

# Turbo Charge CPU Utilization in Fork/Join Using the ManagedBlocker

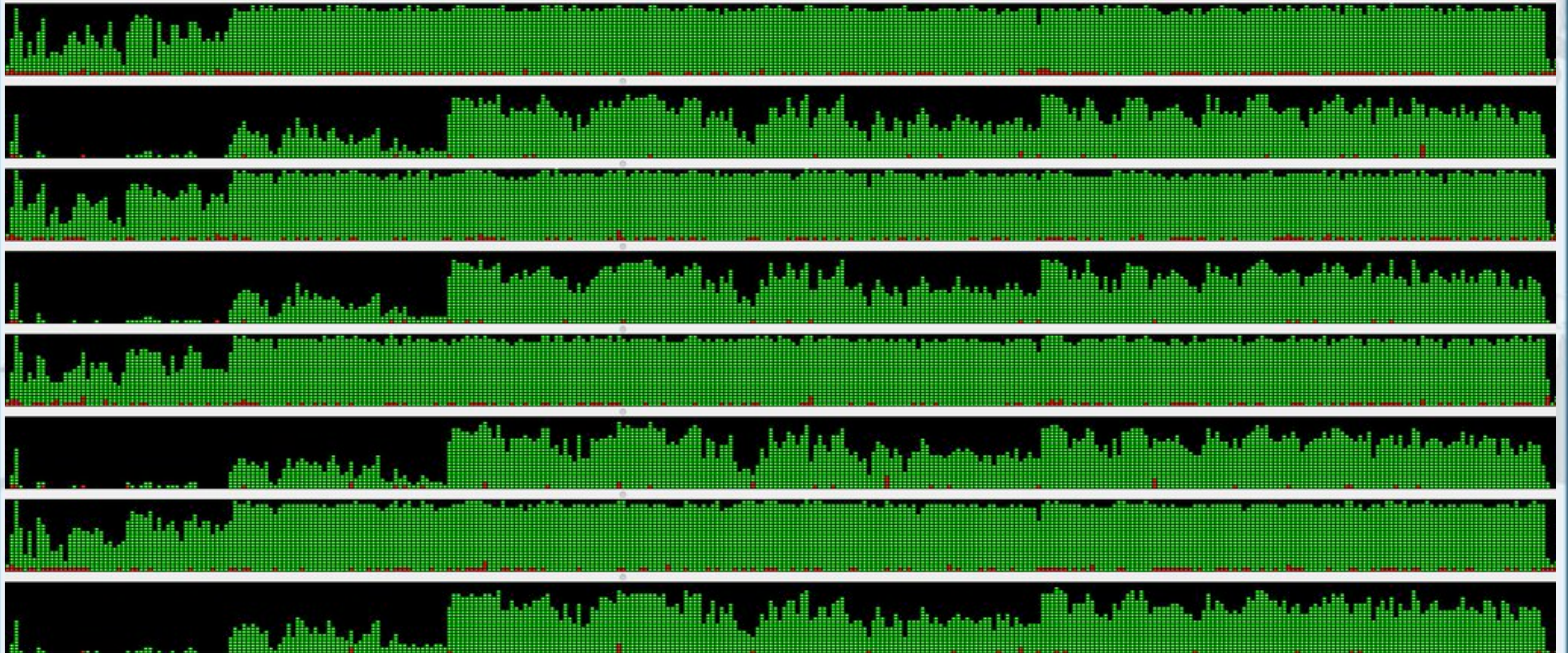
**Dr Heinz M. Kabutz**

Last Updated 2017-11-13

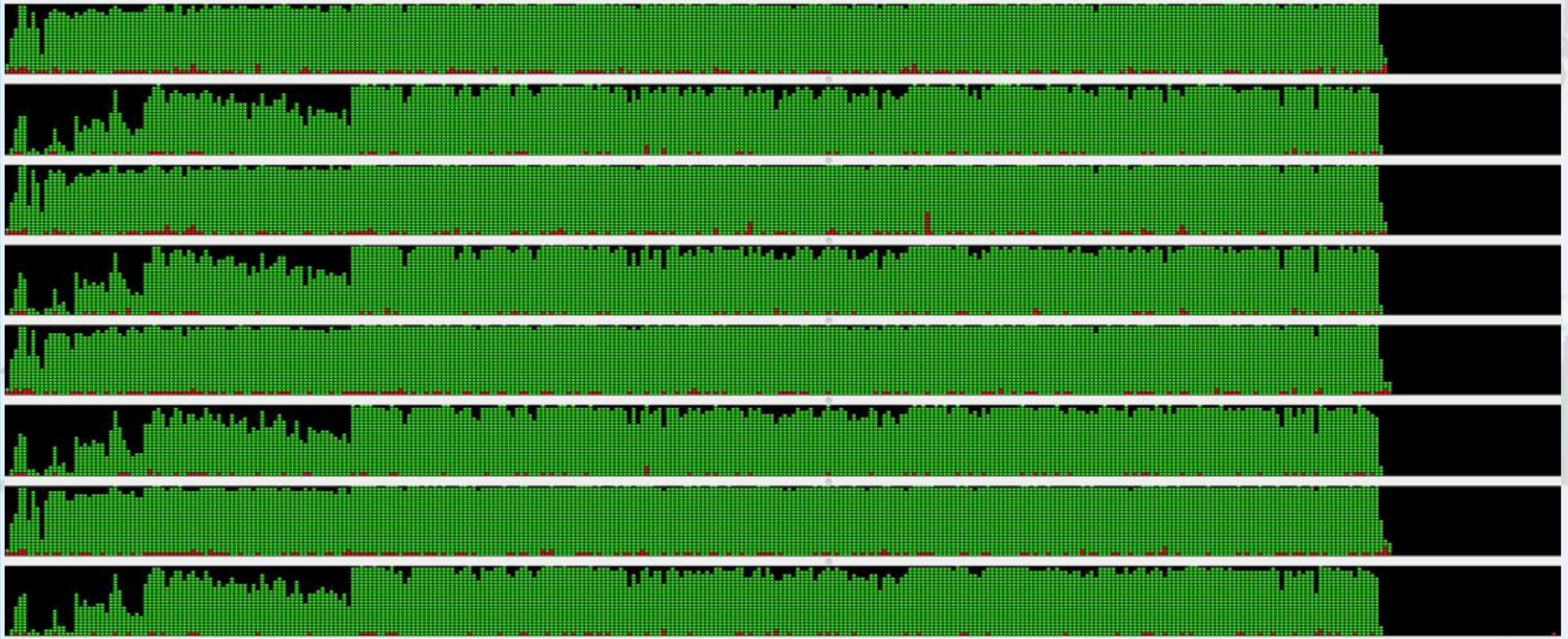


**Javaspecialists.eu**  
java training

# Regular



# ManagedBlocker



# Speeding Up Fibonacci

- **By Leonardo of Pisa**

- $F_0 = 0$

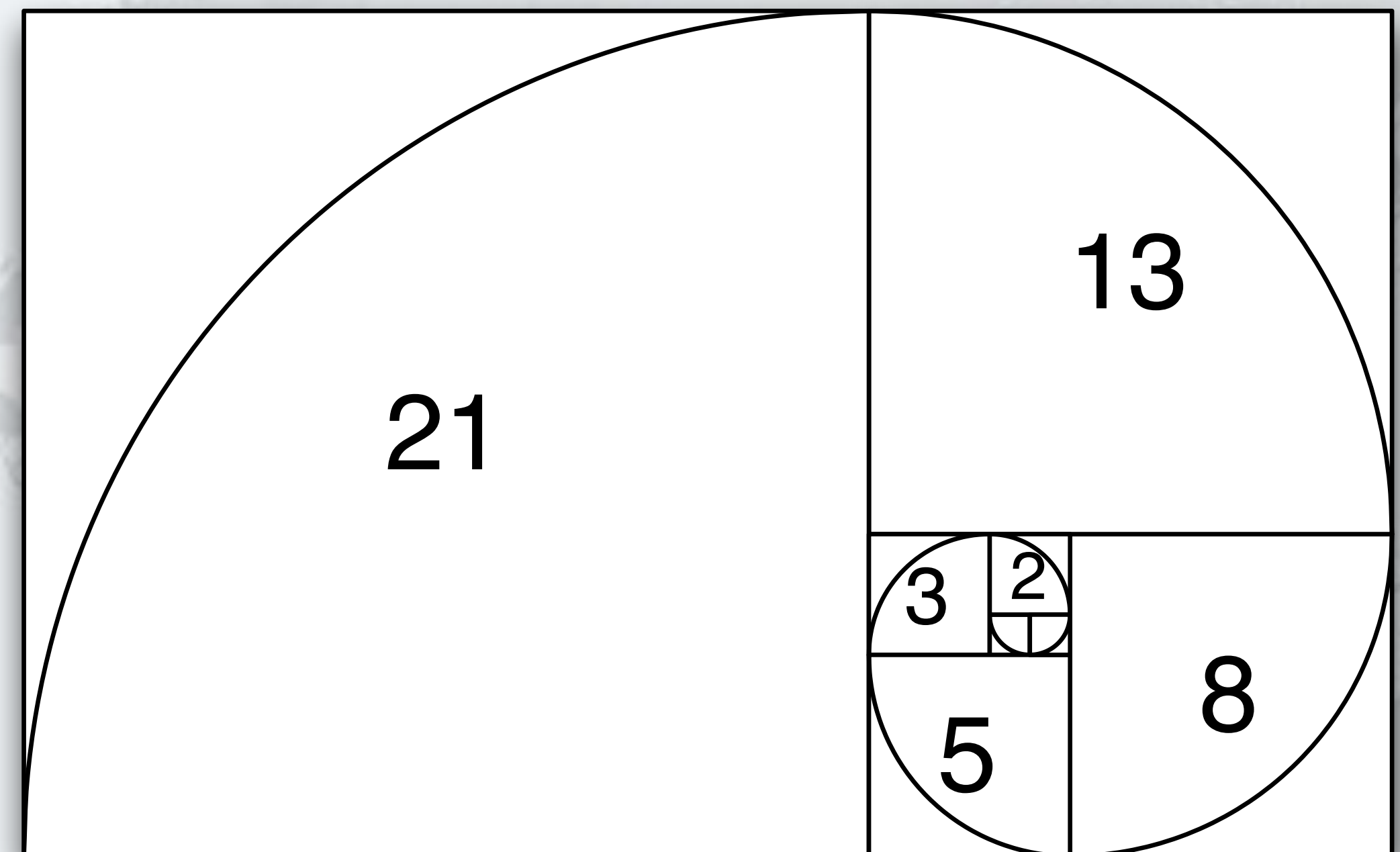
- $F_1 = 1$

- $F_n = F_{n-1} + F_{n-2}$

- **Thus the next number is equal to the sum of the two previous numbers**

- e.g. 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- The numbers get large quickly, like Australian rabbit population



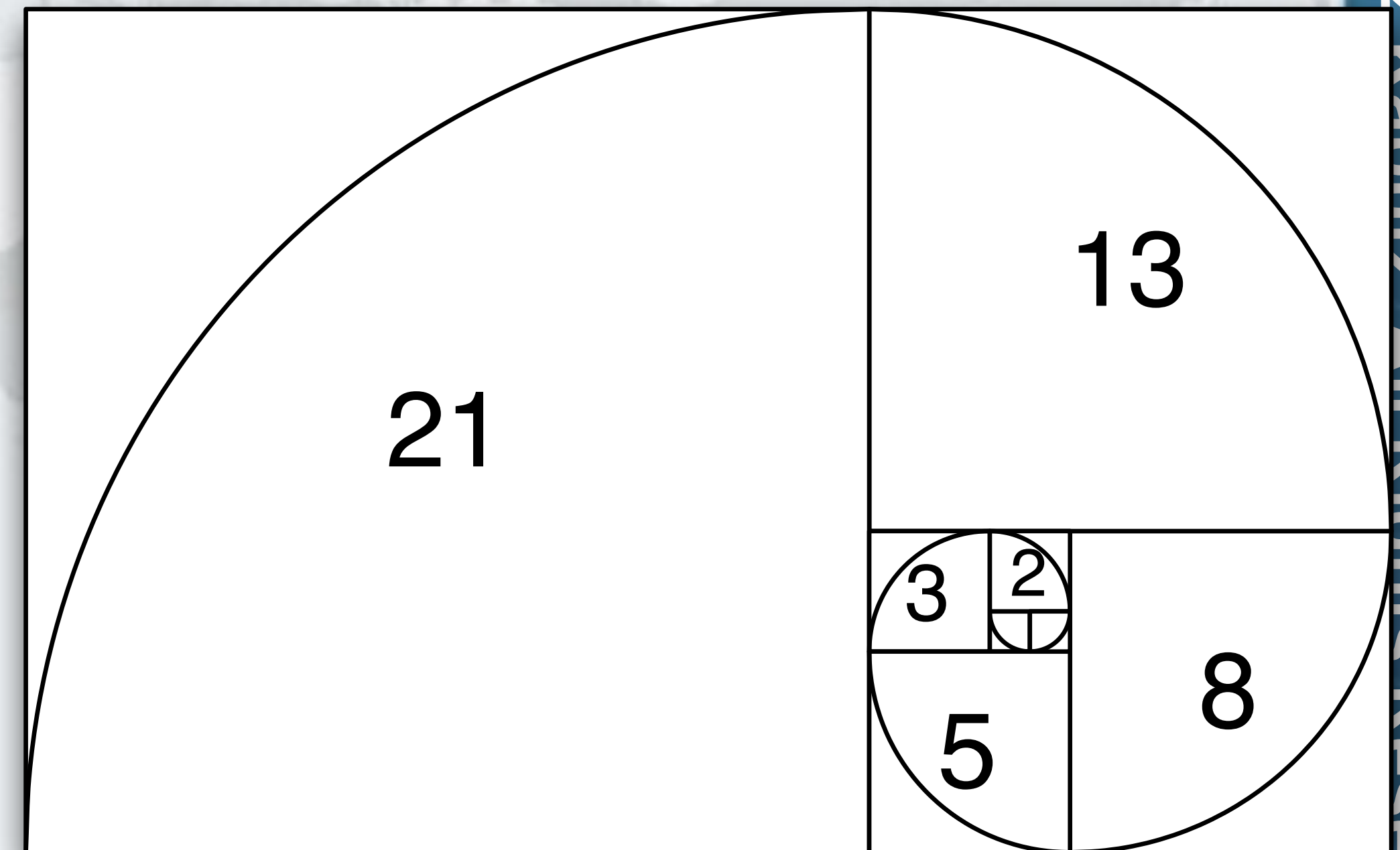
# Naive Implementation

- Taking our recursive definition

- $F_0 = 0, F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

- Converting this into Java:

```
public long f(int n) {  
    if (n <= 1) return n;  
    return f(n-1) + f(n-2);  
}
```



- But this has exponential time complexity, so gets terribly slow

# 2nd Attempt at Coding Fibonacci

- **Iterative algorithm**

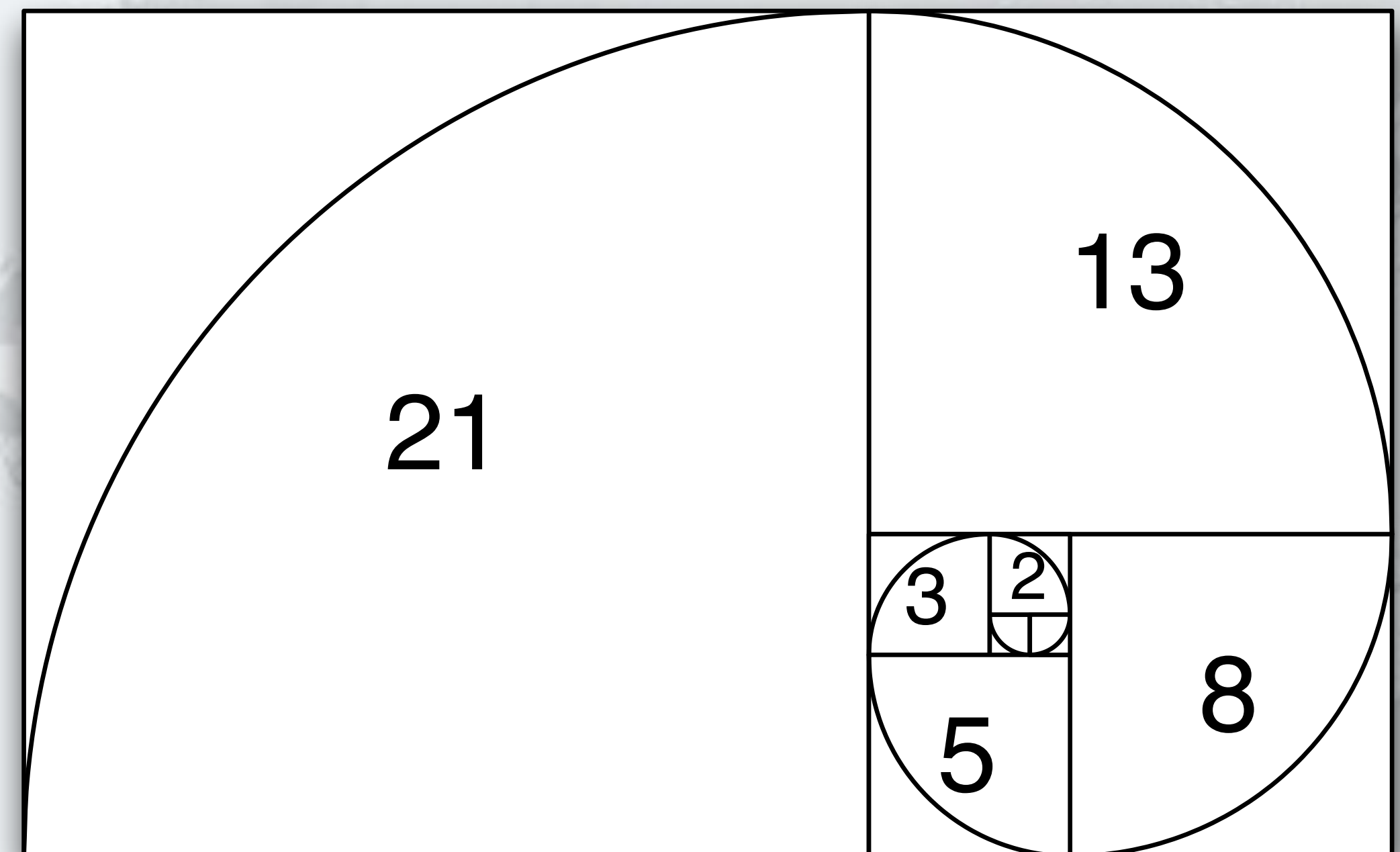
```
public static long f(int n) {  
    long n0 = 0, n1 = 1;  
    for (int i = 0; i < n; i++) {  
        long temp = n1;  
        n1 = n1 + n0;  
        n0 = temp;  
    }  
    return n0;  
}
```

- **Linear time complexity**

- **f(1\_000\_000\_000) in 1.7 seconds**

- However, long overflows when n=93, so the result is incorrect

- We can use BigInteger, but its add() is also linear, so time is quadratic



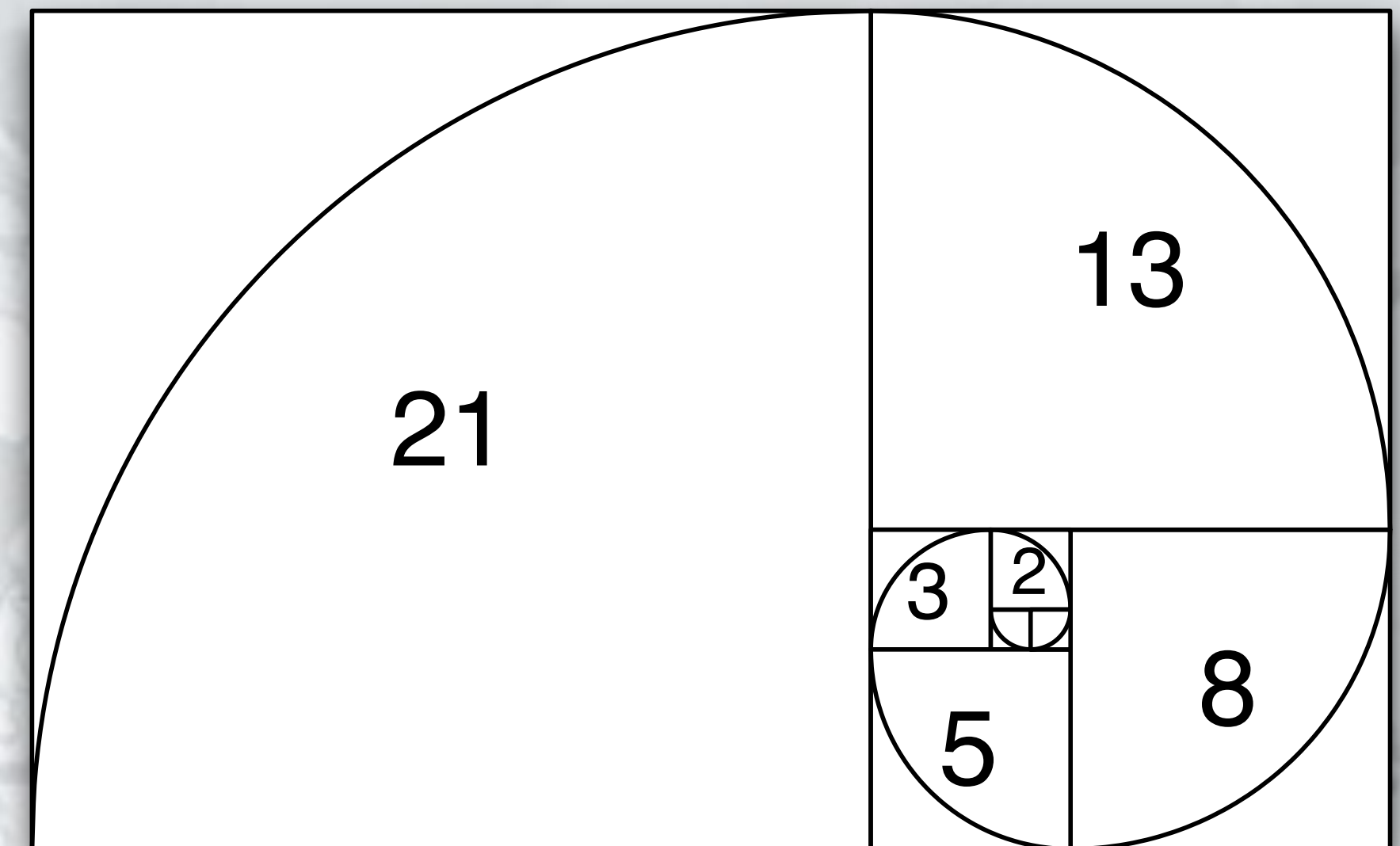
## 3<sup>rd</sup> Attempt Dijkstra's Sum of Squares

- **Dijkstra's clever formula**

- $F_{2n-1} = F_{n-1}^2 + F_n^2$
- $F_{2n} = (2 \times F_{n-1} + F_n) \times F_n$

- **Logarithmic time complexity**

- Multiply in BigInteger used to be  $O(n^2)$ 
  - Karatsuba complexity is  $O(n^{1.585})$
  - 3-way Toom Cook complexity is  $O(n^{1.465})$
- `BigInteger.multiply()` single-threaded in Java - we'll fix that later



# Demo 1: Dijkstra's Sum of Squares

- **We implement this algorithm using BigInteger**
  - $F_{2n-1} = F_{n-1}^2 + F_n^2$
  - $F_{2n} = (2 \times F_{n-1} + F_n) \times F_n$



## Demo 2: Parallelize Our Algorithm

- **We can parallelize by using common Fork/Join Pool**
  - Next we fork() the 1<sup>st</sup> task, do the 2<sup>nd</sup> and then join 1<sup>st</sup>

```
RecursiveTask<BigInteger> f0_task = new RecursiveTask<BigInteger>() {  
    protected BigInteger compute() {  
        return f(half - 1);  
    }  
};  
f0_task.fork();  
BigInteger f1 = f(half);  
BigInteger f0 = f0_task.join();
```

# Expires End Of This Talk

- **Free license to "Refactoring to Java 8 Streams and Lambdas Online Self- Study Workshop" (Price €156)**
- **Free subscription to The Java Specialists Newsletter**
- **<http://tinyurl.com/poznan17>**



<http://tinyurl.com/poznan17>

## Demo 3: Parallelize BigInteger

- **Let's hack fork/join into:**
  - `multiplyToomCook3()`
  - `squareToomCook3()`
- **These probably won't reach the threshold**
  - `multiplyKaratsuba()`
  - `squareKaratsuba()`
- **Choose modified BigInteger with**
  - `-Xbootclasspath/p:<path_to_hack>`
  - Java 9 a bit more complicated - create a patch for module

<http://tinyurl.com/poznan17>

**Level 0**

1 000 000

**Level 1**

499 999

500 000

**Level 2**

249 999

250 000

249 999

250 000

**Level 5**

16 x 31 249

16 x 31 250

**Level 6**

32 x 15 624

32 x 15 625

**Level 7**

32 x 7 811

64 x 7 812

32 x 7 813

**Level 14**

7 904 x 60

8 192 x 61

288 x 62

<http://tinyurl.com/poznan17>

## Demo 4: Cache Results

- **Dijkstra's Sum of Squares needs to work out some values several times. Cache results to avoid this.**
  - **Careful to avoid a memory leak**
    - **No static maps**

<http://tinyurl.com/poznan17>

## Demo 5: Reserved Caching Scheme

- **Instead of calculating same value twice:**
  - Use `putIfAbsent()` to insert special placeholder
  - If result is null, we are first and start work
  - If result is the placeholder, we wait

<http://tinyurl.com/poznan17>

## Demo 6: ManagedBlocker

- **ForkJoinPool is configured with *desired parallelism***
  - Number of active threads
  - ForkJoinPool mostly used with CPU intensive tasks
- **If one of the FJ Threads has to block, a new thread can be started to take its place**
  - This is done with the ManagedBlocker
- **We use ManagedBlocker to keep parallelism high**

<http://tinyurl.com/poznan17>

## Demo 7: CompletableFuture (Homework)

- **Implement Fibonacci using**
  - **CompletableFuture with methods**
    - **thenAcceptBothAsync()**
    - **complete()**
  - **What happens re thread creation when we disable common pool?**
    - **-Djava.util.concurrent.ForkJoinPool.common.parallelism=0**
- **Send your answers to [heinz@javaspecialists.eu](mailto:heinz@javaspecialists.eu)**



## Last chance ...

- **Free license to "Refactoring to Java 8 Streams and Lambdas Online Self- Study Workshop" (Price €156)**
- **Free subscription to The Java Specialists Newsletter**
- **<http://tinyurl.com/poznan17>**



# Turbo Charge CPU Utilization in Fork/Join Using the ManagedBlocker

**Dr Heinz M. Kabutz**

Last Updated 2017-11-13



**Javaspecialists.eu**  
java training