#SHAREorg

SHARE
Technology · Connections · Results

Running Java on Linux on System z

Kishor Patil (patil@ca.ibm.com)
IBM

Wednesday, March 12, 2014
Session 14557
1:30 PM – 2:30 PM

SHARE in Anaheim

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 System z
  - History, overview and roadmap
  - Under the hood: J9 Virtual Machine and IBM Testarossa JIT
- IBM System zEC12 features exploited in IBM Java 7
- New features in IBM Java 7 Release 1
- Preview: Node.js™ support, Multi-tenancy, Java 8 Lambdas
- Garbage collection policies and tuning
- IBM Monitoring and Diagnostic Tools for Java

14709: Need a Support Assistant? Check Out IBM's! (ISA)

Thursday, March 13, 2014: 8:00 AM-9:00 AM
Grand Ballroom Salon A (Anaheim Marriott Hotel)
Speaker: Michael Stephen(IBM Corporation)

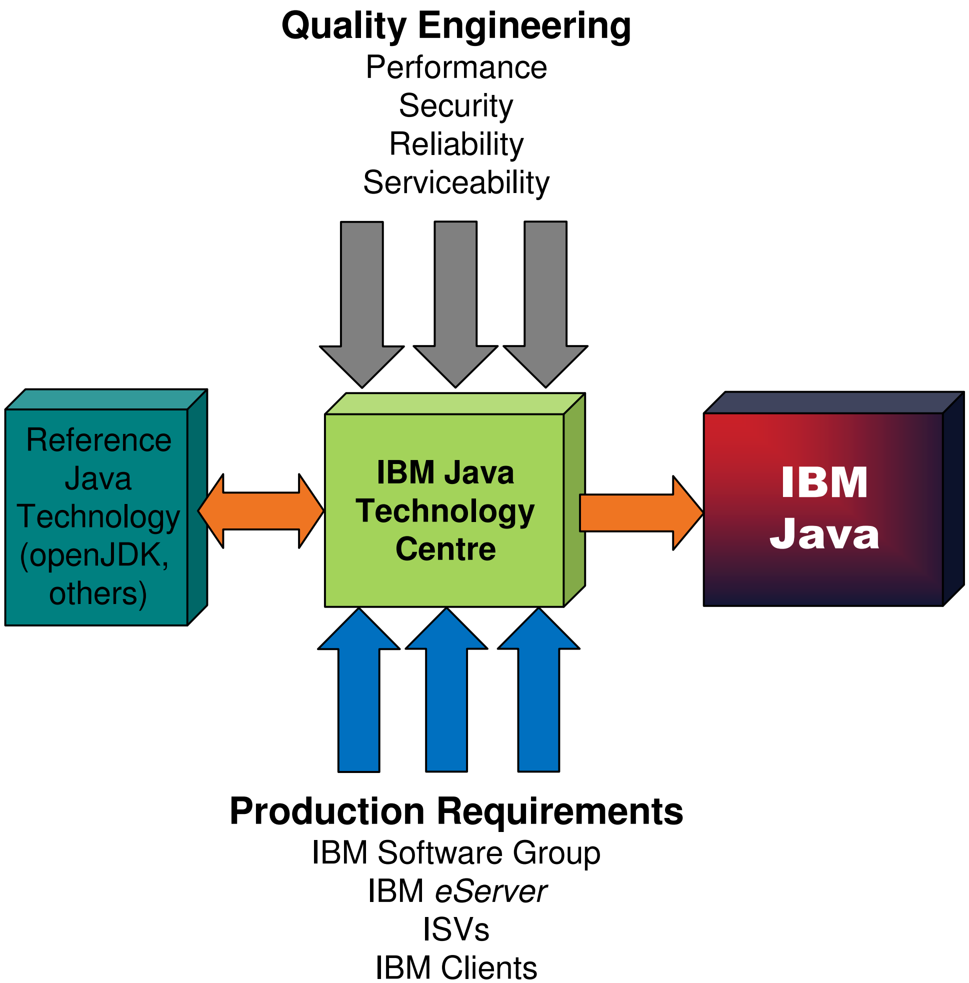
14955: IDDE 1.0 Features and Futures

Thursday, March 13, 2014: 9:30 AM-10:30 AM
Grand Ballroom Salon B (Anaheim Marriott Hotel)
Speaker: Kenneth Irwin(IBM Corporation)

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**
  - A single JVM supports multiple configurations (ME/SE/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*

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)
  - IBM runtimes support and exploit IBM System z and System p platforms

AIX	Linux			Windows	z/OS
PPC-32 PPC-64	x86-32 x86-64	PPC-32 PPC-64	zArch-31 zArch-64	x86-32 x86-64	zArch-31 zArch-64

Porting Java applications to z

- Experience shows there are *subtle differences* between the different JVM™s
  - ***Very important key point***: the IBM® Java® SDK is not a "special" version of Java, it is 100% pure Java, as it passes all compatibility tests from Oracle™

- Differences fall into *2 categories*:
  - ***Infrastructure related*** differences (mostly Java command line parameter differences, for example: garbage collection settings)
  - ***Coding related*** differences (for example: Java class library implementation differences)

Source: http://www.smscs.com

Porting Java applications to z

- *Best practice / strong recommendation*: try to evaluate the to-be-ported application with the IBM Java SDK on any other platform (for example Intel® x86), before going for System z®
  - Most of the porting related issues are related to the mentioned subtle differences in the various JVMs and not System z
  - Following this best practice, the problems can be addressed where they belong to (which is either the application or the IBM Java SDK, but not System z)

- Elements / patterns that are *known to cause trouble*:
  - Heavy usage of platform native libraries / *Java Native Interface* (JNI)
  - Hard-coded path names (happens mostly with Java applications that were developed on / developed for Microsoft® Windows®)
  - Using vendor-specific APIs (for example Java packages starting with `com.sun`)

- Additional problem (project management related): running a *large scale stress test* for the first time as part of the porting
  - Issues in the application that are ***not related to the actual porting*** will surface

Evolving Java on Z

Enable integration of Java-based applications with core z/OS backend database environment for high performance, reliability, availability, security, and lower total cost of ownership

Portable and consumable

- First-class IBM Java SDK for z/OS and z/Linux
- Providing seamless portability across platforms

Pervasive and integrated across the z eco-system

- Java business logic runs with all z middleware (IMS, CICS, WAS etc)
- Inter-operability with legacy batch and OLTP assets

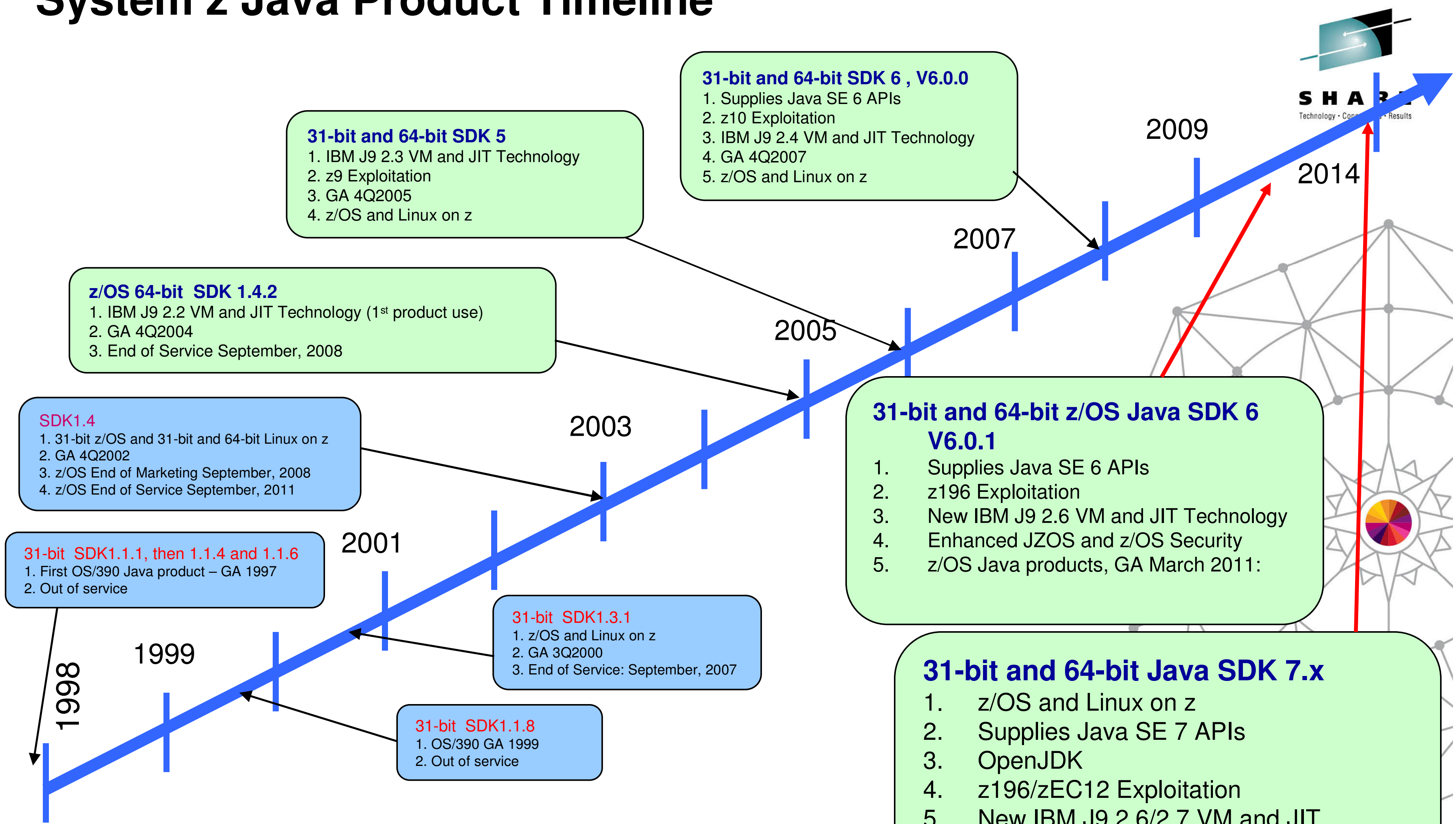
Deep System Z exploitation

- SDK extensions enabled z QoS for full integration with z/OS
- zAAP/zIIP specialty engines provide low-cost Java capacity

Performance

- A decade of hardware/software innovation and optimization
- Industry leading performance with IBM J9 Virtual Machine
- Enabling tight data locality for high-performance and simplified systems

System z Java Product Timeline



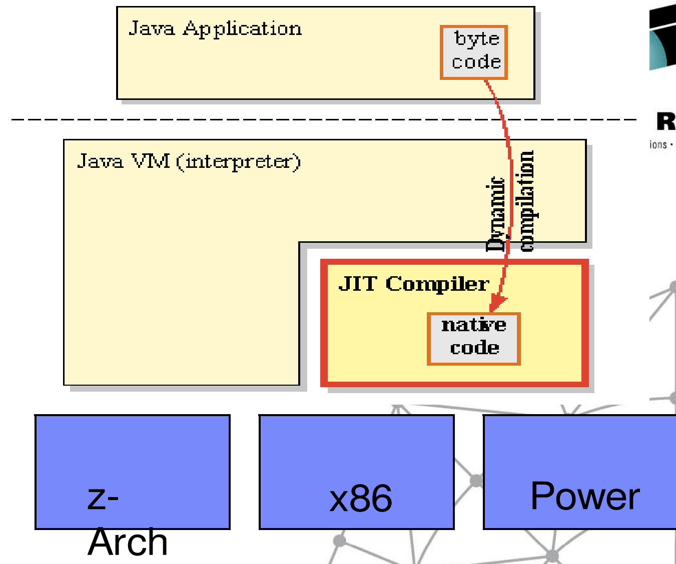
IBM continues to invest aggressively in Java for System z, demonstrating a rich history of innovation and performance improvements.

Testimonials: http://www-01.ibm.com/software/os/systemz/testimonials/

IBM Java Runtime Environment

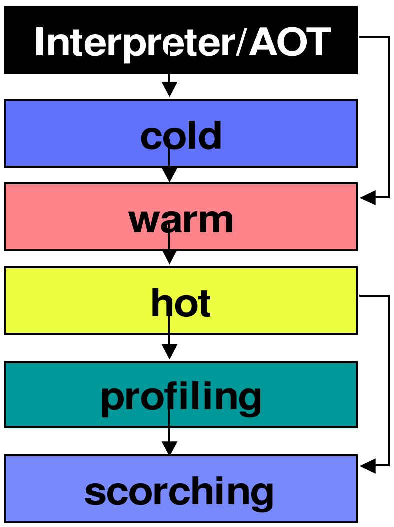
- IBM Java Runtimes since Java 5 are built with **IBM J9 Virtual Machine** and **IBM Testarossa JIT Compiler** technology
  - Independent clean-room JVM runtime & JIT compiler
- Combines best-of breed from embedded, development and server environments… from a cell-phone to a mainframe!
  - Lightweight 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 exploitation
  - Cost-effective for z - zAAP Ready!
- Millions of instances of J9/TR compiler

IBM Testarossa JIT Compiler – Introduction



- Compile byte-code down to native assembly to remove the overhead of interpretation
- Significantly more efficient use of computational resource
  - ~10-100x faster than interpretation
- Discovers and exploits the program's runtime environment to generate optimal assembly
- Compilation cost is included in application runtime, hence uses runtime profiling to direct compilation decisions
  - Choose what to compile
  - How much effort to invest in compilation

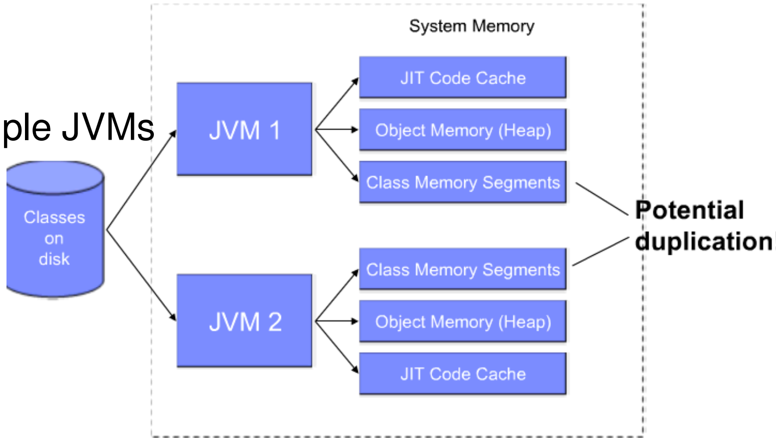
JIT Compilation Strategy:



- **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 for register allocation, branch straightening, etc
  - Values/types are profiled
  - hot paths are specialized/versioned
  - Virtual calls are profiled, hot targets are in-lined

Shared Classes & Ahead-Of-Time (AOT) Compilation

- **Shared Classes**
  - Store classes into a cache that can be 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)
  - Class compression (64-bit class compression)
  - Persistent cache (between reboots)



- **AOT Compilation**
  - Compiled code generated “ahead-of-time” to be used by a subsequent execution
    - Performance of AOT code is poor
      - *Cannot be specialized due multi-instance use and dynamic class loading*
      - *Dynamic class loading imposes overhead of assumption management*
    - Rely on recompilation to make code that matters better
  - Persisted into the same shared cache
  - Startup time improvements
  - CPU utilization reduction

Java Road Map

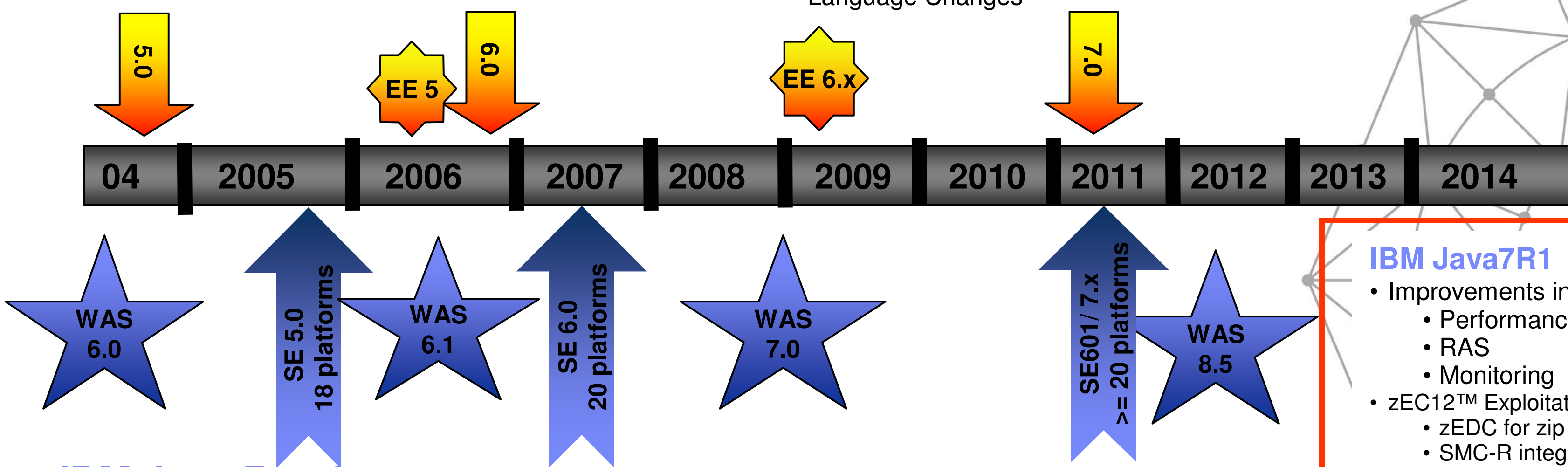
Language Updates

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/Java7.0 (J9 R26)
- Improvements in
  - Performance
  - GC Technology
- z196™ Exploitation
  - OOO Pipeline
  - 70+ New Instructions
- JZOS/Security Enhancements

IBM Java7R1
- Improvements in
  - Performance
  - RAS
  - Monitoring
- zEC12™ Exploitation
  - zEDC for zip acceleration
  - SMC-R integration
  - Transactional Execution
  - Runtime instrumentation
- Hints/traps
- Data Access Accelerator

IBM Java7.0SR3
- Improvements in
  - Performance
- zEC12™ Exploitation
  - Transactional Execution
  - Flash 1Meg pageable LPs
  - 2G large pages
  - Hints/traps

**Timelines and deliveries are subject to change.

zEC12 – More Hardware for Java

Continued aggressive investment in Java on Z

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

Hardware Transaction Memory (HTM) *

Better concurrency for multi-threaded applications
eg. ~2X improvement to juc.ConcurrentLinkedQueue

Run-time Instrumentation (RI)*

Innovation new h/w facility designed for managed runtimes
Enables new expanse of JRE optimizations

2GB page frames **

Improved performance targeting 64-bit heaps

Pageable 1M large pages with Flash Express**

Better versatility of managing memory

Shared-Memory-Communication **

RDMA over Converged Ethernet

zEnterprise Data Compression accelerator **

gzip accelerator

New software hints/directives/traps

Branch preload improves branch prediction
Reduce overhead of implicit bounds/null checks

New 5.5 GHz 6-Core Processor Chip

Large caches to optimize data serving

Second generation OOO design

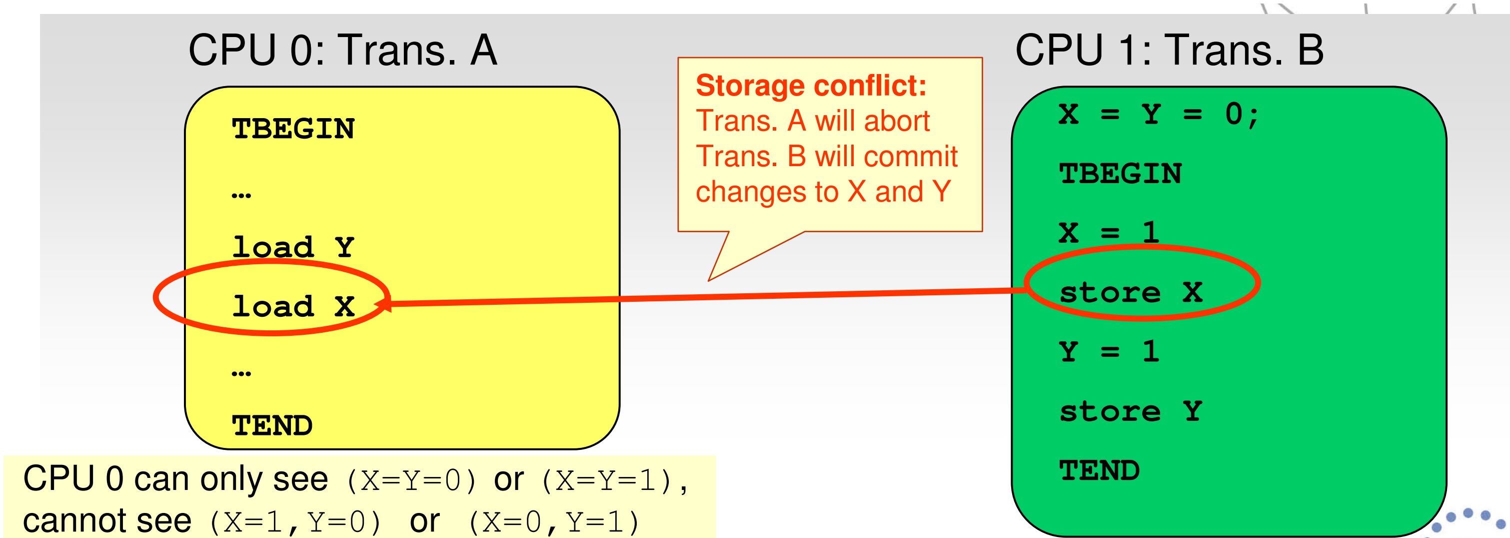
*Up-to **60%** improvement in throughput amongst Java workloads measured with zEC12 and Java7SR3*

* Not supported under zVM, native LPAR needs SLES11 SP3/RHEL6.3
** Linux lp is already pagable, but no flash support
** currenlt only supported on zOS

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

Hardware Transactional Memory (HTM)

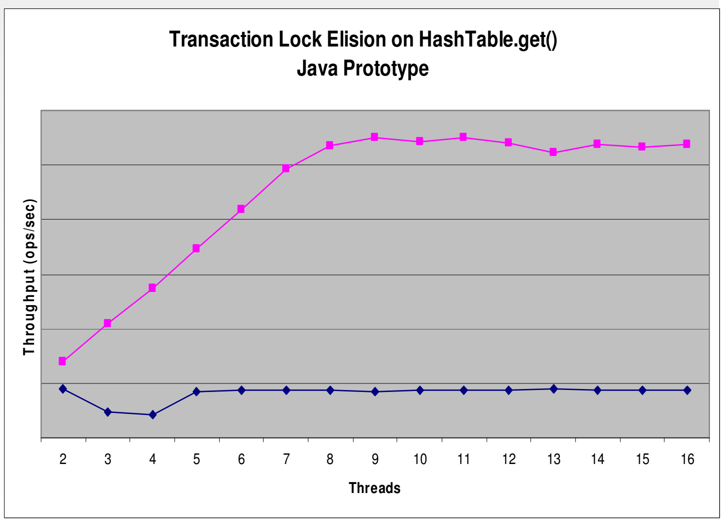
- **Allow lockless interlocked execution of a block of code called a "transaction"**
  - **Transaction**: segment of code that appears to execute "atomically" to other CPUs
    - Other processors in the system will see **either-all-or-none** of the storage updates by the transaction
- **How it works**
  - **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)

Java on zEC12

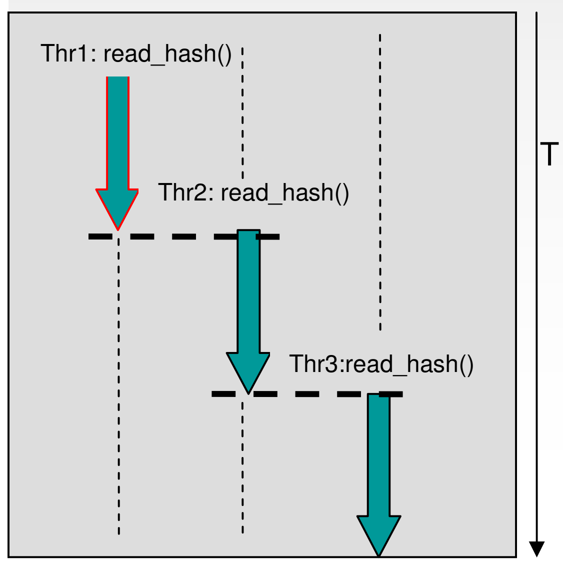
e·lide [ih-lahyd] [?] Show IPA
verb (used with object), **e·lid·ed, e·lid·ing.**
1. to omit (a vowel, consonant, or syllable) in pronunciation.
2. to suppress; omit; ignore; pass over.
3. *Law* . to annul or quash.



(Controlled measurement environment, results may vary)

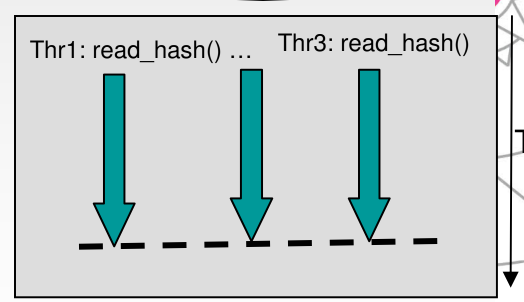
Threads must serialize despite only reading… just in-case a writer updates the hash

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



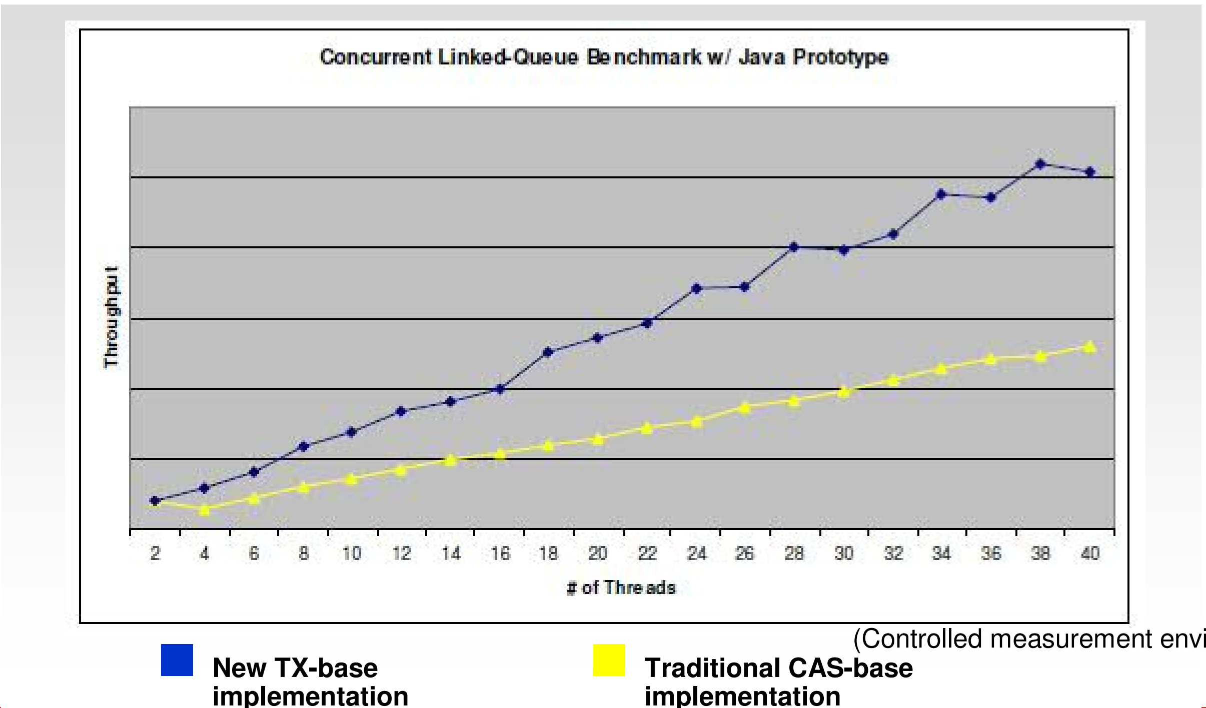
Lock elision allows readers to execute in parallel, and safely back-out should a writer update hash

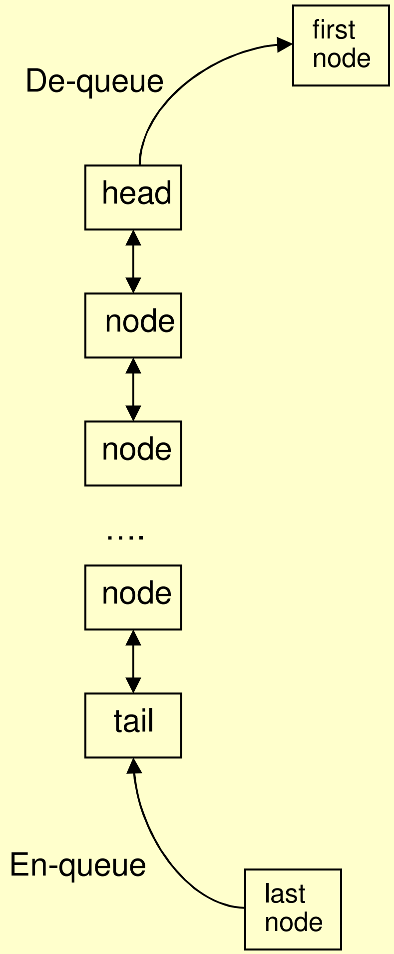
```
read_hash(key)
  TRANSACTION_BEGIN
  read hash.lock;
  BRNE serialize_on_hash_lock
  read (hash, key);
  TRANSACTION_END
```



Transactional Execution: Concurrent Linked Queue

- ***~2x improved scalability of juc.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

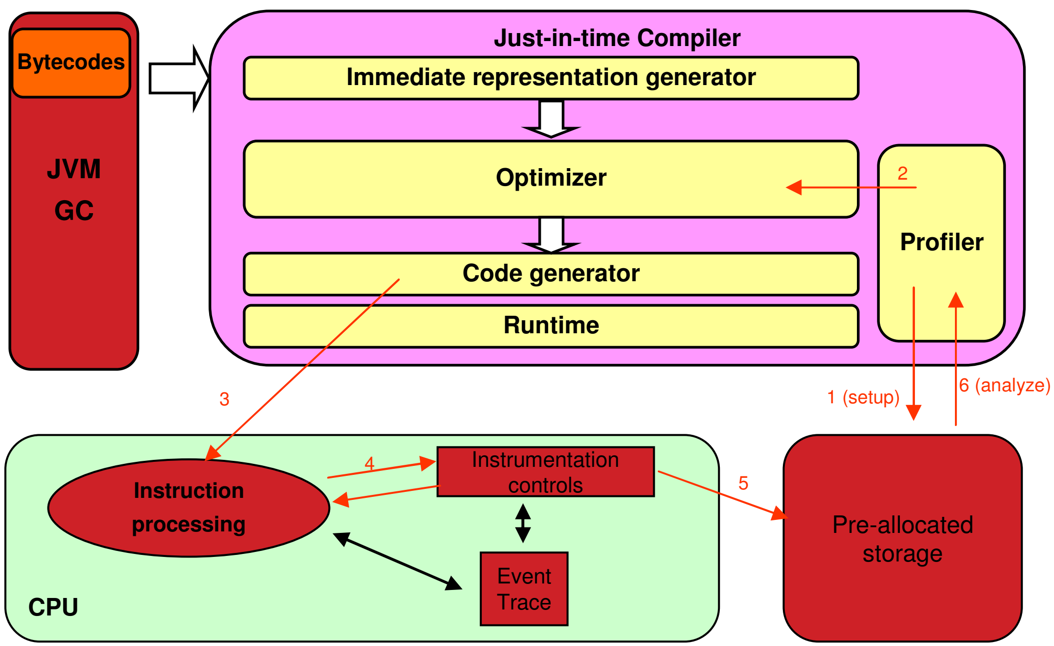


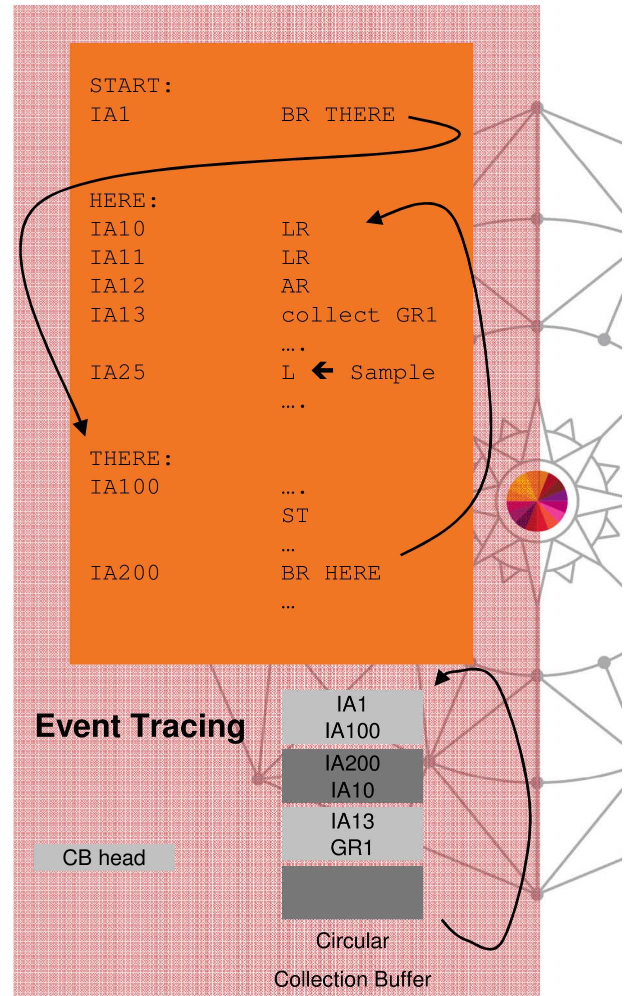


(Controlled measurement environment, results may vary)

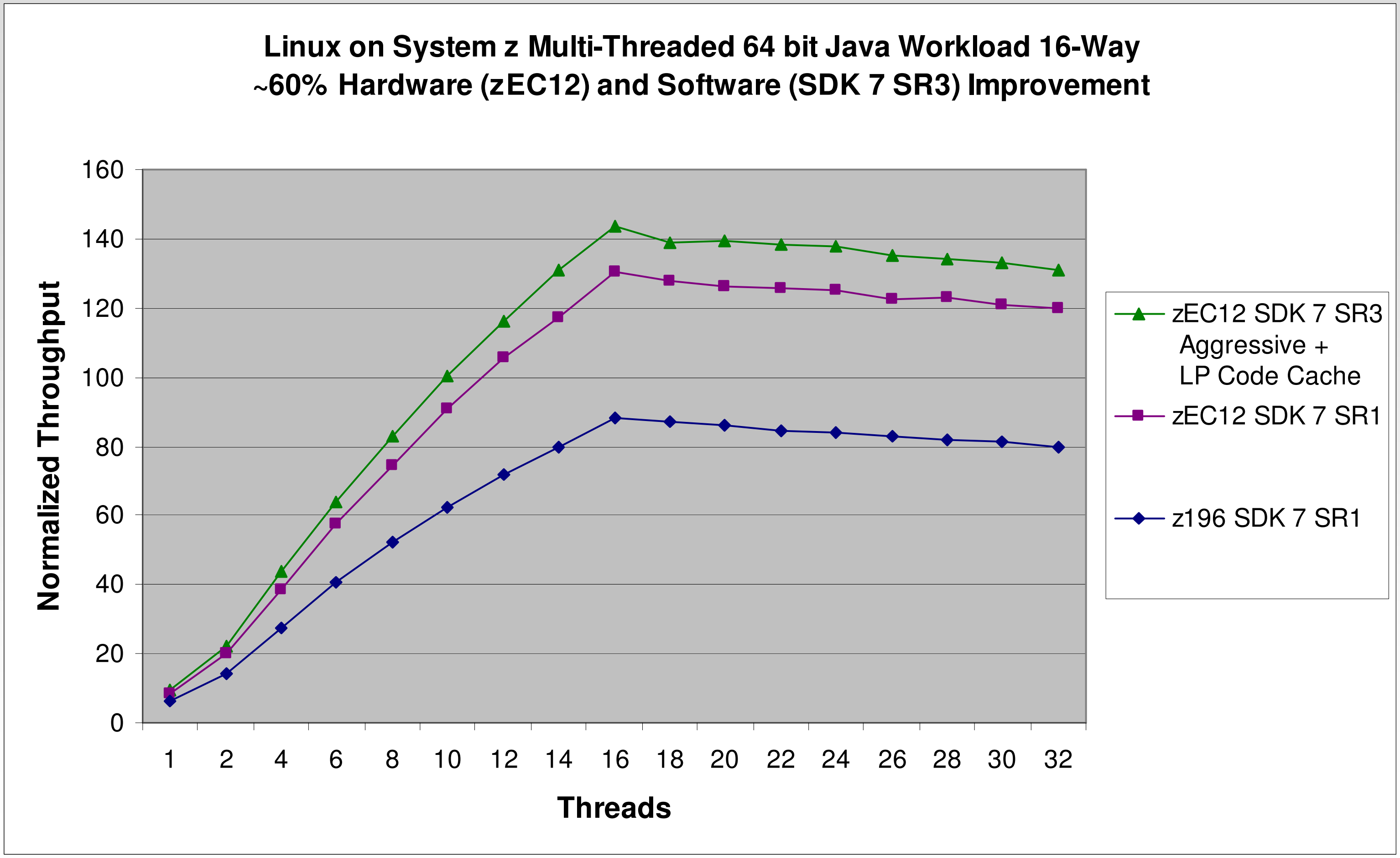
Runtime Instrumentation

- ***Low overhead profiling with hardware support***
- Instruction samples by time, count or explicit marking
- ***Sample reports include hard-to-get information:***
  - Event traces, e.g. taken branch trace
  - "costly" events of interest, e.g. cache miss information
  - GR value profiling
- ***Enables better "self-tuning" opportunities***





Linux on System z and IBM Java 7 on zEC12:

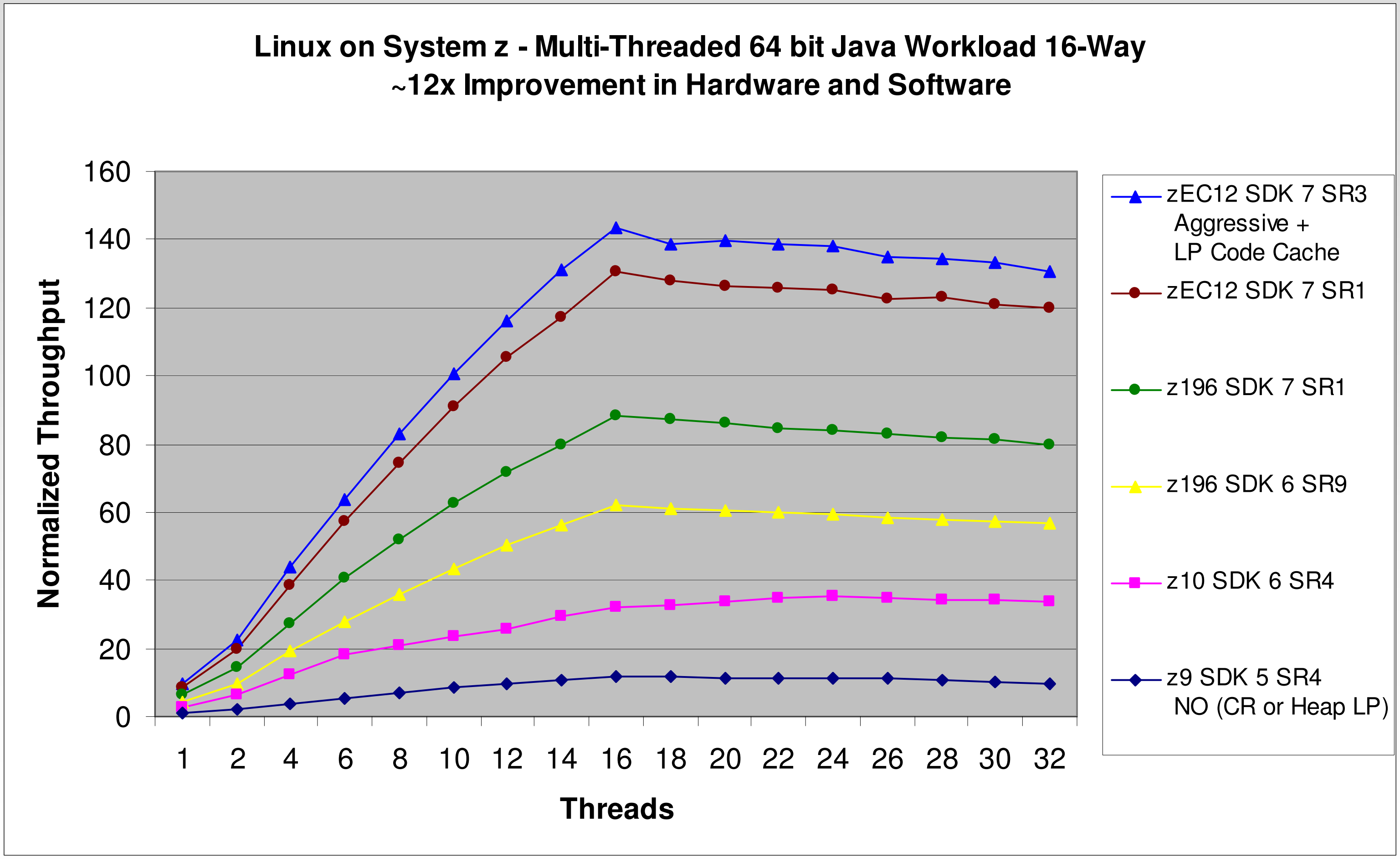


Aggregate 60% improvement from zEC12 and IBM Java 7 (Controlled measurement environment, results may vary)

- **zEC12 offers a ~45% improvement over z196 running the Java Multi-Threaded Benchmark**
- **IBM Java7 offers an additional ~10% improvement** (SR3 and -Xaggressive)

Linux on System z and Java7SR3 on zEC12:

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



~12x aggregate hardware and software improvement comparing IBM Java5 on z9 to IBM Java7 on zEC12

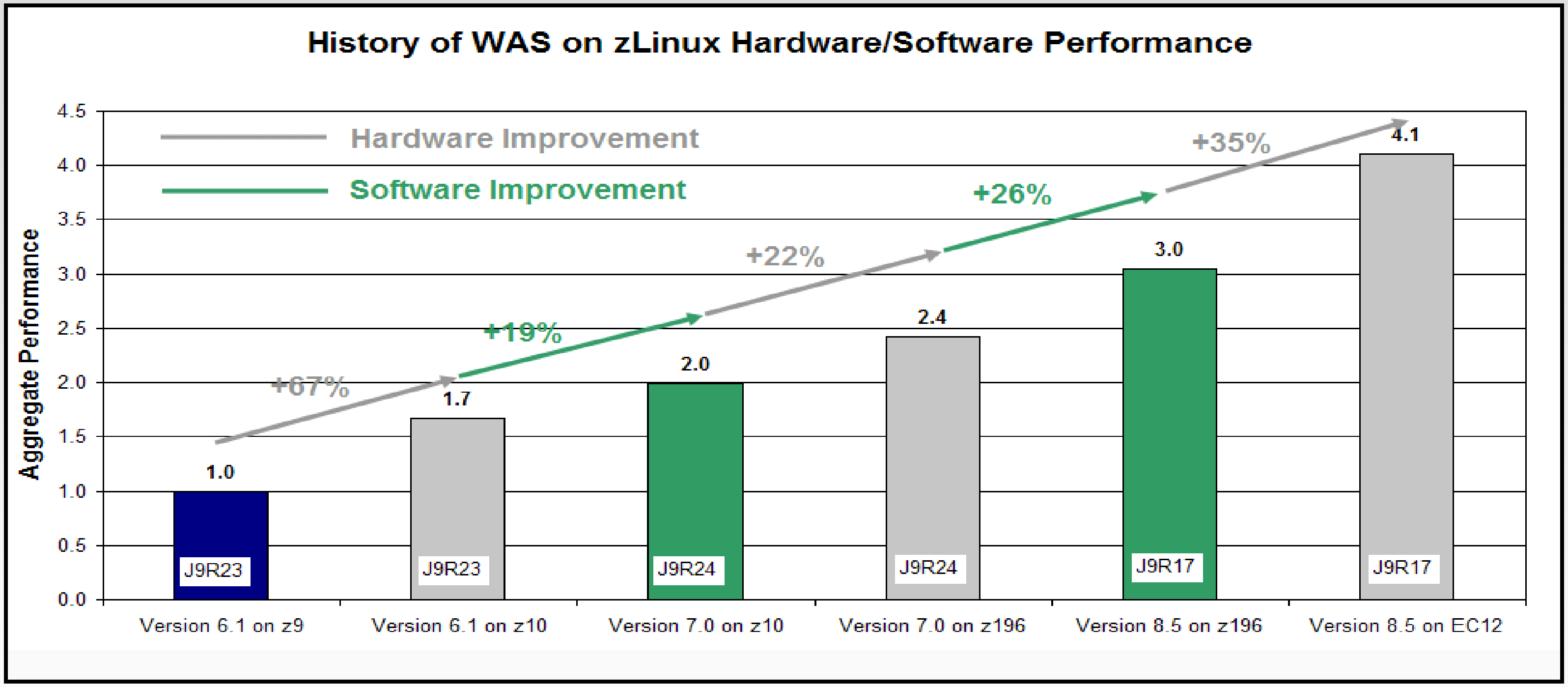
LP=Large Pages for Java heap CR= Java compressed references

IBM Java7SR3 using -Xaggressive + 1Meg large pages

(Controlled measurement environment, results may vary)

WAS on zLinux –

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



~4x aggregate hardware and software improvement comparing WAS 6.1 with IBM Java5 on z9 to WAS 8.5 with IBM Java7 on zEC12

(Controlled measurement environment, results may vary)

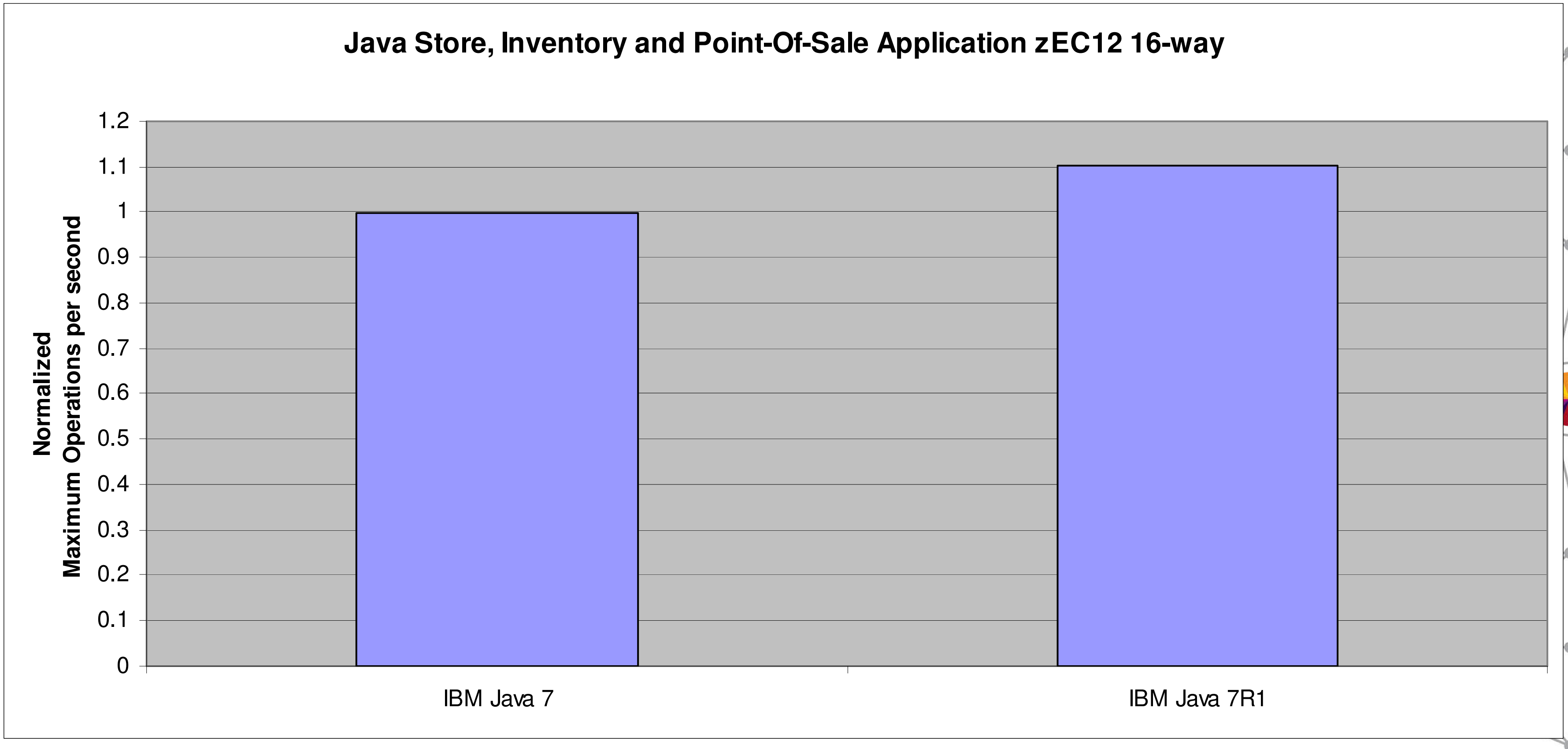
IBM SDK, Java Technology Edition,Version 7 Release 1

http://www.ibm.com/developerworks/java/jdk/linux/download.html

SHARE
Technology • Connections • Results

- **New IBM Java runtime (J9R27) with Java 7 class library**
- **Expand zEC12/zBC12 exploitation**
  - More TX, instruction scheduler, traps, branch preload
  - Runtime instrumentation exploitation
  - zEDC exploitation through java/util/zip
  - Integration of SMC-R
- **Improved native data binding - Data Access Accelerator**
  - Integrated with JZOS native record binding framework
- **Improved general performance/throughput**
  - Up-to 19% improvement to throughput (ODM)
  - Up-to 2.4x savings in CPU-time for record parsing batch applicatic
- **Improved WLM capabilities**
- **Improved SAF and cryptography support**
- **Additional reliability, availability, and serviceability (RAS) enhancements**
- **Enhanced monitoring and diagnostics**

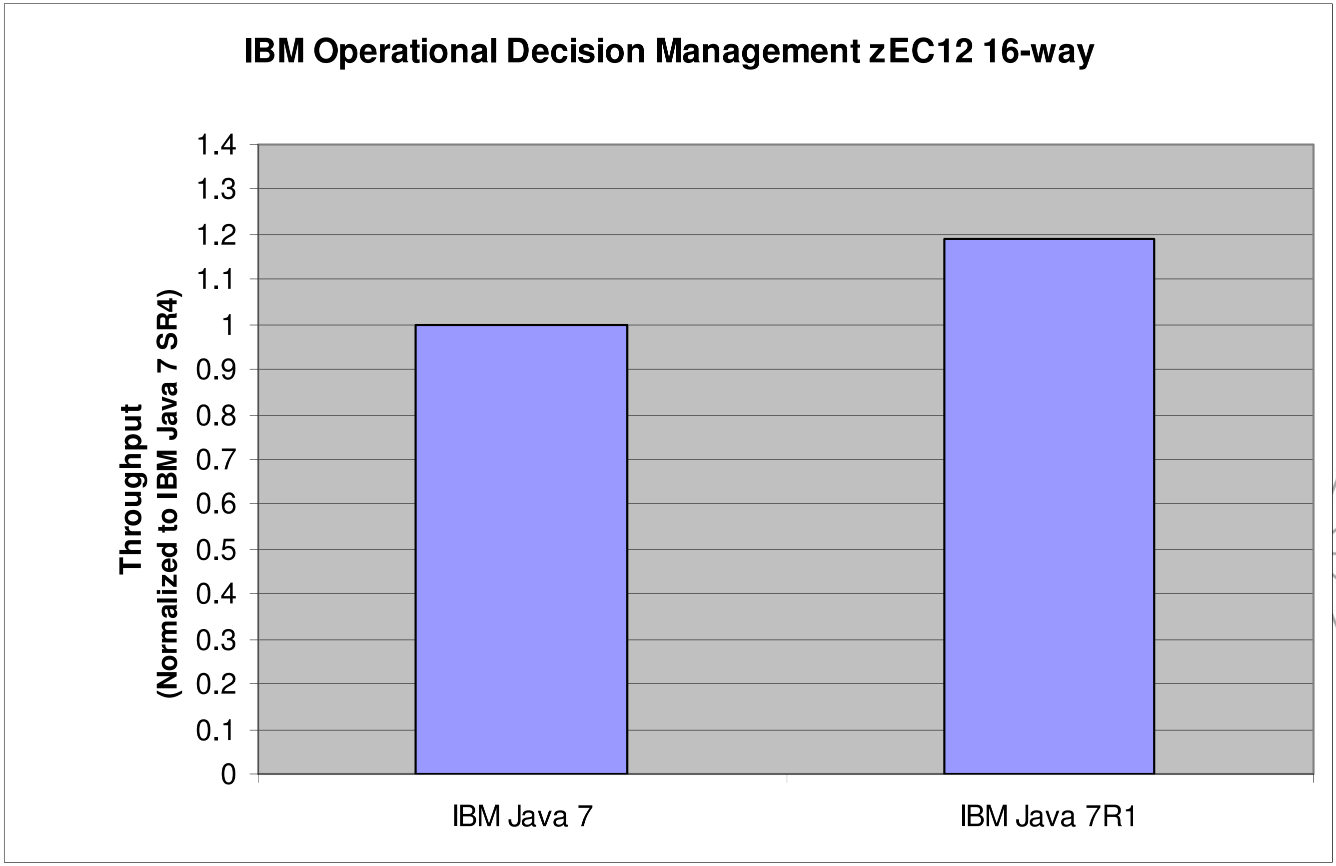
Java-based Store, Inventory and Point-of-Sale App and IBM Java 7R1



- 10% improvement to Java-based Inventory and Point-of-Sale application with IBM Java 7R1 compared to IBM Java 7

(Controlled measurement environment, results may vary)

IBM Operational Decision Manager



- **19% improvement to ODM with IBM Java7R1 compared to IBM Java7 SR4**
  - **19% improvement to ODM with IBM Java7 SR4 compared to IBM Java 7 SR1**
    - **22% improvement to ODM with zEC12 compared to z196**

(Controlled measurement environment, results may vary)

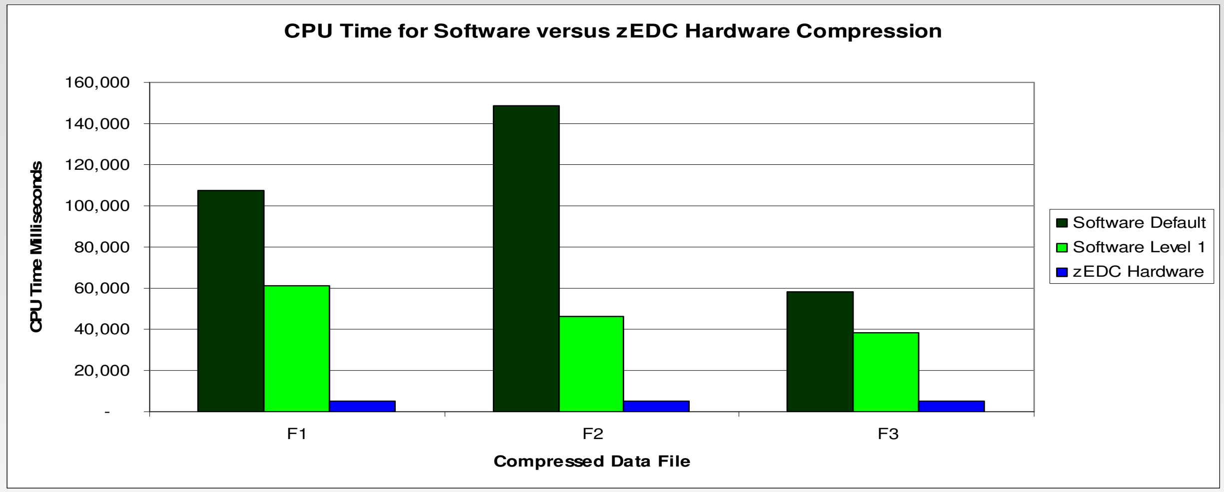
Store your Data - zEnterprise Data Compression and IBM Java 7R1

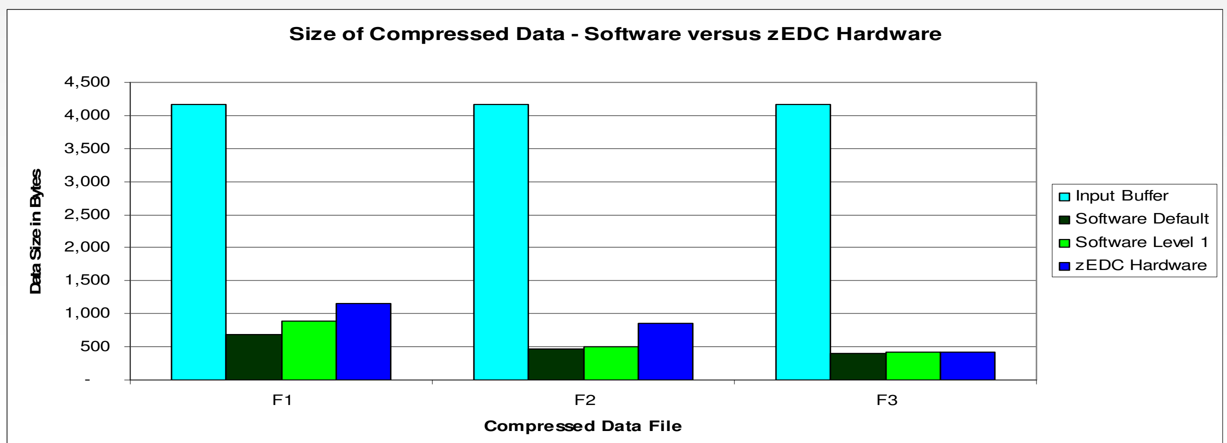
Every day over 2000 petabytes of data are created
Between 2005 to 2020, the digital universe will grow by 300x, going from 130 to 40,000 exa-bytes**
80% of world's data was created in last two years alone.

What is it?

- ✓ *zEDC Express is an IO adapter that does high performance industry standard compression*
- ✓ *Used by z/OS Operating System components, IBM Middleware and ISV products*
- ✓ *Applications can use zEDC via industry standard APIs (zlib and Java)*
- ✓ *Each zEDC Express sharable across 15 LPARs, up to 8 devices per CEC.*
- ✓ *Raw throughput up to **1 GB/s** per zEDC Express Hardware Adapter*

With IBM Java 7R1 : Up-to **12x improvement in CPU time**
Up-to **3x improvement in elapsed time**
Compression ratio of ~4x



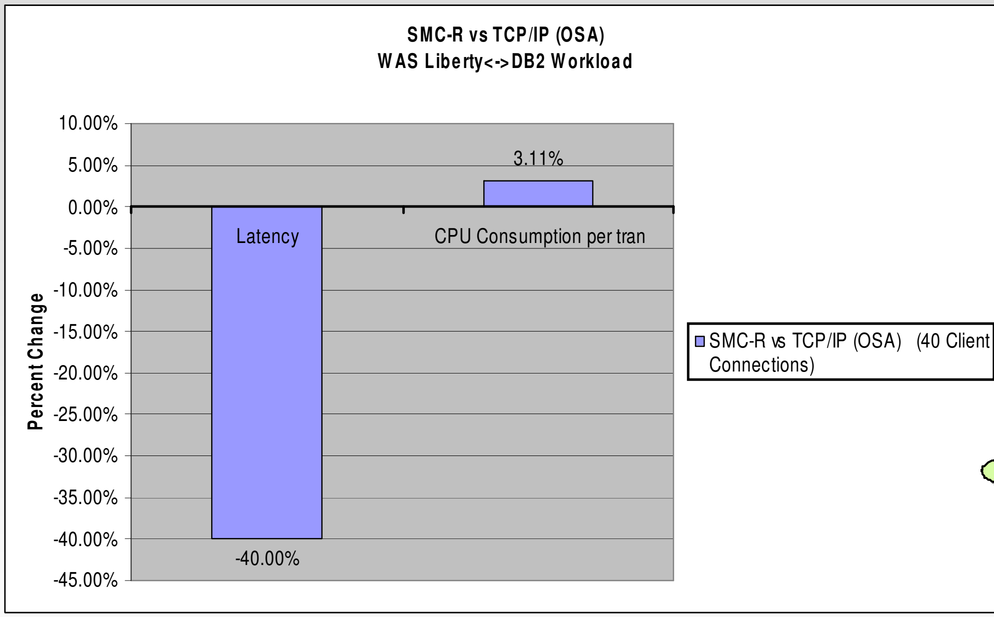


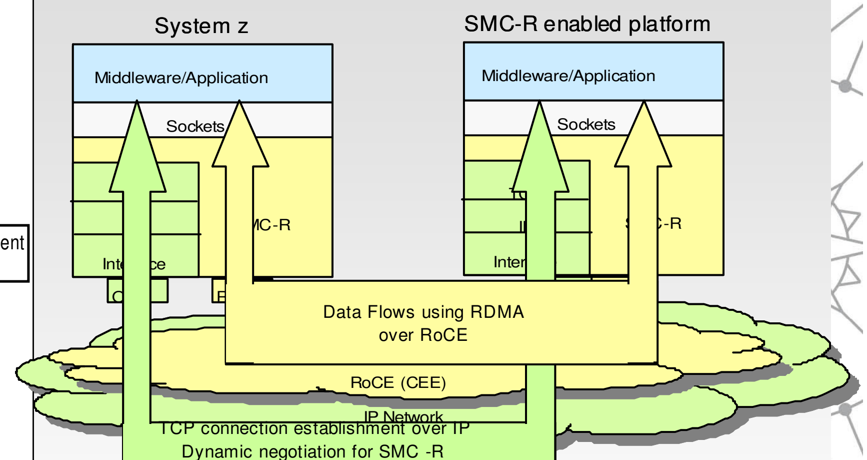
(Controlled measurement environment, results may vary)

** IDC: The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East

Move your Data - Shared Memory Communications (SMC-R):

Exploit RDMA over Converged Ethernet (RoCE) with qualities of service support for dynamic failover to redundant hardware





(Controlled measurement environment, results may vary)

- Transparent exploitation for TCP sockets based applications
- Compatible with existing TCP/IP based load balancing solutions
- Up-to 40% reduction in end-to-end transaction latency
- Slight increase in CPU is due to very small message size in this workload (~100 bytes). Workloads with larger payloads are expected to show a CPU savings

Java7R1: 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

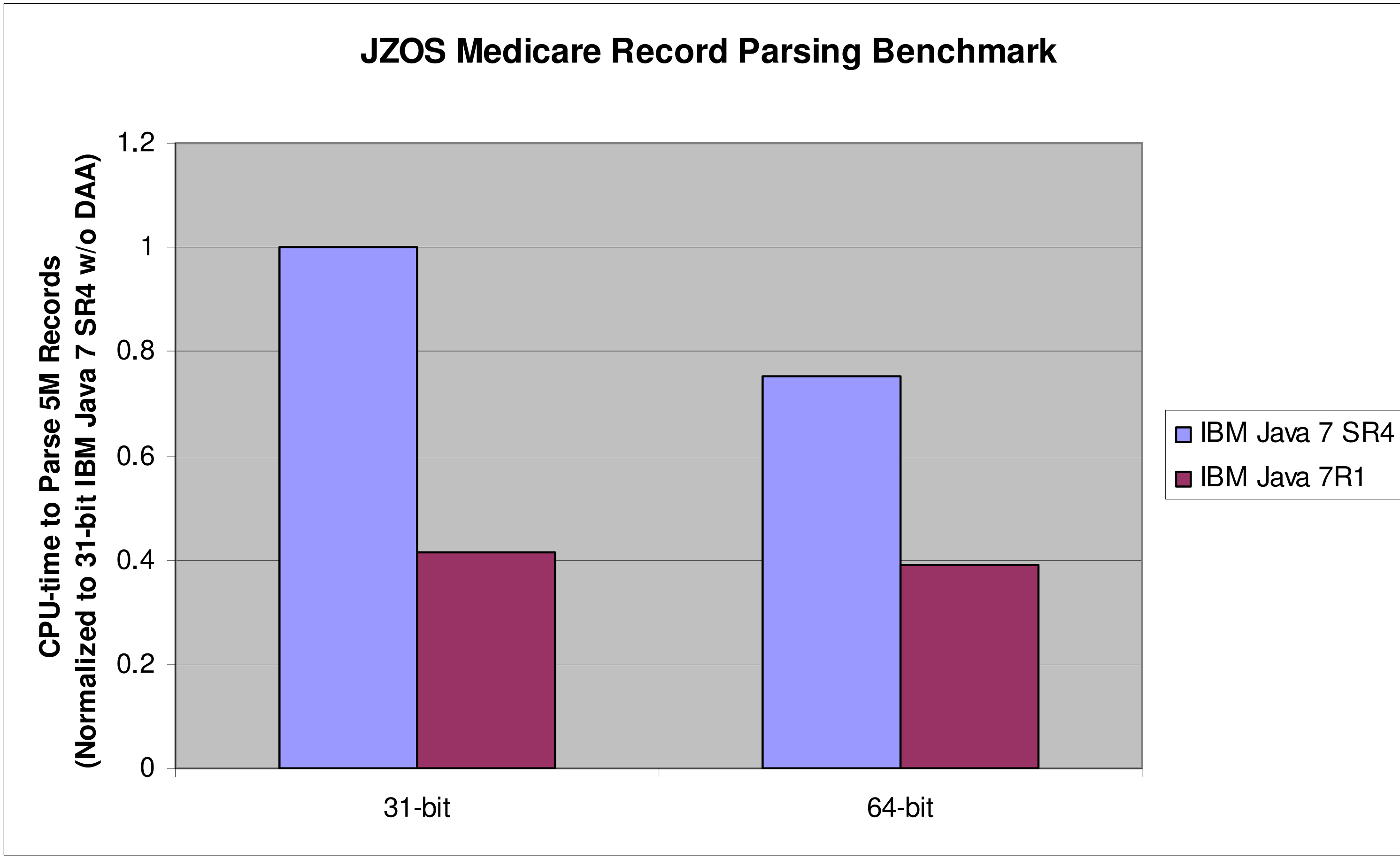
- Arithmetic: +, -, *, /, %
- Logical:>, <, >=, <=, ==, !=
- Validation: verifies if a PD operand is well-formed
- Others: optimized shifts, moves on PD operand

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

Detailed API Specification: https://ibm.biz/BdRvwC

DAA – JZOS Medicare Record Benchmark and IBM Java 7R1



- 31-bit IBM Java 7R1 with DAA versus IBM Java 7 CPU Time improved by by 2.4x
- 64-bit IBM Java 7R1 with DAA versus IBM Java 7 CPU Time improved by by 1.9x

http://www.ibm.com/developerworks/java/zos/javadoc/jzos/index.html?com/ibm/jzos/sample/fields/MedicareRecord.html

(Controlled measurement environment, results may vary)

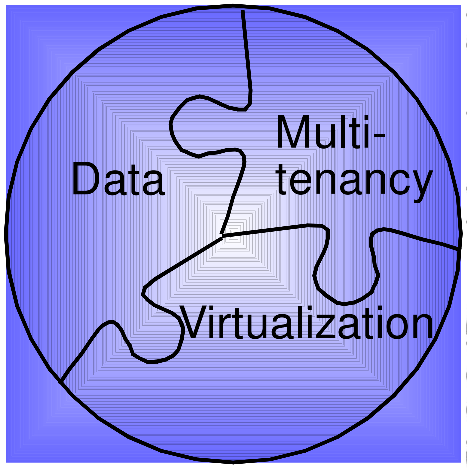
IBM SDK for Node.js™

- **Stand-alone JavaScript® runtime and server-side JavaScript solution for IBM platforms.**
  - Node.js™ (http://nodejs.org) platform built on Google's V8 JavaScript engine (http://code.google.com/p/v8/)
  - **Available:** Binaries for Linux on IBM POWER Systems, and Linux/Windows/Mac OS X on Intel
    - https://www.ibm.com/developerworks/web/nodesdk/
  - Support for other IBM platforms is being developed**.
  - Open source projects with active development in GitHub**
    - *V8 on System z: https://github.com/andrewlow/v8z*
    - *V8 on System p: https://github.com/andrewlow/v8ppc*
    - *Node.js™: https://github.com/andrewlow/node*
    - *Development builds: http://v8ppc.osuosl.org:8080/*
      - *Now includes early AIX builds*
  - Provide feedback via IBM developerWorks community
    - *https://www.ibm.com/developerworks/community/groups/community/node*

**Timelines and deliveries are subject to change.

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** = **m**ulti-**t**enancy)



- **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

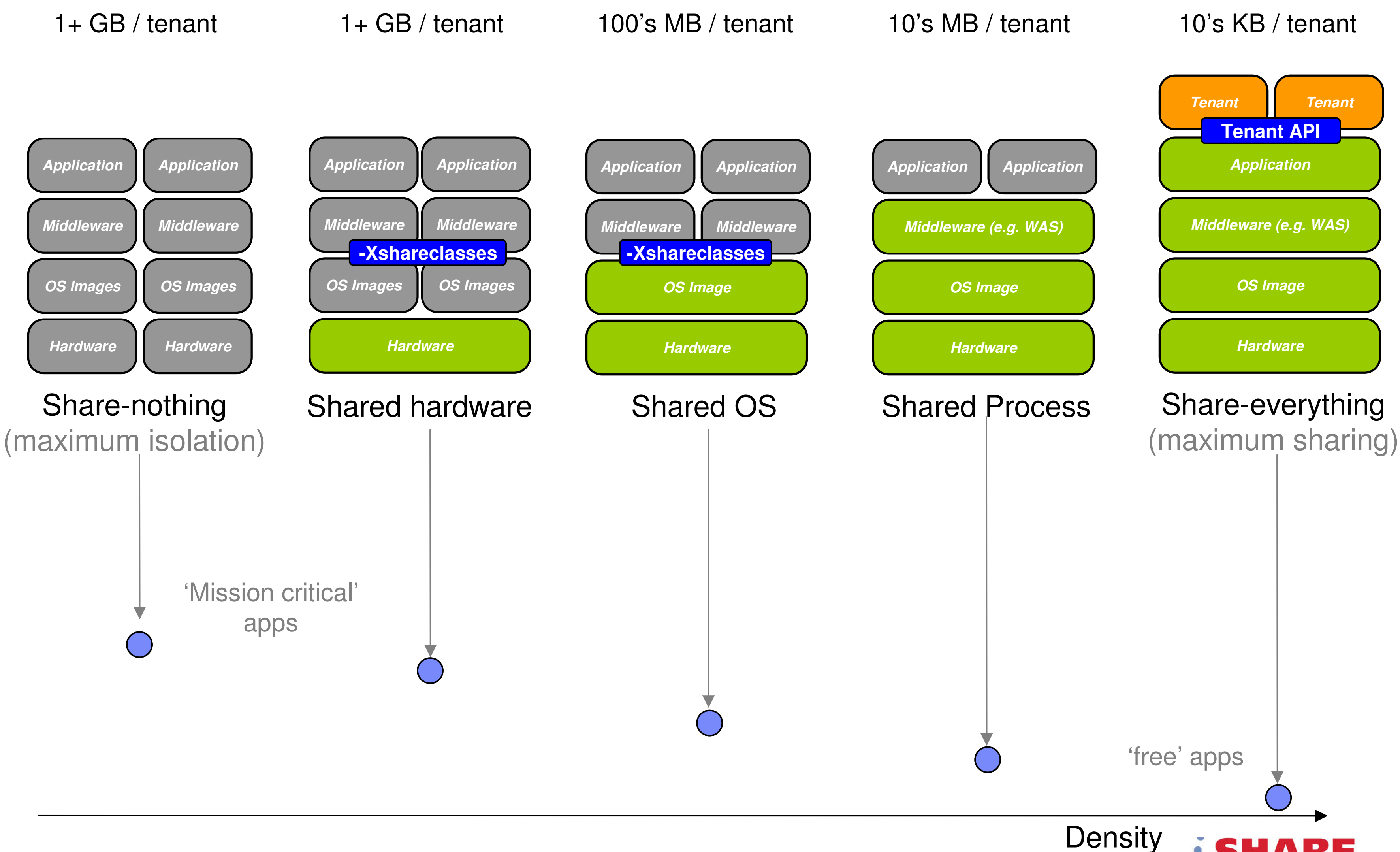
Timelines and deliveries are subject to change.

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

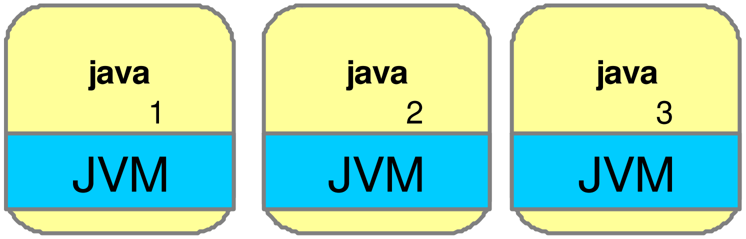
- **JIT support for "deep idle" state**
  - Enabled with **-Xtune:virtualized** (Java 7 SR4)
  - Reduces CPU cycles used by the JIT during idle periods
    - Important for dense virtualized System z environments
    - Early results with WAS Liberty show ~2x to ~6x reduction

Economies of Scale for Java in the Cloud

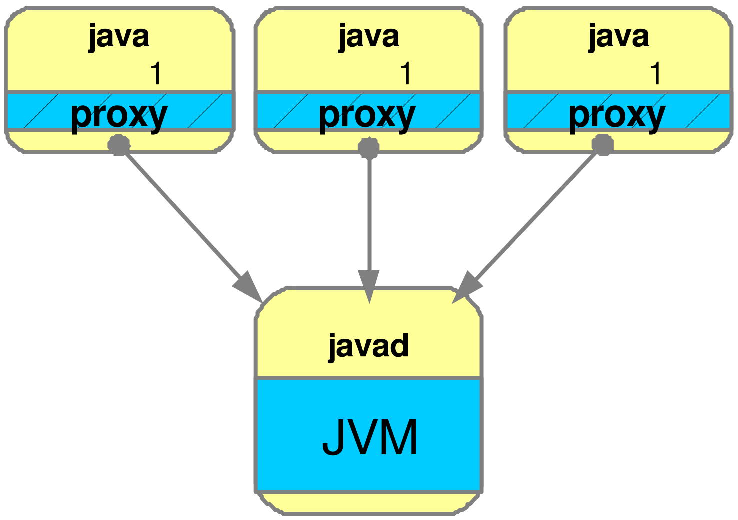


Java 7R1 Tech Preview:
Multi-tenancy: **IBM's approach to 'Virtualized JVMs'**

- A standard 'java' invocation creates a dedicated (non-shared) JVM in each process



- IBM's Multitenant JVM puts a lightweight 'proxy' JVM in each 'java' invocation. The 'proxy' knows how to communicate with the shared JVM daemon called javad.



- 'javad' is launched and shuts down automatically
- No changes required to the application
- 'javad' process is where aggressive sharing of runtime artifacts happens

Java8: Language Innovation – Lambdas and Parallelism

New syntax to allow concise code snippets and expression

- Useful for sending code to java.lang.concurrent
- On the path to enabling more parallelisms

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

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

More Information on Java 8 Lambdas:

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

IBM J9 Garbage Collector Family

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

64-bit Java Performance : Compressed References

- 32-bit Object (24 bytes – 100%)

- 64-bit Object (48 bytes – 50%)

- 64-bit Compressed References (24 bytes – 100%)

Use 32-bit values (offsets) to represent object fields

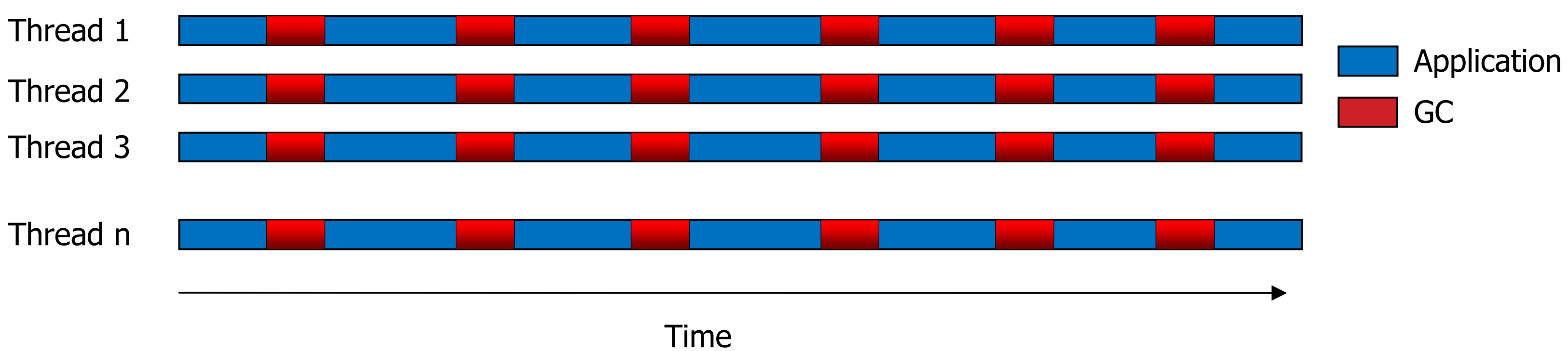
With scaling, between 4 GB and 32 GB can be addressed

- **32-bit address space**
  - Theoretically: 4GB of addressable virtual memory
  - Realistically: less than 4GB due to link libs, execs, stack placement etc
    - Java heap needs to be contiguous
    - Native application code, J9/TR runtimes and data, OS modules
  - Customers reaching limits (OOM Exceptions)
- **Move to 64-bit pointers is not free**
  - Objects on average ~60% bigger
    - ~ 60% increase in Java heap footprint (smaller heap occupancy ratio)
  - Increased Cache/TLB pressure
    - Addressability increased, hardware remained constant (throughput effects)
- **Concerted investment in Java6 JRE**
  - Large Pages Technology
  - Compressed References Technology

- Option to enable compression in 64-bit Java 6 SR4, WAS 7 (Service Pack 3)
  - use –Xcompressedrefs option
- Java objects are 8-byte aligned
  - Low 3 bits of object address = 000
- Address range restriction
  - Java heap allocated in $2^{31} – 2^{35}$ range (2GB – 32GB virtual)
  - High 29 bits of object address = 000 … 000
  - 32 out 64 bits are 0!
- Store 32-bit shifted offset in objects
  - Shift values of 0 through 3 are used
  - Maximum allowable heap is ~32GB, Actual allowed heap depends on shift value and virtual memory fragmentation
- Reference whitepaper: http://tiny.cc/mi4fgw

IBM J9 Garbage Collector: -Xgcpolicy:optthruput

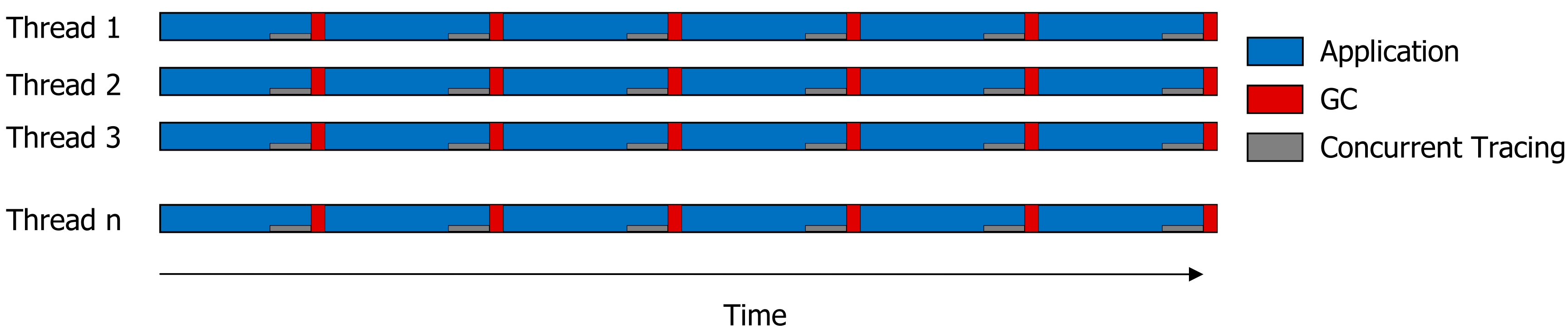
- Default policy in Java 5 and Java 6
- Used where raw throughput is more important than short GC pauses
- Application stopped whenever garbage is collected



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



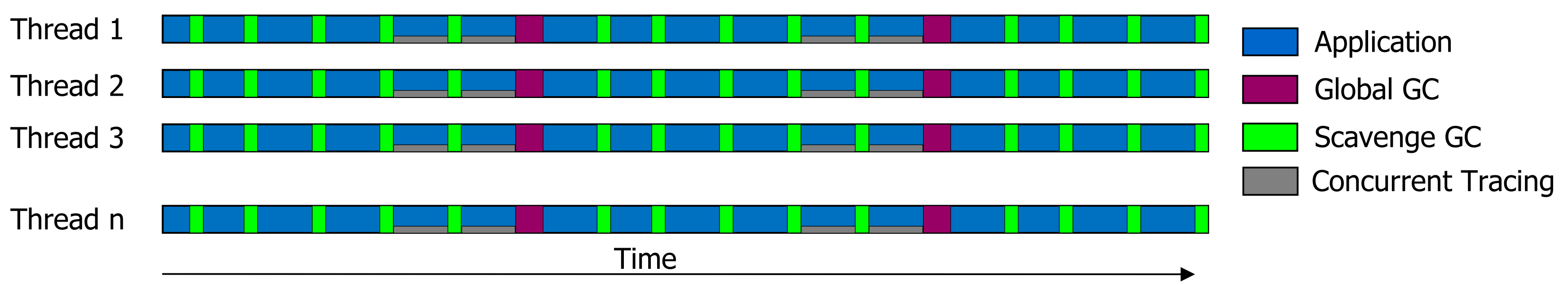
Picture is only illustrative and doesn't reflect any particular real-life application. The purpose is to show theoretical differences in pause times between GC policies.

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

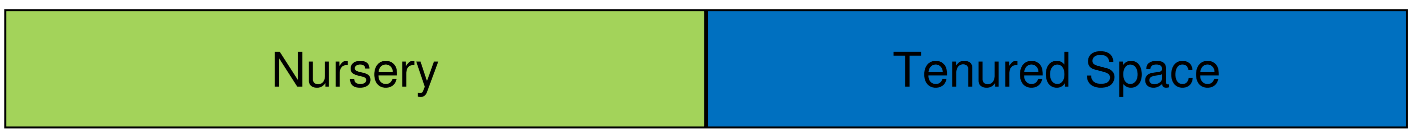
- Default policy in Java 6.0.1 and Java 7
- Applications with many short-lived objects benefit from shorter pause times while still producing good throughput



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.

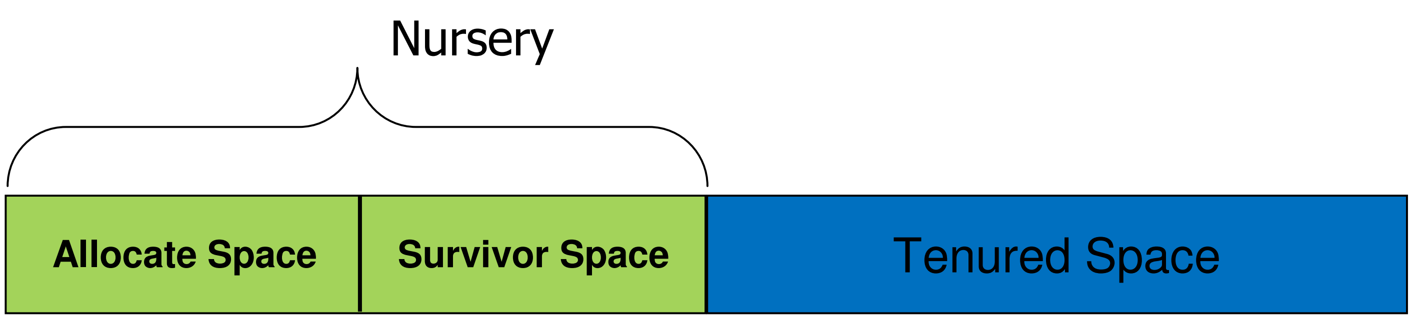
IBM J9 Garbage Collector: -Xgcpolicy:gencon

- 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)



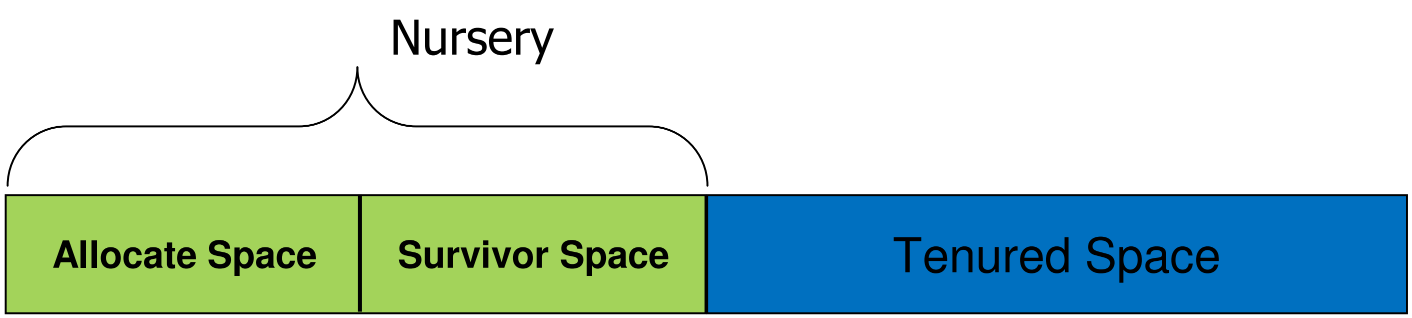
IBM J9 Garbage Collector: -Xgcpolicy:gencon

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



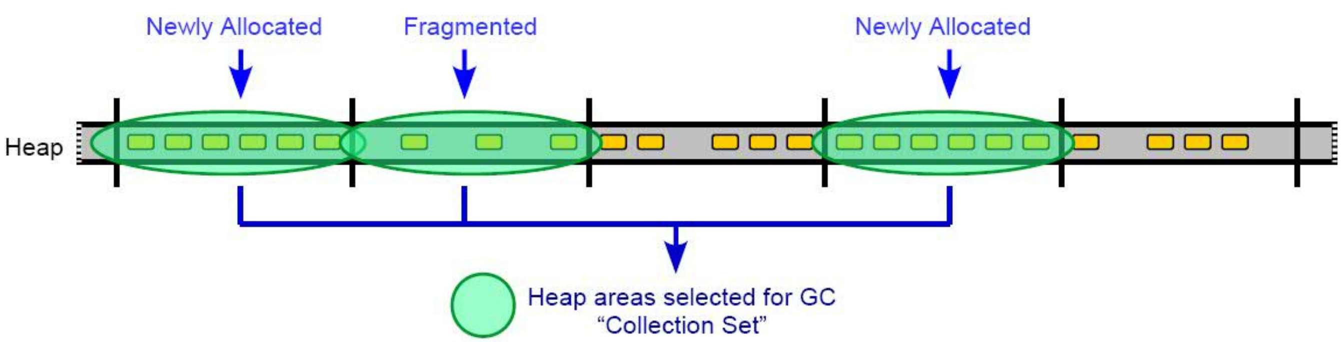
IBM J9 Garbage Collector: -Xgcpolicy:gencon

- 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



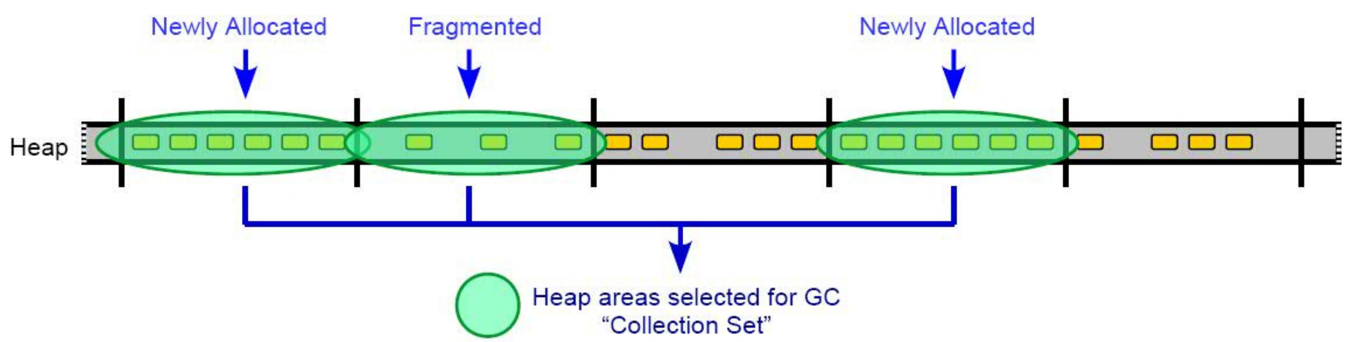
IBM J9 2.6 Enhancement: -Xgcpolicy:balanced

- **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



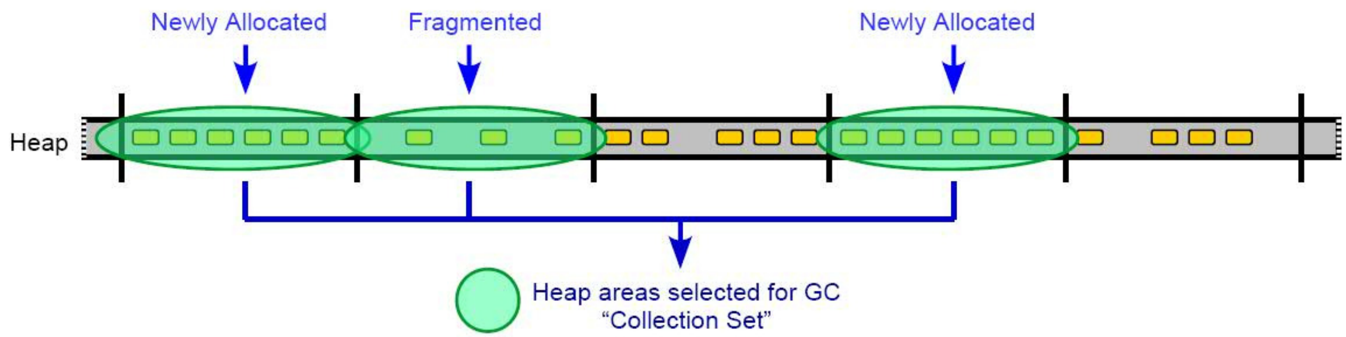
IBM J9 2.6 Enhancement: -Xgcpolicy:balanced

- **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()

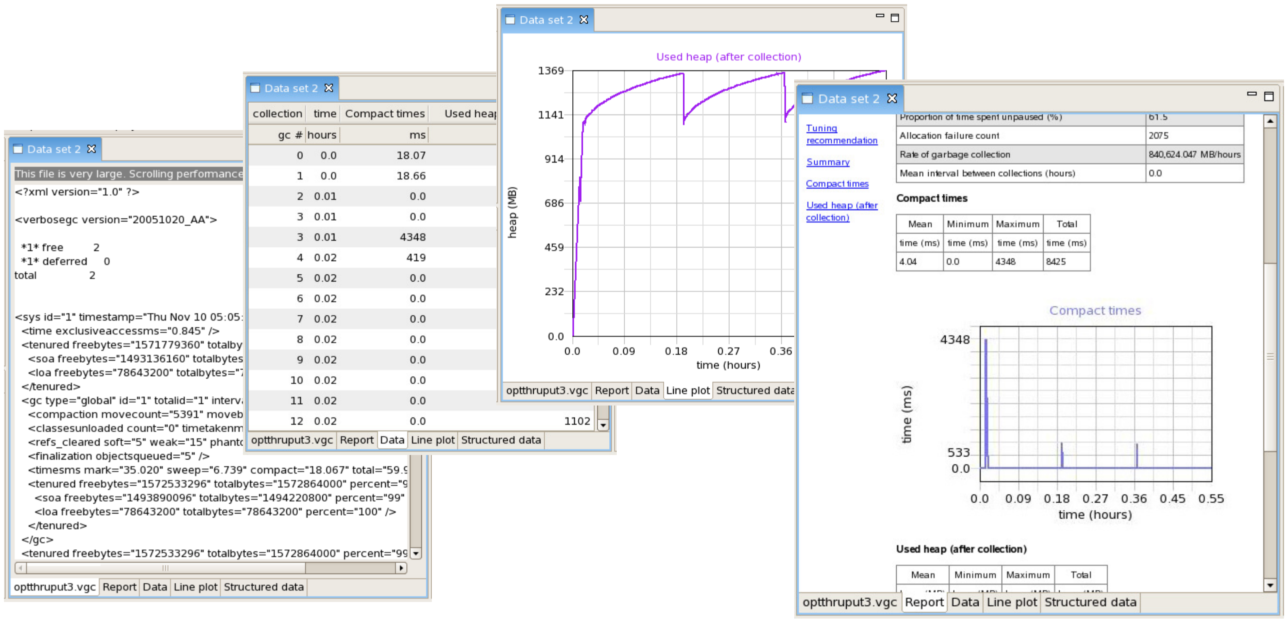
IBM J9 Garbage Collector: Tuning

- 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



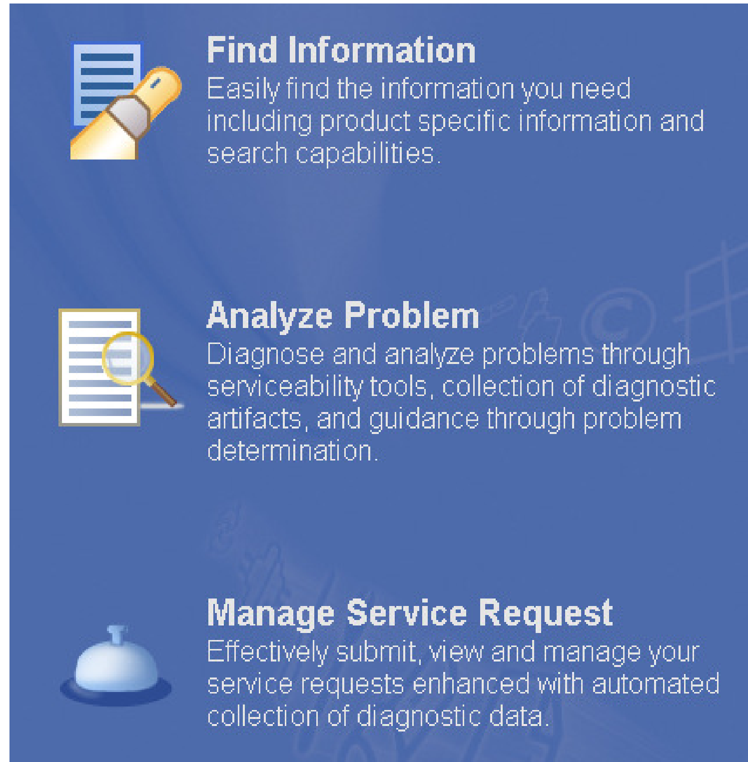
IBM J9 Garbage Collector: Tuning

- 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

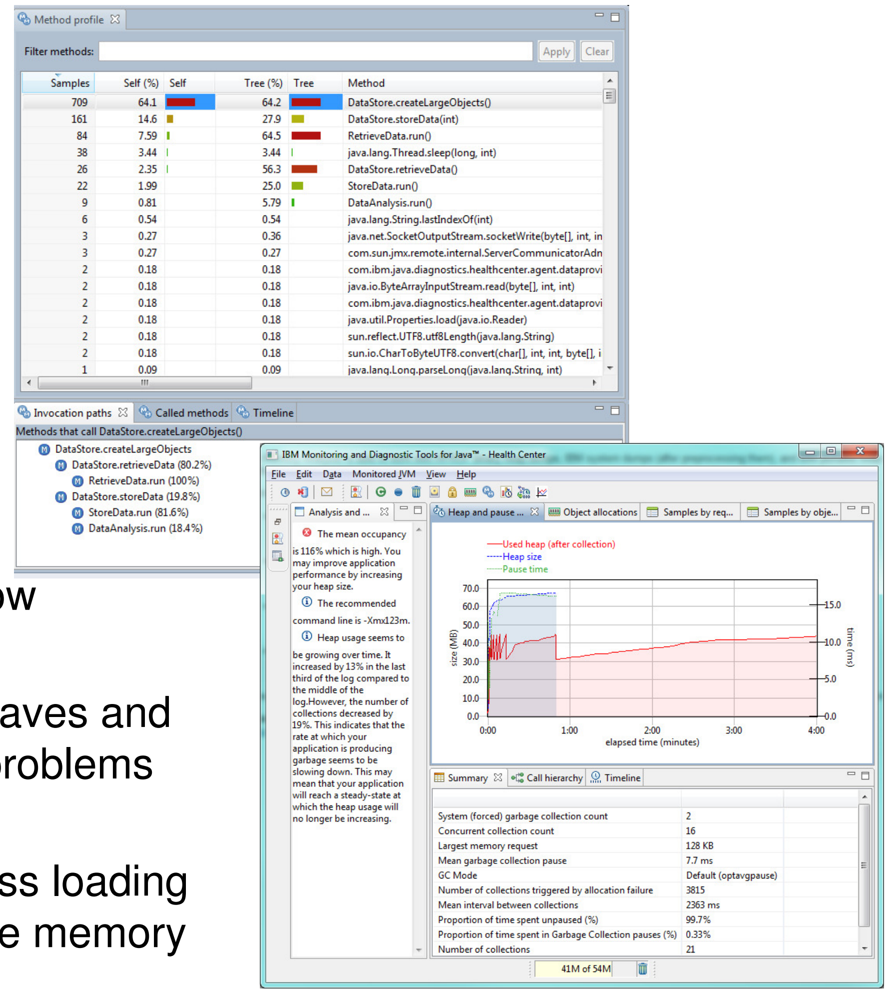


14709: Need a Support Assistant? Check Out IBM's! (ISA)

Thursday, March 13, 2014: 8:00 AM-9:00 AM
Grand Ballroom Salon A (Anaheim Marriott Hotel)
Speaker: Michael Stephen(IBM Corporation)

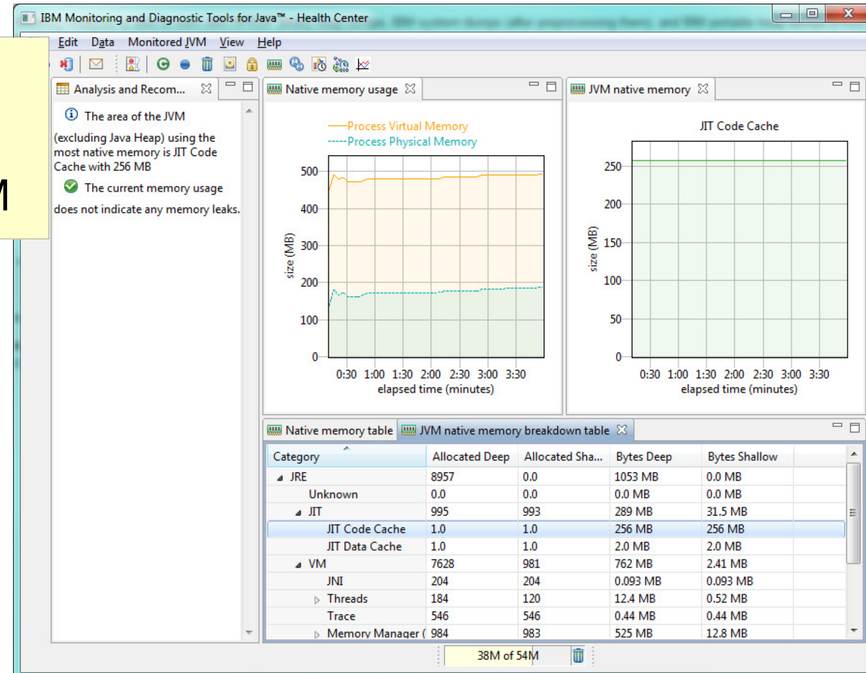
IBM Monitoring and Diagnostic Tools for Java: Health Center

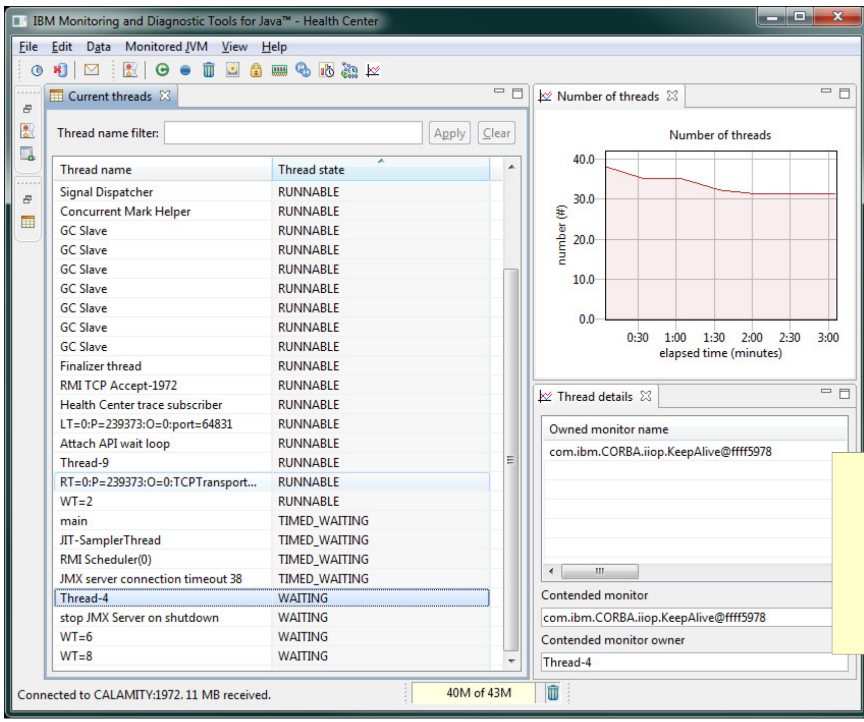
- **What problem am I solving?**
  - What is my JVM doing? Is everything OK?
  - Why is my application running slowly?
  - Why is it not scaling?
  - Am I using the right JVM options?
- **Health Center Overview**
  - Lightweight monitoring tool with very low overhead
  - Understands how your application 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



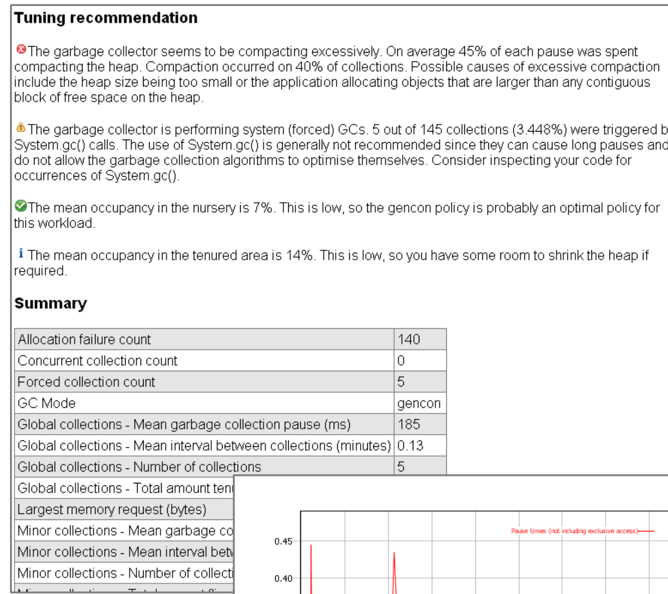


Keep track of running threads and monitor contention

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.

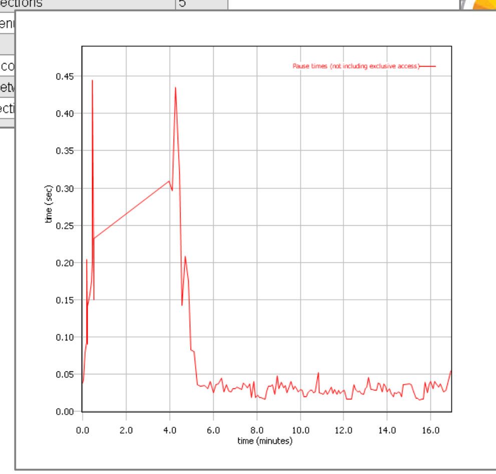
The garbage collector is performing system (forced) GCs. 5 out of 145 collections (3.448%) were triggered by System.gc() calls. The use of System.gc() is generally not recommended since they can cause long pauses and do not allow the garbage collection algorithms to optimise themselves. Consider inspecting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

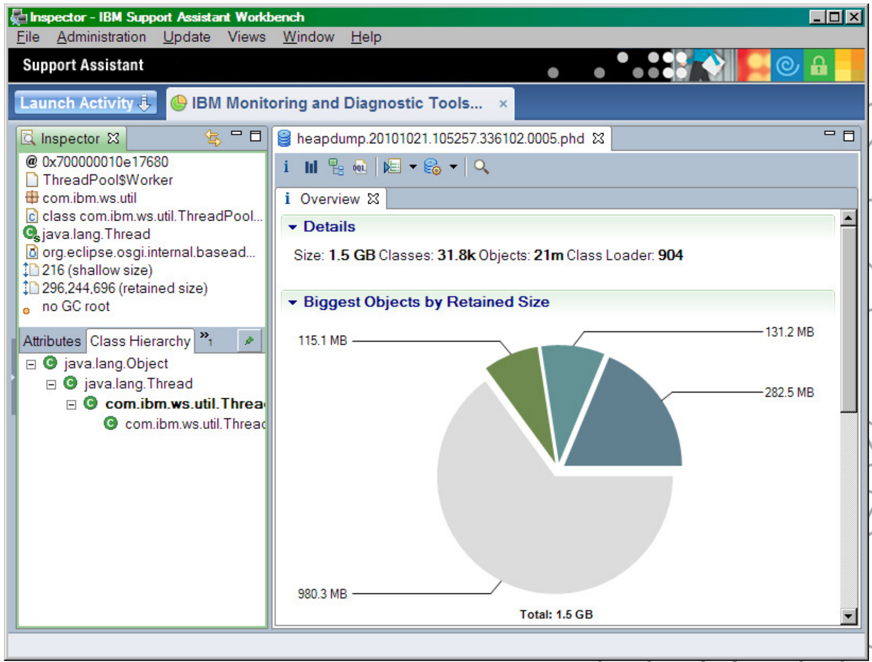
Summary

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
Global collections - Total amount ten[illegible]	[illegible]
Largest memory request (bytes)	[illegible]
Minor collections - Mean garbage co[illegible]	[illegible]
Minor collections - Mean interval betw[illegible]	[illegible]
Minor collections - Number of collecti[illegible]	[illegible]



IBM Monitoring and Diagnostic Tools for Java: Memory Analyzer

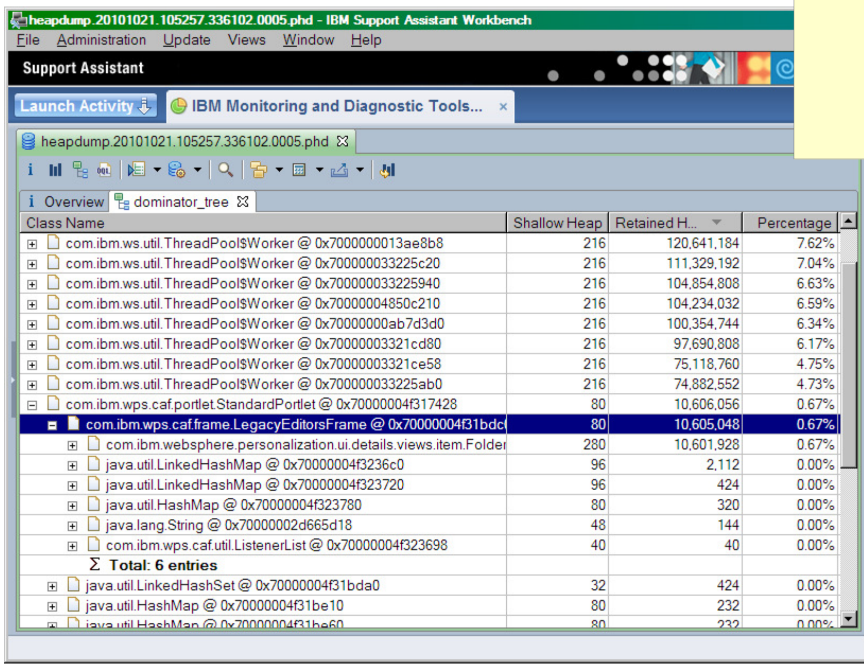
- **What problem am I solving?**
  - Why did I run out of Java memory?
  - What's in my Java heap? How can I explore it and get new insights?
- **Memory Analyzer Overview**
  - Examines memory dumps and identifies Java memory leaks
  - Analyzes footprint and provides insight into wasted space



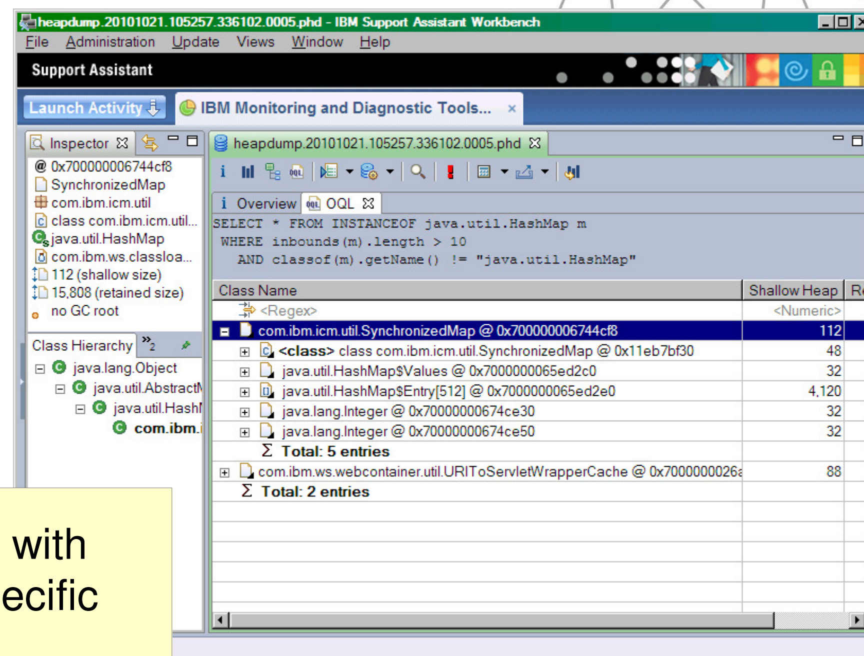
Features: visual objects by size/class/classloader, dominator tree analysis, path to GC roots analysis, object query language (OQL)

- 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



Build custom queries with OQL to search for specific object instances

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

14955: IDDE 1.0 Features and Futures

Thursday, March 13, 2014: 9:30 AM-10:30 AM
Grand Ballroom Salon B (Anaheim Marriott Hotel)
Speaker: Kenneth Irwin(IBM Corporation)

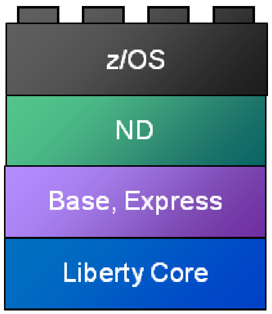
Questions?


Thank You

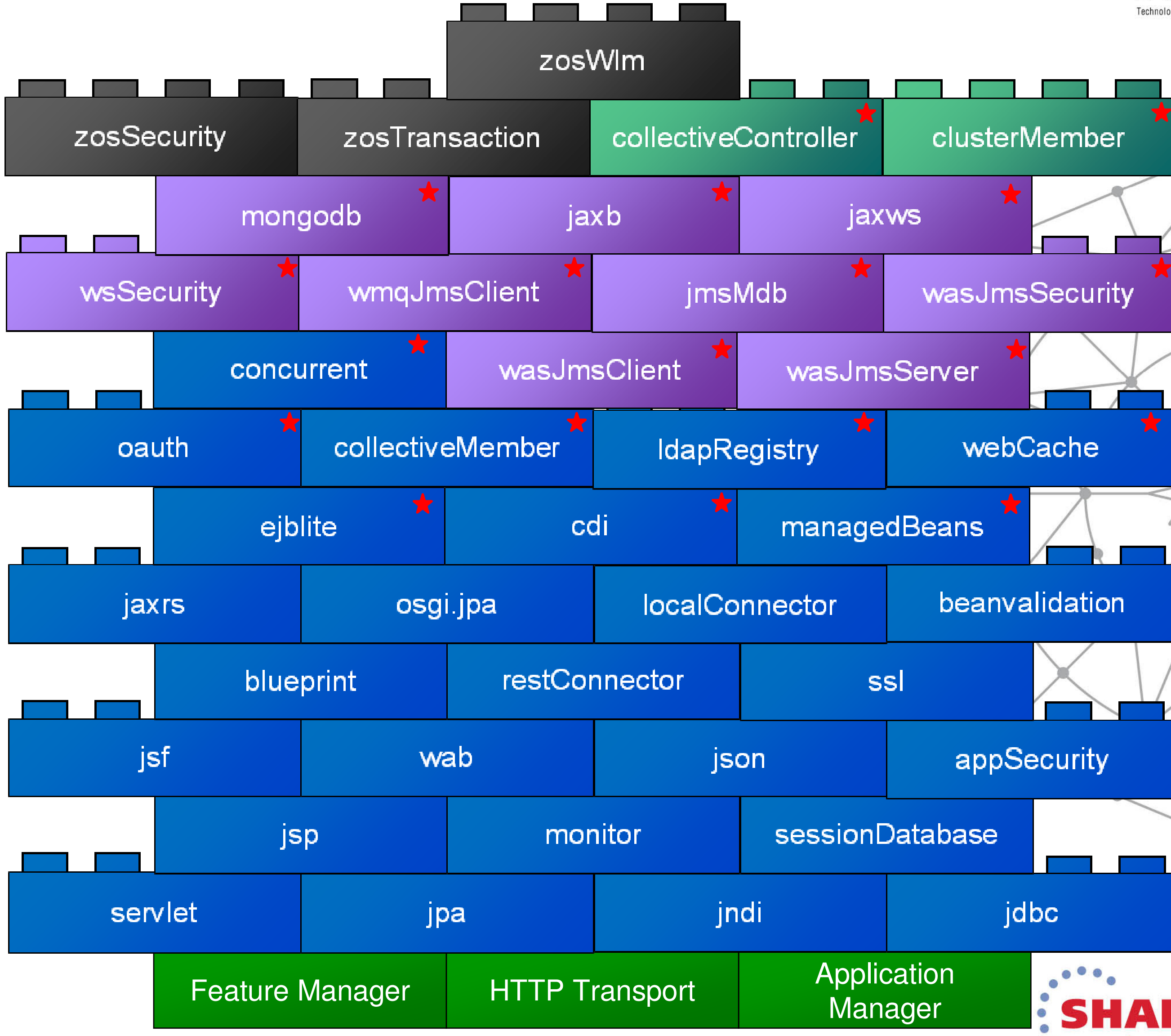
Important references

- IBM Java on Linux for System z
  - http://www.ibm.com/developerworks/java/jdk/linux/download.html
- IBM z/OS Java web site
  - http://www.ibm.com/systems/z/os/zos/tools/java/
- IBM Java documentation
  - http://www.ibm.com/developerworks/java/jdk/docs.html
- IBM Java Diagnostic and Monitoring Tools
  - http://www.ibm.com/developerworks/java/jdk/tools/index.html
- White paper on 64-bit compressed references and large pages features in IBM 64-bit Java SDK on System z
  - https://ibm.biz/BdRmRD
- JZOS Batch Launcher and Toolkit Installation and User's Guide (SA38-0696-00)
  - http://publibz.boulder.ibm.com/epubs/pdf/ajvc0110.pdf

Liberty feature set as of V8.5.5

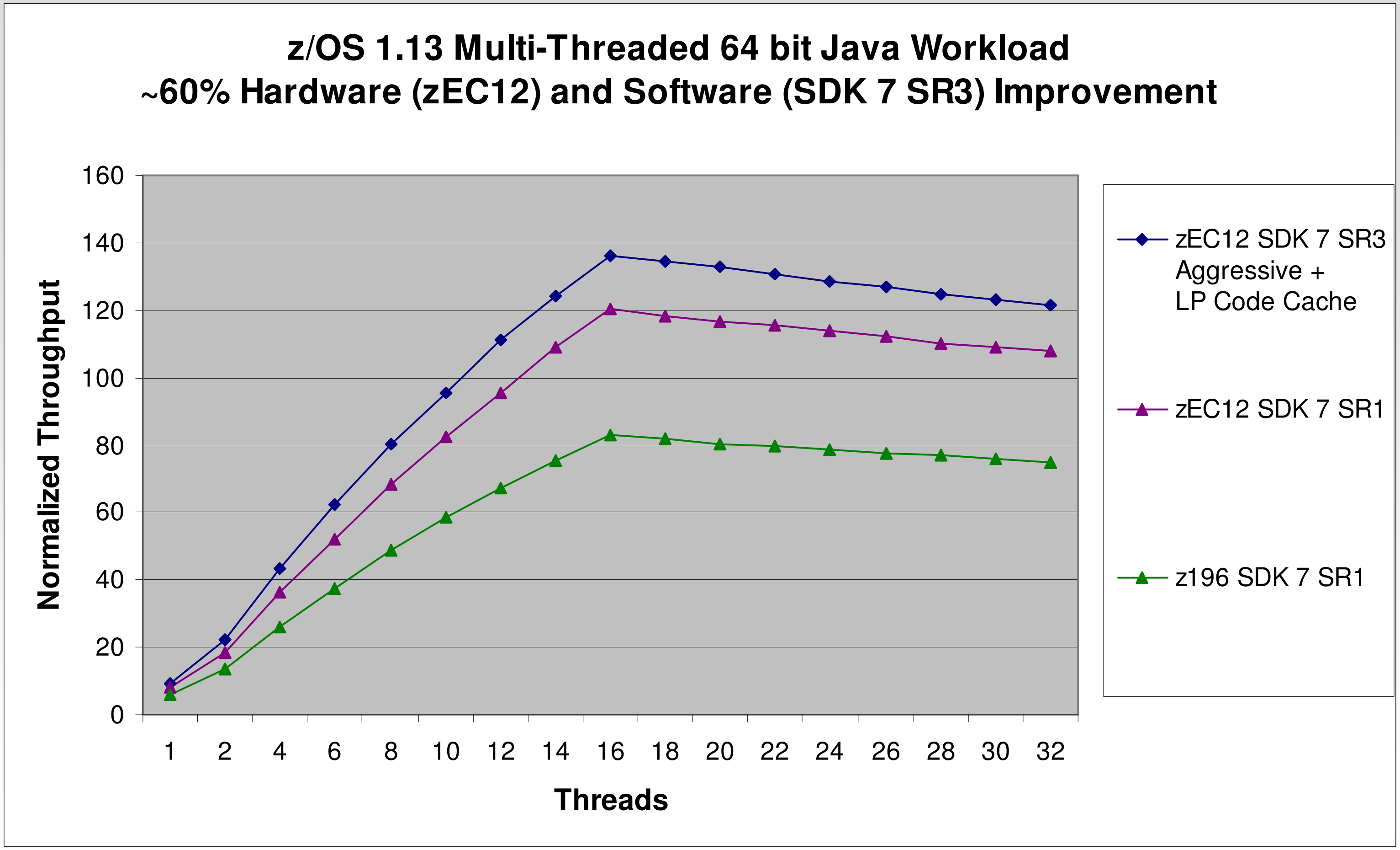


WAS V8.5.5 Liberty Profile★



z/OS IBM Java 7:16-Way Performance

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

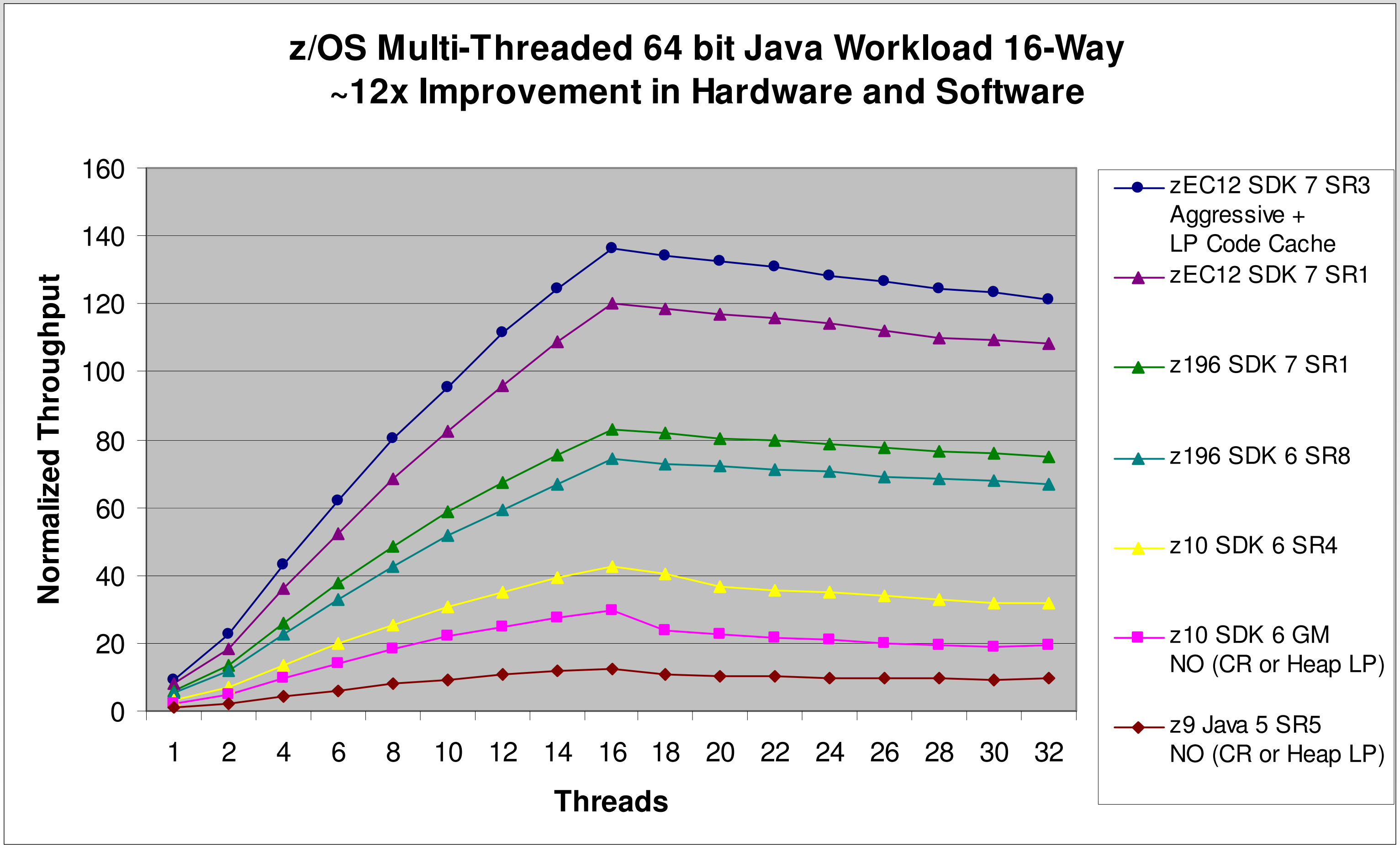


Aggregate 60% improvement from zEC12 and IBM Java7 (Controlled measurement environment, results may vary)

- **zEC12 offers a ~45% improvement over z196 running the Java Multi-Threaded Benchmark**
- **IBM Java 7 offers an additional ~13% improvement** **(sr3 + -Xaggressive + Flash Express pageable 1Meg large pages)**

z/OS IBM Java 7: 16-Way Performance

Aggregate HW and SDK Improvement z9 IBM Java 5 to zEC12 IBM Java 7



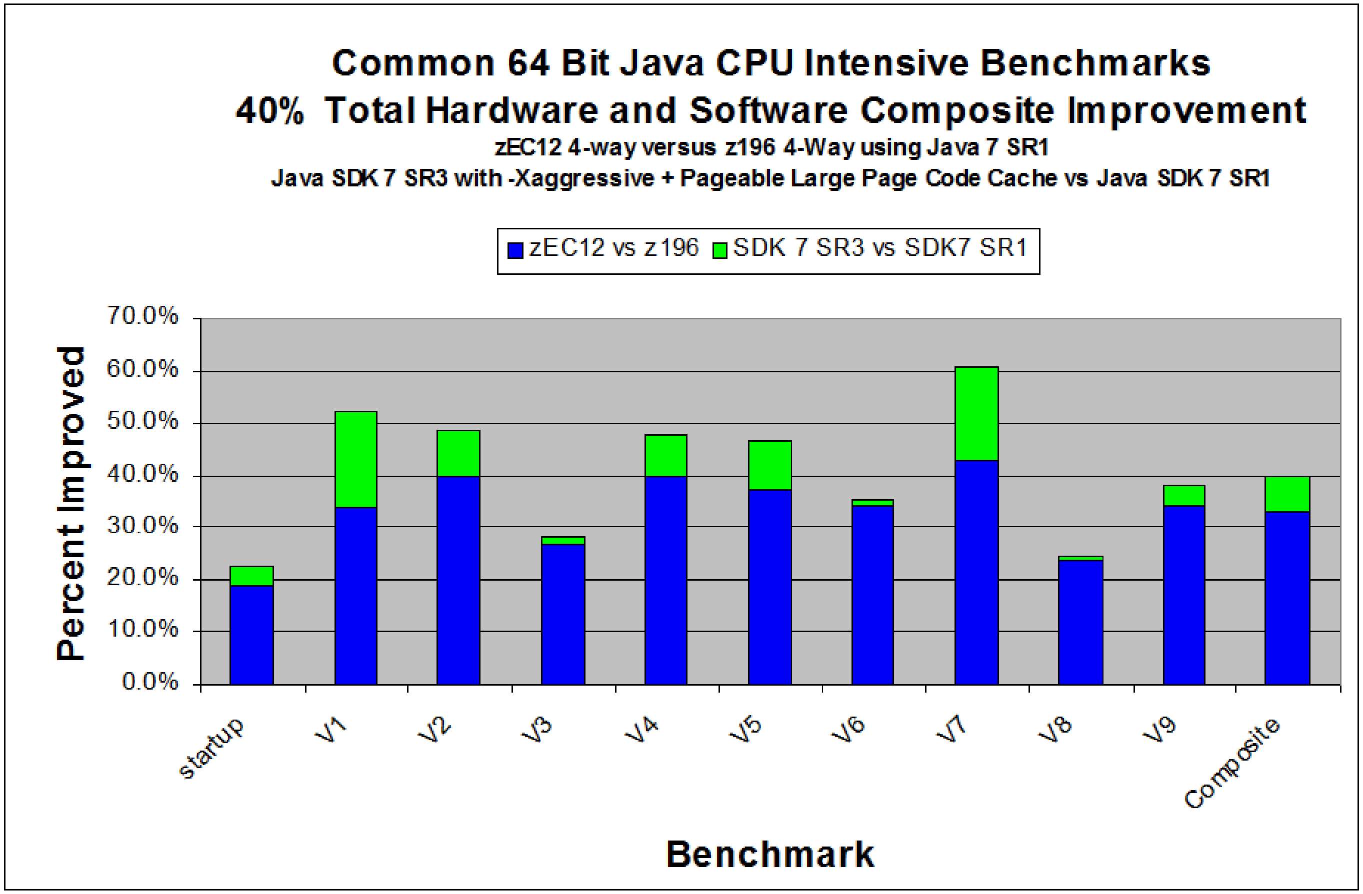
~12x aggregate hardware and software improvement comparing IBM Java5 on z9 to IBM Java 7 on zEC12

LP=Large Pages for Java heap **CR= Java compressed references**

(Controlled measurement environment, results may vary)

Java7SR3 using -Xaggressive + Flash Express pageable 1Meg large pages

z/OS IBM Java 7: CPU-Intensive Benchmark

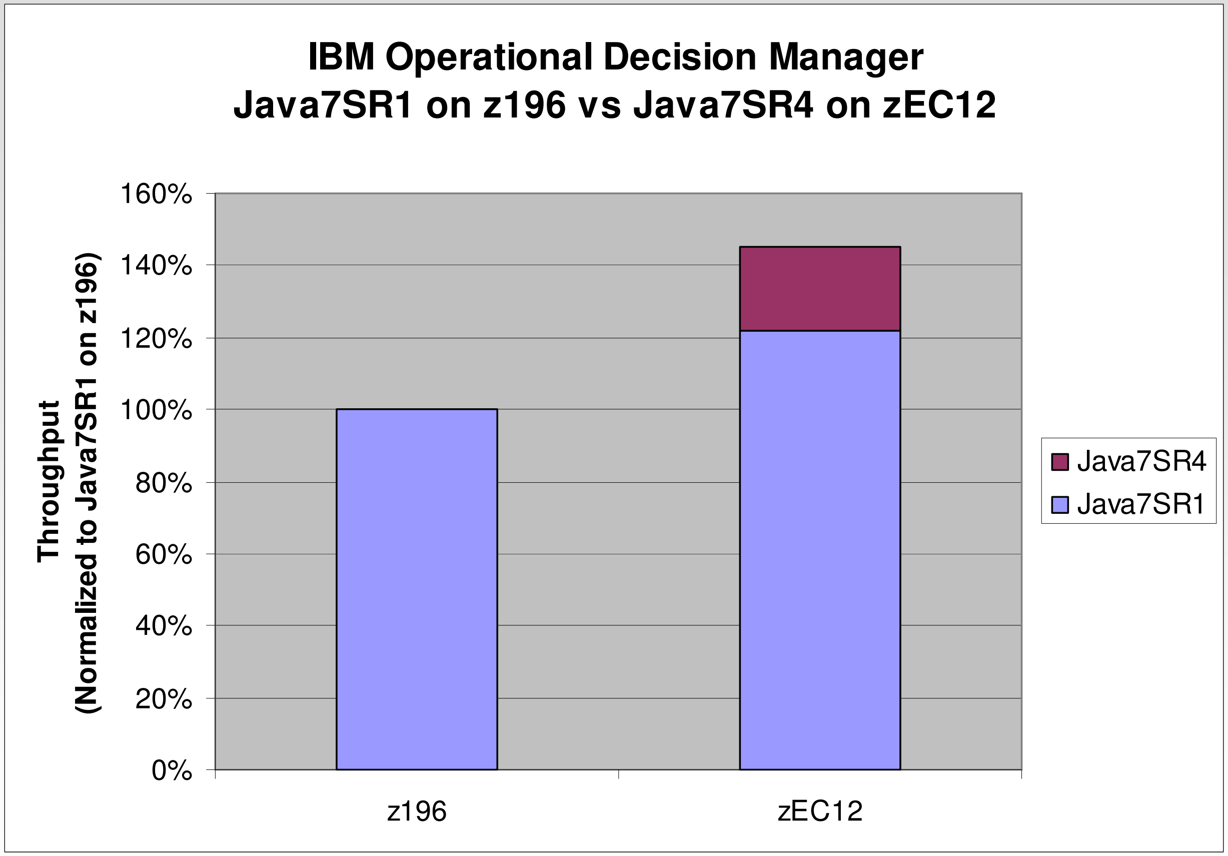


zEC12 and IBM Java 7 offer a ~40% composite improvement over z196 running the CPU Intensive benchmark

- **zEC12 offers a ~33% improvement over z196 running the CPU-Intensive Benchmarks**
- **IBM Java 7 offers an additional ~5% improvement (SR3 + -Xaggressive + Flash Express pageable 1Meg large pages)**

(Controlled measurement environment, results may vary)

IBM Operational Decision Manager with IBM Java 7 and zEC12



(Controlled measurement environment, results may vary)

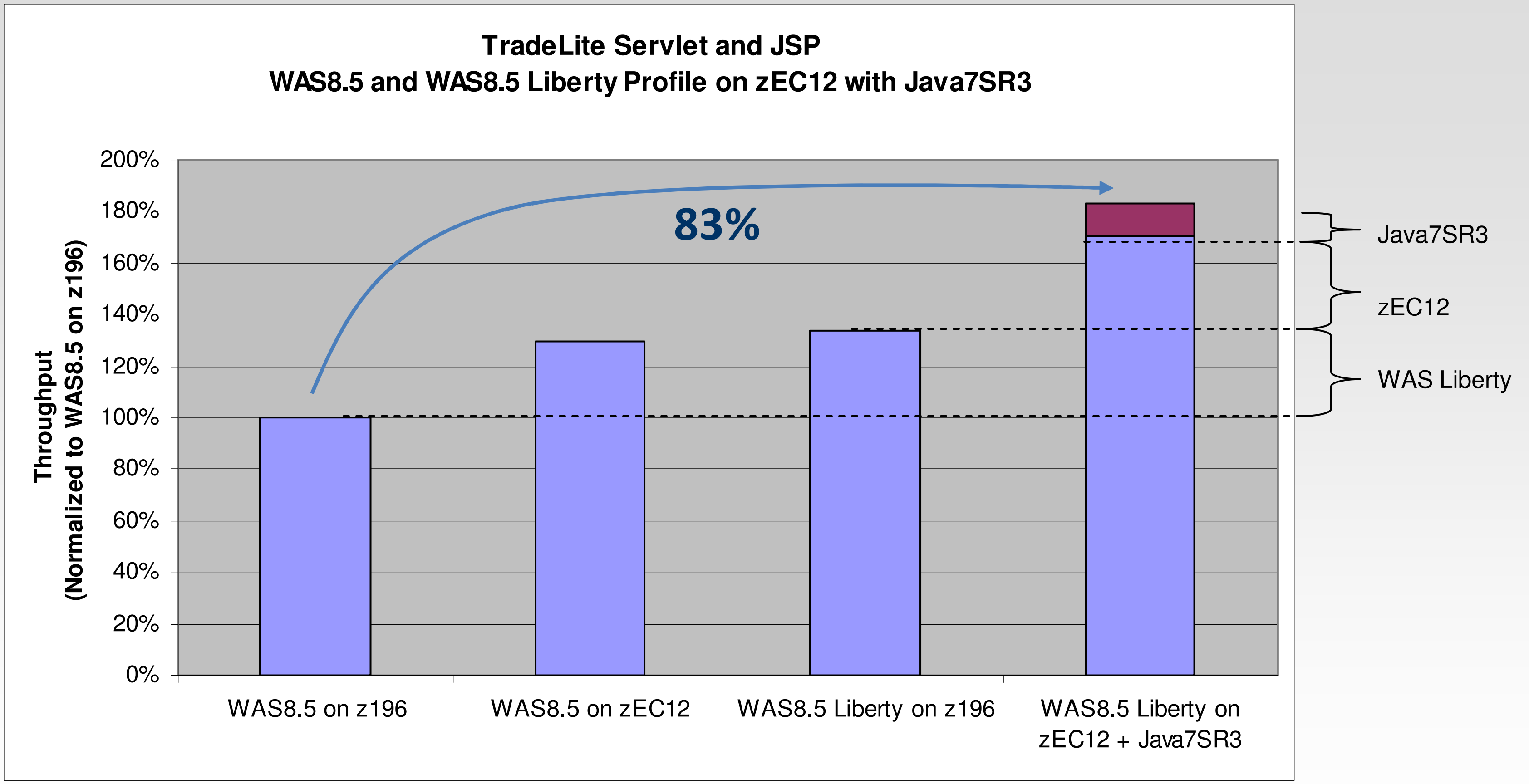
Aggregate 45% improvement from zEC12 and IBM Java7

- **zEC12 offers a ~22% improvement over z196 running the ODM Benchmark**
- **IBM Java7 offers an additional ~19% improvement** **(SR4 + -Xaggressive + Flash Express pageable 1Meg large pages)**

WAS on z/OS – Servlets and JSPs with the Liberty Profile**

** See backup charts for list of Liberty Profile capabilities

++ Java7SR3 using -Xaggressive + Flash Express pageable 1Meg large pages

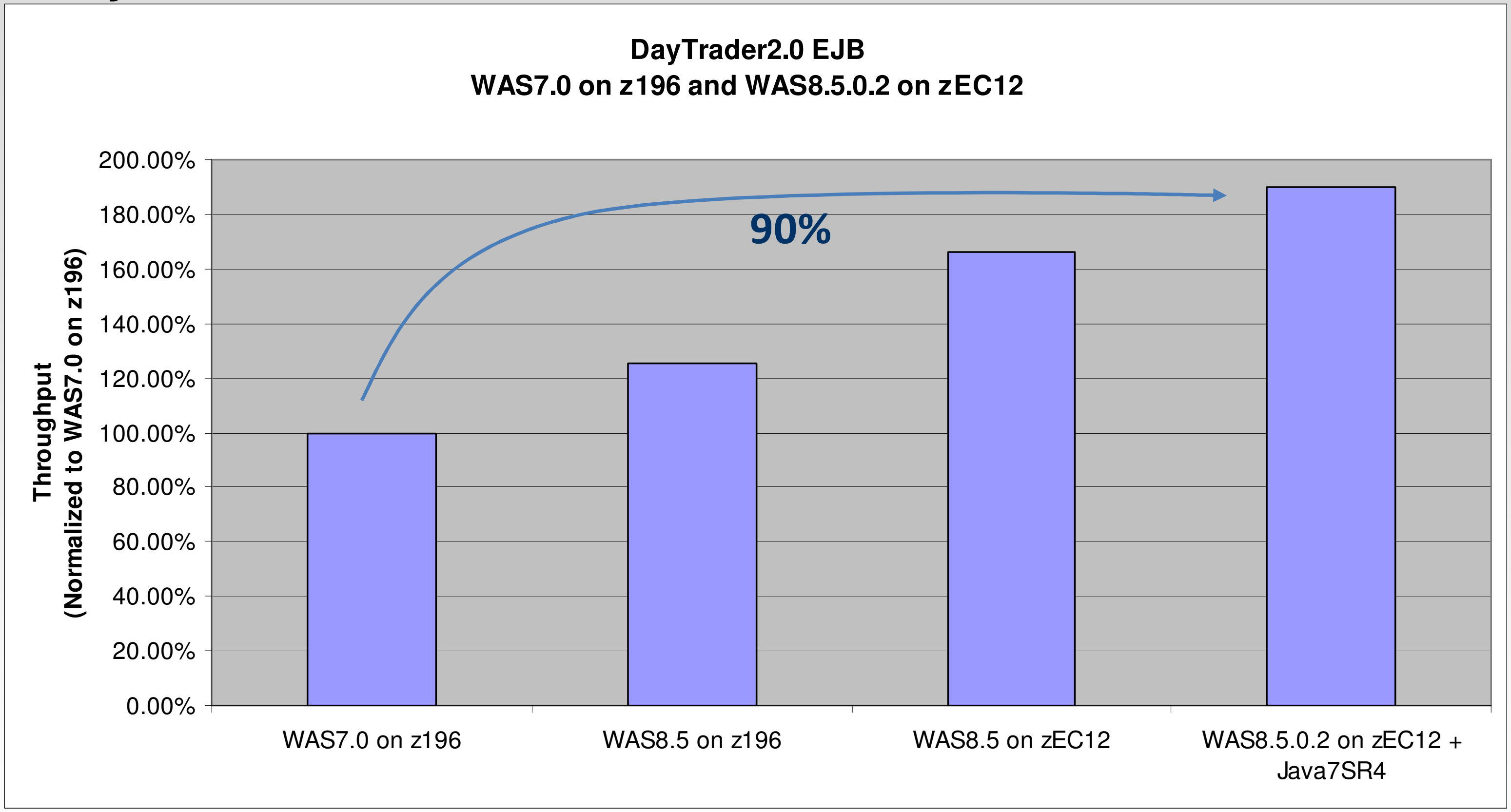


- **WAS8.5 Liberty on zEC12 using IBM Java 7 vs WAS8.5 on z196 running TradeLite demonstrates a 83% improvement to Servlet and JSP throughput.**
- **WAS8.5 Liberty offers up to 5x start-up time reduction vs. WAS8.5 (<5 seconds)**
- **WAS8.5 Liberty offers reduced real-storage requirements up to 81% vs. WAS8.5 (80M versus 420M)**

(Controlled measurement environment, results may vary)

WAS on z/OS –

DayTrader2.0 EJB – Then and Now

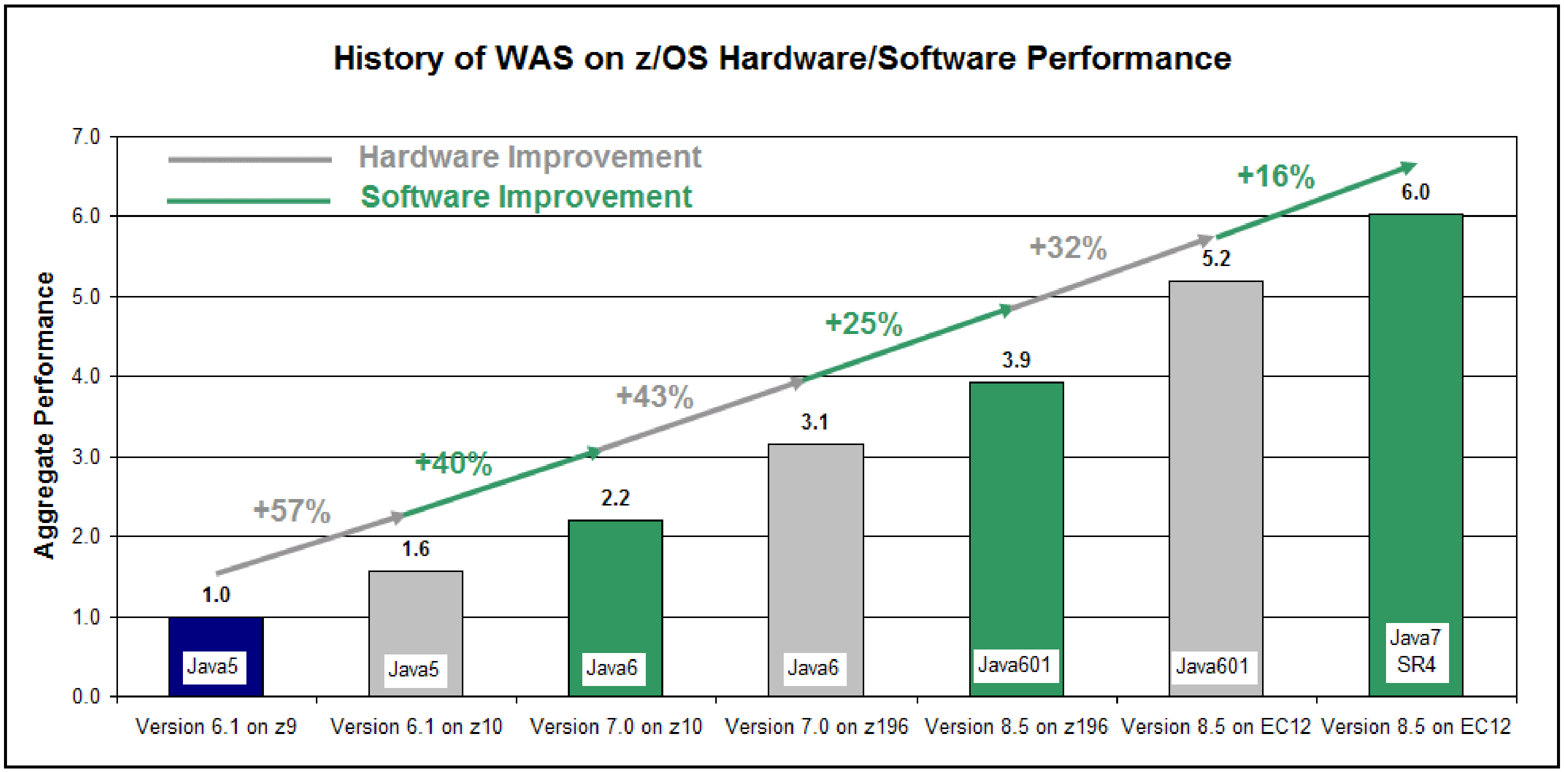


- **WAS8.5.0.2 on zEC12 vs WAS7 on z196 running DayTrader2.0 EJB demonstrates a 90% improvement to throughput**
- **zEC12 improves WAS8.5 throughput by up to 32% over z196**
- **WAS8.5 improves throughput by up to 25% over WAS7.0**
- **WAS8.5.0.2 uses IBM Java 7 which improves throughput by an additional 15% over WAS8.5 on zEC12**

(Controlled measurement environment, results may vary)

WAS on z/OS – DayTrader

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

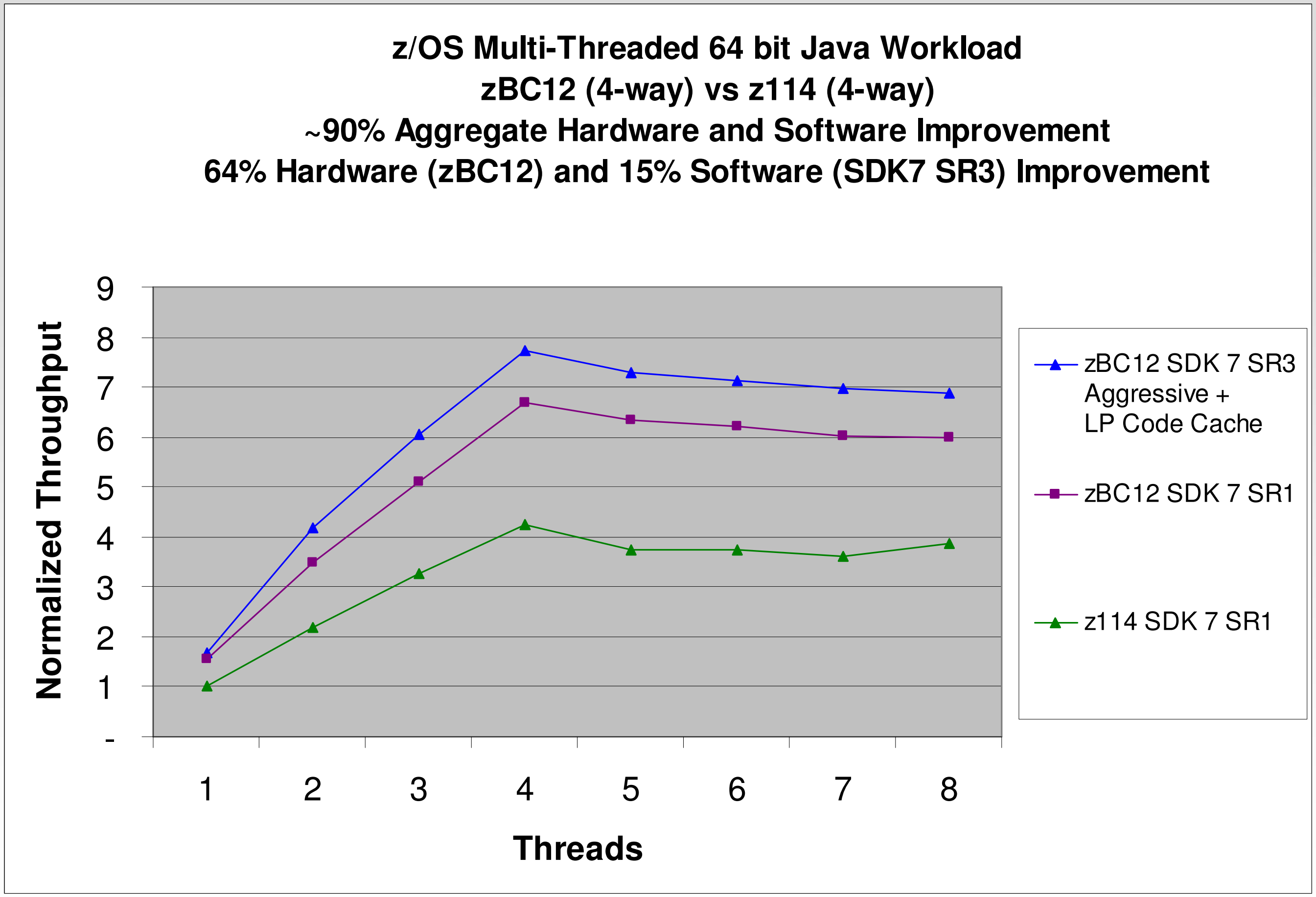


6x aggregate hardware and software improvement comparing WAS 6.1 IBM Java5 on z9 to WAS 8.5 IBM Java7 on zEC12

(Controlled measurement environment, results may vary)

z/OS Java SDK 7: zBC12 4-way Performance

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

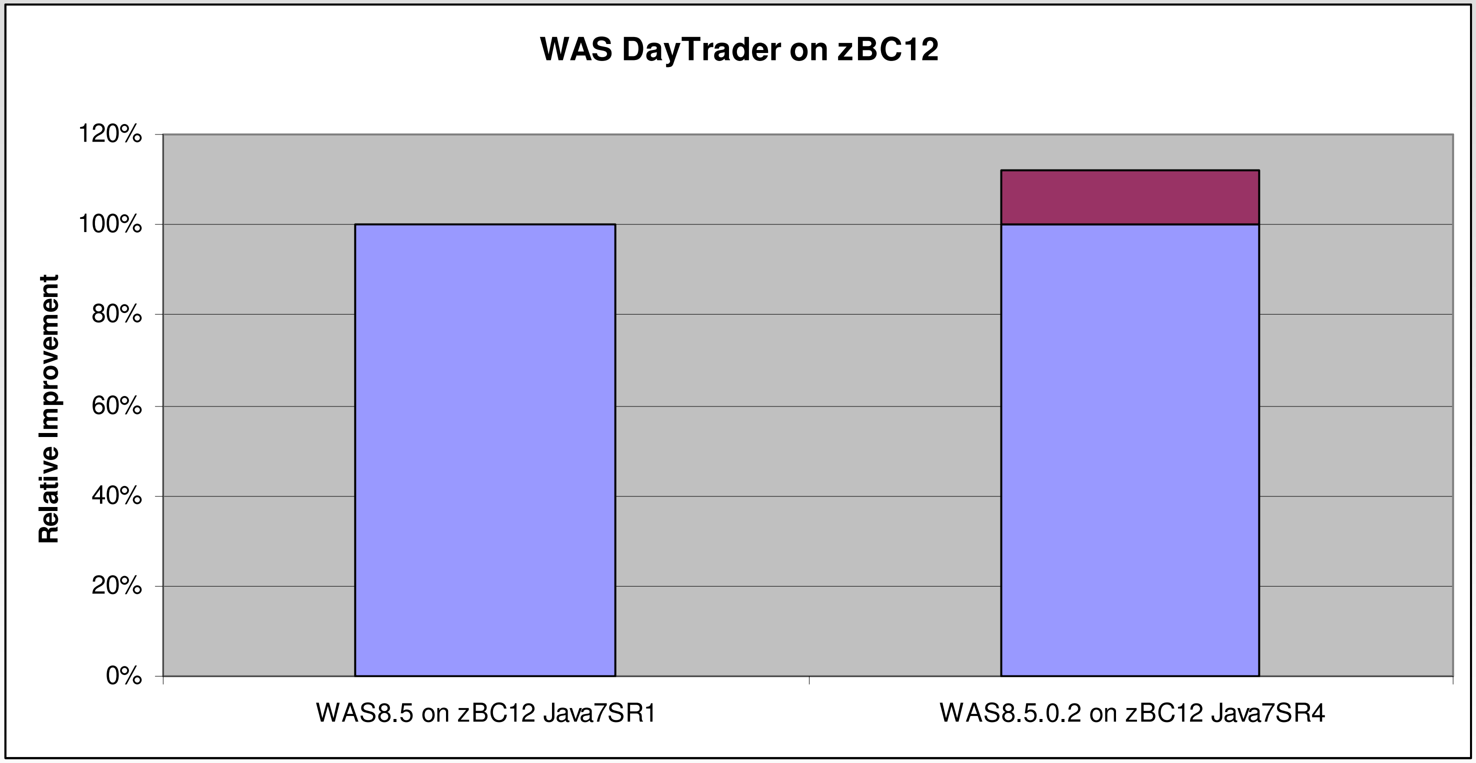


Aggregate 90% improvement from zBC12 and Java7SR3 (Controlled measurement environment, results may vary)

- **zBC12 offers a ~64% improvement over z196 running the Java Multi-Threaded Benchmark**
- **Java7SR3 offers an additional ~15% improvement (-Xaggressive + Flash Express pageable 1Meg large pages)**

WAS on z/OS –

DayTrader2.0 EJB on zBC12 with Java7

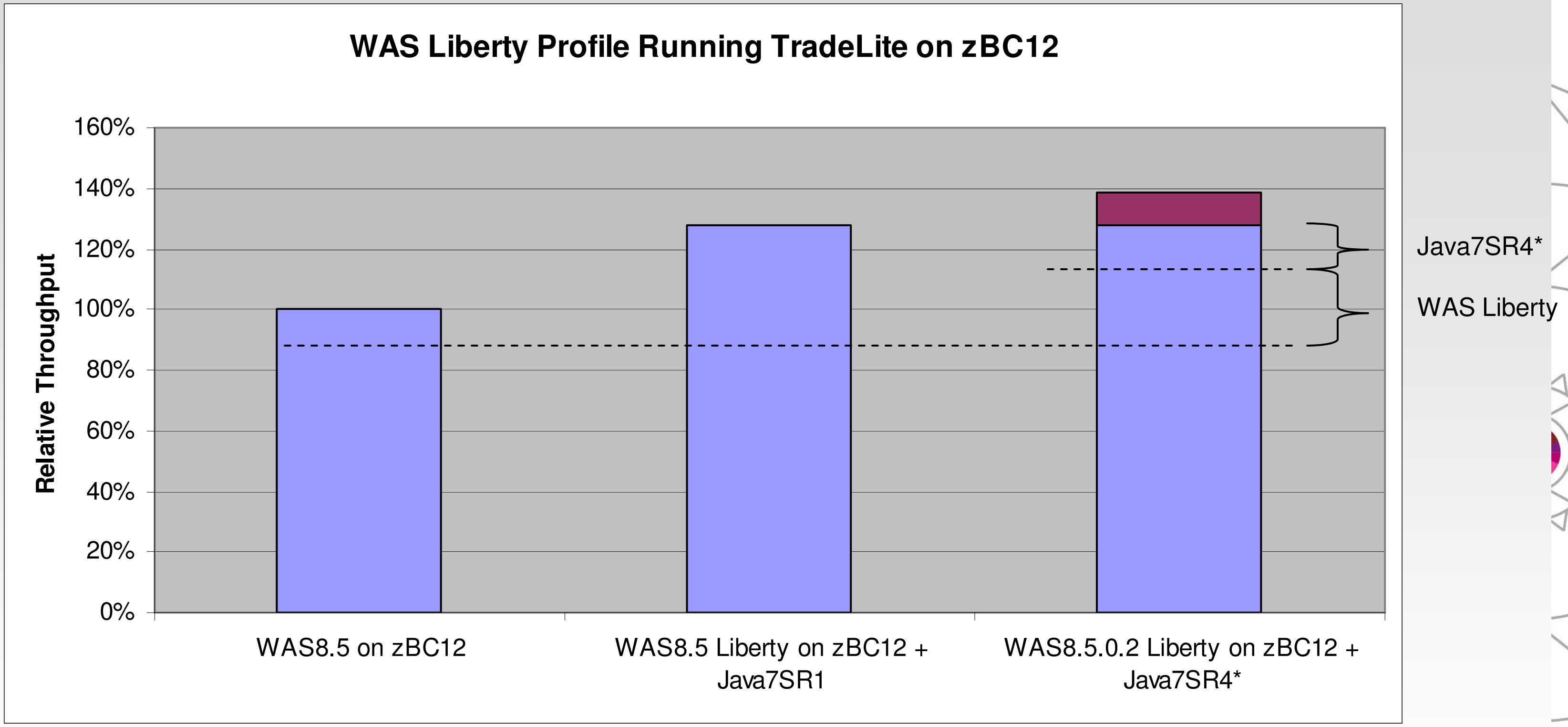


(Controlled measurement environment, results may vary)

- **On zBC12 with z/OS 1.13, WAS 8.5.0.2 full profile (with Java7SR4 *) throughput performance improved 12% over WAS 8.5 full profile running DayTrader.**

* Java7SR4 using -Xaggressive + Flash Express pageable 1Meg large pages

WAS on z/OS – Servlets and JSPs with the Liberty Profile**



(Controlled measurement environment, results may vary)

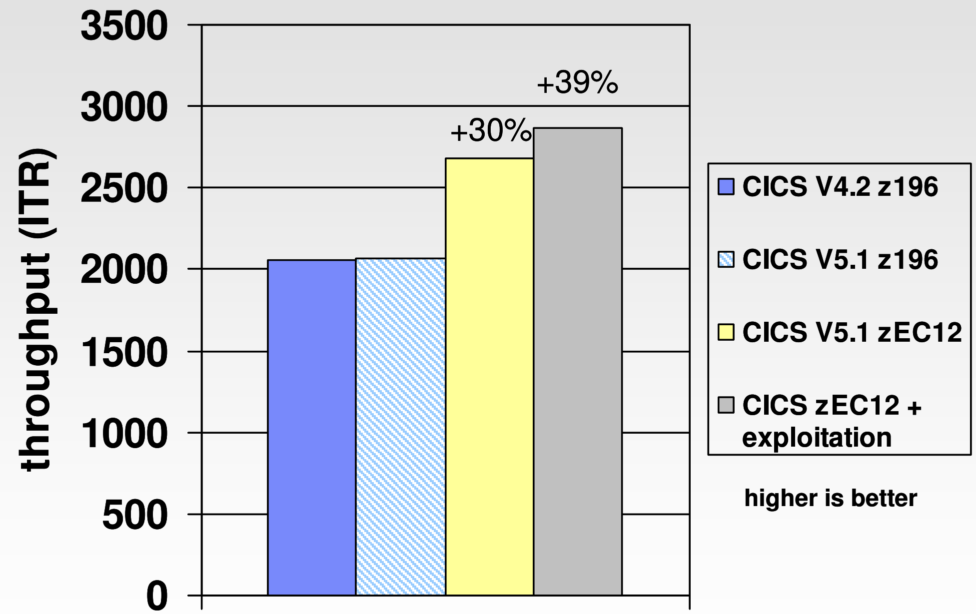
- **On zBC12 with z/OS 1.13, WAS 8.5 liberty profile throughput is improved 28% over WAS 8.5 full profile running TradeLite.**
- **On zBC12 with z/OS 1.13, WAS 8.5.0.2 liberty profile (with Java 7 SR4 *) throughput performance improved 8% over WAS 8.5 liberty profile running TradeLite.**
- **WAS8.5 Liberty using Java7SR4 vs traditional WAS8.5 using Java7SR1 running TradeLite demonstrates a aggregate 38% improvement to Servlet and JSP throughput**

** See backup charts for list of Liberty Profile capabilities

* Java7SR4 using -Xaggressive + Flash Express pageable 1Meg large pages

JCICS and Java7SR3 on zEC12

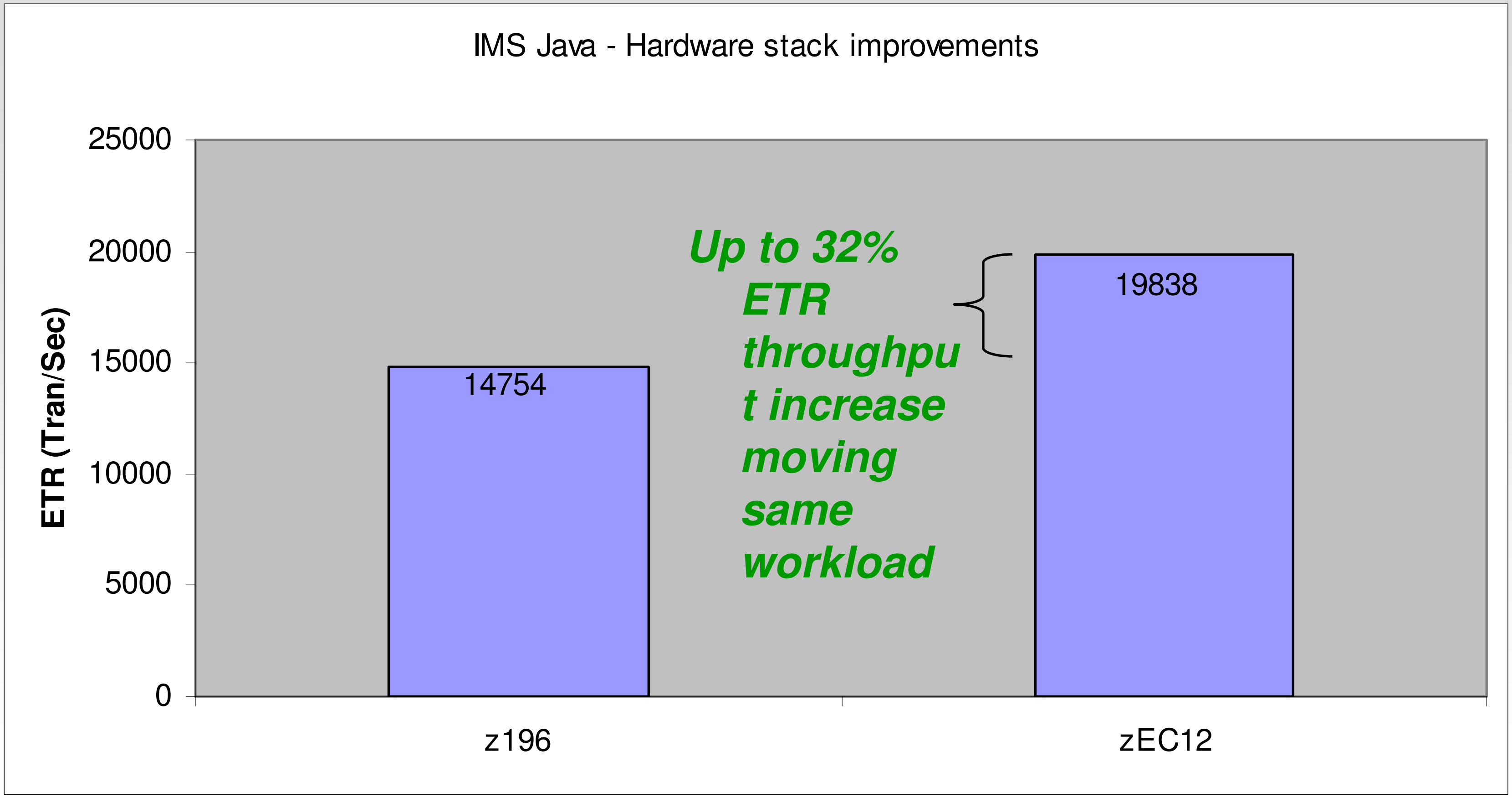
- Using complex Java workload – Axis2 webservice
- Equivalent throughput using CICS V5.1 on z196 compared to CICS V4.2
- 30% improvement in throughput using CICS V5.1 on zEC12 compared to CICS V4.2 on z196
- 39% improvement in throughput using CICS V5.1 with Java 7 zEC12 exploitation compared to CICS V4.2 on z196



(Controlled measurement environment, results may vary)

IMS JMP region performance

Hardware stack improvements

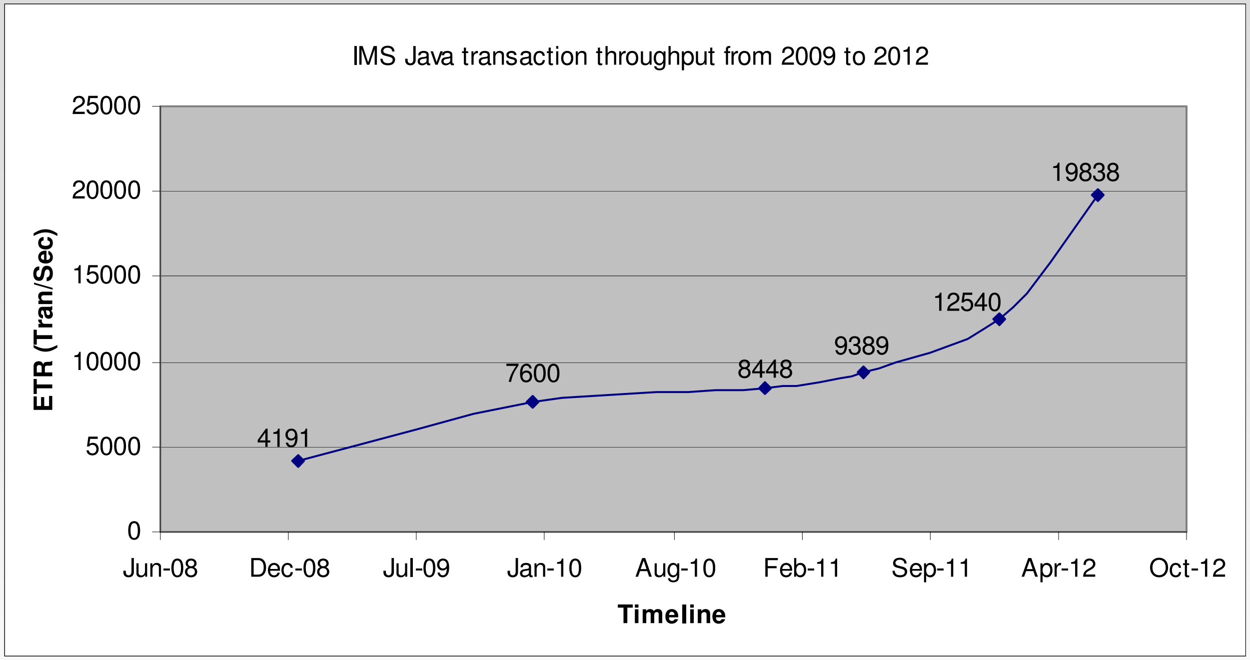


(Controlled measurement environment, results may vary)

(Controlled measurement environment, results may vary)

IMS JMP region performance

Aggregate SDK, software and hardware improvements



(Controlled measurement environment, results may vary)

Over 4x aggregate throughput improvement from 2009 to 2012 due to the following enhancements

- Java version to version performance improvements
- IMS improvements
- Hardware improvements
- DASD improvements