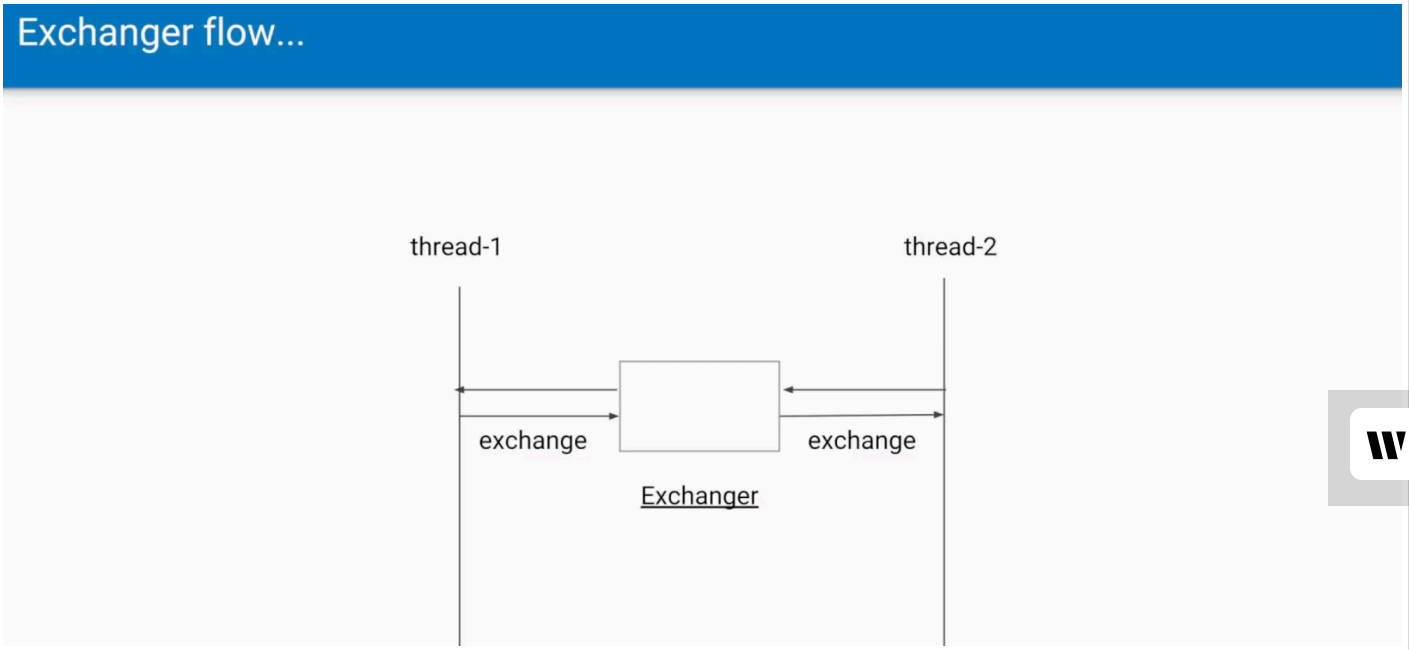
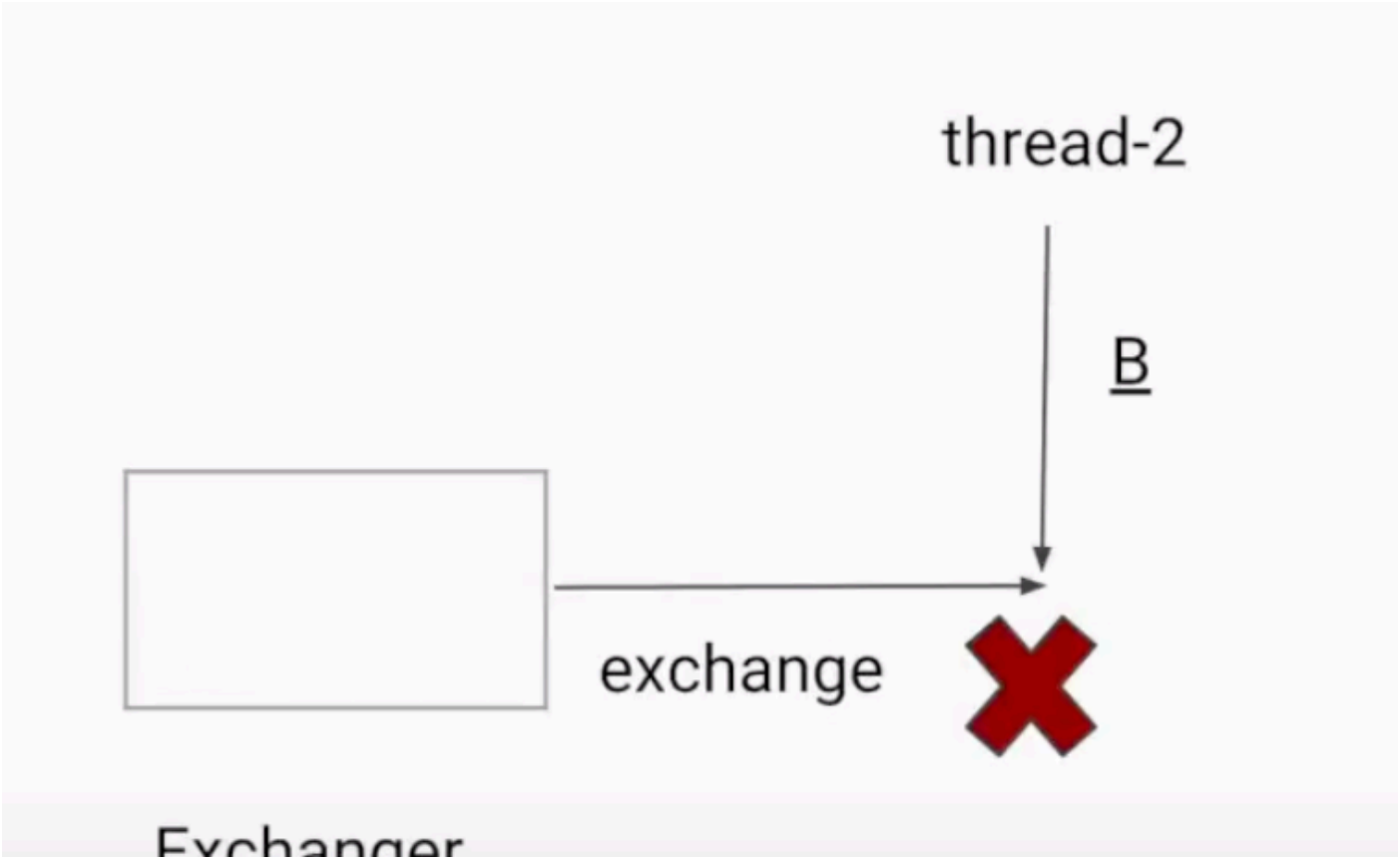
Multithreading 7 : Exchanger Class and Fork Join pool

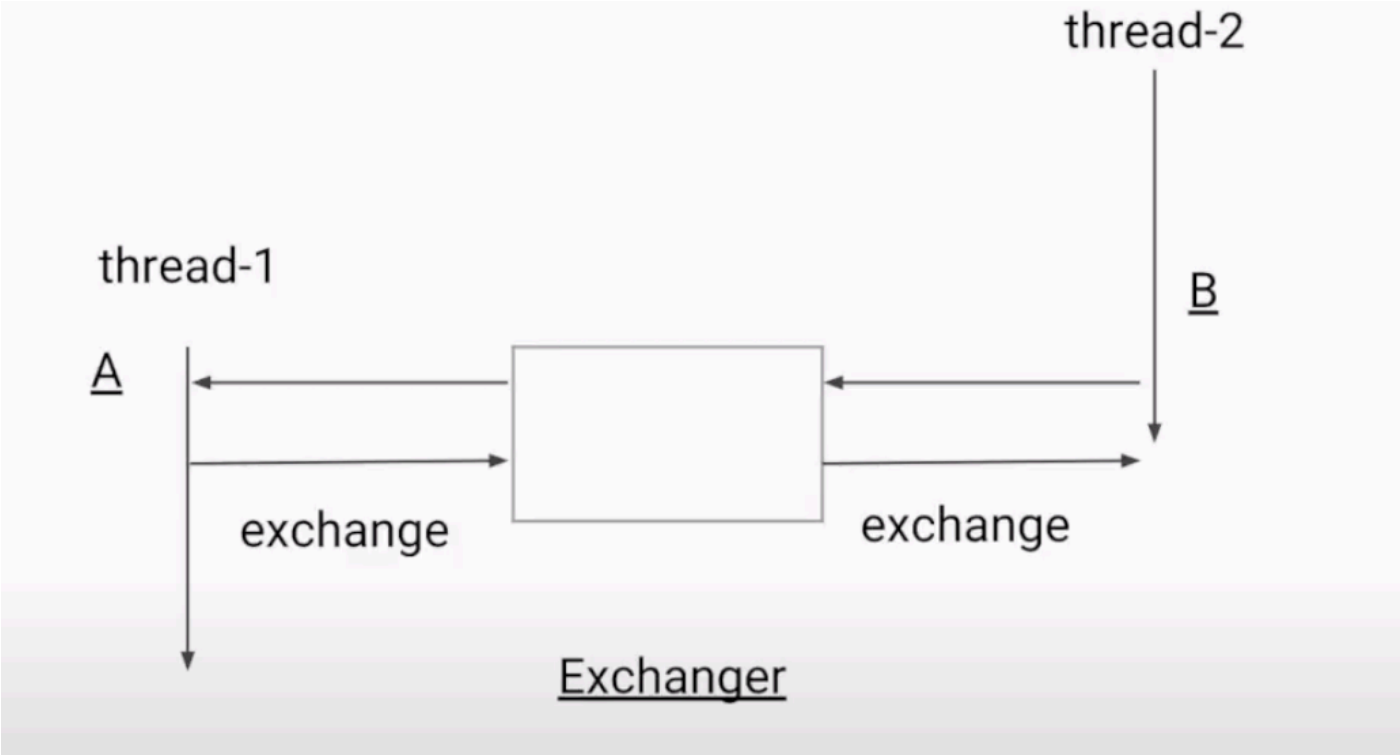
Exchanger is same as Synchronous queue but with handshake in both directions.



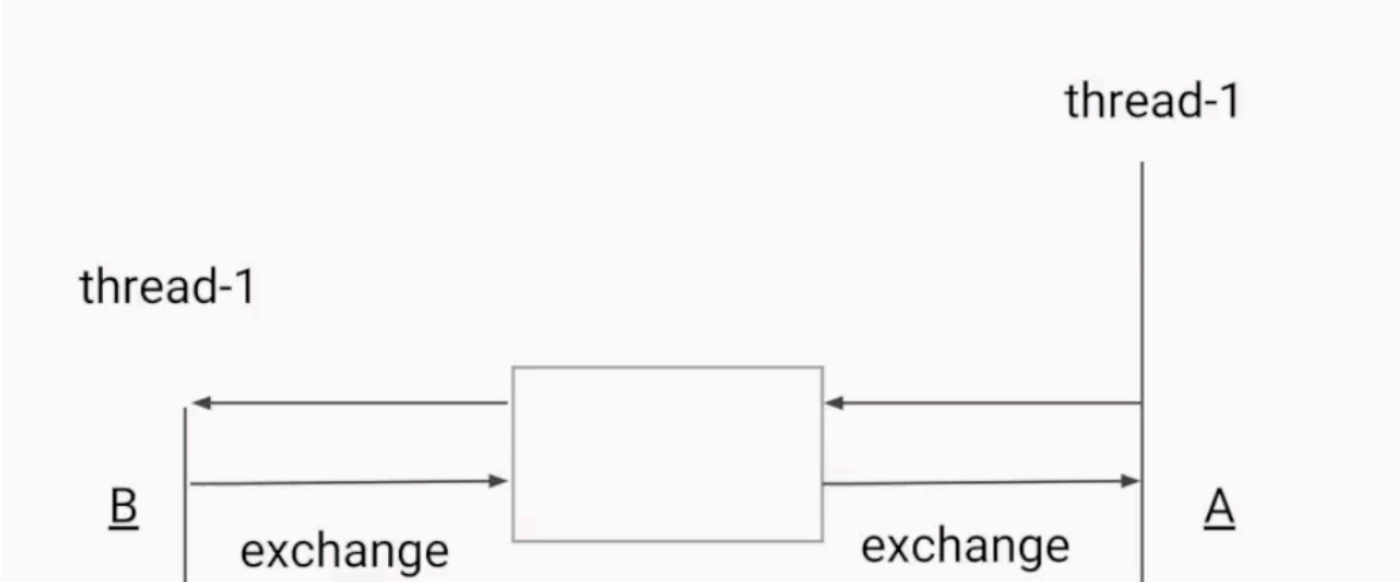


Exchanger

As per the above example Thread 2 wants to exchange item but their is not thread which is ready for exchange it then it is blocked as soon thread A is available for it.

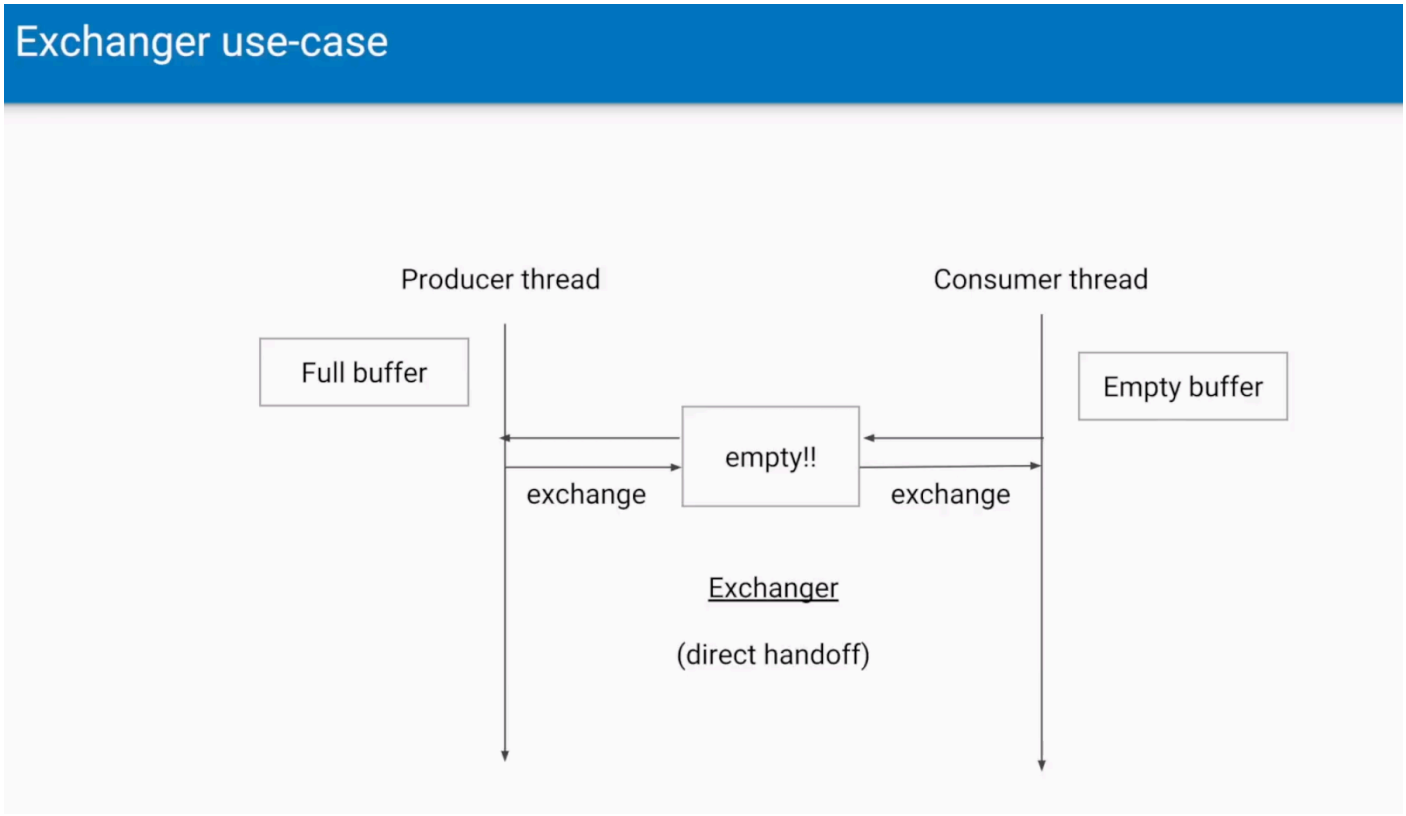


The B element is exchanged from Thread 2 to thread 1 and Item A from 1 to thread 2

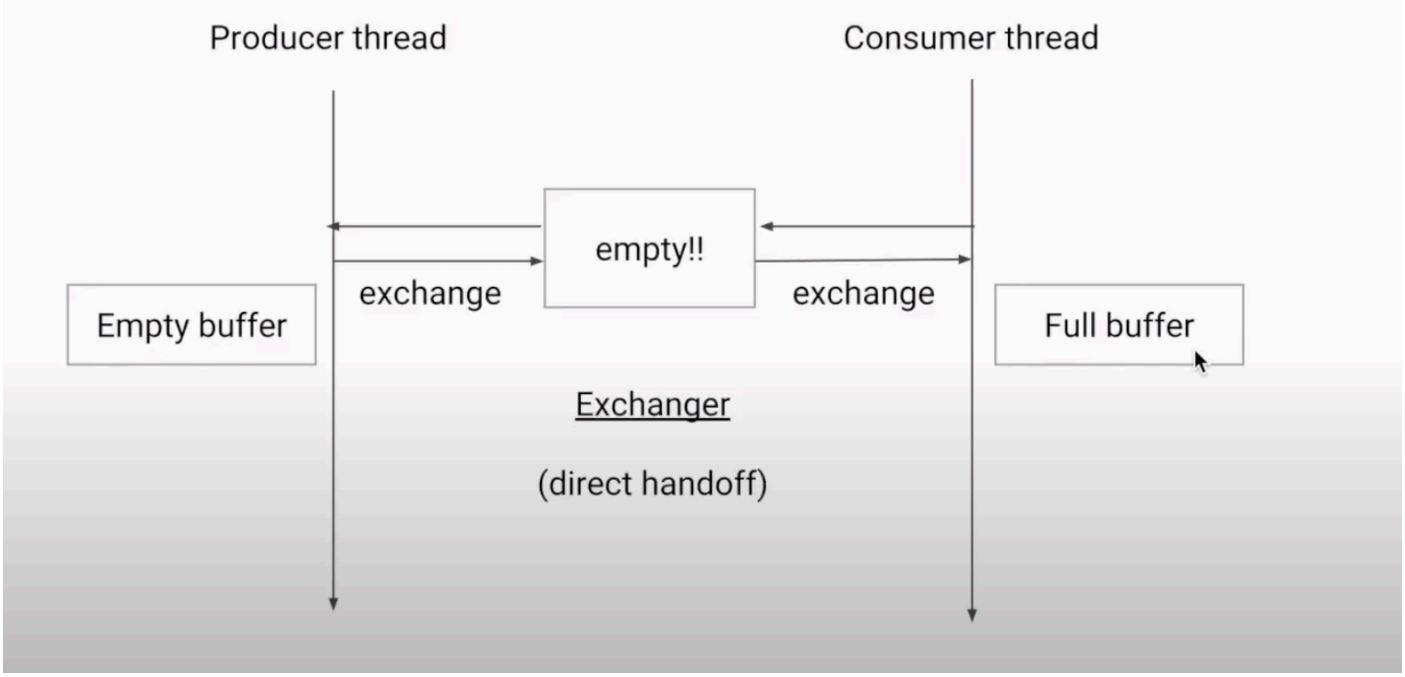


USE CASE: Exchange perfect usecase in buffer:

→ Producer and consumer exchanging full and empty buffer.



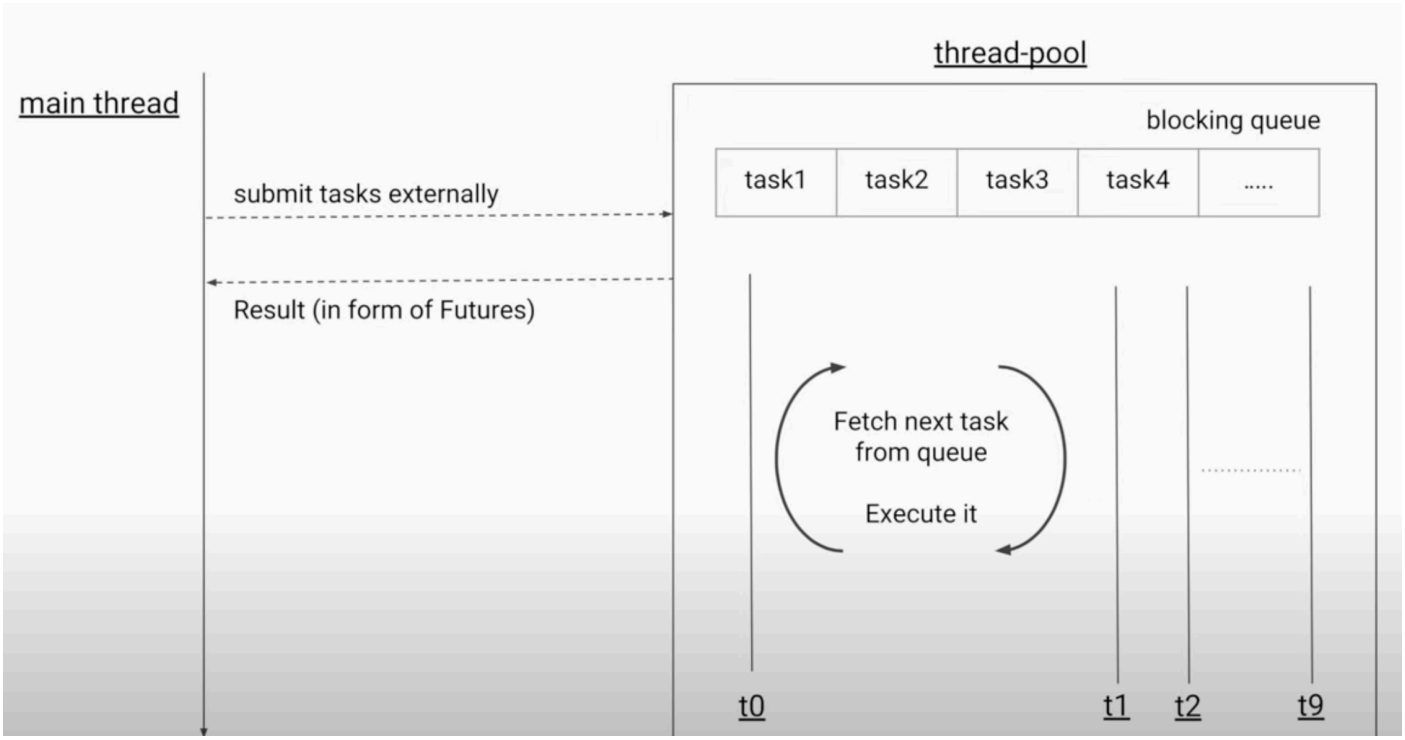
After this they exchange their buffer:



Fork Join pool:

Similar to the executor service

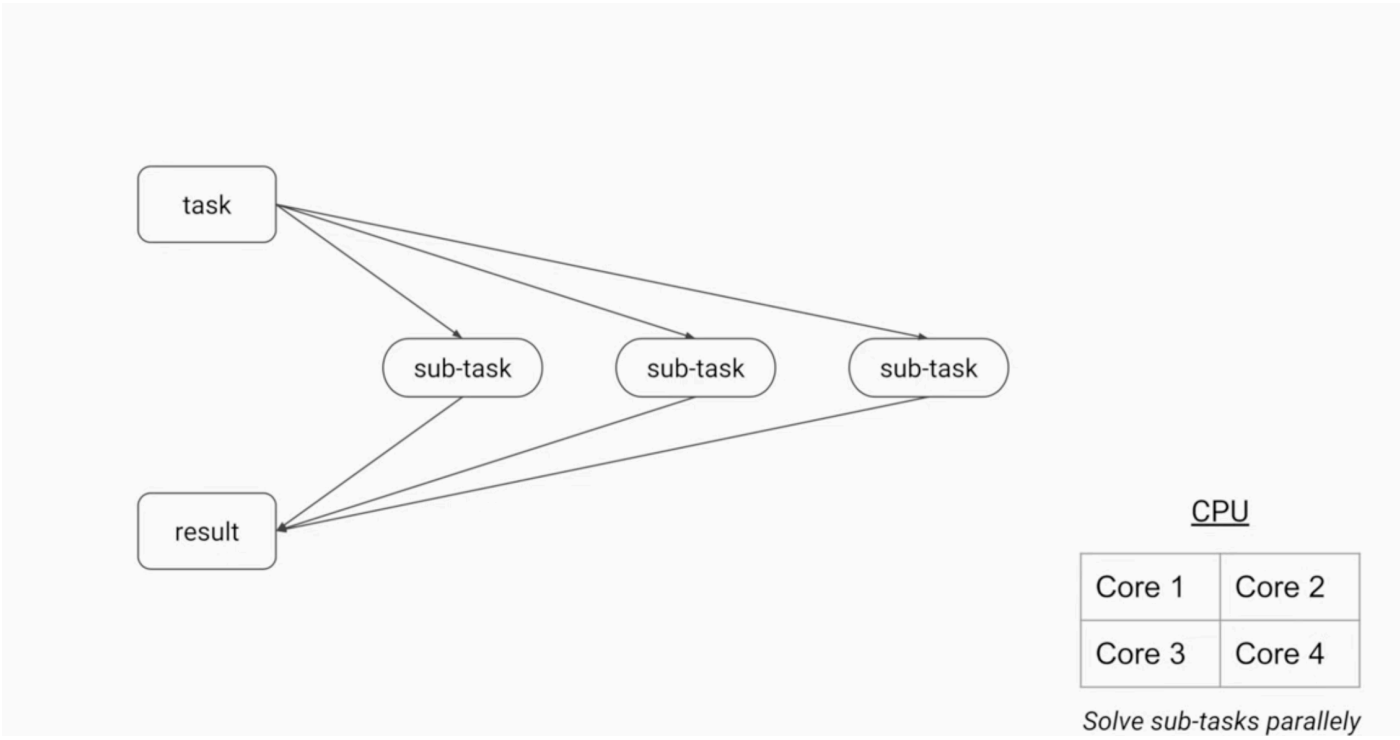
Eg of executor:



How different fork-join from the executor:

1. Task produces subtask aka fork join

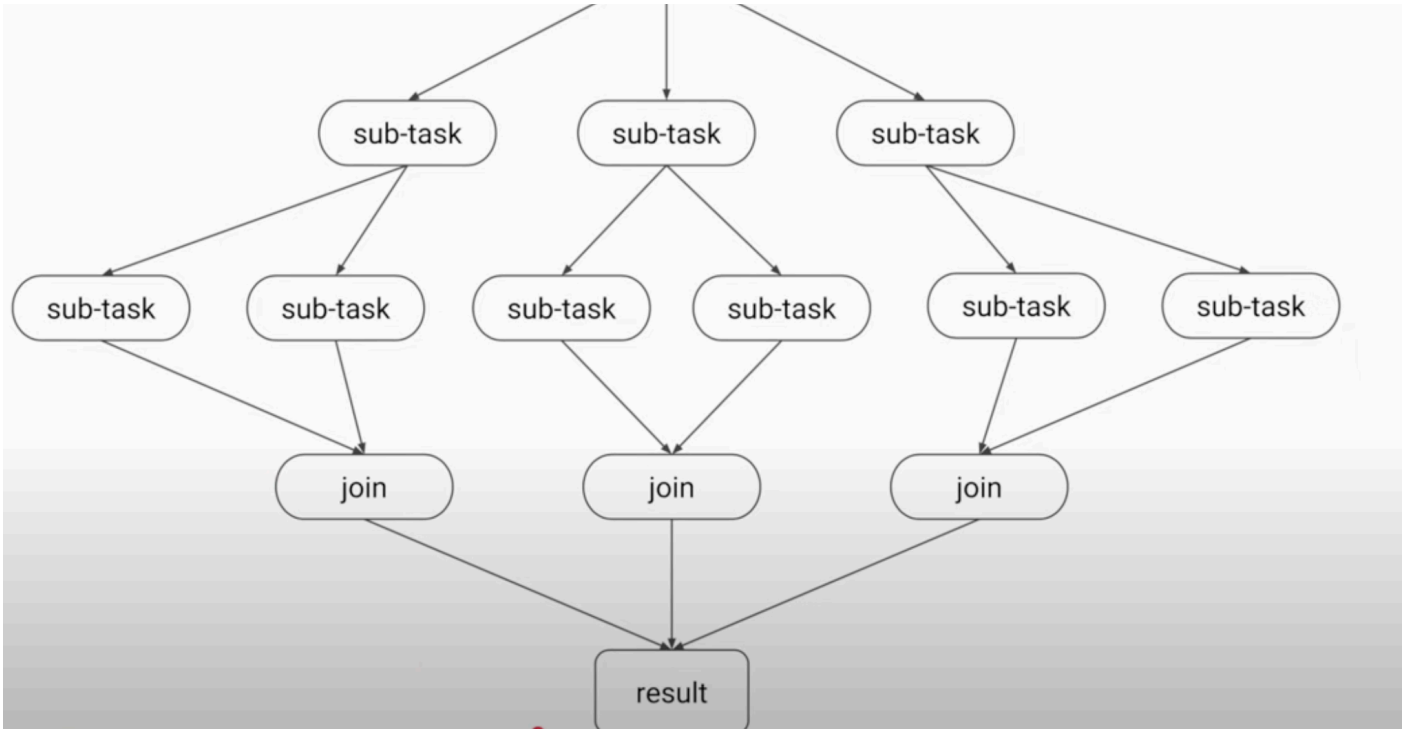
where we can break task CPU intentive into subtask



FORK + JOIN

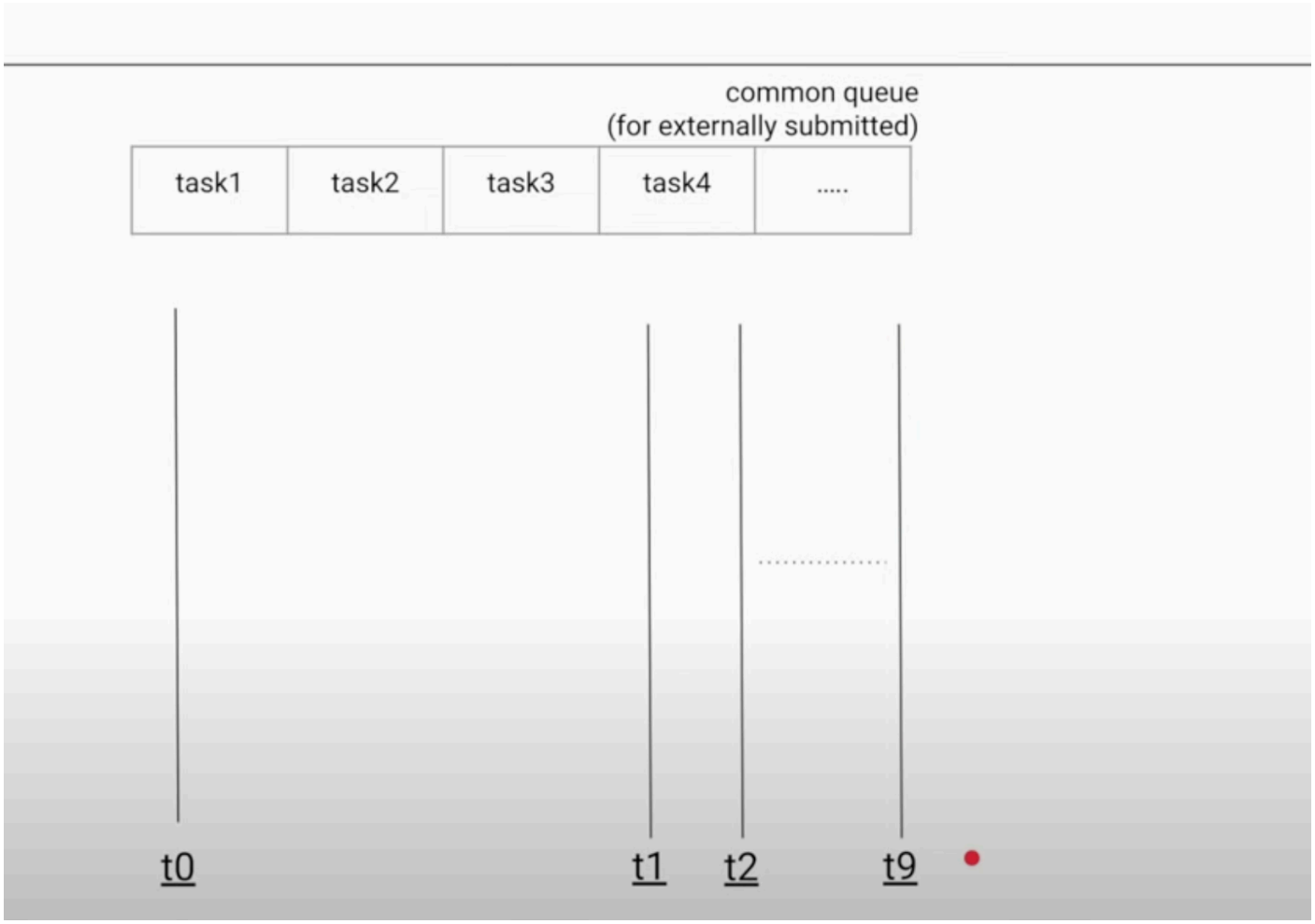
eg:

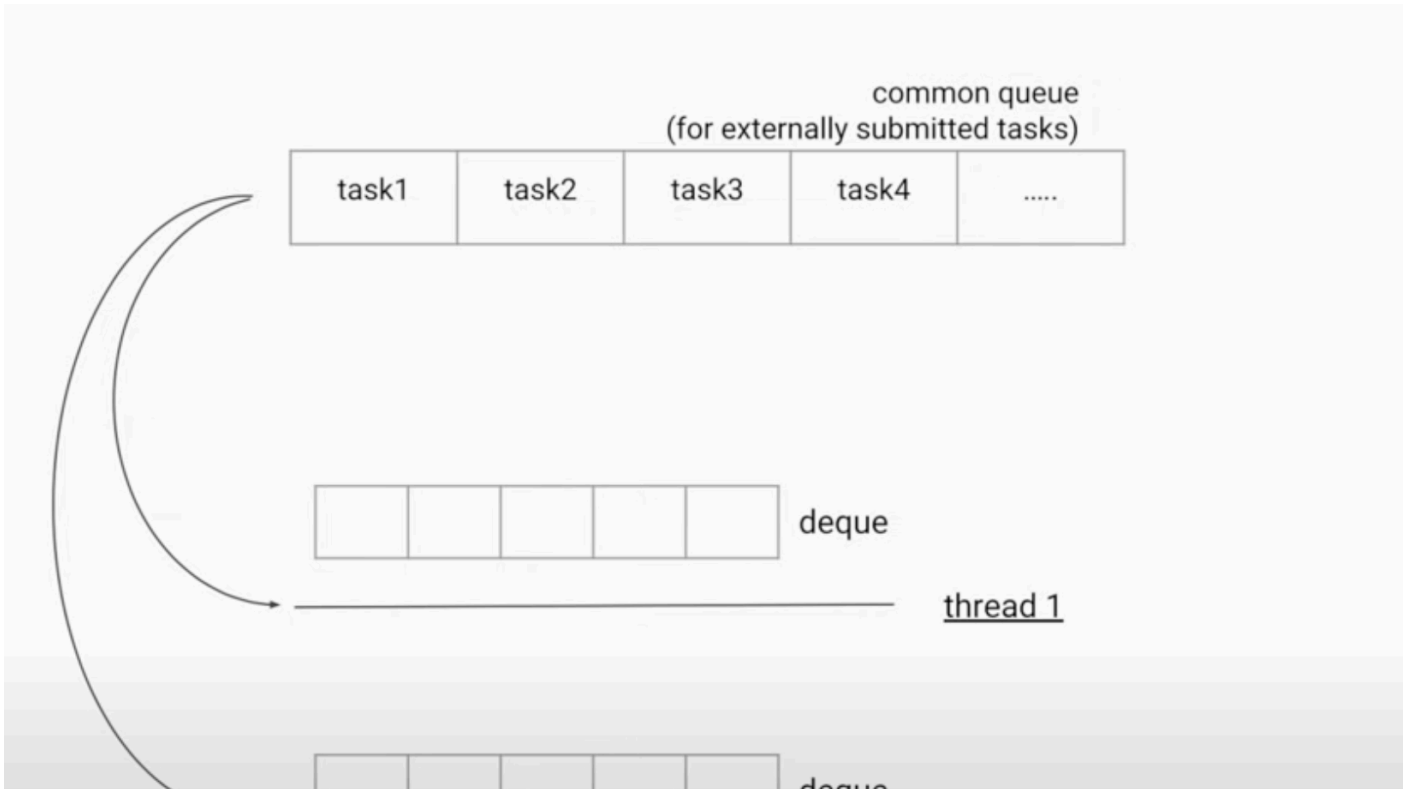
```
if (n <= 1) {
    return n;
} else {
    Fib f1 = new Fib(n - 1);  // sub-task 1
    Fib f2 = new Fib(n - 2);  // sub-task 2
    f1.solve();
    f2.solve();
    number = f1.number + f2.number;  // join results
    return number;
}
```

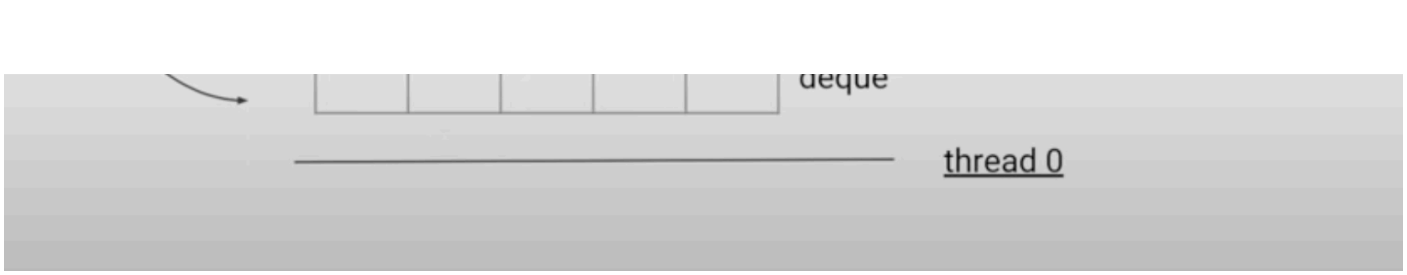


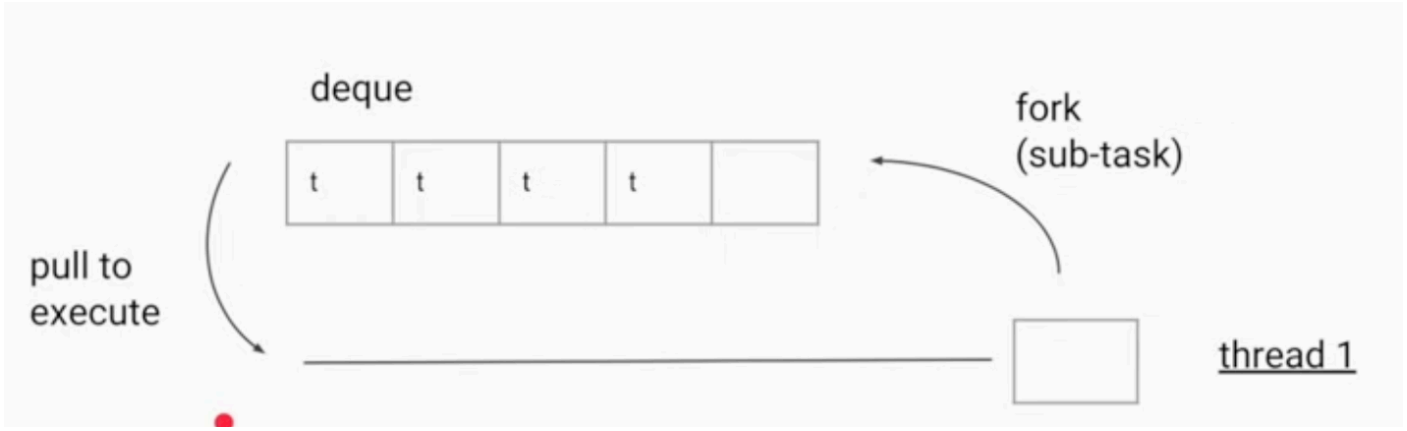

```
public Result solve(Task t) {

    split t into smaller tasks

    for each of these tasks
        solve(t_i)

    wait for all tasks to complete

    join all individual results

    return result
}
```

2. Per thread queuing and work stealing

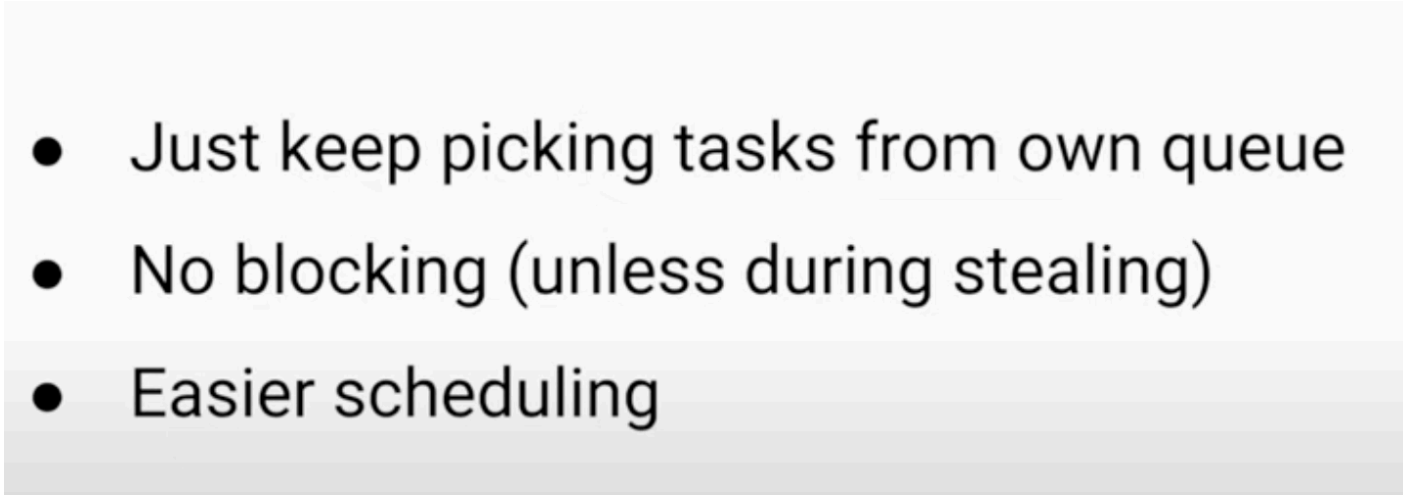




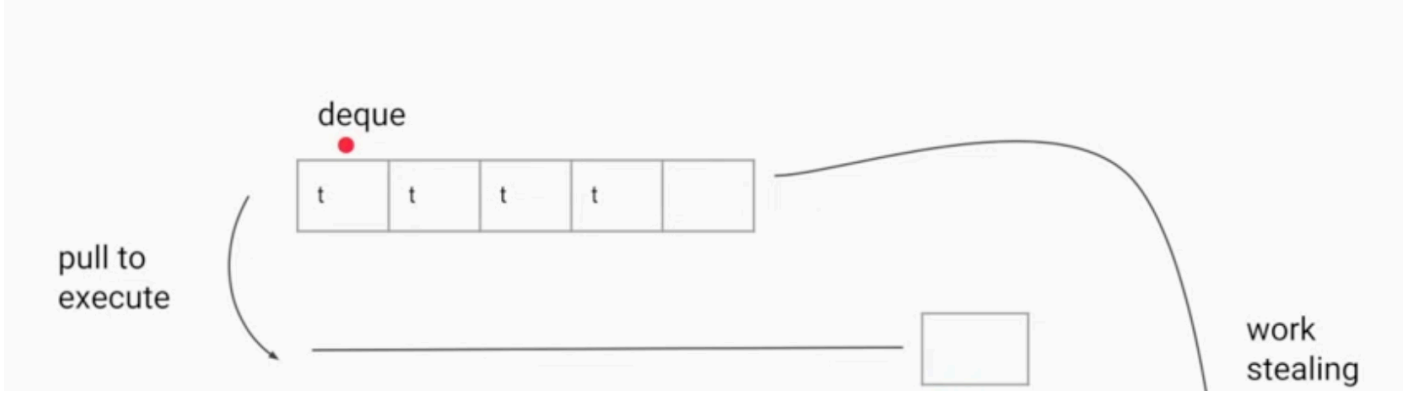


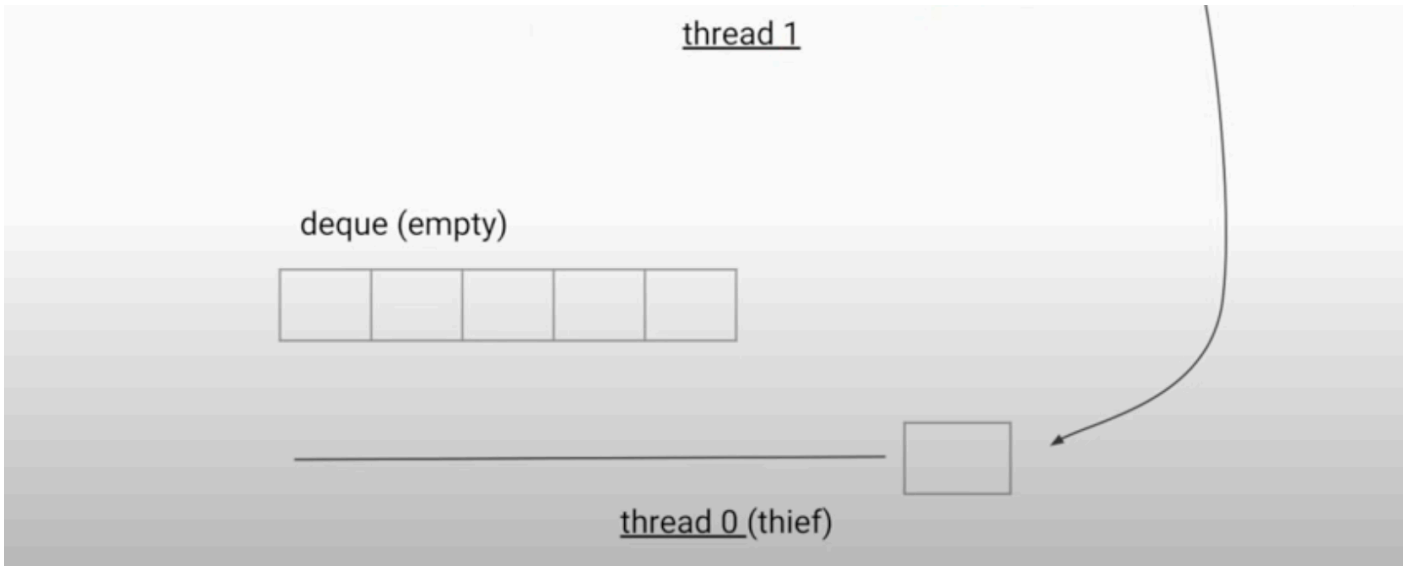


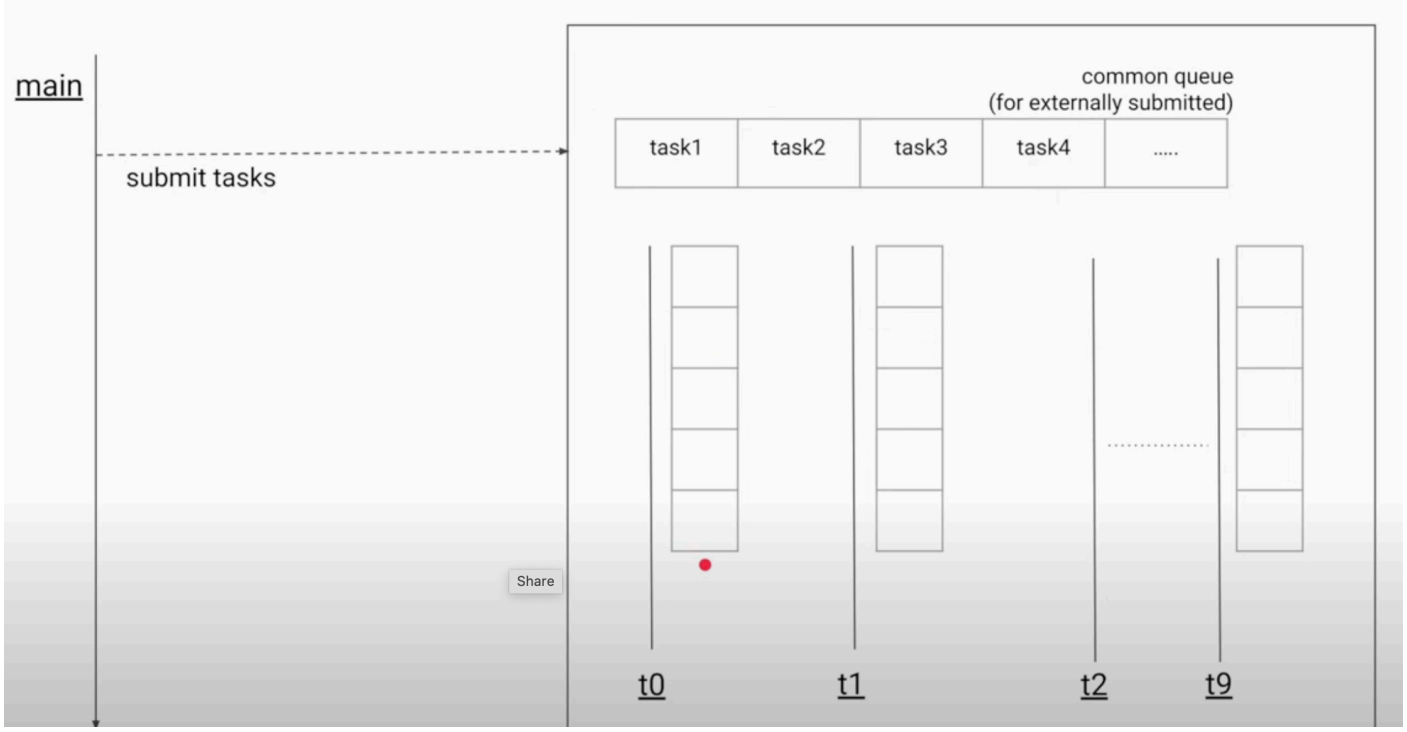
Advantage of queue per thread:



Work stealing:







	Call from non-fork/join clients	Call from within fork/join computations
Arrange async execution	`execute(ForkJoinTask)`	`fork()`
Await and obtain result	`invoke(ForkJoinTask)`	`invoke()`
Arrange exec and obtain Future	`submit(ForkJoinTask)`	`fork()` (ForkJoinTasks *are* Futures)

```
private class Fibonacci extends RecursiveTask<Integer> {

    final int n;
    Fibonacci(int n) { this.n = n; }

    public Integer compute() {
        if (n <= 1)
            return n;
        Fibonacci f1 = new Fibonacci( n: n - 1);
        f1.fork();
        Fibonacci f2 = new Fibonacci( n: n - 2);
        f2.fork();
        return f2.join() + f1.join();
    }
}
```

Ideal ForkJoinTasks...

- Avoid synchronization
- Do not use shared variables across tasks
- Do not perform Blocking IO operations
- Are pure functions
- Are isolated

available methods:

Category	Method	Meaning
Value	get() get(Timeout)	Get value of the tasks
Completion	cancel() isCancelled() isDone()	Cancel or check status of the task.

Use cases

- Sorting
- Matrix multiplication
- Best move finder for a game
- Tree traversal