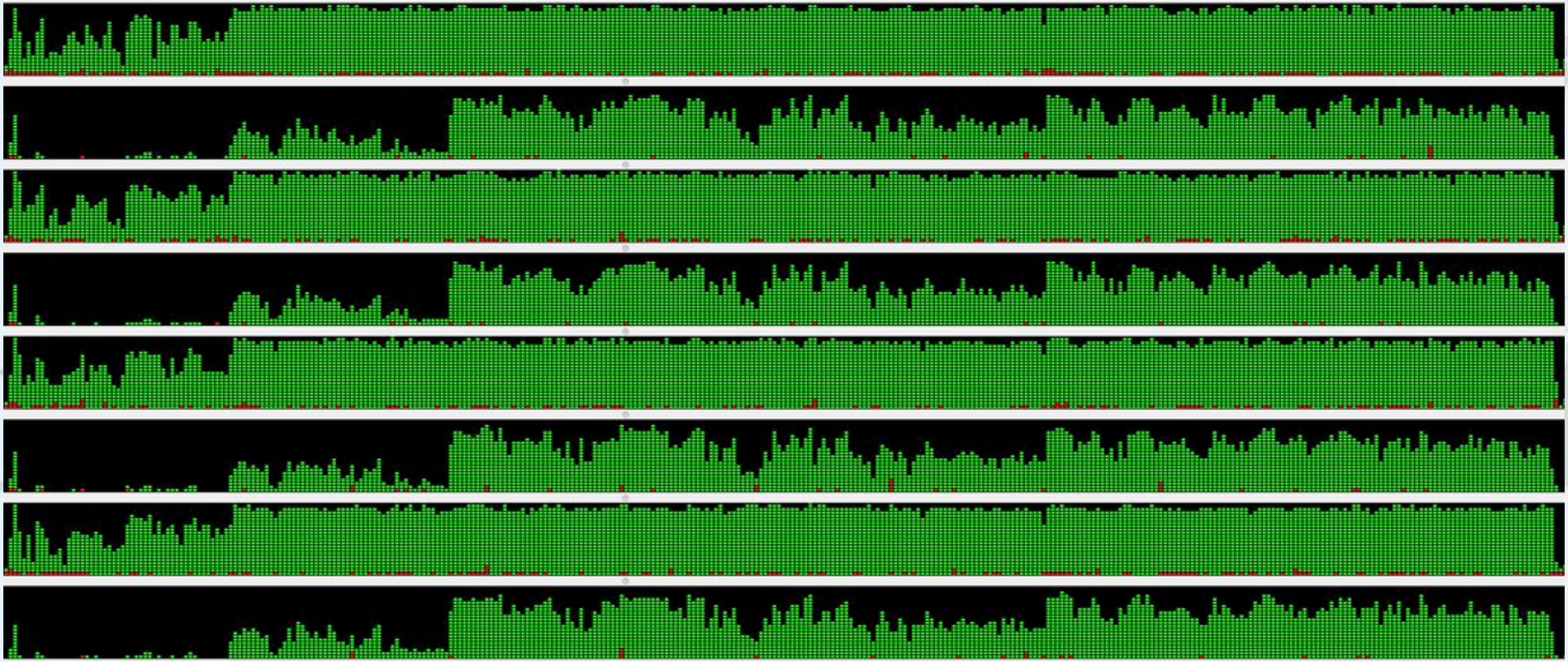
Turbo Charge CPU Utilization in Fork/Join Using the ManagedBlocker

Dr Heinz M. Kabutz

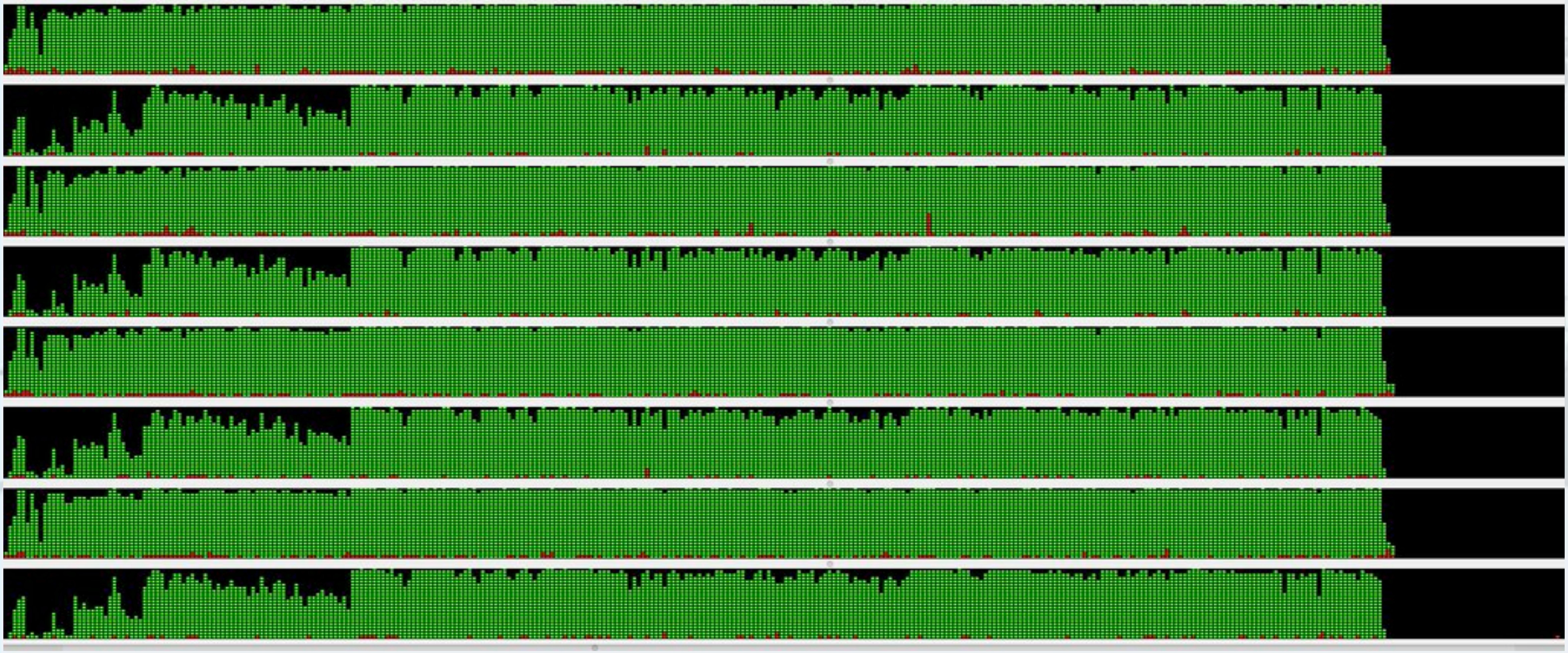
Last Updated 2018-01-26

Javaspecialists.eu
java training

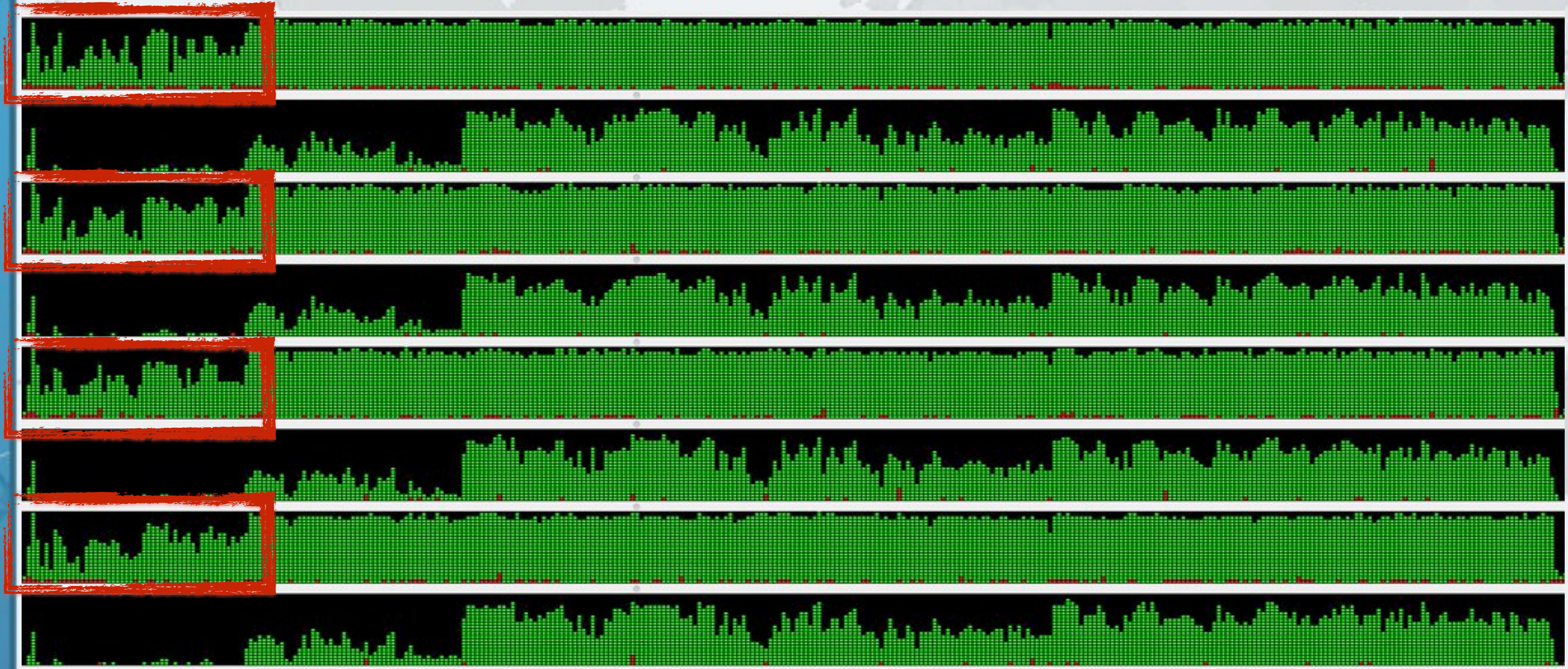
Regular Fibonacci 1.000.000.000



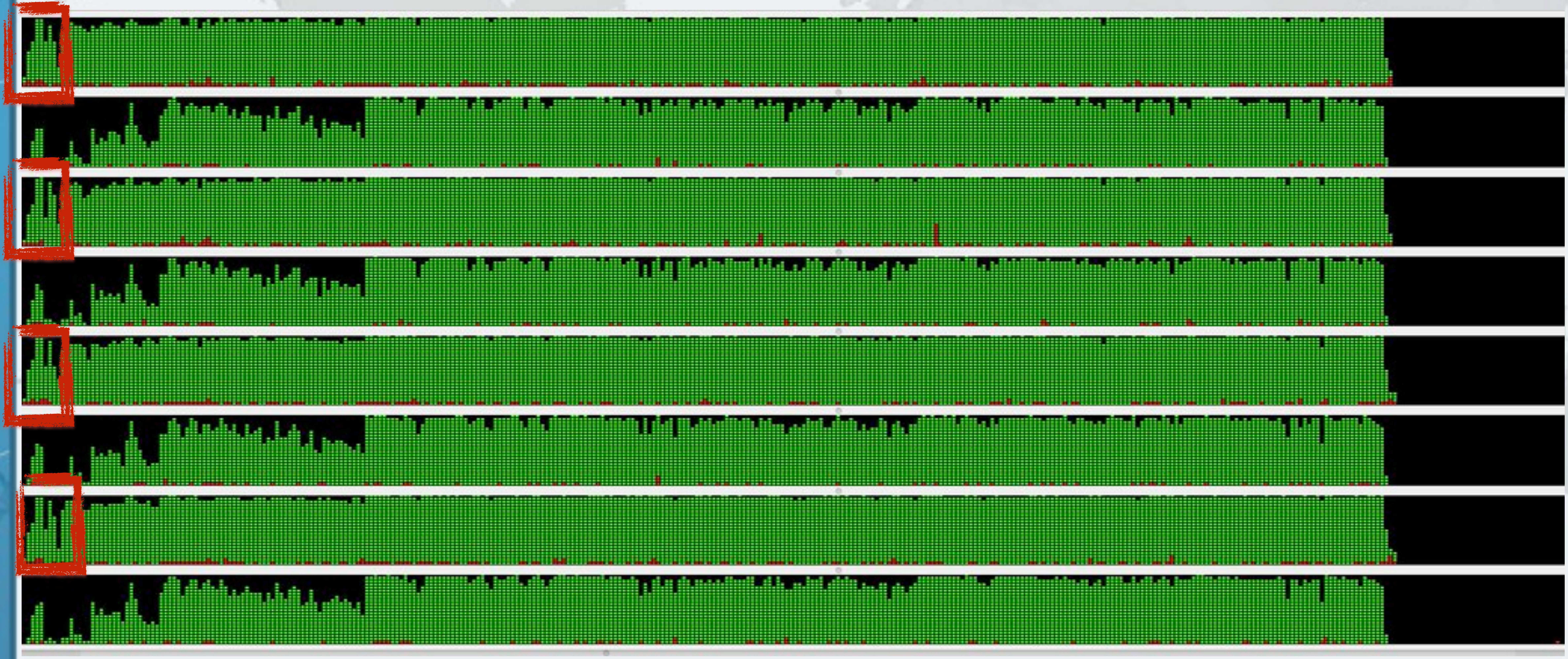
ManagedBlocker Fibonacci 1.000.000.000



Initial Startup Regular Fib 1 billion

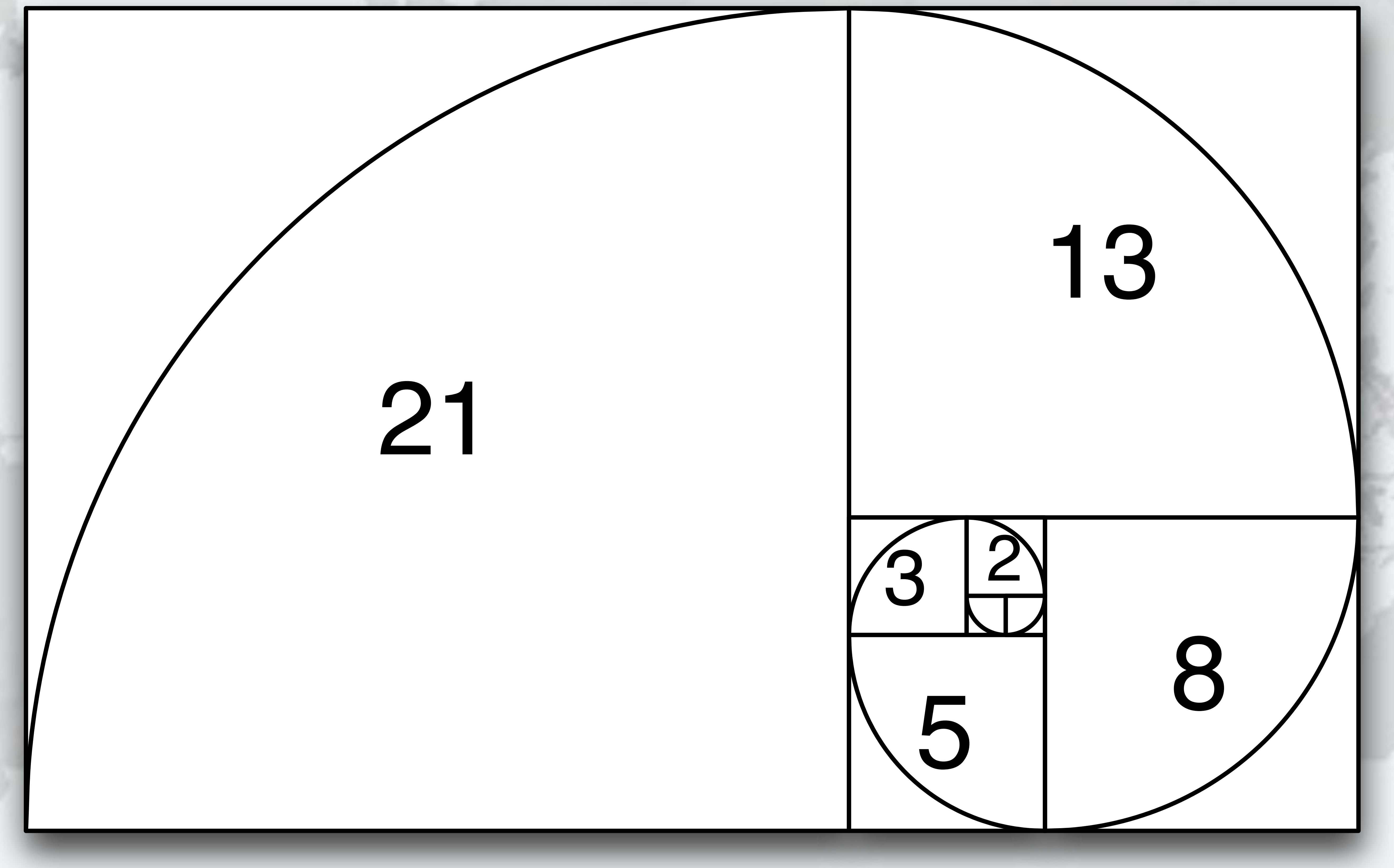


Better Utilization In the Beginning



Speeding Up Fibonacci

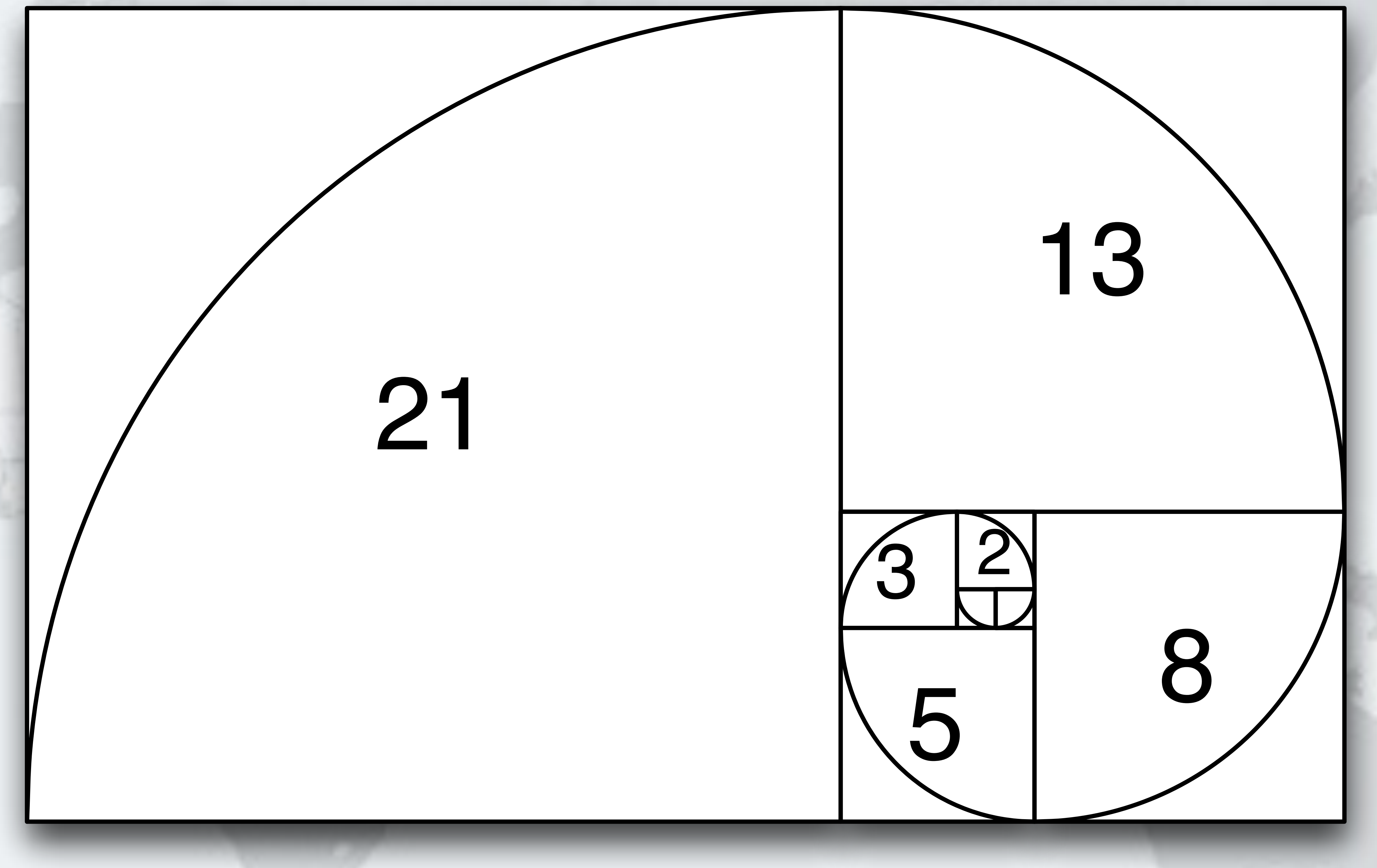
- **By Leonardo of Pisa**
  - $F_0 = 0$
  - $F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$



Naive Implementation

- **Taking our recursive definition**
  - $F_0 = 0, F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$
- **Converting naïvely into Java:**

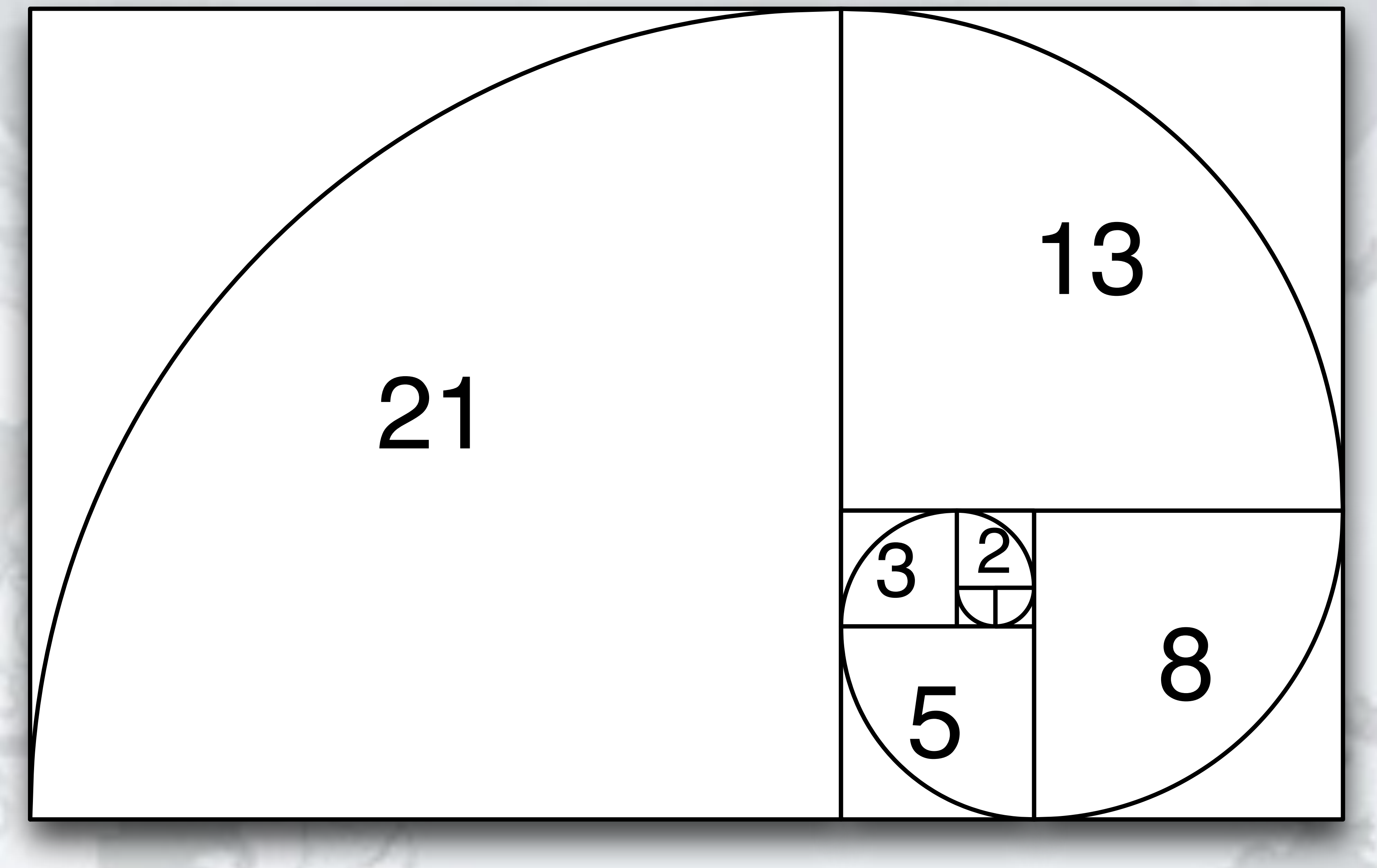
```
public long f(int n) {
  if (n <= 1) return n;
  return f(n-1) + f(n-2);
}
```



- **Computational Time Complexity is itself a fibonacci series**

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 Java BigInteger**
    - **Karatsuba complexity is $O(n^{1.585})$**
    - **3-way Toom Cook complexity is $O(n^{1.465})$**
    - **Prior to Java 8, multiply() had complexity $O(n^2)$**
    - **BigInteger.multiply() single-threaded in Java - we’ll fix that later**



Demo 1: Dijkstra’s Sum of Squares

- **Let's write this in Java with 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 1st task, do the 2nd and then join 1st**

```
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();
```

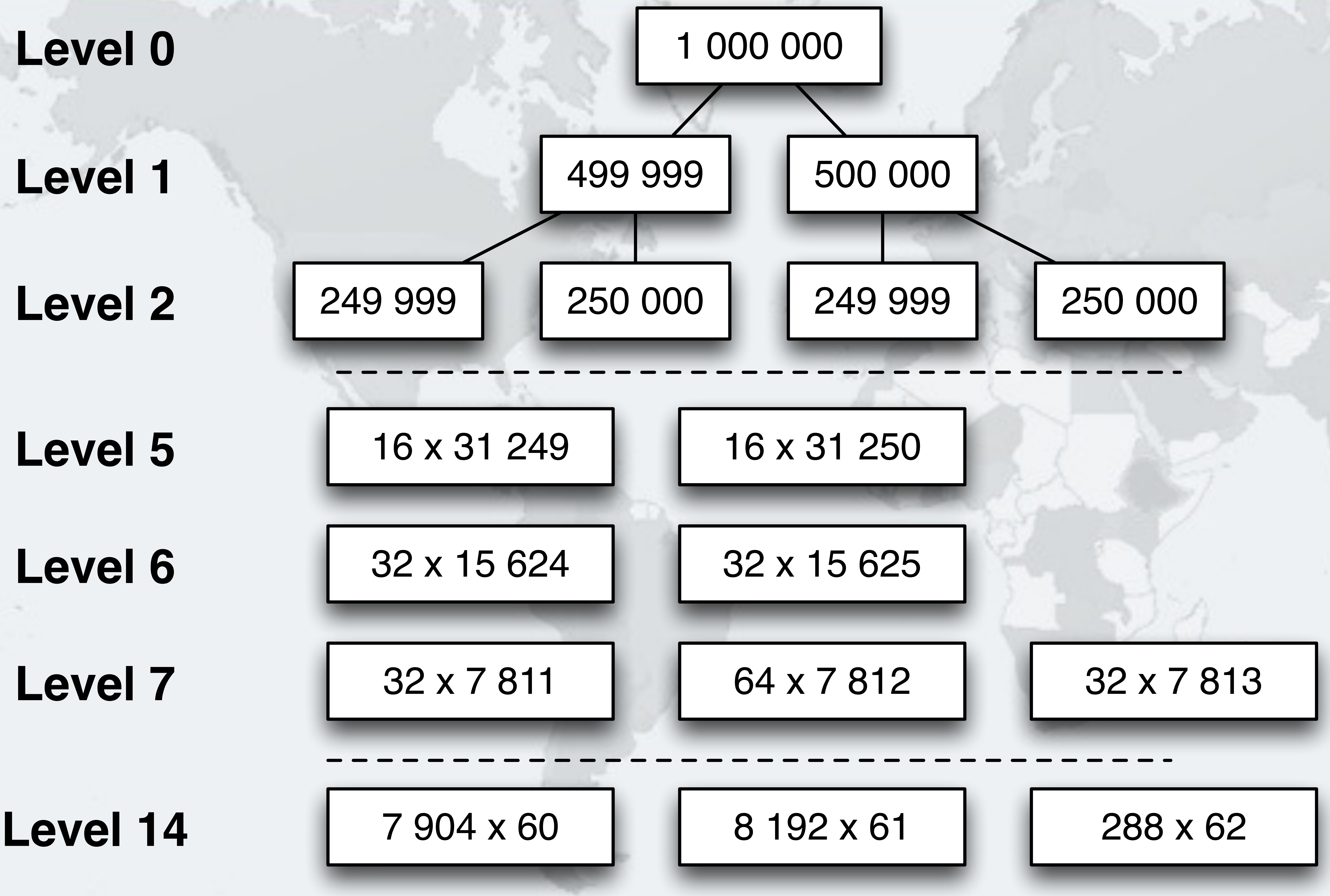
Data Structures in Java 9 Self-Study

- **95% discount until 9pm this evening**
  - **Also gets you onto The Java Specialists' Newsletter list :-)**
- **https://tinyurl.com/skg18**

Demo 3: Parallelize BigInteger

- **Let’s hack fork/join into:**
  - **multiplyToomCook3()**
  - **squareToomCook3()**
- **Choose modified BigInteger with**
  - **-Xbootclasspath/p:<path_to_hack>**
  - **Java 9 a bit more complicated - create a patch for module**

Turbo Charge CPU Utilization in Fork/Join Using the ManagedBlocker



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

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**

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**

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**

Demo 7: CompletableFuture (Homework)

- **Implement Fibonacci using**
  - **CompletableFuture with methods**
    - **thenAcceptBothAsync()**
    - **complete()**
  - **What happens with thread creation with no common ForkJoinPool?**
    - **-Djava.util.concurrent.ForkJoinPool.common.parallelism=0**
- **Send your answers to heinz@javaspecialists.eu**

Data Structures in Java 9 Self-Study

- **95% discount until 9pm this evening**
  - **Also automatically get The Java Specialists' Newsletter**
- **https://tinyurl.com/skg18**
- **And time for questions?**