Stick a fork in it

An attempt to summarise the Fork-Join framework through the same titled series of articles by Goetz.

Oh, and another article which mentions almost the same thing, only 4 years later.

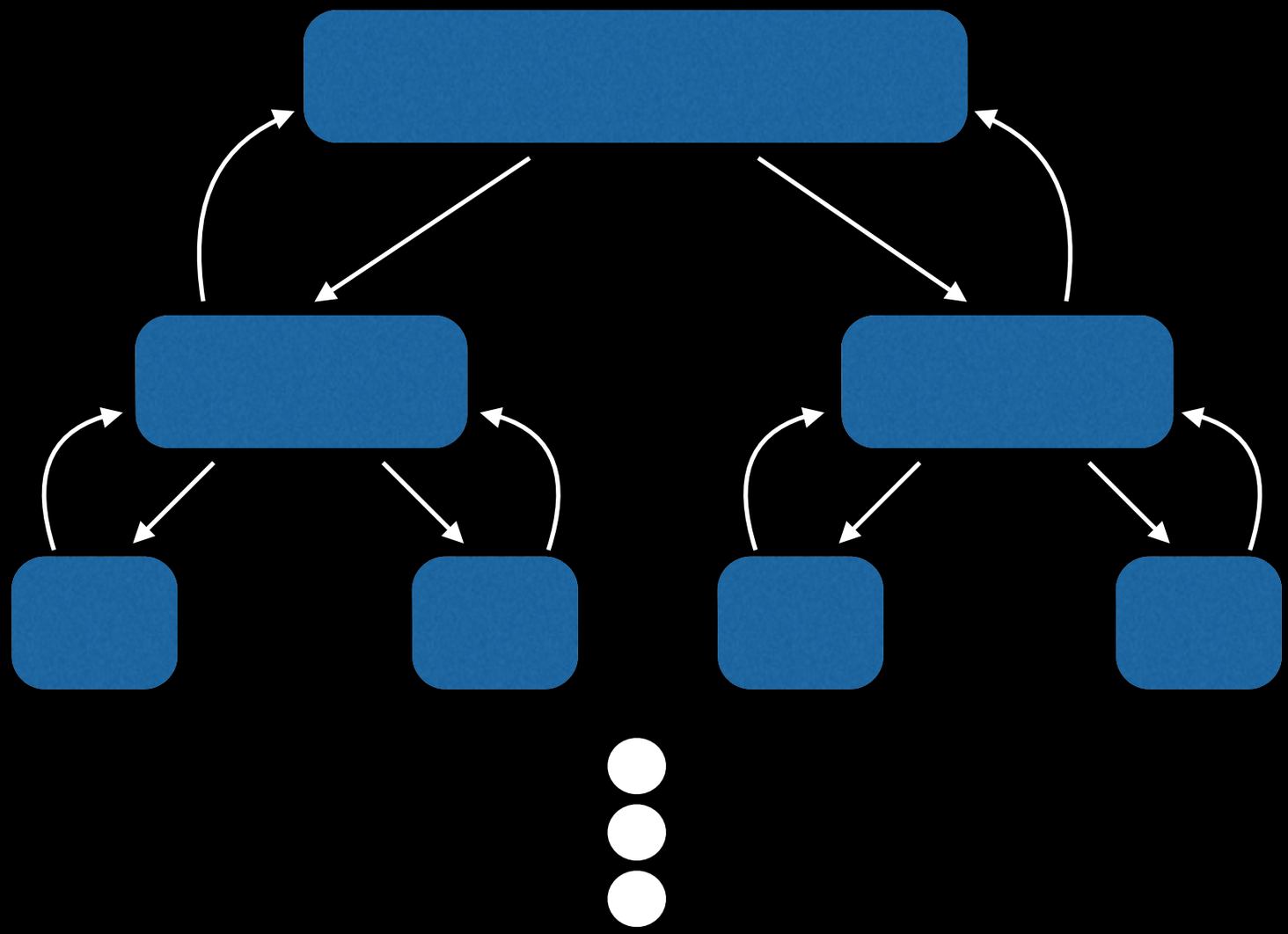
Why another framework?

- *asynchrony* vs. *concurrency*
- finer-grained parallelism

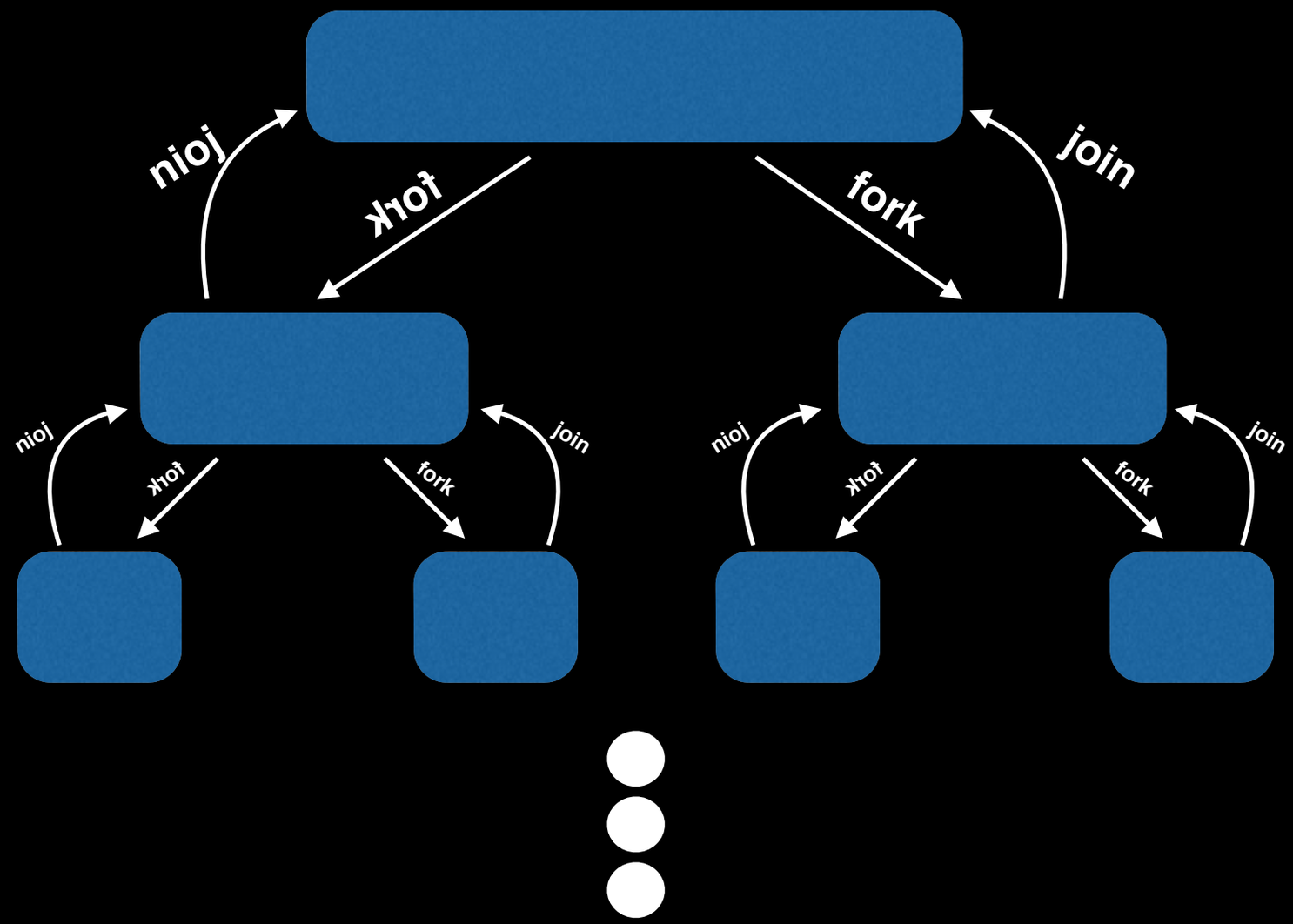
F-i-n-e-r-g-r-a-i-n-e-d P-a-r-a-l-l-e-l-i-s-m

- Compute-intensive,
- independent tasks
- A shift in perf. metric: PU utilisation -> user perception

Divide and conquer



Fork and join



Some classes

- `ForkJoinTask`
  - `RecursiveAction`
  - `RecursiveTask<>`
    - *`compute()`*
- `ForkJoinPool`
  - executor for `ForkJoinTask`
    set of work thieves
  - default constructor adjusts the #threads according to the #PUs

```
class FolderSearchTask extends RecursiveTask<Long> {
    private final Folder folder;
    private final String searchedWord;

    FolderSearchTask(Folder folder, String searchedWord) {
        super();
        this.folder = folder;
        this.searchedWord = searchedWord;
    }

    @Override
    protected Long compute() {
        long count = 0L;
        List<RecursiveTask<Long>> forks = new LinkedList<>();
        for (Folder subFolder : folder.getSubFolders()) {
            FolderSearchTask task = new FolderSearchTask(subFolder, searchedWord);
            forks.add(task);
            task.fork();
        }
        for (Document document : folder.getDocuments()) {
            DocumentSearchTask task = new DocumentSearchTask(document, searchedWord);
            forks.add(task);
            task.fork();
        }
        for (RecursiveTask<Long> task : forks) {
            count = count + task.join();
        }
        return count;
    }
}
private final ForkJoinPool forkJoinPool = new ForkJoinPool();

Long countOccurrencesInParallel(Folder folder, String searchedWord) {
    return forkJoinPool.invoke(new FolderSearchTask(folder, searchedWord));
}
```

ParallelArray

- a more declarative way (of defining operations on data sets)
  - similar to SQL queries
- filtering, application, mapping, replacement, summarisation
- transparent parallelisation

```
ParallelArray<Student> students = new ParallelArray<Student>(fjPool, data);
double bestGpa = students.withFilter(isSenior)
                         .withMapping(selectGpa)
                         .max();

public class Student {
    String name;
    int graduationYear;
    double gpa;
}

static final Ops.Predicate<Student> isSenior = new Ops.Predicate<Student>() {
    public boolean op(Student s) {
        return s.graduationYear == Student.THIS_YEAR;
    }
};

static final Ops.ObjectToDouble<Student> selectGpa = new Ops.ObjectToDouble<Student>() {
    public double op(Student student) {
        return student.gpa;
    }
};
```

Not included in Java 7. May be in 8

Summary

- Fork-Join framework
  - ideal for divide and conquer
- `ParallelArray`
  - uses fork-join under the hood
  - relieves the programmer of parallelisation duties

Exercise

- Implement int BinarySearch(int[] sortedArr, int value)
  - returns the first occurrence
  - you can use Arrays.sort to sort an integer array filled with random integers
- Experiment with the “sequential threshold”. How does it affect the performance?

Double-click to ask a question

- Double-click to discuss