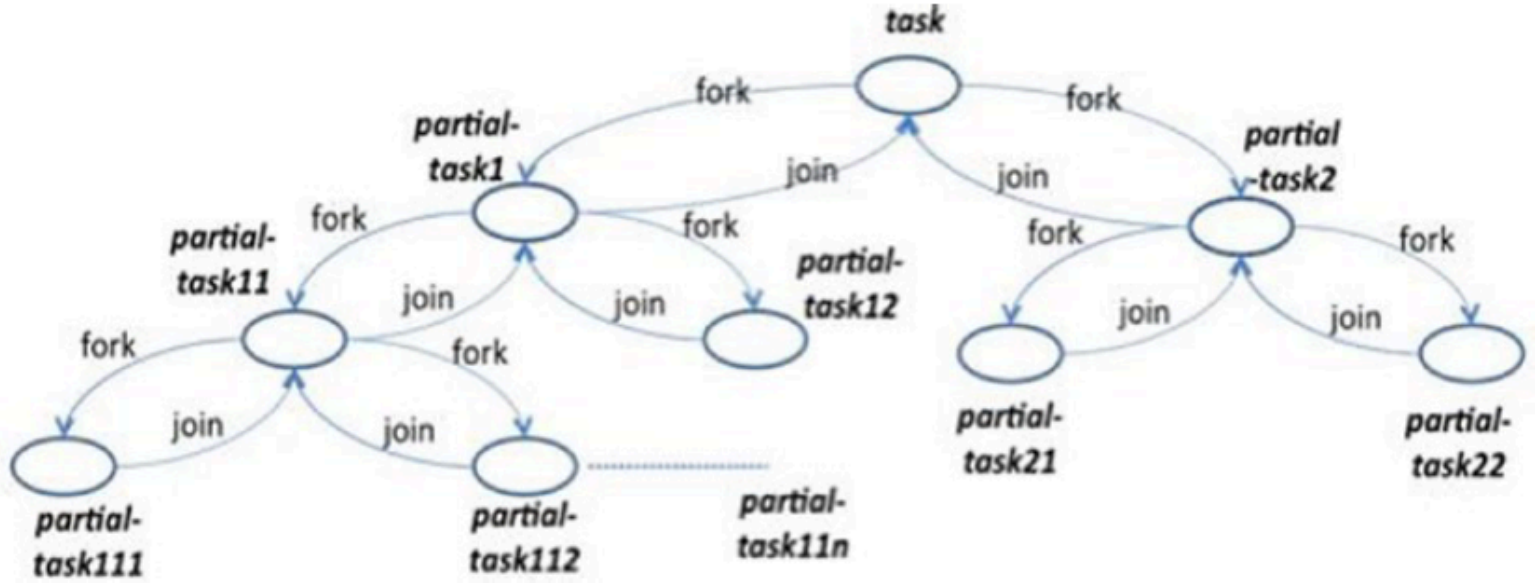
OCP Java SE 8

Fork-Join-Framework

Parallel fork/join framework

- Tasks die prozessorintensiv sind und sich in Teilaufgaben zerlegen lassen sind gute Kandidaten für ForkJoin
- Blockierende Tasks sind keine gute Kandidaten für ForkJoin
- divide-and-conquer parallel algorithm
- ForkJoinPool ist eine konkrete Implementierung
- Implementiert das ExecutorService interface
- Verfügt über eine Queue (Deque) mit Tasks, die an die Worker-Threads verteilt werden
- Work-Stealing

Rekursion

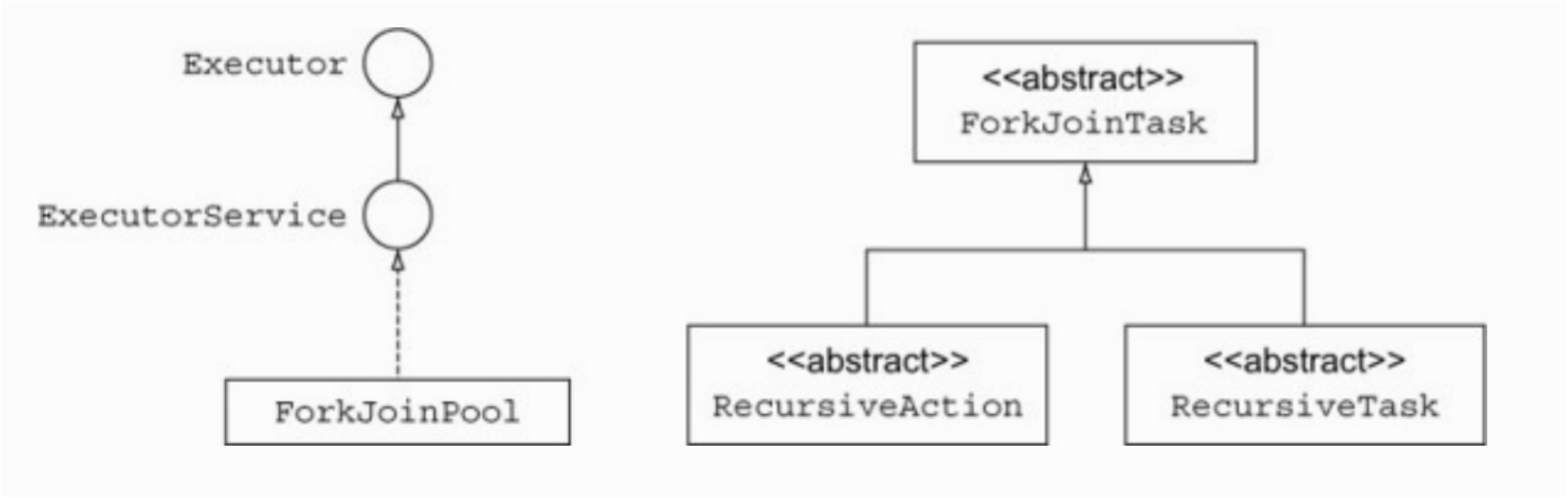

task
fork
fork
partial-
task1
join
join
partial
-task2
fork
fork
partial-
task11
fork
partial-
task12
join
join
fork
fork
fork
join
join
join
join
partial-
task111
partial-
task112
partial-
task11n
partial-
task21
partial-
task22

Parallel fork/join framework

- Fork (split)
  - in kleinere Subtasks zerlegen die konkurrierend ausgeführt werden können
- Join (merge)
  - Ergebnisse der Subtasks zu einem Ergebnis zusammenführen

Psedocode

```
forkJoinAlgorithm() {
    fork (split) the tasks;
    join the tasks;
    compose the results;
}
```


Executor
ExecutorService
ForkJoinPool
<<abstract>>
ForkJoinTask
<<abstract>>
RecursiveAction
<<abstract>>
RecursiveTask

fork/join framework

- RecursiveAction
  - compute() ohne Rückgabewert
- RecursiveTask
  - compute() mit einem Rückgabewert

ForkJoinPool

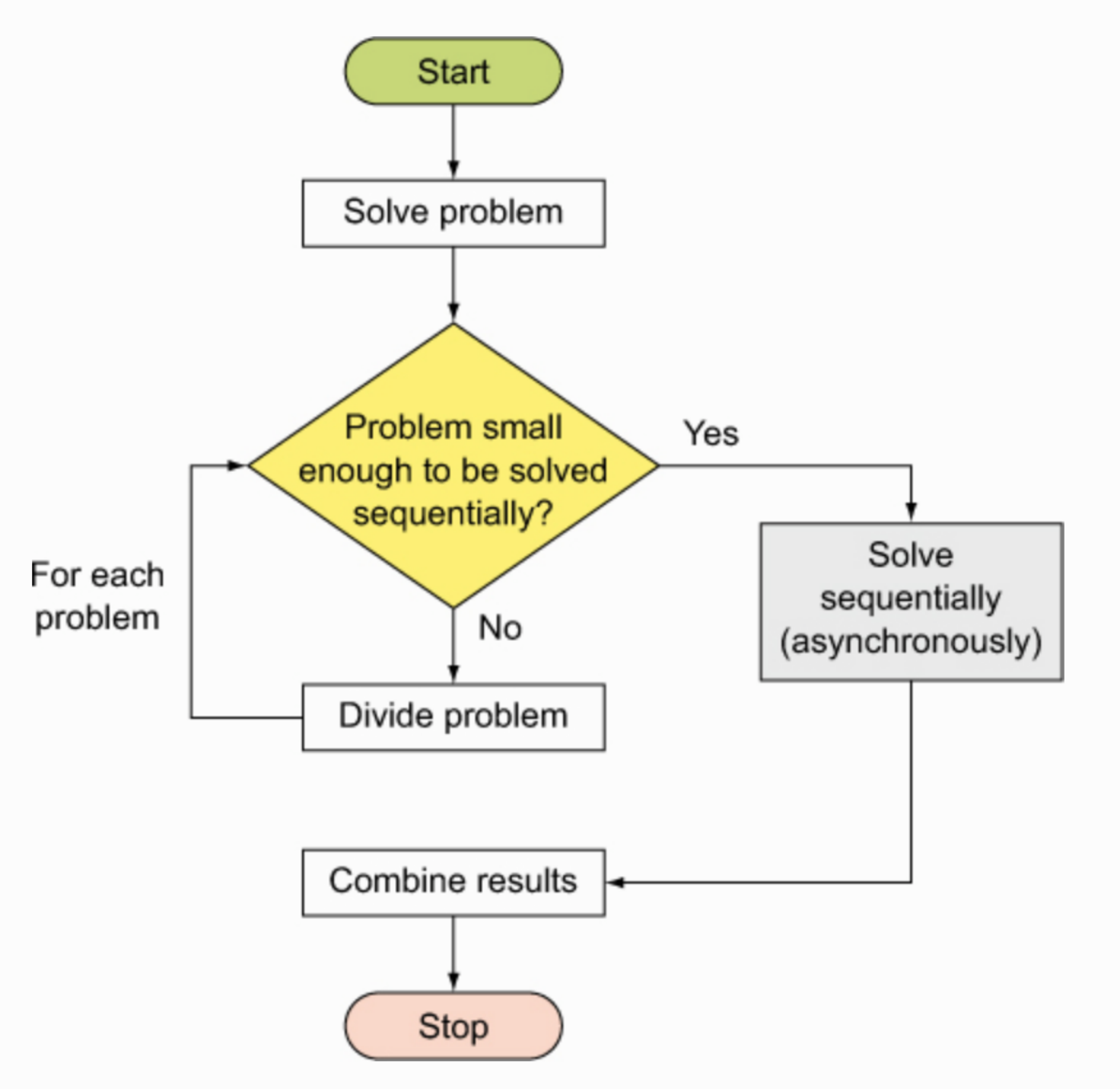
- void execute(ForkJoinTask<?> task)
    - Führt den Task aus
- <T> T invoke(ForkJoinTask<T> task)
    - Führt den Task aus und liefert den berechneten Wert zurück
- <T> List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks)
    - Führt alle übergebenen Tasks aus, liefert eine Liste von Future Objekten
- boolean isTerminated()
    - Liefert true, wenn alle Tasks fertig sind
- int getParallelism(), int getPoolSize(), long getStealCount(), int getActiveThreadCount()
    - Prüfen den Status des Pools
- <T> ForkJoinTask<T> submit(Callable<T> task)
  <T> ForkJoinTask<T> submit(ForkJoinTask<T> task)
  ForkJoinTask<?> submit(Runnable task)
  <T> ForkJoinTask<T> submit(Runnable task, T result)
- Führt den Task aus

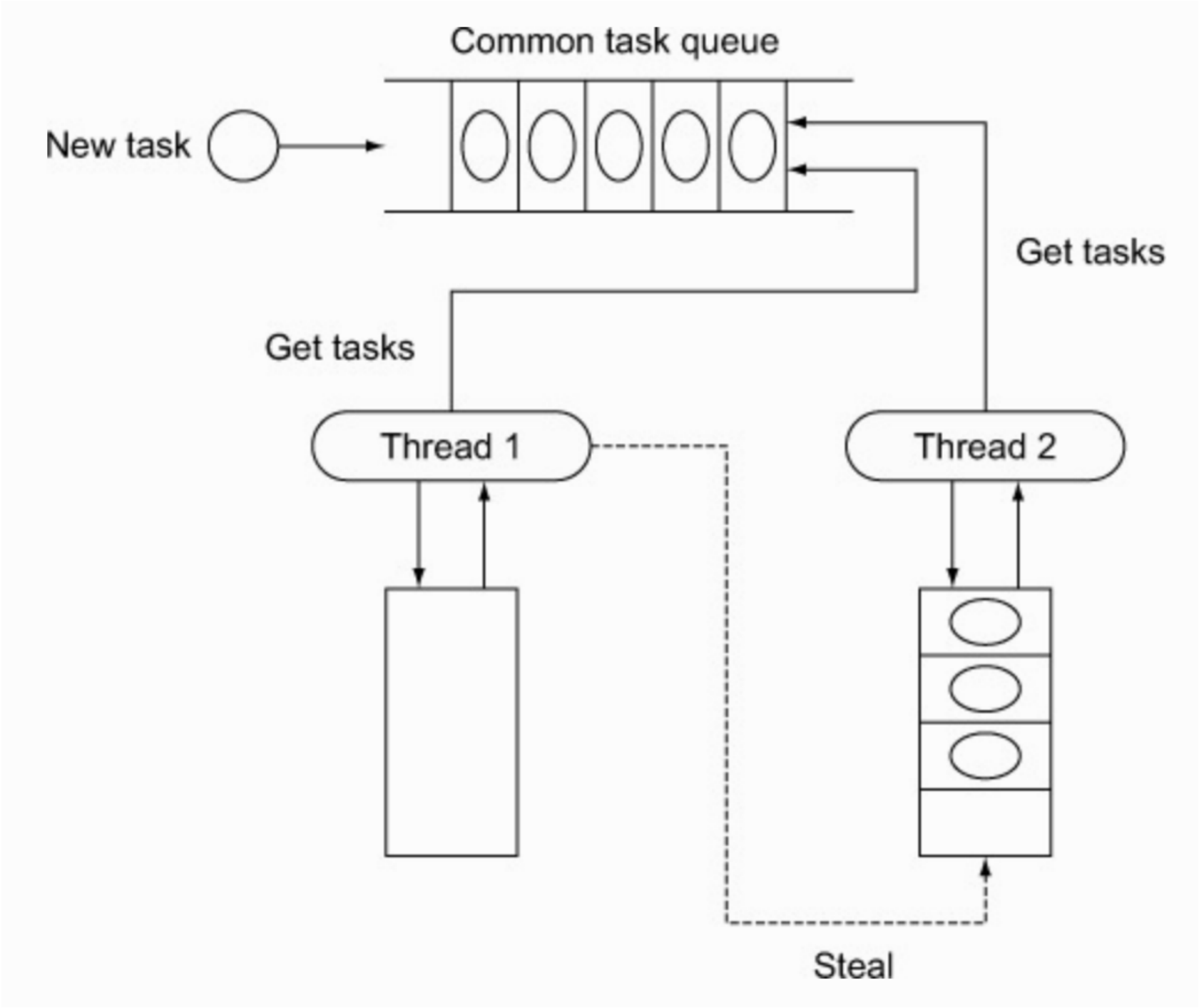
ForkJoinTask

- boolean cancel(boolean mayInterruptIfRunning)
    - Versucht die Ausführung des Tasks abzubrechen
- ForkJoinTask<V> fork()
    - Führt den Task asynchron aus
- V join()
    - Liefert das Ergebnis der Berechnung zurück, wenn sie abgeschlossen ist
- V get()
    - Liefert das Ergebnis der Berechnung zurück. Wartet bis die Berechnung abgeschlossen ist
- V invoke()
  static <T extends ForkJoinTask<?>> Collection<T> invokeAll(Collection<T> tasks)
    - Beginnt die Ausführung des Tasks. Wartet bis die Berechnung beendet wurde und liefert den Wert zurück
- boolean isCancelled()
    - Liefert true, wenn der Task abgebrochen wurde
- boolean isDone()
    - Liefert true, wenn der Task ist fertig

Ablauf

- Create a ForkJoinTask.
- Create the ForkJoinPool.
- Start the ForkJoinTask.


Start
Solve problem
Problem small enough to be solved sequentially?
Yes
Solve sequentially (asynchronously)
For each problem
No
Divide problem
Combine results
Stop


Common task queue
New task
Get tasks
Get tasks
Thread 1
Thread 2
Steal