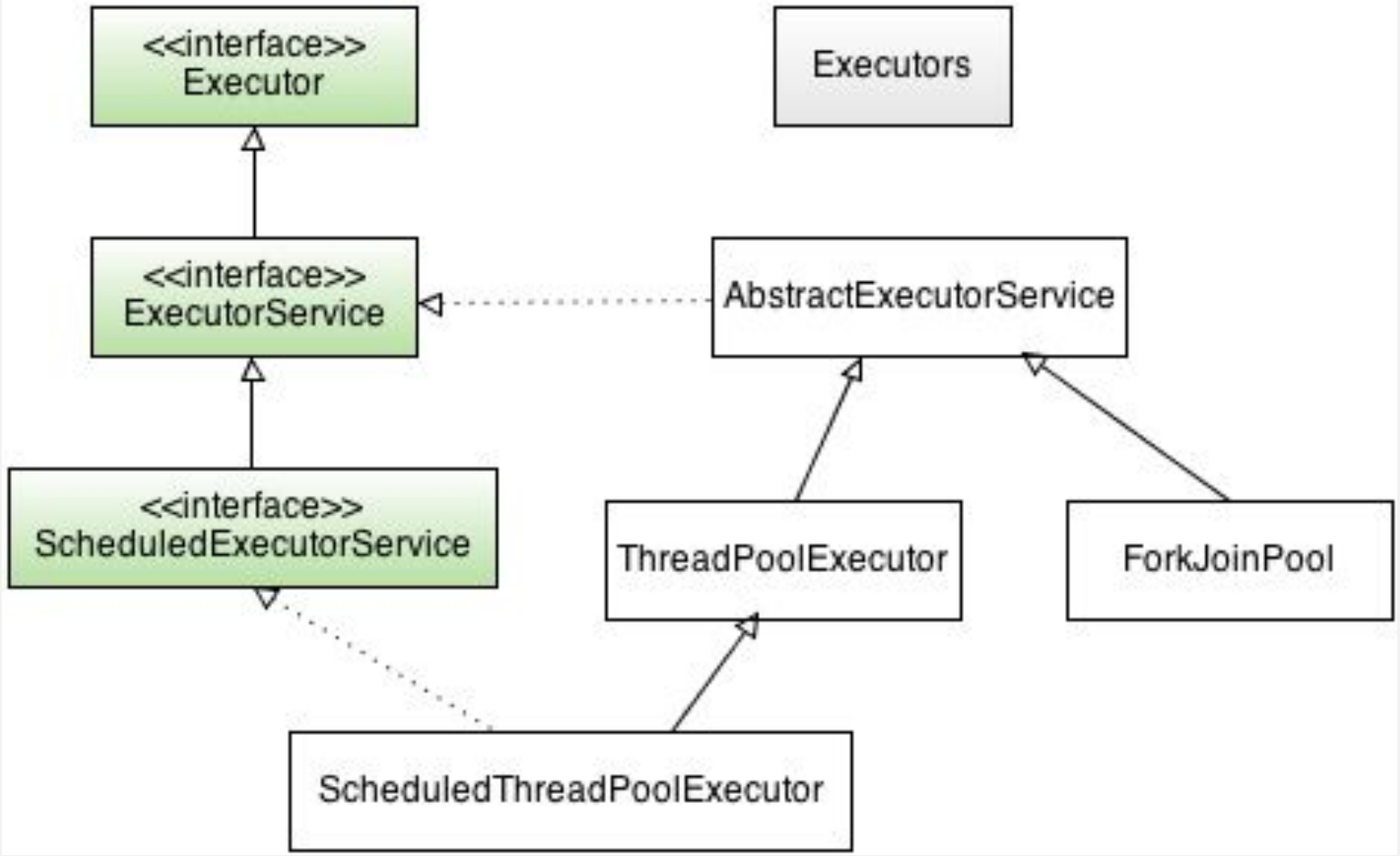
Executors, Thread Pool, Fork/Join

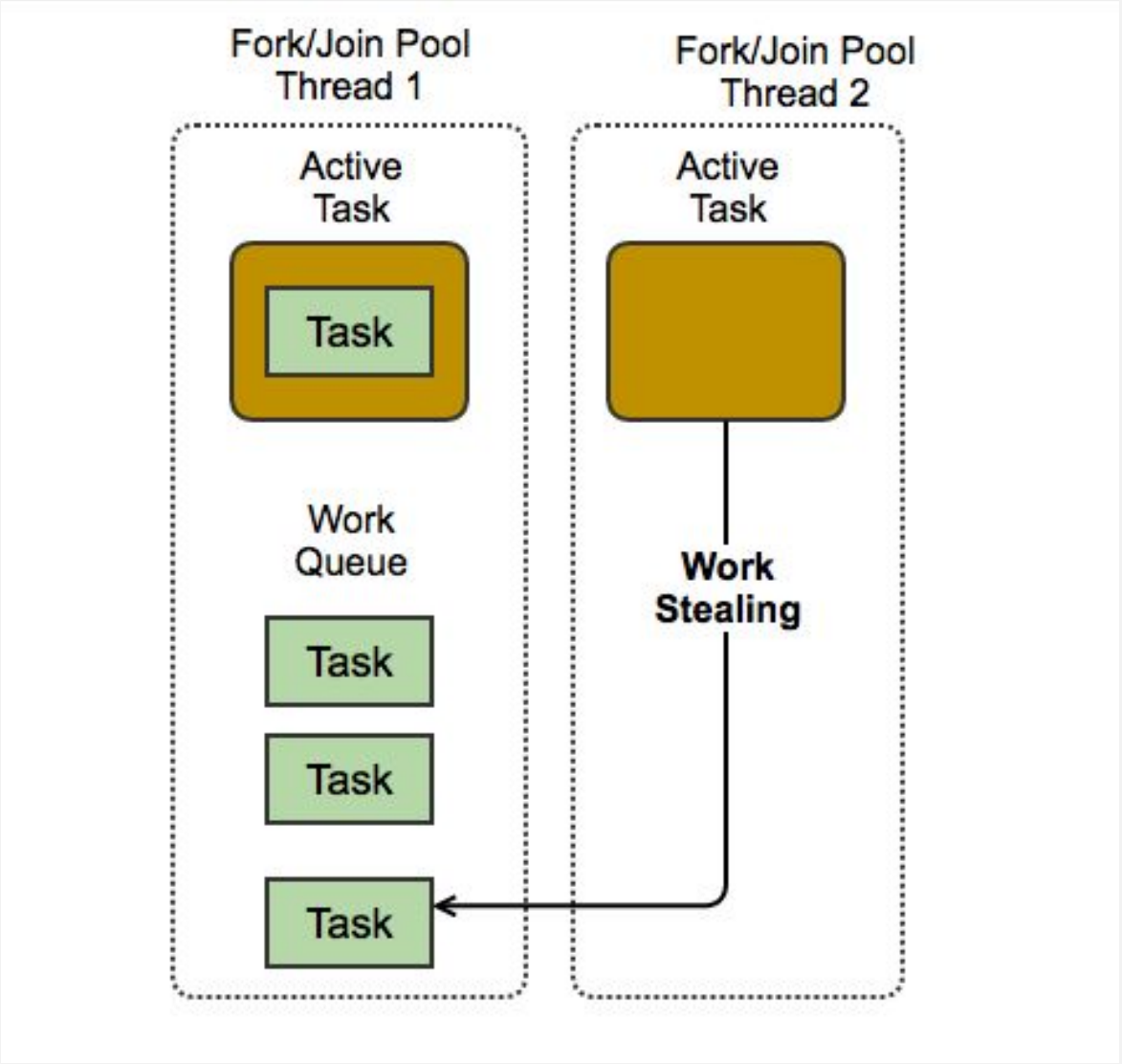
LIU Yuanyuan
GUO Xiaoqing

Executors --- Le framework Executor



Thread Pool

Fork/Join


Fork/Join Pool
Thread 1
Fork/Join Pool
Thread 2
Active
Task
Active
Task
Task
Work
Queue
Work
Stealing
Task
Task
Task

Utilisation_Executors

`Executor` Interface

fournit une méthode unique, *execute*, conçue pour remplacer un idiome commun de création de threads. Si *r* est un objet *Runnable*, et *e* est une objet *Executor*, on peut remplacer < *(new Thread(r)).start();* > par < *e.execute(r);* >

`ExecutorService` Interface

complète *execute* avec une méthode *submit*.

```
threadPoolExecutor.submit(new Runnable()
public Future<?> submit(Runnable task) {
    if (task == null) throw new NullPointerException();
    RunnableFuture<Void> ftask = newTaskFor(task, null);
    execute(ftask);
    return ftask;
}
```

`ScheduledExecutorService` Interface

complète les méthodes de son parent *ExecutorService* avec *schedule*, qui exécute une tâche *Runnable* ou *Callable* après un délai spécifié.

Utilisation_Thread Pool

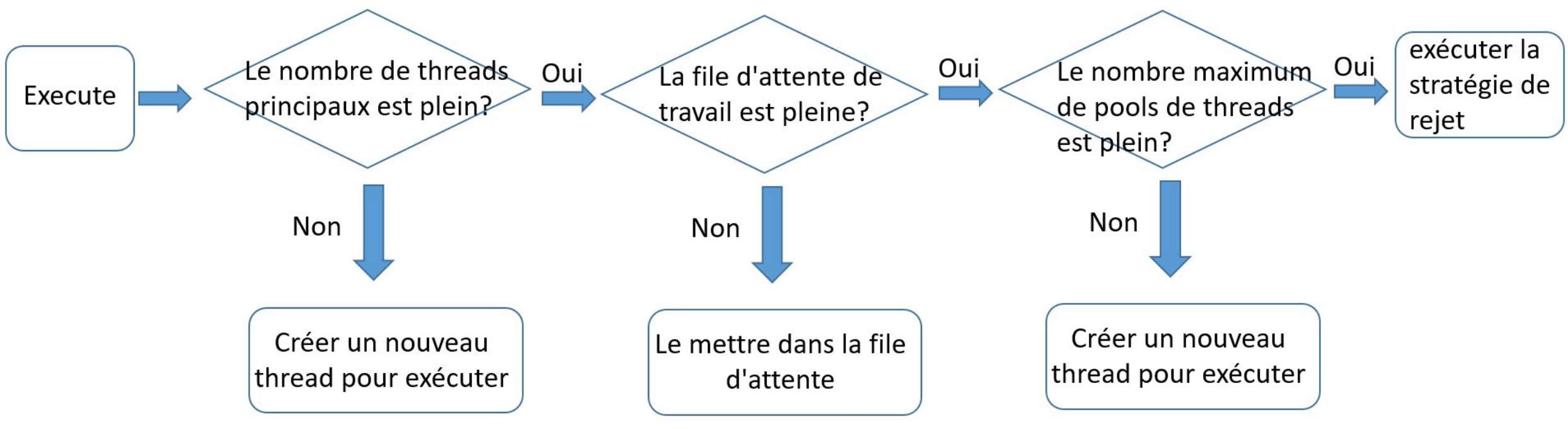
java.util.concurrent.ThreadPoolExecutor
java.util.concurrent.ScheduledThreadPoolExecutor

newFixedThreadPool(int nThreads)

newCachedThreadPool

newSingleThreadExecutor

Utilisation_Thread Pool



Utilisation_Fork/Join

```
if ( problem.size() > DEFAULT_SIZE) {
        divideTasks();
        executeTask();
        taskResults = joinTasksResult();
        return taskResults;
} else {
        taskResults = solveBasicProblem();
        return taskResults;
}
```

La classe ForkJoinTask fournit les deux méthodes suivantes pour la prise en charge :

- Méthode fork (): Cette méthode peut envoyer une sous-tâche à l'exécuteur Fork / Join.
- Méthode join (): Cette méthode peut attendre qu'une sous-tâche termine son exécution et renvoie son résultat.

Les impacts sur les performances

- Avantage de ThreadPool
- Risque de ThreadPool
- ForkJoinPool possède une meilleur performance que les autres ThreadPool d’ExecutorService

Merci pour votre patience!