Troubleshooting Slowdowns, Freezes, Deadlocks Introduction to Thread Dump

[BOF1518]

Yusuke Yamamoto  @yusuke

Yusuke Yamamoto @yusuke

- Author of Samurai
- Project lead of Twitter4J
- 10+ years at Java industry

2000

2002

2006

2008

2011

2013

BOF1518

#BOF1518

Quiz

#BOF1518

#BOF1518

A.

later...

#BOF1518

Troubleshooting

Troubles

#BOF1518

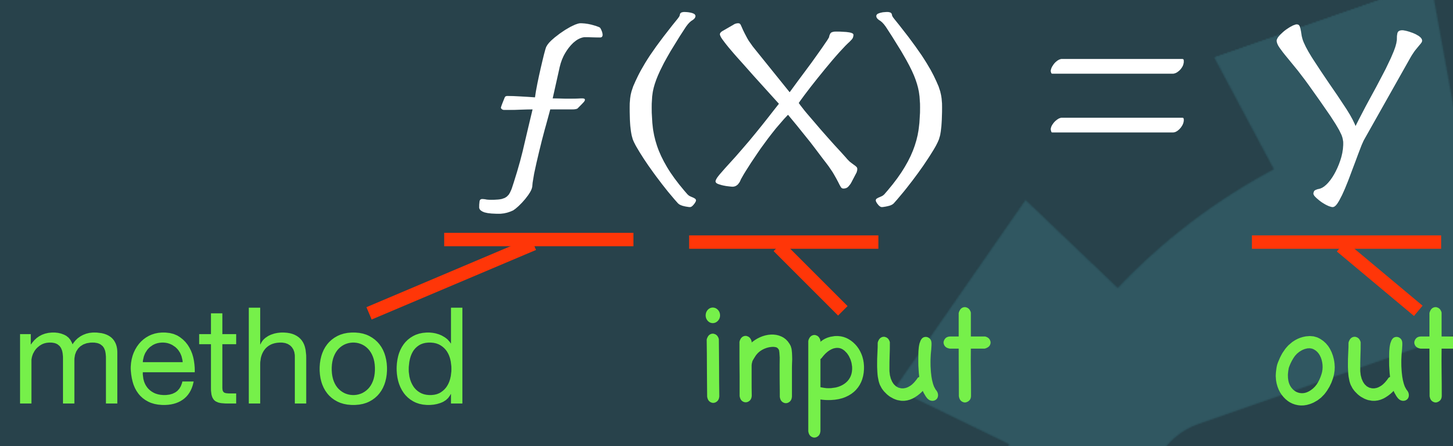
1. Unexpected output
2. Exception
3. Slowdown
4. Freeze
5. Deadlock

Troubles

#BOF1518

1. Unexpected output
2. Exception
3. Slowdown
4. Freeze
5. Deadlock

#BOF1518




```
assertThat(f(X).is(Y))
```

JUnit

#BOF1518

```
assertThat(f(X)).equals(Y)
```

JUnit

Troubles

#BOF1518

1. Unexpected output
2. Exception
3. Slowdown
4. Freeze
5. Deadlock

Troubles

#BOF1518

1. Unexpected output
2. Exception.printStacktrace()
3. Slowdown
4. Freeze
5. Deadlock

Understanding exception stacktrace

Thread name Exception Type

```
Exception in thread "main" java.lang. IllegalStateException:
    Missing Authentication credentials
        at t4j.BaseImpl.ensureAuthorizationEnabled(BaseImpl.java:215)
        at t4j.TwitterImpl.get(TwitterImpl.java:1784)
        at t4j.TwitterImpl.getHomeTimeline(TwitterImpl.java:105)
        at HelloTwitter4J.main(HelloTwitter4J.java:6)
```

Message

Call stack

JUnit

```
@Test(expected=NullPointerException)
```

Troubles

#BOF1518

1. Unexpected output
2. Exception
3. Slowdown
4. Freeze
5. Deadlock

Troubles

1. Unexpected output
2. Exception

single-threaded issues

3. Slowdown
4. Freeze
5. Deadlock

Multi-threaded issues

#BOF1518

Dig into multi-threaded issues

Thread dump

noun /θred dʌmp/

A snapshot of every thread in the JVM at a particular point in time.

#BOF1518

demo /démoʊ/

Taking a thread dump

$ `kill -3` *`PID`* (Linux / Unix / Mac)

Ctrl + Break (Windows)

thread dump goes to the process' standard output

traditional, not suggested today

Taking a thread dump [2015]

```
$ jstack [PID]
```

thread dump comes to the console

check the JVM's process ID

```
$ ps -ef | grep java
```

or

```
$ jps
```

#BOF1518

demo /démoʊ/

jps options

`-l` : show FQCN

`-m` : show arguments

`-v` : show JVM options

Try `-l` first, then `-mlv`

```
$ jps -l

$ jps -lmv
```

Taking thread dump

Small impact: < 300 ms

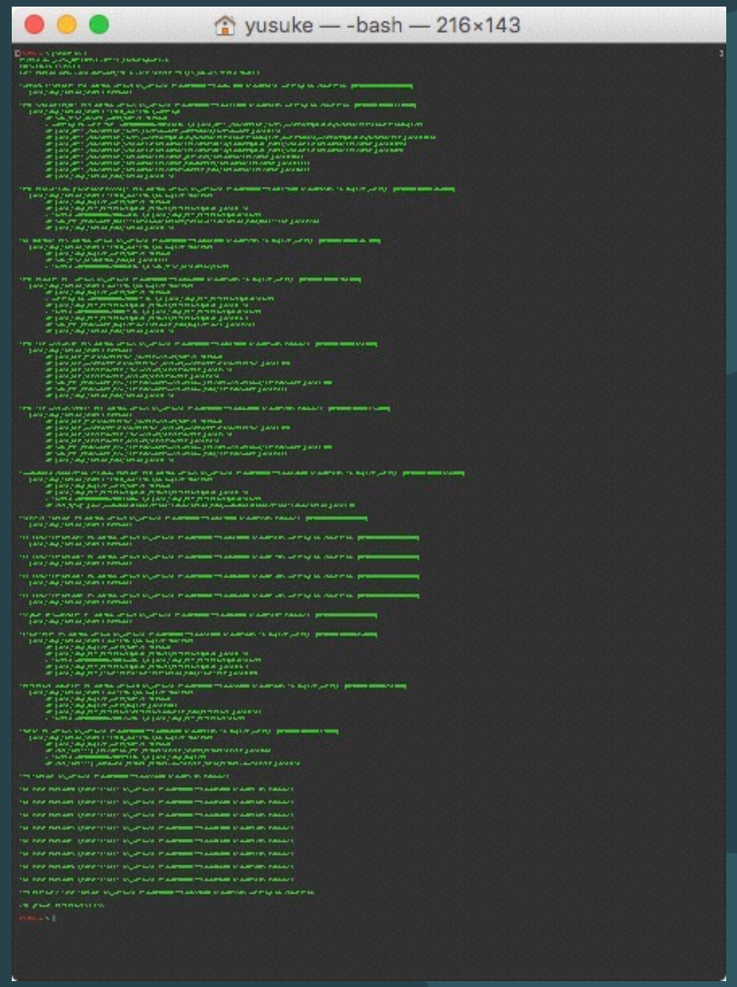
No modification to your application required

No need to attach debuggers

Applicable to production environments

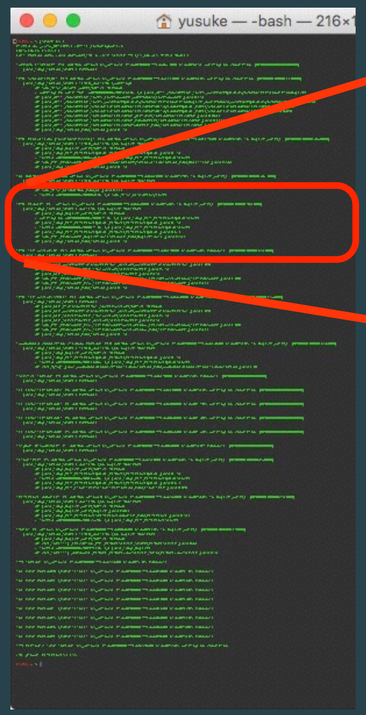
Thread dump

#BOF1518



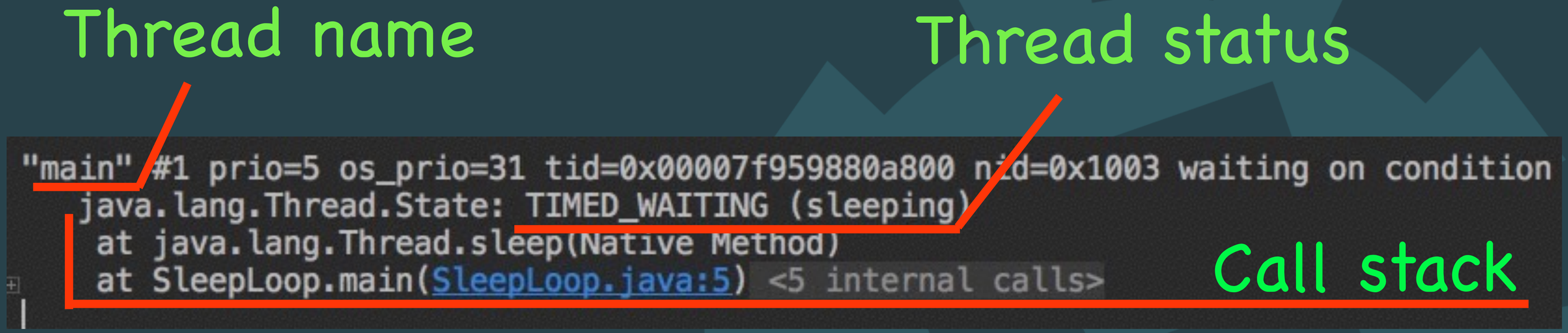
Thread dump

#BOF1518




```
"RMI Scheduler(0)" #19 daemon prio=5 os_prio=31 tid=0x00007ff1ab12e000 nid=0x6703 waiting on condition [0x0000700001e70000]
   java.lang.Thread.State: TIMED_WAITING (parking)
        at sun.misc.Unsafe.park(Native Method)
        - parking to wait for  <0x00000006c0051870> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
        at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:2078)
        at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:1093)
        at java.util.concurrent.ScheduledThreadPoolExecutor$DelayedWorkQueue.take(ScheduledThreadPoolExecutor.java:809)
        at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
```

Stack trace in thread dump



Thread status

RUNNABLE : Running as expected

```
"RMI TCP Connection(3)-127.0.0.1" #19 daemon prio=5 os_prio=31 tid=0x00007ffe5417b800
nid=0x6307 runnable [0x00007000019de000]
   java.lang.Thread.State: RUNNABLE
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    - locked <0x000000079600b878> (a java.net.SocksSocketImpl)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:589)
    at com.mysql.jdbc.StandardSocketFactory.connect(StandardSocketFactory.java:213)
    at com.mysql.jdbc.MysqlIO.<init>(MysqlIO.java:297)
```

Thread status

TIMED_WAITING / WAITING: Idle thread

```
"main" #1 prio=5 os_prio=31 tid=XXXXXXX nid=0x1003 waiting on condition
 java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(Native Method)
  at samurai.core.IdleExample.main(IdleExample.java:20)
```

```
"main" #1 prio=5 os_prio=31 tid=XXXXXXX nid=0x1003 waiting on condition
 java.lang.Thread.State: WAITING (on object monitor)
  at java.lang.Object.wait(Native Method)
  at samurai.core.IdleExample.main(IdleExample.java:20)
```

Thread status

BLOCKED : Trying to acquire a lock

```
"Thread-1" #11 prio=5 os_prio=31 tid=0x00007ffd4409d800 nid=0x5003
waiting for monitor entry [0x00007000012cc000]
   java.lang.Thread.State: BLOCKED (on object monitor)
  at samurai.core.AThread.run(BlockerExample.java:38)
  - waiting to lock <0x000000079579aa18> (a [Ljava.lang.String;)
```

Grabbing the lock:

```
"Thread-0" #10 prio=5 os_prio=31 tid=0x00007ffd4409d000 nid=0x4e03
waiting on condition [0x00007000011c9000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(Native Method)
  at samurai.core.AThread.run(BlockerExample.java:38)
  - locked <0x000000079579aa18> (a [Ljava.lang.String;)
```

So what is he doing?

Again, thread dump is

A snapshot of every thread in the JVM **at a particular point in time.**

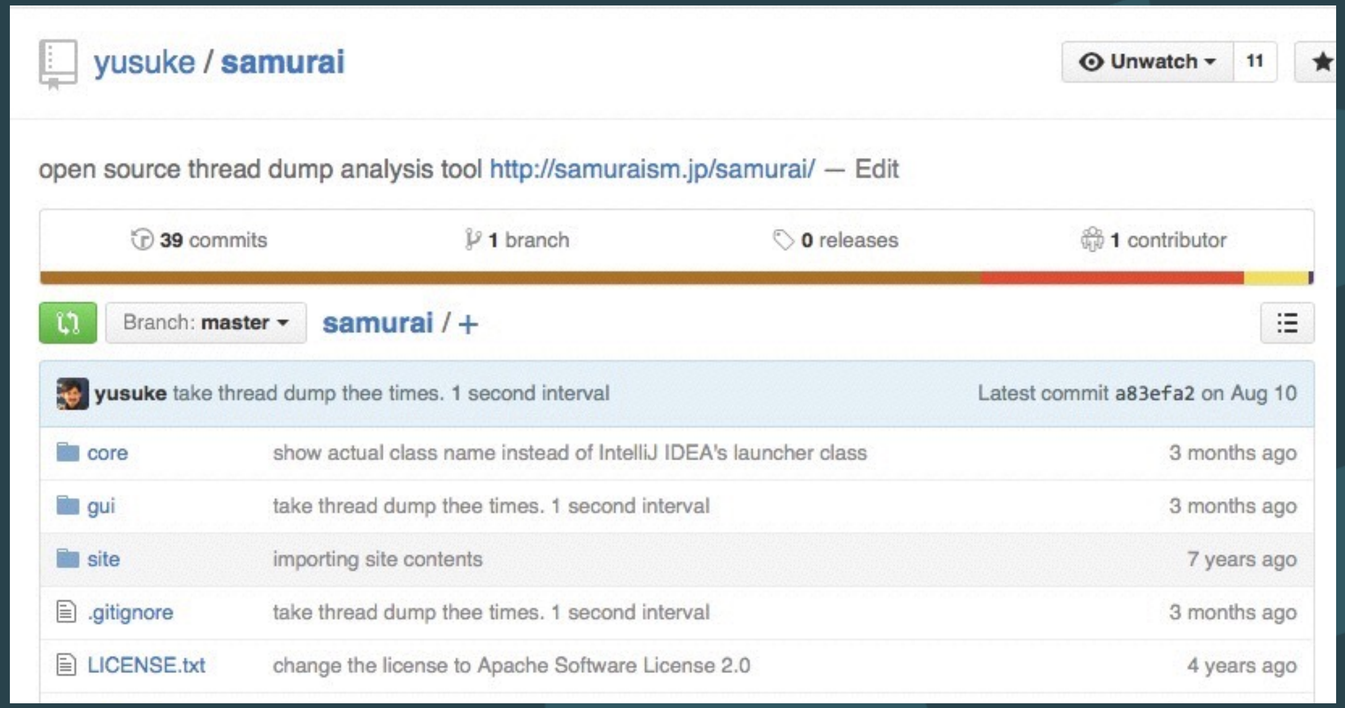
You can't tell what he's doing with a single snapshot

To analyze thread dumps

Take at least 3 times, with 1 to 2 seconds interval

Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9

Analyze with pleasure



Samurai

https://github.com/yusuke/samurai/

#BOF1518

Samurai

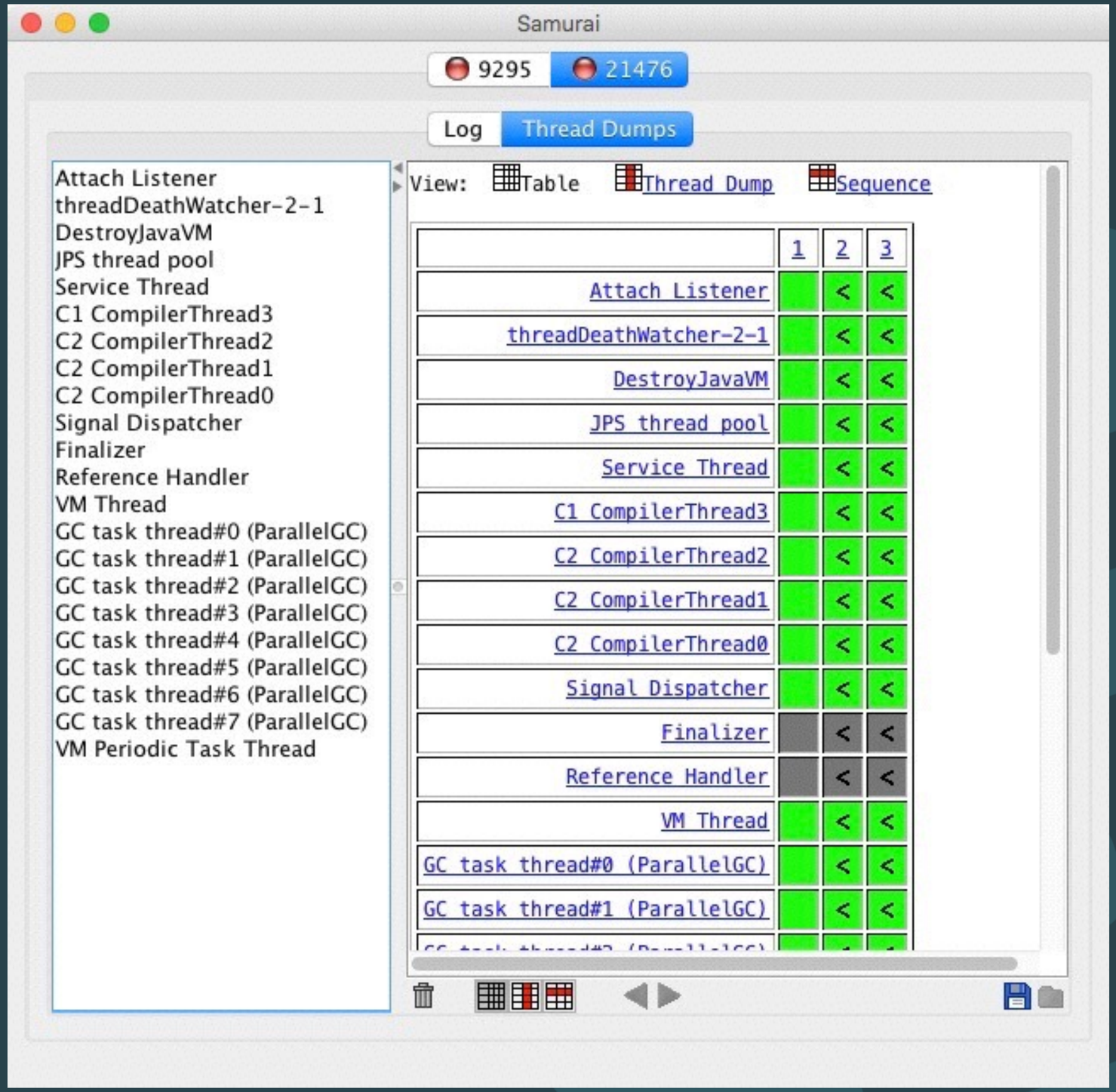
A GUI tool (Swing)

Thread dump visualization

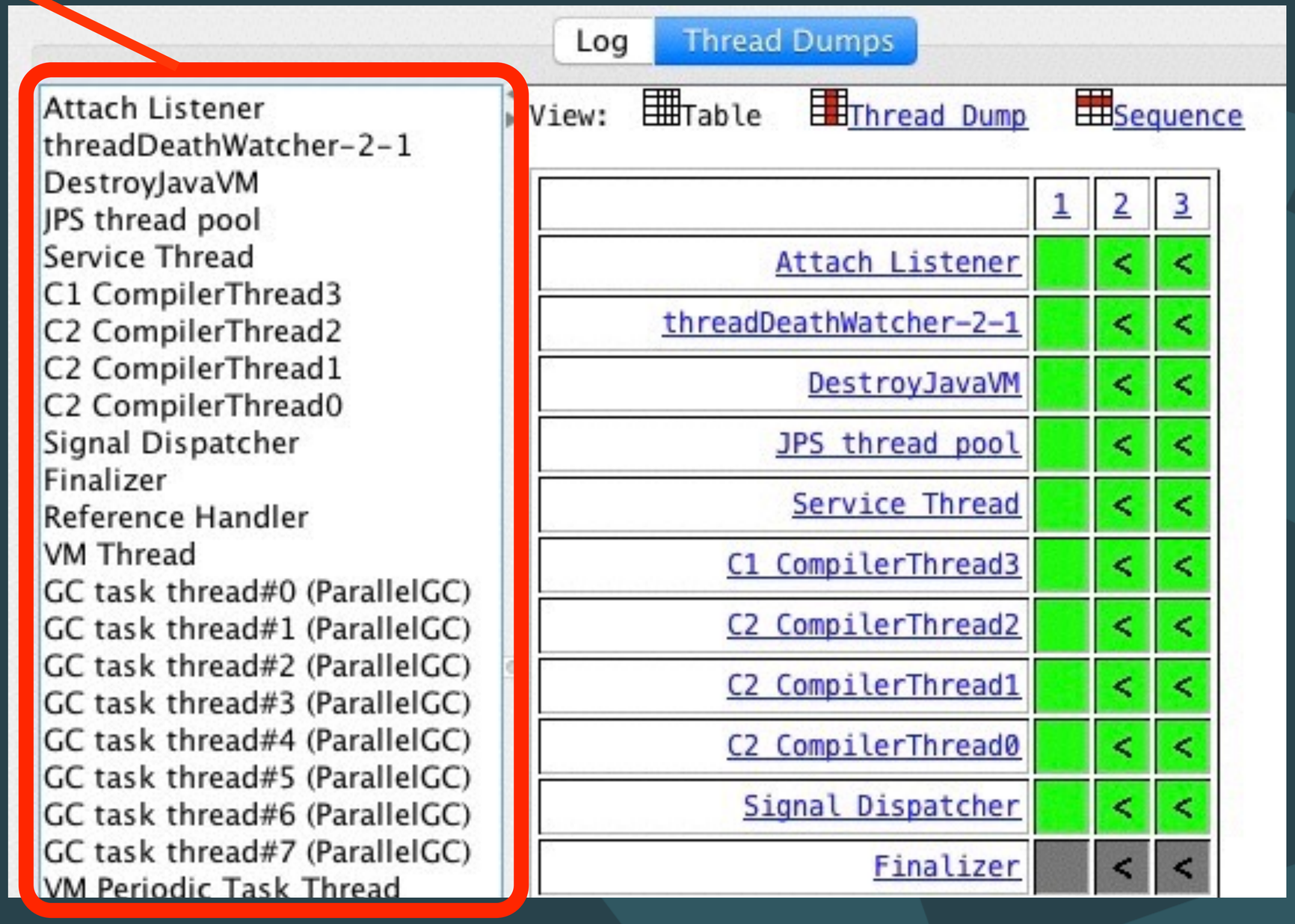
Multiple thread dumps in one compact view

100% open source (Apache License 2.0)

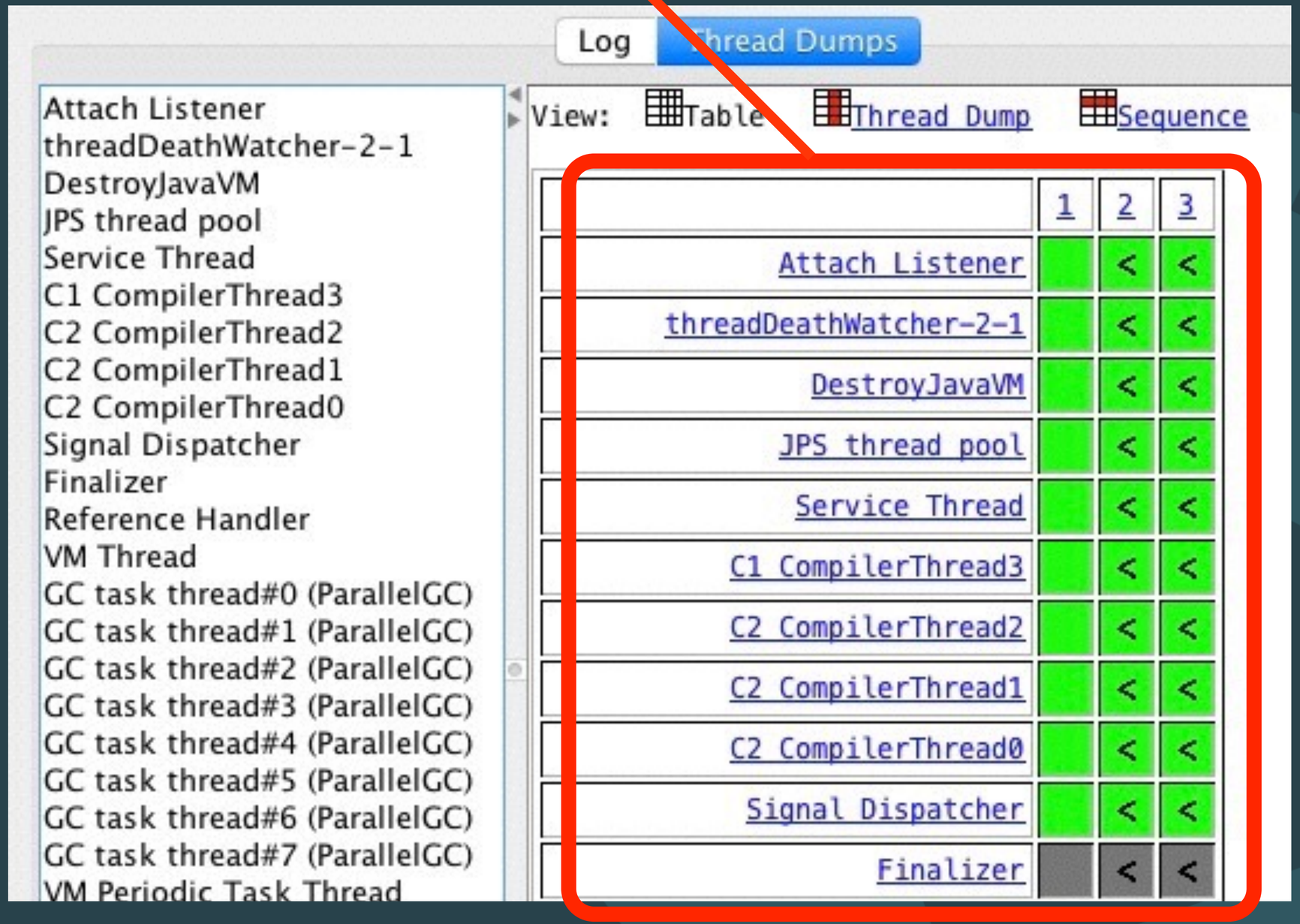
https://github.com/yusuke/samurai/


Samurai
9295
21476
Log
Thread Dumps
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer
Reference Handler
VM Thread
GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)
GC task thread#2 (ParallelGC)
GC task thread#3 (ParallelGC)
GC task thread#4 (ParallelGC)
GC task thread#5 (ParallelGC)
GC task thread#6 (ParallelGC)
GC task thread#7 (ParallelGC)
VM Periodic Task Thread
View:
Table
Thread Dump
Sequence
1
2
3
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer
Reference Handler
VM Thread
GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)

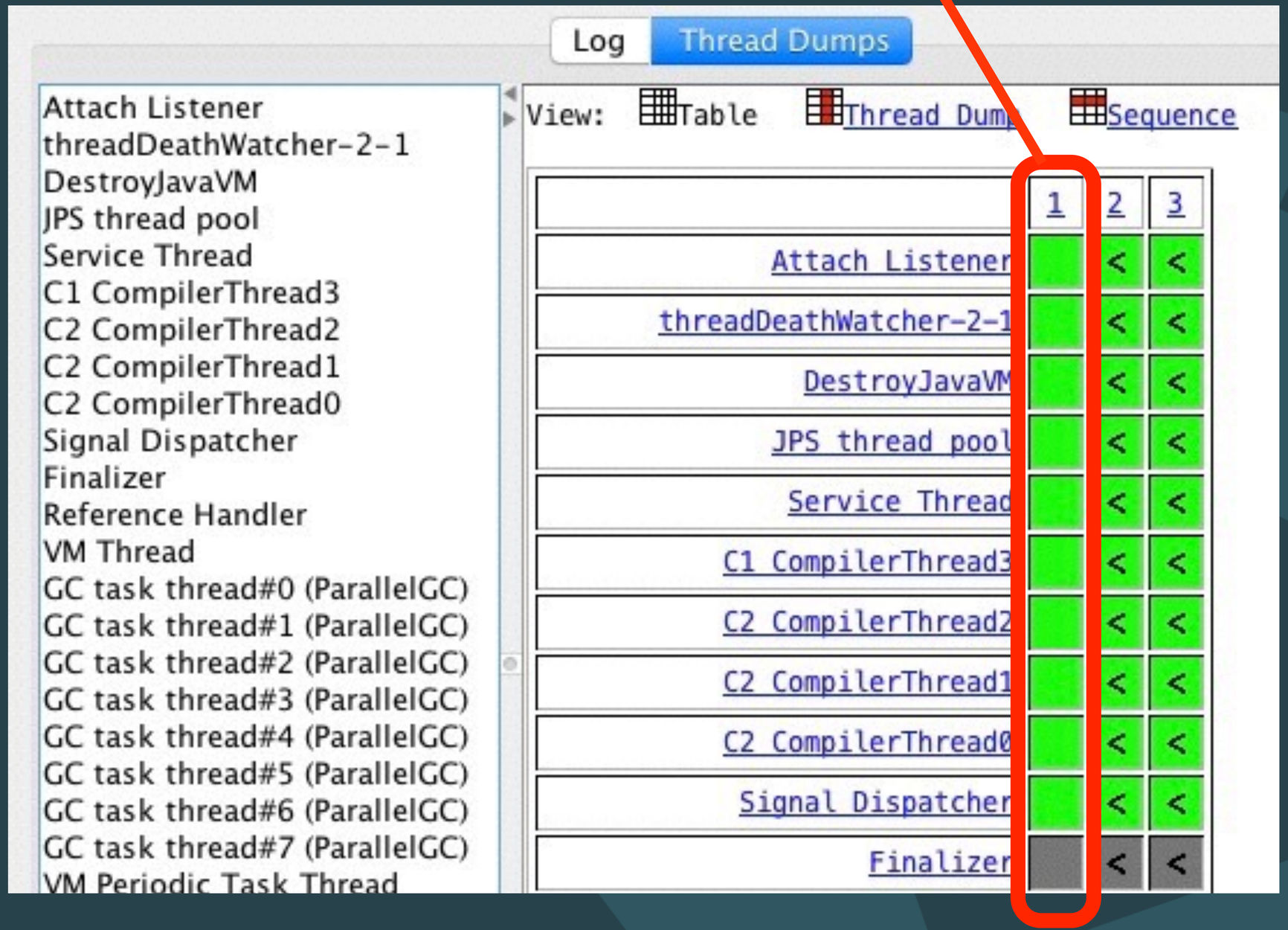
Threads


Log
Thread Dumps
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer
Reference Handler
VM Thread
GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)
GC task thread#2 (ParallelGC)
GC task thread#3 (ParallelGC)
GC task thread#4 (ParallelGC)
GC task thread#5 (ParallelGC)
GC task thread#6 (ParallelGC)
GC task thread#7 (ParallelGC)
VM Periodic Task Thread
View:
Table
Thread Dump
Sequence
1
2
3
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer

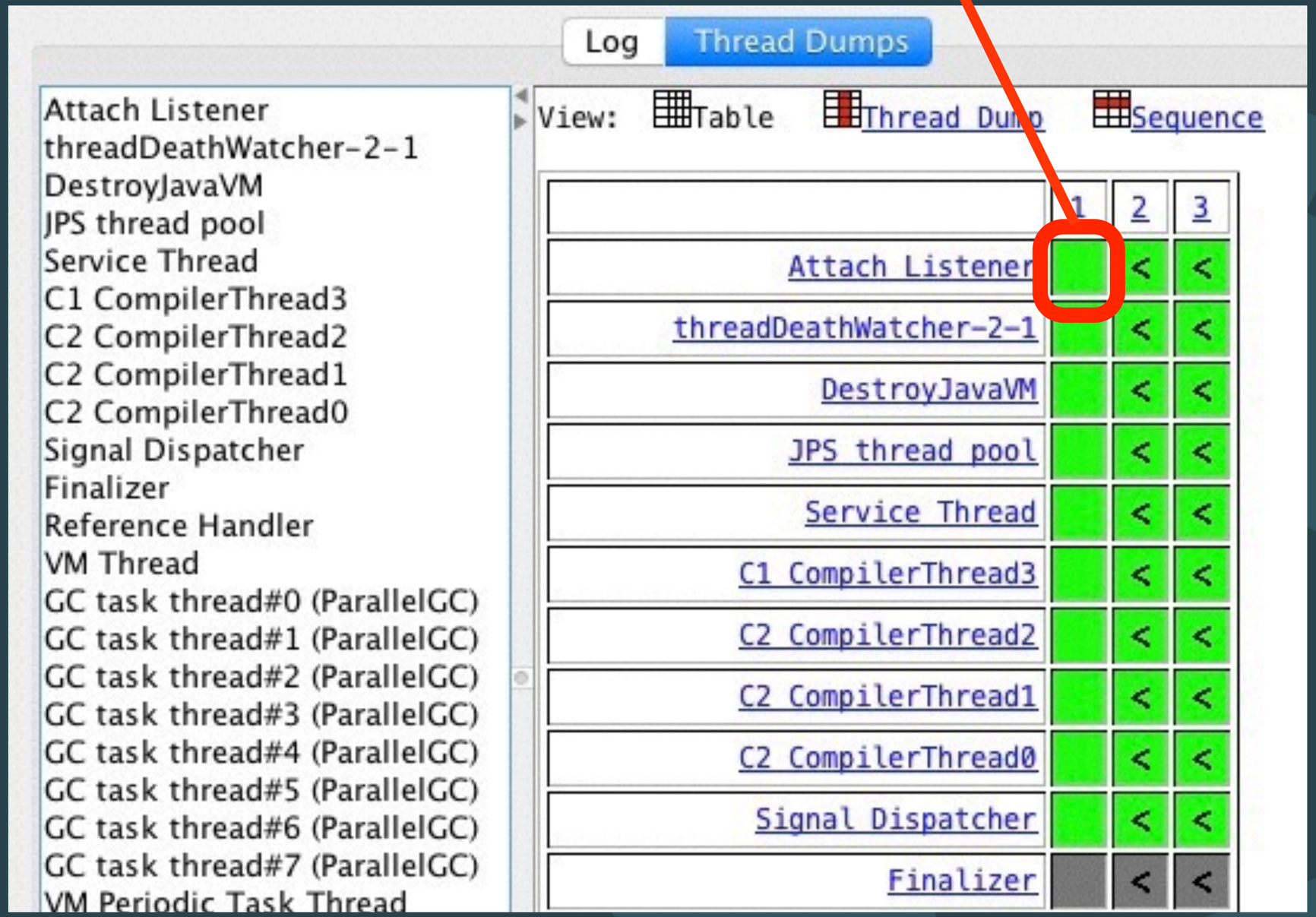
Thread dumps



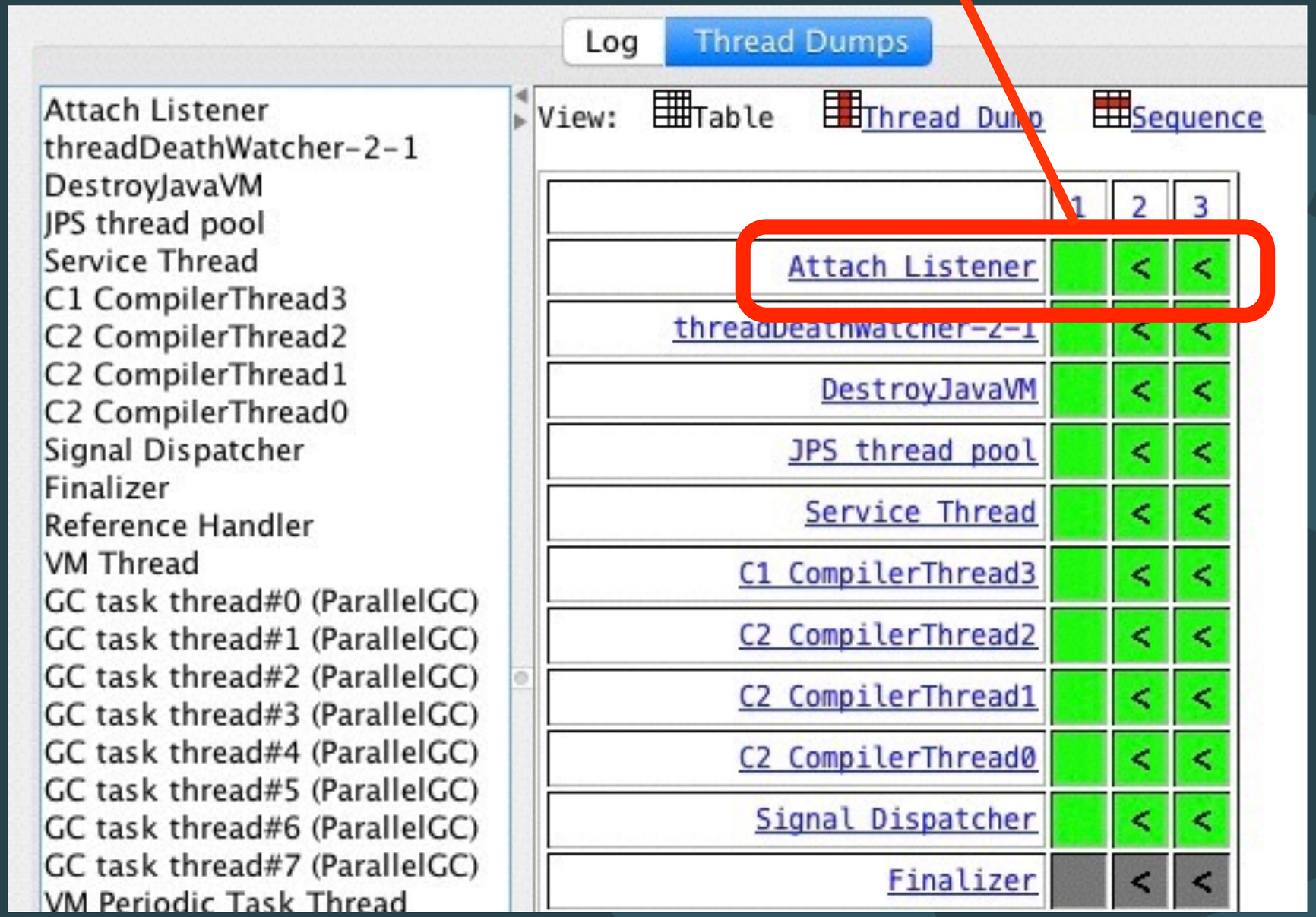
Thread dump


Log
Thread Dumps
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer
Reference Handler
VM Thread
GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)
GC task thread#2 (ParallelGC)
GC task thread#3 (ParallelGC)
GC task thread#4 (ParallelGC)
GC task thread#5 (ParallelGC)
GC task thread#6 (ParallelGC)
GC task thread#7 (ParallelGC)
VM Periodic Task Thread
View:
Table
Thread Dump
Sequence
1
2
3
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer

Stack trace


Log
Thread Dumps
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer
Reference Handler
VM Thread
GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)
GC task thread#2 (ParallelGC)
GC task thread#3 (ParallelGC)
GC task thread#4 (ParallelGC)
GC task thread#5 (ParallelGC)
GC task thread#6 (ParallelGC)
GC task thread#7 (ParallelGC)
VM Periodic Task Thread
View:
Table
Thread Dump
Sequence
1
2
3
Attach Listener
threadDeathWatcher-2-1
DestroyJavaVM
JPS thread pool
Service Thread
C1 CompilerThread3
C2 CompilerThread2
C2 CompilerThread1
C2 CompilerThread0
Signal Dispatcher
Finalizer

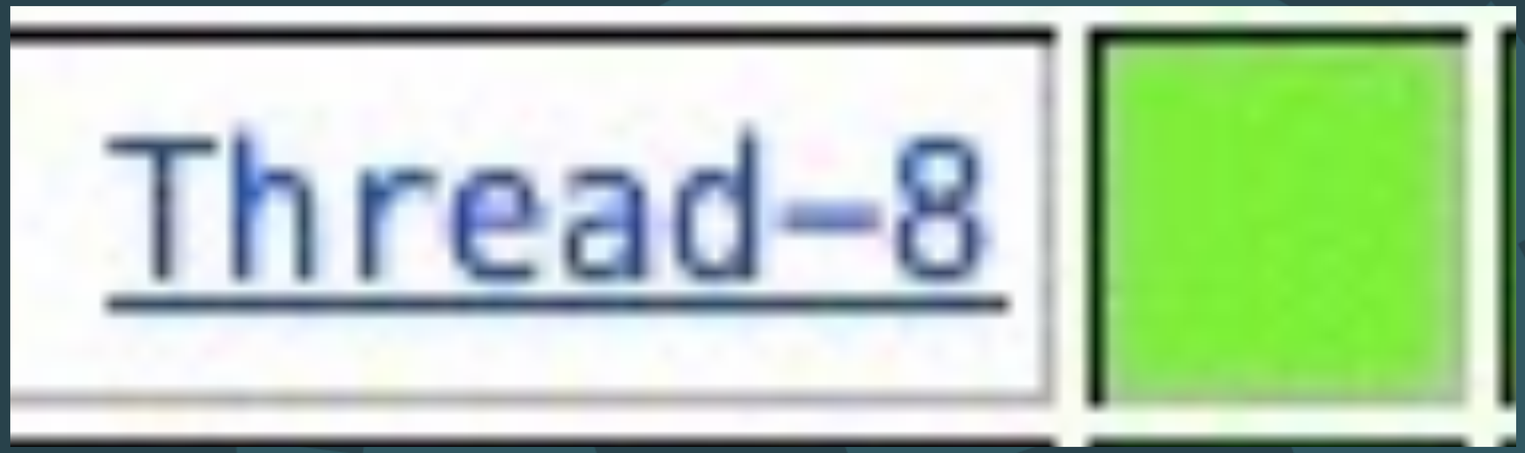
1 single thread



Running - green

#BOF1518

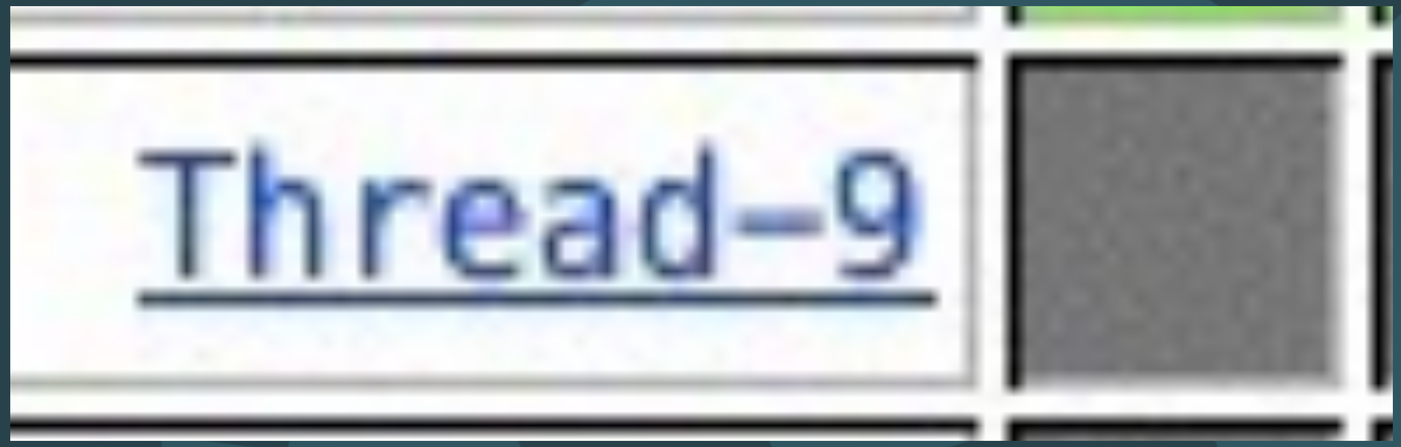
doing() something() meaningful()



Idle thread - gray

#BOF1518

Thread.sleep()
Object.wait()



Blocked(red) / Blocking(orange)

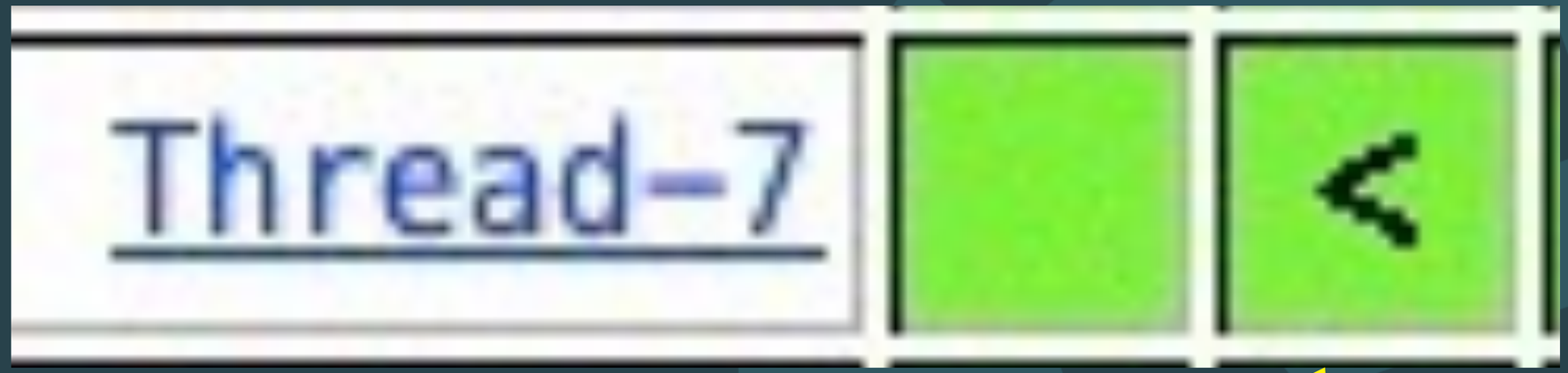
Blocking - trying to acquire lock

```
synchronzed(obj){
 serializedOperation(obj);
}
```

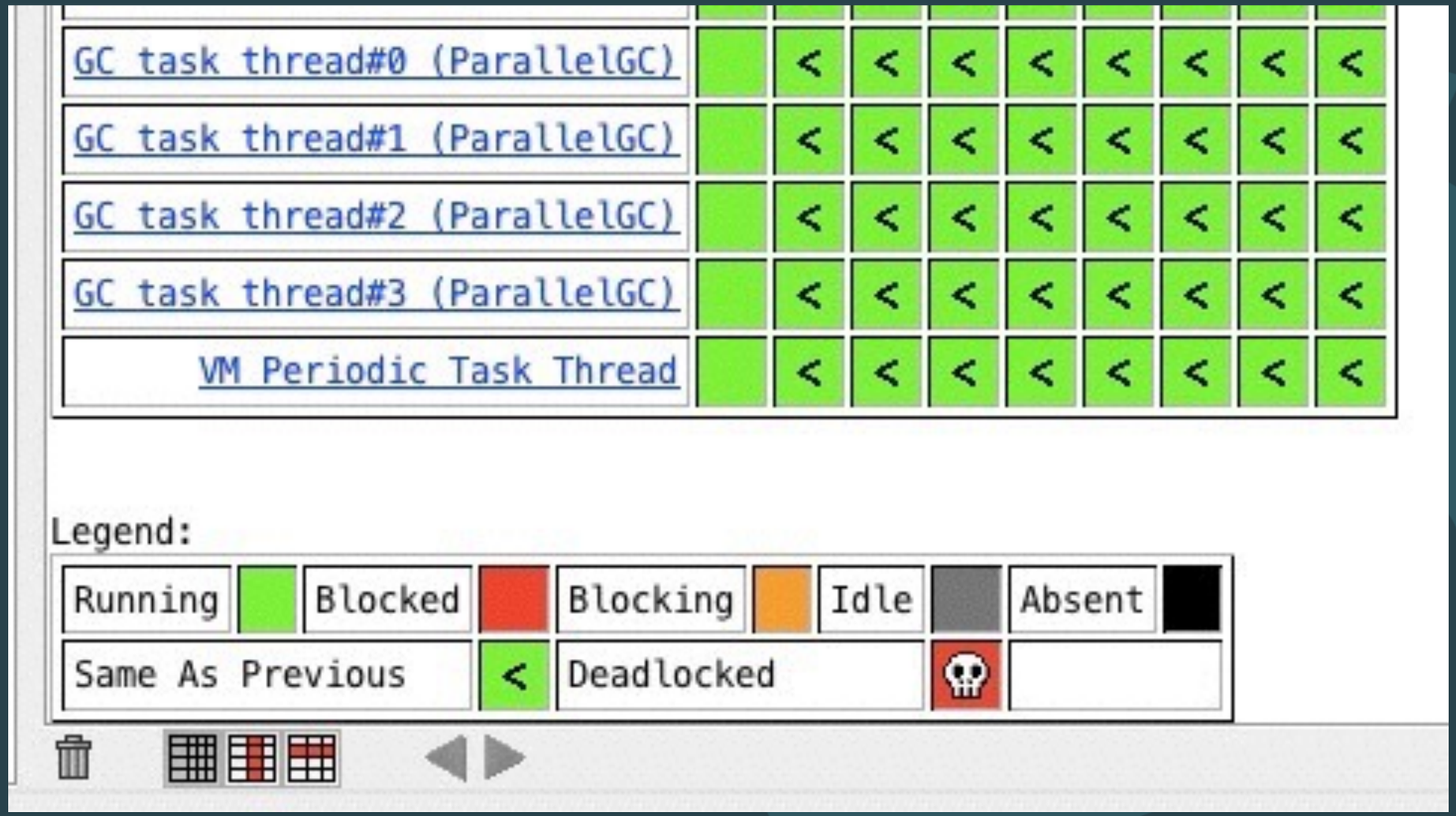
Blocked - acquired lock

```
synchronzed(obj){
 serializedOperation(obj);
}
```

Stack trace comparison



stack trace is as same as previous


GC task thread#0 (ParallelGC)
GC task thread#1 (ParallelGC)
GC task thread#2 (ParallelGC)
GC task thread#3 (ParallelGC)
VM Periodic Task Thread
Legend:
Running
Blocked
Blocking
Idle
Absent
Same As Previous
Deadlocked

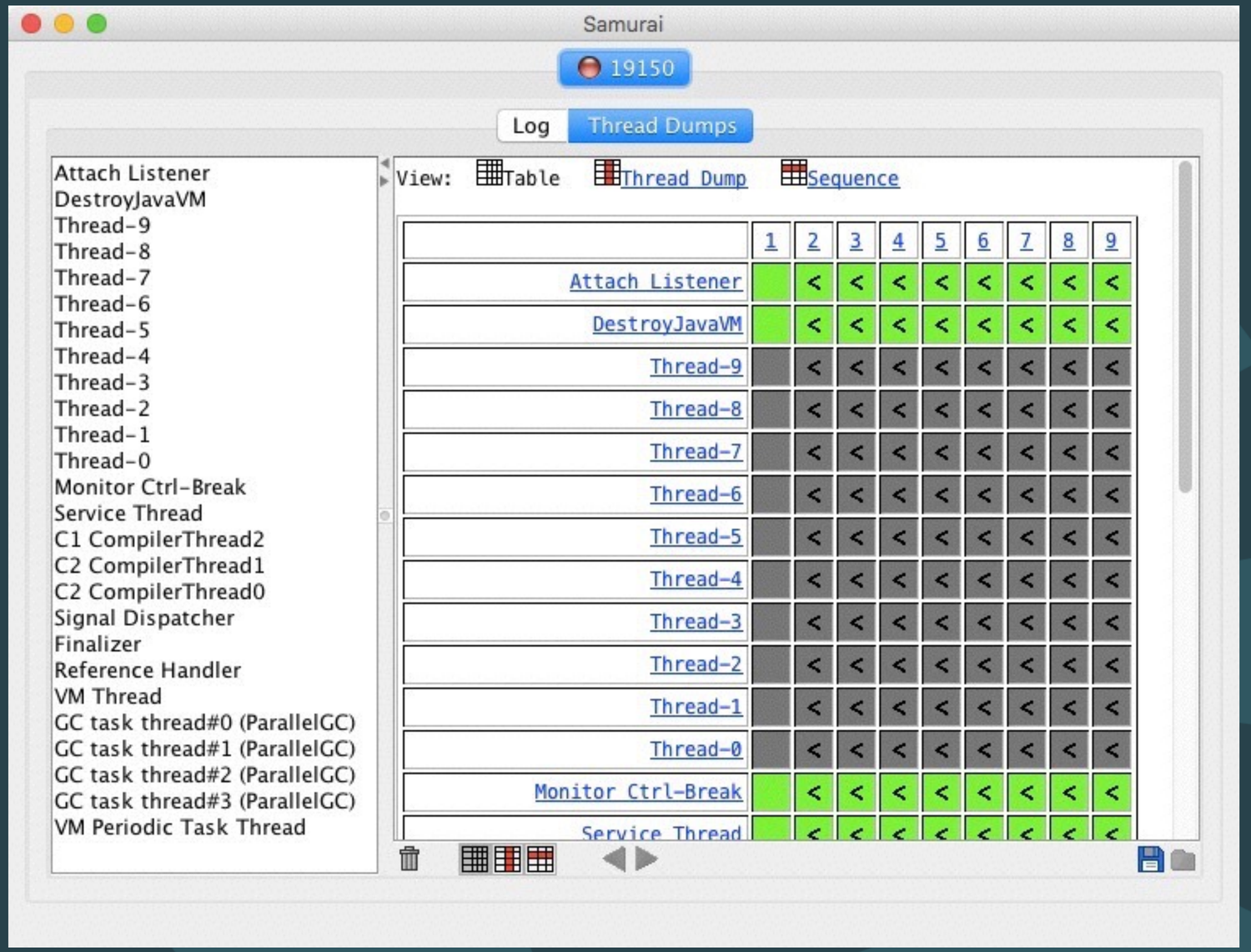
#BOF1518

demo /démoʊ/

Typical look of thread dumps

Idle

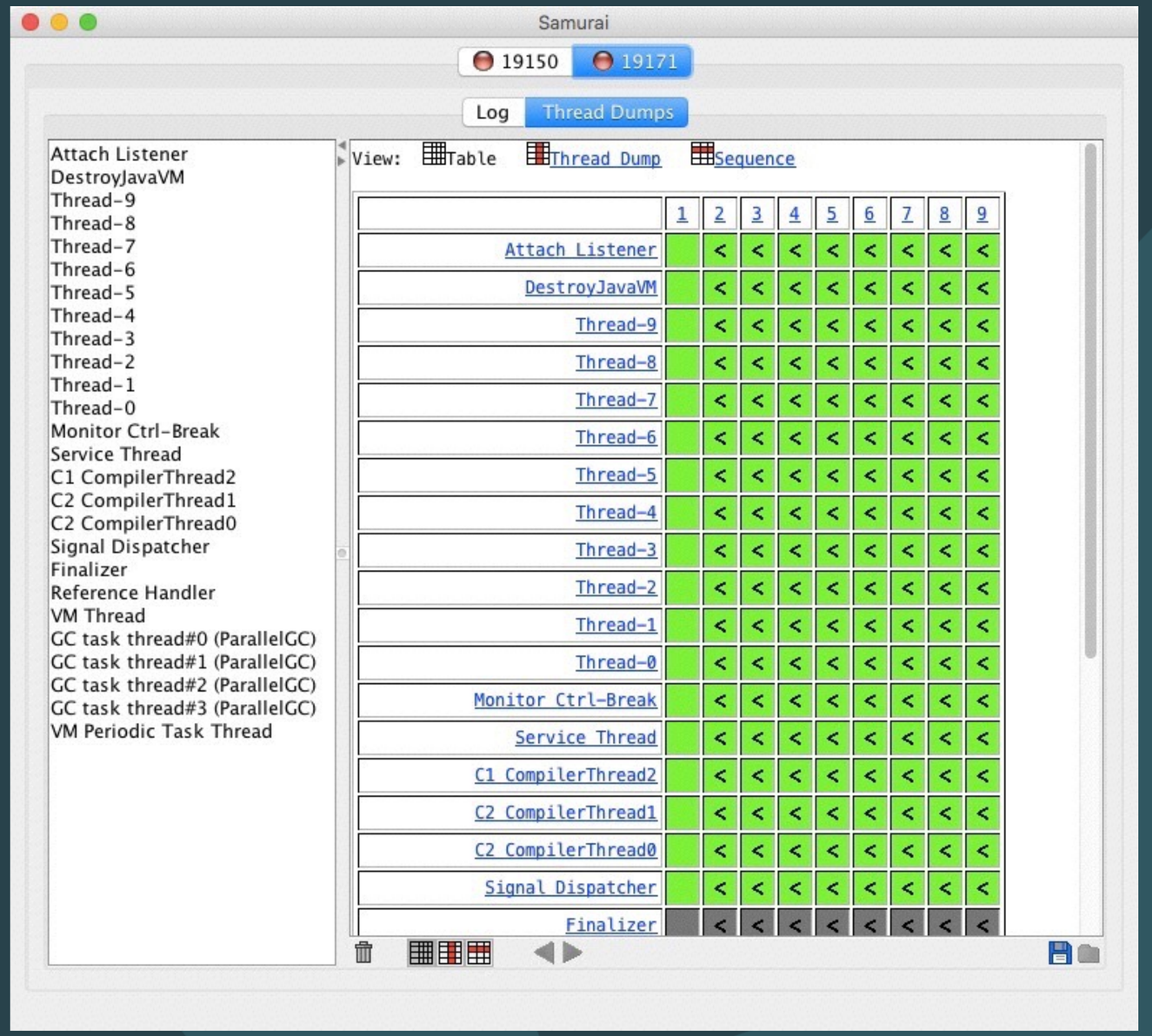
almost all gray



#BOF1518

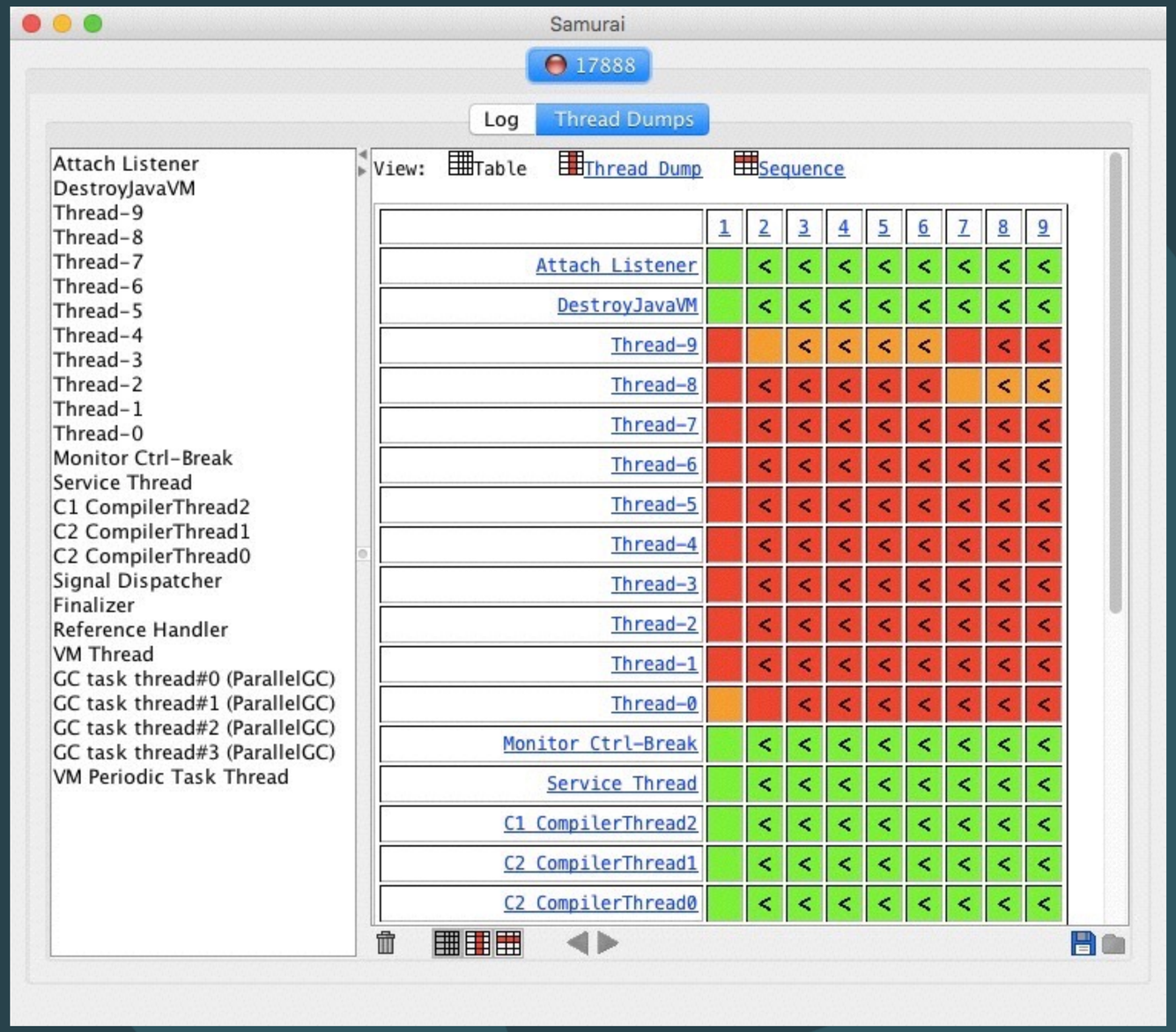
Running

almost all green



Race condition

Many red cells
few Orange cells



Launching Samurai

```
$ wget http://bit.ly/samurai30
$ java -jar samurai-3.0.jar
```

tools.jar required for taking local thread dump

```
$ java -cp $JAVA_HOME/lib/
tools.jar:samurai-3.0.jar samurai.swing.Samurai
```

Compatibility

Tested with:

- Sun JDK 1.4.2+
- Oracle JDK 1.6+
- OpenJDK 1.5+
- JRockit 1.4.2+
- HP JDK 1.4.2+
- IBM JDK 1.3.1+

Samurai roadmap

#BOF1518

Command line UI for headless environment

Migrate to JavaFX

Pattern detection / Similarity calculation

IDE Plug-in

Analysis of CPU usage of each threads

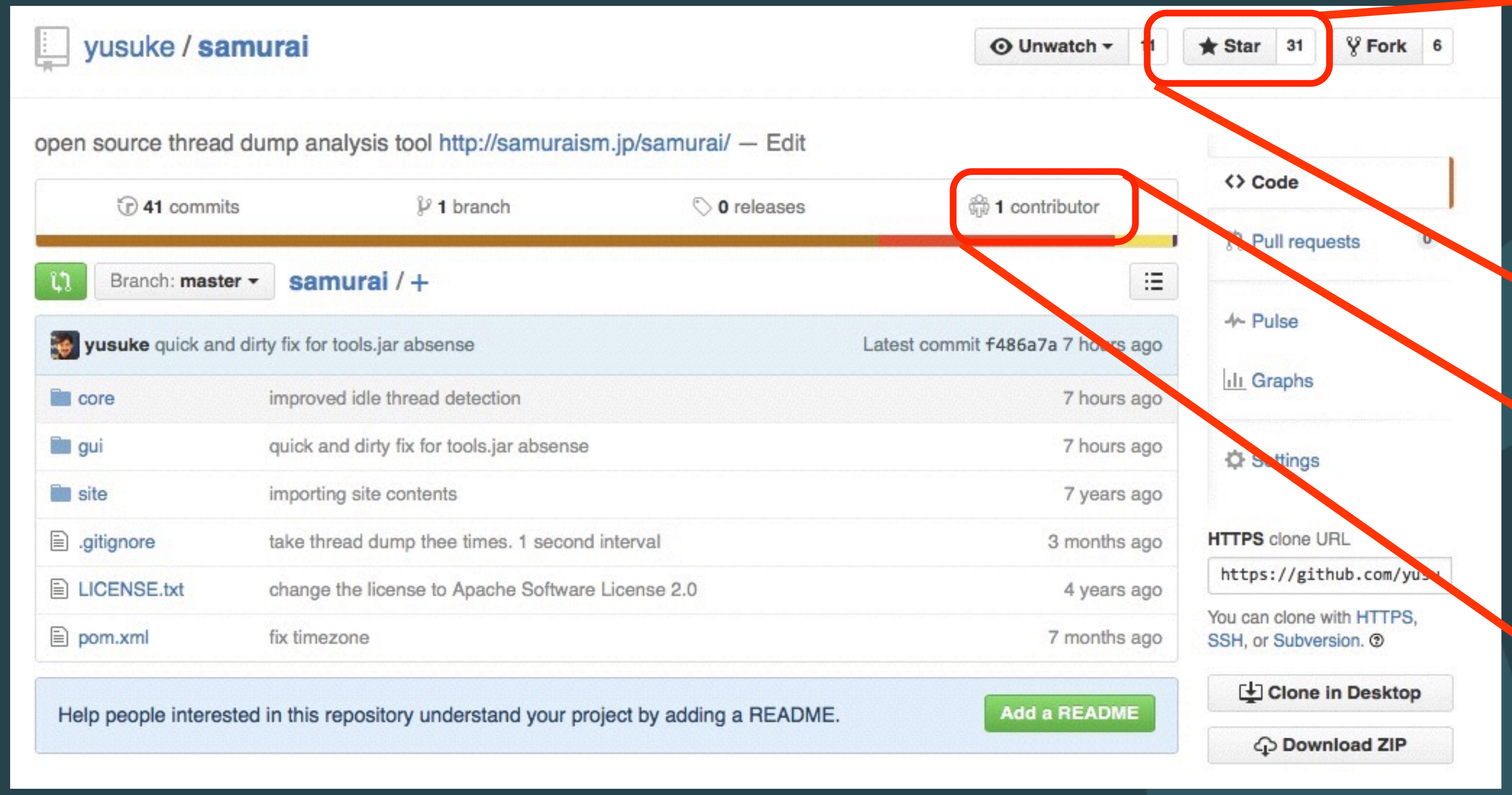
Conclusion

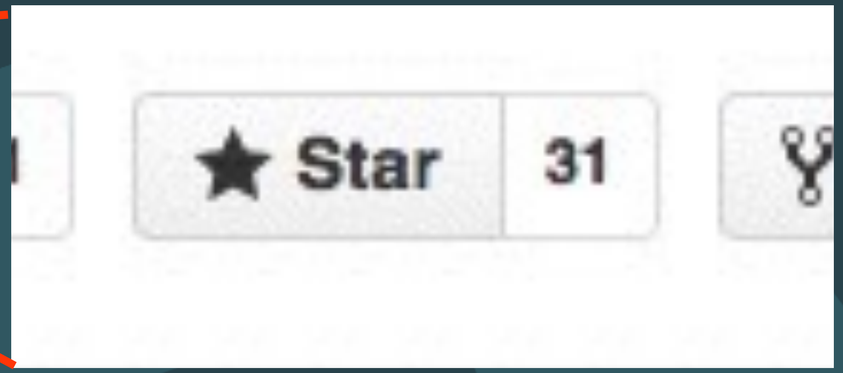
Thread dump is your friend.

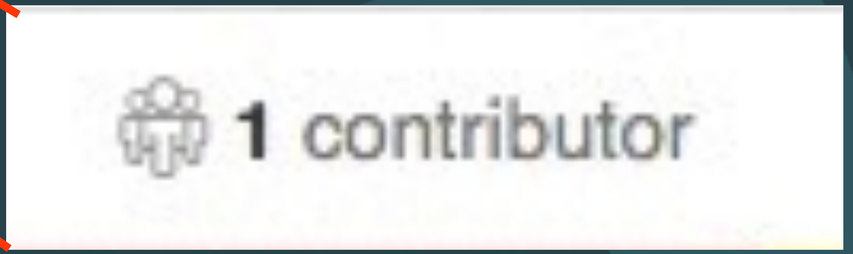
Be familiar with thread dump before you have a serious trouble.

I need your help

#BOF1518








#BOF1518


@yusuke


@yusukey