

Running Java on Linux on System z

Bryan Chan
Java for System z Development, IBM Corporation
bryan.chan@ca.ibm.com

August 14, 2013
Session 13525



Trademarks, Copyrights, Disclaimers

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

Other company, product, or service names may be trademarks or service marks of others.

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2012. All rights reserved.

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

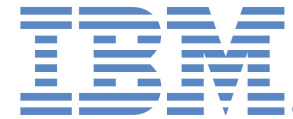
- IBM Java on System z
 - History, overview and roadmap
- IBM J9 Virtual Machine and IBM Testarossa JIT
 - IBM Monitoring and Diagnostic Tools for Java
- Java 7 SR3
 - zEC12 exploitation and performance

IBM and Java



- **Java is critically important to IBM**

- Infrastructure for IBM's software portfolio
 - WebSphere, Lotus, Tivoli, Rational, Information Management



- **IBM is investing strategically for Java in Virtual Machines**

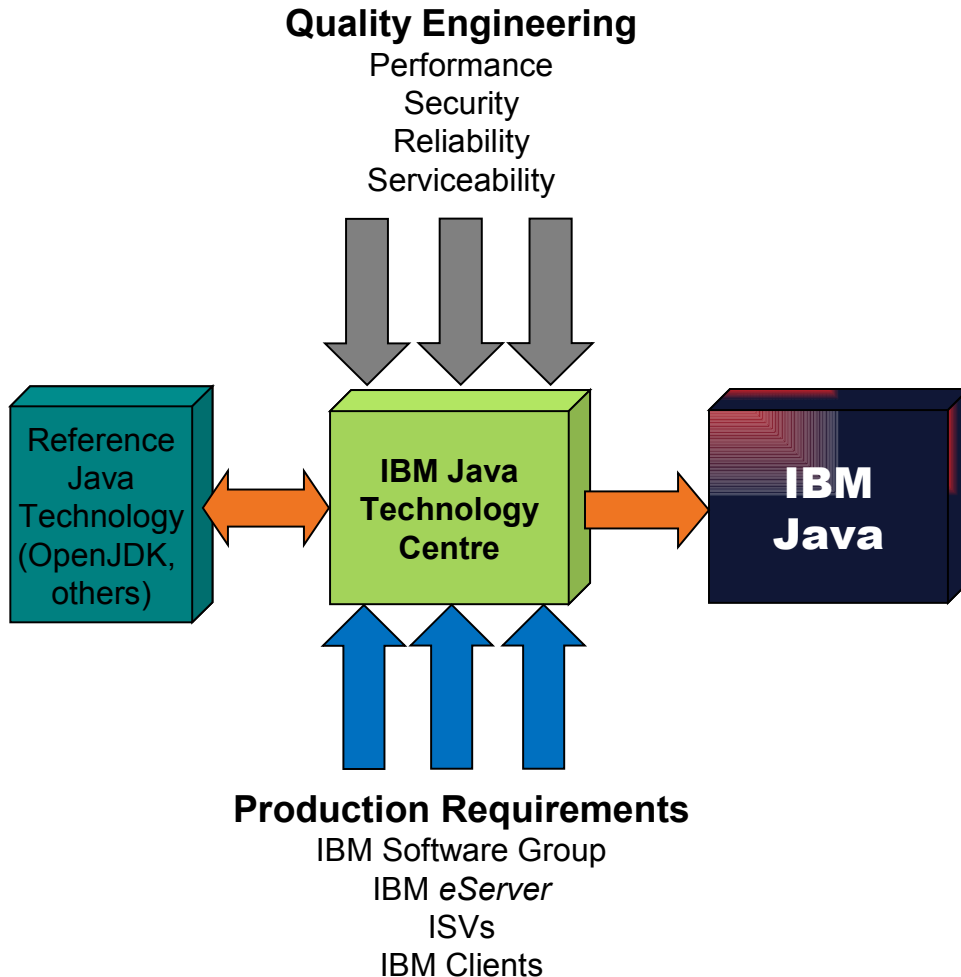
- Since Java 5.0, a single JVM supports multiple configurations
 - Java ME, Java SE, Java EE
- New technology base (J9/Testarossa) on which to deliver improved performance, reliability, serviceability

- **IBM also invests and supports public innovation in Java**

- OpenJDK, Eclipse, Apache (XML, Aries, Derby, Geronimo, Harmony, Tuscany, Hadoop,...)
- Broad participation in relevant open standards (JCP, OSGi)

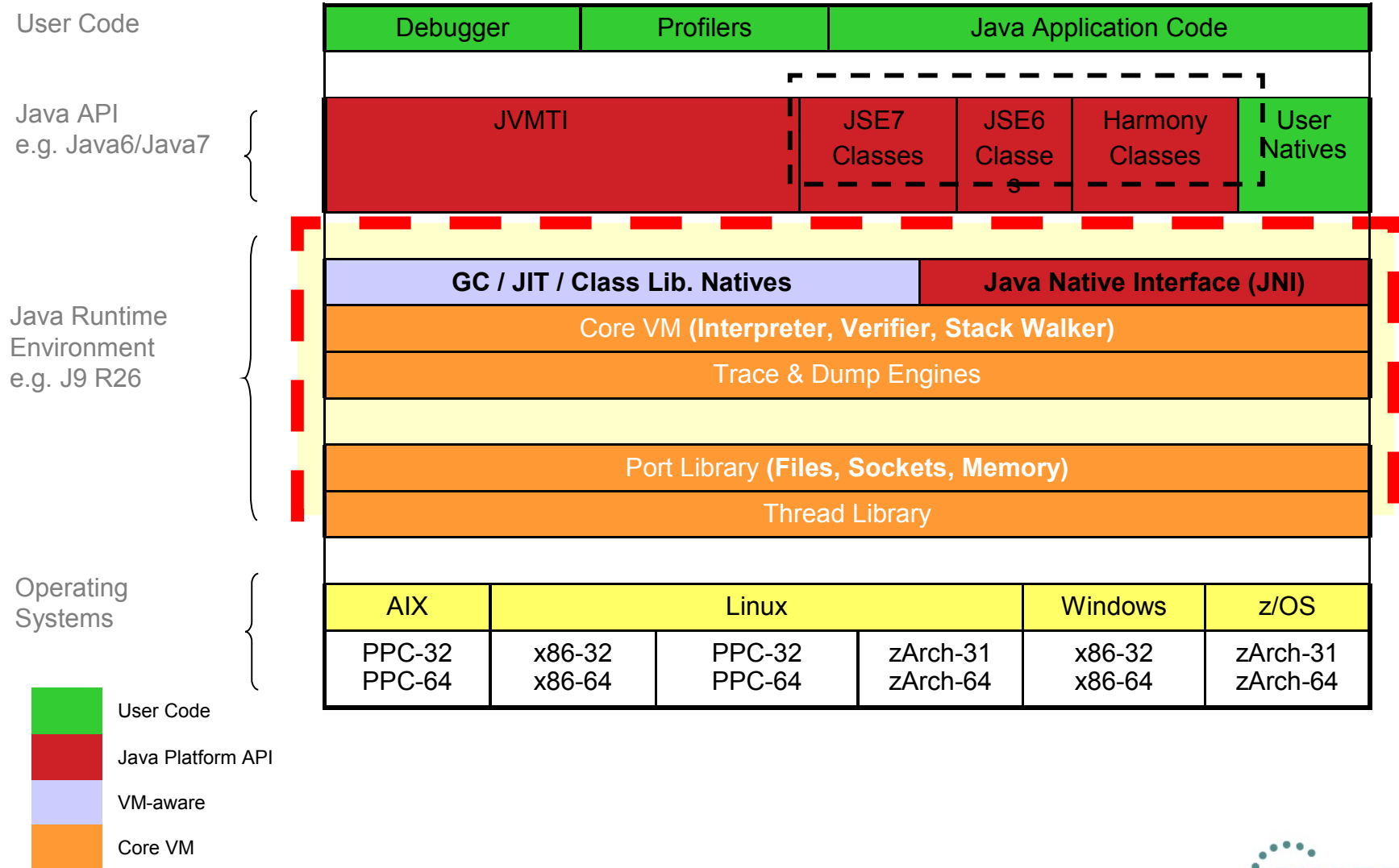


IBM's Approach to Java Technology



- ✓ *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*

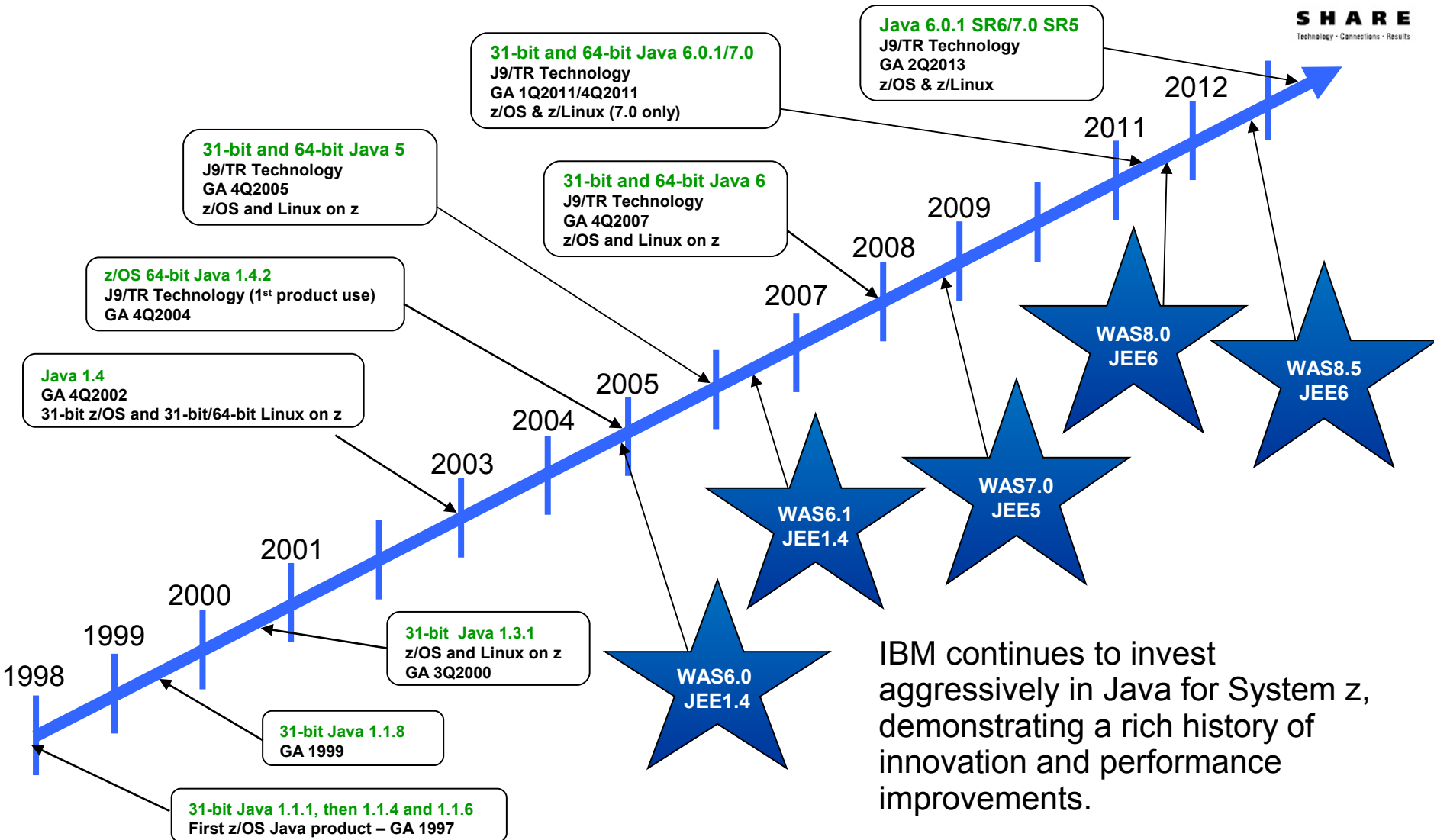
JVM Architectural Overview



Differences between Oracle and IBM Java

- Both use the same reference implementation of Java Class Libraries (e.g. OpenJDK)
- Key differences
 - Security: Standards do not impose strong separation of interest
 - ORB: OMG CORBA standard rules
 - XML: Xerces/Xalan shipped by both vendors since Java 5, although different levels may be used
- IBM J9/Testarossa runtime vs. Oracle HotSpot
 - Different tuning and controls for JVM, JIT and GC
 - Tooling is distinct (e.g. IBM 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



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



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
- Full Speed Debug
- Hot Code Replace
- 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 6.0.1/7.0 (J9 R26)

- Improvements in
 - Performance
 - GC Technology
- z196™ Exploitation
 - OOO Pipeline
 - 70+ New Instructions
- JZOS/Security Enhancements

IBM Java 7.0 SR5/Next**

- Improvements in
 - Performance
 - GC Technology
- zEC12™ Exploitation
 - Transactional Execution
 - Runtime Instrumentation
 - Flash 1Meg pageable LPs
 - 2G large pages
 - Hints/traps
- Data Access Accelerator
- **Cloud:** Multi-tenancy/Virtualization



Java 7.0: What to look for

- **New I/O**
 - Data mining and analytics workloads are increasingly I/O-intensive
 - Asynchronous I/O offers significant performance and footprint gains
- **Concurrency Libraries**
 - Exploit larger multi-core systems (e.g. next-generation POWER and System z) to provide better scalability, higher throughput and lower TCO via 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

Java 8 Beta Program

- **Provides Java SE 8 compatibility, while exploiting the unique capabilities of IBM platforms to achieve performance and usability improvements**
 - To provide early technology access during the development cycle
 - To assist Java 8 in satisfying customer requirements
 - To provide feedback to IBM
- **New in IBM SDK, Java Technology Edition, Version 8:**
 - Compatibility with the new Java SE 8
 - Leveraging new IBM hardware (e.g. IBM zEnterprise EC12)
 - Improved performance for workload optimized runtimes, which delivers better application throughput without changes to application code
 - Enhanced support for Cloud & Multi-tenancy environments
 - Improved efficiency of manipulating native data records/types directly from Java code
- **Managed and Open Beta**
 - <http://www.ibm.com/developerworks/java/jdk/beta/index.html>

Java 8: Language Innovation – Lambdas

- **New syntax to allow concise code snippets/expressions**
 - Useful for sending code to `java.lang.concurrent`
 - On the path to enabling more parallelism

```
Collections.sort(people, new Comparator<Person>() {  
    public int compare(Person x, Person y) {  
        return x.getLastName().compareTo(y.getLastName());  
    }  
})
```



```
Collections.sort(people, Collections.comparing(  
    (Person p) -> p.getLastName()));
```

http://www.dzone.com/links/presentation_languagelibraryvm_coevolution_in_jav.html

Java 8: Language Innovation – Lambdas

- **New syntax to allow concise code snippets/expressions**
 - Useful for sending code to `java.lang.concurrent`
 - On the path to enabling more parallelism

```
Collections.sort(people, new Comparator<Person>() {  
    public int compare(Person x, Person y) {  
        return x.getLastName().compareTo(y.getLastName());  
    }  
})
```



```
people.sort(comparing(Person::getLastName));
```

http://www.dzone.com/links/presentation_languagelibraryvm_coevolution_in_jav.html

Java 8: Data Access Accelerator

- **A Java library for bare-bones data conversion and arithmetic**
 - Operates directly on byte arrays
 - Avoids expensive Java object instantiation
 - Orchestrated with JIT for deep platform opts
 - Library is platform- and JVM-neutral

- **Current approach**

```
byte[] addPacked(byte a[], byte b[]) {
    BigDecimal a_bd = convertPackedToBd(a);
    BigDecimal b_bd = convertPackedToBd(b);
    a_bd.add(b_bd);
    return (convertBDtoPacked(a_bd));
}
```

- **Proposed Solution**

```
byte[] addPacked(byte a[], byte b[]) {
    DAA.addPacked(a, b);
    return a;
}
```

Marshalling and Unmarshalling

- Transforms byte arrays ↔ Java variables
- Supports both big-endian and little-endian byte arrays

Packed Decimal (PD) Operations

- Arithmetic: +, -, *, /, %
- Logical: >, <, >=, <=, ==, !=
- Validation: verifies if a PD operand is well-formed
- Others: optimized shifts, moves on PD operand

Decimal Type Conversions

- Decimal ↔ Primitive
 - Convert Packed Decimal (PD), External Decimal (ED) and Unicode Decimal (UD) ↔ primitive types (int, long)
- Decimal ↔ Decimal
 - Convert between decimal types (PD, ED, UD)
- Decimal ↔ Java
 - Convert decimal types ↔ BigDecimal/BigInteger objects

Looking Ahead: PackedObjects with IBM Java

PackedObjects

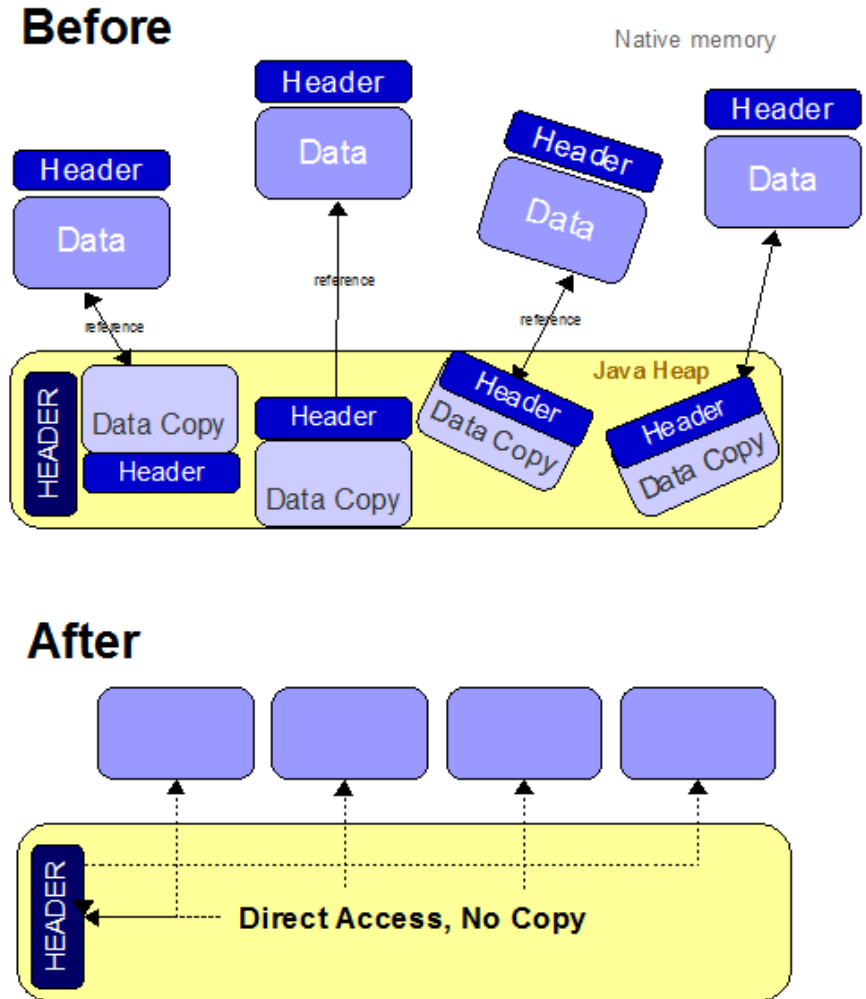
Experimental feature in the IBM JVM. Introduces a new Java type that implements an explicit object model which tightly packs fields allowing for natural and efficient direct mapping of structured data.

Goals

- Allow for explicit source-level representation of structured data in Java
- Improve serialization and I/O performance
- Allow direct access to “native” (off-heap) data

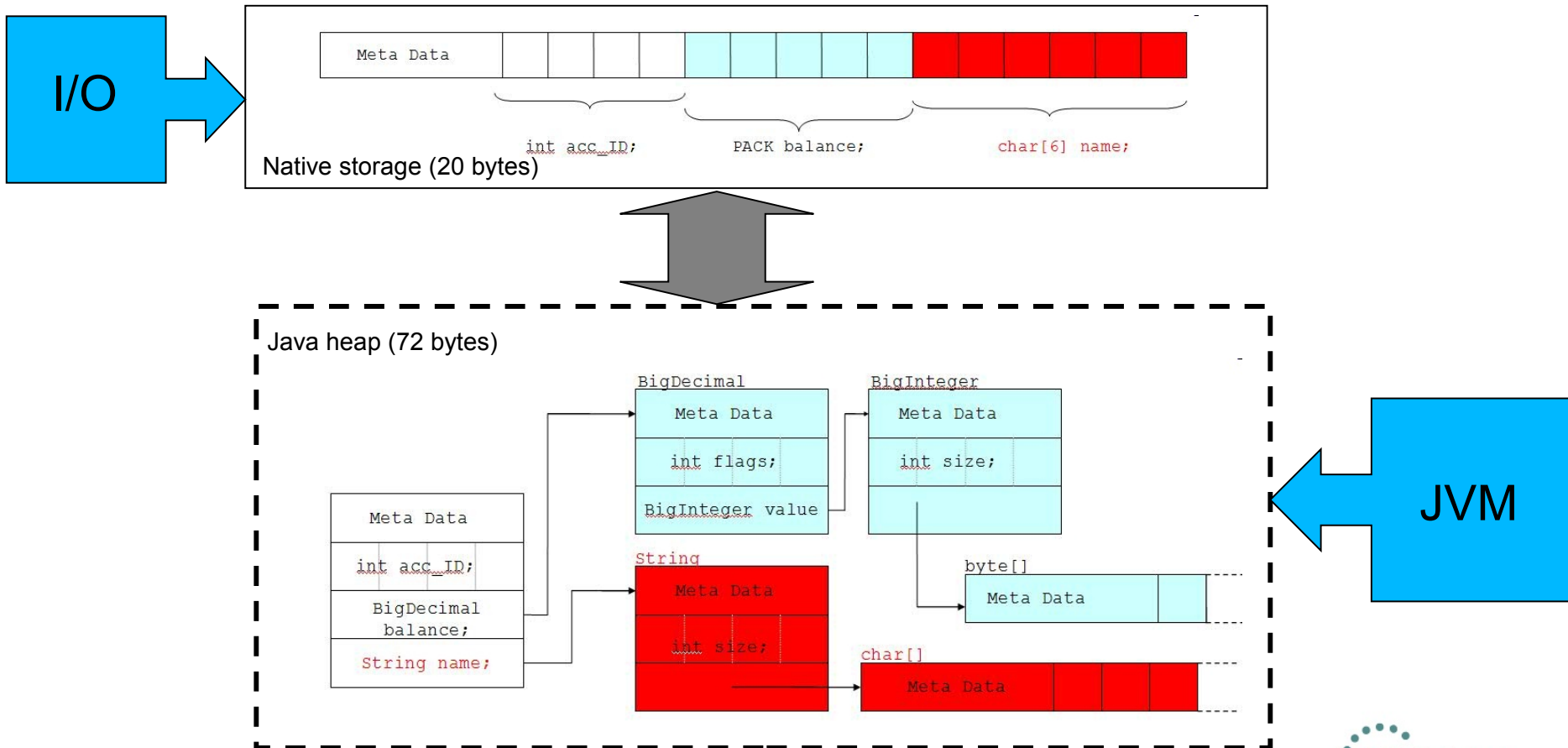
<http://www.slideshare.net/mmitran/ibm-java-packed-objects-mmit-20121120>

<http://duimovich.blogspot.ca/2012/11/packed-objects-in-java.html>



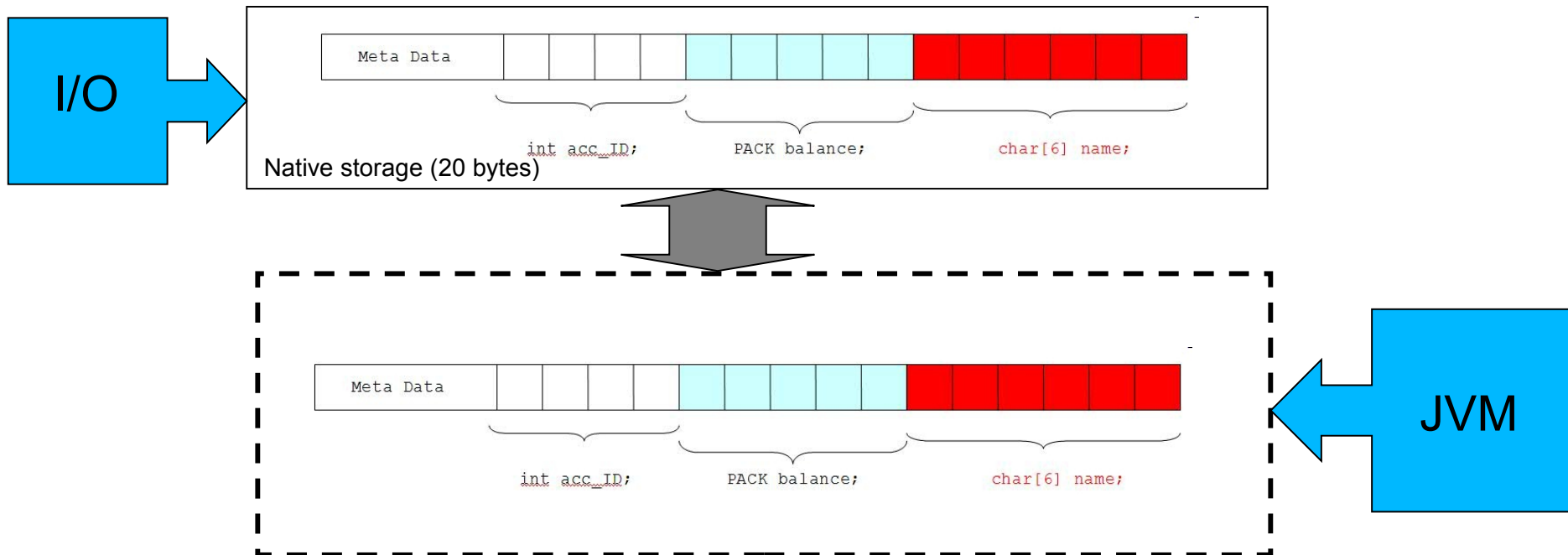
JVM only speaks “Java”

- Data typically must be copied/re-formatted to/from Java heap
- Costly in path length and footprint



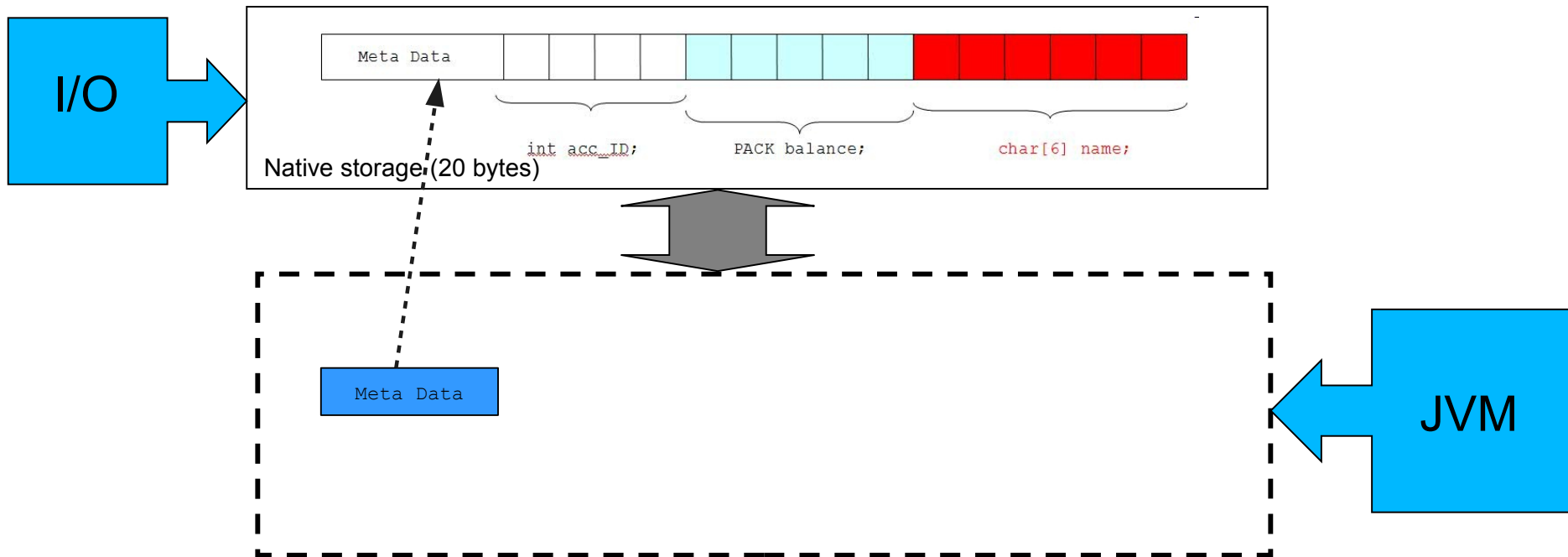
On-Heap PackedObject

- Allows controlled layout of storage of data structures on the Java heap
 - Reduces footprint of data on Java heap
 - No (de)serialization required



Off-Heap PackedObject

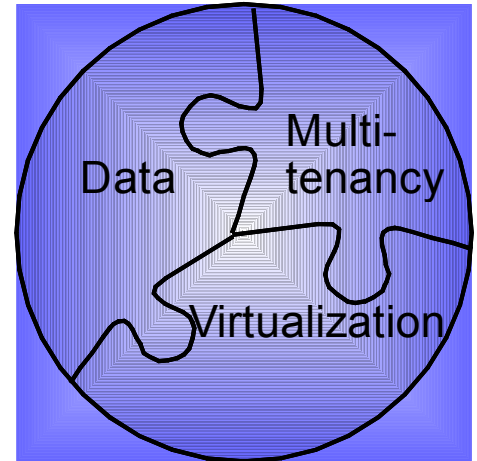
- Enable Java to talk directly to the native data structure
 - Avoid overhead of data copy onto/off Java heap
 - No (de)serialization required



Looking Ahead: Cloud with IBM Java

- **Multi-tenancy support will allow multiple applications to run in a single shared JVM for high-density deployments**

- *Win*: Footprint reduction enabled by sharing runtime and JVM artifacts while enforcing resource consumption quotas
- *Platform Coverage*: 64-bit, balanced GC policy only
- *Ergonomics*: Single new command-line flag (**-Xmt = multi-tenancy**)



- **Hypervisor, Virtual Guest, and Extended-OS JMX Beans**

- Allows applications to detect and identify the installed hypervisor and query attributes of LPAR
- Provides richer access to operating system performance statistics

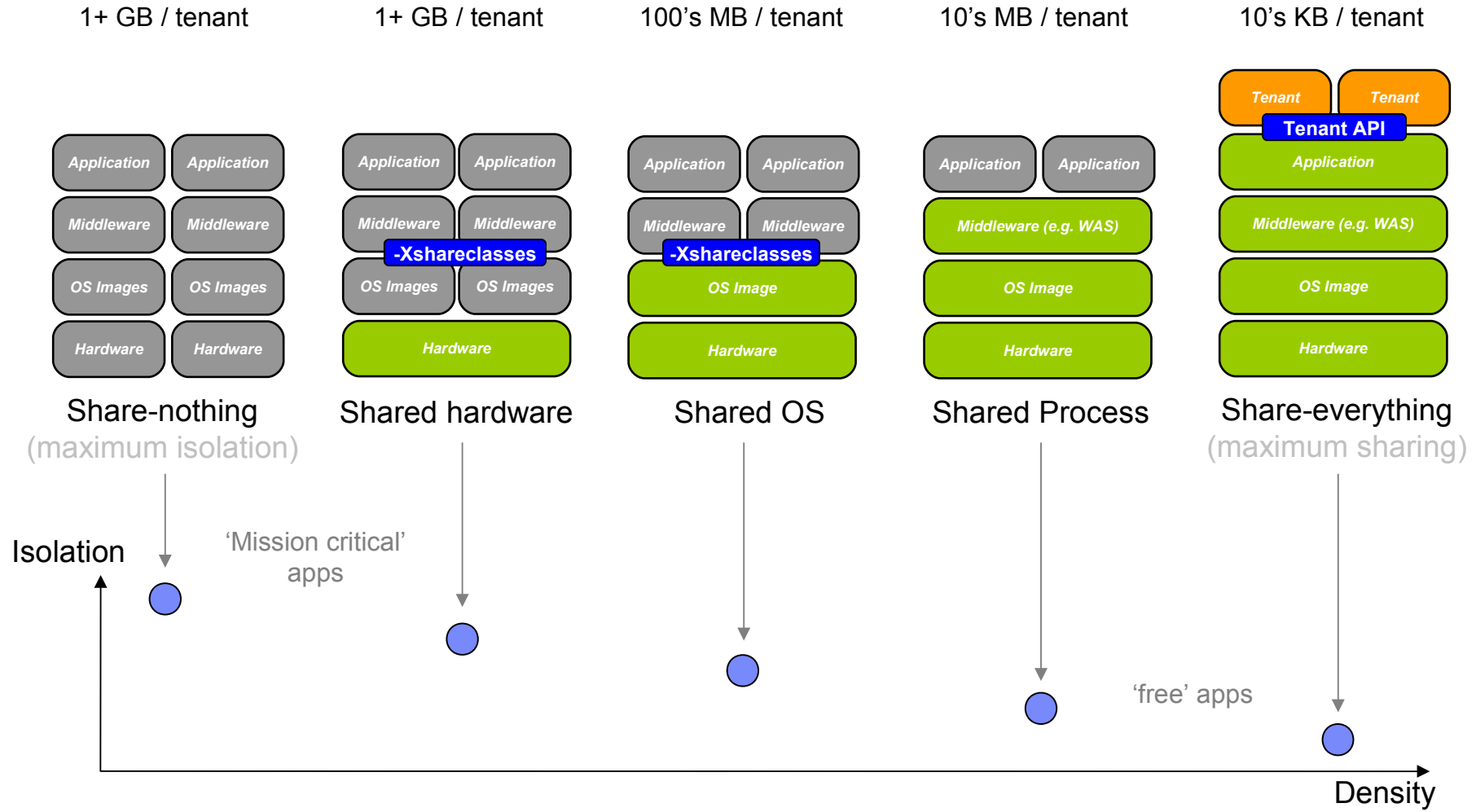
Timelines and deliveries are subject to change.

19 Complete your sessions evaluation online at SHARE.org/BostonEval

Looking Ahead: Cloud with IBM Java

- **Runtime adjustable heap size (-Xsoftmx)**
 - JMX beans allow for dynamically adjusting heap size
 - Allows users to take advantage of hot-add of memory in virtualized environments
 - Available in Java 7 SR3
- **JIT support for “deep idle” state**
 - Enabled with **-Xtune:virtualized** (Java 7 SR4)
 - Reduces CPU cycles used by the JIT during idle periods
 - Important for dense virtualized System z environments
 - Early results with WAS Liberty show ~2x to ~6x reduction

Multi-tenancy: Isolation and Density



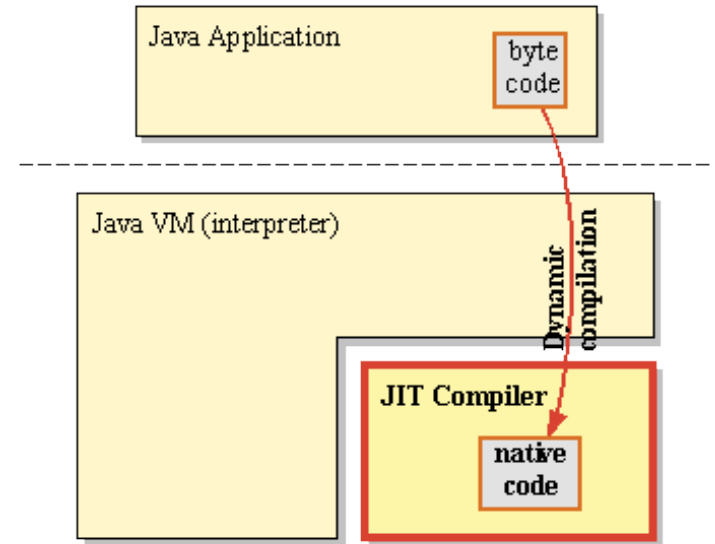
Timelines and deliveries are subject to change.

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 – zEC12, z196, z10, z9, z990
- Millions of installations of J9 VM and Testarossa compiler!

IBM Testarossa JIT Compiler – Introduction

- IBM's production JIT on all platforms since Java 5
- Developed at the IBM Toronto Lab
 - 30+ years of expertise in compilation and optimization technologies
- Close relationship 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)

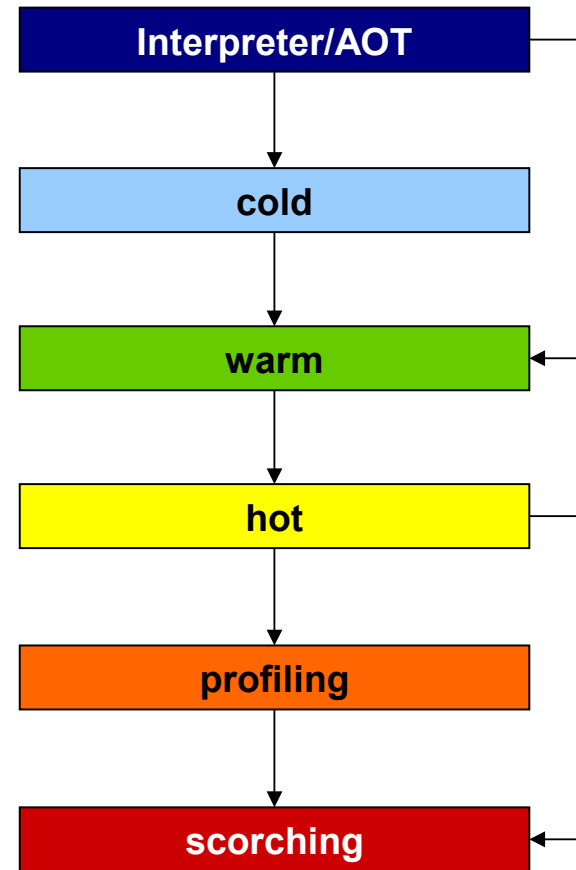


IBM Testarossa JIT Compiler – Features

- **Dynamic**
 - Triggered at run time based on projected profitability
 - Compiled methods can be freely mixed with interpreted callers/callees
 - Multiple versions of methods compiled at different optimization levels
- **Adaptive**
 - Sensitive to program's need for CPU
 - Runs on asynchronous thread, throttled during start-up
 - Profile program and generate tailored, re-optimized code
- **Optimizing**
 - Comprehensive collection of conventional optimizations
 - Control flow simplification, data flow analysis, etc.
 - Speculative and Java-specific optimizations
 - Call devirtualization, partial inlining, lock coarsening, etc.
 - Deep exploitation of System z micro-architecture

IBM Testarossa JIT – Compilation Strategy

- Focus compilation CPU time where it matters
 - Stagger investment over time to amortize cost
- Methods start being interpreted
 - Interpreter profiling improves initial compilations
- After N invocations, methods get compiled at “warm”
- Identify hot methods by sampling
 - Re-compile methods at “hot” or “scorching”
- Transition to “scorching” via temporary profiling step
 - Direct global optimizations using profiled data
 - Identify hot paths through method
 - Register allocation, Branch straightening, etc.
 - Profile values and types
 - Specialize and version hot paths
 - Profile virtual calls
 - Inline hot targets



IBM Testarossa JIT – System z Support

- Idioms are recognized in Java bytecodes
- Bytecodes converted to CISC instructions**
- Examples of CISC instructions
 - TROT, TRTO, TRTT, TROO
(“TR” = translate, “O” = one byte, “T” = two bytes)
 - SRST (search string)
 - MVC (move characters)
 - XC (exclusive or)
 - CLC (compare logical)



** 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 Testarossa JIT – System z Support

- Example

```
while (i < end) {  
    char c = table[B[i]];  
    if (c == terminal_char)  
        break;  
    A[i] = c;  
    ++i;  
}
```

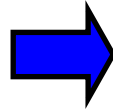


```
        L    R0, terminal_char  
        LA   R1, @table  
        L    R2, @A  
        LR   R3, end  
        L    R4, @B  
LOOP:   L    R5, (R4)  
        L    R5, (R5,R1)  
        AHI  R4, 1  
        CR   R5, R0  
        BRC  0, END  
        S    R5, (R2)  
        AHI  R2, 2  
        AHI  R3, -1  
        BRC  2, LOOP  
END:
```

IBM Testarossa JIT – System z Support

- Example

```
while (i < end) {  
    char c = table[B[i]];  
    if (c == terminal_char)  
        break;  
    A[i] = c;  
    ++i;  
}
```



```
L    R0, terminal_char  
LA   R1, @table  
L    R2, @A  
LR   R3, end  
L    R4, @B  
* Choice of 'xx' depends on  
* types of arrays A and B  
LOOP: TRxx R2, R4  
      BRC 1, LOOP
```

IBM J9 Garbage Collector Family

Policy	Recommended usage	Notes
<code>optThroughput</code>	optimized for throughput	default in Java 5 and Java 6
<code>optAveragePause</code>	optimized to reduce pause times	
<code>gencon</code>	optimized for transactional workloads	default in Java 6.0.1/Java 7
<code>subPools</code>	optimized for large MP systems	deprecated in Java 6.0.1/Java 7
<code>balanced</code>	optimized for large heaps	added in Java 6.0.1/Java 7

- Why have many policies? Why not just “the best?”
 - Cannot always dynamically determine what trade-offs the user/application are willing to make

Pause time vs. Throughput

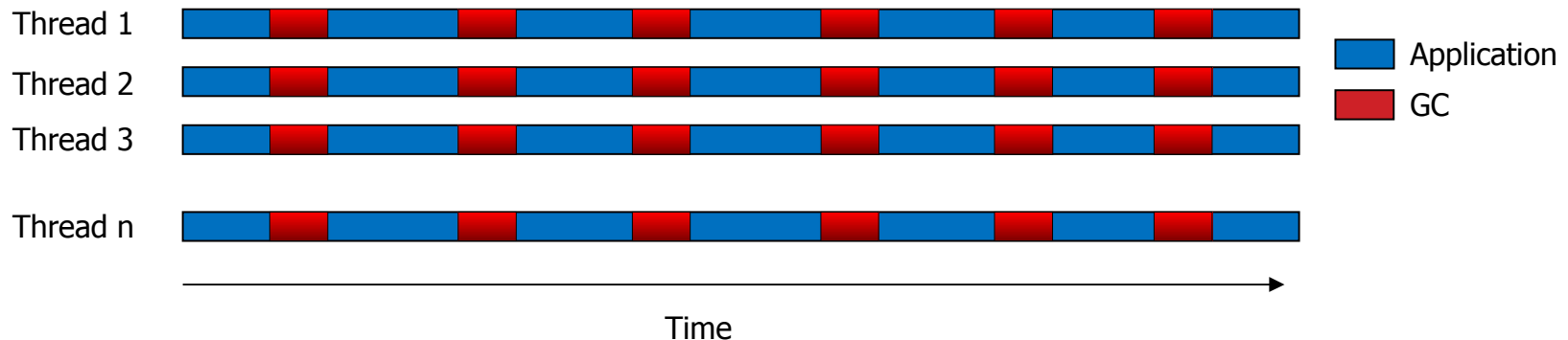
- Trade off frequency and length of pauses vs. throughput

Footprint vs. Frequency

- Trade off smaller footprint vs. frequency of GC pauses/events

IBM J9 Garbage Collector: **-Xgcpolicy:optthruput**

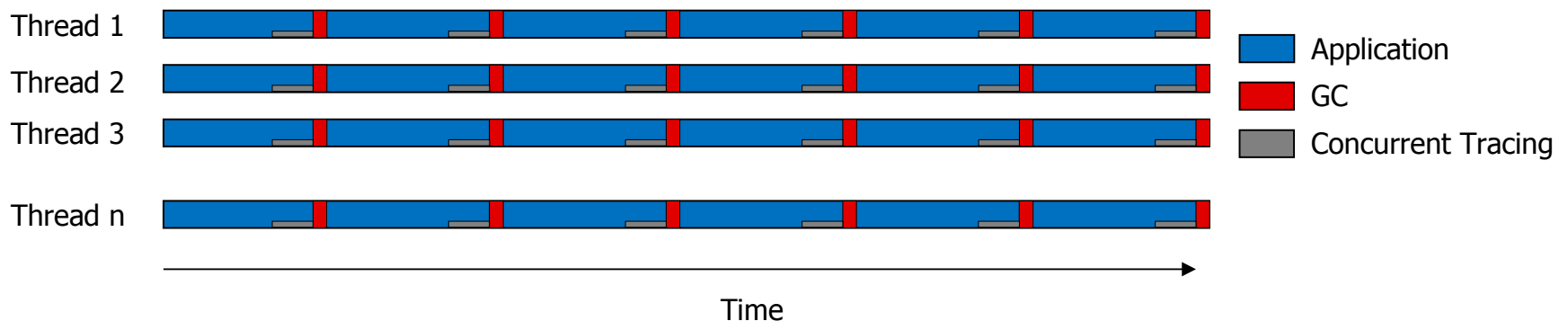
- Default policy in Java 5 and Java 6
- Used where raw throughput is more important than short GC pauses
- Application stopped whenever garbage is collected



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
- Application paused for shorter periods



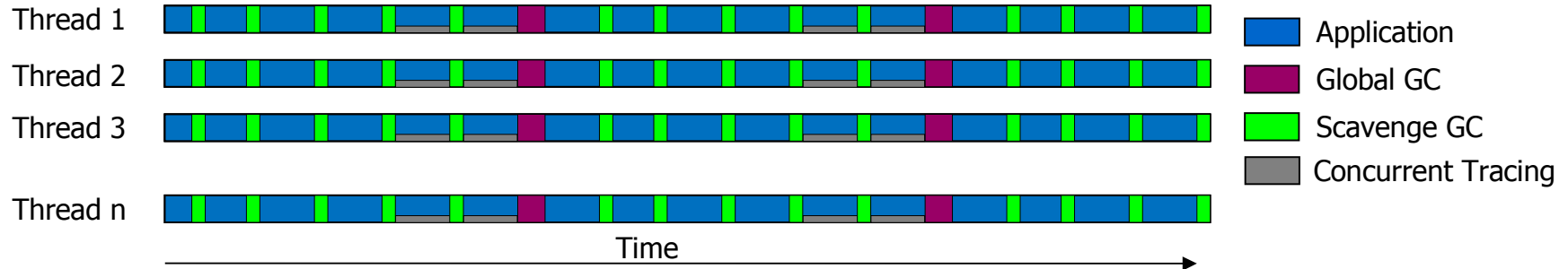
Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

IBM J9 Garbage Collector: **-Xgcpolicy:gencon**

- **Best of both worlds**
 - Good 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?**
 - Objects die young in most workloads
 - Generational GC allows a better ROI (less effort, better reward)
 - Performance is close to or 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 Java 6.0.1 and Java 7
- Applications with many short-lived objects benefit from shorter pause times 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.

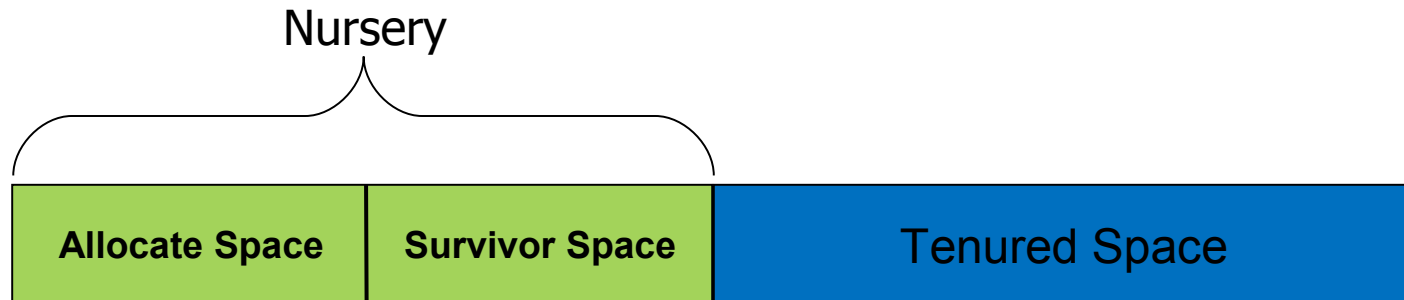
IBM J9 Garbage Collector: **-Xgcpolicy:gencon**

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



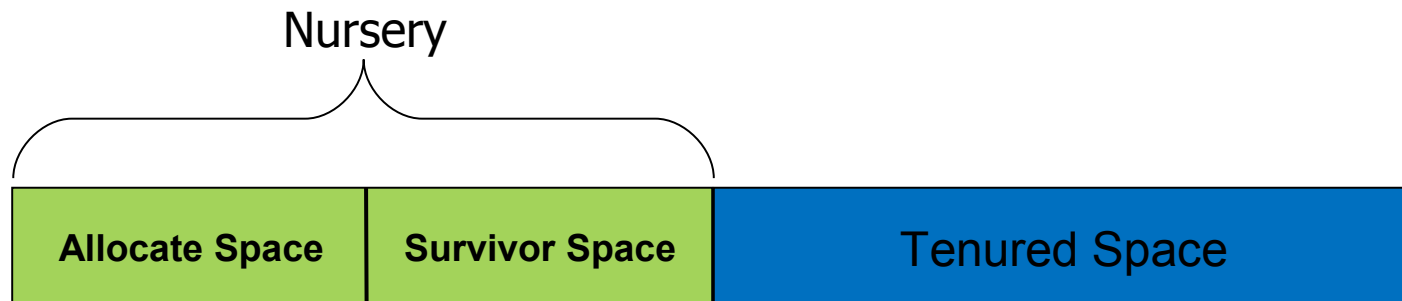
IBM J9 Garbage Collector: **-Xgcpolicy:gencon**

- Nursery is further split into two spaces
 - **allocate** and **survivor**
 - Division dynamically adjusted according to survival rate



IBM J9 Garbage Collector: **-Xgcpolicy:gencon**

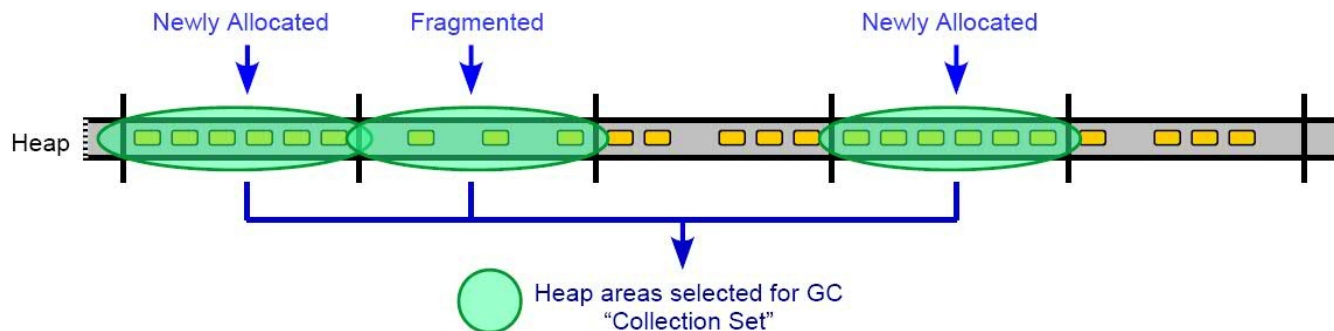
- A **scavenge** copies objects from allocate space to survivor space
 - Less heap fragmentation
 - Better data locality
 - Faster future allocations
- If an object survives X number of scavenges, it is promoted to tenured space



IBM J9 2.6 Enhancement: **-Xgcpolicy:balanced**

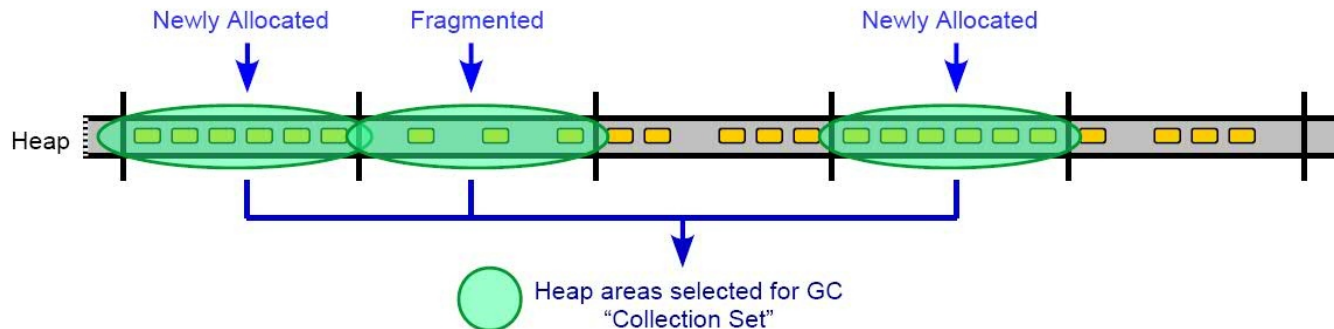
- **Improved application responsiveness**

- 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



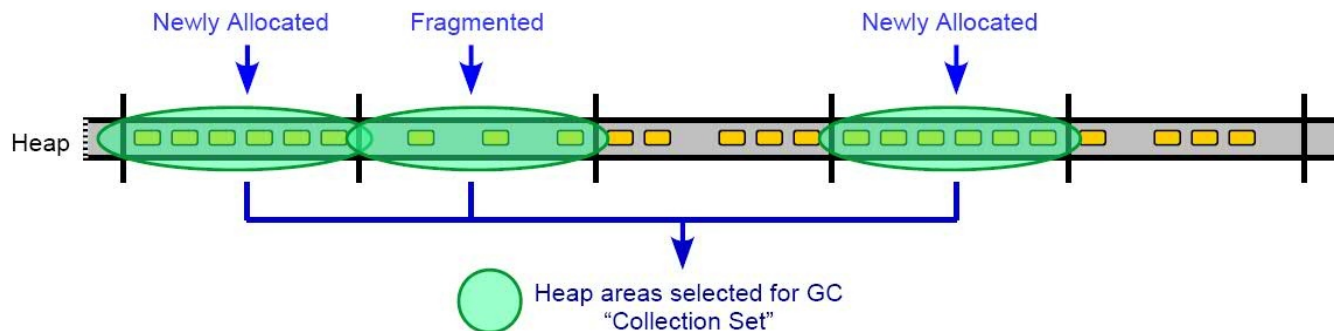
IBM J9 2.6 Enhancement: **-Xgcpolicy:balanced**

- **Next-generation technology expands platform exploitation possibilities**
 - Virtualization: group heap data by frequency of access, direct OS paging decisions
 - Dynamic re-organization of data structures to improve memory hierarchy utilization



IBM J9 2.6 Enhancement: **-Xgcpolicy:balanced**

- 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

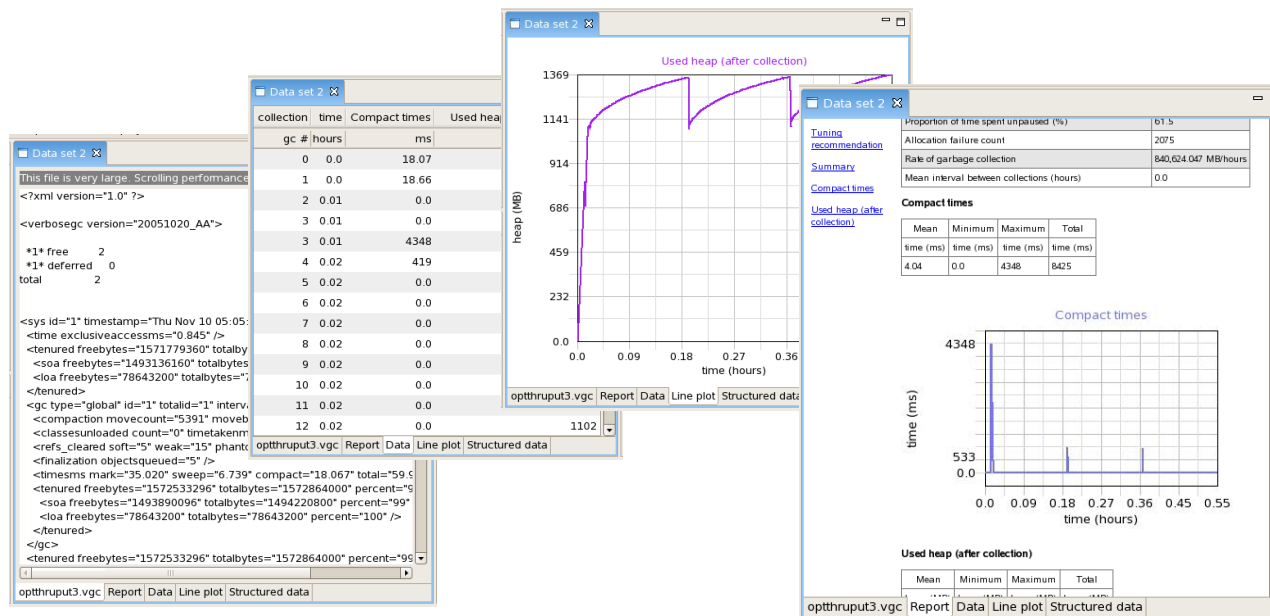


IBM J9 Garbage Collector: Tuning

- Typical approach
 - Pick a policy based on desired application behavior
 - Monitor GC behavior; overhead should be no more than 10%
 - Tune heap sizes (**-Xms**, **-Xmx**)
 - Tune helper threads (**-Xgcthreads**)
 - Many other knobs exist
- Best practices
 - Avoid finalizers
 - Don't use `System.gc()`

IBM J9 Garbage Collector: Tuning

- IBM Garbage Collection and Memory Visualizer (GCMV)
 - Uses **-verbose:gc** output to provide detailed view of Java memory footprint and GC behavior
 - Uses **ps -p \$PID -o pid,vsz,rss** output to plot native footprint



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>
- Memory leaks are possible even with GC
 - Detect large objects/object cycles with IBM Memory Analyzer

What is IBM Support Assistant?

- **ISA Workbench**
 - 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 production monitoring tool



Find Information
Easily find the information you need including product specific information and search capabilities.

Analyze Problem
Diagnose and analyze problems through serviceability tools, collection of diagnostic artifacts, and guidance through problem determination.

Manage Service Request
Effectively submit, view and manage your service requests enhanced with automated collection of diagnostic data.

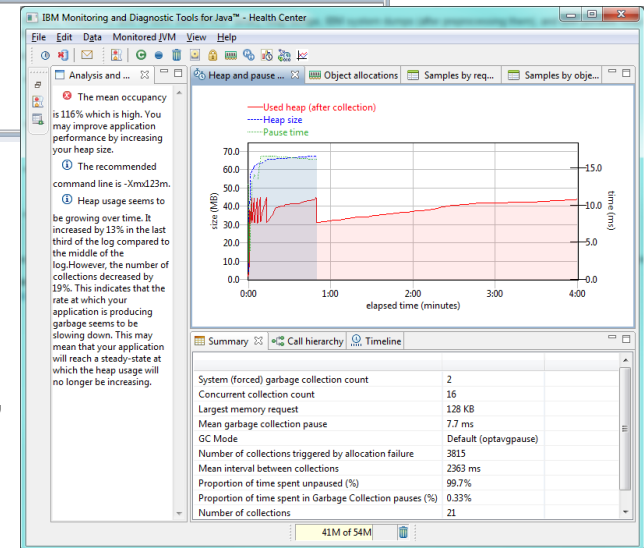
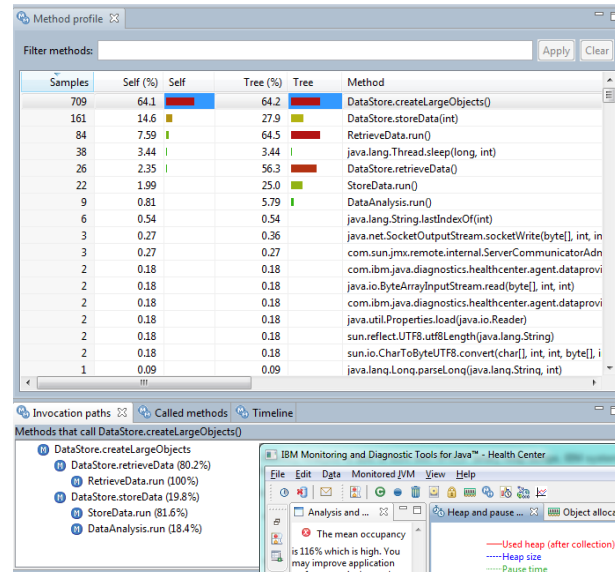
IBM Monitoring and Diagnostic Tools for Java: Health Center

• 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 JVM options?

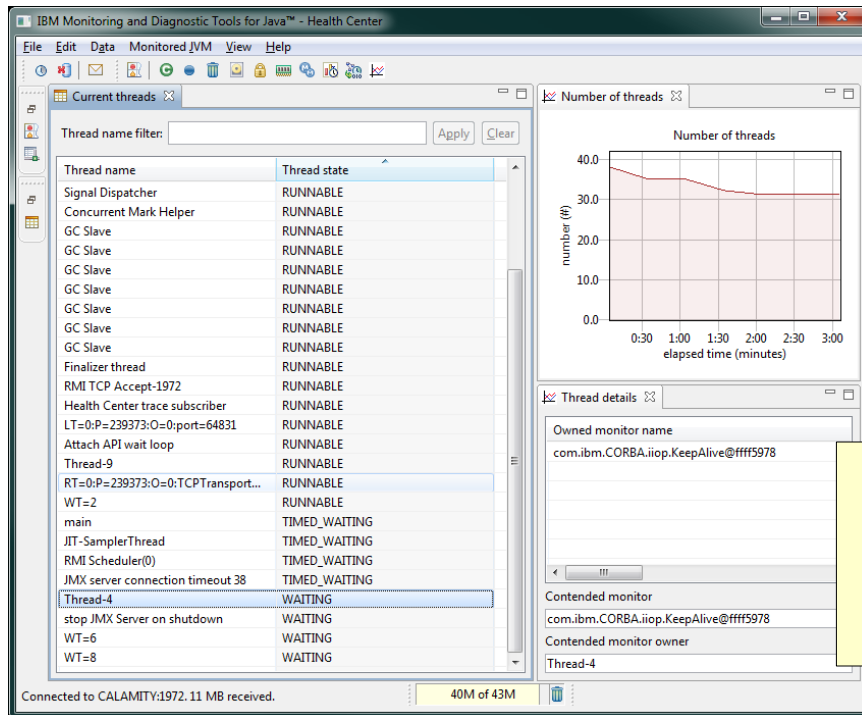
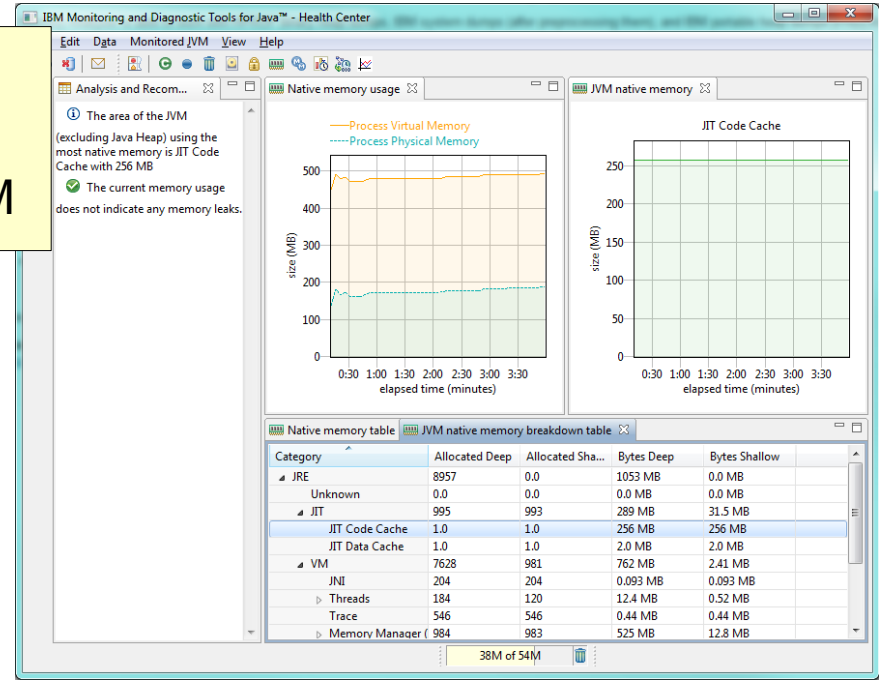
• Health Center Overview

- Lightweight monitoring tool with very low overhead
- Understands how your application behaves and offers recommendations for potential problems
- Features: GC visualization, method profiling/tracing, thread monitoring, class loading history, lock analysis, file I/O and native memory usage tracking
- Suitable for all applications running on IBM JVMs



IBM Monitoring and Diagnostic Tools for Java: Health Center

Track available system memory and native memory used by the JVM



Keep track of running threads and monitor contention

IBM Monitoring and Diagnostic Tools for Java: GCMV

- **What problem am I solving?**
 - How is the garbage collector behaving?
Can I do better?
 - How much time is GC taking?
 - How much free memory does my JVM have?
- **GCMV Overview**
 - Analyzes Java verbose GC logs and provides insight into application behavior
 - Visualize a wide range of GC data and Java heap statistics over time
 - Provides the means to detect memory leaks and to optimize garbage collection
 - Uses heuristics to make recommendations and guide user in tuning GC performance

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.

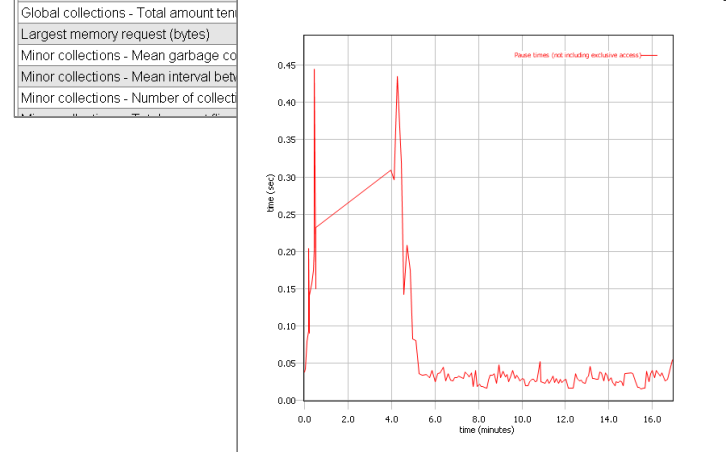
⚠ The garbage collector is performing system (forced) GCs. 5 out of 145 collections (3.448%) were triggered by System.gc() calls. The use of System.gc() is generally not recommended since they can cause long pauses and do not allow the garbage collection algorithms to optimise themselves. Consider inspecting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



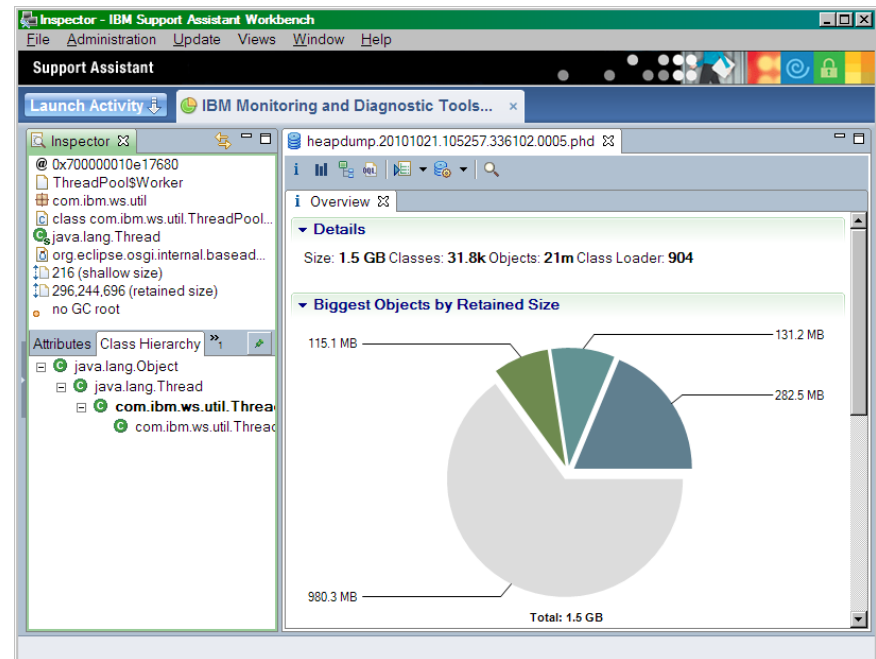
IBM Monitoring and Diagnostic Tools for Java: Memory Analyzer

- **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?

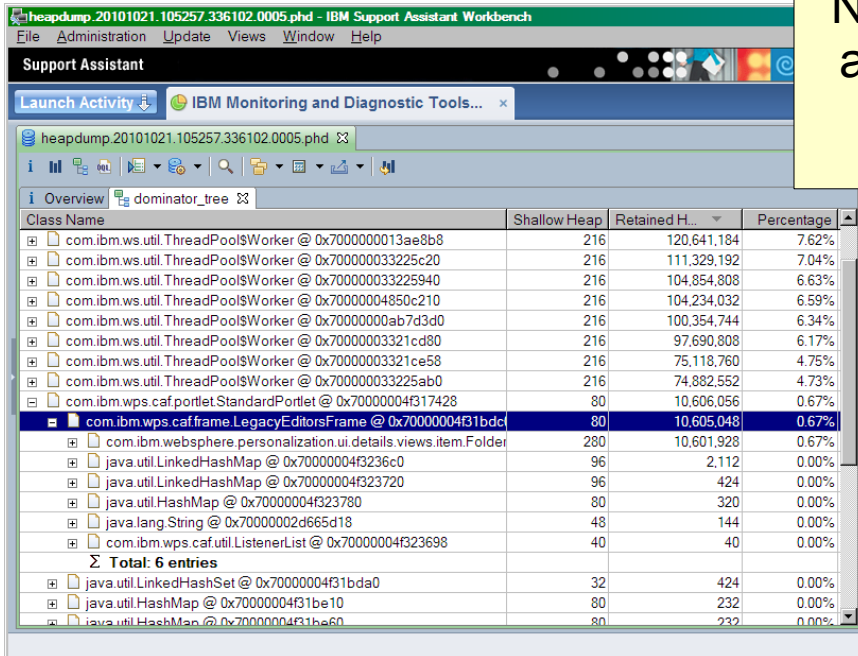
- **Memory Analyzer Overview**

- Examines memory dumps and identifies Java memory leaks
- Analyzes footprint and provides insight into wasted space
- Features: visual objects by size/class/classloader, dominator tree analysis, path to GC roots analysis, object query language (OQL)
- Works with IBM system dumps, IBM portable heap dumps as well as Oracle HPROF binary heap dumps
- IBM Extensions for Memory Analyzer offer additional, product-specific capabilities



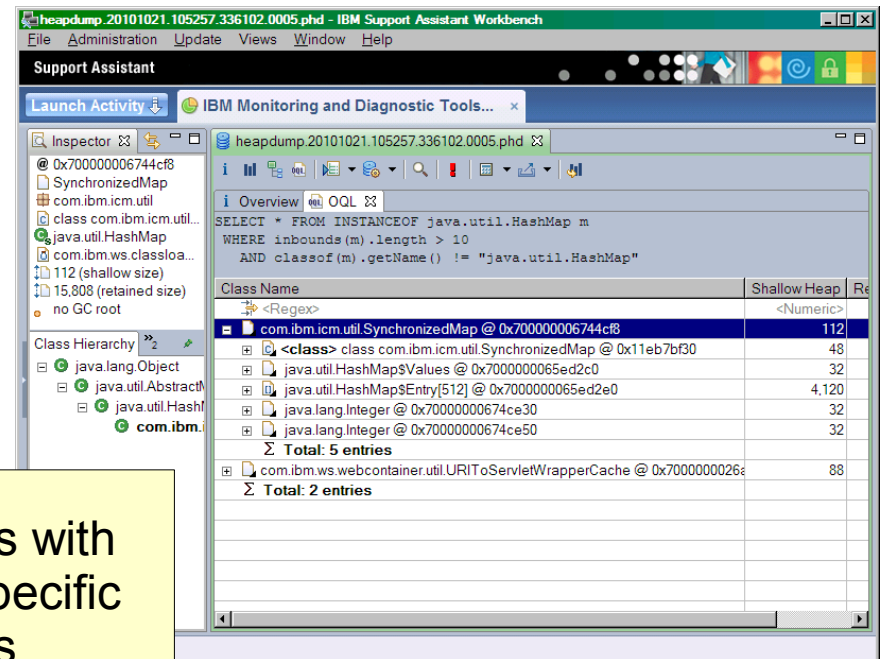
IBM Monitoring and Diagnostic Tools for Java: Memory Analyzer

Navigate the dominator tree, and find which objects keep which other objects alive



Class Name	Shallow Heap	Retained H...	Percentage
com.ibm.ws.util.ThreadPool\$Worker @ 0x7000000013ae8b8	216	120,641,184	7.62%
com.ibm.ws.util.ThreadPool\$Worker @ 0x7000000033225c20	216	111,329,192	7.04%
com.ibm.ws.util.ThreadPool\$Worker @ 0x7000000033225940	216	104,854,808	6.63%
com.ibm.ws.util.ThreadPool\$Worker @ 0x700000004850c210	216	104,234,032	6.59%
com.ibm.ws.util.ThreadPool\$Worker @ 0x70000000ab7d3d0	216	100,354,744	6.34%
com.ibm.ws.util.ThreadPool\$Worker @ 0x700000003321cd80	216	97,690,808	6.17%
com.ibm.ws.util.ThreadPool\$Worker @ 0x700000003321ce58	216	75,118,760	4.75%
com.ibm.ws.util.ThreadPool\$Worker @ 0x7000000033225ab0	216	74,882,552	4.73%
com.ibm.wps.caf.portlet.StandardPortlet @ 0x700000004f317428	80	10,606,056	0.67%
com.ibm.wps.caf.frame.LegacyEditorsFrame @ 0x700000004f31bdcd	80	10,605,048	0.67%
com.ibm.websphere.personalization.ui.details.views.item.Folder	280	10,601,928	0.67%
java.util.LinkedHashMap @ 0x700000004f3236c0	96	2,112	0.00%
java.util.LinkedHashMap @ 0x700000004f323720	96	424	0.00%
java.util.HashMap @ 0x700000004f323780	80	320	0.00%
java.lang.String @ 0x700000002cd665d18	48	144	0.00%
com.ibm.wps.caf.util.ListenerList @ 0x700000004f323698	40	40	0.00%
Σ Total: 6 entries			
java.util.LinkedHashSet @ 0x700000004f31bdad	32	424	0.00%
java.util.HashMap @ 0x700000004f31be10	80	232	0.00%
java.util.HashMap @ 0x700000004f31be60	80	232	0.00%

Build custom queries with OQL to search for specific object instances



```
SELECT * FROM INSTANCEOF java.util.HashMap m
WHERE inbounds(m).length > 10
AND classof(m).getName() != "java.util.HashMap"
```

Class Name	Shallow Heap	Retained H...
<Regex>	<Numeric>	
com.ibm.icm.util.SynchronizedMap @ 0x700000006744cf8	112	
<class> class com.ibm.icm.util.SynchronizedMap @ 0x11eb7bf30	48	
java.util.HashMap\$Values @ 0x70000000665ed2c0	32	
java.util.HashMap\$Entry[512] @ 0x70000000065ed2e0	4,120	
java.lang.Integer @ 0x700000000674ce30	32	
java.lang.Integer @ 0x700000000674ce50	32	
Σ Total: 5 entries		
com.ibm.ws.webcontainer.util.URIToServletWrapperCache @ 0x70000000026e...	88	
Σ Total: 2 entries		

IBM Monitoring and Diagnostic Tools for Java

- All tools can be downloaded/installed as plugins for IBM Support Assistant Workbench
 - <http://www.ibm.com/software/support/isa/workbench.html>
 - <http://www.ibm.com/developerworks/java/jdk/tools/>
- Newest addition: Interactive Diagnostic Data Explorer (IDDE)
 - Postmortem analysis of system core dumps or javacore files
 - Useful for debugging JVM issues

zEC12 – More hardware for Java

Continued aggressive investment in Java on Z
Significant set of new hardware features
tailored for and co-designed with Java

Hardware Transaction Memory (HTM) *(no z/VM)*

Better concurrency for multi-threaded applications
 e.g. ~2X improvement to j.u.c.ConcurrentLinkedQueue

Run-time Instrumentation (RI)

Innovative new HW facility designed for managed runtimes
 Enables new expanse of JRE optimizations

2GB page frames *(no z/VM or z/Linux)*

Improved performance targeting 64-bit heaps

Pageable 1MB large pages using flash *(no z/VM)*

Better versatility of managing memory

New software hints/directives

Data usage intent improves cache management
 Branch pre-load improves branch prediction

New trap instructions

Reduce over-head of implicit bounds/null checks

New **5.5 GHz 6-Core Processor Chip**
Large caches to optimize data serving
 Second generation **OOO design**



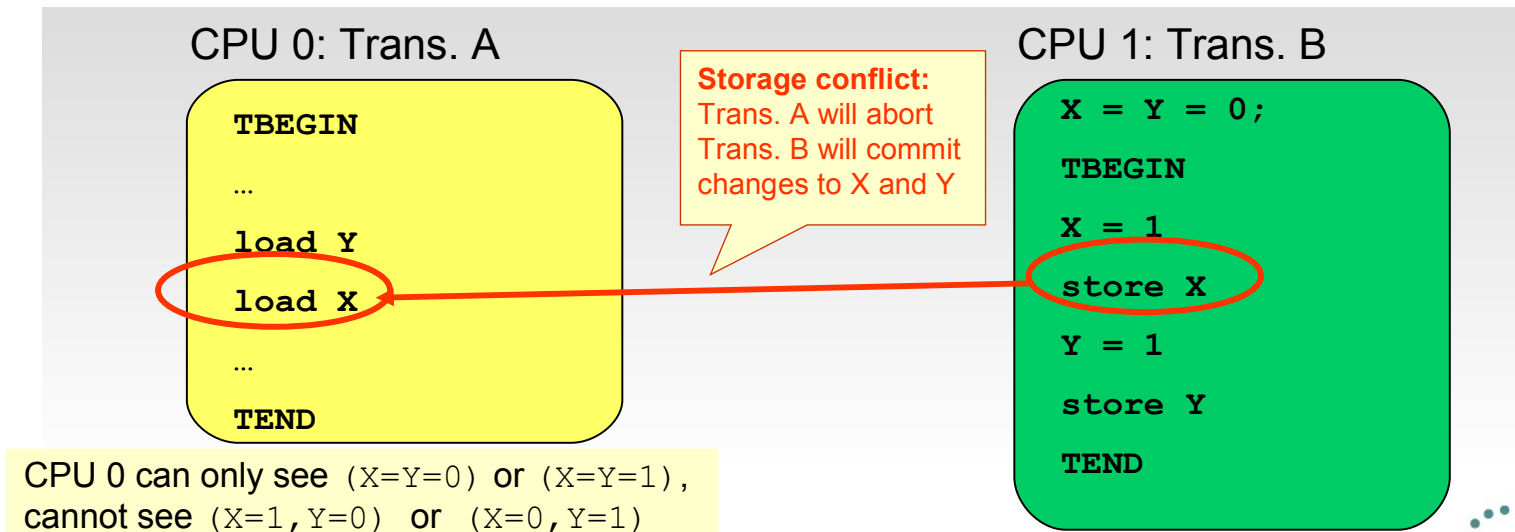
Up-to 60% improvement in throughput amongst Java workloads measured with zEC12 and Java7SR3

Engineered Together—IBM Java and zEC12 Boost Workload Performance
http://www.ibmssystemsmag.com/mainframe/trends/whatsnew/java_compiler/





Hardware Transactional Memory (HTM)

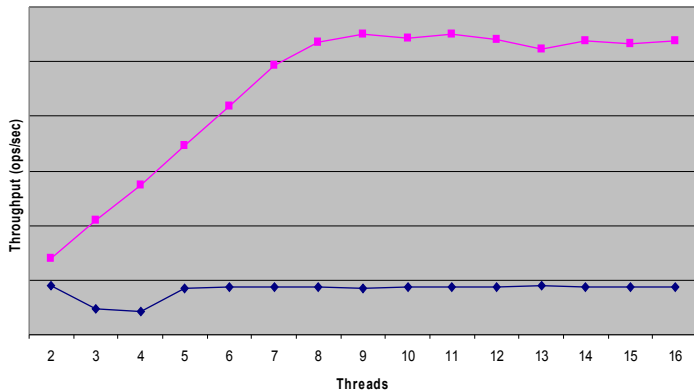
- **Allow lockless interlocked execution of a block of code called a “transaction”**
 - **Transaction:** segment of code that appears to execute “atomically” to other CPUs
 - Other processors in the system will see **either-all-or-none** of the storage updates by the transaction
- **How it works**
 - **TBEGIN** instruction starts speculative execution of transaction
 - Storage conflict detected by hardware and causes roll-back of storage and registers
 - Transaction can be re-tried; or
 - A fall-back code path that performs locking can be used to guarantee forward progress
 - Changes made by transaction become visible to other CPUs after **TEND** instruction



HTM Example: Transactional Lock Elision (TLE)

e·lide  [ih-lahyd]  [Show IPA](#)
verb (used with object), e-lid-ed, e-lid-ing.
 1. to omit (a vowel, consonant, or syllable) in pronunciation.
 2. to suppress; omit; ignore; pass over.
 3. *Law*. to annul or quash.

Transaction Lock Elision on HashTable.get()
Java Prototype

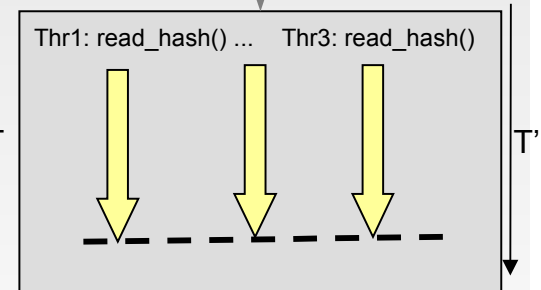
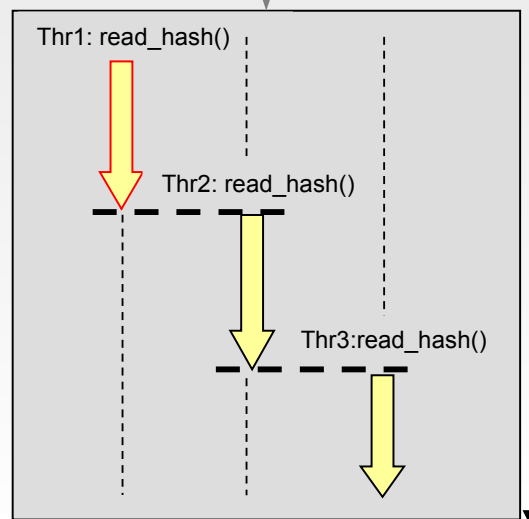


Threads must serialize despite only reading... just in-case a writer updates the hash

```
read_hash(key) {
  Wait_for_lock();
  read(hash, key);
  Release_lock();
}
```

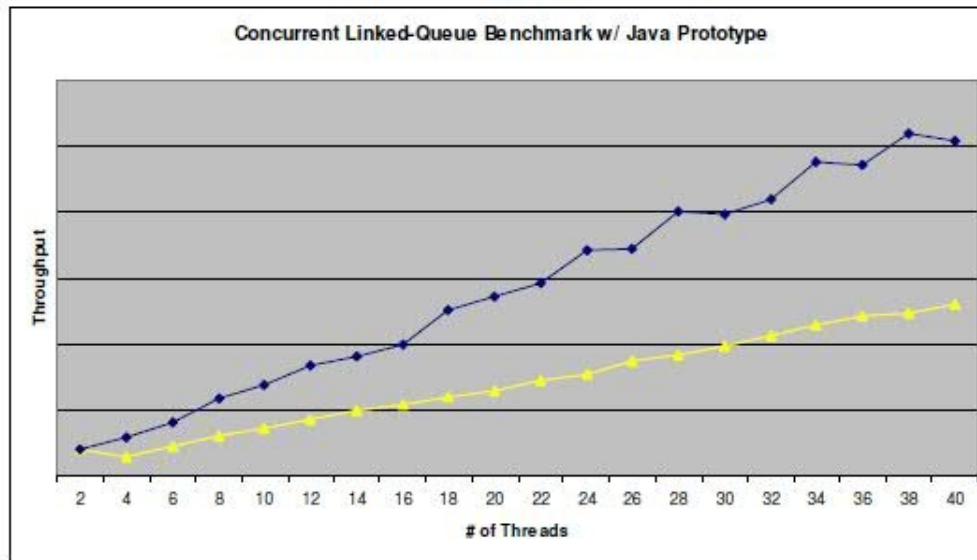
Lock elision allows readers to execute in parallel, and safely back-out should a writer update hash

```
read_hash(key)
  TRANSACTION_BEGIN
  read hash.lock;
  BRNE serialize_on_hash_lock
  read (hash, key);
  TRANSACTION_END
```



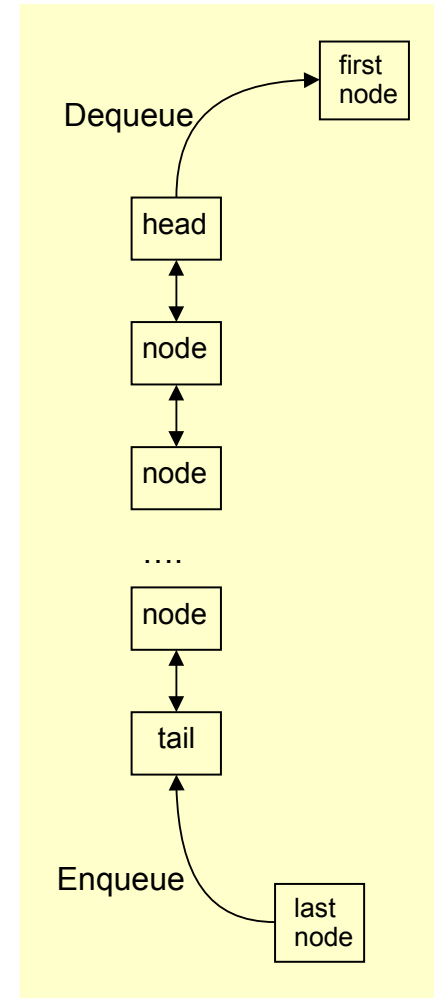
Transactional Execution: ConcurrentLinkedQueue

- **~2x improved scalability of j.u.c.ConcurrentLinkedQueue**
- **Unbound Thread-Safe LinkedQueue**
 - First-in-first-out (FIFO)
 - Insert elements into tail (en-queue)
 - Poll elements from head (de-queue)
 - No explicit locking required
- **Example usage: a multi-threaded work queue**
 - Tasks are inserted into a concurrent linked queue as multiple worker threads poll work from it concurrently



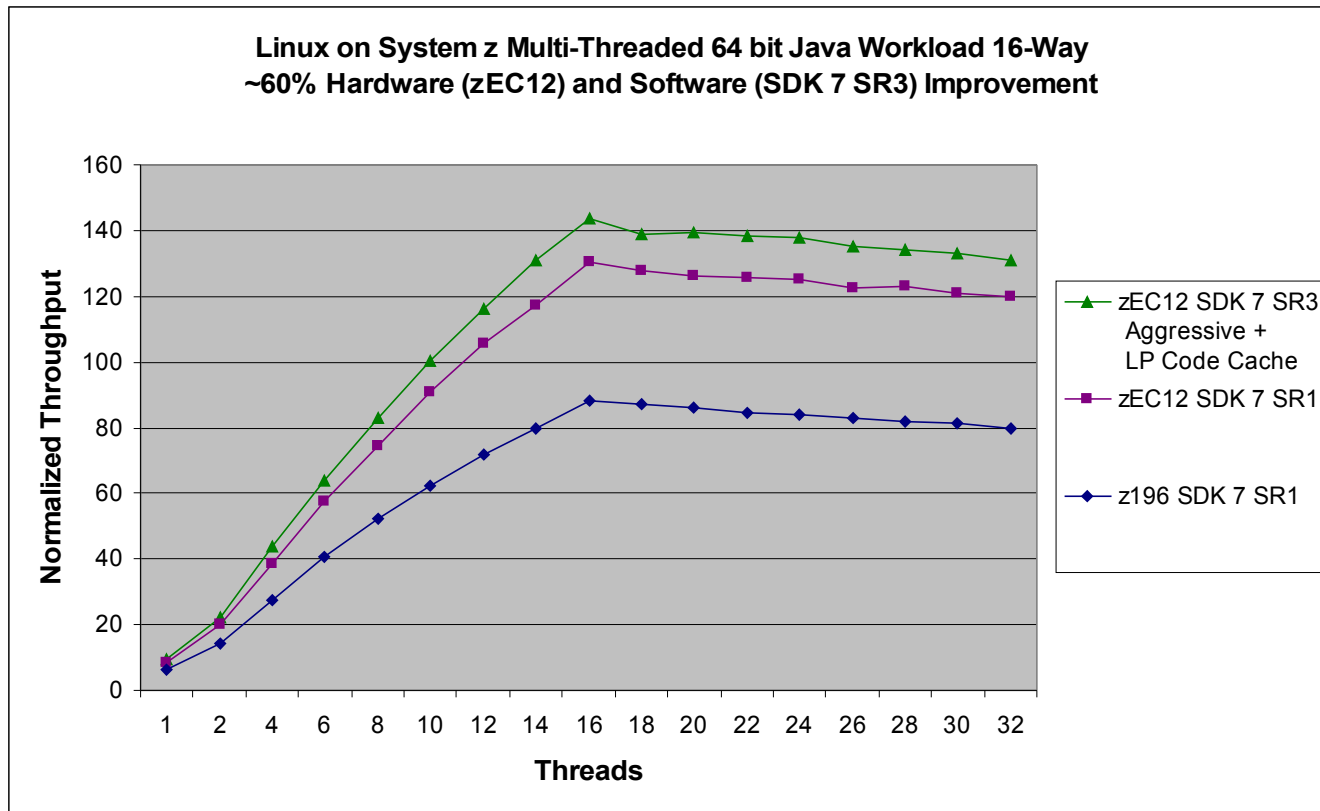
■ New TX-base implementation

■ Traditional CAS-base implementation



Java 7 SR3 on z/Linux on zEC12: Performance

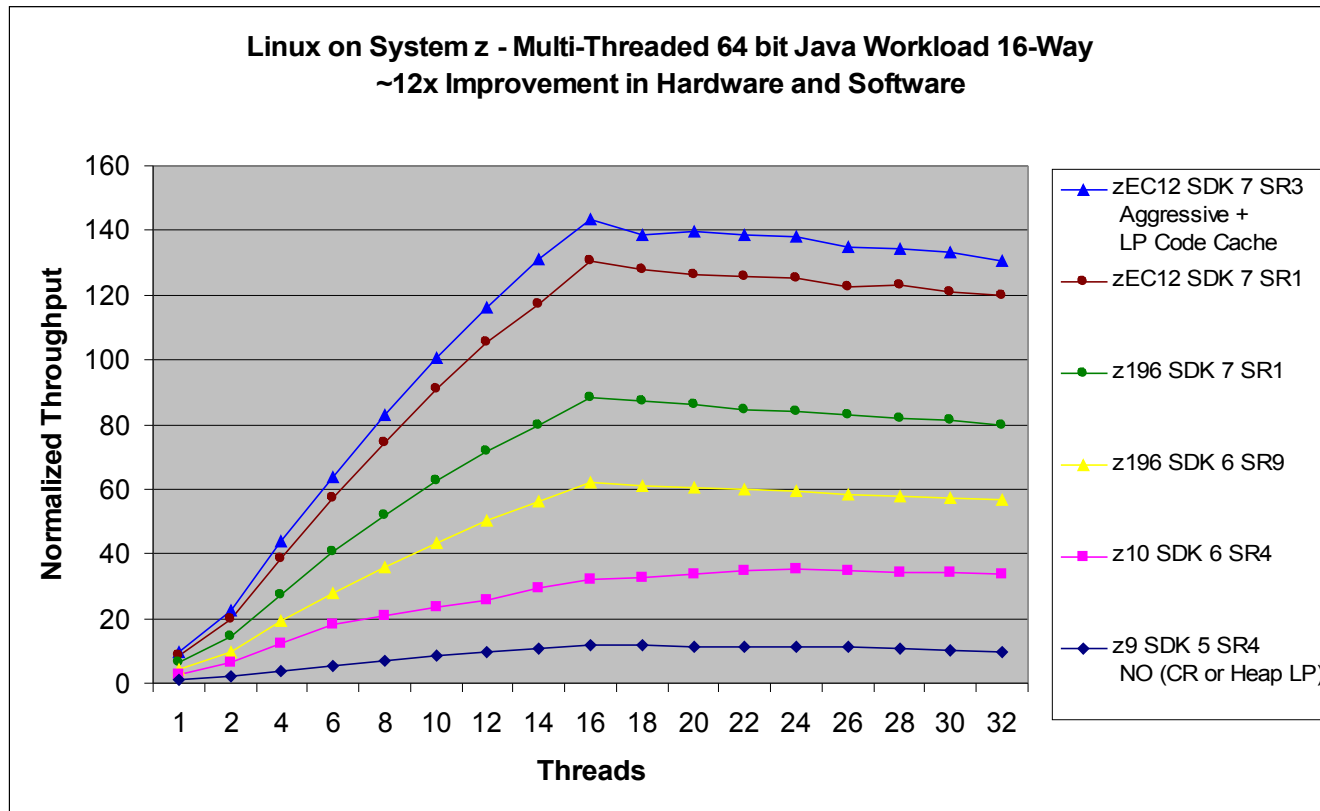
- 64-Bit Java Multi-threaded Benchmark on 16-Way



- Aggregate 60% improvement from zEC12 and Java 7 SR3
 - zEC12 offers a ~45% improvement over z196 running the multi-threaded Java benchmark
 - Java 7 SR3 offers an additional ~10% improvement (with -Xaggressive)

Java 7 SR3 on z/Linux on zEC12: Performance

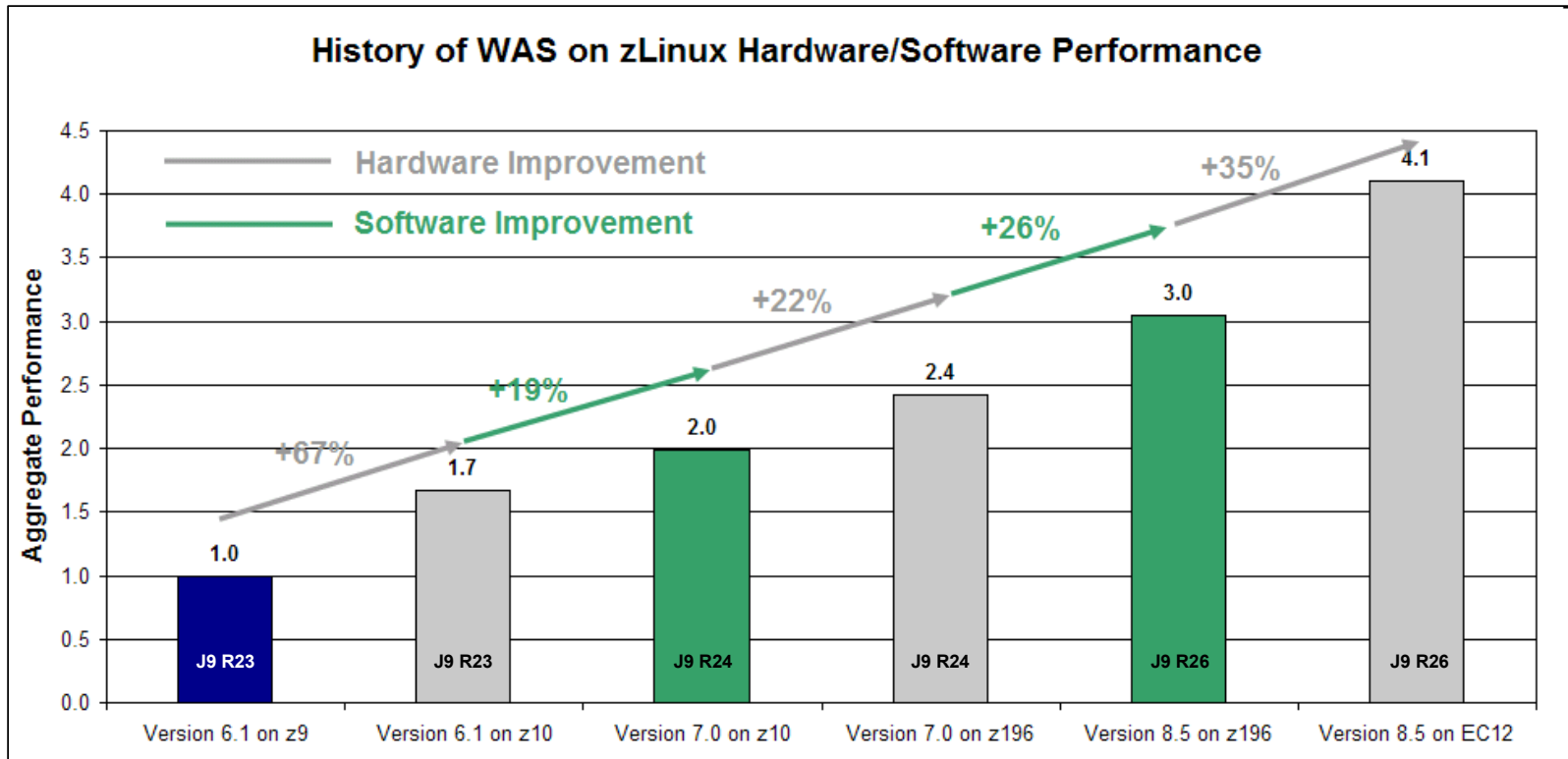
- 64-Bit Java Multi-threaded Benchmark on 16-Way



- ~12x aggregate hardware and software improvement comparing Java 5 SR4 on z9 to Java 7 SR3 on zEC12
 - LP = large pages enabled for Java heap CR = Java compressed references
 - Java 7 SR3 using -Xaggressive and 1MB large pages

WAS on z/Linux: Performance Improvement

- Aggregate improvement in hardware, JDK and WAS



- ~4x aggregate hardware and software improvement comparing WAS 6.1 (Java 5) on z9 to WAS 8.5 (Java 7) on zEC12

Questions

Bryan Chan
bryan.chan@ca.ibm.com



Thank You

© Copyright IBM Corporation 2013. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. IBM, the IBM logo, Rational, the Rational logo, Telelogic, the Telelogic logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.

Summary of Links

- Documentation
 - <http://www.ibm.com/developerworks/java/jdk/docs.html>
- z/OS 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/>