

Java Knobs: How To Change Performance of the Sun JVM

Simon Ritter
Technology Evangelist

Agenda

- Options for the Sun JVM
- Garbage collection
- Threading
- Instruction execution
- Miscellaneous stuff
- Summary and resources

Sun JVM Options

- Standard options
 - All platforms
- -X options
 - Not all platforms
- -XX options
 - Not all platforms
 - May need additional privileges to use

A Multitude of Options

JRE Version	Number of -XX Options	Options Added	Options Removed
1.4	159	--	--
1.4.1	224	70	5
1.4.2	260	44	8
5.0	343	93	10
6	427	102	35

Objects Need Storage Space

- Age old problems
 - How to allocate space efficiently
 - How to reclaim unused space (garbage) efficiently and reliably
- C (malloc and free)
- C++ (new and delete)
- Java (new and Garbage Collection)

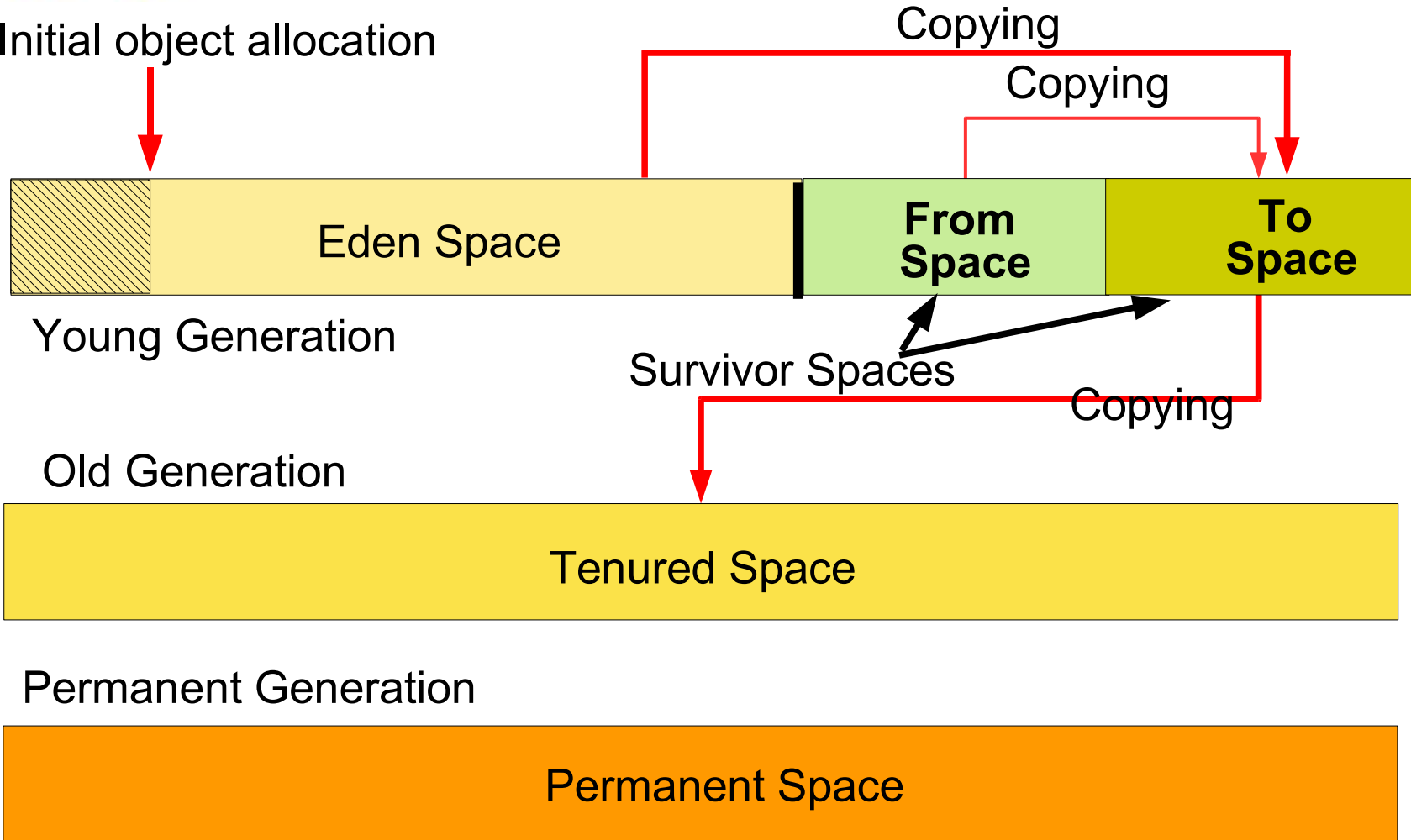
Object Realities

- Most objects are very short lived
 - 80-98% of all newly allocated objects die within a few million instructions
 - 80-98% of all newly allocated objects die before another megabyte has been allocated
- This impacts heavily on choices for GC algorithms



HotSpot VM Heap Layout

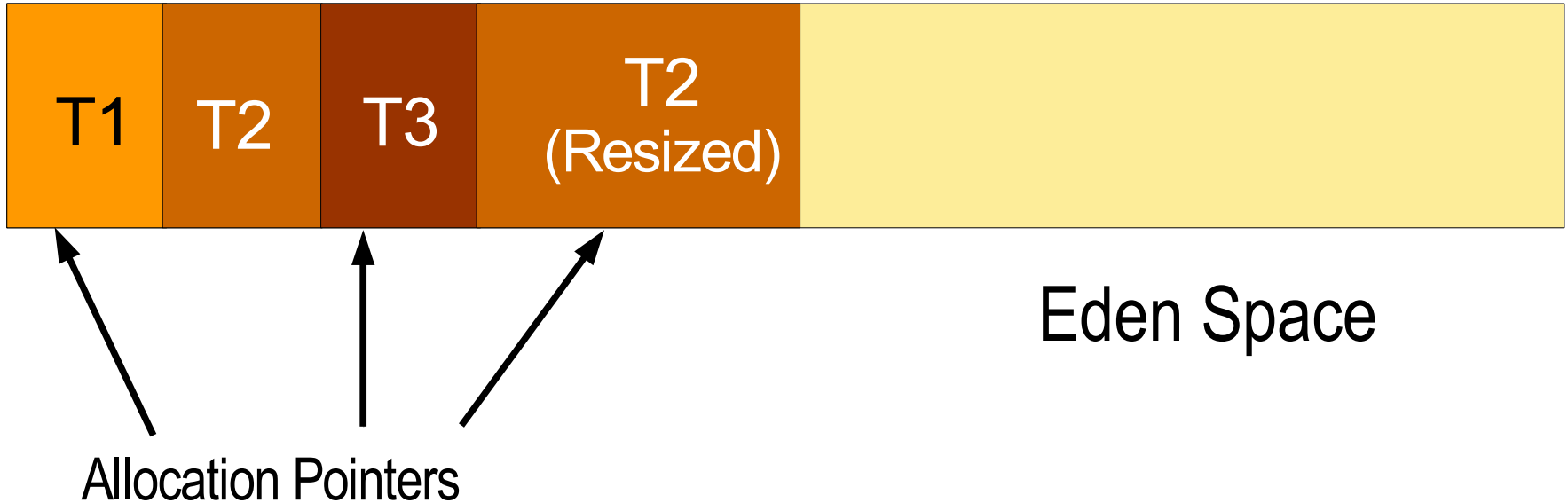
Initial object allocation



jug Eden: Thread Local Allocation

- Problem: multiple threads creating objects
 - All trying to access eden simultaneously
 - Multiple CPU machine: contention
- Solution: Thread local allocation
 - `-XX:+UseTLAB (false)`

Thread Local Allocation



- XX:TLABSize=*size-in-bytes* (256 Kb)
- XX:ResizeTLAB (false)

jug Parallel Copy Collector

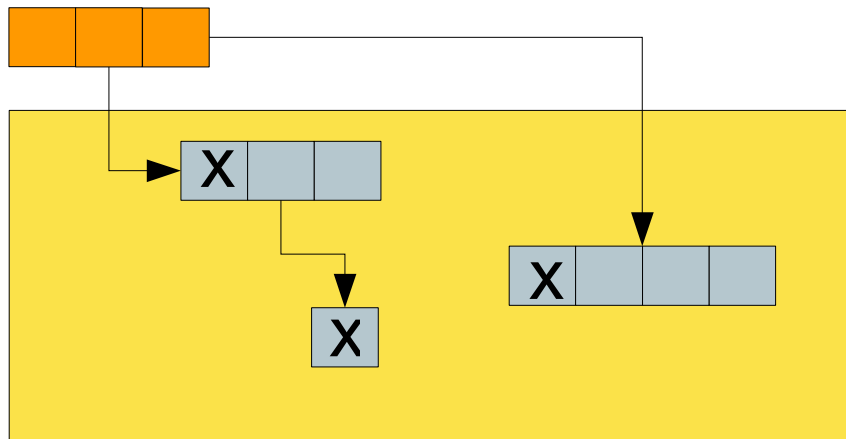
- Young generation
- Allocates as many threads as CPUs
 - Algorithm optimized to minimize contention
- Maximize work throughput
 - Work stealing
- Potential locality of reference issue
 - Each thread has separate destination in tenured space

Parallel Copy Collector

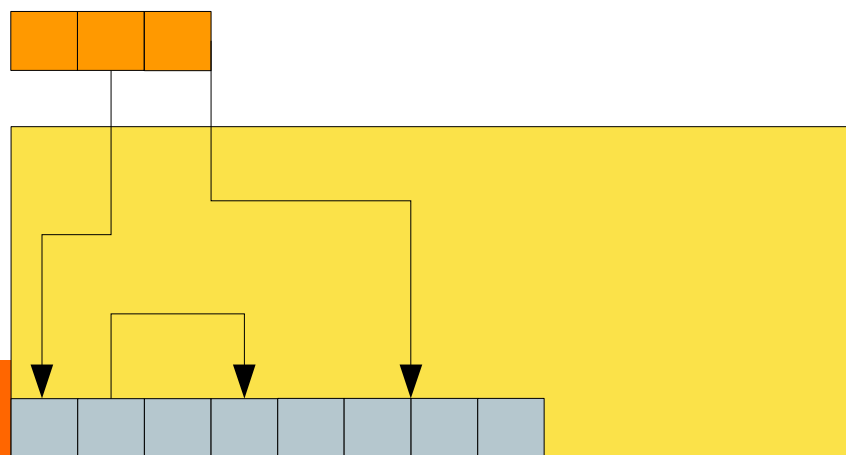
- `-XX:+UseParNewGC (false)`
 - Default copy collector will be used on single CPU machines
- `-XX:ParallelGCThreads=n`
 - Default is number of CPUs
 - Tune down for multi-app machines
 - Tune down for machines with more than 8 CPUs



Mark Sweep Compact GC (Full GC) Old Generation



Before



After

jug Mark Sweep Compact GC

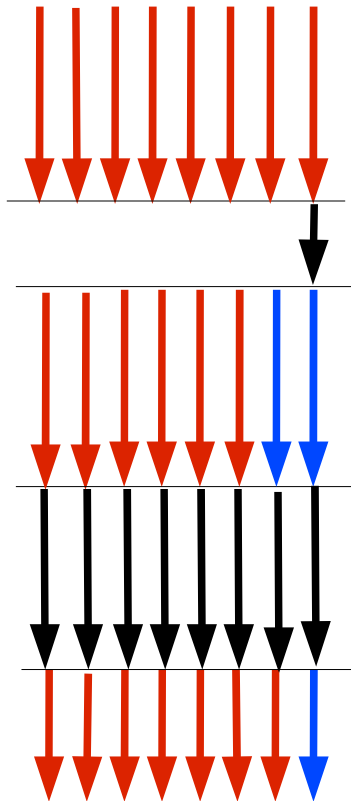
- Eliminates fragmentation issue of Mark-Sweep
- Allocation is stack-based
- Order of objects maintained
 - No locality of reference issues
- Requires multiple passes to complete
 - Mark live objects
 - Compute new location
 - Update pointers
- Can result in 'significant' pauses

Parallel Old Generation GC

- Available from JRE 5.0_u06
- `-XX:+UseParallelOldGC`
- Uses mark-sweep-compact algorithm
- Not all phases are currently parallel
- Can be good for UltraSPARC T1
 - CMS is single threaded
 - CMS cannot keep up with mutator threads
 - Use parallel old, assuming pause times are acceptable



Low Pause Collector Concurrent Mark Sweep



Application Threads

Stop-the-world initial mark phase

Concurrent mark phase

Concurrent pre-clean phase

Stop-the-world re-mark phase

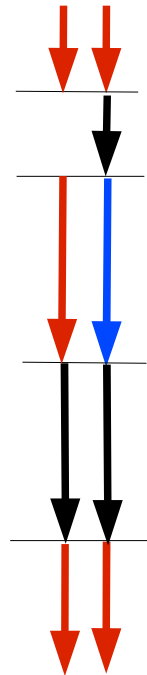
Concurrent sweep phase

Concurrent reset phase

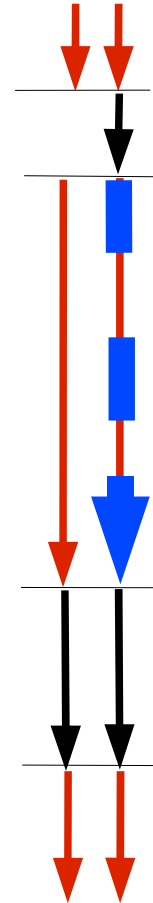
Concurrent Mark Sweep Collector

- `-XX:+UseConcMarkSweepGC`
- Concurrent marking phase parallel in JDK 6
 - `-XX:ParallelCMSThreads=n`
 - Default is $\frac{1}{4}$ of available CPUs
- Scheduling of collection handled by GC
 - Based on statistics in JVM
 - Or Occupancy level of tenured generation
 - `-XX:CMSInitiatingOccupancyFraction`

Incremental CMS



2 CPU Problem



Marking work
interleaved with
application work

Incremental CMS

- `-XX:+CMSIncrementalMode` (off)
- `-XX:CMSIncrementalDutyCycle=n%` (50)
- `-XX:+CMSIncrementalPacing` (on)
- `-XX:CMSIncrementalDutyCycleMin=n%` (10)
- DutyCycle of 10 and DutyCycleMin of 0 can help certain applications



CMS Throughput Problem



Application threads

GC

Throughput Collector

Parallel Scavenge

- Stop-the-world
- Similar to parallel-copy collector
- Aimed at large young spaces
 - 10Gb or larger
- Scales well with more CPUs
- Adaptive tuning policy
 - Survivor ratio
- Promotion undo to prevent out of memory

Throughput Collector

- Available from JDK 1.4.1
- `-XX:+UseParallelGC` (off)
- `-XX:ParallelGCThreads=n` (number of CPUs)
 - Control number of threads
- `-XX:+UseAdaptiveSizePolicy` (on)
 - Automatically sizes the young generation and selects optimum survivor ratio

JVM Ergonomics: Throughput Collector

- Ergonomics enables the following:
 - Throughput garbage collector and Adaptive Sizing
(-XX :+UseParallelGC -XX:+UseAdaptiveSizePolicy)
 - Initial heap size of 1/64 of physical memory up to 1Gbyte
 - Maximum heap size of 1/4 of physical memory up to 1Gb
 - Server runtime compiler (-server)
- To enable server ergonomics on 32-bit Windows, use the following flags:
 - -server -Xmx1g -XX:+UseParallelGC
 - Varying the heap size.

Using JVM Ergonomics

- Maximum pause time goal
 - `-XX:MaxGCPauseMillis=n`
 - This is a hint, not a guarantee
 - GC will adjust parameters to try and meet goal
 - Can adversely effect application throughput
- Throughput goal
 - `-XX:GCTimeRatio=n`
 - GC Time : Application time = $1 / (1 + n)$
 - e.g. `-XX:GCTimeRatio=19` (5% of time in GC)
- Footprint goal
 - Only considered if first two goals are met

Heap Tuning Beyond Ergonomics

- Increase heap size
 - Ergonomics chooses up to 1GB, some applications need more memory for high performance
 - -Xms3g -Xmx3g
- Increase the size of the young generation
 - Generally: $\frac{1}{4}$ to $\frac{1}{2}$ the overall heap size
 - Sizing above $\frac{1}{2}$ the overall heap size is supported
 - Only makes sense with throughput collector

Thread Priority Policy

- `-XX:ThreadPriorityPolicy=[0|1]`
 - 0: Use normal thread priorities
 - 1: Aggressive thread priorities
 - Java threads can use all native OS thread priorities
 - Can have serious implications on system performance

Thread Options For Solaris

- Java thread priorities now map to Solaris threads
 - Java threads now compete/yield correctly to Solaris threads
- -XX:+UseBoundThreads
- -XX:+UseLWPSynchronization (SPARC only)
- -XX:+AdjustConcurrency
- -XX:+DontYieldALot
- -XX:+ConvertSleepToYield
- -XX:+ConvertYieldToSleep (off)
- -XX:MinSleepInterval (1)

Thread Options For Linux

- `-XX:+UseSpinning`
 - Emulate a spin-mutex lock
- `-XX:PreBlockSpin=n (100)`
 - How long to spin before reverting to a mutex
- `-XX:+DontYieldALot`
 - Ignore calls to `yield()`
 - Calls to `yield()` on linux revokes any remaining time quanta as well as cached data
 - Test. Wide variety of results on benchmarks

Biased Locking

- Available from JRE 5.0_u06
- `-XX:+UseBiasedLocking` (off)
- `-XX:BiasedLockingBulkRebiasThreshold=n` (20)
- `-XX:BiasedLockingBulkRevokeThreshold=n` (40)
- `-XX:+TraceBiasedLocking` (for debugging)
- Good for applications with lots of uncontended synchronization
- Can have negative impact for certain locking patterns

Tiered Compilation

- New in JRE 6
- HotSpot has two compilers, -client & -server
- JVM starts with client compiler
 - Fast warmup
- Switches to server compiler
 - Better optimisation
- `-XX:+TieredCompilation`

Other Compilation Options

- `-XX:CompileThreshold=n` (5000)
 - How many times a method is called before compilation
- `-XX:DontCompileHugeMethods` (on)
 - Don't compile methods over 8000 bytecodes
- `-XX:MaxInlineSize=n` (35)
 - Max size of a method that will be in-lined
- `-XX:+PrintCompilation`
 - Information about methods compiled by JVM

jug Large Page Sizes

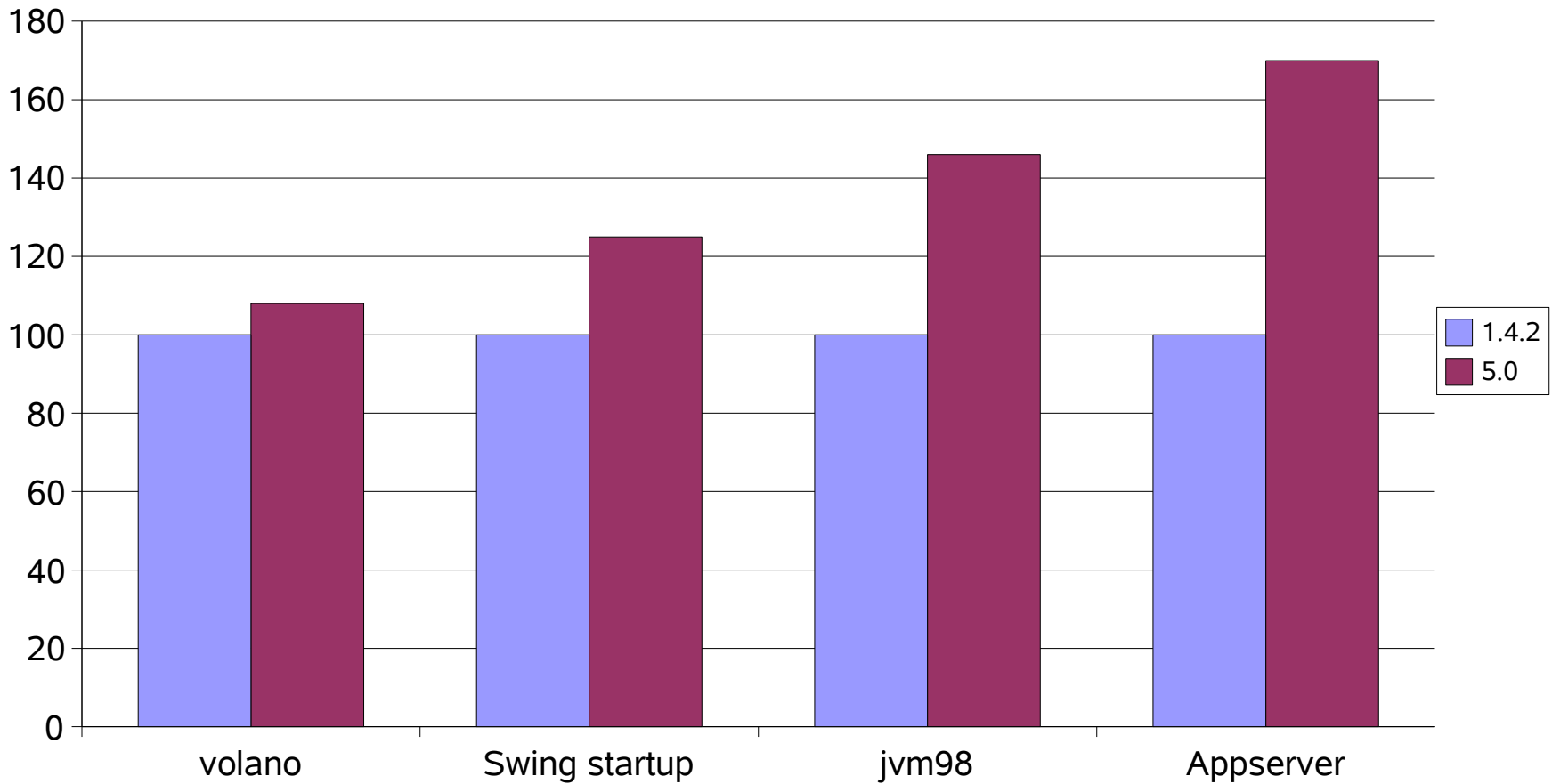
- `-XX:+UseLargePages`
 - Cross platform (only on by default for Solaris)
 - Improves utilisation of TLB
 - Kernel support in Linux 2.6 and Windows 2003 server
 - 4m SPARC/x86 (SPARC supports multiple page sizes)
 - 2m x86_64
 - 256m supported for UltraSPARC T1
- `-XX:LargePageSizeInBytes=n`
 - Set to 2m for AMD Opteron systems

Random Things

- Consider disabling explicit GC
 - `-XX:+DisableExplicitGC`
- Increase size of permanent generation
 - If lots of classes loaded at start
 - Can improve startup time
 - e.g. NetBeans sets permanent heap size to 20MB
- `-XX:+AggressiveOpt`
 - Go fast meta-option (may change between releases)
- `-XX:+Use486InstrOnly`
 - Go slow option

Always Use the Latest JVM

Solaris Sparc



DTrace and Java

- DTrace: dynamic tracing in Solaris 10
 - Innovation in tracing
 - 35,000 probes in the kernel
 - Zero performance impact when not enabled
- Probes define points of instrumentation
- DTrace scripts define how to handle a probe 'firing'
 - One or more clauses for one or more probes
 - Syntax like C and AWK
 - No flow control - use predicates

jug DVM Provider

- Available probes
 - vm-init, vm-death
 - thread-start, thread-end
 - class-load, class-unload
 - gc-start, gc-finish, gc-stats
 - object-alloc, object-free
 - monitor-contended-enter, monitor-contended-entered
 - monitor-wait, monitor-waited
 - method-entry, method-return
 - exception-throw, exception-catch
- jstack() built-in method

jug DTrace and Java

- Need to be root to run DTrace
 - Or have `dtrace_kernel`, `dtrace_proc`, `dtrace_user` privileges
- JDK 1.4.2
 - Use JVMPI
 - Install libs, start JVM with `-Xrundvmpi:all`
- JDK 5.0
 - Use JVMTI
 - Install libs, start JVM with `-Xrundvmti:all`
- JDK 6
 - `dvm` probes built in, no special command line options

jug DTrace Sample Scripts

- Object allocation/free statistics

```
dvm$target:::object-alloc
{
    @alloc[copyinstr(arg0)] = count();
}
dvm$target:::object-free
{
    @free[copyinstr(arg0)] = count();
}
END
{
    printa(@alloc);
    printa(@fee);
}
```

DTrace Sample Scripts

- Make results periodic and limit to top ten

```
tick-10s
{
    trunc(@alloc, 10);
    printa(@alloc);
    trunc(@alloc);
    trunc(@free, 10);
    printa(@free);
    trunc(@free);
}
```

Summary

- Many factors affect Java performance
 - Application code
 - App server settings (Java EE only)
 - JVM settings
- Understanding the JVM is essential to improving performance
- You **MUST** profile your application!
- If possible, always upgrade to the latest version of the JVM

jug Resources

- java.sun.com/j2se
- java.sun.com/products/hotspot
- profiler.netbeans.org
- java.sun.com/j2se/1.5.0/docs/guide/jvmti
- www.sun.com/bigadmin/content/dtrace/
- solaris10-dtrace-vm-agents.dev.java.net

Thank you
simon.ritter@sun.com