Identifying Threads Groups Based on Escaping Objects

Tristan M. Basa, Gerhard Dueck
University of New Brunswick
Faculty of Computer Science
tbasa@unb.ca, gdueck@unb.ca

INTRODUCTION

When a Java thread allocates an object inside the Java Virtual Machine (JVM), the object can get accessed by other objects from other threads (see figure 1). When this happens, the object is considered to have *escaped.*

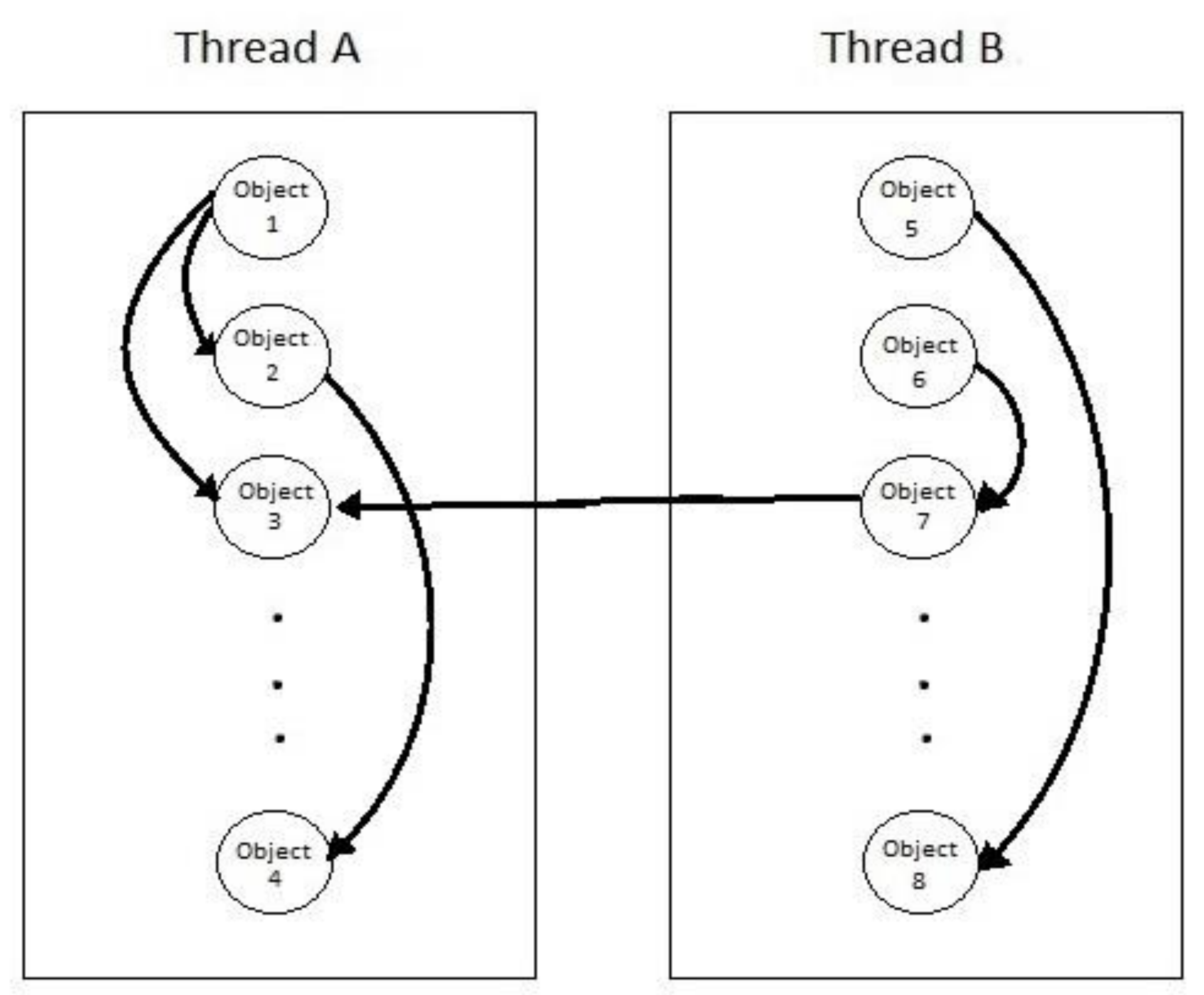

Figure 1. Object 3 is considered escaped.

PROBLEM STATEMENT

- Global garbage collection (GC) can cause long execution pause time which results in degradation of overall system performance.
- According to a recent study, up to 56% of all objects escape.
- Threads involved in escaping objects must be stopped when GC runs. For example in figure 1, both threads must be stopped.

PROPOSED SOLUTION

- Partition the heap into smaller regions.
- Assign threads to regions on the following bases:
  - One region per thread
  - One region per group of threads.

By grouping threads, a *region-based* GC can be performed on one region without having to stop other threads from other regions.

Problem Scenario:

- Objects in thread A of region I are escaping to objects in thread B of region II and vice versa (see figure 2).
- All threads in regions I and II (including thread C) need to be stopped in order to perform region-based GC.

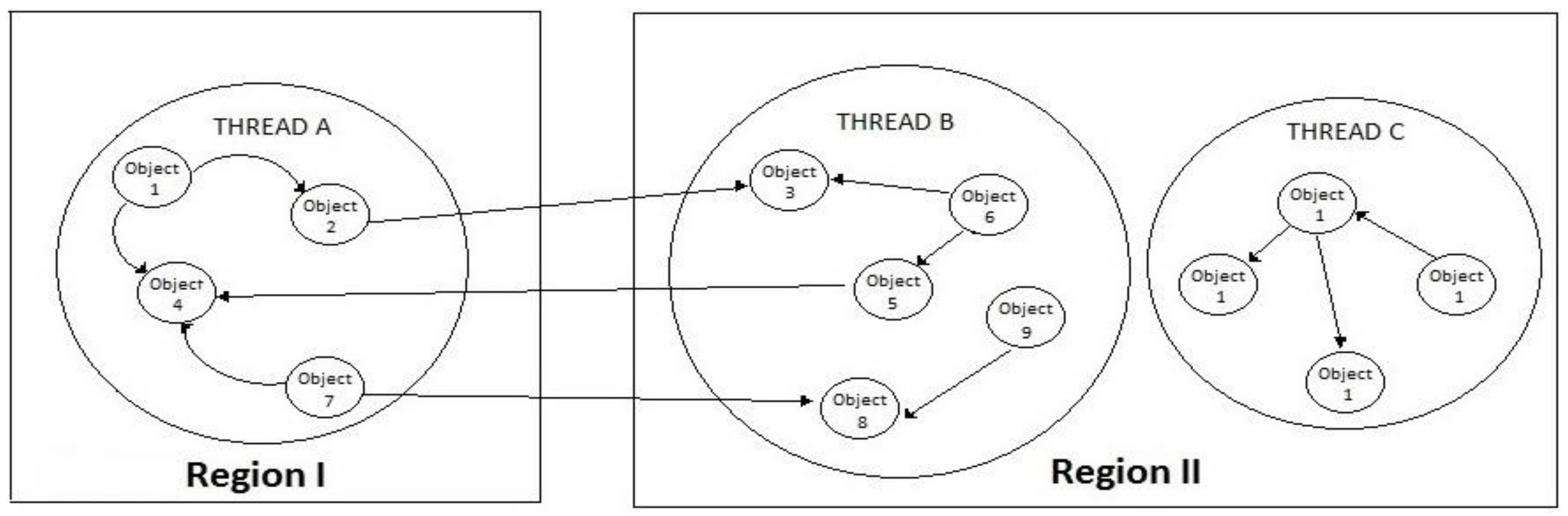

Figure 2. Threads in both regions have to be stopped in order to perform a region-based GC.

Possible Solution:

- Threads A and B can be grouped together in one region as shown in figure 3.
- GC can be done only on Region I (only threads A and B need to be stopped) and thread C can continue its execution.

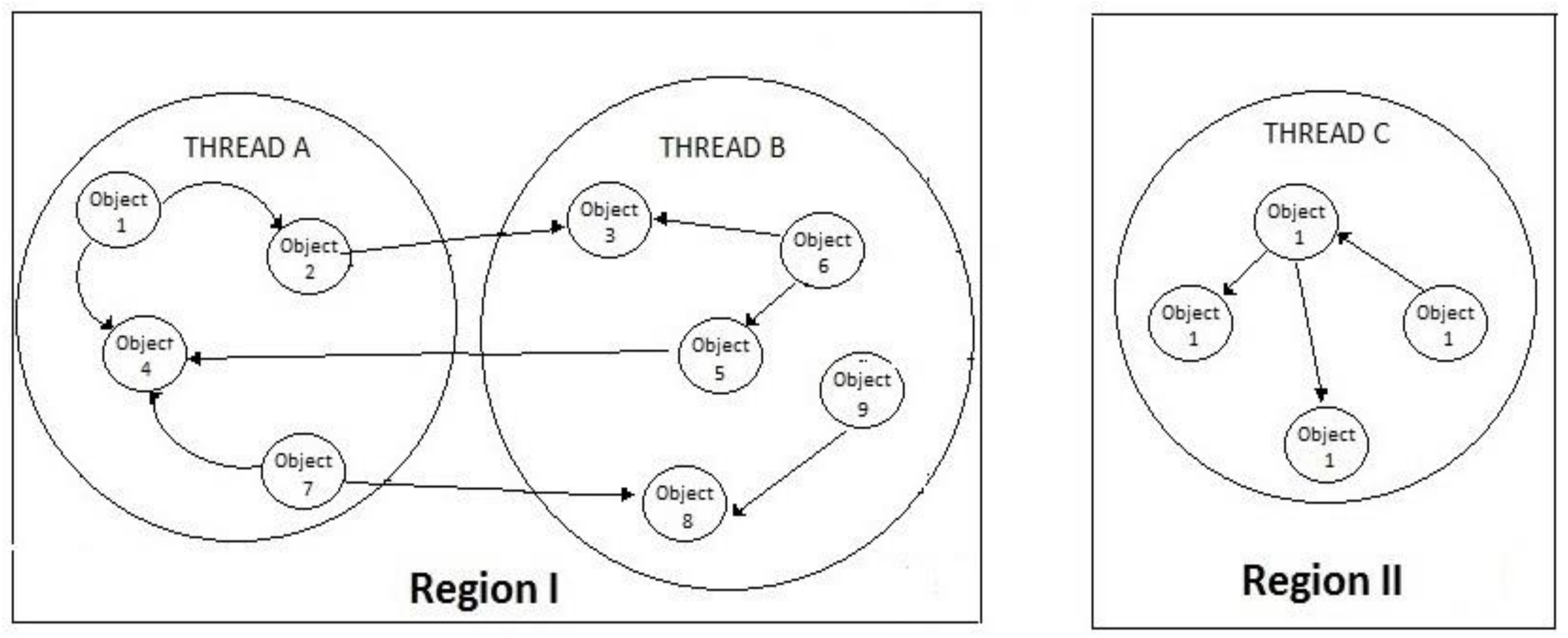

Figure 3. Only threads in Region I are needed to be stopped in order to perform a region-based GC.

PROPOSED APPROACH

- *JVM Instrumentation* – we can add agents to gather data as the program is executing.
- *Tracefile* – we can read a file that contains a series of instructions that emulates memory management operations in the JVM.