Memory Management for Real-time Java

Jan Vitek

PURDUE UNIVERSITY

IBM

Background

- Started working on real-time Java in 2001 within a DARPA funded project. At the time, there was no real RTSJ implementation.
- Developed the Ovm virtual machine framework, a clean-room, open source Java virtual machine. ~15 man/year effort
- Fall 2005, Boeing and Purdue conducted the first flight test with Java on a plane.

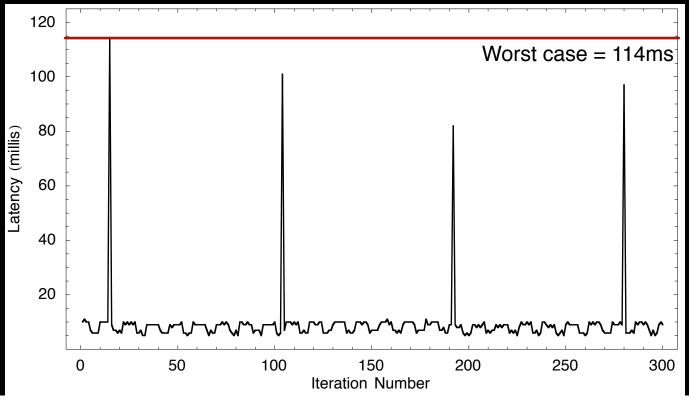
Duke's Choice Award winner application

Background: Java and Real-time

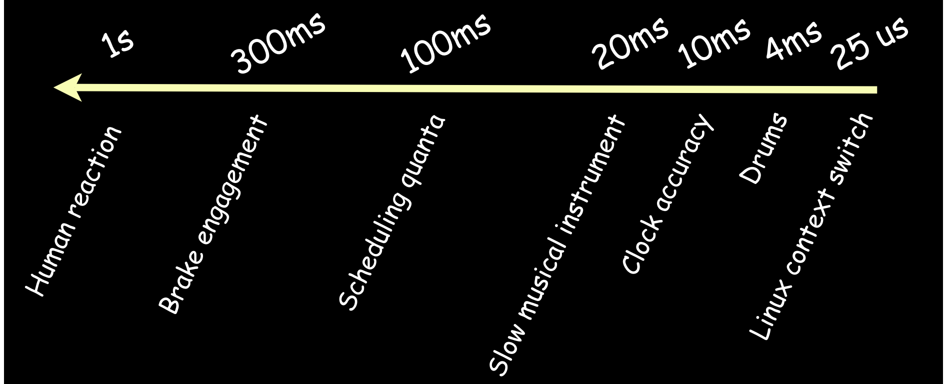
- The Real-time Specification for Java (or RTSJ) provides a standardized extension to the Java platform for hard real-time processing
- Multiple products:
  - PERCS (AONIX) ahead-of-time code generator
  - JamaicaVM (AICAS) ahead-of-time
  - McKinack (SUN) based on Hotspot, JIT, SMP, RTGC
  - Websphere (IBM) based on J9, ahead/JIT, SMP, RTGC
- Applications: Avionics, shipboard computing, banking, telco

Experiment

- Real-time Java collision detector (20Hz)
- Bartlett's Mostly Copying Collector. Ovm. Pentium IV 1600 MHz, 512 MB RAM, Linux 2.6.14, GCC 3.4.4



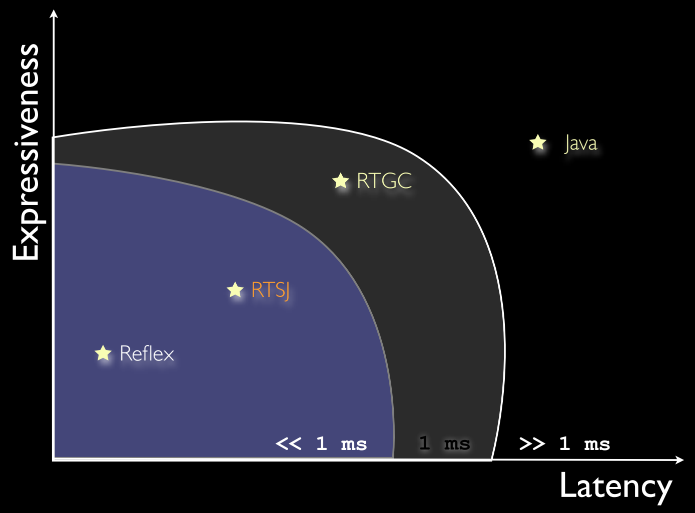
Time scale



1s
300ms
100ms
20ms
10ms
4ms
25 us

Human reaction
Brake engagement
Scheduling quanta
Slow musical instrument
Clock accuracy
Drums
Linux context switch

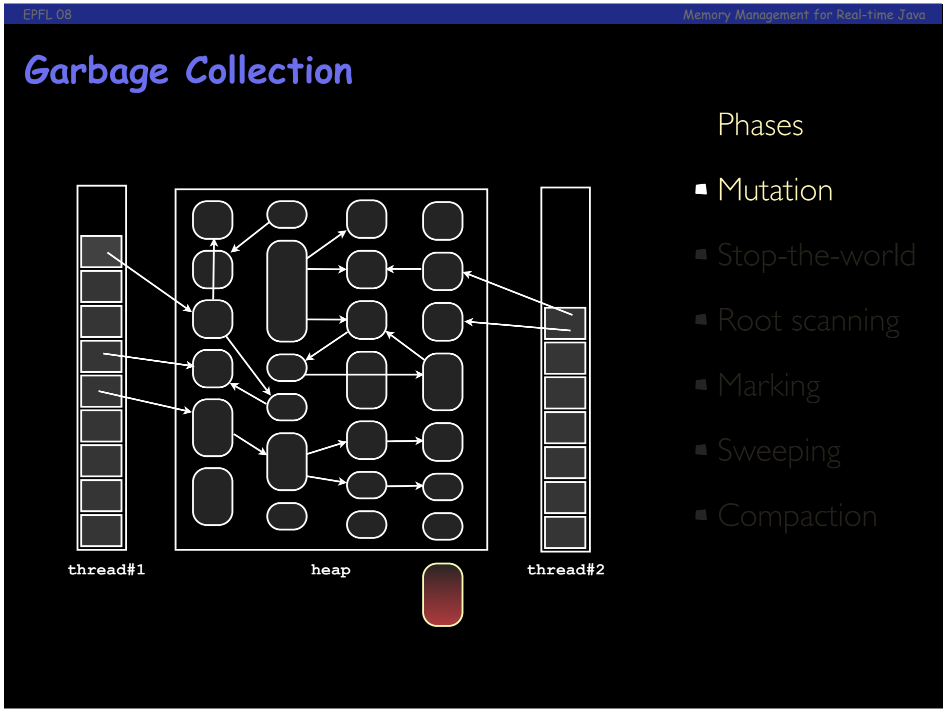
Design space



Garbage Collection

Garbage Collection

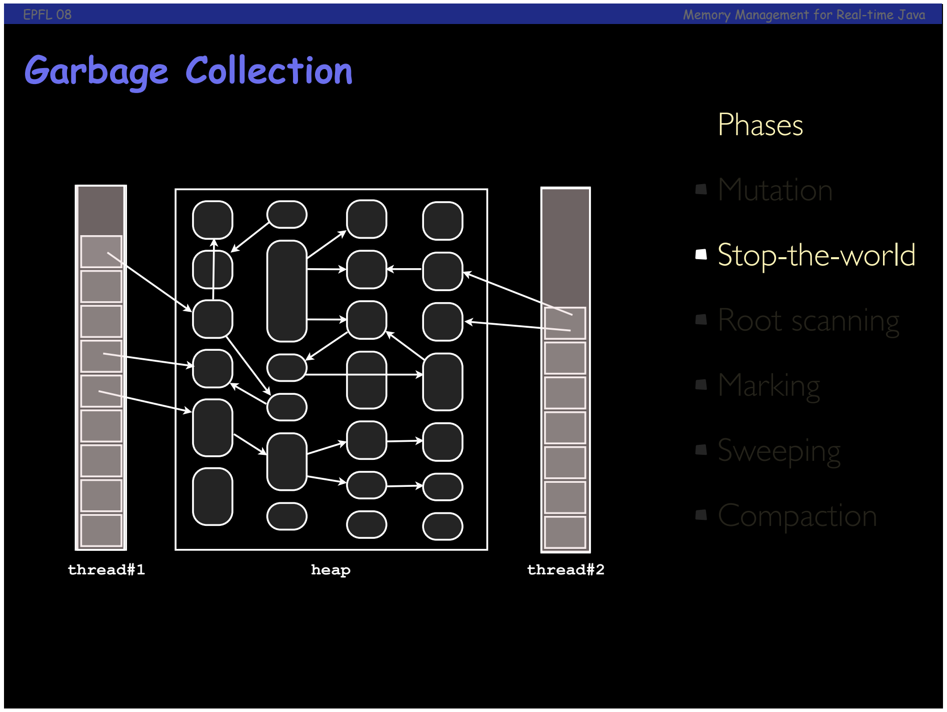
- A Garbage Collector (GC) is an algorithm that automatically finds unused objects in the memory of an application and prepares them for reuse
- GC frees programmers from worrying about the exact lifetime of objects and ensures that the heap will not be corrupted by access to previously freed data
- *... but introduces pauses that may be **O**(heap) and can increase the memory required. Moreover, pauses occur at unpredictable times, especially in concurrent programs*



Garbage Collection

Phases

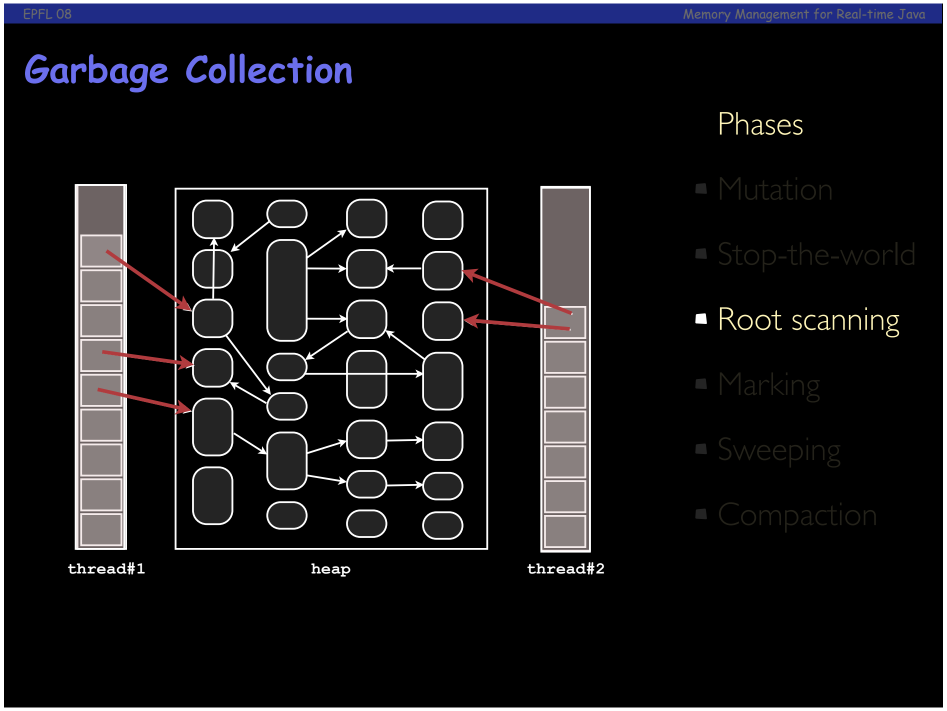
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

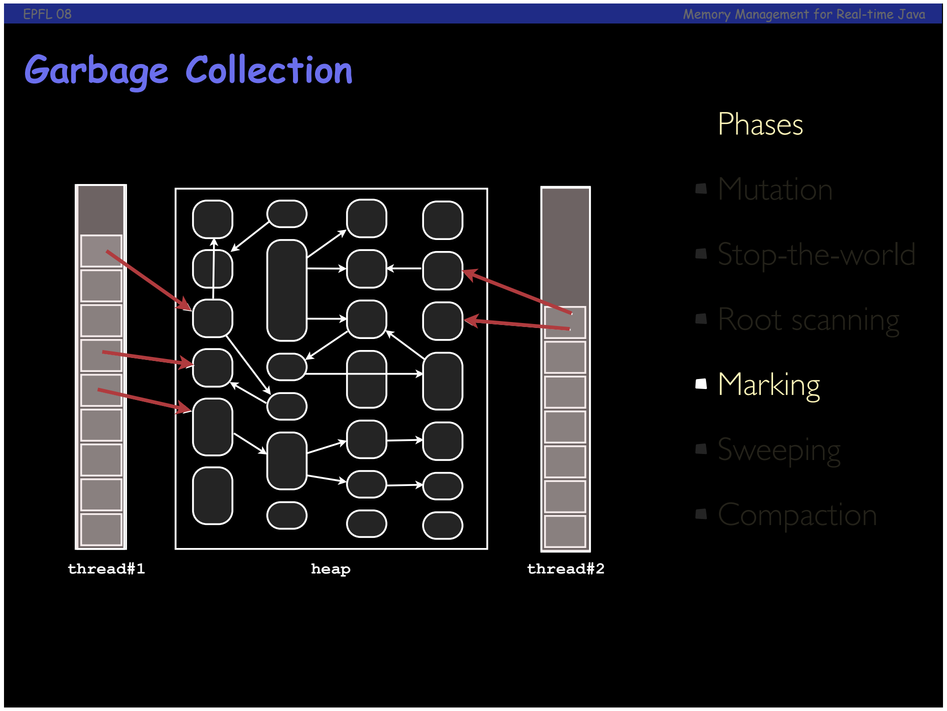
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

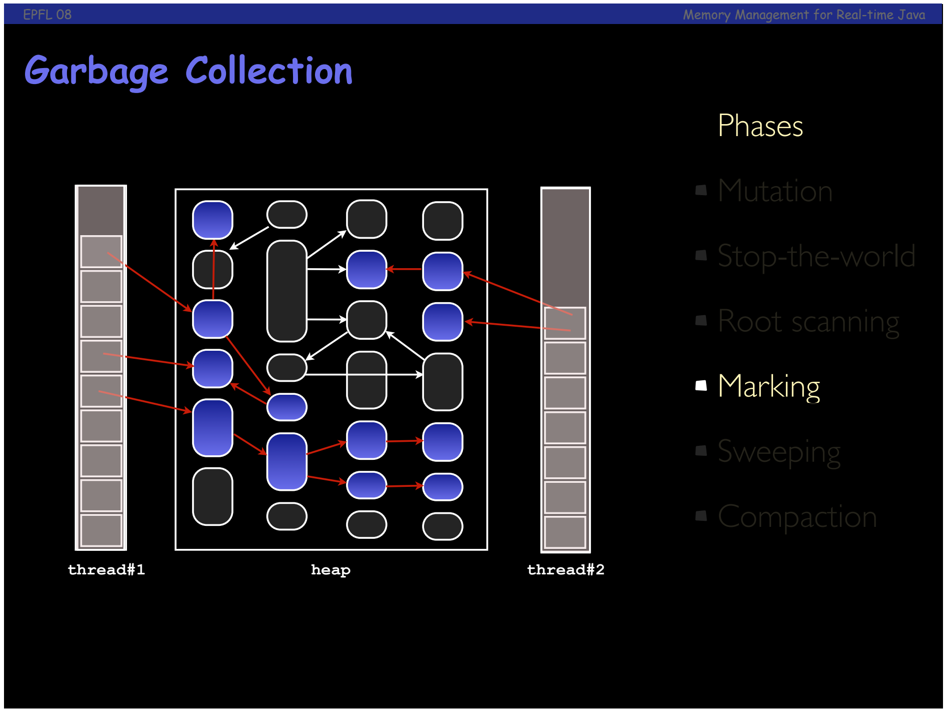
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

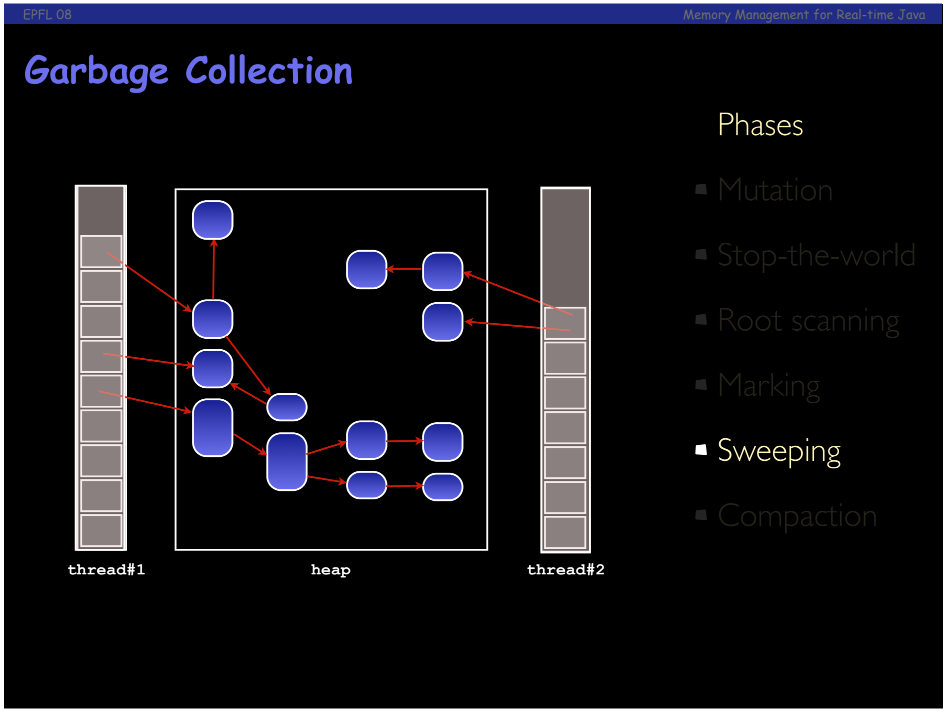
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

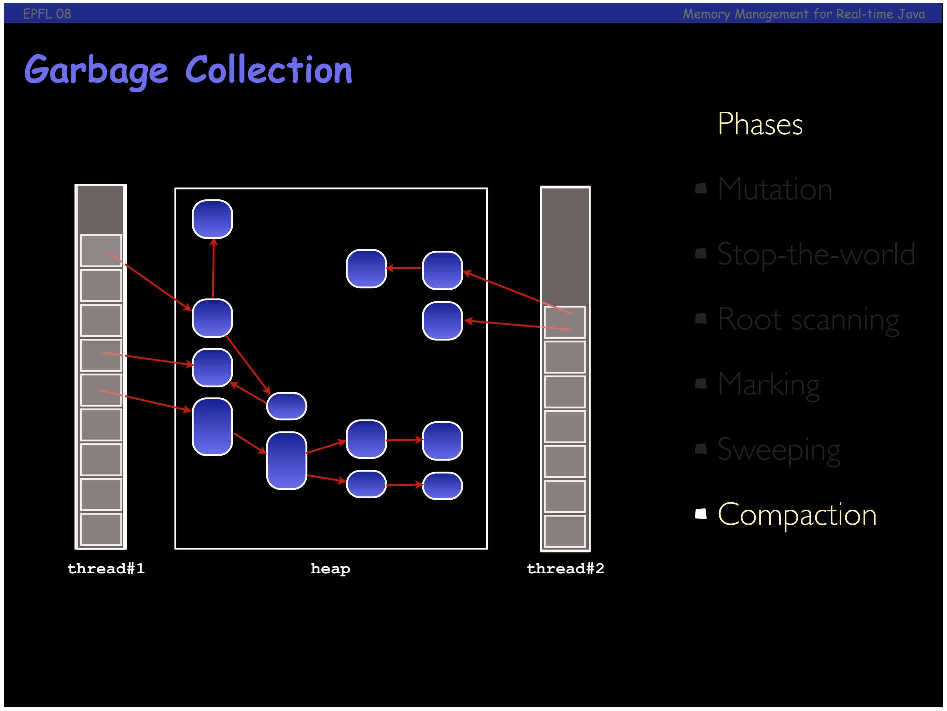
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

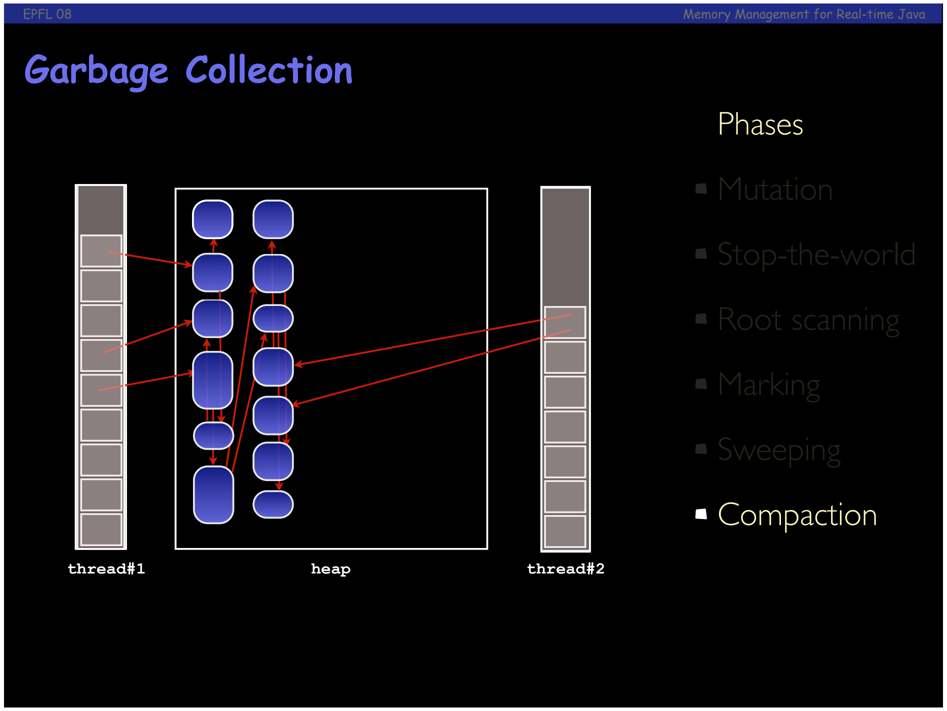
- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction



Garbage Collection

Phases

- Mutation
- Stop-the-world
- Root scanning
- Marking
- Sweeping
- Compaction

GC is Easy

- If responsiveness is not an issue,
  the GC can complete in one long pause under the assumption that there is no interleaved application activity
- Marking is easy
  if the graph does not change while you are searching it.
- Copying/compacting objects and fixing up the heap is easy
  if the application is prevented from accessing the heap

Real-time Garbage Collection

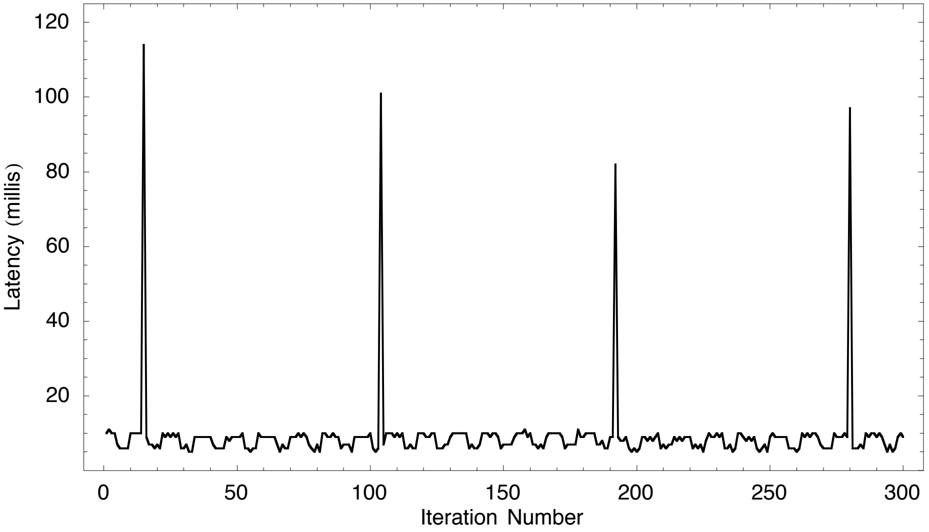
Real-time GC

- A Real-time GC must provide time and space predictability
  - provide a performance model that can be used to guarantee that programs do not run out of memory or experience pauses that violate their timing constraints
- A Real-time-GC must support defragmentation of the heap if it is to be used with long-lived applications
- Multi-processor support is unavoidable
- Throughput should not degrade overly

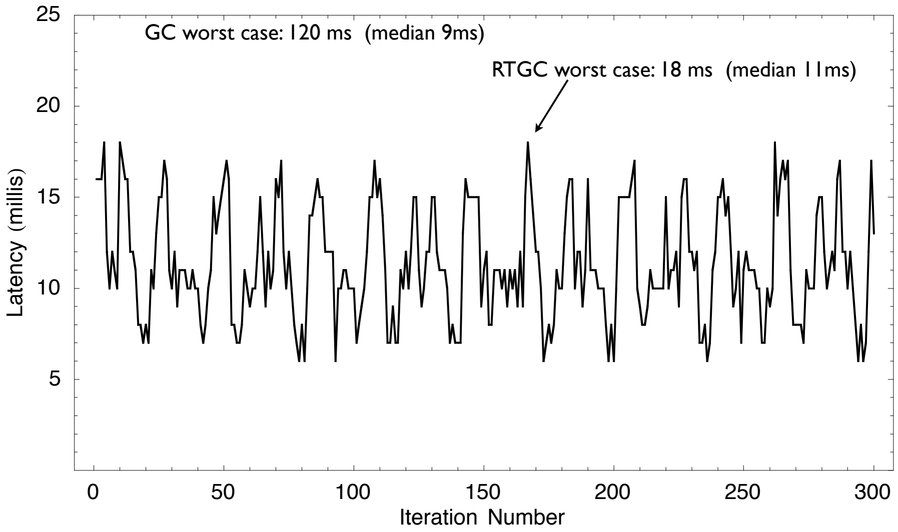
Collision Detector

- Experiment:
  - Pentium IV 1600 MHz, 512 MB RAM, Linux 2.6.14, GCC 3.4.4
  - Application: Real-time Java collision detector (20Hz)
  - Virtual machine: Ovm

CD with GC

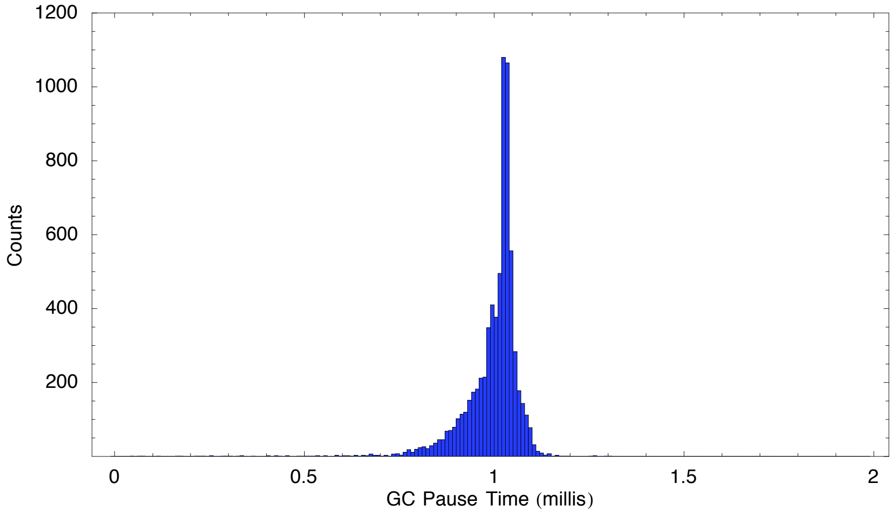


CD with RTGC


GC worst case: 120 ms (median 9ms)

RTGC worst case: 18 ms (median 11ms)

RTGC Pause time distribution


Counts
GC Pause Time (millis)