Java Fork-Join Framework Internals: Work Stealing

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

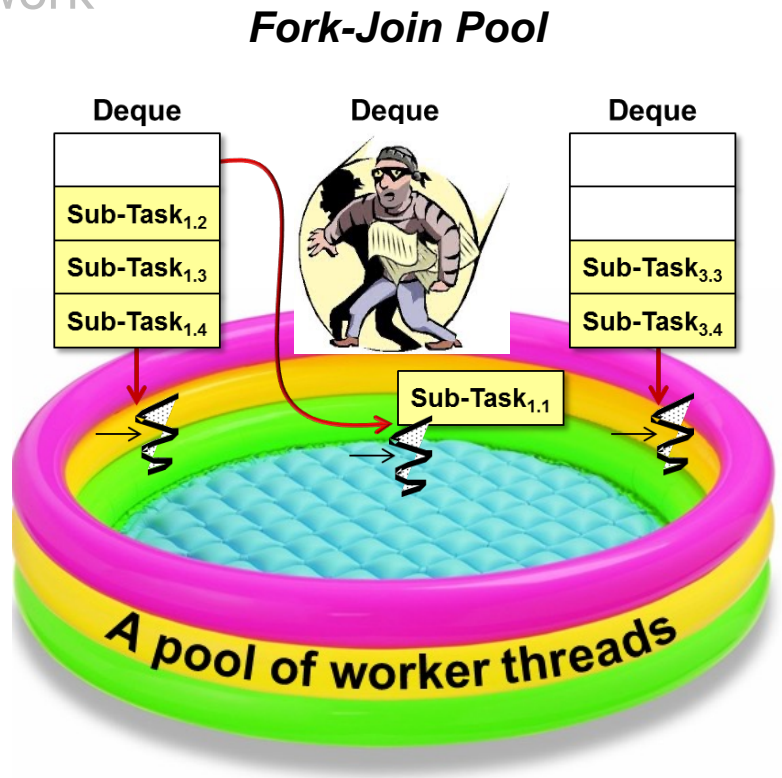
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

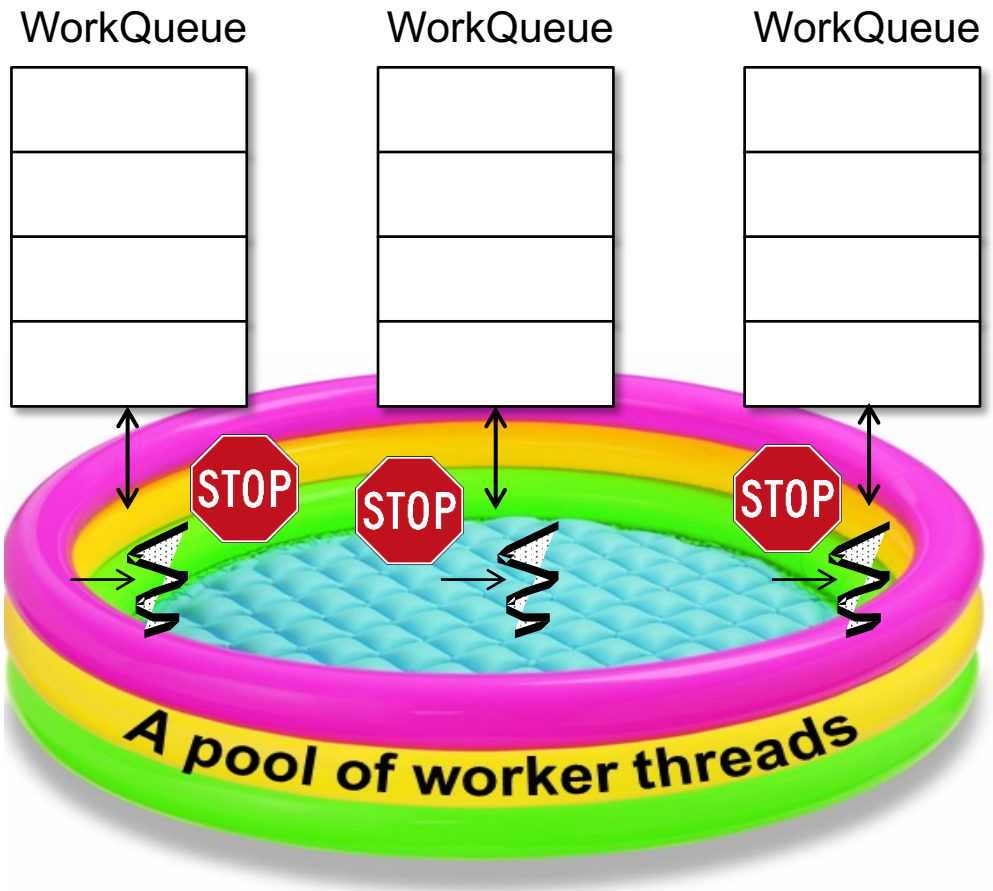
- Understand how the Java fork-join framework implements worker threads
- Understand how the Java fork-join framework implements work stealing



Working Stealing in a Java Fork-Join Pool

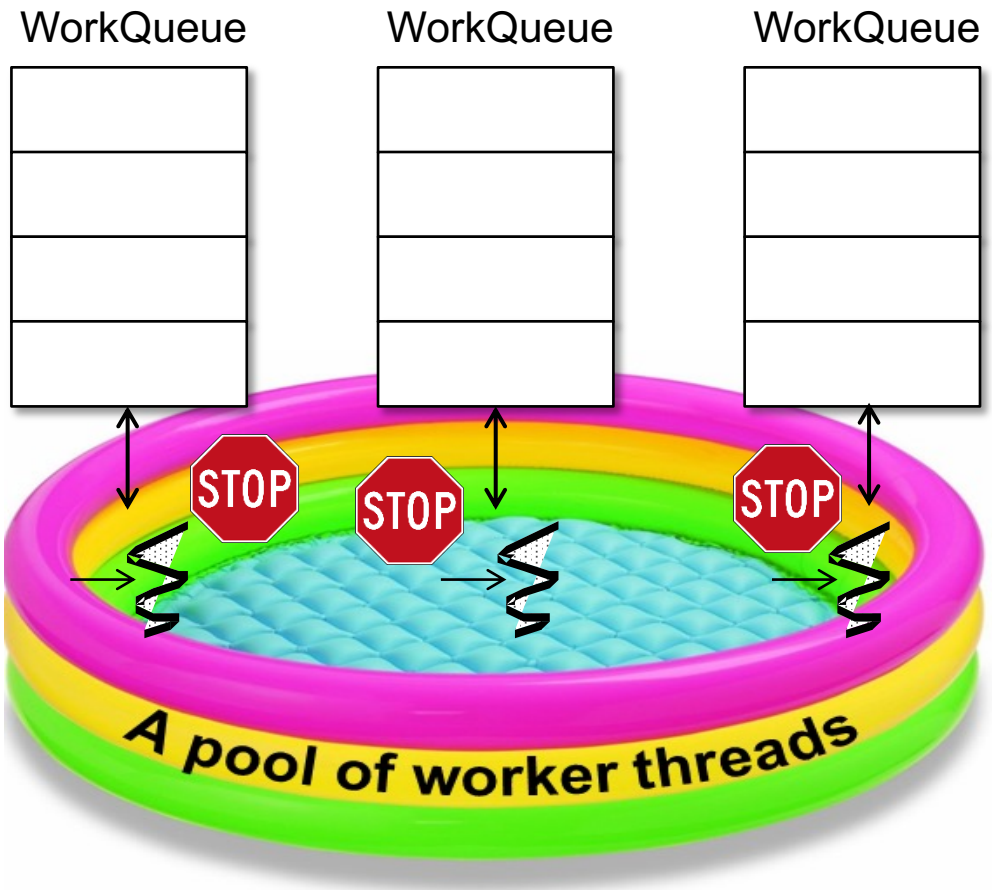
Work Stealing in a Java Fork-Join Pool

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



Work Stealing in a Java Fork-Join Pool

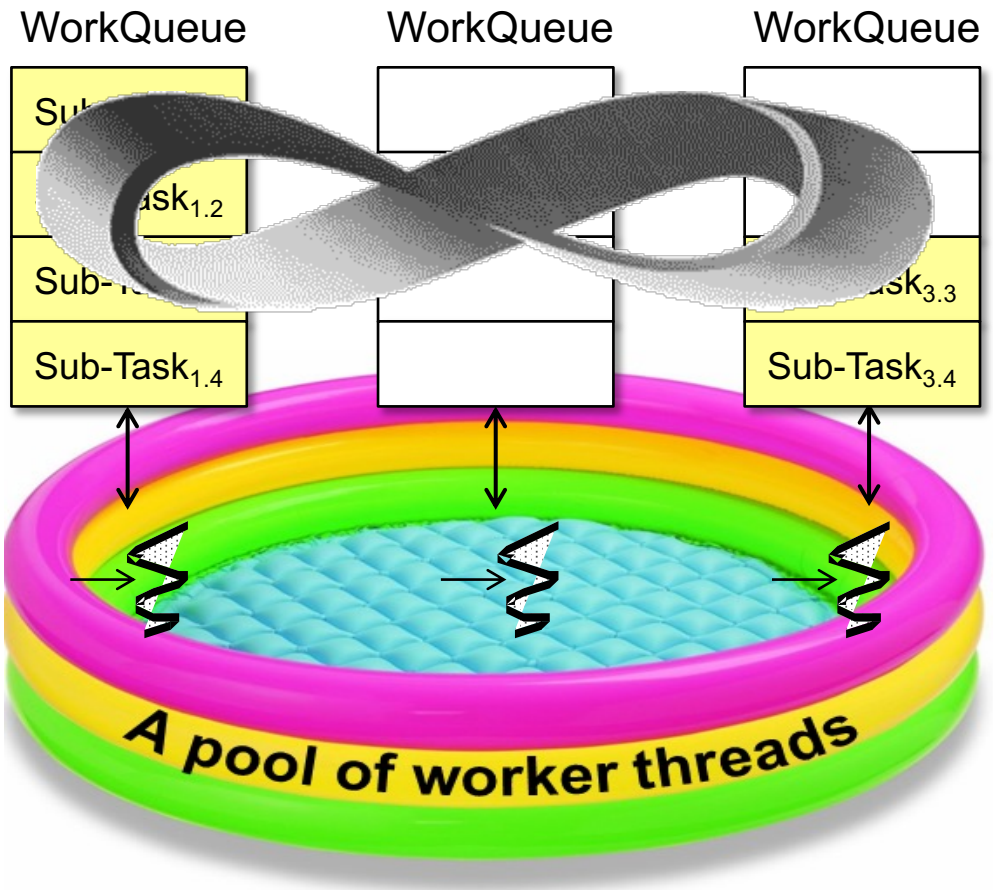
- Worker threads only block if there are no tasks available to run
  - Blocking threads & cores is costly on modern processors



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

Work Stealing in a Java Fork-Join Pool

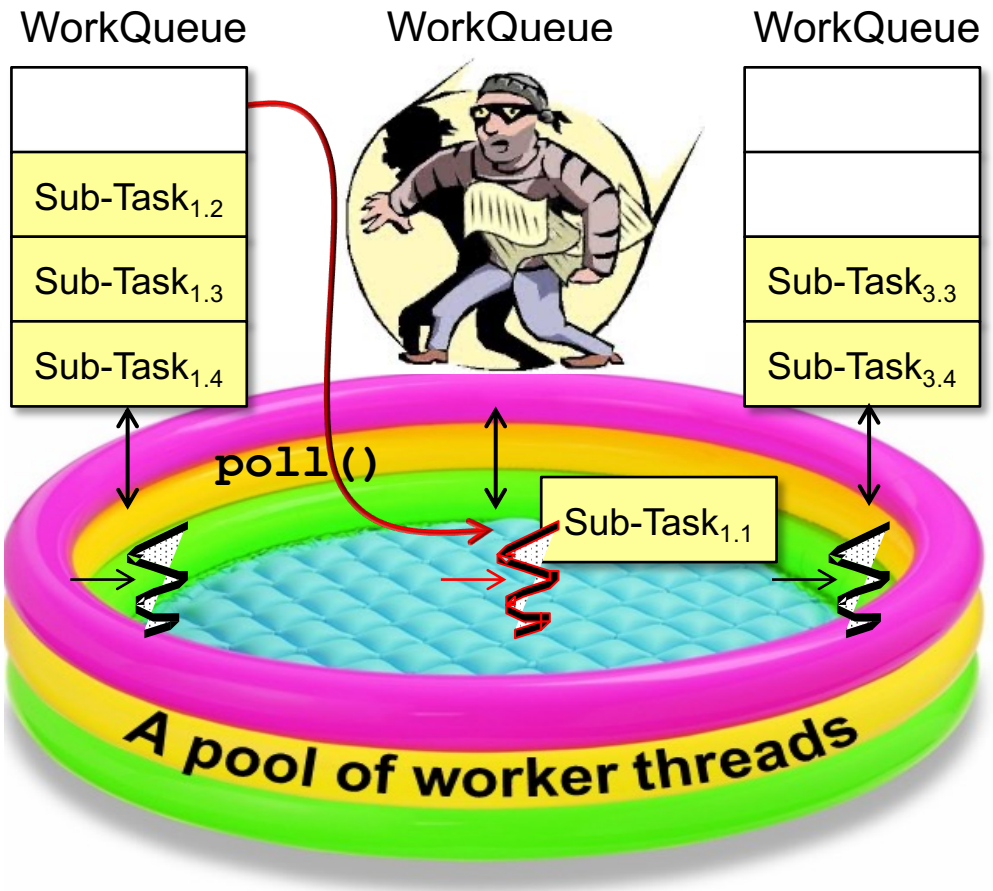
- Worker threads only block if there are no tasks available to run
  - Blocking threads & cores is costly on modern processors
  - A worker thread with an empty deque thus checks other deques in the pool to find tasks to run



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

Work Stealing in a Java Fork-Join Pool

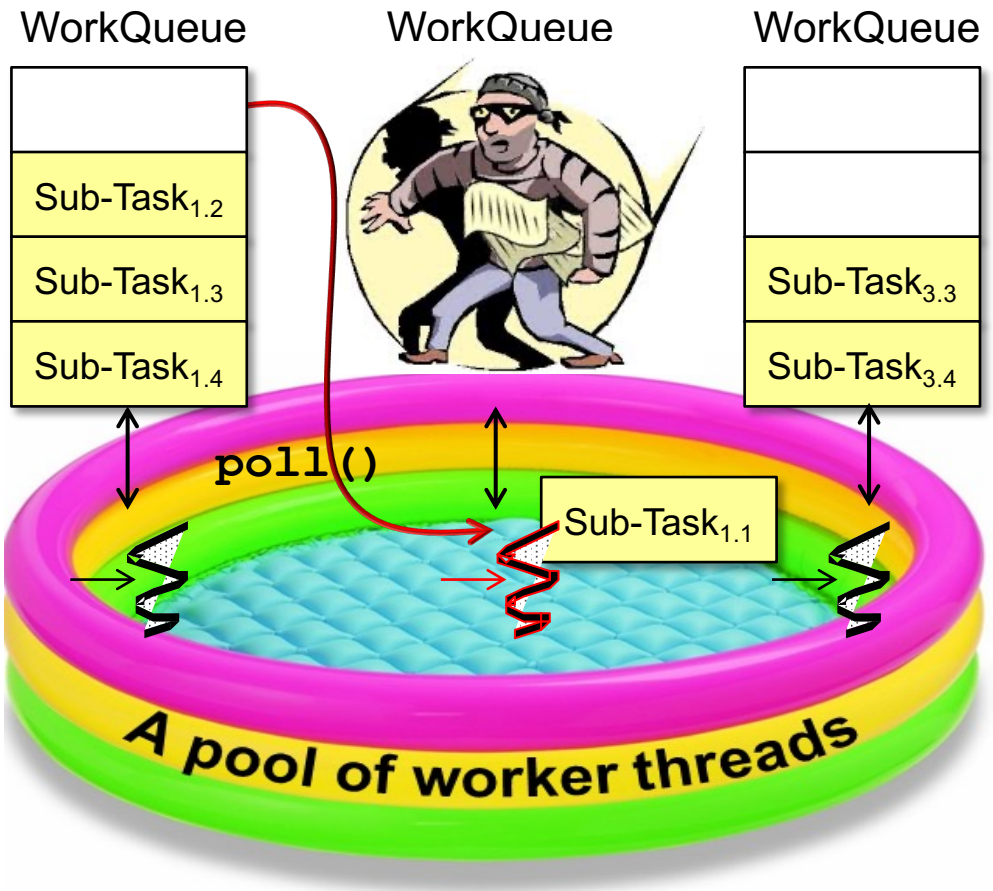
- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques



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

Work Stealing in a Java Fork-Join Pool

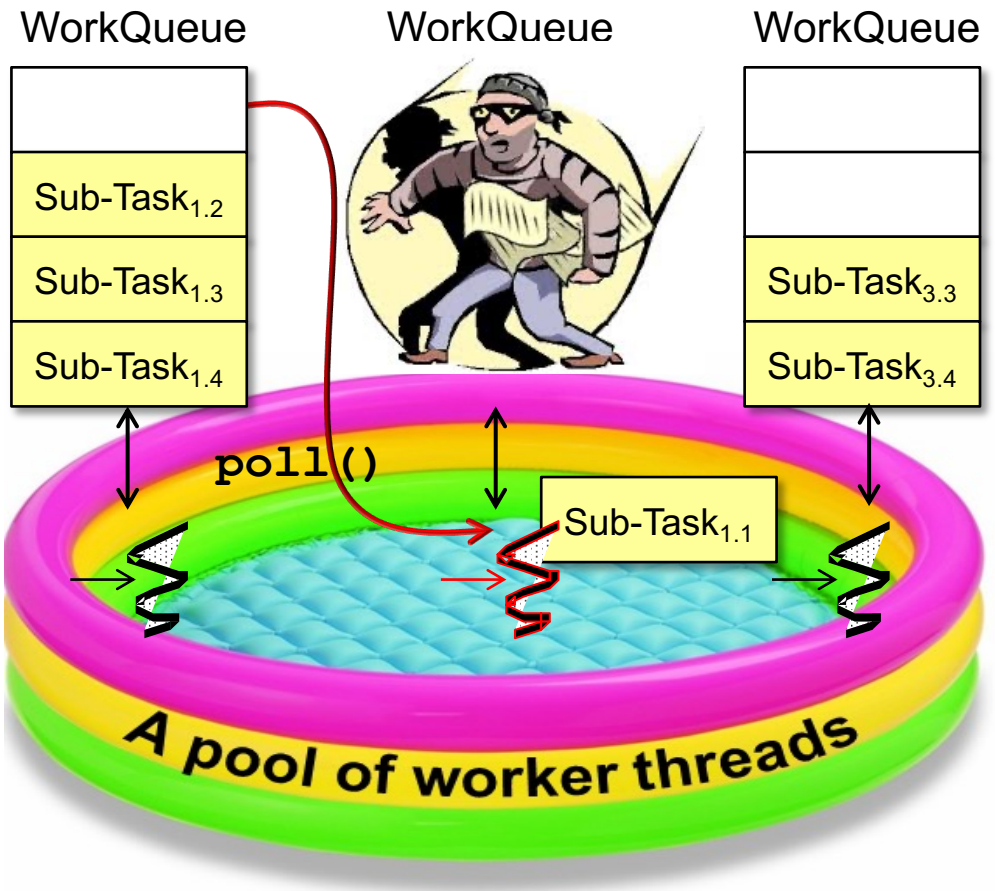
- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques



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

Work Stealing in a Java Fork-Join Pool

- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques
  - Worker threads only steal from other threads in *their* pool
    - i.e., there's no "cross-pool" stealing



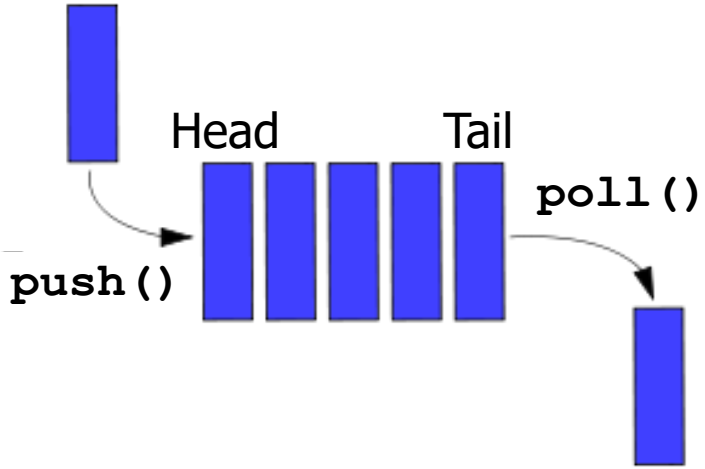
Work Stealing in a Java Fork-Join Pool

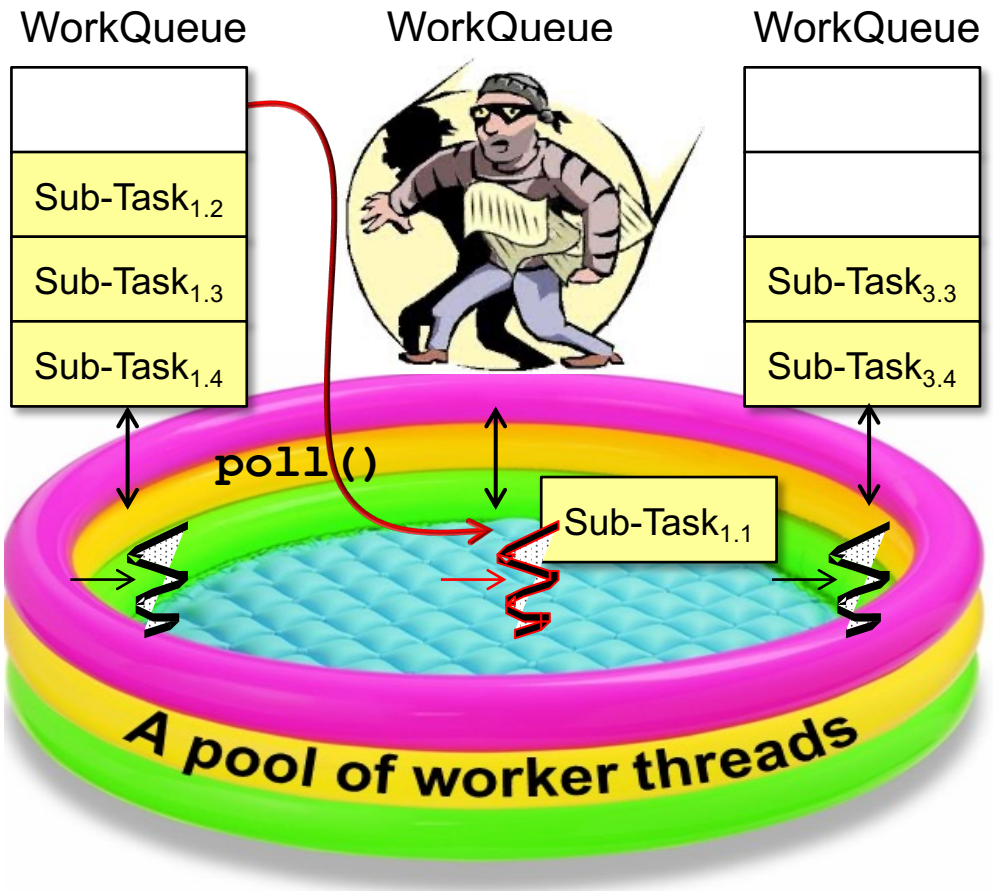
- To maximize core utilization, idle worker threads "steal" work from the tail of busy threads' deques
  - Worker threads only steal from other threads in *their* pool
  - This limitation motivates the use of the common fork-join pool

See upcoming lessons on *"The Common Fork-Join Pool"*

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order

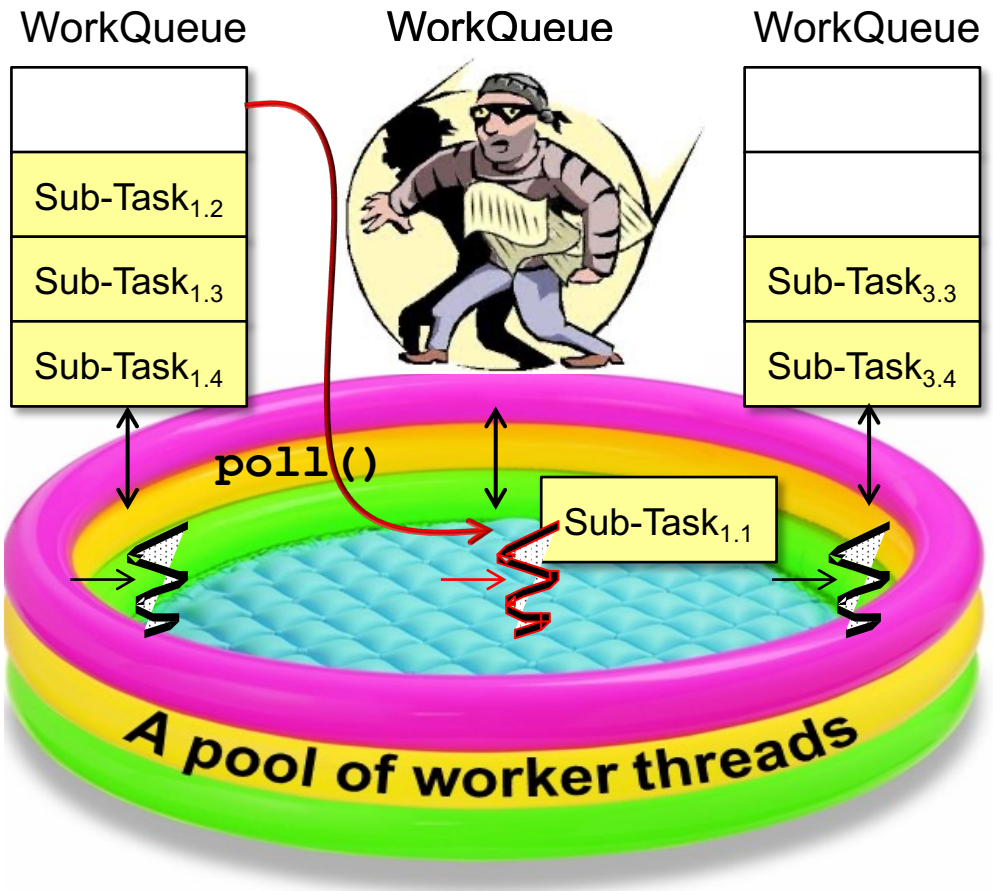




See en.wikipedia.org/wiki/FIFO_(computing_and_electronics)

Work Stealing in a Java Fork-Join Pool

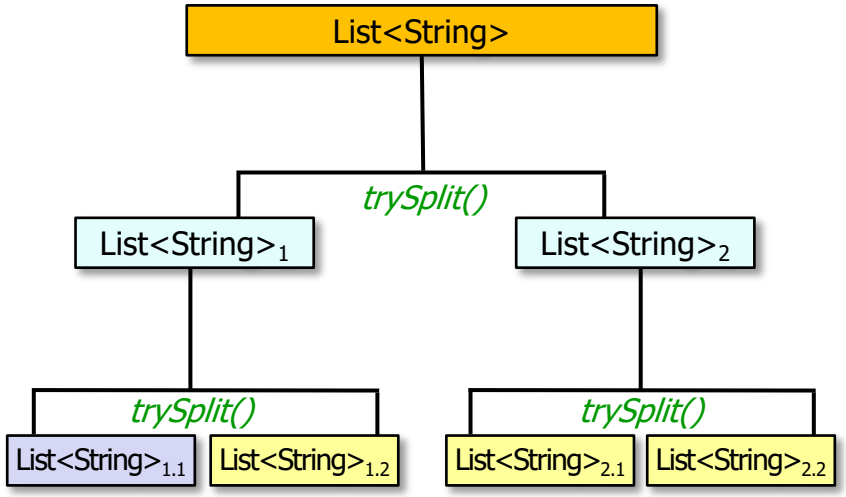
- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque

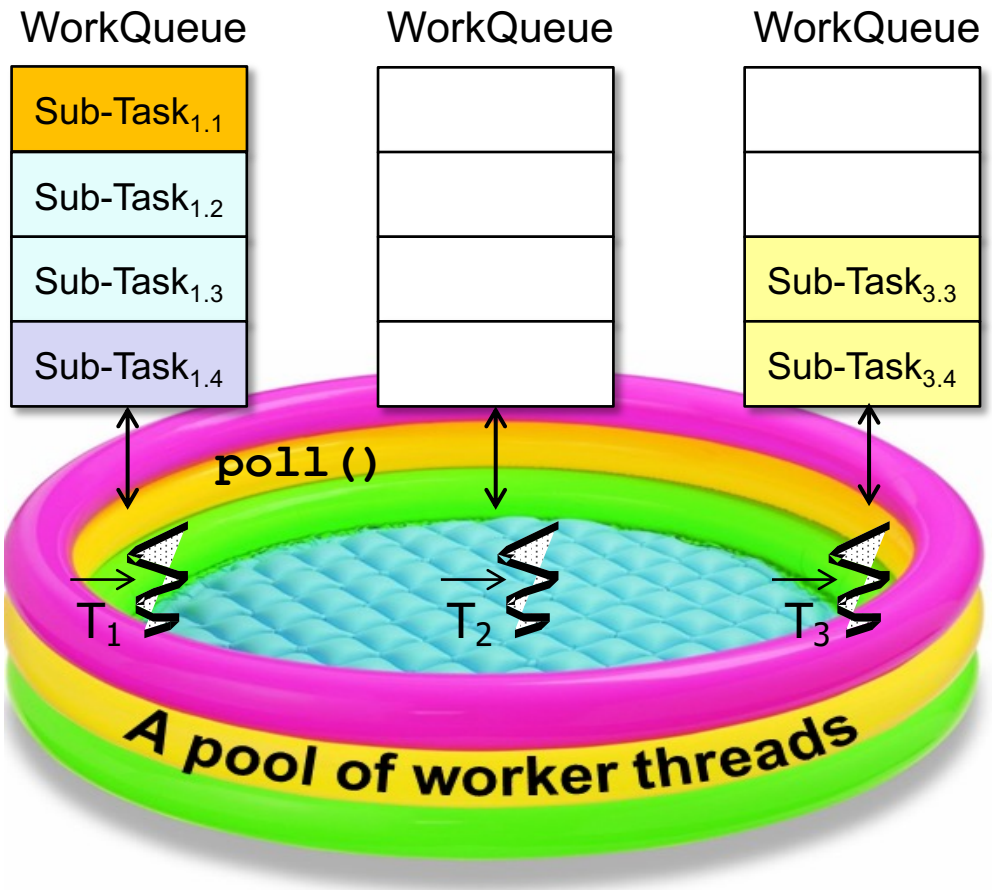


See www.ibm.com/support/knowledgecenter/en/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/resolving.html

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work



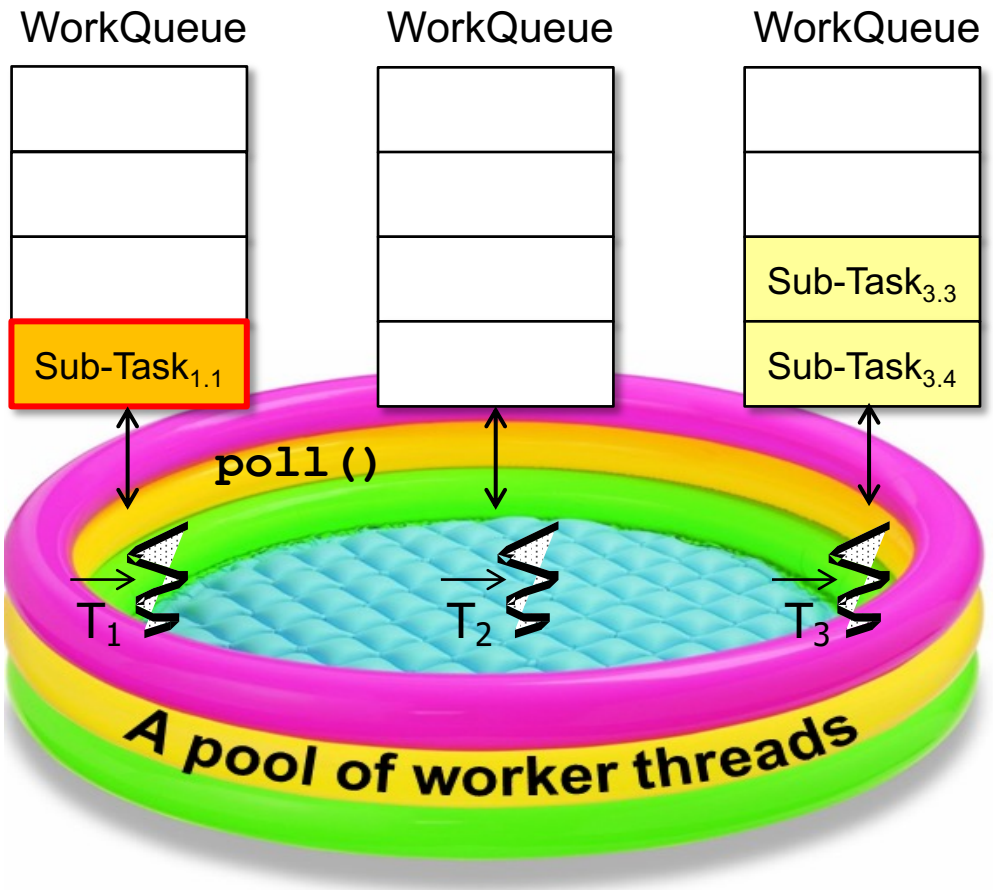


This behavior arises from "divide & conquer" nature of fork-join tasks that split evenly

Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work

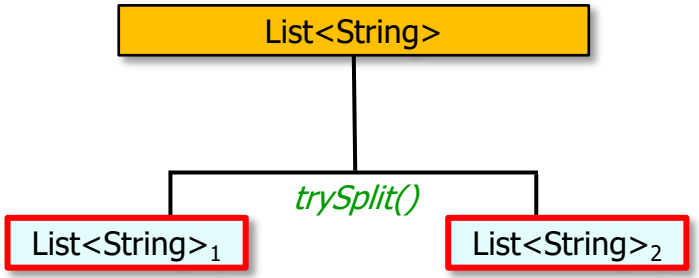
List<String>

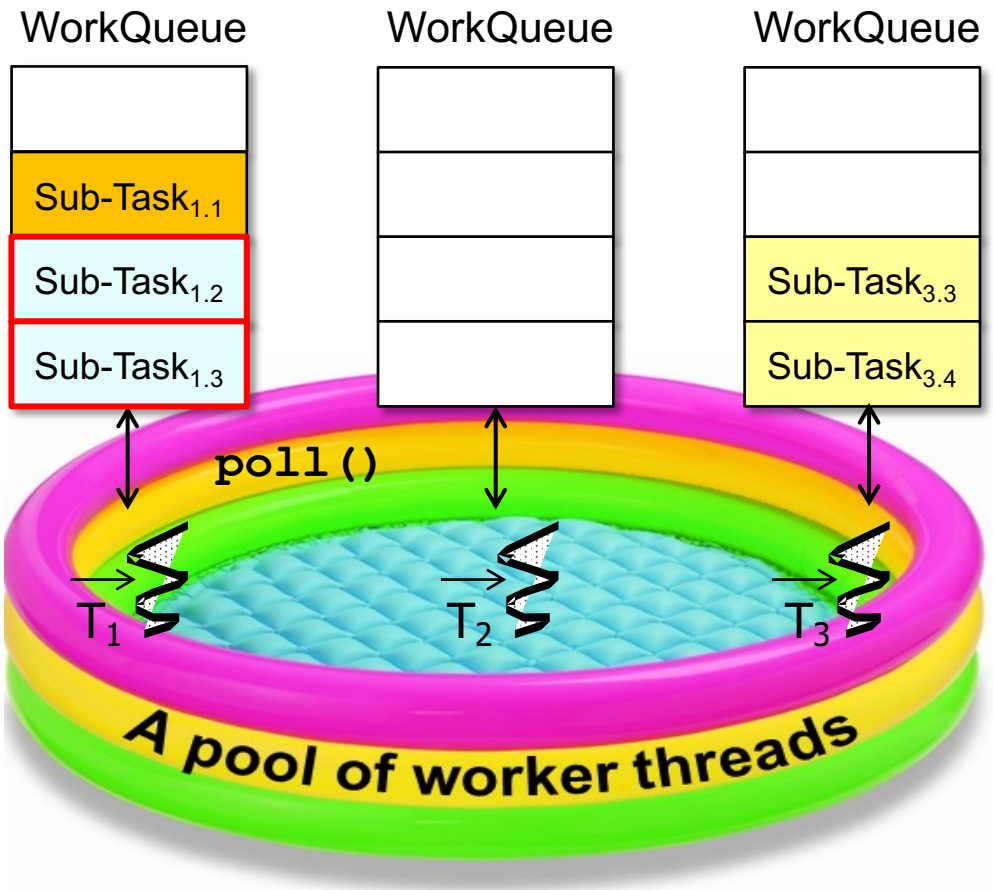


Larger chunks are pushed onto the deque before smaller chunks

Work Stealing in a Java Fork-Join Pool

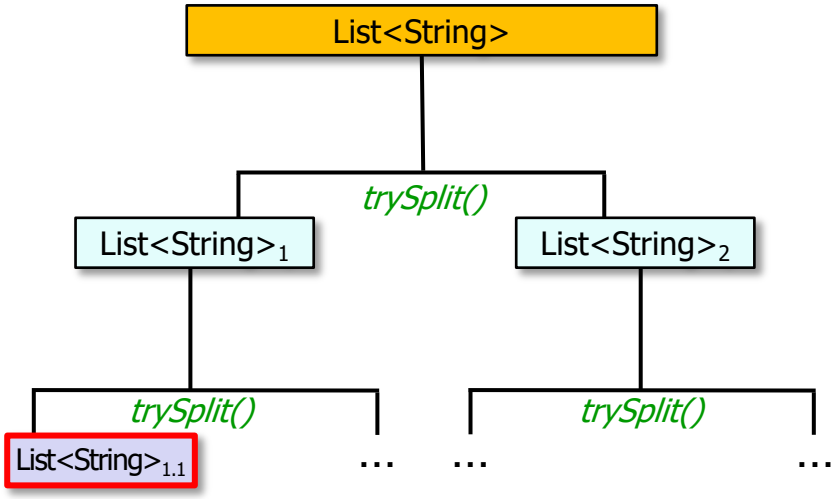
- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work

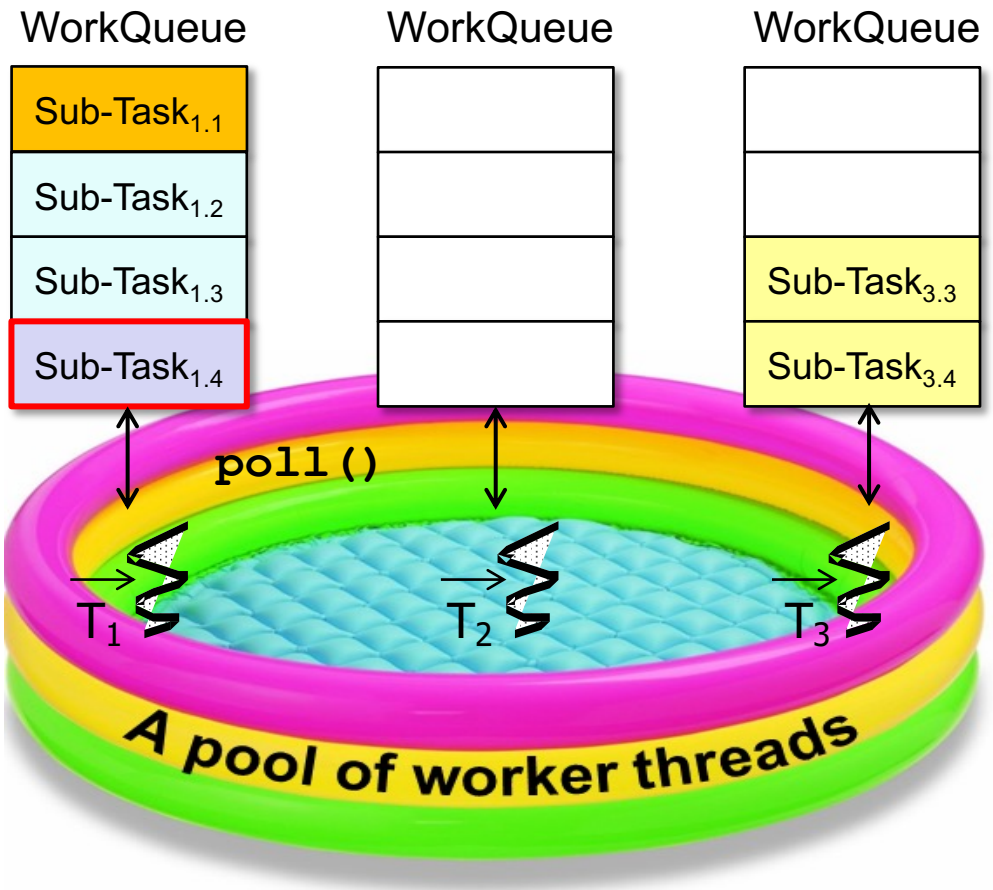




Work Stealing in a Java Fork-Join Pool

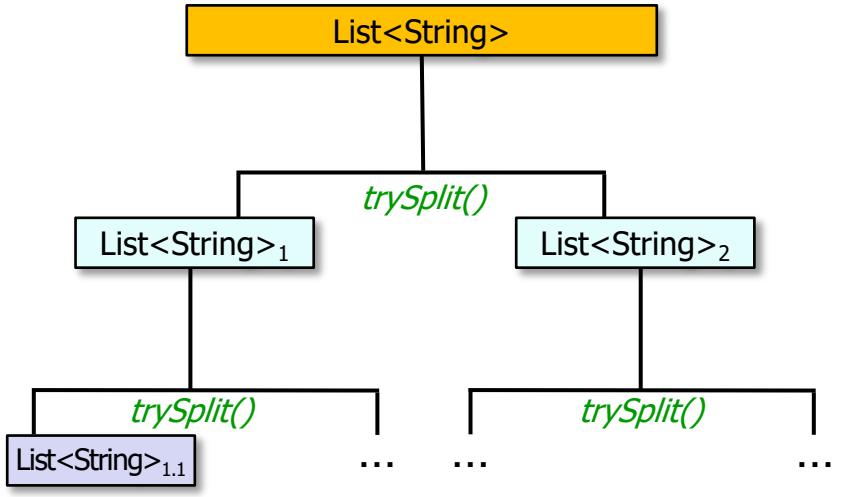
- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work

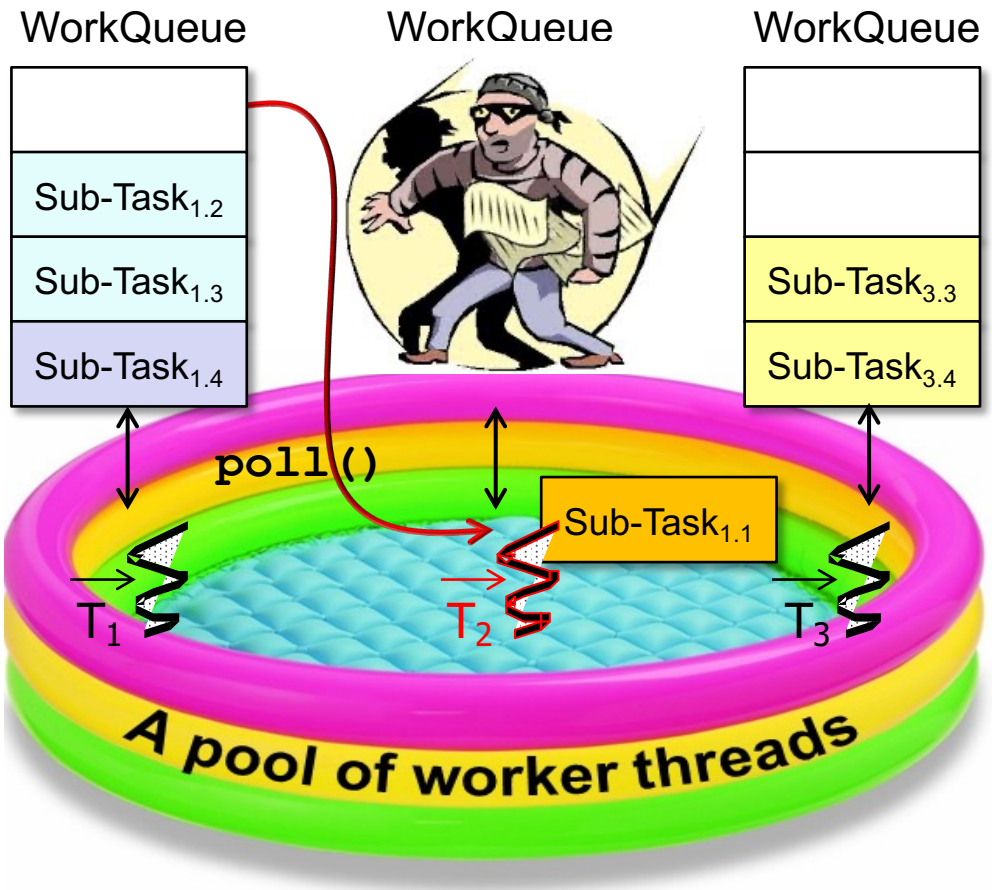




Work Stealing in a Java Fork-Join Pool

- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work

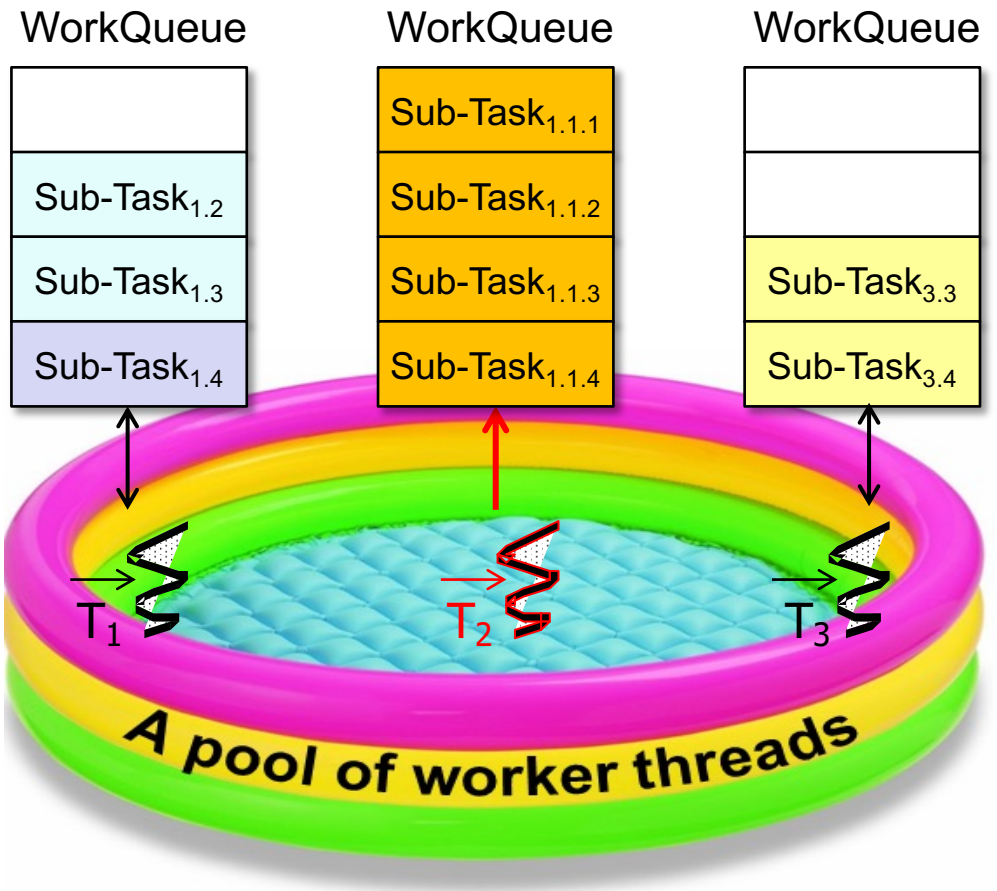




Thread T_2 steals a larger (sub-)task from the end of the deque

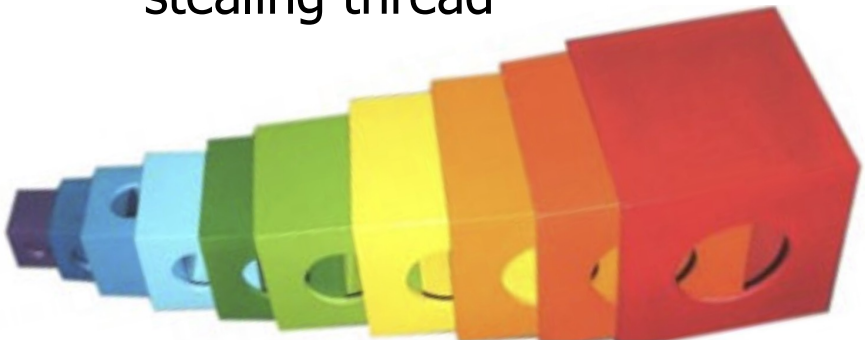
Work Stealing in a Java Fork-Join Pool

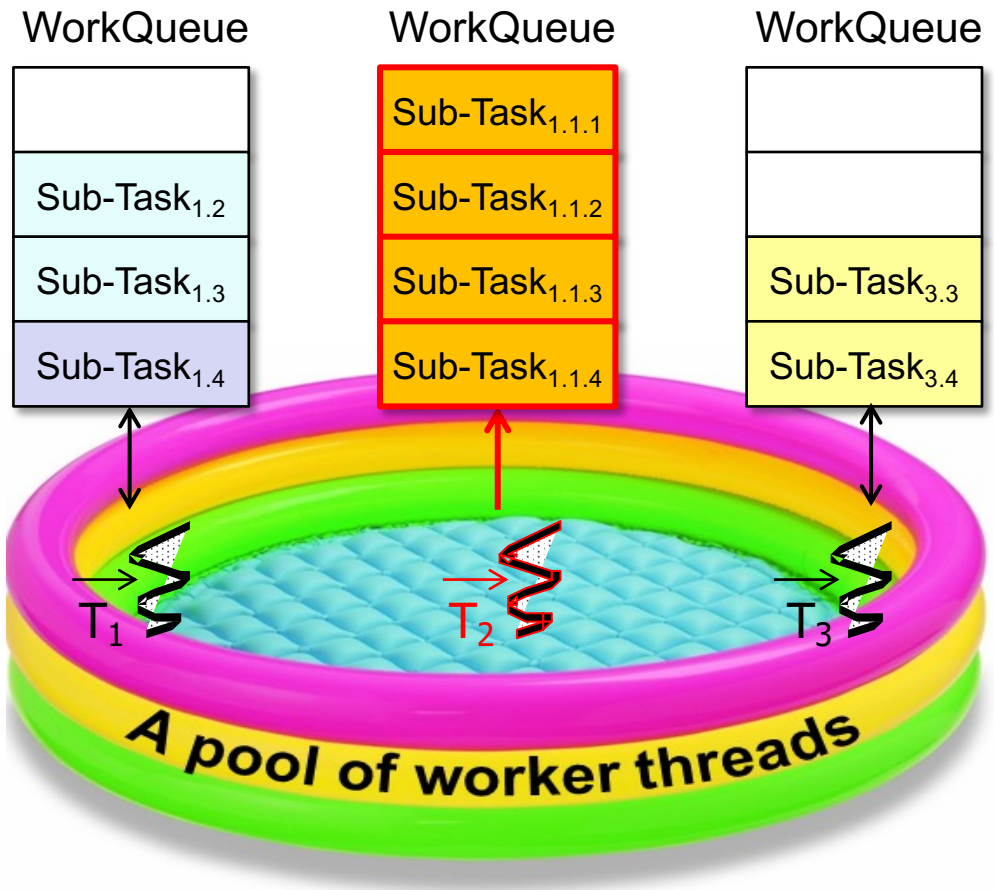
- Tasks are stolen in FIFO order
  - Minimizes contention w/worker thread owning the deque
  - An older stolen task may provide a larger unit of work
    - Enables further recursive decompositions by the stealing thread



Work Stealing in a Java Fork-Join Pool

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

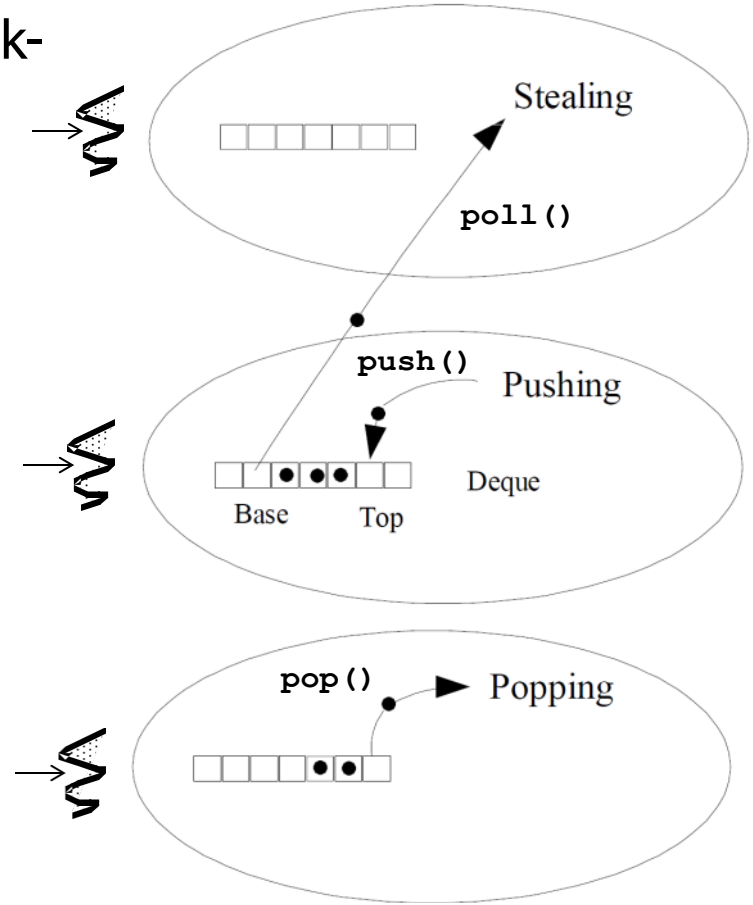




Again, larger chunks are pushed onto the deque before smaller chunks

Work Stealing in a Java Fork-Join Pool

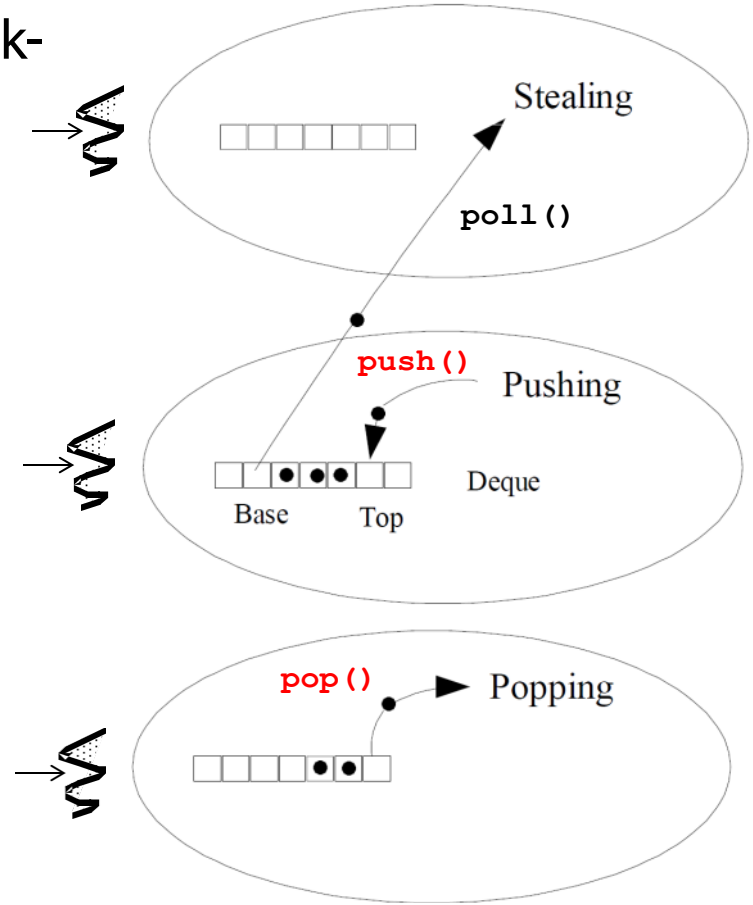
- The WorkQueue deque that implements work-stealing minimizes locking contention



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

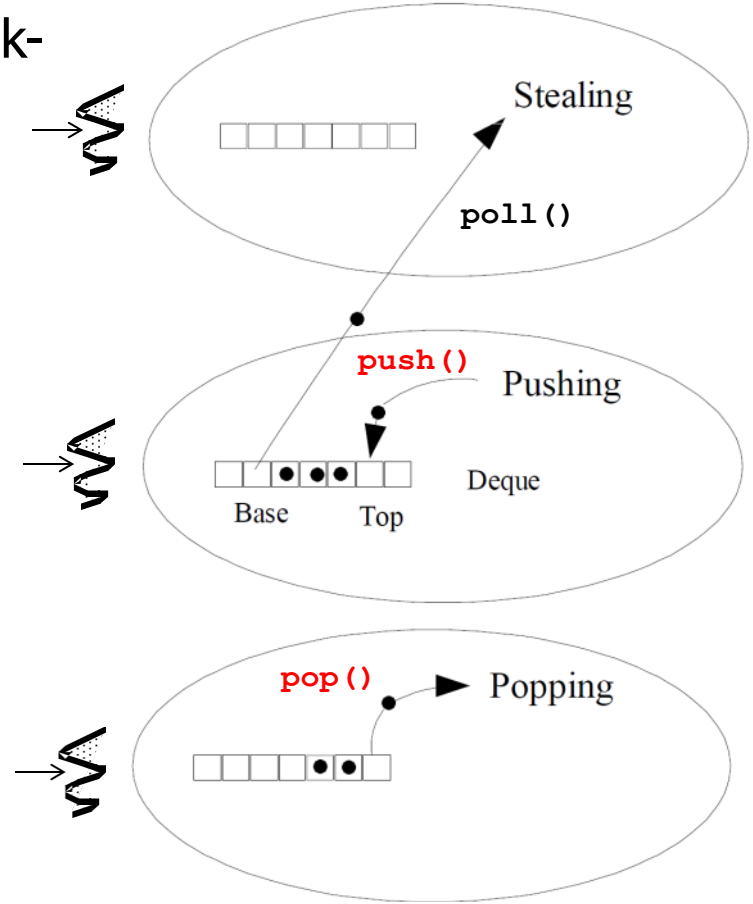
Work Stealing in a Java Fork-Join Pool

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



Work Stealing in a Java Fork-Join Pool

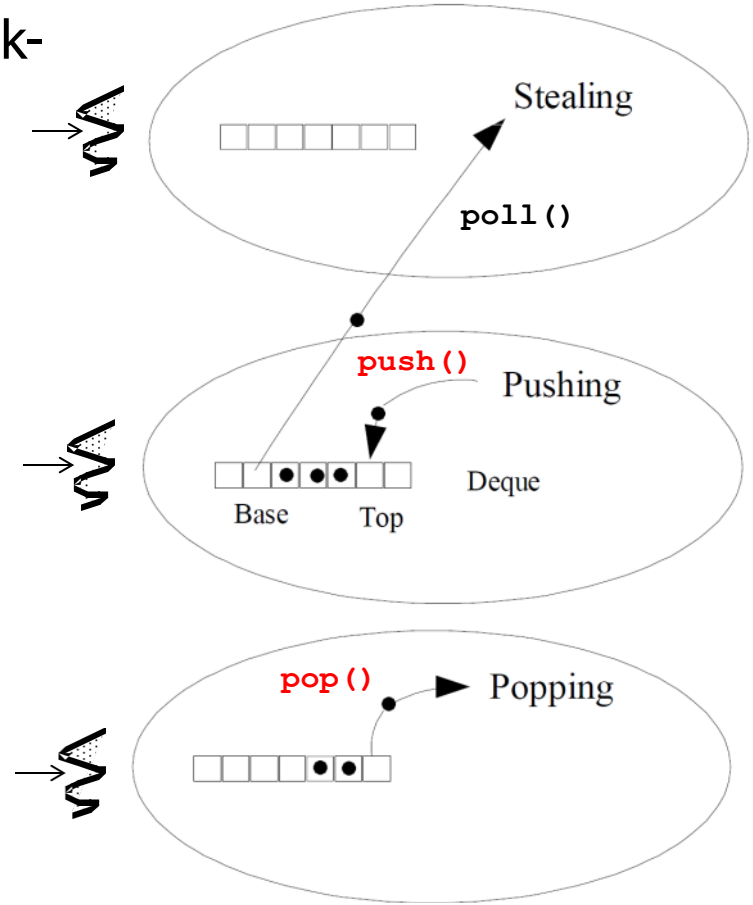
- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & pop() are only called by the owning worker thread
    - These methods use wait-free "compare-and-swap" (CAS) operations



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

Work Stealing in a Java Fork-Join Pool

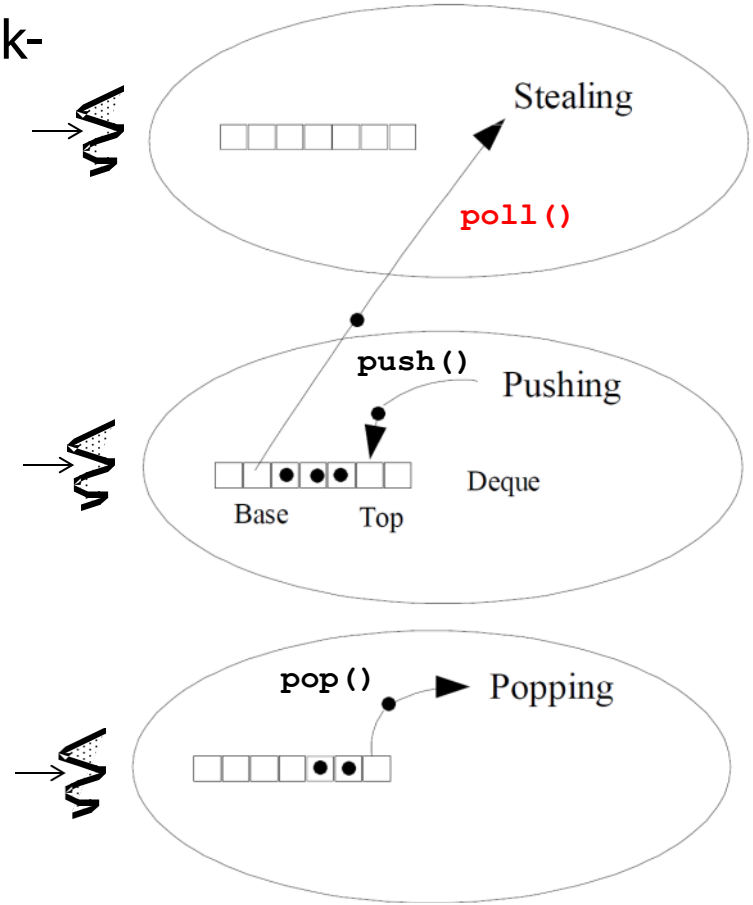
- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & pop() are only called by the owning worker thread
    - These methods use wait-free "compare-and-swap" (CAS) operations
      - An operation is "wait-free" if every thread completes its operation in a bounded # of steps, irrespective of the # of contending threads



See www.justsoftwaresolutions.co.uk/threading/non_blocking_lock_free_and_wait_free.html

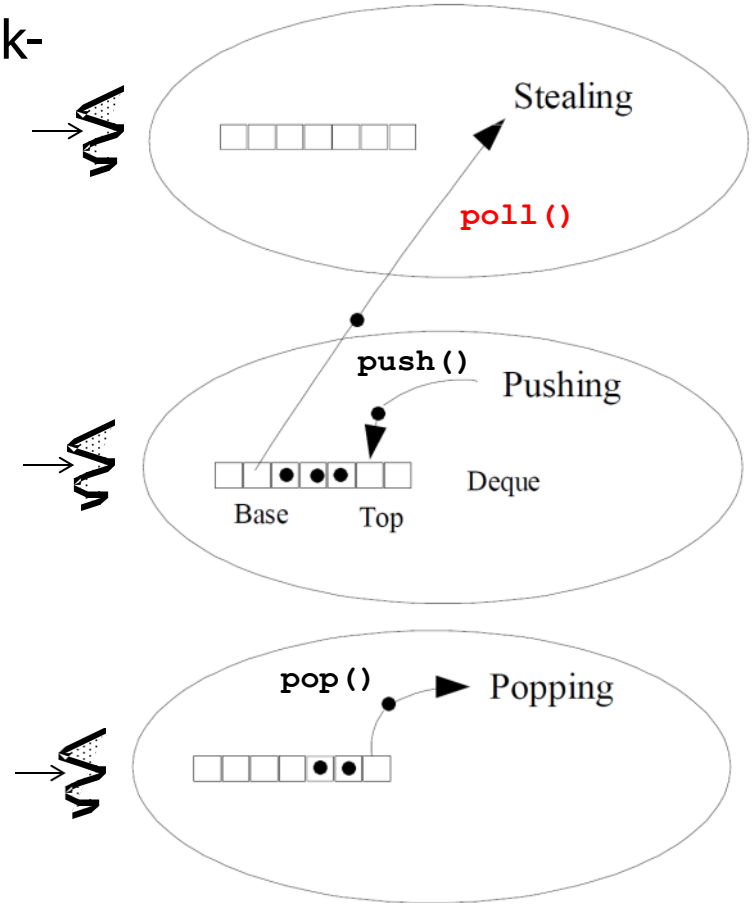
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & 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

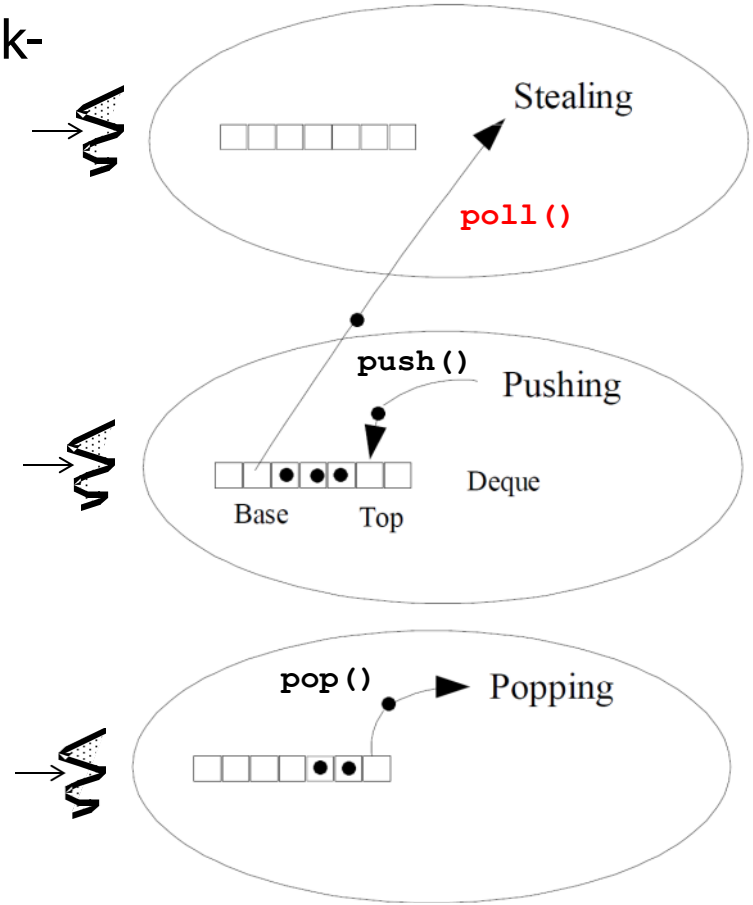
- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & pop() are only called by the owning worker thread
  - poll() may be called from another worker thread to "steal" a (sub-)task
    - May not always be wait-free



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

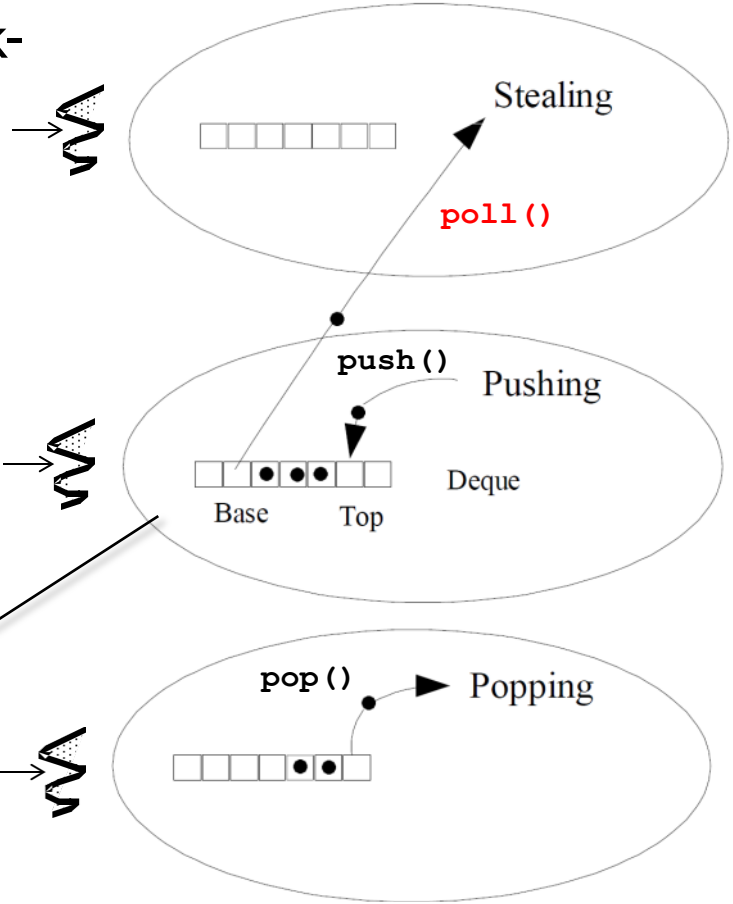
Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & 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
      - i.e., a thread may need to wait an unbounded amount of time to complete due to contention



Work Stealing in a Java Fork-Join Pool

- The WorkQueue deque that implements work-stealing minimizes locking contention
  - push() & 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
      - i.e., a thread may need to wait an unbounded amount of time to complete due to contention



See "Implementation Overview" comments in the ForkJoinPool source code for details..

See java8/util/concurrent/ForkJoinPool.java

End of Java Fork-Join Framework Internals: Work Stealing