

JavaOneSM

Sun's 2004 Worldwide Java Developer Conference™

Using jvmstat and visualgc to Solve Memory Management Problems

java.sun.com/javaone/sf

Wally Wedel

Sun Software Services

Brian Doherty

Sun Microsystems, Inc.



Analyze JVM™ Machine Memory Management Problems

Understanding memory management behavior

Learn how to use `jvmstat` and `visualgc` to analyze and solve Java HotSpot™ virtual machine and Java™ technology-based application memory management problems

Agenda

Monitoring overview

jvmstat overview

Designing for memory management

Memory management problems

“Tiger” update

Agenda

Monitoring overview

jvmstat overview

Designing for memory management

Memory management problems

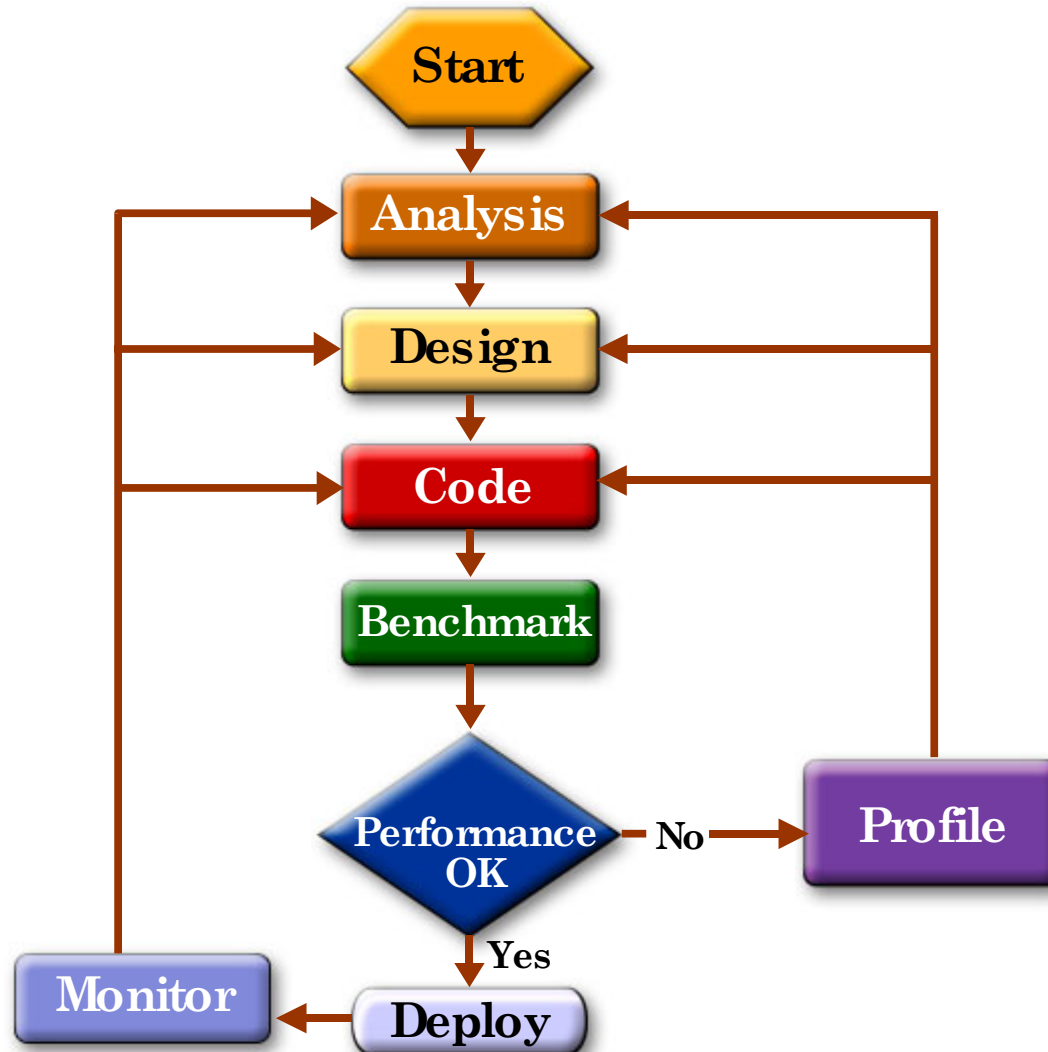
“Tiger” update

Monitoring and You

Monitoring goals

- Maintain and manage system health
 - Detection of performance problems
- Performance analysis
 - Find source of performance problems
- Problem resolution
 - System or application performance tuning
 - System resource improvements
 - Change design or implementation

Performance Management Process



Agenda

Monitoring overview

jvmstat overview

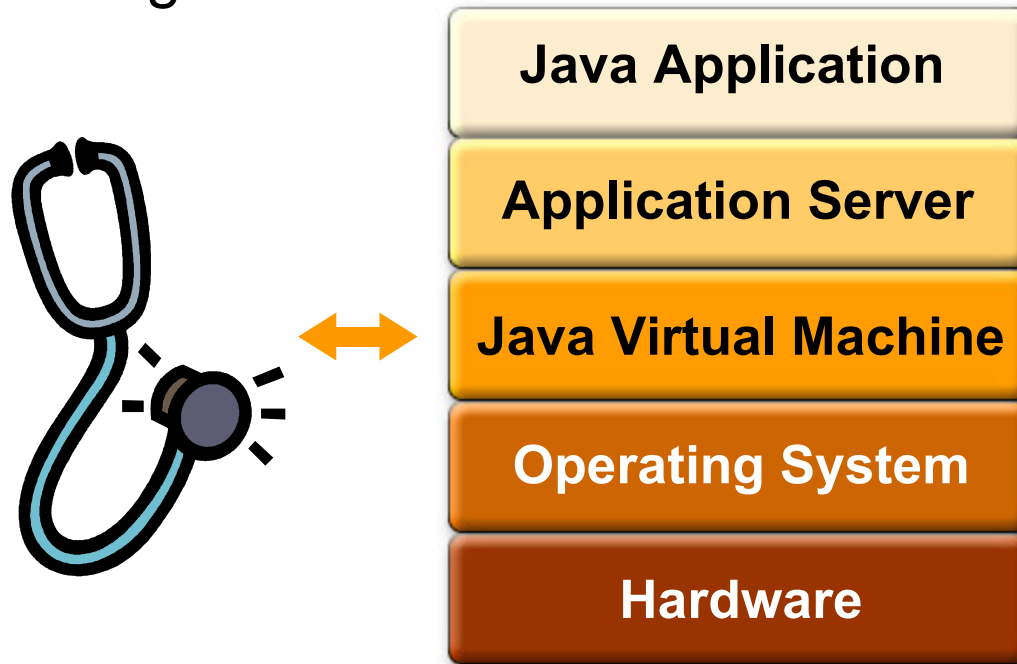
Well optimized applications

Memory management problems

“Tiger” update

What Is jvmstat?

- Light weight performance monitoring for the Java HotSpot virtual machine
 - Java HotSpot VM instrumentation
 - Monitoring tools



Always On

- No special command line options
 - No script modifications
 - No app server configuration modifications
- No application restarts
 - No down time
- No excuses
 - All Java HotSpot virtual machine and J2SE™ platform functional, regression, and performance tests run with instrumentation turned on

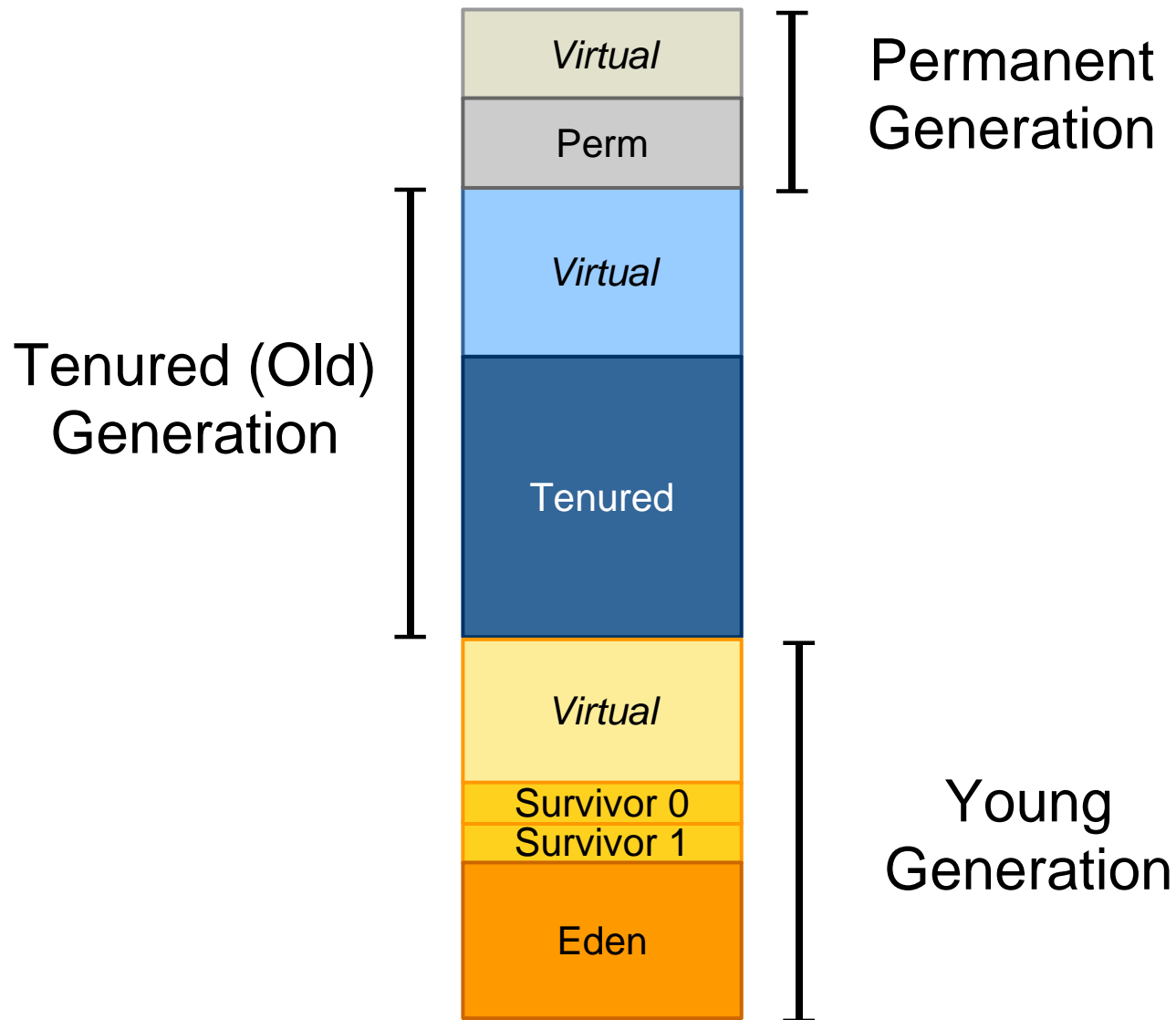
jvmsat Tools

- `jvmps`
 - Java process listing tool
- `jvmsat`
 - Command line statistics logging tool
- `visualgc`
 - JFC/Swing based tool for visualizing the operations of the Java HotSpot VM
- `perfagent`
 - RMI server for remote monitoring

Instrumentation

- Configuration
 - Classpath, path, library paths, etc.
 - Basic Java language properties
 - JVM software arguments, main args
- Class loader
- JIT compiler
- Garbage collector
 - Space sizes, utilization
 - Collection count, times

Generational Garbage Collected Heap



Demo

visualgc demo



Agenda

Monitoring overview

jvmstat overview

Designing for memory management

Memory management problems

“Tiger” update

Design for Memory Management

Don't second guess your garbage collector!

- Don't avoid allocation of transient objects
 - Object creation is fast
 - Thread local
 - JIT optimized
 - Minor GC is fast and efficient
 - Minor GC can be multi-threaded
- Minimize long-lived objects
 - Eventually promote to Old Gen
 - Major (Full) GC takes longer than Minor GC
 - Avoid object caching
 - Move infrequently accessed long-lived objects to persistent storage

Design for Memory Management (Cont.)

- Avoid calling `System.gc()`
 - Results in Full GC event
 - Advisory only
- Avoid frequent allocation of *large* transient objects
 - Result in direct allocation in Old Gen
- Classes are objects too
 - Allocated in Perm Gen
 - Requires Major (Full) GC
 - Well-designed class loads and unloads
 - Watch out for dynamically generated JSP™ pages

Design for Memory Management (Cont.)

- Avoid finalized objects
 - Requires additional GC work
 - Promptness not guaranteed
 - Use `java.lang.ref` classes instead

Typical Memory Management Tuning

Even well-behaved applications may need tuning

- Regular application pauses
 - RMI DGC activity or System.gc() calls
 - Tune RMI DGC period
- Premature Full GC
 - -XX:NewRatio (NewSize/MaxNewSize) too large
- Long application pauses
 - Caused by long full garbage collections
- Premature promotion
 - Undersized Eden and Survivor spaces
- Poor throughput
 - Caused by frequent garbage collections

Agenda

Monitoring overview

jvmstat overview

Designing for memory management

Memory management problems

“Tiger” update

Application Design Problems

- Memory leaks
 - Unproductive Full GCs leading to OOM error
- Too many long lived objects—object caches
 - Frequent, long Full GC events
 - Poor throughput
- Frequent allocation of large objects
- Overriding `Object.finalize()`
- Excessive class loading and unloading
 - Full GC required to unload classes
 - Avoid use of `-Xnoclassgc`

Analyzing Memory Management

- In the following demo:
 - A small application
 - Contrived memory management problems
 - Premature promotion causing Major GC's rather than Minor GC's
 - Out of memory problem due to Perm Gen exhaustion
 - Out of memory problem due to Old Gen exhaustion caused by slow release of object graphs
 - Allocation of very large objects directly to Old Gen
 - Illustrate the problems using visualgc

Demo

Analysis Patterns in visualgc



Demo Summary

Solution strategies

- Premature promotion causing Major GC's rather than Minor GC's
 - Adjust java command parameters to allocate more heap to Young Gen (40% max recommended)
 - Resize Survivor Spaces to allow more objects to be retained in Young Gen
 - Modify application to produce fewer smaller objects that can be collected from Young Gen

Demo Summary (2)

Solution strategies

- Out of memory problem due to Perm Gen exhaustion
 - Enlarge Perm Gen space using `-XX:MaxPermGen`
 - Modify application to use fewer interned Strings or fewer JSP pages
 - Avoid use of `-Xnoclassgc`

Demo Summary (3)

Solution strategies

- Out of memory problem due to Old Gen exhaustion caused by slow release of object graphs
 - Modify application to release unused or unneeded object graphs
 - Avoid unnecessary object caching
 - Debug object leaks
 - Modify java command to use larger heap size
 - Use 64-bit JVM technology to enable heaps larger than 4GB

Agenda

Monitoring overview

jvmstat overview

Well optimized applications

Memory management problems

“Tiger” update

“Tiger” Update

jvmsat technologies

- Subset of jvmsat tools
 - jps – jvmps
 - jstat – jvmsat and jsnap
 - jstatd – perfagent
- Unsupported demo tools in SDK only
- Backward compatible with Java HotSpot VM 1.4.1 and 1.4.2
 - Monitor previous versions with “Tiger” SDK
- visualgc available as a separate download
 - <http://www.sun.com/developers/coolstuff>

“Tiger” Update

JSR 174—JVM software monitoring and management

- New `java.lang.management` APIs
 - Monitoring and management interfaces
 - JMX™ API-based
 - Memory threshold event notification API
- New `java.lang.instrument` APIs
 - Byte-code instrumentation support
- `jconsole` demo application
 - Packaged with SDK only

“Tiger” Update

Serviceability tools

- Demo Tools for debugging and monitoring
 - jstack—java stack trace
 - Native and java frames with -m option
 - jmap—java memory map
 - Heap configuration, object histogram
 - jinfo—JVM software information
 - JVM machine flag settings, system properties
 - jsadebugd—remote access
- Heavier weight than jvmstat
 - Stops process momentarily
- Solaris™ Operating System and Linux only

“Tiger” Update

“Smart tuning”

- Specify pause time, throughput, and memory utilization goals
- Garbage collector monitors various memory allocation and collection patterns
- Dynamically resizes GC spaces
 - Optimize GC performance based on measured memory allocation behavior
 - Server class systems (-XX:+UseParallelGC)
 - Young Gen resizing initially
 - Inter-generation resizing in future
- Heap tuning becomes largely unnecessary for most applications

Summary

- visualgc offers immediate analysis
- jvmstat offers later analysis
- Strive for good memory utilization
- Recognize indicators of memory utilization problems
- Use java command line parameters to alleviate problems

Call to Action

Understand your Java application's memory management characteristics

Monitor your production applications using lightweight monitoring tools. Tune the Java HotSpot VM and improve your Java technology-based application's memory management characteristics to achieve optimal system performance.

For More Information

- Sessions:
 - TS-1216—Choices and Trade-Offs in Garbage Collection in the Java HotSpot™ VM
 - TS-3065—Observability Architecture in the J2SE™ 1.5 Platform
 - TS-2861—Monitoring and Management of the Java™ 2 Platform, Standard Edition (J2SE™)
- BOF
 - BOF-1217—Performance of J2SE, J2EE, Web Server, Web Services, Portal
- Hands-On Lab
 - 7232—Java™ Technology-Based Application Performance Analysis
- Visit our website
 - <http://java.sun.com/performance/>

Q&A

Wally Wedel, Brian Doherty



JavaOne™

Sun's 2004 Worldwide Java Developer Conference™

Using jvmstat and visualgc to Solve Memory Management Problems

java.sun.com/javaone/sf

Wally Wedel

Sun Software Services

Brian Doherty

Sun Microsystems, Inc.

