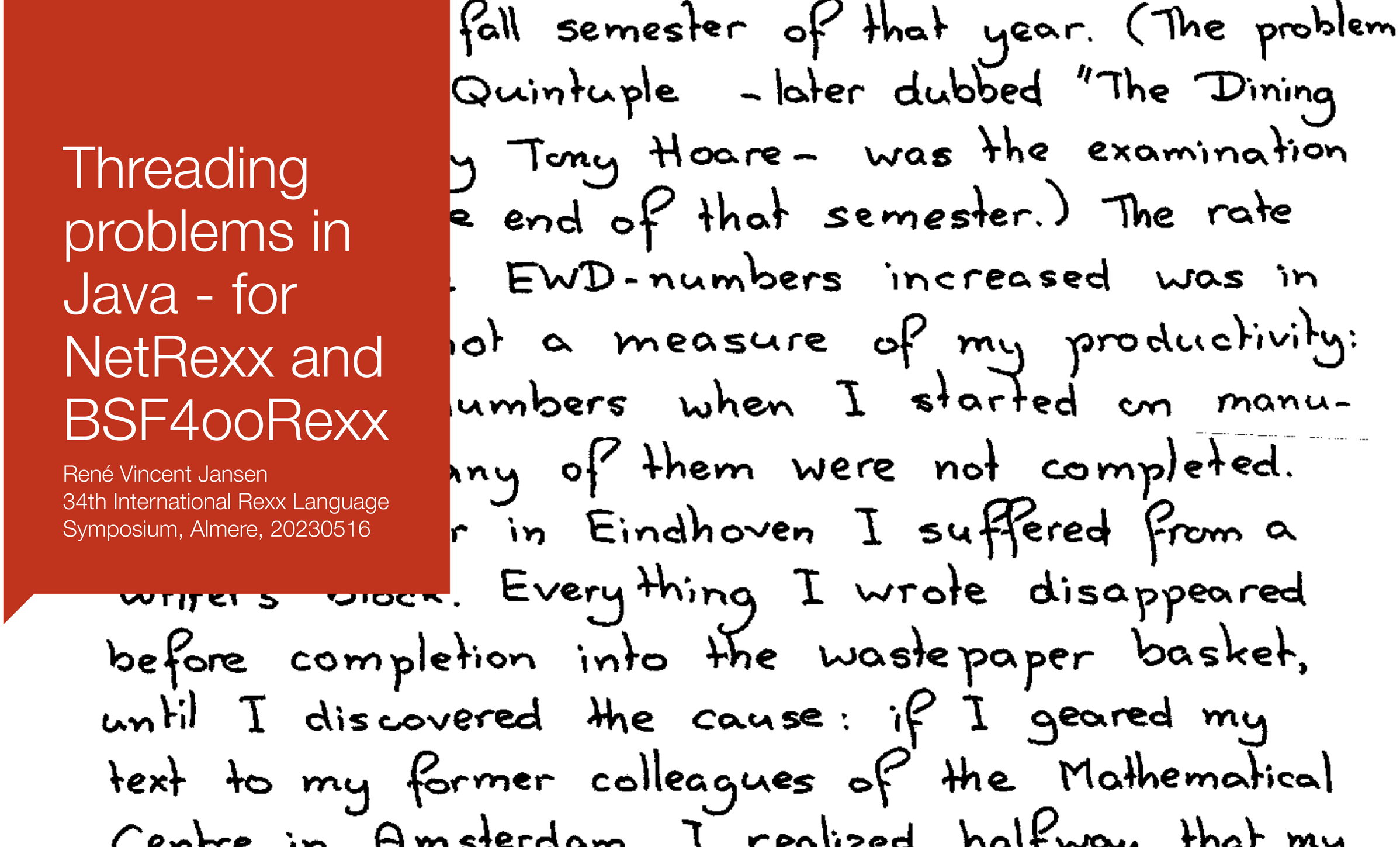
Threading problems in Java - for NetRexx and BSF4ooRexx

René Vincent Jansen

34th International Rexx Language Symposium, Almere, 20230516



Contents

№ 1

Tiered Compilation

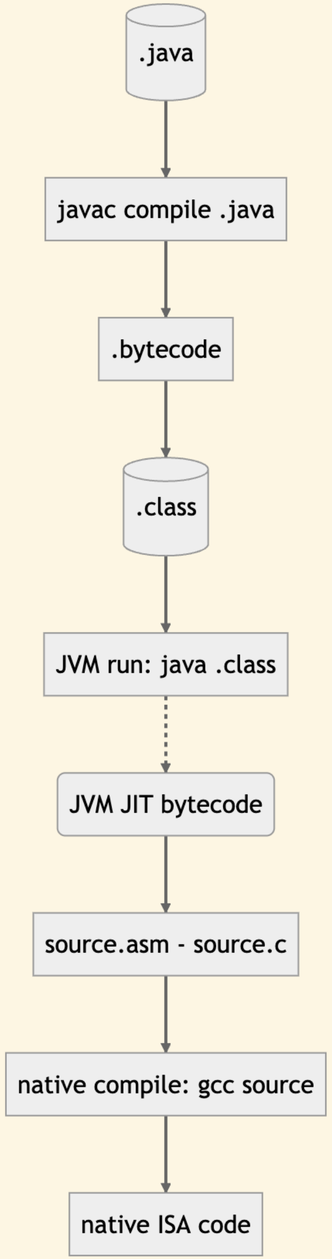
Source, JVM ByteCode, Native Code, Hardware Instruction Set Architectures

Tiered Compilation

Java : compile --> bytecode,
run bytecode in VM
compile --> native

Code Flow

only two files involved: java and class, but many transitions



```java
class hello {
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Hello World

Looking around

- Compile it: javac hello.java
- see the bytecode in the classfile; plain and hex
- disassemble the bytecode with javap -c hello.class
- disassemble the bytecode with gnoloo hello.class
  - look at the generated hello.j
- execute it with the JVM: java hello

See the native compilation

java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly hello

with hex: cafebabe

with hsdis-aarch64.dylib

(download from https://chriswhocodes.com/hsdis/ or build from openJDK source)

№ 2

Race Conditions

Where Threading makes a mess of your memory

```
OPTIONS BINARY
CLASS RaceCondition

PROPERTIES STATIC
sharedVariable = 0
lock=Object()

  METHOD main(args=String[]) STATIC
    thread1 = Thread(Increment())
    thread2 = Thread(Increment())

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    SAY "shared variable = "sharedVariable

CLASS Increment IMPLEMENTS Runnable
  METHOD run()
    LOOP PROTECT RaceCondition.lock FOR 100000
      RaceCondition.sharedVariable = RaceCondition.sharedVariable +
    END
```

Two Threads

Threading can cause race conditions

- We start two threads that sum to
- Compile and Run, see that the answer is not the same two times in a row
- How to solve this probleme
  - Monitors
  - Critical Sections
- We see the monitor construct in the JVM Bytecode

№ 3

Thread Starvation

or Dining Philosophers

The Edsger Dijkstra Problem

```
BJECT[NUMPHILOSOPHERS];
OSOPHERS; I++) {
ECT();

= NEW PHILOSOPHER[NUMPHILOSOPHERS];
OSOPHERS; I++) {
HILOSOPHER(I, CHOPSTICKS[I], CHOPSTICKS[(I + 1) % NUMPHILOSOPHERS]);
);

}
CLAS

CT LEFTCHOPSTICK, OBJECT RIGHTCHOPSTICK) {

OPSTICK;
CHOPSTICK;

        THINK();
        PICKUPCHOPSTICKS();
        EAT();
```

№ 4

Deadlocks

Like database locks but more fierce and hard to diagnose

```
CLASS Deadlock
PROPERTIES CONSTANT
lock1 = Object()
lock2 = Object()

  METHOD main(args=String[]) STATIC
    thread1 = Thread(Task1())
    thread2 = Thread(Task2())
    thread1.start()
    thread2.start()
    thread1.join();
    thread2.join();

CLASS Task1 IMPLEMENTS Runnable
  METHOD run()
    DO PROTECT Deadlock.lock1
      SAY "Task1 acquired lock1"
      SAY "Task1 waiting for lock2"
      DO PROTECT Deadlock.lock2
        SAY "Task1 acquired lock2"
      END
    END

CLASS Task2 IMPLEMENTS Runnable
  METHOD run()
    DO PROTECT Deadlock.lock2
      SAY "Task2 acquired lock2"
      SAY "Task2 waiting for lock1"
      DO PROTECT Deadlock.lock1
        SAY "Task2 acquired lock1"
      END
    END
```

A clear case of Deadlock

The sequence of events counts

Deadlock breaking

- run with a profiler
  - java -XX:HeapDumpPath=/Users/rvjansen/test/javathreading -javaagent:/Users/rvjansen/apps/glowroot/glowroot.jar Deadlock
- look at JVM - thread dump
- when unclear: actually dump the JVM memory image
- format the dump
  - jhat heap-dump-20230420-125101.hprof <== subst actual name