IBM Java: JVM Tuning

Joran Siu – IBM SDK for Java Development

Session 16761

Tues, March 3, 2015: 1:45 PM-2:45 PM

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence**.

Trademarks, Copyrights, Disclaimers

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Java Road Map

Language Updates

Java 5.0
- New Language features:
  - Autoboxing
  - Enumerated types
  - Generics
  - Metadata

Java 6.0
- Performance Improvements
- Client WebServices Support

Java 7.0
- Support for dynamic languages
- Improve ease of use for SWING
- New IO APIs (NIO2)
- Java persistence API
- JMX 2.x and WS connection for JMX agents
- Language Changes

Java 8.0**
- Language improvements
- Closures for simplified fork/join

5.0 | EE 5 | 6.0 | EE 6.x | 7.0

IBM Java 8 (J9 R28)
- Improvements in
  - Performance
  - RAS
  - Monitoring
- z13™ Exploitation
  - SIMD
  - SMT
  - Crypto acceleration

SE8 >= 20 platforms

04 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015

WAS 6.0 | SE 5.0 18 platforms | WAS 6.1 | SE 6.0 20 platforms | WAS 7.0 | SE601/ 7.x >= 20 platforms | WAS 8.5 | SE601/ 7.x >= 20 platforms

IBM Java Runtimes

IBM Java 5.0 (J9 R23)
- Improved performance
  - Generational Garbage Collector
  - Shared classes support
  - New J9 Virtual Machine
  - New Testarossa JIT technology
- First Failure Data Capture
- Common runtime technology
  - ME, SE, EE

IBM Java 6.0 (J9 R24)
- Improvements in
  - Performance
  - Serviceability tooling
  - Class Sharing
- XML parser improvements
- z10™ Exploitation
  - DFP exploitation for BigDecimal
  - Large Pages
  - New ISA features

IBM Java 6.0.1/Java 7 (J9 R26)
- Improvements in
  - Performance
  - GC Technology
- z196™ Exploitation
  - OOO Pipeline
  - 70+ New Instructions
- JZOS/Security Enhanc

IBM Java 7 (J9 R26 SR3)
- Improvements in
  - Performance
- zEC12™ Exploitation
  - Transactional Exec
  - Flash 1Meg LPs
  - 2G large pages
  - Hints/traps

IBM Java 7R1 (J9 R27)
- Improvements in
  - Performance
  - RAS
  - Monitoring
- zEC12™ Exploitation
  - zEDC for zip acceleration
  - SMC-R integration
  - Transactional Execution
  - Runtime instrumentation
  - Hints/traps
- Data Access Accelerator

**Timelines and deliveries are subject to change.

IBM SDK Java Tech. Edition, Version 8 (IBM Java 8)

- **New Java8 Language Features**
  - Lambdas, virtual extension methods
- **IBM z13 exploitation**
  - Vector exploitation and other new instructions
  - Instruction scheduling
- **General throughput improvements**
  - Up-to 17% better application throughput
  - Significant improvements to ORB
- **Improved crypto performance for IBMJCE**
  - Block ciphering, secure hashing and public key
    - Up-to 4x improvement to Public Key using ECC
    - CPACF instructions: AES, 3DES, SHA1, SHA2, etc
- **Significantly improved application ramp-up**
  - Up-to 50% less CPU to ramp-up to steady-state
  - Improved perf of ahead-of-time compiled code
- **Improved Monitoring**
  - JMX beans for precise CPU-time monitoring
- **Enhancements to JZOS Toolkit for Java batch**

IBM z13 – Taking Java Performance to the Next Level

Continued aggressive investment in Java on Z

Significant set of new hardware features tailored and co-designed with Java

Simultaneous Multi-Threading (SMT)

- *2x hardware threads/core for improved throughput*
- *Available on zIIPs and IFLs*

Single Instruction Multiple Data (SIMD)

- *Vector processing unit*
- *Accelerates loops and string operations*

Cryptographic Function (CPACF)

- Improved performance of crypto co-processors

New Instructions

- Packed Decimal ⇔ Decimal Floating Point
- Load Immediate on Condition
- Load Logical and Zero Rightmost Byte

New 5.0 GHz 8-Core Processor Chip

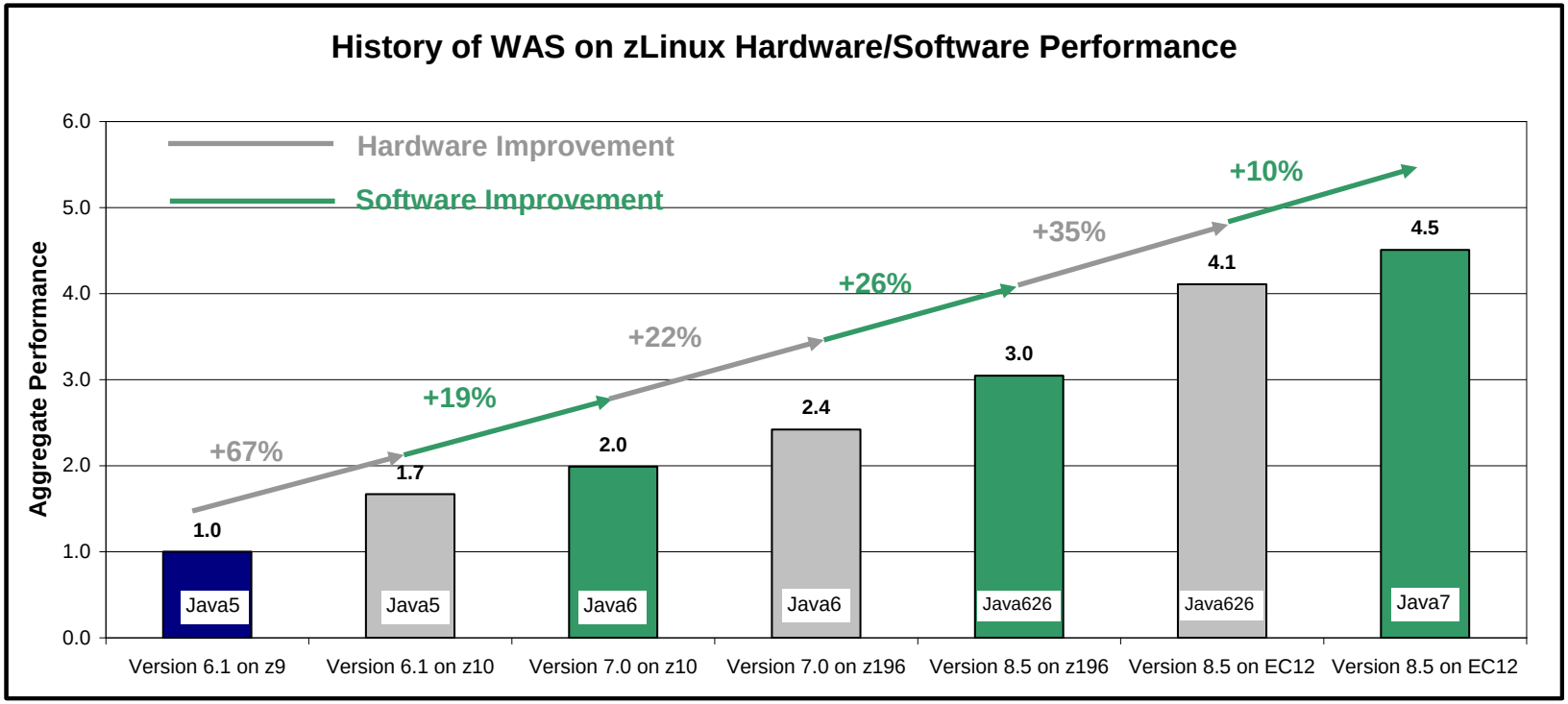
480Mb L4 cache to optimize for data serving

Up to **50%** improvement in throughput for generic applications

Up to **2X** improvement in throughput per core for security enabled applications

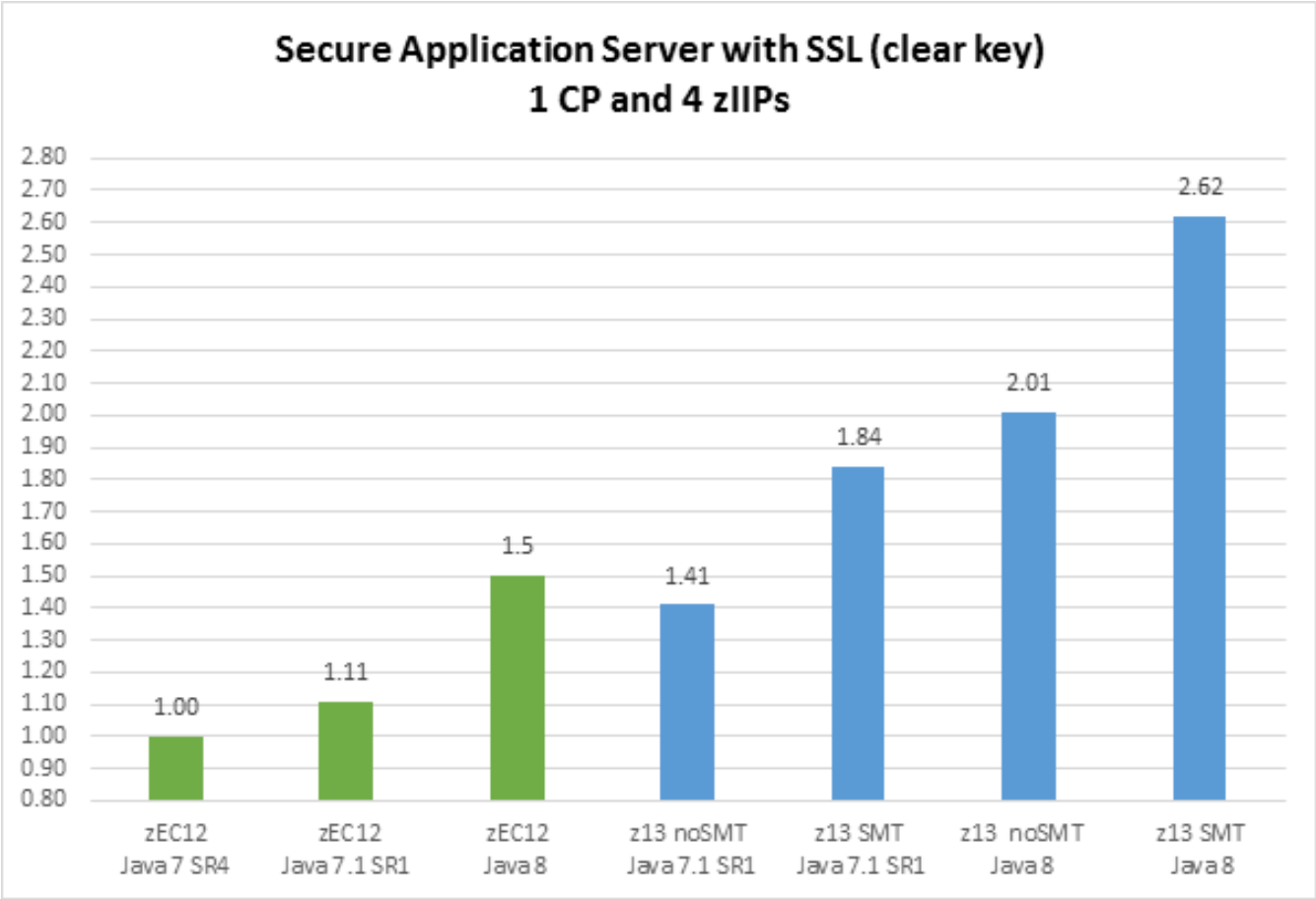
WebSphere Application Server for Linux on z

- Aggregate HW, SDK and WAS Improvement: WAS 6.1 (IBM Java 5) on z9 to WAS 8.5 (IBM Java 7) on zEC12



~4.5x aggregate hardware and software improvement comparing WAS 6.1 IBM Java5 on z9 to WAS 8.5 IBM Java7 on zEC12

Application Serving – SSL-Enabled DayTrader3.0



2.6x improvement in throughput with IBM Java 8 and IBM z13

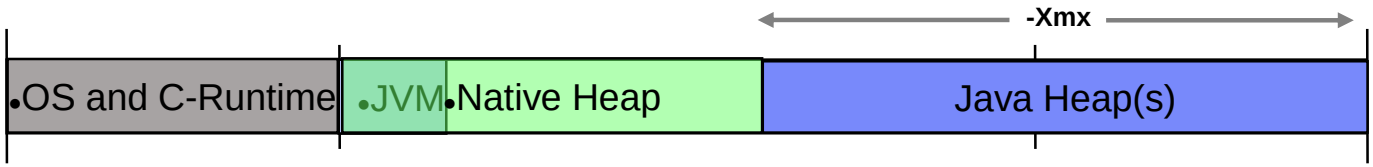
(Controlled measurement environment, results may vary)

Java Tuning: Agenda

- **IBM JVM(J9) Garbage Collector (GC)**
  - Overview
  - Policies
- IBM Monitoring and Diagnostic Tools
- Dynamic Compilations
  - Shared Classes + Ahead Of Time compilations
  - Runtime Instrumentation
- JNI Best Practices

Garbage Collector: Overview
Java Memory Usage

- Java is an Operating System (OS) process. Some memory is used by the OS and C-language runtime
- Area left over is termed the "User Space" and is divided into:
  - Java Virtual Machine (JVM) runtime
  - Java Heap(s)
  - Native heap



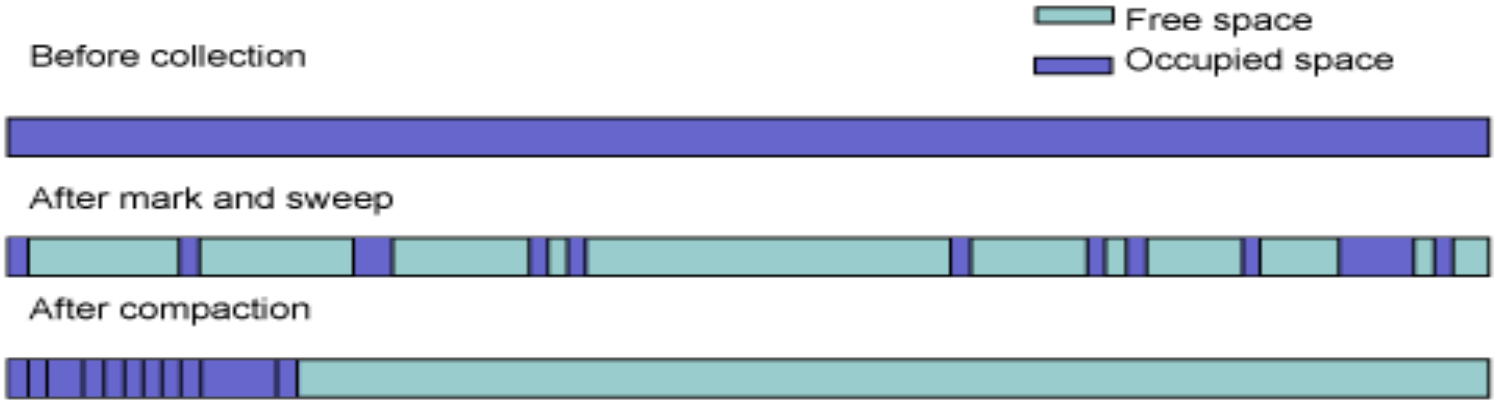
Garbage Collector: Overview

- The GC works on the Java Heap memory
- Responsible for allocation and freeing of:
  - Java objects, arrays and classes
- Allocates objects using a contiguous section of Java heap
- Ensures the object remains as long as it is in use (“live”)
- Determines “live-ness” based on a reference from another “live” object or from a known root
- Reclaims objects that are no longer referenced
- Ensures that any finalize method is run before the object is reclaimed

Garbage Collector: Overview

Two main GC technologies:

1. Mark Sweep Collector
   - Mark: Find all live objects
   - Sweep: Reclaim unused heap memory
   - Compact: Reduce fragmentation (optional)



2. Copy Collector

Garbage Collector: Overview

GC occurs under two scenarios:

1. An allocation failure:
   - Not enough contiguous memory available
2. A programmatically requested GC cycle:
   - Call to System.GC();

GC: Performance Effect

- GC affects application's performance:
  - Pause times (responsiveness/Consistency)
  - Throughput
  - Footprint
- Typical tradeoffs:
  - Pause time vs. Throughput
    - Tradeoff frequency and length of pauses vs. throughput
- Footprint vs. Frequency
  - Tradeoff smaller footprint vs. frequency of GC pauses/events

IBM J9 Garbage Collector Family

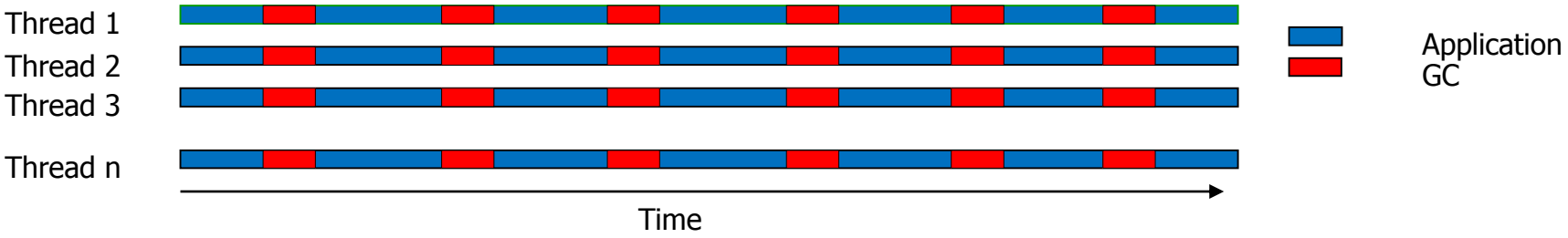
Why have many policies? Why not just the *best*?

- Cannot always dynamically determine what tradeoffs the user/application are willing to make
- Definition of a performance problem is user centric

Policy	Notes
optthruput	Default in Java 5, 6
optavgpause	
gencon	Default in Java 6.0.1, 7, 8
balanced	Added in Java 6.0.1, 7

IBM J9 Garbage Collector: optthruput

- Global Mark and Sweep Garbage Collection
- Uses “flat” heap
- Single stop-the-world (STW) phase
  - Application “pauses” while GC is done
- Parallel GC via use of “GC Helper Threads”
  - “ Parked” set of threads that wake to share GC work
  - Configurable using -Xgcthreads



Picture is only illustrative and doesn’t reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

IBM J9 Garbage Collector: optthruput Recommended Deployment Scenarios

- Best when:
  - Application is optimized for throughput rather than short GC pauses
  - Pause time problems are not evident
  - “Batch” type applications

- Was default in Java 5 and Java 6, but not in Java 7 and Java 8.

Tuning: Fixed vs. Variable Heap Size

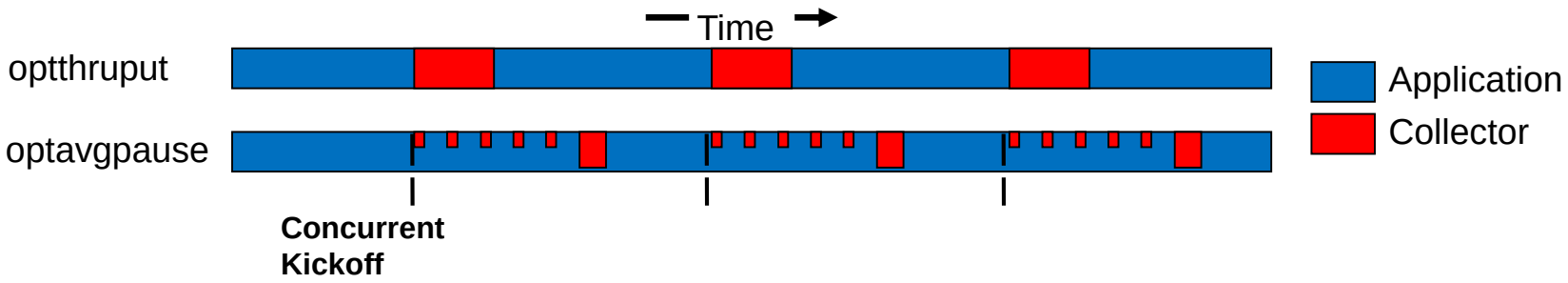
- Should the heap size be “fixed”? … depends on application

- Fixed heap size
  - *Minimum heap size* (-Xms) = *Maximum heap size* (-Xmx)
  - Best when memory usage is fairly constant and known

- Variable heap size
  - Set only -Xmx
  - GC adapts heap size
  - Best when memory usage varies over time or unknown
  - Provides flexibility and avoids OutOfMemoryErrors
  - Allows fine tuning of the heap usage

IBM J9 Garbage Collector: optavgpause

- Reduces pause time spent inside STW GC vs optthruput
- Results in more consistent pauses
- Slight overhead on application's throughput performance
- Uses **Concurrent** Marking and Sweeping
  - Carries out some STW work while application is running
- Policy focuses on responsiveness criteria



IBM J9 Garbage Collector: gencon

- Best of both worlds: Throughput and Small Pause Times
- Default policy in Java 6.0.1, Java 7 and Java 8
- Heap is split into two areas:
  - Objects created in the nursery
  - Objects that survive enough collections are promoted to tenured area
- Why a generational + concurrent solution?
  - For most workloads objects die young
    - Generational allows a better return on investment (less effort, better reward)
  - Reduce large pause times
    - Mitigates cost of object movement, and cache misses

IBM J9 Garbage Collector: gencon Motivation

- Most objects die young - focus on recently created objects
- Example: String concatenation

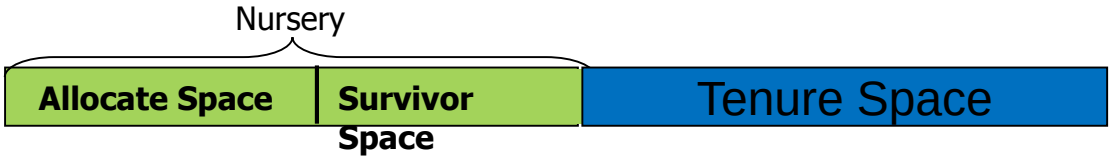
```
String str = new String ("String  ");
str += "Concatenated!";
```

  3 objects created!
  - String object, containing “String”
  - StringBuffer object, containing “String”, then appended with “Concatenated!”
  - String object, containing the result: “String Concatenated!”
- Two of those three objects are no longer required!
- Other examples: transactions in banking /commerce, DB, web page request, GUI functions

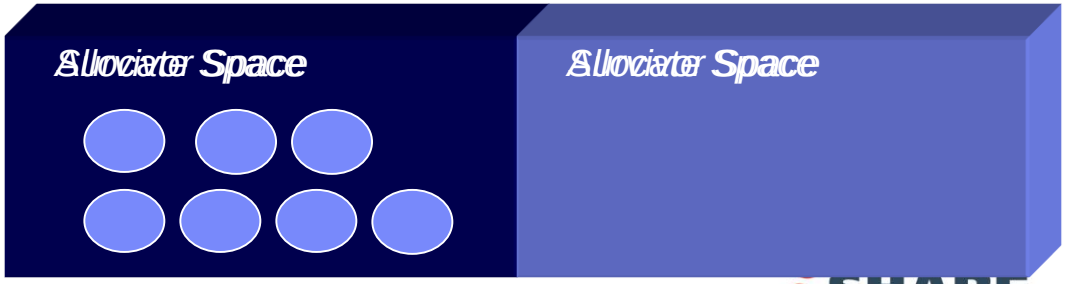
IBM J9 Garbage Collector: gencon Nursery Space Implementation

Nursery uses Copying GC :

- Nursery is split into two spaces:
  - **Allocate space**: used for new allocations and objects that survived previous collections
  - **Survivor space**: used for objects surviving this collection



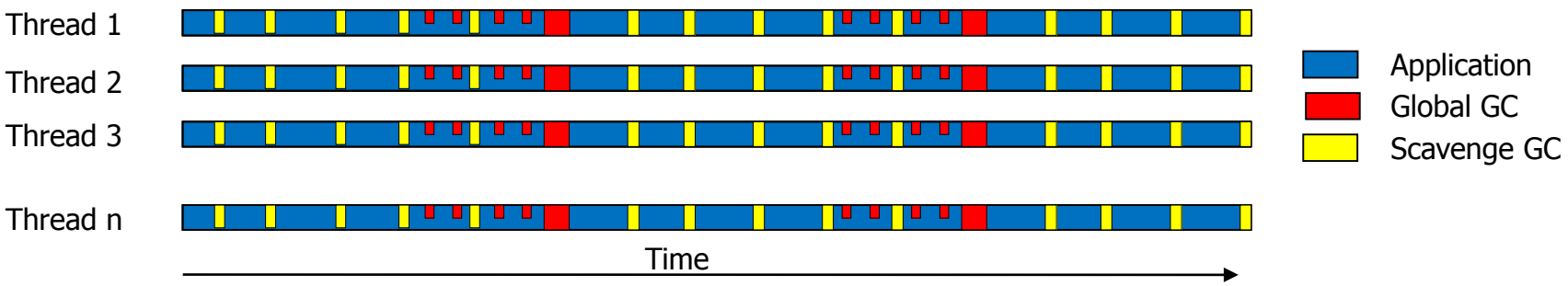
- Collection causes live objects to be:
  - copied from Allocate space to survivor space
  - copied to the Tenured space if they have survived sufficient collections



✓Small but frequently collected area
✓Reduces fragmentation
✓Improves data locality
✓Speeds up future allocations

IBM J9 Garbage Collector: gencon Tenure Space Implementation

- Tenure space uses Concurrent Mark and Sweep GC (same as optavgpause)
- Less frequently collected



Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

IBM J9 Garbage Collector: gencon Recommended Deployment Scenarios

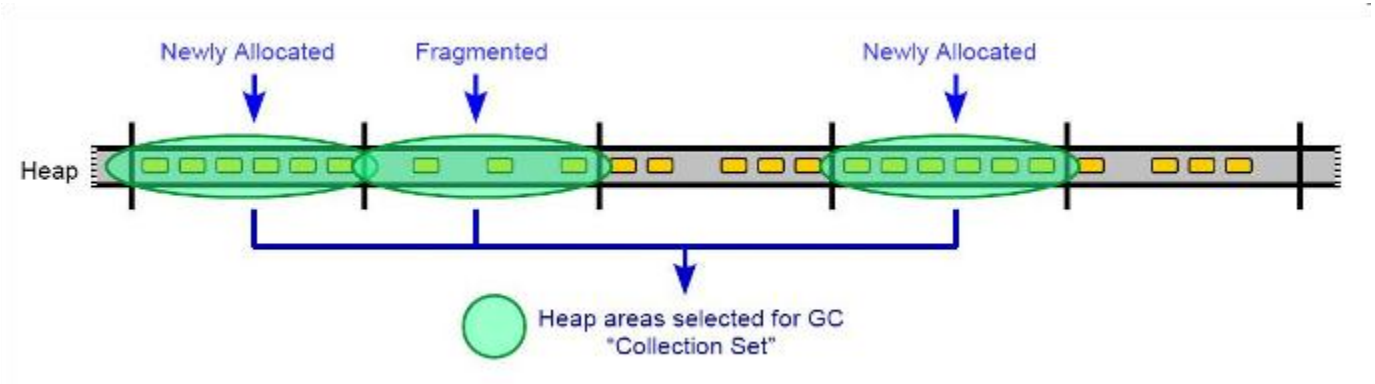
Best when:

- Application allocates many short-lived objects
- Application is transaction-based
- The heap space is fragmented

IBM J9 Garbage Collector: balanced

Goal: Improve responsiveness

- Reduced max pause times to achieve more consistent behaviour
- Incremental collection targets best ROI areas of the heap



Expands platform exploitation possibilities

- Virtualization : group heap data by frequency of access, direct OS paging decisions
- Dynamic reorganization of data structures to improve memory hierarchy utilization (performance)

IBM J9 Garbage Collector: balanced Recommended Deployment Scenarios

Best when:

- Large (>4GB) heaps – to reduce long GC pause
- Frequent global garbage collections
- Excessive time spent in global compaction

Choosing the right GC policy

- Typical configuration
  - Pick a policy based on desired application behaviour
  - Tune heap sizes (use tooling)
  - Helper threads (-Xgcthreads)
  - Lots of other tuning knobs, suggest try hard to ignore, to avoid over-tuning
  - Monitor and re-tune if needed
- Rule of thumb:
  - If GC overhead is > 10%, you've most likely chosen the wrong policy
- Best practice:
  - Don't use System.gc()
  - Avoid finalizers
  - Memory leaks are possible even with a garbage collector

IBM J9 Garbage Collector Family

Policy	Recommended Usage	Notes
optthruput	Throughput (i.e. batch applications)	Default in Java 5, 6
optavgpause	Reduced pause times	
gencon	Transactional Workloads	Default in Java 6.0.1, 7, 8
balanced	Large Heaps	Added in Java 6.0.1, 7

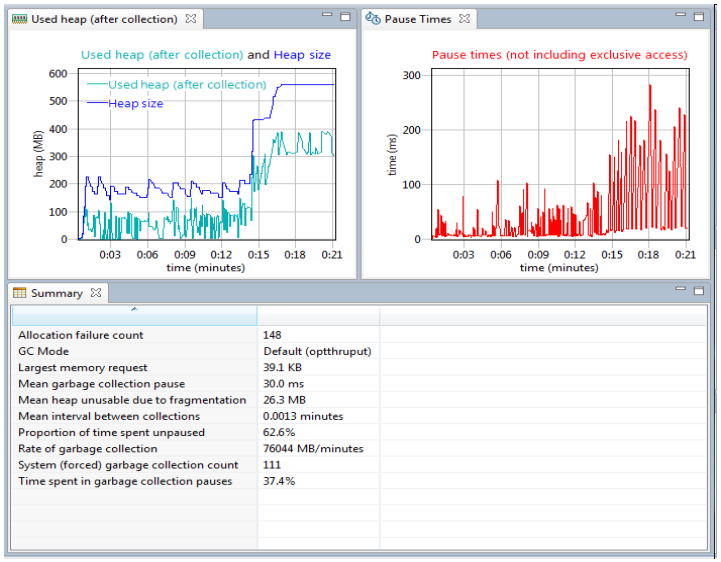
Java Tuning: Agenda

- IBM JVM(J9) Garbage Collector (GC)
  - Overview
  - Policies
  - Choosing the right policy
- **IBM Monitoring and Diagnostic Tools**
- Dynamic Compilations
  - Shared Classes + Ahead Of Time compilations
  - Runtime Instrumentation
- JNI Best Practices

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

Health Center

- Motivating questions:
  - What is my JVM doing? Is everything ok?
  - Why is my application running slowly?
  - Why is it not scaling?
  - Am I using the right options?

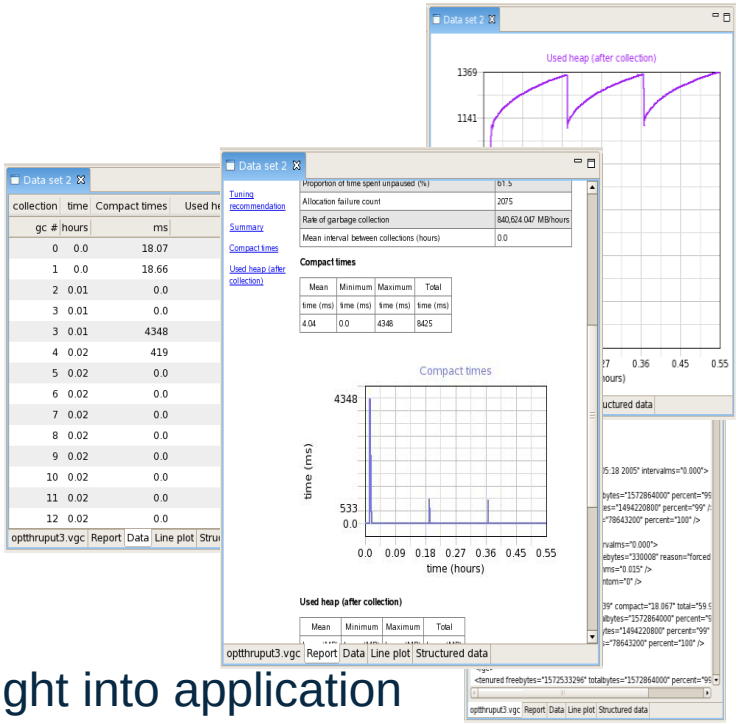


- Overview
  - Lightweight live monitoring tool with very low overhead
  - Understand how your application is behaving, diagnose potential problems with recommendations
  - Visualize garbage collection, method profiling, class loading, lock analysis, file I/O and native memory usage
  - Suitable for all Java applications running on IBM's JVM

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

GC and Memory Visualizer (GCMV)

- Motivating questions:
  - How is the GC behaving? Can I do better?
  - How much time is GC taking?
  - How much free memory does my JVM have?

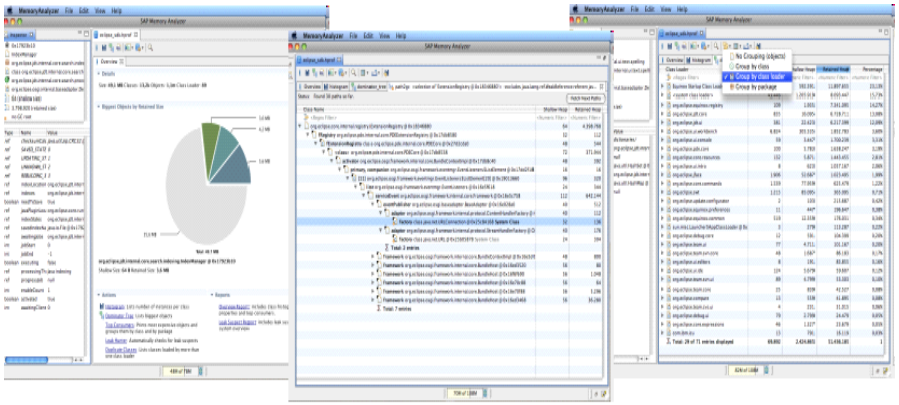


- Overview
  - Analyze Java verbose GC logs, providing insight into application behaviour
  - Visualize a wide range of GC data and Java heap statistics over time
  - Provides the ability to detect memory leaks and optimize GC
  - Recommends tuning options to improve performance

Garbage Collection
IBM Monitoring and Diagnostic Tools for Java

Memory Analyzer

- Motivation questions:
  - Why did I run out of Java memory?
  - What's in my Java heap? How can I explore it and get new insights?



- Overview
  - Tool for analyzing heap dumps and identifying memory leaks from JVMs
  - Works with IBM system dumps, heapdumps and Sun HPROF binary dumps
  - Provides memory leak detection, footprint analysis and insight into wasted space
  - Provides SQL like object query language (OQL)

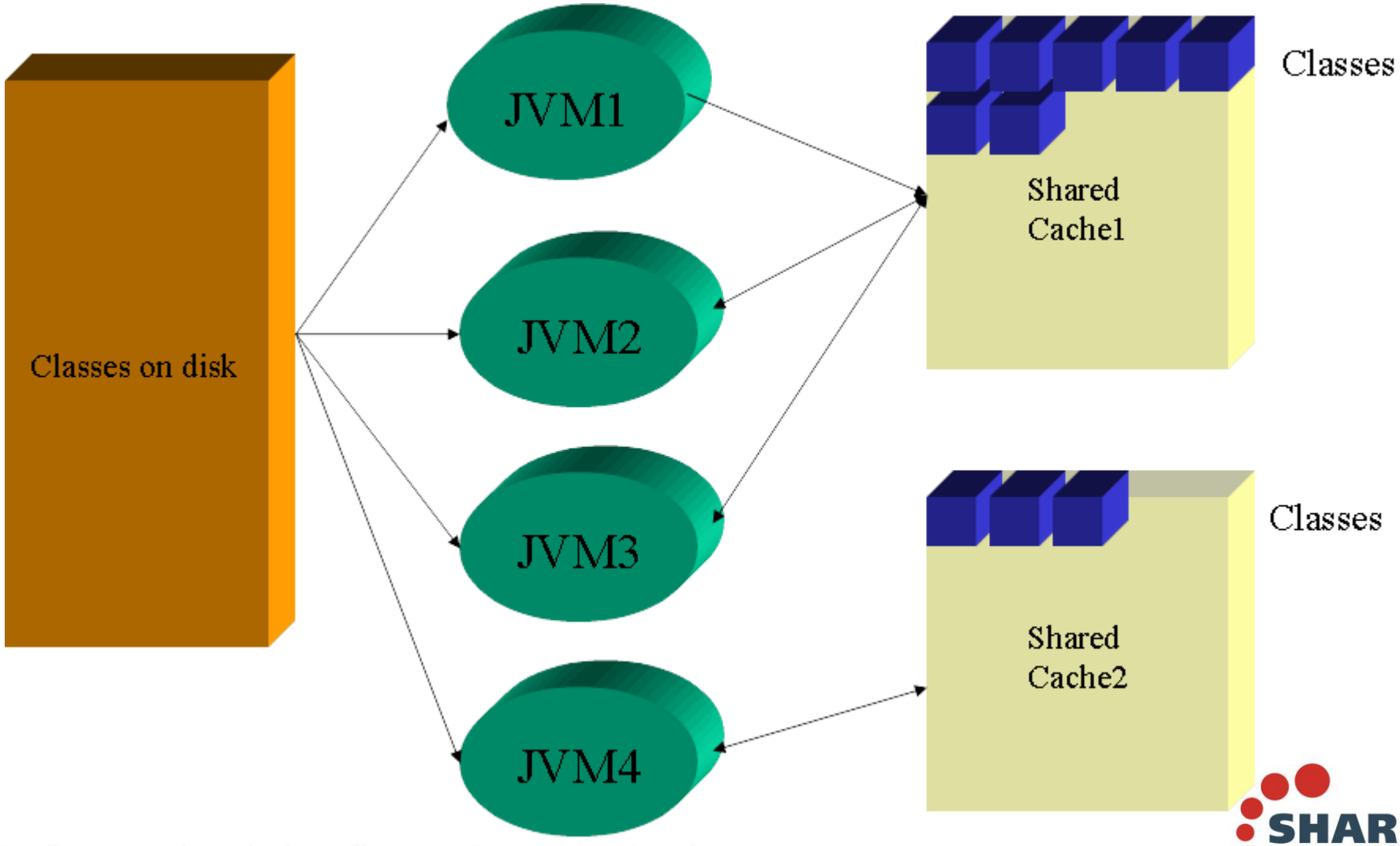
Java Tuning: Agenda

- IBM JVM(J9) Garbage Collector (GC)
  - Overview
  - Policies
  - Choosing the right policy
- IBM Monitoring and Diagnostic Tools
- **Dynamic Compilations**
  - Shared Classes + Ahead Of Time compilations
  - Runtime Instrumentation
- JNI Best Practices

Dynamic Runtime

- Java Runtime Environment **dynamically** loads Java classes at runtime
- The Java Virtual Machine (JVM) can:
  - Interpret Java methods' bytecodes.
  - Compile Java methods' bytecodes into assembly instructions.
- Compilations
  - Pros:
    - Compiles only the methods that matter
    - Profiles your application characteristics for better optimizations
    - Optimize for your exact hardware
  - Cons:
    - Compilation ➔ runtime overhead

Sharing across JVMs!



Shared Classes

- Store classes into a cache that shared by multiple JVMs
- Read-only portions of the class
- Memory footprint reduction
- Startup time improvements (class initialization)
- Cache memory page protection (read-only caches)

Ahead of Time Compilations

- Compiled code generated “ahead-of-time” (AOT)
- Subsequent JVMs can simply load this AOT code
- Startup time improvements
- CPU utilization reduction
- Persisted into the same shared cache

Why not AOT everything?

- Dynamic class loading is a fundamental feature of Java
  - JVM must support dynamic class loading to pass Java certification
  - Classes may not even be stored on disk: Built on-the-fly from raw bytes
    - e.g. Java serialization and reflection services
  - Classes can be transformed on loading to insert new code or adjust existing code

- Dynamic class loading means constraints on compilation change
  - Re-use of a compilation requires verification that same conditions exists in new instance

AOT/JIT Best Practices

- JIT modes:
  - Default — balanced throughput, startup, rampup (server-side)
  - -Xquickstart — faster startup, reduced throughput (client-side)
  - -Xtune:virtualized — reduced compilation overhead, reduced throughput

- Many diagnostic knobs, **not** for performance tuning
  - Impose a new compilation count
  - Impose optimization level
  - Limit compilation to a specific set of methods

- Tuning the shared classes cache size (-Xscmx)

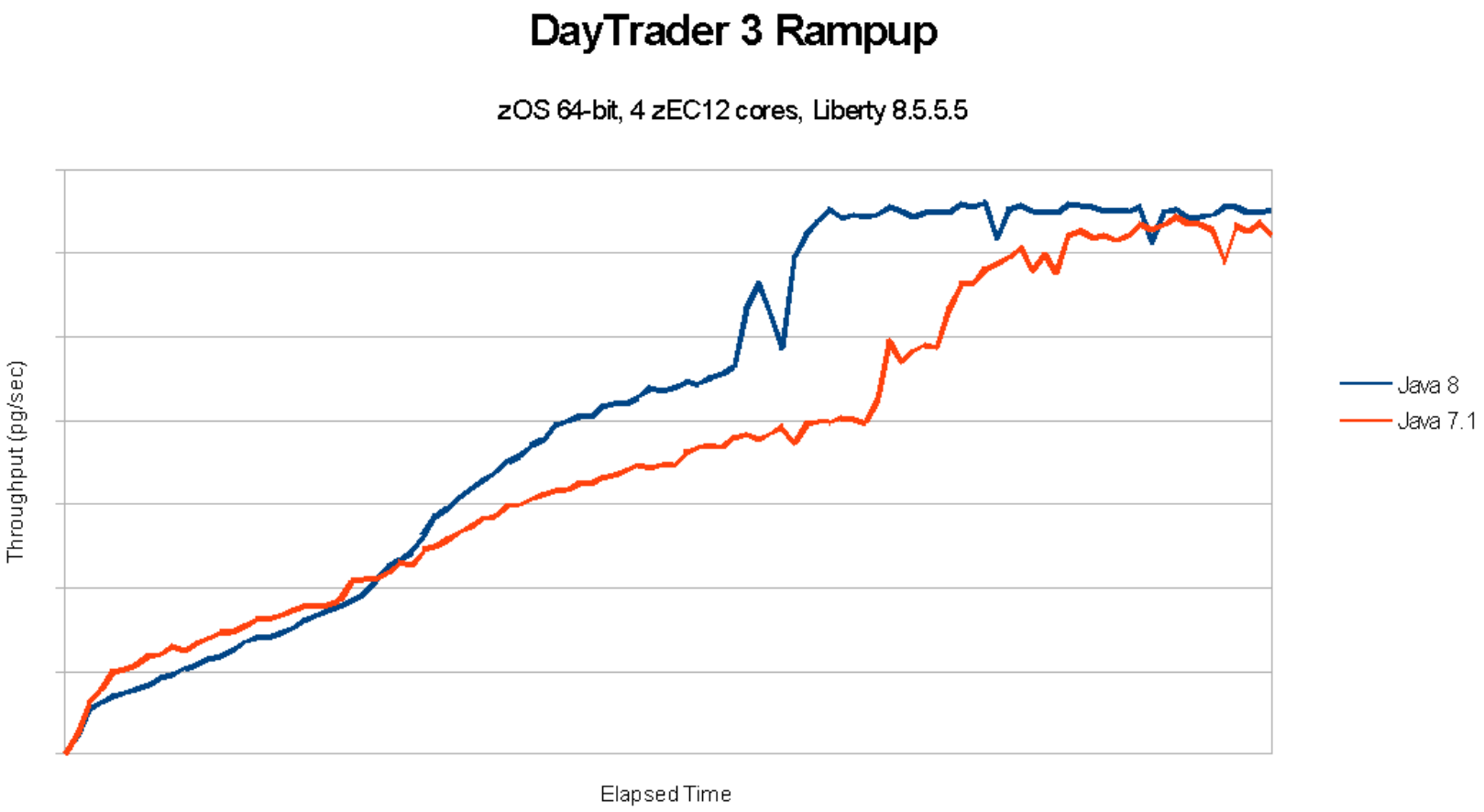
- http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/user/classdatasharing.html?lang=en
- http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/understanding/jit_faq.html?lang=en

Runtime Instrumentation

- Hardware profiling infrastructure added in zEC12
- Low overhead, highly granular data
- Designed and built for dynamic runtimes like Java!

- How is the JIT using Runtime Instrumentation (RI)?
  - Software-based sampling is challenged in detecting 'hot' methods in large, flat Java applications
    - Tens of thousands of compilations → overhead
  - RI provides more granular data
    - JIT initially compiles using cheaper optimizations.
    - RI data to identify 'important' methods to recompile

DayTrader Ramp-up



(Controlled measurement environment, results may vary)

JNI Best Practices: Performance Pitfalls

- See http://www.ibm.com/developerworks/java/library/j-jni
- Common performance pitfalls
  - Not caching method IDs, field IDs, and classes
    - Avoid redundent calls to FindClass(), GetFieldID(), GetMethodId(), and GetStaticMethodID()
  - Triggering array copies
    - Assume arrays are buffered, hence be precise about which elements you really need to avoid needless copying
  - Reaching back instead of passing parameters
    - When possible, flatten object fields into parameters of call. Avoid using JNI services to get to object fields
  - Choosing the wrong boundary between native and Java code
    - Assume Native <-> Java call overhead 10x slower than Native <-> Native or Java <-> Java
- Run –Xcheck:jni!

Thank You!

- Please complete your session evaluations!

Session 16761 :

IBM Java: JVM Tuning

- www.share.org/Seattle-Eval

Joran Siu

Email: joransiu@ca.ibm.com

Summary of Links

- Documentation
  - http://www.ibm.com/developerworks/java/jdk/docs.html
  - http://www.redbooks.ibm.com/redpapers/pdfs/redp3950.pdf
- zOS SDK
  - http://www.ibm.com/servers/eserver/zseries/software/java
- System z Linux SDK
  - http://www.ibm.com/developerworks/java/jdk/linux/download.html
- GC Tuning documentation
  - http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=java+technology+ibm+style:
  - http://www-01.ibm.com/support/docview.wss?uid=swg27013824&aid=1
  - http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html#sec-ng
  - http://www.ibm.com/developerworks/websphere/techjournal/1108_sciampacone/1108_sciampacone.html
- IBM Support Assistant
  - http://www.ibm.com/software/support/isa/