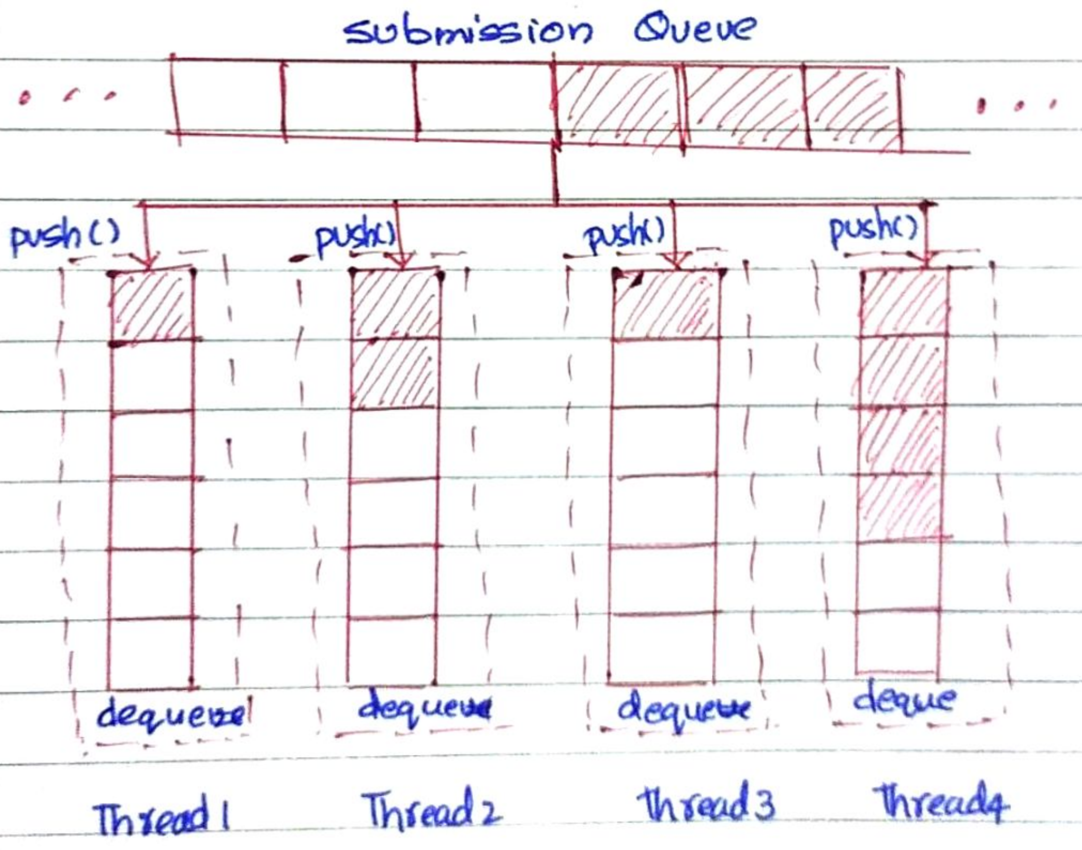
Fork and Join

Fork-Join breaks the task at hand into mini-tasks until the mini-task is simple enough that it can be solved without further breakups. It's like a divide and conquer algorithm.

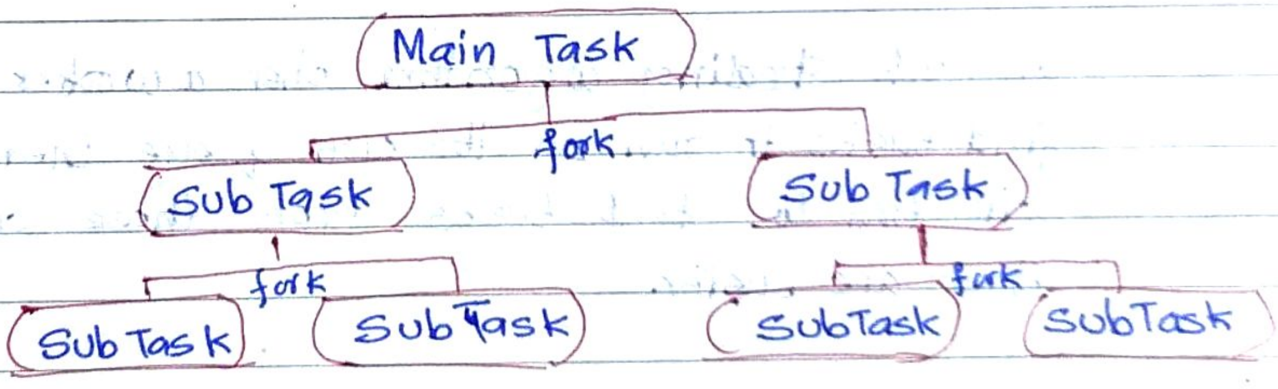
One important concept to note in this framework is that ideally no worker thread is idle. They implement a work stealing algorithm in that idle worker steal the work from those workers who are busy.



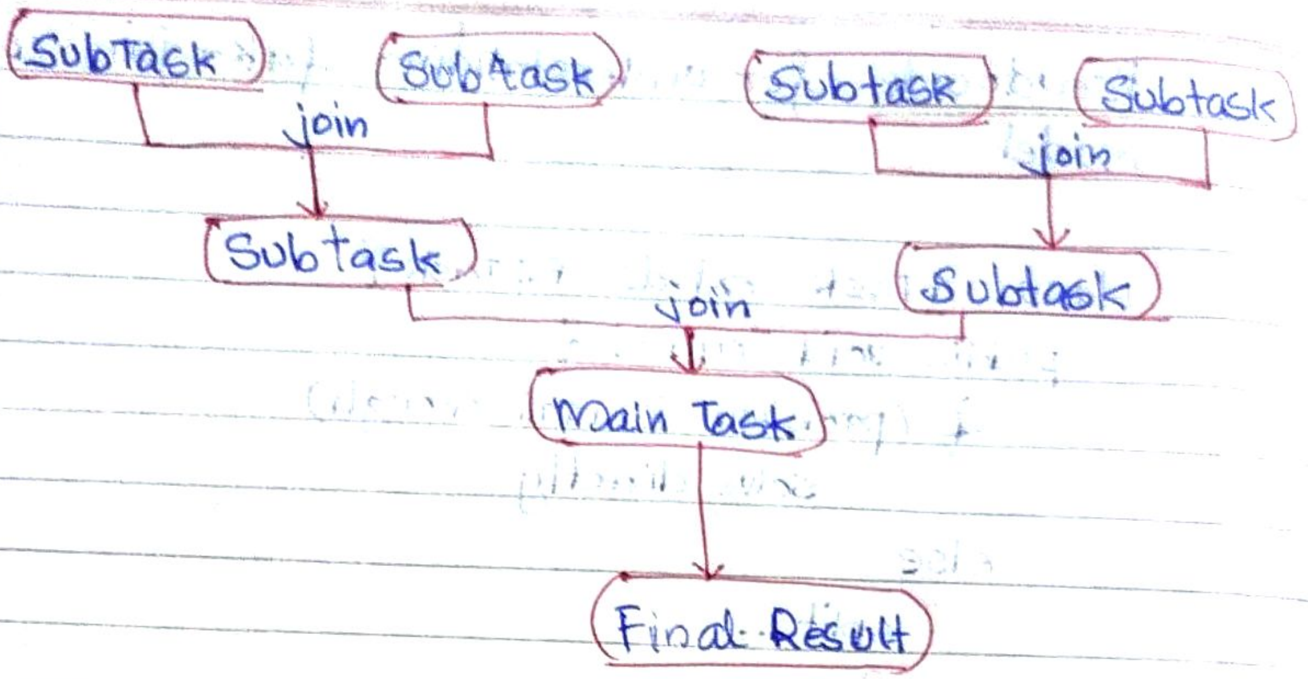
Each worker thread has its own worker queue, a double-ended queue. The local queues are referred as deque.

Pseudocode for understanding fork-join computatio
model

```
class ATask extends FJTask {
    public void run () {
        if (problem is mall enough)
                solve directly
        else
            split...
            fork ...
            join...
            Compose
    }
}
```

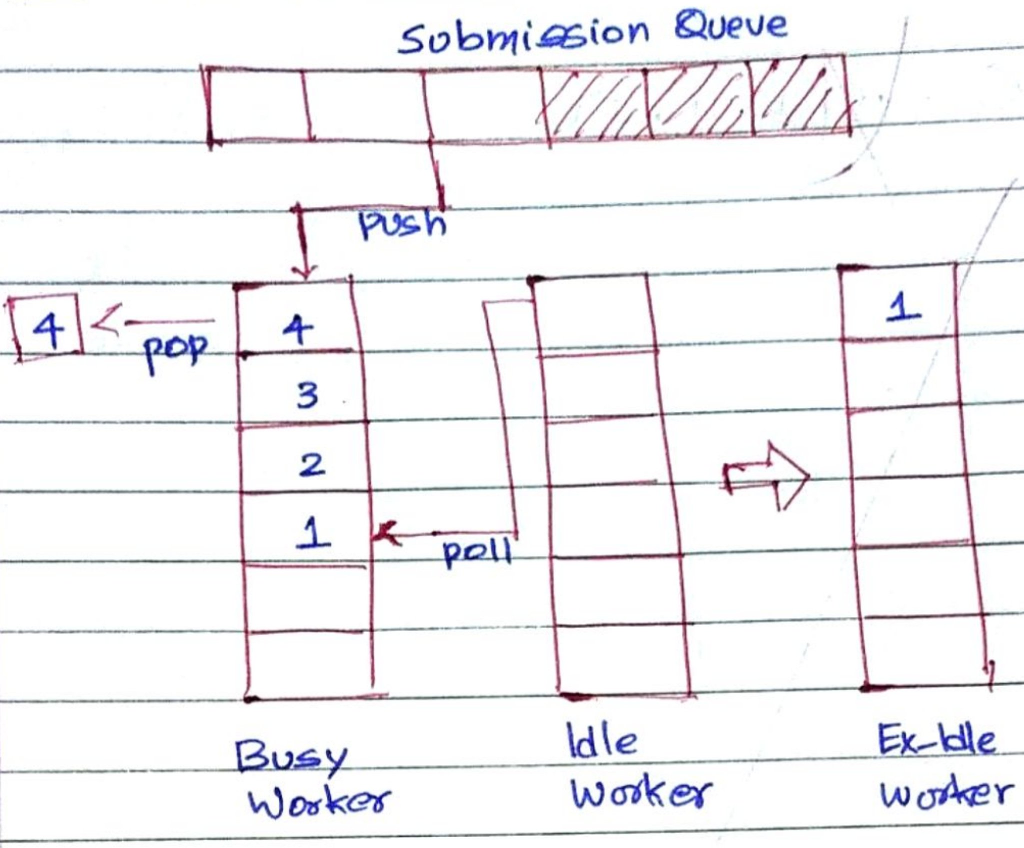


The first phase of fork-join model is to split the task using fork.



The second phase is to wait for completion using join of all these subtasks. once they are completed we can combine and return a result back.

In work stealing algorithm when a worker cannot find tasks to run on its own queue then it will try to steal tasks from those workers that are busier.



The push and pop methods only called by the owner of the queue and poll method is only called by the worker trying to steal work from a different worker

Fork-Join Task

It is a java class ForkJoinTask which behaves similarly to java thread.

So the core classes used in fork/join framework are ForkJoinPool and ForkJoinTask

Fork-Join Pool

- specialized implementation of ExecutorService implementing the work stealing algorithm.
- Another important difference compared to the other Executor Service's is that this pool need not be explicitly ~~shut down~~ shut down upon programs exit. because all its threads are in daemon mode.

There are three different ways of submitting task to the ForkJoinPool

① execute() : desired asynchronous execution; call its fork method to split the work between multiple threads.

② invoke(): await to obtain the result; call the invoke method on the pool

③ submit() : returns a Future object that can be used for checking status. and obtaining result

Fork Join Task

- ~~It is an abstract class that provides several methods for checking the execution status of a task.~~

ForkJoinTask

- an abstract class for creating tasks that run within ForkJoinPool.
- RecursiveAction and RecursiveTask are the only two subclasses of ForkJoinTask.
- RecursiveAction does not return a value while RecursiveTask does have a return value and returns an object of the specific type.

Sample Program using Recursive Action

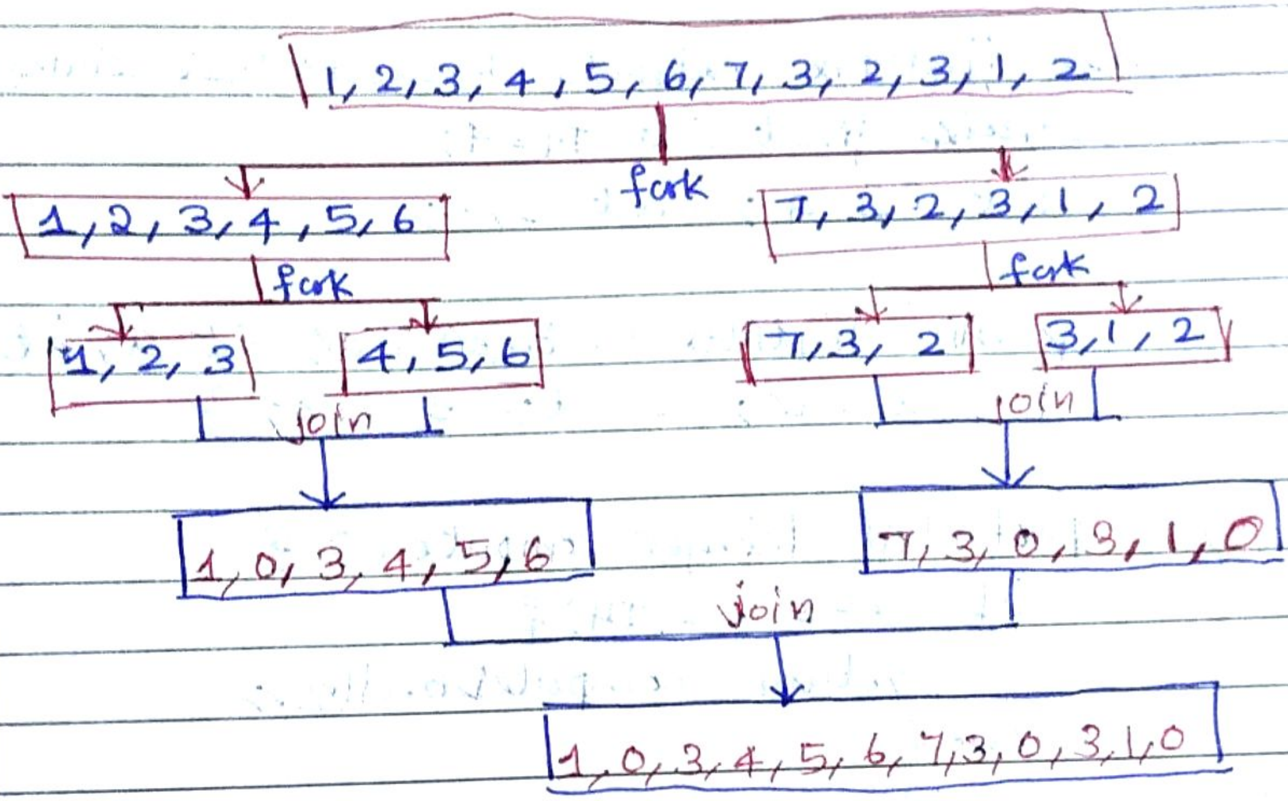
"Replace the value of all the entries of a particular element in an array of integers"

```
① import java.util.concurrent.*;
② public class Demo1 extends RecursiveAction {

      private int ele, th, s, e;
      private int[] ar;

③    protected void compute() {
          if (e-s <= th)
              process (ele, ar, s, e);
          else
      ④       ForkJoinTask.invokeAll (createSubTasks());
      }
private List <Demo1> createSubTasks () {
   List <Demo1> tasklst = new ArrayList<> ();
   tasklst.add (new Demo1(ele, ar, th, s, (s+e)/2));
   tasklst.add (new Demo1 (ele, ar, th, (s+e)/2, e));
   return tasklst;
}

private void process (int ele, int[] a, int s, int e) {
   for (int i = s; i < e ; i++) {
      if (a[i] == ele)
           a[i] = -1;
   }
}
}
```

In the main(), you can write

```
⑤ ForkJoinPool pool = new ForkJoinPool();
   int [] a = new int [] {1,2,3,4,5,6,7,66,3,3};
   ~~pool.invoke (new ForkJoin~~
⑥ pool.invoke (new Demo1(3, a, 2, 0, a.length));
```

Array: 1, 2, 3, 4, 5, 6, 7, 3, 2, 3, 1, 2
Threshhold : 3
Replace number 2 by 0.



Sample program using RecursiveTask

```
① import java.util.concurrent.*;

② class findMax extends RecursiveTask <Integer>{
      static final int TH = 4;
      int [] a; int s, e;

      public findMax (Integer[] ar, int st, int end)
      {   a = ar;  s = st;  e = end; }

③     protected Integer compute () {
         if (e-s <= TH) {
              return computeDirectly();
         else {
            int m = (s+e)/2
            findMax left = new findMax(a, s, m);
            FindMax right = new FindMax (a, m, e);

            invokeAll (left, right);
            return Math.max (left.join(), right.join());
         }
      }

      private Integer computeDirectly() {
         int max = Integer.MIN_VALUE;
         for(int i=s; i<e; i++) {
            if (a[i] > max)   max = a[i]; }
   }  }  return max;
```

```
public class MainDemo {
  public static void main(String[] args) {
    Integer[] data = // read int values

④   ForkJoinPool pool = new ForkJoinPool();
⑤   FindMax task = new FindMax(data, 0, data.length);

⑥   Integer res = pool.invoke(task);

    System.out.println(res);
  }
}
```