What's New In Java 7?

Dominic Brügger
Puzzle ITC

Diamond <>

```
Map<String, List<Person>> personsByName =
    new HashMap<String, List<Person>>();
```

Old school...

```
Map<String, List<Person>> personsByName = new HashMap<>();
```

Java 7

Try with Resources

```java
public void copyFile(File src, File dest) throws IOException {
   InputStream in = new FileInputStream(src);
   OutputStream out = new FileOutputStream(dest);

   byte[] buffer = new byte[1024];
   int n;
   while ((n = in.read(buffer)) >= 0) {
      out.write(buffer, 0, n);
   }
}
```

what if...

```java
public void copyFile(File src, File dest) throws IOException {
   InputStream in = new FileInputStream(src);
   try {
      OutputStream out = new FileOutputStream(dest);
      try {
         byte[] buffer = new byte[1024];
         int n;
         while ((n = in.read(buffer)) >= 0) {
            out.write(buffer, 0, n);
         }
      } finally {
         if (out != null) {
            out.close();
         }
      }
   } finally {
      if (in != null) {
         in.close();
      }
   }
}
```

Correct, but complex!

```
public void copyFile(File src, File dest) throws IOException {
   try (InputStream in = new FileInputStream(src);
        OutputStream out = new FileOutputStream(dest)) {
      byte[] buffer = new byte[1024];
      int n;
      while ((n = in.read(buffer)) >= 0) {
         out.write(buffer, 0, n);
      }
   }
}
```

Java 7

Strings in Switch

```
switch (action) {
case "red":
   stop();
   break;
case "orange":
   slowDown();
   break;
case "green":
   go();
   break;
}
```

Fork / Join

java.lang.Thread
java.lang.Runnable

Java < 5

wait()
notify()
synchronized

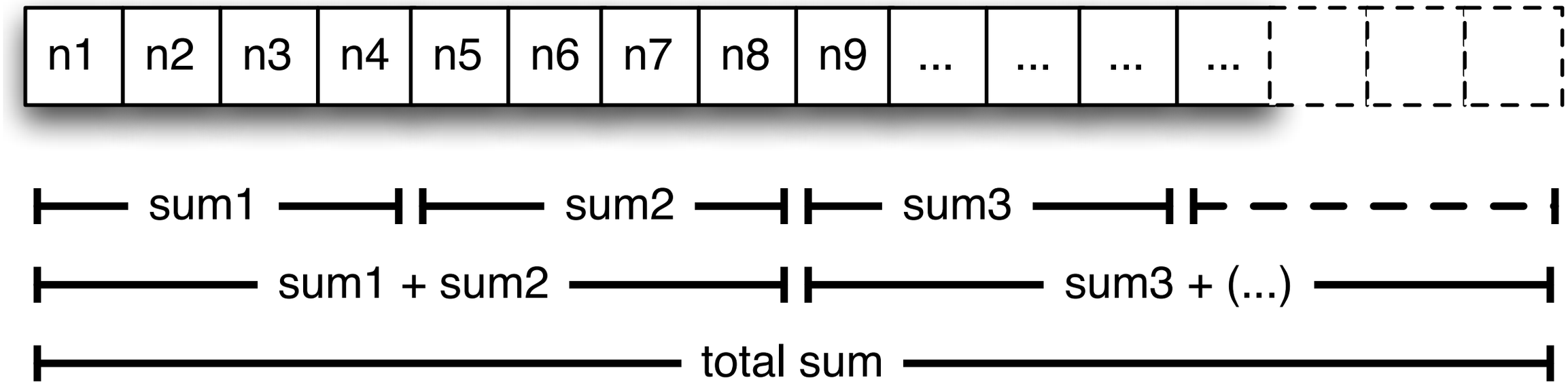
Java 5 + 6

java.util.concurrent

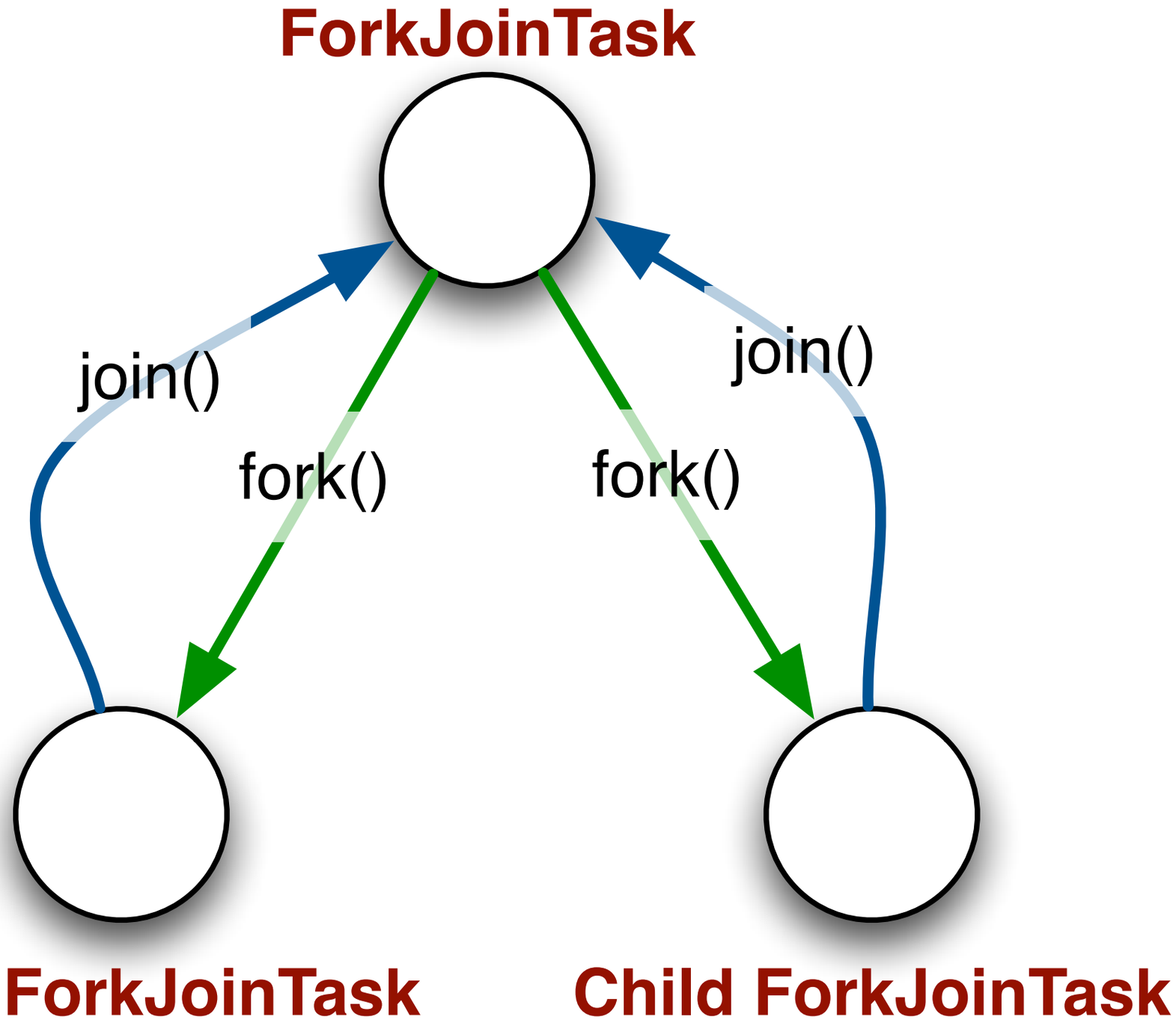
Executors
Concurrent Collections
Concurrent Queues
Atomic Variables
Synchronization Patterns
Locks

Java 7

ForkJoinPool
ForkJoinTask
RecursiveAction
RecursiveTask



Divide and Conquer!


ForkJoinTask
join()
fork()
fork()
join()
Child ForkJoinTask
Child ForkJoinTask

```
public class Sum extends RecursiveTask<Long> {
   private static final int SEQUENTIAL_THRESHOLD = 5000;
   private final int[] array;
   private final int begin, end;

   public Sum(int[] array, int begin, int end) {
      this.array = array; this.begin = begin; this.end = end;
   }

   @Override public Long compute() {
      if (end - begin <= SEQUENTIAL_THRESHOLD) {
         long sum = 0;
         for (int i = begin; i < end; ++i) sum += array[i];
         return sum;
      } else {
         int mid = begin + (end - begin) / 2;
         Sum left = new Sum(array, begin, mid);
         Sum right = new Sum(array, mid, end);
         left.fork();
         long rightResult = right.compute();
         long leftResult = left.join();
         return leftResult + rightResult;
      }
   }
}
```

```
ForkJoinPool pool = new ForkJoinPool();
long result = pool.invoke(new Sum(array, 0, array.length));
```