The Java Fork-Join Pool: Implementing applyAlllter()

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

- Apply the fork-join framework in practice
- Examine the applyAllIter() method
  - This method uses "work-stealing" to disperse tasks to worker threads

```
<T> List<T> applyAllIter
   (List<T> list,
    Function<T, T> op,
    ForkJoinPool fjPool) {
   return fjPool
      .invoke(new
         RecursiveTask
           <List<T>>() {
         protected List<T>
           compute() {
                   ...
           }
         });
}
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex22

Implementing the applyAllIter() Method

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) {
  return fjPool
    .invoke(new RecursiveTask<List<T>>() {
              protected List<T> compute() {
                ...
              }
            });
}
```

See LiveLessons/blob/master/Java8/ex22/src/utils/ForkJoinUtils.java

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) {
  return fjPool
    .invoke(new RecursiveTask<List<T>>() {
              protected List<T> compute() {
                ...
              }
            });
}
```

These parameters are treated as "effectively final" variables in the anonymous inner class below

See www.linkedin.com/pulse/java-8-effective-final-gaurhari-dass

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) {
  return fjPool
    .invoke(new RecursiveTask<List<T>>() {
              protected List<T> compute() {
                ...
              }
            });
}
```

Create an anonymous RecursiveTask instance whose compute() method iterative creates many sub-tasks

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) {
  return fjPool
    .invoke(new RecursiveTask<List<T>>() {
              protected List<T> compute() {
                ...
              }
            });
}
```

Code is verbose due to lack of functional interface (& thus can't use lambdas)..

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) {
  return fjPool
    .invoke(new RecursiveTask<List<T>>() {
              protected List<T> compute() {
                ...
              }
            });
}
```

Invoke the task on the fork-join pool & then wait for & return the results

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

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

Hook method implements the main fork-join task

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

Lists that hold the forked tasks & the results

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

Iterate through input list, fork all the tasks, & add them to the forks list

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

This implementation relies on "work-stealing" to disperse tasks to worker threads

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

Join all results of forked tasks & add to results list

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

Return the results list

Implementing the applyAllIter() Method

- Apply an 'op' to all items in the list by calling fork-join methods iteratively

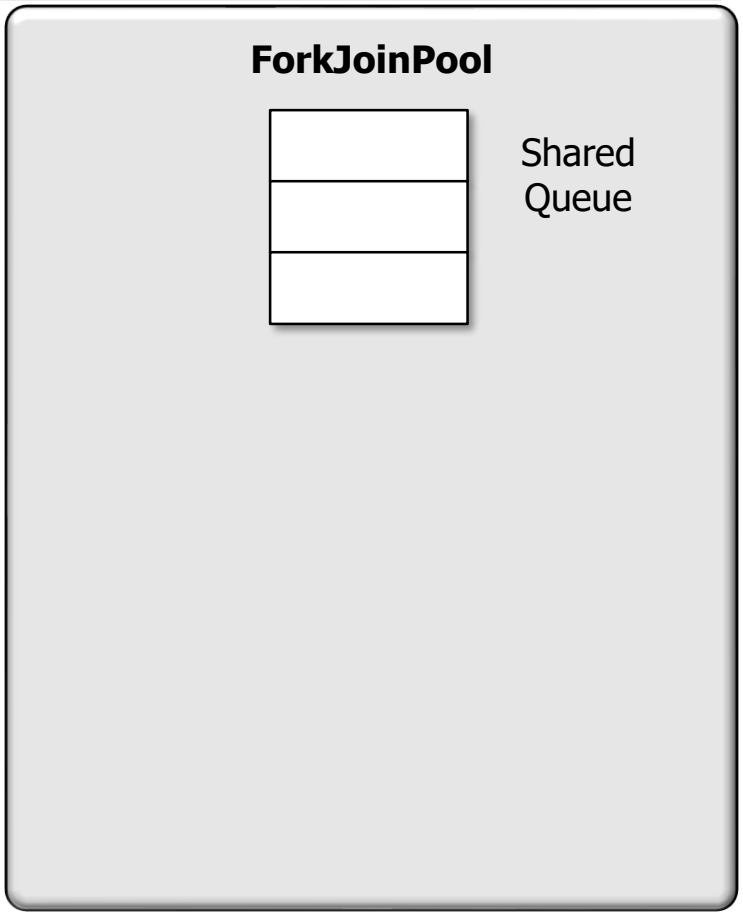
```
<T> List<T> applyAllIter(List<T> list, Function<T, T> op,
                         ForkJoinPool fjPool) { ...
  protected List<T> compute() {
    List<ForkJoinTask<T>> forks = new LinkedList<>();
    List<T> res = new LinkedList<>();

    for (T t : list)
      forks.add(new RecursiveTask<T>() {
                  protected T compute() { return op.apply(t); }
      }.fork());

    for (ForkJoinTask<T> task : forks) res.add(task.join());
    return res;
  } ...
```

This implementation is very simple to program & understand since it's iterative

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  fjPool.invoke
    (new RecursiveTask<List<T>>() {
       protected List<T> compute() {
         ...
       }
  }
  ...
```

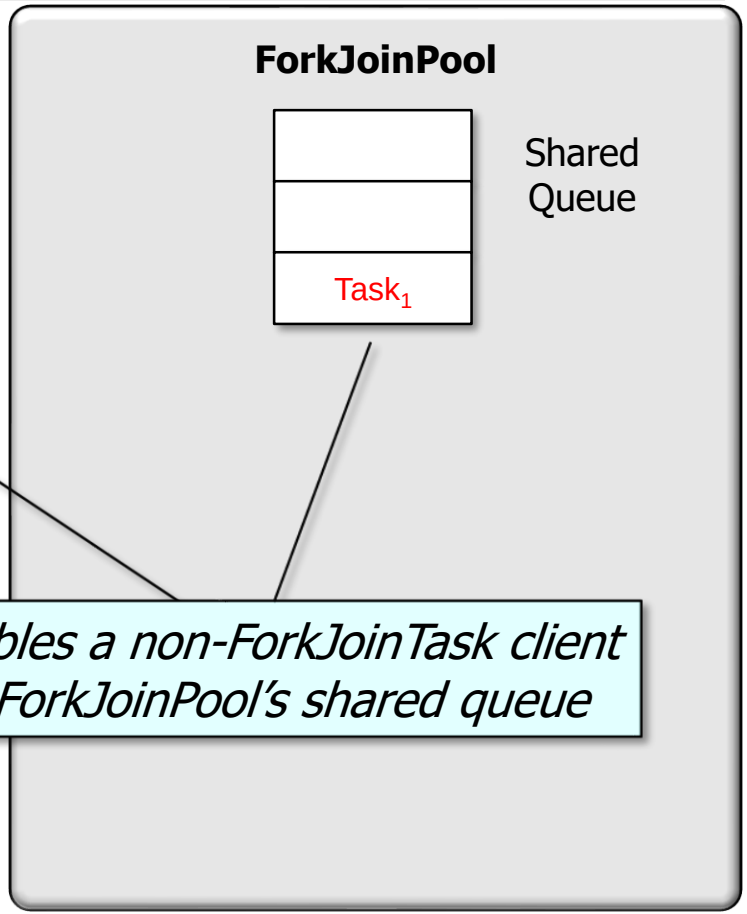


See LiveLessons/blob/master/Java8/ex22/src/utils/ForkJoinUtils.java

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  fjPool.invoke
    (new RecursiveTask<List<T>>() {
       protected List<T> compute() {
         ...
       }
  }
  ...
```



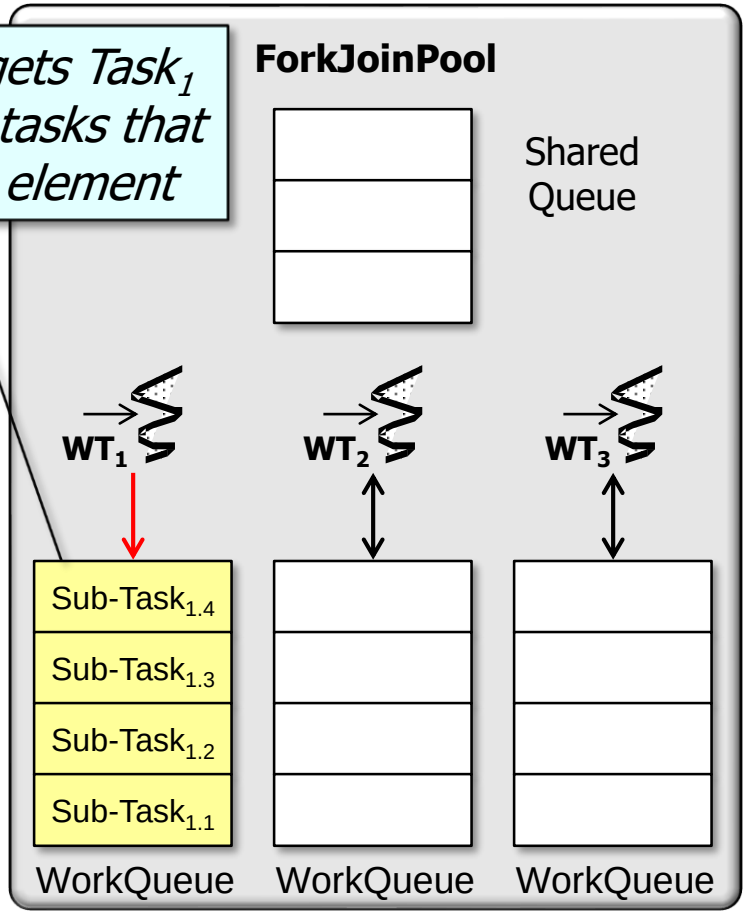
The invoke() method enables a non-ForkJoinTask client to insert a task into the ForkJoinPool's shared queue

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllI
          (List<T> lis
           Function<T, T> op,
           ForkJoinPool fjPool) {
  ...
  for (T t : list)
    forks.add(new RecursiveTask<T>() {
      protected T compute()
      { return op.apply(t); }
    }.fork());

  for (ForkJoinTask<T> task : forks)
    results.add(task.join());
  ...
```

Worker thread WT_1 gets $Task_1$ & creates n new sub-tasks that run 'op' on each list element

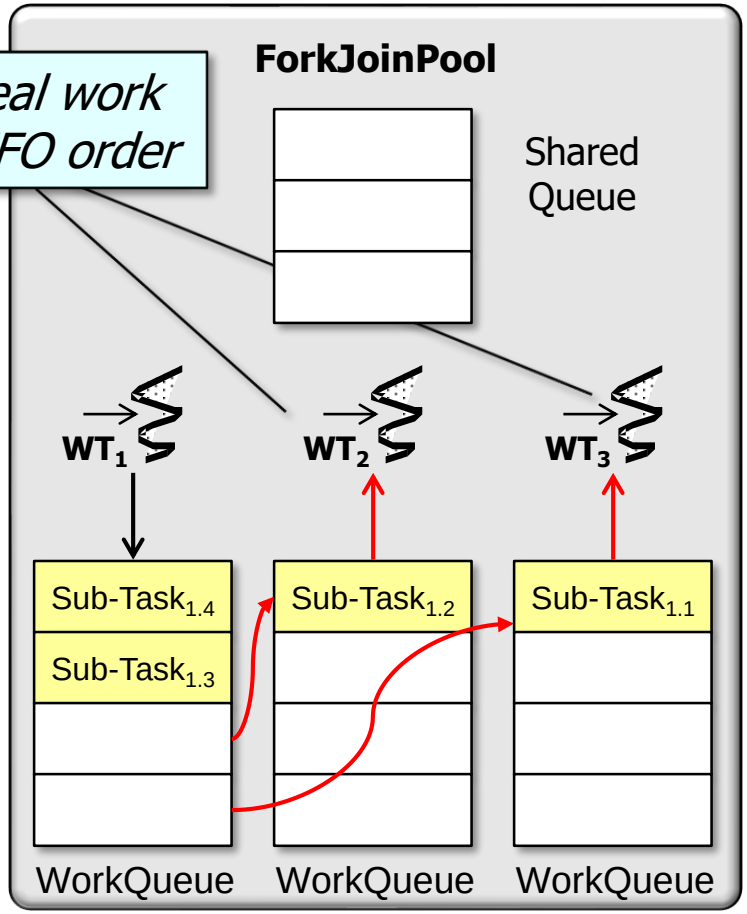


The highlighted code runs in the RecursiveTask's compute() method

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  ...
  for (T t : list)
    forks.add(new RecursiveTask<T>() {
      protected T compute()
      { return op.apply(t); }
    }.fork());

  for (ForkJoinTask<T> task : forks)
    results.add(task.join());
  ...
```



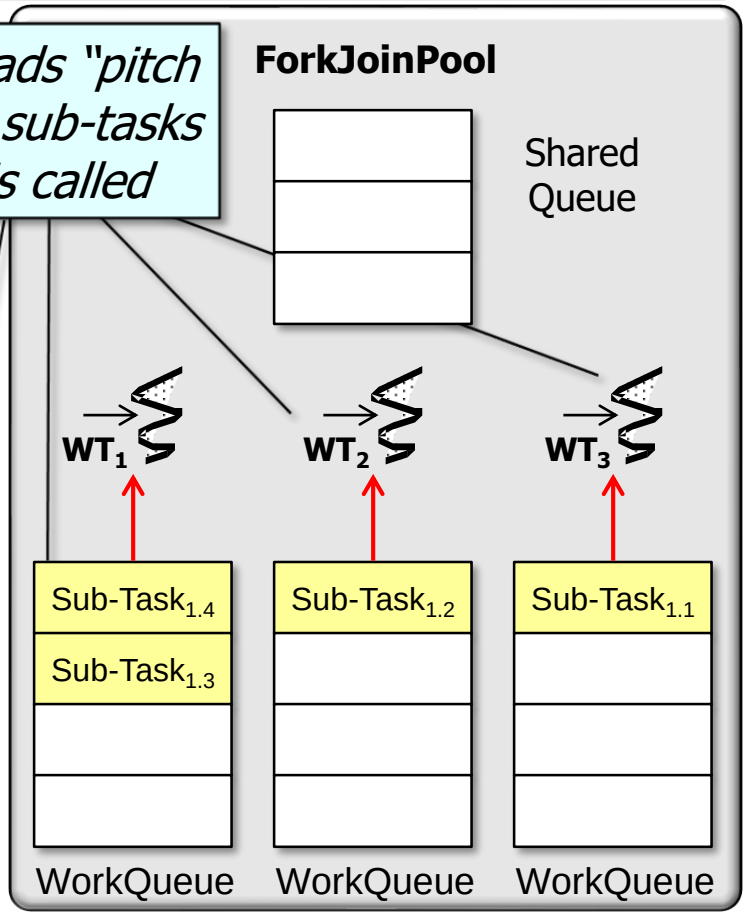
"Work-stealing" overhead is high, but copying & method call overhead is low

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  ...
  for (T t : list)
    forks.add(new RecursiveTask<T>() {
      protected T compute()
      { return op.apply(t); }
    }.fork());

  for (ForkJoinTask<T> task : forks)
    results.add(task.join());
  ...
```

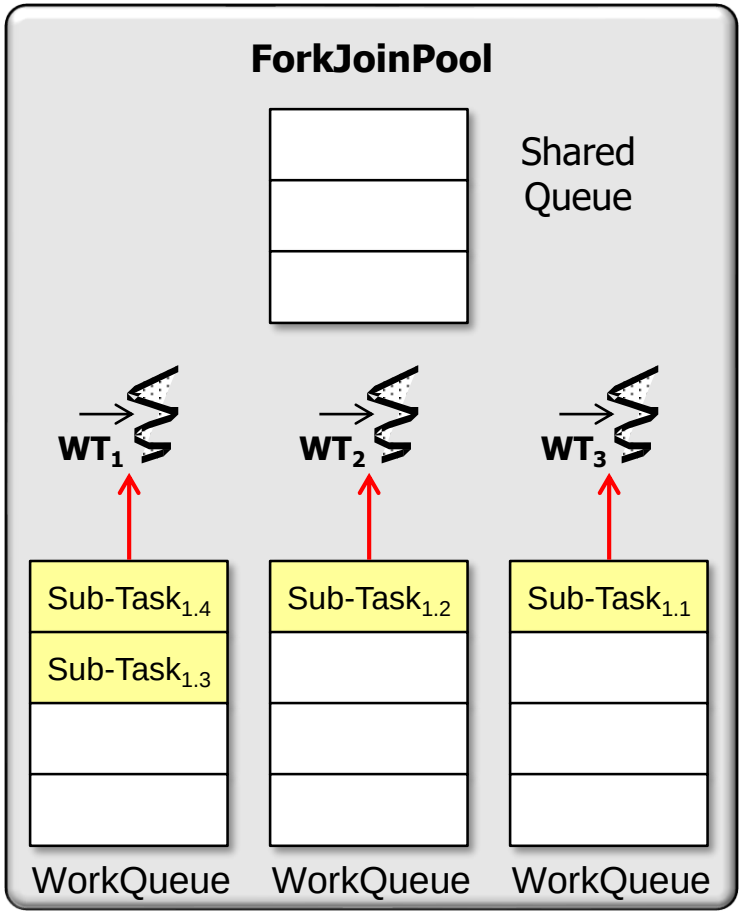
All worker threads "pitch in" to compute sub-tasks when join() is called



Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  ...
  for (T t : list)
    forks.add(new RecursiveTask<T>() {
      protected T compute()
      { return op.apply(t); }
    }.fork());

  for (ForkJoinTask<T> task : forks)
    results.add(task.join());
  ...
```



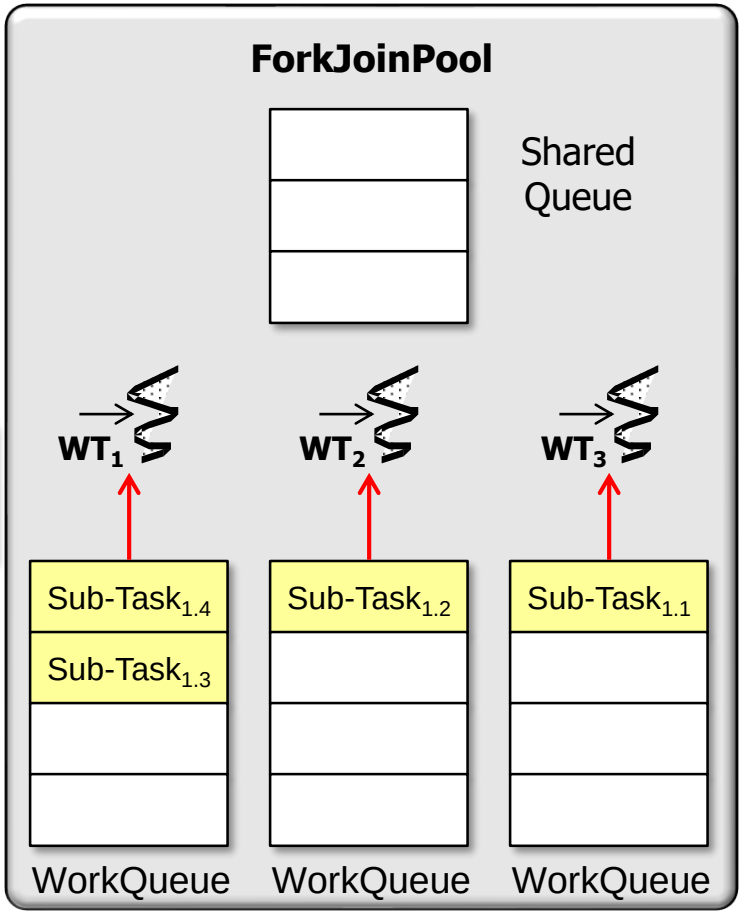
"Collaborative Jiffy Lube" model of processing!

Implementing the applyAllIter() Method

- Visualizing applyAllIter()

```
<T> List<T> applyAllIter
          (List<T> list,
           Function<T, T> op,
           ForkJoinPool fjPool) {
  ...
  for (T t : list)
    forks.add(new
      protected T
      { return op.apply(t); }
    }.fork());

  for (ForkJoinTask<T> task : forks)
    results.add(task.join());
  ...
```

This loop implements "barrier synchronization"



See en.wikipedia.org/wiki/Barrier_(computer_science)

End of the Java Fork-Join Pool: Implementing applyAllIter()