#SHAREorg

SHARE
Technology · Connections · Results

Running Java on Linux on System z

Joran Siu
Java for System z Development, IBM Corporation
joransiu@ca.ibm.com

Session 11823
August 9, 2012: 08:00 AM - 09:00 AM

SHARE in Anaheim 2012

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.

Content

- IBM Java on Z
  - History, overview and roadmap
- IBM J9 Virtual Machine and IBM Testarossa JIT
  - IBM Monitoring and Diagnostic Tools for Java
- J9 R26 for Java 6.0.1 and Java 7
  - z196 exploitation and performance

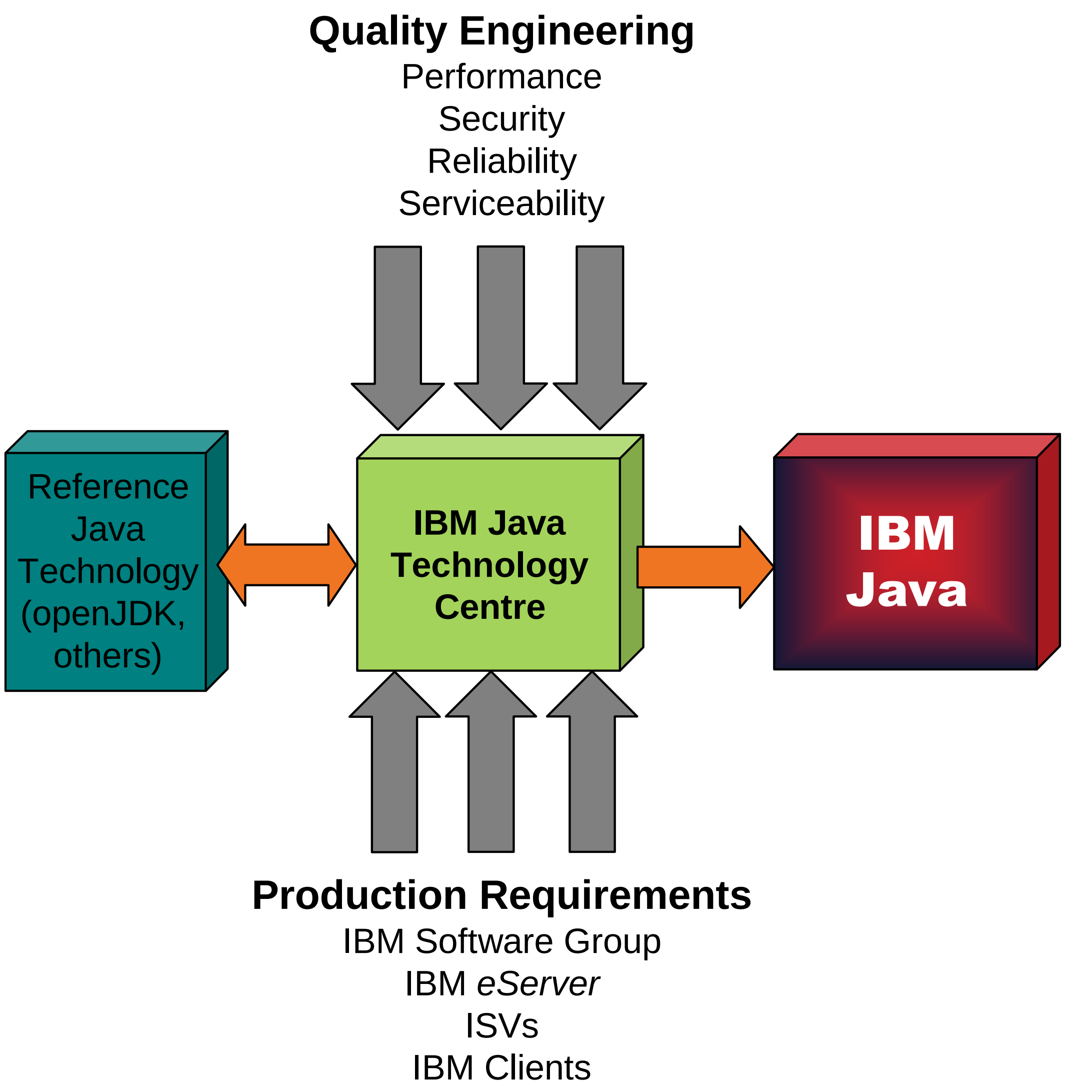
IBM and Java

- **Java is critically important to IBM**
  - Fundamental infrastructure for IBM's software portfolio
  - WebSphere, Lotus, Tivoli, Rational, Information Management (IM)

- **IBM is investing strategically for Java in Virtual Machines**
  - As of Java 5.0, single JVM support
    - JME, JSE, JEE
  - New technology base (J9/TR Compiler) on which to deliver improved performance, reliability, serviceability

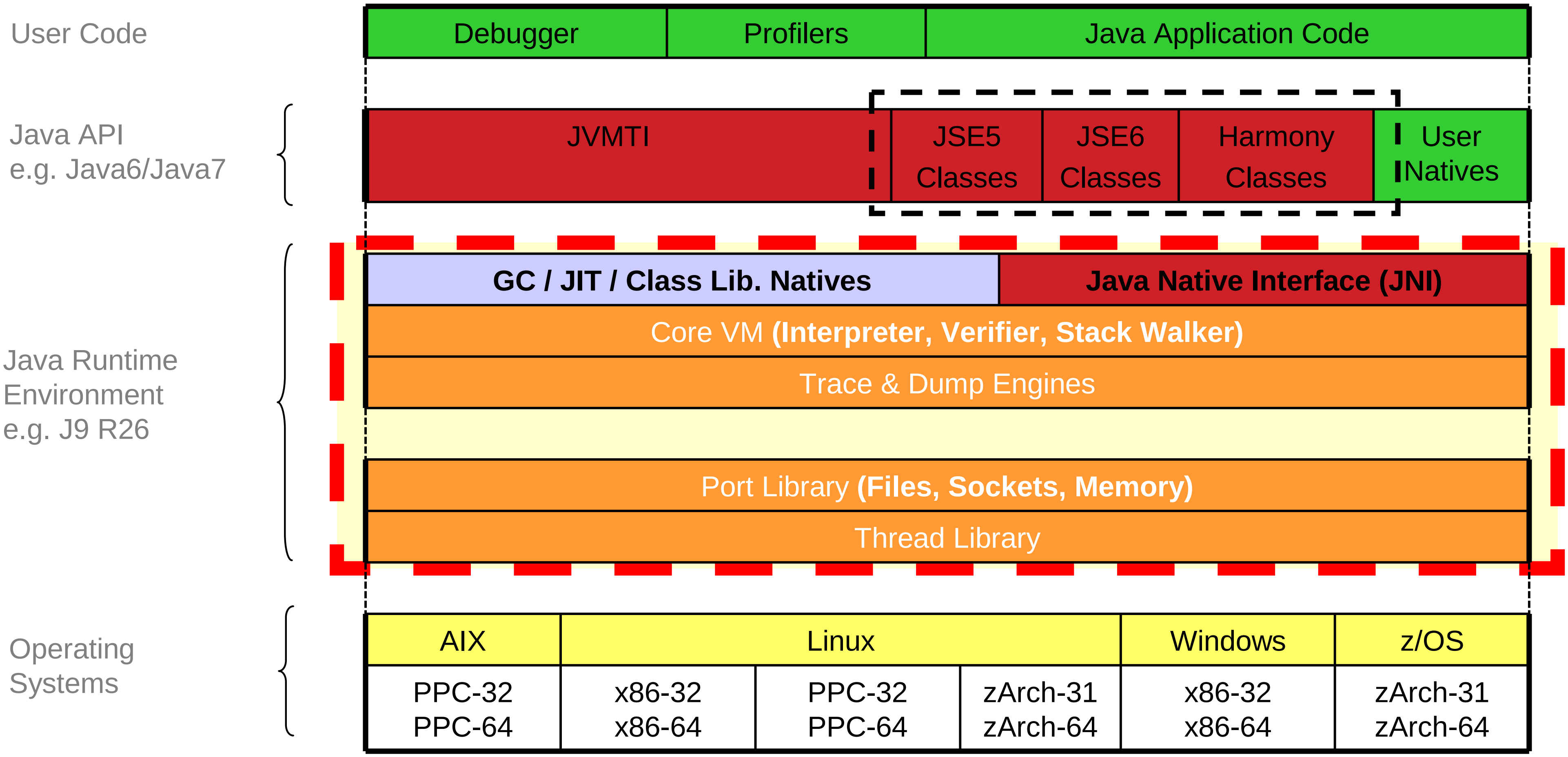
- **IBM also invests in, and supports public innovation in Java**
  - OpenJDK, Eclipse, Apache (XML, Aries, Derby, Geronimo, Harmony, Tuscany, Hadoop …)
  - Broad participation in relevant open standards (JCP, OSGi)

IBM's Approach to Java Technology



- ✓ *Listen to and act upon market requirements*
- ✓ *World class service and support*
- ✓ *Available on more platforms than any other Java implementation*
- ✓ *Highly optimized*
- ✓ *Embedded in IBM's middleware portfolio and available to ISV partners*

JVM Architectural Overview



= User Code

= Java Platform API

= VM-aware

= Core VM

Java 6.0.1:

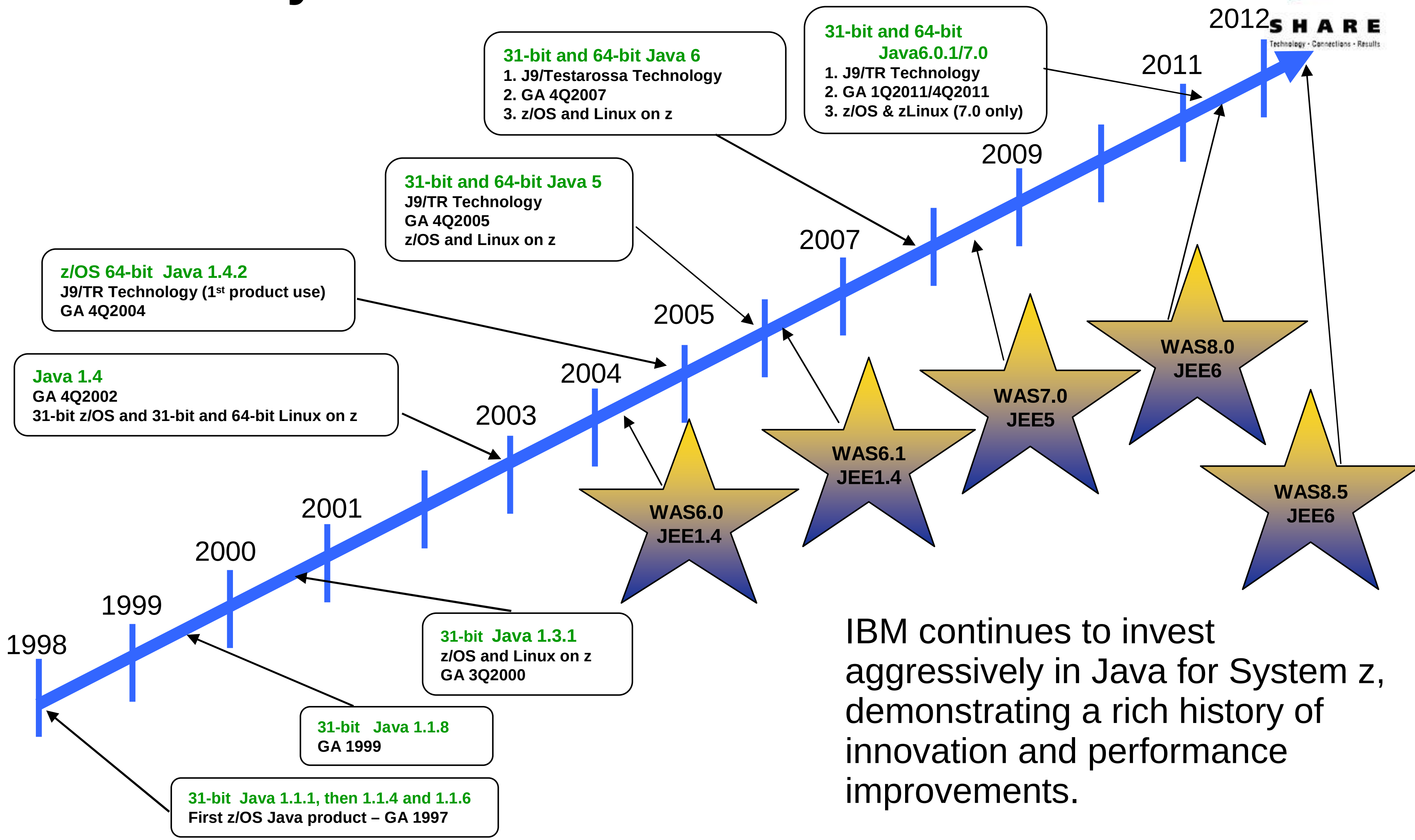
- Fully compatible/compliant Java6 (JSE6)
- Only available with the WAS8 runtime
- Includes new J9 R26 JRE
  - Transparent z196 and new optimization exploitation + New balanced GC policy

Key Differences between Oracle and IBM Java

- IBM and Oracle use the same reference implementation of Java Class Libraries (e.g. OpenJDK)
  - Key differences to be aware of:
    1. Security: Standards do not impose strong separation of interest
    2. ORB: OMG CORBA standard rules
    3. XML: Xerces/Xalan used by both vendors as of Java5, although different levels may be used.

- IBM uses the J9/TR runtime, Oracle uses Hotspot
  - Different JIT/GC/VM tuning and controls
  - Tooling is distinct (e.g. IBM's Health Center)

Java on System z – 15 Years of Innovation



IBM continues to invest aggressively in Java for System z, demonstrating a rich history of innovation and performance improvements.

Testimonials: http://www-01.ibm.com/software/os/systemz/testimonials/
http://www.centerline.net/review/#/3332_B

Timelines and deliveries are subject to change.

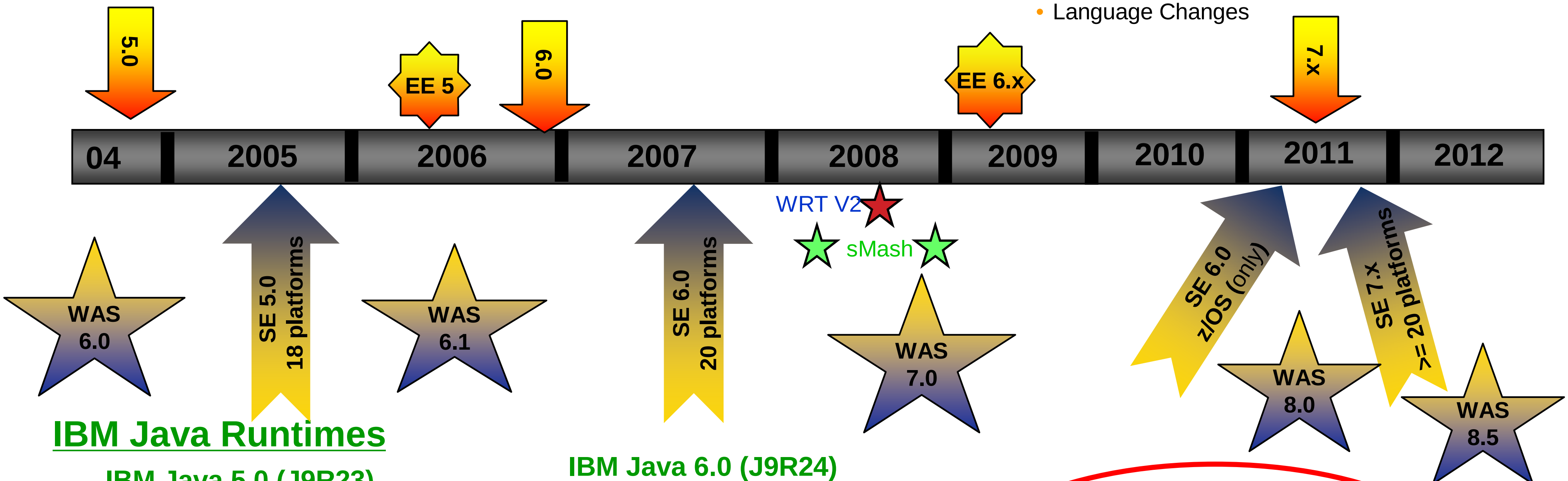
Java Road Map

Oracle Java Runtimes

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



IBM Java Runtimes

IBM Java 5.0 (J9R23)
- Improved performance
  - Generational Garbage Collector
  - Shared classes support
  - New J9 Virtual Machine
  - New Testarossa JIT technology
- First Failure Data Capture
- Full Speed Debug
- Hot Code Replace
- Common runtime technology
  - ME, SE, EE

IBM Java 6.0 (J9R24)
- 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/Java7.0 (J9R26)
- Improvements in
  - Performance
  - GC Technology
- z196™ Exploitation
  - OOO Pipeline
  - 70+ New Instructions

Timelines and deliveries are subject to change.

Java 7.0 – What to look for

New I/O

- Meets the increasingly I/O intensive demands of data mining and analytics workloads
- Significant performance and footprint gains from async I/O

Concurrency Libraries

- Exploit larger multi-core systems, such as next generation Power and System z, by providing better scalability, higher throughput and lower total cost of ownership from server consolidations

Dynamic language support

- Leverage the advantages of a single runtime for dynamic language applications written in PHP, Groovy, jRuby and jython

Language improvements

- Improved efficiency through simplified day-to-day programming tasks
- Protect developer commitment to, and customer/ISV investment in, the Java ecosystem.

IBM Java Runtime Environment

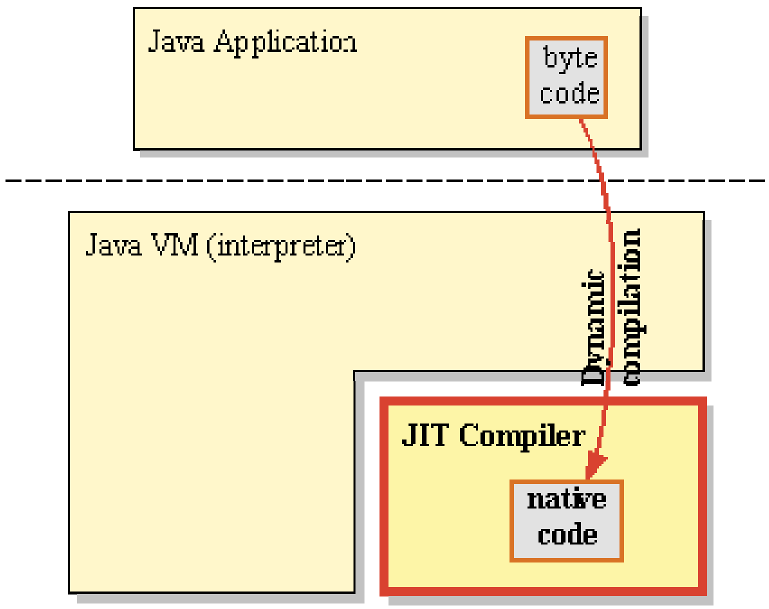
- IBM's implementation of Java 5, Java 6 and Java 7 are built with **IBM J9 Virtual Machine** and **IBM Testarossa JIT Compiler** technology
  - Independent clean-room JVM runtime & JIT compiler

- Combines best-of breed from embedded, development and server environments… from a cell-phone to a mainframe!
  - Lightweight flexible/scalable technology
  - World class garbage collection – gencon, balanced GC policies
  - Startup & Footprint - Shared classes, Ahead-of-time (AOT) compilation
  - 64-bit performance - Compressed references & Large Pages
  - Deep System z exploitation – z196/z10/z9/z990 exploitation

- Millions of instances of J9/TR compiler

IBM Testarossa JIT Compiler – Introduction

- IBM Testarossa JIT is IBM's Production JIT on all Platforms since SDK5
- Developed at the IBM Toronto Lab
- The Toronto Lab has 30+ years of expertise in compilation and optimization technologies



- Close relationships with:
  - Research: productizing innovative ideas and experimental technologies.
    (Tokyo/Watson Research Lab)
  - Hardware: best possible performance with the underlying system and processor.
    (Poughkeepsie, Austin, xSeries)
  - IBM Middleware: work with DB2® , WAS to provide strong performance
    (SVL, Toronto, Raleigh)

IBM Testarossa JIT – Dynamic, adaptive, optimizing compiler

- **Dynamic**
  - Triggered at runtime based on projected profitability of compilation
  - Compiled methods can be freely intermixed with interpreted callers/callees
  - May have multiple versions of methods built with different levels of optimization

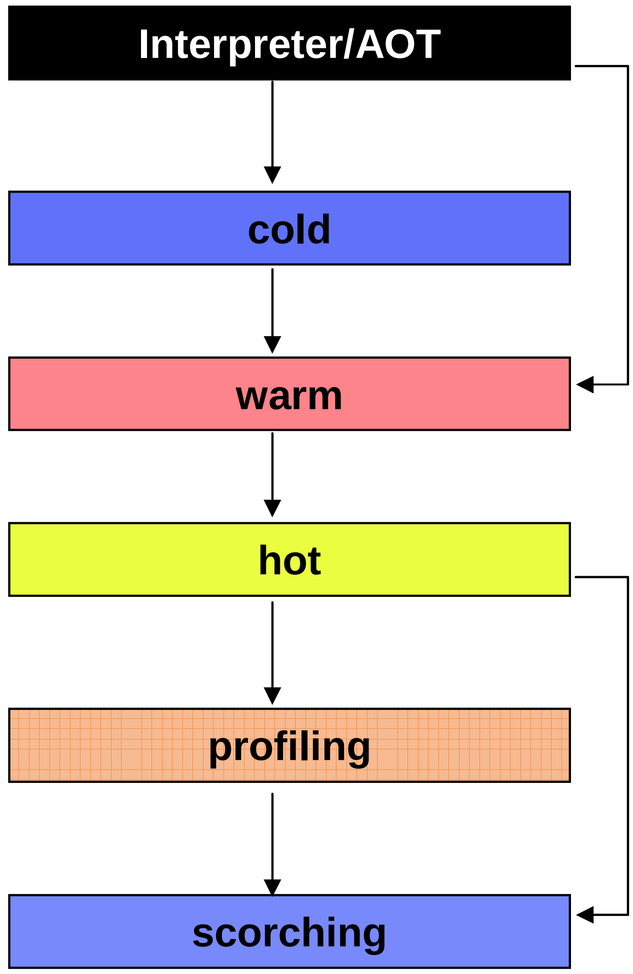
- **Adaptive**
  - Sensitive to need for program to have CPU (e.g. throttled during startup, runs on asynchronous thread)
  - Able to profile program to retrieve common control paths or data values
  - Profile information used in subsequent re-optimizing compilation step

- **Optimizing**
  - Comprehensive collection of conventional optimizations
    - control flow simplification, data flow analysis, etc
  - Speculative and Java-specific optimizations
    - de-virtualization, partial inlining, lock coarsening, etc
  - Deep exploitation of System z micro-architecture

IBM Testarossa JIT – Compilation Strategy

Goals

- Focus compilation CPU time where it matters
  - Stager investment over time to amortize cost
- Methods start as interpreted
  - Interpreter does first level profiling
- After N invocations methods get compiled at ‘warm’ level
- Sampling thread used to identify hot methods
- Methods may get recompiled at ‘hot’ or ‘scorching’ levels
- Transition to ‘scorching’ goes through a temporary profiling step
  - Global optimizations are directed using profiling data
  - Hot paths through methods are identified
    - register allocation, branch straightening, etc
  - Values/types are profiled, hot paths are specialized/versioned
  - Virtual calls are profiled, hot targets are in-lined



IBM Testarossa JIT – System z Support

- Idioms are recognized in Java source/bytecodes
- Bytecodes converted to CISC instructions**
- CISC Instructions:
  - TROT, TRTO, TRTT, TROO (TR = Translate, O = One Byte, T = Two bytes)
  - SRST (search string)
  - MVC (move character)
  - XC (exclusive-or)
  - CLC (compare-logical)

- Example:

```
while (i < end) {
   value = table[arrB[i+offsetB]];
   if (value == termChar) break;
   arrA[i+offsetA] = value;
   ++i;
}
```

```
LB:  DS 0H
     TRxx      // xx depends on arrA/B types
     BRC LB    // re-drive long xlate
```

** M. Kawahito, *et al.*, A new idiom recognition framework for exploiting hardware-assist instructions, ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, 2006

IBM J9 Garbage Collector

- **IBM J9 VM garbage collector family**

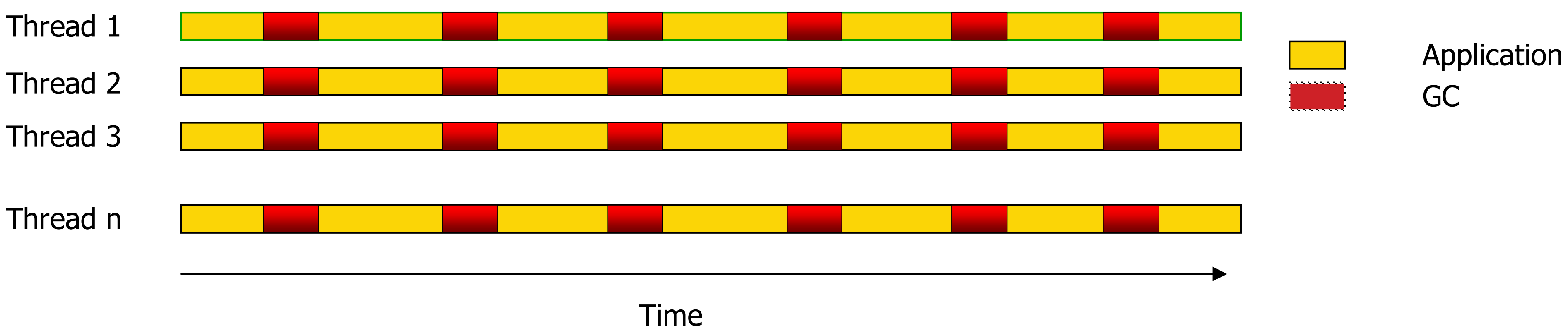
Policy	Recommended usage	Notes
`optThroughput`	optimized for throughput	default in Java5 and Java6
`optAveragePause`	optimized to reduce pause times	
`gencon`	optimized for transactional workloads	default in Java601/Java7
~~**`subPools`**~~	~~optimized for large MP systems~~	~~deprecated in Java601/Java7~~
`balanced`	optimized for large heaps	added in Java601/Java7

- Why have many policies? Why not just “the best”?
  - Cannot always dynamically determine what tradeoffs the user/application are willing to make
  - ***Pause time vs. Throughput***
    - *Trade off frequency and length of pauses vs. throughput*
  - ***Footprint vs. Frequency***
    - *Trade off smaller footprint vs. frequency of GC pauses/events*

IBM J9 Garbage Collector: -Xgcpolicy:optthruput

The default policy in Java5 and Java6.

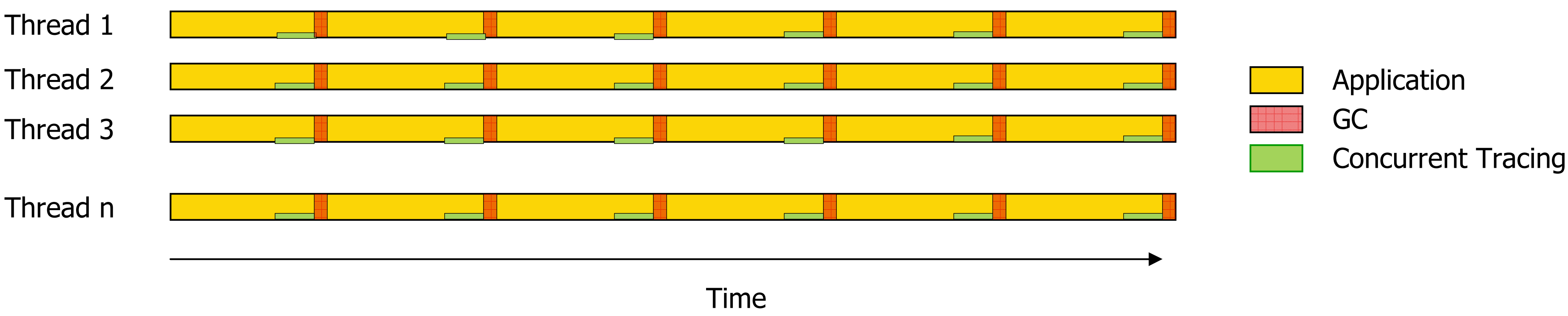
Used for applications where raw throughput is more important than short GC pauses. The application is stopped each time that garbage is collected.



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: -Xgcpolicy:optavgpause

Trades high throughput for shorter GC pauses by performing some of the garbage collection concurrently. The application is paused for shorter periods.



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.

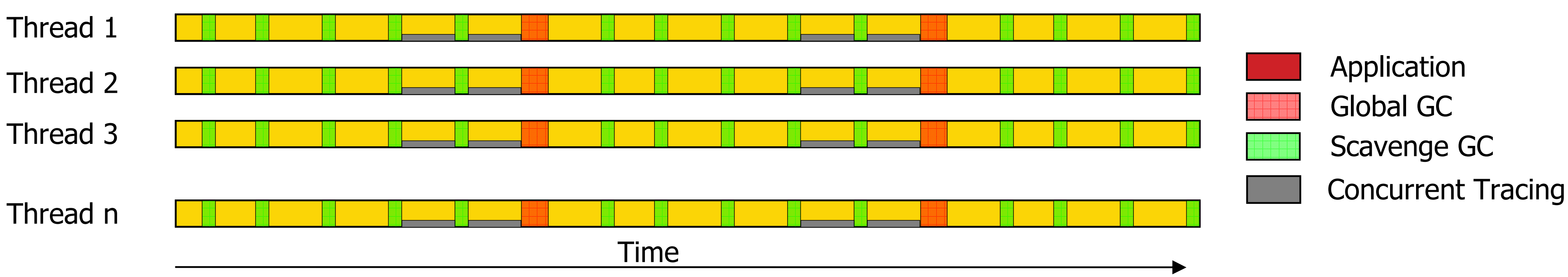
IBM J9 Garbage Collector: -Xgcpolicy:gencon

- Best of both worlds
  - Throughput + Small Pause Times
  - Shown most value with customers
- Two types of collection:
  - Generational nursery (local) collection
  - Partially concurrent nursery & tenured (global) collection
- Why a generational + concurrent solution?
  - For most workloads objects die young
    - Generational allows a better return on investment (less effort, better reward)
    - Performance can be close or even better than standard configuration
  - Reduce large pause times
    - Partially concurrent with application thread (application thread is 'taxed')
    - Mitigates cost of object movement, and cache misses

IBM J9 Garbage Collector: -Xgcpolicy:gencon

Default policy in Java6.0.1/Java7

Handles short-lived objects differently than objects that are long-lived. Applications that have many short-lived objects can see shorter pause times with this policy while still producing good throughput.

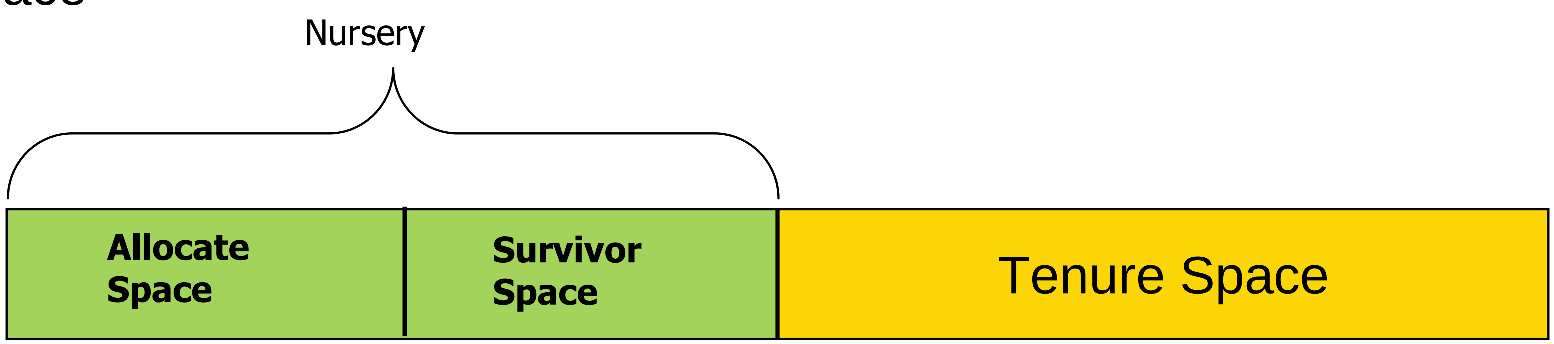


Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

IBM J9 Garbage Collector: A closer look into gencon

- Heap is split into two areas:
  - Objects created in the nursery (a *small but frequently collected area)*
  - Objects that survive a number of collections are promoted to tenured area (*less frequently collected*)

- Nursery is further split into two spaces: ‘allocate’ and ‘survivor’
- A collection in the nursery (scavenge) copies objects from the ‘allocate’ space to the ‘survivor’ space
  - Reduces fragmentation, improves data locality, speeds up future allocations
- If an object survive X number of scavenges it gets promoted to the ‘tenure’ space



The division between allocate and survivor space is dynamic.

It will be adjusted depending on the survival rate.

IBM J9 2.6 Technology Enhancements:
Garbage Collection: Balanced Policy

Improved responsiveness in application behavior

- Reduced maximum pause times to achieve more consistent behavior
- Incremental result-based heap collection targets best ROI areas of the heap
- Native memory aware approach reduces non-object heap consumption

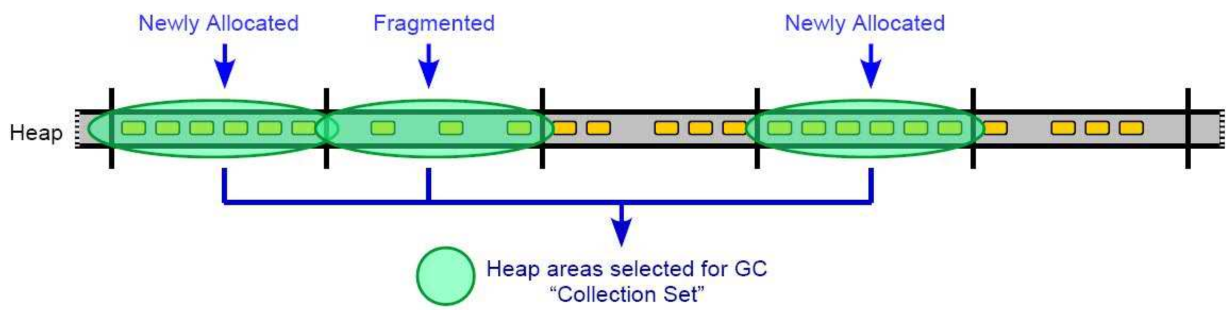
Next generation technology expands platform exploitation possibilities

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

Recommended deployment scenarios

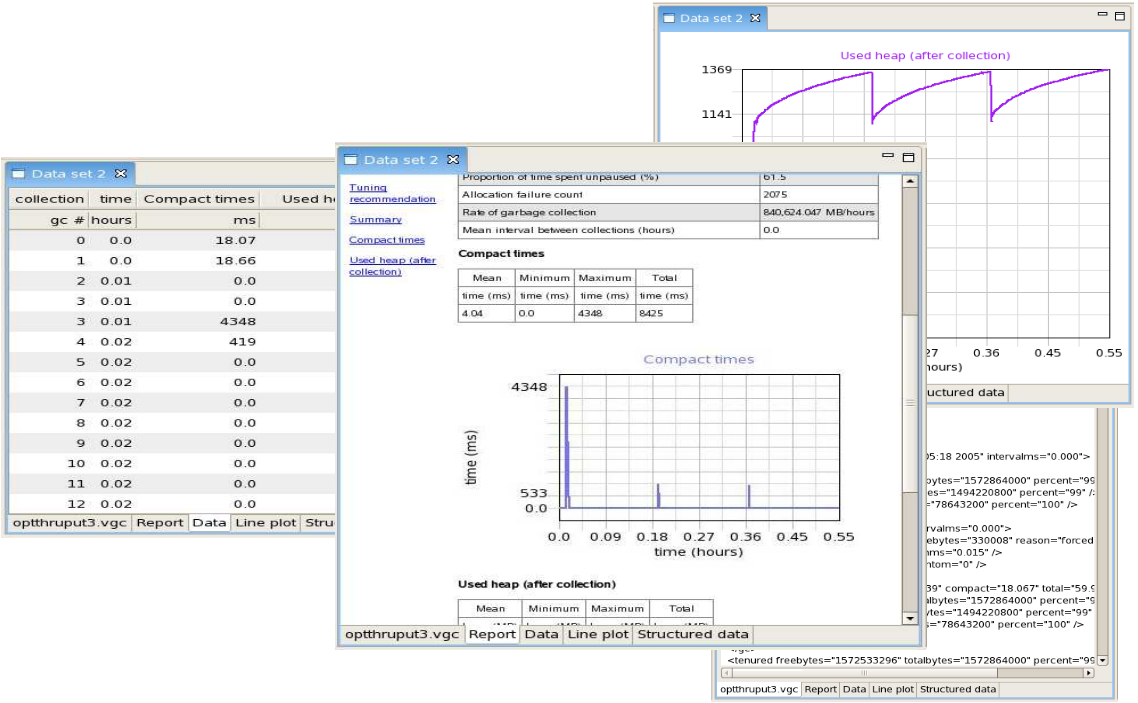
- Large (>4GB) heaps
- Frequent global garbage collections
- Excessive time spent in global compaction
- Relatively frequent allocation of large (>1MB) arrays

Input welcome: Help set directions by telling us your needs



IBM J9 Garbage Collector: Tuning

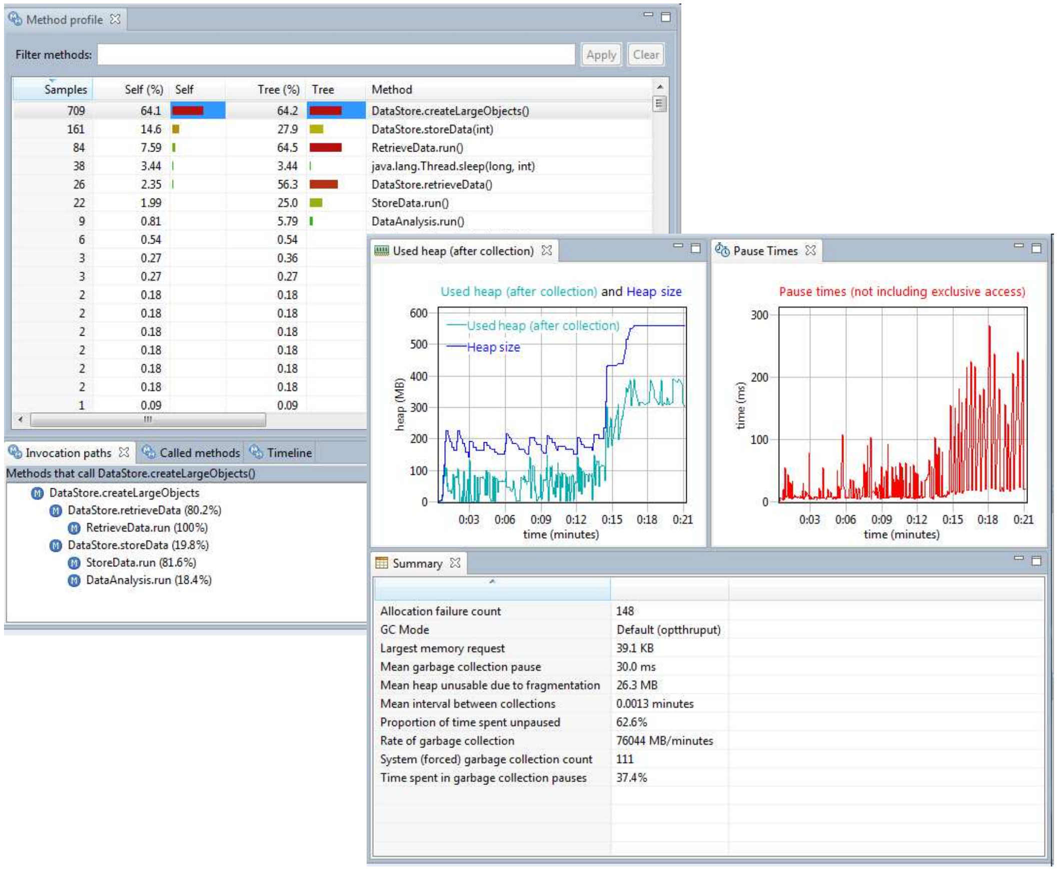
- GC Tuning documentation
  - http://www.ibm.com/developerworks/views/java/libraryview.jsp?search_by=java+technology+ibm+style:
  - http://www-01.ibm.com/support/docview.wss?uid=swg27013824&aid=1
  - http://proceedings.share.org/client_files/SHARE_in_San_Jose/S1448KI161816.pdf
  - http://www.redbooks.ibm.com/redpapers/pdfs/redp3950.pdf
- GC and Memory Visualizer – Views on verbose GC
- Typical configuration
  - Pick a policy based on desired application behaviour
  - Tune heap sizes (use tooling)
  - Helper threads (-Xgcthreads)
  - Avoid finalizers
  - Don't use System.gc()
  - Lots of other tuning knobs, suggest try hard to ignore, to avoid over-tuning
- Memory leaks are possible even with a garbage collector



What is IBM Support Assistant?

- **IBM Support Assistant**
  - A free application that simplifies and automates software support
  - Helps customers analyze and resolve questions and problems with IBM software products.
  - Includes rich features and serviceability tools for quick resolution to problems
- **Meant for diagnostics and problem determination**
  - Not a monitoring tool

IBM Monitoring and Diagnostic Tools for Java - Health Center



What problem am I solving

- 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

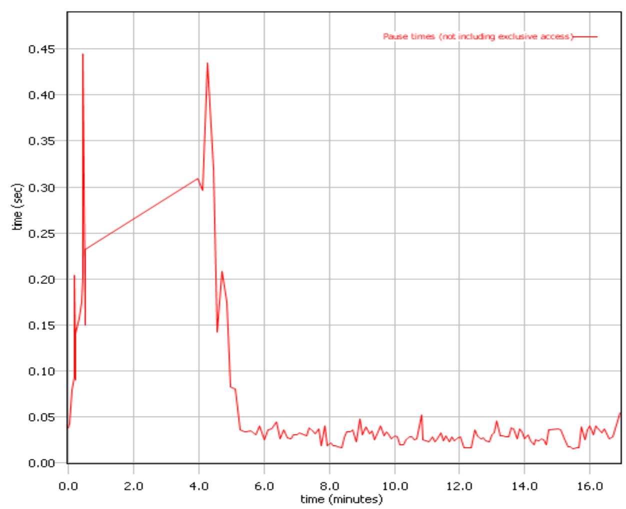
IBM Monitoring and Diagnostic Tools for Java - GCMV

What problem am I solving

- •How is the Garbage Collector (GC) behaving? Can I do better?
- •How much time is GC taking?
- •How much free memory does my JVM have?

Overview

- •Analyse Java verbose GC logs, providing insight into application behaviour
- •Visualize a wide range of garbage collection data and Java heap statistics over time
- •Provides the ability to detect memory leaks and optimized garbage collection
- •Recommendations use heuristics to guide you towards GC performance tuning



Tuning recommendation

The garbage collector seems to be compacting excessively. On average 45% of each pause was spent compacting the heap. Compaction occurred on 40% of collections. Possible causes of excessive compaction include the heap size being too small or the application allocating objects that are larger than any contiguous block of free space on the heap.

The garbage collector is performing system (forced) GCs. 5 out of 145 collections (3.448%) were triggered by System.gc() calls. The use of System.gc() is generally not recommended since they can cause long pauses and do not allow the garbage collection algorithms to optimise themselves. Consider inspecting your code for occurrences of System.gc().

The mean occupancy in the nursery is 7%. This is low, so the gencon policy is probably an optimal policy for this workload.

The mean occupancy in the tenured area is 14%. This is low, so you have some room to shrink the heap if required.

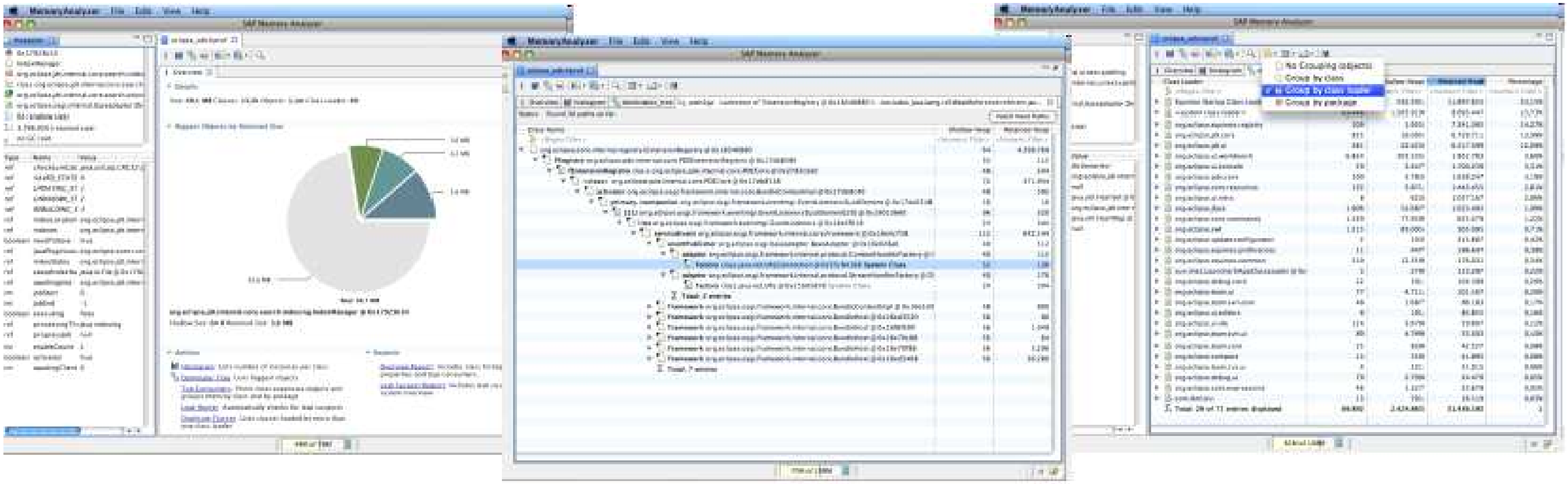
Summary

Allocation failure count	140
Concurrent collection count	0
Forced collection count	5
GC Mode	gencon
Global collections - Mean garbage collection pause (ms)	185
Global collections - Mean interval between collections (minutes)	0.13
Global collections - Number of collections	5
Global collections - Total amount tenured (MB)	93.1
Largest memory request (bytes)	127784
Minor collections - Mean garbage collection pause (ms)	48.2
Minor collections - Mean interval between collections (ms)	7193
Minor collections - Number of collections	140

IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer

What problem am I solving

- 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
- Objects by Class, Dominator Tree Analysis, Path to GC Roots, Dominator Tree by Class Loader
- Provides SQL like object query language (OQL)

Java6R26/Java7 on zLinux Executive Summary

z196 and Java6R26/Java7: Engineered Together

- Up-to 2.7x improvement to Java throughput
- Reduced footprint
- Improved responsiveness in application behavior

J9 R2.6 Virtual Machine

- Significant enhancements to JIT optimization technology
- z196 exploitation of instructions and new pipeline
- New Balanced GC policy to reduce max pause times
- Default GC policy changed to gencon

Performance

- 2.1x improvement to multi-threaded workload
- 1.93x improvement to CPU-intensive workload

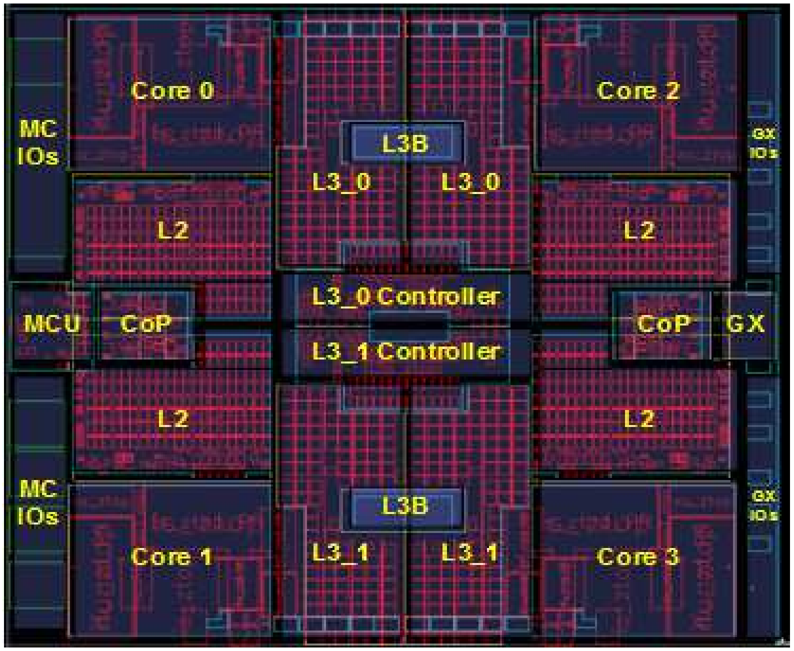
IBM J9 2.6 Technology Enhancements:
System zEnterprise 196 and Java6.0.1/Java7

Hardware for Java

- New Out-Of-Order pipeline design
- New larger cache structure
- Higher clock speed (~5.2GHz)

70+ new instructions used by Java

- Register high-word facility
  - Facilitates use of upper 32-bits of registers
- Interlock facility update
  - Better Java concurrency
- Non-destructive operands
  - Reduce path-length
- Conditional-load/store
  - Remove expensive branches
- Instruction scheduler for Out-of-Order pipeline



IBM J9 2.6 Technology Enhancements: Optimization technology

Reducing pressure on the instruction cache

- Enables better exploitation of new OOO compute bandwidth
  - Partial and general inlining improvements
  - Implicit NULL checks and more aggressive trap exploitation
  - Reduced path length for object allocation
  - Out-of-line instruction selection
  - Loop alignment

Reducing pressure on the data cache

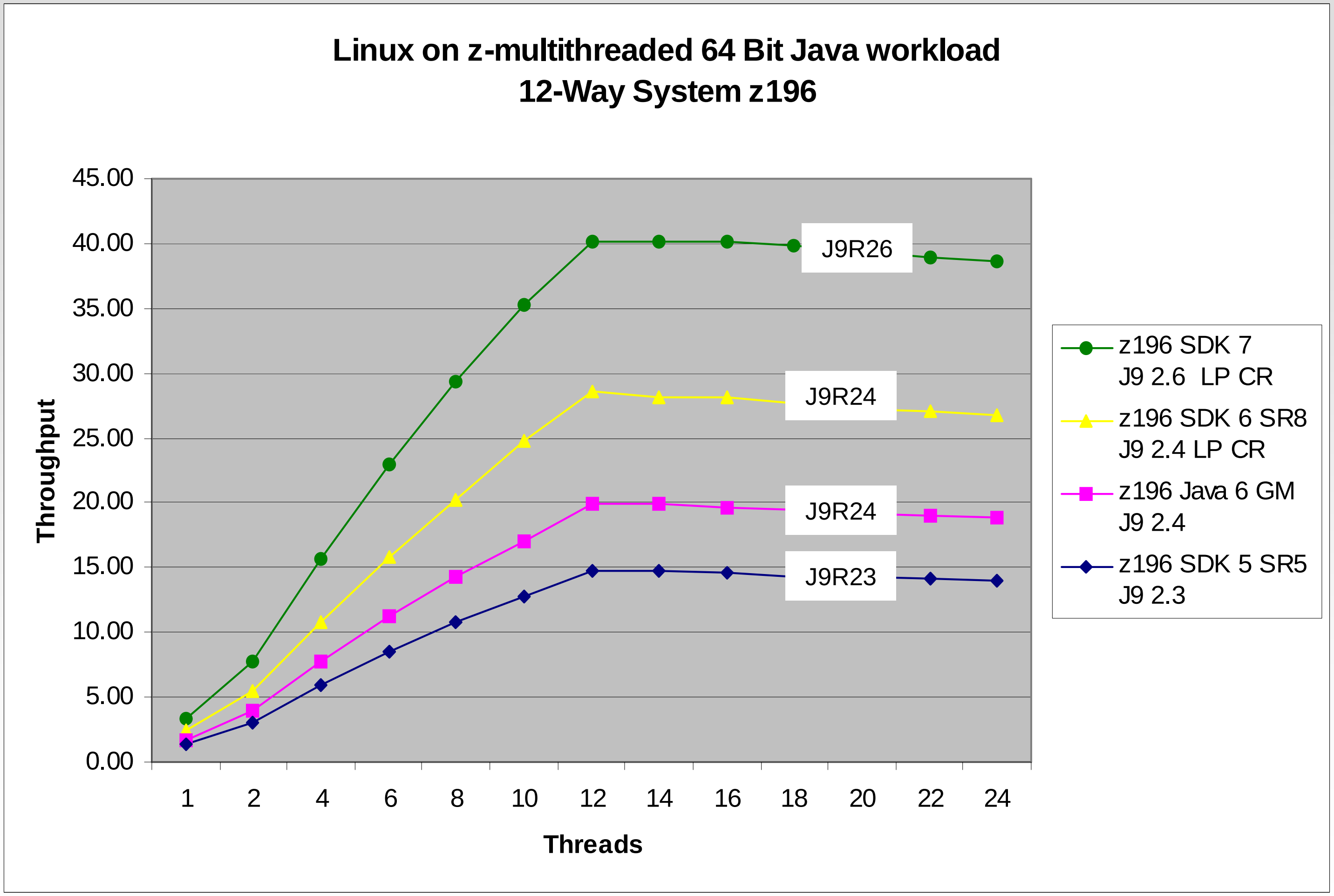
- Mitigates effects of cache latencies on new leveraging core speed
  - Java object header reduction
  - Better escape analysis
  - Pre-fetch exploitation
  - Better exploitation of extended immediate forms
  - Improved interface-dispatch

Scalability and concurrency

- Improved 3-tier lock strategy
- java.util.concurrent optimizations

General optimizer and codegen improvements

Linux on z Java7: 64 Bit Multi-threaded 12-Way Benchmark



2.7x Aggregate Software improvement

- 42% Java 7 (LP,CR) vs Java6 SR8 (LP, CR)
- 42% Java 6 SR8 (LP,CR) vs Java6 GM
- 35% Java 6 GM versus Java 5 SR5

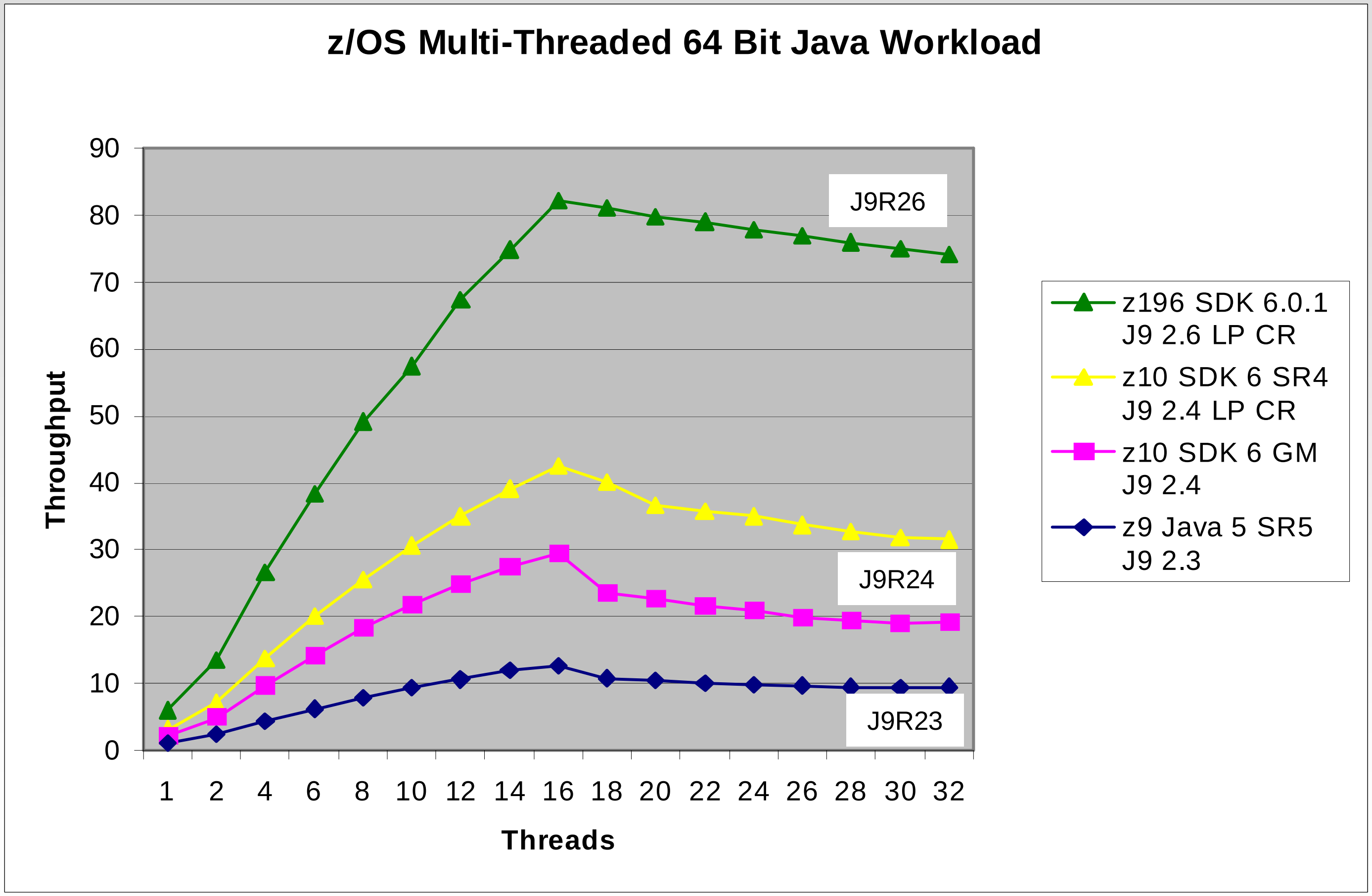
CR = Compressed References

LP = Large Pages

(Controlled measurement environment, results may vary)

z/OS Java SDK 7: 16-Way Performance

Aggregate HW and SDK Improvement z9 Java 5 SR5 to z196 Java 6.0.1 and Java 7



~7x Improvement from z9, Java5 SR5 to z196 Java 6.0.1 and Java 7

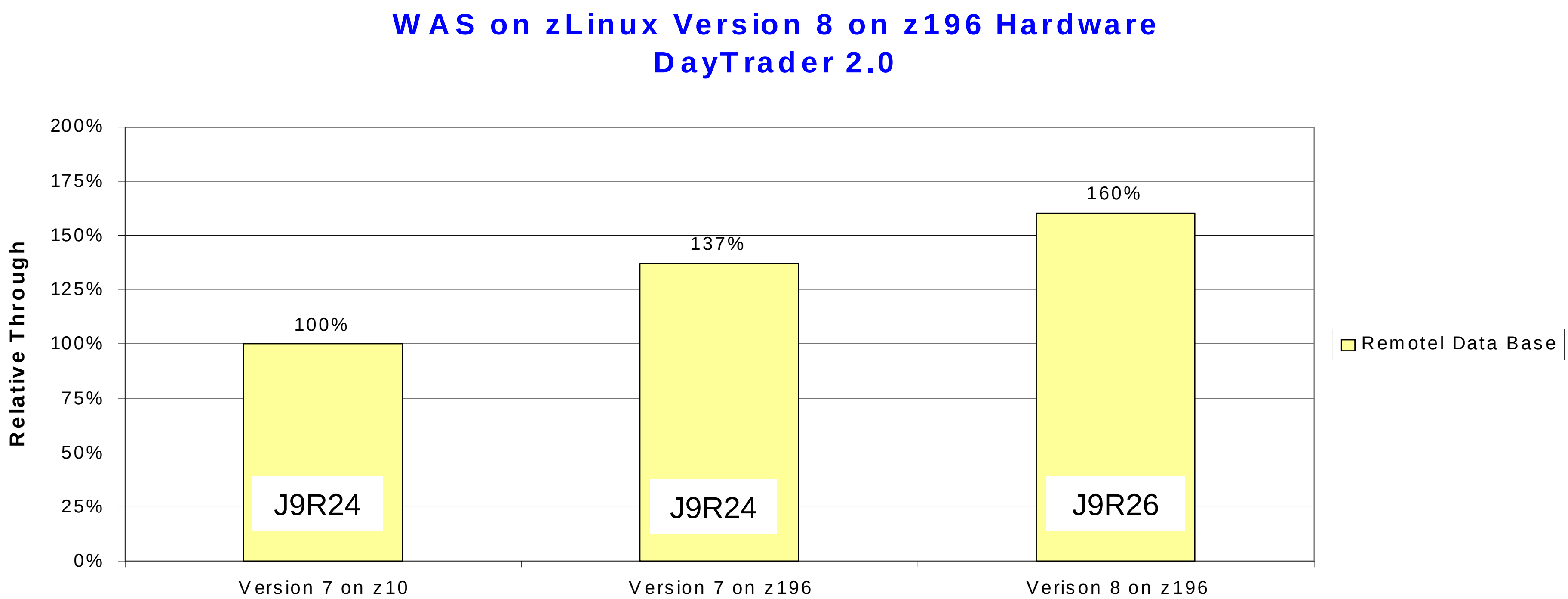
Note: Java 7 and Java 6.0.1 have almost identical performance for this workload

CR = Compressed References

LP = Large Pages

(Controlled measurement environment, results may vary)

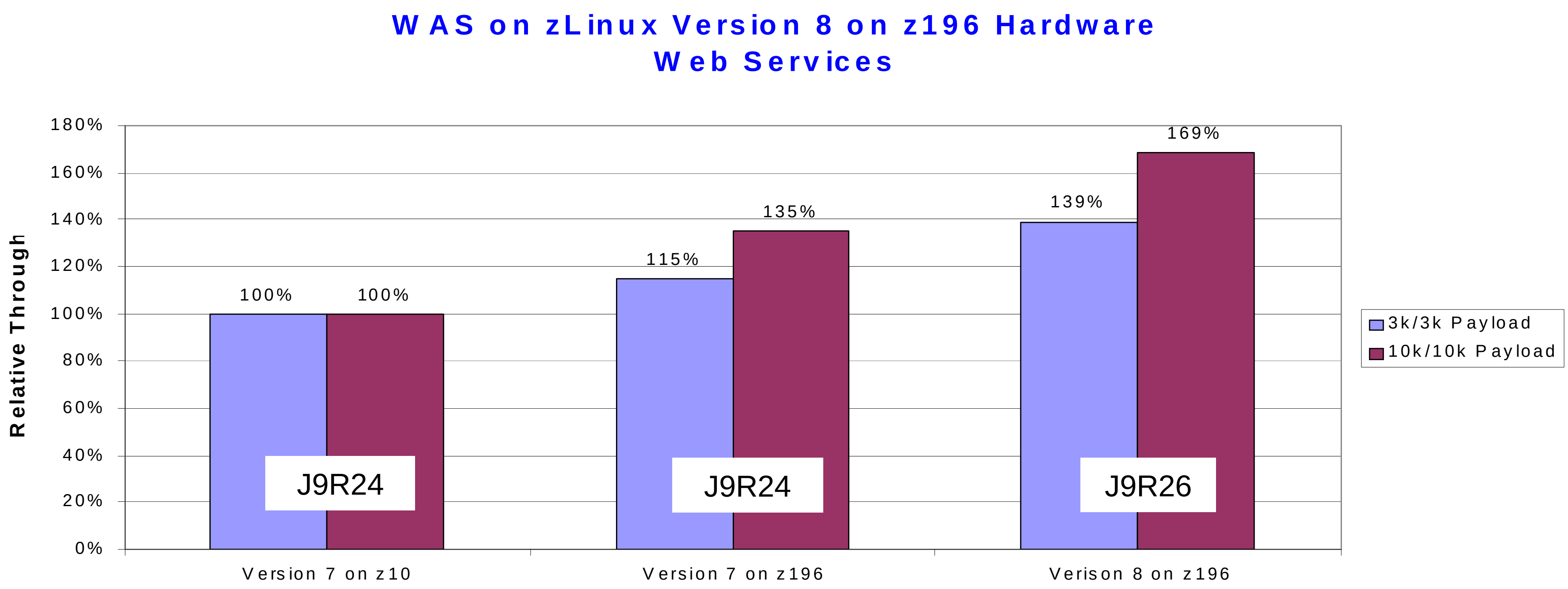
Linux on z - WAS8.0 and Java6 R26



- Upgrading from z10 to z196 improved throughput by 37% using our DayTrader 2.0 EJB benchmark.
- Additionally, upgrading to WAS V8.0 improved performance by another 17%. This increase is a result of improvements to the following areas:
  - JVM and JIT optimizations
  - OpenJPA code paths
- The combine hardware and software improvement is 60%.

(Controlled measurement environment, results may vary)

Linux on z - WAS8.0 and Java6 R26

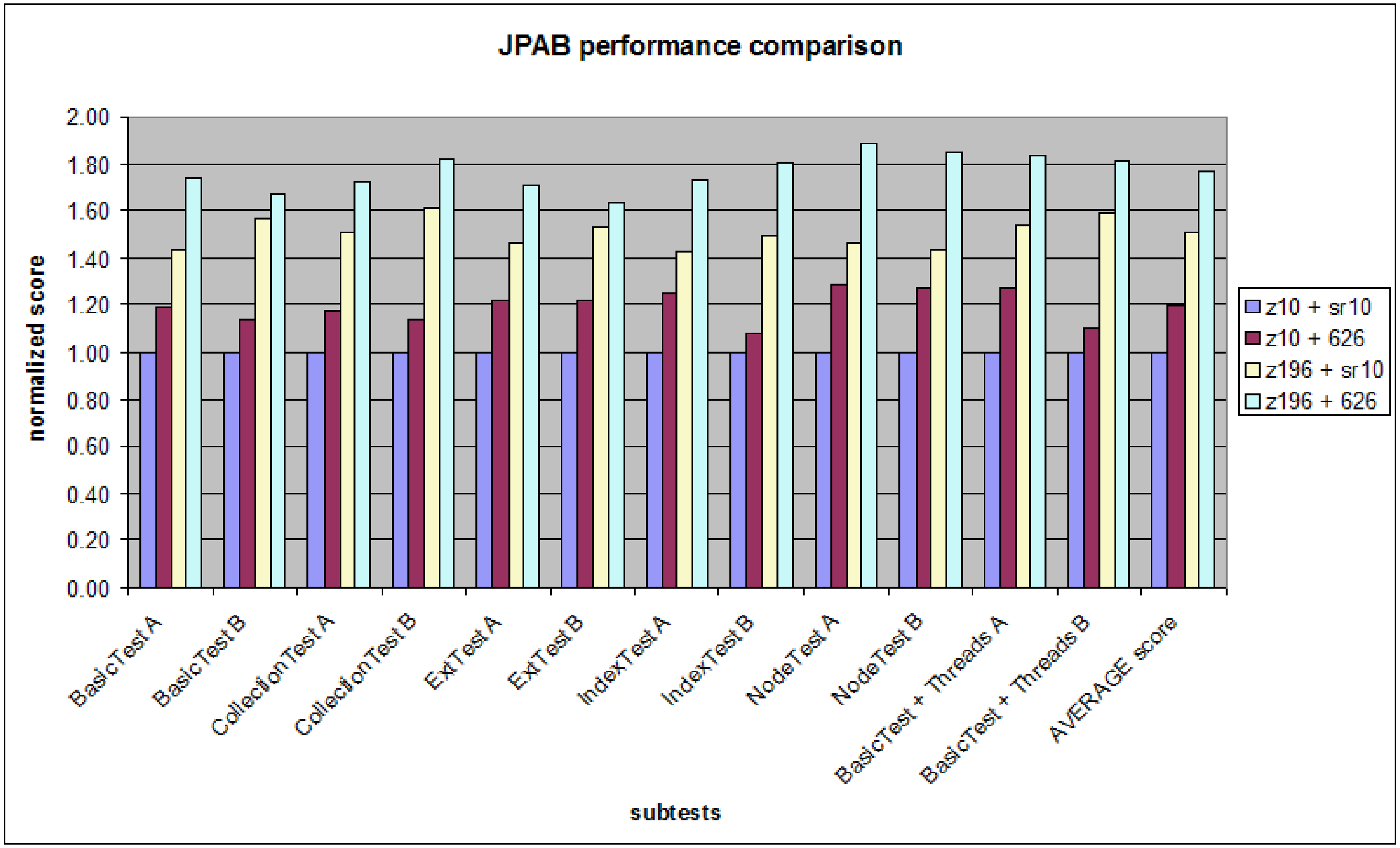


- Upgrading from z10 to z196 improved throughput by as much as 35% using our SOABench webservices benchmark (15% for the 3k/3k payload and 35% for the 10k/10k payload).
- Additionally, upgrading to WAS V8.0 improved performance by another 21% for the 3k/3k payload and 25% for the 10k/10k payload. This increase is a result of improvements to the following areas:
  - JVM and JIT optimizations
  - JAXB fastpath optimizations
- The combine hardware and software improvement is 39% for the 3k/3k case and 69% in the 10k/10k case.

(Controlled measurement environment, results may vary)

Linux on z – Java Persistency API and Java6 R26

- JPAB benchmarks running OpenJPA and HSQLDB (http://www.jpab.org/OpenJPA/HSQLDB/embedded.html)



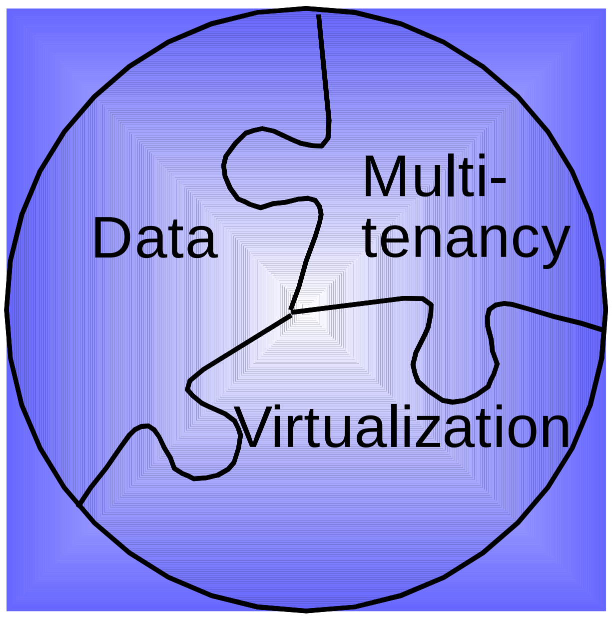
1.77x Aggregate software/ hardware improvement

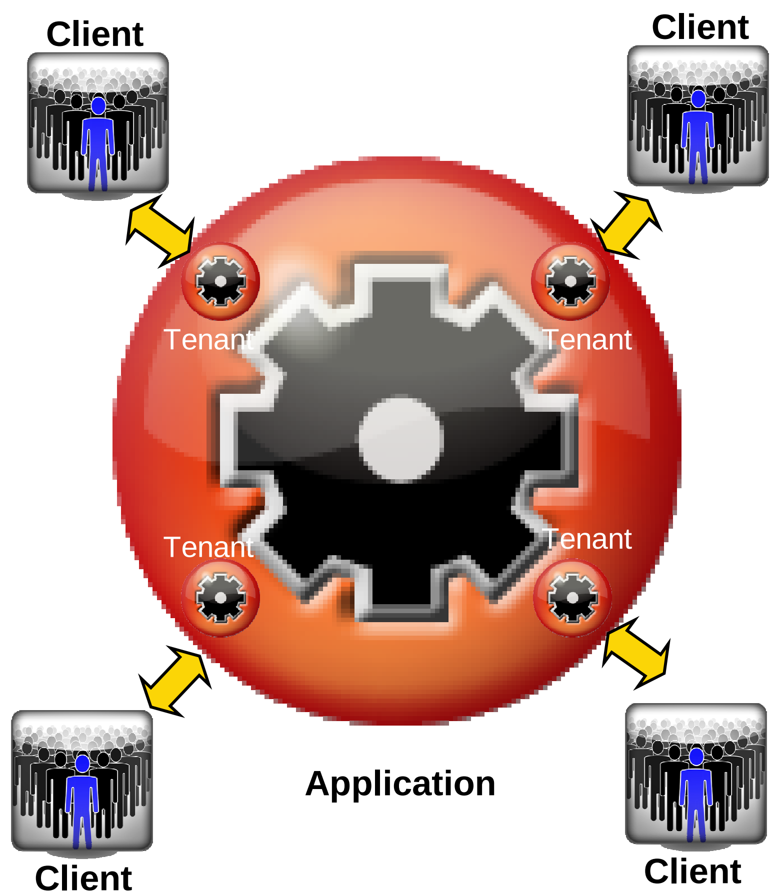
- 19% Java6R26 vs Java6R24 improvement on z10
- 26% Java6R26 vs Java6R24 improvement on z196
- 51% improvement z10 to z196

(Controlled measurement environment, results may vary)

Sneak Peek

- Virtualization and multi-tenancy
- Continued Deep Hardware/Software Synergy
  - zNext
- Continued focus on data/inter-language communication
- Continued focus on improved consumability + performance





Timelines and deliveries are subject to change.

Joran Siu
joransiu@ca.ibm.com


Thank
You

Summary of Links

- Documentation
  - http://www.ibm.com/developerworks/java/jdk/docs.html
- 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
- IBM Support Assistant
  - http://www.ibm.com/software/support/isa/