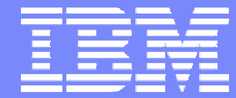


JVM Garbage Collection Theory

Richard Cole (rich.cole@uk.ibm.com)
Stephen Flavell (stephen_flavell@uk.ibm.com)

Java on z/OS L3 Service
Java Technologies - Hursley

http://www.ibm.com/developerworks/java/jdk/diagnosis/

© 2005 IBM Corporation

Overview

- What is the (IBM) JVM Storage Model
  - The Java Heap, Garbage Collection and native storage
  - Performance issues (memory leaks, fragmentation, poorly sized heap, finalization, …)
- What do we wish to achieve from Java Heap Analysis ?
  - Optimal JVM performance characteristics, time in GC, heap occupancy
  - Optimal application performance
  - Problem and cause identification
- What tools and techniques are available ?
  - VerboseGC visualisation using GCCollector
  - Java Heap Analysis using HeapAnalyzer
  - Other tools, 3rd party profilers and documentation

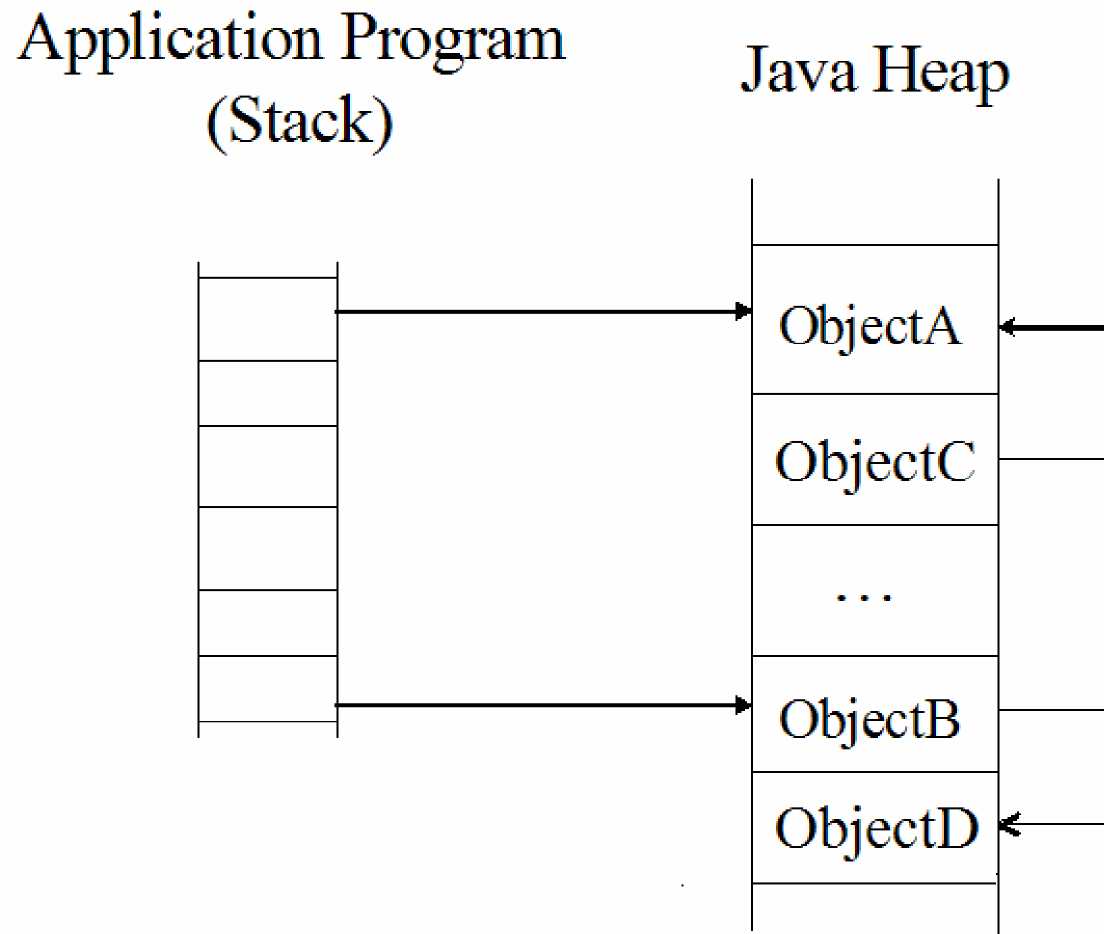
© 2005 IBM Corporation

The (IBM) JVM Storage Model

- The JVM storage requirements are bound by the size of addressable storage (31, 32, 64 bit) and are competed for by other applications and the operating system
- The JVM is allocated a contiguous chunk of storage on initialisation, the Java Heap, which is used by the JVM to support the Java application's storage requirements
- The JVM uses the native heap for other runtime storage requirements
- This presentation focuses on profiling the Java Heap

© 2005 IBM Corporation

The Java Heap



© 2005 IBM Corporation

The (IBM) JVM Garbage Collector

- The JVM allocates application objects on the Java Heap until the Java Heap is exhausted. The JVM thread experiencing the 'Allocation Failure' then executes a Garbage Collection.
- The purpose of a Garbage Collection is to identify Java Heap storage which is no longer being used by the Java application and so is available for reuse by the JVM.
- The IBM JVM uses a conservative stop-the-world Mark-Sweep-Compact Collector

© 2005 IBM Corporation

Mark

- All live objects are marked .
- Live objects are objects referenced by the stack or by other objects.
- Objects that are referenced by the stack are marked and then objects that they reference are marked recursively.
- Everything else in the heap can be reused.

© 2005 IBM Corporation

Sweep

- Sweep identifies the chunks of memory that are no longer reference and creates a links between them to form the “free list”.
- The free list is used by the JVM to find the next available chunk of Java heap when a allocation request is made.
- Chunks that are smaller than 512bytes are not added to this list and are called “dark matter”
- “Dark matter” is recovered when the objects next to them become free.

© 2005 IBM Corporation

Compact

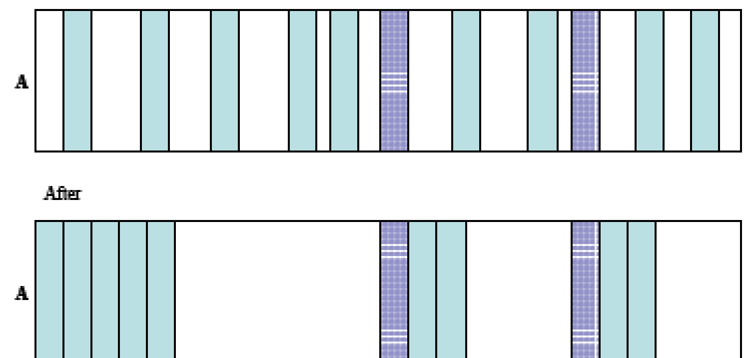
- Compaction does not occur for every GC cycle

- Compaction removes the spaces that are between the objects

- Dosed objects are objects referenced by the stack can not be moved.
- Pinned objects are objects referenced by JNI can not be moved.

Heap before and

after compaction.



- Incremental compaction only compacts a small amount of the heap each GC and hence spreads the load of compaction.

© 2005 IBM Corporation

Performance related issues

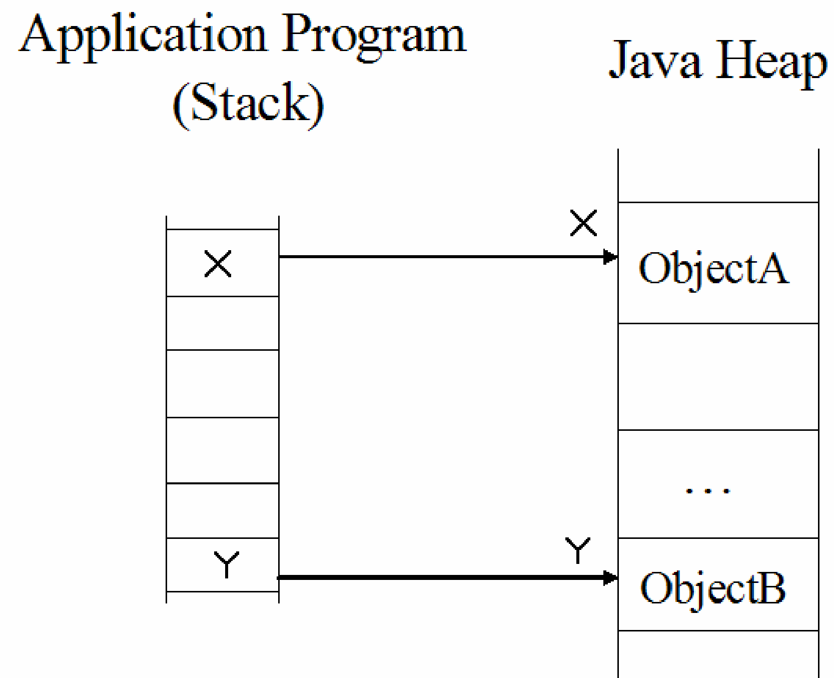
- Conservative collection
- Concurrent mark
- Finalizers
- Soft References

- Verbosegc record

© 2005 IBM Corporation

Conservative collection

- All ‘values’ on the stack which could be interpreted as objects references are.



© 2005 IBM Corporation

Mostly Concurrent GC (1)

- This technology was developed for customers who wish to reduce their pause time
- The principal component of a GC pause time is the mark phase
- In the Mostly Concurrent GC, introduced in 131, the mark phase is performed before stop the world
- Reducing pause times by up to 80%
  www.haifa.il.ibm.com/projects/systems/rs/gc.html
- Enabled using –Xgcpolicy:optavgpause
- How can this work ? …

© 2005 IBM Corporation

Mostly Concurrent GC (2)

- Marking is performed by mutator threads as an ‘tax’ on allocation requests
- Background marking threads use spare CPU cycles
- An adaptive algorithm attempts to start the concurrent mark so that its completion is synchronised with an allocation failure
- Start marking too early and the mark information is obsolete, too late and the mark needs to be done during stop the world
- But the mark information will be incorrect because the Java Heap changes until stop the world …

© 2005 IBM Corporation

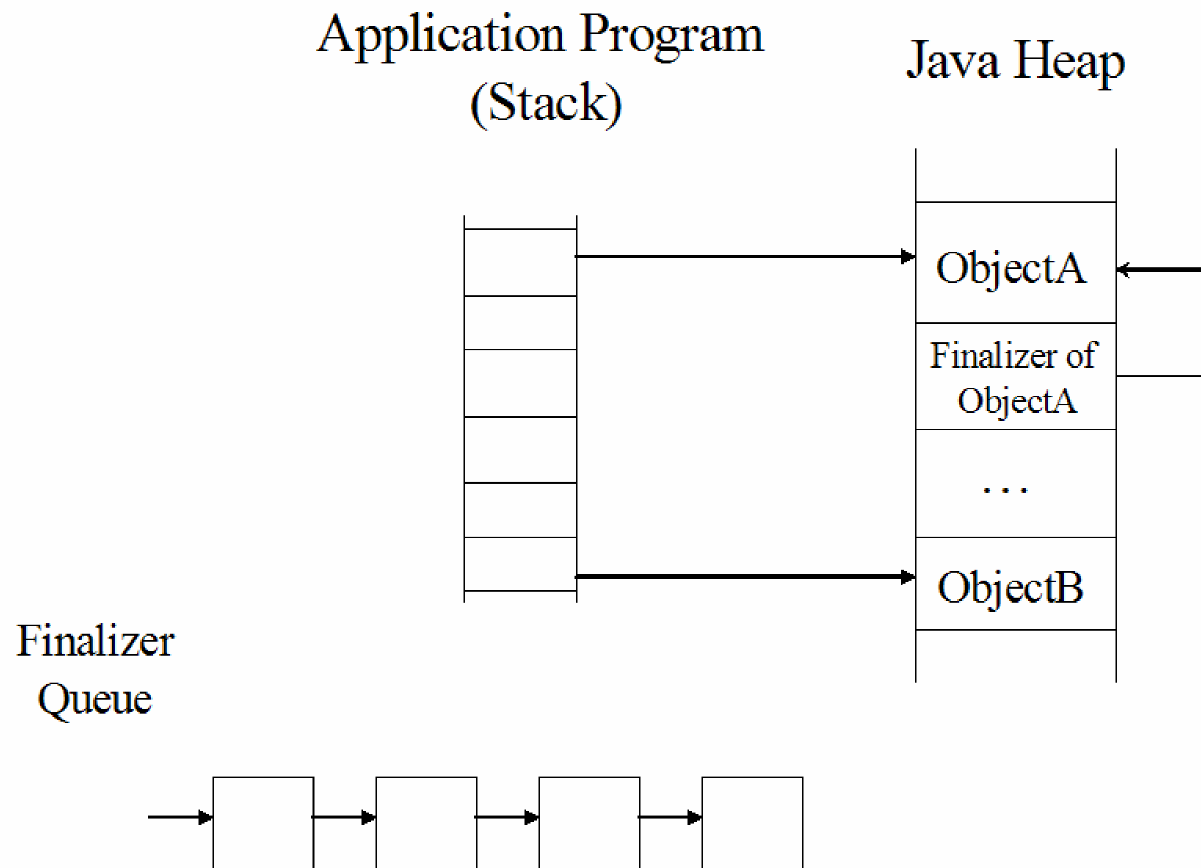
Mostly Concurrent GC (3)

- The Java Heap is divided into ‘cards’
- When concurrent mark is active a write barrier will cause the card to be ‘dirtied’ if a reference is changed in the section of Java Heap represented by that card
- When the threads are suspended GC can use this information to retrace the sections of the heap which have an associated ‘dirty’ card
- Nothing comes for free … concurrent mark is done at the expense of throughput

© 2005 IBM Corporation

Finalizers

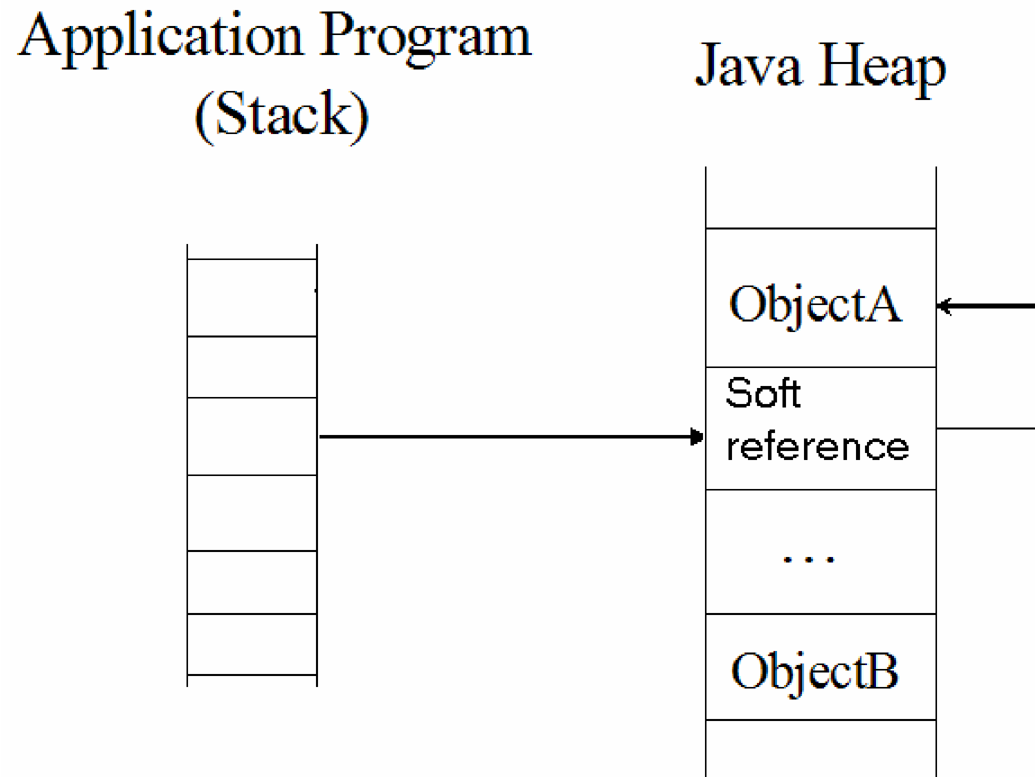
- An object's finalizer method is executed by the JVM prior to object collection



© 2005 IBM Corporation

Soft References (1)

- Reference objects enables the application to interact with the garbage Collector.
  http://java.sun.com/developer/technicalArticles/ALT/RefObj/



© 2005 IBM Corporation

Soft References (2)

- A web-based program may wish to cache objects such as images so that
  - (1) The object is only reclaimed if Java Heap storage becomes an issue
  - (2) If the user returns to the page the object can be used (if not reclaimed)
- The Garbage Collector collects a strongly reachable object when
  - It has aged, 32 GC cycles in the case of the IBM GC
  - Java Heap storage shortage
  - Soft Reference life is adaptive when heap expansion and shrinkage is enabled

© 2005 IBM Corporation

The verbosegc record

- Use the –verbosegc jvm option

<AF[238]: Allocation Failure. need 16400 bytes, 62145 ms since last AF>
<AF[238]: managing allocation failure, action=1 (8248600/686080200) (8040096/11155768)>
<GC(239): freeing class sun.reflect.GeneratedSerializationConstructorAccessor453(26731B40)>
<GC(239): freeing class sun.reflect.GeneratedSerializationConstructorAccessor456(26726B80)>
<GC(239): freeing class sun.reflect.GeneratedSerializationConstructorAccessor457(26726A50)>
<GC(239): unloaded and freed 3 classes>
<GC(239): mark stack overflow[241]>
<GC(239): mark stack overflow processing System Classes>
<GC(239): GC cycle started Mon Jun 20 10:47:40 2005
<GC(239): freed 402042360 bytes, 59% free (418331056/697235968), in 2657 ms>
<GC(239): mark: 2226 ms, sweep: 98 ms, compact: 333 ms>
<GC(239): refs: soft 9 (age >= 32), weak 1, final 2506, phantom 0>
<GC(239): moved 93295 objects, 5030528 bytes, IC reason=16>
<AF[238]: completed in 2671 ms>

© 2005 IBM Corporation