Java Fork-Join Framework Internals: Worker Threads

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

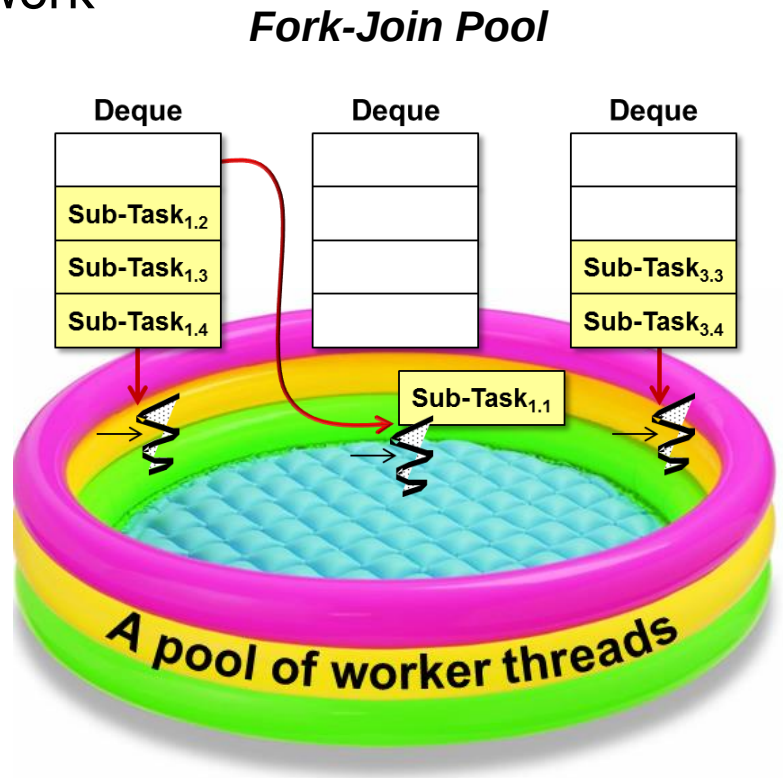
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

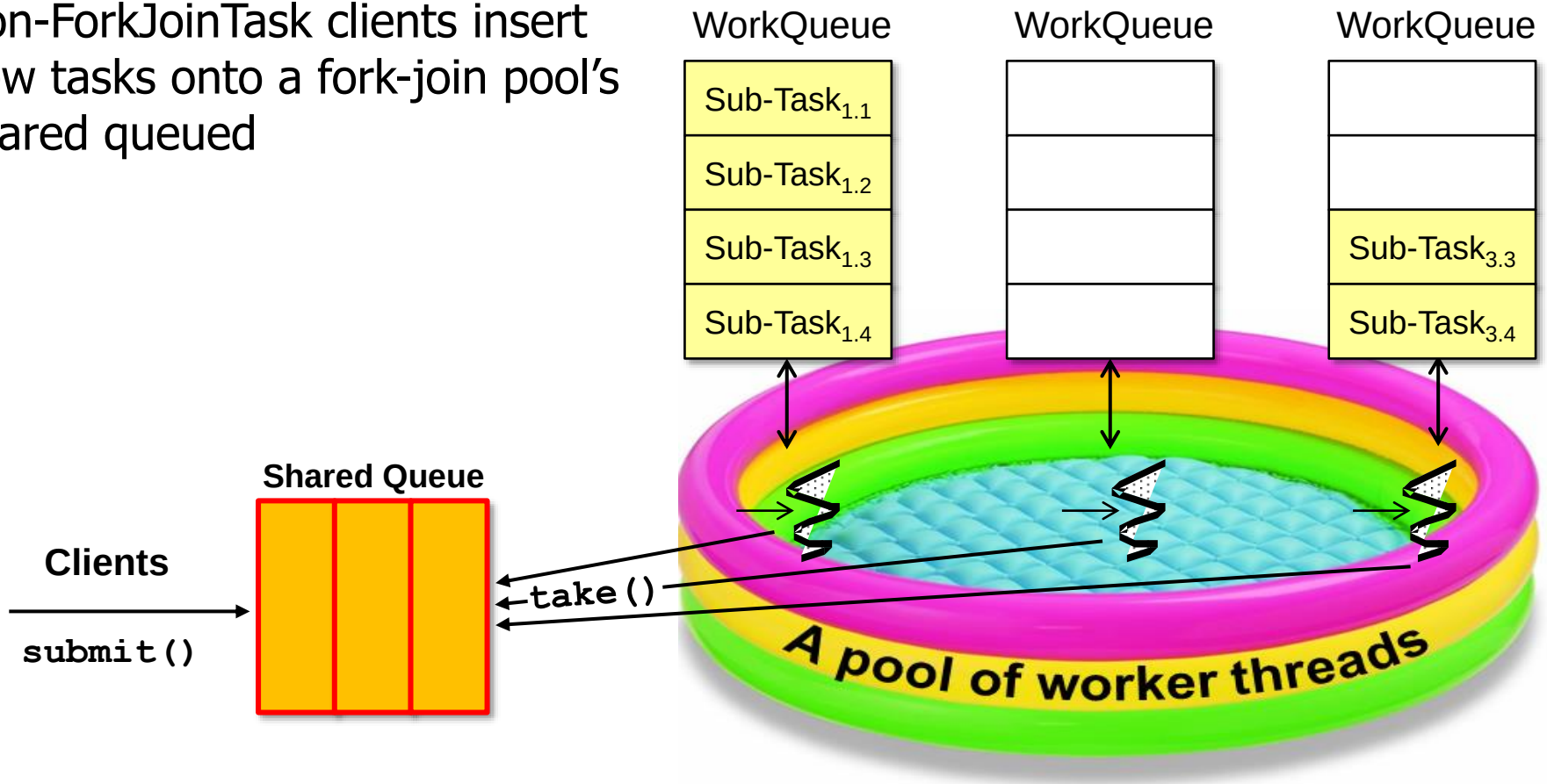
- Understand how the Java fork-join framework implements worker threads



Worker Threads in a Java Fork-Join Pool

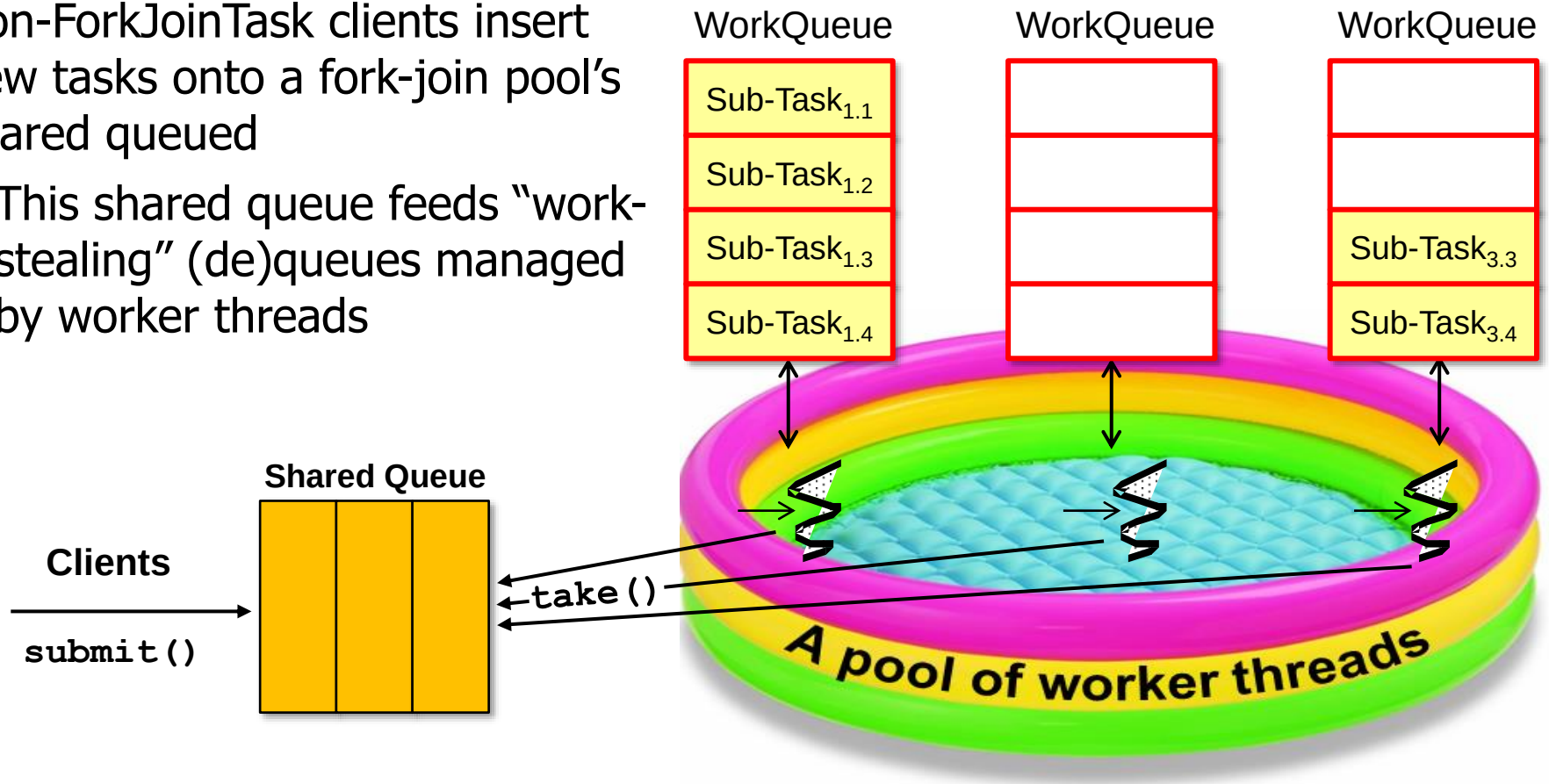
Worker Threads in a Java Fork-Join Pool

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



Worker Threads in a Java Fork-Join Pool

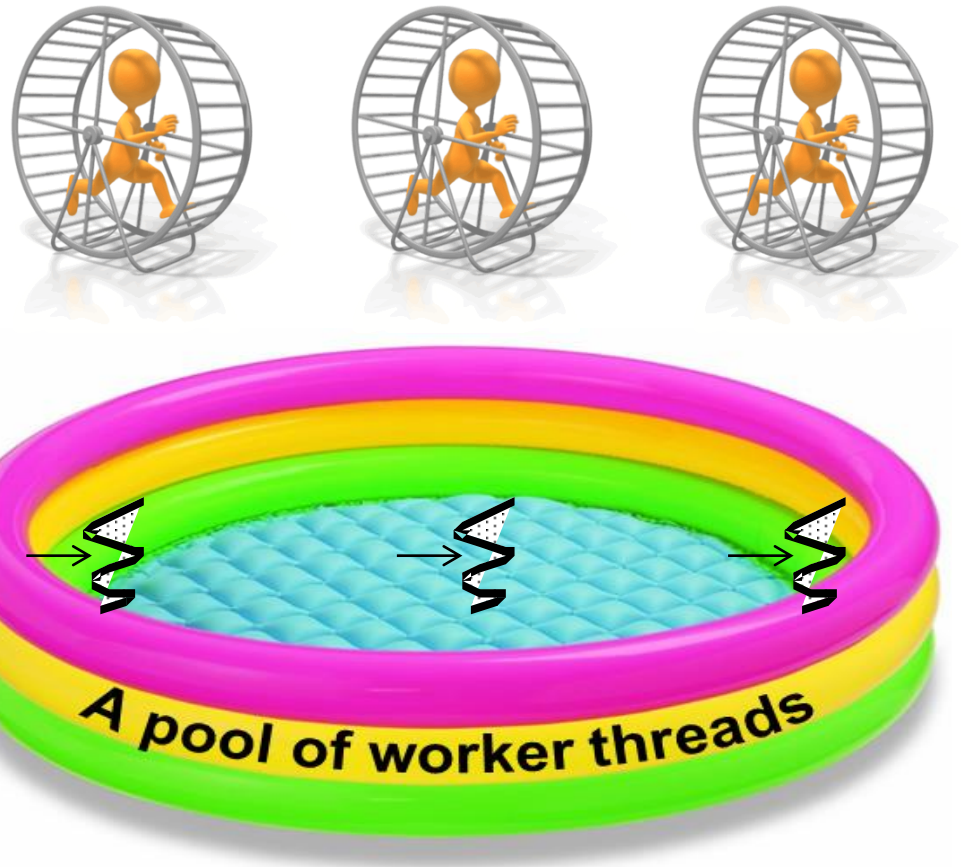
- Non-ForkJoinTask clients insert new tasks onto a fork-join pool's shared queued
  - This shared queue feeds "work-stealing" (de)queues managed by worker threads



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

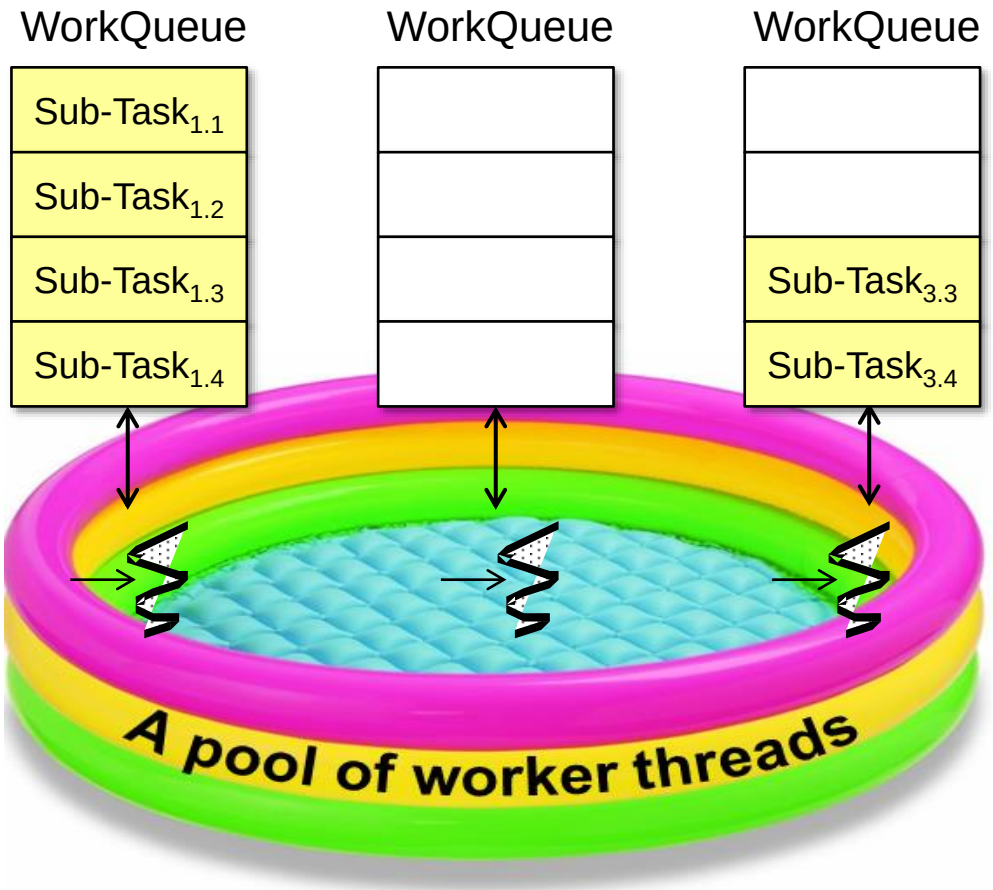


Worker Threads in a Java Fork-Join Pool

- 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



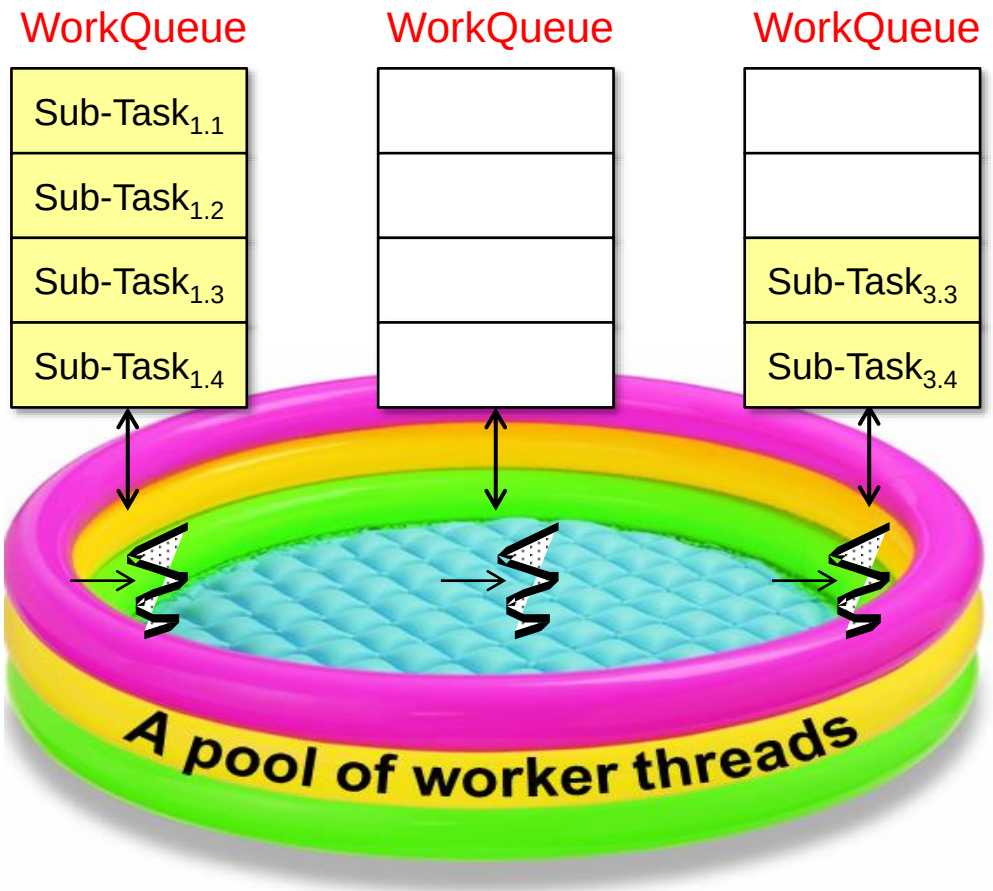
See en.wikipedia.org/wiki/Double-ended_queue

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

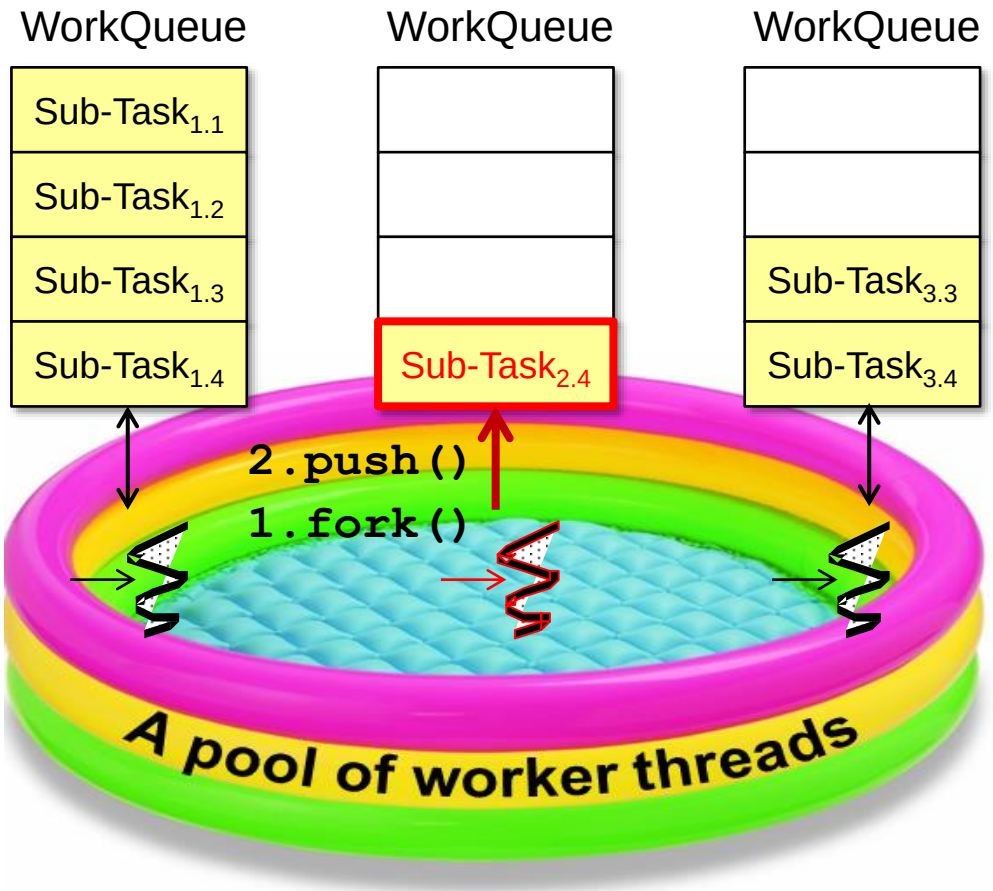
- WorkQueue(ForkJoinPool,ForkJoinWorkerThread,int,int)
- queueSize():int
- isEmpty():boolean
- push(ForkJoinTask<?>):void
- growArray():ForkJoinTask<?>
- pop():ForkJoinTask<?>
- pollAt(int):ForkJoinTask<?>
- poll():ForkJoinTask<?>
- peek():ForkJoinTask<?>
- cancelAll():void
- pollAndExecAll():void
-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



See java8/util/concurrent/ForkJoinPool.java

Worker Threads in a Java Fork-Join Pool

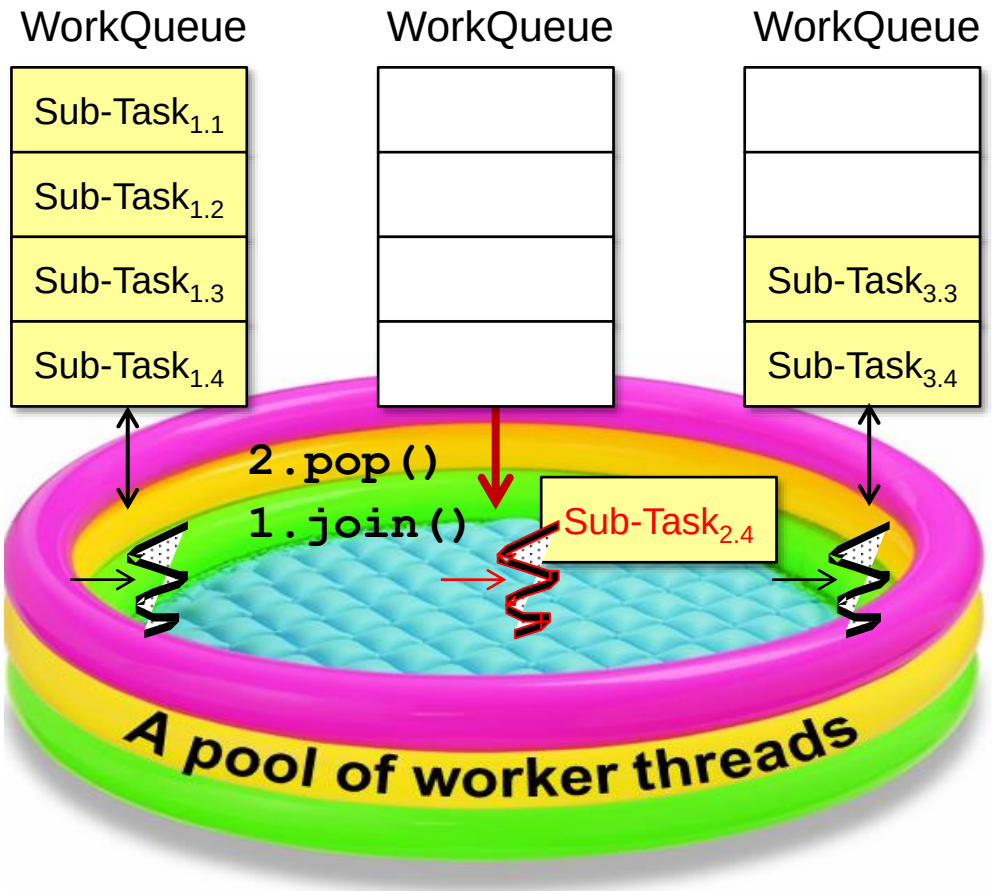
- If a task run by a worker thread calls fork() the new task is pushed on the head of the worker's deque



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

Worker Threads in a Java Fork-Join Pool

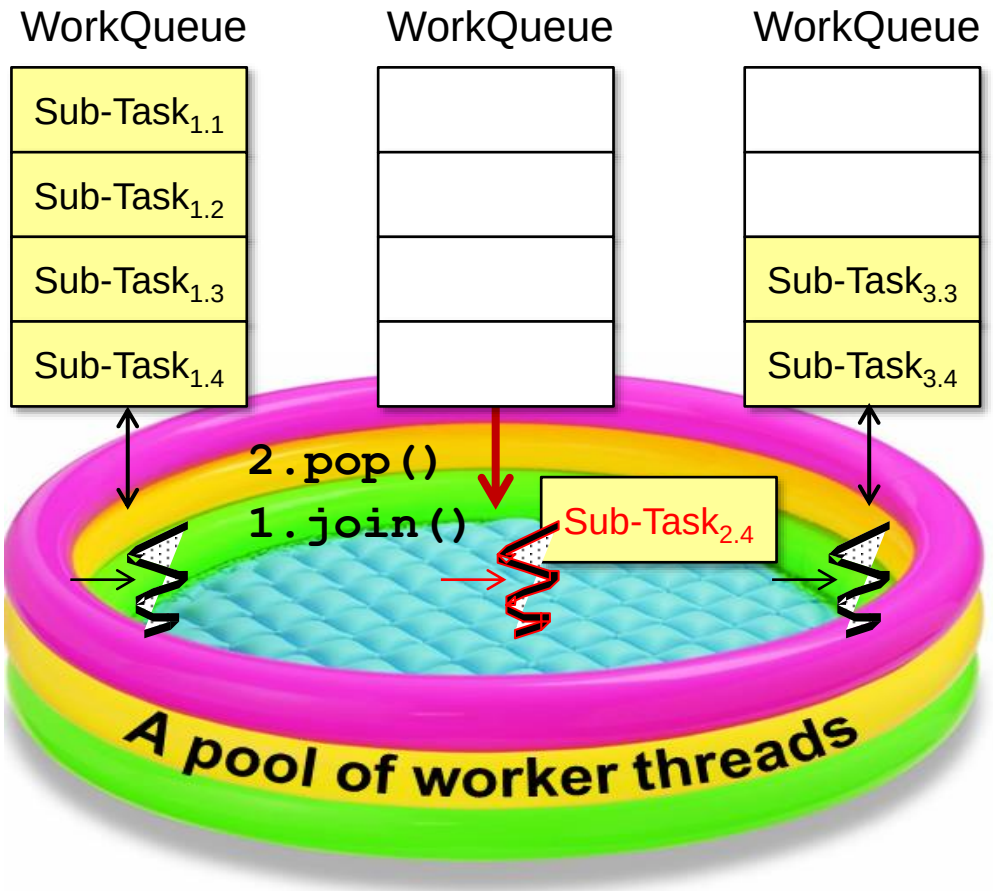
- If a task run by a worker thread calls fork() the new task is pushed on the head of the worker's deque
  - A worker thread processes its deque in LIFO order



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

Worker Threads in a Java Fork-Join Pool

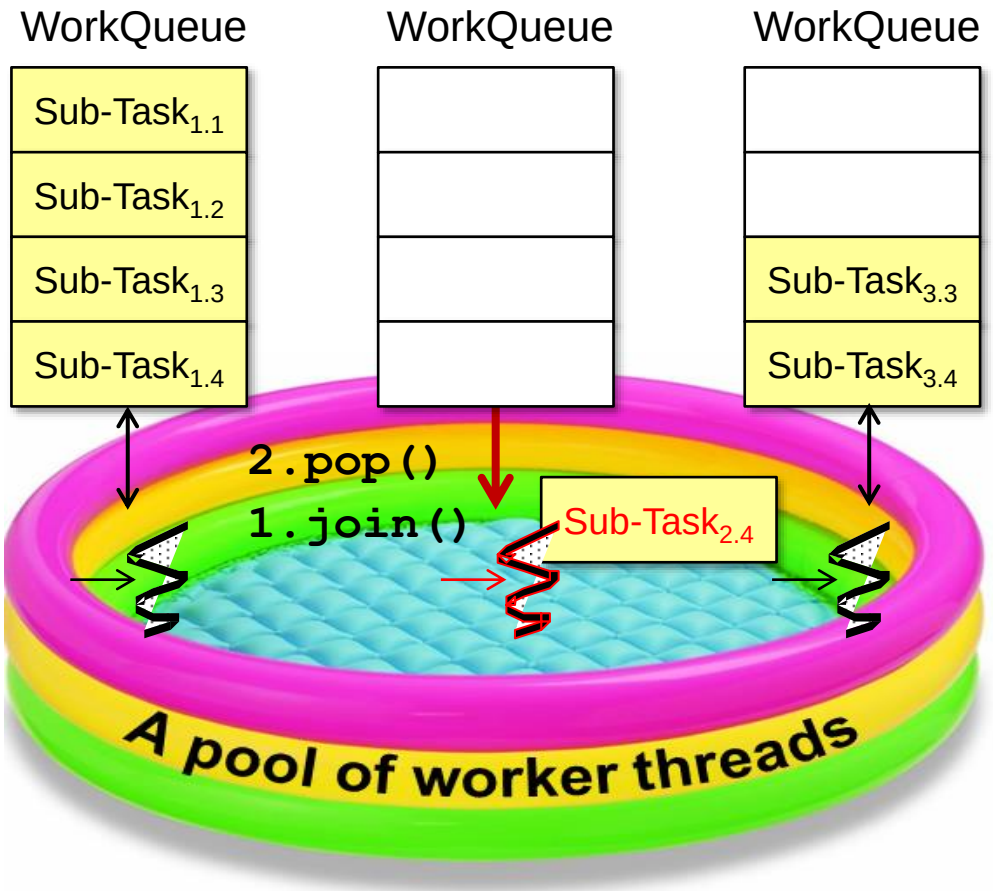
- If a task run by a worker thread calls fork() the new task is pushed on the head of the worker's deque
  - A worker thread processes its deque in LIFO order, i.e.
    - A task pop'd from the head of a deque is run to completion



See en.wikipedia.org/wiki/Run_to_completion_scheduling

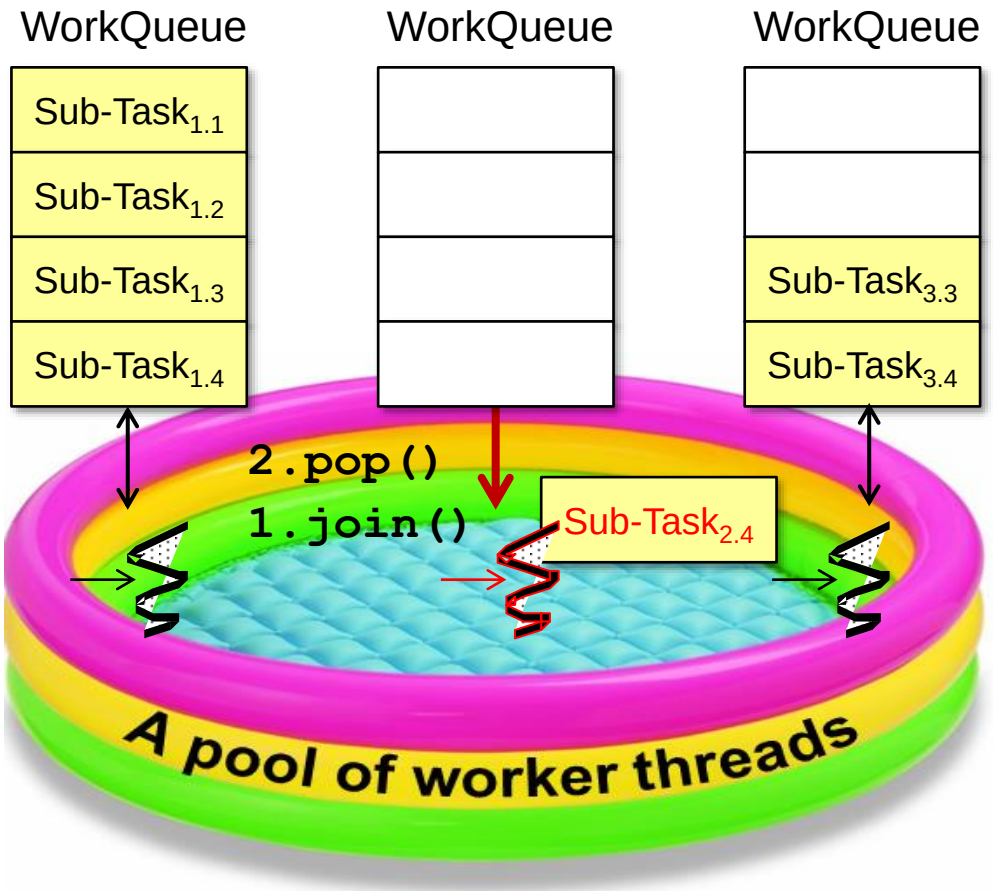
Worker Threads in a Java Fork-Join Pool

- 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, i.e.
    - A task pop'd from the head of a deque is run to completion
    - join() "pitches in" to pop & execute (sub-)tasks



Worker Threads in a Java Fork-Join Pool

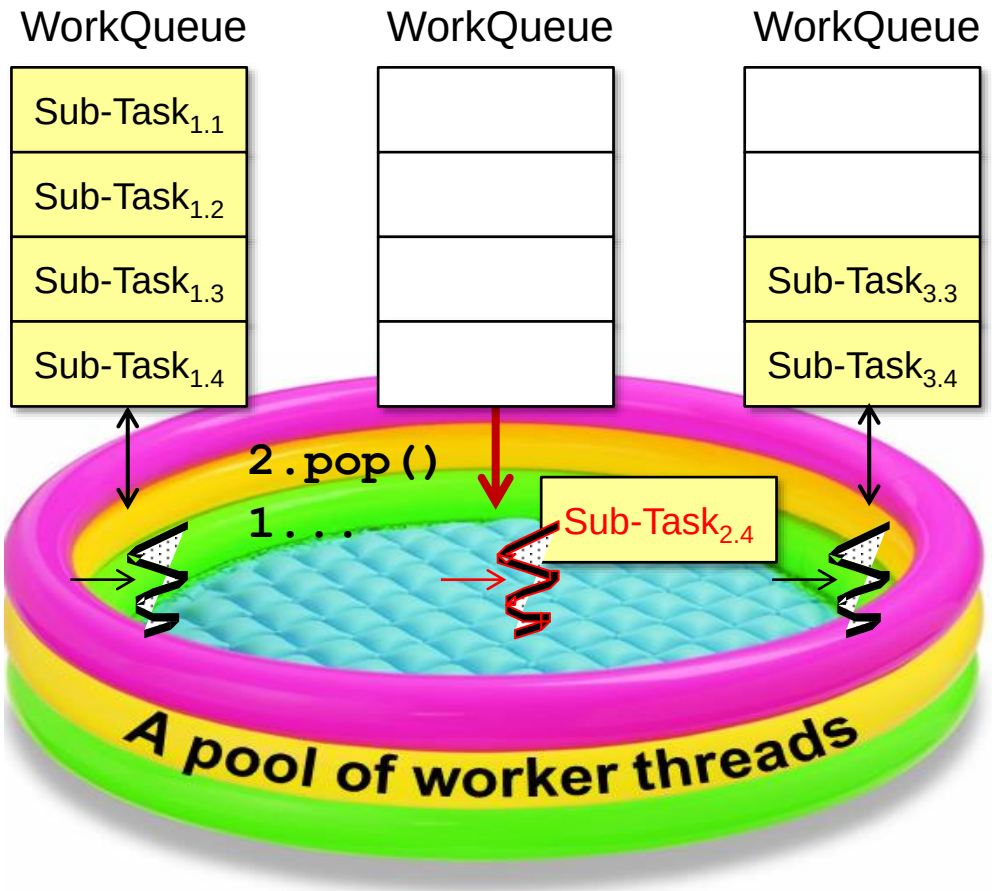
- If a task run by a worker thread calls fork() the new task is pushed on the head of the worker's deque
  - A worker thread processes its deque in LIFO order, i.e.
    - A task pop'd from the head of a deque is run to completion
    - join() "pitches in" to pop & execute (sub-)tasks



"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 & cache performance



See en.wikipedia.org/wiki/Locality_of_reference

End of Java Fork-Join Framework Internals: Worker Threads