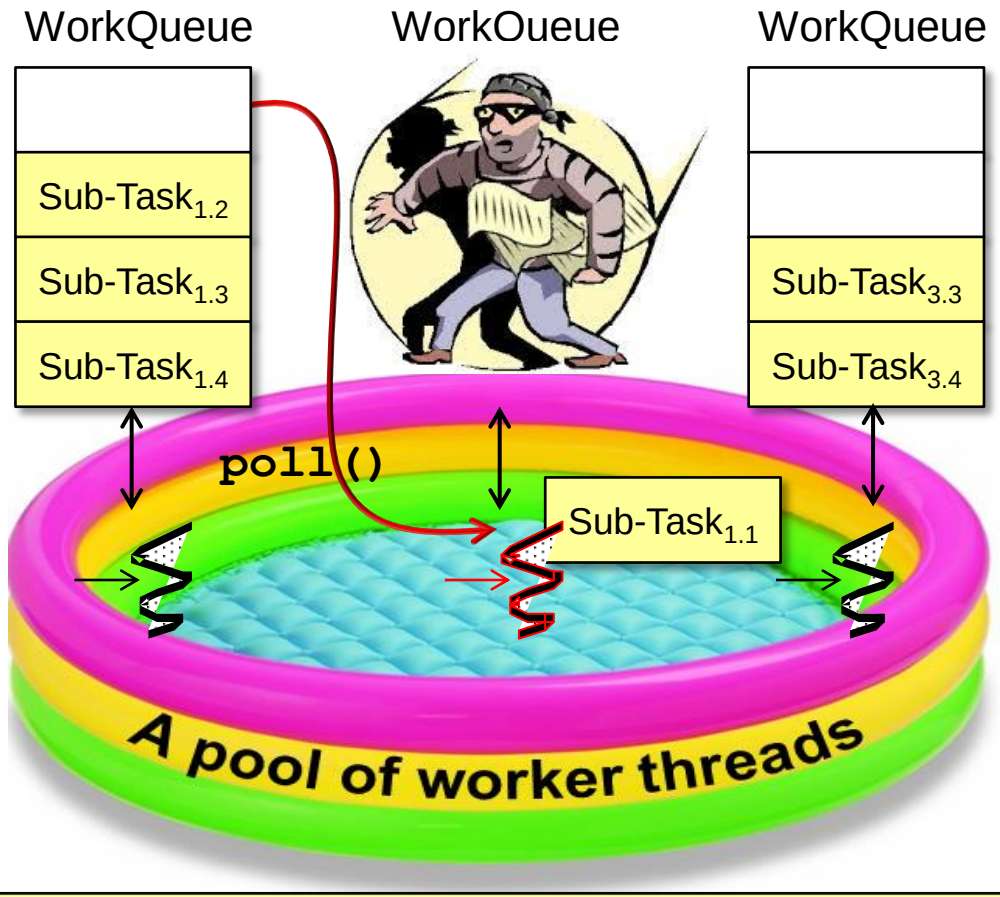
- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques.



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

Work Stealing in a Java Fork-Join Pool

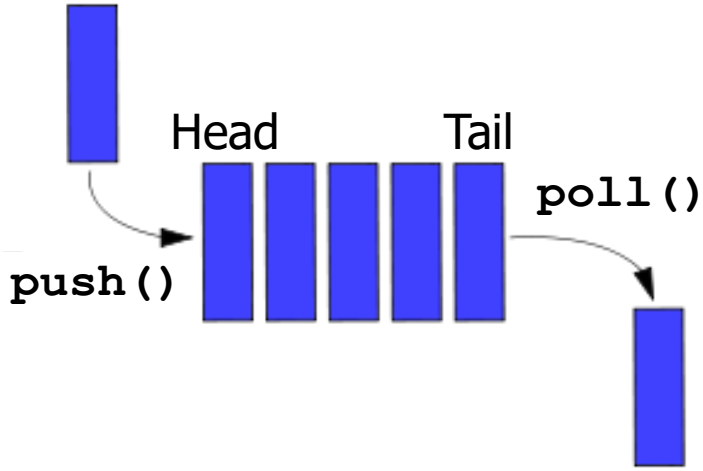
- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques.

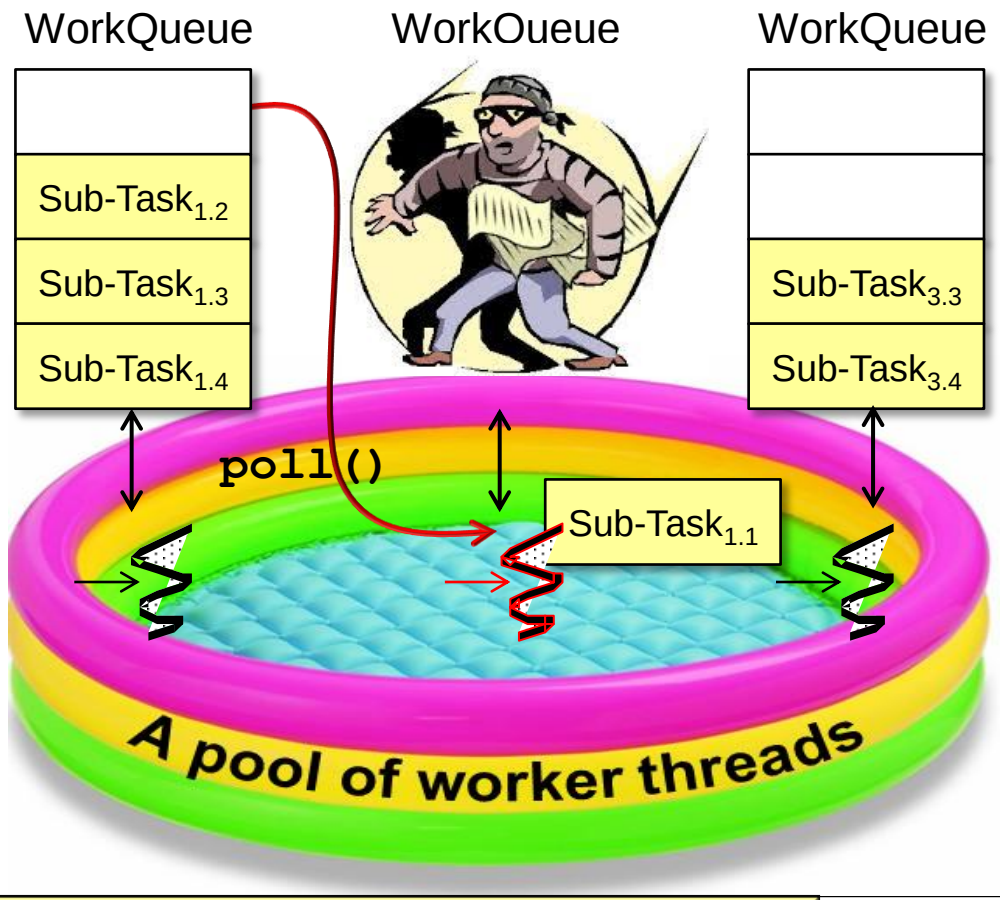


The worker thread deque to steal from is selected randomly to lower contention.

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order.

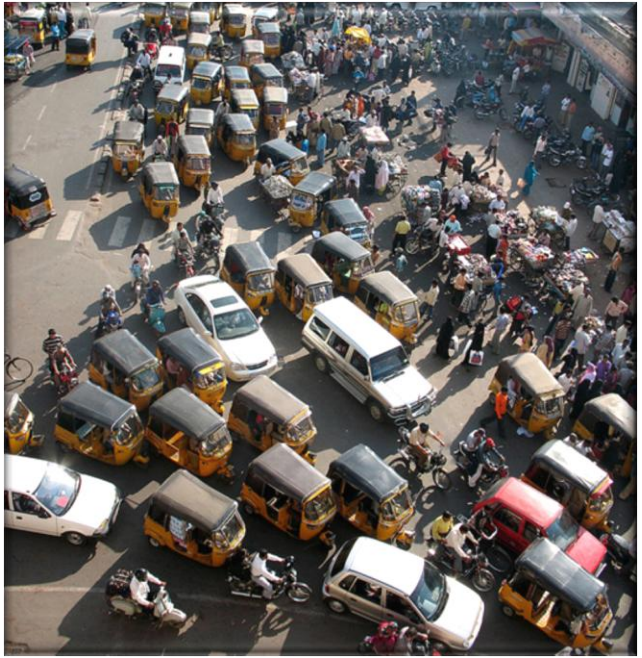


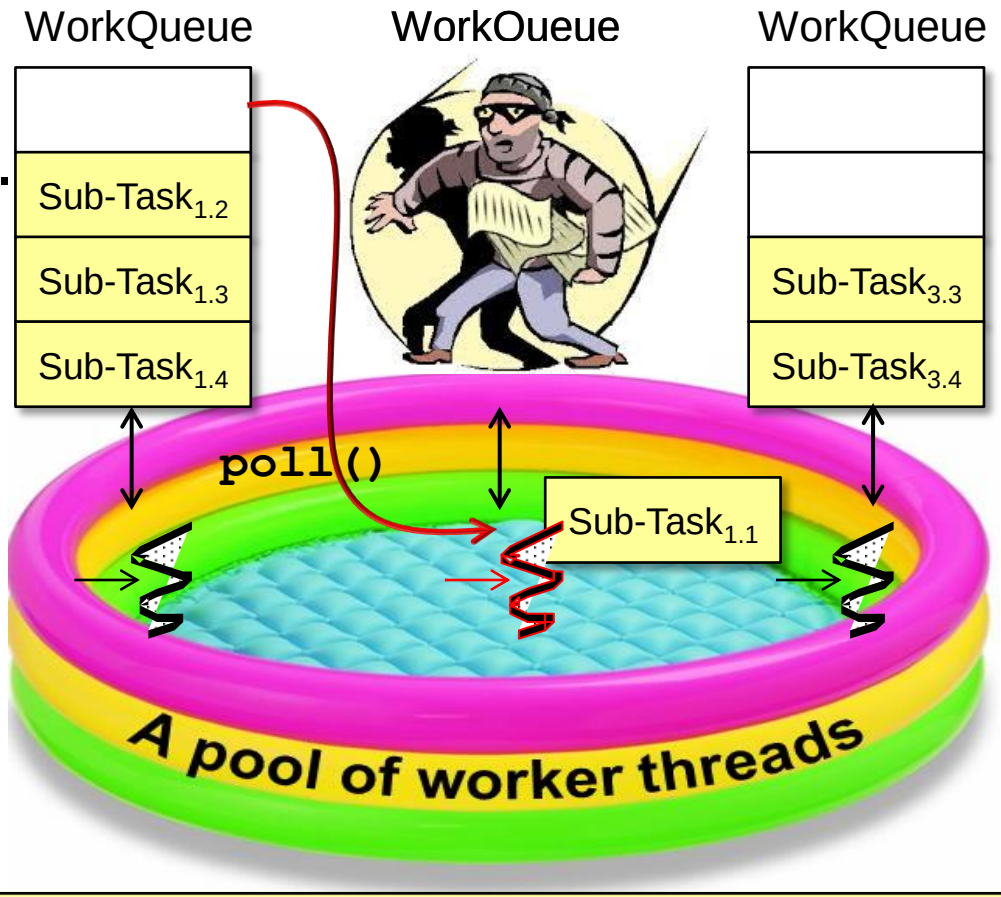


See en.wikipedia.org/wiki/FIFO_(computing_and_electronics)

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order.
  - This minimizes contention with worker thread owning the deque.

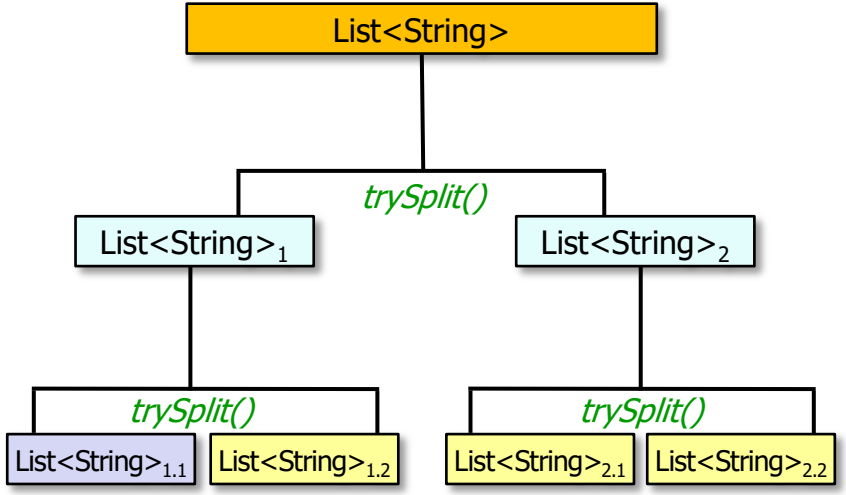


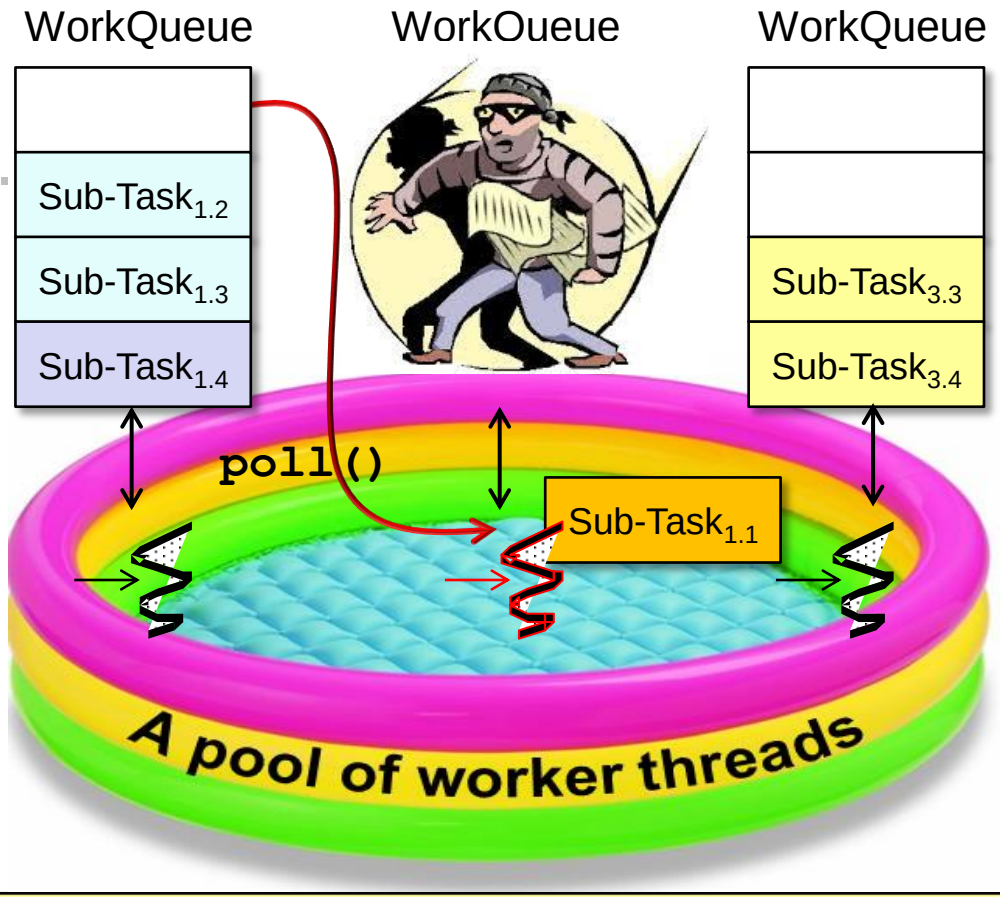


See www.ibm.com/support/knowledgecenter/en/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/resolving.html

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order.
  - This minimizes contention with worker thread owning the deque.
  - An older stolen task may provide a larger unit of work.

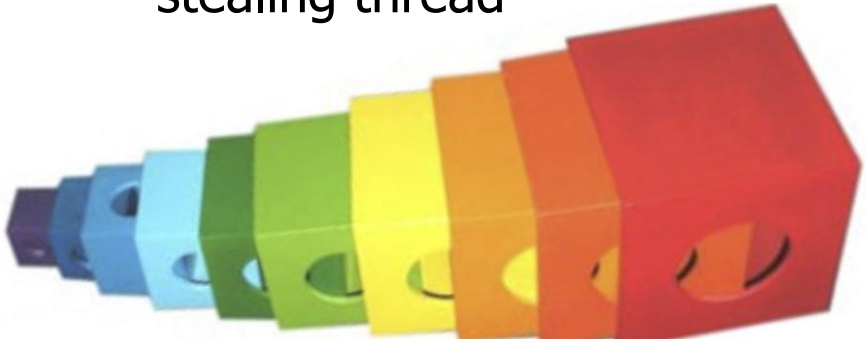


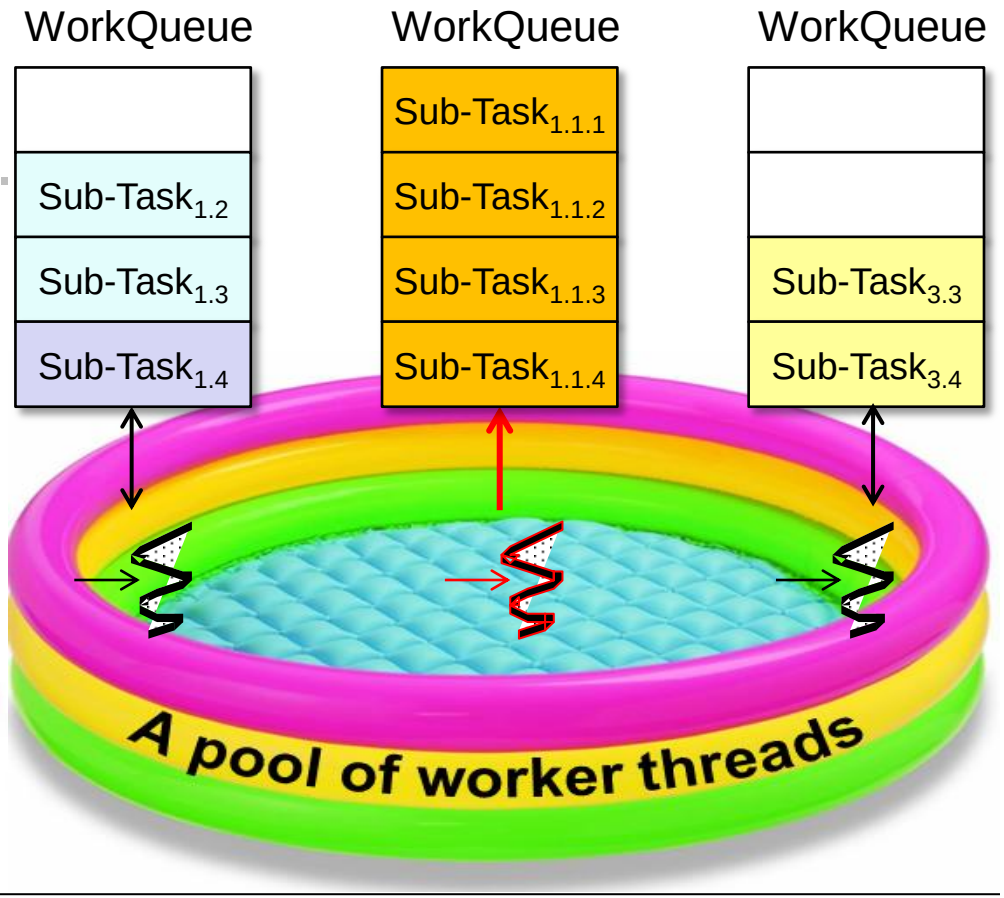


This behavior stems from the "divide and conquer" nature of fork-join tasks that split evenly.

Work Stealing in a Java Fork-Join Pool

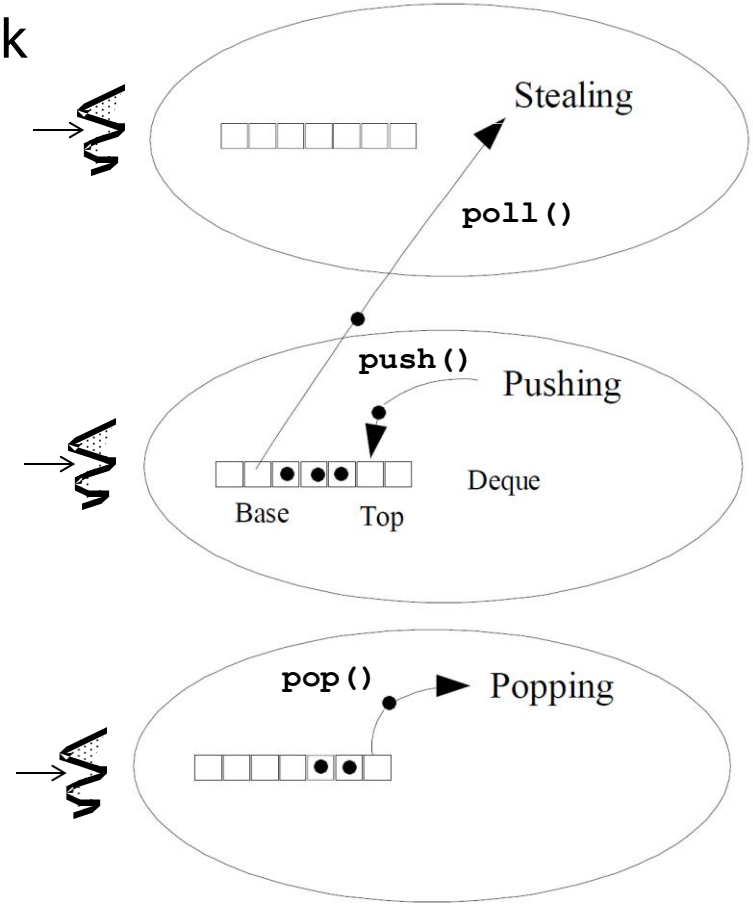
- Tasks are stolen in FIFO order.
  - This minimizes contention with worker thread owning the deque.
  - An older stolen task may provide a larger unit of work.
    - Enables further recursive decompositions by the stealing thread





Work Stealing in a Java Fork-Join Pool

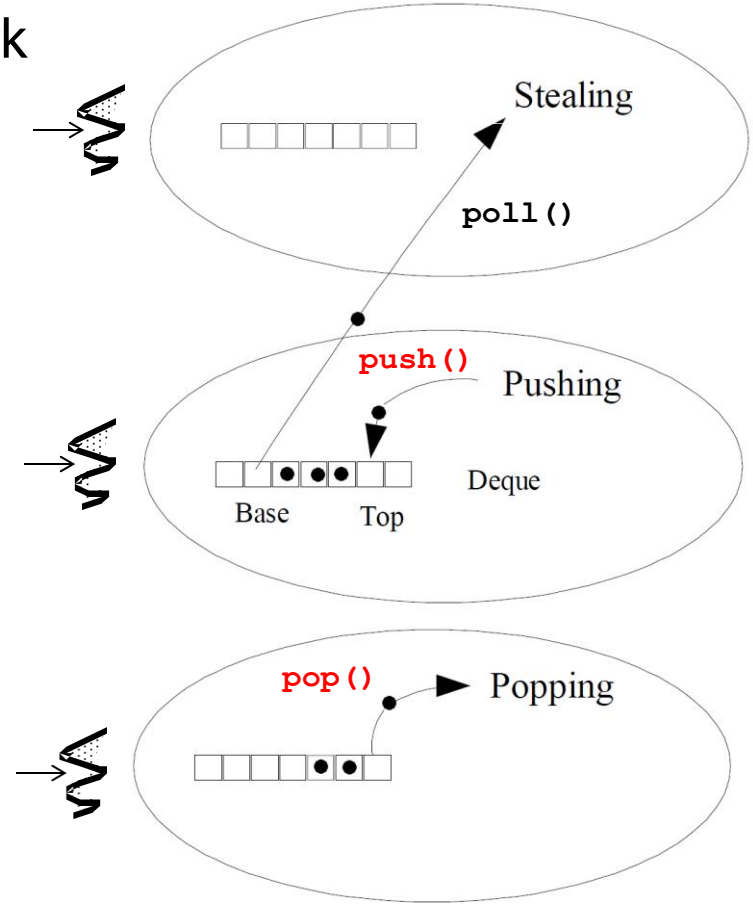
- The WorkQueue deque that implements work stealing minimizes locking contention.



See www.dre.vanderbilt.edu/~schmidt/PDF/work-stealing-deque.pdf

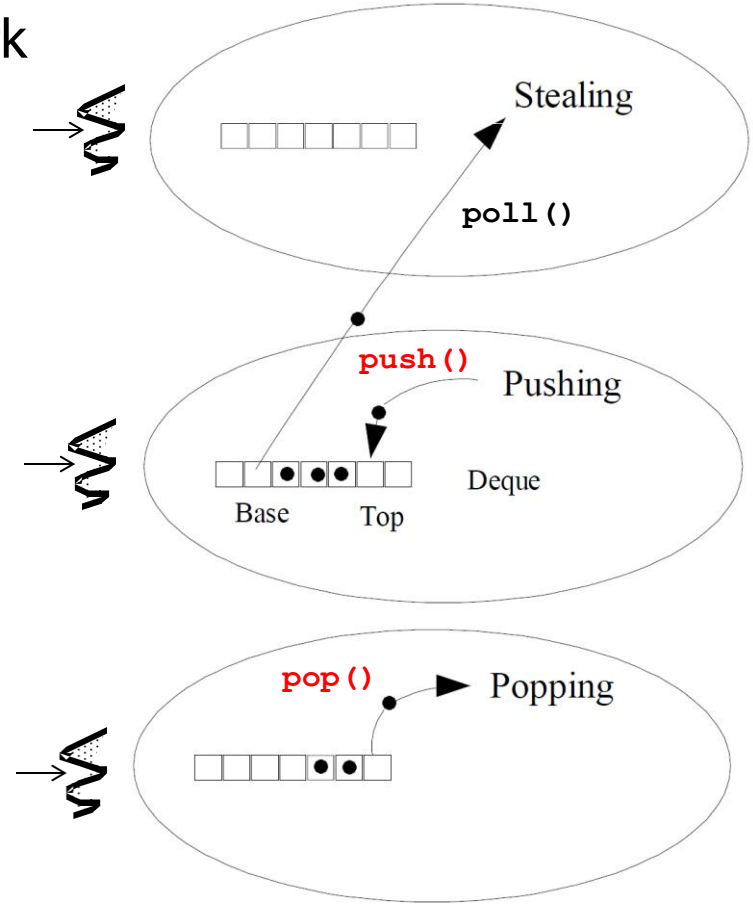
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work stealing minimizes locking contention.
  - push() and pop() are only called by the owning worker thread.



Work Stealing in a Java Fork-Join Pool

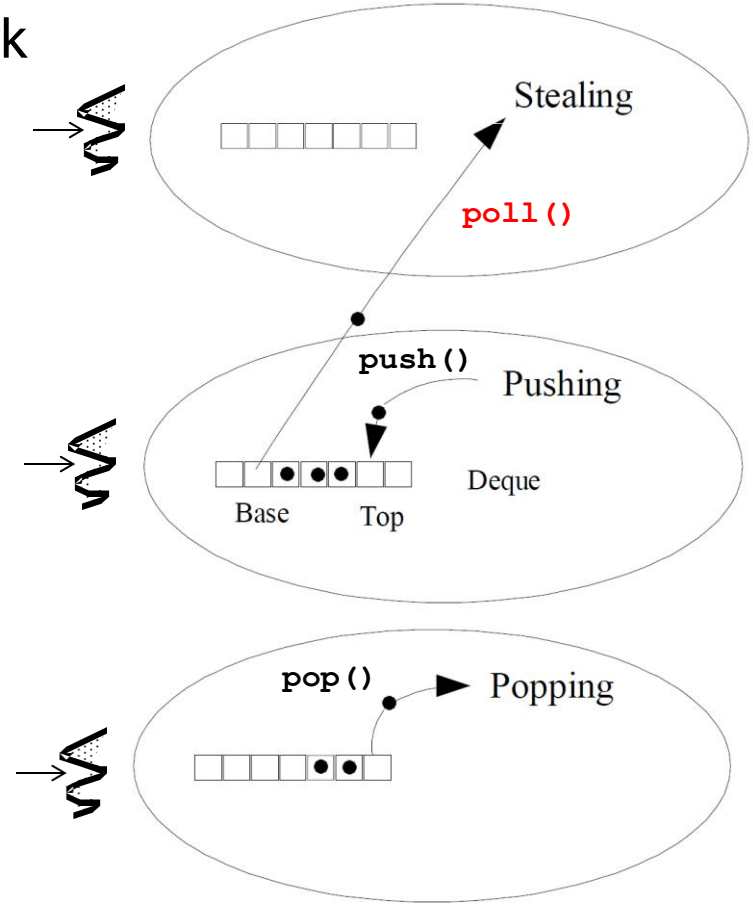
- The WorkQueue deque that implements work stealing minimizes locking contention.
  - push() and pop() are only called by the owning worker thread.
    - These methods use wait-free "compare-and-swap" (CAS) operations.



See en.wikipedia.org/wiki/Compare-and-swap

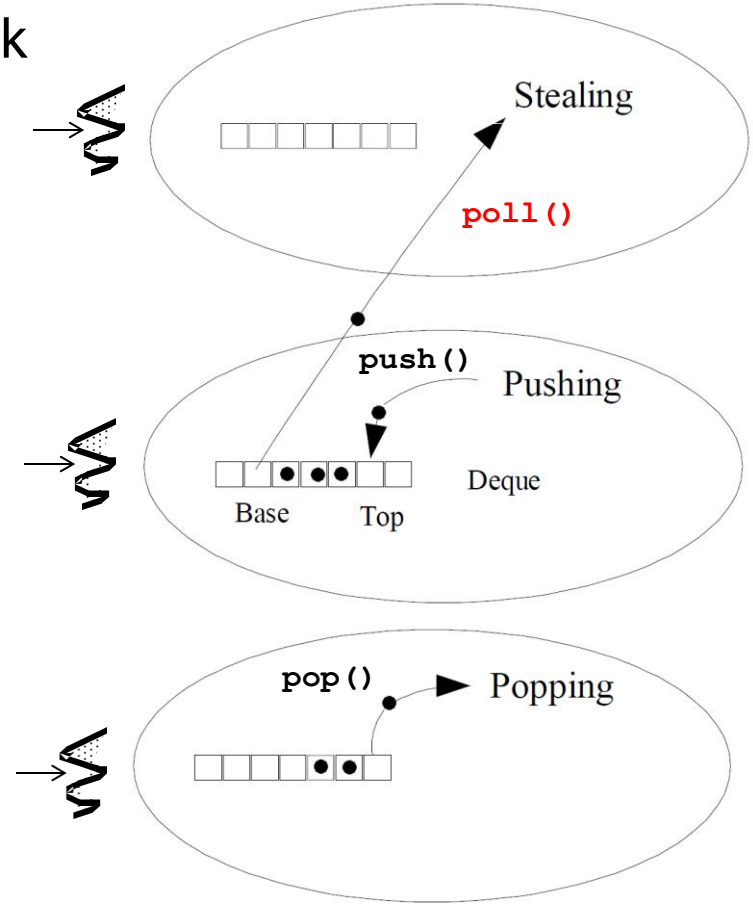
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work stealing minimizes locking contention.
  - push() and pop() are only called by the owning worker thread.
  - poll() may be called from another worker thread to "steal" a (sub)task.



Work Stealing in a Java Fork-Join Pool

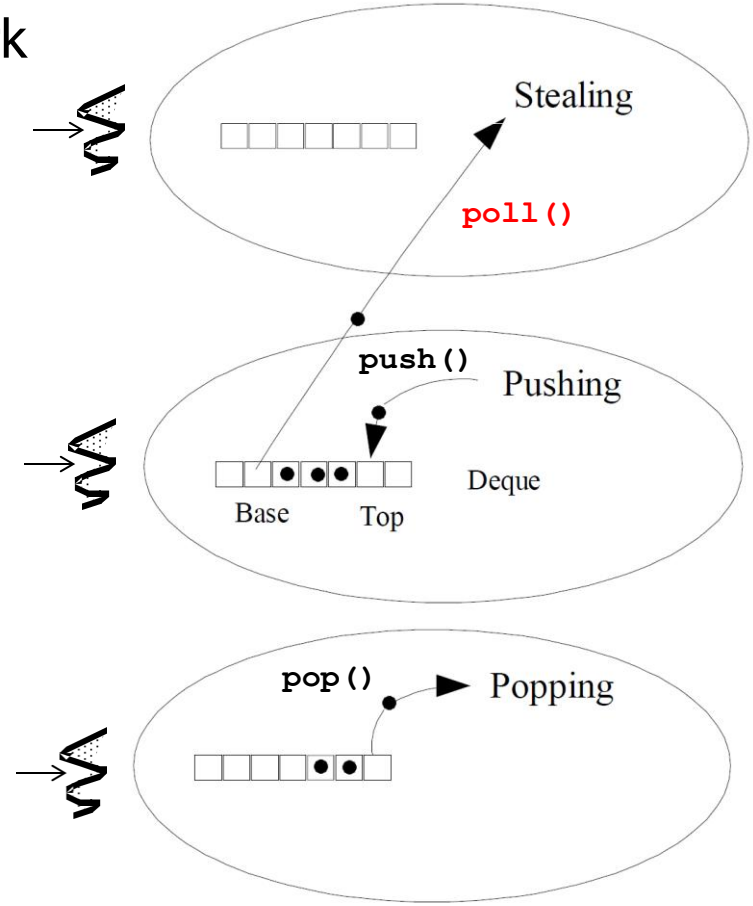
- The WorkQueue deque that implements work stealing minimizes locking contention.
  - push() and pop() are only called by the owning worker thread.
  - poll() may be called from another worker thread to "steal" a (sub)task.
    - May not always be wait-free



See gee.cs.oswego.edu/dl/papers/fj.pdf

Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work stealing minimizes locking contention.
  - push() and pop() are only called by the owning worker thread.
  - poll() may be called from another worker thread to "steal" a (sub)task.
    - May not always be wait-free
      - See "Implementation Overview" comments in the ForkJoinPool source code for details



See [java8/util/concurrent/ForkJoinPool.java]

The End